

PONTIFICIA UNIVERSIDAD CATÓLICA MADRE Y MAESTRA
Facultad De Ciencias e Ingeniería
Escuela de Ingeniería en Computación y Telecomunicaciones



Doble autenticador y gestor de secretos

**Un proyecto presentado como requisito parcial para optar por el título de
Ingeniero en Telemático/Sistemas y Computación en la
Pontificia Universidad Católica Madre y Maestra.**

por

Eliam Pimentel Suárez

Asesor: Lissibonny Eustina Beato

Título, Departamento

Santiago de los Caballeros, República Dominicana

Noviembre del 2023

Tabla de contenido

Resumen Ejecutivo	3
Introducción	4
Antecedentes del Problema	4
Antecedentes del Proyecto	5
Descripción del Problema	7
Planteamiento Inicial de la Solución	8
Objetivo General	9
Objetivos Específicos	9
Justificación del Proyecto	10
Limitaciones del Proyecto	10
Capítulo I - Marco Teórico	11
1.2. Definición de Términos y Glosario:	15
Capítulo II - Solución Propuesta:	17
2.1. Definición del Proyecto:	17
2.2 Productos del proyecto	24
2.4 Plan de Administración de Riesgos	26
2.5 Presupuesto	27
2.6 Definición de la demostración	28
2.7 Análisis y Diseño	29
Desarrollo e Implementación	38
Conclusiones y recomendaciones	46
Referencias	47
Anexos	49
Diseño	49
Mockups	49
Desarrollo e implementación	53

Resumen Ejecutivo

El proyecto de autenticación en dos pasos y manejo de secretos es una iniciativa que maneja información altamente sensible de los usuarios. Durante el desarrollo de la API, se han implementado rigurosas medidas de seguridad para salvaguardar la integridad y confidencialidad de los datos almacenados. Se han considerado diversas bibliotecas para cifrar la información, y se ha integrado el sólido algoritmo AES con este propósito.

La API del proyecto ya ha sido completamente desarrollada, incluyendo la generación de modelos para obtener códigos TOTP y su posterior generación en base a dicho modelo. Asimismo, se ha implementado la capacidad de almacenar secretos, aunque, hasta el momento, esta funcionalidad se encuentra adaptada exclusivamente para el manejo de contraseñas. Es importante destacar que esta funcionalidad es altamente escalable, permitiendo la incorporación de diversos tipos de secretos en el futuro.

En adición al desarrollo de la API, se han creado algunos mockups para la futura aplicación móvil. El siguiente paso es avanzar en el desarrollo de las aplicaciones, y en caso de disponer de tiempo adicional, se contempla la incorporación de características adicionales en la sección de secretos, como el almacenamiento de claves SSH, conexiones a bases de datos, tarjetas de crédito/débito, direcciones, entre otros.

Introducción

Antecedentes del Problema

En la actualidad, la seguridad en línea es más importante que nunca debido al aumento en las violaciones de datos y los ataques cibernéticos. Las personas deben preocuparse por este tipo de seguridad porque ya hemos sido víctimas de hackeos en numerosas ocasiones, lo que ha puesto en riesgo nuestra privacidad y nuestros datos personales.

Por ejemplo, en 2012 se produjo una brecha de datos en LinkedIn, donde se robaron y publicaron en línea aproximadamente 6.5 millones de contraseñas de usuarios. Este incidente pone de manifiesto la importancia de proteger nuestras credenciales de usuario y fue un punto de inflexión en la adopción de medidas de autenticación de dos factores (2FA). [1]

Otro caso notable es el ataque a iCloud en 2014, donde se filtraron en línea fotos personales de celebridades almacenadas en sus cuentas. Aunque Apple negó que hubiera una vulnerabilidad en su sistema de autenticación, este incidente generó conciencia sobre la necesidad de proteger nuestras cuentas en línea con medidas adicionales como el 2FA. [1]

Además, Yahoo sufrió brechas de datos importantes en 2013 y 2014, que afectaron a casi 1.5 mil millones de cuentas de usuario. Estos incidentes masivos resaltaron la importancia de mejorar la seguridad de nuestras cuentas en línea y llevaron a una mayor adopción de soluciones de 2FA en toda la industria. [1]

Antecedentes del Proyecto

En respuesta a la creciente importancia de fortalecer la seguridad en línea, diversas soluciones han surgido para abordar la problemática de proteger las cuentas contra accesos no autorizados. Este desafío ha llevado al desarrollo de aplicaciones especializadas, entre las cuales destacan 2FAS, Authy y Keeper. Cada una de estas herramientas aborda la autenticación y seguridad de manera individual, ofreciendo funciones específicas para mejorar la protección de las cuentas en entornos digitales. La evolución de estas aplicaciones ha demostrado la necesidad de consolidar estas funcionalidades, proporcionando a los usuarios una solución integral y centralizada.

A continuación, exploraremos en detalle cada una de estas herramientas: 2FAS, Authy y Keeper, destacando sus características distintivas y cómo contribuyen a fortalecer la seguridad en línea.

2FAS

2FAS es una aplicación de autenticación de dos factores que ayudará a fortalecer la seguridad de tus cuentas en línea. Funciona generando tokens temporales o de un solo uso que tú deberás ingresar, además de tu contraseña regular, para acceder a tus cuentas. Esto hace que sea mucho más difícil para alguien obtener acceso no autorizado a tus cuentas, incluso si logran robar o adivinar tu contraseña original. 2FAS admite dos tipos de códigos de acceso: basados en tiempo (TOTP) y basados en eventos (HOTP). Esto significa que los códigos son válidos ya sea por un cierto período de tiempo o hasta que se realice un cierto evento, como un inicio de sesión exitoso. También cuenta con la capacidad de funcionar en modo sin conexión, lo que puede ser muy útil si estás en un lugar sin cobertura de celular o wi-fi. [2]

Authy

Authy cumple la misma función básica que 2FAS, proporcionando autenticación de dos factores para aumentar la seguridad de tus cuentas en línea. Lo que distingue a Authy es su función de copia de seguridad. Esto significa que si pierdes tu teléfono o compras uno nuevo, puedes restaurar fácilmente todos tus códigos 2FA a través de la copia de seguridad de Authy, en lugar de tener que configurar la autenticación de dos factores nuevamente para cada cuenta. Esto puede ahorrar bastante tiempo y molestias. Authy también permite la

sincronización de códigos en múltiples dispositivos y tiene una interfaz de usuario intuitiva y fácil de usar. [3]

Keeper

Keeper es una aplicación de gestión de contraseñas que te permite almacenar todas tus contraseñas en un lugar seguro, de manera encriptada. Sólo necesitas recordar una contraseña maestra para acceder a todas las demás. Además, Keeper puede generar contraseñas seguras y únicas para cada cuenta y autofocus (autocompletar) estas contraseñas cuando intentes iniciar sesión en un sitio web o una aplicación. Keeper también ofrece autenticación de dos factores, proporcionando una capa adicional de seguridad cuando accedes a tu bóveda de contraseñas. Keeper incluso te permite almacenar archivos, fotos y documentos confidenciales en su bóveda segura. [4]

Descripción del Problema

La falta de una solución integral que combine la autenticación de dos factores (2FA) y la gestión de contraseñas en un solo sistema o aplicación dificulta la utilización eficiente y segura de estas herramientas por parte de los usuarios. Si bien existen aplicaciones como 2FAS, Authy y Keeper que brindan cada una de estas funcionalidades de manera individual, no hay una solución que integre ambas en un único ecosistema. Esta carencia conlleva varios inconvenientes. Los usuarios deben emplear diferentes aplicaciones y recordar distintas contraseñas para acceder a las funcionalidades de autenticación de dos factores y gestión de contraseñas, generando una mayor complejidad y dificultad para administrar y utilizar estas herramientas de manera efectiva.

Además, al no tener una solución integral, puede existir una falta de interoperabilidad entre estas aplicaciones. Por ejemplo, si un usuario necesita iniciar sesión en una cuenta con autenticación de dos factores y utilizar una contraseña generada y almacenada de manera segura, es posible que deba alternar entre diferentes aplicaciones, lo cual puede resultar inconveniente y propenso a errores. Asimismo, la falta de una solución integral puede implicar la necesidad de configurar y mantener múltiples aplicaciones, lo que puede resultar costoso y consumir más recursos tanto en términos de tiempo como económicos. En este contexto, una solución que integre ambas funcionalidades sería altamente beneficiosa, proporcionando una experiencia más conveniente, reduciendo la complejidad y mejorando la seguridad en la gestión de cuentas y contraseñas en línea.

Planteamiento Inicial de la Solución

La solución al problema planteado implica el desarrollo de una aplicación de software que resolverá dos desafíos principales para los usuarios. En primer lugar, proporcionará una interfaz intuitiva y fácil de usar que permitirá a las personas gestionar sus códigos TOTP y contraseñas de manera eficiente. Los usuarios podrán ingresar, almacenar y recuperar sus contraseñas y códigos TOTP de forma sencilla, eliminando la necesidad de recordar contraseñas complicadas.

Para garantizar la seguridad de las contraseñas, la aplicación empleará algoritmos criptográficos sólidos. Esto significa que las contraseñas se almacenarán de manera segura y estarán protegidas contra posibles amenazas de seguridad. Además, se incorporará la autenticación en dos pasos utilizando códigos TOTP. Esto añade una capa adicional de protección a las cuentas en línea, ya que los usuarios deberán proporcionar un código generado por la aplicación en tiempo real para acceder a sus cuentas, incluso si alguien más tiene su contraseña.

La aplicación se diseñará teniendo en cuenta la facilidad de uso, de modo que cualquier persona, incluso sin conocimientos técnicos avanzados, pueda utilizarla de manera efectiva. Este proyecto busca brindar a los usuarios una solución integral para la gestión de contraseñas y la autenticación de dos pasos, mejorando así su seguridad en línea y simplificando el proceso de acceso a cuentas en internet.

Objetivo General

Desarrollar una aplicación que integre la autenticación de dos pasos (2FA) y la gestión de contraseñas en una única solución, contribuyendo a mejorar la seguridad en línea de los usuarios.

Objetivos Específicos

1. Crear un algoritmo criptográfico personalizado basado en el algoritmo de cifrado AES (Advanced Encryption Standard) para la generación de contraseñas seguras, aplicando los principios y técnicas de la criptografía. El AES es un algoritmo ampliamente reconocido y utilizado para el cifrado de datos, conocido por su seguridad y eficiencia.
2. Estructurar el gestor de tokens y desarrollar un sistema seguro para la generación de códigos TOTP (Time-Based One-Time Password) con el fin de agregar una capa adicional de protección a las cuentas en línea. Este objetivo implica comprender y aplicar los fundamentos de TOTP para generar códigos temporales únicos y asegurar la autenticación en dos pasos.
3. Diseñar interfaces de usuario intuitivas y de fácil manejo para la administración de códigos TOTP, contraseñas y datos sensibles, considerando las características específicas de las aplicaciones móviles y las extensiones de navegadores.

Justificación del Proyecto

La seguridad en línea es una preocupación creciente en la sociedad moderna. La creciente dependencia de las tecnologías digitales ha dado lugar a un aumento significativo en los riesgos de seguridad en línea, incluyendo el robo de contraseñas, el acceso no autorizado a cuentas y las violaciones de datos. Estos incidentes pueden tener graves consecuencias, como la pérdida de información personal, la exposición de datos confidenciales y el robo de identidad. Además, la falta de seguridad en línea puede afectar negativamente la confianza de los usuarios en los servicios en línea y socavar la privacidad digital.

