Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Aplicación Web para la Gestión y Monitoreo de la Seguridad de Contraseñas y Correos

Autor: Daniel Eduardo Paz Peña

Tutor: Ricardo Colomo Palacios

Madrid, junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: APLICACIÓN WEB para la GESTIÓN y MONITOREO de la SEGURIDAD de CONTRASEÑAS y CORREOS
Junio 2025

Autor: Daniel Eduardo Paz Peña

Tutor:
Ricardo Colomo Palacios
Lenguajes y sistemas informáticos de ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Grado aborda el desarrollo completo de una aplicación web *full stack* orientada a la gestión y monitoreo de la seguridad de contraseñas y correos electrónicos. El proyecto propone unificar en una sola herramienta diferentes funcionalidades relacionadas con la ciberseguridad con el fin de aumentar la concienciación sobre el peligro de una incorrecta gestión de claves en internet y mejorar los hábitos de seguridad en un contexto digital cada vez más expuesto a riesgos.

La solución desarrollada permite a los usuarios registrarse e iniciar sesión de manera segura, gestionar un almacén de contraseñas cifradas, generar contraseñas robustas, validar direcciones de correo electrónico y comprobar si sus contraseñas han sido expuestas en brechas de datos conocidas. Para ello se ha creado una arquitectura basada en Node.js y Express.js en el backend, con una base de datos MySQL, y un frontend desarrollado con React y Tailwind CSS, priorizando la usabilidad y la experiencia de usuario.

Se ha hecho especial hincapié en el cumplimiento de buenas prácticas en privacidad y protección de datos, mediante el uso de cifrado robusto, con herramientas como bcrypt o algoritmos como AES-256, el uso de tokens JWT con *cookies* seguras y protección frente a ciberataques comunes como XSS, CSRF e inyecciones SQL.

Este proyecto, a nivel práctico ofrece un producto funcional que contribuye a mejorar la concienciación en ciberseguridad y a facilitar una gestión segura de credenciales digitales, además de servir como base para futuros desarrollos en el creciente e importante ámbito de la protección digital.

Abstract

This Final Degree Project presents the complete development of a full stack web application focused on managing and monitoring the security of passwords and email addresses. The project aims to unify various cybersecurity-related functionalities into a single tool, with the objective of raising awareness about the dangers of poor password management on the internet and promoting better security practices in an increasingly risk-exposed digital environment.

The solution allows users to securely register and log in, manage an encrypted password vault, generate strong passwords, validate email addresses, and check whether their passwords have been exposed in known data breaches. To achieve this, an architecture based on Node.js and Express.js has been implemented for the backend, using a MySQL database, and a frontend developed with React and Tailwind CSS, with a focus on usability and user experience.

Special emphasis has been placed on compliance with best practices in privacy and data protection, through the use of strong encryption tools such as bcrypt and algorithms such as AES-256, the use of secure JWT tokens with HttpOnly cookies, and protection against common cyberattacks such as XSS, CSRF, and SQL injection.

At a practical level, this project delivers a functional product that contributes to improving cybersecurity awareness and enabling secure management of digital credentials, while also providing a solid foundation for future developments in the growing and vital field of digital protection.

Tabla de contenidos

1 Introducción

El presente trabajo consiste en el desarrollo de una aplicación web *full stack* orientada a ofrecer en un mismo lugar diferentes herramientas de utilidad relacionadas con la seguridad de credenciales digitales. La aplicación permitirá a los usuarios registrarse, iniciar sesión y acceder a un conjunto de funcionalidades entre las que destacan el almacenamiento seguro de credenciales, la generación de claves robustas seguras, la verificación de posibles filtraciones de contraseñas y la comprobación de la validez de direcciones de correo electrónico.

Este proyecto busca facilitar a los usuarios una herramienta accesible, segura y fácil de utilizar, que promueva buenas prácticas en el manejo de contraseñas y datos personales en internet.

1.1 Motivación

Es bien sabido que nos encontramos en la era digital, donde la seguridad en internet de la información personal y profesional se ha convertido en una prioridad fundamental. Cada día, millones de usuarios se registran en servicios en línea, creando y almacenando contraseñas sin seguir buenas prácticas de seguridad. Esta situación da lugar a vulnerabilidades que abren las puertas a posibles ataques informáticos, como el robo de contraseñas, suplantación de identidad y accesos no autorizados.

Según el informe de ciberseguridad de Verizon (2023) [1], más del 80% de las violaciones de seguridad relacionadas con accesos no autorizados se deben al uso de contraseñas débiles, reutilizadas o robadas. Datos como este evidencian la necesidad de promover herramientas que ayuden a los usuarios a adoptar mejores hábitos de protección de credenciales digitales.

A pesar de la existencia de gestores de contraseñas y herramientas especializadas, muchas de estas soluciones resultan complejas para el usuario promedio o requieren de suscripciones de pago. En este contexto, surge la necesidad de una aplicación web que combine facilidad de uso, utilidad práctica y medidas de seguridad sólidas, orientada a concienciar y ayudar a los usuarios a mejorar sus hábitos de ciberseguridad.

1.2 Objetivos

El propósito principal de este trabajo es desarrollar una aplicación web que permita a los usuarios gestionar sus contraseñas de forma segura, generar claves robustas y verificar tanto la seguridad de las contraseñas como la validez de correos electrónicos promoviendo así mejores prácticas de ciberseguridad. Para ello se establecen los siguientes objetivos específicos:

- Implementar un sistema de registro e inicio de sesión de usuarios con almacenamiento seguro de credenciales en una base de datos.
- Diseñar un almacén de contraseñas cifradas, accesible únicamente por el usuario correspondiente.
- Integrar un generador ajustable de contraseñas robustas y seguras.
- Integrar una funcionalidad de verificación de filtraciones de contraseñas mediante el uso de APIs externas confiables.
- Integrar una herramienta para la comprobación de la validez de direcciones de correo electrónico.

Estos objetivos están alineados con las recomendaciones de la Agencia Española de Protección de Datos (AEPD) sobre contraseñas, patrones y gestores de contraseñas [2], destacando la importancia de herramientas que faciliten la seguridad sin dejar de lado la experiencia del usuario.

2 Tecnologías Empleadas

2.1 Entorno de desarrollo

Una de las herramientas principales utilizadas ha sido Visual Studio Code (VS Code) como editor de código. Esta elección se debe a varias razones. En primer lugar, Visual Studio Code es un editor potente y gratuito que ofrece una gran cantidad de extensiones específicas para entornos JavaScript y *frameworks* como React y Node.js, lo cual es exactamente lo que se necesita para este trabajo. En segundo lugar, se trata de un entorno en el cual ya tengo una experiencia previa, lo cual me ha ahorrado tiempo al no tener que familiarizarme con un entorno nuevo. Además, se trata de uno de los editores de código más conocidos, por lo que está asegurado un soporte continuo, estabilidad y actualizaciones frecuentes.

Estas herramientas se encuentran entre las más utilizadas por desarrolladores a nivel mundial según encuestas como la realizada por Stack Overflow en 2023 [3], donde Visual Studio Code fue seleccionado como el editor de código favorito por más del 70% de los desarrolladores profesionales.

Junto a VS Code, durante el desarrollo del proyecto se ha utilizado GitHub como plataforma de control de versiones y gestión del código. Esta herramienta ha permitido llevar un seguimiento ordenado de la evolución del proyecto, gestionar los diferentes cambios realizados en el código y contar con un historial completo que facilita la trazabilidad de versiones. Además, al estar alojado en la nube, GitHub ha proporcionado la ventaja de tener el proyecto disponible en todo momento y desde cualquier lugar y dispositivo, aparte de garantizar el tener una copia de seguridad constante del mismo.

2.2 Tecnologías del Backend

2.2.1 Node.js

Node.js es un entorno de ejecución de código abierto que permite ejecutar código en el lenguaje de programación JavaScript en el servidor, fuera del navegador, y está diseñado para construir programas de red escalables y de alto rendimiento mediante un modelo asíncrono basado en eventos.

Una de las grandes ventajas de Node.js es su enfoque event-driven, que permite gestionar múltiples peticiones concurrentes sin necesidad de múltiples hilos, lo que reduce la complejidad y el consumo de recursos en servidores web.

Además, Node.js dispone de un sistema de gestión de paquetes por defecto para llamado npm (Node Package Manager), que proporciona acceso a un ecosistema muy amplio de librerías. Es uno de los frameworks principales usados en este proyecto, y que permite instalar, actualizar y administrar bibliotecas y herramientas necesarias para el desarrollo tanto del frontend como del backend de la aplicación. Con npm ha sido posible integrar las tecnologías y librerías

usadas para esta aplicación de forma sencilla, manteniendo además un control preciso de las versiones utilizadas [4].

Existen otras tecnologías backend que fueron consideradas para el desarrollo de esta aplicación web. La más relevante fue Spring Boot, una herramienta que acelera y simplifica el desarrollo de microservicios y aplicaciones web en Java. Esta opción es segura, robusta y escalable, además de usar el lenguaje de programación Java, que es en el que más experiencia y conocimientos tengo.

A pesar de estos aspectos y de que mis conocimientos de JavaScript eran casi nulos, decidí optar por Node.js al ser una opción mucho más ligera, rápida y adecuada para gestionar múltiples peticiones concurrentes, lo que ocurre en aplicaciones web como la que he desarrollado. También tuve en gran consideración el hecho de que usar Node.js facilitaría el desarrollo al permitir usar el mismo lenguaje tanto para el backend como para el frontend, cosa que no ofrece la opción de Spring Boot.

2.2.2 Express.js

Express es un *framework* minimalista para aplicaciones web en Node.js que facilita la creación de APIs [5]. Su simplicidad y flexibilidad permiten definir rutas, middlewares y gestionar peticiones HTTP de forma clara, permitiendo al desarrollador enfocarse directamente en la lógica del proyecto sin configurar arquitecturas complejas. Es una de las tecnologías más usadas en el desarrollo backend con Node.js, por lo que ofrece buena documentación y comunidad activa y, además, tiene una amplia compatibilidad con librerías del ecosistema npm usadas en este proyecto.

Esta herramienta ha sido utilizada para definir los *endpoints* de la API del backend y gestionar su lógica de negocio, lo cual se explicará con detalle más adelante en el capítulo de Desarrollo del proyecto.

2.2.3 Base de datos MySQL

MySQL es un sistema de gestión de bases de datos relacional ampliamente utilizado en el desarrollo web. Está basado en el lenguaje SQL (Structured Query Language) y sigue un modelo de datos estructurado en tablas relacionadas, lo que favorece la integridad, normalización y consistencia de los datos.

En este proyecto, MySQL alberga la base de datos donde se almacena toda la información relacionada con la gestión de usuarios, contraseñas cifradas y tokens de sesión. Su integración con Node.js se realiza mediante la librería mysql2/promise [6], que ofrece una API basada en promesas para interactuar con la base de datos de forma asíncrona y eficiente.

2.2.4 Vitest

Vitest es la herramienta seleccionada para la realización de las pruebas del proyecto, tanto en el frontend como en el backend. Se trata de un *framework* de testing moderno, rápido y compatible con proyectos basados en JavaScript y TypeScript. Existen otras soluciones ampliamente utilizadas, como Jest, Mocha o AVA. Sin embargo, se optó por Vitest por varios motivos clave:

En primer lugar, dado que el frontend del proyecto está construido con React, la opción de Jest es la que viene por defecto al crear el proyecto, sin embargo, Vitest resulta ser una opción más ligera y optimizada en comparación con herramientas como Jest, que en ocasiones requieren configuraciones adicionales para adaptarse a estos entornos [7]. De hecho, se intentó inicialmente realizar pruebas con Jest, pero los diversos problemas con la configuración llevaron a buscar una mejor opción.

Por otro lado, a nivel de backend, Vitest proporciona una API de testing muy similar a la de Jest (con estructuras familiares como describe y it), pero con mejor compatibilidad con los módulos ES, que son utilizados en este proyecto. Además, la posibilidad de cubrir los tests tanto del frontend como del backend con una misma herramienta facilita en gran medida la implementación de las pruebas y el tiempo empleado para ello.

En definitiva, la elección de Vitest se ha basado en su rapidez, simplicidad de integración y configuración en entornos modernos como Node.js, y en la posibilidad de utilizar una única herramienta para cubrir las necesidades de testing en ambos bloques del sistema.

