UNIVERSIDAD DE VALLADOLID
E.T.S.I DE TELECOMUNICACIÓN

TRABAJO FINAL DE GRADO
GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Aplicación móvil de hábitos saludables para personas con problemas de salud mental: nutrición y deporte

Autor: Marcos Novoa González
Tutor: Alfonso Bahillo Martínez

11 de septiembre de 2023

TÍTULO:	Aplicación móvil de hábitos saludables para personas con problemas de salud mental: nutrición y deporte
AUTOR:	Marcos Novoa González
TUTOR:	Alfonso Bahillo Martínez
DEPARTAMENTO:	Teoría de la Señal y Comunicaciones e Ingeniería Telemática

TRIBUNAL

PRESIDENTE:	D. Rubén M. Lorenzo Toledo
VOCAL:	D. Juan C. Aguado Manzano
SECRETARIO:	D. Alfonso Bahillo Martínez
SUPLENTE:	D. Ramón J. Durán Barroso
SUPLENTE:	D.ª Noemí Merayo Álvarez
FECHA:	18/09/2023
CALIFICACIÓN:	

Resumen

La salud mental es un aspecto fundamental de nuestro bienestar general y afecta directamente nuestra capacidad para funcionar en la vida cotidiana. En los últimos años, se está poniendo más atención que nunca sobre esto, reconociendo que la atención a la salud mental es crucial, no solo para prevenir problemas de salud mental, sino también para tratar y manejar adecuadamente aquellos que ya han sido diagnosticados. Un aspecto a destacar sobre los tratamientos es cómo la nutrición equilibrada y el ejercicio regular pueden tener un impacto positivo en la salud mental, reduciendo los síntomas de problemas como la ansiedad y la depresión. Por lo tanto, la inclusión de hábitos saludables en los planes de tratamiento de pacientes con este tipo de enfermedades resulta muy interesante y prometedor.

El proyecto "Food & Move"tiene como objetivo promover y facilitar la adopción de hábitos saludables en la vida diaria de los pacientes mediante el uso de tecnología. Para ello, se ha desarrollado una plataforma web para los profesionales de la salud y una aplicación móvil para los pacientes. La plataforma permitirá a los enfermeros establecer planes de dieta y ejercicio personalizados y realizar un seguimiento de los resultados del tratamiento en cada paciente, mientras que la aplicación móvil permitirá a los pacientes acceder a la información de sus planes de dieta y ejercicios y consultar recursos multimedia para apoyar sus hábitos saludables.

En este proyecto se han usado tecnologías como Angular e Ionic para el desarrollo de una aplicación multiplataforma, también resulta importante destacar la elección de MongoDB como base de datos para el almacenamiento de la información relacionada con los pacientes y los tratamientos. A través de esta combinación de tecnologías, se busca

mejorar la eficacia de los tratamientos de salud mental y, al mismo tiempo, brindar a los pacientes una herramienta accesible, sencilla de manejar, puesto que los pacientes tendrán variedad de habilidades con la tecnología, y motivadora para tratar de mejorar su calidad de vida mediante hábitos saludables.

A lo largo del desarrollo del proyecto, el autor ha adquirido conocimientos en diferentes lenguajes de programación, enfoque full stack para el desarrollo de aplicaciones y ha experimentado el trabajo en equipo. Además, enfrentó desafíos en el aprendizaje autodidacta de nuevos lenguajes y tecnologías. La satisfacción personal ha sido un motor impulsor, ya que el autor se ha sentido motivado por contribuir al bienestar de personas con problemas de salud mental, sabiendo que su trabajo facilita el camino hacia una vida más saludable y equilibrada para estos pacientes.

PALABRAS CLAVE: Aplicación móvil, Salud mental, Hábitos saludables, Dieta Semanal, Actividad física, Ionic, Multiplataforma, Desarrollo Full Stack, Frontend, Backend.

Abstract

Mental health is a fundamental aspect of our overall well-being and directly affects our ability to function in everyday life. In recent years, more attention than ever is being placed on this, recognizing that mental health care is crucial, not only for preventing mental health issues but also for properly treating and managing those who have already been diagnosed. One notable aspect regarding treatments is how balanced nutrition and regular exercise can have a positive impact on mental health, reducing symptoms of problems such as anxiety and depression. Therefore, the inclusion of healthy habits in the treatment plans for patients with these types of disorders is very interesting and promising.

The "Food & Move"project aims to promote and facilitate the adoption of healthy habits in patients daily lives through the use of technology. To achieve this, a web platform for healthcare professionals and a mobile application for patients have been developed.The platform will allow nurses to establish personalized diet and exercise plans and track treatment outcomes for each patient, while the mobile app will enable patients to access information about their diet and exercise plans and consult multimedia resources to support their healthy habits.

In this project, technologies like Angular and Ionic have been utilized for the development of a cross-platform application. It's worth noting the selection of MongoDB as the database for storing information related to patients and treatments. Through this combination of technologies, the goal is to enhance the effectiveness of mental health treatments and, simultaneously, provide patients with an accessible tool that is easy to manage, considering patients may have

varying levels of technological skills, and motivating them to strive for improving their quality of life through healthy habits.

During the project's development, the author has acquired knowledge in different programming languages, embraced a full-stack approach to application development, and gained experience working in a team. Additionally, the author faced challenges in self-learning new languages and technologies. Personal satisfaction has been a driving force, as the author has felt motivated to contribute to the well-being of individuals with mental health issues, knowing that their work facilitates a path towards a healthier and more balanced life for these patients.

KEYWORDS: Mobile application, Mental health, Healthy habits,Weekly diet, Multiplatform, Physical activity, Ionic, Full Stack development, Frontend, Backend.

Agradecimientos

Ninguna persona y ningún trabajo son una isla, por eso quiero expresar mi más sincero agradecimiento a todos aquellos que han sido parte fundamental en mi camino hasta este punto.

En primer lugar, quiero extender mi mayor gratitud a mis queridos padres. Vuestro constante apoyo, ejemplo y dedicación han sido un faro de inspiración en mi vida. Vuestra capacidad para crear un ambiente propicio, donde el estudio y el amor florecen, ha sido fundamental para mi desarrollo académico y personal. Asimismo, agradezco a mi hermano Diego por su compañía y aliento en cada etapa de este recorrido. Además, quiero expresar un agradecimiento especial a mis abuelos y también al resto de mi familia, cuyo cariño y respaldo han sido un pilar esencial en mi camino.

A lo largo de mi trayecto, he tenido el privilegio de contar con grandes amigos y el apoyo de profesores y personas excepcionales. Desde los días en el colegio Maristas San José en León, hasta mi paso por la Universidad de Valladolid y el Colegio Mayor Menéndez Pelayo, pasando por instituciones como el Club Balonmano Ademar León, entre otros muchos sitios donde he sido feliz. Mi camino por estos lugares me ha dejado grandes amigos, quienes han llenado mi vida de risas, apoyo y momentos inolvidables. Quiero aprovechar estos agradecimientos para decir que vuestra amistad es un regalo valioso que la vida me ha brindado.

También, quiero expresar mi gratitud a Dios, quien me ha mostrado que cada persona tiene su propia manera de relacionarse con Él y que trabajar con amor, honradez y respetando sus tiempos es una senda valiosa. No puedo pasar por alto agradecerme a mí mismo por el sacrificio, el esfuerzo constante y el crecimiento que he experimentado

en este proceso que son testigos de mi dedicación y perseverancia en los momentos de desafío, recordando que el camino hacia el éxito rara vez es sin obstáculos y manteniendo mi mirada en el horizonte y mi fe en mi propia fortaleza, evitando cuestionar mi camino cuando se volvía arduo. Ser Ingeniero de Telecomunicaciones es un gran logro, pero es solo el principio de mi viaje, y estoy decidido a apuntar cada vez más alto, fortaleciendo mi determinación con dedicación y los valores de humildad, sencillez y modestia que se me grabaron como lema de mi querido colegio Maristas San José.

Finalmente, quiero cerrar estos agradecimientos con una frase que trato de hacer resonar profundamente en mi ser: "Trata de dejar este mundo un poco más bonito de lo que lo encontraste."Me gustaría que de aquí en adelante esta cita sea mi brújula, recordándome la importancia de contribuir positivamente en cada paso que doy.

ÍNDICE GENERAL

«Para mi
familia»

—

CAPÍTULO 1

INTRODUCCIÓN

"Da tu primer paso con fe.
No tienes por qué ver toda la escalera.
Basta con que subas el primer peldaño."
— Martin Luther King, *1963*.

1.1. Motivación

La salud mental es un aspecto fundamental de nuestro bienestar general y afecta directamente nuestra capacidad para funcionar en la vida cotidiana. La atención a la salud mental es importante, no solo para prevenir problemas de salud mental, sino también para tratar y manejar adecuadamente aquellos que ya han sido diagnosticados. En este contexto, la nutrición y el deporte se han identificado como dos hábitos saludables que pueden tener un impacto positivo en la salud mental. Numerosos estudios han demostrado que una dieta equilibrada y la práctica regular de ejercicio físico pueden ayudar a reducir los síntomas de problemas de salud mental, como la

ansiedad y la depresión. Por lo tanto, la inclusión de hábitos saludables en los planes de tratamiento de pacientes con este tipo de enfermedades resulta muy interesante.

Actualmente, muchos centros de tratamiento de salud mental han comenzado a incorporar hábitos saludables como parte de su enfoque terapéutico, incluyendo la promoción de una dieta saludable y la actividad física regular como parte del plan de tratamiento. Sin embargo, la implementación de estos hábitos saludables puede resultar difícil para los pacientes, especialmente si no tienen acceso a recursos adecuados o no están motivados para seguirlos. Por lo tanto, resulta interesante explorar cómo la tecnología puede ayudar a implementar estos hábitos de forma más sencilla y eficaz. Las aplicaciones móviles, la monitorización de la actividad física y la orientación en línea de profesionales de la salud pueden ser herramientas útiles para ayudar a los pacientes a adoptar y mantener hábitos saludables en su vida diaria. La implementación de estos enfoques innovadores y basados en la tecnología puede mejorar significativamente la eficacia de los tratamientos de salud mental y la calidad de vida de los pacientes.

1.2. Presentación del Proyecto

El proyecto se llama Food&Move y es una iniciativa diseñada para mejorar la salud mental de los pacientes que reciben tratamiento en el Servicio de Psiquiatría y Salud Mental del Complejo Público Asistencial de Zamora. El proyecto se enfoca en la promoción de hábitos saludables, como una alimentación equilibrada y la práctica regular de ejercicio físico, para ayudar a los pacientes a manejar y reducir los síntomas de problemas de salud mental. Para ello, se han desarrollado diversas herramientas tecnológicas que serán utilizadas tanto por los pacientes como por los enfermeros encargados de su tratamiento.

Una de las herramientas clave del proyecto Food&Move es una plataforma web, que permitirá a los enfermeros encargados del tratamiento de los pacientes anotar y hacer un seguimiento de ciertos parámetros de salud, como el peso o el perímetro abdominal. Además, la plataforma permitirá a los enfermeros establecer planes de dieta y ejercicio personalizados para cada paciente, según sus necesidades específicas y la evolución en el tiempo de los parámetros monitorizados. Los pacientes podrán acceder a esta información desde una aplicación móvil, que se sincronizará con la plataforma web. Esta herramienta no solo permitirá a los enfermeros hacer un seguimiento más

preciso y efectivo de la evolución de los pacientes, sino que también facilitará el acceso de los pacientes a planes personalizados de dieta y ejercicio, lo que ayudará a mejorar la adherencia al tratamiento y la eficacia del mismo.

