Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Diseño e Implantación de un Sistema de Autenticación Cross-platform para React y React Native

Autor: Xiao Peng Ye
Tutor: Vicente Martinez Orga

Madrid, 06 - 2022

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

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Diseño e Implantación de un Sistema de Autenticación Cross-platform para React y React Native

06 - 2022

Autor: Xiao Peng Ye
Tutor: Vicente Martinez Orga
Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

En resumen, el proyecto es un estudio e implementar una arquitectura de referencia que unifique el sistema de autenticación y autorización en React y React Native usando un *back-end* implementado con API GraphQL. El sistema incluye flujos de inicio de sesión, registro, gestión de contraseña y sesiones, e inicio de sesión mediante proveedores OAuth (Google, Facebook y LinkedIn).

Para ello, en primer lugar, se ha realizado un informe en el cual se especifica los requisitos tanto funcionales como no funcionales que tendrá que cumplir el módulo y un estudio sobre las tecnologías que se ha utilizado para conseguir la implementación.

A continuación, se ha logrado efectuar diseños tanto de alto nivel como de bajo nivel, empleando diagramas UML (secuenciales, caso de uso) que ayuden a comprender los flujos no visuales de autentificación y autorización. En cuanto a la implementación, se llevará a cabo documentaciones necesarias para poder emplear como código de referencia.

Finalmente, se someterá a tests unitarios, integración y E2E para pruebas y validación del código en cuanto a las funcionalidades.

Abstract

The project consists of the research and implementation of a reference architecture, which unifies the authentication and authorization system in React and React Native by using the back end of API GraphQL implementation. The system includes login, registration, password, and session management processes, as well as login through OAuth vendors (Google, Facebook and LinkedIn).

Firstly, a report should be prepared detailing the functional and non-functional requirements that the module will meet, and the technologies used to realize the module should be studied.

Then, it is possible to carry out both high-level and low-level designs, using UML diagrams (sequential, use case) that help understand the non-visual flows of authentication and authorization. Regarding the implementation, the necessary documentation will be made so that it can be used as a reference code.

Finally, it will be subjected to unit tests, integration, and E2E for tests and validation of the code in terms of functionality.

Agradecimiento

La elaboración de este proyecto no habrían sido posibles sin el apoyo de mi tutor de TFG Vicente Martinez Orga por la ayuda en la planificación y organización en la memoria, ni de mi tutor profesional Paul Kuhle por el aporte técnico en el proyecto. Quiero agradecérselo a ellos.

Tabla de contenidos

Agradecimiento **v**

1. Introducción y objetivos **1**
1.1. Introducción 1
1.2. Objetivos 1
1.3. Motivación 1

2. Requisitos del proyecto **3**
2.1. Requisitos funcionales 3
2.1.1. Sistema de autenticación 3
2.1.1.1. Registrar usuario 3
2.1.1.2. Iniciar sesión tradicional 3
2.1.1.3. Restablecimiento de la contraseña 3
2.1.1.4. Cerrar la sesión 4
2.1.2. Sistema de social login 4
2.1.2.1. Social login 4
2.1.2.2. Cuentas de social login 4
2.1.2.3. Agregar correo 4
2.1.3. Historial de sesiones 4
2.1.3.1. Visualizar el historial de sesiones 4
2.1.3.2. Control de logout en el historial de sesiones 5
2.1.4. Comprobar el estado del filtro de la contraseña 5
2.2. Requisitos no funcionales 5
2.2.1. Seguir las buenas prácticas de seguridad 5
2.2.2. Tecnologías y compatibilidad 5
2.2.2.1. Tecnologías 5
2.2.2.2. Compatibilidad 6
2.2.3. Calidad del código 6
2.2.4. Internacionalización 7
2.2.5. Open-source 7

3. Trabajos previos al desarrollo **9**
3.1. Autenticación y autorización 9
3.1.1. Autenticación 9
3.1.2. Autorización 9
3.1.3. Comparación 10
3.2. Autenticación básica HTTP 10

3.2.1. Ventajas 12
3.2.2. Desventajas 12
3.3. Autenticación basada en Session-Cookie 12
3.3.1. Ventajas 14
3.3.2. Desventajas 14
3.4. Autenticación basado en Token 14
3.4.1. Ventajas 15
3.4.2. Desventajas 16
3.5. Autenticación basado en JWT 16
3.5.1. Ventajas 17
3.5.2. Desventajas 17
3.6. Single Sign-on 18
3.7. Social Login 18
3.7.1. OAuth 18
3.7.2. OpenID Connect 20
3.8. Vulnerabilidad 20
3.8.1. Cross-Site Request Forgery 20
3.8.2. Cross-Site Scripting 21
3.9. Tecnologías 22
3.9.1. TypeScript 22
3.9.2. React 24
3.9.3. Next.js 27
3.9.4. React Native 28
3.9.5. Expo 30
3.9.6. GraphQL 30

4. Desarrollo **35**
4.1. Configuración del entorno de trabajo 35
4.1.1. Monorepo 35
4.1.2. Mock Service Worker 36
4.1.3. Pruebas unitarias 37
4.1.4. Pruebas de integración y E2E 37
4.2. Resumen de los requisitos 38
4.3. Decisiones tomadas del diseño 39
4.3.1. Next.js 40
4.3.2. React Native 41
4.4. Diseño de los flujos 41
4.4.1. Registro de cuenta 41
4.4.2. Iniciar sesión 42
4.4.3. Cerrar sesión 44
4.4.4. Social login 45
4.4.5. Acceso a recursos autorizados 46
4.4.6. Restablecer y cambiar la contraseña 47
4.4.7. Añadir correo y contraseña 48
4.4.8. Controlar sesiones 49
4.5. Esquema de GraphQL 49
4.5.1. Query 50

4.5.2. Mutation 50
4.6. Vistas utilizadas 51

5. Conclusiones y trabajo futuro **61**
5.1. Conclusiones 61
5.2. Trabajo futuro 62
5.2.1. Desbloquear la App con rastros biométricos 62
5.2.2. MFA (Multi-factor authentication) 62

6. Análisis de impacto **63**
6.1. Personal 64
6.2. Empresarial 64
6.3. Social 64
6.4. Económico 64
6.5. Medioambiental 64
6.6. Cultural 64

Bibliografía **65**

Anexos **71**

Capítulo 1

Introducción y objetivos

1.1. Introducción

En el mundo de las aplicaciones web modernas, hacer el registro o la creación de una cuenta de usuario e iniciarse sesión posteriormente son las acciones más frecuentes que se podría dar hoy en día al navegarse por el internet, así como precondición para disfrutar los servicios de chat, comprar un producto, etc. Son unas funcionalidades que presentan en cualquier aplicación web o App y además es un módulo que refleja criterios importantes en cuanto a los servicios que ofrecen tales como la experiencia de usuario y la seguridad.

1.2. Objetivos

El objetivo es diseñar e implantar una librería/arquitectura de referencia que unifique el sistema de autenticación en React y React Native, usando como proveedor de autenticación un *back-end* con una API GraphQL. El sistema incluye flujos de inicio de sesión, registro, gestión de la contraseña y de sesiones, e inicio de sesión mediante proveedores OAuth (Google, Facebook, y LinkedIn).

1.3. Motivación

En el desarrollo de aplicaciones web modernas se suelen utilizar *tech stacks* con *frameworks* para *single page apps*, *micro-frontends* y tecnologías *cross-platform*. Esas arquitecturas añaden un overhead significativo en algunas áreas, como es el caso de la gestión, la autenticación y de sesiones de usuarios en el *front-end* de las aplicaciones.

Las librerías de autenticación de código abierto todavía carecen de soluciones *cross-platform* (React/React Native). Además, librerías como next-auth operan como *middleware* y las sesiones se gestionan mediante funciones *serverless*. Faltan soluciones estandarizadas para integraciones con *back-end*s separados.

Capítulo 2

Requisitos del proyecto

La arquitectura de referencia tendrá que cumplir los siguientes requisitos tanto funcionales como no funcionales.

2.1. Requisitos funcionales

2.1.1. Sistema de autenticación

2.1.1.1. Registrar usuario

- **Código:** REQ-FUN-1.1
- **Dependencia:** N/A
- **Descripción:** Los usuarios deben registrarse en el sistema, indicando su dirección de correo y contraseña. El sistema les enviará un correo electrónico con un enlace y no se podrá acceder al sistema hasta que se valide la cuenta con el enlace.

2.1.1.2. Iniciar sesión tradicional

- **Código:** REQ-FUN-1.2
- **Dependencia:** REQ-FUN-1.1 (sección 2.1.1.1)
- **Descripción:** Los usuarios podrán iniciar sesión en el sistema mediante correo y contraseña registrado previamente.

2.1.1.3. Restablecimiento de la contraseña

- **Código:** REQ-FUN-1.3
- **Dependencia:** REQ-FUN-1.1 (sección 2.1.1.1)
- **Descripción:** Los usuarios podrán pedir la recuperación y el cambio de su contraseña si lo tienen y les llegarán un correo electrónico con un enlace específico para llevar a cabo.

2.1.1.4. Cerrar la sesión

- **Código:** REQ-FUN-1.4
- **Dependencia:** REQ-FUN-1.1 (sección 2.1.1.1), REQ-FUN-2.1 (sección 2.1.2.1)
- **Descripción:** Los usuarios podrán salir de la sesión actual una vez iniciada la sesión.

2.1.2. Sistema de social login

2.1.2.1. Social login

- **Código:** REQ-FUN-2.1
- **Dependencia:** N/A
- **Descripción:** Se permite iniciar sesión en el sistema mediante cuentas de Google, LinkedIn y Facebook sin tener que estar registrado previamente.
- **Justificación:** Los *social logins* facilitan a los usuarios nuevos el uso del servicio sin tener que crear una cuenta desde cero y recordar una contraseña específica para ello.

2.1.2.2. Cuentas de social login

- **Código:** REQ-FUN-2.2
- **Dependencia:** REQ-FUN-2.1 (sección 2.1.2.1)
- **Descripción:** Todos los *social logins* de cualquier proveedor con la misma cuenta del correo se debe identificar como usuarios distintos.

2.1.2.3. Agregar correo

- **Código:** REQ-FUN-2.3
- **Dependencia:** REQ-FUN-2.1 (sección 2.1.2.1)
- **Descripción:** Se obligará añadir la dirección de correo si el social login fuese del proveedor de Facebook.
- **Justificación:** La cuenta de Facebook es capaz de registrarse sin tener una cuenta de correo.

2.1.3. Historial de sesiones

2.1.3.1. Visualizar el historial de sesiones

- **Código:** REQ-FUN-3.1
- **Dependencia:** REQ-FUN-1.2 (sección 2.1.1.2), REQ-FUN-2.1 (sección 2.1.2.1)

- **Descripción:** Los usuarios podrán visualizar otras sesiones existentes de su cuenta mostrando ubicación, *timestamp*, dirección IP, el nombre del dispositivo y la plataforma (web o app).

2.1.3.2. Control de logout en el historial de sesiones

- **Código:** REQ-FUN-3.2
- **Dependencia:** REQ-FUN-3.1 (sección 2.1.3.1), REQ-FUN-1.4 (sección 2.1.1.4)
- **Descripción:** Los usuarios podrán realizar *logout* tanto para la propia sesión como para sesión de las otras plataformas y dispositivos.

2.1.4. Comprobar el estado del filtro de la contraseña

- **Código:** REQ-FUN-4
- **Dependencia:** REQ-FUN-1.1 (sección 2.1.1.1), REQ-FUN-1.3 (sección 2.1.1.3)
- **Descripción:** Se deben comprobar que todas las contraseñas, que se usan tanto para el registro de cuenta como el restablecimiento de la contraseña, no están filtradas en el internet.

2.2. Requisitos no funcionales

2.2.1. Seguir las buenas prácticas de seguridad

- **Código:** REQ-NO-FUN-1
- **Dependencia:** N/A
- **Descripción:** Se deben seguir los estándares de seguridad en cuanto al desarrollo del sistema ofrecido por las organizaciones, entidades y corporaciones reconocidas.

2.2.2. Tecnologías y compatibilidad

2.2.2.1. Tecnologías

- **Código:** REQ-NO-FUN-2.1
- **Dependencia:** REQ-NO-FUN-5 (sección 2.2.5)
- **Descripción:** Se usará Next.js para garantizar la renderización del web en el servidor y GraphQL como protocolo de comunicación entre *front-end* y *back-end*. En cuanto a los *App* se implementarán mecanismos de *deep link* para llevar el usuario a contenidos específicos desde la web.
- **Tecnologías:**
  - *Server-side rendering* web: Next.js
  - Protocolo de consulta: GraphQL

2.2.2.2. Compatibilidad

- **Código:** REQ-NO-FUN-2.2
- **Dependencia:** REQ-NO-FUN-2.1 (sección 2.2.2.1)
- **Descripción:** Se debe garantizar que todo el módulo sea compatible con las siguientes versiones del *framework*, plataformas, etc. a utilizar.
- **Versiones:**
  - Node.js: v16.14.0 LTS
  - React: 17.0.0
  - Next.js: v12.1
  - React-native: 0.64.3
  - Expo SDK 44
  - Android 9 Pie
  - IOS 13
  - Exploradores
    - Chrome 98
    - Firefox 97
    - Safari 13
    - Edge 94
- **Justificación:** El criterio que se ha elegido las versiones de navegadores es el número de versión actual restando 2 para que puedan soportar las últimas características.

  En el caso de los dispositivos móviles se ha elegido Android 9 por la cuota de mercado actual y las características en cuanto a la seguridad, e IOS 13 por ser la versión a partir donde introdujeron el inicio de sesión privada de Apple en los apps y sitios webs.

2.2.3. Calidad del código

- **Código:** REQ-NO-FUN-3
- **Dependencia:** N/A
- **Descripción:** Se debe seguir los estándares y buenas prácticas en cuanto a la codificación y realizar pruebas necesarias para justificar la estabilidad del funcionamiento.
- **Estándares a seguir:** *Linter* de código: ESLint como herramienta para corregir errores sintácticos y la de establecer las reglas de formato del código

TypeScript: se usará para garantizar la mantenibilidad, limpieza y legibilidad de código mediante el mecanismo del tipado de las variables

- Design patterns:
  - Estándares de diseño para React `https://reactpatterns.com/`
  - Separación de lógica con los *Hooks* de React
- Estructura de ficheros:
  - Se usará *Monorepo* para organizar el proyecto
  - Se seguirán las guías oficiales en cuanto la estructuración de componentes `https://reactjs.org/docs/faq-structure.html`

- **Pruebas:**
  - Unit testing:
    - Jest `https://jestjs.io/`
    - react-testing-library `https://github.com/testing-library/react-testing-l`
  - Integration y E2E:
    - Cypress `https://nextjs.org/docs/testing#cypress`
    - Detox `https://wix.github.io/Detox/`

2.2.4. Internacionalización

- **Código:** REQ-NO-FUN-4
- **Dependencia:** N/A
- **Descripción:** Se deben garantizar mecanismos que faciliten el desarrollo de traducciones del UI.

2.2.5. Open-source

- **Código:** REQ-NO-FUN-5
- **Dependencia:** N/A
- **Descripción:** Se usarán herramientas, librerías y *frameworks* con licencia libre permisiva para desarrollar el sistema que garantiza las libertades de uso, modificación y redistribución, pero también permite obras derivadas patentadas.
- **Licencias:**
  - MIT License (Expat)
  - Apache License 2.0 (Apache-2.0)
  - BSD 3-Clause License (Revised)

- BSD 2-Clause License (FreeBSD/Simplified)
- ISC License
- BSD 0-Clause License (0BSD)
- Boost Software License 1.0 (BSL-1.0)
- X11 License

Capítulo 3

Trabajos previos al desarrollo

Los siguientes puntos son los estudios realizados sobre los protocolos, metodologías y conceptos importantes en cuanto a los sistemas de autenticación.

3.1. Autenticación y autorización

Al desarrollar o administrar una aplicación, a menudo vemos dos nombres llamados autenticación y autorización, que son similares también en inglés: *authentication* y *authorization*. Aunque los dos aparecen a menudo en el mismo contexto, los dos son conceptualmente muy diferentes. La autenticación significa confirmar su identidad, mientras que la autorización significa dar acceso al sistema. En pocas palabras, la autenticación es el proceso de verificar su identidad, mientras que la autorización es el proceso de verificar que tiene acceso [1] [2].

3.1.1. Autenticación

La autenticación consiste en validar sus credenciales, como nombre de usuario/correo electrónico y contraseña, para verificar la identidad del visitante. El sistema determina si el usuario está usando las credenciales como dice. Tanto en redes públicas como privadas, el sistema autentica al usuario a través del inicio de sesión y generalmente se realiza a través de un nombre de usuario y una contraseña. Sin embargo, la autenticación se puede realizar no solo a través de contraseñas, sino también a través de otros factores, como códigos de verificación de teléfonos móviles o datos biométricos.

En algunos sistemas de aplicaciones, para lograr una mayor seguridad, a menudo se requiere el uso conjunto de múltiples factores de autenticación, que es lo que a menudo llamamos *multifactor authentication*.

3.1.2. Autorización

La autorización ocurre después de que el sistema completa la autenticación y, en última instancia, le otorga acceso completo a recursos como información,

archivos, bases de datos, fondos, ubicaciones, casi cualquier cosa. En pocas palabras, la autorización determina su capacidad para acceder al sistema y en qué medida.

La autorización es el proceso de determinar si un usuario autenticado puede acceder a un recurso en particular. Verifica que está autorizado para otorgarle acceso a recursos tales como información, bases de datos, archivos, etc. La autorización generalmente confirma sus permisos después de la verificación.

Por ejemplo, el proceso de verificar tarjeta de identificación de empleado en una organización se denomina autenticación, pero la de determinar qué empleado puede acceder a qué piso se denomina autorización. Suponga que está de viaje y está a punto de abordar un avión. Cuando muestre su boleto y alguna identificación antes del *check-in*, recibirá una tarjeta de embarque que prueba que la autoridad del aeropuerto revisó su identidad. Pero eso no es todo, el asistente de vuelo debe autorizarte a abordar el vuelo en el que se supone que debes estar, dándote acceso al interior de la aeronave y sus recursos.

3.1.3. Comparación

En la tabla 3.1 se resumen brevemente las 2 secciones 3.1.1, 3.1.2.

Autenticación	**Autorización**
La autenticación confirma la identidad para otorgar acceso al sistema	La autorización determina si tiene acceso a un recurso
Este es el proceso de validación de credenciales de usuario para obtener acceso de usuario	Este es el proceso de verificar que el acceso está permitido
La autenticación generalmente requiere un nombre de usuario y una contraseña	Los factores de autenticación necesarios para la autorización pueden variar según el nivel de seguridad
La autenticación es el primer paso	La autorización se realiza después de una autenticación exitosa

Cuadro 3.1: Comparación entre autenticación y autorización

3.2. Autenticación básica HTTP

En HTTP, la Autenticación de acceso básico (*Basic Access Authentication*) permite que los agentes de usuario HTTP (generalmente navegadores web) autentiquen a los usuarios proporcionando nombres de usuario y contraseñas cuando lo soliciten [3].

