Universidad de Valladolid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Trabajo de Fin de Grado

Grado en Ingeniería de Tecnologías de Telecomunicación

Diseño y desarrollo de la interfaz de usuario de una aplicación web para la mejora de la salud basada en un chatbot

Autor:

Pablo Juan Fernández Cotrina

Tutor:

Juan Pablo de Castro Fernández

Valladolid, 1 de julio de 2021

TÍTULO:	Diseño y desarrollo de la interfaz de usuario de una aplicación web para la mejora de la salud basada en un chatbot
AUTOR:	Pablo Juan Fernández Cotrina
TUTOR:	Juan Pablo de Castro Fernández
DEPARTAMENTO:	Teoría de la Señal y Comunicaciones e Ingeniería Telemática

TRIBUNAL

PRESIDENTE:	Juan Pablo de Castro Fernández
VOCAL:	María Jesús Verdú Pérez
SECRETARIO:	Luisa María Regueras Santos
SUPLENTE:	Jaime Gómez Gil
SUPLENTE:	Ramón de la Rosa Steinz
FECHA:	1 de julio de 2021
CALIFICACIÓN:	

Resumen del TFG

En este Trabajo de Fin de Grado, gracias a las tareas realizadas, ha sido posible la creación de la interfaz de usuario de una aplicación que ofrece un seguimiento personalizado a sus usuarios, con el que consiguen desarrollar su máximo potencial. Un proyecto que ha nacido de la detección de la necesidad de reducir el impacto económico consecuencia de la mala salud de los trabajadores. La empresa Ingage ha querido poner solución a este problema enfocándose en su sector de actuación, el sector de los seguros.

Ante las diferentes posibilidades de desarrollo de la aplicación, se ha optado por una aplicación híbrida, que permitiera tanto una versión web como una versión nativa. A lo largo del proyecto se ha seguido un proceso de integración continua, en el que, a medida que se han ido creando nuevas funcionalidades, se han ido testeando, con el objetivo de conseguir una aplicación mucho más robusta.

La aplicación, *Full Power*, ha sido diseñada para ser una aplicación atractiva, fácil de utilizar y adaptativa a cualquier tamaño de dispositivo. La aplicación en su conjunto está formada por la parte cliente y la parte del servidor, por lo que ha sido necesaria la creación de una API (*Application Programming Interface*) que habilitara la comunicación entre ambas. Además, gracias a haberse tratado de una aplicación híbrida, solo ha sido necesario un único desarrollo (en versión web) para obtener la versión final, finalizando con una conversión de web a nativo.

El *chatbot* que contiene la aplicación permite obtener información de los usuarios mediante una encuesta de evaluación inicial, cuyo algoritmo es capaz de detectar posibles problemas de salud en dichos usuarios. Posteriormente, puede obtener nueva información a través de preguntas que realiza el *chatbot* lanzadas en las notificaciones locales del dispositivo (solo disponible en Android). Además, los usuarios pueden consultar los documentos de salud elaborados por el experto naturópata del equipo de trabajo.

Palabras clave

Salud, aplicación híbrida, Angular, IONIC Capacitor, Android, test, API, *frontend*, documento, fichero, integración continua.

Abstract

In this thesis, thanks to completed tasks, it has been possible to create an app user interface that offers customizable tracking to its users, who can reach their full potential. A project which was born because of the need to shrink the economic impact that is consequence of lack of health from workers. The Ingage company has chosen to solve this problem by focusing in their business field, Insurance field.

Given the different possibilities for the app development, a hybrid app, that allows both web and native versions, has been chosen. Throughout the project, continuous integration process has been followed. In this process, by the time new functionalities have been created, they have been being tested, on purpose of getting such a robust application.

The application, *Full Power*, has been designed to be an attractive, easy-to-use and responsive application. The application as a whole is composed of a client part and a server part, consequently there was a need to create an API (Application Programming Interface) that enabled communication between both sides. In addition, thanks to have been a hybrid app, a sole development has been needed (web version) to get final version, ending with a conversion from web to native.

The chatbot that is inside the app enables get information from users by an initial evaluation survey, which algorithm can detect possible health problems on these users. The following times, it can get new information through the questions thrown on the local notifications of the device (only enabled in Android). Furthermore, users can check the health documents written by the team's naturopath expert.

Keywords

Health, hybrid Application, Angular, IONIC Capacitor, Android, test, API, *frontend*, document, file, continuous integration.

Contenido

Índice de figuras

1. Introducción

1.1. Motivación

Cuando se habla de salud desde un punto de vista médico, no solo se trata de curar enfermedades ya perceptibles, sino que también se trata de prevenirlas. Además, no afecta únicamente de manera directa al estado de nuestro cuerpo y mente. Tiene efectos sobre aquello que nos rodea, las actividades en las que estamos involucrados o las personas con las que nos relacionamos.

Existen varios factores importantes que afectan notablemente a la salud entre los que se encuentran el sueño, la alimentación y el estrés. Incluso estos factores se afectan de manera recíproca.

Dormir es una necesidad básica del ser humano. Existen diferentes causas por las que uno no consigue dormir lo suficiente: la temperatura, la posición del cuerpo, preocupaciones, drogas… La falta de sueño provoca cambios de humor, irritabilidad, bajada de la productividad, descenso de la concentración, puede incrementar la obesidad y el riesgo de enfermedades cardiovasculares. Se relaciona dormir poco con un incremento de hasta un 40% de posibilidades de tener una muerte temprana [1].

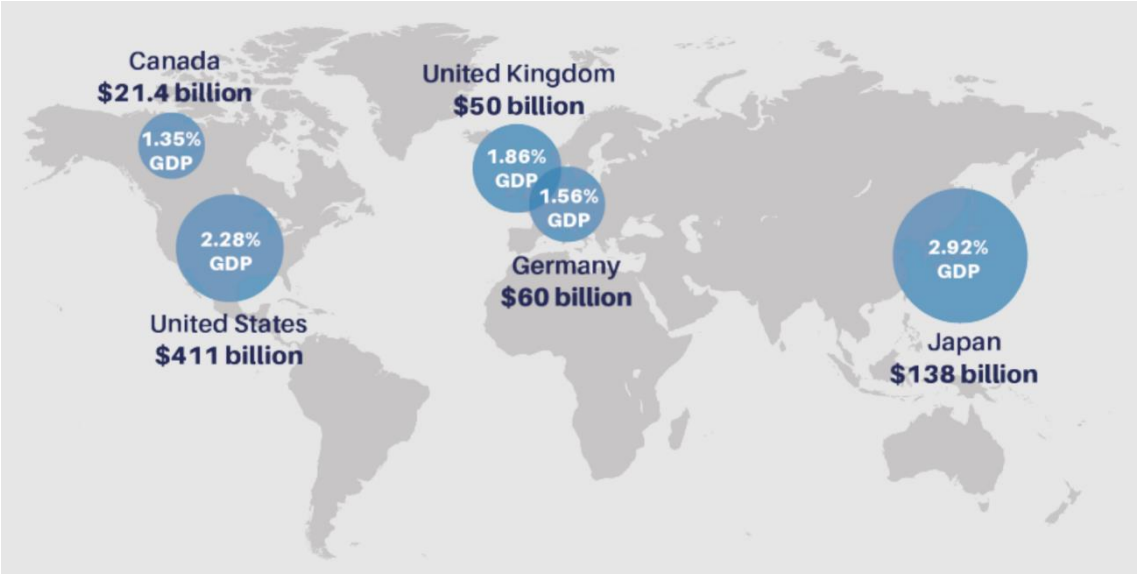

Figura 1.- Pérdidas económicas debidas a un sueño insuficiente en 5 países de la OCDE.

En la *Figura 1* podemos ver el impacto económico de la falta de sueño en diferentes países del mundo y que, aunque sea un efecto pequeño porcentualmente, se trata de unos volúmenes monetarios desmesurados [2].

La alimentación tiene un gran impacto en la salud y en la productividad, por ejemplo, la falta de hierro provoca fatiga extrema a 740 millones de personas en el mundo, provocando que su capacidad de trabajo se vea disminuida. Otro ejemplo, es que se ha calculado que en Estados Unidos se pierden al año 39.2 millones de días de trabajo por motivos relacionados con la obesidad [3].

El estrés en los trabajadores también influye de manera negativa en su salud, pudiendo derivar en problemas más graves relacionados con los desórdenes mentales, y en su rendimiento. Se estima que el coste anual en Europa relacionado con este tipo de enfermedades mentales (como

la depresión) es de 617 miles de millones de euros (teniendo en cuenta las pérdidas por absentismo que produce, la pérdida de productividad, los costes sanitarios de los tratamientos y los costes en caso de incapacidad laboral) [4].

Tras ver los datos mostrados, parece de vital importancia centrarse en mejorar la salud de los trabajadores, lo que no afectará únicamente a su bienestar sino también al crecimiento económico.

Ingage, empresa del sector de seguros, se dedica a desarrollar una plataforma digital modular con el objetivo de diseñar cursos para la formación de empleados de este sector. Tratando de evitar la pesadez habitual de este tipo de cursos, en Ingage tratan de crear cursos que resulten amenos para los empleados, actualmente mediante unas interfaces atractivas visualmente, y con el objetivo de conseguir diseñar los cursos en un mundo de realidad virtual, haciendo del aprendizaje un juego.

Tienen otros proyectos con los que buscan mejorar la productividad de los trabajadores, como el desarrollo de una inteligencia artificial para ayudar a los empleados del mundo de los seguros a la hora de trabajar con los clientes, para poder resolver todas las dudas que puedan surgir.

En esta línea, la empresa Ingage se plantea la realización de una aplicación, que va a ser objeto de desarrollo en el presente Trabajo de Fin de Grado, con la que conseguir la mejora de la salud de los trabajadores del sector de los seguros. Esta mejora de salud les permitirá alcanzar su máximo potencial, evitando problemas futuros de salud más graves y, como fin último, incrementar su productividad en el trabajo. El experto naturópata Stephan Tétart será el encargado de proporcionar toda la información sobre salud. Esta aplicación tendrá que hacer uso de un *chatbot*, que permita adaptar la información más relevante para cada usuario, es decir, tener un servicio completamente personalizado, fácilmente utilizable y atractivo visualmente.

1.2. Objetivos

El objetivo de este Trabajo de Fin de Grado es diseñar y desarrollar una interfaz de usuario de una aplicación híbrida para permitir a sus usuarios mejorar su salud a largo plazo y alcanzar su máximo potencial.

Esta interfaz de usuario será adaptable a todo tipo de tamaños de pantalla y permitirá a los usuarios consultar con un *chatbot* asuntos sobre su salud y, además, obtener documentación acerca de dichos asuntos. La interfaz obtendrá toda esta información de un servidor mediante la utilización de una API REST (*Application Programming Interface Representational State Transfer).*

La interfaz de usuario requiere que, de manera concurrente al desarrollo del objeto de este trabajo, se desarrolle el *backend*, que proporcionará todos los servicios que sean necesarios para cumplir con los requisitos del proyecto. Esta parte del servidor ha sido desarrollada por Diego Alloza González, alumno del Programa de estudios conjunto de Grado en Ingeniería de Tecnologías de Telecomunicación y Grado en Administración y Dirección de Empresas [5].

1.3. Metodología

En este proyecto se ha seguido una metodología de desarrollo ágil mediante Scrum [6]. Toda la gestión de esta metodología se ha realizado con el software Jira [7]. Scrum es una metodología iterativa e incremental, por lo que el trabajo, se configura para el continuo desarrollo de

diferentes funcionalidades a lo largo de *sprints* de duración entre 2 y 4 semanas, en cada *sprint* se añadiría una nueva funcionalidad. La estructura de un *sprint* es la siguiente:

- Reunión inicial en la que se establece el plan para el presente *sprint*, objetivos y plazos.
- Diseño de la funcionalidad que se desea implementar.
- Desarrollo de la funcionalidad.
- Testeo de la funcionalidad.
- *Feedback* final para comprobar si se han cumplido los objetivos.

Además, existen varios roles en esta metodología:

- Dueño del producto: la persona que financia el proyecto y quien obtendría los resultados de este. Se encarga de establecer los objetivos a un alto nivel.
- *Scrum Master*: persona que dirige toda la gestión de Scrum, divide los objetivos de alto nivel marcados por el dueño del producto en tareas más pequeñas y realizables y las asigna a los distintos componentes del equipo.
- Equipo: encargados del desarrollo del producto.

El desarrollo de la funcionalidad de un sprint se divide en tareas más simples, que son asignadas a los trabajadores del equipo. Estas tareas deben contar con una breve descripción del objetivo que se pretende conseguir. El trabajador asignado debe ser quien informe del estado de la tarea, indicando si ha tenido éxito, los resultados y el tiempo que ha empleado en llevar a cabo dicha tarea. Es habitual que en el mismo equipo haya otros miembros encargados de verificar las tareas realizadas y comprobar que se siguen los criterios establecidos.

1.4. Estructura de la memoria

Esta memoria se encuentra dividida en seis capítulos, entre los que se incluye el presente y, se ha desarrollado una introducción del proyecto que permite poner en contexto la aplicación.

En el *Capítulo 2* se describen los tipos de aplicaciones móviles, los lenguajes de programación que permiten su desarrollo y de otras aplicaciones de salud que ya existen.

El *Capítulo 3* explica la descripción del proceso de análisis de la aplicación, mediante la definición de los requisitos del sistema, tanto funcionales como no funcionales, los casos de uso y el prototipado de la aplicación, que han sido las guías para el siguiente capítulo.

En el *Capítulo 4* se describe todo el desarrollo de la aplicación. Comenzando por una breve introducción de las distintas tecnologías utilizadas, seguido de la explicación de la arquitectura del sistema. A continuación, se explican las fases de desarrollo, testeo y despliegue de la aplicación. Por último, se muestran la estructura de ficheros y detalles concretos de la implementación.

En el *Capítulo 5* se muestran los manuales de uso de la aplicación tanto para usuarios básicos como para el administrador.

El *Capítulo 6* cierra la memoria con las conclusiones alcanzadas en el proyecto y los pasos a seguir para continuar con el desarrollo del proyecto. Finalmente, se muestra la bibliografía utilizada para la realización de la memoria.

2. Estado del arte

En este capítulo voy a explicar el contexto en el que se encuentra la aplicación que se va a desarrollar. En primer lugar, con los diferentes tipos de aplicaciones móviles que existen: las aplicaciones nativas, las aplicaciones web y las aplicaciones híbridas. En segundo lugar expondré las aplicaciones de salud que podemos encontrar en las diferentes tiendas de aplicaciones para situar la aplicación a desarrollar en su contexto.

2.1. Tipos de aplicaciones móviles

2.1.1. Aplicaciones nativas

Cuando se habla de aplicaciones nativas de móviles, se refiere a una aplicación que ha sido escrita utilizando el lenguaje de desarrollo y las herramientas específicas de dicha plataforma. Por ejemplo: una aplicación nativa de iOS estaría desarrollada bien en Swift o en Objective-C y compilada mediante Xcode. Una aplicación nativa de Android estaría escrita utilizando Java o Kotlin y compilada utilizando Android Studio.

La gran ventaja de desarrollar una aplicación en el lenguaje nativo es que los desarrolladores tienen acceso total al dispositivo y de una manera sencilla, a todos los sensores, a la lista de contactos del usuario o a cualquier otra característica que pueda ofrecer un teléfono móvil. Además, estas aplicaciones tienden a tener un mejor rendimiento dado que el código se encuentra más cercano la parte física del dispositivo. También se tiene acceso a todos los controles nativos de las diferentes interfaces de usuario. No solo es posible dar estilo a las aplicaciones de para que encaje con el tema de la aplicación, sino que también se puede configurar el comportamiento y las interacciones con cualquier otro elemento de la interfaz de usuario de esa plataforma. El código compilado suele ser más rápido que el código interpretado, como lo es JavaScript [8].

La contra de este desarrollo es que no permite el uso cruzado, es decir, una aplicación de Android programada en Kotlin no puede ser lanzada en un dispositivo iOS y viceversa, por lo que sería necesario especializarse en cada plataforma. Además, no solo influye el sistema operativo, sino que también habrá que tener en cuenta la versión de dicho sistema, porque no todas las funcionalidades estarán disponibles en todos. Todo ello conllevará mayores costes a nivel de equipo desarrollador por necesitar especialistas de cada lenguaje, en el caso de desear lanzar la aplicación tanto en Android como en iOS. Y, como es previsible, solo estará disponible en su respectiva tienda de aplicaciones y sujeta a sus reglas y restricciones, lo que influirá a la hora de lanzar nuevas versiones, ya que tendrán que superar los criterios de dichas tiendas.

2.1.2. Aplicaciones web

Esas limitaciones a la hora de subir una aplicación a estas tiendas y el posible riesgo de que no superen los criterios exigidos por ellas, provocando, finalmente, pérdidas económicas hace que muchos desarrolladores escojan desarrollar únicamente aplicaciones web, ya que les da la libertad de existir fuera de esas tiendas de aplicaciones.

Estas aplicaciones están escritas en JavaScript, HTML y CSS. Y, actualmente, existe un amplio rango de *frameworks* de JavaScript, como Angular, React o Vue. Estos lenguajes sí permiten el soporte multiplataforma, lo cual evita necesitar tanta especialización para cada dispositivo concreto.

El problema de este desarrollo es la limitación de la aplicación a las capacidades del navegador del dispositivo. En este caso, no podremos tener acceso completo al dispositivo. Aunque bien es

cierto que, en los últimos años, se han añadido características como el acceso a la cámara, al GPS, al almacenamiento local, al Bluetooth o el acelerómetro [9].

2.1.2.1. Aplicaciones Web Progresivas (PWA)

En 2017, Google introdujo un nuevo concepto, el de las Aplicaciones Web Progresivas (PWA, *Progressive Web Applications*). Estas aplicaciones se acercan algo más a las aplicaciones nativas ya que incluyen características como las notificaciones *push* (notificaciones que provienen del servidor) y algunas funcionalidades *offline.* Chrome tiene el soporte más robusto, consecuencia de que las PWA han sido uno de los objetivos de Google en los últimos años [10].

Las PWA son una manera de ofrecer una experiencia en una aplicación de móvil que está muy optimizada, es fiable y completamente accesible desde la web. Algunas de sus características de estas aplicaciones además de las que ya tienen las aplicaciones web tradicionales:

- Fiables: funcionan *offline* y en redes de mala calidad gracias a los *service workers* (permiten cachear los recursos de las aplicaciones, Figura 1*Figura 2*). Tampoco tienen problemas de funcionamiento en dispositivos antiguos.

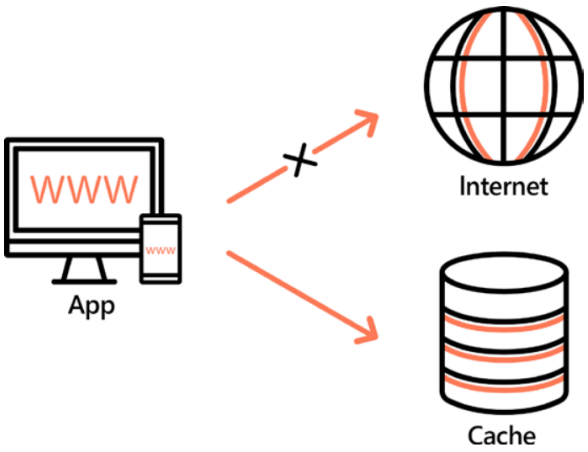

Figura 2.- Uso de la caché cuando la conexión a internet no funciona [10].

- Rápidas: cargan en pocos segundos, con suaves transiciones dentro de la aplicación.
- Fáciles de encontrar: a través de cualquier motor de búsqueda.
- Pequeño tamaño: ocupan una fracción del tamaño normal de una aplicación cualquiera de las tiendas de aplicaciones.
- Al día: su contenido está siempre actualizado con el último contenido del servidor.
- Instalables: se pueden guardar las aplicaciones que los usuarios consideran útiles en el escritorio/pantalla de inicio.
- Accesibles: funcionan en cualquier dispositivo que pueda utilizar un navegador, sea móvil o escritorio.

2.1.3. Aplicaciones híbridas

Las aplicaciones híbridas son otra solución intermedia entre las aplicaciones nativas y las aplicaciones web. El núcleo de la aplicación está escrito empleando tecnologías web: JavaScript, HTML y CSS, que está encapsulado en una aplicación nativa. Estas aplicaciones pueden tener acceso completo a todas las características de los dispositivos móviles a través del uso de las librerías que ofrecen los diferentes *frameworks*.

El código se incrusta dentro de la aplicación nativa utilizando Apache Cordova o IONIC Capacitor. Estas soluciones crean el caparazón de una aplicación nativa que presenta un componente *webview* en el que se carga la aplicación deseada. Esto permite crear y publicar aplicaciones web que pueden ser obtenidas a través de las tiendas de aplicaciones.

La estructura de las aplicaciones híbridas la podemos ver en el esquema de la *Figura 3*.

Figura 3.- Estructura de una aplicación híbrida.

Es posible utilizar librerías para ir más allá de los límites del navegador y acceder a todas las características de los dispositivos. Estas librerías son proporcionadas tanto por Cordova como por Capacitor. Esto permite tener el mismo acceso que una aplicación nativa.

De la misma manera que ya ocurría con las aplicaciones web, en las aplicaciones híbridas también deberemos recrear los componentes de la interfaz de usuario. Aunque existen soluciones robustas proporcionadas por IONIC o React Native [11]. Seremos capaces de personalizar tanto la aplicación como cualquier elemento de la interfaz de usuario del dispositivo, una de las ventajas que tenían las aplicaciones nativas es igualada por las aplicaciones híbridas.

Característica	**Nativa**	**Web**	**PWA**	**Híbrida**
Soporte multiplataforma	No	Limitado	Limitado	Sí
Acceso al hardware	Sí	Limitado	Limitado	Sí

En esta tabla podemos ver las diferencias más destacables entre las modalidades de aplicaciones móviles. En Web y PWA el soporte multiplataforma y acceso al hardware limitados se deben a ciertas incompatibilidades de alguna funcionalidad concreta con algunos navegadores, como por ejemplo el caso del evento relacionado con las notificaciones *push* [12].

Ante las ventajas de desarrollar una aplicación híbrida esta ha sido la opción seleccionada por el equipo para la aplicación *Full Power*, ya que aumentará la capacidad de dispositivos en los que pueda ser empleada sin renunciar a la sencillez de programación del desarrollo web ni a la

capacidad de personalización máxima que ofrecen las aplicaciones nativas tanto a nivel software como hardware.

