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

Aplicación móvil para la gestión de listas de ejercicios de entrenamiento en React Native

Autor: María de los Reyes Machuca Romero
Tutor: María Teresa Ariza Gómez

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

Sevilla, 2022

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

Aplicación móvil para la gestión de listas de ejercicios de entrenamiento en React Native

Autor:
María de los Reyes Machuca Romero

Tutor:
María Teresa Ariza Gómez
Profesora titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo de Fin de Grado: Aplicación móvil para la gestión de listas de ejercicios de entrenamiento en React Native

Autor: María de los Reyes Machuca Romero

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, 2022

El Secretario del Tribunal

A mi familia

A mis amigos

A mis maestros

Agradecimientos

Me gustaría agradecer a las personas que me han apoyado durante estos meses tan difíciles y a la vez tan valiosos en los que he estado elaborando este proyecto.

En primer lugar, a los docentes cuya enseñanza inspira y motiva a los alumnos a aprender sobre su asignatura. Especialmente agradecer a Teresa, por haberme dado las pautas necesarias para llevar a cabo este proyecto y por guiarme en el proceso.

En segundo lugar, a mis amigos y compañeros, que me han ayudado durante la etapa universitaria y que han hecho que esta experiencia sea inolvidable. También quiero agradecer a mi pareja, por prestarme siempre todo su apoyo y por aportarme ideas para el proyecto.

En tercer lugar, a mi familia, que siempre me levantan los ánimos y me acompañan en todos los momentos de la vida, y sin los cuales no habría tenido la oportunidad de estudiar esta carrera. Me gustaría destacar el esfuerzo de mis padres, que se han desvivido por otorgar un futuro a mis hermanos y a mí.

En último lugar, quiero dedicar este proyecto a unas personas muy importantes para mí, mis abuelos y mi madre. Me gustaría que pudieran estar aquí para presenciar este momento y volver a sentir su apoyo incondicional. Gracias a ellos, en especial a mi madre, me he convertido en la persona que soy y espero que se sientan muy orgullosos de mí.

María de los Reyes Machuca Romero

Sevilla, 2022

Resumen

Actualmente, la tecnología ha avanzado a pasos agigantados y cada vez más personas utilizan teléfonos inteligentes. Estos dispositivos nos permiten realizar muchas tareas esenciales en el día a día gracias a aplicaciones móviles, llamadas "apps" de forma abreviada. Existen apps que nos ayudan en todo tipo de áreas, como la medicina, la banca, la interacción social o el ejercicio físico. Nos permiten ahorrar tiempo ya que, por ejemplo, no es necesario acudir presencialmente al supermercado para hacer la compra o al banco para realizar una transferencia, debido a que hay apps creadas específicamente para ello.

Una de las razones por las que esta tecnología es alcanzable para todos es que muchas aplicaciones están desarrolladas para varias plataformas y diversos sistemas operativos. A su vez, esto aumenta la competencia entre las compañías desarrolladoras en el mercado, y hoy en día tenemos millones de aplicaciones a nuestra disposición.

Debido a esto, surge la necesidad de desarrollar apps invirtiendo el menor tiempo posible para estar al día con la demanda actual. Para ello, se han creado entornos de desarrollo de aplicaciones multiplataforma, que permiten usarlas en diferentes dispositivos y sistemas operativos utilizando un solo código fuente.

El entorno en el que se profundizará en este documento es React Native, analizando muchos de sus componentes en el lenguaje de programación JavaScript. Se utilizará Expo como plataforma de creación, despliegue y pruebas de la app. Mediante los módulos de Expo se crearán y compartirán listas de ejercicios en formato JSON. Las listas se almacenarán de forma local en el dispositivo usando funciones asíncronas de JavaScript. Finalmente, se utilizará Visual Studio Code como herramienta de desarrollo de esta tecnología.

Abstract

Nowadays, technology has advanced by leaps and bounds and more people are using smartphones every time. These devices allow us to perform many essential daily tasks thanks to mobile applications, called “apps” for short. There are apps that help us in all types of areas, such as medicine, the bank, social interaction, or physical exercise. They allow us to save time since, for example, it is not necessary to go grocery shopping to buy day-to-day essentials or to go to the bank to transfer money, because there are apps created specifically for that purpose.

One of the reasons why this technology is accessible to everyone is that many applications are developed for several platforms and various operating systems. In turn, this increases competition between developer companies in the marketplace and, at present, we have millions of applications at our disposal.

Because of this, the need to develop apps investing the shortest time possible is emerging to keep up with the current demand. To this effect, development environments of multiplatform applications have been created, that allow to use them in different devices and operating systems using only a single source code.

The environment that will undergo an in-depth analysis in this document is React Native, analyzing many of its components in JavaScript programming language. Expo will be used as a creation, deployment, and testing platform. Lists of exercises will be created and shared in JSON format using Expo modules. The lists will be stored locally in the device using JavaScript asynchronous functions. Finally, Visual Studio Code will be used as a development tool for this technology.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xiv**

Índice de Tablas **xvii**

Índice de Figuras **xix**

1 Introducción **23**
1.1 Motivación *23*
1.2 Objetivos *24*
1.3 Descripción de la solución *25*
1.4 Estructura de la memoria *26*

2 Tecnologías utilizadas **27**
2.1 JavaScript *27*
2.1.1 Descripción y características generales 27
2.1.2 Funciones 28
2.1.3 JavaScript asíncrono 28
2.1.4 Promise 29
2.1.5 Async/await 30
2.2 React Native *30*
2.2.1 Descripción y características generales 30
2.2.2 Componentes de React 31
2.2.3 Hooks 32
2.2.4 AsyncStorage 34
2.3 Expo *35*
2.3.1 Descripción y características generales 35
2.3.2 Instalación y ejecución del entorno 35
2.3.3 Flujos de trabajo 36
2.4 JSON *37*
2.4.1 Descripción y características generales 37
2.4.2 Uso en la aplicación 37

3 Herramientas utilizadas **40**
3.1 Herramientas de desarrollo *40*
3.1.1 Visual Studio Code 40
3.1.2 Expo Go 41
3.1.3 Git 43
3.2 Otras herramientas utilizadas *45*
3.2.1 Vysor 45
3.3 Herramientas para el modelado y diseño *46*
3.3.1 Diagrams.net 46
3.3.2 Visual Paradigm 47
3.4 Recursos adicionales *48*

3.4.1 Pixabay 48
3.4.2 Pexels 49

4 Arquitectura y análisis 50
4.1 Casos de uso generales 50
4.1.1 Identidad de los actores 50
4.1.2 Autenticación de un usuario 50
4.1.3 Operaciones para crear un elemento 52
4.1.4 Operaciones para modificar un elemento 53
4.1.5 Operaciones con una lista 55
4.2 Diagramas de secuencia 57
4.2.1 Registro e inicio de sesión 57
4.2.2 Crear una lista 58
4.2.3 Realizar entrenamiento de una lista 60
4.2.4 Enviar una lista por correo electrónico 62
4.2.5 Importar una lista 62
4.3 Modelo de datos 63

5 Interfaz de usuario y funcionalidad 66
5.1 Pantalla de registro e inicio de sesión 66
5.2 Pantalla de modo de visualización 68
5.3 Pantalla de lista de listas 69
5.3.1 Añadir fondo de lista de listas 70
5.3.2 Eliminar listas 71
5.3.3 Duplicar una lista 72
5.3.4 Insertar una lista 73
5.3.5 Abrir aplicación de Gmail y YouTube 74
5.3.6 Crear una lista nueva 74
5.3.7 Cambiar el correo electrónico del usuario 75
5.4 Pantalla de crear lista 75
5.4.1 Pantalla de nueva lista local 76
5.4.2 Pantalla de nueva lista 80
5.5 Pantalla de lista de ejercicios modo Entrenamiento 80
5.5.1 Editar título de una lista 82
5.5.2 Marcar una lista como realizada 82
5.5.3 Añadir recordatorio de una lista 84
5.5.4 Añadir fondo de lista 86
5.5.5 Eliminar una lista 87
5.5.6 Enviar una lista 88
5.5.7 Modificar ejercicios de una lista 92
5.6 Pantalla entrenamiento de una lista 97
5.7 Pantalla de historial de una lista 98
5.8 Pantalla de lista de ejercicios modo Configuración 101
5.8.1 Añadir un ejercicio a una lista creada 102
5.8.2 Eliminar un ejercicio de una lista 103
5.8.3 Modificar el correo electrónico 104
5.9 Salir de la aplicación 105

6 Instalación y pruebas 107
6.1 Instalación y pruebas en Android 107
6.1.1 Generación del archivo de instalación 107
6.2 Instalación y pruebas en iOS 109

7 Líneas de mejora y conclusiones 113

Anexo A: Instalación y despliegue 115
A.1 Instalación del entorno 115

A.2 Despliegue del proyecto con GitHub *119*

Anexo B: Código de las funciones para enviar una lista **122**

Anexo C: Código de las funciones para realizar un entrenamiento de una lista **126**

Referencias **143**

ÍNDICE DE TABLAS

Tabla 1. CU-01: Registrarse con nombre y correo electrónico 51
Tabla 2. CU-02: Crear un elemento nuevo 53
Tabla 3. CU-03: Modificar un elemento existente 55
Tabla 4. CU-04: Ver lista de elementos 56
Tabla 5. CU-05: Realizar entrenamiento de una lista 56
Tabla 6. CU-06: Enviar una lista por correo electrónico o Bluetooth 57

ÍNDICE DE FIGURAS

Ilustración 1 - Evolución del ejercicio físico por año, edad, sexo y nivel educativo (2010-2020). Ministerio de Cultura y Deporte 23
Ilustración 2 - Porcentaje de población sedentaria en España (2020). INE 24
Ilustración 3 - Arquitectura de la aplicación 25
Ilustración 4 - Logo de JavaScript 27
Ilustración 5 - Definición de una función en JavaScript 28
Ilustración 6 - Función setTimeout 28
Ilustración 7 - Uso de la función setInterval de Entrenamiento.js 29
Ilustración 8 - Creación y uso del objeto Promise 29
Ilustración 9 - Función utilizando async/await 30
Ilustración 10 - Logo de React Native 30
Ilustración 11 - Gráfico de la evolución de React Native con respecto a Android e iOS (2016-2018) 31
Ilustración 12 - Componente Video de React Native 31
Ilustración 13 - Componentes de clase y de función 32
Ilustración 14 - Ejemplo de useEffect 33
Ilustración 15 - useNavigation y useFocusEffect en Lista.js 34
Ilustración 16 - Uso de AsyncStorage en AllLists.js 35
Ilustración 17 - Logo de Expo 35
Ilustración 18 - Expo Development Server 36
Ilustración 19 - Logo de JSON 37
Ilustración 20 - Conversión JSON a objeto JavaScript 37
Ilustración 21 - Lista de ejercicios en formato JSON 38
Ilustración 22 - Logo de Visual Studio Code 40
Ilustración 23 - Interfaz de usuario de Visual Studio Code 41
Ilustración 24 - Logo de Expo Go 41
Ilustración 25 - Funcionamiento de Expo Go 42
Ilustración 26 - Interfaz de usuario de Expo Go 43
Ilustración 27 - Logo de Git 43
Ilustración 28 - Logo de GitHub Desktop 44
Ilustración 29 - Interfaz gráfica de GitHub Desktop 44
Ilustración 30 - Logo de Vysor 45
Ilustración 31 - Interfaz de usuario de Vysor 45
Ilustración 32 - Logo de Diagrams.net 46

Ilustración 33 - Recursos y plantillas de Diagrams.net 46
Ilustración 34 - Página de Diagrams.net 47
Ilustración 35 - Logo de Visual Paradigm 47
Ilustración 36 - Interfaz de usuario de Visual Paradigm 48
Ilustración 37 - Logo de Pixabay 48
Ilustración 38 - Logo de Pexels 49
Ilustración 39 - Casos de uso para la autenticación de un usuario 51
Ilustración 40 - Diagrama de casos de uso para la creación de un elemento 52
Ilustración 41 - Diagrama de casos de uso para modificar un elemento 54
Ilustración 42 - Diagrama de casos de uso para realizar operaciones con una lista 55
Ilustración 43 - Diagrama de secuencia para el registro e inicio de sesión 58
Ilustración 44 - Diagrama de secuencia para crear una lista 59
Ilustración 45 - Diagrama de secuencia para realizar un entrenamiento 61
Ilustración 46 - Diagrama de secuencia para enviar una lista por correo electrónico 62
Ilustración 47 - Diagrama de secuencia para importar una lista 63
Ilustración 48 - Modelo de datos de la aplicación 64
Ilustración 49 - Pantalla de registro e inicio de sesión 66
Ilustración 50 - Ejemplo de nombre y correo electrónico del usuario 67
Ilustración 51 - Pantalla de modo de visualización 68
Ilustración 52 - Pantalla de lista de listas 69
Ilustración 53 - Menú de la pantalla de listas de listas 70
Ilustración 54 - Ejemplo de fondo de pantalla personalizado 71
Ilustración 55 - Opción eliminar listas 71
Ilustración 56 - Opción duplicar lista 72
Ilustración 57 - Ejemplo de duplicar lista 73
Ilustración 58 - Insertar una lista existente 74
Ilustración 59 - Ejemplo de alerta de Crear lista 75
Ilustración 60 - Pantalla de nueva lista local 76
Ilustración 61 - Alerta para seleccionar una imagen 77
Ilustración 62 - Imágenes del almacenamiento interno 78
Ilustración 63 - Ventana de iconos de la aplicación 78
Ilustración 64 - Ejemplo de añadir ejercicio con vídeo local 79
Ilustración 65 - Pantalla de nueva lista 80
Ilustración 66 - Pantalla de lista de ejercicios del modo Entrenamiento 81
Ilustración 67 - Editar título de una lista 82
Ilustración 68 - Menú de la lista de ejercicios modo Entrenamiento 83
Ilustración 69 - Ejemplo de marcar lista como realizada 83
Ilustración 70 - Ventana de añadir recordatorio 84
Ilustración 71 - Ejemplo de calendario para añadir recordatorio 85

Ilustración 72 - Ejemplo de hora para añadir recordatorio 85
Ilustración 73 - Ejemplo de notificación de recordatorio de una lista 86
Ilustración 74 - Ejemplo de fondo personalizado para una lista 87
Ilustración 75 - Ejemplo de alerta para eliminar una lista 88
Ilustración 76 - Menú de opciones para enviar una lista 89
Ilustración 77 - Ejemplo de envío de una lista por Gmail 90
Ilustración 78 - Ejemplo de envío de una lista por Bluetooth 91
Ilustración 79 - Menú desplegable de un ejercicio 92
Ilustración 80 - Ventana para añadir un emoticono a un ejercicio 93
Ilustración 81 - Ejemplo de ejercicio realizado con emoticono 94
Ilustración 82 - Ejemplo de añadir foto a un ejercicio 95
Ilustración 83 - Ventana de añadir foto a un ejercicio 96
Ilustración 84 - Ejemplo de foto en un ejercicio 96
Ilustración 85 - Pantalla de entrenamiento de una lista 97
Ilustración 86 - Ejemplo de entrenamiento pulsando comenzar 98
Ilustración 87 - Ejemplo de historial de una lista 99
Ilustración 88 - Ejemplo de un historial de entrenamiento 100
Ilustración 89 - Ejemplo de historial vacío de una lista 101
Ilustración 90 - Ejemplo de pantalla lista en el modo Configuración 102
Ilustración 91 - Ejemplo de añadir un ejercicio local 103
Ilustración 92 - Ejemplo de añadir un ejercicio de YouTube 103
Ilustración 93 - Ejemplo de eliminar un ejercicio de la lista 104
Ilustración 94 - Ventana para modificar el correo electrónico de la lista 105
Ilustración 95 - Alerta para salir de la aplicación 106
Ilustración 96 - Modificación del archivo app.json para generar el .APK 108
Ilustración 97 - Comando para generar el .APK 108
Ilustración 98 - Instalación de la aplicación en un dispositivo Android 109
Ilustración 99 - Actualización de Expo SDK para compatibilidad con Expo Go 109
Ilustración 100 - Inicio del servidor de desarrollo de Expo y generación de código QR para Expo Go 110
Ilustración 101 - Pantalla de iniciar sesión en iOS 111
Ilustración 102 - Pantalla de lista de listas en iOS 111
Ilustración 103 - Creación de una lista nueva en iOS 111
Ilustración 104 - Pantalla Entrenamiento en iOS 111
Ilustración 105 - Ejemplo de seleccionar fecha para un recordatorio en iOS 112
Ilustración 106 - Pantalla Lista con notificación en iOS 112
Ilustración 107 - Descargar instalador de Node.js para Windows 115
Ilustración 108 - Ejecutar instalador de Node.js 116
Ilustración 109 - Comprobar versión de Node.js y npm 116
Ilustración 110 - Descargar archivo de instalación de Visual Studio Code para Windows 117

Ilustración 111 - Instalar Visual Studio Code 117
Ilustración 112 - Comprobar versión de Visual Studio Code 118
Ilustración 113 - Instalación de Expo CLI 118
Ilustración 114 - Creación de un proyecto React Native en JavaScript utilizando Expo CLI 118
Ilustración 115 - Inicialización de la aplicación con npm 119
Ilustración 116 - Página de descarga de GitHub Desktop 119
Ilustración 117 - Acceder a una cuenta de GitHub con GitHub Desktop 120
Ilustración 118 - Clonar repositorio con GitHub Desktop 121

1 INTRODUCCIÓN

El frenético ritmo de vida junto con el avance en las tecnologías ha provocado un aumento en el uso de los teléfonos inteligentes para realizar todo tipo de actividades. Desde hace unos años, se ha podido observar un cambio en la forma de relacionarnos, de trabajar y de gestionar distintos aspectos de nuestra vida gracias a la llegada de estos dispositivos y a la creación de aplicaciones móviles. Además, muchas de estas aplicaciones o apps se pueden utilizar también en otras plataformas, como televisores "Smart tv" con conexión a internet o tabletas, con el fin de abarcar varios tipos de ámbitos y usuarios. Esto es posible gracias a entornos que permiten desarrollar aplicaciones multiplataforma para distintos sistemas operativos.

1.1 Motivación

La actividad física ha ido incrementando en la última década según la encuesta de hábitos deportivos en España [1] realizada por el Ministerio de Cultura y Deporte. En la Ilustración 1 se muestra la evolución del ejercicio físico practicado semanalmente por hombres y mujeres a partir de los 15 años.

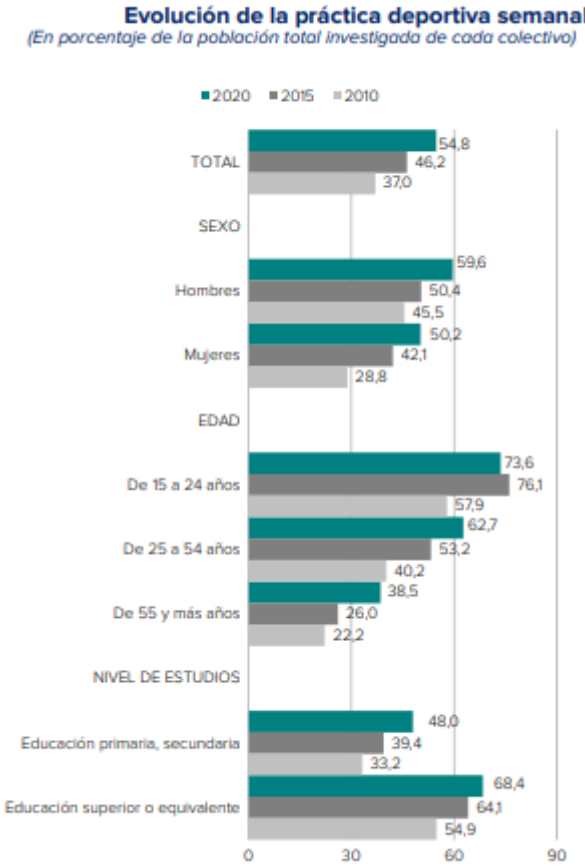

Ilustración 1 – Evolución del ejercicio físico por año, edad, sexo y nivel educativo (2010-2020).

Ministerio de Cultura y Deporte

Se puede observar un aumento considerable en la práctica deportiva en el último año. Sin embargo, según la Encuesta Europea de Salud en España del Instituto Nacional de Estadística [2], sigue habiendo un

elevado porcentaje de sedentarismo en la población. En la Ilustración 2 se puede apreciar que el nivel de sedentarismo es similar en hombres y mujeres.

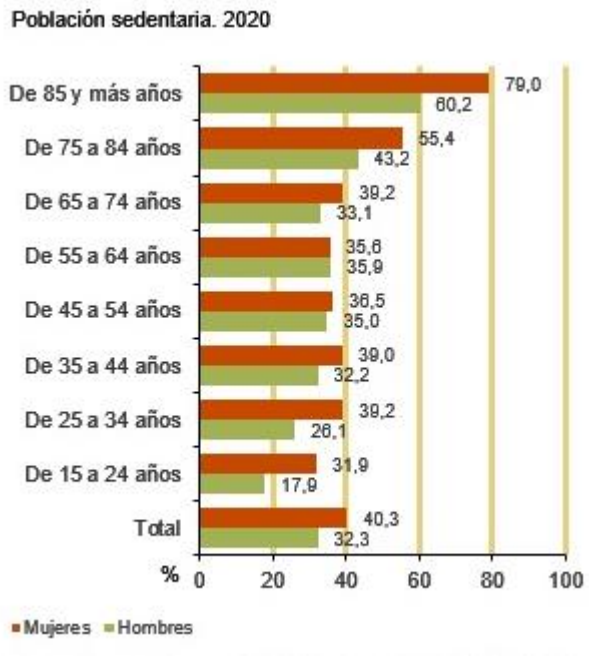

Ilustración 2 – Porcentaje de población sedentaria en España (2020). INE

Al comparar la Ilustración 1 y 2 se puede concluir que, aunque más de la mitad de la población realiza deporte cada semana, hay un notable grado de sedentarismo que, en parte, se debe al periodo de pandemia de Covid-19, en el cual aumentó la inactividad física debido al cese de actividades deportivas en el exterior.

Otra de las razones por las que hay una importante falta de ejercicio es la escasez de tiempo y el desconocimiento a la hora de realizarlo. La mayoría de los deportes que se practican necesitan de unas pautas y, con el ritmo de vida que hay actualmente, es difícil dedicarles tiempo.

Se plantea una idea en este contexto con el objetivo de aumentar y promover el hábito de ejercicio físico, bajo la supervisión de un especialista en actividad física. El usuario podrá realizar ejercicio desde casa teniendo en cuenta sus indicaciones y llevar un seguimiento que le garantice unos resultados óptimos sin invertir más tiempo del necesario. También podrá crear sus propios entrenamientos y marcar sus propios tiempos.

1.2 Objetivos

Durante el desarrollo del proyecto, los objetivos iniciales se han ido variando y modificando, ya que algunos resultaron no ser muy convenientes o eran inalcanzables.

Los objetivos se basan principalmente en explorar las tecnologías y funcionalidades que ofrece React Native para desarrollar una aplicación móvil que funcione de forma local, sin servidor, y son los siguientes:

- Generar un código único en el lenguaje JavaScript que permita usar la aplicación en Android y en iOS.
- Comunicación entre Usuario y Especialista mediante correo electrónico y Bluetooth.
- Almacenar los datos del usuario en una base de datos local.
- Reproducción de listas de entrenamiento con vídeos de YouTube y vídeos del almacenamiento local.

1.3 Descripción de la solución

En vista de lo anterior, se propone una aplicación móvil para realizar listas de entrenamientos para la prescripción de ejercicio físico, orientada a usuarios y especialistas. Esta aplicación, denominada My GymList, tiene como función principal proporcionar un servicio personalizado para cada usuario que maximice su nivel de actividad física. Para ello, se realizará un seguimiento periódico de su rendimiento y así lograr sus objetivos físicos y, en consecuencia, un estilo de vida más saludable. En la Ilustración 3 se muestra la arquitectura de la aplicación:

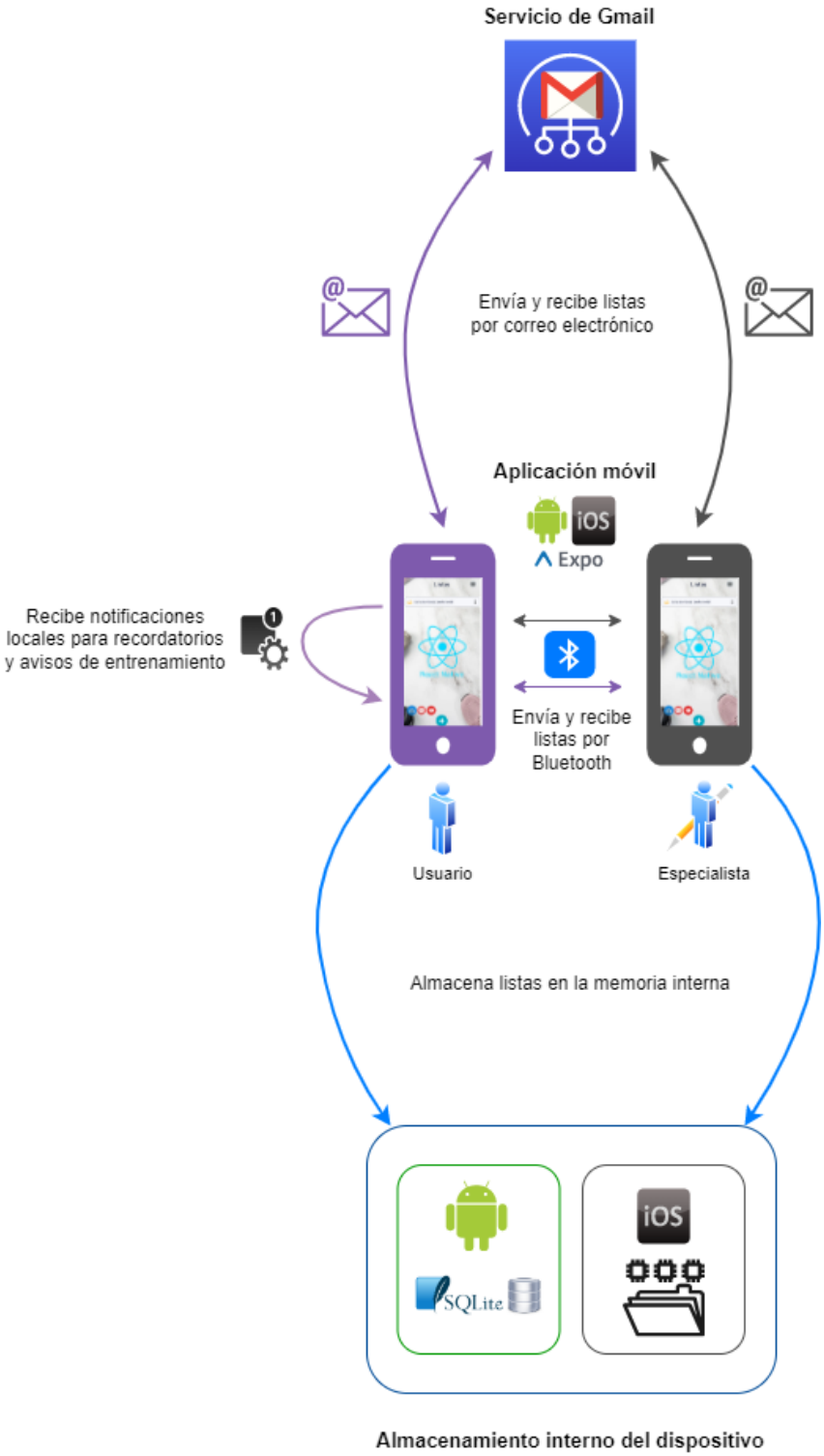

Ilustración 3 – Arquitectura de la aplicación

En la arquitectura se puede observar que la funcionalidad y la comunicación entre los usuarios se realiza sin

servidor. El Usuario y Especialista utilizan la misma aplicación móvil desarrollada en React Native con Expo para crear las listas de ejercicios. En función del sistema operativo del dispositivo que usen, se guardarán en una base de datos local SQLite en el caso de Android, o en unas carpetas del almacenamiento interno en el caso de iOS.

Los dos tipos de usuario pueden enviar las listas por correo electrónico utilizando Gmail y por Bluetooth. Las listas se envían en un archivo .TXT y el formato es JSON, que consta de varios campos, entre los que se encuentran enlaces a los vídeos, la información relacionada con los ejercicios que aparecen y el historial de entrenamiento.

Por último, en el caso de que un usuario quiera recibir una notificación para realizar un entrenamiento de una lista, puede configurarla para recibirla un día concreto de la semana y que esta se repita de forma periódica. Estas notificaciones son locales a la aplicación.

1.4 Estructura de la memoria

La memoria de este proyecto presenta la siguiente estructura:

- **Introducción:** en este apartado se explica de forma general los objetivos, el contexto, la razón de llevar a cabo este proyecto y cómo se organiza la aplicación.
- **Tecnologías utilizadas:** se explican todos los recursos que se han utilizado para la creación de la app.
- **Herramientas utilizadas:** aparecen los medios empleados para desarrollar y visualizar la aplicación.
- **Arquitectura y análisis:** se detalla y analiza la arquitectura de la aplicación, sus secciones y la relación entre ellas.
- **Interfaz de usuario y funcionalidad:** se muestra la funcionalidad de la app desde el punto de vista del usuario final del servicio.
- **Pruebas:** se presenta una demostración de cómo se visualiza la aplicación desde un dispositivo físico.
- **Líneas de mejora y conclusiones:** en este capítulo se indican los resultados, se plantean posibles mejoras y se define una conclusión del proyecto.

2 TECNOLOGÍAS UTILIZADAS

La tecnología nos facilita reducir las barreras
de la distancia y el tiempo.
- Emily Greene Balch -

En este capítulo se detallarán las tecnologías empleadas para el desarrollo del proyecto y su finalidad.

2.1 JavaScript

Ilustración 4 – Logo de JavaScript

2.1.1 Descripción y características generales

JavaScript [3] es un lenguaje de programación interpretado y dinámico que ocupa pocos recursos y que se utiliza principalmente para desarrollar páginas web. Para ello, se crean scripts en el lado del cliente que se ejecutan de forma automática. Estos interactúan con el usuario y permiten que las páginas sean dinámicas. Se integra con HTML, es de código abierto y se puede utilizar para aplicaciones multiplataforma.

Este lenguaje tiene una especificación propia llamada ECMAScript que garantiza la interoperabilidad de páginas webs en diferentes navegadores. Además, han aparecido nuevos lenguajes de programación en los últimos años que se convierten a JavaScript antes de ejecutarse en el navegador. Esto ha permitido que desarrolladores puedan programar en distintos lenguajes y convertirlos de forma automática. Algunos de estos lenguajes de código abierto son:

- TypeScript, que es un lenguaje tipado que extiende a JavaScript y permite añadir tipos, creado por Microsoft.
- Flow, un corrector de tipos para JavaScript que identifica errores durante la programación, que está creado por Facebook.
- Dart, un lenguaje optimizado para desarrollar apps en cualquier plataforma web o móvil, creado por Google.
- Kotlin, un lenguaje estático usado por desarrolladores de Android que mejora la productividad y la seguridad del código, creado por JetBrains.

Hoy en día, JavaScript contiene una gama de frameworks o marcos de trabajo y librerías que se utilizan para simplificar los proyectos más complejos, como AngularJS, jQuery y ReactJS.

En este proyecto se han empleado muchas de las librerías y objetos de JavaScript, entre los cuales podemos destacar los siguientes:

2.1.2 Funciones

Las funciones [4] son unos bloques de código que están diseñados para ejecutar una tarea específica. Una función es ejecutada cuando es llamada desde el código, de forma automática si se llama a sí misma o cuando ocurra un evento. Es una forma de reutilizar el código, es decir, una vez definida se puede usar varias veces. Un ejemplo de la definición de una función se muestra en la Ilustración 5:

```
function myFunction(parámetro1, parámetro2) {
  return parámetro1 * parámetro2;
}

let x = myFunction(2, 3);
```

Ilustración 5 – Definición de una función en JavaScript

Para llamarla, se ha declarado una variable x que llama a la función con dos parámetros. La función realiza la multiplicación de los parámetros en la llamada y devolverá el valor 6 en este caso.

2.1.3 JavaScript asíncrono

La programación asíncrona [5] es una técnica que permite a un programa empezar una tarea prolongada y, al mismo tiempo, poder responder a otros eventos mientras se ejecuta dicha tarea. Ofrece la posibilidad de realizar tareas sin esperar a que estas terminen para poder ejecutar otra parte del código. Algunas de las funciones más destacables son: acceder a la cámara del usuario y al explorador de archivos o establecer un temporizador y un intervalo de tiempo.

La función setTimeout() se utiliza para crear un temporizador que ejecuta una función o parte de un código una vez que el temporizador expire. En la Ilustración 6 se puede observar un ejemplo:

```
setTimeout(() => {
  console.log("Hola Mundo.");
}, "1000")
```

Ilustración 6 – Función setTimeout

En este ejemplo se imprime “Hola Mundo.” en la consola de JavaScript después de 1 segundo de espera.

La función setInterval() realiza la misma tarea que setTimeOut() pero permite llamar a una función en intervalos específicos en milisegundos. Esta función se sigue ejecutando hasta que se llama a clearInterval(). En la Ilustración 7 se muestra un ejemplo que aparece en *Entrenamiento.js*:

```
const startTimer = () => {
  setCustomInterval(
    setInterval(() => {
      changeTime();
    }, 1000)
  );
}

const stopTimer = () => {
  if(customInterval){
    clearInterval(customInterval);
  }
}
```

Ilustración 7 – Uso de la función setInterval de Entrenamiento.js

En este ejemplo se ejecuta una función llamada startTimer que inicia un cronómetro. Para ello, se establece un intervalo con setInterval() donde se ejecuta la función changeTime() cada segundo. Esta función cambia el valor del cronómetro dependiendo del tiempo que marcaba en el instante anterior. Para parar el cronómetro se utiliza la función stopTimer que, en función de si hay un intervalo en funcionamiento, ejecuta clearInterval(). En este caso setInterval() y clearInterval() se usan dentro de componentes de React, que se explicarán más adelante en este capítulo.