La otra herramienta clave del proyecto Food&Move es una aplicación móvil diseñada para ayudar a los pacientes a llevar un seguimiento de su dieta y ejercicio. Esta es la herramienta que se va a desarrollar en este TFG y la aplicación permitirá a los pacientes ver sus planes de dieta y ejercicio personalizados, y les mostrará diversos recursos multimedia que les ayuden a realizar la dieta y ejercicios. Además, la aplicación incluirá una lista de la compra con los productos que deben adquirir para seguir su plan de dieta, lo que ayudará a los pacientes a mantenerse organizados y motivados. Se ha puesto especial énfasis en la simplicidad de la aplicación, reconociendo que el público objetivo puede tener niveles variados de competencia digital. La interfaz de usuario ha sido diseñada de manera intuitiva y amigable, con una navegación sencilla que permite a los pacientes acceder fácilmente a la información esencial. Se ha minimizado la complejidad técnica para asegurar que cualquier persona, independientemente de su familiaridad con la tecnología, pueda utilizar la aplicación sin dificultades. Este enfoque en la simplicidad busca eliminar barreras y maximizar la accesibilidad, garantizando que incluso aquellos con menor experiencia en el ámbito digital puedan beneficiarse plenamente de la herramienta y adoptar hábitos saludables de manera efectiva. En resumen, esta herramienta busca proporcionar una forma fácil y accesible para que los pacientes sigan su tratamiento y se mantengan motivados y comprometidos con este.

Figura 1.1.: Logo de Food&Move

CAPÍTULO 2

ENTORNO DE DESARROLLO

" El cambio es muy difícil al principio,
desorganizado en el medio,
y maravilloso al final."

— Robin Sharma, *2019*.

En esta sección se van a presentar y describir las herramientas tecnológicas usadas para el desarrollo del proyecto. En este caso, se optó por utilizar Angular e Ionic como entorno de desarrollo para la creación de la web y de la aplicación móvil, respectivamente. Angular es un framework de JavaScript ampliamente utilizado para el desarrollo de aplicaciones web y móviles. Proporciona una estructura robusta y modular que facilita la creación de aplicaciones de alta calidad y buen rendimiento. Por otro lado, Ionic es un framework de desarrollo de aplicaciones multiplataforma que permite la creación de aplicaciones móviles utilizando tecnologías web como HTML, SCSS y TypeScript. Esta combinación de Angular e Ionic proporciona una base sólida y flexible para desarrollar una aplicación móvil eficiente y altamente funcional.

Además, para el almacenamiento de datos se eligió MongoDB como la base de datos para la aplicación. MongoDB es una base de datos NoSQL orientada a documentos, que ofrece una estructura de datos flexible y escalable. Su capacidad para manejar grandes volúmenes de datos y su enfoque en la escalabilidad horizontal lo convierten en una opción adecuada para el almacenamiento de información en aplicaciones modernas. Al aprovechar MongoDB, la aplicación será capaz de gestionar eficientemente los datos relacionados con los usuarios, la dieta, los ejercicios y otros aspectos relevantes del proyecto Food&Move, permitiendo un almacenamiento y recuperación de información eficiente y rápido.

2.1. Angular

Angular es un framework de JavaScript desarrollado y mantenido por Google.[13] Es ampliamente utilizado en el desarrollo de aplicaciones web y móviles, permitiendo la creación de interfaces de usuario dinámicas y de alto rendimiento. En este proyecto, se ha seleccionado Angular como el entorno de desarrollo principal para la creación de la aplicación móvil Food&Move.

Angular utiliza una arquitectura de aplicación de tipo MVC (Modelo-Vista-Controlador), lo que facilita la separación de responsabilidades y el desarrollo estructurado. Esta arquitectura promueve la modularidad y la reutilización de código, lo cual es especialmente útil en proyectos complejos. Al seguir el patrón MVC, se divide la aplicación en tres componentes principales:

- Modelo (Model): Representa los datos y la lógica de negocio de la aplicación. En Angular, el modelo se define utilizando clases y estructuras de datos.
- Vista (View): Es la interfaz de usuario con la que los usuarios interactúan. En Angular, la vista se define mediante plantillas HTML y se puede extender con directivas para agregar comportamiento dinámico.
- Controlador (Controller): Es el intermediario entre el modelo y la vista. En Angular, el controlador se implementa mediante componentes, que encapsulan la lógica y la presentación de una parte específica de la aplicación.

Angular también utiliza TypeScript, un lenguaje de programación basado en JavaScript que agrega tipado estático y características avanzadas al desarrollo de

aplicaciones.[12] El tipado estático se refiere a la capacidad de especificar explícitamente los tipos de datos de las variables, parámetros de función y otros elementos en el código. En JavaScript, que es un lenguaje de tipado dinámico, no es necesario declarar los tipos de datos, lo que puede llevar a errores difíciles de detectar y depurar. En cambio, TypeScript permite declarar los tipos de datos de manera explícita. Esto significa que se pueden definir variables con tipos específicos. TypeScript proporciona beneficios como la detección de errores en tiempo de compilación, el autocompletado de código y la mejor legibilidad del mismo.

Una de las características más destacadas de Angular es su enfoque en la creación de aplicaciones SPA (Single-Page Application).[13] Esto significa que la aplicación se carga una sola vez en el navegador y, a partir de ese momento, se actualizan solo las partes necesarias sin necesidad de recargar la página completa. Esto mejora significativamente la experiencia del usuario al reducir los tiempos de carga y mejorar la capacidad de respuesta de la aplicación.

Además, Angular cuenta con una amplia gama de bibliotecas y herramientas que facilitan la integración con otros sistemas y servicios. Al aprovechar estas herramientas, se puede mejorar la eficiencia y la productividad durante el desarrollo de la aplicación Food&Move.

Figura 2.1.: Logo de Angular

2.2. Ionic

Ionic es un framework de desarrollo de aplicaciones híbridas que permite crear aplicaciones móviles utilizando tecnologías web como HTML, SCSS y TypeScript. Es ampliamente utilizado en el desarrollo de aplicaciones móviles debido a su enfoque en la creación de aplicaciones multiplataforma, su amplia gama de componentes predefinidos y su fácil integración con otros frameworks como Angular.

Ionic destaca por su capacidad para desarrollar aplicaciones móviles multiplataforma, lo que significa que una sola base de código puede ejecutarse en diferentes sistemas operativos móviles, como iOS y Android. Esta característica reduce el tiempo y los recursos necesarios para desarrollar y mantener aplicaciones separadas para cada plataforma. Además, Ionic utiliza tecnologías web estándar como HTML, SCSS y TypeScript, lo que brinda a los desarrolladores la ventaja de aprovechar sus conocimientos existentes en estas tecnologías.[18] Cabe destacar, que estas características de Ionic han sido claves para su elección en el proyecto Food&Move, ya que será sencillo integrar la app con el entorno web desarrollado en Angular y la aplicación estará disponible para usuarios de Android e iOS.

El framework Ionic se basa en Angular. La integración de Ionic con Angular permite a los desarrolladores crear interfaces de usuario interactivas y funcionales utilizando un conjunto de herramientas poderosas. Ionic proporciona una amplia biblioteca de componentes predefinidos, como botones, listas, tarjetas, formularios y más, que siguen las mejores prácticas de diseño de aplicaciones móviles, facilitan la tarea al desarrollador y garantiza una apariencia coherente y una experiencia de usuario agradable en todas las plataformas.[26]

Una de las ventajas clave de Ionic es su enfoque en el diseño de aplicaciones móviles nativas. Las aplicaciones creadas con Ionic se ven y se sienten como aplicaciones móviles nativas, brindando una experiencia de usuario de alta calidad. Ionic utiliza estilos y animaciones nativas para garantizar una apariencia y un rendimiento excelentes en diferentes dispositivos móviles.[27]

Ionic también ofrece una variedad de plugins y extensiones que permiten acceder a funcionalidades nativas de los dispositivos móviles, como la cámara, el GPS y las notificaciones push. Estos plugins facilitan la integración de características avanzadas en las aplicaciones y brindan a los desarrolladores la capacidad de aprovechar al máximo las capacidades de los dispositivos móviles.[15]

La Figura 2.2 ilustra que como se ha explicado, Ionic hace uso de Angular, HTML y SCSS para generar aplicaciones móviles tanto en Android como en iOS. Angular actúa como el framework principal, brindando la estructura y la arquitectura necesaria para desarrollar aplicaciones robustas y escalables. HTML se utiliza para definir la estructura y el contenido de las páginas de la aplicación, mientras que SCSS se encarga de la presentación y el estilo visual de la aplicación. Con esta combinación de tecnologías, Ionic es capaz de generar aplicaciones móviles con una apariencia y un rendimiento nativos en ambas plataformas, permitiendo a los desarrolladores aprovechar su experiencia en el desarrollo web para crear aplicaciones multiplataforma de alta calidad.

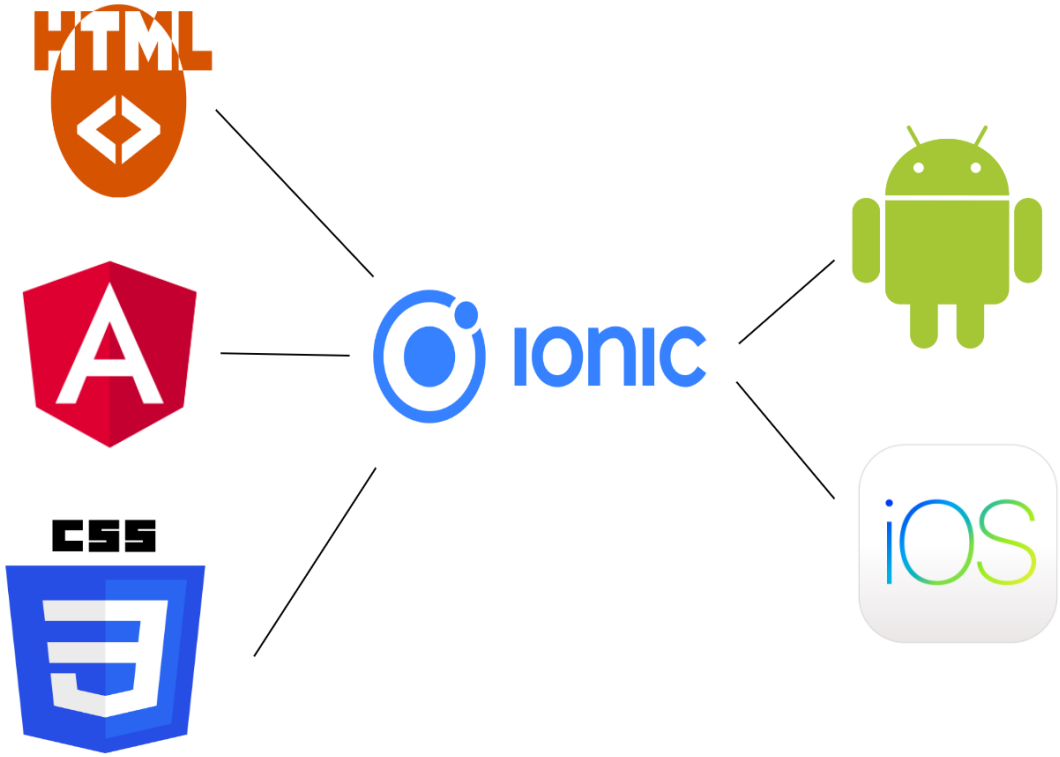


Figura 2.2.: Entorno de desarrollo de la app en Ionic

En este proyecto, se ha utilizado la versión 7 de Ionic. La versión 7 de Ionic trae consigo importantes mejoras y actualizaciones en comparación con las versiones anteriores. Ofrece mejoras significativas en rendimiento, estabilidad y funcionalidades. Con esta versión, se ha podido aprovechar una arquitectura modular, actualizaciones en componentes visuales y animaciones, mejoras en usabilidad y accesibilidad, y nuevas funcionalidades para crear aplicaciones móviles modernas y multiplataforma de alta calidad.

Figura 2.3.: Logo de Ionic

2.3. MongoDB

MongoDB es una base de datos NoSQL de código abierto que se ha utilizado en el proyecto Food&Move. [20] A diferencia de las bases de datos relacionales tradicionales, MongoDB se basa en un modelo de documentos, lo que le brinda flexibilidad y escalabilidad para manejar grandes volúmenes de datos no estructurados. [23]

En MongoDB, los datos se almacenan en documentos BSON (Binary JSON), que son estructuras de datos similares a JSON pero en formato binario. Cada documento en MongoDB representa una entidad o un registro y puede contener diferentes campos y tipos de datos como se aprecia en la Figura 2.4. Esto permite almacenar y recuperar datos de manera eficiente, sin la necesidad de un esquema fijo predefinido, lo que brinda una mayor flexibilidad a medida que evoluciona el proyecto. JSON (JavaScript Object Notation) es un formato de datos ampliamente utilizado para representar datos estructurados en forma legible por humanos. En MongoDB, los documentos BSON son compatibles con JSON, lo que facilita la integración con aplicaciones que utilizan JSON como formato de intercambio de datos.

