Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

KEYSHIELD: UN GESTOR DE CONTRASEÑAS SEGURO BASADO EN WEB

Alumno: Enrique López Alonso

Tutora: María del Pilar Grande González

KEYSHIELD: UN GESTOR DE CONTRASEÑAS SEGURO BASADO EN WEB

Enrique López Alonso

Índice general

Índice de figuras

Índice de cuadros

"Elige un trabajo que te guste y no tendrás que trabajar ni un día de tu vida."
— Confucio

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todas las personas que han hecho posible la realización de este Trabajo de Fin de Grado.

En primer lugar, a mi tutora, Pilar, por su orientación, disponibilidad y valiosas sugerencias durante todo el desarrollo del proyecto. Su apoyo ha sido clave para poder abordar los distintos retos técnicos con seguridad y perspectiva.

Agradezco también a mis profesores y profesoras de la Universidad de Valladolid, cuyo conocimiento y vocación docente han contribuido de forma decisiva a mi formación académica.

A mi familia, por su paciencia, comprensión y constante ánimo, especialmente en los momentos más exigentes del proceso. Sin su apoyo incondicional, este proyecto no habría sido posible.

A mis amigos y compañeros, por su compañía y motivación a lo largo de estos años, y por ayudarme a mantener el equilibrio entre estudio y vida personal.

Finalmente, a todas las personas que han colaborado indirectamente en este proyecto, compartiendo su tiempo, experiencia o recursos, y que han hecho que esta etapa formativa haya sido más enriquecedora.

Gracias a todos por estar ahí.

Resumen

Los usuarios de sistemas informáticos suelen tener dificultades a la hora de crear contraseñas seguras y fáciles de recordar sin acudir a métodos físicos como escribir en papel.

El presente Trabajo de Fin de Grado tiene como objetivo el desarrollo de KeyShield, un gestor de contraseñas seguro basado en tecnología web. Esta aplicación permitirá a los usuarios almacenar, generar y organizar credenciales de forma eficiente y protegida. Para ello, el sistema integrará técnicas avanzadas de cifrado que garanticen la confidencialidad y la integridad de los datos. Además, se prestará especial atención al diseño de una interfaz intuitiva y accesible, con el fin de mejorar la experiencia de usuario.

Palabras claves: seguridad, contraseña, protección, integridad.

Abstract

Users of computer systems often struggle to create secure and memorable passwords without resorting to physical methods such as writing them down on paper.

This Final Degree Project aims to develop KeyShield, a secure password manager based on web technology. This application will allow users to store, generate, and organize credentials efficiently and securely. To achieve this, the system will integrate advanced encryption techniques to ensure the confidentiality and integrity of the data. Additionally, special attention will be paid to designing an intuitive and accessible interface in order to enhance the user experience.

Keywords: security, password, protection, integrity.

Parte I

Memoria del Proyecto

Capítulo 1

Descripción del proyecto

1.1. Introducción

En la era de la digitalización, la gestión segura de las credenciales de acceso se ha convertido en una necesidad crítica tanto para usuarios individuales como para organizaciones. La creciente cantidad de servicios y aplicaciones en red y la constante amenaza y evolución de los ciberataques han puesto de manifiesto la importancia de utilizar contraseñas seguras, únicas y correctamente almacenadas. Esta preocupación por la seguridad digital ha sido el motor principal que impulsó el desarrollo de Keyshield.

El propósito principal de este proyecto es diseñar y desarrollar una aplicación que permita a los usuarios almacenar y gestionar sus contraseñas de forma segura, garantizando dos principios fundamentales: confidencialidad e integridad. A través del uso de técnicas de cifrado robustas, buenas prácticas de seguridad y un entorno en red resistente, Keyshield busca ser una solución que competente frente a otras ofertas existentes.

Esta aplicación se concibe desde el principio de security by design que se define por la integración de la seguridad como un aspecto central desde las primeras etapas de diseño y a lo largo de todo el ciclo de vida del mismo usando técnicas como el cifrado de extremo a extremo o el uso de autenticación multifactor.

Keyshield está concebido como una herramienta independiente, con posibilidad de ser integrada en pequeñas y medianas empresas que requieran de una solución de gestión de contraseñas para sus empleados. En un contexto global, se quiere contribuir a mejorar la higiene digital, reducir el riesgo por malas prácticas y fomentar la concienciación en seguridad informática para todos los usuarios.

1.2. Objetivos del trabajo

El objetivo principal de este proyecto es desarrollar una aplicación web segura destinada a la gestión de contraseñas. A continuación, se describen los objetivos específicos identificados.

ID-OBJ	Descripción
OBJ-01	Desarrollar una aplicación web segura para la gestión de contraseñas.
OBJ-02	Implementar cifrado AES-256 para garantizar la seguridad de las contraseñas almacenadas.
OBJ-03	Proporcionar un sistema de autenticación robusto, basado en contraseñas maestras fuertes y posibles mejoras con autenticación de dos factores (2FA).
OBJ-04	Ofrecer funcionalidad de generación de contraseñas seguras con parámetros configurables por el usuario.
OBJ-05	Permitir la organización de contraseñas mediante etiquetas o categorías.
OBJ-06	Garantizar la accesibilidad y escalabilidad del sistema mediante almacenamiento en la nube.

Cuadro 1.1: Objetivos del proyecto

1.3. Entorno de aplicación

La gestión de contraseñas es un desafío crucial en la ciberseguridad. Muchos usuarios tienden a reutilizar contraseñas débiles, aumentando el riesgo de ataques como el phishing o el credential stuffing. Para mitigar estos riesgos, han surgido gestores de contraseñas como LastPass, Bitwarden y 1Password, que ofrecen almacenamiento seguro y generación de contraseñas robustas. Sin embargo, estos servicios dependen en gran medida de la confianza en terceros y algunos han enfrentado vulnerabilidades de seguridad en el pasado. Entre las más comunes se encuentran el almacenamiento inseguro de credenciales, la exposición de datos por brechas, fallos en la autenticación multifactor, vulnerabilidades en extensiones de navegador y ataques de phishing dirigidos. Estos casos evidencian la necesidad de diseñar soluciones con fuertes garantías criptográficas y buenas prácticas de desarrollo seguro.

1. **LastPass**: Administrador de contraseñas que guarda y genera contraseñas de forma segura. Permite almacenar contraseñas, notas, información personal y tarjetas de crédito. Ofrece una opción gratuita y una premium que incluye más funciones, como compartir contraseñas y almacenamiento ilimitado en múltiples dispositivos. LastPass también tiene autenticación de dos factores para mayor seguridad.

2. **Bitwarden**: Administrador de contraseñas de código abierto y gratuito, con opciones premium a un costo bajo. Al igual que LastPass, guarda contraseñas, tarjetas

de crédito y otra información sensible de manera encriptada. Su principal atractivo es que es completamente transparente por ser de código abierto, lo que da confianza a los usuarios que valoran la seguridad y privacidad. También tiene opciones para compartir contraseñas y autenticación de dos factores.

3. **1Password**: Gestor de contraseñas que destaca por su facilidad de uso y por sus potentes características de seguridad, como el cifrado de extremo a extremo. Almacena contraseñas, tarjetas, documentos e información confidencial. Ofrece planes para individuos, familias y equipos. Es una opción de pago, pero incluye funcionalidades como la recuperación de contraseñas y la capacidad de compartir de manera segura con otros usuarios. Además, ofrece una opción de autenticación multifactor.

KeyShield se diferencia al ofrecer una arquitectura con cifrado AES-256 de extremo a extremo y almacenamiento seguro. Además, busca mejorar la usabilidad y organización avanzada de credenciales, proporcionando una solución que combina seguridad y accesibilidad de manera eficiente.

	LastPass	**Bitwarden**	**1Password**	**KeyShield**
Cifrado	AES-256	AES-256	AES-256	AES-256
Autent. 2FA	✓	✓	✓	✓
Org. contraseñas	Carpetas, etiquetas	Carpetas, etiquetas	Categorías	Categorías, etiquetas
Sincronización nube	✓	✓	✓	✓
Gestión claves	Nube	Local	Interna	Nube con KMS propio

Cuadro 1.2: Comparativa entre gestores de contraseñas

Capítulo 2

Metodología

En este capítulo se presenta el modelo de desarrollo de Keyshield. Se ha adoptado una metodología basada en Scrum, adaptada al contexto de un proyecto individual. Caracterizada por una organización más flexible, mejor gestión del tiempo y una evolución constante del proyecto mediante entregas incrementales.

2.1. Proceso de desarrollo

Aunque Scrum está diseñado para equipos, tanto roles como características principales se han aplicado siguiendo los principios fundamentales de esta metodología.

2.1.1. Roles

- **Product Owner**: Representa al cliente o usuario final. Se encarga de definir y priorizar el product backlog, asegurando el trabajo en lo que aporta más valor. En este proyecto, este rol fue asumido por el propio autor, quien definió las funcionalidades clave basándose en criterios de utilidad, seguridad y usabilidad.
- **Scrum Master**: Actúa como facilitador del proceso, garantizando que se sigan las buenas prácticas de Scrum. También ayuda a eliminar impedimentos que puedan afectar al progreso del equipo. En el desarrollo de esta aplicación se asumió una autoorganización y reflexión sobre la metodología aplicada.
- **Equipo de desarrollo**: Grupo encargado de construir el producto. En este caso representado por una sola persona que asume todas las tareas técnicas: diseño, implementación, pruebas y documentación.

2.1.2. Eventos

- **Sprint**: Ciclo iterativo de trabajo de una o dos semanas con el objetivo de entregar una parte funcional y útil del producto. En el entorno de este proyecto se escogen sprints de 2 semanas.

- **Sprint Planning**: Reunión al inicio de cada sprint donde se seleccionan los elementos del Product Backlog que se van a desarrollar. Se definen los objetivos de sprint y las tareas necesarias.
- **Daily Scrum**: Aunque no se realizaron reuniones diarias, se mantenía un control diario del progreso mediante la revisión del tablero Kanban.
- **Sprint Review**: Al finalizar un sprint se revisaban los resultados: funcionalidades completadas y valor del producto.
- **Sprint Retrospective**: Reflexiones breves sobre el proceso: qué funcionó, qué se puede mejorar y cómo aplicar mejoras de cara al siguiente sprint.

En el contexto de este proyecto, se estableció una duración fija de dos semanas por sprint, con el objetivo de mantener un equilibrio adecuado entre capacidad de entrega y posibilidad de reacción ante cambios. Esta duración permite planificar avances significativos sin que el periodo se vuelva tan largo como para perder visibilidad sobre el progreso.

Las reuniones de coordinación, en particular el Sprint Planning, se realizaban al inicio de cada sprint, normalmente al comienzo de la semana correspondiente (generalmente los lunes). En este encuentro se definían los objetivos específicos del sprint y se descomponían en tareas claras dentro del tablero Kanban.

Aunque no se llevaron a cabo Daily Scrums formales, se optó por una revisión diaria del tablero, lo que permitió mantener un seguimiento continuo del estado de cada tarea sin necesidad de reuniones adicionales. Esta decisión se tomó por cuestiones de eficiencia, dado que el equipo era reducido y existía una comunicación fluida.

La Sprint Review y la Retrospective se llevaban a cabo al finalizar cada sprint (normalmente los viernes), como una única sesión de evaluación y reflexión. Esto permitía revisar las funcionalidades entregadas y plantear mejoras tanto técnicas como metodológicas de cara al siguiente ciclo.

2.1.3. Entorno de trabajo

- **Tablero Kanban (Trello)**: Se creó un tablero Kanban con la herramienta Trello que permitió gestionar visualmente el flujo de trabajo organizando en listas los distintos estados de las tareas:
  - **Backlog**: Funcionalidades generales sin planificar.
  - **To Do**: Tareas del sprint actual.
  - **In Progress**: Tareas en desarrollo.
  - **Testing**: Funcionalidades terminadas pendientes de revisión.
  - **Done**: Tareas finalizadas y entregadas.
- **Git+GitHub**: Se utilizó control de versiones con Git, y el repositorio del proyecto se alojó en Github.

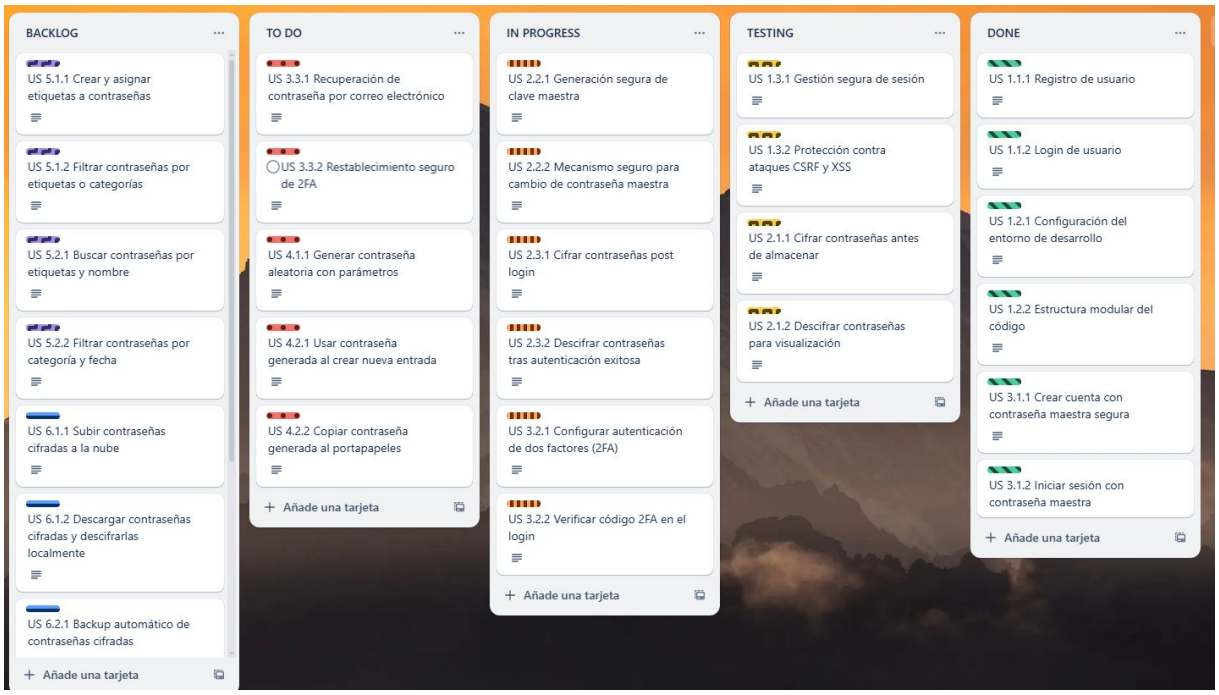


Figura 2.1: Tablero Trello

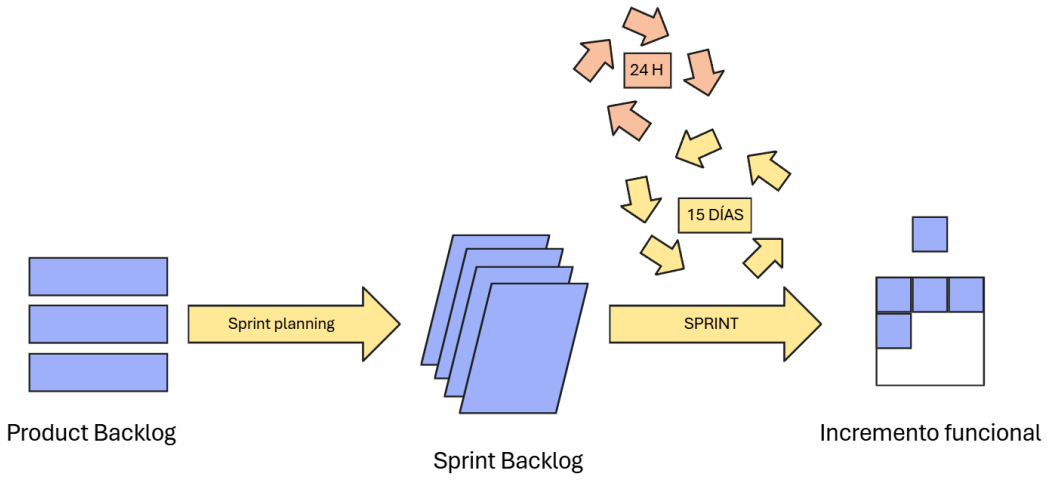


Figura 2.2: Modelo de desarrollo Scrum

En el desarrollo de Keyshield se ha empleado un enfoque basado en la programación orientada a objetos (POO) y en el paradigma de componentes, propio del ecosistema de Node.js y React. En conjunto, el uso de estos paradigmas ha permitido el desarrollo de una arquitectura limpia, escalable y mantenible, adecuada para una aplicación como la propuesta, donde la seguridad, la modularidad y la claridad estructural son fundamentales.

2.2. Herramientas utilizadas

En esta sección se definirán las herramientas usadas en el desarrollo de Keyshield basadas en el ecosistema web. Esta sección describe las herramientas, lenguajes, frameworks y bibliotecas seleccionadas, justificando su elección.

⋆ **Lenguajes y entornos de desarrollo**

- **JavaScript (ES6+)**: Lenguaje principal de programación, utilizado tanto en el frontend como en el backend. permite un desarrollo full-stack coherente y simplifica la comunicación entre cliente y servidor.
- **Node.js**: Entorno de ejecución del backend, basado en JavaScript. Permite manejar operaciones asíncronas, realizar tareas de red y conectar con la base de datos de forma eficiente.

⋆ **Frameworks y liberias principales**

- **Express**: Framework ligero para Node.js que facilita la creación de servidores y APIs REST. Base estructural del backend desarrollado.
- **React**: Biblioteca de frontend basada en componentes reutilizables, usada para la elaboración de la interfaz modular y dinámica.

⋆ **Bibliotecas de seguridad y autenticación**

- **bcrypt/bcryptjs**: Permiten aplicar hash seguro a las contraseñas de usuario.
- **jsonwebtoken (JWT)**: Utilizado para la creación de tokens seguros de autenticación, que permiten el acceso a recursos protegidos sin necesidad de sesiones persistentes.
- **passport**: Middleware de autenticación flexible. Usado para la integración de estrategias de autenticación como JWT.
- **crypto**: Se ha usado para implementar cifrado simétrico (AES-256).

⋆ **Bases de datos**

- **PostgreSQL**: Sistema de gestión de bases de datos relacional elegido por su robustez, seguridad y compatibilidad con el entorno de desarrollo.

- **pg**: Cliente oficial de PostgreSQL para Node.js, utilizado para conectar el backend con la base de datos.
- **Knex**: Query builder SQL que facilita la escritura y gestión de consultas complejas mediante una sintaxis programática y legible.
- **Objection.js**: ORM (Object Relational Mapper) basado en Knex, que permite trabajar con la base de datos mediante modelos orientados a objetos.

⋆ **Entorno de desarrollo**

- **Windows 10**: Sistema operativo en el que se ha desarrollado la aplicación, compatible con todas las herramientas mencionadas.
- **Git+Github**: Utilizados para el control de versiones del proyecto, permitiendo un seguimiento detallado del código.
- **Trello**:Aplicación para la gestión de tareas. Usado para planificar y monitorizar el proceso de desarrollo siguiendo un enfoque ágil.

Categoría	Herramienta / Biblioteca	Función principal
Lenguaje de programación	JavaScript (ES6+)	Lenguaje principal frontend y backend
Entorno backend	Node.js	Ejecución del servidor backend
Framework backend	Express	Creación de API REST y gestión de rutas
Framework frontend	React	Construcción de interfaz de usuario modular
Seguridad y autenticación	bcrypt / bcryptjs	Hashing seguro de contraseñas
	jsonwebtoken (JWT)	Gestión de tokens de acceso
	passport	Middleware de autenticación flexible
	express-validator	Validación y saneamiento de datos de entrada
	crypto (Node.js nativo)	Cifrado y funciones criptográficas
Comunicación HTTP	axios	Cliente HTTP para peticiones frontend-backend
Generación de contraseñas	password-generator	Creación de contraseñas aleatorias seguras
Variables de entorno	dotenv	Gestión segura de variables y configuraciones
CORS	cors	Permite comunicación entre frontend y backend
Base de datos	PostgreSQL	Sistema gestor de base de datos relacional
Cliente PostgreSQL	pg	Conexión y consultas con PostgreSQL
Query builder	Knex	Escritura programática de consultas SQL
ORM	Objection.js	Modelos orientados a objetos para la base de datos
Control de versiones	Git + GitHub	Gestión y seguimiento del código fuente
Herramientas desarrollo	Nodemon	Reinicio automático del servidor en desarrollo
Gestión de proyecto	Trello	Organización y seguimiento de tareas
Sistema operativo	Windows 10	Entorno local de desarrollo

Cuadro 2.1: Stack tecnológico

2.3. Arquitectura

En esta sección se explicará la arquitectura de la aplicación, abarcando tanto su arquitectura lógica como la arquitectura física.