2.2.5 JWT (JSON Web Tokens)

JWT es un estándar que define un método compacto y seguro de transmitir información como un objeto JSON. Su principal uso es en sistemas de autenticación y autorización, como alternativa moderna a las sesiones tradicionales.

Un token de este tipo consta de tres partes codificadas en Base64 [8]:

1. Header: especifica el tipo de token y el algoritmo de firma.
2. Payload: contiene los datos que se quieren transmitir, en el caso de esta aplicación, el ID y el nombre del usuario.
3. Signature: resultado de firmar el contenido anterior con una clave secreta en el servidor.

En este proyecto, JWT se utiliza para gestionar el acceso de los usuarios a zonas protegidas de la aplicación, como el almacén de contraseñas.
Al iniciar sesión correctamente, el backend genera un token que se devuelve al cliente, el cual lo almacena en una *cookie* y lo envía automáticamente en cada petición que haga al servidor.

Los tokens generados en esta aplicación web son *stateless*, lo que se traduce en una autenticación sin estado, es decir, el backend no necesita guardar sesiones activas en la base de datos, lo que reduce carga y complejidad.

Aunque JWT es seguro por diseño, su uso incorrecto puede introducir vulnerabilidades. En este proyecto se han tomado las siguientes precauciones con el fin de garantizar una seguridad robusta:

- El uso de cookies con la bandera *HttpOnly* y *Secure*, lo que evita el acceso al token usando código JavaScript protegiendo así la aplicación de ataques XSS (Cross-Site Scripting).

- El uso de la política SameSite para mitigar riesgos de ataques CSRF (Cross-Site Request Forgery)

- Los tokens de acceso tienen un tiempo de expiración, reduciendo su uso malicioso en caso de que se filtre.

- El uso de tokens de refresco, que permiten obtener un token de acceso nuevo en caso de que haya caducado.

- Usar el algoritmo HMAC SHA-256 (HS256) para firmar los tokens con una clave secreta segura en el backend.

2.2.6 Bcrypt

Bcrypt es una función de *hashing* diseñada específicamente para cifrar contraseñas de forma segura. En lugar de guardar contraseñas en texto plano, lo cual supondría una vulnerabilidad crítica, bcrypt permite almacenar únicamente una versión cifrada de cada contraseña, que es extremadamente difícil de revertir mediante ataques de fuerza bruta o diccionario.

Esta aplicación web utiliza bcrypt en el backend para proteger las contraseñas tanto de los usuarios al momento del registro como las que introduce un usuario en su almacén de contraseñas. Cuando una contraseña es introducida en el sistema, ésta se cifra con bcrypt antes de guardarse en la base de datos. Después del registro, durante el proceso de login, la contraseña ingresada se cifra de la misma manera y se compara con el hash almacenado para comprobar si son la misma contraseña.

2.2.7 CORS (Cross-Origin Resource Sharing)

Por motivos de seguridad, los navegadores bloquean las peticiones HTTP entre diferentes orígenes (dominios, puertos o protocolos). CORS es un mecanismo de seguridad implementado por los navegadores modernos que permite a un servidor indicar cualquier dominio o puerto con un origen distinto del suyo desde el que un navegador debería permitir la carga de recursos [9].

En el caso de este proyecto, el frontend y el backend son dos procesos totalmente independientes, por lo que se ejecutan en puertos distintos. Por tanto, para permitir la comunicación entre estos dos procesos, ha sido necesario habilitar y configurar CORS explícitamente en el servidor backend.

2.2.8 Librerías adicionales

Este proyecto hace uso de diversas librerías auxiliares que aportan funcionalidades concretas y simplifican el desarrollo a pesar de no tener tanto protagonismo como los frameworks principales (Express.js o bcrypt). Estas librerías están disponibles a través de npm y han sido seleccionadas por su compatibilidad con Node.js, popularidad y comunidad activa.

- cookie-parser

  Esta librería se utiliza para analizar las *cookies* enviadas por el navegador en las peticiones HTTP entrantes. Es fundamental para leer los tokens de sesión (JWTs) que han sido almacenados en las cookies con la bandera *HttpOnly*.

- crypto

  Se utiliza para generar identificadores únicos (UUIDs) seguros para identificar a los usuarios y otros recursos del sistema. Ofrece también funciones criptográficas seguras como por ejemplo *hashing*, pero de eso se encarga la librería bcrypt.

- node-fetch

  Permite realizar peticiones HTTP desde el backend. Su uso en este proyecto está orientado a consultar APIs externas, lo que es necesario para dos de las herramientas que ofrece la aplicación.

2.3 Tecnologías del Frontend

2.3.1 React

React es una biblioteca de JavaScript de código abierto diseñada para construir interfaces de usuario de forma eficiente y flexible para facilitar el desarrollo de aplicaciones en una sola página.

Ha sido la tecnología elegida para el desarrollo del frontend de este proyecto debido a su capacidad para construir aplicaciones de una sola página con una arquitectura basada en componentes reutilizables, permitiendo desarrollar una interfaz organizada, mantenible y extensible de forma sencilla. Esto resulta ideal para una aplicación como la que he planteado, que ofrece diferentes herramientas que pueden ser fácilmente asociadas cada una a un componente distinto.

Cabe mencionar también otro punto a favor de la elección de esta herramienta, que es el hecho de que usa JavaScript como lenguaje de programación, el mismo que se usa para el backend.

Existen otras tecnologías que también fueron candidatas para el desarrollo del frontend. La más importante es Angular, que a pesar de ser un *framework* muy completo y estar basado en JavaScript, tiene una curva de aprendizaje más pronunciada que la de React [10]. Otra opción que se tuvo en consideración fue el *framework* Vue.js, el cual resulta sencillo de aprender y de integrar, pero tiene menos popularidad que React, lo que se traduce en menos comunidad y soporte.

2.3.2 Axios

Axios es una librería de JavaScript basada en promesas que permite realizar peticiones HTTP desde el navegador o desde entornos Node.js. En este proyecto, Axios se utiliza en el frontend para comunicarse con el backend desarrollado en Express.js.

La funcionalidad principal de Axios en este caso es facilitar la interacción con la API REST del backend, permitiendo enviar credenciales, registrar usuarios, obtener información protegida (como contraseñas cifradas) o interactuar con servicios externos.

JavaScript incluye una API nativa para realizar peticiones HTTP llamada fetch, sin embargo, Axios ofrece unas ventajas la hacen mejor opción:

- Convierte automáticamente la información que se envía en un JSON, cosa que habría que hacer manualmente si se usa fetch.
- Maneja los errores de forma más intuitiva al separar la respuesta recibida del error.
- Permite enviar las cookies en la petición de forma sencilla con el campo withCredentials.

Estas características hacen de Axios una opción más robusta, especialmente para aplicaciones como esta, que requieren autenticación, manejo de tokens y control detallado del flujo de datos.

2.3.3 Tailwind CSS / CSS personalizado

Tailwind CSS es un *framework* de utilidades para CSS que permite aplicar estilos directamente en los elementos HTML mediante clases predefinidas. Esto facilita la creación rápida de interfaces sin necesidad de escribir grandes cantidades de código CSS tradicional. Esta herramienta ofrece diversas ventajas entre las cuales destacan las siguientes:

- Rapidez de desarrollo: Permite ajustar el diseño de los componentes de manera ágil, ya que la mayoría de los estilos pueden aplicarse directamente en el documento JavaScript mediante clases predefinidas.
- Consistencia visual: Al utilizar un sistema de utilidades, se asegura una coherencia en los márgenes, colores, tipografías y demás propiedades visuales a lo largo de toda la aplicación.
- Personalización sencilla: Tailwind permite modificar su configuración para adaptarse al estilo visual del proyecto, facilitando la integración de la paleta de colores y estilos propios.

Por otro lado, se ha utilizado también CSS personalizado en aquellos casos donde se requería un mayor control sobre estilos específicos o animaciones que no resultaban tan sencillas de implementar únicamente con utilidades de Tailwind. Por ejemplo, se han creado archivos CSS propios para definir los estilos de los formularios de registro e inicio de sesión, y los fondos personalizados.

La elección de esta combinación de herramientas se ha realizado comparándolas con otras alternativas como Bootstrap o el uso exclusivo de CSS tradicional. Frente a estas opciones, Tailwind CSS ofrece mayor flexibilidad y evita la sobrecarga de estilos predefinidos que pueden dificultar la personalización. Además, al no depender de componentes cerrados, se facilita la creación de una interfaz única y adaptada al enfoque del proyecto.

2.4 APIs externas

Esta aplicación se apoya en dos servicios de terceros para implementar la funcionalidad de dos de las herramientas que ofrece esta aplicación sin tener que mantener infraestructuras propias de monitorización de filtraciones ni de validación de correos. A continuación, se describen sus características.

2.4.1 Have I Been Pwned (HIBP)

El objetivo de esta API es comprobar si la contraseña propuesta por el usuario ha aparecido en brechas de datos públicas. Este servicio es gratuito sin la necesidad de autenticarse, es decir, no requiere de una API key para usarse por lo que la única posibilidad de fallo es la pérdida de conectividad externa. Una ventaja clave de este servicio es que sigue un modelo de anonimato con el cual solo necesita el hash de la contraseña para hacer las comprobaciones, garantizando que la contraseña en texto plano nunca salga del servidor de la aplicación. Además, HIBP proporciona una amplia cobertura y actualizaciones continuas.

2.4.2 MailboxLayer

El propósito de este servicio es verificar que la dirección de correo indicada por el usuario es sintácticamente correcta, pertenece a un dominio válido y es probable que acepte un correo. Se trata de una verificación en tres niveles:

1. Sintaxis: filtra posibles errores al teclear el correo como espacios, dobles puntos, etc.
2. Registro MX: comprueba que el dominio realmente tiene servidores de correo declarados en DNS, lo que descarta dominios inventados o caducados.
3. Handshake SMTP: comprueba que al conectarse al servidor de correo éste acepta la dirección concreta, detectando así cuentas inexistentes dentro de dominios reales o que se encuentren en listas negras.

Este servicio requiere de un registro y de una API Key para poder usarse y dispone de varios planes de pago, pero se ha optado por el plan gratuito que permite realizar 100 peticiones gratuitas al mes, que es suficiente para las pruebas. Tiene la ventaja de tener un filtro de e-mails desechables y sugerencias de corrección en caso de que detecte que el usuario quería escribir otra cosa.

3 Análisis de Requisitos

El objetivo de esta sección es describir qué debe hacer la aplicación y qué condiciones debe cumplir para considerarse correcta, segura y viable. Para facilitar la identificación de los requisitos, se etiqueta cada uno con un identificador único (RF-XX para funcionales, RNF-XX para no funcionales).

3.1 Actores y Contexto

Antes de especificar los requisitos, se identifican los actores involucrados y el entorno en el que opera la aplicación. Esto delimita el alcance del sistema y establece los flujos de información sensibles. Los actores que forman parte del sistema son:

Usuario

- Descripción: persona que almacena y gestiona sus contraseñas.
- Interacciones: registro/inicio de sesión, almacenar/eliminar contraseñas de su almacén, uso del generador de contraseñas, del validador de correos y del comprobador de filtraciones.

Administrador

- Descripción: responsable de la operativa y la seguridad del sistema.
- Interacciones: gestión de incidencias y supervisión de los registros.

APIs Externas

- Descripción: servicios externos *Have I Been Pwned* y *MailboxLayer.*
- Interacciones: reciben peticiones del backend para realizar comprobaciones puntuales cuando el usuario lo solicite.

3.2 Requisitos funcionales

RF-01

El sistema debe permitir el registro de nuevos usuarios solicitando nombre, correo electrónico y contraseña.

RF-02

Debe existir en el sistema un proceso de inicio de sesión con verificación de credenciales.

RF-03

Tras iniciar sesión, el usuario podrá ver su lista de entradas guardadas del almacén de contraseñas y podrá añadir y eliminar entradas compuestas por los campos: servicio, nombre de usuario y contraseña.

RF-04

Tras registrarse, el usuario deberá establecer una clave maestra.

RF-05

El usuario podrá revelar y visualizar de forma clara las contraseñas que tiene almacenadas introduciendo correctamente su clave maestra previamente establecida.

RF-06

El sistema ofrecerá un generador de contraseñas que permita configurar la longitud y los tipos de caracteres (mayúsculas, minúsculas, dígitos, símbolos).

RF-07

El sistema debe permitir validar la existencia y validez de un correo electrónico usando la API de MailboxLayer, mostrando información relevante como formato, dominio, disponibilidad, etc.

