Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

MÁSTER UNIVERSITARIO EN INGENIERÍA DE REDES Y SERVICIOS TELEMÁTICOS

TRABAJO FIN DE MÁSTER

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE IDENTIDAD DIGITAL DESCENTRALIZADA PARA CIUDADANOS DE LA UNIÓN EUROPEA EN EL ÁMBITO SANITARIO

GUILLERMO SANZ GONZÁLEZ

2023

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE IDENTIDAD DIGITAL DESCENTRALIZADA PARA CIUDADANOS DE LA UNIÓN EUROPEA EN EL ÁMBITO SANITARIO

Autor
Guillermo Sanz González

Tutor
Juan Carlos Yelmo García

Departamento de Ingeniería de Sistemas Telemáticos

2023

AGRADECIMIENTOS

El desarrollo de este Trabajo Fin de Máster ha acarreado un gran esfuerzo por mi parte, teniendo que invertir una gran cantidad de horas y de constancia, y donde me he encontrado con decisiones difíciles y momentos donde creía que "no llegaba". Sin embargo, gracias a un conjunto de personas que mencionaré a continuación y de las oportunidades que me han brindado, he podido acabar satisfactoriamente el proyecto, del cual he podido adquirir muchos conocimientos y he podido aplicarlos junto con otros anteriores que no había tenido la oportunidad de aplicar antes.

Lo primero, quiero dar las gracias a mi tutor, el Profesor Juan Carlos Yelmo, por darme la oportunidad de realizar un Trabajo Fin de Máster acorde a mis preferencias y por ofrecerme la oportunidad de colaborar con el grupo de investigación. Además, le quiero dar las gracias por ofrecerme una gran libertad durante el desarrollo del proyecto y de guiarme durante el proceso.

También, quiero dar las gracias al grupo de investigación con el que he podido colaborar, el grupo STRAST (*Sistemas en Tiempo Real y Arquitectura de Sistemas Telemáticos*) del Departamento de Ingeniería de Sistemas Telemáticos (DIT) de la Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT) de la Universidad Politécnica de Madrid (UPM). De ellos he podido aprender multitud de cosas, tanto conocimiento técnico como no técnico; y de ellos he recibido ayuda cuando me enfrentaba a decisiones difíciles.

Respecto a la validación de este proyecto, esta ha sido posible gracias al proyecto concedido a la UPM llamado MadridDataSpace4Pandemics-CM, en el cual se enmarca este trabajo. Este proyecto ha sido financiado por un paquete de ayudas europeo llamado REACT-EU.

Además, me gustaría dar las gracias a la ETSIT por permitirme estudiar el Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación, y el Máster Universitario en Ingenierías de Redes y Servicios Telemáticos, el cual es el máster donde estoy presentando este Trabajo Fin de Máster.

Y, por último, pero no por ello menos importante, quiero agradecer a mi familia y a mis amigos por el apoyo que me han brindado y por la paciencia y compresión que han tenido cuando he tenido periodos de ausencia en los que he estado centrado en la finalización de este trabajo.

RESUMEN

La gestión de la identidad digital depende actualmente de una autoridad centralizada, como puede ser el Estado, un banco o un proveedor de servicios de Internet. Esta autoridad tiene control absoluto sobre esta identidad, tanto desde el punto de vista de la emisión como de la verificación de los certificados de identidad. Desde el punto de vista ciudadano, debemos confiar todo el control sobre nuestros certificados a estas autoridades, las cuales podrían actuar de manera maliciosa al, por ejemplo, eliminar todo rastro sobre la emisión de un certificado.

Como respuesta, están surgiendo iniciativas como la europea EBSI (*European Blockchain Services Infrastructure*) que apoyan un enfoque de identidad autosoberana (SSI). Esta identidad autosoberana promueve una identidad con un mayor control por parte del ciudadano, el cual puede generarla y controlarla a lo largo de todo su ciclo de vida. EBSI pretende alinear la identidad autosoberana con el marco europeo para el reconocimiento de identidades electrónicas (eIDAS) y con el reglamento general de protección de datos (RGPD).

En este Trabajo de Fin de Máster se han aplicado conceptos relacionados con la identidad autosoberana en el ámbito sanitario para ciudadanos de la Unión Europea. Estos conceptos son, principalmente, los identificadores descentralizados (DID), las credenciales verificables (VC), los registros de datos verificable (VDR) y los agentes digitales. Todo estos se respaldan en los registros distribuidos (DLT), en las pruebas de conocimiento cero (ZKP) y en la revelación selectiva, entre otros. Las tecnologías seleccionadas que implementan estos conceptos son Hyperledger Indy, Hyperledger Aries y Hyperledger Ursa, las cuales se van a referir en este documento como el *stack* de identidad Hyperledger.

Las credenciales verificables (VC) permiten no solo verificar de manera automática las afirmaciones sobre una identidad digital (*claims*) que contiene, sino que también permiten revelar únicamente la información necesaria o el cumplimiento de ciertas condiciones sobre esta a una tercera entidad verificadora, evitando así desvelar información adicional.

Para ello, se ha diseñado un sistema para la gestión descentralizada de la identidad digital en el ámbito de la Unión Europea para validar su propuesta de valor, así como los conceptos y tecnologías de identidad autosoberana y, tras esto, se ha implementado un Producto Viable Mínimo (MVP) con el fin de validar la aplicación de la SSI en un escenario sanitario realista. Dado que el impacto causado por las pandemias y otros riesgos sanitarios globales se encuentran entre las principales preocupaciones ciudadanas y políticas, se ha elegido el control de la pandemia por COVID-19 como caso concreto para el escenario del proyecto.

Sin embargo, el propósito de este Trabajo de Fin de Máster no es presentar únicamente una solución para el control de la pandemia COVID-19, sino presentar un sistema de gestión de identidad digital paneuropeo extensible a otros escenarios sanitarios. Otros ejemplos de aplicación son el del control de otras pandemias y endemias, el de crear un DNI neonatal o el de perseguir el enfoque propuesto por la OMS de una salud única (*One Health*).

Palabras clave

Identidad, autosoberana, SSI, DID, VC, Credencial Verificable, Hyperledger, Aries, Indy, COVID, sanidad, Unión Europea.

ABSTRACT

The digital identity management currently depends on a centralised authority, such as the State, a bank or an Internet service provider. This authority has full control over this identity, both in terms of issuing and verifying identity certificates. From the point of view of the citizen, we must entrust the full control over our certificates to these authorities, which may behave maliciously by, for instance, removing all traces of the issuance of a certificate.

In response, there are emerging initiatives such as the European EBSI (*European Blockchain Services Infrastructure*) that support a self-sovereign identity (SSI) approach. This self-sovereign identity promotes an identity with greater control by the citizen, who can generate and control it throughout its lifecycle. EBSI aims to align self-sovereign identity with the European framework for the recognition of electronic identities (eIDAS) and with General Data Protection Regulation (GDPR).

In this master's degree project, concepts related to the self-sovereign identity have been applied to the healthcare field for the Europe Union citizens. These concepts are, mainly, decentralised identifiers (DID), verifiable credentials (VC), verifiable data registries (VDR) and digital agents. These are supported by distributed ledgers (DLT), zero-knowledge proofs (ZKP) and selective disclosure, among others. The chosen technologies that implement these concepts are Hyperledger Indy, Hyperledger Aries and Hyperledger Ursa, which will be referred to in this document as the Hyperledger identity stack.

The verifiable credentials (VC) allow not only to automatically verify claims about a digital identity, but also to selectively disclose only the necessary information or to present only the fulfilment of certain conditions about the information to a third-party verifier, thus avoiding the disclosure of additional information.

To achieve this purpose, it has been designed a system for decentralised digital identity management at the European Union scope to validate its value proposal as well as the self-sovereign identity concepts and technologies and, following this, a Minimum Viable Product (MVP) has been implemented in order to validate the application of SSI in a realistic healthcare scenario. Given that the impact caused by pandemics and other global health risks are among the main public and political concerns, COVID-19 pandemic control has been chosen as a specific case for the project scenario.

Nevertheless, the purpose of this master's degree project is not only to provide a solution for the control of the COVID-19 pandemic, but also to present a pan-European digital identity management system that can be extended to other healthcare scenarios. Other examples of application are the control of other pandemics and endemics, the creation of a neonatal ID card or the pursuit of the One Health approach proposed by WHO.

Keywords

Identity, self-sovereign, SSI, DID, VC, Verifiable Credential, Hyperledger, Aries, Indy, COVID, healthcare, European Union.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE FRAGMENTOS DE CÓDIGO

SIGLAS

ACA-Py	Aries Cloud Agent - Python
AF-.NET	Aries Framework - .NET
AF-Go	Aries Framework - Go
AFJ	Aries Framework JavaScript
AIP	Aries Interop Profile
API	Application Programming Interface
AWS	Amazon Web Services
AVCX	Aries Verifiable Credencial eXchange
BASH	Bourne Again SHell
BFT	Byzantine Fault Tolerance
CA	Certificate Authority
CD	Continuous Delivery
CI	Continuous Integration
CIPA	Código de Identificación Personal Autonómico
CM	Comunidad de Madrid
CSS	Cascading Style Sheets
CURL	Curl URL Request Library
DID	Decentralized IDentifier
DIDComm	Decentralized IDentifier Communication
DIDDoc	Decentralized IDentifier Document
DIDResolver	Decentralized IDentifier Resolver
DIF	Decentralized Identity Foundation
DIT	Departamento de Ingeniería de sistemas Telemáticos
DNI	Documento Nacional de Identidad
DNS	Domain Name System
DOM	Document Object Model
DPKI	Decentralized Public Key Infraestructure
DRY	Don’t Repeat Yourself
EBP	European Blockchain Partnership
EBSI	European Blockchain Services Infrastructure

ECDC	European Centre for Disease Prevention and Control
EEE	Espacio Económico Europeo
ETSIT	Escuela Técnica Superior de Ingenieros de Telecomunicación
EULA	End-User License Agreement
eIDAS	electronic IDentification, Authentication and trust Services
FN	Forma Normal
FNMT	Fábrica Nacional de Moneda y Timbre
gAp	general API controller
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IoT	Internet of Things
IP	Internet Protocol
IPFS	InterPlanetary File System
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation – Linked Data
KMS	Key Management Service
MI	Ministerio de Interior
MS	Ministerio de Sanidad
MSP	Membership Service Provider
MVP	Minimum Viable Product
NAAT	Nucleic Acid Amplification Test
OMS	Organización Mundial de la Salud
PBFT	Plenum Byzantine Fault Tolerance
PCR	Polymerase Chain Reaction
PKI	Public Key Infrastructure
PoC	Proof of Concept
PoS	Proof of Stake
PoW	Proof of Work
QR	Quick Response
RAT	Rapid Antigen Test
REST	REpresentational State Transfer

RFC	Request For Comments
RGPD	Regulación General de Protección de Datos
RUC19	Registro Unificado COVID-19
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SQL-DDL	Structured Query Language - Data Definition Language
SQL-DML	Structured Query Language - Data Manipulation Language
SSI	Self-Sovereign Identity
STRAST	Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos
TAA	Transaction Author Agreement
TIC	Tecnologías de la Información y la Comunicación
TFM	Trabajo Fin de Máster
TLS	Transport Layer Security
ToIP	Trust over IP
UE	Unión Europea
UI	User Interface
UPM	Universidad Politécnica de Madrid
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VC	Verifiable Credential
VDR	Verifiable Data Registry
VON	Verifiable Organizations Network
W3C	World Wide Web
YAML	YAML Ain’t Markup Language
ZKP	Zero Knowledge Proof

Capítulo 1. INTRODUCCIÓN Y OBJETIVOS

1.1 Contexto y motivación

1.1.1 Identidad digital

La identidad digital ha cobrado mucha importancia en los últimos años, ya que se ha incrementado la actividad digital de forma global, tanto a nivel individual como a nivel organizacional. Esto ha implicado un aumento de los riesgos y, con ello, un aumento de la conciencia de estos, especialmente a nivel europeo, donde han entrado en vigor un conjunto de normativas relativamente estrictas a lo largo de la última década[1] relacionadas con la identidad y la identificación electrónica.

Las normativas europeas más destacables en este contexto son el Reglamento (UE) 2016/679 [1], más conocido como **RGPD** (Regulación General de Protección de Datos), y la otra es el Reglamento (UE) 910/2014 [2] más conocido como **eIDAS** (*electronic IDentification, Authentication and trust Services*). La primera se centra en la protección de los datos personales de las personas físicas y a la libre circulación de estos datos, mientras que la segunda es relativa a la identificación electrónica y los servicios de confianza para las transacciones electrónicas. Adicionalmente, actualmente existe una propuesta en marcha que propone modificar el reglamento eIDAS para ampliar la cobertura de la identidad electrónica en la población[2], para permitir la integración con soluciones descentralizadas, para ampliar los datos que componen esta identidad y para permitir un intercambio selectivo de estos, entre otras cosas [3]. Esta propuesta es popularmente conocida como **eIDAS 2.0**.

Para entender mejor el contexto de identidad, es necesario definir qué se entiende como identidad digital. Para ello, se va a usar la definición que da el Banco Santander: *"La identidad digital es el conjunto de información que sumado proyecta una imagen o reputación acerca de nosotros en Internet"* [4]. La razón por la cual se ha usado esta definición es porque refleja un concepto clave: una identidad digital que generamos en Internet se conforma de un conjunto de datos relacionados con un perfil. Nótese aquí que se habla de "una" identidad y no de "la" identidad, ya que una misma persona puede tener simultáneamente varias identidades digitales, cada una de ellas siendo la imagen que tiene un servicio en Internet respecto a los datos que el usuario le genera y aporta. Esta idea es importante porque, tal y como se comentará a continuación, una identidad digital centralizada implica confiar todos los datos que la conforman al servicio correspondiente. En la actualidad, se pueden clasificar las identidades en tres tipos: identidad centralizada, identidad federada e identidad descentralizada.

La **identidad centralizada** se entiende como aquella donde existe un servicio centralizado que la crea, la almacena y la gestiona. Estos servicios pueden ser empresas privadas, como puede ser Google, u organizaciones públicas, como puede ser la Fábrica Nacional de Moneda y Timbre

[1] A fecha de la publicación de este documento, en el mes de febrero de 2023.
[2] Únicamente el 59% de la población de la UE ha obtenido una identidad electrónica (dato del año 2018). Con esta propuesta se pretende aumentar al menos a un 80% para el año 2030 [3].

(FNMT) en España. En estos servicios se suele emplear un identificador como mecanismo de acceso y de una contraseña o de un certificado.

La mayor amenaza a la que se presenta este tipo de identidad es a las filtraciones de información, donde se expone no solo el identificador sino toda la información que tenga el servicio relativa a este. Un identificador puede ser un nombre, un email o un DNI, entre otras cosas, y la información que se puede filtrar junto con ese identificador puede ser, por ejemplo, una dirección de domicilio, intereses o *hobbies*, o incluso información de tipo sensible en el caso de redes sociales, como intereses políticos u orientación sexual. Adicionalmente, terceros pueden hacer uso de la información filtrada para acceder a otros servicios donde se haya usado el mismo identificador y la misma clave.

Otra amenaza considerable es la correlación de información que se produce cuando se cruza información proveniente de dos servicios. Esto puede ocurrir ante filtraciones de seguridad o ante una actuación maliciosa de los servicios al comerciar con estos datos. Este cruce de información es posible cuando un servicio contiene al menos un identificador de un usuario (nombre, DNI, número de seguridad social, etc.) que se corresponde al que contiene el otro servicio. Esto supone un aumento del conocimiento que tiene una empresa o una organización de nosotros, lo cual supone un aumento del impacto ante la materialización de una amenaza.

Todos estos riesgos no suponen un problema únicamente al usuario, sino también a la organización, ya que estas deben aumentar y mejorar las medidas de seguridad para poder cumplir con los requisitos de privacidad expuestos en la RGPD, implicando un aumento en el presupuesto de ciberseguridad.

Un método alternativo de autenticación al usuario y contraseña es el de las firmas y los certificados digitales. Este método funciona gracias a la criptografía, con el uso de claves asimétricas y de funciones resumen, lo cual mejora la seguridad en algunas situaciones. Esta alternativa es mucho más compleja que la anterior, ya que necesita de una Infraestructura de Clave Pública (**PKI**) para poder crear y gestionar las identidades. Esta infraestructura hace uso de un elemento clave, una Autoridad de Certificación (CA), que se encarga de las funciones básicas de la emisión y gestión de los certificados (certificados X.509). En España, como ya se ha dicho, un ejemplo de Autoridad Certificadora es la FNMT, que se encarga de generar los certificados digitales de Personas Físicas.

Aunque esté método puede llegar a ser realmente efectivo en el caso de las identidades digitales, sigue presentando varios problemas, siendo el principal la centralización a través de la CA. Esta mantiene un conjunto de claves públicas asociadas a diferentes identificadores, las cuales pueden ser comprometidas, sea por un error humano o por un ataque informático. Por tanto, las políticas de seguridad en estas autoridades deben ser excepcionales, suponiendo unos costes muy elevados. Como se mencionará en el capítulo 2.2 (Conceptos de identidad autosoberana), existen otros problemas asociados a los certificados digitales basados en PKI.

Una alternativa a la identidad centralizada es la **identidad federada**, que se basa en emplear un servicio intermediario, conocido como proveedor de identidad, entre el usuario y el servicio final, evitando tener que crear una cuenta en este último. El intermediario se encarga de autorizar al

usuario en el servicio final. Ejemplos conocidos de proveedores de identidad son Google, Facebook, Twitter o Linkedin. Y los protocolos más usados son SAML, OAuth y OpenID.

Este mecanismo presenta varias ventajas, como evitar que el usuario tenga diferentes pares de usuarios y contraseñas o evitar la exposición de estos ante filtraciones en los servicios finales. Sin embargo, se aumenta la cantidad de información que almacenan los proveedores de identidad de los usuarios, como la actividad de estos en los diferentes servicios que tenga asociados, identificando así hábitos o intereses del usuario.

Aunque se confíe plenamente en que este servicio va a hacer un buen uso de los datos, el riesgo de filtración sigue presente, con la diferencia de que ahora el impacto es mayor, al aumentar la cantidad de información que almacenan estos intermediarios.

Ante estos riesgos y gracias a las nuevas tecnologías, está cogiendo fuerza la **identidad descentralizada**. Este tipo de identidad no hace uso de contraseñas, ni necesita el uso de intermediarios, como las CA. La idea es simple; un nodo se conecta directamente a otro nodo, creando así una conexión directa. A través de esta conexión los nodos se pueden intercambiar credenciales, ya que se trata de una conexión segura, gracias a la criptografía. En cuanto a la emisión y verificación de certificados, ya no es necesario una CA que los emita o que almacene una lista de revocación, ya que con el surgimiento de la tecnología blockchain se presentó la oportunidad de sustituir las funciones que realiza una CA con un *ledger* distribuido. A esta nueva arquitectura se la conoce como **DPKI** (Infraestructura de Clave Pública Descentralizada). Este nuevo paradigma evita algunos de los problemas presentes en la actual PKI, como los problemas asociados a la centralización y los altos costes de las políticas de seguridad.

Con esta identidad descentralizada también ha surgido la filosofía de una **identidad autosoberana** (**SSI** – *Self-Sovereign Identity*), que defiende una identidad controlada por la persona a la que pertenece, de forma que se puede decidir cuándo y cómo se provee a otras personas o entidades. Esto es, que la persona pueda decidir qué afirmaciones sobre sí mismo compartir, sin necesidad de un intermediario que las presente en nombre de este. Además, con el uso de blockchain y la criptografía, el verificador puede verificar automáticamente y con certeza dichas afirmaciones sin hacer uso, de nuevo, de un intermediario. Este tipo de identidad hace más fácil cumplir con los requisitos de la RGPD, tal y como se verá en el capítulo 2.2.1 (Identidad autosoberana (SSI)).

Una prueba del potencial que tiene esta idea de una identidad autosoberana se encuentra en el apoyo que está recibiendo de la Unión Europea; la cual creó en 2018 una asociación llamada **EBP** (*European Blockchain Partnership*) que incluye a todos los estados miembros de la UE y algunos miembros[3] del EEE (Espacio Económico Europeo), y cuyo objetivo es el de construir una Infraestructura de Servicios Blockchain Europea (**EBSI**) [5], que permitirá hacer uso de la tecnología blockchain en administraciones públicas y en negocios, la cual integra diferentes conceptos como las credenciales verificables o los identificadores descentralizados, permitiendo una identidad autosoberana. Además, EBSI asegura cumplir con las regulaciones europeas RGPD y eIDAS, y seguir los estándares de la W3C [5].

[3] Noruega y Liechtenstein

1.1.2 Ámbito sanitario

El **ámbito sanitario** europeo podría adoptar una identidad autosoberana con el fin de aprovecharse de las ventajas que esta ofrece. Para empezar, se podrían hacer uso de credenciales verificables para acreditar ciertas afirmaciones, como por ejemplo el caso de las credenciales de vacunación y el requerimiento de estas en ciertos escenarios, como cuando se va a viajar a un país endémico. Por otra parte, al trasladar información de los pacientes desde las infraestructuras informáticas sanitarias hacia las carteras digitales de los pacientes, sería más fácil cumplir con las normativas europeas, como la RGPD, ya que ahora serían los pacientes quienes controlan la gestión de estos datos. Esta descentralización también permitiría reducir los problemas relativos a la centralización comentados anteriormente.

Un caso de uso sanitario donde puede encajar SSI a la perfección es el caso de la pandemia COVID-19, donde se han tenido que expedir certificados COVID digitales [6] para poder presentarlos posteriormente en diferentes localizaciones, como restaurantes o aeropuertos. Estos pasaportes han presentado ciertos fallos, como por ejemplo el contener en la propia credencial el número necesario de dosis para obtener una pauta de vacunación completa, dado que ha ido variando a lo largo del tiempo. El uso de las credenciales verificables permite eliminar la lógica de verificación del propio contenido de las credenciales, evitando casos como el de la pauta de vacunación. Por otra parte, aumenta la privacidad al permitir a los ciudadanos presentar únicamente las afirmaciones necesarias para poder probar que cumple con los requisitos sanitarios sin tener que presentar la credencial completa.

Tras los efectos de la pandemia COVID-19, la Comisión Europea ha aprobado un paquete de ayudas europeo llamado **REACT-EU** [7], [8] que tiene como finalidad recuperar los daños económicos y sociales causados. Este paquete de ayudas ha destinado parte de los fondos a la investigación, de donde se ha creado el proyecto **MadridDataSpace4Pandemics-CM** que se le ha concedido a la Universidad Politécnica de Madrid (UPM). Este proyecto se compone de otros más pequeños, donde uno de ellos ha sido concedido al grupo STRAST (*Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos*) del Departamento de Ingeniería de Sistemas Telemáticos (DIT) de la UPM y que se ha denominado como **covID Card**. Este proyecto tiene como objetivo la aplicación de SSI en credenciales COVID mediante la creación de un sistema de identidad digital descentralizada en el ámbito sanitario.

Este Trabajo Fin de Máster se presenta como parte de los resultados de la participación con el grupo STRAST en este proyecto y mostrarán únicamente la actividad realizada por el alumno, excluyendo cualquier tipo de actividad realizada por otros investigadores, desarrolladores o alumnos.

1.2 Objetivos

El objetivo principal de este Trabajo Fin de Máster es el de diseñar e implementar un sistema de identidad digital descentralizada basado en las tecnologías Hyperledger para ciudadanos de la Unión Europea en el ámbito sanitario. Para ello, se presentará un escenario de validación relacionado con la pandemia COVID-19.

Para alcanzar este objetivo, se ha granulado en tareas u objetivos menores, descritos a continuación:

1. Estudio de los conceptos relacionados con la identidad digital y, concretamente, con la identidad autosoberana.
2. Análisis superficial de las diferentes regulaciones y proyectos europeos relacionados con SSI para conocer el contexto del proyecto.
3. Estudio de las tecnologías Hyperledger que implementan los conceptos de SSI (Hyperledger Aries, Hyperledger Indy e Hyperledger Ursa), además de una exploración de las alternativas a estas tecnologías.
4. Diseño e implementación de una Prueba de Concepto (PoC) que permita al alumno probar la madurez de las tecnologías y del potencial de la identidad autosoberana. Esta PoC se basará en un sistema de SSI sencillo con lógica de negocio mínima, aunque relacionada con el escenario de validación (pandemia COVID-19).
5. Análisis del certificado COVID digital actual y de las posibles mejoras que pueden realizarse sobre este.
6. Definición y análisis de un escenario realista de validación relacionado con la pandemia COVID-19 para el objetivo siguiente.
7. Diseño e implementación de un Producto Viable Mínimo (MVP) que permita validar el sistema creado en la Prueba de Concepto. Para ello, se ampliará y se aplicará para un escenario de negocio creado en el objetivo anterior.
8. Elaboración de un conjunto de conclusiones con relación al potencial de SSI, a la madurez de las tecnologías utilizadas y sobre el resultado de aplicar SSI en el ámbito sanitario.

Estos objetivos presentados están alineados con los objetivos del proyecto covID Card, por lo que el desarrollo de este Trabajo Fin de Máster también ha servido para generar valor dentro del proyecto de investigación. Por otra parte, este documento también puede servir para que otros investigadores o desarrolladores verifiquen la viabilidad de SSI y las tecnologías Hyperledger, especialmente en el ámbito sanitario.

1.3 Estructura del documento

A continuación, se describe la estructura del documento, detallando qué contiene cada uno de los capítulos y qué objetivos de los enunciados en el apartado anterior cubren:

- **Capítulo 1.** Introducción y objetivos. Este capítulo presenta una introducción al Trabajo Fin de Máster y enuncia los objetivos a cumplir. Se describe el contexto relacionado con la identidad digital y la identidad autosoberana, y el contexto de la identidad en el ámbito sanitario. Este capítulo cubre el objetivo número 2.
- **Capítulo 2.** Estado del Arte. En este capítulo se describe en profundidad la identidad autosoberana y los conceptos que la rodean, las tecnologías que permiten implementarla y otras tecnologías enfocadas al desarrollo y el despliegue de la Prueba de Concepto y del Producto Viable Mínimo. Este capítulo cubre los objetivos número 1, número 2 y número 3.

- **Capítulo 3**. Prueba de Concepto (PoC). En este capítulo se presenta el diseño y la implementación de la Prueba de Concepto necesaria para probar la madurez de las tecnologías y del potencial de SSI. Además, se presentará el caso de negocio simple empleado. Este capítulo cubre el objetivo número 4.
- **Capítulo 4**. Producto Viable Mínimo (MVP). En este capítulo se presenta la validación del Trabajo Fin de Máster. Esta validación incluye la definición y el análisis del escenario que se va a emplear, el diseño y la implementación del Producto Viable Mínimo. Este capítulo cubre los objetivos número 5, número 6 y número 7.
- **Capítulo 5.** Conclusiones y líneas futuras. En este apartado se presentarán un conjunto de conclusiones relacionados con la viabilidad de SSI y de las tecnologías empleadas, especialmente en el ámbito sanitario, además de las conclusiones del propio Trabajo Fin de Máster. Además, se presentarán una serie de tareas que se podrían realizar en el futuro para continuar el proyecto. Este capítulo cubre el objetivo número 8.

Por último, en los anexos se incluirá más información relevante, como capturas del diseño de las interfaces de pantalla, capturas de las interfaces implementadas, las rutas implementadas en los servidores de la PoC y del MVP, los aspectos éticos, sociales y ambientales que rodean el resultado de este trabajo, el presupuesto económico, etc.

Capítulo 2. ESTADO DEL ARTE

En este capítulo se va a describir en detalle los conceptos que rodean la identidad descentralizada y la filosofía de identidad autosoberana, como pueden ser los identificadores descentralizados (DID), las credenciales verificables (VC) o el Registro de Datos Verificable (VDR). Además, se presentarán las tecnologías Hyperledger que se van a emplear y que permiten implementar un sistema de SSI. Por último, se detallarán las tecnologías de desarrollo y despliegue que se han utilizado para implementar el sistema de identidad descentralizada, tanto para la Prueba de Concepto, como para el Producto Viable Mínimo.

2.1 Técnicas de criptografía básicas

Este apartado tiene como objetivo presentar de forma breve las técnicas de criptografía básicas para comprender ciertos conceptos relacionados con la identidad autosoberana. No se explicarán en detalle porque la criptografía no forma parte de los objetivos del Trabajo Fin de Máster.

Las **funciones resumen** o funciones *hash* se pueden concebir como un sistema que toma como entrada un mensaje *m* de longitud y formato aleatorios y obtiene como salida una ristra de caracteres hexadecimales de longitud fija *h(m)* denominada resumen. El mensaje *m* siempre producirá el mismo resumen *h(m)*, dos mensajes *m* y *m'* nunca producirán un *h(m)* y un *h(m')* coincidentes[4] (por muy similares que *m* y *m'* puedan llegar a ser), y a partir de *h(m)* es computacionalmente imposible[4] obtener *m*.

El **cifrado asimétrico**, también denominados algoritmos de clave pública, permiten generar un par de claves relacionadas entre ellas, pero prácticamente imposible obtener una a partir de la otra. Una de ellas se empleará para cifrar un mensaje y la otra se empleará para descifrarlo. Una de las claves se denomina clave privada ya que el nodo que la ha generado no la debe revelar a nadie, teniendo que almacenarla de forma segura. Y la otra clave se denomina clave pública, puesto que el nodo debe compartirla públicamente con el resto de los nodos con los que vaya a interactuar. Los mensajes transmitidos entre nodos que han compartido sus claves públicas pueden ser encriptados y firmados digitalmente.

La **encriptación** se basa en aplicar una clave criptográfica para cifrar un mensaje, de forma que únicamente se pueda descifrar con la misma clave en el caso de la criptografía simétrica o con la clave asimétrica en el caso de la criptografía asimétrica. Un mensaje encriptado no puede ser descifrado de otra forma, salvo con un ataque criptográfico, es decir, encontrando alguna vulnerabilidad en el algoritmo empleado. En el caso de la criptografía asimétrica, un nodo A cifrará el mensaje con la clave pública del nodo B para que únicamente este último sea capaz de descifrar dicho mensaje.

La **firma digital** es un tipo de autenticación criptográfica, que hace uso de las funciones resumen y del cifrado asimétrico. Cuando un nodo B recibe un mensaje firmado digitalmente por el nodo

[4] En realidad, existe una probabilidad infinitesimal de que se puedan generar dos valores que colisionen entre ellos con un algoritmo como SHA-256. En la práctica, este riesgo es tan bajo que puede ser ignorado.

A, puede estar completamente seguro de que el nodo A es quien dice ser. Como se verá en el apartado 2.2.2 (PKI y DPKI. Registro de Datos Verificables (VDR)) hay ciertas excepciones, como es el caso de que un atacante intercepte las claves públicas en el momento del intercambio, pudiendo así suplantar a ambos nodos. El funcionamiento es el siguiente; el nodo A aplica la función resumen al mensaje m para obtener así $h_1(m)$ y, tras esto, firma este resumen con su propia clave privada, obteniendo así $h_2(h_1(m))$, denominada firma digital. Tras esto, envía al nodo B el mensaje en claro m y la firma digital, de forma que deberá descifrar la firma con la clave pública del nodo A, obteniendo $h_1(m)$, y deberá comprobar si este valor se corresponde al de aplicar la función resumen al mensaje m recibido. Si el resumen coincide, significa entonces que el nodo A es, efectivamente, el que ha enviado el mensaje y de que este no se ha modificado en el camino, lo que implica la integridad del mensaje.

La firma digital y la encriptación se pueden aplicar a la vez, de forma que un nodo A que quiere enviar el mensaje m y la firma digital $h_2(h_1(m))$ a un nodo B, puede cifrar este conjunto con la clave pública del nodo B. De esta forma el nodo B se asegura de tres cosas: de que este es el único capaz de descifrar el mensaje, de que no ha sido manipulado y de que el nodo A es el único que ha podido enviarlo.

Las **pruebas de conocimiento cero** (**ZKP**) son una técnica que permiten a un nodo A probar una afirmación al nodo B, de forma que un nodo C que observa la interacción sin autorización no sea capaz de comprobar qué se está afirmando. Esta afirmación se produce sin revelar la propia información. Un ejemplo de aplicación es cuando un ciudadano quiere probar que ha recibido un título universitario a un tercero sin revelar su identidad. El funcionamiento se basa en que el probador genera un problema matemático $f(x)$ que únicamente se puede solucionar con el secreto x que no quiere desvelar. Además, genera otro problema $f'(x')$ a partir del primero, cuya solución x' estará, por tanto, relacionada con el secreto x. Es importante mencionar que no se puede obtener el secreto x a partir de la segunda solución x'. El verificador le pide, de forma aleatoria, que el probador le demuestre la relación entre los problemas $f(x)$ y $f'(x')$ o que le demuestre la solución x' al segundo problema $f'(x')$. Este último paso se realiza un número determinado de veces, de forma que tras un umbral es posible determinar con seguridad que el probador contiene el secreto que se está probando. Existe una explicación más sencilla basada en el cuento *Ali Babá y los cuarenta ladrones*, donde se detalla cómo se puede saber si una persona conoce el secreto que abre la puerta de la cueva sin revelárselo al verificador. Esta explicación la dieron Quisqueater et al. en el artículo "*How to explain Zero-Knowledge protocols to your children*" [9].

Los **acumuladores criptográficos** se pueden entender como una función que toma como entrada un conjunto de valores V y devuelve un único valor $f(V)$ denominado acumulador. El conjunto se puede subdividir por factores, de forma que $f(V)=f(v,V')$, siendo v un valor del conjunto V y V' el resto de valores que, junto con el valor v, conforman el conjunto V, denominado testigo. Por tanto, se tienen tres elementos, el acumulador (valor), el testigo (subconjunto de valores) y v (valor), donde los dos primeros deben estar públicos y relacionados, y el tercero debe administrarse a un nodo. Adicionalmente, el acumulador puede obtenerse directamente a partir de v y del testigo, pero es computacionalmente inviable obtener v a partir del acumulador y del testigo.

Los acumuladores criptográficos permiten entonces a un nodo demostrar que su valor *v* resulta en un valor *f(V)*. Para esto el verificador debe aplicar la función sobre *v* y el testigo, el cual estará público junto con el acumulador, y comprobar que coincide con el acumulador. Si sobre esta técnica se le añade la firma digital, un verificador puede estar completamente seguro de que el valor no ha sido manipulado. Y si se añade la prueba de conocimiento cero, un nodo puede incluso probar que su valor da como resultado el acumulador correspondiente, sin tener que desvelar el propio valor.

2.2 Conceptos de identidad autosoberana

2.2.1 Identidad autosoberana (SSI)

En el apartado 1.1.1 (Identidad digital) se ha definido la identidad autosoberana como una identidad controlada por la persona a la que pertenece. Para completar esta definición, se va a emplear la que dan A. Preukschat y D. Reed en su libro "*Self-Sovereign Identity. Decentralized digital identity and verifiable credentials*": *"A person's identity that is neither dependent on nor subjected to any other power or state"* [10]. Esto es, a diferencia de lo que pasa con la identidad centralizada, que la identidad deja de estar controlada por una organización para estarlo por la propia persona a la que identifica. Al hablar de control sobre la identidad, es referido tanto a la creación, contención y dominación de esta, pudiendo llegar a ser eliminada o compartida sin el consentimiento del usuario. Esta es una de las razones por las que la RGPD[5] es necesaria, puesto que regula casos abusivos sobre la privacidad de los datos de las personas de la Unión Europea.

Dos organismos importantes que están definiendo una serie de estándares y recomendaciones sobre la identidad autosoberana son el consorcio W3C (*World Wide Web*) [11] y la fundación DIF (*Decentralized Identity Foundation*) [12]. Otras fuentes más específicas a las tecnologías que se van a aplicar son la fundación *Sovrin* [13], la documentación de Hyperledger Aries (*aries rfcs*) [14] y la documentación de Hyperledger Indy (*indy hipe*) [15].

En el resto de este apartado se detallarán los conceptos que conforman el modelo de identidad autosoberana, que se basa en tres ideas fundamentales: los identificadores descentralizados (DID), las credenciales verificables (VC) y el Registro de Datos Verificable (VDR). También se mencionarán otras ideas importantes relacionadas con SSI, como los agentes.

2.2.1.1 Seguridad de los datos con una identidad autosoberana

La identidad autosoberana mejora la **seguridad de los datos** gracias a sus características. Por una parte, la creación de canales seguros entre nodos (por ejemplo, entre una cartera digital de un ciudadano y una organización) permite el intercambio de información de manera segura, permitiendo una autenticación y autorización automática, todo esto gracias a los identificadores descentralizados (DID). Por otra parte, las credenciales verificables (VC) pueden dificultar enormemente la suplantación de la identidad, ya que ahora un tercero malicioso necesita algo más

[5] En el caso de Estados Unidos, la regulación más parecida que existe en el ámbito sanitario es HIPAA (*Health Insurance Portability and Accountability Act of 1996*) [136], cuyo objetivo es proteger información médica sensible de sus ciudadanos.

que la propia información, la cual ha podido ser filtrada por otros medios, ahora necesita cada una de las claves privadas asociadas a cada uno de los datos para poder generar afirmaciones (*claims*) verificables. Y, por último, estas claves privadas están almacenadas localmente[6] en cada una de las carteras digitales de los ciudadanos, evitando así filtraciones masivas de datos cuando los atacantes acceden a las bases de datos de las organizaciones.

Además de mejorar la seguridad, mejora la **privacidad**, ya que las credenciales verificables, tal y como se verá en el apartado 2.2.4 (Credenciales Verificables (VC)), permiten afirmar únicamente lo que la organización o compañía verificadora necesita saber del ciudadano, sin tener que revelar otros datos de forma innecesaria. Por otra parte, los usuarios tendrán que consentir explícitamente la compartición de información con otras organizaciones. Aunque con la RGPD esto ya sucede, el matiz es que con SSI el ciudadano sabe exactamente a qué organizaciones o empresas su información está siendo compartida, porque serán las propias organizaciones las que soliciten las afirmaciones al usuario, pudiendo negarse a aceptar en cualquier caso.

En cuanto a la **protección de los datos,** SSI facilita el cumplimiento de las organizaciones con la RGPD por, principalmente, tres motivos. El primer motivo es que las credenciales verificables permiten de una manera sencilla cumplir con el principio de minimización de datos. El segundo motivo es relativo al derecho de supresión ya que, al hacer uso de canales seguros y privados, con autenticación y autorización automática, el individuo puede solicitar acceso a sus datos sin peligro de que sean interceptados sin cifrar y puede mandar una petición para borrar sus datos firmada con sus claves privadas. El tercer motivo viene dado por una de las técnicas que deben aplicar los responsables del tratamiento de los datos: la seudonimización. Esta técnica se basa en tratar la información para que no pueda identificar al ciudadano por sí sola, esto es, eliminando o modificando identificadores como el nombre o el DNI, o eliminando o modificando seudoidentificadores que, en conjunto, puedan identificarlo, como puede ser el conjunto de datos de fecha de nacimiento, código postal y género. SSI facilita el proceso al crear conexiones seguras a través de los identificadores descentralizados privados que, aunque se describirán en el apartado 2.2.3.4 (DID públicos y privados), se puede adelantar que estos son identificadores no correlables, ya que son únicos. De esta forma la organización solo necesita de este identificador no correlable para identificar al individuo, en oposición a la identidad centralizada donde la organización necesita de un identificador correlable, como el email o el DNI.

Por último, SSI facilita en gran medida el **derecho a la portabilidad de los datos** gracias a las conexiones seguras, de forma que las organizaciones tienen a su disposición una herramienta que asegura la autenticación y autorización del individuo.

2.2.2 PKI y DPKI. Registro de Datos Verificables (VDR)

Una entidad puede crear un par de claves criptográficas para que terceros se aseguren de que están interactuando con dicha entidad. Para esto, debe almacenar su clave privada y compartir la clave pública. Aunque la idea es sencilla, hay un problema que no tiene fácil solución: compartir la clave pública sin que sea interceptada. Cuando una clave es interceptada por un atacante, este

[6] Aunque las claves privadas estén almacenadas en los dispositivos de los ciudadanos, es recomendable que estos tengan copias de seguridad de estas, trasladando la responsabilidad sobre los datos al ciudadano.

puede generar otro par de claves y compartir su clave pública con el destinatario original, de forma que todos los mensajes que envía la entidad original se firman digitalmente con la clave del atacante, sin que el destinatario sea consciente de ello.

Es importante, por tanto, asociar una clave pública con un identificador que describa a la entidad que la ha originado. El problema viene en cómo conseguir una asociación fiable entre el identificador y la clave. Para esto, han surgido varias soluciones a lo largo del tiempo, pero solo se van a presentar las dos que se consideran más relevantes: PKI y DPKI.

La solución más extendida es la que ofrece una Infraestructura de Clave Pública (**PKI**), que introduce una organización denominada Autoridad Certificadora (CA), en la cual se debe confiar. Esta CA se encarga de emitir un documento digital que certifica que un identificador y una clave pública están unidas. En el caso de la FNMT, asocia la clave pública con la identidad del ciudadano (DNI y nombre).

Aunque PKI es viable, tal y como ha demostrado la extensión de su uso, presenta algunos problemas que con el aumento de la importancia de la identidad digital aumenta la urgencia con la que deben suplirse:

- Desde el punto de vista de privacidad, tener un punto centralizado produce un riesgo de que todos los certificados digitales puedan ser comprometidos ante ataques informáticos.
- Desde el punto de vista de seguridad, otro riesgo viene asociado al factor humano de aquellos trabajadores que operan para mantener en funcionamiento una CA, que pueden equivocarse o que se les puede condicionar para actuar de manera maliciosa (ante una extorsión, por ejemplo).
- Desde el punto de vista de disponibilidad, la centralización supone un único punto de fallo, por lo que se expone a ataques informáticos de denegación de servicio o al fallo de equipos informáticos.
- Y desde el punto de vista económico, los riesgos mencionados anteriormente suponen costes, ya que implican tener unas políticas de seguridad lo suficientemente maduras como para poder reducirlos lo máximo posible.

Con la identidad descentralizada nace una solución de Infraestructura de Clave Pública Descentralizada (**DPKI**), que elimina a la CA de la arquitectura. En este caso, es la propia entidad quien genera un elemento que asocia la clave pública con un identificador. El identificador es un DID y el elemento generado se denomina DIDDoc, ambos se verán en detalle en el apartado 2.2.3 (Identificadores descentralizados (DID)). Este elemento puede ser publicado en un registro, denominado Registro de Datos Verificable (**VDR**), con el fin de que otras entidades puedan acceder de forma transparente para verificar la asociación identificador-clave pública.

Puede parecer que esta solución no mejora respecto a la anterior, ya que se pasa de depositar la confianza en una CA a depositarse en un VDR. Sin embargo, dado el surgimiento de la tecnología blockchain, esta confianza puede ser sustituida por la certeza criptográfica de que una transacción publicada en la cadena no ha podido ser manipulada. Adicionalmente, se evitan los problemas

relacionados con la centralización que se tienen en PKI, ya que una blockchain es descentralizada. Y, por otra parte, se evitan los costes asociados a mantener una CA[7].

En la práctica, un VDR es un componente flexible, es decir, puede ser una de las redes blockchain públicas conocidas, puede ser una base de datos convencional o puede ser un repositorio en GitHub, entre otras muchas opciones. No obstante, estos elementos no están recomendados puesto que implican centralizar parte de la infraestructura, dejando de tener así una identidad descentralizada.

2.2.3 Identificadores descentralizados (DID)

2.2.3.1 Introducción

Los identificadores descentralizados permiten crear una identidad digital descentralizada y verificable sobre las entidades que los controlan. Esto es, una entidad puede crear un DID y, a partir de este, terceros pueden verificar sin necesidad de intermediarios que esta identidad pertenece realmente a dicha entidad, con una garantía absoluta.

Un DID es una URI (*Uniform Resource Identifier*) que puede ser resuelto a través de un elemento denominado DIDResolver (2.2.3.3 DIDResolver) para recuperar el recurso que identifica. Este recurso se denomina DIDDoc y contiene un conjunto de elementos necesarios para establecer comunicaciones seguras, como claves criptográficas. Este elemento se verá con mayor profundidad en el apartado 2.2.3.2 (DIDDoc).

Esta URI es una concatenación de caracteres que se divide en tres partes, separadas por dos puntos. La primera parte es el identificador URI, cuyo valor es siempre "*did*". La segunda parte es el método del DID, que se detallará en el apartado 2.2.3.5 (Métodos). Y la tercera parte es el identificador específico al método, siendo este una concatenación de caracteres única dentro del método especificado. Un ejemplo de DID es: "*did:sov:WRfXPg8dantKVubE3HX8pw*"

Estos identificadores presentan una serie de propiedades:

- Son un tipo de URI. Pueden ser resueltos para obtener un recurso.
- Son descentralizados. Esto implica que cualquiera puede crear un DID en cualquier momento y sin necesidad de una Autoridad de Certificación u otro tipo de servicios centrales.
- Son globalmente únicos[6].
- Son verificables criptográficamente. Una entidad puede probar que controla un DID de forma criptográfica, a través de su clave privada.

A través de los DID se pueden crear conexiones seguras entre dos nodos. Para esto, cada uno de los dos nodos creará un DID diferente y compartirá con el otro las claves públicas. A partir de ese momento, todos los mensajes (o credenciales) que se intercambien se encriptarán con la clave pública del otro nodo y se firmarán digitalmente, a través de su clave privada. Esto implica, por una parte, que el receptor de un mensaje podrá leer los mensajes del emisor únicamente si tiene

[7] El uso de una blockchain implica otros gastos: la escritura de transacciones en la red [52]. Sin embargo, estos gastos son inapreciables en comparación con los asociados a mantener una CA.

bajo control el DID asociado a la conexión, es decir, si tiene la clave privada. De forma similar, un nodo puede probar que está bajo el control de un DID simplemente enviando un mensaje que el receptor sea capaz de desencriptar con la clave pública de ese DID. Por tanto, una conexión entre dos nodos es totalmente segura siempre y cuando las claves privadas de los DID se mantengan bajo control, y esta responsabilidad recae únicamente en el propio individuo.

Por último, por cada conexión que se establece se crea un DID diferente, por lo que el servicio no podrá correlar información del individuo a través de este identificador.

Es importante destacar que este elemento es actualmente una recomendación de la W3C [16], [17].

2.2.3.2 DIDDoc

Un DIDDoc es un recurso que se devuelve cuando se resuelve un DID a través de un DIDResolver. Mientras que un DID identifica a una entidad, un DIDDoc se dice que lo describe [16].

Aunque más adelante se van a detallar los roles que participan en el modelo de Credenciales Verificables, es importante mencionar ahora la diferencia entre la entidad a la que un DID identifica, y la entidad que controla la propia identidad. En el primer caso se está hablando de un sujeto (*subject*), que puede ser una organización, una persona, un animal o un objeto, y en el segundo caso se habla del titular (*holder*), que será una organización o una persona.

En la siguiente figura de elaboración propia se muestra la relación entre los elementos y los roles de sujeto y titular, en el caso de que estos fuesen entidades diferentes:

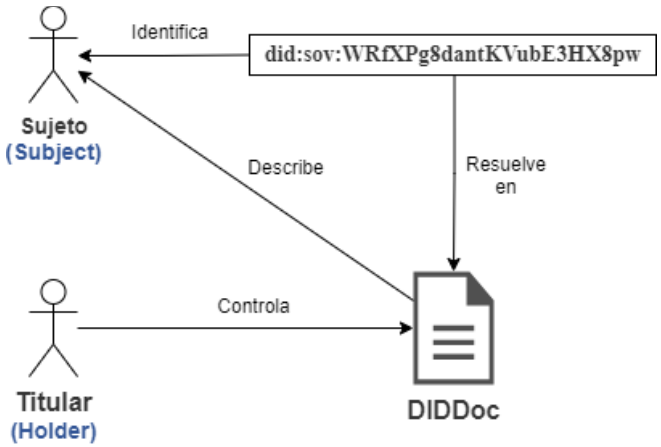

Figura 1: Relación entre un DID y un DIDDoc

Un DIDDoc es, realmente, un documento que contiene un conjunto de datos necesarios para verificar criptográficamente la identidad y otros elementos necesarios para crear conexiones seguras. Este primer punto implica que un DIDDoc asocia un identificador con su clave pública, uno de los problemas que resolvía PKI a través de certificados digitales expedidos por una CA y que DPKI soluciona con estos elementos.

Aunque estos documentos pueden contener datos personales de los sujetos, la W3C recomienda encarecidamente que no se realice esta práctica con entidades no públicas (por ejemplo,

ciudadanos) [16]. La recomendación para compartir datos personales son las Credenciales Verificables (VC) que se verán más adelante.

Un DIDDoc puede contener, por tanto, una multitud de datos. El único elemento requerido en un documento es el propio identificador (DID), aunque esto no tendría ningún sentido (un DID resolvería en un elemento que únicamente contiene ese identificador). Los elementos que suele contener un documento son los siguientes:

- El identificador descentralizado que lo referencia.
- Elementos necesarios para la verificación criptográfica. Se pueden elegir entre varios métodos de verificación, algunos complejos, pero el más simple y en el que se centrará este Trabajo Fin de Máster es el uso de un único par de clave pública y clave privada. En este caso, se debe incluir únicamente la clave pública, mientras que la clave privada se almacenaría en la cartera digital del titular.
- Un punto de acceso por el cual se puede iniciar la interacción entre entidades a través del protocolo correspondiente (por ejemplo, HTTPS).

Estos documentos se serializan en un objeto JSON.

Un ejemplo de DIDDoc es el siguiente, obtenido de la especificación de la W3C [16]:

```
{
      "@context": "https://www.w3.org/ns/did/v1",
      "id": "did:example:123456789abcdefghi",
      "authentication": [{
              "id": "did:example:123456789abcdefghi#keys-1",
              "type": "Ed25519VerificationKey2018",
              "controller": "did:example:123456789abcdefghi",
              "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
      }],
      "service": [{
              "id":"did:example:123456789abcdefghi#vcs",
              "type": "VerifiableCredentialService",
              "serviceEndpoint": "https://example.com/vc/"
      }]
}
```

Fragmento de código 1: Ejemplo DIDDoc [16]

En este ejemplo, el primer elemento se puede ignorar, ya que es relativo a la especificación de JSON-LD, el segundo elemento es el DID, el tercer elemento contiene los elementos necesarios para la verificación criptográfica, donde se puede encontrar una clave pública, y el cuarto elemento contiene información relativa al punto de acceso, donde se puede encontrar una URL HTTPS.

Estos documentos se almacenan en la mayoría de los casos[8] en un Registro de Datos Verificable (VDR). El método del DID es el que dictará si se almacena en un VDR y, ese caso, en cual.

[8] Algunos documentos no se almacenan en un VDR, como, por ejemplo, el documento asociado a un DID con método "*peer*" o con método "*key*".

2.2.3.3 DIDResolver

Como ya se ha explicado, un DIDResolver es un elemento que permite obtener un documento (DIDDoc) a través de un identificador (DID). La idea es la misma que cuando se obtiene un recurso web a través de una URL (con esquema HTTP o HTTPS) en un navegador o cuando se obtiene la dirección IP a través del nombre del dominio con DNS; existe un componente software (aunque también podría ser hardware) que recibe como entrada un DID y produce como salida el DIDDoc asociado [16]. Esto es posible ya que el documento contiene el propio identificador como atributo obligatorio, tal y como se ha mencionado antes.

Aunque existe cierta similitud con DNS y HTTP, existe una diferencia notable; el proceso de resolución depende completamente del método especificado en el DID. La W3C contiene un registro [18] donde se especifican multitud de métodos DID[9], detallando cosas como el VDR asociado (si lo tiene) o los procedimientos asociados al identificador, como el de resolución. Por tanto, en la resolución un método puede requerir la interacción con una blockchain pública, la interacción con DNS o la interacción con una base de datos convencional.

De igual forma, la arquitectura del propio componente varía entre métodos. Existen implementaciones de identidad autosoberana que incluyen una biblioteca software que se encarga de resolver los DID. Otras veces requieren de un servicio externo (centralizado o descentralizado) al que se realiza la petición para la resolución. También existen soluciones virtualizadas a través de contenedores que permiten la integración del componente en arquitecturas más complejas.

Por ejemplo, el *stack* de identidad Hyperledger, del que se hablará más adelante, incluye un DIDResolver híbrido que, por una parte, tiene una biblioteca que soporta los métodos más utilizados en el contexto de Hyperledger Indy y, por otra parte, delegar en un componente externo para resolver el resto de los métodos, aumentando la eficiencia para métodos comunes y evitando crear un componente demasiado grande que incluya todos los métodos soportados por la W3C.

Otro ejemplo de DIDResolver es el *DIF Universal Resolver* [19]. Este componente, desarrollado por DIF, es un componente que permite la resolución de multitud de métodos. Existen dos formas de integración, la primera es hacer uso de su instancia desplegada de forma centralizada [20], y la segunda es desplegar el componente virtualizado a través de contenedores Docker [19]. La primera opción no es recomendable, puesto que se perdería parte de la descentralización que se ofrece con SSI.

Se puede observar por tanto que dependiendo del método y de cómo implemente la resolución de los identificadores, se puede tener mayor o menor grado de descentralización. Esta es una de las razones por las que se recomienda hacer uso de una blockchain como VDR, en lugar de una solución centralizada como una base de datos.

2.2.3.4 DID públicos y privados

Los DID se pueden clasificar en dos tipos dependiendo de su método: privados y públicos. La diferencia entre ambos se encuentra en el propósito.

[9] La W3C ha reportado 103 métodos experimentales registrados a fecha de agosto de 2021 [16]

Un **DID privado** es un identificador cuyo propósito es el de crear conexiones seguras entre dos entidades. Cada uno de los participantes en una conexión genera un DID totalmente nuevo y le envía el DIDDoc asociado al otro participante para que este último lo almacene de forma segura en su cartera digital. De esta forma, ambos puntos de una conexión tienen las claves públicas del contrario, permitiendo la encriptación y la firma digital. Al tener un DID por cada conexión, se evita la correlación de información a partir de dicho identificador.

Un **DID público** es un identificador cuyo propósito es estar accesible públicamente a través de un VDR para que otras entidades puedan verificarlo y crear conexiones con la entidad a la que identifica. Al contrario de lo que pasa con los privados, un DID público permite la correlación de información, ya que el mismo identificador utiliza en diferentes escenarios. Por esta razón, este DID solo está pensado para entidades públicas, como organizaciones. El motivo por el cual este identificador es necesario es porque una organización que expide una credencial a una entidad debe disponer de un mecanismo que permita verificar por terceros que realmente esa credencial ha sido realmente expedida por esta organización, y no ha sido manipulada desde entonces. Para esto, la organización hará uso de sus claves privadas para firmar digitalmente la credencial, y el tercero hará uso de las claves públicas que encuentra en el DIDDoc público para verificar la firma.

2.2.3.5 Métodos

Hasta ahora, se ha explicado qué es un método de un DID, cómo dicta la existencia o no de un VDR y, si procede, de cuál, cómo se resuelve y qué tipo de identificador es, público o privado. Lo único que falta en relación con los métodos, es detallar algunos de los más extendidos y relevantes.

En cuanto a métodos de DID privados, se encuentran *did:key* [21] y *did:peer* [22]. Estos métodos no están pensados para publicarse en un VDR. La diferencia principal entre ambos es que el primero no contiene ningún punto de acceso al que conectarse, mientras que el segundo sí. El método *peer* por tanto se utilizará para la creación de conexiones seguras entre dos entidades. Ambos métodos siguen asociando una identidad con un método de verificación.

En cuanto a métodos de DID relacionados con las blockchain Hyperledger Indy, se encuentran *did:sov* [23] y *did:indy* [24]. Ambos resuelven DID cuyos DIDDoc están almacenados en blockchain basadas en la tecnología Hyperledger Indy. El primer método se usaba originalmente para resolver únicamente DID localizados en la red Sovrin, pero otras blockchain que han aparecido posteriormente han adoptado el mismo método. Esto implica una situación compleja en la que un DIDResolver debe adivinar en qué blockchain un DIDDoc está almacenado a partir del identificador. A raíz de este problema, surgió el segundo método, que aún está adoptándose en las diferentes soluciones de SSI[10]. Este método (*indy*) permite definir en el propio DID la blockchain e, incluso, la rama de la blockchain en la que se encuentra un DIDDoc. Por ejemplo, el siguiente DID indica que su DIDDoc asociado se encuentra en la rama *Staging* de la blockchain *Sovrin*: "*did:indy:**sovrin**:**staging**:6cgbu8ZPoWTnR5Rv5JcSMB*".

[10] Cuando se detalle la tecnología Hyperledger Aries se mencionará que el método *indy* se introduce en una segunda versión que está actualmente en proceso de adopción. De hecho, este TFM se ha desarrollado con DID que usan el método *sov*.

También existen métodos de DID públicos que no implican a una blockchain como VDR, como por ejemplo *did:web* [25] o *did:github* [26]. En el primer caso, se hace uso de DNS para resolver los DID y estos se registran en el servidor web del servicio correspondiente, es decir, se hace uso de DNS como DIDResolver y de un servidor web como VDR. Y en el segundo caso, se hace uso del servicio GitHub como registro centralizado y de una biblioteca para resolver los identificadores. Aunque ambos métodos presentan grandes deficiencias, el método *web* puede ser útil cuando se confía en la organización y el método *github* cuando se está en una de las primeras fases de desarrollo de un producto basado en SSI y se quiere explorar la viabilidad de este.

Por otra parte, mencionar el método *did:ebsi* [27], que es el método que soporta el proyecto EBSI para resolver los DIDDoc almacenados en su propia blockchain que usa como VDR.

Solo se han mencionado unos pocos, los que se han considerado más relevantes en el contexto de este Trabajo Fin de Máster. Sin embargo, existen muchos más, como *did:btcr* (Bitcoin), *did:eth* (Ethereum), *did:cr* (Hyperledger Fabric) o *did:gatc* (Gataca[11]), por poner unos ejemplos. Cada uno de ellos con sus propios métodos de resolución.

2.2.3.6 DIDComm

Hasta ahora se han descrito los DID y una serie de elementos relacionados con estos, como los documentos (DIDDoc). También se han mencionado los DID privados y algunos métodos, como el método *did:peer*. Y, por último, se ha indicado que estos identificadores permiten crear conexiones seguras entre nodos, gracias a los métodos de verificación que se indican en los DIDDoc. Todos estos conceptos son los que permiten construir el protocolo necesario para crear estas conexiones seguras, denominado **DIDComm**, nombre que alude a la comunicación con DID.

Este protocolo es un protocolo orientado a mensajes asíncronos que es agnóstico al protocolo de transporte utilizado y que aplica encriptación punto a punto. Los mensajes se envían con un mecanismo de mejor esfuerzo (*best-effort*) [28].

Cuando se dice que es un protocolo orientado a mensajes asíncrono, quiere decirse que este protocolo está orientado a enviar mensajes entre nodos y que no se espera una respuesta inmediata, si es que la hay. Los mensajes que se envían entre dos nodos son independientes de este protocolo. Es decir, se pueden enviar tanto mensajes en texto plano como credenciales verificables que el protocolo no va a modificar su procedimiento.

Respecto a que DIDComm sea agnóstico al protocolo de transporte, esto quiere decir que los mensajes pueden ser distribuidos con cualquier protocolo, como HTTP(S), SMTP, etc. Incluso, el envío de un mensaje puede atravesar varios nodos y que el protocolo de transporte varie entre caminos.

La encriptación punto a punto permite esta independencia del mensaje enviado y del protocolo de transporte utilizado, ya que, independientemente de la importancia del mensaje o de la seguridad del transporte, DIDComm asegurará ambos haciendo uso de las claves públicas y

[11] Gataca es una empresa emergente española que implementa una plataforma SSI que es conforme con el proyecto EBSI [89].

privadas establecidas en los DIDDoc empleados en la conexión. Cualquiera que intente interceptar un mensaje en la ruta, se encontrará con el contenido de dicho mensaje encriptado.

Y, por último, respecto a la definición dada, los mensajes son entregados con un mecanismo de *best-effort*. Esto quiere decir que cuando se envía un mensaje desde un origen a un destino, la red no garantiza una calidad de servicio mínima, ni siquiera garantiza que el mensaje vaya a llegar. La red asigna los recursos disponibles en ese momento al usuario.