MongoDB ofrece características de escalabilidad y alta disponibilidad que son fundamentales para aplicaciones modernas. Con su arquitectura distribuida, permite escalar horizontalmente agregando más servidores para manejar cargas de trabajo crecientes. Además, MongoDB utiliza un enfoque de réplica, donde los datos se replican automáticamente en varios nodos, lo que garantiza la disponibilidad y la tolerancia a fallos.[8]

Otra de las características de MongoDB es su sistema de índices eficientes, los cuales permiten acelerar la ejecución de consultas incluso en conjuntos de datos grandes. Los índices en MongoDB funcionan de manera similar a los de las bases de datos relacionales, permitiendo un acceso rápido a los datos en función de campos específicos o criterios de búsqueda. Estos índices optimizan el rendimiento de las consultas al reducir el tiempo de búsqueda y minimizar la cantidad de datos que se

```
_id: ObjectId('64039ce67e74b057b1265648')
patient: ObjectId('6368120605980031bdc15fdc')
title: "Café"
description: "No poner mucho azúcar"
dish: "Principal"
▾ links: Array
    0: "https://www.bonka.es/como-preparar-cafe?gclid=Cj0KCQiA9YugBhCZARIsAACX…"
date: 2023-03-13T23:00:00.000+00:00
done: true
▾ ingredients: Array
  ▾ 0: Object
      name: "Leche"
      quantity: 0
      unit: ""
      isChecked: false
      _id: ObjectId('643e48addbc31b88738a4027')
  ▸ 1: Object
  ▸ 2: Object
__v: 0
meal: "Desayuno"
▸ videos: Array
rating: "Normal"
```


Figura 2.4.: Estructura de datos en MongoDB

deben examinar. Además, MongoDB ofrece capacidades de agregación que permiten realizar operaciones de análisis avanzadas directamente en la base de datos. Las agregaciones en MongoDB facilitan tareas como la agrupación de datos y el cálculo de estadísticas, lo que permite obtener información más compleja y útil sin necesidad de procesamiento adicional en la aplicación cliente.[23]

En el contexto del proyecto Food&Move, MongoDB se ha utilizado como base de datos para almacenar la toda la información necesasaria relacionada con los usuarios, las recetas, los datos de actividad física de los pacientes, etc. La elección de MongoDB se basó en su capacidad para manejar datos no estructurados, su escalabilidad y su rendimiento en aplicaciones con altos volúmenes de datos.[8]

Figura 2.5.: Logo de MongoDB

CAPÍTULO 3

LENGUAJES DE PROGRAMACIÓN

"El mundo entero se aparta
cuando ve pasar a un hombre
que sabe a dónde va"

— Antoine de Saint-Exupéry - El Principito, *1943*.

En este capítulo, se profundizará en el estudio de tres lenguajes de programación fundamentales en el desarrollo de la aplicación en Ionic: TypeScript, HTML y SCSS. Estos lenguajes desempeñan un papel central en la creación de aplicaciones móviles en Ionic.

Comenzando con TypeScript, este lenguaje se caracteriza por ser orientado a objetos y de tipado estático, basado en JavaScript. TypeScript ofrece características adicionales, como el tipado estático, interfaces y clases, que contribuyen a mejorar la calidad del código, su legibilidad y su mantenibilidad. Al compilar el código TypeScript, se genera código JavaScript compatible que se ejecutará en el entorno del dispositivo móvil.

Por su parte, HTML (HyperText Markup Language) es el lenguaje de marcado utilizado para estructurar y presentar el contenido en la web. Constituye el lenguaje fundamental para definir la estructura y los elementos visuales de las páginas web. En el contexto de Ionic, HTML se emplea para crear la interfaz de usuario de las aplicaciones móviles. Con una sintaxis clara y concisa, HTML permite definir componentes, elementos de formulario, diseños y otros elementos visuales de manera intuitiva.

Por último, se encuentra SCSS (Sassy CSS). SCSS es una extensión de CSS que aporta características y funcionalidades avanzadas al lenguaje de estilos utilizado en las aplicaciones web. Gracias a SCSS, los estilos pueden escribirse de forma modular, organizada y reutilizable, lo que facilita la gestión de estilos en proyectos de gran envergadura.

A lo largo de este capítulo, exploraremos las características principales de estos lenguajes y su integración en el desarrollo de aplicaciones móviles con Ionic. Este capítulo no pretende ser un manual detallado sobre estos lenguajes, sino una breve contextualización de su uso y características principales.

3.1. TypeScript

TypeScript es un lenguaje de programación de alto nivel, es decir, un lenguaje de programación que se acerca más al lenguaje natural y abstracto que al código de bajo nivel. Fue desarrollado por Microsoft y se basa en JavaScript. A diferencia de JavaScript, que es un lenguaje de programación interpretado, TypeScript es un lenguaje de programación compilado que agrega características adicionales y mejoras a JavaScript. Un lenguaje interpretado, como JavaScript, es aquel cuyo código fuente se ejecuta directamente por un intérprete en tiempo de ejecución, sin necesidad de compilarlo previamente. Por otro lado, TypeScript es un lenguaje compilado, lo que significa que el código fuente escrito en TypeScript debe ser compilado antes de ser ejecutado. El compilador de TypeScript traduce el código fuente a JavaScript, que luego puede ser interpretado y ejecutado por el navegador o el entorno correspondiente. La compilación de TypeScript ofrece varias ventajas, como la detección de errores en tiempo de compilación y la verificación estática de tipos, lo que ayuda a encontrar y corregir errores antes de que el código sea ejecutado. TypeScript se ha convertido en un lenguaje popular para el desarrollo web y, en particular, se usa para la construcción de aplicaciones en Angular como en este proyecto.

Como se mencionó en el Capítulo 2, TypeScript se distingue por su sistema de tipos estáticos, que permite asignar y declarar tipos a variables, parámetros y propiedades de objetos. Esto proporciona seguridad y precisión en el desarrollo, ya que el compilador de TypeScript verifica los tipos en tiempo de compilación, ayudando a detectar errores.[12] Además, TypeScript es un lenguaje orientado a objetos que admite clases, herencia, polimorfismo e interfaces. Las clases definen la estructura y el comportamiento de los objetos, incluyendo propiedades y métodos. La herencia permite que una clase herede propiedades y métodos de otra, fomentando la reutilización de código y la creación de jerarquías de clases.Las interfaces en TypeScript permiten definir contratos y especificaciones para la estructura de un objeto.Y por último, el polimorfismo permite tratar objetos de diferentes clases de manera uniforme.

El sistema de tipos de TypeScript se basa en un enfoque de tipado estructural. Esto significa que el tipo de una variable o objeto se determina por su forma o estructura, en lugar de su nombre explícito. Esto permite una mayor flexibilidad y reutilización de código, ya que los objetos que tienen una estructura similar se consideran compatibles, incluso si se definen en diferentes partes del código. TypeScript también tiene la capacidad de inferir tipos automáticamente en función de la asignación de valores. Esto significa que no es necesario especificar el tipo de cada variable o propiedad de objeto de forma explícita, ya que el compilador puede deducirlo a partir del contexto en el que se utiliza.

TypeScript es compatible con JavaScript, lo que significa que cualquier código JavaScript existente puede ejecutarse en un entorno TypeScript sin modificaciones. Esto facilita la migración gradual de proyectos JavaScript existentes a TypeScript, ya que se pueden aprovechar gradualmente las características y ventajas del lenguaje.

La utilidad y ventajas de TypeScript en el desarrollo de aplicaciones web han sido ampliamente reconocidas. Ofrece mejoras significativas en la calidad y escalabilidad del código, así como una mejor experiencia de desarrollo y rendimiento optimizado. Además, la compatibilidad con JavaScript permite una transición suave desde proyectos existentes hacia el uso de TypeScript.

Figura 3.1.: Logo de TypeScript

3.2. HTML

HTML (Lenguaje de Marcado de Hipertexto) es el lenguaje de marcado estándar utilizado para crear y estructurar el contenido de las páginas web y aplicaciones móvil. Fue desarrollado por el World Wide Web Consortium (W3C). HTML proporciona la estructura básica para presentar contenido en un navegador web o en una app móvil. Una característica fundamentales de HTML es su capacidad para utilizar etiquetas o elementos predefinidos que indican cómo se debe mostrar el contenido en un documento. Estas etiquetas, escritas en forma de marcas angulares <>, permiten definir la estructura lógica de un documento, como encabezados, párrafos, listas, enlaces, imágenes y mucho más.En la Figura 3.2 se muestra un ejemplo de un sencillo código HTML y la apariencia de la web generada con este.

HTML utiliza una estructura de árbol llamada DOM (Modelo de Objetos del Documento) para representar el contenido de una página web. El DOM organiza los elementos HTML en una jerarquía y proporciona una interfaz de programación que permite a los desarrolladores acceder, modificar y manipular los elementos y atributos HTML de manera dinámica. Por ejemplo, se puede utilizar JavaScript para seleccionar un elemento específico del DOM y cambiar su contenido, agregar nuevos elementos, aplicar estilos o agregar eventos de interacción. Esta capacidad de manipulación del DOM en tiempo real es lo que permite crear páginas web y aplicaciones interactivas y dinámicas. [11] En nuestro caso se hará uso de TypeScript, que como se ha explicado en la Sección 3.1 es equivalente a usar JavaScript.

HTML es un lenguaje sencillo , y su sintaxis se basa en el uso de etiquetas de apertura y cierre. Además, HTML se puede combinar con otros lenguajes y tecnologías web, en el caso de este proyecto con SCSS (Sassy Cascading Style Sheets) para el diseño

y la presentación visual, y Typescript para la interactividad y el comportamiento dinámico en la app móvil.

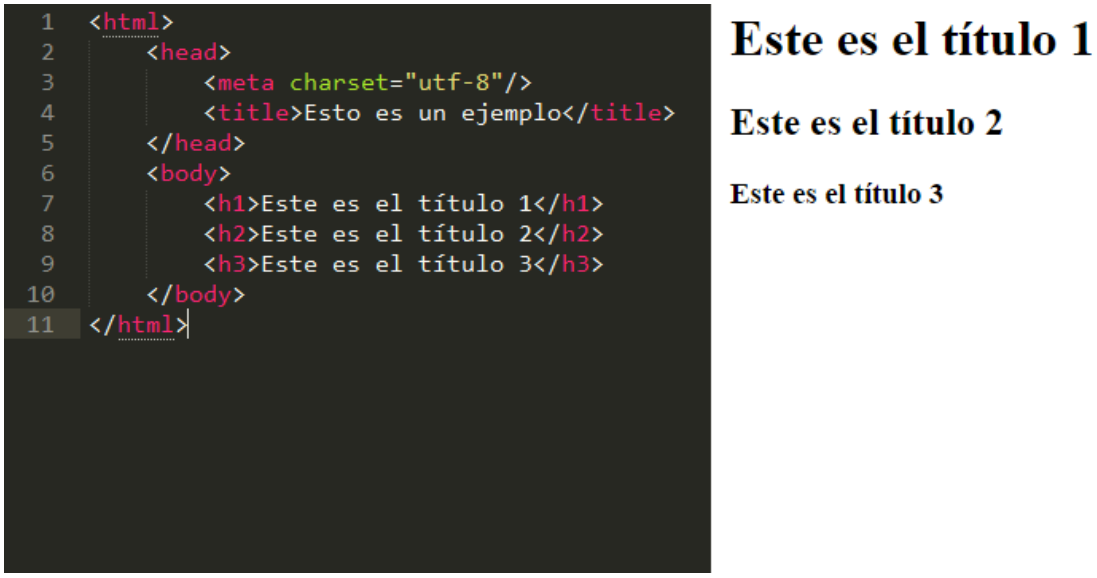


Figura 3.2.: Ejemplo de código HTML

En el desarrollo del proyecto Food&Move, HTML se ha utilizado para crear la estructura y el contenido de las diferentes vistas y páginas de la aplicación móvil. Mediante el uso de etiquetas HTML, se han definido los encabezados, los botones y todos los elementos necesarios para proporcionar una experiencia de usuario interactiva.