2.3.1. Arquitectura lógica

La arquitectura lógica se refiere a la organización conceptual de la aplicación, definiendo los componentes, capas y sus responsabilidades dentro del sistema. Keyshield sigue una estructura en capas compuesta por:

- **Capa de presentación (Frontend)**: Implementada con React, esta capa gestiona la interfaz con el usuario final. Se encarga de mostrar la información, capturar datos y enviar peticiones al backend mediante Axios. También maneja la autenticación basada en tokens JWT.
- **Capa de lógica de negocio (Backend)**: Implementada en Node.js con Express, centraliza la lógica de la aplicación, incluyendo la validación de datos, autenticación y autorización, cifrado de contraseñas y gestión de sesiones. Esta capa recibe las solicitudes del frontend, las procesa y responde adecuadamente.
- **Capa de Persistencia (Base de datos)**: En esta capa reside la base de datos PostgreSQL, que almacena la información de usuarios y contraseñas cifradas. Se utiliza Objection.js como ORM y Knex como query builder para facilitar el acceso y manipulación de los datos.

2.3.2. Arquitectura física

La arquitectura física describe cómo se distribuyen y despliegan los componentes de la aplicación hardware o entornos concretos. Para este proyecto se define de la siguiente manera:

- **Cliente (Frontend)**: La aplicación React se ejecuta en el navegador del usuario, que puede ser cualquier dispositivo con acceso a internet.
- **Servidor (Backend)**: El backend Node.js con Express se ejecuta en un servidor dedicado que recibe las peticiones del cliente, procesa la lógica de negocio y accede a la base de datos.
- **Base de datos**: PostgreSQL se aloja en un servidor de base de datos.
- **Comunicación**: La comunicación entre frontend y backend se realiza mediante HTTPS, asegurando la confidencialidad e integridad de los datos transmitidos. La autenticación se gestiona con tokens JWT que se envían en las cabeceras de las peticiones.

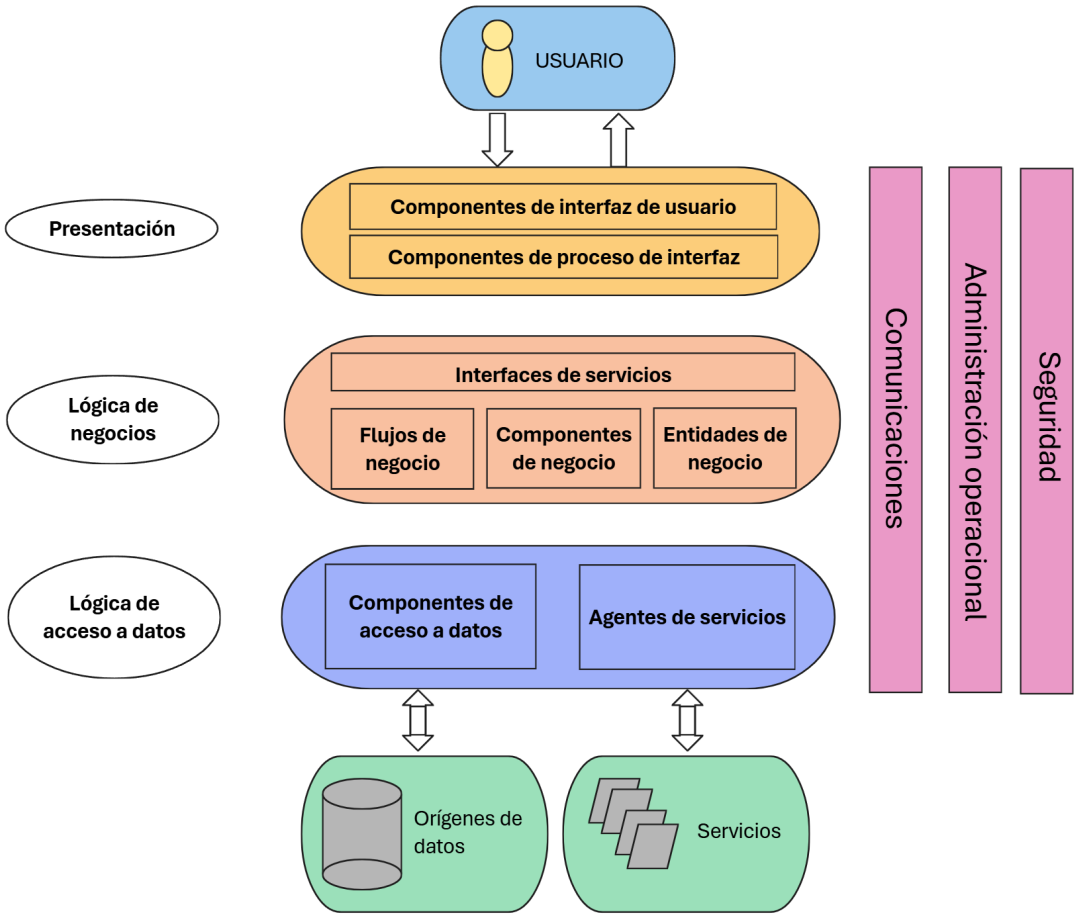


Figura 2.3: Arquitectura por capas

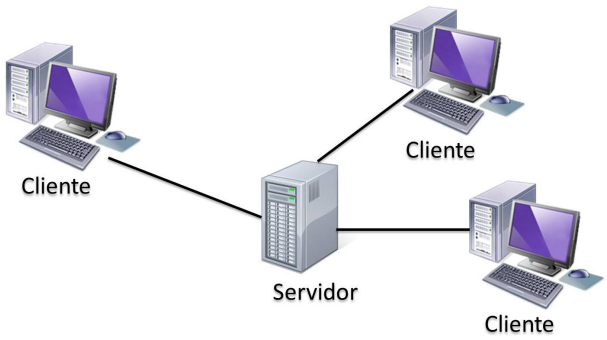


Figura 2.4: Arquitectura física

2.4. Definición de siglas y abreviaturas

Sigla	Significado
API	Application Programming Interface (Interfaz de Programación de Aplicaciones)
JWT	JSON Web Token (Token Web en formato JSON)
KMS	Key Management Service (Servicio de Gestión de Claves)
ORM	Object-Relational Mapping (Mapeo Objeto-Relacional)
UI	User Interface (Interfaz de Usuario)
UX	User Experience (Experiencia de Usuario)
HTTP	Hypertext Transfer Protocol (Protocolo de Transferencia de Hipertexto)
2FA	Two-Factor Authentication (Autenticación en dos factores)
SQL	Structured Query Language (Lenguaje de Consulta Estructurado)

Cuadro 2.2: Definición de siglas y abreviaturas

Capítulo 3

Planificación

En este capítulo se aboradarán las cuestiones relativas a la planificación del trabajo.

3.1. Estimación del esfuerzo

Para llevar a cabo la estimación del esfuerzo se ha optado por el uso de Puntos de Historia, una unidad de medida relativa que permite cuantificar el esfuerzo, la complejidad y la incertidumbre asociada a cada historia, sin necesidad de realizar predicciones exactas. La estimación de puntos de historia se ha realizando siguiendo una serie de Fibonacci modificada: 1, 2, 3, 5, 8, 13, que refleja un crecimiento no lineal del esfuerzo ante el aumento de la complejidad de las tareas. Para el análisis se ha tomado como referencia una historia bien entendida y de bajo esfuerzo para comparar el resto de historias de forma relativa, asignando una puntuación proporcional.

ID-US	Historia de Usuario	Puntos de Historia
US 1.1.1	Registro de usuario	3
US 1.1.2	Login de usuario	3
US 1.2.1	Configuración del entorno de desarrollo	2
US 1.2.2	Implementar estructura modular del código	3
US 1.3.1	Gestión segura de sesión de usuario	5
US 1.3.2	Protección contra ataques CSRF y XSS	5
US 2.1.1	Cifrar contraseñas antes de almacenar	5
US 2.1.2	Descifrar contraseñas para visualización	5
US 2.2.1	Generación segura de clave maestra	5
US 2.2.2	Mecanismo seguro para cambio de contraseña maestra	5
US 2.3.1	Cifrar contraseñas antes de almacenarlas	3
US 2.3.2	Descifrar contraseñas tras autenticación exitosa	3
US 3.1.1	Crear cuenta con contraseña maestra segura	3
US 3.1.2	Iniciar sesión con contraseña maestra	3
US 3.2.1	Configurar autenticación de dos factores (2FA)	5
US 3.2.2	Verificar código 2FA en el login	5
US 3.3.1	Recuperación de contraseña maestra mediante correo electrónico	8
US 3.3.2	Restablecimiento seguro de 2FA	5
US 4.1.1	Generar contraseña aleatoria con parámetros	3
US 4.2.1	Usar contraseña generada al crear nueva entrada	2
US 4.2.2	Copiar contraseña generada al portapapeles	1
US 5.1.1	Crear y asignar etiquetas a contraseñas	3
US 5.1.2	Filtrar contraseñas por etiquetas o categorías	3
US 5.2.1	Buscar contraseñas por etiquetas y nombre	3
US 5.2.2	Filtrar contraseñas por categoría y fecha	3
US 6.1.1	Subir contraseñas cifradas a la nube	5
US 6.1.2	Descargar contraseñas cifradas y descifrarlas localmente	5
US 6.2.1	Backup automático de contraseñas cifradas	5
US 6.2.2	Restaurar contraseñas desde backup	5
US 6.3.1	Control de acceso basado en roles para usuarios empresariales	8
US 6.3.2	Auditoría y registro de actividades de usuario	5
	Total puntos de historia	**125**

Cuadro 3.1: Esfuerzo por puntos de historia

3.2. Planificación temporal

En el presente proyecto, siguiendo la metodología SCRUM, la planificación temporal se basa en la división del trabajo en sprints, períodos cortos y fijos en los que se desarrollan conjuntos específicos de historias de usuario (o tareas).

Este enfoque permite establecer metas claras y alcanzables en cada sprint fomentando la entrega continua de valor y la capacidad de adaptación ante cambios o nuevas prioridades.

Además, la planificación establecida contempla un sprint final destinado a revisiones, pruebas, correcciones y documentación, asegurando que el producto final cumpla con los estándares de calidad y requisitos del proyecto.

Sprint 1 (Semanas 1–2): Configuración del entorno y estructura del código; desarrollo de registro, login y gestión de contraseña maestra en el alta e inicio de sesión; primeras pruebas unitarias.

Sprint 2 (Semanas 3–4): Implementar gestión segura de sesiones, protección contra ataques CSRF/XSS, cifrado y descifrado básico de contraseñas, y pruebas de seguridad iniciales.

Sprint 3 (Semanas 5–6): Desarrollo de clave maestra y su cambio seguro, cifrado/descifrado reforzado, configuración y verificación de 2FA, y pruebas de autenticación avanzada.

Sprint 4 (Semanas 7–8): Implementar recuperación de contraseña maestra por correo, restablecimiento de 2FA, generador de contraseñas seguras, integración en nuevas entradas y función de copia al portapapeles.

Sprint 5 (Semanas 9–10): Creación y gestión de etiquetas para contraseñas, filtrado y búsqueda por múltiples criterios, optimización de consultas y pruebas de rendimiento.

Sprint 6 (Semanas 11–12): Sincronización en la nube, backup y restauración cifrados, gestión de roles y permisos empresariales, auditoría de actividad, y pruebas multidispositivo.

Sprint 7 (Semanas 13–14): Revisión final, pruebas y documentación. Refinamiento, pruebas finales, documentación técnica y de usuario, despliegue en producción y presentación del sistema.

La siguiente tabla resume la organización de los sprints de este proyecto.

Sprint	Semanas	Objetivo principal	Historias de Usuario	Puntos
Sprint 1	Semana 1–2	Setup inicial, Registro y Login	US 1.2.1, 1.2.2, 1.1.1, 1.1.2, 3.1.1, 3.1.2	17
Sprint 2	Semana 3–4	Seguridad de sesión y cifrado básico	US 1.3.1, 1.3.2, 2.1.1, 2.1.2	18
Sprint 3	Semana 5–6	Clave maestra y autenticación avanzada	US 2.2.1, 2.2.2, 2.3.1, 2.3.2, 3.2.1, 3.2.2	26
Sprint 4	Semana 7–8	Recuperación y generación de contraseñas	US 3.3.1, 3.3.2, 4.1.1, 4.2.1, 4.2.2	19
Sprint 5	Semana 9–10	Gestión de etiquetas y filtrado	US 5.1.1, 5.1.2, 5.2.1, 5.2.2	12
Sprint 6	Semana 11–12	Sincronización en la nube, backup y control empresarial	US 6.1.1, 6.1.2, 6.2.1, 6.2.2, 6.3.1, 6.3.2	33
Sprint 7	Semana 13–14	Revisión final, pruebas y documentación	Refinamiento, testing, correcciones, despliegue final	–

Cuadro 3.2: Planificación temporal del proyecto por Sprints (metodología SCRUM)

El correspondiente diagrama de Gantt permite observar de forma gráfica la distribución temporal de las tareas:

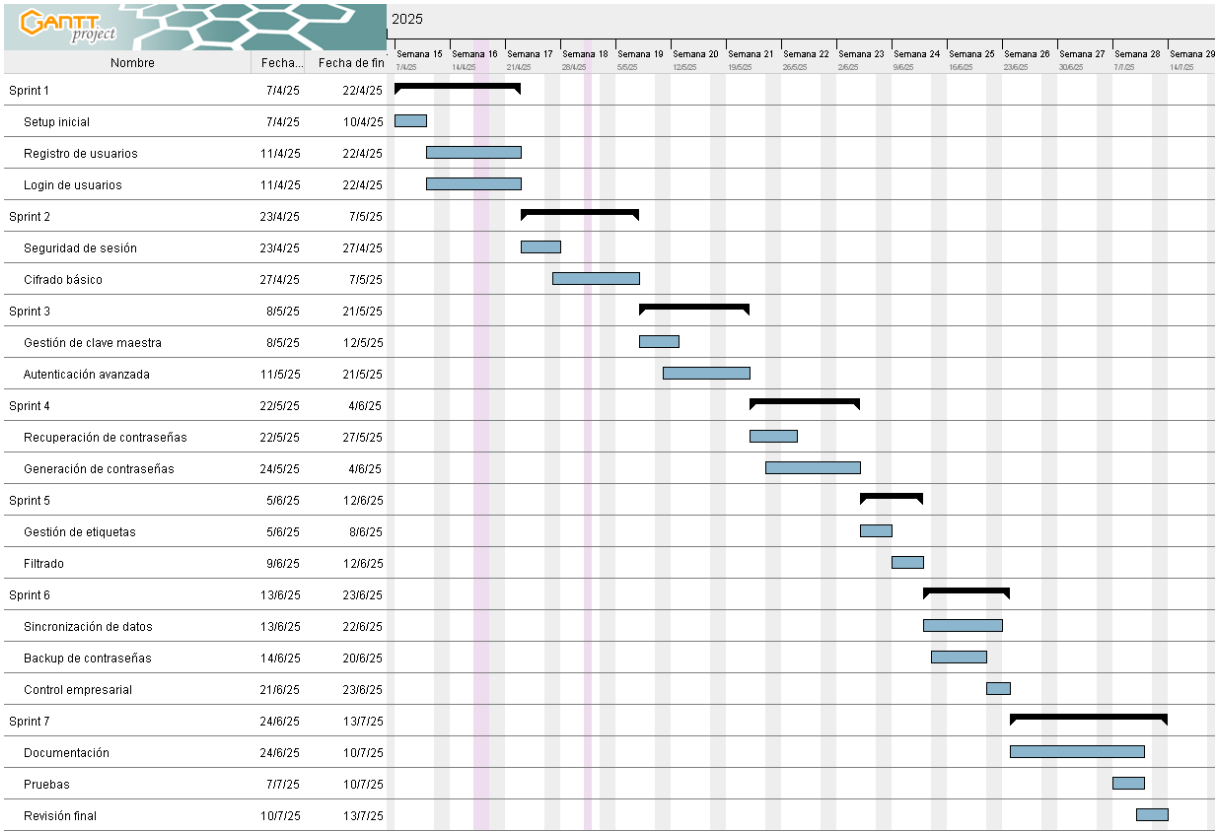


Figura 3.1: Diagrama de Gantt

3.3. Presupuesto económico

Aunque el desarrollo del presente Trabajo de Fin de Grado ha sido realizado de forma individual, utilizando recursos propios, se ha llevado a acabo una estimación teórica del coste del proyecto con el objetivo de valorar económicamente el esfuerzo invertido y los recursos necesarios. Esta estimación contempla tanto el uso de herramientas (hardware y software) como la dedicación personal en los distintos roles de la metodología SCRUM aplicada.

3.3.1. Hardware y software

El equipo utilizado ha sido personal, pero se estima su coste proporcional durante el periodo de desarrollo, así como el uso de herramientas de software de uso libre. Considerando el despliegue empresarial de la aplicación se consideran también un servidor cloud y un apartado adicional donde se incluyen licencias como un servicio de correo automático y uso de un servidor DNS.

Herramienta	Tipo	Coste total (€)	Vida útil (meses)	Meses usados	Coste proporcional (€)
Ordenador portátil	Hardware	1200	36	4	133,33
Visual Studio Code	Software	0	-	-	0,00
GitHub (gratuito)	Software	0	-	-	0,00
Servidor cloud (simulado)	Software	50/mes	-	4	200,00
Otros (plugins, herramientas auxiliares)	Software	100	12	4	33,33
Total estimado					**366,66**

Cuadro 3.3: Coste proporcional estimado de las herramientas

3.3.2. Recursos humanos

Aunque el desarrollo ha sido llevado a cabo íntegramente por el autor, se ha estimado un coste de recursos humanos considerando los diferentes perfiles profesionales implicados en un proyecto real, a fin de valorar el esfuerzo desde una perspectiva profesional y económica.

A continuación, se presenta un análisis actualizado de las tarifas horarias aproximadas para los perfiles de Analista, Programador, Tester y Gestor de Proyecto en España durante 2025, tanto en modalidad laboral como freelance.

Rol	**Coste por hora (€)**	**Horas estimadas**	**Coste total (€)**
Analista	30	50	1.500
Programador	25	180	4.500
Tester	20	40	800
Gestor de proyecto	35	30	1050
Total		**300**	**7.850**

Cuadro 3.4: Estimación teórica de costes de recursos humanos

3.3.3. Presupuesto total

Concepto	**Coste estimado (€)**
Herramientas (hardware y software)	366,66
Recursos humanos	7.850,00
Total	**8.216,66**

Cuadro 3.5: Presupuesto total estimado

Parte II

Documentación técnica

Capítulo 4

Análisis

4.1. Características

Se representan gráficamente en esta sección los componentes funcionales principales de la aplicación, organizados jerárquicamente desde los objetivos generales hacia funcionalidades concretas.

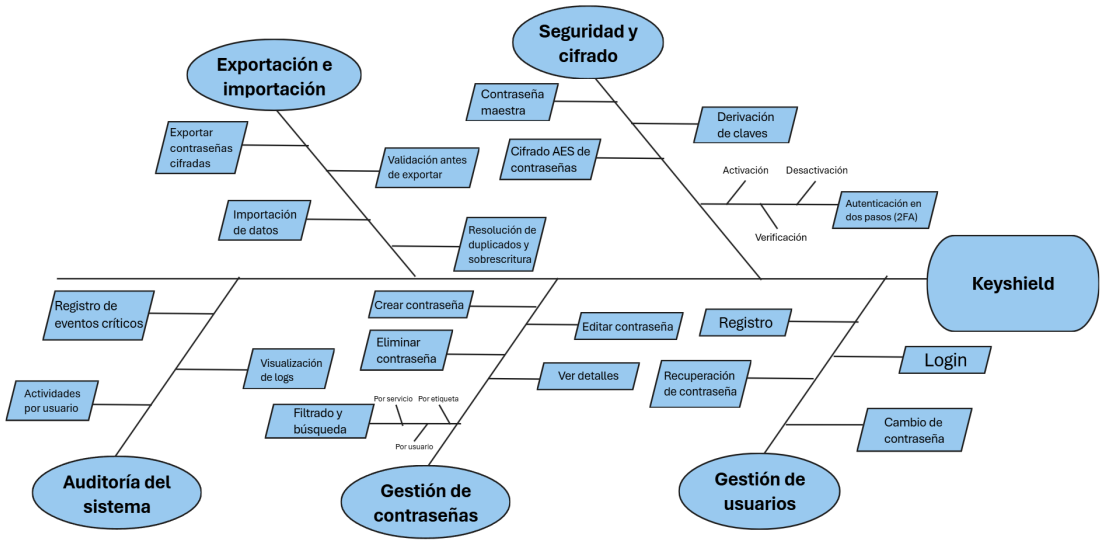


Figura 4.1: Diagrama de características

4.2. Requisitos

Para el análisis de requisitos del sistema, se ha seguido un enfoque ágil basado en User Story Mapping, una técnica visual que permite mapear la experiencia del usuario frente al sistema, estructurando funcionalidades en torno a metas de producto, épicas e historias de usuario. Del Story Mapping se obtendrán los distintos requisitos funcionales y no funcionales.

