Escuela Politécnica Superior

Desarrollo de un servicio de autentificación de factor múltiple

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Luis Pérez Sevilla

Tutor/es:
Rafael Ignacio Álvarez Sánchez

Junio 2019

Universitat d'Alacant
Universidad de Alicante

Índice

Índice de figuras

1. Introducción

La sociedad actual cada vez está más conectada. La informatización de lo cotidiano está llegando a cada rincón de nuestra vida diaria, por lo que cada día aparecen nuevas tecnologías o nuevos servicios que hacen nuestro día a día más sencillo.

Las personas cada vez están más concienciadas con la privacidad de sus datos a la hora de utilizar servicios en el gran mundo de Internet. ¿Pero qué ocurre con la propia seguridad de los servicios o la manera en la que estos manejan los datos de sus usuarios?

Para poder hacer uso de un servicio personal a través de Internet primero debes de tener una manera de autenticarte contra el servicio que vayas a utilizar, es decir, el servicio debe de verificar que el usuario es quién dice ser. En este punto pueden ocurrir dos cosas, la autenticación se realiza a través de un servicio de terceros, o bien, que el propio servicio se encargue de la autenticación de cada persona.

Todo parece fácil si la autenticación la dejamos en manos de otro servicio, en este caso los más comunes suelen ser Facebook o Google. En el caso de Facebook, podemos encontrar multitud de noticias en las que podemos observar lo mal que ha tratado esta compañía los datos de los usuarios, sin ir más lejos el pasado mes de marzo del 2019 Facebook reconoció que llego a almacenar hasta 600 millones de contraseñas en texto plano, es decir sin cifrar, véase [1].

En el caso de Google no podemos encontrar noticias de filtraciones de contraseñas o un mal tratamiento de datos del usuario. Lo que sí podemos leer son bastantes noticias de cómo la gente está preocupada por todos los datos que Google almacena sobre ellos. Aunque Google haga un buen tratamiento de los datos y de la privacidad para sus usuarios, esto no quita que sea perjudicial por la cantidad de datos personales que almacenan de cada usuario por lo que, tampoco debería ser una buena opción para tener en cuenta a la hora de hacer la autenticación en un servicio.

La autenticación de un usuario se puede hacer de varias maneras, entre las que encontramos autenticación basada en algo que se conoce, normalmente una contraseña, autenticación basada en algo que se posee, como por ejemplo un smartphone, autenticación basada en como eres, esto quiere decir en la forma en la que se ha desarrollado el cuerpo del usuario, también llamada autenticación biométrica, por ejemplo, la huella dactilar y por último también puede haber combinaciones de varios métodos de los nombrados. Podemos

encontrar muchos más métodos de autenticación, pero he nombrado a los más utilizados por la comodidad del usuario y/o seguridad que proporcionan.

Las compañías que proporcionan servicios de autenticación, como las que hemos visto previamente arriba, suelen tener su método de autenticación basado en el conocimiento de una contraseña, este método no quiere decir que sea el peor, pero si lo será si el usuario crea una contraseña muy fácil, usando palabras que podamos encontrar en diccionarios, de una longitud muy corta o utilizando la misma contraseña que ya utilizó en otro servicio.

Como todos los métodos tiene una tasa de error, algunas compañías de terceros proporcionan un segundo factor de autenticación, normalmente este segundo factor se basa en un código de verificación enviado por SMS.

Este segundo factor de autenticación no es suficiente, ya que también ha sido probado que es fácil de evitar con ingeniería social. La ingeniería social es la técnica mediante la cual un individuo manipula usuarios legítimos para obtener información confidencial.

Con el avance de la tecnología ahora podemos encontrar mejores soluciones para implantar factores de autenticación, que son más rápidos y seguros. Una buena manera de realizar el segundo factor de autentificación sería de manera biométrica ya que según el método biométrico que utilicemos tendremos muy poca tasa de fallos en la autenticación. La única pega que hay para poder utilizar estos métodos, es el precio, ya que cuanto mayor precisión, mayor es su precio. Pero con el abaratamiento de la tecnología, cada vez encontramos más dispositivos que disponen de sensores de autenticación biométrica, como por ejemplo lectores de huellas, reconocimiento de iris, reconocimiento facial en 3D…

La opción que nos queda es que sea el propio desarrollador el encargado de realizar la autenticación del usuario y el tratamiento de sus datos para dicho fin. El problema que se plantea en este punto es que el servicio puede que no lo haga de una manera adecuada, debido mayoritariamente a la falta de recursos, herramientas o tiempo durante el desarrollo, de lo que deducimos que tampoco es una buena opción para la autenticación de usuarios, además que el servicio también deberá de gestionar sus propios métodos de segundos factores de autenticación. Ya ha habido muchos casos de empresas, en las cuales no se implementa la seguridad necesaria en sus servicios.

Llegados a este punto el problema que se nos plantea es, ¿cuál sería un buen sistema de autenticación de usuarios en los diferentes servicios que utilizamos?

1.1 Justificación del proyecto

Como hemos podido observar, podemos encontrar un problema en el método de autenticación de usuarios en servicios. Este será el problema a tratar en el proyecto.

En este proyecto se va a diseñar y desarrollar un sistema de autenticación de usuarios para cualquier servicio con segundo factor de autenticación biométrica, para garantizar la privacidad de los datos del usuario mediante una buena gestión de estos en el sistema, de manera que la autenticación sea segura, rápida y personalizable para cualquier tipo de servicio o tecnologías, con el fin de cumplir los requisitos de seguridad de cada servicio.

El sistema de autenticación será una API REST construida mediante el lenguaje de programación Go. Este lenguaje nos proporciona una gran flexibilidad, ya que está disponible para los sistemas operativos Windows, GNU/Linux, FreeBSD y Mac OS X, pero Go va más allá y puede ser instalado en otros sistemas operativos con el código fuente. Otra de las razones por las que nos brinda flexibilidad es porque está disponible para múltiples arquitecturas.

Pero sin duda, lo mejor de este lenguaje de programación, y la razón por el que ha sido escogido para este proyecto es su librería estándar, que son incorporadas al proyecto en forma de paquetes, añadiendo de esta manera funcionalidades extra al lenguaje. La librería criptográfica de Go, es una de las mejores librerías criptográficas de todos los lenguajes de programación que se puede encontrar ahora mismo en mercado. Está actualizada para poder ofrecer los últimos métodos criptográficos que utilizaremos en el desarrollo de este proyecto.

Para poder conseguir un buen sistema de autenticación de usuarios, se utilizarán algoritmos de resumen *hash* más *sal* (una sal en un algoritmo criptográfico hace referencia una secuencia de bits aleatoria que se suele usar en combinación con la clave) para todos los datos importantes que se almacenan en la base de datos. De esta manera conseguiremos no almacenar los datos en texto plano en nuestra base de datos, sino que lo que almacenaremos será un resumen de los datos, para que realicemos posteriormente la comparando los bytes del resultado de la función *hash*, con el resultado que tenemos almacenado en la base de datos.

Como el sistema de autenticación será una API REST, que para el proyecto está corriendo en un servidor en Internet, la comunicación entre el cliente y el servidor tiene que ser una comunicación cifrada, para ello, utilizaremos el protocolo TLS, que proporciona

comunicaciones seguras en una red. De esta forma las peticiones a la API con los datos del usuario serán cifradas y no será visibles en el caso de que alguien consiga interceptar la comunicación entre cliente y servidor.

Además de todo lo nombrado antes, también se añadirá un segundo factor de autenticación basado en biometría. Para poder hacer uso de factores biométricos, debemos de poseer el hardware y el software necesario para poder realizar la autenticación usuarios, por suerte para nosotros, hoy en día es muy fácil de encontrar mecanismos de autenticación biométricos en los smartphones o dispositivos inteligentes. Para ello utilizaremos todos los recursos biométricos, que nos ofrezcan los dos sistemas operativos para dispositivos inteligentes más utilizados en el mundo, Android e iOS.

Lo último que nos queda por ver, es el término personalizable, del cual se habla en el apartado anterior. Este término hace referencia al apartado comercial del servicio de este proyecto, ya que cada cliente que use la aplicación será capaz de personalizar los criterios de seguridad que debe cumplir cada usuario de su servicio, como por ejemplo la longitud de contraseña del usuario, estándares y condiciones que deben cumplir las contraseñas o que segundos factores de autenticación quiere que soporte su servicio.

2. Estado del arte

En los tiempos que corren hoy en día, cada vez existen más empresas y *startups* que comienzan o centran sus servicios en Internet. Normalmente, cuando estas empresas empiezan o empresas con más años de trayectoria, quieren dar el paso a dar su servicio en red, lo más aconsejable y la tendencia actual es usar IaaS (Infrastructure as a service), Paas (Platform as a service) y SaaS (Software as a service).

Junto con a estos términos de computación en la nube surge la idea de AaaS (Authorization as a Service). Las APIs desarrolladas en este proyecto seguirán un poco la idea de este término, en el cual podamos usar nuestras APIs de autenticación desplegadas en la nube con todos los veneficios que esto nos aporta.

A continuación, en este apartado, se explicarán estándares que vamos a utilizar a lo largo del desarrollo de este proyecto. Junto a estos estándares, se definirán que tipo de medidas de seguridad son necesarias para que estos estándares se puedan llevar a cabo.

2.1 Cifrado Web

En este apartado se explicarán que estándares han sido utilizados para poder proporcionar un buen cifrado en cuanto a web o se refiere.

Para comenzar, con el cifrado web, tenemos que comenzar hablando del protocolo TLS (Transport Layer Security y en español Seguridad de la capa de transporte). Este protocolo es la versión renovada de su antecesor SSL (Secure Sockets Layer y en español Capa de Puertos Seguros), ambos son protocolos criptográficos que se encargan de proporcionar comunicaciones seguras por una red (Internet). [2]

Esta comunicación segura es una comunicación cifrada, para poder realizar este cifrado se hace uso de certificados X.509 y por lo tanto de criptografía asimétrica de modo que el cliente y el servidor negocian el algoritmo de cifrado y las claves que posteriormente se usarán para cifrar el flujo de datos de ambas partes. De esta manera podemos garantizar la confidencialidad e integridad del mensaje y podemos hablar entonces de una comunicación privada.

TLS por lo tanto tiene parámetros de configuración, con los que podemos jugar para conseguir unas condiciones de seguridad y privacidad diferentes.

Este protocolo, tiene una propiedad importante en los sistemas criptográficos, conocida como *forward secrecy*. Esta propiedad, dicta que, conociendo una clave antigua o actual, no puedas saber cuáles van a ser las que se generen el futuro. De esta manera se dice que un sistema con esta propiedad es un sistema *forward-secure* (sistema seguro-adelante).

El protocolo ha sufrido revisiones por diferentes tipos de amenazas a la seguridad a este. Es por eso por lo que la versión actual del protocolo es TLS 1.3.

Esta versión fue definida en agosto del 2018 y está basada en su versión anterior TLS 1.2 definida en 2008. Todos los principales sitios web utilizan TLS para proteger la comunicación entre sus servidores y el navegador web del cliente.

Junto a este protocolo de cifrado web, encontramos otro protocolo que hace uso de TLS para mantener nuestras comunicaciones seguras, el protocolo del que estoy hablando es HTTPS.

HTTPS (Hypertext Transfer Protocol Secure) es un protocolo de aplicación basado en HTTP, cuyo fin es el mismo, pero haciendo una transferencia segura de hipertexto, es la versión con seguridad de HTTP al usar TLS para crear un canal cifrado.

HTTPS no llego para cambiar todo lo ya establecido con el protocolo HTTP, si no que vino a introducir nuevas mejoras y una de ella en cuanto a seguridad del protocolo se refiere.

Usando este protocolo, logramos que, si estamos transfiriendo información sensible a través de la red y alguien logra interceptar nuestra transferencia, no obtendrá la información en texto plano, si no, que lo que obtendrá será unos datos cifrados.

Para poder usar HTTPS en tu servidor web, el administrador de este, debe de crear un certificado de clave publica para este. Para que este certificado sea válido, debe de estar firmado por una autoridad certificadora, para asegurar que el poseedor del certificado es quién dice ser. Como esta última acción que estamos comentando cuesta dinero y este protocolo salió en el 2015, la adopción de este protocolo está siendo lenta.

2.2 OAuth

Es un estándar abierto para la delegación de acceso seguro [3]. Normalmente utilizado para poder compartir información entre servicios de manera que un usuario en un servicio A pueda acceder a su información en el servicio B sin necesidad de tener que compartir sus credenciales con dicho servicio, para que de esta manera el servicio B no pueda acceder al

usuario y una contraseña del usuario, pero si tener la certeza de que el servicio A ha autorizado al usuario y este es quién dice ser.

Este protocolo de autenticación es comúnmente usado en aplicaciones web, aplicaciones móviles o incluso aplicaciones de escritorio. Este mecanismo también es a su vez usado por todas las grandes compañías de servicios en Internet, como por ejemplo Amazon, Google, Facebook, Microsoft y Twitter entre muchas más.

OAuth ha sido diseñado para funcionar con el protocolo HTTP, utilizando los denominados "Request Token" y "Access Token". De esta manera un servidor de autorización comprueba y aprueba a un usuario y le devuelve a la aplicación de terceros un "Access Token".

En el momento que se redacta este documento, el protocolo OAuth está en la versión 2.0, que salió en octubre del 2012 para sustituir por completo a la versión 1.0 de este protocolo.

Decimos que para sustituir por completo porque fue un protocolo creado desde cero, es decir la versión 1.0 y la 2.0 son completamente diferentes y por lo tanto no son compatible entre dichas versiones.

La versión 2.0 del protocolo llegó para cubrir las carencias de la versión 1.0. Los rasgos más significantes de la versión 2.0 frente a la 1.0 son:

- Un mejor flujo para las aplicaciones que no están basadas en navegadores, al igual que añadir nuevos métodos de flujo, denominados *flows* en OAuth. Esto quiere decir que se ha creado un nuevo flujo para aplicaciones móvil y de escritorio, donde con OAuth 1.0, si, por ejemplo, tu aplicación era de escritorio, para implementar OAuth, obligaba al usuario a abrir una ventana del navegador, autenticarse en el servicio correspondiente, que este servicio le dirá un "Access Token", el cual, el usuario debía de pegar en algún lugar de la aplicación de escritorio. Con la versión 2.0, esto se ha corregido, de manera que es mucho más sencillo ahora para las aplicaciones no basada en navegadores.
- La versión 2.0, no requiere que la aplicación de cliente deba tener criptografía. Esto es así debido a que muchos de los casos en los que el OAuth era implementado de manera errónea en la versión 1.0, era debido a la complejidad de los requisitos criptográficos de la especificación y todo eso causaba que su uso no fuera trivial.
- Las firmas del protocolo son menos complicadas en la versión 2.0, con lo que se gana velocidad en el escalado.

- Y, por último, uno de sus cambios más importantes, es sin duda el tiempo de vida de los "Access Token". En la versión 1.0 del protocolo, los tokens podían ser almacenados incluso para un año y en muchos casos mucho más tiempo, por poner un ejemplo la API de Twitter hacia tokens que no expiraban en el tiempo. OAuth 2.0 introduce el concepto de refrescar el token. Con la versión 2.0, los tokens tienen un tiempo de vida más corto y se pueden ir refrescando en función de si hace falta o no, de esta manera, si no sé usa la aplicación el token caducará y no se podrá seguir interactuando con ella, por lo que tendremos que volver a solicitar un "Access Token".

2.2.1 Ventajas de usar OAuth

- Sencillez: es más sencillo para el usuario el poder usar sus credenciales de un servicio en otro, sin tener que hacer otro registro ni tener que dar información personal a otros servicios de terceros.
- Tiempo: en este caso ahora tiempo a los desarrolladores de servicio, ya que, al delegar la autenticación de usuarios a un servicio, pueden centrarse más en el desarrollo de su propio servicio. También ahorra tiempo al usuario final, ya que si ya tiene su cuenta creada podrá usarla en otro servicio. No solo en ese caso la mayoría de las veces OAuth te permite intercambiar información entre servicios.
- Privacidad: tus credenciales no están expuestas directamente sobre el servicio al que estás intentando acceder.
- Seguridad: El estándar OAuth solo poder ser utilizado bajo TLS, por lo que aseguras que el intercambio de información este cifrado frente a una captura de paquetes.

2.3 JSON Web Token

El estándar abierto JSON Web Token, también conocido por JWT, es un estándar basado en JSON para la creación de los "Access Token", con los que se permite propagar a otros servicios o sistemas información, como la identidad y los privilegios de un usuario, como si fuera un objecto JSON.

Los tokens son generados en el servidor, para poder generar un JWT, lo primero que debemos saber es que normalmente estos tienes tres partes.

La primera parte se le suele llamar *header*, en ella se alberga el algoritmo que se ha usado para generar la firma del token.

La segunda parte se suele denominar *payload*, en ella se suele poner toda la información referente a la identificación o de los privilegios en inglés llamados *claims*, por ejemplo, en esta parte es en la que indicaremos si es un usuario es administrador o cual es el ID identificativo del usuario, por lo general, también se suene indicar la fecha en la que fue creado el token, así como la fecha de expiración del mismo.

Por último, la tercera parte y la más importante, es donde está la firma, esta firma está compuesta por las demás partes, es decir el *header* y el *payload*, ambas partes codificadas en *base64*, todo ello cifrado con una clave secreta que estará almacenada en el servidor. El JWT puede ser firmado usando un secreto con un algoritmo HMAC o con una pareja de clave pública y privada usando RSA y ECDSA.

En los JWT que son firmados, se puede verificar la integridad de los *claims* contenidos en el token. Mientras que los JWT que son encriptados ocultan los *claims* para que los servicios de terceros no puedan verlos.

La finalidad que firmar un JWT con una pareja de clave pública y privada, es el poder demostrar que solo la compañía que posee la clave privada ha sido capaz de firmar el token.

La forma en la que funciona JWT, es sencilla, pongamos por ejemplo que un usuario quiere acceder a un servicio. Este usuario, ingresará su usuario y contraseña del servicio, por poner un ejemplo de autenticación, aunque es válido cualquier otro método, no tiene por qué ser autenticación por contraseña. Al ingresar sus credenciales con éxito, el servidor del servicio generará un "Access Token" del tipo JWT con todos sus privilegio e identificación y se envía al cliente en forma de respuesta. En este punto el usuario debe de almacenar el JWT de manera segura y siempre que quiera acceder a un recurso del servicio, deberá de enviar en la cabecera de su petición el JWT para que el usuario lo identifique y compruebe si tiene permisos para acceder a ese recurso.

Este esquema propuesto, es un sistema de autenticación *stateless*, es decir sin estado, ya que la sesión de usuario, no se guarda en el servicio o servidor del servicio.

2.3.1 Ventajas de usar JWT

Una gran ventaja de los JWT es tu tamaño, al tener un tamaño reducido pueden ir presentes en la cabecera de las peticiones HTTP. El JWT tiene que ser enviado obligatoriamente en cada petición, pero no tiene por qué ser en la cabecera de la petición. Aunque no es obligatorio, si recomendable seguir el esquema *Bearer*, en el cual se envía el JWT en la cabecera de la petición en el encabezamiento *Authorithation*.

Su pequeño tamaño también nos lleva a decir que nos proporciona un mejor rendimiento en la red.

Si comparamos los JWT token con los SWT (Simple Web Token), estos últimos solo pueden ser firmados con un algoritmo HMAC, sin embargo, como hemos visto antes, los JWT pueden ser firmados también con un par de clave pública y privada, lo que nos otorga más seguridad. Y si comparamos con los SAML (Security Assertion Markup Lenguage Tokens), observaremos que el proceso de firma de los JWT es mucho más sencillo que el "XML Digital Signature".

Otra de las ventajas es que ahora el servidor que hace la autenticación delega el almacenamiento de la sesión en el cliente, por lo que se libra de una pesada carga de tener que almacenar sesiones para todos los usuarios, con lo que se puede llegar a conectar muchos más microservicios usando el mismo token. Con todo lo nombrado en este párrafo podemos decir que, gracias a los JWT, logramos un servidor *stateless* es decir sin estado, logrando así que un mismo JWT pueda ser usado en diferentes *backends*.

2.4 Argon2

Argon2, es una función de derivación de clave ganadora de la competición *Password Hashing Competition* en julio del 2015. [4]

La competición *Password Hasing Competition*, es una competición abierta anunciada en 2013 para poder encontrar y seleccionar funciones de derivación de claves que pudieran ser reconocidas como estándares. Fue organizada por criptógrafos y facultativos de la seguridad informática. [5]

Una función de derivación de clave (KDF en inglés) deriva una o más claves a partir de un secreto, por ejemplo, el secreto puede ser una contraseña, todo ello usando una función pseudoaleatoria. Un uso que se les suele dar a las funciones de derivación de clave suele ser

la extender la contraseña, así como para darles formato. Un ejemplo a la hora de dar formato es la de poder usar la salida de una KDF como entrada para una clave en un cifrador AES. El uso más común, el que nosotros le daremos, que es para la verificación de contraseñas.

En la derivación de clave, hay que tener en cuenta más factores, en el caso de Argon2, las otras variables son:

- Memoria: la cantidad de memoria que va a usar el algoritmo, en kibibytes.
- Iteraciones: el número de iteraciones sobre la memoria.
- Threads o hilos: número de hilos en la paralelización del algoritmo.
- Tamaño de *sal*: tamaño para la *sal* (sal criptográfica).
- Tamaño de clave: tamaño para la clave generada, es decir tamaño de salida del *hash*.

Todos estos parámetros, son definidos con la intención de prevenir un ataque de fuerza bruta. La dificultad de un ataque por fuerza bruta aumenta con el número de iteraciones. En resumen, Argon2 trata de crear un llenado de memoria usando varias unidades de computo, explotando así mejor la memoria cache.

Como hemos podido observar, es una función de derivación de claves algo joven. Su licencia de uso es Apache 2.0 y podemos encontrar tres versiones del algoritmo Argon2 [6]:

- Argon2d: donde se maximiza la resistencia contra ataques por GPUs al acceder al array de memoria en un orden dependiente, pero introduce posibles ataques por canal lateral (Side-Channel Attack).
- Argon2i: maximiza la resistencia contra ataques por canal lateral, pero introduce posibles ataques por GPUs ya que se accede al array de memoria de manera independiente.
- Argon2id: es una función hibrida entra las dos versiones de arriba. La primera fase del algoritmo se hace con la aproximación de Argon2i y el resto de algoritmo con la aproximación de la versión Argon2d.

En la documentación del algoritmo se recomienda el uso de la versión Argon2id, siempre y cuando haya razones de peso para usar las otras versiones.

Aquí entra el factor de coste, el cual debemos de tener en cuenta, puesto que ha mayor coste para derivar la clave, mayor será el tiempo necesario para hacerlo, por lo que, si vamos a utilizarlo para la autenticación de usuarios, debemos que hacerlo dentro de unos parámetros

que nos permitan hacer las operaciones en un tiempo razonable y que no generé una mala experiencia de usuario.

Argon2 es muy recomendable cuando se requiere un alto rendimiento en la aplicación que va a hacer uso de él, ya que puede llenar grandes cantidades de memoria RAM en muy poco tiempo y haciendo uso de grandes cantidades de uso computacional paralelo.

2.4.1 Ventajas de Argon2

Para poder hablar sobre las ventajas de Argon2, debemos comparar esta función de derivación de claves con las demás funciones de derivación de claves que podemos encontrar en el mercado a fecha de estar escribiendo este documento, véase [7].

Sin duda, las funciones de derivación de clave más conocidas y de las más usadas son Bcrypt y Pbkdf2.

Como información adicional, la función de Bcrypt está basada en la función de cifrado Blowfish, la cual fue lanzada en 1999, en la que se incorporaba una *sal* para prevenir de ataques por *rainbow tables*. Hasta la fecha no tiene vulnerabilidades conocidas, véase [8].

Pbkdf2 fue lanzada en el 2000 y cuenta con un coste computacional variable basado en el número de rondas que hace el algoritmo, para poder defenderse de ataques por fuerza bruta, además, también incorpora *sal* para evitar ataques basado en *rainbow tables*. Hasta la fecha tampoco cuenta con vulnerabilidades conocidas, véase [9].

Ahora que conocemos un poco las funciones de derivación de claves más usadas y conocidas, podemos compararlas con Argon2.

Argon2 es una función de derivación de claves reciente (2015), dado que las otras funciones fueron lanzadas quince años antes. Esta es una de sus ventajas, ya que ha podido formarse contra los ataque más usados y conocidos en esos quince años de diferencia.

Otra de las ventajas de Argon2, es su diseño, el cual ha sido creado para ser un algoritmo capaz de usar mucha memoria. El uso alto de memoria hace que los ataques por fuerza bruta usando GPUs sean mucho más costosos, ya que no podrán beneficiarse del ahorro de tiempo en la paralelización del ataque con GPUs.

2.5 Función hash criptográfica

Se llaman funciones *hash* criptográficas a aquellas funciones *hash* que se utilizan en el área de la criptografía, véase [10].

Estas funciones son muy útiles en el ámbito de la criptografía, debido a que cumplen unas propiedades que las hacen muy resistentes contra ataques maliciosos.

Estas propiedades son:

- Deterministas: Para un conjunto de datos dados siempre se obtendrá el mismo valor de *hash*.
- Computación eficiente: se puede lograr realizar el proceso de *hash*, de una manera rápida y eficiente en cualquier sistema.
- No reversible: No se hay, o no debería de haber, una manera directa de obtener el conjunto de datos, dado su *hash*.
- La alteración de un bit: La más pequeña alteración producida a los datos, aunque solo sea un bit, implicará grandes cambios en el código del *hash* obtenido.
- Resistente a colisiones: esto quiere decir que la probabilidad de encontrar dos conjuntos de datos que tenga el mismo código *hash* es altamente improbable.

Dadas estás propiedades, hacen que los códigos *hash* sean perfecto para usar en este desarrollo de la mano de la función de derivación de claves Argon2.

3. Planificación del proyecto

En el apartado de planificación del proyecto se explicará que metodología de desarrollo se ha empleado para desarrollar y planificar este proyecto, así como el entorno junto con las tecnologías elegidas para el desarrollo.
También en este apartado se mostrará el *roadmap* del proyecto junto con la descomposición del trabajo y el tiempo asignado a cada fase del desarrollo.
Por último, en este apartado mostraremos los posibles modelos de negocio que tiene este proyecto pese a que este sea un proyecto de código abierto.

3.1 Metodología de desarrollo

Una metodología de desarrollo es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo software.

En la actualidad podemos encontrar muchas metodologías de desarrollo y cada una de ellas tiene más o menos su propio enfoque y modelo, a su vez, cada modelo cuenta con sus fortalezas y sus debilidades.

Para este proyecto y dado que este proyecto va a ser desarrollado por una única persona el modelo elegido ha sido el iterativo e incremental. Este modelo de desarrollo me permite poder dividir el desarrollo en iteraciones de tiempo, donde con cada iteración conseguimos un producto final más complejo.

En cada iteración lo que se busca es evolucionar el producto apoyándose en las iteraciones anteriores, de manera que cada iteración retroalimenta la siguiente para mejorar y ampliar las funcionalidades del producto. Para poder llevar un proyecto con este modelo de desarrollo la clave es priorizar los objetivos en función del valor que aportan estos al producto final y los que realizan funcionamientos esenciales.