Figura 3.3.: Logo de HTML

3.3. SCSS

SCSS (Sassy Cascading Style Sheets) es una extensión de CSS que agrega características y funcionalidades avanzadas al lenguaje de hojas de estilo. SCSS es un preprocesador de CSS, lo que significa que permite utilizar una sintaxis más poderosa y expresiva que facilita la escritura y el mantenimiento del código CSS. Como preprocesador, SCSS toma el código escrito en su sintaxis y lo compila en código CSS válido que se puede aplicar a la página web o aplicación móvil. A través de la utilización de variables, mixins, anidamiento de selectores y otros mecanismos, SCSS permite una mayor modularidad y reutilización de estilos, lo que simplifica el proceso de desarrollo y mantenimiento de la apariencia visual de una página web o aplicación móvil.

Una de las características destacadas de SCSS es su capacidad de utilizar variables, lo que facilita la definición y gestión de valores reutilizables en los estilos. Estas variables se pueden declarar y asignar valores, y luego ser utilizadas en diferentes partes del código SCSS. Esto permite mantener un estilo coherente y consistente en toda la aplicación, ya que los cambios en los valores de las variables se reflejarán automáticamente en todas las instancias en las que se utilicen. SCSS también admite la creación de mixins, que son bloques de código reutilizables que pueden contener estilos CSS. Estos mixins se definen una vez y se pueden invocar en múltiples lugares, lo que permite aplicar estilos complejos y repetitivos de manera sencilla y eficiente. Además, los mixins pueden aceptar argumentos, lo que los hace aún más flexibles y adaptables. [14]

Otra característica importante de SCSS es el anidamiento de selectores. Esto permite anidar selectores CSS dentro de otros selectores, siguiendo la estructura del HTML o de los componentes de una aplicación. Un selector en CSS es una expresión que selecciona elementos específicos en un documento HTML para aplicarles estilos. Al anidar selectores en SCSS, los estilos se aplican a elementos específicos dentro de otros elementos, lo que facilita la comprensión y organización del código, ya que los estilos relacionados entre sí se agrupan de manera intuitiva. En cuanto a la sintaxis, SCSS se asemeja mucho a CSS, lo que facilita la transición y el aprendizaje para aquellos familiarizados con este último. Sin embargo, ofrece una serie de mejoras y atajos que agilizan la escritura del código y mejoran la legibilidad. Esto permite escribir estilos de forma más eficiente y concisa, ahorrando tiempo y esfuerzo en el desarrollo de la apariencia visual de una página web o aplicación móvil. [24]

En la Figura 3.4 se muestra un ejemplo de código SCSS, donde se utilizan variables, anidamiento de selectores y mixins. Este código SCSS se compilaría en CSS válido que se aplicará en la página web o en la aplicación móvil.

```scss
$primary: #007bff;
$font-size: 24px;

.container {
  width: 100%;
  padding: 20px;

  .title {
    font-size: $font-size;
    color: $primary;
  }

  .button {
    background-color: $primary;
    color: white;
    padding: 10px;
    border-radius: 4px;

    &:hover {
      background-color: darken($primary, 10%);
    }
  }
}

.card {
  background-color: #f0f0f0;
  padding: 10px;

  @mixin border($width, $style, $color) {
    border: $width $style $color;
  }

  @include border(1px, solid, #ccc);
}
```

Figura 3.4.: Ejemplo de código SCSS

SCSS es ampliamente utilizado en el desarrollo web y se integra fácilmente con otros lenguajes y tecnologías, como HTML y JavaScript. En el caso del proyecto Food&Move, SCSS se utiliza para definir los estilos y la presentación visual de la aplicación móvil.

Figura 3.5.: Logo de CSS

CAPÍTULO 4

FASES DEL PROYECTO

"No hay viento favorable
para el que no sabe
a que puerto se dirige."
— Sécenca, *s.I dC*

En este capítulo se abordan las diversas fases implicadas en el desarrollo de la aplicación móvil Food&Move. El proceso de creación de una aplicación exitosa va más allá de la implementación de funcionalidades, requiriendo un enfoque metódico en el diseño para asegurar una experiencia de usuario fluida y atractiva.

En la sección 4.1 , titulada "Fase inicial", se exploran los primeros pasos del proyecto, donde se establecen los objetivos en base a los requisitos y necesidades descritas por las partes interesadas (centro de salud y pacientes), y se definen las metas a alcanzar. En esta etapa, se identifican las necesidades de los usuarios y se recopila información relevante para guiar el proceso de diseño. La sección 4.2, "Fase de análisis", se enfoca en el estudio detallado de los requisitos y funcionalidades necesarias para la aplicación. Se obtiene una comprensión profunda de las expectativas de los usuarios y se establecen las bases para el diseño.

La sección 4.3 se crea la estructura y la apariencia visual de la aplicación. Se exploran aspectos como la arquitectura de información, la creación de bocetos, prototipos y wireframes, así como la definición de la identidad visual y los principios de diseño que guiarán el proceso. En la sección 4.4, "Fase de desarrollo", se aborda la implementación práctica del diseño visual en un producto funcional. Se trata de traducir el diseño en código.

La sección 4.5 "Fase de pruebas"se dedica a la evaluación exhaustiva de la aplicación, donde se identifican posibles errores y se comprueba la usabilidad . Esta retroalimentación es esencial para realizar ajustes y mejoras antes del lanzamiento final. Por último, la sección 4.6 "Fase final"se centra en los últimos pasos del proyecto, incluyendo el refinamiento del diseño, la preparación para el lanzamiento y la documentación necesaria para el mantenimiento futuro de la aplicación.

4.1. Fase inicial

En la fase inicial del proyecto Food&Move, se lleva a cabo el planteamiento y la contextualización del proyecto. Esta etapa es crucial para establecer los fundamentos y objetivos del proyecto, así como para comprender la motivación y la necesidad de implementar hábitos saludables en el tratamiento de pacientes con enfermedades mentales.

Mi inclusión en el proyecto Food&Move surge a raíz de una reunión con el tutor en octubre del año pasado, donde se me presentó un centro especializado en Zamora que trata a paciente con problemas de salud mental. En este centro, como parte del tratamiento, incluyen la importancia de promover hábitos saludables, como una alimentación equilibrada y la práctica regular de ejercicio físico. Dado que la imple-

mentación de estos hábitos puede resultar desafiante para los pacientes, especialmente si no cuentan con recursos adecuados o no se encuentran lo suficientemente motivados. Se planteo la necesidad de explorar cómo la tecnología puede facilitar y mejorar la implementación de estos hábitos saludables de manera más sencilla y efectiva. El proyecto Food&Move se presenta como una solución tecnológica innovadora que busca abordar esta necesidad. Combina una plataforma web y una aplicación móvil diseñadas específicamente para ayudar a los pacientes a adoptar y mantener hábitos saludables en su vida diaria. La plataforma web permitirá a los enfermeros encargados del tratamiento hacer un seguimiento más preciso y efectivo de la evolución de los pacientes, así como establecer planes de dieta y ejercicio personalizados según las necesidades de cada individuo. Por su parte, la aplicación móvil permitirá a los pacientes acceder de manera fácil y accesible a sus planes personalizados, así como a recursos multimedia que los apoyen en la adopción de una alimentación equilibrada y la práctica regular de ejercicio.

En la reunión, se detalló que dentro del proyecto Food&Move, mi contribución se centrará en el desarrollo de la aplicación móvil destinada a los pacientes, así como en la colaboración con la implementación de una base de datos conjunta que será compartida tanto por la plataforma web utilizada por los enfermeros encargados del tratamiento como por la aplicación móvil utilizada por los pacientes. Mi objetivo principal será diseñar y programar una interfaz intuitiva y amigable para los pacientes, que les permita acceder a sus planes personalizados de dieta y ejercicio, así como interactuar con los recursos multimedia disponibles. Además, trabajaré en la integración de la aplicación móvil con la base de datos conjunta, asegurando la correcta sincronización de la información entre la plataforma web y la aplicación móvil.

La fase inicial del proyecto Food&Move sienta las bases y establece los objetivos generales del mismo. A partir de aquí, se avanza hacia la fase de análisis, donde se profundiza en los requisitos y especificaciones del proyecto, sentando las bases para el diseño y desarrollo de la aplicación móvil Food&Move. Además, se planteó como fecha de finalización del proyecto Agosto de 2023.

4.2. Fase de análisis

La fase de análisis del proyecto Food&Move es fundamental para comprender en detalle los requisitos y objetivos de la aplicación móvil que se va a desarrollar, para así

establecer bases sólidas necesarias para el diseño y desarrollo de la misma. En esta etapa, se llevará a cabo una serie de actividades clave que nos permitirán obtener una visión clara y completa del proyecto. A través de la identificación de los requisitos, el análisis de los usuarios, la definición de los casos de uso y la evaluación de tecnologías, se sentarán las bases para el desarrollo de una aplicación móvil eficiente y efectiva. A continuación, se detallarán cada una de las subsecciones de esta fase:

4.2.1. Identificación de los requisitos

La identificación de los requisitos es una etapa crítica en el proceso de desarrollo de la aplicación móvil Food&Move. Los requisitos definen las funcionalidades y características que deben ser implementadas en la aplicación para satisfacer las necesidades de los usuarios y lograr los objetivos del proyecto. Este apartado del documento especifica cuáles serán los requisitos que debe satisfacer la herramienta tecnológica a desarrollar. Estos aspectos puede ser aspectos que afecten a las funcionalidades de la herramienta (requisitos funcionales) o aspectos que afecten a otras cuestiones como la seguridad, fiabilidad, experiencia de usuario o rendimiento (requisitos no funcionales).

Se ha decidido dividir los requisitos en dos secciones correspondientes a los dos principales pilares del proyecto. Los requisitos de este proyecto están divididos en aquellos correspondientes a la aplicación móvil y el servidor backend de la aplicación móvil. Es importante destacar el proceso llevado a cabo para la elicitación de los requisitos del proyecto. Los requisitos se han definido mediante la realización de una serie de reuniones entre una enfermera del centro interesado en la aplicación, el tutor del proyecto y el alumno.

Requisitos funcionales de la aplicación

- **Inicio de sesión:** El sistema de inicio de sesión en la aplicación móvil se realizará mediante el número de teléfono del paciente y una contraseña generada aleatoriamente que será visible únicamente para el enfermero en la plataforma web. Esta metodología se implementa con el objetivo de aliviar cualquier preocupación o dificultad que pueda experimentar el paciente al realizar el proceso de inicio de sesión. Para mayor comodidad, la sesión del usuario se mantendrá abierta, evitando así la necesidad de iniciar sesión repetidamente.

Sin embargo, se incluirá la opción de cerrar sesión por si se producen errores o se requiere realizar algún cambio en la cuenta del paciente. El enfermero tendrá acceso a la contraseña y el número del paciente en la plataforma web, lo que permitirá la recuperación de la cuenta en caso de ser necesario.

El objetivo principal del inicio de sesión es obtener el identificador único del usuario, que permitirá acceder a sus dietas, ejercicios y otros datos relevantes almacenados en la base de datos. Esta funcionalidad garantiza la personalización y adaptación de la aplicación a cada paciente, proporcionando un entorno de uso individualizado y seguro.

- **Visualización de dieta:** La aplicación móvil permitirá al usuario visualizar los platos asignados para el desayuno, comida, merienda y cena del día actual. Esta funcionalidad proporcionará una visión clara y organizada de los alimentos recomendados para cada una de las principales comidas diarias.La interfaz de visualización de la dieta estará diseñada de forma clara y concisa, presentando los platos de manera legible.