Los elementos críticos son:

- **uid:** id del usuario
- **password:** contraseña
- **realm:** se refiere el alcance de la protección

En el proceso de autenticación básica, el campo del encabezado de la solicitud HTTP contendrá el campo `Authorization: Basic <credencial de usuario>`, la credencial de usuario es la combinación de la codificación Base64 de nombre de usuario y contraseña [4].

```
GET /securefiles/ HTTP/1.1
Host: www.example.com
Authorization: Basic aHR0cHdhdGNoOmY=
```

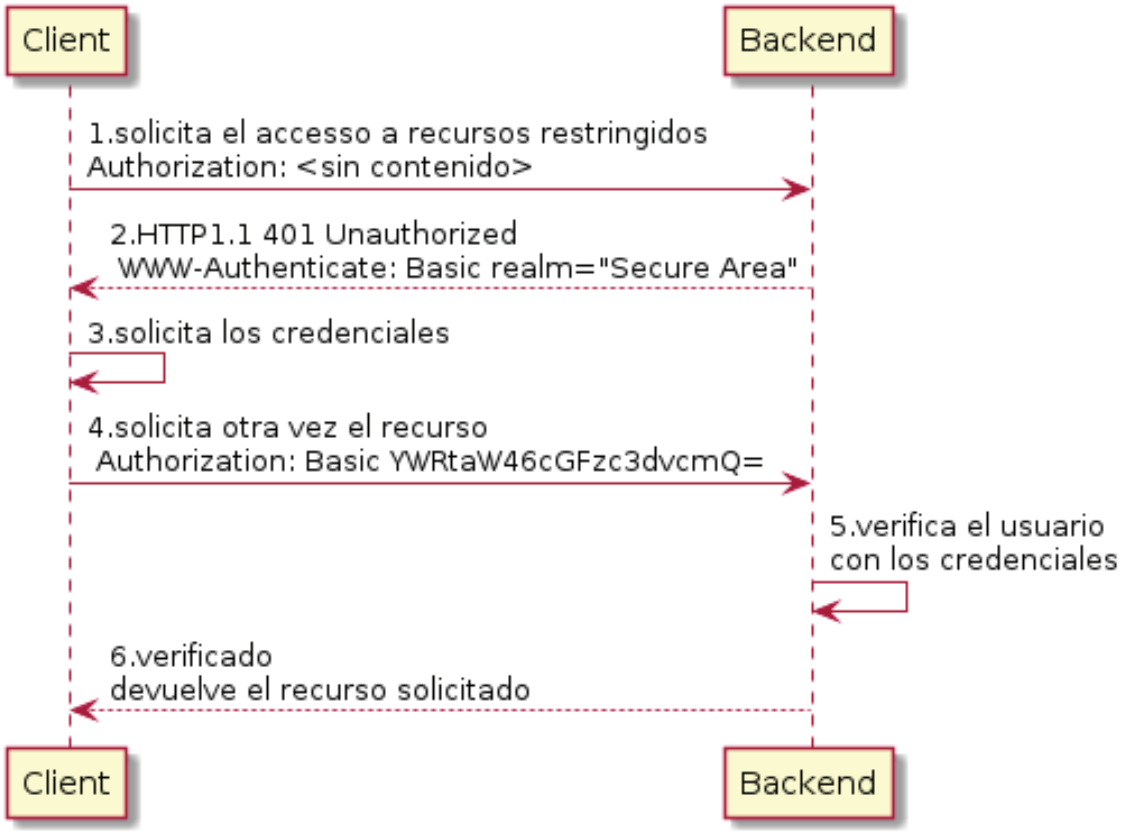

Figura 3.1: Flujo de Basic Access Authentication

Se detalla a continuación la figura 3.1:

1. El usuario accedió a un recurso web restringido en el navegador sin proporcionar la información de identidad del usuario.
2. Una vez recibido la solicitud, el servidor devuelve un código de respuesta 401 (No autorizado) para solicitar autenticación con un campo de autenticación (*Access Authentication*) `WWW-Authenticate` para explicar cómo autenticar, como `WWW-Authenticate: Basic realm="Secure Area"`, `Basic` es el modo de autenticación, y `realm="Secure Area"` es el dominio de protección.
3. Después de que el navegador reciba la respuesta, mostrará el dominio de

autenticación al usuario y le pedirá que ingrese el nombre de usuario y la contraseña. En este momento, el usuario puede elegir confirmar o cancelar la operación antes de ingresar la información.

4. Cuando el usuario ingrese el nombre de usuario y la contraseña, el navegador agregará el campo del mensaje de autenticación `Authorization` al encabezado de la solicitud original y volverá a enviar la solicitud.
5. Finalmente el servidor devuelve el recurso de la página web solicitado por el usuario. Si las credenciales del usuario no son válidas, el servidor puede devolver un código de respuesta 401 nuevamente y el cliente deberá ingresar el nombre de usuario y la contraseña nuevamente.

3.2.1. Ventajas

- La única ventaja es que es simple de implementar y ampliamente compatible.

3.2.2. Desventajas

- No es seguro porque los nombres de usuario y las contraseñas se transmiten a través de la red en texto claro y los rastreadores los detectan fácilmente.
- La codificación Base64 no es un algoritmo de cifrado, lo que no puede garantizar la seguridad y la privacidad. Solo se utiliza para convertir caracteres incompatibles en nombres de usuario y contraseñas en juegos de caracteres compatibles con el protocolo HTTP.
- Los usuarios maliciosos pueden iniciar solicitudes repetidamente después de autenticarse, este es el llamado ataque de repetición.
- Además de los problemas de seguridad, también existe una situación en la que la certificación `Basic` no se puede revocar.

3.3. Autenticación basada en Session-Cookie

La autenticación basado en *Session-Cookie* es un modo de autenticación de comunicación de *front-end* y *back-end* implementado mediante el uso de *Session* del servidor y la *Cookie* del navegador.

Dado que la solicitud HTTP no tiene estado, el servidor no puede conocer la identidad del remitente de la solicitud en circunstancias normales. Si queremos registrar el estado en este momento, debemos crear una sesión en el servidor y mantener las solicitudes del mismo cliente. En el registro de sesión, cada vez que llega una solicitud al servidor, primero verificara si el ID de usuario en la solicitud existe en la sesión. Si es así, significa que la autenticación ha sido exitosa, de lo contrario, significa que la autenticación ha fallado [5].

Las *Cookies* se utilizan principalmente para los siguientes tres aspectos [6]:

- Gestión del estado de la sesión (como el estado de inicio de sesión del usuario, el carrito de la compra, la puntuación del juego u otra información que deba registrarse)
- Configuración de personalización (como configuraciones definidas por el usuario, temas, etc.)
- Seguimiento del comportamiento del navegador (como seguimiento y análisis del comportamiento del usuario, etc.)

La siguiente figura 3.2 muestra el flujo de trabajo de la autenticación *Session-Cookie*:

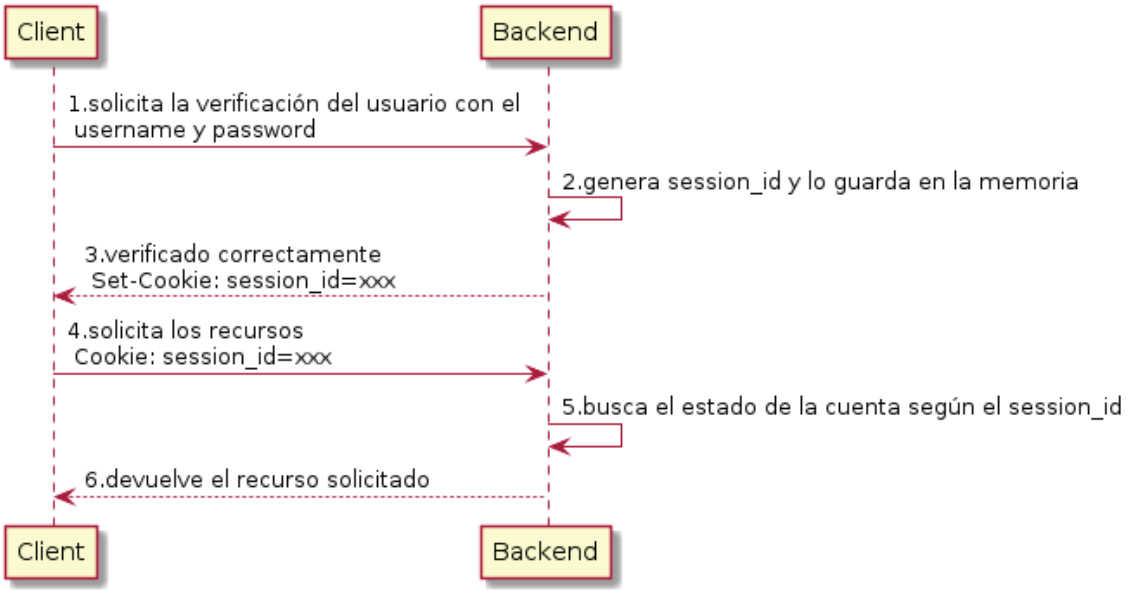

Figura 3.2: Flujo de Session-Cookie

1. Cuando el servidor recibe el primer acceso del cliente, automáticamente creará una sesión, y luego generará una cadena de identificación única `session id` (sid) para la sesión y establece este identificador único en el encabezado de respuesta con `Set-Cookie`.
2. Después de que el navegador reciba la respuesta de la solicitud, analizará el encabezado de la respuesta y guardará automáticamente el sid en la *Cookie* local. El navegador adjuntará automáticamente la información de la *cookie* bajo el nombre de dominio al encabezado de la solicitud en la siguiente solicitud HTTP.
3. Cuando el servidor recibe la solicitud del cliente, analizará el sid en la *cookie* del encabezado de la solicitud y luego encontrará el sid del cliente guardado por el servidor de acuerdo con el sid, y por último juzgará si la solicitud es legal o no.
4. Una vez que el usuario cierra la sesión, el servidor y el navegador destruirán

el ID de sesión guardada por el otro, al mismo tiempo, el servidor verificará el ID de sesión de acuerdo con la base de datos. Si se pasa la verificación, el procesamiento se continuará.

3.3.1. Ventajas

- Las *cookies* son fáciles de usar y, a menudo, son la forma más duradera de retención de datos en el cliente sin la intervención del usuario o la caducidad.
- Los datos de la sesión se almacenan en el lado del servidor, cuando el usuario inicia y cierra sesión activamente, simplemente agregue y elimine la sesión correspondiente, lo cual es conveniente para la administración.

3.3.2. Desventajas

- Muy inseguras, las *cookies* exponen datos al navegador, lo que aumenta el riesgo de robo de datos (fácil de ser atacado por CSRF (sección 3.8.1), etc.).
- La sesión se almacena en el lado del servidor, lo que aumenta la sobrecarga del lado del servidor. Cuando la cantidad de usuarios es grande, el rendimiento del servidor se reduce considerablemente [7].
- Una vez que el usuario se autentica, el servidor realiza un registro de autenticación. Si el registro de autenticación se almacena en la memoria, esto significa que la próxima solicitud del usuario también debe realizarse en el mismo servidor, para que se puedan obtener los recursos autorizados. Por lo que en aplicaciones distribuidas, la escalabilidad horizontal del servidor está limitada.

3.4. Autenticación basado en Token

Con el auge de las API y los microservicios de RESTful, la autenticación basada en tokens ahora es cada vez más común. Los métodos de autenticación token y *Session-Cookie* son diferentes, no solo por el tipo de identificador. El token generalmente contiene información relevante del usuario. Al verificar el token, no solamente se puede completar la verificación de identidad, sino que también se puede obtener información preestablecida. Las API públicas como Twitter y GitHub se autentican según este método, y algunos *framework* de desarrollo como OpenStack y las llamadas internas de API de Kubernetes también se autentican según los tokens.

La siguiente figura 3.3 muestra el flujo de trabajo de la autenticación basado en token [8]:

1. El usuario ingresa sus claves y solicita el inicio de sesión.
2. El servidor recibe la solicitud y verifica la información de inicio de sesión ingresada por el usuario.

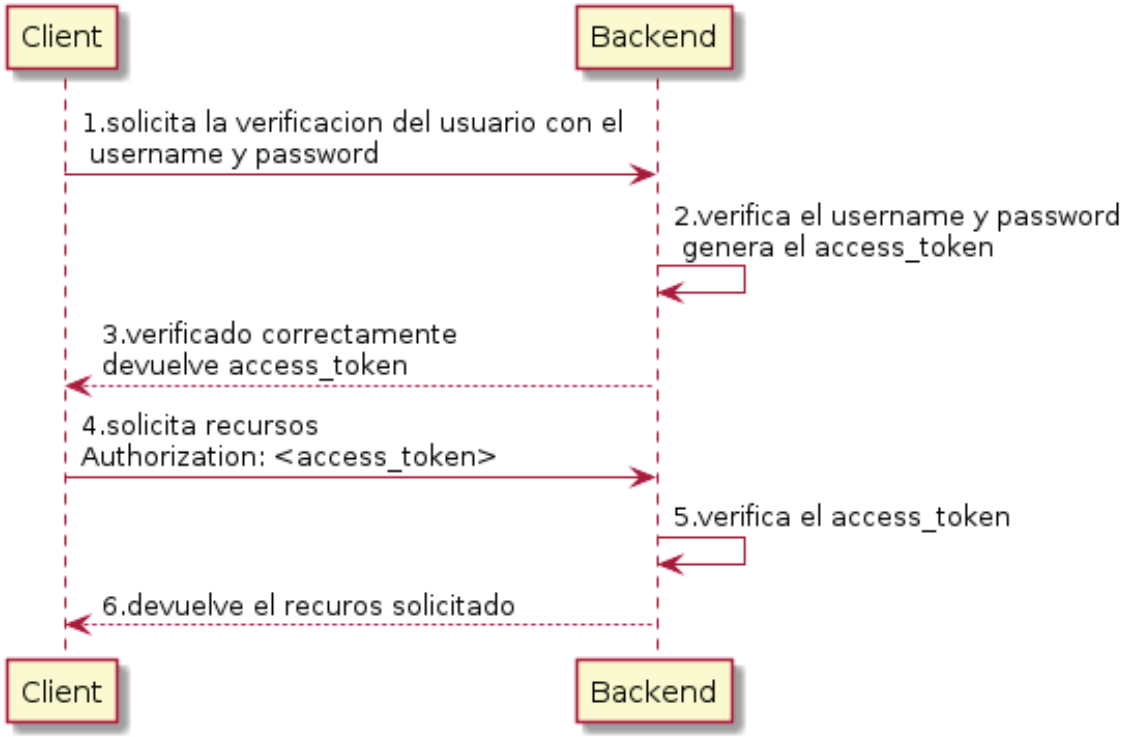

Figura 3.3: Flujo de Token

3. Después de que la verificación sea exitosa, el servidor emitirá un token (que generalmente incluye información básica del usuario, alcance del permiso y tiempo de validez, etc.) y devolverá el token al cliente.
4. Después de que el cliente recibe el token, debe almacenarlo, como en *localStorage* o *sessionStorage* (generalmente no se coloca en *cookies* porque puede haber problemas entre dominios y problemas de seguridad).
5. Cada vez que el cliente solicita recursos del servidor, adjunta el token al campo de autorización del encabezado de la solicitud HTTP y envía la solicitud.
6. Después de que el servidor recibe la solicitud, comprueba el token en la solicitud del cliente. Si la verificación fue exitosa, devuelve los datos solicitados al cliente, de lo contrario, se niega a devolver.

3.4.1. Ventajas

- Sin estado al lado del servidor: el mecanismo del token no necesita almacenar información de la sesión en el lado del servidor, porque el token en sí contiene información relevante sobre el usuario que identifica, lo que conduce a compartir el estado del usuario entre múltiples servicios.
- Rendimiento relativamente bueno: debido a que ya no es necesario acceder

a la base de datos o al servicio remoto para la verificación de permisos al verificar el token, naturalmente puede mejorar mucho el rendimiento.

- Soporte para dispositivos móviles.
- Admite llamadas entre dominios y entre programas, porque las *cookies* no permiten el acceso entre dominios, mientras que token no tiene este problema.
- Deja el cliente la libertad de escoger la forma para guardar el Token (*Cookies*, *Cookies HttpOnly localStorage*, *sessionStorage*, *asyncStorage*, etc.) para evitar ataques como CSRF (sección 3.8.1) o XSS (sección 3.8.2) según la plataforma.

3.4.2. Desventajas

- Complejidad: en comparación con la autenticación basada en *Session-Cookie*, Token requiere que el servidor dedique más tiempo y rendimiento para descifrar y verificar el token. De hecho, token es una solución que cambia tiempo por espacio en cuanto a la complejidad.

3.5. Autenticación basado en JWT

JWT (JSON Web Token) es un estándar abierto basdo en JSON para realizar la verificación de autorización cifrando y firmando JSON (propuesto por RFC7519 [9]) . Después de un inicio de sesión exitoso, el servidor usa la información relevante del usuario para formar en un objeto JSON, luego se cifra de alguna manera devolviendo al cliente. El cliente lleva este token en las próximas solicitudes, y el servidor verifica la validez del token cuando recibe la solicitud.

JWT es una cadena cifrada en formato JSON [10]:

```
1 JSON Data + Signature = JWT
```

Un objeto JWT generalmente consta de tres partes:

1. Encabezados (*Headers*): incluye categoría (*typ*), algoritmo de encriptación (*alg*). El encabezado se utiliza para describir la información más básica sobre el JWT, como su tipo y el algoritmo utilizado para firmarlo.

```
1 {
2   "alg": "HS256",
3   "typ": "JWT"
4 }
```

   En este caso se trata de un JWT y que el algoritmo de firma digital que se utiliza es el HS256.

2. *Claims*: incluye la información del usuario y se puede utilizar para almacenar informaciones insensibles.

```
{
  "iss": "Xiao Peng",
  "iat": 1742343242,
  "exp": 1746666666,
  "aud": "www.example.com",
  "sub": "example@example.com",
  "name": "Xiao Peng",
  "admin": true
}
```

Los primeros cinco campos de abajo están definidos por el estándar JWT:

- iss: el emisor de este JWT
- sub: el usuario para el que está destinado este JWT
- aud: la entidad que recibió el JWT
- exp (expires): cuándo expirar, esta es el *timestamp* de Unix
- iat (issue at): cuándo se emitió, en *timestamp* también

3. Signature:

   Finalmente, cifra la cadena concatenada anterior con el algoritmo especificado por *alg* (HS256) y la clave privada (*Secret*). El contenido cifrado también es una cadena, y la última cadena es la firma. El JWT completo se puede obtener concatenando esta firma detrás de la cadena anterior. Si la parte de *headers* y la parte de *claim* fueron manipulados, este no puede generar una nueva parte de la firma, ya que el manipulador no sabe cuál fue la clave. En JWT, el cuerpo del mensaje es transparente y la firma se utiliza para garantizar que el mensaje no se altere.

```
HMACSHA256(base64UrlEncode(Headers) + '.' + base64UrlEncode(Claims), SECREATE_KEY);
```

3.5.1. Ventajas

- No es necesario guardar la información de la sesión en el lado del servido como se menciona en la ventaja de la sección anterior 3.4, por lo que facilita la escalabilidad de la arquitectura de servidores.
- El *claims* de JWT puede almacenar información común para el intercambio de información. El uso adecuado de JWT puede reducir la cantidad de veces que el servidor consulte la base de datos.

3.5.2. Desventajas

- Problema de tiempo de caducidad: dado que el servidor no guarda el estado de la sesión, es imposible eliminar un Token o cambiar los permisos del Token durante el uso. Es decir, una vez que se emite un JWT, seguirá

siendo válido hasta que caduque, a menos que se implemente una lógica adicional en el lado del servidor. Por lo tanto, si se trata de una aplicación del lado del navegador, el empleo del mecanismo de autenticación JWT también necesita diseñar un mecanismo para la actualización y eliminación activas de JWT, lo que aumenta la complejidad del sistema.