Este modelo ha sido seleccionado ya que las partes esenciales se desarrollan primero y en las siguientes iteraciones, se perfilan y se desarrollan las demás características del producto. De esta manera es más sencillo para una persona encontrar fallos en fases tempranas. Aunque se pueden encontrar iteraciones complicadas, los productos desarrollados con este modelo no suelen fallar ya que las partes esenciales se han probado mucho y en muchas fases, a todo esto, sumamos en que cada fase aprendemos más, con lo que podemos mejorar aspectos que en fases tempranas no teníamos conocimiento.

Como se desarrollarán varios softwares diferentes, habrá iteraciones de trabajando donde nos centremos en el desarrollo de solo uno de ellos, mientras que habrá otras fases en las que se desarrollarán y perfilarán varios de estos softwares a la vez.

Debido a la distancia que hay entre mi casa y la universidad, se ha pactado con el tutor, reuniones presenciales mensuales para poder definir bien los requerimientos, resolver dudas, posibles mejoras e ir determinando el alcance del proyecto, ya que como veremos más adelante, se puede ampliar mucho.

Además de las reuniones presenciales también se contará con un sistema de comunicación basado en el correo electrónico con los mismos objetivos mencionados en el párrafo anterior, resolver dudas, posibles mejoras, etc.

Con todo este procedimiento, creo poder tener una versión más que suficiente para poder realizar una versión básica del producto. Me refiero como versión básica, ya que como he mencionado, este producto se puede ampliar mucho hasta alcanzar lo que se consideraría un producto final.

3.2 Entorno y tecnologías de desarrollo.

En este proyecto, se van a desarrollar varios softwares, el primero y principal, será dos APIs REST en un servidor en el lenguaje de programación Go. Como segundo elemento a desarrollar, tenemos dos aplicaciones para Android, que se utilizarán para la administración de la autenticación, tanto de usuarios, como de clientes, estas aplicaciones serán desarrolladas con el lenguaje de programación C# utilizando el IDE Xamarin, junto con estas aplicaciones y para poder ofrecer una demo del producto más enriquecida, también se desarrollarán páginas web utilizando también el lenguaje C# usando ASP .Net.

3.2.1 API (Application Programming Interface)

Para este proyecto vamos a desarrollar dos APIs. Una API es un conjunto de subrutinas, funciones y procedimientos que ofrece una librería de software para que estas puedan ser utilizadas por otro software.

En el caso del proyecto nuestras APIs, vas a ser del tipo API REST. REST es un estilo que hace referencia a un conjunto de principios de arquitectura en la creación de aplicaciones.

Esos principios con:

- Un protocolo cliente / servidor sin estado. Todos los mensajes HTTP enviados tienen toda la información necesaria para una correcta petición. De esta manera ni el cliente, ni el servidor deben de almacenar un estado o sesión del usuario.
- Operaciones bien definidas en lo que ha operaciones HTTP se refiere. Esto quiere decir que utiliza y hace un buen uso de las operaciones de HTTP, como pueden ser POST, GET, PUT o DELET.
- Sintaxis universal. Cada recurso es accesible desde su URI única.
- Formato XML para cada mensaje y para cada uno de los recursos.

Aunque, en la actualidad, se utiliza REST en un sentido más amplio para indicar cualquier interfaz entre sistemas que utilice directamente HTTP, para obtener datos o para ejecutar operaciones sobe datos, por ejemplo, JSON, todo ello sin protocolos para el intercambio de mensajes.

3.2.2 Go

Go es un lenguaje de programación desarrollado por Google que salió al mercado en 2009. Es un lenguaje de programación concurrente, compilado y orientado a objetos, con una sintaxis muy parecida a la sintaxis de C, esto es debido a que en el desarrollo de Go está presente Ken Thompson el cual, al margen de crear UNIX, también fue el creador del lenguaje B un precursor del lenguaje C, esto hace que sus sintaxis tengan algo de parecido, sobre todo al principio, ya que con cada versión noto que se separa un poco más del lenguaje C, véase [11] [12].

Go además está disponible para todos los sistemas operativos más utilizados en el mundo, como por ejemplo Windows o GNU/Linux. Por si esto no fuera poco, también contamos con este lenguaje en formato binario, por lo que podríamos instalarlo en cualquier sistema operativo que quisiéramos.

Go usa un tipado estático con lo cual le permite una mayor eficiencia en tiempo de ejecución, muy similar a otros lenguajes como JAVA o C++. Además, cuenta con muchas de las características, como la legibilidad, propias de los lenguajes dinámicos como Python o JavaScript. Por último, a todo esto, se le suma un alto rendimiento en la red y multiprocesos.

También está preparado para el paradigma de programación orientada a objetos, pero en este lenguaje no disponemos de herencias y tampoco dispone de excepciones, pero sí contamos con recolector de basura. Se pueden crear clases, pero su uso y creación son algo confusos al igual que la delegación y el polimorfismo.

Gracias a su sencillez, Go tiene una gran comunidad que lo resguarda, pues es un lenguaje mucho menos pesado y más cómodo para los programadores, como por el ejemplo su inferencia implícita de tipos, lo cual hace que crear variables se mucho más rápido y sencillo. Otra de sus grandes ventajas es su librería estándar la cual cuenta con números paquetes para abarcar la mayoría de los algoritmos y funciones que podamos necesitar para el desarrollo de un software.

Quiero destacar de esta librería estándar de Go, el paquete "crypto*"* el cual es el que nos permitirá usar los algoritmos y estándares criptográficos usados en este proyecto. Este paquete es uno de los más completos en cuanto a criptográfica se refiere, que podemos encontrar en estos momentos en los lenguajes de programación actuales.

Para poder comenzar a programar en Go, en mi caso lo voy a instalar en Windows 10. En el proceso de instalación, debes de definir dos variables de entorno, la *GOROOT* y la *GOPATH*. La variable de entorno *GOROOT* se utiliza en el proceso de instalación de Go, ya que Go asume que la ruta de instalación será "usr/local/go" o en "C:\go" en el caso de Windows, gracias a *GOROOT*, podremos instalar Go y todas sus herramientas en una localización diferente.

Con la variable de entorno de *GOPATH*, indicaremos la localización donde Go buscará código escrito en Go. En esta localización es donde se añadirán y buscarán los paquetes que no pertenezcan a la librería estándar de Go, así como el código que nosotros desarrollemos.

Una vez tengamos Go instalado en nuestro ordenador tendremos el entorno listo para poder comenzar a desarrollar nuestras APIs en Go. Para ello la herramienta que utilizaremos, será Visual Studio Code junto al plugin "Go" oficial de Microsoft para este editor de código.

Visual Studio Code es un editor de código de creado por Microsoft de código abierto. Visual Studio Code, nos permitirá gracias al plugin, instalar paquetes para Go, tanto oficiales como de terceros, autocompletado y coloreado de sintaxis, así como desarrollar, ejecutar y depurar aplicaciones escritas en Go, aparte de ofrecernos control de versiones con Git.

3.2.3 GitHub

Ya que hemos hablado del control de versiones, vamos a indicar que en este proyecto se va a utilizar GitHub para almacenar todos los proyectos y desarrollos software utilizando el sistema de control de versiones Git.

Durante el desarrollo de este proyecto, todo se almacenará en GitHub de manera privada, gracias a la cuenta de estudiante que nos proporciona el sitio al ser estudiantes de la Universidad de Alicante. Una vez finalizado el desarrollo la idea en liberar el código bajo una licencia de código abierto, lo cual hará que el repositorio de este proyecto pase a ser público.

3.2.4 Xamarin

Xamarin es una compañía de desarrollo de software, que ofrece su producto "Xamarin" el cual es un *framework* totalmente *Open Source*, para crear aplicaciones nativas en las plataformas móviles más conocidas: Android, iOS y Windows Phone (este último ya descontinuado), compartiendo código C# y XAML.

De forma que todo el contenido visual de la aplicación la desarrollaremos en el lenguaje de formato visual XAML. XAML es un lenguaje declarativo basado en XML, para describir gráficamente interfaces de usuario.

Y toda la parte de *backend*, es decir toda la lógica de la aplicación, será escrita en código C#. Solo tendremos que escribir código especifico en cada plataforma, cuando necesitemos un servicio en concreto de la plataforma, ya que por ejemplo en nuestro caso, para este proyecto necesitaremos hacer uso del lector de huellas de nuestros dispositivos y el manejo de dicho lector hay que hacerlo en cada plataforma de manera individual. Para resumir, se comparte casi todo el código y es el *framework* el encargado de traducir el código a cada plataforma, pero hay ocasiones en las que tendremos que escribir un poco de código en cada plataforma, por lo que necesitaremos tener conocimientos básicos de cada una de ellas.

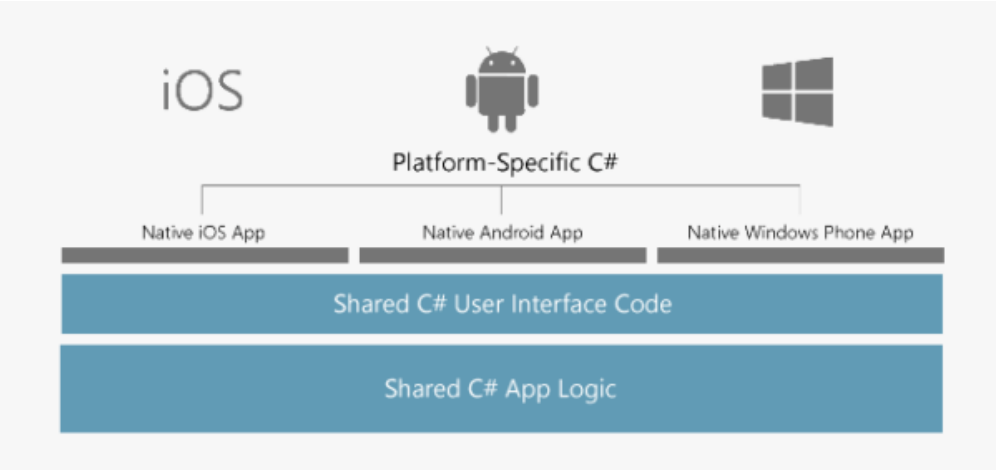

Figura 1 Flujo de trabajo del framework de Xamarin

Xamarin ya está trabajando en una librería la cual se llama "Xamarin Essentials", en la que unifican varios de estos servicios para poder reutilizar mucho más código y poco a poco tener que escribir menos código de cada plataforma. En este proyecto se usará la librería "Xamarin Essentials" en su primera versión estable.

Pero la pregunta que siempre surge en este punto es si realmente Xamarin vale la pena para programar aplicaciones y si realmente ahorra tiempo. La respuesta es sí en ambas peguntas, Xamarin te permite compartir entre un 75% y un 95% de código, lo que se traduce a un ahorro significativo de tiempo.

Por otro lado, el tema del rendimiento a la hora de crear aplicaciones con Xamarin, es muchas veces puesto en duda, pero esto no es verdad ya que este *framework* al traducir el código a aplicaciones nativas, conseguimos el mimo rendimiento que si hubiéramos programado cada aplicación a su plataforma destino. En el único punto que falla un poco, pero es normal si pensamos en la forma en la que trabaja, es el tamaño de la aplicación, donde nos podemos encontrar que las aplicaciones desarrolladas en Xamarin pueden llegar a pesar un 30% más que si lo hubiéramos desarrollado con un *framework* específico para cada plataforma.

El caso del proyecto de Xamarin es un poco raro, pues ha ido pasando por varios propietarios hasta que en 2016 Microsoft compro la compañía para intentar dar más protagonismo a Windows Phone, por lo cual esta es una compañía de Microsoft.

Pese a que Windows Phone no tuvo mucho éxito, no se desperdició todo el trabajo realizado para Windows Phone. Con la salida del sistema Windows 8, llegaron al sistema operativo más conocido del mundo las aplicaciones nativas, por aquel entonces conocidas como WinRT, con las siguientes versiones del sistema operativo se fue perfeccionando el sistema

de las aplicaciones. Ahora denominadas aplicaciones UWP (Universal Windows Platform), pueden ser instaladas en cualquier dispositivo que ejecute Windows 10, en cualquiera de sus versiones.

Gracias a todo el apoyo que Microsoft le ha dado a la compañía para que se puedan desarrollar aplicaciones de UWP en Xamarin, es posible programar en Xamarin utilizado el IDE Visual Studio, instalando todos los componentes de Xamarin.

Estoy muy como usando el IDE Visual Studio ya que trabajo mucho con este IDE. Otra cosa que me gusta es el poder utilizar tanto mi propio ordenador con Windows 10 como mi móvil Android para hacer las pruebas, aunque he de decir que podemos instalar un emulador de Android que funciona muy bien.

Como ya hemos visto, Xamarin nos permite compartir código entre proyectos, dentro de una misma solución de Visual Studio. Es por esto, que nos permite fabricar aplicaciones para los tres sistemas operativos vistos en este apartado, teniendo que escribir solamente una vez el código. El desarrollo de las aplicaciones tanto para los usuarios como para los clientes va a ser para el sistema operativo Android en una primera aproximación del proyecto y para cubrir los requisitos mínimos que podremos ver en el apartado de *RoadMap*. Para poder desarrollar una aplicación para un dispositivo con iOS, necesitaremos un ordenador con MacOS y un dispositivo con iOS, que son dos requisitos con los que no cuento a la hora de desarrollar este proyecto.

3.2.5 ASP .NET con .NET Framework

ASP .NET es un entorno para aplicaciones web desarrollado por Microsoft. Es comúnmente usado para crear sitios web dinámicos o aplicaciones web. Además, ASP .NET está construido sobre *Common Language Runtime*, permitiendo así escribir código ASP .NET usando cualquier lenguaje que este admitido por .NET Framework, es más si en lugar de usar .NET Framework, usáramos .NET Core podríamos ejecutarlo sobre un sistema operativo macOS o GNU/Linux, aparte de Windows.

Para el desarrollo del proyecto utilizaremos ASP .NET para hacer una página de cliente para el proyecto la cual haga uso de nuestra API y la consuma de manera Web.

La siguiente idea, también es poder registrarte como una empresa cliente a través de la página de modo que tu empresa aparezca en el sistema y que pueda decidir todos los parámetros de seguridad que tiene que cumplir la autentificación de sus usuarios.

El desarrollo de la web para el proyecto será sencillo. Utilizaremos Bootstrap para el estilo de la página, JQuery para hacer las peticiones de la API y por último HTML básico para definir un par de aspectos de la página.

3.2.6 Azure

Azure es un servicio en la nube creado y mantenido por Microsoft. Azure es una plataforma general que contiene diferentes servicios.

En el desarrollo de este proyecto, se usarán los servicios de Azure para contener la base de datos del proyecto, así como las APIs desarrolladas en Go corriendo en una máquina virtual de Azure.

Al igual que en GitHub, disponemos de una cuenta de estudiante, por ser alumnos de la Universidad de Alicante, con 75$ gratuitos para usar los servicios durante un año. Con los 75$ gratuitos tenemos más que suficiente para poder gestionar este proyecto a pequeña escala.

Contrataremos pues un servidor SQL Server, el cual contendrá todas nuestras bases de datos ya que durante el desarrollo de las APIs se usará SQL Server para almacenar todos los datos de los usuarios y clientes. En este caso nuestro servidor de SQL Server estará alojado en el Norte de Europa y contará con una capacidad de 2GB de almacenamiento en bases de datos. En este caso se pagará por almacenamiento utilizado.

Por otro lado, contrataremos un servicio de máquina virtual, la cual no hace falta que se sea muy potente ya que el software desarrollado en Go da mucho rendimiento incluso con bajas prestaciones. En nuestro caso la máquina virtual estará situada en el norte de Europa y tendrá el sistema operativo Ubuntu, 1 núcleo y 1.75 GB de memoria RAM junto a un disco de 30 GB HDD. Esta máquina virtual nos costará 15.69€ al mes en caso de que estuviera 24/7 encendida.

Hay varias razones por las que he decidido trabajar con un entorno *cloud*. La primera es debido a que pienso que, al ser un proyecto con una finalidad de seguridad informática, es más fácil mantener un entorno *cloud* seguro con todas las herramientas que este te proporciona, a mantener por uno mismo la seguridad de un de un servidor propio. Otra de las razones es la comodidad y la sencillez para comenzar a trabajar y la última razón es debido a poder trabajar desde cualquier parte en todo el apartado del servidor.

Por último, otra de las aproximaciones a la que quería llegar utilizando Azure, era al concepto de AaaS (Authorization as a Service).

La razón por la que he elegido el sistema de Azure es básicamente porque ya lo había utilizado antes y se me hace cómodo y sencillo todo el manejo de sus servicios.

Todas estas características seleccionadas son utilizadas para pruebas en el comienzo de desarrollo del proyecto de manera inicial, si el proyecto creciera o se pusiera en este caso en producción, tendríamos que ampliar las características de nuestro entorno *cloud* y por lo tanto el precio se incrementaría.

3.2.7 SQL Server

SQL Server es un sistema para gestionar bases de datos relacionales, desarrollado por Microsoft. Normalmente ha estado disponible siempre para sistemas operativos Windows, pero desde 2016 está disponible también para sistemas GNU/Linux y desde 2017 para Docker.

Aunque el lenguaje que usa SQL Server es T-SQL, este lenguaje es un derivado del SQL estándar, por lo que en el desarrollo de este proyecto no usaremos ninguna de las características de T-SQL salvo los *schemas* de los cuales hablaremos en el apartado de desarrollo. Para poder dejar el código lo más abierto posible a la idea de usar otros sistemas de gestión de bases de datos como por ejemplo MariaDB, también se describirán los procesos utilizados en el apartado de desarrollo.

Para poder conectar Go con SQL Server, utilizaremos el *driver* recomendado por Microsoft [13] [14] llamado “go-mssqldb” [15]. Para poder utilizar este *driver* necesitaremos al menos la versión 1.8 de Go, por lo demás, el uso de cadenas de conexión es muy parecido, por no decir igual, que al uso de otros drivers de bases de datos en otros lenguajes de programación.

Al comienzo del proyecto se comenzarán las pruebas en un servidor SQL Server Express instalado en mi maquina local. Cuando el proyecto haya pasado las primeras iteraciones se pasará esa misma base de datos a un servidor SQL Server alojado en Azure.

3.2.8 Postman

Postman es un software especial utilizado para el "API Testing", muy parecido al "Software Testing", este software nos permite poder realizar pruebas a APIs (Interfaz de programación de aplicaciones), de manera directa como parte de los *test* de integración para comprobar que la API cumple con requisitos en cuanto a funcionalidad, fiabilidad, rendimiento y seguridad. La API al carecer de interfaz, las pruebas deben de ser realizadas mediante la capa de intercambio de mensajes.

Postman es un software que actúa como un cliente HTTP, lo cual nos permitirá poder realizar pruebas para APIs. Postman al igual que nos deja crear test y lanzar consultas a nuestras APIs para poder probar de manera manual, también nos permite poder realizar una batería de pruebas automatizada.

Esta última opción, nos posibilita poder lanzar pruebas a la API con cada cambio que realicemos y comprobar que todo funcione correctamente. A todo este proceso lo acompaña el hecho de poder definir colecciones de llamadas, con lo cual podremos tener organizada toda la información de pruebas de nuestra API por cada ruta que tengamos definida en ella.

Por último, una gran ventaja que posee Postman, son las variables de entorno, las cuales nos dejaran definir variable para una colección a para un numero de pruebas de manera común a todos, por ejemplo las pruebas realizadas para este proyecto con Postman, tienen como variable de entorno, la IP del servidor al que se va a realizar la solicitud, de esta manera conseguimos poder tener dos variables de entorno, una apuntando a nuestra API en desarrollo y otra apuntando a nuestra API en producción, de manera que cambiando las variables activas, podemos realizar la misma prueba contra APIs diferentes.

Como ya he mencionado antes, la utilización de Postman en este proyecto a servirá para poder probar todas nuestras APIs de una manera sencilla cómoda y automatizada.

3.3 Descomposición de trabajo

A continuación, descompondré el trabajo en las iteraciones necesarias para poder alcanzar un producto final, en cual tengamos todos los elementos propuestos para este proyecto.

Para esta tarea usaremos una estructura jerárquica, para ello utilizaremos EDT (Estructura de descomposición del trabajo). Comúnmente utilizada para entregar información de manera

jerárquica y simple a un equipo de un proyecto, para poder cumplir con los objetivos de dicho proyecto.

En cada nivel, encontraremos una definición con detalle y cada nivel será una de las iteraciones que realizaremos en el proyecto. De esta forma podremos definir el alcance del proyecto según tengamos definido en el este documento.

1. Desarrollo de API de autenticación de usuarios
    1.1. Análisis
        1.1.1. Hacer reunión inicial
        1.1.2. Determinar el alcance del proyecto
        1.1.3. Determinar los requerimientos del proyecto
        1.1.4. Definir recursos
        1.1.5. Análisis completado
    1.2. Análisis de requerimiento software
        1.2.1. Determinar los requerimientos software
        1.2.2. Seleccionar herramientas de desarrollo
        1.2.3. Seleccionar tecnologías
        1.2.4. Seleccionar entorno de desarrollo
        1.2.5. Análisis de requerimientos software completado
    1.3. Diseño de la API usuario
        1.3.1. Revisar la especificación de software
        1.3.2. Revisar el estado del arte
        1.3.3. Diseñar los diagramas de secuencia
        1.3.4. Diseñar los diagramas de colaboración
        1.3.5. Diseñar los diagramas de clases
        1.3.6. Diseñar los diagramas de estados
        1.3.7. Diseñar los diagramas de actividad
        1.3.8. Diseñar los diagramas de E-R
        1.3.9. Implementar lógica de negocio
        1.3.10. Realizar pruebas unitarias
        1.3.11. Diseño de la API completado

1.4. Diseño de la API del cliente
    1.4.1. Revisar la especificación de software
    1.4.2. Revisar el estado del arte
    1.4.3. Diseñar los diagramas de secuencia
    1.4.4. Diseñar los diagramas de colaboración
    1.4.5. Diseñar los diagramas de clases
    1.4.6. Diseñar los diagramas de estados
    1.4.7. Diseñar los diagramas de actividad
    1.4.8. Diseñar los diagramas de E-R
    1.4.9. Implementar Base de Datos
    1.4.10. Implementar la lógica de negocio
    1.4.11. Realizar pruebas unitarias
    1.4.12. Diseño de la API del cliente completado
1.5. Diseño de la aplicación Android para usuarios
    1.5.1. Revisar la especificación de software
    1.5.2. Revisar el estado del arte
    1.5.3. Diseñar los diagramas de secuencia
    1.5.4. Diseñar los diagramas de colaboración
    1.5.5. Diseñar los diagramas de clases
    1.5.6. Diseñar los diagramas de estados
    1.5.7. Diseñar los diagramas de actividad
    1.5.8. Diseño de interfaz
    1.5.9. Implementar la interfaz de usuario
    1.5.10. Implementar lógica de negocio
    1.5.11. Realizar pruebas unitarias
    1.5.12. Diseño de la aplicación Android completada
1.6. Diseño de la aplicación Android para clientes
    1.6.1. Revisar la especificación de software
    1.6.2. Revisar el estado del arte
    1.6.3. Diseñar los diagramas de secuencia
    1.6.4. Diseñar los diagramas de colaboración
    1.6.5. Diseñar los diagramas de clases
    1.6.6. Diseñar los diagramas de estados
    1.6.7. Diseñar los diagramas de actividad

1.6.8. Diseño de interfaz

1.6.9. Implementar la interfaz de usuario

1.6.10. Implementar lógica de negocio

1.6.11. Realizar pruebas unitarias

1.6.12. Diseño de la aplicación Android completada

1.7. Diseño de Web para usuarios

1.7.1. Revisar la especificación de software

1.7.2. Revisar el estado del arte

1.7.3. Diseñar interfaz de usuario

1.7.4. Implementar interfaz de usuario

1.7.5. Implementar lógica de la página web

1.7.6. Realizar pruebas unitarias

1.7.7. Diseño de Web para usuarios terminado

3.4 Planificación del proyecto (Roadmap)

En este apartado se explicará mediante un diagrama de Gantt como se organiza temporalmente la descomposición de tareas del apartado anterior.

De esta manera, podremos saber cuánto tiempo tenemos que dedicarle a cada tarea a lo largo de un tiempo definido. También podremos saber cuál es la tarea que más tiempo necesita y si vamos bien de tiempo para siguiente iteración.

Nombre de tarea	Duración	Comienzo	Fin
Desarrollo para el servicio de autenticación de usuarios	**183 días**	**mar 11/09/18**	**jue 23/05/19**
Análisis	**5 días**	**mar 11/09/18**	**lun 17/09/18**
Análisis de requerimiento software	**5 días**	**mar 18/09/18**	**lun 24/09/18**
Diseño de la API usuario	**81 días**	**mar 25/09/18**	**mar 15/01/19**
Diseño de la API del cliente	**18 días**	**mié 16/01/19**	**vie 08/02/19**
Diseño de la aplicación Android para usuarios	**54 días**	**lun 11/02/19**	**jue 25/04/19**
Diseño de la aplicación Android para clientes	**15 días**	**vie 26/04/19**	**jue 16/05/19**
Diseño de Web para usuarios	**5 días**	**vie 17/05/19**	**jue 23/05/19**

Figura 2 Planificación de las fases del proyecto

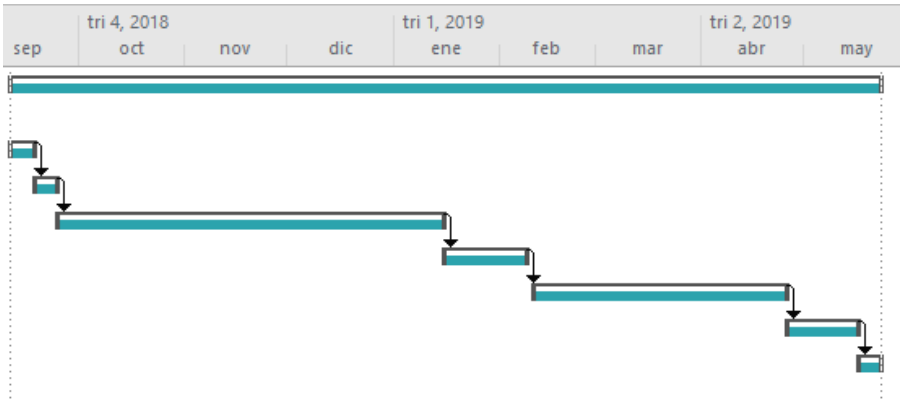

Figura 3 Diagrama de Gantt de las fases del proyecto

Como podemos observar, el proyecto comienza con la iteración de análisis el once de septiembre del 2018, esta iteración comienza con la reunión con el tutor.

Como podemos observar en el diagrama, tenemos todas las horas repartidas en el tiempo de modo que podemos ver de un simple vistazo que tareas de nuestro proyecto nos van a llevar más tiempo y por lo tanto más esfuerzo. También podemos llevar una vista del progreso que debemos hacer diariamente para poder alcanzar nuestra meta sin retrasos. Este diagrama también nos ayuda a restructurar el proyecto en caso de que retraso para poder alcanzar el final de la manera más eficiente posible, por lo que lo veo una buena herramienta para tener en cuenta antes de comenzar con el desarrollo del proyecto.

Debemos tener en cuenta que esta predicción es solo para una persona, es decir con un equipo de desarrollo podríamos hacer tareas en paralelo por lo que agilizaríamos el trabajo y reduciríamos el tiempo.