  Además, se incluirá la capacidad de navegar hacia los días posteriores y anteriores, lo que permitirá al usuario explorar su dieta en diferentes momentos. Esta funcionalidad resultará útil para planificar las comidas con antelación y tener una visión general de la dieta a lo largo del tiempo. La visualización de la dieta en la aplicación móvil garantiza que el usuario tenga acceso rápido y sencillo a la información necesaria para seguir su plan de alimentación de manera efectiva. Esto fomenta una mayor organización y facilita el cumplimiento de los objetivos establecidos en el programa de tratamiento.

- **Visualización de ejercicio pautado:** La aplicación móvil permitirá al usuario visualizar las actividades físicas asignadas al día actual. Esta funcionalidad proporcionará una visión clara y organizada de los ejercicios recomendados para cada día. La interfaz de visualización del plan de entrenamiento estará diseñada de forma clara y concisa, presentando los ejercicios de manera legible.

  Además, se incluirá la capacidad de navegar hacia los días posteriores y anteriores, lo que permitirá al usuario explorar su plan de entrenamiento en diferentes momentos. Esta funcionalidad resultará útil para planificarlos con antelación y tener una visión general del plan a lo largo del tiempo. La visualización del plan de actividad física en la aplicación móvil garantiza que el usuario tenga acceso rápido y sencillo a la información necesaria para seguir su plan de manera

efectiva. Esto fomenta una mayor organización y facilita el cumplimiento de los objetivos establecidos en el programa de tratamiento.

- **Visualización de contenido multimedia sobre la dieta:** La aplicación móvil permitirá al usuario acceder a un contenido multimedia completo y detallado relacionado con la dieta. Esta funcionalidad proporcionará información adicional sobre cada comida y plato recomendado, enriqueciendo la experiencia del usuario y brindándole recursos prácticos y útiles para seguir su plan de alimentación.

  El contenido multimedia incluirá los siguientes elementos:

  1. Título de la comida: Se mostrará el nombre descriptivo de la comida en cuestión, permitiendo al usuario identificar rápidamente el tipo de comida que se está presentando.
  2. Comida a la que pertenece: Se indicará a qué comida específica (desayuno, comida, merienda o cena) corresponde el plato en cuestión.
  3. Plato al que pertenece: Se especificará si el plato es el primero, segundo, postre u otra categoría dentro de la comida.
  4. Descripción: Se proporcionará una descripción detallada del plato, destacando sus características principales.
  5. Comentarios: Se permitirá al enfermero realizar comentarios y añadir indicaciones relacionadas con el plato en cuestión, fomentando la interacción y el intercambio de información entre los enfermeros y el paciente a través de la aplicación.
  6. Listado de ingredientes: Se presentará un listado completo de los ingredientes necesarios para preparar el plato.
  7. Enlaces a recetas: Se incluirán enlaces a recetas detalladas que proporcionen instrucciones paso a paso sobre cómo preparar el plato recomendado.
  8. Fecha de consumo: Se mostrará la fecha específica en la que se debe consumir el plato, ayudando al usuario a organizar su dieta diaria de manera efectiva.
  9. Videos: La aplicación permitirá la reproducción de videos relacionados con la preparación y presentación de los platos recomendados. Algunos de estos videos serán creados por el centro y ofrecerán una guía visual adicional para los usuarios.

10. Documentos PDF con las recetas: La aplicación mostrará documentos PDF que contengan las recetas completas de los platos recomendados. Estos documentos serán creados por el centro y se mostrarán directamente en la aplicación para facilitar el acceso y la consulta.

Cabe destacar que no todos estos campos serán obligatorios en cada comida y debe tenerse en cuenta que no siempre van a mostrase ya que hay casos en los que no existirán.

La visualización de contenido multimedia sobre la dieta en la aplicación móvil brindará a los usuarios una experiencia enriquecedora y práctica, al proporcionarles información detallada, recursos visuales y documentos relacionados con los platos recomendados. Esto les permitirá seguir su plan de alimentación de manera efectiva y obtener todo el apoyo necesario para llevar una dieta saludable y equilibrada.

- **Visualización de contenido multimedia sobre el ejercicio pautado:** La aplicación móvil permitirá al usuario acceder a un contenido multimedia completo y detallado relacionado con la actividad física pautada. Esta funcionalidad proporcionará información adicional sobre cada ejercicio recomendado, enriqueciendo la experiencia del usuario y brindándole recursos prácticos y útiles para seguir su plan de ejercicio físico.

  El contenido multimedia incluirá los siguientes elementos:

  1. Título del ejercicio: Se mostrará el nombre descriptivo del ejercicio en cuestión.
  2. Descripción: Se proporcionará una descripción detallada del ejercicio, destacando sus aspectos relevantes.
  3. Comentarios: Se permitirá al enfermero realizar comentarios y añadir indicaciones relacionadas con el ejercicio en cuestión, fomentando la interacción y el intercambio de información entre los enfermeros y el paciente a través de la aplicación.
  4. Enlaces a páginas web: Se incluirán enlaces a páginas web que proporcionen instrucciones paso a paso sobre cómo realizar el ejercicio recomendado.
  5. Fecha de realización: Se mostrará la fecha específica en la que se debe realizar el ejercicio.

6. Videos: La aplicación permitirá la reproducción de videos relacionados con el ejercicio. Algunos de estos videos serán creados por el centro y ofrecerán una guía visual adicional para los usuarios.

7. Documentos PDF con los ejercicios: La aplicación mostrará documentos PDF que contengan las ejercicios recomendados con indicaciones detalladas. Estos documentos serán creados por el centro y se mostrarán directamente en la aplicación para facilitar el acceso y la consulta.

Cabe destacar que no todos estos campos serán obligatorios en cada ejercicio y debe tenerse en cuenta que no siempre van a mostrase ya que hay casos en los que no existirán.

La visualización de contenido multimedia sobre el plan de entrenamiento en la aplicación móvil brindará a los usuarios una experiencia enriquecedora y práctica, al proporcionarles información detallada, recursos visuales y documentos relacionados con los ejercicios recomendados. Esto les permitirá seguir su plan de manera efectiva y obtener todo el apoyo necesario para llevar un nivel de actividad física adecuado.

- **Visualización de la lista de la compra:** La visualización de la lista de la compra es un componente clave de la aplicación móvil "Food&Move". Esta funcionalidad tiene como objetivo ayudar a los pacientes a mantenerse organizados y motivados al seguir su plan de dieta personalizado. Al proporcionar una lista clara y estructurada de los ingredientes necesarios, los pacientes podrán realizar compras de manera eficiente y asegurarse de tener los elementos necesarios para seguir su plan de dieta correctamente.

  La visualización de la lista de la compra incluirá los siguientes elementos:

  1. Nombre del ingrediente: Cada elemento de la lista mostrará el nombre del ingrediente requerido para la dieta del paciente. Esta información será clara y legible para facilitar la identificación de los productos en el momento de realizar la compra.

  2. Cantidad necesaria: Junto al nombre de cada ingrediente, se indicará la cantidad necesaria para la dieta del paciente. Esto permitirá a los usuarios tener una idea clara de la cantidad de cada producto que deben adquirir.

3. Día de uso: Además de la cantidad necesaria, se mostrará el día específico en el que se utilizará cada ingrediente. Esto ayudará a los pacientes a planificar su dieta y asegurarse de que tienen los ingredientes necesarios disponibles para cada día.

4. Checkbox de disponibilidad: Para facilitar el seguimiento de los ingredientes adquiridos, se incluirá un checkbox junto a cada elemento de la lista. Los pacientes podrán marcar el checkbox para indicar que ya han adquirido o disponen del ingrediente correspondiente. Esta funcionalidad proporcionará una forma visualmente clara de realizar un seguimiento de los ingredientes que se han obtenido o están disponibles.

- **Preguntas diaria:** El sistema debe proporcionar la funcionalidad de mostrar cuatro preguntas diarias al usuario. Estas preguntas se dividirán en dos categorías: dos preguntas relacionadas con la cumplimentación de la comida y el ejercicio pautado, y otras dos pregunta adicional para valorar dichos aspectos. El contenido y el formato de las preguntas serán definidos por el personal del centro sanitario, asegurándose de redactarlas de forma adecuada para suponer un refuerzo positivo al tratamiento del paciente.

  El propósito de estas preguntas es permitir al enfermero hacer un seguimiento de la adherencia del paciente al plan establecido, así como detectar posibles inconformidades del paciente con el plan. Las respuestas a estas preguntas proporcionarán información valiosa para evaluar el progreso del paciente y ajustar el tratamiento de manera apropiada. La implementación de esta funcionalidad garantizará una comunicación efectiva entre el paciente y el equipo de atención médica, facilitando la detección temprana de problemas y la toma de decisiones informadas para mejorar la experiencia del paciente y optimizar los resultados del tratamiento.

- **Notificación preguntas diarias:** El sistema debe tener la capacidad de enviar notificaciones al paciente para recordarle que responda a sus preguntas diarias. Las notificaciones deben ser claras, concisas y amigables, brindando al paciente un recordatorio amable pero efectivo para completar sus respuestas diarias. El objetivo de estas notificaciones es fomentar la participación activa del paciente en su plan de tratamiento y garantizar la recopilación regular de información para el seguimiento adecuado por parte del equipo de atención médica. Al recibir las notificaciones, el paciente debe comprender fácilmente la importancia y el

propósito de responder a las preguntas diarias, lo que contribuirá a una mejor adherencia al plan y una mayor efectividad en el seguimiento de su progreso. La implementación de esta funcionalidad de notificación promoverá la interacción continua entre el paciente y el sistema, mejorando la comunicación y la colaboración para lograr resultados óptimos en el tratamiento.

Requisitos no funcionales de la aplicación

- **User Friendly:** La aplicación debe ser diseñada y desarrollada teniendo en cuenta las diversas competencias tecnológicas de los usuarios. Se buscará garantizar que la aplicación sea fácil de usar, intuitiva y genere una experiencia positiva que brinde apoyo sin generar estrés adicional al paciente.

  Para lograrlo, se deben considerar los siguientes aspectos como un diseño intuitivo y comprensible con una interfaz de usuario clara y organizada, utilizando un lenguaje sencillo y evitando tecnicismos. Los elementos visuales deben ser reconocibles y fáciles de entender para todos los usuarios. La aplicación debe proporcionar instrucciones claras en cada paso del proceso, evitando confusiones.La aplicación debe proporcionar retroalimentación inmediata y comprensible en respuesta a las acciones realizadas por el usuario. Esto puede incluir mensajes informativos o confirmaciones visuales y, que permitan al usuario entender el estado de sus acciones.

  El objetivo principal de este requisitos es asegurar que la aplicación sea una herramienta útil y amigable para todos los usuarios, independientemente de sus capacidades tecnológicas y su patología. Se busca proporcionar una experiencia positiva, minimizando el estrés y permitiendo que la aplicación cumpla su propósito de ser una ayuda efectiva en el seguimiento del tratamiento y la adherencia del paciente.

- **Multiplataforma:** La aplicación debe ser compatible y funcionar de manera óptima en diferentes plataformas, incluyendo dispositivos móviles iOS y Android. Se requiere que la aplicación sea accesible y utilizable de manera consistente en estas plataformas, proporcionando una experiencia fluida y coherente para todos los usuarios.

- **Portable:** La aplicación está diseñada para ser usada en todo tipo de dispositivos móviles. Debe estar adaptada a diferentes tamaños y configuraciones de pantalla.

Asimismo, debe ser responsive con el fin de que los menús e interfaces se adapten a diferentes tamaños y resoluciones de pantalla.

