

FEBRERO 2025

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

GRADO EN TECNOLOGÍAS PARA LA SOCIEDAD DE LA INFORMACIÓN

PROYECTO DE FIN DE GRADO



KeyGuardian

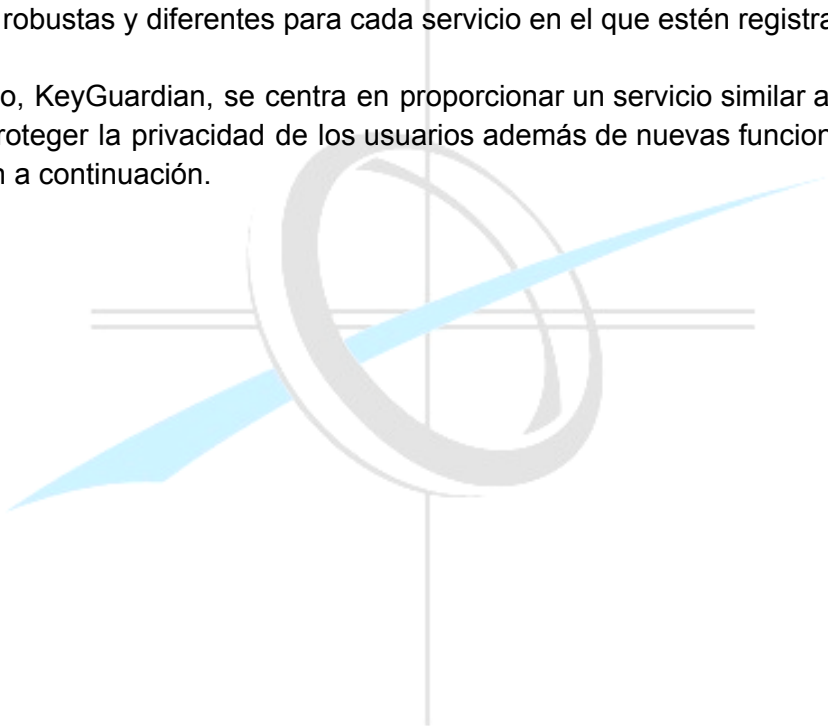
Autor: Álvaro Seoane Alfonso

RESUMEN

La seguridad online no es una materia que se deba tomar a la ligera. Hoy en día todo el mundo sabe que es una cuestión seria y muchos toman precauciones. Sin embargo, muchos usuarios siguen utilizando la misma contraseña en todas sus cuentas, lo cual no es deseable puesto que si una de tus cuentas se viera comprometida en algún tipo de ataque, de repente toda tu actividad online estaría comprometida.

Para solventar este problema surgen los servicios que ayudan a los usuarios a mantener su presencia en la red segura, gestionando sus contraseñas. Habilitando a los usuarios a crear contraseñas robustas y diferentes para cada servicio en el que estén registrados.

Este proyecto, KeyGuardian, se centra en proporcionar un servicio similar añadiendo capas extra para proteger la privacidad de los usuarios además de nuevas funcionalidades que se desarrollarán a continuación.



ABSTRACT

Online security is not something anyone should take lightly. Nowadays everyone knows it is, in fact, a very serious matter and many take action. Even though many people are taking precautions, most of them are still using the same password for every online service they use. Such a thing is a very dangerous practice given that if any of those services gets involved in any sort of data breach, all of your online activity is suddenly compromised.

To solve this problem services known as password managers arise. These services help users keep their online presence more secure by enabling them to manage multiple different strong passwords efficiently and correctly.

This project, KeyGuardian, is centered around providing a similar service with added extra layers to protect users privacy on top of new functionalities that will be developed later on.



ÍNDICE

RESUMEN.....	2
ABSTRACT.....	3
ÍNDICE.....	4
ÍNDICE DE ILUSTRACIONES.....	6
INTRODUCCIÓN.....	8
ALCANCE.....	9
METODOLOGÍA DE TRABAJO.....	10
GIT-FLOW.....	11
SCRUMBAN.....	16
SCRUM.....	17
KANBAN.....	19
SCRUMBAN.....	21
PROYECTO.....	22
TABLERO.....	23
SPRINTS.....	23
BACKLOG.....	24
CRONOGRAMA.....	25
TECNOLOGÍAS UTILIZADAS.....	27
BACKEND.....	28
PYTHON.....	28
FIREBASE AUTHENTICATION.....	30
FIRESTORE DATABASE.....	32
FLASK.....	34
GUNICORN.....	35
GOOGLE CLOUD.....	36
DOCKER.....	37
TAILSCALE.....	38
FRONTEND.....	39
ANDROID STUDIO.....	39
KOTLIN.....	41
REQUISITOS Y PLANIFICACIÓN.....	43
REQUISITOS.....	43
REQUISITOS FUNCIONALES.....	43
REQUISITOS NO FUNCIONALES.....	44
RELACIÓN ENTRE REQUISITOS Y TAREAS.....	45
DIAGRAMAS.....	47
CASOS DE USO.....	47
FLUJO.....	48
SECUENCIA.....	50

RESUMEN DE SPRINTS.....	51
SPRINT 1.....	51
SPRINT 2.....	52
SPRINT 3.....	54
SPRINT 4.....	55
SPRINT 5.....	56
PROYECTO.....	58
ARQUITECTURA DEL PROYECTO.....	58
BACKEND.....	59
ARQUITECTURA BACKEND.....	59
FUNCIONALIDAD.....	61
TRATAMIENTO Y PROCESAMIENTO DE DATOS.....	62
RESUMEN DE ENDPOINTS.....	66
FRONTEND.....	70
ARQUITECTURA FRONTEND.....	70
FUNCIONALIDAD.....	71
TRATAMIENTO DE DATOS.....	78
SEGURIDAD Y PRIVACIDAD EN LA APLICACIÓN.....	80
IMPACTO.....	81
SEGURIDAD.....	81
PRIVACIDAD.....	82
OTRAS APLICACIONES.....	83
CONCLUSIÓN.....	84
BIBLIOGRAFÍA.....	85

ÍNDICE DE ILUSTRACIONES

Ilustración 1	Esquema del uso de git-flow y de la interacción entre ramas	12
Ilustración 2	Git graph del repositorio del backend	14
Ilustración 3	Git graph del repositorio del frontend	15
Ilustración 4	Tablero utilizado en Jira	23
Ilustración 5	Backlog de Jira	24
Ilustración 6	Diagrama de Gantt del proyecto	25
Ilustración 7	Burndown chart de story points del proyecto	26
Ilustración 8	Diagrama de casos de uso	47
Ilustración 9	Diagrama de flujo en la creación de un documento nuevo	48
Ilustración 10	Diagrama de flujo en la edición de un documento	49
Ilustración 11	Diagrama de secuencia	50
Ilustración 12	Burndown chart del Sprint 1	52
Ilustración 13	Burndown chart del Sprint 2	53
Ilustración 14	Burndown chart del Sprint 3	55
Ilustración 15	Burndown chart del Sprint 4	56
Ilustración 16	Burndown chart del Sprint 5	57
Ilustración 17	Arquitectura del proyecto	58
Ilustración 18	Diagrama de arquitectura backend	60
Ilustración 19	Muestra de registro de usuarios en Firebase Authentication	62
Ilustración 20	Muestra de datos almacenados en Firestore	63
Ilustración 21	Pantalla de registro	72
Ilustración 22	Pantalla de inicio de sesión	72
Ilustración 23	Menú principal	73

Ilustración 24	Pantalla de logout	73
Ilustración 25	Pantalla principal	75
Ilustración 26	Pantalla de edición	77



INTRODUCCIÓN

La idea para este proyecto surge a raíz de una genuina preocupación por la seguridad de los usuarios que utilizan diferentes servicios online puesto que el número de persona afectadas en eventos de violaciones de datos en servicios web ha seguido una tendencia incremental en los últimos años, con 2023 a la cabeza con un número de víctimas que supera los 350 millones (1). Muchas de estas personas han sufrido las consecuencias y los efectos secundarios que esto conlleva.

Por tanto KeyGuardian propone una solución para mitigar el impacto que un evento de este tipo podría tener. Utilizando un gestor de contraseñas, un usuario afectado notaría impacto en el servicio comprometido pero no más. En cambio, si otro usuario utiliza las mismas credenciales para toda su actividad online podría tener consecuencias nefastas desde filtración de datos sensibles hasta suplantación de la identidad.

El proyecto en sí ha supuesto un gran reto para mí, ya que ha sido la primera vez que he desarrollado un proyecto de esta magnitud en solitario, desde su concepción hasta su terminación y utilizando tecnologías desconocidas para mí, por ejemplo, el lenguaje Kotlin.

En esta memoria se explicarán todas las características de la aplicación y todo el proceso de desarrollo del proyecto, desde su planificación hasta su despliegue.

ALCANCE

El objetivo principal del proyecto es crear una aplicación intuitiva, eficaz y segura mediante la cual los usuarios puedan gestionar todas sus credenciales de forma sencilla y privada. Todo esto manteniendo una arquitectura de servicio escalable, mantenible y con posibilidad de llegar a ser multiplataforma.

KeyGuardian consigue cubrir todos estos puntos añadiendo una capa extra de seguridad, la habilidad de crear y cifrar diferentes documentos de credenciales que se generan en la aplicación. Estos documentos posteriormente se cifran mediante una segunda contraseña maestra, de forma local.

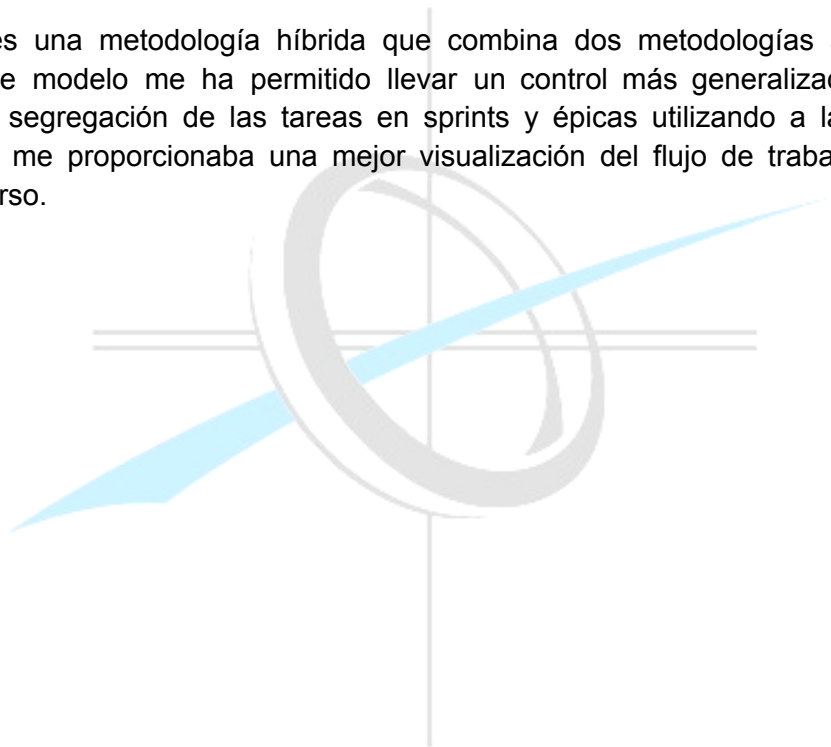
Este cifrado de documentos que reside en una capa más baja que el inicio de sesión, se realiza de forma local en el dispositivo del usuario para evitar para evitar un tratamiento de datos por parte del servicio que pueda llegar a ser un posible punto de fuga. Además de permitir a los usuarios hacer uso de redes públicas, usando la aplicación, de forma segura puesto que toda comunicación entre la aplicación y los servidores tiene esta capa extra de seguridad además de las medidas de seguridad estándar.

METODOLOGÍA DE TRABAJO

A continuación se detallan las dos metodologías de trabajo principales que he utilizado durante el desarrollo del proyecto: git-flow y scrumban.

Git-flow es un modelo de gestión de ramas en Github que facilita el control de versiones durante el desarrollo. Esta forma de trabajar me ha permitido gestionar y estructurar mi código de una manera ordenada, mejorando la trazabilidad del código de cada requisito y me permite probar cada nueva mejora de forma independiente al resto del código ya verificado.

Scrumban es una metodología híbrida que combina dos metodologías ágiles, scrum y kanban. Este modelo me ha permitido llevar un control más generalizado del proyecto mediante la segregación de las tareas en sprints y épicas utilizando a la vez el tablero kanban que me proporcionaba una mejor visualización del flujo de trabajo limitando las tareas en curso.



GIT-FLOW

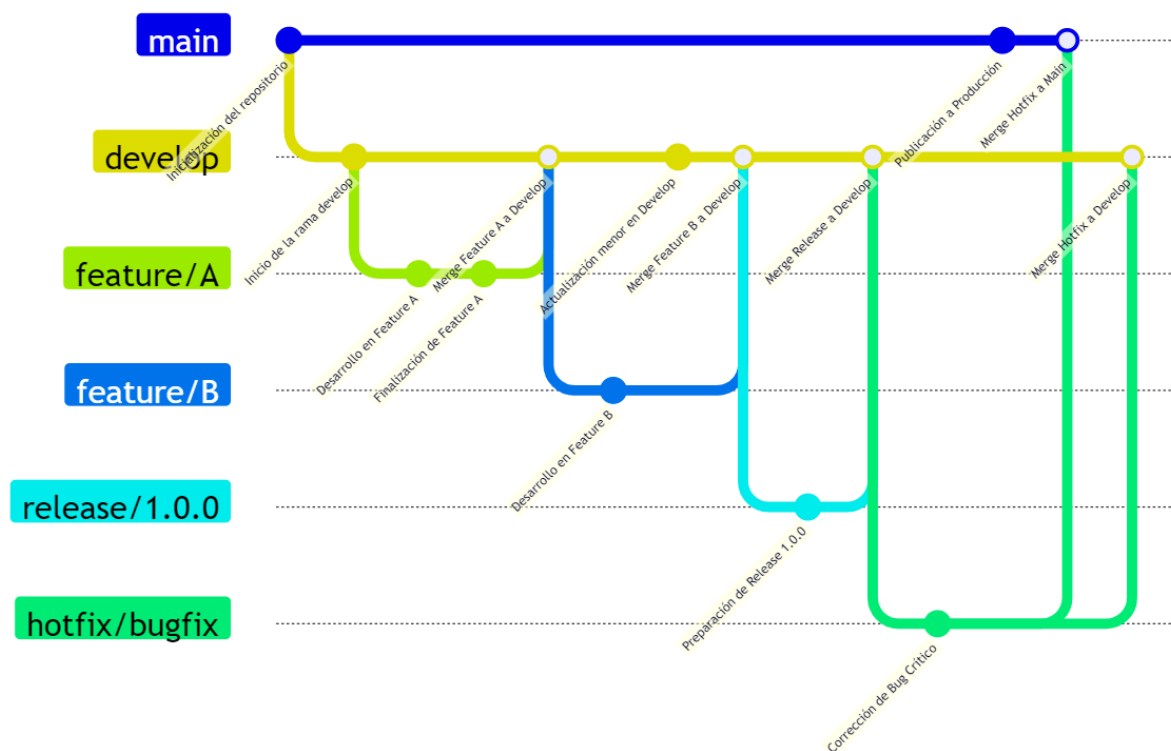
Git-flow es una metodología creada por Vincent Driessen en 2010 cuyo objetivo principal es la gestión de repositorios Git en proyectos con un ciclo de desarrollo bien definido, principalmente en proyectos con una planificación de entregas iterativas.

Con esta metodología se gestionan los repositorios mediante la creación de varios tipos de ramas predefinidas en el repositorio. Cada una de estas ramas tiene unas características que la definen y que le aportan una funcionalidad específica. Por ello, cuando se utiliza git-flow es importante saber con qué tipo de rama hay que trabajar en función de la tarea que se pretenda llevar a cabo.

Las siguientes son las diferentes ramas que hay que utilizar cuando se trabaja con git-flow:

- *main* | *master*: Es la rama principal, de la que parten en la creación del proyecto todas las ramas y que contiene la última versión del código que se considera que está preparado para producción.
- *develop*: Es la rama con más importancia para el desarrollador en la que se encuentran los últimos cambios finalizados en el software pero que aún no son productivos.
- *release*: Las ramas *release/** actúan como puntos de control y versionado de momentos específicos en el ciclo de vida del software.
- *hotfix*: Son ramas de desarrollo orientadas a la solución de un fallo o bug específico identificado en el código productivo.
- *feature*: Son ramas de desarrollo de funcionalidades específicas que ayudan a compartimentalizar y trazar nuevos desarrollos.

(Ilustración 1) Esquema del uso de git-flow y de la interacción entre ramas



El uso de git-flow implica ciertas restricciones y buenas prácticas diseñadas para mantener el repositorio ordenado y estructurado:

- La rama **main** es un histórico de todas las versiones productivas que ha habido del software y su última instancia es una réplica del código que se encuentra desplegado en ese instante.
- De la rama **main** parte la rama **develop** y cualquier cambio que se realice sobre la rama principal se tiene también que replicar en **develop**.
- Las ramas **feature/*** salen de **develop** y se integran en la misma una vez finalizado el desarrollo.
- Las ramas **release/*** parten de **develop** y se integran en la rama **main** en caso de que lleguen a producción.
- Las ramas **hotfix/*** parten de **main** y se integran de nuevo en **main** y en **develop** una vez solucionado el problema con el código.