- Seguridad: dado que los *claims* de JWT están codificados en Base64 y no están encriptados, los datos confidenciales no se pueden almacenar en JWT.
- Problema de capacidad: JWT ocupa demasiado espacio, y el límite de *cookies* es generalmente 4 kB, por lo que JWT generalmente se coloca en *localStorage*, y el usuario llevará el JWT en el encabezado cada vez que haga una solicitud HTTP en el sistema. El encabezado de la solicitud HTTP puede ser de tamaño parecido que el cuerpo, incluso más grande.

3.6. Single Sign-on

Single Sign-on, abreviado como SSO, es una de las soluciones más populares para la integración empresarial. La definición de SSO es que en múltiples sistemas de aplicaciones, los usuarios solo necesitan iniciar sesión una vez para acceder a todos los sistemas de aplicaciones de confianza mutua [11].

En 2001, el Comité técnico de seguridad de OASIS lanzó SAML (Security Assertion Markup Language, basado en XML), que propuso las especificaciones técnicas y de seguridad generales para la implementación de SSO [12].

En el protocolo SAML, se acuerdan las tres roles involucradas en SSO: navegador, proveedor de identidad (IdP) proveedor de servicios (SP) y el flujo de comunicación, el método de cifrado y el formato de transmisión de datos entre estas tres entidades [13].

3.7. Social Login

El *social login* o *social sign-on* se refiere a todos los inicios de sesión que no requieren la participación activa del usuario, como la relación vinculante establecida entre el sistema y el usuario. Se le conoce como una implementación de *Single Sign-On* para usuarios finales. La ventaja de *social login* es que puede utilizar la enorme base de usuarios de terceros para promocionar y comercializar el sitio web y, al mismo tiempo, reducir el tiempo de registro e inicio de sesión de los usuarios [14].

3.7.1. OAuth

OAuth es un estándar de desarrollo que permite a los usuarios autorizar sitios web de terceros para acceder a la información que almacenan con otro proveedor de servicios sin tener que tocar un nombre de usuario y una contraseña. Para proteger la seguridad y la privacidad de los datos, los sitios web de terceros deben solicitar autorización explícita a los usuarios antes de acceder a los datos de

los usuarios [15]. Nuestros proveedores comunes de servicios de autenticación OAuth incluyen Google, Facebook, Apple, etc.

El protocolo OAuth tiene dos versiones, 1.0 y 2.0 Todo el proceso de verificación de autorización en la versión 2.0 es más simple y seguro, y actualmente es el método más importante de autenticación y autorización de usuarios.

Los escenarios de aplicación incluyen: acceso a aplicaciones de terceros, autenticación de microservicios, acceso a plataformas de terceros, inicio de sesión con contraseña de terceros, etc.

Modos de autorización:

- *Authorization Code Grant*
- *Implicit Grant*
- *Resource Owner Password Credentials Grant*
- *Client Credentials Grant*

Independientemente del modo de autorización, debe haber cuatro roles necesarios para participar: cliente, servidor de autorización, servidor de recursos y, a veces, usuario (propietario del recurso)

- *Authorization Server* es responsable de emitir tokens de acceso.
- *Resource Owner*, el usuario de la aplicación y es el propietario del recurso y autoriza a otros para acceder a su recurso.
- *Client*, la aplicación que solicita obtener un *access token* y después de la autorización del usuario, emite un *access token* para él. El cliente puede llevar el *access token* al servidor de recursos para acceder a los recursos del usuario.
- *Resource Server* acepta el *access token*, luego verifica el permiso otorgado y finalmente devuelve el recurso.

Un flujo de autorización común de OAuth 2.0 es la que aparece en la siguiente figura 3.4 :

1. En la aplicación debe disponer un enlace que permite que el usuario acceda al enlace de inicio de sesión, el navegador saltará a la página del proveedor y pedirá que el usuario complete la autenticación.
2. El navegador recibe un código de autorización del servidor de autenticación cuando se cierra el enlace anterior.
3. El navegador envía el código de autorización al *back-end* de la aplicación a través de una redirección.
4. El *back-end* envía el código de autorización a servidor de autenticación para que devuelva el *access token* y un *refresh token* (en caso de pedirlo).
5. Finalmente el *back-end* de la aplicación ahora conoce la identidad del usuario y luego puede guardar la información del usuario, redirigir a otras pá-

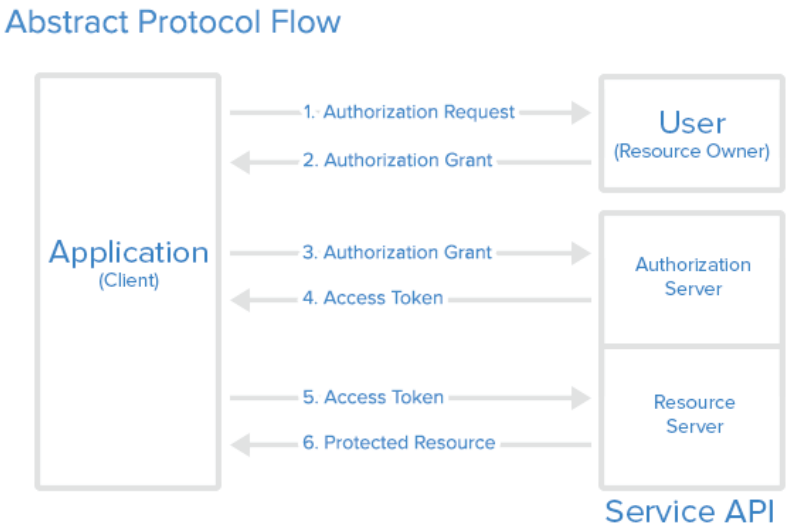

Figura 3.4: Flujo de OAuth 2.0

ginas de *front-end*, usar *access token* para llamar a otras API del servidor de recursos, etc.

3.7.2. OpenID Connect

OpenID Connect (OIDC) es un protocolo de autenticación basado en OAuth 2.0. OIDC también especifica las partes no definidas de OAuth 2.0, como el alcance, el descubrimiento de servicios, los campos de información del usuario, etc [16].

El proceso de autorización de OIDC es el mismo que el de OAuth 2.0, la principal diferencia es que el ID Token se devolverá adicionalmente en el proceso de autorización de OIDC [17].

3.8. Vulnerabilidad

A continuación se detalla los principales tipos de vulnerabilidad que puedan afectar el sistema de autenticación.

3.8.1. Cross-Site Request Forgery

Cross-Site Request Forgery, también conocido como CSRF, es un sitio web o programa malicioso que realiza operaciones anormales en un sitio web confiable a través del navegador de un usuario autenticado. Las acciones maliciosas que se pueden realizar se limitan a las funcionalidades de los usuarios que se han autenticado en el sitio [18].

Es más fácil explicar con un ejemplo:

Un usuario inició sesión en el sitio web de un banco `www.bank.com`, el servidor del banco envió una *cookie*.

```
Set-Cookie:id=ewqrjdsalf;
```

Más tarde, el usuario visitó un sitio web malicioso `www.hacker.com` y había un formulario en este sitio web malicioso.

```
<form action='www.bank.com/transfer' method='POST'>
  ...
</form>
```

Si el usuario activó este formulario sin darse cuenta, el servidor del banco recibirá una solicitud con la *cookie* correcta, y posteriormente el servidor del banco ejecutará la operación de transferencia definida por él mismo. En este caso, es posible que se transferirá el dinero de la cuenta sin que el usuario se entere.

Características principales

- Los ataques generalmente ocurren en sitios web de terceros y no en el web atacado.
- El ataque utiliza las credenciales de inicio de sesión del usuario (*cookie*) en el web atacado para hacerse pasar por la víctima para enviar operaciones en lugar de robar datos directamente.
- Durante todo el proceso, el atacante no puede obtener las credenciales de inicio de sesión, sino que las utiliza de forma fraudulenta.
- La solicitud entre sitios se puede realizar de varias maneras: campo `src` de una imagen, una etiqueta HTML `a`, envío de un formulario `form`, etc.

3.8.2. Cross-Site Scripting

El ataque *Cross-Site Scripting* se abreviaba originalmente como CSS. Pero para distinguirlo de la abreviatura de hojas de estilo (CSS), se abreviaron como XSS.

XSS (*Cross-Site Scripting*), cuya esencia es que el atacante inserta un código de script malicioso (este código puede ser *script* JS, hoja de estilo CSS u otro código inesperado) en la página web, cuando el usuario navega por la página, el código incrustado en el mismo será ejecutada, de manera que logra el propósito de atacar maliciosamente al usuario. Por ejemplo, leer *cookies*, sesiones, tokens u otra información confidencial del sitio web, *phishing* y engañar a los usuarios, etc [19].

Supongamos que tenemos un sistema de comentarios:

Un usuario A envía el comentario “Hola” al servidor, y luego el usuario B visita el sitio web y ve el comentario de A “Hola”, en este caso no hay XSS.

Un usuario malicioso H envía el comentario:

```
<script>console.log(document.cookie)</script>
```

Luego el usuario B visita el sitio web. Este *script* se ejecuta directamente en el navegador de B, y el script del usuario H puede manipularse la *cookie* de B sin

que el usuario B lo sepa. Con la *cookie*, el usuario malicioso H puede falsificar la información de inicio de sesión de B y acceder a los recursos autorizados de B.

Características principales

- Robar la información de la *cookie* a través de `document.cookie`
- Usar JS o CSS para destruir la estructura y el estilo normales de la página
- Secuestro de tráfico (saltar a otras páginas usando una instrucción de JS `window.location.href`)
- Ataque DoS: usa solicitudes de clientes legítimos para ocupar demasiados recursos del servidor, de modo que los usuarios legítimos no puedan obtener respuestas del servidor. Y al llevar la información de la *cookie* del proceso, el servidor puede devolver un código de estado que comienza con 400, rechazando así una solicitud de servicio razonable.

3.9. Tecnologías

A continuación se enumeran y explican las tecnologías utilizadas en el proyecto, así como los beneficios que nos aportan.

3.9.1. TypeScript

La definición de TypeScript según la página oficial [20]:

"TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale."

Destaca las dos características más importantes de TypeScript: el sistema de tipos y que funciona a cualquier escala.

Typed programming language

Como puede ver en el nombre de TypeScript, *type* es su característica principal.

Sabemos que JavaScript es un lenguaje de programación muy flexible:

- No tiene restricciones de tipo y una variable puede inicializarse como una cadena y luego asignarse a un número.
- Debido a la existencia de conversión de tipos implícita, es difícil determinar el tipo de algunas variables antes de ejecutarlas.
- Programación orientada a objetos basada en prototipos, de modo que las propiedades o los métodos del prototipo se puedan modificar en tiempo de ejecución.
- Las funciones son ciudadanos de primera clase en JavaScript [21] y se pueden asignar a variables, como parámetros o valores de retorno.

Esta flexibilidad es como un arma de doble filo. Por un lado, JavaScript ha florecido y es omnipotente. Desde 2013 ocupa el primer lugar en la lista de los lenguajes de programación más utilizados [22]; por otro lado, también hace que la calidad de su código sea desigual, los costos de mantenimiento sean altos y los errores de tiempo de ejecución sean altos.

El sistema de tipos de TypeScript compensa en gran medida las deficiencias de JavaScript.

- **Tipado estático** El tipado estático significa que el tipo de cada variable se puede determinar en la etapa de compilación y los errores de tipo en este lenguaje a menudo conducen a errores de sintaxis. TypeScript debe compilarse en JavaScript antes de ejecutarse, y la verificación de tipo se realizará durante la fase de compilación, por lo que TypeScript se escribe estáticamente y este código de TypeScript informará un error durante la fase de compilación:

```
1 let foo = 1;
2 foo.split(' ');
```

- **Tipado débil** El sistema de tipos se clasifica según si se permite la conversión implícita de tipos y se puede dividir en tipado fuerte y tipado débil.

  El siguiente código puede ejecutarse normalmente tanto en JavaScript como en TypeScript. El número 1 se convertirá implícitamente en la cadena '1' en tiempo de ejecución, y el signo más + se reconoce como concatenación de cadenas, así que imprima El resultado es la cadena '11'.

```
1 console.log(1 + '1');
```

  TypeScript es totalmente compatible con JavaScript, no modifica las características del tiempo de ejecución de JavaScript, tanto uno como otro es tipado débil.

  Dicho sistema de tipos incorpora el concepto de diseño central de TypeScript [23]: sobre la base de preservar por completo el comportamiento del tiempo de ejecución de JavaScript, mediante la introducción de un sistema de tipos estáticos para mejorar la capacidad de mantenimiento del código y reducir posibles errores.

Tooling at any scale

TypeScript es muy adecuado para proyectos grandes: es obvio que el sistema de tipos puede conducir a una mayor capacidad de mantenimiento y menos errores para proyectos grandes.

El mayor obstáculo para implementar TypeScript en proyectos pequeños y medianos es la creencia de que usar TypeScript requiere escribir código adicional, lo que reduce la eficiencia del desarrollo. Pero, de hecho, debido a *type inference*, la mayoría de los tipos no necesitan declararse manualmente. Por el contrario,

TypeScript mejora las funciones del editor (IDE), incluida la finalización de código, sugerencias de interfaz, saltos a definiciones, refactorización de código, etc., lo que mejora en gran medida la eficiencia del desarrollo. Y TypeScript tiene casi cien opciones de compilación. Si cree que la verificación de tipos es demasiado estricta, puede reducir el estándar de verificación de tipos modificando las opciones de compilación.

TypeScript también puede coexistir con JavaScript. Esto significa que si tiene un proyecto antiguo desarrollado en JavaScript y desea utilizar las funciones de TypeScript, entonces no necesita apresurarse para migrar todo el proyecto a TypeScript, puede escribir nuevos archivos en TypeScript y luego migrar gradualmente en iteraciones posteriores. Si el costo de migración de algunos archivos JavaScript es demasiado alto, TypeScript también proporciona una solución que le permite escribir un archivo de declaración de tipo sin modificar los archivos JavaScript para lograr una migración incremental de proyectos antiguos.

3.9.2. React

React, como sugiere su eslogan oficial, es una biblioteca para construir interfaces de usuario [24]. React no es un *framework*: su aplicación ni siquiera se limita al desarrollo web, se puede usar con otras bibliotecas para renderizar en entornos específicos. Por ejemplo, React Native se puede utilizar para crear aplicaciones móviles; React 360 se puede emplear para crear aplicaciones de realidad virtual, etc.

React fue originalmente una herramienta de desarrollo utilizada internamente por Facebook, pero era un proyecto con objetivos elevados: "*Learn once, write anywhere*". Desde la publicación como código abierto en 2013, el ecosistema circundante ha florecido. El surgimiento de ReactJS ha traído muchas ideas innovadoras para el desarrollo *front-end*. Hay varias características importantes que vale la pena discutir:

Pensamiento basado en componentes

- Diseño de interfaz de usuario declarativo con JSX
- Uso de DOM virtual
- El componente es como una máquina de estado (*State Machine*), pero también tiene un ciclo de vida (*Life Cycle*)
- Flujo de datos unidireccional
- Código CSS dentro de JavaScript (*inline style*)

En el mundo React, como se muestra en la figura 3.5 la unidad más básica es un componente. Cada componente también puede contener más de un subcomponente, que se puede ensamblar en un componente combinado de acuerdo con los requisitos, por lo que tiene las características de encapsulación, separación de preocupaciones, reutilización y combinación.

JSX

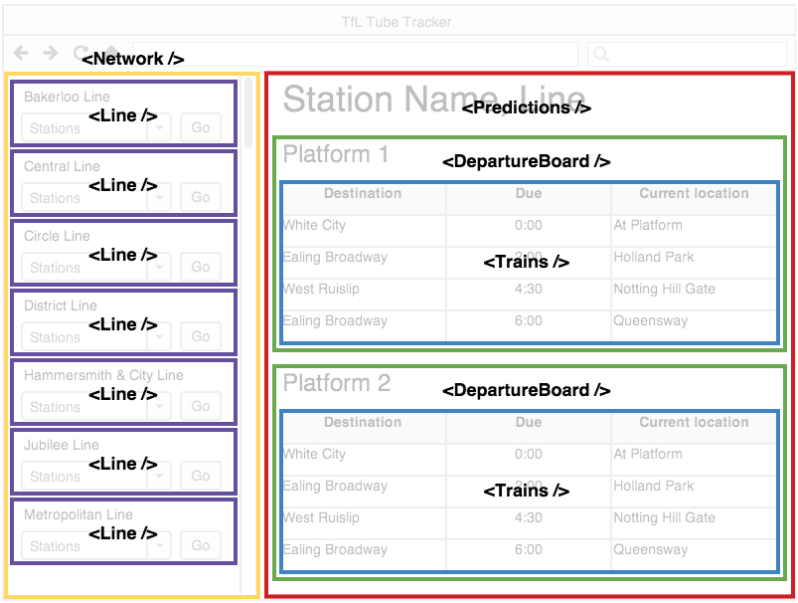

Figura 3.5: Web basado en componentes

JSX es una extensión de la sintaxis del lenguaje JavaScript. JSX es visualmente similar a HTML y proporciona una forma de crear una vista de componentes utilizando una sintaxis familiar para muchos desarrolladores. Los componentes de React generalmente se escriben en JSX, aunque esto es opcional, ya que los componentes también se pueden escribir en JavaScript puro.

```
const todolist =
  <TodoList>
      <Item>Task 1</Item>
      <Item>Task 1</Item>
      <Item>Task 2</Item>
  </TodoList>
```

React Hooks

React Hooks es una nueva característica introducida a partir de la versión React 16.8 [25], el propósito es resolver el problema de compartir el estado de React y la confusión de la gestión del ciclo de vida de los componentes. La aparición de React Hooks marca que React ya no tendrá componentes sin estado, y React solo tendrá el concepto de componentes de clase y componentes de función.

En el desarrollo de la aplicación React, compartir el estado de los componentes es algo muy problemático, y React Hook solo comparte la lógica de procesamiento de datos, no los datos en sí, por lo que no hay necesidad de preocuparse por la vinculación de datos y ciclo de vida. A continuación se muestra un ejemplo de implementación de un contador utilizando un componente de clase.

```
class Example extends React.Component {
```

```
2    constructor(props) {
3      super(props);
4      this.state = {
5        count: 0
6      };
7    }
8
9    render() {
10     return (
11       <div>
12         <p>You clicked {this.state.count} times</p>
13         <button onClick={() => this.setState({ count: this.state.count + 1 })}>
14           Click me
15         </button>
16       </div>
17     );
18   }
19 }
```

Se puede encontrar que los componentes de clase necesitan declarar su propio estado, escribir métodos para operar el estado y también necesitan mantener el ciclo de vida del estado, lo cual es particularmente problemático. Si se usa el `useState()` para manejar el estado, el código será mucho más simple. El código refactorizado se muestra a continuación.

```
1  import React, { useState } from 'react';
2
3  function Example() {
4    const [count, setCount] = useState(0);
5
6    return (
7      <div>
8        <p>You clicked {count} times</p>
9        <button onClick={() => setCount(count + 1)}>
10         Click me
11       </button>
12     </div>
13   );
14 }
```

Como puede ver, *Example* ha cambiado de un componente de clase a un componente de función. Este componente de función tiene su propio estado y puede actualizar su propio estado sin llamar al método `setState()`. Esto funciona porque el componente de clase usa la función `useState()`.

La siguiente especificación de Hooks [26] describe el patrón de código de firma para Hooks, la forma en que React se usa hoy en día para manejar el estado.

- Los Hooks solo se pueden invocar en el nivel superior (prohibido dentro de bucles o declaraciones *if*).
- Los Hooks solo se pueden llamar desde componentes de funciones de React

y Hooks customizados, no desde funciones de componentes o funciones normales.