Tras todas las iteraciones el proyecto acaba el 23 de mayo del 2019, esto es una fecha aproximada, ya que pueden ocurrir imprevistos en el desarrollo que hagan que el proyecto se retrase o también puede ocurrir que se adelanten tareas y se acabe en un tiempo menor al previsto inicialmente, ya que pude ocurrir que hayamos calculado mal el tiempo para algunas de las tareas.

3.5 Modelo de negocio

Este proyecto fue pensado para ser un proyecto *Open Source*, con el que pequeñas empresas que están empezando en el sector puedan usarlo para afrontar el gran gasto que supone la gestión de la autenticación de usuarios.

Sin embargo, que sea un proyecto *Open Source* no quiere decir que no posea modelos de negocio.

Este proyecto tendrá una licencia de GNU (General Public License), también conocida por sus siglas en inglés GNU GPL o GLP para abreviar. Esta licencia es muy utilizada en el mundo del software libre, pues garantiza a los usuarios la libertad de estudiar, utilizar y modificar el software de la manera que ellos deseen. Pero esta licencia tiene un segundo añadido, por la cual implica proteger al software que contiene esta licencia contra un uso en cual se restrinjan las libertades nombradas anteriormente, cuando se distribuya o se modifique el software. De esta manera, todo el software derivado de este o en la distribución de este, debe llevar esta misma licencia para proteger las libertades del software.

3.5.1 Servicio privado

Junto a esta licencia nombrada en este apartado el modelo de negocio será el siguiente. Los clientes podrán descargar de manera libre el software a través de la plataforma GitHub, donde se almacenará el proyecto. Al hacer esto, los propios clientes, serán los encargados de configurar todo el servidor, API y bases de datos para poder hacer posible la gestión de usuarios, es decir se tendrán que hacer cargo de configurar todo el servicio. También serán los encargados de la seguridad de todo el entorno, puesto que, aunque este proyecto sea seguro en sí mismo, con una mala configuración y administración de sus elementos, puede que la seguridad de este se reduzca o incluso desaparezca.

Es decir, cualquier cliente puede usarlo, pero se hace responsable del propio uso que se le pueda dar a este software y es responsable del correcto funcionamiento de este.

Por otro lado, este proyecto estará montado y configurado de una manera correcta en un entorno *cloud* de forma privada, es decir se construirá de la misma manera en la que se ha redactado en este documento. Proporcionando a los clientes la posibilidad de pagar por la comodidad de solo tener que darse de alta para poder comenzar a autenticar usuarios en sus sistemas o servicios, sin necesidad de contar con su propio servidor o entorno *cloud* y sin preocupaciones de tener que configurar todo.

Además, el servicio privado contara con un servicio técnico especializado capaz de resolver cualquier problema que puedan tener con el servicio. De esta manera sigue primando la comodidad y sencillez al poder contar con un soporte en caso de problemas de autenticación.

En ambos casos se contará también con la aplicación para dispositivos inteligentes, la cual es imprescindible para poder crear el segundo factor de autenticación del proyecto, ya sea un segundo factor de autenticación de tipo posesión, como biométrico.

Si el cliente ha decidido montar el servicio por su propia cuneta, contará con una aplicación base para los sistemas operativos que soporta Xamarin. En base a esa aplicación base, el cliente deberá de configurar y amoldar todo al servicio que el mismo se ha creado en su entorno. Esto en la práctica se traduce a que el mismo deberá de configurar la aplicación, pero además será encargado de la propia seguridad de la aplicación, ya que, volviendo al punto de antes, aunque la aplicación sea segura en sí misma, una mala gestión puede causar inseguridad en la aplicación o en el dispositivo. No solo en el punto de la seguridad tendrá que hacer la configuración, sino que también deberá contar con sus propias cuentas de desarrollador para poder publicar sus aplicaciones en tiendas de aplicaciones de confianza y así distribuir entre sus usuarios la aplicación para los diferentes dispositivos inteligentes.

Si, por el contrario, el cliente opta por contratar nuestro servicio, se creará una aplicación personalizada para los usuarios de dicho cliente. De manera que el cliente puede pedir segundos factores de autenticación adicionales a los que aparezcan en la versión base, así como configurar todos los detalles de la aplicación, como colores, logos, iconos…

Además, el cliente no deberá de preocuparse por nada de la aplicación, todo la configuración de seguridad y la distribución de la aplicación corre la cuenta del servicio que él ha contratado. Además, el cliente puede solicitar aplicaciones especiales para diferentes sistemas operativos, de los cuales no soporte Xamarin.

Para el caso en el que no se ha contratado el sistema de manera privada, las opciones de segundo factor de autenticación estarán limitadas en la aplicación base, en cuanto a las posibilidades de selección biométrica se refiere, es decir, habrá menos opciones entre las que puedas elegir para el segundo factor de autenticación, pero dejando opciones igual de válidas para la seguridad. El tener menos opciones no implica que la seguridad de la autenticación se vaya a ver afectada.

Al ser de código abierto con licencia GNU/Linux GPL el cliente tiene la posibilidad de desarrollar sus propios sistemas de segundo factor de autenticación, siempre y cuando respeten la licencia del software, recordando en este punto que todo lo desarrollado a partir de este proyecto, deberá de contener la misma licencia que este.

Por otro lado, como hemos visto en el punto anterior, los clientes que hayan contratado nuestro servicio contarán con un abanico más amplio en la selección de segundo factor de autentificación dentro de la aplicación, con la opción de pedir segundos factores de autenticación necesarios para sus casos de negocio, en caso de que el segundo factor de autenticación que requiera el cliente no se encuentre en las posibilidades de selección.

La parte más imprescindibles del proyecto es la aplicación para dispositivos inteligentes, la cual te permite el segundo factor de autenticación ya sea por posesión, en este caso el smartphone o por biometría en el caso mínimo de este proyecto sería la huella dactilar.

Esta es una de las razones por las que este proyecto va a estar enfocado a este caso de negocio, en el que contaremos con dos APIs, una para usuario y otra para los clientes, con dos aplicaciones para dispositivos inteligentes, que al igual que la API una será para los usuarios y otra para los clientes. De esta manera podremos montar el servicio privado para los clientes que hemos descrito en este apartado.

3.5.2 Servicio comercial para empresas

Este modelo de negocio es parecido al que sigue por ejemplo Canonical con su servicio para empresas. Este modelo de negocio consiste en distribuir todo el software de manera libre como habíamos dicho en el anterior modelo de negocio, pero en este caso, dejaríamos a los clientes que se encargaran de toda la configuración de la API, servidor y bases de datos.

En este caso nosotros no dispondríamos de un servicio privado, si no que dispondríamos de un servicio de administración y configuración del mismo servicio normalmente dirigido a empresas.

En este modelo de negocio, el cliente puede elegir si el mismo quiere configurar el servicio y desarrollar sus segundos factores de autenticación, o si, por el contrario, quieres contratar el servicio de administración que creó el servicio.

De esta manera, un cliente contrataría nuestros servicios para disponer de una solución oficial y de confianza que les configure el servicio, tenga asistencia y servicio técnico 24/7, máxima seguridad en su sistema de autenticación de usuarios, generación de informes, mantenimiento y desarrollo personalizado.

Para resumir el concepto, sería una manera de delegar la autenticación de usuarios en su propio sistema utilizando un servicio libre garantizando la máxima eficiencia y eficacia.

3.5.3 Donaciones

Este sería el modelo de negocio, con que el que menos llegaríamos a lucrarnos ya que varía mucho en función de si a la gente le ha gustado el proyecto y quiere agradecerlo.

La idea sería usar portales como por ejemplo es del “BountySource”. El cual, es un portal que conecta los proyectos *Open Source* que hay alojados en GitHub con las donaciones. De manera que los usuarios pueden simplemente donar para apoyar el proyecto haciendo un seguimiento de este, pero además también tiene la posibilidad de donar en función de funcionalidades nuevas añadidas o soluciones a bugs en el proyecto. De manera que el portal hace un seguimiento de los *pull requests* del proyecto y cuando los usuarios están contentos con el resultado se libera el dinero al desarrollador del proyecto.

4. Desarrollo del proyecto

En este apartado se describirán todos los procesos de desarrollo del proyecto. Como hemos mencionado con anterioridad, el proyecto cuenta con varias partes, las cuales iremos explicando por separado su desarrollo y como se interconectan todas las partes en el proyecto.

Describiremos también todos los procesos que hemos seguido para que cada una de las partes desarrolladas sean seguras para así conseguir un servicio seguro de autenticación de usuarios.

Durante el desarrollo del proyecto, veremos durante todo el proceso que, se añadirá la cabecera "Access-Control-Allow-Origin" con un valor de "*". Esta cabecera solo estará presente durante el desarrollo.

Una vez el servicio pase a producción, estas cabeceras deberán de ser cambias en muchas de las llamadas, ya que no se permitirá el acceso a todos los recursos desde cualquier punto.

En el caso de la API de usuarios, las llamadas solo podrán ser realizadas desde IPs específicas de los clientes o desde host especificados por los clientes.

En el caso de la API de clientes, ocurrirá más o menos lo mismo, solo que solo se aceptarán llamadas desde nuestras propias IPs de servicio, desde nuestros hosts o desde nuestros productos software.

A lo largo del desarrollo podremos observar la importancia que tienen los segundos factores de autenticación, esto es debido a que todo el servicio desarrollado gira en torno a los segundo factores de autenticación, dando mayor importancia a los mecanismos biométricos para la autenticación de usuarios o clientes.

Veremos cómo en desarrollo cuenta con varios procesos que usan el segundo factor de autenticación. Los valores de dichos métodos de segundo factor de autenticación, se almacenará en la base de datos a consultar correspondiente, es por ello, que cuando algún proceso falla o realizamos alguna petición de cierre de sesión o utilizamos un JWT caducado, todos los valores de los segundos factores de autenticación que tengamos como validos en la base de datos, pasará a invalidarse, haciendo así que el usuario o cliente tenga que volver a realizar todo el proceso de autenticación en el servicio o por lo menos el proceso de autentificación con el segundo factor. De esta forma podemos garantizar la seguridad de que

se usará siempre un segundo factor de autenticación en todo proceso de autenticación del servicio.

4.1 Desarrollo de API para usuarios

Este punto del proyecto será diseñado utilizando Go y en Visual Studio Code. Esta parte del proyecto será hecha de manera modular, es decir, teniendo nuestras funcionalidades en diferentes módulos, en el caso de Go serán paquetes, para ayudarnos a:

- Tener una mejor vista general del proyecto de un solo vistazo, de una manera más comprensible que si tuviéramos todo en un mismo directorio o archivo de código.
- Puedes ver cómo funcionan y trabajan todas las partes y que dependencias hay en tu proyecto, ya que debes de insértalas y conectar las partes antes de poder usarlas.
- Haciendo módulos o paquetes, también obtienes la reutilización de código, ya que todo el código que necesitas lo tienes separado para poder usarlo en otra parte o incluso en otro proyecto, teniendo solo que compartir ese modulo o ese paquete.
- Otra de las ventajas, es a la hora de hacer pruebas, ya que de esta manera puede crear pruebas unitarias para cada módulo o paquete de manera individual.

La estructura del proyecto de la API será la siguiente:

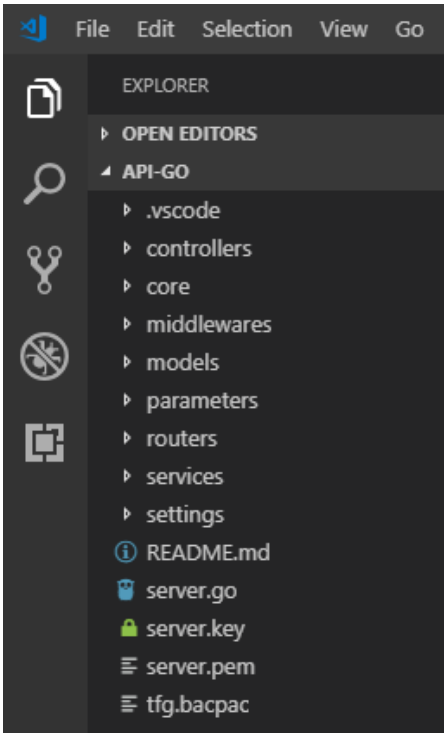

Figura 4 Estructura del proyecto

4.1.1 Servidor Web en Go

Con todo el entorno preparado lo primero que realizaremos será el servidor, el cual será el encargado de levantar el servicio web para poder recibir llamadas a la API, que este servicio contendrá.

Para este proyecto se usará el paquete "net/http" [16] de la librería estándar de Go. Este paquete contiene todo lo necesario para manejar las llamadas entrantes al servidor, así como todas las funcionalidades para configurar el servidor de la manera que se deseé.

Lo primero que crearemos en el servidor, serán las rutas de los servicios de ambas APIs, las cuales serán las que hará uso el servidor para buscar un recurso.

Las rutas las definiremos en otro paquete, es decir en un directorio diferente, denominado "routers", y dentro de este paquete tendremos separadas las rutas por funcionalidad del servicio, versión y en archivos de código diferentes.

Para la creación de rutas, usaremos el paquete de terceros "gorilla/mux" [17], el cual es un potente paquete que implementa un multiplexor de peticiones HTTP, el cual compara las peticiones entrantes con una lista de rutas registradas y para cada petición llamará a su controlador correspondiente, para las llamadas que cumplen las condiciones definidas. Utilizaremos este multiplexor de rutas y no el definido en el paquete "net/http" de la librería estándar de Go, debido a que este paquete nos ofrece más funcionalidades, como por ejemplo la que utilizaremos en este proyecto, me estoy refiriendo a la de poder crear grupos de rutas que comparten unas mismas condiciones, con esta función las llamadas primero tienen que coincidir en la llamada padre, para poder acceder a la subruta del grupo. Esta función es muy interesante pues te permite definir condiciones tanto para la ruta, para el servidor, un prefijo para la ruta o atributos de la ruta.

Este paquete implementa la interfaz del paquete de la librería estándar, por lo que es compatible con el servidor que vamos a montar con la librería estándar.

Continuando la definición de las rutas, nos encontramos con que por cada ruta que hemos creado, debemos de definir un controlador y el tipo de método de la llamada que va a aceptar cada ruta, es decir, podemos definir rutas con un tipo de método GET, POST, PUT, DELETE o OPTIONS. En este caso el paquete nos deja definir una secuencia de métodos para una misma ruta para intentar cumplir las condiciones de la llamada.

Respecto al tema de los controladores, utilizaremos las funciones controladoras de una llamada de manera normal. Denominaremos controlador, a la función que se llamará con cada petición a una ruta definida, pero además utilizaremos otro paquete de terceros llamado "urfave/negroni" [18] que nos ayudará a definir lo que sería una aproximación de un *middleware* en Go. Utilizaremos un middleware para la comprobación del JWT, del cual hablaremos más adelante en el apartado de comprobación de JWT, al igual que del middleware, que hablaremos sobre él en su propia sección.

Una vez creadas las rutas pasaremos a crear la configuración que debe tener el servidor web de manera segura. Como ya hemos visto en el capítulo estado del arte, haremos uso del protocolo TLS.

Para ello deberemos de crear el certificado para el protocolo TLS. Como esta opción cuesta dinero, para el desarrollo del producto, crearemos un certificado de la mano de "Let's Encrypt", la cual es una autoridad de certificación o autoridad certificadora, la cual proporciona certificados X.509 (estándar para certificados de clave publica y un algoritmo de validación de la ruta de certificación) de manera gratuita para el cifrado de TLS, a través de un proceso automatizado que hace más sencillo la instalación y renovación de los certificados de sitios web.

En la guía de "Let's Encrypt", nos aconsejan que, si el certificado lo vamos a crear por terminal, como es el caso para este proyecto, ya que los generaremos en el sistema operativo Ubuntu 16 y para ello usemos el cliente "Certbot ACME", el cual es una página web, que tras indicar el sistema operativo y el software que vas a usar para tu servidor web, este te da una serie de comandos para poder obtener tu certificado de una manera fácil y sencilla.

Ya con los certificados, continuamos con la configuración del servidor, para ello, primero estableceremos la versión mínima la cual será SSL 3.0 (recordemos que SSL es el protocolo anterior a TLS) para poder asegurar la máxima compatibilidad posible, aunque esta versión mínima es muy susceptible a cambiar con el paso del tiempo para asegura una mayor seguridad. Como versión máxima se ha establecido que sea la versión TLS 1.2, no es la última versión de TLS debido a que esta última salió en el verano de 2018, y aun no está disponible dentro del paquete "crypto/tls" [19] [20] de la librería estándar de Go, pero se añadirá en cuanto esté disponible en el paquete.

El resto de la configuración por defecto es muy similar a las que Mozilla tiene es sus guías de seguridad. Sin embargo, indicaremos en la configuración que queremos la opción "PreferServerCipherSuites" activada, lo que hará que en la negociación de la comunicación TLS se prefieran las configuraciones de seguridad de nuestro servidor en Go, lo cual causará más rapidez y más seguridad.

La siguiente opción que indicaremos será "CurvesPreferences" para evitar que se usen curvas no optimizadas, por ejemplo, un cliente usando la "CurveP384" causara que se gaste más CPU en el servidor. Dentro de esta opción pondremos como preferencias la "tls.X25519" y "tls.CurveP256".

Las curvas elípticas en criptografía con utilizadas como una variante de la criptografía asimétrica, que utiliza las matemáticas de las curvas elípticas. La curva 25519 indicada en el párrafo anterior, proporciona 128 bits de seguridad y está diseñada para poder ser utilizada en el esquema de curvas elípticas de *Diffie-Hellman*, es una de las curvas más rápidas. [21]

Por último, en la configuración, añadiremos las diferente *Cipher Suites* que queremos usar como por ejemplo "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384" y algunas más. Con esto ya tenemos toda la configuración para usar TLS en nuestro servidor web en Go.

Una *Cipher Suite* son un conjunto de algoritmos que se utilizan para mejorar y ayudar a asegura una conexión de red que utiliza TLS. Estos conjuntos de algoritmos suelen tener algoritmos de intercambio de claves, algoritmos de cifrado masivo y algoritmos de autenticación de mensajes, véase [22].

Solo nos quedaría crear el servidor con todas las partes que hemos creado, para ello crearemos nuestro servidor indicándole que el puerto que queremos usar es el puerto por defecto para HTTPS, el cual es el 443, como manejador de rutas y multiplexor indicaremos el creado por nosotros en el paquete "routes", como configuración de TLS, le indicaremos que utilice la creada en los párrafos de arriba. Por último, para lanzar este servidor, solo tendríamos que hacer uso de la función "ListenAndServeTLS" pasando como parámetros el directorio donde se encuentran los certificados creados también en los párrafos de arriba, que hemos colocado en este caso en la raíz del proyecto.

De esta forma ya tenemos creado nuestro servidor web en Go, el cual estará en el paquete "main" del proyecto para poder ser ejecutado.

4.1.2 Controladores de las llamadas

A estas alturas del proyecto, tenemos definidas como rutas de usuario las siguientes:

- /login: esta ruta está pensada para realizar el *login* después de haber insertado en el sistema un valor valido para al menos un segundo factor de autenticación, además esta llamada generará un "Access Token".
- /loginMultiFactor: se utilizará para realizar el *login* simple se email y contraseña, no requiere segundo factor de autenticación, generará un "Access Token".
- /insertMultiFactor: esta ruta está pensada para poder enviar un valor de algún método de segundo factor de autenticación.
- /register: esta ruta es para poder hacer un registro de un usuario en el sistema y que posteriormente pueda usar el sistema para autenticarse.
- /edit: esta ruta será utilizada para editar los datos del usuario.
- /logout: la utilizaremos para controlar que el usuario quiere cerrar su sesión
- /refreshToken: esta llamada será utilizada por el cliente para refrescar su "Access Token" sin tener que volver hacer *login*.

Todas las rutas, serán generadas en un paquete generado por nosotros, llamado "routes". Las rutas se agruparán por una URL específica y separadas en ficheros, según el propósito de estas, además las rutas también se versionarán, con el fin de poder hacer actualizaciones en la API sin romper los posibles clientes antiguos que hagan uso de la misma.

Es por ello por lo que, para las rutas de los usuarios, generaremos un fichero nuevo donde estarán todas las rutas dedicadas al usuario con su grupo y su versión. De esta forma, un ejemplo de ruta de usuario sería el siguiente: https://ipServidor:443/v1/users/login.

El motivo por el cual la inserción del valor de un segundo factor de autenticación y el *login* con el valor del segundo factor de autentificación ya insertada está separado, es porque el *login* puede hacerse desde un dispositivo y el envío del valor de un método de segundo factor de autentificación con otro dispositivo diferente. Un ejemplo de este caso sería la autenticación de un usuario contra una página web, el usuario quiere autenticarse en una página web que usa nuestro servicio de autenticación, el cual tiene un segundo factor de autenticación que es la huella dactilar, tras hacer el *login* en la página web, para terminar de autenticar el usuario en la página, este debe de utilizar la aplicación en su smartphone para

hacer la llamada he enviar el valor del segundo factor al servicio. De esta manera logramos poder hacer la autenticación del segundo factor con dispositivos diferentes.

El motivo por el cual tenemos dos llamadas de *login* es porque el servicio solo solicitará el segundo factor de autenticación si el *login* ha sido correcto utilizando la llamada "/loginMultiFactor". En caso de ser correcto el *login* con la llamada "/loginMultiFactor", se procederá a pedir el segundo factor de autenticación al usuario y en caso de un valor valido de este segundo factor, se procede con el *login* con la llamada "/login" para que en la respuesta de esta llamada se devuelva el "Access Token" al usuario, lo cual indicará que el usuario ha sido autenticado correctamente.

Si utilizamos la llamada "/login" antes de haber utilizado el segundo factor de autentificación, la API comenzará a esperar durante dos minutos a que se introduzca un valor valido de algún segundo factor de autentificación para dicho usuario en el sistema. Si pasado los dos minutos, no se ha insertado un valor valido, se enviará una respuesta con código de error y un mensaje de error en el que indicará que debes de utilizar la llamada de "/insertMultiFactor" para añadir algún valor valido de algún método de segundo factor de autentificación.

Para las llamadas de "/login" y "/loginMultiFactor" como para la llamada del registro, el primer paso dentro del controlador es el mismo, se intenta deserializar el cuerpo del mensaje con formato JSON [23] con una estructura de datos, ya definida en un *struct*. Este *struct* será un tipo de dato, en este caso será un tipo de dato usuario.

A estas estructuras de datos, les llamaremos modelos y actuarán como un tipo de dato para el modelo de objeto en el que queremos deserializar el cuerpo del mensaje.

Para poder utilizar estas estructuras con un JSON en la deserialización del cuerpo, debemos de indicar para cada propiedad del *struct* cuál será su correspondiente nombre en el JSON o, mejor dicho, que nombre es el que se tiene que buscar el cuerpo del mensaje para añadir ese valor a la propiedad del *struct*. Un ejemplo de este factor sería la propiedad "Email" que se escribirá como propiedad en el *struct* de esta manera:

```
Email string 'json:"Email,omitempty"'
```

En el proceso de deserialización, se asignará el valor a la propiedad “Email”, cuando en el cuerpo del mensaje que es un JSON se encuentre el nombre “Email” y se omitirá el valor en caso de que ese campo no aparezca en el cuerpo del mensaje, gracias a la propiedad añadida de “omitempty”

Una vez que hemos terminado el proceso de deserialización, lo que le sigue a estos tres controladores es la llamada al servicio. Los servicios estarán en otro paquete, de manera que sigamos la estructura modular nombrada anteriormente. En el paquete de servicio es donde tiene lugar la lógica da cada llamada que recibe la API, como por ejemplo el acceso a base de datos.

En último lugar, encontraremos estos tres controladores añaden a la cabecera de la respuesta información adicional. En este caso, añadimos la cabecera “Content-Type” con valor “application/json” para indicar que el contenido o cuerpo de la respuesta tiene un formato JSON, también en nuestro caso, se le añade la cabecera “Access-Control-Allow-Origin” con un valor de “*” para indicar que cualquier origen pueda acceder a dicho recuso. Se le añade también el estado de la respuesta poniendo código de estado 200 “OK” en el caso de que la petición haya ido bien a lo largo del tratamiento del servicio y poniendo el código de estado 500 “Internal Server Error” cuando en el proceso del servicio ha ocurrido un error. Por ejemplo, si el registro de un usuario ha ido bien, el estado será 200, pero si por ejemplo durante el proceso de registro ha habido problemas con la conexión a la base de datos, entonces se devolverá una respuesta con código 500 para indicar que ha habido un error.

También al final y dependiendo ya de cada controlador, puede haber una cosa más que se escribe en el cuerpo de la repuesta. En el caso del registro, devolverá en el cuerpo de la respuesta la contraseña en caso de que hayas solicitado al servicio que deseabas que te generará una para el registro. O por ejemplo en el caso de *login* de un usuario, también se escribe en el cuerpo de la respuesta el “Access Token” junto a su tiempo de vida.

Solo hemos visto cómo trabajan los controladores de *login* y el controlador de registro. Los demás controladores tienen unas características diferente para poder cumplir su cometido.

El controlador para la llamada “/insertMultiFactor”, usa su propio modelo de estructura de datos, esto es así porque tiene los datos del segundo factor de autentificación. Al igual que los anteriores controladores lo primero que intenta realizar es deserializar el cuerpo de la petición a este modelo de objeto para poder trabajar con él.

Si la deserialización no ha fallado, lo primero que realizaremos será un proceso de autenticación con el email y la contraseñas enviado en el cuerpo de la petición, para comprobar que el usuario existe en la base de datos y que los datos proporcionados por el usuario son correctos. Si el proceso de autenticación ha sido satisfactorio, entonces se realizará la llamada al servicio encargado de la inserción de los valores del segundo factor de autenticación y devuelve escrito en la cabecera de la respuesta el código de estado, en función de cómo haya ido la petición en el servicio, esta llamada no devuelve nada más en el cuerpo del mensaje de respuesta.

Los tres controladores que nos quedan tienen algo especial. Lo que tienen de especial con respecto al resto de controladores para las llamadas es que las llamadas tanto de "/edit", "/logout" como de "/refreshToken" pasarán por un middleware antes de que la petición llegue al controlador de la llamada. Este middleware, como veremos en el siguiente apartado, es el encargado de la comprobación del token.

Para la llamada realizada a "/edit", lo primero que realizaremos, será obtener de la cabecera de la petición, el ID del cliente al que pertenece ese usuario, así como el email de dicho usuario. Este ID y email está en la cabecera gracias al middleware, veremos en su apartado como procede el middleware para este caso.

Con estos valores llamaremos al servicio "CheckMultiFactor", el cual comprobará si hay un valor valido para alguno de los métodos de segundo factor de autenticación. En caso afirmativo se procederá a deserializar el cuerpo de la petición en un tipo de dato, creado por nosotros, el cual contendrá todos los datos necesarios para la actualización de los datos de un usuario en la base de datos.

Para actualizar valores de un usuario, llamaremos al servicio "UpdateUserDataService", la respuesta a este servicio nos devolverá un código de estado y un mensaje, el cual grabaremos en el cuerpo de la respuesta, junto a las cabeceras correspondientes nombradas anteriormente.

En el caso del controlador para la llamada "/logout", lo primero es recuperar el ID del cliente al que pertenece dicho usuario y el email del usuario que tenemos en la cabecera de la petición.