2.2. Aplicaciones de salud

Actualmente, existen múltiples aplicaciones relacionadas con la salud en las tiendas de aplicaciones de Apple y de Google. Tras observar parte de ellas, he establecido varios grupos en función de las características principales que ofrecen: aplicaciones de seguimiento deportivo, de seguimiento general de la salud y para la mejora de la salud.

2.2.1. Aplicaciones de seguimiento deportivo

Estas aplicaciones se basan en realizar un seguimiento de las actividades deportivas que realicemos, como puede ser correr, ciclismo o natación. Entre estas aplicaciones se encuentran Nike Run Club, Runtastic (*Figura 4*), Strava (*Figura 5*) o Sports Tracker.

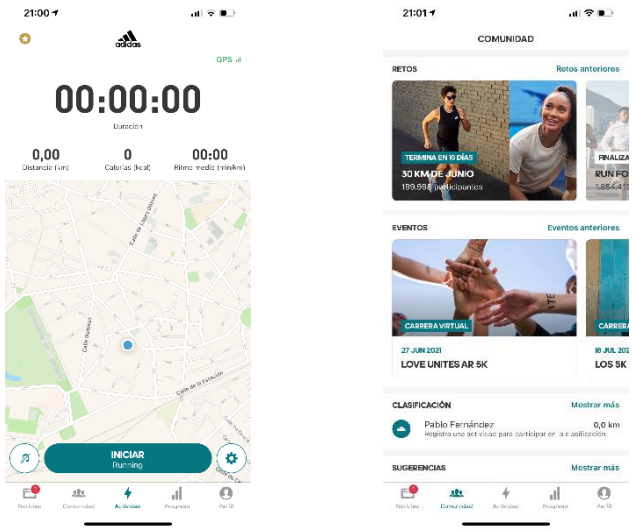

Figura 4.- Interfaz de la aplicación Runtastic.

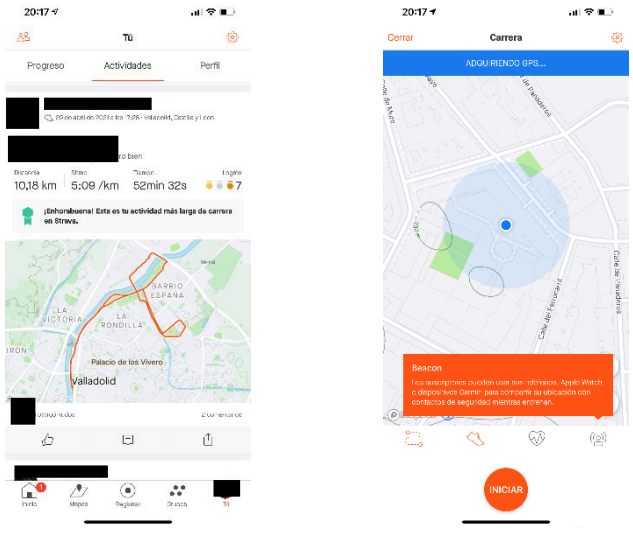

Figura 5.- Interfaz de la aplicación Strava.

2.2.2. Aplicaciones de seguimiento general de la salud

Se puede considerar este tipo de aplicaciones un derivado del grupo anterior, ya que cuentan con seguimiento de las actividades deportivas. No obstante, no se centran en ello y, además,

permiten el seguimiento de otros aspectos, como la cantidad y calidad del sueño, la cantidad de agua ingerida, la alimentación, el nivel de estrés, el ritmo cardíaco, la respiración… Cuentan con los datos de gran cantidad de usuarios, con los que permiten mostrar ciertos estadísticos de utilidad para los usuarios. Algunos ejemplos de este tipo de aplicaciones son Zepp (*Figura 6*), Salud (*Figura 7*), Google Fit, Mi Fit, Huawei Health o Samsung Health.

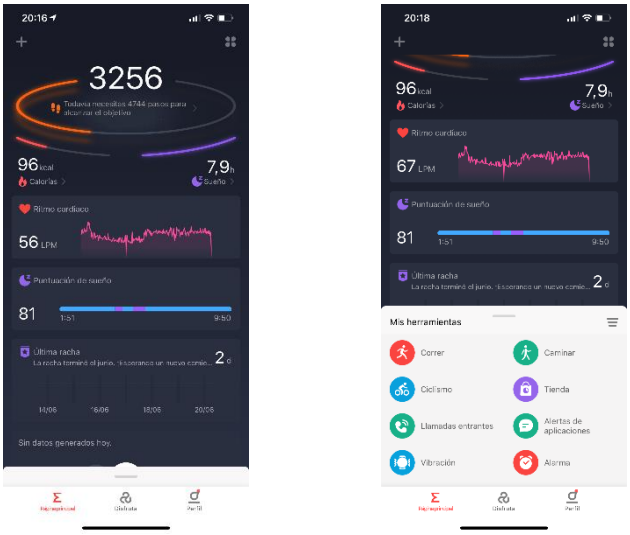

Figura 6.- Interfaz de la aplicación Zepp.

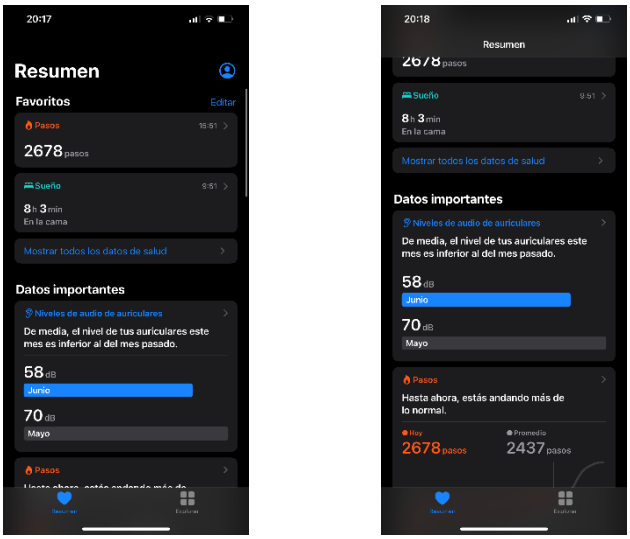

Figura 7.- Interfaz de la aplicación Salud.

2.2.3. Aplicaciones para la mejora de la salud

Aplicaciones que de manera proactiva (las anteriores son pasivas) ayudan a mejorar distintos aspectos de salud (como un entrenador personal dentro del teléfono), como mejorar el sueño o perder peso. Muestran información sobre cómo conseguir los objetivos deseados. Algunos ejemplos de este tipo de aplicaciones son Fabulous (*Figura 8*) o Healthy Habits.

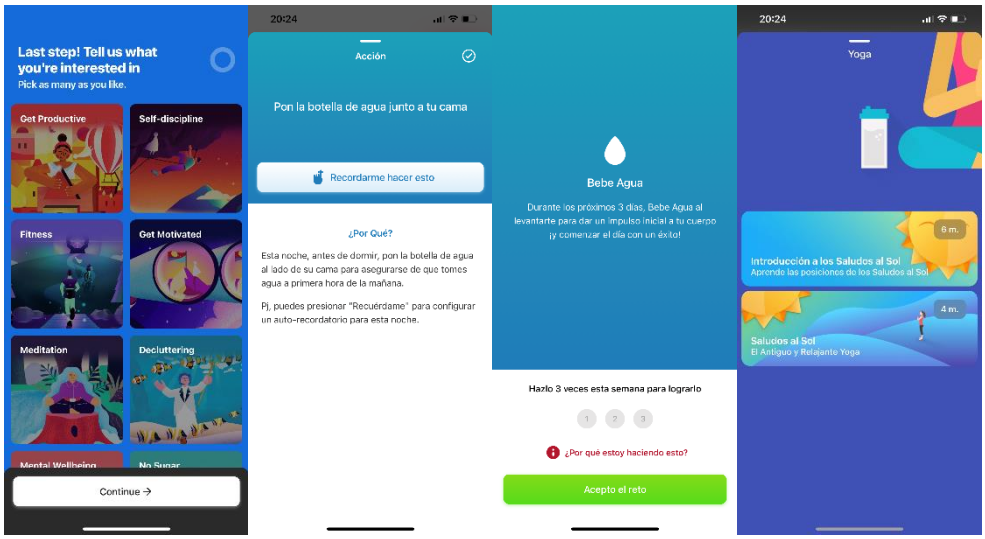

Figura 8.- Interfaz de la aplicación Fabulous.

La aplicación que se va a realizar pertenecerá a ese tercer grupo de aplicaciones. Son aplicaciones que ofrecen soluciones genéricas a todos los usuarios, la posibilidad de adecuación al usuario que ofrecen no es muy alta. La principal diferencia de la aplicación a desarrollar con las aplicaciones ya existentes será la presencia de un *chatbot* que permita comprender mejor las necesidades del usuario para ofrecer un servicio mucho más personalizado y ayudarle de una manera mucho más adaptada a alcanzar su máximo potencial.

3. Análisis del problema

Este capítulo contiene la descripción del proceso de análisis realizado para el desarrollo de la aplicación. Este proceso comienza con la especificación de los requisitos del sistema, analizando: los tipos de usuario que van a utilizar la aplicación y sus características, las condiciones mínimas de hardware y software que permitan utilizar correctamente la aplicación y los requisitos funcionales y no funcionales de la aplicación.

A continuación, se desarrollarán todos los casos de uso de la aplicación, incluyendo: su identificador, una breve descripción, actores que lo realizan, precondiciones, escenario principal de éxito, postcondiciones y posibles alternativas.

Finalmente, se mostrará el prototipado de la interfaz de usuario, que no deja de ser un aspecto que definirá el desarrollo futuro de la aplicación y, es por ello una parte importante del análisis del problema.

3.1. Especificaciones de los requisitos del sistema

El objetivo de esta sección es describir los diferentes requisitos que han sido recogidos en las reuniones del proyecto en la empresa INGAGE, en la que participaron Patrice Séjalon (tutor en la empresa) y Stéphane Tetart (experto naturópata). Estos requisitos estarán adecuados para la comprensión de todos los miembros del equipo.

La aplicación *Full Power* está pensada para proporcionar al usuario una serie de consejos con el objetivo de alcanzar su máximo potencial, evitando problemas que en un futuro le provoquen la necesidad de acudir al médico, gracias a un seguimiento personalizado realizado por un *chatbot* que detecta qué aspectos de salud se pueden mejorar y unos documentos informativos con todo lo necesario para averiguar más sobre dichos ámbitos.

De manera completa, la aplicación se configura con una arquitectura en la que se diferencia la parte del cliente (*frontend*) y la parte del servidor (*backend*). Esto se explicará de manera más detallada en la *Sección 4.2*. Debido a que el ámbito de este TFG comprende la parte correspondiente al *frontend* de la aplicación completa, en la presente sección se centrará el análisis de la parte del cliente. Utilizaré el término sistema o aplicación para referirme de manera más sencilla a la parte del cliente a partir de ahora.

Estos requisitos estarán recogidos de acuerdo con las prácticas recomendadas de IEEE [13].

3.1.1. Tipos de usurario

Existirán dos tipos de usuario en este sistema:

- Básicos: tras haberse registrado en la aplicación, usuarios que pueden utilizar las funciones básicas de la aplicación: el *chatbot* y consultar las diferentes secciones de documentos de salud.
- Administradores: usuarios especiales que, además de poder realizar las funciones básicas, son los encargados de gestionar los documentos disponibles en la aplicación.

3.1.2. Características del usuario

El sistema está pensado para ser utilizado por cualquier tipo de usuario, independientemente de sus conocimientos previos, lo que permitirá que la aplicación pueda ayudar a un mayor número de personas.

3.1.3. Hardware y Software

Se pretende que la aplicación pueda ser utilizada tanto en dispositivos móviles o tabletas como en portátiles u ordenadores de sobremesa. Se podrá acceder a la aplicación mediante el navegador o bien, si contamos con un dispositivo con el sistema operativo de Android, a través del fichero `.apk`, que puede ser instalado. Será necesario que los dispositivos tengan una tarjeta de red con la que conectarse a internet para poder utilizarla correctamente.

La aplicación funcionará correctamente en las últimas versiones de los navegadores Google Chrome, Microsoft Edge, Firefox y Safari. En el caso de Android, funcionará correctamente a partir de la versión de Android 5.0 Lollipop, que se corresponde con un nivel de API 21 y el 94.1% de los dispositivos Android a nivel global tienen esa versión o superior [14].

3.1.4. Requisitos

A la hora de enumerar los requisitos será importante diferenciar entre los Funcionales (*Functional Requirement*, FRQ) y los No Funcionales (*Non Functional Requirement*, NFRQ). Los requisitos funcionales son aquellos que expresan una funcionalidad del sistema, mientras que los requisitos no funcionales son características requeridas que señalan una restricción del sistema. Todos estos requisitos han sido redactados siguiendo la categorización FURPS+ [15], que se compone de las siguientes categorías:

- Funcionalidad (*Functionality*): aquello que la aplicación debería realizar. Se corresponde con los requisitos funcionales.
- El resto de las siglas se corresponden con los requisitos no funcionales:
    - Usabilidad (*Usability*): atributos relacionados con la interacción del usuario con la aplicación, la apariencia de la aplicación o la capacidad de respuesta.
    - Fiabilidad (*Reliability*): requisitos sobre la robustez o fiabilidad de la aplicación.
    - Rendimiento (*Performance*): para describir cómo de eficiente debería ser la aplicación en términos de velocidad y consumo de recursos.
    - Soporte (*Supportability*): requisitos instalación y configuración del sistema, también acerca de la sencillez para administrar la aplicación, testear el código y cómo de flexible es la aplicación.
    - El signo + se refiere a nuevas categorías de requisitos que, normalmente, serán restricciones. En estas categorías nos encontraremos:
        - Restricciones de diseño.
        - Requisitos de implementación.
        - Requisitos de interfaz.
        - Requisitos físicos.

3.1.4.1. Requisitos funcionales

FRQ001. El sistema deberá comprobar si es la primera vez que se accede a la aplicación en el dispositivo de utilización, caso en que se mostrará una presentación sobre la misma.

FRQ002. El sistema deberá permitir el registro de nuevos usuarios. La información necesaria para su registro será el nombre de usuario, la contraseña y la repetición de la contraseña. Esta información deberá ser enviada al *backend* para que pueda almacenarla.

FRQ003. El sistema deberá impedir el registro de más de un usuario con el mismo nombre de usuario, realizando una indicación sobre este suceso. Además, deberá impedir la utilización de ciertos caracteres y comprobar que tenga una longitud determinada.

FRQ004. El sistema deberá comprobar que la contraseña coincida con la repetición.

FRQ005. El sistema deberá permitir a los usuarios iniciar sesión, para lo que requerirá el nombre de usuario y la contraseña, que serán comprobados en el *backend*. También deberá permitir el cierre de sesión.

FRQ006. El sistema deberá impedir a usuarios que no hayan iniciado sesión utilizar las funciones de la aplicación (excepto registrarse e iniciar sesión).

FRQ007. El sistema deberá conseguir del *backend* si se ha aceptado la exención de responsabilidad y, en caso negativo, mostrar un aviso en el momento en el que el usuario inicie sesión. Sin su aceptación no se podrá utilizar la aplicación. Una vez aceptado, se comunicará al *backend* para que actualice la información del usuario.

FRQ008. El sistema deberá recibir del *backend* si es la primera vez que un usuario accede a su cuenta. En caso afirmativo, se redirigirá al usuario automáticamente a la zona de chat para responder al cuestionario inicial.

FRQ009. El sistema deberá permitir a los usuarios registrados consultar los diferentes documentos disponibles en la aplicación, que obtendrá desde el *backend.*

FRQ010. El sistema deberá permitir a los usuarios registrados enviar mensajes al *chatbot*, que serán procesados en el *backend* y, finalmente, respondidos por este.

FRQ011. El sistema deberá detectar el idioma del usuario y mostrar la aplicación en dicho idioma si está disponible. Además, permitirá el cambio de este si el usuario lo desea.

3.1.4.2. Requisitos no funcionales

NFRQ001. El sistema deberá ser sencillo de utilizar e intuitivo, para que usuarios de cualquier nivel de conocimientos puedan dominar su uso sin necesidad de ayuda externa. Contendrá ciertas pistas que permitan aclarar sobre ciertas funcionalidades a los usuarios.

NFRQ002. El sistema deberá funcionar correctamente en las últimas versiones de los navegadores Google Chrome, Microsoft Edge, Mozilla Firefox y Safari. Además, deberá funcionar tanto en versión web como en versión aplicación de Android.

NFRQ003. Los colores seleccionados en la interfaz de usuario deberán ser afines a los colores de la empresa INGAGE.

NFRQ004. La interfaz de usuario deberá adaptarse a cualquier tipo de pantalla (escritorio, móvil o tableta).

NFRQ005. El sistema deberá ser capaz de avisar a los usuarios de que se ha producido un fallo al utilizar la API.

NFRQ006. El sistema solo funcionará en dispositivos con conexión a internet.

NFRQ007. El sistema permitirá una navegación fluida por toda la aplicación.

3.2. Casos de uso

Identificador	DarseDeAlta	
Descripción	Un usuario crea una cuenta de usuario en la aplicación.	
Actor	Usuario no registrado.	
Precondición	Ninguna.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El usuario no registrado hace clic en la sección "¿No tienes una cuenta? Haga clic aquí para registrarse".
	2	El sistema solicita un nombre de usuario y la contraseña (con su confirmación).
	3	El usuario introduce los datos solicitados siguiendo las reglas indicadas.
	4	El sistema manda los datos al *backend*.
	5	En el *backend* se comprueba que no existe ningún usuario registrado con el nombre de usuario a crear, caso en el que se creará la cuenta del usuario.
	6	El sistema conduce al usuario a la página de inicio de sesión de la aplicación.
Postcondición	Un nuevo usuario se crea en la aplicación.	
Alternativas	**Paso**	**Acción**
	3	El usuario no cumple con las reglas al introducir sus datos. Se informa de este hecho al usuario y se vuelve al paso 2.
	5	El *backend* encuentra un nombre de usuario idéntico al que se desea crear. Se informa de este hecho al usuario y se vuelve al paso 2.

Identificador	IniciarSesion	
Descripción	Un usuario puede iniciar sesión con su nombre de usuario y contraseña.	
Actor	Usuario básico y administrador.	
Precondición	Ninguna.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El sistema solicita al usuario su nombre de usuario y su contraseña.
	2	El usuario introduce los datos solicitados.
	3	El sistema envía los datos al *backend.*
	4	El *backend* verifica que los datos introducidos son correctos.
	5	El *backend* responde con la confirmación del acceso y el token de sesión.
	6	El sistema conduce al usuario al menú principal de la aplicación.
Postcondición	El usuario ha iniciado sesión.	
Alternativas	**Paso**	**Acción**
	4	El *backend* no verifica los datos introducidos porque no son correctos. Notificará de este hecho al usuario. Se vuelve al paso 1.

Identificador	CerrarSesion	
Descripción	Un usuario puede cerrar la sesión en la aplicación.	
Actor	Usuario básico y administrador.	
Precondición	El usuario ha iniciado sesión en el sistema.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El usuario accede a la sección “Perfil” de la aplicación.
	2	El usuario selecciona “Cerrar Sesión”.
	3	El sistema elimina el token, cerrando la sesión del usuario y navegando a la página de inicio de sesión.
Postcondición	El usuario ha cerrado la sesión.	

Identificador	ConsultarDocumentos	
Descripción	El usuario puede consultar los documentos disponibles tanto de la sección de salud como de la sección de vitaminas y minerales.	
Actor	Usuario básico y administrador.	
Precondición	El usuario ha iniciado sesión en el sistema.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El usuario accede a la sección “Salud” o “Vits&Mins” de la aplicación.
	2	El sistema solicita al *backend* la lista de documentos disponibles sobre salud/vitaminas en el idioma actual de la aplicación.
	3	El *backend* envía la lista de documentos disponibles.
	4	El sistema muestra la lista de documentos disponibles.
	5	El usuario selecciona cualquier elemento de la lista.
	6	El sistema solicita al *backend* el documento seleccionado por el usuario.
	7	El *backend* envía el documento seleccionado.
	8	El sistema muestra al usuario el documento solicitado.
Postcondición	Ninguna.	

Identificador	AceptarDisclaimer	
Descripción	El usuario debe aceptar la exención de responsabilidad en el caso de que quiera utilizar la aplicación, en caso contrario no podrá utilizarla.	
Actor	Usuario básico y administrador.	
Precondición	El usuario ha iniciado sesión en el sistema.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El sistema muestra al usuario la exención de responsabilidad.
	2	El usuario acepta la exención de responsabilidad.
	3	El sistema envía al *backend* la aceptación de la exención.
	4	El *backend* almacena esa información para el usuario que acepta.
	5	El sistema permite la utilización de la aplicación al usuario.
Postcondición	El usuario ha aceptado la exención de responsabilidad.	
Alternativas	**Paso**	**Acción**
	2	El usuario no acepta la exención de responsabilidad. En este caso no puede utilizar la aplicación. Se vuelve al paso 1.

Identificador	CompletarPrimeraEncuesta	
Descripción	El usuario puede completar la encuesta inicial tras iniciar sesión por primera vez.	
Actor	Usuario básico y administrador.	
Precondición	El usuario ha iniciado sesión por primera vez en el sistema.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El usuario accede a la sección “Chat” de la aplicación.
	2	El sistema solicita al *backend* la primera pregunta del cuestionario.
	3	El *backend* envía la primera pregunta del cuestionario.
	4	El sistema muestra la primera pregunta del cuestionario y la zona donde debe responder el usuario.
	5	El usuario introduce su respuesta a la primera pregunta.
	6	El sistema envía la respuesta al *backend*.
	7	El *backend* almacena la respuesta del usuario.
	8	El *backend* envía la siguiente pregunta.
	9	El sistema muestra la siguiente pregunta y la zona donde debe contestar el usuario.
	10	El usuario introduce su respuesta.
	11	Se repiten los pasos 6 a 10 hasta llegar a la última pregunta.
	12	El sistema envía la respuesta a la última pregunta al *backend*.
	13	El *backend* almacena la respuesta del usuario y marca como completada la primera encuesta.
	14	El *backend* envía al sistema la indicación de finalización de la encuesta, así como de la información acerca de sus respuestas, indicando sus problemas y la gravedad de estos.
	15	El sistema muestra al usuario la indicación de finalización de la encuesta, así como sus problemas, su gravedad y la posibilidad de navegar a la sección de la aplicación en la que se habla acerca de su problema.
	16	El sistema programa una notificación en función de la gravedad del problema para recordar al usuario en el futuro ciertos consejos para mejorar su salud.
Postcondición	El usuario tiene un atributo que indica que ha completado la primera encuesta.	
Alternativas	**Paso**	**Acción**
	2-10	El usuario abandona la sección de “Chat”. Las sucesivas veces que entre de nuevo en dicha sección, volverá al paso 1.
	15	El usuario no está utilizando un dispositivo Android. Se ignora este paso.