RF-08

El sistema debe permitir al usuario comprobar si una contraseña ha sido filtrada en brechas de seguridad conocidas, consultando la API de *Have I Been Pwned*.

RF-09

El usuario debe poder cerrar sesión de forma segura.

RF-10

El usuario debe poder consultar sus datos de perfil (nombre de usuario y correo electrónico).

RF-11

El sistema debe gestionar sesiones seguras mediante tokens JWT y cookies.

RF-12

Solo los usuarios autenticados pueden acceder a las funcionalidades protegidas (almacén, generador, validadores, etc.).

3.3 Requisitos no funcionales

Seguridad

RNF-01

Las contraseñas de los usuarios deben almacenarse cifradas (*hash* con bcrypt).

RNF-02

Los tokens de sesión deben gestionarse mediante cookies con el parámetro HTTP-*only* y con expiración.

RNF-03

El sistema debe protegerse frente a ataques comunes como XSS, CSRF y SQL *injection*.

RNF-04

Los datos sensibles no deben exponerse en el frontend ni en las respuestas de la API.

Usabilidad

RNF-05

La interfaz debe ser intuitiva, clara y fácil de usar para cualquier usuario.

RNF-06

El sistema debe proporcionar mensajes de error claros y útiles.

Rendimiento

RNF-07

Las operaciones habituales (*login*, consulta, inserción y borrado de contraseñas) deben ejecutarse en menos de 2 segundos.

RNF-08

El frontend debe ser responsivo y funcionar correctamente en diferentes tamaños de pantalla.

Compatibilidad

RNF-09

La aplicación debe funcionar correctamente en los navegadores web modernos (Chrome, Firefox, Edge).

Mantenibilidad

RNF-10

El código debe estar estructurado y documentado para facilitar su mantenimiento y futuras ampliaciones.

Privacidad

RNF-11

Los datos de los usuarios deben tratarse de acuerdo con la legislación vigente en materia de protección de datos.

RNF-12

No se debe compartir información sensible con terceros sin consentimiento.

Disponibilidad

RNF-13

El sistema debe estar disponible al menos el 95% del tiempo durante su uso previsto.

4 Desarrollo de la Aplicación

4.1 Arquitectura

La Figura 1 muestra la aplicación como un gran contenedor central e identifica todos los elementos humanos y software que se comunican con ella. El rectángulo denominado Aplicación Web, representa el límite del sistema que se desarrolla en este trabajo y todo lo que queda fuera son actores o servicios externos. Conviene aclarar primero que la aplicación se compone de dos servicios independientes:

- **Frontend**

  SPA (Aplicación de página única) desarrollada en React, responsable de la interfaz de usuario.

- **Backend**

  API REST implementada en Node.js que expone los endpoints para autenticación, gestión, llamadas a las APIs externas y acceso a la base de datos.

Aunque en producción ambos servicios se desplegarán en contenedores separados, comunicándose con HTTPS/TLS 1.3, durante el desarrollo los dos corren en la misma máquina y se comunican a través de HTTP sobre localhost (http://localhost:3000 para el backend y http://localhost:3001 para el frontend).

Se detallan a continuación los actores y servicios externos:

- **Usuario**

  Interactúa desde su navegador (SPA en React). El cliente nunca almacena credenciales de sitios remotos, solo envía y recibe datos (algunos cifrados).

- **Base de datos**

  Almacena la información del usuario ya cifrada (en el caso de la contraseña) y los tokens de sesión. Las flechas “Almacena” y “Extrae” indican la lectura y escritura de esta información, nunca en texto plano si se trata de información sensible como las contraseñas.

- **APIs externas**

  Se dividen en Have I Been Pwned (HIBP) y MailboxLayer. Desde la aplicación se realizan peticiones REST a estos servicios:

  - a HIBP únicamente se envía el prefijo SHA-1 de la contraseña.
  - a MailboxLayer se envía la dirección de correo para validar sintaxis, MX y entrega SMTP.
    Cada flecha está etiquetada con el texto “Envía petición” para remarcar que el flujo es unidireccional y las respuestas se tratan internamente.

El borde la caja de la aplicación web indica el perímetro donde se aplican controles de autenticación y autorización. Cualquier comunicación que atraviesa este límite se cifra (HTTPS en producción) y, en el caso de contraseñas, se anonimiza antes de salir.

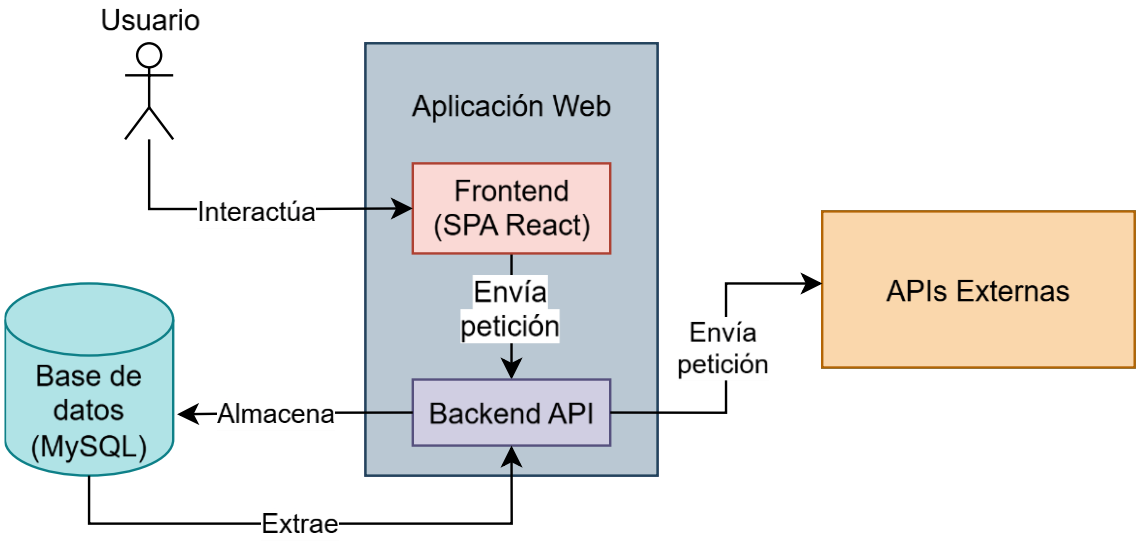

Figura 1 – Diagrama de Contexto

4.2 Diseño

4.2.1 Elementos Internos

En este apartado se identifican los elementos internos de cada componente del sistema y se establecen las interacciones entre ellos con el fin de cumplir con los requisitos de la aplicación. La Figura 2 sirve de mapa entre los componentes que forman parte de la UI (interfaz de usuario) y los servicios REST del backend, además de mostrar los componentes de la base de datos de los servicios externos. A continuación, se detallan los requisitos que se satisfacen con cada interacción entre los elementos de los dos principales bloques del sistema, la API del backend y la página de React:

RF-01: Registro

- Componente frontend: Register
- Endpoint API: /register

El usuario rellenará los campos (nombre, correo electrónico, contraseña y repetir contraseña) del formulario del componente Register. Cuando pulse el botón de registrarse se envía una petición a /register para crear el usuario y almacenarlo en la base de datos.

RF-02: Inicio de sesión

- Componente frontend: Login
- Endpoint API: /login

El usuario introduce su nombre de usuario y contraseña en el formulario del componente Login. Al pulsar "Iniciar sesión" se envía una petición POST a /login, donde el backend verifica las credenciales. Si son correctas, responde con un *access-token* (cookie HttpOnly) y un *refresh-token*.

RF-03: Gestión del almacén de contraseñas

- Componente frontend: PasswordVault
- Endpoints API: /getVaultEntries, /vaultEntry, /deleteVaultEntry

Al iniciarse el componente PasswordVault, se llama a /getVaultEntries para mostrar las entradas ya guardadas del usuario.

El usuario introduce servicio, usuario y contraseña en el formulario y al darle al botón "Guardar", se llama a /vaultEntry para añadir la nueva entrada del almacén en la base de datos.

Al pulsar "Eliminar" en una de las entradas, se llama a /deleteVaultEntry/id para eliminarla de la base de datos además de eliminarla del componente React.

RF-04: Establecer Clave Maestra

- Componente frontend: MasterKeyModal
- Endpoints API: /setMasterKey

Al registrarse un usuario se mostrará un modal, que será una ventana donde se indicará que se debe introducir una clave maestra. Al pulsar el botón de confirmar, se llama a /setMasterKey que guarda dicha clave de forma cifrada en la base de datos.

RF-05: Revelar Contraseñas

- Componente frontend: PasswordVault, MasterKeyModal
- Endpoints API: /vaultEntry/id/getPassword

Cuando un usuario pulse el botón de visualizar contraseña, se mostrará una ventana a través del modal MasterKeyModal, donde se pide introducir la clave para revelar la contraseña seleccionada. Al pulsar el botón de confirmar se llama a /vaultEntry/id/getPassword que se encargará de comprobar que la clave introducida es la correcta para el usuario logeado y devolverá como respuesta la contraseña en texto plano.

RF-06: Generador de contraseñas

- Componente frontend: PasswordGenerator
- Endpoint API: ninguno

El usuario selecciona longitud y los tipos de caracteres que contendrá la contraseña. PasswordGenerator contiene la lógica necesaria para construir una contraseña según los parámetros seleccionados.

RF-07: Validador de correos electrónicos

- Componente frontend: EmailValidator
- Endpoint API: /checkEmail

Cada vez que el usuario escribe un correo, en el input de texto de este componente y pulsa el botón "Comprobar" se envía una petición a /checkEmail. El backend consulta la API de MailboxLayer y devuelve la información relacionada al correo introducido por el usuario. El componente muestra los datos del resultado debajo del input donde se introdujo el correo.

RF-08: Comprobación de contraseñas filtradas

- Componente frontend: LeakDetector
- Endpoint API: /checkPasswordLeak

Cuando el usuario pulsa en "Comprobar", LeakDetector envía la contraseña a /checkPasswordLeak donde se cifra antes de enviarlo a la API externa. El backend reenvía solo los 5 caracteres de prefijo a la API Have I Been Pwned y devuelve la respuesta al componente que muestra un mensaje en pantalla para advertir al usuario de si la clave está comprometida.

RF-09: Cierre de sesión y RF-10: Consulta de perfil

- Componente frontend: UserProfile, Header
- Endpoint API: /logout, /getUserData

Al iniciarse el componente UserProfile, el cual estará contenido en Header, se llama a /getUserData que obtiene y devuelve la información del usuario (Nombre de usuario y correo) de la base de datos. Estos datos se mostrarán en una ventana en el centro de la página donde habrá en la parte inferior un botón de "Cerrar Sesión". Al pulsar este botón se lanza una petición a /logout que da de baja al usuario en la base de datos. Por último, el componente redirige al usuario a la pantalla de *login*.

RF-11: Gestión de sesión y RF-12: Restricción de acceso a funcionalidades protegidas

- Componente frontend: Protected
- Endpoints API: /checkSession, /refreshTokens

El componente Protected se renderiza en el momento en el que el usuario se haya autenticado correctamente y se encargará de renderizar los demás componentes que formen parte de la zona protegida de la web. El endpoint /checkSession comprueba regularmente que el token de acceso siga siendo válido. En caso de que el token haya expirado se llamará a /refreshTokens para intentar renovarlo. En caso de error el usuario será redirigido a la pantalla de *login*. De esta manera se garantiza que solo los usuarios autenticados puedan acceder a las herramientas que ofrece la aplicación.