La implementación de la autenticación en dos pasos y el manejo seguro de contraseñas son medidas fundamentales para abordar estos riesgos y fortalecer la seguridad en línea. Este proyecto busca proporcionar una solución integral que beneficie a los usuarios, permitiéndoles proteger sus cuentas y datos de manera efectiva y sencilla. Al mejorar la seguridad en línea, no solo se brinda una mayor tranquilidad a los usuarios, sino que también se contribuye a mitigar las amenazas ciberneticas y sus efectos perjudiciales en la sociedad, asegurando un entorno digital más seguro y confiable para todos.

Limitaciones del Proyecto

- 1. Limitación de Recursos Técnicos:** La falta de acceso a una computadora Mac con XCode puede limitar el desarrollo de la extensión de Safari, ya que XCode es necesario para la creación de aplicaciones para el ecosistema de Apple. Esto podría requerir la adquisición de hardware adicional o acceso a una plataforma de desarrollo alternativa.
- 2. Costo de Publicación:** La publicación de la aplicación en la Play Store de Google y la App Store de Apple conlleva costos adicionales. Estos costos pueden incluir tarifas de registro de desarrollador y otros gastos asociados con la publicación y mantenimiento de la aplicación en estas plataformas. Esta limitación podría requerir la asignación de recursos financieros adicionales para la distribución de la aplicación.

Capítulo I - Marco Teórico

1.1. Autenticación en Dos Pasos (2FA):

El doble autenticador, también conocido como autenticación de dos factores (2FA, por sus siglas en inglés), es un método de seguridad utilizado para proteger las cuentas en línea. Consiste en agregar una capa adicional de autenticación a la tradicional contraseña, requiriendo que los usuarios proporcionen una segunda forma de verificación antes de acceder a sus cuentas. [5] Esta segunda forma de verificación puede ser algo que el usuario posee, como un código generado por una aplicación en su teléfono móvil, o algo que el usuario es, como su huella dactilar o reconocimiento facial.

Antes de la adopción generalizada de 2FA, las contraseñas solían ser el único factor de autenticación utilizado para acceder a cuentas en línea. Sin embargo, eventos como brechas de datos masivas y ataques cibernéticos han demostrado que las contraseñas pueden ser vulnerables y susceptibles de ser comprometidas.

La implementación de 2FA añade una capa adicional de seguridad al requerir que los usuarios proporcionen un segundo factor de autenticación, además de su contraseña. Esto dificulta enormemente la tarea de los hackers, ya que no solo necesitarían conocer la contraseña, sino también tener acceso físico o información personal adicional del usuario.

El uso creciente de 2FA se debe a la importancia de proteger nuestras cuentas en línea de posibles amenazas. Eventos destacados, como la filtración de información personal de usuarios conocidos a través de brechas de datos o ataques, han resaltado la necesidad de adoptar medidas más seguras para proteger nuestras cuentas en línea.

A medida que la conciencia sobre la seguridad en línea aumenta y la industria enfrenta continuamente desafíos de seguridad, el uso de métodos de autenticación más robustos, como la autenticación de dos factores, se ha convertido en una práctica cada vez más común y recomendada.

Las aplicaciones de autenticación de doble factor (2FA) funcionan de la siguiente manera:

1. Se genera una clave secreta única al configurar la autenticación de dos factores en una cuenta. Esta clave secreta se comparte entre el servidor y la aplicación de autenticación del usuario, generalmente mediante un código QR escaneado con la app. La clave se almacena tanto en el servidor como en la aplicación.
2. Es esencial que el servidor y la aplicación estén sincronizados en cuanto al tiempo. El algoritmo TOTP se basa en el tiempo actual, utilizado en intervalos de 30 segundos, para generar tokens de autenticación.
3. Cuando el usuario intenta iniciar sesión, la aplicación de autenticación genera un token de autenticación utilizando la clave secreta y el tiempo actual. Este token se crea aplicando un algoritmo TOTP, que es una función hash (como HMAC-SHA1, HMAC-SHA256 o HMAC-SHA512) a la combinación de la clave secreta y el contador de tiempo.
4. El usuario ingresa su contraseña y el token de autenticación generado por la aplicación. El servidor, que también tiene la clave secreta y el tiempo sincronizado, genera su propia versión del token de autenticación utilizando el mismo algoritmo TOTP. Luego, el servidor compara el token ingresado por el usuario con el token generado internamente. Si coinciden, se considera una autenticación exitosa y se concede el acceso.
5. Debido a que la sincronización perfecta del tiempo entre el servidor y la aplicación puede ser complicada, se permite una cierta tolerancia en la comparación de los tokens. Esto significa que el servidor puede aceptar tokens generados en un intervalo de tiempo anterior o posterior al actual, evitando así problemas de autenticación innecesarios.

1.2. Códigos TOTP (Time-Based One-Time Password):

El algoritmo Time-based One-time Password (TOTP) se utiliza para proporcionar una mayor seguridad en Internet al generar contraseñas únicas válidas solo durante un breve período de tiempo. Estas contraseñas se utilizan como parte de una autenticación multifactor, donde los usuarios ingresan su contraseña personal junto con una contraseña especial generada durante el proceso de inicio de sesión. [6]

El funcionamiento del algoritmo TOTP se basa en una función hash criptográfica. Se forma una secuencia de caracteres codificada a partir de una contraseña secreta y una marca de tiempo, utilizando el tiempo Unix como base para la indicación temporal. El tiempo Unix es un valor que representa los segundos transcurridos desde el 1 de enero de 1970.

El TOTP es una evolución del algoritmo HOTP (HMAC-based One-time Password), también basado en el procedimiento HMAC y la función hash. Tanto el dispositivo del usuario como el servidor generan un valor hash utilizando la contraseña secreta y un contador, asegurando que los valores sean idénticos para la autenticación.

La función hash utilizada en el algoritmo no está específicamente definida, pero en práctica se suele emplear SHA-1, generando un valor hash de 160 bits. Este valor se puede acortar mediante una función de compresión, resultando en un número de seis cifras que el usuario puede ingresar fácilmente durante el inicio de sesión en el servicio web. Esta contraseña temporal caduca si no se utiliza en un período de tiempo determinado, lo que dificulta a los criminales acceder al segundo factor incluso si conocen la contraseña personal fija.

1.3. Gestión de Contraseñas:

Millones de personas crean cuentas en distintos portales y sitios web para realizar una variedad de tareas, tanto a nivel personal como profesional. En estos sitios, como aplicaciones bancarias, redes sociales o plataformas de compra y venta, es necesario utilizar contraseñas de acceso para proteger la información personal que está en juego. Es fundamental prestar atención a la seguridad de las contraseñas y otros aspectos de seguridad en la web. Cada vez más personas utilizan contraseñas más largas y complejas, que contienen mayúsculas, minúsculas, números y símbolos, con el

objetivo de hacer que sea más difícil para los hackers descifrarlas. También es importante no utilizar la misma contraseña para todas las cuentas. Sin embargo, recordar diferentes contraseñas complejas puede resultar tedioso o casi imposible. Esto lleva a muchas personas a terminar utilizando la misma clave para acceder a todas las plataformas. Para solucionar este problema, es necesario contar con un gestor de contraseñas, que permite a los usuarios gestionar todas sus claves sin tener que memorizarlas o buscarlas en otras fuentes. El uso de gestores de contraseñas ofrece varias ventajas. Por un lado, pueden generar contraseñas aleatorias y seguras, protegiendo a los usuarios de los hackers. Además, los gestores de contraseñas permiten a los usuarios recordar solo una contraseña maestra, que da acceso a todas las demás contraseñas almacenadas en el gestor. Estos gestores de contraseñas tienen características específicas, como la generación de claves de más de 8 caracteres, el uso de contraseñas no personalizadas, la combinación de diferentes tipos de caracteres y la capacidad de llenar automáticamente los formularios de inicio de sesión en los sitios web. Las contraseñas se almacenan de manera cifrada para garantizar su seguridad. [7]

1.4. Criptografía:

La encriptación es una herramienta esencial en la protección de la información y la privacidad. Se basa en la criptografía, que es la ciencia que utiliza métodos y herramientas matemáticas para cifrar mensajes o archivos, con el fin de protegerlos mediante un algoritmo que requiere dos o más claves. Esto permite lograr la confidencialidad y, en algunos casos, la autenticidad de la información. [8]

Un ejemplo común de encriptación es el envío de correos electrónicos. Cuando necesitamos enviar información confidencial, encriptamos el contenido utilizando un algoritmo y lo enviamos al destinatario. Para que este pueda leer el contenido, necesita contar con la clave de descifrado correspondiente. De lo contrario, solo verá un conjunto de caracteres ilegibles. Esta medida asegura que solo las personas autorizadas puedan acceder a la información y garantiza su seguridad incluso si se intercepta el mensaje.

La encriptación no solo se aplica al cifrado de mensajes, sino también a canales de comunicación, archivos, imágenes, documentos y dispositivos USB, entre otros. Protege la información de la mirada de terceros no autorizados y salvaguarda la privacidad.

Es importante tener claras las diferencias entre la criptografía, que es la ciencia que utiliza métodos y herramientas para cifrar información, y el cifrado, que es la técnica de codificación y decodificación. En el ejemplo anterior, se emplean técnicas de claves públicas y privadas para encriptar el contenido de los correos electrónicos.

La encriptación juega un papel crucial en términos de seguridad, especialmente en el ámbito empresarial. Permite generar confianza con los clientes al ofrecer protección y seguridad de primer nivel. Es fundamental adaptar los desarrollos tecnológicos cumpliendo con los estándares internacionales de seguridad, como el Estándar de Seguridad de Datos para la Industria de Tarjeta de Pago (PCI DSS), tanto en hardware como en software.

1.2. Definición de Términos y Glosario:

- **Autenticación:** Es el proceso mediante el cual se verifica la identidad de un usuario o dispositivo antes de permitir el acceso a una cuenta o sistema. En el contexto de nuestro proyecto, la autenticación se refiere al proceso de verificar la identidad del usuario a través de la combinación de una contraseña tradicional y un código TOTP.
[9]
- **Códigos TOTP (Time-Based One-Time Password):** Son códigos de un solo uso que cambian en intervalos de tiempo regulares, generalmente cada 30 segundos. Estos códigos son generados por una aplicación móvil y se utilizan como una medida adicional de seguridad en la autenticación en dos pasos. [10]
- **Contraseñas Robustas:** Son contraseñas que contienen una combinación de letras mayúsculas, minúsculas, números y caracteres especiales, lo que las hace difíciles de

adivinar para los atacantes. Utilizar contraseñas robustas es fundamental para proteger nuestras cuentas y datos personales.

- **Encriptación:** Es el proceso de transformar información en un formato ilegible utilizando algoritmos criptográficos. Esto se hace para proteger la información confidencial, como las contraseñas, de ser leída por personas no autorizadas. [11]
- **Algoritmo Criptográfico:** Es una serie de pasos matemáticos diseñados para encriptar y desencriptar datos. Los algoritmos criptográficos aseguran que la información se mantenga segura y solo sea accesible para aquellos que posean la clave de desencriptación. [12]
- **Extensión de Navegador:** Es un pequeño programa que se instala en el navegador web y agrega funcionalidades adicionales. En nuestro proyecto, la extensión de navegador permitirá a los usuarios acceder y gestionar sus códigos TOTP y contraseñas directamente desde el navegador. [13]
- **Aplicación Móvil:** Es un software diseñado para ser instalado y utilizado en dispositivos móviles, como teléfonos inteligentes y tabletas. Nuestra aplicación móvil proporcionará una interfaz intuitiva y amigable para que los usuarios puedan acceder y gestionar sus códigos TOTP y contraseñas desde sus dispositivos móviles. [14]
- **Autenticación en Dos Pasos:** Es un proceso de seguridad que requiere dos métodos diferentes para verificar la identidad del usuario al acceder a una cuenta. En nuestro proyecto, la autenticación en dos pasos se realizará mediante la combinación de una contraseña tradicional y un código TOTP. [15]
- **Interfaz Intuitiva:** Es una interfaz de usuario diseñada para ser fácil de entender y utilizar, incluso para usuarios no técnicos. La interfaz intuitiva proporcionará instrucciones claras y una experiencia amigable para facilitar el uso de la herramienta. [16]
- **AES (Advanced Encryption Standard):** El algoritmo de encriptación AES (Advanced Encryption Standard) es un estándar criptográfico simétrico ampliamente

adoptado para garantizar la seguridad de la información. Diseñado para operar en bloques de 128 bits, AES admite claves de 128, 192 y 256 bits, con un número variable de rondas (10, 12 o 14) según el tamaño de la clave. Su proceso de encriptación implica etapas como SubBytes, ShiftRows, MixColumns y AddRoundKey, respaldadas por un mecanismo de expansión de clave. La seguridad de AES se basa en su capacidad para resistir diversos ataques criptoanalíticos, y su aplicabilidad abarca desde comunicaciones seguras hasta almacenamiento de datos confidenciales, consolidándose como un estándar robusto y confiable en el ámbito de la criptografía. [17]

Capítulo II - Solución Propuesta:

2.1. Definición del Proyecto:

1. Manejo de códigos TOTP

Obtención del Secret:

El secreto (o secret key) es crucial para la generación de códigos TOTP. Puede ser proporcionado por la aplicación que admite la autenticación de dos factores (2FA) a través de una URI o ingresado directamente en la página. Este secreto es utilizado como base para la generación de tokens temporales.