Identificador	ContestarPreguntaNotificacion	
Descripción	El usuario recibe una notificación con una pregunta relacionada con sus problemas de salud. Al abrirla, podrá contestar a dicha pregunta y recibirá un consejo tras su respuesta.	
Actor	Usuario básico y administrador.	
Precondición	El usuario ha realizado la primera encuesta y tiene algún problema de salud y ha iniciado sesión en la aplicación.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El usuario abre la notificación.
	2	El sistema programa una notificación en función de la gravedad del problema para recordar al usuario en el futuro ciertos consejos para mejorar su salud.
	3	El sistema muestra al usuario la sección de “Chat” con la pregunta de la notificación y el campo para responder.
	4	El usuario introduce su respuesta.
	5	El sistema envía la respuesta al *backend.*
	6	El *backend* envía al sistema el consejo adecuado en función de la respuesta del usuario.
	7	El sistema muestra al usuario el consejo recibido.
Postcondición	Se ha programado una notificación.	
Alternativas	**Paso**	**Acción**
	3-4	El usuario abandona la sección de “Chat”. Se finaliza el caso de uso.

Identificador	CambiarIdioma	
Descripción	El usuario puede cambiar el idioma de la aplicación si el seleccionado por el sistema no es el adecuado.	
Actor	Usuario básico y administrador.	
Precondición	El usuario ha iniciado sesión en el sistema.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El usuario accede a la sección “Perfil” de la aplicación.
	2	El usuario selecciona el idioma deseado entre las opciones disponibles.
	3	El sistema cambia de idioma.
Postcondición	El sistema ha cambiado de idioma.	

Identificador	CrearDocumentos	
Descripción	El administrador puede crear documentos de salud o vitaminas en el idioma que desee.	
Actor	Administrador.	
Precondición	El administrador ha iniciado sesión en el sistema.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El administrador accede a la sección de gestión de documentos.
	2	El sistema solicita al *backend* la lista de documentos disponibles sobre salud/vitaminas de cualquier idioma de la aplicación.
	3	El *backend* envía la lista de documentos disponibles.
	4	El sistema muestra la lista de documentos disponibles.
	5	El administrador selecciona la opción para crear un nuevo documento.
	6	El sistema solicita al administrador el tipo de documento, el nombre del documento, el título del documento y el idioma del documento.
	7	El administrador introduce los datos solicitados (tipo de documento: salud).
	8	El sistema comprueba que no existe ningún documento con el mismo nombre e idioma entre los existentes.
	9	El sistema solicita al administrador el contenido de la sección “Entender” del documento.
	10	El administrador introduce el contenido de la sección “Entender” del documento.
	11	El sistema solicita al administrador el contenido de la sección “Actuar” del documento.
	12	El administrador introduce el contenido de la sección “Entender” del documento.
	13	El sistema envía el documento al *backend.*
	14	El *backend* almacena el documento de salud en la base de datos.
	15	El sistema redirige al administrador a la sección de gestión de documentos.
Postcondición	Se ha creado un nuevo documento.	
Alternativas	**Paso**	**Acción**
	7	El administrador selecciona el tipo: vitaminas. En este caso es el mismo flujo excepto los pasos 11 y 12, que se saltan por no contar con dos secciones.
	1-12	El administrador abandona la creación del documento. Se finaliza el caso de uso sin éxito.

Identificador	ModificarDocumentos	
Descripción	El administrador puede modificar los documentos de salud o vitaminas disponibles en el *backend.*	
Actor	Administrador.	
Precondición	El administrador ha iniciado sesión en el sistema y hay documentos en el servidor.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El administrador accede a la sección “Salud” o “Vits&Mins” de la aplicación.
	2	El sistema solicita al *backend* la lista de documentos disponibles sobre salud/vitaminas en el idioma actual de la aplicación.
	3	El *backend* envía la lista de documentos disponibles.
	4	El sistema muestra la lista de documentos disponibles.
	5	El administrador selecciona cualquier elemento de la lista.
	6	El sistema solicita al *backend* el documento seleccionado por el administrador.
	7	El *backend* envía el documento seleccionado.
	8	El sistema muestra al administrador el documento solicitado.
	9	El administrador modifica el documento solicitado.
	10	El sistema envía al *backend* el documento modificado.
	11	El backend sobreescribe el documento modificado.
Postcondición	Se ha modificado un documento.	
Alternativas	**Paso**	**Acción**
	1-9	El administrador abandona la modificación. Finaliza el caso de uso sin éxito.

Identificador	HablarConBot	
Descripción	El usuario puede hablar con el *chatbot* y hacerle preguntas.	
Actor	Usuario básico y administrador.	
Precondición	El usuario ha iniciado sesión en el sistema.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El usuario se dirige a la sección de “Chat” de la aplicación.
	2	El sistema solicita al *backend* un mensaje de bienvenida.
	3	El *backend* envía al sistema el mensaje de bienvenida.
	4	El sistema muestra al usuario dicho mensaje.
	5	El usuario introduce en el campo habilitado el mensaje para el *chatbot*.
	6	El sistema envía al *backend* el mensaje para el *chatbot*.
	7	El *backend* responde al mensaje recibido.
	8	El sistema muestra al usuario la respuesta del *chatbot*.
Postcondición	Ninguna.	

Identificador	EliminarDocumentos	
Descripción	El administrador puede eliminar los documentos de salud o vitaminas disponibles en el *backend.*	
Actor	Administrador.	
Precondición	El administrador ha iniciado sesión en el sistema y hay documentos en el servidor.	
Escenario principal de éxito	**Paso**	**Acción**
	1	El administrador accede a la sección de gestión de documentos.
	2	El sistema solicita al *backend* la lista de documentos disponibles sobre salud/vitaminas de cualquier idioma de la aplicación.
	3	El *backend* envía la lista de documentos disponibles.
	4	El sistema muestra la lista de documentos disponibles.
	5	El administrador selecciona la opción para eliminar documentos.
	6	El sistema solicita al administrador los documentos que se desea eliminar.
	7	El administrador selecciona los documentos a eliminar.
	8	El sistema envía los documentos seleccionados al *backend* para su eliminación.
	9	El *backend* elimina los documentos seleccionados.
	10	El sistema redirige al administrador a la sección de gestión de documentos.
Postcondición	Se han eliminado uno o varios documentos.	
Alternativas	**Paso**	**Acción**
	1-7	El administrador abandona la eliminación de documentos. Se finaliza el caso de uso sin éxito.

3.3. Prototipado de la Interfaz de Usuario

Para la creación de un prototipo que pudiera cumplir con los requisitos de la aplicación se recurrió a la herramienta Figma [16]. En esta se permite crear con facilidad los distintos diseños de cada ventana y las interacciones que va a haber entre ellas, como puede ser arrastrar la pantalla o pulsar un botón para navegar a otra página. Otra utilidad que aporta Figma es la posibilidad de conseguir el código CSS necesario para cada objeto creado en la interfaz, así como exportar directamente imágenes vectoriales que han sido utilizadas posteriormente en el proyecto. También permite la colaboración con más miembros de la organización para que puedan escribir comentarios aportando sugerencias e, incluso, cambiando ellos mismos la interfaz.

Como la aplicación debía tener un comportamiento diferente en función del momento temporal, se diseñó un camino guiado para las 3 situaciones principales que se valoraron:

- La primera vez que un usuario accede a la aplicación.
- El resto de las veces que un usuario accede a la aplicación.
- Acceso a la aplicación mediante notificaciones.

En las siguientes figuras (desde la *Figura 9* a la *Figura 17*) voy a mostrar los prototipos realizados en la aplicación.

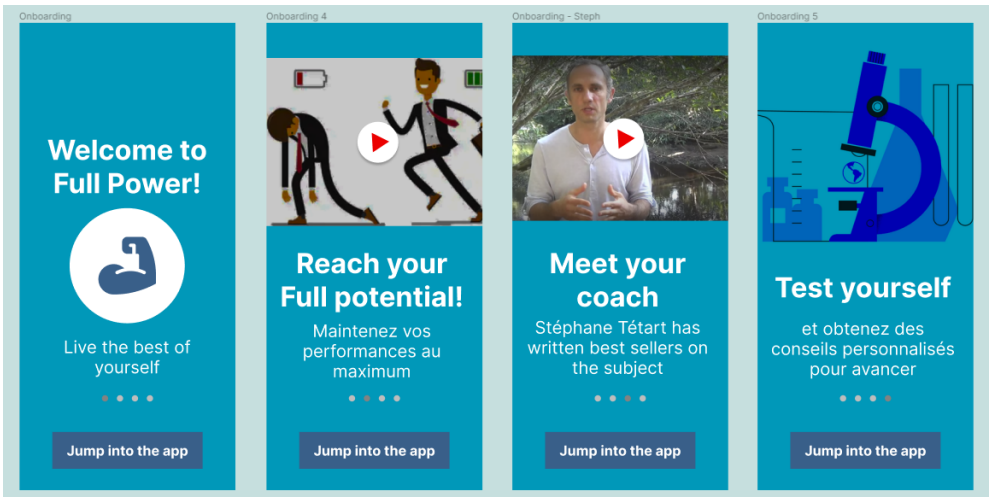

Figura 9.- Prototipos del onboarding.

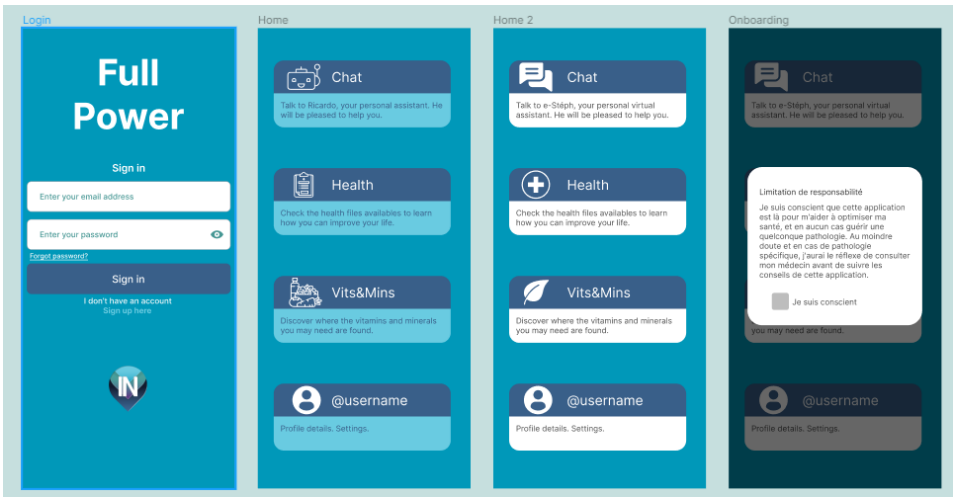

Figura 10.- Prototipos del inicio de sesión (izquierda), del menú principal (centro) y de la exención de responsabilidad (derecha).

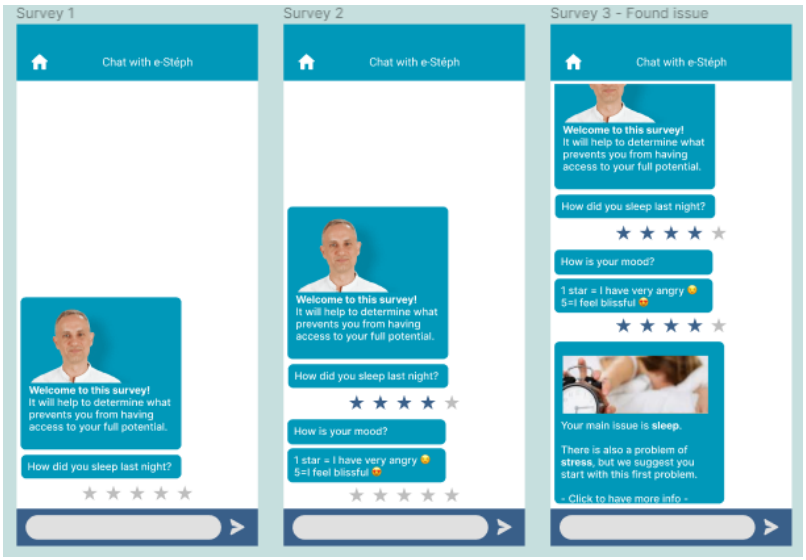

Figura 11.- Prototipos de la encuesta principal.

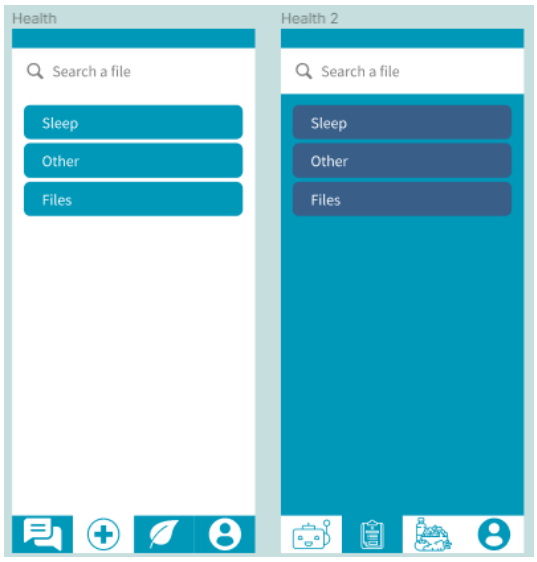

Figura 12.- Prototipos de la sección de salud.

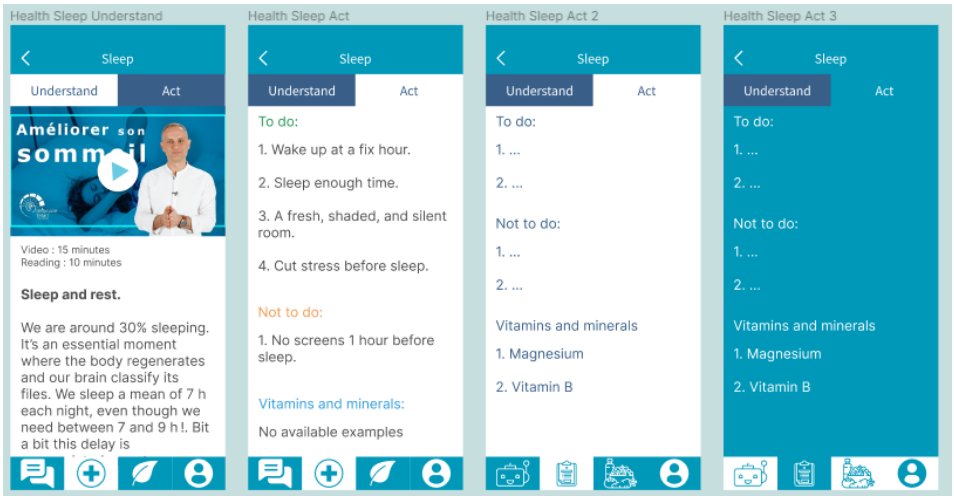

Figura 13.- Prototipos de un documento de salud.

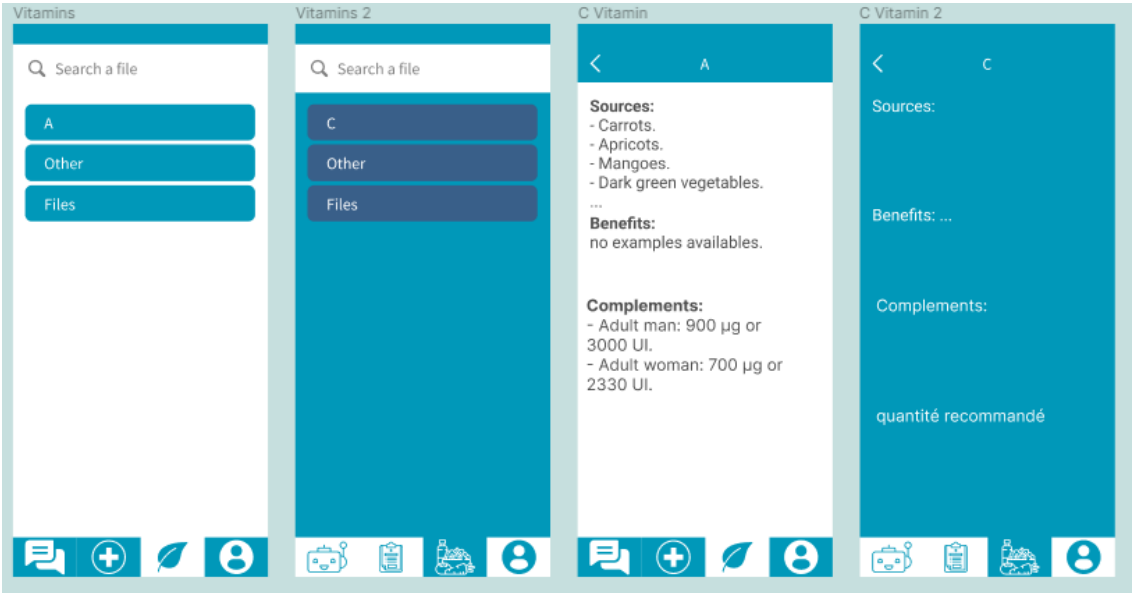

Figura 14.- Prototipos de la sección de vitaminas y un documento de vitaminas.

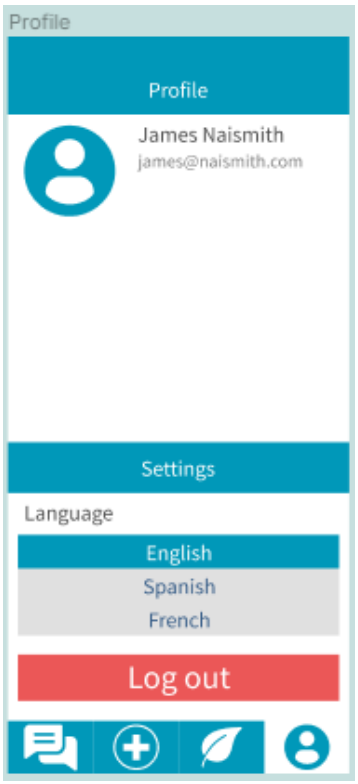

Figura 15.- Prototipo de la sección de perfil.

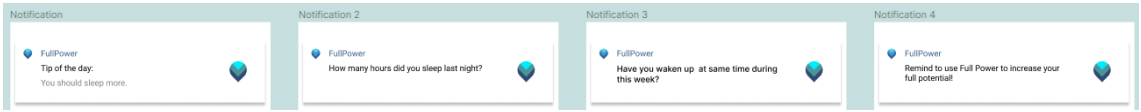

Figura 16.- Prototipos de notificaciones.

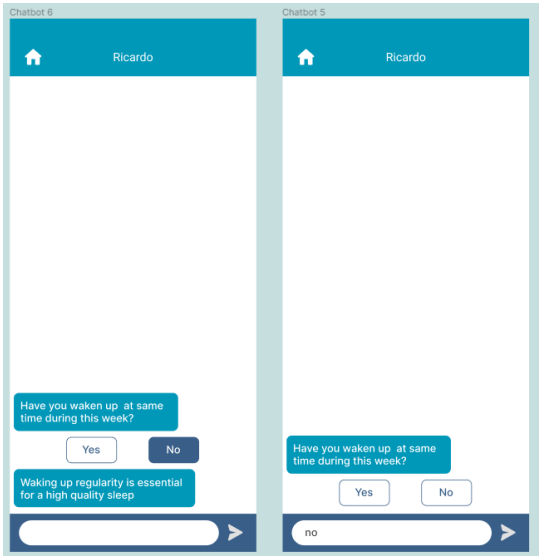

Figura 17.- Prototipos de interacción con el chatbot tras notificación.

4. Desarrollo del proyecto

Este capítulo contiene todo lo relacionado con el desarrollo del proyecto una vez hemos analizado de manera completa el problema que se va a abordar.

Comenzando con un análisis de las diferentes tecnologías que se han utilizado a lo largo del proyecto, describiendo brevemente en qué consisten y destacando sus funcionalidades más importantes.

A continuación, se realizará el análisis de la arquitectura del sistema desde el punto de vista de las conexiones entre la parte cliente y servidor hasta centrarse más concretamente en la arquitectura propia del *frontend*.

Posteriormente, se describirá la configuración del entorno de trabajo en sus diferentes fases: desarrollo, testeo y despliegue. También, la estructura de ficheros del *frontend*, en la que se explicará de forma breve, la función de cada uno de ellos.

Por último, explicaré los aspectos concretos relevantes de la implementación de la aplicación, como son las comunicaciones con la API, el editor de texto, las traducciones y las notificaciones.

4.1. Tecnologías utilizadas

4.1.1. Angular

La parte de *frontend* está desarrollada en Angular, un *framework* desarrollado por Google que permite la creación de aplicaciones web de una sola página (SPA, *Single Page Application*), este tipo de aplicaciones permite cargar todos los recursos de la propia aplicación en la solicitud inicial, sin necesidad de recargas posteriores. Esto permite, además, evitar llamadas al *backend* para generar el propio contenido de la aplicación.

En un inicio se barajaron varias opciones de tecnologías para realizar la aplicación, en una primera aproximación se eligió Python con Flask, no obstante, no permite una separación de *frontend* y *backend*, por lo que se descartó. Posteriormente de entre los distintos *frameworks* disponibles más enfocados a la parte de *frontend*, React, Vue y Angular, se optó por Angular por la posesión de mayor conocimiento de dicho *framework* por parte de la empresa.

Angular se programa mediante TypeScript, un lenguaje de código abierto basado en JavaScript, con la diferencia de permitir la declaración de tipos. Este código al ser compilado es convertido a JavaScript para hacer posible su utilización [17].