- **Robustez:** La aplicación debe contener un mínimo número de fallos y debe ser segura. Para ello debe contar con diferentes características de seguridad como el cifrado de contraseñas entre otras.
- **Tiempos de respuesta:** La aplicación debe tener un tiempo de respuesta rápido y eficiente en todas las interacciones y operaciones realizadas por el usuario. Debe ofrecer un buen tiempo de carga inicial, ser ágil en la respuesta a acciones del usuario y brindar un buen tiempo de procesamiento de solicitudes cuando aplicación realiza solicitudes a servidores externos o realiza operaciones que requieren un procesamiento adicional. Esto incluye, por ejemplo, enviar respuestas a preguntas diarias, sincronizar datos o cargar información desde fuentes externas.
- **Seguridad:** La aplicación debe contar con medidas robustas de seguridad para proteger la integridad, confidencialidad y disponibilidad de los datos del paciente. Se deben implementar controles de acceso adecuados para garantizar que solo los usuarios autorizados puedan acceder a la información sensible. Además, se deben aplicar técnicas de encriptación para proteger la transmisión y el almacenamiento de los datos. En general, se deben respetar las mejores prácticas de seguridad de la industria para garantizar la protección adecuada de los datos del paciente en todo momento.
- **Privacidad:** La aplicación debe garantizar la privacidad y confidencialidad de los datos del paciente y su tratamiento. Esto implica asegurar que los datos personales y de salud del paciente se mantengan de forma segura y accesible solo para los profesionales de la salud autorizados.
- **Recuperación ante fallos:** Dado que la aplicación puede verse en situaciones en las que se producen fallos externos a la misma, debe ser capaz de recuperarse y mantener la persistencia de los datos sin que un fallo temporal pueda acarrear fallos permanentes.
- **Mantenibilidad:** El mantenimiento de la plataforma no debe suponer una gran complejidad para los administradores del sistema.

A continuación, nos centraremos únicamente en los requisitos del servidor backend que están directamente relacionados con la aplicación móvil. Si bien el servidor bac-

kend también puede tener requisitos relacionados con la plataforma web, en este caso, nos enfocaremos específicamente en aquellos que son necesarios para el correcto funcionamiento y la integración con la aplicación móvil.

Requisitos funcionales del servidor backend

- **Inicio de sesión:** El servidor backend debe contar con un sistema de inicio de sesión que permita a los usuarios autenticarse mediante el número de teléfono y la contraseña . Además, el servidor backend debe ofrecer la funcionalidad de mantener la sesión abierta para evitar que los usuarios tengan que iniciar sesión repetidamente en la aplicación móvil. No obstante, se debe incluir la opción de cerrar sesión en caso de errores o cambios en la cuenta del paciente. El sistema de inicio de sesión en el servidor backend tiene como finalidad principal obtener el identificador único del usuario de la base de datos y brindarselo a la app, esto permitirá acceder a sus datos personales, dietas, ejercicios y otro contenido relevante almacenado en la base de datos.
- **Obtención de dieta:** El servidor backend debe proporcionar la funcionalidad de obtener todos los datos disponibles en la base de datos relacionados con la dieta del paciente, utilizando el identificador único de cada usuario obtenido durante el inicio de sesión. Esta información incluirá todo el contenido, incluyendo contenido multimedia, que se pueda mostrar al paciente en la app y se encuentre disponible en la base de datos.

  El objetivo es garantizar que la aplicación móvil tenga acceso a todos los datos necesarios para mostrar de manera completa y precisa la dieta asignada al paciente. El servidor backend deberá realizar las consultas correspondientes a la base de datos y proporcionar los datos de la dieta al dispositivo móvil de manera eficiente y segura. Además, se debe tener en cuenta que los datos de la dieta pueden estar sujetos a modificaciones por parte del personal médico, por lo que el servidor backend debe actualizar la información de manera oportuna para reflejar cualquier cambio realizado en la dieta del paciente.
- **Obtención de ejercicio pautado:** El servidor backend debe proporcionar la funcionalidad de obtener todos los datos disponibles en la base de datos relacionados con el ejercicio pautado al paciente, utilizando el identificador único de cada usuario obtenido durante el inicio de sesión. Esta información incluirá todo el

contenido, incluyendo contenido multimedia, que se pueda mostrar al paciente en la app y se encuentre disponible en la base de datos.

El objetivo es garantizar que la aplicación móvil tenga acceso a todos los datos necesarios para mostrar de manera completa y precisa el plan de actividad física asignado al paciente. El servidor backend deberá realizar las consultas correspondientes a la base de datos y proporcionar los datos de los ejercicios al dispositivo móvil de manera eficiente y segura. Además, se debe tener en cuenta que los datos pueden estar sujetos a modificaciones por parte del personal médico, por lo que el servidor backend debe actualizar la información de manera oportuna para reflejar cualquier cambio realizado.

- **Obtención de ingredientes:** El servidor backend debe proporcionar la funcionalidad para obtener un listado de ingredientes desde la base de datos, de manera similar a la obtención de dietas y ejercicios. Este listado debe incluir la información útil para la lista de la compra detallada que se muestra al paciente en la aplicación móvil. Esta funcionalidad permitirá al paciente acceder de forma conveniente a la lista de ingredientes necesarios para su plan de dieta, facilitando así la organización y preparación de los alimentos requeridos.

- **Marcado de ingredientes:** El servidor backend debe contar con la funcionalidad de marcar en la base de datos los ingredientes que el paciente haya seleccionado o marcado como "adquiridos" en la aplicación móvil. Esta acción permitirá mantener un registro actualizado de los ingredientes y alimentos que el paciente ha obtenido. Cuando el usuario marque un elemento como adquirido en la aplicación, el servidor backend deberá recibir esta información y actualizar el estado correspondiente en la base de datos. De esta manera, se garantiza la sincronización entre la aplicación móvil y la información almacenada, brindando al paciente una visión precisa y actualizada de los elementos que ha adquirido. Es importante que esta funcionalidad sea ágil y confiable, para que los cambios realizados por el paciente en la aplicación se reflejen de manera inmediata en la base de datos.

- **Publicación de valoración diaria:** El servidor backend debe tener la capacidad de recibir y almacenar las respuestas diarias del paciente en la base de datos, de modo que puedan ser visualizadas por el enfermero en su plataforma web. Para ello, se requerirá una interfaz que permita la comunicación entre la aplicación móvil y el servidor. Una vez que el paciente envíe sus respuestas diarias a través

de la aplicación móvil, el servidor backend deberá procesar y registrar dichas respuestas en la base de datos. Esta información debe estar asociada correctamente al paciente correspondiente, utilizando su identificador único.

Requisitos no funcionales del servidor backend

- **Tiempo de respuesta:** El servidor backend debe ser capaz de proporcionar respuestas rápidas a las solicitudes de la aplicación móvil. Asegurando así una experiencia fluida y sin demoras para los usuarios.
- **Aprovechamiento del espacio de almacenamiento:** El servidor backend debe utilizar eficientemente el espacio de almacenamiento disponible para garantizar un uso óptimo de los recursos. Los datos almacenados deben estar comprimidos y optimizados, y se deben implementar estrategias adecuadas de gestión y limpieza de datos para evitar la ocupación innecesaria de espacio.
- **Rendimiento:** El servidor backend debe ser capaz de manejar una carga de trabajo considerable y responder de manera eficiente incluso en momentos de alta demanda. Se deben realizar pruebas de rendimiento para garantizar que el sistema pueda manejar un número significativo de solicitudes simultáneas sin experimentar degradación en el rendimiento.
- **Disponibilidad:** El servidor backend debe estar disponible de manera continua y confiable para garantizar el funcionamiento ininterrumpido de la aplicación móvil. Se pueden implementar medidas de redundancia, tolerancia a fallos y copias de seguridad para mitigar posibles interrupciones del servicio y garantizar altos niveles de disponibilidad.
- **Mantenimiento:** El servidor backend debe ser fácil de mantener y actualizar sin afectar negativamente la disponibilidad y el rendimiento del sistema. Se debe garantizar que el personal técnico tenga acceso a herramientas y documentación adecuadas para llevar a cabo estas tareas de mantenimiento de manera efectiva.

4.2.2. Análisis de usuarios

En esta sección se ha tratado de llevar a cabo un estudio detallado sobre el perfil de los usuarios de la aplicación móvil Food&Move. Los usuarios de esta aplicación

son personas que reciben tratamiento en un centro especializado en salud mental. Es fundamental comprender las características y necesidades específicas de este grupo de usuarios para diseñar una aplicación que sea accesible, fácil de usar y que se ajuste a sus requerimientos particulares. Para cumplir este cometido, ha resultado de gran utilidad escuchar las sugerencias de los miembros del centro especializado en el tratamiento de personas con problemas de salud mental, ya que cuentan con la experiencia y formación necesarias para tratar a sus pacientes de forma adecuada.

Una de las características importantes a tener en cuenta es que los usuarios de la aplicación pueden experimentar diferentes niveles de dificultad para realizar tareas cotidianas debido a su salud mental. Algunos usuarios pueden enfrentar dificultades cognitivas, como problemas de concentración o memoria, mientras que otros pueden presentar síntomas de ansiedad o depresión que afecten su motivación y compromiso. Además, es esencial considerar la sensibilidad emocional de los usuarios. Dado que la aplicación está destinada a ayudarles en su tratamiento y promover hábitos saludables, es importante crear una experiencia de usuario que sea alentadora, motivadora y libre de desencadenantes emocionales negativos. La interfaz y los mensajes de la aplicación deben ser cuidadosamente diseñados para brindar apoyo y motivación, evitando cualquier lenguaje o contenido que pueda causar estrés o ansiedad adicional.

Otro aspecto relevante es la diversidad en las capacidades y habilidades digitales de los usuarios. Algunos usuarios pueden tener limitaciones que dificulten el uso de la aplicación. Por lo tanto, se debe buscar la mayor sencillez posible diseño de la interfaz, como fuentes legibles, tamaños de botones adecuados y opciones de navegación intuitivas.

4.2.3. Definición de casos de uso

En esta sección se identifican y describen los principales escenarios en los que los usuarios de la aplicación móvil Food&Move interactuarán con la misma. Los casos de uso son representaciones de las acciones que los usuarios pueden llevar a cabo y las respuestas que la aplicación proporcionará en cada situación. Estos casos de uso fueron definidos de forma orientativa para tener una primera aproximación sin profundizar en detalles como las relaciones de herencia, extensión u otros detalles como los casos de error o el flujo de datos. Ayudarán a comprender las funcionalidades clave que la aplicación debe ofrecer y a definir los requisitos necesarios para su implementación.

En la figura 4.1 se muestran los principales casos de uso detallados en esta fase para la aplicación: Inicio y cierre de sesión, consulta de dietas y ejercicios, visualización de contenido multimedia sobre estos, valoración de la dieta y ejercicio diarios, consulta de lista de la compra y marcado de productos como adquiridos. Los esquemas mostrados en esta sección se han creado con la herramienta Astah. [4]

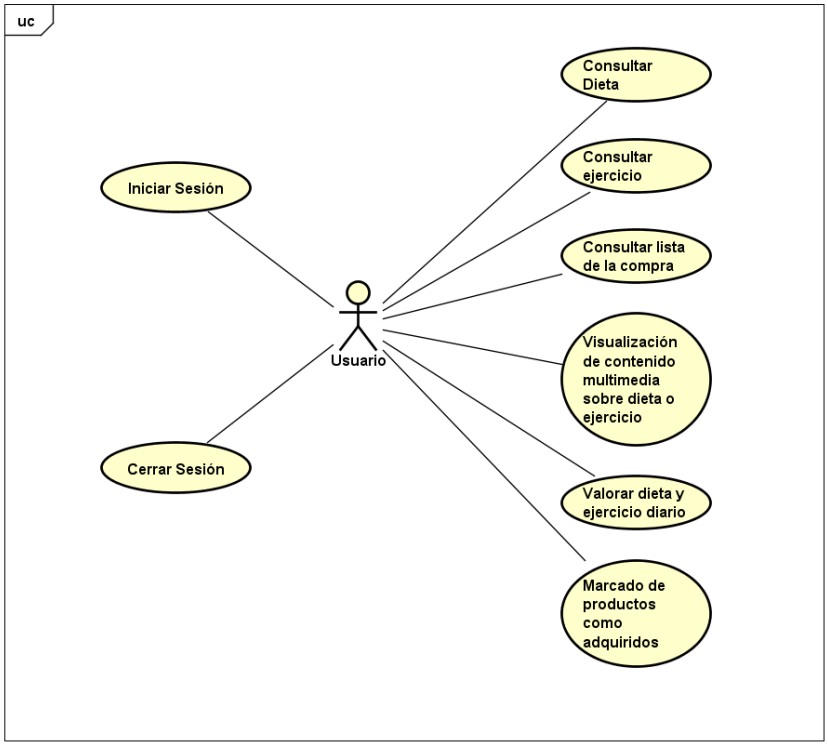


Figura 4.1.: Principales casos de uso

Los casos de uso de inicio y cierre de sesión son fundamentales para el funcionamiento de la aplicación móvil Food&Move. En el caso de uso de inicio de sesión, el usuario introduce sus credenciales para acceder a su cuenta personalizada. Una vez iniciada la sesión, el usuario tiene acceso a su perfil, dietas, ejercicios y otras funcionalidades. Por otro lado, el caso de uso de cierre de sesión permite al usuario finalizar su sesión actual. Ambos casos de uso proporcionan una experiencia fluida y segura al usuario, asegurando un acceso fácil y controlado a los recursos y funcionalidades de la aplicación. El esquema simplificado de estos dos casos de uso se muestra en las figuras 4.2 y 4.3.

