Proyecto Fin de Carrera
Ingeniería de las Tecnologías de Telecomunicación

Aplicación móvil multiplataforma de autocontrol y seguimiento por parte del paciente de dolor lumbar

Autor: Manuel Mansilla Gil

Tutor: José Manuel Fornés Rumbao

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

Sevilla, 2021

Proyecto Fin de Carrera
Ingeniería de las Tecnologías de Telecomunicación

Aplicación móvil multiplataforma de autocontrol y seguimiento por parte del paciente de dolor lumbar

Autor:
Manuel Mansilla Gil

Tutor:
José Manuel Fornés Rumbao
Profesor titular

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

Proyecto Fin de Carrera: Aplicación móvil multiplataforma de autocontrol y seguimiento por parte del paciente de dolor lumbar

Autor: Manuel Mansilla Gil

Tutor: José Manuel Fornés Rumbao

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis amigos

Agradecimientos

Este trabajo simboliza el fin de una etapa de mi vida.

Sin duda llegar a este punto no ha sido una tarea fácil cuanto menos, pero ver que he sido capaz de lograrlo es una demostración a mí mismo de que mediante esfuerzo y constancia, cualquier meta que me proponga es posible. Mi paso por la Escuela Técnica Superior de Ingeniería de Sevilla ha moldeado la persona que soy hoy y, a pesar del duro trabajo, estoy seguro de que la echaré de menos.

Sin duda, mi agradecimiento más profundo va a mis padres, a mi hermano y a mi hermana. Sin ellos no hubiera sido posible. Sin su apoyo día sí y día también no hubiera sido posible. No han sido pocas veces las que me habéis escuchado quejarme. Gracias, por apoyarme en todas mis decisiones académicas, por entender que a veces las cosas me sobrepasan y necesito reducir el ritmo de todo.

Tampoco estaría hoy aquí sin mi otro pilar, mis amigos. Con vosotros he compartido lágrimas, pero sobre todo risas, muchísimas risas. Me llevo conmigo personas que tengo clarísimo que me acompañarán para siempre. Debo daros las gracias, no es fácil hacer que una clase o una práctica se vuelva algo que incluso esperas con ganas. Los días de nueve a nueve en la universidad se han hecho cortos a vuestro lado.

Realmente no hay palabras que expresen lo que ambos significáis para mí, y es que sin vosotros yo hoy no estaría aquí, aunque creo que eso lo sabéis.

Resumen

En la actualidad las tecnologías móviles se han convertido en una parte vital de nuestra vida cotidiana, pasando a ser una parte indispensable de nuestro día a día. El avance de estas tecnologías permite explorar nuevas metodologías a la hora de ofrecer servicios a los usuarios, de una manera más cómoda, rápida y eficaz que la que ofrecen los procedimientos tradicionales, además de brindar una experiencia adaptada a las necesidades específicas de cada individuo.

Ante este hecho, la capacidad de adaptarse al mayor número de plataformas diferentes hace atractiva la capacidad de poder generar un mismo producto para diferentes sistemas. Con este objetivo, el propósito de este proyecto ha sido convertir una aplicación desarrollada de manera nativa para dispositivos Android en una aplicación nativa multiplataforma.

Para ello, nos hemos basado en dos de las estructuras de desarrollo móvil multiplataforma más utilizadas a nivel global por los desarrolladores de software en los últimos años: React Native y Expo.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Hoy en día la sociedad tiende hacia la digitalización: el proceso de transformar procesos analógicos en digitales. Tareas que antes realizábamos sin conexión y en papel se han vuelto algo que a veces incluso parece arcaico. Esto se debe a que la gran mayoría de las personas ya han realizado el cambio hacia la digitalización. Los servicios no son una excepción de esta digitalización, en este momento un usuario desde su smartphone posee la capacidad de guardar sus archivos en línea, pedir la cena a domicilio, o incluso comprar un coche con un par de toques.

Ante este hecho, la capacidad de poder desplegar un servicio en el mayor número de plataformas posible se ha convertido en una necesidad. La existencia de numerosas plataformas hace que esto no sea una tarea simple. Las diferencias entre cada una de ellas complican poder entregar un producto uniforme.

De la solución a este problema nace el desarrollo multiplataforma nativo, aplicaciones que presentan una interfaz y rendimiento equiparables a las diseñadas específicamente para una plataforma en concreto.

1.1 Motivación

Uno de los aspectos a tener en cuenta a la hora de escoger este proyecto ha sido la posibilidad de introducirme en el mundo del desarrollo de aplicaciones móviles, una parte del sector software que me resulta muy interesante personalmente, ya que es cercana a todos nosotros. Todos utilizamos en nuestro día a día aplicaciones móviles de manera constante, y poder adquirir las habilidades para crearlas uno mismo me ha resultado muy satisfactorio.

Además, este proyecto ha conllevado afrontar un problema de diseño software en el que he podido poner en práctica numerosos de los conocimientos que se han impartido en el grado, principalmente centradas en el ámbito de la programación.

Por último, poder crear una aplicación que ofrece ayuda a una persona con una dolencia es una tarea que resulta muy grata. Me ha ofrecido la oportunidad de aportar un granito de arena, y solamente el hecho de que la aplicación pueda ser útil para alguien hace que todo el trabajo puesto en este proyecto merezca el esfuerzo.

1.2 Objetivos

El objetivo de este proyecto es la creación de una aplicación multiplataforma móvil nativa funcional que ofrece al usuario la posibilidad de mantener un seguimiento de dolores lumbares. Para ello se utilizará en una estructura cliente - servidor mediante el uso de peticiones REST, fundamentada en los frameworks React Native y Expo. Dichos frameworks utilizan el lenguaje de programación JSX, extensión de la sintaxis de JavaScript.

1.3 Estructura del trabajo fin de grado

La estructura de este trabajo se compone de los siguientes puntos, donde se describe de forma resumida su contenido:

- Capítulo 1: Introducción.

  Se trata del capítulo actual. Presenta brevemente la motivación de este trabajo, así como los objetivos que se esperan alcanzar con el mismo.

- Capítulo 2: Tecnologías y entorno de trabajo.

  Describe las herramientas usadas en el desarrollo software de la aplicación y desarrolla los conceptos más destacados de ellas.

- Capítulo 3: Aplicación móvil.

  Muestra la división establecida de los componentes desarrollados de la aplicación para su presentación, la estructura de archivos implementada en el proyecto, la descripción de como la aplicación interactúa con una base de datos, y finalmente entra en el funcionamiento los componentes de la aplicación uno a uno.

- Capítulo 4: Instalación de la aplicación.

  Describe como la aplicación creada en este trabajo puede ser accedida por un usuario.

2 TECNOLOGÍAS Y ENTORNO DE TRABAJO

En este capítulo trataremos los diferentes marcos de trabajos utilizados, explicando sus diferentes funcionalidades y la elección de los mismos. Además, describiremos el entorno de trabajo en el que hemos desarrollado el proyecto

2.1 React Native

React Native [1] es un framework JavaScript open-source (accesible al público) creado por Facebook, basado en React. Utiliza la sintaxis JSX [2], una extensión de JavasScript. Permite al desarrollador crear aplicaciones reales nativas para Android e iOS, en lugar de tener que desarrollar una aplicación web o híbrida, las cuales presentan rendimientos, capacidades y experiencias inferiores a las aplicaciones nativas.

Ilustración 1. Logo de React Native.

Las aplicaciones creadas con React Native funcionan de la misma forma que lo harían las aplicaciones nativas reales desarrolladas para un sistema específico usando el lenguaje nativo propio.

2.1.1 Funcionamiento

Primero vamos a explicar el término DOM. DOM, o Document Object Model, es un modelo o representación gráfica del documento de una aplicación web creado por el navegador, y sobre éste aplica los cambios en cada actualización. Actualizar dicho DOM puede ser muy costoso en términos de rendimiento, pues cuanto más compleja es una aplicación más elementos habrá que modificar generalmente.

React se basa en la existencia de un VirtualDOM. Se trata de un concepto de programación donde una representación "virtual" de la interfaz de usuario se mantiene en memoria y se sincroniza con el DOM "real". Es decir, cuando ocurre un cambio el VirtualDOM actúa como intermediario entre la aplicación y el DOM, interpretando dichos cambios y calculando la manera más eficiente de actualizar el DOM, renderizando la menor cantidad de cambios posibles.

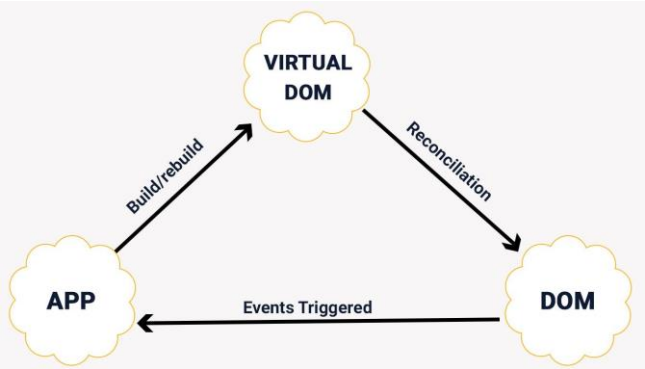

Ilustración 2. Relaciones entre la aplicación, el DOM y el VDOM.

Además de lo mencionado, React Native se apoya en tres elementos:

- Módulos nativos.
- Máquina virtual de JavaScript.
- React Native Bridge.

React Native ejecuta nativamente JavaScript en el terminal, no lo convierte. Esto es logrado gracias al Bridge de React Native, el cual permite la comunicación entre lo nativo y el hilo de JavaScript.

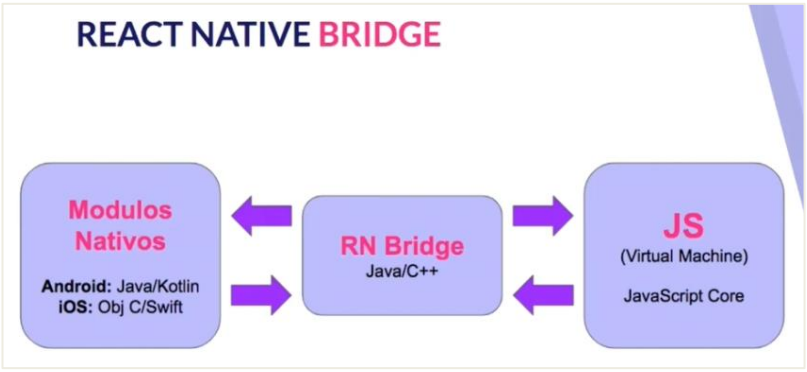

Ilustración 3. Funcionamiento de React Native Bridge.