Dentro de Angular nos encontramos varios conceptos fundamentales, que componen su arquitectura:

- **Módulos:** permiten organizar las inyecciones de código y la compilación. Debe existir un módulo raíz que permita una organización global de todo el proyecto. Otro módulo importante en las aplicaciones de Angular es el que gestiona las rutas de la aplicación.
- **Componentes:** parte elemental de Angular. Permite la creación de las diferentes vistas de la aplicación mediante la configuración de cuatro ficheros asociados al componente:
  - `.ts`: fichero TypeScript que define el componente como una clase, se definen las propiedades y métodos del componente.
  - `.html`: fichero HTML que permite definir la plantilla del componente.
  - `.css`: fichero CSS en el que se definen los estilos del componente definidos en la plantilla HTML.

    - `.spec.ts`: fichero TypeScript que permite definir los test unitarios asociados al componente, no es indispensable para la creación de un proyecto, pero sí recomendable para permitir la integración continua.
- **Servicios:** utilizados por los componentes para una funcionalidad específica. Al igual que los componentes se definen en un fichero TypeScript como clases, también cuentan con un fichero para los test unitarios. No se utilizan para las vistas, por lo que no requieren de plantilla y estilos. Se diferencian de los componentes en que pueden ser reutilizados por varios componentes de manera simultánea, por ejemplo, para realizar métodos comunes entre ellos, como pueden ser llamadas al servidor. También permiten comunicar información entre los componentes que, de otra manera, no podrían enviarse.
- **Tuberías:** utilizados en las plantillas para transformar los datos antes de ser mostrados, como traducir el texto deseado o cambiar de minúsculas a mayúsculas. Se definen como clases en un fichero TypeScript, y cuentan con un segundo fichero para los test unitarios.
- **Interceptores:** utilizados para interceptar mensajes HTTP antes de ser enviados y, por ejemplo, colocar una cabecera específica que indica que un usuario ha iniciado sesión. Se definen con un fichero TypeScript y un segundo fichero para los test unitarios.
- **Guardas:** utilizados para proteger ciertas rutas y que solo puedan ser accedidas por ciertos usuarios (diferenciando registrados, no registrados y administradores).

La utilización de Angular es mucho menos compleja mediante la utilización de Angular CLI (*Command Line Interface,* Interfaz de Línea de Comandos). Esta interfaz permite la utilización de comandos para diversas funciones como la creación de módulos componentes, servicios, tuberías, etc. O también la compilación del proyecto tanto para la versión de desarrollo, como la versión de producción, u otras utilidades como la ejecución de los test y comprobaciones de la calidad del código.

4.1.2. Apache Cordova e IONIC Capacitor

Además de Angular, para tener la posibilidad de transformar la aplicación web en una aplicación para Android, se han utilizado las librerías Apache Cordova [18] e IONIC Capacitor [19]. Existen diferencias entre ambos: Cordova es mucho más sencillo de utilizar que Capacitor y, permite una conversión de la aplicación rápida y sencilla. Por el contrario, IONIC ofrece mucha mayor capacidad de personalización, ya que, mientras que Cordova compila la aplicación directamente al fichero `.apk` gracias a un único fichero de configuración, necesario para instalar aplicaciones en Android, IONIC convierte el proyecto de Angular en un proyecto de Android que puede ser trabajado desde el entorno de desarrollo de Android, Android Studio. El cambio de Cordova a Capacitor no es complicado, ya que las librerías son compatibles. Por ello, aunque se comenzó el proyecto utilizando Cordova, se terminó optando por Capacitor.

4.1.3. Bootstrap

Bootstrap es un *framework* de CSS que permite dar estilos a una plantilla HTML mediante clases predefinidas [20]. El uso de este *framework* no limita la capacidad de introducir otros estilos personalizados.

Se ha utilizado principalmente por su gran flexibilidad para posicionar los diferentes componentes de la plantilla en la interfaz que, además, son adaptables a cualquier tipo de pantalla de una manera rápida.

4.1.4. CKEditor

CKEditor es un *framework* WYSIWYG (What you see is what you get, aquello que ves es lo que obtienes), es decir, es un editor de texto dentro de la aplicación [21].

Se ha utilizado por su gran facilidad de implementación en la aplicación y la gran capacidad de personalización que ofrece. Desde librerías con un editor por defecto hasta el proyecto completo para ser editado y, posteriormente, utilizado en el proyecto de Angular.

4.1.5. Jasmine

Jasmine es el *framework* utilizado para testear el código. Es el proporcionado por defecto por el proyecto de Angular tras la creación de este.

Permite la creación y ejecución de los test, que se utilizan para analizar el comportamiento adecuado del código desarrollado y su cobertura.

4.1.6. GitHub y Git

GitHub es un servicio online de repositorios de código que permite el trabajo colaborativo en la elaboración de proyectos.

Git es un sistema de control de versiones de proyectos que, en combinación con GitHub, es una herramienta muy potente para la gestión de proyectos. Es posible crear un repositorio en Git, que será almacenado en GitHub para que todos los miembros de un equipo o, incluso, todo el mundo, pueda colaborar en su desarrollo.

4.1.7. Codecov

Codecov es un servicio de testeo de aplicaciones que permite la integración continua en el desarrollo. Mediante un fichero de configuración habilita la realización automática de todos los test cada vez que se hace una *Merge Request* en GitHub. Además, ofrece un informe similar al que ofrece Jasmine de manera local con los resultados de los propios test y la cobertura del código [22].

4.1.8. Docker

Docker es un *framework* que permite automatizar el despliegue de aplicaciones dentro de contenedores. Facilita el despliegue de las aplicaciones ya que no es necesario buscar un servidor que admita cada tecnología específica que utilizamos, sino que con permitir el lanzamiento de contenedores Docker es más que suficiente [23].

4.1.9. VSCode

Entorno de desarrollo creado por Microsoft. Ofrece gran versatilidad y permite el uso de un alto número de herramientas que ayudan a generar el código [24].

Estas herramientas abarcan desde sugerencias de autocompletado en el código hasta el control de versiones de Git.

4.1.10. Android Studio

Entorno de desarrollo creado por Google para el desarrollo de aplicaciones de Android. Al convertir la web app en una app híbrida, ha sido necesario para la personalización de la aplicación para Android [25].

4.2. Arquitectura del sistema

La aplicación *Full Power* en su totalidad está compuesta por *frontend* (parte del cliente) y *backend* (parte del servidor). Estas dos partes se comunican entre sí mediante una API REST.

Esto permite reducir la carga de trabajo de la parte del servidor, siendo solo necesaria cuando se solicita un recurso desde la parte del cliente.

Estas solicitudes que se realizan a la API son peticiones HTTP asíncronas y los datos que devuelve se encuentran en formato JSON (*JavaScript Object Notation*).

Además, tanto el *frontend* (versión web), como la API y la base de datos (forman en conjunto la totalidad del *backend*) están envueltos, cada uno, en un contenedor Docker, que permite el despliegue de la aplicación de manera sencilla con la configuración de un único fichero, en el caso del *frontend* y, de varios ficheros en el caso del *backend* [5]. Todo el sistema está siendo ejecutado en una Raspberry Pi y, se requiere que exista un contenedor adicional que redirige las peticiones que llegan al *frontend* y al *backend*, también se encarga de redirigir las peticiones del *frontend* al *backend*. Un esquema de la arquitectura lo podemos ver en la *Figura 18*. No nos podemos olvidar que, en el caso de Android, el *frontend* está compilado en un fichero `.apk` cumpliendo con el esquema de la *Figura 19*.

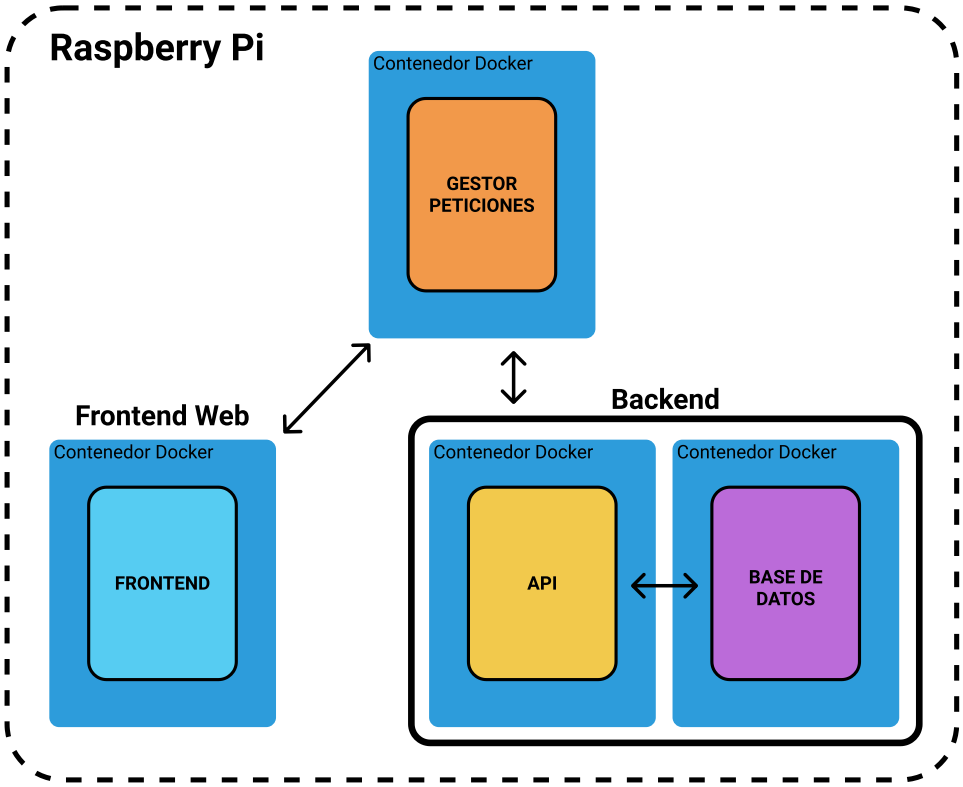

Figura 18.- Arquitectura de la aplicación Full Power para la versión web.

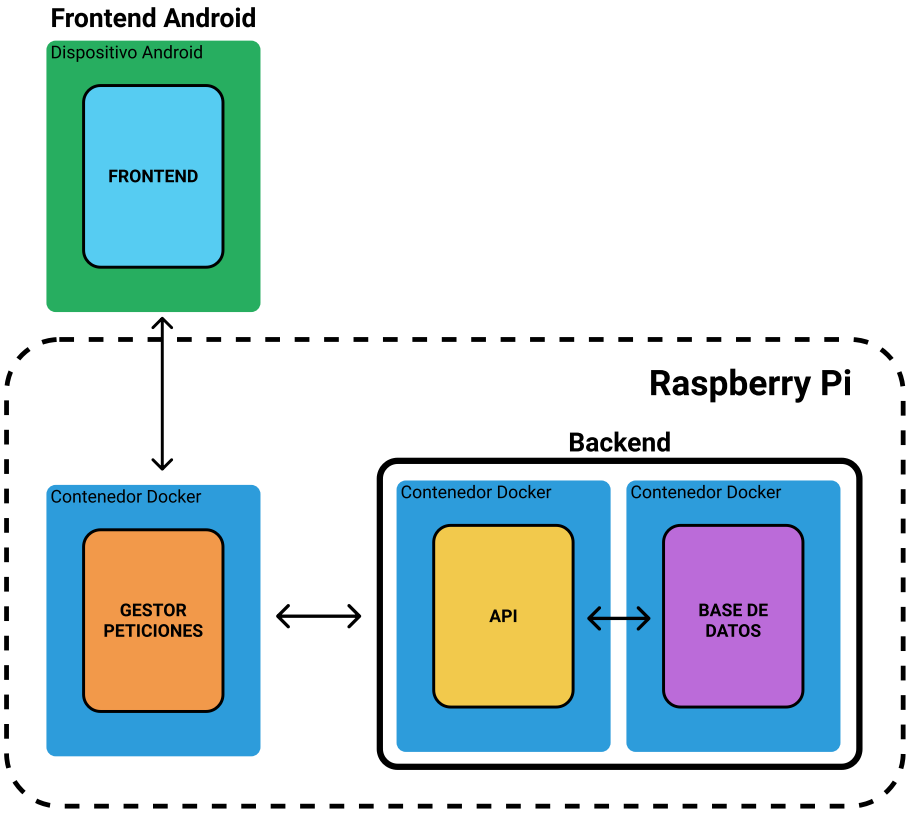

Figura 19.- Arquitectura de la aplicación Full Power para la versión Android.

4.2.1. Rutas

En la aplicación de Angular nos encontramos con las rutas mostradas en la siguiente tabla, a las que un usuario puede navegar, tan solo el *path*, que se escribiría justo después de la URL del *frontend*, en la columna de guarda relacionado, hace referencia a si el acceso a la ruta está restringido de alguna forma. Existen 2 en esta aplicación, el fichero `auth.guard.ts` evita el acceso si un usuario no ha iniciado sesión y, el fichero `admin.guard.ts` evita el acceso si un usuario no es administrador. La configuración de todas las rutas ha sido realizada en el módulo dedicado a las rutas en Angular, en el fichero `app-routing.module.ts`.

Ruta	Descripción	Componentes relacionados	Guarda relacionado

`/`	Se redirige automáticamente a `/login`.	`AppComponent`
		`LoginComponent`
		`OnboardingComponent`
			-

`/login`	Permite iniciar sesión a un usuario registrado.	`AppComponent`
		`LoginComponent`
		`OnboardingComponent`
			-

`/register`	Permite crear una cuenta a un usuario no registrado.	`AppComponent` `RegisterComponent`	-

`/home`	Menú de inicio de la aplicación una vez el usuario ha iniciado sesión. Proporciona una pequeña descripción del resto de partes de la aplicación y	`AppComponent`
		`DisplayComponent`
		`DisclaimerComponent`
		`MenuComponent`
			`AuthGuard`

	permite la navegación a estas.		
/chat	Permite comunicarse con el *chatbot*, realizar preguntas, realizar la encuesta inicial y responder a las preguntas de las notificaciones.	AppComponent ChatbotComponent MenuComponent	AuthGuard
/health	Permite la visualización de los ficheros disponibles en la sección de salud y buscar entre ellos.	AppComponent HealthComponent MenuComponent	AuthGuard
/health/healthFile	Permite la visualización de un fichero de la sección de salud. healthFile hace referencia al fichero que se desea visualizar.	AppComponent HealthFilesComponent MenuComponent	AuthGuard
/vits	Permite la visualización de los ficheros disponibles en la sección de vitaminas y minerales y buscar entre ellos.	AppComponent VitaminsComponent MenuComponent	AuthGuard
/vits/vitFile	Permite la visualización de un fichero de la sección de vitaminas y minerales. vitFile hace referencia al fichero que se desea visualizar.	AppComponent VitaminFilesComponent MenuComponent	AuthGuard
/settings	Permite visualizar el nombre de usuario y cambiar el lenguaje de la aplicación. También se puede cerrar la sesión del usuario.	AppComponent SettingsComponent LogoutComponent MenuComponent	AuthGuard
/files	Disponible solo para administradores. Permite la visualización de un listado de todos los ficheros disponibles, así como de su creación, eliminación y edición (la creación y edición serán llevadas a cabo en /files/edit y en /health/healthFile o /vits/vitFile, respectivamente).	AppComponent FileComponent MenuComponent	AdminGuard
/files/edit	Disponible solo para administradores. Permite la creación de ficheros.	AppComponent FileEditComponent MenuComponent	AdminGuard

4.2.2. Endpoints del *backend*

El *frontend* necesita hacer llamadas al *backend* a través de la API REST para obtener la información deseada. A continuación, se muestra la lista de *endpoints* [5]:

Endpoint	**Métodos**	**Descripción**
`/bot/process-msg`	POST	Se envía un mensaje al bot y se recibe su respuesta. Se requiere enviar '`msg`' o '`question_response`', pero no ambos. En la encuesta inicial, la última respuesta del bot incluye dos cabeceras que permiten conocer los problemas del usuario y su gravedad, como resultado del algoritmo de salud.
`/conversations`	GET, POST	El método GET devuelve una lista de todas las conversaciones con el bot. El método POST crea una nueva conversación asociada a un usuario.
`/conversations/{conversation_id}`	GET, DELETE	El método GET devuelve la conversación con el id solicitado o devuelve 404 si no la encuentra. El método DELETE elimina la conversación del id proporcionado.
`/conversations/user/{user_id}`	GET	Devuelve la lista de conversaciones asociadas al usuario solicitado.
`/files`	GET, POST	El método GET devuelve la lista de los nombres de los documentos escritos en un idioma concreto. Returns a list of the names written in a specific language. El método POST crea un nuevo fichero.
`/files/all`	GET	Lista todos los documentos de la base de datos, agrupados por nombre.
`/files/multi`	GET, POST	El método GET devuelve los documentos solicitados en el idioma marcado en la solicitud o devuelve 404 si no los encuentra. El método POST crea nuevos ficheros.

/files/{name}	GET, PUT, DELETE	El método GET devuelve el documento solicitado por nombre e idioma o devuelve 404 si no lo encuentra. El método PUT actualiza el contenido de un documento dado su nombre e idioma. El método DELETE elimina un documento dado su nombre e idioma.
/files/multiple	DELETE	Elimina el conjunto de documentos utilizando sus nombres y su idioma.
/health-data	GET, POST	El método GET obtiene los resultados de los datos de salud de todos los usuarios. El método POST crea unos resultados de datos de salud.
/health-data/user/{user_id}	GET	Obtiene los resultados de datos de salud de un usuario dado su id.
/health-data/{health_data_id}	GET, DELETE	El método GET obtiene los resultados de datos de salud dado el id de estos datos. El método DELETE elimina los resultados de datos de salud dado su id.
/images/{image_id}	GET, DELETE	El método GET devuelve la imagen del id solicitado o 404 si no la encuentra. El método DELETE elimina la imagen utilizando su id.
/images	GET, POST	El método GET lista todos los id de las imágenes. El método POST crea una imagen.
/images/multiple	DELETE	Elimina un conjunto de imágenes utilizando sus id's.
/notifications-content/second-survey/{problem}	GET	Devuelve una pregunta aleatoria de entre las preguntas categorizadas como segunda encuesta.
/notifications-content/generic	GET	Devuelve una notificación genérica seleccionada aleatoriamente.
/login	POST	Permite iniciar sesión al usuario.
/register	POST	Permite registrar al usuario.
/refresh	POST	Crea un nuevo token válido para el usuario.

`/users`	GET, POST	El método GET devuelve la lista de todos los usuarios. El método POST crea un nuevo usuario que, en este caso, puede ser administrador, no como en el *endpoint* `/register`.
`/users/{user_id}`	GET, DELETE	El método GET devuelve un usuario por su id. El método POST elimina un usuario por su id.
`/`	GET	Devuelve el estado del servidor.
`/version`	GET	Devuelve la version del servidor.
`/settings`	GET	Devuelve la configuración actual de la API. Requiere ser administrador.
`/me`	GET	Devuelve el usuario actual.
`/accept-disclaimer`	POST	Acepta la exención de responsabilidad.
`/survey-filled`	POST	Indica que la encuesta principal ha sido completada.

4.2.3. Servicios

La aplicación se compone de varios servicios, utilizados para realizar las llamadas al servidor o definir variables que son necesarias en varios componentes. Estos servicios son:

- `AuthService`: servicio que gestiona todo lo concerniente a la autorización de los usuarios, como es el inicio de sesión, el registro de nuevos usuarios o el cierre de sesión. Al estar las sesiones gestionadas por un token, también se encarga del refresco de ese token y, en caso necesario, del *autologin* y del *autologout*. Este servicio es llamado desde un amplio número de componentes para conocer si hay un usuario *loggeado* y qué tipo de usuario es. También es utilizado por los guardas y por el interceptor de mensajes al servidor.
- `DisclaimerService`: este servicio está asociado al componente de la exención de responsabilidad y, envía al *backend*, si el usuario ha aceptado dicha exención.
- `NotificationService`: servicio que gestiona las notificaciones, los distintos tipos que hemos establecido y la gestión de estas cuando un usuario pincha en una notificación. Este servicio solo está disponible para usuarios de Android mediante el APK.
- `OnboardingService`: se encarga de la presentación de la aplicación a un usuario que abre la aplicación por primera vez (tanto en su versión web como en versión móvil). Gestiona que una vez se haya saltado esa presentación, ya no se vuelva a mostrar.
- `ChatbotService`: servicio que gestiona la comunicación con el *chatbot*, enviando los mensajes al servidor y recibiendo las respuestas. Se almacena el historial del chat mientras que la aplicación no sea cerrada o se cambie el idioma de la aplicación.
- `FileService`: servicio que gestiona los documentos comunicándose con el servidor. Como los métodos son comunes tanto para la parte de salud como para la de vitaminas,

los métodos de esta sección son llamados por los servicios relacionados con dichas secciones. También cuenta con métodos específicos que solo puede utilizar el administrador.

- `HealthService`: se encarga de gestionar los documentos de salud disponibles. Almacena la lista de documentos del idioma solicitado y, si se solicita un documento de entre los que se encuentran en la lista, almacena las dos partes de las que se compone el documento mientras no se cierre la aplicación o se cambie de idioma la aplicación. También permite la edición de los documentos, aunque es una función exclusiva de los administradores.
- `VitaminService`: se encarga de gestionar los documentos de vitaminas y sales minerales disponibles y, si se solicita un documento de entre los que se encuentran en la lista, lo almacena mientras no se cierre la aplicación o se cambie de idioma la aplicación. También permite la edición de los documentos, aunque es una función exclusiva de los administradores.

En la aplicación se emplean más servicios, son proporcionados por Angular e IONIC Capacitor a través de las diferentes librerías. Los más importantes para esta aplicación son:

- `Router`: otorga funciones que permiten el ruteo en la aplicación.
- `HttpClient`: contiene los métodos utilizadas para conectar con la API: GET, POST, PUT y DELETE.
- `LocalNotifications`: permite controlar las notificaciones, crearlas, eliminarlas y definir el comportamiento de la aplicación cuando ocurre un evento relacionado con las notificaciones: clic, salida de la notificación y borrado de la notificación.
- `TranslateService`: permite utilizar las traducciones dentro de los componentes.

4.2.4. Componentes

Los componentes permiten configurar las diferentes vistas con las que se forma la aplicación y definir las variables y los métodos con los que interactuarán los usuarios al utilizarla.