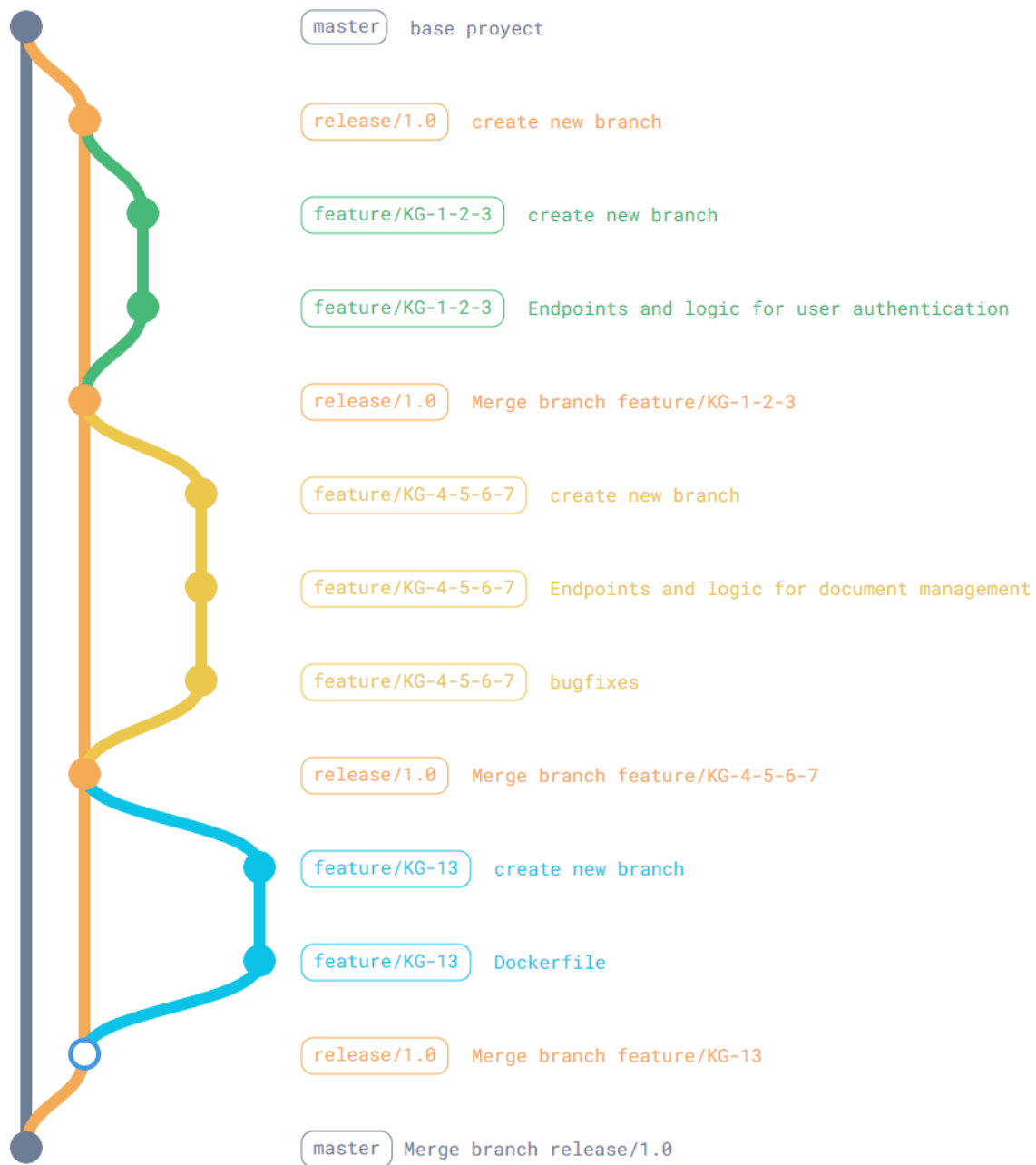
El uso de esta metodología de gestión de repositorios en mi proyecto dará lugar a las siguientes ramas feature:

- feature/KG-1
- feature/KG-2
- feature/KG-3
- feature/KG-4
- feature/KG-5
- feature/KG-6
- feature/KG-7
- feature/KG-8
- feature/KG-9
- feature/KG-10
- feature/KG-11
- feature/KG-12
- feature/KG-13
- feature/KG-14

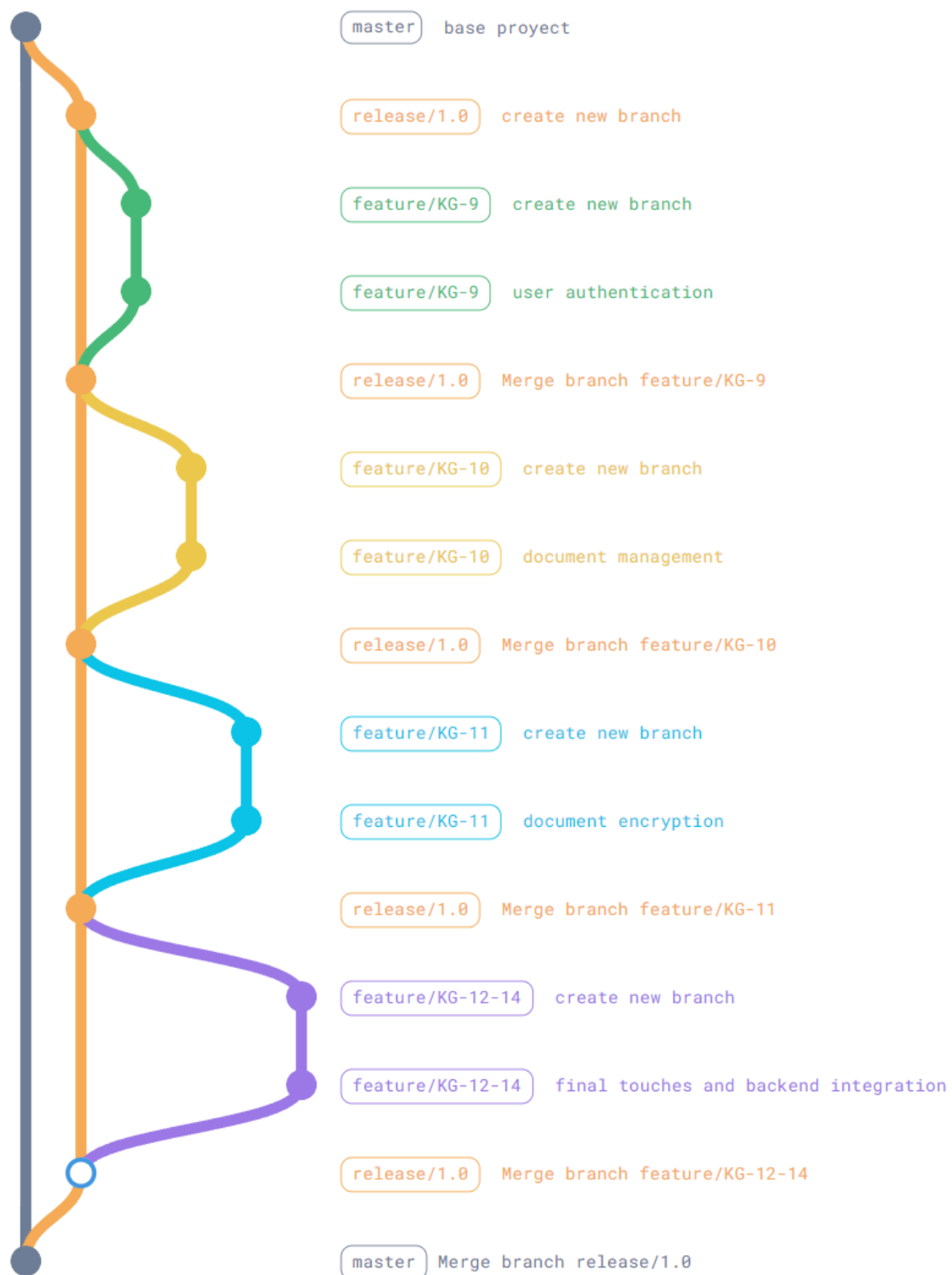
Una por cada tarea de desarrollo necesaria para completar el proyecto, nombradas a partir de las tareas equivalentes generadas en Jira, que es el software empleado para la gestión de tareas en el proyecto, para una mejor trazabilidad.

En mi caso, puesto que el proyecto requiere el desarrollo de dos piezas de software separadas utilizaré dos repositorios, uno para el frontend y otro para el backend por lo que las ramas feature listadas anteriormente estarán divididas entre los dos repositorios.

(Ilustración 2) Git graph del repositorio del backend:



(Ilustración 3) Git graph del repositorio del frontend:



SCRUMBAN

Scrumban es la metodología de desarrollo y planificación escogida para este proyecto. Se trata de una metodología híbrida que combina scrum y kanban, que permite gestionar el desarrollo del proyecto desde el enfoque de scrum dividiendo el trabajo en sprint y épicas incorporando elementos de kanban que proporcionan una buena base de gestión de tareas mediante un tablero.

Scrumban forma parte de la familia de gestión de proyectos conocida como metodologías ágiles que son un conjunto de enfoques para la gestión y desarrollo de proyectos que se caracterizan por su flexibilidad, enfoque en la colaboración, y capacidad para adaptarse rápidamente a los cambios. Estas metodologías fueron diseñadas como una alternativa a los métodos tradicionales de desarrollo, los cuales tienden a ser más rígidos y lineales. Las metodologías ágiles buscan entregar valor al cliente a través de un desarrollo incremental y continuo, manteniendo al equipo y al cliente en constante comunicación para asegurar que el producto final cumpla con las expectativas y se ajuste a posibles cambios en los requisitos.

Las metodologías ágiles fomentan la entrega frecuente de pequeñas porciones de funcionalidad, lo que permite que el equipo realice ajustes según las necesidades del proyecto y los comentarios de los usuarios. Estas metodologías también priorizan el trabajo en equipo, la autogestión y la comunicación constante, lo que contribuye a un ambiente de trabajo colaborativo y orientado a resultados.

Principales ventajas de las metodologías ágiles:

- Mayor flexibilidad y adaptabilidad: Permiten ajustar el desarrollo según los cambios en los requisitos o prioridades sin afectar significativamente el proyecto.
- Enfoque en el cliente: Las revisiones frecuentes y la comunicación constante con el cliente aseguran que el producto final se ajuste a sus expectativas y necesidades reales.
- Entregas rápidas y continuas: A través de iteraciones o sprints, el equipo entrega incrementos del producto de forma continua, lo que permite ver avances y realizar pruebas tempranas.
- Mejora de la calidad del producto: Los ciclos de retroalimentación continua permiten identificar y corregir errores de manera temprana, mejorando la calidad del desarrollo.
- Colaboración y comunicación efectiva: Las metodologías ágiles promueven un ambiente colaborativo entre los miembros del equipo, aumentando la cohesión y el compromiso.

- Reducción de riesgos: La entrega de incrementos funcionales reduce el riesgo de que el producto final no cumpla con los requisitos o sea entregado tarde.
- Mayor satisfacción del equipo: El enfoque en la autogestión y en un ambiente de trabajo adaptable contribuye a la motivación y satisfacción de los miembros del equipo.

SCRUM

Scrum es una de las metodologías ágiles más populares actualmente, diseñada para gestionar proyectos de desarrollo de software, aunque también se utiliza en otros tipos de proyectos. Scrum se basa en la organización del trabajo en ciclos cortos y repetitivos llamados sprints, que suelen durar entre una y cuatro semanas. Cada sprint es un período cerrado en el que el equipo trabaja en un conjunto específico de tareas con el objetivo de entregar un incremento funcional del producto.

Scrum se enfoca en la colaboración, la transparencia y la entrega incremental, permitiendo que el equipo ajuste su trabajo en función del progreso y el feedback. Para garantizar la efectividad de Scrum, se establecen ciertos roles, eventos y artefactos.

Roles en Scrum:

- Product Owner (Propietario del Producto): Es el responsable de definir la visión del producto y de priorizar el Product Backlog (lista de tareas o funcionalidades). Representa los intereses del cliente o usuario.
- Scrum Master: Actúa como facilitador del equipo, asegurándose de que se sigan los principios de Scrum y de que el equipo no enfrente obstáculos que limiten su trabajo.
- Equipo de Desarrollo: Es el grupo de personas encargado de completar las tareas del sprint. Este equipo suele ser multifuncional y autoorganizado.

Eventos en Scrum:

- Sprint: Es el período de trabajo donde se desarrolla un conjunto de tareas. Cada sprint tiene una meta específica y termina con un incremento funcional del producto.
- Sprint Planning: Reunión al inicio de cada sprint donde el equipo selecciona las tareas del Product Backlog que se completarán durante ese sprint.

- Daily Scrum (reunión diaria): Una breve reunión diaria para revisar el progreso y ajustar el plan de trabajo, con un enfoque en lo que cada miembro hizo, hará y los posibles obstáculos.
- Sprint Review: Revisión al final del sprint donde el equipo presenta el trabajo realizado a los interesados y recibe retroalimentación.
- Sprint Retrospective: Reunión de retrospectiva donde el equipo analiza cómo fue el proceso de trabajo y discute posibles mejoras para futuros sprints.

Artefactos en Scrum:

- Product Backlog: Lista priorizada de todas las tareas, funcionalidades y mejoras que deben hacerse en el producto.
- Sprint Backlog: Lista de tareas específicas que el equipo se compromete a completar durante el sprint.
- Incremento: Resultado tangible y funcional que el equipo produce al final de cada sprint. Debe cumplir con los criterios de calidad establecidos para considerarse completo.

KANBAN

Kanban es una metodología ágil visual y flexible para la gestión de proyectos, diseñada para optimizar el flujo de trabajo y mejorar la eficiencia de los equipos. Originada en la industria manufacturera y adoptada ampliamente en desarrollo de software y otros campos, Kanban permite gestionar tareas y proyectos a medida que avanzan, enfocándose en una entrega continua y en la minimización de tiempos de espera.

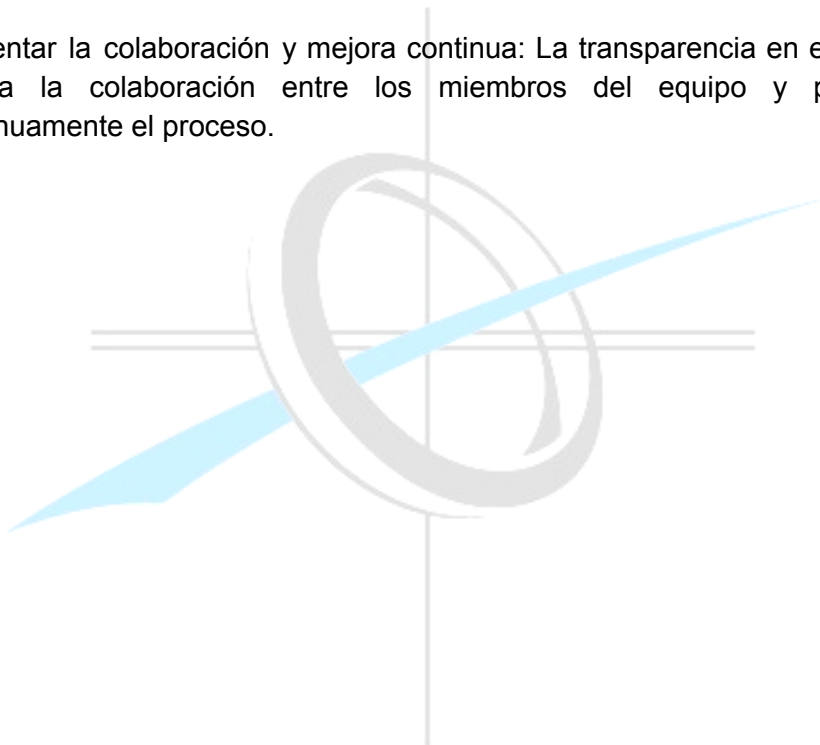
La esencia de Kanban radica en visualizar el flujo de trabajo y limitar el trabajo en proceso (WIP, por sus siglas en inglés) para mejorar la productividad y reducir los cuellos de botella. Para implementarlo, se utiliza un tablero Kanban, que puede ser físico o digital, y está dividido en columnas que representan el estado de cada tarea.

Elementos clave de Kanban:

- **Tablero Kanban:** Es la herramienta principal de Kanban. Las tareas se representan como tarjetas en el tablero, y las columnas indican el estado de estas, como "Por hacer", "En progreso" y "Hecho". Este tablero visualiza todo el flujo de trabajo, permitiendo al equipo ver el progreso y detectar cuellos de botella fácilmente.
- **Tarjetas:** Cada tarea o trabajo pendiente se representa como una tarjeta en el tablero. Las tarjetas pueden contener detalles sobre la tarea, como la descripción, responsable, fecha de entrega, entre otros.
- **Límites de WIP (Work in Progress):** Kanban establece límites en la cantidad de trabajo que puede estar "en progreso" al mismo tiempo. Esto ayuda a evitar la sobrecarga del equipo y garantiza que las tareas se completen de manera eficiente antes de comenzar nuevas.
- **Ciclos de mejora continua:** Kanban fomenta la optimización constante del flujo de trabajo. El equipo revisa regularmente el proceso y los tiempos de entrega para identificar áreas de mejora y ajustar el flujo de trabajo en función de las necesidades.

Principios de Kanban:

- Comenzar con lo que se está haciendo ahora: Kanban se basa en mejorar el proceso de trabajo existente sin introducir grandes cambios de inmediato, lo que facilita su adopción en equipos.
- Aceptar cambios incrementales: Promueve ajustes pequeños y continuos en lugar de cambios radicales, permitiendo que el equipo mejore de manera sostenida.
- Respetar los roles y responsabilidades actuales: Kanban no obliga a cambiar la estructura de los equipos ni los roles, sino que se adapta a la forma de trabajo existente.
- Fomentar la colaboración y mejora continua: La transparencia en el flujo de trabajo facilita la colaboración entre los miembros del equipo y permite mejorar continuamente el proceso.



SCRUMBAN

Scrumban es una metodología híbrida que combina elementos de Scrum y Kanban para crear un flujo de trabajo ágil, flexible y adaptable. Este enfoque es ideal para equipos que desean la estructura y organización de Scrum junto con la fluidez y adaptabilidad de Kanban. Scrumban es especialmente útil en proyectos donde las prioridades pueden cambiar con frecuencia y no es posible planificar sprints rígidos de antemano.

Características principales de Scrumban:

- Estructura de Scrum: Como en Scrum, Scrumban mantiene la reunión diaria (Daily Stand-up) y la retrospectiva para mejorar el proceso. Sin embargo, en lugar de sprints estrictos, permite que el equipo trabaje de forma continua en función de la demanda.
- Tablero Kanban: Scrumban utiliza un tablero Kanban para visualizar el flujo de trabajo. Las tareas se mueven a través de columnas que representan los diferentes estados (por ejemplo, "Por hacer", "En progreso", "Hecho"), lo que permite un control visual del progreso y ayuda a identificar cuellos de botella.
- Límites de trabajo en proceso (WIP): Al igual que en Kanban, Scrumban establece límites de WIP para evitar la sobrecarga del equipo y garantizar que se mantenga el enfoque en las tareas activas antes de comenzar nuevas.
- Reabastecimiento en lugar de sprints: En lugar de realizar una planificación de sprint al inicio de cada ciclo, Scrumban permite un enfoque de "reabastecimiento". Esto significa que cuando un miembro del equipo completa una tarea, selecciona otra del backlog, permitiendo una entrada continua de trabajo.

Ventajas de Scrumban:

- Mayor adaptabilidad: Al no estar limitado por sprints rígidos, Scrumban permite adaptarse rápidamente a cambios en las prioridades.
- Eficiencia en la gestión de tareas: Con el tablero Kanban y los límites de WIP, el equipo puede optimizar el flujo de trabajo y mantener el enfoque.
- Control visual del progreso: El tablero permite una visualización clara del estado de las tareas y facilita la identificación de problemas.
- Mejora continua: La retrospectiva permite que el equipo evalúe y mejore sus procesos regularmente, como en Scrum.

PROYECTO

Aprovechando que scrumban es una metodología híbrida abierta a cambios, en el proyecto decidí utilizar una versión de scrumban adaptada a las necesidades de desarrollo que difiere un poco de la versión detallada anteriormente que es más estándar.

En este caso se mantienen los sprints fijos de scrum como una manera de definir objetivos temporales y segregar el alcance del proyecto en el tiempo. Utilizando también el tablero de kanban y el límite de tareas WIP (Work In Progress).

TABLERO

El tablero kanban utilizado en el proyecto añade una columna más, “Testing”, que sirve como punto de referencia para las tareas cuya fase de desarrollo ha terminado pero que necesitan ser validadas antes de ser integradas en la rama develop. Sirviendo de esta forma como punto de conexión entre scrumban y git-flow, ayudando de esta forma a mantener las ramas limpias.

(Ilustración 4) Tablero utilizado en Jira



SPRINTS

El proyecto está dividido en cinco sprints de una semana cada uno, se ha buscado dividir las tareas en los sprints en base a la estimación de sp (story points) de cada tarea.

- Sprint 1: Proyecto base de backend y gestión de usuarios:
 - KG-1: Creación del proyecto backend base. (3 sp)
 - KG-2: Implementación de la gestión de usuarios mediante Firestore. (4 sp)
 - KG-3: Implementación de la gestión de usuarios mediante token JWT. (4 sp)
- Sprint 2: Integración con Firestore Database y gestión de ficheros
 - KG-4: Integración con Firestore Database. (2 sp)
 - KG-5: Implementación de la lógica de gestión de ficheros. (4 sp)
 - KG-6: Implementación del guardado de ficheros. (3 sp)
 - KG-7: Implementación de la recuperación de ficheros. (3 sp)
- Sprint 3: Proyecto base de frontend y gestión de usuarios:
 - KG-8: Creación del proyecto base del frontend. (6 sp)
 - KG-9: Pantallas de autenticación y registro de usuarios. (6 sp)
- Sprint 4: Gestión de ficheros
 - KG-10: Pantallas de recuperación y guardado de ficheros. (6 sp)
 - KG-11: Implementación del sistema de cifrado de ficheros. (4 sp)
- Sprint 5: Homogeneización del frontend y despliegue del proyecto
 - KG-12: Homogeneización de diseño de la aplicación. (6 sp)
 - KG-13: Despliegue de backend en Google Cloud. (2 sp)

- KG-14: Integración de frontend y backend en Google Cloud. (3 sp)

BACKLOG

Puesto que de esta forma se adapta mejor a los requisitos y la planificación del proyecto, el backlog desde un inicio está dividido en Sprints.

Las siguientes imágenes constituyen una representación del backlog en Jira durante las primeras etapas de desarrollo.

