



Universidad
Internacional
de Valencia

Gestor de contraseñas y documentación sensible en entorno multiusuario basado en Angular y Hashicorp Vault

(Trabajo Fin de Máster)

Titulación:

Máster Universitario en
Desarrollo de Aplicaciones
y Servicios Web

Curso académico:

2022-2023 (Ed. Abril)

Alumno/a: Gómez Olivencia, Rubén

D.N.I.: 78910013-A

Director/a de TFM:

de Fez Lava, Ismael

Co-Director/a de TFM:

Simarro Moncholí, Héctor

Convocatoria:

Primera

Resumen

La gestión de contraseñas y documentación que una empresa acumula puede llegar a resultar un problema si no se sabe administrar de manera correcta. Dicha administración resulta especialmente crítica en empresas de servicios informáticos, en las que se almacena información de varios clientes, diferentes proyectos, contraseñas para distintos servicios...

En el mercado existen diferentes herramientas especializadas en la gestión de contraseñas, así como en la gestión de documentación. Sin embargo, no existen soluciones integrales que ofrezcan ambos servicios. Al respecto, muchas veces es conveniente tener la documentación de lo realizado junto con la contraseña de acceso, para así poder facilitar la administración de servicios.

Otro inconveniente de estas aplicaciones es que suelen ser de pago. Además, el usuario no tiene la certeza de cómo es la seguridad real de los datos, ya que normalmente la información no está en sus servidores y pueden existir fallos de seguridad que expongan la información.

También hay que tener en cuenta que pueden no ajustarse a las necesidades de la empresa y salvo que sea una aplicación de Software Libre, no será posible realizar modificaciones.

Considerando este escenario, el presente Trabajo Fin de Máster propone la creación de un gestor de contraseñas y documentación sensible, utilizando Angular y haciendo uso de Hashicorp Vault como herramienta de almacenamiento seguro.

Palabras clave: gestión de contraseñas, seguridad, autenticación, autorización, documentación sensible, cifrado de datos, hashicorp vault

Índice general

1. Introducción	5
1.1. Motivación del proyecto	5
2. Objetivos	6
2.1. Requisitos del proyecto	6
3. Marco tecnológico	8
3.1. Gestores de contraseñas conocidos	8
3.2. Problemas acontecidos	10
3.3. Herramientas tecnológicas utilizadas	11
3.3.1. <i>Framework</i> Angular	11
3.3.2. Hashicorp Vault	11
3.3.3. Git y despliegues automáticos	12
4. Metodología	12
4.1. Gestión del proyecto	13
4.1.1. Resultado del análisis realizado	13
4.1.2. Mapa de las historias de usuario	17
4.1.3. <i>Sprints</i> generados	18
4.2. Gestión de tareas con Trello	19
5. Resultados obtenidos	20
5.1. Diseño del interfaz	21
5.1.1. Primer boceto	21
5.1.2. Creación inicial del interfaz con un <i>framework</i> CSS	22
5.1.3. Del prototipo al código real	23
5.1.4. Interfaz <i>responsive</i>	25
5.2. Características y uso de la aplicación	25
5.2.1. Creación de un secreto nuevo	25
5.2.2. Acciones sobre secretos	26
5.2.3. Editar un secreto	29
5.2.4. Navegación por los secretos	30
5.2.5. Subir un fichero al sistema	33
5.3. Vault como <i>backend</i> de secretos	34
5.3.1. Configuración inicial de Vault	34
5.3.2. Sistema <i>backend</i> de autenticación	35
5.3.3. <i>Key/Value</i> como motor de secretos	35
5.3.4. <i>Backend</i> de almacenamiento	36
5.4. Angular: Aspectos destacables de la programación	36

5.4.1. Vistas en componentes separados	37
5.4.2. <i>Services</i> e inyección de dependencias	39
5.4.3. Programación asíncrona con <i>promises</i>	39
5.4.4. Envío de mensajes con <i>observables</i>	40
6. Conclusiones	42
7. Trabajo futuro	43
Referencias bibliográficas	45

1. Introducción

En la era digital en la que nos encontramos, guardar información es relativamente sencillo, gracias a que cada vez conseguimos tener discos duros con mayor capacidad. En caso de que un único disco duro no tenga la capacidad suficiente, existen sistemas como RAID (del inglés *redundant array of independent disks*) que permiten combinar varios discos añadiendo tolerancia a fallos en caso de rotura de alguno de ellos.

Por otro lado, existe información sensible como son las contraseñas, con las que se debe tener especial cuidado, ya que no deben ser accedidas ni utilizadas por personas que no estén autorizadas para ello.

En el ámbito personal existen distintas aplicaciones que nos permiten gestionar contraseñas, ya sea de manera integrada en los navegadores web, o a través de aplicaciones externas.

Cuando esta gestión de contraseñas se pasa al ámbito empresarial, a pesar de que también existen distintas alternativas, puede que no cuenten con características necesarias para la empresa: falta de integración con los sistemas de la empresa; en caso de ser una opción de pago, ser demasiado cara por el volumen de empleados que tenemos... Esto causa que resulte complejo decantarse por una solución existente.

Aparte de las contraseñas, también hay que considerar la información sensible que maneja la empresa, ya sea información propia o de terceras empresas (clientes o proveedores). Esa información, ya no tiene por qué ceñirse a “usuario” y “contraseña”, sino que puede ser documentación, ficheros, imágenes, esquemas de red...

Por todo ello, encontrar una aplicación que se ciña a todas las necesidades puede resultar complejo, y más si estas necesidades varían a lo largo del tiempo. También pueden surgir distintos problemas: la aplicación que utilicemos puede ser abandonada por los creadores; si es de pago, el precio o la suscripción puede variar; la posibilidad de añadir características propias no será posible salvo que sea Software Libre, ...

A lo largo de este **Trabajo Final del Máster Universitario en Desarrollo de Aplicaciones y Servicios Web** se analizará la creación de un sistema gestor de contraseñas y documentación sensible para entornos multiusuario, desarrollado con el *framework* de código abierto Angular (2016) y utilizando Hashicorp Vault (2015) como herramienta de gestión de secretos.

1.1. Motivación del proyecto

La motivación principal del proyecto es la de crear una aplicación que gestione contraseñas y documentación tratando de buscar un enfoque generalista para que pueda ser utilizada en

empresas tecnológicas.

Algunas de las aplicaciones de gestión de contraseñas más conocidas están enfocadas sólo al almacenamiento de contraseñas, y aunque permitan subir ficheros para ser cifrados, al querer realizar modificaciones, hay que descargar el fichero, realizar las modificaciones y posteriormente volver a subirlo.

En dichas aplicaciones, algunas de las cuales serán analizadas en el presente trabajo, la versión multi-usuario suele ser de pago, y no permiten la posibilidad de tener el servicio *on premise* (alojar el servicio junto con los datos cifrados en servidores propios).

2. Objetivos

El objetivo principal del presente Trabajo Fin de Máster es la creación de una aplicación que gestione contraseñas y en la que poder escribir y guardar documentación sensible, basada en un entorno web, que pueda ser utilizada en un entorno multiusuario como una empresa.

Este gestor de contraseñas deberá cumplir con unos requisitos mínimos para que pueda ser desplegado y utilizado en una empresa para mejorar la seguridad de los datos que se guardan.

2.1. Requisitos del proyecto

Como propuesta para la creación de un gestor de contraseñas propio, se han identificado los siguientes requisitos que deben cumplirse para llevar a cabo con éxito el desarrollo propuesto:

- Se quiere crear un sistema que **gestione contraseñas e información sensible**. Esta información sensible puede corresponder con imágenes, documentos, o ficheros en general que la aplicación permitirá subir.

También se podrá generar documentación en la propia aplicación a través de un interfaz **WYSIWYG** (del inglés *what you see is what you get*) que guarde la información en formato Markdown.

Markdown es un lenguaje de marcado ligero creado por Gruber y Swartz (2004), que a través de texto plano se puede obtener un documento en formato HTML (siglas en inglés de *HyperText Markup Language*) válido.

- El sistema estará basado en una aplicación web creada con el *framework* de desarrollo Angular y utilizando el lenguaje de programación TypeScript, creado por Microsoft (2012).

Para el desarrollo de la aplicación se va a tratar que el resultado obtenido sea lo más modular posible.

La modularidad va a permitir poder añadir características nuevas a la aplicación en el futuro.

De esta manera, la aplicación se podrá adaptar a posibles exigencias futuras que necesite el usuario o la empresa que haga uso de ella.

- El gestor de contraseñas debe contar con un **sistema de autenticación**. Sólo debe permitir el acceso a las personas que hayan pasado algún sistema de verificación que demuestre ser quién es.

Este sistema de autenticación no tiene por qué limitarse al clásico “usuario y contraseña”, ya que podrá adaptarse para en el futuro utilizar, de forma sencilla, otros sistemas conocidos, como por ejemplo:

- Certificados de seguridad de la capa de transporte (en inglés: *Transport Layer Security* o **TLS**).
 - Usuario y contraseña de un sólo uso basadas en tiempo (**TOTP**).
 - Tokens basados en **JWT** (o JSON Web Token , basados en *JavaScript Object Notation*).
 - Autenticación descentralizada a través del estándar abierto **OpenID**, creado por OpenID Foundation (2014).
- Debe existir un **sistema de autorización, que controle el tipo de acceso y permisos** que tenga en cuenta quién los realiza y a qué contraseñas se quiere acceder.
Estos permisos pueden basarse en **roles**, para de esta manera poder agrupar usuarios, y en base a ellos facilitar la autorización.
- Para tener un registro de lo sucedido, también es importante contar con un **sistema de auditoría**. A través de él se registrarán los accesos y operaciones que se realicen.
- La **seguridad en la transmisión y en el almacenamiento** debe ser la principal prioridad. Scarfone y Souppaya (2009) inciden en que las comunicaciones que contienen contraseñas deben ser cifradas, ya sea mediante uso del protocolo TLS o creando un túnel en la comunicación a través de una red privada virtual (o **VPN**).
En caso de que algún atacante obtuviese acceso físico al sistema de almacenamiento, de nuevo, los datos estarán cifrados, evitando la fuga de información.
- La aplicación debe ser **sencilla de utilizar**, para que al usuario final no le cueste utilizarla. De esta manera **se conseguirá que el usuario final la integre en su metodología de trabajo** a la hora de crear/usar contraseñas y guardar información sensible.
- Es importante que la aplicación sea **responsive**, es decir, **se adapte a la pantalla** donde se esté visualizando. De esta manera el usuario podrá hacer uso de ella a través de dispositivos móviles.
- Para el almacenamiento de la información y el sistema de autorización se hará uso del sistema Vault de la empresa Hashicorp.

Con todo ello, y teniendo en cuenta lo explicado en la sección **Metodologías**, cualquier ampliación a estos requisitos se podrán añadir en etapas posteriores del desarrollo.

3. Marco tecnológico

McCarney, Barrera, Clark, Chiasson y van Oorschot (2012) indican que la categoría de software donde podemos clasificar a los gestores de contraseñas es amplia y pueden contener muchas características diferentes, entre las que destacan:

- Almacenar y devolver contraseñas.
- Detectar la fuerza de las contraseñas.
- Convertir otros tipos de autenticación a contraseñas.
- Detección de formularios en webs para autorrellenado de datos.
- Aplicaciones con almacenamiento en la nube y sincronización entre dispositivos.

Estos gestores de contraseña suelen hacer uso de una contraseña maestra para proteger el “almacén de contraseñas”, pero esto puede desencadenar en dos inconvenientes:

- En el estudio de Belenko y Sklyarov (2012) más de una docena de gestores de contraseñas no proveen un nivel de seguridad apropiado.
- La contraseña elegida por el usuario puede no ser resistente a un ataque si el “almacén de contraseñas” es robado.

Es cierto que en los últimos años, cuando este tipo de software es utilizado en dispositivos móviles la contraseña maestra se puede sustituir por un reconocimiento biométrico (como puede ser reconocimiento de la huella dactilar o reconocimiento facial).

A continuación se analizarán cuatro gestores de contraseñas muy utilizados en la actualidad, los problemas de seguridad que han tenido y qué herramientas tecnológicas se han utilizado para el desarrollo del gestor de contraseñas y documentación que se propone en este Trabajo Fin de Máster.