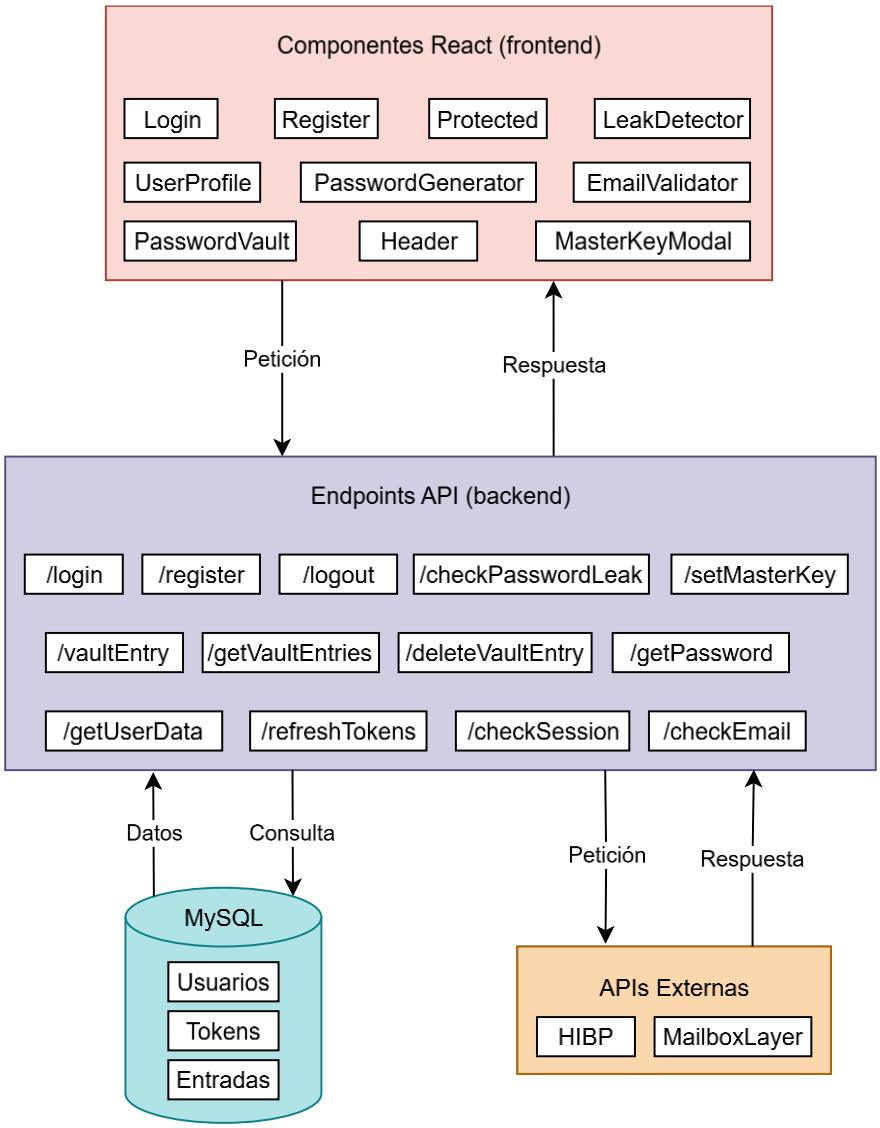

Figura 2 – Elementos internos del sistema

4.2.2 Diseño de la Base de Datos

La Figura 3 recoge la estructura definitiva de la base de datos. Consta únicamente de tres tablas, pero es suficiente para cubrir de buena forma los requisitos de la aplicación descritos en el capítulo 3.

- **User (usuario)**
  Almacena cada usuario registrado con éxito en la aplicación. Se almacena su identificador único, nombre, correo, contraseña cifrada, clave maestra y la fecha de creación.

- **Refresh_token (Token de refresco)**
  Guarda los tokens de refresco activos de los usuarios y permite revocarlos individualmente. Sus campos son un identificador único, el identificador del usuario al que está asociado, el token y la fecha de creación. Cada usuario puede poseer varios tokens de refresco puesto que se permite tener sesiones abiertas en distintos dispositivos.

- **Vault_entry (Entrada del almacén)**
  Almacena los conjuntos de credenciales que forman una entrada en el almacén del usuario. Dispone de un identificador único, el identificador del usuario asociado, nombre del servicio, nombre de usuario, contraseña cifrada y fecha de creación.