En este caso llamaremos al servicio "insertMultiFactor", para que anule todos los valores de los métodos de segundos fatores de autentificación, consiguiendo así anular el *login* de dicho

usuario en nuestro sistema, de modo que si quiere volver a utilizar el servicio deberá volver a insertar el segundo factor de autenticación y hacer el *login* de nuevo. Al igual que como hemos visto en el resto de los casos, se añadirán las cabeceras correspondientes, con un código de estado según haya ido el procesamiento de la petición en el servicio.

Por último, tenemos el controlador para la llamada "/refreshToken". Al igual que ocurría en el controlador para el *logout*, lo primero que haremos será extraer el ID y el email del usuario situado en la cabecera de la petición. Con estos valores, llamaremos a un servicio para comprobar el segundo factor de autenticación, en caso de que el valor de algún segundo factor de autenticación esté disponible y sea válido, entonces procederemos a llamar al servicio para refrescar el token del usuario y devolverlo en el cuerpo de la llamada, y al igual que hemos visto antes añadiremos las cabeceras de "Access-Control-Allow-Origin" con valor "*", la cabecera de "Content-type" para indicar que el cuerpo de la respuesta es en formato JSON, junto al código de estado para indicar que todo ha ido bien o no en el procesamiento de la petición. En caso de que el usuario no tenga el *login* o no tenga un valor valido para el segundo factor de autenticación, devolveremos un código de estado 401 "Unauthorized".

Con este apartado hemos podido ver cómo trabajan todos los controladores con el tratamiento de sus posibles llamadas.

4.1.3 Middleware

Como hemos visto en el apartado anterior, las llamadas para editar los datos de un usuario, refrescar el token y para hacer *logout* en el servicio, utilizaban un middleware antes de pasar al controlador. El middleware solo se usará cuando tengamos que comprobar el "Access Token"

Un middleware, también es llamado lógica de intercambio de información entre aplicaciones, normalmente se habla de middleware como software que permite a una aplicación comunicarse con otras aplicaciones, paquetes de programas, redes, maquinas (refiriéndonos a máquina como ordenador) o incluso con el sistema operativo. También es conocido por ser software que se utiliza para conectar sistemas distribuidos.

Yo he llamado middleware a un paquete generado por mí, que permitirá o no pasar mensajes entre mi propio software, es decir actuara como barrera entre las peticiones y controlador. Aunque no es la definición que se tiene normalmente de middleware, es la forma más clara

que he visto para nombrarlo, ya que solo dejara pasar la petición al controlador de la llamada en el caso de que en la petición aparezca un JWT valido.

Es también una manera de abstracción de la comunicación, ya que a todas las llamadas a las cuales se les añada este middleware, tendrán que pasar por este paquete intermedio para saber si la petición puede seguir propagándose a través de la API o si por el contrario es descartada.

En este paquete de middleware encontraremos únicamente una función, llamada "TokenAuthentication", la cual será encargada de validar el JWT.

Para este paquete haremos uso de dos paquetes de terceros llamados "dgrijalva/jwt-go" [24] [25] y "dgrijalva/jwt-go/request". Estos paquetes nos proporcionarán la implementación de los JSON Web Tokens en Go. Haremos también uso de estos paquetes en el paquete encargado de generar el token, que veremos en otro apartado más adelante.

La función de "TokenAuthentication" funciona de la siguiente manera. Lo primero que realizamos es extraer el JWT de la petición, que estará en la cabecera "Authorization" y comprobar que el JWT ha sido firmado por nuestras claves de RSA (Como veremos en el apartado de generación de JWT, utilizaremos claves RSA para firmar el token). Si no es token firmado por nosotros, se emite un mensaje de error que se registrará en el log de la API y no se deja pasar la petición al controlador, acabando en ese punto el procesamiento de la petición.

Una vez que hemos comprobado que el token ha sido firmado por nosotros, es decir, ha sido generado por nosotros, lo siguiente que tenemos que hacer es comprobar que ese JWT es válido. Sé devuelve un código de estado 401 "Unauthorised", en caso de que no sea un JWT valido.

Si el token es un token valido, se pasará a comprobar si el token ha expirado. Si todo ha ido bien y el token no ha expirado, se extraerá de los *claims* del token, el ID del cliente al que pertenece dicho usuario junto al email de dicho usuario. Toda esta información extraída, se añade a la cabecera de la petición para que pueda ser usado por los controladores como hemos visto en el apartado anterior. Si todo el proceso ha ido correctamente la función deja pasar la petición a la siguiente función o controlador y termina. Si el JWT en lugar de pertenecer a un usuario, fuera de un cliente, el proceso sería exactamente igual.

En caso de que el token haya expirado, ocurren dos cosas, la primera es que se anulan todos los posibles valores de todos los posibles métodos de segundo factor de autenticación que tengamos en la base de datos, es decir ocurre lo mismo que cuando haces *logout*, después se escribe en la cabecera de la respuesta el código de estado 401 "Unauthorized" y no se continúa procesando la petición.

Como vemos este middleware solo dejará pasar la petición al controlador en caso de que todas las comprobaciones del token hayan funcionado. De no ser así se descarta la petición y se devuelve un mensaje de error. Es una comprobación necesaria común a todas las llamadas que requieran de un "Access Token" para funcionar,

4.1.4 Servicios de la API usuarios

Llamaremos servicio a un paquete en el que se encontrarán todas las funciones encargadas de tratar toda la lógica de negocio de las peticiones a la API. Por lo tanto, serán estas funciones las encargadas de acceder, insertar y actualizar los valores de nuestra base de datos.

Para el proyecto vamos a ver que tenemos definidos varios servicios diferente para los propósitos de la API, encontramos los siguientes servicios definidos para la API de los usuarios:

- LoginService: esta función es una de las más importantes, ya que es la función encargada de realizar el *login* y la encargada de devolver el "Access Token" al usuario.
- LoginMultiService: esta función es muy parecida a la función de "LoginService" aunque se distinguen en varios aspectos, la mayor diferencia radica en que, aunque también realiza el *login* de un usuario, en esta función no se genera el "Access Token" para el usuario.
- RefreshToken: función encargada de refrescar el token y devolverlo al usuario.
- CheckMultiFactor: está función es muy útil para utilizar junto a otras funciones ya que comprueba que los valores del segundo factor de autenticación sean válidos, será compartida por ambas API para dicha comprobación.
- InsertMultiService: función para insertar el valor del segundo factor de autenticación en la base de datos.

- DoRegisterService: otra de las funciones importantes, ya que es la encargada de realizar un registro correcto de un usuario en nuestra base de datos.
- UpdateUserDataService: esta función será la encargada de actualizar los datos del usuario.

4.1.4.2 DoRegisterService

Esta función es la más importante junto con la función de "LoginService". Esta función de encarga del registro de un usuario en el sistema.

Como hemos podido ver en el controlador de la función, a esta función del servicio, le llega como parámetro un modelo de tipo de dato usuario como estructura de datos. Esta estructura de datos, recordemos que contiene los valores que venían en el cuerpo de la petición del mensaje.

Por lo que, lo primero que comprobaremos antes de registrar a un usuario en el sistema, es que el email recibido en el cuerpo del mensaje es un email correcto y que, en el caso de haber ingresado una contraseña en lugar de pedir la creación de una al servicio, ésta sea correcta.

El usuario en este punto puede haber enviado la contraseña que desea usar en el servicio o puede haber pedido a la API que le genere una contraseña. En caso de haber solicitado la contraseña, se hará uso del paquete "core", el cual es un paquete generado por nosotros con el fin de que sea el paquete encargado de tener la funciones que fueran necearías para generar datos adicionales necesarios para el correcto funcionamiento de ambas APIs, es decir datos necesarios para el correcto funcionamiento del servicio.

En el caso de haber enviado él su propia contraseña, se comprobará que al menos cumple los requisitos mínimos de fuerza de contraseña para este servicio impuestos por el cliente al que intentando registrarse. Estos requisitos son, que al menos tenga ocho caracteres, así como al menos un número y al menos un carácter espacial o símbolo por defecto, aunque el cliente puede imponer los requisitos que el consideré oportunos.

En caso de haber solicitado la generación de una contraseña, esta será creada con las condiciones que hayan indicado en el cuerpo de la petición y le será devuelta en el cuerpo de la respuesta.

En el caso de que email o la contraseña del usuario no tengan el formato correcto, se terminará la función y se devolverá el código de estado 400 “Bad Request” con el mensaje de error correspondiente a cada caso.

Si todo ha ido bien, tendremos un email y una contraseña válidos. A continuación, y como veremos a lo largo de todas las funciones para el paquete de “services”, lo primero que realizaremos en esta función, será abrir la conexión a la base de datos. Para ello, como hemos nombrado anteriormente, utilizaremos el *driver* recomendado por Microsoft “denisenkom/go-mssqldb”.

Para abrir la conexión a la base de datos, usaremos también un paquete creado para este proyecto, llamado “parameters”. Este paquete ha sido creado con el fin de tener en un único paquete todos los parámetros para establecer la conexión a la base de datos, así como la cadena de conexión, común a todas las funciones. Además, en el paquete “parameters” también encontraremos los parámetros para usar Argon2, para que todos los parámetros sean iguales en todo el servicio. Además, en este paquete “parameters”, indicaremos el directorio para las claves RSA que se usan en la generación del JWT.

Después de abrir la conexión, la configuraremos para que se cierre una vez dejemos de usarla. Este punto es muy importante para el tema de rendimiento, y más teniendo en cuenta que para este proyecto empezamos con máquinas escasas de recursos en Azure. El cierre de la conexión a la base de datos en Go debe hacerse con la palabra clave del leguaje “defer”, la cual evalúa la función que tiene que ejecutar, en nuestro caso “`Close()`”, pero no ejecuta la función, si no que aplaza la ejecución hasta que se termine de ejecutar el código necesario para realizar las operaciones, por lo que “defer” sirve para poder cerrar y liberar recursos.

Una vez conectado a la base de datos y con toda la información del usuario validada, procedemos a realizar resúmenes de dichos datos, para almacenarlos en la base de datos. Lo primero que haremos, será utilizar la función de derivación de claves Argon2.

Utilizaremos para ello el paquete “crypto/argon2” y la función “IDKey”, contenida en el paquete, la cual implementa la función Argon2id. Esta función necesita de parámetros contenidos en el paquete “parameters” para funcionar. Los parámetros que hemos definido en dicho paquete serán la mejor aproximación en base a nuestros recursos para conseguir un correcto funcionamiento junto a un buen rendimiento para esta función de Argon2.

Los parámetros que he definido para la función son los siguientes:

- Numero de bytes mínimos para la *sal*: 32
- Tiempo usado: 1 (Número de iteraciones sobre la memoria)
- Memoria usada: 64 * 1024
- Hilos utilizados: 4
- Longitud mínima de clave: 32

Lo primero que derivaremos y sobre lo que aplicaremos un *hash* es sobre el email del usuario. Para el email utilizaremos como *sal*, la definida en el cliente contra el que se está registrando. Como veremos en el apartado de la API del cliente, cuanto este se registra un cliente, se creará una *sal* para los emails, esta *sal* será la que utilice el usuario cuando intente registrarse en el sistema para un cliente dado. La única intención de aplicar Argon2 sobre el email, es que este no aparezca en texto plano en la base de datos. Y los demás parámetros utilizados para función de Argon2, son los definidos arriba.

Como resultado tendremos un array de bytes, el cual codificaremos en *base64* [26], para posteriormente almacenarlo en la base de datos como una cadena de caracteres.

Ahora deberemos de aplicar la función de Argon2 sobre la contraseña, pero en este caso, no utilizaremos una *sal* definida en el cliente contra el que nos estamos registrando, sino que utilizaremos una función definida en el paquete "core" encargada de generar una *sal* de tamaño del que nosotros le hayamos indicado (en el caso del proyecto serán 32 bytes) de manera pseudoaleatoria. De este modo conseguiremos generar una *sal* para cada usuario de manera que, para cada usuario, sea diferente.

Una vez tengamos esta *sal* creada, podemos pasar a utilizar la función de Argon2 con la contraseña utilizando los mismos parámetros que utilizamos para el email, pero en este caso utilizando la *sal* generada por nuestra función generadora de *sal*, que veremos su funcionamiento en su correspondiente apartado.

La contraseña tras aplicar la función de Argon2, tendremos los datos en bytes junto a la *sal* que hemos generado para el usuario, también en bytes. Tendremos que codificar todo en *base64* y una vez que lo tengamos todo codificado, podemos pasar al siguiente paso que es la inserción del nuevo usuario en la base de datos.

Puede ocurrir un error de clave primaria en la inserción del usuario en la base de datos debido a que la clave primaria de la tabla de usuarios es el email del usuario. Por lo tanto, el error, se debe a que el email ya está en la base de datos, es decir el usuario ya está registrado con ese email. En ese caso se termina la función y se envía el código de estado 400 “Bad Request” junto con el mensaje “User already exists”

Si el usuario se ha insertado correctamente en la base de datos, entonces también insertaremos en la tabla de los métodos de segundo factor, los valores correspondientes. Esta tabla será también por cliente y contendrá un registro con el email del usuario como clave principal y todos los valores de los segundos factores de autenticación con un valor por defecto a falso o valor invalido.

Para terminar si todo ha ido correctamente, se devolverá el código de estado 200 “OK” y el cuerpo de la repuesta, que en este caso pueden ocurrir dos cosas, si el usuario, solicitó una contraseña, en el cuerpo de la respuesta, se le devolverá la contraseña que se generó para ese usuario y si no se solicitó ninguna, se enviará un mensaje para decir al usuario que se ha registrado correctamente.

También para terminar, si el proceso de registro de usuario en la base de datos ha ido mal en cualquier momento del proceso, la respuesta tendrá un código de estado 500 “Internal Server Error” con el mensaje de error “User was not registered”.

4.1.4.2 LoginService

Esta función también comenzaremos como todas las funciones para el paquete de servicio, abriendo la conexión a la base de datos.

Después de abrir la conexión, configuraremos para que se cierre la conexión a la base de datos una vez dejemos de usarla. Como hemos hecho en el caso del registro.

También como hemos podido observar en el controlador de la función, a esta función del servicio, le llega como parámetro un modelo de tipo de dato usuario, como estructura de datos.

Una vez conectado a la base de datos y con toda la información del usuario, procedemos a realizar el *login*. Lo primero que debemos hacer es utilizar la función de derivación de claves Argon2.

Lo primero que realizaremos en este caso será recuperar la *sal* para los emails del cliente en el que se está intentando hacer *login*. Para ello, utilizaremos el ID del cliente que ha sido enviado en la petición. Si todo ha ido correctamente, dispondremos de la *sal* con la que se realizó el *hash* del usuario en el registro. En caso de no haber ido correctamente se devolverá un código error en el estado.

Lo primero que derivaremos y sobre lo que aplicaremos un *hash* es sobre el email del usuario, para poder utilizarlo en la búsqueda del usuario en la base de datos.

Llegados a este punto, ya tenemos el email codificado en *base64*, el cual nos servirá para hacer búsquedas en la base de datos. Pero antes de seguir con el *login*, lo primero que haremos será comprobar los valores de los métodos de segundo factor de autenticación, de dicho usuario, en dicho cliente. Para ello utilizaremos la función que hemos creado en el paquete de "services" "CheckMultiFactor", con ella podremos ver si el usuario ha añadido un valor valido para alguno de los segundos factores de autenticación en la base de datos.

En el caso de no disponer de un valor valido de segundo factor de autenticación en la base de datos, la función esperará dos minutos, durante los cuales esperará a que se añada un valor valido para alguno de los métodos de segundo factor de autenticación en la base de datos. Si pasado los dos minutos, la función no ha sido capaz de conseguir un valor valido, se devolverá como código de estado 401 "Unauthorized" acompañado del mensaje "You have to use the second authentication factor" y termina la función, devolviendo ese error en la respuesta al usuario.

Bien, si todo ha ido bien, en este punto de la función ya podemos lanzar la consulta a la base de datos, para que nos devuelva el *hash* de la contraseña y la *sal* que he se ha utilizado para el *hash* de esa contraseña. De manera que la consulta quedará del estilo, devuélveme la contraseña y la *sal* para este email y en este cliente.

En caso de error con la base de datos, durante el proceso de búsqueda, se devolverá un código de estado 500 "Internal Server Error" junto con el mensaje "Error" y termina la función.

Una vez tengamos la contraseña y la *sal* de la base de datos, lo que debemos de hacer es decodificar en *base64* ambos valores, para poder hacer una comparación entre los bytes de la contraseña almacenada en la base de datos, con los bytes del *hash* de la contraseña que el usuario nos ha enviado en la petición.

Para poder hacer la comparación, realizaremos el proceso de *hash* de la contraseña que el usuario nos envió en la petición junto a la *sal* que hemos recibido de la consulta a la base de datos. De esta manera, que, si es la misma contraseña, obtendremos el mismo *hash*.

En caso de haber coincidencia en la comparación de bytes de contraseñas, el usuario ha sido autenticado en el servicio. Si la comparación fuese errónea, es decir, las contraseñas no coinciden, de devolverá al igual que antes el código de estado 500 "Internal Server Error" con el mismo mensaje de error y termina la función.

Por último, como el usuario ha sido autenticado correctamente, se generará un "Access Token" con la tecnología JWT, para el email de ese usuario y para ese cliente en concreto. Este JWT se devolverá junto al tiempo de vida, es decir, hora en la que caducará dicho JWT, para que el usuario lo guarde y sepa cuando tiene que renovar el "Access Token". Todo lo anterior respecto al JWT sería el cuerpo de la respuesta, que iría acompañado del código de estado 200 "OK".

4.1.4.3 LoginMultiService

Esta función es muy similar a la función de "LoginService". El proceso de *login* es el mismo, es decir, también recibe la información del usuario como parámetro, que será la información que el usuario haya enviado en la petición. Esta información a su vez será la que se comparará con la información de la base de datos, para realizar el proceso de autenticación con la comparación de información.

La diferencia de esta función con respeto a la otra reside en varios factores. El primero es que esta función no comprobará el estado del segundo factor de autenticación. Por lo que no esperará a que el usuario inserte un valor correcto para un segundo factor de autenticación.

La siguiente diferencia es que esta función no generará un token para el usuario que está haciendo el proceso de *login*, solo devolverá verdadero o falso, en función de si el proceso de *login* ha funcionado correctamente o si por el contrario no se ha conseguido autenticar al usuario con éxito.

Por lo tanto, para resumir, es una función cuyo propósito es autenticar sin necesidad de un segundo factor de autenticación. Esta función se utilizará durante el proceso de autenticación, antes de que el cliente solicite la autenticación con la función "LoginService" para saber si el usuario es válido para el servicio o no.

Un ejemplo más claro de su uso sería una aplicación cliente que quiere autenticar a un usuario, primero comprobará con el usuario y contraseña introducidos por el usuario son válidos y que por lo tanto son usuarios válidos para el servicio. Una vez realizada esa comprobación, la aplicación cliente solicitará el *login* con la función "LoginService", la cual comprobará el segundo factor de autenticación y en caso de ser correcto, devolverá el "Access Token" al usuario.

4.1.4.4 InsertMultiService

Esta función es primordial para el segundo factor de autenticación. A esta función le llegará por parámetro un modelo de usuario distinto al de las otras funciones. En este modelo, se indicará el email del usuario y su contraseña, para un cliente dado, pero además se indicarán todos los valores de todos los métodos de segundo factor de autenticación que se deseen insertar en la base de datos.

Una vez la petición ha llegado a este punto, ya se ha realizado el proceso de autenticación llamando al servicio en el controlador y si la autentificación no era correcta, no podemos insertar valores para el segundo método de autenticación, como vimos en el apartado anterior, por lo tanto, llegado a este punto, se considera que el usuario es válido y simplemente se abre la conexión a la base de datos como hemos visto en los apartados anteriores para insertar los valores en la base de datos. Es cierto que, para poder insertar en la base de datos, antes tenemos que hacer el proceso de recuperar la *sal* para los emails del cliente, pasar el email por la función de Argon2 y posteriormente codificarlo en *base64*, ya que el email no está en texto plano y recordemos que será nuestra clave primaria en las tablas de las bases de dato.

Si todo ha ido bien se devuelve el código de estado 200 "OK" y en caso de que haya habido un error a la hora de insertar los datos, de devuelve el código de estado 500 "Internar Server Error"

4.1.4.5 CheckMultiFactor

Esta función será consumida en varias partes ambas APIs, tanto la de usuario como la de cliente, pero no tendrá una llamada directa desde una ruta, es solo de uso interno. El resto de las funciones de servicio y los controladores, la usará para saber si deben de seguir tratando la petición o por si lo contrario, deben descartarla y devolver un error al usuario.

El proceso es muy similar a los ya visto en las otras funciones. A esta función le llega como parámetro, el ID del cliente y el email del usuario, ya que para el proceso de comprobación no necesita nada más.

Para comenzar se abrirá la conexión a la base de datos y al igual que siempre que queremos interactuar con la base de datos, aplicaremos la función de Argon2 con el email y el resultado lo codificaremos en *base64*.

Una vez tengamos el resumen del email, ejecutaremos la consulta en la base de datos, para el cliente dado como parámetro, para buscar todos los valores de los métodos de segundo factor de autenticación. Los valores recuperados de la consulta serán los que se devuelvan en la función.

En caso de que algo vaya mal durante la consulta a la base de datos, se devolverán todos los valores a falso, para evitar un que se lance un error o haya un fallo de seguridad.

4.1.4.6 RefreshTokenService

Al igual que ocurría en la función anterior, una vez que se llega a esta función de servicio, no se comprueba nada más, ya que si la petición a llegado hasta aquí, es que ha conseguido pasar todos los filtros y por lo tanto se trata de un usuario valido. Este proceso es así, para evitar un exceso de computo, que lleve a un exceso de uso de los recursos y que por lo tanto se traduce a un tiempo de respuesta mayor para el usuario.

Esta función, al igual que en las otras funciones, recibe como parámetro el ID del cliente al que pertenece dicho usuario y el email del usuario al que va a refrescar el token. Ambos valores se le envían a la función porque son necesarios para generar un JWT nuevo. El email no hace falta que lo procesemos con la función de Argon2, se le envía para poder utilizarlo en el proceso de creación del JWT.

Esta función hará uso del paquete ya nombrado “core”, creado por nosotros para crear el JWT. Si todo el proceso de creación ha ido bien, se devuelve en el cuerpo de la respuesta el JWT junto a su tiempo de vida. Todo ello acompañado como siempre de un código de estado en la cabecera, en este caso el código de estado 200 “OK”.

Si el proceso de generación de un nuevo JWT falla, no se devuelve nada, solo un mensaje de “Error” acompañado de un código de estado 500 “Server Internal Error” en la cabecera del mensaje de respuesta.

4.1.4.7 UpdateUserDataService

Este servicio, será el encargado de realizar el cambio de contraseña de los usuarios y de los clientes, es decir, este servicio será compartido por ambas APIs, tanto la de usuarios como la de clientes.

Para poder acceder a esta función, hace falta poseer un JWT valido en la cabecera de la petición, de no ser así no se podrá alcanzar esta ruta del servicio.

La función recibirá como parámetro, el ID del cliente al que pertenece el usuario, en caso de cliente, se utilizará un ID genérico para todos los clientes. También se recibirá como parámetro el email del usuario o cliente que se quiera cambiar la contraseña, así como un tipo de dato generado por nosotros en el paquete "models", llamado "UserEditData", que será genérico y usado por ambas APIs. Dicho modelo contiene en su estructura de datos la estructura necesaria para almacenar la contraseña antigua y dos veces la nueva contraseña, ya que será este tipo en el que se deserialice el cuerpo de la petición.

Lo primero que comprobará la función, es que las contraseñas contenidas en la estructura de datos sean todas mayor de ocho caracteres, así como al menos un número, al menos un símbolo o carácter especial y que las contraseñas nuevas repetidas coinciden. En caso de no cumplir alguna de estas condiciones, se devolverá un código de error "Bad Request" junto con un mensaje indicando que hay un error en el cuerpo de la petición.

Si, por el contrario, se cumplen todas las comprobaciones, se abrirá la conexión a la base de datos. Con la conexión abierta realizaremos un proceso de *hash* con la función de Argon2 sobre el email, para poder realizar una consulta a la base de datos, para poder recuperar el resumen que tenemos almacenado para la contraseña de dicho cliente o usuario.

Si el proceso de la consulta ha ido mal, devolveremos un código de estado 500 "Internal Server Error" junto a un mensaje indicando que ha habido un problema para recupera la información de la base de datos.

Realizaremos el proceso de *hash* sobre la contraseña antigua que nos ha enviado el cliente/usuario, para comprobar que el resumen es el mismo que tenemos en la base de datos. En caso negativo, se acaba la función y se devuelve un mensaje de error.

En caso de coincidir, se procede a generar una nueva *sal* para la nueva contraseña enviada en la petición, con esta nueva *sal*, realizaremos el proceso de *hash* con la función de Argon2

y realizaremos la transacción en la base de datos para actualizar el resumen de la contraseña y su *sal* correspondiente.

4.1.5 Núcleo de la API

En este apartado veremos el desarrollo del paquete "core". Este paquete actuará como núcleo de ambas APIs. Este paquete estará compuesto por tres archivos, uno llamado "auth_repository" en el cual se encontrarán todas las funciones para el núcleo de la autenticación, otro archivo llamado "password_respository" que será el encargado de la generación de contraseñas, por último, el archivo "salt_repository" que será el utilizado para la generación de la *sal* para las funciones de resumen.

4.1.5.1 Auth Repository

En este archivo se tratará la generación del JWT, que es utilizado a lo largo de todos los procesos de las APIs.

Para firmar los JWT utilizaremos nuestras propias claves publica/privada de RSA. Lo primero que hará el servidor web de la API cuando arranque, será cargar dichas claves del directorio que nosotros le indiquemos.

Para cargar las claves, lo primero que debemos de hacer leer el fichero de la clave del directorio, de manera que tengamos los bytes del fichero. Una vez que tengamos los bytes, utilizaremos el paquete "crypto/x509" [27] de la librería estándar de Go, para transformar los bytes a una estructura de claves públicas, siguiendo el estándar UIT-T (Unión Internacional de Telecomunicaciones)

Como esto supone bastante carga de trabajo si lo tuviéramos que hacer en cada llamada, lo que hace el servidor web cuando arranca, es cargar las calves y almacenarlas en una variable con una estructura de datos definida por nosotros, siendo esa variable accesible mediante un puntero. De esta forma solo cargamos las claves una vez cuando arranca el servidor web y las almacenamos en memoria para poder usarlas en la ejecución de la API.

Ahora una vez que tengamos la claves de RSA, podemos generar un JWT cada vez que sea necesario. Dentro de este archivo podremos encontrar una función llamada "GenerateToken", la cual recibe como parámetro el ID del cliente al que vamos a generar el JWT y el email del usuario de dicho cliente, que se utilizará más adelante para añadirlo al cuerpo del token.

Para generar un JWT, lo primero que debemos de hacer, es generar un token vacío, indicándole al método de firma que vamos a utilizar, en mi caso utilizaré SHA-512.

Después calcularemos el tiempo de vida que le queramos dar al token, lo más aconsejable es lo que se ha desarrollado en este proyecto, que es un tiempo máximo de una hora de vida. Este tiempo de vida tendrá un formato Unix, es decir, en formato UTC y en segundos.

