Trabajo de Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Servicio y Aplicación Móvil de prescripción y seguimiento de ejercicio físico usando Node.js, React Native y MQTT

Autor: Manuel de la Haba Navarro

Tutor: María Teresa Ariza Gómez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

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

Servicio y Aplicación Móvil de prescripción y seguimiento de ejercicio físico usando Node.js, React Native y MQTT

Autor:
Manuel de la Haba Navarro

Tutor:
María Teresa Ariza Gómez
Profesora titular

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

Trabajo Fin de Grado: Servicio y Aplicación Móvil de prescripción y seguimiento de ejercicio físico usando Node.js, React Native y MQTT

Autor: Manuel de la Haba Navarro

Tutor: María Teresa Ariza Gómez

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 y amigos, por todo su apoyo.

Agradecimientos

Este breve apartado del trabajo, que supone el cierre a una etapa muy importante de mis estudios, me gustaría dedicarlo a quienes me han apoyado durante todo el camino.

En primer lugar, a mi familia y amigos más cercanos, sin cuyo apoyo no me encontraría hoy aquí.

En segundo lugar, a todos los compañeros que han atravesado esta etapa a mi lado.

Por último, mencionar a todos aquellos profesores que aportan calidad docente y humana al grado. Son los responsables de que, además de aprender, lo haya hecho a gusto. Aquí me gustaría destacar especialmente a Teresa, ya que con su guía he conseguido lo que están a punto de leer en este documento.

Como mención especial, dejar constancia de sitios web donde se forma a todos los programadores, como Stack Overflow, que de forma anónima y desinteresada se presta ayuda en mil y una dudas o problemas que surgen en el día a día mientras se programa.

Manuel de la Haba Navarro

Sevilla, 2021

Resumen

La presencia de Internet en prácticamente todo ha ido haciéndose cada vez más y más notoria los últimos años. El uso de teléfonos inteligentes ha acentuado en gran medida este hecho, ya que supone uno de los dispositivos más versátiles de uso diario. Con ellos se puede realizar desde las tareas más básicas, como hacer la lista de la compra o anotar una cita en el calendario, hasta gestionar una cuenta bancaria. El hecho de que prácticamente todo el mundo los utilice, motiva cada vez más a ofrecer una gama más amplia de servicios. Todo esto se consigue por medio de aplicaciones móviles, o abreviado, 'apps'.

El desarrollo de estos servicios suele requerir tecnologías y procedimientos diferentes, que dependen en gran medida del sistema operativo del dispositivo móvil en cuestión.

Además, estos servicios se ven apoyados fuertemente por servidores alojados en Internet, encargados de coordinarlos.

El hecho de que la gama de tecnologías disponibles para desarrollo vaya en crecimiento, al existir formas cada vez más precisas y adecuadas a cada tipo de servicio, dificulta en gran medida el desarrollo de soluciones complejas y/o compuestas por diferentes componentes.

Pero, así como ha crecido la gama de posibilidades para desarrollar, también ha aparecido como corriente opuesta el atractivo de lo sencillo. Esto supone que ahora existen entornos que facilitan el desarrollo de servicios multiplataforma, independientes del sistema operativo, sin tener que distinguir entre lenguajes de programación de unos y de otros.

Si además este lenguaje se puede emplear en entornos para desarrollar el servidor alojado en Internet, resultará más sencillo crear el servicio completo a nivel de trabajo y de uso de recursos.

En la propuesta que recoge este documento, se planteará esta idea: la posibilidad de, salvo la base de datos, tener todo el servicio programado en un único lenguaje, JavaScript. En la parte del servidor se utilizará el entorno Node.js, sobre el que se apoya Express, una infraestructura de servidor web pensada para éste. En la parte del cliente se utilizará React Native, un entorno pensado para desarrollar aplicaciones móviles multiplataforma. Esta tecnología se utilizará bajo el uso de Expo, una plataforma de desarrollo pensada específicamente para ella.

Abstract

Internet's presence in many aspects of our daily life has been becoming more and more noticeable in recent years. This has been greatly increased by the use of smartphones, since it is one of the most versatile devices for daily use. Making use of them you can perform the most basic tasks, such as making a shopping list or writing an appointment on the calendar, to managing a bank account, for example.

The fact that almost everyone uses a smartphone, increasingly motivates to offer a wider services range. This is achieved through mobile applications, or "apps". The development of these services often requires different technologies and procedures, which are highly dependant in most cases on the device operating system.

In addition, these services are frequently supported by Internet hosted servers, which are responsible for coordinating them.

Furthermore, the fact that the available development technologies range is growing, as there are increasingly more precise and appropriate ways to create each kind of service, greatly hinders the development of complex solutions, and/or made up of multiple components.

Though just as the range of possibilities for development has grown, the appeal of simplicity has also emerged as a counter current. This means that, nowadays, there are frameworks that allow the development of multi-platform services with ease, regardless of the operating system, without having to distinguish between one programming language or another.

In addition, if this language can also be used with frameworks to develop the server, it will be easier in terms of work and resources to create the complete service.

The proposal included on this document will raise this idea: the possibility of having the entire service excluding the database, programmed using a single language, JavaScript. On the server side, the Node.js framework will be used, on which Express, a web server infrastructure designed for it, relies. On the client side, React Native will be used. This is a framework designed to develop multiplatform mobile applications. It will be used under Expo, a development platform designed specifically for it.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Si tu negocio no está en Internet, tu negocio no existe.

- Bill Gates -

La última década ha sido una época crucial en lo que a dispositivos móviles se refiere. El desarrollo de las tecnologías siguiendo una curva casi exponencial, ligado al aumento del uso de estos dispositivos, ha fomentado que surja una amplia gama de servicios que ofrecer a través de ellos. Ya no únicamente los llamados teléfonos inteligentes o 'smartphones', sino tabletas, televisores, coches u otros dispositivos susceptibles de utilizar ciertos servicios.

A lo largo de este punto, me gustaría detallar el contexto bajo el que se ha ideado la propuesta, las motivaciones, objetivos y aportar una vista esquemática del documento.

1.1 Motivación

Según estudios publicados por el Instituto Nacional de Estadística [1], a lo largo de la última década, la presencia del ejercicio físico en el día a día de la población en nuestro país ha ido haciéndose más notoria. Sin embargo, aún se encuentra lejos de lo que pudiera ser deseable para poder llevar una vida sana según recomienda la Organización Mundial de la Salud [2].

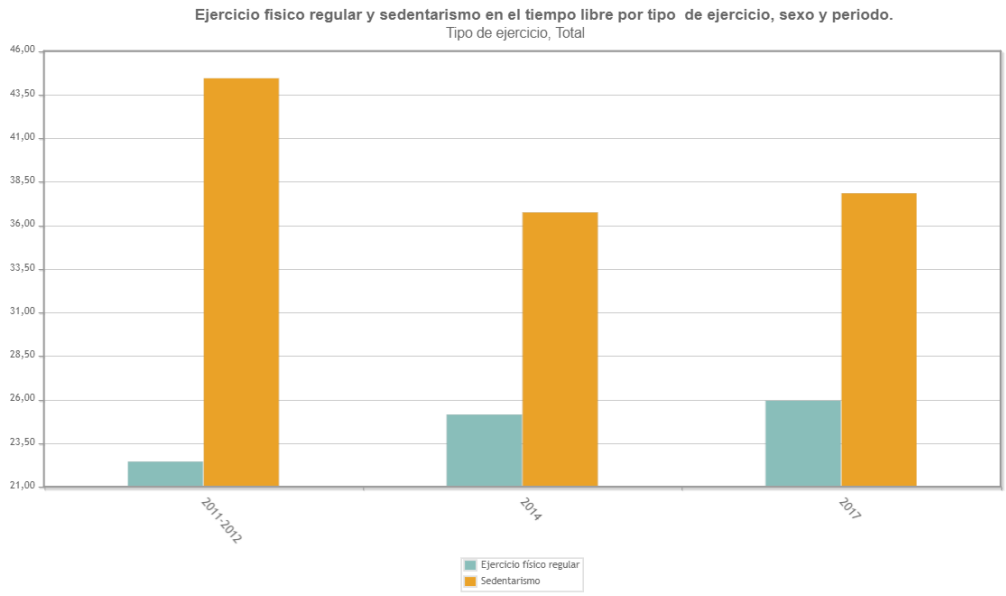

Ilustración 1 - Evolución de la actividad física en España (2011-2017). INE

Además, en los dos últimos años, las circunstancias especiales provocadas por la epidemia de Covid-19 no han ayudado precisamente a mejorar estos datos, ya que un 44% de los ciudadanos de nuestro país han aumentado de peso durante el confinamiento [3]. Esto, si bien un motivo es la peor alimentación, se debe en gran medida al aumento del sedentarismo.

En este contexto surge la idea de compaginar el realizar ejercicio físico, con las recomendaciones y la supervisión de expertos en la materia, con poder hacerlo en un entorno seguro y flexibilizando el horario, facilitando que el usuario adapte el ejercicio físico a su tiempo libre en el día a día.

Otro aspecto importante en la motivación para desarrollar este proyecto es el Trabajo de Fin de Grado realizado por Pablo Carmona Rebollo [4]: '*Plataforma web de prescripción de ejercicios usando Spring'*, en 2018, tutorizado por la Profesora María Teresa Ariza Gómez. No obstante, en esta ocasión se aportará un enfoque distinto, tanto en las tecnologías de desarrollo utilizadas, como en las funcionalidades del resultado final. Sin embargo, conserva bastantes similitudes con el trabajo mencionado, sobre todo en la idea del servicio y en la estructura de la base de datos.

Las principales diferencias en las funcionalidades consistirán en la interacción a través del servicio entre un Especialista y un Usuario en la realización de ejercicio físico.

1.2 Objetivos

A lo largo del desarrollo del proyecto, los objetivos a conseguir han ido sufriendo modificaciones, con supresión de algunos, por su inviabilidad o bien porque tras avanzar en el desarrollo no eran apropiados, o la incorporación de otros nuevos. En este apartado se comentarán los objetivos principales en el desarrollo de este trabajo.

Cabe destacar que algunos de los objetivos son fruto del cambio de enfoque aplicado a la propuesta desarrollada en el Trabajo de Fin de Grado de Pablo Carmona, mencionado en el subapartado anterior.

Los principales objetivos a conseguir durante el desarrollo del trabajo son los siguientes:

- Lenguaje de programación único: Uso de tecnologías en el servidor y en el cliente que utilicen el mismo lenguaje de programación, JavaScript.
- Aprovechar la versatilidad de los dispositivos móviles: Al ser dispositivos que se pueden llevar a

cualquier parte, y estar provistos cada vez de más sensores -o permiten conectar pulseras de actividad u otros 'wearables' que los incorporan-, son la opción ideal para aplicaciones de esta naturaleza.

- Seguimiento de actividad física en tiempo real: Intercambio de información entre Especialista y Usuario.
- Optimización del tráfico que maneja el servidor: En un entorno de producción, con un volumen de usuarios elevado, el servidor debe poder satisfacer todas las peticiones de los usuarios del servicio. Por ello, es interesante desviar todas aquellas comunicaciones que no lo impliquen directamente.

En el último punto de la memoria se volverá sobre estos objetivos a la hora de hablar de las conclusiones finales sobre el proyecto.

1.3 Descripción de la propuesta

La propuesta, bautizada como *RemGym* (Remote Gym), consta de tres componentes diferenciados -base de datos, aplicación web y aplicación móvil-, desarrollados y desplegados como se comentará en detalle en este documento, y un cuarto componente que consiste en una aplicación de terceros.

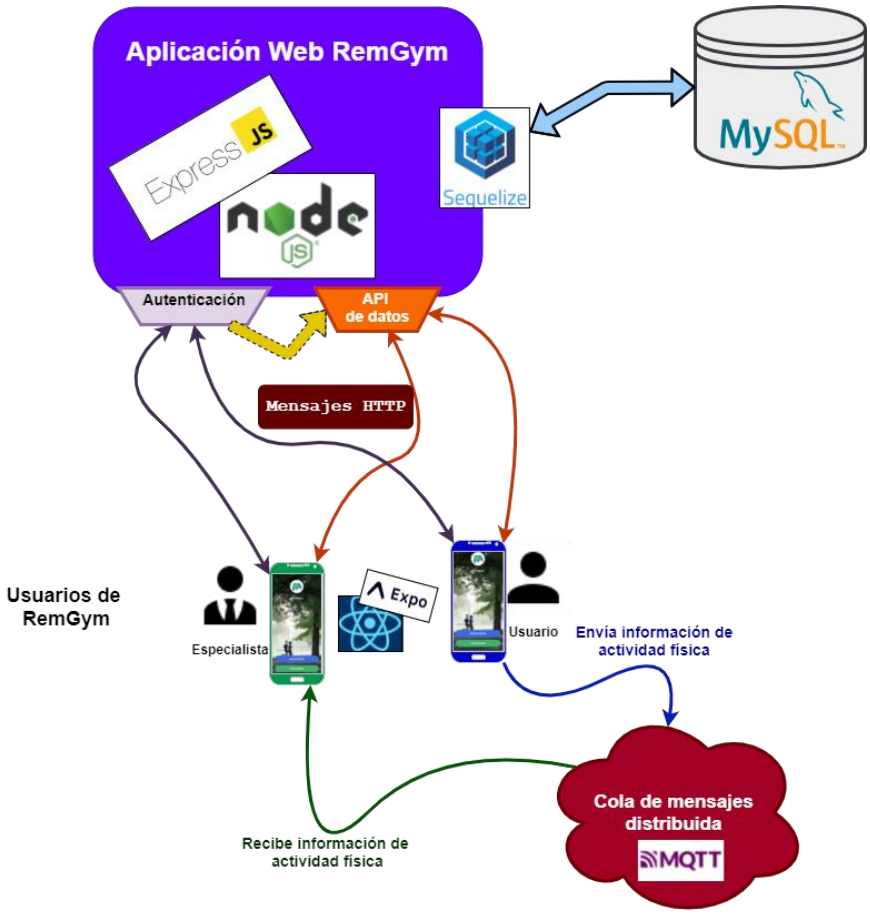

Ilustración 2 – Esquema básico del servicio desarrollado

Como se puede apreciar en la Ilustración 2, la parte de la aplicación web incluye una API REST[1] [5] [6].

[1] Una API (Applications Programming Interface) en español, Interfaz de Programación de Aplicaciones, es un conjunto de protocolos y definiciones usadas para integrar el software de las aplicaciones, permitiendo la comunicación entre dos o más, y estableciendo las reglas y

Mediante esta API, los usuarios de la aplicación móvil intercambiarán información, de forma controlada y restringida, con la base de datos, empleando mensajes HTTP[2] [7]. Además, en el servidor se distingue otra API por separado de la API REST de datos, la API de Autenticación. El motivo de esto es que, aunque ambas sean APIs REST, la de Autenticación hará uso de la de Datos.

Por otra parte, tenemos la aplicación móvil como interfaz de usuario, y constituye toda la parte del proyecto que verán los usuarios finales del servicio. Según el planteamiento de la propuesta, en el uso de la aplicación se distinguen dos roles de usuario diferenciados: Especialista y Usuario. Esto permitirá diferenciar las funcionalidades ofrecidas en la interfaz de usuario.

Por último, mencionar la aplicación de terceros que utilizará la aplicación. Esta no es más que el bróker MQTT, encargado de gestionar la cola de mensajes que utilizará la app para el seguimiento de la actividad física. Se verá en detalle en el capítulo siguiente.

1.4 Estructura de la memoria

La memoria tendrá una estructura por puntos, donde se aportarán diferentes puntos de vista del proyecto:

- Introducción: Contexto, motivación, objetivos y vista general del proyecto.
- Tecnologías utilizadas: Entornos de desarrollo, bibliotecas, etc. utilizados en el proyecto.
- Herramientas y recursos utilizados: Aplicaciones, 'frameworks', herramientas, etc. que se han utilizado para el desarrollo del proyecto.
- Arquitectura y análisis: Organización del código del proyecto, diferenciando las partes que lo componen, así como la interacción entre ellas.
- Interfaz de usuario: Vistas de la aplicación para dispositivos móviles, con las que el usuario interactuará en el uso del servicio.
- Pruebas: Relación de pruebas que se han realizado para garantizar el funcionamiento individual y colectivo de cada uno de los elementos del sistema.
- Líneas de mejora y conclusiones: Aspectos del servicio a mejorar, nuevas funcionalidades que se podrían implementar, y conclusiones alcanzadas tras realizar el proyecto.

procedimientos a seguir para dicha comunicación. Una API REST (Representational State Transfer), en español, Transferencia de Estado Representacional, es un conjunto de reglas de arquitectura software para servicios distribuidos.

[2] HTTP son las siglas de HyperText Transfer Protocol, en español, Protocolo de Transferencia de HiperTexto. Es un protocolo de comunicación que emplea mensajes en formato de texto plano, utilizado ampliamente en las comunicaciones con aplicaciones web. Fue desarrollado por la Internet Engineering Task Force (IETF) y el World Wide Web Consortium (W3C).

2 TECNOLOGÍAS UTILIZADAS

Especialmente en tecnología, necesitamos cambios revolucionarios, no cambios incrementales.

- Larry Page -

En este punto se van a detallar todas las tecnologías que se han utilizado en el desarrollo del proyecto. Éstas se han elegido bien por su simplicidad, bien para satisfacer ciertos objetivos detallados en el punto anterior, como pueda ser el hecho de emplear en todos los aspectos posibles del desarrollo el mismo lenguaje de programación.

Un aspecto deseable y en muchos casos determinante que ha sido tenido en cuenta para elegir ciertas tecnologías ha sido que fueran de código abierto, u '*open source'*.

2.1 MySQL Database

Ilustración 3 - Logo de MySQL

MySQL [8] es un sistema gestor de bases de datos -en adelante SGBD- relacional. De código abierto en su vertiente de licencia GPL, es el SGBD más utilizado en el mundo.

Se utiliza empleando un dialecto SQL del mismo nombre, 'MySQL', con sutiles diferencias respecto al lenguaje base.

2.2 JavaScript

Ilustración 4 – Logo de JavaScript

JavaScript [9] (normalmente abreviado como JS) es un lenguaje de programación interpretado, utilizado muy frecuentemente en programación web dinámica en el cliente, aunque también se utiliza en el servidor en entornos como Node.js.

Es un dialecto del estándar ECMAScript, regulado por ECMA[3]. Tiene soporte para programación orientada a objetos, y soporta programación asíncrona, que se verá en el siguiente subapartado.

2.2.1 Uso de programación asíncrona

Una de las grandes ventajas de JavaScript, que además aumenta su rendimiento, es la posibilidad de usar código asíncrono [10], frente al uso síncrono tradicional del lenguaje.

JavaScript síncrono planteaba el problema de que, al solicitar información de un dispositivo externo, ya sea una API en la red, una base de datos, etc., el código debía esperar para acceder a esos recursos, ya que, al ser un lenguaje monohilo, el programador no podía realizar tareas adicionales mientras se esperaba a la obtención del recurso.

Por ello, si una línea de código JavaScript solicita un recurso de forma síncrona, el estado del programa no puede avanzar hasta que se termina de obtener dicho recurso y, en caso de una página web por ejemplo, se detendría el renderizado. Esto conllevaría una mala experiencia de usuario.

Aquí es donde aparece JavaScript asíncrono, permitiendo mediante múltiples mecanismos ejecutar código en paralelo, y permitiendo esperar el resultado de la ejecución de algunos bloques de código para ejecutar otros. A continuación, se nombran algunos de esos mecanismos, utilizados en el proyecto:

2.2.1.1 Callbacks asíncronos

Aunque su uso está en recesión, los callbacks asíncronos aún se utilizan en multitud de APIs. Este mecanismo plantea la posibilidad de ejecutar en segundo plano las operaciones de entrada/salida que por naturaleza incluyen retardo, y una vez finalizan u ocurre algún tipo de evento lanzar una función, llamada callback, mediante la cual sincronizar el comportamiento de varias partes del código.

Como se ha comentado, su uso está en recesión. No obstante, en el desarrollo del proyecto se utiliza en algunas ocasiones, bien por comodidad, bien por ser la opción más apropiada.

2.2.1.2 Promesas

Como alternativa a los callbacks asíncronos existe el concepto de promesa. Una promesa es un objeto en JavaScript, denominado *Promise*, que representa el resultado de una función que aún no ha sido resuelta. Por ello, el resultado que arrojará eventualmente la promesa no es conocido en el instante de ser creada. Este

[3] ECMA son las siglas de European Computer Manufacturers Association, en español: Asociación Europea de Fabricantes de Computadores.

resultado puede ser un resultado esperado de la función, o bien un error por algún motivo.

Cuando el resultado de la promesa es el funcionamiento esperado, se dice que la promesa ha sido *resuelta (resolved)*. En caso de que hubiera algún error se dice que ha sido *rechazada* (*rejected*).

A continuación, se muestran dos formas de gestionar las promesas en JavaScript:

2.2.1.2.1 Uso de async/await

Un método de gestionar las promesas es el uso de las palabras reservadas *async* y *await* [11]. La palabra *async* se ubica delante de una función cuyo comportamiento se desea que sea asíncrono, de manera que algunas de las funciones son asíncronas por definición. El resultado de la ejecución de una función asíncrona es un objeto Promise.

La otra palabra, *await*, se coloca delante de las llamadas a funciones asíncronas, tales como solicitudes de recursos externos. Sólo se puede utilizar en funciones marcadas con *async*. El funcionamiento de esta palabra reservada consiste en detener la ejecución de la función asíncrona padre hasta que se ha resuelto la ejecución de la función marcada con esta palabra y que ya ha sido llamada, ya que, sin ella, el código seguiría ejecutando línea tras línea, sin esperar al resultado de dicha función.

En la Ilustración 5 se puede ver un pseudocódigo de ejemplo.

```
const myFunction = async () => {
    const response = await funcion_asincrona();
    // La función no imprimirá response hasta que la función asíncrona
    // haya finalizado su ejecución.
    console.log(response);
};
```

Ilustración 5 – Ejemplo de código usando async/await

2.2.1.2.2 Uso del método .then() del objeto Promise

La otra opción de gestión de promesas es el uso de funciones pasadas al método .then(), perteneciente a los objetos de tipo Promise. De esta manera, para ejecutar un bloque de código que utilice el resultado de una función asíncrona, se podrá pasar como función al método .then(), garantizando que ese resultado estará disponible.

Este método acepta hasta dos parámetros: El primero y obligatorio, es un callback que se ejecutará cuando la promesa ha sido resuelta. El segundo, opcional, es un callback a ejecutar si la promesa es rechazada.

En la Ilustración 6 se puede ver un pseudocódigo que ejemplifica el uso de .then().

```
const myFunction = () => {
    funcion_asincrona()
    .then(response => {
        // Este bloque de código no se ejecutará
        // hasta que la función asíncrona haya terminado.
        console.log(response);
    });
};
```

Ilustración 6 – Ejemplo de código usando el método .then()

2.2.1.2.3 Manejo de errores

Antes se ha comentado que las promesas devuelven el resultado de la ejecución de código asíncrono, o bien un error, si lo ha habido durante dicha ejecución. Pues tanto el uso de async/await como el uso del método *then* disponen de mecanismos de manejo de errores:

- El uso de async/await se vale de la forma tradicional de manejo de errores, el uso de los conocidos bloques try/catch. En la Ilustración 7 se muestra un ejemplo de esto:

```
const myFunction = async () => {
    try{
        const response = await funcion_asincrona();
        // La función no imprimirá response hasta que la función asíncrona
        // haya finalizado su ejecución.
        console.log(response);
    } catch(e) {
        console.log(e);
    }
};
```

Ilustración 7 – Ejemplo de código async/await con manejo de errores

- Las promesas admiten, además del segundo parámetro del método .then(), el uso de un método llamado .catch(), ubicado después del método .then(), para gestionar los errores durante la resolución de una promesa. En el código de la Ilustración 8 se muestra su uso:

```
const myFunction = () => {
    funcion_asincrona()
    .then(response => {
        // Este bloque de código no se ejecutará
        // hasta que la función asíncrona haya terminado.
        console.log(response);
    })
    .catch(error => {
        console.log(error);
    })
};
```

Ilustración 8 – Ejemplo de código usando el método .then() con manejo de errores

La gran ventaja de las promesas es que se pueden anidar, encadenar y agrupar, algo mucho más gestionable que el uso de callbacks sin más.

2.3 React Native – Expo

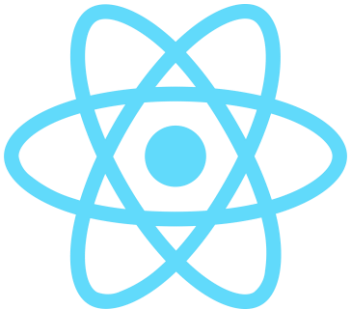

Ilustración 9 – Logo de React Native

Antes de comentar React Native, es necesario hacer unos apuntes sobre React [12]. React es una biblioteca de JavaScript utilizada para construir interfaces de usuario. Aporta sencillez en el desarrollo de interfaces de usuario interactivas. Está pensado para codificar las interfaces de usuario divididas en componentes diferenciados, que manejan su propio estado e interactúan con otros componentes para escalar la complejidad.

React cuenta con la ventaja de poder usarse en el cliente, como comúnmente se emplea JavaScript -enmarcado en el DOM-, o bien en el servidor usando Node.js.

2.3.1 Descripción y características generales

React Native [13] es un entorno de desarrollo de aplicaciones móviles multiplatafoma. Teniendo en cuenta lo comentado acerca de React, React Native es prácticamente lo mismo, pero emplea componentes ‘nativos’ en lugar de componentes ‘web’. Estos componentes ‘nativos’ se traducirán a su equivalente en el código

correspondiente al sistema operativo cuando el proyecto se exporte a una aplicación. Como características generales, comentaremos las que hereda de React:

2.3.1.1 Componentes

Los componentes son la unidad estructural de cualquier interfaz de usuario. Pueden ser más o menos básicos, y estar compuestos de otros componentes a su vez. Se suelen definir con funciones que en su sentencia 'return' devuelven un conjunto de elementos encerrados en uno que los engloba todos, como se muestra en la Ilustración 10:

```
<Text style={[defaultStyles.text, style]} {...otherProps}>
    {children}
</Text>
```

Ilustración 10 – Fragmento de código. Componentes de React

Estos componentes aceptan una serie de parámetros que modifican el aspecto o el comportamiento, dependiendo de su naturaleza.

2.3.1.2 Parámetros de creación, o 'props'

La mayoría de componentes admiten modificación de algunos parámetros una vez son creados. Estos parámetros que establecen las modificaciones se llaman parámetros de creación, o 'props'.

Para establecerlos a la hora de codificar un componente, la función que lo define admitirá un único parámetro llamado 'props', que será un objeto que contenga todos los parámetros a usar.

Otra opción de uso, la más común empleada en el desarrollo de este proyecto, es deconstruir el objeto 'props', estableciendo directamente el contenido del objeto como se puede apreciar en la Ilustración 11:

```
function AppText({children, style, ...otherProps}) {
```

Ilustración 11 – Fragmento de código. Uso de props

2.3.1.3 Estado, o 'state'

Al igual que los parámetros de creación, el estado permite modificar el resultado del componente en la interfaz de usuario, con la diferencia de que no es 'read-only', y que estas modificaciones pueden ser en tiempo de ejecución, y no al crear el componente.

El estado se regulará mediante variables definidas empleando el 'hook'[4] [14] useState, de la forma indicada en la Ilustración 12:

```
const [expanded, setExpanded] = useState(false);
```

Ilustración 12 – Fragmento de código. Uso de hooks

Al declarar variables de estado de esta forma, obtenemos la variable, cuyo valor es accesible a través de 'expanded' en este ejemplo, y una función para modificarlo, 'setExpanded' en este ejemplo.

[4] Los hooks son una forma de controlar el estado de los componentes en React o React Native sin utilizar clases, forma utilizada hasta su aparición. Además su incorporación solucionó otra gran cantidad de problemas que había en las versiones anteriores a su lanzamiento.

2.3.2 Sintaxis y lenguaje

React Native se puede utilizar empleando JavaScript o TypeScript. TypeScript es un lenguaje basado en JavaScript, con la diferencia de que es más tipado. Esto facilita el desarrollo y depurado del código.

2.3.3 Módulos de React Native utilizados

A continuación, se comentan algunos de los módulos más importantes de React Native que se han empleado. Para ver el total de módulos utilizados en la aplicación, se encuentran recogidos en el fichero 'package.json', junto con su versión.

2.3.3.1 Async-storage

Async Storage [15], que antes era un módulo interno de React Native, consiste ahora en un módulo desarrollado por la comunidad, que permite utilizar en React Native un sistema de almacenamiento que emplea el método clave-valor, caracterizado por ser asíncrono y no encriptado.

2.3.3.2 Navigation

React Navigation [16] es un módulo que permite la navegación entre distintas vistas de las aplicaciones desarrolladas en React y en React Native. De este módulo se emplean múltiples submódulos, como Native, Bottom-Tabs, o Stack.

2.3.3.3 Webview

Webview [17], que al igual que ocurre con async-storage, antes era un módulo interno de React Native, ahora es un módulo desarrollado por la comunidad que permite implementar en las aplicaciones el componente 'WebView'.

2.3.4 Expo

Ilustración 13 – Logo de Expo

Expo [18] es una Plataforma para aplicaciones React. Consiste en un conjunto de servicios y herramientas en torno a React Native, que facilitan el desarrollo, compilación, depuración y despliegue de las aplicaciones.

La principal ventaja a la hora de desarrollar aplicaciones es que conforme se va codificando, se pueden ir haciendo pruebas en tiempo real, tanto en navegador web, simulador Android, como dipositivos móviles físicos, tanto Android como iOS, mediante la aplicación Expo Go.

Esta aplicación se encuentra disponible en las tiendas de aplicaciones oficiales de Android (Play Store) y de

iOS (App Store).

2.3.5 Proyectos en Expo. Bare workflow vs Managed workflow

A la hora de desarrollar aplicaciones móviles utilizando Expo, la plataforma ofrece dos plantillas de partida [19], que condicionarán todo el mecanismo de desarrollo, desde la estructura del mismo hasta los lenguajes de programación a emplear.

Antes de nada, es importante destacar el hecho de que los proyectos de desarrollo de aplicaciones móviles por defecto emplean un lenguaje de programación distinto en función del sistema operativo de destino: Java en Android, mediante Android Studio, y Objective-C y/o Swift en iOS.

Volviendo a Expo y sus dos formatos de proyecto, la elección del mismo para desarrollar la aplicación puede depender del alcance que ésta deba tener. Estos formatos son:

- Bare workflow: Ofrece control total sobre el proyecto en React Native, limitando el alcance de Expo. No obstante, esto repercute en que algunas partes del proyecto no podrán ser programadas con JavaScript, sino que se tendrán que codificar empleando el lenguaje que proceda según el sistema operativo de destino.
- Managed workflow: Permite a Expo abstraer todo el contenido del proyecto, por lo que el desarrollador sólo tendrá que programar empleando JavaScript o TypeScript.

En este proyecto se ha empleado esta última opción, para satisfacer ciertos objetivos marcados en el punto 1.2 del documento. Aunque esto ha repercutido en algunos problemas surgidos durante el desarrollo, comentados en el punto 7.6, donde se detallarán las conclusiones extraidas del proyecto en base a los objetivos.

2.3.6 Módulos de Expo utilizados

A continuación, se muestran los módulos más importantes de Expo empleados en la aplicación. Para ver el total de módulos utilizados, están disponibles en el fichero 'package.json' del código de la aplicación, junto con la versión del módulo que se ha utilizado.

2.3.6.1 Vector-icons

El módulo vector-icons [20] consiste en una capa de compatibilidad del módulo react-native-vector-icons para ser usado bajo Expo. Por defecto se instala al crear de cero una aplicación.

2.3.6.2 Constants

El módulo expo-constants [21], que permite acceder a información acerca de la aplicación que permanece constante en el tiempo, se empleará para acceder a algunos parámetros, como 'platform' (conocer si la aplicación se está ejecutando en Android o iOS), o 'statusBarHeight' (conocer el tamaño de la barra de estado en la parte superior de la pantalla).

2.3.6.3 Location

El módulo expo-location [22] permite leer la geolocalización del dispositivo que ejecuta la aplicación. Implementa métodos para suscribirse a actualizaciones de ubicación periódicas u obtener información de forma asíncrona de la ubicación actual.

2.3.6.4 Secure-store

El módulo secure-store [23] ofrece la posibilidad de encriptar y almacenar información en formato clave-valor de forma local en el dispositivo.

2.3.6.5 Task-Manager

TaskManager [24] facilita una API que permite gestionar tareas que se deben ejecutar a lo largo del tiempo, también en segundo plano. En la aplicación se empleará de forma coordinada con Expo-Location.

2.3.7 Otros módulos externos

Por último, se comentarán algunos módulos externos a React Native o Expo empleados en la aplicación.

2.3.7.1 Apisauce

El módulo apisauce [25] permite consumir APIs HTTP. Se ha utilizado en la aplicación para comunicarse con el servidor.

2.3.7.2 Formik

Formik [26] es una biblioteca de código abierto que ofrece todos los elementos necesarios para construir formularios en aplicaciones React y React Native.

2.3.7.3 JWT-decode

El módulo jwt-decode [27] permite decodificar tokens JWT (tecnología explicada más adelante) que utilizan codificación Base64.

2.3.7.4 React Native MQTT

El módulo react-native-mqtt [28] ofrece una implementación en JavaScript de la biblioteca Eclipse Paho MQTT para poder utilizarla en aplicaciones React Native. Está pensado para conexiones utilizando sockets web.

2.3.7.5 Yup

Yup [29] es un constructor de esquemas para validar formatos y valores de un conjunto de elementos. Se emplea en la validación de los formularios previa a su envío al servidor.

2.4 Node.js – Express.js

Este subapartado tratará de la tecnología utilizada para implementar el servidor que actuará de intermediario entre la interfaz de usuario y la base de datos, entre otras cosas. Consiste en un servidor Node.js que utiliza una infraestructura web llamada Express.js para desplegar el servicio.

2.4.1 Descripción y características generales

A continuación, se detallan por separado tanto Node.js como Express.js.

2.4.1.1 Node.js

Ilustración 14 – Logo de Node.js

Node.js [30] es un entorno de ejecución, o servidor, construido con el motor de JavaScript V8 de Google Chrome, para desarrollar aplicaciones en JavaScript.

Su principal ventaja es que a la hora de gestionar peticiones el proceso no se bloquea, por lo que se pueden atender multitud de peticiones sin tener que recurrir a los hilos del sistema operativo. Por este motivo, supone una opción muy a tener en cuenta a la hora de desarrollar sistemas escalables.

Como apunte, destacar que tiene muchas similitudes con EventMachine de Ruby o Twisted de Python.

2.4.1.2 Express.js

Ilustración 15 – Logo de Express.js

Express.js [31] es una infraestructura para desarrollar aplicaciones web de manera sencilla y rápida sobre Node.js.

Desarrollado como un proyecto de la Fundación OpenJS, a diferencia de otras infraestructuras, no hay una forma concreta de desarrollar un servicio. No existe una manera 'correcta' de proceder, sino que deja todos los aspectos que sí puedan estar más restringidos en el desarrollo a bibliotecas de terceros, como puede ser el uso de base de datos o la autenticación.

Como aspecto interesante a destacar, Express soporta motores de plantillas para el renderizado en el servidor de contenido web enviado al cliente, como ocurre por ejemplo en infraestructuras en Python como Flask o Django, que usan un motor de plantillas llamado Jinja.

2.4.2 Sintaxis y lenguaje

Al igual que se ha comentado en el punto de React Native, el lenguaje utilizado en el desarrollo de

aplicaciones con estas tecnologías es JavaScript.

2.4.3 Módulos utilizados

Además de los principales módulos de cada tecnología, a continuación, se mencionan algunos de los módulos más importantes que se han utilizado para dar forma al servidor:

2.4.3.1 Bcrypt

Este módulo de bcrypt para Node.js [32] hace uso de la tecnología bcrypt, explicada más adelante en el punto 2.7. Permite encriptar contraseñas, para no almacenarlas en formato de texto plano en la base de datos. Además, proporciona métodos para comparar la contraseña recibida en una solicitud de autenticación con la ya encriptada en la base de datos.

2.4.3.2 Dotenv

Ilustración 16 – Logo de dotenv

El módulo dotenv [33] permite cargar variables de entorno definidas en un fichero con extensión '.env' en la aplicación.

2.4.3.3 JsonWebToken

El módulo JsonWebToken para Node.js [34] ofrece una implementación de la tecnología JSON Web Token (explicada en un subapartado posterior) para servidores Node.js.

2.4.3.4 MySQL para Node.js

El módulo node-mysql2 [35] consiste en un cliente MySQL para servidores Node.js. La tecnología MySQL ha sido explicada en un subapartado anterior. Permite establecer conexiones con servidores MySQL para gestionarlos, o realizar consultas.

2.4.3.5 Nodemon

Ilustración 17 – Logo de nodemon

Nodemon [36], utilizado para depuración durante el desarrollo, permite rearrancar automáticamente el servidor Node.js cada vez que se guarda una modificación en alguno de sus ficheros.

Para ello, en lugar de lanzar el servidor ejecutando el comando `node` directamente, se hará mediante el ejecutable de Nodemon. Esto supone un ahorro considerable de tiempo y resulta muy cómodo durante las fases de codificación y pruebas del servidor.

2.4.3.6 Sequelize

Ilustración 18 – Logo de Sequelize

Sequelize [37] es un ORM[5], basado en el concepto de promesa en JavaScript, que soporta mútiples dialectos del lenguaje SQL: Postgres, MySQL, MariaDB, SQLite, entre otros. Permite utilizar una base de datos de forma más segura que una simple conexión con ejecución de sentencias en servidores Node.js, evitando así ataques como SQL Injection.

2.4.3.7 Sequelize-auto

El módulo sequelize-auto [38], empleado en el desarrollo del servidor, automatiza la creación de los modelos de la base de datos para poder ser utilizados por Sequelize. Tan sólo requiere conexión a la base de datos, y genera los ficheros de código necesarios al instante.

[5] ORM son las siglas de Object-Relational Mapping. Permite transformar las tablas de la base de datos junto a sus relaciones en entidades en el servidor, para facilitar la comunicación, y la codificación sobre ésta. Se verá más adelante.

2.5 MQTT

Ilustración 19 – Logo de MQTT

Como se ha mencionado anteriormente en el documento. La idea de utilizar una cola de mensajes distribuida surge a raíz de querer librar al servidor de tráfico entre usuarios que no requieran de una gestión explícita por el servidor. En este marco entra la tecnología MQTT [39].