3.1. Gestores de contraseñas conocidos

La siguiente tabla muestra una comparativa con distintos gestores de contraseñas conocidos, cuyo uso está extendido, y distintas características que pueden ser útiles dentro de una empresa:

	<u>KeePass</u>	<u>KeeWeb</u>	<u>1Password</u>	<u>LastPass</u>
Multiusuario	X	✓	X / ✓ **	X / ✓ **
Distintos sistemas de autenticación *	X	X	X / ✓ ***	X / ✓ ***
Permite guardar ficheros	✓	✓	✓	✓
Permite editar documentación	X	X	X	X
Versión web	X	✓	✓	✓
App móvil	X / ✓ ****	X	✓	✓
Versión escritorio	✓	✓	✓	✓
Informes de auditoría	X *****	X	X / ✓ **	X / ✓ **
On-premises	✓	✓	X	X
Coste	0€	0€	\$7,99 user/mes**	5,70€ user/mes**
Software Libre	✓	✓	X	X

Tabla 3.1: Comparativa de gestores de contraseñas.

* Permitir distintos sistemas de autorización como usuario+contraseña, autenticación contra Active Directory, sistema LDAP, TOTP...

** La versión para empresas ("business" en 1password).

*** La versión para empresas y sólo sistemas "single sign on".

**** Versión no oficial.

***** Se puede conseguir algo similar mediante un "trigger".

Tal como se ve en la tabla 3.1, sólo las aplicaciones que son Software Libre tienen versión *on premise*.

3.2. Problemas acontecidos

Dada la importancia de los datos que se pueden guardar en un gestor de contraseñas y documentación, es importante que los datos se encuentren a buen recaudo, ya que cualquier fallo de seguridad puede suponer el robo de dicha información.

Es por eso que los gestores de contraseñas más conocidos han sufrido distintos ataques o se les han descubierto ciertas vulnerabilidades en las que ha habido filtrado y/o pérdida de información.

Algunos ejemplos en los últimos años:

- Schappert (2023) informa que las cuentas de los usuarios que utilizan el programa **Norton Password Manager** han sido potencialmente comprometidas, obteniendo el usuario y contraseña, y habiendo obtenido nombre, apellidos, número de teléfono y la cuenta de correo de los usuarios.

La empresa ha declarado a Owen (2023) que aunque los servidores no han sido comprometidos, es posible que se hayan usado los usuarios y contraseñas para acceder a las cuentas de usuario. Se estiman que el ataque se ha realizado sobre al menos 8.000 cuentas.

- La aplicación **LastPass**, analizada previamente, ha sufrido varios ataques.

En Schappert (2022), la empresa declara que están trabajando “para comprender el alcance del incidente e identificar a qué información específica se ha accedido”. Por otro lado, Tarantola (2022) analiza que aunque la empresa haya indicado que los datos cifrados obtenidos no se pueden descifrar, no cree en la palabra de la empresa.

Para poder ver un breve resumen de los últimos ataques, Vaughan-Nichols (2022) nos muestra los últimos cuatro ataques en tres años, con enlaces a cada noticia.

- Existen varios listados de gestores de contraseñas que han sufrido vulnerabilidades, tal como nos indica Navarro (2019).

En «Which Password Managers Have Been Hacked?» (s.f.) se muestra un listado por año (desde el 2014) con varios gestores de contraseñas que han sufrido vulnerabilidades: 1password, RoboForm, Keepass, My1Login, ...

Como se puede ver, y este listado sólo es una pequeña muestra, los gestores de contraseñas son un objetivo claro de ataques. Wagenseil (2020) analiza parte de estos ataques y da pequeñas recomendaciones sobre cómo mejorar la protección (siendo la primera de la lista modificar la contraseña por una más fuerte).

3.3. Herramientas tecnológicas utilizadas

Para la creación del proyecto se ha hecho uso de distintas herramientas tecnológicas, que a su vez también han ayudado con las metodologías ágiles que se expondrán en el apartado de [Metodología](#).

3.3.1. Framework Angular

Angular es un *framework* que facilita la creación de aplicaciones web, utiliza el lenguaje de programación TypeScript creado por Microsoft (2012) y es Software Libre.

Angular permite programar haciendo uso de la arquitectura conocida como “**Modelo – Vista – Controlador**”, que permite separar la representación visual de la aplicación, el modelo de datos que se va a tener y la lógica de negocio.

Gracias a ello, la aplicación desarrollada va a ser lo más modular posible, para que de esta manera el mantenimiento, y añadir nuevas características en el futuro, se pueda realizar de manera sencilla.

3.3.2. Hashicorp Vault

Vault es el proyecto de código abierto para asegurar, almacenar y controlar el acceso a secretos y datos sensibles creado por la empresa Hashicorp (empresa conocida también por la creación de Terraform).

Se va a utilizar Vault como sistema centralizado para varios aspectos de la aplicación web que se va a desarrollar:

- **Gestión de autenticación:** Vault nos va a permitir tener un sistema de gestión de autenticación centralizado, que nos puede permitir pasar de un sistema de “usuario y contraseña”, a un sistema autenticado contra el Active Directory de la empresa.
- **Gestión de autorización:** A la hora de permitir el acceso y permisos a los secretos Vault cuenta con un sistema interno que nos ayudará a realizar dicha gestión.
- **Gestor de almacenamiento:** Se puede hacer uso de distintos sistemas de *backend* de almacenamiento: desde ficheros cifrados a distintas bases de datos (MySQL, PostgreSQL, Cassandra, ...).
- **Alta Disponibilidad:** Teniendo en cuenta que un gestor de contraseñas y documentación sensible es una pieza fundamental en una empresa, es vital que exista la posibilidad de crear un sistema en Alta Disponibilidad.

Vault permite crear, dependiendo de las necesidades, distintos tipos de sistemas en Alta

Disponibilidad multi-nodo, multi-región y con distintos tipos de sistemas de almacenamiento. Es importante analizar las necesidades para elegir la mejor opción.

Aunque Vault no está pensado inicialmente para guardar documentación, uno de sus sistemas de almacenamiento es “clave-valor”. Este ha sido el sistema elegido para guardar las contraseñas, la documentación que se va a poder generar en la propia aplicación y los ficheros que se van a poder subir.

3.3.3. Git y despliegues automáticos

Git es el sistema de control de versiones creado por Torvalds (2007) para la gestión del código fuente del *kernel* Linux. Hoy en día es el sistema más utilizado gracias a características tan importantes como la facilidad para crear ramas, su versatilidad para adaptarse al flujo de trabajo del desarrollador, integración con los entornos de desarrollo más importantes...

Gracias a los sistemas CI/CD (del inglés “*Continuous Integration/Continuous Delivery*”) integrados en las plataformas de repositorios más conocidas (GitHub o GitLab) o a través de herramientas propias (como Jenkins), realizar despliegues automatizados de las nuevas versiones permitirá que la aplicación pueda ser rápidamente utilizada por los usuarios finales.

4. Metodología

Hoy en día es conveniente hacer uso de las **metodologías ágiles** cuando se realiza la gestión de un proyecto, ya que cuentan con una serie de ventajas que van a permitir dar una respuesta más rápida y flexible ante posibles cambios durante la vida de desarrollo del producto.

Ventajas que se pueden destacar de las metodologías ágiles:

- Poder entregar resultados funcionales al cliente cada poco tiempo y así obtener *feedback* de lo realizado.
- Ser capaces de dar una respuesta ágil y flexible ante posibles cambios que puedan surgir (cambios en los requisitos, ante bajas de personal asignado al proyecto, dificultades durante el desarrollo...).
- Tratar de evitar burocracia innecesaria, para centrarnos de esta manera en el producto y sus funcionalidades.
- Tratar de involucrar al cliente (y/o a los usuarios finales) y colaborar con él, para de esta manera conseguir un mayor valor al producto.

A continuación se detallará cómo se ha realizado la gestión del proyecto que está siendo detallado en el presente Trabajo Fin de Máster.

4.1. Gestión del proyecto

Para comenzar con la gestión del proyecto debemos contar con los requisitos listados previamente y transformarlos en las conocidas como “**historias de usuario**”.

En ellas plasmaremos, en un lenguaje sencillo de entender, las funcionalidades que el producto debe tener, a las que asociaremos otros datos como son:

- Un **identificador único** para poder diferenciarlas del resto.
- Una **estimación de tiempos** de lo que creemos que podemos tardar en realizarlo.
- La **prioridad** inicial para poder ordenarlas entre ellas.
- El **tema principal** al que se relaciona la historia de usuario.
- Se añadirán posibles **dependencias** de otras historias de usuario.
- Una **descripción** de lo que se quiere conseguir.
- Unos **criterios de validación** por parte del cliente.

Teniendo en cuenta los datos que queremos plasmar en las “historias de usuario”, y tras haber realizado el análisis de los requisitos del proyecto, veremos cuál ha sido el resultado obtenido en el siguiente apartado.

4.1.1. Resultado del análisis realizado

A continuación se van a identificar las distintas “historias de usuario” obtenidas, en las que se ha ido plasmando la información necesaria que posteriormente se utilizarán para la creación de las tareas a desarrollar:

ID	Estimación	Prioridad	Tema	Dependencias
1	25 horas	1	Interfaz web	
Desarrollar “esqueleto” del interfaz web				
Descripción: Como usuario necesito un interfaz web para poder acceder a la aplicación y hacer uso de ella. Es la base del proyecto.				
Criterios de validación: Interfaz responsive , visualmente atractiva y funcional.				

Tabla 4.1: Desarrollar “esqueleto” del interfaz web.

ID	Estimación	Prioridad	Tema	Dependencias
2	10 horas	1	Gestión de usuarios	1
Crear “login” de usuario				
Descripción: Como usuario quiero poder acceder a la aplicación utilizando como sistema de autenticación un usuario y contraseña.				
Criterios de validación: Los valores introducidos serán autenticados contra el <i>backend</i> que se decida (base de datos o LDAP/Active Directory inicialmente).				

Tabla 4.2: Crear “login” de usuario.

ID	Estimación	Prioridad	Tema	Dependencias
3	10 horas	1	Gestión de Secretos	2
Crear secretos				
Descripción: Como usuario quiero poder crear un secreto para poder guardar documentación sensible y/o contraseñas.				
Criterios de validación: <ul style="list-style-type: none"> • El botón para crear siempre será visible. • Se podrá elegir la ruta (por defecto será donde nos encontramos) y el nombre del secreto. 				

Tabla 4.3: Crear secretos.

ID	Estimación	Prioridad	Tema	Dependencias
4	30 horas	1	Gestión de Secretos	2, 3
Listado de secretos				
Descripción: Como usuario quiero poder obtener un listado de los secretos existentes para poder navegar por ellos y seleccionar el que me interese.				
Criterios de validación: <ul style="list-style-type: none"> • El listado completo se visualiza mediante un árbol jerarquizado. • También se puede navegar como si fuera un explorador de archivos. • El interfaz mostrará un <i>breadcrumb</i> para indicar la ruta en la que nos encontramos. • Se puede buscar por el nombre de los secretos. 				

Tabla 4.4: Listado de secretos.

ID	Estimación	Prioridad	Tema	Dependencias
5	15 horas	2	Gestión de Secretos	2, 3, 4
Visualizar un secreto				
Descripción: Como usuario quiero acceder a un secreto para visualizar su contenido.				
Criterios de validación: <ul style="list-style-type: none"> La visualización mostrará el secreto en formato HTML. Al acceder a un secreto “bloqueado” el sistema lo debe alertar y no se permitirá editarlo. El interfaz nos mostrará la última vez que se modificó el secreto. Al visualizar el secreto aparecerán botones con distintas acciones que se podrán realizar sobre él (editar, imprimir, ver históricos, borrar). 				

Tabla 4.5: Visualizar un secreto.

ID	Estimación	Prioridad	Tema	Dependencias
6	35 horas	2	Gestión de Secretos	2, 3
Modificar un secreto				
Descripción: Como usuario quiero poder editar un secreto para realizar modificaciones.				
Criterios de validación: <ul style="list-style-type: none"> La modificación se realizará a través de un editor WYSIWYG. El editor permitirá realizar modificaciones en formato Markdown. Al editar el secreto se pondrá en estado “bloqueado”. 				

Tabla 4.7: Modificar un secreto.

ID	Estimación	Prioridad	Tema	Dependencias
7	10 horas	3	Gestión de Secretos	2, 3, 5
Borrado de secretos				
Descripción: Como usuario quiero poder borrar secretos para que desaparezcan.				
Criterios de validación: <ul style="list-style-type: none"> Antes de borrar debe existir un mensaje de confirmación. Al borrar un secreto, el árbol de secretos se debe actualizar. 				

Tabla 4.8: Borrado de secretos.