2.1.4 Promise

En JavaScript solo se puede realizar una acción a la vez, ya que hay un único hilo de ejecución disponible. Por tanto, si hay una secuencia de operaciones, estas se deben ejecutar una a una. La solución planteada para poder ejecutar varias tareas a la vez es el objeto "Promise".

"Promise" o Promesa [6] es el fundamento de la programación asíncrona en JavaScript. Es un objeto devuelto por una función asíncrona que representa el estado actual de la operación. Un objeto Promesa puede estar en cuatro estados diferentes: "resuelta", donde el resultado es un valor; "rechazada", donde el resultado es un error; "pendiente", donde el resultado es indefinido y aún no se ha determinado si la promesa ha sido resuelta o rechazada; y "establecida", si se ha determinado que la respuesta ha sido resuelta o rechazada.

Una vez que la Promesa es devuelta, la operación no ha terminado, por lo que existen métodos para manejarla. Estos métodos se muestran en la Ilustración 8, reutilizando el ejemplo de la Ilustración 6:

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => resolve('Hola Mundo'), 1000);
});

myPromise.then(value => console.log(value));

console.log(myPromise);
```

Ilustración 8 – Creación y uso del objeto Promise

En este ejemplo, se ha creado un objeto Promesa que acepta dos funciones: resolve() y reject(). La función resolve() se ejecuta cuando el proceso se ha resuelto, y reject() cuando ocurre un error en la resolución. En este caso, la Promesa tendrá el estado "resuelta" si se puede ejecutar setTimeout(), y se llamará al método then que indica que se imprimirá por pantalla el valor devuelto por la Promesa. Tras ejecutar la función myPromise, se espera 1 segundo y se resuelve la Promesa, y finalmente se muestra por consola el mensaje "Hola Mundo".

2.1.5 Async/await

Para trabajar con Promesas, existe una sintaxis llamada "async/await" [7] que permite utilizar funciones asíncronas de forma más simple. Todas las funciones que comienzan con la palabra "async" devuelven una Promesa. La palabra "await" se utiliza dentro de las funciones "async" y permite esperar hasta que la Promesa se establezca y devuelva el resultado. Un ejemplo de esto se puede observar en la Ilustración 9:

```
async function funcion() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("Hola Mundo"), 1000)
  });

  let result = await promise;

  alert(result);
}

funcion();
```

Ilustración 9 – Función utilizando async/await

Se ha creado una función que, al llamarla, crea una Promesa y se detiene en la palabra "await". Luego, espera a que la Promesa se resuelva y el resultado se guardará en la variable "result". Finalmente, tras 1 segundo se muestra una alerta con el mensaje "Hola Mundo" si la Promesa se ha resuelto.

En el caso de que no se escriba la palabra "async" antes de la función, si se intenta utilizar "await" dentro de esta, aparecerá un error de sintaxis. Por tanto, "await" solo funciona si se añade "async" al principio.

Si ocurre un error durante la ejecución de la Promesa se generará una excepción.

2.2 React Native

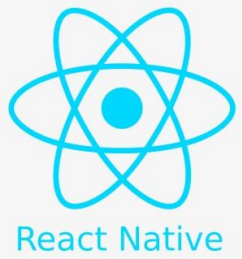

Ilustración 10 – Logo de React Native

2.2.1 Descripción y características generales

React Native [8] es un framework de JavaScript que se utiliza para programar aplicaciones móviles multiplataforma nativas para Android e iOS. Está basada en ReactJS, una librería de código abierto de JavaScript creada por Facebook para desarrollar interfaces de usuario y sus componentes. React Native fue publicado por primera vez en 2015 en una conferencia de ReactJS [9] y, desde entonces, ha sido utilizado por muchos desarrolladores y organizaciones por su capacidad para producir apps nativas. En la Ilustración 11 se puede observar como superó el desarrollo de aplicaciones en iOS con ObjectiveC/Swift y Android con Java/Kotlin [10]:

Ilustración 11 – Gráfico de la evolución de React Native con respecto a Android e iOS (2015-2018)

React Native utiliza una mezcla entre JavaScript y JSX para su programación. JSX es una extensión que permite encapsular XML dentro de JavaScript, y con este formato se pueden renderizar componentes nativos en forma de vistas o "views". Ejemplos de estos componentes son "Image", "View" y "Text". La sintaxis de estos se puede observar en la Ilustración 12, donde se ha añadido un componente "Video" y sus propiedades dentro de una vista o componente "View":

```
<View>
  <Video
    ref={videoRef}
    source={{uri: video}}
    useNativeControls
    resizeMode="contain"
    isLooping={false}
    isMuted={false}
    style={styles.videoPlayer}
    onPlaybackStatusUpdate={status => setStatusVideo(() => status)}
  />
</View>
```

Ilustración 12 – Componente Video de React Native

Una de las principales ventajas de utilizar React Native es la oportunidad de construir aplicaciones para distintos sistemas operativos ahorrando tiempo en el proceso, ya que solo hay que emplear un lenguaje de programación. Además, está la posibilidad de ejecutar la aplicación mientras se actualiza a nuevas versiones y los cambios de la interfaz de usuario se pueden observar durante la ejecución.

2.2.2 Componentes de React

El archivo inicial en el cual se basa una aplicación en React Native es *App.js*. Puede ser una clase o una función dependiendo del tipo de componentes que se utilicen para crear la aplicación. Un componente de React [11] es un elemento que tiene su propio estado o "state" y propiedades o "props", e implementa su propia lógica de renderizado. Inicialmente, React solo utilizaba componentes de clases que contenían métodos para manejar sus ciclos de vida, entre los cuales se destacan: componentDidMount(),

componentDidUpdate() y componentWillUnmount(). Estos se utilizan para mantener o actualizar un estado. El estado es un objeto que contiene un par clave-valor, determina cómo se comportan los componentes y permite que estos sean dinámicos e interactivos.

En cambio, los componentes funcionales están formados por una función que obtiene las propiedades y devuelve un JSX. No tienen métodos para manejar el ciclo de vida. Existen dos tipos de componentes funcionales: componentes "Stateless" o sin estado y "Hooks". Los Hooks son los únicos que pueden tener un estado.

Hay una clara diferencia entre ambos componentes [4] que se muestra en la Ilustración 13 a continuación:

```jsx
import React, { useState } from 'react';
function Example() {
  const [count, setCount] = useState(0);
    return (
      <div>
        <p> You clicked {count} times </p>
        <button onClick={() => setCount(count + 1)}>
          Click me
        </button>
      </div>
    );
}
```

```jsx
import React from 'react'
class Counter extends React.Component {
  constructor() {
    super()
    this.state = {
      counter: 0
    }
  this.handleIncrement = this.handleIncrement.bind(this)
  }
  componentDidMount() {
    document.title = this.state.counter;
  }
  componentDidUpdate() {
    document.title = this.state.counter;
  }
  handleIncrement() {
    this.setState({
      counter: this.state.counter += 1
    })
  }
  render() {
    return (
      <div>
        <div> {this.state.counter} </div>
        <hr />
        <button type="button" onClick={this.handleIncrement}>+</button>
      </div>
    )
  }
}
```

Ilustración 13 – Componentes de clase y de función

En este ejemplo se implementa un contador que se incrementa en una unidad cada vez que se pulsa un botón. En el componente de clase de la derecha se utilizan los métodos mencionados anteriormente para manejar su ciclo de vida y actualizar el estado del contador. Con el método handleIncrement() se incrementa el contador. Sin embargo, en el componente funcional de la izquierda se utiliza el Hook de estado useState para crear la variable count que guarda el estado del contador con setCount. Se puede observar que es una manera más sencilla, efectiva y rápida de implementar funcionalidades. Por esta razón se han utilizado componentes funcionales (Hooks) para desarrollar la app.

2.2.3 Hooks

Los Hooks [12] que se han utilizado en este proyecto son: useState, useEffect, useRef, useCallback, useNavigation y useFocusEffect.

2.2.3.1 Descripción general de los Hooks

useEffect realiza la misma función que componentDidMount(), componentDidUpdate() y componentWillUnmount(), pero unificado en una sola API. React ejecuta esta función después de cada renderizado. Si en el ejemplo de la Ilustración 14 se añade este Hook en el componente funcional [5], se le está indicando a React que debe ejecutar useEffect después de pulsar el botón. El resultado sería como

el de la Ilustración 14 a continuación:

```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Ilustración 14 – ejemplo de useEffect

useRef hace una referencia al DOM directamente en el componente funcional. Devuelve un objeto de tipo referencia que tiene una propiedad llamada "current" que contiene un valor mutable. Este valor persiste entre renderizados y, si es actualizado, no causa un nuevo renderizado. Se puede utilizar para guardar la referencia de un componente "Video".

useCallback devuelve una versión memorizada de una llamada o "callback" que cambia solo cuando una de sus dependencias ha cambiado. De esta forma solo se ejecuta cuando sea necesario, permitiendo un mayor rendimiento de la app.

useNavigation [13] y useFocusEffect [14] pertenecen a una librería llamada React Navigation que permite implementar funcionalidades para la navegación entre pantallas de la app. useNavigation concede acceso a un objeto de tipo navegación o "navigation" y devuelve la propiedad "navigation" de la pantalla. useFocusEffect permite que se realicen tareas cuando una pantalla está enfocada o "focused". Su efecto es igual al de useEffect pero solo funciona cuando la pantalla está enfocada. En la Ilustración 15 se muestra una forma de usar estos Hooks en *Lista.js*:

```
import { useNavigation, useFocusEffect } from '@react-navigation/native';
import { BackHandler } from 'react-native';
const navigation = useNavigation();
  useFocusEffect(
    React.useCallback(() => {
      isMounted = true;

      if(isMounted){
        getLista();
        BackHandler.addEventListener('hardwareBackPress', backAction);
      }
      const backHandler = BackHandler.addEventListener('hardwareBackPress',
      backAction);

      return () => {
        backHandler.remove();
        isMounted = false;
      };

    }, [flag])
  );
```

Ilustración 15 – useNavigation y useFocusEffect en Lista.js

Dentro de useCallback se ha llamado a la función getLista, que obtiene la lista de ejercicios seleccionada. También se ha añadido un "listener" que, cuando se pulsa el botón de atrás del dispositivo, realice las tareas indicadas en el método backAction. Cuando se sale de esta pantalla, se ejecutan las sentencias dentro de "return". En este caso, se elimina el "listener" y se indica que el componente está desmontado. Esto impide que haya pérdidas de memoria en la app.

2.2.4 AsyncStorage

AsyncStorage [15] es un sistema de almacenamiento de pares clave-valor asíncrono y persistente. Permite guardar datos en la aplicación de forma local y, en función del sistema operativo, almacena la información de un modo concreto. En iOS, AsyncStorage está soportado por código nativo que guarda valores ligeros en un diccionario serializado y valores más pesados en ficheros. En cambio, AsyncStorage utiliza SQLite para almacenar datos en Android.

2.2.4.1 Métodos de AsyncStorage

Esta librería de React Native proporciona métodos que devuelven un objeto "Promise" o Promesa, mencionado anteriormente en el apartado 2.1.3. Los métodos que se han utilizado en este proyecto son: getItem y setItem.

getItem busca un elemento utilizando su clave en formato "string" o cadena, y devuelve un objeto Promesa. Si se ha podido encontrar el elemento, lo devuelve como "result" o resultado. Sin embargo, si no ha encontrado el elemento devuelve un error.

setItem establece el valor junto con su clave y devuelve un objeto Promesa. Si no se ha podido realizar la operación devuelve un error.

Se puede observar un ejemplo obtenido del archivo *AllLists.js* en la Ilustración 16 a continuación:

```
<Button onPress = { async() => {
  var dia = new Date().getDate();
  var mes = new Date().getMonth() + 1; //Para obtener el mes actual

  const valorListas = await AsyncStorage.getItem("LISTAS");
  const l = valorListas ? JSON.parse(valorListas) : [];

  const listMonth = l.filter((lista) => ((JSON.parse(lista).fechacreacion.split('-')[1]) >=
    (mes-listValueNumber)));
  await AsyncStorage.setItem("LISTAS", JSON.stringify(listMonth))

  await AsyncStorage.getItem("LISTAS").then((listas) => {
    setListas(JSON.parse(listas));
    setFlagList(!flagList);
  });
  console.log('Se han eliminado los elementos creados hace más de x meses', listValueNumber)
  setModalVisible(false);
  }
}
```

Ilustración 16 – Uso de AsyncStorage en AllLists.js

Se ha añadido un botón que, una vez pulsado, ejecuta una función asíncrona (se indica con la palabra "async" al principio). Esta función obtiene "LISTAS" de AsyncStorage, que contiene todas las listas de la app, y elimina las que tengan una fecha de creación anterior a los meses indicados en la variable listValueNumber. Para obtener las listas se invoca al método getItem con la palabra "await", que devuelve un objeto Promesa (el resultado si la Promesa se ha resuelto) que se guarda en una variable llamada valorListas. Seguidamente, se filtran las listas devueltas por AsyncStorage (con el método filter) cuya fecha de creación sea mayor o igual a la fecha indicada por listValueNumber. Finalmente, se utiliza el método setItem para guardar el resultado en AsyncStorage y se actualizan las listas con la variable de estado setListas en getItem.

2.3 Expo

Ilustración 17 – Logo de Expo

2.3.1 Descripción y características generales

Expo [16] es un framework de código abierto que se usa para crear aplicaciones en React Native. Es un conjunto de herramientas y servicios que facilitan el desarrollo de React Native debido a que no se necesita conocer lenguaje nativo para ello. Además, contiene APIs nativas para iOS y para Android, entre las cuales se encuentran "FileSystem", "Camera" y "Video". Estas APIs están disponibles en el Expo SDK [17], que proporciona acceso a las funcionalidades del dispositivo, como la cámara, la localización o los contactos en forma de paquetes.

2.3.2 Instalación y ejecución del entorno

Para instalar Expo se necesita Node.js [18], que es un entorno de servidor de código abierto que utiliza JavaScript y programación asíncrona. Esto se debe a que la gestión de herramientas y dependencias de la línea de comandos de Expo depende de Node.js. La interfaz de línea de comandos de Expo se denomina Expo CLI, que se utiliza para realizar pruebas en la app mientras está siendo desarrollada.

Para probar la funcionalidad de la aplicación móvil se necesita la aplicación de cliente de Expo para iPhone y Android, que se puede descargar de App Store y de Play Store respectivamente.

Cuando se inicia la aplicación con el comando "expo start" en Expo CLI, un empaquetador de JavaScript llamado Metro [19] se inicializa a la vez. Su función es combinar el código JavaScript en un único fichero y traducir código JavaScript y JSX [20]. También convierte recursos o "assets", que son imágenes principalmente, en objetos que se pueden visualizar en un componente de tipo imagen o "Image". Además, con este comando se inicializa también el servidor de Expo llamado Expo Development Server. Su función es obtener el código JavaScript generado por Metro y ejecutarlo en la aplicación de cliente de Expo en el dispositivo o emulador. En la Ilustración 18 se muestra el Expo Development Server al abrir la página localhost:19002, donde se puede escanear el código QR desde la aplicación móvil de Expo o ejecutar la app desde un emulador.

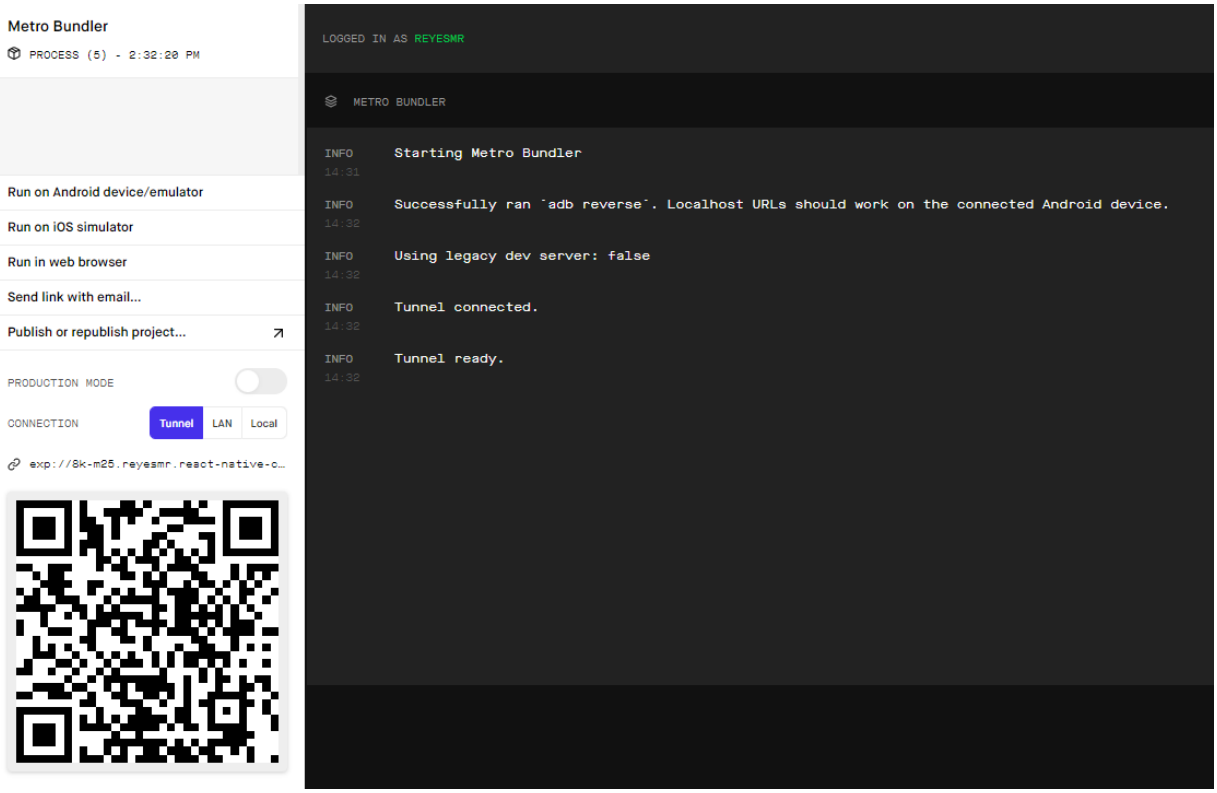

Ilustración 18 – Expo Development Server

2.3.3 Flujos de trabajo

La forma de desarrollo en Expo mencionada anteriormente se corresponde con uno de los dos flujos de trabajo [21] que existen, denominado "managed". En este flujo se escribe solo en JavaScript o TypeScript y las herramientas de Expo se encargan de realizar las tareas principales. El otro flujo se denomina "bare", donde se puede usar cualquier servicio o librería de Expo. Es el usuario quien debe desarrollar los proyectos nativos en Android, utilizando Android Studio, y en iOS, utilizando Xcode. Los desarrolladores que utilizan el flujo "managed" no tienen esa responsabilidad y la complejidad es gestionada por Expo. En el caso de que se encuentren limitaciones y se deba acceder a código nativo, siempre se puede cambiar de "managed" a "bare" y seguir desarrollando el código en ese flujo.

En este proyecto se ha utilizado el flujo de trabajo "managed", donde se configura el icono de la app y la pantalla de carga en un archivo llamado "app.json", y la configuración de Metro se indica en el archivo "metro.config.js". Los paquetes de Expo que se han utilizado se indican en el archivo "package.json", y entre ellos se encuentran: Camera, Video, Document Picker, Image Picker, FileSystem, MediaLibrary, Notifications y Sharing.

2.4 JSON

Ilustración 19 – Logo de JSON

2.4.1 Descripción y características generales

JSON [22] es la forma abreviada de JavaScript Object Notation. Es un formato ligero usado para almacenar y transmitir datos. Es autodescriptivo y fácil de entender.

Los datos están separados por comas y se guardan en pares nombre-valor. Los objetos se guardan entre llaves "{ }" y las matrices entre corchetes "[ ]". Cada objeto puede contener varios pares nombre-valor y cada matriz puede contener varios objetos.

Un programa JavaScript puede convertir datos JSON en objetos JavaScript nativos. El formato JSON es solo texto, y se debe diferenciar de la sintaxis JSON, que deriva de la sintaxis de notación de objetos de JavaScript. En la Ilustración 20 se presenta un ejemplo donde se ha creado una cadena JavaScript que contiene sintaxis de JSON, y se utiliza la función "JSON.parse()" para convertirla a un objeto JavaScript:

```
var texto = '{ "lista" : [' +
'{ "titulo":"Lista de Vídeos" },' +
'{ "fecha":"10-11-2022" },' +
'{ "primerVideo":"video1" , "segundoVideo":"video2" } ]}';

var objeto = JSON.parse(texto);
```

Ilustración 20 – Conversión JSON a objeto JavaScript

2.4.2 Uso en la aplicación

En este proyecto se han creado listas de ejercicios en formato JSON, como se puede observar en la Ilustración 21:

```
{
  "imagen": 45,
  "titulo": "Lista de ejercicios",
  "videolista": [
                "{\"series\":\"2\",\"repeticiones\":\"9\",\"tiempo\":\"7\",\"videos\":\"SAbPjOFoL4Q\",\"id\":\"169\",
                    \"realizado\":\"no\",\"imagenRealizado\":60,\"emoticono\":\"\",\"foto\":\"\"}",
                "{\"series\":\"6\",\"repeticiones\":\"5\",\"tiempo\":\"2\",\"videos\":\"Pvxj-1dgVk0\",\"id\":\"487\",
                    \"realizado\":\"no\",\"imagenRealizado\":60,\"emoticono\":\"\",\"foto\":\"\"}",
                "{\"series\":\"9\",\"repeticiones\":\"7\",\"tiempo\":\"5\",\"videos\":\"mEcTv_omtPY\",\"id\":\"572\",
                    \"realizado\":\"no\",\"imagenRealizado\":60,\"emoticono\":\"\",\"foto\":\"\"}"
              ],
  "email": "reyesmr97@gmail.com",
  "idlista": "319",
  "fechacreacion": "19-10-2022",
  "listaRealizado": "no",
  "imagenListaRealizado": 60,
  "historial": [
                "{\"idHistorial\":\"100\",
                 \"fechaRealizado\":\"28-10-2022\",
                 \"tiempoTotal\":\"03:00:00\",
                 \"tiemposEjercicios\":[
                    "{\"id\":\"169\",\"series\":\"2\",\"repeticiones\":\"9\",\"tiempoRealizacion\":\"00:30:00\",
                      \"tiempoDescanso\":\"00:03:00\",\"video\":\"SAbPjOFoL4Q\"}",
                    "{\"id\":\"487\",\"series\":\"6\",\"repeticiones\":\"5\",\"tiempoRealizacion\":\"01:00:00\",
                      \"tiempoDescanso\":\"00:04:00\",\"video\":\"Pvxj-1dgVk0\"}",
                    "{\"id\":\"572\",\"series\":\"9\",\"repeticiones\":\"7\",\"tiempoRealizacion\":\"01:30:00\",
                      \"tiempoDescanso\":\"00:05:00\",\"video\":\"mEcTv_omtPY\"}"
                   ]"}",
                "{\"idHistorial\":\"300\",
                 \"fechaRealizado\":\"31-10-2022\",
                 \"tiempoTotal\":\"02:30:00\",
                 \"tiemposEjercicios\":[
                    "{\"id\":\"169\",\"series\":\"2\",\"repeticiones\":\"9\",\"tiempoRealizacion\":\"00:20:00\",
                      \"tiempoDescanso\":\"00:02:00\",\"video\":\"SAbPjOFoL4Q\"}",
                    "{\"id\":\"487\",\"series\":\"6\",\"repeticiones\":\"5\",\"tiempoRealizacion\":\"00:50:00\",
                      \"tiempoDescanso\":\"00:03:00\",\"video\":\"Pvxj-1dgVk0\"}",
                    "{\"id\":\"572\",\"series\":\"9\",\"repeticiones\":\"7\",\"tiempoRealizacion\":\"01:20:00\",
                    \"tiempoDescanso\":\"00:04:00\",\"video\":\"mEcTv_omtPY\"}"
                   ]"}",
              ],
  "fondo": ""
}
```


Ilustración 21 – Lista de ejercicios en formato JSON

En la lista se pueden identificar los siguientes campos:

- imagen: contiene un número que corresponde a un icono que se encuentra en la carpeta “assets” del proyecto, o la uri de una imagen seleccionada del almacenamiento interno del dispositivo.
- titulo: contiene el título de la lista “Lista de ejercicios”.
- videolista: en este campo se almacenan los enlaces de vídeos de los ejercicios que conforman la lista y todas sus características en una matriz. Algunas de las características o campos de los vídeos son: series, repeticiones, tiempo, videos e id. El campo “videos” puede ser parte de un enlace de un vídeo de Youtube o una uri de un vídeo local.
- email: guarda la dirección de correo electrónico del usuario. Esta dirección se almacena de forma local la primera vez que el usuario ingresa en la app.
- idlista: es el identificador de la lista.
- fechacreacion: indica la fecha en la que se ha creado la lista.
- listaRealizado: en este campo se indica si la lista se ha realizado.
- imagenListaRealizado: contiene un icono que aparece para indicar si la lista se ha realizado o si está pendiente por realizarse.
- historial: en este campo se almacena un historial en una matriz, y cada elemento guarda un intento de entrenamiento de esa lista de ejercicios. Es decir, cada vez que se realiza la lista se guarda una entrada en esa matriz. Los campos de cada entrenamiento son: idHistorial,

fechaRealizado, tiempoTotal y tiemposEjercicios. En este último se almacena una matriz donde cada entrada corresponde a uno de los vídeos de la lista. Para cada vídeo hay un campo llamado tiempoRealizacion, donde se guarda el tiempo que ha tardado en realizarse el ejercicio, y uno llamado tiempoDescanso, donde se guarda el tiempo de descanso del ejercicio.

- fondo: por último, en este campo se almacena la uri de la imagen seleccionada para el fondo de la lista.

Todos los campos tienen el formato de una cadena o "string" excepto "imagen", "imagenRealizado" e "imagenListaRealizado", ya que pueden ser números que correspondan a una imagen de la carpeta "assets" del proyecto.

3 HERRAMIENTAS UTILIZADAS

3.1 Herramientas de desarrollo

En este capítulo se especificarán las herramientas empleadas durante el desarrollo del proyecto.

3.1.1 Visual Studio Code

Ilustración 22 – Logo de Visual Studio Code

Visual Studio Code o VS Code [23] de forma abreviada es un editor de código fuente que es gratuito y de código abierto. Contiene una herramienta llamada IntelliSense que completa el código de forma automática para variables, métodos y módulos importados. Además, incluye un depurador interactivo que permite procesar el código en búsqueda de errores, inspeccionar variables y ejecutar comandos en la consola. Ofrece unas características que facilitan el desarrollo del código, como el subrayado de la sintaxis, correspondencia instantánea de llaves y corchetes, e indentación automática. Integra control de código fuente con Git. Está creado por Microsoft y está disponible para varios sistemas operativos. Contiene soporte para JavaScript, TypeScript y Node.js, y tiene un conjunto de extensiones para otros lenguajes de programación. Está desarrollado con Electron, un marco de trabajo que permite escribir aplicaciones multiplataforma utilizando JavaScript, HTML y CSS.

La interfaz de usuario de VS Code [24] se muestra en la Ilustración 23:

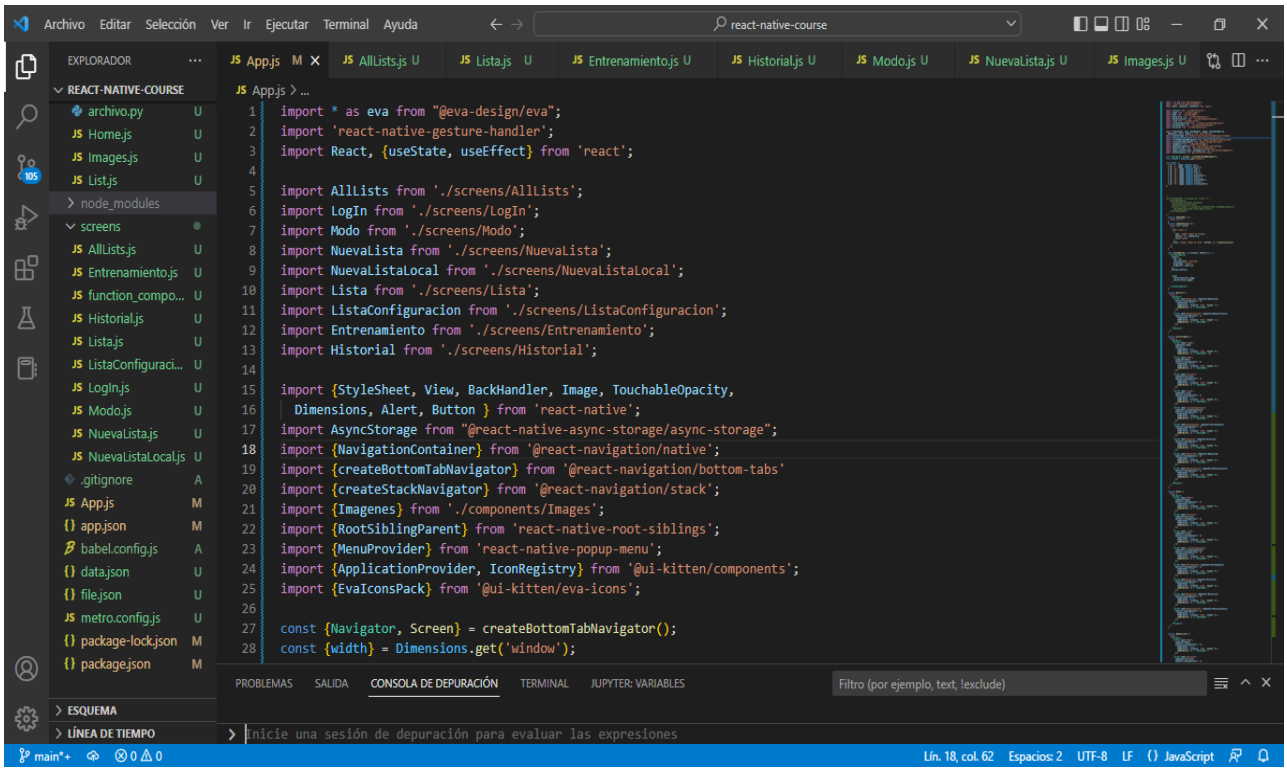

Ilustración 23 – Interfaz de usuario de Visual Studio Code

Se puede dividir en 5 áreas: el editor, la barra que se sitúa abajo a la izquierda, la barra de estado, la barra de actividad y los paneles. El editor es el área principal donde se muestra el contenido de los ficheros y donde se pueden editar. Se pueden abrir varios editores a la vez de forma vertical u horizontal. La barra situada abajo a la izquierda contiene un botón que indica el número de fallos que tiene el proyecto y un botón para subir los cambios a GitHub. La barra de estado es la barra inferior de color azul, que contiene información sobre el proyecto y los ficheros. La barra de actividad está localizada a la izquierda del explorador de archivos y ficheros del proyecto. Entre estos archivos se pueden destacar *App.js*, *app.json* y *metro.config.js* mencionados en el capítulo 2. Permite cambiar entre diferentes vistas y proporciona indicadores como el número de cambios pendientes por subir a GitHub. Finalmente, los paneles se encuentran en la región situada debajo del editor y se usan para la depuración, informar sobre errores y escribir en el terminal.

Cada vez que se inicia VS Code, se abre en el mismo proyecto y estado en el que se encontraba antes de cerrarlo por última vez. Se conservan los ficheros y las carpetas.

3.1.2 Expo Go

Ilustración 24 – Logo de Expo Go

Expo Go [25] es una aplicación de código abierto que se utiliza para ejecutar aplicaciones de React

Native en Android e iOS. Gracias a Expo Go no es necesario instalar la aplicación del proyecto de forma local en el dispositivo para comprobar su funcionamiento mientras se está desarrollando. Es una manera rápida y efectiva de probar los cambios realizados en el código tanto en dispositivos físicos como en emuladores.

Se puede observar el funcionamiento de Expo Go en la Ilustración 25 a continuación:

Ilustración 25 – Funcionamiento de Expo Go

Primero, se debe instalar la app en el dispositivo. Después, en el terminal se debe ejecutar el comando "expo start" en la carpeta del proyecto y, de esta forma, Expo CLI iniciará el Expo Development Server y se generará un código QR. Si el dispositivo es un Android, se escaneará el QR desde la app Expo Go. Si es un iOS, se escaneará desde la aplicación de la cámara. A través de este código, el usuario podrá conectarse al servidor de Expo y devolverá un fichero de JavaScript que contendrá las características más importantes del proyecto. Con este fichero, Expo Go puede descargar un empaquetador de JavaScript y los recursos necesarios para ejecutar el proyecto. Finalmente, el motor JavaScript de Expo Go ejecutará este empaquetador para renderizar la aplicación.

En la Ilustración 26 se muestra la interfaz de usuario de Expo Go en un dispositivo Android:

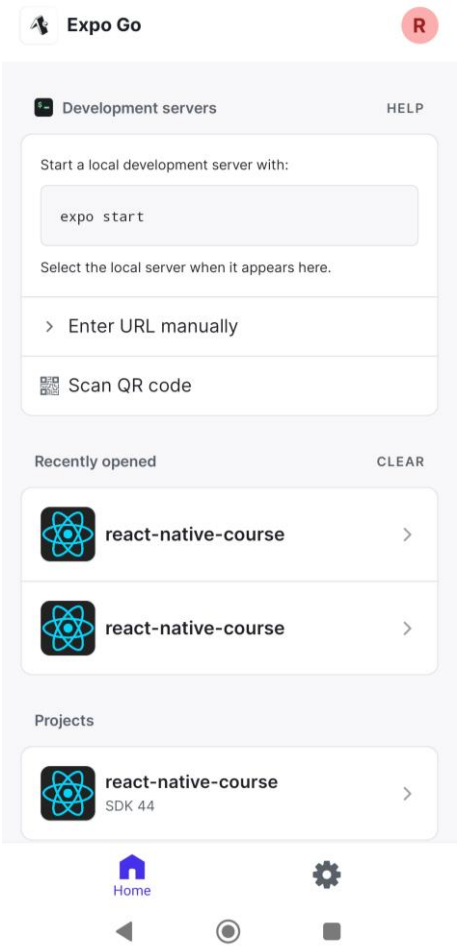

Ilustración 26 – Interfaz de usuario de Expo Go

3.1.3 Git

Ilustración 27 – Logo de Git

Git [26] es un sistema de control de versiones distribuido utilizado en muchos proyectos de software. Su uso se extiende en sistemas operativos y entornos de desarrollo diferentes. Permite gestionar y almacenar proyectos de forma local. Los archivos de un proyecto se almacenan en un repositorio de Git, que contiene todo el historial de revisiones. Se puede acceder a Git mediante la línea de comandos en un terminal o en una aplicación de escritorio que tiene una interfaz de usuario gráfica o GUI llamada GitHub Desktop.