La biblioteca `totpauth` de npm simplifica la generación de códigos TOTP. Esta biblioteca sigue el estándar TOTP (Time-based One-Time Password).

Se utiliza el secreto obtenido para generar tokens temporales, que cambian en intervalos de tiempo predefinidos.

Encriptación de Datos:

Se emplea la biblioteca CryptoJS para realizar operaciones criptográficas en el lado del cliente y servidor. Además, el algoritmo criptográfico AES (Advanced Encryption Standard) se utiliza para cifrar la información del usuario.

Proceso de Encriptación:

La información del usuario, como el secreto TOTP y posiblemente otros datos relevantes, se cifra utilizando AES, se utiliza una clave de cifrado.

Desencriptación de Datos:

Para desencriptar los datos almacenados, se utiliza nuevamente la biblioteca CryptoJS. La misma clave de cifrado que se utilizó para encriptar la información se emplea para desencriptarla.

Proceso de Desencriptación:

La información cifrada se somete al proceso de desencriptación utilizando el algoritmo AES y la clave correspondiente.

Una vez desencriptada, la información se presenta al cliente para su uso.

2. Generación segura de contraseñas:

La generación segura de contraseñas es un componente crucial en la seguridad de las cuentas y sistemas informáticos. Este proceso implica crear contraseñas que cumplan con los requisitos de complejidad establecidos por el usuario y luego cifrar y almacenar de manera segura en la base de datos.

Proceso de Generación de Contraseñas:

Parámetros de Configuración:

El usuario define los parámetros para la generación de la contraseña, tales como la longitud deseada, la inclusión de mayúsculas, minúsculas, números y caracteres especiales.

Generación de Contraseña:

Con base en los parámetros proporcionados, se genera una contraseña aleatoria, utilizando la librería nanoid, que cumple con las especificaciones del usuario. Esto garantiza una combinación segura de caracteres para fortalecer la contraseña.

Encriptación de la Contraseña:

Utilizando la biblioteca CryptoJS y el algoritmo de cifrado AES, la contraseña generada se encripta antes de ser almacenada. La encriptación añade una capa adicional de seguridad, asegurando que incluso en caso de acceso no autorizado a la base de datos, las contraseñas permanezcan protegidas.

Almacenamiento en la Base de Datos:

Almacenamiento Seguro:

La contraseña encriptada se almacena de manera segura en la base de datos. La base de datos debe implementar prácticas seguras, como el cifrado de datos, para prevenir posibles filtraciones de información.

3. Autenticación en dos pasos:

Autenticación en Dos Pasos (2FA):

La autenticación en dos pasos (2FA) es una capa adicional de seguridad que requiere que los usuarios proporcionen dos formas distintas de verificación antes de acceder a una cuenta o sistema. En este caso, se emplea la generación de códigos TOTP como uno de los factores para la autenticación.

Proceso de Autenticación en Dos Pasos:

Inicio de Sesión Tradicional:

El usuario inicia sesión proporcionando su nombre de usuario y contraseña como parte del proceso de autenticación tradicional.

Solicitud del Código TOTP:

Después de ingresar las credenciales, el sistema solicita al usuario un código TOTP adicional como segundo factor de autenticación.

Generación del Código TOTP:

El usuario utiliza la aplicación de autenticación (que generó el secreto TOTP) para obtener el código TOTP actual. Este código cambia cada 30 segundos.

Ingreso del Código TOTP:

El usuario ingresa el código TOTP generado en el campo correspondiente en la interfaz de inicio de sesión.

Verificación del Código TOTP:

El sistema verifica la validez del código TOTP ingresado. Si es correcto y coincide con el esperado para ese momento específico, se autentica al usuario.

4. Interfaz intuitiva y fácil de usar

El diseño de una interfaz intuitiva y fácil de usar es esencial para cualquier proyecto, especialmente cuando se trata de aplicaciones móviles, extensiones de navegador y páginas web. En este caso, el proyecto se desarrollará con una aplicación móvil en Flutter, una extensión de navegador, y una página web con Next.js.

Razones para una Interfaz Intuitiva:

Experiencia del Usuario (UX):

- Una interfaz intuitiva mejora la experiencia del usuario. Los usuarios deben poder navegar y utilizar las funciones de la aplicación de manera natural, sin sentirse abrumados por opciones complejas o funciones confusas.

Accesibilidad:

- Una interfaz intuitiva hace que la aplicación sea accesible para una amplia gama de usuarios, independientemente de su nivel de experiencia tecnológica. Esto es especialmente importante para garantizar que la seguridad del proyecto sea efectiva y utilizada por un público diverso.

Aprendizaje Rápido:

- Una interfaz fácil de entender acelera el proceso de aprendizaje para los nuevos usuarios. Esto es crucial para un sistema de autenticación, donde los usuarios deben comprender rápidamente cómo utilizar las funciones de seguridad.

Reducción de Errores:

- Una interfaz intuitiva reduce la posibilidad de errores por parte de los usuarios. Menos errores significa una mayor seguridad y confianza en el uso del sistema.

Adaptabilidad a Diferentes Plataformas:

- Desarrollar la aplicación para múltiples plataformas (móvil, navegador y web) proporciona una mayor flexibilidad y accesibilidad para los usuarios en diversos contextos.

Desarrollo en Flutter y Next.js:

Flutter para la Aplicación Móvil:

- Flutter es un marco de desarrollo de código abierto desarrollado por Google para crear aplicaciones nativas tanto para Android como para iOS desde una única base de código. La aplicación móvil desarrollada en Flutter garantizará una experiencia uniforme en ambas plataformas.

Next.js para la Página Web:

- Next.js es un marco de React que se utiliza para construir aplicaciones web. Su capacidad de representación del lado del servidor (SSR) y generación de sitios estáticos (SSG) lo hace ideal para desarrollar páginas web eficientes y rápidas. [18]

Extensión de Navegador:

- El desarrollo de una extensión de navegador brinda una capa adicional de accesibilidad y comodidad para los usuarios que prefieren utilizar el sistema desde su navegador favorito.

Elementos Clave de una Interfaz Intuitiva:

Diseño Consistente:

- Mantener la coherencia en el diseño y la disposición de elementos en todas las plataformas ayuda a los usuarios a sentirse cómodos y familiarizados.

Navegación Sencilla:

- Una estructura de navegación lógica facilita a los usuarios encontrar y utilizar las funciones que necesitan.

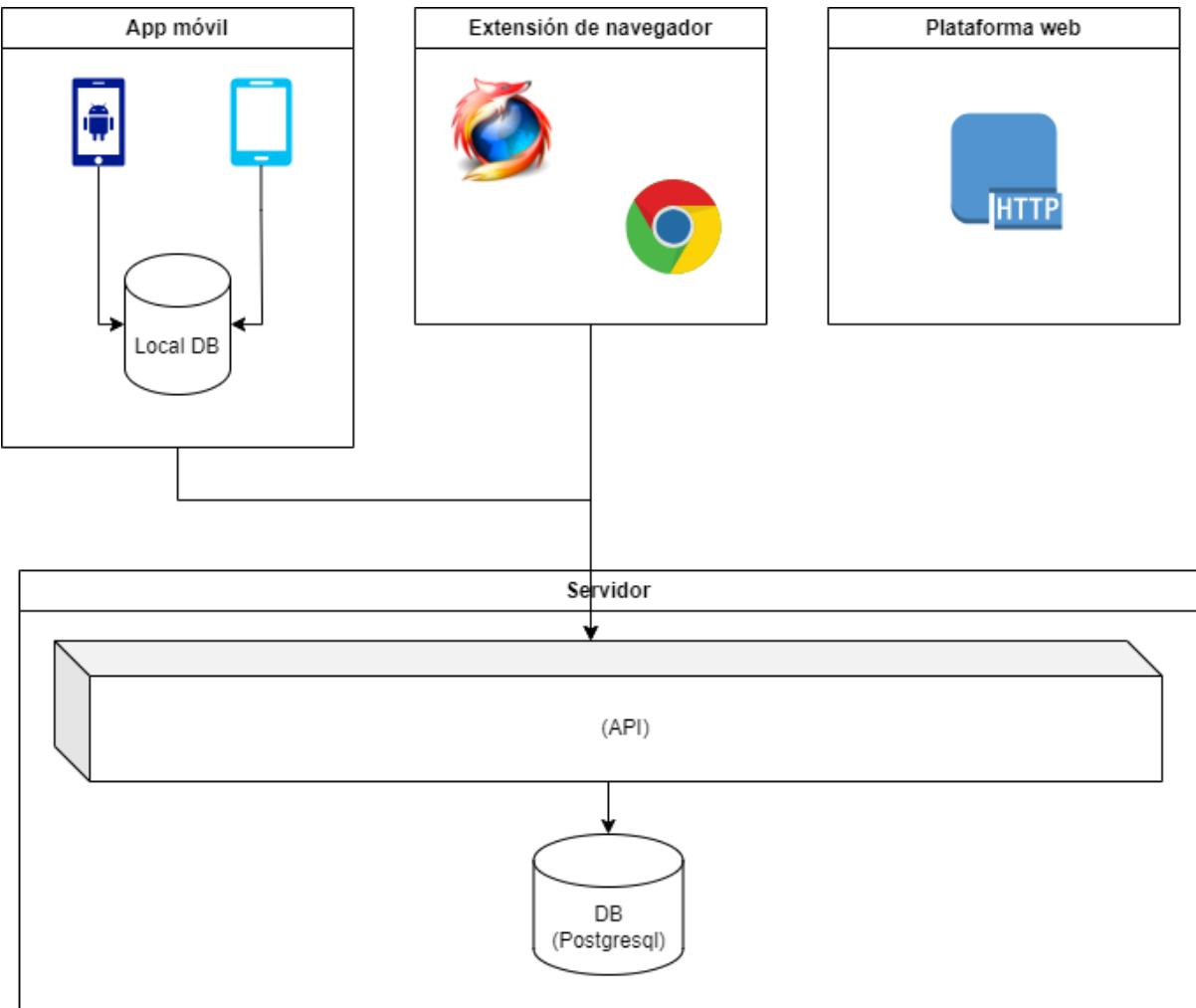
Retroalimentación Visual:

- Proporcionar retroalimentación visual, como indicadores de carga o confirmaciones de acción, ayuda a los usuarios a comprender lo que está sucediendo en la aplicación.