4.2.1. User Story Mapping

4.2.2. Metas

META-01 Aplicación Web Segura	
Descripción	Diseñar y desarrollar una aplicación web funcional, intuitiva y centrada en la seguridad, para la gestión eficiente de contraseñas personales o empresariales.
Épicas asociadas	▪ Épica 1.1: Diseño y desarrollo de la interfaz de usuario ▪ Épica 1.2: Arquitectura base y estructura del proyecto ▪ Épica 1.3: Gestión de sesiones y seguridad en el front-end
Prioridad	Alta
Esfuerzo estimado	40.84

Cuadro 4.1: Meta-01: Aplicación Web Segura

META-02 Cifrado AES-256	
Descripción	Integrar mecanismos de cifrado robustos (AES-256) para garantizar que las contraseñas almacenadas estén protegidas en todo momento, tanto en almacenamiento local como en la nube.
Épicas asociadas	▪ Épica 2.1: Cifrado y descifrado local de contraseñas ▪ Épica 2.2: Gestión segura de claves criptográficas ▪ Épica 2.3: Integración del cifrado con almacenamiento en la nube
Prioridad	Alta
Esfuerzo estimado	45.59

Cuadro 4.2: Meta-02: Experiencia de usuario accesible

META-03 Autenticación Segura y 2FA	
Descripción	Desarrollar un sistema de autenticación basado en contraseñas maestras fuertes y con opción de autenticación de dos factores (2FA), para reforzar la seguridad del acceso.
Épicas asociadas	▪ Épica 3.1: Autenticación con contraseña maestra ▪ Épica 3.2: Integración de autenticación en dos factores (2FA) ▪ Épica 3.3: Recuperación y restablecimiento seguro de credenciales
Prioridad	Alta
Esfuerzo estimado	43.99

Cuadro 4.3: Meta-03: Autenticación Segura y 2FA

META-04 Generador de Contraseñas Seguras	
Descripción	Proporcionar a los usuarios una herramienta integrada para generar contraseñas fuertes, personalizables según longitud, tipo de caracteres y otros parámetros.
Épicas asociadas	▪ Épica 4.1: Herramienta de generación de contraseñas personalizables ▪ Épica 4.2: Integración del generador en el flujo de gestión
Prioridad	Media
Esfuerzo estimado	35.00

Cuadro 4.4: Meta-04: Generador de Contraseñas Seguras

META-05 Organización por Etiquetas o Categorías	
Descripción	Permitir a los usuarios clasificar sus contraseñas mediante etiquetas, carpetas o categorías para una navegación más eficiente y ordenada.
Épicas asociadas	▪ Épica 5.1: Sistema de etiquetas y categorías para contraseñas ▪ Épica 5.2: Funcionalidad de búsqueda y filtrado avanzado
Prioridad	Media
Esfuerzo estimado	35.00

Cuadro 4.5: Meta-05: Organización por Etiquetas o Categorías

META-06 Escalabilidad y Almacenamiento en la Nube	
Descripción	Diseñar una arquitectura escalable que permita el almacenamiento y acceso desde la nube, manteniendo siempre el cifrado y la privacidad de los datos.
Épicas asociadas	▪ Épica 6.1: Sincronización segura con la nube ▪ Épica 6.2: Gestión de backups automáticos y restauración ▪ Épica 6.3: Acceso multiplataforma y sincronización en tiempo real
Prioridad	Media
Esfuerzo estimado	39.00

Cuadro 4.6: Meta-06: Escalabilidad y Almacenamiento en la Nube

4.2.3. Épicas

ÉPICA 1.1 - Diseño y desarrollo de la interfaz de usuario	
Descripción	Crear una interfaz intuitiva, accesible y responsive que permita gestionar contraseñas de forma sencilla y segura.
Metas asociadas	▪ Meta 1 – Aplicación Web Segura
Historias de usuario asociadas	▪ US 1.1.1: Registro de usuario ▪ US 1.1.2: Login de usuario
Prioridad	Alta
Esfuerzo estimado	15.67

Cuadro 4.7: Épica-1.1: Diseño y desarrollo de la interfaz de usuario

ÉPICA 1.2 - Arquitectura base y estructura del proyecto	
Descripción	Establecer una arquitectura modular y escalable que permita añadir funcionalidades futuras y facilite el mantenimiento.
Metas asociadas	▪ Meta 1 – Aplicación Web Segura
Historias de usuario asociadas	▪ US 1.2.1: Configuración del entorno de desarrollo ▪ US 1.2.2: Implementar estructura modular del código
Prioridad	Alta
Esfuerzo estimado	11.00

Cuadro 4.8: Épica-1.2: Arquitectura base y estructura del proyecto

ÉPICA 1.3 - Gestión de sesiones y seguridad en el front-end	
Descripción	Implementar manejo seguro de sesiones, protección contra ataques comunes (CSRF, XSS) y validaciones del lado cliente.
Metas asociadas	▪ Meta 1 – Aplicación Web Segura
Historias de usuario asociadas	▪ US 1.3.1: Gestión segura de sesión de usuario ▪ US 1.3.2: Protección contra ataques CSRF y XSS
Prioridad	Media
Esfuerzo estimado	14.17

Cuadro 4.9: Épica-1.3: Gestión de sesiones y seguridad en el front-end

ÉPICA 2.1 - Cifrado y descifrado local de contraseñas	
Descripción	Aplicar cifrado AES-256 en el cliente para asegurar que las contraseñas nunca se almacenen ni transmitan en texto plano.
Metas asociadas	▪ Meta 2 – Cifrado AES-256
Historias de usuario asociadas	▪ US 2.1.1: Cifrar contraseñas antes de almacenar ▪ US 2.1.2: Descifrar contraseñas para visualización
Prioridad	Media-Alta
Esfuerzo estimado	15.25

Cuadro 4.10: Épica-2.1: Cifrado y descifrado local de contraseñas

ÉPICA 2.2 - Gestión segura de claves criptográficas	
Descripción	Diseñar un sistema seguro para generación, almacenamiento y recuperación de claves maestras de cifrado.
Metas asociadas	▪ Meta 2 – Cifrado AES-256
Historias de usuario asociadas	▪ US 2.2.1: Generación segura de clave maestra ▪ US 2.2.2: Mecanismo seguro para cambio de contraseña maestra
Prioridad	Alta
Esfuerzo estimado	15.34

Cuadro 4.11: Épica-2.2: Gestión segura de claves criptográficas

ÉPICA 2.3 - Integración del cifrado con almacenamiento en la nube	
Descripción	Asegurar que todas las contraseñas almacenadas en la nube mantengan la confidencialidad mediante cifrado extremo a extremo.
Metas asociadas	▪ Meta 2 – Cifrado AES-256
Historias de usuario asociadas	▪ US 2.3.1: Cifrar contraseñas antes de almacenarlas ▪ US 2.3.2: Descifrar contraseñas tras autenticación exitosa
Prioridad	Media-Alta
Esfuerzo estimado	15.00

Cuadro 4.12: Épica-2.3: Integración del cifrado con almacenamiento en la nube

ÉPICA 3.1 - Autenticación con contraseña maestra	
Descripción	Implementar el sistema principal de autenticación con contraseña maestra, con validaciones de fuerza y protección contra ataques.
Metas asociadas	▪ Meta 3 – Autenticación Segura y 2FA
Historias de usuario asociadas	▪ US 3.1.1: Crear cuenta con contraseña maestra segura ▪ US 3.1.2: Iniciar sesión con contraseña maestra
Prioridad	Alta
Esfuerzo estimado	16.67

Cuadro 4.13: Épica-3.1: Autenticación con contraseña maestra

ÉPICA 3.2 - Integración de autenticación en dos factores (2FA)	
Descripción	Añadir soporte para 2FA mediante TOTP o email/sms para reforzar la seguridad en el acceso.
Metas asociadas	▪ Meta 3 – Autenticación Segura y 2FA
Historias de usuario asociadas	▪ US 3.2.1: Activar autenticación de dos factores (2FA) ▪ US 3.2.2: Verificar código 2FA en el login
Prioridad	Alta
Esfuerzo estimado	14.66

Cuadro 4.14: Épica-3.2: Integración de autenticación en dos factores (2FA)

ÉPICA 3.3 - Recuperación y restablecimiento seguro de credenciales	
Descripción	Crear mecanismos seguros para recuperar el acceso al sistema en caso de olvido de la contraseña maestra o pérdida de 2FA.
Metas asociadas	▪ Meta 3 – Autenticación Segura y 2FA
Historias de usuario asociadas	▪ US 3.3.1: Recuperación de contraseña maestra mediante correo electrónico ▪ US 3.3.2: Restablecimiento seguro de 2FA
Prioridad	Media
Esfuerzo estimado	12.66

Cuadro 4.15: Épica-3.3: Recuperación y restablecimiento seguro de credenciales

ÉPICA 4.1 - Herramienta de generación de contraseñas personalizables	
Descripción	Permitir generar contraseñas aleatorias con opciones para longitud, tipos de caracteres y complejidad.
Metas asociadas	▪ Meta 4 – Generador de Contraseñas Seguras
Historias de usuario asociadas	▪ US 4.1.1: Generar contraseña aleatoria con parámetros
Prioridad	Media
Esfuerzo estimado	9.50

Cuadro 4.16: Épica-4.1: Herramienta de generación de contraseñas personalizables

ÉPICA 4.2 - Integración del generador en el flujo de gestión	
Descripción	Facilitar el uso del generador al añadir o modificar contraseñas, con funcionalidades como copiar al portapapeles.
Metas asociadas	▪ Meta 4 – Generador de Contraseñas Seguras
Historias de usuario asociadas	▪ US 4.2.1: Usar contraseña generada al crear nueva entrada ▪ US 4.2.2: Copiar contraseña generada al portapapeles
Prioridad	Media
Esfuerzo estimado	25.50

Cuadro 4.17: Épica-4.2: Integración del generador en el flujo de gestión

ÉPICA 5.1 - Sistema de etiquetas y categorías para contraseñas	
Descripción	Permitir a los usuarios organizar sus contraseñas mediante etiquetas o categorías personalizadas.
Metas asociadas	▪ Meta 5 – Organización por Etiquetas o Categorías
Historias de usuario asociadas	▪ US 5.1.1: Crear y asignar etiquetas a contraseñas ▪ US 5.1.2: Filtrar contraseñas por etiquetas o categorías
Prioridad	Media
Esfuerzo estimado	16.00

Cuadro 4.18: Épica-5.1: Sistema de etiquetas y categorías para contraseñas

ÉPICA 5.2 - Funcionalidad de búsqueda y filtrado avanzado	
Descripción	Implementar búsqueda eficiente y filtros por etiquetas, categorías, nombre o fecha para mejorar la usabilidad.
Metas asociadas	▪ Meta 5 – Organización por Etiquetas o Categorías
Historias de usuario asociadas	▪ US 5.2.1: Buscar contraseñas por etiquetas y nombre ▪ US 5.2.2: Filtrar contraseñas por categoría y fecha
Prioridad	Media
Esfuerzo estimado	19.00

Cuadro 4.19: Épica-5.2: Funcionalidad de búsqueda y filtrado avanzado

ÉPICA 6.1 - Sincronización segura con la nube	
Descripción	Permitir la sincronización de contraseñas cifradas en servidores seguros, para acceso multi-dispositivo.
Metas asociadas	▪ Meta 6 – Escalabilidad y Almacenamiento en la Nube
Historias de usuario asociadas	▪ US 6.1.1: Subir contraseñas cifradas a la nube ▪ US 6.1.2: Descargar contraseñas cifradas y descifrarlas localmente
Prioridad	Media
Esfuerzo estimado	12.00

Cuadro 4.20: Épica-6.1: Sincronización segura con la nube

ÉPICA 6.2 - Gestión de backups automáticos y restauración	
Descripción	Crear un sistema de copias de seguridad automáticas y recuperación sencilla de datos cifrados.
Metas asociadas	▪ Meta 6 – Escalabilidad y Almacenamiento en la Nube
Historias de usuario asociadas	▪ US 6.2.1: Backup automático de contraseñas cifradas ▪ US 6.2.2: Restaurar contraseñas desde backup
Prioridad	Media
Esfuerzo estimado	14.00

Cuadro 4.21: Épica-6.2: Gestión de backups automáticos y restauración

ÉPICA 6.3 - Acceso multiplataforma y sincronización en tiempo real	
Descripción	Facilitar el uso simultáneo de la aplicación en distintos dispositivos con sincronización continua y segura.
Metas asociadas	▪ Meta 6 – Escalabilidad y Almacenamiento en la Nube
Historias de usuario asociadas	▪ US 6.3.1: Control de acceso basado en roles para usuarios empresariales ▪ US 6.3.2: Auditoría y registro de actividades de usuario
Prioridad	Baja
Esfuerzo estimado	13.00

Cuadro 4.22: Épica-6.3: Acceso multiplataforma y sincronización en tiempo real

4.2.4. Historias de usuario

US 1.1.1 - Registro de usuario	
Descripción	Como usuario, quiero registrarme para gestionar mis contraseñas.
Metas asociadas	▪ Meta 1 – Aplicación Web Segura
Épicas asociadas	▪ ÉPICA 1.1 – Interfaz de usuario
Tareas	1. Crear formulario de registro. 2. Validación de email y contraseña. 3. Enviar datos al backend.
Pruebas de aceptación	▪ Registro válido. ▪ Mensajes de error claros. ▪ No se permite email duplicado.
Prioridad	Alta
Esfuerzo estimado	7.00

Cuadro 4.23: US 1.1.1 - Registro de usuario

US 1.1.2 - Login de usuario	
Descripción	Como usuario, quiero iniciar sesión para acceder a mis datos.
Metas asociadas	▪ Meta 1 – Aplicación Web Segura
Épicas asociadas	▪ ÉPICA 1.1 – Interfaz de usuario
Tareas	1. Crear formulario de login. 2. Validar campos. 3. Autenticar y redirigir. 4. Manejar sesión.
Pruebas de aceptación	▪ Login con credenciales correctas. ▪ Acceso bloqueado si son incorrectas. ▪ Sesión persistente.
Prioridad	Alta
Esfuerzo estimado	8.67

Cuadro 4.24: US 1.1.2 - Login de usuario

US 1.2.1 - Configuración del entorno de desarrollo	
Descripción	Como desarrollador, quiero un entorno preconfigurado con frameworks y librerías necesarias.
Metas asociadas	▪ Meta 1 – Aplicación Web Segura
Épicas asociadas	▪ ÉPICA 1.2 – Arquitectura base del proyecto
Tareas	1. Elegir stack tecnológico. 2. Configurar control de versiones. 3. Crear scripts de build y despliegue.
Pruebas de aceptación	▪ Entorno funcional y reproducible. ▪ Proyecto se compila sin errores.
Prioridad	Alta
Esfuerzo estimado	6.00

Cuadro 4.25: US 1.2.1 - Configuración del entorno de desarrollo

US 1.2.2 - Estructura modular del código	
Descripción	Como desarrollador, quiero una estructura modular clara para mantener el proyecto organizado.
Metas asociadas	▪ Meta 1 – Aplicación Web Segura
Épicas asociadas	▪ ÉPICA 1.2 – Arquitectura base del proyecto
Tareas	1. Definir módulos y carpetas. 2. Implementar ejemplo básico. 3. Documentar estructura y convenciones.
Pruebas de aceptación	▪ Código navegable y organizado. ▪ Nuevos desarrolladores entienden estructura rápidamente.
Prioridad	Alta
Esfuerzo estimado	5.00

Cuadro 4.26: US 1.2.2 - Estructura modular del código

US 1.3.1 - Gestión segura de sesión de usuario	
Descripción	Como usuario, quiero que mi sesión sea segura durante la interacción para evitar accesos no autorizados.
Metas asociadas	▪ Meta 1 – Aplicación Web Segura
Épicas asociadas	▪ ÉPICA 1.3 – Gestión de sesiones y seguridad en el front-end
Tareas	1. Implementar tokens JWT para sesiones. 2. Establecer expiración y renovación automática. 3. Usar almacenamiento seguro (cookies HttpOnly o local).
Pruebas de aceptación	▪ La sesión expira tras inactividad. ▪ Sin token válido no hay acceso. ▪ Renovación funciona sin interrupciones.
Prioridad	Media
Esfuerzo estimado	7.67

Cuadro 4.27: US 1.3.1 - Gestión segura de sesión de usuario

US 1.3.2 - Protección contra ataques CSRF y XSS	
Descripción	Como usuario, quiero estar protegido contra ataques comunes como CSRF y XSS.
Metas asociadas	▪ Meta 1 – Aplicación Web Segura
Épicas asociadas	▪ ÉPICA 1.3 – Gestión de sesiones y seguridad en el front-end
Tareas	1. Implementar tokens CSRF en formularios. 2. Validar y sanitizar entradas del usuario. 3. Testear contra XSS y CSRF.
Pruebas de aceptación	▪ Pruebas de penetración exitosas. ▪ Formularios previenen inyecciones XSS.
Prioridad	Media
Esfuerzo estimado	6.50

Cuadro 4.28: US 1.3.2 - Protección contra ataques CSRF y XSS

US 2.1.1 - Cifrar contraseñas antes de almacenar	
Descripción	Como usuario, quiero que mis contraseñas se cifren antes de guardarse para proteger su confidencialidad.
Metas asociadas	▪ Meta 2 – Cifrado AES-256
Épicas asociadas	▪ ÉPICA 2.1 – Cifrado y descifrado local de contraseñas
Tareas	1. Implementar cifrado AES-256 en cliente. 2. Integrar cifrado al guardar contraseñas. 3. Validar almacenamiento cifrado.
Pruebas de aceptación	▪ No se almacenan contraseñas en texto plano. ▪ Las contraseñas pueden descifrarse correctamente tras autenticación.
Prioridad	Alta
Esfuerzo estimado	7.25

Cuadro 4.29: US 2.1.1 - Cifrado de contraseñas antes de almacenar

US 2.1.2 - Descifrar contraseñas para visualización	
Descripción	Como usuario autenticado, quiero ver mis contraseñas descifradas para usarlas cuando las necesite.
Metas asociadas	▪ Meta 2 – Cifrado AES-256
Épicas asociadas	▪ ÉPICA 2.1 – Cifrado y descifrado local de contraseñas
Tareas	1. Implementar descifrado tras autenticación. 2. Integrar cifrado al guardar contraseñas. 3. Mostrar contraseñas solo tras verificación del usuario.
Pruebas de aceptación	▪ Solo usuarios autenticados ven contraseñas descifradas. ▪ Las contraseñas descifradas coinciden con las originales.
Prioridad	Alta
Esfuerzo estimado	8.00

Cuadro 4.30: US 2.1.2 - Descifrado para visualización de contraseñas

US 2.2.1 - Generación segura de clave maestra	
Descripción	Como usuario, quiero que la clave maestra se genere y almacene de forma segura para proteger mis datos.
Metas asociadas	▪ Meta 2 – Cifrado AES-256
Épicas asociadas	▪ ÉPICA 2.2: Gestión segura de claves criptográficas
Tareas	1. Implementar algoritmo seguro para generación de clave. 2. Almacenar la clave localmente de forma cifrada. 3. Validar recuperación segura en sesiones posteriores.
Pruebas de aceptación	▪ La clave no es accesible en texto plano. ▪ La clave se recupera correctamente sin pérdida ni exposición.
Prioridad	Alta
Esfuerzo estimado	7.67

Cuadro 4.31: US 2.2.1 - Generación segura de clave maestra

US 2.2.2 - Mecanismo seguro para cambio de contraseña maestra	
Descripción	Como usuario, quiero poder cambiar mi contraseña maestra sin comprometer la seguridad de mis datos almacenados.
Metas asociadas	▪ Meta 2 – Cifrado AES-256
Épicas asociadas	▪ ÉPICA 2.2: Gestión segura de claves criptográficas
Tareas	1. Implementar flujo para cambio seguro de contraseña maestra. 2. Re-encriptar contraseñas almacenadas con la nueva clave. 3. Validar integridad tras cambio.
Pruebas de aceptación	▪ Cambio de contraseña es efectivo y seguro. ▪ Contraseñas previas siguen accesibles tras cambio.
Prioridad	Alta
Esfuerzo estimado	7.67