Este protocolo también permite encaminamiento entre nodos. Para preservar la privacidad de la información que contienen los mensajes enviados, este encaminamiento limita la información de la ruta que va a seguir un mensaje hasta llegar a su destino, de forma que los nodos que reciben un mensaje que deben reenviar a otro nodo únicamente conozcan de quién ha recibido el mensaje y a quién debe enviárselo, sin saber en ningún momento quienes son el remitente y el destinatario.

Para entender el funcionamiento del encaminamiento, supongamos que la conexión entre un remitente y un destinatario pasa por un nodo intermediario. El remitente deberá encriptar y firmar el mensaje de acuerdo con los DIDDoc intercambiados entre remitente y destinatario, así, este último será el único de desencriptar el mensaje. Además, el remitente deberá encriptar y firmar el mensaje una vez más antes de enviarlo al intermediario, esta vez de acuerdo con los DIDDoc intercambiados entre el remitente y el intermediario. Cuando el mensaje es recibido por el nodo intermediario, deberá desencriptarlo para encontrar un mensaje encriptado (que no puede desencriptar) y un campo que indica a quién debe enviárselo, en este caso, al destinatario. Por último, el receptor recibe el mensaje y lo desencripta para obtener el mensaje original. La información de la ruta a seguir se incluye en los DIDDoc intercambiados entre remitente y destinatario.

El ejemplo que se ha dado es un ejemplo común de aplicación, cuando el destinatario es un teléfono móvil que puede estar o no conectado a una red en ese instante. En estos casos, se añade un nodo denominado mediador, que en el ejemplo anterior sería el nodo intermediario. Este mediador conserva el mensaje hasta que el teléfono móvil se encuentre accesible.

Este proceso se puede ampliar a varios nodos que encaminan los mensajes. Para ello, el mensaje deberá encriptarse las veces que sean necesarias, incluyendo siempre un único campo que haga referencia exclusivamente al siguiente nodo.

2.2.4 Credenciales Verificables (VC)

2.2.4.1 Introducción

Como ya se mencionó, los DID (y DIDDoc) no están pensados para contener información personal de los sujetos. Para esto, se hacen uso de las credenciales verificables (VC), las cuales hacen uso de las conexiones seguras formadas sobre los DID y DIDDoc a través de DIDComm para emitirse y verificarse.

La separación que ofrecen los DID y las credenciales verificables sobre la identidad de una entidad, permite aumentar la privacidad y mejorar la seguridad respecto a los certificados digitales actuales basados en PKI, tal y como se ha visto hasta ahora. El identificador utilizado para asociarse con la clave pública de la entidad deja de ser un identificador correlable, como era el

caso del DNI. Con SSI, gracias a las credenciales verificables, un nodo puede compartir una serie de afirmaciones (*claims*) que forman parte o derivan de una credencial sin tener que compartir la credencial entera. Esta separación permite, por tanto, que una serie de atributos que conforman una identidad no se relacionen siempre con un mismo identificador.

La W3C define un ecosistema de credenciales verificables (VC) en el que existen varios roles, elementos e interacciones, los cuales conforman la infraestructura necesaria para el intercambio de VC [29]. Este ecosistema se refleja en la siguiente figura de elaboración propia, aunque basada en la proporcionada por la W3C [29]:

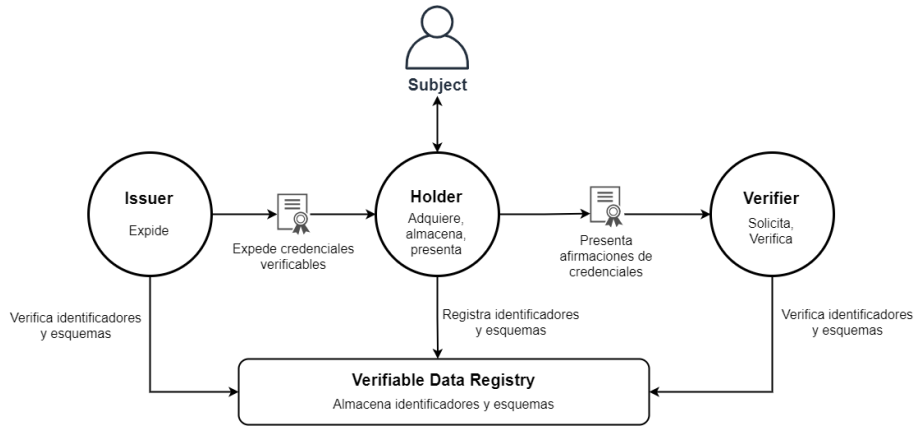

Figura 2: Ecosistema de credenciales verificables

Las conexiones entre entidades son conexiones seguras formadas a través de identificadores descentralizados (DID) y del protocolo DIDComm.

Los roles que se muestran en la figura anterior son los siguientes:

- Titular (*holder*). Entidad que posee credenciales verificables y que genera presentaciones con afirmaciones verificables sobre ellas. A veces se refiere a esta entidad como "controlador" o como "probador", dependiendo del contexto.
    - Sujeto (*subject*). Entidad a la que identifica el identificador y sobre la que las presentaciones son generadas. En la mayoría de los casos, el sujeto y el titular son la misma entidad, salvo en casos especiales, como el caso de un tutor (titular) que tiene el control sobre un menor de edad (sujeto). Otro ejemplo es el caso de una mascota (sujeto) y su dueño (titular).
- Emisor o expedidor (*issuer*). Entidad que emite credenciales verificables.
- Verificador (*verifier*). Entidad que recibe credenciales verificables.

A lo largo del documento se van a referir a los roles de manera indistinta con su nombre en inglés o con su traducción al español. Además, salvo que se especifique lo contrario se va a suponer que un titular y un sujeto son la misma entidad, por lo que cuando se refiera a un titular, se supondrá que este es también el sujeto.

El Registro de Datos Verificable (VDR) ya se ha definido en la sección 2.2.2 (PKI y DPKI. Registro de Datos Verificables (VDR)), y como indica la figura, se encarga de almacenar DID (públicos) y esquemas de credenciales, entre otras cosas.

Un esquema de credencial es un documento que indica los atributos que debe contener una credencial, de forma que todas las credenciales expedidas con un mismo esquema coincidan en formato. De esta forma, un verificador es capaz de averiguar qué afirmaciones de una credencial necesita que el titular pruebe.

Además del esquema, en el VDR se publica otro elemento fundamental: la definición de credencial. Este elemento permite relacionar un esquema con el DID del emisor que va a expedir credenciales sobre dicho esquema y permite asociar a cada uno de los atributos del esquema una clave pública diferente. Esto último implica que cuando un *issuer* expide una credencial a un *holder*, firmará con sus claves privadas cada una de las afirmaciones. Esta técnica puede parecer excesiva, pero es lo que permite al titular revelar de forma selectiva los atributos que se quieren presentar al verificador y es lo que permite realizar predicados o preguntas sobre las afirmaciones, técnica que se basa en la técnica criptográfica de prueba de conocimiento cero (ZKP). Por tanto, un verificador deberá verificar criptográficamente cada una de las afirmaciones por separado.

En contraposición a PKI, el *holder* almacena en su cartera sus credenciales y es el que decide cuándo y de qué manera las comparte con otros, mejorando así la privacidad.

Cuando el titular quiere probar algo al verificador, creará una o varias presentaciones sobre una o varias credenciales verificables, de forma que cada una de las presentaciones contiene una serie de afirmaciones reveladas y/o una serie de predicados. Esto implica que un titular no entrega la credencial que se le ha expedido, sino un elemento generado de esta. En la siguiente figura se muestra un ejemplo de cómo se forma una única presentación a partir de dos credenciales basadas en credenciales JSON-LD (formato que se describirá más adelante), donde se recogen una serie de afirmaciones a revelar y una serie de afirmaciones que probar:

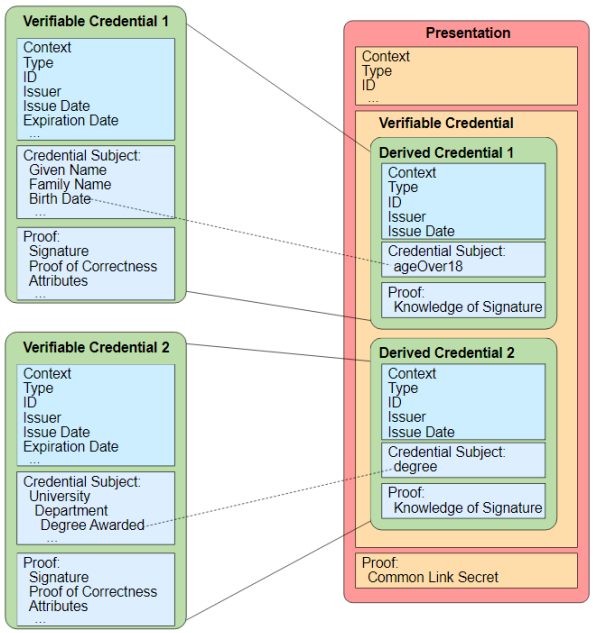

Figura 3: Presentación de credencial, por la W3C [29]

Una presentación de credencial permite al verificador asegurarse de varias cosas:

1. La credencial fue expedida por el emisor indicado. Esto se verifica gracias a las firmas digitales que ha producido el emisor sobre cada uno de los atributos de la credencial.

2. Las afirmaciones no han sido alteradas. Al igual que antes, esto se verifica gracias a las firmas digitales que produce el emisor.
3. El titular es realmente a quien la credencial fue expedida. Esto se consigue gracias a las pruebas de conocimiento cero (ZKP), y se verá más adelante.
4. La credencial no ha sido revocada. Existen diferentes mecanismos que se verán más adelante.

Todo esto permite al verificador evitar tener que confiar en el titular, puesto que esta confianza se suple con la veracidad criptográfica. Sin embargo, el verificador aún deberá decidir si confía o no en el emisor de dicha credencial.

El proceso de verificación se produce sin la interacción del *issuer*, ya que solo se necesitan las claves públicas que se encuentran accesibles en el VDR para verificar una presentación. No obstante, existe una excepción para la cual un emisor puede tener que intervenir, y esta es cuando un verificador quiere asegurarse de que está autorizado para expedir las credenciales correspondientes, permitiendo así confiar (o no) en este. Un ejemplo concreto sería cuando un verificador recibe una presentación de credencial de un test PCR COVID-19 de un titular. Al revisar la presentación, se encuentra que la credencial la ha expedido un centro que no conoce. Para asegurarse de que ha sido expedida por un centro autorizado por el Ministerio de Sanidad (en el caso de España), crea una conexión segura basada en DIDComm con el centro y le solicita una presentación de credencial donde se muestre que está autorizado para expedir dichas credenciales y que ha sido expedida por el gobierno. Esta autorización puede ocurrir en cadena, donde se requiera interactuar con varios emisores hasta llegar al organismo en el que se confía.

Por último, se va a mostrar la interacción simplificada que se produce cuando un titular envía una presentación de credencial a un verificador:

1. El titular crea una conexión con el verificador a través de DIDComm. El verificador recibe un DIDDoc que asocia un identificador (DID) del titular con una clave pública.
2. Tras el establecimiento de la conexión, el titular construye una presentación de credencial en la que indica los atributos y los predicados solicitados por el verificador.
3. Firma y encripta la presentación y se la envía al verificador, el cual desencriptará y verificará la firma digital del titular.
4. El verificador accederá al VDR para revisar la definición de credencial correspondiente, obteniendo las claves públicas necesarias. Con estas claves, el verificador tratará de verificar que las firmas digitales del emisor contenidas en la presentación recibida son correctas.
5. Por último, el verificador tratara de verificar que la credencial no ha sido revocada.

2.2.4.2 Revelación selectiva y pruebas de conocimiento cero (ZKP)

Una presentación verificable (presentación sobre una credencial verificable) contendrá una serie de afirmaciones a revelar y una serie de predicados (pruebas sobre campos de la credencial). Esto se consigue gracias a dos conceptos fundamentales: la revelación selectiva y las pruebas de conocimiento cero (ZKP).

La **revelación selectiva** (*selective disclosure*) se basa en presentar únicamente las afirmaciones de una credencial verificable que necesita el verificador.

Las pruebas de conocimiento cero o **ZKP** (*Zero-Knowledge Proof*) son una técnica criptográfica presentada en el apartado 2.1 (Técnicas de criptografía básicas) que permiten al titular probar a un verificador que contiene un dato sin tener que revelárselo. ZKP tiene cuatro aplicaciones sumamente relevantes en el ámbito de las credenciales verificables:

- Predicados ZKP. Un predicado es una prueba que permite al probador (titular) probar que cumple con una cierta condición sobre un campo de la credencial o que, directamente, contiene dicho campo, todo ello sin tener que revelar el propio dato.
- Verificar que una credencial fue expedida a un titular sin que este revele necesariamente un identificador correlable, como un DNI. Con esto, el titular puede mandar una prueba de que es, realmente, el titular.
- Ocultar las firmas del emisor en las presentaciones que el titular envía al verificador, con el objeto de que este último sea incapaz de correlar de alguna forma información del titular a través de estas firmas digitales.
- Revocación. Aunque se entrará en más detalle sobre esto en el próximo apartado, se puede adelantar que ZKP se puede utilizar en algunas implementaciones de SSI para probar que una credencial no ha sido revocada, sin tener que exponer unas listas de revocación.

Un ejemplo cotidiano donde se aplican ambas técnicas es el siguiente: se puede requerir revisar una credencial verificable que contenga los campos del DNI (foto, nombre, apellidos, fecha de nacimiento, domicilio, etc.) para evitar que menores de edad consuman cerveza en un bar. El verificador, que en este caso podría ser el dueño del local, únicamente necesitaría que el titular revelase su foto, con el fin de identificar visualmente al sujeto, y que revelase su fecha de nacimiento, con el fin de verificar que es mayor de edad.

En el ejemplo anterior, se está haciendo uso de la revelación selectiva para revelar únicamente las dos afirmaciones correspondientes y se está haciendo uso de ZKP para, al menos, probar que es el titular del DNI sin tener que mostrar el número del documento y para ocultar las firmas del emisor para evitar una posible correlación. Además, dependiendo de la implementación, se podría hacer uso de las otras capacidades de ZKP. Por una parte, se podría probar que la credencial no ha sido revocada. Y, por otra, se podría evitar tener que revelar la fecha de nacimiento al, simplemente, probar que es mayor de edad a través de los predicados ZKP.

2.2.4.3 Revocación

Una credencial puede dejar de ser válida en cualquier momento y por cualquier motivo. Por ejemplo, un ciudadano puede perder todos los puntos del carné de conducir y perder la licencia o, un caso más común, una credencial puede contener información obsoleta, como el domicilio en el caso de un documento de identidad. En estos casos es necesario un método de revocación que pueda ser verificable por un verificador.

En el ámbito de las credenciales verificables, existen dos técnicas para que un verificador pueda verificar que una credencial de la cual ha recibido una presentación verificable no ha sido revocada.

La primera técnica son las **listas de revocación** [30], propuesta por el grupo de la W3C, aunque actualmente no es una recomendación. Estas listas las deberá contener el emisor de credenciales y deberán ser accesibles para que los verificadores puedan consultarlas. Una lista no es más que una ristra de ceros y unos. Cuando una credencial es expedida, se le asocia un índice de esta lista, cuya referencia apuntará a un valor que inicialmente será cero. En el momento que el emisor revoque la credencial, cambiará el número de la lista por un uno.

Según la W3C, estas listas mejoran la privacidad respecto a las técnicas comunes que se usan actualmente, donde un emisor referencia una credencial a un único lugar donde se encuentra si esa credencial ha sido revocada, permitiendo al emisor seguir el rastro de qué hace el titular con dicha credencial y a qué verificadores se la presenta.

Sin embargo, Sovrin (e Hyperledger) no está de acuerdo con hacer uso de estas listas [31], puesto que afirma que la mejora de la privacidad no es suficiente, ya que permite al emisor correlar la identidad del titular con los verificadores a los que presenta credenciales, al estar bajo control del lugar donde las listas se publican. Por otra parte, afirma que las listas suponen problemas técnicos a resolver, como el tener que mantener una API accesible para dar acceso a estar listas.

Sovrin e Hyperledger proponen hacer uso de **ZKP** y de los **acumuladores criptográficos** [32] vistos en el apartado 2.1 (Técnicas de criptografía básicas). Para ello, el emisor publica en el VDR un registro de revocación, donde indica qué acumulador criptográfico puede emplearse para verificar la no revocación de un número determinado de credenciales y, también, indica un punto de acceso (y un *hash*) para acceder a una localización donde se encuentran un fichero que incluye todos los factores del conjunto que resulta en el acumulador indicado. Estos ficheros se denominan ficheros *tails* y el servidor donde se publican se denomina servidor *Tails*.

Cuando un emisor expide una credencial, se le indica al titular qué factor le corresponde de los que se encuentran en el fichero *tails*. El emisor podrá hacer uso de varios de los factores incluidos en este fichero que resultan en el mismo acumulador para expedir varias credenciales. Cuando una de estas credenciales se revoca, un nuevo acumulador que no incluye dicho factor es publicado en el VDR[12], en conjunto con un "corrector" que permite a los verificadores aplicarlo para que se pueda obtener dicho acumulador a partir de todos los factores del fichero *tails*, el cual sigue conteniendo el factor de la credencial revocada.

El titular entonces hará uso de ZKP para enviar al verificador una prueba de no revocación que, en realidad, es una prueba de que contiene un factor que resulta en el acumulador publicado en el VDR. De esta forma, se evita la correlación a partir del factor, el cual podría actuar como seudoidentificador.

Sin embargo, ambas técnicas suponen una mejoría en cuanto a la privacidad, al evitar tener que publicar listas de revocación donde se asocian identificadores con el estado de la revocación.

Como en este TFM se va a hacer uso de Hyperledger, el método de revocación que se aplicará internamente será el basado en ZKP y los acumuladores criptográficos.

[12] En la práctica es común actualizar este acumulador cada cierto tiempo, en lugar de cada vez que una credencial es revocada. De esta forma, un acumulador actualizado puede suponer una diferencia de varios factores eliminados.

2.2.4.4 Tipos de credenciales y tipos de firmas

Existen dos tipos de credenciales relevantes en la actualidad, las recomendadas por la W3C basadas en la tecnología **JSON-LD** [29], y las utilizadas por Hyperledger, denominadas **AnonCreds** [33]. Ambos tipos de credenciales presentan una serie de puntos fuertes y de puntos débiles respecto a la otra.

Uno de los puntos diferenciales más destacables es la capacidad que tienen de soportar la revelación selectiva y las funcionalidades de ZKP descritas anteriormente. Por un lado, las credenciales AnonCreds soportan la revelación selectiva y las cuatro aplicaciones descritas [33]: predicados, no revelación de un identificador potencialmente correlable del titular, no revelación de las firmas digitales del emisor y prueba de no revocación. Y, por el otro lado, las credenciales recomendadas por la W3C dependen de las firmas digitales que se utilicen [34]:

- Las firmas LD no permiten revelación selectiva ni ningún tipo de forma de ZKP.
- Las firmas BBS+ soportan revelación selectiva y soportan únicamente una forma sencilla de ZKP, la cual no soporta ni los predicados ni las pruebas de no revocación.

Profundizando un poco más en las credenciales de Hyperledger, estas ofrecen no solo poder probar la no revocación de una credencial, sino también probar en qué rango temporal una credencial no ha sido revocada [33].

Si se comparan las credenciales AnonCreds con las credenciales JSON-LD con firmas BBS+ únicamente basándose en las capacidades de ZKP que ofrecen cada una, puede parecer que las primeras son superiores ya que ofrecen pruebas de no revocación y el uso de predicados. Sin embargo, no hay que olvidar que la W3C y, por tanto, las credenciales JSON-LD propone un método de revocación y de verificación de esta diferente al de Hyperledger y que se basa en unas listas de revocación [30]. La ventaja, por tanto, dependerá de si se considera o no superior la revocación basada en acumuladores criptográficos. Y en cuanto a los predicados ZKP, es importante mencionar que estos están muy limitados en las AnonCreds, permitiendo únicamente probar campos de la credencial que contengan datos numéricos y con operaciones de comprobación ($<$, $\leq$, $>$, $\geq$) frente a un valor dado [33]. Si permitiesen una lógica más compleja con, por ejemplo, composición de comprobaciones con diferentes campos, esta capacidad podría suponer un punto de superioridad frente a las credenciales JSON-LD pero, al únicamente permitir operaciones sencillas, puede ser una ventaja con un impacto discutible según qué contexto de aplicación.

Otro punto destacable a considerar es la extensión de uso en el mercado. Mientras que las credenciales JSON-LD están recomendadas por la W3C, las credenciales AnonCreds están recomendadas por Hyperledger Indy, una de las soluciones más extendidas actualmente respecto a identidad autosoberana. Además, según Hyperledger, las credenciales AnonCreds son las más utilizadas actualmente [35].

Y con, probablemente, menor relevancia, se pueden tener en cuenta otros aspectos, como la flexibilidad que ofrecen las credenciales JSON-LD al permitir estructuras más complejas de credenciales, frente a la rigidez de las AnonCreds que únicamente permiten credenciales con esquemas sencillos.

2.2.5 Carteras y agentes digitales

2.2.5.1 Cartera digital

El término cartera digital puede hacer referencia a un amplio número de cosas. Por ejemplo, actualmente existen productos como Google Pay o Apple Pay que permiten vincular tarjetas bancarias para realizar pagos seguros a través de ellos. Estos productos son popularmente conocidos como carteras digitales, puesto que "almacenan" las tarjetas bancarias. Sin embargo, para que estos productos fuesen realmente unas carteras digitales, deberían permitir almacenar otros elementos fundamentales además de nuestros métodos de pago, como nuestro documento de identidad o la tarjeta de la universidad o el trabajo.

Una cartera digital, por tanto, se debería entender como aquel producto (software o hardware) que permita almacenar y gestionar digitalmente aquellos elementos que un ciudadano contiene actualmente en su cartera física, como pueden ser las tarjetas bancarias, el documento de identidad o, incluso, una tarjeta de puntos de un supermercado. Estas carteras deberán ser seguras y ofrecer sistemas de recuperación (segura) en caso de pérdida. Por otra parte, deberían permitir la portabilidad de la información que contienen entre diferentes productos.

El concepto de la cartera digital es totalmente aplicable al contexto de la identidad autosoberana. Por tanto, una cartera digital en SSI debería permitir lo siguiente:

- Generar, almacenar, recuperar y eliminar las claves criptográficas necesarias.
- Generar, almacenar, recuperar y eliminar identificades descentralizados, específicamente, los DID privados empleados en las conexiones seguras (*did:peer* y *did:key*).
- Almacenar, recuperar y eliminar credenciales verificables.
- Adicionalmente, poder gestionar otro tipo de datos sensibles del usuario, como contraseñas.

Una cartera digital puede estar contenida en diferentes localizaciones. Para un usuario corriente, la cartera debería estar contenida en alguno de sus dispositivos personales, como el teléfono móvil. Sin embargo, se podría delegar la responsabilidad a un servicio en la nube, de forma que el usuario interacciona con este para gestionar la cartera. Esta última opción presenta muchas deficiencias, como la pérdida de la descentralización, la transmisión en claro (o a través del mecanismo que emplee el servicio) de la información entre el usuario y el servicio, o la necesidad de confiar en dicho servicio.

En la práctica, una cartera digital suele ser una base de datos corriente (como PostgreSQL) con un software que la rodea y que se encarga de cifrarla, además de realizar las funciones descritas anteriormente y de ofrecer una interfaz hacia el exterior. Este software en concreto se denomina servicio de gestión de claves (**KMS**, *Key Management Service*).

2.2.5.2 Agente digital

Aunque una cartera digital permite almacenar elementos descritos anteriormente, como los DID o las VC, aún se necesita de un elemento software que permita interactuar con esta para poder hacer uso de estos elementos para, por ejemplo, crear conexiones o recibir y presentar

credenciales. Además, una cartera digital por sí sola no presenta una interfaz de usuario fácilmente accesible (en términos de usabilidad) ni tampoco presenta un mecanismo de autenticación y autorización de acceso a esta.

Un agente digital suple las deficiencias descritas anteriormente. Por tanto, un agente digital es un recubrimiento software de una cartera digital que permite una interacción accesible desde el punto de vista del usuario con esta, que añade mecanismos de autenticación y autorización para limitar el acceso a esta y que permite hacer uso de los elementos almacenados y gestionados por esta para realizar las diferentes acciones que se han visto hasta ahora, como la de crear conexiones privadas seguras con otros agentes. Hasta ahora, se ha referido como nodo a lo que se conoce como agente. Por tanto, un agente es el elemento software gestionado por una entidad.

Desde el punto de vista del ecosistema de credenciales verificables (Figura 2), un agente digital puede asumir uno de los roles especificados en un momento dado para poder realizar las funciones asociadas a dicho rol, como el caso de expedir credenciales verificables en el caso del emisor. Esto no implica que un agente deba elegir un rol y adherirse indefinidamente a este, sino que un agente puede asumir libremente cualquiera de los roles según la interacción que esté teniendo en ese momento con otro agente.

Por un lado, un agente digital debe permitir a una entidad solicitar a la cartera realizar todas las funciones descritas en el punto anterior: gestionar claves criptográficas, identificadores descentralizados, credenciales verificables, etc. Por otro lado, el agente debe permitir a la entidad hacer uso de estos elementos para realizar algunas de estas funciones:

- Crear conexiones seguras con otros agentes a través de DIDComm.
- Emitir credenciales verificables a un titular.
- Recibir, notificar y aceptar credenciales verificables emitidas por un emisor.
- Construir presentaciones sobre credenciales verificables que incluyan técnicas como la revelación selectiva o ZKP.
- Enviar estas presentaciones verificables un verificador.
- Verificar las presentaciones recibidas por un titular.
- Interactuar con el VDR correspondiente para escribir o leer transacciones de elementos relevantes, como los DID públicos o los esquemas de credenciales.
- Intercambiar mensajes con otro agente a través de las conexiones seguras.

Se han mencionado las acciones más comunes, pero eso no significa que puedan existir otras.

La entidad que controla un agente se la conoce como el controlador (*controller*) de este. Dependiendo de la naturaleza del controlador, un agente puede ser de un tipo o de otro:

- **Agente personal**. El controlador de un agente personal es una persona física. Normalmente, un agente personal es una aplicación contenida en el teléfono móvil de la persona, aunque también puede ser un programa informático contenido en otro tipo de dispositivo personal, como en un ordenador. Sin embargo, un agente personal podría incluso ser un servicio en la nube accesible a través de, por ejemplo, un navegador web, aunque esta práctica no es recomendable para agentes personales.

- **Agente empresarial** (o de servidor). En este caso, el controlador es una persona jurídica. La organización o empresa tendrá un único agente que la representa y que le permitirá tomar los roles de verificador y emisor. Adicionalmente, un agente empresarial podrá tomar el rol de titular cuando necesite, por ejemplo, recibir ciertas acreditaciones de organismos públicos. Estos agentes, al contrario que los agentes personales, suelen estar contenidos en servidores privados o en públicos bajo el control de la entidad. Es posible, también, que la gobernanza de una organización requiera de todos los individuos que la componen tener agentes personales con credenciales de autorización emitidas por el agente empresarial para poder operar en nombre de la organización.
- **Agentes de encaminamiento**. Como se vio en el apartado 2.2.3.6 (DIDComm), una conexión puede requerir de nodos intermediarios que encaminen los mensajes intercambiados entre dos agentes. Estos agentes son estos nodos intermediarios.
- **Agentes IoT**. Un agente IoT (Internet de las Cosas) es un agente que es controlado por un dispositivo dentro de un entorno IoT. La existencia de estos agentes posibilita la verificación en ambos sentidos entre un servidor y un dispositivo para evitar la personificación de alguno de estos por parte de un tercero malicioso. Además, permite al dispositivo generar credenciales verificables con los datos que recoge para permitir al servidor que los recibe la posibilidad de presentárselos a un verificador (por ejemplo, un perito) y que este verifique que no han sido manipulados.

2.2.6 El proyecto ToIP

La Fundación Linux tiene un proyecto denominado **ToIP** (*Trust over IP Foundation*) cuyo objetivo es definir una arquitectura estándar y robusta de confianza digital en Internet [36]. Para ello, presenta la siguiente pila de protocolos de 4 niveles con una vista dual:

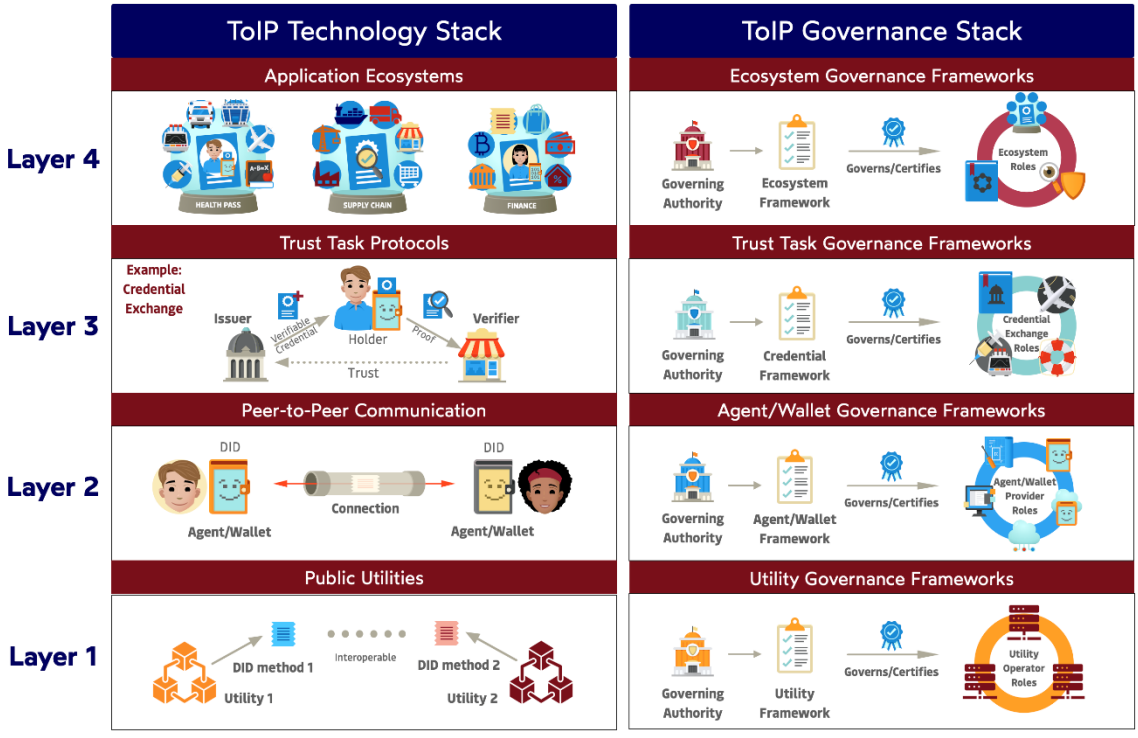

Figura 4: Pila ToIP [36]

Esta pila de protocolos presenta una doble visión para todos los niveles de protocolos, apareciendo a la izquierda la visión tecnológica y a la derecha la visión de la gobernanza asociada. Se van a describir los cuatro niveles de este modelo desde el punto de vista tecnológico.

En el **nivel 1** se encuentran los elementos básicos que permiten asentar las bases para establecer una confianza digital en Internet. Estos elementos son los identificadores descentralizados descritos en el apartado 2.2.3 y los registros de datos verificables vistos en el apartado 2.2.2. En este nivel, cuando se habla de confianza se refiere a la confianza criptográfica, ya que únicamente se asegura la asociación de los identificadores con las claves públicas.

En el **nivel 2** se encuentran las conexiones seguras establecidas a través de DIDComm (apartado 2.2.3.6) a través de los agentes (apartado 2.2.5). En este nivel la confianza sigue siendo desde el punto de vista criptográfico, ya que únicamente se asegura que el intercambio de mensajes entre dos agentes es seguro.

En el **nivel 3** se introduce el intercambio de credenciales verificables entre agentes y el establecimiento de los roles de los agentes según el ecosistema de credenciales verificables (apartado 2.2.4). En este nivel la confianza es tanto criptográfica, con la verificación de las credenciales, como humana, con la información intercambiada entrando en el escenario.

En el **nivel 4** se establece el ecosistema de confianza al hacer uso de los niveles anteriores, donde un conjunto de agentes se intercambian credenciales verificables con un propósito concreto. Por ejemplo, un sistema de confianza puede ser un sistema sanitario que expide credenciales sanitarias a sus ciudadanos. La confianza en este caso es humana.

El sistema implementado en este TFM seguirá la arquitectura estándar definida en este apartado.

2.3 Tecnologías de identidad autosoberana

Ya se han visto los conceptos fundamentales de la identidad autosoberana. En este apartado se va a describir las tecnologías Hyperledger que permiten implementar un ecosistema de SSI. Además, se presentarán alternativas de implementación a Hyperledger al final del apartado.

2.3.1 Blockchain e Hyperledger

Antes de pasar con las tecnologías Hyperledger que implementan soluciones SSI, es importante mencionar qué es Hyperledger y presentar brevemente la tecnología blockchain.

2.3.1.1 Blockchain

Ya se ha mencionado la importancia de utilizar blockchain como tecnología que implementa un VDR, con el fin de tener una identidad verdaderamente descentralizada y de poder sustituir la confianza depositada en terceros por confianza criptográfica. En este punto se presentará brevemente la tecnología blockchain.

Blockchain es un registro de transacciones distribuido entre un conjunto de nodos conectados entre sí a través de una red descentralizada. Todas las transacciones que contiene el registro son completamente inmutables y transparentes, por lo que una vez que una transacción se integra en

este, esta transacción quedará para siempre accesible en la blockchain, sin posibilidad de modificarla o eliminarla.

Esta inmutabilidad es posible gracias al funcionamiento interno de la tecnología que, a grandes rasgos, hace uso de las funciones criptográficas resumen para encadenar agrupaciones de transacciones entre sí, formando una cadena. Cada agrupación de transacciones se conoce como bloque e incluye, además de las propias transacciones, una referencia al bloque anterior generada a partir del contenido de este. Por tanto, si una transacción de un bloque cambia, la referencia cambiará y no coincidirá con la que incluye el bloque siguiente, rompiéndose así la cadena.

Al ser una red descentralizada, no existe un único administrador que se encarga de escribir transacciones en el registro. Por tanto, es necesario un mecanismo de consenso que involucre a los nodos de confianza de la red para que estos acuerden de forma consensuada publicar un bloque de transacciones en la cadena. El mecanismo de consenso que se utilice es el que marca qué nodos pueden ser de confianza y cuál es la forma de llegar a un consenso. Por ejemplo, el mecanismo de consenso **PoW** (*Proof of Work*) permite a cualquier nodo de la red participar en él, y se basa en resolver un problema que es computacionalmente muy costoso, pero que es muy sencillo de validar una vez se tiene la solución. Este problema se basa en aplicar la función resumen en la cabecera de un bloque y en un número desconocido para encontrar un resumen cuyo valor hexadecimal sea menor a un número dado. Sin embargo, existen muchos mecanismos de consenso diferentes, algunos más útiles para redes abiertas y otros más adecuados para redes privadas.

Una blockchain se puede clasificar según si su acceso es abierto o cerrado y según si su validación (escritura) es abierto o cerrado.

Según el **acceso**, una blockchain se dice que es pública si permite a cualquier nodo recuperar y leer transacciones almacenadas en el registro sin tener una invitación previa. Y una blockchain se dice que es privada si, para ello, necesita una invitación previa y un proceso de autorización.

Según la **validación**, una blockchain se dice que es no "permisionada" (*permissionless*) si cualquier nodo puede participar en el mecanismo de consenso en el momento de publicar una transacción en la cadena. Y una blockchain se dice que es "permisionada" (*permissioned*) si únicamente existe un subconjunto de nodos del conjunto total que tiene permiso para participar en el mecanismo de consenso. Normalmente, este subconjunto de nodos suelen ser nodos de cierta manera confiables (organismos públicos, empresas grandes, etc.).

Por dar algunos ejemplos, bitcoin es una blockchain pública no permisionada, una red Hyperledger Fabric es una blockchain privada permisionada, y una red Hyperledger Indy es una blockchain pública permisionada. Como se puede ver, no todas las blockchain tienen que ver con las criptomonedas.

En este Trabajo Fin de Máster se van a emplear varias redes Hyperledger Indy, que como se ha visto son redes con un acceso público y una validación permisionada. Por tanto, una transacción escrita en esta blockchain quedará pública para todo el mundo que quiera acceder a ella. Esto facilita la función de un Registro de Datos Verificables (VDR), donde cualquier verificador del mundo pueda acceder al registro para recuperar un identificador descentralizado público, un esquema de credencial, un registro de revocación, etc.

2.3.1.2 Fundación Hyperledger

La Fundación Hyperledger es una asociación global colaborativa y abierta de empresas y otro tipo de organismos o individuos de diferentes sectores que tiene como objetivo la creación de proyectos de código abierto orientados a ofrecer soluciones blockchain para un uso industrial [35]. Esta asociación incluye grandes compañías como IBM, Huawei, Oracle o Sovrin.

Actualmente tienen un gran número de proyectos con diferentes grados de madurez. Los proyectos que están en la fase más madura son los siguientes [35]: Fabric, Sawtooth, Iroha, Besu, Indy y Aries.

Hyperledger **Fabric** es un proyecto enfocado en el desarrollo de soluciones con una arquitectura modular para redes privadas. Este carácter modular permite emplear este proyecto de multitud de casos de uso industriales. Presenta varias características interesantes, como la posibilidad de crear los llamados canales, que son diferentes registros en los que participan diferentes subconjuntos de nodos. También presenta un proveedor de servicios de suscripción (MSP) que gestiona la autenticación y autorización de la red.

Hyperledger **Sawtooth** es un proyecto con una arquitectura modular pensado para redes privadas que permite abstraer el núcleo de la aplicación de dominio, facilitando así a los desarrolladores que implementan las reglas de negocio.

Hyperledger **Iroha** es un proyecto que permite integraciones simples y sencillas en proyectos que incluyen redes IoT y que necesitan de un registro distribuido.

Hyperledger **Besu** es un cliente Ethereum que está diseñado para ser fácil de usar por las empresas. Se puede emplear tanto en redes públicas como en redes privadas. Incluye varios mecanismos de consenso, como PoS (*Proof of Stake*) o PoW (*Proof of Work*). Un ejemplo industrial relevante en el contexto de este TFM donde se usa este proyecto es el proyecto europeo EBSI [37] mencionado anteriormente.

Hyperledger **Indy** es el elemento central en el que se basa la identidad autosoberana que ofrece Hyperledger. El proyecto se compone de diferentes módulos que implementan un registro distribuido público permisionado pensado para funcionar como un VDR e implementa un conjunto de herramientas que permiten interactuar con la blockchain.

Hyperledger **Aries** es un proyecto que se contiene una serie de herramientas que permiten el desarrollo de agentes digitales capaces de crear, transmitir y almacenar credenciales digitales. Este proyecto no incluye un VDR específico siendo, de hecho, agnóstico de este. Esto significa que Hyperledger Aries puede operar con Hyperledger Indy o con cualquier otro VDR.

Adicionalmente, existen otros dos proyectos importantes en el contexto de este trabajo que tienen un nivel de madurez menor: Ursa y AnonCreds.

Hyperledger **Ursa** es una biblioteca criptográfica separada de todos los proyectos Hyperledger y que puede ser aprovechada por estos e, incluso, por otros proyectos no basados en Hyperledger. La razón de ser de este proyecto es la de abstraer la complejidad criptográfica de otros proyectos. En el caso de la identidad autosoberana, tanto Aries como Indy harán uso de las herramientas de Ursa para generar pares de claves asimétricas, encriptar y desencriptar, firmar digitalmente y

verificar estas firmas, aplicar funciones resumen, generar y verificar pruebas ZKP y hacer uso de los acumuladores criptográficos para revocar credenciales.

Hyperledger **AnonCreds** es un proyecto reciente que aísla las credenciales Hyperledger denominadas AnonCreds del proyecto Hyperledger Indy con el objetivo de que se convierta en un formato de credencial agnóstico a la blockchain en la que se quiera usar. Las capacidades y características de estas credenciales ya se han mencionado anteriormente.

Los proyectos que se van a emplear en este TFM son Hyperledger Indy, Hyperledger Aries e, indirectamente, Hyperledger Ursa. Los tres proyectos forman lo que se denomina en este documento como la pila (*stack*) de identidad Hyperledger, la cual permiten implementar los conceptos de SSI vistos anteriormente. En los siguientes capítulos se profundizará en el funcionamiento y las capacidades de los proyectos Indy y Aries, pero no en el proyecto Ursa, ya que este último es consumido por los otros dos proyectos y su funcionamiento no transciende al nivel del desarrollador.

La razón por la cual no se hace uso del proyecto Hyperledger AnonCreds es porque al comienzo de este Trabajo Fin de Máster, este proyecto aún no existía.

Es importante mencionar que originalmente no existían los proyectos Hyperledger Aries, Ursa y AnonCreds. Estos se han aislado del proyecto original Hyperledger Indy según iba evolucionando este.

2.3.2 Hyperledger Indy

Hyperledger Indy es un proyecto que se compone de varios módulos que ofrecen una solución blockchain como VDR y una interfaz hacia esta. Estos módulos son los siguientes repositorios alojados en GitHub: *indy-node*, *indy-plenum*, *indy-sdk*, *indy-vdr* e *indy-shared-rs* [38], [39], [40], [41], [42]. Los dos primeros repositorios implementan la propia blockchain, mientras que el tercero implementa una interfaz que permite operar con esta. Este tercer repositorio se está dividiendo en varios repositorios para aumentar la modularidad, de forma que se están concibiendo los repositorios cuatro y cinco de la lista.

En cuanto a la blockchain empleada, esta utiliza un mecanismo de consenso basado en el protocolo de la Tolerancia a Fallos Bizantinos (BFT), denominado **PBFT** (*Plenum Byzantine Fault Tolerant*) [39]. El mecanismo BFT se basa en que como máximo N' nodos pueden ser maliciosos o incorrectos, donde $N' = \frac{(Nt-1)}{3}$, siendo Nt el número total de nodos que participan en el proceso de validación [43]. Esta blockchain es, como se ha mencionado anteriormente, una red con acceso público y con validación permisionada.

Las publicaciones relacionadas con este proyecto se encuentran en un formato de *Request For Comments* (RFC) en el repositorio alojado en GitHub denominado *indy-hipe* [15]. Estas publicaciones describen los estándares relacionados con el ecosistema Indy. El funcionamiento de estas publicaciones es el siguiente: un contribuidor propone una publicación que define una funcionalidad o un estándar para que posteriormente sea debatido por la comunidad y pueda pasar a un estado de aceptación, donde pasará a implementarse. Posteriormente, una publicación puede pasar por la adopción por parte de la comunidad o por reemplazarse por otra publicación.

2.3.2.1 AnonCreds

Hyperledger Indy permite utilizar tanto las credenciales JSON-LD recomendadas por la W3C como las credenciales AnonCreds creadas por Hyperledger. Ya se han mencionado las ventajas y desventajas de cada una de ellas (apartado 2.2.4.4) y, en base a estas, se ha tomado la decisión de usar las credenciales **AnonCreds** en este Trabajo Fin de Máster por, principalmente, los dos siguientes motivos: (1) permite la aplicación de ZKP para probar predicados y la no revocación de credenciales, y (2) son las credenciales que Hyperledger recomienda.

La siguiente figura realizada por Hyperledger muestra la implementación de las credenciales AnonCreds y, además, muestra los tipos de transacciones que se publican en la blockchain:

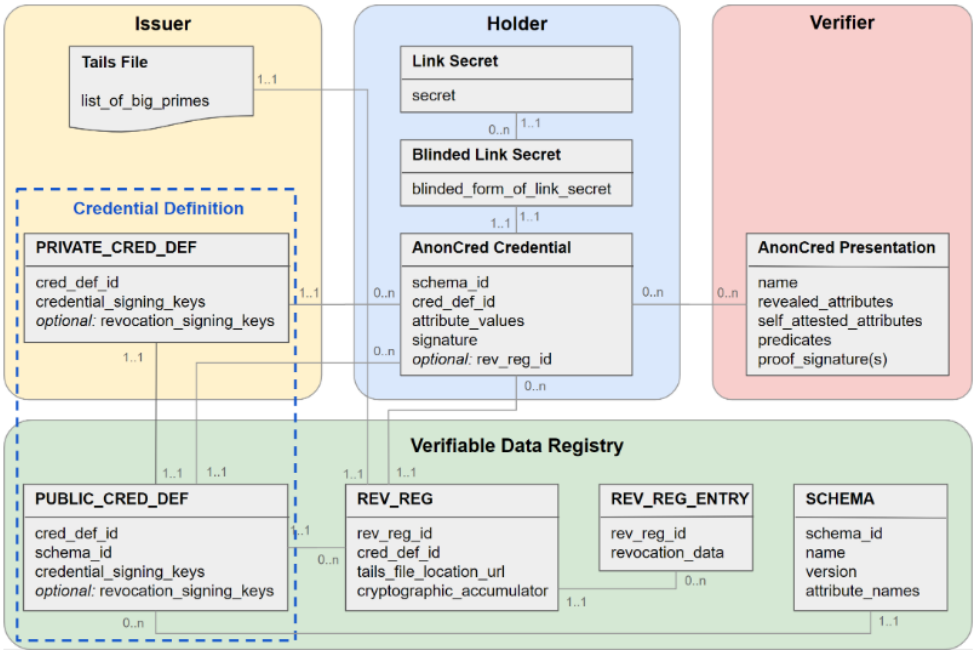

Figura 5: AnonCreds, por Hyperledger [33]

En la figura anterior se pueden observar varias cosas relevantes que se relacionan con los conceptos de SSI detallados en apartados anteriores.

Respecto a las credenciales y presentaciones verificables, se puede ver como una credencial que almacena un titular contiene el identificador del esquema y el de la definición de credencial empleados por el emisor, la firma digital del emisor y los atributos de la propia credencial. Luego, la presentación de esta credencial no es la propia credencial, sino los atributos a revelar, los predicados ZKP a probar y una prueba ZKP de la firma del emisor.

Respecto a los elementos que se almacenan en el VDR, se puede ver como se almacenan los siguientes elementos:

- Esquemas de credenciales. Un esquema contiene los atributos que van a tener las credenciales que se expidan usando este esquema. Un esquema además tiene un nombre y un identificador, utilizados para manipularlos desde el punto de vista humano (nombre) y desde el punto de vista técnico (identificador). Un esquema se puede actualizar, actualizándose así su versión.

- Definiciones de credenciales. Una definición de credencial contiene todas las claves públicas que utilizará el emisor para firmar cada uno de los atributos de un esquema. Incluye, también, un identificador al esquema empleado y el identificador propio de la definición. Es importante fijarse en que esta definición de credencial realmente se compone de dos elementos, el que se almacena en el VDR con las claves públicas y el que se almacena en la cartera del emisor con las claves privadas. Otro detalle a mencionar es que un emisor puede emplear un esquema publicado por otro emisor, pero debe publicar su propia definición de credencial sobre dicho esquema.
- Registros de revocación. Cuando un emisor publica una definición de credencial, deberá también publicar un registro de revocación que incluya el identificador de la definición asociada, el acumulador criptográfico, la localización del fichero tails con los factores del acumulador y el identificador del propio registro. Además, irá publicando según vaya revocando credenciales unas transacciones que actualizan el acumulador.
- DID público. Aunque no aparece en la figura anterior, la blockchain Hyperledger Indy también almacena los DID (DID y DIDDoc) públicos de los emisores.

Por último, se puede observar como el emisor es el que está a cargo de los ficheros tails asociados. En el VDR únicamente se publica una referencia a la localización de este. En la práctica, un emisor puede publicar su propio servidor Tails o hacer uso de un servidor mantenido por terceros.

Existen varios motivos por los cuales no se publica otro tipo de información al VDR, como las propias credenciales o los DID privados. El motivo principal es el de preservar lo máximo posible la privacidad de la información al evitar filtraciones de información o la correlación de esta a partir de, por ejemplo, DID privados. Otro motivo es el de los costes de escritura de transacciones que suelen tener asociados las redes Indy públicas, ya que mantener los nodos de una blockchain cuesta dinero. Por tanto, económicamente no es viable tener que publicar transacciones cada vez que, por ejemplo, se crea una conexión privada entre dos agentes. Todas las transacciones que se publican en la red están asociadas a los emisores, por lo que son los únicos que deben pagar para ello. Por último, una blockchain no es eficiente en tiempos de respuesta, ya que por cada escritura se tiene que pasar por un proceso de consenso.

2.3.2.2 Interactuar con una red Indy

Para que un agente sea capaz de conectarse de forma segura a una red Indy necesita recuperar un elemento de la red denominado **fichero génesis**. Este fichero contiene los puntos de acceso físicos a los nodos de la red y material criptográfico para asegurar las conexiones. En el caso de utilizar Hyperledger Aries, será este el que se encargue de procesar el fichero y de conectarse automáticamente a la red.

Dependiendo de la red, un emisor puede necesitar pedir permiso de escritura en la red. El mecanismo común para pedir este acceso es a través de registrar a través de un formulario el DID correspondiente e información sobre la organización que lo controla. Es importante distinguir aquí a un nodo que tiene permiso de escribir transacciones en la red y a un nodo que participa en

el proceso de validación. Estos últimos son nodos de confianza autorizados por los administradores de la red, Sovrin se refiere a estos nodos como *Stewards* [44].

Por último, la mayoría de las redes públicas requieren a los agentes aceptar un acuerdo denominado **TAA** (*Transaction Author Agreement*) cada vez que quieran publicar una transacción en la red. Este acuerdo es similar a los Acuerdos de Licencia con el Usuario Final (ALUF), más conocido por sus siglas en inglés EULA (*End-User License Agreement*), donde un usuario debe aceptar una serie de condiciones establecidas por el propietario de un servicio software para poder hacer uso de este. En el caso de un acuerdo TAA, el controlador de un agente que vaya a solicitar escribir transacciones en la blockchain debe aceptar las condiciones con implicaciones legales establecidas por los administradores de la blockchain. Por ejemplo, la red Sovrin tiene establecido un TAA que implican una serie de condiciones, como la de aceptar los costes de escritura en la red o que no se podrá publicar transacciones que contengan datos personales sin la aprobación explícita de la fundación Sovrin [45].

La existencia de un acuerdo previo (TAA) al uso del servicio que ofrece una blockchain favorece el cumplimiento de las regulaciones europeas, como la RGPD, ya que el agente debe aceptar explícitamente estas condiciones por cada transacción que publique en la red. Sin embargo, en el caso de Hyperledger Aries, existe la posibilidad de configurar un agente para que acepte estos acuerdos automáticamente por cada transacción publicada, facilitando la experiencia de usuario. Por otra parte, se puede especificar a la hora de aceptar un TAA si se quiere que la aceptación sea válida para la sesión actual o únicamente para esa transacción en concreto.

Aunque no se ha mencionado antes, existen otras dos transacciones que se escriben en un registro VDR en las redes que requieran la aceptación de un TAA. Una de estas transacciones contiene los campos necesarios para que un agente sea capaz de aceptar, desde el punto de vista técnico, el acuerdo. Y la otra transacción es el propio acuerdo y se usará para referenciarla internamente, a través del hash de la transacción, en todas las transacciones que se escriban en la red por parte del agente que ha aceptado el TAA. Por ejemplo, Sovrin tiene publicada en su blockchain principal una transacción con el mecanismo de aceptación[13] y dos transacciones con los propios TAA, una para la versión 1[14] del acuerdo y otra para la versión 2[15].

2.3.2.3 Proyecto Indy Tails Server

Hyperledger Indy y las credenciales AnonCreds hacen uso de una revocación basada en acumuladores criptográficos, de forma que publica en el VDR unas transacciones donde se indica el acumulador criptográfico y una referencia a un fichero tails, que contiene todos los factores que resultan en dicho acumulador. Cuando se revocan varias credenciales, el acumulador es actualizado por el emisor.

Este fichero tails se encuentra en un servidor de ficheros controlado por el emisor, idealmente. Este servidor es conocido como servidor Tails. Por tanto, un emisor debería desplegar y configurar su propio servidor Tails en lugar de emplear uno público ofrecido por otra entidad.

[13] https://indyscan.io/tx/SOVRIN_BUILDERNET/config/9
[14] https://indyscan.io/tx/SOVRIN_BUILDERNET/config/10
[15] https://indyscan.io/tx/SOVRIN_BUILDERNET/config/265

Para facilitar el proceso, la comunidad de British Columbia provee un proyecto de código libre llamado **Indy Tails Server** [46] que permite desplegar fácilmente un servidor Tails a través de contenedores Docker.

2.3.2.4 Proyecto VON

Como ya se ha mencionado, una red Hyperledger Indy es pública en acceso y permisionada en validación. Si nos centramos únicamente en la parte pública, esta característica es necesaria para los requisitos que tiene un VDR, ya que cualquier verificador del mundo puede necesitar acceder en algún momento a este registro. Por esta razón, es importante que una red Hyperledger Indy sea, no solo pública, sino extendida y conocida. No obstante, estas redes suelen tener una capa de autenticación y autorización al tener que solicitar acceso a estas, suelen tener asociado un acuerdo TAA que hay que aceptar y suelen tener unos costes de escritura asociados.

Todas las particularidades descritas en el párrafo anterior pueden resultar poco adecuadas para un proyecto de SSI que está en sus primeras fases de desarrollo. Como consecuencia, en estas fases iniciales se pueden tomar tres decisiones. La primera sería, directamente, lidiar con estas barreras descritas desde el principio, pero esto no es lo ideal. La segunda sería emplear un VDR diferente a Hyperledger Indy, tal y como se describió en el apartado 2.2.3.5 (Métodos), como el uso GitHub para ello con el método *did:github*. Esta segunda opción es perfectamente viable, aunque lo ideal sería poder emplear el mismo tipo de VDR tanto para la fase de desarrollo como para la fase de producción. Por tanto, una tercera opción es la de desplegar una red Hyperledger Indy en local o en un entorno controlado para hacer pruebas con esta. El problema de esta última opción es la complejidad que lleva asociada al, por ejemplo, tener que lidiar con cosas como la configuración del registro o la creación de una interfaz gráfica que permita revisar las transacciones.

El proyecto **VON** (*Verifiable Organizations Network*) [47] ofrece un despliegue sencillo de una red Hyperledger Indy basado en contenedores Docker. Este proyecto se encarga de toda la complejidad asociada a la configuración de una red Indy y ofrece una interfaz web que permite varias cosas, como revisar transacciones, acceder al fichero génesis o autorizar de forma sencilla a un agente para que pueda escribir en el registro.

El despliegue es muy sencillo, simplemente hace falta clonar el repositorio y ejecutar un comando en el terminal. Este despliegue se compone de un contenedor que contiene la interfaz web y de cuatro contenedores que forman el clúster de nodos que operan la red.

Este proyecto ha sido creado por el programa de identidad digital y confianza del Ministerio de Servicios al Ciudadano del gobierno de British Columbia, Canadá [48]. Este programa tiene varios proyectos activos relacionados con la identidad autosoberana, por lo que se puede observar que el gobierno de British Columbia está apostando por esta idea.

En este Trabajo Fin de Máster se va a utilizar la red VON tanto para la Prueba de Concepto como para el Producto Viable Mínimo. Adicionalmente, en el MVP se integrarán al final del desarrollo dos redes públicas para validar el proyecto.

2.3.2.5 Redes públicas

En este apartado se van a presentar las redes basadas en Hyperledger Indy actualmente desplegadas con mayor relevancia y con mayor extensión.

La red **BCovrin** es mantenida por la comunidad de British Columbia. Esta blockchain dispone de dos ramas, una de desarrollo (*dev*) [49] y otra de pruebas (*test*) [50]. En ambas ramas cualquier agente puede auto concederse los permisos para poder escribir transacciones en la blockchain, por lo que no hay ningún tipo de control sobre quién puede escribir y quién no. Adicionalmente, ninguna de las ramas solicita aceptar ningún acuerdo TAA. Los propios desarrolladores no garantizan su fiabilidad [34]. Sin embargo, es una solución perfectamente viable para las primeras fases del desarrollo de un proyecto o para hacer pruebas con una red Indy.

La red **Sovrin** es mantenida por la Fundación Sovrin, una organización sin ánimo de lucro cuyo objetivo es el de garantizar un sistema de identidad pública y accesible de forma global [13]. La Fundación Sovrin contribuyó directamente al código inicial de Hyperledger Indy [51]. En este caso, existen tres ramas: la rama *BuilderNet*, una blockchain gratuita [52] para probar proyectos que aún no están en producción[16]; la rama *StagingNet*, una blockchain de pago (500$ por seis meses de acceso más gastos específicos por transacción [52]) para los prototipos y demostraciones de aplicaciones en preproducción; y la rama *MainNet*, una blockchain de pago (5.000$ por un año de acceso más gastos específicos por transacción [52]) para aplicaciones en fase de producción. Todas las ramas necesitan de una solicitud formal para solicitar permiso para publicar transacciones en la red, aunque la rama *BuilderNet* permite hacerlo manualmente a través de un simple formulario. Adicionalmente, se necesita aceptar un acuerdo TAA [45] para escribir en cualquiera de las ramas.

La red **IDunion** [53] es una propuesta europea (alemana) secundada por grandes organizaciones, como Deutsche Bank o el Ministerio de Economía alemán, y que, según afirman, satisface los reglamentos eIDAS y RGPD [54]. Aunque se pueden observar las transacciones de dos ramas diferentes, llamadas *Test* y *Pilot*, a través de un explorador de transacciones [55], el alumno solo ha encontrado un formulario de solicitud de escritura [56] y un acuerdo TAA definido [57] para la rama *Test*. Además, a fecha de publicación de este documento, la rama *Pilot* solo contiene un número limitado de transacciones con algunas de ellas muy espaciadas en el tiempo (en el orden de días), con lo que el alumno ha concluido que dicha rama aún está en fase de desarrollo por parte del equipo IDunion y que aún no está preparada para ser accesible públicamente. En cuanto al formulario de solicitud de acceso de la rama *Test*, este solo acepta a personas jurídicas como solicitantes [56].

La red **Indicio** dispone de tres ramas [58]: la rama gratuita *TestNet* enfocada a pruebas, la rama gratuita *DemoNet* enfocada a demostraciones y prototipos, y la rama de pago (5.000$ al año) *MainNet* para aplicaciones en fase de producción. Para escribir en estas ramas, se debe aceptar un acuerdo TAA [59].

[16] La rama *BuildetNet* dejó de proporcionar acceso al público desde el día 01/02/2023 [137]. El comunicado fue posterior al desarrollo del TFM, por lo que esto no se pudo tener en cuenta para las decisiones tomadas.

La existencia de múltiples redes y ramas donde escoger hace fundamental que exista un mecanismo de interoperabilidad, de forma que un agente verificador sea capaz de saber en qué red y en qué rama debe buscar un elemento, como un DID público. En este momento, esta responsabilidad cae en el propio agente (Hyperledger Aries), pero se está trabajando en un método DID llamado *did:indy* [24] que permite especificar explícitamente en el propio DID qué red y qué rama almacena el documento asociado. Este método ya se ha visto en el apartado 2.2.3.5 (Métodos).

En este Trabajo Fin de Máster se hará uso de la rama "*dev*" de la red **BCovrin** para las partes del desarrollo en las que se necesite hacer pruebas con redes públicas, y la rama "*BuilderNet*" de la red **Sovrin** para las demostraciones. Se han descartado el resto de las ramas de la red Sovrin por ser de pago, la red Indicio por tener menor extensión que la de Sovrin y la red IDunion por la dificultad a la hora de solicitar acceso de escritura, ya que el alumno debería contactar con el rectorado de la Universidad Politécnica de Madrid para que realice esta solicitud por él. En otras aplicaciones de identidad autosoberana en el ámbito de la Unión Europea, sin embargo, parece buena opción usar la red IDunion, al estar alineada con las regulaciones europeas eIDAS y RGPD.

2.3.3 Hyperledger Aries

Hyperledger Aries es un proyecto que permite la creación de agentes digitales que pueden crear conexiones seguras basadas en DIDComm y expedir, almacenar, presentar y verificar credenciales verificables. Este proyecto es agnóstico al VDR empleado y permite tanto el uso de credenciales AnonCreds como de credenciales JSON-LD.