ID	Estimación	Prioridad	Tema	Dependencias
8	15 horas	3	Gestión de Secretos	2
Permitir gestionar ficheros como secretos				
Descripción: Como usuario quiero poder subir ficheros al sistema para guardarlos cifrados.				
Criterios de validación: <ul style="list-style-type: none"> • Se permite subir cualquier tipo de fichero. • Se permite descargar el fichero original. • Se debe poder visualizar los ficheros más habituales en la propia web (imágenes y ficheros PDF) sin necesidad de descargarlos. 				

Tabla 4.9: Permitir gestionar ficheros como secretos.

ID	Estimación	Prioridad	Tema	Dependencias
9	20 horas	4	Gestión de Secretos	3, 6
Versionado de secretos				
Descripción: Como usuario quiero poder ver las veces que el fichero ha sido modificado para poder ver los cambios realizados.				
Criterios de validación: <ul style="list-style-type: none"> • Poder ver el número de versiones que tiene un secreto. • Poder ver una versión concreta del secreto. • Poder ver las diferencias entre versiones. • Poder restaurar una versión concreta del secreto. 				

Tabla 4.10: Versionado de secretos.

ID	Estimación	Prioridad	Tema	Dependencias
10	5 horas	4	Gestión de Secretos	2, 5
Imprimir secretos				
Descripción: Como usuario quiero poder imprimir un secreto.				
Criterios de validación: Existe un botón para imprimir un secreto.				

Tabla 4.11: Imprimir secretos.

ID	Estimación	Prioridad	Tema	Dependencias
11	35 horas	2	Producción	1
Puesta en producción segura				
Descripción: Como usuario quiero poder acceder a la plataforma de manera segura para realizar todo lo comentado hasta ahora				
Criterios de validación: <ul style="list-style-type: none"> El acceso web tiene que ser por HTTPS (certificado válido). El servidor debe estar securizado ante accesos externos. Los servicios deben auto-arrancarse en caso de caída. Existe un registro de lo sucedido a cada secreto (auditoría). 				

Tabla 4.12: Puesta en producción segura.

4.1.2. Mapa de las historias de usuario

Dado que las historias de usuario pueden suponer la agrupación de distintas tareas técnicas que son más pequeñas, es conveniente crear un mapa que englobe todas las tareas a realizar.

El mapa resultante inicial para la gestión del proyecto es el siguiente:

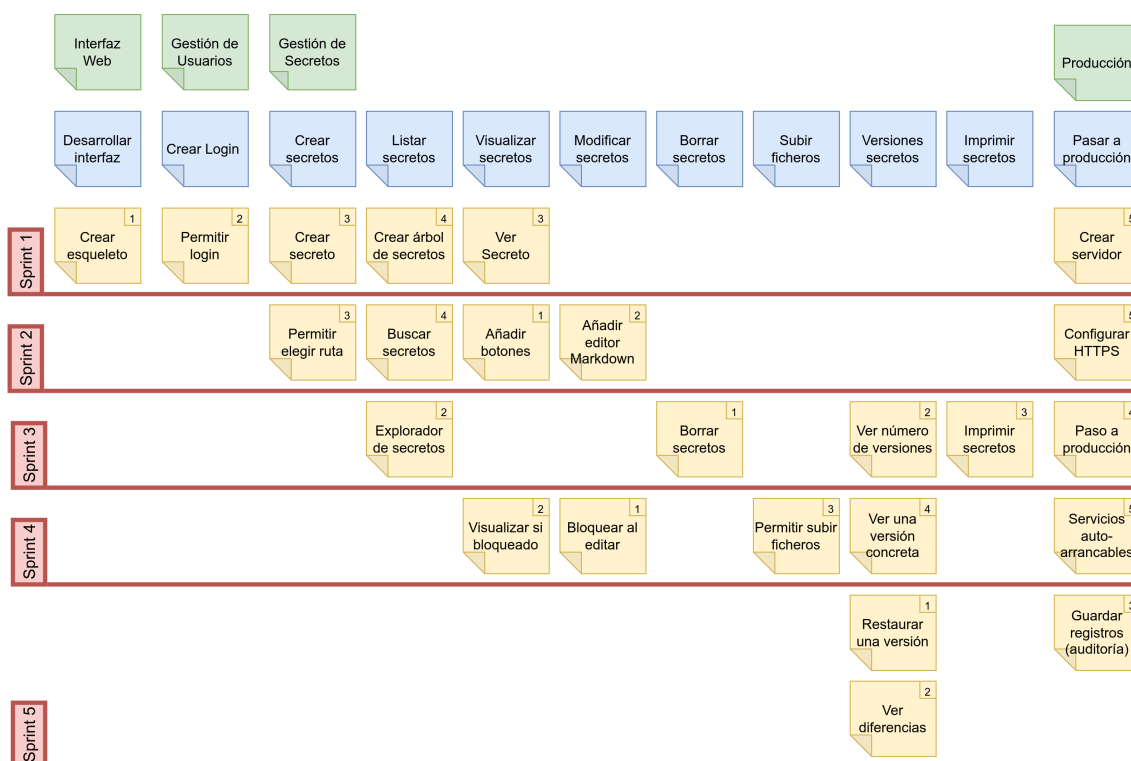


Figura 4.1: Mapa de las tareas a realizar.

Para la organización de este mapa se pueden identificar los siguientes apartados:

- Los **temas** son las características a alto nivel que va a tener la aplicación, y coinciden con el tema de las historias de usuario. En la figura 4.1 se ha diferenciado en color verde.
- Los **epics**, en color azul, son características que normalmente no pueden realizarse en un *sprint*, que a veces pueden ser divididas en distintas historias de usuarios, y que engloban en su mayoría varias tareas.
- Las **tareas**, en este caso en amarillo, son las unidades concretas de trabajo. Se ha tratado que sean lo más cortas posibles y que a nivel técnico sean auto-contenidas y aisladas.

4.1.3. Sprints generados

A la par que se ha ido creando el mapa anterior, se han diferenciado diferentes *sprints* con las tareas que deberían completarse para cada uno de ellos.

Bien es cierto que aunque haya sido realizada esa primera estimación, se podrían efectuar modificaciones en caso de que fuese necesario.

Para el primer *sprint* se han identificado las siguientes tareas que deben realizarse:

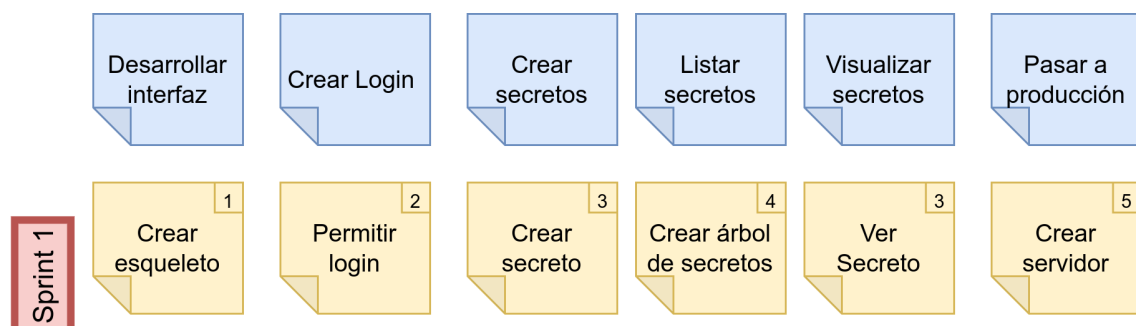


Figura 4.2: Tareas a realizar en el *Sprint 1*.

Tal como se puede ver, las tareas están representadas con su prioridad dentro del *sprint*, para así también tener claro qué tareas deben tratar de terminarse antes que otras.

Al terminar este primer *sprint*, tendremos la base del proyecto terminada, y podríamos mostrarle al cliente una primera versión con unas funcionalidades básicas.

De esta manera, podemos obtener *feedback* de lo realizado, con una base ya terminada. Ese *feedback* podremos utilizarlo como retroalimentación para el siguiente *sprint*, ya sea para realizar modificaciones de lo ya realizado (hacer cambios nuevos que el cliente pida) o para utilizarlo para las tareas ya planificadas.

Para llevar una mejor gestión de las tareas expuestas en este apartado, lo ideal es utilizar un software especializado creado con dicho objetivo, como el que vamos a analizar a continuación.

4.2. Gestión de tareas con Trello

Trello es una aplicación web, creada por la empresa Atlassian (2011), que nos permite tener un tablero virtual (denominado **Kanban**) donde se podrá añadir las tareas de nuestro proyecto, e ir moviéndolas entre distintas etapas.

Aunque cuenta con una versión de suscripción, la versión gratuita dispone de las características suficientes como para poder ser utilizado para la gestión de proyectos sin ningún problema.

Una vez definidas las tareas, tal como se ha visto anteriormente, se han añadido a Trello, en el que se han creado diferentes columnas, o fases de desarrollo. Las tareas serán movidas entre las diferentes fases de desarrollo a medida que se van realizando.

Se ha determinado contar con las siguientes fases:

- **Tareas pendientes:** donde situaremos las tareas que se deben realizar para llevar a cabo el proyecto.
- **Desarrollo:** Son las tareas que se han empezado a desarrollar.
- **Testing:** Son las tareas que se dan por terminadas en el desarrollo y que pasan a una fase de testeo. En algunos casos puede ser directamente testado por el cliente para dar su feedback (por ejemplo para el interfaz web), aunque lo habitual es que el testeo lo realice otra persona del proyecto (que no haya sido quien ha hecho el desarrollo).
En caso de que una tarea no pase esta fase, se anotará los inconvenientes y volverá a la columna de **desarrollo**.
- **Producción:** Una vez el *testing* ha terminado, se puede dar por terminada la tarea y podrá ser incluida en producción.

Tal como se puede ver en la Figura 4.3, cada tarea se representa como una pequeña tarjeta, apareciendo como un listado de todas ellas en la columna correspondiente en las que están situadas (en este caso todavía en la lista de tareas pendientes).

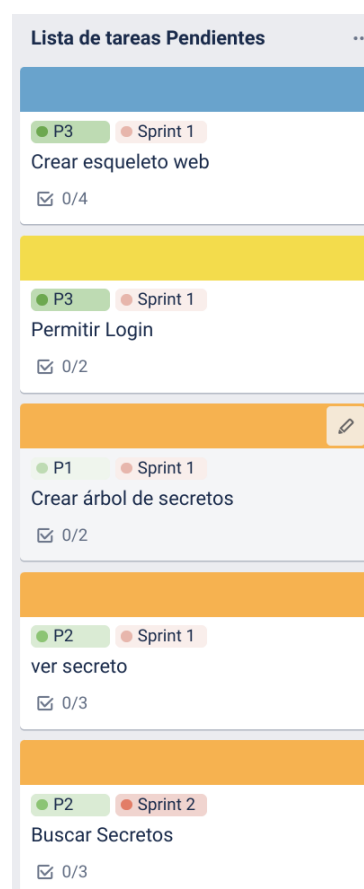


Figura 4.3: Tareas pendientes en Trello.

Aparte, y para que a nivel visual sea más sencillo de determinar de qué tipo es cada tarea, a cada una de ellas se les ha asignado un color teniendo en cuenta el tema al que pertenecen, y varias etiquetas que representan el *sprint* y la prioridad dentro del *sprint*. De esta manera, Trello nos permitirá realizar búsquedas o visualizar las tareas con la etiqueta que nos interese.

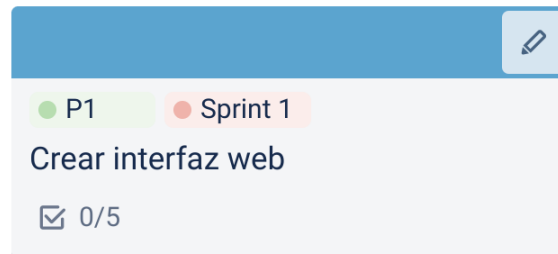


Figura 4.4: Detalle de la tarea.

Por último, cada tarea puede tener un listado de pequeños ítems. Estos han sido detallados para que a la hora de desarrollar se tenga en cuenta qué es lo que se debe conseguir, y de esta manera dar por finalizada la tarea, tal como se puede ver en la figura 4.5.