Si bien estas especificaciones no se pueden aplicar en el momento de la ejecución, las herramientas de análisis de código (como linters) se pueden configurar para detectar muchos errores durante el desarrollo. Estas especificaciones se aplican al uso de Hooks y la implementación de Hooks personalizados, [27] que pueden invocar a otros Hooks.

3.9.3. Next.js

Aunque los *front-end* modernos basados en JavaScript como React nos ayudan a desarrollar sitios web dinámicos y potentes, tienen una gran desventaja: el cliente debe esperar hasta que se cargue todo el código JavaScript antes de renderizar la página y mostrar el contenido al usuario. Esto se ha convertido en un problema importante en el desarrollo web, ya que la velocidad de carga de una página puede afectar significativamente la experiencia del usuario y SEO.

Next.js es un *framework* de renderizado en el lado del servidor basado en React [28] y proporciona una solución a este problema. Promete aumentar el rendimiento del sitio web con un conjunto de funciones fáciles de usar, incluida la renderización previa. Teniendo en cuenta la comunidad de desarrolladores que pululaba poco tiempo después del lanzamiento del framework.

Caracteristicas principales

La función más destacada de Next.js es la renderización previa. Como se mencionó anteriormente, renderizar una página web en el lado del cliente después de recibir el código JavaScript es un proceso lento. Next.js resuelve este problema enviando al cliente una versión renderizada previamente de cada página.

Utiliza tres tipos de renderizado previo:

- Generación de estática de página

```
export async function getStaticProps(context) {
  return {
    props: {}, // will be passed to the page component as props
  }
}
```

- Renderización al lado del servidor

```
export async function getServerSideProps(context) {
  return {
    props: {}, // will be passed to the page component as props
  }
}
```

- Regeneración estática incremental

Si bien la generación de sitios estáticos es mejor para crear páginas web estáticas, la renderización al lado del servidor es mejor para páginas con contenido dinámico. Lo que tiene de especial Next.js es que te permite decidir individualmente qué tipo de renderización se va a utilizar para cada página web.

Además de la renderización previa, Next.js proporciona un conjunto de características cuidadosamente diseñadas para simplificar el proceso de desarrollo que se detalla a continuación.

Empaquetado automático y división de código: a diferencia de React, donde Webpack debe configurarse para agrupar el código manualmente, Next.js usa Webpack para agrupar el código automáticamente. También admite la división de código basada en rutas separadas e importaciones dinámicas. Reduce el tiempo de carga de páginas web y componentes dinámicos.

- **Optimización de imágenes:** Next.js admite de forma nativa el cambio de tamaño y la optimización de imágenes. Puede emparejarlo con la opción de carga diferida predeterminada para obtener el mejor rendimiento.
- **Actualización rápida (*fast refresh*)**: cuando el código cambia durante el desarrollo, los componentes se vuelven a renderizar inmediatamente.

3.9.4. React Native

React Native [29] permite a los desarrolladores usar JavaScript y React para desarrollar nuestras aplicaciones. React Native proporciona componentes similares a la que tenemos en React con los componentes de HTML, incluso podemos anidar componentes relacionados para formar componentes nuevos como funciona en React. Usando React Native podemos mantener el mismo código de lógica para múltiples plataformas (Web, Android e IOS) para crear aplicaciones nativas. Hoy en día, la experiencia de la aplicación móvil basada en web aún no puede alcanzar la experiencia de la aplicación nativa, por lo que Facebook enfatiza el pensamiento de "*Learn once, write everywhere*". La idea central de React Native es que no solo tiene la experiencia de usuario de Native, sino que también conserva la eficiencia de desarrollo de React.

Una de las ventajas de usar React Native es compartir código entre diferentes dispositivos. Pero se limita a la lógica de sistema, ya que dependiendo de las plataformas, sea web, Android o iOS, poseen cualidades diferentes, así como mecanismos para salvar datos temporales (localStorage, Cookie, IndexedDB, etc.) o componentes visuales, por lo que se tiene que dar un paso más para abstraer códigos que sean comunes entre las plataformas, sobre todo los flujos comunes.

Vistas

En el navegador, JavaScript puede llamar a la API DOM para completar la creación de la interfaz de usuario, mientras que en React Native mediante se utiliza *UI manager* para crear vistas. Basado en Virtual DOM, React Native encapsula la lógica de generar vistas en diferentes plataformas, en concreto llama a UI Manager a través de Bridge para producir diferentes vistas nativas.

Hilos de ejecución

En React Native, en realidad hay tres subprocesos importantes en ejecución: el hilo Shadow, el hilo UI y el hilo JavaScript como se muestra en la figura 3.6

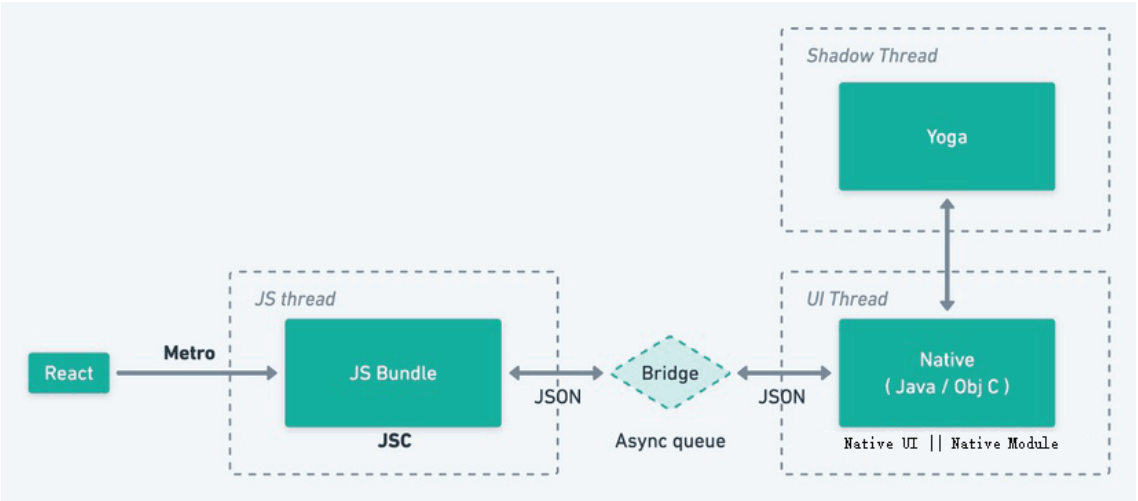

Figura 3.6: Hilos de ejecución de React Native

- **JS thread:** es un hilo de JavaScript y es el responsable de la interacción entre JS y el código nativo. Debido a que JS es un modelo de ejecución de un solo hilo, necesita un hilo separado para controlar y a demás la interacción entre JS y Native es asíncrona.
- **Shadow thread:** este hilo de ejecución es el responsable del *layout* del Native.
- **UI thread:** se puede ver como el hilo principal, es el encargado de administrar la interfaz de usuario, responsable de la interacción de la página y la de renderizar componentes.

React Fiber

Fiber es el nuevo motor de reconciliación adoptado en React 16, el objetivo principal es admitir la renderización progresiva del DOM virtual.

Fiber reemplazó el Stack Reconciler original por Fiber Reconciler, lo que mejora la capacidad de respuesta y el rendimiento de las aplicaciones complejas.

Debido al nuevo método de programación de Fiber, el proceso de actualización de las tareas puede interrumpirse, lo que significa que el procesamiento y sus funciones de ciclo de vida pueden llamarse varias veces durante el proceso de actualización del componente. Por lo tanto, se podrá evitar los efectos secundarios generados del ciclo de la vida.

3.9.5. Expo

Expo es un *framework* y una plataforma para las aplicaciones de React Native. El Expo SDK proporciona componentes o herramientas React Native y Native para ayudar a los desarrolladores a desarrollar, construir, implementar e iterar rápidamente en iOS, Android y aplicaciones web usando solo JavaScript/TypeScript [30].

Al desarrollar, crear y publicar aplicaciones basadas en React Native, es posible que deba modificar el código nativo o la configuración de la aplicación, lo que requiere instalar Xcode o Android Studio, configurar el entorno y estar familiarizado con el desarrollo de iOS o Android. Hay un costo considerable para el desarrollador.

Expo ya los ha integrado y permite concentrarse en el desarrollo de código de JavaScript/TypeScript. Puede importar componentes React Native o Native desde Expo SDK. Además, puede descargar la aplicación Expo Go, que es conveniente para depurando el código y previsualizando el efecto en la máquina real.

3.9.6. GraphQL

GraphQL es un estilo de consulta de API orientado a datos [31]. La API tradicional obtiene el formato de datos acordado por el *front-end* y *back-end*. GraphQL proporciona una descripción completa y fácil de entender de los datos en la API. El cliente puede obtener con precisión los datos que necesita sin ninguna redundancia como se muestra en la figura 3.7 comparando con RESTful.

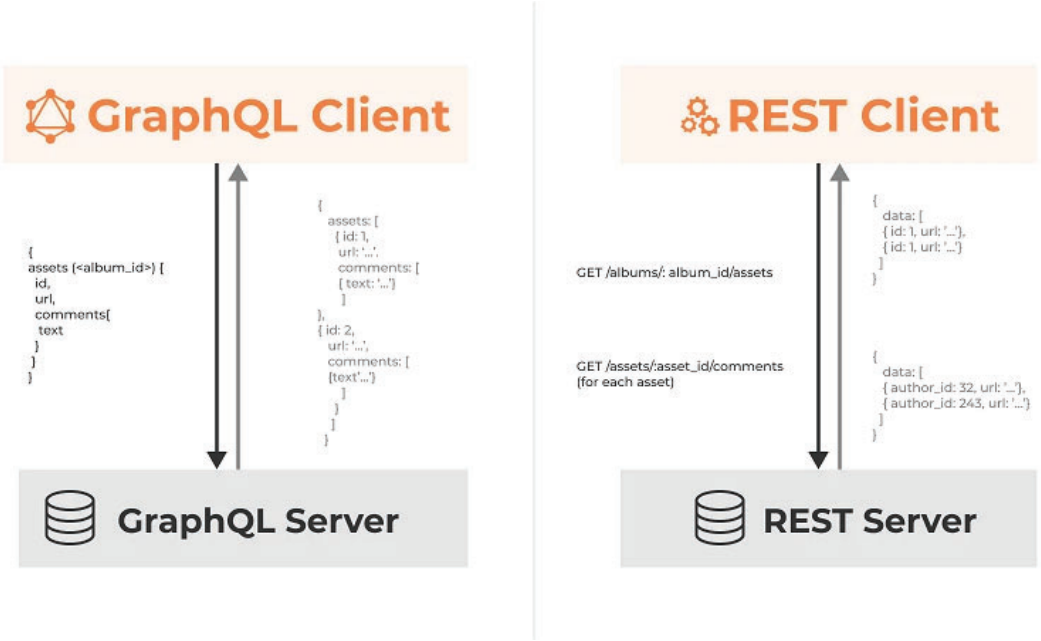

Figura 3.7: Comparación entre GraphQL y RESTful

Con la popularización del *framework* SPA en el desarrollo *front-end*, el desarro-

llo basado en componentes también se ha convertido en la tendencia general. Cada componente administra su propio estado. El basado en componentes brinda comodidad al *front-end* y también trae algunos problemas. Por ejemplo, un componente debe ser responsable de distribuir el estado de las solicitudes asíncronas a los componentes secundarios o notificar a los componentes principales. En este proceso, la complejidad estructural causada por la comunicación entre componentes, las fuentes de datos de origen desconocido y las respuestas de datos de fuentes desconocidas hará que el flujo de datos se desorganice, lo que también hace que el código sea menos legible y mantenible, lo que trae grandes dificultades para la iteración futura del proyecto.

En algunos escenarios complejos, el *back-end* debe proporcionar API correspondientes a WebApp, WebPC, APP, etc. En este caso, la granularidad de la API es particularmente importante. La granularidad gruesa generará tráfico de redes innecesarias en el *front-end*. Con este escenario, los ingenieros de Facebook libero al público la especificación de GraphQL en 2015 [32], lo que permitió que el *front-end* describiera el formulario de datos que desea y que el *back-end* devuelva la estructura de datos descrita por el *front-end*.

Los siguientes puntos son conceptos básicos e importantes para entender GraphQL.

Operation Type

El tipo de operación de GraphQL puede ser *query*, *mutation* o *subscription*, y que describe qué tipo de operación desea realizar el cliente.

- **query:** obtener datos, como búsqueda, R (*read*) del CRUD (*create*, *read*, *update*, *delete*)
- **mutation:** realiza cambios en los datos, como agregar, eliminar, modificar, CUD (*create*, *update*, *delete*) del CRUD
- **subscription:** notifica cuando los datos cambian

Por ejemplo, en el siguiente código realice una operación de consulta:

```
query {
  user { id }
}
```

Object Type and Scalar Type

Si un servicio de GraphQL recibe una *query*, la consulta buscará desde la *root query*. Cuando se encuentra el tipo de objeto (*Object Type*), su función de resolución (*Resolver*) se usa para obtener el contenido. Si se devuelve el tipo de objeto, la función de resolución se utiliza para obtener el contenido. Mientras que si devuelve tipo escalar (*Scalar Type*) se finaliza la adquisición.

- **Object Type:** el tipo definido por el usuario en el esquema
- **Scalar Type:** GraphQL tiene tipos escalares incorporados *String*, *Int*, *Float*, *Boolean*, *ID*, y los usuarios también pueden definir sus propios tipos escalares.

```
type User {
    id: ID!
    name: String!
    age: Int
    gender: Gender
    email: String!
    emailConfirmed: Boolean!
    provider: String!
}
```

El tipo de objeto *User* tiene dos campos, el campo *Name* es un escalar de cadena que no acepta valores nulos y el campo de edad es un escalar de *Int* que acepta valores nulos.

Schema

Schema define el tipo de campos, la estructura de los datos y las reglas para solicitar datos de la interfaz. Cuando hacemos alguna consulta incorrecta, el motor GraphQL se encargará de decirnos dónde hay un problema y la información detallada del error, lo cual es muy amigable al desarrollo y depuración.

Schema usa una sintaxis de esquema fuertemente tipada simple llamada lenguaje de definición de esquema (SDL), y podemos usar un ejemplo real para mostrar cómo se escribe un archivo *schema* real en SDL:

```
# Query
type Query {
    users: [User]!
    user(id: ID!): User!
}

# Mutation
type Mutation {
    login(email: String!, password: String!): LoginResult!
    logout: String!
    register(
        email: String!
        password: String!
        name: String!
  ): LoginResult!
}

# Subscription
type Subscription {
    subsUser(id: ID!): User
}

type User implements UserInterface {
    id: ID!
    name: String!
    age: Int
    gender: Gender
    email: String!
```

```
29      emailConfirmed: Boolean!
30      provider: String!
31  }
32
33  type LoginResult {
34      user: User!
35      accessToken: String!
36      accessTokenExpiresAt: Int!
37      refreshToken: String!
38      refreshTokenExpiresAt: Int!
39  }
40
41  # enum type
42  enum Gender {
43      MAN
44      WOMAN
45  }
46
47  # interfaces
48  interface UserInterface {
49      id: ID!
50      name: String!
51      age: Int
52      gender: Gender
53  }
```

Este código define varios tipos de objetos o tipos escalares a partir de las entradas *query*, *mutation* y *subscription*. Los tipos de estos campos también pueden ser otros tipos de objetos o tipos escalares, formando una estructura similar a un árbol. Al enviar una solicitud, se pueden obtener conjuntos de información seleccionando una o más ramas a lo largo del árbol.

Capítulo 4

Desarrollo

En este capítulo se ofrecerá un resumen en las configuraciones, decisiones en cuanto a la seguridad y diseños de bajo nivel desarrollados para llevar a cabo el proyecto.

4.1. Configuración del entorno de trabajo

En esta sección se va a explicar las configuraciones de entorno de trabajo importantes que se ha seguido en el proyecto.

4.1.1. Monorepo

En los escenarios generales de desarrollo, los desarrolladores esperan que cada proyecto pueda ser lo suficientemente independiente, y que su desarrollo y lanzamiento respectivos no produzcan demasiado acoplamiento.

Sin embargo, el modo anterior parecerá ineficiente y engorroso en algunos escenarios. Por ejemplo, muchos otros repositorios relacionados hacen referencia al código de un repositorio, por lo que mientras se publique este repositorio, todos los repositorios que dependen de este código también se actualizarán y publicarán en consecuencia. Si todo el código con dependencias se coloca en un almacén para un mantenimiento unificado, cuando cambia una biblioteca, otros códigos pueden actualizar automáticamente sus dependencias, entonces el proceso de desarrollo se puede simplificar y se puede mejorar la eficiencia del desarrollo. Este repositorio de código de paquetes múltiples se llama *monorepo*.

Se ha optado este tipo administración para desarrollar el código, ya que la idea principal es la necesidad de desarrollar lógicas que sean capaz de compartir entre las aplicaciones de Next.js y React Native como librerías externas. En este caso se ha instalado Turborepo [33] para establecer la estructura del *monorepo*.

Estructura del proyecto:

```
.
|-- apps
|   |-- mobile
|   |   |-- src
|   |   |   |-- components
|   |   |   |-- screens
|   |   |   |-- mocks
|   |   |-- app.tsx
|   |-- web
|       |-- public
|       |-- pages
|       |-- src
|           |-- components
|           |-- mocks
|-- packages
    |-- shared-lib
    |   |-- src
    |       |-- generated
    |       |-- grahql
    |       |-- hooks
    |       |-- jest
    |       |-- mocks
    |-- config
```

4.1.2. Mock Service Worker

Una parte inevitable del desarrollo es que necesitamos un servidor API para asegurarnos de que nuestro código *front-end* funcione. Y tendríamos los siguientes problemas: qué sucede si el servidor API aún no está listo, dónde debería nuestro código de vistas enviar sus solicitudes de API, etc.

Podemos resolver este problema con la simulación de las APIs. Básicamente, es solo un servidor "falso" al que las vistas pueden realizar solicitudes HTTP.

Hay varias herramientas de simulación de API en el mercado y están resolviendo el mismo problema de diferentes maneras. Algunos le permiten configurar un servidor simulado Node.js real. Algunos interceptarán solicitudes de obtención y las redirigirán a funciones de controlador que existen en el *front-end*.

Mock Service Worker (MSW) [34] es una herramienta de simulación de API que utiliza Service Workers para interceptar sus solicitudes HTTP. Esto le permitirá realizar solicitudes HTTP reales que se pueden inspeccionar con DevTools de los navegadores como se muestra en la figura 4.1.