3.1.3.1 GitHub

Ilustración 28 - Logo de GitHub Desktop

GitHub [27] es un servicio en la nube de código abierto que permite guardar y gestionar el código fuente. Además, se pueden rastrear y controlar los cambios en el código. GitHub Desktop es una aplicación que permite a un usuario interactuar con GitHub utilizando una GUI, que resulta más sencillo de utilizar que la línea de comandos o el navegador web. Se pueden subir cambios, usar herramientas colaborativas y clonar repositorios remotos con esta aplicación.

En la Ilustración 29 se muestra la interfaz de usuario de GitHub Desktop y la subida del proyecto al repositorio:

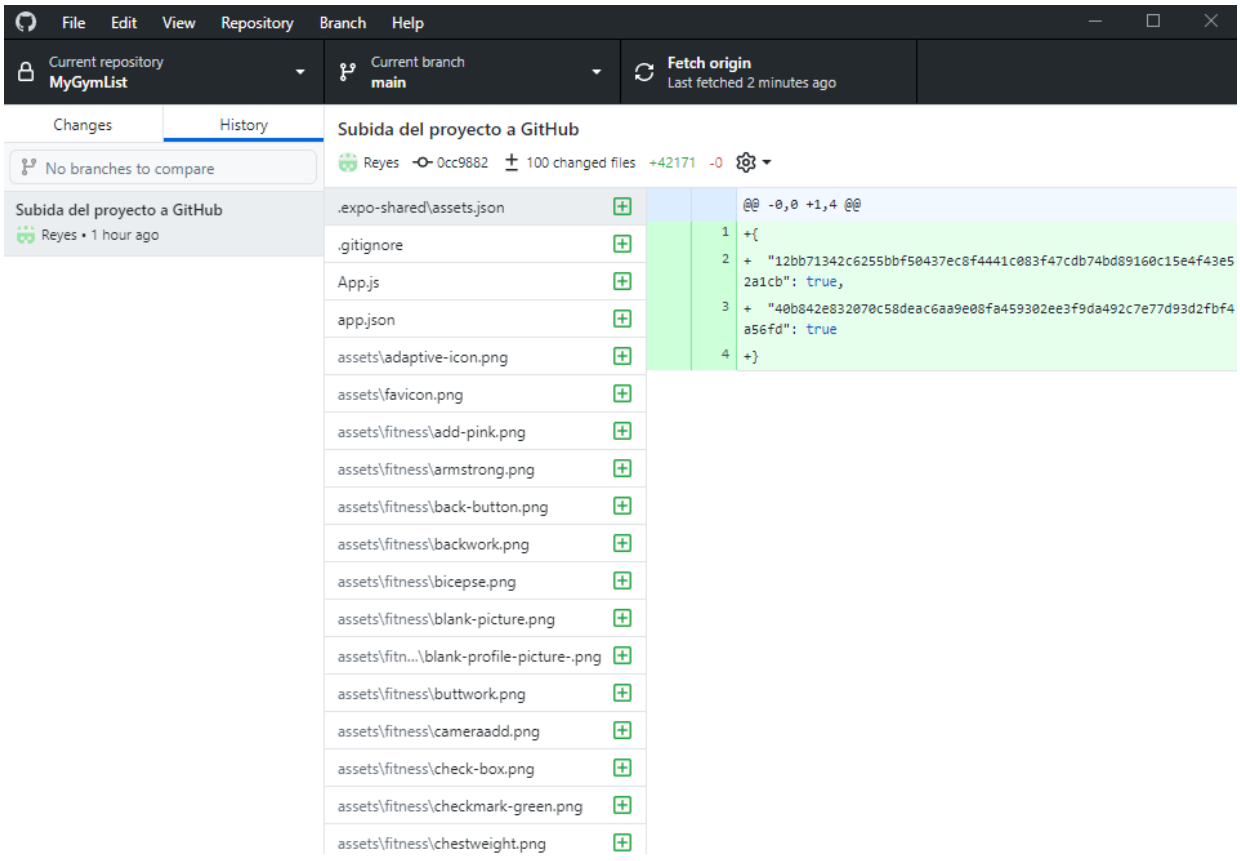

Ilustración 29 – Interfaz gráfica de GitHub Desktop

3.2 Otras herramientas utilizadas

3.2.1 Vysor

Ilustración 30 – Logo de Vysor

Vysor [28] es una aplicación mediante la cual los usuarios pueden visualizar y controlar su dispositivo Android o iOS en un PC. Para ello, se permite utilizar el teclado y el ratón del ordenador. Se puede utilizar para usar aplicaciones de forma remota y realizar presentaciones de proyectos que hagan uso del dispositivo. Tiene la funcionalidad de un emulador, pero es mucho más rápido y eficiente.

En este proyecto se ha utilizado Vysor en las tutorías para visualizar la interfaz de usuario de My GymList, y así poder hacer un seguimiento de los avances en la aplicación.

Para utilizar Vysor es necesario instalar la aplicación tanto en el dispositivo móvil como en el PC. Se debe conectar el dispositivo con el ordenador y abrir la app en el PC. Una vez haya detectado el dispositivo, se conectará con él y se mostrará la pantalla del teléfono móvil en la app de Vysor del PC.

En la Ilustración 31 se puede observar cómo es la interfaz de usuario de Vysor en el ordenador:

Ilustración 31 – Interfaz de usuario de Vysor

3.3 Herramientas para el modelado y diseño

3.3.1 Diagrams.net

Ilustración 32 - Logo de Diagrams.net

Diagrams.net [29], también conocido como draw.io, es una tecnología de código abierto que permite crear diagramas. Es una herramienta que se puede utilizar online en un navegador web, y también tiene una aplicación de PC. Se puede sincronizar con una cuenta de Google Drive para almacenar los cambios de forma automática y se puede modificar un diagrama en grupo. Las modificaciones tienen lugar en tiempo real, lo cual facilita colaborar en un mismo proyecto a la vez. Algunos de los diagramas que se pueden diseñar son: diagramas de flujo, de entidad-relación, de red, UML y la arquitectura de una app. Contiene una gran variedad de elementos gráficos y de plantillas.

Un ejemplo de los recursos que se pueden encontrar en esta herramienta se puede observar en la Ilustración 33:

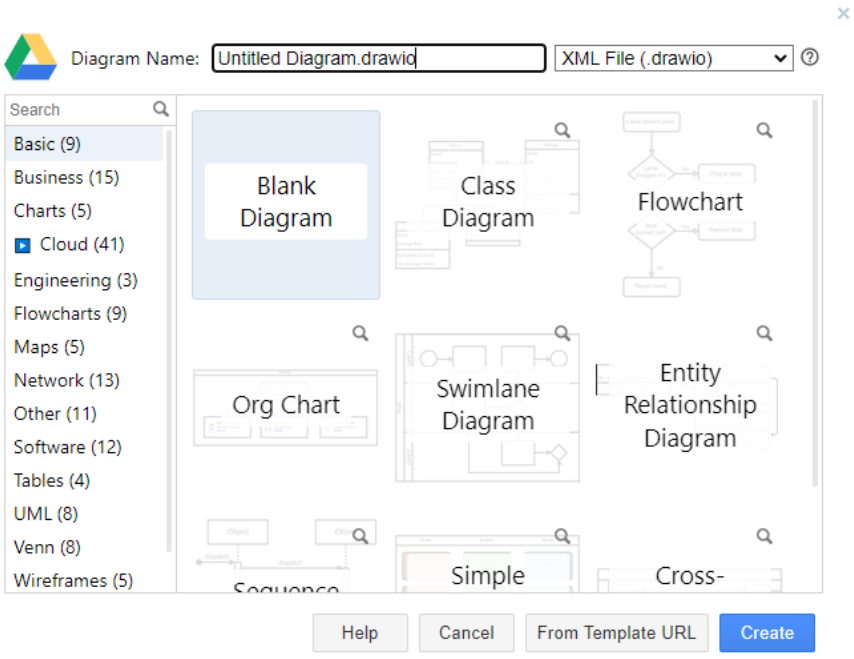

Ilustración 33 – Recursos y plantillas de Diagrams.net

Al inicio se muestran las opciones de plantillas y de elementos que se pueden elegir para crear el diagrama, y se puede modificar el nombre. Una vez se ha pulsado el botón “Create” o crear, aparece la pantalla que se observa en la Ilustración 34:

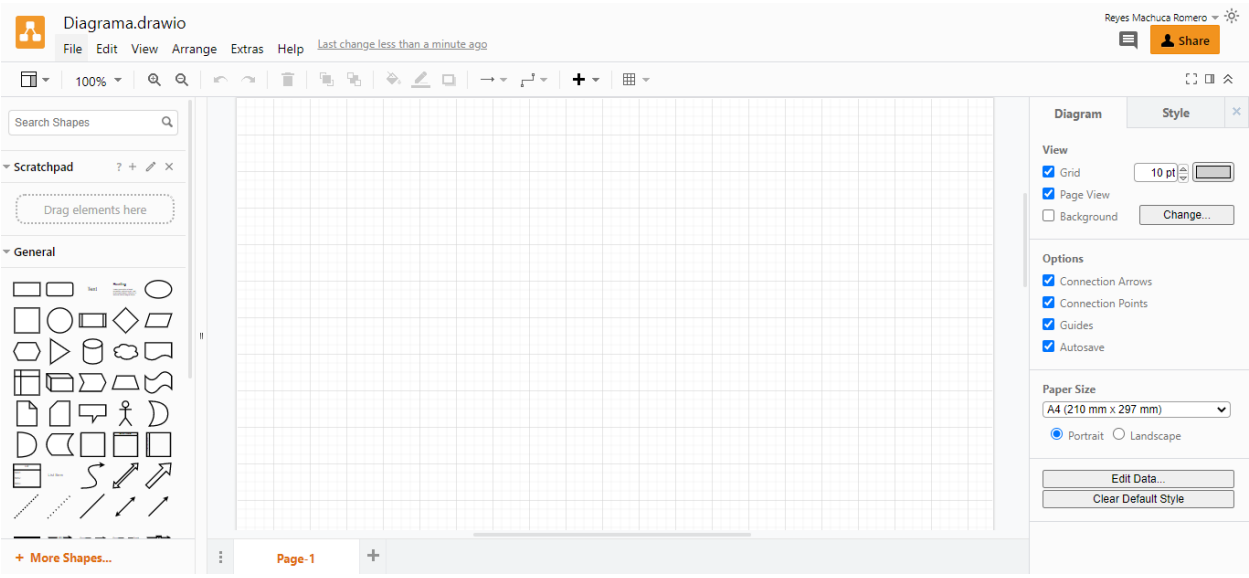

Ilustración 34 – Página de Diagrams.net

A la izquierda de la imagen se muestran todas las figuras disponibles y, si el usuario quiere añadir más, puede pulsar el botón "More Shapes". Se modifica la visualización y los estilos del diagrama con las herramientas ubicadas a la derecha. Se pueden añadir varias páginas a un mismo proyecto. La barra de herramientas en la parte superior contiene varios elementos muy útiles como, por ejemplo, abrir un diagrama existente o guardarlo en formato imagen o draw.io para poder editarlo en otro momento. Por último, se puede compartir con otras cuentas de Google Drive pulsando en el botón "Share" o añadir comentarios.

Los diagramas de casos de uso de este proyecto se han diseñado con esta tecnología.

3.3.2 Visual Paradigm

Ilustración 35 – Logo de Visual Paradigm

Visual Paradigm [30] es una herramienta creada para diseñar y modelar sistemas. Proporciona diferentes tipos de diagramas UML y plantillas que facilitan el modelado. Posee un repositorio en la nube al que pueden acceder varios miembros para desarrollar un proyecto en grupo. La interfaz de usuario de este programa se muestra en la Ilustración 36:

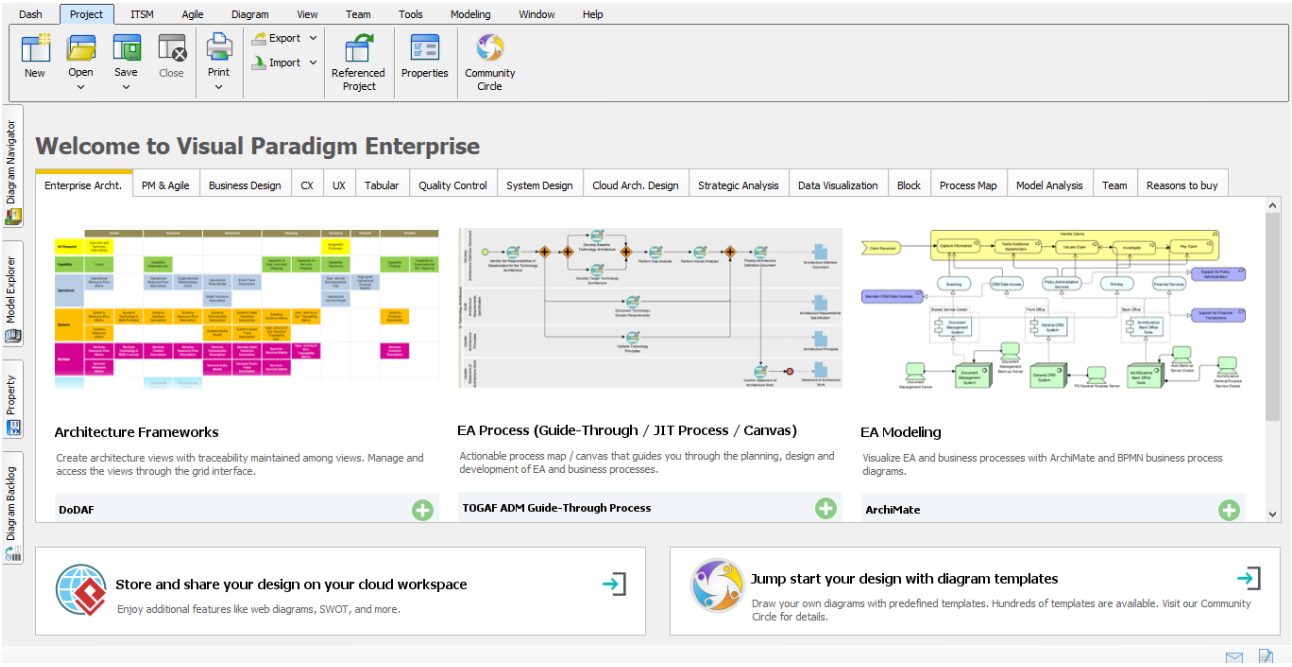

Ilustración 36 – Interfaz de usuario de Visual Paradigm

El modelado de datos y los diagramas de secuencia han sido creados con esta tecnología.

3.4 Recursos adicionales

3.4.1 Pixabay

Ilustración 37 – Logo de Pixabay

Pixabay [31] es una comunidad donde se comparten imágenes, vídeos y música que están bajo la licencia de Pixabay, por lo que se pueden utilizar sin tener que referenciarlas, incluso para fines comerciales. Cualquier usuario registrado en la página puede subir imágenes a Pixabay (bajo unas normas de calidad), y no es necesario registrarse para poder copiar, modificar, usar y distribuir el contenido. Esto se puede hacer mientras que no se vulneren los derechos de privacidad.

En este proyecto se han empleado imágenes de esta página para los iconos de la aplicación, entre los cuales se encuentran: eliminar un vídeo, importar una lista, crear una nueva lista y añadir emoticonos en los vídeos. Las imágenes se guardan en la carpeta “assets” del proyecto, y se obtiene la URL de cada una de ellas en un fichero llamado Images.js con la sentencia “require( )”.

3.4.2 Pexels

Ilustración 38 – Logo de Pexels

Pexels [32] es una página web y app donde se alojan fotos y vídeos sin derechos de autor. Proporciona recursos a los usuarios y diseñadores que se pueden utilizar de forma gratuita. Las condiciones de uso son muy similares a las de Pixabay, y se pueden encontrar una gran variedad de imágenes en ambas páginas.

Se han utilizado imágenes de Pexels para el fondo de pantalla por defecto de la aplicación.

4 ARQUITECTURA Y ANÁLISIS

La función de un buen software es hacer que
lo complejo aparente ser simple.
- Grady Booch -

En este capítulo se analizará la arquitectura de la aplicación y se crearán diagramas UML que especifiquen su funcionamiento.

4.1 Casos de uso generales

A continuación, se van a describir los casos de uso y se mostrarán los diagramas correspondientes junto con los actores implicados. Esto servirá para explicar las relaciones entre el sistema y los actores y, además, para mostrar el funcionamiento de la aplicación desde la perspectiva del usuario final.

4.1.1 Identidad de los actores

El actor que va a interactuar con el sistema y acceder a sus funciones es el usuario. Si no se ha autenticado se denominará Usuario no autenticado. En cambio, si se ha autenticado se denominará Usuario.

El usuario puede tener el rol de Usuario o Especialista indistintamente.

4.1.2 Autenticación de un usuario

Para la autenticación de un usuario se determinarán los procesos para su registro en la aplicación. Se detallarán los procesos tanto en la Ilustración 39 como en las tablas a continuación:

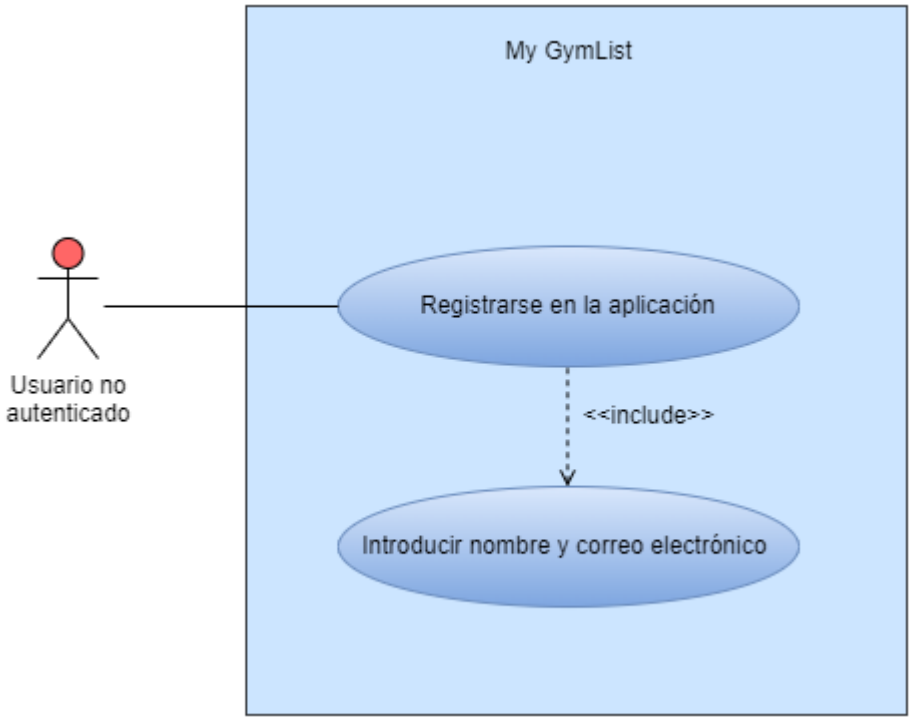

Ilustración 39 – Casos de uso para la autenticación de un usuario

CU-01	Registrarse con nombre y correo electrónico	
Precondición	El usuario no debe haberse registrado previamente en la aplicación	
Descripción	En este caso de uso se describe el comportamiento de la aplicación para registrar un usuario por primera vez.	
Secuencia normal	Paso	Acción
	1	El usuario introduce su nombre y correo electrónico
	2	El usuario pulsa el botón de iniciar sesión
Postcondición	El usuario inicia sesión y se encuentra autenticado en la aplicación	
Excepciones	Paso	Acción
	1	El usuario introduce un correo electrónico inválido
	E.1	La aplicación advierte al usuario del error y le solicita modificar e introducir los datos de nuevo
Comentarios	Ninguno	

Tabla 1. CU-01: Registrarse con nombre y correo electrónico

4.1.3 Operaciones para crear un elemento

El usuario podrá realizar operaciones para crear un elemento en la aplicación, que se detallarán en las tablas correspondientes y en el diagrama de casos de uso de la Ilustración 40:

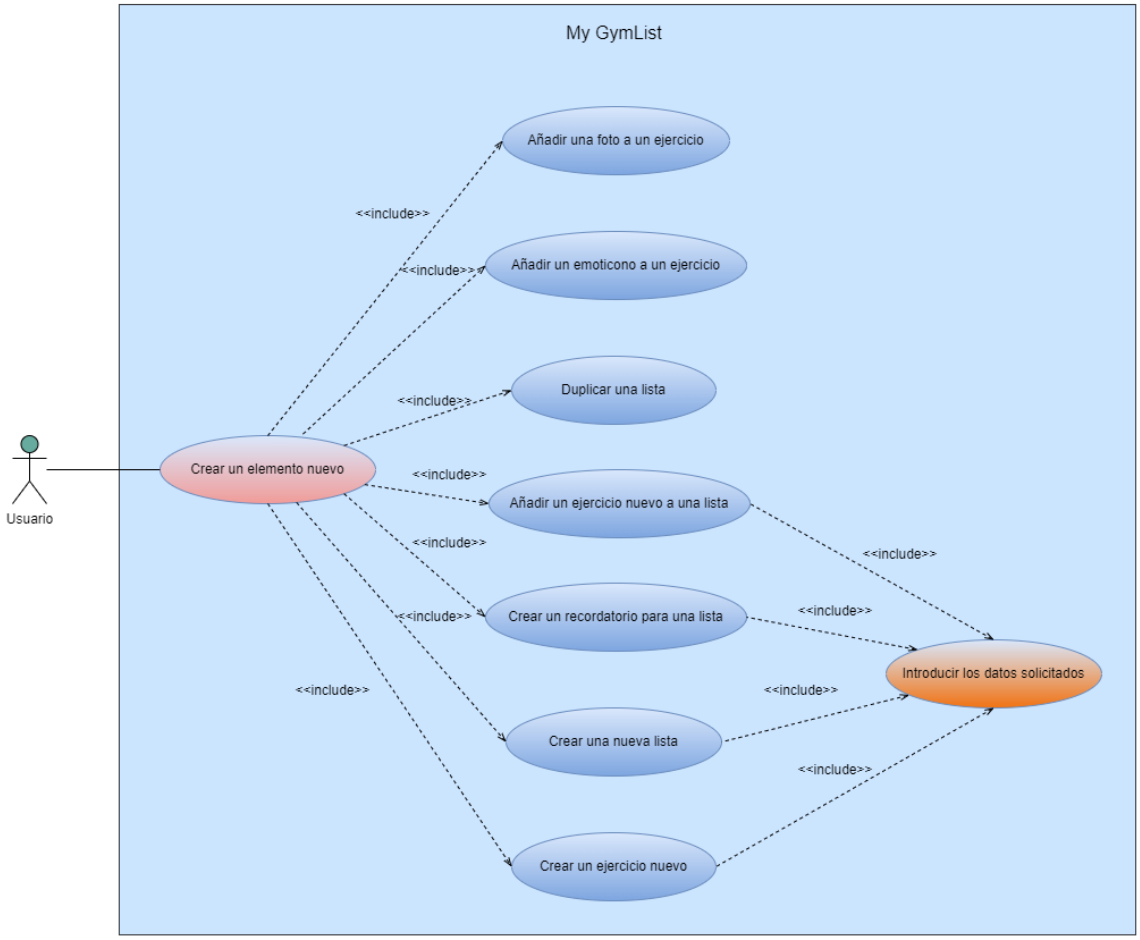

Ilustración 40 – Diagrama de casos de uso para la creación de un elemento

CU-02	Crear un elemento nuevo (un ejercicio, una lista, un recordatorio para una lista, duplicar una lista, o añadir un ejercicio a una lista, un emoticono o una foto a un ejercicio)	
Precondición	El usuario debe estar autenticado	
Descripción	En este caso de uso se describe el comportamiento de la aplicación para crear un nuevo elemento.	
Secuencia normal	Paso	Acción
	1	El usuario pulsa el botón de crear una lista
	2	El usuario pulsa el botón de añadir un ejercicio e introduce los datos solicitados

	3	El usuario pulsa el botón de guardar lista	
	4	El usuario selecciona una lista	
	5	El usuario pulsa el botón de duplicar la lista	
	6	El usuario pulsa el botón de añadir un recordatorio para la lista	
	7	El usuario pulsa el botón de añadir un ejercicio a la lista y rellena los datos solicitados	
	8	El usuario selecciona la opción de añadir un emoticono a un ejercicio de la lista	
	9	El usuario selecciona la opción de añadir una foto a un ejercicio	
	10	El usuario pulsa el botón de hacer una foto	
	11	El usuario pulsa el botón de guardar foto	
Postcondición	El usuario ha creado un nuevo elemento correctamente.		
Excepciones	Paso	Acción	
	1	El usuario introduce un enlace de un vídeo inválido (si se ha seleccionado previamente la opción de crear lista de ejercicios de YouTube)	
		E.1	La aplicación advierte al usuario del error y le solicita modificar e introducir los datos nuevamente
Comentarios	Si el usuario pulsa el botón de cancelar, no se crea el elemento y no se produce error. Todos los casos de uso en los que se crean nuevos elementos son muy similares, por lo que se han agrupado en esta tabla.		

Tabla 2. CU-02: Crear un elemento nuevo

4.1.4 Operaciones para modificar un elemento

En la Ilustración 41 se mostrarán las operaciones que podrá llevar a cabo un usuario para modificar un elemento existente en la aplicación. Asimismo, en las tablas de este apartado se explicarán los casos de uso del diagrama con mayor detalle.

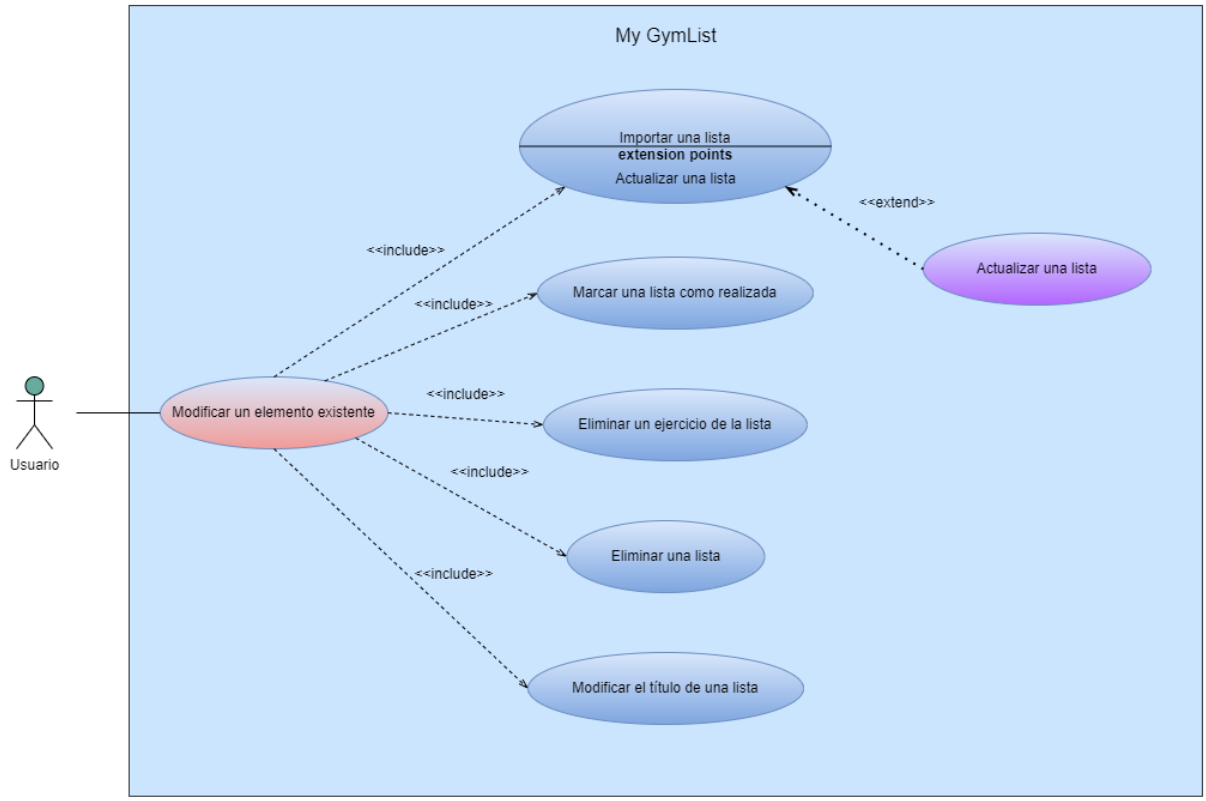

Ilustración 41 – Diagrama de casos de uso para modificar un elemento

CU-03	Modificar un elemento existente (título de lista, eliminar un ejercicio, eliminar una lista, marcar una lista como realizada, actualizar una lista)	
Precondición	El usuario debe estar autenticado	
Descripción	En este caso de uso se describe el comportamiento de la aplicación para modificar un elemento que existe.	
Secuencia normal	Paso	Acción
	1	El usuario pulsa el botón de editar título de una lista y pulsa el botón de aceptar
	2	El usuario pulsa el botón de eliminar ejercicio
	3	El usuario pulsa el botón de eliminar una lista
	4	El usuario selecciona la opción de marcar una lista como realizada
	5	El usuario pulsa el botón de importar una lista y, si la lista ya existe, puede actualizarla (se realiza el caso de uso “Actualizar una lista”)
Postcondición	El usuario ha modificado un elemento correctamente.	
Excepciones	Ninguna	

Comentarios	Si el usuario pulsa el botón de cancelar, no se modifica el elemento y no se produce error. Debido a que todos los casos de uso en los que se modifican elementos son muy similares, se han agrupado en esta tabla.

Tabla 3. CU-03: Modificar un elemento existente

4.1.5 Operaciones con una lista

Se detallarán las operaciones que se pueden efectuar con una lista de la aplicación en la Ilustración 42. En las tablas que aparecerán en este apartado se definirán los casos de uso del diagrama.

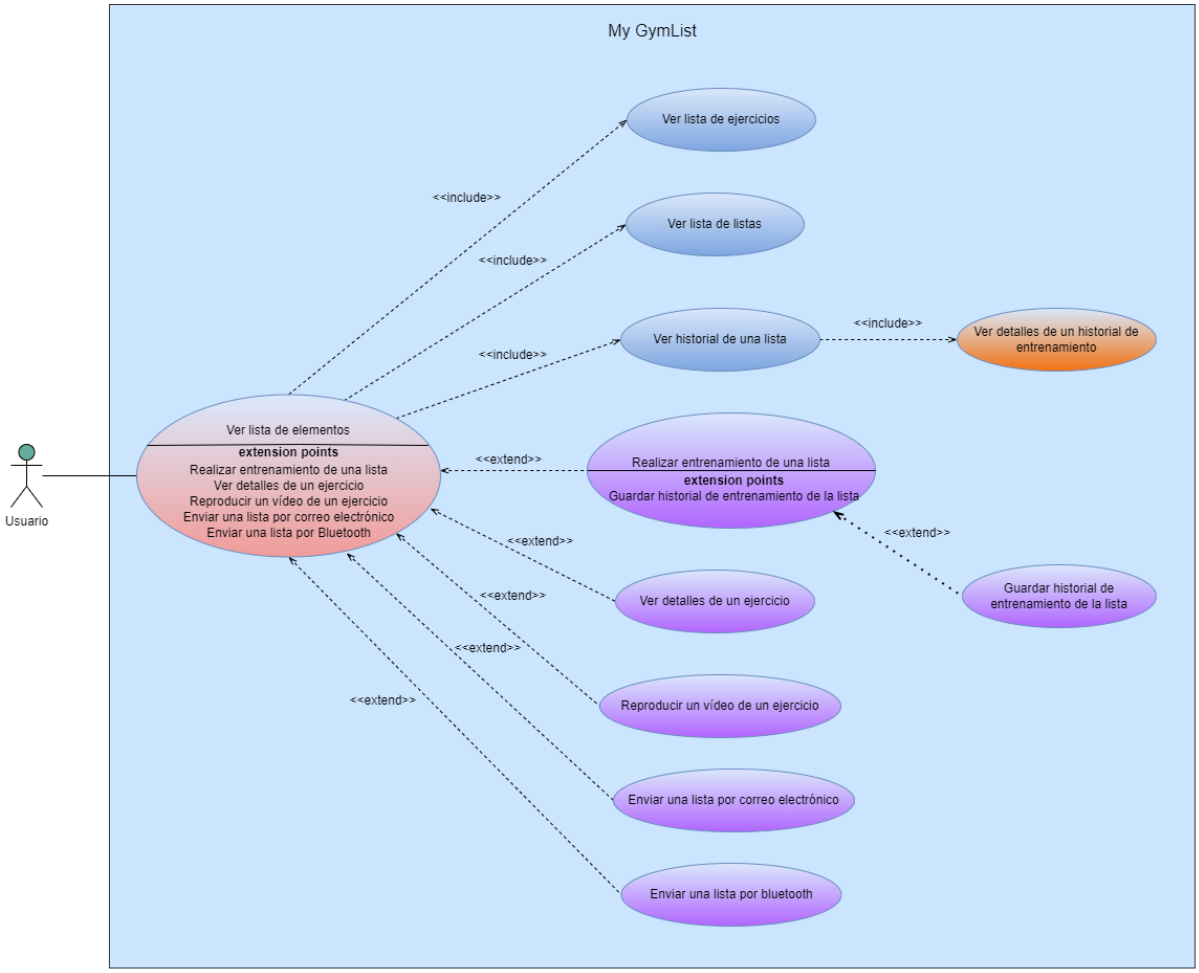

Ilustración 42 – Diagrama de casos de uso para realizar operaciones con una lista

CU-04	Ver lista de elementos (de ejercicios, de listas, historial de una lista, detalles de historial de entrenamiento)	
Precondición	El usuario debe estar autenticado	
Descripción	En este caso de uso se describe el comportamiento de la aplicación para mostrar una lista de elementos (ejercicios, listas, historial de una lista o detalles de un historial de entrenamiento).	
Secuencia	Paso	Acción

normal		
	1	El usuario selecciona una lista que existe en la aplicación
Postcondición	El usuario visualiza la lista de elementos e interactúa con ella	
Excepciones	Ninguna	
Comentarios	Si se selecciona el historial de una lista y este está vacío no se devuelve una lista, ni aparece ningún error, sino que se muestra el mensaje: “El historial está vacío”. Se han agrupado en esta tabla todos los casos de uso en los que se visualizan listas porque son muy similares entre sí.	