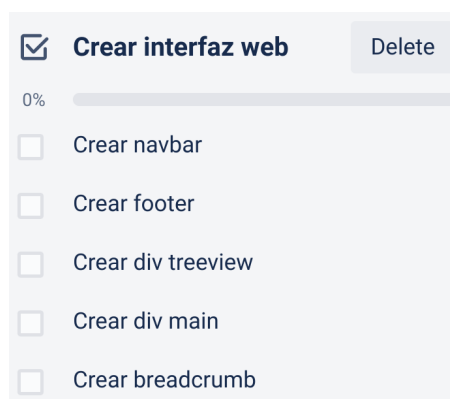


Figura 4.5: Ítems de la tarea.

Tras terminar la tarea, se pasará a la columna **testing**, para de esta manera, tal como se ha dicho previamente, comenzar con la fase de testeo.

5. Resultados obtenidos

A continuación se va a exponer los pasos realizados para el diseño y la creación de la aplicación teniendo en cuenta los requisitos del proyecto y las historias de usuario detalladas previamente.

Se van a diferenciar distintas secciones generales:

- Diseño del **interfaz**.

- **Características y uso** de la aplicación.
- Vault como *backend* de secretos.
- **Angular**: Aspectos destacables de la programación.

Cada una de estas secciones serán tratadas en profundidad a continuación, dando lugar a subsecciones para diferenciar la información tratada en las mismas.

5.1. Diseño del interfaz

La interfaz web de la aplicación es el punto de entrada del usuario, por lo que resulta un aspecto clave, ya que puede determinar el éxito o el fracaso en el uso de la aplicación.

A la hora de hacer el diseño, se han tenido en cuenta:

- **Facilidad de uso**: Es importante que la aplicación tenga una buena usabilidad, para que el usuario final no requiera de ningún esfuerzo a la hora de utilizarla.
- **Diseño funcional**: Continuando con el punto anterior, se ha tratado de realizar un diseño funcional, para que cada parte de la interfaz se sepa para qué va a ser utilizada. Para facilitar dicha tarea, se ha trabajado tratando de conseguir un diseño **minimalista**.
- Se ha realizado un **diseño adaptable a la pantalla** (también conocido como diseño *responsive*) para facilitar el uso de la aplicación en dispositivos móviles.

5.1.1. Primer boceto

Para la creación de la interfaz se decidió crear un primer prototipo a boli en un cuaderno, tratando de plasmar las ideas generales, y la colocación inicial de los elementos con los que los usuarios van a interactuar.

Este primer boceto sirve para asentar las historias de usuario y las distintas acciones que los usuarios van a poder realizar en la aplicación final. Es una manera rápida para empezar a tomar consciencia del proyecto y cómo queremos encauzar la interfaz.

Tal como se puede ver en la figura 5.1 existen zonas diferenciadas marcadas en color azul, donde también aparece en color rojo el nombre para cada zona. Ya en este primer boceto se estaban identificando lo que posteriormente serían distintos componentes independientes.

En rojo, en la parte central de la figura 5.1 también aparece cómo sería el login, que daría paso al interfaz general. Por último, se pueden diferenciar, también en la parte central, nombrados como "Opt1" y "Opt2", las vistas que dependerán de qué estemos haciendo: si navegamos por los secretos o si estamos visualizando/editando un secreto.

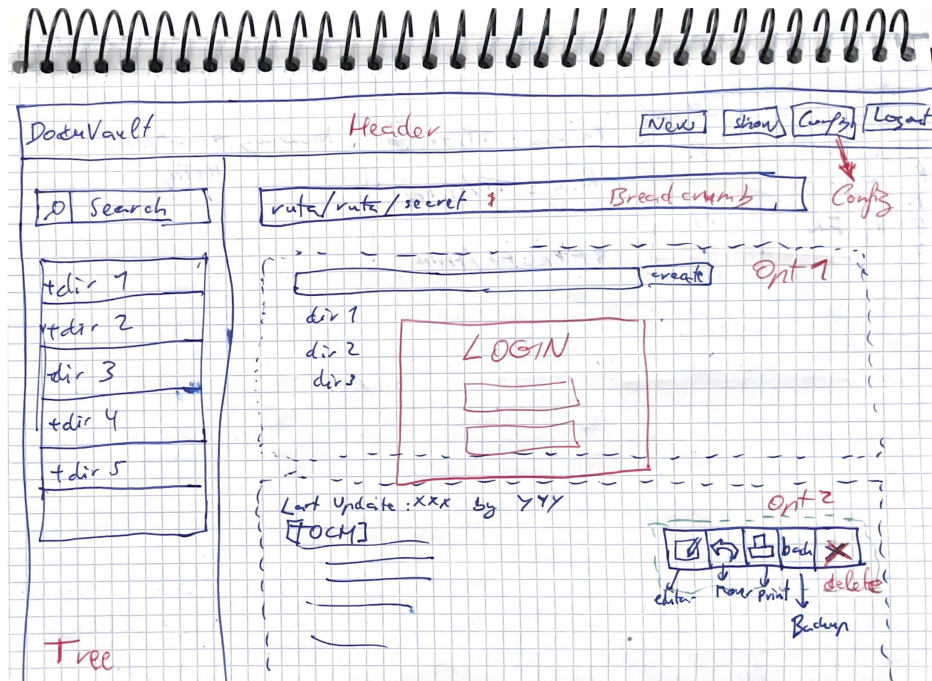


Figura 5.1: Boceto del interfaz.

Ya en este primer boceto se tenía una idea de cómo debía ser el interfaz final.

5.1.2. Creación inicial del interfaz con un *framework* CSS

Una vez se tuvieron claros los aspectos generales del boceto, se realizó un primer prototipo haciendo uso de un *framework* CSS.

El *framework* elegido ha sido Bootstrap, creado por Twitter (2011), ya que va a permitir crear un diseño funcional, de manera sencilla, y dispone de distintas características que van a resultar útiles durante el uso de la aplicación (como por ejemplo los “*modal*” y el “*breadcrumb*”).



Figura 5.2: Boceto del interfaz con Bootstrap y datos estáticos.

Bootstrap ha permitido crear un código sencillo, pero visualmente atractivo, y que posteriormente será reutilizado para el interfaz final.

5.1.3. Del prototipo al código real

Tal como se ha comentado previamente, Bootstrap permitió crear el prototipo del interfaz de manera rápida y sencilla. Aunque el aspecto visual a grandes rasgos no ha variado en exceso, a medida que se iba avanzando en el proyecto se han ido realizando pequeñas modificaciones.

El interfaz final se puede descomponer en distintos apartados que cumplen con distintas funcionalidades.

La pantalla de **login** nos permite introducir el usuario y la contraseña del mismo para acceder a la aplicación.

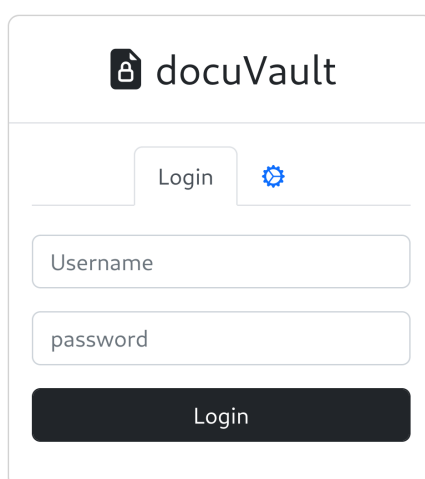


Figura 5.3: Interfaz para el login.

En esta ventana de **login** se ha añadido una pestaña extra con el símbolo del engranaje que permite modificar la URL del servidor al que nos queremos conectar, tal como se ve en la figura 5.4.

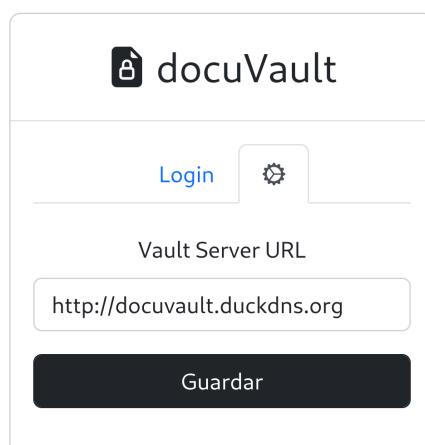


Figura 5.4: Configurar URL del servidor.

Tras realizar la autenticación la aplicación nos llevará al directorio principal donde podremos navegar a través del interfaz para acceder a los secretos que nos interese.

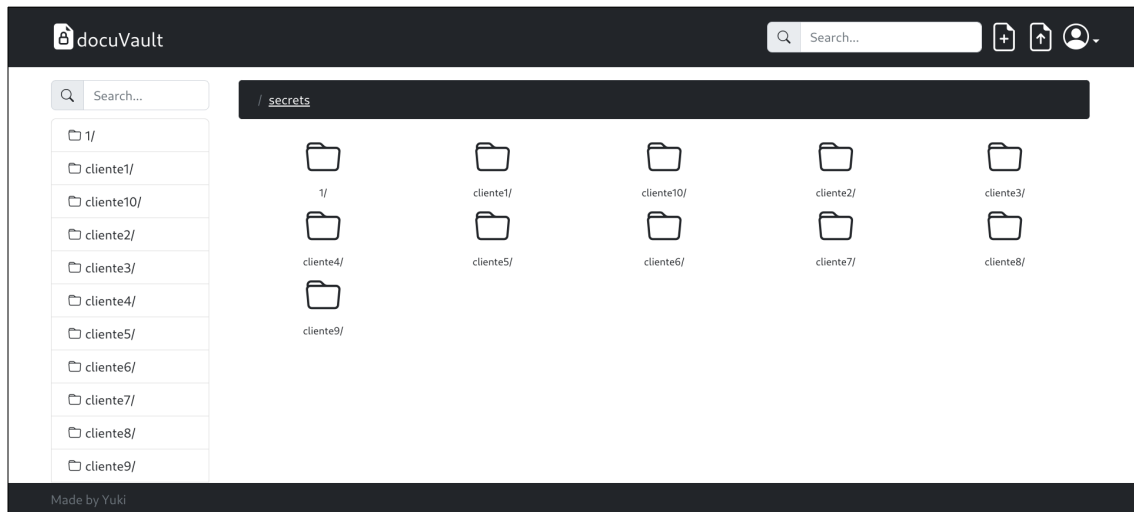


Figura 5.5: Interfaz principal de la aplicación.

Al acceder al secreto que nos interesa, el interfaz nos muestra el documento descifrado junto con unos botones con los que realizar ciertas acciones sobre los secretos.

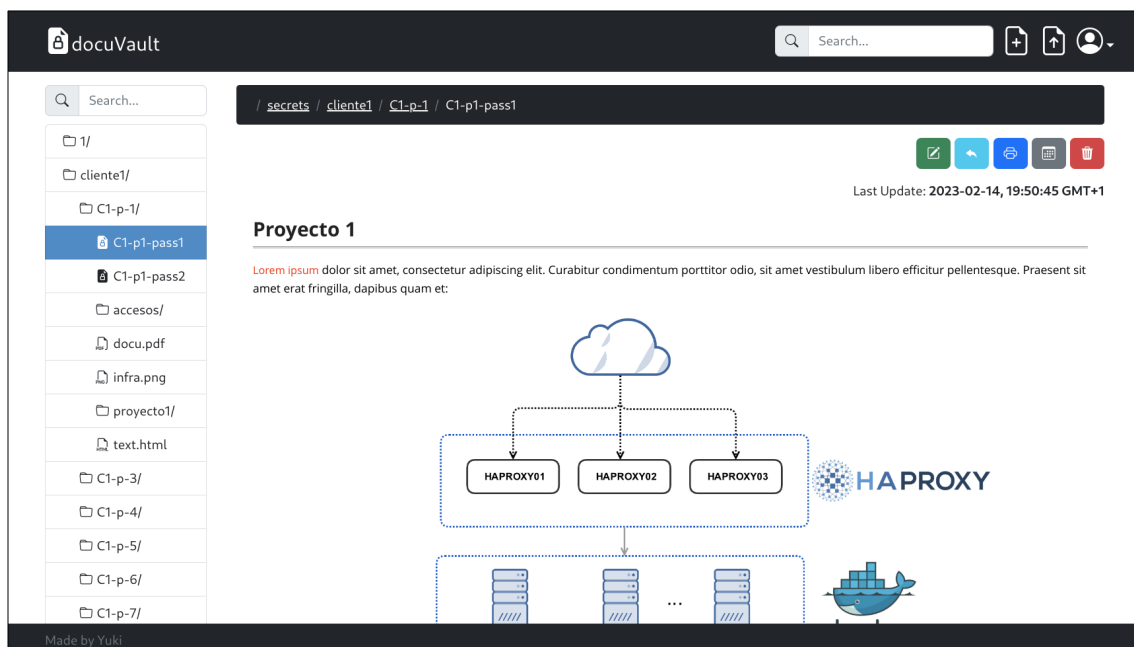


Figura 5.6: Interfaz al visualizar un secreto.