(Ilustración 5) Backlog de Jira

☒ Tablero Sprint 1	27 oct – 3 nov (3 incidencias)	4	4	3	Completar sprint	...
Proyecto de backend base y gestión de usuarios						
☒ KG-1	Creación de la base del proyecto (backend)	BACKEND	FINALIZADA	3	AS	
☒ KG-2	Implementación del la gestión de usuarios mediante firestore (backend)	BACKEND	PRUEBAS	4	AS	
☒ KG-3	Implementación de autenticación de usuarios mediante token JWT (backend)	BACKEND	TAREAS POR HACER	4	AS	
☐ Tablero Sprint 2	3 nov – 10 nov (4 incidencias)	12	3	0	Iniciar sprint	...
Integración con firestore database y gestión de ficheros						
☒ KG-4	Integración con firestore database (backend)	BACKEND	TAREAS POR HACER	2	AS	
☒ KG-5	Implementación de la lógica de gestión de ficheros (backend)	BACKEND	TAREAS POR HACER	4	AS	
☒ KG-6	Implementación del guardado de ficheros (backend)	BACKEND	TAREAS POR HACER	3	AS	
☒ KG-7	Implementación de la recuperación de ficheros (backend)	BACKEND	TAREAS POR HACER	3	AS	
☐ Tablero Sprint 3	10 nov – 17 nov (2 incidencias)	12	0	0	Iniciar sprint	...
Proyecto de frontend base y gestión de usuarios						
☒ KG-8	Creación de la base del proyecto (frontend)	FRONTEND	TAREAS POR HACER	6	AS	
☒ KG-9	Pantallas de autenticación y registro de usuarios (frontend)	FRONTEND	TAREAS POR HACER	6	AS	
☐ Tablero Sprint 4	17 nov – 24 nov (2 incidencias)	10	0	0	Iniciar sprint	...
Gestión de ficheros en frontend						
☒ KG-10	Pantallas de recuperación y guardado de ficheros (frontend)	FRONTEND	TAREAS POR HACER	6	AS	
☒ KG-11	Implementación del sistema de cifrado de ficheros (frontend)	FRONTEND	TAREAS POR HACER	4	AS	
☐ Tablero Sprint 5	24 nov – 1 dic (3 incidencias)	11	0	0	Iniciar sprint	...
Homogeneización del frontend y despliegue del proyecto						
☒ KG-12	Homogeneización de diseño de la aplicación	INTEGRACIÓN Y DESPLI...	TAREAS POR HACER	6	AS	
☒ KG-13	Despliegue de backend en Google Cloud	INTEGRACIÓN Y DESPLI...	TAREAS POR HACER	2	AS	
☒ KG-14	Integración front con backend en Google Cloud	INTEGRACIÓN Y DESPLI...	TAREAS POR HACER	3	AS	

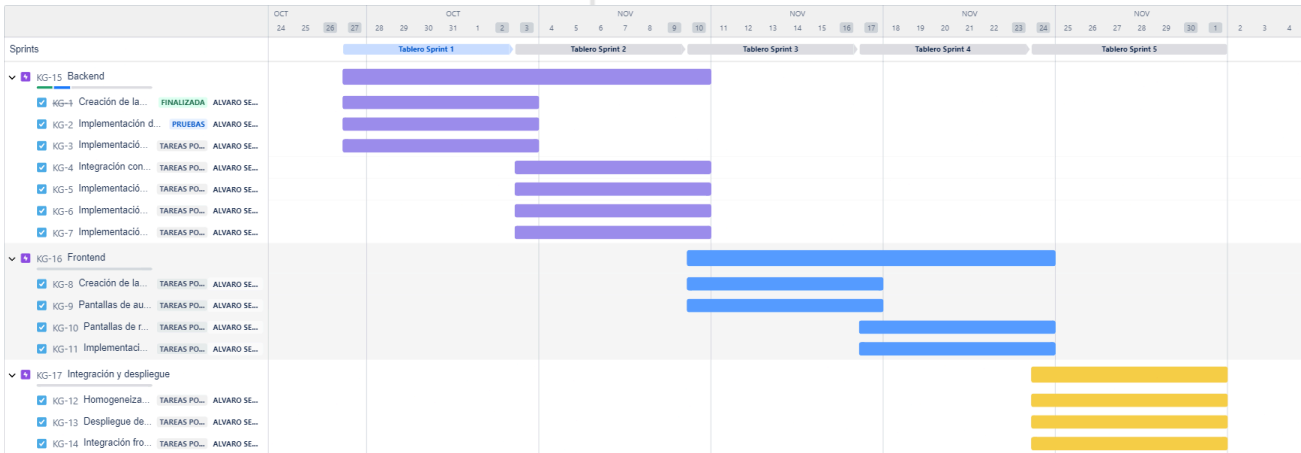
CRONOGRAMA

El proyecto está planificado para completar en cinco sprints de 1 semana cada uno. Comenzando el desarrollo el 20 de octubre de 2024 y finalizando el 29 de noviembre de 2024.

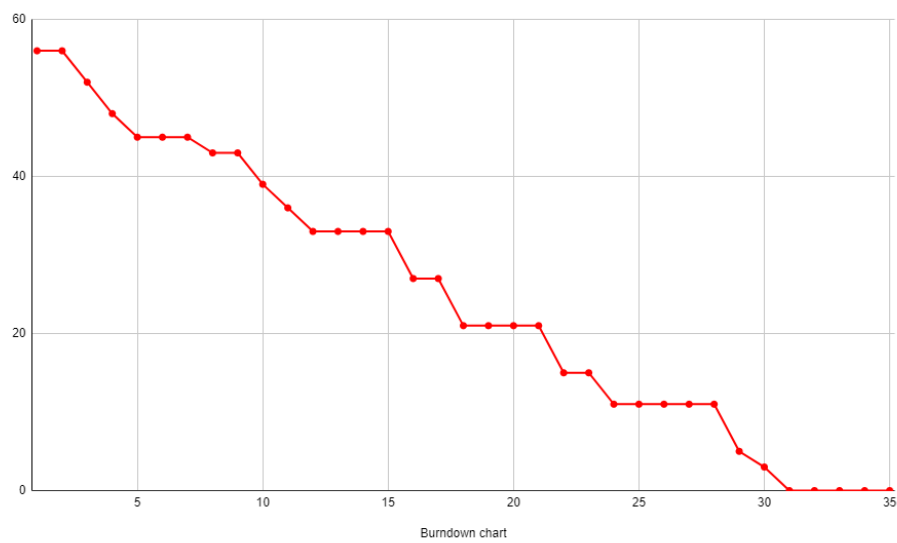
El proyecto se encuentra dividido en tres épicas principales: backend, frontend e integración y despliegue.

El siguiente diagrama de Gantt muestra la planificación del proyecto en base a sus tareas y épicas

(Ilustración 6) Diagrama de Gantt del proyecto



(Ilustración 7) Burndown chart de los story points en relación a los días planificados para los Sprints del proyecto



TECNOLOGÍAS UTILIZADAS

El proyecto ha requerido para su correcto funcionamiento un repertorio amplio y variado de herramientas.

A pesar de que la premisa y funcionalidad de KeyGuardian sea relativamente sencilla de comprender, su implementación ha requerido de una selección cuidadosa de tecnologías que permiten su desarrollo. En este apartado se explican estas herramientas principales y se discute su aporte y funcionalidad.

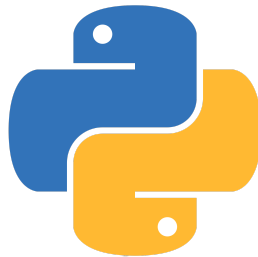
Resumen de tecnologías utilizadas:

- Python: Proporciona la base de la lógica del backend.
- JWT: Implementa la autenticación y autorización de usuarios de forma segura.
- Gunicorn: Optimiza el rendimiento del backend en entornos de producción.
- Flask: Estructura la API REST del backend y gestiona las solicitudes y respuestas.
- Firestore Authentication: Controla la autenticación de usuarios y la gestión de sesiones.
- Firestore Database: Almacena los datos de los usuarios y sus contraseñas de manera segura.
- Google Cloud: Aloja el backend en un entorno seguro, escalable y de alta disponibilidad.
- Docker: Facilita el despliegue en contenedores y la consistencia entre entornos de desarrollo y producción.
- Tailscale: Asegura el acceso remoto al backend durante el desarrollo y las pruebas.
- Android Studio: IDE oficial para el desarrollo en Android
- Kotlin: Proporciona base de la lógica para el frontend.

Cada herramienta contribuye de forma integral a crear un sistema seguro, escalable y fácil de mantener para la gestión de contraseñas, asegurando que tanto la privacidad del usuario como la eficiencia del sistema sean sostenibles.

BACKEND

PYTHON



Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, diseñado para ser fácil de leer y escribir. Fue creado a finales de los años 80 por Guido van Rossum y se lanzó en 1991.

Principales características:

- **Sintaxis simple y legible:** Python está diseñado para ser fácil de leer y entender. Su sintaxis es clara, y se basa en la indentación para definir bloques de código.
- **Interpretado y dinámico:** Python es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea, sin necesidad de ser compilado previamente. Además, es dinámico, por lo que los tipos de datos se asignan automáticamente y no requieren ser declarados explícitamente.
- **Multiparadigma:** Python admite varios estilos de programación, incluyendo programación orientada a objetos, programación imperativa, programación funcional y programación estructurada. Esto lo hace flexible y adecuado para diferentes tipos de proyectos.
- **Amplia biblioteca estándar:** Python incluye una extensa biblioteca estándar que ofrece una gran variedad de módulos y paquetes para realizar tareas comunes.
- **Portabilidad y compatibilidad:** Python es compatible con la mayoría de los sistemas operativos, como Windows, macOS y Linux.

Python fue el lenguaje elegido para desarrollar el backend del proyecto por tres características clave:

1. Interpretado: Al ser un lenguaje interpretado los tiempos para probar las funcionalidades recién desarrolladas se acortan considerablemente.
2. Compatibilidad con Firestore: Existen multitud de librerías para Python que facilitan la integración con Firestore, estas librerías cuentan además con una amplia comunidad detrás.
3. Portabilidad y compatibilidad: Al ser compatible con la mayoría de sistemas el uso de Python facilita también el despliegue del backend en la nube.



FIREBASE AUTHENTICATION



Firebase Authentication es un sistema de autenticación proporcionado por Firebase, una plataforma de desarrollo de aplicaciones de Google. Este sistema permite a los desarrolladores autenticar usuarios en sus aplicaciones de manera segura y sencilla, integrándose de forma nativa con otros servicios de Firebase.

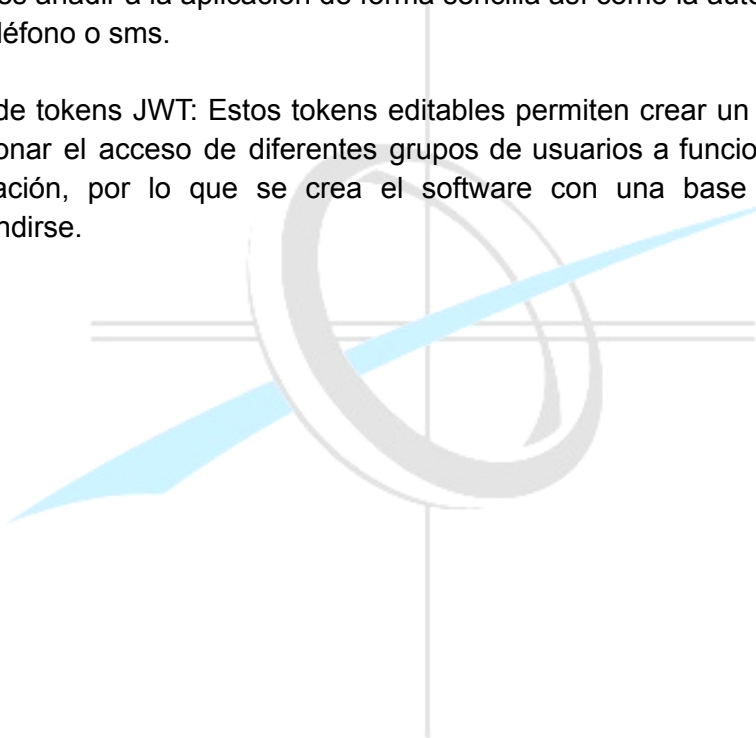
Principales características:

- Soporte para múltiples métodos de autenticación: Firebase Authentication admite varios métodos de autenticación:
 - Correo electrónico y contraseña
 - Autenticación con proveedores externos, como Google, Facebook, Twitter y Apple.
 - Autenticación anónima, para usuarios que no desean registrarse pero necesitan interactuar con la aplicación.
 - Autenticación con número de teléfono y verificación de código SMS.
 - Autenticación con proveedores personalizados, lo cual permite integrar sistemas de autenticación propios o de terceros.
- Seguridad de autenticación gestionada por Firebase: Firebase se encarga de manejar la seguridad y el almacenamiento de contraseñas, utilizando técnicas de cifrado seguras, lo que elimina la necesidad de que los desarrolladores manejen estos datos sensibles directamente.
- Integración nativa con Firestore y otros servicios de Firebase: Permite definir reglas de acceso basadas en el estado de autenticación de los usuarios.
- Tokens de usuario seguros: Una vez autenticado, Firebase genera un JWT (JSON Web Token) para cada usuario, que puede ser verificado por el servidor para controlar el acceso a los recursos de la aplicación.

- Soporte multiplataforma: Firestore Authentication es compatible con Android, iOS y web.

Los principales motivos por los que se eligió este método de autenticación para la aplicación son:

1. Seguridad gestionada por Firebase: El servicio lo proporciona una empresa reputable que se encarga de gestionar la seguridad y privacidad de los usuarios.
2. Múltiples métodos de autenticación: Una de las mejores ofertas de Firebase como método de autenticación es la gran variedad de métodos de autenticación que puedes añadir a la aplicación de forma sencilla así como la autenticación por número de teléfono o sms.
3. Uso de tokens JWT: Estos tokens editables permiten crear un sistema de roles para gestionar el acceso de diferentes grupos de usuarios a funciones específicas de la aplicación, por lo que se crea el software con una base sólida sobre la que expandirse.



FIRESTORE DATABASE



Firestore Database es una base de datos en tiempo real, NoSQL y gestionada por Firebase, diseñada para desarrollar aplicaciones que requieren sincronización de datos en tiempo real y escalabilidad automática. Firestore se enfoca en aplicaciones móviles y web, ofreciendo un modelo de datos flexible, fácil de usar, y altamente eficiente para gestionar datos estructurados y no estructurados.

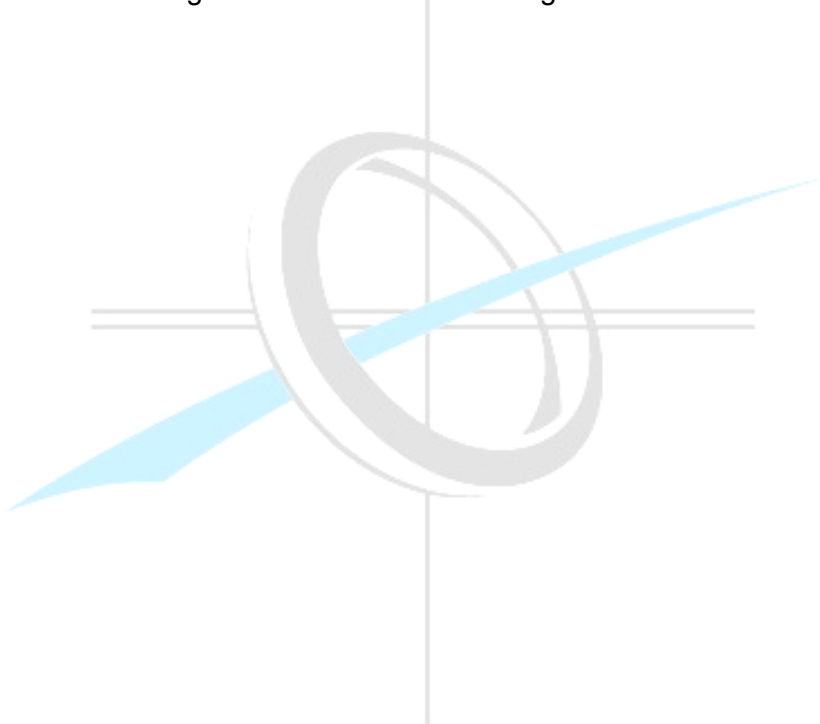
Principales características:

- **Modelo de datos basado en documentos:** Firestore organiza los datos en una estructura de documentos y colecciones:
 - **Documento:** Cada documento es un objeto JSON que contiene pares clave-valor y puede almacenar datos como strings, números, arrays, mapas y otros tipos de datos complejos.
 - **Colección:** Cada colección es un contenedor de documentos y puede anidar otras sub-colecciones, lo que permite organizar datos de manera jerárquica.
- **Consultas avanzadas:** Firestore soporta consultas complejas que permiten filtrar, ordenar y limitar resultados, y todo esto es posible sin requerir el uso de un servidor de backend adicional. Además, Firestore indexa automáticamente los datos para mejorar el rendimiento de las consultas.
- **Sincronización en tiempo real:** Firestore permite sincronizar datos en tiempo real entre dispositivos. Esto significa que cualquier cambio en la base de datos se refleja inmediatamente en todos los clientes conectados.
- **Escalabilidad automática:** Firestore está diseñado para escalar automáticamente según la demanda. Puede manejar grandes volúmenes de datos y usuarios.
- **Integración con Firebase y Google Cloud:** Firestore se integra de forma nativa con otros servicios de Firebase, como Authentication, Storage y Functions, así como con el ecosistema de Google Cloud.

- Soporte para acceso offline: Firestore almacena datos en caché localmente, lo que permite a los usuarios acceder a los datos sin conexión. Cuando el dispositivo vuelve a estar en línea, Firestore sincroniza automáticamente los datos locales con la nube.

Los principales motivos por los que se eligió este método de almacenamiento de datos son:

1. Escalabilidad: Puesto que Firestore Database escala de forma automática supone un punto de fallo menos, sin perder fiabilidad o eficiencia.
2. Modelo de datos: El modelo de datos que se almacenan en Firestore Database encaja perfectamente con la funcionalidad de la aplicación, que mediante el uso de colecciones y documentos se puede permitir que los usuarios tengan las credenciales catalogadas.



FLASK



Flask es un microframework de desarrollo web escrito en Python, diseñado para ser sencillo y flexible, permitiendo a los desarrolladores construir aplicaciones web y APIs de forma rápida y eficiente. Aunque "micro" indica que Flask es más ligero y tiene menos funcionalidades integradas en comparación con otros frameworks como Django, es altamente extensible y se puede expandir mediante módulos adicionales según las necesidades del proyecto.

Características Principales de Flask

- **Ligereza y Flexibilidad:** Flask sigue la filosofía de "lo mínimo necesario" (minimalismo), proporcionando sólo lo esencial para que el desarrollador tenga el control completo sobre las decisiones de arquitectura y estructura de la aplicación.
- **Soporte para APIs RESTful:** Flask facilita la creación de APIs REST, permitiendo a los desarrolladores definir métodos HTTP (GET, POST, PUT, DELETE, etc.) en cada ruta y recibir datos en formato JSON o mediante parámetros de solicitud.
- **Sistema de Desarrollo y Depuración:** Flask incluye un servidor de desarrollo que permite ejecutar la aplicación localmente y ver los cambios en tiempo real. Además, ofrece un modo de depuración que ayuda a identificar errores en el código, mostrando rastros detallados en el navegador.

Los principales motivos a la hora de utilizar Flask en la aplicación son:

1. El soporte para la creación de API RESTful: Este tipo de API permite una comunicación sencilla y segura con el frontend.
2. Sistema de desarrollo y depuración: Que aporta mayor facilidad a la hora de probar los endpoints y las funcionalidades del software.

GUNICORN



Gunicorn (Green Unicorn) es un servidor WSGI (Web Server Gateway Interface) para aplicaciones web en Python. Gunicorn actúa como intermediario entre la aplicación y el servidor web, gestionando múltiples solicitudes simultáneas de manera eficiente.

Principales características:

- **Modelo de Concurrencia de Workers:** Gunicorn utiliza un modelo de multiprocesos basado en trabajadores o "workers". Cada worker maneja una o más solicitudes, mejorando el rendimiento y la capacidad de respuesta de la aplicación.
- **Compatibilidad WSGI:** Como servidor WSGI, Gunicorn permite la interoperabilidad entre el servidor web (como Nginx) y cualquier aplicación Python que cumpla con el estándar WSGI. Esto es útil para aplicaciones desarrolladas con frameworks populares como Flask, Django y Pyramid.
- **Soporte para Distintos Tipos de Workers:** Gunicorn soporta varios tipos de workers que determinan cómo maneja las solicitudes: síncronos, asíncronos y basados en hilos.
- **Escalabilidad:** Gunicorn permite aumentar el número de workers o procesos según la carga y las necesidades de la aplicación. Esta escalabilidad horizontal permite que la aplicación responda mejor a picos de tráfico o altas demandas de procesamiento.