Los casos de uso de consulta de la dieta y ejercicio diarios son de gran importancia en la aplicación móvil Food&Move. En el caso de uso de consulta de la dieta diaria, el usuario puede acceder a la información de su dieta planificada para la semana. Esto

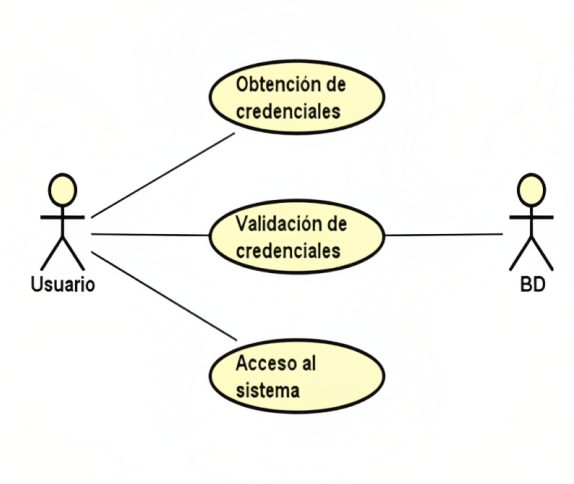


Figura 4.2.: Caso de uso: Login

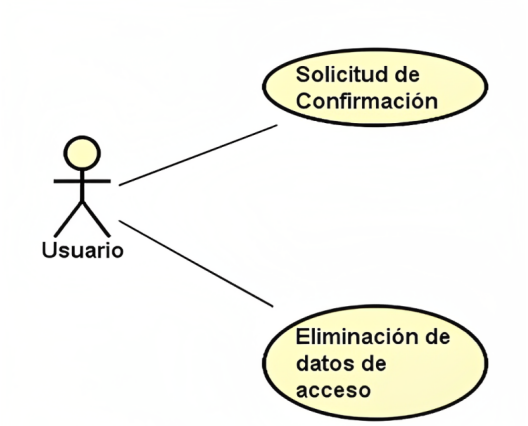


Figura 4.3.: Caso de uso: Logout

le permite tener un seguimiento preciso de su ingesta alimentaria y cumplir con su plan nutricional. Por otro lado, el caso de uso de consulta de ejercicio diario permite al usuario acceder a los detalles de los ejercicios programados para el día. Estos casos de uso brindan al usuario la información necesaria para llevar a cabo su dieta y ejercicio de manera adecuada, facilitando así el seguimiento de su plan de salud y bienestar. El esquema simplificado de estos dos casos de uso se muestra en las figuras 4.4 y 4.5.

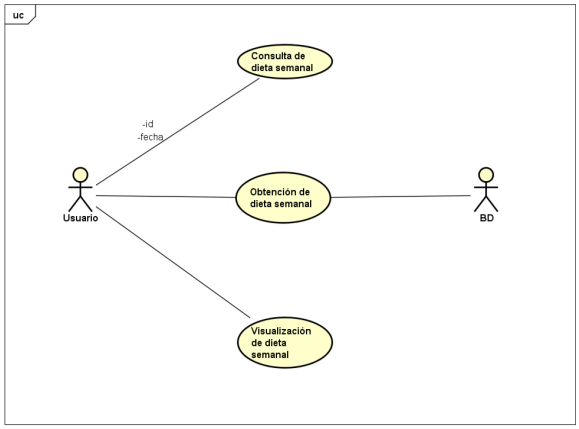


Figura 4.4.: Caso de uso: Consulta de dieta

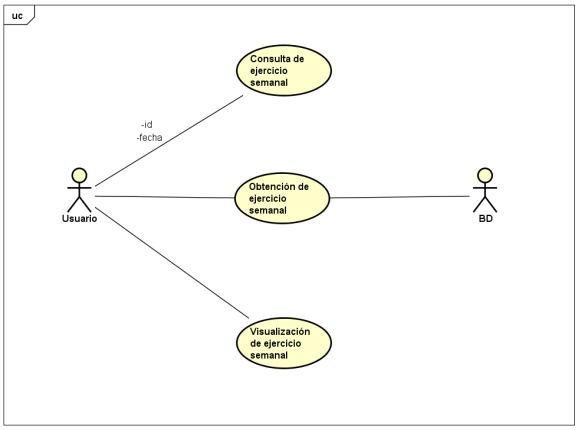


Figura 4.5.: Caso de uso: Consulta de ejercicio

El siguiente caso de uso es la visualización de todos los detalles de una comida o ejercicio concretos, incluyendo el contenido multimedia. Permite al usuario acceder de manera fácil y rápida a toda la información relevante de una comida o ejercicio específico, brindando una visión completa y detallada. A través de esta función, el usuario puede explorar los ingredientes, recetas, instrucciones y consejos asociados a una comida, así como los ejercicios recomendados, indicaciones de técnica y videos explicativos. La inclusión de contenido multimedia en forma de documentos PDF y videos enriquece la experiencia del usuario, proporcionando una comprensión más completa y facilitando la correcta realización de las actividades. Este caso de uso promueve la autonomía y el empoderamiento del usuario al brindarle todas las herramientas necesarias para llevar a cabo su plan de alimentación y ejercicio de manera efectiva. El esquema simplificado de este caso de uso se muestra en las figura 4.6.

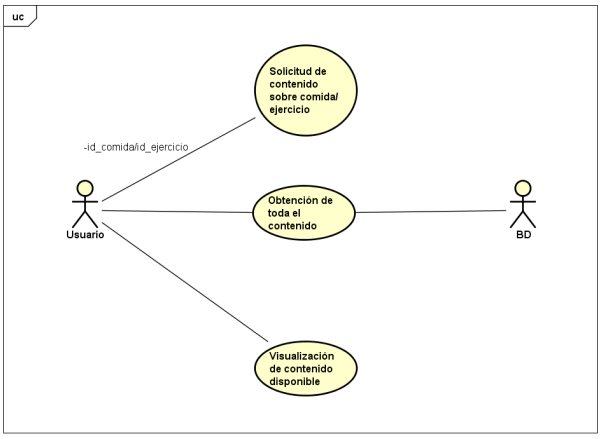


Figura 4.6.: Caso de uso: Consulta de contenido relacionado con comida/ejercio

Los casos de uso de consulta de la lista de la compra semanal y marcado de ingredientes como ya adquiridos son otras de las funcionalidades clave en la aplicación móvil Food&Move. En el caso de uso de consulta de la lista de la compra semanal, el usuario puede acceder a una lista detallada de los ingredientes necesarios para preparar sus comidas a lo largo de la semana. Esta lista le permite planificar y organizar sus compras de manera eficiente. Además, el caso de uso de marcado de ingredientes como ya adquiridos permite al usuario gestionar su lista de la compra, marcando aquellos ingredientes que ya ha comprado. Esto le ayuda a llevar un registro actualizado de los elementos que aún necesita adquirir y evita confusiones al realizar las compras. Estos casos de uso contribuyen a una experiencia de usuario más conveniente y facilitan la gestión de los ingredientes necesarios para seguir su plan de alimentación de manera exitosa.El esquema simplificado de estos dos casos de uso se muestra en las figuras 4.7 y 4.8.

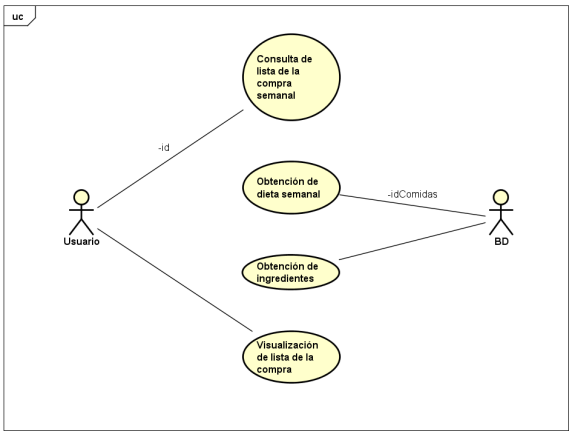


Figura 4.7.: Caso de uso: Consulta de lista de la compra

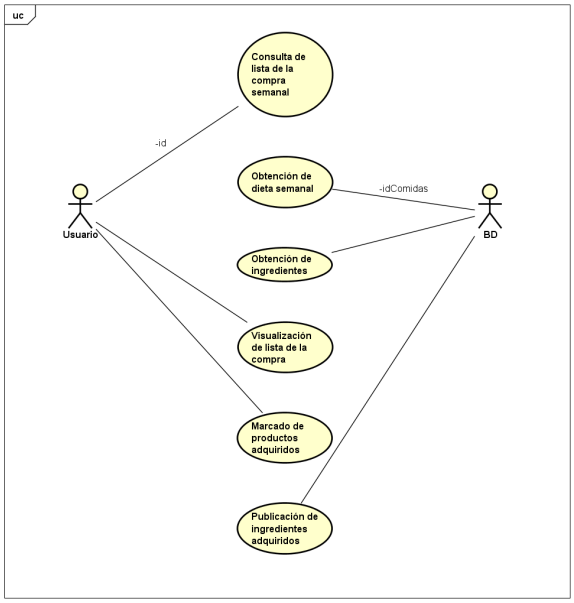


Figura 4.8.: Caso de uso: Marcado de ingredientes adquiridos

Por último el caso de uso de respuesta a las preguntas diarias permite al usuario proporcionar sus respuestas a las preguntas diarias relacionadas con la adherencia a su plan de alimentación y ejercicio. A través de esta función, el usuario brinda información importante al personal de enfermería para realizar un seguimiento adecuado de su evolución. Responder a estas preguntas facilita la comunicación y posibilita ajustes en el plan de tratamiento según sea necesario. Este caso de uso promueve la participación activa del usuario en su propio proceso de atención y contribuye a una mayor eficacia

en la prestación de cuidados personalizados. El esquema simplificado de este caso de uso se muestra en las figura 4.9.

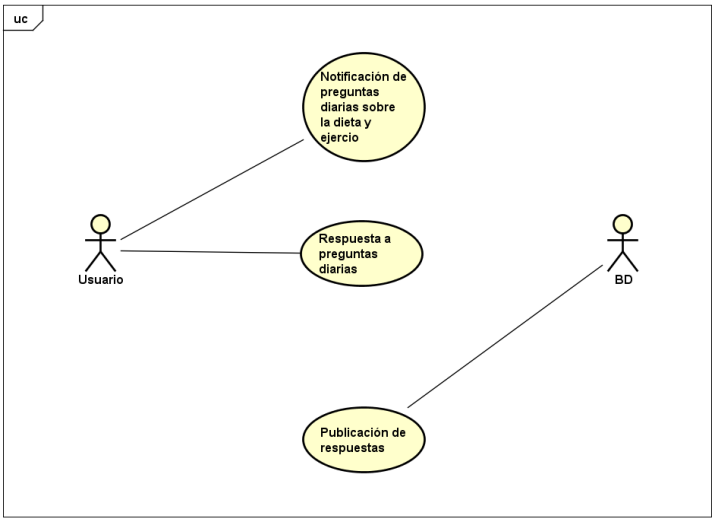


Figura 4.9.: Caso de uso: Pregunta diaria

4.2.4. Evaluación de tecnologías

La evaluación de tecnologías es una etapa importante en el proceso de desarrollo de la aplicación, en esta etapa se analizan y se seleccionan las herramientas y tecnologías adecuadas para implementar el proyecto. En el caso de este proyecto, por sugerencia del tutor, se ha decidido utilizar Ionic como el framework principal para el desarrollo de la aplicación móvil.