Cuadro 4.32: US 2.2.2 - Mecanismo seguro para cambio de contraseña maestra

US 2.3.1 - Cifrar contraseñas antes de almacenarlas	
Descripción	Como usuario, quiero que todas mis contraseñas se cifren con AES-256 antes de ser almacenadas para garantizar su seguridad.
Metas asociadas	▪ Meta 2 – Cifrado AES-256
Épicas asociadas	▪ ÉPICA 2.3: Integración del cifrado en el flujo de almacenamiento y recuperación
Tareas	1. Implementar función de cifrado AES-256 en el backend o frontend según arquitectura. 2. Modificar flujo de almacenamiento para cifrar datos antes de guardarlos. 3. Validar que no se almacenan contraseñas en texto plano.
Pruebas de aceptación	▪ Contraseñas se almacenan siempre cifradas. ▪ No es posible acceder a contraseñas sin el proceso de descifrado.
Prioridad	Alta
Esfuerzo estimado	7.00

Cuadro 4.33: US 2.3.1 - Cifrar contraseñas antes de almacenarlas

US 2.3.2 - Descifrar contraseñas tras autenticación exitosa	
Descripción	Como usuario, quiero que mis contraseñas se descifren solo después de autenticarme correctamente para asegurar que nadie más pueda acceder a ellas.
Metas asociadas	▪ Meta 2 – Cifrado AES-256
Épicas asociadas	▪ ÉPICA 2.3: Integración del cifrado en el flujo de almacenamiento y recuperación
Tareas	1. Implementar función de descifrado AES-256 tras validación del usuario. 2. Asegurar que la clave de descifrado sólo está en memoria temporalmente. 3. Probar flujo completo de almacenamiento y recuperación.
Pruebas de aceptación	▪ Contraseñas aparecen descifradas solo tras autenticación válida. ▪ No queda información descifrada persistente tras sesión.
Prioridad	Alta
Esfuerzo estimado	8.00

Cuadro 4.34: US 2.3.2 - Descifrar contraseñas tras autenticación exitosa

US 3.1.1 - Crear cuenta con contraseña maestra segura	
Descripción	Como nuevo usuario, quiero registrarme en la aplicación utilizando una contraseña maestra robusta, para asegurar que mi cuenta sea difícil de comprometer.
Metas asociadas	▪ Meta 3 – Autenticación Segura y 2FA
Épicas asociadas	▪ ÉPICA 3.1: Autenticación mediante contraseña maestra segura
Tareas	1. Diseñar formulario de registro con validación de contraseña fuerte. 2. Establecer reglas de complejidad (mínimo de caracteres, mayúsculas, símbolos, etc.). 3. Hash seguro de la contraseña maestra antes de almacenamiento.
Pruebas de aceptación	▪ El sistema rechaza contraseñas débiles. ▪ La contraseña se almacena hasheada (no en texto plano). ▪ Usuario puede completar registro correctamente con una contraseña válida.
Prioridad	Alta
Esfuerzo estimado	8.00

Cuadro 4.35: US 3.1.1 - Crear cuenta con contraseña maestra segura

US 3.1.2 - Iniciar sesión con contraseña maestra	
Descripción	Como usuario, quiero poder acceder a mi cuenta mediante la contraseña maestra para que mis datos se mantengan protegidos.
Metas asociadas	▪ Meta 3 – Autenticación Segura y 2FA
Épicas asociadas	▪ ÉPICA 3.1: Autenticación mediante contraseña maestra segura
Tareas	1. Crear formulario de login que valide la contraseña maestra. 2. Verificar hash en servidor y autorizar acceso solo si coincide. 3. Limitar intentos fallidos e implementar medidas antifuerza bruta.
Pruebas de aceptación	▪ El login funciona solo si la contraseña es correcta. ▪ Fallos consecutivos provocan bloqueo temporal. ▪ El sistema nunca muestra si el usuario o contraseña es incorrecto (por separado), para no dar pistas.
Prioridad	Alta
Esfuerzo estimado	8.67

Cuadro 4.36: US 3.1.2 - Iniciar sesión con contraseña maestra

US 3.2.1 - Activar autenticación en dos factores	
Descripción	Como usuario, quiero poder activar 2FA para añadir una capa extra de seguridad en mi cuenta.
Metas asociadas	▪ Meta 3 – Autenticación Segura y 2FA
Épicas asociadas	▪ ÉPICA 3.2: Implementación y gestión de autenticación multifactor (2FA)
Tareas	1. Crear opción para activar 2FA en ajustes de usuario. 2. Implementar generación de códigos TOTP y escaneo QR. 3. Validar códigos de 2FA en login.
Pruebas de aceptación	▪ Usuario puede activar y configurar 2FA correctamente. ▪ Login requiere código 2FA cuando está activado. ▪ Código 2FA es rechazado si es incorrecto o expirado.
Prioridad	Alta
Esfuerzo estimado	7.33

Cuadro 4.37: US 3.2.1 - Activar autenticación en dos factores

US 3.2.2 - Verificar código 2FA en el login	
Descripción	Como usuario, quiero ingresar el código de 2FA tras introducir mi contraseña para completar la autenticación.
Metas asociadas	▪ Meta 3 – Autenticación Segura y 2FA
Épicas asociadas	▪ ÉPICA 3.2: Implementación y gestión de autenticación multifactor (2FA)
Tareas	1. Implementar lógica de verificación de códigos 2FA en backend. 2. Crear pantalla/interfaz para ingresar código tras autenticación inicial. 3. Manejar errores y códigos expirados.
Pruebas de aceptación	▪ Solo se permite acceso con código 2FA válido. ▪ Mensajes claros ante códigos inválidos o expirados.
Prioridad	Alta
Esfuerzo estimado	7.33

Cuadro 4.38: US 3.2.2 - Verificar código 2FA en el login

US 3.3.1 - Recuperación de contraseña maestra mediante correo electrónico	
Descripción	Como usuario, quiero recuperar el acceso a mi cuenta si olvido mi contraseña maestra a través de un enlace seguro enviado a mi correo.
Metas asociadas	▪ Meta 3 – Autenticación Segura y 2FA
Épicas asociadas	▪ ÉPICA 3.3: Recuperación y restablecimiento seguro de credenciales
Tareas	1. Implementar sistema para solicitar recuperación mediante email. 2. Generar y enviar enlace temporal seguro. 3. Permitir restablecer contraseña tras validación.
Pruebas de aceptación	▪ El enlace funciona solo una vez y expira. ▪ El usuario puede restablecer contraseña y acceder con la nueva.
Prioridad	Media
Esfuerzo estimado	6.33

Cuadro 4.39: US 3.3.1 - Recuperación de contraseña maestra mediante correo electrónico

US 3.3.2 - Restablecimiento seguro de 2FA	
Descripción	Como usuario, quiero poder restablecer o desactivar 2FA en caso de pérdida de acceso a mi método de autenticación.
Metas asociadas	▪ Meta 3 – Autenticación Segura y 2FA
Épicas asociadas	▪ ÉPICA 3.3: Recuperación y restablecimiento seguro de credenciales
Tareas	1. Implementar proceso de verificación alternativo para restablecer 2FA. 2. Añadir soporte para contacto con soporte o preguntas de seguridad.
Pruebas de aceptación	▪ Usuario puede restablecer 2FA tras pasar verificación. ▪ Seguridad no se ve comprometida durante el proceso.
Prioridad	Media
Esfuerzo estimado	6.33

Cuadro 4.40: US 3.3.2 - Restablecimiento seguro de 2FA

US 4.1.1 - Generar contraseña aleatoria con parámetros	
Descripción	Como usuario, quiero generar contraseñas con longitud y tipos de caracteres configurables para aumentar la seguridad.
Metas asociadas	▪ Meta 4 – Generador de Contraseñas Seguras
Épicas asociadas	▪ ÉPICA 4.1: Herramienta de generación de contraseñas personalizables
Tareas	1. Crear interfaz para definir parámetros (longitud, símbolos, números, mayúsculas). 2. Implementar lógica para generar contraseñas según parámetros. 3. Mostrar contraseña generada con opción de copiar.
Pruebas de aceptación	▪ Contraseña generada cumple con los parámetros seleccionados. ▪ Se puede copiar fácilmente al portapapeles.
Prioridad	Media
Esfuerzo estimado	9.50

Cuadro 4.41: US 4.1.1 - Generar contraseña aleatoria con parámetros

US 4.2.1 - Usar contraseña generada al crear nueva entrada	
Descripción	Como usuario, quiero poder generar una contraseña segura y asignarla directamente al crear una nueva contraseña en el sistema.
Metas asociadas	▪ Meta 4 – Generador de Contraseñas Seguras
Épicas asociadas	▪ ÉPICA 4.2: Integración del generador en el flujo de gestión
Tareas	1. Añadir botón “Generar contraseña” en formulario de creación. 2. Permitir seleccionar y modificar parámetros antes de asignar. 3. Copiar contraseña generada automáticamente al campo correspondiente.
Pruebas de aceptación	▪ La contraseña generada se asigna correctamente al formulario. ▪ Usuario puede modificar parámetros y generar nuevas contraseñas fácilmente.
Prioridad	Media
Esfuerzo estimado	9.50

Cuadro 4.42: US 4.2.1 - Usar contraseña generada al crear nueva entrada

US 4.2.2 - Copiar contraseña generada al portapapeles	
Descripción	Como usuario, quiero copiar rápidamente la contraseña generada para usarla en otras aplicaciones si lo deseo.
Metas asociadas	▪ Meta 4 – Generador de Contraseñas Seguras
Épicas asociadas	▪ ÉPICA 4.2: Integración del generador en el flujo de gestión
Tareas	1. Implementar funcionalidad para copiar al portapapeles con un clic. 2. Mostrar confirmación visual de copiado exitoso.
Pruebas de aceptación	▪ La contraseña se copia correctamente. ▪ Se muestra feedback claro al usuario.
Prioridad	Media
Esfuerzo estimado	16.00

Cuadro 4.43: US 4.2.2 - Copiar contraseña generada al portapapeles

US 5.1.1 - Crear y asignar etiquetas a contraseñas	
Descripción	Como usuario, quiero poder crear etiquetas personalizadas y asignarlas a mis contraseñas para organizarlas mejor.
Metas asociadas	▪ Meta 5 – Organización por Etiquetas o Categorías
Épicas asociadas	▪ ÉPICA 5.1: Sistema de etiquetas y categorías para contraseñas
Tareas	1. Crear interfaz para añadir/editar etiquetas. 2. Añadir funcionalidad para asignar etiquetas al crear o modificar contraseñas.
Pruebas de aceptación	▪ Usuario puede crear nuevas etiquetas y asignarlas a contraseñas. ▪ Etiquetas se muestran correctamente junto a cada contraseña.
Prioridad	Media
Esfuerzo estimado	8.00

Cuadro 4.44: US 5.1.1 - Crear y asignar etiquetas a contraseñas

US 5.1.2 - Filtrar contraseñas por etiquetas o categorías	
Descripción	Como usuario, quiero poder filtrar mis contraseñas usando etiquetas o categorías, para encontrar más fácilmente las que necesito según su contexto.
Metas asociadas	▪ Meta 5 – Organización por Etiquetas o Categorías
Épicas asociadas	▪ ÉPICA 5.1: Sistema de etiquetas y categorías para contraseñas
Tareas	1. Implementar sistema de filtrado en el frontend por etiquetas/categorías. 2. Añadir búsqueda combinada por varias etiquetas. 3. Validar que el filtrado no afecta la seguridad ni rendimiento. 4. Ajustar la interfaz para que sea clara y fácil de usar.
Pruebas de aceptación	▪ El usuario puede seleccionar una o más etiquetas para filtrar. ▪ Solo se muestran contraseñas que coinciden con el filtro. ▪ El filtro funciona correctamente incluso con muchas contraseñas. ▪ No se muestran datos de otras categorías no seleccionadas.
Prioridad	Media
Esfuerzo estimado	8.00

Cuadro 4.45: US 5.1.2 - Filtrar contraseñas por etiquetas o categorías

US 5.2.1 - Buscar contraseñas por etiquetas y nombre	
Descripción	Como usuario, quiero buscar contraseñas usando etiquetas y nombres para encontrar la que necesito rápidamente.
Metas asociadas	▪ Meta 5 – Organización por Etiquetas o Categorías
Épicas asociadas	▪ ÉPICA 5.2: Funcionalidad de búsqueda y filtrado avanzado
Tareas	1. Implementar barra de búsqueda con soporte para etiquetas. 2. Mostrar resultados en tiempo real con filtros aplicados.
Pruebas de aceptación	▪ Los resultados coinciden con los criterios de búsqueda. ▪ La búsqueda es rápida y eficiente.
Prioridad	Media
Esfuerzo estimado	9.50

Cuadro 4.46: US 5.2.1 - Buscar contraseñas por etiquetas y nombre

US 5.2.2 - Filtrar contraseñas por categoría y fecha	
Descripción	Como usuario, quiero aplicar filtros por categorías y fecha de creación/modificación para organizar mejor mis contraseñas.
Metas asociadas	▪ Meta 5 – Organización por Etiquetas o Categorías
Épicas asociadas	▪ ÉPICA 5.2: Funcionalidad de búsqueda y filtrado avanzado
Tareas	1. Añadir filtros desplegables para categorías y fecha. 2. Integrar filtros con la vista de listado.
Pruebas de aceptación	▪ El filtrado muestra solo las contraseñas que cumplen criterios. ▪ La interfaz es fácil de usar y responde rápidamente.
Prioridad	Media
Esfuerzo estimado	9.50

Cuadro 4.47: US 5.2.2 - Filtrar contraseñas por categoría y fecha

US 6.1.1 - Subir contraseñas cifradas a la nube	
Descripción	Como usuario, quiero que mis contraseñas se sincronicen automáticamente con la nube de forma segura para acceder desde cualquier dispositivo.
Metas asociadas	▪ Meta 6 – Escalabilidad y Almacenamiento en la Nube
Épicas asociadas	▪ ÉPICA 6.1: Sincronización segura con la nube
Tareas	1. Implementar API para subida de datos cifrados. 2. Integrar subida automática tras cambios locales. 3. Manejar errores y reconexión automática.
Pruebas de aceptación	▪ Contraseñas se suben cifradas correctamente y sin pérdida. ▪ Sincronización se realiza tras cada cambio sin intervención manual.
Prioridad	Media
Esfuerzo estimado	6.00

Cuadro 4.48: US 6.1.1 - Subir contraseñas cifradas a la nube

US 6.1.2 - Descargar contraseñas cifradas y descifrarlas localmente	
Descripción	Como usuario, quiero que mis contraseñas se descarguen cifradas desde la nube y se descifren solo en mi dispositivo para mantener la privacidad.
Metas asociadas	▪ Meta 6 – Escalabilidad y Almacenamiento en la Nube
Épicas asociadas	▪ ÉPICA 6.1: Sincronización segura con la nube
Tareas	1. Implementar API para descarga de datos cifrados. 2. Integrar descifrado local tras descarga. 3. Mostrar información sincronizada en la interfaz.
Pruebas de aceptación	▪ Contraseñas se descargan y descifran correctamente solo con usuario autenticado. ▪ Los datos nunca se almacenan descifrados en la nube.
Prioridad	Media
Esfuerzo estimado	6.00

Cuadro 4.49: US 6.1.2 - Descargar contraseñas cifradas y descifrarlas localmente

US 6.2.1 - Backup automático de contraseñas cifradas	
Descripción	Como usuario, quiero que la aplicación haga copias de seguridad automáticas de mis datos cifrados para evitar pérdidas.
Metas asociadas	▪ Meta 6 – Escalabilidad y Almacenamiento en la Nube
Épicas asociadas	▪ ÉPICA 6.2: Gestión de backups automáticos y restauración
Tareas	1. Implementar proceso automático de backup periódico. 2. Asegurar que los backups están cifrados y almacenados seguros.
Pruebas de aceptación	▪ Se generan backups automáticos sin intervención del usuario. ▪ Los backups pueden ser restaurados correctamente.
Prioridad	Media
Esfuerzo estimado	7.00

Cuadro 4.50: US 6.2.1 - Backup automático de contraseñas cifradas

US 6.2.2 - Restaurar contraseñas desde backup	
Descripción	Como usuario, quiero restaurar mis contraseñas desde un backup para recuperar mi información en caso de pérdida.
Metas asociadas	▪ Meta 6 – Escalabilidad y Almacenamiento en la Nube
Épicas asociadas	▪ ÉPICA 6.2: Gestión de backups automáticos y restauración
Tareas	1. Crear interfaz para seleccionar y restaurar backup. 2. Validar integridad y autenticidad de datos restaurados.
Pruebas de aceptación	▪ Restauración es exitosa y sin pérdida de datos. ▪ Solo usuario autorizado puede realizar restauraciones.
Prioridad	Media
Esfuerzo estimado	7.00

Cuadro 4.51: US 6.2.2 - Restaurar contraseñas desde backup

US 6.3.1 - Control de acceso basado en roles para usuarios empresariales	
Descripción	Como administrador de una empresa, quiero definir roles y permisos para los empleados que usen Keyshield, para controlar qué información pueden ver o modificar.
Metas asociadas	▪ Meta 6 – Escalabilidad y Almacenamiento en la Nube
Épicas asociadas	▪ ÉPICA 6.3: Seguridad y gestión de permisos en la nube
Tareas	1. Diseñar sistema de roles (admin, usuario estándar, auditor, etc.). 2. Implementar lógica de permisos para cada rol. 3. Crear interfaz para asignar roles a usuarios.
Pruebas de aceptación	▪ Usuarios solo pueden realizar acciones permitidas por su rol. ▪ Cambios de rol se aplican en tiempo real.
Prioridad	Alta
Esfuerzo estimado	6.00

Cuadro 4.52: US 6.3.1 - Control de acceso basado en roles para usuarios empresariales

US 6.3.2 - Auditoría y registro de actividades de usuario	
Descripción	Como administrador, quiero que Keyshield registre las acciones relevantes realizadas por usuarios para auditar y detectar posibles problemas de seguridad.
Metas asociadas	▪ Meta 6 – Escalabilidad y Almacenamiento en la Nube
Épicas asociadas	▪ ÉPICA 6.3: Seguridad y gestión de permisos en la nube
Tareas	1. Implementar sistema de logs para eventos críticos (login, cambios de contraseña, modificación de permisos). 2. Crear panel de auditoría accesible para administradores.
Pruebas de aceptación	▪ Los eventos quedan registrados con fecha, hora y usuario. ▪ Los administradores pueden consultar los registros fácilmente.
Prioridad	Media
Esfuerzo estimado	7.00

Cuadro 4.53: US 6.3.2 - Auditoría y registro de actividades de usuario

4.2.5. Backlog priorizado

ID-US	Valor	Criticidad	Oportunidad	CoD	Duración	WSJF	Posición
US 4.2.2	6	5	5	16	1	16.00	1
US 4.1.1	7	6	6	19	2	9.50	2
US 4.2.1	7	6	6	19	2	9.50	3
US 5.2.1	7	6	6	19	2	9.50	4
US 5.2.2	7	6	6	19	2	9.50	5
US 1.1.2	9	9	8	26	3	8.67	6
US 3.1.2	9	9	8	26	3	8.67	7
US 2.1.2	9	8	7	24	3	8.00	8
US 2.3.2	9	8	7	24	3	8.00	9
US 3.1.1	9	8	7	24	3	8.00	10
US 5.1.1	6	5	5	16	2	8.00	11
US 5.1.2	6	5	5	16	2	8.00	12
US 1.3.1	8	8	7	23	3	7.67	13
US 2.2.1	9	7	7	23	3	7.67	14
US 2.2.2	9	7	7	23	3	7.67	15
US 3.2.1	8	7	7	22	3	7.33	16
US 3.2.2	8	7	7	22	3	7.33	17
US 2.1.1	10	10	9	29	4	7.25	18
US 1.1.1	8	7	6	21	3	7.00	19
US 2.3.1	10	9	9	28	4	7.00	20
US 6.2.1	8	7	6	21	3	7.00	21
US 6.2.2	8	7	6	21	3	7.00	22
US 6.3.2	8	7	6	21	3	7.00	23
US 1.2.1	5	3	4	12	2	6.00	24
US 6.1.1	9	8	7	24	4	6.00	25
US 6.1.2	9	8	7	24	4	6.00	26
US 6.3.1	9	8	7	24	4	6.00	27
US 3.3.1	7	6	6	19	3	6.33	28
US 3.3.2	7	6	6	19	3	6.33	29
US 1.2.2	6	4	5	15	3	5.00	30
US 1.3.2	9	9	8	26	4	6.50	31

Cuadro 4.54: Backlog ordenado por prioridad usando WSJF

4.2.6. Requisitos Funcionales

En este apartado se describen los requisitos del sistema que se va a desarrollar: requisitos funcionales, de interfaz de usuario, de información, etc. Asimismo, se especificará si la aplicación debe ser o no multiplataforma (o multilingüe), si se establecen restricciones de uso o de otro tipo (por ejemplo heredadas del entorno organizativo en el que se integrará), etc.