Las publicaciones relacionadas con este proyecto están en un formato de RFC, al igual que pasa con Hyperledger Indy. En este caso, están alojadas en GitHub en un repositorio denominado *aries-rfcs* [14]. El funcionamiento de esas publicaciones es igual al de las publicaciones de Indy.

En este proyecto entra en juego un aspecto que es fundamental tratar con el fin de evitar problemas y es la interoperabilidad entre agentes. El repositorio *aries-rfcs* contiene un gran número de publicaciones que van evolucionando a lo largo del tiempo, de forma que un agente puede estar implementado siguiendo unas publicaciones en una versión concreta y otro agente puede estar implementado siguiendo las mismas publicaciones con otra versión o, incluso, siguiendo otras diferentes. Esto implica que un agente puede ser incapaz de interactuar con otro. Por este motivo, existen los denominados Perfiles de Interoperabilidad Aries (**AIP**, *Aries Interop Profiles*) (**RFC 0302** [60]) que establecen qué protocolos y qué versiones de estos se deben emplear en la implementación de un agente.

Actualmente existen dos versiones de AIP, la versión 1.0 y la versión 2.0. Una de las diferencias principales, es que AIP 2.0 permite el uso de otros VDR diferentes a Hyperledger Indy y de otros formatos de credenciales. Sin embargo, tal y como se verá en el apartado 2.3.3.3 (Los frameworks de Hyperledger Aries), la versión 2.0 aún no está integrada completamente. Dado que en este TFM se va a hacer uso de una blockchain Hyperledger Indy y de credenciales AnonCreds, no existe ningún motivo relevante para usar AIP 2.0 en lugar de AIP 1.0.

2.3.3.1 Protocolos Aries

Como se ha mencionado, en el repositorio *aries-rfcs* se publican todas las publicaciones de diferentes protocolos que implementa Hyperledger Aries. Sin embargo, no hay que confundir el protocolo DIDComm con estos protocolos, comúnmente denominados protocolos Aries.

DIDComm es un protocolo que se encarga de intercambiar mensajes entre dos agentes, independientemente del contenido de estos mensajes. Este protocolo funciona en un nivel inferior.

Los protocolos Aries especifican el contenido de los mensajes para lograr un objetivo, como el poder expedir, presentar o verificar una credencial. Estos protocolos funcionan a un nivel superior a DIDComm. En el resto de este apartado se van a detallar algunos protocolos Aries relevantes que forman parte del AIP 1.0.

Protocolo de establecimiento de una conexión (**RFC 0160** [61]). En este protocolo se definen una serie de pasos necesarios para que un **invitador** y un **invitado** establezcan una conexión. El invitador debe crear una invitación y enviársela o mostrársela al que será el invitado a través de un mecanismo externo al ecosistema SSI, como la presentación de un código QR. Tras esto, el invitado deberá solicitar el establecimiento de una conexión al invitador. Ante esta solicitud, el invitador podrá aceptarla y la conexión se creará entre ambos agentes. Sin embargo, para que la conexión esté completamente activa, debe producirse alguna interacción entre ambos agentes, como un ping o un mensaje. A continuación se muestra un diagrama de secuencia de elaboración propia con los pasos mencionados y con los estados a nivel técnico por los que una conexión va pasando:

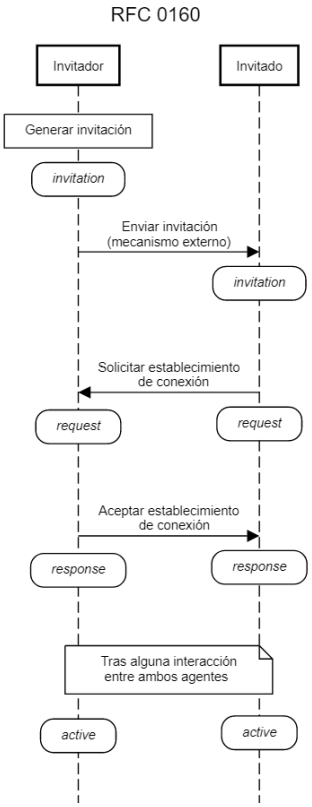

Figura 6: Protocolo Aries 0160

Este protocolo podrá ser sustituido por el protocolo de intercambio de DID (RFC 0023 [62]) en un futuro [61].

Protocolo de mensajes simples (**RFC 0095** [63]). Este sencillo protocolo detalla el contenido de un mensaje para que un **emisor** y un **receptor** puedan intercambiarse mensajes simples, como en un sistema de mensajería. En este caso no existen estados ni secuencia de interacciones.

Protocolo de expedición de credenciales (**RFC 0036** [64]). En este protocolo se detallan una serie de interacciones para que un **emisor** pueda expedir una credencial a un **titular**. El emisor envía una oferta de credencial al titular, el cual deberá decidir si la acepta o no y mandar una solicitud de expedición de dicha credencial al emisor. Tras esto, el emisor mandará al titular la credencial verificable, completando así el proceso. Existe un paso opcional previo a la oferta que permite a un titular iniciar el proceso de expedición, de forma que este envía una propuesta de credencial al emisor, con la que el emisor podrá formar una oferta de credencial. A continuación se muestra un diagrama donde se muestran los pasos mencionados y con los estados a nivel técnico por los que el proceso va pasando, aunque en este caso el diagrama está tomado de la publicación oficial, puesto que este diagrama es lo suficientemente claro:

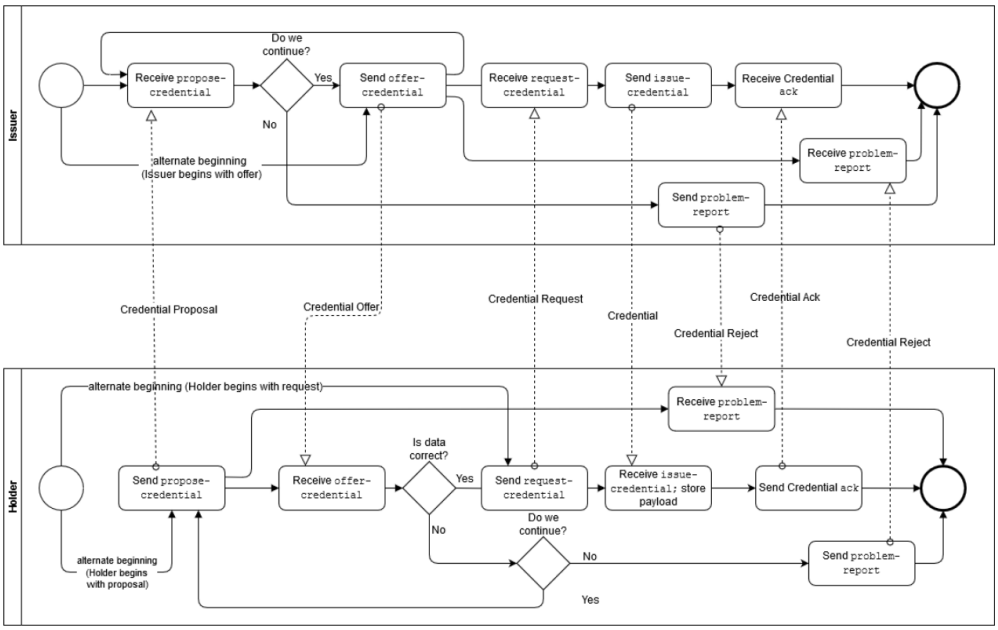

Figura 7: Protocolo Aries 0036 [64]

Protocolo de presentación de credencial (**RFC 0037** [65]). En este protocolo se detallan los pasos necesarios para que un **probador** (titular) presente una credencial verificable a un **verificador**. El verificador envía una solicitud de presentación de una credencial determinada en base a un esquema de credencial al probador, el cual deberá decidir si acepta o no presentar dicha credencial. En el caso de que sí acepte, el probador deberá construir la presentación y enviársela al verificador, el cual deberá verificarla. Existe un paso opcional previo a la solicitud de presentación que permite a un probador iniciar el proceso, de forma que este envía una propuesta de presentación al verificador, el cual podrá formular una solicitud en base a esta propuesta. A continuación, se muestra un diagrama tomado de la publicación oficial donde se muestran los pasos mencionados y con los estados a nivel técnico por los que el proceso va pasando:

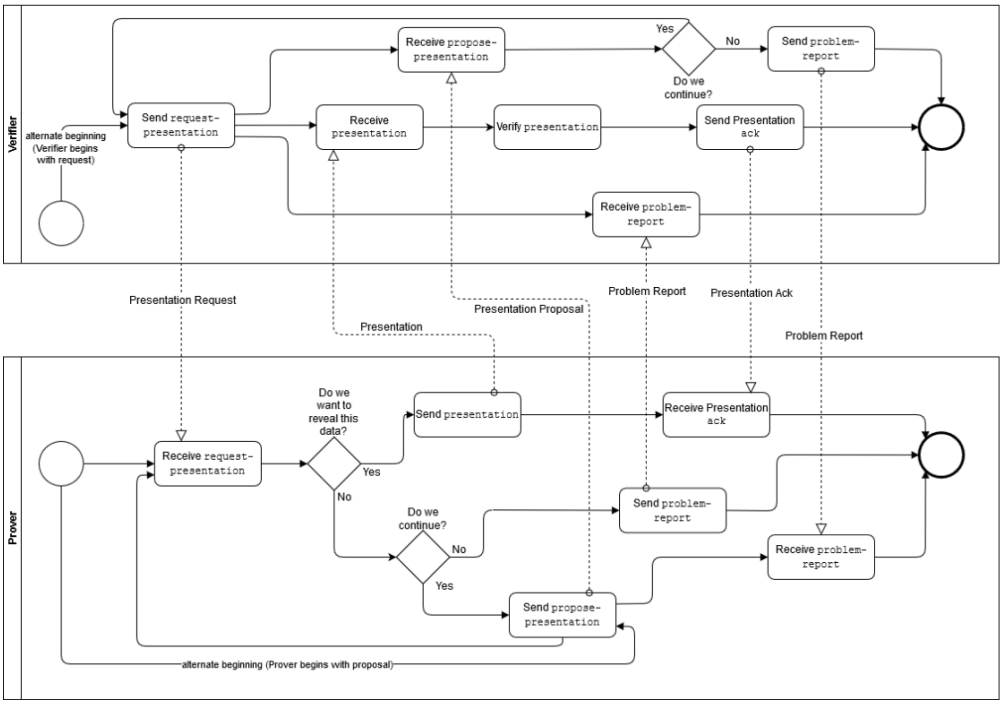

Figura 8: Protocolo Aries 0037 [65]

2.3.3.2 Arquitectura de un agente Aries

Hyperledger Aries proporciona herramientas para desarrollar agentes digitales de acuerdo a una arquitectura formada por dos componentes; un controlador (*controller*) y un componente denominado *framework*.

El **framework** es el componente que se encarga de abstraer al desarrollador parte de la complejidad de los protocolos (Aries y DIDComm) y de otras operaciones como la gestión de la cartera digital o la interacción con la interfaz del VDR correspondiente. El desarrollador no tiene por qué desarrollar este componente, sino que puede integrarlo en su solución software a través de las diferentes opciones disponibles, las cuales se detallarán en el próximo apartado.

El **controlador** es el componente que se encarga de interaccionar con el framework y el que dicta el funcionamiento de este en el contexto del ámbito de negocio relativo a la solución software. Este componente debe ser diseñado e implementado por el desarrollador, y debe estar correctamente integrado con el framework, de forma que exista una comunicación bidireccional con este.

Un framework recibirá eventos del exterior y, en función de su configuración, decidirá qué hacer con estos. Podrá descartarlos, reaccionar sin requerir la interacción del controlador o notificar al controlador para que decida qué hacer. Si la configuración dicta que el framework debe notificar al controlador, lo hará de forma asíncrona, de manera que el framework seguirá gestionando otros eventos mientras que no reciba una respuesta del controlador. Una vez el framework recibe una respuesta del controlador, se encargará de actuar conforme a dicha respuesta, al propio protocolo y a su configuración. El controlador por tanto deberá tener un canal entrante para recibir eventos del framework y un canal saliente para enviar solicitudes a este, resultando así en una comunicación bidireccional.

A continuación, se muestra la arquitectura de un agente Aries, donde se puede apreciar que el controlador contiene la lógica de negocio de la aplicación y que dispone de los dos canales mencionados. El framework es el componente denominado "*Aries Cloud Agent – Python*", cuyo título hace referencia al framework con el mismo nombre, que se detallará en el próximo apartado. Nótese que la comunicación saliente del controlador llega a un módulo "REST API" del framework; el mecanismo de este módulo varía entre los diferentes framework.

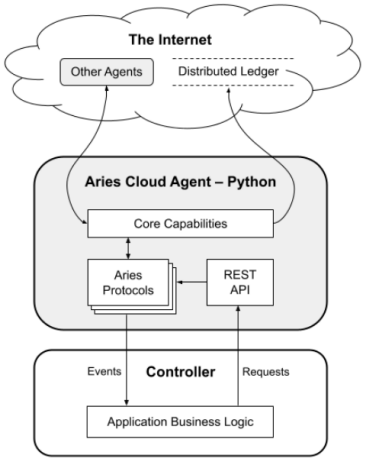

Figura 9: Arquitectura Agente Aries (ACA-Py) [66]

Un ejemplo de interacción podría ser el caso de un agente con rol de titular que recibe una oferta de expedición de credencial de un emisor. El framework, por tanto, recibirá dicha oferta y enviará un evento al controlador con los detalles de la oferta, de acuerdo con sus parámetros configurados. El controlador recibirá dicho evento y avisará al propio usuario a través de, por ejemplo, una notificación en su teléfono móvil. El usuario revisará los detalles y aceptará a través de algún botón en la aplicación móvil del agente digital. Tras esto, el controlador solicitará al framework que acepte dicha oferta, enviando así una solicitud de emisión de credencial al emisor, de acuerdo con el protocolo 0036 visto anteriormente. En este ejemplo se puede apreciar como el controlador es la propia aplicación móvil.

2.3.3.3 Los frameworks de Hyperledger Aries

Como se ha mencionado, existen diferentes alternativas a la hora de elegir un framework de Hyperledger Aries. A continuación, se presentan los cinco framework más relevantes en la actualidad [34]:

ACA-Py (*aries-cloudagent-python*) [66] es un framework escrito en Python enfocado al desarrollo de aplicaciones de servidor que integra una REST API que permite al controlador interaccionar con este a través de llamadas HTTP, posibilitando así el desarrollo de controladores con cualquier lenguaje de programación. Actualmente soporta por completo AIP 1.0 y, parcialmente, AIP 2.0. Soporta tanto los formatos de credencial AnonCreds como JSON-LD, e incluye un DIDResolver universal que permite a los agentes desarrollados con este framework ser agnósticos del VDR utilizado. En la Figura 9 se mostró la arquitectura de un agente ACA-Py.

AF-.NET (*aries-framework-dotnet*) [67] es un framework implementado con .NET enfocado al desarrollo de controladores móviles y de servidor. En este caso, el framework se debe importar

en forma de biblioteca en el controlador, de forma que la interacción se produce a través de llamadas a la propia biblioteca. Esto implica que los controladores deben estar en un lenguaje de programación que permita importar bibliotecas .NET. Respecto al soporte con las diferentes versiones de AIP, únicamente soporta la versión 1.0. Y, por último, AF-.NET únicamente permite credenciales AnonCreds y el método *did:sov* (redes Hyperledger Indy como VDR).

AF-Go (*aries-framework-go*) [68] es un framework escrito en Go enfocado al desarrollo de aplicaciones móviles y de servidor que, al igual que ACA-Py, integra una REST API para la interacción desde el controlador, posibilitando el desarrollo de las aplicaciones con múltiples lenguajes de programación. Opuesto a AF-.NET, AF-Go soporta únicamente AIP 2.0 y las credenciales JSON-LD de la W3C. Por último, este framework define su propio método, denominado *did:orb* [69] y únicamente ofrece soporte para este, por lo que no se pueden emplear otros VDR, como Hyperledger Indy.

AFJ (*aries-framework-javascript*) [70] es un framework escrito en TypeScript enfocado al desarrollo de aplicaciones móviles y de servidor. Al igual que pasa con AF-.NET, el controlador debe importar este framework en forma de biblioteca, obligando al desarrollador utilizar JavaScript (o TypeScript) como lenguaje de programación en la implementación del controlador. Para las aplicaciones móviles se debe emplear React Native, mientras que para las aplicaciones de servidor se debe emplear Node.JS. Este framework solo soporta AIP 1.0 y redes Hyperledger Indy como VDR. Por último, soporta tanto credenciales JSON-LD como AnonCreds.

AVCX (*aries-vcx*) [71] es un framework escrito en Rust que permite desarrollar aplicaciones móviles y de servidor. Este framework se ha construido en forma de biblioteca con una interfaz accesible por lenguajes compatibles con C (como Java o C++), permitiendo así desarrollar el controlador con estos lenguajes. Únicamente soporta AIP 1.0, credenciales AnonCreds y redes Hyperledger Indy como VDR.

En el **ANEXO III: Tabla comparativa de los framework de Hyperledger Aries** se incluye una tabla comparativa que resume los detalles mencionados hasta ahora en este apartado.

Cuando se quiera elegir un framework para desarrollar un agente digital, se debe tener en cuenta con qué tipo de agentes va a interactuar para encontrar la opción que mayor compatibilidad ofrezca con estos. Hyperledger proporciona a través de GitHub un proyecto [72] que permite ejecutar una serie de *test* sobre los diferentes framework disponibles para producir unos resultados de interoperabilidad, detallando qué *test* se pasan y cuáles no. Adicionalmente, estos resultados los van publicando en una página web [73] para que sean fácilmente accesibles y rastreados a lo largo del tiempo por los desarrolladores.

A continuación se muestra como modo de ejemplo una captura de pantalla de unos resultados publicados el día 23 de enero de 2023, donde se prueba la interoperabilidad entre el framework ACA-Py y el framework AFJ [74]:

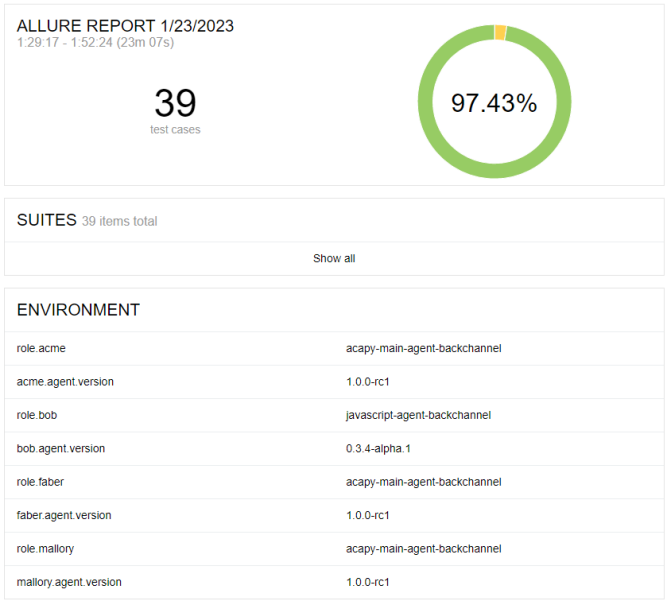

Figura 10: Resultados interoperabilidad entre ACA-Py y AFJ [74]

Se puede ver como la compatibilidad no es del 100%, y es que en este caso el *test* que fallaba estaba relacionado con la notificación por parte del emisor al titular sobre la revocación de una credencial. Nótese que aquí ACA-Py actuaba como emisor y como verificador, mientras que AFJ como titular[17].

2.3.3.3.1 Selección del framework para agentes de servidor

En este TFM se van a desarrollar varios agentes digitales de servidor cuyos controladores serán, sobre el papel, diferentes organismos públicos o empresas. Para estos casos, se ha seleccionado el framework **ACA-Py** sobre el resto de las opciones, puesto que es uno de los más utilizados en la lista, con mejor documentación y con mayor comunidad[18]. Además, incluye un soporte en cuanto a las versiones AIP, los formatos de credencial y métodos DID. Aunque este soporte no resulte relevante para el propio desarrollo de este TFM, puede serlo para una posible continuación de este proyecto. Adicionalmente, este framework obtiene, en general, unos buenos resultados de interoperabilidad respecto a otras opciones. Por último, la flexibilidad que ofrece en cuanto a la elección del lenguaje de programación para desarrollar los agentes ha tenido bastante peso a la hora de decantarse por este framework, ofreciendo así total libertad en la elección de este lenguaje.

Respecto a la arquitectura de un agente basado en ACA-Py vista en la Figura 9, el propio componente del framework se puede integrar de dos maneras diferentes. La primera opción sería descargando y configurando todas las dependencias en local y ejecutando ACA-Py como un proceso en el sistema operativo. La segunda opción es empleando virtualización basada en contenedores, de forma que todas las dependencias quedan empaquetadas en una imagen. Esta segunda opción es una opción mucho más portable y gestionable que la primera. Aunque uno puede crear manualmente dicha imagen, la comunidad de British Columbia ofrece directamente una imagen Docker [75] que nos evita el proceso.

[17] En estos *test* se emplean los roles de acme, bob, faber y mallory para describir a un emisor, un titular, un verificador y un titular malicioso, respectivamente.
[18] Este último dato se ha basado en el número de *issues* creados en los repositorios.

La configuración del framework se puede realizar al especificar un conjunto de parámetros en la ejecución de este, como a través de variables de entorno. En este TFM se emplea, principalmente, esta segunda opción al permitir así configurar con mayor facilidad el framework desde Docker.

En la configuración existen dos operaciones importantes: el abastecimiento y el arranque. La primera operación se ejecuta una única vez, mientras que la segunda se ejecuta cada vez que se arranquen los servicios. Esta separación de operaciones respecto a la configuración tiene sentido para los casos de producción. En este TFM, sin embargo, no se contempla esta fase de producción por lo que no se ha realizado esta distinción en la configuración.

Existen multitud de opciones de configuración, algunas de ellas se verán en detalle en los apartados de implementación de la PoC y del MVP. Estas opciones se agrupan según su función, teniendo grupos de opciones relacionadas con el VDR, con las interfaces de transporte de las conexiones, con la cartera o con la depuración, entre otros muchos grupos. En la versión 0.75 del framework [66] existen más de cien opciones diferentes. Con todo esto se puede observar que un framework ACA-Py se puede configurar con un gran nivel de personalización.

2.3.3.3.2 Selección del framework para el agente móvil

Como se ha mencionado anteriormente, un agente móvil necesita de un agente mediador además del propio agente móvil. Esto implica que se necesitan desarrollar dos controladores y configurar dos framework diferentes. En el caso del agente mediador, este hace uso de un framework de servidor, y su elección se basa en los mismos principios que un agente de servidor corriente, buscándose así la mayor interoperabilidad posible.

Para los propios agentes móviles, existen proyectos que facilitan el desarrollo de estos, como es el caso de *aries-mobile-agent-react-native* (también llamado *Aries Bifold*) [76] y de *aries-mobile-agent-xamarin* [77], el primero desarrollado sobre AFJ y el segundo sobre AF-.NET. Estos proyectos se encargan de ciertos detalles complejos, como el de interactuar con una red Hyperledger Indy.

Por otra parte, también existen multitud de agentes digitales (controlador y framework) ya desplegados en Google Play y Apple Store, como es el caso de Trinsic [78], Lissi (solución europea) [79], Esatus (solución europea) [80], BC Wallet (solución de la comunidad de British Columbia) [81] o Connect.Me (solución de la empresa Evernym) [82]. Todos estos integran sus propios mediadores.

Se han utilizado con éxito las aplicaciones Trinsic, Lissi y Esatus en gran parte del desarrollo de del MVP. Sin embargo, todas estas aplicaciones dejaron de funcionar con el sistema desarrollado en este TFM a pocas semanas de la entrega por, probablemente, problemas de compatibilidad[19] entre los framework en los que están desarrolladas estas aplicaciones (AF-.NET en la mayoría de los casos) con una de las herramientas utilizadas para el despliegue (ngrok), relacionados con el cifrado de los túneles TLS empleados por esta herramienta.

[19] Este problema fue encontrado en un repositorio ajeno a los mencionados hasta ahora, en un *issue* que se abrió donde se mencionaba que se usaba una de estas aplicaciones (https://github.com/bcgov/vc-authn-oidc/issues/135).

La decisión inicial fue la de emplear carteras digitales de terceros. Esta decisión tuvo sus inconvenientes, como tener que sufrir las inestabilidades de estas aplicaciones. Además, estas no ofrecían ciertas necesidades que el proyecto podría llegar a necesitar. Sin embargo, eran detalles que podrían pasarse por alto a cambio de mejorar la calidad del resto de componentes.

Ante el problema descrito, se tuvo que elegir si dedicar tiempo a arreglarlo para que volviesen a funcionar las aplicaciones o dedicar tiempo en desarrollar una aplicación propia que cubriese los defectos mencionados en el párrafo anterior. Se decidió, por tanto, la opción de un desarrollo propio. Sin embargo, el imprevisto tardío ha obligado a hacer uso de ACA-Py en lugar de un framework como AFJ para desarrollar a tiempo el controlador del agente móvil, el cual está desplegado en un servidor y se comunica con una aplicación móvil a través de protocolos convencionales (REST API), tal y como se verá en el capítulo del MVP. Esta solución presenta una deficiencia considerable; la cartera está fuera del entorno del dispositivo móvil y, por tanto, fuera de control del controlador del agente.

2.3.4 Alternativas al *stack* de identidad Hyperledger

Aunque se ha presentado con gran nivel de detalle el conjunto de tecnologías que forman el *stack* de identidad Hyperledger, todavía no se han mencionado las diferentes alternativas a estas tecnologías para implementar un ecosistema de identidad autosoberana, ni se han mencionado los motivos por los cuales se ha elegido esta opción frente al resto. Por tanto, en este apartado se van a presentar brevemente las alternativas más relevantes que el alumno ha encontrado y sus deficiencias notables frente a Hyperledger Ursa, Indy y Aries que han llevado a tomar la decisión de descartarlas.

La primera alternativa ya se ha mencionado anteriormente, esta es la propuesta europea **EBSI** (*European Blockchain Services Infrastructure*) concebida por la asociación europea **EBP** (*European Blockchain Partnership*) que, como recordatorio, es una asociación de los estados miembros de la UE y algunos del EEE. Esta propuesta tiene como propósito principal el de definir, desarrollar e integrar una infraestructura blockchain en las administraciones públicas y en los negocios. El proyecto clasifica los posibles casos de uso en tres familias: credenciales verificables, intercambio de datos confiables y trazabilidad. Sin embargo, actualmente únicamente han desarrollado dos casos de uso [83] (pertenecientes a la familia de las credenciales verificables): educación y seguridad social.

EBSI hace uso de credenciales JSON-LD [84] siguiendo la recomendación de la W3C, aunque no permiten firmas BBS+ que posibilitan el uso de revelación selectiva y ZKP. Y respecto al VDR, EBSI implementa su propia blockchain basada en Hyperledger Besu [37] y define su propio método DID (*did:ebsi*).

Se decidió no usar EBSI en el desarrollo de este TFM porque, a juicio del alumno, el proyecto aún está muy inmaduro, lo que podría llegar a limitar o impedir la realización de los objetivos marcados. Sin embargo, en el futuro esta opción debería ser, al menos, considerada por cualquier desarrollador que quiera implementar un sistema de SSI en el ámbito de la Unión Europea, puesto que es un proyecto apoyado directamente por la Comisión Europea [85], lo cual es un indicio de continuidad y de prosperidad del proyecto.

Otra opción que se llegó a considerar fue **TrustID**, un proyecto de código libre desarrollado por Telefónica y acogido por Hyperledger Labs [86]. Este proyecto tiene como objetivo la gestión de la identidad de proyectos basados en Hyperledger Fabric a través del uso de identificadores descentralizados, pero sin emplear las credenciales verificables. TrustID se compone de un contrato inteligente o, como se conoce en Hyperledger Fabric, un *chaincode* y de un conjunto de herramientas externas al propio canal (registro de Hyperledger Fabric) y que permiten interactuar con este.

El alumno, junto con el grupo de investigación del proyecto covID Card, realizó una serie de pruebas con este proyecto y encontró que esta solución introduce dos amenazas de gravedad elevada.

La primera amenaza que se observó era que las claves privadas se almacenaban sin cifrar en un fichero, de forma que cualquiera que consiguiese acceso al servidor donde se encuentra dicho fichero podría comprometer las claves.

La segunda amenaza que el grupo de investigación encontró venía dada por la propia naturaleza de Hyperledger Fabric y es que el usuario debe confiar sus claves privadas a los administradores de la red correspondiente. Esto implica que el sujeto deja de ser el controlador del identificador, delegando esta responsabilidad al propio servicio con el que está interactuando, de forma que este podría, por ejemplo, revocar el propio DID. El alumno considera que esta práctica no cumple con los principios de una identidad autosoberana, ya que la identidad del sujeto pasa a estar a merced del servicio con el que interactúa, al igual que pasa con la identidad centralizada actual. La única alternativa que se encontró para acercarse a una identidad autosoberana fue la de tener que desplegar un nodo de la red Fabric por cada usuario que accede a la red, lo cual se podría asemejar en el entorno de Ursa-Aries-Indy a que todos los agentes digitales incluyesen su propio nodo de la Indy, incluso en los teléfonos móviles.

Adicionalmente a los problemas comentados, se observó que el repositorio GitHub donde se aloja el proyecto se archivó el día 9 de noviembre de 2022 [87], unos días después del comienzo del proyecto (covID Card), lo que implica que el proyecto ha podido ser abandonado[20].

Otra alternativa al *stack* de identidad Hyperledger es el **Ecosistema Trinsic** [88], una propuesta de los desarrolladores de la aplicación móvil Trinsic mencionada anteriormente. En este caso, el proyecto sigue la recomendación de la W3C al usar credenciales con formato JSON-LD y admite la aplicación de la revelación selectiva y de ZKP. Además, hace uso de las listas de revocación detalladas por la W3C. Sobre el VDR, este ecosistema no usa un registro distribuido por defecto, aunque permite el uso de estos.

En este caso, esta opción no se consideró por, primero, no conocerla hasta una fase avanzada del proyecto y, segundo, por estar asociada directamente a una empresa privada y ser de pago.

Por último, se encontró una solución española en la plataforma de SSI que ofrece la empresa Gataca [89], pero el alumno no profundizó en esta alternativa por los mismos motivos por los que se descartó el Ecosistema Trinsic.

[20] El proyecto sigue archivado a mes febrero de 2023 (mes de publicación de este documento).

2.4 Tecnologías de desarrollo

En este apartado se presentarán lo más brevemente posible las diferentes tecnologías utilizadas para el desarrollo de la Prueba de Concepto y del Producto Viable Mínimo.

2.4.1 Tecnologías frontend

En la **PoC** no se ha desarrollado ningún cliente, sino que se ha empleado la herramienta **Postman** como forma de interactuar con los servidores desarrollados. Postman [90] es una plataforma que permite construir y consumir API (*Application Programming Interface*) de forma sencilla.

Los clientes que se han desarrollado en el **MVP** utilizan las tecnologías HTML, CSS y JavaScript. Adicionalmente, se ha empleado React como biblioteca principal de desarrollo, especialmente React DOM para las interfaces web y React Native para el caso de la aplicación móvil. Además, se ha empleado Postman como cliente para algunos casos concretos, como en el caso de agentes secundarios o en las primeras fases de desarrollo.

React [91] es una biblioteca creada por Facebook que opera sobre un DOM (*Document Object Model*) virtual en lugar de sobre el propio DOM original, de forma que se pueden producir cambios sin tener que reescribir el árbol HTML. Se basa en la creación e integración de componentes, donde un componente es un elemento acotado que forma parte de la página completa, que tiene su código HTML, CSS y JavaScript y que contiene un estado interno. Esta dinámica de componentes fomenta la aplicación de la filosofía DRY (*Don't Repeat Yourself*) del desarrollo software y, además, posibilita el poder aislar diferentes partes de una interfaz en diferentes módulos, facilitando así el diseño, el desarrollo y el mantenimiento. Es importante detallar que las interfaces web utilizan una biblioteca llamada *react* y otra llamada *react-dom*, donde la primera es el núcleo del funcionamiento de React y la segunda es la que permite aplicar React en interfaces web.

Se ha optado por React frente a otras alternativas como Angular o Vue.js por los siguientes motivos: el alumno tiene experiencia desarrollando con React, la comunidad es mucho más extensa respecto al resto de opciones y permite el desarrollo de aplicaciones móviles nativas a través de React Native.

React Native [92] es una biblioteca (*react-native*) que opera junto con la biblioteca *react* para crear aplicaciones móviles nativas haciendo uso de herramientas web (HTML, CSS y JavaScript). Sin embargo, presenta ciertas diferencias notables en comparación con el desarrollo web ya que, por poner un ejemplo, no se emplean elementos HTML convencionales como el contenedor * * o la etiqueta de texto *<p>*, sino otros, como *<View>* o *<Text>*.

En la aplicación móvil desarrollada se ha empleado también la herramienta **Expo** [93], la cual facilita el desarrollo con React Native y ofrece ciertas utilidades, como iconos o medidas relevantes (como la barra de estado de un móvil). Expo permite desarrollar fácilmente sobre un teléfono virtual, a través de Android Studio o Xcode, o sobre un teléfono físico a través de una aplicación disponible en Google Play y Apple Store.

Tanto para las interfaces web como para las aplicaciones móviles se ha empleado el uso del "*context*" y de los "*hooks*" que ofrece React para manejar el estado global de las aplicaciones.

Para manejar la navegación entre páginas en las interfaces web se ha empleado la biblioteca *react-router-dom* [94] y para la navegación entre pantallas de la aplicación móvil se ha empleado la biblioteca *react-navigation* [95].

En el caso de las interfaces web, se ha utilizado **Bootstrap** [96], una biblioteca que contiene HTML, CSS y JavaScript, y que ofrece una serie de componentes estilizados y con funcionamiento. Esto permite agilizar el proceso de implementación de la interfaz de usuario (UI).

Para la comunicación con el servidor, se ha empleado la biblioteca *axios* [97] como cliente para realizar peticiones HTTP y se ha empleado *Socket.IO* [98] (biblioteca *socket.io-client*) como herramienta para la comunicación en el sentido inverso, es decir, desde el servidor hacia el cliente. Esta comunicación bidireccional permite el envío de peticiones al servidor y la recepción de eventos.

En el caso de las interfaces web, se ha empleado un proxy (biblioteca *http-proxy-middleware* [99]), ya que React funciona sobre un servidor en la fase de desarrollo, de forma que todas las rutas relativas que se indiquen en el código irán encaminadas hacia el propio servidor de React, en lugar de hacia el servidor. En lugar de incluir rutas absolutas, obligando a tener que cambiarlas cada vez que se cambie de un entorno de desarrollo a uno de producción y al revés, se ha empleado este proxy para que reencamine todas las rutas relativas hacia el servidor.

Para concluir, se ha empleado la biblioteca *grid.js* [100] para la creación de tablas dinámicas con React, la biblioteca *react-select* [101] para la creación de menús de opciones complejos con React, la biblioteca *react-qr-code* [102] para la generación de códigos QR a partir de una URL y la biblioteca *expo-barcode-scanner* [103] (biblioteca ofrecida por Expo) para escanear códigos QR en aplicaciones React Native.

2.4.2 Tecnologías backend

El desarrollo de todos los servidores, tanto en la PoC como en el MVP, se ha realizado con Node.js y con el framework Express. Los servidores desarrollados son servidores API REST.

Node.js [104] es un entorno de ejecución para JavaScript que permite ejecutar código JavaScript fuera del entorno de un navegador. Esto posibilita el uso de JavaScript más allá del desarrollo de interfaces web, como es la creación de servidores o de *scripts*. Se ha elegido Node.js frente a otros lenguajes de programación, como Java, dada la experiencia del alumno en el desarrollo de servidores con Node.js y porque es una opción que proporciona un prototipado más rápido que otras opciones.

Express [105] es un framework para Node.js que se caracteriza por ser ligero, rápido y minimalista. Se ha elegido esta opción frente a otras, como Nest.js, puesto que se trata de la opción con mayor comunidad y puesto que el alumno tiene experiencia desarrollando con este. Adicionalmente, otros frameworks como Nest.js en el caso de Node.js o Spring en el caso de Java se caracterizan por seguir reglas más estrictas a cambio de ofrecer mayor funcionalidad con menor configuración necesaria. El problema que tienen estas soluciones es que a veces es difícil

implementar ciertas funcionalidades siguiendo sus reglas, por lo que se ha decidido usar un framework minimalista para evitar estas situaciones.

Como se ha mencionado anteriormente, el cliente y el servidor disponen de una comunicación bidireccional, donde la comunicación desde el servidor hacia el cliente se implementa a través de Socket.IO (biblioteca *socket.io* en el caso del servidor).

Como se verá en los capítulos de la Prueba de Concepto y del Producto Viable Mínimo, existirán varios componentes conectados en cadena, de forma que habrá servidores que interaccionen con otros servidores. Estas comunicaciones también deberán ser bidireccionales para permitir la recepción de eventos. En este caso no es necesario implementar ningún socket, ya que los eventos se recibirán a través de los llamados *webhooks*, que básicamente son puntos de acceso que funcionan con el método POST, de forma que otros servidores llaman a esta API para enviar información. No se ha implementado ningún sistema de mensajería como Kafka para tratar los eventos porque estos servidores siempre tienen un único emisor y receptor, que siempre son los mismos, y que siempre están disponibles (uno dependerá del otro).

En un servidor se ha requerido emplear una caché, la cual se ha implementado con la biblioteca *node-cache* [106], la cual permite crear cachés sencillas y gestionarlas fácilmente. Dado que únicamente se necesitaba integrar una caché sencilla, no se han utilizado otras opciones más potentes, como la integración de una basada en la base de datos Redis.

Y, por último, en un servidor se ha necesitado integrar unos temporizadores asociados a un identificador, por lo que se ha empleado la biblioteca *named-timers* [107] para facilitar el desarrollo.

2.4.3 Base de datos

En un agente del Producto Viable Mínimo se ha tenido que añadir una base de datos, por lo que en este apartado se mencionará rápidamente el tipo de base de datos elegido y la tecnología en concreto. La decisión del tipo de base de datos se dará en el apartado 4.2.1 (Arquitectura y tecnologías de desarrollo) ya que esta se ha tomado en base a los datos que va a almacenar y en base a los detalles de disponibilidad, consistencia y tolerancia al fallo.

Se trata de una base de datos relacional (**SQL**), por lo que no podrá garantizar el funcionamiento del sistema ante posibles fallos, de acuerdo con el teorema CAP.

El **teorema CAP** indica que una sistema distribuido no podrá garantizar simultáneamente la consistencia de los datos, la disponibilidad del sistema y tolerancia al particionado del sistema ante fallos en enlaces o en nodos. La razón es sencilla de entender ante un ejemplo, supongamos una base de datos con dos nodos sincronizados entre ellos. Si el enlace entre los nodos falla, las escrituras en los nodos producirán datos inconsistentes ante la falta de sincronización. Por lo tanto, se deberá decidir si dejar de dar el servicio (indisponibilidad) o aceptar la falta de sincronización (inconsistencia). Esta situación no se reproduce en sistemas no particionados, como una base de datos de un solo nodo.

En cuanto a la base de datos específica, se ha empleado **MySQL** [108], ya que se trata de la base de datos relacional de código abierto con mayor comunidad [108], [109]. Además, se ha utilizado

una versión superior a la 8.0.16, ya que a partir de esta versión no se ignora la restricción CHECK al crear las tablas [110].

Para interactuar con la base de datos desde Express, se ha utilizado la biblioteca *mysql2* [111], dado que es la biblioteca más popular (basándose en las descargas semanales y en el número de *issues*) para este propósito.

2.5 Tecnologías de despliegue

Aunque no se ha realizado un despliegue basado en la nube, sí que se ha realizado en local a través de tecnologías como Docker o ngrok y a través del lenguaje BASH.

BASH es una interfaz de usuario de línea de comandos del terminal de Unix que permite la creación de *scripts* para la automatización de la ejecución de comandos en el terminal. Se ha empleado para automatizar el despliegue del sistema completo, tanto en la PoC como en el MVP. Como parte de la automatización, se ha programado la creación y gestión automática de ficheros que contienen variables de entorno, de forma que otras herramientas, como Docker o Node.js, puedan leer estas variables y cambiar su comportamiento en función de estas.

Se han utilizado las herramientas cURL y jq en los *scripts* BASH como apoyo a la automatización. **CURL** [112] es una herramienta que proporciona una interfaz para la transferencia de datos en base a una URL y se ha empleado para realizar peticiones REST a diferentes servicios. Y **jq** [113] es una herramienta de procesado de estructuras de datos JSON, y se ha empleado para recuperar información dentro de estas estructuras.

Docker [114] es una plataforma de código abierto que permite crear y desplegar contenedores virtualizados ligeros, permitiendo aislar diferentes servicios. Cada contenedor incluye sus propias dependencias, de forma que estos son portables. Este aislamiento también proporciona una capa de seguridad ante situaciones donde existen varios contenedores desplegados sobre un mismo equipo físico. Docker se ha empleado para la virtualización de los diferentes servicios del sistema y las redes que los comunican, tanto en la PoC como en el MVP. Además, Docker ha permitido seguir una filosofía DRY hasta el extremo, ya que en algunos casos se han creado contenedores Docker para diferentes servicios en base a un código común y en base a deltas de código que definían las particularidades de cada uno de los servicios.

Como complemento a Docker, se ha utilizado **Docker Compose** [115] como herramienta que permite configurar múltiples contenedores Docker en ficheros YAML y ejecutarlos con un único comando. Docker Compose ha permitido al alumno especificar multitud de variables de entorno en los contenedores a partir de los ficheros generados por los *scripts* BASH de una forma muy sencilla, además de facilitar la configuración de todos los contenedores.

Por último, se ha empleado **Ngrok** [116]. Esta es una herramienta que permite exponer públicamente puntos de acceso de servicios funcionando en local, sin tener que configurar una IP estática en la red del hogar. Para ello, ngrok crea un túnel TLS entre el servicio local y un servicio desplegado por ngrok, accesible a través de una URL con formato "https://<nombre aleatorio>.ngrok.io". Si el servicio local ofrece una interfaz gráfica, entonces al acceder a esa URL se podrá visualizar, independientemente de la red desde donde se acceda. Esta herramienta

se ha utilizado para que la comunicación entre los agentes sea posible, de forma que se han expuesto todos los contenedores que incluían el servicio del framework Aries. Para evitar tener que instalar la dependencia ngrok, se ha utilizado una imagen docker [117].

Capítulo 3. PRUEBA DE CONCEPTO (POC)

En este capítulo se va a presentar la Prueba de Concepto que se ha implementado. Esta se ha realizado con el objetivo de probar el potencial de la identidad autosoberana y de las tecnologías seleccionadas, especialmente el *stack* de identidad Hyperledger. Adicionalmente, se ha aprovechado para crear un componente de controlador Aries genérico con intención de reutilizarlo en el MVP.

Este capítulo se plantea en tres apartados principales que reflejan las diferentes fases que se han seguido. Primero, se realizará un análisis del escenario empleado para así establecer los detalles concretos de este. Segundo, se detallará cómo se ha diseñado el sistema. Y, tercero, se comentará cómo se ha implementado el sistema teniendo en cuenta el escenario y el diseño presentados, detallando algunas de las decisiones de desarrollo que se han tomado en el proceso.

3.1 Análisis

Para poder diseñar e implementar un sistema de SSI, por muy sencillo que sea, es necesario definir cierta lógica de negocio para poder así establecer los roles (emisor, verificador, titular) de los agentes digitales a crear y para poder establecer el contenido de las credenciales verificables que se van a intercambiar.

Es por eso por lo que se ha tenido que definir un escenario de negocio simple para esta Prueba de Concepto, aprovechando la necesidad para acercarse ligeramente al escenario de validación empleado en el Producto Viable Mínimo. Por tanto, esta PoC va a tratar de un sistema de identidad autosoberana rudimentario en el ámbito sanitario europeo.

El escenario plantea a un ciudadano español que necesita viajar desde España hacia otro país de la Unión Europea a través de un avión. Para esto, el ciudadano deberá solicitar a la FNMT una credencial verificable de identidad, recibir una dosis de una vacuna COVID-19 y solicitar una credencial verificable de vacunación y, por último, acceder al aeropuerto y presentar dichas credenciales.

Para analizar el escenario en detalle, primero se van a comentar cuáles son las credenciales verificables intercambiadas y el contenido de estas. Luego, se presentarán todos los agentes que intervienen. Y, por último, se establecerán todos los detalles del propio escenario.

3.1.1 Esquemas de credenciales

Para este escenario se van a emplear dos credenciales verificables, una de identidad y otra de vacunación. Estas están simplificadas respecto a cómo serían en la realidad y para su definición no se ha seguido ningún esquema oficial. La razón de esto es porque en la PoC el escenario es secundario, al contrario que pasa con el MVP, donde sí se han seguido esquemas oficiales.

Los esquemas de las credenciales se van a presentar a continuación en formato tabla, donde el título y los atributos de cada esquema están en inglés y en formato *snake case* (en lugar de espacios se utiliza la barra baja "_") puesto que son los que se han utilizado en el propio esquema.

Credencial de identidad: **covID_PoC_Identity**.

Atributo	Formato	Detalles
full_name	String	Nombre y apellidos del ciudadano
birthday	String	Fecha de nacimiento en formato ISO 8601 [118]
birthday_epoch	Number	Fecha de nacimiento en tiempo Unix[21]. Fechas de nacimiento anteriores serán ajustadas a la marca temporal 0 (comienzo del tiempo Unix).
address	String	Dirección del domicilio donde reside el ciudadano.
dni_number	String	Número del DNI del ciudadano.
photo_url	String (URL)	URL de la foto del DNI del ciudadano.

Tabla 1: Esquema de credencial de identidad (PoC)

Credencial de vacunación: **covID_PoC_Vaccination**.

Atributo	Formato	Detalles
agent	String	Identifica la enfermedad o el agente objetivo (COVID-19).
expiration	Number	Fecha (tiempo Unix) de expiración de la credencial.
n_dose	Number	Número de dosis total que se han administrado.
date_last_dose	Number	Fecha (tiempo Unix) en la cual se administró la última dosis.
name_last_dose	String	Nombre de la vacuna que se administró en la última dosis.
country_last_dosis	String	País en el cual se administró la última dosis. Formato en estándar ISO-3166 de dos letras [119].

Tabla 2: Esquema de credencial de vacunación COVID-19 (PoC)

Algunas fechas se han especificado en tiempo Unix ya que se pretende aplicar ZKP sobre estas. Esto es porque, como ya se dijo, las credenciales AnonCreds únicamente admiten predicados ZKP sobre atributos numéricos.

3.1.2 Agentes

En este escenario intervienen los siguientes cuatro actores, cada uno con su propio agente digital:

1. **Ciudadano** español. Controlará un agente de servidor, con el fin de simplificar la PoC.
2. **FNMT** (*Fábrica Nacional de Moneda y Timbre*), como la autoridad última encargada de expedir las credenciales verificables de identidad. El agente será de tipo servidor y será controlado por un trabajador autorizado por la FNMT.
3. **ECDC** (*European Centre for Disease Prevention and Control*), como la autoridad última encargada de expedir las credenciales de vacunación COVID-19. El agente será de tipo servidor y será controlado por un enfermero autorizado por la ECDC.
4. **Aena**, como la autoridad encargada de verificar las credenciales de identidad y de COVID-19 dentro de un aeropuerto español, con el objetivo de identificar y de regular quién puede volar y quién no. El agente será de tipo servidor y será controlado por un trabajador autorizado por Aena.

[21] Un tiempo Unix se entiende como el número de segundos desde las 00:00 del día 01-01-1970 hasta el instante determinado.

Es importante destacar que en la realidad algunos de estos agentes no serían los responsables de las acciones especificadas, pero se ha establecido así para simplificar el escenario lo máximo posible. Téngase en cuenta esto también para el próximo apartado.

3.1.3 Escenario

El escenario se puede describir en tres etapas bien diferenciadas.

En la **primera etapa**, el ciudadano español se presenta en un centro autorizado por la FNMT para expedir credenciales digitales de identidad, con el fin de obtener dicha credencial. Aunque el mecanismo es similar al actual de cara al ciudadano, internamente se trata de una infraestructura descentralizada (DPKI), donde el centro únicamente se encarga de expedir la credencial. Se puede suponer que el trabajador del centro autorizado por la FNMT tiene acceso al propio agente de la FNMT, por lo que la credencial irá emitida a nombre de esta entidad.

Una vez el ciudadano se presenta en un puesto del centro, el trabajador autorizado le ofrecerá un punto de acceso basado en un código QR, el cual representa una invitación para establecer una conexión segura basada en los protocolos Aries y DIDComm entre el agente de la FNMT y el agente del ciudadano.

Una vez establecida la conexión entre los dos agentes, el trabajador le manda una oferta de credencial al ciudadano y el ciudadano la acepta. Tras esto, el ciudadano ya tiene en su cartera la credencial verificable de identidad.

A continuación, se muestra un diagrama de secuencia de elaboración propia de esta etapa:

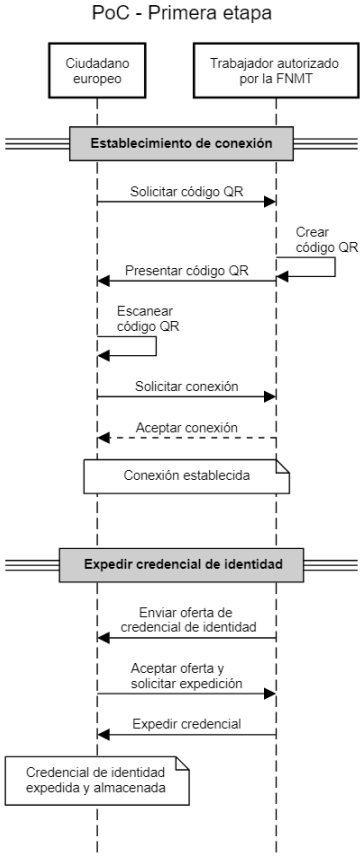

Figura 11: Diagrama de secuencia de la etapa 1 de la PoC

En la **segunda etapa**, el ciudadano se presenta en un centro de salud con el fin de obtener la credencial de vacunación COVID-19. Para ello, el enfermero le presenta un punto de acceso basado en un código QR con el fin de establecer una conexión segura entre el agente de la ECDC y el agente del ciudadano.

Una vez establecida la conexión, el enfermero, a través del agente de la ECDC, le manda al ciudadano una petición de presentación de la credencial de identidad, con el fin de identificarle. El ciudadano aceptará y le mandará dicha presentación.

Tras recibir y verificar la identidad del ciudadano, el enfermero, a través del agente de la ECDC, le manda una oferta de credencial COVID al ciudadano y el ciudadano la acepta. Tras esto, el ciudadano ya tiene en su cartera el certificado de vacunación COVID.

A continuación, se muestra un diagrama de secuencia de elaboración propia de esta etapa:

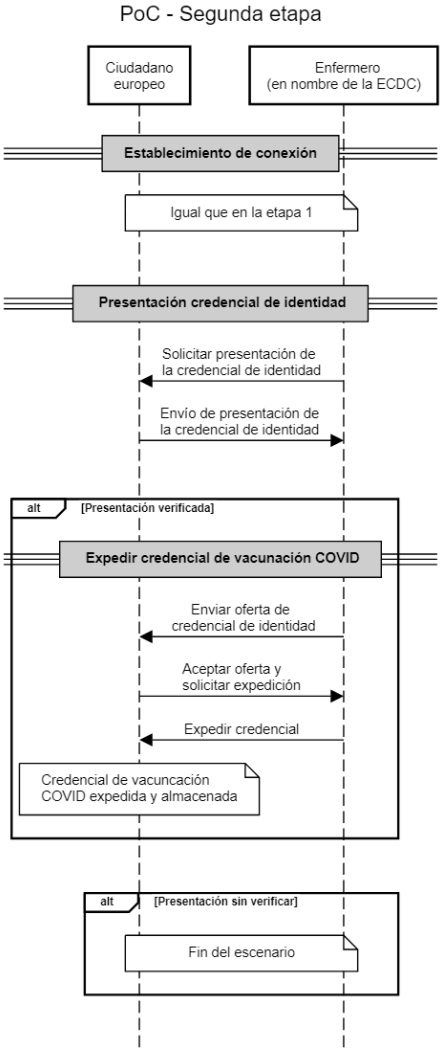

Figura 12: Diagrama de secuencia de la etapa 2 de la PoC

La presentación de identidad requerirá desvelar los atributos "*full_name*", "*DNI_number*" y "*photo_url*", los cuales permiten identificar al ciudadano. Además, requerirá un predicado ZKP que pruebe que el ciudadano es mayor de edad (*birthday_epoch* ≤ fecha actual – 18 años), de forma que no necesita de un tutor legal que gestione sus credenciales. Este sería un ejemplo donde

el sujeto y el titular son diferentes personas. Para simplificar el escenario, se supone que el escenario acabaría aquí si el ciudadano fuese menor de edad.

En la **tercera etapa**, el ciudadano se dirige a un aeropuerto español con el objetivo de volar a otro país dentro de la Unión Europea. Una vez ahí, un trabajador autorizado por Aena solicitará al ciudadano que, primero, le presente la credencial de identidad para poder identificarlo y que, segundo, presente la credencial de vacunación COVID-19 para poder valorar si cumple las condiciones sanitarias necesarias para viajar.

Para ello, el trabajador presentará un punto de acceso basado en un código QR con el fin de establecer una conexión entre el agente de Aena y el agente del ciudadano. Una vez la conexión se ha establecido, el trabajador enviará una petición de presentación de la credencial de identidad al ciudadano y este deberá aceptarla. Y, tras esto, el trabajador le enviará una petición de presentación de la credencial de vacunación, que también deberá aceptar. Con el contenido de la presentación de la credencial de vacunación, el trabajador de Aena verificará si cumple o no los requisitos sanitarios y, en función de esto, le autorizará o no para abordar el avión

A continuación, se muestra un diagrama de secuencia de elaboración propia de esta etapa:

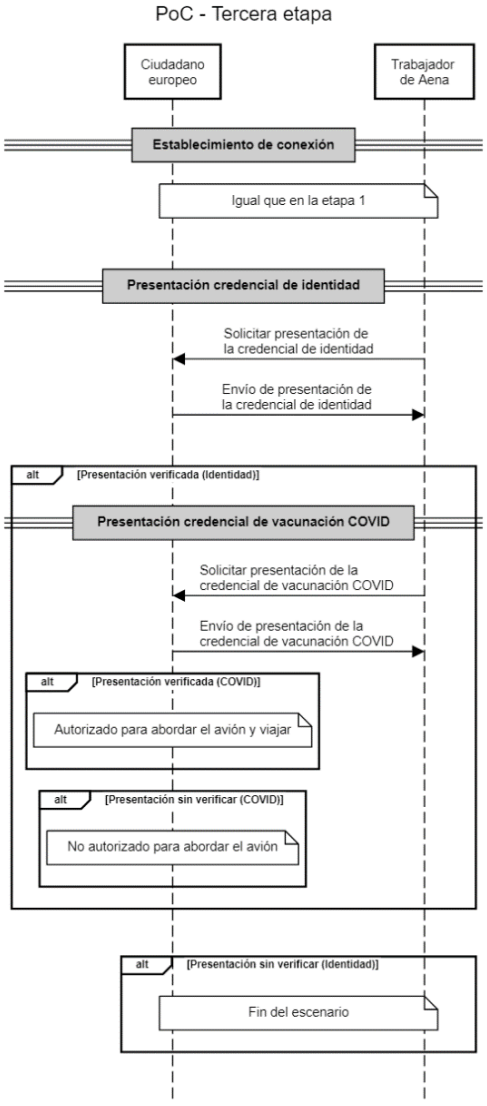

Figura 13: Diagrama de secuencia de la etapa 3 de la PoC

Las presentaciones solicitarán los siguientes detalles:

- Para la credencial de identidad, se solicitará desvelar los campos "*full_name*", "*dni_number*" y "*photo_url*", con el fin de identificar adecuadamente al ciudadano.
- Para la credencial de vacunación COVID, se solicitará desvelar el campo "" con el fin de verificar que se trata de una credencial de vacunación COVID-19 y no de otro agente. Además, se le solicitarán los predicados ZKP (*date_last_dose* ≤ 270), (*n_dose* ≥ 2) y (*expiration* ≥ fecha actual) para así comprobar que se le administró al menos una segunda dosis hace menos de 270 días y que la credencial no ha expirado.

Para que resulte apto para volar debe cumplir todos los predicados ZKP y el campo "*agent*" revelado debe tener el valor "COVID-19".

En el ANEXO IV: Diagramas de secuencia detallados se incluyen varios diagramas de secuencia de las tres etapas con diferentes niveles de detalle, con el objetivo de mostrar la complejidad interna.

Como se puede intuir, el ciudadano actuará únicamente como titular, FNMT como emisor de la credencial de identidad, ECDC como verificador de la credencial de vacunación y emisor de la credencial de identidad y AENA como verificador de ambas credenciales.

Como se verá en los próximos apartados, se ha decidido no implementar interfaces gráficas para esta PoC, ahorrando así esfuerzo y tiempo para luego dedicárselo al MVP. Esto implica que las invitaciones de conexión se gestionarán en la práctica directamente a través de las URL en lugar de los códigos QR generados a partir de estas. Aun así, se ha desarrollado la funcionalidad que permite generar estos códigos.

En todas las presentaciones se verificará también si las credenciales han sido o no revocadas, a través de las pruebas ZKP de no revocación.

Como se puede observar, el diseño de los esquemas de las credenciales, la elección de agentes y los detalles del escenario se han pensado para que se puedan probar todas las capacidades que ofrecen las credenciales AnonCreds. Esto es, revelación selectiva, predicados ZKP, pruebas de no revocación e, indirectamente, el uso de ZKP para la no revelación de las firmas digitales del emisor y la no revelación de un identificador único del titular. Adicionalmente, se están usando todos los conceptos de identidad autosoberana, haciendo uso de agentes que gestionan canales seguros basados en DIDComm y credenciales verificables, y que interaccionan con un VDR basado, en este caso, en Hyperledger Indy. Por tanto, este escenario cumple los requisitos para que se pueda diseñar e implementar una PoC de acuerdo con los objetivos marcados para esta.

3.2 Diseño

El sistema SSI a implementar sobre el escenario anterior se compone de cuatro agentes de servidor y necesita acceso a un VDR y a un servidor Tails. Y, como recordatorio, cada uno de los agentes Aries se compone de un controlador y de un framework.

Puesto que el objetivo de la Prueba de Concepto es probar las tecnologías Hyperledger y familiarizarse con los conceptos de SSI, se ha descartado el diseño y la implementación de unas interfaces gráficas.

3.2.1 Arquitectura

A continuación se muestra la **arquitectura general simplificada** del sistema mediante una figura de elaboración propia, donde aparecen los cuatro agentes y el VDR. Nótese que las líneas más gruesas son comunicaciones seguras basadas en DIDComm.

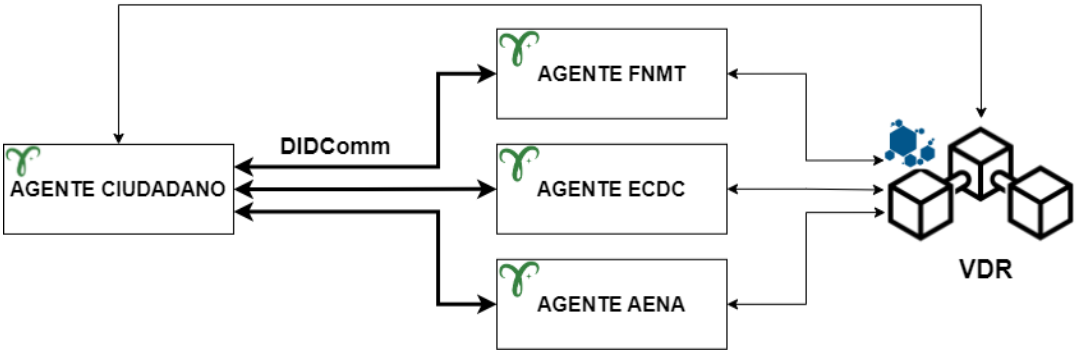

Figura 14: Arquitectura simplificada de la PoC

Los cuatro agentes son de tipo servidor o empresarial y se componen de un controlador y un framework. Ambos componentes tienen una base de configuración y desarrollo muy similares entre todos los agentes, con ciertas diferencias.