Instrucciones Claras:

- Incluir instrucciones claras y ayuda contextual asegura que los usuarios comprendan cómo utilizar las funciones de seguridad, como la autenticación en dos pasos.

Diagrama del proyecto



El proyecto lo podemos dividir en 4 partes:

- **App móvil:** es la herramienta principal del proyecto, la más importante. Esta tendrá su propia base de datos en cada dispositivo para que pueda funcionar offline.
- **Extensión de navegador:** al igual que la app móvil, servirá para gestionar los tokens y todos los secretos del proyecto desde el navegador.
- **Página web:** esta no permite obtener o generar tokens, la web solo servirá para informar sobre el funcionamiento del proyecto.
- **Servidor:** desarrollado en nestjs, la API es el corazón del proyecto, ya que sin este no sería posible su funcionamiento. Esta está conectada directamente a la base de datos.

2.2 Productos del proyecto

El proyecto se centrará en los siguientes aspectos:

Lo que hace:

- Generar códigos TOTP cada 30 segundos para proporcionar autenticación de 2 pasos (2FA) a las cuentas registradas, lo que añade una capa adicional de seguridad a las cuentas en línea.
- Almacenar y generar contraseñas robustas de manera segura para las cuentas de los usuarios. Esto incluye la gestión de contraseñas y garantizar la seguridad de las mismas.
- Diseñar una interfaz de usuario intuitiva y amigable que permita a los usuarios gestionar sus contraseñas y códigos TOTP de manera sencilla.
- Implementar algoritmos criptográficos sólidos para asegurar la seguridad de las contraseñas almacenadas y de los códigos TOTP generados.
- Proporcionar una solución integral que mejore la seguridad en línea de los usuarios al simplificar el proceso de acceso en línea y reducir los riesgos de seguridad.

Lo que no hace, pero podría hacer en una versión futura:

- Almacenar información crítica adicional, como tarjetas de pago, cuentas bancarias, información de bases de datos, servidores, pasaportes, licencias de software, claves SSH, carnets de identificación, suscripciones, notas seguras o privadas. Esto ampliaría la funcionalidad de la aplicación para abordar más aspectos de la seguridad y gestión de datos personales de los usuarios.

Etapa de pre proyecto:

- **Estructuración de datos:** se espera tener una documentación clara que defina la estructura de datos necesaria para almacenar contraseñas y códigos TOTP de manera segura. Esto podría incluir modelos de datos, esquemas de bases de datos y consideraciones sobre la seguridad de los datos.
- **Diseño para Aplicaciones Móviles:** Deberías entregar los diseños de las aplicaciones móviles que incluyan la interfaz de usuario (UI) y la experiencia de usuario (UX). Esto puede incluir bocetos, mockups y wireframes que representen cómo se verá y funcionará la aplicación en dispositivos móviles.
- **Diseño para Extensiones de Navegador:** Proporcionar los diseños de las extensiones de navegador, que deben ser coherentes con la marca y la funcionalidad de la aplicación móvil. Esto también puede incluir bocetos y mockups.
- **Desarrollo de la API:** Entregar la documentación y los recursos relacionados con la API que se utilizará para interactuar con la aplicación. Esto podría incluir especificaciones de endpoints, autenticación y ejemplos de uso.

Etapa del Proyecto Final:

- **Aplicaciones Móviles:** Entregar las aplicaciones móviles funcionales para diferentes plataformas, como iOS y Android. Esto incluirá la implementación de la interfaz de usuario diseñada en la etapa de pre proyecto, así como la funcionalidad para generar códigos TOTP y gestionar contraseñas.
- **Extensión de Navegador:** Proporcionar la extensión de navegador que permita a los usuarios acceder a la funcionalidad de la aplicación directamente desde su navegador. Esto debería ser coherente con el diseño previamente establecido.
- **Página Web:** Desarrollar y lanzar el sitio web asociado al servicio. Este sitio web podría ser utilizado para proporcionar información, soporte, acceso a cuentas en línea y otras funcionalidades relacionadas con el servicio.

2.4 Plan de Administración de Riesgos

Riesgo: Vulnerabilidades de Seguridad

Possible Impacto: Compromiso de la seguridad de contraseñas y códigos TOTP almacenados.

Posibles Soluciones: Realizar auditorías de seguridad periódicas, implementar prácticas de desarrollo seguro, y mantenerse actualizado con las mejores prácticas de seguridad.

Riesgo: Problemas de Integración

Possible Impacto: Dificultades al integrar la aplicación con diferentes plataformas (iOS, Android, navegadores).

Posibles Soluciones: Realizar pruebas exhaustivas de integración, seguir las pautas y estándares de desarrollo de cada plataforma, y mantener una comunicación constante con las comunidades de desarrollo.

Riesgo: Pérdida de Datos

Possible Impacto: Pérdida de contraseñas o códigos TOTP almacenados debido a fallos en la aplicación o en la infraestructura.

Posibles Soluciones: Implementar copias de seguridad regulares y almacenar datos de forma redundante en servidores seguros.

Riesgo: Cambio en Requisitos de Plataforma

Possible Impacto: Cambios inesperados en las políticas de desarrollo de Apple, Google u otras plataformas pueden afectar la funcionalidad de la aplicación.

Posibles Soluciones: Mantenerse informado sobre las actualizaciones de políticas, ajustar la aplicación según sea necesario y planificar con anticipación para cualquier cambio previsto.

Riesgo: Problemas de Escalabilidad

Possible Impacto: A medida que la base de usuarios crece, la aplicación podría enfrentar problemas de rendimiento y escalabilidad.

Posibles Soluciones: Diseñar la arquitectura de la aplicación con la escalabilidad en mente, realizar pruebas de carga y optimizar el rendimiento de manera continua.

2.5 Presupuesto

A continuación se presenta la tabla de presupuesto del **primer año** del proyecto.

Descripción	RD\$	US\$
Hardware y software (PC)	45,000	796.46
Apple developer (anual)	5,650	100
Google developer (anual)	1412.5	25
Hosting (AWS -> Amplify, Elastic Beanstalk, RDS)	X	X
iPhone 11	15,000	265.48
Samsung A56	10,000	176.99
Macbook Pro 16"	141,250	2,500
Contingencia (10%)	21,831.25	386.393
Total	240,143.75	4,250.323

i. En el caso del hosting sería gratuito por un año y posterior a esto se cobraría por uso. [20]

2.6 Definición de la demostración

La realización de la presentación de este proyecto requiere la consideración de las siguientes especificaciones:

Lugar:

- No existen limitaciones en cuanto al lugar de presentación.

Escenario/Ambientación:

- La situación del momento no es relevante, ya que el sistema, al tratarse de una aplicación móvil, depende únicamente de la disponibilidad de conexión a internet.

Condiciones:

- Conexión a internet

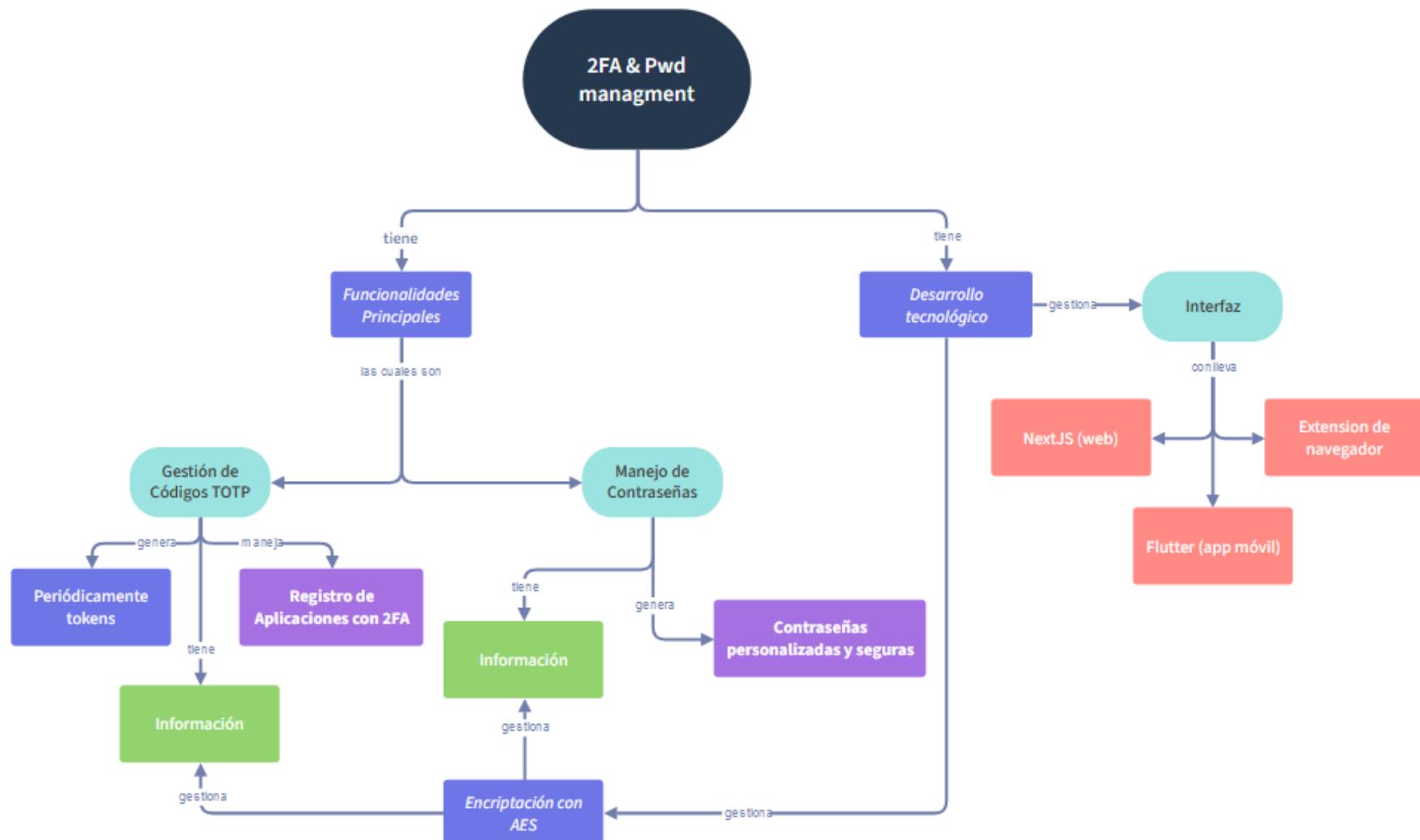
Equipos Necesarios:

1. Celular con sistema operativo Android.
2. iPhone.
3. Macbook.

2.7 Análisis y Diseño

2.7.1 Análisis

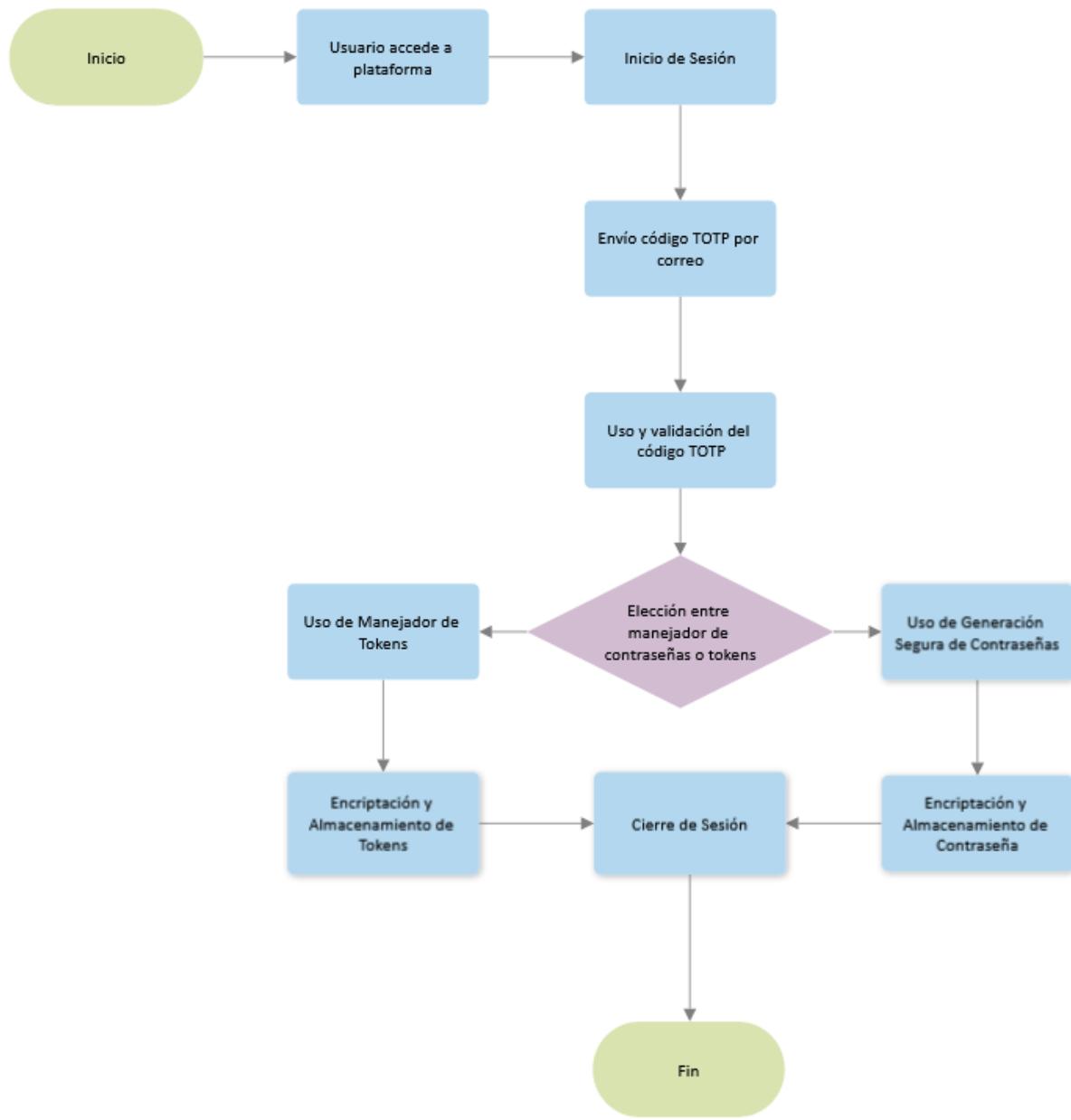
Mapa conceptual



El proyecto se puede desglosar en dos dimensiones fundamentales: funcionalidades principales y desarrollo tecnológico. Las funcionalidades principales se centran en la gestión de códigos TOTP y el manejo de secretos, específicamente contraseñas. En cuanto a la gestión de códigos TOTP, se generan periódicamente tokens y se permite registrar diversas aplicaciones que admitan la doble autenticación. Por otro lado, el manejo de secretos posibilita el almacenamiento y generación de contraseñas seguras.

En lo que respecta al desarrollo tecnológico, este se puede dividir en dos componentes esenciales: interfaz y encriptación mediante el algoritmo AES. La encriptación con AES reviste una importancia crucial, ya que toda la información recopilada de cada usuario es sensible y, en caso de ser adquirida por una persona no autorizada, podría utilizarse de manera indebida. Por su parte, la interfaz engloba todo el cliente del proyecto, abarcando la aplicación móvil, la extensión del navegador y la página web.

Diagrama de flujo:



Este diagrama ilustra el flujo que un usuario seguiría al utilizar la aplicación. En primer lugar, el usuario accede a la plataforma y procede a iniciar sesión. Para verificar su autenticidad, se le envía un código TOTP a su correo electrónico. Una vez que el usuario recibe y valida este código TOTP, se le presentan dos opciones para utilizar la aplicación: el gestor de tokens o el gestor de secretos. Después de completar alguna de estas funcionalidades, el usuario tiene la opción de elegir la otra o cerrar la aplicación para concluir este flujo.

Casos de uso

1. Sign Up

Actor Principal	Usuario no registrado
Descripción	El usuario completa el proceso de registro proporcionando información como nombre, correo electrónico.
Extensiones	Ninguna
Incluye	Ninguna

2. Sign In

Actor Principal	Usuario registrado
Descripción	El usuario inicia sesión proporcionando su correo electrónico y valida su inicio de sesión con un código enviado a su correo. Se verifica la autenticidad de las credenciales.
Extensiones	Ninguna
Incluye	<ul style="list-style-type: none">- Manejar tokens- Manejar contraseñas

3. Manejar tokens

Actor Principal	Usuario autenticado
Descripción	<ul style="list-style-type: none">- Crear Token: el usuario crea un token leyendo un código QR o ingresando manualmente el secreto. Se genera un token válido para la autenticación.- Generar Tokens Periódicamente: el usuario utiliza la funcionalidad para generar tokens periódicamente, cada 30 segundos.
Extensiones	Ninguna
Incluye	Ninguna

4. Manejar contraseñas

Actor Principal	Usuario registrado
Descripción	El usuario inicia sesión proporcionando su correo electrónico y valida su inicio de sesión con un código enviado a su correo. Se verifica la autenticidad de las credenciales.
Extensiones	Ninguna
Incluye	Ninguna

2.7.2 Diseño

Mockups

Sección de tokens:

En este apartado se podrán manejar todos los tokens almacenados de las diversas aplicaciones. En el mockup hay 3 diseños distintos de las tarjetas.

Estos tipos de tarjetas se conocen como:

- **Tarjeta explícita:** muestra todos los detalles del código e incluye un botón que indica que puede ser copiado dándole clic al botón o a la carta en general. En el mockup es la primera.
- **Tarjeta implícita:** al igual que la primera, muestra los detalles pero sin el botón de copiar. De igual manera se puede copiar el token dándole clic a la tarjeta. Este diseño en el mockup sería el 2do y 3ro. En el caso del 3er diseño, se muestra un código en rojo, esto es para los usuarios que quieran visualizar el siguiente código antes que se expire el ya generado.
- **Tarjeta compacta:** esta no tiene tanto diseño, ya que reducimos el tamaño para que el usuario pueda visualizar más tokens en una misma pantalla. Al igual que el segundo, tiene los 2 diseños de cómo se vería con la opción de “ver siguiente token”.

Sección de secrets

En este apartado se podrán manejar todos los secretos de la aplicación. Los secretos son toda la data “sensible” o la data que se quiera almacenar de manera segura. Esta incluye la generación y manejo de las contraseñas, almacenamiento de tarjetas de crédito/débito, cuentas bancarias, clave de conexión a base de datos, clave ssh, etc.

Cada secreto tendrá un tipo y según el tipo que se elija el usuario se adaptarán los formularios al tipo de secreto.

Sección de dispositivos

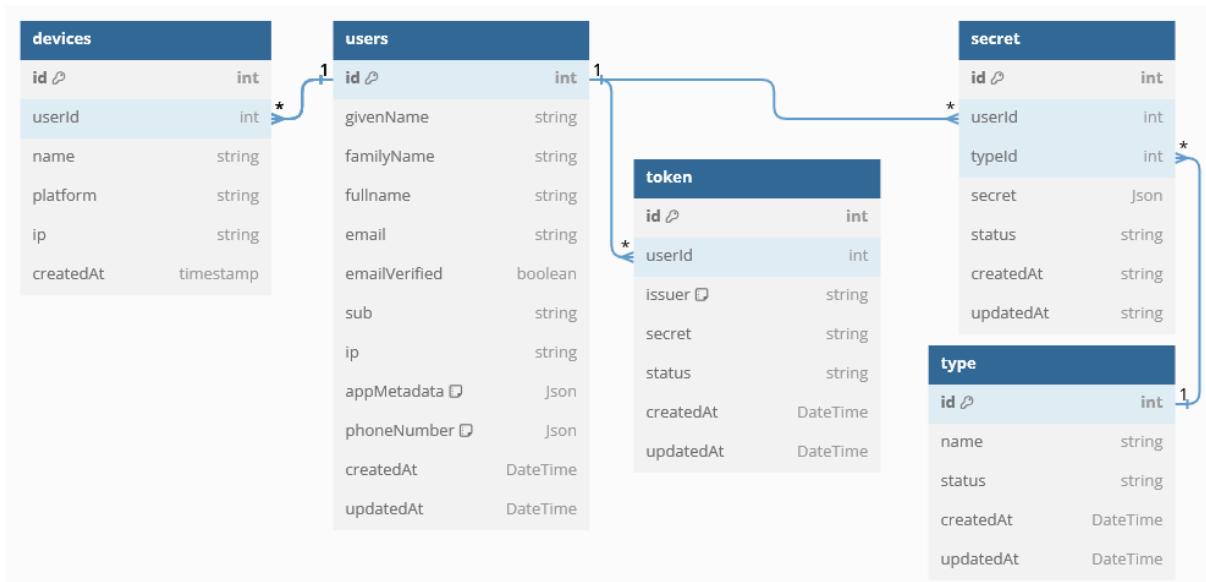
En este apartado se mostrarán todos los dispositivos que tengan la sesión iniciada. Desde este se podrá cerrar sesión o eliminar cualquier dispositivo. **Sección de configuración**

En este apartado se podrán realizar diversas funciones, tales como:

- **Seguridad:** poder activar el inicio de sesión con los datos biométricos.
- **Apariencia:** para alternar entre los diversos estilos de la aplicación. *i. puede no estar en la versión de salida.*
- **Tokens:** este apartado serviría para importar los tokens desde otras aplicaciones de doble autenticación. *i. puede no estar en la versión de salida.*
- **Browser extensión:** este apartado, en caso de llegar a ser incluído desde la app móvil, nos serviría para leer un código QR, en caso que se termine desarrollando de esa manera, que estará en la extensión de navegador, así puede vincular todos los tokens y los secretos de este usuario.

Es importante aclarar que todos estos mockups están sujetos a cambios.

Modelo de datos [19]



En el modelo de datos, todas las tablas, a excepción de la de “type” tienen una relación directa con el usuario.

Token: este modelo hace referencia al objeto que se necesita para poder generar los códigos TOTP (token) a los usuario por cada cuenta registrada.