2.1.2 JSX

Se trata de una extensión de la sintaxis de JavaScript, creada por Facebook, para el uso con su librería React [3]. Parecido al HTML, utiliza componentes que después se convertirán en componentes nativos para cada plataforma. JSX no es escribir HTML dentro de JavaScript, sino una forma de crear JavaScript más práctica.

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

Ilustración 4. Ejemplo de código JSX.

2.1.3 Componentes

React Native se basa también en los componentes. Conceptualmente, un componente es como una función de JavaScript. Aceptan entradas y devuelven elementos que describen la interfaz gráfica que debe aparecer por pantalla. Es decir, permiten separar la interfaz de usuario en partes independientes y reutilizables. Esto nos concede la posibilidad de pensar en cada parte de manera separada.

2.1.4 Hooks

Los Hooks de React son una de las últimas características añadidas a a la plataforma [4]. Resuelven una amplia variedad de problemas al permitirte usar funciones de React sin clases, evitando usar técnicas complejas de programación funcional o reactiva. A continuación, vamos a hablar de los dos más usados en este proyecto.

2.1.4.1 Hook de estado

Podemos definir un estado como un almacén personal de un dato. Mediante la llamada a useState, declaramos una "variable de estado". Es una forma de preservar algunos valores entre llamadas de la función, ya que las variables desaparecen cuando se sale de la función, pero las variables de estado son conservadas por React.

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

function Example() {
  // Declaración de una variable de estado que llamaremos "count"
  const [count, setCount] = useState(0);

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

Ilustración 5. Ejemplo de uso del Hook de estado.

2.1.4.2 Hook de efecto

El Hook de efecto en un componente nos permite ejecutar un código la primera vez que se monta el propio componente. También permite ejecutar el código cada vez que se actualice una variable si lo deseamos.

Este hook nos permite reemplazar los métodos del ciclo de vida de los componentes de clase típicos de React (ComponentDidMount, ComponentDidUpdate, ComponentWillUnmount) por un único método.

```
useEffect(() => {
  effect
  return () => {
    cleanup
  }
}, [input])
```

Ilustración 6. Ejemplo de la definición de un Hook de efecto.

2.2 Expo

El framework y plataforma Expo es un conjunto de herramientas y servicios construido en torno a React Native que ayudan a desarrollar y construir aplicaciones iOS, Android y web desde código JavaScript [5].

Ilustración 7. Logotipos de Expo

Encontramos dos componentes principales en Expo:

- Expo CLI: se trata de un conjunto de utilidades en modo comando, que permite crear, compilar y desplegar los proyectos de React Native. Además, permite el despliegue en terminales y simuladores de Android e iOS sin necesidad de los SDK nativos de dichas plataformas.
- La aplicación Expo: consiste en una aplicación móvil que permite abrir los proyectos mientras se está trabajando en ellos. Se sincroniza con un servidor web en la plataforma Expo para cargar y ejecutar el código desarrollado

2.3 Paquetes usados

React Native ofrece la posibilidad de utilizar paquetes desarrollados para su marco de trabajo. Procedemos a comentar los más usados en el desarrollo:

- React Native Elements: se trata de una implementación del "Material Design System" (un sistema adaptable de directrices, componentes y herramientas que se fundamentan en las mejores prácticas del diseño de interfaz de usuarios. Contiene un conjunto de componentes de propósito general de un estilo similar [6].
- React Navigation: provee una solución directa a la navegación entre los diferentes componentes que forman la aplicación. Por defecto está configurado para tener el look familiar de iOS y Android, incorporando las animaciones entre pantallas típicas de cada plataforma [7].
- Moment: librería de JavaScript que permite trabajar y manipular fechas de una manera rápida y eficaz [8].
- React Native Chart Kit: ofrece potentes herramientas para la construcción y dibujado de gráficas personalizadas [9].
- React Native YouTube iFrame: librería que otorga la capacidad de integrar vídeos de YouTube de manera embebida en nuestra aplicación, así como la posibilidad de diseñar controles personalizados para la reproducción de dichos vídeos [10].

2.4 Visual Studio Code

Como editor de código fuente para este proyecto hemos escogido Visual Studio Code [11]. Desarrollado por Microsoft, es una potente herramienta que permite incorpora soporte para la depuración, resaltado de sintaxis, finalización de código inteligente, además de la posibilidad de descargar plugins que amplían considerablemente las capacidades del editor.

Ilustración 8. Interfaz gráfica de Visual Studio Code.

3 APLICACIÓN MÓVIL

A continuación vamos a entrar en profundidad en el trabajo realizado para la creación de la aplicación. Nuestra aplicación recibe el nombre de "Appache", y su objetivo es permitir a un paciente que sufre de dolencia lumbar poder acceder cómodamente a un único punto en el que puede llevar un seguimiento propio de su dolencia al mismo tiempo que es asistido por un médico.

3.1 División de contenidos

Para comprender esta sección vamos a separar en diferentes partes la aplicación, donde agruparemos los componentes según su funcionalidad:

- Inicio de la aplicación: Incluye los componentes destinados al arranque del código y creación del sistema de navegación.
- Autenticación en el sistema: Incluye los componentes cuya función es realizar un inicio de sesión del usuario en el sistema. Una vez autenticados da paso al menú.
- Registro: Es la sección más compleja. Incluye los componentes encargados del registro por parte del usuario de los datos sobre su dolencia.
- Histórico: En esta sección se encuentran los componentes relacionados con la representación gráfica de los datos obtenidos en "Registro" para el usuario.
- Medicación: Dentro de ella tenemos los componentes relacionados con informar al paciente de su medicación y las instrucciones que debe seguir durante su uso.
- Ejercicio: Posee los componentes cuya función es presentar al usuario los diferentes ejercicios recomendados para tratar la dolencia lumbar.
- Consejos: Encontramos los componentes destinados a mostrar consejos sobre la dolencia lumbar al usuario.

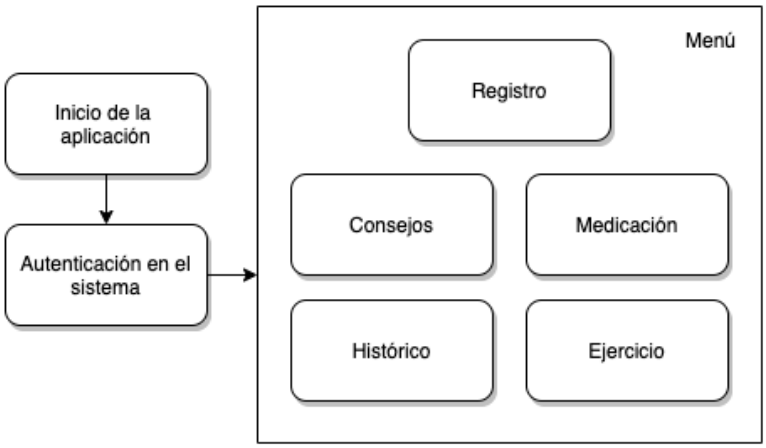

Ilustración 9. División de contenidos.

3.2 Estructura de archivos

3.2.1 Proyecto general de Expo

Cuando ejecutamos el comando de creación de un proyecto Expo desde el terminal se nos crearán una serie de archivos y carpetas plantilla.

La estructura de un proyecto Expo recién creado es la que podemos visualizar en la ilustración 9, y que consta de:

- Carpetas "expo" y ".expo-shared": Contienen archivos de configuración tanto del proyecto en sí como de Expo. Generalmente no es necesario modificarlos.
- Carpeta "assets": Esta carpeta es donde ubicaremos todo tipo de contenido gráfico que mantendremos de manera local en el dispositivo móvil. Algunos de archivos típicos de esta carpeta son imágenes, vídeos o fuentes de tipografía.
- Carpeta "node-modules": Contiene todas las dependencias y paquetes instalados para el funcionamiento del proyecto. Siempre que instalamos un nuevo paquete es añadido a esta carpeta. Posee un tamaño extremadamente grande.
- Archivo "git.ignore": Es usado por Git, un sistema de control de versiones que permite registrar cambios realizados en un conjunto de archivos a lo largo del tiempo, cuya finalidad es poder recuperar versiones específicas si fuera necesario. Este fichero en concreto sirve para informar a Git que archivos queremos que sean ignorados durante este proceso. La carpeta "node-modules" se encuentra añadida por defecto a este fichero.
- Archivo "App.js": Este archivo es el punto de entrada en nuestra aplicación. En él se escribe el código de cualquier aplicación Expo.
- Archivo app.json: Describe elementos como el nombre de la aplicación, su versión o el icono entre otros. Además, especifica también ciertas capacidades del proyecto, como por ejemplo el soporte para tablets en iOS.
- Archivo babel.config.json: Configura el funcionamiento de Expo con el compilador Babel.
- Archivo package.json: contiene los metadatos del proyecto, donde destacamos los scripts utilizados para el manejo del proyecto desde línea de comandos, y la lista de dependencias. En ella encontramos listadas todas las dependencias usadas en la aplicación junto con su versión. Gracias a este archivo podemos mantener siempre la carpeta "node-modules" de forma local, ya que al comprobar este

fichero otra persona que instale el proyecto en su ordenador podrá ver que dependencias necesita para el funcionamiento del mismo e instalarlas.

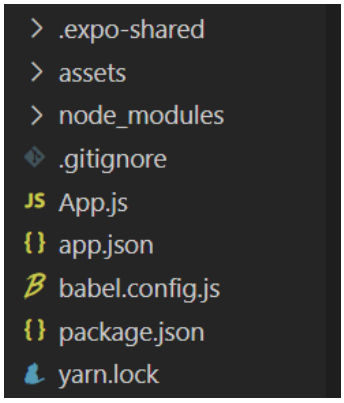

Ilustración 10. Estructura de archivos en la creación de un proyecto Expo.

3.2.2 Nuestro proyecto

Además de todos los archivos y carpetas mencionados en la subsección 3.2.1, hemos utilizado el esqueleto de carpetas mostrado en la ilustración 11.

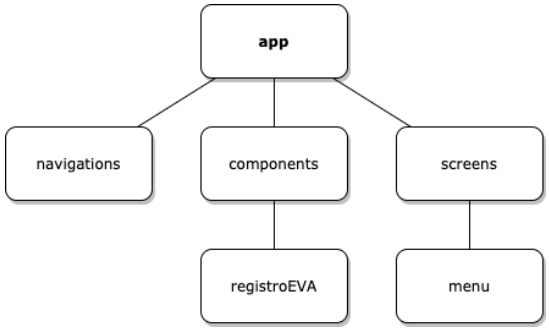

Ilustración 11. Estructura del directorio del contenido no creado por Expo.

La funcionalidad de esta estructura es la siguiente:

- Carpeta “app”: Es la carpeta padre de todos los archivos creados por nosotros para la aplicación, separándolos de todos los creados por Expo.
- Carpeta “navigations”: En ella situaremos todos los componentes relacionados con la navegación de la aplicación. Hablaremos en profundidad del sistema utilizado en el apartado 3.5.1.