Hasta ahora se ha comprobado cómo es el aspecto visual en pantallas grandes, donde todos los aspectos de la interfaz caben en ella.

5.1.4. Interfaz responsive

Para facilitar el uso de la aplicación en distintos tipos de pantallas, se ha hecho que el interfaz se adapte al tamaño donde se está visualizando la aplicación. Esto va a permitir que el usuario pueda acceder a través de su teléfono móvil, lo que puede ser útil en entornos de empresa donde se tengan que realizar salidas a clientes.

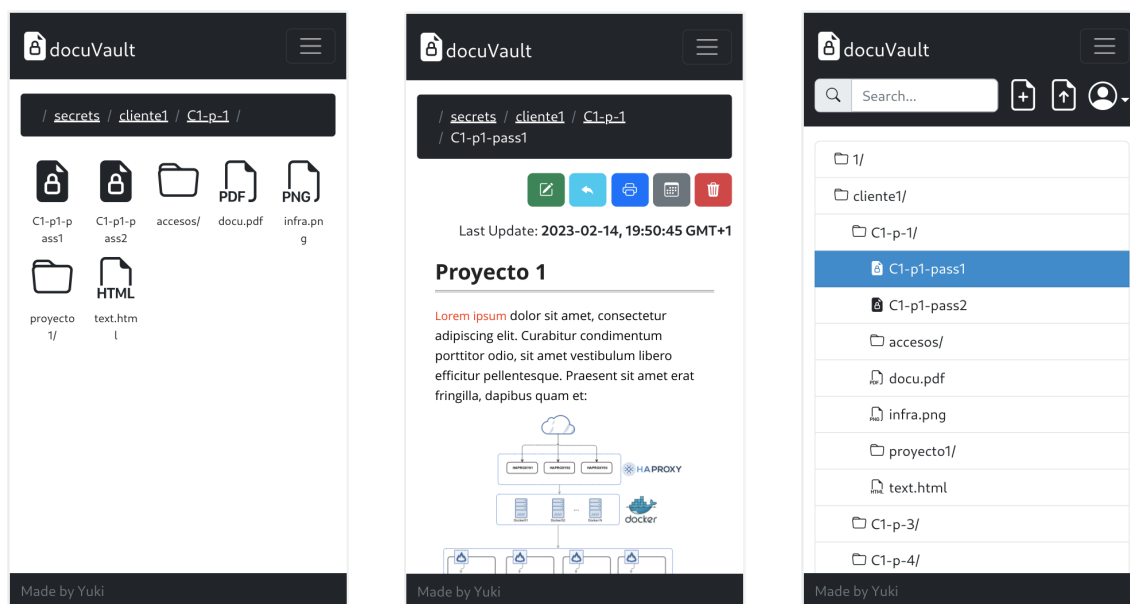


Figura 5.7: Interfaz en modo responsive.

Tal como se puede ver, para mejorar la navegación, el interfaz que muestra en modo árbol la jerarquía de secretos se oculta, de esta manera navegamos a través de los secretos como un explorador de archivos.

5.2. Características y uso de la aplicación

Teniendo en cuenta los requisitos planteados al inicio del proyecto, siendo uno de ellos que la aplicación debe ser fácil de utilizar, se ha conseguido que las distintas características que dispone la aplicación y el uso de la misma recuerde a aplicaciones que los usuarios ya hayan utilizado.

A continuación se van a detallar distintas características de la aplicación y cómo se crearon pensando en el usuario final.

5.2.1. Creación de un secreto nuevo

La creación de secretos es la característica principal de la aplicación, ya que con ello se va a conseguir guardar la información sensible que nos interese.



En la cabecera de la aplicación existe un icono con el que se va a permitir crear un nuevo secreto en el sistema.

Al hacer *click* sobre el icono, aparecerá un *modal* (una pequeña ventana emergente en el interfaz) donde preguntará por la ruta donde se quiere crear el secreto.

Al aparecer el *modal* por defecto mostrará la ruta en la que el usuario se encuentra en ese instante.

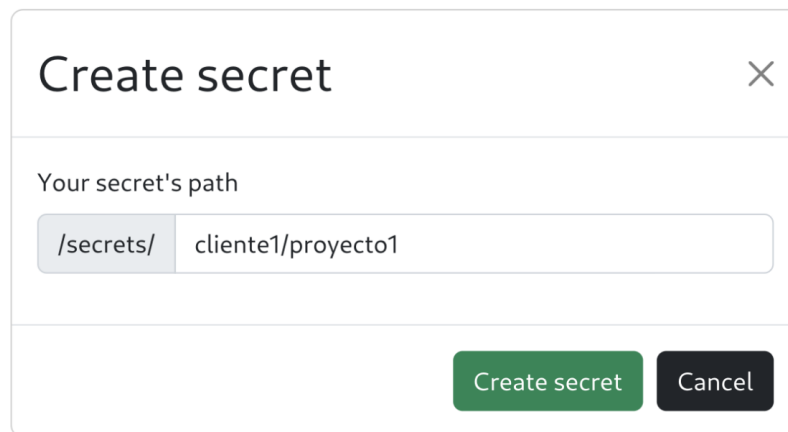


Figura 5.8: *Modal* para crear un secreto.

Si en la ruta que indica el usuario termina en "/" el botón para crear el secreto permanecerá deshabilitado, ya que una "barra" al final del nombre identifica un directorio, al igual que sucede en sistemas UNIX.

Al hacer *click* en el botón de creación del secreto la aplicación creará un secreto nuevo en la aplicación y redirigirá al usuario a la ruta donde se ha creado.

5.2.2. Acciones sobre secretos

Una vez creado el secreto, o una vez que se navegue a un secreto creado previamente, se podrán realizar distintas acciones sobre el mismo. Estas acciones dependerán de los permisos que se tengan sobre el secreto, pero a continuación se van a detallar todos ellos.

Las acciones que el usuario podrá realizar se identifican a través de distintos iconos que aparecen en el interfaz y que se van a detallar a continuación.

Editar secreto



Una de las acciones principales es la de **editar el secreto**, ya sea para añadir, modificar o borrar información en el mismo. Esta edición se hará a través de un editor completo WYSIWYG, tal como se explica en el apartado "**editar un secreto**".

Mover secreto



Una vez creado un secreto, para realizar una reestructuración de los secretos, puede interesar moverlo. Al hacer *click* en este botón se abrirá un *modal* en el que aparecerá la ruta actual y donde habrá que indicar la nueva ruta.

Modal titled "Move secret" with a close button (X). It contains two sections: "Secret's path" and "New path". Each section has a text input field. The "Secret's path" field contains "/secrets/ cliente1/proyecto1". The "New path" field contains "/secrets/ cliente1/proyecto1". At the bottom right, there are two buttons: "Move file" (green) and "Cancel" (grey).

Figura 5.9: Elegir nueva ruta para un secreto.

Imprimir secreto



En algunas ocasiones puede ser interesante **imprimir la documentación guardada**. Este icono abrirá el asistente para imprimir del navegador y mostrará una previsualización, en la que desaparecen partes del interfaz (la vista en árbol y los botones de acción) para mejorar el formato.

Visualizar histórico del secreto



Como más adelante se explicará, cuando un secreto es modificado **se guarda un histórico** del mismo. A través de este botón se pueden ver las distintas versiones del mismo e ir a una versión anterior.

Modal titled "Secret's History" with a close button (X). It contains a list of three items, each representing a version of the secret:

- [v3: 2023-02-14, 18:13:36 GMT+1](#)
- [v2: 2022-12-28, 10:40:57 GMT+1](#)
- [v1: 2022-11-02, 01:00:25 GMT+1](#)

Figura 5.10: Histórico de un secreto.

De esta manera, se podrá ver cualquier cambio que haya sufrido el secreto a lo largo del tiempo. Al hacer *click* en alguno de los enlaces para visualizar alguna versión anterior, aparecerá un mensaje encima del secreto para indicarlo:

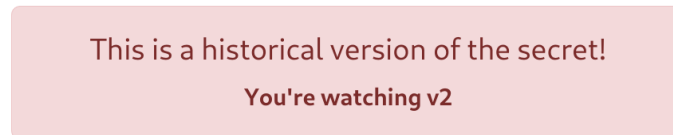


Figura 5.11: Mensaje indicando la versión del secreto.

Desbloquear secreto



Cuando un usuario entra a modificar un secreto, éste se pondrá en modo “bloqueado”. Si, por alguna razón, abandona la aplicación antes de guardar los cambios el secreto se mantiene en dicho estado. A través de este botón **un usuario administrador podrá desbloquear el secreto**.

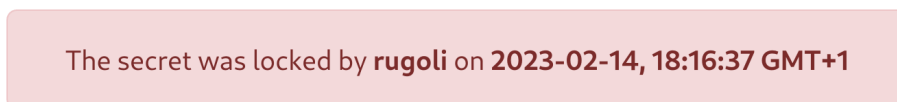


Figura 5.12: Aviso del estado bloqueado de un secreto.

Borrar secreto



Un administrador, o usuario que tenga permisos para ello, podrá borrar un secreto con todas las consecuencias que ello conlleva: perder los datos, el histórico, no poder recuperarlo...

Debido a que es una acción que no se va a poder deshacer, se requerirá que el usuario realice una confirmación antes de que el secreto sea borrado.

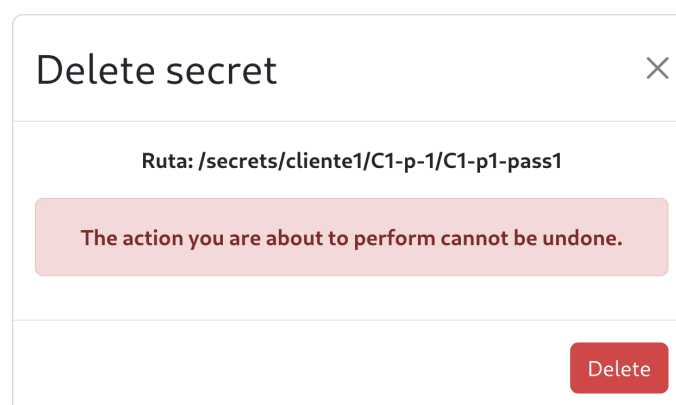


Figura 5.13: Confirmación al borrar un secreto.

Una vez el usuario haga *click* en el botón de confirmación, el secreto se borrará y la aplicación redirigirá al usuario al directorio superior de donde se encontraba el secreto borrado.

5.2.3. Editar un secreto

Dado que esta acción es la más importante de la aplicación, se va a profundizar a continuación en las características del sistema de edición de secretos.

Tal como se ha visto en el apartado anterior, para editar un secreto existe un botón dedicado a tal acción. Al hacer *click* sobre él, los botones de acción se ocultan y aparece el editor con los datos del secreto:

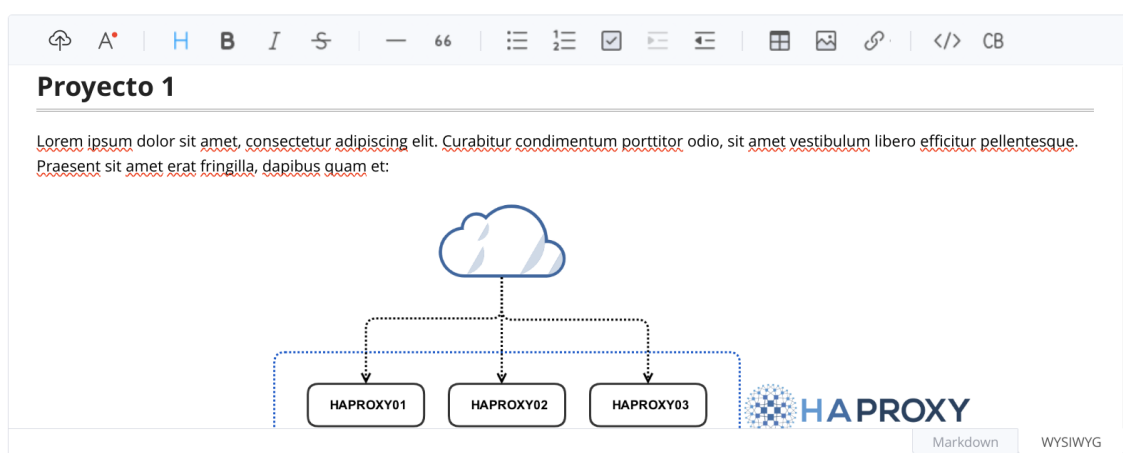


Figura 5.14: Editor WYSIWYG.