2.5.1 Descripción y características generales

MQTT son las siglas de Message Queuing Telemetry Transport. Es un protocolo de mensajería regulado por la agencia de estándares OASIS. Está gestionado por el OASIS MQTT Techinal Committee.

Comúnmente utilizado en Internet-of-Things (IoT), está diseñado como un sistema de intercambio de mensajes siguiendo el patrón de Publicador-Suscriptor, que requiere un ancho de banda muy reducido, por lo que es ideal para comunicar dispositivos móviles, primando el uso de batería sobre el ancho de banda en redes que no tienen por qué ser 'fiables'.

MQTT acepta encriptación SSL para conseguir comunicación de datos segura.

2.5.1.1 Componentes MQTT

En un intercambio de mensajes MQTT hay dos actores [40] principales diferenciados: el bróker y el cliente:

- El bróker MQTT consiste en el 'servidor', normalmente alojado en la nube, que se encarga de estructurar los mensajes en las distintas colas que haya definidas. Puede haber varios implicados, interconectados formando un puente, o 'bridge'.
- El cliente MQTT es un origen o destinatario del mensaje, en función de si es publicador o suscriptor en una cola de mensajes concreta.

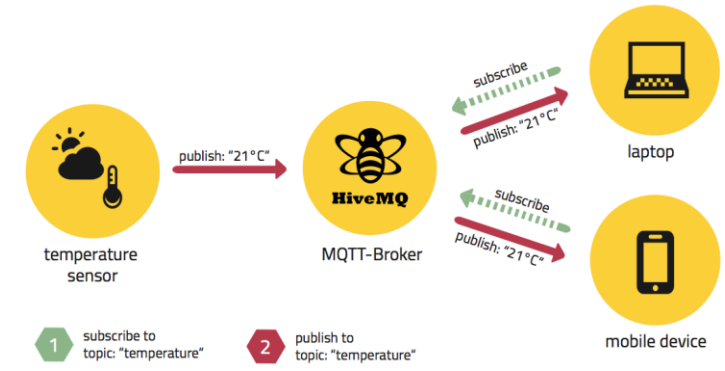

Ilustración 20 – Ejemplo de comunicación mediante MQTT

En la Ilustración 20 se muestra un esquema de ejemplo de conexión MQTT en la que intervienen múltiples clientes y un bróker.

2.5.1.2 Conceptos importantes

Una cola de mensajes se distingue de otra por medio del 'topic' o tema, una palabra clave utilizada para definir a qué cola se publica o se suscribe un cliente. De esta manera, se estructura el tráfico limitando los destinatarios de un mensaje a los que se hayan suscrito a dicha cola de mensajes.

Dentro de MQTT, hay dos tipos de conexiones: a través de sockets web, o mediante TCP. En el proyecto se han empleado sockets web para las conexiones.

2.5.2 Uso en la propuesta

Aunque en origen la tecnología MQTT está pensada para su uso en el campo de IoT, en esta propuesta constituye una solución muy interesante para ahorrar tráfico al servidor, de manera que éste tan sólo se encargue de atender peticiones de autenticación o que impliquen a la base de datos.

En la propuesta, un usuario con rol Especialista se suscribirá a una cola de mensajes usando como 'topic' su propia dirección de correo electrónico o email, que será a la que publiquen todos aquellos usuarios con rol Usuario a los que haya prescrito ejercicios que estén realizando.

De esta manera, el especialista podrá ver en tiempo real información sobre la actividad física que estén realizando los usuarios, pudiendo más tarde consultar el resumen de la actividad, esta vez sí, con la base de datos.

2.6 JSON Web Tokens (JWT)

Ilustración 21 – Logo de JWT

JSON Web Token [41] es un estándar definido en la RFC-7519 [42] para intercambiar información en formato JSON entre dos interlocutores de forma segura. Esta información va almacenada en el payload de una estructura JSON Web Signature, o en el plaintext de una estructura JSON Web Encryption, permitiendo que la información vaya firmada con un código de autenticación de mensaje (MAC por sus siglas en inglés) o bien encriptada.

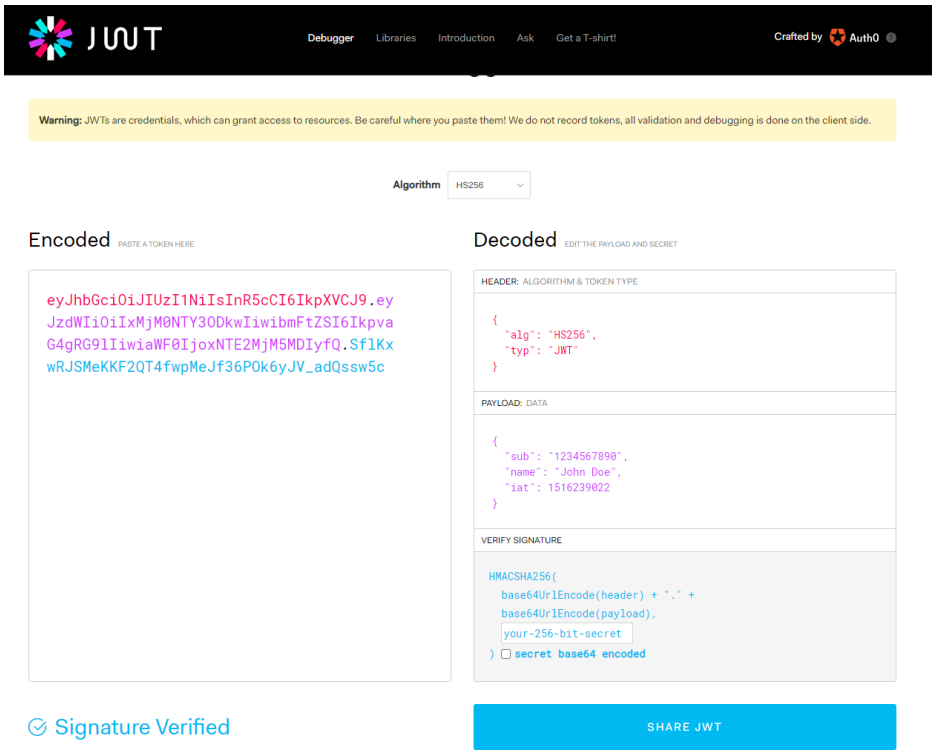

Ilustración 22 – Captura de JWT Debugger

En la propia web de jwt.io se puede acceder a un debugger, en el que se puede decodificar un token y acceder a su contenido. Además, permite conocer qué parte del token se corresponde con el encabezado, el payload o la firma, como se muestra en la Ilustración 22.

2.7 Bcrypt

Bcrypt [43] es un procedimiento de encriptación de contraseñas, creado por Niels Provos y David Mazières en el año 1999 y publicado bajo la organización sin ánimo de lucro para la investigación de sistemas informáticos avanzados USENIX [44].

Este procedimiento de encriptación surgió a raíz de los problemas que había con los hashes[6] tradicionales de contraseñas en sistemas operativos UNIX. Por ello idearon un nuevo esquema de contraseña basado en el algoritmo de cifrado Blowfish[7] [45].

En este proyecto se emplea para cifrar las contraseñas a la hora de almacenarlas en la base de datos, de manera que, aunque se acceda al contenido de la base de datos, la contraseña no se revela como una cadena de texto plano, ya que lo que se almacena es el hash.

[6] Un hash el resultado a la salida de una función de encriptación.
[7] Blowfish es un codificador de bloques de 64 bits con clave secreta. Fue publicado en el año 1993 por Bruce Scheier.

3 HERRAMIENTAS Y RECURSOS UTILIZADOS

Sólo hay un truco en el desarrollo software, y es utilizar un componente software que ya haya sido escrito.

- Bill Gates -

A la hora de seleccionar las herramientas para el desarrollo de este proyecto, se han tenido en cuenta las tecnologías a emplear, y el entorno en que se iba a desarrollar. Como el desarrollo se ha llevado a cabo en un equipo con SO Windows 10, las herramientas se han elegido para que sean lo más apropiadas a este sistema operativo, sin perder calidad ni comodidad respecto a las que pudieran haberse elegido para un SO Linux.

En este punto se van a enumerar de forma detallada todas y cada una de las herramientas que han sido utilizadas a lo largo del desarrollo del proyecto.

3.1 Herramientas de desarrollo

En este subapartado se van a detallar todas las herramientas que han sido útiles en el desarrollo de la propuesta.

3.1.1 Visual Studio Code

Ilustración 23 – Logo de VSCode

Visual Studio Code [46] es un editor de texto multiplataforma desarrollado por Microsoft. Soporta por defecto JavaScript, TypeScript y Node.JS, pero también tiene una amplia gama de extensiones para otros lenguajes, como C++, C#, Java, Python, PHP, etc., así como frameworks como .NET o Unity.

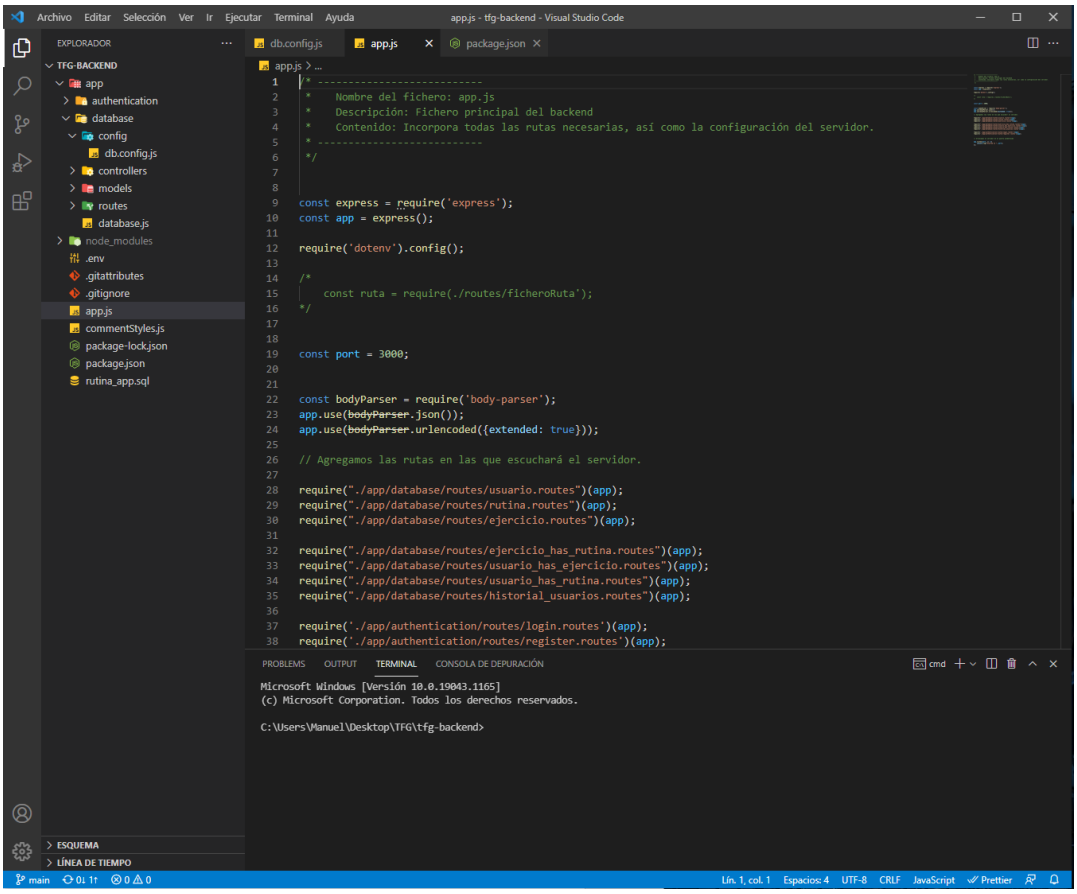

Ilustración 24 – Interfaz de VSCode

Es una opción muy interesante a la hora de programar, ya que permite incorporar en la interfaz de usuario un terminal de comandos, una vista general del proyecto, y otros aspectos que facilitan el desarrollo como los llamados 'code snippets' que permiten automatizar estructuras de código, como esquemas de funciones, clases, etc.

Otro aspecto a destacar sobre Visual Studio Code es que dispone de la posibilidad de instalar ciertas extensiones que facilitan mucho el desarrollo de aplicaciones, ya que pueden hacer el entorno más intuitivo, o ahorrar tiempo de programación, por ejemplo.

Soporta comunicación con Git -que se verá en el siguiente punto- para gestionar repositorios, así como permite abrir proyectos en máquinas remotas mediante extensiones SSH[8].

En la Ilustración 24 se puede observar la interfaz de Visual Studio Code, a veces abreviado como VSCode, con el editor de texto (centro), el explorador del proyecto (izquierda), y el terminal (abajo).

3.1.2 Git – GitHub Desktop

Ilustración 25 – Logo de git

[8] SSH son las siglas de Secure Shell. Es un protocolo de acceso remoto a máquinas a través de la red. Utiliza el puerto TCP 22 por defecto, y la comunicación va cifrada de extremo a extremo.

Git [47] es un sistema de control de versiones distribuido de código abierto diseñado para gestionar proyectos. Permite organizar las distintas versiones del código del proyecto, y documentar todos los cambios realizados a lo largo del desarrollo.

Para utilizar Git existen diversas opciones. Se puede utilizar en un terminal de comandos, algo muy útil en equipos sin interfaz gráfica. Otra opción es utilizar GitHub, un portal web que permite de forma muy sencilla gestionar todos los proyectos, organizados en repositorios, a través de un navegador.

Ilustración 26 – Logo de GitHub Desktop

Para utilizar este sistema, en el desarrollo del proyecto se ha utilizado GitHub Desktop [48], una herramienta software que permite interactuar con GitHub utilizando una interfaz gráfica de usuario, o GUI, en lugar de un terminal de comandos o un navegador web. A continuación, se muestra una vista de la interfaz de GitHub Desktop en la Ilustración 27:

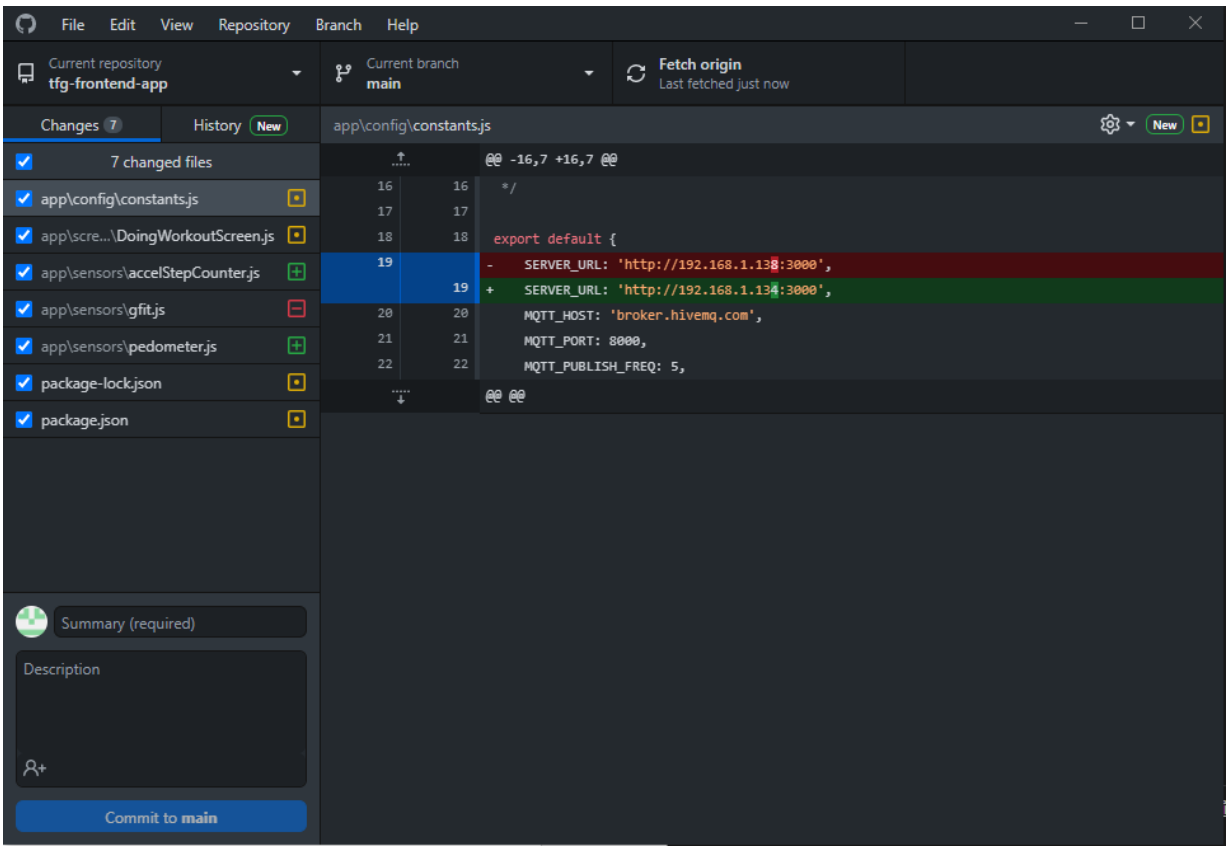

Ilustración 27 – Interfaz de GitHub Desktop

A nivel de funcionalidades y facilidad de uso es muy similar a GitHub visto desde un navegador web.

3.1.3 Entorno de desarrollo para Android

Para ir comprobando el funcionamiento del código de la aplicación móvil se han empleado dos vías, con el objetivo de poder comparar el comportamiento de la aplicación en diferentes entornos.

3.1.3.1 Expo Go App

Ilustración 28 – Logo de Expo Go App

Expo Go [49] es una aplicación, disponible en Android y en iOS, cada una en sus respectivas tiendas de aplicaciones oficiales, que permite desplegar proyectos de desarrollo en el dispositivo móvil sin tener que generar la aplicación en formato final (APK en Android, o IPA en iOS).

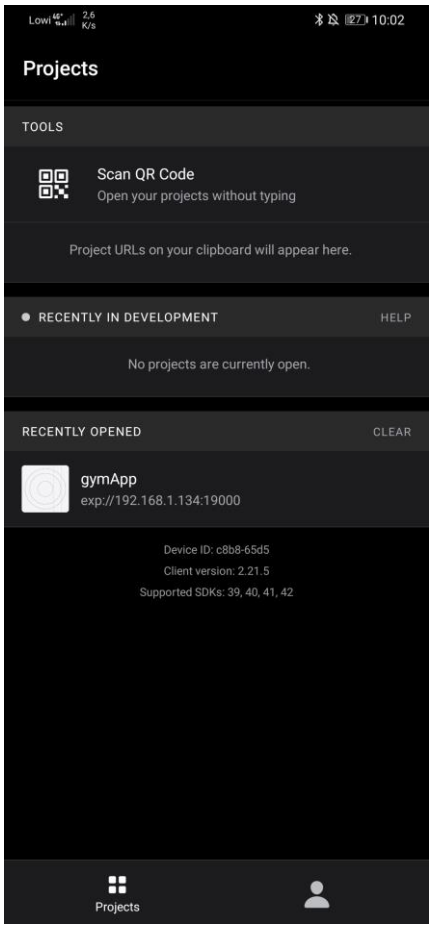

Ilustración 29 – Interfaz de Expo Go App en Android

Para desplegar la aplicación en el dispositivo móvil tan sólo hay que escanear el código QR generado al arrancar Expo en la máquina donde se desarrolla la app. Como se puede apreciar en la Ilustración 29, esta opción se ofrece en el apartado 'Tools' de la aplicación Expo Go.

Además, también permite probar la aplicación en dispositivos simulados o en un navegador web, así como mostrar información sobre la ejecución en un terminal con ayuda de Metro Bundler.

3.1.3.2 Metro bundler

Ilustración 30 – Logo de Metro Bundler

Metro [50] es un empaquetador de JavaScript, ideado para React Native y desarrollado bajo licencia MIT[9] por Facebook. Toma un archivo de entrada y una serie de opciones y devuelve un único archivo JavaScript con todo el código y sus dependencias.

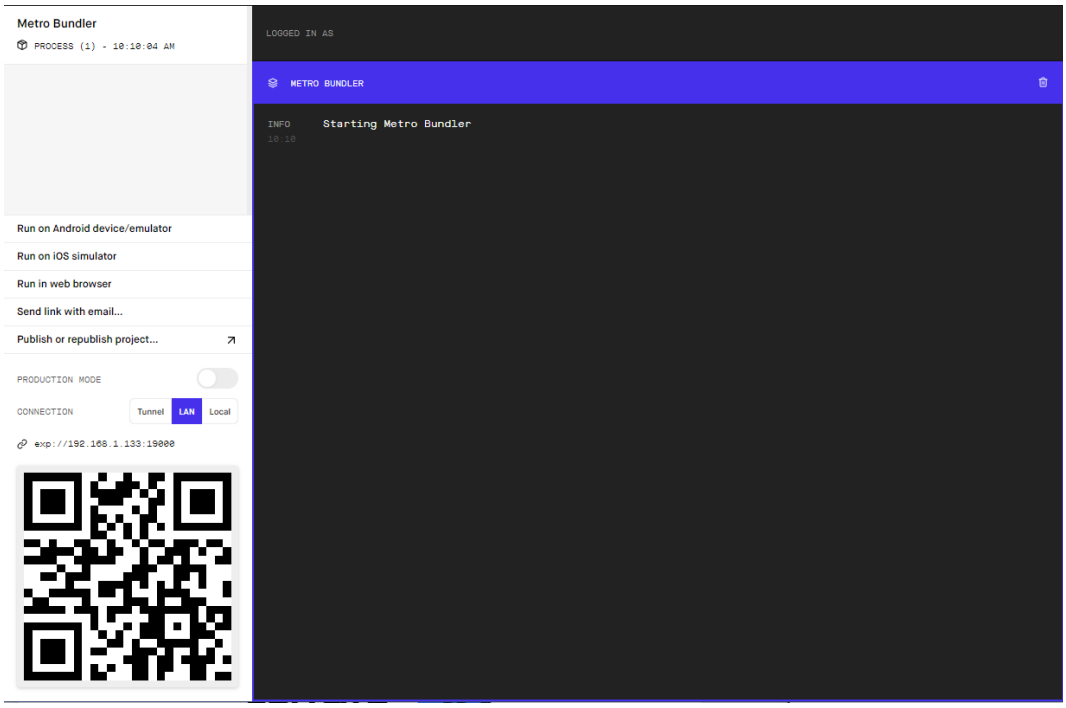

Ilustración 31 – Interfaz de Metro Bundler en un navegador web

Como se puede ver en la Ilustración 31, la interfaz de Metro bundler en el navegador tiene un terminal que informará acerca de la ejecución de la aplicación en el dispositivo móvil. Además, en el panel de la izquierda dará a elegir al usuario dónde quiere probar la aplicación, mostrando también el código QR a escanear con la aplicación Expo Go en dispositivos físicos.

[9] La licencia MIT, llamada así por las siglas del Massachusetts Institute of Technology, en español, Instituto Tecnológico de Massachusetts, es una licencia bajo la que el software es libre, por lo que puede ser reutilizado.

3.1.3.3 Android Emulator – Android Studio

Ilustración 32 – Logo de Android Studio

Android Studio [51] es el entorno de desarrollo integrado, o IDE en sus siglas en inglés, para crear aplicaciones para dispositivos Android. Contiene un editor de código, múltiples herramientas para desarrolladores o un sistema de compilación propio basado en gradle, entre otras muchas funciones.

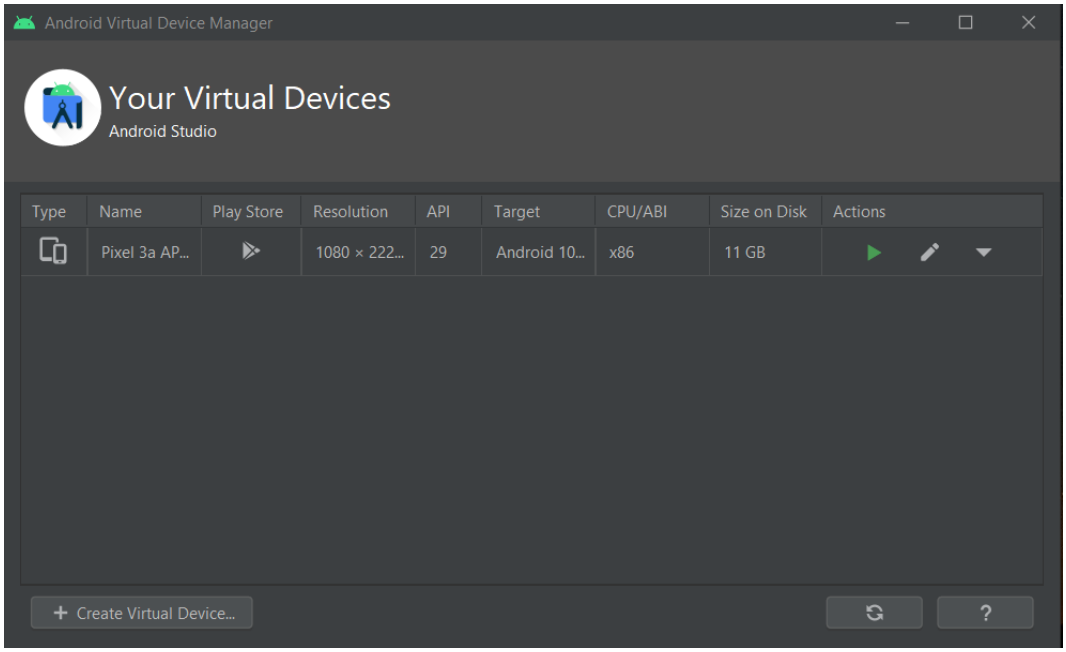

Ilustración 33 – Interfaz de Android Virtual Device Manager (Android Studio)

Lo único que se ha empleado de Android Studio en el desarrollo de la aplicación, y que no se ha mencionado entre las funciones de Android Studio, es el simulador de Android que incorpora. Permite instalar simuladores Android de multitud de terminales y versiones del sistema operativo, como se puede ver en la Ilustración 33, lo cual resulta de utilidad para analizar el comportamiento de la aplicación en entornos distintos.

3.1.4 MySQL

Las herramientas de MySQL utilizadas son proporcionadas por MySQL Community, que ofrece de forma gratuita un instalador para desplegar servidores, gestores de base de datos, etc. llamado MySQL Installer, disponible en https://dev.mysql.com/downloads/mysql/. En la Ilustración 34 se muestra una captura de pantalla de la interfaz de MySQL Installer.

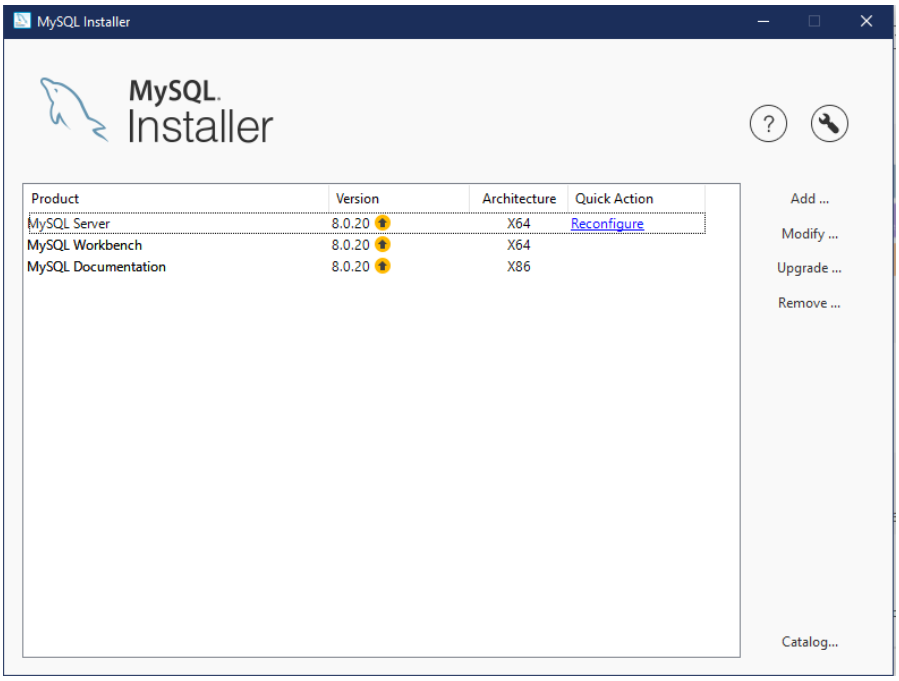

Ilustración 34 – Interfaz de MySQL Installer

Un aspecto muy interesante de este installer es que permite instalar versiones anteriores de servidores y/o Workbench, así como actualizarlos a la versión deseada. También permite instalar la documentación MySQL de la/s versión/es deseada/s.

3.1.4.1 MySQL Server

Por medio del instalador mencionado, se ha desplegado en el equipo de desarrollo del proyecto un servidor MySQL, versión 8.0.20. Este servidor, según su configuración se puede emplear para pruebas y desarrollo, aunque también es posible utilizarlo en entornos de producción.

3.1.4.2 MySQL Workbench

Ilustración 35 – Logo de MySQL Workbench

MySQL Workbench es una herramienta visual unificada para el desarrollo, gestión y administración de bases de datos MySQL. Además, permite también gestionar el servidor, y proporciona modelado de datos, editor de texto entre otras cosas. En la Ilustración 36 se muestra una vista de la interfaz de MySQL Workbench:

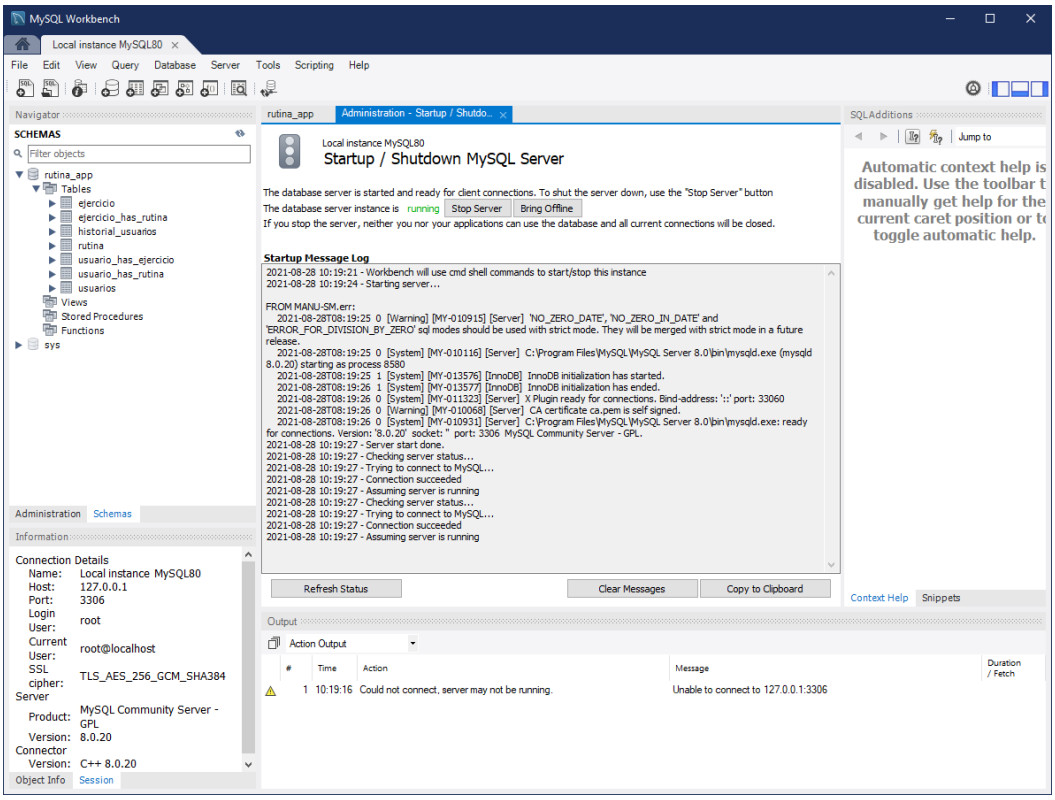

Ilustración 36 – Interfaz de MySQL Workbench. Administración de MySQL Server

La versión de MySQL Workbench utilizada ha sido la 8.0.20.

3.2 Otras herramientas utilizadas

Además de las herramientas de desarrollo utilizadas, se han empleado otras para realizar pruebas y depuración del funcionamiento de ciertas partes del proyecto.

3.2.1 Vysor

Ilustración 37 – Logo de Vysor

Vysor [52] es una aplicación para retransmitir en tiempo real la pantalla de un dispositivo móvil a un ordenador. Se ha empleado para las revisiones de la interfaz de usuario de la aplicación. En la Ilustración 38 se puede observar la interfaz de Vysor en su versión de escritorio para Windows 10.

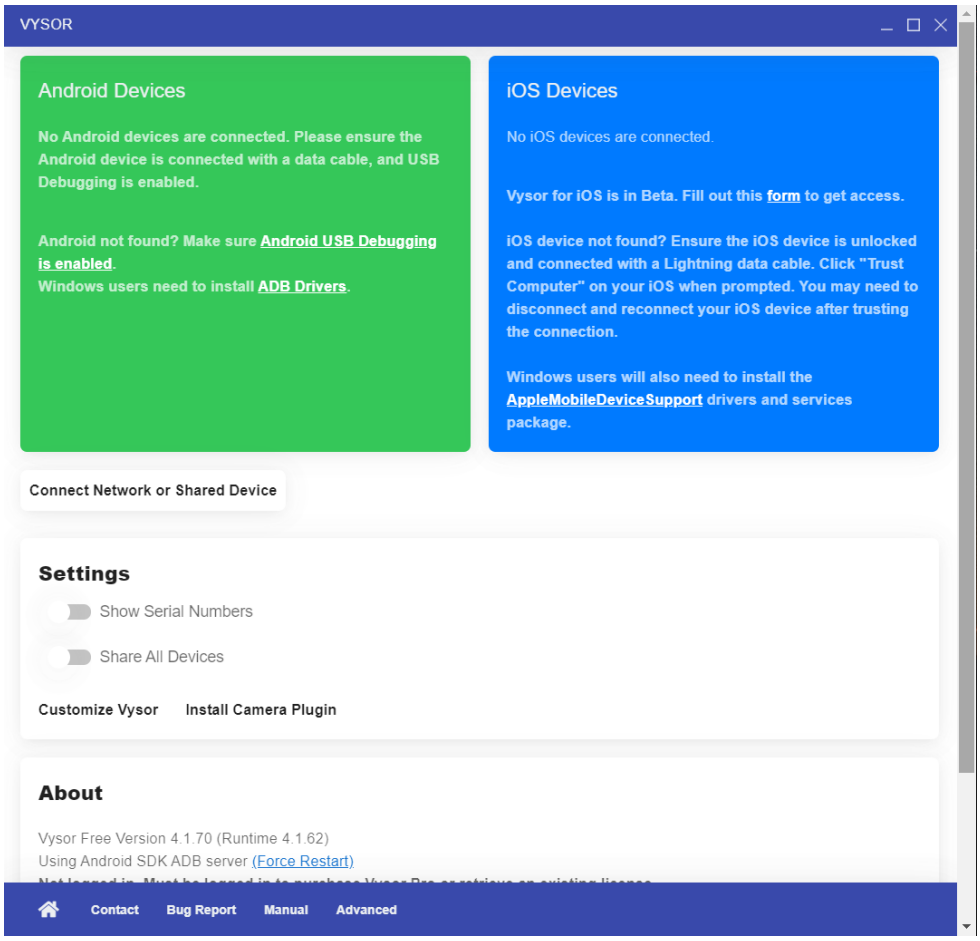

Ilustración 38 – Interfaz de Vysor

3.2.2 Postman

Ilustración 39 – Logo de Postman

Postman [53] es una herramienta de desarrollo y depuración de APIs. Permite lanzar peticiones a las API definidas en un servicio para comprobar su funcionamiento.

Permite editar todos los aspectos de la petición HTTP, desde el método utilizado, las cabeceras, el contenido y tipo de contenido del cuerpo de la petición, etc. Además, permite analizar la respuesta que se ha recibido. En la Ilustración 39, se puede ver la intefaz de Postman.

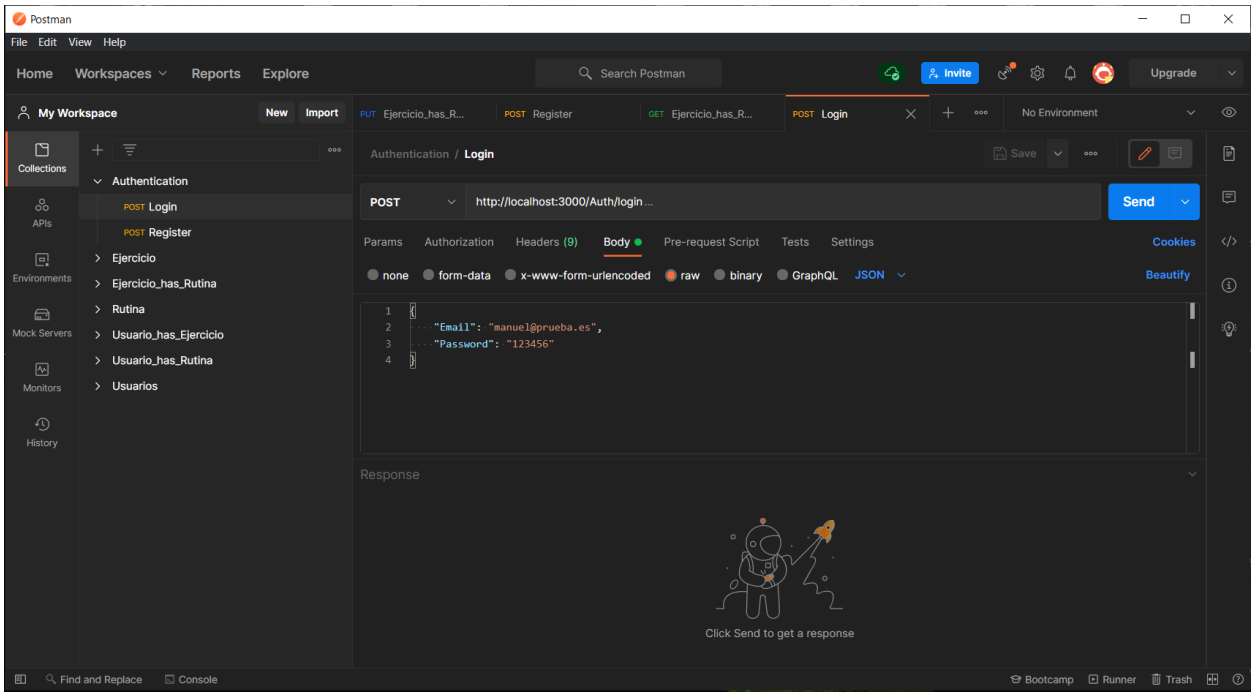

Ilustración 40 – Interfaz de Postman

Postman ha sido utilizada para depurar en fases tempranas del desarrollo del proyecto el funcionamiento de las rutas en las que escucha la API Rest del servidor (tanto las referidas a gestión de la base de datos como las de autenticación). Se mencionará posteriormente en el capítulo 6: Pruebas e Instalación.