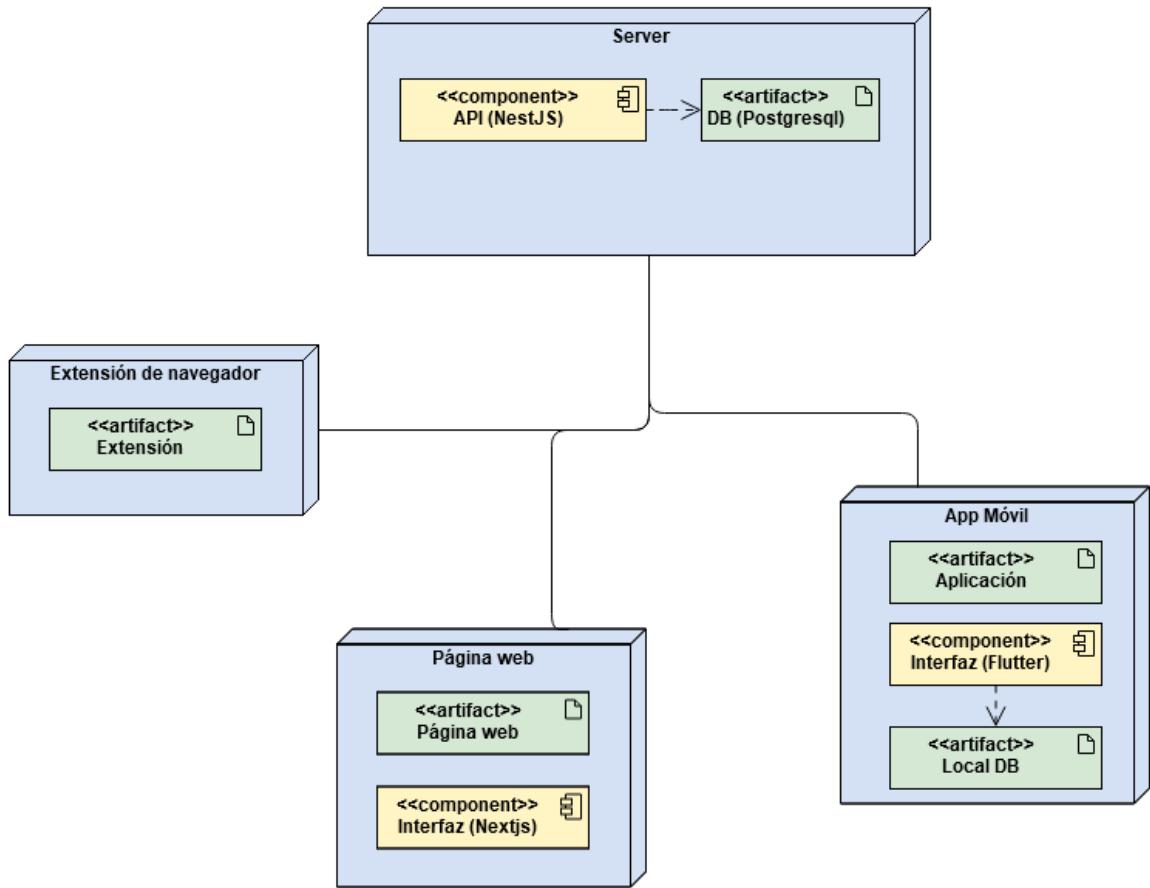
Secret: este modelo es el que se encarga de almacenar cualquier tipo de secreto (contraseña, claves ssh, etc) y se adapta a cualquier información, sin importar la cantidad de características que tenga.

Type: hace referencia al tipo de secret. Cada secret tiene un tipo, con el cual se hace referencia a lo que contiene cada JSON.

Devices: almacena la información de cada dispositivo vinculado al usuario.

Users: almacena toda la información relevante de un usuario.

Diagrama de despliegue



En este diagrama de despliegue, podemos descomponer el proyecto en dos capas principales: el servidor y el cliente, este último subdividido en tres componentes.

En el lado del servidor, encontramos el componente principal, la API, que se conecta directamente con la base de datos, implementada en Postgresql.

En el lado del cliente, las tres subdivisiones son las siguientes:

- APP móvil:** La interfaz de la aplicación está conectada al localDB. Este localDB es esencial para el funcionamiento offline de la aplicación y también es necesario para generar códigos TOTP desde el lado del cliente. Su presencia es crucial para el funcionamiento sin conexión, reduciendo los costos asociados con el mantenimiento del servidor.
- Extensión de navegador:** Esta extensión comparte funcionalidades similares a la aplicación móvil, permitiendo la obtención de códigos TOTP y la gestión de secretos.

Al igual que en la aplicación móvil, la conexión con el localDB facilita el funcionamiento offline.

3. **Página web:** Diseñada para presentar el proyecto, sus características, funcionalidades y modo de operación. Desde la página web, no es posible acceder al apartado de generación de tokens y/o contraseñas, sirviendo principalmente como un medio informativo sobre el proyecto.

Desarrollo e Implementación

Método: *Authenticate*

Descripción: Este método recibe el correo del usuario y valida si tiene una cuenta. Si la tiene, se le envía un token para su validación. Si no tiene una cuenta, se crea una y luego se envía el token para validar su acceso.

i: Servicio y controlador en los anexos.

Test del endpoint:

Responses

Curl

```
curl -X 'POST' \
'http://localhost:3000/auth' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "email": "eliamamps07@outlook.com"
}'
```



Request URL

```
http://localhost:3000/auth
```

Server response

[Code](#) [Details](#)

200

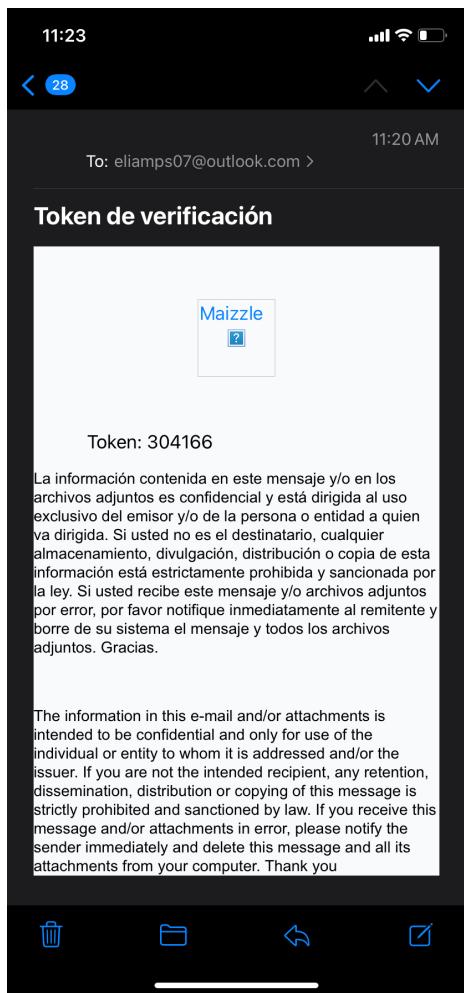
Response body

```
{
  "message": "Token enviado"
}
```



Download

Mensaje en el correo:



De esta manera se recibe el token, el cual validamos en nuestro próximo método.

Método: *Verify*

Descripción: Este método valida la vigencia del token proporcionado junto con la dirección de correo. Si el token está activo, devuelve el access_token, permitiendo así llamadas a los demás endpoints.

Test del endpoint

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:3000/auth/verify' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "email": "eliamps07@outlook.com",
    "token": 304166
}'
```



Request URL

```
http://localhost:3000/auth/verify
```

Server response

Code Details

200

Response body

```
{  
  "message": "Usuario autenticado",  
  "access_token": {  
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjIsImhdCI6MTcwMDc1MzUzNSwiZXhwIjoxNzAwODM5OTM1fQ.VDm_wLtchfLXU18minL4o0RY11PGhArqFY14XDbSG-Q"  
  }  
}
```



Download

Response headers

```
connection: keep-alive  
content-length: 210  
content-type: application/json; charset=utf-8  
date: Thu, 23 Nov 2023 15:32:15 GMT  
etag: W/"d2-86H1qZ3JcqFFkL+aM6ZVwj1t7IU"  
keep-alive: timeout=5  
x-powered-by: Express
```

De esta manera obtenemos el accessToken.

Método: *Crear objeto para generar tokens*

Descripción: Este bloque toma como parámetro una URI que contiene toda la información necesaria para generar tokens. Alternativamente, puede recibir el secreto y otros parámetros para complementar este objeto y así generar tokens.

Método: *Generar tokens*

Descripción: Se buscan todos los objetos almacenados por el usuario, se itera sobre cada uno para generar los tokens y, finalmente, se muestran al usuario.

Test del endpoints:

Link: <http://bit.ly/3RioXrt>

Método: *Crear un secreto*

Descripción: El usuario debe elegir qué tipo de secreto quiere almacenar. Algunos ejemplos pueden ser: contraseñas, tarjetas de crédito, cuentas de banco, conexiones a bases de datos, claves SSH, etc. Dependiendo del tipo que elija, debe enviar la data correspondiente.

En este caso sólo se mostrarán los endpoints para las contraseñas, ya que es el feature que estará desde un inicio en la plataforma y es lo que está pautado para la entrega final.

Test del endpoint:

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:3000/secrets/pwd' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlc2VySWQiOjIsImhdCi6MTcwMDc1ODE4OCwiZXhwIjoxNzAwODQ0NTg4fQ.5W_gZCmd-uulfd3rFzxPZaPpX08272CgE6qFZyMGXk' \
  -H 'Content-Type: application/json' \
-d '{
  "uppers": true,
  "lowers": false,
  "numbers": true,
  "specials": true,
  "length": 12,
  "userId": 2,
  "issuer": "Instagram",
  "typeId": 1
}'
```



Request URL

```
http://localhost:3000/secrets/pwd
```

Server response

Code Details

200

Undocumented Response body

```
{
  "id": 10,
  "userId": 2,
  "typeId": 1,
  "secret": {
    "issuer": "U2FsdGVkX19MtpGPJ8iiV9KZsMUMRa0bCKr1DVBtLOA=",
    "password": "U2FsdGVkX1900ySJ66643URf516BpjstTrbrAn1LtR88="
  },
  "status": "enable",
  "createdAt": "2023-11-23T16:50:09.451Z",
  "updatedAt": null
}
```



Download

Método: Obtener todas las contraseñas descriptadas

Descripción: Este método obtiene el user ID y en base a esto obtiene todas las contraseñas del usuario y las descripta para ser presentadas.

Test del endpoint

Responses

Curl

```
curl -X 'GET' \
'http://localhost:3000/secrets/2' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlc2VySWQiOjIsImhdCI6MTcwMDc1ODE40CiZXhwIjoxNzAwODQ0NTg4fQ.5W_gZCmd-uulfod3rFzxPZaPpX08272CgE6qFZyMGXk'
```



Request URL

```
http://localhost:3000/secrets/2
```

Server response

Code Details

200

Response body

```
[  
  {  
    "password": "4.T[%.U9-[^.,"  
    "issuer": "Instagram",  
    "type": "Passwords"  
  }  
]
```



Download

Es crucial destacar que toda la información se encripta y desencripta en el mismo servidor para realizar pruebas y presentaciones. Se adopta la estrategia de realizar el proceso de encriptación y desencriptación desde el cliente. Esta elección se basa en evitar posibles intervenciones y riesgos de seguridad entre el cliente y el servidor, asegurando la integridad de la información.

Conclusiones y recomendaciones

El proyecto de autenticación en dos pasos y manejo de secretos ha logrado desarrollar una API robusta que garantiza la seguridad y confidencialidad de la información sensible de los usuarios. Se ha implementado el algoritmo AES y otras medidas de seguridad, asegurando la integridad de los datos almacenados. La API, completamente desarrollada, incluye la generación de códigos TOTP y la capacidad de manejar contraseñas, con una funcionalidad altamente escalable para la incorporación de diferentes tipos de secretos en el futuro.

Además del avance en la API, se ha creado un diseño preliminar de la aplicación móvil, marcando un hito significativo en la planificación visual de la interfaz de usuario. Las conclusiones reflejan el cumplimiento de los objetivos, destacando la seguridad, escalabilidad y versatilidad del proyecto.

Las recomendaciones se centran en el siguiente paso crítico: el desarrollo de las aplicaciones móviles. Asimismo, se propone la exploración de nuevas funcionalidades para la gestión de secretos, enriqueciendo la aplicación y aumentando su atractivo.

Se reconoce la limitación de recursos técnicos, específicamente la falta de acceso a una computadora Mac con XCode. Se insta a buscar alternativas para superar este obstáculo. Además, se destaca la consideración de costos asociados con la publicación en las tiendas de aplicaciones, sugiriendo un análisis detallado para una asignación eficiente de recursos financieros.