Ilustración 12. Contenido del directorio “navigations”.

- Carpeta “screens”: Es el destino de todo componente que posea una función gráfica completa, es decir, necesite ocupar una pantalla. Encontramos los archivos relacionados con la inicialización de la aplicación, así como la carpeta “Menu”. En este directorio encontramos una carpeta por cada sección de la aplicación donde hemos situado los componentes empleados en la respectiva sección.

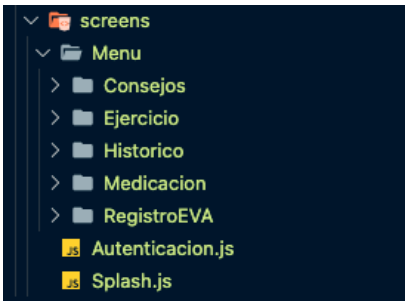

Ilustración 13. Contenido del directorio “screens”.

- Carpeta “components”: Aquí encontraremos componentes que son parte de otro componente mayor, en este caso de los componentes pertenecientes a la carpeta “screens”. Sirven para reducir la complejidad de la lógica de un componente en piezas separadas, haciendo su construcción una tarea más sencilla. En este proyecto sólo hemos utilizado esta técnica en componentes de la sección RegistroEVA, de ahí que sólo encontremos dentro un directorio con el mismo nombre.

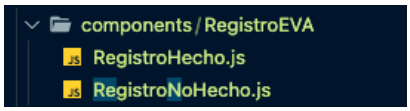

Ilustración 14. Contenido del directorio “components”.

3.3 Variables

Antes de pasar a la descripción de los componentes, es necesario hablar de las variables que marcarán el resultado de los algoritmos construidos en ellos. Las más importantes son las siguientes

- nivelDolor: Representa el dolor que el usuario siente el día que realiza el registro. Según su valor, definimos cuatro rangos que establecerán diferentes pautas a la hora de tratar al paciente:
    - Entre 0 y 3 (ambos incluidos) nos encontraremos en el nivel bajo. El paciente
    - Entre 4 y 5 (ambos incluidos) nos encontraremos en el nivel medio.
    - Entre 6 y 7 (ambos incluidos) nos encontraremos en el nivel alto.
    - Entre 8 y 10 (ambos incluidos) nos encontraremos en el nivel muy alto.
- días: Representa el número de días que el usuario ha presentado una dolencia baja. Es una variable del nivel bajo.
- ejercicio: Representa el número de días que el usuario ha realizado los ejercicios recomendados a lo largo de una semana. Es una variable del nivel bajo.
- nivelEjercicio: Representa el nivel actual de dificultad de los ejercicios propuestos al usuario. Es una variable del nivel bajo.
- pautaAnalgésica: A la hora de recetar un tipo de medicación, el médico suele recetar tres tipos de tratamientos. Una pauta basal (de ingesta diaria), una pauta analgésica (cuando el dolor aumenta) y una pauta de rescate (cuando el dolor es alto). Esta variable representa el número de días que el

usuario lleva con una pauta analgésica asignada. Es una variable del nivel medio.

- pautaRescate: Similar a pautaAnalgésica, representa el número de días que el usuario lleva con una pauta de rescate asignada. Es una variable del nivel ato.
- dolorAlto: Randera activada al entrar en un nivel de dolor muy alto.

3.4 Interacción con la base de datos

Es conveniente explicar la metodología empleada en el uso de la base de datos antes de continuar. Toda interacción con la base de datos se realiza mediante un servicio REST.

Un servicio REST (abreviatura de Representational State Transfer) es una interfaz que permite conectar varios sistemas basados en el protocolo HTTP y nos permite obtener datos en formatos específicos. En nuestro caso, el formato usado es JSON.

En nuestro proyecto, dichas peticiones presentan el siguiente formato de la ilustración 15. Vamos a proceder al análisis de este ejemplo.

```
var url = "http://80.211.228.126/";
url += "variables";
url += "/" + nid + "," + numVariable;

try {
  fetch(url, {
    method: "PUT",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      NID: nid,
      variable: numVariable,
      valor: valor,
    }),
  });
} catch (error) {
  console.error(error);
}
```

Ilustración 15. Ejemplo de petición REST en React Native.

En primer lugar, se define la dirección del servicio que nos interesa, en este ejemplo estamos actualizando el valor de una variable.

Dicha dirección es accedida mediante la función “fetch”, cuya traducción sería “ir a buscar”. Dicha función nos permite realizar una petición HTTP en la que podemos especificar los parámetros.

Mediante el parámetro “method” estableceremos si nuestra petición será de tipo GET, PUT o POST, que son los tipos de peticiones usadas en este proyecto. Mediante GET obtenemos un objeto JSON respuesta del servicio, mientras que con PUT o POST nosotros somos los que estamos enviando al servicio un objeto JSON, bien para que actualice algún valor (PUT) o lo añada directamente (POST).

En las peticiones donde enviamos un objeto al servicio, es necesario añadir dicho objeto al cuerpo de la petición. Dicho objeto debe enviarse en una cadena de texto JSON para que el servicio reconozca los datos.

En las peticiones inversas, donde pedimos un objeto, el proceso se hace al revés. Es decir, convertimos esa

cadena de texto JSON en un objeto para poder manipularlo y trabajar con él.

Las direcciones deben presentar el en el siguiente formato:

- http: :// dirección IP del servicio / nombre del servicio

Finalmente, podemos ver una tabla con los diferentes tipos de peticiones que se realizan en la aplicación en la tabla 1.

Tabla 1. Peticiones REST usadas en la aplicación.

Dirección	Tipo de petición	Funcionalidad
autenticacion/	GET	Obtener el identificador del dispositivo móvil.
variable/	GET	Obtener las variables del usuario.
pacientes/	GET	Obtener los datos de todos los pacientes.
consejos/	GET	Obtener todos los consejos.
ejercicio/	GET	Obtener todos los ejercicios.
historico/NID, Valor, Fecha	GET	Obtener el historial de registros de un usuario.
medicacion/	GET	Obtener todos los medicamentos.
mensajefor/	GET	Obtener todos los mensajes formativos.
mensajeinfor/	GET	Obtener todos los mensajes informativos.
recordatorios/	GET	Obtener todos los recordatorios.
autenticacions/+nid	PUT	Actualizar el identificador del dispositivo móvil.
variables/+nid, variable	PUT	Actualizar el valor de una variable.
historicos/	POST	Registra el nivel de dolor del usuario en el histórico.
rests/	PUT	Registra cuando se ha utilizado uno de los servicios REST.

Como aclaración, donde aparece un signo más es necesario que en la dirección a la que realizamos la petición REST lleve los valores de las variables que aparecen en la tabla.

3.5 Componentes

Aquí vamos a describir pieza a pieza los diferentes componentes por secciones de la aplicación, siguiendo el orden que el usuario encontrará al iniciar la aplicación.

Las capturas de pantalla mostradas han sido tomadas en los siguientes dispositivos:

- Para el sistema operativo móvil iOS: iPhone 12 Mini
- Para el sistema operativo móvil Android: Huawei P9 Plus

Por claridad, la mayoría de las pantallas que pondremos serán de iOS para no sobrecargar la memoria.

3.5.1 Inicio de la aplicación: App.js y Navigation.js

El componente “App.js”, como hemos comentado previamente, es el componente padre y el primero en ejecutarse al iniciar la aplicación. En él se sitúa todo el código de la aplicación. En nuestro caso, desde él cargaremos el componente encargado del manejo de la navegación.

Se trata de “Navigation.js”. Para entender su contenido, vamos a detenernos un momento y explicar cómo funciona la navegación mediante React Navigation.

Lo primero que debemos saber es que cualquier tipo de navegación debe ir envuelta en un contenedor, y dentro de él los componentes entre los que se quiere navegar, a los que se les llama pantalla. Este contenedor es que el que crea el objeto "navigation", el cual posee los métodos necesarios para el control de la navegación entre sus miembros.

Podemos encontrar diferentes tipos de contenedores, los cuales nos ofrecerán maneras alternativas de navegación. En este proyecto hemos utilizado dos clases de contenedores para la implementación de la navegación en toda la aplicación:

- Contenedor Stack: Un Stack de navegación es similar a una pila de pantallas. Al navegar entre los componentes del contenedor lo que se hace es mostrar la pantalla seleccionada, ocultado las demás detrás de ésta. Esto ofrece una continuidad entre los miembros del contenedor, al no existir una distinción aparente entre pantallas [12].
- Contenedor Tab: Es posiblemente uno de los estilos de navegación móvil más utilizados. Consiste en una barra de pestañas, en la que cada pestaña está formada por uno de los elementos pertenecientes al contenedor. En nuestro caso hemos incorporado al proyecto una barra de pestañas en la parte inferior de la pantalla [13].

Ambos contenedores proporcionan métodos diferentes de manejar el estilo de navegación que aportan, permitiendo alcanzar una navegación óptima a nuestras necesidades.

Una vez introducidos estos conceptos podemos introducir la composición formada por ambos contenedores para el sistema de navegación de la aplicación:

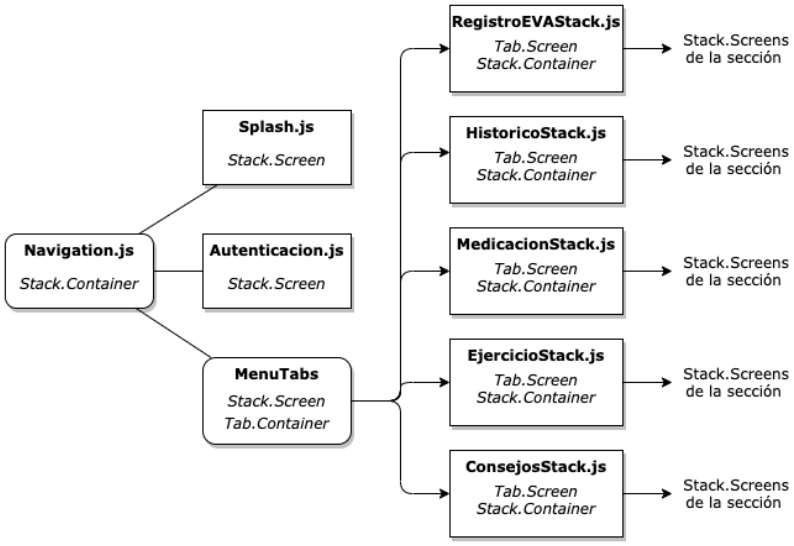

Ilustración 16. Diagrama de navegación.