El editor elegido es **Tui.editor**, creado por NHNCloud (2015), ya que es un proyecto con licencia **MIT** (en inglés *Massachusetts Institute of Technology*), creado en el lenguaje Typescript y que cumple con los requisitos para el proyecto.

Entre las características que tiene el editor, se pueden destacar:

- Interfaz **WYSIWYG** completo que permite editar y guardar el documento generado en formato Markdown.
- Permite añadir imágenes arrastrándolas sobre el interfaz.
- Se puede expandir las funcionalidades a través de **plugins**. Para este proyecto se están utilizando dos:
 - Posibilidad de **añadir color al texto**, expandiendo las posibilidades de Markdown, ya que por defecto este sistema no lo permite.
 - Mejoras en el **resaltado de sintaxis** a la hora de añadir bloques de código fuente o de configuración.


Esto resulta es muy útil cuando se necesita guardar en la documentación configuración de algún tipo o fragmentos de código fuente. El plugin hace uso de [PrismJS](#) que soporta 297 lenguajes.

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Figura 5.15: Ejemplo de resaltado de sintaxis.

- Posibilidad de añadir iconos propios a la barra de herramientas del editor. Esta característica ha sido clave ya que era necesario poder crear al menos un botón para guardar el secreto y cerrar el editor.

El icono añadido tiene el siguiente aspecto . Al hacer *click* sobre él, el secreto será guardado, el editor se cerrará y la aplicación nos redirigirá para que visualicemos el nuevo aspecto del secreto recién editado.

5.2.4. Navegación por los secretos

A la hora de navegar por las contraseñas y documentación se ha hecho uso de un sistema jerárquico que simula las carpetas de un explorador de archivos de cualquier sistema operativo actual.

Los usuarios están acostumbrados a hacer uso de cualquier tipo de ficheros y a ordenarlos en sus sistemas de almacenamiento, creando una jerarquía decidida previamente.

Dado que la aplicación se ha creado teniendo en cuenta las exigencias de una empresa para lidiar con contraseñas y documentación, se creía conveniente simular este sistema.

Para ello, se han creado 3 sistemas diferenciados en el interfaz, que funcionan en sincronía. Esto quiere decir, que al utilizar cualquier de ellos, hará que se actualice el resto.

Vista de árbol

Es una visualización jerárquica de la información, que se ha situado en el lado izquierdo del interfaz. Este apartado visual siempre será visible en una pantalla de grandes dimensiones, como las de un ordenador de sobremesa.

Si por el contrario, la aplicación se está visualizando a través de un teléfono móvil, el árbol permanecerá oculto, aunque se podrá desplegar a través del botón de la parte superior derecha.

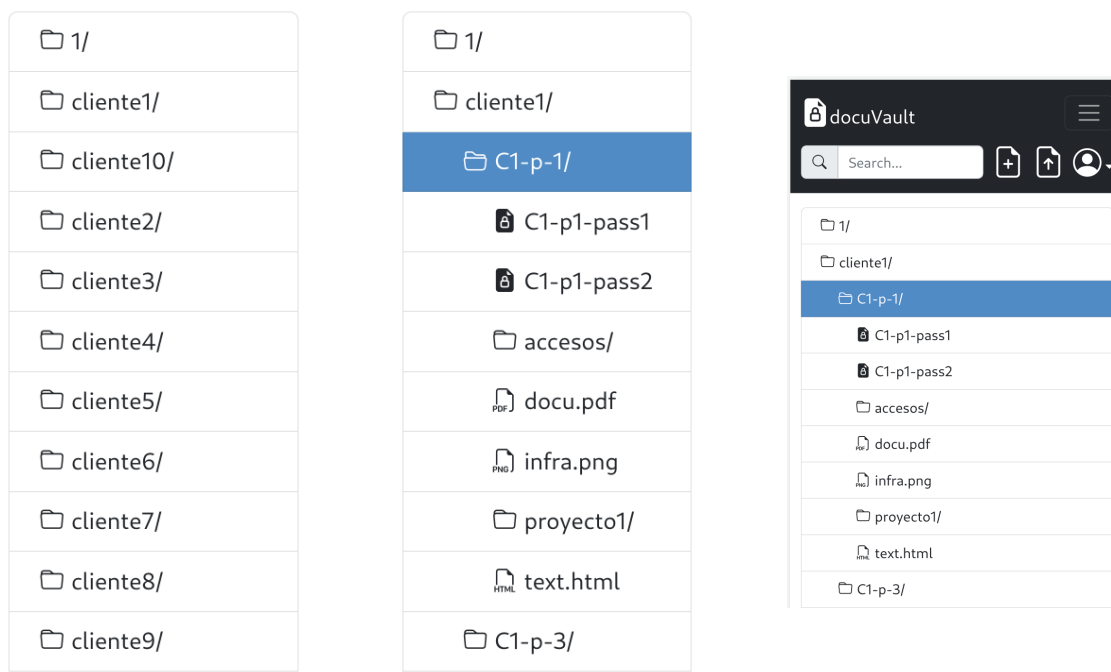


Figura 5.16: Detalles de la "vista de árbol" en modo normal y responsive.

En la vista de árbol se pueden diferenciar:

- **Ramas:** Son los nodos que a su vez contiene otros nodos. En un explorador de ficheros serían los directorios.
- **"Hojas":** Lo que en un explorador de archivos serían los ficheros.

Tal como se puede ver en la figura 5.16, al seleccionar cualquiera de los nodos, se resaltará la opción elegida con un color azul y se mostrará su contenido en caso de ser una rama.

Explorador de ficheros

Es el método más habitual a día de hoy para navegar por un sistema de ficheros en los sistemas operativos modernos.



Figura 5.17: Detalle del explorador de ficheros.

Para identificar el tipo de ficheros que se están visualizando se han identificado mediante distintos iconos. Los iconos utilizados son:



Es un directorio, y al igual que sucede en un sistema de ficheros, puede contener a su vez otros ficheros y/o directorios.

Al hacer *click* sobre él, se visualizará su contenido. Esto afectará a la vista de árbol, que se actualizará para mostrar el directorio desplegado.



Es un secreto creado y editado a través de la aplicación. Este icono nos identifica que al hacer *click* visualizaremos su contenido, y posteriormente podremos realizar otras acciones como editarlo, imprimirlo, borrarlo...



Existen distintas representaciones de iconos para identificar a ficheros que se han subido a la aplicación para ser cifrados y almacenados en ella.

Dependiendo del tipo de fichero (identificado mediante la extensión) mostrará el tipo de fichero que es junto al icono. Existen iconos para las extensiones más habituales (png, doc, pdf, jpg, docx, zip, ...).

Tal como se ha venido explicando hasta ahora, simulando una característica que el usuario ya conoce se va a conseguir que el usuario utilice la aplicación para navegar y guardar la información, consiguiendo que la seguridad de la empresa mejore.

Navegación “miga de pan”

La “miga de pan” (en inglés **breadcrumb**) es una técnica de navegación que se utiliza en distintos tipos de interfaces gráficas de usuario para mostrar el camino recorrido.

Dependiendo de si estamos navegando en un directorio o si estamos en un fichero, al final aparecerá un símbolo “/” o no.

/ secrets / cliente1 / C1-p-1 /

/ secrets / cliente1 / C1-p-1 / docu.pdf

Figura 5.18: Detalle de la navegación con “miga de pan”.

Para poder volver sobre nuestros pasos, o para ir hacia arriba en la jerarquía, cada apartado de la “miga de pan” es un enlace, por lo que al hacer *click* sobre cualquiera de ellos nos llevará a su correspondiente directorio.

5.2.5. Subir un fichero al sistema

Dado que la creación y edición de secretos puede resultar limitante, la aplicación permite la posibilidad de subir un fichero para que sea guardado y securizado.



En la cabecera de la aplicación existe un icono que va a permitir subir a la aplicación un fichero de cualquier tipo. Al hacer *click* sobre el icono, aparecerá un *modal* que preguntará por la ruta donde se quiere guardar el fichero y para elegir el fichero del sistema anfitrión.

Figura 5.19: Modal para subir un fichero.

Al igual que sucedía con la ventana para crear un secreto, este *modal* también cuenta con un sistema de validación que no activará el botón hasta que no se elija del sistema un fichero o una ruta correcta.

Una vez la aplicación ha guardado el fichero de forma segura se nos redirigirá a la ruta donde se ha guardado. Para poder visualizar el fichero sin tener que descargarlo, el sistema detecta (a través del “**MIME Type**”, del inglés *Multipurpose Internet Mail Extensions*) si es un fichero que se puede mostrar, y de esta manera lo visualizará.

Los ficheros que automáticamente se muestran son:

- Ficheros **PDF**: El fichero se abrirá a través del lector de documentos del navegador, lo que posibilita su visualización o descarga.
- Ficheros de tipo **imagen**: Si el “MIME Type” es de tipo “**data:image**”, la aplicación creará un elemento en el HTML para poder visualizarlo.

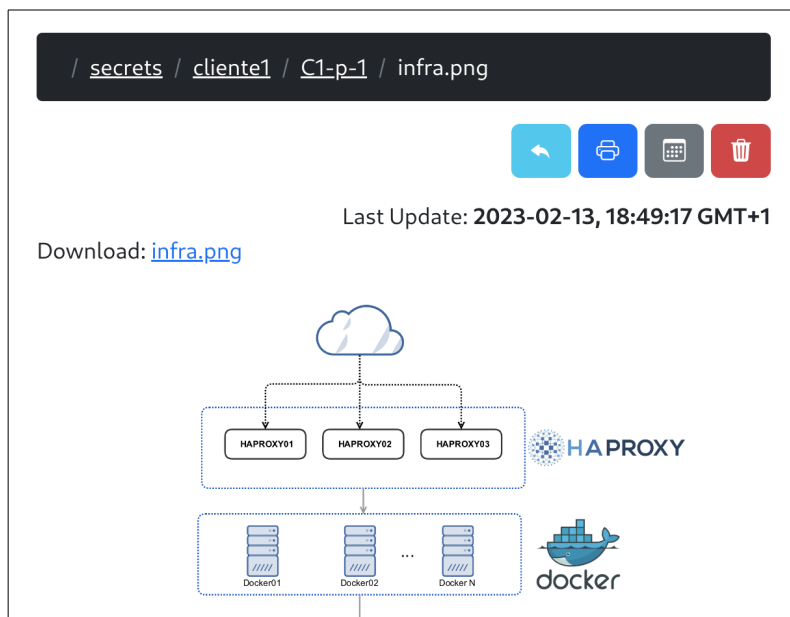


Figura 5.20: Visualización de un fichero subido a la aplicación.

Aparte, y tal como se puede ver en la imagen superior, aparecerá un enlace para poder descargar el fichero subido, lo que permitirá recuperar el archivo sin cifrar.

5.3. Vault como *backend* de secretos

Ya se ha comentado previamente cómo Vault es un proyecto para asegurar, almacenar y controlar el acceso a secretos. La aplicación desarrollada en este Trabajo Fin de Máster explota esas funcionalidades y las lleva un paso más lejos para adaptarlas a un entorno de documentación.

La comunicación con Vault se realiza a través de su API (en inglés *application programming interface*), por lo que el servicio debe de estar funcionando y en parte configurado para que la aplicación funcione sobre él.

Tal como se lleva destacando a lo largo del documento, la aplicación creada trata de buscar ser una herramienta que pueda servir en una empresa como sistema centralizado para la securización de contraseñas y documentación. Es por eso que debe de realizarse una configuración previa para que la aplicación funcione.

5.3.1. Configuración inicial de Vault

Dado que Vault va a ser el sistema que se va a encargar del almacenamiento de los secretos es vital que su instalación y configuración inicial sea correcta.

Los pasos que deben realizarse están explicados en su documentación oficial, pero a continuación se van a simplificar rápidamente:

- Crear un **servicio de inicio automático** para que en caso de que el servidor se reinicie, el servicio arranque de manera automática. En sistemas GNU/Linux se puede hacer a través de Systemd.
- **Configurar Vault**: Teniendo distintos puntos a tener en cuenta que se van a detallar en los apartados “*sistema backend de autenticación*” y “*backend de almacenamiento*”.
- **Inicializar Vault**: Al inicializar Vault se generarán el **token inicial de root** que es el que permite la administración completa del servicio. Es fundamental guardarlo a buen recaudo. También se generan las **claves de unseal** para abrir el acceso a los secretos. Cuando Vault arranca no es posible acceder a los secretos hasta que no se ha realizado el proceso de *unseal*.