Tabla 4. CU-04: Ver lista de elementos

CU-05	Realizar entrenamiento de una lista	
Precondición	El usuario debe estar autenticado	
Descripción	En este caso de uso se describe el comportamiento de la aplicación para reproducir una lista de ejercicios.	
Secuencia normal	Paso	Acción
	1	El usuario pulsa el botón de iniciar entrenamiento
	2	El usuario pulsa el botón de comenzar ejercicio
	3	El usuario reproduce el vídeo del ejercicio
	4	El usuario pulsa el botón de siguiente
	5	El usuario pulsa el botón de guardar (se realiza el caso de uso “Guardar historial de entrenamiento de la lista”)
Postcondición	El usuario visualiza los vídeos de los ejercicios y guarda el historial de entrenamiento.	
Excepciones	Ninguna	
Comentarios	Si el usuario pulsa el botón de cancelar, no se guarda el historial de entrenamiento de la lista y no se produce error.	

Tabla 5. CU-05: Realizar entrenamiento de una lista

CU-06	Enviar una lista por correo electrónico o Bluetooth	
Precondición	El usuario debe estar autenticado	
Descripción	En este caso de uso se describe el comportamiento de la aplicación para enviar una lista	
Secuencia normal	Paso	Acción
	1	El usuario selecciona una lista
	2	El usuario pulsa el botón de enviar lista
	3	El usuario selecciona la opción de enviar por correo electrónico o Bluetooth
Postcondición	El usuario ha enviado una lista correctamente.	
Excepciones	Ninguna	
Comentarios	Si el usuario pulsa el botón de atrás del dispositivo, no se envía la lista y no se produce error.	

Tabla 6. CU-06: Enviar una lista por correo electrónico o Bluetooth

4.2 Diagramas de secuencia

En este apartado se representarán las interacciones más relevantes que se llevan a cabo entre los elementos de la aplicación mediante diagramas de secuencia. En ellos se detallarán los componentes que intervienen en las acciones realizadas.

4.2.1 Registro e inicio de sesión

Primero, se analizará el proceso de registro e inicio de sesión de un usuario en la aplicación. Cuando el usuario accede a la app y se inicia, este debe introducir su nombre y correo electrónico y presionar el botón de “Login”. Una vez pulsado, la base de datos local guarda los datos del usuario. A partir de ahora, cada vez que el usuario abra la aplicación tendrá la sesión iniciada, es decir, estará autenticado, y no tendrá que volver a rellenar sus datos. Los elementos que aparecen en la Ilustración 43 son los siguientes:

- Usuario: es el usuario de la aplicación, que se va a registrar e iniciar sesión
- App: es el componente principal en el que se basa la aplicación, desde el cual se va a llamar al resto de componentes que intervienen.
- LogIn: es el componente que muestra la pantalla en la que el usuario va a registrarse por primera vez.
- Base Datos Local: es donde se almacenan los datos de la aplicación, y es SQLite en Android y un fichero local en iOS.

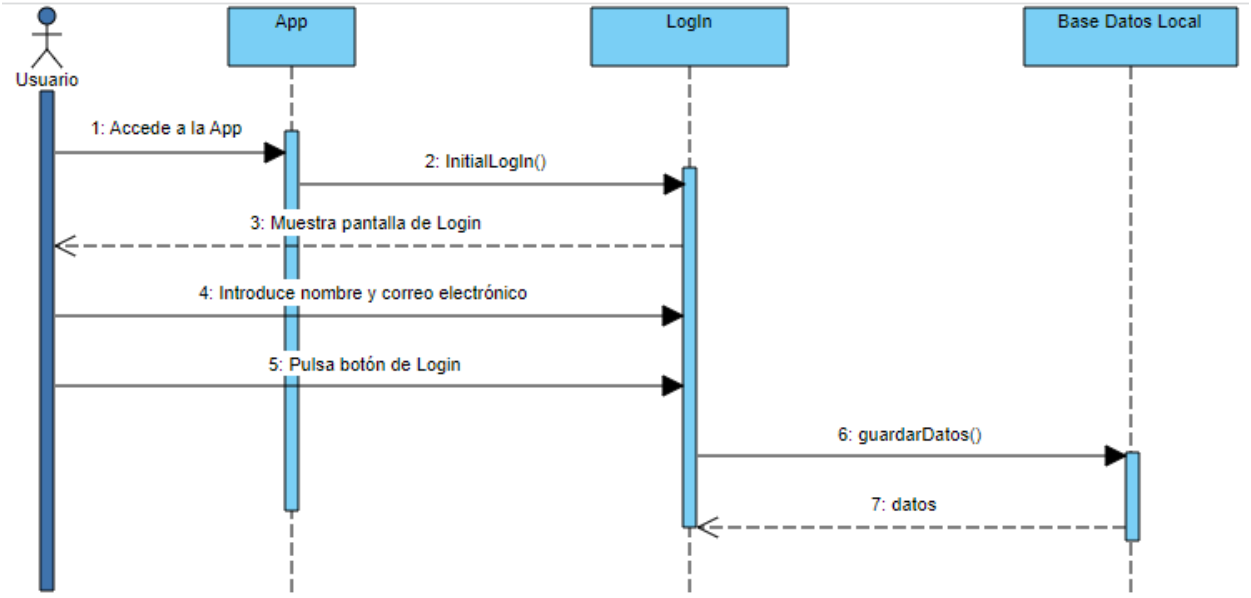

Ilustración 43 – Diagrama de secuencia para el registro e inicio de sesión

4.2.2 Crear una lista

Un usuario registrado en la aplicación puede crear una lista de ejercicios. En la Ilustración 44 se representan los procedimientos que se llevan a cabo para crear una lista y los elementos que participan, que son los siguientes:

- Usuario, App y Base Datos Local, que se han descrito en el punto anterior.
- Modo: es un componente que muestra dos botones que corresponden a dos modos de funcionamiento diferentes de la aplicación: Entrenamiento y Configuración.
- AllLists: es el componente que muestra la lista que recoge todas las listas de ejercicios de la aplicación.
- Lista Nueva: es el componente NuevaLista si el usuario quiere crear una lista que contiene ejercicios con vídeos de YouTube, y NuevaListaLocal si quiere que contenga ejercicios con vídeos almacenados en local.
- Almacenamiento interno: hace referencia al directorio interno del dispositivo.

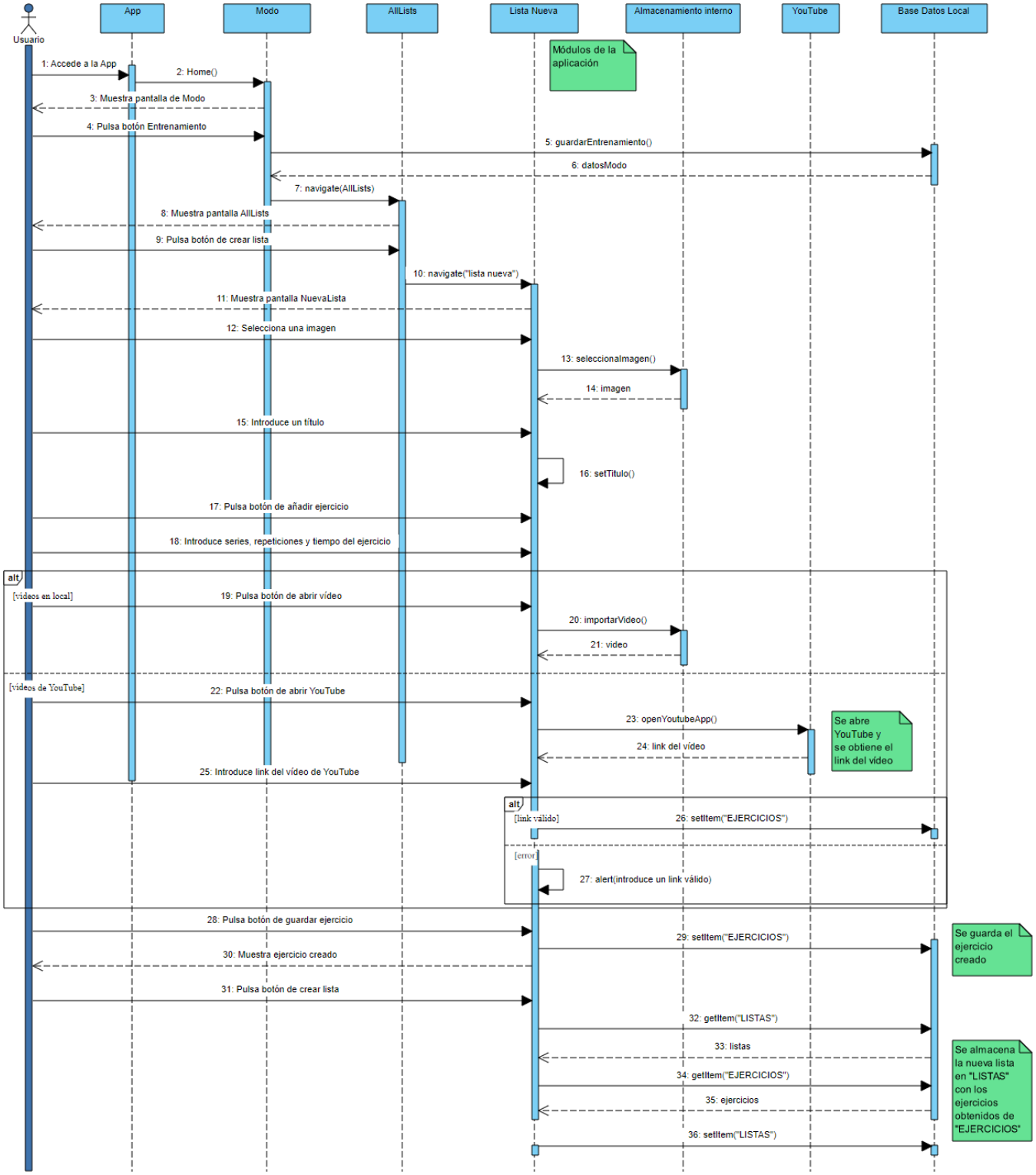

Ilustración 44 – Diagrama de secuencia para crear una lista

En el mensaje 13, si el usuario elige la opción de añadir una imagen de la galería, se utiliza el módulo ImagePicker [33] de Expo, que permite acceder al almacenamiento interno del dispositivo y seleccionar imágenes. Para ello, primero debe pedir permisos al usuario para acceder a su galería y, una vez aceptado los permisos, puede acceder al almacenamiento llamando al método ImagePicker.launchImageLibraryAsync(). En el caso de que elija la opción de añadir un icono, no será necesario acceder al almacenamiento, ya que los iconos se guardan en la carpeta "assets" de la aplicación.

En el mensaje 20, se utilizan los mismos métodos que en el mensaje 13 para elegir un vídeo del

almacenamiento interno, ya que ImagePicker permite seleccionar tanto imágenes como vídeos.

4.2.3 Realizar entrenamiento de una lista

Una vez que el usuario está registrado puede acceder a la pantalla de Inicio, donde se indican dos modos: Entrenamiento y Configuración. Si escoge el modo Entrenamiento, aparece una lista donde se muestran todas las listas de ejercicios de la app. Cuando selecciona una lista concreta, puede realizar el entrenamiento de esta pulsando el botón de "Iniciar entrenamiento". Una vez iniciado, puede realizar cada ejercicio y se va guardando el tiempo de realización y de descanso y, una vez terminados, al pulsar el botón de "Guardar" se almacenan los tiempos en el historial de entrenamiento de la lista. En la Ilustración 45 se representan estas acciones en un diagrama de secuencia, cuyos elementos son los siguientes:

- Usuario, App, Modo, AllLists y Base Datos Local, que se han descrito en el punto anterior.
- Lista: este componente muestra los elementos de una lista: el título, foto de la lista, los ejercicios, detalle de los ejercicios (foto, descripción y vídeo) y menús desplegables con varias opciones.
- Entrenamiento: en este componente se muestra cada ejercicio y los botones "Comenzar ejercicio", "Siguiente", "Guardar" y "Cancelar". Un usuario puede realizar un entrenamiento interactuando con los botones, que permiten medir el tiempo de realización y de descanso de los ejercicios.

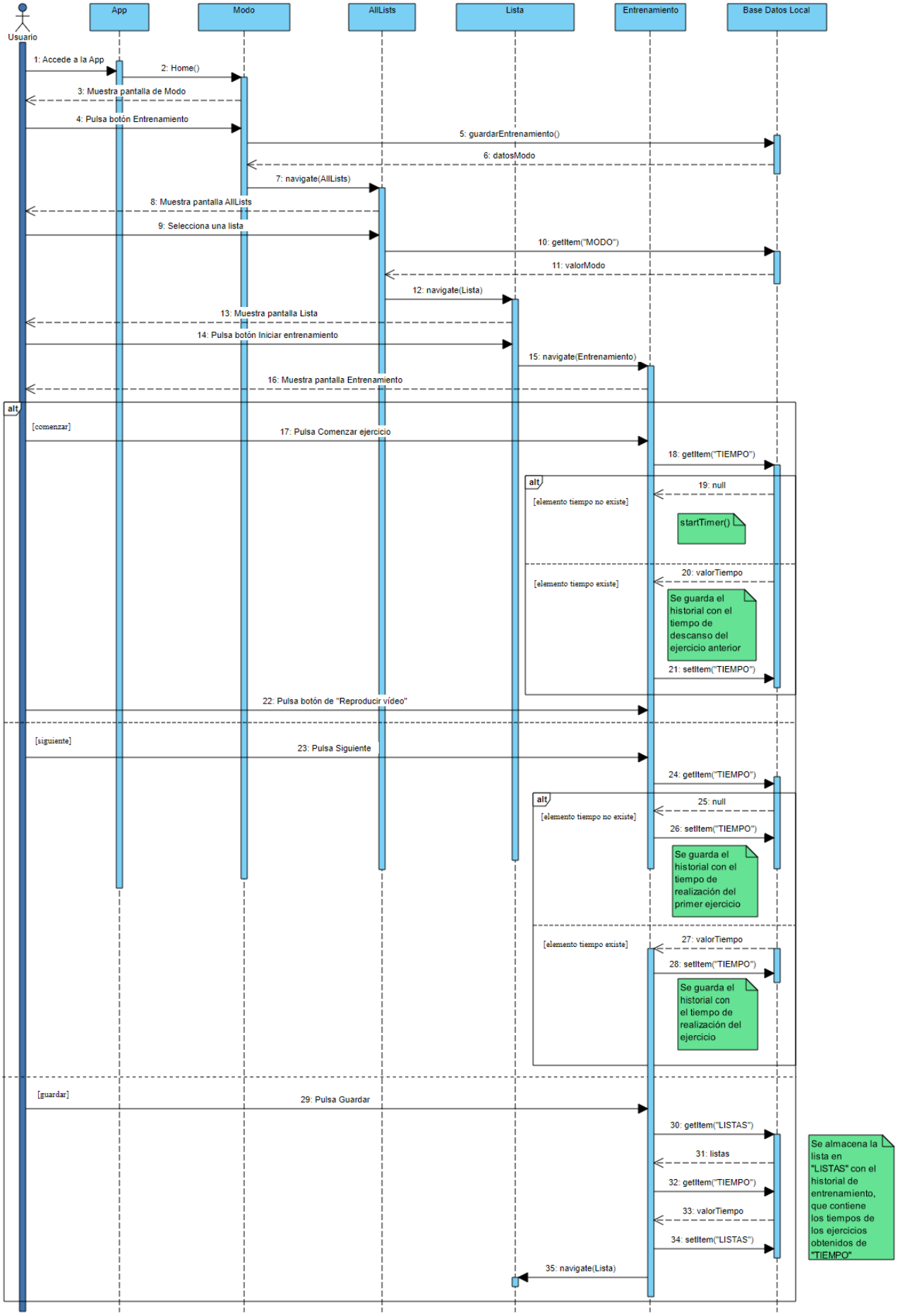

Ilustración 45 – Diagrama de secuencia para realizar un entrenamiento

4.2.4 Enviar una lista por correo electrónico

Un usuario puede enviar una lista por correo electrónico o Bluetooth. Se representa en la Ilustración 46 un diagrama de secuencia que muestra los procesos que tienen lugar al enviar una lista por correo electrónico, que son similares a los que se realizan para enviarla por Bluetooth. Los elementos que intervienen en dichos procesos se han mencionado anteriormente, excepto el elemento Gmail.

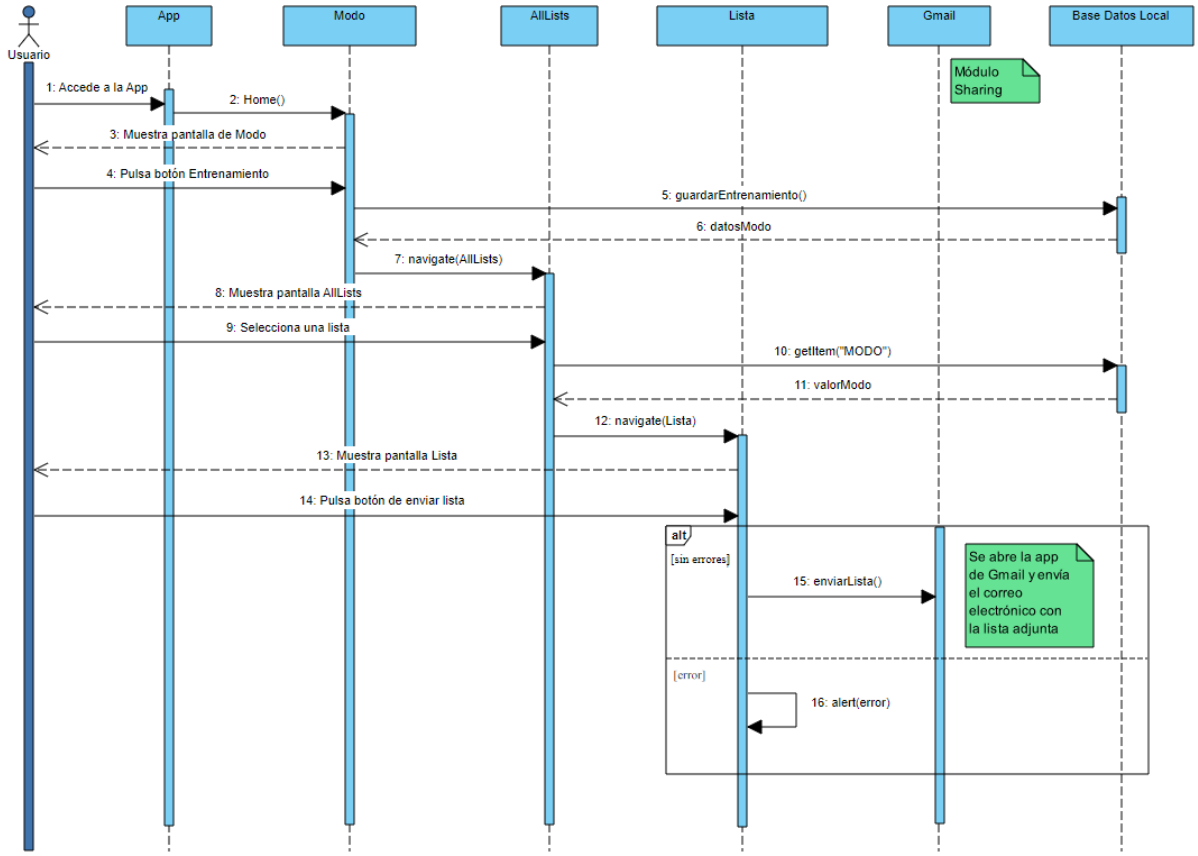

Ilustración 46 – Diagrama de secuencia para enviar una lista por correo electrónico

Para enviar una lista por correo electrónico y Bluetooth, se utiliza el método Sharing de Expo [34], que permite enviar una URL local del archivo elegido a varias aplicaciones, entre ellas Gmail y Bluetooth. En el mensaje 15 se llama al método Sharing.isAvailableAsync(), que devuelve true si la API Sharing se puede utilizar en la aplicación. Sin embargo, si no se puede utilizar, devuelve false y se muestra un error en la pantalla de la app.

La principal diferencia entre enviar una lista por correo electrónico y por Bluetooth es que, en el primer caso, se abre la aplicación de Gmail y, en el segundo caso, se abre Bluetooth dentro de los ajustes del dispositivo y se elige el dispositivo al que le va a enviar la lista. Tanto el usuario que envía como el que recibe la lista debe tener el Bluetooth activado en sus dispositivos respectivos.

4.2.5 Importar una lista

Una vez recibida una lista por correo electrónico o Bluetooth, el usuario puede importarla desde el explorador de archivos del dispositivo. Esto se muestra en el diagrama de secuencia de la Ilustración 47.

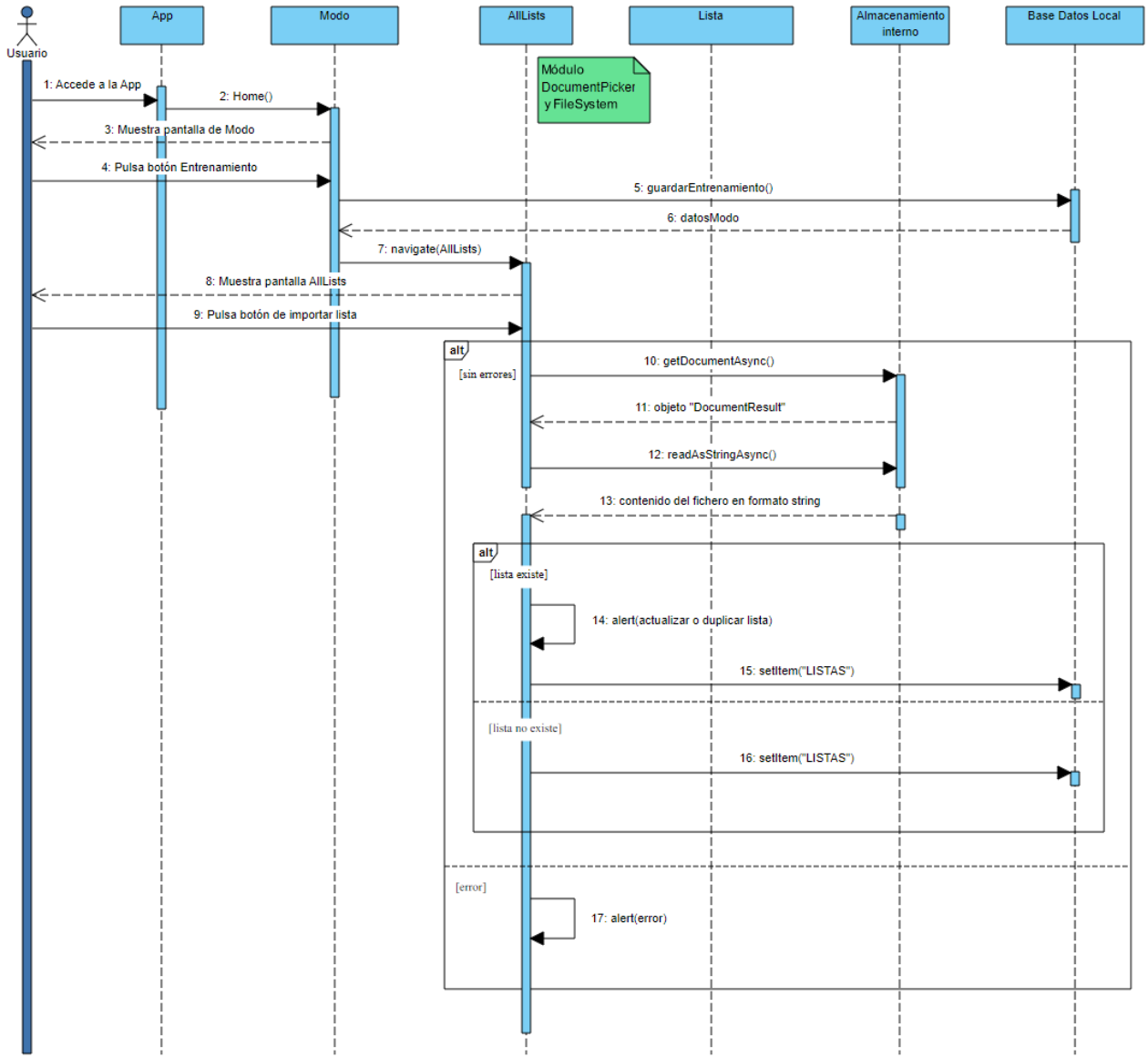

Ilustración 47 – Diagrama de secuencia para importar una lista

Para importar la lista se utiliza el módulo de Expo llamado DocumentPicker [35], que proporciona acceso a la interfaz de usuario del sistema para seleccionar archivos del almacenamiento interno del dispositivo. El archivo escogido se guarda por defecto en el directorio caché interno de la app. Si se ha podido realizar la acción, devuelve un objeto de tipo DocumentResult que contiene la uri del archivo. Además, se utiliza el módulo de Expo denominado FileSystem [36]. Este módulo proporciona acceso al sistema de archivos interno de la aplicación, donde se encuentra el directorio caché. Para obtener el archivo elegido se llama al método FileSystem.readAsStringAsync(), al que se le pasa la uri y devuelve el contenido del archivo en formato cadena o "string".

4.3 Modelo de datos

En este apartado se detallará el modelo de datos de la aplicación. Se han usado componentes funcionales para desarrollar la app, y se definen las relaciones entre ellos en la Ilustración 48 a continuación:

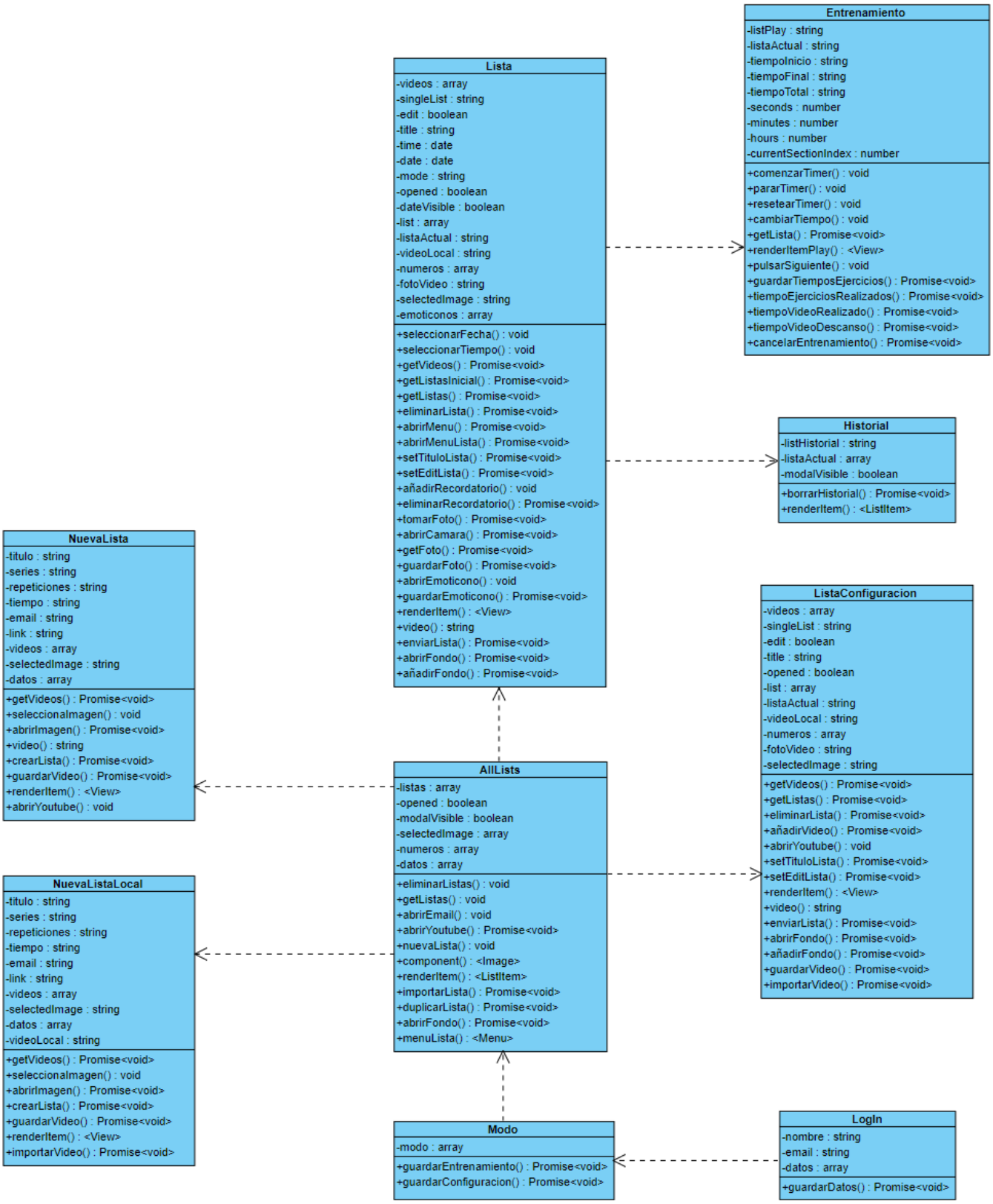

Ilustración 48 – Modelo de datos de la aplicación

Para mayor claridad, se han incluido los métodos más importantes de los componentes, sin los parámetros asociados. Se puede observar que el componente Lista contiene una mayor cantidad de métodos que el resto. Esto es debido a que hay un mayor número de operaciones a realizar con una lista, que han sido mencionadas en el apartado 4.1 en este capítulo.

5 INTERFAZ DE USUARIO Y FUNCIONALIDAD

El diseño accesible es un buen diseño.

- Steve Ballmer -

En este capítulo se detallará la interfaz de usuario de la aplicación. Se mostrará la distribución de sus elementos y sus funcionalidades, y se definirá el tipo de interacciones que se puede realizar con cada uno de los componentes.

5.1 Pantalla de registro e inicio de sesión

Ilustración 49 – Pantalla de registro e inicio de sesión

En la Ilustración 49 se puede observar la pantalla inicial, que se muestra cuando el usuario no está autenticado. Dispone de un título que se denomina "Login", el logo de la aplicación situado debajo de este, y un formulario que contiene dos campos: nombre y correo electrónico. Estos campos sonobligatorios por lo que, si no se rellenan, saltará una alerta informando al usuario de que debe

rellenarlos. En el caso de que se introduzca una dirección de correo electrónico errónea, aparecerá una alerta indicando el error.

Un ejemplo de nombre y correo electrónico de un usuario se indica en la Ilustración 50 a continuación:

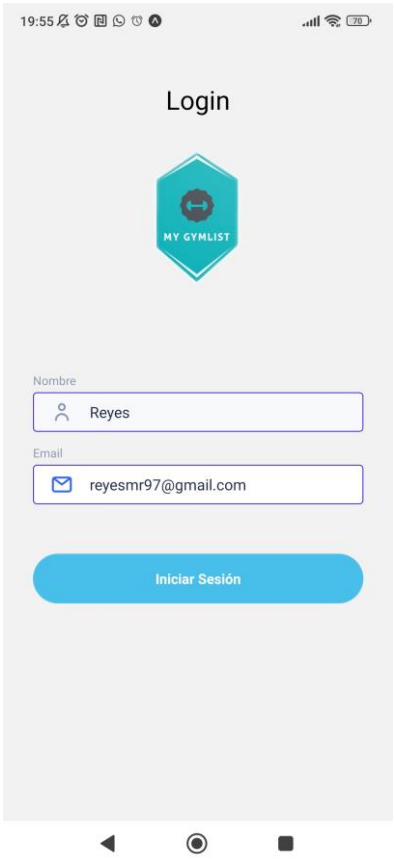

Ilustración 50 – Ejemplo de nombre y correo electrónico del usuario

El correo electrónico que ha introducido el usuario será añadido por defecto en las listas de ejercicios creadas y, por tanto, no tendrá que volver a introducirlo cada vez que cree una lista. Cuando el usuario envíe una lista, el destinatario del mensaje podrá ver el correo electrónico del usuario que la ha creado y, por tanto, saber a quién pertenece la lista.

5.2 Pantalla de modo de visualización

Ilustración 51 – Pantalla de modo de visualización

En la pantalla de la Ilustración 51 aparecen dos botones: “Entrenamiento” y “Configuración”. Al pulsar cualquiera de los dos botones, se mostrará la pantalla donde se visualizan las listas. Sin embargo, al seleccionar una de las listas, se podrán realizar acciones diferentes en función del modo de visualización escogido.

5.3 Pantalla de lista de listas

Ilustración 52 – Pantalla de lista de listas

En la Ilustración 52 se puede observar la lista que recoge todas las listas de ejercicios de la aplicación. Las funcionalidades que ofrece son las que se especifican a continuación:

5.3.1 Añadir fondo de lista de listas

Ilustración 53 – Menú de la pantalla de listas de listas

Para añadir un fondo de pantalla, el usuario debe pulsar en el menú de la esquina superior derecha de la Ilustración 53, donde aparecen dos opciones: "Añadir fondo de pantalla" y "Eliminar listas". Una vez seleccionada la opción de añadir un fondo, se mostrarán las imágenes del almacenamiento interno del dispositivo. Tras escoger una de ellas, se colocará la imagen en el fondo de la pantalla tal y como aparece en la Ilustración 54:

Ilustración 54 – Ejemplo de fondo de pantalla personalizado

5.3.2 Eliminar listas

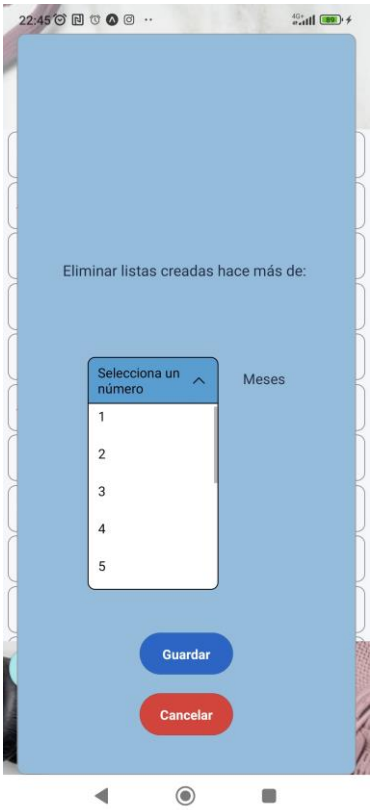

Ilustración 55 – Opción eliminar listas

Si el usuario escoge la opción de eliminar listas, aparecerá un componente de tipo “Modal” [37] mostrado en la Ilustración 55, que permite que se visualice contenido encima de una vista. En este componente se muestra un botón con una lista de números, donde el usuario puede escoger el número de meses deseado. Al pulsar el botón “Guardar”, se eliminan las listas que se han creado en una fecha anterior al número de meses escogido.