El contenedor padre de la navegación es "Navigation.js". Se trata de un contenedor de tipo Stack, el cual dentro posee los componentes: "Splash.js", "Autentication.js" y "MenuTabs". Este último componente está definido dentro del propio componente "Navigation.js". A su vez, "MenuTabs" no sólo es una de las pantallas del Stack perteneciente a Navigation.js, sino que también es un contenedor de tipo Tab.

¿Qué conseguimos con esta distribución de los contenedores? El resultado conseguido es que la barra de pestañas no aparecerá a menos que entremos en el componente "MenuTabs". De esta forma conseguimos establecer dos partes de la aplicación de manera clara: una primera parte donde todavía no hemos accedido al menú en la que nos movemos de pantalla a pantalla dirigidos por la aplicación, y una segunda parte donde encontramos la barra de pestañas que permite navegar libremente al usuario entre sus diferentes componentes.

Prosiguiendo con la definición de la estructura de navegación, dentro de "MenuTabs" encontramos cinco pantallas, una por cada pestaña que hemos creado para el menú. Cada una de estas pantallas es al mismo tiempo un contenedor de tipo Stack. Así logramos el efecto inverso: dentro de una pestaña obtenemos una

navegación guiada por la aplicación, logrando así una sensación de unidad entre las pantallas que conforman cada pestaña.

Finalmente, dentro de cada uno de estos contenedores encontramos las pantallas finales que conforman el Stack de cada pestaña.

Todas las secciones están formadas por un único componente, exceptuando la pestaña Registro, que es la que posee la mayor complejidad y cuenta con once componentes pantalla dentro de ella. ¿Por qué utilizar contenedores entonces en el resto de las secciones? Al utilizar esta técnica, conseguimos hacer la modificación de una pestaña una tarea sencilla, ya que sólo habría que añadir o eliminar componentes pantalla dentro del contenedor de su pestaña. De esta forma si en un futuro hiciera falta realizar modificaciones a la navegación dentro de la pestaña la tarea sería trivial.

Una vez aclarada la navegación, el resultado es el siguiente: Al iniciar "App.js" se cargará el componente "Navigation.js", el cual tiene como pantalla inicial por defecto el componente "Splash.js"

3.5.2 Autenticación en el sistema: Splash.js y Autentication.js

Se trata de los componentes encargados del inicio de la aplicación y del acceso del usuario al menú respectivamente. Su función conjunta es la de comprobar si el dispositivo del usuario está registrado en la base de datos, mediante una comprobación en la base de datos. En ella se busca el identificador del dispositivo, que permiten, valga la redundancia, identificar inequívocamente el teléfono del paciente.

Si esta comprobación nos diera un resultado negativo, pasamos a solicitar la autenticación por parte del paciente. Esto se realiza solicitando al usuario que introduzca una serie de credenciales que su médico le ha facilitado previamente.

Una vez autenticados de manera correcta, se inserta en la base de datos el identificador del dispositivo. Así logramos que la próxima vez que el usuario acceda a la aplicación pase directamente desde la pantalla de Splash al menú, sin tener que autenticarse.

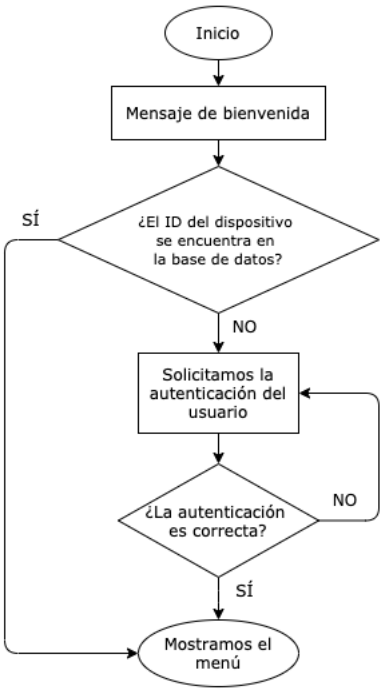

Ilustración 17. Inicio de la aplicación.

Las pantallas Splash son una técnica recurrente en el diseño de aplicaciones móviles. Cuando una aplicación es de un tamaño considerable puede necesitar cierto tiempo para cargar datos en el arranque. Debido a esto,

puede darse el caso de que mientras se cargan estos datos se muestren pantallas vacías o incompletas.

La pantalla Splash se encarga de solucionar esto: mientras se están cargando los datos muestra una imagen o mensaje (normalmente centrados). Tras la carga, da paso a la primera pantalla real de la aplicación. De esta forma el usuario es consciente de que la aplicación está arrancando.

Con la finalidad descrita, nuestro componente Splash muestra un mensaje de bienvenida acompañado de un icono que muestra "un saludo" mediante una animación en la que va aumentando la opacidad, proporcionando la sensación de que está apareciendo el mensaje.

Mientras, el componente accede a la base de datos y realiza la comprobación del identificador de dispositivo descrita anteriormente. Además, en caso de encontrarla extrae el número identificador del paciente (NID) y lo envía al componente MenuTabs, además de redirigirnos a él. Al pasarle dicha información al menú estamos consiguiendo que todas las secciones tengan acceso a ella, de esta forma nos ahorramos tener que volver a solicitarla.

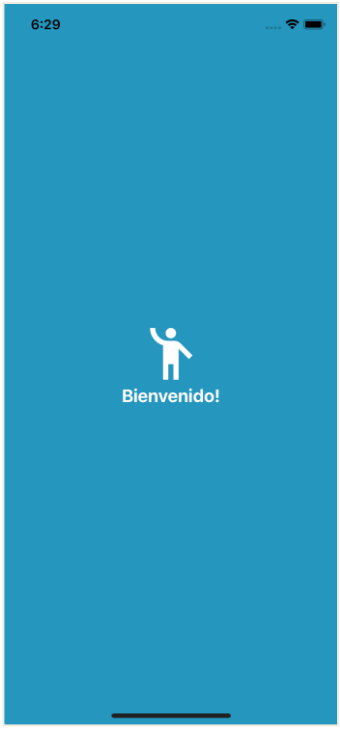

Ilustración 18. Splash.js en iOS.

En el caso de que la comprobación del identificador de dispositivo fuera negativa o errónea, seríamos redirigidos al componente "Autentication.js".

En ella se nos mostrará un mensaje solicitándonos que introduzcamos las credenciales en las tres entradas de texto que aparecen. Dichas credenciales por introducir son:

- El número identificador del paciente (NID).
- La clave de acceso. Este campo posee la posibilidad de ocultar el texto introducido o mostrarlo, mediante la pulsación del icono con forma de ojo que aparece en él. Cada vez que se pulse, el icono cambiará y se alternará entre un modo de texto seguro y uno inseguro.
- El número de teléfono del paciente.

Al pulsar el botón de "Validar datos", se comprobará que ninguno de los campos está vacío, y que él numero de teléfono consta de nueve dígitos, imprimiendo un mensaje de error si se dieran alguno de los dos casos.

Tras ello, realizará una petición de autenticación a la base de datos. Si el resultado fuera negativo o se produjera un fallo en la petición, se mostrará un mensaje de error en el que se volverán a pedir las credenciales por parte del usuario.

Si por el contrario la autenticación es correcta, al igual que en “Splash.js”, se extraerá el NID del paciente para ser mandado al componente “MenuTabs”, además de redirigirnos a él. Aquí aparecerá la barra de pestañas que nos permitirá navegar entre las diferentes secciones.

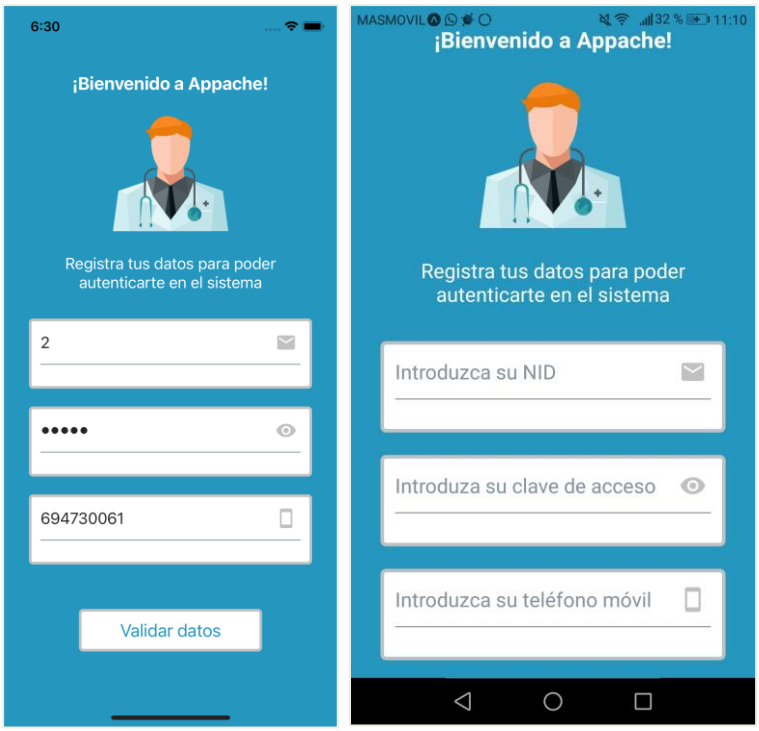

Ilustración 19. Autentication.js en iOS y Android.

3.5.3 Pestaña “Registro”

En esta subsección encontramos todos los componentes pantalla pertenecientes a “RegistroEVAStack.js”, los cuales detallaremos en profundidad.

3.5.3.1 RegistroEVA.js

Es el componente inicial del Stack, y posee dos componentes internos de los que hablaremos a continuación.

En su carga hace uso del hook de efecto para recuperar el nombre del paciente y para comprobar el historial del usuario en la base de datos. En esta comprobación, miraremos la fecha de la última entrada de la base de datos, así como el NID de usuario. Si la fecha coincide con la actual, significa que ya hemos realizado el registro hoy. Por el contrario, si la fecha encontrada no coincide con la actual el usuario todavía debe realizar el registro.

Basándonos en el resultado de esta comprobación, “RegistroEVA.js” mostrará uno de los siguientes componentes internos: “RegistroHecho.js” y “RegistroNoHecho.js”

En ambos casos, “RegistroEVA.js” muestra un mensaje de saludo personalizado con el nombre del paciente que obtuvimos en la carga del componente.

Ilustración 20. Funcionamiento RegistroEVA.js

Si la comprobación ha dado un resultado positivo, "RegistroHecho.js" informa al paciente de que sus datos ya han sido registrados ese día, y se le insta a navegar entre las diferentes pestañas para utilizar los diferentes servicios proporcionados por la aplicación.

Ilustración 21. RegistroEVA.js cargando RegistroHecho.js en iOS.