ID-RF	Descripción del requisito funcional	US relacionadas
RF-01	El sistema debe permitir a un nuevo usuario registrarse proporcionando una contraseña maestra segura.	US 1.1.1
RF-02	El sistema debe permitir a los usuarios autenticarse mediante su contraseña maestra.	US 1.1.2, US 3.1.2
RF-03	El sistema debe proteger las sesiones de usuario mediante tokens seguros y técnicas contra CSRF y XSS.	US 1.3.1, US 1.3.2
RF-04	El sistema debe cifrar las contraseñas utilizando el algoritmo AES-256 antes de almacenarlas.	US 2.1.1, US 2.3.1
RF-05	El sistema debe permitir descifrar contraseñas solo tras una autenticación exitosa.	US 2.1.2, US 2.3.2
RF-06	El sistema debe permitir generar una contraseña maestra segura durante el registro.	US 2.2.1
RF-07	El sistema debe permitir cambiar la contraseña maestra de forma segura.	US 2.2.2
RF-08	El sistema debe permitir activar y verificar un sistema de autenticación en dos factores (2FA).	US 3.2.1, US 3.2.2
RF-09	El sistema debe permitir recuperar la contraseña maestra mediante correo electrónico.	US 3.3.1
RF-10	El sistema debe permitir restablecer el 2FA de manera segura.	US 3.3.2
RF-11	El sistema debe permitir generar contraseñas aleatorias configurando longitud y tipo de caracteres.	US 4.1.1
RF-12	El sistema debe permitir usar contraseñas generadas al crear nuevas entradas.	US 4.2.1
RF-13	El sistema debe permitir copiar una contraseña generada al portapapeles.	US 4.2.2
RF-14	El sistema debe permitir crear y asignar etiquetas o categorías a las contraseñas.	US 5.1.1
RF-15	El sistema debe permitir filtrar contraseñas por etiquetas o categorías.	US 5.1.2
RF-16	El sistema debe permitir buscar contraseñas por nombre o etiquetas.	US 5.2.1
RF-17	El sistema debe permitir filtrar contraseñas por categoría y fecha de creación/modificación.	US 5.2.2
RF-18	El sistema debe subir contraseñas cifradas a la nube de forma segura.	US 6.1.1
RF-19	El sistema debe descargar contraseñas cifradas y descifrarlas localmente.	US 6.1.2
RF-20	El sistema debe realizar backups automáticos de las contraseñas cifradas.	US 6.2.1
RF-21	El sistema debe restaurar contraseñas desde un backup previo.	US 6.2.2

Tabla continua		
ID-RF	**Descripción del requisito funcional**	**US relacionadas**
RF-22	El sistema debe implementar control de acceso basado en roles para usuarios empresariales.	US 6.3.1
RF-23	El sistema debe registrar las actividades del usuario para su auditoría.	US 6.3.2
RF-24	El sistema debe validar los datos introducidos por el usuario en todos los formularios.	-
RF-25	El sistema debe bloquear el acceso tras múltiples intentos fallidos de autenticación.	-
RF-26	El sistema debe cerrar la sesión automáticamente tras un período de inactividad.	-
RF-27	El sistema debe permitir al usuario modificar su información de perfil.	-
RF-28	El sistema debe permitir mostrar y ocultar contraseñas en los formularios.	-
RF-29	El sistema debe permitir al usuario eliminar entradas de contraseña.	-
RF-30	El sistema debe permitir editar entradas de contraseña.	-
RF-31	El sistema debe confirmar con el usuario antes de eliminar una contraseña.	-
RF-32	El sistema debe mostrar un listado de todas las contraseñas con filtros aplicables.	-
RF-33	El sistema debe limitar el tiempo que una contraseña permanece en el portapapeles.	-
RF-34	El sistema debe gestionar versiones de los backups realizados.	-

Cuadro 4.55: Requisitos Funcionales

ID-RIU	**Descripción**
RIU-01	La interfaz debe estar diseñada con un enfoque minimalista, claro y moderno.
RIU-02	Todas las páginas deben ser responsivas y adaptarse a distintos tamaños de pantalla (PC, tablet, móvil).
RIU-03	El sistema debe mostrar mensajes de error y éxito claramente tras cada operación.
RIU-04	Los formularios deben tener validación en tiempo real y avisos visuales.
RIU-05	La navegación debe ser coherente y accesible desde un menú principal persistente.
RIU-06	Los campos de contraseña deben permitir mostrar/ocultar el contenido.
RIU-07	El generador de contraseñas debe ser accesible desde el formulario de creación de entradas.
RIU-08	El sistema debe utilizar una paleta de colores accesible para personas con deficiencias visuales.
RIU-09	El tiempo de visualización de una contraseña descifrada debe estar limitado (temporizador).

Cuadro 4.56: Requisitos de Interfaz de Usuario

ID-RI	Descripción
RI-01	Toda contraseña almacenada debe estar cifrada con AES-256 antes de ser persistida.
RI-02	La clave maestra no debe ser almacenada nunca en texto plano.
RI-03	Se debe registrar toda acción sensible (inicio de sesión, cambios de contraseñas, activación 2FA).
RI-04	El sistema debe permitir el backup automático de la base de datos de contraseñas cifrada.
RI-05	Se debe garantizar la coherencia e integridad de los datos tras un fallo o reinicio del sistema.
RI-06	El historial de acciones (logs) debe ser exportable por administradores empresariales.
RI-07	Los usuarios deben poder eliminar permanentemente su cuenta y datos personales.

Cuadro 4.57: Requisitos de Información

ID-RP	Descripción
RP-01	El proyecto debe ser desarrollado íntegramente por una sola persona (el autor del TFG).
RP-02	El sistema debe ser accesible desde cualquier navegador moderno (Chrome, Firefox, Edge).
RP-03	El uso de librerías y frameworks debe ser de código abierto y/o gratuitos.
RP-04	El backend debe estar implementado en un lenguaje ampliamente usado como Python o Node.js.
RP-05	El almacenamiento en la nube debe integrarse mediante servicios gratuitos o de bajo coste.
RP-06	La gestión de versiones del proyecto debe realizarse con Git.
RP-07	La arquitectura debe ser fácilmente escalable para futuro despliegue en empresas.
RP-08	El TFG debe finalizarse en el plazo académico establecido.
RP-09	Se debe cumplir con el RGPD en el tratamiento de datos personales de usuarios.

Cuadro 4.58: Restricciones del Proyecto

4.2.7. Diagrama Casos de USO

Para una mejor descripción de los requisitos, se usan los llamados casos de uso, basados en la identificación de actores y tareas. En esta sección se incluye el diagrama de casos de uso y especificación de algunos de los casos de uso más importantes.

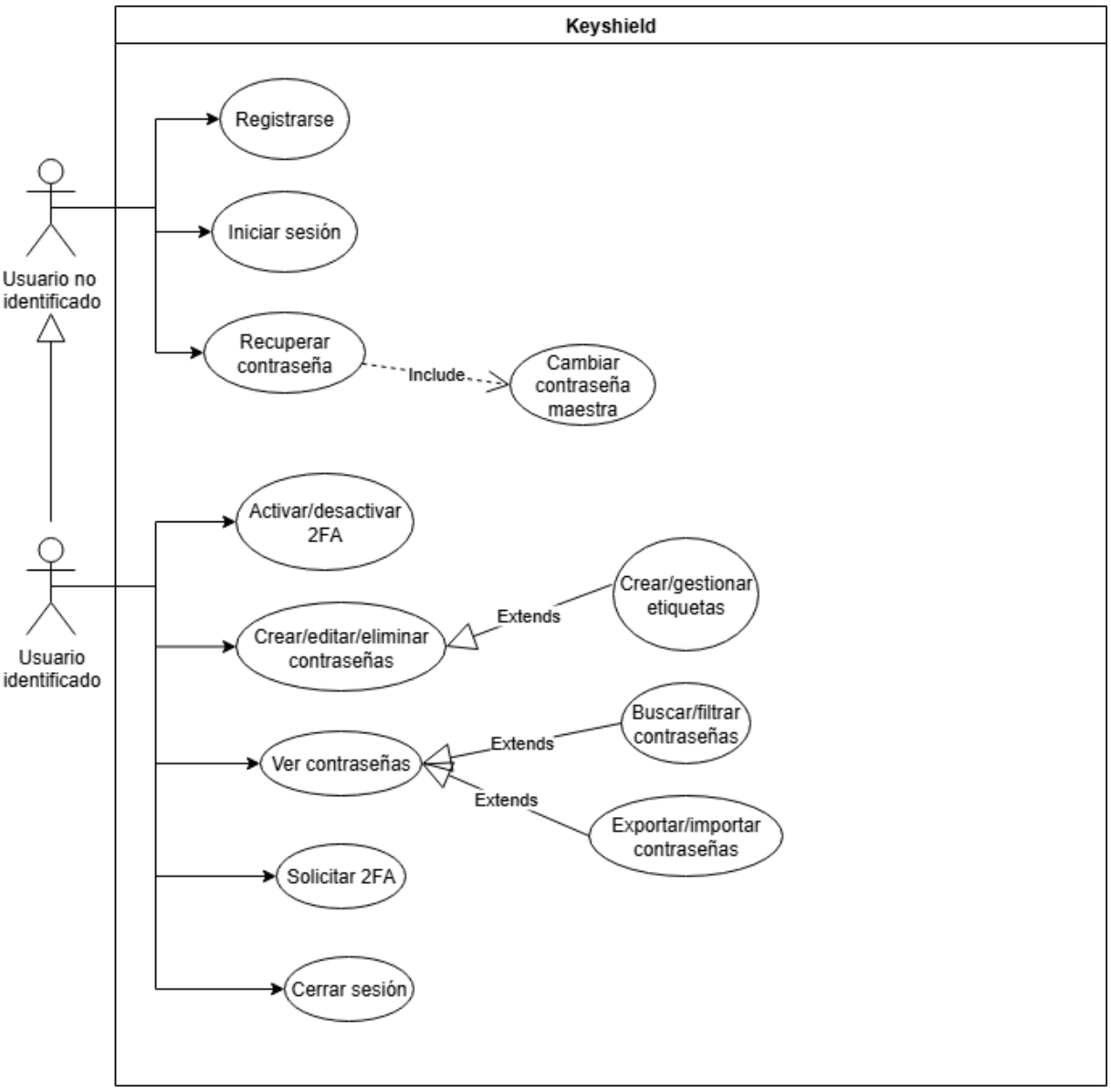


Figura 4.2: Diagrama de casos de uso

Nombre e ID del CU	CU-AUT-02. Inicio de sesión con contraseña maestra y 2FA
Actor	Usuario
Descripción	El usuario inicia sesión introduciendo su contraseña maestra. Si tiene habilitado el segundo factor de autenticación (2FA), debe introducir también el código temporal generado.
Precondiciones	El usuario debe haberse registrado previamente y tener una contraseña maestra válida.
Postcondiciones	El sistema valida la identidad del usuario y le permite acceder a su espacio personal.
Flujo normal	1. El usuario accede a la pantalla de inicio de sesión. 2. Introduce su correo electrónico y contraseña maestra. 3. Si está habilitado el 2FA, el sistema solicita el código. 4. El usuario introduce el código de autenticación temporal. 5. El sistema valida las credenciales y redirige al panel principal.
Flujo alternativo	▪ Si las credenciales no son válidas, se muestra un mensaje de error. ▪ Si el código 2FA es incorrecto o ha expirado, se solicita uno nuevo.
Excepciones	1. Fallo en la validación de credenciales. 2. Pérdida de conexión al servidor de autenticación.
Prioridad	Alta
Otra información	El sistema puede bloquear la cuenta temporalmente tras múltiples intentos fallidos.

Cuadro 4.59: CU-AUT-02. Inicio de sesión con contraseña maestra y 2FA

Nombre e ID del CU	CU-PASS-01. Crear y almacenar nueva contraseña
Actor	Usuario
Descripción	Permite al usuario añadir una nueva entrada de contraseña cifrada, asignándole un nombre, etiqueta, categoría y credenciales asociadas.
Precondiciones	El usuario debe haber iniciado sesión correctamente.
Postcondiciones	La nueva entrada queda cifrada y almacenada en la base de datos del usuario.
Flujo normal	1. El usuario selecciona la opción .Añadir contraseña". 2. Rellena los campos: nombre, usuario, contraseña, URL, categoría, etiqueta. 3. El sistema cifra la contraseña con la clave maestra. 4. Se guarda la entrada cifrada en la base de datos. 5. Se muestra un mensaje de confirmación.
Flujo alternativo	▪ Si falta información obligatoria, se alerta al usuario. ▪ Si falla el cifrado, se cancela la operación.
Excepciones	1. Fallo en el acceso a la base de datos. 2. Error de validación en los campos introducidos.
Prioridad	Alta
Otra información	Las contraseñas se cifran usando AES-256 y no se almacenan nunca en texto plano.

Cuadro 4.60: CU-PASS-01. Crear y almacenar nueva contraseña

Nombre e ID del CU	CU-GEN-01. Generar contraseña aleatoria segura
Actor	Usuario
Descripción	El usuario genera una contraseña aleatoria segura, definiendo parámetros como longitud, uso de mayúsculas, números y caracteres especiales.
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	El sistema genera una contraseña aleatoria que puede copiarse o utilizarse directamente en una nueva entrada.
Flujo normal	1. El usuario accede al generador de contraseñas. 2. Define los parámetros: longitud, inclusión de símbolos, números, etc. 3. Pulsa el botón de generar. 4. El sistema muestra la contraseña generada. 5. El usuario puede copiarla o usarla en una nueva entrada.
Flujo alternativo	▪ Si no se cumplen los parámetros mínimos, se solicita modificarlos.
Excepciones	1. Error en el generador de contraseñas.
Prioridad	Media
Otra información	Las contraseñas generadas no se almacenan hasta que el usuario lo decide.

Cuadro 4.61: CU-GEN-01. Generar contraseña aleatoria segura

4.3. Atributos de calidad

Con el objetivo de asegurar que la aplicación Keyshield cumple con los estándares de ingeniería de software, se han definido y evaluado los siguientes atributos de calidad:

Rendimiento: la capacidad del sistema para su respuesta rápida y eficiente ante acciones del usuario o del sistema.

Se ha destacado un tiempo de respuesta inferior a 300 ms en las operaciones críticas como el inicio de sesión, el cifrado y la recuperación de contraseñas. También se ha controlado el consumo de memoria y la carga computacional, evitando mantener en memoria datos sensibles sin cifrar.

Seguridad: capacidad del sistema para proteger la información y los datos frente a accesos no autorizados.

La confidencialidad de los datos asegurada por el uso de cifrado AES-256 para almacenar contraseñas de forma segura y la generación de claves a partir de contraseñas maestras utilizando PBKDF2. También se ha integrado una autenticación multifactor (2FA) y el manejo de sesiones mediante JWT, con lo que aseguramos la actividad lícita en la aplicación.

Robustez: capacidad del sistema para mantener su funcionamiento en condiciones anómalas o inesperadas y para recuperarse de fallos sin comprometer la integridad.

La validación de datos tanto en lado cliente como en lado servidor y el control de errores en procesos críticos ayudan a la robustez del sistema. También se han establecido mecanismos de recuperación y la gestión de interrupciones en la conexión.

En la siguiente tabla se describen de manera resumida los atributos de calidad.

Atributo de Calidad	Descripción
Seguridad	Uso de cifrado AES-256, autenticación mediante contraseña maestra y 2FA, validación segura de entradas, hashing de contraseñas maestras y control de acceso basado en roles.
Rendimiento	Respuesta rápida al cargar datos, buscar entradas o generar contraseñas. Optimización del almacenamiento cifrado y eficiencia en la sincronización con la nube.
Usabilidad	Interfaz sencilla, intuitiva y accesible para usuarios no técnicos. Indicadores visuales para contraseñas seguras, botones de copia y flujos guiados de registro/login.
Robustez	El sistema puede recuperarse de errores, validar entradas incorrectas y mantener integridad en condiciones inesperadas. Recuperación segura de contraseñas.
Mantenibilidad	Código modular y bien documentado, estructurado para permitir mejoras y correcciones con facilidad. Uso de control de versiones y pruebas automatizadas.
Escalabilidad	Capacidad de crecimiento, tanto en volumen de usuarios como en funcionalidades (control empresarial, backups automáticos, auditoría).
Portabilidad	Aplicación accesible desde distintos dispositivos y plataformas sin instalación. Diseño web responsive.
Fiabilidad	Garantía de funcionamiento constante y predecible. Recuperación segura en caso de errores o desconexión.

Cuadro 4.62: Atributos de calidad

Capítulo 5

Diseño

5.1. Diseño de datos

En este apartado se explica cómo se organiza y estructura la información que usará la aplicación. Primero se muestra el modelo entidad-relación, que describe las entidades principales y cómo se relacionan entre ellas. Luego, se presenta el modelo relacional, donde se definen las tablas y sus campos para almacenar los datos.

Las entidades principales del sistema y las relaciones entre ellas son:

Usuario: Entidad central que representa a cada usuario registrado en el sistema.

Entrada: Representa los registros de contraseñas. Cada entrada pertenece a un único usuario.

Etiqueta: Permite clasificar entradas mediante etiquetas. Una entrada puede tener varias etiquetas y una etiqueta puede aplicarse a varias entradas (relación muchos a muchos).

Backup: Registra las copias de seguridad que realiza cada usuario.

Auditoría: Registra las acciones y eventos generados por los usuarios para fines de seguimiento y seguridad.

