Trabajo de Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación móvil para la autogestión de la dieta en paciente con medicamentos anticoagulantes e interacciones con otros fármacos.

Autor: Alfonso Jesús Bachot Cornejo

Tutor: José Manuel Fornés Rumbao

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación móvil para la autogestión de la dieta en paciente con medicamentos anticoagulantes e interacciones con otros fármacos.

Autor:
Alfonso Jesús Bachot Cornejo

Tutor:
José Manuel Fornés Rumbao
Profesor titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo de Fin de Grado: Aplicación móvil para la autogestión de la dieta en paciente con medicamentos anticoagulantes e interacciones con otros fármacos.

Autor: Alfonso Jesús Bachot Cornejo

Tutor: José Manuel Fornés Rumbao

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis maestros

A mis amigos

Agradecimientos

Con este proyecto concluye mi etapa universitaria, una época que ha sido larga y muy dura, pero que también ha estado llena de grandes momentos que voy a recordar para siempre.

Todo lo que he conseguido no hubiese sido posible sin las personas que en algún momento me han acompañado y ayudado, por eso, ahora me toca dar las gracias.

En primer lugar, gracias a mis padres, que lo han dado todo para que yo pueda conseguir mi meta, a mi padre por sacrificarse tanto y por educarme siempre con el esfuerzo y a mi madre por cuidarme tanto y estar siempre preocupándose por que todo salga bien.

A mis hermanos, Santi, por contagiarme con su alegría y buen rollo, y a Jose, por ser mi protector y mi mano derecha.

A mi cuñada Aloha por haberme acompañado estos últimos meses tan duros de estudio, y a mis sobrinas Aloha, Candela e Isabel por ser la forma de amor más puro que existe y alegrarme sólo con su presencia.

Al resto de mi familia, tíos y primos, que siempre se han preocupado por mi y me han ayudado cuando me ha hecho falta.

A mis amigos, a los que han estado desde antes y durante todos estos años, a lo que he conocido a lo largo del camino y a los que he recuperado después de mucho tiempo.

A mis compañeros de piso, en especial a Álvaro y Manu que hicieron que los años que compartí con ellos en Sevilla fueran mucho más divertidos.

A todos los maestros que he tenido a lo largo de todos estos años, que me han enseñado tanto y han hecho que ame esta carrera.

Alfonso Jesús Bachot Cornejo

Puente Genil, 2021

Resumen

La llegada de los smartphones junto con Internet ha supuesto una revolución tecnológica, haciendo posible la comunicación de manera casi instantánea entre varias personas sin importar la distancia. Son numerosos los elementos cotidianos que se han aprovechado de este hecho: la forma de relacionarnos, comprar, entretenernos, etc, han cambiado gracias a esto, y la medicina no iba a ser menos.

Es por ello que en los últimos años, y más desde que comenzó el Covid-19, los servicios de salud han tratado de adaptarse con el objetivo de acercarse al paciente de forma telemática, haciendo que las consultas puedan estar más despejadas.

Otro de los elementos que ha hecho posible el surgimiento de grandes cantidades de aplicaciones que nos ayudan con nuestro día a dia es la aparición de marcos de desarrollo multiplataforma, que nos permiten un desarrollo más sencillo y rápido sin tener que preocuparnos de las características de los sistemas finales en los que se ejecuta la aplicación.

En este documento se analiza y utiliza una de estas herramientas, React Native, con el fin de crear una aplicación para la autogestión de los pacientes con enfermedades de coagulación.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Objetivos

En los últimos años se ha producido una revolución tecnológica con la llegada y constante evolución de los smartphones junto con unas conexiones a internet cada vez más accesibles y de mayor calidad. Este hecho ha producido que todo el mundo que nos rodea haya intentado adaptarse a estas nuevas formas de comunicación.

Las tiendas pasan por un modelo de negocio centrado en internet, las formas de entretenimiento cada vez están más ligadas a una pantalla y la forma de relacionarnos entre nosotros pasa a través de nuestros teléfonos móviles.

Además, los efectos de la pandemia en la que nos encontramos en estos momentos sumergidos han favorecido a los desarrollos de formas de interactuar de una manera telemática.

Por estos motivos, la medicina ha tratado de buscar herramientas que se adapten a estos nuevos métodos, como con la telemedicina, que, según explica la Organización Mundial de la Salud, es el uso de las tecnologías de información y comunicación para mejorar los resultados al incrementar el acceso al cuidado y la información médica [1]. Por supuesto, la telemedicina no sustituye a la interacción presencial, si no que se trata de un apoyo [2].

Estos han sido los principales motivos que han dado lugar a este trabajo de fin de grado, que trata sobre el desarrollo de una aplicación móvil para la autogestión de los pacientes con medicamentos anticoagulantes.

En cuanto a las enfermedades relacionadas con la coagulación, el número de pacientes que reciben un tratamiento anticoagulante oral (TAO) ha crecido de forma exponencial en los últimos años [3]. Una de sus indicaciones más frecuentes, la fibrilación auricular, afecta al 4.4% de la población adulta y a más del 17% de los mayores de 80 años. Además, la mayoría de los pacientes presentan riesgos embólicos adicionales (edad avanzada, hipertensión, diabetes, enfermedades cardiovasculares). Estos hechos hacen que los pacienten deban recibir una adecuada información sobre el tratamiento que van a recibir, sus efectos adversos y cómo actuar ante situaciones frecuentes provocadas por estas enfermedades.

Por ello, los médicos que tratan con este tipo de patologías nos han pedido la creación de una herramienta especializada para poder llevar un seguimiento más minucioso de los pacientes, así como herramientas para que estos puedan tener acceso a la información relacionada con la enfermedad.

Las requisitos de estas herramientas deben ser los siguientes:

- Permitir al paciente un seguimiento de su enfermedad, mediante la posibilidad de visualización de los valores de INR que ha registrado en sus visitas al centro de salud. El INR (siglas en inglés de índice internacional normalizado) es un tipo de cálculo que se basa en los resultados de las pruebas de tiempo de protombina. La protombina es una proteína producidad por el hígado, y es uno de los varios factores de la coagulación.
- La posibilidad de que el paciente pueda registrar una reacción adversa provocada por alguno de los medicamentos que está tomando. Las reacciones adversas más frecuentes en estos tipos de enfermedades son los sangrados y los coágulos, de esta forma, cuando se produzca alguno de estos problemas, el paciente podrá notificar desde la aplicación inmediatamente, haciendo que el médico tenga constancia de ello.
- La posibilidad de que el paciente pueda contestar a tests, en especial el test de Morisky-Green, que es un test indicado para los pacientes con enfermedades crónicas, que sirve para valorar el cumplimiento de la medicación [4]. El test está formado por cuatro preguntas que deben ser respondidas con “Sí” o “No”. Estas cuatro preguntas son:

1. ¿Olvida alguna vez tomar los medicamentos para tratar su enfermedad?
2. ¿Toma los medicamentos a las horas indicadas?
3. Cuando se encuentra bien, ¿deja de tomar la medicación?
4. Si alguna vez le sienta mal, ¿deja usted de tomarla?

Se considera que el paciente cumple con la medicación si responde de forma correcta, es decir, No/Sí/No/No. Mediante la posibilidad de realizar este test de forma periódica, el médico podrá conocer si el paciente incumple con la toma de sus medicamentos.

- Proporcionar guías útiles para los pacientes: alimentos contraindicados, interacciones con otros fármacos, etc.
- Poder enviar mensajes en forma de recordatorio o notificaciones a los pacientes de parte de los médicos.

Para llevar a cabo el desarrollo de estas herramientas, se ha dividido en dos partes, un servidor con un cliente web que permita la administración de los pacientes por parte del personal médico y que contenga toda la información de estos, haciéndola accesible a través de la red mediante servicios web, y por otra parte un cliente, una aplicación móvil que pueda ser utilizada por los pacientes para llevar a cabo todas las tareas mencionadas anteriormente.

La primera parte ha sido desarrollada por una compañera, alumna de la escuela también como proyecto de final de grado, y la segunda parte es la que aquí se trata en este documento y que se va a exponer a lo largo de este.

Para llevar a cabo el desarrollo de la aplicación, se ha optado por usar un marco de desarrollo de aplicaciones multiplataforma. Estas herramientas han crecido mucho en popularidad en los últimos años, por lo que se ha decidido por el uso de ellas, no solo por las ventajas que ofrecen al hacer un desarrollo más sencillo y rápido al tener que evitar reescribir todo el código para que funcione en los distintos sistemas operativos a los que la aplicación va dirigida, si no también como forma de aprendizaje y de descubrimiento por mi parte de nuevas herramientas que nos faciliten con el desarrollo de software.

Existen diversas herramientas que nos permiten hacer esto, y la elegida ha sido React Native, creada por Facebook y una de las más populares y con mayor cantidad de usuarios que trabajan con ella, que funciona mediante JavaScript, un lenguaje del que ya disponía conocimientos previos.

1.2 Estructura de la memoria

La estructura del documento que estamos tratando es la siguiente:

1. Introducción: Capítulo en el que nos encontramos, en el que se cuenta cuáles han sido los motivos y las causas que han llevado a este trabajo
2. Estado del arte: En este capítulo se hace un análisis de las herramientas y conceptos que se van a tratar a lo largo de esta memoria.
3. Descripción del sistema: Este capítulo muestra la arquitectura y estructura del sistema desarrollado con la ayuda de distintos diagramas y esquemas.
4. Funcionalidad: En este capítulo se muestra el funcionamiento de la aplicación desarrollada.
5. Pliego de condiciones: Este capítulo recoge los requisitos necesarios para llevar a cabo este trabajo.
6. Conclusiones y Línea de desarrollo: Capítulo final en el que se pone de manifiesto cuáles han sido los resultados del trabajo y se trata qué aspectos pueden ser mejorados.

2 ESTADO DEL ARTE

Antes de comenzar a exponer el trabajo de fin de grado propiamente dicho, y con el fin de aclarar conceptos que serán repetidos de forma reiterada a lo largo de este, se analizará la situación actual, haciendo un estudio de las tecnologías y componentes útiles para el desarrollo de este proyecto, así como también se tendrán en cuenta las distintas alternativas que existen, haciendo una comparativa entre estas y las soluciones que finalmente han sido las seleccionadas para llevar a cabo el trabajo que estamos tratando.

Los conceptos a desarrollar en este apartado serán los siguientes:

- JavaScript
- React Native
- Expo
- Rest
- Firebase

2.1. JavaScript

Figura 1: Logo de JavaScript

JavaScript es un lenguaje de programación interpretado o compilado en tiempo de ejecución, que actualmente se encuentra bajo el desarrollo de la Fundación Mozilla [5] y que según una encuesta realizada en 2020 en StackOverflow [6], una web enfocada a programadores con el fin de poder ayudarles a resolver dudas, es el lenguaje de programación más popular por octavo año consecutivo, siendo este usado por el 67.7% de los encuestados.

Según su documentación oficial, JavaScript se define como un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para la programación orientada a objetos, imperativa y declarativa.

JavaScript es comúnmente conocido como un lenguaje de scripting y desarrollo web aunque también se puede utilizar en entornos fuera de navegador como Node.js para poder construir aplicaciones del lado de servidor y de red.

2.2 React Native

Figura 2: Logo de React Native

React Native es un marco de trabajo creado por Facebook usado para el desarrollo de aplicaciones multiplataforma [7]. React Native se apoya en React, una biblioteca JavaScript para construir interfaces de usuario, para crear aplicaciones nativas Android e iOS [8].

Antes de entrar en detalle sobre el funcionamiento y las ventajas de React Native es necesario comentar brevemente las tecnologías en las que este se basa.

2.2.1 React

Figura 3: Logo de React

React es una biblioteca de código abierto de JavaScript para construir interfaces de usuario de forma declarativa y basada en componentes. React nos permite diseñar vistas simples para los estados de nuestra aplicación, de manera que cuando los datos cambian, React se encarga de actualizar y renderizar de manera eficiente los componentes necesarios.

React viene normalmente acompañado de JSX, que no es más que una extensión de la sintaxis de JavaScript basada en etiquetas, que nos puede recordar a HTML [9]:

```
const element = <h1>Hello, world!</h1>;
```

Figura 4: Elemento de texto usando sintaxis JSX

Este es un ejemplo de JSX para la creación de un componente “element” que consiste en un texto con un formato de encabezado de nivel 1.

Los componentes son una de las partes más importantes de React, en concepto son como las funciones de JavaScript y en la práctica, cuando estamos desarrollando una aplicación usando React, lo que hacemos es crear componentes independientes que manejan su propio estado para dar lugar a interfaces de usuario complejas [10].

Una de las propiedades más notables que incluye React y que se han utilizado en el desarrollo de la aplicación han sido los "Hooks" [11], una característica añadida en la versión 16.8 de React, que nos permite utilizar estados y otras características de React sin escribir clases y que aunque es un componente opcional, han sido utilizados en el desarrollo de la aplicación ya que hacen que el código sea más sencillo y legible.

Dos de los Hooks que nos han resultado más útiles a lo largo de todo el desarrollo de esta aplicación han sido los hooks de estado y los hooks de efecto.

Hooks de estado:

Ejemplo de uso del hook de estado:

```
const [contador, setContador] = useState(0);
```

Figura 5: Hook de estado en React

En este ejemplo hemos declarado una variable de estado "contador" con un valor inicial de 0, y una función de actualización de estado asociada a la variable "contador", llamada "setContador". Cuando en nuestra aplicación queramos mostrar el valor de contador, podremos usar "contador" directamente:

```
<p>Has hecho click {contador} veces</p>
```

Figura 6: Uso del estado "contador"

Y cuando queramos actualizar el valor de contador, tendremos que hacer una llamada a "setContador" con el valor al que queremos que se actualice:

```
<button onClick={() => setContador(contador + 1)}>    Púlsame
  </button>
```

Figura 7: Componente de Botón que al ser pulsado llama a la variable setContador

De esta forma, cuando se actualiza el valor de la variable "contador", React se encarga de actualizar el texto con el nuevo valor de "contador"

Hooks de efecto:

El hook de efecto, equivale a los ciclos de vida de las clases de React y sus métodos, componentDidMount, componentDidUpdate y componentWillUnmount, todos ellos combinados en useEffect [12].

El hook useEffect por defecto se ejecuta después del primer renderizado y después de cada actualización, y es usado cuando queremos que un componente haga algo después de renderizarse.

React también puede ser usado para renderizar desde el servidor usando Node, o como en el caso que aquí se trata, para impulsar aplicaciones móviles gracias a React Native.

2.2.2 Características

Como ya se ha comentado al principio de este apartado, React Native es una librería de código abierto que usa React para crear aplicaciones nativas Android e iOS a partir de un único código JavaScript.

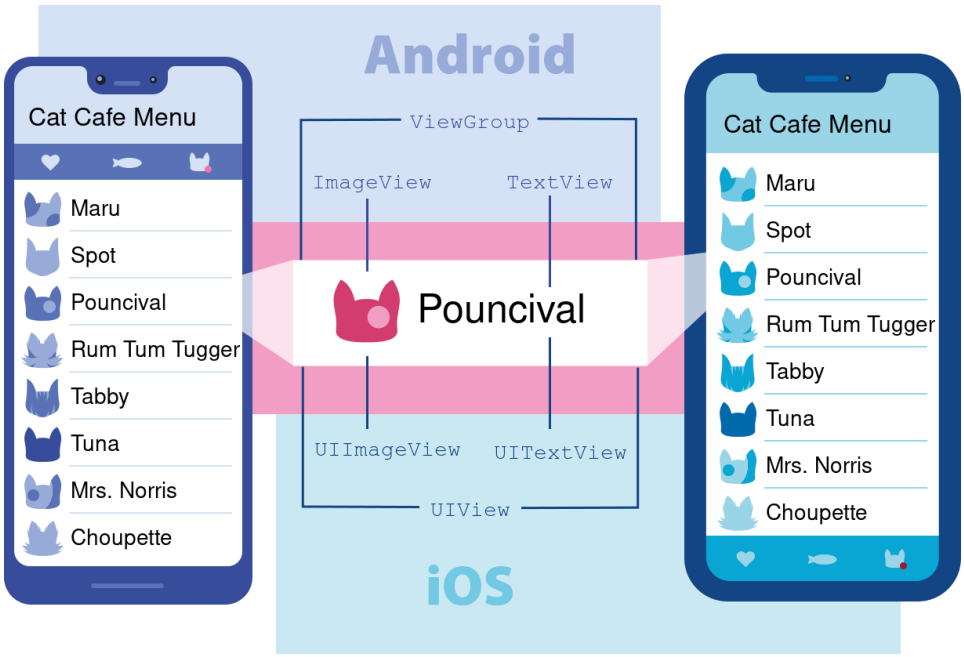


Figura 8: Esquema de funcionamiento de React Native.

2.2.3 Componentes Nativos

React Native se encarga de crear los correspondientes componentes nativos como si de estar programando en Swift o Objective-C en el caso de iOS o Java en el caso de Android se tratase.

Estos componentes se muestran y se ejecutan de forma prácticamente idéntica a los de cualquier aplicación desarrollada con las herramientas mencionadas anteriormente.

React Native viene de forma predefinida con una colección de componentes listos para ser usados llamados Core Components. Se muestra a continuación una tabla con los componentes más típicos y sus análogos en las diferentes plataformas que ejecuta React Native [13]:

React Native	Android	iOS	Web	Descripción
`<View>`	`<ViewGroup>`	`<UIView>`	`<div>`	Contenedor genérico sin posibilidad de hacer scroll
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Muestra cadena de texto
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Muestra imagen
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	Contenedor genérico con scroll
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Permite al usuario introducir texto

Tabla 1: Principales Core Components de React Native

Estos no son los únicos componentes que trae React Native, en su documentación podemos encontrar muchos más, incluyendo algunos componentes específicos para una plataforma determinada, y, además, React Native nos permite crear nuestros componentes personalizados así como utilizar colecciones de componentes ya creados por otros usuarios, como React Native Paper**,** unos componentes que siguen las líneas de estilo de Google Material Design [14], y que en mi caso, he utilizado en esta aplicación.

De forma idéntica a React, en React Native se usa JSX como sintaxis para escribir elementos dentro de JavaScript:

```
<Text>Hello World, {nombre}</Text>
```

Figura 9: Componente de texto de React Native.

En este ejemplo de combinan los componentes de React Native, en este caso de texto, con la sintaxis de JSX.

2.2.4 Props

Término para referirse a las propiedades, y que nos permite personalizar los componentes, ya sean los creados por el propio usuario, o los Core Components ya mencionados anteriormente. Por ejemplo, cuando utilizamos el componente <Image> para insertar una imagen, si le pasamos la propiedad "source", le estamos indicando a ese componente de donde tiene que leer la imagen que va a mostrar.

```
<Image
        source={{uri: "https://reactnative.dev/docs/assets/p_cat1.png"}}
      />
```

Figura 10: Componente de Imagen con el uso del prop "source"

Los componentes tienen varios props, incluyendo "style" que se es utilizado muy a menudo en el desarrollo de esta aplicación para el diseño del layout de los componentes de la interfaz en las distintas pantallas de la aplicación.