Por el contrario, si el resultado ha sido negativo, "RegistroNoHecho.js" imprime un mensaje al paciente recordando que todavía no ha realizado el registro. También se mostrará un botón con el texto "Continuar", el cual al ser pulsado hará uso de la navegación para redirigirnos a nuestro próximo componente, "NivelDolor.js".

Ilustración 22. RegistroEVA.js cargando RegistroNoHecho.js en iOS.

3.5.3.2 NivelDolor.js

Este componente se encarga de pedir al usuario que introduzca el nivel de dolencia lumbar que tiene en ese día. Para pedir este valor al usuario, se le muestra por pantalla una escala visual analógica (EVA).

Una escala EVA es una herramienta que ayuda a una persona a evaluar la intensidad de ciertas sensaciones y sentimientos. En este caso nuestra intención es evaluar el dolor que siente el paciente. Consiste en una línea recta donde el extremo izquierdo representa la ausencia de dolor, mientras que el extremo derecho representa el dolor máximo [14].

La escala puede ayudar al paciente a identificar de forma más precisa la intensidad de su dolencia. De esta forma al tener una medición cercana a la realidad del dolor que siente el usuario podemos refinar que tipo de necesidades hace falta proporcionarle.

Para introducir el valor, se le presenta por pantalla una entrada de texto. Esta entrada de texto posee una confirmación para comprobar que el dato introducido es un número entre cero y diez, mostrando un mensaje por pantalla de error en caso de que no se cumpla esta condición.

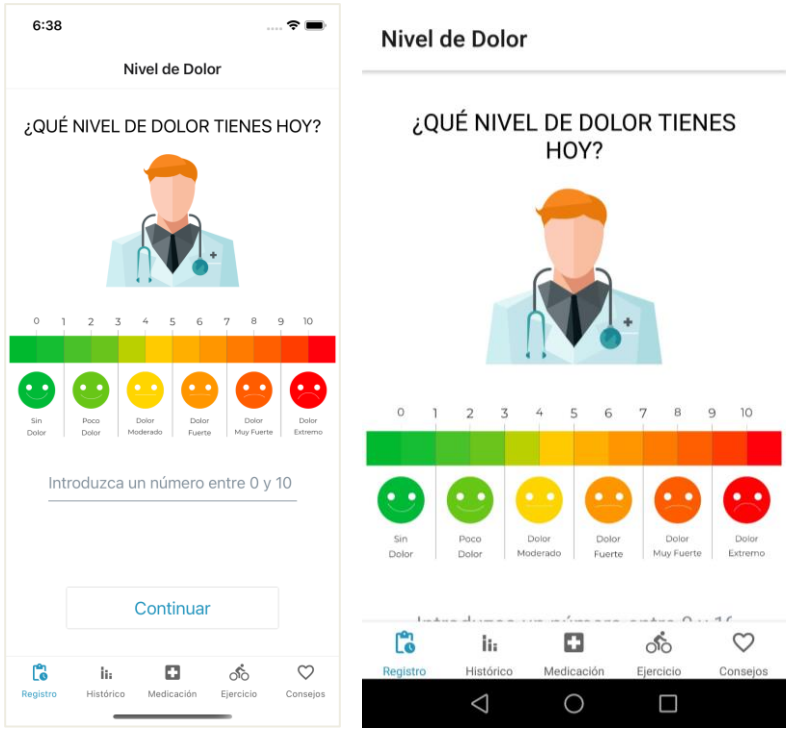

Ilustración 23. NivelDolor.js en iOS y Android.

Una vez introducido el nivel de dolor y comprobado de que es se trata de un dato válido, el componente ejecuta un algoritmo en el que pondrá a 0 las variables de los otros niveles (activando la bandera de nivel muy alto si hubiera sido el nivel introducido) y procede a navegar hacia el siguiente componente.

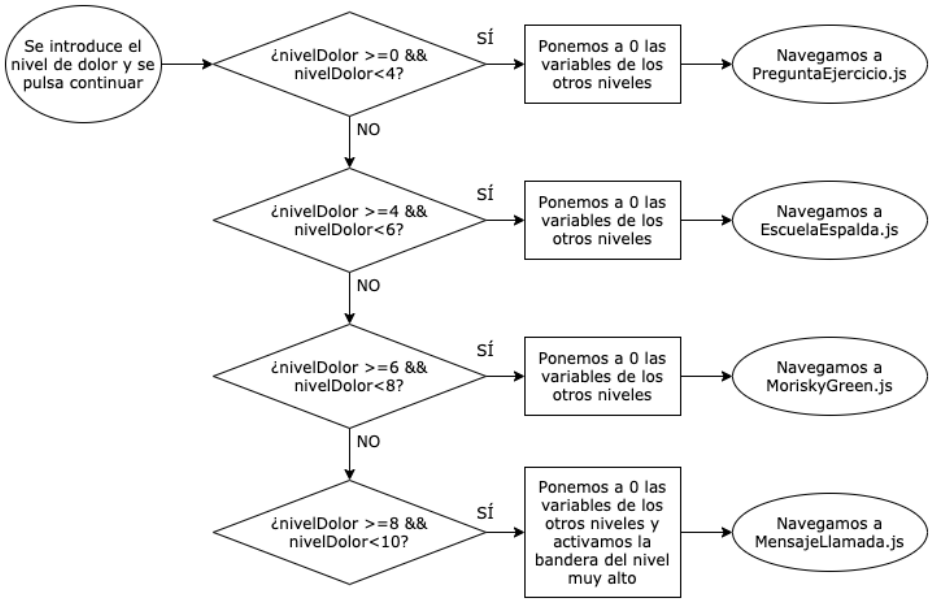

Ilustración 24. Diagrama de funcionamiento de NivelDolor.js

3.5.3.3 PreguntaEjercicio.js

Este componente aparece cuando el nivel de dolor introducido por el usuario pertenece al nivel bajo, y se encarga de comprobar si el usuario ha estado realizando los ejercicios recomendados. Para ello, mostrará dos iconos (uno afirmativo, uno negativo) en la pantalla. Al ser pulsados, aparece un texto de confirmación en el

que se le muestra al paciente la opción elegida, además de habilitar el botón “Continuar”, el cual hasta entonces permanece oculto.

Para el funcionamiento del algoritmo de este componente, en la carga se recuperan las variables involucradas en el nivel bajo, es decir, las variables día, ejercicio y nivelEjercicio.

Ilustración 25. PreguntaEjercicio.js en iOS.

Al pulsar el botón continuar, se llevará a cabo el siguiente algoritmo:

- Se aumentará la variable día en una unidad.
- Se aumentará la variable ejercicio en una unidad si el usuario ha respondido afirmativamente a la pregunta. En caso negativo su valor permanece igual.
- Se comprobará si ha pasado una semana, para ello se comprobará si la variable día es igual a 7.
    - Si el resultado de la comprobación es negativo, el siguiente componente al que navegaremos será “MensajeFormativo.js”
    - En el caso de que el resultado de la comprobación sea verdadero, se pasará a mirar si el paciente ha estado realizando ejercicio al menos 3 días de la semana.
        - Si el paciente ha superado la comprobación, y el nivel de ejercicio no es el máximo, se procederá a aumentar el nivel de ejercicio del paciente. Acto seguido navegaremos al componente “MensajeEntrenamiento.js”, donde se informa al paciente de este hecho.
        - Si el paciente no ha superado la comprobación, y el nivel de ejercicio no es el mínimo, se procederá a reducir el nivel de ejercicio del paciente. De la misma forma que si hubiera superado la comprobación, navegaremos al componente “MensajeEntrenamiento.js”, donde se informa al paciente del cambio de nivel en sus ejercicios.

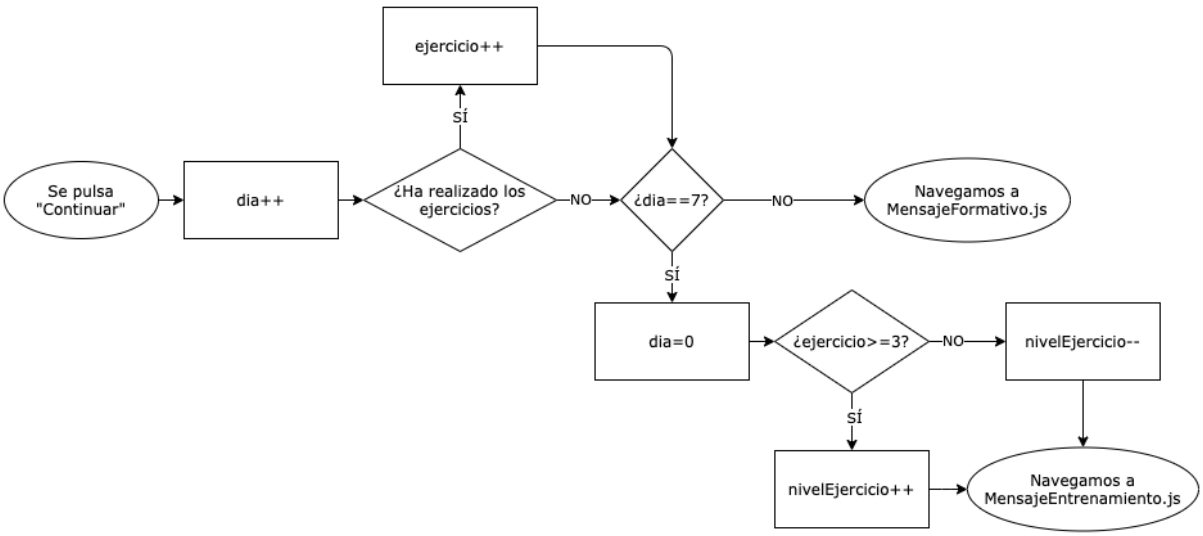

Ilustración 26. Diagrama de funcionamiento de PreguntaEjercicio.js.

3.5.3.4 EscuelaEspalda.js

Este componente es cargado cuando el paciente posee un nivel de dolor medio. Su objetivo es informar al usuario de la existencia de la Escuela Española de Espalda (EEDE).

La EEDE se define como un programa de educación sanitaria destinado a toda la población en general, especialmente a los pacientes con dolencias de espalda o colectivos predispuestos a padecerlas [15].

El componente muestra por pantalla el logotipo de la EEDE, el cual al ser pulsado redirige al usuario a la página web de la organización en el navegador web por defecto de su dispositivo móvil.

Ilustración 27. EscuelaEspalda.js en iOS.

En dicha página web el paciente puede encontrar normas de higiene postural, conocimientos sobre el

funcionamiento de la espalda, centros recomendados a los que acudir, e incluso productos y empresas recomendadas por la organización.

Al pulsar el botón de Continuar, el componente consultará en la base de datos el valor de la variable pautaAnalgésica, el cual recordemos representa el número de días que lleva un paciente, redundantemente, bajo el tratamiento de una pauta analgésica.