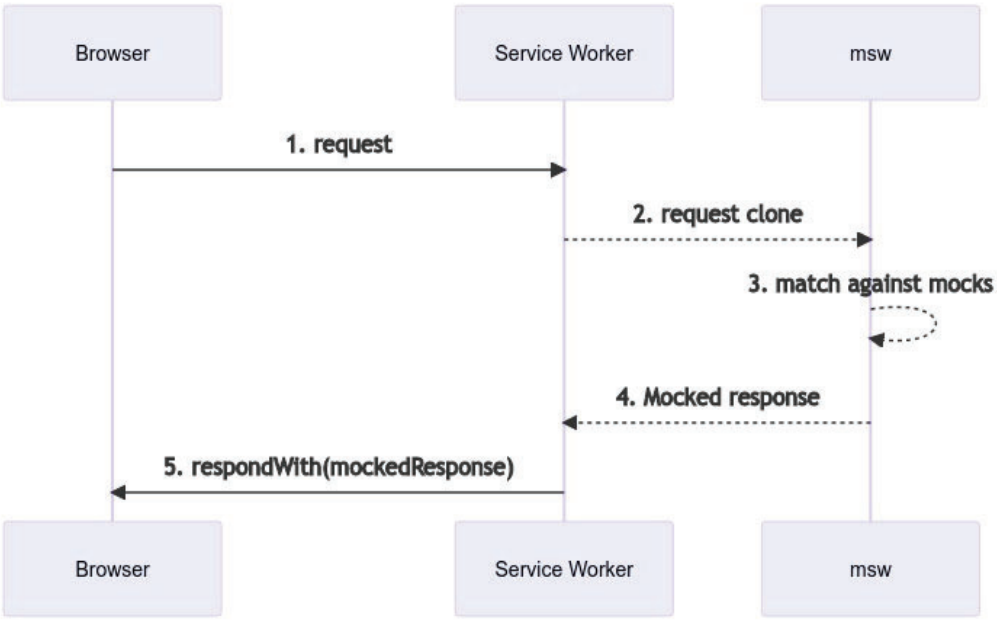

Figura 4.1: Flujo de MSW

MSW también se puede usar en el código de testing con las siguientes secciones 4.1.3 y 4.1.4 para que no tenga que configurar simulacros de prueba adicionales para las respuestas HTTP [35].

4.1.3. Pruebas unitarias

Las pruebas son una parte importante del trabajo de desarrollo de software. Se clasifican de varias maneras según diferentes dimensiones. Según la fase de prueba, existen principalmente pruebas unitarias, pruebas de integración y pruebas de sistema, y las pruebas unitarias son las más fundamentales para garantizar la corrección básica del programa.

La prueba unitaria es un método de prueba para la parte más pequeña del programa para verificar si el código funcionará como se espera. En la programación de procedimientos, el más pequeño es una función, en la programación orientada a objetos, el componente más pequeño es un método de objeto.

Para las pruebas unitarias se ha elegido Jest, es un *framework* para *testing* creado por Facebook. En comparación con otros *frameworks*, una de sus características principales es que es una herramienta de prueba que integra las funcionalidades comunes, como aserciones, *mocks*, cobertura de pruebas, etc [36].

4.1.4. Pruebas de integración y E2E

Las pruebas E2E no son misteriosas y probablemente cualquier desarrollador lo habría realizado manualmente sin darse cuenta. Por ejemplo, para la fun-

ción “iniciar sesión”, podemos simular el comportamiento del usuario con los siguientes pasos:

1. Pulsar el botón *Sign In* para iniciar sesión.

2. Ingresar nombre de usuario y contraseña en los campos.

3. Hacer clic en el botón enviar.

4. Si tiene éxito, salta a la página de recursos y el icono de usuario se mostrará en el lado derecho de la navegación, si falla, aparecerá un mensaje de error indicando el fallo.

Las pruebas de automatización E2E es convertir los pasos manuales a códigos como el siguiente ejemplo, en este caso se ha utilizado Cypress como el *framework* de E2E para Next.js [37]:

```
it('Perform Sign in', () => {

  cy.visit('http://web/')

  cy.contains('Sign In').click()

  cy.url().should('include', 'signin')

  cy.get('#user-name').type("xiaopeng")

  cy.get("#password").type('my_password')

  cy.get('.btn').click()
}
```

De esto se puede ver que la mayor diferencia entre *E2E Testing* y *Unit Testing* es que E2E realmente simula el comportamiento del usuario, y también puede ser la navegación entre páginas, mientras que pruebas unitarias es para probar un comportamiento o función con diferentes propósitos.

4.2. Resumen de los requisitos

En la figura 4.2 se resumen brevemente mediante un diagrama de caso de uso las relaciones que hay entre cada uno de los requisitos funcionales que se había detallado en el capítulo 2.

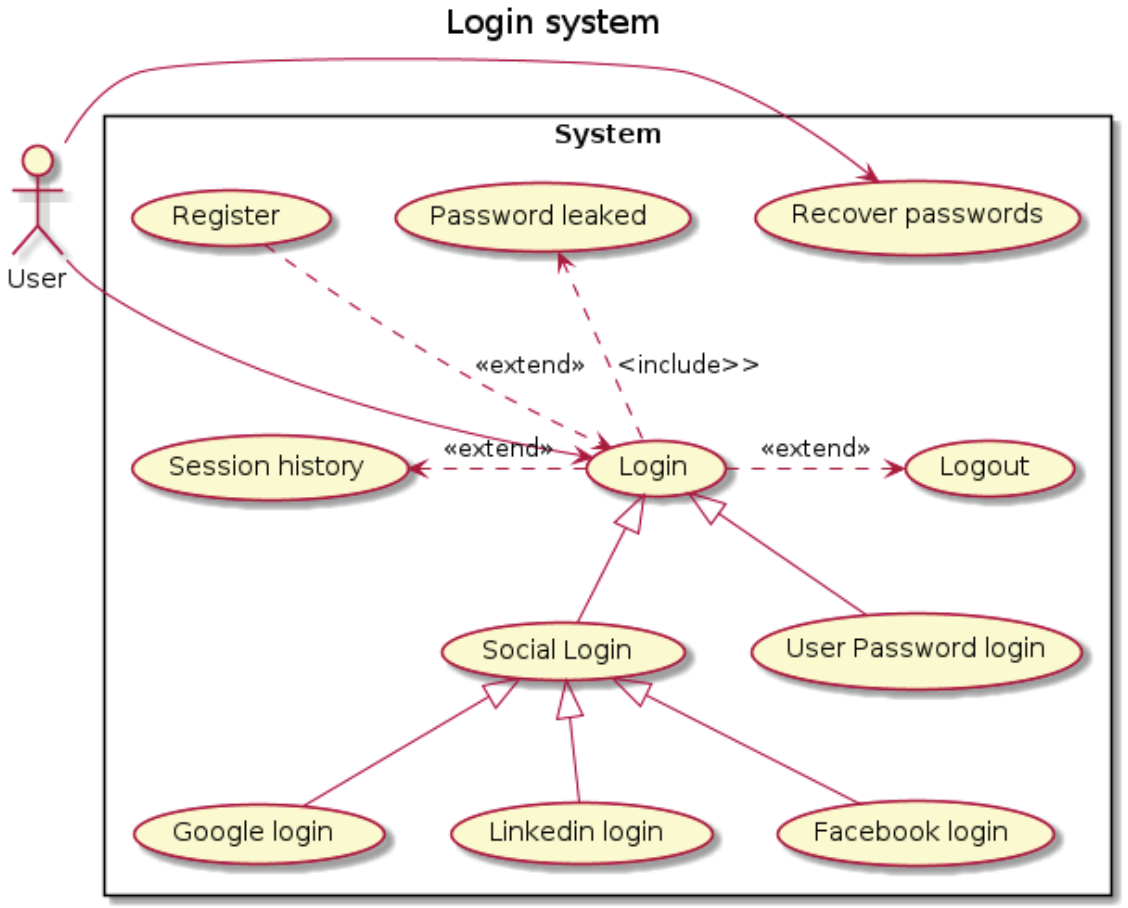

Figura 4.2: Caso de uso

4.3. Decisiones tomadas del diseño

En cuanto a la gestión de acceso a los recursos, se ha utilizado un sistema de autenticación basado en JWT más concreto formado por un *access token* cuyo formato es JWT, y el *refresh token* que es el usado para actualizar los tokens una vez expirado.

Respecto a los tiempos de expiración se ha utilizado los siguientes criterios:

- El período de tiempo de JWT en general debe ser breve (menor de 30 minutos) y usar la *mutation* de GraphQL *refresh token* para obtener un nuevo *access token*.
- El periodo de *refresh token* debe ser duradera, en este caso se ha utilizado una semana y de validez única.

Según la sección 10 de RFC6749 [38] (OAuth 2.0), el servicio de autorización debe mantener la relación estrecha entre *refresh Token* y el cliente, por lo que sugiere que el usuario legítimo se debe identificarse a base de IP u otros datos

del usuario en la llamada del *refresh token*.

La ventaja de que el *refresh token* sea válido una vez es que nos garantiza que en el caso de ataque como el siguiente ejemplo: si tanto el usuario legítimo y como el atacante tiene el *refresh token* e intenta obtener el *access token*, pues no nos importa quien sea el primer usuario en acceder a la llamada del API, ya que se informara la invalidez de *access token* y les obligara a refrescar el *refresh token* mostrando también la invalidez de este, por lo que les obliga a iniciarse la sesión desde el principio con los datos de autenticación.

En cuanto a la seguridad de los *endpoints* de GraphQL se usara los siguientes criterios:

- Cualquier petición se deben usar HTTPS como la capa de seguridad en caso contrario rechazaría la petición.
- Para acceder a las operaciones o recursos autorizados se pasara el *access token* como *headers* de la petición.

```
GET / HTTP/1.1
Authorization: access token
```

En los siguientes apartados se resumen las medidas de seguridad que se ha tomado en cuanto a la gestión de tokens tanto para Next.js como React Native.

4.3.1. Next.js

- Se ha utilizado las *cookies* HttpOnly como muestra en la figura 4.3 para salvar el *resfresh token* y nos evitamos de los ataques XSS (sección 3.8.2), ya que las *cookies HttpOnly* son inaccesibles desde la API de JavaScript como con las instrucciones. `Document.cookie;` únicamente se enviarán al servidor [6].

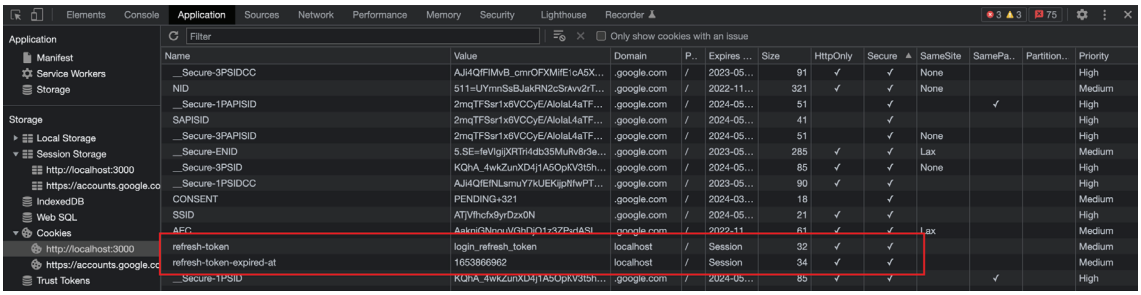

Figura 4.3: Cookies

- Mientras que para el *access token* se ha usado *session storage* (figura 4.4), ya que son tokens con tiempo de expiración corta y en el *session storage* toda la información almacenada será eliminada al finalizar la sesión de la página, es decir al cerrar las páginas [39].

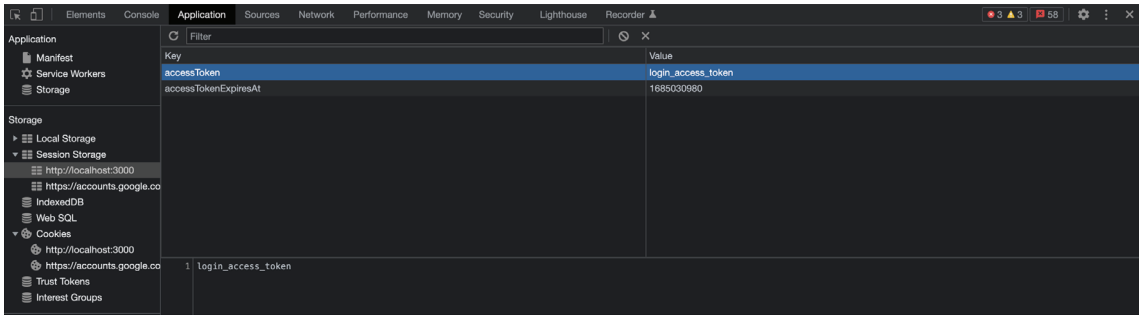

Figura 4.4: Session storage

4.3.2. React Native

Tanto para *acess token* como *refresh token* se ha utilizado Secure Store [40], es una SDK para gestionar los datos de tipo clave-valor de forma encriptada y es válida tanto para la plataforma IOS como Android por lo que ideal para salvar los tokens de manera segura.

4.4. Diseño de los flujos

Para representar los flujos de las llamadas de API y su relación con el *back-end* se ha utilizado diagramas secuenciales para llevar a cabo. Los diagramas siguientes son generadas con PlantUML [41], una herramienta *open-source* para crear diagramas de UML basado en el uso de un lenguaje declarativo a diferencia de las herramientas tradicionales de *click and drag* como Visio.

A base de los flujos siguientes se ha implementado los Hooks personalizados que permiten extraer la lógica de los componentes de la vista y son los usados para compartir las mismas lógicas tanto en Next.js como React Native.

4.4.1. Registro de cuenta

La figura 4.5 corresponde a los flujos necesarios cuando un usuario intenta crear una cuenta nueva:

1. Se debe comprobar mediante la API gratuita de “Have I Been Pwned” [42] que la contraseña no tuvieron fugas y están públicas en Internet.

2. En caso de que no fue filtrada la contraseña se procederá llamar el *endpoint* de *back-end* para registrar la contraseña.

3. En caso de que fuera una cuenta nueva, pedirá que el usuario revise su correo para confirmar el correo.

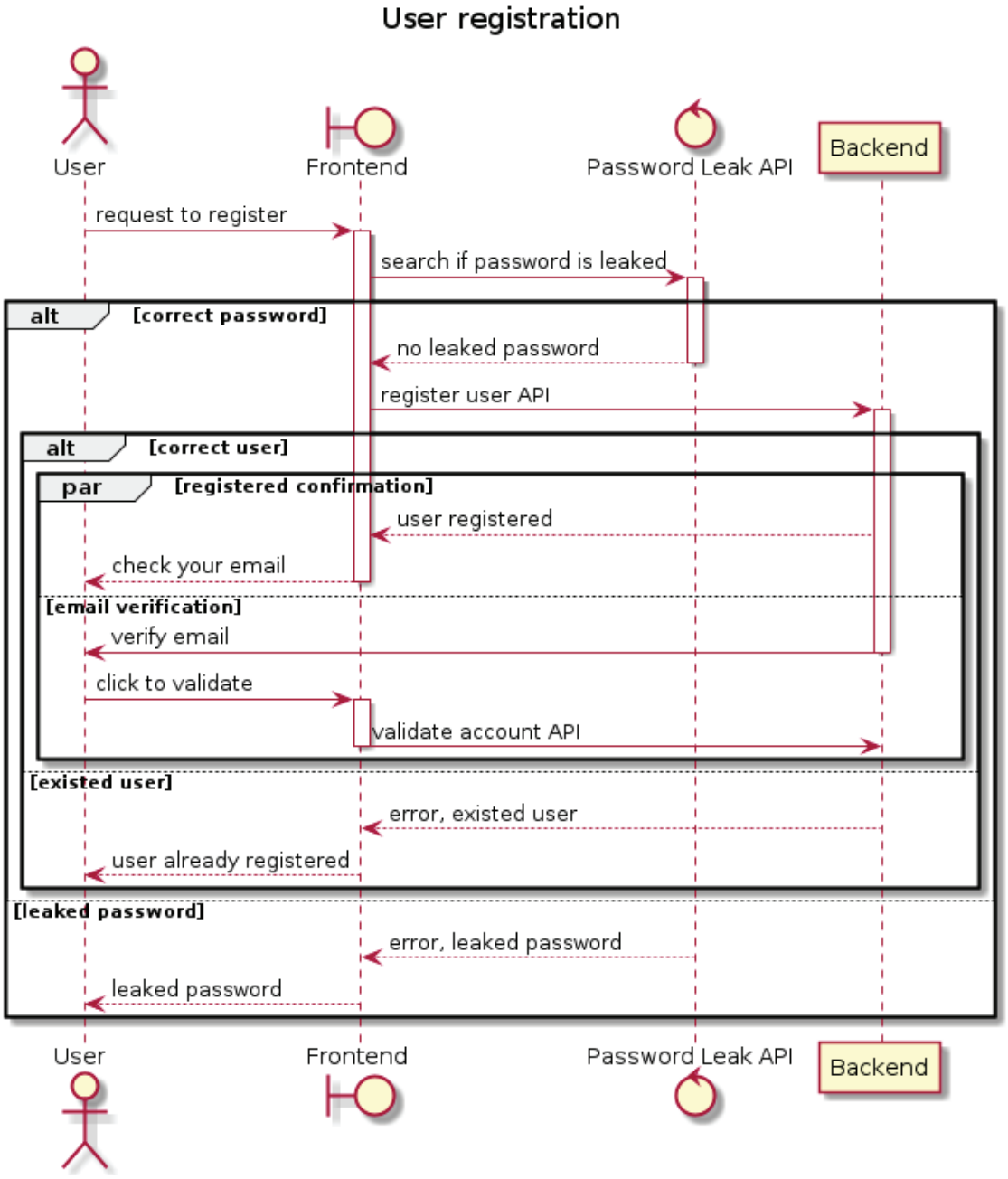

Figura 4.5: Flujo para registrar cuenta nueva

4.4.2. Iniciar sesión

Una vez registrado la cuenta, el usuario podría proceder a iniciar la sesión con los datos anteriores como aparece en la figura 4.6.

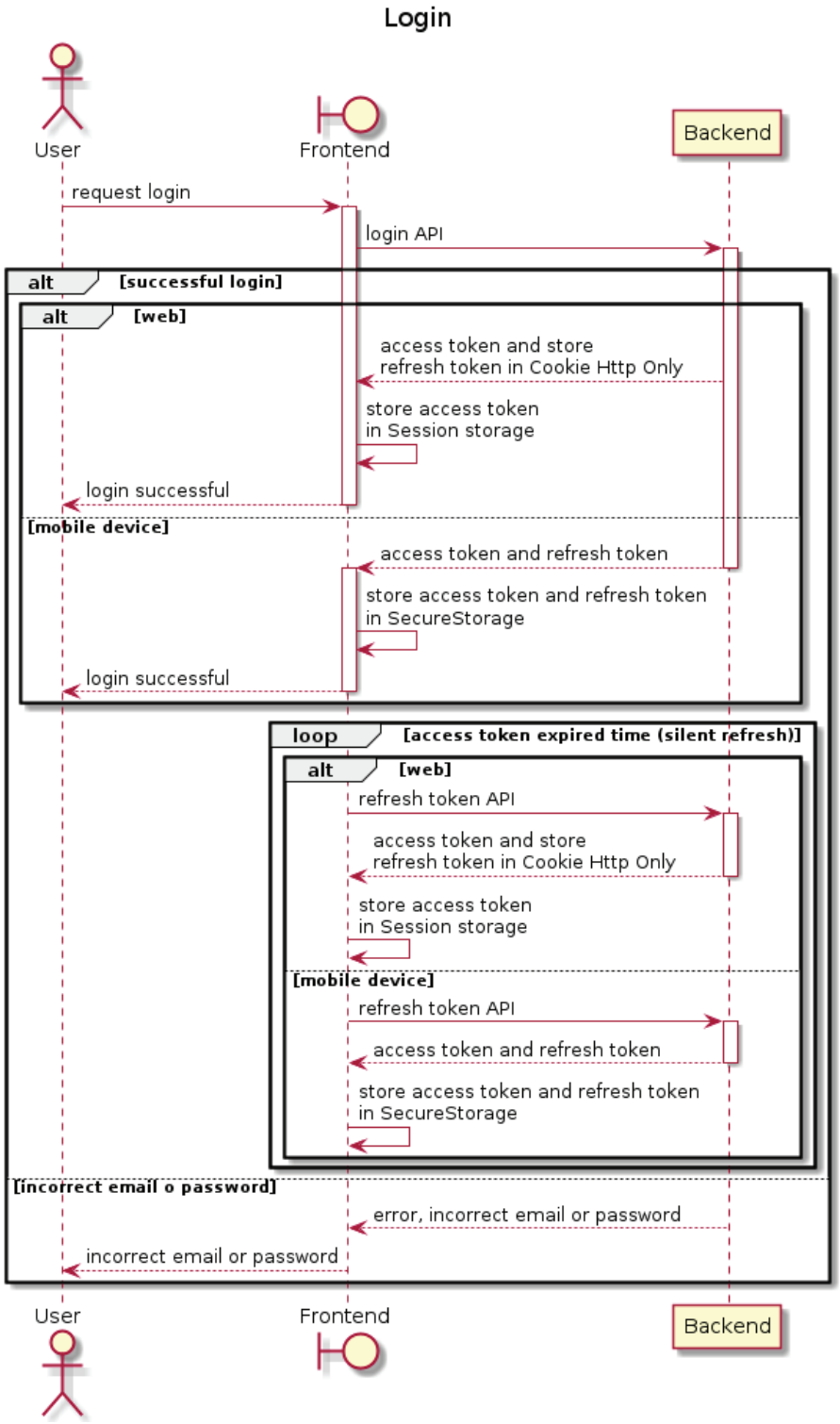

Figura 4.6: Flujo para iniciar sesión

1. En primer lugar llama el *enpoint* correspondiente del *back-end*.
2. Dependiendo del cliente, la salvación tanto para *access token* y *refresh to-*

ken se procesan de manera diferente.