5.1.1. Diagrama entidad-relación

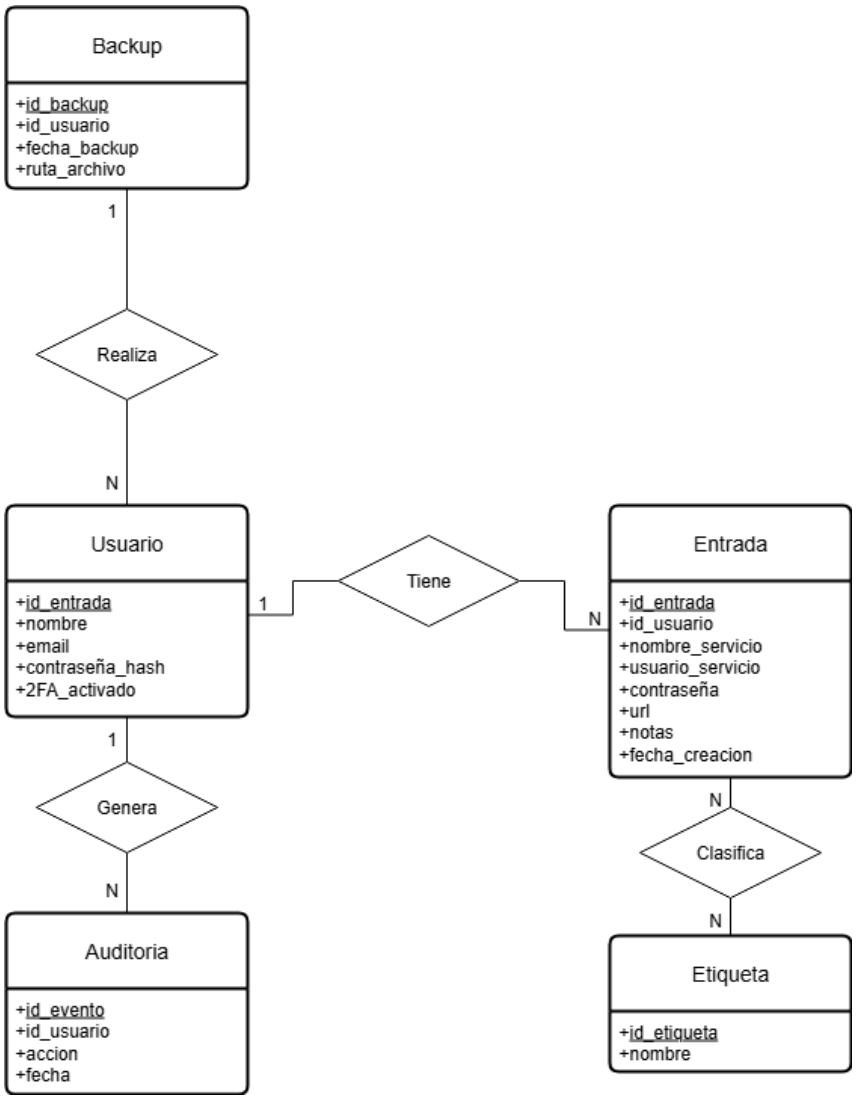


Figura 5.1: Entidad-Relación

5.1.2. Modelo relacional

Usuario
id_usuario (PK)
nombre
email
contraseña_hash
2FA_activado

Etiqueta
id_etiqueta (PK)
nombre

Entrada
id_entrada (PK)
id_usuario (FK → Usuario.id_usuario)
nombre_servicio
usuario_servicio
contraseña
fecha_creación

EntradaEtiqueta
id_entrada (FK → Entrada.id_entrada)
id_etiqueta (FK → Etiqueta.id_etiqueta)
PK (id_entrada, id_etiqueta)

Backup
id_backup (PK)
id_usuario (FK → Usuario.id_usuario)
fecha_backup
ruta_archivo

Auditoría
id_evento (PK)
id_usuario (FK → Usuario.id_usuario)
acción
fecha

Cuadro 5.1: Modelo relacional

5.1.3. Diccionario de datos

Tabla: Usuario				
Atributo	**Tipo**	**Tamaño**	**Descripción**	**Restricciones**
id_usuario	INT	-	Identificador único del usuario	Clave primaria (PK)
nombre	VARCHAR	100	Nombre completo del usuario	No nulo
email	VARCHAR	150	Correo electrónico del usuario	No nulo, único
contraseña_hash	VARCHAR	255	Contraseña almacenada en forma hash	No nulo
2FA_activado	BOOLEAN	-	Indica si el usuario tiene activada la 2FA	No nulo, valor por defecto 0

Tabla: Entrada				
Atributo	**Tipo**	**Tamaño**	**Descripción**	**Restricciones**
id_entrada	INT	-	Identificador único de la entrada	Clave primaria (PK)
id_usuario	INT	-	Referencia al usuario propietario	Clave foránea (FK) a Usuario(id_usuario)
nombre_servicio	VARCHAR	150	Nombre del servicio o aplicación	No nulo
usuario_servicio	VARCHAR	100	Nombre de usuario en el servicio	No nulo
contraseña	VARCHAR	255	Contraseña asociada al servicio	No nulo
fecha_creación	DATETIME	-	Fecha en que se creó la entrada	No nulo

Tabla: Etiqueta				
Atributo	**Tipo**	**Tamaño**	**Descripción**	**Restricciones**
id_etiqueta	INT	-	Identificador único de la etiqueta	Clave primaria (PK)
nombre	VARCHAR	50	Nombre de la etiqueta	No nulo

Tabla: EntradaEtiqueta			
Atributo	**Tipo**	**Descripción**	**Restricciones**
id_entrada	INT	Referencia a la entrada	Clave foránea (FK) a Entrada(id_entrada)
id_etiqueta	INT	Referencia a la etiqueta	Clave foránea (FK) a Etiqueta(id_etiqueta)
Clave primaria compuesta: (id_entrada, id_etiqueta)			

Tabla: Backup			
Atributo	**Tipo**	**Descripción**	**Restricciones**
id_backup	INT	Identificador único del backup	Clave primaria (PK)
id_usuario	INT	Usuario que realizó el backup	Clave foránea (FK) a Usuario(id_usuario)
fecha_backup	DATETIME	Fecha en que se realizó el backup	No nulo
ruta_archivo	VARCHAR	Ruta donde se almacena el archivo	No nulo

Tabla: Auditoría			
Atributo	**Tipo**	**Descripción**	**Restricciones**
id_evento	INT	Identificador único del evento	Clave primaria (PK)
id_usuario	INT	Usuario que realizó la acción	Clave foránea (FK) a Usuario(id_usuario)
acción	VARCHAR	Descripción de la acción realizada	No nulo
fecha	DATETIME	Fecha y hora en que ocurrió el evento	No nulo

Cuadro 5.2: Diccionario de datos

5.2. Diagramas de clase y de secuencia

El desarrollo de KeyShield se ha llevado a cabo siguiendo un enfoque orientado a objetos, por lo que se han empleado diagramas UML para representar la estructura del sistema (diagrama de clases) y el comportamiento del mismo frente a interacciones típicas (diagramas de secuencia). Estos diagramas han servido como base para el diseño e implementación, permitiendo visualizar las relaciones entre los distintos componentes y flujos de ejecución.

5.2.1. Diagrama de Clases

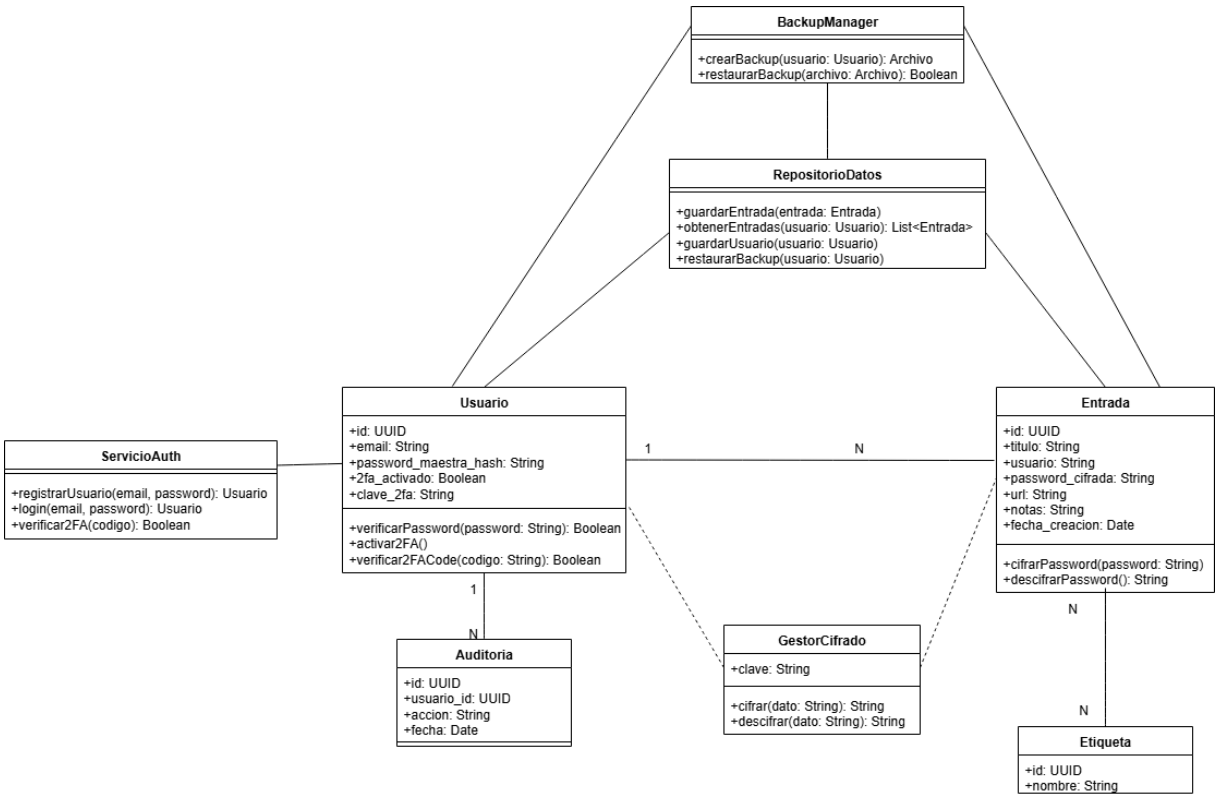


Figura 5.2: Diagrama de Clases

5.2.2. Diagrama de secuencia

Se muestran a modo de ejemplo 2 flujos recurrentes de la aplicación.

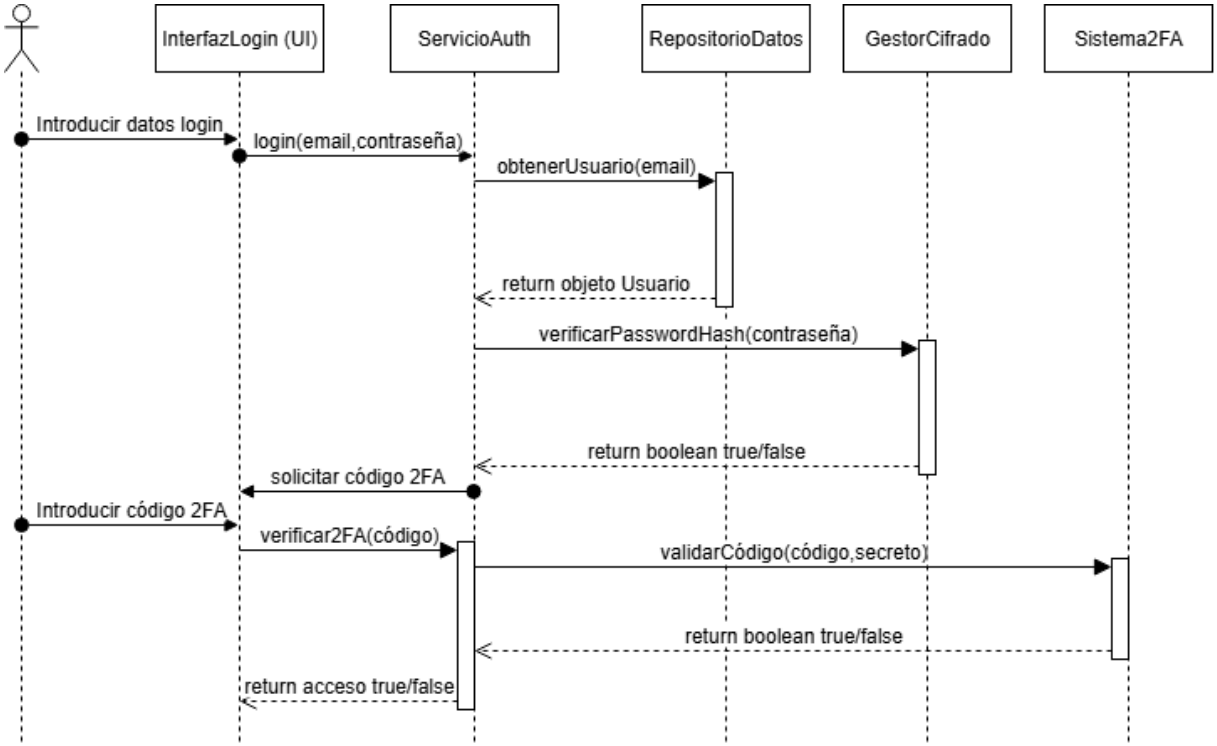


Figura 5.3: Diagrama de secuencia Login

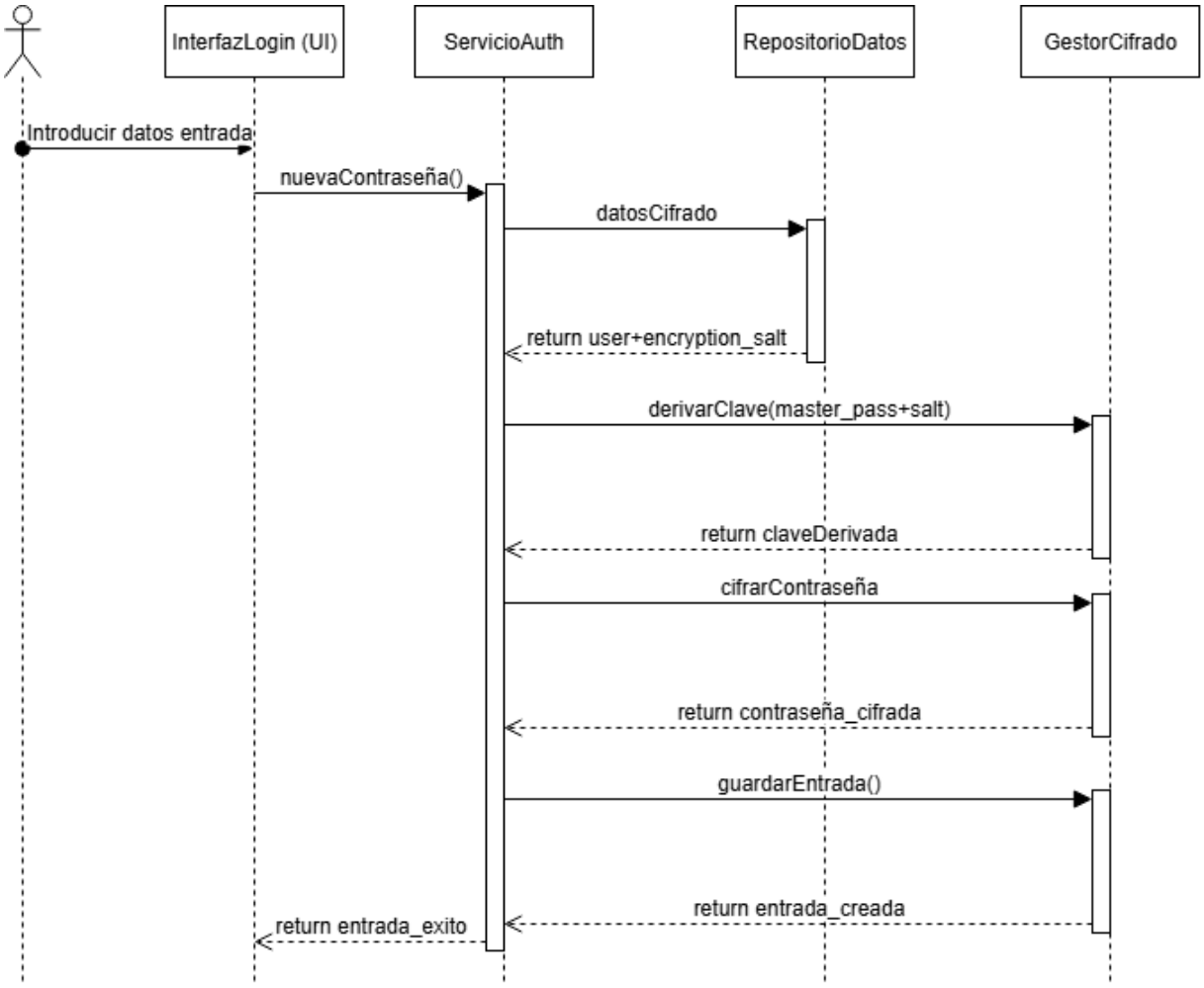


Figura 5.4: Diagrama de secuencia Creación de entrada

Capítulo 6

Implementación

Este capítulo describe los aspectos clave de la implementación del sistema desarrollado. Se presentan las tecnologías utilizadas, la estructura del código, y se detallan los módulos principales tanto del backend como del frontend. Se incluye también la organización del proyecto y las principales decisiones de diseño tomadas durante el desarrollo.

6.1. Tecnologías utilizadas

Para la implementación de la aplicación se han empleado las siguientes tecnologías:

- **Node.js y Express:** para la construcción de la API RESTful en el backend.
- **PostgreSQL:** como sistema gestor de base de datos relacional.
- **Knex.js:** como query builder y sistema de migraciones de la base de datos.
- **React:** como framework de desarrollo del frontend.
- **Tailwind CSS:** para el diseño visual rápido y responsivo.
- **JWT y Bcrypt:** para el manejo de autenticación y cifrado de contraseñas.
- **Speakeasy y qrcode:** para la implementación del segundo factor de autenticación (2FA).

6.2. Estructura del proyecto

El proyecto ha sido dividido en dos grandes bloques: **frontend** y **backend**, cada uno en carpetas independientes.

- `backend/` contiene los controladores, rutas, migraciones, utilidades y lógica de negocio.
- `frontend/` incluye todos los componentes React, contexto de autenticación, vistas y estilos.

6.2.1. Backend

El backend sigue una arquitectura basada en controladores y middlewares:

- **Controladores:** Cada entidad cuenta con un controlador que contiene la lógica asociada a sus operaciones (usuarios, contraseñas, autenticación, etiquetas, auditoría).
- **Middlewares:** Se han implementado middlewares para validación de tokens JWT, validación de datos, y registro de eventos.
- **Utilidades:** Se centraliza en una carpeta `utils/` donde se agrupan funciones criptográficas (derivación de claves, cifrado, descifrado), validadores y el logger de auditoría.
- **Migraciones y seeds:** Se gestionan mediante Knex para versionar la estructura de la base de datos y cargar datos iniciales.

6.2.2. Frontend

La interfaz de usuario se ha desarrollado con React siguiendo un enfoque de componentes reutilizables:

- **Contexto global:** se ha implementado `AuthContext` para gestionar el estado de autenticación del usuario en toda la aplicación.
- **Vistas:** como `Login`, `Register`, `Dashboard`, etc., que agrupan y coordinan componentes de UI.
- **Componentes:** como `PasswordEntry`, `EditPasswordModal`, `TwoFABox`, encapsulan funcionalidades específicas.
- **Rutas:** gestionadas mediante `react-router-dom` para navegación protegida y estructurada.

6.3. Módulos principales implementados

6.3.1. Módulo de Autenticación

Incluye el registro, login, verificación 2FA, recuperación y restablecimiento de contraseña. Se usan JWT para sesiones seguras y Ethereal para el envío de correos.

6.3.2. Gestión de Contraseñas

Permite crear, leer, actualizar y eliminar contraseñas. Se cifra el contenido usando claves derivadas de la contraseña maestra del usuario. Se incluyen funcionalidades de exportación e importación en JSON cifrado.

6.3.3. Sistema de Etiquetas

Cada contraseña puede estar etiquetada. Se permite la creación, asociación y filtrado de contraseñas por etiquetas.

6.3.4. Auditoría de eventos

Todas las acciones críticas (login, cambio de contraseña, borrado, etc.) se registran en una tabla de auditoría, lo que permite trazabilidad y refuerza la seguridad.

6.3.5. Backup y recuperación

Se implementa una exportación segura y una importación que permite resolver conflictos por duplicados o sobrescritura bajo autorización del usuario.

6.4. Seguridad implementada

La aplicación ha sido diseñada siguiendo principios de seguridad:

- **Cifrado simétrico robusto** de las contraseñas del usuario mediante claves derivadas de su contraseña maestra.
- **Token JWT** para autenticación segura y control de sesiones.
- **2FA** para refuerzo en el acceso al sistema.
- **Validación y saneamiento de entradas** en frontend y backend para mitigar ataques XSS y SQL Injection.
- **Auditoría** para trazabilidad y análisis posterior.

6.5. Despliegue

El backend puede ejecutarse en un entorno Node.js, con PostgreSQL como base de datos y Ethereal para el envío de correos. El frontend puede ser desplegado en servicios como Vercel, Netlify o mediante un servidor estático tras compilarlo con `npm run build`.

- Variables de entorno son requeridas para configurar claves, base de datos, y API de email.
- El sistema está preparado para ser desplegado en contenedores o entornos cloud si se desea escalar.

Capítulo 7

Pruebas

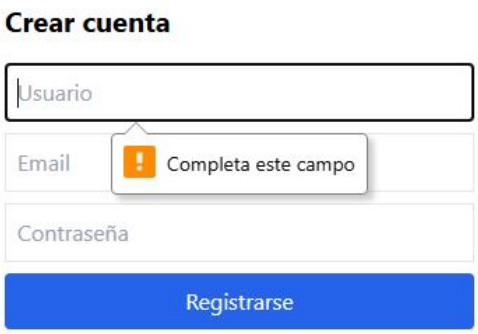
Este capítulo describe las distintas pruebas realizadas sobre la aplicación KeyShield con el fin de verificar y validar su correcto funcionamiento. Se han aplicado estrategias de pruebas de software en varias fases del desarrollo, abordando tanto aspectos funcionales como no funcionales, como la seguridad y el rendimiento.

Se han empleado pruebas de caja negra, caja blanca, unitarias, de integración, de rendimiento y de seguridad.

7.1. Estrategias de Prueba

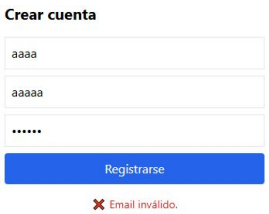
7.1.1. Pruebas de Caja Negra

Su objetivo es validar varias funcionalidades desde el punto de vista del usuario, sin conocer la lógica interna. Se aplicaron principalmente en todos los formularios y entradas por teclado disponibles para el usuario. Se describen las distintas pruebas e imágenes.