2.2.5 Estados

De la misma forma que lo explicado en relación a los estados de React, estos son utilizados de forma muy común en el desarrollo de aplicaciones con React Native ya que a menudo tendremos elementos que cambien al interactuar con nuestra aplicación, como al pulsar un botón, de forma que cuando este es pulsado, por ejemplo, podremos hacer una llamada a un elemento de red para obtener información de un servidor remoto y que esta información sea mostrada en pantalla.

2.2.6 Alternativas

Los frameworks de desarrollo multiplataforma han crecido en popularidad enormemente en los últimos años debido a la gran utilidad de estos al permitirnos acortar de forma considerable el tiempo de desarrollo de una aplicación, y no son pocas las corporaciones que han lanzado su propio framework, como Facebook en el caso de React Native, Google con Flutter y Apache con Cordova.

Cada uno de estos frameworks, aunque en esencia nos permiten hacer lo mismo, tienen una serie de diferencias que les proporcionan distintas ventajas y desventajas frente al otro. En este apartado me centraré en React Native y Flutter, los dos más populares en la actualidad.

Flutter es un kit de desarrollo de interfaces de usuario (UI) creado por Google y que utiliza el lenguaje de programación Dart [15], y React Native es un framework para el desarrollo de aplicaciones móviles creado por Facebook que utiliza JavaScript como lenguaje de programación y permite utilizar las características de React de forma nativa en dispositivos móviles.

Ambos son de código abierto y disponen de gran cantidad de documentación en la red así como una gran cantidad de usuarios que trabajan con ellos.

En este caso, se ha decidido por utilizar React Native ya que se tienen conocimientos previos de JavaScript y no de Dart, y además, React Native es un framework más maduro, ya que fue lanzado en 2015 frente a Flutter, el cual lleva activo desde 2018, y además React Native posee una gran cantidad de tutoriales, librerías creadas por usuarios y otras utilidades que nos pueden ayudar en nuestro desarrollo, como es el caso de Expo, una colección de herramientas enfocadas a facilitar el desarrollo de aplicaciones creadas mediante React Native.

Figura 11: Imagen de Google Trends que muestra las tendencias de búsqueda de Flutter y React Native en los últimos 5 años [16]

Aún así, es destacable mencionar que las tendencias muestran que Flutter está superando a React Native en términos de búsqueda y popularidad.

2.3 Expo

Figura 12: Logo de Expo

Como ya se ha mencionado, Expo es una colección de herramientas que nos facilitan el desarrollo de aplicaciones creadas usando React Native [17]. Expo nos ayuda con el desarrollo, el despliegue, la construcción y el testeo rápido de nuestra aplicación, así como también nos proporciona acceso a algunas características de nuestros dispositivos como la cámara, el micrófono y el servicio de notificaciones.

Una de las herramientas que acompaña a Expo y que hace que sea de una gran utilidad es Expo Go, una aplicación que se encuentra disponible para dispositivos Android e iOS en sus respectivas tiendas de aplicaciones, y que nos permite probar en tiempo real en nuestro dispositivo la aplicación que estamos desarrollando con tan solo escanear un código QR [18].

Son numerosas las ventajas de usar Expo frente al cliente tradicional de React Native, en primer lugar, ya que si usamos Expo, solo necesitamos realizar la instalación de Expo junto a Node.js para comenzar a desarrollar y probar nuestra aplicación, mientras que si prescindimos de este, necesitaremos instalar Android Studio, configurarlo correctamente para que trabaje con las aplicaciones de React Native y no nos sería posible probar nuestra aplicación en nuestro dispositivo físico.

2.4 REST

Del inglés Representational State Transfer, o transferencia de estado representacional, es un estilo de arquitectura [19] para la comuncicación entre sistemas. En la actualidad se denomina REST a cualquier interfaz que utilice HTTP para la obtención y el envío de datos en cualquier tipo de formato, por ello, los métodos utilizados en una API REST normalmente serán POST (envío de datos en el cuerpo del mensaje), GET (obtención de datos), PUT (orientado a la actualización de datos) y DELETE (borra un recurso especificado)

La elección de REST frente a otro protocolo como SOAP es debida principalmente a la posibilidad que nos proporciona REST de enviar JSON, y ya que estamos trabajando con un framework cuyo lenguaje es JavaScript, es lo ideal gracias a que los datos JSON son interpretados de forma natural en JavaScript siendo interpretados de forma mucho más ligera.

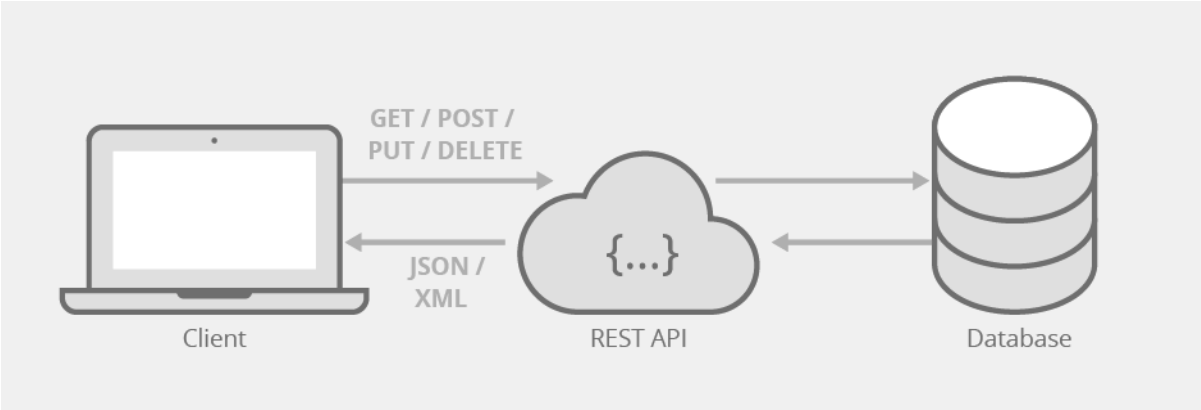


Figura 13: Diagrama que muestra el funcionamiento de una API REST

Una descripción más detallada de los servicios REST utilizados en nuestra aplicación vendrá dada en el siguiente capítulo.

2.5 Firebase

Figura 14: Logo de Firebase

Firebase es una plataforma de desarrollo de aplicaciones móviles creada por Google [20] que consiste en una colección de herramientas que nos proporciona una gran cantidad de servicios que nos pueden ayudar con varios aspectos esenciales en una aplicación móvil, como son la autenticación, el alojamiento, la mensajería, así como otros muchos más.

Firebase nos proporciona una interfaz web desde la que podemos administrar todos los servicios que nos ofrece, así como ver estadísticas de uso de nuestra aplicación.

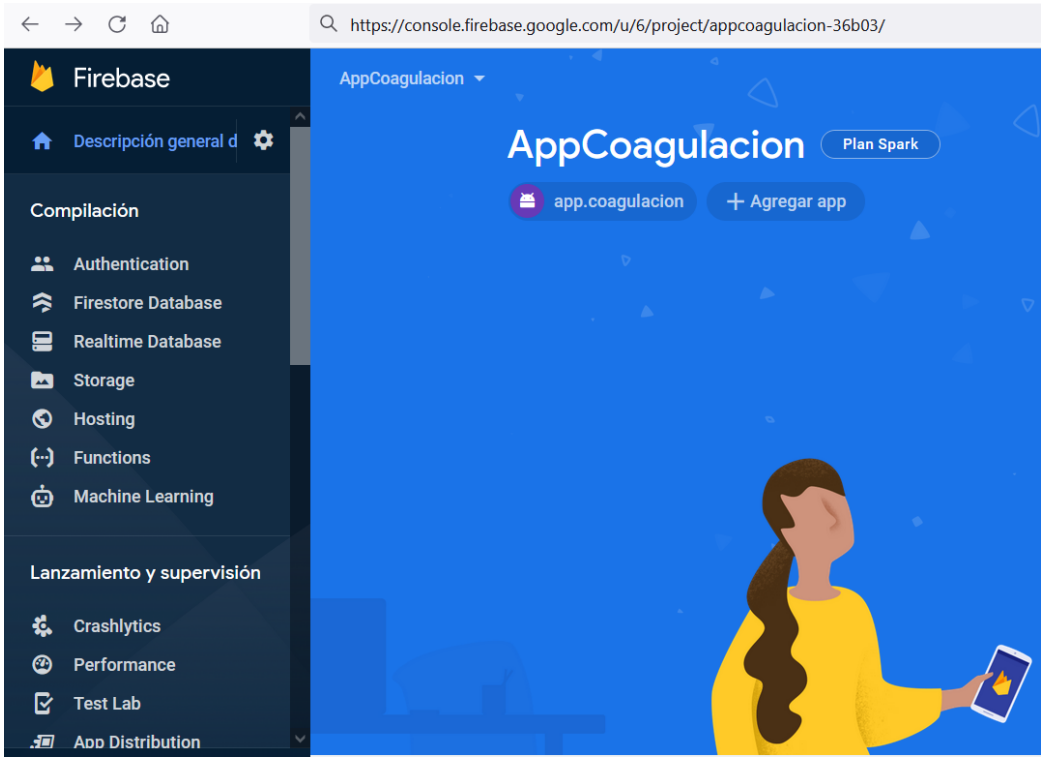


Figura 15: Consola de Firebase

En este trabajo se utiliza Firebase para la gestión de notificaciones a los usuarios a través de su servicio de mensajería en la nube Firebase Cloud Messaging (FCM) [21] a través de la API que proporciona, y que está integrada en el servidor, es decir, no se hace uso de la interfaz que proporciona Firebase para el envío de notificaciones.

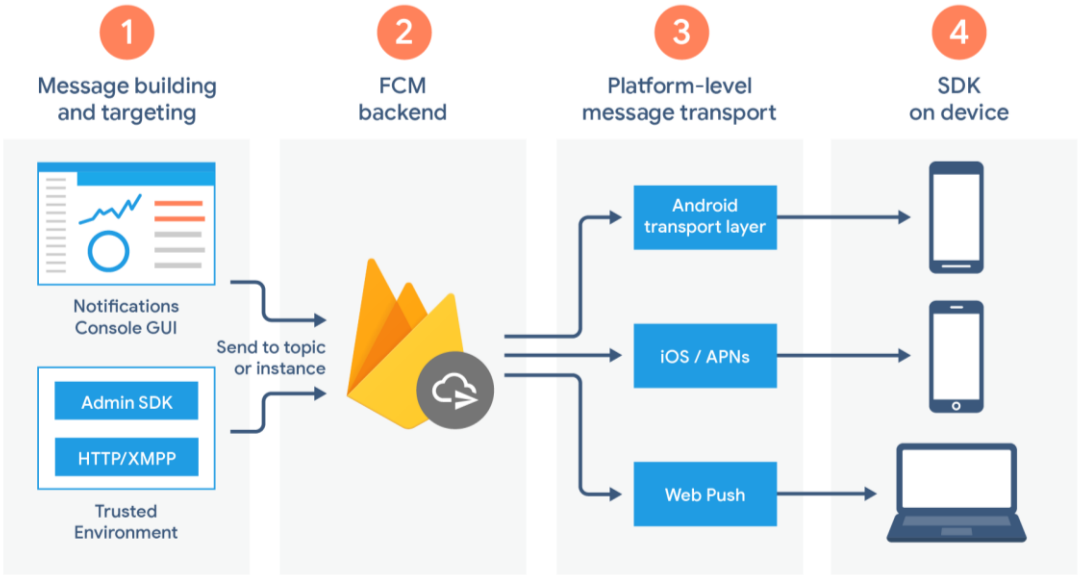


Figura 16: Diagrama de funcionamiento de FCM

La API que se ha utilizado ha sido FCM HTTP v1 [22], que es la más actualizada y flexible, y que en este caso es la única que nos permite el envío de notificaciones con imágenes.

3 DESCRIPCIÓN DEL SISTEMA

En este capítulo se va a mostrar la arquitectura y estructura del sistema a través de distintos diagramas con el fin de poder obtener una visión detallada del diseño e implementación de la aplicación.

Comenzando con un esquema general del escenario de funcionamiento en el que se pueden ver los distintos agentes implicados y sus relaciones entre ellos:

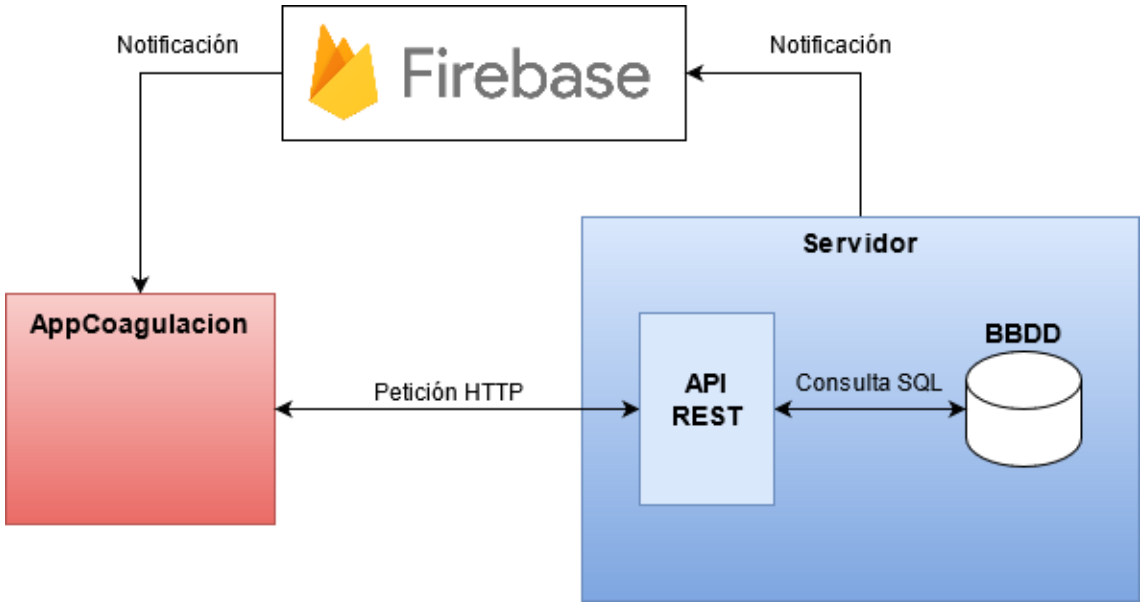


Figura 17: Esquema Funcionamiento Cliente-Servidor

Así, se pueden distinguir tres partes diferenciadas:

- AppCoagulación: Se trata del cliente, es la aplicación desarrollada en este trabajo de fin de grado. Se comunica con el Servidor a través de peticiones HTTP mediante la API REST que este tiene integrada.
- Servidor: Servidor web con el que la aplicación ha de contactar con el fin de intercambiar la información necesaria para el funcionamiento de esta. Incorpora una Base de Datos al que hace las consultas necesarias para suministrar o añadir la información que le requiera el cliente.
- Firebase: Se trata de un intermediario que nos permite que el servidor envíe notificaciones al dispositivo móvil del usuario de la aplicación a través de su plataforma Firebase Cloud Messaging.

3.1 Cliente

A continuación se muestra un esquema en el que se pueden ver los componentes de la aplicación cliente y la relación que existe entre las distintas partes:

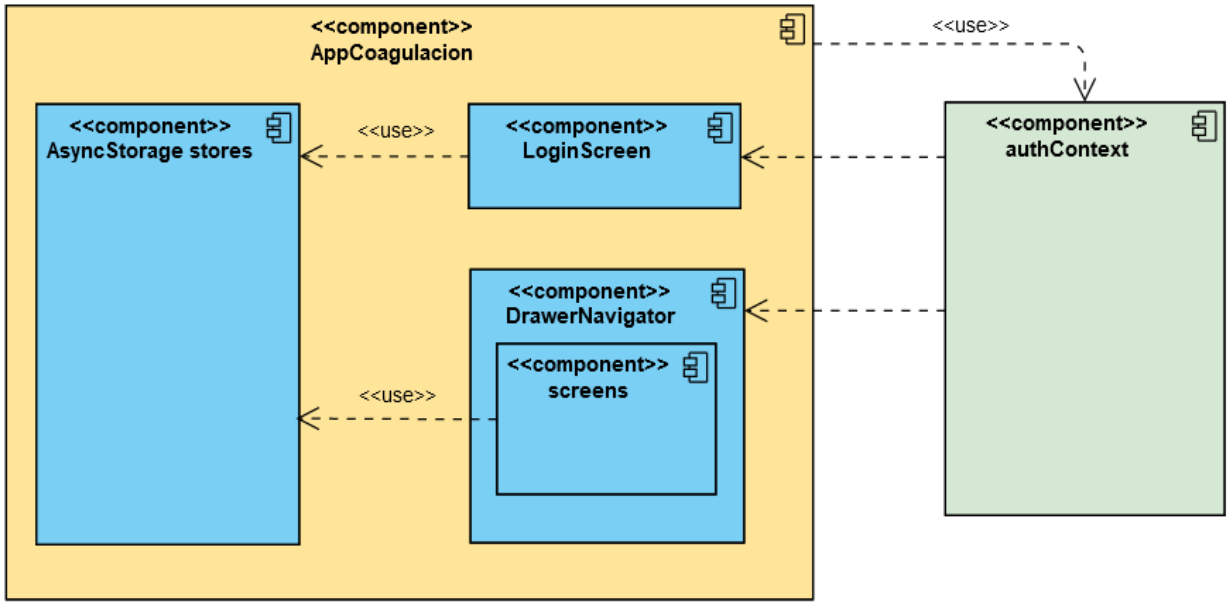


Figura 18: Diagrama de Componentes del Cliente

3.1.1 Pantallas

La aplicación consta de distintas pantallas entre las que se puede navegar a través de un menú lateral (Drawer Lateral). Al abrir por primera vez la aplicación, se muestra la pantalla de Login para poder iniciar sesión, una vez se han introducido el usuario y la contraseña correctos, se accede a la aplicación, abriéndose por defecto la pantalla de Inicio. En todas las pantallas excepto en la de Login se encuentra disponible el Menú Lateral que contiene acceso al resto de pantallas de la aplicación, así como un botón de Cerrar Sesión que nos devuelve a la pantalla de Login.