Como se mencionaba en la *Sección 4.1.1*, un componente está formado por un fichero TypeScript, un fichero HTML y un fichero CSS (el fichero dedicado a los test unitarios solo es necesario para la fase de testeo). Es importante entender cómo se comunican entre estos ficheros. El fichero TypeScript tiene un decorador `@component` en el que se definen tres parámetros:

- `selector`: define al componente en la plantilla HTML. Se utiliza para introducir el componente en otros. Se introduce en las plantillas con la siguiente etiqueta: `<selectorName></selectorName>`.
- `templateUrl`: indica el PATH al fichero que contiene la plantilla del componente.
- `styleUrls`: indica el PATH de los ficheros de estilos del componente.

Se han creado los siguientes componentes:

- `LoginComponent`: componente inicial que se muestra en la aplicación. Asociado al inicio de sesión del usuario, permite comunicación con el componente de registro para usuarios no registrados. Además, si es la primera vez que se lanza la aplicación, se muestra el componente de *Onboarding.*
- `LogoutComponent`: componente asociado al cierre de sesión de la aplicación. Este componente se introduce dentro del componente de *Settings*.

- `RegisterComponent`: componente que permite el registro de nuevos usuarios. Permite conectar con el componente de inicio de sesión si ya tenemos un usuario.
- `MenuComponent`: componente que contiene la barra de iconos que permite navegar a las diferentes secciones de la aplicación. Se encuentra presente en el AppComponent, es decir, se encuentra siempre en la aplicación. No obstante, solo se muestra si el usuario ha iniciado sesión. En el componente del *chatbot*, también se esconde por motivos de usabilidad.
- `OnboardingComponent`: componente cuyo objetivo es permitir a los nuevos usuarios entender para qué sirve la aplicación. Se muestra únicamente la primera vez que se accede a la aplicación.
- `ChatbotComponent`: componente encargado del *chatbot*. Se utiliza para que el usuario pueda interactuar con él. Muestra los mensajes recibidos desde el servidor y habilita un tipo de respuesta en función de la información que se solicite. También programa las notificaciones relacionadas con los problemas de salud resultantes tras la encuesta inicial.
- `FileComponent`: componente que permite al administrador listar todos los documentos disponibles en el servidor y comprobar los idiomas en los que se encuentra. Puede acceder a dichos documentos (saltando al componente asociado en función si es de salud o vitaminas), eliminar los documentos seleccionados y acceder al componente dedicado a la creación de documentos.
- `FileEditComponent`: componente desarrollado para la creación de documentos, tanto de salud como de vitaminas.
- `HealthComponent`: componente dedicado a listar los documentos de salud disponibles en el idioma en el que se encuentre la aplicación. Permite seleccionar el documento que se desea consultar, saltando al componente `HealthFilesComponent`, que permitirá su muestra.
- `HealthFilesComponent`: componente que permite visualizar un documento de salud. Contiene dos secciones con información seleccionables. Permite volver al listado de documentos de salud. Los administradores, además, podrán editar los documentos.
- `VitaminComponent`: componente dedicado a listar los documentos de vitaminas disponibles en el idioma en el que se encuentre la aplicación. Permite seleccionar el documento que se desea consultar, saltando al componente `VitaminFilesComponent`, que permitirá su muestra.
- `VitaminFilesComponent`: componente que permite visualizar un documento de salud. Permite volver al listado de documentos de vitaminas. Los administradores, además, podrán editar los documentos.
- `SettingsComponent`: componente que muestra el nombre de usuario del usuario registrado. Además, permite el cambio de idioma de la aplicación y contiene el componente para cerrar la sesión.
- `DisplayComponent`: componente encargado de mostrar las diferentes secciones de las que se compone la aplicación, con una pequeña explicación de lo que hace cada una de ellas. Además, gestiona la programación de notificaciones con mensajes genéricos.
- `AppComponent`: componente básico del proyecto, que permite el soporte de toda la aplicación.

4.2.5. Módulos

En la aplicación se han desarrollado dos módulos, el ofrecido por Angular en la creación del proyecto y el que gestiona las rutas de la aplicación:

- `app.module.ts`: en este fichero se declaran todos los componentes que contiene la aplicación, ciertos servicios (no es necesario incluirlos si en el decorador de los servicios se introduce la línea: `providedIn: 'root'`) y otros módulos que hay en la aplicación, el del ruteo y, además, otros módulos extraídos de las librerías de Angular para posibilitar la implementación correcta de la aplicación, como `HttpClientModule` o `TranslateModule`.
- `app-routing.module.ts`: en este fichero se establecen todas las rutas de la aplicación. Para la configuración de una ruta se introducen los siguientes parámetros:
  - `path`: PATH relativo de la ruta.
  - `canActivate`: si la ruta requiere de algún tipo de autentificación se añade el nombre de los guardas necesarios.
  - `component`: el nombre del componente asociado a la ruta. Se cargará dicho componente al navegar a esa ruta.

4.3. Configuración del entorno de trabajo

Una vez definida la arquitectura del sistema se puede proceder con el desarrollo del proyecto, en el que se han diferenciado tres fases principales. La primera es la fase de desarrollo, en la que se ha creado toda la aplicación. En segundo lugar, la fase de testeo, en la que se ha comprobado que todo aquello que hemos creado en la fase anterior funciona correctamente. Por último, la fase de despliegue, en la que se termina la aplicación consiguiendo que funcione en el mundo real.

4.3.1. Fase de desarrollo

En esta fase contamos con dos partes diferenciadas, como es el desarrollo en Angular y el desarrollo en Android, de cara a la configuración del entorno de trabajo. No obstante, veremos que estas dos partes están comunicadas entre sí y que existe relación entre ellas.

4.3.1.1. Desarrollo en Angular

Para crear el proyecto desde cero se instaló Angular CLI con el comando `npm install -g @angular/cli`. Una vez instalado se creó el proyecto con `ng new full-power`. Al ejecutarlo te pregunta si se desea incluir el ruteo de Angular (no es necesario aceptar porque se puede incluir posteriormente) y, también el formato de estilos que se utilizará (entre CSS, SCSS, Sass, Less y Stylus), ya que con la creación del proyecto se generan los ficheros de estilos globales.

Una vez creado el proyecto ya es posible utilizar Angular CLI para crear el resto de los módulos, componentes, servicios… del proyecto. Estos se generan mediante el siguiente comando: `ng generate` o `g` seguido de aquello que deseamos crear y el directorio donde se va a generar junto con el nombre de dicha creación (no es necesario que exista porque lo creará en caso contrario), tomando como raíz el directorio `src/app`. En este proyecto se han creado:

- Componentes: `component` o `c`. Genera un fichero `.ts`, el fichero de test, una plantilla HTML y un fichero de estilos.
- Servicios: `service` o `s`. Genera un fichero `.ts` y el fichero de test.
- Módulos: `module` o `m`. Genera un fichero `.ts`.
- Guardas: `guard`. Genera un fichero `.ts` y el fichero de test.

- Tuberías: `pipe`. Genera un fichero `.ts` y el fichero de test.
- Interceptores: `interceptor`. Genera un fichero `.ts` y el fichero de test.

Por ejemplo: `ng g c auth/login` para crear un `LoginComponent`.

Angular permite tener variables de entorno, que internamente son seleccionadas en función de si nos encontramos en desarrollo o producción. Entre estas variables tenemos un *boolean* que indica si nos encontramos en producción o no, la url del *backend* y otras variables de carácter global.

Para poder trabajar en desarrollo y compilar la aplicación, se necesita utilizar el comando `ng serve`. Por defecto, el servidor estará escuchando en el puerto 4200. Si deseamos cambiar el puerto, solo hay que añadir el *flag* `--port` junto con el número de puerto en el que se desea que el *frontend* escuche. Además, con el *flag* `-o` (`--open`) permitiremos que se ejecute el navegador una vez se ha compilado el proyecto, lo que será útil para rápidamente poder probar la aplicación. Cada vez que se guarda, el proyecto es recompilado y servido de nuevo.

La nueva versión de Angular (Angular 11) ha introducido una interesante utilidad que permite evitar la necesidad de recarga de la página cada vez que el proyecto es recompilado, un aspecto muy importante que facilita la comprobación de valores y el debuggeado de errores. Esto se consigue con otro *flag*, en este caso `--hmr`.

También es posible servir la aplicación sobre HTTPS (*HyperText Transfer Protocol Secure,* Protocolo Seguro de Transferencia de Hipertexto), ya que normalmente la aplicación se servirá sobre HTTP. Para ello necesitaremos un certificado y una llave SSH. Una vez conseguidos tan solo deberemos emplear los siguientes *flags*: `--ssl --ssl-key rutaLlave --ssl-cert rutaCertificado`. Esta utilidad ha sido empleada para comprobar que ciertas partes del código funcionaran independientemente de si la información llegara cifrada o no.

Todos estos comandos eran utilizados para lanzar un servidor de desarrollo. No obstante, el comando `ng serve` no genera un directorio de salida, ya que está pensado para dar una respuesta rápida durante el desarrollo. Si deseamos obtener ese directorio de salida, necesitaremos el comando `ng build`. Esto generará en el directorio `dist` la salida de la compilación del proyecto. Para este comando existen dos *flags* interesantes:

- `--prod`: habilita el compilado de producción. Para el desarrollo no es útil, sin embargo, sí en la fase de despliegue.
- `--watch`: habilita la compilación tras el guardado.

Aunque tras ejecutar este comando, la aplicación no estará sirviéndose en ningún puerto. Para poder servir la aplicación compilada, que se encuentra en `dist`, debemos emplear el siguiente comando `http-server -p 8080 -c-1 dist/full-power`:

- `-p 8080`: puerto donde se servirá la aplicación.
- `-c-1`: indicamos que no queremos utilizar la caché.
- `dist/full-power`: directorio donde se encuentran los ficheros compilados a servir.

4.3.1.2. Desarrollo en Android

Como se ha comentado en la *Sección 4.1.2*, se comenzó utilizando Cordova para la conversión de la aplicación a Android, voy a realizar una breve mención del proceso a seguir. Se debe realizar la instalación de Cordova, tras esto crear un proyecto nuevo y añadirle las plataformas deseadas, en este caso Android. Por último, solo hay que ejecutar un último comando que

compila la aplicación en un fichero `.apk`. Se puede personalizar a través del fichero `config.xml`, que permite editar la configuración de la aplicación de Android a través de etiquetas.

No obstante, esta personalización es mucho más limitada que la que ofrece IONIC Capacitor, ya que permite la creación de un proyecto de Android, que es personalizable desde Android Studio. Para poder convertir la aplicación web en una aplicación híbrida hay que seguir los siguientes pasos:

1. Se añade Capacitor al proyecto: `ng add @capacitor/angular`.
2. Es necesario tener el proyecto compilado, si no se tiene: `ng build`.
3. Añadimos la plataforma de Android: `npx cap add android`.
4. Para abrir el entorno de desarrollo Adroid Studio: `npx cap open android`.
5. Para sincronizar el proyecto de Angular con el de Android: `npx cap copy android`.

Habría que sincronizar el proyecto cada vez que se compila de nuevo, por ello, se ha generado un comando que permite ejecutar ambos comandos en una sola línea. Este comando es: `npm run build-cap`, que sustituye a: `ng build && npx cap copy android`. También se ha creado otro comando específico para producción: `npm run build-cap:prod`, que sustituye a: `ng build --prod && npx cap copy android`. De esta manera, una vez termina el compilado, se copia directamente la salida en el proyecto Android.

Durante este proceso se ha creado el fichero `capacitor.config.json`, que es un fichero de configuración, se han dejado los valores por defecto, excepto en la sección `plugins`, se ha cambiado `SplashScreen` (pantalla que aparece al abrir la aplicación) para añadirle el atributo `"androidScaleType": "CENTER_CROP"`, que permite que la imagen que aparezca no se deforme si la pantalla no coincide exactamente con las medidas estándares.

Entrando más detalladamente en el proyecto Android, los cambios respecto al proyecto generado por defecto han sido:

- **Fichero `android/gradle.properties`**: para evitar errores de compilación debidos a los caracteres del *path* se ha añadido la línea `android.overridePathCheck=true`.
- **Colores:** se pueden personalizar los colores de la barra superior y de la barra inferior de la pantalla del dispositivo Android. Para ello hay que editar dos ficheros del directorio `android/app/src/res/values`: `styles.xml`, donde se incluyen los colores establecidos en `colors.xml`.
- **Iconos:** se han cambiado los iconos por defecto por los iconos de la empresa. En varios tamaños para su adaptación a todo tipo de dimensiones de pantalla. Android Studio permite su cambio de una manera muy sencilla. Situándose en la estructura de ficheros del proyecto, clic derecho sobre el directorio `android/app/src/main/res`, se selecciona "*New*" y dentro del menú desplegable "*Image Asset*" (*Figura 20*). En el menú que se abre se selecciona la imagen que se desea como nuevo icono, el color de fondo del icono y el resto de las opciones por defecto (*Figura 21*).

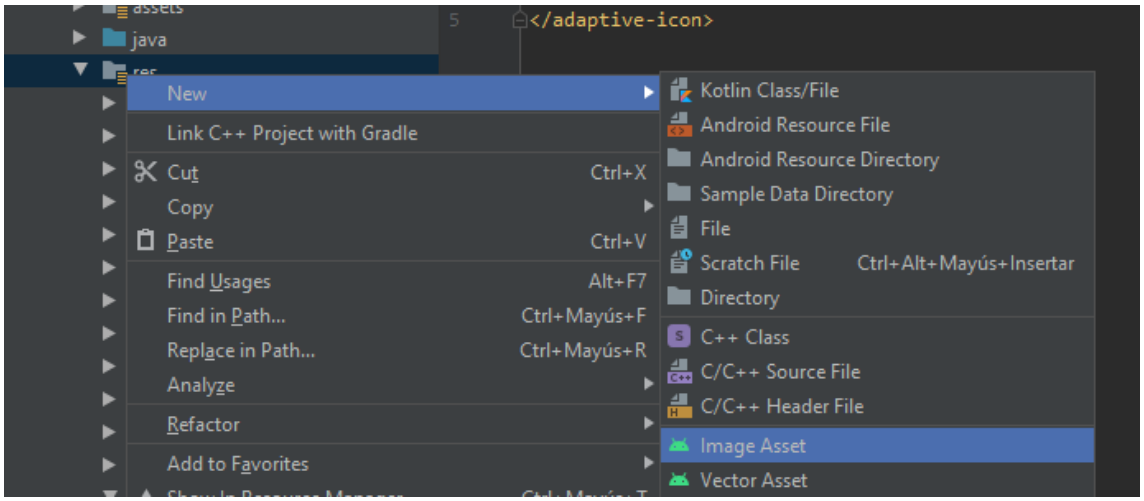

Figura 20.- Creación del icono de la aplicación Android.

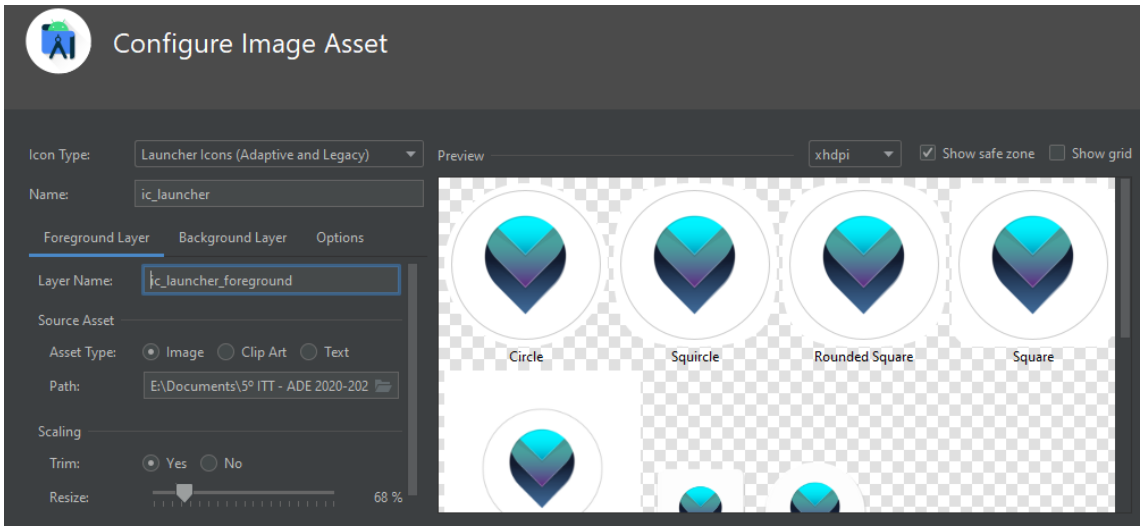

Figura 21.- Configuración del icono de la aplicación Android.

- ***Splash Screen*:** se ha cambiado la pantalla que aparece al lanzarse la aplicación. La única posibilidad de cambiarla es manualmente, por lo que hay que cambiar en cada directorio `android/app/src/res/drawable*` la imagen `splash.png` por la deseada, cada directorio está asignada a un tamaño de pantalla concreto y se diferencia entre vertical y horizontal.
- **Permisos:** esta aplicación solo requiere dos permisos para funcionar, acceso a internet y acceso al estado de la red. Todos los demás son prescindibles y se pueden eliminar en el fichero `android/app/src/res/AndroidManifest.xml`.

Para conseguir el fichero `.apk`, deberemos pinchar en la parte superior de Android Studio en "*Build*". Se abrirá un menú desplegable que seleccionaremos "*Build Bundle(s) / APK(s)*", de donde saldrá otro desplegable en el que seleccionaremos "*Build APK(s)*" (*Figura 22*). El fichero `.apk` se encontrará en el directorio `android/app/build/outputs/apk/debug`.

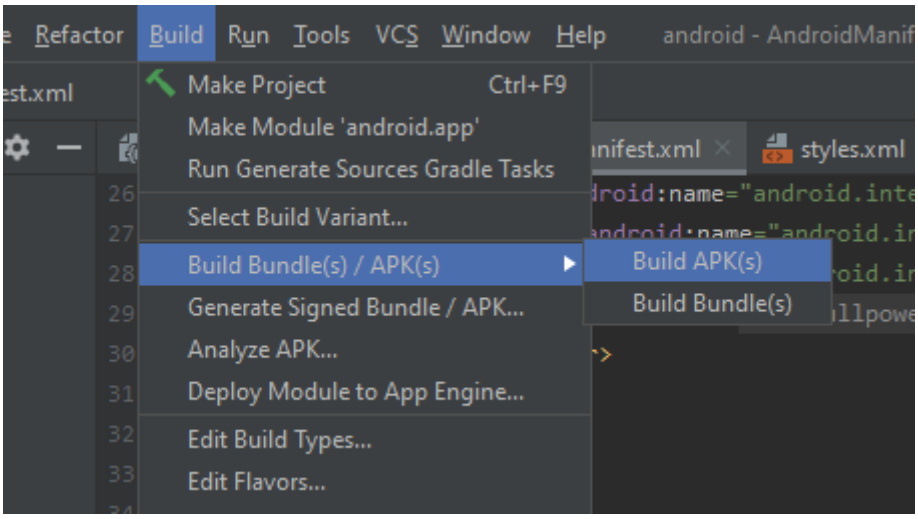

Figura 22.- Proceso para conseguir el fichero `.apk` empleando Android Studio.

4.3.2. Testeo de la aplicación

Con el objetivo de llevar un control del código, que permita conocer cambios en el comportamiento de este y, saber cuándo el código no se está comportando de la forma deseada, se han desarrollado los test unitarios de Angular para poder comprobar el máximo de líneas de código posible.

La estructura de un fichero de test es la siguiente:

- Descripción: función `describe`, que contiene todos los test unitarios de un componente, servicio… normalmente el primer parámetro es una cadena que identifica con el nombre de dicho componente o servicio, el segundo parámetro es un *callback* que contiene los distintos test unitarios y su configuración.
    - Configuración: se pueden definir variables globales para los test y mediante la utilización de métodos como `beforeEach` y `afterEach`, se controla el comportamiento de todos los test antes y después de ejecutarse. Es muy recomendable el primer método ya que permite establecer todas las dependencias que se necesitan para poder ejecutar sin errores los test. El segundo método es más útil cuando se desea borrar ciertos valores tras los test, como el almacenamiento local, evitando así que puedan afectar a sucesivos test.
    - Test unitarios: definidos mediante el método `it`, contiene dos parámetros, en el primero es una cadena en la que se describe el objetivo del test, por ejemplo: *'should show disclaimer'*. Esto facilita la identificación de cada test unitario. El segundo parámetro es un *callback* que ejecuta todos los pasos para completar el test. Todos los test tienen que esperar al menos que se cumpla una condición, para dar el test como válido, en caso de no existir saltará un aviso notificando de este hecho, esta condición se realiza mediante el método `expect`. Se pueden comprobar las variables de los componentes y de los servicios, incluso es posible comprobar las etiquetas de la plantilla HTML.

Además, existe un fichero de configuración del testeo (`karma.conf.js`), que permite la ejecución de todos los test. Los parámetros de configuración con mayor interés:

- `coverageReporter`: aquí se configura la parte correspondiente al reporte de cobertura de los test, el directorio donde se almacenarán los resultados y el tipo de ficheros que se crearán. Para el programador individual, el tipo de fichero más conveniente mientras va generando los test es `html`, ya que se puede ver de una manera visual cuál es la cobertura de cada fichero. De cara a la integración continua, el tipo de fichero necesario es `lcovonly`.
- `autoWatch`: permite habilitar la ejecución de nuevo de los test cada vez que se guarda un fichero del proyecto cuando su valor es `true`, algo muy útil para comprobar rápidamente los cambios en el código durante la generación de los test unitarios.
- `browsers`: navegadores donde se van a ejecutar los test, el valor que aparece por defecto es `Chrome`, se ha añadido `ChromeHeadlessNoSandbox`, que permite la ejecución de los test automáticos para una integración continua, debido a que se trata de un navegador que se ejecuta sin ventana.