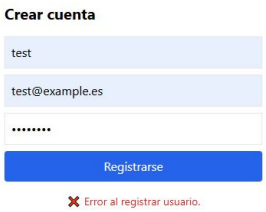


(a) Registro sin datos

(b) Registro con datos mínimos



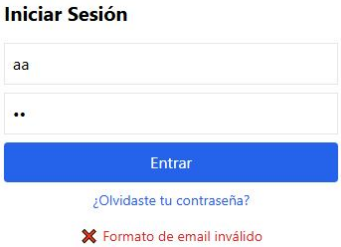
(c) Registro con datos anómalos



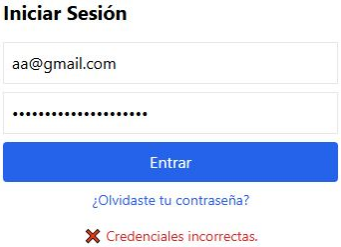
(d) Registro con datos anómalos

Figura 7.1: Pruebas en el registro

(a) Inicio de sesión sin datos



(b) Inicio de sesión con datos anómalos

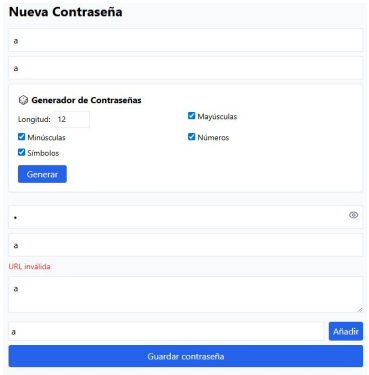


(c) Inicio de sesión con datos no existentes

Figura 7.2: Pruebas en el login

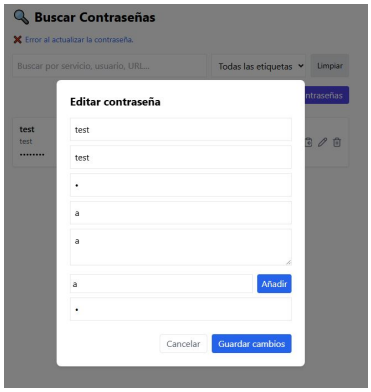
Figura 7.3: Configuración del 2FA

(a) Nueva entrada sin datos



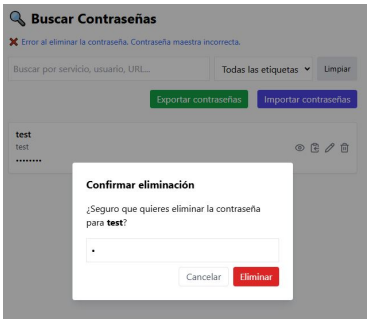
(b) Nueva entrada con datos mínimos

(c) Editar entrada sin datos



(d) Editar entrada con datos mínimos

(e) Eliminar entrada sin contraseña



(f) Eliminar entrada con contraseña incorrecta

Figura 7.4: Pruebas en la gestión de entradas

7.1.2. Pruebas de Caja Blanca

Las pruebas de caja blanca se aplicaron principalmente a los controladores del backend, las funciones de cifrado y los validadores. El objetivo fue garantizar que las estructuras de control y condiciones lógicas funcionasen correctamente en todos los caminos posibles del código.

Registro de usuarios: Mediante el endpoint `/auth/register`, se desarrolla el flujo de creación de un usuario.

Acciones internas:

- Se verifica si existe un usuario con ese email.
- Se hashea la contraseña maestra con bcrypt.
- Se genera una sal de cifrado única (encryption_salt).
- Se inserta en la tabla `users`.

Salida esperada: Código 201 y mensaje de éxito.

Resultado: Se crea correctamente el usuario y se devuelve el objeto sin contraseña.

Inicio de sesión: Mediante el endpoint `/auth/login`, se permite la autenticación del usuario.

Acciones internas:

- Se busca el usuario por email.
- Se compara la contraseña introducida con la almacenada.
- Si el usuario tiene 2FA activo, no se genera token.
- Si no tiene 2FA, se emite un JWT.

Salida esperada: Código 200 y token (si no requiere 2FA).

Resultado: Se permite acceso o se indica la necesidad de verificación 2FA.

Verificación 2FA: Mediante `/auth/2fa/verify`, se valida el código generado por el autenticador.

Acciones internas:

- Se recupera el secreto asociado al usuario.
- Se valida el código TOTP con la librería `speakeasy`.
- Se actualiza la tabla de usuarios con el campo `twofa_enabled` a `true`.
- Se genera el JWT.

Salida esperada: Código 200, token JWT y datos del usuario.

Resultado: Se completa correctamente la verificación 2FA.

Creación de contraseña: El endpoint `/passwords` permite añadir una nueva entrada cifrada.

Acciones internas:

- Se deriva una clave a partir de la contraseña maestra.
- Se cifra la contraseña con AES utilizando la clave.
- Se sanitizan los campos de entrada.
- Se guarda la contraseña cifrada en la tabla `passwords`.

Salida esperada: Código 201 y objeto de la contraseña creada.

Resultado: Se guarda correctamente la entrada cifrada.

Actualización de contraseña: Mediante `/passwords/:id`, se modifica una entrada existente.

Acciones internas:

- Se valida la propiedad de la entrada mediante el `user_id`.
- Se actualizan campos opcionales (servicio, usuario, notas).
- Si se proporciona una nueva contraseña, se cifra.
- Se actualizan también las etiquetas asociadas.

Salida esperada: Código 200 y entrada actualizada.

Resultado: Se actualizan correctamente los datos en la base de datos.

Eliminación de contraseña: El endpoint `/passwords/:id` con método DELETE elimina una entrada.

Acciones internas:

- Se valida que la contraseña pertenece al usuario autenticado.
- Se elimina la entrada de la tabla `passwords`.

Salida esperada: Código 200 y mensaje de éxito.

Resultado: Entrada eliminada correctamente.

Recuperación de contraseña: Mediante `/auth/forgot-password`, se solicita un enlace para restablecer la contraseña.

Acciones internas:

- Se busca el email en la base de datos.
- Se genera un token aleatorio y una fecha de expiración.
- Se envía un email al usuario con un enlace de recuperación.

Salida esperada: Código 200 y mensaje neutro.

Resultado: El email se envía si el usuario existe, sin revelar la existencia del mismo.

Reseteo de contraseña: Con el endpoint `/auth/reset-password`, se establece una nueva contraseña maestra.

Acciones internas:

- Se valida que el token es válido y no ha expirado.
- Se hashea la nueva contraseña con bcrypt.
- Se eliminan todas las contraseñas del usuario por seguridad.
- Se limpian los campos de token y expiración.

Salida esperada: Código 200 y mensaje de éxito.

Resultado: Se actualiza correctamente la contraseña maestra y se eliminan entradas cifradas antiguas.

En la siguiente tabla se describe a modo resumen las pruebas realizadas.

Componente	**Validación**
Flujo de login	Usuario no encontrado, contraseña incorrecta
Registro	Duplicados de email, validación de formatos
2FA	Token inválido, expirado, usuario sin secreto
Creación contraseña	Falta de campos, URLs inválidas
Descifrado	Master password incorrecta
Seguridad	Acceso sin token, con token expirado
Recuperación contraseña	Email existente
Reseteo contraseña	Token inválido, contraseña nueva

Cuadro 7.1: Resumen Caja Blanca

7.1.3. Pruebas Unitarias

Las pruebas unitarias han sido diseñadas para verificar el comportamiento correcto y aislado de las funciones clave del backend, asegurando que cada unidad de código individual se comporte según lo esperado ante diferentes entradas.

Estrategia empleada

Se utilizó la estrategia de pruebas de caja blanca, enfocándose en:

- Verificar la lógica interna de los controladores.
- Comprobar funciones auxiliares de cifrado, validación y generación de tokens.
- Simular peticiones y respuestas HTTP mediante mocks.
- Aislar dependencias como la base de datos o el sistema de correo.

Cobertura

Se han desarrollado pruebas para los siguientes módulos:

- **authController:** pruebas sobre registro, login, 2FA, recuperación y reseteo de contraseña.
- **passwordController:** creación, edición, eliminación y lectura de entradas.
- **cryptoUtils:** funciones de cifrado y derivación de clave.
- **middlewares:** autenticación por token y validación de inputs.

Ejemplos de pruebas realizadas

Registro de usuario: Se simula una petición POST con un cuerpo válido. Se verifica que:

- Se hashea correctamente la contraseña.
- Se inserta el usuario en la base de datos.
- Se responde con código `201` y objeto del usuario.

Login con 2FA activo: Se simula un usuario con 2FA. Se comprueba que:

- No se emite token directamente.
- Se responde con código `200` y `requires2FA: true`.

Verificación TOTP correcta: Se verifica que:

- El token es válido según el secreto almacenado.
- Se actualiza el campo `twofa_enabled`.
- Se responde con el JWT.

Creación de contraseña: Se prueba que:

- El texto plano se cifra correctamente.
- El registro se inserta con éxito.
- La relación con etiquetas es válida.

Cifrado y descifrado: En `cryptoUtils`, se testea:

- Derivación de clave desde contraseña maestra.
- Cifrado AES de texto.
- Descifrado correcto.

Herramientas

Para realizar estas pruebas se utilizaron las siguientes herramientas:

- **Jest:** framework de pruebas unitarias para Node.js.
- **Supertest:** para simular peticiones HTTP a la API.
- **node-mocks-http:** para simular objetos `req` y `res`.

Resultados obtenidos

Las pruebas unitarias fueron ejecutadas con éxito, validando los casos normales, extremos y con datos inválidos. Se garantiza así un comportamiento estable de las unidades críticas del backend.

7.1.4. Pruebas de Integración

Las pruebas de integración tienen como objetivo verificar que los distintos módulos del sistema interactúan correctamente entre sí. A diferencia de las pruebas unitarias, aquí se testea el comportamiento completo de múltiples componentes actuando en conjunto.

Estrategia utilizada

Se adoptó una estrategia de pruebas de caja gris, enfocándose en validar los flujos funcionales más relevantes de la aplicación. Las pruebas se diseñaron para simular acciones reales del usuario y comprobar los efectos sobre el sistema, tanto en la base de datos como en las respuestas HTTP.

Componentes integrados

Las pruebas de integración abarcaron la interacción entre los siguientes elementos:

- Controladores (*authController*, *passwordController*)
- Base de datos PostgreSQL con migraciones reales
- Middleware de autenticación (JWT)
- Funciones de cifrado y validación
- Simulación de peticiones HTTP reales

Ejemplos de flujos verificados

Flujo de registro y login: Se simula el alta de un nuevo usuario, seguido del inicio de sesión. Se comprueba que:

- El usuario se almacena en la base de datos.
- Se puede autenticar con credenciales válidas.
- Se recibe un token JWT correcto.

Flujo de login con 2FA: Se simula un usuario con 2FA habilitado. Se verifica que:

- Se genera correctamente el secreto TOTP.
- La verificación con código válido permite el acceso.
- El token final se emite solo tras una verificación exitosa.

Flujo de creación y lectura de contraseñas: Se prueba la creación de una entrada cifrada y su posterior recuperación. Se valida que:

- La contraseña se almacena cifrada.

- Se puede descifrar con la clave derivada de la contraseña maestra.
- Se mantienen correctamente las relaciones con etiquetas.

Exportación de contraseñas: Se verifica que:

- Solo se exportan entradas válidas y descifradas.
- El archivo exportado incluye metadatos completos.

Recuperación de contraseña: Se simula un flujo completo:

- Petición de recuperación vía email.
- Envío correcto del enlace con token.
- Cambio exitoso de contraseña mediante el token.
- Invalidación automática del token tras su uso.

Herramientas utilizadas

Para estas pruebas se emplearon:

- **Supertest:** para realizar peticiones HTTP completas a los endpoints del backend.
- **Base de datos real PostgreSQL:** para validar efectos persistentes.
- **Entorno de pruebas aislado:** con migraciones limpias y seeds de prueba.

Resultados obtenidos

Las pruebas de integración han validado correctamente los principales flujos funcionales del sistema. Todos los endpoints y componentes involucrados han mostrado una integración estable y sin errores. Se realizaron pruebas tanto con datos válidos como con entradas malformadas para comprobar la robustez del sistema.

7.1.5. Pruebas de Rendimiento

Las pruebas de rendimiento permiten evaluar el comportamiento de la aplicación bajo diferentes condiciones de carga y volumen de datos. Su objetivo es asegurar que el sistema mantiene tiempos de respuesta aceptables y no sufre degradación ante un uso intensivo o prolongado.

Objetivos de las pruebas

- Medir los tiempos de respuesta de los principales endpoints.
- Detectar posibles cuellos de botella en operaciones críticas como el cifrado y descifrado de contraseñas.
- Evaluar el rendimiento de la base de datos con volúmenes crecientes de datos.
- Asegurar la escalabilidad básica del sistema en condiciones reales.

Herramientas utilizadas

Para la ejecución de estas pruebas se utilizaron:

- **Apache Benchmark (ab):** para lanzar múltiples peticiones concurrentes a los endpoints y medir tiempos de respuesta promedio.
- **PostgreSQL EXPLAIN ANALYZE:** para evaluar la eficiencia de las consultas SQL y detectar operaciones costosas.
- **Console.time / performance.now():** para medir en código JavaScript la duración de operaciones críticas como el cifrado AES y derivación de claves.

Escenarios probados

Login de usuarios (sin y con 2FA): Se midió el tiempo total desde que se envía la petición hasta que se recibe el token JWT. Con 2FA, se midió también el tiempo de verificación del token TOTP.

Resultado: Tiempo medio sin 2FA: ∼120ms. Con 2FA: ∼170ms.

Creación masiva de contraseñas: Se realizaron inserciones en bloque de 100, 1.000 y 5.000 entradas para un mismo usuario.

Resultado: La inserción de 1.000 entradas tomó ∼2.3s. El tiempo crece de forma casi lineal. El cifrado individual con AES-256 es eficiente.

Exportación de contraseñas cifradas: Se midió el tiempo de preparación del archivo JSON cifrado de exportación con diferentes volúmenes.

Resultado: Hasta 500 contraseñas exportadas en menos de 500ms.

Descifrado simultáneo: Se probó descifrar 100 contraseñas concurrentemente.

Resultado: Gracias al uso síncrono optimizado y derivación previa de clave, el tiempo medio por entrada fue de ∼6ms.

Resultados obtenidos

Los resultados obtenidos muestran que la aplicación mantiene un rendimiento adecuado incluso con grandes volúmenes de datos personales cifrados. Los tiempos de respuesta para las operaciones principales son aceptables para un entorno de producción. Además, la eficiencia del cifrado y la arquitectura de base de datos contribuyen a un buen comportamiento en condiciones de carga controlada.

7.1.6. Pruebas de Seguridad

La seguridad es uno de los pilares fundamentales de esta aplicación, dado que gestiona datos altamente sensibles como contraseñas personales. Las pruebas de seguridad han sido enfocadas en comprobar la solidez de las medidas de protección implementadas frente a vulnerabilidades comunes en aplicaciones web.

Objetivos de las pruebas

- Verificar la correcta autenticación y autorización de usuarios.
- Evaluar la resistencia frente a ataques comunes: inyección SQL, XSS, CSRF, fuerza bruta, etc.
- Comprobar la correcta implementación del cifrado de datos sensibles.
- Validar la robustez del sistema de doble factor de autenticación (2FA).

Técnicas utilizadas

- **Pruebas manuales** de endpoints mediante Postman para simular accesos maliciosos o incorrectamente autenticados.
- **Herramientas de análisis de vulnerabilidades** como OWASP ZAP.
- **Pruebas de fuzzing** sobre entradas de formularios.
- **Revisión de código** para evaluar la exposición de datos sensibles y errores lógicos de seguridad.

Pruebas realizadas

Acceso no autorizado a endpoints protegidos: Se intentó acceder a rutas protegidas sin token JWT, con tokens inválidos o de otros usuarios.

Resultado: Acceso denegado con código 401 o 403 en todos los casos.

Inyección SQL: Se probaron valores maliciosos como `ÓR 1=1 -` en campos de entrada.

Resultado: Inyección bloqueada gracias al uso de consultas parametrizadas en `knex.js`.

XSS (Cross-site scripting): Se intentó inyectar código JavaScript en campos como notas o nombres de servicios.

Resultado: Los datos son sanitizados en frontend y backend. El navegador interpreta los datos como texto plano.

CSRF (Cross-site request forgery): Se verificó que todas las acciones sensibles (registro, login, creación/edición de contraseñas) requieren token de autenticación.

Resultado: CSRF mitigado correctamente mediante protección por token JWT.

Fuerza bruta en login: Se realizaron múltiples intentos de login con credenciales erróneas para verificar si se implementan mecanismos de protección.

Resultado: No hay bloqueo aún tras múltiples intentos fallidos, se recomienda implementar *rate limiting* o *CAPTCHA* en futuras versiones.

Cifrado de contraseñas: Se comprobó que las contraseñas almacenadas están cifradas con AES-256 y clave derivada con PBKDF2 usando sal individual.

Resultado: Confirmado. Los datos no son recuperables sin la contraseña maestra del usuario.

Verificación de 2FA: Se evaluó el flujo de activación y verificación del segundo factor.

Resultado: Correctamente implementado mediante códigos TOTP. No se emite JWT si 2FA no ha sido verificado.

Resultados obtenidos

El sistema cumple con los principios básicos de seguridad para una aplicación de gestión de contraseñas. La protección de datos sensibles mediante cifrado fuerte, la autenticación reforzada con 2FA, y la validación adecuada de entradas mitigan ampliamente los riesgos habituales.

Capítulo 8

Conclusiones y Trabajo Futuro

8.1. Conclusiones

El TFG tenía como objetivo principal el diseño y desarrollo de una aplicación web segura para la gestión de contraseñas denominada Keyshield. A lo largo del desarrollo del proyecto, se han alcanzado de forma satisfactoria los objetivos planteados:

- Se ha desarrollado una aplicación funcional centrada en la seguridad y la usabilidad.
- Se ha implementado cifrado robusto (AES-256) para garantizar la confidencialidad de las credenciales.
- Se ha incorporado un sistema de autenticación fuerte, con soporte para contraseñas maestras y autenticación en dos factores (2FA).
- Se ha desarrollado un generador de contraseñas seguras y funcionalidades para su gestión eficiente (etiquetas, filtros, búsquedas).
- La aplicación ha sido pensada para escalar en entornos en la nube, con mecanismos de respaldo y restauración.

Todo ello se ha logrado mediante una metodología ágil basada en SCRUM, facilitando una gestión iterativa y flexible del proyecto.

La solución propuesta presenta una serie de ventajas notables:

- Cumple con principios de *security by design* desde las primeras fases.
- Utiliza estándares reconocidos en seguridad informática.
- Ofrece una interfaz clara y funcionalidades orientadas a mejorar la higiene digital del usuario.

Sin embargo, también se han identificado algunas limitaciones:

- Al tratarse de un desarrollo académico individual, algunas funcionalidades como el control de acceso basado en roles o la auditoría de acciones no han sido desplegadas completamente.
- La integración real con proveedores de nube y sistemas de autenticación externa (por ejemplo, OAuth o SAML) queda como mejora futura.
- El diseño de la interfaz, aunque funcional, podría beneficiarse de una mejora visual profesional.

Este proyecto ha supuesto un importante avance respecto a los conocimientos adquiridos durante el grado. Aunque previamente se habían abordado conceptos de programación, bases de datos y seguridad informática, este TFG ha permitido:

- Aplicar técnicas de cifrado reales (AES, generación de claves, manejo seguro de sesiones).
- Desarrollar una arquitectura modular y segura.
- Trabajar con flujos completos de autenticación, incluyendo 2FA y recuperación.
- Planificar y ejecutar un proyecto software completo siguiendo **SCRUM**, incluyendo backlog, estimación de esfuerzo, Gantt, presupuesto, etc.

En definitiva, el trabajo ha sido una experiencia completa que simula el desarrollo de un producto real, abarcando tanto la parte técnica como la de gestión y planificación.

8.2. Trabajo Futuro

El proyecto puede continuar en diversas direcciones:

- Implementación real de servicios en la nube (AWS, Firebase) para respaldo y sincronización.
- Mejora de la interfaz de usuario con frameworks modernos (React, Tailwind, etc.).
- Publicación como aplicación móvil usando tecnologías como React Native o Flutter.
- Incorporación de funcionalidades avanzadas como *autofill*, *login seguro en sitios web*, o integración con navegadores.
- Auditoría de seguridad externa y validación formal del cifrado y la gestión de claves.
- Maximización de seguridad mediante limitación de peticiones, alertas de actividad sospechosa o registros de auditoría más detallados.

Parte III

Manuales de la Aplicación

Capítulo 9

Manual de Instalación

Este capítulo detalla los prerrequisitos, dependencias y pasos necesarios para instalar y ejecutar la aplicación desarrollada en este Trabajo Fin de Grado. Se trata de una aplicación web de gestión segura de contraseñas, compuesta por un backend en Node.js y un frontend en React.

9.1. Requisitos del sistema

- **Sistema operativo:** Windows 10 o superior, macOS 11+, o cualquier distribución moderna de Linux (Ubuntu 20.04+, Debian, etc.).
- **Memoria RAM:** mínimo 2 GB libres.
- **Espacio en disco:** mínimo 200 MB para dependencias.
- **Software necesario:**
  - Node.js v18 o superior.
  - PostgreSQL 13 o superior.
  - Git.