3.2.3 HiveMQ

Ilustración 41 – Logo de HiveMQ

HiveMQ [54] es una empresa que ofrece servicios de brókers y clientes MQTT, apropiadamente configurados para poder utilizarlos fácilmente por terceros.

Ofrece un amplio abanico de soluciones a empresas, pero en el desarrollo del proyecto se ha utilizado un bróker publico que tienen habilitado para uso libre, ubicado en la dirección broker.hivemq.com. Facilitan un dashboard en su portal web con información acerca del uso que se hace en tiempo real del bróker, como se puede apreciar en la Ilustración 42.

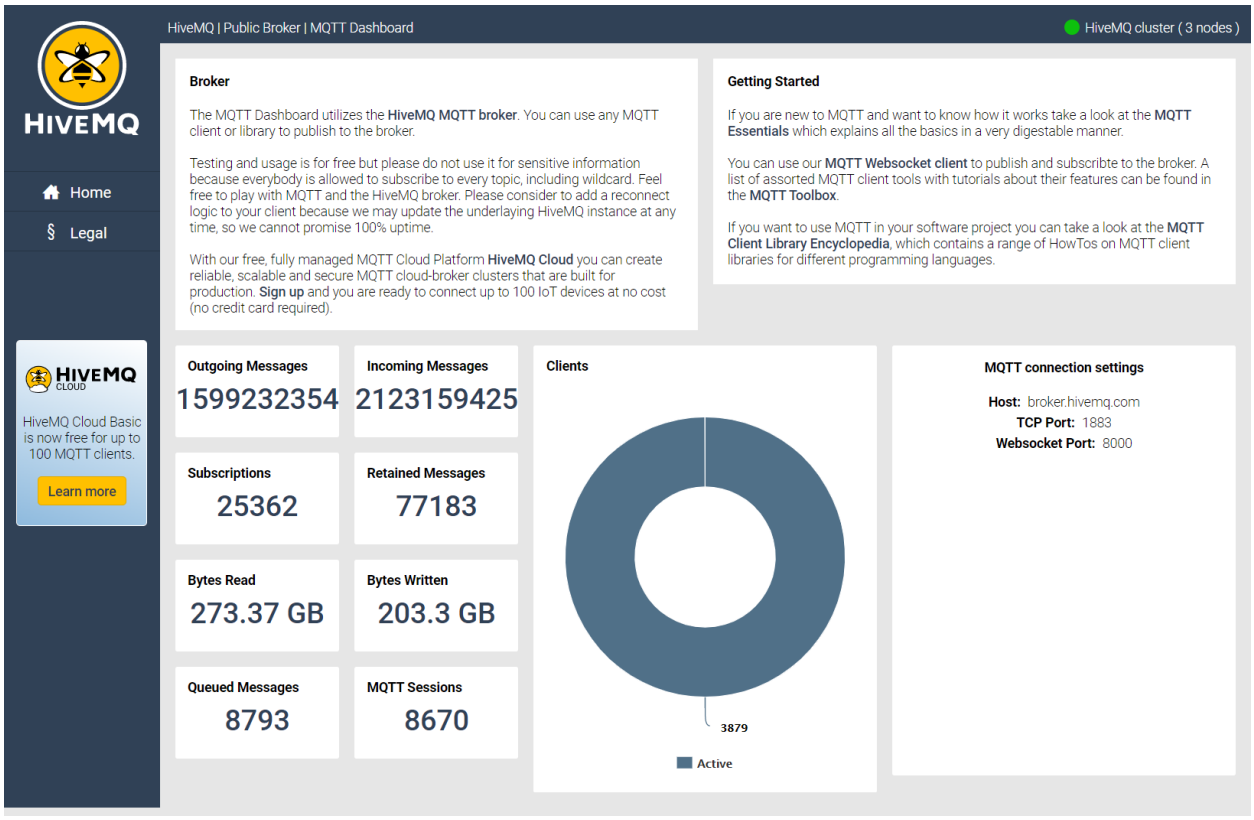

Ilustración 42 – Interfaz de HiveMQ MQTT Dashboard

Además, también proporcionan un cliente websocket a través del navegador, para probar la cola de mensajes, y poder depurar el uso en la aplicación, tanto de publicador como de suscriptor. Se puede observar una captura del cliente en la Ilustración 43.

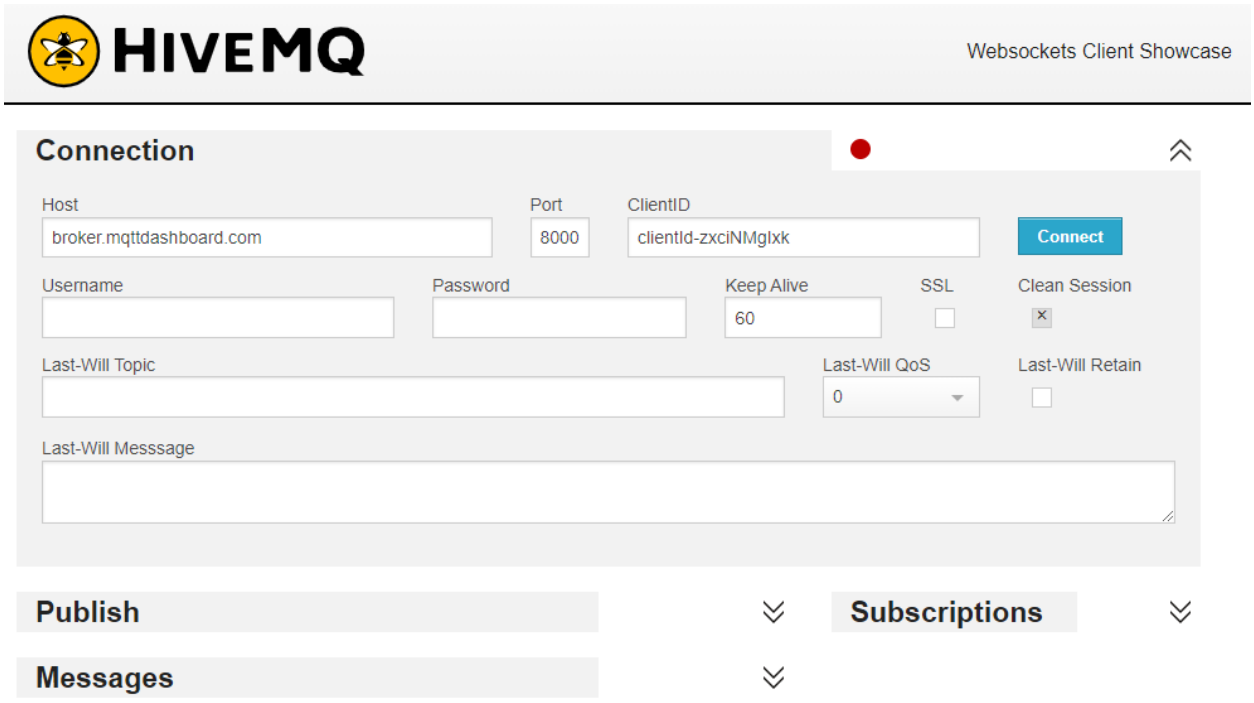

Ilustración 43 – Interfaz de HiveMQ Websockets Client Showcase

3.3 Herramientas para el modelado y diseño

En este apartado se comentan las diferentes herramientas utilizadas para el diseño, prototipado y modelado, tanto de diagramas como de componentes del servicio.

3.3.1 Editor de MySQL Workbench

Además de las funcionalidades descritas anteriormente, MySQL Workbench permite crear diagramas de bases de datos, tanto de forma inicial como realizando ingeniería inversa sobre una base de datos ya existente.

3.3.2 Diagrams.net

Ilustración 44 – Logo de Diagrams.net

Diagrams.net [55] es un servicio web gratuito para crear diagramas de diversos tipos (flujos, barras, red, secuencia, casos de uso, etc.) con una interfaz muy cómoda e intuitiva. Se ha empleado para el diagrama genérico del punto 1, los diagramas de casos de uso, y los diagramas de secuencia. Además, también ha sido utilizado para el diseño de un logo sencillo para la aplicación.

3.4 Recursos adicionales

En este punto comentaremos el origen de ciertos recursos empleados para el diseño de la aplicación móvil, distintos de las herramientas mencionadas en el apartado anterior.

3.4.1 Pixabay

Ilustración 45 – Logo de Pixabay

Pixabay [56] es un sitio web en el que su comunidad comparte recursos como videos o imágenes sin derechos de autor. La licencia que emplean es la suya propia, que permite en todo caso su uso para cualquier fin. Este sitio se ha utilizado para las imágenes de fondo de la pantalla de carga de la aplicación y de la pantalla de bienvenida.

4 ARQUITECTURA Y ANÁLISIS

Cuando escribí este código, sólo Dios y yo sabíamos lo que hacía. Ahora, sólo Dios lo sabe.

- Programador anónimo -

Conocer la arquitectura del servicio que plantea este proyecto es muy importante para entender el funcionamiento del mismo. En este punto del documento se pretende que el lector tenga una idea concreta de la organización y el funcionamiento del servicio. Por ello, se aportarán diagramas de varios tipos, mostrando puntos de vista diferenciados, para abordar las distintas funcionalidades que lo componen.

Ya que se tienen tres partes fuertemente diferenciadas -base de datos, aplicación web y aplicación móvil- se dedicará un apartado a cada una de ellas. No obstante, antes de nada, se mostrará y comentará detalladamente un diagrama de componentes para tener una visión global del proyecto.

4.1 Diagrama de Componentes

En este apartado se aportará un diagrama de componentes para tener una visión lo más general posible de todas las partes que componen el servicio y su interacción entre ellas. Dentro de cada una de ellas, se mostrarán a su vez los elementos más importantes que la integran y su relación.

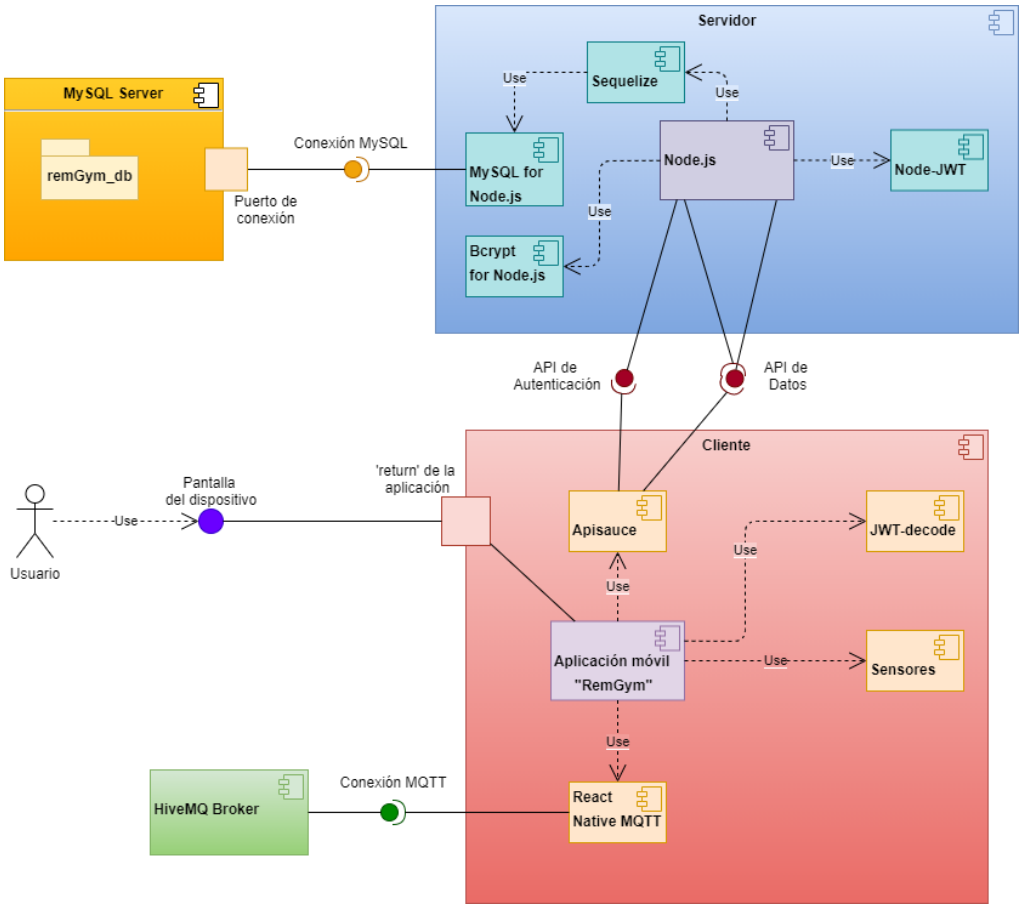

Ilustración 46 – Diagrama de componentes del servicio

En el diagrama de la Ilustración 46, se pueden apreciar todas las interfaces que existen en el servicio, y que permiten intercomunicar los distintos componentes:

- Entre el servidor y la base de datos se puede apreciar la interfaz de conexión a la base de datos, para lo cual el servidor emplea Sequelize, que a su vez hace uso del módulo de MySQL para Node.js.
- Entre el servidor y el cliente existen dos interfaces:
    - API de Autenticación: Se utilizará para verificar que el usuario está autenticado en el sistema.
    - API de Datos: La utilizará el cliente para obtener datos necesarios para el uso de la aplicación. También la utilizará el servidor desde la parte de Autenticación para obtener información acerca del usuario que pretende acceder al servicio.
- Entre el cliente y el usuario se aprecia una interfaz, constituida simplemente por la interacción del usuario con el dispositivo móvil. Ahí tanto la información mostrada por pantalla como las entradas del usuario a través de la misma constituyen dicha interfaz.
- Entre el cliente y el bróker de HiveMQ se aprecia la interfaz de conexión a la cola de mensajes distribuida MQTT. La utilizará el cliente para publicar o suscribirse -en función del rol del usuario autenticado-.

Para aclarar la interacción entre la API de Autenticación y la de Datos, se ofrece una vista de caja blanca del componente Node.js en la Ilustración 47.

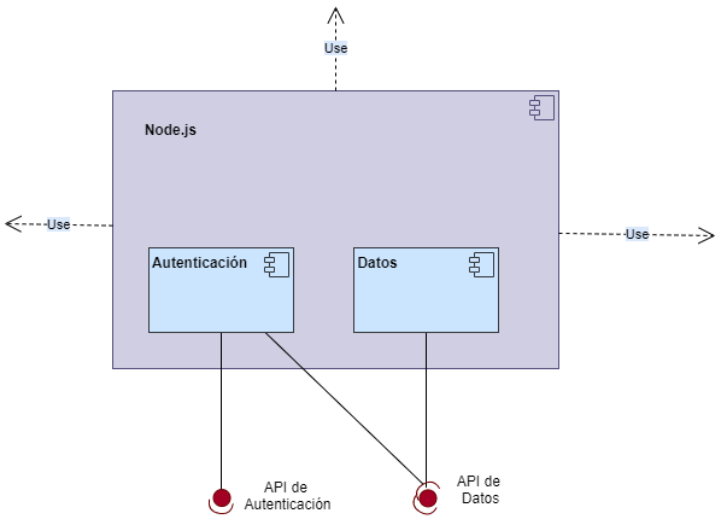

Ilustración 47 – Vista en detalle del componente Node.js

4.1.1 Patrón Modelo-Vista-Controlador (MVC)

La organización de la propuesta está planteada siguiendo el patrón Modelo-Vista-Controlador [57], como es usual en aplicaciones web. La diferencia en este caso es que la vista se encuentra algo más separada del resto de elementos. A continuación, se muestra un ejemplo del patrón en la Ilustración 48:

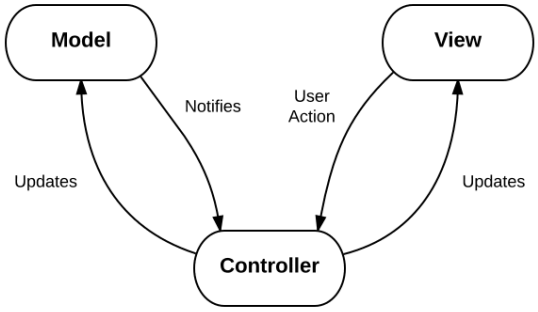

Ilustración 48 – Patrón MVC

Sobre esta imagen, se puede detallar cómo encaja cada elemento en el servicio:

- Modelo: Consiste tanto en la Base de datos como la sección de modelos del código del servidor, generado para el uso de Node.js a través de Sequelize.
- Vista: Consiste en la Interfaz de usuario, esto es, la aplicación móvil.
- Controlador: Consiste en el resto de lógica del servidor, encargada de atender las peticiones de la vista -aplicación móvil- y extraer información del modelo para satisfacerlas apropiadamente.

4.2 Base de datos

En este apartado se va a ver en detalle la estructura de la base de datos, la función de cada una de sus tablas y el significado que pueda tener cada campo en el contexto de uso del servicio.

Esta base de datos supone una modificación de la base de datos creada por Pablo Carmona en su trabajo de fin de grado '*Plataforma web de prescripción de ejercicios usando Spring*', la cual fue un punto de partida en el desarrollo del proyecto.

4.2.1 Diagrama ER de la base de datos

En este subapartado se va a mostrar y comentar la organización del diagrama entidad-relación de la base de datos. El diagrama entidad-relación va a identificar cada tabla, sus campos, sus claves, tanto primarias como externas si las tiene, y las relaciones que tiene con otras tablas por medio de dichas claves externas.

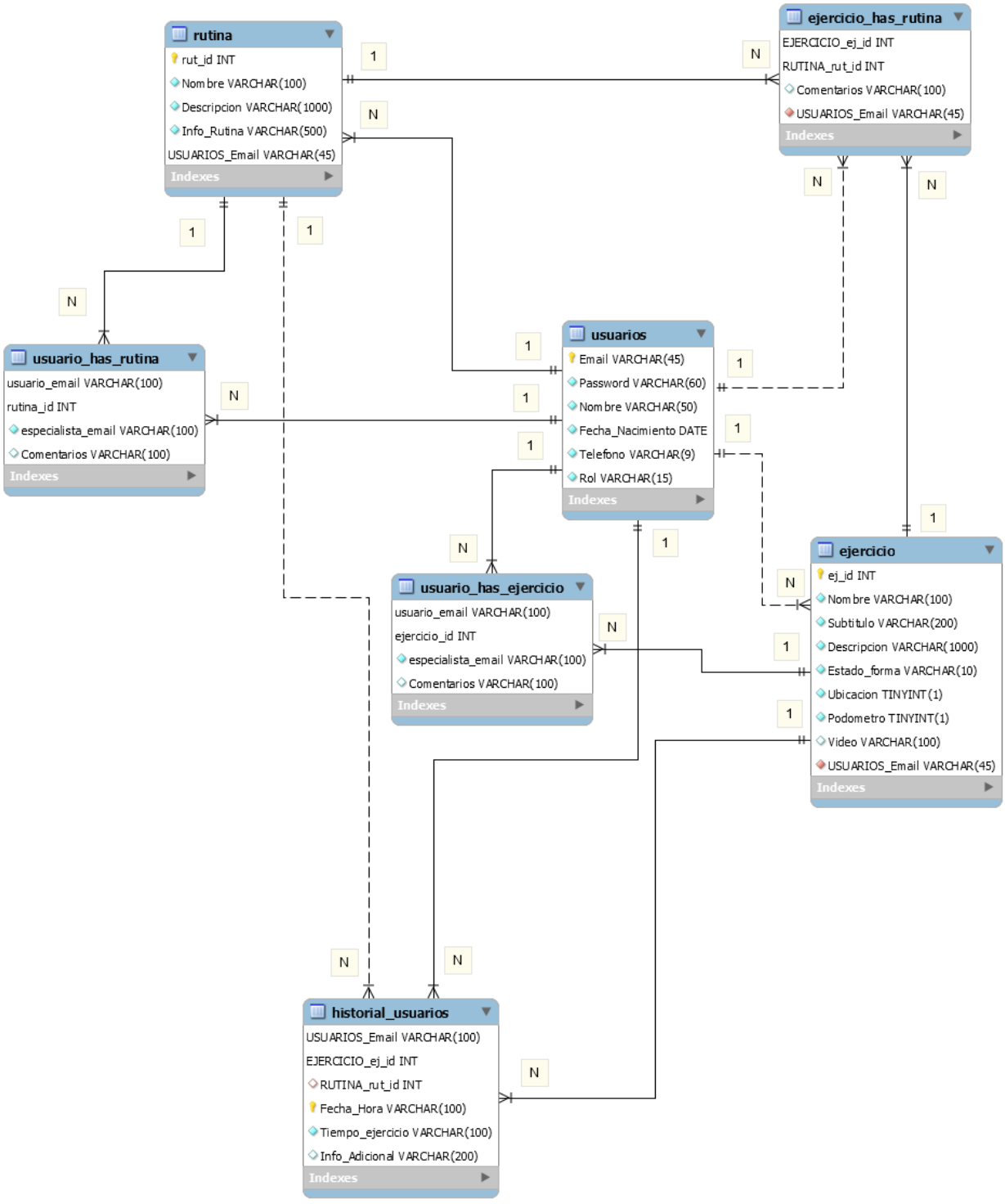

Ilustración 49 – Diagrama ER de la Base de Datos

Como se puede apreciar en la Ilustración 49, el esquema de la base de datos se asienta sobre tres tablas principales: 'usuarios', 'ejercicio' y 'rutina'. Sobre estas tres tablas surgirán otras, encargadas de aplicar restricciones a las relaciones que puedan tener en el servicio, y/o mostrarán información adicional, como puede ser el caso de la tabla 'historial_usuarios'.

En el siguiente punto se verán en detalle las tablas una a una, explicando el significado de cada uno de los campos.

4.2.2 Tablas que componen la base de datos

Las tablas que forman el esquema de la base de datos son las siguientes:

4.2.2.1 Tabla USUARIOS

Esta tabla se encarga de almacenar toda la información relativa a un usuario del servicio. Su estructura se muestra en la Tabla 1.

Campo	Descripción
Email	Cadena de caracteres. Este campo contiene la dirección de correo electrónico del usuario. Es **clave primaria** de esta tabla. No nulo.
Password	Cadena de caracteres. Este campo contiene la contraseña cifrada del usuario para inicio de sesión. No nulo.
Nombre	Cadena de caracteres. Este campo contiene el nombre y los apellidos del usuario. No nulo.
Fecha_Nacimiento	Fecha (DATE). Este campo contiene la fecha de nacimiento del usuario, en formato AAAA-MM-DD. No nulo.
Telefono	Cadena de caracteres. Este campo contiene el número de teléfono del usuario. Único. No nulo.
Rol	Cadena de caracteres. Este campo indica el rol que desempeña el usuario en el servicio. Podrá ser, por limitaciones en el servicio, 'Especialista' o 'Usuario'. No nulo.

Tabla 1 – Estructura de la tabla Usuarios

4.2.2.2 Tabla RUTINA

Esta tabla se encarga de almacenar toda la información relativa a una rutina de ejercicios. Su estructura se muestra en la Tabla 2.

Campo	Descripción
rut_id	Entero. Este campo contiene el identificador de la rutina en el sistema. Es **clave primaria** de esta tabla. No nulo.
Nombre	Cadena de caracteres. Este campo contiene el nombre de la rutina de ejercicios. No nulo.
Descripcion	Cadena de caracteres. Este campo contiene la descripción de la rutina de ejercicios. No nulo.
Info_Rutina	Cadena de caracteres. Este campo contiene información adicional sobre la rutina. No nulo.
USUARIOS_Email	Cadena de caracteres. Este campo contiene el email del especialista que ha creado la rutina. Es clave externa de la tabla usuarios. No nulo.

Tabla 2 – Estructura de la tabla Rutina

4.2.2.3 Tabla EJERCICIO

Esta tabla se encarga de almacenar toda la información relativa a un ejercicio. Su estructura se muestra en la Tabla 3.

Campo	Descripción
ej_id	Entero. Este campo contiene el identificador del ejercicio en el sistema. Es **clave primaria** de esta tabla. No nulo.
Nombre	Cadena de caracteres. Este campo contiene el nombre del ejercicio. No nulo.
Subtitulo	Cadena de caracteres. Este campo contiene el subtitulo del ejercicio. No nulo.
Descripcion	Cadena de caracteres. Este campo contiene una descripción detallada de cómo realizar el ejercicio. No nulo.
Estado_forma	Cadena de caracteres. Este campo indica el estado de forma recomendado para realizar el ejercicio. No nulo.
Ubicacion	Boolean. Este campo indica si el ejercicio hará uso de los sensores de geolocalización o no. No nulo.
Podometro	Boolean. Este campo indica si el ejercicio hará uso del contador de pasos o no. No nulo.
Video	Cadena de caracteres. Este campo contiene una URL de un video de YouTube, que se mostrará al realizar ciertos ejercicios. Puede ser nulo.
USUARIOS_Email	Cadena de caracteres. Este campo contiene el email del especialista que ha creado el ejercicio. Es clave externa de la tabla usuarios. No nulo.

Tabla 3 – Estructura de la tabla Ejercicio

4.2.2.4 Tabla EJERCICIO_has_RUTINA

Esta tabla define las relaciones existentes entre las tablas EJERCICIO y RUTINA. Como se ha mostrado anteriormente en el diagrama, la cardinalidad entre estas tablas es de muchos a muchos (M:N), ya que una rutina puede contener muchos ejercicios y un único ejercicio puede estar incluido en muchas rutinas. Su estructura se muestra en la Tabla 4.

Campo	Descripción
EJERCICIO_ej_id	Entero. Este campo contiene el identificador del ejercicio. Es clave externa de la tabla ejercicio. No nulo.
RUTINA_rut_id	Entero. Este campo contiene el identificado de la rutina. Es clave externa de la tabla rutina. No nulo.
Comentarios	Cadena de caracteres. Este campo contiene los comentarios acerca de la asociación de un ejercicio a una rutina introducidos por el especialista a la hora de crear dicha asociación.
USUARIOS_Email	Cadena de caracteres. Este campo contiene la dirección de correo del especialista que realiza la asociación. No nulo.

Tabla 4 – Estructura de la tabla Ejercicio_has_Rutina

En el caso de esta tabla, la clave primaria es compuesta, formada por los campos EJERCICIO_ej_id y RUTINA_rut_id.

4.2.2.5 Tabla USUARIO_has_RUTINA

Esta tabla define las relaciones existentes entre las tablas USUARIOS y RUTINA. Como se ha mostrado anteriormente en el diagrama, la cardinalidad entre estas tablas es de muchos a muchos (M:N), ya que un usuario puede tener prescritas varias rutinas, y una única rutina puede ser prescrita a muchos usuarios. Su estructura se muestra en la Tabla 5.

Campo	Descripción
usuario_email	Cadena de caracteres. Este campo contiene el correo electrónico del usuario al que se prescribe la rutina. Es clave externa de la tabla usuarios. No nulo.
rutina_id	Entero. Este campo contiene el identificador de la rutina que se prescribe. Es clave externa de la tabla rutina. No nulo.
especialista_email	Cadena de caracteres. Este campo contiene la dirección de correo electrónico del especialista que realiza la prescripción. Es clave externa de la tabla usuarios. No nulo.
Comentarios	Cadena de caracteres. Este campo contiene los comentarios acerca de la prescripción, establecidos por el especialista que la crea.

Tabla 5 – Estructura de la tabla Usuario_has_Rutina

En el caso de esta tabla, la clave primaria es compuesta, formada por los campos usuario_email y rutina_id.

4.2.2.6 Tabla USUARIO_has_EJERCICIO

Esta tabla define las relaciones existentes entre las tablas USUARIOS y EJERCICIO. Como se ha mostrado anteriormente en el diagrama, la cardinalidad entre estas tablas es de muchos a muchos (M:N), ya que un

usuario puede tener prescritos varios ejercicios, y un único ejercicio puede estar prescrito a muchos usuarios. Su estructura se muestra en la Tabla 6.

Campo	Descripción
usuario_email	Cadena de caracteres. Este campo contiene el correo electrónico del usuario al que se prescribe el ejercicio. Es clave externa de la tabla usuarios. No nulo.
ejercicio_id	Entero. Este campo contiene el identificador del ejercicio que se prescribe. Es clave externa de la tabla ejercicio. No nulo.
especialista_email	Cadena de caracteres. Este campo contiene la dirección de correo electrónico del especialista que realiza la prescripción. Es clave externa de la tabla usuarios. No nulo.
Comentarios	Cadena de caracteres. Este campo contiene los comentarios acerca de la prescripción, establecidos por el especialista que la crea.

Tabla 6 – Estructura de la tabla Usuario_has_Ejercicio

En el caso de esta tabla, la clave primaria es compuesta, formada por los campos usuario_email y ejercicio_id.

4.2.2.7 Tabla HISTORIAL_USUARIOS

Esta tabla recogerá la información sobre el histórico de actividad física de los usuarios del servicio. Su estructura se muestra en la Tabla 7.

Campo	Descripción
USUARIOS_Email	Cadena de caracteres. Este campo contiene la dirección de correo electrónico del usuario que ha realizado el ejercicio. Es clave externa de la tabla usuarios. No nulo.
EJERCICIO_ej_id	Entero. Este campo contiene el identificador del ejercicio realizado. Es clave externa de la tabla ejercicio. No nulo.
RUTINA_rut_id	Entero. Este campo contiene el identificador de la rutina a la que se asocia el ejercicio realizado, si procede. Es clave externa de la tabla rutina. Puede ser nulo.
Fecha_Hora	Cadena de caracteres. Este campo contiene la fecha y la hora en formato 'DD/MM/AAAA HH:MM:SS' en la que el usuario comenzó el ejercicio. No nulo.
Tiempo_ejercicio	Cadena de caracteres. Este campo contiene el tiempo transcurrido en formato 'HH:MM:SS' durante la realización del ejercicio. No nulo.
Info_Adicional	Cadena de caracteres. Este campo contiene información extra sobre el ejercicio, si existe.

Tabla 7 – Estructura de la tabla Historial_Usuarios

En el caso de esta tabla, la clave primaria es compuesta, formada por los campos USUARIOS_Email, EJERCICIO_ej_id y Fecha_Hora. El motivo de esto es que un mismo usuario puede realizar todas las veces que desee el mismo ejercicio, pero nunca lo realizará más de una vez a la misma hora. Puede parecer algo obvio, pero es la forma más sencilla de garantizar la unicidad de la clave primaria en la tabla.

4.3 Aplicación web

En este apartado veremos en detalle la estructura de la aplicación web de la propuesta, así como la función de cada una de las partes que lo componen.

4.3.1 Estructura

Como se ha indicado al principio de este punto del documento, se ha seguido el patrón Modelo-Vista-Controlador. En el caso del servidor, comprende tanto el modelo -en parte, ya que la base de datos es el resto-, como el controlador.

El código del servidor está recogido en una carpeta general llamada 'app'. Esto tiene un par de excepciones:

- Fichero app.js: Es el fichero principal del servidor. Al arrancar el servidor, Node.js ejecuta este fichero, que hará uso y cargará lo necesario del resto.
- Fichero .env: Fichero de variables de entorno. Define algunos parámetros necesarios para la ejecución del servidor.

A continuación, se muestra la organización del código del servidor:

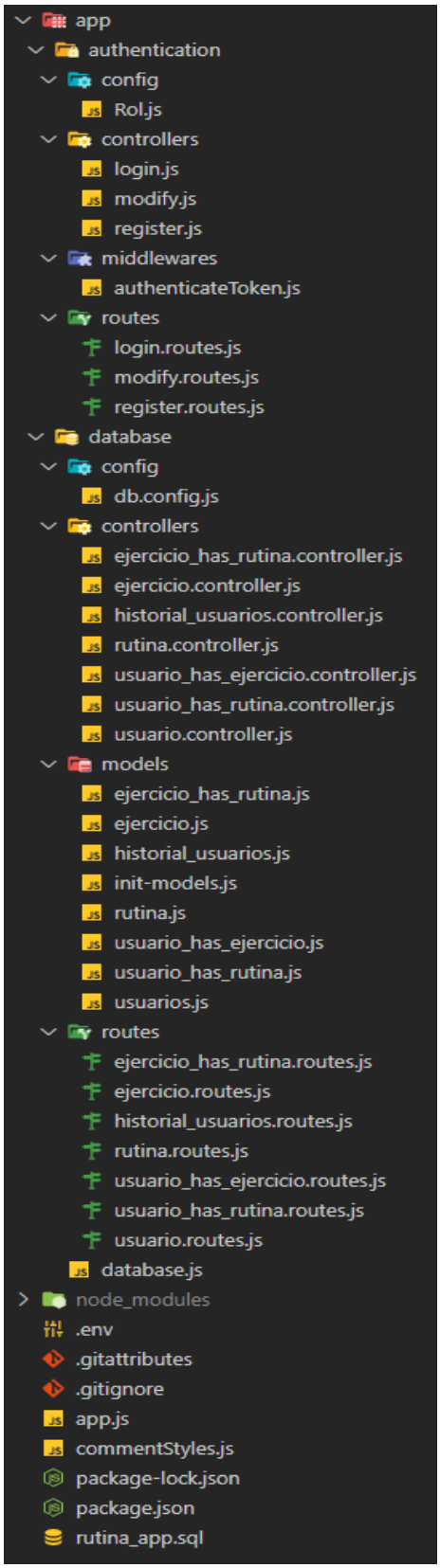

Ilustración 50 – Árbol de directorios del código del servidor

Como se puede ver en la Ilustración 50, se distingue entre la API Rest general para gestionar la base de datos del sistema, y la API de Autenticación. No obstante, al final el funcionamiento es similar, aunque para tener el código organizado de forma más clara, se optó por esta estrategia.

Tanto la API de uso de la base de datos como la de Autenticación tienen dividido su funcionamiento en tres partes diferenciadas:

- Modelos: Contienen los modelos definidos usando la librería Sequelize para hacer uso de la base de datos desde el servidor. La API de autenticación no tiene esta carpeta ya que usará a su vez la API de la base de datos, mediante peticiones al propio servidor.
- Controladores: Contienen la lógica del servidor. Serán los encargados de utilizar los modelos para elaborar el resultado que solicite la aplicación a través de la API.
- Rutas: Definen las rutas HTTP bajo las que escuchará el servidor para consumir la API. Se comentarán en el apartado de URIs, explicando qué significa cada una.

Además de estas partes, la API de Autenticación tiene otra denominada 'middlewares'. Un middleware es un fragmento de código que se ejecutará 'en medio'. Como el procesado de la petición consiste en pasar la petición, en la ruta que el servidor la escucha, al controlador correspondiente, si existe un middleware en este caso, será el primero en procesarla y determinar si la desecha o la redirige al controlador inicialmente definido. Contiene la lógica de autenticación que permitirá restringir las URIs de la API a usuarios autenticados. Además, en algunos casos, también se restringirán exclusivamente a usuarios con rol 'Especialista'.

Adicionalmente, se puede observar que cada subcarpeta -authentication y database- tiene una llamada 'config'. Estas carpetas contendrán ficheros destinados a definir parámetros de configuración que serán usados durante la ejecución del servidor.

Para más información, el código del servidor se encuentra alojado en un repositorio de GitHub, indicado en el ANEXO A - Instalación.

4.3.2 Uso de ORM frente a DAO

En este subapartado comentaré las diferencias en el método de gestión de la base de datos desde el servidor con la propuesta del trabajo de fin de grado de Pablo Carmona.

En su propuesta, Pablo planteaba un patrón DAO[10] [58] para acceder a la base de datos. El patrón DAO permite crear una capa de abstracción que separa al servidor del sistema de persistencia de datos, en este caso una base de datos relacional. La principal ventaja de este patrón es que oculta a la aplicación cualquier información acerca de la base de datos, de manera que simplifica mucho el uso de operaciones de tipo CRUD -siglas de Create, Read, Update, Delete-.

Frente a esta propuesta, el uso de Sequelize como biblioteca para gestionar la base de datos desde el servidor plantea un formato distinto, ORM [59].

Un ORM, explicado de forma simplista, mapea el contenido de una base de datos a objetos que son identificables por el servidor, de manera que se vale de éstos para gestionarla. Así, las operaciones básicas CRUD se realizarán utilizando estos objetos como intermediarios con la base de datos.

El proceso de creación de estos objetos, llamados modelos en el código del servidor, y que conforman en parte el componente 'Modelo' en el patrón MVC descrito en el punto 4.1.1, se detalla en el Anexo A – Instalación. En este anexo, se comentará el uso de la librería 'sequelize-auto', mencionada en el punto 2.4.3.7 del documento. A continuación, se muestra un fragmento de código en la Ilustración 51, en la que se recoge la definición del modelo de la rutina:

[10] DAO son las siglas de Data Access Object.

```
const Sequelize = require('sequelize');
module.exports = function(sequelize, DataTypes) {
  return sequelize.define('rutina', {
    rut_id: {
      autoIncrement: true,
      type: DataTypes.INTEGER,
      allowNull: false,
      primaryKey: true
    },
    Nombre: {
      type: DataTypes.STRING(100),
      allowNull: false
    },
    Descripcion: {
      type: DataTypes.STRING(1000),
      allowNull: false
    },
    Info_Rutina: {
      type: DataTypes.STRING(500),
      allowNull: false
    },
    USUARIOS_Email: {
      type: DataTypes.STRING(45),
      allowNull: false,
      primaryKey: true,
      references: {
        model: 'usuarios',
        key: 'Email'
      }
    }
  }, {
    sequelize,
    tableName: 'rutina',
    timestamps: false,
    indexes: [
      {
        name: "PRIMARY",
        unique: true,
        using: "BTREE",
        fields: [
          { name: "rut_id" },
          { name: "USUARIOS_Email" },
        ]
      },
      {
        name: "fk_RUTINA_USUARIOS1",
        using: "BTREE",
        fields: [
          { name: "USUARIOS_Email" },
        ]
      },
    ]
  });
};
```

Ilustración 51 – Fragmento de código. Modelo de rutina

La gran ventaja de ORM, como se comentó de forma superficial al explicar el uso de Sequelize, es que se libera al programador de escribir sentencias SQL concretas, ya que todo se hace sobre los modelos. Esto incorpora de forma indirecta una medida de seguridad adicional, al prevenir de ataques empleando SQL Injection, siempre dependiendo de la forma en que se acceda a la base de datos, por supuesto.