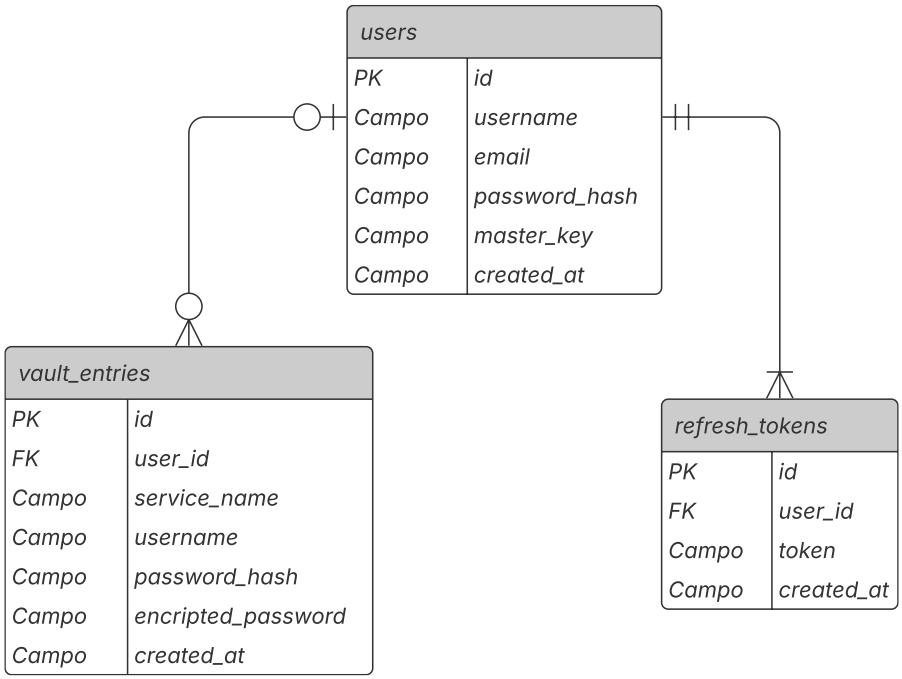

Figura 3 – Diagrama Entidad-Relación

A continuación, se muestra el código SQL para la creación de estas tablas, donde se detalla cómo cada decisión de tipado y restricción se alinea con los requisitos funcionales y con buenas prácticas de seguridad.

```
CREATE TABLE users (
      id BINARY(16) PRIMARY KEY DEFAULT (UUID_TO_BIN(UUID())),
      username VARCHAR(30) NOT NULL UNIQUE,
            email VARCHAR(30) NOT NULL UNIQUE,
      password_hash VARCHAR(255) NOT NULL,
      master_key VARCHAR(255) NOT NULL,
      created_at TIMESTAMP NOT NULL DEFAULT (NOW())
);
```

```
CREATE TABLE refresh_tokens (
    id CHAR(36) PRIMARY KEY,
    user_id BINARY(16) NOT NULL,
    token VARCHAR(255) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE vault_entries (
    id CHAR(36) PRIMARY KEY,
    user_id BINARY(16) NOT NULL,
    service_name VARCHAR(50) NOT NULL,
    username VARCHAR(30) NOT NULL,
    password_hash TEXT NOT NULL,
    encrypted_password TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

En esta aplicación no se permite que los usuarios tengan un mismo nombre de usuario o correo, además de ser datos obligatorios para el registro, por lo que se les ha asignado la opción NOT NULL UNIQUE.

Todos los identificadores son generados en el backend con una función que se encarga de crear UUIDs (Identificadores único de usuario) de forma aleatoria. Se ha optado por esta forma de generar los IDs en vez de usar un valor incremental para evitar la posibilidad de que se pueda hacer un seguimiento de los datos almacenados con intenciones maliciosas desde el frontend.

Se ha usado la opción ON DELETE CASCADE para asegurar que cuando se elimine un usuario, se eliminen a su vez el resto de las entradas asociadas en las demás tablas, garantizando la mínima retención de datos innecesarios.

El uso de BINARY(16) como clave primaria para los usuarios mejora la localización de filas y reduce la profundidad de los índices al ocupar un espacio de 16 Bytes respecto a los 36 de CHAR(36), reduciendo la latencia media de las consultas.

El campo created_at facilita la auditoría de cambios y análisis de crecimiento de usuarios, además de permitir la ordenación cronológica en la UI de las entradas del almacén de contraseñas.

4.2.3 Diseño de la interfaz (UI/UX)

4.2.3.1 Prototipo

Se partió de un prototipo de baja fidelidad en Figma, un software online diseñado para generar prototipos de todo tipo, con el objetivo de tener clara la estructuración de las herramientas que ofrece la aplicación y de la disposición de los elementos en pantalla. El prototipo define únicamente dos de las herramientas de las que dispondrá la aplicación (Figura 4 y Figura 5) pero ha sido suficiente para dejar claro el estilo que tendrá la aplicación.

Las decisiones de diseño para el prototipo han sido tomadas con la intención de crear una interfaz cómoda de usar, en la que resulte sencillo navegar e interactuar. En definitiva, se busca ofrecer las mayores facilidades al usuario durante la utilización de la web.

Figura 4 – Prototipo de Figma del almacén de contraseñas

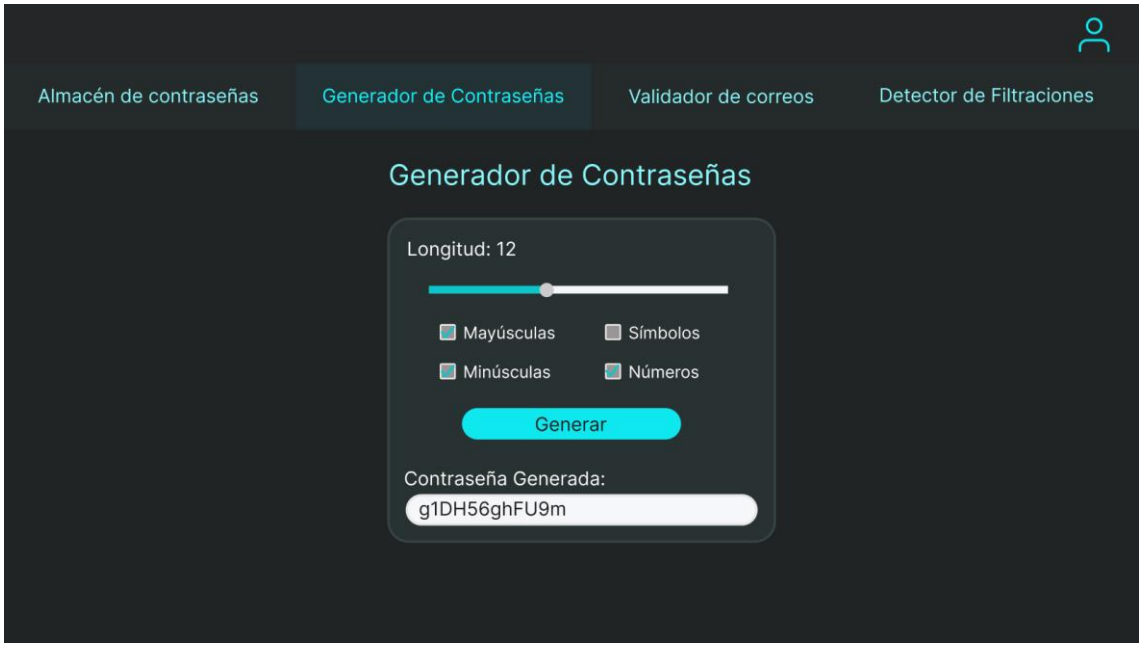

Figura 5 – Prototipo de Figma del generador de contraseñas

4.2.3.2 Diseño Final

El diseño final de la interfaz de usuario (UI) de la aplicación ha sido concebido siguiendo un enfoque centrado en la simplicidad, la claridad visual y la usabilidad, con el objetivo de ofrecer una experiencia de usuario (UX) fluida y accesible para perfiles con conocimientos técnicos limitados. En líneas generales, se ha mantenido el mismo diseño planteado en el prototipo, aunque se ha cambiado la paleta de colores por una que denota algo más de seriedad y modernidad.

La aplicación cuenta con una interfaz de tipo *Single Page Application* (SPA**)**, desarrollada con React, que permite una navegación fluida entre las diferentes secciones sin necesidad de recargar la página. Se ha utilizado el *framework* Tailwind CSS para implementar un diseño moderno y coherente, con una estética visual que intenta transmitir confianza y profesionalidad, lo cual es adecuado para el ámbito de ciberseguridad que maneja el proyecto.

El estilo de colores se basa en tonos oscuros, con un fondo degradado en tonos violeta y negro para las pantallas de inicio de sesión y registro (Figura 6), aporta una estética elegante y además facilita la lectura y reduce la fatiga visual del usuario.

Los elementos interactivos, como botones y campos de entrada (*inputs*), utilizan contrastes de color adecuados: botones en tono morado/azul para acciones "positivas" y en color rojo para acciones "negativas" como el cierre de sesión (Figura 11).

En relación con la experiencia de usuario, cabe mencionar que, al centralizar la gestión de credenciales, el usuario solo necesita recordar una contraseña (la clave maestra) para poder acceder a las contraseñas almacenadas en la aplicación, en lugar de tener que recordar múltiples contraseñas complejas.

Esto mejora enormemente la experiencia de uso, ahorrando tiempo y evitando frustraciones del usuario al iniciar sesión en distintos servicios.

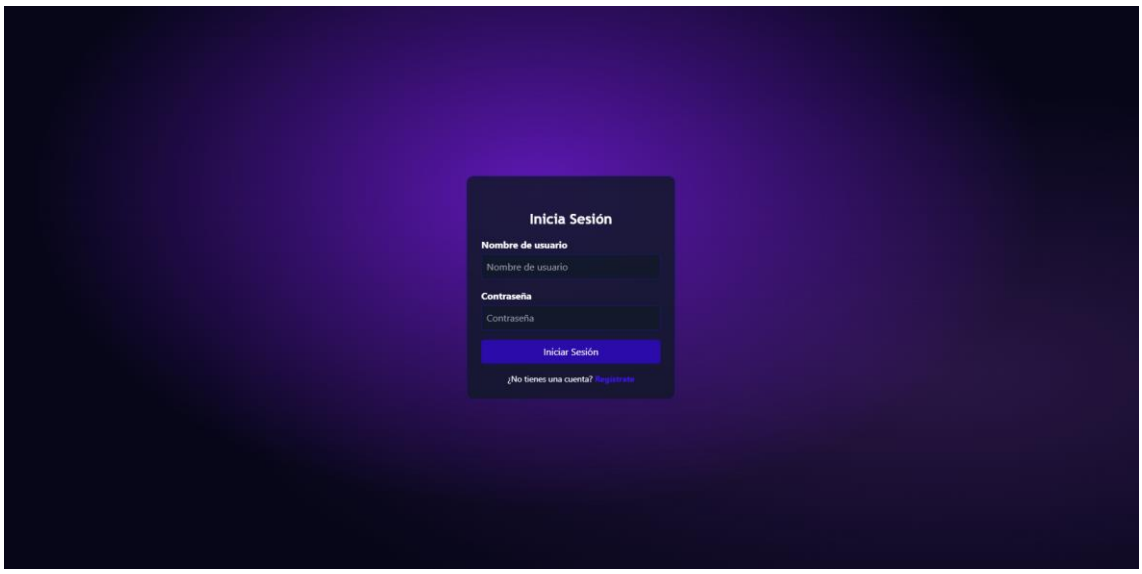

Figura 6 – Pantalla de inicio de sesión

Figura 7 – Pantalla del almacén de contraseñas

Figura 8 – Pantalla del generador de contraseñas

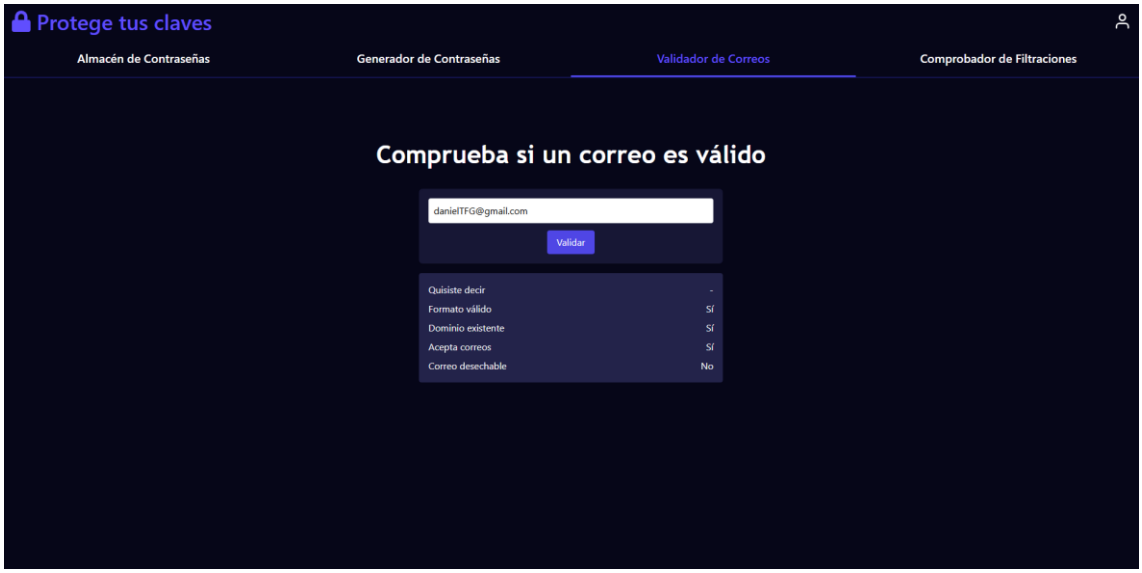

Figura 9 – Pantalla del validador de correos

Figura 10 – Pantalla del comprobador de filtraciones

Figura 11 – Panel con los datos de usuario y cierre de sesión

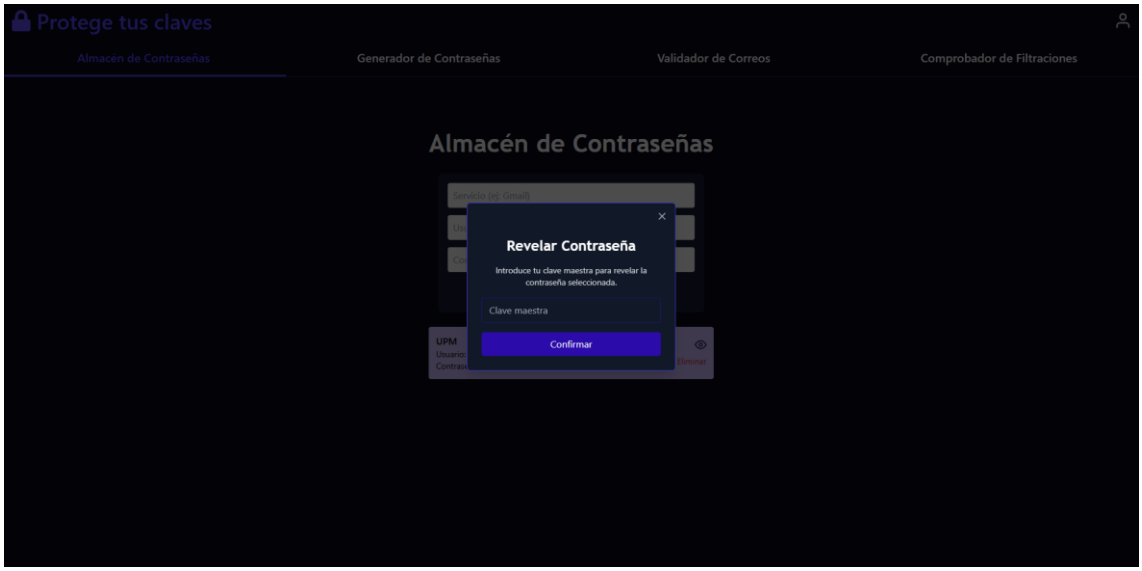

Figura 12 – Ventana para introducir clave maestra

4.3 Implementación

Antes de entrar en detalle en la implementación de cada funcionalidad, conviene explicar cómo se ha manejado en la comunicación entre el backend y la base de datos.

Algunos endpoints de la API del backend delegan su interacción con la base de datos a uno de los siguientes repositorios: user-repository, vault-repository y refToken-repository. Estos repositorios se encargan de establecer la conexión con MySQL, cifrar y validar datos, y realizar las peticiones correspondientes. Hay un repositorio por cada recurso del sistema que se almacena en la base de datos (usuario, token, entrada del almacén de contraseñas). Todas las consultas realizadas a la base de datos se hacen de la forma correcta para evitar ataques de inyección SQL, cubriendo el requisito RNF-03.

Conviene mencionar que, a nivel de código, las variables sensibles que pueden ser accedidas por múltiples partes del proyecto del backend (puerto, API_Key, secreto para cifrar, etc), se almacenan como variables de entorno en un archivo oculto (.env). Esta es una buena práctica orientada a mantener la seguridad de esta información en el caso de que la aplicación se lance a producción.

4.3.1 Registro e Inicio de Sesión

La funcionalidad de alta de usuarios, autenticación y gestión de sesión se ha dividido en dos servicios independientes, el frontend (página de React) y el backend (API de Express), que se ejecutan en contextos distintos sobre localhost durante la fase de desarrollo. En este apartado se detalla cómo cooperan para cumplir los requisitos funcionales RF-01 y RF-02, y los no funcionales de seguridad RNF-01 y RNF-02.

Registro

El componente React para el registro valida que las contraseñas introducidas coincidan y que se hayan rellenado todos los campos del formulario antes de invocar al endpoint /register. La llamada al endpoint se hace a través de Axios.

Una vez hecha la llamada a /register, el backend llama al repositorio de usuarios (user-repository) y hace unas comprobaciones para asegurar que se cumplan las siguientes condiciones:

- El nombre de usuario debe ser de al menos 4 caracteres de longitud.
- La contraseña debe ser de al menos 6 caracteres de longitud.
- El usuario introducido no debe existir en la base de datos. (Figura 13)

Si todo se verifica correctamente, se genera el hash de la contraseña usando bcrypt, cumpliendo con lo que exige el requisito RNF-02, y el identificador único de usuario (UUID). Estos datos junto al correo se introducen en la base de datos, finalizando así la creación del usuario. El frontend, al recibir una respuesta exitosa del endpoint /register, redirige al usuario a la pantalla de login para que inicie sesión.

Login

El login, por su parte, envía las credenciales al /login que, de la misma forma que /register, llama al repositorio de usuarios para hace unas verificaciones iniciales:

- El usuario debe existir en la base de datos.
- La contraseña introducida debe ser la correcta.

Para comprobar que la contraseña es correcta, se obtiene su hash de la misma forma que en el registro y se compara con el hash guardado en la base de datos. Una vez hechas las comprobaciones en el repositorio, /login procede con la creación de los tokens de acceso y refresco, que se les asigna una expiración de 1 hora y de 7 días respectivamente. Después, se llama al repositorio de tokens de refresco (refTokens-repository) que se encarga de introducir el token en la base de datos tras previamente haber comprobado que no existía. El backend finaliza añadiendo en la respuesta las cookies generadas.

Si el login se ha realizado correctamente, se redirige al usuario a la página /tools donde se encuentran todas las demás funcionalidades que ofrece la aplicación.