En el caso del framework, existirán diferencias de configuración de puertos, puntos de acceso internos o de opciones que definan el rol que va a tomar el agente, pero el resto de la configuración será igual, como es el caso de las opciones relacionadas con la cartera, con la generación de registros (*logs*) o con puntos de acceso externos (VDR y servidor Tails).

Y en el caso del controlador, existirá una lógica relacionada con el funcionamiento de Hyperledger Aries que no sea lógica de negocio que es interesante abstraer de cara al resto de componentes. De esta forma, se puede aislar en un único componente todo este funcionamiento y ofrecer un servidor API REST (con un canal para recibir eventos de este) que permita interactuar con el framework con un conocimiento menor de Hyperledger Aries.

En el caso del framework, esta similitud se maneja en el despliegue, tal y como se verá en el apartado correspondiente.

Y en el caso del controlador, se va a implementar un componente de controlador genérico que se ha denominado "*general API controller*" (**gAp**), que es un módulo que recibe peticiones con pocos parámetros del cliente y las transforma para mandárselas al framework. De igual forma, los eventos se filtran y se simplifican en gran medida para que al cliente le lleguen los detalles más básicos. Este componente ha sido reutilizado y refinado en el MVP.

Sin embargo, para esta PoC se ha añadido al componente gAp la lógica de negocio necesaria para cumplir con el escenario definido. Se ha tomado esta decisión en lugar de la decisión de crear otros componentes específicos para esto por la razón de que la lógica de negocio en la PoC es mínima, y no merecía la pena separarla en este caso.

Por tanto, los componentes de cada uno de los agentes son exactamente los mismos. A continuación, se muestra la **arquitectura de un agente** cualquiera (figura de elaboración propia), donde se puede ver que se han agrupado los componentes que forman el controlador, el frontend y el backend. Nótese que el componente con el fondo azul es el controlador general que se ha comentado anteriormente. Y nótese también que el socket tiene dirección unidireccional desde el servidor hacia el cliente, ya que es el canal por donde circulan los eventos.

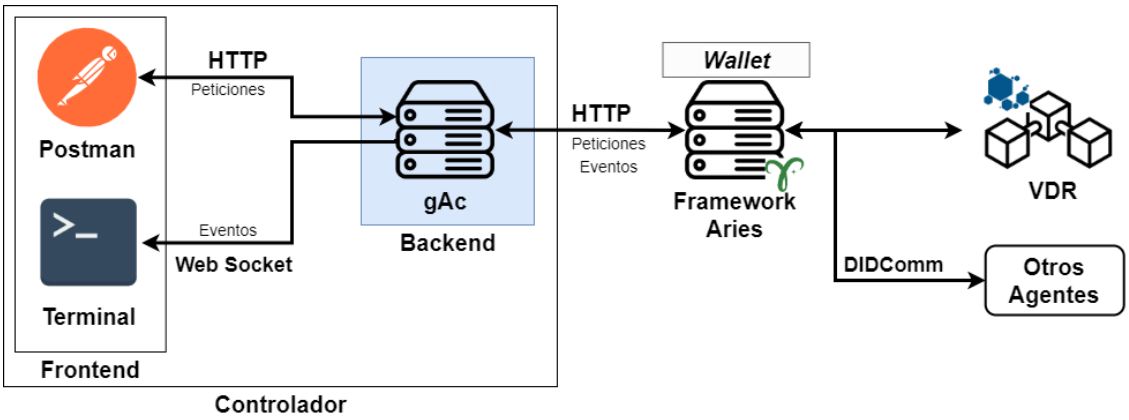

Figura 15: Arquitectura de un agente (PoC)

El framework es el componente que contiene la cartera digital y el que se encarga de interactuar con ella en base a su configuración y a las órdenes del controlador.

El frontend se compone de Postman como cliente que realiza peticiones HTTP y de un servidor mínimo cuyo único propósito es mostrar por terminal los eventos recibidos. Este servidor no se ha incluido como tal en la figura anterior porque es tan sencillo que resulta irrelevante.

En los diagramas anteriores se ha omitido el servidor Tails dada su baja relevancia respecto al resto de componentes. Sin embargo, en la implementación este componente está presente y es accesible por todos los agentes.

3.2.2 Tecnologías de desarrollo

En los diagramas anteriores se puede observar un icono verde y un icono azul. El icono verde implica que en ese componente se está empleando Hyperledger Aries, y el icono azul indica que se está empleando Hyperledger Indy.

Por tanto, el VDR es un registro distribuido basado en Hyperledger Indy. En el caso de la PoC, se ha desplegado un registro propio a través del proyecto VON, de forma que únicamente está disponible en local. Así, nos evitamos tener que tratar con el registro de DID en redes públicas y con la aceptación de los TAA.

El servidor Tails se ha desplegado en local a través del proyecto Indy Tails Server.

El framework se ha desplegado y configurado a través de la imagen Docker ofrecida por la comunidad de British Columbia. Hay que recordar que el framework elegido es ACA-Py.

El backend se ha implementado con Node.js y Express, junto con el resto de las tecnologías especificadas en el Estado del Arte. Y, como ya se ha dicho, el frontend incluye un servidor

mínimo para mostrar los eventos recibidos por un terminal y de una colección Postman para realizar peticiones al servidor.

Adicionalmente, se han empleado las tecnologías descritas en el apartado de despliegue del Estado del Arte. Esto quiere decir que todos los componentes (salvo el frontend) de la arquitectura mostrada anteriormente están contenidos en sus propios contenedores Docker. Además, todos los frameworks están expuestos públicamente a través de los túneles ngrok.

3.3 Implementación

Como ya se ha mencionado, todos los agentes comparten los mismos componentes. Además, todos comparten un mismo controlador general con ligeros cambios según el agente y una configuración similar entre frameworks. Por tanto, gracias a las variables de entorno, BASH y Docker, se ha creado un único servidor gAp y una única configuración base para todos los agentes.

El desarrollo de la PoC se ha realizado manteniendo un control de las versiones con git y publicando el código a GitHub en un repositorio privado.

3.3.1 Despliegue

El objetivo del despliegue del sistema de identidad autosoberana que cubre el escenario de la Prueba de Concepto es conseguir una **automatización** total de las operaciones de construcción, arranque, parada o eliminación, entre otras, de este sistema. Esta automatización permite al usuario interaccionar con la PoC sin tener que conocer la arquitectura ni los detalles técnicos de operación de los componentes. Un objetivo adicional al anterior es conseguir la máxima **virtualización** posible a través de contenedores para aprovechar así las ventajas que ofrecen estos: aislamiento y seguridad, instalación menos dependencias, interconexión de servicios que usan los mismos puertos, etc.

Este despliegue se basa en la ejecución de un *script* escrito en BASH llamado "**manage**", el cual crea un fichero llamado "**.env**" que contiene todas las variables de entorno que van a consumir los servicios Docker y otras variables necesarias para su configuración. Tras construir este fichero, ejecuta los ficheros docker-compose, los cuales contienen todos los servicios del sistema, redes virtualizadas, configuración de puertos y volúmenes necesarios.

Para alcanzar este objetivo, el *script* hace uso de los proyectos VON e INDY TAILS para desplegar en local una red Hyperledger Indy y un servidor Tails. Adicionalmente, define y crea los túneles ngrok necesarios para exponer los puntos de acceso de los frameworks. Mientras que en el MVP los túneles ngrok se definen en servicios Docker, evitando tener que instalar la dependencia en local, en la PoC estos túneles se definen en local a través de un fichero llamado "ngrok.yaml", por lo que para esta implementación es necesario instalar ngrok en local. Este fichero se crea automáticamente en el despliegue.

3.3.1.1 Docker

Docker se basa en la creación de contenedores, donde cada uno de los contenedores se crea a partir de una imagen. Estas imágenes pueden ser descargadas de repositorios, como Docker Hub,

o pueden construirse a partir unos ficheros llamados “Dockerfile”, los cuales contienen un conjunto de instrucciones que se aplican sobre una imagen base.

En esta Prueba de Concepto se descarga una imagen del repositorio Docker Hub publicada por la comunidad de British Columbia, llamada “*aries-cloudagent*”, que contiene las instrucciones necesarias para lanzar el servicio ACA-Py. Esta imagen se utiliza para crear los contenedores de los frameworks, existiendo uno por cada agente.

Y para la creación de las imágenes de los controladores (gAp), se ha creado un fichero Dockerfile con las siguientes instrucciones:

```
FROM node:16.7.0-alpine
ENV PORT 5000
RUN mkdir /api
WORKDIR /api
VOLUME /api
EXPOSE 5000
CMD [”npm”, “start”]
```

Fragmento de código 2: Dockerfile gAp PoC

La primera instrucción indica que la imagen va a partir de la imagen llamada “node” [120], la cual es una imagen que contiene todas las dependencias necesarias para ejecutar Node.js. Nótese que la versión de la imagen indica que usa Alpine como imagen base [121], que es una de las distribuciones Linux más ligeras que existen, pesando menos de 5MB.

Nótese que existe una carpeta en el proyecto PoC que contiene el servidor Node.js a ejecutar. En este caso, se ha tomado la decisión de usar la instrucción “VOLUME” en lugar de “COPY”, de forma que los cambios realizados en el anfitrión se reflejan en el contenedor sin tener que reiniciarlo. Esto tiene dos implicaciones: la primera es que permite desarrollar y comprobar los cambios sin tener que redesplegar el sistema; y la segunda es que obliga al usuario a instalar node y npm en el anfitrión, ya que se deben instalar las dependencias del proyecto Node.js. En producción se cambiaría a la instrucción “COPY” para evitar la segunda implicación, ya que la primera no supone ninguna ventaja para el usuario final.

El puerto 5000 es el que usa Node.js para desplegar el servidor dentro del contenedor.

En la PoC se ha decidido separar los servicios en dos ficheros docker-compose diferentes para facilitar el desarrollo, uno llamado “*docker-compose.frameworks.yml*” que contiene los frameworks de los agentes y otro llamado “*docker-compose.controllers.yml*” que contiene los controladores gAp.

Al contrario que pasa en el MVP, todos los servicios comparten una misma red llamada “main”, de forma que se facilita y agiliza el tiempo de desarrollo de esta PoC.

3.3.1.1.1 Docker Compose - Frameworks

En este fichero todos los servicios parten de la imagen “*aries-cloudagent*” y cuyos nombres son: “*acapy-TEST*”, “*acapy-ECDC*”, “*acapy-AENA*” y “*acapy-FNMT*”. El primero es el framework del agente del ciudadano europeo y el resto son los frameworks de los agentes indicados en el nombre.

Todos los servicios siguen la misma estructura y contienen las mismas declaraciones. A continuación, se muestra como ejemplo el caso del servicio *acapy-ECDC*:

```
  acapy-ECDC:
    image: bcgovimages/aries-cloudagent:py36-1.16-1_0.7.2
    container_name: ${ECDC_DOCKER_LABEL}
    hostname: ${ECDC_HOSTNAME}
    labels:
      name: ${ECDC_DOCKER_LABEL}
      description: "Aries Cloud Agent Python Container for the ECDC"
    environment:
      …
    entrypoint: /bin/bash
    command: [
      "-c",
      "curl—silent -d … -X POST ${LEDGER_URL}/register; \
      aca-py start …
    ]
    volumes:
      - ${PWD}/logs:/logs
    ports:
      - "${ECDC_AGENT_PORT}:${ECDC_AGENT_PORT}"
      - "${ECDC_ADMIN_PORT}:${ECDC_ADMIN_PORT}"
    networks:
      - main
```

Fragmento de código 3: Servicio del framework ACA-Py de la ECDC (PoC)

Nótese que se han omitido con puntos suspensivos las variables de entorno y parte de los comandos para que el fragmento sea más legible. Tanto las variables de entorno como los argumentos del comando "aca-py start" son las opciones que permiten definir la configuración del framework. Estas opciones se verán en el apartado 3.3.2 (Configuración framework Aries).

Se puede ver como casi no se han definido valores en el propio servicio, sino que estos se recuperan de las variables de entorno contenidas en el fichero ".env". De esta forma, se pueden definir estos valores dinámicamente en el despliegue en función de diferentes circunstancias, como de los argumentos especificados al ejecutar el *script* de despliegue "manage".

Los puertos de los servicios ejecutados dentro del contendor deberán ser accesibles fuera de este. Para ello, se crea una asociación entre estos puertos y puertos libres del anfitrión. De esta forma, cuando dentro del anfitrión se accede al puerto indicado, Docker expone el servicio del contenedor. En todos los frameworks ACA-Py se deben asociar dos puertos: el del propio servicio y el del administrador. El servicio del administrador provee de una interfaz REST que permiten administrar al agente e indicarle las operaciones que se quiere que ejecute. Este servicio es la API REST que ofrece ACA-Py y que evita al desarrollador tener que implementar el controlador con el lenguaje de programación que se ha usado para desarrollar el framework.

Se indica explícitamente el nombre del host a través de la declaración *hostname*, de forma que este valor se puede usar junto con el puerto para indicar a otros contenedores el punto de acceso que les permite conectarse a este servicio.

En estos servicios se asocia una carpeta en el anfitrión con un volumen en el contenedor donde se guardan todos los *logs* generados por el framework. De esta forma, el usuario puede revisar estos ficheros para revisar errores u otro tipo de información relevante.

Por último, si se observa la declaración de los comandos a ejecutar en el contenedor, se pueden distinguir que aparecen dos. El primero permite registrar un DID (DID público) en el registro desplegado a través de VON. Esta operación solo se permite en redes desplegadas con el proyecto VON, ya que para redes públicas es necesario solicitar a los administradores de la red que se registre el DID. Nótese que esta operación solo es necesaria para aquellos agentes que tengan vayan a actuar como emisor, ya que el resto de los agentes no necesitan un DID público. El segundo comando arranca el servicio ACA-Py con una serie de opciones que se verán en el apartado 3.3.2 (Configuración framework Aries).

3.3.1.1.2 Docker Compose - Controladores

En este fichero todos los servicios parten de la imagen construida a partir del Dockerfile visto anteriormente. Estos servicios son: "*back-controller-TEST*", "*back-controller-ECDC*", "*back-controller-AENA*" y "*back-controller-FNMT*". El primero es el controlador gAp del agente del ciudadano europeo y el resto son los controladores gAp de los agentes indicados en el nombre.

Todos los servicios siguen la misma estructura y contienen las mismas declaraciones. A continuación, se muestra como ejemplo el caso del servicio *back-controller-ECDC*:

```
back-controller-ECDC:
   image: ${CTRL_ECDC_DOCKER_LABEL}
   build:
     context: .
     dockerfile: ./docker/Dockerfile
   container_name: ${CTRL_ECDC_DOCKER_LABEL}
   hostname: ${CTRL_ECDC_HOSTNAME}
   labels:
     name: ${CTRL_ECDC_DOCKER_LABEL}
     description: "Aries API back controller container of the ECDC"
   environment:
     FRAMEWORK_ENDPOINT: ${CTRL__FRAMEWORK_ECDC_ENDPOINT}
     FRONT_CTRL: ${FRONT_CTRL_ECDC_ENDPOINT}
     AIP: ${CTRL_AIP}
     CONTROLLER: ${ECDC_NAME}
   volumes:
     - ${PWD}/controllers/back-controller:/api
   depends_on:
     - acapy-ECDC
   ports:
     - "${CTRL_ECDC_OUTSIDE_PORT}:${CTRL_INSIDE_PORT}"
   networks:
     - main
   extra hosts:
     - "host.docker.internal:host-gateway"
```

Fragmento de código 4: Servicio del controlador gAp de la ECDC (PoC)

Se repite la dinámica anterior al definir los valores de las declaraciones y de las variables de entorno en el fichero ".env", al indicar explícitamente el nombre del *host* y al asociar los puertos

de los servicios en el contenedor los puertos libres en el anfitrión. En este caso, existe un único servicio, el del servidor Node.js.

La asociación de puertos permite conectarse fácilmente desde el anfitrión a un contenedor IP. Teniendo en cuenta la arquitectura de la PoC, esto permite a Postman realizar peticiones a los controladores gAp que están contenidos en contenedores Docker, ya que Postman está instalado en el anfitrión. Sin embargo, se ha visto que se necesita una comunicación bidireccional para permitir la recepción de eventos al servidor que se encarga de mostrarlos por el terminal y, como se ha mencionado en el diseño, estos servidores no están virtualizados. Esta recepción de eventos requiere que un contenedor Docker sea capaz de comunicarse con un servicio en el anfitrión, lo cual no está disponible por defecto.

Para permitir esta comunicación, hay que asociar una interfaz en el contenedor con la dirección IP que emplea Docker en el anfitrión. Esta asociación se realiza a través de la declaración *extra_hosts*, donde *host.docker.internal* es el nombre de la interfaz y *host-gateway* se traduce a la dirección IP[22]. De esta forma, un contenedor podrá acceder a un servicio en el anfitrión a través de la IP especificada en esta declaración.

Aunque ya se ha definido el volumen en el Dockerfile, no se ha llegado a asociar con una carpeta en el anfitrión. Esto se realiza en la declaración *volumes* del docker-compose.

Por último, se han especificado algunas variables de entorno que permiten definir el funcionamiento del controlador gAp:

- La variable "*FRAMEWORK_ENDPOINT*" contiene el punto de acceso del framework para que el servidor pueda realizar peticiones a este. El punto de acceso es el nombre del *host* del contenedor ACA-Py correspondiente y el puerto del servicio del administrador.
- La variable "*FRONT_CTRL*" contiene el punto de acceso del controlador frontal, que en el caso de la PoC es el servidor que muestra por el terminal los eventos recibidos. Este punto de acceso se emplea para transmitir estos eventos por el HTTP.
- La variable "*AIP*" indica si se emplea las funciones de AIP 1.0 o 2.0. En la práctica siempre se emplea AIP 1.0, pero se ha incluido para realizar pruebas con AIP 2.0.
- La variable "*CONTROLLER*" incluye el nombre del agente. Con esta variable conseguimos condicionar la lógica de negocio incluida en este servidor en función del agente que sea.

Con todas estas variables se consigue que un único controlador general (gAp) se pueda emplear en diferentes agentes, ya que los puntos de acceso permiten comunicarse con los contenedores adecuados y el nombre del agente pasado como variable permite personalizar la lógica de negocio.

3.3.1.2 Automatización con BASH

Para la automatización de las operaciones de despliegue se ha creado un único *script* a partir del lenguaje BASH llamado "**manage**". Este script permite realizar operaciones de construcción, arranque del escenario, parada, reanudación, eliminación o monitorización de *logs*. En el siguiente

[22] Aunque generalmente esta IP es 172.17.0.1, es posible que entre despliegues pueda cambiar. Por tanto, es mejor emplear *host-gateway* y que sea el propio Docker el que traduzca a la IP empleada.

fragmento se muestra la ayuda generada por el *script* (en inglés), donde se han resaltado los comandos para que se distingan mejor:

```
Usage: $0 [command] [options]

Commands:
  build - Build the docker images for the project.

  start|up - Creates the application containers from the built images and starts the
services based on the docker-compose.yml files.
      [-v] [--verbose] - Start up in verbose mode (default: background mode)
      [-a] [--agents] - Specify which agent(s) you want to deploy, by adding:
                        [TEST], [ECDC], [AENA], [FNMT]

  logs - Display the logs from the docker compose run (ctrl-c to exit). Options:
      [-c] [--controllers] - Print the logs only of the controllers.
           It is optional, but you can also specify which controller, by adding:
           [TEST], [ECDC], [AENA], [FNMT]
      [-f] [--frameworks] - Print the logs only of the frameworks.
           It is optional, but you can also specify which framework, by adding:
           [TEST], [ECDC], [AENA], [FNMT]
      [-t] [--tails] - Print the logs only of the tail server.
      [-a] [--all] - Print the logs of all the containers. DEFAULT OPTION.
      Example: ./manage logs -c ECDC ---> This will display the logs of the ECDC
              controller

  mock|startmock|mockservers - Start all the mock-servers.
     NOTE: Only works with gnome-terminal
     [-a] [--agents] - Specify which mock server(s) you want to deploy, by adding:
                       [TEST], [ECDC], [AENA], [FNMT]

  main|startall - Execute the following script commands: start / logs -t / logs -f /
                  logs -c / startmock
     NOTE: Only works with gnome-terminal

  stop - Stops the services. This is a non-destructive process (the volumes and
         containers are not deleted so they can be restarted).

  resume|restart - Resume stopped services (agents, VON and Tails SV)

  down|rm - Brings down the agent services and removes the agent volumes (storage)
            and agent containers. It doesn't remove the VON network.

  vondown - Brings down the VON network (container and volumes)

  alldown - Brinds down the agents and the VON network (container and volumes)

  logsrm - Remove all the logs from the /logs directory.

NOTE: You can override the default values of the ENV variables by passing them before
calling the script, for example:
    - LEDGER URL=http://test.bcovrin.vonx.io/ ./manage start
```

Fragmento de código 5: Ayuda del script manage (PoC)

Como apunte respecto a esta ayuda, los servidores "*mock*" arrancados con el comando "*mock*" son los servidores que muestran los eventos por el terminal.

Únicamente se va a detallar el proceso de arranque del escenario a través del comando "*start*", ya que se trata de la operación más importante.

Antes de ejecutar cualquier comando de arranque, el *script* se encarga de comprobar que el usuario tiene instaladas todas las dependencias en su equipo anfitrión. Para la PoC estas son cURL, git, docker, docker-compose, npm, ngrok y jq. Si falta alguna de ellas, el proceso se para y avisa al usuario.

Otro prerrequisito que cumple antes de empezar con el arranque es el de crear un fichero en blanco con el nombre ".env", el cual contendrá todas las variables de entorno consumidas por los contenedores Docker y una serie de variables de configuración.

Tras estas preparaciones, se procede a arrancar cada uno de los componentes del sistema a través de los siguientes pasos:

- Arranca el registro Indy a través del proyecto VON. Para esto, comprueba si existen ya los contenedores correspondientes. Si ya existen, se prosigue con el arranque, reanudando los contenedores en el caso de que estuviesen parados. Si no existen, se procede a llamar al proyecto VON para que arranque la red, clonando el proyecto del repositorio sólo en el caso de que no estuviese ya clonado y construyendo las imágenes necesarias sólo en el caso de que no estuviesen ya construidas.
- Arranca el servidor Tails a través del proyecto INDY TAILS. El proceso seguido es similar al seguido para arrancar el registro Indy a través del proyecto VON.
- Comprueba qué agentes han de desplegarse en función de los argumentos que se le hayan pasado en la llamada al *script*.
- Define la mayoría de las variables de entorno que va a contener el fichero ".env". En este proceso, se definen variables generales, como la IP que va a usar Docker para identificar al anfitrión o el punto de acceso del registro Indy, y variables específicas de cada agente. Para esto último, se emplea un bucle que recorre un *array* con los nombres de los agentes y va definiendo algunas variables en función del nombre y otras en función de la posición del *array*. Por ejemplo, los puertos se calculan sumando la multiplicación de la posición del *array* por 10 más un puerto base definido al inicio del *script*, y los nombres de *host* se definen con el nombre del componente junto con el nombre del agente (ejemplo: *CTRL_ECDC_ENDPOINT = controllerECDC*). Este mecanismo, aunque sencillo, permite una gran flexibilidad a la hora de añadir o eliminar agentes.
- Prepara y configura los *logs* que producen los frameworks. Lo primero que hace es crear la carpeta que va a contenerlos en el anfitrión, si es que no estaba previamente creada. Luego, por cada agente, se indica el nombre del fichero a crear en el que se van a incluir todos los *logs* del framework correspondiente. Este nombre incluye el nombre del agente y la fecha en formato IS0 8601 (YYYYMMDDTHHMMSS), de forma que sea sencillo identificarlo. Por último, crea los ficheros y les da permisos de escritura.
- Crea y recupera los túneles ngrok para los frameworks. Para esto, crea un fichero "ngrok.yml" donde se declaran los túneles a crear. Después, arranca los túneles a través del comando "ngrok". Una vez creados, se necesitan recuperar automáticamente los puntos de acceso que se han generado aleatoriamente para cada uno de los servicios, indicados

todos en la interfaz web que ofrece ngrok en el puerto 4040 del anfitrión. Para ello, se recorre un *array* con todos los agentes a desplegar y, uno a uno, se hace uso de "cURL" para recuperar la estructura de datos correspondiente y se hace uso de "jq" para recuperar el punto de acceso de esa estructura de datos. Puesto que a veces ngrok tarda unos segundos en crear estos túneles, se ha incluido está lógica en un bucle donde cada iteración sucede cada varios segundos y con un límite de intentos.

- Antes de proseguir con el arranque, el *script* se pausa hasta que la red Indy se termina de arrancar, ya que no se pueden iniciar los agentes sin que la red esté operativa, puesto que los emisores intentarán registrar sus DID públicos en esta. Para comprobar esto, se vuelve a incluir un bucle que, cada varios segundos, intenta recuperar con "cURL" el cuerpo de la interfaz web que incluye el proyecto VON.
- Por último, arranca con docker compose los diferentes servicios. Para esto, se le indica los ficheros docker-compose.yml a ejecutar, el nombre de los servicios a arrancar y la ruta del fichero ".env". Este mecanismo permite añadir o eliminar con facilidad servicios, variables de entorno e, incluso, ficheros docker-compose.yml.

Como nota adicional, si se ejecuta el comando "*main*" del *script*, este ejecutará todos los pasos anteriores y hará uso del comando "*gnome-terminal*" para abrir en diferentes ventanas del terminal los *logs* de los diferentes servicios Docker. Para esto, el *script* comprueba que el usuario tiene instalada esa dependencia en el anfitrión. Además, arrancará los servidores "*mock*".

El *script* se reutilizará y mejorará en el Producto Viable Mínimo, manteniendo la misma mecánica de definir variables de entorno para pasárselas a los contenedores Docker a través de los ficheros docker-compose y el fichero ".env".

3.3.2 Configuración framework Aries

Un framework Aries, tal y como se ha comentado anteriormente, se debe configurar para definir su comportamiento, de forma que ante ciertas situaciones sepa si informar al controlador y actuar en función de su respuesta o si lidiar directamente con estas. Esta configuración también puede determinar si el agente puede tomar el rol de emisor o no.

El framework ACA-Py se configura a través de variables de entorno o a través de argumentos al ejecutar el comando "**aca-py start**". Puesto que estos servicios se están configurando desde docker-compose, las variables de entorno se definen en este fichero. Como ya se mencionó en su momento, la versión 0.75 del framework contiene más de cien opciones diferentes, las cuales se han recogido en el ANEXO V: Opciones de configuración de ACA-Py.

En este apartado se van a comentar las opciones de configuración más relevantes que se han especificado para los frameworks. Para esto, se van a emplear los nombres de las opciones en minúscula, según el formato visto en el ANEXO V: Opciones de configuración de ACA-Py.

La opción "*auto-provision*" se ha activado (valor "*true*") en todos los frameworks, e indica que el arranque del servicio conlleva también un abastecimiento, según lo visto en el Estado del Arte.

Respecto a la configuración del transporte de mensajes entrantes y salientes de un agente, se usan las opciones "*it*" y "*ot*", respectivamente. Por ejemplo, para ECDC se han dado los siguientes

valores "*-it http '0.0.0.0' ${ECDC_AGENT_PORT} -ot http*", donde la primera variable es el puerto del servicio del agente. Este puerto deberá asociarse a un puerto del anfitrión a través de Docker. Adicionalmente, la opción "*endpoint*" indica el punto de acceso por el cual el agente escuchará mensajes entrantes recibidos por otros agentes, el cual será el que se incluya en los DIDDoc que genere. Para cada framework, el valor que se ha dado a esta opción es la URL generada por ngrok, ya que el agente debe estar disponible fuera del entorno local. La última opción a tener en cuenta respecto al transporte es "*webhook-url*", la cual indica la URL donde el controlador escuchará la recepción de los eventos generados por el framework. Este *webhook* se definirá el apartado 3.3.3 (Backend).

Respecto al administrador del framework, la principal opción a configurar es "*admin*", donde se establece el punto de acceso del administrador. El puerto seleccionado aquí deberá asociarse a un puerto del anfitrión a través de Docker. Otra opción a considerar es "*admin-insecure-mode*", que permite acceder a la interfaz del administrador sin una contraseña. Esta opción se ha activado, ya que no se está en un despliegue en producción.

Para que el framework sea capaz de conectarse con una red Indy, ya se ha visto que es necesario que recupere el fichero génesis de la red. El valor indicado en la opción "*genesis-url*" será el que permita al framework recuperar este fichero.

De igual forma, aunque únicamente para los agentes que vayan a actuar de emisor, se le debe indicar al framework la dirección del servidor Tails a emplear en la emisión y revocación de credenciales, en este caso con la opción "*tails-server-base-url*".

Respecto a las opciones de configuración de la cartera, se van a mencionar únicamente algunas. La opción "*seed*" especifica la semilla que se va a utilizar para crear el DID público del agente. Si el agente no va a actuar de emisor, entonces necesita activar la opción "*wallet-local-did*", la cual usa la semilla anterior para crear un DID local en lugar de uno público. Y, por último, la opción "*wallet-key*" establece la contraseña para acceder a la cartera.

Respecto a las opciones relativas a la generación de los *logs*, se emplean dos. La primera es "*log-file*", que indica la ruta dentro del contenedor al fichero que va a incluir los *logs*. Y la segunda es "*log-level*", la cual permite incluir todos los *logs* a partir del nivel de gravedad indicado. Esta última opción admite los valores "*info*", "*warning*", "*error*" y "*critical*".

Adicionalmente a las opciones de los *logs*, se pueden activar una serie de opciones de depuración, las cuales enriquecen estos registros. En esta PoC se han activado las opciones "*debug-connections*", "*debug-credentials*" y "*debug-presentations*", las cuales depuran las acciones de establecimiento de conexión, expedición de credenciales y presentación de credenciales, respectivamente.

Y, por último, existen opciones que indican al framework que realice una serie de opciones sin consultar con el controlador. Para esta PoC, se han activado en todos los frameworks las opciones "*auto-store-credential*", "*auto-respond-credential_request*" y "*auto-verify-presentation*", las cuales almacenan credenciales expedidas automáticamente, responden peticiones de expedición de credenciales tras mandar una oferta y verifican automáticamente presentaciones recibidas, respectivamente. Adicionalmente, todos los framework menos el del ciudadano (*acapy-TEST*)

tienen activada la opción "*auto-respond-presentation-request*", la cual permite presentar automáticamente peticiones de presentación recibidas. No se han indicado otras opciones menos relevantes. Nótese que algunas de estas opciones se deberían desactivar en producción.

Otra opción interesante es "*label*", la cual permite especificar el nombre con el cual otros agentes van a conocer a este agente cuando se establezca una conexión.

A continuación, se muestra como ejemplo las variables de entorno definidas en el framework de ECDC:

```
ACAPY_AUTO_PROVISION: "true"
ACAPY_ENDPOINT: ${ECDC_SITE_URL}
ACAPY_WEBHOOK_URL: "${CTRL_ECDC_ENDPOINT}/webhooks"
ACAPY_ADMIN_INSECURE_MODE: "true"
ACAPY_GENESIS_URL: ${GENESIS_URL}
ACAPY_TAILS_SERVER_BASE_URL: "${TAILS_URL}"
ACAPY_WALLET_SEED: ${ECDC_WALLET_SEED}
ACAPY_WALLET_TYPE: ${WALLET_TYPE}
ACAPY_WALLET_NAME: ${ECDC_WALLET_NAME}
ACAPY_WALLET_KEY: ${ECDC_WALLET_KEY}
ACAPY_LOG_FILE: ${LOG_ECDC_PATH}
ACAPY_LOG_LEVEL: ${LOG_LEVEL}
ACAPY_DEBUG_CONNECTIONS: "true"
ACAPY_DEBUG_CREDENTIALS: "true"
ACAPY_DEBUG_PRESENTATIONS: "true"
ACAPY_AUTO_ACCEPT_INVITES: "true"
ACAPY_AUTO_PING_CONNECTION: "true"
ACAPY_AUTO_RESPOND_MESSAGES: "true"
ACAPY AUTO STORE CREDENTIAL: "true"
ACAPY AUTO RESPOND CREDENTIAL REQUEST: "true"
ACAPY AUTO VERIFY PRESENTATION: "true"
ACAPY_AUTO_RESPOND_PRESENTATION_REQUEST: "true"
ACAPY_LABEL: ${ECDC_ACAPY_LABEL}
```

Fragmento de código 6: Variables de entorno del framework de ECDC (PoC)

Y, a continuación, se muestra el comando de arranque del framework de ECDC:

```
aca-py start
    -it http '0.0.0.0' ${ECDC AGENT PORT}
    -ot http
    --admin '0.0.0.0' ${ECDC_ADMIN_PORT}"
```

Fragmento de código 7: Comando de arranque de ACA-Py para ECDC (PoC)

3.3.3 Backend

Teniendo en cuenta el diseño de la arquitectura de la Prueba de Concepto, el backend del controlador contiene únicamente el controlador general gAp. Como recordatorio, este componente tiene como propósito abstraer parte de la complejidad de la interacción con el framework ACA-Py, de forma que las peticiones que se realizan desde el cliente al gAp son mucho más sencillas que las que realiza este último al framework, de forma que internamente completa el cuerpo y los parámetros de las peticiones. Esta simplificación también se da hacia el sentido contrario, con el envío de eventos recibidos por el framework al cliente, eliminando campos innecesarios (en el contexto de esta PoC) o renombrándolos por otros que resulten más

familiares. Por ejemplo, los elementos criptográficos no se transmiten hacia el cliente, puesto que en general una persona no tendrá conocimientos suficientes para manipularlos. Nótese que esta información se elimina en la transmisión de eventos hacia el cliente, pero que se mantiene dentro del propio framework, por lo que el agente seguirá disponiendo de la información criptográfica.

En el caso de la Prueba de Concepto, la lógica de negocio se ha añadido dentro del propio componente para evitar tener que desplegar otro servidor por cada agente. Esta decisión se ha tomado por dos razones. La primera es para ahorrar tiempo en el desarrollo de la PoC para poder invertirlo luego en el MVP. La segunda viene dada por la poca lógica de negocio que hay, dada la simplicidad del escenario y por la falta de interfaces web.

Para poder reutilizar fácilmente este componente en el MVP, se ha decidido separar lo máximo posible la lógica de negocio, de forma que existe un único fichero por agente donde se incluye toda esta lógica. Adicionalmente, se han creado otros ficheros con los esquemas de credenciales, las cuales también pertenecen a esta parte de negocio.

Este servidor API REST se ha desarrollado con Node.js y Express. Todos los servidores desarrollados en este TFM sobre estas tecnologías contienen tres ficheros importantes. El primero es el llamado "*package.json*", el cual contiene una serie de metadatos acerca del servidor (como el nombre del autor o la licencia), una lista de dependencias que NPM deberá instalar en la carpeta "*/node_modules*" y una lista de comandos. Un comando relevante que se ha incluido es "*start*", el cual permitirá iniciar el servidor a través del fichero "*/bin/www*" cuando se ejecute "*npm start*" en el terminal. El segundo fichero es el mencionado "*www*", el cual contiene todas las órdenes necesarias para desplegar el servidor en local en el puerto indicado. El puerto a emplear para el despliegue en local se extrae de la variable de entorno definida en el Dockerfile. El tercer fichero es el llamado "app.js", el cual es el punto de entrada a toda la lógica del servidor.

Adicionalmente, todos los proyectos de este TFM desarrollados sobre Node.js y Express tienen la misma organización del código en carpetas, de forma que se tienen las carpetas "*/bin*", "*/config*", "*/controllers*", "*/material*", "*/middlewares*", "*/routes*" y "*/services*". La carpeta "*/bin*" contiene la lógica relacionada con el despliegue, como el propio servidor o el socket en los servidores que lo implementen. La carpeta "*/config*" contiene ficheros de configuración, donde se definen cosas como la configuración para conectarse a una base de datos o información estática, como la traducción de nombres en inglés al español. La carpeta "*/material*" contiene archivos consumidos o generados por el propio servidor, como los esquemas de las credenciales o códigos QR. La carpeta "*/routes*" contiene la definición de la API REST, donde se definen los métodos, las rutas, los *middlewares* a emplear que se encuentran en la carpeta "*/middlewares*" y los controladores que gestionan la lógica de la API y que se encuentran en la carpeta "*/controllers*". Por último, la carpeta "*/services*" contiene la lógica de diferentes servicios y que se consume por los controladores de la carpeta "*/controllers*", como el servicio que conecta a una base de datos o el servicio que comunica con el framework. Como nota adicional, los *webhooks* se definen en la carpeta "*/routes*" y se gestionan en la carpeta "*/controllers*", ya que se tratan como cualquier otra API del servidor.

En este apartado no se va a detallar todo el código que conforma el controlador gAp, pero sí se van a presentar las diferentes agrupaciones de rutas disponibles y algunas decisiones de

implementación importantes. Adicionalmente, todas las rutas relativas únicamente al controlador general están detalladas en el ANEXO VI: API del controlador general, todos los eventos enviados por el controlador general están detallados en el ANEXO VII: Eventos del controlador general y todas las rutas relativas a la lógica de negocio de la PoC están detalladas en el ANEXO VIII: Rutas específicas de la Prueba de Concepto.

Las diferentes agrupaciones de rutas del controlador gAp son las siguientes:

- Estado. Son todas las rutas relacionadas con proporcionar al cliente detalles sobre el estado actual del agente, desde el punto de vista de la disponibilidad de los componentes.
- Conexiones. Son todas las rutas relativas a la creación y gestión de las comunicaciones seguras entre agentes. También incluye rutas para el envío de mensajes sencillos.
- Cartera. Son todas las rutas relacionadas con la creación y gestión de elementos almacenados en la cartera o en el VDR. Por ejemplo, recuperación de los DID que ha creado el agente o la publicación de un esquema de credencial en el VDR.
- Expedición. Son todas las rutas relacionadas con la expedición de credenciales, tanto desde el punto de vista del emisor como del titular. Son agnósticas al esquema de credencial, esto quiere decir que, por ejemplo, la ruta que envía una oferta de credencial funciona para cualquier credencial.
- Presentación. Son todas las rutas relativas a la presentación y verificación de credenciales verificables, tanto desde el punto de vista del emisor como del probador (titular). Al igual que antes, son agnósticas al esquema de credencial.
- Revocación. Son todas las rutas relacionadas con la revocación de credenciales.
- *Webhooks*. Son las rutas que usa el framework para enviar eventos al controlador. Estas rutas las define el propio framework [122][23], por lo que hay que definirlas en función de esto. Todas ellas empiezan por la ruta "*/webhooks/topic*".

Prácticamente todas las rutas anteriores, salvo los *webhooks* y algunas rutas específicas, invocan al framework a través del administrador. Como ya se ha mencionado, este administrador expone una interfaz API REST para esta interacción. Esta interfaz incluye también una interfaz gráfica web accesible en local a través del puerto asignado al administrador en la configuración del framework. Esta interfaz web se genera automáticamente a través de Swagger [123] y permite realizar llamadas desde la propia web. Esto tiene dos utilidades: la primera es que permite hacer pruebas con el framework sin la necesidad de desarrollar un agente, y la segunda es que permite al desarrollador conocer qué rutas expone el framework y cómo se usan. Por tanto, esta interfaz ha sido la fuente principal de aprendizaje sobre ACA-Py que ha usado el alumno.

Los *middlewares* y controladores que gestionan la lógica de las rutas anteriores son las que permiten simplificar y abstraer la interacción con el framework. En ellos, se realizan, entre otras omitidas, las siguientes operaciones generales:

- Se obtienen de las variables de entorno el punto de acceso del framework.

[23] En la referencia indicada, la cual es la referencia oficial, solo aparecen un número limitado de estas rutas [138].

- Se realizan comprobaciones para evitar que el usuario introduzca valores erróneos, como un predicado ZKP con una operación que no se está entre las disponibles ($<$, $\leq$, $>$, $\geq$).
- Se formatean las estructuras de datos que van a incluirse en los cuerpos de las peticiones hacia el framework, evitando errores en el framework.
- Se codifican y decodifican elementos, como las invitaciones de conexión basadas en enlaces o en códigos QR.
- Se generan códigos QR al generar una invitación de conexión y se guarda en la carpeta de "*/material*".
- Se comprueba que existen ciertos elementos antes de operar sobre ellos, para evitar errores en el framework. Por ejemplo, se comprueba que existe una conexión antes de expedir una credencial sobre ella.
- Se obtienen elementos del framework antes de realizar ciertas operaciones que las puedan necesitar, evitando así al usuario realizarlo manualmente. Por ejemplo, se recupera automáticamente el esquema y la definición de credencial del framework antes de enviar una oferta de credencial a partir de la información contenida en la petición que ha realizado el usuario. De esta forma, el usuario necesita una única petición para enviar una oferta de credencial en lugar de tener que realizar varias peticiones para recuperar todos los elementos necesarios.
- Se preparan automáticamente los cuerpos de algunas peticiones. Esta operación se basa en todas las anteriores además de en otros detalles: se rellenan automáticamente ciertos campos, como campos de configuración; se obtiene de una variable de entorno el valor de AIP para establecer la versión; y se calculan ciertos valores automáticamente, como fechas actuales. Por ejemplo, el cuerpo de una oferta de credencial se rellena automáticamente en función de todos los detalles comentados.

Aunque se han omitido ciertos detalles, se puede observar como este componente, en efecto, cumple con el objetivo de abstraer cierta dificultad de cara al usuario y de facilitarle las cosas.

Para la propia comunicación con el framework, se ha creado un servicio en la carpeta "*/services*" que contiene una clase JavaScript, la cual contiene una serie de métodos que permiten interactuar con las diferentes rutas del framework ACA-Py. Por ejemplo, hay un método llamado "*acceptConnection(connection_id)*" que permite aceptar una conexión a partir de su identificador, de forma que el método realiza una petición a la ruta "*/connections/{connection_id}/accept-request*" que expone el framework y espera una respuesta. Se ha empleado la biblioteca *axios* para esta comunicación.

Para el caso de los eventos recibidos por el framework en los *webhooks* definidos, la simplificación de cara al cliente se basa en los siguientes elementos:

- Se obtiene de las variables de entorno del punto de acceso del cliente.
- Se clasifican y separan los eventos por su tema y por su estado. Por ejemplo, un evento con el tema "conexiones" (*connections*) incluirá información sobre una conexión, donde aparecerá su estado (*invitation*, *request*, *response*, etc.).
- Sobre los diferentes casos (tema, estado), se realizan diferentes cosas. Por ejemplo, para algunos casos se imprimirá por la pantalla del terminal lo que ha sucedido, mientras que

en otros casos se enviará el evento al cliente. De esta forma, se filtran los eventos que le llegan al cliente, descartando los menos relevantes.

- En cada evento se modifica y descarta información. Por ejemplo, se eliminan las pruebas criptográficas asociadas a una presentación, puesto que el cliente no va a querer hacer nada con ello.
- Se envían los eventos al cliente a través del *webhook* que tenga definido. Esto implica que todos los clientes deberán implementar la ruta "*/webhooks*" con el método POST. Los eventos que le llegan al cliente tendrán dos campos que identifican el tema y el estado, para que pueda filtrar estos eventos, y un campo con el contenido del propio evento.

Nótese que en el caso de los *webhooks*¸ cuando se menciona al cliente, se refiere realmente a un servidor que está actuando como cliente desde el punto de vista de este servidor gAp. Es por esta razón que esta comunicación de eventos se ha empleado la biblioteca *axios* y no se ha creado un socket.

En el caso de la lógica de negocio, se ha creado un fichero separado por cada agente que donde se incluyen todos los controladores de las rutas. Las rutas específicas de cada agente implican operaciones como la publicación de los esquemas de las credenciales de identidad y vacunación, las definiciones de las credenciales o la recuperación de estos elementos. Para esto, se incluyen los esquemas de las credenciales en formato JSON en la carpeta "*/material*", junto con los códigos QR generados al crear invitaciones de establecimiento de conexión.

3.3.4 Frontend

Como ya se ha mencionado en el diseño de la PoC, se ha descartado el diseño y la implementación de interfaces gráficas en esta implementación. El frontend, entonces, se compone de un cliente Postman instalado en el anfitrión que permite realizar peticiones HTTP al controlador gAp y de un servidor denominado servidor "*mock*" mínimo cuyo único propósito es el de mostrar por terminal los eventos recibidos del controlador gAp.

Respecto a Postman, se ha creado una colección por agente con llamadas a todos los puntos de acceso descritos en el ANEXO VI: API del controlador general y en el ANEXO VIII: Rutas específicas de la Prueba de Concepto. Con este cliente se consigue realizar las llamadas necesarias para poder reproducir el escenario descrito anteriormente, con ayuda de los eventos visualizados en el servidor "*mock*".

Respecto al servidor "*mock*", este es un servidor desplegado en el anfitrión a través del *script* de despliegue desarrollado sobre Node.js y Express, que se compone únicamente de los elementos básicos para que se pueda arrancar y de una ruta "*/webhook*" con el método POST por donde se reciben los eventos del controlador general (gAp) y que su única lógica es la de imprimir el evento por el terminal. Este componente era necesario para poder recibir los eventos, los cuales contienen información que permiten realizar ciertas peticiones a través de Postman, y los cuales están descritos en el ANEXO VII: Eventos del controlador general.

Capítulo 4. PRODUCTO VIABLE MÍNIMO (MVP)

En este capítulo se va a presentar el Producto Viable Mínimo que se ha implementado, el cual ha permitido validar los conceptos de SSI y las tecnologías empleadas en la Prueba de Concepto.

Puesto que los componentes de la Prueba de Concepto se crearon con el objetivo de poder reaprovecharse, la carga de esfuerzo y tiempo de esta implementación incide sobre la especificación de un escenario lo suficientemente realista y complejo; y sobre el diseño e implementación de un sistema que, con la reutilización de estos componentes, permita reproducir dicho escenario. Adicionalmente, se pretende ocultar la complejidad de conceptos complejos a los usuarios, como los DID o el VDR; facilitando así la usabilidad de los sistemas resultantes del sistema.

Al igual que el capítulo anterior, este se plantea en tres apartados principales que reflejan las diferentes fases que se han seguido: análisis del escenario, diseño del sistema e implementación.

4.1 Análisis

Como se mencionó en el capítulo anterior, la necesidad de especificar un escenario en la Prueba de Concepto se aprovechó para especificar uno cercano al escenario del Producto Viable Mínimo. Por esta razón, el resumen del escenario es prácticamente el mismo; donde hay un ciudadano español que quiere viajar en avión a otro país de la Unión Europea y que debe presentar unas credenciales que le permitan identificarse y probar que cumple los requisitos sanitarios.

Sin embargo, esta vez se ha tratado de detallar un escenario más realista, donde aparecen más credenciales, donde los esquemas de las credenciales se basan en esquemas oficiales, donde los agentes se acercan un poco más a la realidad y donde las comprobaciones sobre requisitos sanitarios son más cercanas a las solicitadas durante la pandemia.

4.1.1 Esquemas de credenciales

En este escenario se emplean varias credenciales. Por un lado, está la credencial de identidad, cuyo esquema se ha mejorado respecto al visto en la Prueba de Concepto. Y, por otro lado, están las credenciales COVID-19, las cuales se han basado en los esquemas oficiales publicados por la Comisión Europea [124], específicamente, en la versión 1.3.0 de la especificación oficial del esquema [125] y en la versión 1.0 de los valores oficiales admitidos [126]. Este conjunto de credenciales COVID-19 se compone de una credencial de vacunación, de una credencial de prueba (test) y de una credencial de recuperación.

En estos esquemas de credenciales COVID-19 se han realizado modificaciones respecto a los esquemas oficiales para adaptarlos a las capacidades de la identidad autosoberana y de las tecnologías empleadas, y con el objetivo de mejorarlos.

Los atributos de estos esquemas se han nombrado de acuerdo con el esquema oficial. Sin embargo, los valores que permiten estas credenciales se han cambiado por valores legibles por humanos, ya que los esquemas oficiales sugerían emplear códigos oficiales que identifican estos

valores. Por ejemplo, en la credencial oficial de vacunación el agente objetivo se especifica con el código "*840539006*" de la SNOMED CT (GPS) [125], en lugar de por su nombre "*COVID-19*". En producción quizá sí sea más apropiado utilizar estos códigos, pero en este MVP no se van a emplear.

Nótese que cuando se dice que un atributo admite unos valores u otros, se refiere a que la lógica del emisor los limita, no el propio esquema de la credencial. Uno de los principales propósitos de la identidad autosoberana es separar la lógica de negocio del contenido de las credenciales.

Adicionalmente, se ha añadido una credencial que autoriza a una entidad a expedir credenciales COVID-19, con el fin de mostrar como funcionaría una cadena de autorización en SSI.

Por último, todas las fechas de todas las credenciales se han transformado a tiempo Unix para poder aplicar predicados ZKP sobre este campo, sin incluir la hora para minimizar los datos y evitar la correlación de estos, de acuerdo con las indicaciones de la RGPD.

4.1.1.1 Esquema de credencial de identidad

Nombre técnico del esquema: **covID_MVP_Identity**.

Atributo	Formato	Detalles
full_name	String	Nombre y apellidos del ciudadano.
birthday	Number	Fecha de nacimiento en formato ISO 8601 [118].
address	String	Dirección del domicilio donde reside el ciudadano.
ID_number	String	Número de identidad europeo del ciudadano.
photo_url	String (URL)	URL de la foto del DNI del ciudadano.
photo_hash	String	Resumen de la URL de la foto.

Tabla 3: Esquema de credencial de identidad (MVP)

Este esquema de credencial de identidad incluye tres mejoras respecto al visto en la PoC:

1. El formato del atributo "*birthday*" es ahora numérico, de forma que se permiten directamente predicados ZKP sobre este, sin tener que crear otro campo con el mismo dato en formato tiempo Unix. Además, se evita la limitación de fechas de nacimiento anteriores al 1 de enero de 1970.
2. El identificador del ciudadano ahora ya no se limita al número español (DNI), sino que se emplea el formato extendido europeo [127]. En este caso se ha simplificado, omitiendo dígitos como los que representan el género, la fecha de nacimiento o el nombre y apellidos. Un ejemplo de valor válido es: ID**ESP**BAA0005895**99999999R**, donde se destacan el país y el número de identidad en ese país (número del DNI en el caso de España).
3. Se incluye el resumen de la URL de la foto, aumentando así su seguridad. En la práctica, se podría hacer uso de tecnologías como IPFS para aumentar aún más la seguridad.

4.1.1.2 Esquema de credencial de vacunación

Credencial de vacunación que incluirá información de la dosis administrada. Además, incluye un campo con el número total de dosis administradas hasta el momento de la expedición.

Teniendo en cuenta el esquema oficial, se han realizado las siguientes mejoras:

1. No se indica el número necesario de dosis para completar la pauta de vacunación. La razón radica en que este detalle implica una lógica que puede variar en el tiempo, por lo que este tipo de campos deben moverse a la lógica que usa el verificador para comprobar si una presentación cumple con los requisitos.
2. No se incluye el campo que indicaba que dosis era la que se había administrado respecto al número total (por ejemplo: dosis número 2 de 4). Esto es una consecuencia directa de no incluir el número total de dosis necesario para completar la pauta de vacunación.
3. No se incluye el identificador de la propia credencial, ya que este era un campo sobre el que se podía correlar información del individuo y gracias a que las credenciales AnonCreds permiten presentar pruebas ZKP que verifican que la credencial pertenece al individuo.
4. No se incluye el identificador del emisor. La razón viene dada porque con las credenciales AnonCreds esto no es necesario, ya que incluyen el DID del emisor y presentación de pruebas ZKP de las firmas empleadas por este.

Por último, se genera una credencial con información sobre la propia dosis y no con información sobre de esta y de las anteriores porque el emisor de la última credencial de vacunación puede ser diferente al de la penúltima. Esto implicaría que el último tendría que certificar, en su nombre, información que no le corresponde certificar.

Nombre técnico del esquema: **covID_MVP_Vaccination**.

Atributo	Formato	Detalles
tg	String	Identifica la enfermedad o el agente objetivo (“*COVID-19*”).
sd	Number	Número de dosis total que se han administrado.
vp	String	Tipo de dosis administrada (“*mRNA*” o “*antigen*”).
mp	String	Producto de la dosis administrada (por ejemplo: “*Moderna*”).
ma	String	Fabricante de la dosis que se ha administrado.
dt	Number	Fecha (tiempo Unix) en la cual se administró la última dosis.
co	String	País en el cual se administró la vacuna. Formato en estándar ISO-3166 de dos letras [119].

Tabla 4: Esquema de credencial de vacunación COVID-19 (MVP)

4.1.1.3 Esquema de credencial de test

Credencial que incluye información de una prueba COVID-19 (PCR o antígenos) realizada

Teniendo en cuenta el esquema oficial, se han realizado las siguientes mejoras:

1. Por los mismos motivos que en la credencial de vacunación, se han omitido el campo del identificador de la credencial del emisor y el campo del identificador del certificado.
2. Se han unificado los campos “nm” y “ma” del esquema oficial, que identificaban el producto de una prueba PCR y de una prueba de antígenos, respectivamente. El campo resultante se ha nombrado “tn” (*test name*). El alumno cree que, dada la existencia del campo “tt”, incluir dos campos diferentes para identificar a la prueba era innecesario.

3. No se incluye el campo que identifica al centro que realizó el test. Esto es porque existen dos alternativas: o el centro es el propio emisor de la credencial, lo que implica que su DID ya se incluye en la credencial; o el emisor es otra entidad, lo que implica que esta información debería estar almacenada de forma interna de acuerdo con la responsabilidad contractual que exista entre estas dos entidades. Al no incluir este campo se evita la identificación del ciudadano a partir de este y de otros campos, como la fecha y el resultado de la prueba.

Nombre técnico del esquema: **covID_MVP_Test**.

Atributo	Formato	Detalles
tg	String	Identifica la enfermedad o el agente objetivo ("*COVID-19*").
tt	String	Tipo de prueba realizada. Valores: "NAAT" (*Nucleic Acid Amplification Test*) para las pruebas PCR y "RAT" (*Rapid Antigen Test*) para las pruebas de antígenos.
tn	String	Nombre (NAAT)[24] o identificador (RAT)[25] del test realizado.
tr	String	Resultado del test ("*Detected*" o "*Not Detected*").
sc	Number	Fecha (tiempo Unix) en la cual se tomó la muestra.
co	String	País en el cual se tomó la muestra para el test. Formato en estándar ISO-3166 de dos letras [119].

Tabla 5: Esquema de credencial de test COVID-19 (MVP)

4.1.1.4 Esquema de credencial de recuperación

Credencial que incluye un rango temporal durante el cual el ciudadano tiene menos posibilidades de contagiarse. Esta credencial no parece que se expida en la actualidad por regla general, pero puede ser un mecanismo más para que un ciudadano demuestre inmunidad.

La lógica acerca de qué fechas han de indicarse en la credencial es la misma que indica el esquema oficial, donde estas fechas se establecen a partir de la última PCR positiva. Esta lógica no pertenece al propio esquema de la credencial, sino que debe establecerse en el agente emisor.

Teniendo en cuenta el esquema oficial, se han realizado las siguientes mejoras:

1. Por los mismos motivos que en la credencial de vacunación, se han omitido el campo del identificador de la credencial del emisor y el campo del identificador del certificado.
2. No se incluye el campo que indicaba la fecha de la última PCR positiva, ya que esta información se debería indicar a través de la presentación de una credencial de test PCR. Sin embargo, se puede llegar a considerar que se debe verificar que un centro haya expedido de forma legítima una credencial de revocación al revisar la fecha de la última PCR positiva. No obstante, el alumno considera que esto es un tema legal que se debería auditar de formas alternativas, puesto que toda esta información deberá almacenarse en el sistema informático del centro.

[24] Contendrá, separados por una coma, primero el nombre del fabricante y luego el del producto comercial.
[25] Existe una base de datos oficial con los identificadores en formato legible y en formato JSON [132].

Nombre técnico del esquema: **covID_MVP_Recovery**.

Atributo	Formato	Detalles
tg	String	Identifica la enfermedad o el agente objetivo ("*COVID-19*").
df	Number	Fecha (tiempo Unix) de inicio del rango de recuperación.
du	Number	Fecha (tiempo Unix) final del rango de recuperación.
co	String	País en el cual se expide la credencial. Formato en estándar ISO-3166 de dos letras [119].

Tabla 6: Esquema de credencial de recuperación COVID-19 (MVP)

4.1.1.5 Esquema de credencial de autorización

Credencial que autoriza a un emisor la expedición de credenciales COVID-19 y que debe ser expedida por un emisor de mayor rango dentro de la estructura organizacional de cada país. Por ejemplo, en España, el Ministerio de Sanidad expediría estas credenciales a los diferentes centros u hospitales del territorio español. Así, si un verificador no conoce el centro que ha expedido una credencial COVID-19, puede establecer una conexión segura con este a partir de su DID público y solicitarle una presentación de esta credencial de autorización. Esta credencial permite implementar una cadena de autorización.

Este esquema incluye un atributo que indica desde cuando una credencial es válida, es decir, desde cuando la institución está autorizada a expedir credenciales COVID-19. Esta fecha, junto con una prueba de no revocación para un instante de tiempo dado, permiten al verificador asegurarse que una credencial fue expedida en un periodo válido.

Además, contiene un atributo que indica el agente objetivo autorizado para expedir credenciales, de forma que esta credencial se extiende a otros tipos de agentes además del COVID-19.

Nombre técnico del esquema: **covID_MVP_Authorization**.

Atributo	Formato	Detalles
tg	String	Identifica la enfermedad o el agente objetivo ("*COVID-19*").
df	Number	Fecha (tiempo Unix) de inicio de validez.
co	String	País en el que se encuentra la institución. Formato en estándar ISO-3166 de dos letras [119].

Tabla 7: Esquema de credencial de autorización (MVP)

4.1.2 Agentes

Este MVP se compone de tres agentes principales (sistema sanitario, aerolínea, ciudadano) y de varios agentes secundarios que se presentarán en este apartado.

4.1.2.1 Personal Sanitario de la Comunidad de Madrid

El agente será una aplicación software que simulará la función que ofrece actualmente el Registro Unificado de Vacunación COVID-19 [128] en el ámbito sanitario dentro de la Comunidad de Madrid. Esta aplicación web permite al personal sanitario registrar todos los detalles de vacunación en el sistema informático de la Comunidad de Madrid.

En esta aplicación se podrá añadir y consultar, por paciente, información relativa a las dosis de vacunación COVID-19 administradas y a las pruebas PCR o antígenos realizadas. Dado que se está añadiendo la opción de incluir más información además de la de vacunación, en este TFM se va a llamar a este agente Registro Unificado COVID-19 (**RUC19**).

El sistema informático está compartido por todos los hospitales de la Comunidad de Madrid. Esto que implica que el personal sanitario debería tener un mecanismo de autenticación y autorización interno a través de usuario / contraseña o certificados electrónicos con el que podrán acceder a este. Este mecanismo interno se va a omitir, dado que queda fuera de los objetivos del TFM. Nótese que también se podría emplear las credenciales verificables para esta autenticación y autorización.

Por tanto, los sanitarios harán uso de esta aplicación para registrar datos de vacunación o de pruebas COVID-19 de los pacientes. Esta información quedará registrada en las bases de datos y, tras un tiempo indefinido, será el propio sistema el que expida las credenciales correspondientes, bien porque el paciente las solicita o bien porque estas se expiden a la vez y automáticamente al final del día. Sin embargo, en el MVP, por motivos prácticos para la demostración, las credenciales se expedirán a través de un botón en la interfaz de la aplicación sanitaria.

El personal también puede hacer uso de esta aplicación para visualizar los datos relacionados con vacunas y a las pruebas COVID-19 de los pacientes.