En la Ilustración 52 se muestra un fragmento de código en el que se hace uso de estos modelos para realizar alguna operación sobre la base de datos:

```
/* -------------------------
 *    Nombre de la Función: create
 *    Funcionamiento: Crea una nueva entrada en la tabla.
 *    Argumentos que recibe:
 *          - req: Request (Petición). Objeto con información de la petición enviada por
 *                el usuario.
 *          - res: Response (Respuesta). Objeto con información de la respuesta que se enviará.
 *    Devuelve: Nada (void).
 * -------------------------
 */

exports.create = (req, res) => {
    if (!req.body) {
        res.status(400).send({
          message: "No se puede crear una rutina vacía."
        });
        return;
    }

    // Extraemos la información de la petición
    const rutina = {
        Nombre: req.body.Nombre,
        Descripcion: req.body.Descripcion,
        Info_Rutina: req.body.Info_Rutina,
        Pub_priv: req.body.Pub_priv,
        USUARIOS_Email: req.body.USUARIOS_Email
    };


    // Ejecutamos el método create definido en el modelo de la bbdd correspondiente.
    Rutina.create(rutina)
    .then(data => {
        res.send(data);
    })
    .catch(err => {
        res.status(500).send({
            message:
              err.message || "Ha habido un error al crear la Rutina."
          });
    });
};
```

Ilustración 52 – Fragmento de código. Función create de rutina.controller.js

4.3.3 URIs

Este subapartado está para listar todas y cada una de las URIs en las que escucha el servidor para interactuar con la aplicación móvil. Se separarán por autenticación e interacción con la bbdd en los puntos siguientes.

4.3.3.1 Autenticación

A continuación, se muestra en la Tabla 8 la información sobre las distintas URIs en las que escuchará la API de Autenticación.

Fichero que la define	Método del controller asociado	Método HTTP	Ruta relativa	Descripción
login.routes.js	login	POST	/Auth/login	Atiende una solicitud de inicio de sesión.
modify.routes.js	modify	POST	/Auth/modify	Modifica los datos de usuario.
register.routes.js	register	POST	/Auth/register	Registra a un nuevo usuario en el sistema.

Tabla 8 – URIs de Autenticación

4.3.3.2 Interacción con la BBDD

A continuación, se muestra una serie de tablas con información de las distintas URIs en las que escuchará la API de la base de datos, organizadas por las tablas a las que afecta:

- Tabla EJERCICIO_has_RUTINA: Las URIs se encuentran definidas en el fichero ejercicio_has_rutina.routes.js. Se recopilan en la Tabla 9.

Método del controller asociado	Método HTTP	Ruta relativa	Descripción
create	POST	/WorkoutRoutine/	Crea una nueva asociación de ejercicio a rutina.
findAllWorkouts	GET	/WorkoutRoutine/ Routine/:RUTINA_rut_id	Obtiene todas las asociaciones de ejercicio a una rutina concreta.
update	PUT	/WorkoutRoutine/ :EJERCICIO_ej_id/:RUTINA _rut_id	Actualiza la información de una asociación de ejercicio a rutina.
delete	DELETE	/WorkoutRoutine/ Workout-&-Routine/ :EJERCICIO_ej_id/:RUTINA _rut_id	Elimina una asociación de ejercicio a rutina.

Tabla 9 – URIs de Ejercicio_has_Rutina

- Tabla EJERCICIO: Las URIs se encuentran definidas en el fichero ejercicio.routes.js. Se recopilan en la Tabla 10.

Método del controller asociado	Método HTTP	Ruta relativa	Descripción
create	POST	/Workout/	Crea un nuevo ejercicio en la base de datos.
findAll	GET	/Workout/	Obtiene todos los ejercicios.
update	PUT	/Workout/:ej_id	Actualiza la información de un ejercicio concreto.
delete	DELETE	/Workout/:ej_id	Elimina un ejercicio concreto.

Tabla 10 – URIs de Ejercicio

- Tabla HISTORIAL_USUARIOS: Las URIs se encuentran definidas en el fichero historial_usuarios.routes.js. Se recopilan en la Tabla 11.

Método del controller asociado	Método HTTP	Ruta relativa	Descripción
create	POST	/Records/	Crea un registro de ejercicio realizado.
findAllFromUser	GET	/Records/:USUARIOS_Email	Obtiene los registros de ejercicios de un usuario concreto.

Tabla 11 – URIs de Historial_Usuarios

- Tabla RUTINA: Las URIs se encuentran definidas en el fichero rutina.routes.js. Se recopilan en la Tabla 12.

Método del controller asociado	Método HTTP	Ruta relativa	Descripción
create	POST	/Routine/	Crea una nueva rutina en la base de datos.
findAll	GET	/Routine/	Obtiene todas las rutinas.
update	PUT	/Routine/:rut_id	Actualiza la información de una rutina concreta.
delete	DELETE	/Routine/:rut_id	Elimina una rutina concreta.

Tabla 12 – URIs de Rutina

- Tabla USUARIO_has_EJERCICIO: Las URIs se encuentran definidas en el fichero usuario_has_ejercicio.routes.js. Se recopilan en la Tabla 13.

Método del controller asociado	Método HTTP	Ruta relativa	Descripción
create	POST	/UserWorkout/	Añade una prescripción de ejercicio a un usuario.
findAllWorkouts	GET	/UserWorkout/User/:usuario_email	Obtiene todas las prescripciones de ejercicios a un usuario concreto.
update	PUT	/UserWorkout/User-&-Workout/ :usuario_email/:ejercicio_id	Actualiza la información de una prescripción de ejercicio a usuario.
delete	DELETE	/UserWorkout/User-&-Workout/ :usuario_email/:ejercicio_id	Elimina una prescripción de ejercicio a usuario.

Tabla 13 – URIs de Usuario_has_Ejercicio

- Tabla USUARIO_has_RUTINA: Las URIs se encuentran definidas en el fichero usuario_has_rutina.routes.js. Se recopilan en la Tabla 14.

Método del controller asociado	Método HTTP	Ruta relativa	Descripción
create	POST	/UserRoutine/	Añade una nueva prescripción de rutina a un usuario.
findAllRoutines	GET	/UserRoutine/User/:usuario_email	Obtiene todas las prescripciones de rutinas a un usuario concreto.
update	PUT	/UserRoutine/User-&-Routine/:usuario_email/:rutina:id	Actualiza la información de una prescripción de rutina a usuario.
delete	DELETE	/UserRoutine/User-&-Routine/:usuario_email/:rutina:id	Elimina una prescripción de rutina a usuario.

Tabla 14 – URIs de Usuario_has_Rutina

- Tabla USUARIOS: Las URIs se encuentran definidas en el fichero usuario.routes.js. Se recopilan en la Tabla 15.

Método del controller asociado	Método HTTP	Ruta relativa	Descripción
create	POST	/User/	Crea un nuevo usuario en la base de datos.
findAll	GET	/User/	Obtiene todos los usuarios.
findOne	GET	/User/:email	Obtiene un usuario.
update	PUT	/User/:email	Actualiza la información de un usuario.

Tabla 15 – URIs de Usuarios

Nota: Algunas rutas, así como algunos métodos, que pudiera parecer interesante añadirlos, como eliminar un usuario, no se contemplan en la solución propuesta. Aun así, se encuentran definidos los métodos correspondientes en los controladores, de cara a posibles líneas de mejora o ampliación del servicio. Esto se comentará al final del documento, en el punto 7 – Líneas de mejora. No obstante, al no estar en uso estas rutas, se ha optado por que el servidor no las escuche con el único objetivo de que estén ociosas.

4.4 Aplicación móvil

La interfaz de usuario está constituida por una aplicación móvil. Ésta ofrecerá diversas funcionalidades al usuario, mediante la comunicación con el servidor y con el bróker MQTT. Estas funcionalidades se comentarán mediante casos de uso en el siguiente punto. A continuación, se muestra la estructura del código de la aplicación móvil.

4.4.1 Estructura del código

En este subapartado se va a comentar en detalle la organización del código de la aplicación móvil. No se entrará en detalles de ficheros de código concretos, ya que para ello se encuentra el código debidamente comentado para explicar su contenido y funcionamiento. En la Ilustración 53 se muestra el árbol de directorios del código de la aplicación.

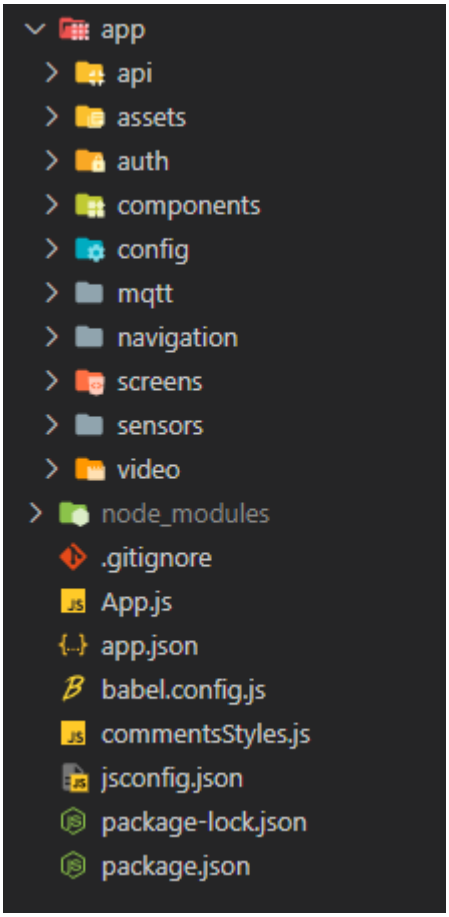

Ilustración 53 – Árbol de directorios del código de la aplicación móvil

Sobre esta imagen, vamos a comentar la forma de estructurar el código, aunque, antes de nada, dejar claro que hay algunos directorios y/o ficheros que no pertenecen propiamente al código de la aplicación, sino que forman parte del proyecto en Expo. Aquí hablaremos del contenido del directorio 'app', así como del fichero App.js, el cual contiene el código principal de la aplicación, que irá utilizando el del resto de ficheros.

El directorio app contiene a su vez una serie de subdirectorios, organizados debidamente para distinguir su función en la aplicación. Estos subdirectorios son:

- api: Contiene todos los ficheros relacionados con las APIs del servidor.
- assets: Contiene las imágenes que se utilizan en la aplicación, como el logo, imágenes de fondo, etc.
- auth: Contiene todos los ficheros relacionados con el proceso de autenticación en la propia aplicación.
- components: Contiene todos los componentes de React Native que se utilizarán en las pantallas y navegadores. Se encuentran estructurados en carpetas por funcionalidades.
- config: Carpeta de ficheros de configuración. Contiene ficheros cuyo contenido será solicitado en diversas partes de la aplicación.
- mqtt: Contiene todo el código relacionado con operaciones con la cola de mensajes MQTT.
- navigation: Contiene todo el código relacionado con la navegación dentro de la aplicación. Aquí se definen los navegadores, sus rutas, etc.
- screens: Contiene todas las pantallas que componen la aplicación. Los navegadores las mostrarán

según se organicen.

- sensors: Contiene todo el código relacionado con el uso de sensores, como la ubicación y el podómetro.
- video: Contiene un fichero de código encargado de dar formato a los links de vídeo de ejercicios al introducirlos en el formulario, antes de enviar los datos al servidor.

4.4.2 Estructura de la aplicación. Navegadores

En este subapartado se pretende que el lector tenga una idea clara de cómo se organizan las vistas en la aplicación móvil. Se irá apoyando la explicación en fragmentos del código que muestran las distintas partes que componen la aplicación.

4.4.2.1 NavigationContainer

Toda la aplicación se encuentra enmarcada en un contenedor de navegación, o 'NavigationContainer'. En su interior se ubicarán los navegadores encargados de regular el comportamiento de la aplicación a nivel general, y que a su vez incluirán el resto de navegadores. A continuación, se muestra un fragmento del fichero App.js:

```
return (
  <AuthContext.Provider value={{user, setUser}}>
    <NavigationContainer theme={navigationTheme}>
      {user ? <AppNavigator user={user}/> :  <AuthNavigator /> }
    </NavigationContainer>
  </AuthContext.Provider>
);
```

Ilustración 54 – Fragmento de código. NavigationContainer

Como se puede observar en la Ilustración 54, lo que devuelve la función App, función principal de la aplicación móvil, es un contexto de React personalizado que regula las funciones de autenticación, que enmarca el NavigationContainer. Éste, a su vez, engloba dos navegadores: AppNavigator y AuthNavigator. Aunque se verá su contenido en los subapartados siguientes, se puede apreciar que cuando el usuario se encuentre autenticado, se cargará el navegador AppNavigator, y en caso contrario se cargará el navegador AuthNavigator.

4.4.2.2 AuthNavigator

Este navegador se encarga de controlar el cambio de vistas de la aplicación para un usuario que no se encuentra autenticado aún. Se va a emplear la imagen de su estructura para explicar el código de los navegadores a nivel general, al ser el más sencillo de todos.

```
const AuthNavigator = () => (
    <Stack.Navigator>
        <Stack.Screen name="Welcome" component={WelcomeScreen} options={{headerShown: false}}/>
        <Stack.Screen name="Iniciar Sesión" component={LoginScreen} />
        <Stack.Screen name="Registrarse" component={RegisterScreen} />
    </Stack.Navigator>
)
```

Ilustración 55 – Fragmento de código. AuthNavigator

Se puede ver en la Ilustración 55 que un navegador no es más que una función que devuelve un Stack

Navigator (importado del módulo React Navigation). Éste a su vez estructura las diferentes vistas del navegador empleando las etiquetas 'Stack.Screen'. En cada una, se le asocia un nombre para poder desplazarse de una a otra empleando rutas, se define el componente asociado a cada vista, y se añaden diversas opciones de personalización.

4.4.2.3 AppNavigator

Este navegador se encarga de controlar el funcionamiento general de la aplicación cuando el usuario se encuentra autenticado. Engloba a los siguientes cuatro navegadores: AccountNavigator, UserNavigator, RoutineNavigator y WorkoutNavigator.

```
const Tab = createBottomTabNavigator();
```

Ilustración 56 – Fragmento de código. Creación del panel de navegación

Para organizar apropiadamente la navegación, AppNavigator define un navegador en la parte baja de la pantalla de nombre BottomTabNavigator, con botones para facilitar el desplazamiento, como se puede apreciar en la Ilustración 56.

```
const AppNavigator = ({user}) => (
    // Definimos el navegador, con su ruta inicial.
    <Tab.Navigator
        tabBarOptions={{
            activeBackgroundColor: colors.primary,
            activeTintColor: colors.white,
            inactiveBackgroundColor: colors.primary,
            inactiveTintColor: colors.lightprimary,
            style: styles.tab,
        }}
        initialRouteName="Yo"
    >
```

Ilustración 57 – Fragmento de código. Opciones del panel de navegación

En la Ilustración 57 se pueden apreciar las opciones de personalización del panel, así como la ruta inicial en la que arrancará el navegador al autenticarse el usuario.

```
<Tab.Screen
    name="Yo"
    component={AccountNavigator}
    options={{
        tabBarIcon: ({size, color}) => <MaterialCommunityIcons name="account" size={size} color={color}/>
    }}
/>
```

Ilustración 58 – Fragmento de código. Definición de botón en el panel y asociación de navegador

Por último, en la Ilustración 58 se puede apreciar cómo se definen los botones del panel. Cada uno de estos botones se asocia a un navegador de los mencionados, que se verán en detalle en los subapartados siguientes.

4.4.2.4 AccountNavigator

Este navegador, asociado al botón 'Yo' del panel de navegación definido en AppNavigator, se encarga de regular el cambio de vistas de esta sección de la aplicación. Es el navegador que se muestra por defecto al iniciar sesión correctamente en la aplicación, como se vio en el fragmento de código mostrado en la Ilustración 54.

Aunque su contenido se verá de forma más detallada en el punto siguiente del documento, agrupa las funcionalidades relacionadas con el propio usuario y su rol dentro del servicio, como puede ser modificar datos personales, ver historial de ejercicios (sólo para el Usuario), hacer seguimiento de actividad física (sólo para el Especialista), o cerrar sesión.

Además, aquí se da la opción de acceder a la pantalla de 'Acerca de', mostrando información sobre el servicio.

4.4.2.5 UserNavigator

Este navegador, asociado al botón 'Usuarios' del panel de navegación definido en AppNavigator, se encarga de regular el cambio de vistas de esta sección de la aplicación. Este navegador está oculto al rol Usuario, ya que no tiene acceso a las operaciones sobre éstos.

4.4.2.6 RoutineNavigator

Este navegador, asociado al botón 'Rutinas' del panel de navegación definido en AppNavigator, se encarga de regular el cambio de vistas de esta sección de la aplicación. Las funcionalidades son sutilmente distintas en algunas de las vistas para Usuario o Especialista. Se profundizará en esto en el punto siguiente.

4.4.2.7 WorkoutNavigator

Este navegador, asociado al botón 'Ejercicios' del panel de navegación definido en AppNavigator, se encarga de regular el cambio de vistas de esta sección de la aplicación. Al igual que ocurre con el navegador de Rutinas, este ofrece en sus vistas distintas funcionalidades para según qué rol de usuario acceda. Se verá más en detalle en el punto siguiente del documento.

4.4.2.8 Uso de rutas de navegación

Para el cambio de vistas en un mismo navegador, se utilizan rutas de navegación. Estas se encuentran definidas en un fichero, llamado routes.js, ubicado en el mismo directorio navigation, junto al resto de ficheros de navegadores.

En cada pantalla, si hay alguna funcionalidad definida que implica el cambio de pantalla, este cambio se hará utilizando la función mostrada en el fragmento de código de la Ilustración 59:

```
navigation.navigate(routes.MY_DETAILS, user);
```

Ilustración 59 – Fragmento de código. Método de navegación entre pantallas

La función acepta dos parámetros:

- Ruta de navegación: Accedida mediante el fichero routes.js antes mencionado.
- Parámetros de navegación: La navegación acepta parámetros englobados en un objeto. Este objeto irá contenido en la pantalla de destino en la 'prop' route, con el nombre de 'params'. Todo lo que se haya enviado a través de esta función se encontrará contenido en ese objeto.

Además de esta función de navegación, existen otras que no requieren del uso de rutas, ya que su desplazamiento entre pantallas es relativo a la pantalla de origen, como puede ser goBack(). Adicionalmente, existen funciones para reiniciar la pila de navegación, como reset(). Para más información, se puede consultar la documentación oficial de React Navigation, referenciada en el punto 2.3.3.2.

4.5 Casos de uso generales

En este apartado, veremos las distintas funcionalidades que implementa la aplicación, apoyadas en diagramas de casos de uso. Se ha organizado en tres subapartados: uno para la autenticación, que es común a todos los usuarios, otro para las operaciones de usuarios con rol 'Especialista', y otro para usuarios con rol 'Usuario'.

A la hora de distinguir entre los casos de uso de Especialista y Usuario, habrá casos de uso muy similares, denominados de igual forma en ambos subapartados. A la hora de comentarlos en la tabla correspondiente, tendrán el mismo identificador, diferenciados al final con las letras 'E' ó 'U', según corresponda a Especialista o Usuario, respectivamente.

4.5.1 Autenticación de un usuario

Este subapartado es común a cualquier rol de usuario del servicio. A continuación, se muestra el diagrama correspondiente a los casos de uso de autenticación en la Ilustración 60, y una serie de tablas explicativas de los procedimientos que se llevan a cabo en el sistema.

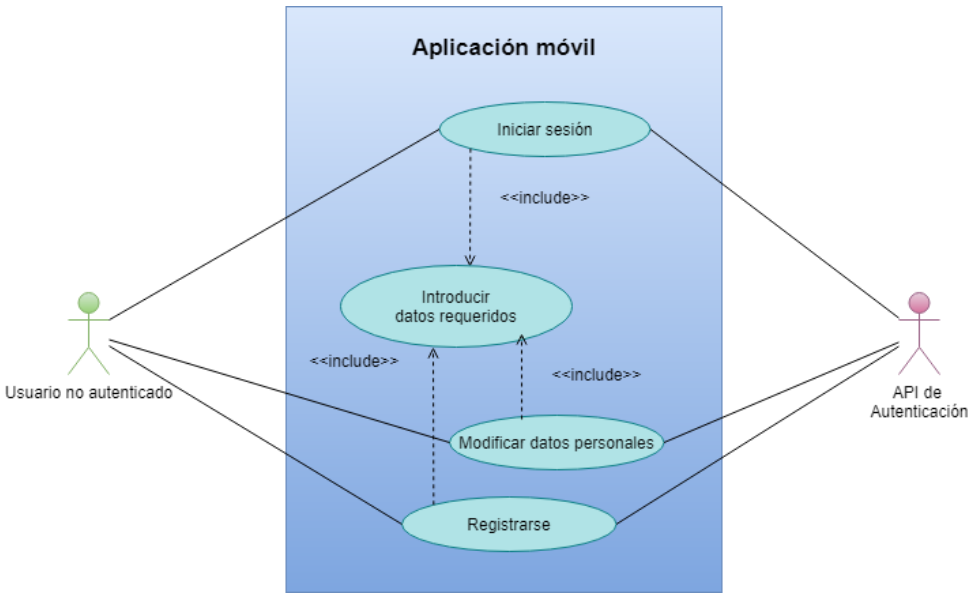

Ilustración 60 – Casos de Uso. Autenticación de un usuario

CU-01	**Iniciar sesión en la aplicación**		
Precondición	El usuario no se encuentra autenticado en el servicio, pero sí registrado.		
Descripción	El usuario proporciona la información que se le requiere mediante un formulario para autenticarse en el sistema y poder acceder a sus funcionalidades. La API de autenticación comprueba que los datos sean correctos, y devuelve un token al usuario.		
Secuencia Nominal	Paso	Acción	
	1	El usuario introduce usuario y contraseña en el formulario.	
	2	El usuario presiona el botón de iniciar sesión.	
	3	La API de autenticación comprueba que la información introducida es correcta.	
	4	La API de autenticación devuelve al usuario un token codificado mediante JWT.	
Postcondición	El usuario se encuentra autenticado en el servicio.		
Excepciones	Paso	Acción	
	1	La información introducida no concuerda con los formatos de validación del formulario.	
		E.1	La aplicación informa al usuario del error en el/los campos correspondientes.
		E.2	Se cancela el caso de uso.
	2	La API de autenticación resuelve que las credenciales son incorrectas.	
		E.1	Se informa al usuario de que los datos introducidos son erróneos.
		E.2	Se cancela el caso de uso.
Comentarios	Ninguno.		

Tabla 16 – CU-01. Iniciar sesión en la aplicación

CU-02	**Registrarse en la aplicación**		
Precondición	El usuario no se encuentra registrado ni autenticado en el servicio.		
Descripción	El usuario proporciona la información que se le requiere mediante un formulario para registrarse en el sistema y poder iniciar sesión posteriormente. La API de autenticación trata de crear un nuevo usuario en la base de datos, informando al usuario del resultado.		
Secuencia Nominal	Paso	Acción	
	1	El usuario introduce los datos requeridos en el formulario.	
	2	El usuario presiona el botón de registrarse.	
	3	La API de autenticación se encarga de crear el nuevo usuario en el sistema.	
	4	La API de autenticación informa al usuario del resultado del registro.	
Postcondición	El usuario se ha registrado correctamente en el sistema.		
Excepciones	Paso	Acción	
	1	La información introducida no concuerda con los formatos de validación del formulario.	
		E.1	Se informa al usuario del error en el/los campos correspondientes.
		E.2	Se cancela el caso de uso.
	3	La API de autenticación resuelve que ya se encuentra registrado un usuario con esa dirección de correo electrónico.	
		E.1	Se informa al usuario de que el correo electrónico ya se corresponde con un usuario en el sistema.
		E.1	Se cancela el caso de uso.
Comentarios	Ninguno.		

Tabla 17 – CU-02. Registrarse en la aplicación

CU-03	**Modificar datos personales del usuario**		
Precondición	El usuario se encuentra debidamente registrado y autenticado.		
Descripción	El usuario proporciona la información que desea modificar mediante un formulario, inicialmente relleno con los datos actuales. La API de autenticación modifica la información en la base de datos, informando al usuario del resultado.		
Secuencia Nominal	Paso	Acción	
	1	El usuario modifica los datos deseados en el formulario.	
	2	El usuario presiona el botón de modificar datos.	
	3	La API de autenticación se encarga de actualizar los datos del usuario en el sistema.	
	4	La API de autenticación informa al usuario del resultado del proceso.	
Postcondición	El usuario ha actualizado sus datos correctamente en el sistema.		
Excepciones	Paso	Acción	
	1	La información introducida no concuerda con los formatos de validación del formulario.	
		E.1	Se informa al usuario del error en el/los campos correspondientes.
		E.2	Se cancela el caso de uso.
	3	La API de autenticación encuentra un error al actualizar los datos.	
		E.1	Se informa al usuario de que no se ha podido modificar los datos en el sistema.
		E.1	Se cancela el caso de uso.
Comentarios	Ninguno.		

Tabla 18 – CU-03. Modificar datos personales del usuario

4.5.2 Operaciones de un usuario con rol 'Especialista'

En este subapartado veremos con casos de uso las distintas operaciones que puede realizar un usuario con rol 'Especialista'. A continuación, se muestra en la Ilustración 61 un diagrama de casos de uso de las operaciones que lleva a cabo un usuario con rol Especialista. Seguidamente, se mostrará una serie de tablas, recogiendo cada uno de esos casos de uso.

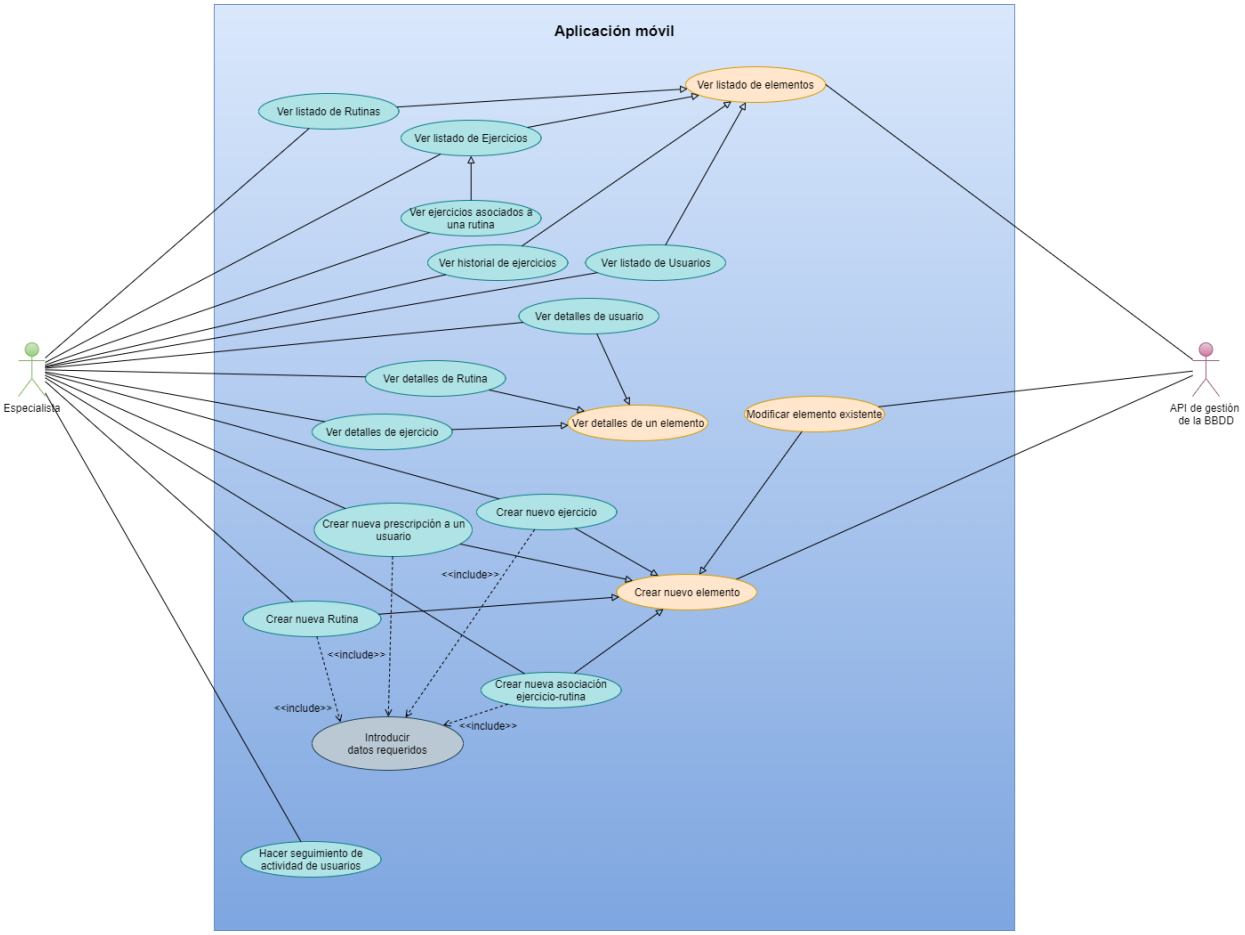

Ilustración 61 – Casos de Uso. Operaciones de un Especialista

CU-04-E	**Ver listado de elementos [rutina, ejercicio, ejercicios asociados a una rutina, usuarios o historial de ejercicios de un usuario]**	
Precondición	El especialista debe estar autenticado.	
Descripción	El especialista solicita al servidor un listado de elementos (rutinas, ejercicios, ejercicios asociados a una rutina, usuarios, o historial de ejercicios de un usuario) que existen en el sistema.	
Secuencia Nominal	Paso	Acción
	1	La aplicación solicita la información sobre el elemento deseado teniendo en cuenta el rol de 'Especialista'.
	2	El servidor verifica el token de la petición.
	3	El servidor solicita a la base de datos la información sobre el elemento solicitado.
	4	El servidor devuelve el listado extraído.
Postcondición	El especialista puede ver el listado solicitado en la pantalla.	
Excepciones	Paso	Acción
	2-A	El token no es verificado correctamente.
		E.1 Se informa al usuario del error.
		E.2 Se cancela el caso de uso.
	2-B	El rol encontrado en el token no se corresponde con 'Especialista'.
		E.1 Se informa al usuario del error.
		E.2 Se cancela el caso de uso.
	4	No se han encontrado elementos de tipo solicitado en el servidor.
		E.1 Se informa al usuario.
		E.2 Se cancela el caso de uso.
Comentarios	En el paso 1 de la secuencia nominal, la ruta a la que se hace la petición a la API varía en función del rol del usuario autenticado, para las solicitudes de ejercicios o rutinas. Así, para el rol 'Usuario', el resto del procedimiento será distinto. Como todos los casos de uso de las solicitudes de listados de elementos son muy parecidos, se ha recogido todo en una misma tabla.	

Tabla 19 – CU-04-E. Ver listado de elementos (Especialista)

CU-05	**Crear -o modificar- elemento** **[rutina, ejercicio, asociación ejercicio-rutina, prescripción]**	
Precondición	El especialista debe estar autenticado.	
Descripción	El especialista introduce los datos en el formulario	
Secuencia Nominal	Paso	Acción
	1	El especialista introduce los datos requeridos en el formulario.
	2	El especialista pulsa el botón de crear -o modificar- y envía los datos al servidor.
	3	El servidor procesa la petición y trata de crear o modificar el elemento.
	4	El servidor informa a la aplicación del resultado.
Postcondición	El especialista ha creado -o modificado- con éxito el elemento.	
Excepciones	Paso	Acción
	2	Los datos introducidos no cumplen con el formato de validación del formulario antes de enviarse.
		E.1 La aplicación muestra mensaje de error para el/los campos con mal formato.
		E.2 Se cancela el caso de uso.
	3-A	El token de la petición no es correcto, o el rol no es Especialista
		E.1 El servidor informa al especialista.
		E.2 Se cancela el caso de uso.
	3-B	Hay un error a la hora de crear o modificar el elemento.
		E.1 El servidor informa al especialista.
		E.2 Se cancela el caso de uso.
Comentarios	En el paso 2 de la secuencia nominal, la ruta a la que se hace la petición varía en función del elemento a crear. Como todos los casos de uso de crear o modificar elementos son muy parecidos, se ha recogido todo en una tabla.	

Tabla 20 – CU-05. Crear o modificar elemento

CU-06	Hacer seguimiento de actividad de usuarios.		
Precondición	El especialista debe estar autenticado.		
Descripción	El especialista accede a la vista de seguimiento en directo, y la aplicación se suscribe a la cola MQTT asociada al topic definido por el correo electrónico del propio especialista.		
Secuencia Nominal	Paso	Acción	
	1	El especialista accede a la vista de seguimiento en directo.	
	2	La aplicación se conecta y suscribe a la cola MQTT, en el topic definido por el email del especialista.	
Postcondición	El especialista puede ver un listado que se actualiza automáticamente de los usuarios que están realizando un ejercicio.		
Excepciones	Paso	Acción	
	2	Hay un error en la conexión o suscripción.	
		E.1	La aplicación muestra mensaje de error.
		E.2	Se cancela el caso de uso.
Comentarios	Ninguno.		

Tabla 21 – CU-06. Hacer seguimiento de actividad de usuarios

4.5.3 Operaciones de un usuario con rol 'Usuario'

En este subapartado se comentarán en detalle todas las operaciones realizadas por un usuario con rol 'Usuario'. A continuación, se verá en la Ilustración 62 un diagrama de casos de uso con dichas operaciones, y seguidamente, una serie de tablas que comentarán en detalle cada una de estas operaciones.

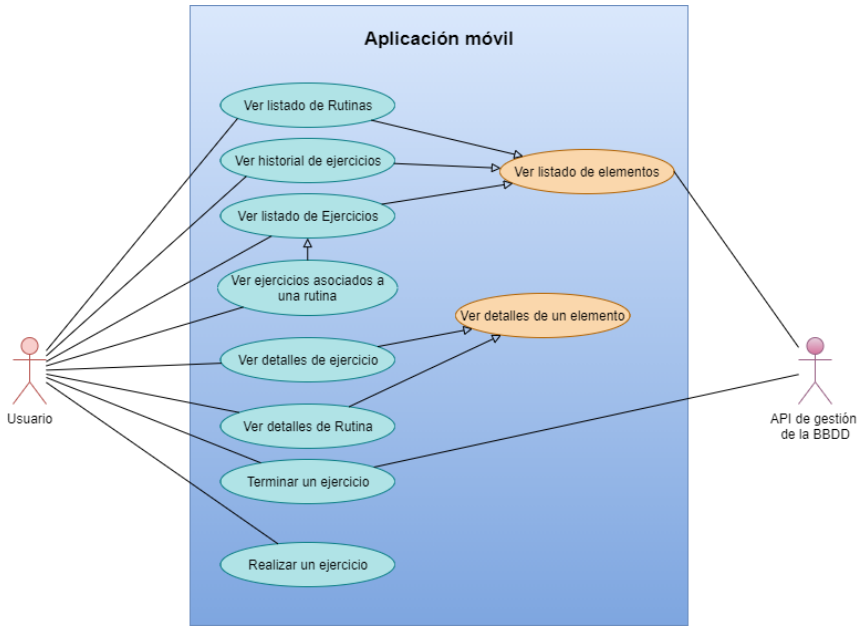

Ilustración 62 – Casos de Uso. Operaciones de un Usuario

CU-04-U	**Ver listado de elementos [rutina, ejercicio, ejercicios asociados a una rutina o historial de ejercicios del usuario]**	
Precondición	El usuario debe estar autenticado.	
Descripción	El usuario solicita al servidor un listado de elementos (rutinas prescritas, ejercicios prescritos, ejercicios asociados a una rutina prescrita, o historial de ejercicios del usuario) del sistema.	
Secuencia Nominal	Paso	Acción
	1	La aplicación solicita la información sobre el elemento deseado teniendo en cuenta el rol de 'Usuario'.
	2	El servidor verifica el token de la petición.
	3	El servidor solicita a la base de datos la información sobre el elemento solicitado.
	4	El servidor devuelve el listado extraído.
Postcondición	El usuario puede ver el listado solicitado en la pantalla.	
Excepciones	Paso	Acción
	2	El token no es verificado correctamente.
		E.1 Se informa al usuario del error.
		E.2 Se cancela el caso de uso.
	4	No se han encontrado elementos de tipo solicitado en el servidor.
		E.1 Se informa al usuario.

		E.2	Se cancela el caso de uso.
Comentarios	En el paso 1 de la secuencia nominal, la ruta a la que se hace la petición a la API varía en función del rol del usuario autenticado, para las solicitudes de ejercicios o rutinas. Así, para el rol 'Usuario', el resto del procedimiento será distinto. En la verificación del token, en el CU-04-E, se comprobaba también el rol. En este caso, como las rutas son distintas, no se hace esta comprobación. Como todos los casos de uso de las solicitudes de listados de elementos son muy parecidos, se ha recogido todo en una misma tabla.		

Tabla 22 – CU-04-U. Ver listado de elementos (Usuario)

CU-07	**Comenzar un ejercicio**		
Precondición	El usuario debe estar debidamente autenticado.		
Descripción	El usuario comienza a realizar un ejercicio. Según corresponda, ciertos sensores serán leídos, y se publicará en la cola MQTT información periódica acerca del desarrollo del ejercicio.		
Secuencia Nominal	Paso	Acción	
	1	El usuario pulsa el botón de comenzar ejercicio.	
	2	La aplicación pone en marcha el temporizador y se sucribe a actualizaciones de los sensores correspondientes.	
	3	La aplicación se conecta a la cola MQTT con topic definido por el correo electrónico del especialista que prescribió el ejercicio al usuario para publicar información periódicamente.	
Postcondición	El usuario se encuentra realizando el ejercicio.		
Excepciones	Paso	Acción	
	2	No se han otorgado los permisos correspondientes.	
		E.1	No se puede llevar a cabo la recolección de información de sensores.
		E.2	Se cancela el caso de uso.
	3	Hay un error al conectar a la cola MQTT.	
		E.1	Se informa al usuario del error.
		E.2	Se cancela el caso de uso.
Comentarios	Ninguno.		