A continuación se muestra un diagrama de flujo de la navegación entre pantalla de la aplicación:

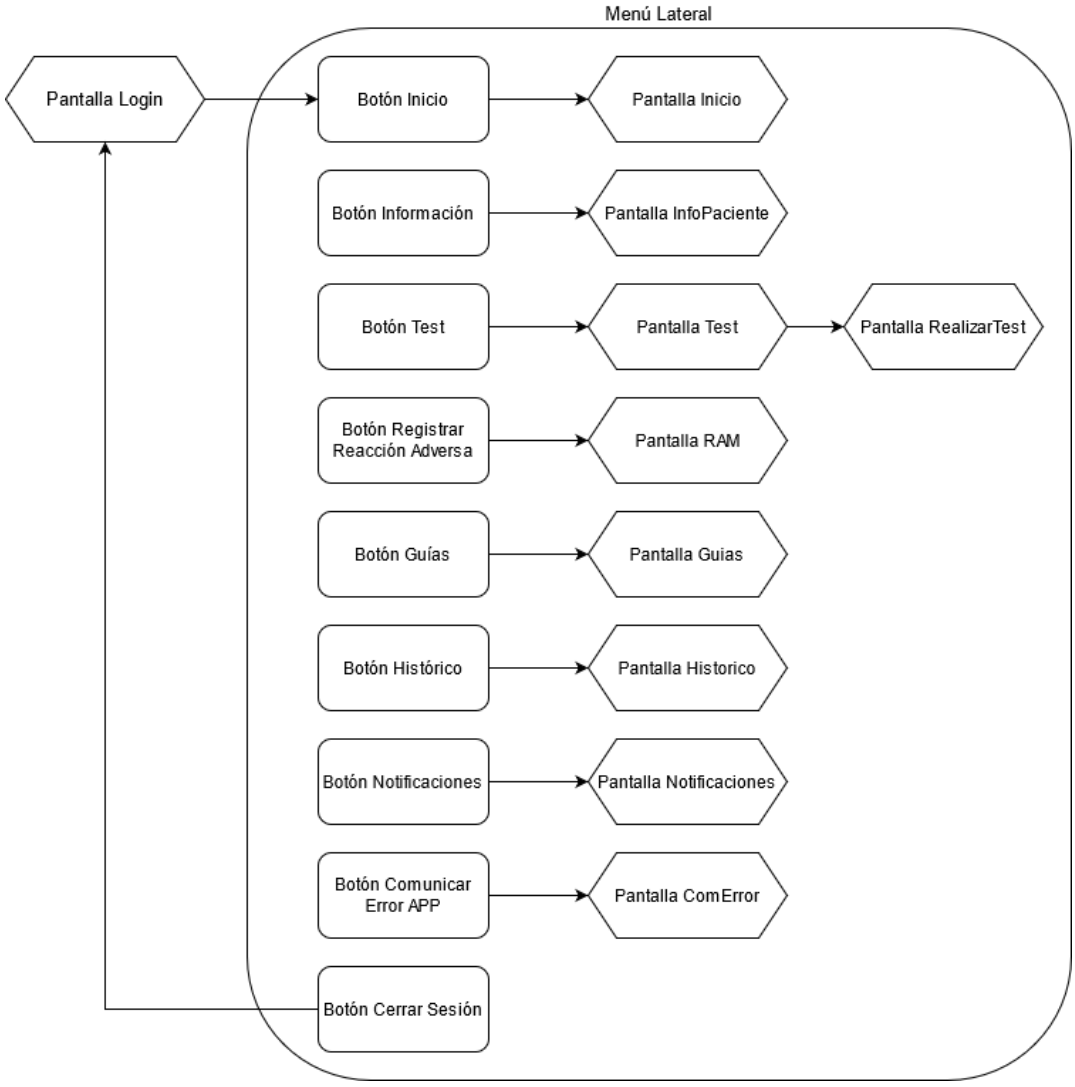


Figura 19: Diagrama de Flujo Navegación APP

En total existen diez pantallas distintas:

- LoginScreen.js: Pantalla de inicio de sesión a través de un usuario y contraseña
- InicioScreen.js: Pantalla de Bienvenida de la aplicación.
- InfoPacienteScreen.js: Pantalla donde se muestran los datos del paciente, así como las reacciones adversas al medicamento que este haya podido registrar a través de la pantalla RAMScreen.js
- TestScreen.js: En esta pantalla se mostrarán los tests disponibles que el usuario tiene para ser realizados. Cuando un médico asocia un nuevo test a un paciente, aparecerá en esta pantalla, estos tests podrán ser realizados como máximo una vez a la semana, por lo que para saber si un paciente tiene un test disponible para ser realizado, se tiene que consultar cúando fue la última vez que se respondió dicho test, y si ha pasado más de una semana se considera que el test debe estar disponible.
- RealizarTestScreen.js: A esta pantalla se accede desde la pantalla Test.js, al haber pulsado en el botón que aparece cuando un test está disponible, y mostrará las dintintas preguntas del test, que deben ser respondidas con “Sí” o “No”.
- RAMScreen.js: En esta pantalla se mostrarán dos listas seleccionables que contendrán los tratamientos que tiene asignados el paciente y las posibles reacciones adversas que pueden causar los medicamentos anticoagulantes, permitiendo al paciente registrar una reacción adversa que ha sido provocada por un tratamiento específico.
- GuiasScreen.js: Pantalla que muestra una serie de guías útiles para el paciente.
- HistoricoScreen.js: En esta pantalla el paciente puede seleccionar un mes y año para que sea mostrada una gráfica con los datos de INR del mes seleccionado.
- NotificacionesScreen.js: Pantalla que muestra las notificaciones recibidas por el paciente.
- ComErrorScreen.js: Pantalla que permite al paciente enviar un mensaje con el posible error que haya encontrado al navegar por la aplicación.

3.1.2 Navigators

Los navigators o navegadores son un componente de React Native que nos permite la navegación entre las distintas pantallas que forman nuestra aplicación [23]. Existen un total de nueve Navigators:

- DrawerNavigator: Este navegador es proporcionado por React Native Paper, consiste en un menú lateral desplazable, que incorpora botones que nos permiten navegar a cada una de las pantallas de la aplicación. Es el navegador principal de la aplicación, ya que todos los demás navegadores están anidados en este. Por defecto el Drawer redirige a la pantalla Inicio.
- InicioNavigator: Navegador que gestiona la pantalla de Inicio.
- InfoPacienteNavigator: Navegador que gestiona la pantalla de InfoPaciente.
- TestNavigator: Navegador que gestiona la pantalla de Test y RealizarTest.
- RAMNavigator: Navegador que gestiona la pantalla de Reacciones Aversas al Medicamento.
- GuiasNavigator: Navegador que gestiona la pantalla de Guías.
- HistoricoNavigator: Navegador que gestiona la pantalla de Históricos.
- NotificacionesNavigator: Navegador que gestiona la pantalla de Notificaciones.
- ComunicarErrorNavigator: Navegador que gestiona la pantalla de Comunicar Error de la APP.

```
1  import React from "react";
2  import { TouchableOpacity } from 'react-native';
3  import Icon from 'react-native-vector-icons/MaterialCommunityIcons';
4  import { createStackNavigator } from "@react-navigation/stack";
5  import GuiasScreen from "../screens/GuiasScreen";
6
7  const Stack = createStackNavigator();  // creates object for Stack Navigator
8
9  export default function GuiasNavigator({ navigation }) {
10   return (
11     <Stack.Navigator >
12       <Stack.Screen
13         name="Guias"
14         component={GuiasScreen}
15         options={{
16           headerTitle: 'Guias',
17           headerStyle: {
18             backgroundColor: 'crimson',
19           },
20           headerTitleStyle: {
21             fontSize: 30,
22             fontWeight: 'bold',
23             color: 'white',
24             marginLeft: 20,
25           },
26           headerLeft: () => (
27             <TouchableOpacity style={{ marginLeft: 20 }}
28               onPress={() => navigation.toggleDrawer()}>
29               <Icon style={{ marginLeft: 5 }}
30                 name="forwardburger"
31                 color="white"
32                 size={35}
33               />
34             </TouchableOpacity>
35           ),
36         }}
37       />
38     </Stack.Navigator>
39   );
40 }
41
```

Figura 20: GuiasNavigator.js

3.1.3 Contexto

authContext: Consiste en el contexto de autenticación de la aplicación. En React, los contextos son una forma de pasar datos a través del árbol de componentes [24] y son usados cuando se quieren compartir datos que pueden ser considerados globales.

En el caso que aquí se trata, el contexto authContext está formado por dos funciones distintas, signUp para iniciar sesión y signOut para cerrar sesión.

Estas funciones actualizan los estados pertenecientes a un Reducer [25], los cuales contienen los datos de autenticación de la aplicación, es decir, un estado "token", que consiste en un token OAuth que es obtenido al iniciar sesión, un estado "isLoading" que es "true" cuando se espera la respuesta del servidor al iniciar sesión y es "false" cuando ya se ha recibido dicha respuesta, y un estado "isSignout" que utilizamos para tratar la lógica de autenticación y poder saber cúando un token es válido o no.

```
72  const [state, dispatch] = React.useReducer(
73      (prevState, action) => {
74        switch (action.type) {
75          case 'RESTORE_TOKEN':
76            return {
77              ...prevState,
78              userToken: action.token,
79              isLoading: false,
80            };
81          case 'SIGN_IN':
82            return {
83              ...prevState,
84              isSignout: false,
85              userToken: action.token,
86            };
87          case 'SIGN_OUT':
88            return {
89              ...prevState,
90              isSignout: true,
91              userToken: null,
92            };
93        }
94      },
95      {
96        isLoading: true,
97        isSignout: false,
98        userToken: null,
99      }
100   );
```

Figura 21: Reducer que contiene los estados necesarios para el flujo de autenticación

En el siguiente diagrama de flujo se puede ver cómo funciona el ciclo de autenticación, donde se puede ver cómo de forma constante se comprueba el valor del token tras ejecutar el dispatch(RESTORE_TOKEN), y dependiendo de si existe o no, nos es mostrado el Menú o la pantalla de Login:

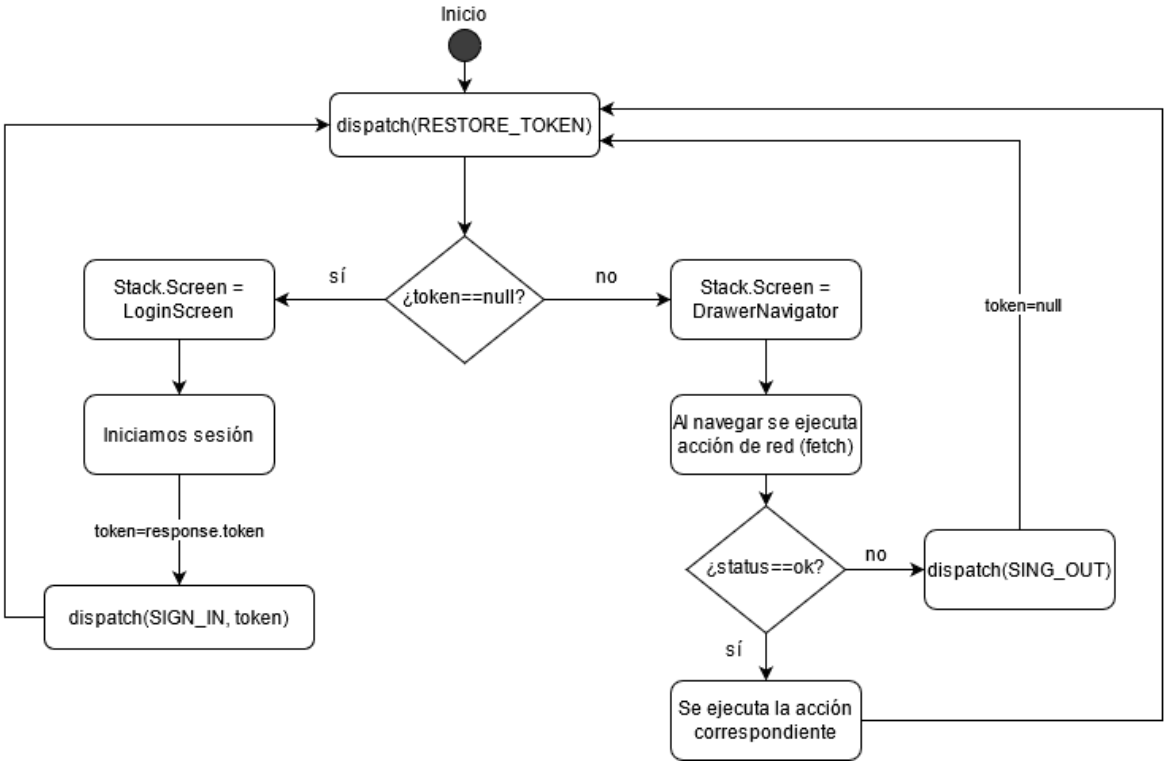


Figura 22: Diagrama de flujo estados Reducer.

En el siguiente fragmento de código se muestra cómo se ha implementado la lógica del flujo de autenticación, donde puede verse que toda la navegación de la aplicación está dentro del contexto "authContext", y en función de los estados del Reducer mostrado anteriormente, se carga una pantalla u otra:

```
276  return (
277      <PaperProvider>
278        <AuthContext.Provider value={authContext}>
279          <NavigationContainer>
280            <Stack.Navigator headerMode="none">
281              {state.isLoading ? (
282                <Stack.Screen name="Splash" component={SplashScreen} />
283              ) : state.userToken == null ? (
284                <Stack.Screen name="Login" component={LoginScreen} />
285              ) : (
286                <Stack.Screen name="Drawer" component={DrawerNavigator} />
287              )}
288            </Stack.Navigator>
289          </NavigationContainer>
290        </AuthContext.Provider>
291      </PaperProvider>
292    );
```

Figura 23: Flujo de autenticación

3.1.4 Stores

Las Stores son utilizadas para guardar datos en la aplicación. Se utiliza AsyncStorage, un sistema de guardado de datos asíncrono, sin cifrado, persistente y que funciona mediante pares clave-valor [26].

En la aplicación han sido utilizadas cuatro stores distintas:

tokenStore: guarda el valor del token OAuth obtenido al iniciar sesión.

userIdStore: guarda el número de usuario, necesario para su identificación y para las peticiones de datos al servidor.

infoPacienteStore: contiene la información relativa a los datos del paciente.

notificacionesStore: guarda las notificaciones recibidas por el usuario.

```
1  import AsyncStorage from '@react-native-async-storage/async-storage';
2
3  const storeUserID = async (value) => {
4      try {
5        await AsyncStorage.setItem('@userid', value)
6      } catch (e) {
7        // error al guardar
8      }
9    }
10
11   const getUserID = async () => {
12     try {
13       const value = await AsyncStorage.getItem('@userid')
14       if(value !== null) {
15         // devuelve el valor pedido
16         return value;
17       }
18     } catch(e) {
19       // error al leer el valor
20     }
21   }
22
23   export {storeUserID, getUserID}
```

Figura 24: userIdStore.js

3.2 Servidor

Para el funcionamiento de la aplicación cliente han sido definidos nueve servicios REST que se encuentran en el Servidor Web, estos servicios son:

1. Servicio Login
2. Servicio Histórico
3. Servicio Responder Test

4. Servicio Comunicar Reacción Adversa
5. Servicio Notificacion
6. Servicio Error
7. Servicio Información Paciente
8. Servicio Información Autenticación
9. Servicio Solicitar Reacciones Adversas

Nombre del Servicio	Login
Descripción	Servicio usado para realizar la autenticación entre el cliente y el servidor.
Método	POST
Autorización	No
Parámetros de entrada	• username • password • tokenDevice • modelo
Respuesta	• status • payload ○ accessToken
Formato	JSON
URL	http://www.appsmedicas.com/appcoag/login

Ejemplo Solicitud	{
	"username": "paciente",
	"password": "password",
	"tokenDevice": "token",
	"modelo": "modelo"
	}

Ejemplo Respuesta	{
	"status": "ok",
	"payload": {
	"accessToken": "eyJhbGciOiJIUzI1NiIsI"
	}
	}

Tabla 2 – Servicio Login

Nombre del Servicio	Histórico
Descripción	Usado para consultar los datos del histórico de INR de un paciente. El usuario selecciona un mes y el año para el que quiere que se muestre la gráfica.
Método	POST
Autorización	Java Web Token
Parámetros de entrada	• pacientefk • mes • anio

Respuesta	• status
	• payload
	  ○ inrs
	  ▪ idinr
	  ▪ pacientefk
	  ▪ valor
	  ▪ fecha

Formato	JSON
URL	http://www.appsmedicas.com/appcoag/historico

Ejemplo Solicitud	{
	"pacientefk":2,
	"mes": 1,
	"anio": 2021
	}

Ejemplo Respuesta	{
	"status": "ok",
	"payload": {
	"inrs":
	[{
	"idinr": 2,
	"pacientefk": 2,
	"valor": 7.3,
	"fecha": "2021-01-04T23:00:00.000Z"
	},]
	}
	}

Tabla 3 – Servicio Histórico

Nombre del Servicio	Responder Test
Descripción	El usuario envía las respuestas del test y la fecha de realización. Las respuestas de este test solo pueden ser “SI” o “NO”
Método	POST
Autorización	Java Web Token

Parámetros de entrada	• pacientefk
	• testfk
	• respuestas
	  ○ id
	  ○ respuesta
	• fecha

Respuesta	• payload • status
Formato	JSON
URL	http://www.appsmedicas.com/appcoag/send-test

Ejemplo Solicitud	{
	"pacientefk":2,
	"testfk": 1,
	"respuestas":"[{\"id\":0,\"respuesta\":\"NO\"},
	{\"id\":1,\"respuesta\":\"NO\"},
	{\"id\":2,\"respuesta\":\"NO\"}]",
	"fecha": "2021-06-02 21:02:21"
	}

Ejemplo Respuesta	{
	"status": "ok",
	"payload": “Test respondido correctamente”
	}

Tabla 4 – Servicio Responder Test

Nombre del Servicio	Comunicar Reacción Adversa
Descripción	El usuario envía una Reacción Adversa al Medicamento con la fecha de cuándo sucedió.
Método	POST
Autorización	Java Web Token
Parámetros de entrada	• tratamientofk • tipo_ramfk • fecha
Respuesta	• payload • status
Formato	JSON
URL	http://www.appsmedicas.com/appcoag/ram

Ejemplo Solicitud	{
	"tratamientofk": 10,
	"tipo_ramfk":3,
	"fecha":"2021-06-01 10:00:05"
	}

Ejemplo Respuesta	{
	"status": "ok",
	"payload": 27
	}

Tabla 5 – Servicio Reacción Adversa

Nombre del Servicio	Notificación
Descripción	Mediante este servicio se obtienen las notificaciones enviadas al usuario.
Método	GET
Autorización	Java Web Token
Parámetros de entrada	• pacientefk

Respuesta	• status
	• payload
	  ○ notificaciones
	  ▪ idnotificacion
	  ▪ title
	  ▪ body
	  ▪ pacientefk
	  ▪ fecha
	  ▪ image

Formato	JSON
URL	http://www.appsmedicas.com/appcoag/notificaciones/
Ejemplo Solicitud	HTTP GET http://www.appsmedicas.com/appcoag/notificaciones/2

Ejemplo Respuesta	{
	"status": "ok",
	"payload": {
	"notificaciones": [
	{
	"idnotificacion": 5,
	"title": "asunto maria",
	"body": "msg maria",
	"pacientefk": "2",
	"fecha": "2021-06-26T10:59:53.000Z",
	"image": "1624726793107_chulo.jpg"
	},
	]
	}
	}

Tabla 6 – Servicio Notificaciones

Nombre del Servicio	Error
Descripción	El usuario de la aplicación envía un comentario con la información del error que le ha surgido al usar la aplicación
Método	POST
Autorización	Java Web Token
Parámetros de entrada	• pacientefk • comentario
Respuesta	• status • payload
Formato	JSON
URL	http://www.appsmedicas.com/appcoag/error

Ejemplo Solicitud	{
	"pacientefk": 10,
	"comentario":"error"
	}

Ejemplo Respuesta	{
	"status": "ok",
	"payload": 8
	}

Tabla 7 – Servicio Error

Nombre del Servicio	Información Paciente
Descripción	Servicio utilizado para obtener la información relativa al paciente, su tratamiento, etc, que será mostrado en la aplicación.
Método	POST
Autorización	Java Web Token
Parámetros de entrada	• pacientefk