Referencias

- [1] Gallegos, L. (2023, 27 abril). 2FA por aplicaciones y el protocolo responsable - Luis Gallegos - Medium. *Medium*. <https://medium.com/@luisgallegosxt/2fa-por-aplicaciones-y-el-protocolo-responsable-8fb29144f83>
- Brecha de datos de LinkedIn (2012), Ataque a iCloud (2014), Brechas de datos en Yahoo (2013–2014)*
- [2] Zajac, G. (2023, 27 noviembre). *Who are we? Meet the people behind 2FAS app.* 2FA Authenticator App (2FAS). <https://2fas.com/about-us/>
- [3] Features - Authy. (2023, 27 noviembre). Authy. <https://authy.com/features/>
- [4] Password & Secrets Management | Keeper Security. (2023, 27 noviembre). Keeper® Password Manager & Digital Vault. <https://www.keepersecurity.com/>
- [5] Padua, M., & Padua, M. (2023, 4 noviembre). *2FA, una estrategia de prevención ante ciberataques.* IT Masters Mag. <https://www.itmastersmag.com/ciberseguridad/2fa-una-estrategia-de-prevencion-ante-ciberataques/>
- [6] Equipo editorial de IONOS. (2019, 9 abril). *Time-based one-time password: TOTP explicado.* IONOS Digital Guide. <https://www.ionos.es/digitalguide/servidores/seguidad/totp/>
- [7] Periódico, E. (2019, 20 junio). La importancia de usar un gestor de contraseñas. *El Periódico de Aragón.* <https://www.elperiodicodearagon.com/el-mostrador/2019/06/20/importancia-gestor-contrasenas-46634132.html>
- [8] Hurtado, J. S. (2023, 29 mayo). *Qué es la criptografía y para qué sirve.* Thinking for Innovation. <https://www.iebschool.com/blog/que-es-la-criptografia-y-para-que-sirve-finanzas/>
- [9] GeneTecTM. (s. f.). Genetec™. <https://www.genetec.com/es/blog/ciberseguridad/como-funciona-la-autenticacion>
- [10] Equipo editorial de IONOS. (2019, 9 abril). *Time-based one-time password: TOTP explicado.* IONOS Digital Guide. <https://www.ionos.es/digitalguide/servidores/seguidad/totp/>
- [11] ¿Qué es la encriptación y cómo funciona? | Google Cloud | Google Cloud. (s. f.). Google Cloud. <https://cloud.google.com/learn/what-is-encryption?hl=es-419>

[12] *Algoritmo criptográfico - Glosario de MDN Web Docs: Definiciones de términos relacionados con la web* | MDN. (s. f.). MDN Web Docs.

<https://developer.mozilla.org/es/docs/Glossary/Cipher>

[13] *¿Qué son las extensiones?* - Mozilla | MDN. (2023, 2 agosto). MDN Web Docs.
https://developer.mozilla.org/es/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions

[14] *¿Qué es una aplicación móvil?* – Anincubator. (s. f.).

<https://anincubator.com/que-es-una-aplicacion-movil/>

[15] a3SIDES. (2023, 20 febrero). Qué es el doble factor de autenticación y para qué sirve. a3SIDES.

<https://www.a3sides.es/blog/que-es-doble-factor-autenticacion/>

[16] Araya, S. (2023, 3 febrero). Interfaz intuitiva: ¿Qué es y su importancia? Nubox: *Interfaz Intuitiva*.

<https://blog.nubox.com/software/interfaz-intuitiva-que-es>

[17] *Explicación de la encriptación AES (Advanced Encryption Standard)* - Kraden Blog. (2022, 17 marzo). Explicación de la

encriptación AES (Advanced Encryption Standard) - Kraden Blog.
<https://blog.kraden.com/es/aes-256-encryption>

[18] Morales, J., & Morales, J. (2023, 21 febrero).

¿Qué es Next.js y cuáles son sus beneficios? WWWhat's new.

<https://wwwwhatsnew.com/2023/02/21/que-es-next-js-y-cuales-son-sus-beneficios/>

[19] *A free database designer for developers and analysts.* (s. f.). <https://dbdiagram.io/home>

[20] *Amazon RDS Free Tier | Cloud Relational Database* | Amazon Web Services. (s. f.).

Amazon Web Services, Inc.
<https://aws.amazon.com/rds/free/>

Intro to AWS Amplify (4:00). (s. f.). [Vídeo]. Amazon Web Services, Inc.

<https://aws.amazon.com/amplify/>
Introduction to AWS Elastic Beanstalk (2:14). (s. f.). [Vídeo]. Amazon Web Services, Inc.

<https://aws.amazon.com/es/elasticbeanstalk/>

Anexos

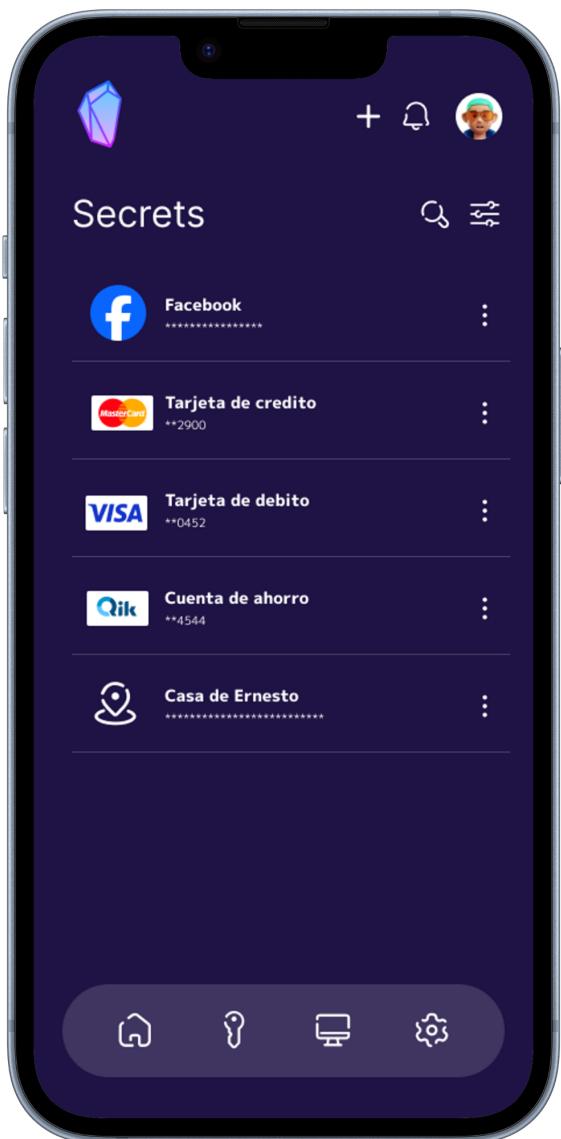
Diseño

Mockups

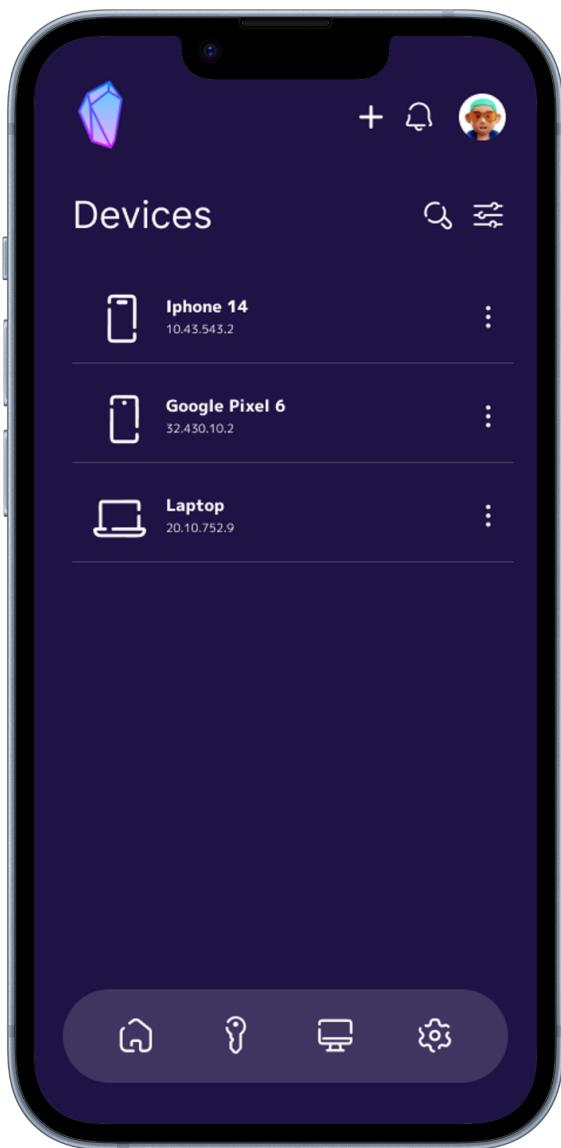
Sección de tokens:



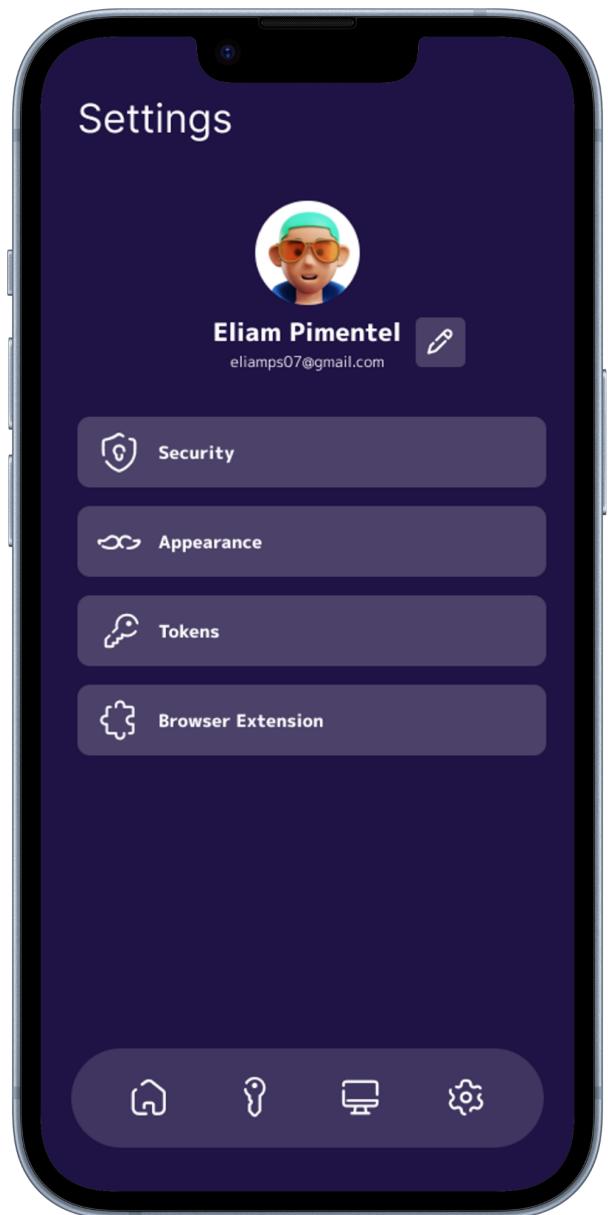
Sección de secrets:



Sección de dispositivos



Sección de configuración



Desarrollo e implementación

Servicio para autenticar usuarios



```
1 // Iniciar sesión o crear cuenta y enviar token al correo
2 async authenticate(email: string): Promise<AuthEntity> {
3     try {
4         // Paso 1: Buscar el usuario por email
5         const user = await this.prisma.user.findUnique({
6             where: { email: email },
7         });
8         // En caso que el usuario no exista, debemos crear uno.
9         if (!user) {
10             // Paso 1.1: Generar un sub de 12 caracteres con nanoid, id unico para el usuario
11             const generateSub = async () => {
12                 const sub = customAlphabet(
13                     `${this.uppers}${this.lowers}${this.numbers}`,
14                     12,
15                 );
16                 const user = await this.findUniqueBySub(sub());
17                 return user ? generateSub() : sub();
18             };
19             const sub = await generateSub();
20             // Paso 1.2: Crear el usuario con el email y el sub
21             await this.prisma.user.create({
22                 data: {
23                     email: email,
24                     sub,
25                 },
26             });
27         }
28
29         // Paso 2: Generar un token de 6 digitos con math.random y enviarlo al correo
30         return await this.createEmailToken(email);
31     } catch (error) {
32         console.log(error);
33         throw new InternalServerErrorException('Error creating user');
34     }
35 }
```