Si el paciente lleva menos de cuatro días asignado a una pauta analgésica, se procede a aumentar la variable en una unidad, y acto seguido se navega al componente "MensajeInformativo.js".

Por el contrario, si el paciente ha alcanzado los cuatro días bajo asignado a una pauta procederemos a navegar al componente "MoriskyGreen.js".

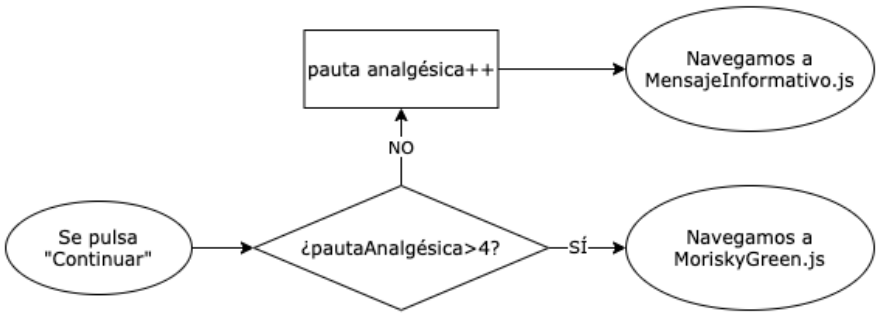

Ilustración 28. Diagrama de funcionamiento de EscuelaEspalda.js

3.5.3.5 MoriskyGreen.js

Este componente se encarga de realizar un test de Morisky-Green al usuario. Vamos a explicar cuál es la motivación de dicha prueba y en qué consiste.

Las enfermedades crónicas exigen cambios en el estilo de vida del paciente y un correcto seguimiento del tratamiento farmacológico. Esto supone un problema clínico considerable, ya que no seguir las recomendaciones de los profesionales conllevan implicaciones como:

- Un aumento de la dificultad en el control de la enfermedad, ya que los profesionales pueden considerar que el paciente sigue las recomendaciones, pero no consigue el resultado esperado. Esto puede provocar que el profesional modifique el tratamiento cuando no debería de hacerlo.
- Una reducción de la eficacia esperada, ya que no se están siguiendo las recomendaciones.
- Un malgasto de recursos del sistema sanitario o del propio paciente al no tomar o tomar de forma incorrecta su medicación.

Para comprobar que el paciente cumple de la manera esperada con el tratamiento aparece el test de Morisky-Green [16].

Dicho test cuenta con 4 preguntas las cuales el paciente debe responder con SÍ o NO.

El resultado del se considera positivo si el resultado de las cuatro preguntas si todas las respuestas proporcionadas por el paciente son las que se describen a continuación:

- ¿Olvida alguna vez tomar los medicamentos para tratar su enfermedad? (Respuesta correcta: No)
- ¿Toma los medicamentos a las horas indicadas? (Respuesta correcta: Sí)
- Cuando se encuentra bien, ¿Deja de tomar la medicación? (Respuesta correcta: No)
- Si alguna vez le sienta mal, ¿Deja usted de tomar la medicación? (Respuesta correcta: Sí)

En caso contrario, el resultado del test se considera negativo.

Ilustración 29. MoriskyGreen.js en iOS.

Una vez introducido el test de Morisky-Green, para presentarlo por pantalla el componente muestra cada pregunta seguida de dos iconos, uno de afirmación y otro de negación, para responderlas. Al pulsar en los iconos aparece un mensaje de confirmación para comprobar que el paciente está seguro de su respuesta.

Una vez se ha pulsado uno de los iconos de cada pregunta, aparecerá el botón para continuar, el cual permanecía oculto hasta ahora.

Cuando el usuario pulsa el botón, se da paso al algoritmo del componente:

- Si el resultado del test es negativo, se redirigirá la navegación al componente "MensajeRecordatorio.js".
- Si el resultado del test es positivo, se comprobará el valor de la variable dolorAlto. Si la bandera está activa significa que hemos llegado al test desde "NivelDolor.js" al introducir un valor dentro del rango del nivel alto, y navegaremos al componente "MensajeMedico.js"

  Por el contrario, si la bandera no está activa se comprobará cuantos días lleva el paciente bajo una pauta de rescate, mediante la variable pautaRescate. Si el paciente no lleva dos días bajo el tratamiento, se incrementará en una unidad la variable y se procederá a navegar al componente "MensajeInformativo.js".

  Si resulta que llevamos dos días bajo el tratamiento, procederemos una vez más navegando al componente "MensajeMédico.js"

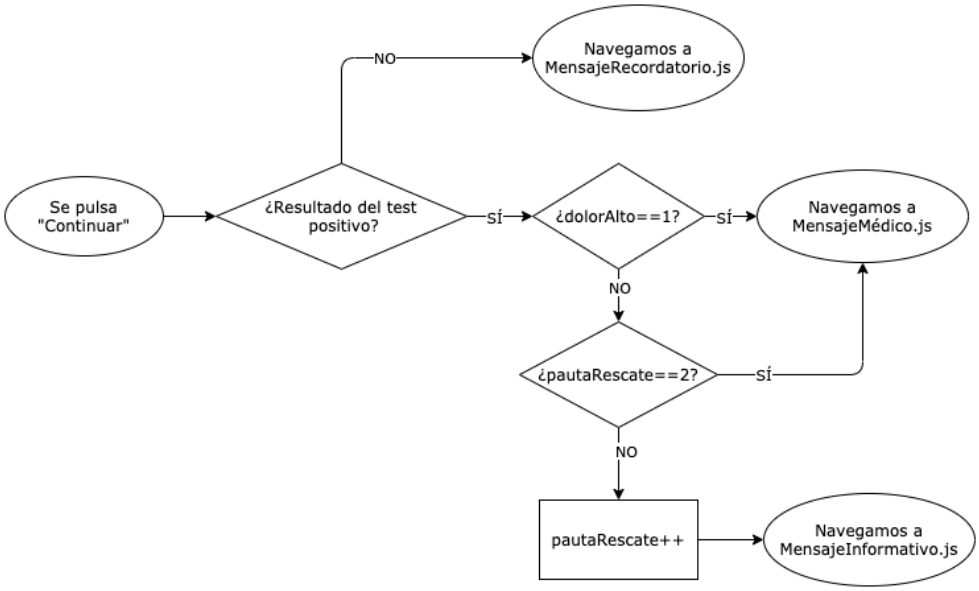

Ilustración 30. Diagrama de funcionamiento de MoriskyGreen.js

3.5.3.6 MensajeLlamada.js

Este componente sólo es cargado cuando el nivel de dolor es muy alto. En esta pantalla se insta al paciente a llamar a su médico.

Para ello se muestra por pantalla una entrada de texto en la que el paciente puede introducir el número de teléfono de su médico. Tras introducirlo, al pulsar el icono con forma de teléfono se comprobará que el teléfono introducido es un número telefónico valido, mostrando un mensaje de error en caso de no serlo. Si el número es correcto, se procede a realizar una llamada telefónica.

Además, se aconseja al paciente que pida cita en su médico cuanto antes, ya que un diagnóstico personal por parte de un profesional es necesario para poder tratar el nivel de dolor muy alto de la manera más adecuada.

Al pulsar en el botón de finalizar se nos redirigirá al componente inicial de la pestaña, “RegistroEVA.js”.

Ilustración 31. MensajeLlamada.js en iOS

3.5.3.7 MensajeEntrenamiento.js

Este componente aparece cuando el nivel de ejercicio ha sido modificado. Recordemos que esto ocurre cuando pasa una semana y se comprueba si si el paciente ha realizado o no los ejercicios recomendados más de 3 días.

En la pantalla se muestra un mensaje que informa al paciente de la actualización de su rutina de entrenamiento, y le insta a navegar a la pestaña “Ejercicio”, donde podrá visualizar los cambios.

Al pulsar en el botón de finalizar se nos redirigirá al componente inicial de la pestaña, “RegistroEVA.js”.

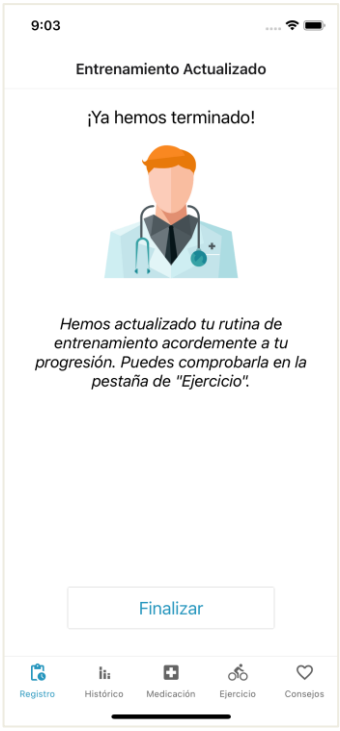

Ilustración 32. MensajeEntrenamiento.js en iOS

3.5.3.8 MensajeMedico.js

La función de este componente es avisar al paciente de que el tratamiento que tiene asignado no está funcionando. En este caso el paciente debe pedir cita con su médico para que un profesional pueda establecer un diagnóstico apropiado.

Al pulsar en el botón de “Finalizar”, seremos redirigidos al componente inicial de la pestaña, “RegistroEVA.js”

Ilustración 33. MensajeMedico.js en iOS

3.5.3.9 MensajeRecordatorio.js

El objetivo de este componente es recordar al paciente que debe seguir las recomendaciones de su médico sobre el tratamiento que tenga asignado, ya que si no es así no se puede garantizar la efectividad de este. El componente "MensajeRecordatorio.js" es cargado cuando el resultado del test de Morisky-Green es negativo.

El componente por pantalla un mensaje recordando al paciente que debe seguir las instrucciones de su médico respecto a la toma de la medicación.

Además, en su carga extrae de la base de datos un recordatorio. Dicho recordatorio está formado por un texto y una imagen, las cuales son mostradas en pantalla.

Al pulsar en el botón de finalizar, seremos redirigidos al componente inicial de la pestaña, "RegistroEVA.js"

Ilustración 34. MensajeRecordatorio.js en iOS

3.5.3.10 MensajeFormativo.js y MensajeInformativo.js

Ambos componentes presentan un estilo similar. Su función es extraer un mensaje formativo o informativo, respectivamente, de la base de datos. Los mensajes están compuestos de un texto y una imagen, los cuales son mostrados por pantalla.

Ilustración 35. MensajeFormativo.js en iOS

En el caso de “MensajeInformativo.js”, el componente es llamado cuando la medicación asignada al paciente ha cambiado. Debido a esto, el componente imprime por pantalla un mensaje adicional recomendando al usuario que navegue a la pestaña de “Medicación” para poder observar los cambios en su medicación.