Los principales motivos por los que se ha elegido gunicorn para operar con la aplicación son la escalabilidad que aportan los workers de gunicorn a la hora de gestionar las peticiones API y por la seguridad que ofrece haciendo de intermediario mediante HTTPS.



Google Cloud Platform (GCP) es una suite de servicios en la nube proporcionada por Google, que permite desarrollar, desplegar y operar aplicaciones y servicios en una infraestructura global y escalable. GCP ofrece una amplia gama de servicios de computación, almacenamiento, redes, bases de datos, inteligencia artificial y herramientas de gestión, entre otros, diseñados para facilitar el desarrollo y despliegue de aplicaciones en la nube.

Opciones de Despliegue con Google Cloud:

GCP cuenta con varias opciones y servicios diseñados para desplegar aplicaciones de manera eficiente y ajustada a distintas necesidades. Aquí describo las opciones principales que suelen usarse para desplegar aplicaciones, especialmente aplicaciones web.

- **Google App Engine:** Google App Engine (GAE) es un servicio de Platform-as-a-Service (PaaS) que permite desplegar aplicaciones directamente sin necesidad de gestionar la infraestructura subyacente.
- **Google Kubernetes Engine (GKE):** Google Kubernetes Engine es un servicio de orquestación de contenedores basado en Kubernetes. Es ideal para aplicaciones complejas o para arquitecturas de microservicios que necesitan un alto control y flexibilidad.
- **Cloud Run:** Google Cloud Run es un servicio serverless que permite desplegar contenedores en la nube sin necesidad de gestionar servidores. Es ideal para aplicaciones que necesitan el poder y la flexibilidad de los contenedores pero no quieren complicarse con la administración de un clúster de Kubernetes.

Para el despliegue del backend de la aplicación se ha elegido Google Cloud Run que permite de forma sencilla el despliegue del backend como un microservicio automáticamente escalable.

DOCKER



Docker es una plataforma de código abierto que permite crear, desplegar y gestionar aplicaciones en contenedores. Los contenedores son entornos aislados y portátiles que incluyen todas las dependencias necesarias para ejecutar una aplicación, como bibliotecas, archivos de configuración y el sistema operativo base. Docker facilita el desarrollo y despliegue de aplicaciones, asegurando que se ejecuten de la misma manera en cualquier entorno, ya sea en una máquina local, un servidor en la nube o una infraestructura híbrida.

Características principales:

- **Portabilidad y Consistencia:** Docker permite empaquetar la aplicación y sus dependencias en una imagen de contenedor, lo que garantiza que la aplicación se ejecute de la misma manera en cualquier entorno.
- **Aislamiento de Recursos:** Cada contenedor funciona de forma aislada, con sus propios recursos de CPU, memoria y almacenamiento. Esto significa que los contenedores no interfieren entre sí, proporcionando un entorno seguro y controlado.
- **Escalabilidad y Flexibilidad:** Docker facilita la creación de entornos de desarrollo, pruebas y producción que pueden escalar rápidamente. Los contenedores pueden iniciarse o detenerse en segundos, y Docker se integra bien con herramientas de orquestación como Kubernetes para manejar el escalado automático.
- **Imágenes Ligeras y Reutilizables:** Las imágenes de Docker son ligeras y se construyen a partir de capas, lo que facilita la reutilización y la actualización de imágenes sin necesidad de reconstruirlas completamente.
- **Compatibilidad con Herramientas de Orquestación:** Docker se integra con herramientas como Kubernetes y Docker Swarm para la gestión y orquestación de contenedores en producción, facilitando el despliegue y administración de aplicaciones complejas.

El método de despliegue seleccionado, Google Cloud Run, requiere del uso de docker para la construcción de imágenes y arranque en la nube. Aun así, la herramienta es interesante también para hacer pruebas, ya que permite desplegar en local la misma imagen que arrancará en producción permitiendo hacer prueba de integración de manera sencilla.

TAILSCALE



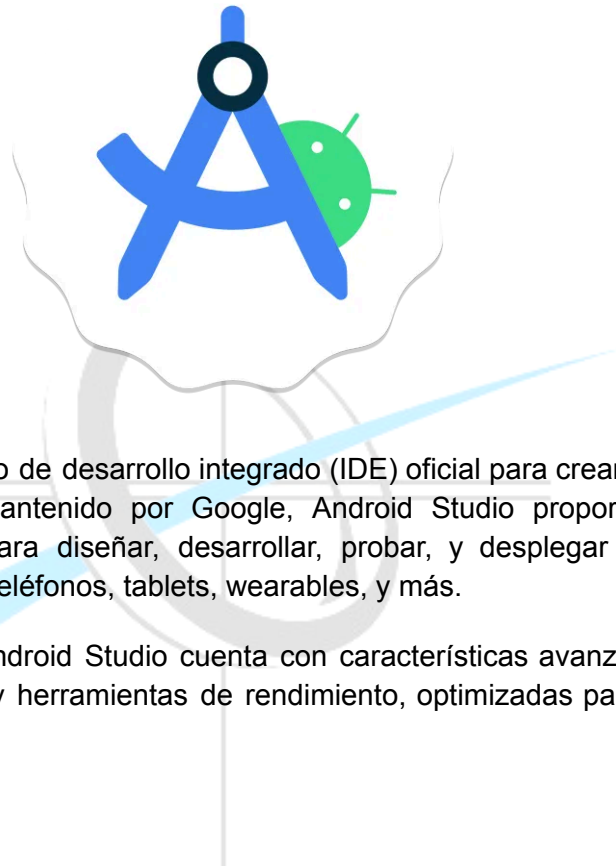
Tailscale es una solución de red privada virtual (VPN) que utiliza el protocolo WireGuard para crear redes privadas seguras y sencillas entre dispositivos, sin la complejidad de las configuraciones tradicionales de VPN. A diferencia de las VPN tradicionales, Tailscale está diseñado para facilitar la creación de redes privadas entre dispositivos y servidores, permitiendo el acceso seguro a recursos internos desde cualquier lugar y sin la necesidad de configurar routers, puertos o firewalls complejos.

Tailscale se destaca por su simplicidad de uso y administración, especialmente en organizaciones que requieren conexiones seguras para trabajadores remotos, equipos distribuidos, o para aplicaciones que necesitan acceder a recursos internos en una infraestructura segura.

En el proyecto, tailscale ha sido utilizado para facilitar la integración entre front y back. La VPN local que proporciona la herramienta se puede utilizar para facilitar la comunicación entre dispositivos sin necesidad de abrir puertos en tu red local.

FRONTEND

ANDROID STUDIO



Android Studio es el entorno de desarrollo integrado (IDE) oficial para crear aplicaciones de Android. Desarrollado y mantenido por Google, Android Studio proporciona todas las herramientas necesarias para diseñar, desarrollar, probar, y desplegar aplicaciones en dispositivos Android, como teléfonos, tablets, wearables, y más.

Basado en IntelliJ IDEA, Android Studio cuenta con características avanzadas de edición, diseño visual, depuración, y herramientas de rendimiento, optimizadas para el ecosistema de Android.

Principales características:

- **Editor de Código Inteligente:** Android Studio proporciona un editor de código avanzado que incluye completado de código, sugerencias de refactorización y verificación de errores en tiempo real.
- **Editor de Diseño Visual:** Incluye un editor de diseño visual que permite arrastrar y soltar componentes de la interfaz de usuario (UI) directamente en la pantalla de la aplicación. El editor de diseño admite tanto diseño basado en XML como en Compose, el nuevo toolkit de UI declarativa para Android.
- **Emulador de Android:** Android Studio incluye un emulador completo que permite probar aplicaciones en una gran variedad de dispositivos y configuraciones, desde diferentes versiones de Android hasta distintos tamaños de pantalla. Esto facilita el desarrollo sin la necesidad de dispositivos físicos.

- Integración con Git y Control de Versiones: Android Studio tiene soporte integrado para Git, lo que permite a los desarrolladores realizar cambios en el código y administrarlos con facilidad.
- Integración con Firebase: Firebase es una plataforma de desarrollo de aplicaciones de Google que proporciona servicios como autenticación, almacenamiento en la nube y análisis.
- Soporte para Lenguajes de Programación Kotlin y Java: Aunque históricamente Android se ha desarrollado en Java, Kotlin es ahora el lenguaje oficial recomendado para el desarrollo de aplicaciones de Android.
- Despliegue y Testing Automatizado: Android Studio permite ejecutar pruebas unitarias, pruebas instrumentadas y pruebas UI automatizadas, facilitando la verificación del funcionamiento correcto de la aplicación antes de su despliegue.

KOTLIN



Kotlin es un lenguaje de programación de tipado estático desarrollado por JetBrains. Está diseñado para ser conciso, expresivo, seguro y compatible con Java, lo que facilita su adopción en proyectos Android y en aplicaciones que anteriormente utilizaban Java. Desde 2017, Kotlin es el lenguaje oficial recomendado para el desarrollo de aplicaciones Android y ha ganado popularidad en múltiples plataformas, incluyendo backend, desarrollo web y multiplataforma.

Kotlin es interoperable con Java, lo que permite mezclar código Kotlin y Java en el mismo proyecto sin problemas. Además, gracias a Kotlin Multiplatform, los desarrolladores pueden compartir código entre diferentes plataformas, como Android, iOS, web y escritorio.

Principales características:

- Conciso: Kotlin permite escribir menos código para lograr la misma funcionalidad que en Java.
- Null Safety (Seguridad de Nulos): Kotlin tiene un sistema de tipos que ayuda a evitar los errores de punteros nulos (NullPointerException).
- Interoperabilidad con Java: Esto significa que se puede usar en proyectos Java existentes y que el código Kotlin puede llamar a código Java (y viceversa) sin problemas.
- Funciones de Extensión: Kotlin permite agregar nuevas funciones a clases existentes sin modificarlas directamente.

- Programación Funcional y Orientada a Objetos: Kotlin soporta tanto la programación orientada a objetos como la programación funcional.
- Expresiones Lambda y Funciones de Orden Superior: Kotlin permite el uso de funciones de orden superior (funciones que pueden recibir otras funciones como argumentos) y expresiones lambda.
- Coroutines para Concurrencia: Las coroutines de Kotlin facilitan la concurrencia y el manejo de tareas asíncronas.



REQUISITOS Y PLANIFICACIÓN

REQUISITOS

REQUISITOS FUNCIONALES

Backend:

RF-1	El backend debe permitir registrar usuarios.
RF-2	El backend debe permitir autenticar usuarios.
RF-3	El backend debe permitir eliminar usuarios.
RF-4	El sistema debe proporcionar un token JWT a cada usuario dedicado a la gestión de roles.
RF-5	El sistema debe permitir la creación de ficheros de tipo clave-valor.
RF-6	El sistema debe permitir el guardado y persistencia de ficheros de tipo clave-valor.
RF-7	El sistema debe permitir la edición de ficheros de tipo clave-valor.
RF-8	El sistema debe permitir la recuperación de ficheros eliminados.

Frontend:

RF-9	El frontend debe permitir el registro de nuevos usuarios.
RF-10	El frontend debe permitir la autenticación de usuarios.
RF-11	El frontend debe permitir la eliminación de usuarios.
RF-12	La aplicación debe permitir la creación de ficheros de tipo clave-valor.
RF-13	La aplicación debe permitir la edición de ficheros de tipo clave-valor.
RF-14	La aplicación debe permitir la eliminación de ficheros de tipo clave-valor.
RF-15	El frontend requerirá una contraseña para cifrar cada fichero nuevo.
RF-16	El frontend requerirá una contraseña para descifrar ficheros antes de editarlos y consultarlos.

REQUISITOS NO FUNCIONALES

RNF-1	El backend estará desarrollado en Python.
RNF-2	El sistema de gestión de usuarios estará implementado mediante Firebase Authentication.
RNF-3	El sistema de gestión de ficheros estará implementado mediante Firestore Database.
RNF-4	El backend utilizará Flask para exponer los endpoints para gestionar a los usuarios y los documentos de credenciales.
RNF-5	El sistema utilizará Gunicorn para la gestión de solicitudes.
RNF-6	El backend debe ser capaz de gestionar un número creciente de usuarios y solicitudes sin degradar el rendimiento.
RNF-7	La gestión de usuarios mediante token JWT debe garantizar la autenticación y autorización seguras.
RNF-8	La aplicación debe cumplir con las normativas de protección de datos aplicables.
RNF-9	Los ficheros deben ser cifrados antes de ser almacenados, protegiendo la información sensible.
RNF-10	El backend y el frontend deben estar diseñados para un tiempo de actividad continuo y alta disponibilidad, soportando interrupciones mínimas.
RNF-11	El backend debe estar desplegado en Google Cloud.
RNF-12	Las operaciones de recuperación y guardado de ficheros deben ser rápidas y eficientes, minimizando los tiempos de respuesta.
RNF-13	El sistema debe soportar múltiples solicitudes concurrentes sin ralentizar el rendimiento.
RNF-14	El frontend debe estar desarrollado con Android Studio.
RNF-15	Las pantallas de autenticación y gestión de ficheros deben ser intuitivas y fáciles de usar.
RNF-16	El diseño de la interfaz debe ser consistente en todo el proyecto para ofrecer una experiencia fluida al usuario.

RELACIÓN ENTRE REQUISITOS Y TAREAS

En este apartado se muestra la relación que existe entre las tareas planificadas, vistas en el módulo de scrumban, y los requisitos descritos anteriormente.

En cualquier proyecto es importante tener una visión clara del qué (los requisitos) y el cómo (las tareas planificadas) así como de la relación entre estas partes, simplificado un problema grande en fragmentos manejables y convirtiéndolos en tareas que se pueden completar.

Requisitos que acomete cada tarea:

- KG-1: Creación del proyecto backend base:
 - RNF-1, RNF-4, RNF-5 y RNF-13
- KG-2: Implementación de la gestión de usuarios mediante Firebase Authentication:
 - RF-1, RF-2, RF-3, RNF-2 y RNF-8
- KG-3: Implementación de la gestión de usuarios mediante token JWT:
 - RF-4 y RNF-7.
- KG-4: Integración con Firestore Database:
 - RF-5, RF-6, RF-7, RF-8, RNF-2 y RNF-8.
- KG-5: Implementación de la lógica de gestión de ficheros:
 - RF-5, RF-6, RF-7 y RF-8.
- KG-6: Implementación del guardado de ficheros:
 - RF-5, RF-6 y RF-7.
- KG-7: Implementación de la recuperación de ficheros:
 - RF-8.
- KG-8: Creación del proyecto base del frontend:
 - RNF-14.
- KG-9: Pantallas de autenticación y registro de usuarios:
 - RF-9, RF-10, RF-11 y RNF-15.
- KG-10: Pantallas de recuperación y guardado de ficheros:
 - RF-12, RF-13, RF-14 y RNF-15.

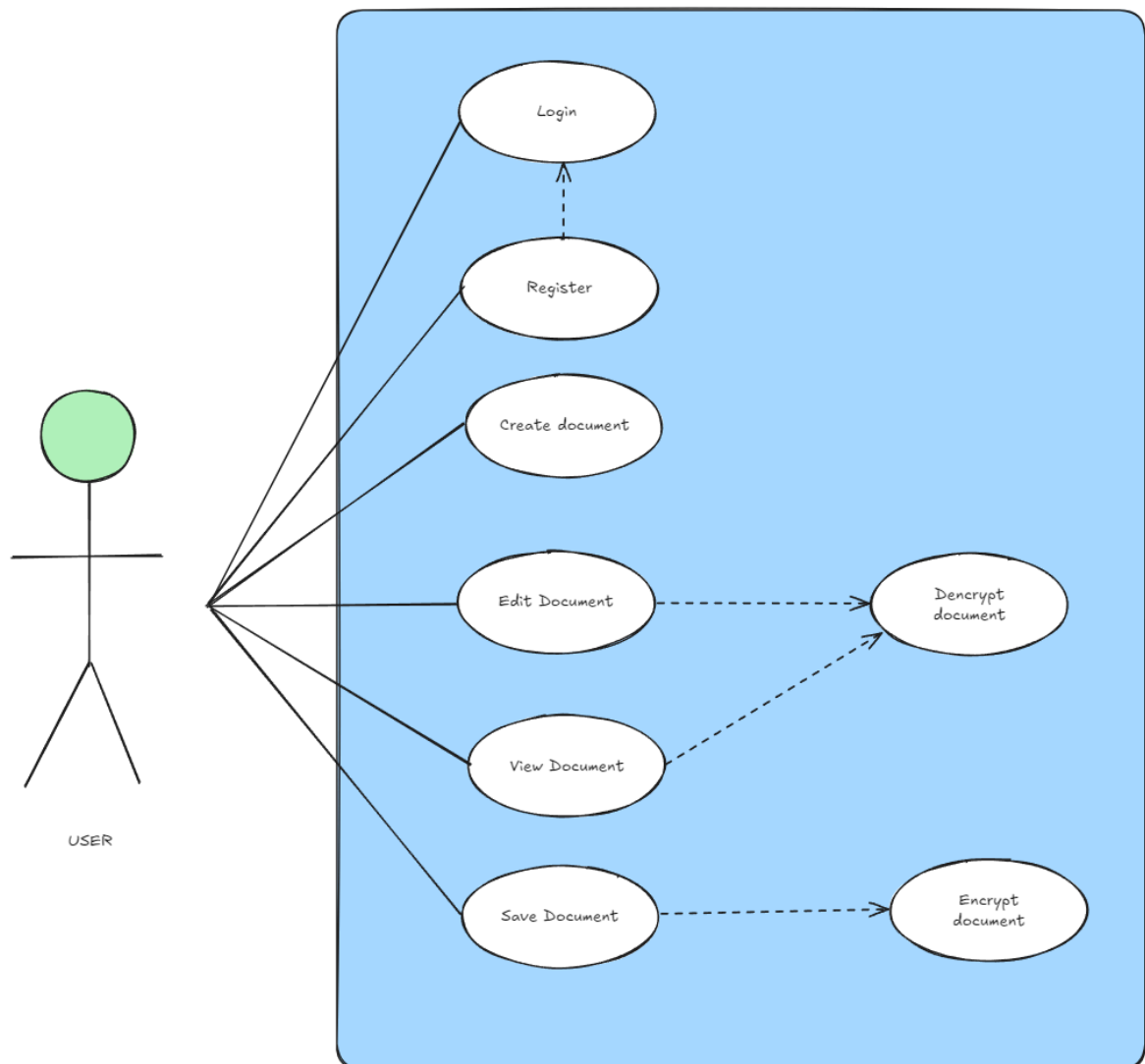
- KG-11: Implementación del sistema de cifrado de ficheros:
 - RF-15, RF-16 y RNF-9.
- KG-12: Homogeneización de diseño de la aplicación:
 - RNF-16
- KG-13: Despliegue de backend en Google Cloud:
 - RNF-6, RNF-10 y RNF-11
- KG-14: Integración de frontend y backend en Google Cloud:
 - RNF-10.