```
// 2. Comprobar que usuario no existe
const [users] = await connection.query(
  'SELECT id FROM users WHERE username = ?', [username]
)
if (users[0]) throw new Error('USER_EXISTS')
```

Figura 13 – Consulta SQL para comprobar existencia de usuario

4.3.2 Almacén de Contraseñas

El Almacén de Contraseñas es el primer componente que se muestra en pantalla una vez el usuario es autenticado y redirigido a la página /tools. Implementa los requisitos RF-03 (creación, visualización y eliminación de entradas), RF-04 (Establecer clave maestra) y RF-05 (Revelación de contraseñas). A continuación se explica el funcionamiento en detalle de cada uno de los procesos que son involucrados en esta funcionalidad de la aplicación:

Establecimiento de la clave maestra

Si es la primera vez que el usuario entra a la página, es decir, se acaba de registrar, se mostrará una ventana por encima del almacén y en medio de la pantalla donde se le indica al usuario que debe ingresar una clave maestra para poder revelar las contraseñas de su almacén, las cuales se muestran por defecto censuradas. Una vez introducida una clave, se llama al endpoint del backend /setMasterKey que delega al user-repository la generación del hash de la clave y la inserción a la base de datos. Si el frontend recibe una respuesta de éxito, elimina de la pantalla la ventana, mostrando el almacén de contraseñas en su totalidad.

Visualización de la lista de contraseñas

Si el usuario ya tiene guardadas contraseñas al momento de entrar a /tools, se lanza una petición a /getVaultEntries, que a su vez llama al repositorio de entradas del almacén (vault-repository) enviándole el id del usuario. El repositorio comprueba que el usuario existe antes de realizar la consulta sql para obtener las entradas. El endpoint devuelve estas entradas al frontend para mostrarlas en la pantalla.

Creación de entrada

Por encima de la lista de entradas guardadas, habrá un pequeño formulario donde se podrá rellenar la información para añadir una nueva. El componente se encarga de comprobar que se rellenen todos los campos del formulario antes de lanzar la petición al backend.

Al pulsar el botón “Guardar”, se llama a /vaultEntry, que mandará a vault-repository a insertar en la base de datos la entrada. Antes de introducir la entrada en la base de datos, se comprueba que el usuario que realiza la petición existe y se encripta la contraseña introducida. Tiene gran importancia mencionar el hecho de que a esta contraseña no se le genera un hash como se ha hecho hasta el momento, sino que se encripta usando un mecanismo de encriptación segura (AES 256-CBC). Esto es necesario debido a que para poder mostrar posteriormente al usuario la contraseña de forma clara se necesitará poder obtener la contraseña descifrada en texto plano, cosa que no se puede obtener a partir de un hash de bcrypt. Para poder realizar esto, el repositorio cuenta con una función de encriptación y desencriptación que implementa un mecanismo de robusta seguridad para evitar todo lo posible ataques de fuerza bruta.

Una vez añadida la entrada a la base de datos, el componente del frontend actualiza la vista para mostrar la nueva entrada en la lista.

Revelar contraseña

Cada entrada de la lista cuenta con un botón para revelar la contraseña. Al pulsar este botón, se renderiza el mismo componente que se usa para cuando un usuario nuevo debe establecer una clave maestra, el cual consiste en un recuadro o ventana central con un texto y un input para escribir texto y un botón para confirmar. En este caso, este componente pide al usuario introducir su clave maestra para revelar la contraseña solicitada.

Una vez que el usuario la introduzca y pulse el botón, se llama a /vaultEntry/${entryId}/getPassword que solicita la comprobación de que la clave introducida es la correcta a user-repository. La comprobación de las claves se hace de la misma manera que las contraseñas en el proceso de *login*, generando el hash de la introducida y comparándola con la de la base de datos.

Después de validar la clave, el *endpoint* solicita al repositorio de entradas la contraseña desencriptada. Para ello el repositorio realiza lo siguiente:

1. Comprobar que el usuario existe.
2. Obtener las entradas del usuario.
3. Desencriptar la contraseña.

Después de estos pasos, el frontend recibe como respuesta la contraseña en texto plano y la muestra durante 5 segundos (Figura 14), después se vuelve a ocultar ya que por motivos de seguridad conviene evitar la posibilidad de mostrar la contraseña de forma clara en pantalla de forma indefinida.

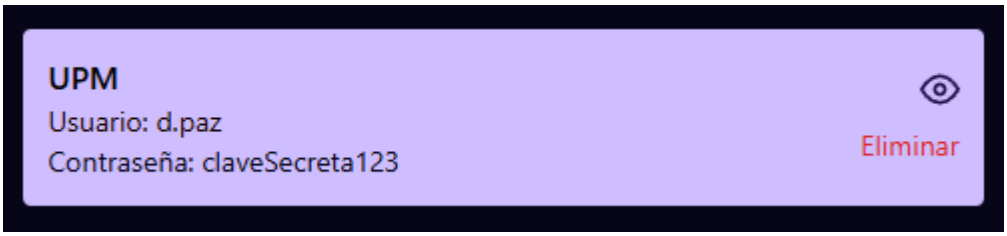

Figura 14 – Entrada con la contraseña revelada

Eliminar entrada

La otra funcionalidad que implementa el almacén es la de eliminar entradas. Para conseguirlo, las entradas cuentan con un botón debajo del de revelar la contraseña que al ser pulsado muestra un recuadro de confirmación que tiene dos botones, uno para confirmar y otro para cancelar. Esto se ha implementado para evitar eliminaciones accidentales. Una vez que el usuario confirma, se envía una petición a /vaultEntry/${id} que delega la eliminación a vault-repository. El componente del frontend, al recibir la respuesta del backend quita de la lista la entrada eliminada.

4.3.3 Generador de Contraseñas

La segunda herramienta de la aplicación es un generador de contraseñas robustas pensado para, además de cumplir con el requisito RF-06, cubrir dos necesidades: facilitar al usuario claves aleatorias resistentes ante ataques de fuerza bruta y garantizar que esas claves se crean enteramente en el cliente, sin que el servidor ni entidades externas las conozcan jamás. Para ello, la lógica reside completamente en el componente React, donde no hay ni peticiones HTTP ni persistencia de las contraseñas generadas.

El componente muestra cuatro casillas de verificación para cada tipo de carácter que el usuario decida incluir en la generación (mayúsculas, minúsculas, números y símbolos) y un *slider* (control deslizante) que permite seleccionar la longitud entre 4 y 50 caracteres. El botón "Generar" invoca la función que aplica la lógica y muestra la clave en un *input* de solo lectura. En la parte derecha de este *input*, hay un botón a modo de icono que permite copiar la contraseña generada en el portapapeles, mostrando un texto "¡Copiado!" (Figura 15) durante 1,5 segundos para indicar al usuario que se ha copiado correctamente.

La lógica está contenida en una función llamada generatePassword. En lugar de apoyarse en la clásica función para generar valores aleatorios, Math.random, se declara un *buffer* que se rellena con valores obtenidos por window.crypto.getRandomValues, un generador de números pseudoaleatorios criptográficamente seguro (CSPRNG) que tienen todos los navegadores modernos. Esta estrategia es mucho más segura y fiable para una generación aleatoria como la que se busca para esta herramienta. Cada valor del *buffer* se mapea al conjunto de caracteres construido en tiempo real según las opciones marcadas en las casillas por el usuario.

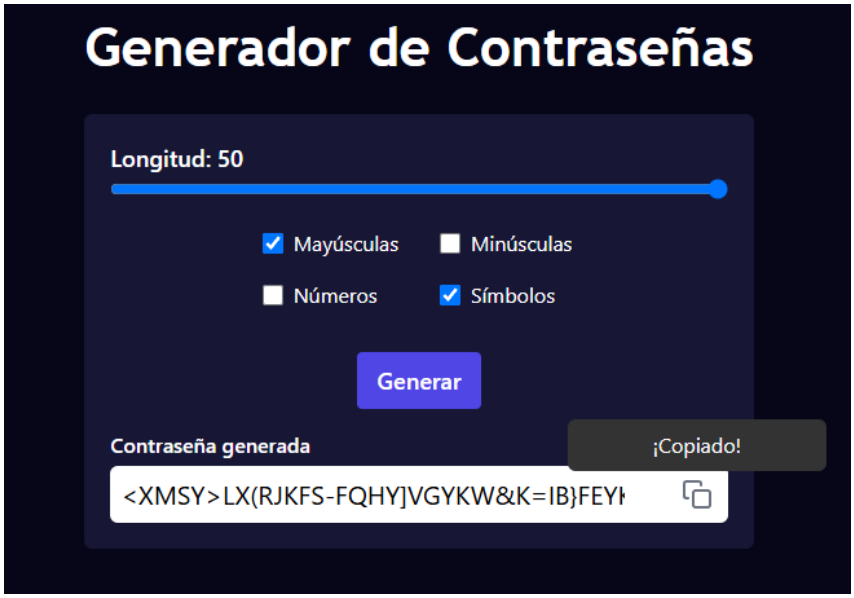

Figura 15 – Contraseña copiada al portapapeles en el generador

4.3.4 Validador de Correos

La herramienta para validar correos está formada por un componente React que consiste en un formulario con tan solo un *input*, donde el usuario debe introducir el correo, y un botón para proceder con la validación. Cuando el botón es pulsado, se llama al *endpoint* /checkEmail, el cual realiza una llamada a la API externa de MailboxLayer. El *endpoint* para este servicio es el siguiente:

https://apilayer.net/api/check?access_key=${API_KEY}&smtp=1&format=1&email=${encodeURIComponent(email)}

La API_KEY es la llave obtenida al registrarse en la página del servicio, la cual se guarda como variable de entorno.

La respuesta dada por el servicio es una estructura extensa, pero para esta aplicación solo se consideran los siguientes valores:

- did_you_mean: sugerencia ortográfica en caso de que detecte que el usuario pretendía escribir otra cosa. Por ejemplo, si se introduce "usuario@mgail.com", este campo mostrará "usuario@gmail.com" (Figura 16).

- format_valid: si la sintaxis es correcta.

- mx_found: si el dominio es válido y existe.

- smtp_check: si la dirección puede recibir correos.

- disposable: si se trata de un correo desechable o temporal.

El frontend recibe estos valores como respuesta del backend y los muestra debajo del formulario. El requisito implementado con esta herramienta es el RF-07 (Validar correos).

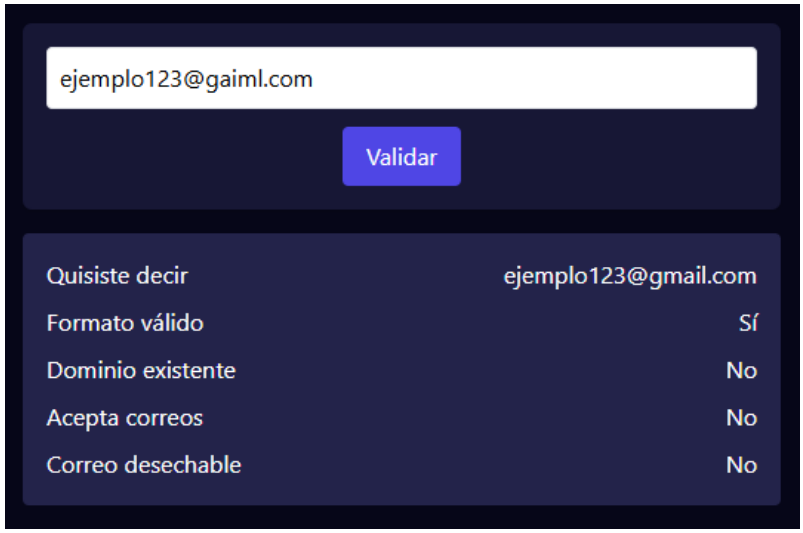

Figura 16 – Validación de correo mal escrito

4.3.5 Comprobador de Filtraciones

La herramienta para comprobar filtraciones de claves permite al usuario contrastar cualquier clave frente a la base de datos *Pwned Passwords* del servicio *Have I Been Pwned* (HIBP). Con esta funcionalidad la aplicación cumple con el requisito RF-08.

De la misma forma que el validador de correos, el componente de esta herramienta está formado por un pequeño formulario que contiene un *input* para introducir la clave y un botón para proceder con la comprobación. Una función añadida a esta herramienta es la de alternar entre ocultar y mostrar la contraseña introducida a través de un botón en forma de icono a la derecha del *input*. Cuando el usuario pulsa el botón, se llama a /checkPasswordLeak donde se envía una petición a la API de HIBP:

https://api.pwnedpasswords.com/range/{hashPrefix}

El parámetro hashPrefix corresponde a los 5 primeros caracteres del *hash* SHA-1 de la contraseña introducida. Este hash se obtiene usando la librería nativa de Node.js llamada crypto, la cual funciona de diferente forma que bcrypt, que es la que se usa para las contraseñas de los usuarios. Para este proceso se ha implementado una función fuera del *endpoint* que se encarga de generar el hash, hacer la petición y comprobar en base a la respuesta si la clave ha sido filtrada. El *endpoint* recibe una respuesta de esta función y la envía como respuesta al frontend. En el frontend se muestra un texto indicando si ha sido filtrada o no (Figura 17).

Figura 17 – Comprobación de contraseña robusta

4.3.6 Perfil de usuario y Cierre de sesión

La aplicación mantiene un perfil de usuario con información reducida pero esencial: nombre de usuario y dirección de correo, junto con la capacidad de cerrar la sesión de forma inmediata. Con esto se asegura cumplir los requisitos funcionales RF-09 y RF-10. Su funcionamiento involucra dos componentes de interfaz, el encabezado y un modal de perfil, además de un único *endpoint* del backend.

El componente de la cabecera se muestra en todas las herramientas de la página protegida /tools, y muestra el logotipo de la aplicación y un icono de usuario. Al pulsar sobre este icono se muestra el modal de perfil de usuario, que se trata de un panelñ central que se superpone a lo que se veía en la pantalla, con un fondo semitransparente que oscurece todo lo que no sea el panel, dirigiendo la atención del usuario a este elemento. Entro del panel se muestra el nombre del usuario, su correo electrónico y un botón con el texto “Cerrar sesión” junto a un icono que representa la acción (Figura 11).

Cuando el usuario pulsa el botón, se lanza una petición a /logout que extrae el *refresh token* de la cookie e invoca una función del repositorio de tokens para invalidar el que está en la base de datos. Después, el *endpoint* indica en la respuesta que se vacíen tanto el token de acceso como el de refresco, dejando el navegador limpio de credenciales.

4.3.7 Control de Sesiones

La aplicación adopta el patrón: *access-token* de corta duración y *refresh-token* de larga duración, almacenados ambos como *cookies httpOnly*, para equilibrar seguridad, usabilidad y trazabilidad. De esta forma se satisfacen los últimos dos requisitos de la aplicación: RF-11 y RF-12.

A continuación, se describe cómo estos dos tokens se manejan y controlan durante el funcionamiento de la aplicación y qué controles adicionales (enrutadores de React, interceptores Axios) garantizan su correcta renovación y revocación.

Los tokens JWT (Json Web Tokens) se generan al iniciar sesión de forma correcta, los de acceso con una duración de 1 hora y los de refresco de 7 días, y ambos con los campos *httpOnly y* sameSite=strict, para garantizar seguridad. Después, se envían en cookies junto a la respuesta al frontend, así el navegador del usuario las almacena y las envía en futuras peticiones en caso de que se requieran.

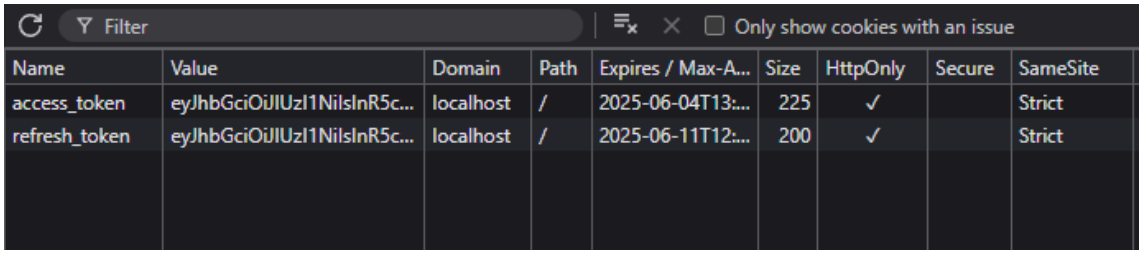

Figura 18 – Cookies HttpOnly almacenados en el navegador

Control de rutas en el cliente

Para evitar posibles problemas de acceso a rutas, se han implementado dos componentes de React que funcionan como un "envoltorio" para las rutas privadas (en el caso de esta aplicación solo hay una) y las públicas que actúan siempre antes de que se renderice una página:

- PrivateRoute: protege la pantalla protegida (/tools). Al montarse, llama a /checkSession en el backend y comprueba los tokens del usuario en la *cookie*. Si son válidos, se procede a renderizar el componente que se estaba cargando (/tools en este caso). Si el token de acceso es inválido, se intenta renovar de forma automática, y si algo falla, se redirige al usuario a la página de *login* (/login-form).

- PublicRoute: hace lo opuesto, bloquea el acceso a las rutas públicas, es decir, a los formularios de *login* y registro en caso de que el usuario ya esté autenticado. Si el usuario autenticado intenta acceder a estas páginas será redirigido a /tools.

Este doble filtro impide tanto el acceso anónimo a zonas protegidas como la visualización redundante de formularios de *login* cuando la sesión ya está activa.