Ilustración 36. MensajeInformativo.js en iOS

3.5.4 Pestaña “Histórico”

Una de las partes más visuales e interactivas de la aplicación, la pestaña “Histórico” muestra al usuario el registro a lo largo del tiempo de su dolencia lumbar. El paciente así puede directamente ver la progresión de su dolencia de una manera sencilla.

La pestaña está formada por todos los componentes pantalla pertenecientes a “HistoricoStack.js”. En nuestro caso sólo es uno, pero como comentamos anteriormente esta distribución nos permite poder añadir componentes pantallas adicionales a la pestaña si fuera necesario en un futuro.

3.5.4.1 Historico.js

El objetivo de este componente es mostrar por pantalla los datos guardados en la base de datos del paciente. Para ello, muestra en la parte inferior una barra de doce botones para escoger el mes que queremos seleccionar.

Hasta que no se pulse ninguno de los botones, las dos gráficas aparecerán vacías y se instará al usuario mediante un mensaje a seleccionar un mes.

Una vez pulsado, el componente tomará los datos del histórico del paciente. Para ello comprobará que el NID de los datos coincide con el del usuario, y guardará en dos tablas diferentes el día del mes y el nivel de dolor que se introdujo.

Con esos datos, haciendo uso del paquete “React Native Chart Kit”, dibujará dos gráficas.

- La primera gráfica es una gráfica PieChart (gráfica circular). Este tipo de gráficas es adecuado cuando se pretende mostrar la magnitud de las proporciones que forman un conjunto y no hace falta un nivel de exactitud grande de los datos. En nuestro caso, mostramos que porcentaje de los días del mes el paciente ha estado en un nivel de dolor.
- La segunda gráfica es una gráfica lineal. Este tipo de gráficas consiste en una serie de datos

representados por puntos, los cuales son unidos por segmentos lineales. Estas gráficas son útiles para examinar el cambio en los datos y su tendencia. Nuestra gráfica representará en el eje horizontal el día del mes en el que se registró el dato, y en el eje vertical el valor de este.

En caso de que el mes seleccionado no tuviera datos, ambas gráficas se pintan vacías y se imprime por pantalla un mensaje avisando al usuario de que el mes seleccionado no tiene datos.

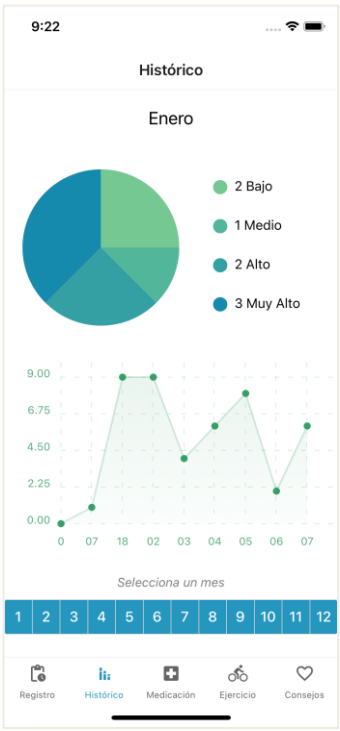

Ilustración 37. Historico.js en iOS

3.5.5 Pestaña “Medicación”

El objetivo de esta pestaña es mostrar al paciente todo lo relacionado con su medicación. El conocimiento por parte del usuario de su medicación asignada, así como su uso es de vital importancia a la hora de tratar enfermedades crónicas, ya que un uso indebido de la misma puede llegar incluso a afectar negativamente al paciente. Esta pestaña está compuesta de los componentes pantalla pertenecientes a “MedicacionStack.js”.

3.5.5.1 Medicacion.js

Al cargarse el componente, comprobará en la base de datos el valor de las variables pautaRescate y pautaAnalgésica.

La pauta de rescate es la más restrictiva de las tres pautas (recordemos: basal, analgésica y de rescate), por lo cual, si el paciente tiene asignada una pauta analgésica y una de rescate, esta última será la que se mostrará por pantalla. También es conveniente remarcar que como norma general un médico no asignará más de 5 medicamentos diferentes a una pauta.

Tras comprobar el tipo de pauta que tiene asignada el paciente, el componente muestra por pantalla un mensaje donde informa al usuario sobre la asignación de esta.

A continuación, utilizando el tipo de pauta que tiene asignada el paciente y su NID, procede a extraer de la base de datos los medicamentos que posee su tratamiento.

Dichos medicamentos constan de tres datos: el nombre del fármaco, la cantidad que debe tomar el paciente, y

el número de veces que debe usar el fármaco en un día.

Una vez el componente posee los datos, los imprime en una tabla cuya estructura podemos ver en la ilustración 38.

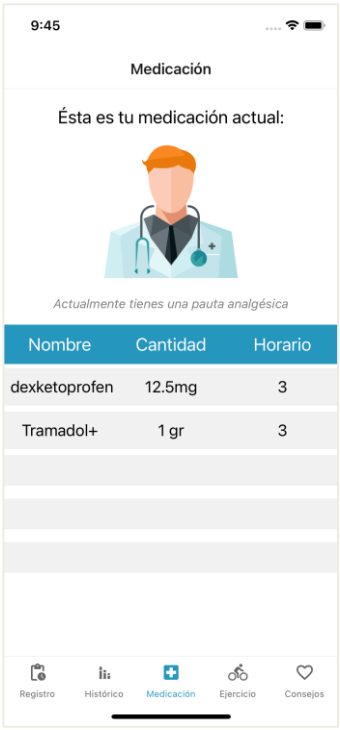

Ilustración 38. Medicacion.js en iOS

3.5.6 Pestaña "Ejercicio"

En esta pestaña el paciente tendrá acceso a los ejercicios recomendados según su nivel físico. Recordemos que este nivel es incrementado o reducido en función de si el usuario los realiza habitualmente o no.

Se ha demostrado que los ejercicios lumbares adecuados pueden reducir el riesgo de padecer dolor de espalda, además de mejorar la evolución de este en pacientes crónicos [17].

Por ello, el ejercicio físico es una de las claves en el tratamiento del dolor crónico lumbar. Gracias al ejercicio, el paciente puede desarrollar la potencia, resistencia y elasticidad de los músculos implicados en el sostén y funcionamiento de la espalda. Cuanto mejor desarrollada esté la musculatura lumbar del usuario, menor es la probabilidad de sufrir crisis dolorosas de la espalda.

Esta pestaña abarca todos los componentes pantalla pertenecientes a "EjercicioStack.js"

3.5.6.1 Ejercicio.js

La función del componente es mostrar al usuario su nivel actual de ejercicio y mostrar por pantalla un vídeo embebido de YouTube por cada ejercicio que tenga asignado ese nivel.

Para ello, en su carga, consulta la variable nivelEjercicio del paciente en la base de datos. Una vez comprobado en nivel asignado, realiza otra petición a la base de datos para obtener los ejercicios.

Cada ejercicio está compuesto de un identificador y de un enlace que redirige a un vídeo explicativo sobre cómo realizar dicho ejercicio.

El componente manipula ese enlace para poder incrustar el vídeo en la propia aplicación, sin necesidad de que se nos redirija al navegador móvil por defecto de nuestro dispositivo.

La muestra en pantalla, así como la reproducción y manejo de los controles del vídeo, son posibles gracias al paquete "React Native YouTube iFrame".

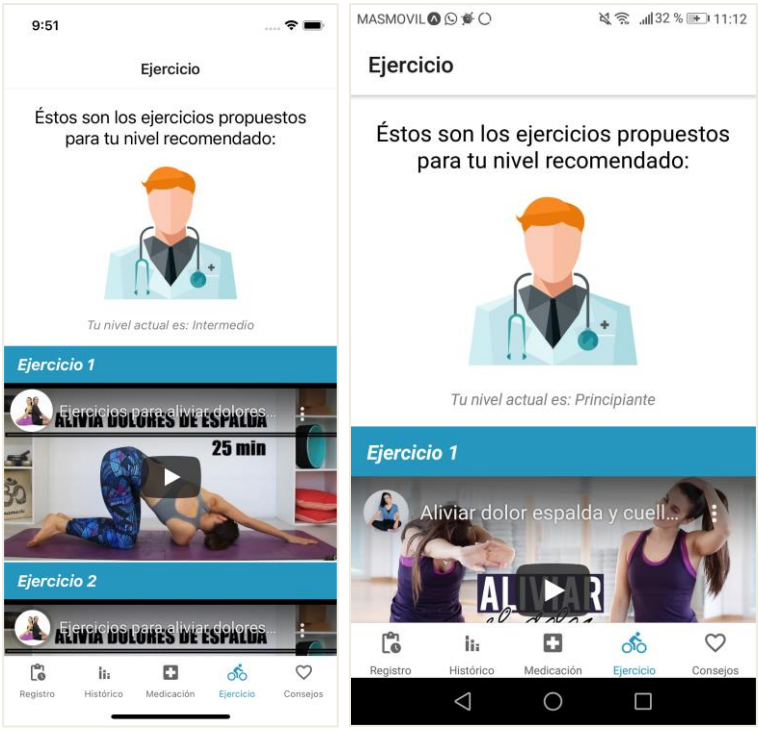

Ilustración 39. Ejercicio.js en iOS y Android.

3.5.7 Pestaña “Consejos”

La última pestaña de la aplicación, su objetivo es mostrar al paciente un consejo sobre medicación, alimentación o ejercicio físico que pueda ayudar al paciente con su dolencia. Pretende instruir al paciente sobre su dolencia a base de pequeños trozos de información, con la intención de que sea más fácil recordarlos al presentarlos en un formato reducido.

Esta pestaña comprende todos los componentes pantalla pertenecientes a “ConsejosStack.js”.

3.5.7.1 Consejos.js

Al cargarse, el componente realiza una petición a la base de datos para cargar todos los consejos. Un consejo está formado por un identificador, una imagen y un texto.

Una vez el componente obtiene la respuesta, escoge aleatoriamente uno de los consejos obtenidos. Tras ello, procede a mostrar por pantalla la imagen y el texto asociados a este como podemos ver en la ilustración 40.

Además, el componente incorpora un botón en la parte inferior con el texto “Recargar”. Al ser pulsado, vuelve a escoger otro de los consejos y refresca la pantalla para cambiar la imagen y el texto a las del nuevo consejo.