A continuación, se muestra un diagrama de casos de uso de elaboración propia de este agente:

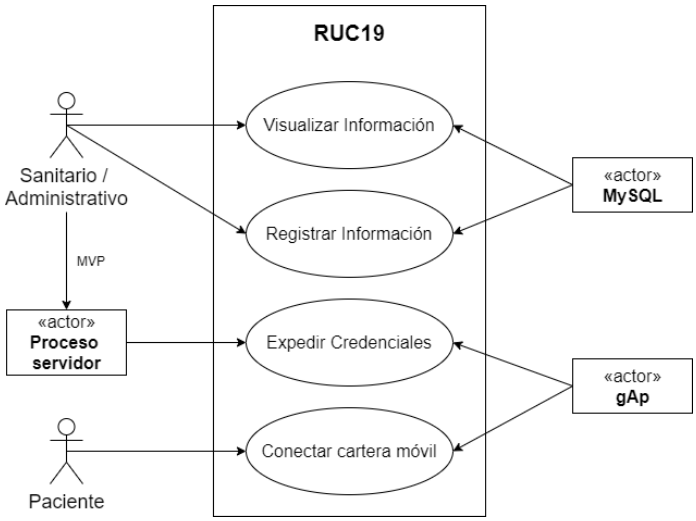

Figura 16: Diagrama de casos de uso de RUC19 (MVP)

Por otra parte, el servidor expedirá automáticamente las credenciales de recuperación a los pacientes cuando se expida una credencial de prueba PCR positiva. Como trabajo futuro, el sistema informático podría revocar automáticamente las credenciales de recuperación en la fecha de límite de validez indicada en ellas.

La aplicación contendrá entonces una lista con todos los pacientes de la Comunidad de Madrid identificados por su número de identidad y por su número CIPA. Adicionalmente, cada paciente tendrá una sección con información relativa a las vacunas y a las pruebas COVID-19, a la que se puede acceder manualmente a través de la búsqueda del paciente en la lista o automáticamente al recibir una presentación de la credencial de identidad por parte del paciente.

El agente deberá responder automáticamente a las peticiones de presentaciones de la credencial de autorización recibidas por un verificador que, a su vez, ha recibido una credencial COVID-19 y no conoce al emisor. Esta cadena de autorización se ha planteado de la siguiente forma: cada uno de los 27 países de la UE deberá expedir un número de credenciales de autorización al nivel inmediatamente inferior en su jerarquía sanitaria. En este escenario, el Ministerio de Sanidad español expedirá la credencial a este agente (RUC19).

4.1.2.2 Aerolínea Iberia

El agente será una aplicación integrada en una máquina con una pantalla (quiosco), similar a la que nos podemos encontrar en el metro. Este quiosco estará situado en la puerta de embarque de un aeropuerto y su función será la de comprobar que los viajeros cumplen unas condiciones sanitarias relativas al COVID-19. Para esto, la aplicación mostrará un código QR por la pantalla que permita al ciudadano establecer una conexión y, una vez la conexión esté establecida, enviará varias peticiones de presentación de credencial de forma automática.

Al lado del quiosco se encontrará un vigilante de seguridad controlando que las personas no están suplantando la identidad de otros y que las personas que no cumplan con los requisitos sanitarios no embarquen en el avión, todo en base a las presentaciones recibidas del viajero.

Tras recibir las presentaciones, la aplicación mostrará por pantalla la foto del ciudadano y un indicativo simple que muestra si puede embarcar o no, dependiendo de si cumple con los requisitos sanitarios.

En este MVP, el agente será un agente de servidor desplegado únicamente en local que ofrece una interfaz web. Sin embargo, en producción sería más conveniente desarrollar un software a medida de la máquina.

Como posible trabajo futuro, se podría incluir un algoritmo de reconocimiento facial en lugar de mostrar la foto por pantalla y se podría sustituir al vigilante de seguridad por algún tipo de barrera física. Adicionalmente, el hardware y el software lo podría ofrecer y auditar el gobierno para evitar acciones maliciosas, como el almacenamiento de la información obtenida de las presentaciones recibidas.

A continuación, se muestra un diagrama de casos de uso de elaboración propia de este agente:

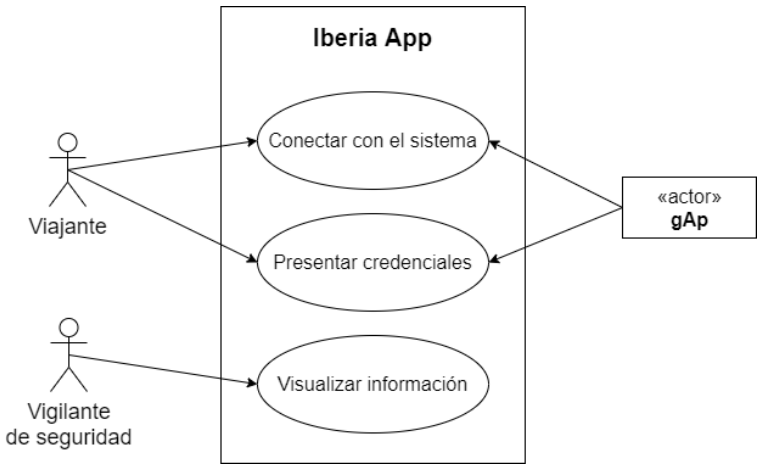

Figura 17: Diagrama de casos de uso de Iberia (MVP)

4.1.2.3 Ciudadano español

Ciudadano español registrado en el sistema sanitario de la Comunidad de Madrid. Esta persona tendrá un agente en forma de aplicación móvil donde almacenará sus DID y sus credenciales. Este agente permitirá gestionar las conexiones y la aceptación y presentación de credenciales.

Inicialmente este agente era una aplicación ofrecida por terceros a través de Google Play y Apple Store. Sin embargo, tal y como se comentó en el Estado del Arte, se tuvo que desarrollar una aplicación móvil a pocas semanas de entregar este TFM. Esta aplicación se ha llamado **Cartera digital SaludMadrid**.

4.1.2.4 Agentes secundarios

Se trata de agentes que no tendrán ni interfaz gráfica ni mucha lógica de negocio asociada. Esto es porque no presentan un papel relevante durante los pasos del escenario. Se recurrirá a ellos para preparar los prerrequisitos del escenario y no volverán a participar. Son los siguientes:

- **Ministerio del Interior** (**MI**). Publica el esquema y la definición de la credencial de identidad. Expide estas credenciales a los ciudadanos. Se supone a un funcionario del estado controlando el agente y con autorización sobre el mismo.
- **Ministerio de Sanidad** (**MS**). Publica el esquema y la definición de la credencial de autorización. Expide estas credenciales a las diferentes comunidades autónomas. Por otra parte, publica los esquemas de credencial de las credenciales COVID, aunque las definiciones de las credenciales no, puesto que esto lo debe hacer cada comunidad autónoma (agente RUC19). Se supone a un funcionario del estado controlando el agente y con autorización sobre el mismo.

En otros países europeos estos agentes podrían ser diferentes, dependiendo de la jerarquía organizacional de cada uno.

4.1.3 Escenario

El escenario cuenta con una serie de **prerrequisitos**:

1. Todos los esquemas y definiciones de credencial han sido publicados en el VDR.
2. El Ministerio de Interior deberá haber expedido una credencial de identidad al ciudadano.
3. El Ministerio de Sanidad deberá haber expedido una credencial de autorización a la Comunidad de Madrid.
4. Los pacientes están ya registrados en el sistema informático sanitario de la Comunidad de Madrid.

Una vez cumplidos estos puntos, el escenario se puede completar sin inconvenientes. Para analizarlo, se va a separar en dos etapas bien diferenciadas.

En la **primera etapa**, el ciudadano español se presenta en un centro sanitario de la CM para realizar vacunarse con una nueva dosis contra la COVID-19 o para realizar un test de antígenos o PCR para detectar la COVID-19, dependiendo de sus necesidades.

Una vez realizada una de las dos acciones anteriores, el administrativo o sanitario correspondiente mostrará un código QR que el ciudadano debe escanear con su aplicación, con el fin de establecer una conexión entre ambos agentes.

Tras establecer la conexión, el ciudadano recibirá automáticamente (sin la participación del sanitario) una petición de presentación de credencial de identidad, en la cual se le exigirá revelar el atributo correspondiente a su número de identidad nacional (número de DNI en el caso de España). El paciente deberá aceptar dicha petición.

El número de identidad permitirá a la aplicación hospitalaria identificar automáticamente al paciente y abrir el espacio de este, donde el sanitario podrá revisar datos de vacunación y de pruebas COVID-19 del paciente. En este espacio, el sanitario registrará la información correspondiente a la vacuna administrada o al test realizado, de forma que esta información se almacenará en el sistema informático.

En otro punto temporal, el sistema informático emitirá la credencial correspondiente. En el MVP esta acción se acometerá cuando un administrativo o sanitario pulse un botón en la interfaz de la aplicación, por fines demostrativos. Esta persona puede ser diferente a la que registró la información. En producción esto se produciría de forma automática durante la noche o cuando el ciudadano solicitase la credencial.

En el caso de las credenciales de pruebas PCR (no antígenos) positivas, cuando se expida una de estas, además se expedirá una credencial de recuperación. El rango temporal que contendrá esta credencial se calculará a partir de la fecha actual más 15 días como fecha de inicio y 15+180 después días como fecha final de validez.

La lógica anterior se ha obtenido de la especificación del esquema oficial [125]. Sin embargo, el alumno cree que en producción está se debería revisar, ya que se ha observado como diferentes variantes de la COVID-19 pueden afectar en estos rangos temporales donde, por ejemplo, las variantes BA.4 y BA.5 pueden llegar a infectar a los 28 días tras una infección previa, según el Ministerio de Salud australiano [129].

El diseño de los esquemas de credenciales realizados, donde se ha evitado incluir lógica variante en el tiempo, permite reutilizar las mismas credenciales para las diferentes variaciones que se produzcan, como el caso mencionado donde el periodo de recuperación variar.

En la **segunda etapa**, un ciudadano español con intención de viajar a otro país de la UE se presenta en la puerta de embarque del aeropuerto, donde tendrá que escanear un código QR presente en un quiosco. El viajero escaneará el código con su aplicación móvil y, automáticamente, el quiosco le mandará una petición de establecimiento de conexión, la cual deberá ser aceptada.

Tras esto, el viajero recibirá varias peticiones de presentación de credencial, una tras otra. La primera le solicitará revelar su foto de su credencial de identidad. Las restantes le solicitarán varios datos de las credenciales COVID-19, con el fin de comprobar que el ciudadano cumple con los requisitos sanitarios.

Tras aceptar las solicitudes, en la pantalla del quiosco aparecerá la foto y un indicativo sobre si cumple o no los requisitos. El vigilante de seguridad comprobará si la foto corresponde con el ciudadano y si cumple con los requisitos para decidir si le deja embarcar en el avión.

Los datos que se solicitan en las peticiones de presentaciones de credenciales COVID-19 son los siguientes:

- Para la **credencial de vacunación**:
  - Revelación de la información: *tg* (agente objetivo).
  - Predicados ZKP: *sd* (número de dosis) ≥ 2, *dt* (fecha) ≥ (fecha actual – 270 días).
- Para la **credencial de test**:
  - Revelación de la información: *tg* (agente objetivo), *tr* (resultado del test).
  - Predicados ZKP: *sc* (fecha) ≥ (fecha actual – 15 días).
- Para la **credencial de recuperación**:
  - Revelación de la información: *tg* (agente objetivo).
  - Predicados ZKP: *df* (fecha inicial) ≤ (fecha actual), *du* (fecha final) ≥ (fecha actual).

Los datos anteriores y la siguiente lógica cambiarán a lo largo del tiempo, con los pacientes y con las enfermedades objetivo.

La lógica que indica si un viajero puede embarcar en el avión o no, es la siguiente:

- Primero se comprueba la **credencial test**:
  - Si el viajero presenta la credencial test con agente objetivo COVID, con resultado positivo y fecha posterior a dos días hacia atrás, entonces impedir el paso.
  - Si no se cumple lo anterior, hay que comprobar la credencial de recuperación.
- Para comprobar la **credencial de recuperación**:
  - Si el viajero presenta la credencial de recuperación con agente objetivo COVID:
    - Si la fecha de inicio es posterior a la actual y la fecha final es anterior a la actual, permitir el paso.
    - Si las fechas de inicio y final son posteriores a la actual, comprobar la credencial de vacunación.
    - Si no se cumplen las dos condiciones anteriores, impedir el paso.
  - Si no se cumple lo anterior, comprobar la credencial de vacunación.
- Para comprobar la **credencial de vacunación**:
  - Si el viajero presenta la credencial de recuperación con agente objetivo COVID:
    - Si el número de dosis es mayor o igual a 2 y la fecha de la última dosis es posterior a la fecha actual menos 270 días, permitir el paso.
    - Si no se cumple la condición anterior, impedir el paso.
  - Si no se cumple lo anterior, impedir el paso.

Por último, indicar que no se han incluido diagramas de secuencia para el análisis de este escenario porque estos son muy similares a los vistos en la Prueba de Concepto para la etapa 2 (Figura 12) y etapa 3 (Figura 13), cambiando únicamente las entidades que participan y aumentando el número de presentaciones intercambiadas.

4.2 Diseño

El sistema SSI a implementar sobre el escenario anterior se compone de 5 agentes, un servidor Tails y un VDR (local o externo). Únicamente se han implementado interfaces gráficas para los tres agentes principales, de las cuales se han diseñado la de RUC19 y la de Iberia.

4.2.1 Arquitectura y tecnologías de desarrollo

En este apartado se van a presentar una serie de figuras de elaboración propia que muestran el diseño de los componentes del sistema y, en conjunto, de la arquitectura de este. En estas figuras se ha omitido el servidor Tails dada su baja relevancia respecto al resto de componentes. Para su correcta compresión, hay que tener en cuenta una serie de detalles: algunas comunicaciones son unidireccionales, los componentes con un icono que recuerda a una base de datos indican que hacen uso de esta, los componentes con el logo de Hyperledger Aries (logo verde) indican que incluyen un framework Aries y los componentes con el logo de Hyperledger Indy (logo azul) indican que funcionan sobre Indy.

Adicionalmente, el controlador general (gAp) desarrollado en la PoC se ha reutilizado en este MVP. En las figuras que aparezca este componente, aparecerá con un fondo azul para poder distinguirlo mejor del resto de componentes.

A continuación, se muestra la **arquitectura general simplificada** del sistema. Nótese que las líneas más gruesas indican comunicaciones seguras basadas en DIDComm.

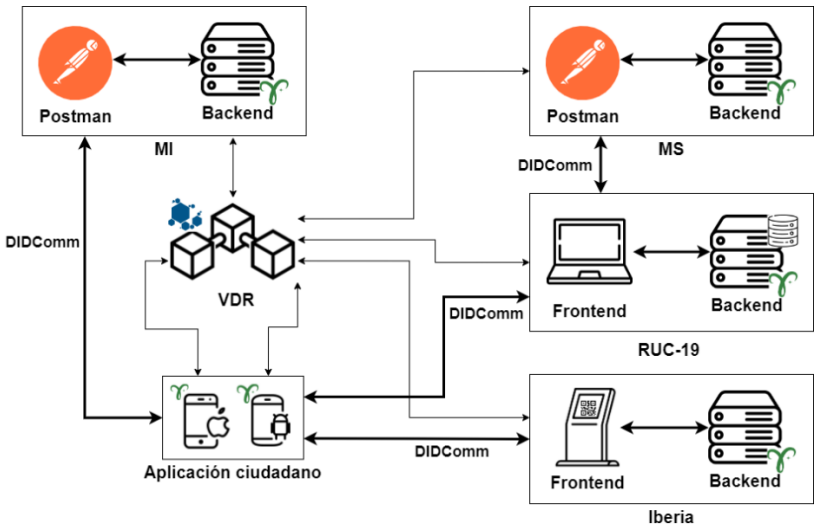

Figura 18: Arquitectura simplificada del MVP

Todos los agentes son de tipo servidor o empresarial, incluyendo a la aplicación del ciudadano, tal y como se comentó en su momento. Estos agentes emplean el framework ACA-Py.

El VDR será una red de Hyperledger Indy. En el caso del MVP, se puede elegir en el despliegue si desplegar una en local mediante el proyecto VON, hacer uso de la red pública de desarrollo (*dev*) de **BCovrin** o hacer uso de la red pública de pruebas (*BuilderNet*) de **Sovrin**. Como ya se mencionó, Sovrin requiere solicitar permiso para publicar transacciones en la red y requiere aceptar un acuerdo TAA.

El servidor Tails se desplegará en local a través del proyecto Indy Tails Server.

Todos los servidores se han implementado con Node.js y Express, junto con el resto de las tecnologías especificadas en el Estado del Arte. En cuanto a las tecnologías frontend, varían en función del agente.

A continuación, se mostrarán las arquitecturas de cada uno de los agentes que componen el sistema.

Arquitectura de los agentes MI y MS:

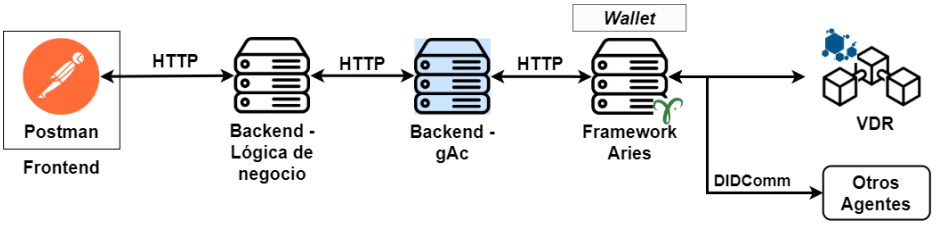

Figura 19: Arquitectura de los agentes MI y MS (MVP)

La arquitectura es muy similar a la empleada para los agentes de la PoC (Figura 15), pero con ciertos matices. Por una parte, se ha separado la lógica de negocio a un servidor propio. Y, por otra parte, se ha eliminado el servidor "mock", de forma que ahora los eventos se muestran por el terminal del propio servidor de la lógica de negocio. Ahora, todos los componentes (salvo el cliente Postman) estarán virtualizados.

En cuanto al frontend, en este caso no se van a emplear interfaces gráficas, ya que estos agentes son secundarios y no participan como tal en el escenario. Por tanto, se usa Postman.

Arquitectura del agente RUC19:

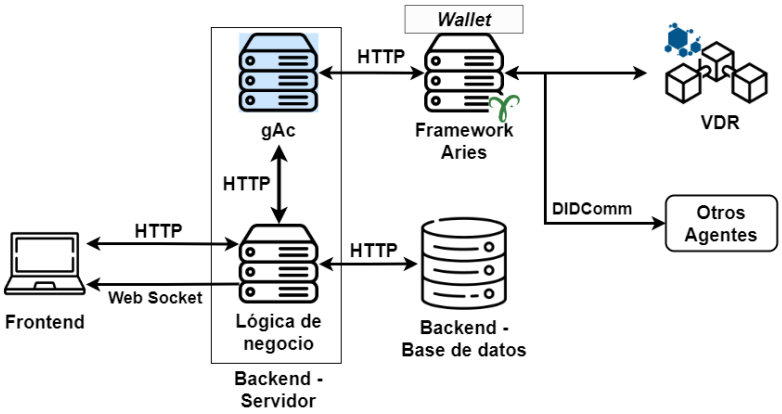

Figura 20: Arquitectura del agente RUC19 (MVP)

Este agente es el más complejo del sistema. Por una parte, se tiene un servidor que contiene lógica de negocio, en mayor cantidad que los otros servidores vistos hasta ahora. Por otra parte, aparece una base de datos que no está presente en el resto de los agentes y que almacena todos los datos sanitarios del sistema informático. Y, por último, el frontend es una interfaz medianamente compleja.

En cuanto al frontend, es una interfaz web desarrollada sobre HTML, CSS, JavaScript y React, junto con el resto de las tecnologías especificadas en el Estado del Arte.

Y, en cuanto a la base de datos, la tecnología que se emplea es MySQL, por los motivos que se mencionaron en el Estado del Arte. Sin embargo, no se llegó a dar las razones de por qué se ha elegido una base de datos relacional en lugar de una base de datos NoSQL, como una base de datos de documentos como puede ser MongoDB.

Por una parte, se han tenido en cuenta los tipos de datos que va a almacenar esta base de datos. En este caso, los datos son datos médicos de ciudadanos (datos personales), lo que significa que son críticos, es decir, la pérdida o inconsistencia de un dato puede generar situaciones desastrosas. Además, este tipo de datos tiene una estructura fija y compleja con relaciones 1-N, y constituyen un tamaño pequeño o mediano, en comparación con datos como los "me gusta" de una red social. Por último, las relaciones entre los datos no tienen tanta importancia como los propios datos. Todas estas características implican que una base de datos relacional es una solución conveniente.

Y, por otra parte, se han tenido en cuenta las necesidades de consistencia, disponibilidad y tolerancia a la partición, con el objetivo de seleccionar las dos más importantes para cumplir con el teorema CAP. En este caso, la base de datos está centralizada y, adicionalmente, debe tener una alta disponibilidad y consistencia. Al igual que antes, esto implica que una base de datos relacional es una solución conveniente.

Arquitectura del agente Iberia:

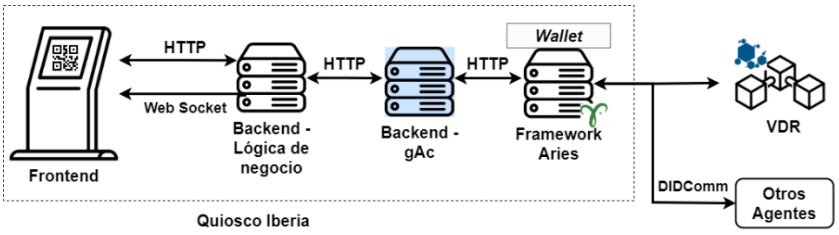

Figura 21: Arquitectura del agente Iberia (MVP)

Este agente contiene también un servidor que incluye lógica de negocio y una interfaz. En cuanto a este servidor, en comparación con el servidor de RUC19, contiene menos rutas, pero la complejidad interna de los controladores es mucho mayor.

En cuanto al frontend, también es una interfaz web desarrollada sobre React y el resto de las tecnologías. Esta vez, sin embargo, es una interfaz mucho más sencilla.

Por último, hay que recordar que todos los componentes del agente estarían desplegados localmente dentro del propio quiosco.

Arquitectura del agente del ciudadano (aplicación móvil):

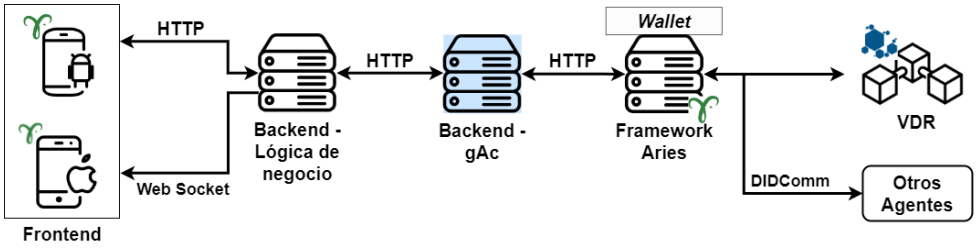

Figura 22: Arquitectura del agente del ciudadano (MVP)

Como ya se ha mencionado, este agente se basa en un framework de servidor (ACA-Py), siguiendo así una arquitectura similar al resto de agentes mostrados. El servidor de lógica de negocio contiene algunas rutas útiles con poca complejidad.

En cuanto al frontend, este es una aplicación móvil desarrollada sobre HTML, CSS, JavaScript y React Native, junto con el resto de las tecnologías vistas en el Estado del Arte.

Originalmente, la arquitectura de este agente se componía de una aplicación móvil obtenida de una tienda de aplicaciones y de un agente mediador, ofrecido por la propia organización de la aplicación.

4.2.2 Diseño de las aplicaciones

4.2.2.1 Aplicación RUC19

El diseño de esta aplicación ha consistido en diseñar la base de datos y la interfaz gráfica.

4.2.2.1.1 Base de datos

El diseño de las tablas que contiene la base de datos es el siguiente (figura de elaboración propia):

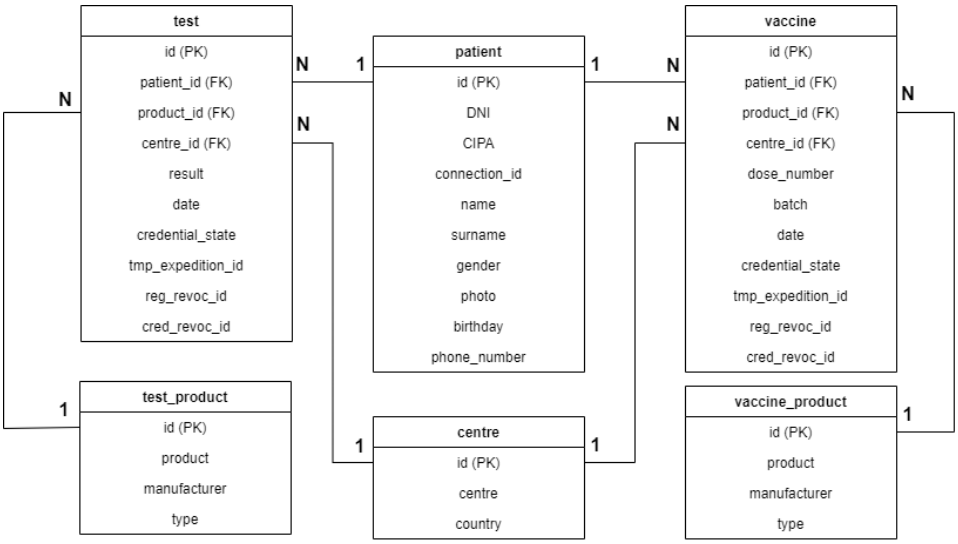

Figura 23: Diseño de tablas de la base de datos de RUC19 (MVP)

Este diseño cumple con la quinta forma normal de normalización y, por tanto, con las anteriores (1FN, 2FN, 3FN, BCFN y 4FN). En el ANEXO IX: Normalización de la base de datos (RUC19) se detalla el proceso seguido para la normalización de esta base de datos.

Se podría mejorar aún más el diseño aplicando el concepto de herencia, creando una única tabla común para los productos COVID, de la cual heredarían las tablas de productos de vacunas y de pruebas. Los atributos comunes estarían en la tabla común y en las tablas heredadas estarían unos identificadores que actuarían como claves primarias y también como foráneas respecto a la tabla común. Sin embargo, esta mejora no se ha llegado a aplicar ya que no supone una diferencia sustancial y añadiría una complejidad extra a las consultas SQL a la base de datos que no se alinean con el objetivo demostrativo del MVP.

Se harán uso de los índices para los campos que contienen claves foráneas, con el fin de optimizar las búsquedas que involucran varias tablas relacionadas.

Para implementar campos como el del género del paciente, el resultado de un test o el tipo de vacuna o test, se pueden realizar de tres formas:

1. Utilizar un VARCHAR como tipo de dato y establecer restricciones para que solo acepte un cierto número de valores. Esto supone almacenar para cada fila una cadena de texto, la cual ocupa más que un tipo numérico.
2. Separar en otra tabla el género / resultado / tipo y establecer como clave primaria un dato numérico. Después se referencia hacia la clave primaria. Esto ahorra espacio, pero aumenta la complejidad y el retardo de las consultas SQL, al tener que hacer JOIN.
3. Utilizar el tipo ENUM, el cual acepta únicamente una serie de valores y almacena un dato numérico con la posición del array. Esto ahorra espacio y evita realizar JOIN. La contraparte es que no es un tipo de dato estándar entre las bases de datos SQL, por lo que quizá no se pueda reproducir con otras tecnologías.

Puesto que la tercera opción es la más eficiente y simple y, además, no incumple ninguna forma normal, se ha decidido que es la mejor solución.

Ahora, se van a detallar el tipo de dato y las restricciones de cada columna de cada tabla.

Tabla "patient". Ciudadano registrado en el sistema informático hospitalario de la CM.

Nombre	Descripción	Tipo (tamaño)	Restricciones	Clave
id	Identificador numérico auto incremental.	INTEGER	UNIQUE NOT NULL AUTO_INCREMENT	PK
DNI[26]	Número de Identidad Nacional Español del paciente.	CHAR(9)	UNIQUE NOT NULL CHECK (DNI REGEXP '^[0-9]{8}[a-zA-Z]$')	-
CIPA	Código de Identificación de Personal Autonómico.	CHAR(10)	CHECK (LENGTH(CIPA) = 10)	-
connection_id	Identificador de la conexión con el agente del ciudadano.	VARCHAR(256)	-	-
name	Nombre del paciente.	VARCHAR(256)	NOT NULL	-
surname	Apellido(s) del paciente.	VARCHAR(256)	NOT NULL	-
gender	Género del paciente.	ENUM ('male', 'female', 'other')	NOT NULL	-
photo[27]	Foto del paciente (URL).	VARCHAR(2083)	-	-
birthday	Fecha de nacimiento del paciente.	DATE	CHECK (birthday > DATE('1900-01-01'))	-
phone_number	Número de teléfono del paciente.	VARCHAR(20)	-	-

Tabla 8: Tabla "patient" de la base de datos RUC19 (MVP)

[26] Aunque en la realidad habría que considerar pacientes registrados con su NIF y que el DNI no contiene ciertas letras del alfabeto, estas situaciones se han decidido omitirlas.

[27] Se establece 2083 como límite dado que Internet Explorer lo establece así [139].

En las dos siguientes tablas, nótese que las fechas se almacenan con hora incluida, pero cuando se expidan las credenciales correspondientes esta se omitirá tal y como se comentó. Además, los campos *tmp_expedition_id*, *reg_revoc_id* y *cred_revoc_id* contendrán valores nulos hasta el momento en el que se expida una credencial. Los dos últimos identificadores son necesarios para poder revocar una credencial

Tabla "vaccine". Dosis de una vacuna COVID-19 administrada a un paciente.

Nombre	Descripción	Tipo (tamaño)	Restricciones	Clave
id	Identificador numérico auto incremental.	INTEGER	UNIQUE NOT NULL AUTO_INCREMENT	PK
patient_id	Identificador que referencia al paciente vacunado.	INTEGER	NOT NULL INDEX	FK ON UPDATE CASCADE ON DELETE SET NULL
product_id	Identificador que referencia al producto de la vacuna.	INTEGER	NOT NULL INDEX	FK ON UPDATE CASCADE ON DELETE RESTRICT
centre_id	Identificador que referencia al centro que administró la vacuna.	INTEGER	NOT NULL INDEX	FK ON UPDATE CASCADE ON DELETE RESTRICT
dose_number	Número de dosis que representa respecto al total.	SMALLINT (8)	NOT NULL CHECK (dose_number > 0)	-
batch	Lote al que pertenece la vacuna administrada.	VARCHAR(256)	NOT NULL	-
date	Fecha y hora de la administración de la vacuna.	DATETIME	NOT NULL CHECK (date >= DATE('1970-01-01'))	-
credential_state	Estado de la credencial.	ENUM ('not issued', 'issued', 'revoked')	NOT NULL	-
tmp_expedition_id	Identificador temporal de expedición de credencial.	VARCHAR(256)	DEFAULT NULL	-
reg_revoc_id	Identificador del registro de revocación.	VARCHAR(256)	DEFAULT NULL	-
cred_revoc_id	Identificador de la credencial expedida respecto al anterior.	VARCHAR(256)	DEFAULT NULL	-

Tabla 9: Tabla "vaccine" de la base de datos RUC19 (MVP)

Tabla "test". Prueba COVID-19 PCR o antígenos realizada a un paciente.

Nombre	Descripción	Tipo (tamaño)	Restricciones	Clave
id	Identificador numérico auto incremental.	INTEGER	UNIQUE NOT NULL AUTO_INCREMENT	PK
patient_id	Identificador que referencia al paciente vacunado.	INTEGER	NOT NULL INDEX	FK ON UPDATE CASCADE ON DELETE SET NULL
product_id	Identificador que referencia al producto de la vacuna.	INTEGER	NOT NULL INDEX	FK ON UPDATE CASCADE ON DELETE RESTRICT
centre_id	Identificador que referencia al centro que administró la vacuna.	INTEGER	NOT NULL INDEX	FK ON UPDATE CASCADE ON DELETE RESTRICT
result	Resultado del test.	ENUM ('Detected, 'Not detected')	-	-
date	Fecha y hora de la administración de la vacuna.	DATETIME	NOT NULL CHECK (date >= DATE('1970-01-01'))	-
credential_state	Estado de la credencial.	ENUM ('not issued', 'issued', 'revoked')	NOT NULL	-
tmp_expedition_id	Identificador temporal de expedición de credencial Aries.	VARCHAR(256)	DEFAULT NULL	-
reg_revoc_id	Identificador del registro de revocación.	VARCHAR(256)	DEFAULT NULL	-
cred_revoc_id	Identificador de la credencial expedida respecto al registro anterior.	VARCHAR(256)	DEFAULT NULL	-

Tabla 10: Tabla "test" de la base de datos RUC19 (MVP)

Tabla "centre". Centro hospitalario donde se ha realizado una prueba o se ha administrado una vacuna.

Nombre	Descripción	Tipo (tamaño)	Restricciones	Clave
id	Identificador numérico auto incremental.	INTEGER	UNIQUE NOT NULL AUTO_INCREMENT	PK
centre	Nombre del centro hospitalario.	VARCHAR(256)	NOT NULL	-
country	País donde se encuentra el centro, formato ISO-3166-1 de 3 letras.	CHAR(3)	NOT NULL CHECK (LENGTH(country) = 3)	-

Tabla 11: Tabla "centre" de la base de datos RUC19 (MVP)

Tabla "vaccine_product". Producto de una vacuna COVID.

Nombre	Descripción	Tipo (tamaño)	Restricciones	Clave
id	Identificador numérico auto incremental.	INTEGER	UNIQUE NOT NULL AUTO_INCREMENT	PK
product	Producto de la dosis adminitrada.	VARCHAR(256)	NOT NULL	-
manufacturer	Fabricante de la vacuna.	VARCHAR(256)	NOT NULL	-
type	Tipo de vacuna administrada.	ENUM ['mRNA', 'antigen', 'viral vector', 'inactivated', 'other']	NOT NULL	-

Tabla 12: Tabla "vaccine_product" de la base de datos RUC19 (MVP)

Tabla "test_product". Producto de una prueba COVID.

Nombre	Descripción	Tipo (tamaño)	Restricciones	Clave
id	Identificador numérico auto incremental.	INTEGER	UNIQUE NOT NULL AUTO_INCREMENT	PK
product	Nombre o identificador del test realizado	VARCHAR(256)	NOT NULL	-
manufacturer	Fabricante del test.	VARCHAR(256)	-	-
type	Tipo de test realizado.	ENUM ['NAAT', 'RAT']	NOT NULL	-

Tabla 13: Tabla "test_product" de la base de datos RUC19 (MVP)

4.2.2.1.2 Interfaz gráfica

La realización de este diseño sirve como guía general para la posterior fase de implementación. Por lo tanto, esto no implica que los colores, la fuente, los tamaños de los elementos u otros detalles no cambien con respecto al resultado final.

El diseño de las interfaces se ha realizado con la herramienta Draw.io [130] y está inspirado en el Registro Unificado de Vacunación COVID-19 [128].

En el ANEXO X: Diseño de la interfaz del agente RUC19 se puede observar el resultado final de este diseño.

4.2.2.2 Aplicación Iberia

En este caso, el diseño ha consistido únicamente en diseñar la interfaz gráfica.

4.2.2.2.1 Interfaz gráfica

Este diseño se ha realizado por los mismos motivos que por los que se ha realizado el diseño de la interfaz gráfica del agente RUC19 y también se ha realizado con la herramienta Draw.io.

En el ANEXO XI: Diseño de la interfaz del agente Iberia se puede observar el resultado final de este diseño.

4.2.2.3 Aplicación Cartera digital SaludMadrid

Inicialmente se tomó la decisión de que este agente fuese una aplicación móvil obtenida de Google Play o de la Apple Store, entre las opciones mencionadas en el apartado 2.3.3.3.2 (Selección del framework para el agente móvil). Esta decisión se tomó teniendo en cuenta la baja prioridad que tenía el desarrollo de un agente móvil frente al resto de componentes del sistema. Sin embargo, tal y como se comenta en ese apartado, ocurrió un imprevisto en las etapas finales del trabajo que imponían la dedicación de un tiempo y esfuerzo considerables del alumno, inclinando entonces la balanza hacia un desarrollo propio.

El tiempo limitado impidió al alumno el poder diseñar la interfaz de esta aplicación, por lo que se decidió pasar a la fase de implementación directamente y desarrollar una interfaz lo más sencilla posible para no sufrir los inconvenientes de no disponer de un diseño con el que guiarse.

4.3 Implementación

Para la fase de implementación se ha decidido virtualizar todos los componentes con Docker. Además, se pretende reutilizar lo máximo posible de la Prueba de Concepto. Esto implica reutilizar el despliegue y el controlador general (gAp), eliminando toda la lógica de negocio de este último. Aunque el comportamiento general del despliegue y del controlador general se mantenga, el funcionamiento interno se ha mejorado considerablemente.

Además de discutir el despliegue y la configuración de los frameworks, en este apartado se va a presentar la implementación para los agentes RUC19, Iberia y Cartera Digital SaludMadrid, los cuales contienen un alto grado de lógica de negocio. En cuanto a los agentes MI y MS, estos únicamente se componen de un servidor (Express) con la lógica de negocio mínima necesaria para cumplimentar los prerrequisitos del escenario, por lo que no se va a profundizar en ellos. En el ANEXO XIV: API del Producto Viable Mínimo (XIV.1 Ministerio del Interior y XIV.2 Ministerio de Sanidad) se detallan las rutas implementadas en estos servidores.

4.3.1 Despliegue

Como se ha mencionado, se va a reutilizar el despliegue de la PoC. Esto implica reutilizar la mecánica de generar un fichero que contenga variables de entorno con un *script* y pasárselo a los servicios Docker, reutilizar el propio *script* en la medida de lo posible y reutilizar los ficheros "Dockerfile" y "docker-compose.yml". Además de modificar estos elementos para adaptarlos al escenario y a la arquitectura del MVP, se pretende mejorarlos lo máximo posible.

4.3.1.1 Docker

Respecto a la PoC, se va a reutilizar el Dockerfile empleado para construir la imagen del controlador general (gAp) visto en el Fragmento de código 2 (Dockerfile gAp PoC), sin necesidad de introducir ningún cambio. En este caso, se va a emplear también para generar las imágenes de los servidores de negocio.

Adicionalmente, se ha creado un Dockerfile para virtualizar las interfaces web de Iberia y de RUC19. Este Dockerfile es exactamente igual que el anterior, pero expone el puerto 3000 en lugar del puerto 5000, ya que el primero es el que emplea React para desplegar su servidor.

En el MVP se han creado varias redes virtuales para interconectar todos los servicios, teniendo una por cada agente. De esta forma, un componente de un agente no podrá interactuar con el componente de otro agente. Se tienen, por tanto, las redes virtuales: *RUC19*, *IBERIA*, *MS*, *MI* y *HOLDER*. Nótese que *HOLDER* hace referencia al ciudadano, de aquí para adelante.

Para exponer los servicios, se han empleado los siguientes puertos libres del anfitrión, donde la "X" será un número diferente en cada agente: 80X0 (servidor de negocio), 80X1 (controlador general), 80X2 (interfaz web), 80X3 (framework), 80X4 (túnel ngrok), 80X5 (administrador del framework) y 80X6 (base de datos MySQL). En algunos agentes no están disponibles algunos de estos servicios.

Al igual que en la PoC, en el MVP se han separado los servicios en diferentes ficheros docker-compose en función del tipo de componente. Por tanto, ahora se tienen los siguientes seis ficheros: "*docker-compose.frameworks.yml*" (frameworks), "*docker-compose.backends.ctrl.yml*" (controladores generales), "*docker-compose.dbs.yml*" (bases de datos), "*docker-compose.backends.bsns.yml*" (servidores de negocio), "*docker-compose.frontends.yml*" (interfaces web) y "*docker-compose.ngrok.yml*" (túneles ngrok).

4.3.1.1.1 Docker Compose – Frameworks

En este fichero están los servicios con los siguientes nombres: "*acapy-RUC19*", "*acapy-IBERIA*", "*acapy-MS*", "*acapy-MI*" y "*acapy-HOLDER*".

En este caso, cada uno de los servicios se declara con prácticamente las mismas instrucciones que en la PoC (Fragmento de código 3: Servicio del framework ACA-Py de la ECDC (PoC)). Sin embargo, el comando indicado ahora se ha cambiado, ya que al incluir más redes Indy, se ha complicado la lógica detrás de este.

Lo que se ha hecho es crear un pequeño *script* y llamarle desde docker-compose a través de la instrucción "*command*". Este *script* se encarga de, en función de las variables de entorno generadas por el *script* principal, decidir si debe registrar un DID público en el VDR automáticamente en el caso de utilizar la red VON local o la red BCovrin pública o si aceptar el acuerdo TAA en el caso de utilizar la red Sovrin pública.

En cuanto a las variables de entorno, en el apartado 4.3.2 (Configuración framework Aries) se verán los cambios introducidos respecto a la PoC.

4.3.1.1.2 Docker Compose – Controladores generales

En este fichero están los servicios de los controladores generales (gAp), cuyos nombres son: "*backend-ctrl-RUC19*", "*backend-ctrl-IBERIA*", "*backend-ctrl-MS*", "*backend-ctrl-MI*" y "*backend-ctrl-HOLDER*".

Al igual que antes, cada uno de estos servicios se declara con las mismas instrucciones que en la PoC (Fragmento de código 4: Servicio del controlador gAp de la ECDC (PoC)), aunque con algunos cambios menores en las variables de entorno que reciben.

4.3.1.1.3 Docker Compose – Bases de datos

En este fichero se encuentra el servicio de la base de datos empleada en el agente RUC19, cuyo nombre es "*db-RUC19*". En este caso, parte de la imagen "*mysql*" [131].

Aunque no se va a detallar la declaración del servicio completo, sí que se va a incluir las variables de entorno y el volumen especificados:

```
environment:
   MYSQL_DATABASE: ${DB_RUC19_DATABASE}
   MYSQL_USER: ${DB_RUC19_USER}
   MYSQL_PASSWORD: ${DB_RUC19_PASSWORD}
   MYSQL_ROOT_PASSWORD: ${DB_RUC19_ROOT_PASSWORD}
volumes:
   ${PWD}/controllers/RUC19/database/ruc19.sql:/docker-entrypoint-initdb.d/init.sql
```

Fragmento de código 8: Parte del servicio de la base de datos de RUC19 (MVP)

Las variables de entorno permiten configurar la base de datos MySQL. La variable "*MYSQL_DATABASE*" permite crear la base de datos al iniciar el servicio, lo cual será imprescindible para poder importar los datos a través de la instrucción que contiene el volumen. El resto de las variables especifican las contraseñas y el nombre de usuario que permiten acceder a la base de datos.

Volviendo al detalle de importar los datos, en el volumen no solo se especifica cual es la carpeta del anfitrión que contiene estos datos, sino que también se especifica un punto de entrada en el contenedor que se encarga de importar los datos en la base de datos MySQL. Es decir, esta instrucción no solo crea un volumen, sino que importa los datos.

4.3.1.1.4 Docker Compose – Servidores de negocio

En este fichero están los servicios de los servidores de negocio, cuyos nombres son: "*backend-bsns-RUC19*", "*backend-bsns-IBERIA*", "*backend-bsns-MS*", "*backend-bsns-MI*" y "*backend-bsns-HOLDER*".

Como se ha comentado, estos servicios emplean imágenes construidas a partir del mismo Dockerfile que emplean los servicios "*backend-ctrl*", puesto que ambos son servidores Node.js desplegados sobre el mismo puerto.

Todos los servicios siguen la misma estructura y contienen las mismas declaraciones. A continuación, se muestra como ejemplo el caso del servicio *backend-bsns-RUC19*:

```
  backend-bsns-RUC19:
     image: ${BACK_BSNS_RUC19_DOCKER_LABEL}
     build:
       context: .
       dockerfile: ./docker/Dockerfile.backend
     container_name: ${BACK_BSNS_RUC19_DOCKER_LABEL}
     hostname: ${BACK_BSNS_RUC19_HOSTNAME}
```

```
    labels:
      name: ${BACK_BSNS_RUC19_DOCKER_LABEL}
      description: "Aries API business backend container of the RUC19 agent"
    environment:
      CTRL_ENDPOINT: ${BACK_CTRL_RUC19_HOST_ENDPOINT}
      MI_ENDPOINT: ${BACK_BSNS_MI_HOST_ENDPOINT}
      MS_ENDPOINT: ${BACK_BSNS_MS_HOST_ENDPOINT}
      WEBHOOK_PORT: ${WEB_RUC19_OUTSIDE_PORT}
      DB_HOSTNAME: ${HOST_HOSTNAME}
      DB_PORT: ${DB_RUC19_OUTSIDE_PORT}
      DB_DATABASE: ${DB_RUC19_DATABASE}
      DB_USER: ${DB_RUC19_USER}
      DB_PASSWORD: ${DB_RUC19_PASSWORD}
    volumes:
      - ${PWD}/controllers/RUC19/backend:/api
    depends_on:
      - acapy-RUC19
    ports:
      - "${BACK_BSNS_RUC19_OUTSIDE_PORT}:${BACK_INSIDE_PORT}"
    networks:
      - RUC19
```

Fragmento de código 9: Servicio del servidor de negocio de RUC19 (MVP)

En las variables de entorno se pueden distinguir los puntos de acceso que va a emplear el servidor para comunicarse con otros componentes. Por una parte, necesita conectarse con el controlador general (servicio *backend-ctrl*) para interactuar con el framework. Por otra parte, necesita tener acceso a los agentes MI y MS que exponen unas rutas que devuelven los esquemas de la credencial de identidad y de las credenciales COVID-19, respectivamente. Estos esquemas los necesita para expedir credenciales. Y, por último, necesita disponer del puerto en el que la interfaz gráfica está disponible, para poder enviarle los eventos que recibe del controlador general.

El resto de las variables de entorno son para poder conectarse con la base de datos, con la que necesitará interactuar.

4.3.1.1.5 Docker Compose – Interfaces web

En este fichero se encuentran los servicios de las interfaces web basadas en React de los agentes RUC19 e Iberia y cuyos nombres son: "*frontend-RUC19*" y "*frontend -IBERIA*". Como se ha comentado, estos servicios emplean imágenes construidas a partir del nuevo Dockerfile añadido.

Los dos servicios siguen la misma estructura y contienen las mismas declaraciones. A continuación, se muestra como ejemplo el caso del servicio *frontend-RUC19*:

```
frontend-RUC19:
    image: ${WEB_RUC19_DOCKER_LABEL}
    build:
      context: .
      dockerfile: ./docker/Dockerfile.frontend
    container name: ${WEB RUC19 DOCKER LABEL}
    hostname: ${WEB RUC19 HOSTNAME}
    labels:
      name: ${WEB RUC19 DOCKER LABEL}
      description: "Webapp for the RUC19"
```

```
    environment:
      REACT_APP_BACK_ENDPOINT: ${BACK_BSNS_RUC19_ENDPOINT}
    volumes:
      - ${PWD}/controllers/RUC19/frontend:/web
    depends_on:
      - acapy-RUC19
    ports:
      - "${WEB_RUC19_OUTSIDE_PORT}:${WEB_INSIDE_PORT}"
    networks:
      - RUC19
```

Fragmento de código 10: Servicio de la interfaz web de RUC19 (MVP)

En este caso solo se tiene una variable de entorno con el punto de acceso al servidor de negocio, con el que necesitará interactuar.

4.3.1.1.6 Docker Compose – Túneles ngrok

En este fichero están los servicios de los túneles ngrok, cuyos nombres son: "*ngrok-RUC19*", "*ngrok-IBERIA*", "*ngrok-MS*", "*ngrok-MI*" y "*ngrok-HOLDER*". En este caso, parte de la imagen "*ngrok*" [117].

Como se puede ver, al contrario que pasaba en la PoC, estos túneles ahora se generan desde sus propios contenedores Docker, evitando así tener que instalar la dependencia en el anfitrión.

4.3.1.2 Automatización con BASH

En el MVP se ha reutilizado el *script* de la PoC, de forma que se ha adaptado para incluir todos los componentes de la arquitectura. El funcionamiento general permanece prácticamente igual, donde los pasos que sigue el comando "*start*" que se vieron en su momento son los mismos. Por tanto, no hace falta volver a repetirlos.

Sin embargo, este *script* se ha mejorado considerablemente de forma interna, mejorando la eficiencia y evitando repetir código (principio DRY). Además, se han añadido más opciones para el despliegue. Por ejemplo, se puede elegir la red Indy a elegir entre las disponibles (local o pública), si no desea comprobar la revocación de las credenciales, qué agente hará de ciudadano (si una aplicación de terceros o la desarrollada por el alumno) o la complejidad de las presentaciones que solicita Iberia. Esto último quiere decir que se puede elegir qué credenciales COVID-19 va a solicitar al ciudadano, desde pedir únicamente la de vacunación hasta pedir todas ellas.

Por otra parte, este *script* permite iniciar las interfaces web y el servidor expo para la aplicación móvil. Y, por último, incluye una opción para realizar de forma automática todas las acciones necesarias para cumplir los prerrequisitos del escenario que se especificaron.

En el ANEXO XII: Ayuda del *script* de despliegue (MVP) se encuentra la ayuda generada por el *script* "**manage**" actualizada para el MVP, donde aparecen todas las opciones nuevas añadidas.

4.3.2 Configuración framework Aries

Para el MVP se han empleado prácticamente las mismas opciones que en la PoC (3.3.2 Configuración framework Aries) para configurar los frameworks. Sin embargo, se han introducido algunos cambios menores.

Primero, en el agente del ciudadano (*WALLET*), se ha activado la opción "*auto-respond-credential-offer*" que permite aceptar automáticamente las ofertas de credencial recibidas. Esta decisión viene dada por motivos de simplificación.

Luego, en los agentes que actúan de emisor (MI, MS, RUC19) se ha añadido la opción "*accept-taa*" para los despliegues que utilicen la red Sovrin. Esta opción permite aceptar automáticamente el acuerdo TAA por cada transacción que publique en el VDR, evitando tener que preguntar al usuario si quiere aceptar el usuario todas las veces. En esta opción se debe incluir la versión del acuerdo a aceptar y el mecanismo de aceptación. En el caso de esta MVP y de la red Sovrin, se ha tomado la decisión de aceptar la última versión del acuerdo (2.0) y de usar un mecanismo que permite que la aceptación sea válida durante la sesión.

Por último, respecto a la opción "*auto-respond-presentation-request*", esta se ha activado para todos los agentes salvo para el del ciudadano, de forma que en estos agentes las solicitudes de presentación de credenciales se responden automáticamente con la propia presentación.

4.3.3 Aplicación RUC19

4.3.3.1 Base de datos

A partir del diseño de las tablas de la base de datos realizado anteriormente, la implementación ha sido inmediata. Para esta implementación, se ha empleado el lenguaje de definición de datos de SQL (SQL-DDL). Las declaraciones de las tablas se encuentran en el ANEXO XIII: Tablas de la base de datos (SQL-DDL).

Para que el MVP se sintiese un caso de validación, el alumno decidió que era importante rellenar las tablas de la base de datos con información con sentido. De esta forma, las tablas visuales de la aplicación que contuviesen pacientes, vacunas o pruebas no estuviesen vacías. Aunque una opción era rellenarlas a mano, se vio que para ciertas tablas esto iba a ser un trabajo repetitivo y vacío. Por tanto, se creó un *script* basado en Node.js para rellenar automáticamente estas tablas a través del lenguaje de manipulación de datos de SQL (SQL-DML), específicamente, de la operación INSERT.

Para esto, se empezó creando primero un JSON con los datos de las tablas "*centre*" y "*vaccine_product*", ya que solo existen unos pocos centros hospitalarios en la Comunidad de Madrid y productos de vacunas oficiales. El *script* recoge los valores del JSON y los inserta en las tablas.

Para el caso de los productos de las pruebas (tabla "*test_product*"), la base de datos oficial [132] contiene cientos de ellos, descartando entonces el relleno manual de estos. Lo que se hizo entonces fue hacer uso de la biblioteca *axios* para recuperarlos todos ellos a través de la API que expone la base de datos oficial. Antes de insertarlos en la tabla, era necesario limpiar estos datos. Este

proceso de limpiado consistía en eliminar productos repetidos y "escapar" las comillas simples, ya que MySQL hace uso de estar para separar valores en una sentencia INSERT.

Para la generación de los datos de los pacientes (tabla "*patient*") se empleó la biblioteca "*faker*" [133], la cual permite generar datos realistas en varios idiomas. Se ha empleado para generar datos como el género, el nombre, el DNI, el domicilio, la foto o la fecha de nacimiento.

Para la generación de los datos de las vacunas (tabla "*vaccine*") y las pruebas (tabla "*test*"), se ha empleado "*faker*" para la generación de datos como las fechas. Además, para las claves foráneas, se han elegido de forma aleatoria entre todas las claves primarias para el caso de los centros, los pacientes y los productos.

Con todo esto se ha conseguido generar una base de datos realista con más de 2.000 filas, todo esto con un esfuerzo relativamente bajo.

4.3.3.2 Backend (servidor de negocio)

El servidor de negocio del agente RUC19 se ha implementado empleando las mismas tecnologías principales y la misma estructuración del código que el controlador general (gAp) de la PoC (3.3.3 Backend).

Las rutas de este servidor de negocio se encuentran detalladas en el ANEXO XIV: API del Producto Viable Mínimo (XIV.3 RUC19), las cuales se pueden agrupar en las siguientes categorías:

- Definiciones de credencial. Son las rutas que permiten publicar en el VDR y recuperar las definiciones de credencial COVID-19 (vacunación, test, recuperación).
- Pacientes. Son las rutas que permiten recuperar a los pacientes almacenados en el sistema.
- Vacunas COVID de pacientes. Son las rutas relativas a la creación y gestión de las vacunas asociadas a un paciente.
- Pruebas COVID de pacientes. Son las rutas relativas a la creación y gestión de las pruebas asociadas a un paciente.
- Otras. Variedad de rutas con sus propios propósitos, como la recuperación de los productos de las vacunas o la aceptación de una conexión segura basada en DIDComm.

Estas rutas son gestionadas por unos controladores que se ayudan de una serie de servicios creados para este servidor, de los cuales destacan tres de ellos: servicio de interacción con la base de datos, servicio de establecimiento de conexión y servicio de gestión de credenciales.

El **servicio de interacción con la base de datos** se encarga de configurar el acceso a la base de datos a partir de las variables de entorno recibidas en Docker y de realizar llamadas a la base de datos con las declaraciones SQL-DML indicadas. Para esto, se ayuda de la biblioteca *mysql2*.

El **servicio de establecimiento de conexión** se encarga de establecer una conexión con el paciente e identificarlo. Para esto, el proceso se compone de tres etapas:

1. Aceptación. Cuando se recibe un evento de solicitud de establecimiento de conexión, el servicio la acepta automáticamente.

2. Identificación. Cuando se recibe un evento de conexión establecida correctamente, el servicio le solicita al ciudadano una presentación de la credencial de identidad que le pide revelar el número de DNI.
3. Asociación. Cuando se recibe la presentación, el servicio busca en la base de datos el paciente (tabla *patient*) a través de su DNI, y entonces almacena en la fila el identificador de la conexión creada entre el paciente y el agente RUC19.

Gracias a este servicio, el enfermero no tendrá que interactuar cuando el paciente escanee el código QR de invitación y solicite el establecimiento de la conexión.

El **servicio de gestión de credenciales** permite realizar dos operaciones:

- Expedición de credenciales. Interactúa con el controlador general (gAp) para expedir una credencial. En el proceso, se encarga de recuperar el identificador temporal de expedición y los identificadores de revocación para almacenarlos en la tabla correspondiente (tabla *vaccine* o tabla *test*). Además, actualiza el campo *credential_state* para indicar que el estado de la credencial es *issued* (credencial expedida).
- Revocación. Recupera los identificadores de revocación de la tabla correspondiente y los emplea en la interacción con el controlador general (gAp) para revocar la credencial. Además, actualiza el campo *credential_state* con el valor *revoked* (credencial revocada).

Por último, este servidor de negocio se encarga de generar y enviar al cliente los siguientes eventos a través del socket implementado:

- Evento de error. Este evento indica al cliente si se ha producido un error y cuál. El cliente se encargará de mostrarlo por pantalla.
- Evento de paciente conectado. Este evento se genera cuando la conexión se ha establecido correctamente y le sirve al cliente para transitar a la pantalla del espacio del paciente desde la pantalla de invitación a través de un código QR. Para que el cliente sepa qué paciente debe mostrar, el evento incluye el número de DNI de este.
- Evento de credencial expedida. Este evento permite actualizar la tabla donde se muestran las vacunas o las pruebas, dependiendo de qué credencial se haya expedido, para que muestre un botón de revocar en lugar de un botón de expedir.
- Evento de credencial revocada. Similar al anterior, pero en el caso de revocación.

4.3.3.3 Frontend

Como se ha comentado en el apartado 2.4.1 (Tecnologías frontend), se hacen uso de las tecnologías HTML, CSS y JavaScript para implementar una interfaz web y de Bootstrap y React como bibliotecas principales. Para la gestión del estado global de React se emplea *context* y los *hooks* que ofrece la biblioteca, y para la navegación entre páginas se emplea la biblioteca *react-router-dom*. Adicionalmente, se usan las bibliotecas *grid.js*, *react-select*, *react-qr-code* y *http-proxy-middleware*.

En el ANEXO XV: Capturas de la interfaz del agente RUC19 se muestran las capturas de la interfaz web implementada.

En todos los clientes React desarrollados en este TFM existen dos carpetas principales y dos ficheros importantes. Respecto a las carpetas, se tiene una carpeta "*/public*", que contiene todo el material estático (ficheros .html, imágenes, etc.), y una carpeta "*/src*", que contiene el código React. Respecto a los ficheros, se tiene un fichero llamado "*package.json*", que tiene la misma función que la descrita para el caso de los servidores, y un fichero "*/src/index.js*" que contiene la lógica que permite renderizar el código JavaScript de React en una etiqueta HTML.

Adicionalmente, todos los clientes React de este TFM tienen la misma estructuración de carpetas y código bajo la carpeta "*src*". El fichero "*App.js*" contiene el componente React raíz, cuyo objetivo es manejar estados React y renderizar otros componentes. La carpeta "*routes*" contienen los ficheros que especifican las diferentes rutas de la aplicación. La carpeta "*context*" contiene la lógica que permite crear un estado global a través de *hooks* y *context*. La carpeta "*components*" contiene una serie de componentes reutilizables de apoyo. La carpeta "*pages*" contiene los componentes principales, donde cada uno de estos representa el componente que se renderiza en cada una de las rutas definidas, haciendo uso de los componentes de apoyo para ello. La carpeta "*styles*" contiene todos los ficheros CSS que permiten definir los estilos de la interfaz. Y, por último, la carpeta "*config*" contiene una serie de ficheros de configuración.

Por otra parte, se emplea la biblioteca *socket.io-client* para configurar el socket que gestiona los eventos recibidos del servidor a través de un fichero llamado "Socket.js".

En este cliente, se definen las rutas con *react-router-dom* en un fichero de la carpeta "*routes*", donde se tienen las rutas para las páginas "*Home*", "*Connection*" y "*Patient*". Estas definiciones incluyen la importación de los componentes con el mismo nombre bajo la carpeta "*pages*".

Por tanto, el componente "*App.js*" importa los ficheros de rutas y del socket, permitiendo así renderizar un componente u otro en función de la ruta y gestionar el estado global en función del evento recibido, respectivamente.

Por último, aunque no se van a listar todos los componentes que se han creado ni la lógica interna de estos, sí que hay que mencionar que se ha seguido un enfoque de partición de la interfaz en componentes pequeños y reutilizables para mejorar la operación de estos.

4.3.4 Aplicación Iberia

4.3.4.1 Backend (servidor de negocio)

El servidor de negocio del agente Iberia se ha implementado empleando las mismas tecnologías principales y la misma estructuración del código que el controlador general (gAp) de la PoC (3.3.3 Backend).

En este servidor se ha añadido una caché sencilla a través de la biblioteca *node-cache*. El uso principal de esta es el de almacenar el estado del proceso de envío y recepción de presentaciones de credencial junto con una clave formada a partir del identificador de conexión. Cuando se inicia el proceso, se crea la tupla con el estado inicial, y cuando el proceso acaba, por cualquier motivo, el valor del estado cambia al estado final.

Este servidor incluye una única ruta que permite aceptar conexiones automáticamente, la cual está detallada en el ANEXO XIV: API del Producto Viable Mínimo (XIV.4 IBERIA).