Respuesta	• status
	• payload
	  ○ info
	  ▪ nombre
	  ▪ apellidos
	  ▪ nid
	  ▪ tratamientos
	  • idtratamiento
	  • pacientefk
	  • medicamento
	  • comienzo
	  • fin
	  • dosis
	  • intervalo_horas
	  • recordatorio
	  • rams
	  ○ tratamientofk
	  ○ tipo_ramfk
	  ○ descripción
	  ○ fecha
	  ▪ tests
	  • idtest
	  • nombre
	  • preguntas
	  • pacientefk
	  ▪ testsRespondidos
	  • pacientefk
	  • testfk
	  • nombre
	  • preguntas
	  • respuestas
	  • fecha

Formato	JSON
URL	http://www.appsmedicas.com/appcoag/info

Ejemplo Solicitud	{
	"pacientefk": 2
	}

Ejemplo Respuesta	{
	"status": "ok",
	"payload": {
	"info": {
	"nombre": "p1",
	"apellidos": "p11",
	"nid": "10",
	"tratamientos": [],
	"tests": [],
	"testsRespondidos": [],
	}
	}
	}

Tabla 8 – Servicio Información Paciente

Nombre del Servicio	Información Autenticación
Descripción	Servicio utilizado para obtener el información relativa al paciente necesaria para enviar solicitudes al servidor. El paciente es identificado por el token JWT enviado.
Método	GET
Autorización	Java Web Token
Parámetros de entrada	Ninguno

Respuesta	• status
	• payload
	  ○ idpaciente
	  ○ nid
	  ○ username
	  ○ rol
	  ○ nombre
	  ○ apellidos

Formato	JSON
URL	http://www.appsmedicas.com/appcoag/auth
Ejemplo Solicitud	HTTP GET http://www.appsmedicas.com/appcoag/auth

Ejemplo Respuesta	{
	"status": "ok",
	"payload": {
	"idpaciente": 2,
	"nid": "222222222",
	"username": "maria",
	"rol": "paciente",
	"nombre": "María José",
	"apellidos": "Fernández Perez"
	}
	}

Tabla 9 – Servicio Información Autenticación

Nombre del Servicio	Solicitar Reacciones Adversas
Descripción	Servicio utilizado para obtener una lista con las distintas Reacciones Adversas al Medicamento que un paciente puede registrar en la aplicación
Método	GET
Autorización	Java Web Token
Parámetros de entrada	Ninguno

Respuesta	• status
	• payload
	  ○ tiposRam
	  ▪ idtipo_ram
	  ▪ descripcion

Formato	JSON
URL	http://www.appsmedicas.com/appcoag/listaram
Ejemplo Solicitud	HTTP GET http://www.appsmedicas.com/appcoag/listaram

Ejemplo Respuesta	{
	"status": "ok",
	"payload": {
	"tiposRam": [
	{
	"idtipo_ram": 1,
	"descripcion": "dolor de cabeza"
	},
	{
	"idtipo_ram": 2,
	"descripcion": "vómito"
	},
	]
	}
	}

Tabla 10 – Servicio Solicitar Reacciones Adversas

3.3 Firebase y notificaciones

Firebase, y más concretamente su solución Firebase Cloud Messaging es el componente necesario para hacer funcionar las notificaciones push de AppCoagulación.

Las notificaciones en la aplicación están administradas mediante Expo, y, aunque Expo permite la gestion de notificaciones a través de la API disponible para ello, siendo esta API la encargada de enviar las notificaciones a FCM, y de ahí a los distintos dispositivos finales, se decidió por no utilizar esta API ya que está mucho más limitada que si nos comunicamos directamente con FCM, como por ejemplo, la imposibilidad de enviar imágenes en la notificación.

En la siguiente imagen se puede ver un esquema del funcionamiento de la API de Expo de notificaciones:

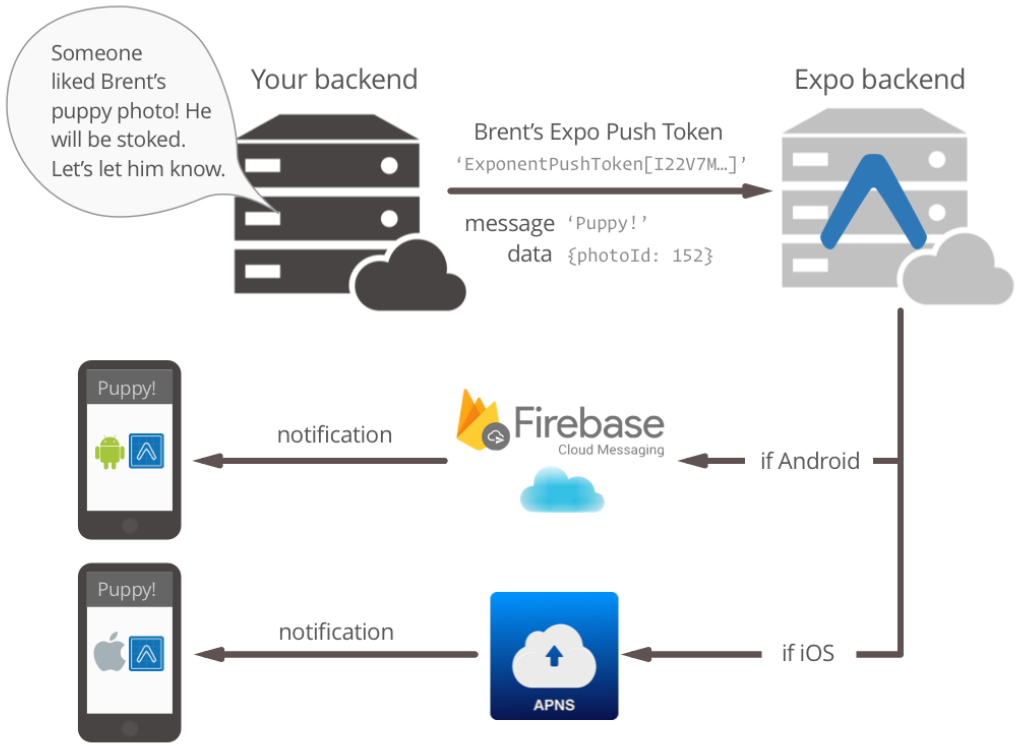


Figura 25: Notificaciones API Expo

Como el envío de imágenes en las notificaciones es un requisito de nuestra aplicación, se ha trabajado en el servidor directamente con FCM, para lo que se ha tenido que crear un servidor personalizado, usando además la API de FCM HTTP v1, que es la que permite imágenes y proporciona una mayor seguridad.

Para ello, es necesario generar un token OAuth 2.0, lo cual se hace de la siguiente manera:

```
32  // Carga la key del archivo json.
33  var serviceAccount =
34      require("./appcoagulacion-36b03-firebase-adminsdk-wg4q2-1b86f76840.json");
35
36  var scopes = [
37      "https://www.googleapis.com/auth/userinfo.email",
38      "https://www.googleapis.com/auth/firebase.messaging"
39  ];
40
41  // Autentica un cliente JWT con la cuenta de servicio.
42  var jwtClient = new google.auth.JWT(serviceAccount.client_email,
43      null, serviceAccount.private_key, scopes);
44
45  // Usa el cliente JWT para generar un token de acceso
46  jwtClient.authorize(function (error, tokens) {
47      if (error) {
48          console.log("Error generando token de acceso:",
49              error);
50      } else if (tokens.access_token === null) {
51          console.log("El servicio no tiene permiso para generar tokens de acceso");
52      } else {
53          var accessToken = tokens.access_token;
54          sendPushNotification(accessToken, token, title, body, image);
55      }
56  });
```

Figura 26: Obtención token OAuth 2.0

En la función jwtClient.authorize se obtiene el token de acceso necesario y se llama a la función sendPushNotification a la que se le pasa dicho token de acceso y el token de identificación de dispositivo, asunto, cuerpo e imagen que irán en la notificación.

```
async function sendPushNotification(accessToken, token, title, body, image) {
    fetch('https://fcm.googleapis.com/v1/projects/appcoagulacion-36b03/messages:send', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            Authorization: 'Bearer ' + accessToken,
        },
        body: JSON.stringify({
            message: {
                token,
                notification: {
                    title,
                    body,
                    image
                },
                data: {
                    experienceId: '@appcoagulacion/AppTFG2',
                    title,
                    message: {body},
                },
            },

        }),
    }).catch((error) => {
        console.error(error);
    });
}
```

Figura 27: Función de envío de notificaciones

En la función sendPushNotification se hará una solicitud HTTP POST a la URL de FCM, donde dentro de esta URL se indica el nombre de nuestra aplicación, en nuestro caso appcoagulacion-36b03.

En el cuerpo del mensaje enviado se pueden diferenciar tres campos: token, message y data, esto es debido a que la gestion de la notificación en Android es distinta dependiendo de si la aplicación está cerrada, que en ese caso la notificación es gestionada por el sistema operativo, o si la aplicación se encuentra abierta, en cuyo caso la notificación es administrada por la propia aplicación.

En el campo notification, que es el usado por el sistema de notificaciones de Android, se distinguen los atributos title, body e image, donde van respectivamente el asunto de la notificación, el cuerpo y la imagen.

Dentro del campo data, que es el que lee la aplicación para mostrar la notificación cuando está abierta, existen tres atributos: experienceId, que es el nombre con el que Expo conoce la aplicación a la que va dirigida la notificación, title, el asunto de la notificación, y message, el cuerpo de la notificación.

Como se puede ver, dentro de data no existe un campo image, esto es debido a lo comentado anteriormente de que Expo no permite el envío de imágenes en las notificaciones, por lo que tenemos la situación de que cuando el servidor envía la notificación, si la aplicación está cerrada o en segundo plano, la muestra el sistema Android junto con la imagen, y si la aplicación se encuentra abierta, no se verá la imagen en la notificación.

A continuación se muestra un diagrama de flujo en el que se explica de forma gráfica esta situación:

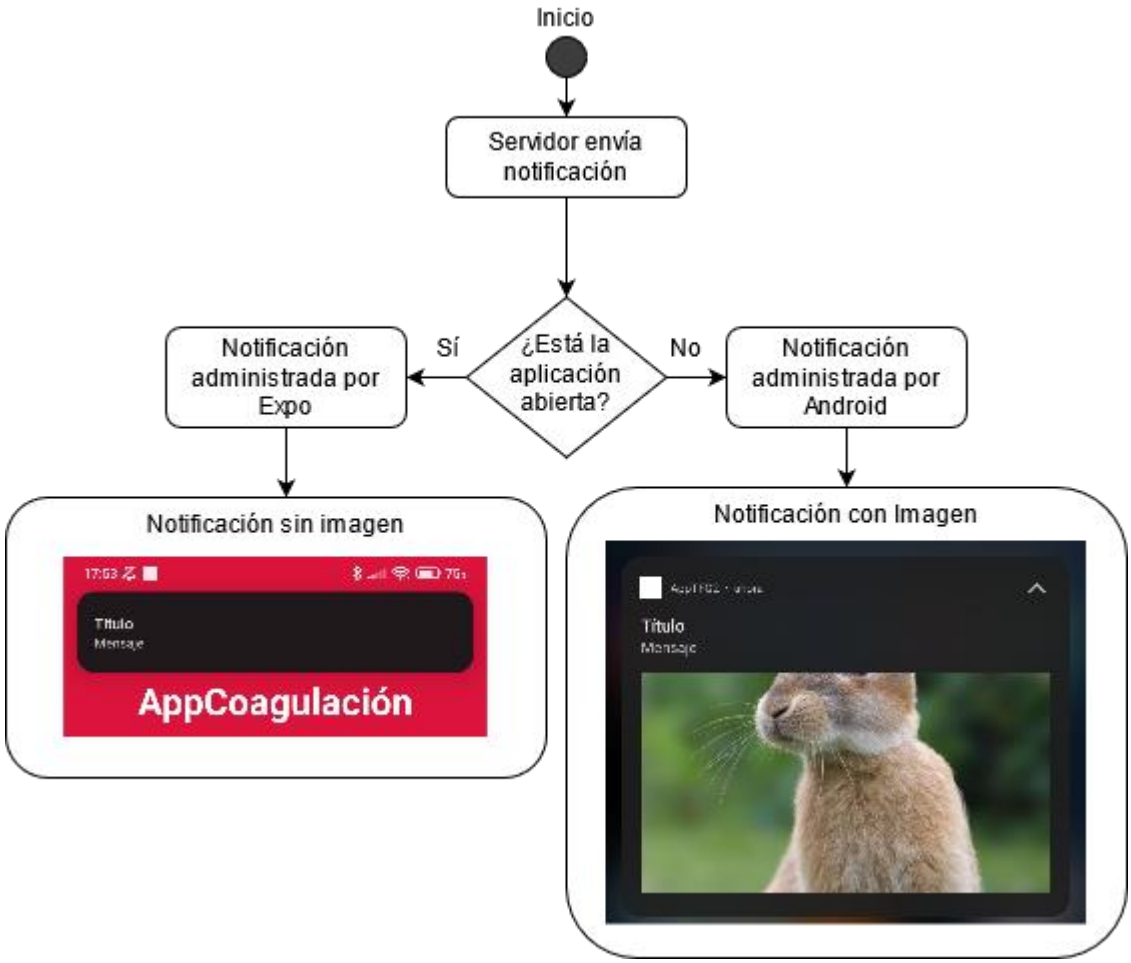


Figura 28: Diagrama de Flujo Comportamiento Notificaciones

3.4 Diagramas de secuencia

Con el fin de ilustrar cómo interactuan los distintos componentes de nuestro sistema se han realizado diagramas de secuencia en los que se muestran las distintas acciones que puede llevar a cabo en la aplicación.

3.4.1 Inicio de sesión

La autenticación entre el cliente y el servidor se lleva a cabo mediante JSON Web Token o JWT, que es un estándar creado por la IETF con el objetivo de proporcionar una forma de comunicación segura a través de un objeto JSON, y que en nuestro caso es usado para no tener que transmitir el usuario y la contraseña en los mensajes de petición de datos al servidor, si no que en su lugar, usuario y contraseña solo son enviados cuando iniciamos sesión, el servidor entonces comprueba que los datos sean correctos, y en caso de que así sea, nos envía nuestro token JWT, que será enviado en las solicitudes posteriores, y el servidor nos identificará a través de este.

En la aplicación, una vez se recibe este token, es guardado en la store correspondiente y se ejecuta la acción dispatch(SIGN_IN) lo que provoca que se actualice el valor del token en el Reducer que maneja los estados del flujo de autenticación, por lo que al existir un token válido, se navegaría a la pantalla de Inicio de la aplicación.

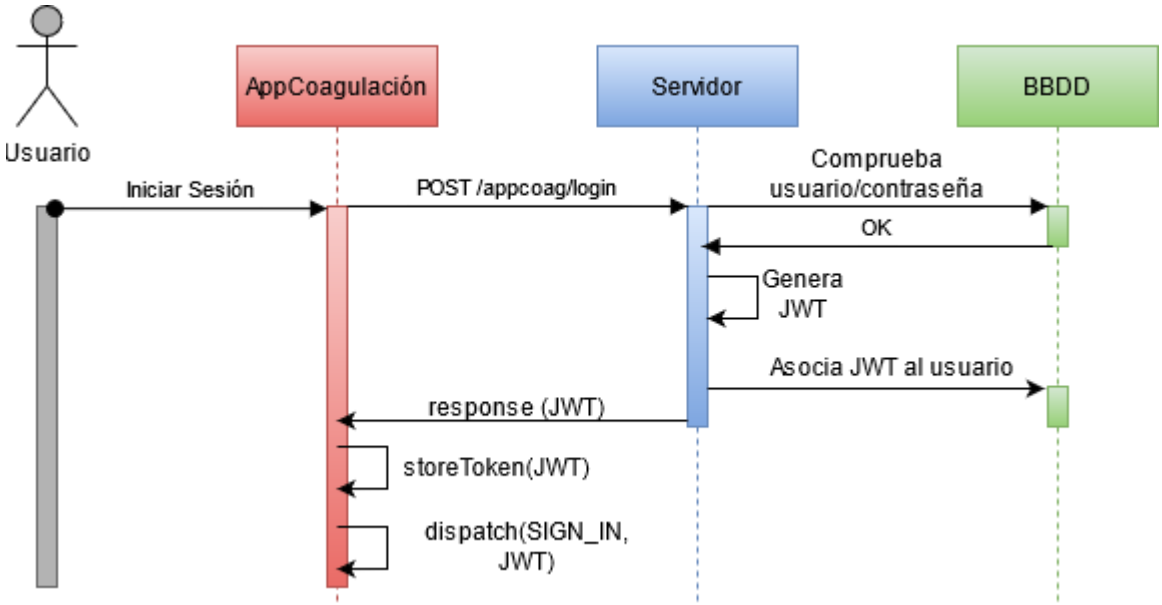


Figura 29: Diagrama de secuencia Login

3.4.2 Petición información de autenticación

Una vez se ha iniciado sesión correctamente, es necesario hacer una petición al servicio Información Autenticación para poder acceder al número de identificación del usuario, que será introducido en una store para las posteriores solicitudes de información.

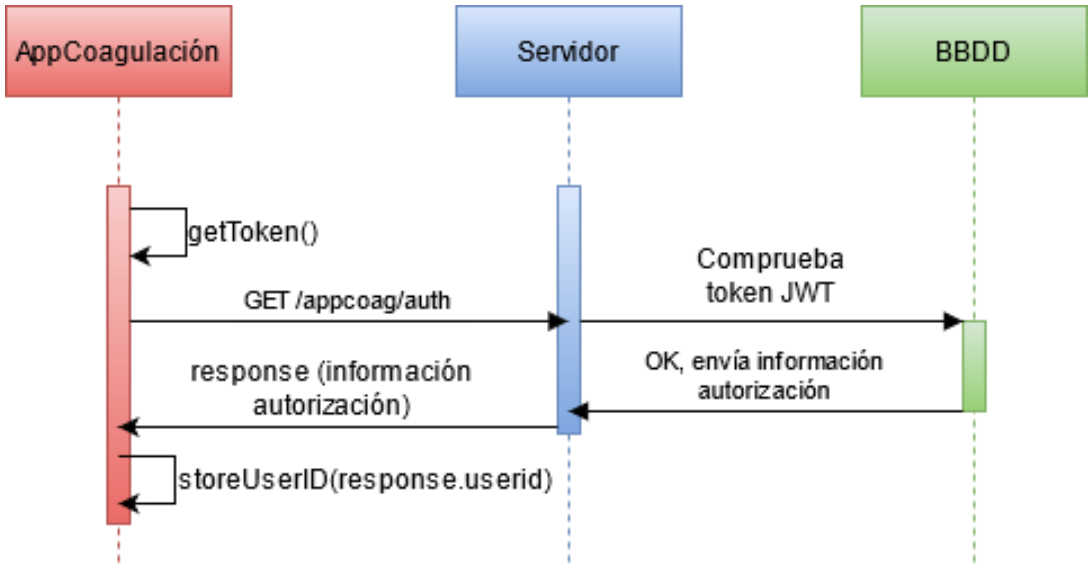


Figura 30: Diagrama de secuencia Petición información de autenticación.

3.4.3 Información Paciente

A continuación se muestra el diagrama de secuencia de las acciones que se producen y los mensajes que se envían entre los distintos agentes del sistema al entrar en la pantalla de Información del Paciente. En primer lugar se obtiene el token JWT y el numero de identificación de usuario para poder hacer la solicitud al servicio correctamente, una vez se ha obtenido la respuesta, se guarda en la storeInfo, y se navega a la pantalla de Información donde aparecerán los datos del paciente que acabamos de obtener.