Tabla 23 – CU-07. Comenzar un ejercicio

CU-08	Terminar un ejercicio	
Precondición	El usuario debe estar debidamente autenticado, y encontrarse realizando un ejercicio (CU-06).	
Descripción	El usuario termina la realización del ejercicio. Se cierra la suscripción a información de sensores, se cierra la conexión con la cola MQTT, y se envía la información recopilada al servidor mediante la API.	
Secuencia Nominal	Paso	Acción
	1	El usuario pulsa el botón de terminar ejercicio.
	2	La aplicación envía el último mensaje a través de la cola MQTT con la información más reciente y cierra la conexión.
	3	La aplicación se desuscribe de la actualización de información de sensores.
	4	La aplicación envía la información recopilada a la API de la base de datos para añadirla al historial de ejercicios del usuario.
	5	El servidor procesa la petición y crea una nueva entrada en el historial.
	6	La aplicación cierra la pantalla de ejercicio en proceso.
Postcondición	El usuario ya no se encuentra realizando un ejercicio.	
Excepciones	Paso	Acción
	5	Hay un error a la hora de crear la entrada.
		E.1 No se puede almacenar la información en la base de datos.
Comentarios	En la excepción al paso 5 de la secuencia nominal, el caso de uso no se cancela ya que no se tiene previsto un comportamiento alternativo en caso de que no se almacene la información.	

Tabla 24 – CU-08. Finalizar un ejercicio

4.6 Diagramas de secuencia

Con este apartado se pretende complementar las funcionalidades del servicio descritas mediante los casos de uso en el apartado anterior. En este caso, haciendo uso de diagramas de secuencia, se verán en detalle todas las partes que intervienen en los procesos de dichas funcionalidades.

En cada subapartado se especificará, si procede, qué rol de usuario interviene como actor, además de enumerar y comentar cada uno de los módulos que participan en la secuencia descrita por el diagrama.

4.6.1 Registrar un usuario

En el siguiente diagrama, mostrado en la Ilustración 63, se detalla la sucesión de interacciones que tienen lugar

durante el proceso de registro de un usuario no autenticado previamente. Los elementos que participan en este proceso, reflejados en el encabezado del diagrama, son los siguientes:

- Usuario: Persona que hace uso de la aplicación. Al no estar autenticado, no dispone de un rol concreto, por lo que aquí la palabra 'Usuario' hace mención a un usuario genérico de la aplicación, y no al rol en sí.
- App: Aplicación móvil. Mostrará las interacciones con el servidor y consigo misma -entrada y salida por pantalla- para tratar de obtener la información solicitada por el usuario, así como informar del resultado de la operación.
- Authentication: Módulo del servidor encargado de la parte de autenticación. Se encarga de atender las peticiones de este tipo, y a su vez hará uso de la API de datos para interactuar con la base de datos.
- DataAPI: API de datos del servidor. Sirve de intermediaria con la base de datos, y se encargará de solicitar la información requerida, y devolver el resultado.
- Database: Base de datos del sistema. Almacena toda la información que puede ser requerida por el servicio. Además, será donde se almacene el usuario nuevo a crear en el sistema, si no ha habido errores.

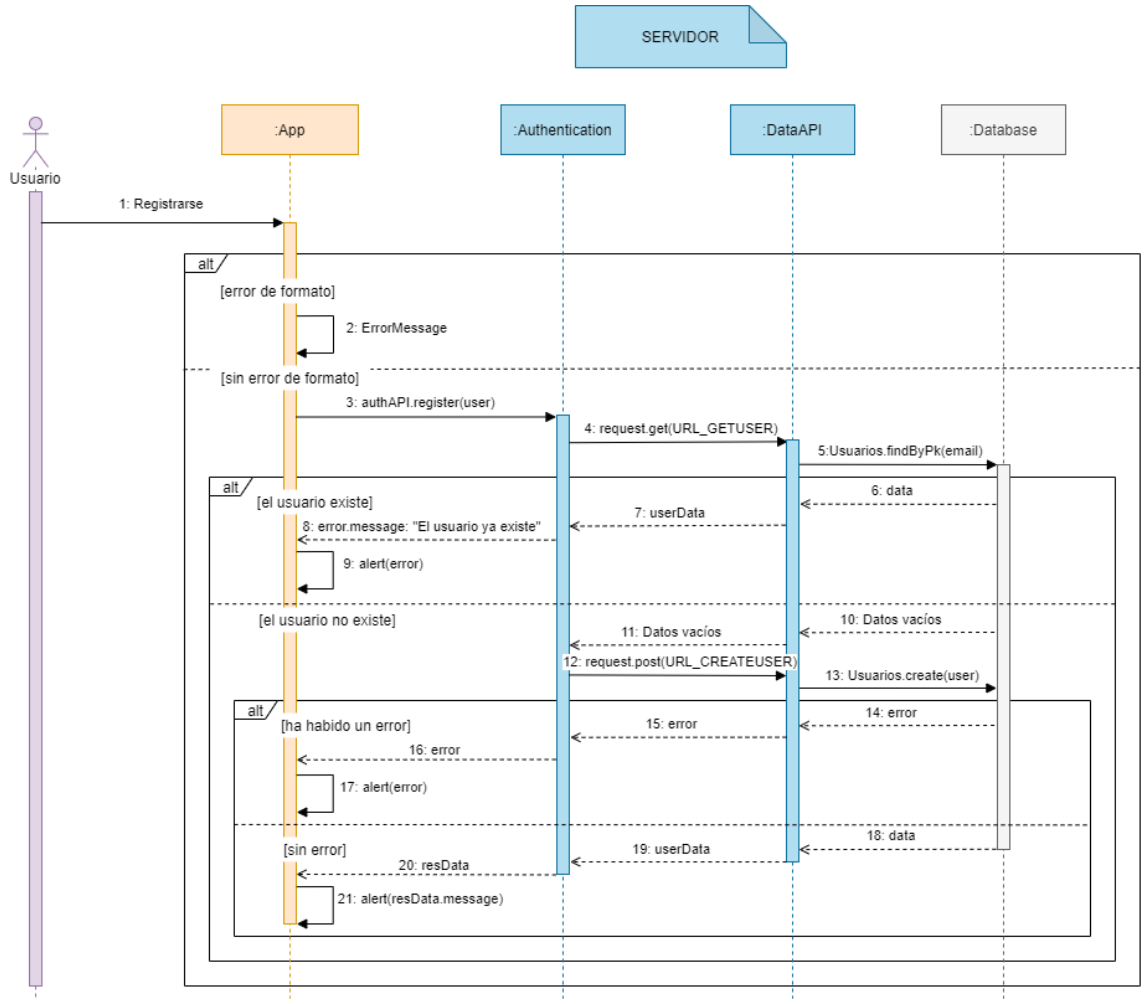

Ilustración 63 – Diagrama de secuencia. Registro de un usuario

4.6.2 Prescribir una rutina o ejercicio (Especialista)

En este caso se va a representar, bajo un único diagrama en la Ilustración 64, la prescripción tanto de ejercicios como de rutinas de forma indiferente, ya que la secuencia es prácticamente la misma. Los elementos que intervienen en el diagrama son los siguientes:

- Especialista: Usuario de la aplicación, debidamente autenticado, con rol 'Especialista'. Interviene al introducir los datos y pulsar en 'Enviar'.
- App: Aplicación móvil del servicio. Mostrará interacción con el servidor y consigo misma, en la comprobación de campos del formulario, por ejemplo. Se encargará de recoger por pantalla la información enviada por el usuario y de transmitirla al servidor.
- DataApi: API de Datos de la aplicación web. Escuchará en ciertas rutas para interactuar con la base de datos haciendo de intermediaria entre el cliente (aplicación móvil) y la base de datos.
- Database: Base de datos del sistema. Almacena toda la información que puede ser requerida por el servicio. Además, será donde se almacene el usuario nuevo a crear en el sistema, si no ha habido errores.

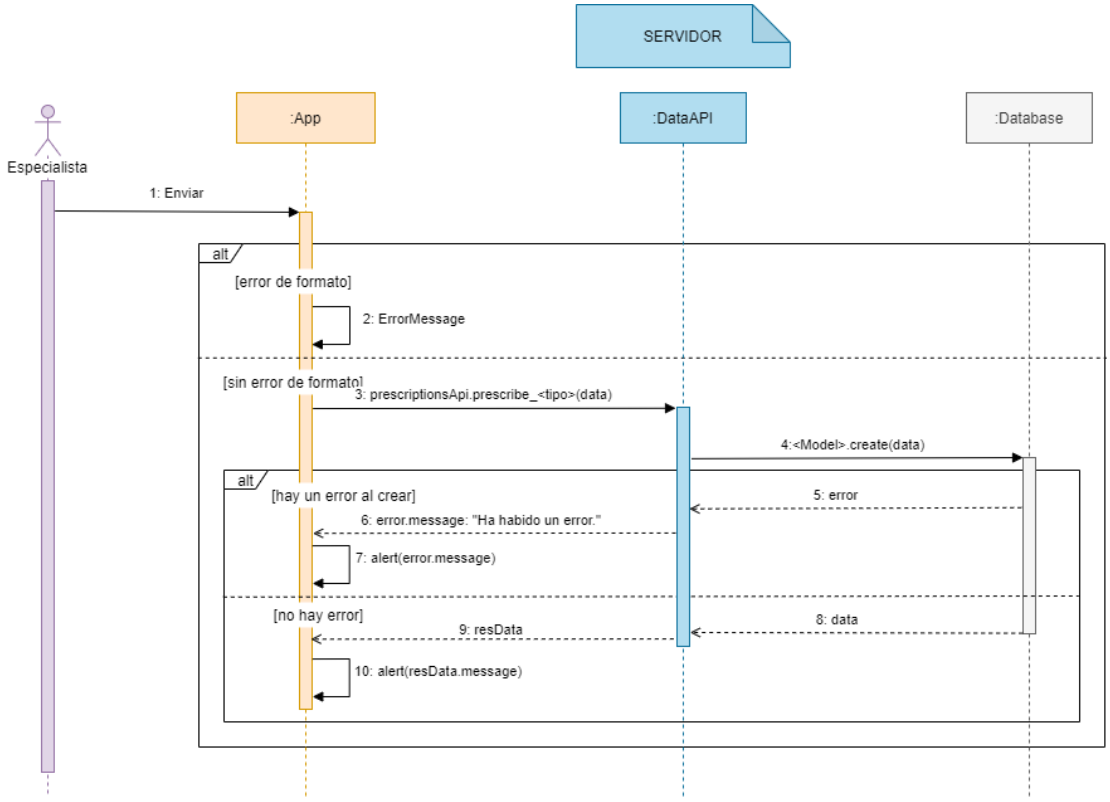

Ilustración 64 – Diagrama de secuencia. Prescribir a un usuario

A continuación, analizaremos las diferencias entre la prescripción de ejercicio y la de rutina. Estas diferencias tan sólo se verían reflejadas en el diagrama en los siguientes mensajes:

- Mensaje 3: En el lugar que aparece <*tipo*>, deberá ser 'UR' (UserRoutine) para prescribir rutinas, o bien 'UW' (UserWorkout) para prescribir ejercicios.
- Mensaje 4: En el lugar que aparece <*Model*>, deberá ser 'Usuario_has_Rutina' para prescribir rutinas, o 'Usuario_has_Ejercicio' para prescribir ejercicios.

Nota: Este diagrama también refleja el funcionamiento de la funcionalidad 'Modificar una prescripción'. La diferencia con esto serían igualmente los mensajes 3 y 4, en los que los métodos serían *update<Workout ó Routine>fromUser*, y *update*, respectivamente.

4.6.3 Asociar ejercicio a rutina (Especialista)

En este subapartado se verá el diagrama de secuencia en la Ilustración 65 que describe el proceso de asociación de un ejercicio a una rutina. Aunque se ha decidido incluirlo, su parecido es sensiblemente alto con el diagrama del punto anterior. Los elementos que intervienen son los mismos que en el caso del punto

anterior, indicados para la Ilustración 64.

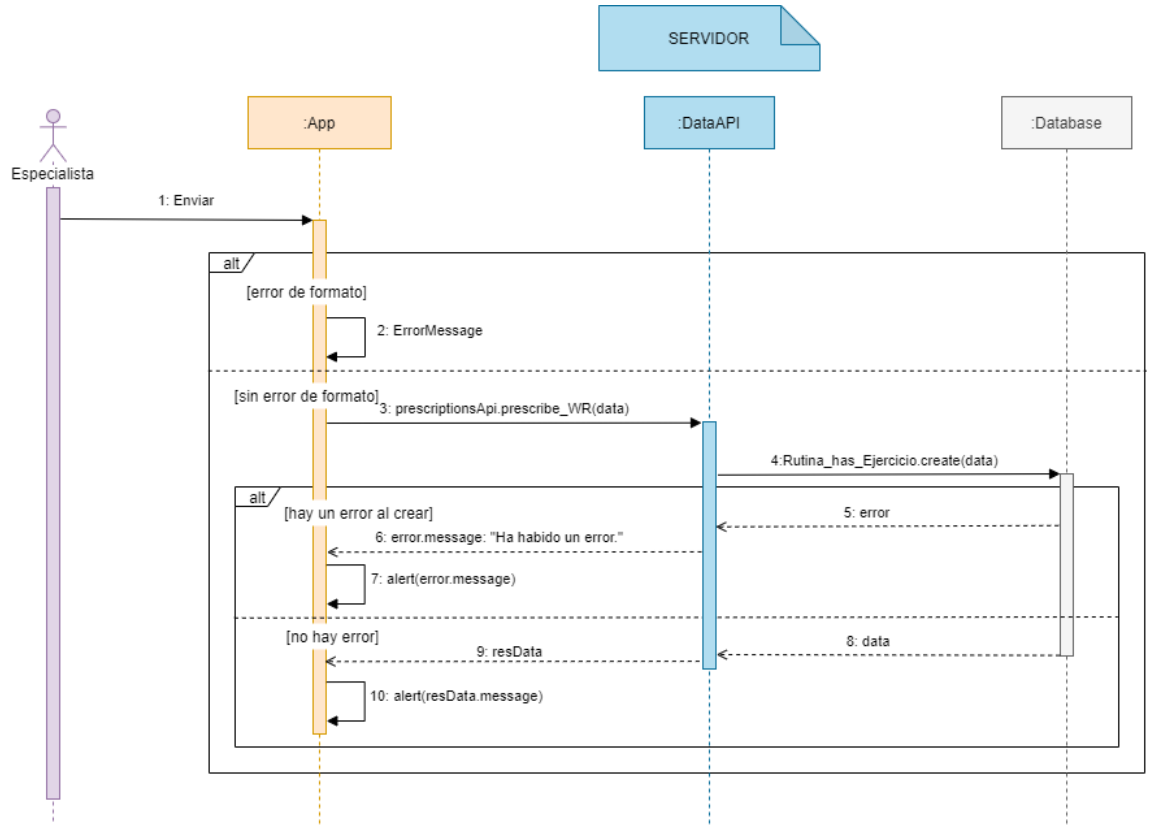

Ilustración 65 – Diagrama de secuencia. Asociación de ejercicio a rutina

Nota: Este diagrama también refleja el funcionamiento de la funcionalidad 'Modificar una asociación'. La diferencia con esto serían los mensajes 3 y 4, en los que los métodos serían *updateWorkoutfromRoutine* y *update*, respectivamente.

4.6.4 Listar elementos

En este punto se mostrará mediante un diagrama en la Ilustración 66 todo el proceso de listado de elementos en la aplicación móvil. Aquí se recoge el listado de elementos de cualquier tipo, ya sean usuarios -exclusivo de Especialistas-, rutinas, ejercicios, prescripciones, asociaciones ejercicio-rutina, o datos de historial de ejercicios realizados. El motivo de esto es que de forma general el comportamiento del sistema es muy similar, al ser los detalles más concretos los que marcan la diferencias en algunos aspectos.

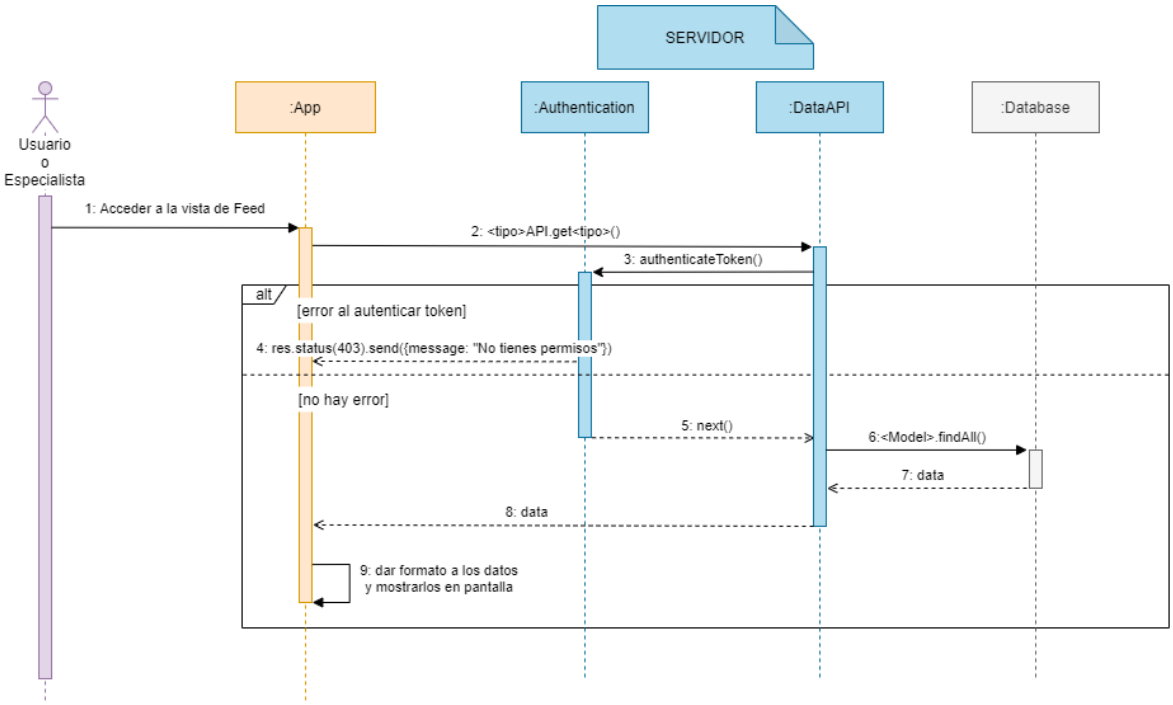

Ilustración 66 – Diagrama de secuencia. Listar elementos

Los elementos que intervienen en el diagrama son los siguientes:

- Usuario o Especialista: Usuario de la aplicación, debidamente autenticado. El rol del usuario en la App determinará el carácter de la información devuelta en algunos casos.
- App: Aplicación móvil. Mostrará las interacciones con el servidor y consigo misma -entrada y salida por pantalla- para tratar de obtener la información solicitada por el usuario, así como informar del resultado de la operación.
- Authentication: Módulo del servidor encargado de la parte de autenticación. Se encarga de atender las peticiones de este tipo. En este caso ofrece el middleware 'authenticateToken' para verificar el usuario que realiza la petición.
- DataAPI: API de datos del servidor. Sirve de intermediaria con la base de datos, y se encargará de solicitar la información requerida, y devolver el resultado.
- Database: Base de datos del sistema. Almacena toda la información que puede ser requerida por el servicio.

4.6.5 Realizar un ejercicio (Usuario)

En este punto se va a reflejar en el diagrama toda la sucesión de interacciones desde que un usuario inicia un ejercicio, hasta que lo finaliza. Esto se muestra en la Ilustración 67:

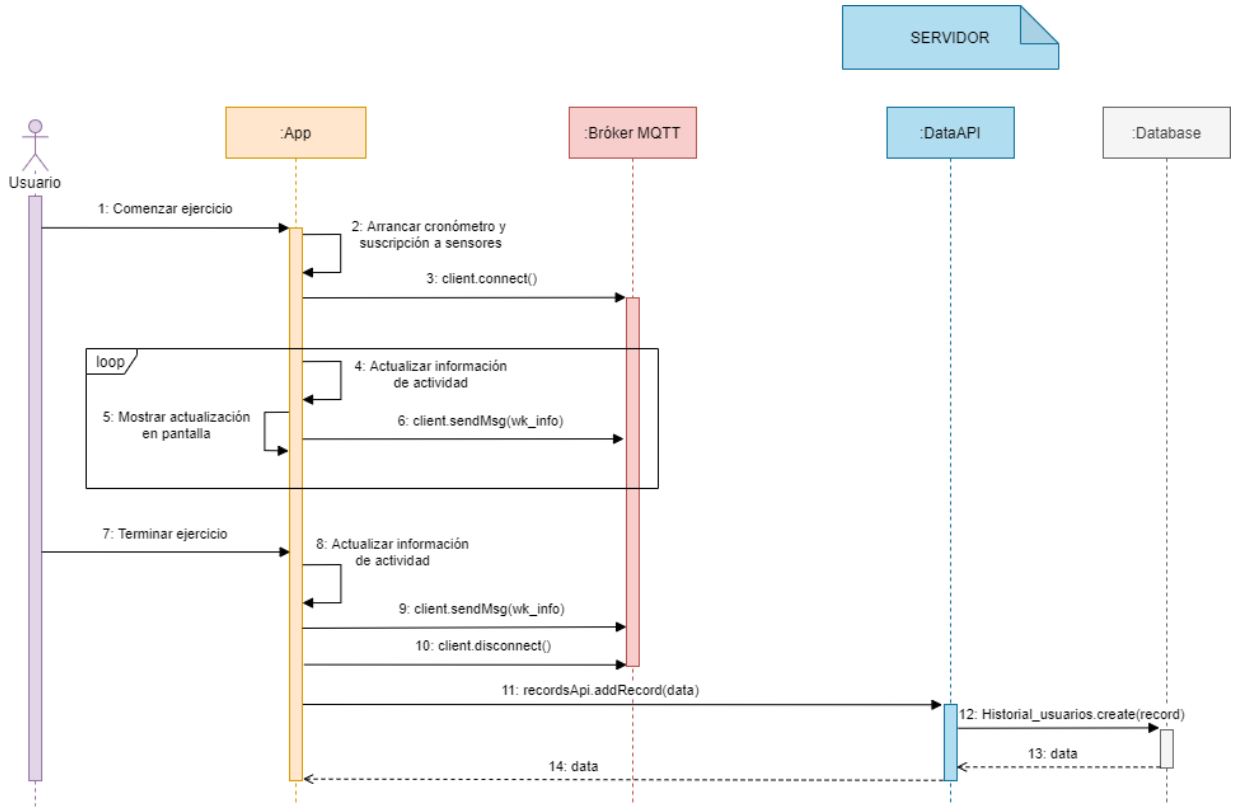

Ilustración 67 – Diagrama de secuencia. Realizar un ejercicio

Los elementos que intervienen en este diagrama son los siguientes:

- Usuario: Usuario de la aplicación, debidamente autenticado, con rol 'Usuario'.
- App: Aplicación móvil. Se encargará de conectarse a la cola MQTT para publicar información de actividad física de forma periódica. Al finalizar el ejercicio, llamará a la API de Datos para almacenar la información recopilada.
- Bróker MQTT: Servicio de terceros, encargado de coordinar la cola de mensajes. Será receptor de los mensajes publicados, y los redirigirá a los suscriptores. En esta propuesta el servicio es alojado por HiveMQ.
- DataAPI: API de datos del servidor. Sirve de intermediaria con la base de datos. Procesará la petición de almacenamiento de información e informará del resultado.
- Database: Base de datos del sistema. Almacena toda la información que puede ser requerida por el servicio. En este caso, alojará la información enviada por la aplicación móvil.

4.6.6 Hacer un seguimiento de los Usuarios que están realizando ejercicios (Especialista)

En este punto, complementario al anterior, ya que ambas secuencias pueden ocurrir a la vez y muchos de los elementos implicados son los mismos, se comentará la actividad de realizar un ejercicio bajo el punto de vista del Especialista que realiza el seguimiento. Para ello, se muestra el correspondiente diagrama en la Ilustración 68. A continuación, se indican los elementos que intervienen:

- Especialista: Usuario de la aplicación, debidamente autenticado, con rol 'Especialista'.
- App: Aplicación móvil. Se encargará de suscribirse a la cola MQTT para recibir información de actividad física de los usuarios que se encuentren realizando ejercicios.
- Bróker MQTT: Servicio de terceros, encargado de coordinar la cola de mensajes. Será receptor de los mensajes publicados, y los redirigirá a los suscriptores. En esta propuesta el servicio es alojado por HiveMQ.

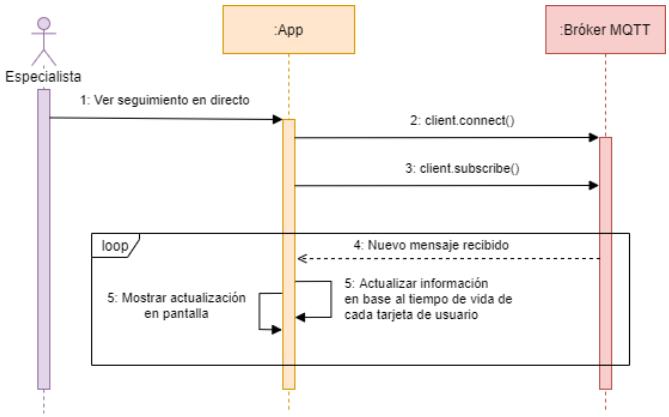

Ilustración 68 – Diagrama de secuencia. Hacer un seguimiento de actividad física en directo

5 INTERFAZ DE USUARIO

La gente ignora el diseño que ignora a la gente.

- Frank Chimero -

En este punto del documento se van a mostrar las distintas vistas que componen la interfaz de usuario, es decir, la aplicación móvil del servicio propuesto. Para ello, se va a diferenciar por apartados según cada navegador, esto es, según cada bloque de funcionalidades diferenciado. Estos bloques son los siguientes:

- Registro e Inicio de sesión
- Vistas del panel de navegación:
    - Mi cuenta
    - Usuarios
    - Rutinas
    - Ejercicios

Estos bloques tienen su paralelismo en la estructura de los navegadores de la aplicación, explicada en el punto 4.4.2 "*Estructura de la aplicación. Navegadores*" del documento. En cada bloque se detallarán todas las pantallas de las vistas que los componen.

Nota: En algunas de las pantallas de la interfaz de usuario de la aplicación se muestran formularios. En algunos de estos formularios, aparecerán campos sombreados con un tono gris más oscuro que el resto de campos. Esto indica que el campo no se puede modificar.

5.1 Registro e Inicio de Sesión

Este apartado recoge las vistas de la aplicación destinadas a permitir al usuario iniciar sesión o registrarse, así como una pantalla de bienvenida.

5.1.1 Pantalla de bienvenida

La pantalla de bienvenida es la primera que se muestra a un usuario no autenticado al acceder a la aplicación. Muestra el logotipo del servicio, dos botones para iniciar sesión o registrarse, y una imagen de fondo distorsionada.

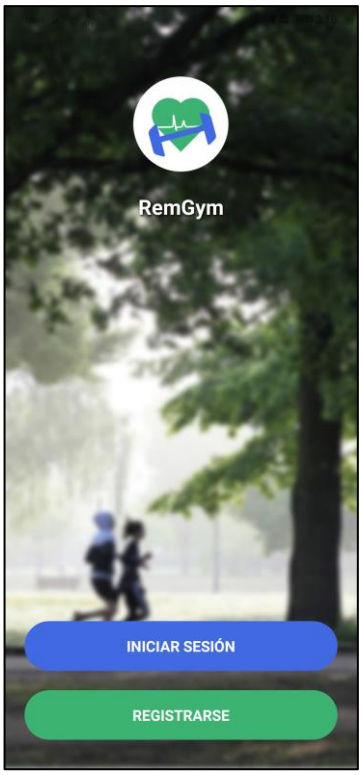

Ilustración 69 – Pantalla de bienvenida

5.1.2 Pantalla de inicio de sesión

Esta pantalla consta de un formulario en el que se solicita al usuario correo electrónico y contraseña para acceder al servicio.

Ilustración 70 – Pantalla de inicio de sesión

5.1.3 Pantalla de registro

En esta pantalla se muestra un formulario que el usuario deberá cumplimentar con el formato requerido para registrarse correctamente.

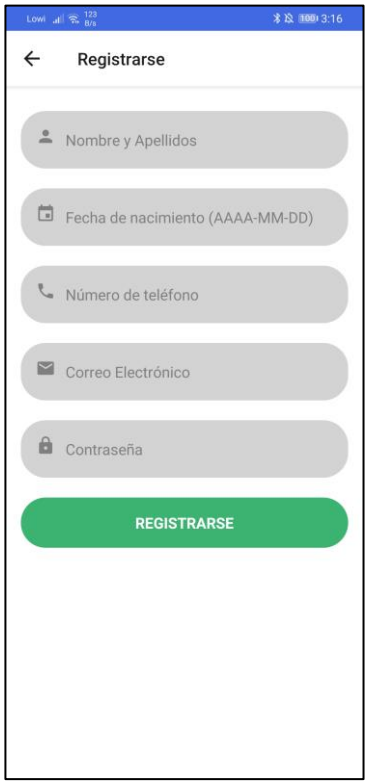

Ilustración 71 – Pantalla de registro

5.2 Navegación

Una vez autenticado el usuario, se le mostrará un navegador en la parte inferior de la pantalla. Con él, puede desplazarse entre las distintas vistas y actuar sobre ellas según las funcionalidades descritas en el punto anterior del documento. Este panel de navegación, así como el aspecto de según qué pantallas, será distinto para los dos tipos de roles de usuario en la aplicación. En las Ilustraciones 72 y 73 se muestran los paneles de navegación de Especialista y Usuario, respectivamente.

Ilustración 72 – Panel de navegación de Especialista

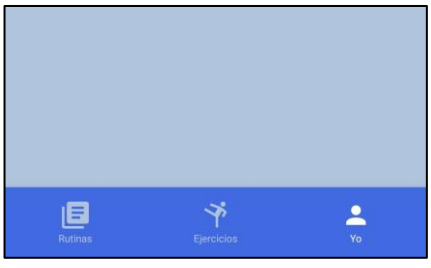

Ilustración 73 – Panel de navegación del Usuario

Nota: Los distintos navegadores tienen ‘memoria’. Esto quiere decir que la pantalla que muestren no desaparece al cambiar de navegador en el panel inferior. El diseño de la aplicación se ha hecho aprovechando esta idea, para que, al crear un nuevo elemento en algún navegador, el Especialista se pueda mover por el resto de la aplicación dejando en espera el formulario, si tiene que buscar información para cumplimentarlo.

5.3 Navegador ‘Mi cuenta’

Esta pantalla es la primera que aparece en la aplicación tras iniciar sesión en el sistema. Se accede también al pulsar la opción ‘Yo’ en el panel de navegación. Muestra un listado de opciones que seleccionar, que dirigirán a las pantallas mostradas en los subapartados siguientes. Las opciones varían respecto el rol del usuario autenticado. En las Ilustraciones 74 y 75 se muestran las diferencias:

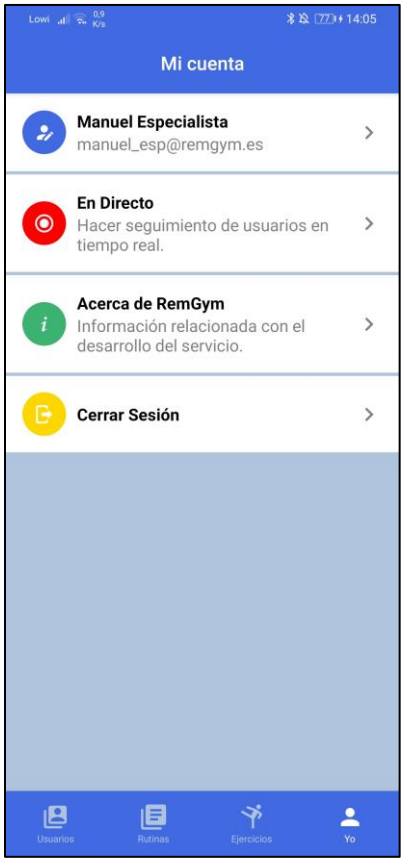

Ilustración 74 – Pantalla ‘Mi cuenta’. Especialista

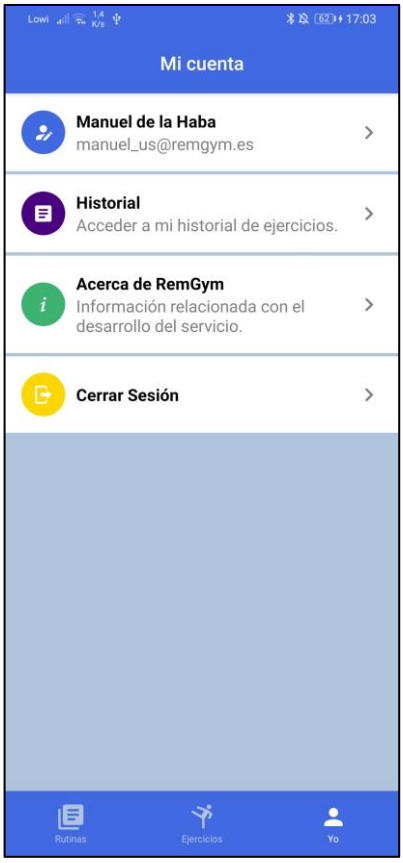

Ilustración 75 – Pantalla ‘Mi cuenta’. Usuario

Cada una de estas opciones mostradas redirigirán la aplicación a una nueva pantalla. La opción de Cerrar Sesión volverá a la pantalla de bienvenida.

5.3.1 Modificar datos personales

A esta pantalla se accede pulsando el primer elemento del listado de la pantalla “Mi cuenta”, donde aparece el nombre del usuario y su dirección de correo electrónico. En esta pantalla se muestra un formulario relleno con los actuales datos personales del usuario. El objetivo de esto es que el usuario pueda modificar cualquier dato que desee -excepto el correo electrónico- y actualizar la información en el sistema. Tan sólo hay que seleccionar el campo a modificar, y cambiar su valor. Se muestra en la Ilustración 76.

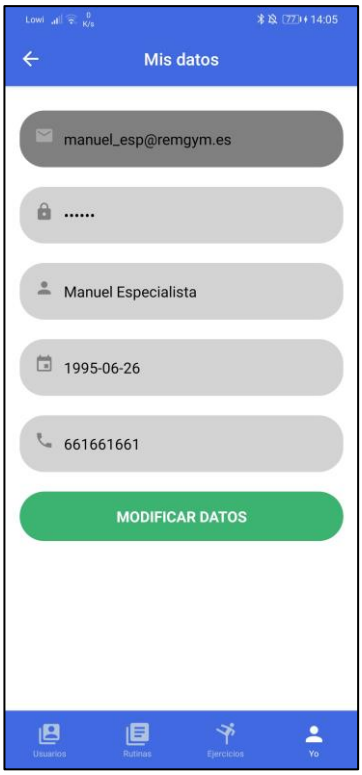

Ilustración 76 – Pantalla de modificar datos personales

5.3.2 Seguimiento en directo (Especialista)

En esta pantalla, por defecto con un listado vacío, se irá mostrando la información de aquellos usuarios que se encuentren realizando ejercicio en ese momento. Al pulsar sobre el elemento correspondiente a uno de los usuarios, se ampliará la tarjeta mostrando información adicional, si existe. Esta pantalla se muestra en la Ilustración 77.

Ilustración 77 – Pantalla de seguimiento de actividad en directo

5.3.3 Ver historial de ejercicios (Usuario)

Esta pantalla, mostrada en la Ilustración 78, permite al usuario acceder al histórico de información de ejercicios realizados. Cada tarjeta será un ejercicio realizado en un momento concreto. Al seleccionarla, se ampliará mostrando información adicional, si existe.

Ilustración 78 – Pantalla de Historial de ejercicios

Nota: A esta pantalla también se accede desde el rol 'Especialista'. En el punto 5.4.2 se indicará cómo.

5.3.4 Acerca de gymApp

Esta pantalla existe con carácter informativo acerca del proyecto. Muestra información sobre el motivo del desarrollo y el autor, entre otros. A continuación, se muestra una captura de esta pantalla, en la Ilustración 79:

Ilustración 79 – Pantalla 'Acerca de'

5.4 Navegador 'Usuarios' (Especialista)

Este navegador es accedido al presionar la opción 'Usuarios' en el panel de navegación (Ilustración 72). A continuación, veremos en cada subapartado las diferentes pantallas que son accesibles mediante él. Al no existir esta opción para el rol 'Usuario', éste no puede acceder a ninguna de las pantallas de este punto.

5.4.1 Feed de Usuarios

Esta pantalla, reflejada en la Ilustración 80, muestra un listado de los usuarios registrados en el sistema, tanto Especialistas como Usuarios. Se diferencian por el icono asociado a cada tarjeta. El icono con traje y corbata representa un Especialista, y el icono más simple, un Usuario. La lista se va ampliando con nuevas solicitudes al deslizar hacia abajo, hasta llegar al final.

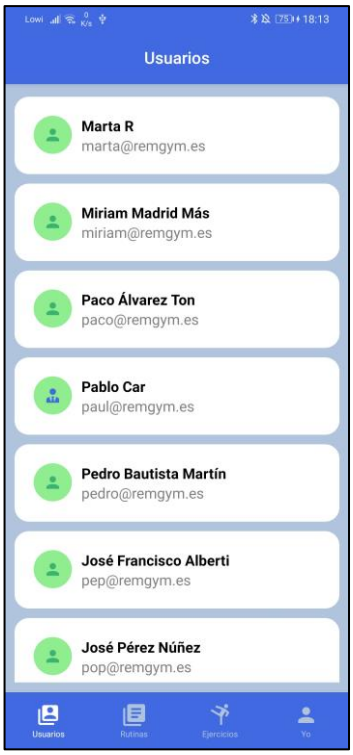

Ilustración 80 – Pantalla de listado de usuarios

5.4.2 Ver detalles de un usuario

Esta pantalla muestra la información del usuario seleccionado en la pantalla '*Feed de Usuarios*'. Si el usuario es un Especialista, tan sólo mostrará la información relacionada con sus datos personales -excepto la contraseña-. Si, por el contrario, es un Usuario, permitirá acceder a sus prescripciones y su historial de ejercicios. Aquí se podrá acceder a la pantalla vista en la Ilustración 78, donde se podrá ver el historial de cada Usuario, si existe.

A continuación, se muestra la pantalla de detalles de usuario en la Ilustración 81:

Ilustración 81 – Pantalla de detalles de un usuario

5.4.3 Ver rutinas prescritas a un usuario

Tanto esta pantalla, como la del siguiente subapartado (Ver ejercicios prescritos), tienen el mismo formato base, sólo que dinámicamente se cambian algunos aspectos en función de la información solicitada. Muestra las prescripciones de rutinas a un usuario, como se puede apreciar en la Ilustración 82.