DIAGRAMAS

CASOS DE USO

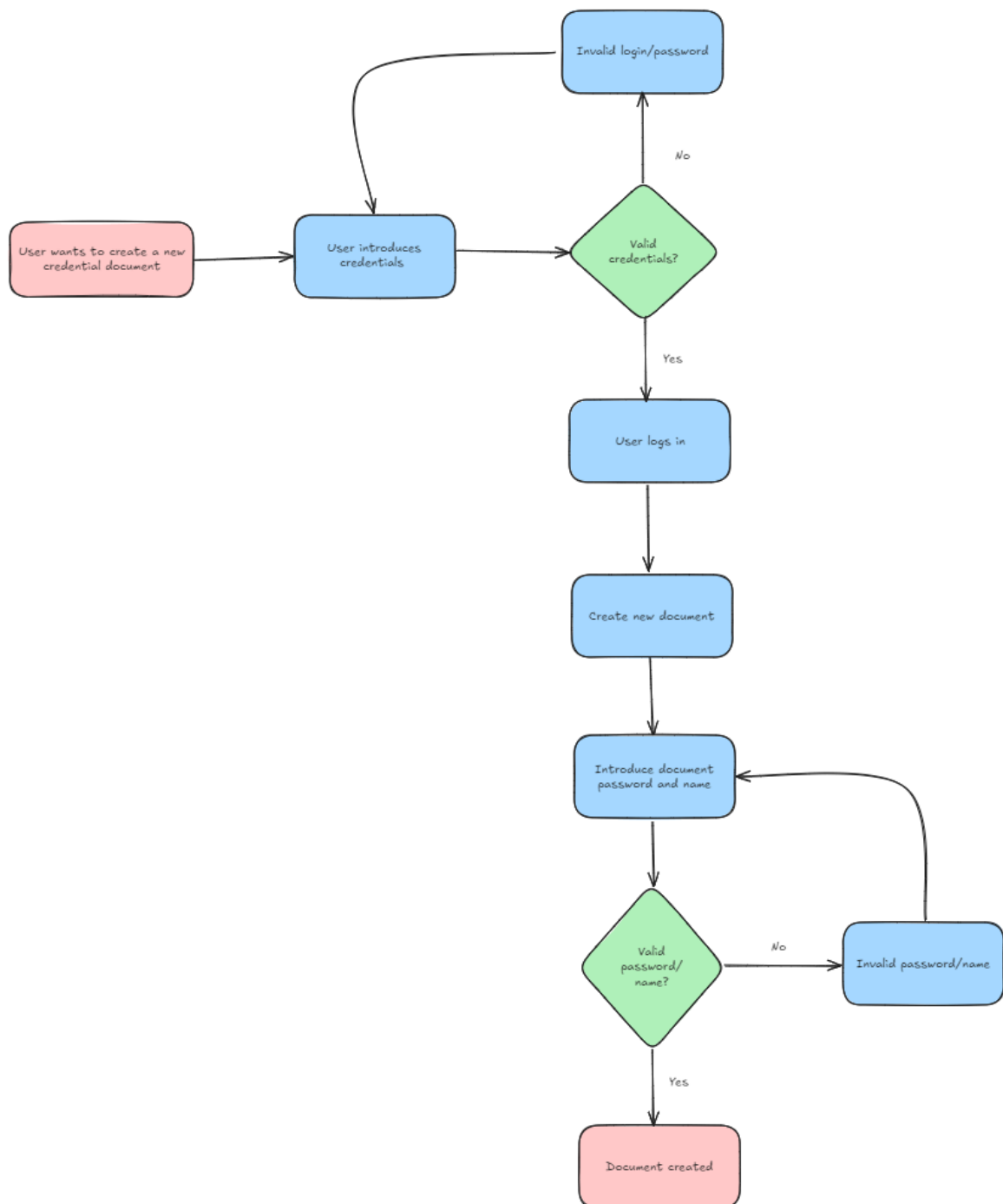
(Ilustración 8) Diagrama de casos de uso



FLUJO

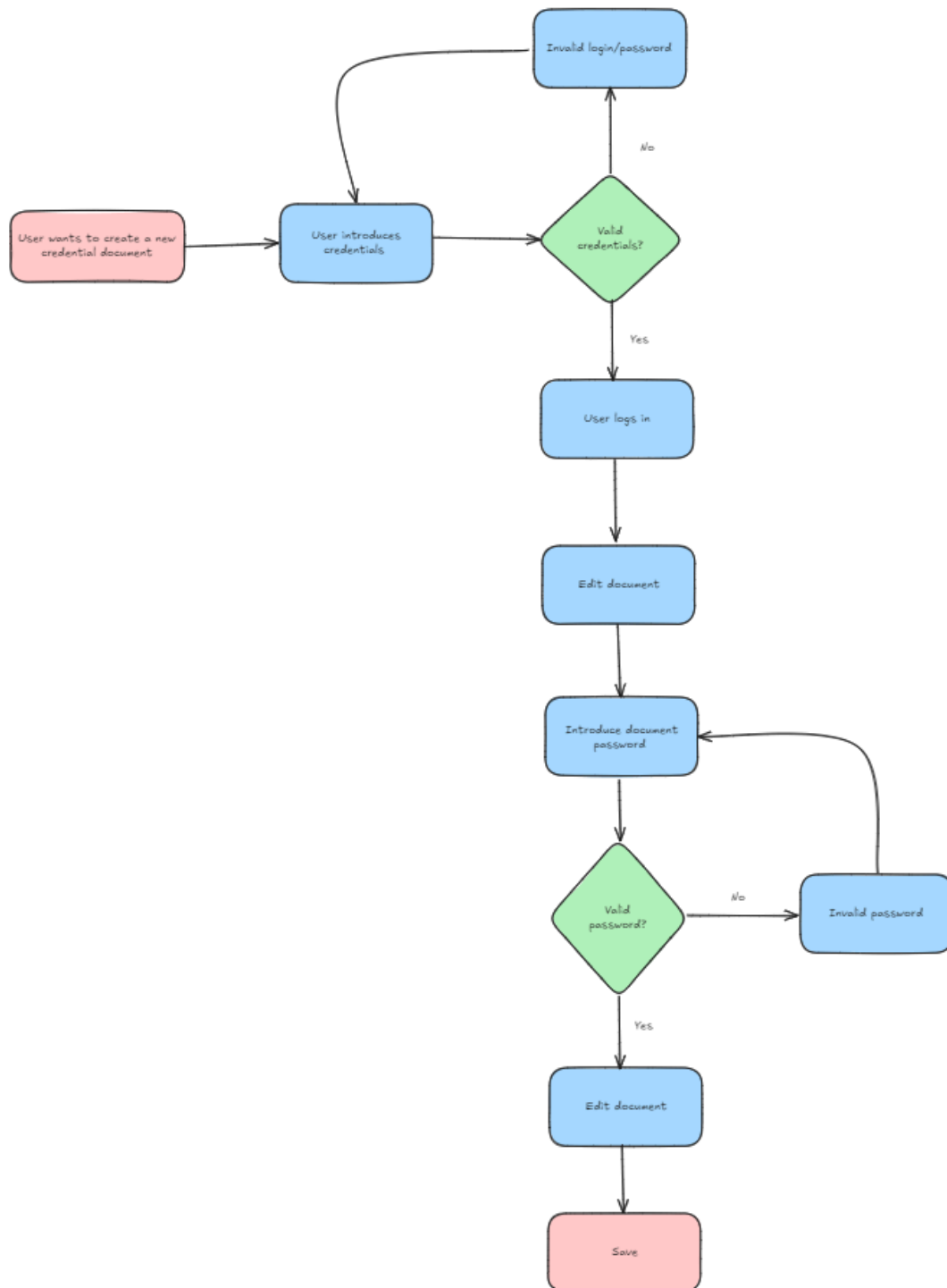
Crear documento nuevo:

(Ilustración 9) Diagrama de flujo en la creación de un documento nuevo



Editar documento:

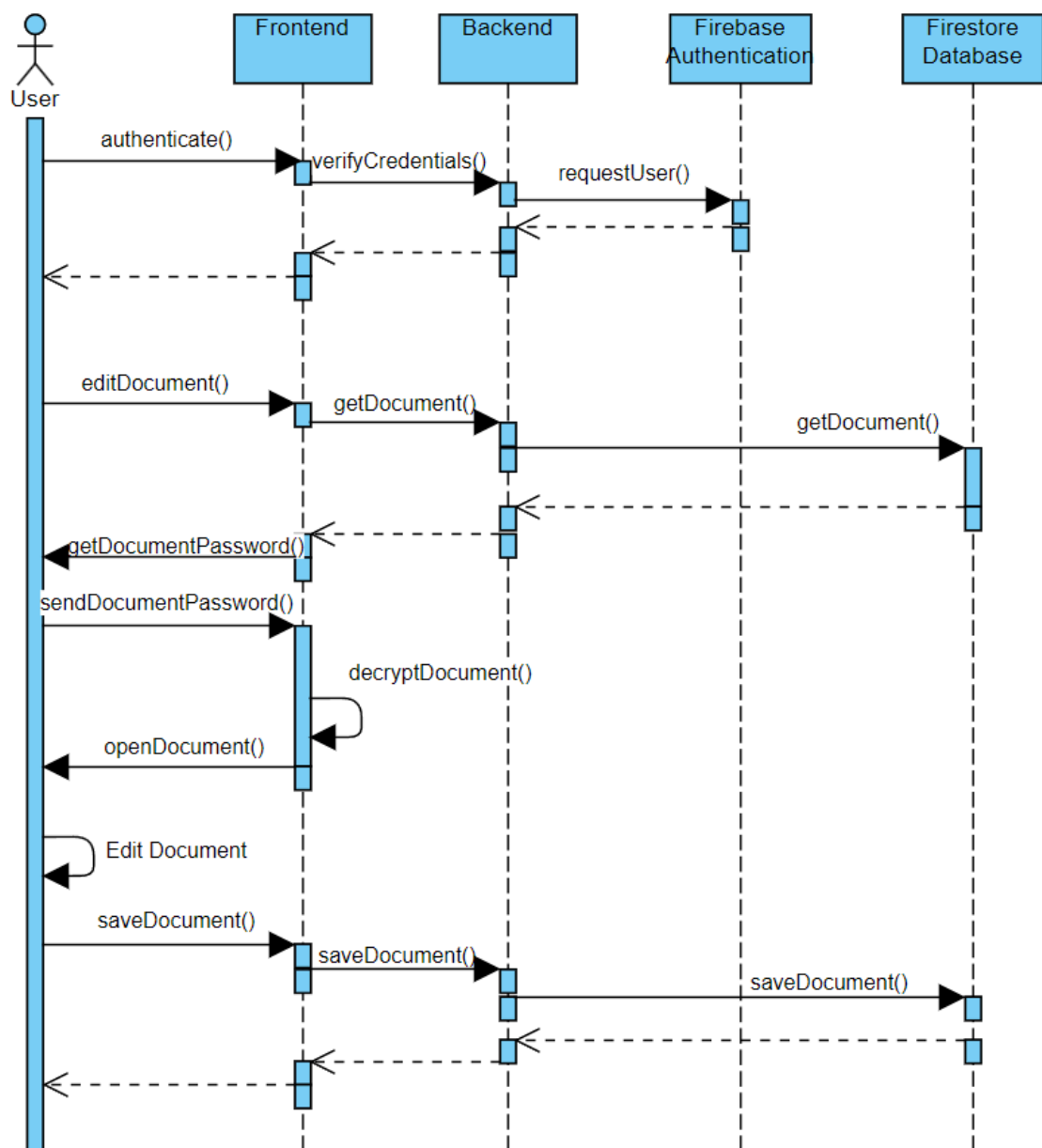
(Ilustración 10) Diagrama de flujo en la edición de un documento



SECUENCIA

Editar documento:

(Ilustración 11) Diagrama de secuencia



RESUMEN DE SPRINTS

SPRINT 1

El sprint 1 marca el inicio del proyecto y a grandes rasgos el objetivo a cumplir es sentar las bases del backend e implementar la gestión de usuarios mediante una integración con Firebase Authentication.

Dentro de las tareas a realizar durante el sprint se incluyen:

- KG-1: Creación del proyecto backend base. (3 sp)

El objetivo de esta tarea es la creación de un proyecto base en Python sobre el cual construir el resto de la aplicación, definiendo la estructura general y un endpoint GET/alive que informa sobre el estado de despliegue del backend.

- KG-2: Implementación de la gestión de usuarios mediante Firebase. (4 sp)

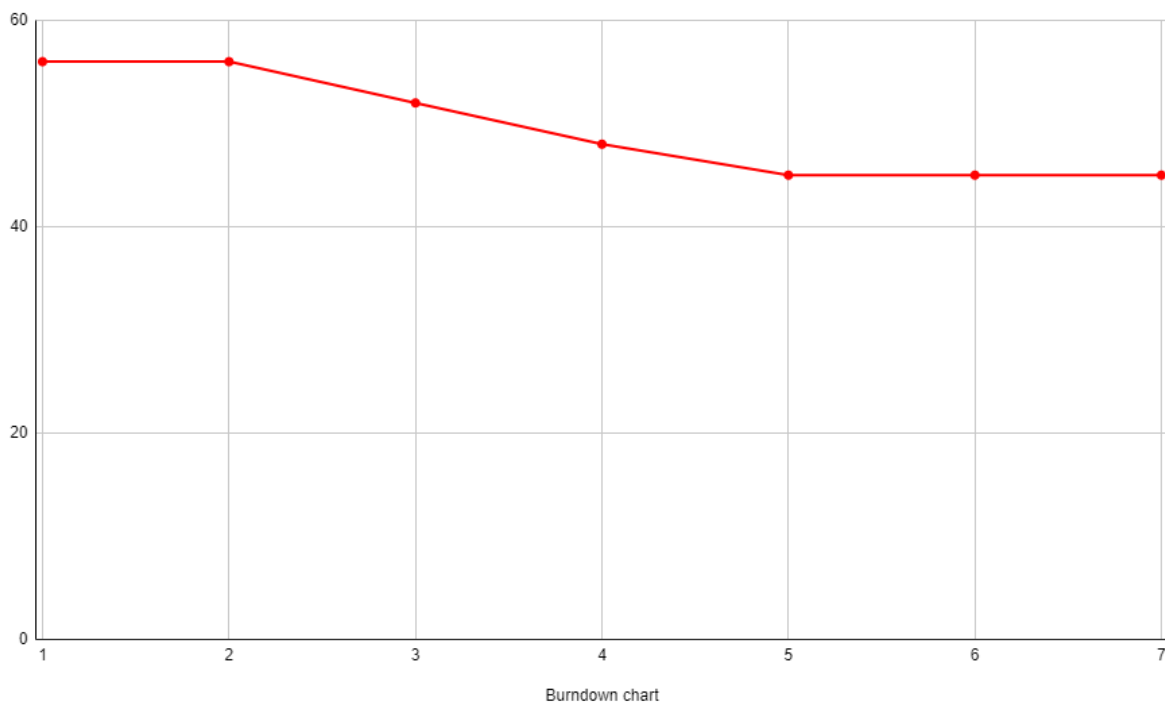
Durante el transcurso de esta tarea se integra la aplicación con Firebase Authentication y se crean endpoints para registrar y autenticar usuarios.

- KG-3: Implementación de la gestión de usuarios mediante token JWT. (4 sp)

En la última tarea del sprint se implementa la gestión de tokens JWT que complementa la autenticación, añadiendo un nivel adicional de seguridad y control sobre el acceso de los usuarios.

La finalización del primer sprint contribuye a la realización de aproximadamente el 50% de las tareas que componen la épica de *backend*.

(Ilustración 12) Burndown chart del Sprint 1



SPRINT 2

El sprint 2 se centra en la integración del backend con Firestore Database y la implementación de la gestión de ficheros, asegurando que la aplicación pueda manejar y almacenar información de manera efectiva.

Dentro de las tareas a realizar durante el sprint se incluyen:

- KG-4: Integración con Firestore Database. (2 sp)

Se lleva a cabo la integración completa del backend con Firestore Database, garantizando que la aplicación pueda interactuar de manera eficiente con la base de datos y almacenar los datos necesarios.

- KG-5: Implementación de la lógica de gestión de ficheros. (4 sp)

Esta tarea implica el desarrollo de la lógica necesaria para gestionar ficheros, estableciendo los procesos para cargar, organizar y eliminar archivos en el sistema.

- KG-6: Implementación del guardado de ficheros. (3 sp)

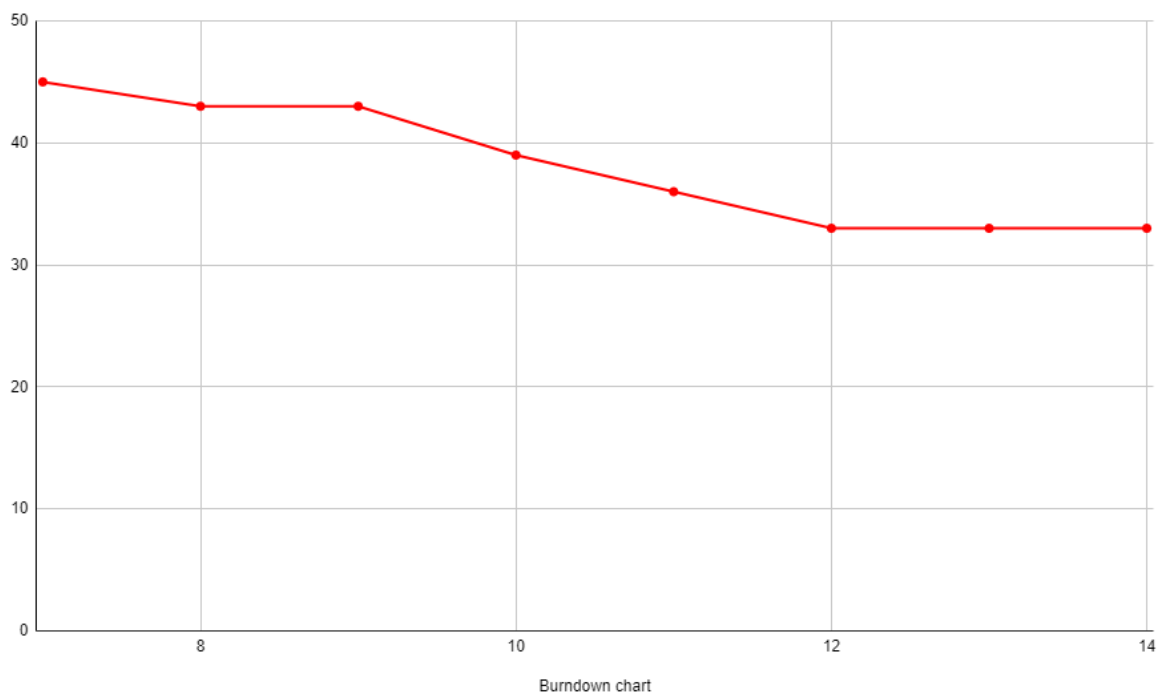
En esta tarea se implementa la funcionalidad de almacenamiento de ficheros en Firestore.

- KG-7: Implementación de la recuperación de ficheros. (3 sp)

Finalmente, se desarrolla la funcionalidad de recuperación de ficheros, proporcionando a los usuarios la capacidad de acceder a los archivos previamente almacenados y recuperar archivos eliminados.

La finalización del segundo sprint representa la finalización de todas las tareas que componen la épica de *backend*.

(Ilustración 13) Burndown chart del Sprint 2



SPRINT 3

El sprint 3 se centra en el desarrollo del proyecto base del frontend, asegurando que la aplicación tenga una interfaz sólida y moderna para la interacción con los usuarios.

Dentro de las tareas a realizar durante el sprint se incluyen:

- KG-8: Creación del proyecto base del frontend. (6 sp)

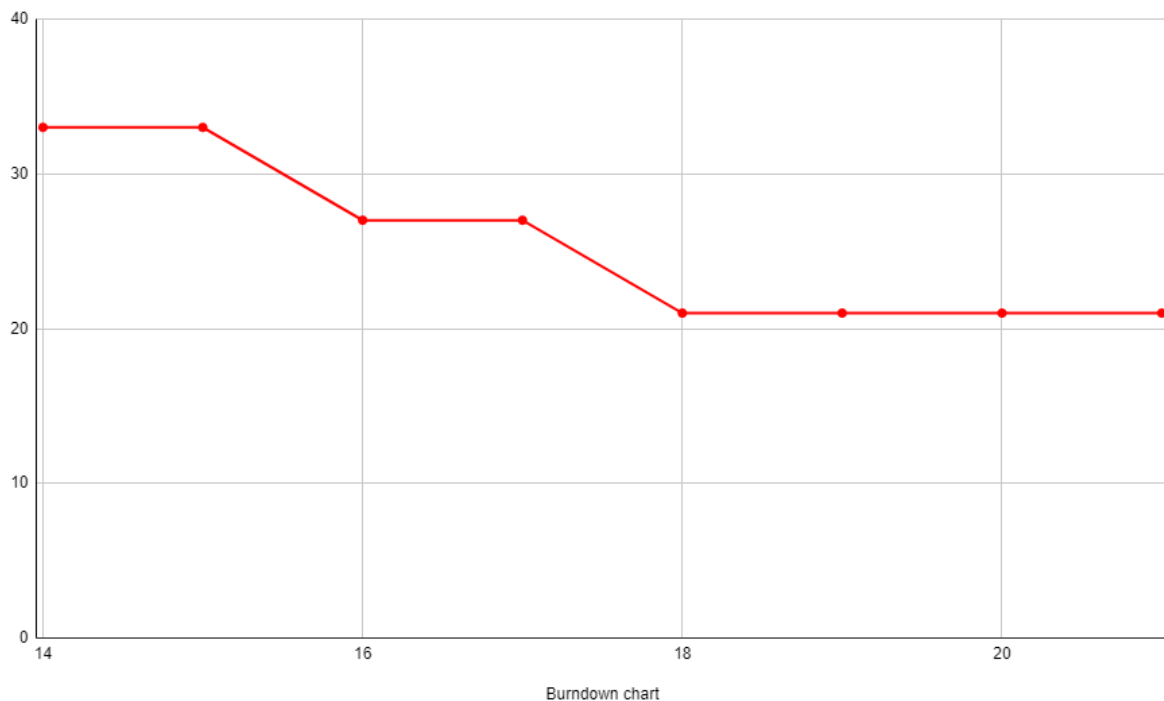
Se construye el proyecto base del frontend, estableciendo la estructura general de la aplicación y sentando las bases para las futuras funcionalidades.