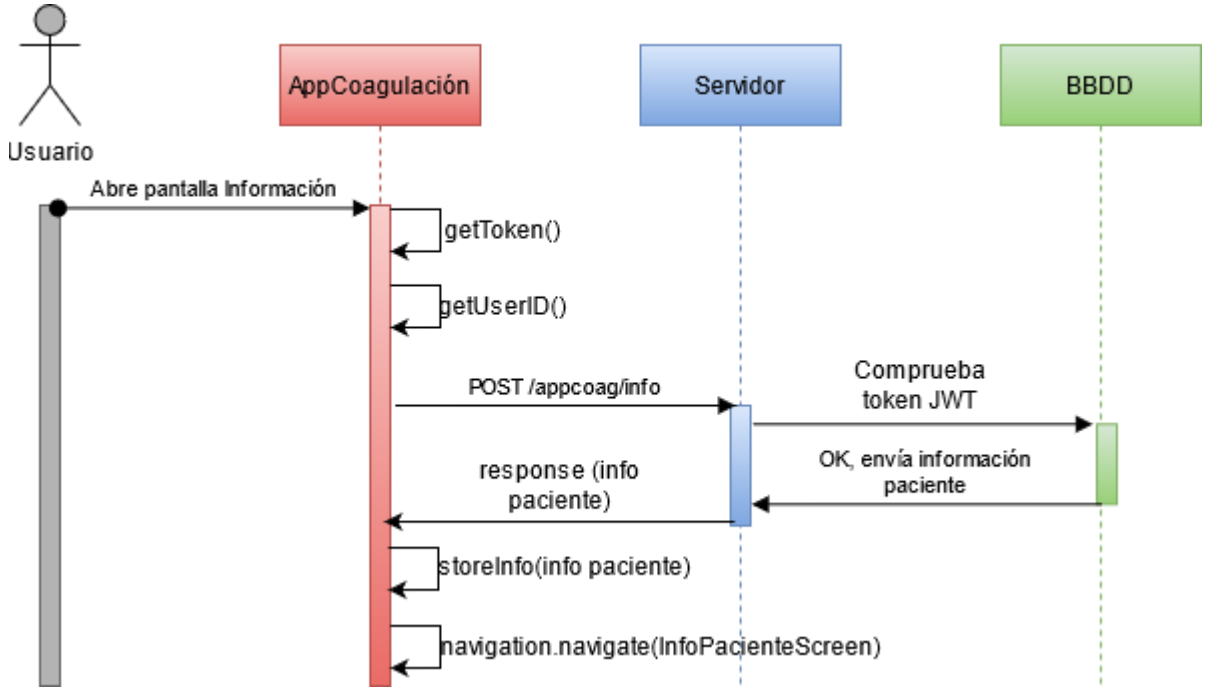


Figura 31: Diagrama de secuencia Información Paciente

3.4.4 Solicitar histórico

A continuación se muestra el diagrama de secuencia de las acciones producidad cuando el usuario solicita ver una gráfica del histórico de los valores de INR para un mes determinado. Una vez se ha entrado en la pantalla Histórico, se lee el token y el número de usuario necesarios para hacer la petición de la información, y se hace una solicitud al servicio Información que es el que contiene los valores de INR del paciente, por lo que al pulsar en el botón, se hace un mapeo seleccionando los valores referentes al mes y año indicados, y se muestran en pantalla en forma de gráfica.

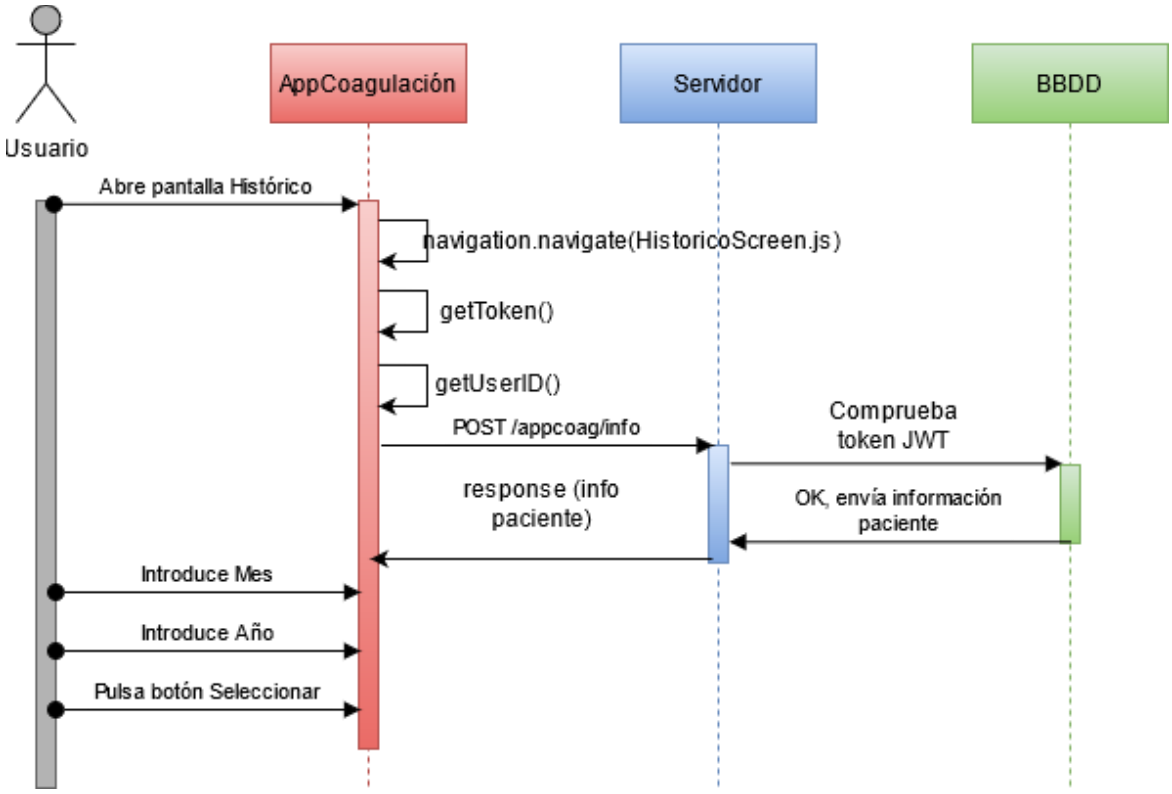


Figura 32: Diagrama de secuencia Solicitar histórico

3.4.5 Comunicar Reacción Adversa

Al entrar en la pantalla Comunicar Reacción Adversa, se lee el valor del token y del id del usuario necesarios para las peticiones posteriores, y se hace una llamada al servicio Información y al servicio Listar Reacciones Adversas, lo que nos proporciona una lista de las reacciones adversas posibles, que se muestran en una lista desplegable, y se mapea desde la información obtenida cuáles son los tratamientos que tiene el paciente, siendo estos mostrados en una lista seleccionable de la misma forma.

Una vez el paciente ha introducido los datos correctos, pulsa el botón enviar y se hace una solicitud al servicio Comunicar Reacción Adversa con dichos datos.

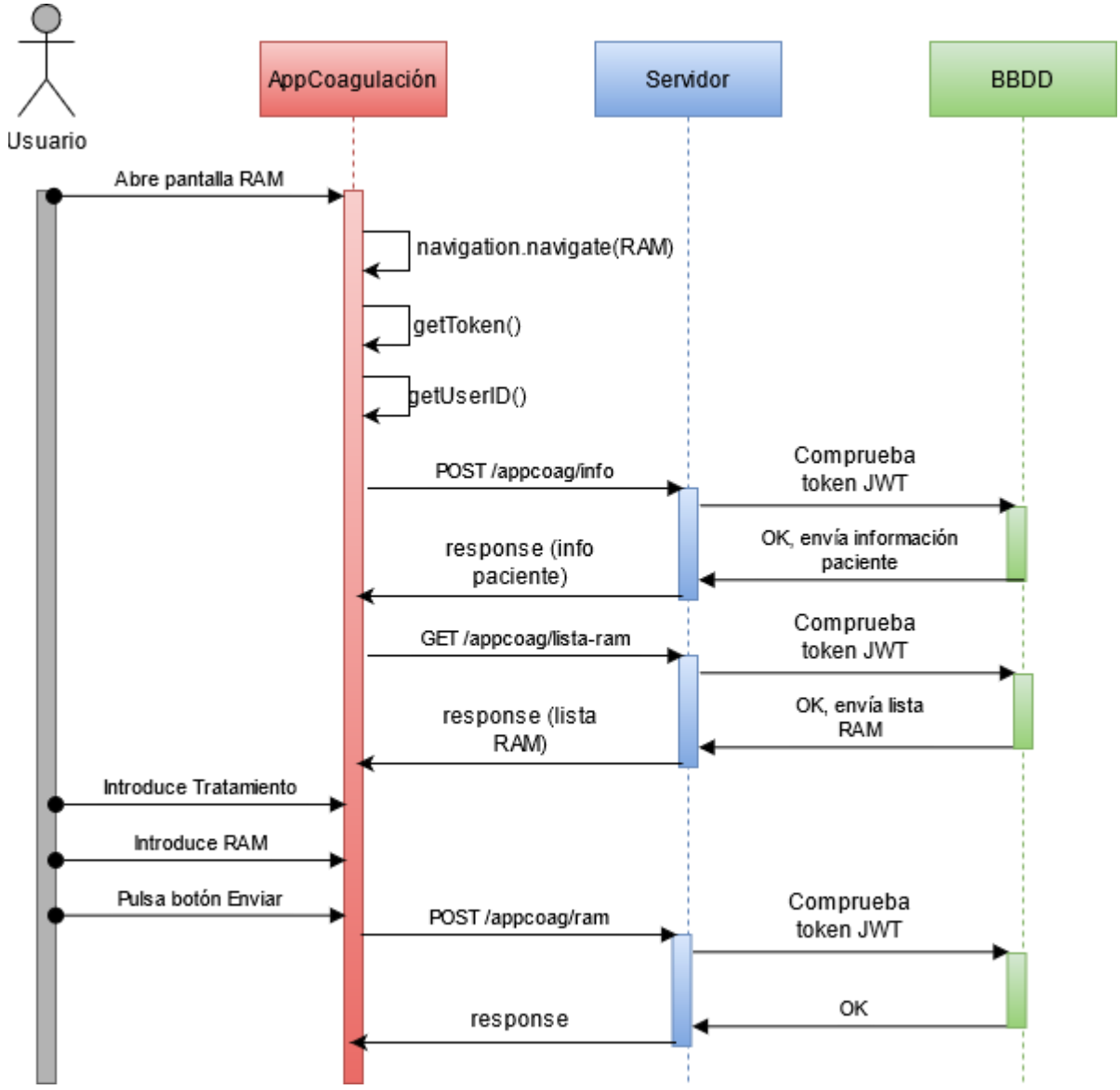


Figura 33: Diagrama de secuencia Comunucar Reacción Adversa

3.4.6 Envíar error app

Al entrar en la pantalla de envío de errores de la aplicación, se leen el token y el id del usuario necesarios para la solicitud posterior, y cuando el usuario introduce el error y pulsa en el botón de envío, se hace una solicitud al servicio Error, con dicho comentario.

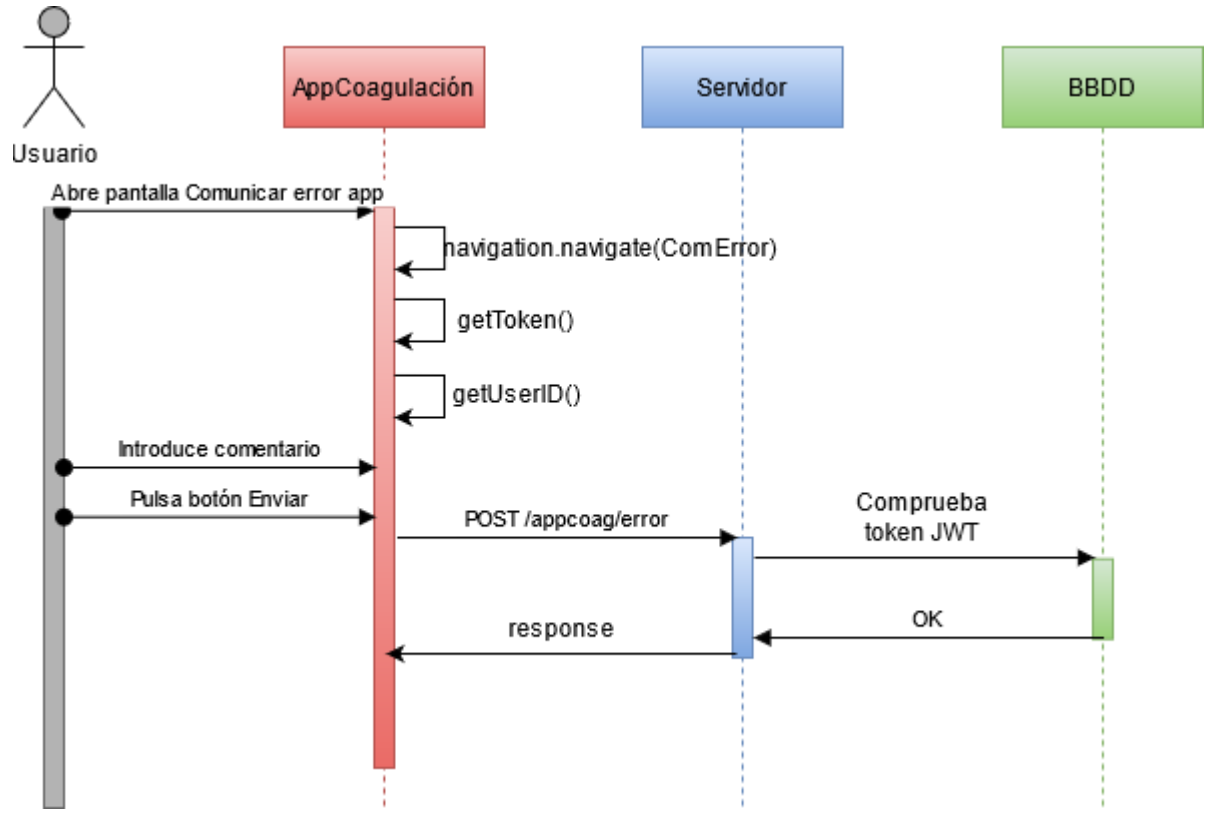


Figura 34: Diagrama de secuencia Enviar error app

4 FUNCIONALIDAD

En este capítulo se tratará de mostrar el funcionamiento de forma detallada de la aplicación AppCoagulación explicando cada uno de sus apartados y cómo se maneja, y servirá a su vez de manual de usuario de la aplicación.

Para ello, se han tomado intantáneas de todas sus pantallas con el fin de poder enseñar de forma gráfica todas y cada una de sus funciones.

4.1 Pantalla Login

En la Pantalla de Login se muestra el nombre de la aplicación junto con su logo, y consiste en dos campos de entrada de texto, para el nombre de usuario y su correspondiente contraseña, un botón para iniciar sesión y entrar en la aplicación, y otro botón que de forma predeterminada se mostrará como un ojo abierto, indicando que si es pulsado, se podrá ver la contraseña en su campo de texto.

En la siguiente imagen se puede ver el aspecto de la pantalla de Login.

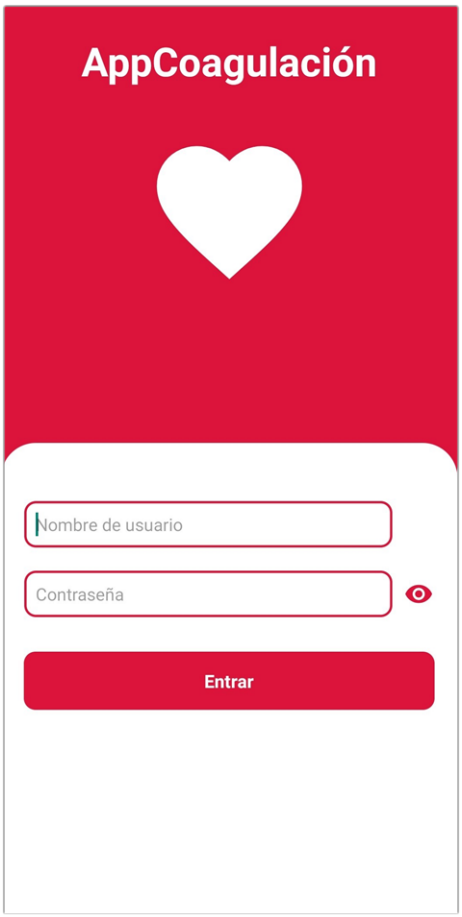


Figura 35: Pantalla de Login

De forma predeterminada la contraseña estará oculta, y al pulsar encima del botón con el ojo, la contraseña que se esté escribiendo pasará a ser visible y el botón pasará a ser un ojo tachado.

En las siguientes figuras se muestra el comportamiento descrito anteriormente al pulsar el botón de mostrar/ocultar contraseña:

Figuras 36 y 37: Pantallas de Login, contraseña oculta y Pantalla de Login contraseña visible.

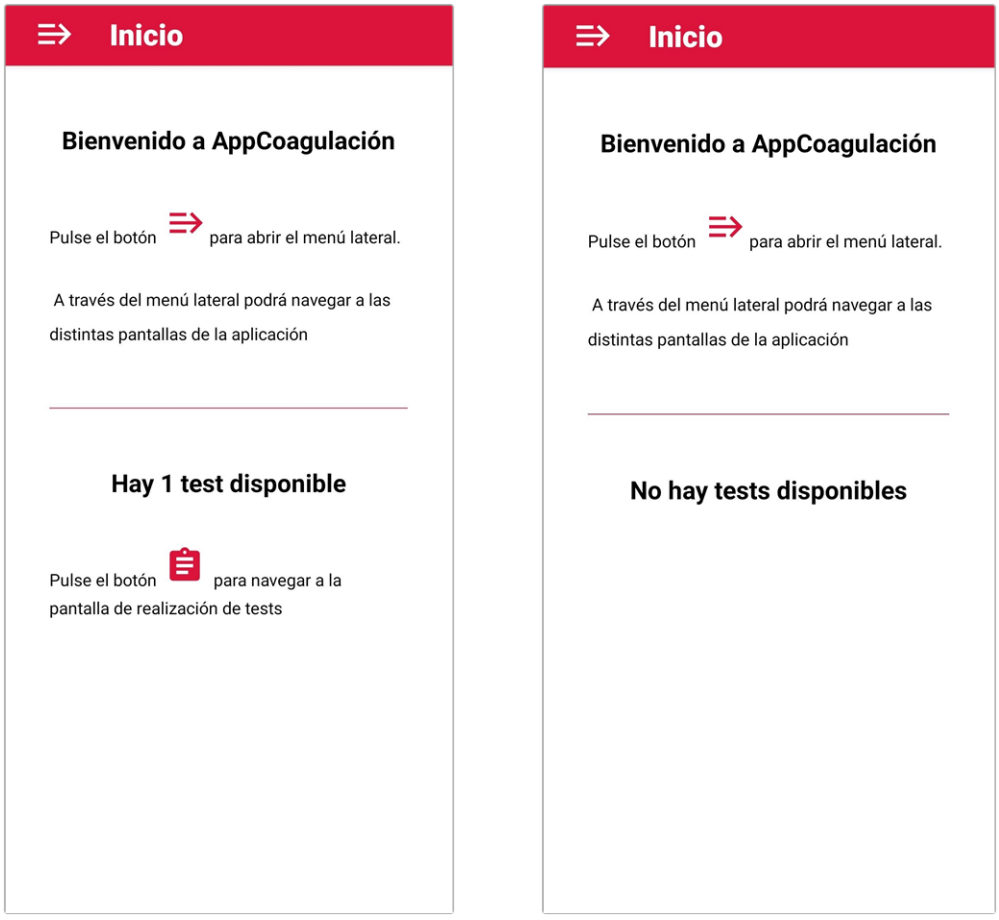
4.2 Pantalla Inicio

Al introducir el usuario y la contraseña de forma correcta, entraremos en la aplicación y la pantalla de Inicio será la primera pantalla que veamos, en ella nos encontraremos con un texto de bienvenida y una explicación de cómo se abre el menu lateral para poder navegar a través de la aplicación.