5.3.3 Duplicar una lista

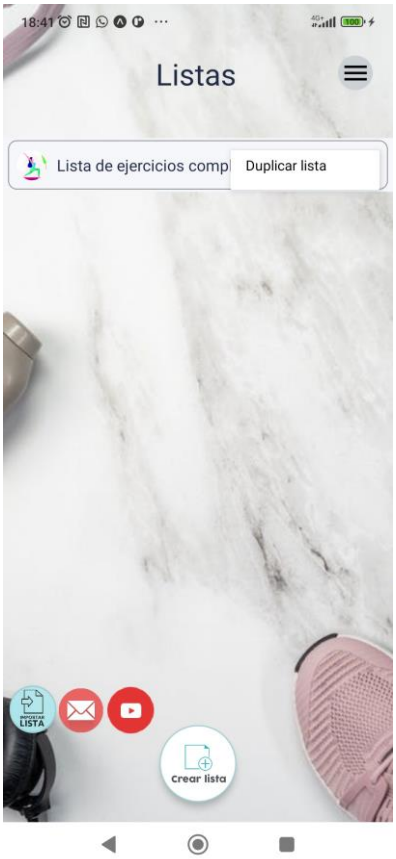

Ilustración 56 - Opción duplicar lista

En el menú situado a la derecha de cada lista de ejercicios aparece una opción llamada “Duplicar lista”, que se puede observar en la Ilustración 56. Si el usuario escoge esta opción aparece una lista idéntica a la anterior, solo que con id y título diferentes, tal y como se aprecia en la Ilustración 57:

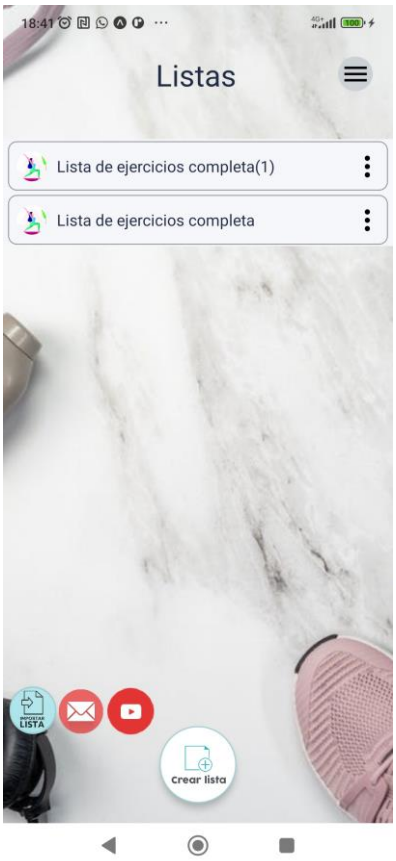

Ilustración 57 – Ejemplo de duplicar lista

5.3.4 Insertar una lista

Cuando un usuario quiere insertar una lista en la aplicación, debe pulsar en el botón “Importar lista” que se sitúa en la esquina inferior izquierda de la pantalla. Tras pulsarlo, aparecerán los ficheros del almacenamiento interno y escogerá la lista que desea insertar. Una vez elegida la lista con formato .TXT, se añadirá a la lista de listas. En el caso de que se quiera importar una lista que ya exista en la aplicación, aparecerá una alerta en la pantalla tal y como se muestra en la Ilustración 58:

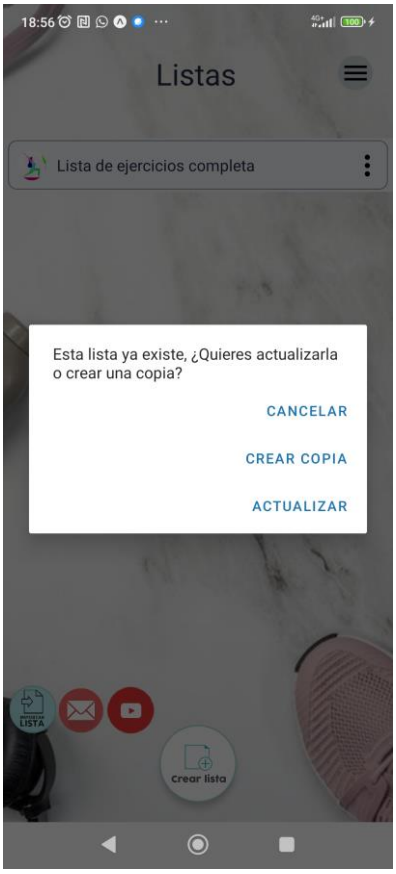

Ilustración 58 – Insertar una lista existente

Si se elige la opción de "Crear copia", se añadirá la lista de la misma forma en la que aparece en la Ilustración 57. En cambio, si se elige la opción de "Actualizar", se mantendrá la lista existente, pero se actualizarán sus campos.

5.3.5 Abrir aplicación de Gmail y YouTube

En la esquina inferior izquierda de la pantalla se sitúa un botón que abre la aplicación de Gmail y otro que abre la aplicación de YouTube. En el caso de que un usuario quiera descargar una lista recibida por Gmail, puede acceder a la aplicación pulsando el botón correspondiente sin tener que salir de la app. Asimismo, si quiere acceder a la aplicación de YouTube para ver los vídeos de los ejercicios que desee y obtener el enlace lo puede hacer desde esta pantalla y desde otras que se mencionarán más adelante.

5.3.6 Crear una lista nueva

En la parte inferior de la pantalla se sitúa el botón de "Crear lista". En la Ilustración 59 se puede observar que, una vez pulsado, aparecerá una alerta que mostrará dos opciones: "Añadir vídeos de YouTube" y "Añadir vídeos en local".

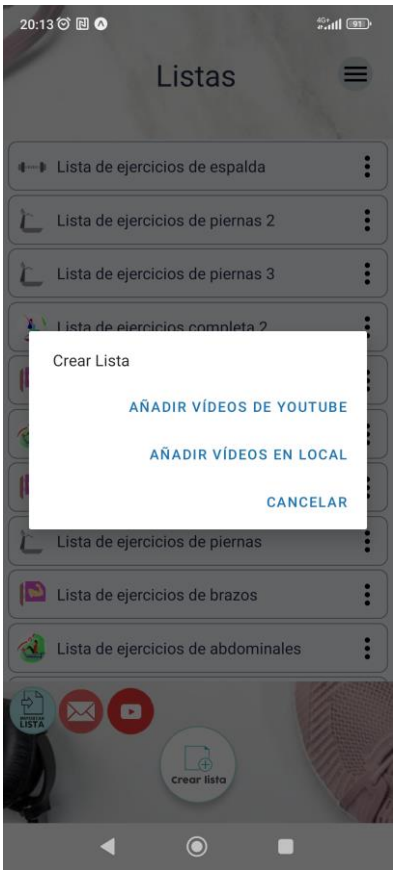

Ilustración 59 – Ejemplo de alerta de Crear lista

5.3.7 Cambiar el correo electrónico del usuario

Si el usuario desea cambiar su correo electrónico, debe seleccionar la opción "Cambiar correo electrónico", y aparecerá una alerta de confirmación. Si el usuario pulsa la opción "Sí", se mostrará la pantalla de "Inicio de sesión", donde podrá volver a introducir un nombre y correo electrónico. El nuevo correo electrónico se almacenará en las listas nuevas que se creen en la aplicación.

5.4 Pantalla de crear lista

En función de la opción que el usuario escoja a la hora de crear una lista nueva, se abrirá una pantalla diferente. Si escoge la opción de "Añadir vídeos de YouTube", se abrirá la pantalla de "Nueva lista". Sin embargo, si escoge la opción de "Añadir vídeos en local", se abrirá la pantalla de "Nueva lista local".

5.4.1 Pantalla de nueva lista local

Ilustración 60 – Pantalla de nueva lista local

En la Ilustración 60 se puede observar la pantalla para crear una lista de ejercicios en local. Consta de un botón en forma de flecha que, una vez pulsado, vuelve a la pantalla anterior; un botón para añadir una imagen; un campo para insertar el título de la lista; el correo electrónico del usuario; un botón para añadir un ejercicio y un botón para crear la lista. En la Ilustración 61 se muestra la alerta que aparece cuando se pulsa el botón para añadir una imagen, donde se plantean dos posibilidades: escoger una imagen del dispositivo o elegir un icono de la aplicación.

Ilustración 61 – Alerta para seleccionar una imagen

Si el usuario selecciona la opción “Insertar imagen de la galería”, se abrirá el explorador de archivos del dispositivo. En la Ilustración 62 se muestra el almacenamiento interno del dispositivo. Si selecciona la opción “Icono de la app”, se mostrará una ventana de tipo “Modal” donde podrá elegir el icono deseado, tal y como aparece en la Ilustración 63:

Ilustración 62 – Imágenes del almacenamiento interno

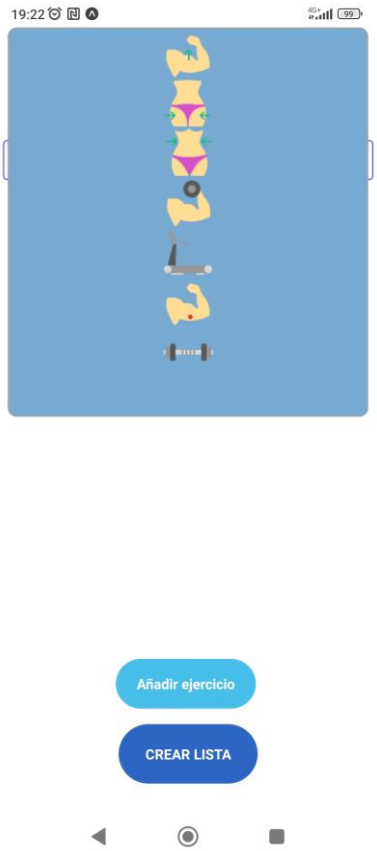

Ilustración 63 – Ventana de iconos de la aplicación

Al pulsar el botón de “Añadir ejercicio”, se muestra una ventana con un formulario que aparece en la Ilustración 64 a continuación:

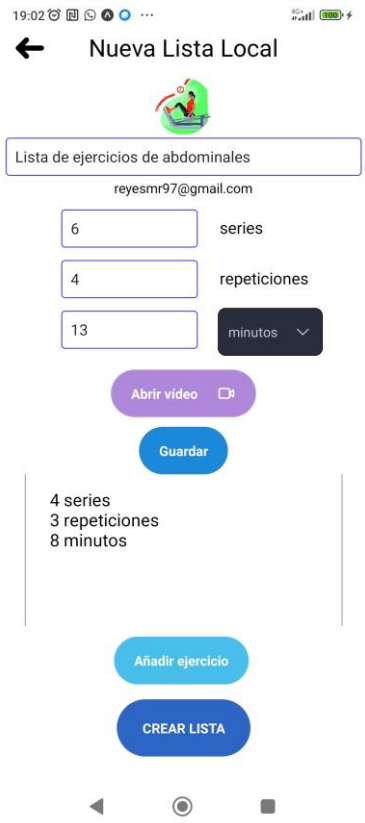

Ilustración 64 – Ejemplo de añadir ejercicio con vídeo local

En la ventana de "Añadir ejercicio", el usuario podrá insertar el número de series, repeticiones y el tiempo del ejercicio. Estos campos son opcionales, ya que habrá ciertos tipos de ejercicios que no necesiten estas propiedades. Para escoger el tiempo, puede seleccionar si se mide en horas o minutos en el botón de "tiempo". Además, para insertar un vídeo podrá pulsar el botón "Abrir vídeo", que muestra los vídeos del almacenamiento interno del dispositivo. Podrá visualizar el vídeo antes de guardar el ejercicio, y cambiarlo si desea insertar otro en su lugar. Una vez seleccionadas todas las opciones, y tras pulsar el botón de guardar, aparecerá el ejercicio en la lista. Asimismo, se podrá visualizar la lista con todos los ejercicios añadidos antes de pulsar el botón de "Crear lista". Los campos que son obligatorios son: el título de la lista, la imagen y el vídeo del ejercicio. Si no se rellenan, saltará una alerta.

5.4.2 Pantalla de nueva lista

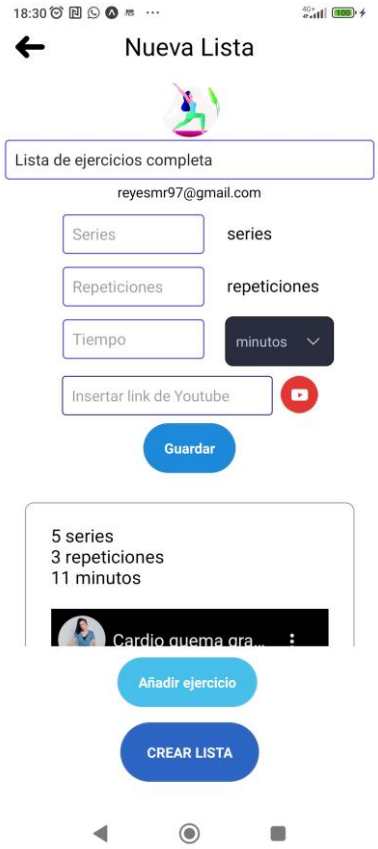

Ilustración 65 – Pantalla de nueva lista

En la Ilustración 65 se muestra la pantalla para crear una lista nueva con ejercicios de YouTube. El procedimiento es el mismo que para crear una lista con ejercicios en local excepto que, en este caso, hay un botón para acceder a la aplicación de YouTube, y así obtener un enlace a un vídeo que se insertará en el formulario.

5.5 Pantalla de lista de ejercicios modo Entrenamiento

Al seleccionar una lista, se pueden abrir dos tipos de pantallas diferentes: la pantalla de “Lista” si se ha escogido el modo “Entrenamiento”, y la pantalla de “Lista Configuración” si se ha escogido el modo “Configuración”.

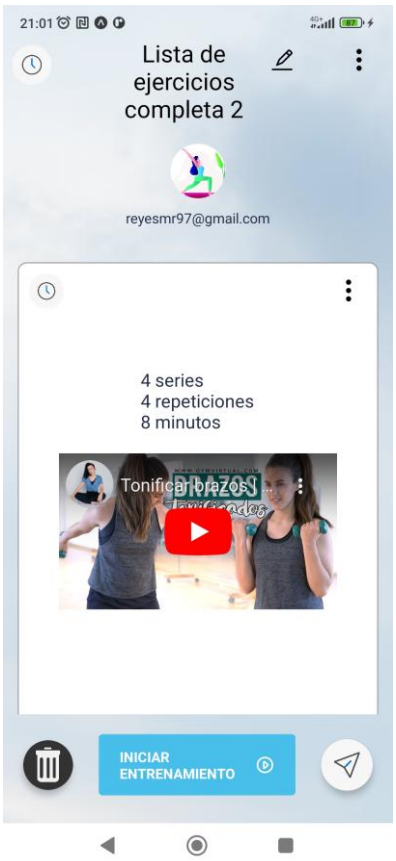

Ilustración 66 – Pantalla de lista de ejercicios del modo Entrenamiento

En la Ilustración 66 se detallan los elementos de una lista de ejercicios en el modo “Entrenamiento”, que son los siguientes: un icono para indicar si la lista o el ejercicio se ha realizado; un icono para editar el título de la lista; un menú desplegable tanto en la lista como en el ejercicio; un botón para eliminar la lista; un botón para iniciar un entrenamiento y un botón para enviar la lista.

5.5.1 Editar título de una lista

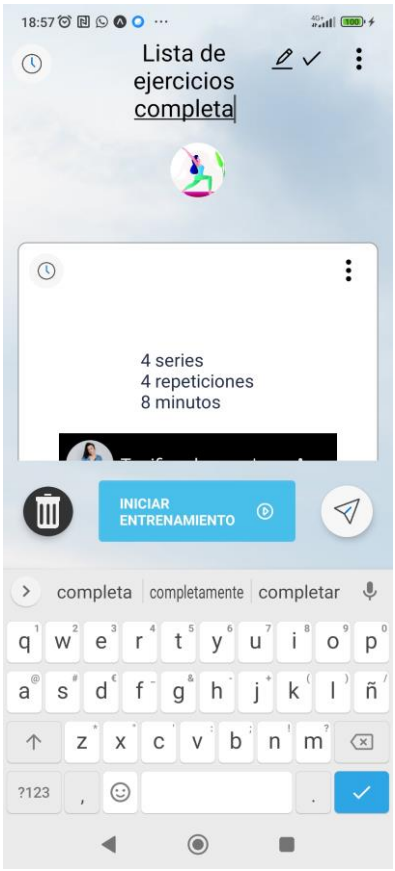

Ilustración 67 – Editar título de una lista

Para modificar el título de una lista, se debe presionar en el icono de "editar lista" situado al lado del título. Tras pulsarlo, aparecerá un icono de confirmación y el teclado del teléfono tal y como se muestra en la Ilustración 67. Una vez editado el título, se podrá pulsar el botón de confirmación del teclado o el icono de confirmación ubicado al lado del título para guardar los cambios.

5.5.2 Marcar una lista como realizada

En la Ilustración 68 se puede observar el menú de una lista situado en la esquina superior derecha, donde aparecen las siguientes opciones:

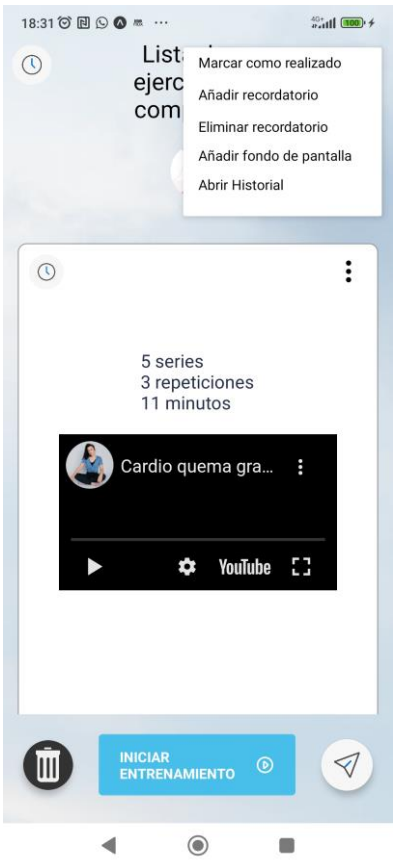

Ilustración 68 – Menú de la lista de ejercicios modo Entrenamiento

Si se escoge la opción de “Marcar como realizado”, se cambiará el icono ubicado en la esquina superior izquierda y aparecerá un símbolo de “realizado”, como se puede observar en la Ilustración 69:

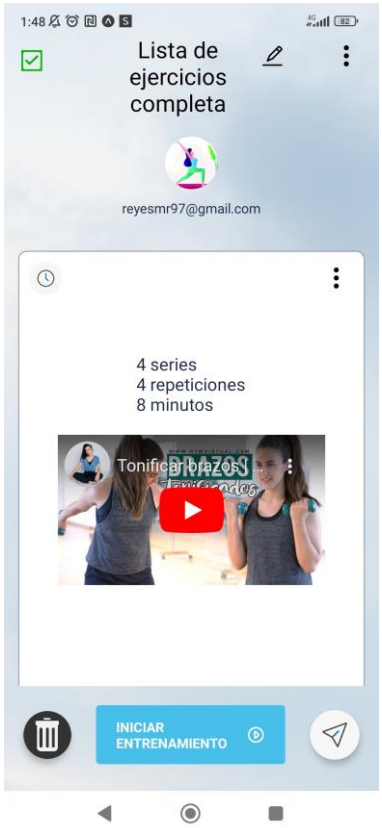

Ilustración 69 – Ejemplo de marcar lista como realizada

Si se elige la opción de “Marcar como realizado” en uno de los ejercicios, aparecerá el mismo símbolo en el ejercicio y cambiará a estado “realizado”.

5.5.3 Añadir recordatorio de una lista

Al pulsar el botón de “Añadir recordatorio”, aparece una ventana de tipo “Modal” que se muestra en la Ilustración 70 a continuación:

Ilustración 70 – Ventana de añadir recordatorio

Para añadir un recordatorio, el usuario puede seleccionar el día pulsando el botón “Seleccionar día”, que abre el calendario, y puede seleccionar la hora pulsando el botón “Seleccionar hora”. En la Ilustración 71 y 72 se muestran el calendario y el reloj respectivamente para elegir la fecha y la hora:

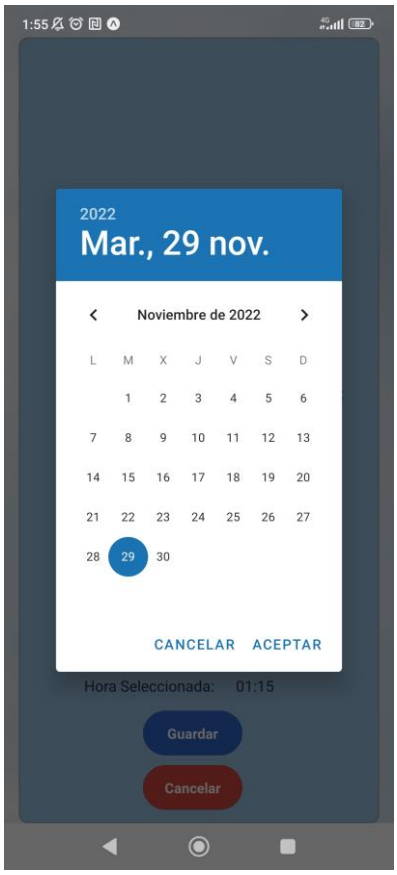

Ilustración 71 – Ejemplo de calendario para añadir recordatorio

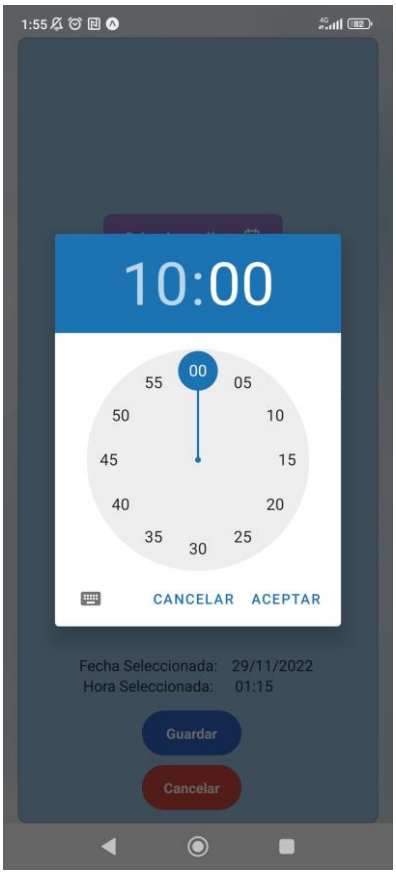

Ilustración 72 – Ejemplo de hora para añadir recordatorio

Una vez seleccionada la fecha y la hora, se puede elegir si se prefiere recibir la notificación con una antelación de 0, 10, 15, 30 o 60 minutos. En la parte inferior de la ventana se muestra la fecha y la hora en la cual se recibirá la notificación. Una vez finalizado, se pulsará el botón "Guardar" y se habrá guardado el recordatorio. La notificación se recibirá de forma periódica el mismo día de la semana que se ha elegido. Es decir, si se ha elegido un lunes, se repetirá cada lunes a la misma hora. En el caso de que quiera dejar de recibir un recordatorio para esa lista, puede pulsar el botón de "Eliminar recordatorio". En la Ilustración 73 se muestra un ejemplo de notificación de un recordatorio de una lista. La notificación aparece 60 minutos antes de la hora del entrenamiento.

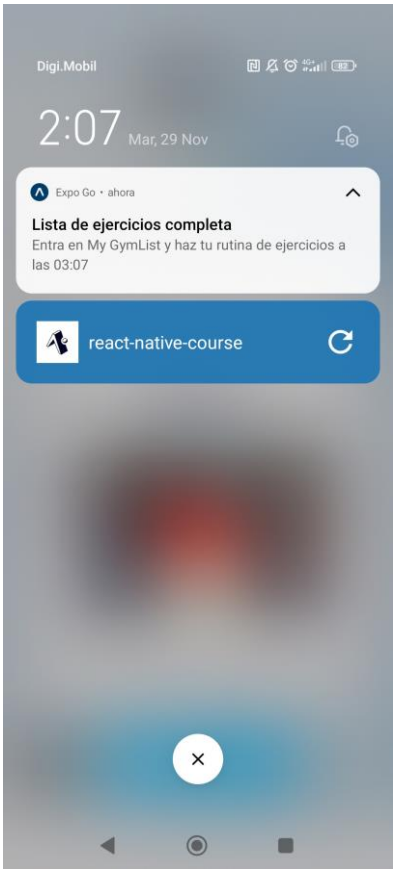

Ilustración 73 – Ejemplo de notificación de recordatorio de una lista

5.5.4 Añadir fondo de lista

Para añadir un fondo de pantalla en la lista se utiliza el mismo procedimiento que en el apartado 5.3.1 excepto que, en este caso, se guarda la uri del fondo escogido en un campo de la lista. En la Ilustración 74 se muestra un ejemplo de un fondo personalizado para una lista:

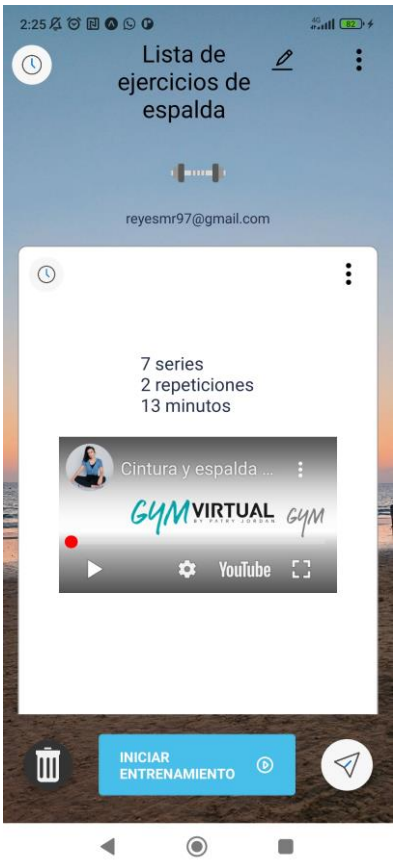

Ilustración 74 – Ejemplo de fondo personalizado para una lista

5.5.5 Eliminar una lista

Al pulsar el botón de “Eliminar lista” situado en la esquina inferior izquierda, aparece una alerta tal y como se indica en la Ilustración 75:

Ilustración 75 – Ejemplo de alerta para eliminar una lista

Al pulsar “Sí”, la lista se eliminará y se volverá a la pantalla anterior (de lista de listas).

5.5.6 Enviar una lista

Una vez pulsado el botón de “Enviar lista”, se mostrará el menú de opciones para compartir la lista que aparece en la Ilustración 76:

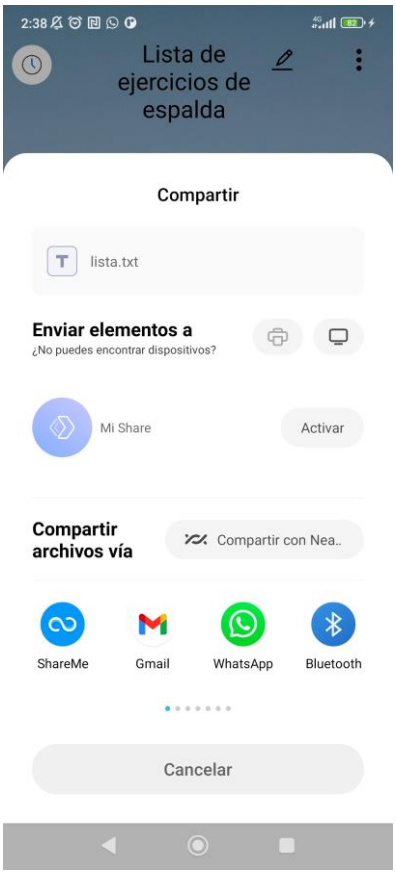

Ilustración 76 – Menú de opciones para enviar una lista

Entre las opciones que se muestran se puede escoger enviar la lista por Gmail o por Bluetooth. En el caso de enviar la lista por Gmail, se abrirá la aplicación de Gmail y se mostrará un correo electrónico como el de la Ilustración 77 donde se introducirá el destinatario, el asunto y la descripción. La lista se adjuntará de forma automática.

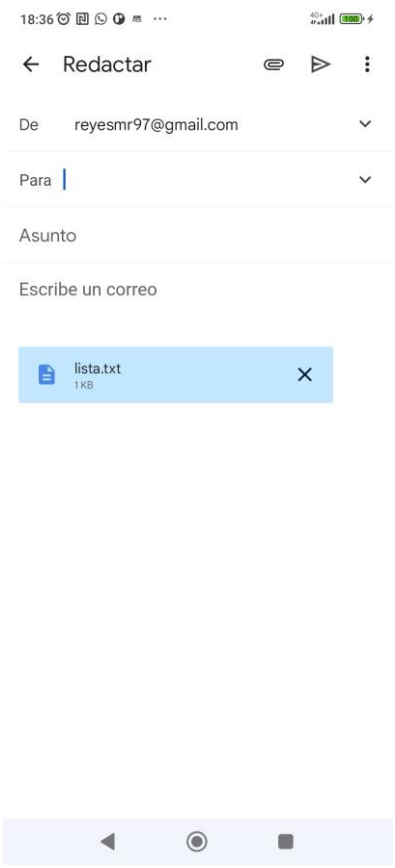

Ilustración 77 – Ejemplo de envío de una lista por Gmail

En el caso de escoger la opción de enviar una lista por Bluetooth, aparecerá una ventana donde se podrá activar el Bluetooth en el dispositivo, si aún no está activado. Seguidamente, se abrirá el apartado Bluetooth en los ajustes para encontrar dispositivos cercanos. Una vez seleccionado el dispositivo, se enviará y el destinatario la recibirá correctamente. En la Ilustración 78 aparece el mensaje de envío correcto de la lista por Bluetooth:

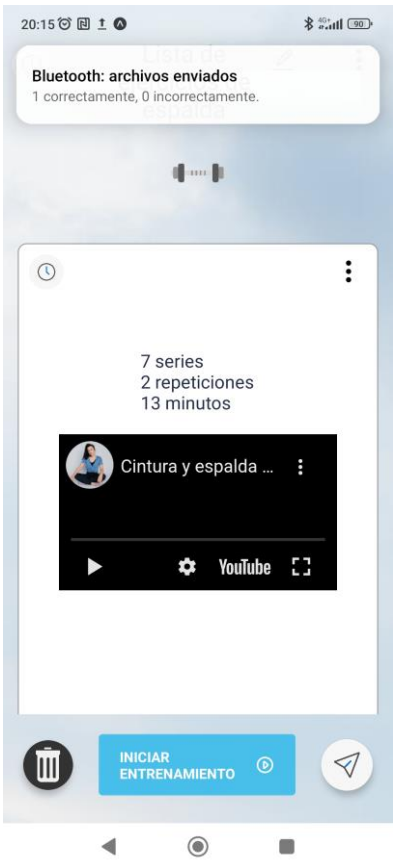

Ilustración 78 – Ejemplo de envío de una lista por Bluetooth

5.5.7 Modificar ejercicios de una lista

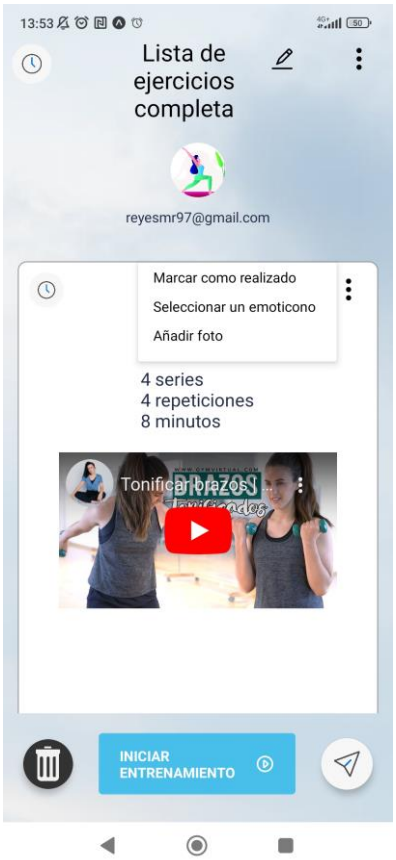

Ilustración 79 – Menú desplegable de un ejercicio

En la Ilustración 79 se detalla el menú de opciones para modificar un ejercicio de una lista. Se pueden elegir tres opciones: “Seleccionar un emoticono”, “Marcar como realizado” y “Añadir foto”.