La complejidad de este servidor se centra en la gestión de eventos entrantes a través de varios servicios implementados, de los cuales destacan cuatro de ellos: servicio de establecimiento de conexión, servicio de gestión de temporizadores, servicio de solicitación de presentación y servicio de comprobación de presentaciones.

El **servicio de establecimiento de conexión** se encarga de aceptar automáticamente una solicitud de establecimiento de conexión tras la recepción del evento correspondiente.

El **servicio de gestión de temporizadores** se encarga de crear y eliminar temporizadores asociados a un identificador que funcionan entre eventos, a través de la biblioteca *named-timers*. De esta forma, se puede crear un temporizador con un nombre durante la ejecución de un proceso y, más adelante, con la ejecución de otro proceso, eliminar el temporizador. Si un temporizador llega a cero, se modifica el estado de la caché a su estado final. El objetivo del temporizador es acabar automáticamente con el proceso si el viajero no presenta alguna credencial solicitada. Este servicio lo utilizan los dos servicios siguientes.

El **servicio de solicitación de presentaciones** define las presentaciones que deben solicitarse al ciudadano y expone una función por cada una de ellas, de forma que cuando se llama a una, se envía la solicitud de la presentación correspondiente al ciudadano. Cuando se envía una solicitud, hace uso del servicio de gestión de temporizadores para iniciar uno con un identificador compuesto por el identificador de la conexión y el nombre de la presentación solicitada.

El **servicio de comprobación de presentaciones** se encarga de procesar las presentaciones recibidas y de emplear el servicio de solicitación de presentaciones para cumplir con la lógica definida en el apartado 4.1.3 (Escenario). En el siguiente diagrama de elaboración propia se refleja la misma lógica, pero desde un punto de vista más técnico, donde se parte desde el punto en el que se recibe una presentación de credencial de identidad válida y se finaliza en uno de los siguientes estados: “impedir paso” o “permitir paso”.

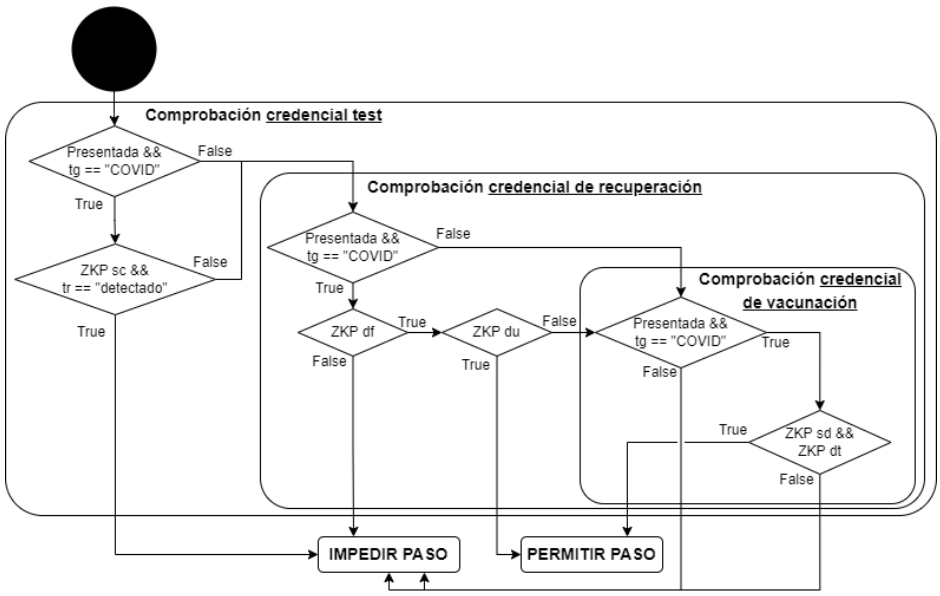

Figura 24: Lógica de presentaciones de credenciales COVID-19 (MVP)

Nótese que los nombres de los atributos son los vistos en los esquemas en el análisis del escenario, que los rombos implican condiciones con dos posibles salidas (*True* o *False*) y que la nomenclatura "*ZKP sc*" significa que se ha cumplido el predicado ZKP sobre el atributo *sc*.

Cuando recibe una presentación del ciudadano, lo primero que hace es eliminar el temporizador asociado a la presentación a través de su identificador. Luego, comprueba si el estado de la caché sigue siendo el estado inicial, porque si ha pasado a su estado final, se para el proceso. Tras esto, comprueba si cumple con los requisitos de la lógica de negocio y comprueba la validez de la presentación (credencial verificada correctamente, credencial no revocada, se han revelado todos los atributos solicitados y se cumplen todos los predicados ZKP). Si una presentación es válida, hace uso del servicio de solicitación de presentaciones para solicitar la siguiente. Y si la presentación recibida es la última, el temporizador se ha acabado o la presentación no es válida, el estado de la caché se modifica a su estado final.

Por último, este servidor de negocio se encarga de generar y enviar al cliente los siguientes eventos a través del socket implementado:

- Evento de error. Este evento indica al cliente si se ha producido un error y cuál. El cliente se encargará de mostrarlo por pantalla.
- Evento de viajero conectado. Este evento se genera cuando la conexión se ha establecido correctamente y le sirve al cliente para transitar a la pantalla del resultado final desde la pantalla de invitación a través de un código QR.
- Evento de revelación de foto. Este evento se genera cuando se recibe la presentación de la credencial de identidad con la foto del viajero y le sirve al cliente para mostrarla por pantalla.
- Evento de resolución final. Este evento se genera cuando el estado de la caché pasa a su estado final, de forma que se ha resuelto si el viajero está autorizado o no para embarcar en el avión. El cliente muestra una información u otra en la pantalla del resultado final en función del contenido de este evento.

4.3.4.2 Frontend

Este frontend se basa en una interfaz web React que emplea las mismas tecnologías y la misma estructuración del código que el cliente de la aplicación RUC19. Sin embargo, en esta aplicación no se han empleado las bibliotecas *react-router-dom*, *react-select* y *grid.js*.

Es importante fijarse en que no se usa la biblioteca *react-router-dom*, lo que implica que no se han definido rutas en el cliente. Esto es así porque, al tratarse de una aplicación muy sencilla con dos únicas páginas, es más eficiente tener un único componente principal que renderiza condicionalmente un componente u otro en función de un estado de React.

En el ANEXO XVI: Capturas de la interfaz del agente Iberia se muestran las capturas de la interfaz web implementada.

Por último, se ha mejorado la interfaz respecto al diseño para incluir en la página *Result* dos agujas giratorias (*spinners*) a través de la biblioteca *react-spinners* [134]. Estos aparecen inicialmente, tanto a la izquierda donde debería aparecer la foto, como a la derecha donde debería

aparecer la decisión final. Según se vayan recibiendo los eventos correspondientes del servidor, se van renderizando en lugar de los *spinners*.

4.3.5 Aplicación Cartera digital SaludMadrid

4.3.5.1 Backend (servidor de negocio)

El servidor de negocio del agente Cartera digital SaludMadrid se ha implementado empleando las mismas tecnologías principales y la misma estructuración del código que el controlador general (gAp) de la PoC (3.3.3 Backend).

Las rutas de este servidor de negocio se encuentran detalladas en el ANEXO XIV: API del Producto Viable Mínimo (XIV.5 Cartera digital SaludMadrid), las cuales se pueden agrupar en las siguientes categorías:

- Conexiones. Son las rutas que permiten crear y gestionar conexiones.
- Cartera. Son las rutas que permiten recuperar y eliminar credenciales.
- Presentación. En este caso es una única ruta que permite enviar una presentación al agente indicado.

Estas rutas son gestionadas por unos controladores cuya lógica principal se basa en interactuar con el controlador general (gAp). Adicionalmente, se encargan de traducir atributos y hacerlos legibles de cara a un humano. Por ejemplo, en lugar de enviar el nombre de atributo "*tg*", envía el texto "*Agente objetivo*".

Por último, este servidor de negocio se encarga de generar y enviar al cliente los siguientes eventos a través del socket implementado:

- Evento de error. Este evento indica al cliente si se ha producido un error y cuál. El cliente se encargará de mostrarlo por pantalla.
- Evento de nueva conexión. Este evento se genera cuando la conexión se ha establecido correctamente y le sirve al cliente para actualizar la lista de conexiones.
- Evento de nueva credencial. Este evento se genera cuando se ha recibido una credencial expedida por un emisor y le sirve al cliente para actualizar la lista de credenciales.
- Evento de solicitud de presentación. Este evento se recibe una solicitud de presentación de credencial y le sirve al cliente para notificar al ciudadano y permitirle aceptar o denegar dicha presentación.

4.3.5.2 Frontend

Este frontend se basa en una aplicación móvil React Native que emplea las prácticamente las mismas tecnologías que los clientes del agente RUC19 y del agente Iberia, además de una estructuración del código similar.

Respecto a las tecnologías, la principal diferencia reside en que en lugar de usar React (bibliotecas *react* y *react-dom*) se emplea React Native (bibliotecas *react* y *react-native*). Esto implica que en lugar de rutas se tienen pantallas, por lo que se ha considerado más adecuado utilizar la biblioteca *react-navigation* en lugar de *react-router-dom* para la navegación. Otra implicación es que Bootstrap no se puede usar con React Native, ya que Bootstrap genera

componentes basados en etiquetas HTML. Por otra parte, se ha empleado Expo para facilitar el desarrollo, haciendo uso del terminal físico del alumno para ver y probar los cambios realizados.

Luego, las bibliotecas *grid.js*, *react-select*, *react-spinners*, *react-qr-code* y *http-proxy-middleware* no tienen uso en esta aplicación, por lo que no se han utilizado. A cambio, se ha empleado la biblioteca *expo-barcode-scanner* para escanear códigos QR.

Respecto a la estructuración del código, las diferencias son menores y son impuestas por la tecnología React Native. Primero, la carpeta "*/public*" se llama "*/assets*" y no contiene ficheros ".html". Luego, la carpeta "*pages*" ahora se llama "*screens*", lo cual identifica mejor al caso de una aplicación móvil. Y, por último, la carpeta "*styles*" ahora contiene ficheros JavaScript con las reglas CSS, en lugar de ficheros CSS. Adicionalmente, se ha creado una carpeta "*hooks*" que contiene algunos *hooks* creados a mano, como los que gestionan eventos o el que pide permisos de uso de cámara al usuario.

En el ANEXO XVII: Capturas de la interfaz del agente Cartera Digital SaludMadrid se muestran las capturas de la aplicación móvil implementada.

Para la navegación entre pantallas con *react-navigation* se ha creado una navegación basada en pestañas para las pantallas *Home*, *Connections* y *Credentials*. Adicionalmente, se han creado navegaciones basadas en pilas de pantallas para las pantallas *Presentation* (desde *Home*), *Scan* (desde *Connections*) y *Credential* (desde *Credentials*).

Luego, los eventos que se reciben del servidor se muestran en forma de notificaciones en la pantalla *Home* que se pueden descartar, donde las notificaciones de solicitud de envío de presentación muestran un botón adicional para aceptar o denegar dicha solicitud.

En producción, las funcionalidades que implementa esta aplicación deberían añadirse en la aplicación "Tarjeta Sanitaria" de la Comunidad de Madrid.

Capítulo 5. CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

Tras finalizar el proyecto, se puede observar que el alumno ha conseguido cumplir los objetivos marcados inicialmente, salvo el último, que trataba de elaborar un conjunto de conclusiones. Este último objetivo se va a cubrir en este apartado.

5.1.1 Resumen de los objetivos logrados

La introducción y el Estado del Arte han permitido al alumno aprender sobre los diferentes conceptos de la identidad autosoberana, sobre el contexto regulatorio y sobre las diferentes tecnologías que permiten implementarlas, en concreto Hyperledger Indy y Hyperledger Aries.

Tras esto, el alumno se decantó por dividir el proceso de desarrollo en dos fases, una primera con una Prueba de Concepto (PoC) y una segunda con un Producto Viable Mínimo (MVP), de forma que en la primera fase se desarrollasen los componentes principales del sistema y en la segunda este se extendiese para poder aplicarlo a un escenario sanitario realista centrado en la pandemia COVID-19 para validarlo. De esta forma, se ha podido aislar en una fase toda la dificultad centrada en las herramientas Hyperledger y los conceptos de SSI, y en la otra fase toda la lógica de negocio y los componentes necesarios para integrarla (servidores de negocio y clientes).

La Prueba de Concepto ha permitido al alumno probar el potencial de la identidad autosoberana y la madurez de las tecnologías Hyperledger a través del diseño e implementación de un sistema de identidad autosoberana rudimentario sobre un escenario sencillo y poco realista.

El Producto Viable Mínimo ha permitido al alumno validar el sistema anterior tras diseñar e implementar un sistema más completo que el generado en la Prueba de Concepto que ha reutilizado la mayor cantidad posible de componentes de este. La validación también ha contado con un análisis de un escenario realista, el cual ha sido sobre el que se ha diseñado e implementado el sistema anterior. Para este análisis, se ha necesitado analizar el certificado COVID digital actual.

5.1.2 Conclusiones respecto a la identidad autosoberana

Tras la realización de este TFM, el alumno ha conseguido profundizar en los conceptos de la identidad autosoberana y aplicarlos a un sistema de identidad descentralizada en el ámbito sanitario europeo. Durante el proceso, se ha podido comprobar que el sistema no necesita de una autoridad certificadora (CA) para funcionar correctamente gracias a la confianza criptográfica ofrecida por las diferentes conceptos criptográficos vistos y por el uso de la tecnología blockchain, ahorrando así los costes que llevan asociadas las infraestructuras centralizadas.

Por un lado, se ha visto como las conexiones basadas en DIDComm permiten aumentar la seguridad en el intercambio de información entre dos entidades. Y, por el otro lado, se ha visto como las credenciales verificables (VC) dificultan la suplantación de identidad, reducen el

impacto de las filtraciones de datos masivas, mejoran la privacidad de los ciudadanos gracias a la revelación selectiva y a la aplicación de ZKP.

En cuanto al cumplimiento de la RGPD, se puede observar cómo estos conceptos permiten cumplir de forma más eficiente con el principio de minimización de datos, solicitar el derecho a la supresión de forma segura, mejorar la portabilidad de los datos y aumentar la seudonimización a través de los identificadores descentralizados (DID) y la aplicación de ZKP sobre los identificadores del titular.

Adicionalmente, se ha visto como la implementación de cadenas de autorización basadas en credenciales verificables permiten a un verificador que no conoce a un emisor averiguar si este es de confianza.

A pesar de todas las ventajas que ofrece la identidad autosoberana, existen ciertos inconvenientes de aplicar esta sobre una identidad centralizada basada en PKI.

Por una parte, se debe tener especial cuidado en el primer paso para establecer una conexión entre dos agentes, ya que este se basa en presentar una invitación por medios ajenos al protocolo DIDComm, como la presentación de un código QR o de una URL. Por ejemplo, imaginemos que el metro de una ciudad introduce el uso de credenciales verificables para el uso del abono mensual, de forma que la recarga se realiza en las máquinas de compra de billetes que suele haber antes de los torniquetes. Para comprarlo, antes el ciudadano deberá establecer una conexión con el sistema informático del metro. Para esto, lo más probable es que la máquina presente un código QR en la pantalla, permitiendo así al ciudadano escanear la invitación. El problema aparece en este momento, ya que existe la posibilidad de que un atacante haya superpuesto una pantalla propia a la pantalla de la máquina del metro, de forma que la conexión se estableciese con el agente atacante. En general, los casos que se pueden imaginar en los que se puede producir algo parecido son enrevesados y difíciles de acometer, pero es un riesgo que debe considerarse.

Y, por otra parte, sin un apoyo gubernamental la identidad autosoberana no se puede integrar realmente en la sociedad. Sí que se pueden pensar casos concretos y de pequeño alcance, pero el máximo esplendor de esta identidad se alcanza cuando las instituciones públicas la integran. Las buenas noticias son que algunos gobiernos están generando iniciativas y adaptando regulaciones.

El gobierno que quizá destaque más en este sentido es el de British Columbia (Canadá), que ha creado una comunidad para el desarrollo de diferentes proyectos relacionados, donde alguno de ellos se ha extendido enormemente en la comunidad global de identidad autosoberana (por ejemplo, el proyecto VON).

La Unión Europea también se está involucrando considerablemente. Primero, la propuesta "eIDAS 2.0" acerca la regulación eIDAS a la identidad descentralizada. Luego, el proyecto EBSI incluye varios casos de uso de aplicación de la identidad autosoberana. Y, por último, la Unión Europea está apostando por la creación e integración de una cartera digital [135].

Adicionalmente, actualmente ya existen empresas emergentes donde su modelo de negocio se basa en ofrecer un sistema de identidad autosoberana a diferentes organizaciones, como es el caso de Trinsic o Gataca.

Por tanto, aunque se necesite una fuerte contribución de los gobiernos para implementar una identidad autosoberana en la sociedad, se puede observar que ya hay unos buenos indicadores de que esto está pasando. Sin embargo, aún hay que esperar a que grandes compañías con un modelo de negocio centrado en el tratamiento de los datos de sus usuarios se posicionen, como puede ser el caso de Google o Facebook. Aun así, el alumno cree que, aunque estas compañías presenten inicialmente ciertos obstáculos, al final tendrán que adaptarse a las regulaciones impuestas por la Unión Europea y otros gobiernos, al igual que pasó con la RGPD.

5.1.3 Conclusiones respecto a la madurez del *stack* de identidad Hyperleder

En el apartado anterior se han visto que las conclusiones que el alumno ha elaborado sobre la identidad autosoberana son, en general, favorables, aún no ha comentado nada sobre la madurez de las tecnologías que permiten su implementación. En este sentido, se puede adelantar que el alumno considera que aún falta tiempo para que estas alcancen un nivel de madurez necesario para su implementación completa en la sociedad. Esta implementación completa implica una integración donde se prescinda de la alternativa (PKI). Aun así, el alumno cree que el nivel de madurez es suficiente para iniciar el proceso de implantación. A continuación se mencionarán los motivos que han llevado al alumno a llegar a esta conclusión.

Primero, se ha visto como las credenciales basadas en JSON-LD no permiten la aplicación de predicados ZKP ni, en algunos casos, de la revelación selectiva de atributos. Y las credenciales AnonCreds no permiten predicados ZKP con lógica compleja, ya que únicamente permiten comparar (operaciones $<, \leq, >$ y $\geq$) un atributo numérico con un valor límite. Sería interesante que las credenciales verificables soportasen una lógica con operaciones más complejas y con lógica compuesta. Por ejemplo, probar que un atributo de texto contiene una cadena de texto y que un atributo numérico es mayor a un valor dado, a la vez (operación “AND”).

Luego, en cuanto a las credenciales AnonCreds, sería interesante que los esquemas de credenciales permitan estructuras complejas de datos, de forma que un atributo pudiese contener un *array* de atributos o formar un objeto con otros atributos. Alternativamente, se podría incluir la opción de encadenar esquemas, de forma que un atributo de un esquema permita incluir otro esquema (por ejemplo, un atributo llamado “multas” de una credencial verificable asociada a un carnet de conducir pueda incluir las credenciales verificables de las multas recibidas). Esta estructuración compleja facilitaría enormemente la implementación de las solicitudes de presentaciones de credencial que un verificador debe construir. Adicionalmente, las credenciales AnonCreds podrían permitir incluir ficheros en las propias credenciales. Por ejemplo, una foto en una credencial de identidad.

Por otra parte, el alumno ha tenido que lidiar constantemente con un problema importante. Cuando una solicitud de presentación enviada por un verificador a un titular contiene varios atributos a revelar y varios predicados ZKP, puede pasar que la credencial que quiera presentar el titular no cumpla un único predicado. El problema viene cuando el titular no puede enviar la presentación solicitada, indicando que un predicado no se cumple. Es decir, si un único predicado falla, todo falla. Aunque es posible que esta situación haya sido una decisión tomada

explícitamente por los desarrolladores, el alumno cree que limita enormemente la aplicación en algunos escenarios y que sería más conveniente poder enviar una presentación indicando qué predicado ha fallado.

Para lidiar con este problema, el alumno ha tenido que programar el envío de muchas presentaciones pequeñas, donde se pruebe un único predicado a la vez, en lugar de enviar una única presentación completa. Por ejemplo, en el caso del agente Iberia (MVP), este solicita al viajero seis presentaciones (identidad, test, vacunación, revelación selectiva de la de recuperación, un predicado ZKP de la de recuperación y otro predicado ZKP de la de recuperación) para poder cubrir todos los escenarios. Si se indicase qué predicado ha fallado, Iberia únicamente necesitaría solicitar una sola presentación que cubra todas las credenciales. Como se puede apreciar, esta situación dificulta el desarrollo y, sobre todo, empeora la experiencia del usuario.

Y, por último, en cuanto a la madurez de las tecnologías empleadas, el alumno ha observado ciertos problemas de compatibilidad entre frameworks y con otras tecnologías.

Todas estas limitaciones suponen una dificultad de implementación completa en la sociedad, pero sí que permiten iniciar el proceso de implementación.

En cuanto a otras tecnologías, el alumno cree que en un futuro próximo las implementaciones de identidad autosoberana en la Unión Europea deberían hacerse sobre el proyecto EBSI, en el momento en el que este alcance un nivel de madurez mayor. Esto se debe a que se trata de un proyecto europeo que está alineado con las regulaciones europeas.

5.1.4 Conclusiones respecto al caso de validación

El diseño e implementación del Producto Viable Mínimo ha servido para validar el sistema de identidad autosoberana en el ámbito sanitario europeo, específicamente, en el ámbito de la pandemia COVID-19. La razón por la cual el alumno considera que se ha validado correctamente es porque se ha visto como el sistema ha podido desarrollar el escenario analizado de forma completa. Por tanto, este sistema se podría llegar a llevar a producción, integrándolo con el sistema sanitario actual en la Comunidad de Madrid. Para esto, no hace falta decir que se necesitaría mucho trabajo futuro donde el sistema se mejoraría y se prepararía para su operación y mantenimiento.

Sin embargo, tal y como se mencionó en el resumen del trabajo, el propósito de este Trabajo Fin de Máster no es presentar únicamente una solución para el control de la pandemia COVID-19, sino presentar un sistema de gestión de identidad digital paneuropeo extensible a otros escenarios sanitarios. El alumno cree que este sistema es perfectamente extensible a otros ejemplos de aplicación en el ámbito de la sanidad, donde únicamente habría que añadir los esquemas de credenciales necesarias y definir qué presentaciones debe solicitar el verificador. Otros ejemplos de aplicación son el del control de otras pandemias y endemias, el de crear un DNI neonatal o el de perseguir el enfoque propuesto por la OMS de una salud única (*One Health*).

Por último, es importante volver a mencionar que el proyecto **covID Card** ha permitido al alumno desarrollar este Trabajo Fin de Máster, donde ha tenido especial importancia para el caso

de validación a través del MVP. Esto es así, porque no se ha validado únicamente el trabajo sobre un escenario definido sobre el papel, sino que se ha validado al desarrollarse dentro de un proyecto competitivo subvencionado por la Comunidad de Madrid. Esto ha supuesto una oportunidad para el alumno de participar con el grupo de investigación STRAST (UPM) para aplicar conocimientos de identidad autosoberana y de otras tecnologías en un proyecto real.

5.2 Líneas futuras

Como se ha ido mencionando a lo largo de la memoria, este Trabajo Fin de Máster invita a realizar diferentes tareas como trabajo futuro. Las más importantes se recogen a continuación:

- Ampliar el sistema diseñado y desarrollado para más casos de uso sanitarios
    - Un ejemplo sería el de un DNI neonatal donde se permite la identificación de los bebés en el momento de su nacimiento, reduciendo así situaciones tan crueles como la del robo o el intercambio de niños.
    - Otro ejemplo interesante sería el de extender el sistema para el enfoque de salud única (*One Health*) propuesto por la OMS.
- En el caso de que las credenciales AnonCreds empiecen a soportar algunas de las capacidades descritas en el apartado de conclusiones, integrarlas en el sistema.
- En agentes como el de RUC19, se ha observado como un sanitario o administrativo se debería identificar de forma convencional en el sistema sanitario de la Comunidad de Madrid para hacer uso del agente. En producción, sería más interesante que la identificación se produjese a través de credenciales verificables o que el agente que emite las credenciales sanitarias fuese el mismo empleado, el cual tendría expedida del agente RUC19 una credencial de autorización.
- Sobre el agente Iberia hay varias mejoras posibles:
    - La aplicación podría ser una aplicación hecha a medida para la máquina (quiosco), en lugar de una aplicación web.
    - Se podría implementar un algoritmo de reconocimiento facial que verifique internamente si la foto recibida por la presentación enviada por el viajero coincide con la persona. Así, se evitaría mostrar la foto por pantalla y la necesidad de aplicar el juicio del vigilante de seguridad.
    - Se podría sustituir el vigilante de seguridad de forma completa si, además del punto anterior, se integrase algún tipo de barrera o puerta física que únicamente se abriese si el viajero cumple con todos los requisitos.
    - El hardware y el software podrían ser ofrecidos y auditados por el estado, con el fin de evitar posibles acciones maliciosas. Algunos ejemplos de estas acciones pueden ser las llamadas a API externas no autorizadas, las capturas de pantalla o el almacenamiento de información en una base de datos.
- La funcionalidad del agente Cartera Digital SaludMadrid podría incluirse en la aplicación “Tarjeta Sanitaria” de la Comunidad de Madrid.
- Se deberían integrar pruebas unitarias, de integración y de aceptación al sistema.

- Se podría mejorar el despliegue a través de la orquestación de contenedores con Kubernetes y del despliegue en plataformas como Google Cloud o AWS.
- Se deberían añadir y configurar diferentes herramientas de operación y mantenimiento, como herramientas de integración y distribución continuas (CI/CD) como GitHub Actions, como herramientas que ofrecen una Infraestructura como Código como Terraform o como herramientas de análisis de *logs* como Elastic Search.

La mayoría de las tareas mencionadas se centran en acercar el proyecto a la fase de producción.

BIBLIOGRAFÍA

[1] *Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento general de protección de datos)*. Diario Oficial de la Unión Europea, 2016, p. L 119/1. Accedido: ene. 05, 2023. [En línea]. Available: http://data.europa.eu/eli/reg/2016/679/oj

[2] *Reglamento (UE) 910/2014 del Parlamento Europeo y del Consejo, de 23 de julio de 2014 , relativo a la identificación electrónica y los servicios de confianza para las transacciones electrónicas en el mercado interior y por la que se deroga la Directiva 1999/93/CE*. Diario Oficial de la Unión Europea, 2014, p. L 257/73. Accedido: ene. 05, 2023. [En línea]. Available: http://data.europa.eu/eli/reg/2014/910/oj

[3] *Propuesta de REGLAMENTO DEL PARLAMENTO EUROPEO Y DEL CONSEJO por el que se modifica el Reglamento (UE) n.º 910/2014 en lo que respecta al establecimiento de un Marco para una Identidad Digital Europea*. Comisión Europea, 2021. Accedido: ene. 06, 2023. [En línea]. Available: https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A52021PC0281

[4] Santander, «¿Qué es la identidad digital?», nov. 04, 2022. https://www.santander.com/es/stories/que-es-la-identidad-digital (accedido ene. 06, 2023).

[5] «EBSI». https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home (accedido ene. 07, 2023).

[6] Comisión Europea, «Sanidad electrónica y COVID-19». https://health.ec.europa.eu/ehealth-digital-health-and-care/ehealth-and-covid-19_es (accedido ene. 07, 2023).

[7] C. de Madrid, «Fondos Estructurales en la Dirección General de Investigación e Innovación Tecnológica». https://www.comunidad.madrid/servicios/educacion/fondos-estructurales-direccion-general-investigacion-e-innovacion-tecnologica (accedido ene. 07, 2023).

[8] Comisión Europea, «REACT-EU». https://commission.europa.eu/funding-tenders/find-funding/eu-funding-programmes/react-eu_es (accedido ene. 07, 2023).

[9] J. J. Quisquater *et al.*, «How to explain zero-knowledge protocols to your children», *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 435 LNCS, pp. 628-631, 1990, doi: 10.1007/0-387-34805-0_60.

[10] A. Preukschat y D. Reed, *Self-Sovereign Identity. Decentralized digital identity and verifiable credentials*. Manning Publications, 2021. Accedido: ene. 08, 2023. [En línea]. Available: https://www.manning.com/books/self-sovereign-identity

[11] W3C, «World Wide Web Consortium (W3C)». https://www.w3.org/ (accedido ene. 08, 2023).

[12] DIF, «DIF - Decentralized Identity Foundation». https://identity.foundation/ (accedido ene. 08, 2023).

[13] Sovrin, «Sovrin». https://sovrin.org/ (accedido ene. 10, 2023).

[14] Hyperledger, «aries-rfcs». GitHub. Accedido: ene. 10, 2023. [En línea]. Available: https://github.com/hyperledger/aries-rfcs

[15] Hyperledger, «indy-hipe». GitHub. Accedido: ene. 10, 2023. [En línea]. Available: https://github.com/hyperledger/indy-hipe

[16] W3C, «Decentralized Identifiers (DIDs) v1.0». https://w3c.github.io/did-core/ (accedido ene. 09, 2023).

[17] W3C, «Documents published at W3C». https://www.w3.org/standards/types#REC (accedido ene. 09, 2023).

[18] W3C, «DID Specification Registries». https://w3c.github.io/did-spec-registries/ (accedido ene. 10, 2023).

[19] DIF, «universal-resolver». GitHub. Accedido: ene. 10, 2023. [En línea]. Available: https://github.com/decentralized-identity/universal-resolver/

[20] DIF, «Universal Resolver». https://dev.uniresolver.io/ (accedido ene. 10, 2023).

[21] W3C, «The did:key Method v0.7». https://w3c-ccg.github.io/did-method-key/ (accedido ene. 10, 2023).

[22] DIF, «Peer DID Method Specification». https://identity.foundation/peer-did-method-spec/index.html (accedido ene. 10, 2023).

[23] Sovrin, «Sovrin DID Method Specification». https://sovrin-foundation.github.io/sovrin/spec/did-method-spec-template.html (accedido ene. 10, 2023).

[24] Hyperledger, «Indy DID Method Specification». https://hyperledger.github.io/indy-did-method/ (accedido ene. 10, 2023).

[25] W3C, «did:web Method Specification». https://w3c-ccg.github.io/did-method-web/ (accedido ene. 10, 2023).

[26] DIF, «github-did». GitHub. Accedido: ene. 10, 2023. [En línea]. Available: https://github.com/decentralized-identity/github-did/blob/master/docs/did-method-spec/index.md

[27] EBP, «EBSI DID Method». https://ec.europa.eu/digital-building-blocks/wikis/display/EBSIDOC/EBSI+DID+Method (accedido ene. 10, 2023).

[28] DIF, «DIDComm Messaging Specification v2 Editor's Draft». https://identity.foundation/didcomm-messaging/spec/ (accedido ene. 14, 2023).

[29] W3C, «Verifiable Credentials Data Model v1.1». https://www.w3.org/TR/vc-data-model/ (accedido ene. 08, 2023).

[30] W3C, «Revocation List 2020». https://w3c-ccg.github.io/vc-status-rl-2020/ (accedido ene. 15, 2023).

[31] A. Tobin, «Sovrin: What Goes on the Ledger? Revision History Minor changes Replace identity "owner" with identity "holder" Executive Summary», 2017.

[32] Hyperledger, «0011-cred-revocation». GitHub. Accedido: ene. 15, 2023. [En línea]. Available: https://github.com/hyperledger/indy-hipe/tree/master/text/0011-cred-revocation

[33] Hyperledger, «AnonCreds Specification». https://hyperledger.github.io/anoncreds-spec/ (accedido ene. 15, 2023).

[34] The Linux Foundation, «Becoming a Hyperledger Aries Developer», *edX*. https://www.edx.org/course/becoming-a-hyperledger-aries-developer (accedido ene. 15, 2023).

[35] Hyperledger, «Hyperledger Wiki», 2017. https://wiki.hyperledger.org/ (accedido ene. 21, 2023).

[36] «Trust Over IP - Defining a complete architecture for Internet-scale digital trust». https://trustoverip.org/ (accedido ene. 20, 2023).

[37] EBSI, «Trusted Ledgers and Smart Contracts Registry – Smart Contract». https://ec.europa.eu/digital-building-blocks/wikis/pages/viewpage.action?pageId=555222971 (accedido ene. 21, 2023).

[38] Hyperledger, «indy-node: The server portion of a distributed ledger purpose-built for decentralized identity.» GitHub. Accedido: ene. 21, 2023. [En línea]. Available: https://github.com/hyperledger/indy-node

[39] Hyperledger, «indy-plenum: Plenum Byzantine Fault Tolerant Protocol». GitHub. Accedido: ene. 21, 2023. [En línea]. Available: https://github.com/hyperledger/indy-plenum

[40] Hyperledger, «indy-sdk». GitHub. Accedido: ene. 21, 2023. [En línea]. Available: https://github.com/hyperledger/indy-sdk

[41] Hyperledger, «indy-vdr: A library and proxy server for interacting with Hyperledger Indy Node ledger instances». GitHub. Accedido: ene. 21, 2023. [En línea]. Available: https://github.com/hyperledger/indy-vdr

[42] Hyperledger, «indy-shared-rs: Shared Rust data types and utility functions for Hyperledger Indy.» GitHub. Accedido: ene. 21, 2023. [En línea]. Available: https://github.com/hyperledger/indy-shared-rs

[43] P. L. Aublin, S. ben Mokhtar, y V. Quema, «RBFT: Redundant byzantine fault tolerance», *Proc Int Conf Distrib Comput Syst*, pp. 297-306, 2013, doi: 10.1109/ICDCS.2013.53.

[44] Sovrin, «Glossary». https://sovrin.org/library/glossary/ (accedido ene. 21, 2023).

[45] Sovrin, «TAA». GitHub. Accedido: ene. 22, 2023. [En línea]. Available: https://github.com/sovrin-foundation/sovrin/blob/master/TAA/TAA.md

[46] British Columbia, «indy-tails-server: This software stores and makes available tails files for use with Hyperledger Indy». GitHub. Accedido: ene. 27, 2023. [En línea]. Available: https://github.com/bcgov/indy-tails-server

[47] Bcgov, «von-network: A portable development level Indy Node network.» GitHub. Accedido: ene. 22, 2023. [En línea]. Available: https://github.com/bcgov/von-network

[48] B. Columbia, «Home - Digital Government - Province of British Columbia». https://digital.gov.bc.ca/ (accedido ene. 22, 2023).

[49] «BCovrin Dev Indy Network». http://dev.bcovrin.vonx.io/ (accedido ene. 22, 2023).

[50] «BCovrin Test Indy Network». http://test.bcovrin.vonx.io/ (accedido ene. 22, 2023).

[51] Sovrin, «What Is Hyperledger Indy?» https://sovrin.org/faq/what-is-hyperledger-indy/ (accedido ene. 22, 2023).

[52] Sovrin, «Sovrin Price Plan». https://sovrin.org/sovrin-price-plan/ (accedido ene. 10, 2023).

[53] IDunion, «IDunion». https://idunion.org/?lang=en (accedido ene. 22, 2023).

[54] IDunion, «Project». https://idunion.org/projekt/?lang=en (accedido ene. 22, 2023).

[55] «HL Indy Tx Explorer». https://idunion.esatus.com/home/IDunion_Pilot (accedido ene. 22, 2023).

[56] Lissi, «IDunion Test Ledger». https://docs.lissi.id/lissi-agent/idunion-ledger (accedido ene. 22, 2023).

[57] IDunion, «IDunion Test Network Transaction Author Agreement». https://idunion.org/wp-content/uploads/2021/12/EUI_1208174996_14_IDunion-Transaction-Author-Agreement_JD_20210708-1-attachment-to-Konsortialvertrag-July-2021.pdf (accedido ene. 22, 2023).

[58] Indicio, «Indicio Networks». https://indicio.tech/indicio-networks/ (accedido ene. 22, 2023).

[59] Indicio, «Indicio Transaction Author Agreement». GitHub. Accedido: ene. 22, 2023. [En línea]. Available: https://github.com/Indicio-tech/indicio-governance/blob/main/Indicio Transaction Author Agreement.pdf

[60] Hyperledger Aries, «0302-aries-interop-profile». GitHub. Accedido: ene. 22, 2023. [En línea]. Available: https://github.com/hyperledger/aries-rfcs/tree/main/concepts/0302-aries-interop-profile

[61] Hyperledger Aries, «0160-connection-protocol». https://github.com/hyperledger/aries-rfcs/tree/main/features/0160-connection-protocol (accedido ene. 22, 2023).

[62] Hyperledger Aries, «0023-did-exchange». https://github.com/hyperledger/aries-rfcs/blob/main/features/0023-did-exchange/README.md (accedido ene. 22, 2023).

[63] Hyperledger Aries, «0095-basic-message». https://github.com/hyperledger/aries-rfcs/tree/main/features/0095-basic-message (accedido ene. 22, 2023).

[64] Hyperledger Aries, «0036-issue-credential». https://github.com/hyperledger/aries-rfcs/tree/main/features/0036-issue-credential (accedido ene. 22, 2023).

[65] Hyperledger Aries, «0037-present-proof». https://github.com/hyperledger/aries-rfcs/tree/main/features/0037-present-proof (accedido ene. 22, 2023).

[66] Hyperledger Aries, «aries-cloudagent-python: Hyperledger Aries Cloud Agent Python (ACA-Py) is a foundation for building decentralized identity applications and services running in non-mobile environments.» GitHub. Accedido: ene. 23, 2023. [En línea]. Available: https://github.com/hyperledger/aries-cloudagent-python

[67] Hyperledger Aries, «aries-framework-dotnet: Aries Framework .NET for building multiplatform SSI services». GitHub. Accedido: ene. 23, 2023. [En línea]. Available: https://github.com/hyperledger/aries-framework-dotnet

[68] Hyperledger Aries, «aries-framework-go: Hyperledger Aries Framework Go provides packages for building Agent / DIDComm services.» GitHub. Accedido: ene. 23, 2023. [En línea]. Available: https://github.com/hyperledger/aries-framework-go/

[69] W3C, «The did:orb Method v0.2». https://trustbloc.github.io/did-method-orb/ (accedido ene. 23, 2023).

[70] Hyperledger Aries, «aries-framework-javascript: Aries Framework JavaScript (Built using TypeScript)». GitHub. Accedido: ene. 23, 2023. [En línea]. Available: https://github.com/hyperledger/aries-framework-javascript

[71] Hyperledger Aries, «aries-vcx: AriesVCX is a Rust library for building web and mobile applications issuing, holding, presenting and verifying Verifiable Credentials in accordance to the standards set by Hyperledger Aries.» GitHub. Accedido: ene. 23, 2023. [En línea]. Available: https://github.com/hyperledger/aries-vcx

[72] Hyperledger Aries, «aries-agent-test-harness: Aries agent test framework, with agent backchannel support». GitHub. Accedido: ene. 23, 2023. [En línea]. Available: https://github.com/hyperledger/aries-agent-test-harness

[73] «Aries Interoperability Test Results». https://aries-interop.info/ (accedido ene. 23, 2023).

[74] «Allure Docker Service UI». https://allure.vonx.io/allure-docker-service-ui/projects/acapy-b-javascript/reports/latest (accedido ene. 23, 2023).

[75] British Columbia, «aries-cloudagent - Docker Image». Docker Hub. Accedido: ene. 23, 2023. [En línea]. Available: https://hub.docker.com/r/bcgovimages/aries-cloudagent

[76] Hyperledger Aries, «aries-mobile-agent-react-native: Aries Mobile Agent React Native - Part of the Aries Bifold effort to provide SSI capabilities in a production ready app.» GitHub. Accedido: ene. 23, 2023. [En línea]. Available: https://github.com/hyperledger/aries-mobile-agent-react-native

[77] Hyperledger Aries, «aries-mobile-agent-xamarin». GitHub. Accedido: ene. 23, 2023. [En línea]. Available: https://github.com/hyperledger/aries-mobile-agent-xamarin

[78] «Identity Wallets • Trinsic». https://trinsic.id/identity-wallets/ (accedido ene. 23, 2023).

[79] «Lissi - Building trusted relationships». https://www.lissi.id/ (accedido ene. 23, 2023).

[80] «Self-Sovereign Identity – esatus AG». https://esatus.com/index.html%3Fp=7663&lang=en.html (accedido ene. 23, 2023).

[81] «BC Wallet – Technology Overview | BC Digital Trust». https://digital.gov.bc.ca/digital-trust/projects-and-initiatives/bc-wallet-technology-overview/ (accedido ene. 23, 2023).

[82] «Connect.me». https://connect.me/ (accedido ene. 23, 2023).

[83] «What is EBSI». https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/What+is+ebsi#anchor-ebsi-use-cases (accedido ene. 24, 2023).

[84] «EBSI W3C Verifiable Credentials (VCs) and W3C Verifiable Presentations (VPs). - EBSI Specifications -». https://ec.europa.eu/digital-building-blocks/wikis/pages/viewpage.action?pageId=555222155 (accedido ene. 24, 2023).

[85] Comisión Europea, «About CEF European Blockchain Services Infrastructure (EBSI)». https://joinup.ec.europa.eu/collection/connecting-europe-facility-cef/solution/cef-european-blockchain-services-infrastructure-ebsi/about (accedido ene. 24, 2023).

[86] «TrustID | Hyperledger Labs». https://labs.hyperledger.org/labs/trustID.html (accedido ene. 24, 2023).

[87] Hyperledger Labs, «TrustID: Decentralized Identity solution compatible with different Hyperledger platforms.» GitHub. Accedido: ene. 24, 2023. [En línea]. Available: https://github.com/hyperledger-labs/TrustID

[88] «Trinsic Ecosystems Is Now Our Exclusive Platform for New Users». https://trinsic.id/ecosystems-exclusive-platform-for-new-users/ (accedido ene. 24, 2023).

[89] Gataca, «Decentralized Identity Management». https://gataca.io/ (accedido ene. 10, 2023).

[90] «Postman API Platform». https://www.postman.com/ (accedido ene. 24, 2023).

[91] «React – Una biblioteca de JavaScript para construir interfaces de usuario». https://es.reactjs.org/ (accedido ene. 24, 2023).

[92] «React Native · Learn once, write anywhere». https://reactnative.dev/ (accedido ene. 24, 2023).

[93] «Expo». https://expo.dev/ (accedido ene. 24, 2023).

[94] «Home v6.7.0 | React Router». https://reactrouter.com/en/main (accedido ene. 24, 2023).

[95] «React Navigation». https://reactnavigation.org/ (accedido ene. 24, 2023).

[96] «Bootstrap · The most popular HTML, CSS, and JS library in the world.» https://getbootstrap.com/ (accedido ene. 24, 2023).

[97] «Axios Docs». https://axios-http.com/es/docs/intro (accedido ene. 24, 2023).

[98] «Socket.IO». https://socket.io/ (accedido ene. 24, 2023).

[99] «http-proxy-middleware». NPM. Accedido: ene. 24, 2023. [En línea]. Available: https://www.npmjs.com/package/http-proxy-middleware

[100] «Grid.js - Advanced JavaScript table plugin». https://gridjs.io/ (accedido ene. 24, 2023).

[101] «React Select». https://react-select.com/home (accedido ene. 24, 2023).

[102] «react-qr-code». NPM. Accedido: ene. 24, 2023. [En línea]. Available: https://www.npmjs.com/package/react-qr-code

[103] «BarCodeScanner - Expo Documentation». https://docs.expo.dev/versions/latest/sdk/bar-code-scanner/ (accedido ene. 24, 2023).

[104] «Node.js». https://nodejs.org/es/ (accedido ene. 24, 2023).

[105] «Express - Node.js web application framework». https://expressjs.com/ (accedido ene. 24, 2023).

[106] «node-cache». NPM. Accedido: ene. 25, 2023. [En línea]. Available: https://www.npmjs.com/package/node-cache

[107] «named-timers». NPM. Accedido: ene. 25, 2023. [En línea]. Available: https://www.npmjs.com/package/named-timers

[108] «MySQL Community Edition». https://www.mysql.com/products/community/ (accedido ene. 25, 2023).

[109] «DB-Engines Ranking - popularity ranking of database management systems». https://db-engines.com/en/ranking (accedido ene. 25, 2023).

[110] «MySQL 8.0 Reference Manual :: 13.1.20.6 CHECK Constraints». https://dev.mysql.com/doc/refman/8.0/en/create-table-check-constraints.html (accedido ene. 25, 2023).

[111] «mysql2». NPM. Accedido: ene. 25, 2023. [En línea]. Available: https://www.npmjs.com/package/mysql2

[112] «curl». https://curl.se/ (accedido ene. 25, 2023).

[113] «jq». https://stedolan.github.io/jq/ (accedido ene. 25, 2023).

[114] «Docker: Accelerated, Containerized Application Development». https://www.docker.com/ (accedido ene. 25, 2023).

[115] «Overview | Docker Documentation». https://docs.docker.com/compose/ (accedido ene. 25, 2023).

[116] «ngrok - Online in One Line». https://ngrok.com/ (accedido ene. 25, 2023).

[117] «wernight/ngrok - Docker Image». Docker Hub. Accedido: ene. 25, 2023. [En línea]. Available: https://hub.docker.com/r/wernight/ngrok/

[118] «ISO - ISO 8601 — Date and time format». https://www.iso.org/iso-8601-date-and-time-format.html (accedido ene. 26, 2023).

[119] «List of all countries with their 2 digit codes (ISO 3166-1) - Dataset - DataHub - Frictionless Data». https://datahub.io/core/country-list (accedido ene. 26, 2023).

[120] «node - Official Image». Docker Hub. Accedido: ene. 28, 2023. [En línea]. Available: https://hub.docker.com/_/node

[121] «alpine - Official Image». Docker Hub. Accedido: ene. 28, 2023. [En línea]. Available: https://hub.docker.com/_/alpine

[122] Hyperledger Aries, «AdminAPI.md · hyperledger/aries-cloudagent-python». GitHub. Accedido: ene. 29, 2023. [En línea]. Available: https://github.com/hyperledger/aries-cloudagent-python/blob/main/AdminAPI.md#administration-api-webhooks

[123] «API Documentation & Design Tools for Teams | Swagger». https://swagger.io/ (accedido ene. 29, 2023).

[124] Comisión Europea, «Sanidad electrónica y COVID-19». https://health.ec.europa.eu/ehealth-digital-health-and-care/ehealth-and-covid-19_es (accedido ene. 30, 2023).

[125] Comisión Europea, «Guidelines on Technical Specification for EU Digital COVID Certificates, Schema Version 1.3.0», *Version 1.0*, jun. 09, 2021. Accedido: ene. 30, 2023. [En línea]. Available: https://health.ec.europa.eu/system/files/2021-06/covid-certificate_json_specification_en_0.pdf

[126] Comisión Europea, «Guidelines on Value Sets for Digital Green Certificates», *Version 1.0*, abr. 21, 2021. Accedido: ene. 30, 2023. [En línea]. Available: https://health.ec.europa.eu/system/files/2021-04/digital-green-certificates_dt-specifications_en_0.pdf

[127] *Reglamento (UE) 2019/1157 del Parlamento Europeo y del Consejo, de 20 de junio de 2019, sobre el refuerzo de la seguridad de los documentos de identidad de los ciudadanos de la Unión y de los documentos de residencia expedidos a ciudadanos de la Unión y a*. Diario Oficial de la Unión Europea, 2019, p. L 188/67. Accedido: ene. 30, 2023. [En línea]. Available: http://data.europa.eu/eli/reg/2019/1157/oj

[128] Comunidad de Madrid, «Manual de Usuario | Registro Unificado de Vacunación COVID19». https://www.comunidad.madrid/sites/default/files/doc/sanidad/prev/ruv_manual_usuario.pdf (accedido ene. 30, 2023).

[129] «AHPPC statement on COVID-19 winter update and ongoing health protection measures to support our community | Australian Government Department of Health and Aged Care», jul. 08, 2022. https://www.health.gov.au/news/ahppc-statement-on-covid-19-winter-update-and-ongoing-health-protection-measures-to-support-our-community (accedido ene. 31, 2023).

[130] «Diagram Software and Flowchart Maker». https://www.diagrams.net/ (accedido ene. 31, 2023).

[131] «mysql - Official Image». Docker Hub. Accedido: ene. 31, 2023. [En línea]. Available: https://hub.docker.com/_/mysql

[132] «COVID-19 In Vitro Diagnostic Devices and Test Methods Database». https://covid-19-diagnostics.jrc.ec.europa.eu/devices?manufacturer&text_name&marking&rapid_diag&format&target_type&field-1=HSC%20common%20list%20%28RAT%29&value-1=1&search_method=AND#form_content (accedido ene. 30, 2023).

[133] «Faker». https://fakerjs.dev/ (accedido feb. 02, 2023).

[134] «react-spinners». NPM. Accedido: feb. 03, 2023. [En línea]. Available: https://www.npmjs.com/package/react-spinners

[135] «Identidad digital europea: el Consejo avanza hacia la cartera europea digital de la UE, un cambio de paradigma para la identidad digital en Europa», *Consilium*, dic. 2022, Accedido: feb. 04, 2023. [En línea]. Available: https://www.consilium.europa.eu/es/press/press-releases/2022/12/06/european-digital-identity-eid-council-adopts-its-position-on-a-new-regulation-for-a-digital-wallet-at-eu-level/

[136] N. A. and R. A. Office of the Federal Register, *Public Law 104 - 191 - Health Insurance Portability and Accountability Act of 1996*. U.S. Government Printing Office, 1996. Accedido: ene. 08, 2023. [En línea]. Available: https%3A%2F%2Fwww.govinfo.gov%2Fapp%2Fdetails%2FPLAW-104publ191

[137] Sovrin, «Announcement: enforcement of accountability requirements on the Sovrin Networks by 1 February 2023». https://sovrin.org/announcement-enforcement-of-accountability-requirements-on-the-sovrin-networks-by-1-february-2023/ (accedido ene. 22, 2023).

[138] L. de Jong, «Aries Cloud Agent Python (ACA-py) Webhooks», 2021. https://ldej.nl/post/aries-cloudagent-python-webhooks/ (accedido ene. 29, 2023).

[139] «WWW FAQs: What is the maximum length of a URL?» https://web.archive.org/web/20060218052923/http:/www.boutell.com/newfaq/misc/urllength.html (accedido ene. 31, 2023).

ANEXO I: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

En este anexo se van a presentar los posibles aspectos éticos, económicos, sociales y ambientales que rodean este Trabajo Fin de Máster.

I.1 INTRODUCCIÓN

En cuanto al **sector tecnológico**, este TFM se enmarca en los sistemas telemáticos, perteneciente al sector de las TIC (Tecnologías de la Información y la Comunicación).

Respecto al **ámbito organizativo y estratégico**, ya se ha mencionado en el apartado 1.1.2 que este TFM se enmarca en un proyecto concedido a la UPM (**MadridDataSpace4Pandemics-CM**), financiado por un paquete de ayudas aprobado por la Comisión Europea llamado REACT-EU. Este proyecto a su vez de varios proyectos de menor tamaño, uno de ellos llamado **covID Card**, concedido al grupo STRAST de la escuela (ETSIT) y sobre el cual el alumno ha desarrollado este TFM. El alumno se ha encargado de la adquisición de conocimiento y del diseño y desarrollo de una Prueba de Concepto y de un Producto Viable Mínimo.

Respecto al **ciclo de vida del proyecto**, este se compone de las siguientes fases: adquisición de conocimiento, diseño del sistema, implementación, distribución y puesta en producción. En este TFM se cubren todas las fases menos las dos últimas.

En cuanto al **contexto socio-económico**, ya se ha mencionado durante toda la memoria y, especialmente, en el apartado 1.1 y en el apartado 2.2.2 los diferentes problemas que presenta la actual identidad centralizada basada en PKI en cuanto a seguridad y privacidad de los datos y en cuanto a los costes económicos de la infraestructura asociada, y cómo la fase de producción de este proyecto podría aliviar dichos problemas. Y, respecto al contexto geográfico, este TFM se centra en el territorio de la Unión Europea, con cierto énfasis en la Comunidad de Madrid.

En cuanto al **contexto legal**, ya se ha visto en el apartado 1.1 y en el apartado 2.2.1.1 como existen diferentes regulaciones europeas que el proyecto debe seguir, como eIDAS y la RGPD. Este TFM conlleva una serie de impactos éticos y sociales con respecto a estas regulaciones, como una mejoría de la privacidad y la seguridad de los datos de los ciudadanos.

Y, por último, en cuanto a los **grupos de interés**, se tienen las instituciones públicas que tengan que adoptar estos sistemas de identidad autosoberana, los ciudadanos, las empresas privadas que reducen los datos que tratan de los ciudadanos y los trabajadores asociados a la infraestructura PKI. Existiría un impacto positivo en los ciudadanos al mejorar la privacidad y seguridad de sus datos, de las instituciones públicas al facilitar la emisión de credenciales y de las empresas privadas al facilitar la verificación de credenciales recibidas por los ciudadanos. Y existiría un impacto negativo en las empresas privadas cuyo modelo de negocio se centra en el tratamiento de datos de los ciudadanos y en los trabajadores asociados a la infraestructura PKI al peligrar sus trabajos. No obstante, este proyecto incentivaría también la creación de empleos cualificados.

I.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

Los impactos más relevantes relacionados con el proyecto se resumen en la siguiente tabla:

Aspectos más relevantes	Descripción	Grupos afectados	Reglamentos	Posibilidades de evaluación	Implicaciones económicas
Mayor privacidad de los datos de los ciudadanos. **[Impacto ético]**	Las credenciales verificables controladas por los ciudadanos permiten aumentar la privacidad de los datos.	Ciudadanos. Empresas Instituciones.	RGPD.	Comparando las sanciones impuestas por el incumplimiento de la RGPD.	Reducción de las sanciones de la RGPD.
Mayor seguridad de los datos de los ciudadanos. **[Impacto social]**	Las conexiones seguras y las credenciales verificables aumentan la seguridad de los datos.	Ciudadanos. Empresas. Instituciones.	RGPD.	Comparando la cantidad de datos filtrados y del impacto percibido de estas filtraciones.	Menor coste en políticas de ciberseguridad.
Eliminación de los costes asociados a PKI. **[Impacto económico]**	Los VDR permiten prescindir de la infraestructura PKI, eliminado los costes asociados.	Instituciones.	-	Comparando el presupuesto destinado a PKI y a DPKI.	Menores costes asociados a la infraestructura PKI.
Reducción de la falsificación de documentos. **[Impacto social]**	Las credenciales verificables permiten reducir la falsificación de credenciales y la suplantación de identidad.	Empresas. Instituciones.	RGPD. eIDAS.	Comparando las sanciones impuestas por la falsificación de documentos.	Menor coste en métodos para comprobar la veracidad de los documentos.

Tabla 14: Descripción de impactos relevantes relacionados con el proyecto

I.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS

De los impactos descritos en la tabla anterior, los que más impacto cree el alumno que tienen son la mejoría de la privacidad de los datos de los ciudadanos y la reducción de la falsificación de documentos. Estos se van a describir con más detalle.

El aspecto de **mayor privacidad de los datos de los ciudadanos** se produce gracias al uso de las credenciales verificables por parte de los ciudadanos, donde estos deciden qué quieren revelar, a quién, cuándo y cómo. Esto quiere decir que un ciudadano puede elegir exactamente que información de su credencial quiere mostrar, estando seguro de a quién está presentándosela y siendo consciente en todo momento de cuándo se está produciendo. Esto no solo afecta a los ciudadanos, sino también a las instituciones públicas que deben emitir estas credenciales y a las empresas privadas que deben adaptarse a esta minimización de datos y a un tratamiento de datos más controlado por el ciudadano. Todo esto facilita el cumplimiento del Reglamento General de Protección de Datos europeo, reduciendo por tanto todos los costes asociados a las sanciones impuestas a las empresas por el incumplimiento de esta regulación.

Y en cuanto al aspecto de **reducción de falsificación de documentos**, estas credenciales verificables permiten reducir la falsificación de documentos, al establecerse una confianza criptográfica que asegura que una credencial pertenece a la persona que está presentándola y de que no se ha alterado. Esto implica también una reducción de suplantación de identidad, cuando se tratan de credenciales que contienen datos identificativos de una persona, como una credencial de identidad. Todo esto se apoya también sobre las conexiones seguras y los registros de datos verificable (VDR).

I.4 CONCLUSIONES

La valoración de estos aspectos éticos, sociales, económicos y medioambientales han añadido un valor al proyecto, permitiendo al alumno enfocar ciertas decisiones tomadas hacia una mayor contribución de estos impactos.

Un ejemplo de este valor añadido fue la decisión de elegir el tipo de credencial AnonCreds sobre las credenciales JSON-LD, ya que se tuvo en cuenta el impacto sobre la privacidad de los ciudadanos que tendría un tipo frente al otro.

Y otro ejemplo fue el diseño realizado sobre los esquemas de las credenciales y sobre las solicitudes de presentaciones de los verificadores, en donde se tuvo en cuenta el principio de minimización de la RGPD.

Por tanto, esta consideración de los diferentes aspectos en relación con el proyecto ha permitido al alumno tomar decisiones más acertadas, implicando así un valor añadido al proyecto.

ANEXO II: PRESUPUESTO ECONÓMICO

Este anexo incluye el presupuesto económico estimado asociado a la elaboración de este Trabajo Fin de Máster:

COSTE DE MANO DE OBRA (coste directo)	Horas	Precio/hora	Total
	880	18 €	**15.840 €**

COSTE DE RECURSOS MATERIALES (coste directo)	Precio de compra	Uso en meses	Amortización (en años)	Total
Ordenador personal (Software incluido)	817,29 €	10	5	136,21 €

COSTE TOTAL DE RECURSOS MATERIALES	**136,21 €**

GASTOS GENERALES (costes indirectos)	15%	sobre CD	**2.396,43 €**
BENEFICIO INDUSTRIAL	6%	sobre CD+CI	**1.102,36 €**

SUBTOTAL PRESUPUESTO		**19.475 €**
IVA APLICABLE	21%	**4.089,75 €**
TOTAL PRESUPUESTO		**23.564,75 €**

Tabla 15: Presupuesto económico

II.1 Costes detallados

En este apartado se detallan los costes directos indicados en la tabla anterior.

En cuanto a los **costes materiales**, estos están asociados únicamente al ordenador personal del alumno. Este ordenador se trata de un ordenador portátil marca Lenovo y producto *ThinkBook 14s Yoga*, que le costó al alumno 817,29€ (IVA incluido). La amortización se ha tenido en cuenta solo con los meses activos del alumno en el proyecto.

Y en cuanto a los **costes de mano de obra**, estos se han calculado teniendo en cuenta un salario bruto anual de 33.000 €/año, que se corresponde con la media de los sueldos ofrecidos por ciertas ofertas de trabajo para las cuales el alumno cumple con los requisitos. Este salario bruto se traduce en un precio por hora de 18€.

Además, se ha tenido en cuenta una duración de proyecto de 12 meses, de los cuales durante 2 meses el alumno no dedicó tiempo. Esto es, un total de 10 meses con una dedicación de 4 horas al día durante 5 días a la semana, resultando un total de 880 horas dedicadas al proyecto.

ANEXO III: TABLA COMPARATIVA DE LOS FRAMEWORK DE HYPERLEDGER ARIES

A continuación se muestra una tabla comparativa de los principales framework disponibles en Hyperledger Aries, donde se detalla el lenguaje en el que está programado el propio framework, el lenguaje en el que se debe programar el controlador (indicando entre paréntesis cómo se integra el framework en el controlador), si permite aplicarlo en agentes móviles y en agentes de servidor (empresariales), si soporta AIP 1.0 y AIP 2.0, los formatos de credenciales permitidas y si es agnóstico al VDR utilizado [34], [66], [67], [68], [70], [71]:

Nombre	Lenguaje Framework	Lenguaje Controlador	Agentes móviles	Agentes de servidor	AIP 1.0	AIP 2.0	Formatos de credenciales	Agnóstico al VDR
ACA-Py	Python	Cualquiera (Interfaz HTTP)	No	Sí	Sí	Sí (soporte incompleto)	AnonCreds JSON-LD (W3C)	Sí
AF-.NET	.NET	Compatibles con .NET (biblioteca importable)	Sí	Sí	Sí	No	AnonCreds	No (*did:sov*)
AF-Go	Go	Cualquiera (Interfaz HTTP)	Sí	Sí	No	Sí (soporte incompleto)	JSON-LD (W3C)	No (método propio *did:orb*)
AFJ	JavaScript	JavaScript (biblioteca importable)	Sí (React Native)	Sí (Node.js)	Sí	No	AnonCreds JSON-LD (W3C)	No (*did:sov*)
AVCX	Rust	Compatible con C (biblioteca importable)	Sí	Sí	Sí	No	AnonCreds	No (*did:sov*)

Tabla 16: Comparativa de framework de Hyperledger Aries

ANEXO IV: DIAGRAMAS DE SECUENCIA DETALLADOS (POC)

En la Prueba de Concepto se han presentado unos diagramas de secuencia para las tres etapas con un nivel de abstracción alto, sin incluir detalles técnicos. En este anexo se van a incluir diagramas de secuencia de elaboración propia con diferentes niveles de detalle. En total, se presentarán dos niveles de detalle para cada una de las tres etapas. El objetivo de estos diagramas es intentar transmitir el nivel de complejidad interno a nivel técnico de las interacciones entre agentes. Los diagramas están en inglés, ya que los detalles técnicos están en este idioma (estados, parámetros, recursos, etc.) y no se ha querido mezclar idiomas.