A continuación se van a detallar las decisiones que una empresa puede tomar a la hora de configurar Vault para que se adapte mejor a sus necesidades.

5.3.2. Sistema *backend* de autenticación

En una empresa es habitual hacer uso de un sistema centralizado de autenticación, para que todos los usuarios se generen en la misma plataforma.

Hoy en día, los métodos más habituales suelen ser:

- **Directorio Activo de Windows Server**: en empresas donde se hace uso de Windows como sistemas de escritorio, es habitual que el sistema de autenticación esté centralizado y junto con él se hagan usos de sistemas de seguridad como GPOs (*Group Policy Object*), carpetas compartidas, ...
- **LDAP**: Del inglés *Lightweight Directory Access Protocol*, es un sistema más estandarizado que el anterior, y también hace uso de un sistema centralizado de autenticación. LDAP ha sido utilizado en servicios como Red Hat Directory Server y Apache Directory Server.
- **Gestión de usuarios local**: En caso de que no se quiera integrar Vault con sistemas internos de la empresa, Vault cuenta con su propia gestión de autenticación para usuarios.

Cada empresa tendrá que decidir cuál es el mejor método que se adapta a ella, y adaptar la aplicación a ella para que haga uso del *backend* seleccionado.

5.3.3. Key/Value como motor de secretos

Para la interacción con los secretos se ha decidido hacer uso del sistema KVv2 que tiene integrado Vault. Este sistema nos ofrece:

- Sistema de interacción **clave-valor**, habitual en bases de datos no relacionales.
- Interactuar con los secretos de manera sencilla a través de la API.
- Modificar los metadatos para poder guardar un histórico de lo sucedido.
- Tener un sistema de versiones por cada secreto.

Este motor de secretos KV2 es la característica más importante utilizada por parte de la aplicación a la hora de interactuar con Vault.

5.3.4. *Backend* de almacenamiento

Este es otro punto que la empresa debe tener en cuenta a la hora de configurar Vault, ya que de ello dependerá cómo se almacenan los secretos en el servidor final (aunque siempre se realiza de manera cifrada).

Existen distintos *backends* de almacenamiento y dependiendo de las necesidades se deberá optar por uno u otro:

- **Filesystem**: Es el método más sencillo de implementar, ya que Vault almacena los secretos en formato de ficheros en el servidor.
- **Bases de datos**: Dependiendo del sistema de base de datos utilizado podríamos hacer uso de su sistema en alta disponibilidad para tener un sistema escalable (como se puede realizar con MySQL).
- **S3**: Quizá interese almacenar los secretos en un *bucket* S3 de Amazon, y de esta manera no depender de almacenamiento local.

Estos son sólo unos ejemplos de los distintos sistemas de almacenamiento que se pueden utilizar. La empresa debe tomar la decisión de cuál elegir y realizar la configuración en Vault, aunque en caso de dudas se recomienda hacer uso de la opción Filesystem. La opción elegida será transparente para la aplicación que se ha creado.

5.4. Angular: Aspectos destacables de la programación

Tras explicar cómo funciona la aplicación desde el punto de vista del usuario, se va a profundizar en algunos aspectos técnicos realizados durante el desarrollo de la misma.

Tal como se ha dicho, la aplicación está programada utilizando el *framework* Angular y por tanto se ha hecho uso del lenguaje TypeScript.

A continuación se van a detallar aspectos importantes y características utilizadas durante el desarrollo de la aplicación.

5.4.1. Vistas en componentes separados

Durante la creación del interfaz gráfico se ha dividido en distintos componentes para, de esta manera, diferenciar el código de la vista en cada uno de ellos.

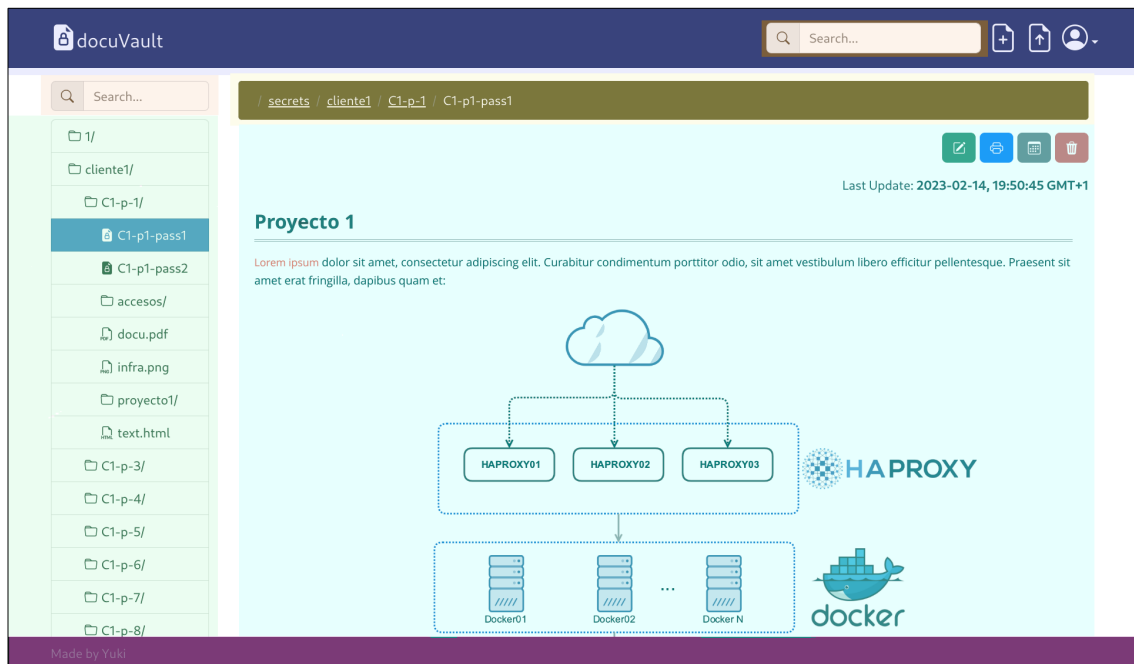


Figura 5.21: Interfaz principal con los componentes coloreados.

En la imagen se han coloreado los distintos componentes que existen a la hora de visualizar un secreto:

- **Cabecera:** La cabecera de la aplicación una vez el usuario se ha logueado.
- **Cajón de búsqueda:** Este componente se reutiliza en dos partes del interfaz: en la cabecera y encima de la vista de árbol de secretos.
- **Vista de árbol:** Este componente se oculta cuando el interfaz está siendo visualizado en una pantalla pequeña.
- **Breadcrumb** o “miga de pan”: Este es el componente que nos visualiza la ruta en la que nos encontramos.
- **Vista del secreto:** Esta es la vista principal cuando se está visualizando el secreto. En caso de que se esté editando el secreto, la vista mostrará el editor.
- **Pie de página:** El pie de página de la aplicación.

Aparte de los componentes explicados previamente, existen otros dos:

- **Login:** Es el encargado de mostrar el *modal* del *login* cuando no se está logueado en la aplicación.
- **Explorador de ficheros:** Es el componente que visualiza los secretos como si fuese un explorador de ficheros.

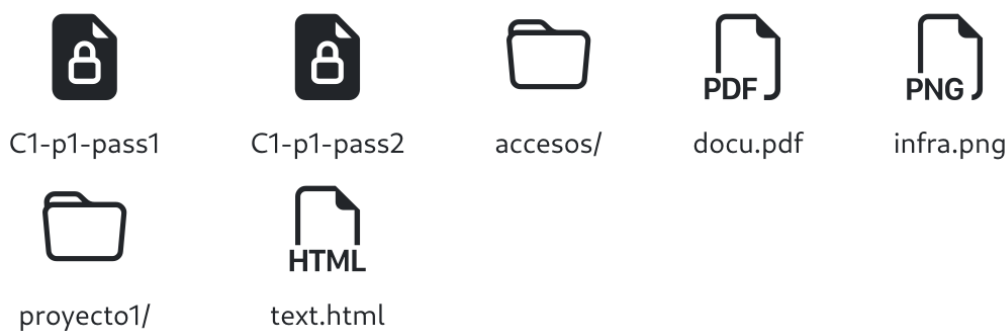


Figura 5.22: Detalle del explorador de ficheros.

Varios de estos componentes están integrados en la vista principal a la hora de iniciar la aplicación, en el fichero `app.component.html`. Desde este fichero se invocan a los componentes junto con parte de código para hacer que la aplicación sea **responsive**.

`</>` Fichero `app.component.html`

```
<app-header></app-header>
<div class="container pt-3 mb-5">
  <div class="row">
    <div id="tree_navbar" class="col-md-3 col-lg-2 d-md-block sidebar
      collapse d-print-none">
      <app-tree></app-tree>
    </div>
    <main class="col-md-9 ms-sm-auto col-lg-10 px-md-4">
      <app-breadcrumb></app-breadcrumb>
      <app-browser></app-browser>
      <router-outlet></router-outlet>
    </main>
  </div>
</div>
<app-footer></app-footer>
```

Dado que los componentes idealmente deben centrarse en la experiencia de usuario, se detallará cómo se ha realizado el paso de información entre ellos.

5.4.2. Services e inyección de dependencias

Para la obtención de datos de los secretos a través del *backend* Vault utilizando su API, y para la sincronización y envío de información entre los distintos componentes de la vista, se ha creado un servicio en Angular cuya dependencia se ha inyectado al inicio de la aplicación.

El servicio ha facilitado que distintos componentes que van a visualizar parte de la misma información (como la vista de árbol y el explorador de ficheros), o que dependen unos de otros, puedan estar en sincronía a través del servicio.

5.4.3. Programación asíncrona con *promises*

Dado que gran parte de la aplicación se basa en comunicarse con la API de Vault, ha sido necesario hacer uso en todo momento de la programación asíncrona al realizar las peticiones.

El ejemplo más claro es la generación del árbol de secretos. Para ello se ha implementado un algoritmo recursivo de peticiones a la API de los secretos. Al realizar una petición y comprobar que existía una jerarquía de directorios, debe entrar al directorio y recorrerlo recursivamente.

Al finalizar, las *promises*, que son la representación de que una petición asíncrona se ha completado o ha fallado, hay que resolverlas para generar la estructura jerárquica que luego se convierte en la vista de árbol.

Resolviendo *promises* de la función recursiva

```
// Cogemos las promesas y las resolvemos
Promise.all(promises).then(data => {
  data.forEach((value:any) => {
    if (value != null){
      var x = nodes.map(function(e:any){
        return e.href
      }).indexOf(value[0].path)
      // cada nodo pertenece a un padre
      nodes[x].nodes = value
    }
  });
  resolve(nodes)
});
```

Angular, y ciertas peticiones que podemos realizar a través de la red, hace uso de *observables*, que funcionan de manera similar a las *promises*.

5.4.4. Envío de mensajes con *observables*

Para la comunicación entre componentes, y también para consolidar la información a través de la API de Vault con la vista, se ha hecho uso de distintas variables de tipo **Observable**.

Este tipo de variables hacen uso del patrón conocido como “**publish-subscribe**”, en el que desde un *subject* se mantiene una lista de *observers* que dependen de él.

De esta manera, cuando se envía un mensaje, los observadores reciben esa información de manera asíncrona para posteriormente utilizarla en la vista en la que está escuchando.

En el siguiente ejemplo se va a detallar parte de la información que se puede encontrar en el *Service* creado llamado **VaultServices**, en el que se inicializa una variable “secret_history” que contendrá información de las distintas versiones de un secreto.

A través de la función “get_secret_history()” se realizarán peticiones a la API de Vault (usando otra función propia), para posteriormente generar un *array* de la información que posteriormente se enviará a los subscriptores que están a la espera.

🔗 Creamos *Observable* y mandamos información

```
export class VaultService {
  //...
  public secret_history = new Subject<object>()
  public secret_history$ = this.secret_history.asObservable()
  //...
  public get_secret_history(){
    this.get_path(this.actual_url,"metadata").subscribe((resp:any) => {
      let versions:any = []
      let versions_keys = Object.keys(resp.data.versions)
      for (let index = 0; index <= versions_keys.length-1; index++) {
        versions.push([versions_keys[index],
          resp.data.versions[versions_keys[index]]])
      }
      //enviamos información
      this.secret_history.next(versions.reverse())
    })
  }
}
```


Continuando con el ejemplo, desde la vista en la que se visualiza el secreto, se puede ver cómo se ha creado una suscripción a la variable detallada previamente, que al recibir información actualizará la variable local “versions”.