```
return (
    <Routes>
      <Route path="/" element={<Navigate to="/login-form" />} />
      <Route path="/login-form" element={<PublicRoute><Login /></PublicRoute>} />
      <Route path="/register-form" element={<PublicRoute><Register /></PublicRoute>} />
      <Route path="/tools" element={<PrivateRoute><Protected /></PrivateRoute>} />
    </Routes>
)
```

Figura 19 – Páginas envueltas por los controles de rutas en App.js

Intercepción centralizada de errores de tokens

Mientras el usuario se mantiene en la misma página restringida, puede interactuar con ella realizando peticiones al backend sin que se recargue la página, dejando al "envoltorio" que la protege inutilizado. Para cubrir la validación de los tokens en estos casos se ha implementado un interceptor de peticiones a través de Axios. Con este interceptor, cada petición realizada pasará por un filtro donde se comprobará la validez de los tokens y se intentará una renovación automática en caso de ser posible, de la misma forma que se hacía en los componentes "envoltorio" para las rutas.

A modo de conclusión, el manejo de sesiones de esta aplicación combina controles en el servidor y cliente para ofrecer una experiencia fluida sin sacrificar principios de seguridad ni requisitos normativos. La estrategia garantiza que los tokens estén siempre al día y que la revocación sea inmediata.

4.3.8 Manejo de Errores

En esta aplicación el tratamiento de errores se apoya en 2 pilares coordinados:

Convención de códigos en el backend

Todas las rutas definidas en la API del backend devuelven una respuesta en formato JSON con dos campos:

- errorCode: etiqueta breve y descriptiva en mayúsculas.

- message: mensaje que describe de forma clara el error. En algunos casos son mensajes orientados al desarrollador y en otros al usuario. Es importante mantener el control de cuales se envían al frontend para evitar mostrar información sensible al usuario.

```
if (error.message === 'USER_NOT_FOUND') {
  return res.status(404).json({ errorCode: 'USER_NOT_FOUND', message: 'El usuario introducido no existe' })
}
else if (error.message === 'INVALID_PASSWORD') {
  return res.status(401).json({ errorCode: 'INVALID_PASSWORD', message: 'Contraseña incorrecta' })
}
return res.status(400).json({ errorCode: 'LOGIN_ERR', message: 'Error al iniciar sesión' })
```

Figura 20 – Detección y envío de errores de /login

Presentación sencilla en la interfaz

Cada componente React mantiene un estado llamado “error” al que se le asigna el mensaje de error correspondiente en base a la etiqueta recibida por el backend. El mensaje de error siempre es breve y de fácil comprensión, y se muestra en la página de forma clara, con un color rojizo que destaca sobre los demás elementos de la página, como se puede apreciar en la Figura 21.

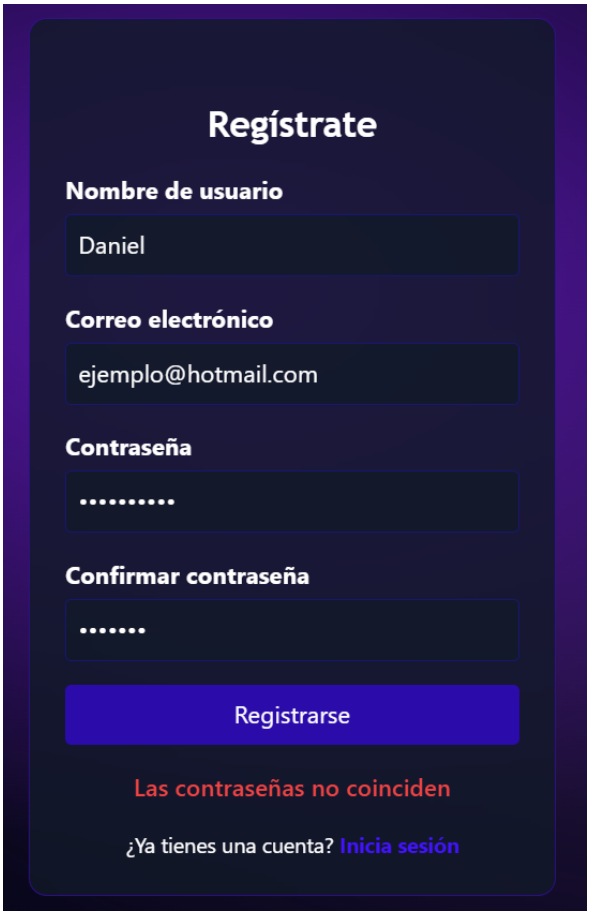

Figura 21 – Error al poner contraseñas distintas en el registro

4.4 Pruebas

En este apartado se detalla el proceso de pruebas implementado tanto para el backend como para el frontend de la aplicación. El objetivo principal de estas pruebas es garantizar la calidad del software, verificar que las funcionalidades cumplen con los requisitos establecidos y asegurar que el sistema responde correctamente ante diferentes escenarios, incluyendo casos de éxito y errores.

Para las pruebas, se ha utilizado la librería Vitest, una herramienta moderna y eficiente para realizar pruebas unitarias y de integración en proyectos de JavaScript y TypeScript. Una de las ventajas de Vitest es que permite unificar el entorno de pruebas tanto en el backend como en el frontend, lo que simplifica enormemente la configuración y el mantenimiento de los tests.

Junto a Vitest se han usado otras librerías que permiten que funcionen correctamente los tests: para el backend se ha usado Supertest para simular solicitudes HTTP a los *endpoints* de la API, y para el frontend se ha usado React Testing Library para interactuar con los componentes y verificar su comportamiento.

Cabe mencionar que, para evitar dependencias reales durante las pruebas, se utilizan *mocks*, que son unas funciones que permiten simular el comportamiento de:

- Base de datos: Por ejemplo, en el test de /getUserData, se simula el repositorio de usuarios (UserRepository) para devolver datos simulados o lanzar errores según el caso (Figura 22).

- Servicios externos: En el test de /checkEmail, se simula la librería node-fetch para simular las respuestas de la API de validación de correos.

4.4.1 Pruebas del Backend

El backend de la aplicación está diseñado como una API RESTful que expone múltiples *endpoints* para gestionar los recursos (usuarios, tokens, etc) y otras funcionalidades relacionadas con la seguridad. Las pruebas en el backend se centran en verificar el correcto funcionamiento de estos *endpoints*, simulando solicitudes HTTP y evaluando las respuestas del servidor.

Cada *endpoint* tiene un archivo de prueba dedicado donde se definen diversos casos de prueba. Los casos de prueba usados de manera general son:

Caso de éxito

- Que el *endpoint* responde con el código de estado correcto (200 o 201) y con los datos esperados cuando se proporcionan entradas válidas.

Casos de error

- Envío de solicitudes sin los datos necesarios (por ejemplo, petición a /checkEmail sin haber introducido un correo). Se verifica que el servidor responde con un estado de error 400 y un mensaje adecuado.

- En *endpoints* protegidos, como /getUserData, se simulan tokens de acceso inválidos o expirados, y se verifica que el servidor responde con un estado de error 401.

- Se simulan fallos en servicios externos, como la API de validación de correos, y se verifica que el servidor responde con un estado de error 500.