Ya con el tiempo de vida calculado, se deben de crear los *claims*, que será en contenido que formaran el cuerpo del token. Los *claims* cuentan con una estructura en forma de clave/valor. En el caso de este proyecto y haciendo una primera aproximación hemos indicado tres *claims*, pero estos se pueden ir ampliando conforme el proyecto vaya creciendo en el caso de que nos haga falta, por ejemplo, en próximas aplicaciones de ambas APIs.

Lo más aconsejable a la hora de generar las *claims*, es seguir la estructura estándar que hay definida para las claves. En una primera aproximación el token tendrá la siguientes *claims*:

- "sub": la cual contendrá el valor del ID del cliente.
- "email": como su propio nombre indica, contendrá el email del usuario.
- "exp": albergara el tiempo de vida del token en formato Unix.
- "nbf": el instante en el tiempo en el que se ha generado el token, también en formato Unix.

Una vez hayamos creado la estructura de los *claims*, solo nos queda realizar el proceso de firma del token utilizando nuestra clave privada de RSA. Con el token firmado, la función lo devuelve junto al tiempo de vida.

4.1.5.2 Password Respository

Este archivo será el encargado de generar las contraseñas cuando el usuario solicite una, en el proceso de registro.

Las contraseñas podrán ser generadas según los parámetros que indiquemos. Podemos seleccionar la longitud de la contraseña, si deseamos mayúsculas, minúsculas, números, símbolos o una combinación de ambos.

Para generar contraseñas, utilizaremos el paquete "crypto/rand" de la librería estándar de Go, para generar números pseudoaleatorios de una manera segura. Ya que este paquete implementa un generador de números pseudoaleatorios que es criptográficamente seguro (CSPRNG), el cual cumple unas ciertas características que lo hacen adecuado para su uso.

Por ejemplo, un CSPRNG, puede obtener una entropía de calidad, gracias a un generador de números pseudoaleatorios en hardware.

Para algunas de las posibles opciones de configuración de una contraseña, es decir para las mayúsculas, minúsculas y los símbolos, tendremos un mapa por cada una de estas opciones. Un tipo mapa o "map" en Go, es lo que se conoce en otros lenguajes de programación como diccionarios, es decir estructuras de clave/valor. En estos mapas, nuestra clave será un número, para poder usar el numero pseudoaleatorio para acceder al mapa y el valor será de tipo "rune" (32-bits de valor entero) para representar un carácter Unicode.

De este modo, usando el mapa junto a los números pseudoaleatorios, podemos generar caracteres del tipo que queremos de una manera pseudoaleatoria, que, al unirlos, nos dará la contraseña para el usuario.

En el caso de los números para la contraseña, tan solo utilizaremos el generados de números pseudoaleatorios, para generar números comprendidos entre el 0 y el 9, ambos incluidos.

La única restricción que hay a la hora para generar una contraseña para el usuario, es que este en su solitud de contraseña haya indicado una longitud de contraseña mayor o igual a ocho, ya que es la longitud mínima que se considera de segura para ataques de fuerza bruta.

Las minúsculas se usarán por defecto en caso de no haber indicado ninguna otra condición, así como un número y un carácter especial o símbolo. Por lo tanto, si el usuario indica que quiere una contraseña de ocho caracteres con mayúsculas, números y minúsculas, un ejemplo de contraseña sería: H23tY5vT.

4.1.5.3 Salt Repository

El ultimo fichero es el encargado de generar la *sal* que es utilizada a lo largo de todos los procesos del servicio.

Esta *sal* será un array que contendrá bytes, tantos como nosotros le indiquemos. En este caso utilizaremos 32 como tamaño de *sal* y utilizaremos el paquete "crypto/rand", mencionado en el apartado anterior, para generar los bytes de manera pseudoaleatoria, pero de una manera segura para su uso.

4.2 Desarrollo de API para Clientes

Esta API al igual que ocurría con la API de usuarios, también será creada en el lenguaje de programación Go y bajo el editor de código Visual Studio Code. Esta API a su vez será creada y almacenada en el mismo servidor web que creamos en el apartado anterior para contener la API de usuarios.

Como vimos es el apartado de *Controladores para las llamadas*, las llamadas se agrupan según su propósito y versión, además están separadas en distintos archivos dentro del mismo paquete que nosotros generamos. Gracias a esto podemos generar varias APIs diferentes, dentro de un mismo servidor web, de manera que ambas APIs están separadas y son diferentes, tanto la manera de procesar los datos, como su lógica, por ejemplo, diferentes bases de datos. Pero, aunque estén separadas y sean diferentes, al estar en el mismo servidor web, pueden interactuar entre ellas e incluso compartir código para algunas de sus funcionalidades, como vimos en algunos casos en el apartado anterior, de este modo conseguimos que no tener código duplicado.

Eso es lo que hemos realizado para el caso de este proyecto. Como vimos en el apartado del caso de negocio, un servicio privado, donde las empresas actuarán como clientes, que, al darse de alta en nuestro servicio, sus usuarios podrán usar nuestro servicio de autenticación.

Para aprovechar el servidor web que hemos montado ya, con toda la seguridad necesaria para considerarlo un servidor web seguro, hemos creado otra API junto a la que ya teníamos para la autenticación de usuarios, pero esta vez será una API para nuestros clientes, en nuestro servidor web. Esta API también será creada de manera modular por los mismos motivos vistos en el apartado anterior cuando describimos como iba a ser el desarrollo de este proyecto.

4.2.1 Controladores para las llamadas

Al igual que ocurría para la API de usuarios, tenemos que crear un controlador por cada llamada. Las llamadas para poder ofrecer un servicio mínimo a los clientes será las siguientes:

- /: esta llamada será la encargada de devolvernos todos los clientes que tenemos registrados en nuestra base de datos.

- /data: esta llamada será la encargada de devolvernos todas las preferencias de un cliente dado.
- /edit: utilizaremos esta llamada para modificar los datos como cliente, pero no la contraseña.
- /editPassword: utilizaremos esta llamada para editar solo la contraseña del cliente
- /register: llamada encargada de registrar un cliente.
- /login: esta llamada será para autenticarte como cliente.
- /loginMultiFactor: esta llamada será igual que la llamada de *login*, pero no devolverá un token.
- /insertMultiFactor: al igual que ocurre con la autenticación de usuarios, los clientes también tendrán un segundo factor de autenticación.
- /logout: llamada para cerrar la sesión en nuestro servicio.

Contamos con dos llamadas diferente para modificar los datos ya que la lógica del cambio de contraseña es compartida por ambas APIs. De esta manera podemos tener una interfaz y lógica más sencilla de cara a solo querer cambiar la contraseña. Este diseño ha sido diseñado de esta manera para que sea más fácil diseñar las aplicaciones para dispositivos inteligentes, así como para las distintas versiones web.

Para poder tener la API separada de la API de usuarios, tenemos que hacer el mismo procedimiento que hicimos con la API de usuarios. Agruparemos las llamadas y la versionaremos, para poder actualizar la API de cliente sin afectar a la versión anterior. Por tanto, crearemos un fichero, el cual contendrá todas las llamadas para la API de clientes y la agrupación junto al versionado será "/v1/clients". De esta manera la ruta "/" quedaría de la siguiente manera: https://IpServidor:443/v1/clients/. Como vemos queda una ruta de un directorio raíz, que será la encargada de devolver todos los clientes de nuestro servicio.

Como podemos observar, tenemos llamadas que son iguales a las llamadas que tenemos en la API de usuario. Estas llamadas cumplen el mismo propósito que en la API de usuarios, pero su implementación será diferente.

El caso más sencillo de controlador que encontramos en esta API es el controlador para la llamada "/", la cual recordemos, nos proporciona una lista con los nombres de todos los clientes. Esta llamada ha sido pensada para propósitos futuros, pero a su vez también nos ha servido para realizar comprobaciones internas. Es una llamada de tipo GET, que simplemente hace una llamada al servicio y devuelve el código de estado en la cabecera, así

como la cabecera de “Content-type” con valor “application/json” y la cabecera “Access-Control-Allow-Origin” con valor “*”. Todo ello acompañado con la lista de clientes en el cuerpo del mensaje de respuesta.

Las cabeceras que hemos indicado en este apartado sobre “Content-type” y “Access-Control-Allow-Origin” junto a código de estado de la petición HTTP, será añadidas en todas las respuestas de todos los controladores por cada petición que se envíe.

La siguiente llamada que le sigue a ésta, es la llamada de “/data”. En esta llamada lo que conseguiremos serán los parámetros necesarios para configurar a los usuarios de dicho cliente, es decir, conseguiremos parámetros como, por ejemplo, el tamaño mínimo de longitud de contraseña para sus usuarios.

Un cliente tiene los siguientes parámetros:

- Id: para distinguirlos en algunos casos internos.
- Nombre: nombre del cliente, en este caso nombre de la empresa.
- Email: email para la administración de la cuenta de cliente.
- *Sal* para los emails: cada cliente o empresa tendrá una *sal* diferente que se usará para los diferentes emails de sus usuarios.
- Contraseña: para la cuenta de administración del cliente o empresa.
- *Sal* de la contraseña de cliente: *sal* utilizada para la contraseña del cliente.
- Logo: URL con la imagen del logo de la empresa o cliente.
- Verificación: para saber si un cliente está verificado o no.
- Teléfono: número de teléfono de contacto del cliente.
- Tamaño de contraseña: indica el tamaño mínimo que debe cumplir la contraseña de los usuarios de dicho cliente.
- Lista de segundos factores de autentificación: se pueden elegir qué factores de la lista pueden ser utilizados por sus usuarios en el proceso de autenticación, para un cliente dado.

De todos estos parámetros que conforman un cliente, la llamada “/data” solo nos devolverá el nombre, la *sal* para los emails, la URL del logo, la verificación, el teléfono, el tamaño mínimo de contraseña para los usuarios y la lista de los segundos factores de autentificación que puede usar el usuario para autentificarse. Junto a toda la información que irá contenida

en el cuerpo del mensaje también irán las cabeceras correspondientes que hemos indicado en el apartado correspondiente.

Al igual que ocurría en la API de usuarios, disponemos dos llamadas de *login* con propósitos diferentes, recordemos que una devolvía en JWT y la otra solo autentificaba, sin esperar un valor para algún segundo factor de autentificación.

En los controladores de las llamadas "/regiter", "/login" y "/loginMultifactor" nos encontraremos al igual que ocurría en la API de usuarios, con tres controladores cuya lógica es muy parecida, ya que los tres siguen el mismo proceso con mínimas variaciones. Los tres controladores comienzan deserializando el contenido del cuerpo de la petición en un tipo de dato *cliente* creado por nosotros en el paquete de "models".

Tras tener el contenido del mensaje, cada uno llama a su servicio correspondiente, que es este punto el cual cambia en los tres controladores, puesto que cada uno llama a un servicio diferente y la respuesta del servicio es diferente. Por último, los tres controladores añaden las cabeceras nombradas en el apartado de cabeceras y junto al cuerpo de la respuesta del mensaje.

En este caso es el registro devolverá la contraseña en caso de que se haya generado una por petición del cliente o un mensaje, diciendo que el cliente ha sido registrado correctamente, en caso de que el cliente definiera su propia contraseña. En el caso del *login*, se devolverá el token y el momento del tiempo en el que expirará, recordemos que seguirá un formato basado en segundos de hora UTC y con formato Unix.

La llamada "/insertMultiFactor" en su controlador, es también muy similar a las otras tres llamadas, pero al igual que ocurría en la API de usuarios, después de deserializar el contenido del cuerpo en nuestros tipos de datos, antes de llamar al servicio encargado de la inserción del valor del segundo factor de autentificación, realizaremos una llamada al servicio encargado de realizar el loginMultifactor, para asegurar que el usuario que está haciendo la llamada está autenticado correctamente en el servicio.

Si todo el proceso de autenticación ha ido correctamente llamaremos al servicio de inserción del valor de segundo factor de autentificación. En este caso no contaremos con las cabeceras que teníamos en los demás apartados, ya que en este caso no se devuelve nada en el cuerpo de la respuesta y queremos que se pueda acceder a este recurso desde cual cualquier sitio.

Los controladores que quedan son los que requieren de un JWT para poder acceder a ellos. En este caso si no se cuenta con JWT valido, no se podrá acceder a estos recursos, aparecerán como dirección no valida o dirección no encontrada.

En el caso de las llamadas para la edición de datos. Son iguales los controladores, pero varía el servicio al que se llamada y el tipo de dato que llega al controlador. Recordemos que el uso de JWT es debido a que esta llamada pasará por el middleware que describimos en su apartado correspondiente, donde se valida el JWT y se extraen de la cabecera el email del cliente, así como su ID.

Junto con el ID y el email, en amabas llamadas de edición de datos, se llamará al servicio encargado de comprobar los valores de los segundos factores de autenticación. En caso de ir todo correctamente, se deserializará el cuerpo de la petición en la estructura de datos indicada y se llamará al servicio correspondiente. Al igual que ocurre con los demás controladores se añadirán las cabeceras definidas en el apartado de cabeceras y el cuerpo del mensaje de respuesta será en función de servicio al que hemos llamado.

Por último, en el caso de la llamada de *logout*, también necesitaremos el JWT, pero en este caso cuando la petición llegue al controlador, se llamará al servicio de *logout* para que invalide todos los valores de los métodos de segundo factor de autenticación del cliente e ID contenidos en el JWT, es por ese detalle que no se necesita enviar ningún tipo de contenido o cuerpo en la petición al servicio. La respuesta de este controlador para la petición será el código de estado enviado por el servicio para esa petición HTTP.

4.2.2 Servicios de la API de clientes

En este apartado al igual que ocurría en la API de los usuarios se mostrarán los diferentes funciones en el paquete de nombre “services” que hemos generado. En este paquete encontraremos las funciones que realizan la lógica de las peticiones a la API de clientes. Recordemos que estas funciones son las encargadas de acceder, insertar y actualizar los valores de nuestra base de datos.

Para la API de clientes hemos definido, al igual que ocurría en la otra API, varias funciones para cubrir los diferentes servicios que ofrecemos en ésta. Muchos de los servicios son parecidos a los que ya ofrecíamos en la API de usuarios, pero como hay varios cambios a nivel de base de datos, es por eso por lo que he decidido fraccionar y distinguir las funciones de servicios de clientes de las de usuarios, de esta forma, si en algún momento tenemos que

separar el servidor web, será mucho más sencillo el proceso. Aun así, encontraremos como hay funciones que se comparten por ambos servicios de ambas APIs.

Encontramos las siguientes funciones de servicios definidos para la API de los usuarios:

- GetClientService: esta función será la encardada de devolvernos una lista con todos los nombres de los clientes que tenemos en la base de datos.
- GetClientDataService: esta función será la encargada de devolvernos todos los parámetros para configurar los usuarios de dicho cliente.
- RegisterClientService: función encargada de dar de alta un nuevo cliente en el servicio.
- UpdateClientDataService: está función es la que encargará de actualizar los parámetros de configuración de usuarios de un cliente.
- LoginClientService: función encargada de autenticar a un cliente en el servicio y generarle un "Access Token" y devolvérselo al cliente.
- LoginMultiClientService: función encargada de la autenticación de un cliente, pero no genera un "Access Token".
- InsertMultiClientService: esta función es la encargada de insertar los valores de los métodos de segundo factor de autenticación en la base de datos para el cliente.

4.2.2.1 GetClientService

Como ya ocurría en la API para los usuarios, en esta lo primero que realizaremos en todas las funciones de servicio, será abrir la conexión con la base de datos y recordemos, que dicha conexión la cerraríamos más tarde con la palabra reservada "defer" del leguaje Go y su función correspondiente.

En esta función una vez tenemos la conexión abierta con la base de datos. Realizaremos una consulta en la base de datos en la que buscaremos el nombre de todos los clientes en la base de datos.

En el caso de que algo falle en la consulta, se devolverá una lista vacía junto un código de error, que en este caso va a ser el código de estado 500 "Internal Server Error". Sí, por el contrario, todo ha ido bien, se devolverá una lista con todos los nombres y el código de estado 200 "OK".

4.2.2.2 GetClientDataService

Al igual que ocurría en el caso anterior, esta función de servicio realizará una consulta de los parámetros que se utilizará para configurar a los usuarios de un cliente dado. Estos parámetros son los vistos en el apartado de controladores de esta API.

Esta función recibe tres parámetros que se utilizaran en la búsqueda de los datos cuando se realice la consulta a la base de datos. Estos parámetros de búsqueda son el email del cliente, el nombre del cliente o bien el ID del cliente. Podemos realizar la consulta a la base de datos con cualquiera de estos valores para poder conseguir los parámetros.

Comenzamos el servicio abriendo la conexión a la base de datos y lanzando la consulta para recibir todos los parámetros. Dicha consulta se realizará dependiendo de la condición que le haya llegado como parámetro a la función.

En caso de que vaya todo bien se devolverán todos los parámetros junto al código de estado 200 “OK” y en caso de que algo vaya mal en el proceso de consulta, como por ejemplo que no se encuentre el cliente indicado en los parámetros en la base de datos, se devolverá al controlador de la llamada, el código de estado 500 “Internal Server Error” junto a parámetros sin valor o vacíos.

4.2.2.3 RegisterClientService

Esta función es la encargada de registrar a un cliente en el servicio. La función recibe como parámetro un tipo de dato cliente creado por nosotros que está en el paquete de “models” de la API. En dicho tipo de dato, encontraremos todos los parámetros necesarios para poder dar de alta a un cliente nuevo en el servicio, así como los parámetros necesarios para configurar a los usuarios de dicho cliente, son estos los parámetros que obtendremos con el servicio que hemos visto en el apartado anterior.

Antes de abrir la conexión de la base de datos, se realizan comprobaciones de seguridad para saber si se tiene que seguir procesando la petición dentro de la función de servicio.

Las comprobaciones son: que el tamaño mínimo de contraseña para los usuarios sea mayor igual que ocho, al menos un número, al menos un carácter especial o símbolo, que el nombre del cliente no este vacío o sea de longitud cero (ya que como veremos el nombre es fundamental para registrar el cliente) y que el email que se utilizará para la administración de la cuenta de cliente tiene formato de email y que no sea vacío. En el caso de que no se

pase alguna de estas comprobaciones, se devolverá un código de estado 400 “Bad Request” y un mensaje de error para indicar que hay un error en el cuerpo de la petición, indicando que campo está mal y porque razón.

Si todo ha ido bien el siguiente paso es similar al que se sigue en el registro de usuarios. En este punto se comprueba si el cliente ha solicitado que se le genere una contraseña y con qué parámetros o si, por el contrario, el cliente ha indicado una contraseña para su cuenta de administración del servicio. En caso de que haya solicitado una contraseña se hará uso de del paquete “core”, que recordemos que era el núcleo de la API, donde teníamos funciones generadoras de datos para la API. Utilizamos la función definida para generar contraseñas que también usábamos para generar las contraseñas de los usuarios.

Una vez tengamos la contraseña, procederemos a abrir la conexión con la base de datos para poder hacer una inserción de los datos del nuevo cliente. Antes de hacer la inserción en la base de datos, la función genera varios parámetros para el cliente.

Se generarán dos sales, una para la contraseña y otra que se almacenará en la base de datos y servirá para utilizarla en el proceso de *hash* de los emails de los usuarios para dicho cliente, como ya hemos visto en el aparato de la API de usuarios.

También se generará un UUID para cada cliente con el paquete de Google llamado “Google/uuid” [28], este UUID será el ID del cliente, que será una de las formas de identificar al cliente.

Por supuesto antes de insertar todos los datos del cliente nuevo en la base de datos, se utilizará la función de Argon2 para realizar un *hash* del email y de la contraseña. Una vez que ya hayamos realizado todos estos pasos, se introducirá los valores del nuevo cliente en la base de datos.

Si todo el proceso de inserción de datos ha ido bien, empezará la parte realmente importante del registro de un cliente. En este punto, se crearán dos tablas por cada cliente nuevo en el servicio, estas tablas serán las encargadas de almacenar los usuarios para dicho cliente. De esta forma conseguimos que cada cuenta de usuario, para cada servicio, sea de manera individual y vaya en tablas diferentes, es decir que no se compartan cuentas del mismo usuario con los diferentes servicios, podremos tener el mismo usuario en diferentes tablas, de servicios diferentes y con resúmenes diferentes. De esta forma conseguiremos dos cosas, tener un mayor control de usuarios por cliente y otra, las más importante, que el robo de una

cuenta de usuario para un servicio, no expone el resto de las cuentas para ese usuario en el resto de los servicios.

Para generar las tablas usaremos el nombre del cliente, de manera que el nombre del cliente es único en la tabla de clientes del servicio. En una primera aproximación, como este servicio se está construyendo sobre SQL Server, utilizaremos los *schemas* [29] para poder agrupar las tablas por clientes. Los *schemas* en SQL Server funcionan a modo de contenedor o carpeta en la que se agrupan objetos lógicos y a los culés puedes poner los mismos permisos o restricciones.

En el caso de usar otro lenguaje de SQL, solo tendríamos que adaptar las consultas para que se amolden al lenguaje SQL en cuestión de creación de tablas y consultas, entraremos en mayor detalle en el apartado de mejoras futuras.

Si todo el proceso de creación de *schema* y tablas para el nuevo cliente ha ido correctamente, la función devolverá el código de estado 200 “OK” junto al mensaje “The client was registered” y en caso de haber solicitado una contraseña, también se le devolverá la contraseña creado para dicho cliente.

En caso de que algún proceso de inserción de datos o de creación de tablas haya ido mal durante el proceso, se realizará un *roll back* de todas las transacciones para dejar la base de datos como estaba antes y se devolverá el código de estado 500 “Internal Server Error” junto a un mensaje especificando cuál de los procesos ha fallado.

4.2.2.4 UpdateClientDataService

En este caso, es necesario poseer un JWT valido para poder acceder a esta función de servicio. Como parámetro recibirá los nuevos parámetros que se van a cambiar para la configuración de los usuarios de un cliente dado. Ese cliente es dado en forma de email e ID que aparece en el JWT del cliente.

Al igual que ocurría en el apartado anterior antes de abrir la conexión a la base de datos, realizaremos las mismas comprobaciones de los parámetros que en el apartado anterior. Es decir, se comprobará que el nombre no sea vacío y que la longitud mínima para la contraseña de los usuarios sea mayor igual a ocho, al menos contenga un número y un carácter especial o símbolo. Si todas las comprobaciones son correctas, abriremos la conexión a la base de datos, en caso contrario, procederemos del mismo modo, código de error y mensaje indicando que el contenido del cuerpo de la petición ha ido mal.

Lo primero que comprobaremos antes de actualizar nada, es si el cliente está intentando cambiar el nombre, ya que tendremos que hacer el proceso de creación de tablas con el nuevo nombre del cliente y realizar el proceso de traspaso de datos de las tablas viejas a las tablas nuevas.

Si el proceso de creación de tablas y traspaso va mal, no continuaremos con el proceso de actualización, volvemos a realizar un *roll back* para dejar la base de datos como estaba y devolveremos un código de error junto a un mensaje indicando que parte del proceso ha fallado.

En el caso de no querer actualizar el nombre o que el proceso de creación de tablas y traspaso haya ido correctamente. Se procederá con la transacción de actualización para actualizar los parámetros de configuración de los usuarios del cliente dado.

Si todo el proceso de actualización ha ido correctamente, se enviará un código de estado 200 "OK" junto al mensaje de confirmación.

4.2.2.5 LoginClientsService

Esta función es muy similar a la que podemos encontrar en la API de los usuarios, pero tiene alguna diferencia. Que hace que sea mejor separarlo en una función diferente a la que ya teníamos.

Uno de los cambios es que esta función recibe un tipo de dato cliente, que tiene diferentes campos que el tipo de dato usuario.

Comenzaremos abriendo la conexión a la base de datos. Utilizaremos Argon2 sobre el email del usuario que quiere realizar la autenticación y comprobaremos el segundo factor de autenticación, en caso de que no se disponga valores de un segundo factor de autenticación, esperará hasta dos minutos. En caso de que durante esos dos minutos no se introduzca ningún valor para los métodos de segundo factor de autentificación, se enviara un código de estado 401 "Unauthorized" junto a un mensaje de error indicando que tiene que usar algún segundo factor de autentificación.

El proceso de autenticación será igual que para que se hacía en el proceso de autenticación de usuarios, se consultará en la base de datos el resumen de la contraseña y la *sal* que se utilizó para dicho resumen. Con la contraseña que nos ha enviado el cliente, realizaremos el proceso de resumen con la función de Argon2, utilizando la *sal* que hemos recuperado de la

consulta. En caso de que el resumen que acabamos de crear y el que teníamos en la base de datos sean iguales en una comparación de bytes, habremos realizado el proceso de autenticación del cliente.

El siguiente paso es la creación del "Access Token" y es aquí donde hay otro cambio, puedes debemos de generar un JWT para el cliente, en lugar de para un usuario, es decir, el JWT deberá de ser firmado con la firma genérica para los clientes del servicio en lugar de usar la firma correspondiente para de ese cliente.

Si todo el proceso de autenticación ha funcionado correctamente, devolvernos un código de estado 200 "OK" junto al JWT y su hora de expiración.

4.2.2.6 LoginMultiClientService

En esta función de servicio, nos encontraremos un código muy similar al que teníamos en la API de usuarios.

Esta función recibe como parámetro un tipo de dato cliente. Al igual que ocurría en el resto de las funciones, lo que haremos en primer lugar es el abrir la conexión a la base de datos.

A continuación, realizaremos el mismo proceso que en el apartado de LoginClientService, pero con la salvedad de que no esperaremos ni comprobaremos que el cliente introduzca valores para algún método de segundo factor de autentificación.

En caso de que todo el proceso de autenticación vaya bien, devolvernos un código de estado 200 "OK" en la cabecera, pero no devolveremos nada en el cuerpo de la respuesta. En el caso de que algo vaya mal, devolveremos un código de estado 500 "Internal Server Error".

4.2.2.7 InsertMultiClientService

Función de servicio para insertar los valores del método de autentificación para un cliente. Es una función muy pequeña ya que lo único que contiene es abrir la conexión con la base de datos y una vez que disponemos de conexión, actualiza la tabla de segundo factor de autentificación con los valores que se le han pasado a la función como parámetro, para un cliente dado, también pasado como parámetro a la función.

Al igual que ocurría en el caso de la API de usuarios, si hemos llegado a este punto del procesado de la llamada, es porque ya hemos autenticado al cliente previamente.

En caso de que todo el proceso de actualización de los valores haya ido correctamente, devolveremos al igual que en el apartado anterior el código de estado 200 “OK” y en caso de que alguna parte del proceso no vaya como esperábamos, se devolverá un código de estado 500 “Internal Server Error”

4.3 Aplicación Android para los usuarios

La aplicación Android ha sido construida con Xamarin. Como hemos podido ver en el apartado de Xamarin, este IDE dispone de muchas ventajas de las que haremos uso en este apartado. La aplicación ha sido desarrollada utilizando el patrón de arquitectura de software MVVM(Model-View-ViewModel). Este patrón se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario con la lógica de la aplicación.

Lo primero que hará la aplicación al inicio, será intentar recuperar el “Access Token” que estará almacenado de forma segura en la memoria del dispositivo.

La manera de poder almacenar información segura en nuestros dispositivos es utilizando “Keychain services” [30] en iOS y “KeyStore class” [31] en Android, para asegurarnos que la información se almacena cifrada en disco.