Ilustración 82 – Pantalla de rutinas prescritas a un usuario

5.4.4 Ver ejercicios prescritos a un usuario

Como se ha comentado en el subapartado anterior, esta pantalla y la vista en la Ilustración 82 se diferencian en el tipo de información mostrada. A continuación, se aporta una captura de la pantalla en la Ilustración 83:

Ilustración 83 – Pantalla de ejercicios prescritos a un usuario

Nota: Tanto esta pantalla como la de rutinas prescritas permiten añadir un nuevo elemento al especialista, pulsando el botón (+) en la parte baja-derecha de la pantalla. Esta pantalla de creación de prescripciones se verá en el punto 5.4.6.

5.4.5 Ver detalles de la prescripción a un usuario

En los dos puntos anteriores vimos pantallas que mostraban un listado de elementos. Al seleccionar cualquiera de esos elementos, se accede a la pantalla de detalles de la prescripción, mostrada en la Ilustración 84. En ella hay dos botones: uno para editar, accediendo a la pantalla descrita en el subapartado siguiente, y otro para eliminar la prescripción.

Ilustración 84 – Pantalla de detalles de la prescripción. Rutina

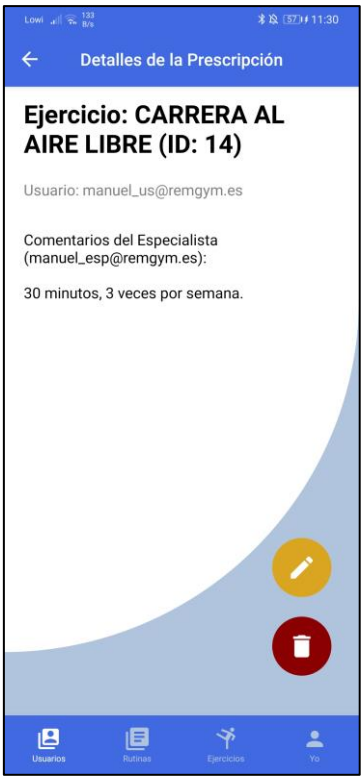

Ilustración 85 – Pantalla de detalles de la prescripción. Ejercicio

5.4.6 Crear -o modificar- prescripción a usuario

A esta pantalla se accede tanto desde la descrita en el punto 5.4.4, pulsando el botón de añadir prescripción, como desde la descrita en el punto 5.4.5, pulsando el botón de editar prescripción. En esta pantalla, reflejada en la Ilustración 86, se muestra un formulario que estará parcialmente completo o no, en función de si se modifica una ya existente o se crea una de cero.

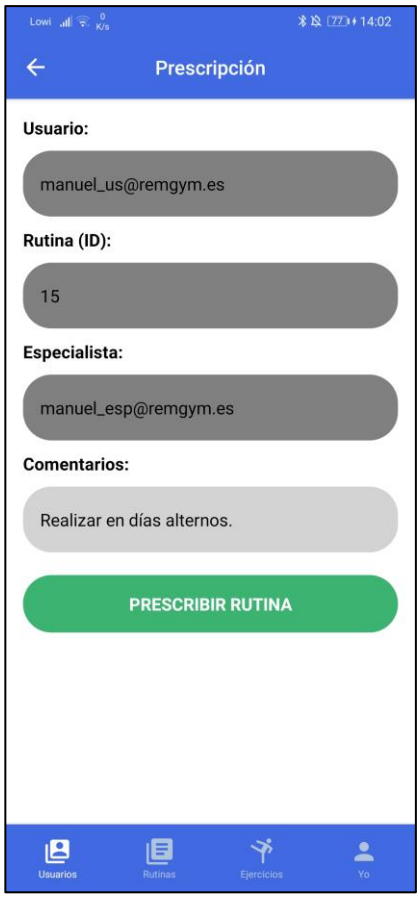

Ilustración 86 – Pantalla de crear o modificar prescripción a usuario

5.5 Navegador 'Rutinas'

Este navegador es accedido al presionar la opción correspondiente en el panel de navegación. A continuación, veremos en cada subapartado las diferentes pantallas que son accesibles mediante él.

5.5.1 Feed de Rutinas

En esta pantalla se mostrará un listado de rutinas. El aspecto de la pantalla y la información mostrada dependerá del rol del usuario de la siguiente manera:

- Especialista: Le serán mostradas todas las rutinas existentes en el sistema, con su identificador asociado. Además, dispondrá de un botón (+) en la parte baja-derecha de la pantalla para añadir nuevas rutinas, pantalla que se mostrará en el punto 5.5.5.
- Usuario: Le serán mostradas tan sólo aquellas rutinas que le han sido prescritas, sin poder ver el identificador.

A continuación, se muestra la pantalla del listado de rutinas bajo el rol 'Especialista' en la Ilustración 87:

Ilustración 87 – Pantalla de listado de rutinas

5.5.2 Ver detalles de una rutina

Al seleccionar alguna de las rutinas mostradas en la pantalla del punto anterior, se accederá a esta pantalla, reflejada en las Ilustraciones 88 y 89. En ellas, se muestran todos los detalles de la rutina, junto con un botón común a ambos roles para ver los ejercicios asociados existentes. Este botón dirigirá al usuario a la pantalla descrita en el punto 5.6.1, enfocada a las asociaciones de ejercicios a dicha rutina. Además, para el rol 'Especialista' se mostrarán dos botones: uno para editar los datos de la rutina, y otro para borrarla completamente.

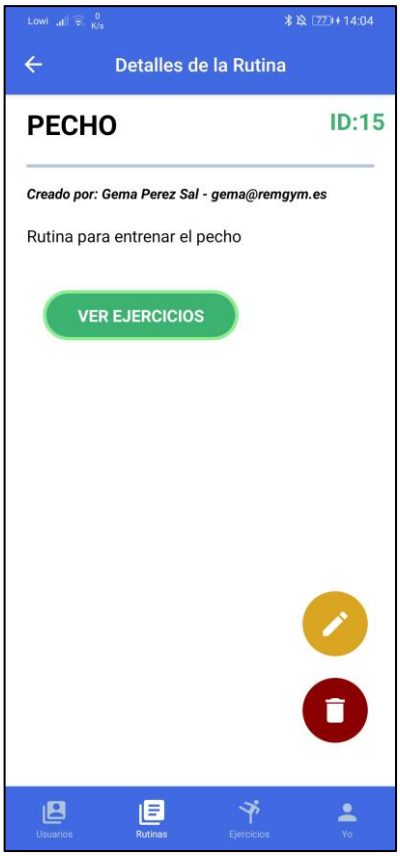

Ilustración 88 – Pantalla de detalles de rutina. Especialista

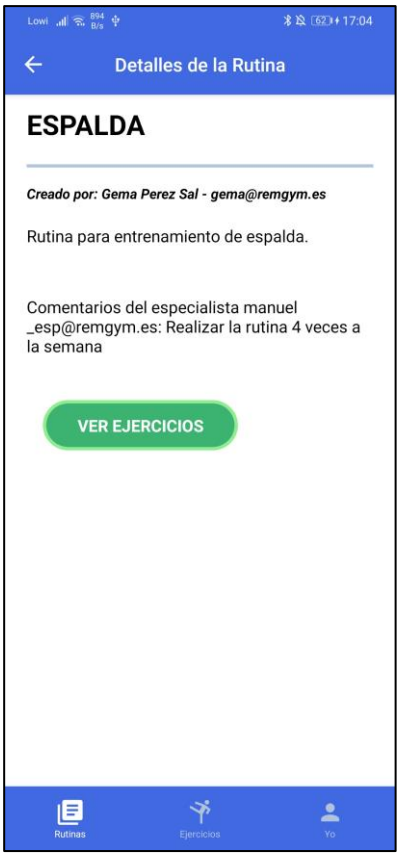

Ilustración 89 – Pantalla de detalles de rutina. Usuario

5.5.3 Ver ejercicios asociados a una rutina

En la pantalla del punto anterior, al pulsar el botón de 'Ver ejercicios', se mostrará una pantalla con el listado de ejercicios incluidos en una rutina. Esta pantalla incorpora un botón (+) en la parte baja-derecha de la pantalla para incorporar nuevos ejercicios a la rutina. Se muestra a continuación en la Ilustración 90:

Ilustración 90 – Pantalla de ejercicios asociados a una rutina

5.5.4 Ver detalles de una asociación de ejercicio a rutina (Especialista)

Esta pantalla es prácticamente idéntica a la mostrada más adelante en el punto 5.6.2, con la diferencia de que incorpora los comentarios establecidos por el especialista en el momento de realizar la asociación. Como son muy similares, no se aporta captura en este punto. También incorporará dos botones como en otras pantallas de detalles, uno para editar la asociación, y otro para borrarla. La edición se verá en el subapartado siguiente.

5.5.5 Crear o Modificar asociación de ejercicio a rutina (Especialista)

Esta pantalla contiene un formulario para establecer los datos de la asociación de un ejercicio a una rutina. Se puede acceder tanto el botón indicado en la Ilustración 90 -Pantalla de ejercicios asociados a una rutina-, en el punto 5.5.3, como en el botón de editar de la pantalla de detalles de asociación, en el punto 5.5.4. A continuación, se muestra en la Ilustración 91 la pantalla de crear o modificar asociación:

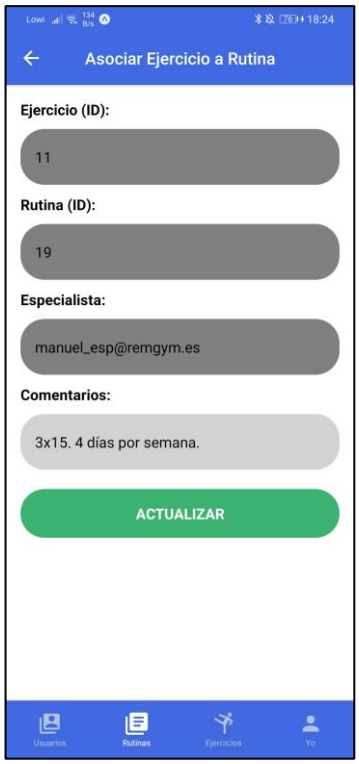

Ilustración 91 – Pantalla de crear o modificar asociación de ejercicio a rutina

Nota: El botón para enviar el formulario cambiará el texto mostrado según se modifique una asociación existente o se cree una nueva. La pantalla mostrada en la Ilustración 91 se correspondería con la edición de una asociación ya existente en el sistema.

5.5.6 Crear o modificar datos de una rutina (Especialista)

Esta pantalla muestra un formulario para crear o modificar una rutina. Se accede desde la pantalla de listado de rutinas, pulsando el botón de añadir una nueva, o desde la pantalla de detalles de rutina, pulsando el botón de editar. A continuación, se muestra en la Ilustración 92 el aspecto de la pantalla:

Ilustración 92 – Pantalla de crear o modificar datos de una rutina

5.6 Navegador 'Ejercicios'

Este navegador es accedido al presionar la opción correspondiente en el panel de navegación. A continuación, veremos en cada subapartado las diferentes pantallas que son accesibles mediante él.

5.6.1 Feed de Ejercicios

Esta pantalla muestra el listado de ejercicios. El aspecto de la pantalla y la información mostrada dependerá del rol del usuario de la siguiente manera:

- Especialista: Le serán mostrados todos los ejercicios existentes en el sistema, con su identificador asociado. Además, dispondrá de un botón (+) en la parte baja-derecha de la pantalla para añadir nuevos ejercicios, pantalla que se mostrará en el punto 5.6.3.
- Usuario: Le serán mostrados tan sólo aquellos ejercicios que le han sido prescritos, sin poder ver el identificador.

A continuación, se muestra en la Ilustración 93 una captura de esta pantalla:

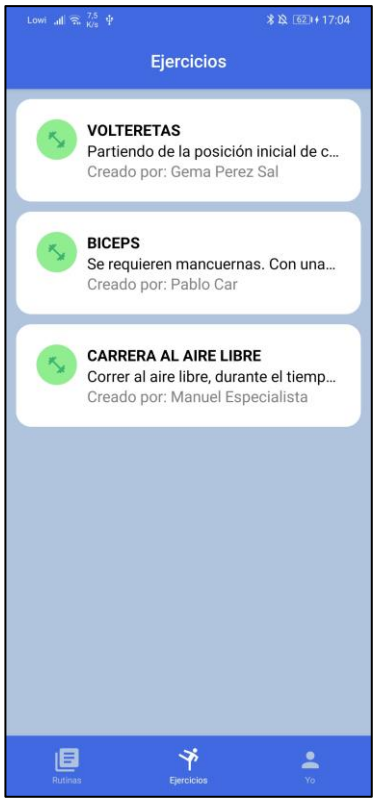

Ilustración 93 – Pantalla de listado de ejercicios

5.6.2 Ver detalles de un ejercicio

Esta pantalla, reflejada en las Ilustraciones 94 y 95 con las diferencias para Especialista y Usuario respectivamente, muestra toda la información de un ejercicio. En el caso del Especialista, se mostrarán dos botones, uno para editar los detalles y otro para eliminar el ejercicio. En el caso del Usuario, sólo se mostrará un botón, para proceder a realizar el ejercicio, y los comentarios del Especialista sobre la prescripción.

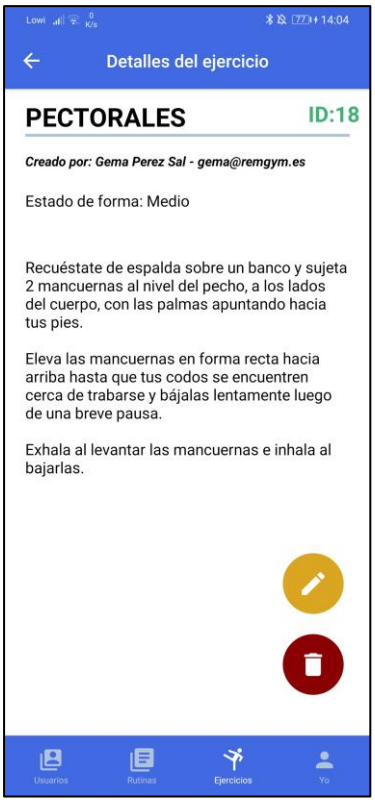

Ilustración 94 – Pantalla de detalles de ejercicio. Especialista

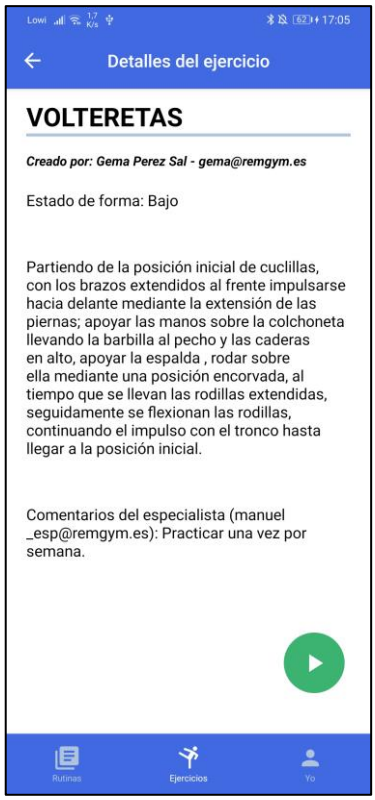

Ilustración 95 – Pantalla de detalles de ejercicio. Usuario

5.6.3 Crear o modificar datos de un ejercicio (Especialista)

Esta pantalla, accedida desde la pantalla de listado comentada en el punto 5.6.1, pulsando el botón de añadir, o bien desde la pantalla de detalles comentada en el punto 5.6.2, muestra un formulario para introducir los datos del ejercicio a crear o modificar. En el formulario se deberán indicar detalles como si el ejercicio deberá recopilar información de ciertos sensores, así como un campo de texto donde ubicar un enlace de YouTube para mostrar un vídeo sobre el mismo ejercicio. Se muestra a continuación, en la Ilustración 96:

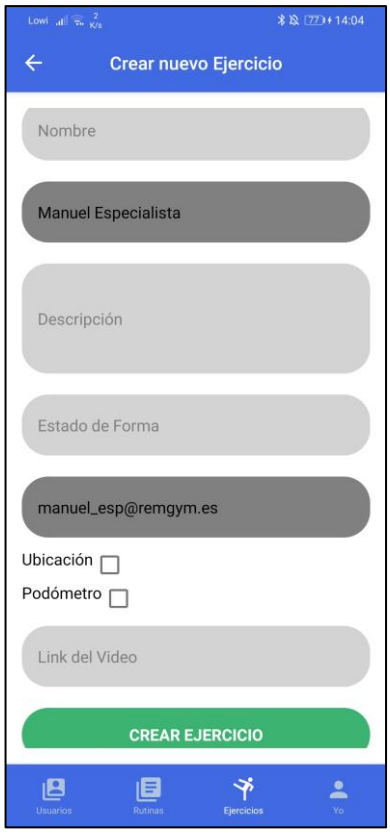

Ilustración 96 – Pantalla de crear o modificar ejercicio

5.6.4 Realizar un ejercicio (Usuario)

Esta pantalla es la que se muestra cuando el usuario presiona el botón de comenzar ejercicio, desde la pantalla de detalles. Según la naturaleza del ejercicio a realizar, se mostrará en una zona una 'Webview' con un vídeo de YouTube insertado para seguir el ejercicio, o bien, un panel con información que la aplicación va recopilando acerca de ciertos sensores. A continuación, en las Ilustraciones 97 y 98 se muestran ambas pantallas:

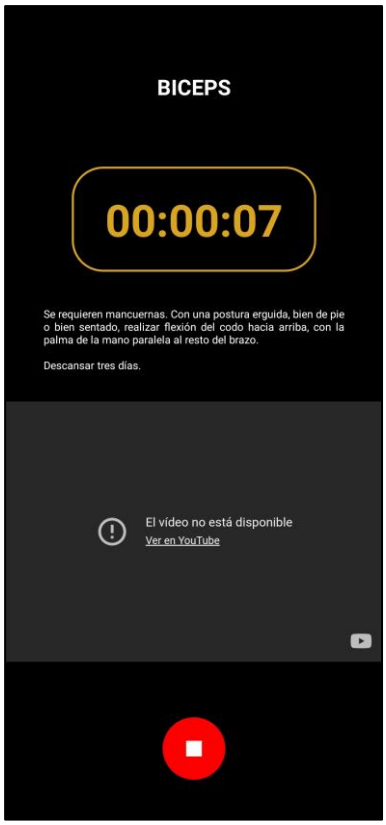

Ilustración 97 – Pantalla de realizar ejercicio. Vídeo

Ilustración 98 – Pantalla de realizar ejercicio. Sensores

Nota: En la Ilustración 97, aparece el vídeo como no disponible, ya que el vídeo de muestra que se empleó para hacer pruebas no permite su inserción. No obstante, y dado que no se dispone de material sobre el que se tenga permiso para publicarlo en el documento, se ha optado por mostrar la vista de esta manera, ya que es con interés de mostrar la distribución de los elementos que la componen, y para que el lector tenga una idea del aspecto que tiene.

6 PRUEBAS

Las pruebas se pueden emplear para demostrar la presencia de errores, pero nunca para demostrar su ausencia.

- Edsger W. Dijkstra -

Este punto pretende comentar el procedimiento seguido para realizar pruebas, y un listado de pruebas que se han hecho sobre el sistema durante el desarrollo. En cuanto al procedimiento, ha consistido en controlar el funcionamiento de los componentes del sistema mediante el análisis de las entradas y salidas de cada una de ellas.

La estructura del punto será separar por puntos las pruebas realizadas sobre cada uno de los elementos del proyecto.

6.1 Pruebas de la Base de Datos

Para realizar las pruebas relacionadas con la base de datos se ha empleado tanto el código del servicio web en sus fases más tempranas de desarrollo -prueba de conectividad y algunas consultas iniciales- como la herramienta MySQL Workbench, analizada en el punto 3.1.4.2 del documento.

El objetivo de las pruebas era lanzar consultas al servidor, más o menos complejas, para analizar que el resultado obtenido era el esperado.

6.2 Pruebas del servidor. Uso de las API

Una vez comprobado que la base de datos funciona correctamente, se pudo analizar el funcionamiento del servicio web.

En la depuración del código, la realización de pruebas para análisis de resultados ha estado basada en el uso del terminal de comandos, como salida (‘output’) de la aplicación, llegando a imprimir por pantalla el resultado de ciertas operaciones para garantizar que, tanto en forma como en contenido, era apropiado.

Concretamente, también se analizó el comportamiento de las API que tiene la aplicación web. Para ello, antes de tener la aplicación móvil desarrollada, se ha empleado la herramienta Postman, comentada en el punto 3.2.2 del documento.

Para ello, se redactaron peticiones HTTP en la herramienta, de manera que se hicieran llamadas al servidor en las rutas correspondientes, y haciendo uso de los métodos definidos para cada ruta. El objetivo era analizar que tanto la información devuelta como su formato era la apropiada para ser utilizada por la aplicación móvil en el futuro.

Nota: Para el análisis constante y fluido de cualquier cambio realizado en el código del servidor, se ha empleado la biblioteca nodemon, comentada en el punto 2.4.3.5 del documento. Con esta biblioteca, se permite arrancar el servidor de manera controlada, para que, ante cualquier cambio guardado en los ficheros de código, el servidor se reinicie automáticamente. Con esto se consigue dinamizar la solución de errores o comportamientos indeseados en el funcionamiento del servidor.

6.3 Pruebas de la Aplicación móvil

Aunque el desarrollo de la aplicación móvil se inició posteriormente al del servidor, gran parte del mismo se ha realizado en paralelo, modificando aspectos de alguna de las dos partes para satisfacer necesidades en la contraria.

Las pruebas de funcionamiento realizadas en la Aplicación móvil han sido realizadas en parte mediante el simulador de Android proporcionado por Android Studio. El modelo y versión del terminal emulado ha sido un Google Pixel 3 con SO Android versión 10.

De forma complementaria al uso del simulador de Android Studio, las pruebas también han sido realizadas en un terminal físico mediante el uso de la aplicación Expo Go. El modelo del terminal ha sido un Huawei P30, con SO Android versión 10.

Aunque se comentará también en el apartado 7.3 del capítulo de Líneas de mejora y conclusiones, en el que se dedica el apartado a este tema, no se han realizado pruebas en dispositivos iOS. El motivo de esto ha sido que Apple no facilita simuladores en dispositivos que no tengan SO Mac -mediante el uso de XCode-, ni tampoco he dispuesto de un iPhone físico sobre el que desplegar la aplicación.

Nota: Al igual que ha ocurrido con el desarrollo del servidor, las herramientas empleadas han facilitado probar el resultado de cada cambio realizado de forma dinámica. Esto es debido a que, al igual que nodemon en un servidor Node.js, la herramienta Expo permite que, al modificar el código de la aplicación, ésta se recargue automáticamente para mostrar los cambios realizados.

6.4 Pruebas de la cola de mensajes MQTT

Para las pruebas relacionadas con el uso de la librería MQTT de la aplicación móvil como el uso del bróker MQTT externo gestionado por HiveMQ, se ha empleado la herramienta HiveMQ Websocket Client, comentada en el punto 3.2.3.

En esta herramienta, accedida mediante el navegador web, se permite tanto publicar como suscribirse a mensajes empleando un tópico concreto. Esto ha permitido analizar los mensajes publicados por la aplicación en la vista de ‘Realizar un ejercicio’, como poder comprobar la recepción y adecuado formateado de mensajes en la vista de ‘Seguimiento de actividad en directo’.

6.5 Pruebas del despliegue en la nube

En el Anexo A: Instalación y Despliegue se ofrece un proceso guiado para instalar y desplegar tanto la base de datos como la aplicación web bajo un servicio en la nube llamado Heroku.

Para las pruebas de la base de datos, incialmente se ha utilizado una conexión remota realizada con MySQL Workbench para enviar consultas sencillas y analizar el resultado.

Para las pruebas del servidor, se ha utilizado tanto la aplicación móvil -ya convertida a formato .apk y también desde Expo- como la herramienta Postman para realizar peticiones HTTP y observar la respuesta. Al comprobar el correcto funcionamiento de las APIs del servidor, también se comprobaba el correcto

funcionamiento de la comunicación de éste con la base de datos.

Además, ha sido muy útil que Heroku proporcione una vista de la salida por terminal de comandos de las aplicaciones desplegadas, para poder depurar el código más fácilmente.

7 LÍNEAS DE MEJORA Y CONCLUSIONES

Cada solución da pie a una nueva pregunta.

- David Hume -

La principal idea de este punto del documento es analizar el proyecto de forma global, en base a los objetivos marcados, y extraer conclusiones sobre el desarrollo. Además, se propondrán una serie de líneas de mejora del servicio propuesto, en las que puede ser interesante enfocar el trabajo futuro sobre este proyecto.

7.1 Seguridad en las comunicaciones cliente-servidor

Aunque el nivel de seguridad es el adecuado en ciertos momentos durante el intercambio de información de la aplicación, por ejemplo, al almacenar la constraseña del usuario cifrada en la base de datos, o al transmitir información del usuario empleando un token cifrado, esto no ocurre en todas las comunicaciones.

Tal como está diseñado el servicio actualmente, tan sólo se usa HTTP en las comunicaciones, de manera que, al iniciar sesión o al registrarse, los datos pueden ser interceptables, lo que supone una importante brecha de seguridad.

Por ello, una de las líneas de mejora más interesantes de cara a garantizar la integridad de los datos, sería el implementar el uso de HTTPS para la comunicación cliente-servidor. De esta manera, las comunicaciones desde el momento del registro o el login no serían interceptables fácilmente.

No obstante, dada la entidad del proyecto, vi más interesante enfocar el tiempo y el esfuerzo en otros aspectos.

7.2 Sincronización con API de Google Fit

Durante el desarrollo de la aplicación móvil, concretamente, durante el desarrollo de las funcionalidades relacionadas con sensores, he encontrado múltiples problemas a la hora de implementar ciertos sensores en la aplicación. Tal es el caso del podómetro, ya que por la naturaleza del proyecto -basado en Expo, con sus limitaciones [60]- y por problemas de compatibilidades con Android en algunos aspectos -el Podómetro del módulo Expo-sensors no funciona correctamente en dispositivos con SO Android-, ha sido muy complicado implementar todas las funcionalidades deseadas.

Por ello, durante la constante búsqueda de información al respecto, he llegado a la conclusión de que lo ideal sería conectar la aplicación con la API de Google Fit, la cual otorga acceso a todos los sensores disponibles en el dispositivo -que son muchos si se encuentra enlazada una pulsera de actividad u otros wearables que sean

compatibles-.

Esto sería dotar al servicio de un amplio abanico de posibilidades y permitir elaborar ejercicios más complejos o mediciones más precisas y con más información sobre el usuario.

7.3 Soporte en iOS

A lo largo de todo el desarrollo del proyecto, todas las pruebas realizadas sobre la aplicación móvil han sido hechas en dispositivos físicos o bien en simuladores, todos ellos con un SO Android. El motivo de esto es que no disponía de un dispositivo físico con iOS y, por la política de hermetismo de Apple, es difícil tener acceso a un emulador de iPhone mediante XCode -sólo disponible en entornos con SO Mac-.

Por ello, no se encuentra garantizado el funcionamiento deseado al 100% de la aplicación móvil en dispositivos iOS, aunque el desarrollo en Expo facilita su exportación a ambas plataformas.

De esta manera, una línea de mejora que probablemente no requeriría demasiado trabajo sería asegurar la compatibilidad de la aplicación con dispositivos Apple, con iOS.

7.4 Ergonomía de la aplicación móvil

El proceso de desarrollo de la aplicación, junto a la evolución de algunos objetivos y funcionalidades, ha propiciado cambios en el aspecto de la aplicación en numerosas ocasiones. Si bien el producto final es lo suficientemente intuitivo y sencillo de usar, siempre hay lugar para la mejora.

Concretamente, hay un espacio importante para la mejora del diseño: el método para prescribir ejercicios o rutinas, y el de incorporar un ejercicio a una rutina. El diseño actual de la aplicación incorpora formularios para estos métodos en los que se solicita el identificador del ejercicio o rutina en cuestión, según proceda. Una opción de mejora para estos mecanismos, sería incorporar menús desplegables para poder seleccionar de forma más cómoda las rutinas o los ejercicios, sin tener que desplazarse entre navegadores.

7.5 Portal de Administración del servicio

La interfaz de usuario en la propuesta que recoge este documento se limita exclusivamente a la aplicación móvil presentada. No obstante, el servicio carece de un módulo de administración del sistema, en el que se podrían implementar nuevamente aspectos del Trabajo Fin de Grado de Pablo Carmona -como, por ejemplo, el que un usuario pueda estar activado o no-, así como funcionalidades nuevas -en general, una gestión más completa de los usuarios-.

Así, una interesante línea de mejora podría ser el desarrollo de un portal web de administración del servicio, por ejemplo, implementando un nuevo rol, que existiera para llevar a cabo esta labor.

7.6 Conclusiones

En este apartado se comentarán las conclusiones extraídas sobre todo el transcurso del proyecto. Para ello, volveremos sobre los objetivos marcados en el punto 1:

- Lenguaje de programación único: JavaScript supone una opción muy potente para un desarrollador full-stack, ya que puede programar tanto el cliente como el servidor utilizando un único lenguaje de

programación. Esto permite profundizar y aprender más sobre este lenguaje, en lugar de dividir tiempo y esfuerzo en más de uno.

Además, las opciones de desarrollo con este lenguaje no son opciones residuales en el mercado. Node.js es un entorno ampliamente usado para desplegar servidores, y React Native coge más fuerza cada año.

Otro aspecto muy interesante de JavaScript es la enorme cantidad de bibliotecas y módulos desarrollados por la comunidad, de código abierto, que se pueden usar libremente para implementar casi cualquier funcionalidad que se desee.

- Versatilidad de los dispositivos móviles: Como se comentó al principio del documento, el uso de dispositivos móviles está a la orden del día. Esto, sumado a la gran cantidad de sensores de que dispone, ya no por sí solos, sino mediante pulseras de actividad, relojes inteligentes, etc., lo convierten en la opción ideal para desarrollar la interfaz de usuario de un servicio de estas características. Un ejemplo de esto es la descomunal gama de bibliotecas para uso de sensores existente en React Native -que, desgraciadamente, debido a las limitaciones de Expo son en muchos casos imposibles de usar-.

- Seguimiento de actividad física en directo: Este objetivo no se planteó desde el principio del desarrollo, sino que surgió fruto de la evolución del uso que se iba a dar a la tecnología MQTT en el servicio. Al final acabó siendo una comunicación exclusiva entre Usuarios y Especialista, planteada como unidireccional, para informar en tiempo real de la actividad de los Usuarios.

  Esto en definitiva permitiría al Especialista analizar el nivel de esfuerzo que requiere cierto ejercicio para algunos Usuarios, permitiéndole adaptarlos más adecuadamente a cada uno.

  Dadas las características de la tecnología MQTT, acabó siendo la elección ideal para llevar a cabo esta funcionalidad, ya que no es mucha la información a transmitir, y el volumen de usuarios publicando podría llegar a ser alto en términos de tráfico.

- Optimización del tráfico que maneja el servidor: Muy ligado al objetivo anterior. Al librar al servidor de gestionar el tráfico para comunicar Usuario y Especialista, el servidor quedaría más libre para atender solicitudes que sí lo involucrasen directamente, como registros, solicitudes de listados de ejercicios, o creación de nuevos elementos, por ejemplo.

ANEXO A: INSTALACIÓN Y DESPLIEGUE

En este anexo del documento se va a detallar paso a paso cómo instalar y desplegar el servicio y la aplicación móvil, tanto para desarrollo como para un entorno de producción. Se va a seguir un procedimiento por puntos igual que en el resto de capítulos del documento, separando base de datos, servidor y cliente. Adicionalmente, se comentarán las opciones de configuración para establecer correctamente la conexión MQTT en la aplicación móvil.

Además de las herramientas definidas en el capítulo 3 del documento, para el despliegue en la nube desarrollado en este anexo se va a emplear una adicional: Heroku.

Ilustración 99 – Logo de Heroku

Heroku [61] es una plataforma en la nube, de Salesforce Developers, que permite desplegar, monitorizar y escalar aplicaciones. Ofrece una opción gratuita para desplegar de manera limitada ciertas aplicaciones. Además, dispone de manuales para instalación y despliegue de aplicaciones que empleen las tecnologías y lenguajes de programación más utilizados, como puede ser Flask (Python), Node.js (JavaScript), Bases de datos, etc.

Esta herramienta se va a emplear tanto en el despliegue de la base de datos como de la aplicación web, ya que la base de datos se instalará como complemento del servidor.

Antes de continuar, es necesario indicar dónde el lector puede obtener el código del proyecto, que será requerido para llevar a cabo el procedimiento. Se encuentra disponible en dos repositorios de github distintos: uno para el código de la aplicación móvil, y otro que contiene el código de la aplicación web y el fichero de extensión .sql con la relación de tablas y su contenido, que hará falta en el despliegue de la base de datos. La dirección de cada repositorio es la siguiente:

- Repositorio con el código del servidor de la aplicación y el fichero .sql: https://github.com/manueldlhn/tfg-backend
- Repositorio con el código de la aplicación móvil: https://github.com/manueldlhn/tfg-frontend-app

Nota: Todos los procedimientos explicados en este punto para la instalación y despliegue de las distintas partes en local, están enfocados en el sistema operativo Windows 10, tal como se indicó al principio del documento.

Para su instalación y despliegue en otros sistemas operativos, el procedimiento cambiará sensiblemente.

A.1 Servidor MySQL

Este primer punto del anexo va a ir dedicado a la instalación del servidor MySQL y el despliegue de la base de datos en él. A continuación, se distinguen dos apartados: uno para replicar el escenario en el que se ha desarrollado el proyecto, en local, y otro para su uso en la red, alojando la base de datos en la nube.

A.1.1 Instalación y despliegue en local. MySQL Workbench

Para instalar el servidor MySQL y desplegar en él la base de datos del servicio, recurriremos al instalador proporcionado por MySQL Community, mencionado en el punto 3.1.4 del documento. Al ejecutarlo, nos dará la opción de elegir qué recurso instalar: si MySQL Server, MySQL Workbench o Documentación MySQL. Todos ellos con una gran variedad de versiones disponibles. Como se comentó en el punto 3.1.4, la versión instalada, tanto del servidor como de MySQL Workbench, es la 8.0.20.