Para poder ejecutar los test se necesita utilizar el comando `ng test`, no obstante, si se desea obtener la cobertura de los test se debe añadir el *flag* `--code-coverage`. Además, al estar configurado para ejecutar los test automáticos en la integración continua, debemos indicar en que navegador deseamos ejecutar los test, puesto que, en caso contrario, se ejecutarán dos veces, una por cada navegador. Esto se consigue con el *flag* `–browsers`, en el caso de querer ejecutar los test y ver en la interfaz web los resultados de los test igualaremos ese *flag* a `Chrome` y, en el caso de que no (para la integración continua), lo igualaremos a `ChromeHeadlessNoSandbox`. De cara a evitar escribir todo el comando cada vez que se desee testear la aplicación, se puede simplificar gracias a la configuración del fichero package.json, en el que podemos configurar entre otros, comandos personalizados. En este proyecto, para ejecutar los test con cobertura en Chrome se ha utilizado el comando `npm test`. En el caso de la integración continua, se ha utilizado el comando `npm run test:prod`.

En cuanto a la cobertura de los test, se proporcionan 4 parámetros principales que nos aportan información sobre la cobertura del código:

- *Lines*: número de líneas que contiene el fichero.
- *Statements*: similar al número de líneas, no obstante, en una misma línea pueden existir dos *statements*.
- *Branches*: cada rama indica un posible flujo en la ejecución del fichero. Una declaración de un `if` abre dos posibles ramas, que se cumpla o que no la condición a estudiar. Será necesario pasar por las dos ramas para completar la cobertura. Hay ocasiones en las que no es necesario testear una rama si, por ejemplo, nunca se va a dar el caso o no se hace nada en caso de no cumplirse la condición. Jasmine permite saltarse esas ramas escribiendo antes de la declaración `if`: `/* istanbul ignore else*/`.
- *Functions*: número de funciones del fichero.

Además de poder saltar condiciones en la cobertura, de una manera más general se pueden saltar funciones completas o ficheros enteros con la línea: `/* istanbul ignore next */`. No obstante, no es una buena práctica porque puede dar la falsa sensación de tener el código bien cubierto, 100% de cobertura, cuando realmente no es así. En este proyecto se ha llegado a un 96% de cobertura, dejándose sin cubrir por completo dos ficheros:

- `app.component.ts`: no se ha podido testear el método `subscribeWithPriority`, utilizada para escuchar el evento del *backButton* físico

y habilitar la posibilidad de ir hacia el menú desde cualquier parte de la aplicación y salir de la aplicación al encontrarnos en el menú. No se ha podido testear por no haber encontrado una forma de comprobar esa subscripción con prioridad.

- `upload-adapter.model.ts`: no se ha podido testear ninguna de las funciones de este fichero, dedicado a la subida de imágenes con el editor de texto proporcionado por CKEditor, por no haber podido replicar una instancia de esta clase de cara a los test para poder comprobar sus métodos.

En las siguientes figuras (*Figura 23* y *Figura 24*) se muestra un ejemplo de la salida del comando `npm test`. Tanto de los propios test como de su cobertura.

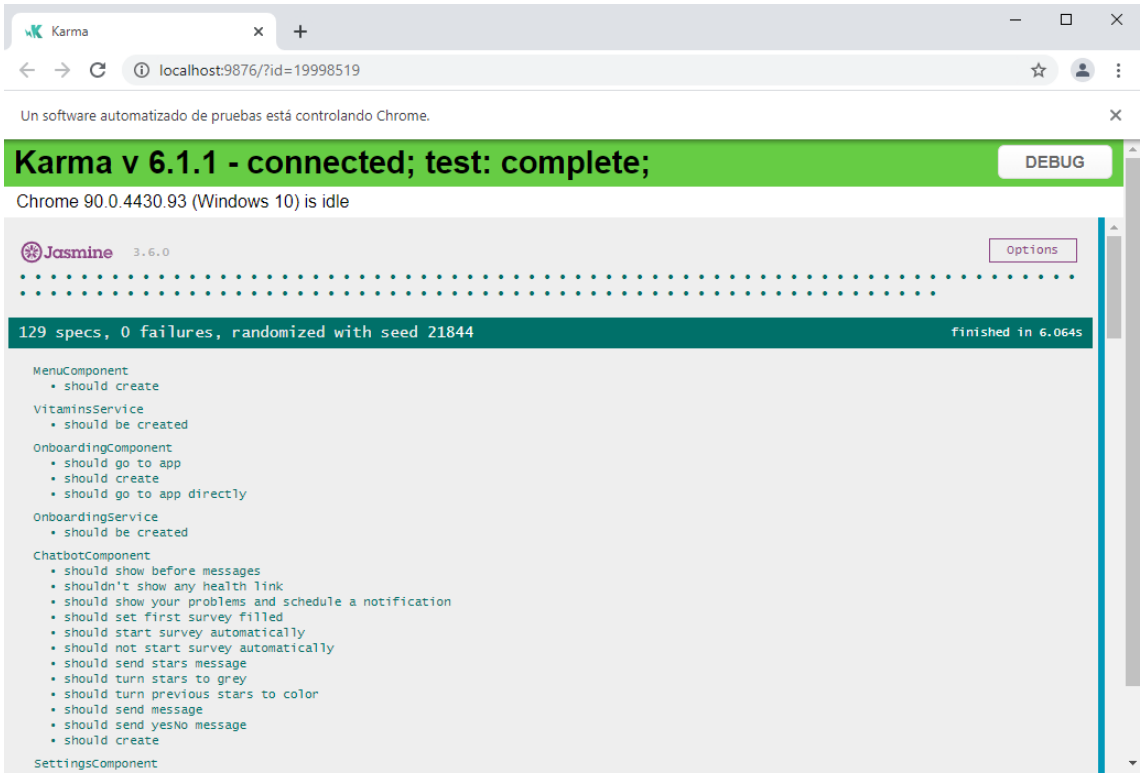

Figura 23.- Ejemplo de salida de los test en el navegador.

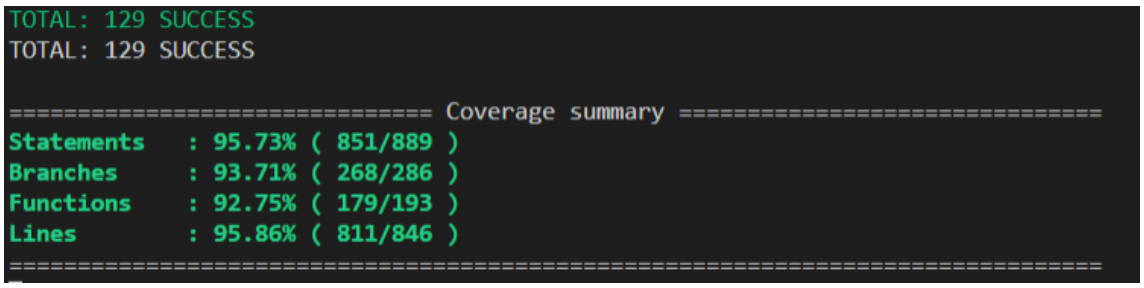

Figura 24.- Ejemplo de salida de la cobertura del código en la consola.

Además, podemos ver la cobertura fácilmente con el fichero HTML generado, en la *Figura 25* podemos ver de manera general la cobertura de todos los directorios y, si vamos pinchando en cada uno, veremos los ficheros que contienen y su cobertura, llegando a observar fragmentos similares a los de la *Figura 26*, cuando se hace la cobertura (podemos observar en la parte izquierda el número de veces que se ha ejecutado cada línea) y, en la *Figura 27*, cuando esa parte de código no está cubierta (el *callback* de la función no se ha llegado a ejecutar).

All files

95.73% Statements 851/889 **93.71%** Branches 268/286 **92.75%** Functions 179/193 **95.86%** Lines 811/846

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
src	100%	3/3	100%	0/0	100%	0/0	100%	3/3
src/app	87.5%	21/24	75%	12/16	66.67%	2/3	86.96%	20/23
src/app/auth	100%	98/98	100%	28/28	100%	28/28	100%	91/91
src/app/auth/disclaimer	100%	7/7	100%	0/0	100%	3/3	100%	6/6
src/app/auth/login	100%	41/41	100%	19/19	100%	6/6	100%	40/40
src/app/auth/logout	100%	6/6	100%	0/0	100%	2/2	100%	4/4
src/app/auth/register	100%	64/64	100%	45/45	100%	6/6	100%	63/63
src/app/display	100%	23/23	100%	2/2	100%	7/7	100%	22/22
src/app/display/menu	100%	7/7	100%	0/0	100%	2/2	100%	5/5
src/app/display/onboarding	100%	12/12	100%	2/2	100%	4/4	100%	10/10
src/app/display/sections/chatbot	100%	144/144	100%	38/38	100%	35/35	100%	138/138
src/app/display/sections/file	100%	10/10	100%	1/1	100%	7/7	100%	9/9

Figura 25.- Cobertura vista con el fichero `index.html` generado en con los test.

```
63
64        ngAfterViewChecked(): void {
65  61x     if (!!this.fileName) {
66  10x       const fileName = this.fileName.nativeElement.value;
67  10x       if (!!fileName) {
68   6x         this.createEnabled = true;
69   6x         this.chDet.detectChanges();
70           }
71           else {
72   4x         this.createEnabled = false;
73   4x         this.chDet.detectChanges();
74           }
75         }
76       }
```

Figura 26.- Fragmento de la cobertura del fichero `file.component.ts`.

```
22  7x    this.platform.backButton.subscribeWithPriority(9999, () => {
23          if (this.router.url !== '/home' && this.router.url !== '/login')
24            {this.router.navigate(['/home']);}
25          else {App.exitApp();}
26        });
```

Figura 27.- Fragmento de la cobertura del fichero `app.component.ts`.

Para terminar la fase de testeo, es importante mencionar la integración continua. En este proyecto se ha configurado en el fichero `./github/workflows/test.yml`. En este fichero se configura el momento en el que se realizan los test, en este caso cada vez que se realiza una *Pull Request* a la rama `main`.

Posteriormente se definen las versiones del entorno en las que se van a ejecutar los test. En este caso se ha decidido realizarla en las tres últimas versiones *major* disponibles.

Finalmente, se establecen todos los pasos a seguir para la ejecución de los test. Comenzando por la configuración del entorno y continuando por el lanzamiento de los test. Este proceso termina con la creación del fichero que muestra el reporte con los resultados y la cobertura de los test, que se realizan con Codecov.

4.3.3. Fase de despliegue

Una vez la aplicación está terminada y se han superado todos los test. Se puede pasar a la fase de despliegue, en la que pasamos la aplicación al entorno de producción para que pueda ser utilizada por los usuarios finales.

Para poder pasar la aplicación de Angular a producción, se va a utilizar el comando ya mencionado en la fase de desarrollo: `npm run build-cap:prod`. Este comando nos proporcionará los ficheros necesarios para poder desplegar la aplicación en cualquier servidor preparado para páginas web y, además, la actualización de los ficheros en el proyecto de Android.

La aplicación se encuentra dentro de un contenedor Docker para ser desplegada. Esto se consigue con el fichero de configuración `Dockerfile`. Se configuran varias instrucciones con el objetivo de compilar la aplicación en versión de producción.

Para construir la imagen de Docker que puede ser desplegada (ha sido desplegada en una Raspberry Pi 4): `docker build -t <username>/frontend:<tag>`. Esta imagen debe ser publicada en Docker Hub [26]: `docker push -t <username>/frontend:<tag>`. Como ha sido lanzada en una arquitectura ARM (la arquitectura de la Raspberry Pi), se ha modificado el comando anterior añadiendo los flags: `--platform=linux/arm/v7 --push`.

Para lanzar el contenedor hay que ejecutar: `docker run -dp 80:80 <username>/frontend:<tag>`.

De cara a obtener la aplicación de Android en su versión de producción deberemos tener en cuenta dos posibilidades:

- Compilar directamente a un fichero `.apk`.
- Generar un fichero Android App Bundles (`.aab`): es un formato de publicación de la aplicación que delega la generación del APK a Google Play. La diferencia con la compilación directa es que Google Play optimiza directamente el APK para cada dispositivo, mejorando su compatibilidad y permitiendo a los usuarios descargas de menor tamaño. Requerirá entregar la firma del desarrollador por separado.

En ambos casos hay que generar una firma de desarrollador, para evitar que las aplicaciones de la tienda de Google Play sean anónimas. Para generar esta firma habrá que seguir el siguiente procedimiento:

1. Dentro de Android Studio, en la parte superior, hay que hacer clic en la sección "*Build*" y dentro del desplegable, seleccionar "*Generate Signed Bundle / APK…*" (*Figura 28*).

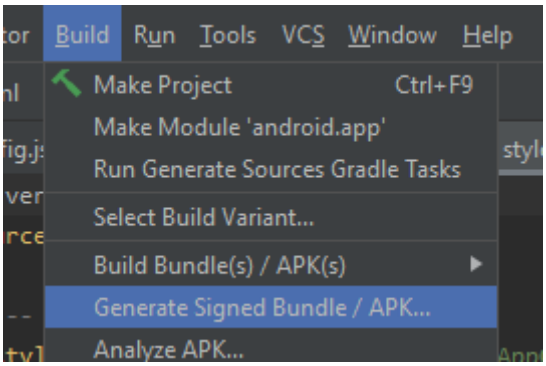

Figura 28.- Desplegable de la sección "Build" seleccionando "Generate Signed Bundle / APK…".

2. Se abrirá una ventana en la que tendremos que decidir entre las dos opciones de generar la salida del proyecto, seleccionamos la deseada (*Figura 29*).

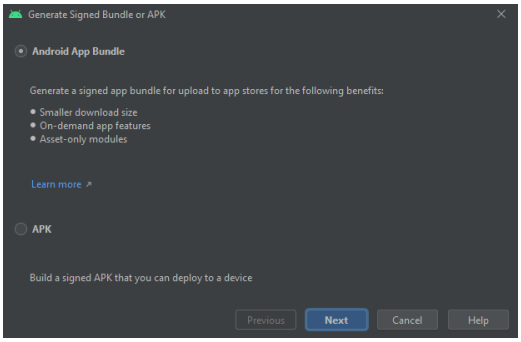

Figura 29.- Selección del formato de salida del proyecto.

3. Se nos solicitará el lugar donde se encuentra la firma, si no tenemos ninguna, crearemos una nueva haciendo clic en "*Create new…*", en este caso completaríamos los campos solicitados para crear la firma. Si ya disponemos de una y, no aparece en el PATH por defecto, seleccionamos "*Choose existing…*" y buscamos en el sistema de ficheros el lugar donde se almacena la firma (*Figura 30*).
    - La única diferencia entre el APK y el Bundle es el campo de exportación de la firma encriptada, ya que en el caso del APK, se introduce en el momento de la compilación y, como ya mencionaba previamente, en el otro caso se hará al ser subida a Google Play. Seleccionaremos el directorio de almacenamiento de la firma.

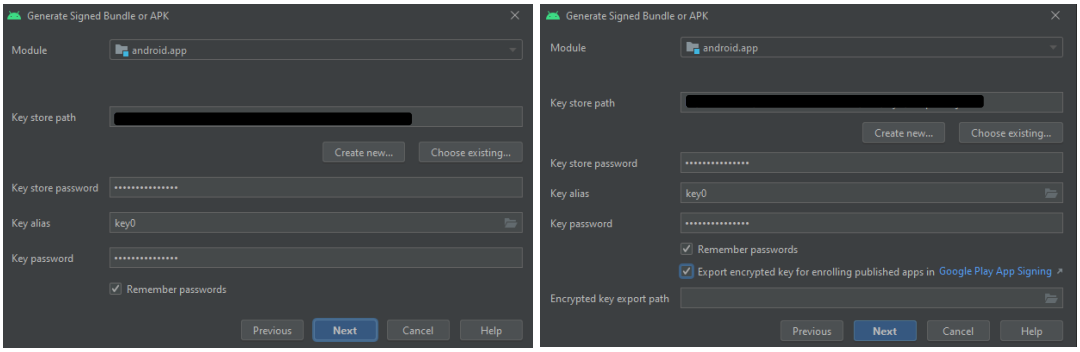

Figura 30.- Generación de la firma. A la izquierda para el APK y a la derecha para el Bundle.

4. Por último, al encontrarnos en producción debemos seleccionar la opción "*Release*" y, al aceptar, obtendremos nuestra aplicación lista para subir a Google Play (*Figura*

31). En el caso del APK tendremos que seleccionar el tipo de firma (compatible con diferentes versiones de Android).

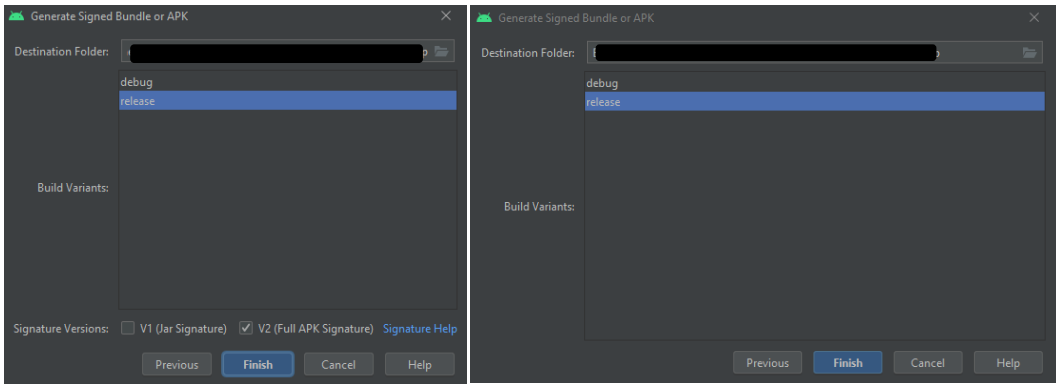

Figura 31.- Selección entre producción y desarrollo. A la izquierda para el APK y a la derecha para el Bundle.

4.4. Estructura de ficheros

Se ha seguido la estructura de ficheros proporcionada inicialmente por Angular en la creación del proyecto con las variaciones necesarias para poder implementar todas las funcionalidades que permiten el correcto funcionamiento de la aplicación. La estructura es la siguiente (partiendo del directorio raíz del proyecto):

- `.github`: en este directorio se encuentra la configuración que permite realizar los test automáticamente cuando se realiza un mergeado de una rama en Git.
- `.vscode`: en este directorio se encuentra una configuración para VSCode.
- `android`: en este directorio se encuentra toda la configuración para poder crear el fichero ejecutable en Android. Esta configuración es proporcionada por IONIC excepto algunos cambios mínimos de estilos, imágenes, etc. Necesarios para la personalización de la aplicación. Dentro de este directorio, los ficheros más relevantes se encuentran dentro del directorio `app`:
  - `build`: directorio donde se encuentra el fichero `.apk` obtenido tras la compilación del proyecto de Android durante la fase de desarrollo.
  - `release`: directorio donde se encuentra el fichero `.apk` obtenido tras la compilación del proyecto de Android durante la fase de producción.
  - `src`: directorio que contiene el proyecto de Angular compilado en el subdirectorio `assets`, el subdirectorio `java`, que contiene el fichero que controla la ejecución del proyecto de Android y el subdirectorio `res`, que contiene distintos formatos de colores, iconos y pantallas de lanzamiento de la aplicación para los diferentes tamaños de dispositivos móviles. Además, el directorio contiene el fichero `AndroidManifest.xml`, que contiene toda la configuración de la aplicación.
- `ckeditor5-build-classic`: directorio que contiene el repositorio del editor de texto utilizado para la creación y edición de ficheros. Dentro del subdirectorio `src` se puede encontrar el código fuente para personalizar el editor de texto que, al ser compilado, puede ser obtenido el código resultante en el subdirectorio `build`, donde puede ser copiado para su utilización dentro del directorio `src/assets`, descrito posteriormente.
- `coverage`: directorio que contiene un único subdirectorio con el nombre del proyecto, donde se almacenan los resultados de las coberturas de código tras las ejecuciones de

los test unitarios. El fichero para poder observar de manera sencilla la cobertura es `index.html`, se puede observar en cualquier navegador.

- `dist`: directorio que contiene un único subdirectorio con el nombre del proyecto, donde se almacena el proyecto una vez ha sido compilado, independientemente de la fase del proyecto (desarrollo o producción).
- `e2e`: directorio que contiene los resultados de la ejecución de los test *end-to-end* (test dedicados a comprobar la aplicación a nivel usuario y no a nivel de código). No se ha utilizado en este proyecto.
- `node_modules`: directorio que contiene todas las dependencias del proyecto.
- `src`: directorio en el que se realiza el desarrollo del proyecto. Contiene todos los ficheros que serán compilados posteriormente. Estos ficheros siguen la siguiente estructura:
    - `app`: directorio que contiene todos los módulos, servicios, clases, componentes, tuberías… del proyecto.
    - `assets`: directorio que contiene recursos necesarios para el correcto funcionamiento de la aplicación, entre ellos, el editor de texto de CKEditor, imágenes y traducciones en los idiomas disponibles.
    - `environments`: directorio que contiene las configuraciones de entorno de desarrollo y de producción.
    - `.htcaccess`: fichero que permite cargar la página con cualquier *path* o recargarla, debido a que el *path* en Angular no apunta a un fichero, sino a un componente.
    - `favicon.ico`: icono de la aplicación. Se puede observar en la pestaña del navegador.
    - `index.html`: fichero que contiene el encabezado del código HTML de la aplicación y lo configura.
    - `main.ts`: fichero que contiene la configuración global del proyecto en TypeScript.
    - `polyfills.ts`: fichero para hacer compatible la aplicación con los distintos navegadores.
    - `styles.css`: fichero CSS que contiene variables globales de estilos para todo el proyecto.
    - `test.ts`: fichero TypeScript requerido para la ejecución de todos los test.
- `.browserslistrc`: fichero que contiene la lista de navegadores para los que debe funcionar.
- `.editorconfig`: configuración para el entorno que se utilice, estableciendo, entre otros, el formato de los caracteres.
- `.eslintrc.json`: fichero que configura las normas de estilo del código.
- `.gitignore`: fichero utilizado para que los ficheros incluidos en este no sean incluidos en el repositorio.
- `.gitmodules`: fichero que contiene el submódulo de Git utilizado para el editor de texto.
- `Dockerfile`; fichero que configura el despliegue de la aplicación en un contenedor de Docker.
- `README.md`: fichero escrito en Markdown en el que se explican los pasos a realizar para configurar el proyecto.
- `angular.json`: fichero que contiene la configuración de Angular.

- `capacitor.config.json`: fichero que contiene la configuración de IONIC Capacitor.
- `karma.conf.js`: fichero que contiene la configuración de los test.
- `package-lock.json`: fichero que especifica las dependencias del fichero `package.json`.
- `package.json`: fichero que contiene todas las dependencias que son requeridas para la configuración del proyecto.
- `tsconfig.app.json`, `tsconfig.json` y `tsconfig.spec.json`: ficheros de configuración de TypeScript.