✎ Ejemplo de suscripción a un *Observable*

```
export class SecretComponent implements OnInit {
  versions: any = null
  constructor(
    protected vault: VaultService,
  ) {
    //...
    this.vault.secret_history.subscribe(secret_history => {
      this.versions = secret_history
    })
  }
}
```

Esta variable, para terminar, es utilizada en la vista para generar una lista con las distintas versiones con las que cuenta el secreto:

✎ Uso de la variable en la vista

```
<ul *ngFor="let version of versions; index as i">
  <li><a routerLink="/v/{{version[0]}}{{this.vault.actual_url}}"
    (click)="modal.dismiss('Cross click')">
    <strong>v{{version[0]}}</strong>:
    {{version[1].created_time|date: "yyyy-MM-dd, HH:mm:ss z"}}
  </a>
</li>
</ul>
```

Este tipo de secuencia “llamada a la API de Vault → envío de información al suscriptor/suscriptores → recibir información → actualizar vista” se repite de manera continuada a lo largo de toda la aplicación.

6. Conclusiones

La seguridad a la hora de almacenar datos importantes, y salvaguardar las contraseñas de accesos a servicios debe ser una prioridad a nivel personal, pero mucho más cuando hablamos a nivel empresarial. En este último caso puede llegar a haber consecuencias legales en caso de perder esa información (ya sea por borrado o por filtraciones por ataques de agentes externos).

A pesar de que ya existen herramientas dedicadas a guardar contraseñas, se ha demostrado que este tipo de aplicaciones comerciales no están exentas de fallos de seguridad, y que debido a la importancia de las mismas, suelen ser objetivos de ataque para la obtención de información sensible.

Aún existiendo aplicaciones de Software Libre para el mismo fin, en las que la comunidad se vuelca para que tales fallos de seguridad no existan, normalmente sólo están dedicadas a guardar contraseñas, sin posibilidad de crear documentación en la propia aplicación. Existe la posibilidad de modificar estas aplicaciones, gracias a las licencias libres que las componen, pero puede suponer un esfuerzo que puede no estar al alcance de muchos.

A lo largo de este Trabajo Fin de Máster se ha demostrado cómo crear una herramienta *ad hoc* para guardar contraseñas, subir ficheros y donde se puede crear y editar documentación es un proyecto asequible que puede ser llevado por cualquier empresa tecnológica que cuente con un desarrollador que tenga conocimientos de programación.

El hecho de crear una aplicación propia, y securizarla para limitar el acceso a la misma, trae consigo una serie de ventajas que un producto comercial es difícil que consiga. Podremos adaptar la aplicación a medida que la empresa vaya necesitando nuevas características, y por tanto no nos limitaremos a lo que un producto de terceros nos permita hacer.

Si para ello, al igual que se ha expuesto en este documento, se hace uso de un *framework* consolidado como es Angular, y se le acompaña de un sistema dedicado al almacenamiento seguro de información como es Vault, la combinación resultante es muy potente para una empresa.

Por último, y dado que para la creación de la aplicación nos hemos basado en herramientas con licencia de Software Libre, al dotar a nuestra aplicación del mismo tipo de licencia, estaremos permitiendo a la comunidad que incluyan mejoras de cualquier tipo. Esto no sólo es beneficioso para la propia comunidad, sino también para las empresas, que podrán añadir esas nuevas características a su propio desarrollo, mejorando así su propia aplicación interna.

7. Trabajo futuro

Dada la naturaleza de la aplicación realizada, y las distintas posibilidades que puede aportar dentro de una empresa, se podría decir que el trabajo futuro a desarrollar es tan diverso como las necesidades de las empresas que lo puedan utilizar.

Actualmente la aplicación se considera que es funcional dentro de las tareas que puede realizar, y que por tanto se podría hacer uso de ella para un entorno en producción dentro de cualquier tipo de empresa.


Es cierto que las funcionalidades que tiene actualmente se pueden considerar básicas, pero eso facilita que la aplicación ya pueda ser utilizada. Teniendo en cuenta las metodologías ágiles de desarrollo utilizadas, es cuestión de escuchar la opinión de los usuarios para ir aportando valor y de esta manera ir ampliando las funcionalidades.

Con todo ello, dado que existen muchas funcionalidades que se han quedado sin realizar, y que podrían interesar a nivel general, a continuación se detallan un número de funcionalidades que podrían ir añadiéndose a la aplicación en futuros *sprints*:

- **Panel de control para administradores:** Como muchas aplicaciones, va a existir el rol de administrador, y que por tanto será el encargado de administrar no sólo la gestión del servicio, si no también la creación de usuarios y asignar permisos.

Es por eso que sería interesante crear una ruta dentro de la aplicación a la que sólo los administradores puedan acceder y donde se podrían efectuar las siguientes tareas:

- **Gestión de usuarios:** Si no se hace uso de un sistema centralizado de usuarios (como *Active Directory* o *LDAP*), es conveniente tener un lugar donde crear usuarios, borrarlos, cambiarles la contraseña...
- **Gestión de permisos:** Para crear las políticas de permisos de los usuarios.
- **Gestionar secretos bloqueados:** Tener un lugar donde desbloquear posibles secretos que se hayan quedado bloqueados.
- **Auditoría completa:** Vault cuenta con un sistema completo de auditoría, que es capaz de guardar logs por cada acción de usuario. Dada la importancia de este tipo de información, podría ser recomendable guardarlo en un sistema de base de datos no-relacional y poder acceder a dicha información en cualquier momento.
- **Crear configuración para el usuario:** Crear un área de configuración para el usuario. En este apartado, algunas opciones que se podrían configurar son:
 - Cambiar el aspecto visual de la aplicación. Ya sea a través de distintos temas o simplemente entre un tema claro y otro oscuro.

- Permitir forzar la actualización de la vista de árbol.
- Decidir si se quiere tener la vista de árbol siempre visible.
- **Auditoría propia para los secretos:** Crear un sistema propio más sencillo que el que tiene de serie Vault para conocer las modificaciones históricas de quién ha modificado el secreto.
- **Ver diferencias entre versiones:** Actualmente se pueden ver las distintas versiones de un secreto, pero puede ser interesante seleccionar dos versiones y ver cuáles son las diferencias entre las dos versiones. Sería tener una herramienta similar a .
- **Creación de “timers”:** La idea es que si un usuario no utiliza la aplicación en un tiempo configurable (por ejemplo, 5 minutos), se realice un deslogueo de la aplicación. De esta manera, se salvaguardan los datos ante posibles accesos a un equipo que no está siendo utilizado en un momento dado.
- **Mejoras en la visualización de secretos:** Mejorar y ampliar la visualización de los distintos tipos de ficheros que se pueden subir a la aplicación.
Puede ser interesante permitir la visualización de documentos tipo Word, LibreOffice, Excel,... haciendo uso de librerías externas.
- **Crear un índice en los secretos creados:** Dado que la aplicación permite crear documentos gracias al sistema Markdown, se entiende que estos documentos pueden contar con una jerarquía interna basada en encabezados.
Sería interesante que antes de visualizar el secreto, *parsearlo* para generar una “tabla de contenidos” (TOC o *table of contents* en inglés), y de esta manera generar índices para poder llegar a la parte que interese del documento.
- **Generar e-mail para enviar un secreto:** Junto con la opción de imprimir el secreto, puede ser interesante la opción de crear un botón para generar automáticamente un e-mail para enviar el secreto.
Como alternativa, y para no tener que enviar la información, se podría generar un token que dure un periodo de tiempo determinado, y que al generar el e-mail se mande dicho token. El usuario receptor deberá loguearse con dicho token para poder visualizar la información.

Tal como se puede observar, el número de nuevas funcionalidades con las que mejorar la aplicación puede resultar muy amplia, y tal como se ha dicho previamente, todo ello sin tener en cuenta la opinión de los usuarios finales.

Con todo ello, y teniendo en cuenta las posibilidades que este tipo de aplicación puede llevar a una empresa, resulta interesante pensar las posibles funcionalidades que podría llegar a tener en un futuro.

Referencias bibliográficas

- Angular. (2016). The web development framework for building the future. Consultado el 18 de noviembre de 2022, desde <https://angular.io/>
- Atlassian. (2011, 13 de septiembre). Trello brings all your tasks, teammates, and tools together. Consultado el 18 de enero de 2023, desde <https://trello.com/>
- Belenko, A. & Sklyarov, D. (2012). "Secure Password Managers" and "Military-Grade Encryption" on Smartphones: Oh, Really? *Blackhat Europe*, 56. Consultado el 4 de enero de 2023, desde <https://us.elcomsoft.com/WP/BH-EU-2012-WP.pdf>
- Gruber, J. & Swartz, A. (2004). Lenguaje de marcado ligero. Consultado el 18 de enero de 2023, desde <https://es.wikipedia.org/wiki/Markdown>
- Hashicorp Vault. (2015). Manage Secrets & Protect Sensitive Data. Consultado el 18 de noviembre de 2022, desde <https://www.vaultproject.io/>
- McCarney, D., Barrera, D., Clark, J., Chiasson, S. & van Oorschot, P. C. (2012). Tapas: Design, Implementation, and Usability Evaluation of a Password Manager. En *Proceedings of the 28th Annual Computer Security Applications Conference* (pp. 89-98). doi:[10.1145/2420950.2420964](https://doi.org/10.1145/2420950.2420964)
- Microsoft. (2012, 1 de octubre). TypeScript is JavaScript with syntax for types. Consultado el 18 de enero de 2023, desde <https://www.typescriptlang.org/>
- Navarro, F. (2019). Hackers find security flaws in 5 popular password managers. Are you safe? *Komando*. Consultado el 29 de diciembre de 2022, desde <https://www.komando.com/security-privacy/hackers-find-security-flaws-in-5-popular-password-managers-are-you-safe/547660/>
- NHNCloud. (2015, 28 de agosto). Markdown WYSIWYG Editor. GFM Standard, Chart and UML Extensible. Consultado el 20 de noviembre de 2022, desde <https://getbootstrap.com/>
- OpenID Foundation. (2014). The OpenID Foundation (OIDF) promotes, protects and nurtures the OpenID community and technologies. Consultado el 18 de noviembre de 2022, desde <https://openid.net/foundation/>
- Owen, M. (2023). Norton Password Manager hacked, warning users about breaches. *appleinsider*. Consultado el 15 de enero de 2023, desde <https://appleinsider.com/articles/23/01/14/nortonlifelock-warns-of-password-manager-account-breaches>
- Scarfone, K. & Souppaya, M. (2009). Guide to enterprise password management (draft). *NIST special publication, 800(118)*, 3-2. Consultado desde <https://www.tier3md.com/media/800-118.pdf>

- Schappert, S. (2022). LastPass confirms another breach. *Cybernews*. Consultado el 27 de diciembre de 2022, desde <https://cybernews.com/news/lastpass-second-breach/>
- Schappert, S. (2023). Norton Password Manager breach: nearly one million users targeted. *Cybernews*. Consultado el 18 de enero de 2023, desde <https://cybernews.com/security/hackers-compromise-norton-password-manager/>
- Tarantola, A. (2022). The Lastpass hack was worse than the company first reported. *Engadget*. Consultado el 27 de diciembre de 2022, desde <https://www.engadget.com/the-lastpass-hack-was-worse-than-the-company-first-reported-000501559.html>
- Torvalds, L. (2007, 19 de octubre). Git is a free and open source distributed version control system. Consultado el 18 de noviembre de 2022, desde <https://git-scm.com/>
- Twitter. (2011, 19 de agosto). Build fast, responsive sites with Bootstrap. Consultado el 18 de noviembre de 2022, desde <https://getbootstrap.com/>
- Vaughan-Nichols, S. (2022). LastPass was hacked – again. *ZDnet*. Consultado el 29 de diciembre de 2022, desde <https://www.zdnet.com/article/lastpass-hacked/>
- Wagenseil, P. (2020). LastPass, 1Password and other password managers can be hacked: What to do now. *Tom's guide*. Consultado el 10 de enero de 2023, desde <https://www.tomsguide.com/news/password-manager-hacks>
- Which Password Managers Have Been Hacked? (s.f.). *Best Reviews*. Consultado el 4 de enero de 2023, desde <https://password-managers.bestreviews.net/faq/which-password-managers-have-been-hacked/>