A continuación, se muestra en la Ilustración 100 la vista principal del MySQL Comminuty Installer, en el que deberemos seleccionar en el menú de la derecha el botón '*Add …*' para, según se muestra en la Ilustración 101, instalar todos los elementos necesarios:

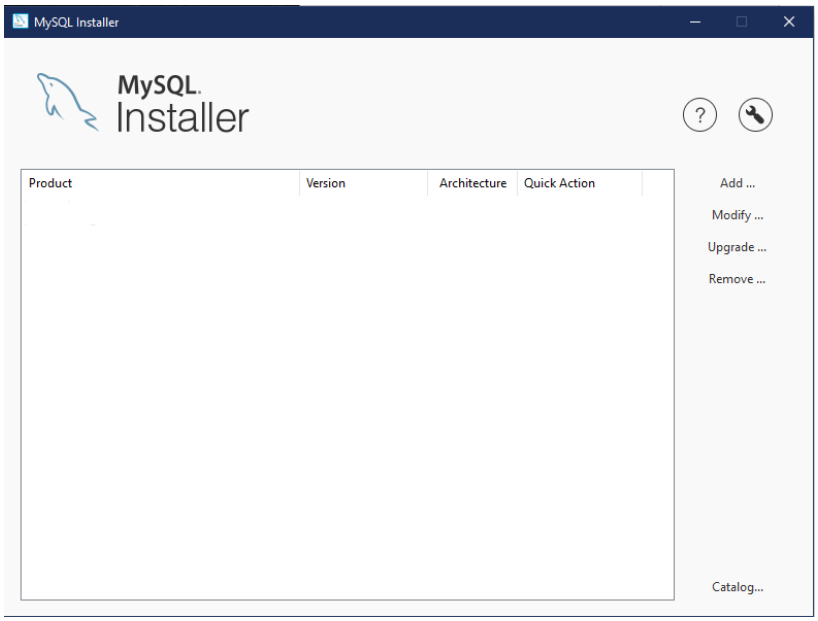

Ilustración 100 – MySQL Installer. Pantalla principal

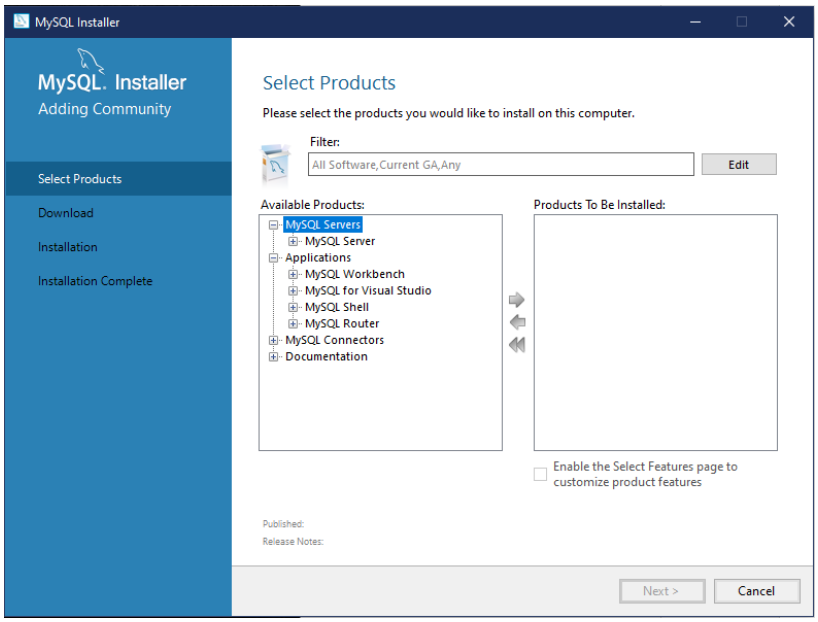

Ilustración 101 – MySQL Installer. Añadir productos

A lo largo del proceso de instalación del servidor, la herramienta nos guiará a lo largo de diversos puntos en los que deberemos especificar ciertos aspectos de configuración -uso para desarrollo, despliegue, etc- así como definir usuarios y contraseñas (al menos habremos de indicar la contraseña del superusuario, 'root'). Aquí, además habrá que crear el usuario con el que el servidor de la aplicación se comunicará con la base de datos. Esto se muestra en las Ilustraciones desde la 102 hasta la 106, con la configuración que debe tener cada paso:

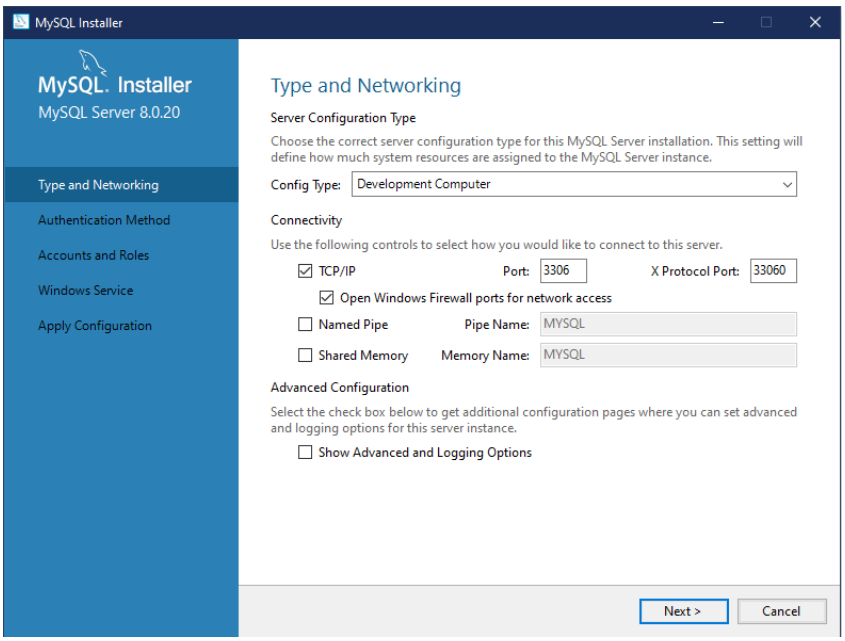

Ilustración 102 – MySQL Installer. Paso 1 de configuración de servidor MySQL

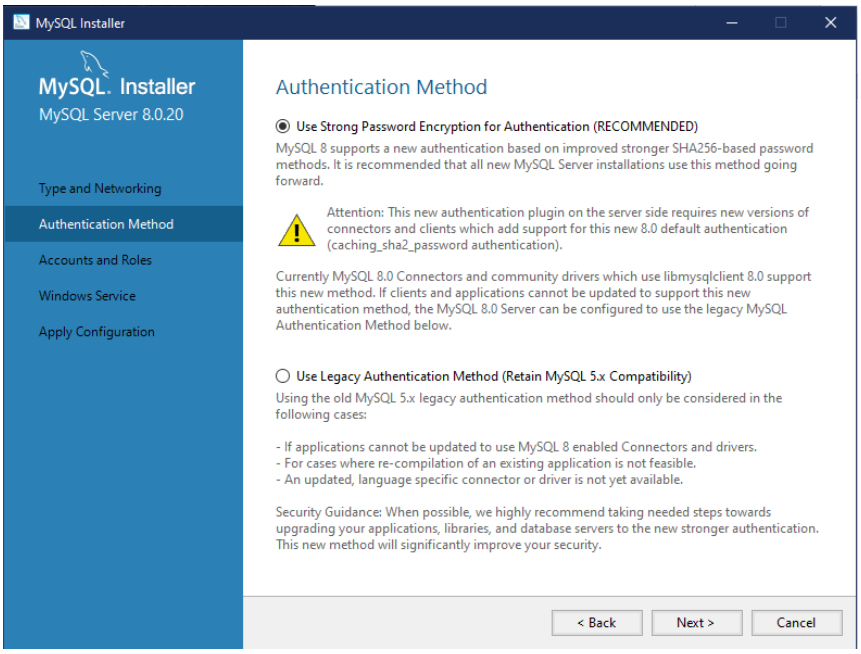

Ilustración 103 – MySQL Installer. Paso 2 de configuración de servidor MySQL

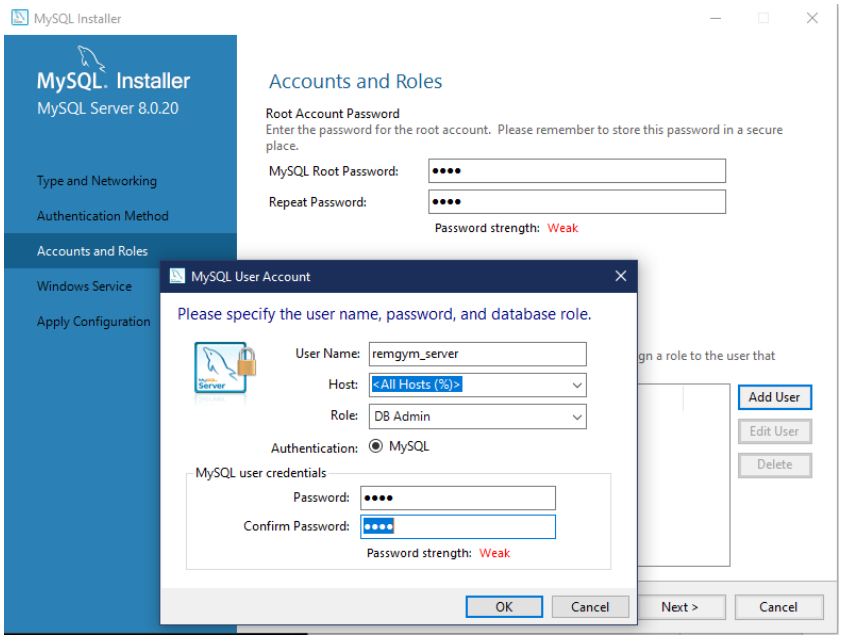

Ilustración 104 – MySQL Installer. Paso 3 de configuración de servidor MySQL

Nota: En el paso de la Ilustración 104 habrá que definir la contraseña de la cuenta *root* y crear el usuario que vaya a utilizar la aplicación web para acceder a la base de datos.

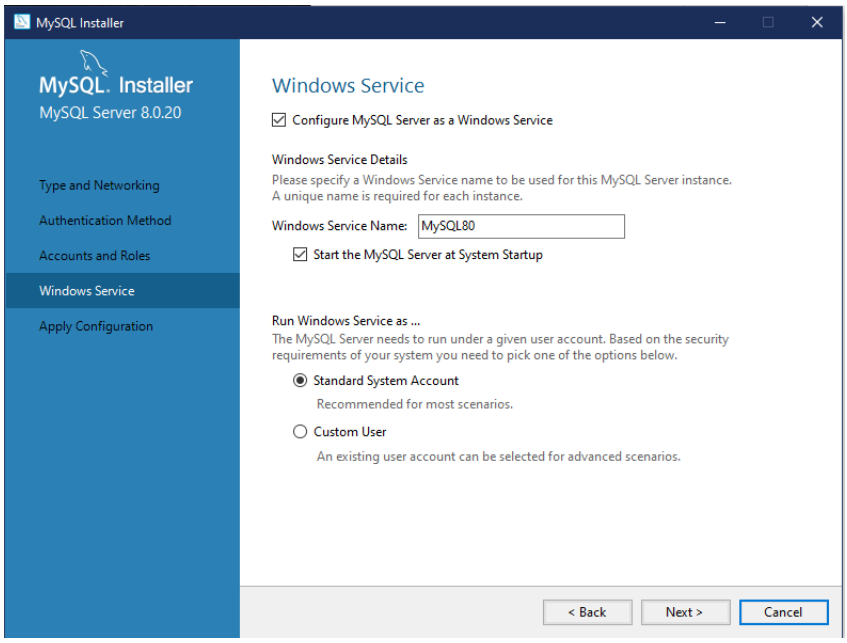

Ilustración 105 – MySQL Installer. Paso 4 de configuración de servidor MySQL

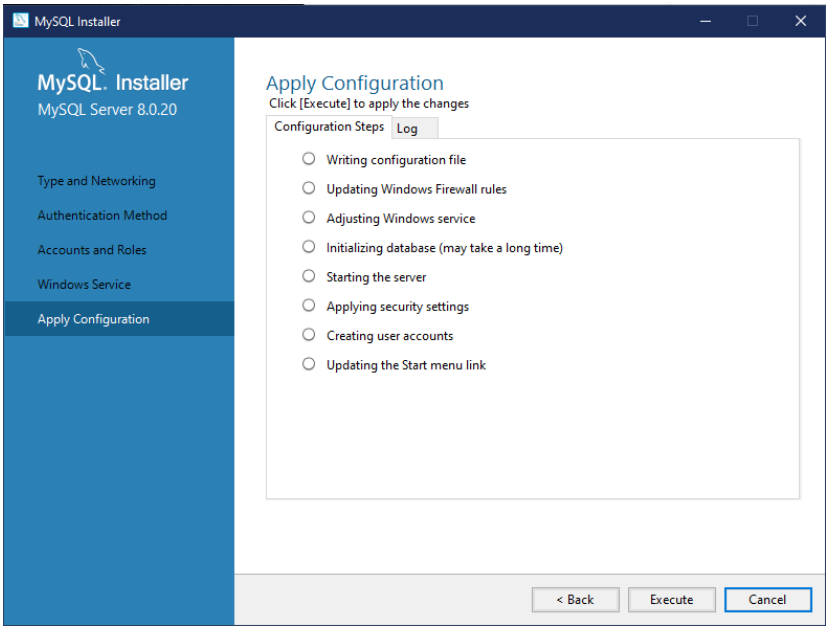

Ilustración 106 – MySQL Installer. Paso 5 de configuración de servidor MySQL

Una vez finalizada la instalación, tanto de MySQL Server como de MySQL Workbench, podremos abrir este último y comenzar a gestionar el servidor. Lo primero que se nos mostrará será algo similar a la Ilustración 107:

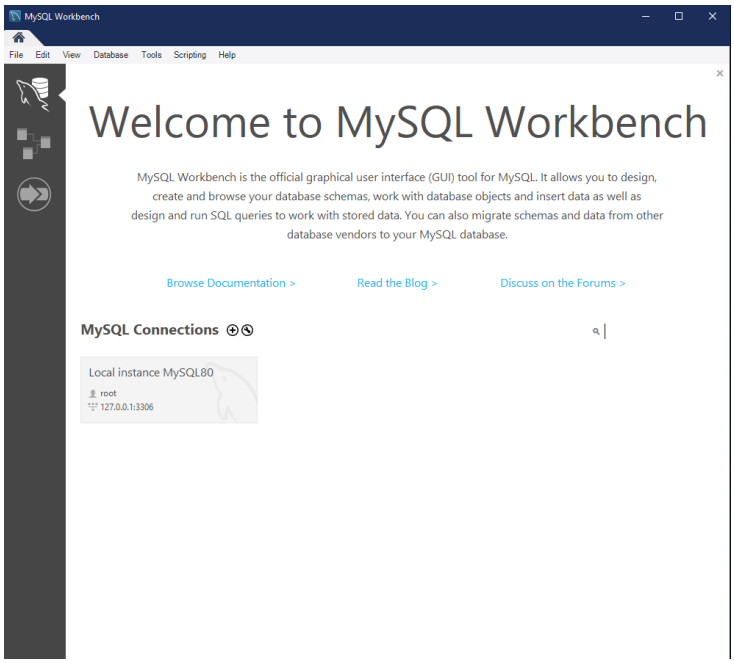

Ilustración 107 – Arranque de MySQL Workbench

A continuación, accedemos a la instancia local, en la que estará alojado nuestro servidor. En ella, se nos muestra la interfaz completa de MySQL Workbench. Lo siguiente es ir al menú '*Server*' y ahí en '*Startup/Shutdown*'.

Se abrirá una pestaña nueva en el editor con información sobre el arranque o parada del servidor MySQL, que se muestra en la Ilustración 108. Si no está arrancado ya, pulsaremos en '*Start Server*'.

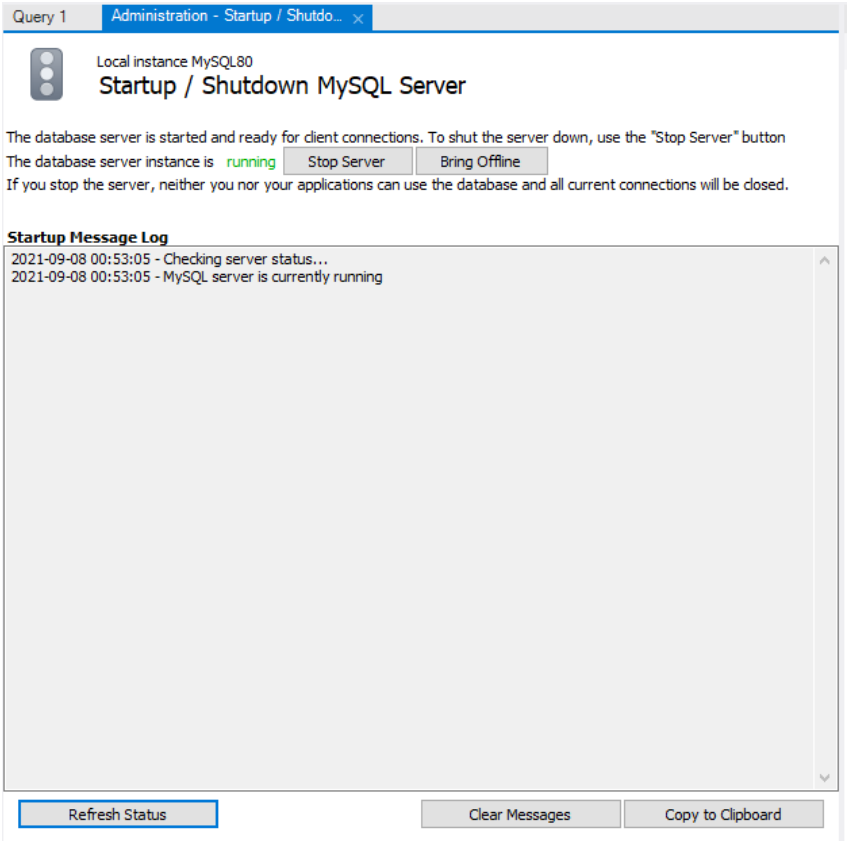

Ilustración 108 – MySQL Workbench. Administración del servidor MySQL

Una vez está el servidor arrancado, es hora de desplegar la base de datos del servicio. Para ello, en la misma herramienta Workbench, Seleccionamos el menú '*File*' y dentro de él, '*New Query Tab*'. Se abrirá un fichero

en blanco de código MySQL. Ahí deberemos escribir lo que aparece en la Ilustración 109, y hacer clic en ejecutar, icono marcado en rojo:

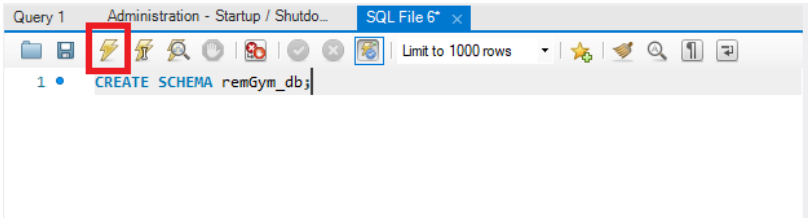

Ilustración 109 – MySQL Workbench. Crear esquema

Tras esto, tenemos creado el esquema de la base de datos. Ahora procederemos a cargar todas las tablas y su contenido para que pueda usarlo el servicio. Para ello, vamos de nuevo al menú '*File*' y ahora '*Open SQL Script*'. Aquí habrá que seleccionar el fichero con extensión .sql facilitado en el repositorio correspondiente del código de este proyecto. Una vez abierto, hacemos clic en el mismo icono de ejecución que en el paso anterior. Tras esto, y si todo ha ido bien, tendremos la base de datos desplegada con éxito de forma local.

Nota: El nombre del esquema deberá conocerlo la aplicación web para poder acceder a la base de datos.

A.1.2 Instalación y despliegue en la nube. Heroku Addon

Este modo de despliegue en la nube implica haber implementado la aplicación web del proyecto también mediante Heroku, que se explica más adelante en el punto A.2.2. El motivo de esto es que la base de datos que se despliega se hace mediante el uso de addons en la plataforma de Heroku. Por eso, es necesario tener a priori la aplicación web previamente desplegada, aunque tras este proceso haya que actualizar algunos parámetros.

El primer paso parte de la vista mostrada en la Ilustración 123, con la aplicación web funcionando. Aquí, iremos a la vista de 'Resources', que tendrá el aspecto mostrado en la Ilustración 110:

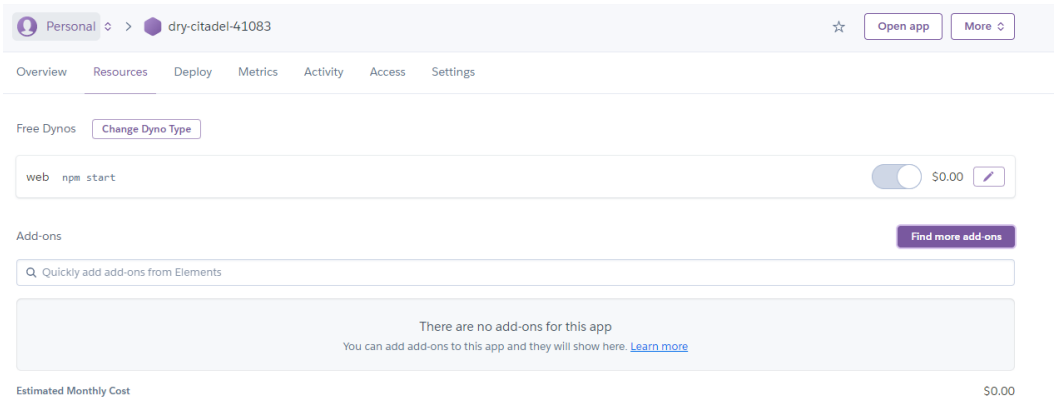

Ilustración 110 – Vista de addons de la aplicación web en Heroku

Aquí pulsaremos en el botón 'Find more addons', que nos redirigirá a un listado de complementos disponibles para incorporar a la aplicación. Para incorporar una base de datos, seleccionaremos JawsDB, como se muestra en la Ilustración 111:

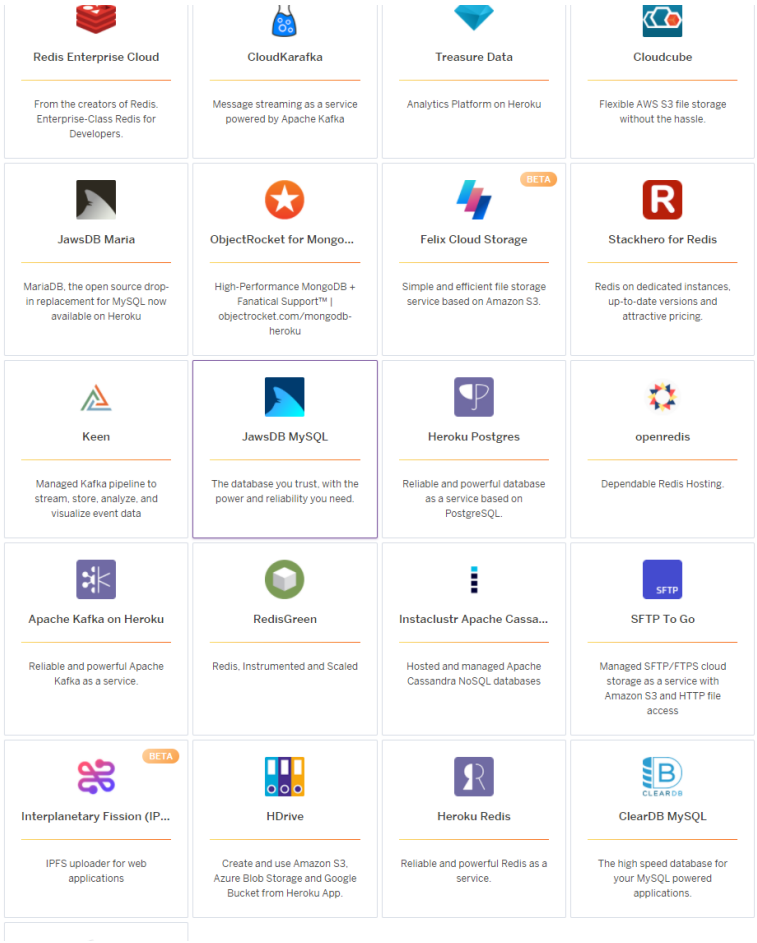

Ilustración 111 – Listado de addons en Heroku

Al seleccionarlo, nos abrirá una nueva vista en la que deberemos seleccionar el plan deseado para la base de datos, como se refleja en la Ilustración 112. Al existir un plan gratuito, es deseable elegir esta opción para bases de datos no muy grandes.

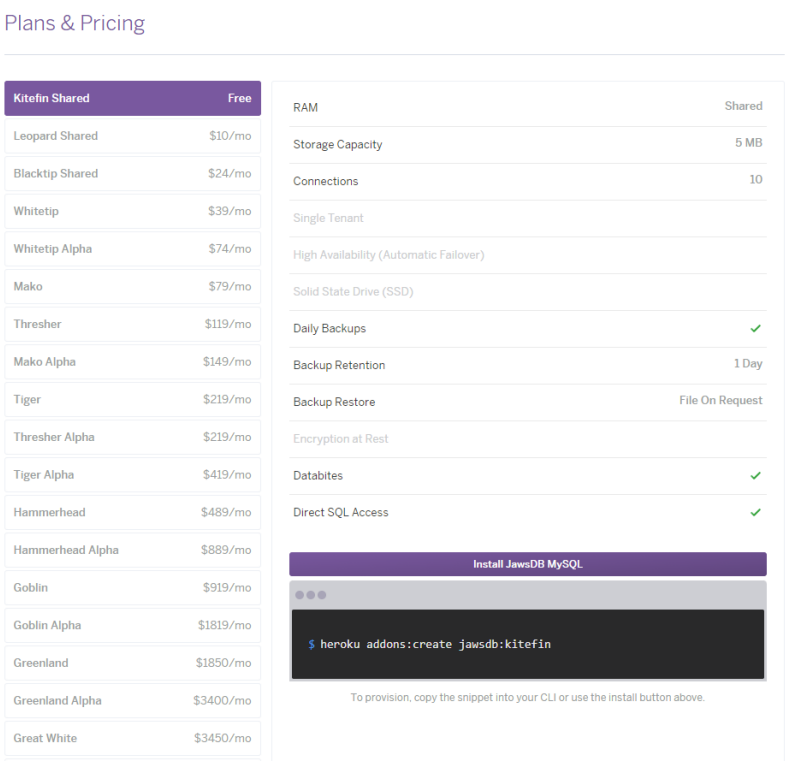

Ilustración 112 – Opciones de configuración del addon JawsDB

Nota: Para incorporar addons en una aplicación de Heroku, la cuenta de usuario debe estar verificada. Esto implica que se deben proporcionar datos de una tarjeta bancaria, aunque el addon sea gratuito.

En la vista de la última ilustración mostrada, pulsaremos en Install JawsDB MySQL. Con ello indicaremos a Heroku que la aplicación va a emplear un complemento de base de datos. En la siguiente vista nos solicitará indicar a qué aplicación web queremos asociar este complemento, así como el plan del mismo – en este caso, se seleccionó el gratuito -. Se puede apreciar en la Ilustración 113:

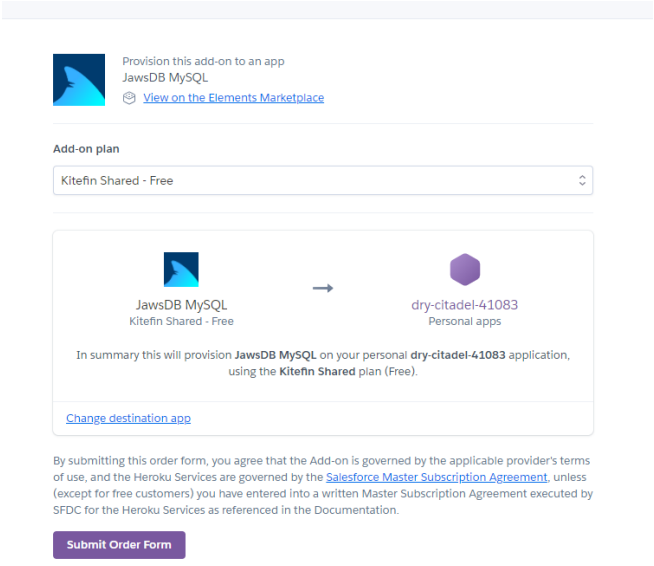

Ilustración 113 – Asociar JawsDB a una aplicación en Heroku

Tras esto, podemos volver a la pantalla de la Ilustración 109, que ahora tendrá el aspecto mostrado en la

Ilustración 114:

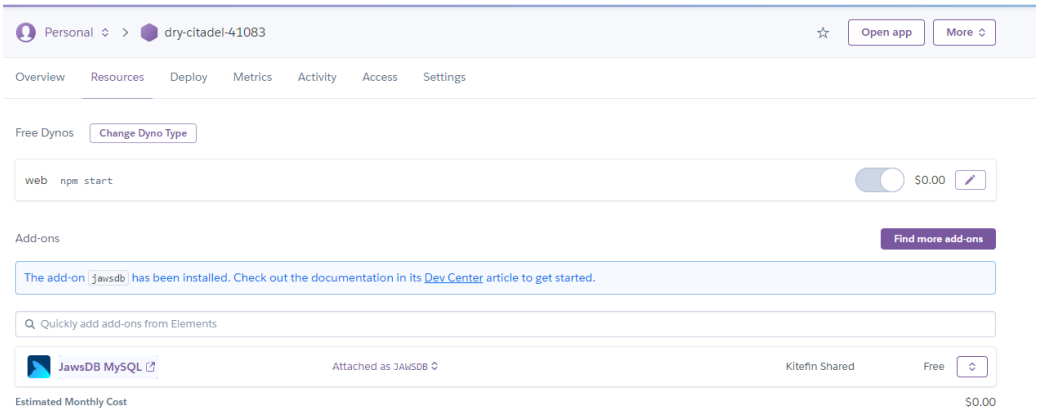

Ilustración 114 – Addon JawsDB MySQL incorporado

En esta vista, si pulsamos sobre el nombre del addon, se abrirá una nueva pestaña en la que se indicará la información para conectarse a la base de datos. Con esta información, podremos establecer una conexión desde MySQL Workbench y cargar toda la estructura. La conexión a la base de datos y la carga de la estructura se muestran en las Ilustraciones 115 y 116 respectivamente.

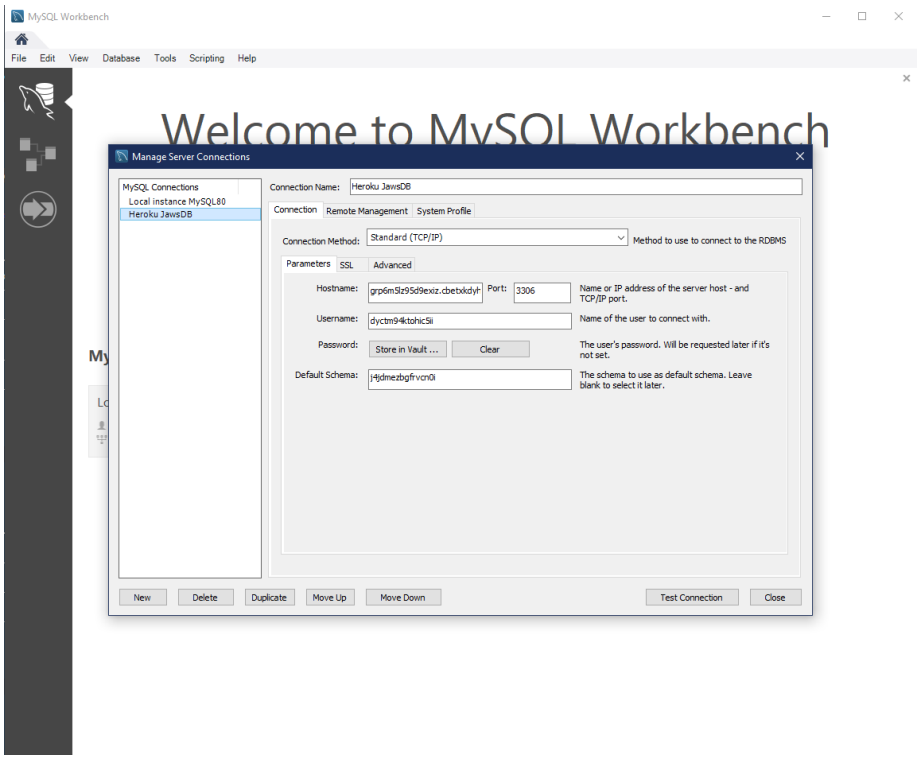

Ilustración 115 – Conexión en MySQL Workbench a JawsDB

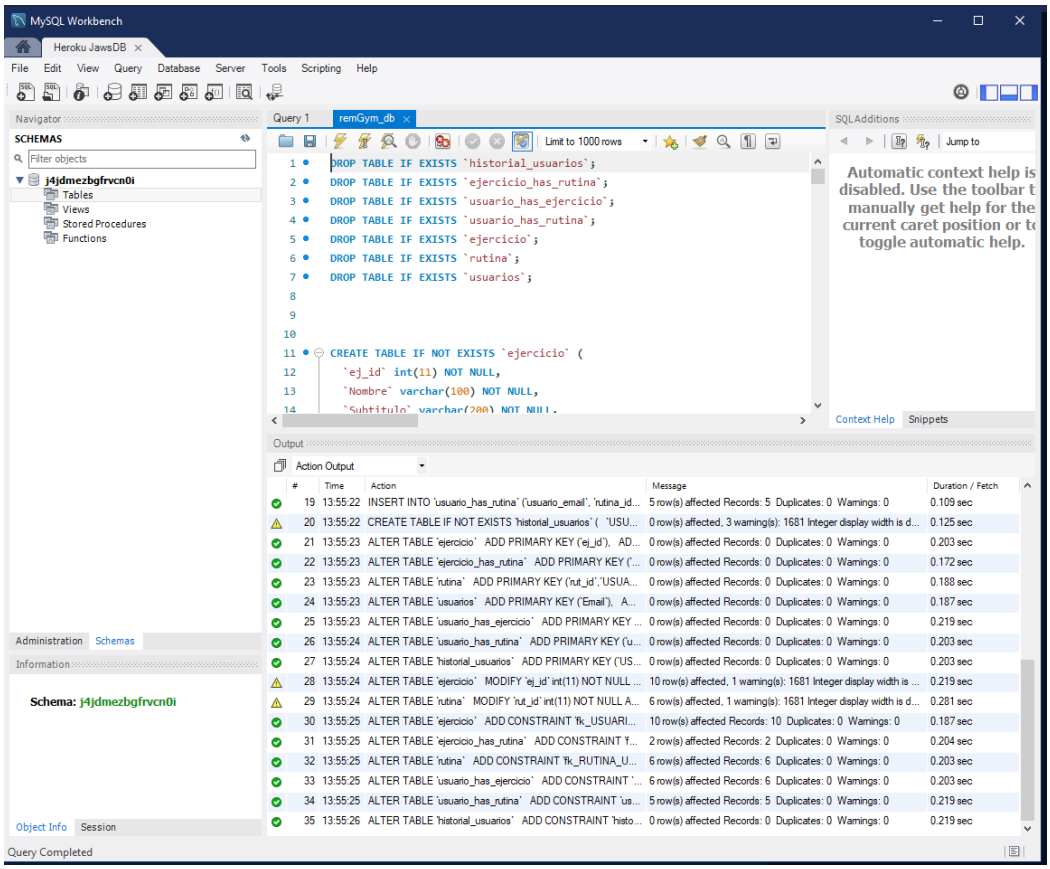

Ilustración 116 – Carga de la estructura de la base de datos a JawsDB

Tras esto, la base de datos estará instalada, desplegada y accesible de forma remota. Para que la aplicación web tenga acceso, habrá que modificar en el fichero db.config.js los parámetros de conexión para ajustarlos a este escenario.

A.2 Back-End

En este punto del anexo se va a comentar cómo, partiendo del código del servidor de la aplicación, se va a desplegar correctamente este componente del servicio. Al igual que el punto anterior, se comentarán dos formas de desplegar el servidor: en forma local, para desarrollo, y en la nube, para su uso final.

Nota: Para la configuración de las conexiones, en el servidor habrá que modificar el fichero */app/database/config/db.config.js* estableciendo los parámetros de conexión con la base de datos.

A.2.1 Instalación y despliegue en local. Node y nodemon

Para instalar y desplegar el servidor en local, lo primero es tener en la máquina instalado Node.js y su gestor de paquetes propio, npm. Para ello, accedemos a la web www.nodejs.org, de donde podremos descargar la versión de Node.js apropiada para nuestro sistema operativo, en este caso Windows 10, como aparece en la Ilustración 117:

Ilustración 117 – Sitio web de Node.js

Tras descargar e instalar node apropiadamente, procedemos a abrir un terminal[11] de comandos de Windows en el que podremos comprobar la versión de Node.js y su gestor de paquetes npm ejecutando lo mostrado en la Ilustración 118:

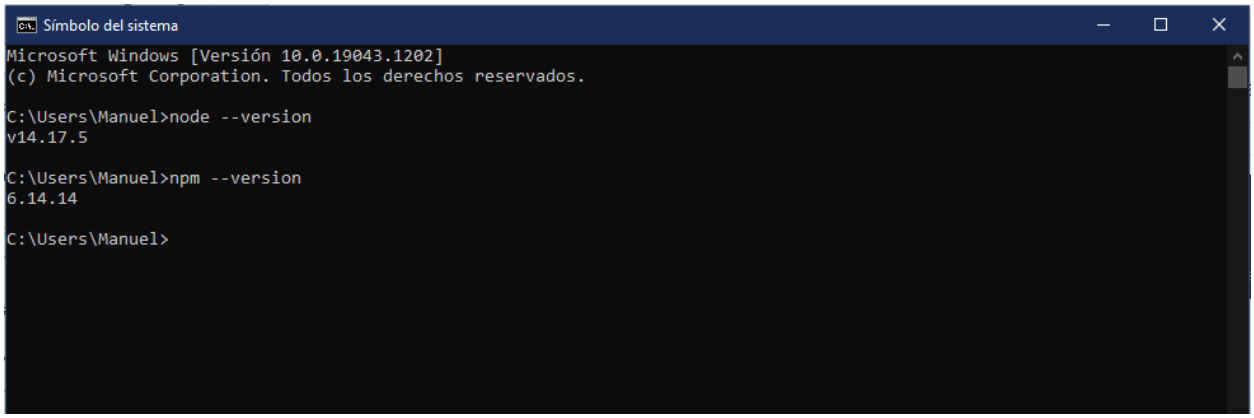

Ilustración 118 – Terminal. Versiones de Node.js y npm

Una vez comprobado, procedemos a ubicar el código del servidor en un directorio, donde se instalarán todas las dependencias necesarias para desplegar el servidor posteriormente. Para ello, ejecutamos el comando `npm install [62]`, que tendrá la salida que se muestra en la Ilustración 119:

[11] Es recomendable que el terminal no sea PowerShell, sino el clásico Símbolo del sistema, y que el usuario con el que realice las operaciones disponga de permisos de Administrador. Con esto, se evitarán problemas.

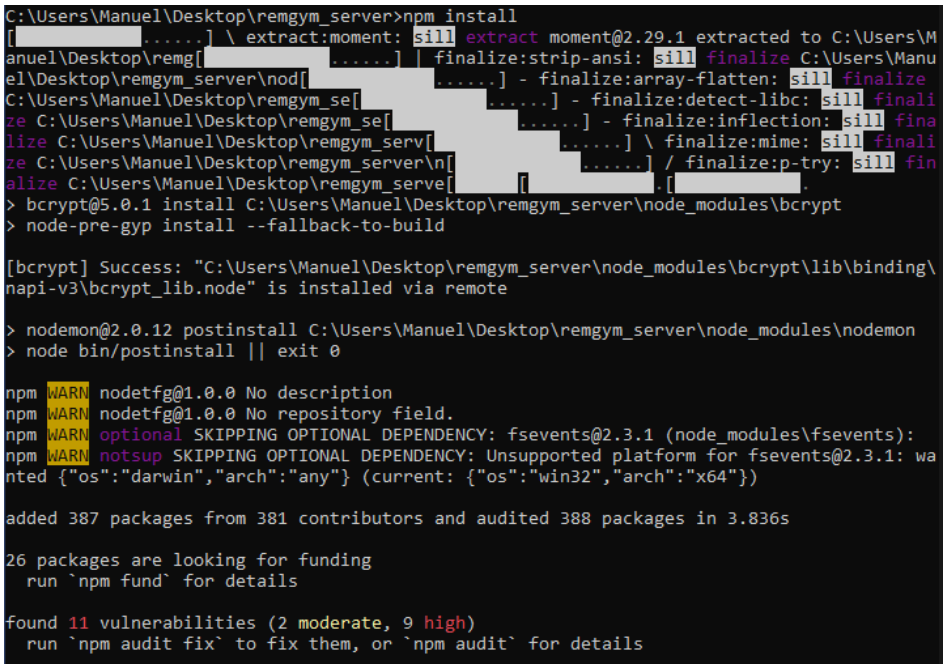

Ilustración 119 – Terminal. Ejecución de npm install para el servidor

Tras esto, tendremos todo lo necesario para arrancar el servidor. Aquí, en función del objetivo de arrancarlo, se puede hacer de dos maneras:

- Ejecutando el comando: `node app.js`
- Ejecutando el comando: `node node_modules\nodemon\bin\nodemon.js`

Con este último arrancamos el servidor a través de nodemon, una biblioteca comentada en el punto 2.4.3.5 con la que el servidor se recarga automáticamente al guardar modificaciones en alguno de sus ficheros.

Nota: Si se realizan modificaciones en la base de datos, hay que modificar en igual medida los modelos definidos en el código del servidor. Esto puede ser muy tedioso para modificaciones sustanciales o que afecten a las relaciones entre tablas. Por ello, se recomienda usar la herramienta facilitada por la biblioteca sequelize-auto, comentada en el punto 2.4.3.7. Antes de generar los modelos se recomienda eliminar los anteriores. Para generar los modelos nuevos tan sólo hace falta ejecutar un comando siguiendo el siguiente formato, indicado en la Tabla 25:

➢ `node node_modules\sequelize-auto\bin\sequelize-auto -o "./ruta/a/los/modelos" -d \ <nombre_bbdd> -h <hostname> -u \ <usuario_bbdd> -p <puerto_bbdd> -x <password_bbdd> -e \ <dialecto_sql>`

Tabla 25 – Comando para generar modelos de la BBDD usando Sequelize-auto

A.2.2 Instalación y despliegue en la nube. Heroku

Para desplegar el servidor en la nube, haremos uso de un servicio gratuito, Heroku [63]. Como requisitos previos, es necesario tener instalado localmente Node.js, su gestor de paquetes npm, estar registrado en Heroku, y haber instalado git y Heroku CLI [64]. Esto último lo veremos a continuación. Además, es recomendable haber seguido todo el proceso del subapartado anterior, ya que facilitará mucho el despliegue.