5.5.7.1 Seleccionar un emoticono

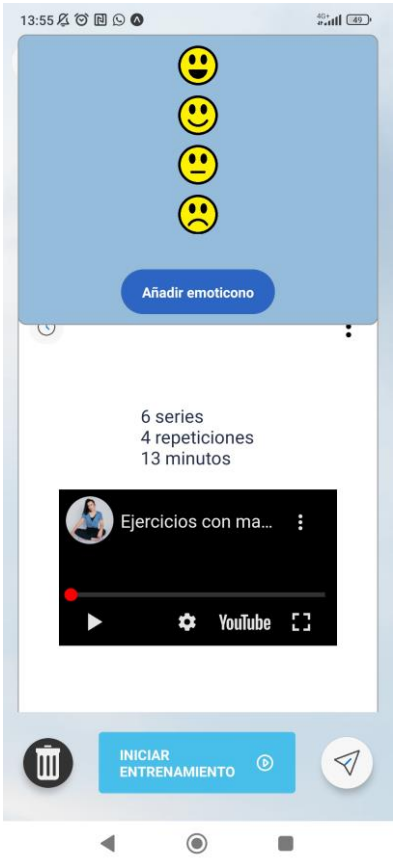

Ilustración 80 – Ventana para añadir un emoticono a un ejercicio

Cuando un usuario quiere indicar qué le ha parecido un ejercicio o, lo que es lo mismo, cómo se ha sentido realizándolo, puede añadir un emoticono que lo exprese. Para ello, debe pulsar el botón de "Seleccionar un emoticono". Una vez pulsado, se mostrará una ventana de tipo "Modal" con la lista de emoticonos disponibles en la aplicación, tal y como se muestra en la Ilustración 80. Una vez elegido y seleccionado el botón de guardar emoticono, se añadirá en la parte superior del ejercicio, como se puede observar en la Ilustración 81:

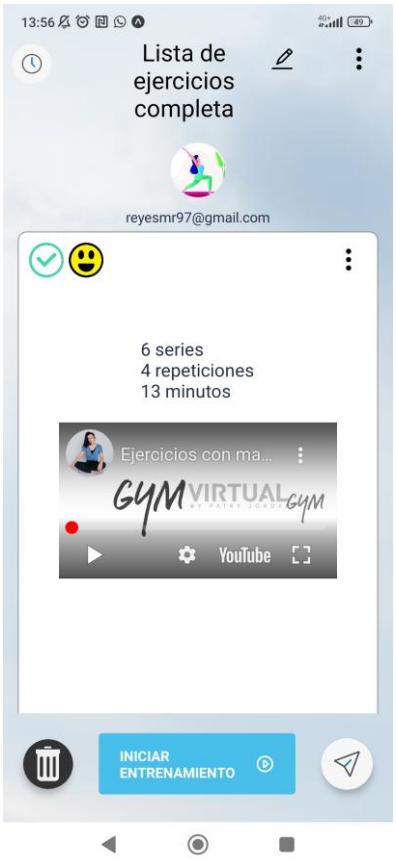

Ilustración 81 – Ejemplo de ejercicio realizado con emoticono

En este ejemplo, se le ha asignado un emoticono al ejercicio y se ha marcado el ejercicio como realizado. En el caso de que el usuario quiera dejar de asignar el ejercicio como "realizado", puede seleccionar la opción de "Desmarcar como realizado". Esta opción solo aparece si el ejercicio ha sido marcado como realizado. Para marcar y desmarcar una lista como realizada se lleva a cabo el mismo proceso que para un ejercicio.

5.5.7.2 Añadir una foto a un ejercicio

Cuando un usuario quiere añadir una foto a un ejercicio de una lista, debe pulsar el botón "Añadir foto". Una vez pulsado, se mostrará la cámara de Expo con el botón de "tomar foto", "encender y apagar flash", "voltear cámara" y "cancelar". En la Ilustración 82 se puede observar un ejemplo:

Ilustración 82 – Ejemplo de añadir foto a un ejercicio

Una vez tomada la foto, aparecerá una ventana de tipo “Modal” con la foto y los botones “Añadir foto” y “Repetir foto”. Si el usuario no está conforme con la foto tomada, puede volver a hacerla pulsando “Repetir foto”. Si está conforme, al pulsar en “Añadir foto” se guardará y aparecerá en el ejercicio. En la Ilustración 83 y 84 se muestra la ventana de añadir foto y el resultado respectivamente.

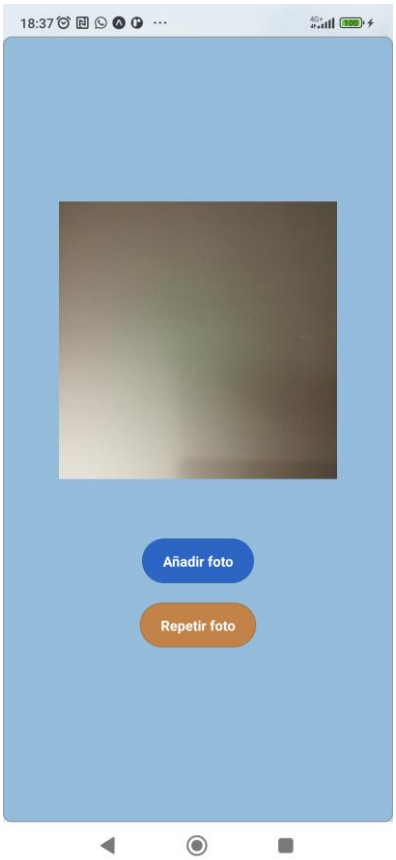

Ilustración 83 – Ventana de añadir foto a un ejercicio

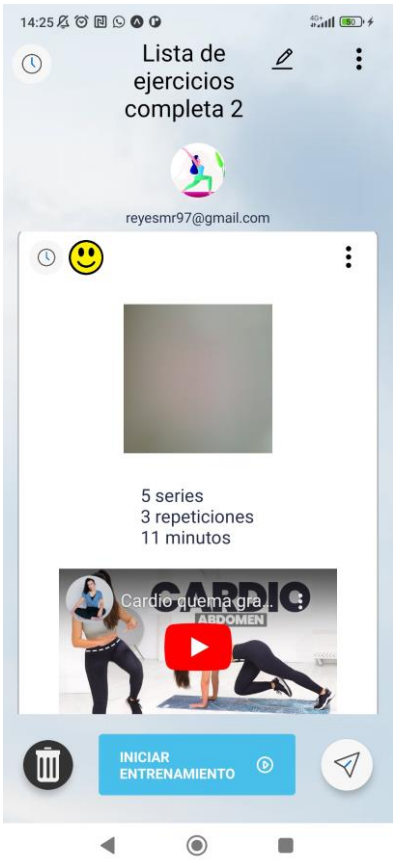

Ilustración 84 – Ejemplo de foto en un ejercicio

En el caso de que el usuario quiera borrar la foto tomada, aparecerá la opción de “Borrar foto” en el menú despegable del ejercicio y, una vez pulsada, se mostrará una alerta donde el usuario podrá confirmar que desea eliminar la foto.

Estas tres funcionalidades son opcionales a la hora de realizar un entrenamiento de una lista y se han añadido para ofrecer una mayor personalización de una lista al usuario.

5.6 Pantalla entrenamiento de una lista

Para realizar un entrenamiento de una lista, se debe seleccionar una lista en el modo “Entrenamiento” y pulsar en el botón de “Iniciar entrenamiento”. En la Ilustración 85 se muestra un ejemplo de la pantalla de entrenamiento de la lista seleccionada:

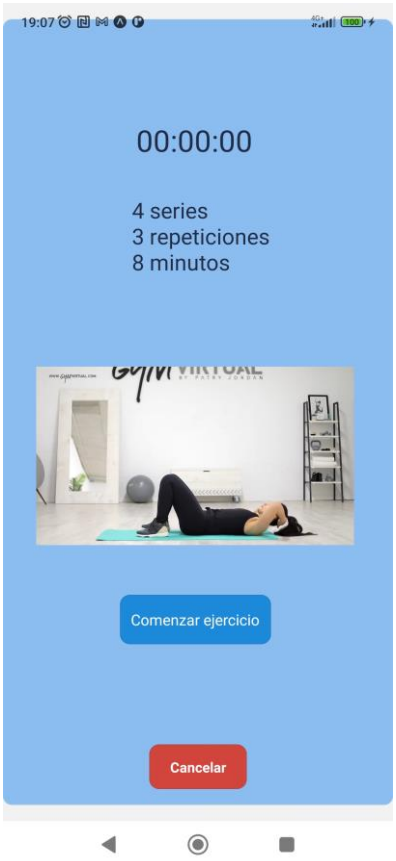

Ilustración 85 – Pantalla de entrenamiento de una lista

Se puede observar que aparece un cronómetro, el número de series, repeticiones y tiempo, y el vídeo del ejercicio. Si se pulsa el botón de “Comenzar ejercicio”, el cronómetro empieza a contar y aparece el botón “Siguiente” en la pantalla, tal y como se muestra en la Ilustración 86:

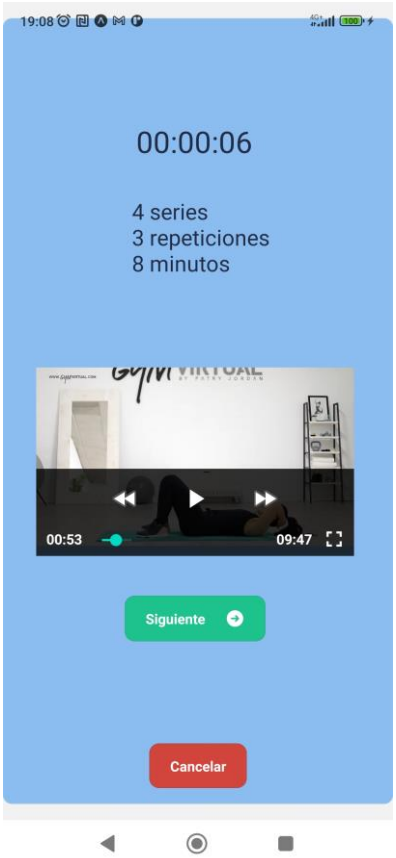

Ilustración 86 – Ejemplo de entrenamiento pulsando comenzar

Una vez terminado el ejercicio, se pulsará el botón "Siguiente" para que aparezca el siguiente ejercicio de la lista. Además, se habrá guardado el tiempo de realización del ejercicio tras pulsar el botón. Para empezar el siguiente ejercicio, se pulsará el botón "Comenzar ejercicio", y de esta forma se guardará el tiempo de descanso del ejercicio anterior. Una vez terminado el último ejercicio de la lista, aparecerá el botón "Guardar" y, al pulsarlo, se pausará el cronómetro y se almacenarán los resultados en el historial de la lista.

5.7 Pantalla de historial de una lista

Para visualizar el historial de una lista, se debe seleccionar la opción "Abrir historial" del menú desplegable de la lista. En la Ilustración 87 se detalla la pantalla del historial de una lista:

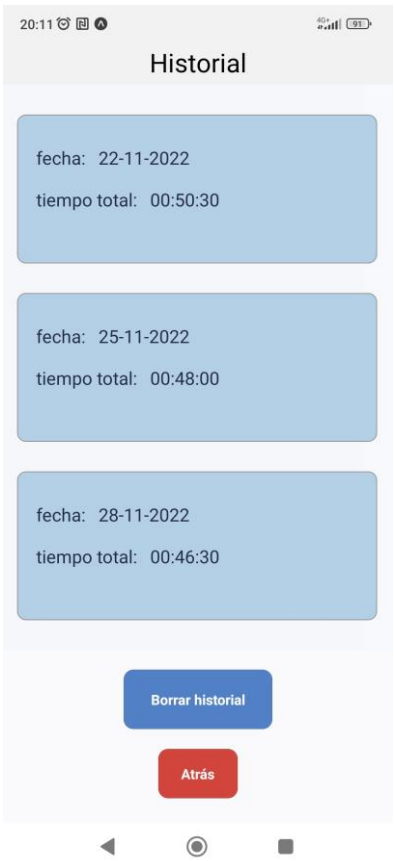

Ilustración 87 – Ejemplo de historial de una lista

En el historial se almacena cada historial de entrenamiento de la lista, donde se muestra la fecha del entrenamiento y el tiempo total empleado en realizarlo. Asimismo, se muestra el botón "Borrar historial" y "Atrás" en la parte inferior de la pantalla.

Para visualizar un historial de entrenamiento concreto, se pulsará en el elemento y se abrirá una pantalla que muestra el número de series y repeticiones, y el tiempo de realización y de descanso de cada ejercicio de la lista. En la Ilustración 88 se puede observar un ejemplo de un historial de entrenamiento específico:

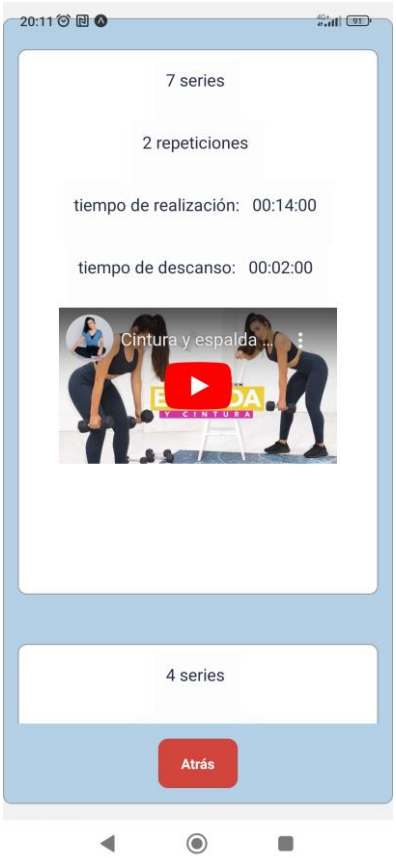

Ilustración 88 – Ejemplo de un historial de entrenamiento

Si se pulsa el botón “Borrar historial”, se eliminará el historial completo de la lista y aparecerá por pantalla el texto “El historial está vacío”, y desaparecerá el botón “Borrar historial”, como se muestra en la Ilustración 89:

Ilustración 89 – Ejemplo de historial vacío de una lista

5.8 Pantalla de lista de ejercicios modo Configuración

En el modo “Configuración”, se muestra la lista de listas igual que en el modo “Entrenamiento” pero, al seleccionar una lista, aparecen algunas funcionalidades diferentes. En la Ilustración 90 se detallan los elementos de una lista en el modo “Configuración”:

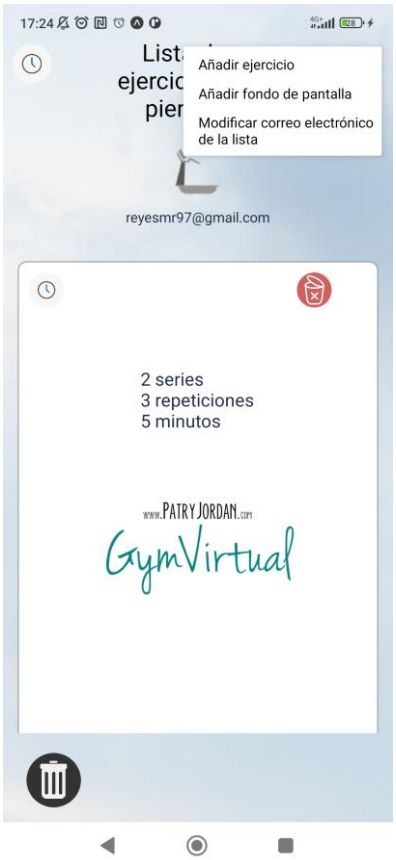

Ilustración 90 – Ejemplo de pantalla lista en el modo Configuración

Las funcionalidades que tienen en común una lista en el modo "Entrenamiento" y en el modo "Configuración" son: eliminar la lista, enviar la lista y añadir fondo de pantalla. Sin embargo, las funcionalidades que se encuentran en una lista en el modo "Configuración" y que no aparecen en el modo "Entrenamiento" son: "Añadir ejercicio", "Modificar correo electrónico" y "Eliminar ejercicio". De esta forma, se separan las funcionalidades que se relacionan con realizar un entrenamiento y las que se atribuyen a configurar una lista y sus elementos.

5.8.1 Añadir un ejercicio a una lista creada

Al pulsar el botón de "Añadir ejercicio", en función de si la lista es de ejercicios de YouTube o de ejercicios del almacenamiento local, aparecerá una ventana de tipo "Modal" diferente. En la Ilustración 91 aparece un ejemplo de una ventana para añadir un ejercicio local, y en la Ilustración 92 aparece una ventana para añadir un ejercicio de YouTube:

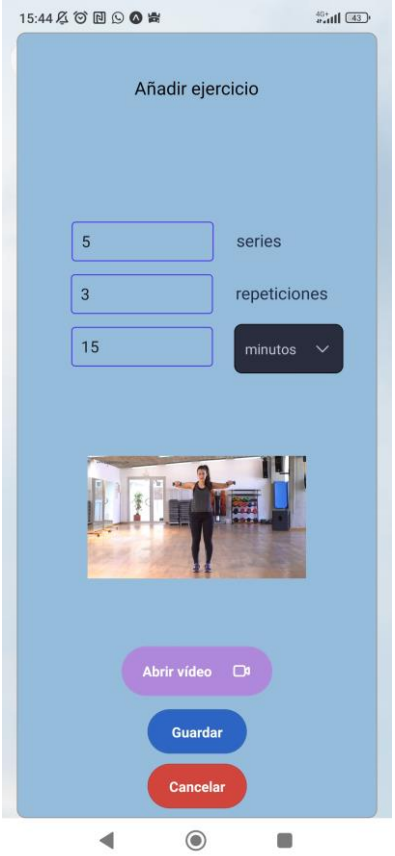

Ilustración 91 – Ejemplo de añadir un ejercicio local

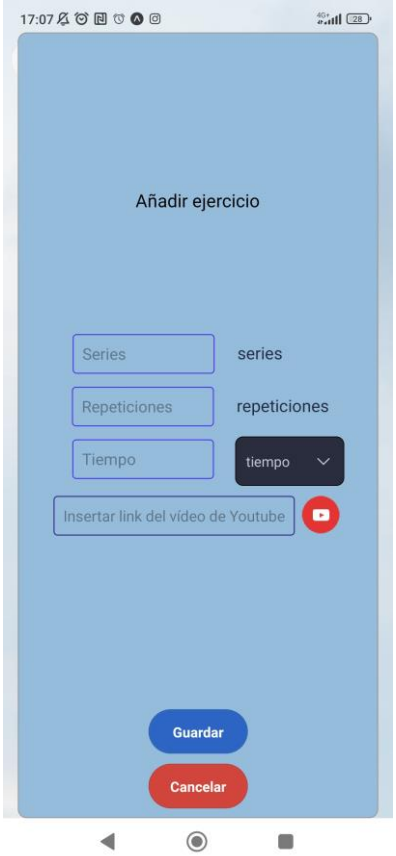

Ilustración 92 – Ejemplo de añadir un ejercicio de YouTube

5.8.2 Eliminar un ejercicio de una lista

Al pulsar el botón de “Eliminar ejercicio”, aparece una alerta de confirmación como en el ejemplo de la Ilustración 93. Tras presionar la opción “Sí”, se borrará el ejercicio de la lista.

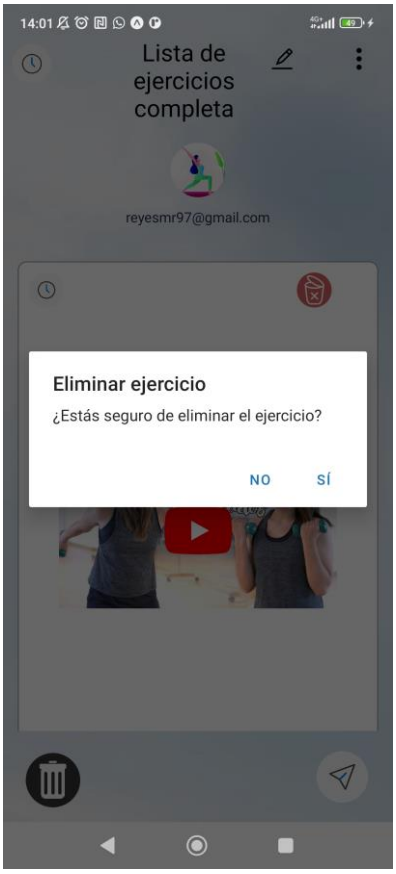

Ilustración 93 – Ejemplo de eliminar un ejercicio de la lista

5.8.3 Modificar el correo electrónico

Al pulsar el botón de "Modificar correo electrónico de la lista", aparecerá una ventana de tipo "Modal" donde se podrá introducir el correo electrónico nuevo. Si el usuario pulsa el botón de "Guardar" y el campo está vacío, aparecerá una alerta indicándolo. Si se ha introducido un correo electrónico con formato inválido, se alertará del error. En la Ilustración 94 se muestra la ventana para modificar el correo electrónico:

Ilustración 94 – Ventana para modificar el correo electrónico de la lista

5.9 Salir de la aplicación

Para salir de la aplicación, se debe pulsar el botón "volver atrás" del dispositivo desde la pantalla de "Modo". En la Ilustración 95 aparece la alerta que se muestra tras pulsar el botón, para confirmar si el usuario desea salir de la aplicación.

Ilustración 95 – Alerta para salir de la aplicación

6 INSTALACIÓN Y PRUEBAS

Las pruebas son un proceso infinito de comparación de lo invisible con lo ambiguo para evitar que ocurra lo impensable a lo anónimo.

- James Marcus Bach -

La comprobación y prueba del funcionamiento de la aplicación es esencial para detectar posibles errores y solventarlos de la forma más eficaz posible. Asimismo, es necesario verificar que las funcionalidades se lleven a cabo de la manera esperada y que la interfaz de usuario se visualice de forma correcta en los dispositivos.

La instalación de la aplicación y las pruebas en dispositivos iOS y Android se detallarán en este capítulo.

6.1 Instalación y pruebas en Android

Para probar la aplicación en un dispositivo Android existen dos opciones: realizar pruebas en un emulador con Android Studio o en un dispositivo físico con Android. En este caso, se ha escogido la opción de instalar la aplicación en un dispositivo físico ya que, de este modo, se puede probar el funcionamiento para un uso real de la app.

6.1.1 Generación del archivo de instalación

Se creará el archivo .APK de la aplicación para realizar pruebas en el dispositivo. Existe la opción de generar un .APK para dispositivos Android y un .API para probar la aplicación en dispositivos iOS. Para crear cualquiera de los dos ficheros y subirlo a la Play Store o App Store, se puede utilizar el servicio "Classic Build" [38], que permite crear un archivo independiente que se puede instalar o subir a la tienda correspondiente en función del sistema operativo. En el caso de iOS, solo se puede crear un archivo de instalación con una cuenta de desarrollador de Apple, es decir, se debe subir a la App Store para poder realizar pruebas en el dispositivo físico. En el caso de Android, no es necesario crear una cuenta de desarrollador en Google Play para instalarlo.

Otro servicio que permite crear un fichero de instalación es "EAS Build" [39], que automatiza el proceso de creación del archivo binario para subirlo a la tienda de App Store o Google Play. Además, permite instalar la aplicación directamente en un emulador o dispositivo Android y en un emulador iOS. Esta forma de instalación es más sencilla, ya que no requiere tener una cuenta de desarrollador.

En este caso, se ha utilizado el servicio "Classic Build" para generar el archivo de instalación para Android, ya que no se va a subir la aplicación a Google Play y el proceso es más directo de esta forma. Los pasos para generar el fichero .APK son los siguientes:

En primer lugar, se debe configurar el archivo "app.json" que aparece en la Ilustración 96 para indicar el nombre del paquete de instalación y la versión para iOS y Android. Seguidamente, se debe escribir el comando "expo build:android" y seleccionar la opción de generar el "apk", tal y como se muestra en la Ilustración 97:

```
"ios": {
  "supportsTablet": true,
  "infoPlist": {
    "NSPhotoLibraryUsageDescription": "Allow $(PRODUCT_NAME) to access your photos.",
    "NSPhotoLibraryAddUsageDescription": "Allow $(PRODUCT_NAME) to save photos."
  },
  "bundleIdentifier": "com.reyes.mygymlist"
},
"android": {
  "package": "com.reyes.mygymlist",
  "versionCode": 1,
  "adaptiveIcon": {
    "foregroundImage": "./assets/logo.png",
    "backgroundColor": "#FFFFFF"
  },
  "useNextNotificationsApi": true,
  "permissions": [
    "android.permission.READ_EXTERNAL_STORAGE",
    "android.permission.WRITE_EXTERNAL_STORAGE",
    "android.permission.ACCESS_MEDIA_LOCATION"
  ]
},
```

Ilustración 96 – Modificación del archivo app.json para generar el .APK

```
C:\Windows\system32\cmd.exe

C:\Users\reyes\react-native-course>expo build:android

   There is a new version of expo-cli available (6.0.8).
   You are currently using expo-cli 5.4.12
   Install expo-cli globally using the package manager of your choice;
   for example: `npm install -g expo-cli` to get the latest version

expo build:android has been superseded by eas build. Learn more: https://blog.expo.dev/turtle-goes-out-to-sea-d334db2a6b
60

Run the following:

› npm install -g eas-cli
› eas build -p android https://docs.expo.dev/build/setup/

expo build:android will be discontinued on January 4, 2023 (35 days left).

√ Choose the build type you would like: » apk
Checking if there is a build in progress...

Accessing credentials for reyesmr in project react-native-course

› Expo SDK: 44.0.0
› Release channel: default
› Workflow: Managed
```

Ilustración 97 – Comando para generar el .APK

En segundo lugar, se debe generar una clave para poder crear el fichero .APK. Se puede generar de forma independiente o permitir que Expo la genere de forma automática, que es mucho más eficaz.

Finalmente, se puede descargar el .APK desde la cuenta de Expo y guardarlo en el dispositivo físico. En la Ilustración 98 se muestra el mensaje de confirmación para instalar la aplicación en el dispositivo Android:

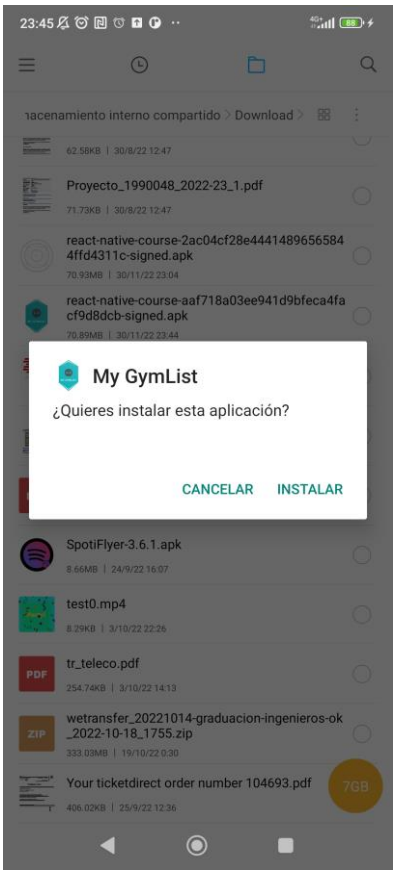

Ilustración 98 – Instalación de la aplicación en un dispositivo Android

6.2 Instalación y pruebas en iOS

La instalación de la aplicación en un dispositivo físico iOS solo se puede realizar subiendo la aplicación a la App Store, por lo que es necesario tener una cuenta de desarrollador en Apple. Por tanto, para realizar pruebas sin tener una cuenta de desarrollador se debe tener un emulador iOS. Este emulador funciona solo en un ordenador con sistema operativo macOS utilizando Xcode, o en una máquina virtual instalada en un ordenador Windows. En este caso, ha habido restricciones para instalar la aplicación en iOS, ya que no he tenido a mi disposición un ordenador con macOS y en el portátil que he utilizado para desarrollar la aplicación no funcionaba de forma eficiente la máquina virtual. Por tanto, he realizado las pruebas de la app utilizando la aplicación Expo Go desde un dispositivo móvil iPhone. Para ello, ha sido necesario actualizar el Expo SDK a la versión compatible con Expo Go. En la Ilustración 99 se puede observar el comando "expo upgrade" utilizado para el proceso de actualización:

```
C:\Users\reyes\react-native-course>expo upgrade
Your git working tree is clean
To revert the changes after this command completes, you can run the following:
  git clean --force && git reset --hard
√ You are currently using SDK 44.0.0. Would you like to update to the latest version, 47.0.0? ... no

√ Choose a SDK version to upgrade to: » 45.0.0


/ Installing the expo@^45.0.0 package...
```

Ilustración 99 – Actualización de Expo SDK para compatibilidad con Expo Go

Una vez actualizado el Expo SDK y, con ello, los módulos de Expo utilizados en el proyecto, se ejecuta el comando "expo start" para iniciar el servidor de desarrollo de Expo para probar la aplicación, tal y como se muestra en la Ilustración 100:

Ilustración 100 – Inicio del servidor de desarrollo de Expo y generación de código QR para Expo Go

Escaneando el código QR generado con la cámara del iPhone, se inicia la aplicación y se pueden llevar a cabo las pruebas. Algunas de las pantallas de la aplicación se muestran en la Ilustración 101, 102, 103, 104, 105 y 106:

Ilustración 101 – Pantalla de iniciar sesión en iOS

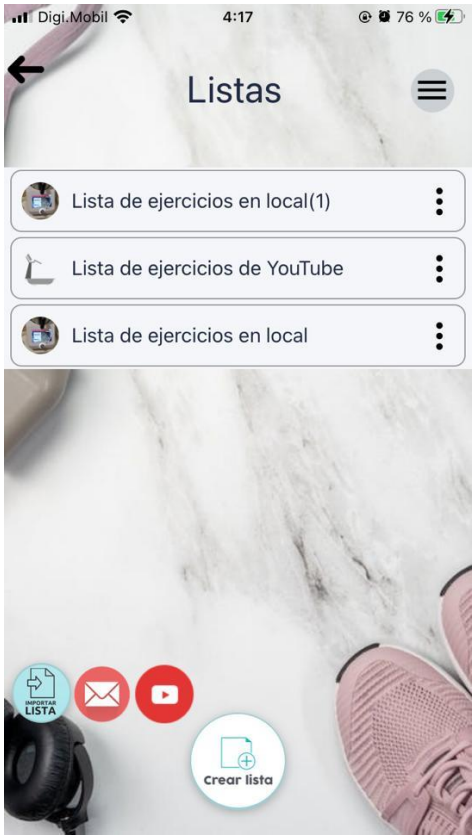

Ilustración 102 – Pantalla de listas de listas en iOS

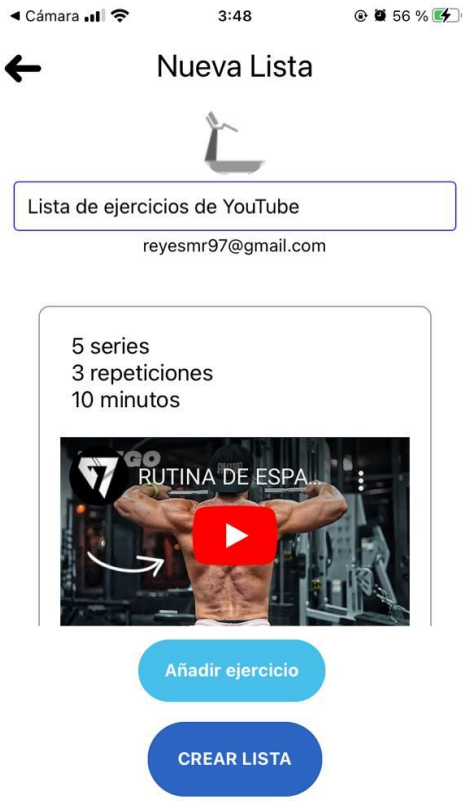

Ilustración 103 – Creación de una lista nueva en iOS

Ilustración 104 – Pantalla Entrenamiento en iOS

Ilustración 105 – Ejemplo de seleccionar fecha para un recordatorio en iOS

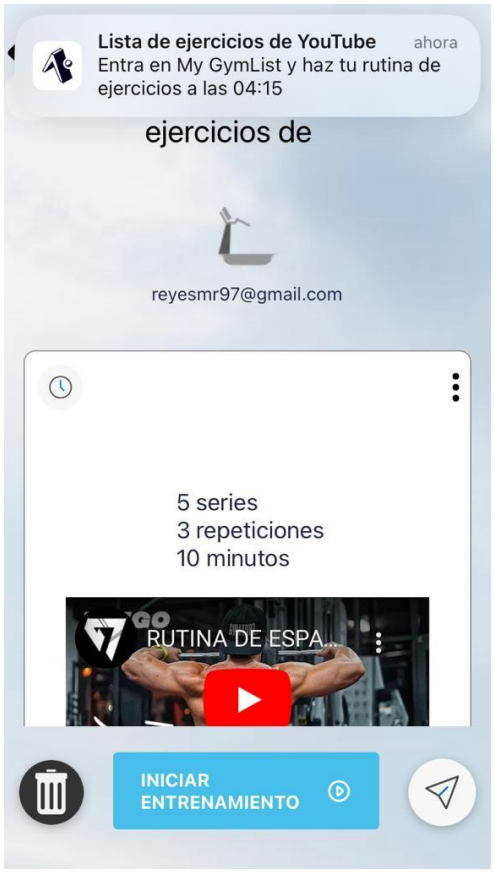

Ilustración 106 – Pantalla Lista con notificación en iOS

7 LÍNEAS DE MEJORA Y CONCLUSIONES

El propósito de este capítulo es sintetizar y reunir las tecnologías aprendidas durante el desarrollo del proyecto, proponer posibles mejoras y extraer conclusiones a raíz de los resultados obtenidos.

El uso de módulos de Expo ha permitido que se pueda acceder a funciones y aplicaciones del dispositivo, como la cámara y el sistema de archivos, que de otra forma habría sido más complejo. Sin embargo, estos módulos están en constante cambio y se debe tener el Expo SDK actualizado cada cierto tiempo para que funcionen correctamente. Además, algunos de estos módulos se vuelven obsoletos con el tiempo, y es necesario comprobarlo de forma periódica para garantizar el correcto funcionamiento de la aplicación.

A la hora de realizar la instalación de la aplicación, el proceso ha sido más sencillo en el caso de Android, ya que solo ha sido necesario crear el archivo .APK ejecutando el comando correspondiente e instalarlo directamente en el dispositivo físico. No obstante, en el caso de iOS ha habido un inconveniente, que es el de necesitar un ordenador con macOS para instalar la app en el emulador de iOS con Xcode, del que no disponía durante el desarrollo del proyecto. La otra opción disponible era crear una cuenta de desarrollador de Apple, que rondaba los 99 dólares al año, para subir la aplicación a la App Store, que no era viable para el proyecto.

Para realizar las pruebas de la app, no ha habido mayores inconvenientes debido a que se ha utilizado la aplicación Expo Go en los dispositivos Android e iOS, que permite observar el comportamiento de la aplicación en tiempo real, actualiza los cambios de la interfaz de usuario en cortos periodos de tiempo y que indica los fallos en el momento. El único impedimento encontrado ha sido a la hora de instalar Expo Go en el iPhone, ya que la versión de Expo SDK del proyecto era inferior a la soportada en Expo Go, por lo que ha sido necesario actualizar la versión de Expo SDK de la 44 a la 45 y desinstalar algunos módulos que eran incompatibles o que estaban obsoletos en la nueva versión.