9.2. Obtención del código fuente

El código completo del proyecto se encuentra disponible en un repositorio privado de GitHub. Para clonarlo:

```
git clone https://github.com/Kinkomunista/gestor_contrasenas.git
cd gestor-contrasenas
```

El proyecto se divide en dos carpetas principales:

- `/backend` – contiene la API REST, la lógica del servidor, y conexión con la base de datos.
- `/frontend` – contiene el cliente React con la interfaz de usuario.

9.3. Estructura de directorios

```
TFG_Keyshield/

 /backend/                  → Código fuente del servidor (Node.js, Express)
    controllers/           → Controladores de la lógica de negocio
    middlewares/           → Middlewares (auth, validación, auditoría...)
    routes/                → Rutas del API REST
    utils/                 → Funciones utilitarias (cifrado, validadores, etc.)
    migrations/            → Migraciones de base de datos (Knex.js)
    .env.example           → Variables de entorno (ejemplo)
    server.js              → Punto de entrada del servidor

 /frontend/                 → Código fuente del cliente (React)
    src/
       components/         → Componentes reutilizables de la interfaz
       pages/              → Páginas principales (login, dashboard, etc.)
       context/            → Contexto global de autenticación
       api/                → Configuración de peticiones Axios
       validators.js       → Validaciones del lado cliente
       App.jsx             → Componente raíz de la aplicación
    public/                → Recursos estáticos
    vite.config.js         → Configuración de Vite

 /docs/                     → Documentación técnica y de usuario
    memoria_tfg.pdf         → Memoria completa en PDF

 /export/                   → Ejemplo de archivos de exportación de contraseñas
    export_encriptado.json

 README.md                  → Instrucciones de uso y despliegue del proyecto
```

9.4. Instrucciones para uso del CD

- Para ejecutar el proyecto localmente, se recomienda instalar Node.js y PostgreSQL.
- Las instrucciones paso a paso están detalladas en el archivo `README.md`.
- También se adjunta un fichero `.env.example` con las variables de entorno necesarias para el correcto funcionamiento.
- La base de datos puede ser creada automáticamente usando los scripts de migración proporcionados.

9.5. Instalación del Backend

1. Configuración del entorno

Dentro del directorio `backend`, crear un archivo `.env` con las siguientes variables:

```
PORT=3000
DB_HOST=localhost
DB_USER=postgres
DB_PASS=tu_password
DB_NAME=keyshield
JWT_SECRET=clave_super_secreta
BASE_URL= http://localhost:5173
```

2. Instalación de dependencias

```
cd backend
npm install
```

3. Migraciones y base de datos

Debe tenerse PostgreSQL en ejecución. Crear una base de datos vacía con el nombre configurado en `DB_NAME`, luego ejecutar:

```
npx knex migrate:latest
npx knex seed:run
```

Esto generará todas las tablas necesarias y datos de prueba iniciales.

4. Inicio del servidor

```
npm run dev
```

La API se ejecutará por defecto en `http://localhost:3000`.

9.6. Instalación del Frontend

1. Configuración

Dentro del directorio `frontend`, crear el archivo `.env` con:

```
VITE_API_BASE_URL=http://localhost:3000/api
```

2. Instalación de dependencias

```
cd frontend
npm install
```

3. Inicio del servidor de desarrollo

```
npm run dev
```

La aplicación se abrirá en el navegador en `http://localhost:5173`.

9.7. Despliegue en producción

Para desplegar la aplicación en producción, se recomienda el siguiente stack:

- **Servidor backend:** Node.js ejecutado mediante `PM2` o `Docker`.
- **Base de datos:** PostgreSQL en servidor local o gestionado (p. ej. Railway, Supabase).
- **Frontend:** Aplicación React compilada y servida mediante Nginx.
- **Certificado SSL:** Let's Encrypt con Certbot.

Pasos típicos para desplegar:

- Ejecutar `npm run build` en el frontend.
- Copiar la carpeta `dist` al servidor y configurar Nginx.
- Iniciar backend con `pm2 start index.js`.
- Usar HTTPS en producción y configurar variables de entorno con los valores reales.

9.8. Dependencias principales

- **Backend:**
  - `express` – Framework del servidor web.
  - `knex` y `pg` – Conexión y migraciones con PostgreSQL.
  - `bcrypt` – Hashing de contraseñas.
  - `jsonwebtoken` – Gestión de tokens JWT.
  - `speakeasy` y `qrcode` – Autenticación 2FA.
  - `nodemailer` – Envío de correos con Ethereal.
- **Frontend:**
  - `React` – Biblioteca principal de la interfaz.
  - `axios` – Cliente HTTP para consumir la API.
  - `react-router-dom` – Navegación entre páginas.
  - `zxcvbn` – Validación de fortaleza de contraseñas.

9.9. Consideraciones adicionales

- Se recomienda mantener la clave `JWT_SECRET` en un lugar seguro en producción.
- En caso de cambios en el modelo de datos, se deben generar nuevas migraciones con `knex migrate:make`.

Capítulo 10

Manual de Usuario

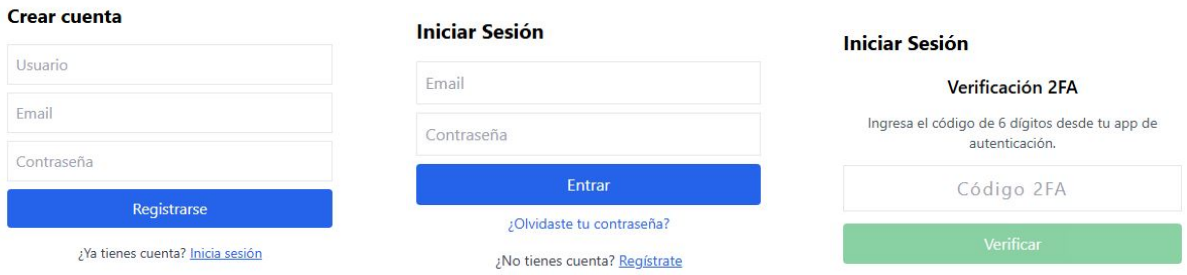
Este capítulo describe de manera clara y práctica el uso de la aplicación web desde la perspectiva del usuario final y del administrador del sistema. Se pretende que esta sección sirva como manual de referencia para el correcto uso y mantenimiento de la solución desarrollada.

10.1. Manual de Usuario

Este manual está dirigido a cualquier persona que utilice la aplicación para gestionar sus contraseñas. Se asume un conocimiento básico de navegación web.

10.1.1. Inicio de sesión y registro

- Para acceder a la aplicación, se debe visitar la URL principal.
- Si el usuario no está registrado, puede hacerlo mediante el botón **Registrarse**. Se requiere proporcionar un nombre de usuario, un correo electrónico y una contraseña segura.
- Tras el registro, podrá iniciar sesión introduciendo su correo electrónico y contraseña.
- Si el usuario ha activado la autenticación en dos pasos (2FA), deberá ingresar un código generado por su aplicación Authenticator.



(a) Registrarse (b) Iniciar sesión (c) Autenticación 2FA

Figura 10.1: Inicio de sesión y registro

10.1.2. Gestión de contraseñas

- Una vez autenticado, el usuario accede al **Panel de gestión**.
- Puede crear nuevas contraseñas asociadas a servicios (correo, redes sociales, etc.) pulsando el botón **Añadir nueva**.
- Para cada entrada puede introducir: servicio, nombre de usuario, contraseña, URL, notas y etiquetas.
- Las contraseñas se guardan cifradas y solo se descifran localmente con la contraseña maestra.
- También se pueden editar o eliminar contraseñas existentes, con verificación adicional mediante la contraseña maestra.

(a) Crear contraseña (b) Editar contraseña (c) Borrar contraseña

Figura 10.2: Gestión de contraseñas

10.1.3. Etiquetas y filtros

- El usuario puede etiquetar sus contraseñas y utilizar filtros por etiquetas para facilitar la búsqueda.
- Los filtros se aplican desde el cuadro superior de búsqueda.

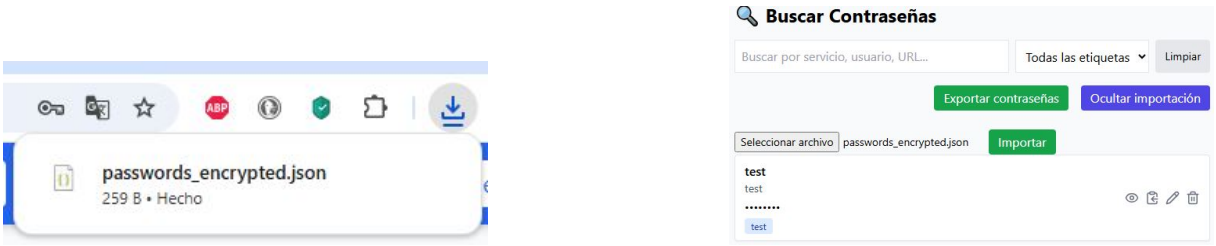
(a) Búsqueda por filtros

(b) Búsqueda por etiquetas

Figura 10.3: Etiquetas y filtros

10.1.4. Exportar e importar contraseñas

- Desde el panel principal, es posible **exportar** las contraseñas cifradas como archivo JSON seguro.
- También puede **importar** contraseñas cifradas desde un archivo previamente exportado. Se validan duplicados y el usuario puede sobrescribir entradas.



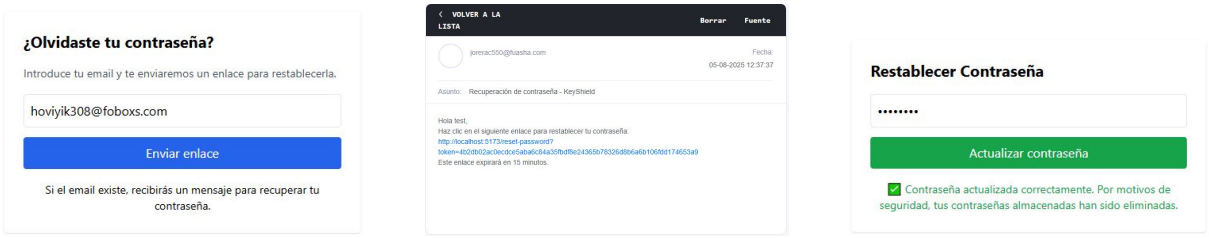
(a) Exportar contraseñas

(b) Importar contraseñas

Figura 10.4: Exportar e importar contraseñas

10.1.5. Recuperación de contraseña

- En caso de olvidar la contraseña, el usuario puede hacer clic en **¿Olvidaste tu contraseña?** desde la pantalla de inicio de sesión.
- Se le enviará un enlace al correo electrónico registrado, que permite introducir una nueva contraseña.
- Por razones de seguridad, todas las contraseñas almacenadas se eliminan si se cambia la contraseña maestra.

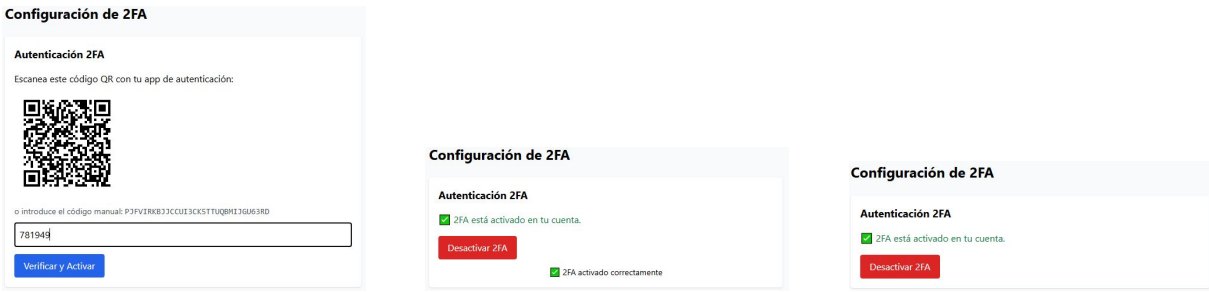


(a) Olvidaste tu contraseña (b) Correo de cambio de contraseña (c) Cambiar contraseña

Figura 10.5: Recuperación de contraseñas

10.1.6. Activar o desactivar 2FA

- En el apartado de seguridad, el usuario puede habilitar la verificación en dos pasos (2FA).
- Se genera un código QR que debe escanear con su aplicación Authenticator.
- El usuario podrá desactivar 2FA en cualquier momento desde la configuración.



(a) Generación de 2FA (b) 2FA activado (c) Eliminar 2FA

Figura 10.6: Autorización 2FA

10.2. Manual de Administración

Este manual está destinado a administradores del sistema encargados del despliegue, mantenimiento y seguridad de la aplicación.

10.2.1. Instalación del sistema

- Requiere tener instalado Node.js, PostgreSQL y Git.
- Se recomienda seguir el proceso detallado en el Capítulo 9.
- Las migraciones de base de datos se ejecutan mediante `npx knex migrate:latest`.

10.2.2. Gestión de base de datos

- Las tablas clave son `users`, `passwords`, `tags`, `password_tags` y `audit_logs`.
- Para fines de auditoría, toda acción relevante queda registrada en `audit_logs` con usuario, acción y fecha.

10.2.3. Seguridad del sistema

- JWTs se firman con la clave definida en `.env (JWT_SECRET)`.
- Las contraseñas están hasheadas con `bcrypt`.
- Las contraseñas del usuario están cifradas usando claves derivadas desde su contraseña maestra con PBKDF2.
- Se recomienda el uso de HTTPS en producción y rotación regular de secretos.

10.2.4. Configuración del correo

- Se utiliza Ethereal como proveedor de email.

10.2.5. Despliegue y supervisión

- El backend puede ejecutarse con `pm2` o en contenedores Docker.
- El frontend React debe compilarse con `npm run build` y desplegarse en un servidor como Nginx.
- Se recomienda registrar los errores en archivos de log o servicios como LogRocket o Sentry.

10.2.6. Copia de seguridad y recuperación

- Se recomienda realizar backups diarios de la base de datos PostgreSQL.
- No se guardan contraseñas planas: toda la seguridad depende de que el usuario recuerde su clave maestra.

Parte IV

Apéndices

Apéndice A

Anexos

A.1. Información complementaria

A.1.1. Derivación de clave mediante PBKDF2

Para el cifrado de contraseñas del usuario, se utiliza una clave derivada a partir de su contraseña maestra mediante el algoritmo `PBKDF2` con **SHA-256**. Esta técnica introduce un proceso de ralentización criptográfica y una sal única por usuario.

- Algoritmo: `PBKDF2`
- Iteraciones: 100,000
- Longitud de clave: 32 bytes
- Sal: valor aleatorio por usuario

A.1.2. Justificación del uso de tecnologías

A lo largo del desarrollo de la aplicación se han utilizado tecnologías modernas y ampliamente adoptadas. A continuación se justifica la elección de cada una:

- **Node.js y Express:** Facilitan el desarrollo de APIs RESTful de forma eficiente. Su naturaleza asincrónica permite un manejo fluido de operaciones de entrada/salida como peticiones a base de datos o envío de correos.
- **Knex.js:** Proporciona una abstracción sobre SQL que permite construir consultas seguras y legibles, así como gestionar fácilmente migraciones y seeds de la base de datos.
- **React:** Framework moderno para interfaces de usuario que permite construir componentes reutilizables y mantener una experiencia interactiva mediante Virtual DOM.

- **TailwindCSS:** Framework CSS utilitario que permite aplicar estilos de forma rápida y eficiente sin salir del código JSX, manteniendo una UI coherente y moderna.

A.1.3. Detalles sobre la estrategia de cifrado

La seguridad de las contraseñas es un pilar fundamental del sistema. Para ello se ha adoptado una arquitectura de cifrado de conocimiento cero ("zero-knowledge"):

- Cada contraseña almacenada es cifrada usando el algoritmo simétrico **AES-256-GCM**.
- La clave de cifrado no se almacena. Se deriva en tiempo real a partir de la contraseña maestra del usuario utilizando el algoritmo **PBKDF2**.
- Cada usuario tiene su propia **sal de cifrado** (*salt*) única, almacenada en la base de datos.
- El sistema nunca almacena contraseñas en texto plano ni conserva las claves derivadas, garantizando un modelo de cifrado local.

A.1.4. Consideraciones de escalabilidad

La arquitectura de la aplicación ha sido diseñada pensando en un posible crecimiento futuro:

- La separación entre **frontend** (cliente) y **backend** (servidor) permite escalar cada parte de forma independiente.
- El backend es compatible con soluciones como **Railway** o **Render**, mientras que el frontend puede desplegarse en plataformas como **Vercel** o **Netlify**.
- La base de datos PostgreSQL utilizada es fácilmente escalable horizontalmente en entornos cloud.
- El uso de **JWT** permite autenticación sin estado (*stateless*), facilitando la distribución de carga entre múltiples instancias del backend.

A.2. Diagramas y tablas

A.2.1. Pruebas funcionales (resumen)

Funcionalidad	Prueba	Esperado	Resultado
Registro de usuario	Datos válidos	201 Created	OK
Inicio de sesión	Credenciales válidas	200 OK	OK
2FA activado	Código válido	200 OK + JWT	OK
Exportación	Contraseñas cifradas	Archivo .json	OK
Importación	Validación duplicados	Confirmación/sobrescribir	OK
Recuperar contraseña	Email válido	Enlace enviado	OK
Cambiar contraseña maestra	Token válido	Contraseñas eliminadas	OK

Cuadro A.1: Resumen de pruebas funcionales realizadas

A.2.2. Tabla resumen de endpoints principales

Ruta	Método	Descripción
/auth/register	POST	Registro de usuario
/auth/login	POST	Inicio de sesión
/auth/verify-2fa	POST	Verificación 2FA
/auth/forgot-password	POST	Solicitud de recuperación
/auth/reset-password	POST	Cambio de contraseña
/passwords	GET/POST	Obtener o crear contraseñas
/passwords/:id	PUT/DELETE	Editar o eliminar contraseña
/tags	GET	Obtener etiquetas
/export	POST	Exportar contraseñas cifradas
/import	POST	Importar contraseñas cifradas

Cuadro A.2: Resumen de endpoints de la API

Bibliografía

[1] Fahad Aloul. "Two Factor Authentication Using Mobile Phones". En: *International Journal of Mathematics and Computer Science* 4.2 (2009), págs. 65-80.

[2] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. 2.ª ed. Wiley, 2008.

[3] Alex Banks y Eve Porcello. *Learning React: Functional Web Development with React and Redux*. O'Reilly Media, 2020.

[4] Joseph Bonneau et al. "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes". En: *IEEE Symposium on Security and Privacy*. 2012.

[5] Kent C. Dodds. *Testing JavaScript Applications with Jest*. Disponible en línea. Self-published, 2020.

[6] Paul A. Grassi, Michael E. Garcia y James L. Fenton. *Digital Identity Guidelines - Authentication and Lifecycle Management (SP 800-63B)*. Inf. téc. National Institute of Standards y Technology, 2017. URL: `https://doi.org/10.6028/NIST.SP.800-63b` (visitado 15-07-2025).

[7] Gregory Harter. *Professional Node.js: Building JavaScript Based Scalable Software*. Wrox Press, 2018.

[8] James F. Kurose y Keith W. Ross. *Computer Networking: A Top-Down Approach*. 8.ª ed. Pearson, 2020.

[9] Alfred Menezes, Paul van Oorschot y Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[10] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015.

[11] OWASP Foundation. "OWASP Top 10: The Ten Most Critical Web Application Security Risks". En: (2021). URL: `https://owasp.org/Top10/` (visitado 10-07-2025).

[12] Tom O'Connor. "Access control and identity management in a digital environment". En: *Information Security Technical Report* 8.4 (2003), págs. 34-42.

[13] Roger S. Pressman y Bruce R. Maxim. *Software Engineering: A Practitioner's Approach*. 8.ª ed. McGraw-Hill, 2014.

[14] Ken Schwaber y Jeff Sutherland. *The Scrum Guide: The Definitive Guide to Scrum.* 2020. URL: `https://scrumguides.org` (visitado 05-07-2025).

[15] Ian Sommerville. *Software Engineering.* 10.ª ed. Pearson, 2016.

[16] William Stallings. *Cryptography and Network Security: Principles and Practice.* 7.ª ed. Pearson, 2018.

[17] Stefan Tilkov y Steve Vinoski. "Node.js: Using JavaScript to Build High-Performance Network Programs". En: *IEEE Internet Computing* 14.6 (2010), págs. 80-83.

[18] Gerd Wagner y Thomas A. Powell. *Modern JavaScript for the Impatient.* Addison-Wesley, 2019.