También se muestra un texto en el que podremos ver si existen tests disponibles para ser realizados. En el caso de que existan tests disponibles, se verá un texto indicando cómo ir a la pantalla de realización de los tests.

Ambos botones ilustrativos son interactivos, es decir, si el botón de menu lateral que se muestra en el texto de explicación es pulsado, se abrirá el menú lateral, de la misma forma con el botón de realización de tests, que nos enviará a la pantalla correspondiente.

En las siguientes imágenes se muestra la pantalla de Inicio cuando hay un test disponible y cuando no existen tests disponibles respectivamente.



Figuras 38 y 39: Pantalla de Inicio con test y Pantalla de Inicio sin test

4.3 Menú Lateral

La navegación a través de la aplicación será realizada mediente este menu lateral desplazable, que podrá ser abierto deslizando un dedo desde el borde izquierdo de la pantalla hacia la derecha, o a través del botón que se encontrará disponible en los títulos de todas las pantallas de la aplicación.

Este menú contiene botones a las pantallas de la aplicación acompañados por iconos representativos de cada una de las pantallas, y un botón para cerrar sesión que al ser pulsado nos llevará de vuelta a la pantalla de Login.

El menú puede cerrarse desplazándolo hacia la izquierda, pulsando en el botón superior, o pulsando fuera del menú.

En la siguiente imagen se muestra cómo se ve el Menú una vez abierto:

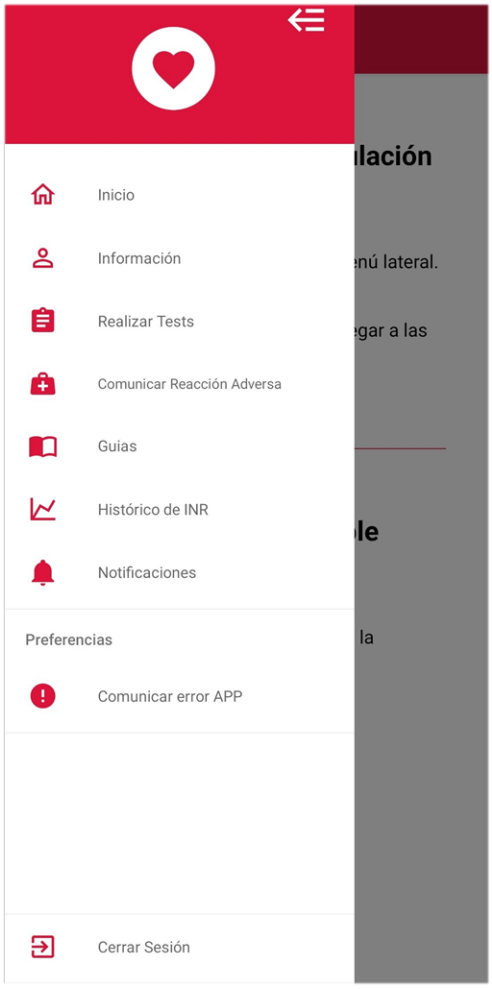


Figura 40: Menú Lateral

4.4 Pantalla Información

En la pantalla de información se muestran los datos del paciente: nombre y apellidos, número de identificación y distintas fichas en las que se muestran los tratamientos que el paciente tiene o ha tenido asignados, junto con las fechas en las que comenzó esos tratamientos y las fechas en las que termina, así como una lista de las reacciones adversas que han sido registradas, tanto por el paciente mediante la aplicación a través de la pantalla de Comunicar Reacción Adversa, como por el médico.

En la siguiente imagen se ve el aspecto de la pantalla de Información con los datos del paciente:

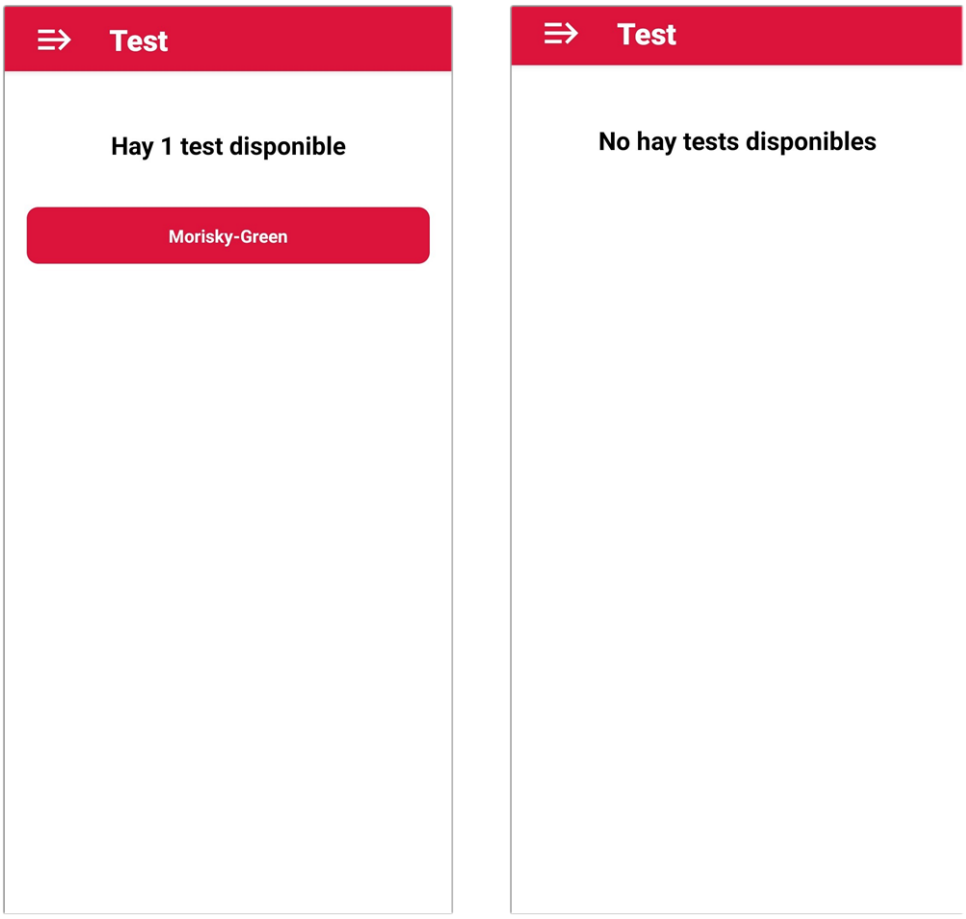
Figura 41: Pantalla Información

4.5 Pantalla Test

En la pantalla Test se mostrará en primer lugar un texto que nos indicará si tenemos tests disponibles o no, y el número de tests disponibles en el caso de que sí haya tests listos para ser realizados.

Si existen tests disponibles, se verá una lista con distintos botones con los nombres de los tests que al ser pulsados nos llevarán a la pantalla de Realización de Test con los datos de dicho test.

En las siguientes imágenes se muestra la pantalla de Test cuando existe un test disponible con su botón correspondiente (Figura 42), y cuando no hay ningún test listo para ser realizado (Figura 43).



Figuras 42 y 43: Pantalla Test con test disponible y Pantalla Test sin test disponible

4.6 Pantalla Realizar Test

A esta pantalla solo se puede acceder mediante la Pantalla de Test cuando hay algún test disponible, al ser pulsado el botón del test que se quiere realizar.

Una vez dentro de esta pantalla, aparecerá el nombre del test y una lista de preguntas que deben ser respondidas con "Si" o "No" a través de los botones radiales que tendrán asociados cada una de estas preguntas.

Al pulsar en el botón "Enviar", recibiremos un mensaje indicando que el test se ha enviado correctamente y volveremos a la pantalla de Test.

En la siguiente imagen se ve la pantalla de Realizar Test con el test de Morisky-Green cargado, con sus respectivas preguntas y botones de "Si" y "No".

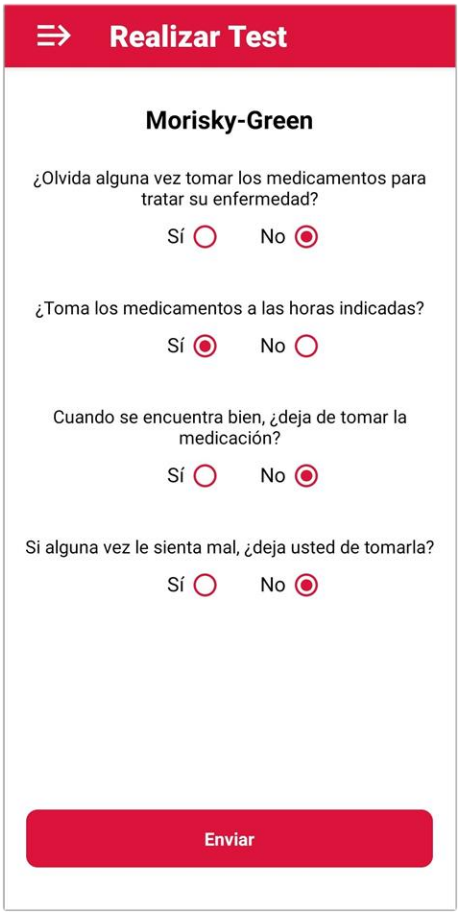


Figura 44: Pantalla Realizar Test

4.7 Pantalla Comunicar Reacción Adversa al Medicamento

Al acceder a la pantalla de Comunicar Reacción Adcersa al Medicamento se nos mostrarán dos listas seleccionables, la primera lista nos permitirá elegir un tratamiento entre todos los tratamientos que el paciente tiene asociados, y la segunda lista contendrá todas las reacciones adversas posibles que pueden ser producidas por acción de estos fármacos.

Cuando se haya producido alguna de estas reacciones en el paciente, este deberá registrarla utilizando esta pantalla y de esta forma el médico podrá tener constancia del problema.

En la Figura 45 se muestra el aspecto de la Pantalla Comunicar Reacción Adversa al acceder a ella.

Si se pulsa el botón sin que se haya seleccionado adecuadamente un tratamiento o reacción, el usuario recibirá un aviso indicando dónde está el problema. En las Figuras 46, 47 y 48 se muestran los distintos errores indicando que el usuario debe seleccionar de forma adecuada los parámetros necesarios.

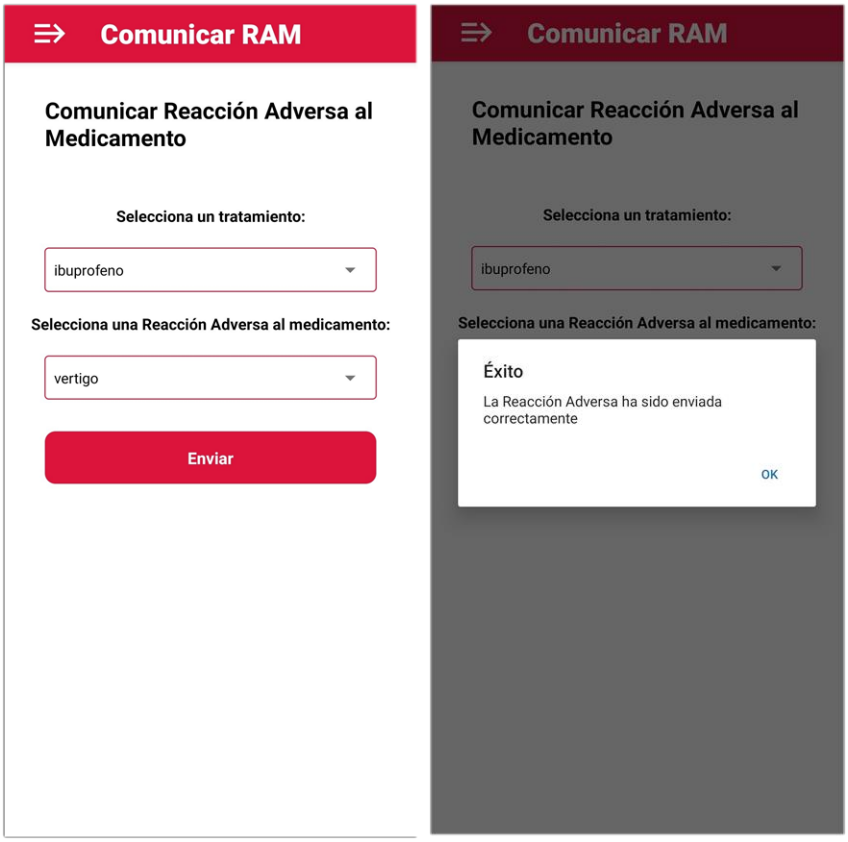
En las figuras 49 y 50 se muestran las listas desplegadas de tratamientos y reacciones respectivamente.

Una vez se haya registrado de forma correcta la reacción adversa por el usuario, esta aparecerá en la pantalla de Información, como se puede ver en las Figuras 51 y 52, donde se está registrando una reacción de vértigo para el tratamiento de ibuprofeno, y en la figura 53 se puede ver que la información ha sido introducida correctamente y ahora aparece la nueva reacción.

Figura 45: Pantalla Comunicar Reacción Adversa al Medicamento

Figuras 46, 47, y 48: Error RAM tratamiento y reacción, Error RAM reacción y Error RAM tratamiento

Figuras 49 y 50: Lista seleccionable Tratamientos y Lista seleccionable Reacción



Figuras 51 y 52: Pantalla Comunicar RAM y Éxito envío RAM

Figura 53: Pantalla Información Paciente actualizada con datos de la Reacción Adversa registrada

4.8 Pantalla Guías

En esta pantalla se encontrarán una serie de guías en forma de enlaces a páginas web, imágenes, vídeos, etc, útiles para el paciente a través de enlaces que al ser pulsados abrirán un navegador con el enlace correspondiente.

En la siguiente imagen se muestra el aspecto de la pantalla Guías:

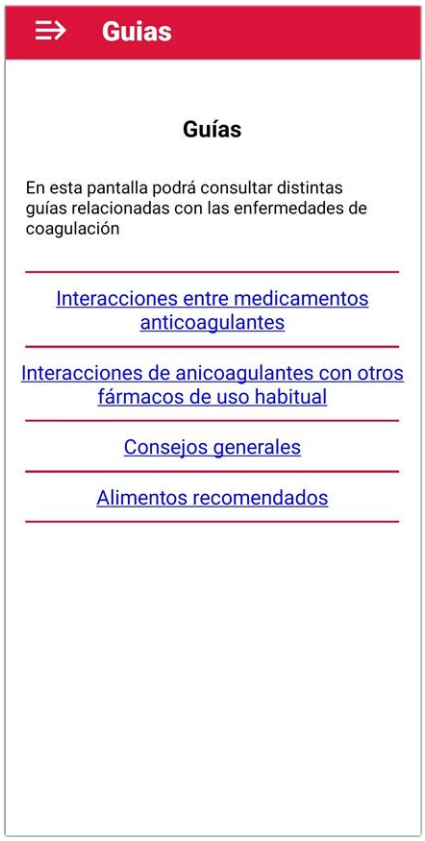


Figura 54: Pantalla Guías

4.9 Pantalla Histórico

En la pantalla Histórico se encuentran dos listas desplegables para seleccionar un mes y un año respectivamente, de manera que al pulsar en el botón "Seleccionar" se mostrará una gráfica con los datos de los valores de INR registrados del paciente en dicho mes seleccionado.

En la siguiente imagen se muestra la pantalla Histórico:

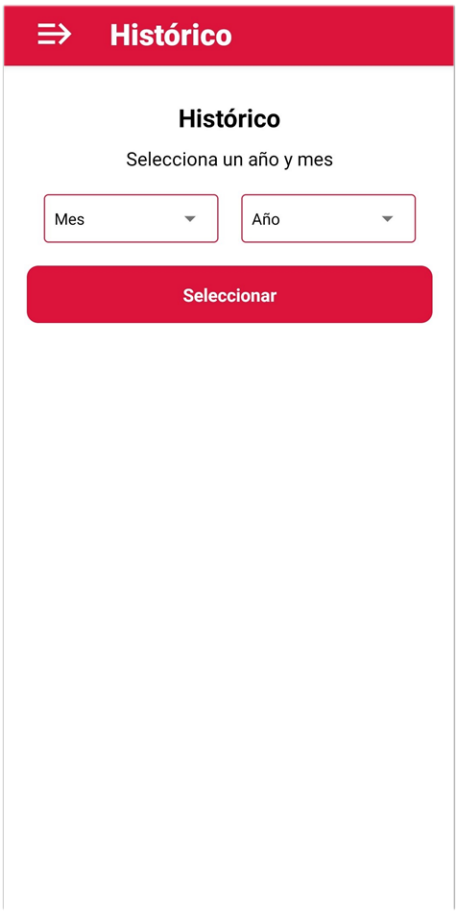


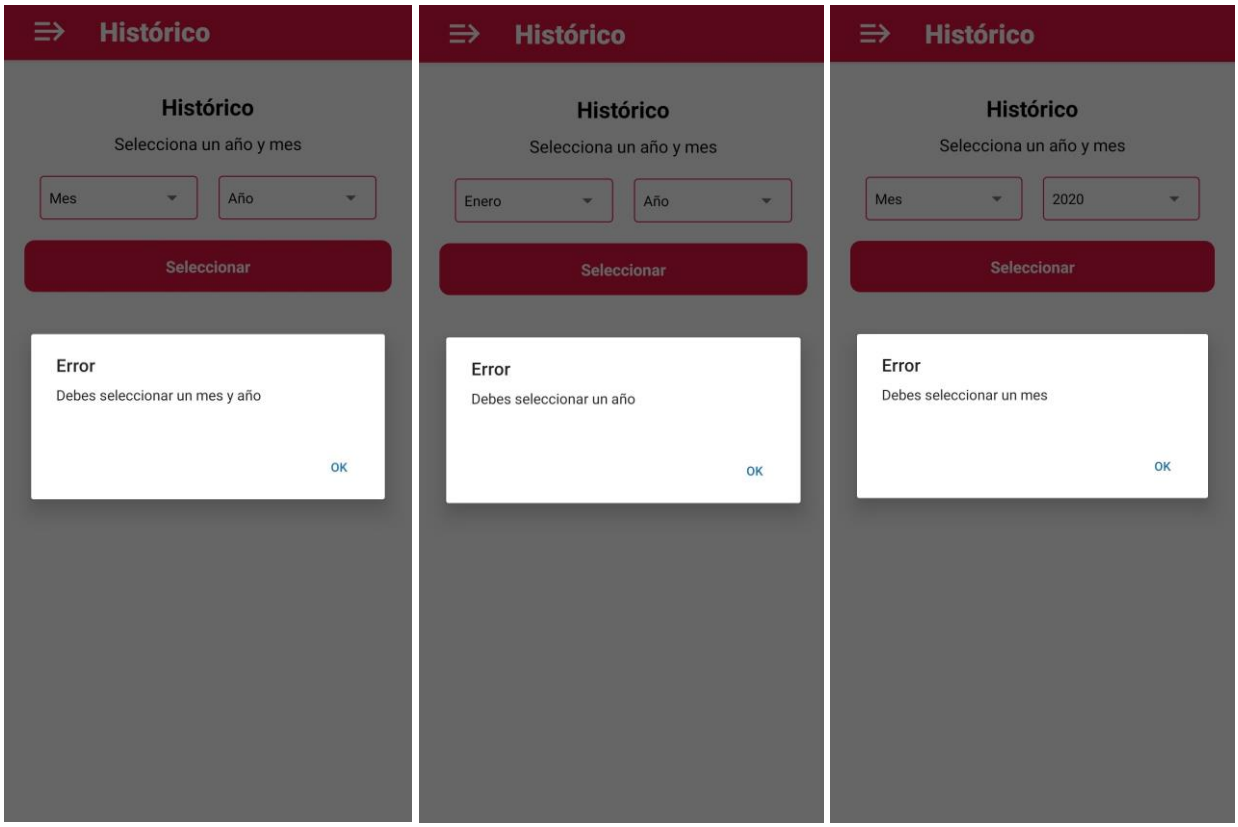
Figura 55: Pantalla Histórico

Cuando se selecciona un mes y año de forma correcta, si existen datos para esa fecha, se mostrará una gráfica con los valores de INR en el eje vertical y los valores de fechas en el eje horizontal. En el caso de que no exista ningún valor para el mes seleccionado, se mostrará un texto que indica que no hay datos para ese mes.