En cuanto a los objetivos del proyecto, analizando cada uno de ellos se puede concluir lo siguiente:

- Generar un código único en el lenguaje JavaScript que permita usar la aplicación en Android y en iOS: El desarrollo de la aplicación en React Native con Expo ha aportado grandes ventajas, como el generar un único código que se pueda utilizar para las plataformas con sistema operativo iOS y Android. Esto ha permitido que no haya sido necesario escribir código nativo en ningún momento, es decir, escribir en código Java o Kotlin en el caso de Android, o escribir en código ObjectiveC o Swift en el caso de iOS. Esto ha propiciado el no necesitar aprender más de un lenguaje de programación y poder profundizar en el lenguaje JavaScript, ahorrando tiempo y recursos en el proceso.
- Comunicación entre Usuario y Especialista mediante correo electrónico y Bluetooth: en un principio se propuso realizar una aplicación para el Usuario y otra para el Especialista con funcionalidades diferentes. En cambio, este objetivo fue modificado y se decidió crear una única aplicación que pueda usar cualquier tipo de usuario ya que, al no hacer uso de una API de autenticación ni de un servidor, no fue necesario distinguir funcionalidades entre los dos roles.

  El módulo "Sharing" de Expo ha permitido que el envío de listas por correo electrónico (Gmail en concreto) y Bluetooth sea eficiente y sencillo. Si se hubiera desarrollado el código en Android Studio o Xcode, se habría necesitado utilizar librerías diferentes y métodos más complejos para poder enviar las listas de ambas formas. Sin embargo, Expo no proporciona ningún módulo para comprimir una carpeta en formato .zip, por lo que no es posible enviar una lista de ejercicios en local. Por tanto, una posible mejora sería exportar el proyecto para poder escribir código nativo y utilizar la librería correspondiente para comprimir los vídeos y enviar la lista. No obstante, si el archivo .zip tiene un tamaño mayor al

soportado por Gmail o Bluetooth, no sería posible compartir la lista. En ese caso, una solución posible sería permitir al usuario autenticarse con su cuenta de Google Drive desde la app para poder subir la carpeta con los vídeos de la lista, y compartirla.

- Almacenar los datos del usuario en una base de datos local: esto ha sido posible gracias a la librería "AsyncStorage" de React Native, que permite almacenar los datos en la aplicación sin tener que utilizar un servidor de base de datos en la nube. Una ventaja de no depender de un servidor para almacenar las listas es que no es necesario mantener un servidor y se puede ahorrar el coste que eso conlleva. En cambio, una desventaja es que no se puede acceder a las listas de ejercicios almacenadas en otro dispositivo. Para obtener las listas habría que enviarlas e importarlas una a una en el otro dispositivo, lo cual llevaría más tiempo. Por tanto, una posible mejora sería almacenar las listas de cada usuario en una base de datos remota.
- Reproducción de listas de entrenamiento con vídeos de YouTube y vídeos del almacenamiento local: el módulo "AV" de Expo y la librería "React Native YouTube iframe" de React Native ha permitido que se puedan reproducir vídeos en local y de YouTube respectivamente. Una ventaja de que estos vídeos se almacenen en el dispositivo es que se puede usar la aplicación y realizar un entrenamiento para una lista de ejercicios en local sin necesidad de una conexión a Internet. Solo es necesaria en el caso de realizar un entrenamiento de una lista de ejercicios de YouTube.

ANEXO A: INSTALACIÓN Y DESPLIEGUE

En este anexo se explicará el proceso llevado a cabo para crear y desplegar el proyecto.

A.1 Instalación del entorno

Primero, se debe instalar Node.js que, en función del sistema operativo, se utilizará un método distinto. En este caso, se utilizará el método para un ordenador con Windows 10. Para instalarlo, primero debemos acceder a la página web oficial y descargar el instalador para Windows [40] que aparece en la Ilustración 107:

Ilustración 107 – Descargar instalador de Node.js para Windows

Una vez descargado, se debe ejecutar el instalador y seleccionar las opciones requeridas de la Ilustración 108. Se puede observar que también se instalará el Node Package Manager (npm) [41], que es el sistema de gestor de paquetes para Node.js. Permite instalar componentes y módulos de React Native y Expo desde la ventana de comandos.

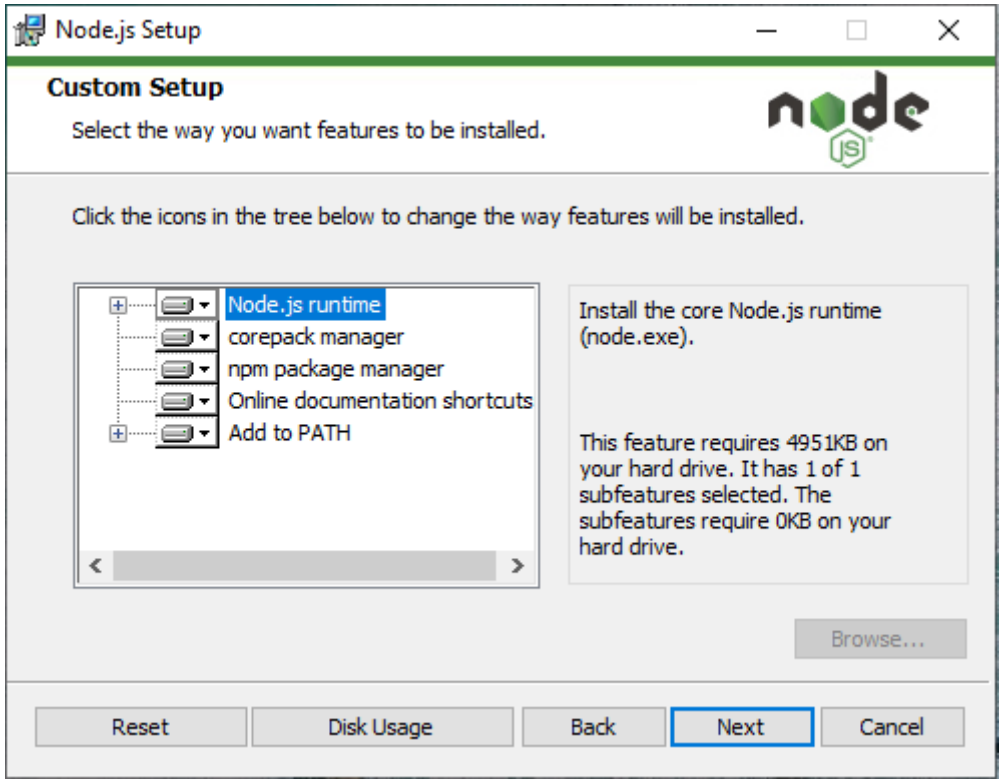

Ilustración 108 – Ejecutar instalador de Node.js

Tras finalizar el proceso de instalación, se puede comprobar la versión de Node.js y de npm instaladas en el sistema ejecutando los comandos que aparecen en la Ilustración 109:

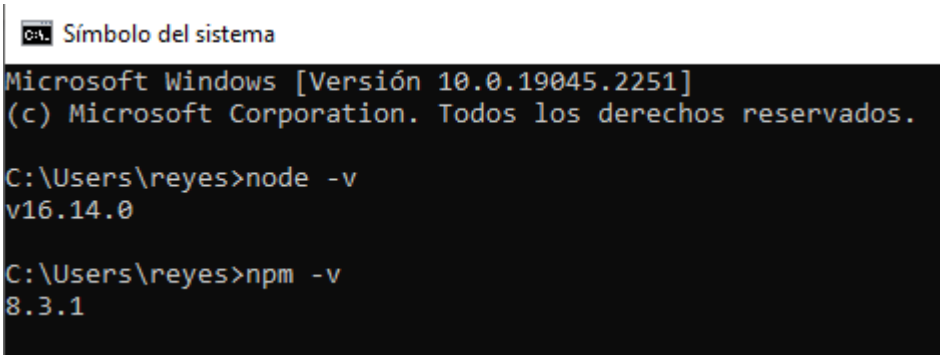

Ilustración 109 – Comprobar la versión de Node.js y npm

Para escribir el código del proyecto se ha empleado la herramienta Visual Studio Code. Para poder instalarla, se debe acceder a la página oficial [42] y seleccionar el archivo de instalación para Windows que aparece en la Ilustración 110:

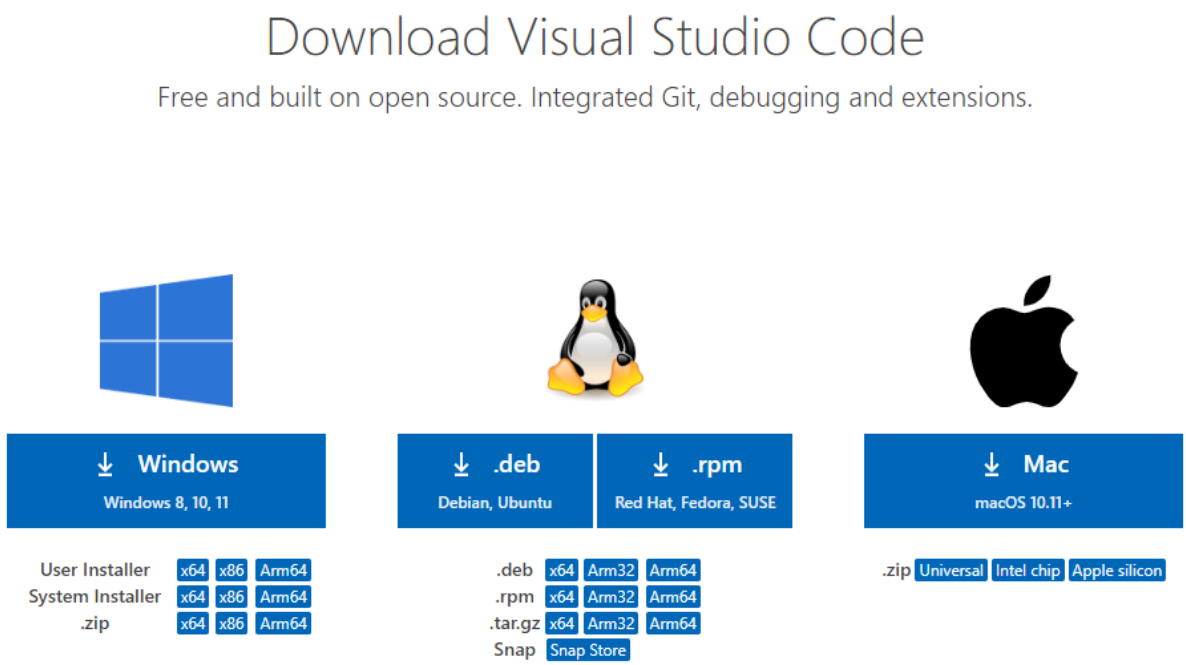

Ilustración 110 – Descargar archivo de instalación de Visual Studio Code para Windows

Una vez descargado, se debe abrir el paquete de instalación y escoger la configuración deseada, tal y como se muestra en la Ilustración 111:

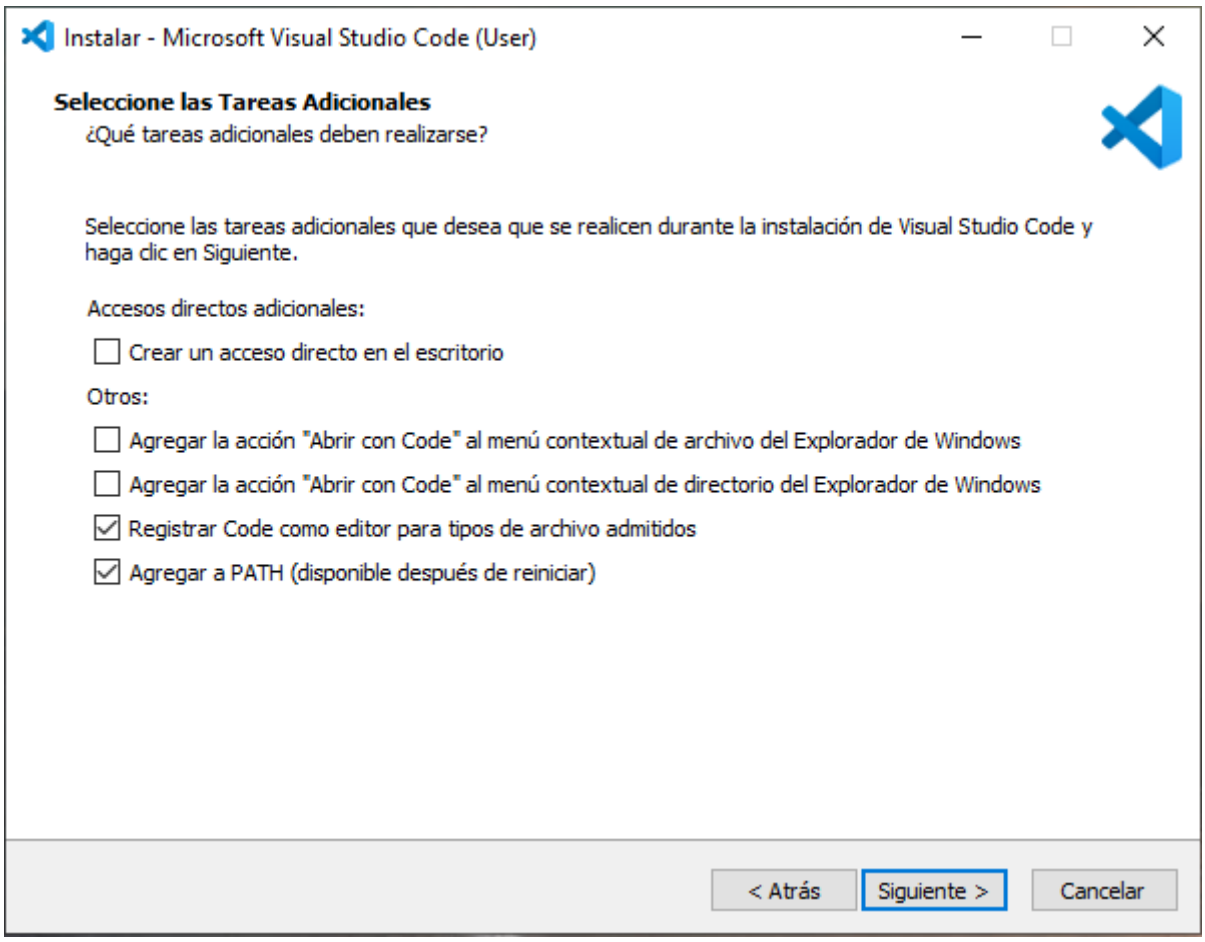

Ilustración 111 – Instalar Visual Studio Code

Para verificar que se ha instalado correctamente y confirmar la versión, se ejecutará el comando que aparece en la Ilustración 112 a continuación:

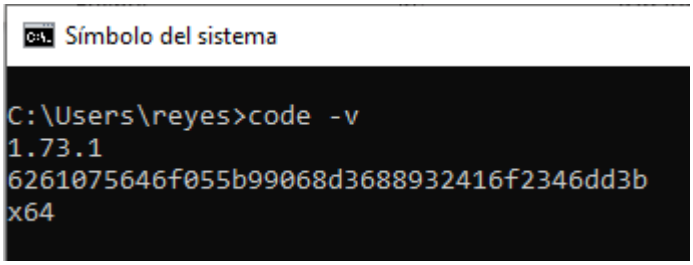

Ilustración 112 – Comprobar versión de Visual Studio Code

Tras instalarlo, ya se puede inicializar una aplicación con Expo [43]. Para ello, primero se debe instalar la interfaz de línea de comandos Expo CLI utilizando el comando que se muestra en la Ilustración 113:

Ilustración 113 – Instalación de Expo CLI

Una vez instalada, se puede crear un proyecto de React Native con Expo ejecutando el comando "expo init". En la consola aparecerá la opción de crear un proyecto JavaScript o TypeScript, y elegiremos el lenguaje JavaScript. En la Ilustración 114 se puede observar la ejecución del comando:

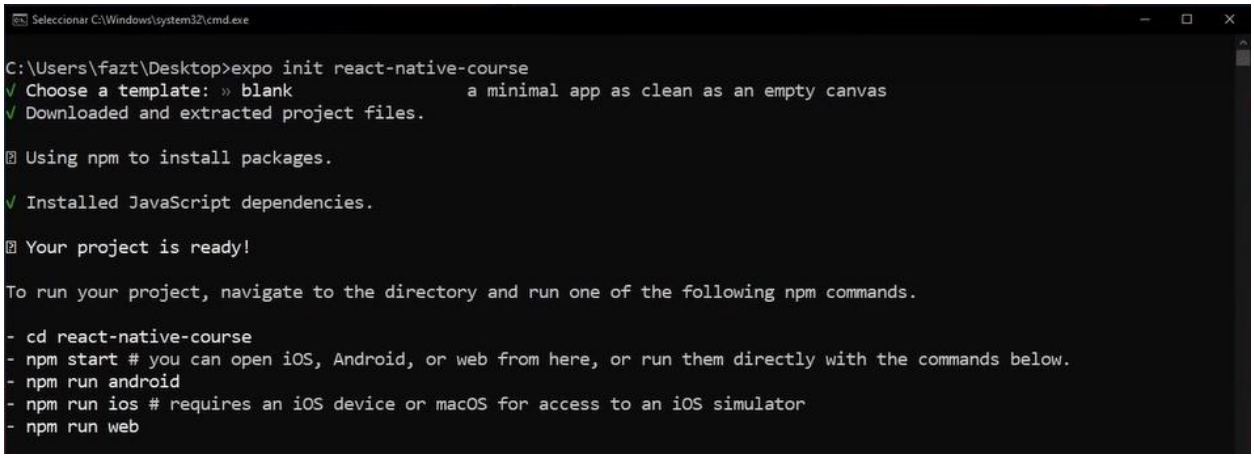

Ilustración 114 – Creación de un proyecto React Native en JavaScript utilizando Expo CLI

Tras crear el proyecto, se puede inicializar utilizando el comando "npm start" que se muestra en la Ilustración 115. Esto ejecuta el comando "expo start" mencionado en el capítulo 2 para iniciar el empaquetador de JavaScript "Metro" y el servidor de desarrollo de Expo.

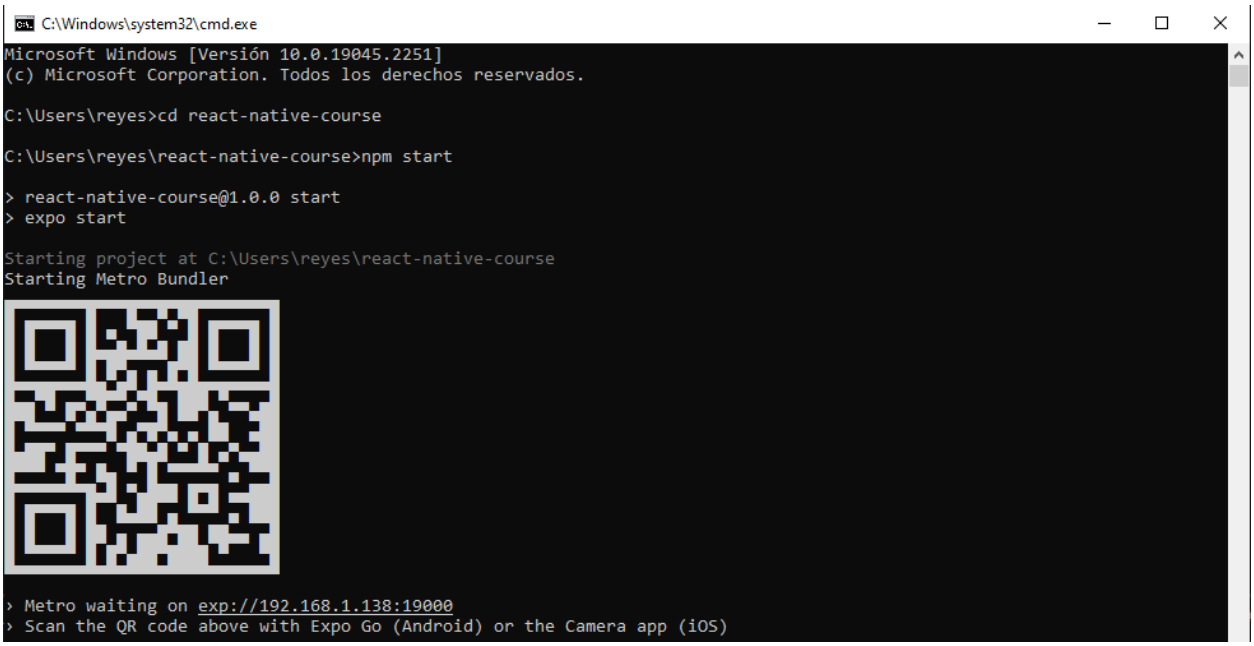

Ilustración 115 – Inicialización de la aplicación con npm

A.2 Despliegue del proyecto con GitHub

Para guardar el progreso del proyecto y realizar un control de versiones es recomendable subir la carpeta del proyecto a GitHub. Para ello, se deben seguir los siguientes pasos:

Primero, se debe instalar la aplicación GitHub Desktop desde la página web oficial [44] y presionar el botón de descargar el instalador para Windows que aparece en la Ilustración 116:

Ilustración 116 – Página de descarga de GitHub Desktop

Una vez descargado, se debe abrir el instalador y avanzar hasta que termine la instalación. De este modo, aparecerá una ventana en la cual se podrá iniciar sesión con una cuenta de GitHub o crear una nueva, tal y como se muestra en la Ilustración 117:

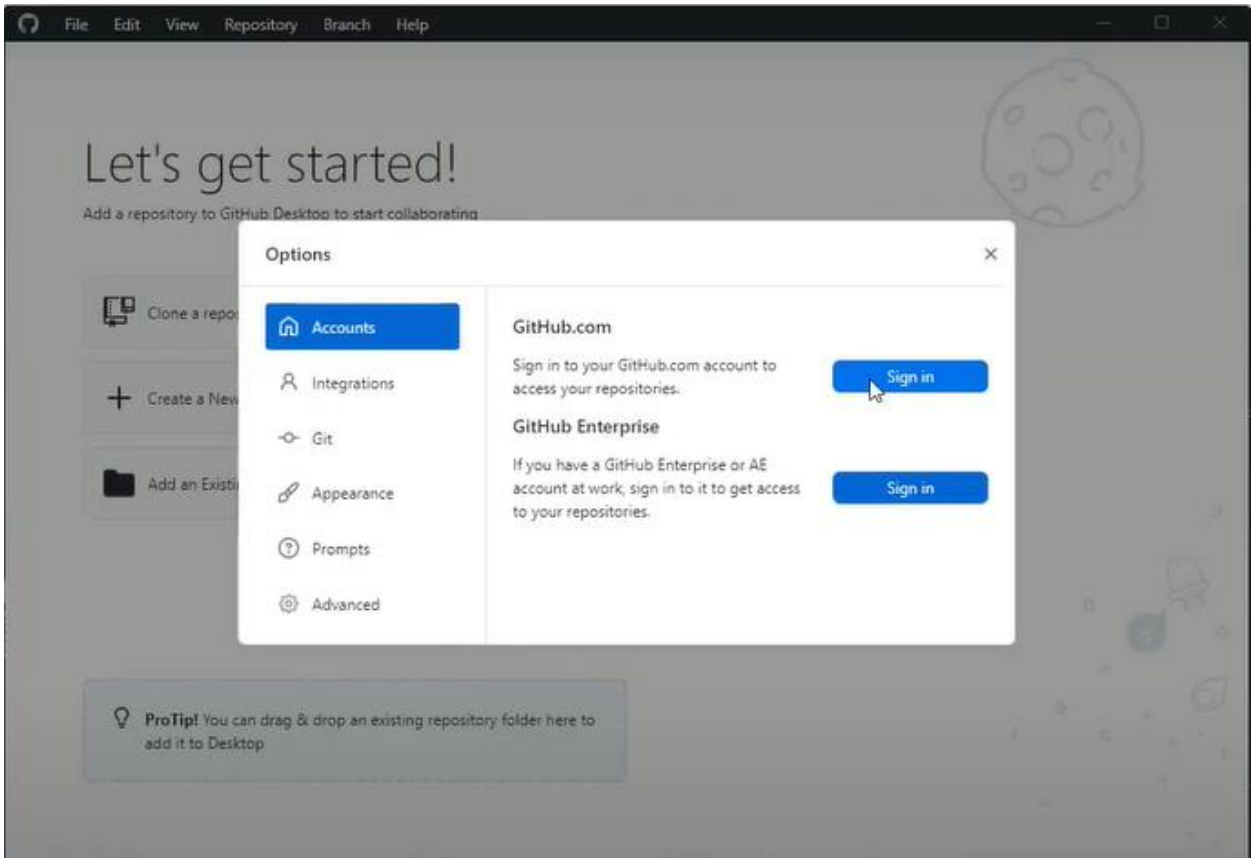

Ilustración 117 – Acceder a una cuenta de GitHub con GitHub Desktop

Una vez accedido a una cuenta de GitHub, se puede crear un nuevo repositorio, añadir un repositorio existente en local o clonar un repositorio de la nube. Para guardar los avances de la aplicación, se seleccionará la opción de crear un nuevo repositorio. Este se guardará en el sistema de archivos del ordenador. Para subir o publicar el repositorio en la nube, se debe seleccionar la opción de "Publicar repositorio" que aparece en GitHub Desktop. Una vez publicado, si se quiere guardar los cambios realizados en el proyecto, se pueden subir a la nube presionando el botón de "Push origin".

En el caso de querer obtener un repositorio alojado en la nube para poder realizar pruebas se deben seguir los pasos siguientes:

En primer lugar, se debe obtener la carpeta donde se encuentra el código de la aplicación, que está alojada en el repositorio de GitHub en la nube. Para almacenar la carpeta de forma local, se debe clonar el repositorio [45] desde la aplicación GitHub Desktop y así crear un repositorio local en el ordenador. En GitHub Desktop aparece una opción para "Clonar repositorio" y, tras pulsarla, aparecerá la ventana de la Ilustración 107, donde se podrá seleccionar el repositorio deseado o introducir la URL para obtener el repositorio de forma manual. Además, se podrá elegir la carpeta donde se almacenará de forma local.

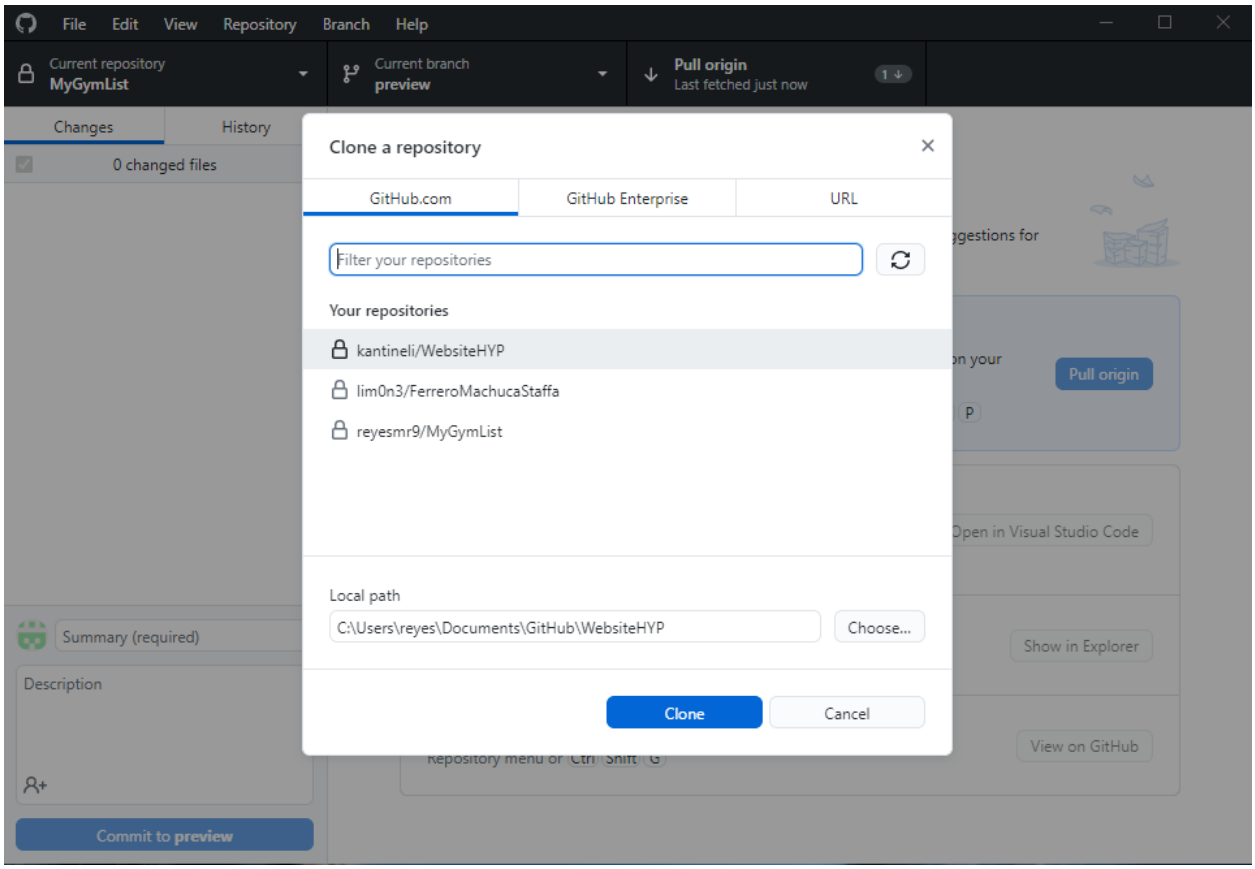

Ilustración 118 – Clonar repositorio con GitHub Desktop

ANEXO B: CÓDIGO DE LAS FUNCIONES PARA ENVIAR UNA LISTA

En este anexo se detallarán las funciones de *Lista.js* que permiten enviar una lista. "enviarLista", "comprobarLista" y "getLista" son funciones asíncronas, y sus definiciones son las siguientes:

Lista.js

"enviarLista" comprueba si se puede compartir la lista y, en el caso de que no se pueda, salta una alerta. En el caso de que se pueda compartir, se llama a la función "comprobarLista".

```
  const enviarLista = async () => {
    // Comprobamos que se pueda compartir la lista
    if (!(await Sharing.isAvailableAsync())) {
      Alert.alert('No se puede compartir la lista');
      return;
    }
    // Llamamos a la función que comprueba los permisos para compartir la lista
    comprobarLista();
  }
```

La función "comprobarLista", si el dispositivo es Android, comprueba los permisos para el acceso al almacenamiento interno y, si se han otorgado los permisos requeridos, llama a la función "getLista".

```
  const comprobarLista = async () => {
    // Si el sistema operativo es iOS, obtenemos la lista
    if (Platform.OS === 'ios') {
      getLista();
    } else {
        try{
          // Si el sistema operativo es Android, comprobamos los permisos
          const writegranted = await PermissionsAndroid.check(
            PermissionsAndroid.PERMISSIONS.WRITE_EXTERNAL_STORAGE,
          );
          if (writegranted) {
            try {
              // Si existen permisos para escribir en el almacenamiento externo,
obtenemos la lista
                getLista();
            } catch (error) {
              console.log("No tiene permisos para escribir en el almacenamiento
externo", error);
            }
          }
          else {
            const granted = await PermissionsAndroid.request(
```

```
              PermissionsAndroid.PERMISSIONS.WRITE_EXTERNAL_STORAGE,
              {
                title: 'Permiso de almacenamiento requerido',
                message:
                  'La aplicación necesita acceso al almacenamiento para enviar
la lista',
                buttonNegative: "Cancelar",
                buttonPositive: "OK"
              }
            );
            if (granted === PermissionsAndroid.RESULTS.GRANTED) {
              try {
                // Si se puede escribir en el almacenamiento externo, obtener
permisos para acceder
                // a la galería del dispositivo
                const permission = await MediaLibrary.getPermissionsAsync();

                if (permission.granted) {
                  // Obtenemos la lista si se han otorgado los permisos para
acceder a la galería
                  getLista();
                }
                if (!permission.granted && permission.canAskAgain) {
                  const { status, canAskAgain } = await
MediaLibrary.requestPermissionsAsync();

                  if (status === 'denied' && canAskAgain) {
                    Alert.alert('El usuario debe aceptar los permisos para
enviar la lista');
                  }
                  if (status === 'granted'){
                    getLista();
                  }
                  if (status === 'denied' && !canAskAgain) {
                    Alert.alert("Se ha denegado el permiso y no se puede volver
a preguntar");
                  }
                }
                if (!permission.canAskAgain && !permission.granted) {
                  Alert.alert("Se ha denegado el permiso y no se puede volver a
preguntar");
                }
              } catch (e) {
                console.log(e);
              }
          }
          else if (granted === PermissionsAndroid.RESULTS.DENIED){
            // Si el permiso ha sido denegado, se muestra una alerta de error
```

```
            Alert.alert('Error, no se han aceptado los permisos de
almacenamiento');
          }
          else {
            Alert.alert('Error, no se han aceptado los permisos de
almacenamiento');
          }
        }
      }
        catch (err) {
          console.log(err);
        }
      }
  }
```

La función “getLista”, obtiene la lista de la base de datos local, crea la uri de la lista y la envía utilizando módulo “Sharing”.

```
  const getLista = async () => {
    try{
      var lista = {};

      // Obtenemos las listas de la base de datos local
      const valorListas = await AsyncStorage.getItem("LISTAS");
      const l = valorListas ? JSON.parse(valorListas) : [];

      // Obtenemos la lista cuyo id coincide con la lista actual
      for (let i=0; i < l.length; i++){
        if((JSON.parse(l[i])).idlista == listaInicial.idlista){
          lista.imagen = (JSON.parse(l[i])).imagen;
          lista.titulo = (JSON.parse(l[i])).titulo;
          lista.videolista = (JSON.parse(l[i])).videolista;
          lista.email = (JSON.parse(l[i])).email;
          lista.idlista = (JSON.parse(l[i])).idlista;
          lista.fechacreacion = (JSON.parse(l[i])).fechacreacion;
          lista.listaRealizado = (JSON.parse(l[i])).listaRealizado;
          lista.imagenListaRealizado = (JSON.parse(l[i])).imagenListaRealizado;
          lista.historial = (JSON.parse(l[i])).historial;
          lista.fondo = (JSON.parse(l[i])).fondo;
          break;
        }
      }
      // Convertimos el objeto lista en una cadena en formato JSON
      var jsonLista = JSON.stringify(lista, null, 2);

      // Creamos la uri de la lista con formato .TXT
      fileUri = FileSystem.documentDirectory + "lista.txt";
```

```
      // Escribimos el contenido de la lista en la uri
      await FileSystem.writeAsStringAsync(fileUri, jsonLista, { encoding:
FileSystem.EncodingType.UTF8 });

      // Leemos el contenido de la lista de la uri
      file = await FileSystem.readAsStringAsync(fileUri, { encoding:
FileSystem.EncodingType.UTF8 });

      // Enviamos la uri de la lista llamando al módulo "Sharing"
      await Sharing.shareAsync(fileUri);

      } catch (error){
        console.log(error)
      }
  };
```