3. En el caso de Next.js el *refresh token es guardado en Cookie HttpOnly* y el *access token* en la *sesssion storage*.

4. En el caso de React Native, los dos tokens son salvados en el *secure storage*

4.4.3. Cerrar sesión

Después de iniciar la sesión, el usuario es capaz de cerrar sesión como muestra en la figura 4.7.

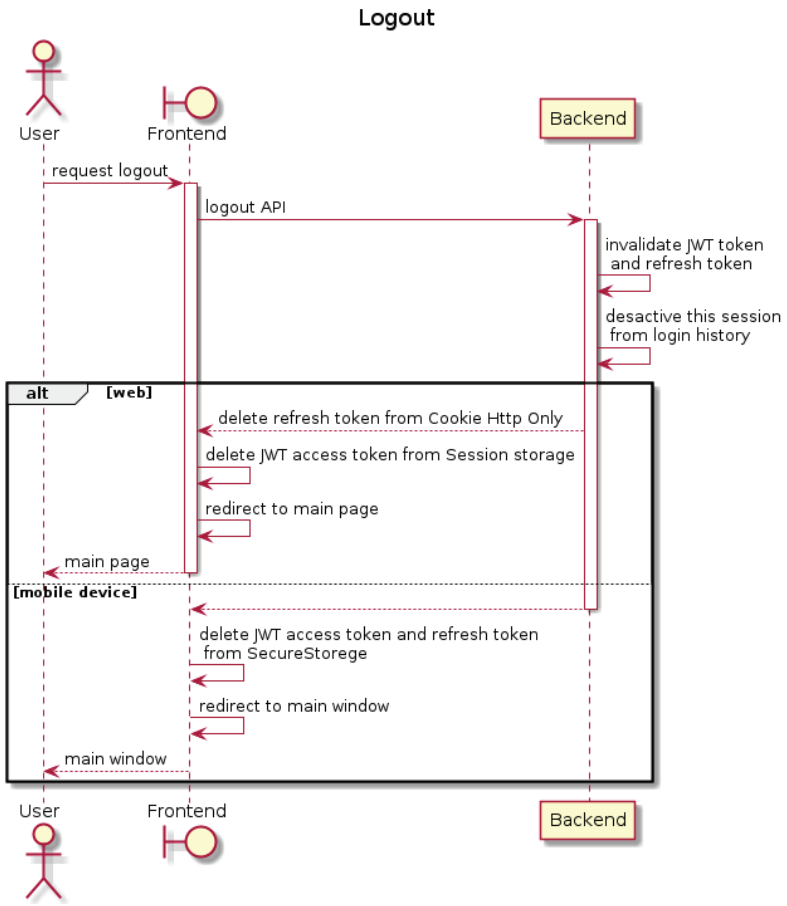

Figura 4.7: Flujo para salir de sesión

1. La llamada del *endpoint* de *logout* es meramente para avisar a la parte de *back-end* la invalidación de los tokens y proceder a eliminar los tokens en el *front-end*.

2. En caso de Next.js se necesita la ayuda del *back-end* para borrar el *refresh token* al ser *HttpOnly* la *Cookie*.

3. Mientras en React Native lo podría procesar de forma autónoma.

4.4.4. Social login

Si el usuario prefiere usar entidades de tercero como Google, Facebook y LinkedIn para iniciar la sesión sin tener que registrarse, tendríamos el flujo de la siguiente figura 4.8.

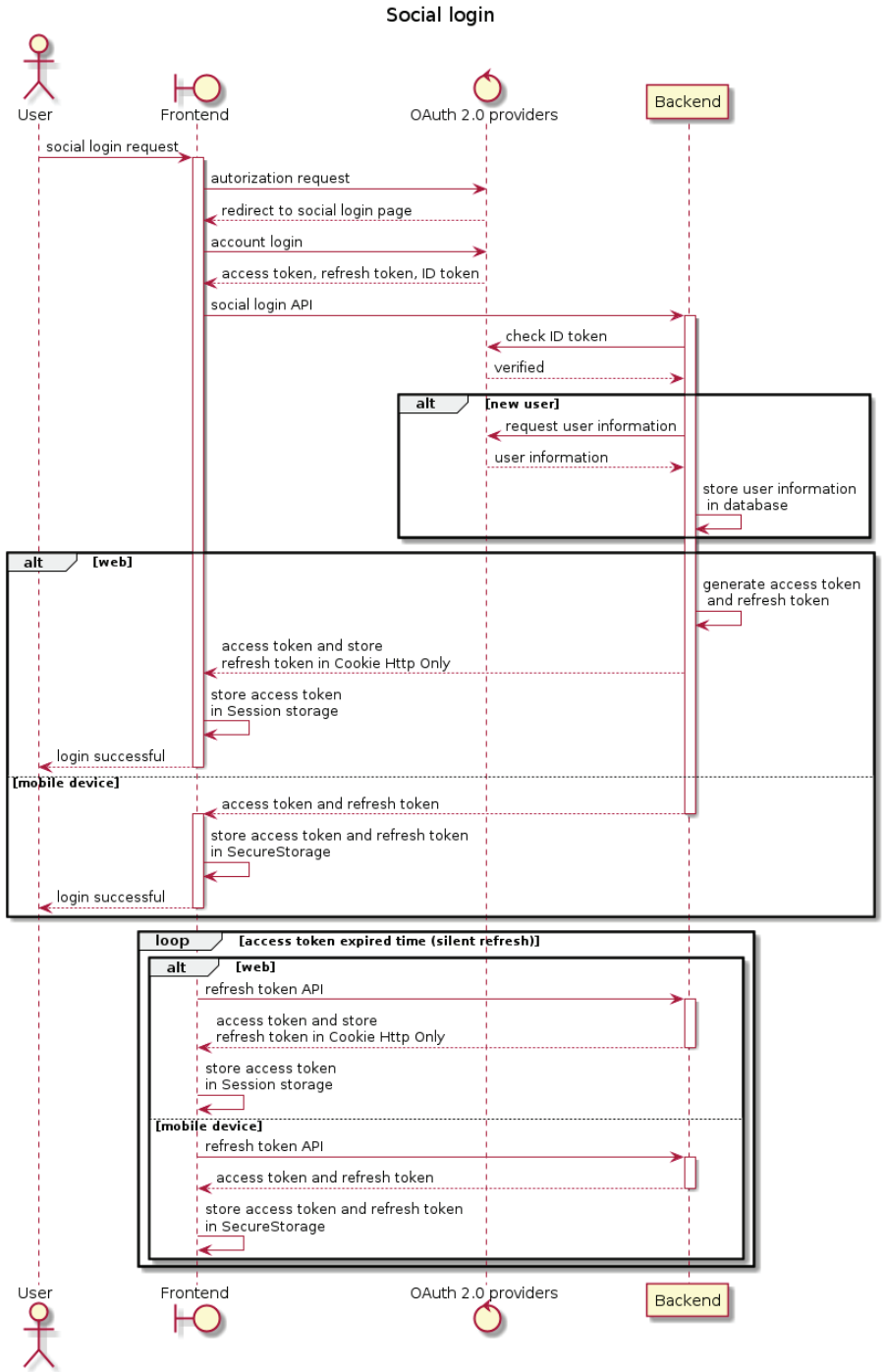

Figura 4.8: Flujo de social login

1. Primero se le redirige al usuario a la página del proveedor para autentificarse.
2. Devolvería los tokens necesarios para realizar la llamada del *endpoint*.
3. Se comprobaría la existencia del usuario en el *back-end* y se crearía si no existiera.
4. De la misma forma que pasa en los flujos de inicio de sesión de la sección 4.4.2, los tokens son tratados de forma diferente según la plataforma.

4.4.5. Acceso a recursos autorizados

Una vez iniciado la sesión tanto por el *login* tradicional como por el *social login* es necesario establecer mecanismos y flujos con los tokens para acceder los recursos, ya que como comentaba en la sección 4.3 los tokens posen un tiempo de expiración.

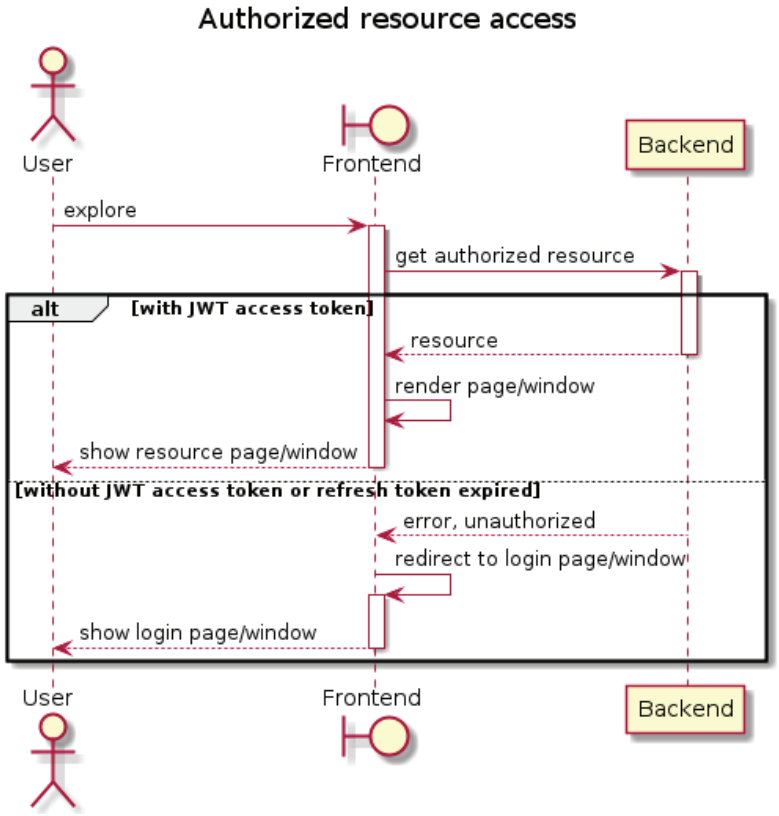

Figura 4.9: Flujo para acceso a recursos autorizados

- En el caso de Next.js realizaría la llamada de `refreshToken` cada vez que se acceda a una página autorizada, ya que Next.js no puede conseguir el *access token* desde las funciones de *server side rendering*.
- En React Native comprobaría la existencia y la expiración en el *secure storage* como aparece en la figura 4.9.

4.4.6. Restablecer y cambiar la contraseña

Se debe garantizar a los usuarios la recuperación de contraseña cuando se olviden de contraseña o que cuando lo quieran cambiar por cuestión de seguridad mediante el flujo 4.10.

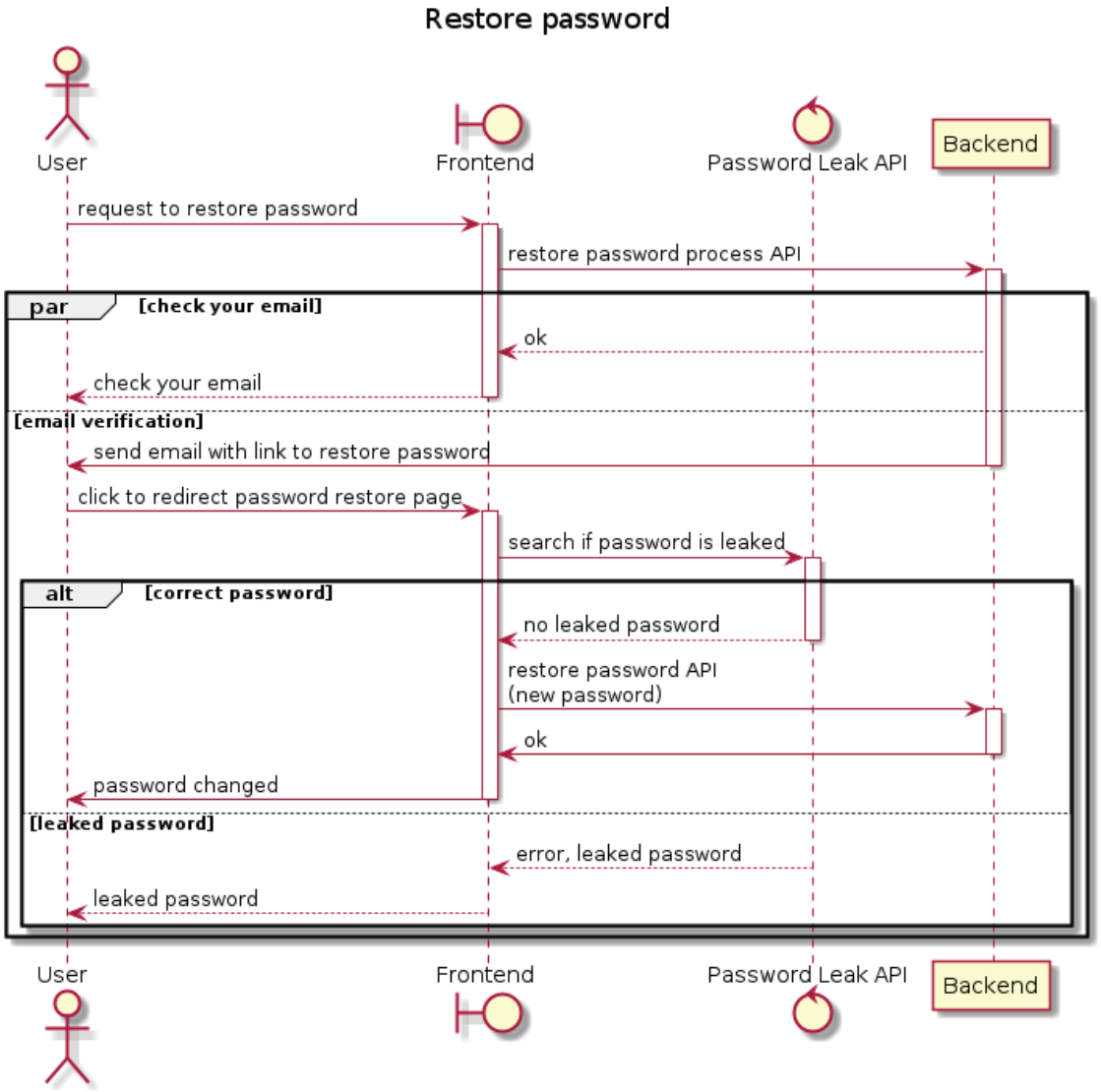

Figura 4.10: Flujo para restablecer la contraseña

1. El proceso de restablecimiento es similar cuando se intenta registrar una cuenta nueva.
2. Comprueba la filtración de la contraseña.
3. Enviar a la cuenta del correo un enlace para restablecer la contraseña.

Mientras que para cambiar la contraseña lo haría una vez iniciado la sesión,

en una vista específica para ello con un simple flujo de llamar únicamente el *endpoint* de GraphQL.

4.4.7. Añadir correo y contraseña

Resulta que el caso de *social login* como la de Facebook no nos compromete la existencia de un correo como para verificar la entidad del usuario, por lo que tendríamos que pedir al usuario manualmente en nuestra aplicación con el flujo 4.11. Es decir, se podría crear una cuenta en Facebook solo con el número de telefono por lo que Facebook se puede autenticar el usuario sin correo asociado. Debido a la necesidad del proyecto necesitamos enviar notificaciones, *newsletter*, etc. al usuario y que en un futuro nos podría servir para implementar funcionalidades como MFA.

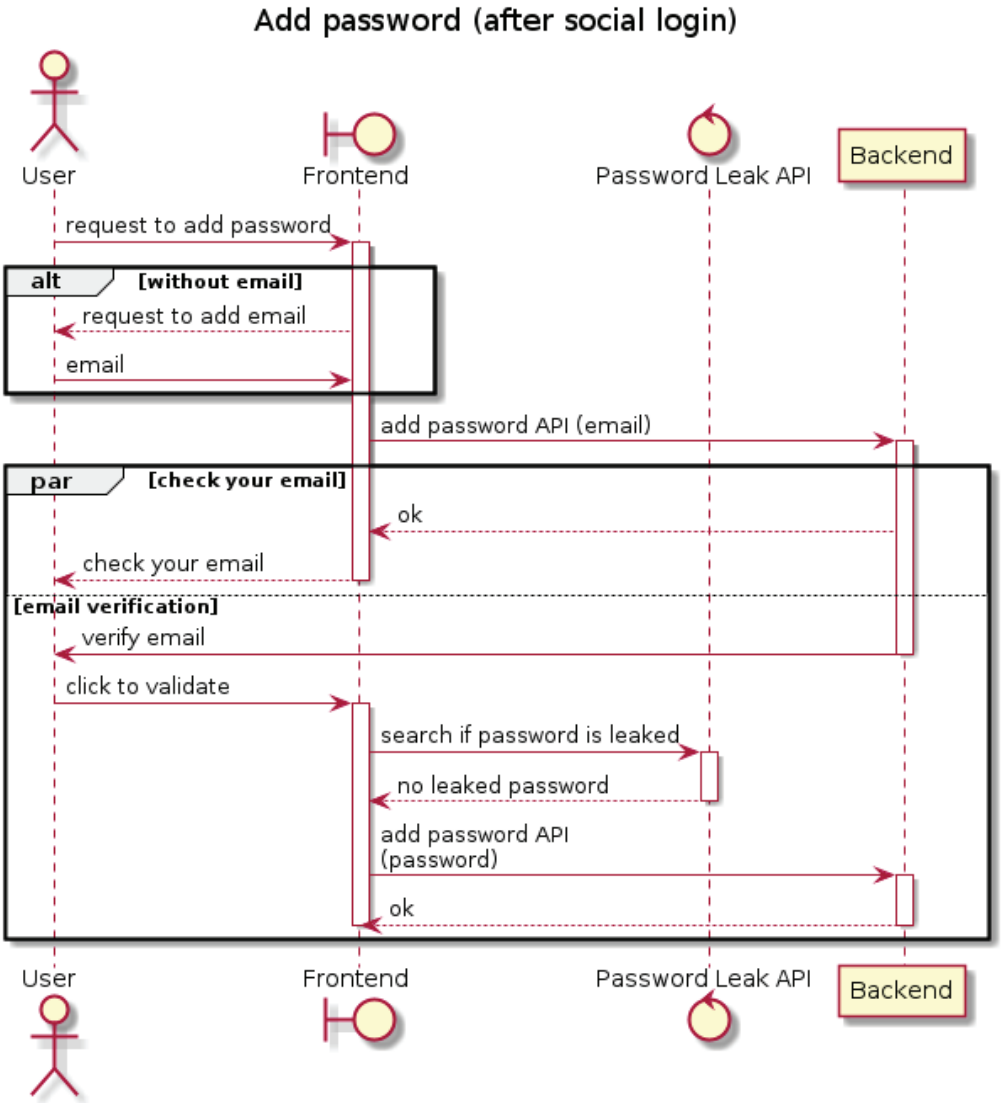

Figura 4.11: Flujo para añadir correo

1. Se registraría el correo que se quiera verificar y envía el correo de confirmación.
2. En el correo tendría el enlace para redirigir a la página para añadir la contraseña.