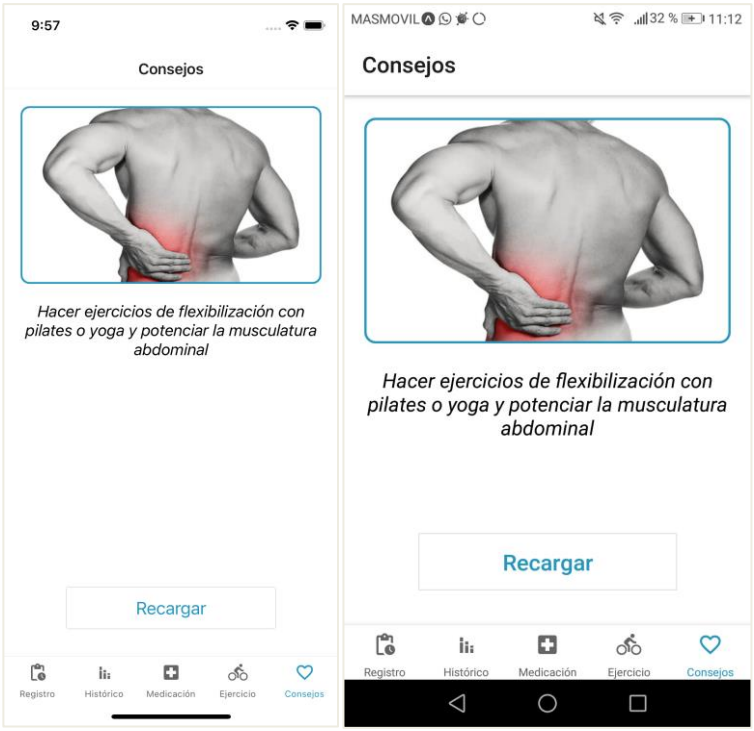

Ilustración 40. Consejos.js en iOS y Android.

4 INSTALACIÓN DE LA APLICACIÓN

En este capítulo vamos a explicar el procedimiento necesario para la instalación y funcionamiento de la aplicación a través de la plataforma Expo. Lo hacemos a través de la plataforma Expo ya que para poder incluir una aplicación móvil en las plataformas de aplicaciones es necesario la aprobación, en este caso, de Google Play y App Store. Mediante Expo, podemos ejecutar nuestra aplicación como si se tratara de una aplicación obtenida a través de estos portales y probarla en nuestro teléfono.

4.1 Subiendo el proyecto a Expo

Asumiendo que poseemos instalado el cliente Expo-CLI (el cual ha sido necesario para la creación del proyecto), lo primero que debemos hacer es iniciar sesión con nuestra cuenta de Expo, o bien crear una si no tenemos. Esto se en la carpeta de nuestro proyecto, mediante los comandos "expo login" o "expo register" respectivamente.

Una vez realizado este paso, desplegar nuestro proyecto en la plataforma es tan sencillo como ejecutar el comando "expo publish". Una vez lo hagamos, si no ha habido ningún problema se imprimirá por pantalla un mensaje similar al siguiente.

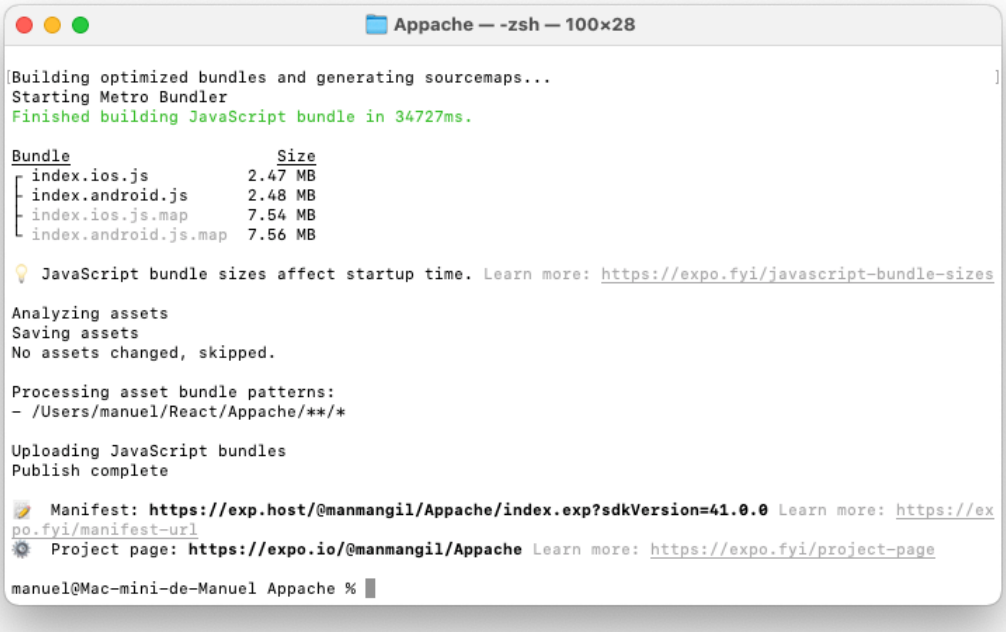

Ilustración 41. Resultado de la ejecución del comando "expo publish"

En el resultado obtendremos un enlace a la página de nuestro proyecto. Si accedemos desde un navegador web lo que veremos será el icono de la aplicación y su nombre, seguido de un Código QR y un enlace.

Tanto el código QR como el nos servirán para poder abrir el proyecto desde el cliente móvil de Expo.

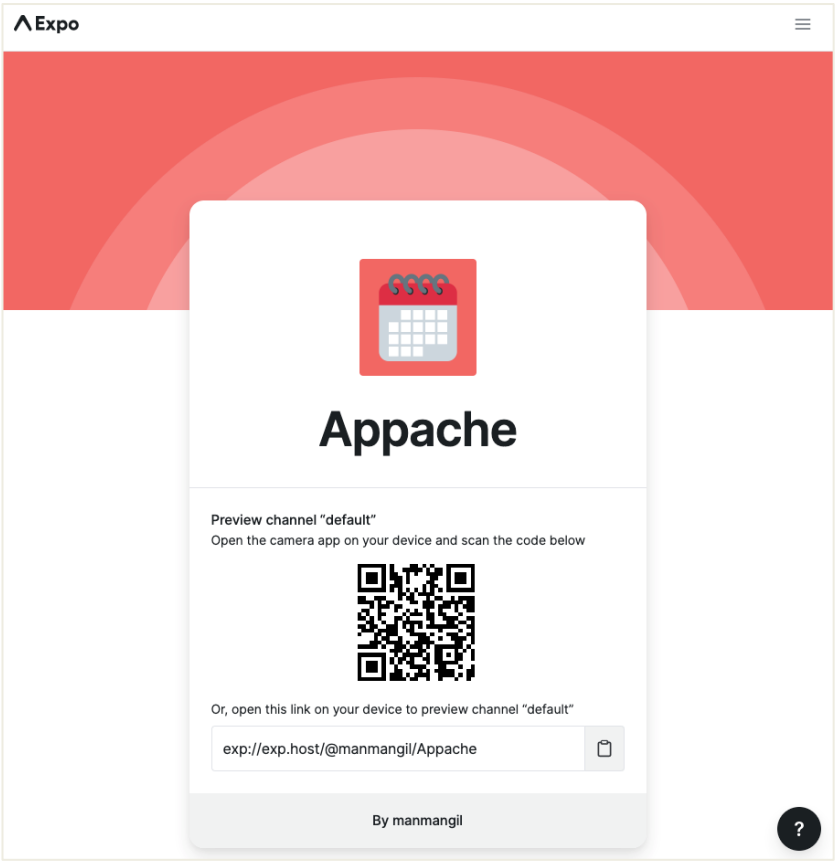

Ilustración 42. Página web del proyecto en la plataforma Expo.

4.2 Expo Go

Una vez ya tenemos el proyecto en la plataforma Expo, hacerlo funcionar en nuestro dispositivo móvil no nos llevará mucho.

El primer paso es acceder a la tienda de aplicaciones móviles de nuestro dispositivo (Google Play para Android, App Store para iOS) y buscar la aplicación "Expo Go". Una vez encontrada, la descargamos e instalamos en nuestro dispositivo.

Una vez instalada, al abrirla obtendremos la pantalla que aparece en la ilustración 43.

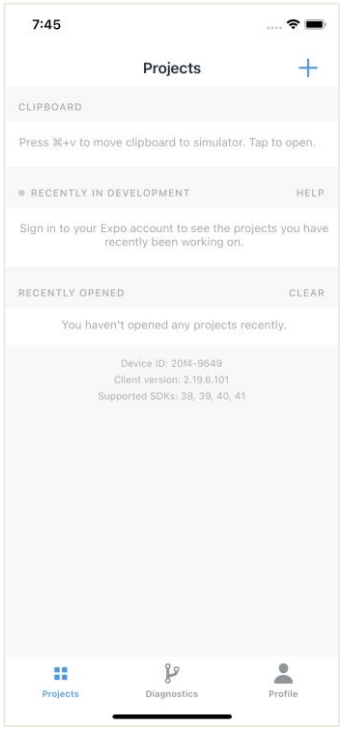

Ilustración 43. Aplicación Expo Go.

Desde aquí, en la pestaña “Profile” podemos iniciar sesión en nuestra cuenta. Una vez realizado el inicio de sesión tendremos acceso a nuestros proyectos publicados y podremos ejecutarlos como si fueran una aplicación aparte.

Pero esto no es necesario para poder ejecutar la aplicación. Cualquier usuario con la aplicación instalada puede acceder a nuestro proyecto utilizando el código QR o el enlace que vimos en la sección 4.1 [18]

Ilustración 44. Código QR de la aplicación "Appache"

REFERENCIAS

[1] https://reactnative.dev/

[2] https://es.reactjs.org/docs/introducing-jsx.html

[3] https://es.reactjs.org/

[4] https://es.reactjs.org/docs/hooks-intro.html

[5] https://expo.dev/

[6] https://reactnativeelements.com/

[7] https://reactnavigation.org/

[8] https://momentjs.com/

[9] https://github.com/indiespirit/react-native-chart-kit

[10] https://www.npmjs.com/package/react-native-youtube-iframe

[11] https://code.visualstudio.com/

[12] https://reactnavigation.org/docs/stack-navigator/

[13] https://reactnavigation.org/docs/bottom-tab-navigator/

[14] https://www.cancer.gov/espanol/publicaciones/diccionarios/diccionario-cancer/def/escala-visual-analogica

[15] http://eede.es/

[16] https://www.ayudasdinamicas.com/blog/test-de-morisky-green-spd/

[17] http://eede.es/la-espalda/ejercicios/

[18] https://expo.dev/@manmangil/Appache

GLOSARIO

REST: Representational State Transfer

Framework: Marco de trabajo

JSX: JavaScript XML

JS: JavaScript

RN: React Native

DOM: Document Object Model

VDOM: Virtual Object Model

HTML: HyperText Markup Language

SDK: Software Development Kit

JSON: JavaScript Object Notation

HTTP: HyperText Transfer Protocol

NID: Número Identificador

EVA: Escala Visual Analógica

EEDE: Escuela Española de Espalda

QR: Quick Response