Para lograr el almacenamiento seguro en memoria, utilizaremos el plugin “Xamarin Essentials”. Gracias a este plugin, podremos almacenar información sensible en el almacenamiento seguro del dispositivo, es decir de manera cifrada. Esto es posible gracias a que la API de “SecureStorage” dentro del plugin de “Xamarin Essentials”, hace uso de “Keychain” y “KeyStore” para almacenar información, de manera que no tengamos que implementar nada más, ya que Xamarin se encargará de traducir el código a los diferentes sistemas operativos, para poder utilizar las mejores aproximaciones en cada plataforma.

Todo lo que se almacena en el “Secure Storage” [32](Almacenamiento seguro) está encriptado en disco. En el caso de Android, además podemos indicar si queremos que esa información tenga una copia de seguridad, esto es así por si quieres que esta información se tenga en cuenta por ejemplo en nuevos dispositivos con tu cuenta. En nuestro caso, esa opción estará desactivada, ya que no estamos interesados en mover esa información entre dispositivos y tampoco en hacer una copia de seguridad de los mismos.

Utilizando este método, tendremos la información almacenada en el dispositivo y esta información no podrá ser robada.

Volviendo al principio, si tenemos almacenado un “Access Token”, cuando se inicie la aplicación, se recuperará, en caso de no recuperar el token, se navegará automáticamente a la pantalla de *login*, que será nuestra pantalla principal.

Si hemos podido recuperar el “Access Token”, lo que siguiente que comprobaremos, será que el token no haya agotado su tiempo de vida. Si hubiera agotado su tiempo de vida, otra vez navegaríamos a la pantalla principal.

En caso de ser un “Access Token” valido, intentaremos recuperar las credenciales del usuario, que recordemos, están almacenadas en una parte segura del dispositivo. Si no conseguimos recuperar las credenciales, navegaremos a la página principal.

En caso de haber recuperado correctamente las credenciales, procederemos a realizar el *login* contra el servicio de autenticación. En caso afirmativo navegaremos a la página del segundo factor de autenticación. En caso de no ser una autenticación valida, navegaremos a la página principal para que el usuario vuelva a introducir sus credenciales.

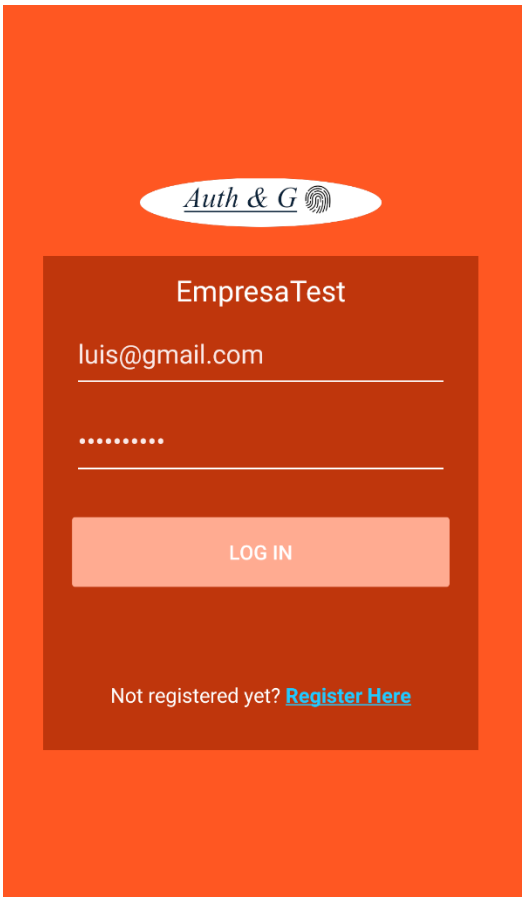

Figura 5 Página principal de la aplicación de usuarios

En todos los procesos en los cuales hemos navegado a la página principal, es decir, la página de *login*, se han eliminado todos los datos del usuario que estaban almacenados de manera segura en el dispositivo.

Todo este proceso de comprobación también se volverá a realizar siempre que se vuelva a la aplicación desde un estado en el que la aplicación estaba trabajando en un segundo plano, es decir no era la aplicación activa en ese momento, o si se vuelve de un estado en el cual el dispositivo estaba bloqueado a un estado de desbloqueo con nuestra aplicación como aplicación activa. En resumen, siempre que se acceda a la aplicación y esta no sea la aplicación activa, se realizará la comprobación que se ha detallado arriba.

4.3.1 Registro de un usuario a través de la de aplicación

Para acceder a esta pantalla, podremos hacerlo desde la pantalla principal, indicando que te quieres registrar en el servicio.

Un usuario se podrá registrar en un servicio para posteriormente hacer su autenticación. La aplicación le permitirá introducir el email con el que desea ser identificado y posteriormente para la contraseña dispone de dos opciones.

La primera opción, es que el usuario introduzca una contraseña valida cumpliendo los requisitos que dicha contraseña tenga que cumplir, como por ejemplo que tenga una longitud mínima de ocho caracteres. Si después de enviar la petición a la API para que registre un usuario, ha ido todo bien, el usuario pasará a estar registrado en el sistema y la aplicación continuará su flujo normal.

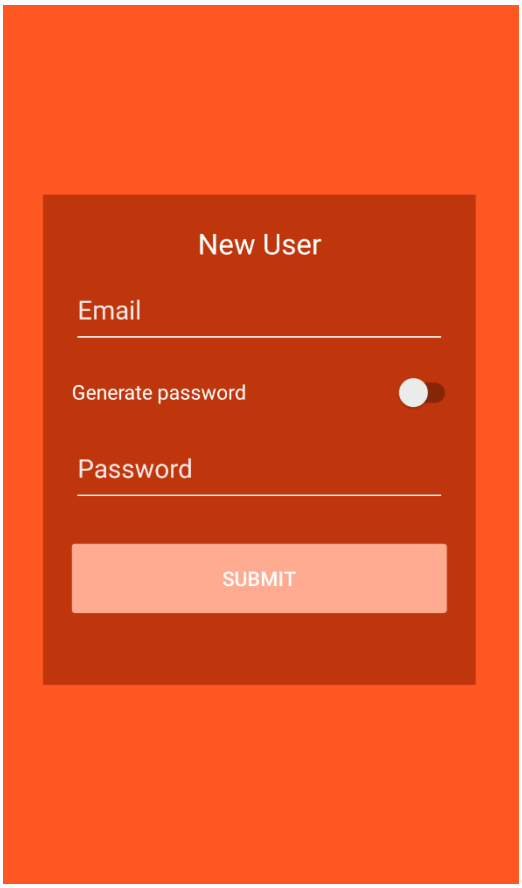

Figura 6 Página de registro de un usuario opción 1

La segunda opción que tiene el usuario de la API es la de marcar el *switch* de autogeneración de contraseña. Cuando el usuario marca la opción de autogeneración de contraseña, desaparecerá la opción de introducción de contraseña, pero aparecerán nuevas opciones, estas opciones son: longitud de contraseña que se desea crear(recordemos que por defecto el tamaño mínimo será ocho), si se desea la contraseña con mayúsculas, con números o con símbolos especiales, la opción de minúsculas no está presente ya que por defecto, como mínimo la contraseña será de longitud por defecto y caracteres en minúscula por defecto, junto a un número y un carácter especial o símbolo.

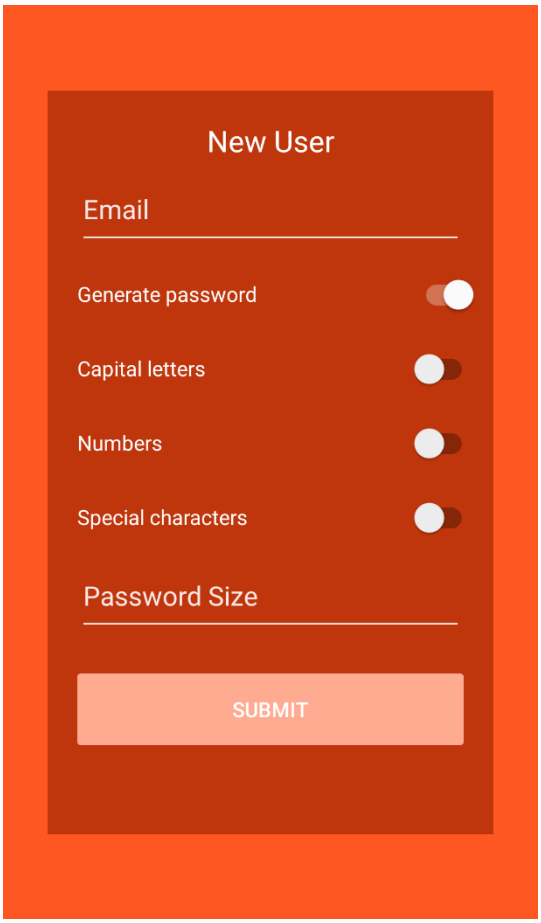

Figura 7 Página de registro de un usuario opción 2

Tras la selección de las preferencias del usuario para la contraseña, la petición será de nuevo enviada a la API para registrar al usuario, en caso de que la petición haya procedido correctamente en la API, el usuario pasará a estar registrado y se le mostrará en pantalla la contraseña que debe de memorizar o almacenar en un gestor de contraseñas, pues esa será su contraseña para autenticarse en el servicio.

En ambas opciones, si el registro no ha ido bien en la API tras la petición, se tendrá que volver a realizar el registro introduciendo los datos nuevamente y volviendo a enviar la petición.

La opción que haya elegido el usuario no influye en el resultado final, ya que, si el registro en ambas opciones ha ido correctamente, ocurrirá lo mismo para ambas, la aplicación seguirá su flujo normal y realizará la autenticación con las nuevas credenciales creadas para el usuario. Si el proceso de autenticación ha funcionado correctamente, el flujo de la aplicación seguirá a la pantalla en la que se pedirá el segundo factor de autenticación.

Si, por lo contrario, fallará por cualquier razón ajena al usuario, ya que todo este proceso es transparente para él, se le enviaría a la página principal, la cual es la página de *login*, donde se introducen las credenciales para el proceso de autenticación y donde puede navegar de nuevo a la pantalla de registro.

4.3.2 Autenticación a través de la aplicación.

La aplicación cuenta con una pantalla de *login*, en esta pantalla encontraremos los elementos necesarios para poder introducir las credenciales del usuario.

Esta pantalla será nuestra pantalla principal, por lo tanto, siempre que ocurra un error o siempre que las credenciales del usuario en la aplicación no estén disponibles o el token haya expirado está será la pantalla que se muestre ya que será necesario volver autenticar el usuario.

Cuando el usuario ha introducido las credenciales, se hace uso de una clase de servicio, la cual es la encargada de hacer todas las peticiones a la API. Esta clase es usada a lo largo de la aplicación, en todos los procesos que involucren una petición a la API.

Si todo ha ido bien el proceso de *login*, el flujo de la aplicación continuará hacia la petición del segundo factor de autenticación del usuario. En caso de error, se tendrán que volver a introducir las credenciales para volver a enviar una petición a la API.

4.3.2.1 Servicios de la aplicación.

Tendremos una clase, la cual he llamado “LoginService”, la cual será la encargada de hacer todas las peticiones a la API para poder autenticar o registrar a un usuario en el servicio.

La clase cuenta con diferentes funciones para las diferentes tareas que tiene que afrontar la aplicación, pero todas tiene una base común a la hora de realizar la petición a la API.

Todas las peticiones envían la petición a treves del envío de peticiones con el que cuanta C#, estos envíos se realizan indicando que se va a utilizar el protocolo TSL 1.2, se utiliza esta versión ya que la versión actual del paquete de C# encargada de realizar las peticiones HTTP no dispone de la nueva versión del protocolo.

Además de indicar el tipo de protocolo que vamos a usar en las peticiones, también indicaremos en la cabecera que el contenido del cuerpo de la petición estará en formato JSON y codificado en UTF8.

Una vez tenemos todo eso indicado, realizaremos una petición a la API de manera asíncrona, la cual nos devolverá el resultado en función de la petición enviada.

En caso de una petición de *login* normal contra la API solo se enviará la petición y se esperará una respuesta que podrá ser afirmativa o negativa, como ya hemos visto en el apartado de desarrollo de la API. En caso de un *login* con un valor correcto del segundo factor de autenticación insertado, la API devolverá el token y el tiempo de expiración, como estos datos solo serán devueltos en caso de una autenticación correcta, en este punto del servicio es donde se almacenarán de manera segura tanto las credenciales del usuario, como el "Access Token" y el tiempo de expiración de este.

Por último, en la inserción del valor de segundo factor de autenticación, se intentará recuperar las credenciales del usuario del almacenamiento seguro, para poder enviarlas junto con el valor a insertar.

4.3.2.2 Segundo factor de autenticación

El caso de lectores biométricos para la autentificación en los smartphones o dispositivos inteligentes es curioso, puesto que el sistema operativo no da información del usuario a la aplicación, si no que actúa como una caja negra en la que es la aplicación la encargada de preguntar al sistema operativo, si la persona que está usando el segundo factor de autenticación es quién dice ser.

En el caso de iOS, para el uso de "Face ID" o el "Touch ID" tendremos que usar el *framework* "LocalAuthentication" para poder usar esos mecanismos de autenticación en tu aplicación. Este *framework* para mayor seguridad no te dará información del usuario y será él, el encargado de realizar todo el proceso de autenticación.

En el caso de Android, el proceso es similar, pero Android tiene una particularidad que iOS no tiene, los fabricantes de móviles con sistema operativo Android, han implementado diferentes lectores biométricos en diferentes dispositivos y marcas. Por lo que por un lado tendremos dispositivos solo con lector de huella, otros dispositivos con lector de iris, otros con reconocimiento facial en 3D…

Android hasta la versión 9.0, ha ido mantenido solo una clase para poder utilizar el lector de huellas de los dispositivos junto a la clase "FingerprintManager" [33], los demás lectores biométricos, han sido manejados por el fabricante del dispositivo que ha añadido dicho lector al dispositivo. Por ejemplo, en el caso de Samsung, varios de sus dispositivos cuentan con

lector de iris incorporado, además es de las pocas compañías que cuentan con lector de iris en sus dispositivos, en su caso ninguna aplicación fuera de las aplicaciones de Samsung pudo utilizar el lector de iris hasta que pasado un tiempo abrieron un SDK para poder usurarlo el resto de aplicaciones, pero son pocas las que se han atrevido a intentar implementarlo ya que es solo una característica disponible para dispositivos muy concretos.

En los últimos años hemos visto un incremento en la preocupación en la seguridad de nuestra información por parte de los usuarios, los fabricantes de dispositivos han querido aprovechar la situación y cada vez tenemos más lectores biométricos en nuestros dispositivos. Es por ello por lo que a partir de la versión 9.0 de Android, la clase "FingerprintManager" entra en desuso y se empezará a utilizar la clase "BiometricPromt" [34], que proporciona comunicación con todos los lectores biométricos del dispositivo, dando así la posibilidad de que todas las aplicaciones puedan hacer uso de todos los sistemas biométricos del mercado que utilicen el sistema operativo Android. Aunque podamos utilizar más sistemas biométricos, estos seguirán siendo una caja negra para nosotros.

La versión 9.0 de Android fue lanzada al mercado en agosto de 2018, eso quiere decir que, a la fecha de comienzo de este proyecto, los dispositivos que cuentan con dicha versión son inferior al 1% de todos los dispositivos Android del mercado. Es por ello, que para el desarrollo de este proyecto se utilizarán versiones inferiores para el desarrollo, esto implica que de momento no podemos usar la clase "BiometricPromt". Esto se podrá mejorar con el tiempo con actualizaciones de la aplicación, para este proyecto se usará como requisito mínimo el uso de lector de huellas como segundo factor de autenticación y para ello utilizaremos la clase "FingerprintManager".

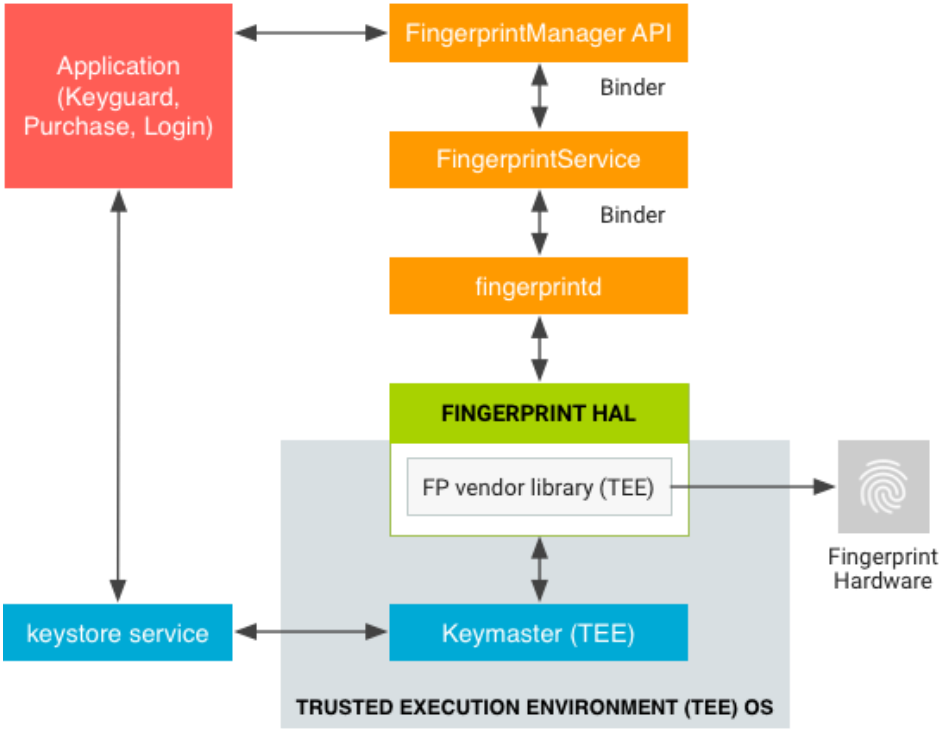

Figura 8 Diagrama de flujo para el uso de sensores biometrícos en Android

El diagrama es igual para la versión 9.0, solo cambiaríamos “FingerprintManager” por “BiometricPromt”, véase [35].

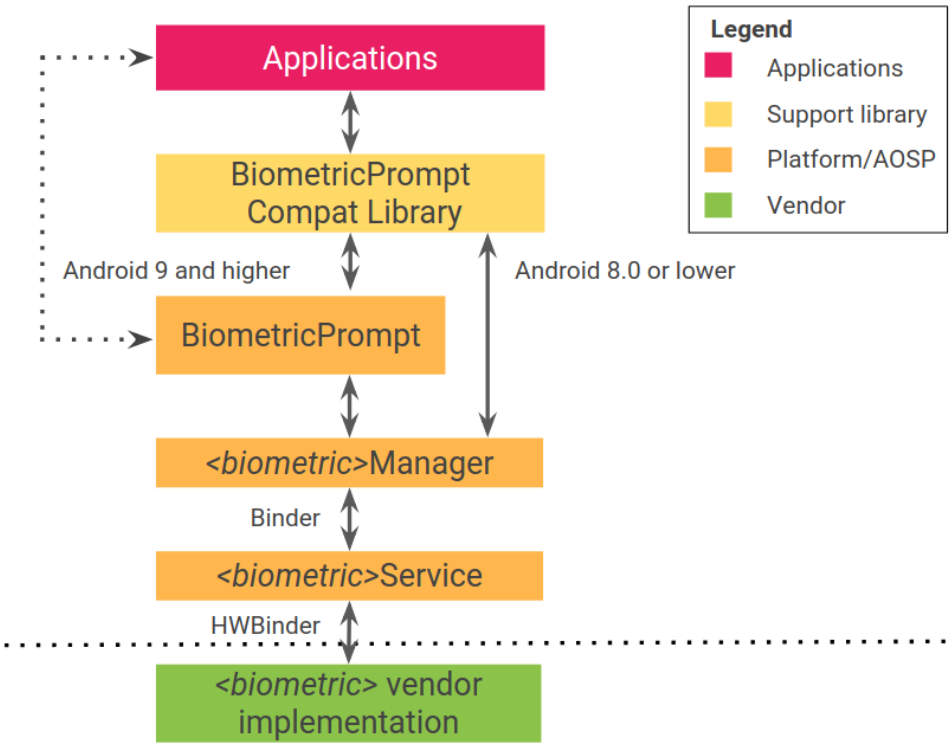

Figura 9 Arquitectura de "BiometricPrompt" para la versión 9.0 o superior [46]

En el caso de Xamarin, no es capaz de implementar estas clases de manera nativa. Este será el caso que comentamos en el apartado de Xamarin, en el cual deberíamos de implementar código de manera nativa en cada sistema, es decir tendremos que hacer uso de los diferentes sistemas biométricos en sus sistemas nativos.

Para lograr este propósito, deberemos de hacer uso de "DependencyService" [36], que permite a las aplicaciones llamar a funciones específicas de una plataforma desde el código compartido. Esta funcionalidad permite que las aplicaciones de "Xamarin.Forms" hagan todo aquello que puede hacer una aplicación nativa. "DependencyService" es un localizador de servicios, este localizador, en la práctica hace que tengamos que crear una interfaz que implementará cada plataforma según su sistema y en tiempo de ejecución "DependecyService" busca la implementación correcta de una interfaz en los diferentes proyectos de cada plataforma.

Para el correcto funcionamiento de “DependencyService” tendremos que definir:

- Interfaz: una interfaz que defina las funciones requeridas por el código compartido.
- Implementación en cada plataforma: cada plataforma deberá de disponer de una clase que implemente la interfaz.
- Registro: cada clase implementadora de la interfaz deberá de registrarse en el “DependencyService” a través de metadatos. Esto es necesario para la resolución en tiempo de ejecución.
- Uso de “DependencyService”: En el código compartido se debe de llamar explícitamente al “DependencyService” cuando desee usar una de las funcionalidades.

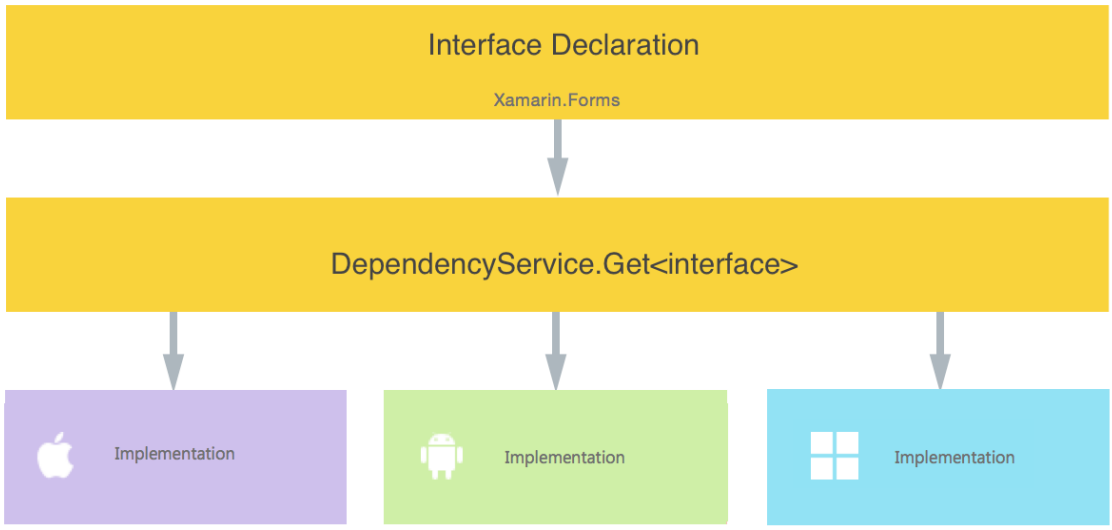

Figura 10 Diagrama de uso para “DependencyService”

Si no tenemos una clase que implemente cada interfaz en cada plataforma, en tiempo de ejecución de dicha plataforma nos dará error cuando intente resolver la dependencia de la clase.

Bien, una vez visto cómo hacer uso de implementaciones específicas de cada plataforma, lo utilizaremos para nuestro propósito. En el proyecto de código compartido de Xamarin definiremos una interfaz, con el propósito de implementar dicha interfaz en el proyecto de Android, esta interfaz solo tendrá un método para poder llamar a “FingerprintManager”.

La implementación en Android lo primero que hará es realizar una serie de permisos para comprobar que todo el proceso del lector de huellas funciona correctamente. El orden de comprobaciones es el siguiente:

1. Comprobar que la aplicación tiene permisos para usar el lector de huellas.
2. Comprobar que el dispositivo posee lector de huellas.
3. Comprobar que tiene huellas guardadas para la comprobación del lector, es decir que usa el lector de huellas.
4. Que tienes seguridad en tu pantalla de bloqueo, es decir que tiene una pantalla de bloqueo en el dispositivo y no puedes acceder directamente a él cuándo pulsas un botón.

Si todas las comprobaciones han ido bien, el dispositivo comenzará a escuchar el lector de huellas, esperando a que el usuario coloque su dedo para hacer la comprobación pertinente.

En caso de que alguna comprobación falle, se le mostrara un mensaje de error al usuario para que solucione todas las posibles causas. Por ejemplo, en caso de no disponer de una huella guardada, el mensaje será el siguiente: “You need to enroll at least one fingerprint with the device”

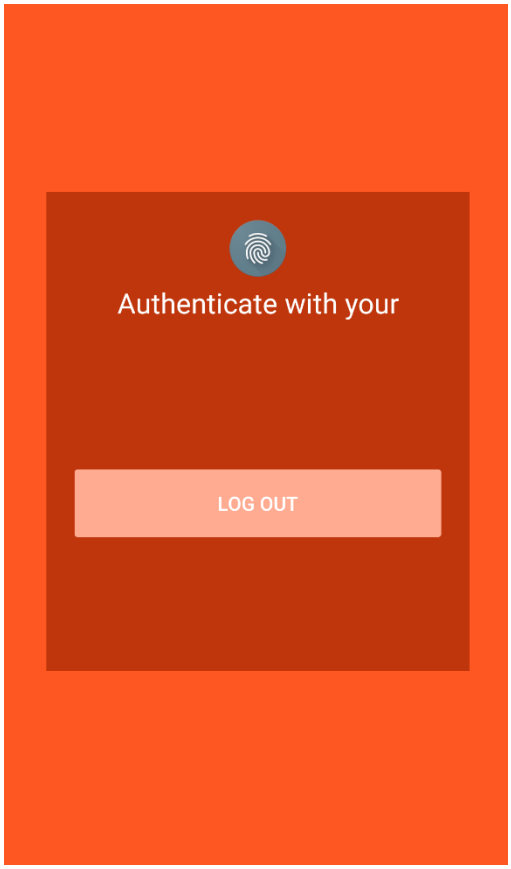

Figura 11 Página de segundo factor de autenticación

Cuando el usuario ponga su dedo sobre el lector de huellas, pueden ocurrir tres situaciones y para cada situación, debemos de escribir cual va a ser el flujo de la aplicación.

El primer caso es que el usuario haya puesto el dedo correcto, la lectura haya sido correcta y que la autenticación haya sido correcta. En ese caso nos encontramos que el usuario es quién dice ser con el segundo factor de autenticación. En ese caso realizaremos la llamada a la API para que inserte el valor del lector de hullas en la base de datos para el usuario en el servicio, después de esa llamada realizaremos otra más a la API para que nos autentifique y nos devuelva el token junto a su tiempo de vida. Si todos los procesos han ido correctamente, el flujo de la aplicación continua y te lleva a una página final en la que se muestra información al usuario y un mensaje de autenticación correcta. En caso de que algún proceso haya ido mal se volverá a pedir al usuario que coloque su dedo sobre el lector.