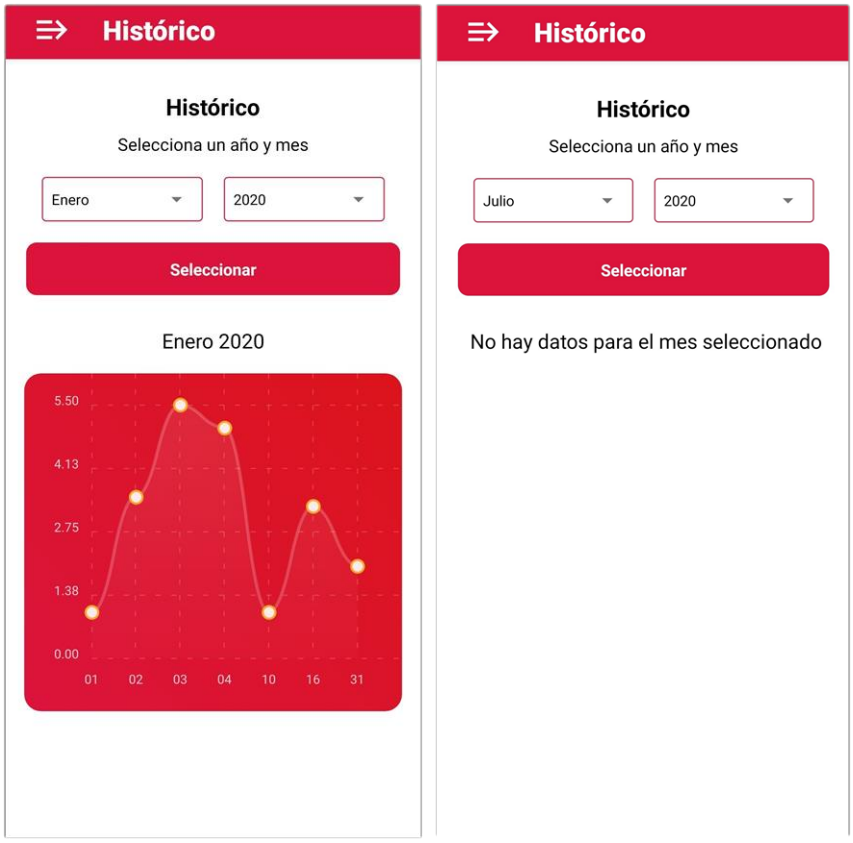
En las Figuras 59 se muestra la gráfica dibujada al seleccionar enero del 2020, y en la Figura 60 se puede ver el mensaje indicando que no existen datos para julio de 2020.

De la misma forma que en la anterior pantalla Comunicar Reacción Adversa, existen avisos de error para asegurar que el mes y el año han sido seleccionados correctamente.

En las Figuras 56, 57 y 58, se muestran los distintos mensajes de error que pueden verse.



Figuras 56, 57 y 58: Error año y mes, Error año y Error mes



Figuras 59 y 60: Gráfica Histórico y Sin datos Histórico

4.10 Pantalla Notificaciones

En la pantalla Notificaciones se podrán ver todas las notificaciones que el usuario ha recibido a través de la aplicación. Estas notificaciones se mostrarán en forma de tarjetas separadas con un Asunto, Mensaje, la Fecha en la que se envió la notificación, y de forma opcional una imagen. Dicha imagen es un elemento clickable, que al ser pulsado abrirá un navegador con en enlace de la imagen para que el usuario pueda ampliarla para verla mejor o guardarla en su dispositivo.

Figura 61: Pantalla Notificaciones

4.11 Pantalla Comunicar Error APP

En la Pantalla Comunicar Error APP nos encontramos con un campo de texto en el que el usuario puede escribir un comentario con los posibles errores que se haya encontrado al usar la aplicación.

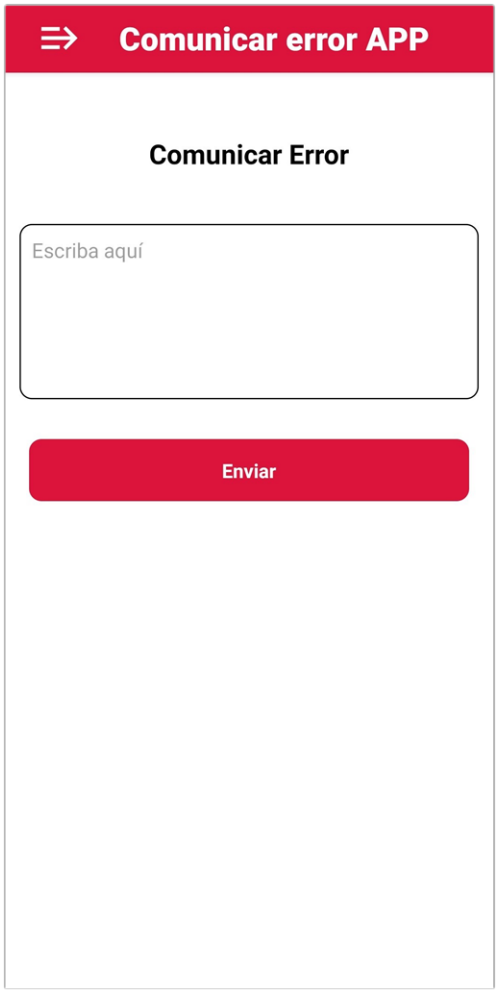


Figura 62: Pantalla Comunicar Error APP

5 PLIEGO DE CONDICIONES

En este capítulo se recogerán cuáles han sido los requisitos necesarios para llevar a cabo este proyecto, tanto de hardware como de software, de forma que si alguien quisiera replicarlo o poder seguir trabajando en la aplicación de cara a añadirle nuevas funciones, conozca cuáles son los elementos necesarios para ello.

Este proyecto ha sido realizado trabajando en un ordenador con un sistema operativo Windows, por lo que los pasos que comentaré para hacerlo funcionar son lo que se han seguido para este sistema operativo, pero es conveniente decir que todas las herramientas indispensables para su realización están también disponibles tanto en Linux como en Macintosh.

Las características del ordenador utilizado son las siguientes:

- Sistema operativo: Windows 10
- Procesador: Intel Core i5 7th Gen
- Memoria: 8 GB RAM

También ha sido utilizado un dispositivo móvil Android en el que se ha instalado la aplicación Expo Go, disponible a través de Google Play Store, que ha sido utilizado para ejecutar y testear la aplicación desarrollada. Este no es un elemento indispensable, ya que es posible ejecutar un simulador de Android a través de Android Studio en el ordenador donde se esté desarrollando la aplicación para realizar las tareas de testeo.

Una vez tenemos todos los elementos de hardware necesarios, proseguimos con los elementos necesarios para llevar a cabo el proyecto:

5.1 Node.js

Es el entorno de ejecución de JavaScript [27] necesario para hacer correr las aplicaciones React Native, por ello es el primer elemento que tendremos que instalar.

La versión de Node.js mínima necesaria para hacer funcionar la versión de React Native con la que vamos a desarrollar es la 12 LTS, aunque existe una versión LTS (Long Time Support) más actualizada, la 14, que es la que ha sido utilizada en mi caso.

Junto con Node.js también se instalará npm, que es un sistema de gestión de paquetes que será necesario para realizar las instalaciones de los siguientes elementos.

5.2 React Native con Expo

Como ya se había comentado en los capítulos previos, existen dos formas de utilizar React Native, mediante React Native CLI, o a través del Expo CLI. En este caso se decidió por usar Expo CLI por las numerosas ventajas que supone con respecto a React Native CLI, y estos son los pasos necesarios para instalar y crear nuestra aplicación React Native a través de Expo:

En nuestro terminal, en el caso de Windows será el "Símbolo del Sistema" o Windows PowerShell, ejecutaremos el siguiente comando:

```
npm install -g expo-cli
```

De esta forma se instalará React Native de forma conjunta con Expo, y a través de los siguientes comandos:

```
expo init nueva-app
```

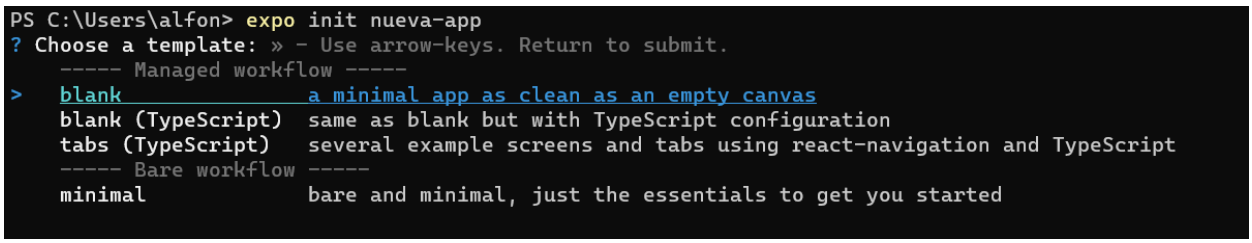


Figura 63: Crear app Expo

Lo cual nos mostrará un asistente que nos preguntará si queremos empezar con un workflow administrado o simple, y nos dará a elegir varias plantillas. Una vez hayamos realizado nuestra elección, navegamos al directorio correspondiente, con el nombre que le hayamos dado anteriormente y ejecutamos el servidor de desarrollo mediante Expo:

```
cd nueva-app
expo start
```

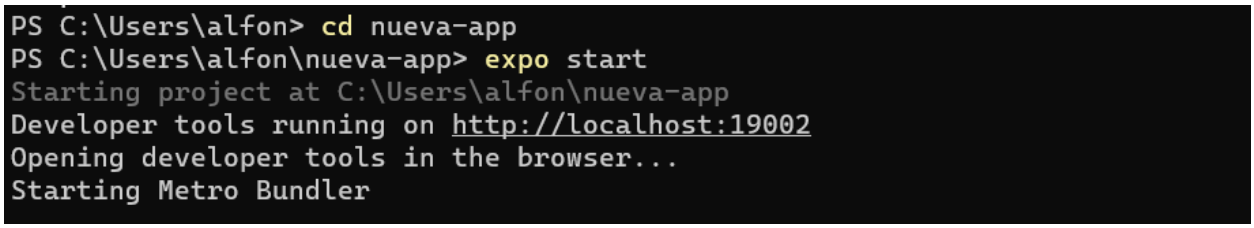


Figura 64: Ejecutar servidor app Expo

De esta forma, ya tendremos nuestra aplicación creada y siendo ejecutada en un servidor local lista para ser ejecutada a través de las distintas opciones que nos da Expo. Este último comando nos habrá abierto una ventana de un navegador como se nos muestra en la Figura 65, donde podremos ver información sobre el estado de nuestra app y nos proporciona el código QR necesario para ejecutar la aplicación en nuestro dispositivo mediante Expo Go, así como la posibilidad de ejecutar la aplicación en web o mediante un simulador en caso de que lo tengamos correctamente configurado.

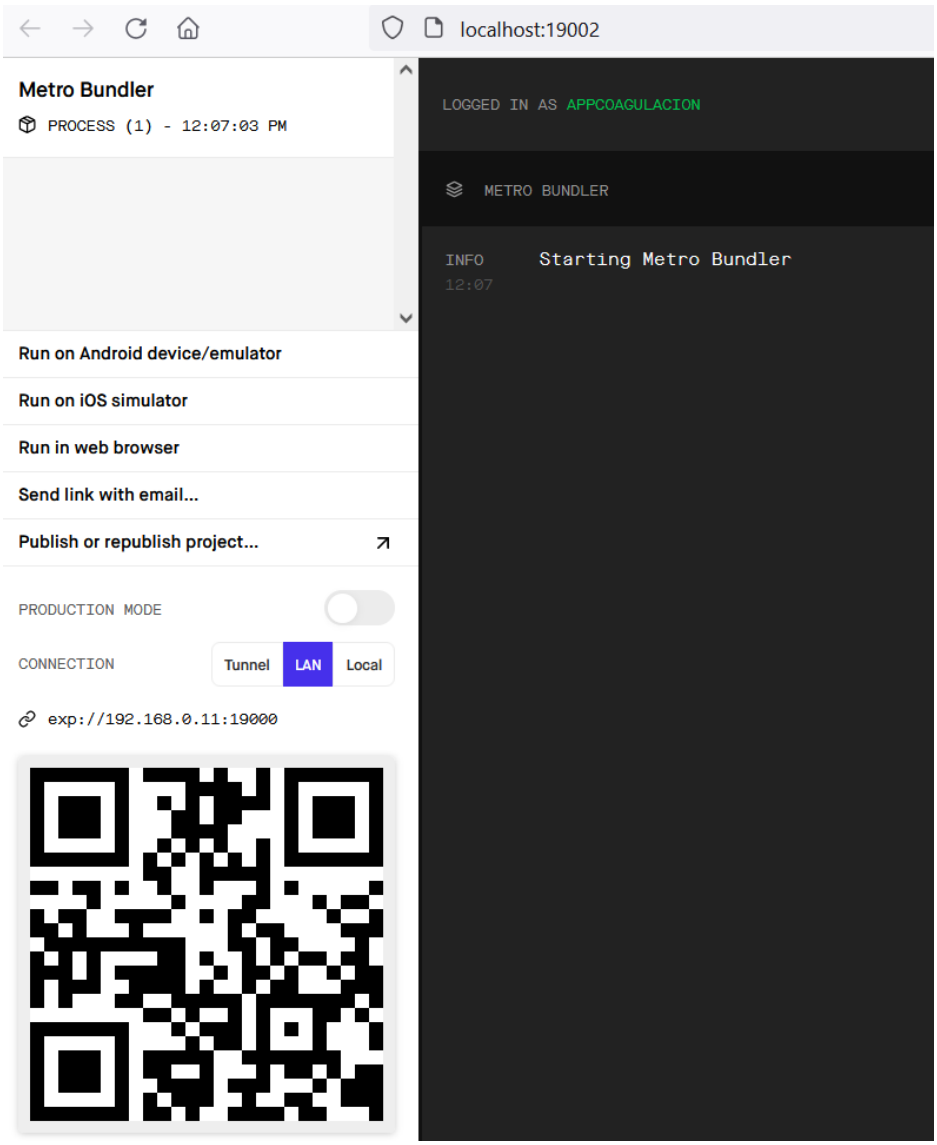


Figura 65: Interfaz del servidor creado por Expo ejecutándose en un navegador web

Así, si escaneamos en nuestro dispositivo móvil mediante la aplicación Expo Go este código QR, la aplicación será mostrada en nuestro teléfono, y al editar el código de la aplicación podremos ver los cambios realizados en tiempo real.

Para la tarea del desarrollo del código de la aplicación he utilizado Visual Studio Code, el editor de código de Microsoft, aunque se puede utilizar cualquier otro.

5.3 Módulos y dependencias

Una vez tenemos todas las herramientas necesarias para el desarrollo de la aplicación, se entrará en materia en cúales han sido los módulos utilizados en React Native y sus dependencias.

En el archivo "package.json" de AppCoagulación, en el apartado "dependencies", se encuentran las dependencias y las versiones de los módulos utilizados:

```
"dependencies": {
  "@react-native-async-storage/async-storage": "^1.15.2",
  "@react-native-community/masked-view": "0.1.10",
  "@react-native-picker/picker": "1.9.2",
  "@react-navigation/bottom-tabs": "^5.11.9",
  "@react-navigation/drawer": "^5.12.5",
  "@react-navigation/native": "^5.9.4",
  "@react-navigation/stack": "^5.14.4",
  "expo": "~40.0.0",
  "expo-status-bar": "~1.0.3",
  "expo-web-browser": "~8.6.0",
  "react": "16.13.1",
  "react-dom": "16.13.1",
  "react-native": "https://github.com/expo/react-native/archive/sdk-40.0.1.tar.gz",
  "react-native-chart-kit": "^6.11.0",
  "react-native-gesture-handler": "~1.8.0",
  "react-native-paper": "^4.8.1",
  "react-native-reanimated": "~1.13.0",
  "react-native-safe-area-context": "3.1.9",
  "react-native-screens": "~2.15.2",
  "react-native-svg": "12.1.0",
  "react-native-svg-charts": "^5.4.0",
  "react-native-web": "~0.13.12",
  "expo-notifications": "~0.8.2"
},
```

Figura 66: Dependencias de la aplicación.

Muchos de los elementos que aparecen se instalan de forma predeterminada al crear nuestra aplicación mediante el Expo CLI, pero voy a destacar los siguientes módulos que han sido clave en el desarrollo y que han sido creados por la comunidad:

- React Navigation: Es una librería que nos proporciona todas las herramientas necesarias para la navegación en nuestra aplicación. Nos permite crear Stacks, Navegadores anidados, pestañas de navegación, menús, etc. Es completamente indispensable en cualquier aplicación que esté formada por más de una pantalla.
- React Native Async Storage: es una librería que nos proporciona una forma asíncrona y persistente de guardar datos en nuestra aplicación. Es indispensable en el caso de una aplicación como la nuestra, en la que trabajamos con tokens de autorización JWT, para poder almacenarlos y permitir las peticiones de red mediante JWT, o para almacenar datos que no queremos que se pierdan al cerrar nuestra aplicación.
- React Native Picker [28]: Permite utilizar listas desplegables. En AppCoagulación ha sido utilizado en las pantallas Comunicar Reacción Adversa al Medicamento e Histórico.
- React Native Chart Kit [29]: es un módulo que utiliza React Native SVG y React Native SVG Charts para pintar gráficas con un estilo moderno y estilizado. Estas gráficas son completamente personalizables y más sencillas de ser usadas que utilizando React Native SVG de forma

independiente.