4.4.8. Controlar sesiones

Una vez iniciado la sesión, el usuario tiene el derecho de visualizar las sesiones activas de la misma cuenta y el poder de invalidar la sesión con el flujo 4.12

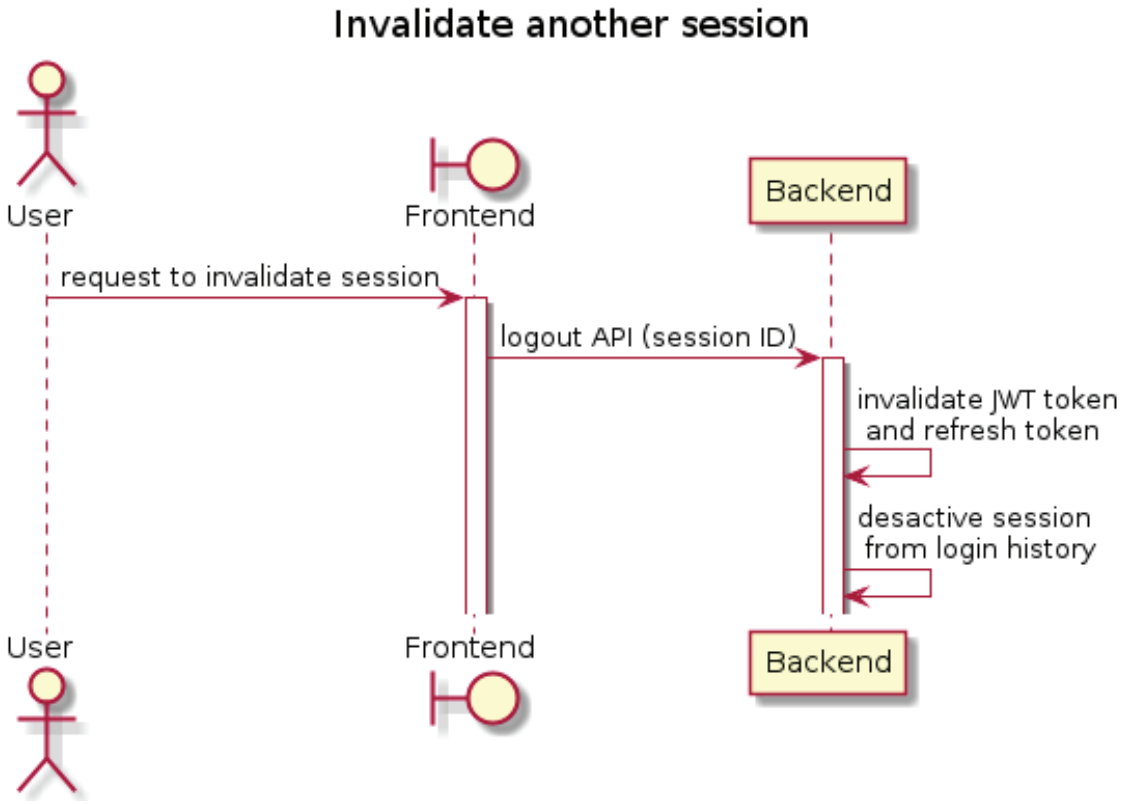

Figura 4.12: Flujo para controlar sesiones

El flujo es similar al de *logout*, pero con la necesidad de pasar como argumento el ID de la sesión que se quiera invalidar.

4.5. Esquema de GraphQL

En este apartado se ha definido los *endpoints* necesarios, consultas de tipo *query* y *mutation*, para los flujos diseñados de la sección anterior 4.4.

Para generar códigos de consulta basada en la librería de Apollo Client [43] en TypeScript se ha utilizado la herramienta GraphQL Code Generator [44]. Con

el Code Generator nos evitaríamos codificar todas las definiciones de interfaces de tipo en TypeScript, siendo esto un trabajo repetitivo y costoso de hacerlo manualmente.

4.5.1. Query

En la figura 4.13 se detalla las consultas de tipo *query* y sus relaciones con los nodos. El tipo *User* define los datos básicos de un usuario, mientras que el *Session* representa una sesión de un usuario específico, como podemos observar los campos que relacionan los dos tipos, es decir el usuario contendrá un conjunto de sesiones. El *Location* es el nodo reutilizado por los dos campos de *lastSeen* y *originalSigIn* para representar sobre zona geográfica y el tiempo de registro.

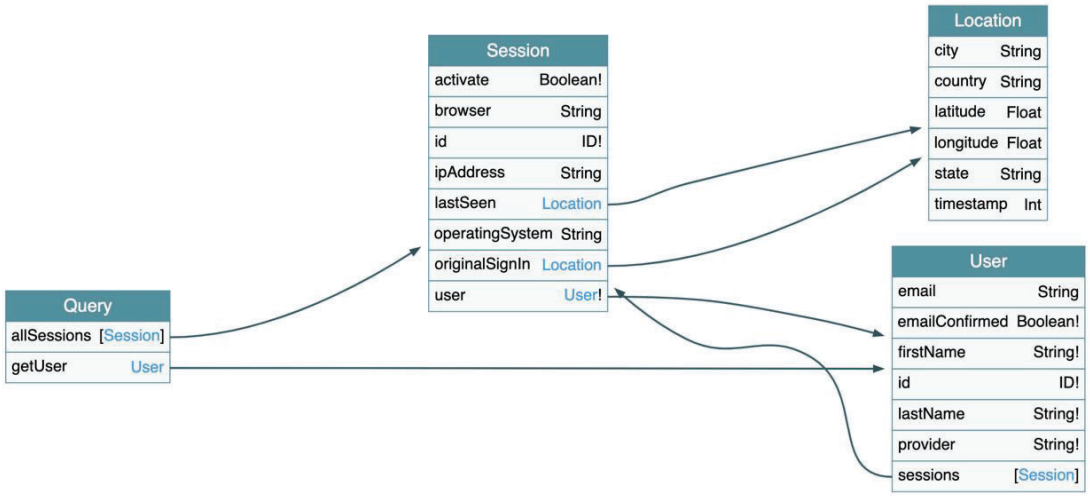

Figura 4.13: Queries

- **allSessions:** es la consulta para obtener todas las sesiones de un usuario especificado mediante el *header* `Authorization` (*access token*) de la solicitud.
- **getUser:** la consulta que devuelve los datos básicos de un usuario (se requiere *access token* como la consulta anterior).

4.5.2. Mutation

Como se puede observar en la figura 4.14 aparecen aquí también los dos tipos *User* y *Session* anteriores. Aparte de estas dos se deben destacar los esquemas de *LoginResult* y *RefreshTokenResult*, que contienen los tokens necesarios para acceder a los recursos autorizados.

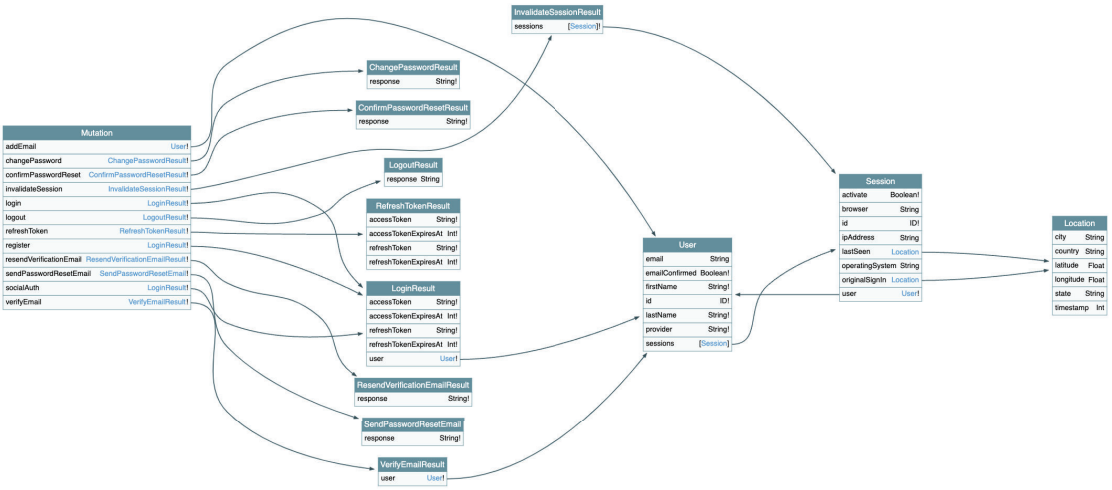

Figura 4.14: Mutations

- **addEmail:** *endpoint* para añadir el correo cuando pertenece una cuenta iniciada mediante *social login*.
- **changePassword:** es el *endpoint* que modifica las contraseñas mediante el *access token*.
- **confirmPasswordReset:** se usa este *endpoint* cuando el usuario recibe un enlace para redirigir a la página donde pueda restablecer la contraseña.
- **invalidateSession:** es similar al *endpoint* del *logout* pero con parámetro el ID de la sesión.
- **login:** *mutation* para iniciar la sesión y devuelve los tokens necesarios para los accesos posteriores.
- **logout:** cerrar la sesión.
- **refreshToken:** la llamada del *endpoint* nos devuelve los tokens actualizados con nuevo tiempo de expiración.
- **resendVerificationEmail:** *endpoint* auxiliar cuando el usuario no consigue recibir el correo en el registro de la cuenta y desea reenviarlo.
- **socialAuth:** gestiona el inicio de sesión con social login de cualquier proveedor.
- **verifyEmail:** *endpoint* usado en el enlace recibido después de registrarse para verificar la cuenta del correo.

4.6. Vistas utilizadas

En cuanto a la implementación de las componentes visuales se ha utilizado la librería de componentes Material UI [45] para crear las vistas en Next.js y la

librería NativeBase [46] para React Native.

En esta sección, se comenta a través de las capturas de pantallas de la aplicación Web el funcionamiento con respecto a los flujos de la sección 4.4.

En primer lugar, para registrar una nueva cuenta debemos acceder a la página específica como aparecen en la figura 4.15. Nos pedirá las informaciones básicas como el nombre y apellidos, la dirección de correo y la contraseña. Se verificara localmente que los campos no son vacíos y que el correo insertado cumpla con el formato estándar, mientras que la contraseña se obligara al usuario usar combinaciones mayúsculas, minúsculas, numéricos y caracteres especiales mediante la comprobación de una expresión regular. Y que el primer campo de la contraseña introducida debe coincidir con el segundo campo de la contraseña de confirmación.

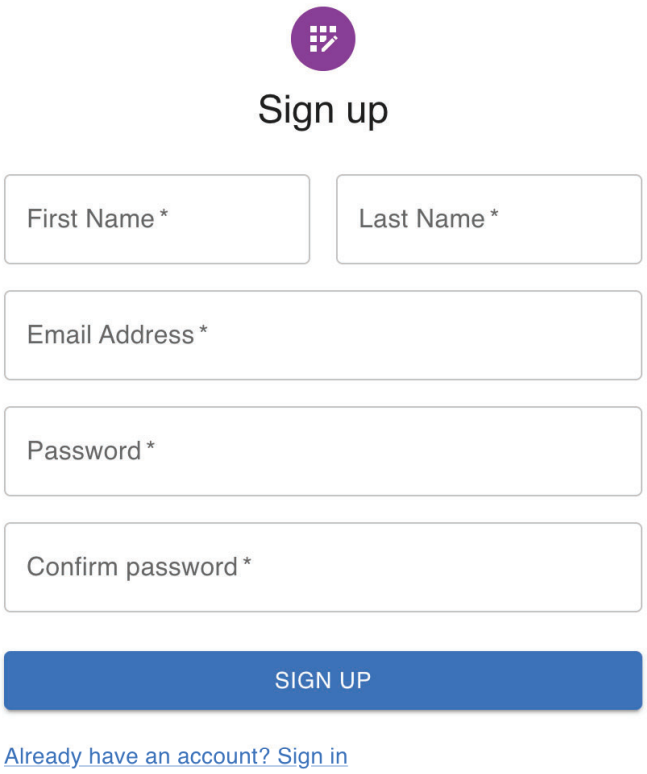

Figura 4.15: Vista para registrar cuenta nueva

Si ya disponemos una cuenta podemos redirigir a la vista de *login* 4.16. Nos pedirá el correo y la contraseña, y comprobará que los campos no son vacíos. Si preferimos usar *social login* disponemos los tres botones con los correspondientes proveedores.

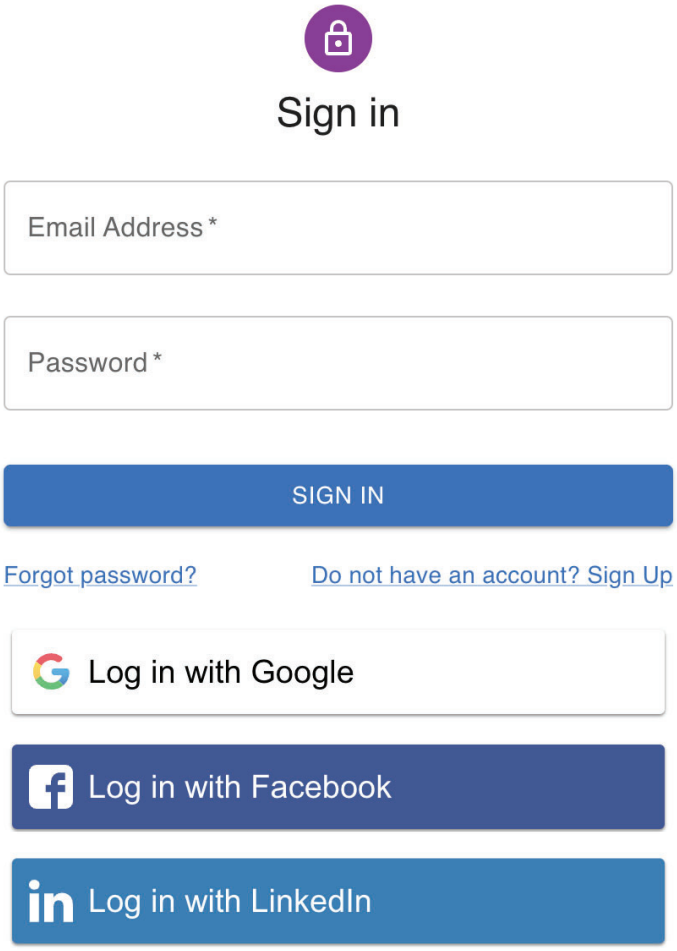

Figura 4.16: Vista para iniciar sesión

Tomando como ejemplo el *social login* de Google, nos redirige a la página de autenticación 4.17 para preguntarnos con la cuenta que se quiera acceder.

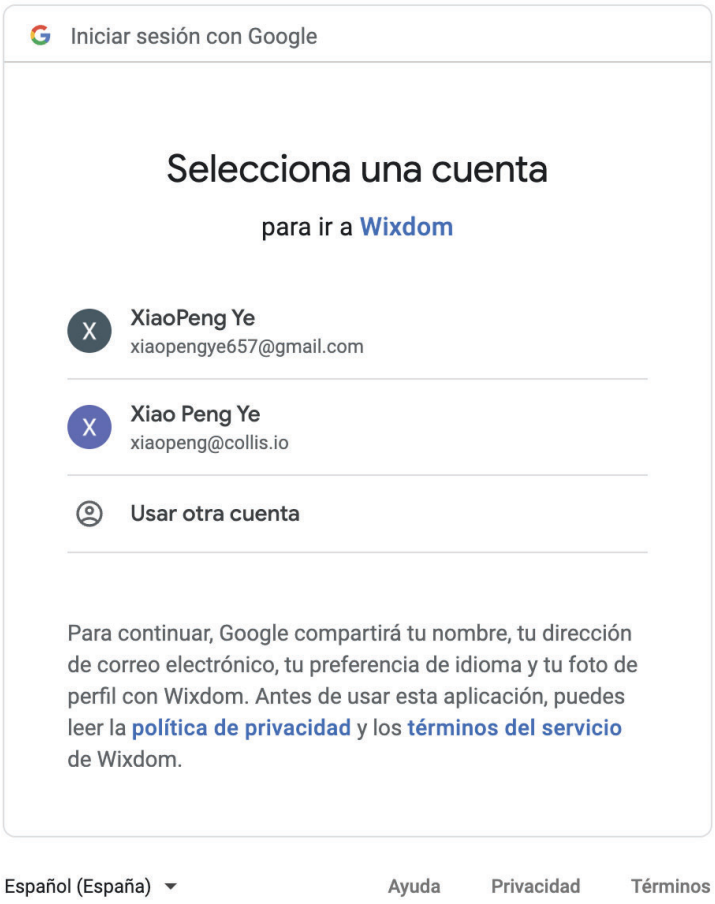

Figura 4.17: Google social login

Una vez iniciado la sesión podemos consultar nuestro perfil, visualizar las sesiones activas y cerrar la sesión como se muestra en la figura del menú 4.18.

Figura 4.18: Menú

En la vista 4.19 tendríamos el perfil de un usuario ya registrado con el correo verificado.

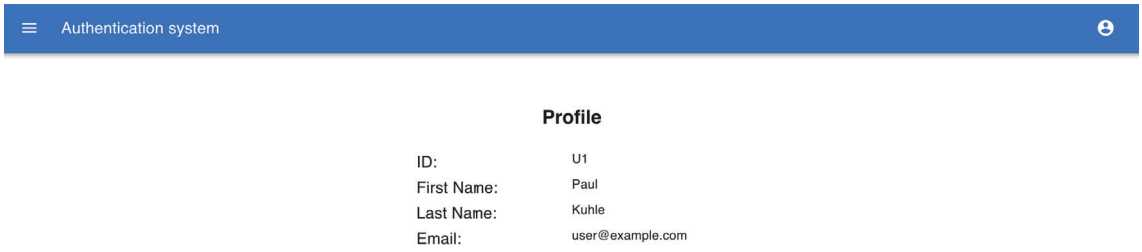

Figura 4.19: Perfil de un usuario verificado

Si el usuario se acaba de registrarse tendríamos una vista 4.20 distinta a la vista anterior al no tener el correo confirmado.

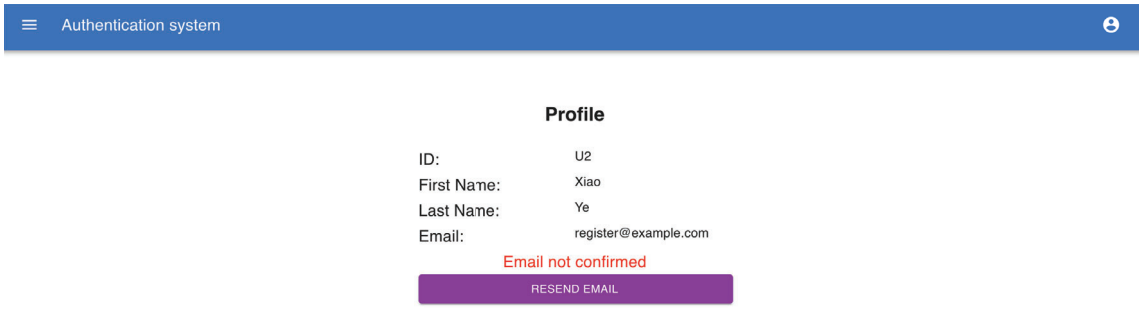

Figura 4.20: Perfil de un usuario que acaba de registrar

Si el inicio de sesión fue logrado con el *social login* y que tratase de una cuenta de Facebook, es necesario pedir el usuario agregar el correo desde el perfil del usuario como se muestra en la figura 4.21.