```
// Mock de UserRepository.getById para devolver datos simulados del usuario
UserRepository.getById.mockResolvedValueOnce({
  id: 'user-id',
  username: 'testuser',
  email: 'testuser@example.com',
  master_key: '',
});
```

Figura 22 – Mock de los datos del usuario devueltos por MySQL

4.4.2 Pruebas del Frontend

Las pruebas en el frontend se centran en verificar que los componentes de React se renderizan correctamente, manejan bien los eventos de usuario y se comunican con el backend de manera adecuada.

No todos los componentes tienen un archivo de prueba dedicado, pero sí todos los que representan una funcionalidad importante en la aplicación. Los casos de prueba usados para estas pruebas son:

Renderización inicial

- Verificar que el componente se renderiza correctamente con los elementos esperados. Por ejemplo, que en el componente del almacén se compruebe que exista el botón “Guardar”.

Interacción del usuario

- Simular eventos como clics, cambios en los campos de entrada y envíos de formularios, y verificar que el componente responde correctamente. Por ejemplo, en el almacén se simula el clic en el botón "Guardar" sin completar los campos y se verifica que se muestra un mensaje de error.

Comunicación con el backend

- Simular (a través de *mocks*) las llamadas a la API realizadas por el componente y verificar que se manejan correctamente. Por ejemplo, en el almacén se simula la llamada a la API para añadir una nueva entrada y se verifica que la entrada se muestra en la página después de la respuesta exitosa.

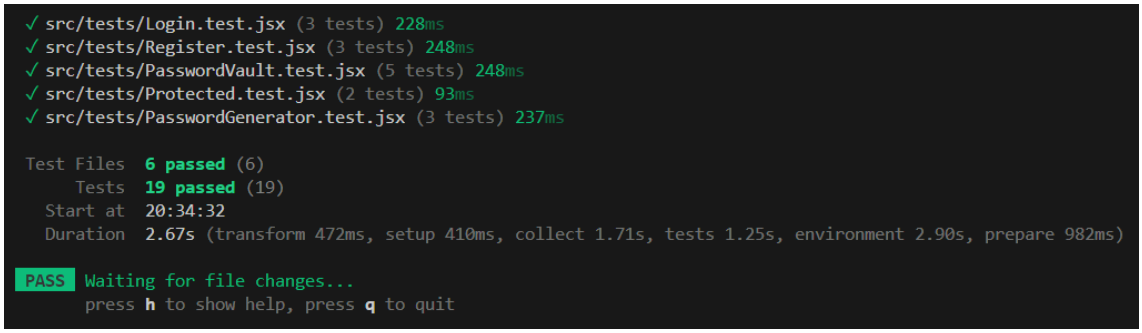

Figura 23 – Ejecución de Vitest exitosa de las pruebas del frontend

5 Conclusiones

Este trabajo ha tenido como propósito el desarrollo de una aplicación web destinada a facilitar a los usuarios la gestión segura de sus contraseñas y la verificación de la validez y seguridad de sus credenciales digitales. Para ello se establecieron una serie de objetivos específicos, que se han cumplido de manera satisfactoria a lo largo del desarrollo del proyecto.

En primer lugar, se ha logrado implementar un sistema de registro e inicio de sesión de usuarios, basado en tokens JWT y cookies seguras, que garantiza tanto la autenticación como la protección frente a ciberataques comunes como XSS o CSRF. Las contraseñas de los usuarios se almacenan de forma cifrada mediante *hash* usando la herramienta bcrypt, cumpliendo así con las mejores prácticas en cuanto a seguridad.

En cuanto al almacenamiento de contraseñas, se ha desarrollado un almacén de contraseñas cifradas, accesible únicamente por el usuario autenticado. La implementación de un sistema de cifrado simétrico robusto (AES-256-CBC) asegura que las contraseñas almacenadas sean resistentes ante cualquier ataque o incluso que se mantengas seguras en caso de una posible brecha de la base de datos.

El generador ajustable de contraseñas seguras ha sido otra de las herramientas planteadas como objetivo del trabajo. Se ha desarrollo esta funcionalidad permitiendo al usuario configurar características de la contraseña a generar como su longitud o los tipos de caracteres que incluye, además de añadir la posibilidad de copiarla al portapapeles con tan solo un click.

Otro de los objetivos especificados ha sido integrar funcionalidades que permitan a los usuarios verificar si sus contraseñas han sido filtradas y la validez de sus correos electrónicos**.** La aplicación ofrece una herramienta de comprobación de contraseñas filtradas mediante la API de *Have I Been Pwned*, garantizando la privacidad del usuario gracias a un modelo de consulta basado en *hashes* anónimos. Asimismo, se ha implementado la validación de correos electrónicos utilizando la API de *MailboxLayer*, permitiendo verificar aspectos como la sintaxis, la existencia del dominio y la disponibilidad.

Cabe destacar que todas estas funcionalidades se han integrado en una interfaz de usuario clara e intuitiva, desarrollada con React, siguiendo principios de usabilidad que facilitan su adopción incluso por parte de usuarios no expertos. Esto responde al objetivo de promover mejores hábitos de ciberseguridad entre los usuarios, facilitando el uso de herramientas que normalmente resultan complejas o inaccesibles.

Finalmente, a nivel de cumplimiento normativo y ético, la aplicación ha sido diseñada y desarrollada respetando los principios del Reglamento General de Protección de Datos (RGPD) [11], garantizando la protección y privacidad de los datos personales de los usuarios. Todas las funcionalidades han sido implementadas siguiendo buenas prácticas de seguridad desde el diseño, lo que refuerza la confianza del usuario en la herramienta.

En conclusión, los objetivos planteados al inicio de este trabajo se han alcanzado en su totalidad. Se ha desarrollado una aplicación que no solo cumple su función principal, facilitar la gestión y mejora de la seguridad de credenciales digitales, sino que también cumple con la concienciación de buenas prácticas en ciberseguridad. Además, se ha logrado construir una base tecnológica sólida sobre la que se podrán implementar futuras mejoras.

6 Trabajos Futuros

El desarrollo de este Trabajo de Fin de Grado ha permitido sentar las bases de una aplicación web funcional y segura para la gestión y monitoreo de contraseñas y correos electrónicos. Sin embargo, como en todo proyecto software, existen múltiples líneas de mejora y ampliación que permitirían evolucionar la herramienta hacia un producto mucho más completo y reconocido. A continuación, se detallan algunas de las principales propuestas de trabajos futuros.

1. Mejora de la experiencia de usuario y accesibilidad

Aunque la aplicación ya cuenta con una interfaz clara y funcional, se podría realizar un análisis de la experiencia de usuario, lo cual no se ha realizado para la versión actual, e incorporar elementos que mejoren tanto la usabilidad como la accesibilidad, como, por ejemplo:

- Opción de cambio de idioma.
- Soporte completo para navegación por teclado.
- Posibilidad de recuperar contraseña olvidada.

2. Soporte para distintos dispositivos

Actualmente, la aplicación es accesible desde cualquier navegador, pero el diseño ha sido enfocado para ordenador. Una mejora sería implementar la adaptación de la página a diferentes dispositivos

3. Integración de autenticación multifactor (MFA)

Una mejora importante del sistema sería ofrecer soporte para autenticación multifactor (MFA), permitiendo a los usuarios reforzar aún más la seguridad de su cuenta. Se podría implementar MFA mediante:

- Envío de códigos temporales por correo electrónico o SMS.
- Integración con aplicaciones de autenticación TOTP como Google Authenticator o Authy.

4. Auditoría y registro de actividad

En entornos corporativos o de mayor exigencia en seguridad, sería muy útil proporcionar un sistema de auditoría que permita registrar los eventos relevantes en la aplicación:

- Inicios de sesión y cierres de sesión.
- Modificaciones en el almacén de contraseñas.
- Cambios en la clave maestra.
- Intentos de acceso fallidos.

Estos registros podrían almacenarse cifrados y ser accesibles por el usuario (o por un administrador en el caso de uso empresarial). Actualmente la aplicación cuenta con una pequeña intención de esta funcionalidad al incluir la fecha de creación de cada recurso en la base de datos.

5. Mejora de la protección de la clave maestra

Si bien actualmente la clave maestra se almacena protegida mediante hash, al ser la llave que permite el acceso a las demás contraseñas del usuario, podrían explorarse estrategias más avanzadas para protegerla, como:

- Derivación de claves mediante algoritmos como Argon2
- Implementación de un sistema de recuperación de clave maestra seguro, basado en frases de recuperación o sistemas de compartición de secretos.

6. Proporcionar escalabilidad

Actualmente, la aplicación está pensada para correr en un entorno de pruebas o de pequeñas cargas de peticiones. De cara a un futuro despliegue más amplio sería recomendable implementar:

- Balanceo de carga y redundancia.
- Añadir pruebas de carga y optimización de rendimiento.
- Migrar a un sistema de base de datos más escalable si fuera necesario.

8. Nuevas funcionalidades

Por último, existen múltiples funcionalidades complementarias que podrían incorporarse para ampliar el valor de la aplicación, por ejemplo:

- Análisis de la fortaleza de las contraseñas generadas o almacenadas en tiempo real.
- Detección automática de contraseñas reutilizadas.
- Alertas automáticas en caso de filtraciones de datos relacionadas con el correo del usuario.
- Integración con servicios de notificación (correo, Telegram, etc.) para alertas de seguridad en todo momento.

En resumen, el proyecto desarrollado constituye una base sólida que ofrece diversas posibilidades de evolución y mejora. Las propuestas de trabajo descritas permitirían no solo mejorar e incrementar la funcionalidad, sino también escalar la aplicación a un nivel de producto real en producción, adaptable tanto para usuarios individuales como para entornos corporativos. La aplicación podría posicionarse como un servicio de ciberseguridad accesible, robusto y alineado con las mejores prácticas actuales, ofreciendo una contribución real a la mejora de los hábitos de seguridad en la era digital.

7 Análisis de Impacto

Desde una perspectiva empresarial, la aplicación desarrollada es altamente aplicable tanto para pequeñas y medianas empresas (*pymes*) además de para usuarios individuales, sobre todo si le aplican mejoras enfocadas a llevar el proyecto a producción.

En el contexto de las pymes, suele ser un desafío gestionar la seguridad de un gran número de cuentas con recursos limitados. Una herramienta de gestión de contraseñas proporciona una forma asequible y eficaz de mejorar su nivel de seguridad. Esta aplicación podría implantarse internamente en una pyme u ofrecerse como un producto *Software-as-a-Service* (SaaS). Su desarrollo con Node.js y React facilita su despliegue multiplataforma y escalabilidad, lo que da lugar a una mayor viabilidad comercial como producto software.

En el aspecto personal, la aplicación tiene un impacto positivo directo en los hábitos de seguridad del usuario y en su experiencia cotidiana en internet al manejar credenciales.

En primer lugar, al utilizar este gestor el usuario adopta mejores prácticas de seguridad, por ejemplo, la generación aleatoria de contraseñas fuertes y únicas elimina la tentación de reciclar contraseñas o elegir unas que sean débiles. Esto supone una mejora en la protección de sus cuentas personales, ya que reduce en gran medida la probabilidad de comprometer varios servicios por culpa de una sola contraseña filtrada.

Muchos usuarios son conscientes de que sus hábitos de seguridad son mejorables, pero aun así incurren en prácticas de riesgo, lo que puede derivar en problemas de seguridad no solo personales, sino que también empresariales. Según un informe, la mayoría de los empleados en empresas saben que tienen un problema con las contraseñas, pero no hacen nada al respecto [12]. La introducción de una herramienta cómoda y confiable como la que se ha desarrollado podría ayudar a evitar esto, facilitando que el usuario dé el salto hacia un comportamiento más seguro sin esfuerzo adicional. Además, la función de comprobación de filtraciones de contraseñas no solo protege las cuentas de los usuarios, sino que incrementa su concienciación sobre las amenazas digitales presentes, educándole en la importancia de usar credenciales resistentes y de reaccionar ante posibles compromisos de seguridad.

En conclusión, el proyecto mejora el mantenimiento de la seguridad personal del usuario promedio y le proporciona mayor tranquilidad en su vida digital, al saber que sus claves están protegidas y bajo control.

8 Bibliografía

[1] Verizon. (2023). *Data Breach Investigations Report* [Online]. Disponible en:

https://www.verizon.com/business/resources/reports/2023-data-breach-investigations-report-dbir.pdf

[2] Agencia Española de Protección de Datos. (2023). *Guía de privacidad y seguridad en internet* [Online]. Disponible en:

https://www.aepd.es/areas-de-actuacion/recomendaciones/medidas

[3] Stack Overflow. (2023). *Stack Overflow Developer Survey 2023* [Online]. Disponible en: https://survey.stackoverflow.co/2023/

[4] UDIT. (2024, oct 16). *¿Qué es Node.js y qué ventajas ofrece?* [Online]. Disponible en:

https://udit.es/actualidad/que-es-node-js-y-que-ventajas-ofrece/

[5] Express. *Framework para aplicaciones web en Node.js* [Online]. Disponible en: https://expressjs.com/

[6] npm. *MySQL2 – Cliente MySQL para Node.js* [Online]. Disponible en:

https://www.npmjs.com/package/mysql2

[7] Chris Tozzi. (2023, sep 20). *Vitest vs. Jest: Differences, Benefits, and Challenges* [Online]. Disponible en:

https://saucelabs.com/resources/blog/vitest-vs-jest-comparison

[8] Equipo editorial de IONOS. (2020). *Introducción a JSON Web Token (JWT)* [Online]. Disponible en:

https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/json-web-token-jwt/

[9] Mozilla Developer Network (MDN). *Intercambio de recursos de origen cruzado (CORS)* [Online]. Disponible en:

https://developer.mozilla.org/es/docs/Web/HTTP/Guides/CORS

[10] Geekflare Team, (2025, ene 2). *Angular Vs. React: ¿Cuál debe elegir en 2025?* [Online]. Disponible en: https://geekflare.com/es/angular-vs-react/

[11] EU. (2022). *Reglamento general de protección de datos (RGPD)* [Online]. Disponible en:

https://eur-lex.europa.eu/ES/legal-content/summary/general-data-protection-regulation-gdpr.html

[12] Rose de Ferrency. (2023, oct 9). *Gestores de contraseñas: protección al servicio de su empresa* [Online]. Disponible en: blog.lastpass.com

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jun 04 16:41:45 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)