Controlador para autenticar al usuario

```
1  @Post()
2    @ApiOperation({ type: AuthEntity })
3    async authenticate(@Body() { email }: LoginDto, @Res() res) {
4      try {
5        const login = await this.authService.authenticate(email);
6        return res.status(200).json({ message: 'Token enviado' });
7      } catch (error) {
8        console.log(error);
9        return res.status(200).json({ error });
10     }
11   }
```

Servicio para verificar el usuario

```
1 // Verificar el token y actualizar el usuario
2 async verifyEmailToken(email: string, token: number): Promise<any> {
3     try {
4         // Paso 1: Buscar el usuario por email
5         const user = await this.prisma.user.findUnique({ where: { email } });
6         // Paso 2: Buscar el token y el email para verificar que la fecha de este no esté expirada
7         const emailToken = await this.prisma.emailToken.findFirst({
8             where: { email, token, expiredAt: { gt: new Date() } },
9         });
10        // En caso que no se encuentre el token, lanzar un error
11        if (!emailToken) {
12            throw new NotFoundException('Token no encontrado');
13        }
14        // Paso 3: Verificar que el usuario no esté verificado, en caso de no estarlo actualizarlo
15        if (!user.emailVerified) {
16            await this.prisma.user.update({
17                where: { email },
18                data: { emailVerified: true },
19            });
20        }
21        // Paso 4: Eliminar todos los tokens del usuario
22        await this.prisma.emailToken.deleteMany({ where: { email } });
23
24        // Paso 5: Retornar el token de acceso
25        return { accessToken: this.jwtService.sign({ userId: user.id }) };
26    } catch (error) {
27        console.log(error);
28        throw new NotFoundException('Token not found');
29    }
30}
```

Servicio para crear el modelo de los códigos TOTP

```
● ● ●
1 // Crear objeto para generar tokens
2 async create(data: Prisma.TokenCreateInput): Promise<Token> {
3     // Paso 1: Verificamos si en la data viene una URI
4     if (data.uri) {
5         // Paso 1.1: Si viene una URI, la parseamos con la libreria otpauth y esta nos devuelve un objeto con los datos del token
6         const otpauth = OTPAuth.URI.parse(data.uri);
7         // Paso 1.2: Encriptamos los datos del token con el algoritmo AES y la llave que tenemos en el archivo .env
8         data.issuer = CryptoJS.AES.encrypt(
9             otpauth.issuer,
10            process.env.CRYPTO_KEY,
11            ).toString();
12         data.label = CryptoJS.AES.encrypt(
13             otpauth.label,
14             process.env.CRYPTO_KEY,
15             ).toString();
16         data.secret = CryptoJS.AES.encrypt(
17             otpauth.secret.toString(),
18             process.env.CRYPTO_KEY,
19             ).toString();
20         data.uri = CryptoJS.AES.encrypt(
21             data.uri,
22             process.env.CRYPTO_KEY,
23             ).toString();
24         // Paso 1.3: Creamos el cuerpo para generar tokens en la base de datos
25         return this.prismaService.token.create({ data });
26     }
27     // Paso 2: Si no viene una URI, creamos el cuerpo para generar tokens y los encriptamos
28     data.issuer = CryptoJS.AES.encrypt(
29         data.issuer,
30         process.env.CRYPTO_KEY,
31         ).toString();
32     data.label = CryptoJS.AES.encrypt(
33         data.label,
34         process.env.CRYPTO_KEY,
35         ).toString();
36     data.secret = CryptoJS.AES.encrypt(
37         data.secret.toString(),
38         process.env.CRYPTO_KEY,
39         ).toString();
40     // Paso 3: Almacenamos el objeto para generar los tokens en la base de datos
41     return this.prismaService.token.create({ data });
42 }
```

Controlador para crear el modelo de los códigos TOTP

```
● ● ●  
1  @Post()  
2    @UseGuards(JwtAuthGuard)  
3    @ApiBearerAuth()  
4    async create(@Body() data: TokenDto, @Res() res) {  
5      try {  
6        const token = await this.tokenService.create(data);  
7        return res.status(200).json(token);  
8      } catch (error) {  
9        return res.status(500).json({ error: 'Internal Server Error' });  
10     }  
11   }
```

Servicio para generar los tokens

```
● ● ●
1 // Obtener todos los tokens validados de un usuario
2 async findAllValidatedById(id: number): Promise<any[]> {
3     // Paso 1: Buscamos todos los objetos almacenados por el usuario
4     const data = await this.findAllById(id);
5     // Paso 2: Iteramos sobre los objetos para generar los tokens
6     const tokens = data.map((t: Token) => {
7         // Paso 2.1: Verificamos que el token esté activo, si no lo está, retornamos null
8         if (t.status !== 'enable') return;
9         // Paso 2.2: Desencriptamos los datos del objeto para generar el token
10        const labelBytes = CryptoJS.AES.decrypt(t.label, process.env.CRYPTO_KEY);
11        const label = labelBytes.toString(CryptoJS.enc.Utf8);
12        const issuerBytes = CryptoJS.AES.decrypt(
13            t.issuer,
14            process.env.CRYPTO_KEY,
15        );
16        const issuer = issuerBytes.toString(CryptoJS.enc.Utf8);
17        const secretBytes = CryptoJS.AES.decrypt(
18            t.secret,
19            process.env.CRYPTO_KEY,
20        );
21        const secret = secretBytes.toString(CryptoJS.enc.Utf8);
22        // Creamos display, que sera lo que mostraremos al final
23        const display: any = {
24            label: label,
25            issuer: issuer,
26        };
27        // Paso 2.3: Verificamos si el objeto tiene una URI
28        if (t.uri) {
29            // Paso 2.3.1: Si tiene una URI, la desencriptamos
30            const uriBytes = CryptoJS.AES.decrypt(t.uri, process.env.CRYPTO_KEY);
31            const uri = uriBytes.toString(CryptoJS.enc.Utf8);
32            // Paso 2.3.2: Parseamos la URI con la libreria otpauth y esta nos devuelve un objeto con los datos del token
33            const totp = OTPAuth.URI.parse(uri);
34            // Paso 2.3.3: Generamos el token con los datos del objeto
35            const token = totp.generate();
36            // Paso 2.3.4: Agregamos el token al objeto display
37            display.token = token;
38            return display;
39        }
40        // Paso 2.4: Si no tiene una URI, creamos un objeto para generar el token
41        const totp = new OTPAuth.TOTP({
42            issuer: issuer,
43            label: label,
44            secret: secret,
45            algorithm: 'SHA1',
46            digits: 6,
47            period: 30,
48        });
49        // Paso 2.5: Generamos el token con los datos del objeto
50        const token = totp.generate();
51        // Paso 2.6: Agregamos el token al objeto display
52        display.token = token;
53        return display;
54    });
55    // Paso 3: Retornamos los tokens
56    return tokens;
57 }
```

Controlador para generar tokens:

```
1 @Get('/t/:id')
2 @UseGuards(JwtAuthGuard)
3 @ApiBearerAuth()
4 async findAllValidatedByUserId(@Param('id') id: number, @Res() res) {
5   try {
6     const token = await this.tokenService.findAllValidatedByUserId(id);
7     return res.status(200).json(token);
8   } catch (error) {
9     return res.status(500).json({ error });
10  }
11 }
```

Crear un secreto

```
1 // Generar contraseña por userId
2 async generate(data: GeneratePasswordDto): Promise<Secrets> {
3     // Paso 1: Desestructuramos la data que viene del cliente
4     try {
5         const {
6             uppers,
7             lowers,
8             numbers,
9             specials,
10            length,
11            userId,
12            issuer,
13            typeId,
14        } = data;
15     // Paso 2: Creamos la configuración del nanoid según los parametros que nos envia el cliente
16     const _nanoid = customAlphabet(
17         `${uppers ? this._uppers : null}${lowers ? this._lowers : null}${
18             numbers ? this._numbers : null
19         }${specials ? this._specials : null}`,
20         length,
21     );
22     // Paso 3: Generamos la contraseña con nanoid
23     const password = _nanoid();
24     // Paso 4: Encriptamos la contraseña y el issuer con el algoritmo AES y la llave que tenemos en el archivo .env
25     const passwordEncrypted = CryptoJS.AES.encrypt(
26         password,
27         process.env.CRYPTO_KEY,
28     ).toString();
29
30     const issuerEncrypted = CryptoJS.AES.encrypt(
31         issuer,
32         process.env.CRYPTO_KEY,
33     ).toString();
34     // Paso 5: Creamos el cuerpo para almacenar la contraseña en la base de datos
35     const secret = {
36         password: passwordEncrypted,
37         issuer: issuerEncrypted,
38     };
39     // Paso 6: Almacenamos la contraseña en la base de datos
40     return this.prismaService.secrets.create({
41         data: {
42             userId,
43             typeId,
44             secret,
45         },
46     });
47     } catch (error) {
48         console.error(error);
49         throw error;
50     }
51 }
```

Controlador para crear un secreto

```
1  @Post('pwd')
2    @UseGuards(JwtAuthGuard)
3    @ApiBearerAuth()
4    async generate(@Body() data: GeneratePasswordDto, @Res() res) {
5      try {
6        const password = await this.secretsService.generate(data);
7        return res.status(200).json(password);
8      } catch (error) {
9        return res.status(500).json({ error });
10     }
11   }
```

Servicio para desencriptar todos los secrets

```
1 // Listar contraseña e issuer desencriptadas por userId
2 async getDecryptedPasswordsByUserId(userId: number): Promise<any[]> {
3     try {
4         // Paso 1: Buscamos todos los secretos del usuario
5         const secrets = await this.findAllByUserId(userId);
6         // Paso 2: Iteramos entre todos los secretos
7         const passwordsDecrypted = await Promise.all(
8             secrets.map(async (p: Secrets) => {
9                 const type: Type = await this.typeService.findUniqueType(p.typeId);
10                // Paso 2.1: Creamos una variable para almacenar el secreto.
11                // Lo almacenamos as unknown para que no nos de error al momento de
12                // desestructurar el JSON, ya que no tiene una estructura fija
13                const secrets = p.secret as unknown as Pwd;
14                // Paso 2.2: Desencriptamos la contraseña y el issuer con el algoritmo AES
15                const issuerBytes = CryptoJS.AES.decrypt(
16                    secrets.issuer,
17                    process.env.CRYPTO_KEY,
18                );
19                const issuer = issuerBytes.toString(CryptoJS.enc.Utf8);
20                const passwordBytes = CryptoJS.AES.decrypt(
21                    secrets.password,
22                    process.env.CRYPTO_KEY,
23                );
24                const password = passwordBytes.toString(CryptoJS.enc.Utf8);
25                // Paso 2.3: Retornamos la contraseña y el issuer desencriptados
26                return {
27                    password,
28                    issuer,
29                    type: type.name,
30                };
31            }),
32        );
33        return passwordsDecrypted;
34    } catch (error) {
35        console.error(error);
36        throw error;
37    }
38}
```

Controlador para desencriptar todos los secrets

```
1  @Get(':id')
2    @UseGuards(JwtAuthGuard)
3    @ApiBearerAuth()
4    async getDecryptedPasswords(@Param('id') id: number, @Res() res) {
5      try {
6        const passwords =
7          await this.secretsService.getDecryptedPasswordsByUserId(id);
8        return res.status(200).json(passwords);
9      } catch (error) {
10        return res.status(500).json({ error });
11      }
12    }

```