- KG-9: Pantallas de autenticación y registro de usuarios. (6 sp)

Se diseñan e implementan las pantallas de autenticación y registro de usuarios, conectando estas funcionalidades con el backend y permitiendo que los usuarios interactúen con el sistema de gestión de usuarios del backend.

La finalización del sprint 3 contribuye a la realización de aproximadamente el 50% de las tareas que componen la épica de *frontend*.

(Ilustración 14) Burndown chart del Sprint 3



SPRINT 4

El sprint 4 se enfoca en el desarrollo de funcionalidades para la gestión de ficheros desde el frontend, asegurando que los usuarios puedan interactuar fácilmente con sus archivos.

Dentro de las tareas a realizar durante el sprint se incluyen:

- KG-10: Pantallas de recuperación y guardado de ficheros. (6 sp)

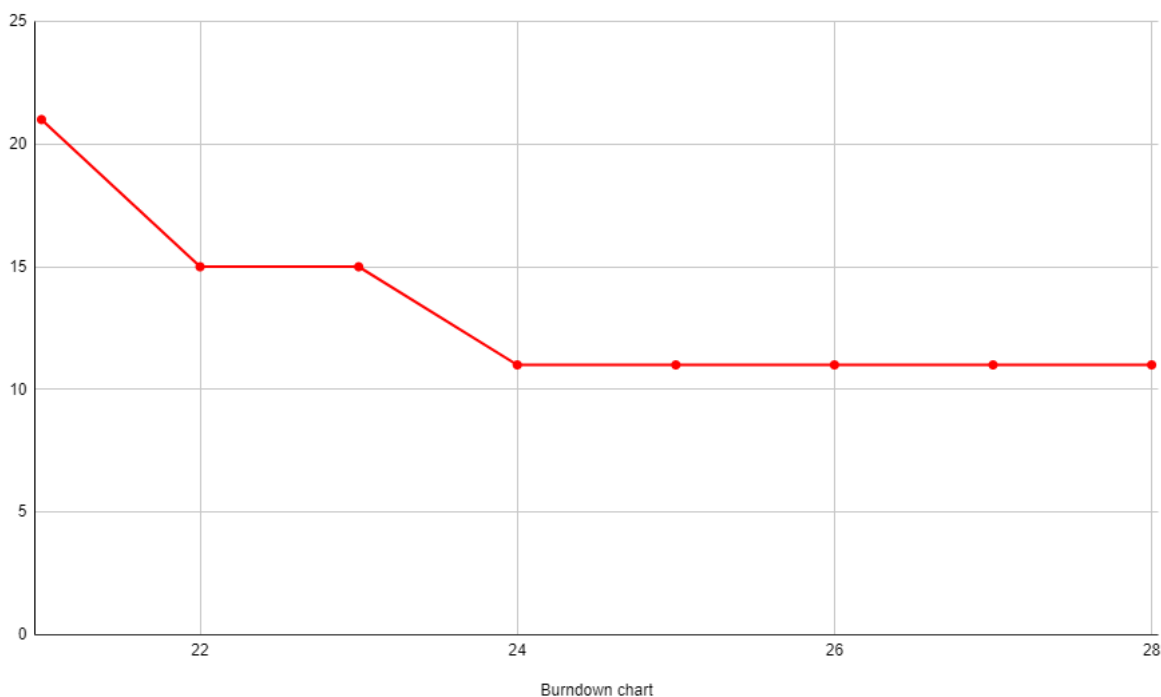
Se desarrollan las pantallas de la interfaz de usuario para la recuperación y almacenamiento de ficheros, facilitando la interacción con el sistema de archivos.

- KG-11: Implementación del sistema de cifrado de ficheros. (4 sp)

Se implementa un sistema de cifrado para los ficheros, mediante el cual durante la creación de un fichero el usuario establece una clave para cifrarlo.

La finalización del cuarto sprint representa la finalización de todas las tareas que componen la épica de *frontend*.

(Ilustración 15) Burndown chart del Sprint 4



SPRINT 5

El sprint 5 está destinado a unificar el diseño del frontend y a desplegar la aplicación completa en la nube, asegurando que el proyecto sea accesible y funcional.

Dentro de las tareas a realizar durante el sprint se incluyen:

- KG-12: Homogeneización de diseño de la aplicación. (6 sp)

Se lleva a cabo un proceso de homogeneización del diseño, unificando la estética y la experiencia de usuario de la aplicación para garantizar una apariencia y funcionalidad consistentes.

- KG-13: Despliegue de backend en Google Cloud. (2 sp)

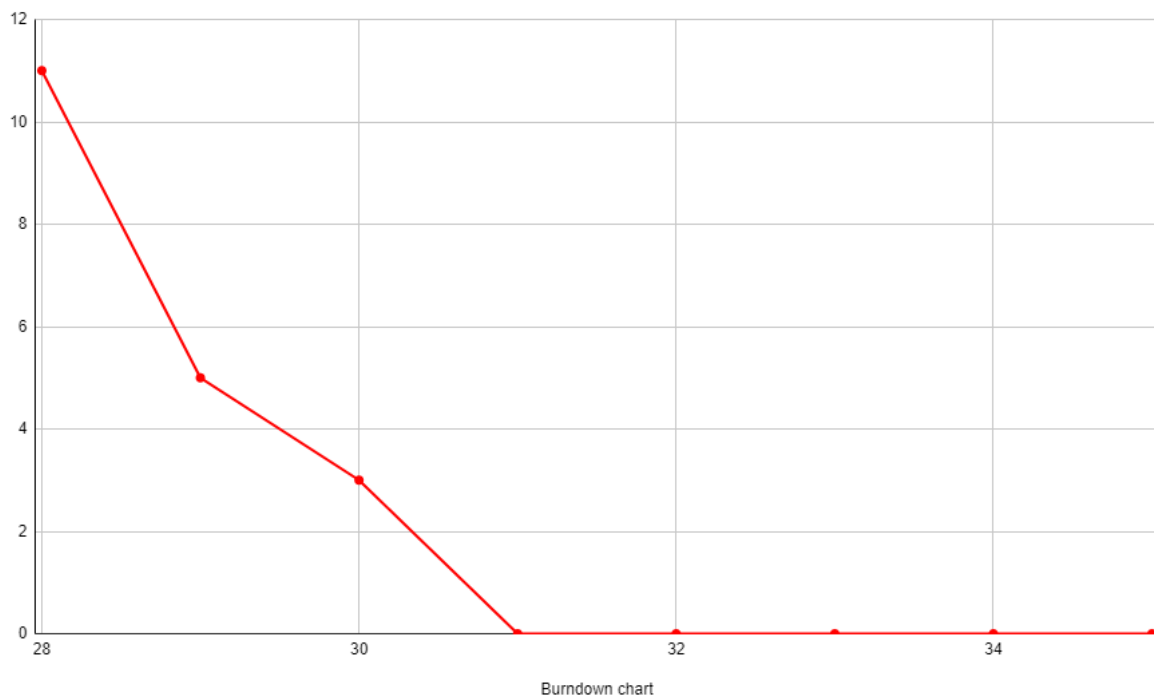
Se realiza el despliegue del backend en Google Cloud, asegurando que la aplicación esté disponible y se ejecute con un rendimiento adecuado.

- KG-14: Integración de frontend y backend en Google Cloud. (3 sp)

Se integra el frontend con el backend, completando la infraestructura del proyecto y asegurando que ambos sistemas funcionen correctamente.

La finalización del cuarto sprint representa la finalización de todas las tareas que componen la épica de *integración*.

(Ilustración 16) Burndown chart del Sprint 5



PROYECTO

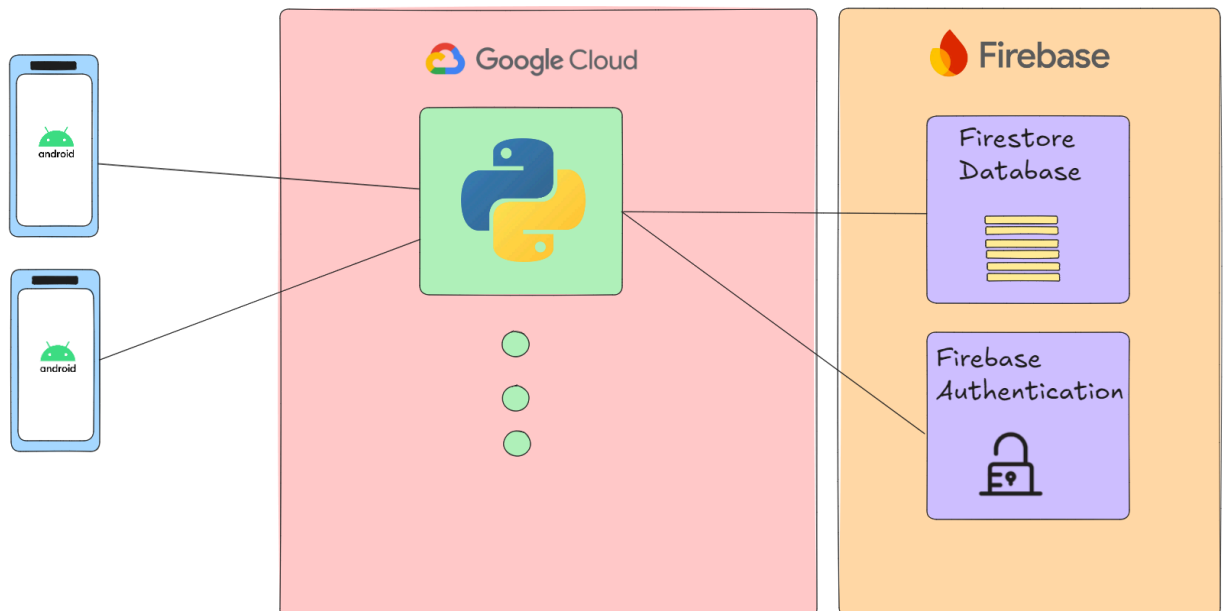
ARQUITECTURA DEL PROYECTO

El proyecto está compuesto por dos partes principales, backend (desarrollado en python) y frontend (desarrollado en kotlin con Android Studio).

A grandes rasgos la arquitectura del proyecto consiste en un despliegue en la nube del backend con el cual interactúan los usuarios a través de la aplicación móvil.

El backend a su vez gestiona los datos de los usuarios en la aplicación mediante integraciones que tiene con Firestore Database Firebase Authentication.

(Ilustración 17) Arquitectura del proyecto

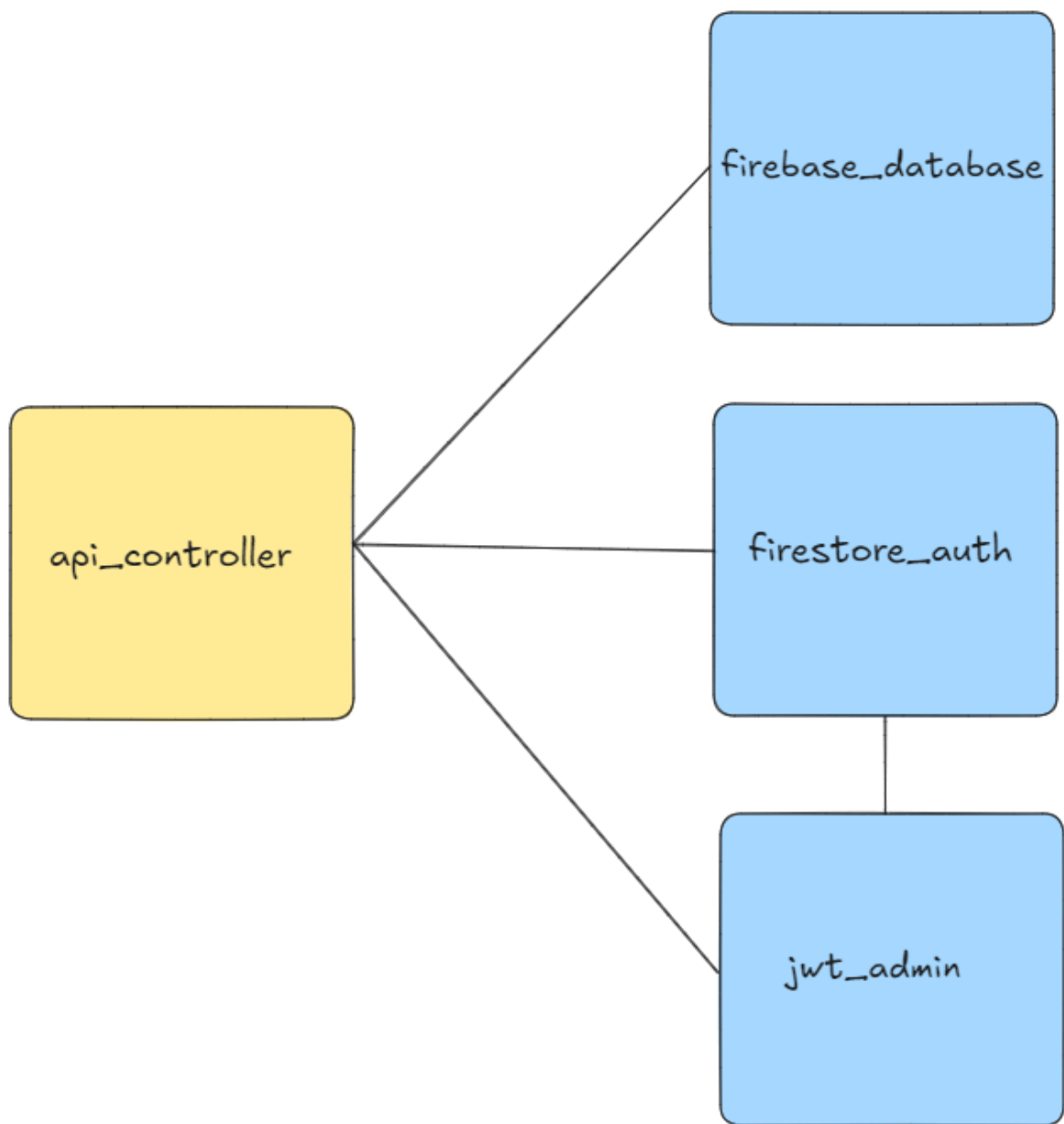


BACKEND

ARQUITECTURA BACKEND

Además de un `entrypoint.py` que se utiliza para cargar la configuración y arrancar el proyecto. El backend consta de las siguientes clases:

1. `APIController`: Esta clase define los endpoints que usa el proyecto y se encarga de recibir y validar las solicitudes de los usuarios y coordinar las interacciones entre las otras clases para ejecutar la lógica correspondiente en cada situación.
2. `FirebaseFirestore`: Esta clase se encarga de establecer una conexión con la base de datos de Firebase y gestionar las operaciones que se realizan con ella.
3. `FirestoreAuth`: esta clase implementa la lógica relacionada con la gestión de usuarios mediante Firestore Authentication, es decir, la verificación, el alta y la baja de usuarios. Así como de integrarse con `jwt_admin` para generar tokens JWT al autenticar a un usuario.
4. `JWTAdmin`: Esta clase es la encargada de generar tokens JWT para los usuarios autenticados y de verificar tokens JWT para garantizar que sean válidos y no hayan sido manipulados.



(Ilustración 18) Diagrama de arquitectura backend

FUNCIONALIDAD

A continuación, se describen en detalle todas las funcionalidades desarrolladas para para cada clase que aporta lógica al proyecto:

1. **FirebaseFirestore:**
 - Establecer un conexión con el Firebase del proyecto utilizando un certificado que se carga en la configuración del proyecto.
 - Extraer los nombres de todos los documentos asociados a un usuario.
 - Extraer el contenido de un documento en específico.
 - Crear un documento nuevo.
 - Actualizar un documento existente con datos nuevos.
 - Eliminar un documento específico.
2. **FirestoreAuth:**
 - Establecer un conexión con el Firestore Authentication del proyecto utilizando un certificado que se carga en la configuración del proyecto.
 - Registra a un usuario mediante email y contraseña.
 - Autentica a un usuario mediante email y contraseña.
3. **JWTAdmin:**
 - Genera un token jwt en el que se incluye como contenido el nombre del usuario.
 - Decodifica un token JWT y devuelve el contenido.

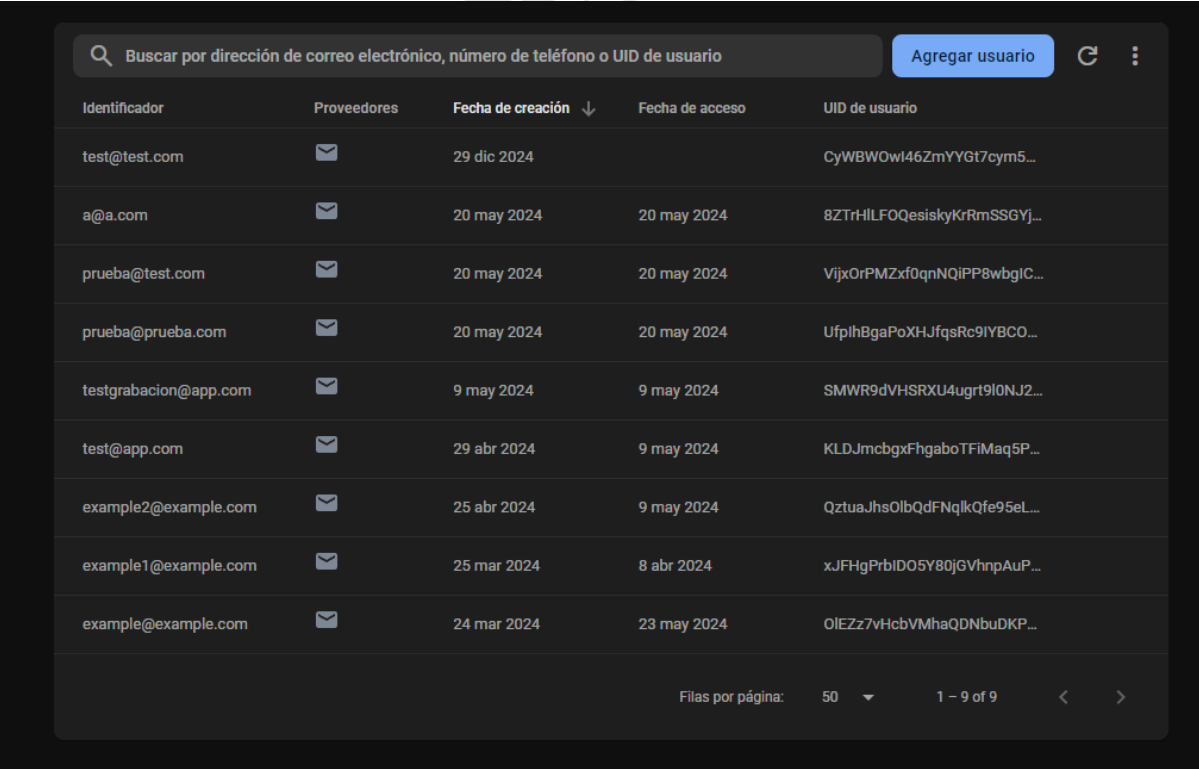
TRATAMIENTO Y PROCESAMIENTO DE DATOS

En este apartado se explicará el tratamiento que se realiza con los datos de los usuarios.