- React Native Paper: Es un conjunto de componentes que siguen las líneas de estilo de Google Material Design. Esta librería ha sido utilizada para el Menú Lateral ya que nos permite una mayor personalización al poder introducir iconos, dividir el menú en secciones, etc, y también añade botones radiales que han sido utilizados en la pantalla de Realización de Test.

Por último, no aparece en la lista anterior, pero también se ha utilizado un repositorio de iconos hechos por la comunidad llamado React Native Vector Icons [30], que incorpora gran cantidad de iconos útiles para crear el estilo de nuestra aplicación.

Todas estas herramientas han de ser instaladas a través de la consola de comandos utilizando para ello npm.

5.4 Notificaciones Expo y Firebase

Como ya se ha explicado en los apartados anteriores, se ha utilizado Expo Notifications para gestionar las notificaciones en nuestra app, que trabaja junto a Firebase Cloud Message, por lo que ahora se van a comentar los pasos necesarios ello:

En primer lugar, la instalación de Expo Notifications a través de npm:

```
expo install expo-notifications
```

Una vez instaladas las dependencias e integradas en el código de nuestra aplicación, debemos configurar Firebase correctamente, para ello tenemos que agregar nuestro proyecto en la consola de Firebase:

Figura 67: Agregar proyecto Firebase

Debemos darle un nombre y aceptar el uso de Google Analytics para poder utilizar las funciones de Cloud, por lo que aceptaremos.

Una vez creada la aplicación en Firebase, debemos agregar nuestra app de Android al proyecto:

Figura 68: Agregar app Firebase

Lo siguiente que debemos hacer es darle un nombre de paquete a nuestra app, el nombre de paquete que le ponemos debe ser el mismo que tengamos en nuestro archivo app.json en android.package.

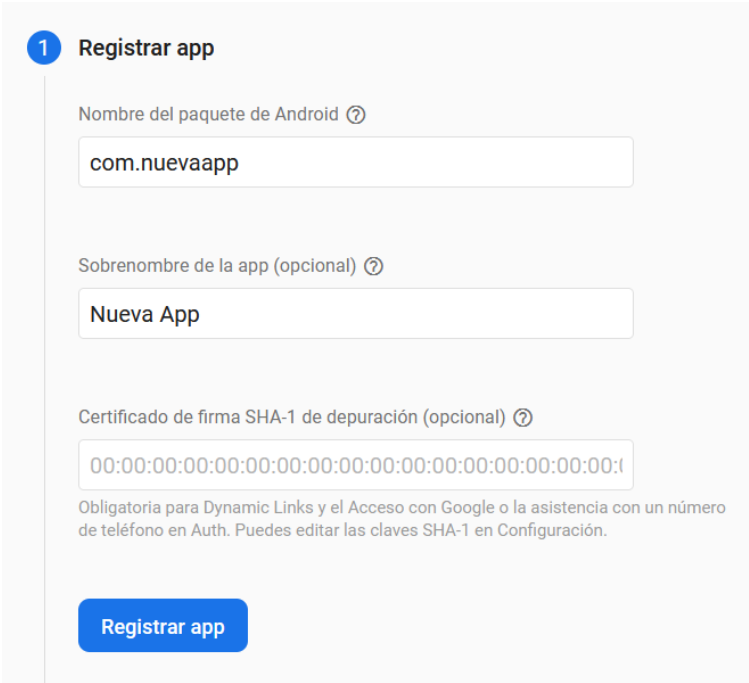


Figura 69: Dar nombre app Firebase

Al pulsar el botón Registrar App, nos aparecerá el siguiente paso, que nos indica que descarguemos un archivo de servicio:

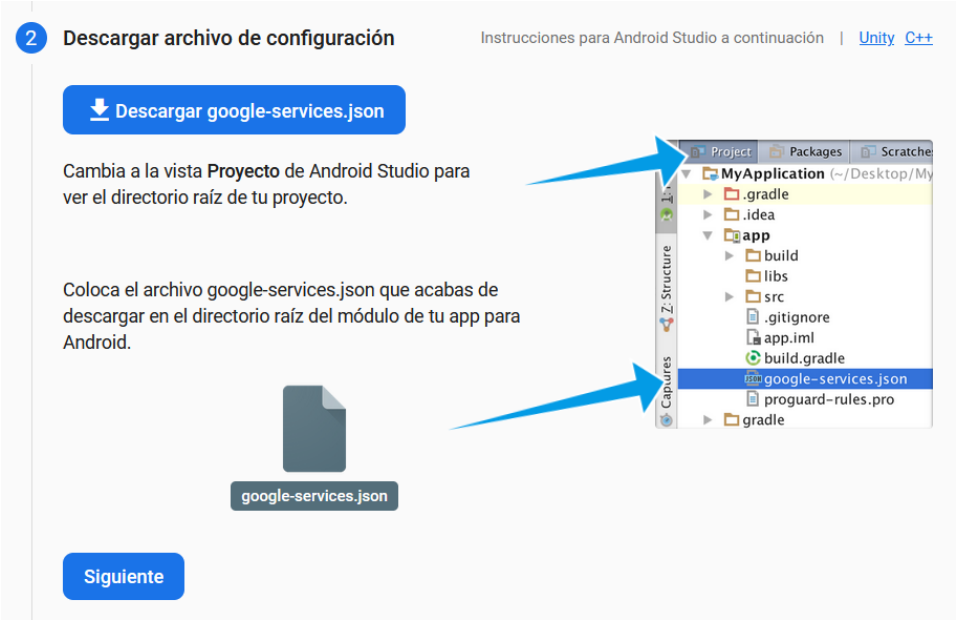


Figura 70: google-services.json Firebase

No debemos seguir las instrucciones que ahí se nos indica, ya que son los pasos que habría que seguir si nuestra aplicación estuviera desarrollada mediante Android Studio. En el caso que a nosotros nos interesa, ponemos el archivo google-services.json en la raíz del directorio de nuestra app, y en el archivo app.json, debemos añadir el campo android.googleServicesFile con el camino relativo al google-services.json, quedando así:

```
{
  "android": {
    "package": "com.nuevaapp",
    "googleServicesFile": "./google-services.json",
  }
}
```

Figura 71: app.json

Por último, para terminar de enlazar Expo con Firebase, debemos hacer click en el icono del engranaje de la app Android que acabamos de crear para nuestro proyecto de Firebase:

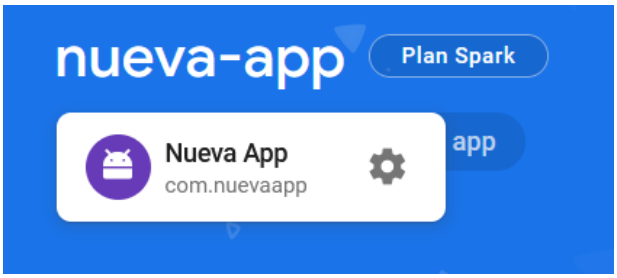


Figura 72: Configuración app Firebase

Y en la pestaña Cloud Messaging, debemos copiar el Token que nos aparece como Clave del servidor:

Figura 73: Clave del servidor Firebase

Y por último, una vez tenemos esta clave, debemos ejecutar el siguiente comando mientras estamos en el directorio de nuestra app:

```
expo push:android:upload --api-key <token>
```

Sustituyendo `<token>` por el valor de la clave que hemos obtenido, lo que hará que se suban estas credenciales a nuestra cuenta de Expo, quedando así finalmente conectados los servicios de Expo con los de Firebase.

5.5 Creación de la APK para Android

Todo el desarrollo de la aplicación puede ser probado en nuestro teléfono móvil mediante la aplicación Expo Go ya comentada anteriormente, pero el fin último es crear un archivo de aplicación que pueda ser utilizado para instalar la aplicación en nuestro dispositivo de forma autónoma.

Para ello, tendremos que ejecutar el siguiente comando mientras estemos ubicados en el directorio de la aplicación:

```
expo build:android
```

Tras lo que tendremos que seleccionar que queremos una APK, y comenzará el proceso de creación del archivo de instalación. Este proceso no se realiza de forma entera en nuestro ordenador, si no que de forma local Expo crea la build JavaScript de la app, y lo envía a los servidores de Expo que son los que se encargan de crear la apk.

En la ventana de comandos podremos seguir el progreso de este proceso, junto con un enlace en el que podremos ver cómo de ocupados se encuentran los servidores de Expo y el tiempo de espera medio para la creación de una apk, que suele tardar entre unos 15 y 20 minutos, y por último nos proporcionará un enlace con el que podremos acceder al archivo creado todo este proceso haya terminado.

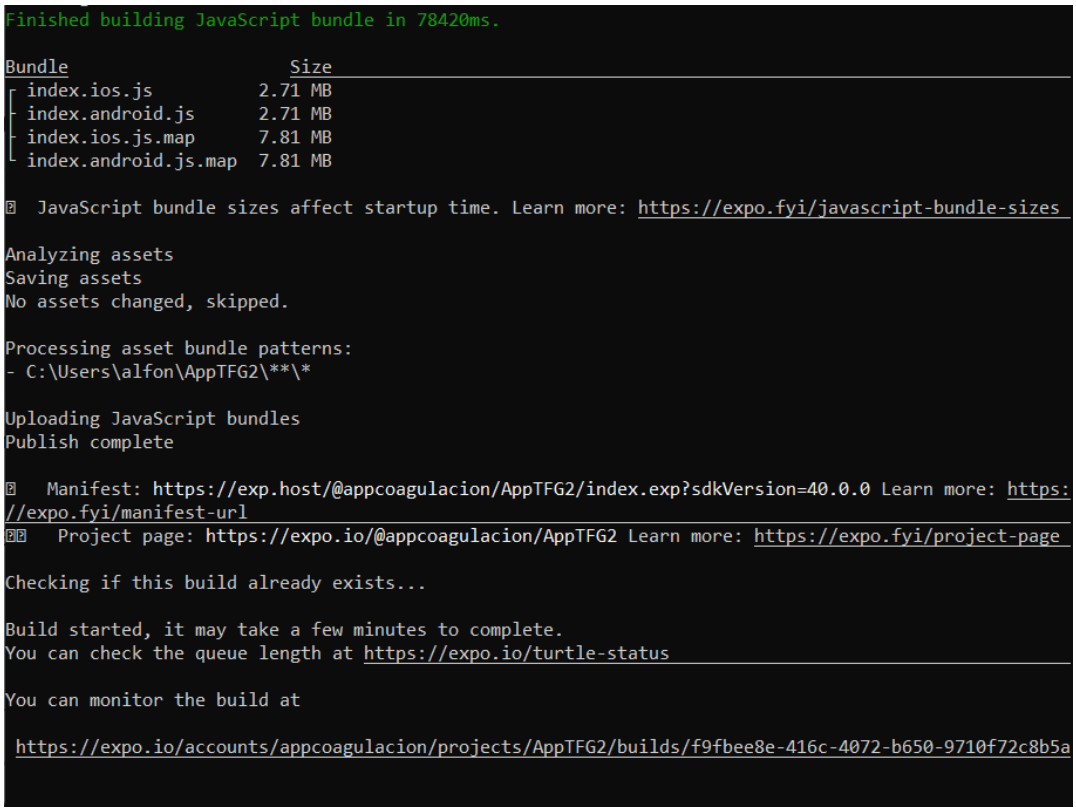


Figura 74: Build APK Android con Expo

6 CONCLUSIONES Y LÍNEA DE DESARROLLO

En este capítulo quiero poner de manifiesto cuáles han sido mis impresiones a lo largo de todo el trabajo que se ha realizado para la realización de este proyecto, así como una serie de posibles mejoras que podrían ser implementadas en un futuro.

6.1 Conclusiones

Puedo decir que la realización de este trabajo ha sido realmente provechosa, ya que creo que he aprendido mucho al trabajar con React Native, que es una herramienta que consider que tiene una curva de aprendizaje media, ya que si bien utiliza JavaScript, que es un lenguaje ampliamente conocido y con una sintaxis bastante sencilla, tiene algunas particularidades y características, como son los hooks, que hacen que comprender el funcionamiento de algunas acciones no sea tan sencillo, pero que tras un proceso de aprendizaje de estos, finalmente no suponen ninguna dificultad y hacen que el código JavaScript escrito sea más simple y entendible.

Una de las características de las que dispone React Native y que me ha parecido increiblemente útil y fácil de utilizar es la forma que tiene de crear interfaces de usuario.

En mi caso, nunca había tenido que crear una interfaz de usuario y no tengo ningún conocimiento de diseño, pero con React Native las interfaces se crean con una sintaxis simple, parecida a la de los elementos web, y que además de ser completamente personalizable, existen gran cantidad de recursos creados por la comunidad que nos facilitan en las tareas de creación de interfaces a través de diseños y layouts ya creados y muy simples de implementar en la aplicación que estamos creando.

Por otra parte, también ha sido interesante el hecho de tener que trabajar en equipo al estar dividido en cliente y servidor todo el sistema que nos habían propuesto con estos trabajos. El uso de la API REST creada en el servidor ha sido muy sencillo, ya que desde mi lado, no necesito saber cuál es la estructura de la base de datos ni cómo funciona el servidor, solamente trabajo con servicios HTTP a los que llamo cuando necesito información o cuando es el usuario el que añade información al servidor, todo a través de objetos JSON con los que son muy fáciles trabajar.

Por ultimo, el uso de Expo junto con React Native ha hecho que el desarrollo de la aplicación haya sido una experiencia más agradable, ya que gracias a Expo Go, durante todo el desarrollo he podido ver los resultados de todos los cambios que he ido introduciendo en la aplicación en tiempo real en mi teléfono móvil, sin utilizar simuladores que puden funcionar de forma menos efectiva en el ordenador, y sin necesidad de tener que crear un archivo de instalación para probar la aplicación en el dispositivo final.

6.2 Línea de desarrollo

En este apartado quiero comentar algunas posibles mejoras que pueden ser incorporadas en la aplicación en un futuro:

1. Versión para iOS: Aunque React Native es una herramienta centrada en el desarrollo multiplataforma, más concretamente Android e iOS, en este trabajo solo ha sido posible centrarme en la version de Android, ya que no dispongo de un smartphone con iOS, ni un PC con Macintosh, y aunque es posible simular Macintosh en un PC tradicional, existen elementos, como por ejemplo las notificaciones, que no funcionan en simulador, lo que ha hecho que se descartaran las pruebas en esta plataforma, aunque cabe decir que los elementos más críticos de la aplicación deben ser totalmente

compatibles con ambas versiones.

2. Chat directo entre paciente y medico: Uno de los objetivos de este trabajo es permitir un acercamiento a través de las herramientas de telemedicina entre médico y paciente, y aunque considero que se ha cumplido este objetivo, también hay lugar a más espacio de mejora, como podría ser un chat directo entre ambas partes, pero que por la complejidad que hubiese llevado a cabo hacer esto, fue descartado.
3. Cambio de la gestión de notificaciones de la app: Se han usado las herramientas que proporciona Expo para la gestión de las notificaciones en la aplicación desde el primer momento del desarrollo de esta, pero la única ventaja que nos da es una mayor simplicidad si utilizamos también su API de envío de notificaciones, pero como en nuestro caso, si se crea un servidor de envío de notificaciones personalizado, no nos ayuda en nada, por lo que creo que podría ser interesante prescindir de ella y utilizar directamente las librerías de Firebase, con la que por ejemplo, nos evitaríamos la problemática surgida con las imágenes en las notificaciones al estar abierta la aplicación.

REFERENCIAS

[1] OMS, «Telemedicine, Opportunities and developments in Member States» *artículo*, 2010. https://www.who.int/goe/publications/goe_telemedicine_2010.pdf

[2] OMS, «La OMS publica las primeras directrices sobre intervenciones de salud digital» *comunicado de prensa,* 17 de abril de 2019. https://www.who.int/es/news/item/17-04-2019-who-releases-first-guideline-on-digital-health-interventions

[3] Ángel Castellanos Rodríguez, María Sagrario Barrientos Serrano, Luis Egido Flores, Marta Torija Barrientos, Snatiago Díaz Sánchez, «Consejos para pacientes en tratamiento con anticoagulantes orales» *arículo*, 2021.

[4] Miguel Ángel Rodríguez Chamorro, Emilio García-Jiménez, Pedro Amariles, Alfonso Rodríguez Chamorro, María José Faus, «Revisión de tests de medición del cumplimiento terapéutico utilizados en la práctica clínica» *artículo*, agosto de 2008. https://www.elsevier.es/es-revista-atencion-primaria-27-articulo-revision-tests-medicion-del-cumplimiento-13125407

[5] Fundación Mozilla, JavaScript, *documentación.* https://developer.mozilla.org/es/docs/Web/JavaScript

[6] StackOverflow, 2020 Developer Survey, *encuesta*, 2020. https://insights.stackoverflow.com/survey/2020

[7] React Native, *documentación.* https://reactnative.dev/

[8] React, *documentación.* https://es.reactjs.org/

[9] React, «Presentando JSX», *documentación.* https://es.reactjs.org/docs/introducing-jsx.html

[10] React, «Componentes y propiedades», *documentación*. https://es.reactjs.org/docs/components-and-props.html

[11] React, «Presentando Hooks» *documentación*. https://es.reactjs.org/docs/hooks-intro.html

[12] React, «Presentando el Hook de efecto» *documentación*. https://es.reactjs.org/docs/hooks-effect.html

[13] React Native, «Core Components and Native Components», *documentación.*. https://reactnative.dev/docs/intro-react-native-components

[14] Paper, «Cross-platform Material Design for React Native», *documentación.*. https://callstack.github.io/react-native-paper/

[15] Flutter, *documentación.*. https://flutter.dev/

[16] Google Trends, https://trends.google.com/trends/explore?cat=31&q=flutter,React%20Native.

[17] Expo, *documentación.* https://expo.dev/

[18] Expo, *documentación.* https://expo.dev/client

[19] Roy Thomas Fielding, «Representational State Transfer» *artículo*, 2000. https://roy.gbiv.com/pubs/dissertation/rest_arch_style.htm

[20] Google, *documentación*. https://firebase.google.com/docs?hl=es

[21] Google, *documentación*. https://firebase.google.com/docs/cloud-messaging

[22] Google, «Migra de HTTP heredado a HTTP v1» *documentación*. https://firebase.google.com/docs/cloud-messaging/migrate-v1?hl=es-419

[23] React Navigation, *documentación*. https://reactnavigation.org/

[24] React, «Context» *documentación*. https://es.reactjs.org/docs/context.html

[25] React, «Referencia de los hooks» *documentación*. https://es.reactjs.org/docs/hooks-reference.html#usereducer

[26] React Native Async Storage, *documentación*. https://github.com/react-native-async-storage/async-storage

[27] Node.js, *documentación*. https://nodejs.org/es/docs/

[28] React Native Picker, *documentación*. https://github.com/react-native-picker/picker

[29] React Native Chart Kit, *documentación*. https://github.com/indiespirit/react-native-chart-kit

[30] React Native Vector Icons, *documentación*. https://github.com/oblador/react-native-vector-icons