Diagramas con un **nivel de detalle técnico simplificado**. Se incluyen las llamadas que realiza el cliente (entidades con la etiqueta "*FRONT*") hacia el servidor (entidades con la etiqueta "*BACK*"), indicando los parámetros más relevantes. Adicionalmente, se muestran los eventos que se reciben, con los parámetros más relevantes.

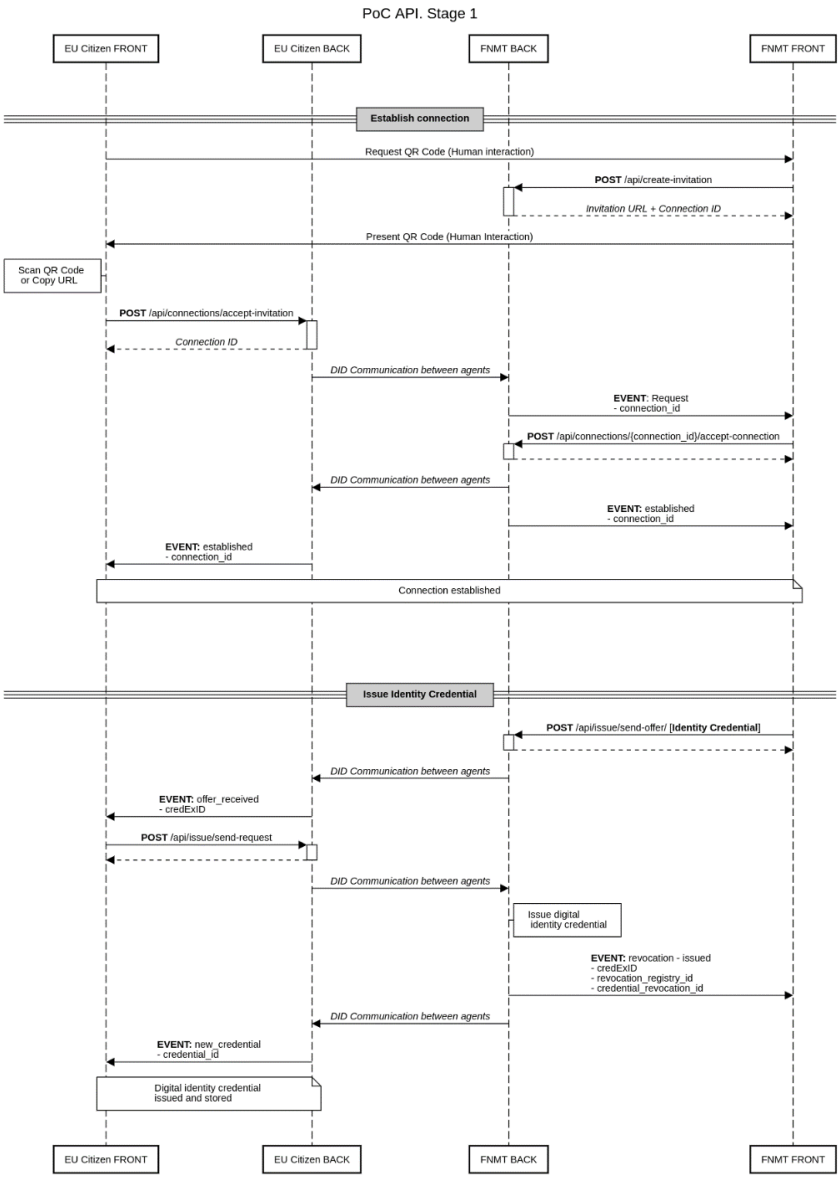

Figura 25: Diagrama de secuencia técnico simplificado de la etapa 1 de la PoC

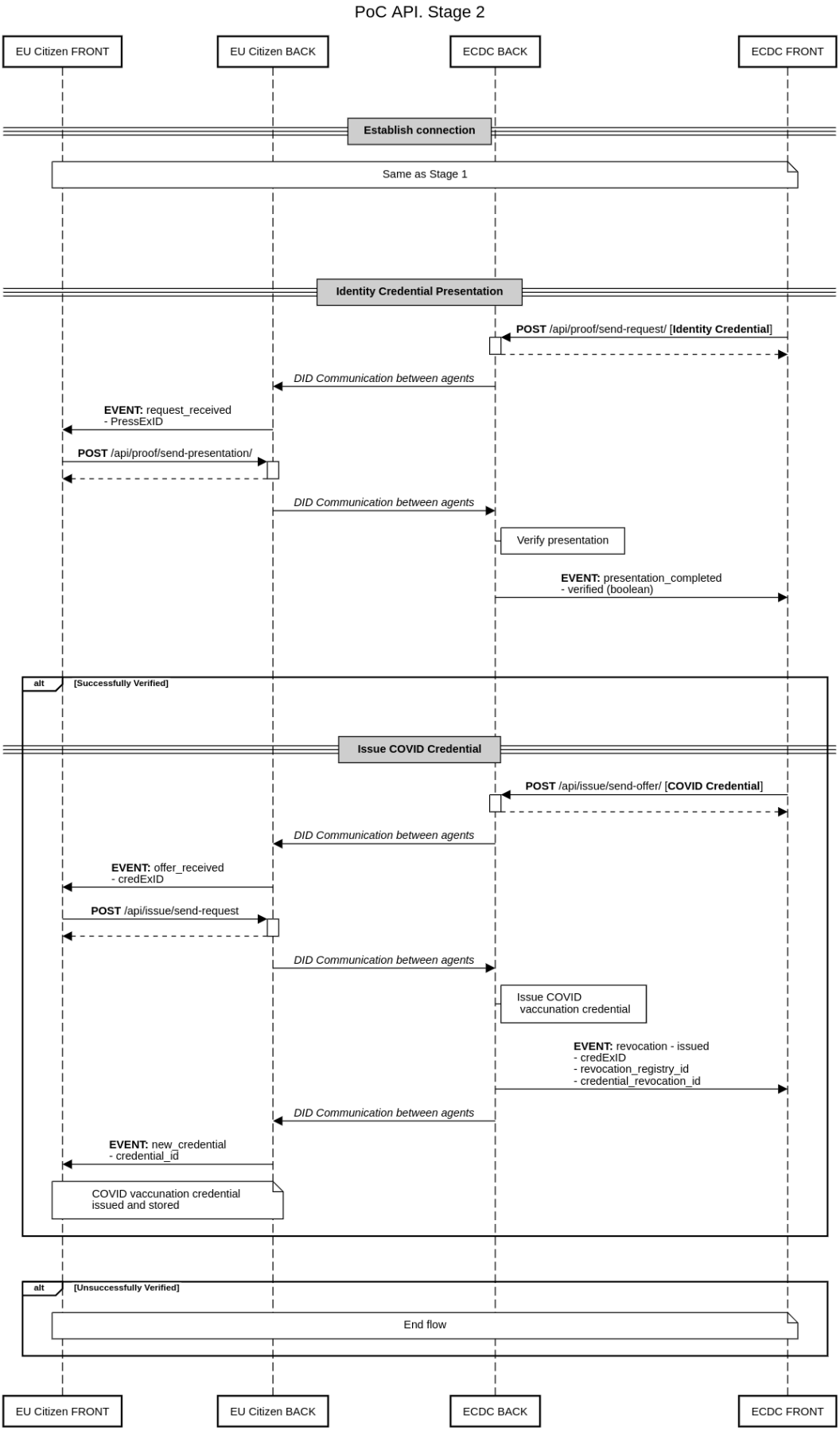

Figura 26: Diagrama de secuencia técnico simplificado de la etapa 2 de la PoC

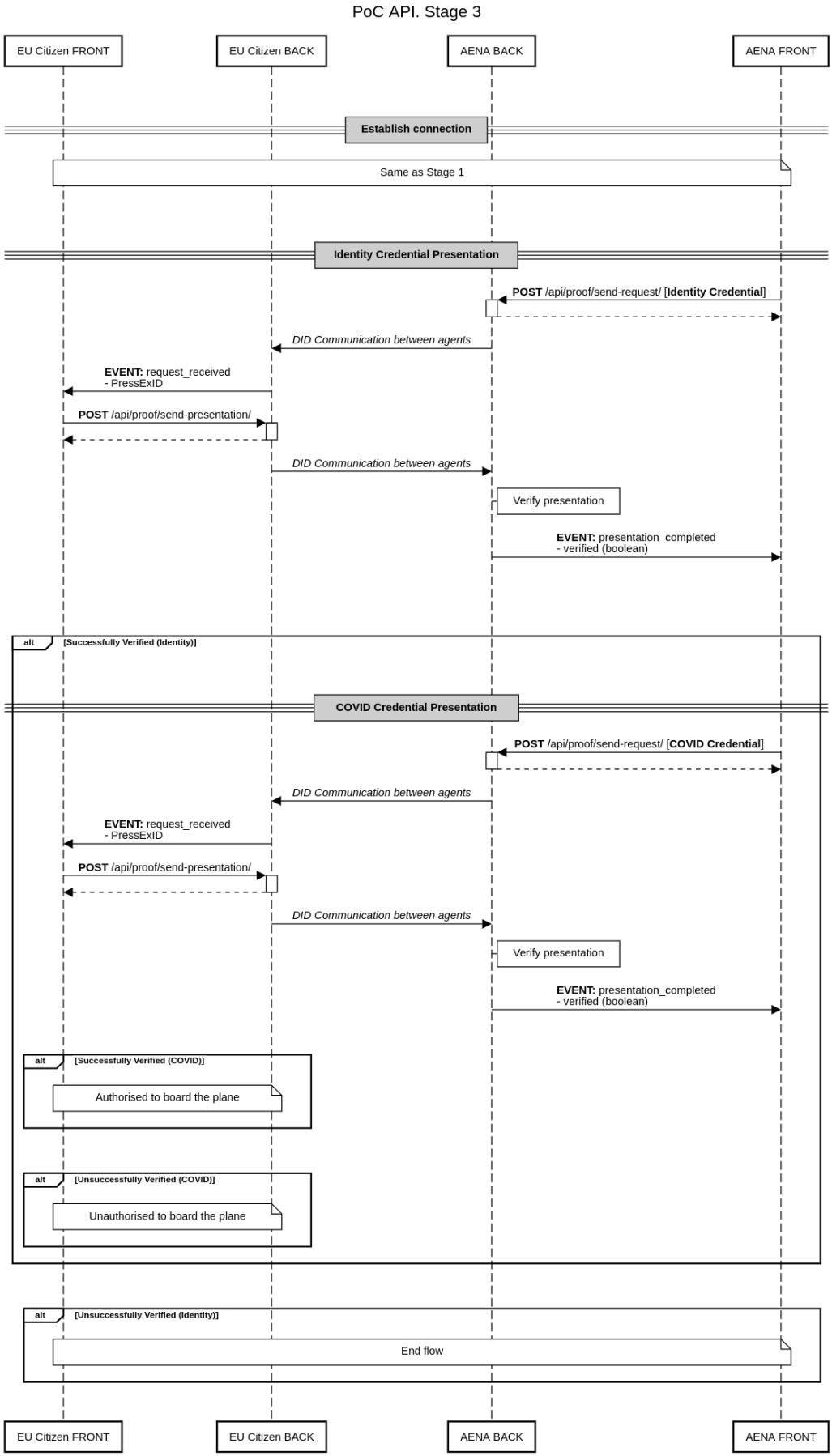

Figura 27: Diagrama de secuencia técnico simplificado de la etapa 3 de la PoC

Diagramas con un **nivel de detalle técnico algo más avanzado**. Se incluyen las llamadas que realiza el servidor al framework. Además, se muestran algunas de las interacciones más relevantes con el registro Indy y con el servidor Tails. Aun así, se han omitido o abstraído bastantes detalles.

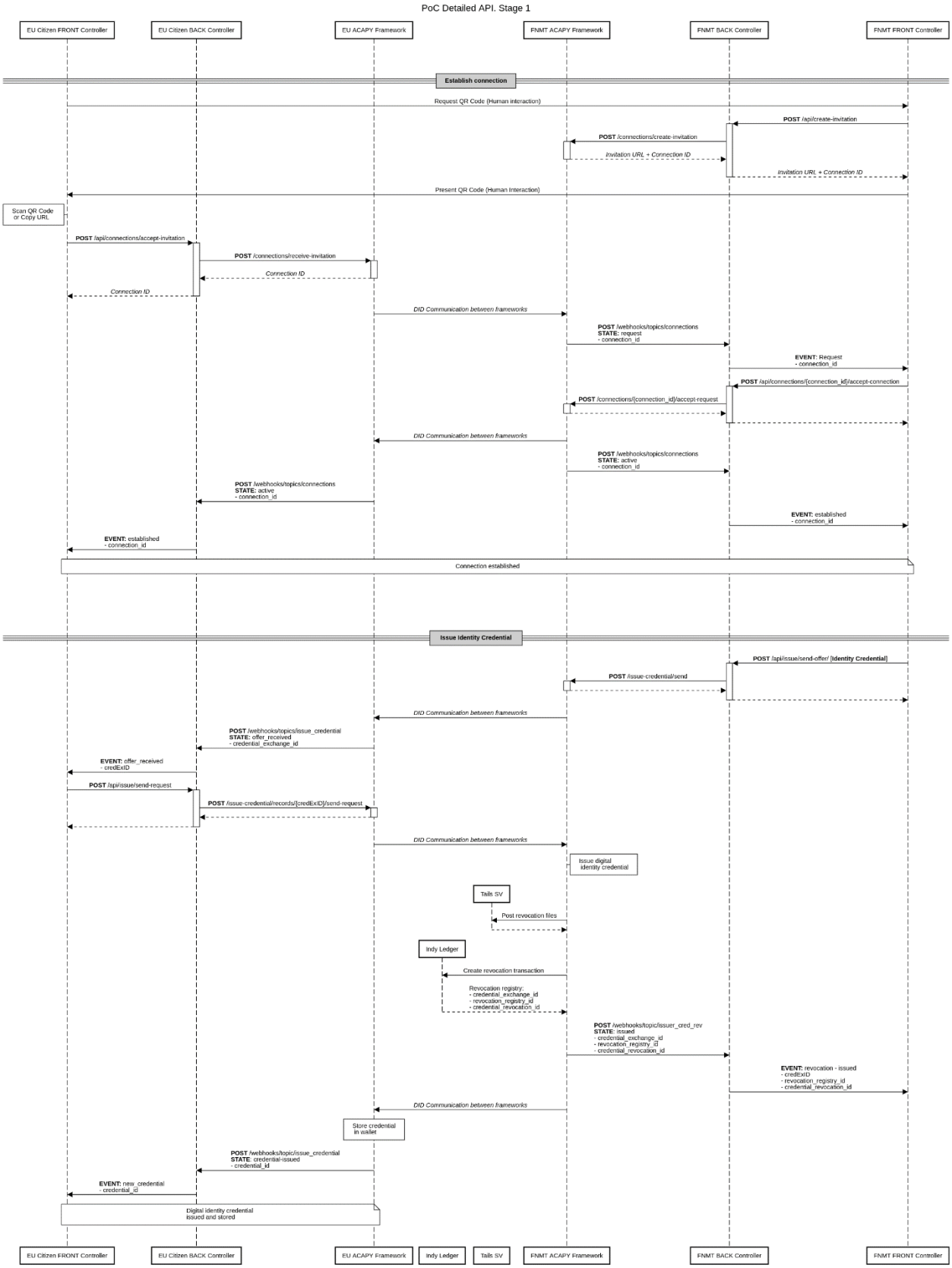

Figura 28: Diagrama de secuencia técnico avanzado de la etapa 1 de la PoC

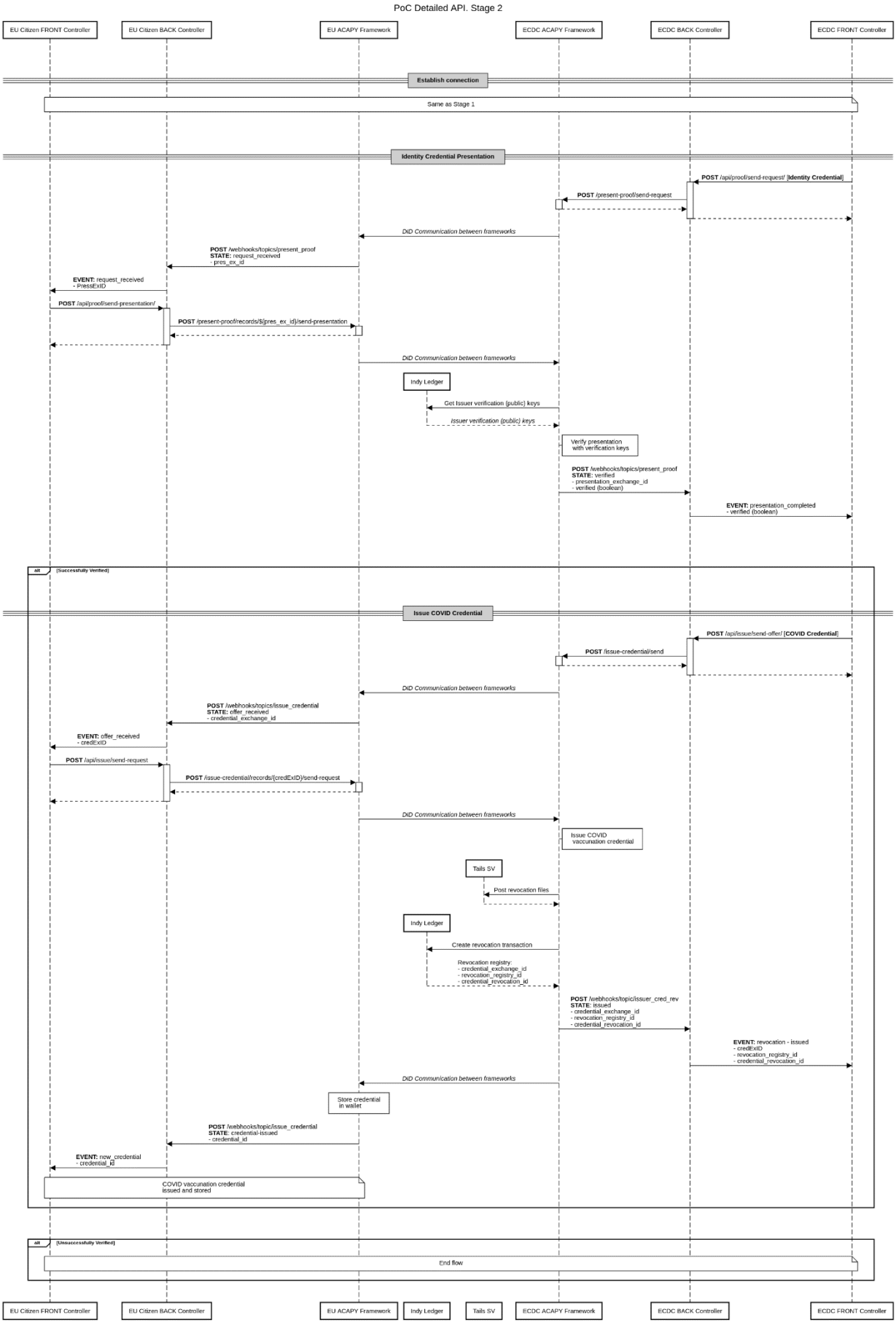

Figura 29: Diagrama de secuencia técnico avanzado de la etapa 2 de la PoC

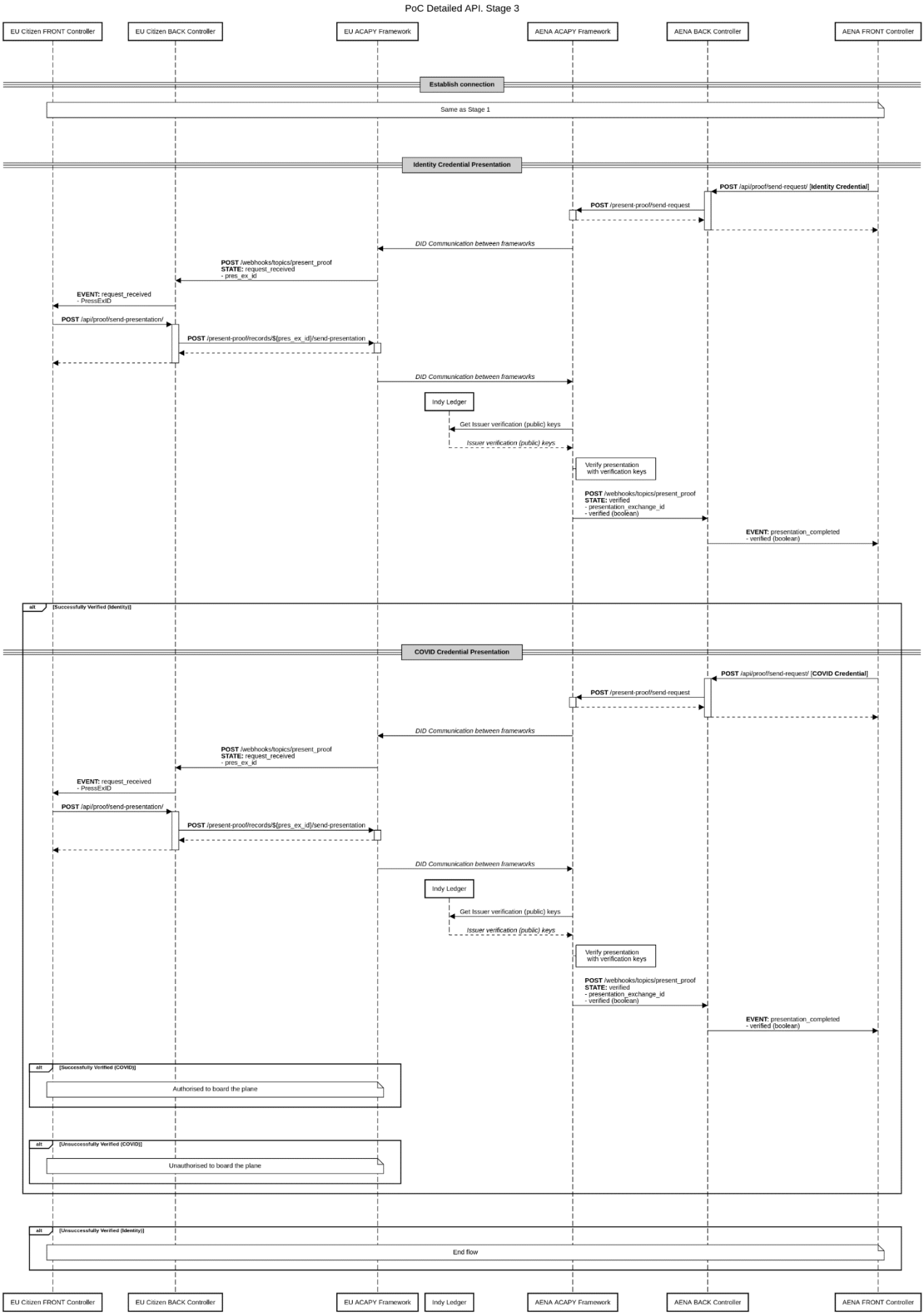

Figura 30: Diagrama de secuencia técnico avanzado de la etapa 3 de la PoC

ANEXO V: OPCIONES DE CONFIGURACIÓN DE ACA-PY

En este anexo se van a recoger las opciones existentes para la versión 0.75 de ACA-Py, que se han obtenido de ejecutar el comando "*aca-py start -h*".

Nótese que el objetivo de este anexo es informar al lector de las diferentes opciones que hay disponibles, y que no aporta ningún trabajo realizado por el propio alumno. El código que genera la siguiente configuración está disponible en el repositorio oficial de Aries Cloud Agent Python [66], principalmente en este enlace[28]. A fecha de publicación de este TFM, estas opciones no se encuentran disponibles en ninguna fuente oficial con un formato legible, es por esta razón que se han incluido directamente en el documento y no referenciado. Es importante destacar que esta práctica de distribución está permitida, según la licencia que han publicado con el repositorio, disponible en este enlace[29].

Las opciones son las siguientes:

```
usage: aca-py start [-h] [--admin <host> <port>] [--admin-api-key <api-key>]
                    [--admin-insecure-mode] [--no-receive-invites]
                    [--help-link <help-url>] [--webhook-url <url#api_key>]
                    [--admin-client-max-request-size ADMIN_CLIENT_MAX_REQUEST_SIZE]
                    [--debug] [--debug-seed <debug-did-seed>]
                    [--debug-connections] [--debug-credentials]
                    [--debug-presentations] [--invite] [--connections-invite]
                    [--invite-label <label>] [--invite-multi-use]
                    [--invite-public] [--invite-metadata-json <metadata-json>]
                    [--test-suite-endpoint <endpoint>] [--auto-accept-invites]
                    [--auto-accept-requests] [--auto-respond-messages]
                    [--auto-respond-credential-proposal]
                    [--auto-respond-credential-offer]
                    [--auto-respond-credential-request]
                    [--auto-respond-presentation-proposal]
                    [--auto-respond-presentation-request]
                    [--auto-store-credential] [--auto-verify-presentation]
                    [--auto-disclose-features]
                    [--disclose-features-list DISCLOSE_FEATURES_LIST]
                    [--arg-file ARG_FILE] [--plugin <module>]
                    [--block-plugin <module>] [--plugin-config PLUGIN_CONFIG]
                    [-o <KEY=VALUE> [<KEY=VALUE> ...]]
                    [--storage-type <storage-type>]
                    [-e <endpoint> [<endpoint> ...]]
                    [--profile-endpoint <profile_endpoint>]
                    [--read-only-ledger]
                    [--tails-server-base-url <tails-server-base-url>]
                    [--tails-server-upload-url <tails-server-upload-url>]
                    [--notify-revocation] [--monitor-revocation-notification]
                    [--ledger-pool-name <ledger-pool-name>]
                    [--genesis-transactions <genesis-transactions>]
                    [--genesis-file <genesis-file>]
                    [--genesis-url <genesis-url>] [--no-ledger]
                    [--ledger-keepalive LEDGER_KEEPALIVE]
```

[28] https://github.com/hyperledger/aries-cloudagent-python/blob/0.7.5/aries_cloudagent/config/argparse.py
[29] https://github.com/hyperledger/aries-cloudagent-python/blob/0.7.5/LICENSE

```
                    [--ledger-socks-proxy <host:port>]
                    [--genesis-transactions-list <genesis-transactions-list>]
                    [--accept-taa <acceptance-mechanism> <taa-version>]
                    [--log-config <path-to-config>] [--log-file <log-file>]
                    [--log-level <log-level>] [--auto-ping-connection]
                    [--auto-accept-intro-invitation-requests]
                    [--invite-base-url <base-url>] [--monitor-ping]
                    [--monitor-forward] [--public-invites] [--timing]
                    [--timing-log <log-path>] [--trace]
                    [--trace-target <trace-target>] [--trace-tag <trace-tag>]
                    [--trace-label <trace-label>]
                    [--preserve-exchange-records] [--emit-new-didcomm-prefix]
                    [--emit-new-didcomm-mime-type]
                    [--exch-use-unencrypted-tags] [--auto-provision]
                    [-it <module> <host> <port>] [-ot <module>] [-l <label>]
                    [--image-url IMAGE_URL]
                    [--max-message-size <message-size>]
                    [--enable-undelivered-queue]
                    [--max-outbound-retry MAX_OUTBOUND_RETRY]
                    [--ws-heartbeat-interval <interval>]
                    [--ws-timeout-interval <interval>]
                    [--mediator-invitation <invite URL to mediator>]
                    [--mediator-connections-invite] [--open-mediation]
                    [--default-mediator-id <mediation id>]
                    [--clear-default-mediator] [--seed <wallet-seed>]
                    [--wallet-local-did] [--wallet-allow-insecure-seed]
                    [--wallet-key <wallet-key>]
                    [--wallet-rekey <wallet-rekey>]
                    [--wallet-name <wallet-name>]
                    [--wallet-type <wallet-type>]
                    [--wallet-storage-type <storage-type>]
                    [--wallet-storage-config <storage-config>]
                    [--wallet-key-derivation-method <key-derivation-method>]
                    [--wallet-storage-creds <storage-creds>]
                    [--replace-public-did] [--recreate-wallet] [--multitenant]
                    [--jwt-secret <jwt-secret>] [--multitenant-admin]
                    [--multitenancy-config key=value [key=value ...]]
                    [--endorser-protocol-role <endorser-role>]
                    [--endorser-invitation <endorser-invitation>]
                    [--endorser-public-did <endorser-public-did>]
                    [--endorser-endorse-with-did <endorser-endorse-with-did>]
                    [--endorser-alias <endorser-alias>]
                    [--auto-request-endorsement] [--auto-endorse-transactions]
                    [--auto-write-transactions]
                    [--auto-create-revocation-transactions]
                    [--auto-promote-author-did]
```

Fragmento de código 11: Opciones de configuración del framework ACA-Py

No se han incluido las descripciones detalladas de cada una de las opciones, las cuales aparecen en la propia ayuda, ya que esto ocuparía cerca de 10 páginas. Para cada una de estas opciones, se indica también cual es el nombre de la variable de entorno por si se quiere configurar de esta manera. Por ejemplo, el argumento "—admin" se puede sustituir con la variable "ACAPY_ADMIN".

ANEXO VI: API DEL CONTROLADOR GENERAL

En este anexo se presentan las rutas implementadas del controlador general (gAp) usado tanto en la Prueba de Concepto como en el Producto Viable Mínimo.

El funcionamiento interno de los controladores de algunas rutas ha sido mejorado ligeramente en el MVP respecto a la PoC, pero las entradas (parámetros) y las salidas (respuestas) son las mismas.

VI.1 Estado

GET /api/status

Descripción: Recupera el estado del framework. Recibir una respuesta implica el buen funcionamiento y conectividad del controlador.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: *up / down*.

GET /api/status/controller

Descripción: Recupera el estado del controlador general.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: *up*.

GET /api/status/connectivity

Descripción: Recupera el estado de la conexión entre el servidor cliente y el controlador general. Se encarga de enviar un evento de estado al cliente con el tema “status”.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La respuesta que reciba del servidor (depende del cómo se haya programado su *webhook*).

VI.2 Conexiones

Ejemplo simplificado de una conexión (en la práctica aparecen más campos):

```
{
     “connection_id”: “5442645d-218b-4f5a-9b01-902c2518ebaf”, // Identificador de la conexión
     “state”: “request”,   // Estado de la conexión
     “their_label”: “ECDC Agent”,
     “created_at”: “2022-02-22T09:53:47.962068Z”,
     “their_role”: “inviter”,
}
```

Fragmento de código 12: Ejemplo simplificado de un objeto conexión recibido por el gAp

GET /api/connections

Descripción: Recupera todas las conexiones existentes.

Parámetros de consulta: *alias* (opcional, filtra las conexiones por este alias).

Parámetros de ruta: No.

Respuesta: Lista de conexiones.

GET /api/connections/active

Descripción: Recupera todas las conexiones existentes que estén activas (estados *active*, *response* o *completed*).

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: Lista de conexiones activas.

GET /api/connections/pending

Descripción: Recupera todas las conexiones existentes que estén pendientes de activarse (estados *invitation* o *request*).

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: Lista de conexiones pendientes.

GET /api/connections/{conn_id}

Descripción: Recupera la conexión especificada por su identificador.

Parámetros de consulta: No.

Parámetros de ruta: *conn_id* (identificador de la conexión).

Respuesta: La conexión espeficidada.

POST /api/connections/create-invitation

Descripción: El invitador crea una invitación de conexión. La conexión tendrá estado *invitation*.

Parámetros de consulta: *alias* (opcional, alias a establecer para la conexión).

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: La conexión creada.

POST /api/connections/accept-invitation

Descripción: El invitado acepta una invitación de conexión. La conexión pasará al estado *request*.

Parámetros de consulta: *alias* (opcional, alias a establecer para la conexión).

Parámetros de ruta: No.

Cuerpo de la petición: {invitation_url} // *URL generado al crear la invitación*

Respuesta: La conexión aceptada.

POST /api/connections/{conn_id}/accept-connection

Descripción: El invitador acepta la petición del invitado para establecer la conexión. La conexión pasará al estado *active*[30].

Parámetros de consulta: No.

Parámetros de ruta: *conn_id* (identificador de la conexión).

Cuerpo de la petición: {}

Respuesta: La conexión establecida.

DELETE /api/connections/{conn_id}

Descripción: Elimina la conexión identificada por su identificador.

Parámetros de consulta: No.

Parámetros de ruta: *conn_id* (identificador de la conexión).

Respuesta: El identificador de la conexión eliminada.

POST /api/connections/{conn_id}/send-message

Descripción: Envía un mensaje simple de texto a la conexión indicada por el identificador.

Parámetros de consulta: No.

Parámetros de ruta: *conn_id* (identificador de la conexión).

Cuerpo de la petición: {msg} // *Mensaje a enviar*

Respuesta: El mensaje enviado.

VI.3 Cartera

Ejemplo de un DID devuelto por ACA-Py:

```
{
   "did": "EcoK42WEosgKjQ4r3C6v5b",
   "verkey": "8RU62QppPLMFaetCBhwZMtJiGiKq1rfL4MSkQ6X4ELRH",
   "posture": "posted",
   "key_type": "ed25519",
   "method": "sov"
}
```

Fragmento de código 13: Ejemplo de un objeto DID recibido por el gAp

Ejemplo de un esquema de credencial simplificado devuelto por ACA-Py:

```
{
  "id": "EcoK42WEosgKjQ4r3C6v5b:2:covID_PoC_Identity:1.00",
  "name": "covID_PoC_Identity",
  "version": "1.00",
  "attrNames": ["full_name","birthday","birthday_epoch","address","dni_number","photo_url"]
}
```

Fragmento de código 14: Ejemplo de un esquema de credencial recibido por el gAp

[30] Se ha configurado el *framework* para que automáticamente cambie el estado desde *response* a *completed* y luego a *active*. Esta opción se llama "*auto-ping-connections*".

Ejemplo de una definición de credencial simplificada devuelta por ACA-Py:

```
{
   "id": "EcoK42WEosgKjQ4r3C6v5b:3:CL:12:covID_PoC_Identity_7hzYQ",
   "tag": "covID_PoC_Identity_7hzYQ",
   "related_schema": "EcoK42WEosgKjQ4r3C6v5b:2:covID_PoC_Identity:1.00"
   "value": {
      "primary": { … },     // Claves públicas asociadas a todos los atributos
      "revocation": { … }   // Información criptográfica
   }
}
```

Fragmento de código 15: Ejemplo de una definición de credencial recibido por el gAp

GET /api/wallet/dids

Descripción: Recupera todos los objetos DID de la cartera del agente (públicos y privados).

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La lista de DID.

GET /api/wallet/dids/public

Descripción: Recupera el objeto DID público de la cartera del agente.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: El objeto DID público.

GET /api/wallet/dids/{did}

Descripción: Recupera el objeto DID indicado de la cartera del agente.

Parámetros de consulta: No.

Parámetros de ruta: *did*.

Respuesta: El objeto DID indicado.

GET /api/wallet/schemas/created

Descripción: Recupera todos los esquemas de credencial publicados al VDR por el agente.

Parámetros de consulta: Los siguientes parámetros opcionales para filtrar el resultado:

- *schema_id*. Identificador del esquema.
- *schema_name*. Nombre del esquema.
- *schema_version*. Versión del esquema.

Parámetros de ruta: No.

Respuesta: La lista de esquemas.

GET /api/wallet/schemas/{schema_id}

Descripción: Recupera el esquema de credencial especificado por su identificador del VDR.

Parámetros de consulta: No.

Parámetros de ruta: *schema_id* (identificador del esquema).

Respuesta: El esquema indicado.

POST /api/wallet/credentials/schemas/

Descripción: Publica al VDR el esquema de la credencial incluida en el cuerpo de la petición.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición:

```
{
    schema_name,     // String que identifica el nombre del esquema. Ejemplo: covID_MVP_Identity
    schema_version,  // String que identifica la versión del esquema
    attributes: [attr1, attr2, …]    // Lista de String, con los nombres de los atributos
}
```

Respuesta: El esquema de credencial publicado.

GET /api/wallet/credentials/definitions/created

Descripción: Recupera todas las definiciones de credencial publicados al VDR por el agente.

Parámetros de consulta: Los siguientes parámetros opcionales para filtrar el resultado:

- *schema_id*. Identificador del esquema.
- *schema_name*. Nombre del esquema.
- *schema_version*. Versión del esquema.
- *cred_def_id*. Identificador de la definición.
- *issuer_did*. DID del emisor que la publicó.

Parámetros de ruta: No.

Respuesta: La lista de definiciones de credencial.

GET /api/wallet/credentials/{cred_def_id}

Descripción: Recupera la definición de credencial especificado por su identificador del VDR.

Parámetros de consulta: No.

Parámetros de ruta: *cred_def_id* (identificador de la definición).

Respuesta: La definición indicada.

POST /api/wallet/credentials/definitions/

Descripción: Publica al VDR la definición de la credencial incluida en la petición.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición:

```
{
    schema_id,   // String que identifica el esquema a emplear
    tag          // String que nombra la definición de credencial. Ejemplo: covID_MVP_Identity
}
```

Respuesta: La definición de credencial publicada.

GET /api/wallet/credentials

Descripción: Recupera todas las credenciales almacenadas en la cartera del agente.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La lista de credenciales.

GET /api/wallet/credentials/{credential_id}

Descripción: Recupera la credencial especificada por el identificador de la cartera del agente.

Parámetros de consulta: No.

Parámetros de ruta: *credential_id* (El identificador de la credencial)..

Respuesta: La credencial.

DELETE /api/wallet/credentials/{credential_id}

Descripción: El *holder* elimina la credencial especificada por el identificador de la cartera del agente.

Parámetros de consulta: No.

Parámetros de ruta: *credential_id* (El identificador de la credencial).

Respuesta: El identificador de la credencial eliminada.

VI.4 Expedición

Para el flujo de expedición de credenciales es necesario destacar que se ha establecido que el que inicia el proceso es el *issuer* al enviar una oferta de credencial al *holder*.

POST /api/issue/send-offer

Descripción: El *issuer* envía una oferta de credencial al agente especificado.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición:

```
{
    schema_id,      // Identificador del esquema a emplear
    comment,        // String que le aparecerá al holder y que detalla de qué credencial se trata
    connection_id,     // Identificador de la conexión con el holder
    attributes: [{name, value}, …]   // Lista de atributos
                                     // Para cada atributo se especifica su nombre y el valor
}
```

Respuesta: La oferta de credencial enviada.

POST /api/issue/send-request

Descripción: El *holder* acepta la oferta de credencial recibida por el *issuer*.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {cred_ex_id} // *El identificador temporal de expedición de credencial*

Respuesta: La confirmación de aceptación.

Nota respecto a este punto de acceso: El identificador temporal se obtiene del evento que recibe el *holder* cuando recibe una oferta de credencial.

GET /api/issue/records

Descripción: Recupera todos los registros temporales de expedición de credenciales.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La lista de registros.

GET /api/issue/records/{cred_ex_id}

Descripción: Recupera el registro temporal de expedición de credencial indicado.

Parámetros de consulta: No.

Parámetros de ruta: *cred_ex_id* (El identificador temporal de expedición de credencial).

Respuesta: El registro temporal de expedición indicado.

Estos dos últimos puntos de acceso no se han llegado a utilizar.

VI.5 Presentación

Para el flujo de verificación de credenciales es necesario destacar que se ha establecido que el que inicia el proceso es el *verifier* al enviar una petición de presentación de credencial al *holder*.

POST /api/proof/send-request

Descripción: El *verifier* envía una petición de presentación de credencial al *holder*.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición:

```
{
    name,      // Alias que se le asocia a la presentación (atributo opcional)
    comment,   // Comentario que le va a aparecer al holder con la solicitud de presentación
    connectionID,   // Identificador de la conexión con el holder
    cred_def_id,   // Identificador de la definición de credencial que el issuer va a emplear
    from,      // Fecha (tiempo Unix) inicio del rango temporal donde se prueba la no revocación
    to,        // Fecha final del rango. Si no se indica, se asigna el mismo valor que a "from".
    attributes: [attr1, attr2, …],   // Atributos que el issuer quiere que el holder desvele
    predicates: [{name, condition, comparisonValue}, …]  // Predicados ZKP.
            // Por cada predicado, se especifica su nombre, la condición y el valor a comparar.
}
```

Respuesta: La oferta enviada.

POST /api/proof/send-presentation

Descripción: El *holder* envía la presentación de credencial solicitada al *verifier*.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición:

```
{
    pres_ex_id,     // El identificador temporal de verificación de la presentación
    req_attributes: [{0_attr1_uuid}, ..],     // Lista de atributos solicitados asociados a un
                                              // Boolean que indica si se revela o no
```

```
    req_predicates: [0_pred1_predOperation_uuid, …] // Lista de predicados solicitados a probar
}
```

Respuesta: La presentación enviada.

Nota respecto a este punto de acceso: El identificador temporal se obtiene del evento que recibe el *holder* cuando recibe una solicitud de envío de presentación de credencial.

GET /api/proof/{pres_ex_id}

Descripción: El *verifier* recupera la prueba de verificación de la presentación.

Parámetros de consulta: No.

Parámetros de ruta: *pres_ex_id* (El identificador temporal de presentación).

Respuesta: La presentación completa.

Nota respecto a este punto de acceso: Esta prueba de verificación se recibe automáticamente a través de un evento, por lo que normalmente no hará falta llamar a este punto de acceso.

GET /api/proof/records

Descripción: Recupera todos los registros temporales de presentación.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La lista de registros.

GET /api/proof/records/{pres_ex_id}

Descripción: Recupera el registro temporal de expedición de presentación indicado.

Parámetros de consulta: No.

Parámetros de ruta: *pres_ex_id* (El identificador temporal de presentación).

Respuesta: El registro temporal de presentación indicado.

Estos dos últimos puntos de acceso no se han llegado a utilizar.

VI.6 Revocación

POST /api/revocation/revoke

Descripción: El *issuer* revoca una credencial que ha expedido a un *holder*.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: Hay dos alternativas:

```
{
    rev_reg_id,       // Identificador del registro de revocación
    cred_rev_id,      // Identifica la credencial expedida con el registro anterior
}
{
    cred_ex_id        // El identificador temporal de expedición de credencial
}
```

Respuesta: La confirmación de la revocación.

GET /api/revocation/status

Descripción: Recupera el estado de revocación de una credencial.

Parámetros de consulta: Dos opciones: *rev_reg_id* y *cred_rev_id*, o *cred_ex_id*.

Parámetros de ruta: No.

Respuesta: El estado de revocación.

GET /api/revocation/issued

Descripción: Recupera el número de credenciales expedidas con el registro de revocación indicado.

Parámetros de consulta: *reg_reg_id*.

Parámetros de ruta: No.

Respuesta: El número de credenciales expedidas.

ANEXO VII: EVENTOS DEL CONTROLADOR GENERAL

A través de las API **POST /webhooks/*topic*** implementadas en este controlador general (gAp) llegan eventos complejos del framework. En este servidor se tratan los eventos y se simplifican, con fin de enviarlos al servidor de negocio correspondiente a través de una API **POST /webhooks/** que debe estar implementada en este último.

Los eventos que se describen en este anexo son los simplificados que se envían al servidor de negocio. Además, muchos otros eventos que se reciben del framework no se envían al servidor cliente, sino que simplemente se muestran por el terminal

Los eventos tienen dos campos con los cuales es posible identificarlos, el primero es el campo *topic* que indica el tipo de evento, y el segundo es el campo *event_name* que indica de qué evento se trata.

VII.1 Evento de estado

Evento que se produce cuando el cliente llama a la ruta GET /api/status/connectivity y que permite asegurar que la conexión entre el servidor cliente y el controlador general funciona correctamente:

```
{
    topic: "status",
    event_name: "connectivity",
    status: "working"
}
```

Fragmento de código 16: Evento de estado (gAp)

VII.2 Eventos de conexiones

Eventos que se producen en el proceso de establecimiento de conexión. Evento:

```
{
    topic: "connections",
    event_name, // [request / established / error]
    error,              // Mensaje de error. Solo si event_name="error"
    payload: {
      connection_id,            // Identificador de conexión del agente
      connection_state, // Estado de la conexión
      connection_alias, // Nombre establecido para identificar la conexión
      more_details: {their_label, their_role, their_did, my_did, date} // Detalles extra
      }
    }
}
```

Fragmento de código 17: Eventos de conexiones (gAp)

Evento request. Evento enviado a un agente cuando el otro le ha enviado una petición de conexión tras aceptar la invitación. Con este evento, el agente que lo ha recibido se percata de que otro quiere establecer una conexión, con el fin de poder aceptarla.

Evento established. Cuando ambos agentes han establecido la conexión, a ambos les llega este evento. Con este evento a los agentes se les informa de que la conexión ya está establecida.

Evento error. Si ocurre un error en el proceso, se le notifica al agente.

VII.3 Eventos de mensajes básicos

Eventos que se producen en el proceso de intercambio de mensajes básicos. Evento:

```
{
    topic: "basicmessages",
    event_name  // [message_received / message_acked]
    payload: {
      connection_id,            // Identificador de conexión del agente
      message                   // Mensaje enviado. Solo si event_name="message_received"
    }
}
```

Fragmento de código 18: Eventos de mensajes básicos (gAp)

Evento message_received. Evento que contiene un mensaje enviado por otro agente.

Evento message_acked. Evento de confirmación que le llega al agente que ha enviado un mensaje cuando el otro agente lo recibe.

VII.4 Eventos de expedición

Eventos que se producen en el proceso de expedición de credenciales. Evento:

```
{
   topic: "issuecredential",
   event_name, // [offer_sent / offer_received / new_credential / credential_acked / error]
   error,      // Mensaje de error. Solo si event_name="error"
   payload: {
      connection_id,    // Identificador de conexión del agente
      state,            // Estado del proceso de expedición de la credencial
      credential_proposal : {    // Solo si event_name=
                                 // =("offer_received","offer_sent", "credential_acked")
         credential_exchange_id,  // Identificador temporal de expedición de credencial
         comment,                      // Comentario con el concepto de la credencial
         content: [{name, value}, …]   // Lista de atributos (el nombre y su valor)
      },
      credential : {    // Solo si event_name="new_credential"
         credential_id,        // Identificador de la credencial
         comment,              // Comentario con el concepto de la credencial
         content: {attr1: {raw, encoded}, attr2: {…}, …}    // Atributos desvelados.
      },
      more_details: {schema_id, credential_definition_id, role, date} // Detalles extra
   }
}
```

Fragmento de código 19: Eventos de expedición (gAp)

Evento offer_sent. Evento generado al mismo agente que ha enviado la oferta de credencial.

Evento offer_received. Evento recibido por el *holder* cuando llega la oferta de credencial del *issuer*. Con este evento, el *holder* puede decidir si aceptar o no la oferta en función del contenido del evento.

Evento *new_credential*. Evento que informa al *holder* que la credencial ya le ha sido expedida. Incluye los datos de dicha credencial, junto con el identificador de esta.

Evento *credential_acked*. Evento que informa al *issuer* que la credencial ha sido expedida con éxito.

Evento *error*. Si ocurre un error en el proceso, se le notifica al agente.

VII.5 Eventos de presentación

Eventos que se producen en el proceso de verificación de la presentación de credenciales. Evento:

```
{
    topic: "presentproof",
    event_name,         // [request_received  / presentation_completed / error]
    error,              // Mensaje de error. Solo si event_name="error"
    payload: {
       connection_id,          // Identificador de conexión del agente
       state,                  // Estado del proceso de verificación de la credencial
       presentation_request : {    // Solo si event_name="request_received"
          presentation_exchange_id,     // El identificador temporal de presentación
          comment,                      // Comentario con el concepto de la credencial
          presentation_name,           // Apodo de la presentación puesto por el verifier
          requested_attributes,         // Lista de atributos cuyo valor se quiere desvelar
          requested_predicates  // Lista de predicados (atributo, operación, umbral)
       },
       presentation : {            // Solo si event_name="presentation_completed"
          presentation_exchange_id,     // El identificador temporal de presentación
           presentation_name,           // Apodo de la presentación puesto por el verifier
          verified,             // Si ha sido verificado usando la definición de credencial
          revealed_attrs: {0_attr1_uuid: {raw, encoded}, …},  // Atributos revelados
          unrevealed_attrs,     // Lista de atributos que el holder ha decidido no revelar
          predicates,           // Pruebas de predicados ZKP
          identifiers: {schema_id, cred_def_id}// Esquema y definición empleados
       },
       more_details: {role, date} // Detalles extra
    }
}
```

Fragmento de código 20: Eventos de presentación (gAp)

Evento request_received. Evento recibido por el *holder* cuando le llega una petición de presentación de credencial.

Evento presentation_completed. Evento recibido por el *verifier* una vez se ha recibido la presentación de credencial del *holder*. Incluye los atributos y los predicados ZKP. A través del atributo *verified* se conoce si la credencial se ha verificado satisfactoriamente o no.

Evento *error*. Si ocurre un error en el proceso, se le notifica al agente.

VII.6 Eventos de revocación

Eventos que se producen en el proceso de revocación de credenciales. Evento:

```
{
    topic: "revocation",
    event_name        // [issued / revoked]
    payload: {
      credential_exchange_id,    // Identificador temporal de expedición de la credencial
      state,                     // Estado del proceso de revocación de la credencial
      revocation_details: {
        revocation_registry_id,  // Identificador del registro de revocación
       credential_revocation_id  // Identifica la credencial expedida con el registro anterior
      },
      more_details: {date}    // Detalles extra
    },
}
```

Fragmento de código 21: Eventos de revocación (gAp)

Evento issued. Evento recibido por el *issuer* cuando expide una credencial a un *holder*. Contiene los identificadores necesarios para luego poder revocar la credencial.

Evento revoked. Evento recibido por el *issuer* una vez la credencial correspondiente ha sido revocada.

VII.7 Eventos de registros de revocación

Evento que se produce cuando un *issuer* publica en el VDR una definición de credencial, generando un fichero tails en el servidor correspondiente y publicando dicho registro en el VDR. Evento:

```
{
    topic: "revocationregistry",
    event_name: "active",
    payload: {
      revocation_registry_id,   // Identificador del registro de revocación
       cred_def_id,            // Definición de credencial asociada
      state,                   // Estado del proceso de revocación de la credencial
      max_cred_num,     // Número máximo de credenciales que permite expedir este registro
      more_details: {date, issuer_did}         // Detalles extra del evento
    },
}
```

Fragmento de código 22: Eventos de registros de revocación (gAp)

Evento active. Evento recibido por el *issuer* una vez el registro se ha publicado en el VDR y ha pasado a estar activo.

ANEXO VIII: RUTAS ESPECÍFICAS DE LA PRUEBA DE CONCEPTO

En este anexo se presentan las rutas implementadas en la Prueba de Concepto. Realmente, en la PoC existe un único servidor que contiene las API presentadas a continuación y las presentadas en el ANEXO VI: API del controlador general.

En el MVP se eliminarán las rutas de este anexo del controlador general (gAp).

VIII.1 Rutas del controlador de ECDC

POST /api/wallet/credentials/schemas/vaccination

Descripción: Publica al VDR el esquema de credencial de vacunación COVID, obtenido del fichero JSON almacenado en la carpeta "/material".

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: El esquema de credencial publicado.

POST /api/wallet/credentials/definitions/vaccination

Descripción: Publica al VDR la definición de credencial de vacunación.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {schema_id} // Identificador del esquema de vacunación COVID a emplear

Respuesta: La definición de credencial publicada.

VIII.2 Rutas del controlador de FNMT

POST /api/wallet/credentials/schemas/identity

Descripción: Publica al VDR el esquema de credencial de identidad.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: El esquema de credencial publicado.

POST /api/wallet/credentials/definitions/identity

Descripción: Publica al VDR la definición de credencial de identidad.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {schema_id} // Identificador del esquema de identidad a emplear

Respuesta: La definición de credencial publicada.

ANEXO IX: NORMALIZACIÓN DE LA BASE DE DATOS (RUC19)

En este anexo se detalla el proceso seguido para normalizar y mejorar las tablas de la base de datos que aparece en la arquitectura del agente RUC19.

En primer lugar, se realizó un diseño preliminar con todos los datos que se debían almacenar:

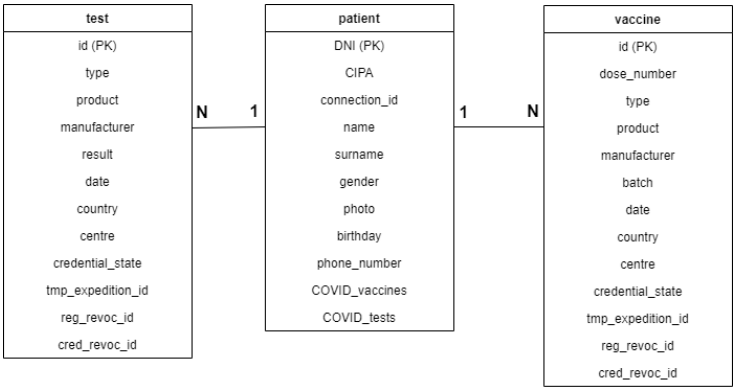

Figura 31: Diseño original de las tablas de la base de datos de RUC19 (MVP)

Lo siguiente fue empezar a normalizar la base de datos para que cumpliese la primera forma normal. Para ello, se cambió de manera de implementar la relación 1-N entre los pacientes, y las vacunas y test. De esta forma, la referencia la tienen los test y las vacunas del paciente, y no al revés. Así, todos los valores pasarían a ser atómicos en los dominios de cada relación.

Al hacer esto, dada la naturaleza de los datos y del diseño original, también pasó a cumplir la segunda forma normal, ya que todos los atributos que no son clave forman parte de toda la clave primaria, en todas las tablas. El proceso resultó en el siguiente diseño:

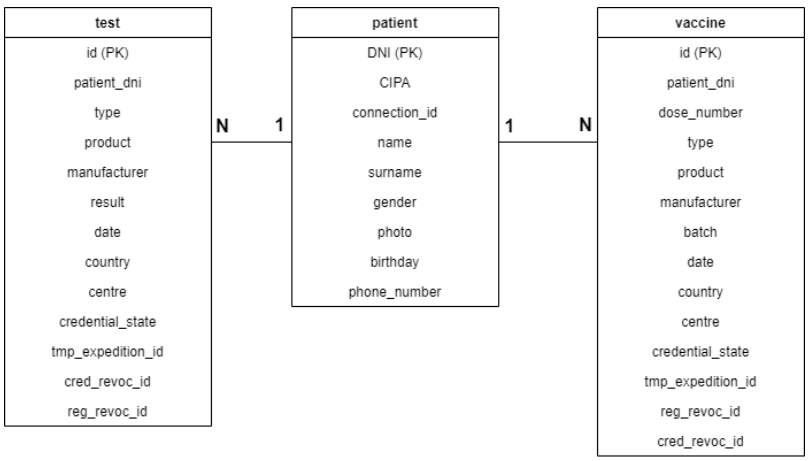

Figura 32: Diseño 2FN de las tablas de la base de datos de RUC19 (MVP)

Tras esto, se propuso cumplir con la tercera norma formal al evitar que ciertos atributos no primarios se pudiesen determinar a través de otros atributos no primarios. Esto ocurría en las tablas *test* y *vaccine* en dos situaciones:

- Columnas *country* y *centre*, al poder determinarse el país a partir del centro médico.
- Columnas *product*, *type* y *manufacturer*, al poder determinarse el tipo de producto y el fabricante a partir del producto.

Para evitar esta situación, se crearon tablas independientes para cada uno de estos casos. Para el caso de los centros, se creó una única tabla, ya que ambas tablas (*test* y *vaccine*) contenían valores dentro del mismo dominio (centros hospitalarios autorizados). Sin embargo, para el caso de los productos, se han creado dos tablas, ya que los datos que contienen son de diferentes dominios (pruebas y vacunas).

Al hacer estos cambios, se pasa a cumplir con el nivel BCNF, ya que todos los atributos que determinan otros atributos son claves primarias. Adicionalmente, pasó automáticamente a cumplir la cuarta y quinta forma normal. El diseño resultante fue el siguiente:

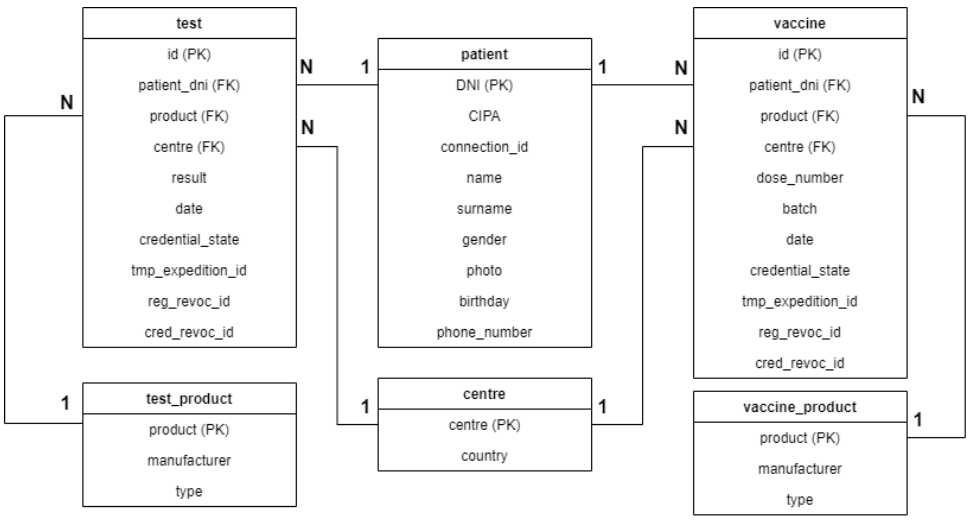

Figura 33: Diseño 5FN de las tablas de la base de datos de RUC19 (MVP)

Por otra parte, se mejoró el diseño al añadir en todas las tablas identificadores numéricos ajenos a la lógica de negocio que actúan como claves primarias. Esto permite mayor facilidad de consultas SQL al juntar tablas, por ejemplo, al evitar tener que escribir una cadena de texto como clave para el caso del centro hospitalario, lo que puede llevar a errores de escritura. El diseño resultante fue el que se presentó en el diseño de la base de datos.

ANEXO X: DISEÑO DE LA INTERFAZ DEL AGENTE RUC19

En este anexo se presenta el resultado final del diseño realizado para la interfaz web del agente RUC19.

Página "Home".