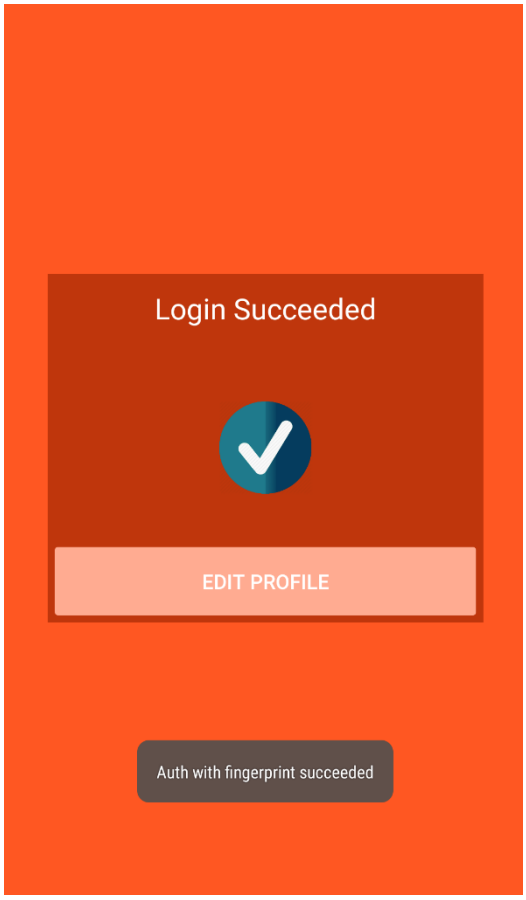

Figura 12 Página de autenticación correcta

La siguiente situación es la de fallo en el lector de huellas. Esta situación produce cuando el usuario ha colocado un dedo erróneo o en el caso que por alguna razón el lector no ha podido validar la huella. En este caso se le mostrará al usuario un mensaje para que lo vuelva a intentar y el lector volverá a ponerse en modo escucha para leer la huella de nuevo.

La última situación se produce tras haber fallado el lector de huellas tres veces, es decir tras cinco intentos si el lector no ha conseguido leer la huella correctamente o validar al usuario correctamente, se pasa a esta situación de error. Este caso es el más radical, puesto que una vez llegamos a este punto, el flujo de la aplicación será el de borrar todos los datos almacenados en la zona segura del dispositivo y volver a la pantalla principal de *login* para que el usuario tenga que volver a introducir de nuevo sus credenciales y volver a empezar de este modo el flujo de la aplicación.

4.3.3 Edición de datos del usuario

En una primera iteración de desarrollo del servicio para los usuarios, solo podremos editar la contraseña como tal.

Para el proceso de edición de contraseña, una vez que la autenticación haya finalizado, el flujo de la aplicación hará que naveguemos hacia una pantalla de información del usuario. En la pantalla de información podremos encontrar un botón para navegar a la vista donde podremos editar la contraseña.

Para editar la contraseña, debemos tener en cuenta que para hacer lo más seguro posible este proceso, no se mostrará nada de lo que tenemos almacenada del usuario en la base de datos, es por ello que solicitaremos la contraseña antigua, que será utilizada para comprobar que se conoce la información que tenemos almacenada en la base de datos, también se pedirá que se introduzcan dos veces la nueva contraseña, esto es debido a que necesitamos estar seguros de que el usuario es conocedor de la nueva contraseña y que no va haber inconvenientes posteriormente.

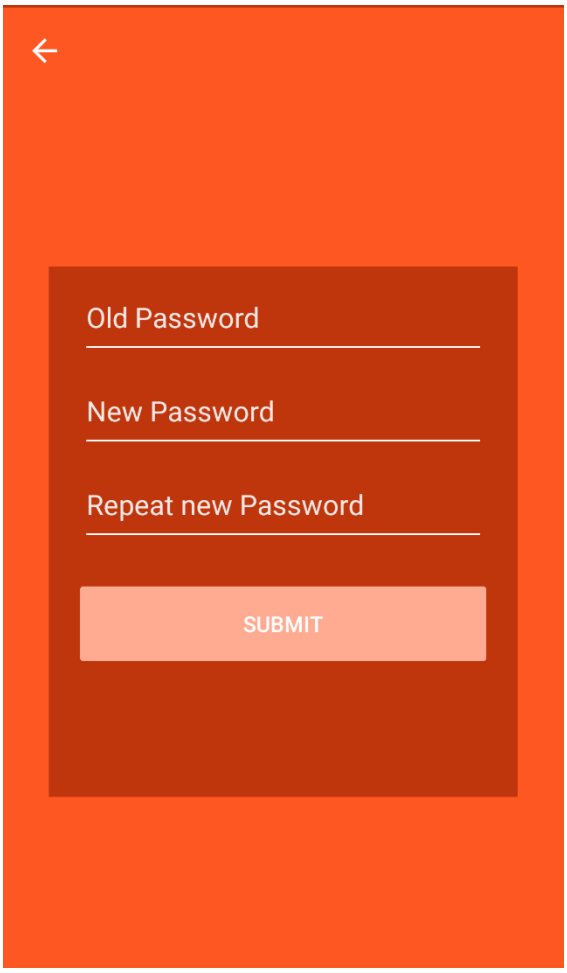

Figura 13 Página de cambio de contraseña

Si las nuevas contraseñas no coinciden o no cumplen con los parámetros de configuración del cliente al que pertenece dicho usuario, se mostrará un mensaje de error y el usuario deberá de volver a introducir los valores nuevamente.

Si todos los datos están correctos, se procederá a enviar la petición a la API, en caso de error por parte de la API se mostrará un mensaje de error avisando al usuario que no ha sido posible actualizar la contraseña y que por lo tanto deberá de intentarlo de nuevo. En caso correcto, se mostrará un mensaje indicando que se ha actualizado la contraseña y que por lo tanto deberá de volver a proceder con el proceso de autenticación. Esto provocará que se vuelva otra vez a la pantalla de inicio, para que vuelva a comenzar el proceso de autenticación.

4.4 Aplicación Android para los clientes

En este apartado se mostrará cual es la lógica de la aplicación para los clientes. Esta aplicación, se ha desarrollado en base a la anterior, por lo que al igual que ocurría en la aplicación para los usuarios, esta aplicación también está diseñada utilizando Xamarin.

También al igual que ocurría con la aplicación para usuarios, esta aplicación también ha sido desarrollada utilizando el patrón de arquitectura software MVVM(Model-View-ViewModel), recordemos que este patrón se caracteriza por la tratar de desacoplar al máximo la interfaz gráfica del software con la lógica de la aplicación.

La aplicación para los clientes será la que no será personalizable, ya que ésta es la aplicación para que nuestros clientes puedan utilizar nuestro servicio. En cambio, la aplicación de los usuarios será personalizable por los clientes, para adaptarla y amoldarla a su corporación, como por ejemplo adaptar la aplicación con sus colores corporativos.

Otra particularidad de la aplicación para los clientes es que ha sido desarrollada después de la aplicación para los usuarios, por lo que comparten alguna lógica con respeto a la autenticación de clientes.

En una primera aproximación, lo primero que hará la aplicación al inicio, será intentar recuperar el "Access Token" del cliente que estará almacenado de forma segura en la memoria del dispositivo.

En esta aplicación de clientes nuestra página principal también será la página de *login*, cuando nos refiramos a dicha página, ésta será en la que el cliente debe de ingresar sus credenciales para poder proceder al proceso de autenticación.

Si se ha recuperado el "Access Token" del almacenamiento seguro del dispositivo, se comprobará que el ese token aún tenga validez, es decir, que no haya caducado en el tiempo. Si el "Access Token" es válido, se procederá a intentar recuperar también del almacenamiento seguro, el email y la contraseña de la cuenta de administración del cliente. Si todos los procesos de recuperación de los datos han ido correctamente, se procederá al proceso de autenticación, que veremos más adelante en otro apartado.

En caso de que alguno de los procesos falle, se eliminará toda la información que se haya almacenado de algún cliente en el almacenamiento seguro del dispositivo y se navegará hasta

la pantalla principal para que el cliente vuelva a ingresar sus credenciales, de esta manera volveremos a comenzar el flujo de autenticación.

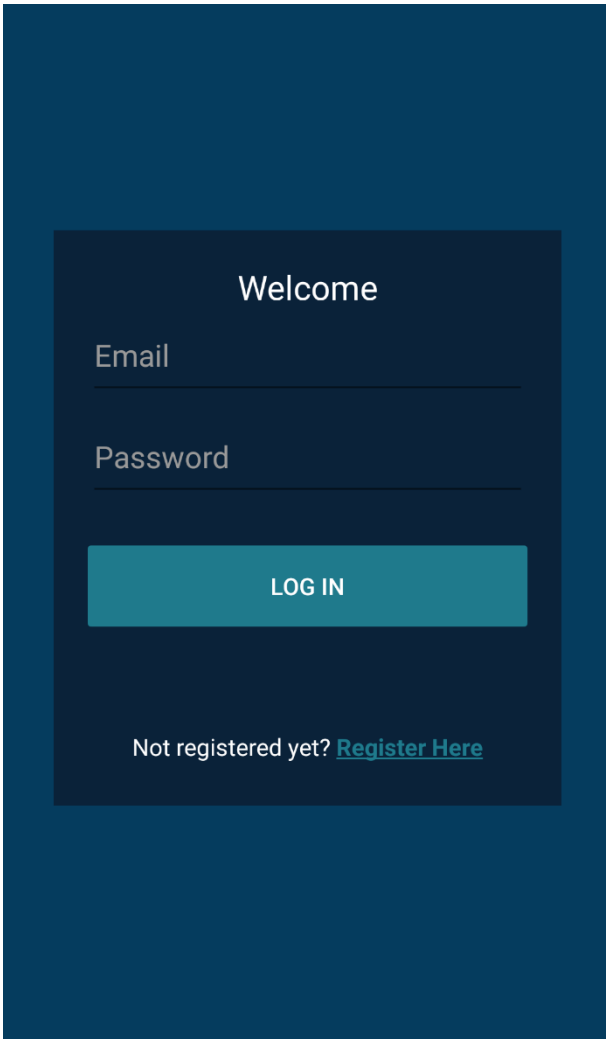

Figura 14 Página principal de la aplicación de clientes

4.4.1 Registro de un cliente en la aplicación Android

El registro de un cliente es diferente al registro de un usuario. En primer lugar, para acceder a esta pantalla, podremos hacerlo desde la pantalla principal, indicando que te quieres registrar en el servicio.

La aplicación en este punto te dará muchas más opciones de las que le daba al usuario, pero al igual que ocurría en la aplicación de los usuarios, el cliente también puede decidir durante el registro entre las dos opciones para la contraseña, la opción de ingresar su propia contraseña o la opción de que sea el servicio el que le generé una contraseña basada en sus preferencias.

Si el usuario decide solicitar una contraseña al servicio, vimos en el apartado de la API de cliente que la generación de contraseñas para los clientes usaba la misma función generadora de contraseñas que se usaba para los usuarios, por lo que la aplicación otorgará las mismas opciones para la creación de contraseñas: longitud de la contraseña, mayúsculas, números y símbolos. Al igual que ocurría en la aplicación de clientes, por defecto el tamaño mínimo será ocho y minúsculas con un número y un carácter especial, en caso de no indicar ninguna opción más

Además de indicar el email y como desea la contraseña, el cliente deberá de ingresar el nombre de su empresa/servicio, la longitud que desea de contraseña mínima para sus usuarios, número de teléfono, una URL con la imagen de su empresa/servicio y seleccionar de la lista de métodos de segundo factor de autentificación, cuáles serán los que podrán usar sus usuarios para autenticarse.

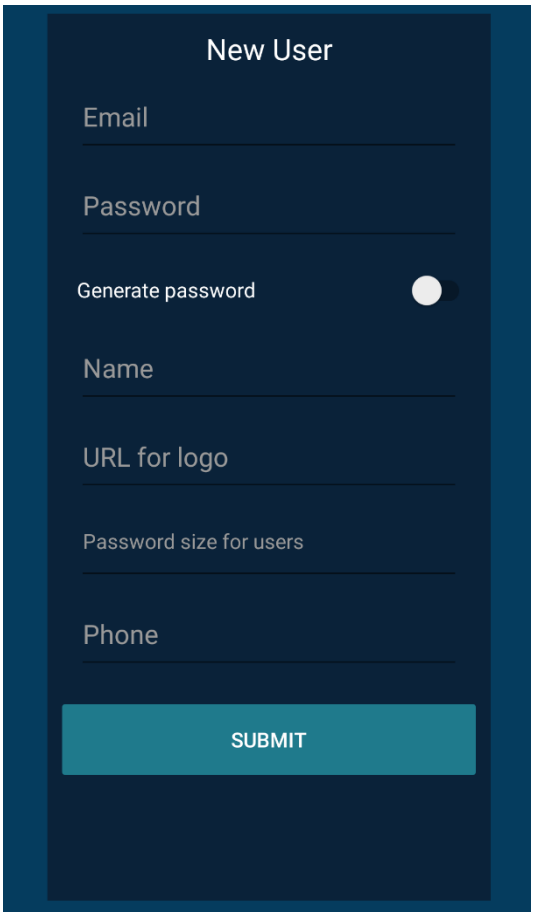

Figura 15 Página para el registro de un cliente opción 1

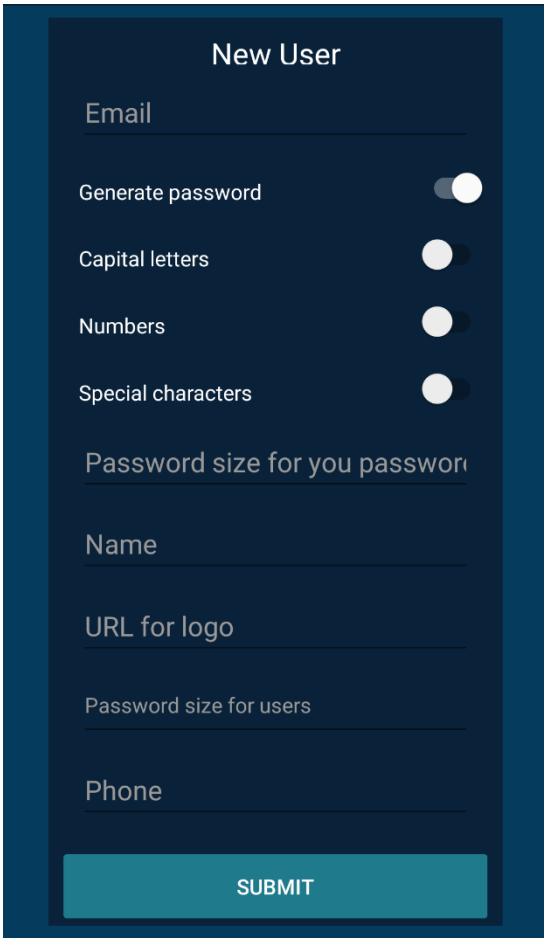

Figura 16 Página para el registro de un cliente opción 2

Con todos estos datos, se procederá a utilizar al igual que ocurría en la aplicación de usuarios, una clase de servicio, que será la encargada de realizar todas las peticiones a la API de clientes utilizando todas las utilidades que el lenguaje C# nos proporciona.

Si todo el proceso de registro ha ido correctamente, pueden ocurrir dos cosas, que se muestre un mensaje con la contraseña que el cliente ha solicitado, de manera que el cliente deberá de memorizar o almacenar en un gestor de contraseñas, o en caso de no haber solicitado una contraseña, se mostrará un mensaje en el que indicará que el usuario ha sido registrado correctamente. Si todo ha ido bien, el flujo de la aplicación consistirá en navegar al segundo factor de autenticación para que el cliente pueda terminar de autenticarse. En caso de que alguna parte del proceso haya salido mal se volverá a solicitar al cliente que vuelva a introducir todos los datos y vuelva a intentarlo, en caso de que esto ocurra, esta situación irá acompañada de un mensaje que dirá porque ha fallado el proceso de registro.

4.4.2 Autenticación de cliente con la aplicación

El proceso de autenticación de la aplicación de clientes es el mismo que hemos visto en el proceso de autenticación en la aplicación para usuarios.

La única diferencia es que, en el caso de la aplicación de usuarios, durante todos los procesos de petición a la API, debíamos de indicar el ID del cliente contra el que queríamos hacer el proceso de autenticación. En este caso, no hace falta especificar ningún ID, ya que todo el proceso será contra el servicio autenticación.

A la hora de tratar con el segundo factor de autenticación, utilizaremos de la misma manera "DependencyService", que recordemos, es un localizador de servicio, el cual resolverá la dependencia en tiempo de ejecución de la implementación de una interfaz, según en la plataforma que estemos ejecutando la aplicación en ese momento.

Al igual que ocurría en la aplicación para los usuarios, en estos momentos y debido a las versiones de Android, solo vamos a utilizar la aplicación de Android con el lector de huellas dactilar. En este caso el procedimiento será el mismo. Definiremos una interfaz que será implantada en el proyecto de Android y que dará implementación al uso del lector de huellas, al igual que vimos en el apartado de la aplicación para los usuarios.

Todo lo demás, como el flujo de la aplicación, así como los errores durante el proceso de autenticación y demás, es el mismo proceso que podemos ver en la aplicación de los usuarios.

4.4.3 Edición de datos del cliente.

Tras el proceso de autenticación y en caso de todo haya ido correctamente, el flujo de la aplicación nos hará navegar a la pantalla de información para el cliente.

En esta ocasión, en la pantalla de información del cliente, nos encontraremos dos botones, uno para cambiar la contraseña al igual que ocurría en la aplicación para los usuarios, pero además nos encontraremos con otro botón para poder editar los parámetros con los que se configuran los usuarios del cliente actual.

Si pulsamos sobre el botón de cambiar contraseña, navegaremos a una página en la que deberemos de ingresar la contraseña actual y escribir dos veces la contraseña nueva que deseamos para nuestra autenticación. Tras completar de rellenar los campos necesarios, se procederá a enviar una petición a la API para que gestione el cambio. En el caso de haber

ingresado mal las contraseñas nuevas, como por ejemplo porque no coinciden o porque la contraseña nueva no cumple los riquitos mínimos de contraseña para el servicio, se mostrará un mensaje para que se cambien los valores erróneos. En caso de que la petición haya ido mal en la parte de la API, también se mostrará un mensaje junto al problema que ha ocurrido y se le recordará al cliente que no se ha podido cambiar la contraseña y que por lo tanto sigue teniendo que utilizar la misma contraseña que tenía o volver a intentar cambiarla.

Si todo el proceso de cambio de contraseña ha ido correctamente en todas sus partes, tanto en la petición como en la introducción de los valores, de avisará al cliente de que debe de volver a realizar el proceso de autenticación, volviendo a navegar a la página principal de la aplicación.

Si pulsamos sobre el botón de editar datos del cliente, navegaremos a una vista en la que se cargará los datos de configuración de dicho cliente para que pueda visualizarlos y editarlos a su gusto. Al igual que ocurría en el caso de la edición de contraseñas, si alguno de los campos no cumple la condición mínima para los parámetros, como por ejemplo que el teléfono sea de nueve dígitos, se mostrará un mensaje por pantalla que te dirá cuál de los campos de los parámetros no cumple con las especificaciones y por qué.

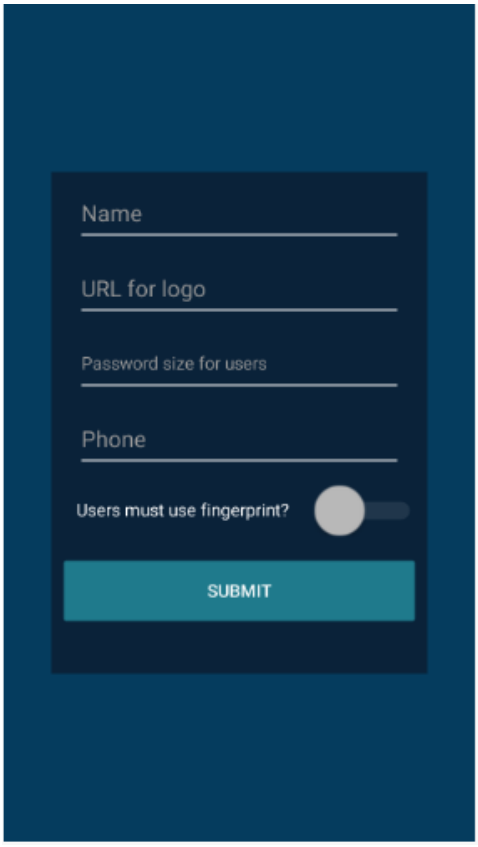

Figura 17 Página para la edición de datos del cliente

Si todo el proceso de edición de los campos ha ido correctamente en la petición a la API, se mostrará un mensaje al cliente diciendo que todo ha ido correctamente y en caso contrario, deberá de volver a proceder con la edición de los datos y volver a intentarlo, también se le avisará de que los cambios no se han cambiado y que los usuarios seguirán configurándose como lo estaban haciendo hasta ahora.

Recordemos que los campos que puede editar en este apartado serán: el nombre de la empresa/servicio, la longitud mínima de contraseña para los usuarios del servicio, la URL del logo del servicio, el número de teléfono y seleccionar de una lista los métodos que queremos que los usuarios tengan disponibles para el segundo factor de autenticación.

4.5 Página web para los usuarios

Para poder tener el servicio lo más parecido a un entorno real y con el que poder mostrar el servicio. Vamos a registrar un cliente en el servicio el cual disponga en este caso de una página web.

En esta página web utilizaremos el servicio para poder realizar la autenticación de los usuarios en la web.

La página web no voy a entrar en profundidad a desarrollarla, ya que no es lo que comprende el proyecto ni el servicio, tendremos de esta forma tres vistas o páginas.

La primera de las vistas o pantalla será la página principal y publica, a la que todo el mundo desde la web puede acceder.

La segunda pantalla o vista será el *login* de la página, es decir donde el usuario indicará las credenciales para realizar la autenticación.

Por último, tendremos la página o vista privada, en la cual solo los usuarios autenticados podrán acceder a esa área de la web.

Tanto la primera vista principal como la vista privada serán páginas bastante básicas. Utilizaremos HTML básico junto al *framework* Bootstrap, el cual nos proporcionará una plantilla para el diseño de la página.

4.5.1 Página de autenticación del usuario.

El diseño de la página es un poco básico también, pero lo que nos interesa en este punto es como realizar la autenticación contra nuestro servicio.

En la página podremos encontrar un input de tipo *text* para el email, otro input de tipo *password* para la contraseña y otro input que será un *button* al cual le asignaremos que cuando se lance el evento de clic de dicho botón, lance una función de JavaScript.

Para el JavaScript de la página utilizaremos JQuery, la cual es una biblioteca multiplataforma que facilita el uso para interactuar entre JavaScript y documentos HTML, el DOM, eventos y la más importante y la que en este proyecto nos ocupa que es, que esta librería ayuda a agregar integración con la técnica AJAX con nuestras páginas web.

AJAX (Asynchronous JavaScript And XML), es la técnica de desarrollo web, mediante la cual se consiguen crear aplicaciones interactivas. Estas aplicaciones o web interactivas se consiguen gracias a que se ejecutan en el cliente, pero mantienen una comunicación asíncrona con el servidor en un segundo plano. De esta forma podemos cambiar datos de nuestras páginas sin necesidad de tener que recargar la página web.

Nosotros para esta página web que concierne al proyecto, vamos a desarrollar una función que utilizará AJAX, la razón es la nombra anteriormente, podemos realizar peticiones al servidor para poder realizar cambios, sin tener que recargar, además de que no congelaremos el hilo de ejecución de JavaScript (tenemos un hilo por pestaña en el navegador).

Nuestra petición de AJAX será de tipo POST, ya que vamos a enviar en el cuerpo de la petición los datos del email y la contraseña que tenemos en el input. La URL a la que apuntará la petición será la IP de mi servidor donde está corriendo el servicio web con ambas APIs y a la ruta /loginMultiFactor que recordemos no es la llamada que nos devolverá el *login*, pero si realizará la autenticación.

Como cuerpo de la petición, crearnos un JSON con el email y la contraseña que tenemos en sus correspondientes inputs, como el contenido va a ser un JSON, también le indicaremos a la llamada que el contenido va a ser de este tipo y codificada en UTF-8

Indicaremos que queremos hacer la llamada de manera asíncrona y definiremos la funciones que queremos que se llamen cuando el código de estado sea correcto, es decir código de estado 200 “OK” y que llamada cuando llegue cualquier código de error.

En caso de ir mal la petición, mostraremos una alerta en la página con el mensaje de error que nos ha devuelto la API. En caso de que la petición haya ido correctamente, llamaremos una función que se encargara de ahora sí hacer la petición a ruta “/login” de nuestro servidor.

La petición con AJAX a esta nueva ruta será igual que la que hemos realizado antes, pero en este caso varía la función a la que llamaremos si todo va ha ido bien. Esta llamada también va a ser asíncrona ya que cuando llegue la petición a la API si no hemos ingresado el valor de segundo factor de autenticación, se nos va a pedir que lo ingresemos y recordemos que tienes hasta dos minutos para hacerlo de lo contrario la API devolvía un error al usuario indicándole que tiene que usar algún método de segundo factor de autenticación.

En este punto es cuando el usuario deberá usar la aplicación de su dispositivo en la que tiene que haberse autenticado previamente y autenticarse con el segundo factor de autenticación contra la API. De esta manera estamos consiguiendo añadir un segundo factor de autenticación, en este caso biométrico, al proceso de *login* de una página web normal y corriente. Como hemos podido observar, este proceso puede ser trasladado a cualquier ámbito en que se puedan realizar peticiones HTTP, es decir prácticamente todos los dispositivos que encontraremos con conectividad a internet.

Si este segundo proceso ha ido mal en la petición a la API al igual que antes mostraremos un mensaje indicando que el mensaje de error que ésta nos haya devuelto. En caso de haber ido todo bien, se almacenará el token y su hora de expiración en el cliente y se navegará a la vista privada de la página web.