4.5. Implementación

4.5.1. Comunicación con la API

Angular, a través de la librería `http`, permite utilizar el servicio `HttpClient` [17] para lanzar llamadas al *backend*. Se ha empleado el patrón CRUD (*Create, Read, Update, Delete*), para lo cual, se han utilizado los métodos que permiten hacer las llamadas GET, POST, PUT y DELETE, cuyo nombre es homónimo con la función que realizan.

En función del tipo de *endpoint* al que se dirija la llamada, el *backend* requiere o no la autenticación por parte del cliente. Esta autenticación se realiza mediante un token generado en el *backend* utilizando JWT (JSON Web Tokens) [27] y que es enviado como respuesta al iniciar sesión junto con el tiempo de validez del token [5].

El *frontend* almacena el token mientras que el token sea válido, esto permite controlar la duración de la sesión también en el *frontend* y, cerrar la sesión automáticamente si el tiempo de validez se ha agotado. Para evitar que a un usuario se le cierre la sesión mientras utiliza la aplicación, se solicita un nuevo token con un nuevo tiempo de validez.

Además, se utiliza el interceptor con el objetivo de capturar la petición HTTP antes de ser emitida e introducir una cabecera con el token y que dicha petición permita autenticar al usuario en el *backend*. Este interceptor se utiliza siempre que haya un usuario con sesión iniciada, en caso contrario, no se envía la cabecera. Esto ocurrirá, por ejemplo, al crearse una cuenta de usuario o al iniciar sesión. En la *Figura 32* podemos ver un esquema explicativo de este suceso.

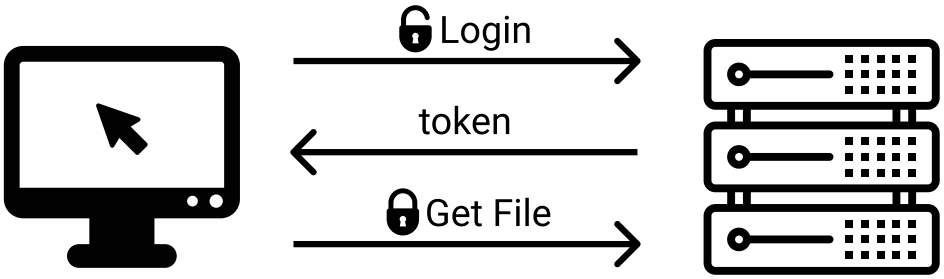

Figura 32.- Proceso de obtención y uso del token de autenticación.

4.5.2. Editor de texto

El editor de texto se ha empleado para que el administrador pueda crear los documentos de salud que se encuentran en la aplicación.

Este editor se ha generado a partir de una base proporcionada por CKEditor, añadiendo los *plugins* deseados para obtener el editor ideal.

Los plugins que se han añadido han sido:

- Negrita.
- Cursiva.
- Adición y modificación de imágenes.
- Añadir títulos.
- Variación del tamaño de letra.
- Subrayado.
- Justificación del texto.
- Centrado del texto.
- Links.
- Listas.
- Listas numéricas.
- Youtube.

Este editor es compilado y añadido como un recurso en el directorio `src/assets/ckeditor`.

De manera específica, para la subida de imágenes ha sido necesario añadir el fichero `upload-adapter.model.ts`, con un código proporcionado por CKEditor, que ha sido modificado para adecuarlo a la aplicación. Permitiendo así que al añadir las imágenes al editor se guarden también en el *backend*.

4.5.3. Traducciones

Angular ofrece muchas facilidades para implementar las traducciones. Es suficiente con añadir en el fichero `app.module.ts` dentro de `imports` en el decorador `@NgModule` las siguientes líneas:

```
TranslateModule.forRoot({
      loader: {
        provide: TranslateLoader,
        useFactory: httpTranslateLoader,
        deps: [HttpClient]
      }
    })
```

Dentro del mismo fichero hay que añadir el siguiente método:

```
export function httpTranslateLoader(http: HttpClient) {
  return new TranslateHttpLoader(http, './assets/i18n/',
'.json');
}
```

En esas líneas se ha definido el PATH del directorio donde se van a encontrar los ficheros con las traducciones. En ese directorio hay que crear un fichero en formato JSON por cada idioma del que se desea traducción, tres para esta aplicación: español, inglés y francés.

Para poder asignar las traducciones en los lugares deseados hay que crear los identificadores de las traducciones. Por cada frase diferente a traducir habrá un identificador. Lo habitual es incluir estos identificadores en las plantillas HTML y aplicar la tubería `translate` para que aparezca traducida en cada idioma. Voy a mostrar un ejemplo de traducción.

- En la plantilla HTML: `{{'Logout'|translate}}`.
- En el fichero `es.json`: `"Logout": "Cerrar sesión"`.
- En el fichero `en.json`: `"Logout": "Log out"`.
- En el fichero `fr.json`: `"Logout": "Se déconnecter"`.

Al arrancar la aplicación se selecciona el idioma del navegador/dispositivo en el que se utiliza la aplicación, en caso de no ser ninguno de los disponibles se selecciona el inglés.

4.5.4. Notificaciones

En esta aplicación se utilizan las notificaciones locales por dos motivos:

- Invitar al usuario a volver a la aplicación.
- Recordar al usuario a seguir los consejos de la aplicación para conseguir alcanzar el máximo potencial.

Como consecuencia, existen dos tipos de implementaciones de estas notificaciones. Por un lado, las notificaciones que invitan al usuario a volver a la aplicación las denominaré "Notificaciones genéricas" o solo "genéricas" y, las otras notificaciones las llamaré "Notificaciones de salud" para referirme en sucesivas ocasiones.

Todas las notificaciones están gestionadas desde el servicio `NotificationService` [19], tal y como se mencionaba en el apartado 4.2.3 y, en última instancia se utiliza el servicio proporcionado por IONIC para la utilización de las notificaciones: `LocalNotifications`. La diferencia se encuentra en que las notificaciones genéricas son programadas desde `DisplayComponent` y, las notificaciones de salud desde `ChatComponent`.

Al tratarse de notificaciones a nivel local, estas solo pueden ser programadas cuando la aplicación está siendo utilizada, es por ello, que todas las notificaciones se programan con repetición, es decir, la notificación aparecerá periódicamente (con el periodo marcado inicialmente) hasta que se pinche en la notificación y se entre de nuevo en la aplicación.

En el caso de las notificaciones generales, una vez se entra a la aplicación, en la página `/home`, se programa una notificación llamando al *backend* al *endpoint* `/notifications-content/generic`, que permite obtener un mensaje de entre los disponibles para ese tipo de notificación. Cada vez que se hace clic en una notificación de este tipo, se vuelve a programar una nueva y se navega de vuelta al `/home`.

Por otro lado, las notificaciones de salud son programadas por primera vez cuando se completa la encuesta principal si tenemos algún problema. En este caso, se hace una llamada al *backend* al *endpoint* `/notifications-content/second-survey/{problem}`, que permite obtener un mensaje con una de las preguntas relacionadas con el problema de salud `{problem}`. Finalmente, se programa la notificación con una frecuencia mayor o menor en función de la gravedad del problema (relación directa entre gravedad del problema y frecuencia de notificación).

Una vez el usuario hace clic en la notificación de salud, se programa de nuevo otra notificación del mismo tipo con una pregunta nueva, con la misma frecuencia que la anterior. A continuación, se redirige al usuario a la zona de chat y se le hace la pregunta que aparecía en el mensaje de la notificación. Se produce una interacción con el *backend* en la que se intercambian la respuesta a la pregunta y el consejo relacionado con la pregunta.

Es importante destacar que, en ambos casos, es necesario iniciar sesión para poder acceder a la aplicación, lo que no impide que se programe la notificación y que posteriormente se redirige al destino al que se pretendía llegar.

5. Manuales de uso

En este capítulo se explican los manuales de usuario básico y administrador, estableciendo previamente en cada tipo de usuario los requisitos necesarios para poder utilizar la aplicación. Los administradores pueden realizar todas las funciones que realizan los usuarios básicos y, para evitar redundancias, en la sección dedicada al administrador solo se han incluido las funciones restringidas a este tipo de usuarios.

5.1. Manual de usuario básico

5.1.1. Requisitos

Los únicos requisitos para acceder a la aplicación es disponer de un dispositivo con conexión a internet y un navegador web. No hay requisito de sistema operativo, excepto para la versión híbrida de Android, que requerirá la instalación del APK para poder utilizarla. Aunque la aplicación no esté instalada se podrá utilizar la versión web, en cualquier caso. Por defecto, el idioma seleccionado es el idioma del navegador (ordenador) o dispositivo, en el caso de móviles.

5.1.2. Primera vez en la aplicación

Al entrar por primera vez en la aplicación, se le mostrará una serie de ventanas explicativas sobre la aplicación (*Figura 33*) deslizando de izquierda a derecha (o viceversa), se podrán saltar haciendo clic en el botón habilitado para ello, saltando a la parte de inicio de sesión.

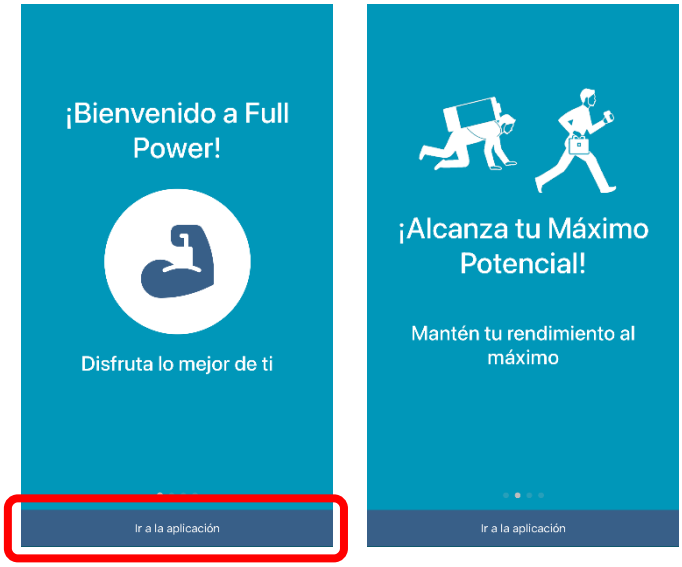

Figura 33.- Presentación de la aplicación.

5.1.2.1. Registro

Como todavía no tenemos una cuenta con la que iniciar sesión, deberemos ir a la sección que permite crearnos una. Para ello, hacemos clic en el enlace que nos indica para registrarnos (*Figura 34*).

Figura 34.- Pantalla de inicio de sesión.

Seguimos las instrucciones rellenando los campos disponibles y hacemos clic en "Registrarse". Si los datos son correctos, se redirigirá a la página de inicio de sesión (*Figura 35*).

Figura 35.- Página de registro.

5.1.2.2. Inicio de sesión

Una vez hemos creado nuestra cuenta, podemos iniciar sesión. Para ello, introducimos nuestro nombre de usuario y contraseña y, si son correctos, podremos acceder a la aplicación (*Figura 36*).

Figura 36.- Iniciando sesión.

5.1.2.3. Aceptar exención de responsabilidad

Como estamos accediendo por primera vez, deberemos aceptar la exención de responsabilidad haciendo clic en el espacio habilitado para ello, si no, no podremos utilizar la aplicación (*Figura 37*).

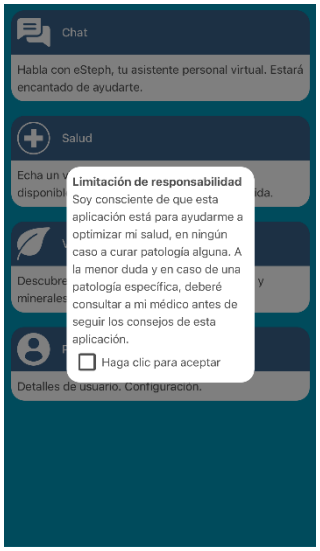

Figura 37.- Aceptación de la exención de responsabilidad.

5.1.2.4. Completar primera encuesta

Si ahora accedemos a la sección de “Chat” veremos que el *bot* nos invita a rellenar una encuesta inicial (*Figura 38*).

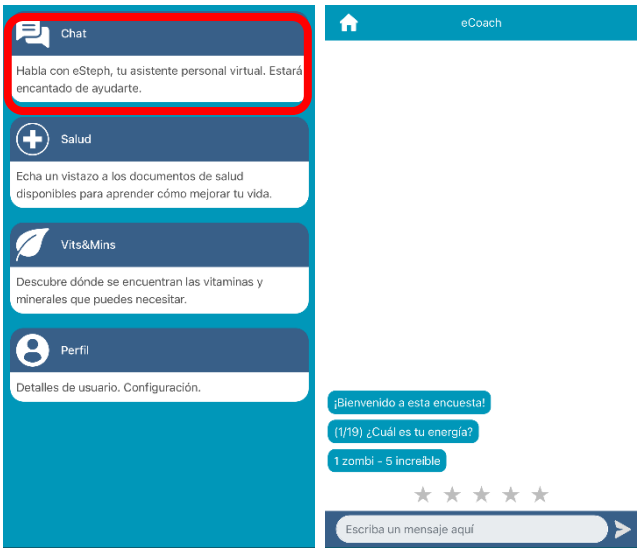

Figura 38.- Selección de sección de "Chat" y comienzo de primera encuesta.

Las preguntas pueden ser contestadas haciendo clic en las estrellas, siendo marcadas en función de cómo nos encontremos en el contexto de cada pregunta (*Figura 39*).

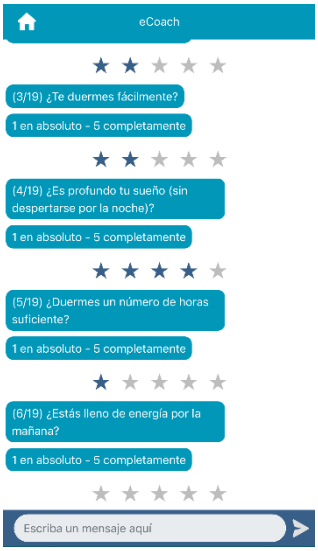

Figura 39.- Respuesta de varias preguntas de la encuesta.

Al finalizar la encuesta, se nos informará de nuestra situación y si tenemos algún problema de salud que debemos tener en cuenta (*Figura 40*). En caso afirmativo, se nos mostrará cuál es ese problema, su gravedad, un enlace al documento relacionado (al que podemos navegar) y, en el caso de la aplicación de Android, se programará una notificación relacionada con dicho problema para los días sucesivos.

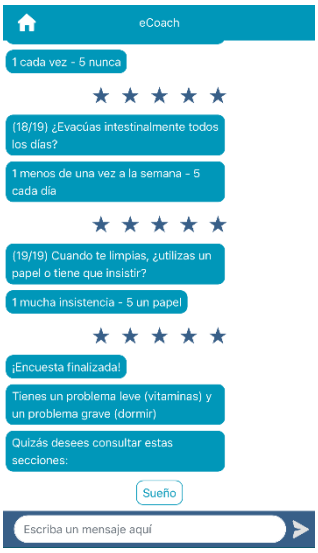

Figura 40.- Resultado de la encuesta.

5.1.3. Navegar por la aplicación

La navegación por la aplicación se realiza principalmente con la barra de navegación, que tiene acceso a las siguientes secciones (*Figura 41*):

1. Chat.
2. Salud.
3. Vit&Mins.
4. Perfil.

Figura 41.- Barra de navegación.

En la sección de Chat, por motivos funcionales se ha eliminado esta barra, en su lugar se puede hacer clic en el icono de la casa para ir a la página principal (*Figura 42*).

Figura 42.- Icono para ir a la página principal desde el chat.

5.1.4. Cierre de sesión

Para cerrar sesión, deberemos ir a la sección de “Perfil”, una vez allí, hacemos clic en el botón de “Cerrar sesión”. Navegaremos a la página de inicio de sesión (*Figura 43*).

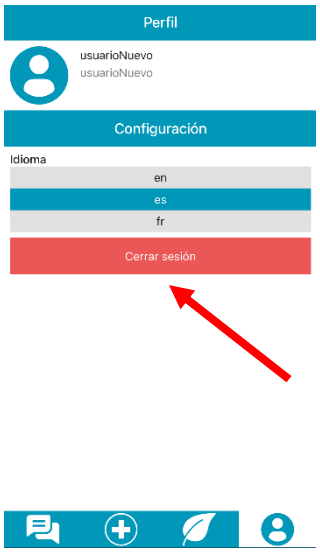

Figura 43.-Cierre de sesión.

5.1.5. Consultar documentos

Para poder consultar documentos deberemos ir a las secciones habilitadas para esta labor, "Salud" y "Vits&Mins", al acceder a una de estas secciones, veremos los documentos disponibles en dicha sección, pudiendo seleccionar cualquiera de ellos (*Figura 44*).

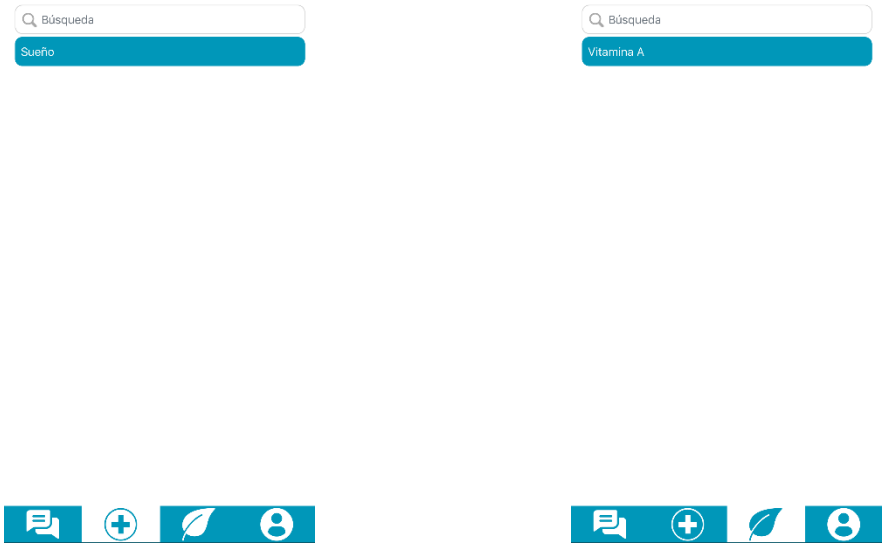

Figura 44.- Lista de documentos de salud (izquierda) y lista de documentos de vits&mins (derecha)

Al seleccionar, podremos ver el documento, en el caso de "Vits&Mins", veremos el documento directamente (*Figura 45*). En el caso de "Salud", deberemos pinchar en uno de los botones: "Entender" o "Actuar", para elegir cuál de las secciones del documento queremos leer (*Figura 46).*

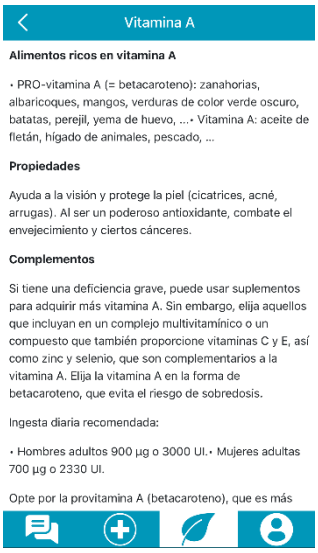

Figura 45.- Documento de vitaminas.

Figura 46.- Documento de salud.

5.1.6. Contestar pregunta de notificación

Una vez recibimos una notificación (*Figura 47*) que contiene una pregunta de salud, al pinchar en ella, se abrirá la aplicación, iniciaremos sesión en el caso que sea necesario y podremos responder a dicha pregunta en la sección del *chatbot* (*Figura 48*).

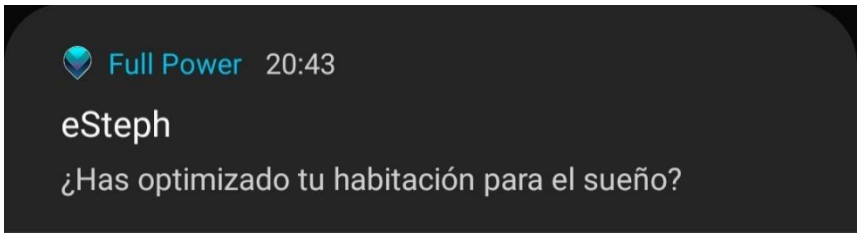

Figura 47.- Notificación con pregunta.

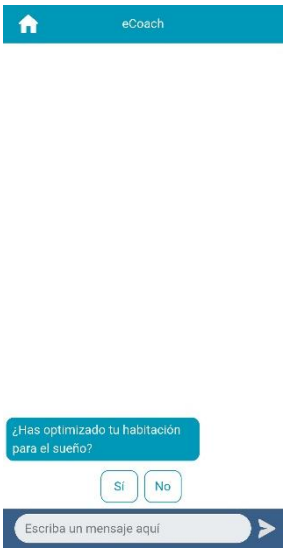

Figura 48.- Pregunta de la notificación.

Obtendremos un consejo del *bot* en función de nuestra respuesta (*Figura 49*).

Figura 49.- Respuesta del chatbot ante respuesta afirmativa de la pregunta.

5.1.7. Cambiar idioma

Para cerrar sesión, deberemos ir a la sección de “Perfil”, una vez allí, hacemos clic en el idioma deseado entre los disponibles (*Figura 50*).

Figura 50.- Cambiar idioma de la aplicación.

5.1.8. Hablar con el *bot*

Para hablar con el *bot*, tan solo deberemos acceder a la sección de "Chat" y escribir en el espacio habilitado para ello (*Figura 51*).

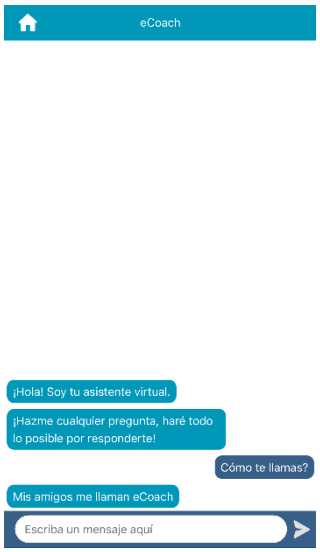

Figura 51.- Posible conversación con el chatbot.