Estructura de almacenamiento de datos en los servicios de Google:

- Datos de autenticación: El modelo de datos de Firestore Authentication es muy cerrado, una vez que los usuarios se registran en la aplicación el único dato accesible es su dirección de correo electrónico, y las únicas operaciones de gestión de usuarios que se pueden realizar a mano son la eliminación y el registro de usuarios.

(Ilustración 19) Muestra de registro de usuarios en Firebase Authentication



The screenshot shows the Firebase Authentication console interface. At the top, there is a search bar with the placeholder text 'Buscar por dirección de correo electrónico, número de teléfono o UID de usuario' and a blue button labeled 'Agregar usuario'. Below the search bar is a table with the following columns: 'Identificador', 'Proveedores', 'Fecha de creación', 'Fecha de acceso', and 'UID de usuario'. The table contains 9 rows of user data. At the bottom right of the table, there is a pagination control showing 'Filas por página: 50' and '1 - 9 of 9'.

Identificador	Proveedores	Fecha de creación	Fecha de acceso	UID de usuario
test@test.com	📧	29 dic 2024		CyWBWOWl46ZmYYGt7cym5...
a@a.com	📧	20 may 2024	20 may 2024	8ZTrHILFOQesiskyKrRmSSGYj...
prueba@test.com	📧	20 may 2024	20 may 2024	VijxOrPMZxf0qnNQiPP8wbglC...
prueba@prueba.com	📧	20 may 2024	20 may 2024	UfplhBgaPoXHJfqsRc9IYBCO...
testgrabacion@app.com	📧	9 may 2024	9 may 2024	SMWR9dVHSRXU4ugrt9l0NJ2...
test@app.com	📧	29 abr 2024	9 may 2024	KLDJmcbgxFhgaboTFIMaq5P...
example2@example.com	📧	25 abr 2024	9 may 2024	QztuaJhsOlQdFNqIkQfe95eL...
example1@example.com	📧	25 mar 2024	8 abr 2024	xJFHgPrbID05Y80jGVhnpAuP...
example@example.com	📧	24 mar 2024	23 may 2024	OIEZz7vHcbVMhaQDNbuDKP...

- Almacenamiento de credenciales de los usuarios:

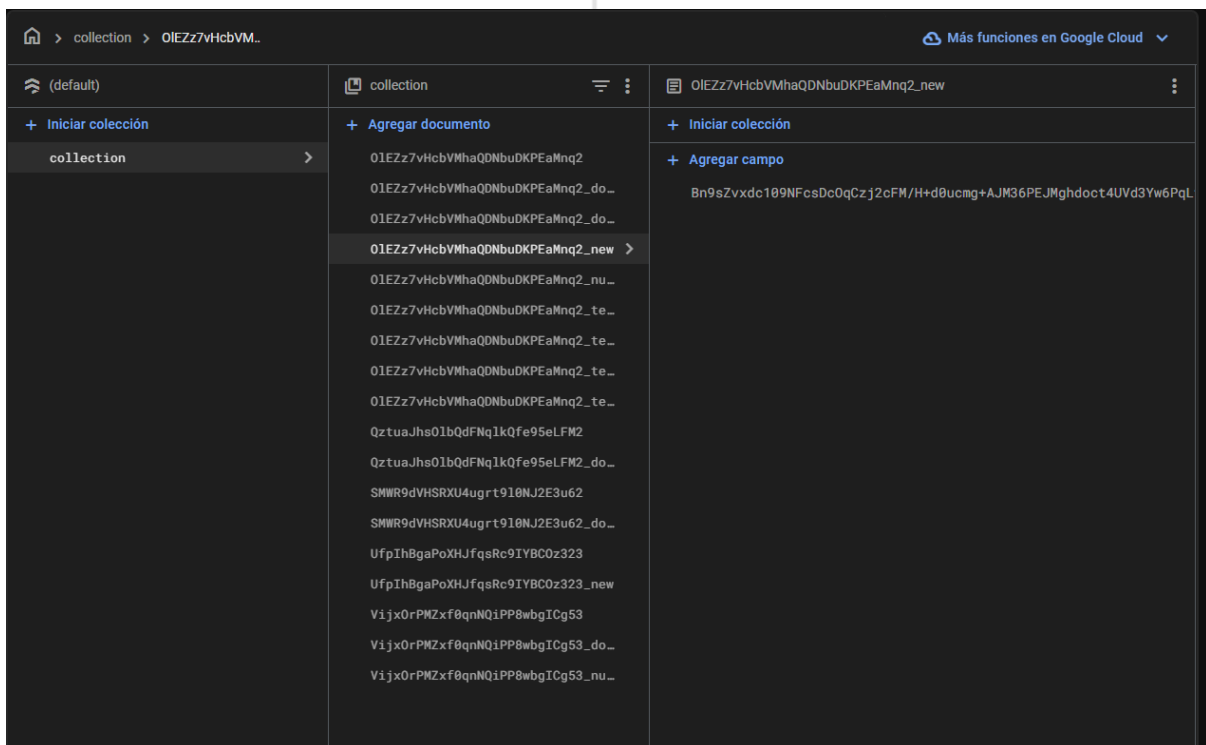
El sistema de almacenamiento de Firestore Database se basa en colecciones de documentos que contienen n parejas de elementos clave-valor.

El sistema que se ha elegido para almacenar documentos con las credenciales de los usuarios es el siguiente:

Un documento maestro por usuario que contiene una lista con los nombres de todos los documentos que pertenecen a ese usuario.

Los documentos de los usuarios en los que se recopilan parejas clave-valor con las credenciales que los usuarios quieran almacenar.

(Ilustración 20) Muestra de datos almacenados en Firestore



Tratamiento de datos en el proyecto:

Las dos únicas clases que de alguna forma procesan o transforman los datos de los usuarios son JWTAdmin y FirestoreDatabase. El resto de clases, ApiController y FirebaseAuth no tratan los datos de los usuarios en ninguna medida, tan solo sirven de pasarela entre el usuario y los servicios de Firestore.

JWTAdmin

El procesamiento de datos que lleva a cabo esta clase es muy reducido ya que tan solo incluye el identificador del usuario en el payload del token JWT para utilizar como medida de seguridad al realizar las operaciones.

FirestoreDatabase

Esta clase, además de conectar con la base de datos de Google y realizar operaciones de lectura y escritura, se encarga de organizar ficheros de los usuarios de forma que la estructura de datos de la aplicación encaje con la nativa de Firestore.

Para ello realiza las siguientes operaciones en función de lo que pretenda conseguir el usuario:

- Creación de un documento nuevo:
 1. La aplicación recibe mediante el endpoint de creación de documento nuevo el nombre del documento que se quiere crear y el id del usuario en cuestión, este último encriptado en el token JWT.
 2. El sistema realiza una query en Firestore para obtener el documento que tiene como nombre el id del usuario, que sirve de catálogo para almacenar los distintos documentos de ese usuario en concreto.
 3. En caso de que el documento exista, no realiza ninguna operación más y envía la información como respuesta al frontal. Si el documento a crear no está en el catálogo, inserta un registro en el catálogo con el nombre del documento nuevo.
 4. Acto seguido, se genera un documento nuevo en Firestore con nombre *id_usuario-nombre_documento*. Si ya existe un documento con ese nombre no genera ninguno nuevo puesto que se trata de un usuario recuperando un documento sobre el que ha realizado un borrado lógico.

- Edición de un documento:
 1. La aplicación recibe mediante el endpoint de edición de documento el nombre del documento que se quiere editar, el contenido del documento y el id del usuario en cuestión, este último encriptado en el token JWT.
 2. El sistema sustituye el contenido del documento de nombre *id_usuario-nombre_documento* en Firestore.

- Eliminación de un documento:
 1. La aplicación recibe mediante el endpoint de eliminación de documento el nombre del documento que se quiere eliminar y el id del usuario en cuestión, este último encriptado en el token JWT.
 2. El sistema realiza un query a Firestore para eliminar el el nombre del documento del fichero de gestión de documento en Firestore. De esta forma si el usuario lo desea, podrá recuperar el fichero en un futuro.

Cabe destacar que los documentos con los datos de los usuarios que maneja el backend no se almacenan en claro en Firestore. Tampoco se realiza ningún cifrado en el servidor. Los datos llegan encriptados desde el frontal con una clave privada que el usuario elige para cada documento, aumentando la seguridad y privacidad.

RESUMEN DE ENDPOINTS

Todas las operaciones que se realizan en el sistema ocurren a través de los siguientes endpoints. Todos los endpoints excepto el de registro y autenticación de usuarios están protegidos con roles mediante el uso de token JWT.

Lista de endpoints de la aplicación:

Endpoints de sistema

- **GET /alive:** Este endpoint devuelve simplemente sirve para obtener una respuesta como comprobación para observar si la aplicación está operativa y funcionando correctamente.

Endpoints de gestión de usuarios

- **POST /auth/register:** Este endpoint recibe un json con el email y la contraseña de un usuario nuevo para registrarlo en la aplicación.
 - Cuerpo:

```
{
  "email": "email",
  "password": "password"
}
```
 - Respuestas:
 - 200 - OK.

```
{
  "message": "User sign up successful",
  "user_id": "user_id"
}
```
 - 400 - Error en el cuerpo de la petición.

- **POST /auth/authenticate:** Este endpoint recibe un json con el email y la contraseña de un usuario nuevo para autenticarlo en la aplicación.

- Cuerpo:

```
{  
  "email": "email",  
  "password": "password"  
}
```

- Respuestas:

- 200 - OK.

```
{  
  "message": "Login successful",  
  "user_id": "user_id",  
  "token": "token"  
}
```

- 400 - Error en el cuerpo de la petición.

Endpoints de gestión de documentos

- **POST /documents/get_user_documents:** Este endpoint recibe un json con el id del usuario y devuelve una lista con los nombres de los documentos que tiene el usuario almacenados.

- Cuerpo:

```
{  
  "user": "user_id"  
}
```

- Respuestas:

- 200 - OK.

```
{  
  "documents": ["document1", "document2"]  
}
```

- 201 - OK vacío.

```
{  
  "documents": []  
}
```

- 400 - Error en el cuerpo de la petición.

- 403 - Acceso inválido.

- 500 - Error interno.

- **POST /documents/get_document:** Este endpoint recibe un json con el id del usuario y el nombre del documento y devuelve json con el contenido de dicho documento.
 - Cuerpo:

```
{
  "user_id": "user_id",
  "document": "document"
}
```
 - Respuestas:
 - 200 - OK.

```
{
  "response": {
    "usr": "key",
    "usr2": "key2"
  }
}
```
 - 201 - OK vacío.

```
{
  "response": {}
}
```
 - 400 - Error en el cuerpo de la petición.
 - 401 - Documento no existe.
 - 403 - Acceso inválido.
 - 500 - Error interno.
- **POST /documents/add_user_document:** Este endpoint recibe un json con el id del usuario y el nombre del documento y crea un registro para ese nuevo documento
 - Cuerpo:

```
{
  "user_id": "user_id",
  "document": "document"
}
```
 - Respuestas:
 - 200 - OK.
 - 400 - Error en el cuerpo de la petición.
 - 403 - Acceso inválido.
 - 500 - Error interno.

- **POST /documents/create_or_update_document:** Este endpoint recibe un json con el id del usuario, el nombre del documento y el contenido del documento y almacena ese documento en Firestore.

- Cuerpo:

```
{
  "user": "user_id",
  "document_name": "document",
  "data": {
    "usr": "key",
    "usr2": "key2"
  }
}
```

- Respuestas:

- 200 - OK documento guardado correctamente.
- 400 - Error en el cuerpo de la petición.
- 403 - Acceso inválido.
- 500 - Error interno.

- **POST /documents/delete_user_document:** Este endpoint recibe un json con el id del usuario y el nombre del documento y realiza un borrado lógico de ese documento.

- Cuerpo:

```
{
  "user_id": "user_id",
  "document": "document"
}
```

- Respuestas:

- 200 - OK.
- 400 - Error en el cuerpo de la petición.
- 403 - Acceso inválido.
- 500 - Error interno.

FRONTEND

ARQUITECTURA FRONTEND

En el desarrollo del proyecto, se ha optado por un modelo híbrido que combina elementos del patrón MVVM (Model-View-ViewModel) con una estructura más tradicional en la que algunas clases de la interfaz de usuario (UI) manejan directamente la comunicación con el backend. Esta decisión responde a la naturaleza del sistema, en el cual la mayor parte del procesamiento de datos y la lógica se realizan en el backend.

MVVM es una evolución de la arquitectura MVC (Model-View-Controller).

Los modelos difieren en pequeños detalles, sin embargo, importantes. Las principales diferencias se encuentran en el ViewModel que sustituye al Controller, y son:

- ViewModel expone datos que View puede consultar de forma pasiva en contrapartida con Controller que se comunica directamente con la vista.
- Los ViewModel y las vistas están desacoplados, por lo que un ViewModel puede ser utilizado por varias vistas.

MVVM se divide en tres capas principales:

1. Model (Modelo):
 - Representa los datos y la lógica de negocio.
 - Se encarga de recuperar datos desde una API, base de datos local, o cualquier otra fuente.
2. View (Vista):
 - Representa la interfaz de usuario.
 - Solo se encarga de mostrar los datos al usuario sin realizar lógica de negocio.
3. ViewModel (Modelo de Vista)
 - Intermediario entre el Modelo y la Vista.
 - Contiene la lógica de presentación y mantiene el estado de la interfaz de usuario.

FUNCIONALIDAD

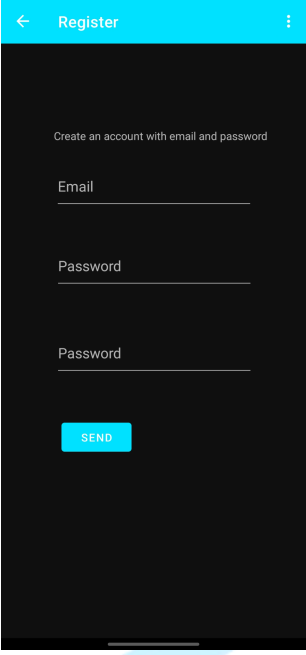
Resumen de clases

- AESUtils (Objeto estático): Maneja el cifrado y descifrado AES utilizando el modo "AES/CBC/PKCS5Padding".
- ApiService: Define las llamadas a la API usando Retrofit.
- DocumentDetailsActivity: Maneja la vista y funcionalidad de detalles de documentos.
- DocumentListAdapter: Adaptador para mostrar la lista de documentos en un RecyclerView.
- LoginFragment: Gestiona la pantalla de inicio de sesión.
- LogoutFragment (Extiende Fragment): Maneja la lógica de cierre de sesión.
- MainActivity: Contiene la estructura principal de la aplicación, incluyendo navegación y menú lateral.
- RegisterFragment: Gestiona el proceso de registro de usuario.
- RetrofitClient: Configura Retrofit para la comunicación con la API.
- SensitiveDataStore: Almacena datos sensibles temporalmente (como contraseñas).
- HomeFragment: Representa la pantalla principal de la aplicación.
- HomeViewModel: Mantiene y gestiona los datos de HomeFragment.

Vista previa de la aplicación

En esta sección se presentan capturas de pantalla de la aplicación junto con una breve descripción de sus funcionalidades principales.

(Ilustración 21) Pantallas de registro e inicio de sesión

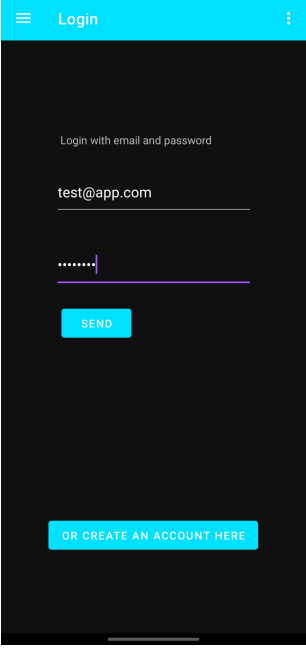
A screenshot of a mobile application's 'Register' screen. The screen has a dark background with a light blue header bar containing a back arrow and the word 'Register'. Below the header, there is a light blue text prompt: 'Create an account with email and password'. The form consists of three input fields: 'Email', 'Password', and 'Password' (repeated). Each field has a light blue underline. At the bottom of the form is a light blue button with the text 'SEND' in white capital letters.

Pantalla de registro.

A través de este formulario los usuarios se registran en Firebase Authentication.

Al realizar un registro exitoso, el usuario es automáticamente redirigido a la pantalla de inicio de sesión.

(Ilustración 21) Pantalla de registro

A screenshot of a mobile application's 'Login' screen. The screen has a dark background with a light blue header bar containing a menu icon, the word 'Login', and a settings icon. Below the header, there is a light blue text prompt: 'Login with email and password'. The form consists of two input fields: 'Email' (containing the text 'test@app.com') and 'Password' (containing masked characters '.....'). Each field has a light blue underline. Below the password field is a light blue button with the text 'SEND' in white capital letters. At the bottom of the screen is a light blue button with the text 'OR CREATE AN ACCOUNT HERE' in white capital letters.

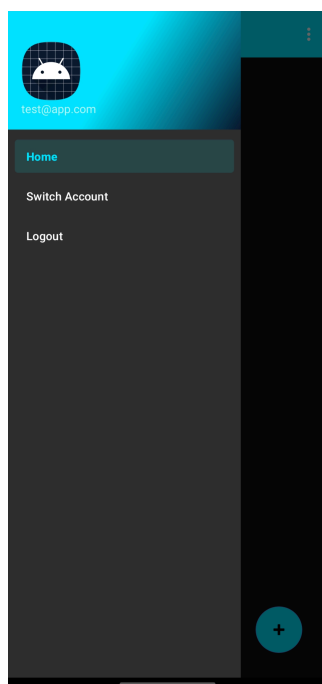
Pantalla de inicio de sesión.

Es la primera pantalla que se muestra al iniciar la aplicación.

A través de esta pantalla los usuarios pueden iniciar sesión en la aplicación mediante un usuario y contraseña previamente registrados. Obteniendo un token JWT para utilizar el resto de funcionalidades que se ofrecen.

Al realizar un inicio de sesión exitoso, el usuario es automáticamente redirigido a la pantalla principal de la aplicación.

(Ilustración 22) Pantalla de inicio de sesión

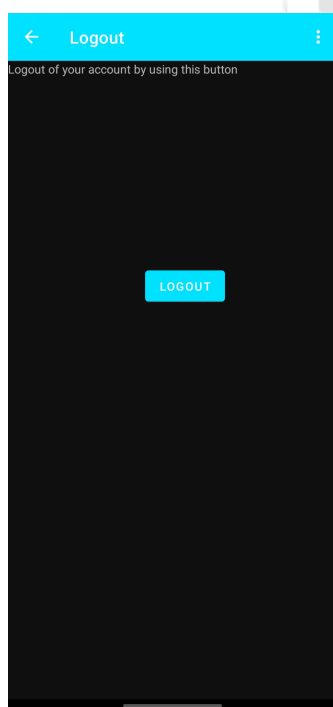


(Ilustración 23) Menú principal

Menú principal.

A esta lista se accede a través del menú hamburguesa de la parte superior izquierda de la aplicación.

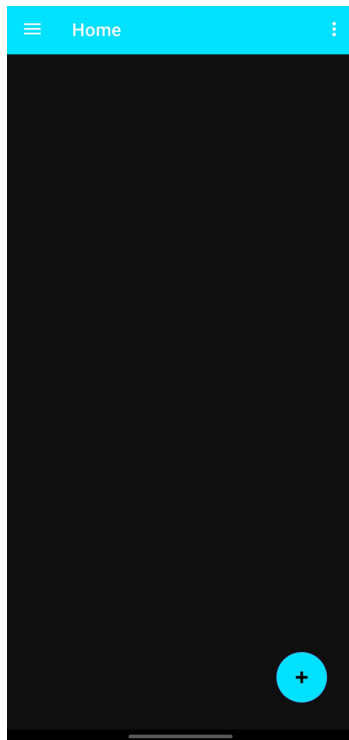
Permite el acceso tanto a la pantalla principal (*home*) como al *logout* y a una pantalla para cambiar de usuario (*switch account*).