4.6 Flujo del servicio

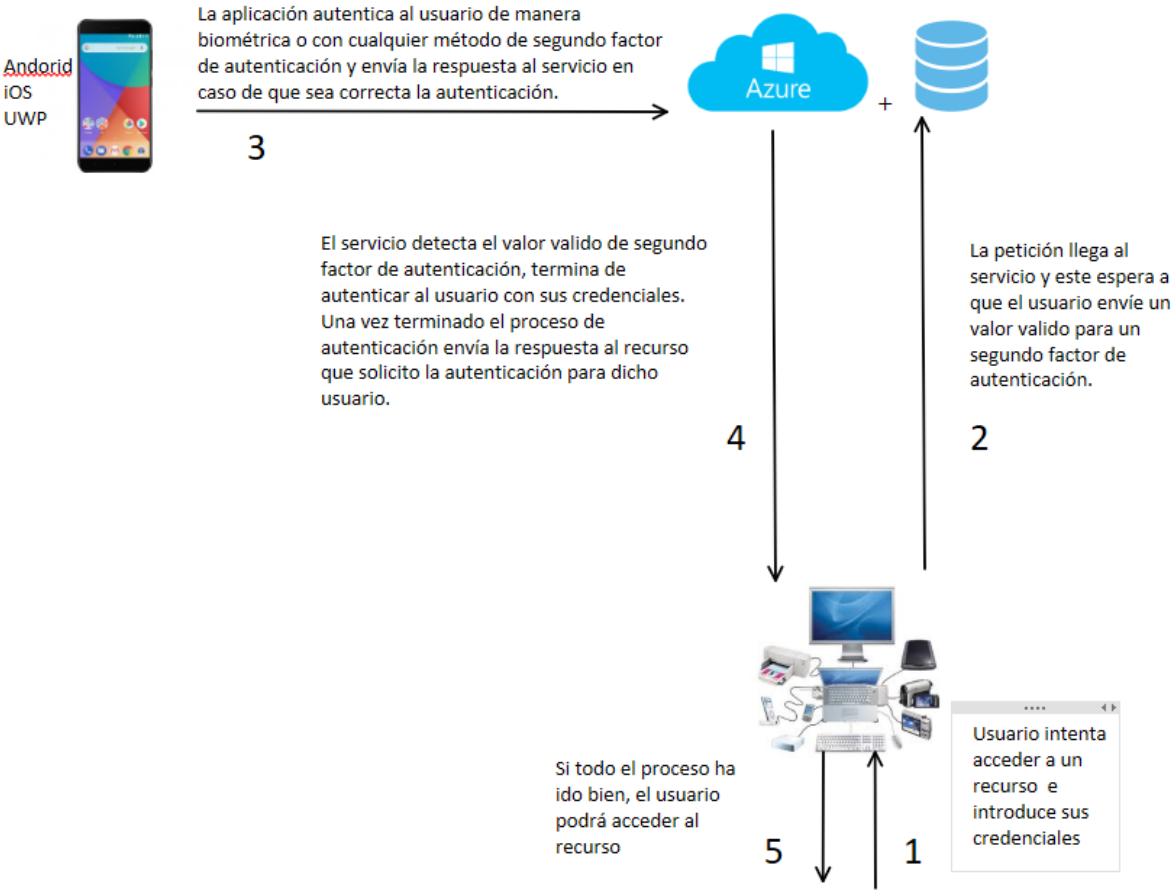

Figura 18 Diagrama de flujo para la autenticación de un usuario

4.7 Diagrama EER de la base de datos del proyecto

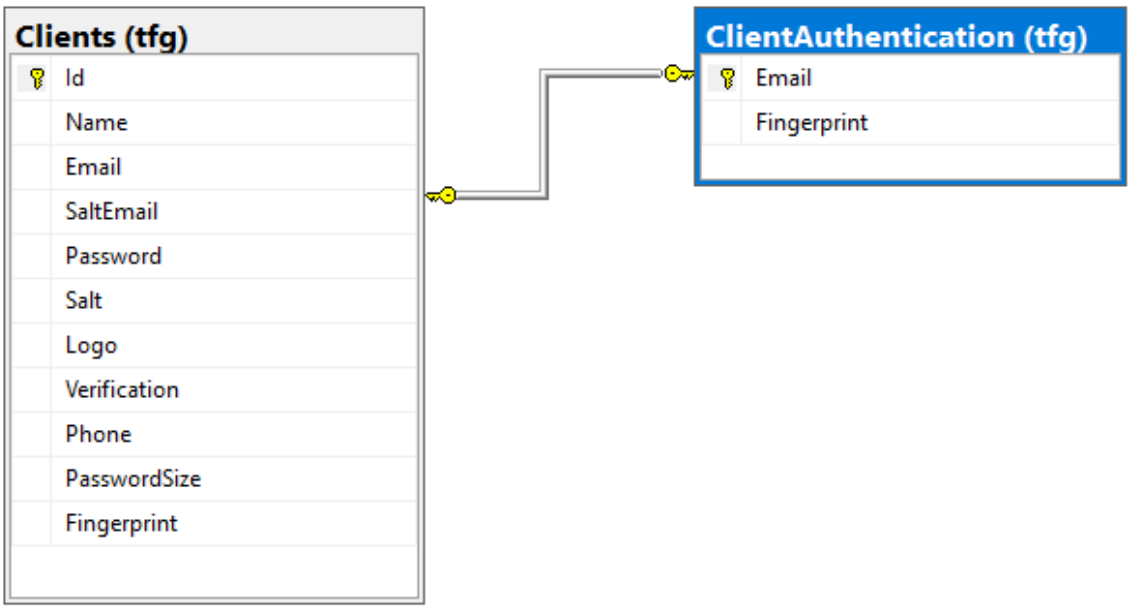

Figura 19 Diagrama de las tablas para los clientes

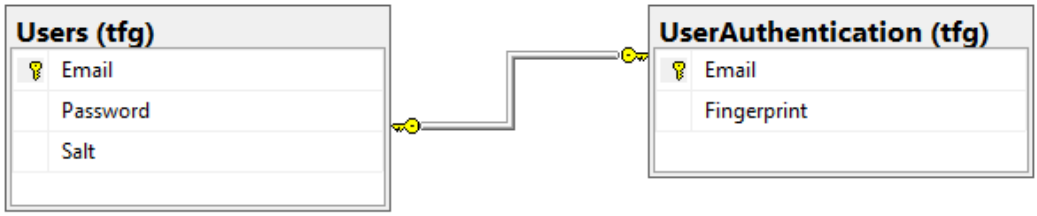

Figura 20 Diagrama de las tablas para los usuarios

5. Problemas encontrados

El desarrollo de este proyecto ha sido largo, pero en general no se han encontrado problemas muy graves que nos impidieran continuar el desarrollo de este.

Uno de los mayores problemas que se me ha planteado durante el desarrollo, ha sido la generación de los certificados de X.509 para el cifrado de las comunicaciones con TLS. El problema vino en la creación de los certificados, ya que al estar trabajando y desarrollando estos proyectos sobre el sistema operativo Windows 10, se me plantearon grandes dificultades para la generación del certificado, ya que era complicado cuando el certificado solo lo querías para desarrollar y no para un producto en producción.

Es por esta razón que, durante el proceso de creación de los certificados, tuve problemas para crearlos en el sistema operativo Windows 10, por lo que los certificados fueron creados en un sistema Ubuntu 16.04 siguiendo la guía de "Let's Encrypt" y posteriormente fueron copiados al proyecto desarrollado en Windows 10, bajo Visual Studio Code. Utilizando este método para la creación de los certificados ha sido la manera más sencilla que he encontrado para poder tener certificados para el entorno de desarrollo.

Otro gran problema fue la generación del JWT. El problema que se me planteó con la generación de JWT fue la firma de los propios tokens. Para poder firmar los JWT, hemos visto que necesitábamos una clave publica y una privada, dedicadas una a firmar y otra para la comprobación de esta. El problema fue que al principio el sistema no reconocía mis propias claves RSA y tuve que plantear otra opción, la cual era generar las claves con curvas elípticas. Generando las claves con las curvas elípticas la firma funcionaba y pude comprobar todo lo referente al uso de JWT en la aplicación. La única pega de utilizar curvas elípticas en el proceso de firma es que sigues necesitando amabas claves, es decir la pública y la privada para posteriormente poder realizar la generación del token, pero en este punto el problema era que, para cada token generado, se creaban nuevas claves, de manera, que se tenían que almacenar en memoria para poder hacer la comprobación cuando fuera necesario. Como vemos el proceso de tener que estar generando claves y almacenándolas nos suponía un coste importante para un gran número de usuarios.

Finalmente pude crear unas claves de manera correcta para que el servidor pudiera reconócelas y de esta manera utilizar las claves como se describe en el proceso de generación de JWT en este documento.

La comunicación entre las aplicaciones Android y la página web con la API fueron otro de los problemas a tratar durante el desarrollo del proyecto.

El problema que supuso la comunicación de entre las aplicaciones Android con la API fue en el tema de las peticiones junto a TLS. Para poder realizar las peticiones a la API, ésta solo permite realizar peticiones a través de una comunicación que utilice el protocolo TLS. En principio no debería de suponer un problema, ya que los métodos que posee C# para el envío de las peticiones HTTP posee propiedades para poder habilitar el protocolo TLS en las comunicaciones. El problema fue que tras configurarlo en la aplicación para su correcto funcionamiento no funcionaba y las peticiones eran bloqueadas por la API. Esto era debido a que la aplicación tiene que configurar el uso de TLS de manera navita [37], para que funcione correctamente en todos los dispositivos. Tras configúralo para usar el protocolo TLS de manera nativa todas las demás configuraciones funcionaron correctamente y se pudo activar la comunicación entre la aplicación y la API.

Durante el proceso de desarrollo de la aplicación de Android, estuve realizando las pruebas con un dispositivo inteligente. Pero durante el tiempo de desarrollo el dispositivo sufrió una actualización de seguridad, la cual a primera vista no debía de entorpecer el proceso de desarrollo. El problema vino a la hora de realizar las peticiones, ese desarrollo ya estaba terminado y funcionaba correctamente, pero tras la actualización, las peticiones que se enviaban a la API cambiaron, de manera que el cuerpo de la petición llegaba en un formato distinto al que estaba llegando antes. Este problema supuso tener que cambiar el método de deserialización y transformación del cuerpo de la petición en todos los controladores de las llamadas, para que lo hicieran de una forma más estándar y segura para todas las peticiones. Este error vino bien para poder darme cuenta de las cosas que se estaban realizando mal en los controladores de ambas APIs.

El último problema, como se comentaba más arriba, fue la comunicación de la página web con la API, ya no por las peticiones o por el protocolo TLS, sino porque el certificado que disponemos en la API para el uso de TLS era un certificado de desarrollo y el problema reside en que los nuevos navegadores, no confían en los certificados que nosotros hemos generado para el proyecto.

Por lo que para poder solucionar el problema tuve que agregar de manera manual a mi navegador el certificado para que pudiera confiar en él y así permitirme realizar las peticiones a la API.

6. Líneas futuras

A este proyecto, se pueden añadir muchas más funcionalidades, así como muchas mejoras al sistema que hay ahora desarrollado. Podríamos decir que el proyecto desarrollado y descrito en este documento es la base para un proyecto y servicio mucho mayor.

Es por ello por lo que pese a quedarse fuera del alcance del proyecto, a continuación, se describirán las líneas futuras para este proyecto:

- Conforme vaya avanzando la tecnología y con el paso del tiempo, se actualizarán las librerías de los lenguajes de programación con nuevos métodos, propiedades y estándares de seguridad, este servicio añadirá como mejora futura soportar siempre la última versión de TLS, para así poder disponer siempre de las mejores comunicaciones posibles. Como veíamos en el apartado de desarrollo, el proyecto solo soporta desde la versión SSL 3.0 a la versión TLS 1.2. La idea es que con el tiempo se añada soporte a la versión TLS 1.3 lanzada en agosto de 2018.

- La siguiente mejora también tiene que ver son el protocolo TLS, ya que como comentábamos en el caso anterior, la tecnología avanza muy rápido y el protocolo SSL 3.0 no es el más seguro actualmente. Está disponible en el desarrollo de este proyecto, solo para que sea mucho más compatible con más tecnologías y el desarrollo sea más comodo, pero a la larga al igual que ocurre en el apartado anterior, que se añaden nuevas versiones del protocolo, también se irán eliminado versiones más antiguas del protocolo que haya en el servicio.

- En el desarrollo inicial de ambas APIs, hemos podido observar que siempre que se tiene que trabajar con la base de datos, se abre la conexión y se realiza la consulta o transacción.

  Cuando solo son consultas a la base de datos este proceso es el correcto, pero en el caso por ejemplo del registro de un cliente, se deben de realizar muchas acciones con la base de datos y en caso de que algo salga mal, hay que realizar un *roll back* para poder dejar toda la base de datos tal y como estaba antes de comenzar con dicha transacción. Este proceso se ha realizado de esta manera, ya que es un proyecto de

código abierto y quería que se pudiera ver, como se procedía con la base datos, en el proyecto.

Por lo que, para este caso, se requiere que hagamos muchas operaciones seguidas con comprobación de error a la base de datos, por lo que, como línea futura, una mejor aproximación sería cambiar todos estos pequeños procesos, por un procedimiento almacenado en la base de datos. Este procedimiento almacenado, realizaría todas las operaciones pertinentes y nos devolvería un valor u error, en función de cómo haya ido su ejecución con los valores dados.

- El JWT, también tiene su mejora futura. Ahora mismo el servicio, genera todos los JWT con la misma clave de RSA, esto quiere decir que, tanto para los clientes como para los usuarios, el token es firmado de la misma manera. Esta situación, es debida a que ha sido desarrollado en un entorno de desarrollo y no en un entorno de producción con cliente reales. La línea futura para este punto es otorgar la posibilidad al cliente que se registra, de proporcionar sus propias claves RSA, para la generación del JWT de sus usuarios, de esta forma los usuarios de cada cliente utilizarán la firma otorgada por el cliente y nadie más hará uso de esas claves para el JWT, haciendo JWT únicos para cada cliente.

  Además de otorgar esa posibilidad al cliente, otra de las posibilidades que se le brindarán, será la posibilidad de que el propio servicio te generé las claves RSA y te las envíe al correo proporcionado por el cliente durante el registro.

  Con este cambio, además el cliente podrá integrar de una manera más sencilla la autenticación de usuarios y el JWT con otros de sus sistemas o software.

  Por último, como la integración sería mucho mejor, también se le brindaría la posibilidad al cliente de poder seleccionar algunos apartado o información que quiera que contenga el JWT de sus usuarios.

- Las bases de datos del proyecto, también tiene líneas futuras. En esta ocasión, una línea futura importante es otorgar a los clientes la posibilidad de manejar la base de datos de dos formas diferentes.

  Como primera posibilidad que se otorgaría al cliente, sería la de poder seleccionar que sistema de gestión de base de datos prefiere el cliente para almacenar sus tablas de información. En el proyecto, se ha desarrollado todo el entorno en SQL Sever,

pero podemos dejar la selección del sistema al cliente para poder elegir entre MySQL, MariaDB…

De esta forma, el cliente trabajaría con la base de datos que el consideré más correcta o segura para sus datos. Este aspecto también podría estar presente en el modelo de negoción, ya que se puede cobrar al cliente en función de su selección de sistema de gestión de base de datos.

La segunda posibilidad que se otorgaría a los clientes es la de poder tener ellos el control de sus propias tablas en la base de datos utilizando nuestro servicio. Si los clientes ya cuentan con su propio sistema de base de datos y quieren ahorrarse el coste del almacenamiento de la información en el servicio, o por seguridad prefieren ellos tener el control de la información en la base de datos, pueden decidir tener las tablas de nuestro servicio para dicho cliente almacenado en sus sistemas de base de datos. Para ello el cliente nos debería dar acceso a su sistema para poder gestionar las tablas durante el proceso de creación de usuarios y la posibilidad de realizar consultas para el proceso de autenticación.

- Otra línea futura para las bases de datos y las tablas sería la encriptación de estas. Muchos de los sistemas de gestión de bases de datos, poseen TDE (Transparent Data Encryption). TDE consiste en encriptación de datos a nivel de fichero, en el caso de las bases de datos a nivel de disco duro y los backups. Solo sirve para proteger los datos que están almacenados, pero no protege el intercambio de información como por ejemplo en una consulta a través de una cadena de conexión en un cliente. El cifrado se realiza mediante un certificado y clave maestra, por lo que en una situación de robo de archivos de bases de datos o backups, a menos que se disponga del certificado y la clave maestra no se podrán visualizar los datos.

- A lo largo del desarrollo del servicio, hemos podido observar que se ha utilizado Argon2 como la función principal de derivación de claves. Mediante esta función el servicio realiza el proceso de resumen o *hash* sobre el email y la contraseña del usuario o del cliente.

  Al igual que como hemos visto antes el cliente puede configurar cada aspecto del servicio para que sus usuarios cumplan todas las restricciones de seguridad que el cliente o empresa quiera imponerles. Es por ello por lo que la línea futura de cara a este punto radica en que el cliente pueda seleccionar que función de derivación de

claves se va a utilizar y la configuración de esta función, con el fin de que sea utilizada en el proceso de registro y autenticación de sus usuarios.

Entre las funciones de derivación de claves tendremos disponibles las siguientes: PBKDF2 [38], BCRYPT [39], SCRYPT [40] y Argon2. Durante el desarrollo de este proyecto, los parámetros para la función de Argon2, los hemos definido nosotros dentro de unos límites, para obtener seguridad y un buen rendimiento. En este caso ocurrirá lo mismo, el cliente podrá definir los parámetros de las funciones de derivación de clave, pero siempre dentro de unos límites que nos permitan tener seguridad y rendimiento.

- En el apartado del desarrollo de las aplicaciones Android para dispositivos inteligentes, vimos que con la versión 9.0 de este sistema operativo la clase "FingerprintManager entra en desuso y se empezará a utilizar la clase "BiometricPromt" de esta versión en adelante, la cual proporciona comunicación con todos los lectores biométricos del dispositivo.

  Esta clase nos brinda la posibilidad de introducción de nuevos métodos de autenticación biométricos para poder usarlos como segundo factor de autenticación. Es en este punto, donde encontramos una línea futura, que consistirá en añadir poco a poco todos los sensores biométricos que podamos a nuestras aplicaciones.

  En este punto el servicio solo cuenta con lector de huellas dactilares como segundo factor de autenticación biométrico, la idea es que poco a poco se vayan integrando más conforme se vaya trabajando en el desarrollo del servicio. Este servicio está preparado para usar siempre algún tipo de método de segundo factor de autenticación.

  Otro aspecto que tratar en esta línea futura es el de utilizar en las APIs los SDK de las compañías de dispositivos inteligentes, para poder ofrecer a un mayor publico soluciones biométricas que se encuentran presente en sus dispositivos. De esta manera se llegaría a un mayor público, ya que no es necesario la adquisición de un dispositivo nuevo para poder usar el servicio, cubriendo así más porcentaje y tasa de mercado, que como vimos, Android 9.0, solo representaba el 1% del mercado actual de Android.

  Un ejemplo de esto sería integrar en las APIs el SDK de Samsung, el cual nos permitirá utilizar el lector de Iris que portan algunos de sus dispositivos.

- Como penúltima línea futura tendríamos el desarrollo del resto de aplicaciones para los diferentes sistemas operativos de los dispositivos móviles, me refiero a iOS y a Windows con UWP, recordando que hablamos de estos sistemas operativos, ya que son lo que son soportados por Xamarin. Estas aplicaciones nos aportarían otros métodos de segundo factor de autenticación.

  En el caso de iOS, para los dispositivos de Apple, encontraríamos a parte del reconocimiento a través de huellas dactilares, el reconocimiento facial en 3D.

  En el caso de desarrollar una aplicación UWP para Windows, tendríamos la posibilidad de utilizar Windows Hello, el cual nos otorgaría la posibilidad de usar el lector de huellas dactilares de nuestros ordenadores, el reconociendo facial si cuentas con una cámara con infrarrojos en el ordenador y desbloqueo por posesión de un objeto, como puede ser una pulsera inteligente, el teléfono…

  Como vemos, la línea futura de las aplicaciones del servicio para dispositivos móviles, no es solo añadir nuevos métodos de autenticación biométricos con el avance y abaratamiento de la tecnología o no biométricos, también debemos de desarrollar aplicaciones para el resto de sistemas operativos, recordemos que gracias a Xamarin, esta opción es mucho más sencilla y rápida, pues solo debemos añadir los métodos e implantarlos en cada plataforma de una marera rápida gracias a "DepencencyService", ya que todas las aplicaciones compartirán la misma aplicación base.

- La última línea futura está orientada a los dispositivos inteligente que son un poco antiguos y aunque pueden correr la aplicación sin problemas, carecen de sensores biométricos.

  Como línea futura para los métodos de segundo factor de autenticación, se dispondrá en el servicio, de métodos no biométricos como el SMS, email, contraseñas del tipo "Time-base One-Time Password" (TOTP, contraseñas de un solo uso basadas en el tiempo), contraseñas del tipo "HMAC-base One-time Password" (HOTP, contraseñas basadas en eventos) …

  Además de todo lo nombra en el apartado de arriba, también podremos disponer de sistema basado en la posesión, el cual es un poco menos seguro.

Este sistema de autenticación basado en la posesión, pueden ser por ejemplo tarjetas NFC únicas o un dispositivo *wearable*, como pueden ser las famosas pulseras inteligentes, relojes…

De esta forma con esta línea futura podremos dar mucho más apoyo a los usuarios que no cuentan con un dispositivo de última generación y que no por ello vayan a tener menos sistemas de seguridad que el resto de los demás dispositivos.

7. Conclusiones

Con este proyecto, se ha logrado desarrollar desde cero, un sistema de autenticación de usuarios con factor múltiple. La integración de más sistemas para el segundo factor de autenticación es sencilla, ya que el servicio desarrollado posee las capacidades y está preparado para poder escalar los métodos de segundo factor de una manera cómoda.

Al estar diseñado con una estructura *cloud* desde el principio y al haber distribuido ya en Azure, el servicio está preparado para escalar tanto vertical como horizontal sin dificultades, ya que Azure nos proporciona la posibilidad de escalar recurso de una manera rápida.

El desarrollo del proyecto comenzó con el objetivo de ser un servicio más seguro y privado que los que podemos encontrar ahora mismo en el mercado para poder realizar autenticación OAuth. Con el proyecto ha conseguido lo propuesto, ya que no se almacena ningún dado sensible del usuario sin cifrar ni tampoco recauda información privada o sensible sobre el usuario.

El desarrollo ha sido pensado para ser seguro y privado en todos los aspectos del proceso de autenticación de usuarios y de los clientes. Al igual que ha sido pensado para ser seguro en todos sus puntos, también ha sido pensado para poder obtener un gran rendimiento con pocos recursos, sin tener que perder seguridad en el camino.

El servicio ha sido desarrollado con la finalidad de poder ser integrado y usado en cualquier sistema que tenga conectividad a Internet, como por ejemplo cualquier tipo de software como pueden ser programas de oficina, páginas web, aplicaciones de dispositivos móviles, etc.

Al igual que ha sido pensado para poder ser usado en cualquier sistema, también ha sido pensado y desarrollado para que el servicio se amolde a las necesidades de seguridad de cada cliente, de modo que cualquier empresa que este empezando pueda hacer uso del servicio o que cualquier empresa con tiempo en el sector pase a usar nuestro servicio, ya que ha sido pensado para que todas puedan hacer uso de él.

Este proyecto tiene mucho potencial y puede ampliarse fácilmente, puesto que las tecnologías asociadas a la autenticación de usuarios evolucionan continuamente.

El servicio desarrollado junto con las mejoras nombradas en las líneas futuras previamente, conformarían un servicio realmente potente de autenticación de usuarios.

8. Referencias

[1] J. Pastor, «Xataka,» 19 Marzo 2018. [En línea]. Available: https://www.xataka.com/privacidad/el-escandalo-de-cambridge-analytica-resume-todo-lo-que-esta-terriblemente-mal-con-facebook.

[2] «TLS Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security.

[3] «OAuth Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/OAuth.

[4] «Argon2 Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Argon2.

[5] «Password Hashing Competition Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Password_Hashing_Competition.

[6] «Package Argon2,» [En línea]. Available: https://godoc.org/golang.org/x/crypto/argon2.

[7] F. Ziegelmayer, «Comparation Bcrypt/ PBKDF2 vs Argon2.,» [En línea]. Available: https://hexdocs.pm/argon2_elixir/Argon2.html.

[8] «Bcrypt Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Bcrypt.

[9] «PBKDF2 Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/PBKDF2.

[10] «Cryptographic hash function Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Cryptographic_hash_function.

[11] «Go Tour,» [En línea]. Available: https://tour.golang.org/list.

[12] «Go Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Go_(programming_language).

[13] «Golang with Azure Data Base,» [En línea]. Available: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-connect-query-go.

[14] «Create Go apps with SQL Server,» [En línea]. Available: https://sqlchoice.azurewebsites.net/en-us/sql-server/developer-get-started/go/rhel/step/2.html .

[15] «Package go-mssqldb,» [En línea]. Available: https://github.com/denisenkom/go-mssqldb.

[16] «Package HTTP,» [En línea]. Available: https://golang.org/pkg/net/http/.

[17] «Package Gorilla Mux,» [En línea]. Available: https://github.com/gorilla/mux.

[18] «Package Negroni,» [En línea]. Available: https://github.com/urfave/negroni.

[19] «Package TLS,» [En línea]. Available: https://golang.org/pkg/crypto/tls/.

[20] Denji, «Golang-TLS,» [En línea]. Available: https://github.com/denji/golang-tls .

[21] «tls.X25519 Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Curve25519.

[22] «Cipher Suite Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Cipher_suite.

[23] «Package JSON,» [En línea]. Available: https://golang.org/pkg/encoding/json/.

[24] Dgrijalva, «JWT Go,» [En línea]. Available: . https://github.com/dgrijalva/jwt-go.

[25] Dgrijalva, «JWT Go example,» [En línea]. Available: https://godoc.org/github.com/dgrijalva/jwt-go#example-Parse--Hmac.

[26] «Package Base64,» [En línea]. Available: https://golang.org/pkg/encoding/base64/#Encoding.Decode.

[27] «Package X509,» [En línea]. Available: https://golang.org/pkg/crypto/x509/#ParsePKIXPublicKey.

[28] «Package UUID,» [En línea]. Available: https://godoc.org/github.com/google/uuid.

[29] Microsoft, «Schema SQL Server.,» [En línea]. Available: https://docs.microsoft.com/es-es/sql/t-sql/statements/create-schema-transact-sql?view=sql-server-2017.

[30] Apple, «Keychain Services,» [En línea]. Available: https://developer.apple.com/documentation/security/keychain_services.

[31] «Android Keystore,» [En línea]. Available: https://developer.android.com/training/articles/keystore.

[32] «Xamarin Secure Storage.,» [En línea]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/secure-storage?tabs=android.

[33] Microsoft, «Fingerprint Android,» [En línea]. Available: https://docs.microsoft.com/es-es/xamarin/android/platform/fingerprint-authentication/get-started?tabs=windows.

[34] «BiometricPrompt,» [En línea]. Available: https://developer.android.com/reference/android/hardware/biometrics/BiometricPrompt.

[35] «Fingerprint HAL,» [En línea]. Available: https://source.android.com/security/authentication/fingerprint-hal.

[36] Microsoft, «Dependency Service,» [En línea]. Available: https://docs.microsoft.com/es-es/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction.

[37] «TLS Android,» [En línea]. Available: https://docs.microsoft.com/es-es/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client.

[38] «Package PBKDF2,» [En línea]. Available: https://godoc.org/golang.org/x/crypto/pbkdf2.

[39] «Package BCRYPT,» [En línea]. Available: https://godoc.org/golang.org/x/crypto/bcrypt.

[40] «Package SCRYPT,» [En línea]. Available: https://godoc.org/golang.org/x/crypto/scrypt.

[41] «Go and SQL.,» [En línea]. Available: https://github.com/golang/go/wiki/SQLInterface.

[42] Ericchiang, «go-tls,» [En línea]. Available: https://ericchiang.github.io/post/go-tls/ .

[43] «Store credentials Android.,» [En línea]. Available: https://developers.google.com/identity/smartlock-passwords/android/store-credentials .

[44] «Android SharedPreferences.,» [En línea]. Available: https://developer.android.com/reference/android/content/SharedPreferences.

[45] Apple, «Biometric Auth Apple.,» [En línea]. Available: https://developer.apple.com/documentation/localauthentication .

[46] «Biometrics,» [En línea]. Available: https://source.android.com/security/biometric.