Para instalar Heroku CLI, los dos métodos más sencillos son los siguientes:

- Acceder a la dirección https://devcenter.heroku.com/articles/heroku-cli y descargar el instalador de ahí, en este caso la versión para Windows 10 de 64 bits.

- Mediante npm, ejecutando `npm install -g heroku`

Una vez instalado, podemos comprobar la versión ejecutando `heroku –version`, que tendrá la salida mostrada en la Ilustración 120:

```
C:\Users\Manuel>heroku --version
 »   Warning: heroku update available from 7.53.0 to 7.57.0.
heroku/7.53.0 win32-x64 node-v12.21.0
```

Ilustración 120 – Terminal. Versión de Heroku CLI

Tras cumplir con todos los requisitos previos, podemos comenzar a adaptar ligeramente el código para que funcione apropiadamente en Heroku.

Lo primero es añadir la versión de Node.js que se utilizará en el fichero 'package.json'. Se puede comprobar la versión de Node.js instalada mediante el comando `node –version`. El código añadido debe tener este aspecto que se muestra en la Ilustración 121:

```
"engines": {
  "node": "14.17"
},
```

Ilustración 121 – Fragmento de código. Versión de Node.js en el fichero package.json

El siguiente paso es modificar una línea en el fichero 'app.js' que afecta al puerto en el que escuchará el servidor. Dado que Heroku comparte recursos entre múltiples proyectos, no siempre estará disponible el puerto elegido en el código. Para ello, al definir el puerto debe quedar como se refleja en la Ilustración 122:

```
const port = process.env.PORT || 3000;
```

Ilustración 122 – Fragmento de código. Definición del puerto en el fichero app.js

Nota: Este paso es para poder hacer un seguimiento hasta cierto punto de en qué puerto escucha el servidor. No obstante, cuando esté desplegado en Heroku, las peticiones se harán a la dirección proporcionada por el servicio, sin indicar puerto, y éste realizará un enrutamiento hasta la aplicación web. Por ello, al desplegar en Heroku no importa realmente en qué puerto escuche la aplicación.

Ahora procedemos a crear la aplicación de Heroku en el directorio del código. A continuación, se muestra en la Tabla 26 un listado de los comandos que se deben ejecutar en orden en el directorio del código, junto a su significado:

Comando	Funcionamiento
`heroku login`	Permite autenticarse en Heroku. Abrirá una ventana externa del navegador para introducir las credenciales.
`git init`	Crea un repositorio git local en el directorio del código del servidor.
`git add .`	Añade todo el contenido del directorio al repositorio local.
`git commit -m "commit inicial"`	Realiza el commit inicial, paso previo a la subida para actualizarlo.
`heroku create`	Crea la aplicación Heroku, asociándole un repositorio git y una dirección en la que será accesible.
`git push heroku master`	Carga todo el contenido del repositorio local en el repositorio de Heroku. Con esto el servidor se despliega automáticamente si no hay errores.

Tabla 26 – Listado de comandos. Despliegue del servidor en Heroku

Con la ejecución de estos comandos ya estaría desplegada la aplicación web en la nube mediante Heroku.

Nota: Cada vez que se realicen modificaciones en el código y se pretenda actualizar el servidor en Heroku, se deberá repetir la ejecución de los comandos `git add`, `git commit` y `git push`, con el formato mostrado en la Tabla 26.

Para gestionar la aplicación web, se puede acceder al sitio https://dashboard.heroku.com/apps para gestionar las aplicaciones desplegadas. La Ilustración 123 muestra la vista de este sitio:

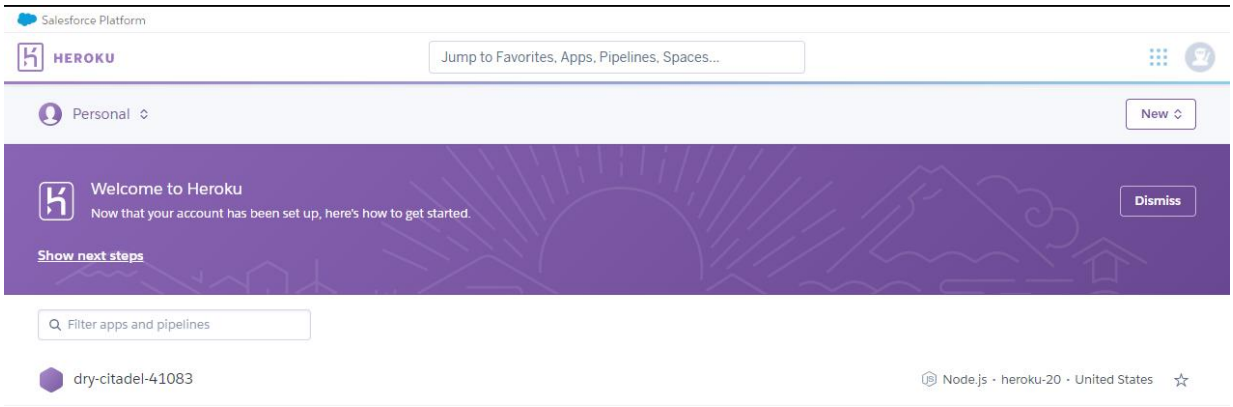

Ilustración 123 – Panel principal del dashboard de Heroku web

Si seleccionamos la aplicación recién creada, podemos acceder a toda la información relacionada, que se muestra a continuación en la Ilustración 124:

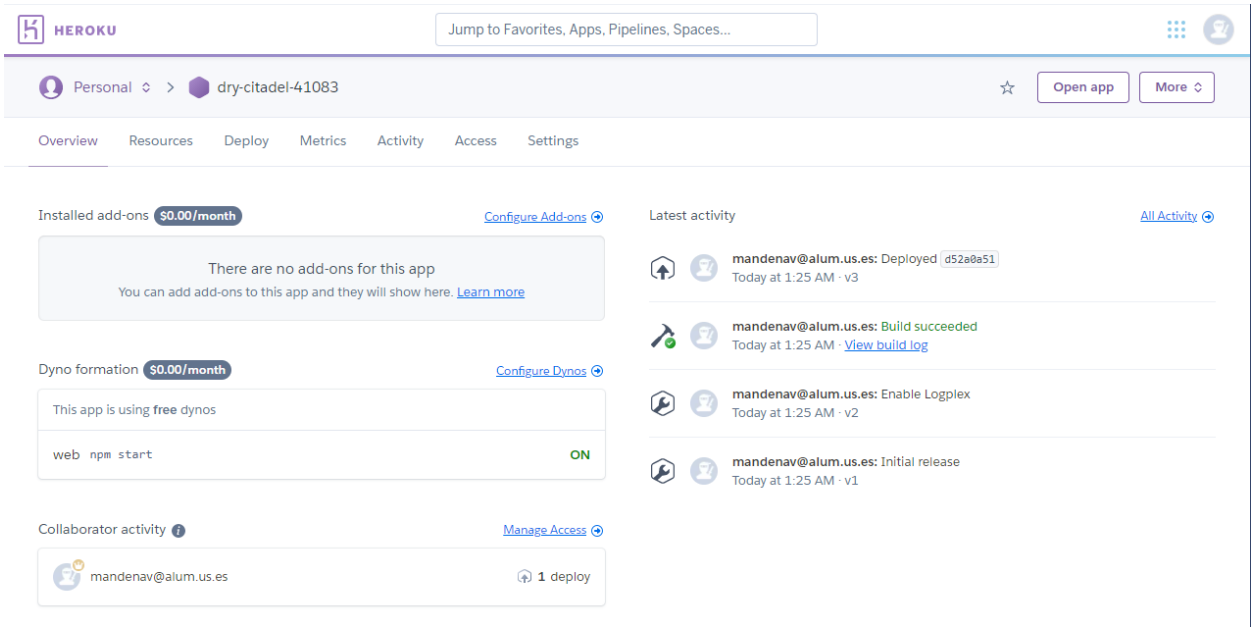

Ilustración 124 – Panel de la aplicación desplegada en Heroku web

Como aspecto interesante, si seleccionamos arriba a la derecha en el botón 'More', podemos acceder a los Logs de ejecución de la aplicación, y analizar su funcionamiento. Esto se muestra a continuación en la Ilustración 125:

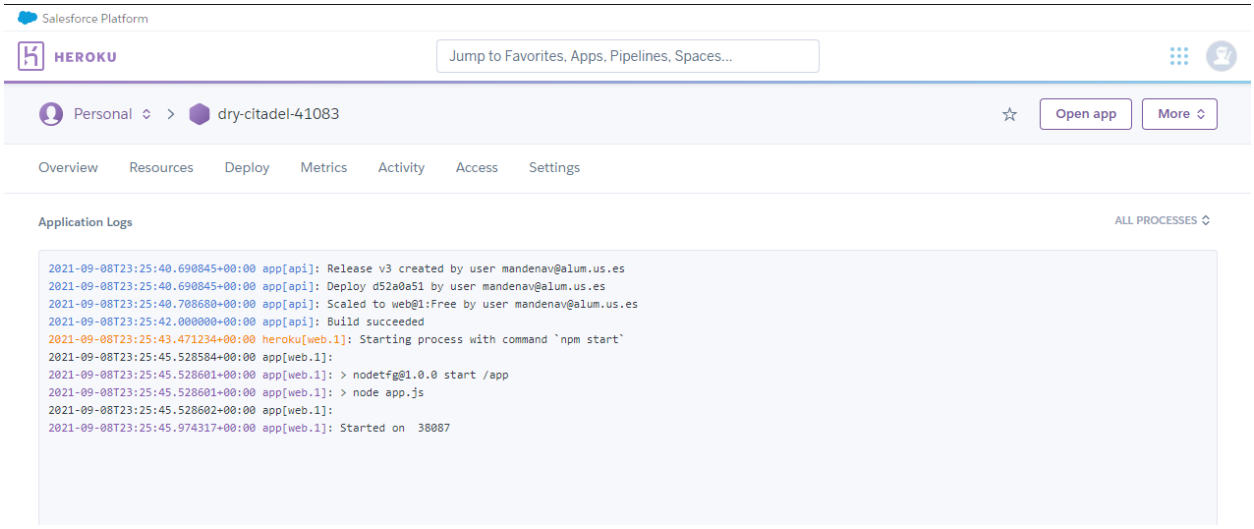

Ilustración 125 – Logs de la aplicación desplegada en Heroku

Nota: Para que la aplicación sea 100% funcional, hay que hacer modificaciones en el código después de haber desplegado la aplicación. El motivo de esto es que hay llamadas en algunos controladores al propio servidor, que por defecto estaba alojado en local, y empleando un puerto definido. Por ello, es necesario modificar estas peticiones – realizadas en los controladores de los ficheros *login.controller.js*, *modify.controller.js* y *register.controller.js*.

Aquí, deberemos sustituir el principio de la url (http://localhost:3000) por la dirección de conexión de la aplicación en Heroku. Ahora no hay que indicar puerto, ya que es indiferente el puerto en que arranque la aplicación, ya que Heroku escuchará en el puerto http estándar (80) y enrutará las peticiones.

A.3 Front-End

En este punto comentaremos el procedimiento para instalar y desplegar tanto el proyecto en Expo para desarrollo de la aplicación móvil, como su compilación para obtener el fichero .apk con el que podremos instalar la aplicación en cualquier terminal Android.

Nota: Para establecer correctamente las conexiones, tanto con la aplicación web como con el bróker MQTT, se debe modificar el fichero */app/config/constants.js*, especificando correctamente todos los parámetros.

A.3.1 Instalación y despliegue del proyecto. Expo

Para instalar y desplegar el proyecto en modo desarrollo bajo el uso de Expo, necesitaremos tener instalado el gestor de paquetes de Node.js, npm. El procedimiento para instalarlo se encuentra en el punto anterior.

Una vez ya tenemos instalado el gestor de paquetes de Node.js, alojamos el código de la aplicación descargado del repositorio indicado al principio de este Anexo en una carpeta. Ahora, para instalar todas las dependencias y hacer que funcione, abrimos un terminal de comandos en la ruta del código – donde se ubica el fichero package.json – y ejecutamos el comando `npm install`. La salida debe ser como se indica a continuación, en la Ilustración 126:

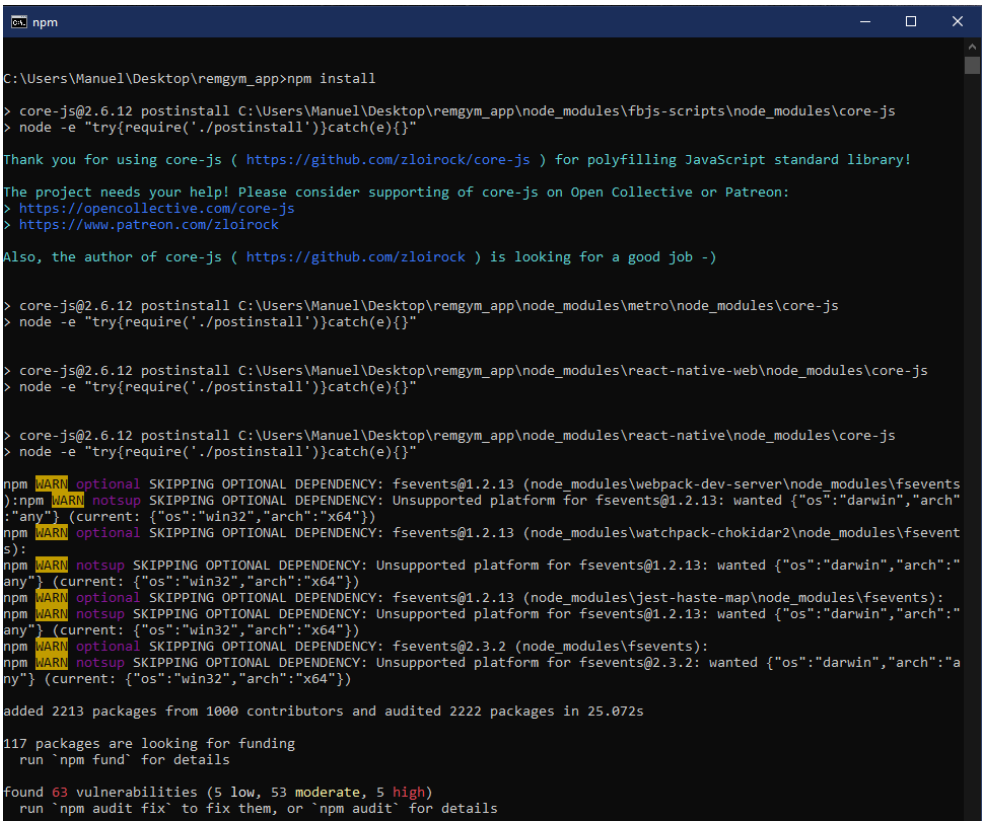

Ilustración 126 – Terminal. Ejecución de npm install para la aplicación móvil

Este comando lo que hace es leer el fichero package.json, que contiene todas las dependencias software que requiere el proyecto para funcionar, y las intenta instalar de forma local en una carpeta llamada node_modules.

Entre todas las dependencias, una muy importante es Expo CLI [65], una aplicación de línea de comandos para gestionar proyectos expo y poder utilizar las herramientas que proporciona.

Una vez hemos instalado todas las dependencias, se puede arrancar el proyecto bajo Expo con el comando `expo start` que, mediante Metro Bundler, arrancará el proyecto. La salida de la ejecución de dicho comando debería ser similar a la mostrada en la Ilustración 127:

Ilustración 127 – Terminal. Ejecución del comando expo start

Paralelamente a esta salida del terminal de comandos, se abrirá una pestaña en el navegador web con la vista de navegador de Metro Bundler, con un aspecto similar al reflejado en la Ilustración 128:

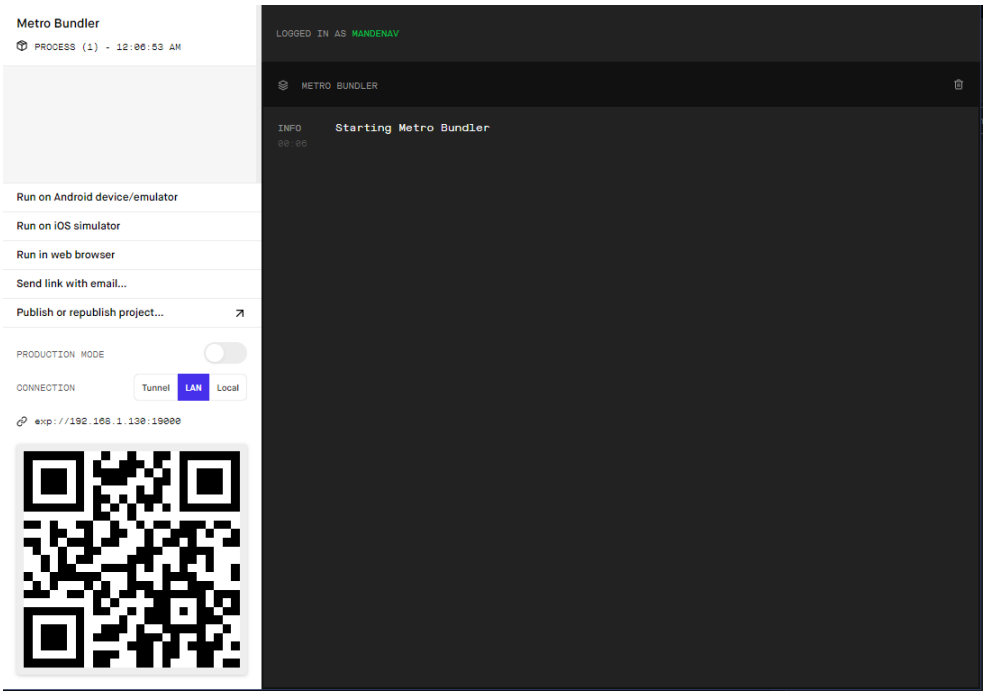

Ilustración 128 – Vista de navegador web de Metro Bundler

Esto nos permitirá desplegar el proyecto en un terminal de forma rápida escaneando el código QR con la aplicación Expo Go para móviles.

A.3.2 Compilación de la aplicación e instalación en un terminal físico

Para compilar la aplicación [66] y generar el fichero .apk que nos permitirá instalarla de forma adecuada en nuestro terminal Android, es necesario haber completado los pasos de instalación del subapartado anterior, de manera que el proyecto se pueda arrancar.

La compilación se realizará mediante la subida del proyecto a la plataforma online de Expo, en la que se compilará y nos dará la opción de descargar el fichero .apk después. Para ello es necesario estar registrado en dicha plataforma, y haber iniciado sesión mediante el comando `expo login`.

El primer paso para compilarla es ejecutar el comando `expo build:android -t apk`-. Tras ejecutarlo, como se muestra en la Ilustración 129, el terminal nos irá solicitando información acerca del proyecto si no la encuentra en el fichero package.json. Puede solicitar el nombre del paquete para Android – en el desarrollo de la aplicación se eligió llamarlo 'tfg.mandenav.remgym' –.

En un punto de la compilación, nos solicitará un fichero keystore, utilizado para almacenar las claves de seguridad. Dará la opción de generar uno en el momento, o cargar uno ya existente. En el desarrollo se dejó que expo generase uno nuevo.

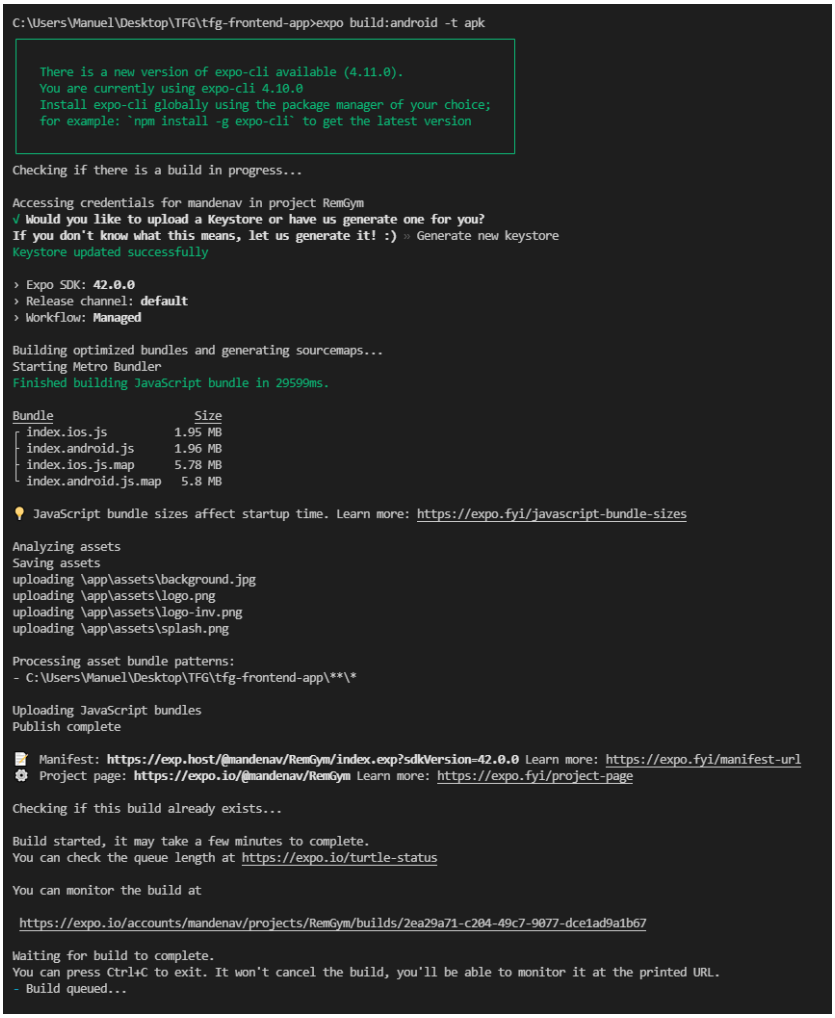

Ilustración 129 – Terminal. Ejecución del comando expo build:android -t apk

Al final de esto, nos ofrecerá un enlace en el que podremos seguir el proceso de compilación en tiempo real, teniendo acceso a un gráfico informando de cuántas compilaciones están en cola, etc. Este proceso se muestra en las Ilustraciones 130 y 131.

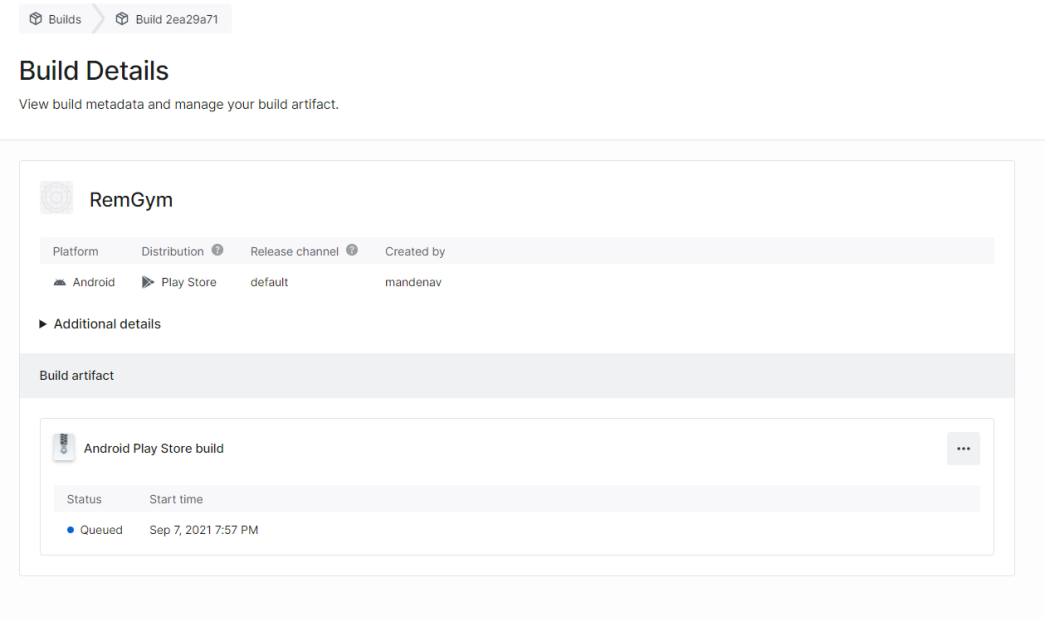

Ilustración 130 – Expo builder. Aplicación en cola

Una vez comience la compilación, nos informará sobre el proceso como si de la salida de un terminal de comandos se tratase, arrojando el resultado de cada paso que realiza:

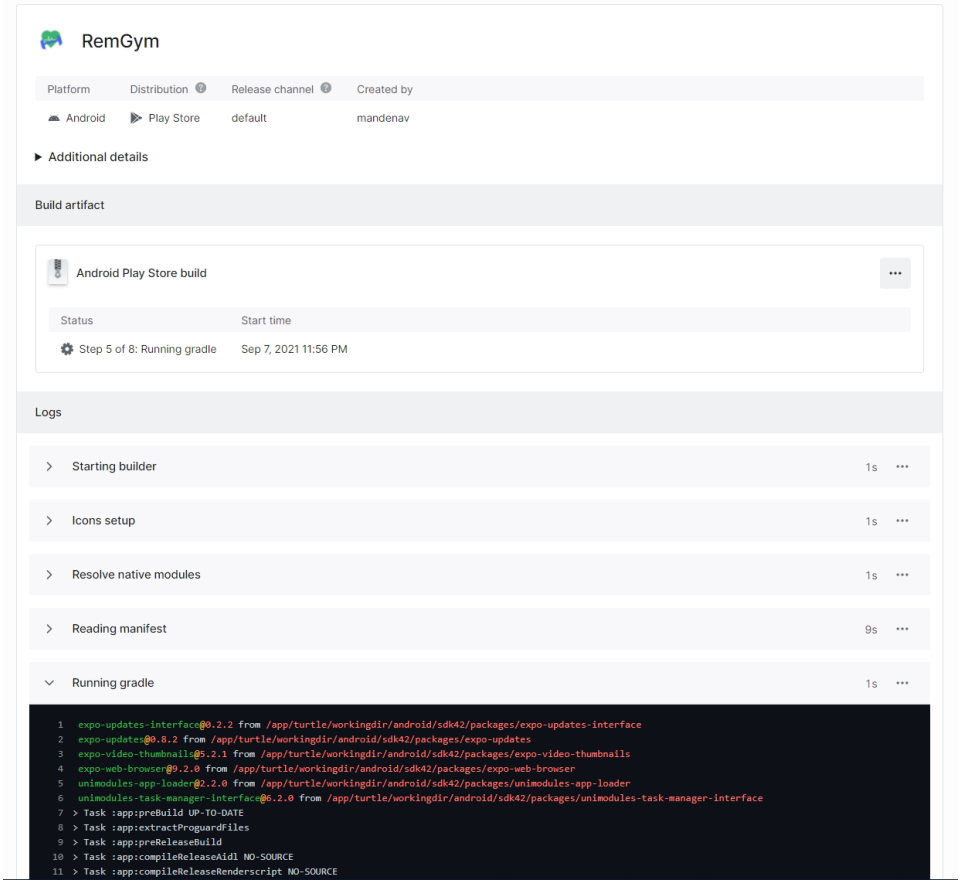

Ilustración 131 – Expo builder. Aplicación compilándose

Una vez terminado, nos dará la opción de descargar el fichero en formato .apk, el cual podrá ser instalado en un dispositivo con SO Android, como se aprecia en la Ilustración 132.

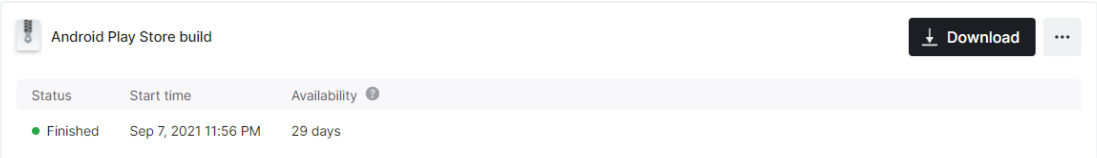

Ilustración 132 – Expo builder. Aplicación disponible para descarga

Nota: Para cualquier modificación sobre la aplicación, habrá que repetir todo el proceso de compilación de nuevo.

A.3.3 Uso de bróker MQTT externo

Como se ha comentado a lo largo del documento, el intercambio de información entre Usuario y Especialista se realiza a través de un componente externo al proyecto. En el código de la aplicación web, en el fichero */app/config/constants.js* hay dos líneas que especifican tanto la dirección ip como el puerto al que realizar la solicitud de conexión, donde estará alojado el bróker MQTT.

Ya que el bróker MQTT externo utilizado está desplegado de forma abierta para cualquier usuario pueda hacer uso de él, no se requiere ningún tipo de configuración adicional ni registro previo. La gestión de la conexión y los 'topics' usados para distinguir los flujos de mensajes están predeterminados en el código.

Sólo en caso de que se pretenda utilizar un bróker MQTT diferente habría que modificar la dirección y el puerto.

REFERENCIAS

[1] Instituto Nacional de Estadística, «Ejercicio físico regular y sedentarismo en el tiempo libre por tipo de ejercicio, CCAA y periodo,» 2017. [En línea]. Available: https://www.ine.es/jaxi/Datos.htm?path=/t00/ICV/dim3/&file=33301.px#!tabs-grafico.

[2] Organización Mundial de la Salud, «Cada movimiento cuenta para mejorar la salud,» 2020. [En línea]. Available: https://www.who.int/es/news/item/25-11-2020-every-move-counts-towards-better-health-says-who.

[3] Sociedad Española de Obesidad, «Un 44% de los españoles aumentaron de peso durante el confinamiento,» 2020. [En línea]. Available: https://www.seedo.es/index.php/sala-de-prensa/noticias-prensa/708-22-07-20-ndp-un-44-de-los-espanoles-aumentaron-de-peso-durante-el-confinamiento.

[4] P. C. Rebollo, «Plataforma Web de prescripción de ejercicios usando Spring,» Universidad de Sevilla, 2018. [En línea]. Available: https://biblus.us.es/bibing/proyectos/abreproy/91731/.

[5] RedHat, Inc, «What is a REST API?,» 2020. [En línea]. Available: https://www.redhat.com/es/topics/api/what-is-a-rest-api.

[6] BezKoder, «Node.js Rest APIs example with Express, Sequelize & MySQL,» 2021. [En línea]. Available: https://www.bezkoder.com/node-js-express-sequelize-mysql/.

[7] World Wide Web Consortium, «The Original HTTP as defined in 1991,» 1991. [En línea]. Available: https://www.w3.org/Protocols/HTTP/AsImplemented.html.

[8] MySQL, Oracle, 2021. [En línea]. Available: https://www.mysql.com/.

[9] MDN Web Docs, «About JavaScript,» Mozilla, 2021. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript?redirectlocale=en-US&redirectslug=JavaScript%2FAbout_JavaScript.

[10] MDN Web Docs, «Introducing asynchronous JavaScript,» Mozilla, 2021. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing.

[11] MDN Web Docs, «Making asynchronous programming easier with async and await,» Mozilla, 2021. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await.

[12] React Native, «React Fundamentals,» Facebook, Inc, 2021. [En línea]. Available: https://reactnative.dev/docs/intro-react.

[13] React Native, [En línea]. Available: https://reactnative.dev/.

[14] React, «Un vistazo a los Hooks,» Facebook Inc., 2021. [En línea]. Available: https://es.reactjs.org/docs/hooks-overview.html.

[15] Tommy Nguyen y Krzysztof, «React Native Async Storage,» 2021. [En línea]. Available: https://github.com/react-native-async-storage/async-storage#readme.

[16] Expo y Software Mansion, «React Navigation,» 2021. [En línea]. Available: https://reactnavigation.org/.

[17] Jamon Holmgren, «React Native Webview - a Modern, Cross-Platform WebView for React Native,» 2021. [En línea]. Available: https://github.com/react-native-webview/react-native-webview#readme.

[18] Expo, «Introduction to Expo,» 2021. [En línea]. Available: https://docs.expo.dev/.

[19] Expo, «Workflows,» 2021. [En línea]. Available: https://docs.expo.dev/introduction/managed-vs-bare/.

[20] Expo, «@expo/vector-icons,» 2021. [En línea]. Available: https://github.com/expo/vector-icons#readme.

[21] Expo, «Constants,» 2021. [En línea]. Available: https://docs.expo.dev/versions/latest/sdk/constants/.

[22] Expo, «Location,» 2021. [En línea]. Available: https://docs.expo.dev/versions/latest/sdk/location/.

[23] Expo, «SecureStore,» 2021. [En línea]. Available: https://docs.expo.dev/versions/latest/sdk/securestore/.

[24] Expo, «TaskManager,» [En línea]. Available: https://docs.expo.dev/versions/latest/sdk/task-manager/] .

[25] Infinite Red, Inc, «Apisauce,» 2021. [En línea]. Available: https://github.com/infinitered/apisauce#readme.

[26] Formium, «Formik,» 2020. [En línea]. Available: https://formik.org/] .

[27] Auth0, «jwt-decode is a small browser library that helps decoding JWTs token which are Base64Url encoded,» 2021. [En línea]. Available: https://github.com/auth0/jwt-decode#readme.

[28] introvertuous-Fun (GitHub), «React Native Mqtt,» 2018. [En línea]. Available: https://github.com/Introvertuous-Fun/react-native-mqtt#readme.

[29] Jason Quense, «Yup,» 2016. [En línea]. Available: https://github.com/jquense/yup.

[30] OpenJS Foundation, «About Node.js,» 2021. [En línea]. Available: https://nodejs.org/en/about/.

[31] OpenJS Foundation, «Express - Infraestructura web rápida, minimalista y flexible para Node.js,» 2021. [En línea]. Available: https://expressjs.com/es/.

[32] Nick Campbell, «node.bcrypt.js,» 2021. [En línea]. Available: https://github.com/kelektiv/node.bcrypt.js#readme] .

[33] Scott Motte, «dotenv,» 2021. [En línea]. Available: https://github.com/motdotla/dotenv#readme.

[34] Auth0, «jsonwebtoken,» 2021. [En línea]. Available: https://github.com/auth0/node-jsonwebtoken#readme.

[35] Andrey Sidorov, «Node MySQL 2,» 2021. [En línea]. Available: https://github.com/sidorares/node-mysql2#readme.

[36] Remy Sharp, «Nodemon reload, automatically,» 2021. [En línea]. Available: https://nodemon.io/.

[37] Sequelize.org, «Sequelize ORM,» 2021. [En línea]. Available: https://sequelize.org/.

[38] Sequelize.org, «Sequelize-Auto,» 2021. [En línea]. Available: https://github.com/sequelize/sequelize-auto#readme.

[39] OASIS, «MQTT Version 5.0,» 2019. [En línea]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[40] Eclipse Newsletter, «MQTT 101 - How to Get Started with the lightweight IoT Potocol,» 2014. [En línea]. Available: https://www.eclipse.org/community/eclipse_newsletter/2014/october/article2.php.

[41] JWT, Auth0, 2021. [En línea]. Available: https://jwt.io/].

[42] Internet Engineering Task Force (IETF), «JSON Web Token (JWT),» 2015. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc7519.

[43] The USENIX Association, «A Future-Adaptable Password Scheme,» 1999. [En línea]. Available: https://www.usenix.org/legacy/event/usenix99/provos/provos.pdf.

[44] USENIX - The Advanced Computing System Association, 2021. [En línea]. Available: https://www.usenix.org/.

[45] Bruce Schneier, «Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish),» 1994. [En línea]. Available: https://www.schneier.com/academic/archives/1994/09/description_of_a_new.html.

[46] Microsoft, «Visual Studio Code - Getting Started,» 2021. [En línea]. Available: https://code.visualstudio.com/docs.

[47] Git, «git --everything-is-local,» 2021. [En línea]. Available: https://git-scm.com/.

[48] GitHub, Inc, «GitHub Desktop,» 2021. [En línea]. Available: https://desktop.github.com/.

[49] Expo, «Expo Go,» 2021. [En línea]. Available: https://expo.dev/client.

[50] Facebook, Inc, «Metro - The JavaScript bundler for React Native,» 2020. [En línea]. Available: https://facebook.github.io/metro/.

[51] Google, «Android Studio,» 2021. [En línea]. Available: https://developer.android.com/studio/features.

[52] Vysor, Inc, «Vysor - A Windows to your Phone,» 2021. [En línea]. Available: https://www.vysor.io/.

[53] Postman, 2021. [En línea]. Available: https://www.postman.com/.

[54] HiveMQ GmbH, 2021. [En línea]. Available: https://www.hivemq.com/.

[55] JGraph Ltd, 2021. [En línea]. Available: https://www.diagrams.net/.

[56] Pixabay, 2021. [En línea]. Available: https://pixabay.com/es/.

[57] Donny (Usuario en dev.to), «What's this Model-View-Controller MVC thing? I never did quite understand it!,» 2020. [En línea]. Available: https://dev.to/kuddleman/what-s-this-model-view-controller-mvc-thing-i-never-did-quite-understand-it-333.

[58] Baeldung, «The DAO Pattern in Java,» 2020. [En línea]. Available: https://www.baeldung.com/java-dao-pattern.

[59] Deloitte, «¿Qué es un ORM?,» 2021. [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html.

[60] Expo, «Expo - Limitaciones,» 2021. [En línea]. Available: https://docs.expo.dev/introduction/why-not-expo/.

[61] Heroku, «What is Heroku?,» 2021. [En línea]. Available: https://www.heroku.com/what.

[62] npm Docs, «npm-install,» 2021. [En línea]. Available: https://docs.npmjs.com/cli/v7/commands/npm-install .

[63] Heroku Dev Center, «Deploying Node.js Apps on Heroku,» 2021. [En línea]. Available: https://devcenter.heroku.com/articles/deploying-nodejs.

[64] Heroku Dev Center, «The Heroku CLI,» 2021. [En línea]. Available: https://devcenter.heroku.com/articles/heroku-cli.

[65] Expo, «Expo CLI,» 2021. [En línea]. Available: https://docs.expo.dev/workflow/expo-cli/.

[66] Expo, «Building Standalone Apps,» 2021. [En línea]. Available: https://docs.expo.dev/distribution/building-standalone-apps/.