(Ilustración 24) Pantalla de logout

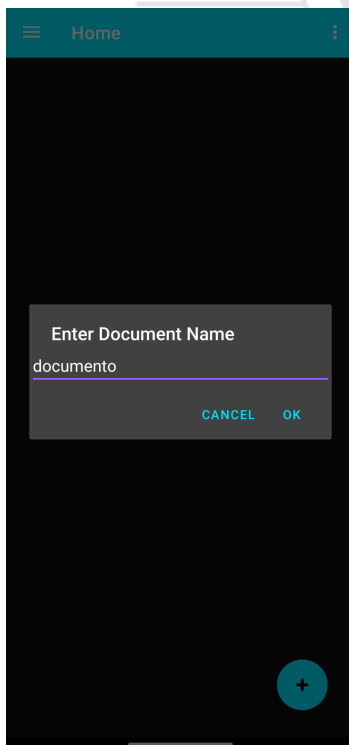
Pantalla de *logout*.

Permite al usuario cerrar la sesión en la aplicación al interactuar con el botón de *logout*. Esta opción elimina el token JWT de memoria, lo que provoca la redirección a la pantalla de inicio de sesión y que se actualicen el resto de pantallas acorde a este evento, vaciando la pantalla principal de documentos relacionados con la sesión anterior.



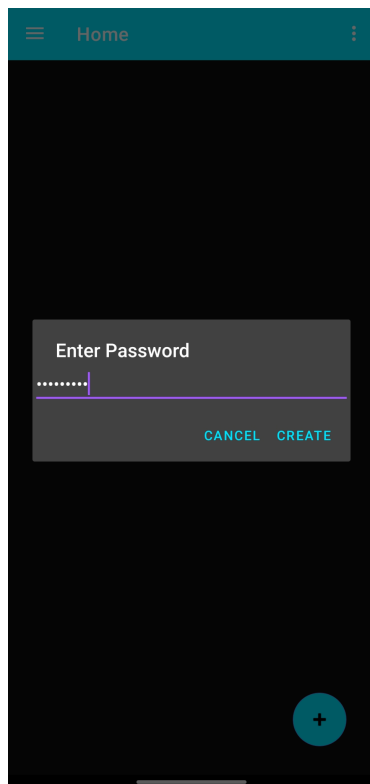
Pantalla principal vacía.

En esta pantalla se muestran los documentos del usuario. En el caso que se muestra en la imagen, el usuario no ha registrado ningún documento.



Proceso de creación de un documento (1).

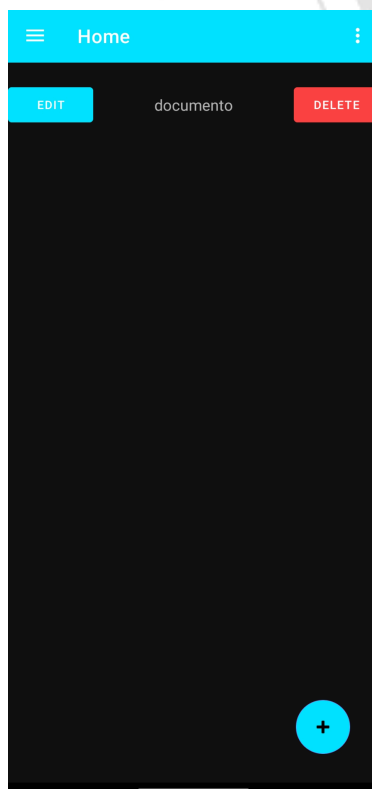
Al pulsar el botón (+) en la pantalla principal se despliega un *pop-up* que indica al usuario que introduzca un nombre para el documento que desea crear.



Proceso de creación de un documento (2).

Al introducir un nombre válido para el documento se despliega un *pop-up* que indica al usuario que introduzca una contraseña para cifrar los datos del documento que está a punto de crear.

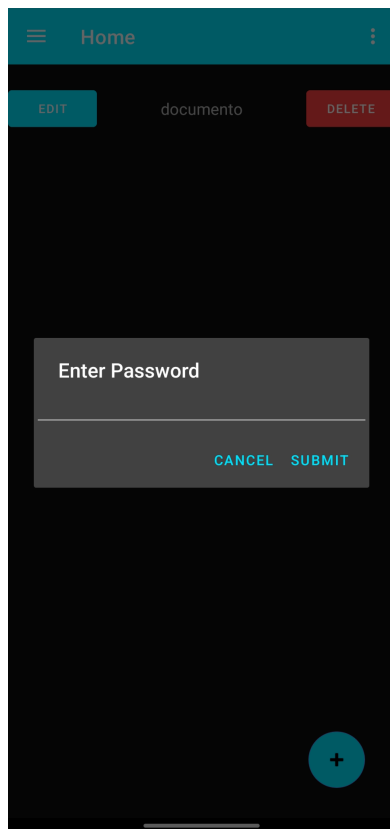
Al pulsar *CREATE* se guarda un documento con datos de ejemplo. Y se refresca la pantalla principal para mostrar el documento nuevo.



Proceso de edición de un documento (1).

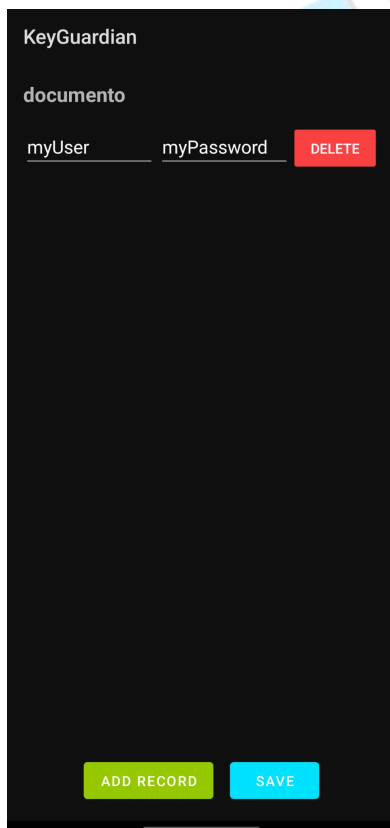
Pantalla principal en la que se muestra el documento creado anteriormente.

(Ilustración 25) Pantalla principal



Proceso de edición de un documento (2).

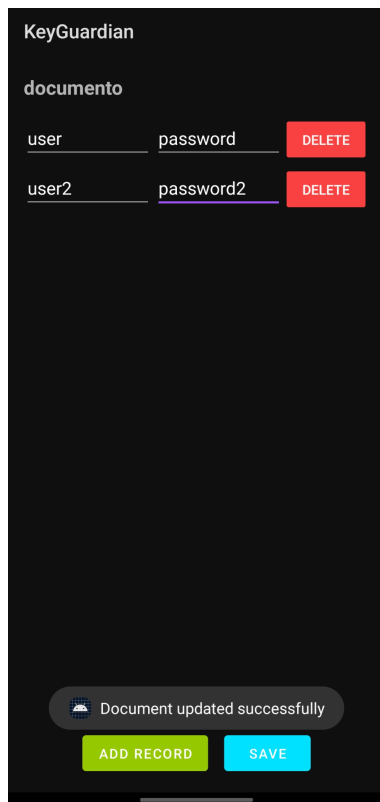
Al pulsar el botón de *EDIT* de cualquier documento de la pantalla principal se despliega un *pop-up* indicando al usuario que introduzca la contraseña que eligió durante la creación del documento para descifrar su contenido.



Proceso de edición de un documento (2).

Al introducir una contraseña correcta al editar el documento, se muestran los datos que este contiene.

En la imagen se muestran los datos de ejemplo con los que se crea el documento.



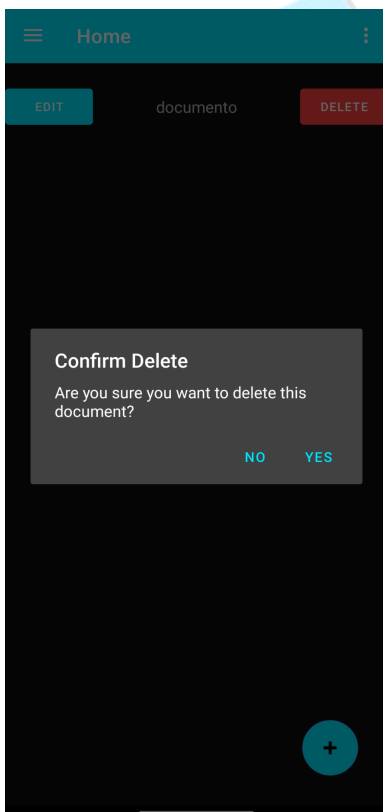
(Ilustración 26) Pantalla de edición

Proceso de edición de un documento (3).

El botón de *ADD RECORD* permite añadir una nueva pareja clave-valor. El texto de todos los elementos es editable.

Al pulsar el botón *DELETE* se puede borrar un registro.

Al pulsar el botón *SAVE* se cifran los datos del documento con la contraseña utilizada para descifrarlo y se guardan los datos Firestore.



Proceso de borrado de un documento.

En la pantalla principal al pulsar el botón *DELETE* en cualquier documento, se muestra un pop-up de confirmación. Al confirmar se realiza un borrado lógico del documento en Firestore.

TRATAMIENTO DE DATOS

Tal y como se ha expuesto en apartados anteriores, el proyecto está minimiza el tratamiento de datos en el backend como medida para maximizar la seguridad y privacidad de los datos de los usuarios.

Este enfoque transfiere todas las operaciones que se realizan sobre los datos a la aplicación (frontend) que se ejecuta de forma local en los dispositivos de los usuarios. Además de la capacidad de crear, editar y eliminar documentos; es decir, acciones imprescindibles para el correcto funcionamiento y usabilidad de la herramienta, la aplicación realiza un cifrado de los documentos antes de enviarlos al backend de forma que en ningún momento se almacenan datos de los usuarios en claro, un cifrado *end-to-end*.

El cifrado que se utiliza es AES (Advanced Encryption Standard) en modo CBC (Cipher Block Chaining) con un relleno PKCS5Padding.

AES es un algoritmo de criptografía simétrica en bloques. En modo Cipher Block Chaining se cifran los datos en bloques de 128 bits y se utiliza un vector de inicialización para encadenar los bloques de cifrado. En este modo el primer bloque se combina con un vector aleatorio y los siguientes se combinan con el bloque anterior, de esta forma se introduce aleatoriedad en el cifrado, produciendo resultados diferentes con vectores iniciales diferentes.

Cifrado

Dado que el input que se requiere del usuario para realizar el cifrado es una contraseña, se emplea PBKDF2 para producir una clave compatible con AES a partir de esa contraseña.

PBKDF2 es un algoritmo de generación de claves que utiliza un proceso iterativo para transformar una contraseña en una clave criptográfica segura y resistente a ataques por fuerza bruta. El tamaño de la salida es configurable y se emplea un *salt* (un valor fijo) para realizar las iteraciones y producir la clave, en este caso se usa un tamaño de 256 bits ya que el cifrado emplea AES-256.

Al generar la clave a partir de la contraseña se crea un vector aleatorio y se cifran los datos en modo CBC añadiendo relleno si es necesario.

Este proceso se repite para cada elemento en el documento, manteniendo el formato.

Ejemplo de cifrado:

Documento antes de ser cifrado, representado en formato JSON.

```
{  
  "user": "password",  
  "user2": "password2"  
}
```

Al aplicar el cifrado sobre el documento anterior se cifran por separado y usando la misma contraseña cada clave y cada valor; es decir, “user”, “password”, “user2” y “password2” se cifran por separado. Una vez cifrados se vuelve a construir el JSON con los valores nuevos, manteniendo el formato.

```
{  
  "4U2tI0ZCrfwyTWnqaOdnR3rqGZcrgesu+MlFq1+DY0c=": "Ni9NTMmFq16if026BV5acMINrVrzdndZYtqSL43ICKg=",  
  "eb6ZCQgGuYMi1n6zkQ8nI2Fto=": "CaF2isSiNLg4oKMC9G4z1ZTtYQVb1RhPuJpTNpGXMHM="  
}
```

El JSON cifrado resultante es el que se envía al backend y se almacena en Firebase.

Descifrado

El proceso de descifrado es el contrario al proceso de cifrado.

Se introduce una contraseña y se genera la clave AES-256 a partir de esa contraseña, utilizando de nuevo el algoritmo PBKDF2.

Una vez generada la clave se emplea CBC en modo decrypt y se obtiene el texto en claro.

Este proceso se realiza para cada uno de los componentes del JSON, puesto que hemos mantenido el formato.

SEGURIDAD Y PRIVACIDAD EN LA APLICACIÓN

La seguridad y la privacidad de los datos de los usuarios han sido un pilar fundamental desde la concepción del proyecto hasta el final de su desarrollo. En este apartado se van a destacar las principales medidas implementadas y principios seguidos para lograr este objetivo.

Además de la seguridad que proporciona Firebase Authentication para el registro y el inicio de sesión de los usuarios, se han seguido los principios de seguridad *end-to-end*, es decir, asegurando que los datos de los usuarios estén protegidos en todo momento. Para ello se han tomado las siguientes decisiones de diseño:

- Cifrado en el cliente: Todos los datos producidos por el usuario se cifran antes de salir del dispositivo.
- Contraseñas creadas por el usuario: Las claves de cifrado de los datos de los usuarios están definidas por ellos mismos y nunca se almacenan en los servidores ni en local de forma permanente.
- El backend nunca recibe información en texto plano, minimizando la exposición en caso de vulneraciones.

El algoritmo de cifrado utilizado ha sido cuidadosamente seleccionado para ser compatible con los requisitos funcionales y no funcionales, manteniendo un buen balance entre rendimiento y seguridad. A su vez, implementando algoritmos como PBKDF2 para aumentar la resiliencia contra ataques de fuerza bruta.

También se han seguido buenas prácticas de seguridad en la comunicación, haciendo uso de HTTPS en las interacciones cliente-servidor.

IMPACTO

El proyecto tiene como objetivo facilitar la autogestión de claves personales sin exponer información sensible en la nube. Para alcanzarlo, ha sido desarrollado sobre dos pilares fundamentales, la seguridad y la privacidad.

SEGURIDAD

La seguridad digital es más importante que nunca y a medida que transcurre el tiempo cada vez más actividades se realizan de forma online, contribuyendo al crecimiento de nuestra vida digital, aumentando de esta forma la importancia de la seguridad en este ámbito.

Cuando un usuario se registra en un servicio online siempre corre el riesgo de ser víctima de una violación de datos personales, durante la cual puede quedar expuesta la contraseña que se utilizó durante el registro en ese servicio. Por eso mismo, es importante, además de usar contraseñas seguras, no utilizar la misma para todos los servicios a los que se registra. Puesto que si uno de estos servicios se ve comprometido, estarían comprometidos todos los demás en los cuales se hayan utilizado las mismas credenciales.

Uno de los objetivos de este proyecto, posiblemente el más importante, es contribuir a la seguridad digital, aportando una base escalable, fácil de mantener y que cumple con unas funciones básicas a las que cualquier persona debería tener acceso hoy en día.

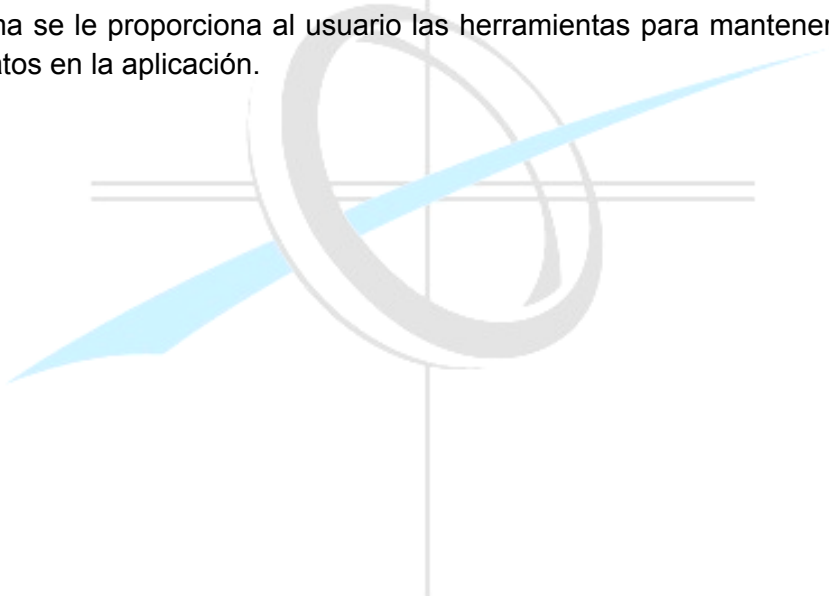
PRIVACIDAD

Además de la seguridad en la red, considero que otro punto igual de importante es la privacidad.

En una violación de datos personales no solo se filtran claves y contraseñas, también se filtran multitud de datos diferentes que se pueden explotar de infinidad de formas por malos actores y otras entidades.

Es por eso que el otro pilar sobre el que se ha desarrollado el proyecto ha sido la privacidad de los datos de los usuarios. El servicio cifra los datos localmente, asegurando que ni siquiera el backend pueda acceder a ellos. De esta forma se protege a los usuarios de un posible brecha de seguridad en el servidor, minimizando o eliminando el riesgo de un servidor comprometido, en el peor caso.

De esta forma se le proporciona al usuario las herramientas para mantener un control total sobre sus datos en la aplicación.



OTRAS APLICACIONES

KeyGuardian no solo es un gestor de credenciales seguro, sino que también se basa en dos componentes fundamentales que pueden aplicarse en otros contextos tecnológicos.

Frontal con Cifrado Robusto

El sistema de cifrado de KeyGuardian es una de sus principales fortalezas. Se ha diseñado para proporcionar un método seguro y eficiente de protección de datos sensibles mediante cifrado de extremo a extremo. Este mecanismo podría aplicarse a otro tipo de documentos digitales o implementarse en un aplicación de mensajería privada y segura.

Backend como Gestor de Documentos JSON

El backend de KeyGuardian actúa como un gestor de almacenamiento para documentos cifrados en formato clave-valor dentro de una base de datos no relacional.

Este diseño, permite que la infraestructura se utilice como una API para la distribución de configuraciones a otros sistemas de software. Por ejemplo, en arquitecturas distribuidas o microservicios, donde las aplicaciones necesitan recuperar configuraciones de forma dinámica y segura, este backend puede proporcionar un punto centralizado para administrar dichos datos.

CONCLUSIÓN

En un mundo cada vez más digitalizado, la privacidad y la seguridad en la red son aspectos fundamentales que no se pueden pasar por alto. El aumento en la cantidad de información personal que circula en línea hacen necesarias herramientas que permitan a los usuarios proteger su información personal.

Uno de los pilares clave de esta protección es la autogestión de claves personales. Depender de terceros para almacenar y administrar credenciales expone a los usuarios a riesgos innecesarios. Asimismo, limitar la exposición de datos personales en la nube es una estrategia esencial para minimizar el riesgo de filtraciones y accesos no autorizados. En este sentido, la combinación de cifrado de extremo a extremo y almacenamiento seguro permite que la información sólo pueda ser accesible por quienes realmente deben tener acceso.

Por último, resulta fundamental proporcionar a los usuarios herramientas que les otorguen el control total sobre sus datos. El objetivo no es solo ofrecer seguridad, sino también empoderar a las personas para que tomen decisiones informadas sobre cómo, dónde y con quién comparten su información.

KeyGuardian pretende dar a los usuarios el control absoluto de su información, garantizando privacidad y seguridad.

BIBLIOGRAFÍA

1. <https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/>
2. <https://ruralhack.es/que-es-gitflow/>
3. <https://aulasoftwarelibre.github.io/taller-de-git/gitflow/#las-ramas-principales>
4. <https://www.excentia.es/que-es-git-flow>
5. <https://desarrollowp.com/blog/tutoriales/aprende-git-de-manera-sencilla-git-flow/>
6. <https://firebase.google.com/docs/projects/learn-more?hl=es&authuser=0>
7. <https://flask.palletsprojects.com/en/stable/>
8. <https://gunicorn.org/>
9. <https://cloud.google.com/why-google-cloud/>
10. <https://www.docker.com/blog/>
11. <https://tailscale.com/kb/1017/install>
12. <https://developer.android.com/studio/intro?hl=es-419>
13. <https://www.pandasecurity.com/es/mediacenter/cifrado-aes-guia/>
14. <https://www.sciencedirect.com/topics/computer-science/cipher-block-chaining>
15. <https://keepcoding.io/blog/que-es-pbkdf2/>