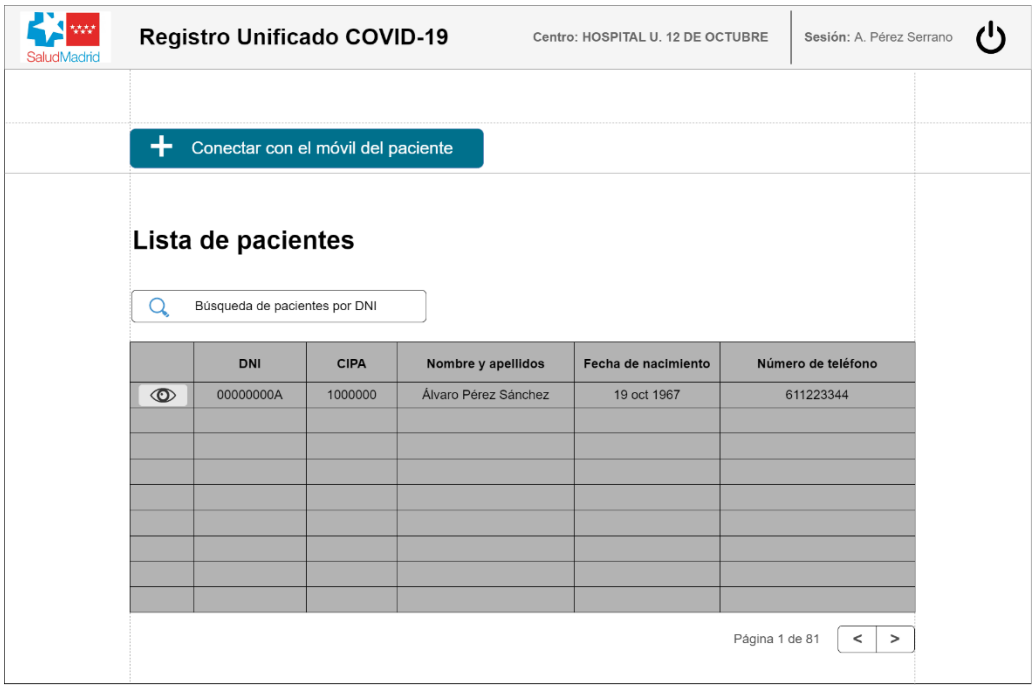

Figura 34: Captura del diseño de la interfaz RUC19 - Página Home

En la parte superior de la página aparece el nombre de un centro y del usuario (enfermero o personal administrativo), los cuales estarán escritos manualmente en el código del servidor.

El botón superior ("Conectar con el móvil del paciente") llevará a la página *Connection*.

El botón de la primera columna de cada fila llevará a la página *Patient*.

La tabla deberá ser una tabla interactiva que permita búsqueda, ordenación y paginación.

Página "Connection".

Figura 35: Captura del diseño de la interfaz RUC19 - Página Connection

Al acceder a esta página, el servidor creará una invitación de conexión automáticamente y el frontend transformará la URL a un código QR. Al escanear el código QR, el backend reenviará el evento recibido del frontend con el DNI del paciente, de forma que el frontend navegará automáticamente a la página *Patient*. Internamente el backend guardará en la base de datos, dentro del perfil correspondiente en la tabla "*patient*", el identificador descentralizado de la conexión Aries.

Página "Patient". Esta página refleja el espacio de cada paciente. Se compone de dos partes, una primera a la izquierda, donde aparecen los datos personales del paciente; y una segunda a la derecha, que mostrará una de las siguientes pestañas.

En la parte izquierda aparecerá un botón ("Conectar con el móvil del paciente") que llevaría al usuario a la página *Connection*. Si la conexión ya estuviese establecida con el paciente, el botón se cambiará por un texto informativo que informa de ello. En algunas de las capturas siguientes aparecerá el caso donde el paciente no ha establecido aún una conexión y el caso donde sí lo haya hecho.

Pestaña: *Datos de vacunación*.

Figura 36: Captura del diseño de la interfaz RUC19 - Página Patient – Pestaña Vacunación

El botón superior de la sección izquierda ("Nueva vacunación") redirige a la pestaña *Nueva Vacuna*.

En este caso el paciente no tiene establecida una conexión con el hospital, por lo que el enfermero o administrativo no le podrá expedir credenciales, tal y como se refleja en los botones de la derecha de la tabla al tener un color grisáceo. Si existiese una conexión previa, los botones pasarían a estar activos.

Esta interfaz abstrae todos los conceptos relacionados con las credenciales verificables y DID. Por ejemplo, si se le diese click al botón de expedir una credencial, sería el backend el que se encargase de gestionar la interacción con el framework y de almacenar en la base de datos el identificador temporal de expedición (*tmp_expedition_id*) necesario para completar el proceso.

Los botones de la izquierda de la tabla permiten visualizar en detalle cada vacuna, de forma que al darle click aparece un modal en página como el siguiente:

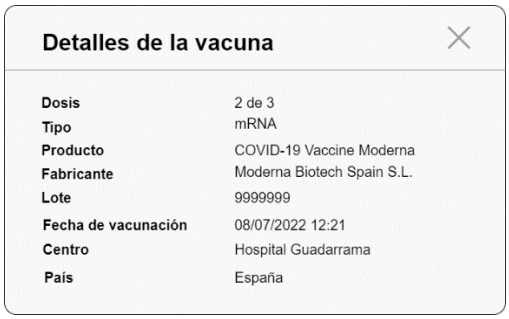

Figura 37: Captura del diseño de la interfaz RUC19 - Página Patient – Modal Vacunación

Pestaña: *Datos de pruebas*.

Figura 38: Captura del diseño de la interfaz RUC19 - Página Patient – Pestaña Test

El botón superior de la sección izquierda (“Nueva prueba”) redirige a la página *Nuevo Test*.

En este caso el paciente sí tiene establecida una conexión con el hospital, por lo que el enfermero o administrativo le podrá expedir credenciales, tal y como se refleja en el primer botón de la derecha de la tabla al tener un color azulado. También se muestra los casos en los que una credencial se está expidiendo y cuando está ya expedida, pudiendo revocarse.

Los botones de la izquierda de la tabla permiten visualizar en detalle cada prueba, de forma que al darle click aparece un modal en página como el siguiente:

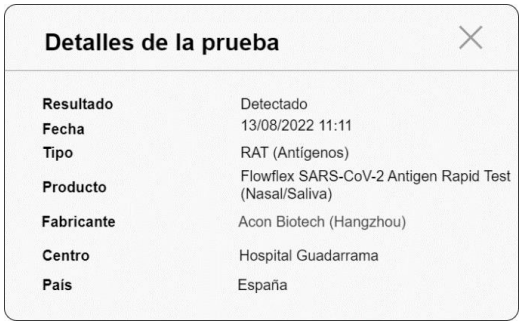

Figura 39: Captura del diseño de la interfaz RUC19 - Página Patient – Modal Test

Pestaña: *Nueva vacuna*.

Figura 40: Captura del diseño de la interfaz RUC19 – Página Patient - Pestaña Nueva Vacuna

El producto será un desplegable con todos los valores de productos de vacunas COVID-19 y el campo de fecha de vacunación aparecerá por defecto con valor del día actual, pero este valor se podrá editar.

Hay ciertos detalles que se rellenan automáticamente en el backend antes de guardar la vacuna en la base de datos:

- *centre_id*. Se rellenará automáticamente con la clave de la tabla *centre* que relaciona con el valor "Hospital Universitario 12 de Octubre".
- *dose_number*. Se rellenará con un valor incremental, respecto al valor más alto de este campo de las vacunas que tiene el paciente.

Por tanto, el sanitario únicamente tendrá que rellenar los valores de producto y de lote, a menos que tenga que cambiar el valor de la fecha de vacunación.

Pestaña: *Nuevo Test*.

Figura 41: Captura del diseño de la interfaz RUC19 - Página Patient – Pestaña Nuevo Test

El campo de fecha de realización de la prueba aparecerá por defecto con valor del día actual, pero este valor se podrá editar y el campo del resultado aparecerá por defecto con el valor “No detectado”.

El campo *centre_id* se rellenará automáticamente en el backend antes de guardar la vacuna en la base de datos con la clave de la tabla *centre* que relaciona con el valor “Hospital Universitario 12 de Octubre”.

Por tanto, el sanitario únicamente tendrá que rellenar el valor del producto, a menos que tenga que cambiar el valor de la fecha de realización de la prueba o editar el resultado de la prueba.

ANEXO XI: DISEÑO DE LA INTERFAZ DEL AGENTE IBERIA

En este anexo se presenta el resultado final del diseño realizado para la interfaz web del agente Iberia.

Página "QRCode".

Figura 42: Captura del diseño de la interfaz Iberia - Página QRCode

El logo del Ministerio de Sanidad indicaría en producción que la aplicación estaría auditada por el gobierno.

El botón ("Regenerar código") permite regenerar el código QR de la invitación de conexión. Esto está para poder actualizar el código por si hubiese un error con el anterior. Al igual que con la aplicación RUC19, este código se genera automáticamente al acceder a la página.

Al escanear el código, el *holder* recibirá de esta aplicación varias peticiones de presentación de las credenciales, tal y como se mencionó en el análisis del escenario. Una vez el titular acepte todas las presentaciones, le redirigirá a la siguiente página (*Result*).

Página "Result". Esta página muestra dos cosas: la foto del documento de identidad del ciudadano, con el fin de que el vigilante de seguridad se asegure de que el ciudadano no está suplantando otra identidad; y un indicador que muestra si el ciudadano cumple con todos los requisitos sanitarios.

Esta primera captura muestra el caso en el que el ciudadano cumple con los requisitos:

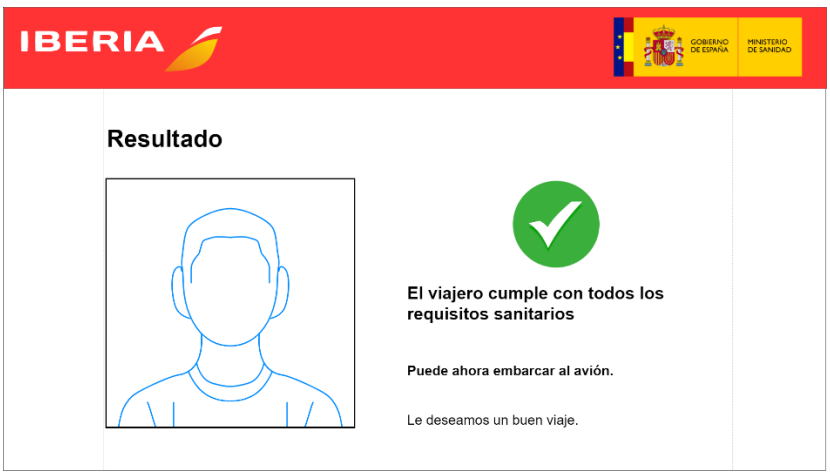

Figura 43: Captura del diseño de la interfaz Iberia - Página Result – Paso permitido

Y esta segunda captura muestra el caso en el que el ciudadano no cumple con los requisitos:

Figura 44: Captura del diseño de la interfaz Iberia - Página Result – Paso no permitido

ANEXO XII: AYUDA DEL *SCRIPT* DE DESPLIEGUE (MVP)

A continuación se muestra la ayuda generada por el *script* "manage" de despliegue del MVP:

```
Usage: $0 [command] [options]
Commands:
   prepare|build - Build the docker images AND install al the NPM dependencies.

   start|up - Creates the application containers from the built images and starts the
              services based on the docker-compose.yml files.
       [-v] [--verbose] - Start up in verbose mode (default: background mode)
       [-l] [--ledger] - Specify the ledger to use. Values:
            - [local] (DEFAULT). Deploy locally a VON ledger.
            - [bcovrin]. Use the public BCovrin [DEV] ledger.
            - [sovrin]. Use the public SOVRIN [Builder] ledger.
       [-a] [--agents] - Specify which agent(s) you want to deploy, by adding:
                         [MI], [MS], [RUC19], [IBERIA], [WALLET], [TEST]
       [-nr] [--no-revocation] - If you want to disabled revocation proofs.
       [-d] [--demo] - Specify the complexity of the demo (which credentials IBERIA is
                       going to request). Values:
            - [simple]. Identity and vaccination credentials.
            - [default] (DEFAULT). Identity, test and vaccination credentials.
            - [complex]. Identity, test, recovery and vaccination credentials.
       [-h] [--holder] - Specify what holder you are going to use. Values:
            - [test]. Will use a server (API) based agent. Use with Postman or with the
                      CLI Test agent (see the startTEST option).
            - [own] (DEFAULT). Will use the application that was developed for the project.
            - [third]. Will use third party wallets.

   main|deployall|upall|startall - Creates the application containers and print all the
        container logs in different tabs. NOTE: Only works with gnome-terminal

   startWALLET|wallet - Starts the EXPO Application in other terminal tab.
                        NOTE: Only works with gnome-terminal

   startTEST|testcli - Starts the TEST CLI client in the same terminal.

   logs - Display the logs from the docker compose run (ctrl-c to exit). It is optional,
          but we can also specify which framework, by adding:
          [MI], [MS], [RUC19], [IBERIA], [WALLET], [TEST]
       [-f] [--framework] - Print the logs only of the frameworks.
       [-c] [--controller] - Print the logs only of the backend controllers.
       [-b] [--business] - Print the logs only of the business backend.
       [-w] [--webapp] - Print the logs only of the web pages.
       [-d] [--db] - Print the logs only of the databases.
       [-t] [--tails] - Print the logs only of the tail server.
       [-n] [--ngrok] - Print the logs only of the ngrok tunnels.
       [-a] [--all] - Print the logs of all containers. (DEFAULT OPTION).
        Example: ./manage logs -c RUC19 ---> This will display the logs of the RUC19 backend
                 Controller

   mainlogs|logsall - Print all the container logs in different tabs.
        NOTE: Only works with gnome-terminal

   stop - Stops the services.  This is a non-destructive process (the volumes and containers
```

```
          are not deleted so they can be restarted).).
        [-p] [--public] - Doesn't stop the INDY network, since it will use the public service.

   resume|restart - Resume stopped services (agents, VON and Tails SV)
      [-p] [--public]- Doesn't resume the INDY network, since it will use the public service.

   mainresume|mainrestart - Resume the stopped services and print the container logs in
                            different tabs

   down|rm - Brings the agents down.

   startvon - Start a local VON network.


   vondown - Brings down the local VON network.

   vonrebuild - Rebuild the local VON network's docker images.

   webrestart|webre - Restart the web containers. Useful when a container never finishes
                      compiling.

   setInitialState|initialState|initialstate - Set the system's initial state. This is:
     - Publish all the credential schemas:
          MI (identity), MS (authorization, vaccination, test, recovery)
     - Publish all the credential definitions:
          MI (identity), MS (authorization), RUC19 (vaccination, test, recovery)
     - Connect MI with the CITIZEN agent and MS with RUC19
     - Issue identity credential from MI to CITIZEN
     - Issue authorization credential from MS to RUC19.
     Options:
         [--public] - Initial state with sovrin builder network (or test bcovrin network)
                      and the holder is a wallet application
         [--local] - Initial state with local VON and TEST / own developed agent
         [--schemas] - Publish all the credential schemas.
         [--definitions] - Publish all the credential definitions.

NOTE: You can override the default values of the ENV variables by passing them before calling
the script, for example:
    - LEDGER URL=http://test.bcovrin.vonx.io/ ./manage start
```

Fragmento de código 23: Ayuda del script manage (MVP)

ANEXO XIII: TABLAS DE LA BASE DE DATOS (SQL-DDL)

A continuación se muestra la declaración de las tablas de la base de datos MySQL integrada en el MVP, donde se emplea el lenguaje de definición de datos de SQL (SQL-DDL) para ello.

Tabla "patient":

```
CREATE TABLE IF NOT EXISTS patient (
        id INTEGER AUTO_INCREMENT NOT NULL UNIQUE,
        DNI CHAR(9) NOT NULL UNIQUE,
        CIPA CHAR(10),
        connection_id VARCHAR(256),
        name VARCHAR(256) NOT NULL,
        surname VARCHAR(256) NOT NULL,
        gender ENUM('male', 'female', 'other') NOT NULL,
        photo VARCHAR(2083),
        birthday DATE,
        phone_number VARCHAR(20),
        PRIMARY KEY(id),
        CONSTRAINT dni_format CHECK (DNI REGEXP '^[0-9]{8}[a-zA-Z]$'),
        CONSTRAINT birthday_gt_1900 CHECK (birthday >= DATE('1900-01-01'))
);
```

Fragmento de código 24: Declaración tabla "patient" de la base de datos (MVP)

Tabla "vaccine":

```
CREATE TABLE IF NOT EXISTS vaccine (
        id INTEGER AUTO_INCREMENT NOT NULL UNIQUE,
        patient id INTEGER,
        product id INTEGER NOT NULL,
        centre id INTEGER NOT NULL,
        dose number SMALLINT(8) NOT NULL,
        batch VARCHAR(256) NOT NULL,
        date DATETIME NOT NULL,
        credential state ENUM('not issued', 'issued', 'revoked') NOT NULL,
        tmp expedition id VARCHAR(256) DEFAULT NULL,
        reg revoc id VARCHAR(256) DEFAULT NULL,
        cred revoc id VARCHAR(256) DEFAULT NULL,
        PRIMARY KEY(id),
        CONSTRAINT fk vaccine patient FOREIGN KEY (patient id)
              REFERENCES patient(id) ON UPDATE CASCADE ON DELETE SET NULL,
        CONSTRAINT fk_vaccine_product FOREIGN KEY (product_id)
              REFERENCES vaccine_product(id) ON UPDATE CASCADE ON DELETE RESTRICT,
        CONSTRAINT fk_vaccine_centre FOREIGN KEY (centre_id)
              REFERENCES centre(id) ON UPDATE CASCADE ON DELETE RESTRICT,
        CONSTRAINT dose_gt_zero CHECK (dose_number > 0),
        CONSTRAINT vaccine_date_gt_epoch CHECK (date >= DATE('1970-01-01')),
        INDEX (patient_id, product_id, centre_id)
);
```

Fragmento de código 25: Declaración tabla "vaccine" de la base de datos (MVP)

Tabla “test”:

```
CREATE TABLE IF NOT EXISTS test (
	id INTEGER AUTO_INCREMENT NOT NULL UNIQUE,
	patient_id INTEGER,
	product_id INTEGER NOT NULL,
	centre_id INTEGER NOT NULL,
	result ENUM('detected', 'not detected') NOT NULL,
	date DATETIME NOT NULL,
	credential_state ENUM('not issued', 'issued', 'revoked') NOT NULL,
	tmp_expedition_id VARCHAR(256) DEFAULT NULL,
	reg_revoc_id VARCHAR(256) DEFAULT NULL,
	cred_revoc_id VARCHAR(256) DEFAULT NULL,
	PRIMARY KEY(id),
	CONSTRAINT fk_test_patient FOREIGN KEY (patient_id)
		REFERENCES patient(id) ON UPDATE CASCADE ON DELETE SET NULL,
	CONSTRAINT fk_test_product FOREIGN KEY (product_id)
		REFERENCES test_product(id) ON UPDATE CASCADE ON DELETE RESTRICT,
	CONSTRAINT fk_test_centre FOREIGN KEY (centre_id)
		REFERENCES centre(id) ON UPDATE CASCADE ON DELETE RESTRICT,
	CONSTRAINT test_date_gt_epoch CHECK (date >= DATE('1970-01-01')),
	INDEX (patient_id, product_id, centre_id)
);
```

Fragmento de código 26: Declaración tabla “test” de la base de datos (MVP)

Tabla “centre”:

```
CREATE TABLE IF NOT EXISTS centre (
	id INTEGER AUTO_INCREMENT NOT NULL UNIQUE,
	centre VARCHAR(256) NOT NULL,
	country CHAR(3) NOT NULL,
	PRIMARY KEY(id),
	CONSTRAINT country_iso_format CHECK (LENGTH(country) = 3)
);
```

Fragmento de código 27: Declaración tabla “centre” de la base de datos (MVP)

Tabla “vaccine_product”:

```
CREATE TABLE IF NOT EXISTS vaccine_product (
	id INTEGER AUTO_INCREMENT NOT NULL UNIQUE,
	product VARCHAR(256) NOT NULL,
	manufacturer VARCHAR(256) NOT NULL,
	type ENUM('mRNA', 'antigen', 'viral vector', 'inactivated', 'other') NOT NULL,
	PRIMARY KEY(id)
);
```

Fragmento de código 28: Declaración tabla “vaccine_product” de la base de datos (MVP)

Tabla “test_product”:

```
CREATE TABLE IF NOT EXISTS test_product (
	id INTEGER AUTO_INCREMENT NOT NULL UNIQUE,
	product VARCHAR(256) NOT NULL,
	manufacturer VARCHAR(256),
	type ENUM('NAAT', 'RAT') NOT NULL,
	PRIMARY KEY(id)
);
```

Fragmento de código 29: Declaración tabla “test_product” de la base de datos (MVP)

ANEXO XIV: API DEL PRODUCTO VIABLE MÍNIMO

En este anexo se presentan las rutas implementadas en los servidores de negocio del Producto Viable Mínimo (MVP). En las API de cada uno de estos servidores se realizan peticiones a las API del controlador general, presentadas en el ANEXO VI: API del controlador general.

XIV.1 Ministerio del Interior

GET /schemas/identity

Descripción: Recupera el esquema de credencial de identidad del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: El esquema de credencial.

POST /schemas/identity

Descripción: Publica al VDR el esquema de credencial de identidad.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: El esquema de credencial publicado.

GET /definitions/identity

Descripción: Recupera la definición de credencial de identidad del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La definición de credencial.

POST /definitions/identity

Descripción: Publica al VDR la definición de credencial de identidad.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {schema_id} // *String que identifica el esquema a emplear*

Respuesta: La definición de credencial publicada.

XIV.2 Ministerio de Sanidad

GET /schemas/authorization

Descripción: Recupera el esquema de credencial de autorización del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: El esquema de credencial.

POST /schemas/authorization

Descripción: Publica al VDR el esquema de credencial de autorización.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: El esquema de credencial publicado.

GET /schemas/vaccination

Descripción: Recupera el esquema de credencial de vacunación del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: El esquema de credencial.

POST /schemas/vaccination

Descripción: Publica al VDR el esquema de credencial de vacunación.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: El esquema de credencial publicado.

GET /schemas/test

Descripción: Recupera el esquema de credencial de test del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: El esquema de credencial.

POST /schemas/test

Descripción: Publica al VDR el esquema de credencial de test.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: El esquema de credencial publicado.

GET /schemas/recovery

Descripción: Recupera el esquema de credencial de recuperación del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: El esquema de credencial.

POST /schemas/recovery

Descripción: Publica al VDR el esquema de credencial de autorización.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: El esquema de credencial publicado.

GET /definitions/authorization

Descripción: Recupera la definición de credencial de autorización del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La definición de credencial.

POST /definitions/authorization

Descripción: Publica al VDR la definición de credencial de autorización.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {schema_id} // *String que identifica el esquema a emplear*

Respuesta: La definición de credencial publicada.

XIV.3 RUC19

XIV.3.1 Definiciones de credencial

GET /definitions/vaccination

Descripción: Recupera la definición de credencial de vacunación del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La definición de credencial.

POST /definitions/vaccination

Descripción: Publica al VDR la definición de credencial de vacunación COVID.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {schema_id} // *String que identifica el esquema a emplear*

Respuesta: La definición de credencial publicada.

GET /definitions/test

Descripción: Recupera la definición de credencial de test del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La definición de credencial.

POST /definitions/test

Descripción: Publica al VDR la definición de credencial de test COVID.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {schema_id} // *String que identifica el esquema a emplear*

Respuesta: La definición de credencial publicada.

GET /definitions/recovery

Descripción: Recupera la definición de credencial de recuperación del VDR.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: La definición de credencial.

POST /definitions/recovery

Descripción: Publica al VDR la definición de credencial de recuperación COVID.

Parámetros de consulta: No.

Parámetros de ruta: La definición de credencial publicada.

Cuerpo de la petición: {schema_id} // *String que identifica el esquema a emplear*

Respuesta: No.

XIV.3.2 Pacientes

GET /patients

Descripción: Recupera todos los pacientes de la base de datos.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: Lista con todos los pacientes que están almacenados en la base de datos:

```
[{DNI, CIPA, full_name, birthday, phone_number}]
```

GET /patients/{dni}

Descripción Recupera el paciente identificado por su DNI.

Parámetros de consulta: No.

Parámetros de ruta: *dni* (DNI del paciente).

Respuesta: El paciente identificado por su DNI:

```
{
  DNI, CIPA, birthday, phone_number,        // Mismos campos que en la base de datos
  full_name,   // Concatenación de name y surname
  age,         // Edad calculada a partir de la fecha de nacimiento
  connected,   // Booleano que indica si existe una conexión con el paciente o no
}
```

XIV.3.3 Vacunas

GET /patients/{dni}/vaccines

Descripción: Recupera las vacunas administradas al paciente identificado por su DNI, junto con los datos de los centros y los productos de vacunas asociados.

Parámetros de consulta: No.

Parámetros de ruta: *dni* (DNI del paciente).

Respuesta: Lista con las vacunas del paciente:

```
[{
 vaccine_id, dose_number, batch, date, credential_state // Mismos campos que en la base de datos
 centre_name,          // Nombre del centro donde se administró la vacuna
 centre_country,       // País donde está localizado el centro
 product,              // Nombre del producto de vacuna administrada
 manufacturer,         // Fabricante del producto de vacuna
 product_type,         // Tipo de vacuna
}]
```

POST /patients/{dni}/vaccines

Descripción: Crea y almacena en la base de datos una nueva vacuna.

Parámetros de consulta: No.

Parámetros de ruta: *dni* (DNI del paciente).

Cuerpo de la petición:

```
{
  product_id,  // Identificador del producto de vacuna empleado
  batch,       // Lote
  date,        // Fecha de administración
}
```

Respuesta: No.

POST /patients/{dni}/vaccines/{vaccine_id}/issue

Descripción: Expide la credencial de vacunación indicada al paciente correspondiente.

Parámetros de consulta: No.

Parámetros de ruta: *dni* (DNI del paciente), *vaccine_id* (identificador de la vacuna).

Cuerpo de la petición: {}

Respuesta: No.

POST /patients/{dni}/vaccines/{vaccine_id}/revoke

Descripción: Revoca la credencial de vacunación indicada al paciente correspondiente.

Parámetros de consulta: No.

Parámetros de ruta: *dni* (DNI del paciente), *vaccine_id* (identificador de la vacuna).

Cuerpo de la petición: {}

Respuesta: No.

XIV.3.4 Pruebas

GET /patients/{dni}/tests

Descripción: Recupera los test realizados al paciente identificado por su DNI, junto con los datos de los centros y los productos de test asociados.

Parámetros de consulta: No.

Parámetros de ruta: *dni* (DNI del paciente).

Respuesta: Lista con los test del paciente. Test:

```
 {
    test_id, result, date, credential_state // Mismos campos que en la base de datos
    centre_name,        // Nombre del centro donde se realizó el test
    centre_country,     // País donde está localizado el centro
    product,            // Nombre del producto de test empleado
    manufacturer,       // Fabricante del producto de test
    product_type,       // Tipo de test
}
```

POST /patients/{dni}/tests

Descripción: Crea y almacena en la base de datos un nuevo test.

Parámetros de consulta: No.

Parámetros de ruta: *dni* (DNI del paciente).

Cuerpo de la petición:

```
{
    product_id,       // Identificador del producto de test empleado
    result,           // Resultado del test
    date,             // Fecha de la prueba
}
```

Respuesta: No.

POST /patients/{dni}/tests/{test_id}/issue

Descripción: Expide la credencial de test indicada al paciente correspondiente.

Parámetros de consulta: No.

Parámetros de ruta: *dni* (DNI del paciente), *test_id* (identificador del test).

Cuerpo de la petición: {}

Respuesta: No.

POST /patients/{dni}/tests/{test_id}/revoke

Descripción: Revoca la credencial de test indicada al paciente correspondiente.

Parámetros de consulta: No.

Parámetros de ruta: *dni* (DNI del paciente), *test_id* (identificador del test).

Cuerpo de la petición: {}

Respuesta: No.

XIV.3.5 Otras

GET /centre

Descripción: Recupera el centro sanitario desde el cual se está accediendo a la aplicación.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: El nombre del centro sanitario.

GET /user

Descripción: Recupera el usuario (personal sanitario o administrativo) que tiene abierta la sesión en la aplicación.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: El nombre del usuario.

GET /vaccines/products

Descripción Recupera todos los productos de vacunas.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: Una lista con los productos: [{product_id, product}]

GET /tests/products

Descripción Recupera todos los productos de test.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta: Una lista con los productos: [{product_id, product}]

POST /connection/invitation

Descripción: Crea una invitación de conexión.

Parámetros de consulta: *alias* (opcional, alias a establecer para la conexión).

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: La URL de la invitación a la conexión creada.

XIV.4 IBERIA

POST /connection/invitation

Descripción: Crea una invitación de conexión.

Parámetros de consulta: *alias* (opcional, alias a establecer para la conexión).

Parámetros de ruta: No.

Cuerpo de la petición: {}

Respuesta: La invitación a la conexión creada.

XIV.5 Cartera digital SaludMadrid

XIV.5.1 Conexiones

GET /connections

Descripción: Recupera las conexiones establecidas por el ciudadano.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta:

```
[{
    connection_id,     // Identificador de la conexión
    name               // Nombre del otro agente
}]
```

POST /connections/accept

Descripción: Acepta una nueva conexión.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición: {invitation_url} // *La URL de la invitación*

Respuesta: {connection_id, name}.

DELETE /connections/{connection_id}

Descripción: Elimina la conexión especificada por el identificador.

Parámetros de consulta: No.

Parámetros de ruta: *connection_id* (El identificador de la conexión).

Respuesta: El identificador de la conexión eliminada.

XIV.5.2 Credenciales

GET /wallet/credentials

Descripción: Recupera las credenciales que tiene el ciudadano en la cartera.

Parámetros de consulta: No.

Parámetros de ruta: No.

Respuesta:

```
[{
    credential_id,     // Identificador de la credencial
    name               // Nombre del credencial
}]
```

GET /wallet/credentials/{credential_id}

Descripción: Recupera la credencial indicada de la cartera.

Parámetros de consulta: No.

Parámetros de ruta: *connection_id* (El identificador de la conexión).

Respuesta: La credencial completa.

DELETE /wallet/credentials/{credential_id}

Descripción: Elimina la credencial especificada por el identificador.

Parámetros de consulta: No.

Parámetros de ruta: *credential_id* (El identificador de la credencial).

Respuesta: El identificador de la credencial eliminada.

XIV.5.3 Presentación

POST /proof/send-presentation

Descripción: Envía la presentación de la credencial solicitada por el verificador.

Parámetros de consulta: No.

Parámetros de ruta: No.

Cuerpo de la petición:

```
{
    pres_ex_id,     // El identificador temporal de verificación de la presentación
    req_attributes: [{0_attr1_uuid}, ..],     // Lista de atributos solicitados
    req_predicates: [0_pred1_predOperation_uuid, …] // Lista de predicados solicitados a probar
}
```

Respuesta: No.

ANEXO XV: CAPTURAS DE LA INTERFAZ DEL AGENTE RUC19

En este anexo se presentan las capturas realizadas a la interfaz web de la implementación final del agente RUC19.

Página "Home".

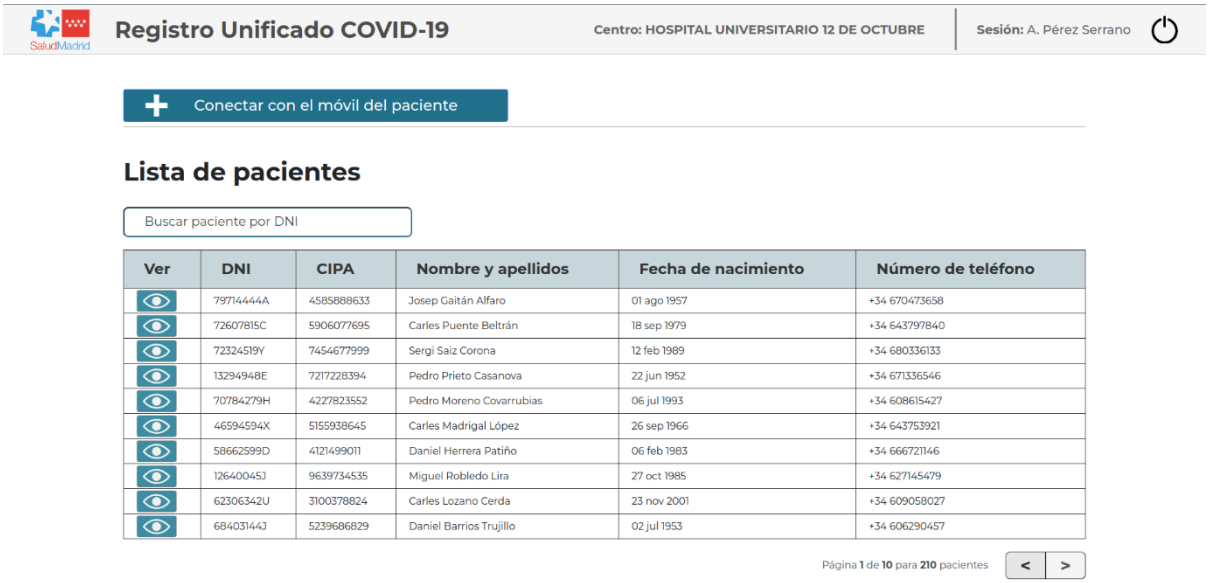

Figura 45: Captura de la interfaz RUC19 implementada - Página Home

La tabla es interactiva, pudiendo realizar búsquedas sobre ella, ordenar las columnas y navegar entre las páginas de la tabla.

Los datos mostrados en esta tabla son los obtenidos de la base de datos implementada en este agente.

Página "Connection".

Figura 46: Captura de la interfaz RUC19 implementada - Página Connection

Página "Patient". Al igual que en las capturas del diseño, en estas capturas aparecerán casos donde el agente tenga establecida una conexión con el ciudadano y casos donde no.

Pestaña: *Datos de vacunación*.

Figura 47: Captura de la interfaz RUC19 implementada - Página Patient – Pestaña Vacunación

En este caso, el paciente no tiene establecida una conexión con el agente, por lo que el enfermero o administrativo no le podrá expedir credenciales. Esto se puede apreciar en el botón de la derecha de la tabla ("Expedir"), el cual está desactivado.

Si se pulsa sobre el botón de la izquierda de la tabla (visualizar), aparecerá el siguiente modal en pantalla:

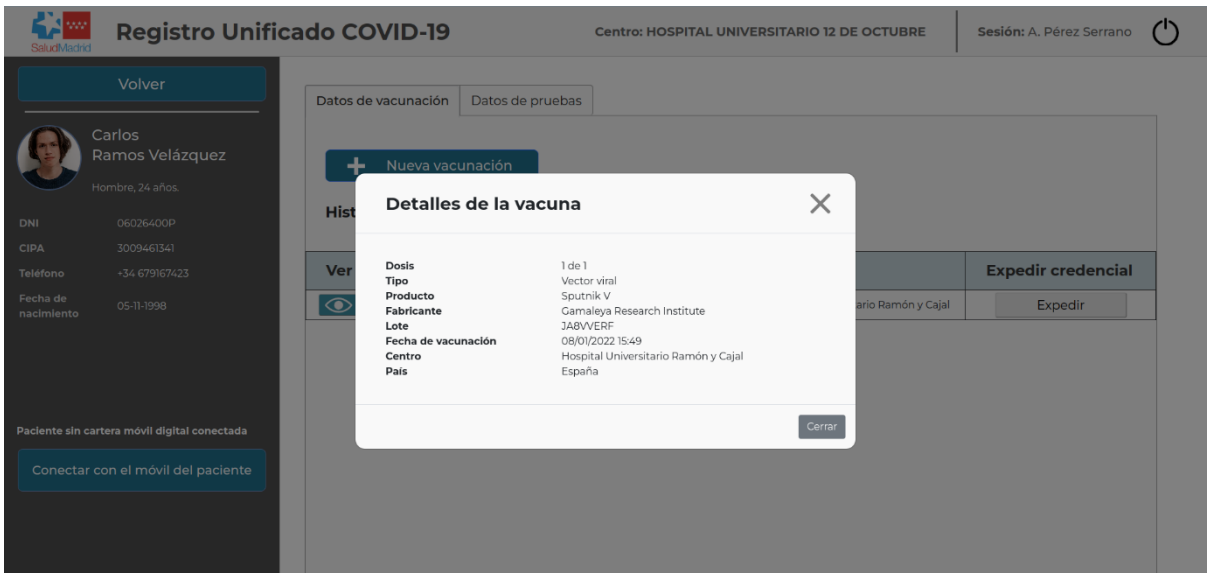

Figura 48: Captura de la interfaz RUC19 implementada - Página Patient – Modal Vacunación

Pestaña: *Datos de pruebas*.

Figura 49: Captura de la interfaz RUC19 implementada - Página Patient – Pestaña Test

En este caso, el paciente sí tiene establecida una conexión con el agente, por lo que el enfermero o administrativo le podrá expedir y revocar credenciales. Esto se puede apreciar en los botones de la derecha de la tabla (“Expedir” y “Revocar”), los cuales se pueden pulsar. También se muestra el caso en el que una credencial está aún en proceso de expedirse, al mostrar el mensaje “En proceso”.

Si se pulsa sobre el botón de la izquierda de la tabla (visualizar), aparecerá el siguiente modal en pantalla:

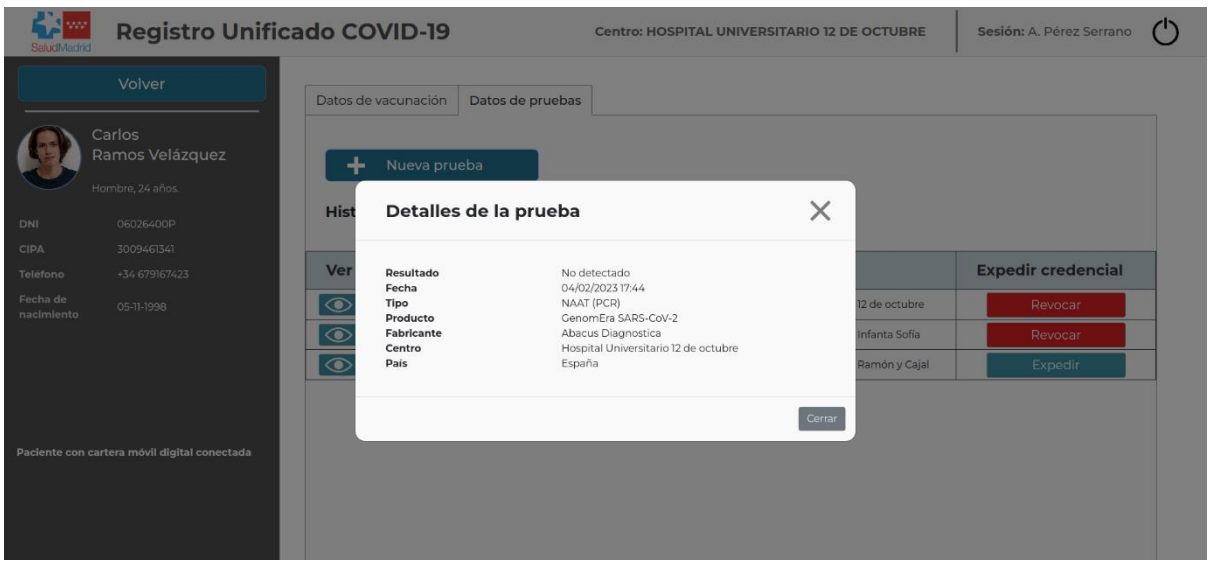

Figura 50: Captura de la interfaz RUC19 implementada - Página Patient – Modal Test

Pestaña: *Nueva Vacuna*.

Figura 51: Captura de la interfaz RUC19 implementada - Página Patient – Pestaña Nuevo Test

El despliegue del campo "Producto" permite una búsqueda y este se ha implementado utilizando la biblioteca *react-select*.

Pestaña: *Nuevo Test*.

Figura 52: Captura de la interfaz RUC19 implementada - Página Patient – Pestaña Nueva Vacuna

El despliegue del campo "Producto" permite una búsqueda y este se ha implementado utilizando la biblioteca *react-select*.

ANEXO XVI: CAPTURAS DE LA INTERFAZ DEL AGENTE IBERIA

En este anexo se presentan las capturas realizadas a la interfaz web de la implementación final del agente Iberia.

Página "QRCode".

Figura 53: Captura de la interfaz Iberia implementada - Página QRCode

Página "Result". Todos los *spinners* que aparecen en las capturas se han implementado con la biblioteca *react-spinners*.

En esta primera captura se muestra el caso en el que el ciudadano aún no ha presentado ninguna credencial.

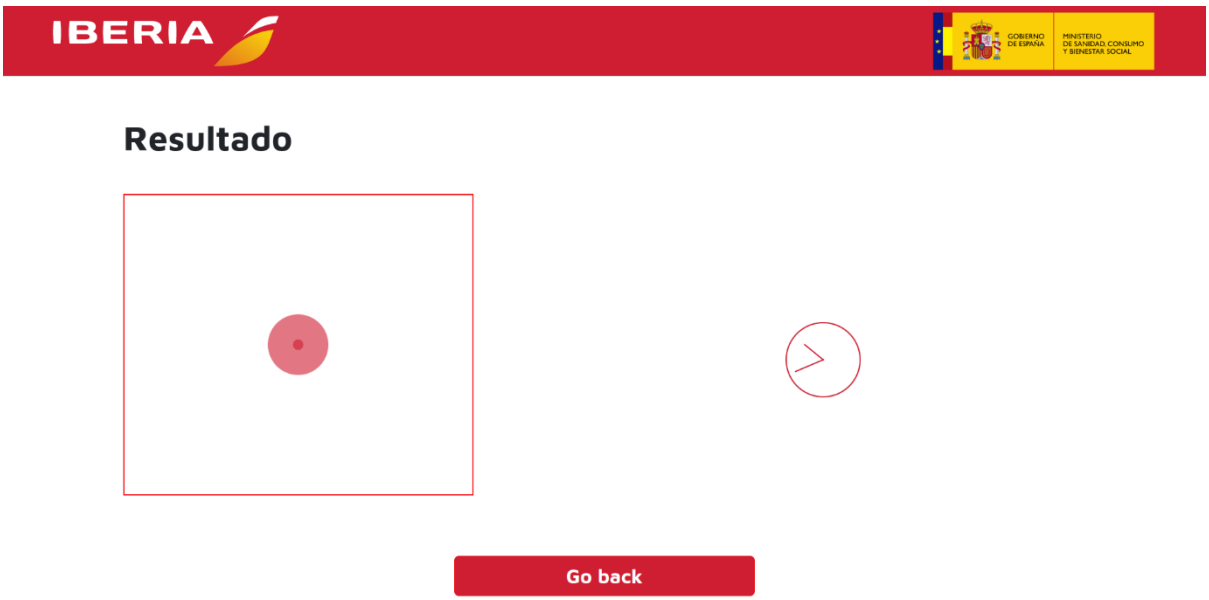

Figura 54: Captura de la interfaz Iberia implementada - Página Result – Vacía

En esta segunda captura se muestra el caso en el que el ciudadano ya ha presentado la credencial de identidad e Iberia está esperando a que presente el resto.

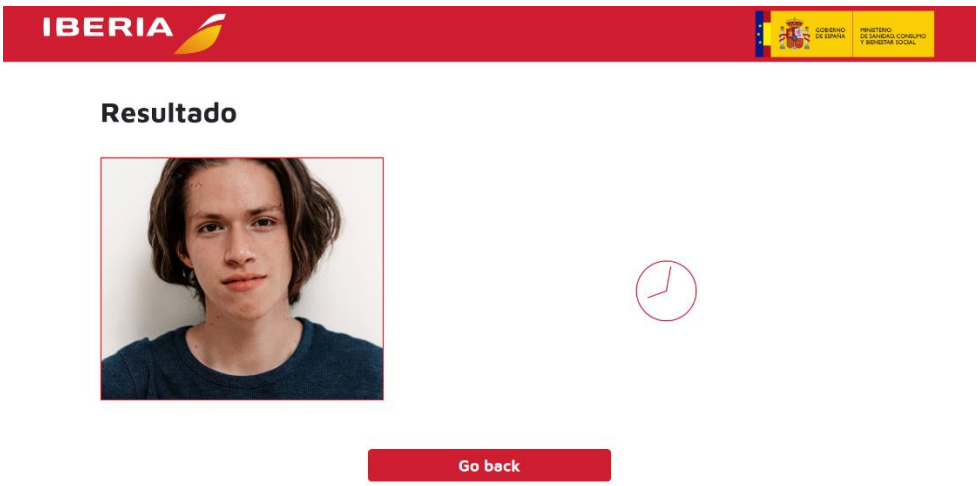

Figura 55: Captura de la interfaz Iberia implementada - Página Result - Foto

En esta tercera captura se muestra el caso en el que el ciudadano ha presentado todas las credenciales y cumple con todos los requisitos sanitarios.

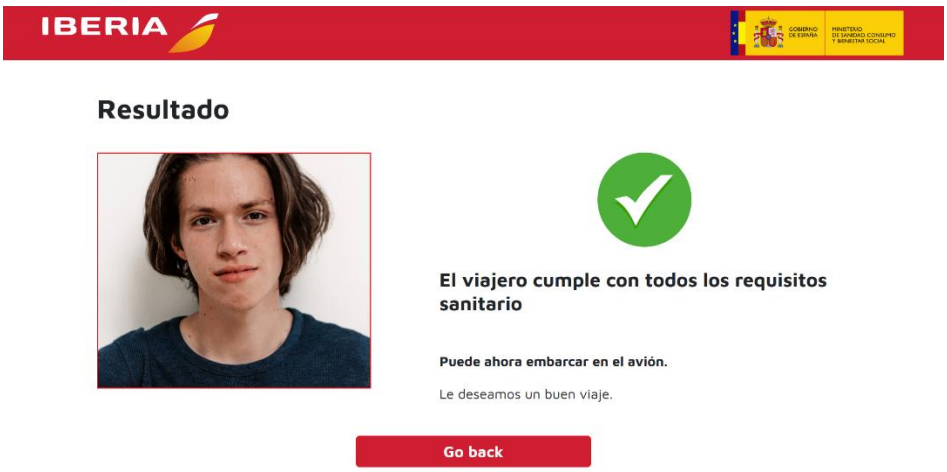

Figura 56: Captura de la interfaz Iberia implementada - Página Result - Paso permitido

Y en esta cuarta captura se muestra el caso en el que no cumple con los requisitos sanitarios.

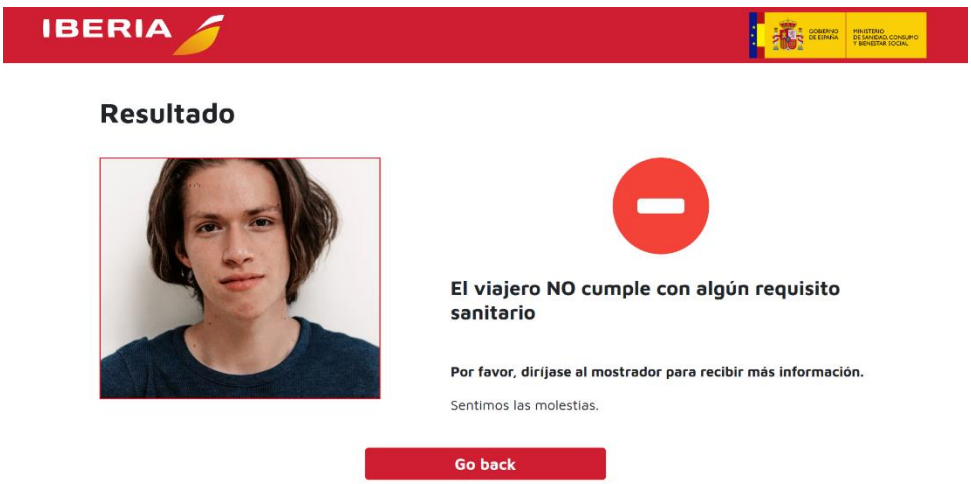

Figura 57: Captura de la interfaz Iberia implementada - Página Result - Paso no permitido

ANEXO XVII: CAPTURAS DE LA INTERFAZ DEL AGENTE CARTERA DIGITAL SALUDMADRID

En este anexo se presentan las capturas realizadas a la interfaz web de la implementación final del agente Cartera Digital SaludMadrid.

En las pantallas principales (*Home*, *Connections* y *Credentials*) se puede apreciar una pequeña navegación de pestañas en la parte inferior de la pantalla, la cual permite navegar entre estas pantallas. Para la navegación con las pantallas secundarias (*Presentation*, *Scan* y *Credential*) aparecerán diferentes botones en la interfaz de las pantallas principales para navegar a estas. En estas pantallas secundarias aparecerá un botón en la parte superior izquierda que permite volver a la pantalla principal desde la cual se ha navegado.

Pantalla "Home". En esta pantalla aparece el nombre del usuario junto con el nombre de la aplicación. Además, aparecen las diferentes notificaciones recibidas del servidor.

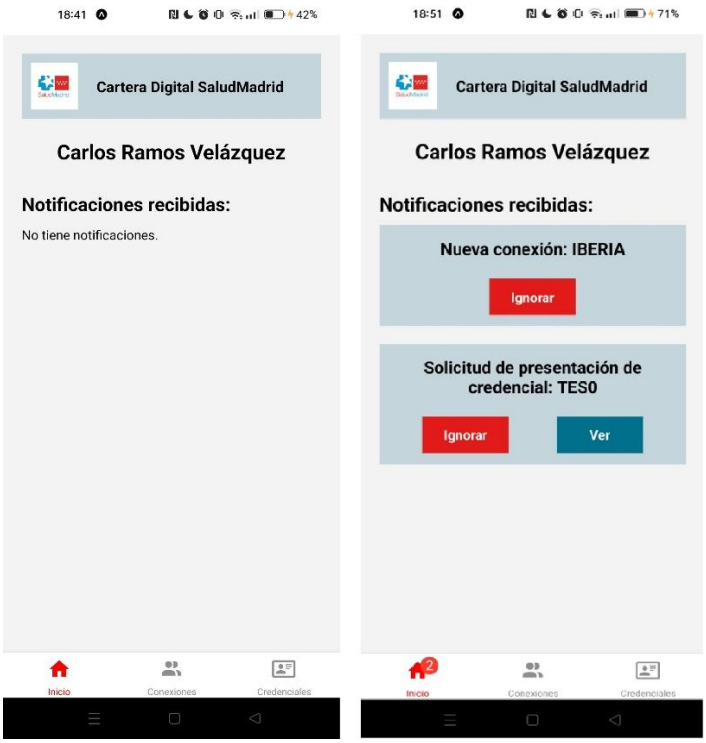

Figura 58: Capturas de la interfaz SaludMadrid implementada - Pantalla Connections

En la captura de la izquierda se puede observar la pantalla *Home* sin notificaciones, mientras que en la pantalla derecha se puede apreciar la misma pantalla cuando existe alguna notificación.

Adicionalmente, se puede ver un pequeño círculo rojo con un número sobre el icono "Inicio" de la navegación inferior en la captura de la derecha, el cual indica que hay varias notificaciones sin revisar.

En las notificaciones de solicitud de presentación de credencial recibidas se puede apreciar como aparece un botón ("Ver"). Este botón lleva a la pantalla *Presentation*, la cual permite gestionar la solicitud recibida.

Pantalla “Presentation”. En esta pantalla puede el usuario gestionar las solicitudes de presentación.

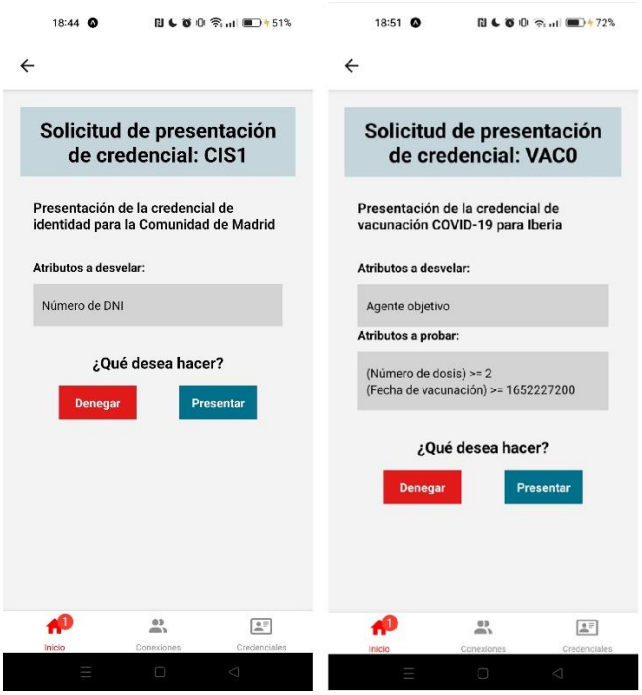

Figura 59: Capturas de la interfaz SaludMadrid implementada - Pantalla Presentation

En la captura de la izquierda se puede apreciar una solicitud de la credencial de identidad por parte del agente RUC19, donde únicamente se pide revelar un atributo; mientras que en la captura de la derecha se aprecia una solicitud de la credencial de vacunación donde se pide revelar un atributo y probar con ZKP otros dos. El botón “Presentar” permite presentar la credencial.

Pantalla “Connections”.

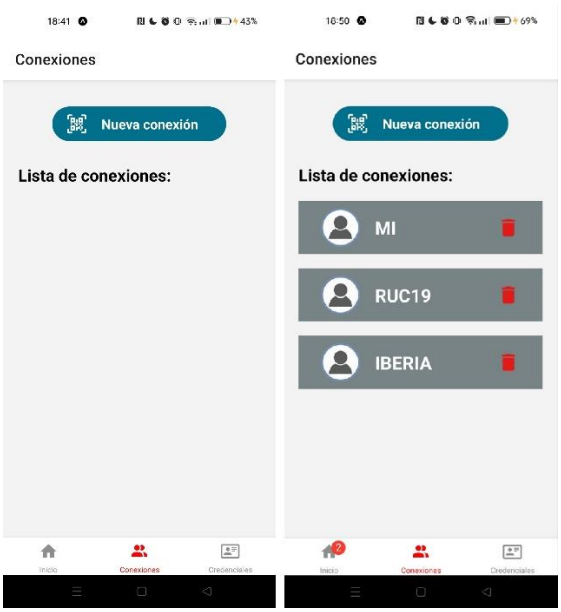

Figura 60: Capturas de la interfaz SaludMadrid implementada - Pantalla Connections

En la captura de la izquierda se muestra la pantalla cuando el agente no tiene establecida aún ninguna conexión, mientras que en la de la derecha se pueden apreciar diferentes conexiones establecidas. El botón “Nueva conexión” lleva al usuario a la pantalla *Scan*.

Pantalla “Scan”. En esta pantalla se puede escanear una invitación de conexión.

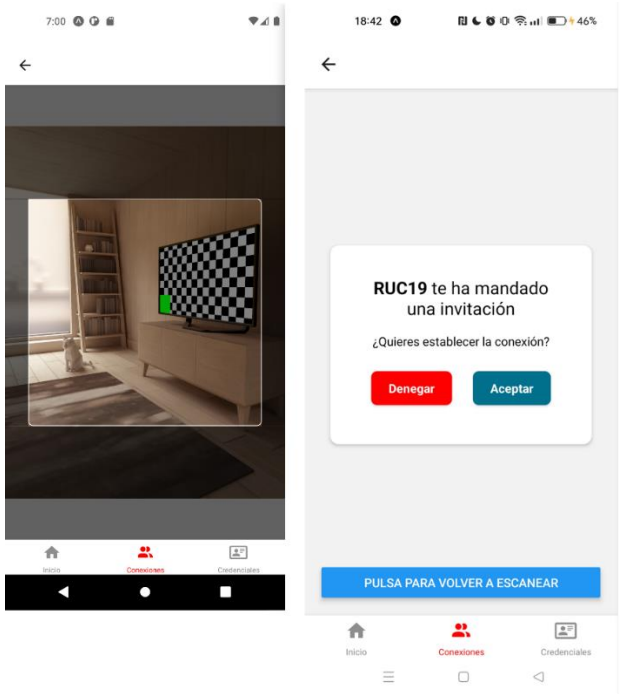

Figura 61: Capturas de la interfaz SaludMadrid implementada – Pantalla Scan

En la captura de la izquierda se aprecia la aplicación haciendo uso de la cámara para escanear un código QR, mientras que en la captura de la derecha se muestra el modal que aparece cuando se consigue escanear un código QR de invitación. El botón “Aceptar” permite aceptar la invitación. Nótese el botón inferior (“Pulsa para volver a escanear”) que permite volver a escanear un código QR.

Pantalla “Credentials”.

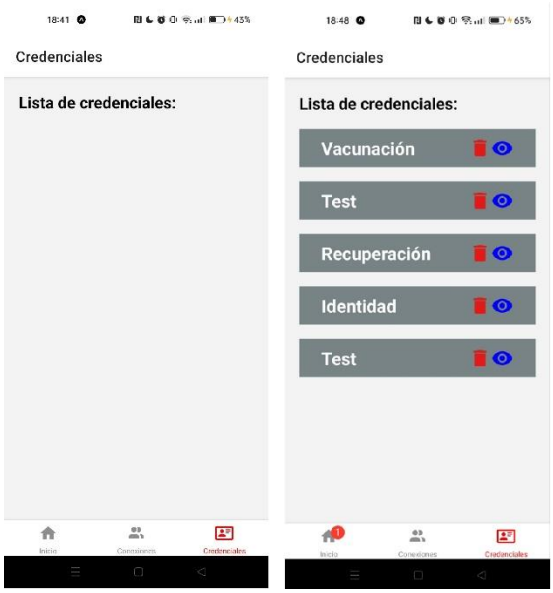

Figura 62: Capturas de la interfaz SaludMadrid implementada - Pantalla Credentials

En la captura de la izquierda se muestra la pantalla cuando el agente no tiene almacenada ninguna credencial, mientras que en la de la derecha se pueden apreciar diferentes credenciales almacenadas. El botón de visualizar que hay en todas las credenciales lleva al usuario a la pantalla *Credential*.

Pantalla "Credential". En esta pantalla se pueden observar las diferentes credenciales almacenadas.

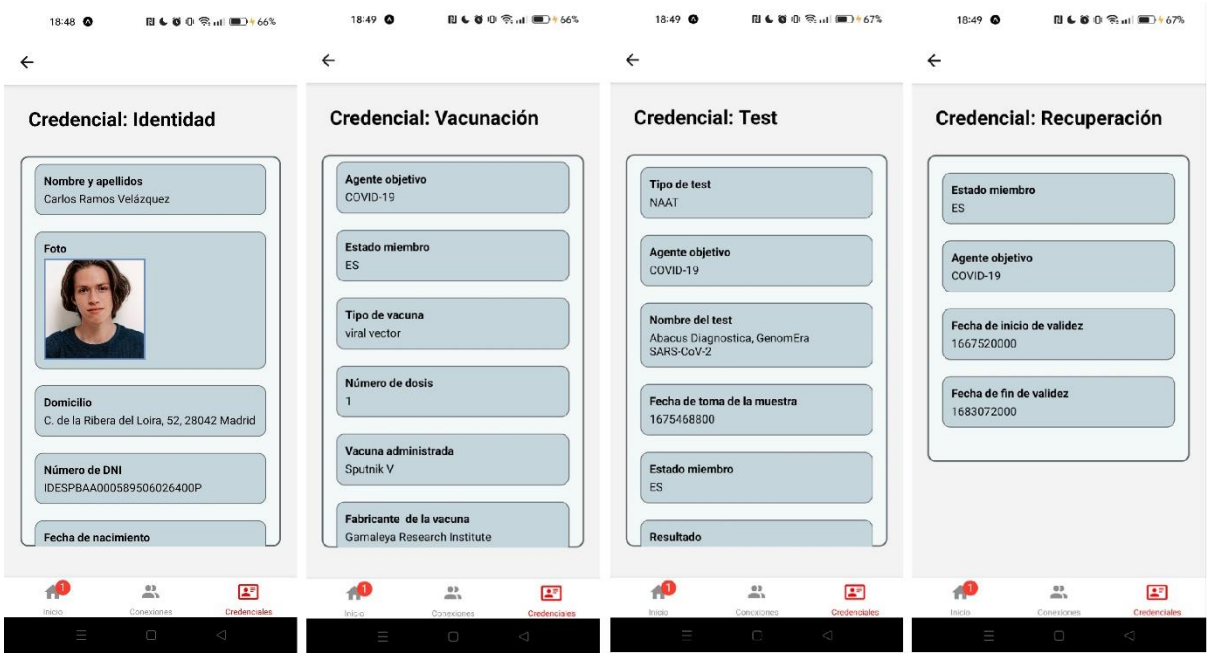

Figura 63: Capturas de la interfaz SaludMadrid implementada - Pantalla Credential

De izquierda a derecha se muestran los casos de las credenciales: identidad, vacunación, test y recuperación. Se puede observar que los nombres de los nombres de los atributos son legibles por un humano y que aparece una foto en lugar de una URL en el caso de la credencial de identidad.