5.2. Manual de administrador

5.2.1. Requisitos

Los requisitos son idénticos a los del usuario básico, con la excepción de que es necesaria una cuenta de administrador para poder acceder a las funcionalidades específicas de gestión de documentos.

Un administrador puede realizar todas las operaciones descritas en el manual de usuario básico, excepto el registro, ya que un administrador no puede registrarse como administrador desde la aplicación.

5.2.2. Gestionar documentos

5.2.2.1. Modificar documentos

Para modificar documentos hay que repetir los pasos descritos en la sección 5.1.5 para acceder al documento que se desea modificar. Una vez en el documento, solo hay que pinchar en el lapicero que aparece para editar el documento (*Figura 52*).

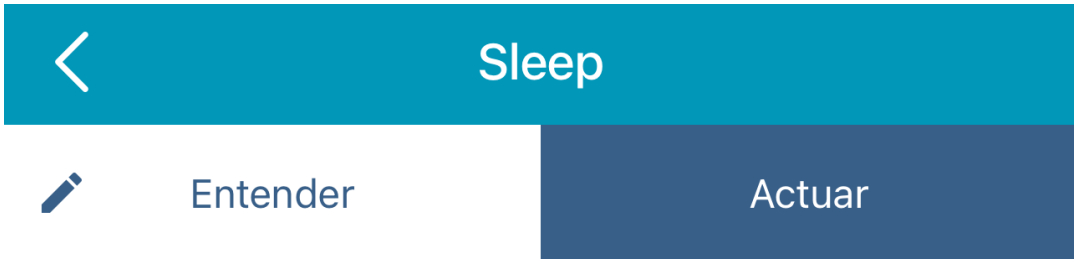

Figura 52.- Lapicero para editar los documentos.

Una vez editado, si pulsamos en "Guardar", se guardará la modificación (*Figura 53*).

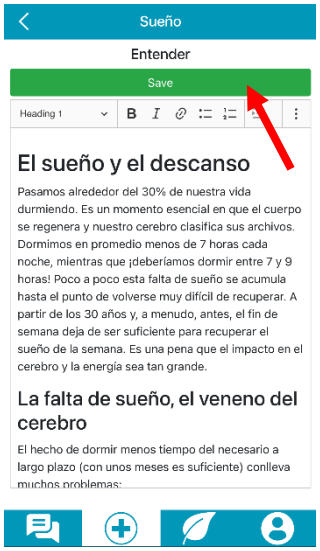

Figura 53.- Guardar documento editado.

También se puede acceder a los documentos desde la sección de gestionar documentos, en la que aparece el listado de todos los documentos disponibles sea cual sea el idioma en el que se encuentren. Para acceder a esta sección, tan solo hay que pulsar el botón que aparece en las secciones de "Salud" y "Vits&Mins", que dice "Gestionar Documentos" (*Figura 54*).

Figura 54.- Sección de "Salud" para administradores.

Una vez dentro, aparecerán todos los documentos disponibles (*Figura 55*).

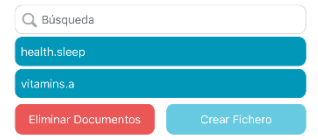

Figura 55.- Documentos disponibles en la aplicación.

Pinchando en uno de ellos, podremos ver los idiomas en los que se encuentra disponible, pinchando en el idioma del respectivo documento, navegaremos a ver el documento y podremos modificarlo de la misma manera que se explica anteriormente (*Figura 56*).

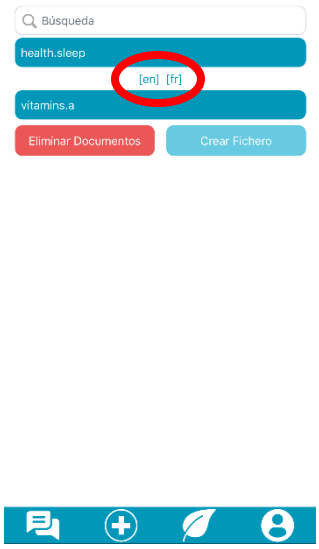

Figura 56.- Documentos disponibles por idiomas.

5.2.2.2. *Crear documentos*

Para crear documentos hay que ir a la sección de gestionar documentos siguiendo el mismo proceso que se menciona en la sección anterior. Una vez allí, seleccionamos el botón que dice "Crear Fichero" (*Figura 57*).

Figura 57.- Botón de crear documento.

Ahora tenemos que rellenar los campos que van a definir el documento (*Figura 58*):

- **Nombre:** identificador para agrupar aquellos documentos en distintos idiomas que hacen referencia a la misma información.
- **Título:** nombre que aparecerá encabezando el documento en la aplicación, en el idioma para el que se prepara el documento.
- **Idioma:** elegir entre los disponibles.

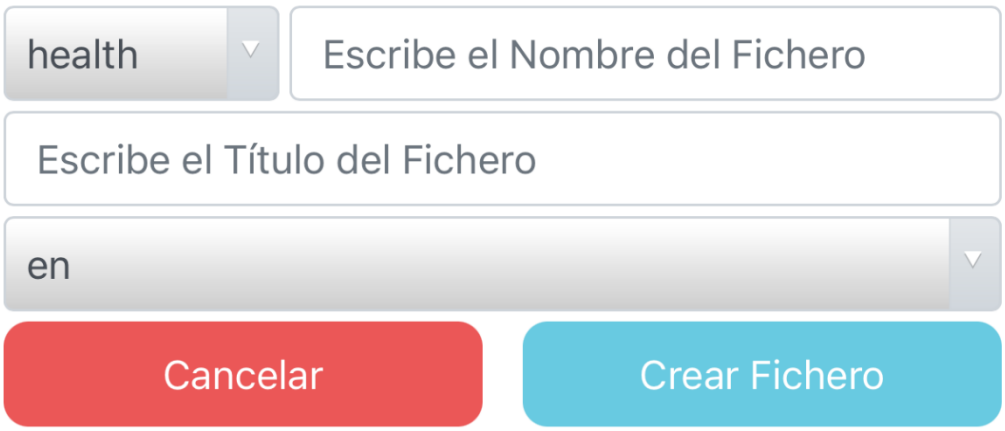

Figura 58.- Campos a rellenar al crear un documento.

No se pueden crear dos documentos con el mismo nombre e idioma.

Si hemos seleccionado que se trata de un documento de salud (*Figura 59*), tendremos que rellenar dos secciones (Entender y Actuar), pasando de una sección a otra haciendo clic en "Siguiente", en caso contrario, para las vitaminas solo deberemos completar una única sección (*Figura 60*). Para guardar el documento en el servidor, pulsaremos en "Crear".

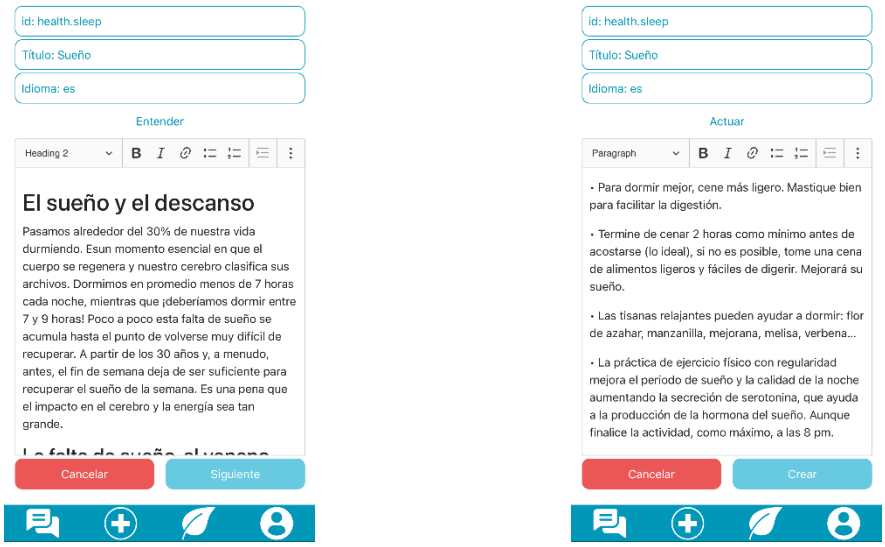

Figura 59.- Creando un documento de salud.

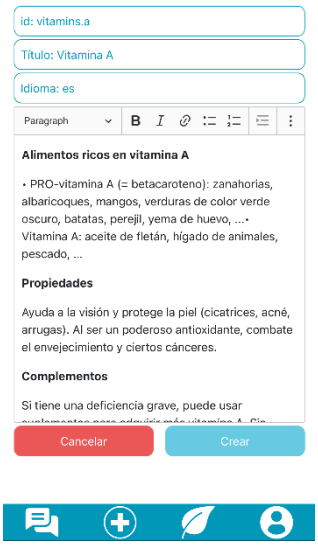

Figura 60.- Creando un documento de vitaminas.

5.2.2.3. Eliminar documentos

Por último, el proceso de eliminación de documentos. Como ya hemos realizado anteriormente, accederemos a la sección de gestión de documentos. Una vez allí, pincharemos en el botón que dice "Eliminar Documentos" (*Figura 61*).

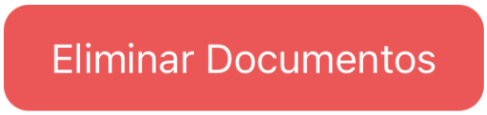

Figura 61.- Botón para eliminar documentos.

Aparecerán unos cuadrados para chequear los documentos que se desean eliminar. Se pueden elegir tanto documentos individuales como todos los documentos de un mismo nombre, el único límite es el número de documentos disponibles en ese momento. Para confirmar la eliminación pulsaremos el botón que dice “Confirmar Eliminación” (*Figura 62*).

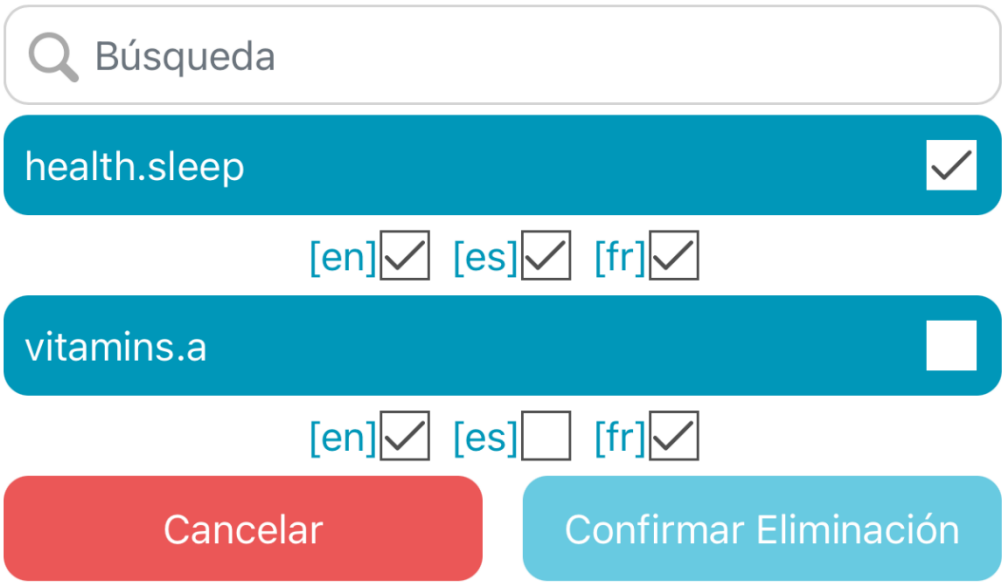

Figura 62.- Eliminación de documentos.

6. Conclusiones y líneas futuras

6.1. Conclusiones

El impacto económico consecuencia de la falta de salud de los trabajadores requiere de la toma de medidas para revertir la situación, tanto por los propios trabajadores, que se encontrarán más felices por tener una mejor salud, como por la economía, ya que un trabajador más sano será más probable que alcance su máximo potencial y obtendrá un mayor rendimiento. Esto supondrá una mayor rentabilidad y una mejora económica.

Se ha observado que optar por el desarrollo aplicaciones híbridas permite obtener las ventajas de las aplicaciones web, como son la sencillez de su programación y la capacidad multiplataforma. Siendo en este último aspecto incluso mejor las híbridas, ya que evitan posibles incompatibilidades con ciertos navegadores. Además, se suman las ventajas de las aplicaciones nativas, como son la gran personalización de la interfaz de usuario completa y la capacidad de control sobre todas las características del dispositivo. Añadiendo que las desventajas de las aplicaciones web y de las aplicaciones nativas son eliminadas en las aplicaciones híbridas, hacen de estas la alternativa perfecta para el desarrollo de aplicaciones móviles.

Con el objetivo de ayudar a los trabajadores a alcanzar su máximo potencial nació, en la empresa Ingage, la idea diseñar una aplicación atractiva, fácil de usar y que, mediante el uso de un *chatbot*, ofrecer un servicio completamente personalizado y adaptado a las necesidades de los trabajadores.

En este Trabajo de Fin de Grado se ha abordado la parte de la aplicación correspondiente con el diseño y desarrollo de la parte del *frontend*. Se han elaborado todos los requisitos en conjunto con el equipo en las reuniones iniciales y, el posterior desarrollo, de manera concurrente con Diego Alloza González, quien ha sido el encargado de elaborar el *backend* [5]. La aplicación fue pensada para diferenciar claramente la parte del cliente de la parte del servidor y, como consecuencia, ha sido necesaria la elaboración de una API robusta que permitiera la transmisión de información entre las dos partes.

Para el desarrollo de la aplicación se utilizó el *framework* de Angular, que ha permitido crear una SPA. Con esta SPA se han evitado llamadas innecesarias al *backend*, por estar todos los recursos de la aplicación ya cargados en la solicitud inicial. El uso de Angular junto con IONIC Capacitor ha habilitado la posibilidad de obtención de una aplicación híbrida fácilmente personalizable.

Además, gracias a haber realizado este trabajo junto con las prácticas de empresa, han sido tenidas en cuenta prácticas habituales de las empresas de desarrollo software. Como ejemplo, la elaboración de los test unitarios que han permitido comprobar si el código tenía el comportamiento esperado. En conjunto con la integración continua (otra práctica habitual en las empresas), la elaboración iterativa e incremental seguida al utilizar Scrum ha resultado más sencilla porque cada vez que se ha añadido una nueva funcionalidad se ha podido testear toda la aplicación y comprobar que tanto las nuevas funcionalidades como las antiguas funcionaran y obtener el correspondiente reporte. Como último ejemplo de práctica habitual en las empresas, el despliegue de la aplicación en contenedores que envuelvan las diferentes partes de la aplicación, facilitándolo en gran medida.

El empleo de notificaciones también ha sido una parte importante de la aplicación, porque es la que invita al usuario a volver a la aplicación cuando este no la está utilizando. La conexión de un evento externo a la aplicación (notificación) con el resto de la aplicación ha sido complicado por la falta de documentación disponible y, por la dificultad del testeo de las propias notificaciones,

que han requerido de continuas pruebas para entender realmente cómo gestiona Android la creación y lanzamiento de las notificaciones.

6.2. Líneas futuras

La realización de esta primera versión de la aplicación *Full Power* nos ofrece varias líneas de trabajo que se pueden seguir.

En primer lugar, respecto al trabajo realizado, la posibilidad de mejorar la automatización de las funciones existentes, creando funciones propias que contengan las llamadas a las funciones de los servicios ofrecidos en las librerías de Angular. De tal manera que un cambio en una función, por ejemplo, relacionada con el método GET ofrecida en la librería `http`, solo afectaría a mi función `get` que sería la única que utilizaría ese método, en lugar de necesitar cambiar cada vez que se llama al servicio `httpClient`. Ya que el resto de los métodos llamarán a mi `get` personalizado.

Un segundo aspecto importante es desarrollo de los test end-to-end. Consiguiendo automatizar estos test, se alcanzará un mayor control sobre los posibles defectos de la aplicación. Requiere la utilización de algún *framework* de testeo, como Selenium [28], que permita abrir un navegador o la aplicación de Android y realizar las actividades que realizaría cualquier usuario, comprobando que los resultados son los esperados en cada momento. Como también permiten un desarrollo iterativo e incremental, son ideales para realizar la integración continua de una manera más robusta.

Para tener una aplicación con mayor alcance, será necesario convertir la aplicación al formato de iOS. Se puede realizar de manera análoga al proceso de Android empleando IONIC Capacitor. No obstante, no se ha realizado para el presente trabajo por el elevado coste que suponen las licencias para desarrollar en Apple, además de necesitar dispositivos de la marca que, en ese momento, no se encontraban a nuestro alcance.

Con vistas a la inclusión de nuevas funcionalidades: creación de cuenta con una mayor cantidad de datos, la posibilidad de eliminar la cuenta de usuario, la creación de una sección con estadísticas a recopilar del usuario y desarrollar el carácter socializador de la aplicación.

6.2.1. Creación de cuenta con una mayor cantidad de datos

Actualmente, solo es posible crearse una cuenta añadiendo el nombre de usuario y eligiendo una contraseña. Sería de utilidad poder añadir el nombre y los apellidos del usuario, la dirección de correo electrónico. Incluso permitir la conexión con a través de otras plataformas como Google o Facebook, de manera similar a otras aplicaciones. Facilitando a los usuarios la conexión con la aplicación.

6.2.2. Eliminar cuenta de usuario

La posibilidad de eliminar la cuenta de usuario por parte del usuario que desee borrar su cuenta, ya que solo es posible siendo administrador en el *backend*. Se ahorraría tiempo, ya que no sería necesario contactar con el administrador cada vez que un usuario quiera dejar de tener cuenta en *Full Power*.

6.2.3. Sección de estadísticas

Una sección de estadísticas permitiría a los usuarios conocer la progresión que han tenido como consecuencia de haber seguido los consejos de la aplicación. Estas estadísticas tendrían que ser elegidas por el experto, ya que es quien comprende cuáles son los datos más importantes y los que más interés tendrían para los usuarios.

6.2.4. Carácter socializador de la aplicación

Desarrollar el carácter socializador de la aplicación, es decir, permitir a los usuarios compartir sus progresos con sus compañeros fuera de la aplicación. Aunque no solo eso, también permitir la creación de grupos dentro de la aplicación con los compañeros de trabajo, en los que se pueda ver el perfil del resto de compañeros o su progreso. Esto sería un gran punto positivo ya que puede provocar la mejora en la relación con los compañeros y servir como motivación para superarse a sí mismo.

Referencias

[1] «WHO technical meeting on sleep and health», World Health Organization Regional Office for Europe European Centre for Environment and Health Bonn Office, ene. 2004.

[2] «The Costs of Insufficient Sleep». https://www.rand.org/randeurope/research/projects/the-value-of-the-sleep-economy.html (accedido may 26, 2021).

[3] «Nutrition», *International Labour Organization*, jun. 15, 2009. http://www.ilo.org/global/topics/safety-and-health-at-work/areasofwork/WCMS_118393/lang--en/index.htm (accedido may 26, 2021).

[4] International Labour Office y L. I. and O. S. and H. B. Labour Administration, *Workplace stress: a collective challenge*. Geneva: ILO, 2016.

[5] D. Alloza González, «Diseño y desarrollo de un API y su Back-end para una aplicación para la mejora de la salud basada en un chatbot.» Universidad de Valladolid, 2021.

[6] K. Schwaber y M. Beedle, *Agile software development with Scrum*, vol. 1. Prentice Hall Upper Saddle River, 2002.

[7] Atlassian, «Jira | Issue & Project Tracking Software», *Atlassian*. https://www.atlassian.com/software/jira (accedido may 27, 2021).

[8] A. Charland y B. Leroux, «Mobile application development: web vs. native», *Commun. ACM*, vol. 54, n.º 5, pp. 49-53, may 2011, doi: 10.1145/1941487.1941504.

[9] «Declare permissions», *Chrome Developers*. https://developer.chrome.com/docs/extensions/mv3/declare_permissions/ (accedido jun. 20, 2021).

[10] D. Sheppard y D. Sheppard, *Beginning progressive web app development*. Springer, 2017.

[11] «Ionic Article: What is Hybrid App Development?», feb. 06, 2019. https://ionic.io/resources/articles/what-is-hybrid-app-development (accedido abr. 30, 2021).

[12] «NotificationEvent - Web APIs | MDN». https://developer.mozilla.org/en-US/docs/Web/API/NotificationEvent (accedido jun. 21, 2021).

[13] «IEEE Recommended Practice for Software Requirements Specifications», *IEEE Std 830-1998*, pp. 1-40, oct. 1998, doi: 10.1109/IEEESTD.1998.88286.

[14] «Android Developers», *Android Developers*. https://developer.android.com/?hl=es (accedido may 27, 2021).

[15] R. Stephens, *Beginning Software Engineering*, John Wiley&Sons. 2015.

[16] «Figma: the collaborative interface design tool.», *Figma*. https://www.figma.com/ (accedido abr. 14, 2021).

[17] «Angular». https://angular.io/ (accedido abr. 20, 2021).

[18] «Apache Cordova». https://cordova.apache.org/ (accedido abr. 20, 2021).

[19] «Capacitor: Cross-platform native runtime for web apps», *Capacitor: Cross-platform native runtime for web apps*. https://capacitorjs.com/ (accedido abr. 28, 2021).

[20] «Bootstrap · The most popular HTML, CSS, and JS library in the world.» https://getbootstrap.com/ (accedido abr. 22, 2021).

[21] «Rich text editor of tomorrow | CKEditor 5». https://ckeditor.com/ckeditor-5/ (accedido abr. 22, 2021).

[22] «Codecov - The Leading Code Coverage Solution», *Codecov*. https://about.codecov.io/ (accedido jun. 22, 2021).

[23] «Empowering App Development for Developers | Docker». https://www.docker.com/ (accedido jun. 22, 2021).

[24] «Visual Studio Code - Code Editing. Redefined». https://code.visualstudio.com/ (accedido abr. 29, 2021).

[25] «Download Android Studio and SDK tools | Android Studio», *Android Developers*. https://developer.android.com/studio (accedido jun. 23, 2021).

[26] «Docker Hub». https://hub.docker.com/ (accedido jun. 22, 2021).
[27] auth0.com, «JWT.IO». http://jwt.io/ (accedido jul. 01, 2021).
[28] «SeleniumHQ Browser Automation». https://www.selenium.dev/ (accedido jun. 22, 2021).