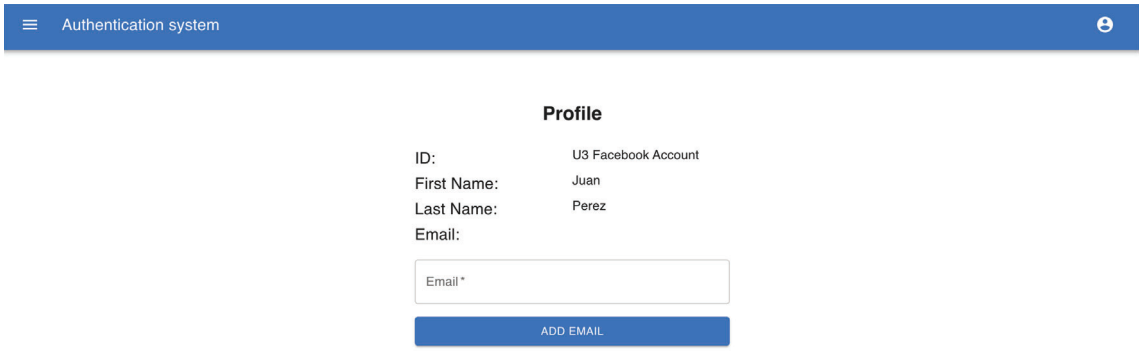

Figura 4.21: Perfil de un usuario con cuenta Facebook sin correo

Desde la vista de login 4.16 podríamos acceder a la vista para recuperar la contraseña 4.22. Nos pedirá la dirección de correo que se quiera recibir el correo de restablecimiento.

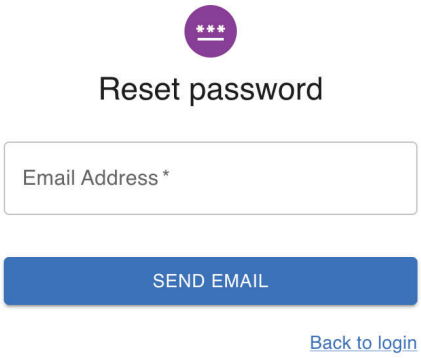

Figura 4.22: Vista para recuperar contraseña

En el correo recibido contendrá un enlace para acceder a la vista para restablecer la contraseña 4.23. Se verificará de la misma forma las contraseñas cuando registramos una cuenta nueva.

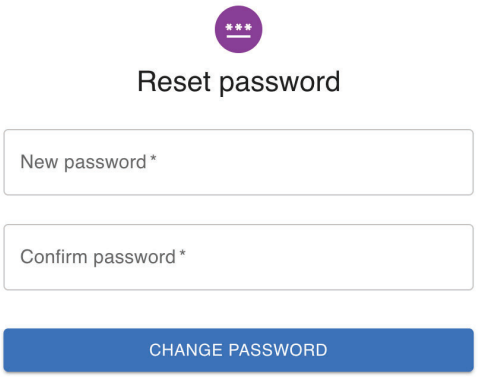

Figura 4.23: Vista para restablecer contraseña

Si nos encontramos con la sesión iniciada y queremos cambiar la contraseña por cuestión de seguridad lo haríamos en la vista 4.24.

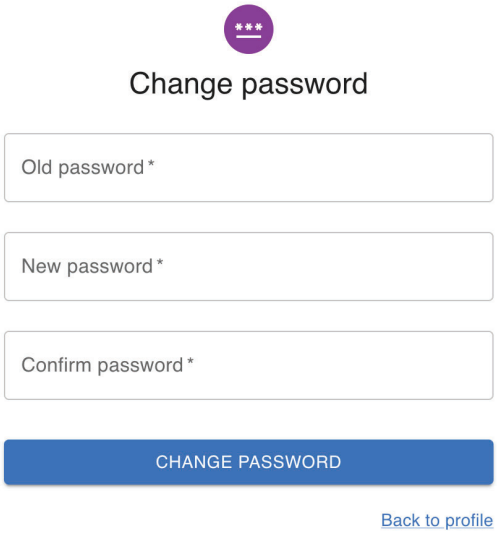

Figura 4.24: Vista para cambiar contraseña

Otra opción que permitía desde el menú de la figura 4.18 es la de visualizar las sesiones como aparece en la vista 4.25 y el poder de invalidarlos mediante el botón que se encuentra debajo de cada sesión.

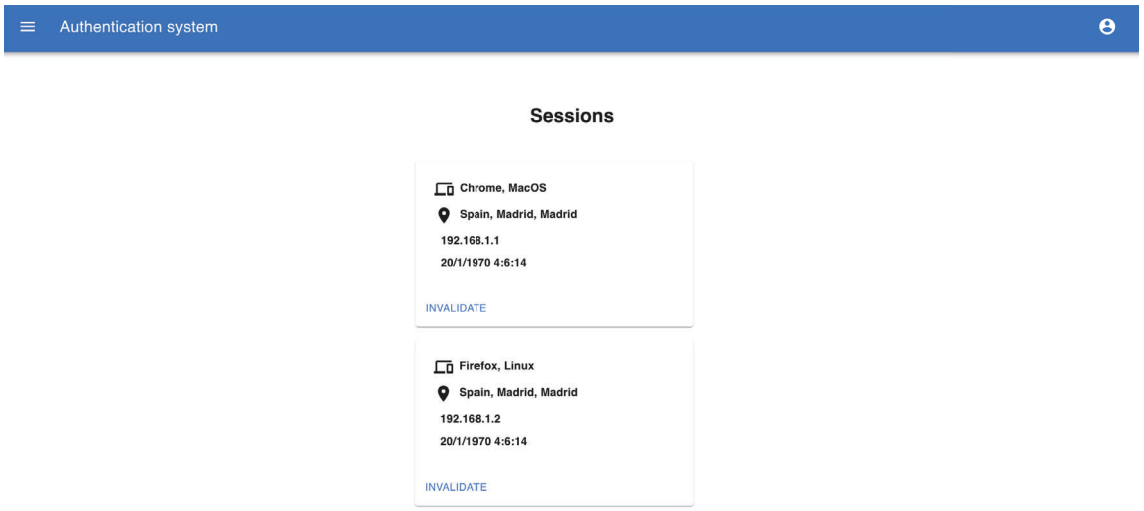

Figura 4.25: Vista de sesiones

Los figuras 4.26 y 4.27 son capturas correspondientes para iniciar sesión y crear una cuenta desde React Native.

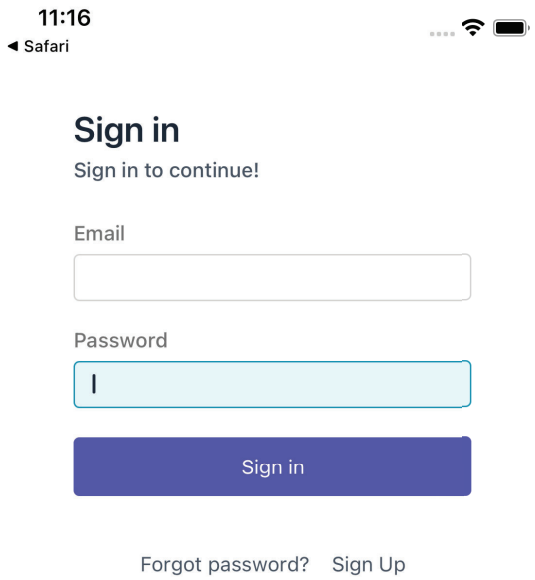

Figura 4.26: Vista para iniciar sesión desde móvil

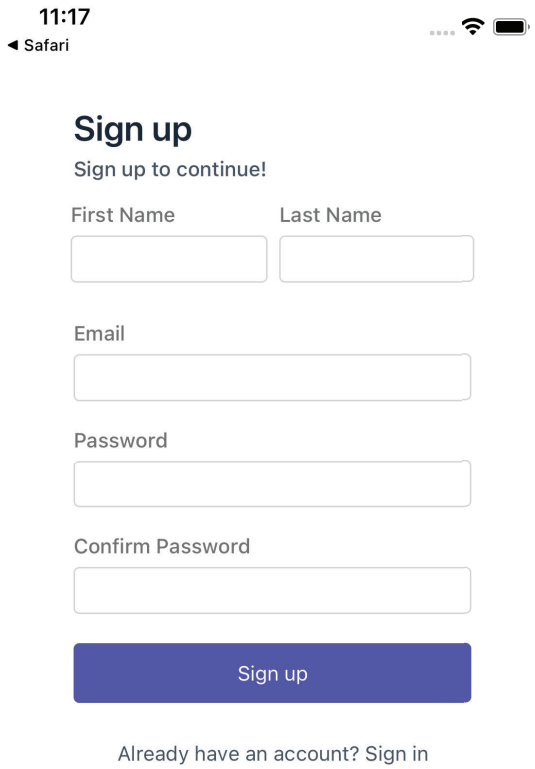

Figura 4.27: Vista para crear cuenta desde móvil

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

La autenticación y autorización es un problema técnico ineludible en el campo informático. Y este proyecto surge justamente por la necesidad de crear una arquitectura de referencia en cuanto a la gestión de autenticación y control de acceso.

En la implementación se ha generado como resultado final vistas en plataformas distintas que sean capaz de testear las lógicas del sistema de autenticación. Para lograr este objetivo se ha iniciado con los estudios de las tecnologías y *framework* usadas en el proyecto, así como TypeScript, GraphQL, Next.js, React Native, Expo, etc. Posteriormente, el estudio en cuanto los estándares de seguridad y las implementaciones o soluciones existentes. Tras los estudios se empezó a definir los requisitos tanto funcionales como no funcionales que tenían que cumplir el sistema. El siguiente paso fue la creación de los diagramas de flujo que se servía para comprender el comportamiento del sistema y las diferencias en cuanto a la gestión del token que podría haber entre Next.js y React Native. Después, se comenzó la definición de los *endpoints* necesarios. A continuación, se crearon los Hooks personalizados según los flujos anteriores y *endpoints* que podían compartir las lógicas entre distintas plataformas junto con las pruebas. Y finalmente, para probarlo en un entorno simulado se desarrollaron las vistas para realizar las pruebas de integración y E2E.

En lo personal, gracias a este proyecto he podido ampliar mis conocimientos sobre el mundo de web moderna, especialmente utilizando herramientas y técnicas que nunca había utilizado, y en mi opinión, sin duda, la mejor de las opciones a día de hoy. También me permitió fortalecer mis conocimientos sobre la seguridad, el cumplimiento de los estándares y el seguimiento de las buenas prácticas existentes.

5.2. Trabajo futuro

En esta sección se detalla aquellos requisitos funcionales y configuraciones que se podría implementar en un futuro.

5.2.1. Desbloquear la App con rastros biométricos

Una vez iniciado la sesión en las plataformas móviles se podrá activar el desbloqueo de la aplicación mediante huella digital y/o Face ID (si se tratase de un iPhone con el hardware disponible) para las próximas sesiones.

5.2.2. MFA (Multi-factor authentication)

El inicio de sesión a través de dispositivo desconocido que se ha utilizado por primera vez por el usuario, se le pedirá autentificarse introduciendo 4 dígitos con la llegada del mensaje por SMS.

Capítulo 6

Análisis de impacto

En este capítulo se analizará el impacto potencial de los resultados obtenidos durante la realización del TFG en línea con los objetivos fijados por el ODS en el marco de la Agenda 2030 y en los diferente contextos para los que se apliquen:

- Personal
- Empresarial
- Social
- Económico
- Medioambiental
- Cultural

Hablando de la agenda 2030, primero debemos saber de qué se trata, así que haré un breve resumen.

En septiembre de 2015, las Naciones Unidas (ONU) adoptaron un conjunto de objetivos para erradicar la pobreza, combatir el cambio climático y proteger el planeta, entre otros. Para lograrlo, los ciudadanos, los gobiernos y las empresas deben poner de su parte en 15 años, de ahí el nombre de Agenda 2030. Estos objetivos se han organizado en 17 objetivos, que a su vez se dividen en diferentes tareas.

De cara a la Agenda 2030, el TFG pondrá especial énfasis en los puntos 4 y 8 [47] sobre educación de calidad, trabajo decente y crecimiento económico. En concreto, se desarrolla una arquitectura de referencia para que todos los que la necesiten puedan acceder a ella como código de referencia, lo que conduce al desarrollo del conocimiento y por tanto al desarrollo económico.

Está comprometida con una educación más completa que subyace en los puntos identificados en la transacción de desarrollo. Por ejemplo, si previamente ha elegido comunidades sostenibles e industrias innovadoras, es fácil integrar un enfoque en el clima, la vida submarina o los ecosistemas terrestres, lo que solo se puede lograr procesando grandes cantidades de información, obtenien-

do documentación y facilitando la difusión de beneficios, los sistemas, valores y conocimientos desarrollados en este TFG.

6.1. Personal

A nivel personal, esto tiene un efecto muy positivo, ya que el tiempo dedicado a aprender las técnicas necesarias que pueden contribuir al funcionamiento del sistema probablemente nunca se dedique a otros contextos. Son tecnologías innovadoras que no se enseñan en las universidades porque son las últimas tecnologías de la industria.

6.2. Empresarial

A nivel empresarial, si nos hacemos cargo de la empresa Collisio Technologies donde hice prácticas, el impacto también es positivo, ya que con este sistema de autenticación se utilizará como arquitectura de referencia en proyectos posteriores, ahorrará tiempo de desarrollo y podrán enfocarse en investigación o desarrollo de otras funcionalidades.

6.3. Social

A nivel social, el proyecto tendrá un impacto positivo en la reputación de los productos de la empresa, ya que implementa un sistema de autenticación que garantiza una funcionalidad muy completa.

6.4. Económico

A nivel económico, este trabajo tiene un impacto positivo, ya que ahorrará el costo adicional de reinventar todo el sistema de autenticación, lo que ahorrará tiempo de desarrollo y, por lo tanto, coste económico.

6.5. Medioambiental

A nivel medioambiental, haciendo referencia a Collisio Technologies con el apartado anterior, junto con el ahorro en tiempo de desarrollo, también se reducirán los costes energéticos y de consumibles.

6.6. Cultural

A nivel cultural, habrá implicaciones al utilizar un sistema de autenticación como ejemplo de desarrollo en el futuro, se utilizarán los mismos estándares de diseño y la necesidad de modificaciones funcionales se limitará a los estándares de seguridad establecidos.

Bibliografía

[1] Wikipedia | authentication. [Online]. Available: https://en.wikipedia.org/wiki/Authentication

[2] Wikipedia | authorization. [Online]. Available: https://en.wikipedia.org/wiki/Authorization

[3] Wikipedia | basic access authentication. [Online]. Available: https://en.wikipedia.org/wiki/Basic_access_authentication

[4] Mdn | http authentication. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

[5] What is the difference between server side cookie and client side cookie? [Online]. Available: https://stackoverflow.com/questions/6922145/what-is-the-difference-between-server-side-cookie-and-client-side-cookie

[6] Mdn | cookies. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

[7] Y. Balaj, "Token-based vs session-based authentication: A survey," 09 2017.

[8] Token based authentication – implementation demonstration. [Online]. Available: https://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/

[9] Json web token (jwt). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519

[10] Json web token structure. [Online]. Available: https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-structure

[11] Wikipedia | single sign-on. [Online]. Available: https://en.wikipedia.org/wiki/Single_sign-on

[12] Wikipedia | security assertion markup language. [Online]. Available: https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language

[13] H. Guevara. How saml authentication works. [Online]. Available: https://auth0.com/blog/how-saml-authentication-works/

[14] Wikipedia | social login. [Online]. Available: https://en.wikipedia.org/wiki/Social_login

[15] The oauth 2.0 authorization framework. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749

[16] Welcome to openid connect. [Online]. Available: https://openid.net/connect/

[17] Saml vs. openid (oidc). [Online]. Available: https://auth0.com/intro-to-iam/saml-vs-openid-connect-oidc/

[18] KirstenS. Cross site request forgery (csrf). [Online]. Available: https://owasp.org/www-community/attacks/csrf

[19] ——. Cross site scripting (xss). [Online]. Available: https://owasp.org/www-community/attacks/xss/

[20] Typescript oficial page. [Online]. Available: https://www.typescriptlang.org/

[21] Chapter 02: First class functions. [Online]. Available: https://mostly-adequate.gitbook.io/mostly-adequate-guide/ch02

[22] Stakcoverflow 2020 developer survey. [Online]. Available: https://insights.stackoverflow.com/survey/2020

[23] Typescript design goals. [Online]. Available: https://github.com/microsoft/TypeScript/wiki/TypeScript-Design-Goals

[24] React | a javascript library for building user interfaces. [Online]. Available: https://reactjs.org/

[25] React | using the state hook. [Online]. Available: https://reactjs.org/docs/hooks-state.html

[26] Rules of hooks. [Online]. Available: https://reactjs.org/docs/hooks-rules.html

[27] Building your own hooks. [Online]. Available: https://reactjs.org/docs/hooks-custom.html

[28] Next.js | the react framework for production. [Online]. Available: https://nextjs.org/

[29] React native. [Online]. Available: https://reactnative.dev/

[30] Expo documentation. [Online]. Available: https://docs.expo.dev/

[31] Graphql | introduction to graphql. [Online]. Available: https://graphql.org/learn/

[32] Graphql: A data query language. [Online]. Available: https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/

[33] Getting started with turborepo. [Online]. Available: https://turborepo.org/docs/getting-started

[34] Mock service worker. [Online]. Available: https://mswjs.io/docs/getting-started/install

[35] Example mock service worker and testing-library. [Online]. Available: https://testing-library.com/docs/react-testing-library/example-intro

[36] Jest | getting started. [Online]. Available: https://jestjs.io/docs/getting-started

[37] Cypress | getting started. [Online]. Available: https://docs.cypress.io/guides/getting-started/installing-cypress

[38] Rfc6749. [Online]. Available: http://www.rfcreader.com/#rfc6749

[39] Mdn | session storage. [Online]. Available: https://developer.mozilla.org/docs/Web/API/Window/sessionStorage

[40] Expo | securestore. [Online]. Available: https://docs.expo.dev/versions/v45.0.0/sdk/securestore/#usage

[41] Plantuml. [Online]. Available: https://plantuml.com/en/

[42] Leaked password api v2. [Online]. Available: https://haveibeenpwned.com/API/v2

[43] Apollo client. [Online]. Available: https://www.apollographql.com/docs/react/

[44] Graphql code generator. [Online]. Available: https://www.graphql-code-generator.com/

[45] Material ui. [Online]. Available: https://mui.com/

[46] Nativebase. [Online]. Available: https://nativebase.io/

[47] Objetivos del desarrollo sostenible. [Online]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

Anexos

Anexo I

Repositorio del proyecto (GitHub):

`https://github.com/xiaopeng-ye/authentication-system-template`

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
Fecha/Hora	Wed Jun 01 23:55:18 CEST 2022
Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
Numero de Serie	561
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)