ANEXO C: CÓDIGO DE LAS FUNCIONES PARA REALIZAR UN ENTRENAMIENTO DE UNA LISTA

En este anexo se indicarán las funciones de *Entrenamiento.js* que permiten realizar el entrenamiento de una lista. "tiempoVideoDescanso", "tiempoVideoRealizado", "tiemposEjerciciosRealizados" y "guardarTiemposEjercicios" son funciones asíncronas, y sus definiciones son las siguientes:

Entrenamiento.js

"tiempoVideoDescanso" es la función que se llama cuando se pulsa el botón de "Comenzar ejercicio". Esta función establece el tiempo inicial del ejercicio y, si se han guardado previamente los tiempos de los ejercicios en la base de datos, calcula y almacena el tiempo de descanso del ejercicio anterior.

```
const tiempoVideoDescanso = async(item) => {
  try{
    // Establecemos el tiempo inicial del ejercicio
    setTiempoInicio(hours + ":" + minutes + ":" + seconds);

    // Obtenemos los tiempos de los ejercicios
    const valorTiempo = await AsyncStorage.getItem("TIEMPO");
    const t = valorTiempo ? JSON.parse(valorTiempo) : [];

    if(t.length==0){
      // Si no se han guardado los tiempos, iniciamos el cronómetro
      comenzarTimer();
      setFlagSaveTime(true);
      setFlagButton(true);

      var listaTiempo = {};
      listaTiempo.id = JSON.parse(item).id;
      listaTiempo.series = JSON.parse(item).series;
      listaTiempo.repeticiones = JSON.parse(item).repeticiones;
      listaTiempo.tiempoRealizacion = '';
      listaTiempo.tiempoDescanso = '';
      listaTiempo.video = JSON.parse(item).videos;

      // Convertimos el objeto lista en una cadena en formato JSON
      let jsonLista = JSON.stringify(listaTiempo);

      let newTiempo = [];

      // Guardamos los tiempos del ejercicio en la base de datos
      newTiempo.push(jsonLista);
      await AsyncStorage.setItem("TIEMPO", JSON.stringify(newTiempo));
      await AsyncStorage.getItem("TIEMPO").then((tiempo) => {
        console.log('El valor de los tiempos es: ', tiempo)
      });
```

```
      }
      else {
        if(tiempoFinal !== ""){

          let listArray = t;
          let jsonListArray = '';
          let listVideo = '';
          let numLista = '';

          // Si se han guardado los tiempos, calculamos el tiempo de descanso
del ejercicio anterior
          for (let i = 0; i < listaInicialT.videolista.length; i++){
            if(listaActualT == ''){
              if((JSON.parse(item).id) ==
(JSON.parse(listaInicialT.videolista[i]).id)){
                numLista = i;
                let tiempoEjFinal = tiempoFinal;
                let horasFinal = (tiempoEjFinal.split(":")[0]) * 3600;
                let minutosFinal = (tiempoEjFinal.split(":")[1]) * 60;
                let segundosFinal = tiempoEjFinal.split(":")[2] * 1;

                let segundosTotalActual = (hours * 3600) + (minutes * 60) +
seconds;
                let segundosTotalFinal = horasFinal + minutosFinal +
segundosFinal;

                let segundosDescansoTotal = Math.abs(segundosTotalActual -
segundosTotalFinal);

                let horasDescanso = Math.round(segundosDescansoTotal/60/60);
                let minutosparaDescanso = Math.round(segundosDescansoTotal/60);
                let minutosDescanso = Math.round(minutosparaDescanso % 60);
                let segundosDminutos = Math.round(segundosDescansoTotal % 60);

                listVideo = JSON.parse(listArray[i-1]);

                if(horasDescanso < 10){
                  horasDescanso = "0" + horasDescanso;
                }
                if(minutosDescanso < 10){
                  minutosDescanso = "0" + minutosDescanso;
                }
                if(segundosDminutos < 10){
                  segundosDminutos = "0" + segundosDminutos;
                }
                // Guardamos el tiempo de descanso en la lista
                listVideo.tiempoDescanso = horasDescanso + ":" + minutosDescanso
+ ":" + segundosDminutos;
```

```
                // Convertimos el objeto lista en una cadena en formato JSON
                jsonListArray = JSON.stringify(listVideo);
                break;
              }
            }
            else {
              if((JSON.parse(item).id) ==
(JSON.parse((JSON.parse(listaActualT)).videolista[i]).id)){
                numLista = i;
                let tiempoEjFinal = tiempoFinal;
                let horasFinal = (tiempoEjFinal.split(":")[0]) * 3600;
                let minutosFinal = tiempoEjFinal.split(":")[1] * 60;

                let segundosFinal = tiempoEjFinal.split(":")[2] * 1;
                let segundosTotalActual = (hours * 3600) + (minutes * 60) +
seconds;
                let segundosTotalFinal = horasFinal + minutosFinal +
segundosFinal;

                let segundosDescansoTotal = Math.abs(segundosTotalActual -
segundosTotalFinal);

                let horasDescanso = Math.round(segundosDescansoTotal/60/60);
                let minutosparaDescanso = Math.round(segundosDescansoTotal/60);
                let minutosDescanso = Math.round(minutosparaDescanso % 60);
                let segundosDminutos = Math.round(segundosDescansoTotal %
60);

                listVideo = JSON.parse(listArray[i-1]);

                if(horasDescanso < 10){
                  horasDescanso = "0" + horasDescanso;
                }
                if(minutosDescanso < 10){
                  minutosDescanso = "0" + minutosDescanso;
                }
                if(segundosDminutos < 10){
                  segundosDminutos = "0" + segundosDminutos;
                }
                listVideo.tiempoDescanso = horasDescanso + ":" + minutosDescanso
+ ":" + segundosDminutos;
                jsonListArray = JSON.stringify(listVideo);
                break;
                }
              }
          }

          var listaTiempo = {};
```

```
          listaTiempo.id = JSON.parse(jsonListArray).id;
          listaTiempo.series = JSON.parse(jsonListArray).series;
          listaTiempo.repeticiones = JSON.parse(jsonListArray).repeticiones;
          listaTiempo.tiempoRealizacion =
JSON.parse(jsonListArray).tiempoRealizacion;
          if(jsonListArray !== ''){
            listaTiempo.tiempoDescanso =
JSON.parse(jsonListArray).tiempoDescanso;
            listaTiempo.video = JSON.parse(jsonListArray).video;
          }
          else {
            listaTiempo.tiempoDescanso = '';
            listaTiempo.video = JSON.parse(item).videos;
          }

          let newTiempo5 = '';

          if(listVideo !== ''){
            if(numLista !== 1){
              // Si hay más de un ejercicio en la lista, se guardará el tiempo
de descanso del ejercicio anterior en la base de datos
              newTiempo5 = t.filter((lista) => lista !== listArray[numLista-1]);
            }
            else{
              // Si solo hay un ejercicio en la lista, se borrarán los tiempos
de la base de datos
              newTiempo5 = t;
              newTiempo5.length=0;
            }
            console.log('newTiempo5 es: ', newTiempo5)
          }

          var jsonLista = JSON.stringify(listaTiempo);

          newTiempo5.push(jsonLista);

          await AsyncStorage.setItem("TIEMPO", JSON.stringify(newTiempo5));

          await AsyncStorage.getItem("TIEMPO").then((tiempo) => {
            console.log('El valor de los tiempos es: ', tiempo)
          });
          setFlagSaveTime(true);
          setFlagButton(true);
        }
      }
    } catch(error){
        console.log(error)
      }
  }
```

"tiempoVideoRealizado" es la función que se llama cuando se pulsa el botón de "Siguiente ejercicio". Esta función establece el tiempo final del ejercicio y, si se han guardado previamente los tiempos de los ejercicios en la base de datos, calcula el tiempo de realización del ejercicio actual. Si no se han guardado previamente, asigna el tiempo actual como tiempo de realización del ejercicio.

```
const tiempoVideoRealizado = async(item) => {
  try{
    // Guardamos el tiempo de realización final del ejercicio
    setTiempoFinal(hours + ":" + minutes + ":" + seconds);

    // Obtenemos los tiempos de la base de datos
    const valorTiempo = await AsyncStorage.getItem("TIEMPO");
    const t = valorTiempo ? JSON.parse(valorTiempo) : [];

    if(t.length==0){
      var listaTiempo = {};
      listaTiempo.id = JSON.parse(item).id;
      listaTiempo.series = JSON.parse(item).series;
      listaTiempo.repeticiones = JSON.parse(item).repeticiones;

      let horas = '';
      let minutos = '';
      let segundos = '';

      if(hours < 10){
        horas = "0" + hours;
      }
      else {
        horas = hours;
      }
      if(minutes < 10){
        minutos = "0" + minutes;
      }
      else {
        minutos = minutes;
      }
      if(seconds < 10){
        segundos = "0" + seconds;
      }
      else {
        segundos = seconds;
      }

      listaTiempo.tiempoRealizacion = horas + ":" + minutos + ":" + segundos;
      listaTiempo.tiempoDescanso = '';
      listaTiempo.video = JSON.parse(item).videos;

      let newTiempo3 = t;
```

```
        var jsonLista = JSON.stringify(listaTiempo);

        newTiempo3.push(jsonLista);
        // Si no se han guardado los tiempos en la base de datos, se guarda el
tiempo de realización del ejercicio
        await AsyncStorage.setItem("TIEMPO", JSON.stringify(newTiempo3));

        await AsyncStorage.getItem("TIEMPO").then((tiempo) => {
          console.log('El valor de los tiempos es: ', tiempo)
        });
          setFlagButton(false);
          // Si hay más de un ejercicio en la lista, se avanza al siguiente
ejercicio
          if(flagOneVideo === false){
            setFlagTiempoInicial(true);
          }
          setFlagTiempo(!flagTiempo);
      }

      else {
        if(tiempoInicio !== ""){
          let listArray = t;
          let listVideo = '';
          let videoArray = '';
          let numVideo = '';

          // Si se han guardado los tiempos en la base de datos, se calcula el
tiempo de realización del ejercicio
          for (let i = 0; i < listaInicialT.videolista.length; i++){
            if((JSON.parse(item).id) ==
(JSON.parse(listaInicialT.videolista[i]).id)){
              numVideo = i;
              let tiempoEjInicial = tiempoInicio;

              let horasInicial = (tiempoEjInicial.split(":")[0]) * 3600;
              let minutosInicial = tiempoEjInicial.split(":")[1] * 60;

              let segundosInicial = tiempoEjInicial.split(":")[2] * 1;

              let segundosTotalActual = (hours * 3600) + (minutes * 60) +
seconds;
              let segundosTotalInicial = horasInicial + minutosInicial +
segundosInicial;

              let segundosRealizacionTotal = Math.abs(segundosTotalActual -
segundosTotalInicial);

              listVideo = JSON.parse(listaInicialT.videolista[i]);
              videoArray = JSON.parse(listaInicialT.videolista[i]);
```

```
            let horasRealizacion = Math.round(segundosRealizacionTotal/60/60);
            let minutosparaRealizacion =
Math.round(segundosRealizacionTotal/60);
            let minutosRealizacion = Math.round(minutosparaRealizacion % 60);
            let segundosRminutos = Math.round(segundosRealizacionTotal % 60);

            if(horasRealizacion < 10){
              horasRealizacion = "0" + horasRealizacion;
            }
            if(minutosRealizacion < 10){
              minutosRealizacion = "0" + minutosRealizacion;
            }
            if(segundosRminutos < 10){
              segundosRminutos = "0" + segundosRminutos;
            }

            listVideo.tiempoRealizacion = horasRealizacion + ":" +
minutosRealizacion + ":" + segundosRminutos;
            break;
          }
        }

        var listaTiempo = {};
        listaTiempo.id = JSON.parse(item).id;
        listaTiempo.series = JSON.parse(item).series;
        listaTiempo.repeticiones = JSON.parse(item).repeticiones;
        if(listVideo !== ''){
          listaTiempo.tiempoRealizacion = listVideo.tiempoRealizacion;
        }
        else{
          listaTiempo.tiempoRealizacion = JSON.parse(item).tiempoRealizacion;
        }
        listaTiempo.tiempoDescanso = '';
        listaTiempo.video = JSON.parse(item).videos;

        let newTiempo4 = '';

        if(videoArray !== ''){
          newTiempo4 = t.filter((lista) => lista !== listArray[numVideo]);
        }

        var jsonLista = JSON.stringify(listaTiempo);

        newTiempo4.push(jsonLista);
        // Almacenamos el tiempo de realización del ejercicio
        await AsyncStorage.setItem("TIEMPO", JSON.stringify(newTiempo4));

        await AsyncStorage.getItem("TIEMPO").then((tiempo) => {
```

```
            console.log('El valor de los tiempos es: ', tiempo)
          });

          // Si hay más de un ejercicio en la lista, se avanza al siguiente
ejercicio
          setFlagButton(false);
          if(flagOneVideo === false){
            setFlagTiempoInicial(true);
          }
          setFlagTiempo(!flagTiempo);

        }
      }

    } catch(error){
      console.log(error)
    }
  }
```

"tiempoEjerciciosRealizados" es la función que se llama cuando se pulsa el botón de "Guardar ejercicios". Esta función establece el tiempo final del ejercicio y el tiempo total del entrenamiento. Si se han guardado previamente los tiempos de los ejercicios en la base de datos, calcula el tiempo de realización del ejercicio actual. Si no se han guardado previamente, asigna el tiempo actual como tiempo de realización del ejercicio.

```
  const tiempoEjerciciosRealizados = async(item) => {
    try{
      // Guardamos el tiempo final del ejercicio realizado y el tiempo total de
entrenamiento
      setTiempoFinal(hours + ":" + minutes + ":" + seconds);
      setTiempoTotal(hours + ":" + minutes + ":" + seconds);

      // Obtenemos los tiempos de la base de datos
      const valorTiempo = await AsyncStorage.getItem("TIEMPO");
      const t = valorTiempo ? JSON.parse(valorTiempo) : [];

      if(t.length==0){
        var listaTiempo = {};
        listaTiempo.id = JSON.parse(item).id;
        listaTiempo.series = JSON.parse(item).series;
        listaTiempo.repeticiones = JSON.parse(item).repeticiones;

        let horas = '';
        let minutos = '';
        let segundos = '';
        if(hours < 10){
          horas = "0" + hours;
        }
        else {
          horas = hours;
```

```
        }
        if(minutes < 10){
          minutos = "0" + minutes;
        }
        else {
          minutos = minutes;
        }
        if(seconds < 10){
          segundos = "0" + seconds;
        }
        else {
          segundos = seconds;
        }

        listaTiempo.tiempoRealizacion = horas + ":" + minutos + ":" + segundos;
        listaTiempo.tiempoDescanso = '';
        listaTiempo.video = JSON.parse(item).videos;

        let newTiempo6 = t;

        var jsonLista = JSON.stringify(listaTiempo);

        newTiempo6.push(jsonLista);
        // Si no se han guardado los tiempos, almacenamos el tiempo de
realización del ejercicio en la base de datos
        await AsyncStorage.setItem("TIEMPO", JSON.stringify(newTiempo6));

        await AsyncStorage.getItem("TIEMPO").then((tiempo) => {
          console.log('El valor de los tiempos es: ', tiempo)
        });

        setFlagButton(false);

        if(flagOneVideo === false){
            setFlagTiempoInicial(true);
        }
        setFlagTiempo(!flagTiempo);
      }

      else {
        if(tiempoInicio !== ""){
          let listVideo = '';
          let videoArray = '';

          // Si se han guardado los tiempos, calculamos el tiempo de realización
del ejercicio
          for (let i = 0; i < listaInicialT.videolista.length; i++){
            if((JSON.parse(item).id) ==
(JSON.parse(listaInicialT.videolista[i]).id)){
```

```
              let tiempoEjInicial = tiempoInicio;
              let horasInicial = (tiempoEjInicial.split(":")[0]) * 3600;
              let minutosInicial = tiempoEjInicial.split(":")[1] * 60;

              let segundosInicial = tiempoEjInicial.split(":")[2] * 1;

              let segundosTotalActual = (hours * 3600) + (minutes * 60) +
seconds;
              let segundosTotalInicial = horasInicial + minutosInicial +
segundosInicial;

              let segundosRealizacionTotal = Math.abs(segundosTotalActual -
segundosTotalInicial);

              listVideo = JSON.parse(listaInicialT.videolista[i]);
              videoArray = JSON.parse(listaInicialT.videolista[i]);

              let horasRealizacion = Math.round(segundosRealizacionTotal/60/60);
              let minutosparaRealizacion =
Math.round(segundosRealizacionTotal/60);
              let minutosRealizacion = Math.round(minutosparaRealizacion % 60);
              let segundosRminutos = Math.round(segundosRealizacionTotal % 60);

              if(horasRealizacion < 10){
                horasRealizacion = "0" + horasRealizacion;
              }
              if(minutosRealizacion < 10){
                minutosRealizacion = "0" + minutosRealizacion;
              }
              if(segundosRminutos < 10){
                segundosRminutos = "0" + segundosRminutos;
              }

              listVideo.tiempoRealizacion = horasRealizacion + ":" +
minutosRealizacion + ":" + segundosRminutos;
              break;
            }
          }

          var listaTiempo = {};
          listaTiempo.id = JSON.parse(item).id;
          listaTiempo.series = JSON.parse(item).series;
          listaTiempo.repeticiones = JSON.parse(item).repeticiones;
          if(listVideo !== ''){
            listaTiempo.tiempoRealizacion = listVideo.tiempoRealizacion;
          }
          else{
            listaTiempo.tiempoRealizacion = '';
          }
```

```
          listaTiempo.tiempoDescanso = '';
          listaTiempo.video = JSON.parse(item).videos;

          let newTiempo1 = '';

          if(videoArray !== ''){
            newTiempo1 = t.filter((lista) => (JSON.parse(lista).id !==
videoArray.id));
          }

          var jsonLista = JSON.stringify(listaTiempo);

          newTiempo1.push(jsonLista);

          // Almacenamos el tiempo de realización del ejercicio en la base de
datos
          await AsyncStorage.setItem("TIEMPO", JSON.stringify(newTiempo1));

          await AsyncStorage.getItem("TIEMPO").then((tiempo) => {
            console.log('El valor de los tiempos es: ', tiempo)
          });

          setFlagButton(false);
          if(flagOneVideo === false){
            setFlagTiempoInicial(true);
          }
          setFlagTiempo(!flagTiempo);

        }
      }
    } catch(error){
      console.log(error)
    }
  }
```

"useEffect" es el Hook que se ejecuta cuando cambia el valor de la variable de estado "flagTiempo". Si se ha pulsado el botón Siguiente, y no se ha llegado al final de la lista, avanza al próximo ejercicio. Si se pulsa el botón "Guardar ejercicio", llama a la función "guardarTiemposEjercicios".

```
    useEffect(() => {
        if(isMounted===true){
        // Si se ha pulsado el botón Siguiente, se avanza al próximo ejercicio
        if(flagOneVideo === false && flagTiempoInicial === true){
          pulsarSiguiente();
          // Si se ha llegado al último ejercicio, se actualiza la lista
          if(currentSectionIndex >= ((listaActualT !== '') ?
          (JSON.parse(listaActualT).videolista.length-2) :
(listaInicialT.videolista.length-2))) {
```

```
            setFlagButtonGuardar(true);
          }
        }
        // Si se ha pulsado el botón Guardar, guardamos el tiempo de realización
y descanso de los ejercicios
        if(flagTiempoPress === true){
          guardarTiemposEjercicios();
        }
        }
        return () => {
          isMounted = false;
        };

      }, [flagTiempo]);
```

"guardarTiemposEjercicios" es la función que se llama si se ha pulsado el botón de "Guardar ejercicios". Esta función guarda el tiempo de realización y de descanso de los ejercicios en el campo "tiemposEjercicios" del historial de entrenamiento actual. Además, rellena el resto de los campos del historial de entrenamiento y lo almacena en el campo "historial" de la lista. Finalmente, almacena la lista en la base de datos local.

```
  const guardarTiemposEjercicios = async() => {
    try{
      // Obtenemos las listas y los tiempos de la base de datos
      const valorListas = await AsyncStorage.getItem("LISTAS");
      const l = valorListas ? JSON.parse(valorListas) : [];

      const valorTiempo = await AsyncStorage.getItem("TIEMPO");
      const t = valorTiempo ? JSON.parse(valorTiempo) : [];

      // Obtenemos la fecha actual
      var dia = new Date().getDate();
      var mes = new Date().getMonth() + 1;
      var año = new Date().getFullYear();
      var fecha = dia + '-' + mes + '-' + año;

      let listaHist = '';
      var listaHistorial = {};
      listaHistorial.idHistorial = (Math.round(Math.random() *
1000)).toString();

      if (listaActualT !== '' && listaActualT !== undefined) {
        listaHist = JSON.parse(listaActualT);
      }
      else {
        listaHist = listaInicialT;
      }

      // Comprobamos que los id de los historiales de entrenamiento sean únicos
en la lista
```

```
      for(let i=0; i<l.length; i++){
        if(listaHist.idlista === (JSON.parse(l[i]).idlista)) {
          for(let j=0; j<((listaHist.historial).length); j++){
            if((listaHist.historial)[j].idHistorial ===
listaHistorial.idHistorial){
              listaHistorial.idHistorial = (Math.round(Math.random() *
1000)).toString();
              break;
            }
          }
          break;
        }
      }

      // Guardamos la fecha de realización del entrenamiento en el historial
      listaHistorial.fechaRealizado = fecha;

      let tiempoT = tiempoTotal;
      let horasTotal = '';
      let minutosTotal = '';
      let segundosTotal = '';
      if(tiempoT.split(":")[0] < 10){
        horasTotal = "0" + tiempoT.split(":")[0];
      }
      else{
        horasTotal = tiempoT.split(":")[0];
      }
      if(tiempoT.split(":")[1] < 10){
        minutosTotal = "0" + tiempoT.split(":")[1];
      }
      else{
        minutosTotal = tiempoT.split(":")[1];
      }
      if(tiempoT.split(":")[2] < 10){
        segundosTotal = "0" + tiempoT.split(":")[2];
      }
      else{
        segundosTotal = tiempoT.split(":")[2];
      }

      listaHistorial.tiempoTotal = horasTotal + ":" + minutosTotal + ":" +
segundosTotal;

      // Guardamos los tiempos de la base de datos en el historial de
entrenamiento
      listaHistorial.tiemposEjercicios = t;

      if(t.length !== 0){
        let jsonListTiempo = null;
```

```
        let listFilterVideos = t;

        if (listaActualT !== '' && listaActualT !== undefined) {
          jsonListTiempo = JSON.parse(listaActualT);
        }
        else {
          jsonListTiempo = listaInicialT;
        }

        jsonListTiempo.videolista = listFilterVideos;

        let newLists = '';
        let listaH = null;

        for (let i = 0; i < l.length; i++){
          if(jsonListTiempo.idlista === (JSON.parse(l[i]).idlista)) {

            if (listaActualT !== '' && listaActualT !== undefined)
{
              listaH = JSON.parse(listaActualT);
            }
            else {
              listaH = listaInicialT;
            }

            let arrayListaH = [];

            // Si el historial está vacío inicialmente, se crea una matriz vacía
            if(JSON.parse(l[i]).historial == ''){
              arrayListaH = [];
              console.log('el historial está inicialmente vacío')
            }
            // Si el historial no está vacío, se añade a la variable arrayListaH
            else{
              arrayListaH = JSON.parse(l[i]).historial;
              console.log('existe ya un historial')
            }

            // Guardamos el historial de entrenamiento en el historial de la
lista
            arrayListaH.push(JSON.stringify(listaHistorial));

            listaH.historial = arrayListaH;

            break;
          }
        }
        setFlagSaveTime(true);
        setFlagOneVideo(false);
```

```
        // Filtramos las listas cuyo id sea diferente al de la lista actual
        newLists = l.filter((lista) => JSON.parse(lista).idlista !==
JSON.parse(listPlay).idlista);

        // Añadimos la lista actual con el historial actualizado en la lista de
listas
        newLists.push(JSON.stringify(listaH));

        const vid = AsyncStorage.getItem("TIEMPO");
        const varray = [vid];
        var f=varray;
        f.length=0;

        // Almacenamos una matriz vacía en los tiempos de la base de datos para
eliminarlos
        await AsyncStorage.setItem("TIEMPO", JSON.stringify(f));

        // Almacenamos las listas en la base de datos
        await AsyncStorage.setItem("LISTAS", JSON.stringify(newLists));

        let listaFinal = '';

        await AsyncStorage.getItem("LISTAS").then((listas) => {
          setListas(JSON.parse(listas));

          listaFinal = JSON.stringify(listaH);
          setFlagTiempoPress(false);
          let listaf = listT;
          listaf.length=0;
          setListT(listaf);

          // Reiniciamos el cronómetro
          setCurrentSectionIndex(0);
          if(customInterval){
            clearInterval(customInterval);
            setSeconds(0);
            setMinutes(0);
            setHours(0);
          }

          // Navegamos a la pantalla Lista
          navigation.reset({
            index: 0,
            routes: [
              {
                name: 'Lista',
                params: { singleList: listPlay,
                          itemId: JSON.parse(listPlay).listaid,
```

```
                        },
                    },
                ],
            })
            navigation.navigate("Lista", {
              singleList: ((listaFinal !== '') ? listaFinal : listPlay),
              itemId: JSON.parse(listPlay).listaid,
            });
            isMounted=false;
          });

        }
        else {
          console.log('No se ha guardado el tiempo en ninguno de los vídeos')
        }

      } catch(error){
        console.log(error)
      }

    }
```

REFERENCIAS

[1] Gobierno de España, junio de 2021. *Encuesta de hábitos deportivos en España*. Recuperado de: https://www.culturaydeporte.gob.es/dam/jcr:07b62374-bfe9-4a65-9e7e-03a09c8778c3/encuesta-de-habitos-deportivos-2020.pdf. [Último acceso: 7/11/2022]

[2] Instituto Nacional de Estadística, 2020. *Encuesta Europea de Salud en España*. Recuperado de: https://www.ine.es/ss/Satellite?L=es_ES&c=INESeccion_C&cid=1259926457058&p=

1254735110672&pagename=ProductosYServicios%2FPYS

Layout&param1=PYSDetalle&param3=12599248228. [Último acceso: 7/11/2022]

[3] Organización Mozilla, 29 de noviembre de 2022. *JavaScript*. Recuperado de: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Último acceso: 9/11/2022]

[4] Organización Mozilla, 31 de octubre de 2022. *Functions*. Recuperado de: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions. [Último acceso: 9/11/2022]

[5] Organización Mozilla, 9 de septiembre de 2022. A*synchronous JavaScript*. Recuperado de: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous. [Último acceso: 9/11/2022]

[6] Organización Mozilla, 7 de noviembre de 2022. *Promise*. Recuperado de: https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Promise. [Último acceso: 9/11/2022]

[7] Organización Mozilla, 17 de noviembre de 2022. *async function*. Recuperado de: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function. [Último acceso: 9/11/2022]

[8] Facebook, 24 de febrero de 2020. *React Native*. Recuperado de: https://reactnative.dev/. [Último acceso: 10/11/2022]

[9] Facebook, 22 de noviembre de 2022. *React Native Core Contributor Summit 2022*. Recuperado de: https://reactnative.dev/blog. [Último acceso: 10/11/2022]

[10] Google, 1 de febrero de 2009. *Google Trends*. Recuperado de: https://trends.google.es/trends/?geo=ES. [Último acceso: 10/11/2022]

[11] Facebook, 5 de septiembre de 2022. *React Fundamentals*. Recuperado de: https://reactnative.dev/docs/intro-react. [Último acceso: 10/11/2022]

[12] Facebook, 25 de octubre de 2018. *Hooks API Reference*. Recuperado de: https://reactjs.org/docs/hooks-reference.html. [Último acceso: 10/11/2022]

[13] Facebook, 3 de septiembre de 2019. *useNavigation*. Recuperado de: https://reactnavigation.org/docs/use-navigation/. [Último acceso: 10/11/2022]

[14] Facebook, 3 de septiembre de 2019. *useFocusEffect*. Recuperado de: https://reactnavigation.org/docs/use-focus-effect/. [Último acceso: 10/11/2022]

[15] Facebook, 5 de septiembre de 2022. *AsyncStorage*. Recuperado de: https://reactnative.dev/docs/asyncstorage. [Último acceso: 10/11/2022]

[16] Brex, 17 de julio de 2014. *Expo*. Recuperado de: https://expo.dev/. [Último acceso: 11/11/2022]

[17] Brex, 3 de agosto de 2017. *API Reference*. Recuperado de: https://docs.expo.dev/versions/latest/. [Último acceso: 11/11/2022]

[18] Fundación OpenJS, 9 de octubre de 2009. *Node.js*. Recuperado de: https://nodejs.org/en/. [Último acceso: 11/11/2022]

[19] Facebook, 5 de diciembre de 2017. *Metro, The JavaScript bundler for React Native.* Recuperado de: https://facebook.github.io/metro/. [Último acceso: 11/11/2022]

[20] Brex, 24 de abril de 2020. *Expo CLI.* Recuperado de: https://docs.expo.dev/workflow/expo-cli/. [Último acceso: 11/11/2022]

[21] Brex, 22 de abril de 2020. *Workflows.* Recuperado de: https://docs.expo.dev/introduction/managed-vs-bare/. [Último acceso: 11/11/2022]

[22] State Software, 26 de enero de 2009. *Introducción a JSON.* Recuperado de: https://www.json.org/json-es.html. [Último acceso: 12/11/2022]

[23] Microsoft, 1 de enero de 2017. *Why did we build Visual Studio Code?.* Recuperado de: https://code.visualstudio.com/docs/editor/whyvscode. [Último acceso: 13/11/2022]

[24] Microsoft, 29 de agosto de 2022. *User Interface.* Recuperado de: https://code.visualstudio.com/docs/getstarted/userinterface. [Último acceso: 13/11/2022]

[25] Brex, 14 de septiembre de 2022. *Expo Go.* Recuperado de: https://docs.expo.dev/workflow/expo-go/. [Último acceso: 13/11/2022]

[26] Linux, 22 de abril de 2012. G*it*. Recuperado de: https://git-scm.com/. [Último acceso: 13/11/2022]

[27] Microsoft, 27 de agosto de 2020. *Getting started with GitHub Desktop*. Recuperado de: https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop. [Último acceso: 14/11/2022]

[28] Koushik Dutta, 25 de agosto de 2015. *Vysor - A window to your Phone*. Recuperado de: https://www.vysor.io/. [Último acceso: 14/11/2022]

[29] JGraph Ltd, 22 de diciembre de 2019. *Diagrams.net.* Recuperado de: https://www.diagrams.net/. [Último acceso: 14/11/2022]

[30] Visual Paradigm International Ltd, 17 de marzo de 2009. *Visual Paradigm.* Recuperado de: https://www.visual-paradigm.com/. [Último acceso: 20/11/2022]

[31] Canva, 16 de junio de 2015. *Pixabay.* Recuperado de: https://pixabay.com/es/. [Último acceso: 14/11/2022]

[32] Bruno Joseph, 26 de febrero de 2019. *Pexels.* Recuperado de: https://www.pexels.com/es-es/. [Último acceso: 14/11/2022]

[33] Brex, 27 de marzo de 2018. *ImagePicker*. Recuperado de: https://docs.expo.dev/versions/latest/sdk/imagepicker/. [Último acceso: 19/11/2022]

[34] Brex, 27 de febrero de 2018. *Sharing.* Recuperado de: https://docs.expo.dev/versions/v47.0.0/sdk/sharing/. [Último acceso: 19/11/2022]

[35] Brex, 29 de julio de 2021. *DocumentPicker.* Recuperado de: https://docs.expo.dev/versions/latest/sdk/document-picker/. [Último acceso: 20/11/2022]

[36] Brex, 22 de noviembre de 2022. *FileSystem.* Recuperado de: https://docs.expo.dev/versions/latest/sdk/filesystem/. [Último acceso: 20/11/2022]

[37] Facebook, 5 de septiembre de 2022. *Modal.* Recuperado de: https://reactnative.dev/docs/modal. [Último acceso: 28/11/2022]

[38] Brex, agosto de 2022. *Building Standalone Apps.* Recuperado de: https://docs.expo.dev/archive/classic-updates/building-standalone-apps/?redirected. [Último acceso: 01/12/2022]

[39] Brex, 11 de diciembre de 2020. *Creating your first build.* Recuperado de: https://docs.expo.dev/build/setup/. [Último acceso: 01/12/2022]

[40] Fundación OpenJS, 27 de julio de 2016. *Descargas*. Recuperado de: https://nodejs.org/es/download/. [Último acceso: 02/12/2022]

[41] npm Inc, 17 de febrero de 2014. *npm*. Recuperado de: https://www.npmjs.com/. [Último acceso: 02/12/2022]

[42] Microsoft, 29 de abril de 2015. *Download Visual Studio Code*. Recuperado de: https://code.visualstudio.com/download. [Último acceso: 02/12/2022]

[43] Brex, 10 de noviembre de 2022. *Create your first app*. Recuperado de: https://docs.expo.dev/tutorial/create-your-first-app/. [Último acceso: 03/12/2022]

[44] Microsoft, 21 de mayo de 2012. *GitHub Desktop*. Recuperado de: https://desktop.github.com/. [Último acceso: 03/12/2022]

[45] Microsoft, 17 de mayo de 2022. *Cloning a repository*. Recuperado de: https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository. [Último acceso: 03/12/2022]