La elección de Ionic se basa en varias razones. La más destacada, que Ionic es un framework híbrido de desarrollo de aplicaciones móviles que permite crear aplicaciones tanto para iOS como para Android. Esto significa que se puede desarrollar una sola aplicación que funcione en múltiples plataformas, que es justo lo que se busca. Las características pricipales de Ionic están resumidas en el capítulo 2.

Por otra parte, la elección de MongoDB como base de datos ha sido sugerido por el compañero de proyecto encargado de desarrollar la plataforma web para los sanitarios. Esta elección, se debe a sus características, resumidas también en el capítulo 2 y a su capacidad para adaptarse a las necesidades del proyecto.

4.3. Fase de diseño

En la fase de diseño, se crea la estructura y la apariencia visual de la aplicación, teniendo en cuenta las necesidades y características del público objetivo. Para la aplicación Food&Move, se busca principalmente la sencillez, ofreciendo pocas alternativas y priorizando el refuerzo positivo para los usuarios. Estos objetivos se han seleccionado teniendo en cuenta las características de los usuarios descritas en la sección 4.2.2 Algunos elementos clave que se utilizarán para cumplir estos objetivos son:

- **Pantallas simples:** Se diseñarán pantallas con poca información y muy clara, evitando la sobrecarga de contenido. La simplicidad en la presentación permitirá que los usuarios puedan comprender fácilmente las opciones disponibles y tomar decisiones de manera intuitiva.
- **Elementos gráficos:** Se utilizarán iconos grandes e ilustrativos para guiar a los usuarios a través de la aplicación. Los iconos ayudarán a comunicar visualmente las funcionalidades y acciones disponibles, facilitando la comprensión y el uso de la aplicación.

- **Mensajes positivos:** Se incorporarán mensajes positivos en la interfaz de usuario para motivar y reforzar el progreso de los usuarios. Estos mensajes podrán incluir frases motivadoras, felicitaciones por el cumplimiento de metas o cualquier otro elemento que promueva una experiencia positiva y motivadora para los usuarios.
- **Colores cálidos:** Se utilizarán colores cálidos en la paleta de colores de la aplicación, y se hará especial énfasis en el color amarillo como el color principal de la aplicación móvil. Se ha elegido el amarillo porque se cree que tiene efectos positivos para las personas con problemas de salud mental. El uso de colores cálidos creará una atmósfera acogedora y agradable en la aplicación, contribuyendo a generar una experiencia positiva para los usuarios. [19]

El primer mockup de la aplicación representa una parte relevante de la fase diseño. Un mockup es una representación estática de una interfaz, que proporciona una vista previa de cómo se verá y se organizará la aplicación final. En esta etapa temprana del proceso de diseño, se creó un mockup para capturar la esencia y la disposición general de la aplicación, permitiendo evaluar su usabilidad. A continuación, se van a mostrar y explicar estas capturas del primer mockup que ofrecen una visión temprana de cómo se planteo la experiencia de usuario y sirvieron como punto de partida para iteraciones y mejoras posteriores en el diseño. Es importante aclarar que forman parte de la fase de diseño y, por lo tanto, tienen varias diferencias con el resultado final y no representan la totalidad de la aplicación.

La primera captura del mockup inicial se muestra en la Figura 4.10 y corresponde a la pantalla del login. Para este primer mockup, se diseñó una pantalla clásica de log in para que el usuario acceda a su cuenta y pueda ver sus datos con la opción de que se recuerde sus credenciales para no tener que introducirlas cada vez que se use la app. Esta vista sufrió varias modificaciones para adaptarse a la forma de autenticación que se detalla en el capítulo 5 y otras como la incorporación del logo de la app en el lugar del icono central. Las demás vistas que se presentarán a continuación, también sufrieron modificaciones, pero no se detallarán puesto que ya se ha aclarado que esto forma parte de la fase de diseño y no del resultado final.

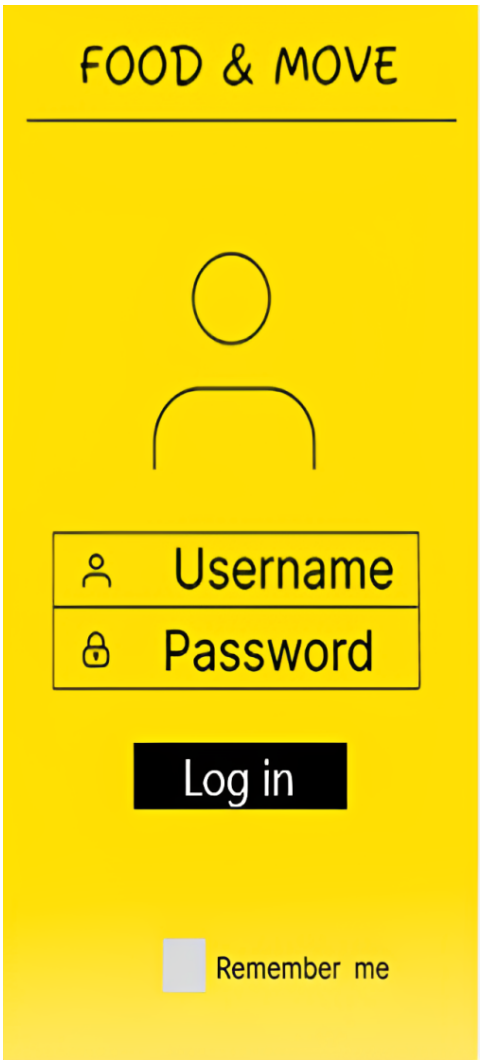


Figura 4.10.: Mockup: Log in

La Figura 4.11 representa la pantalla inicial que se muestra al usuario después de iniciar sesión en la aplicación. En ella, el usuario es recibido por una frase motivadora seleccionada para generar un impacto positivo en su estado de ánimo y proporcionar un estímulo adicional para participar en las actividades saludables propuestas. Además, se presentan dos iconos grandes y visualmente llamativos que representan las dos opciones principales disponibles en la aplicación: el apartado de comida y el apartado de ejercicio. Estos iconos facilitan la navegación y permiten al usuario elegir rápidamente la opción que desee explorar.

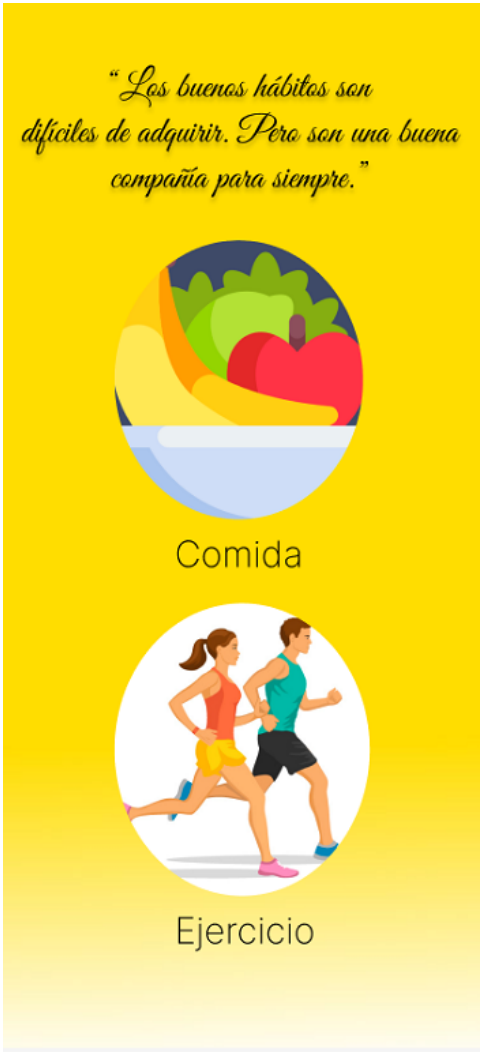


Figura 4.11.: Mockup: Comida o Ejercicio

La siguiente captura del mockup se muestra en la Figura 4.12 y corresponde a la página a la que el usuario es dirigido después de seleccionar la opción de comida en la pantalla inicial (Figura 4.11). En esta página, se presentan dos iconos grandes que ofrecen al usuario dos opciones específicas relacionadas con la comida: el apartado de dieta y el apartado de lista de la compra. Estos iconos permiten al usuario acceder rápidamente a las funciones relevantes y facilitan la navegación dentro de la aplicación. El apartado de dieta proporciona información sobre los planes de alimentación personalizados, mientras que el apartado de lista de la compra ayuda al usuario a mantenerse organizado al mostrar los productos necesarios para seguir su plan de dieta.

Figura 4.12.: Mockup: Dieta o Lista de la compra

La Figura 4.13 muestra la página del mockup dedicada a la visualización de la dieta personalizada. En esta página, el usuario puede observar de manera clara y organizada los platos de cada comida a lo largo del día. Además, se incluye una navegación sencilla que permite desplazarse fácilmente entre los diferentes días de la semana para acceder a las dietas correspondientes. Esta funcionalidad proporciona al usuario una visión global de su plan de alimentación a lo largo del tiempo. La disposición clara y la navegación intuitiva permiten al usuario acceder rápidamente a la información relevante y seguir su dieta de manera efectiva.

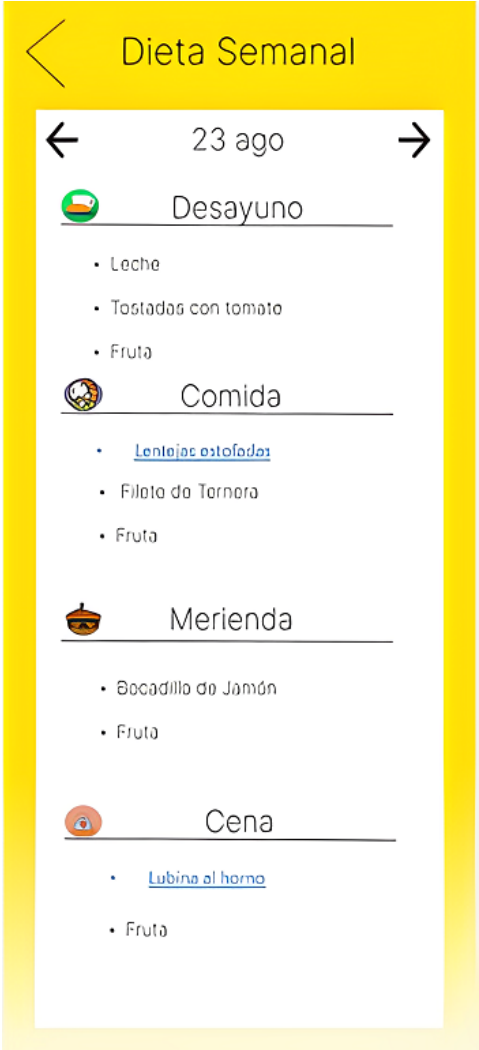


Figura 4.13.: Mockup: Dieta

En la Figura 4.14, se muestra la página del mockup dedicada a la visualización del ejercicio pautado. En esta página, el usuario puede observar de manera clara y organizada los ejercicios a realizar a lo largo del día. Además, se incluye una navegación sencilla que permite desplazarse fácilmente entre los diferentes días de la semana para acceder a las ejercicios correspondientes. Esta funcionalidad proporciona al usuario una visión global de su plan de ejercio a lo largo del tiempo. Además, en esta captura se puede observar como se muestra el contenido multimedia que se utilizará para apoyar el plan de entrenamiento en este caso, pero esto mismo ocurriría con el plan de alimentación, aunque como se verá en el capítulo 5, finalmente se optó por mostrar los contenidos de los platos y ejercicio en una página aparte.

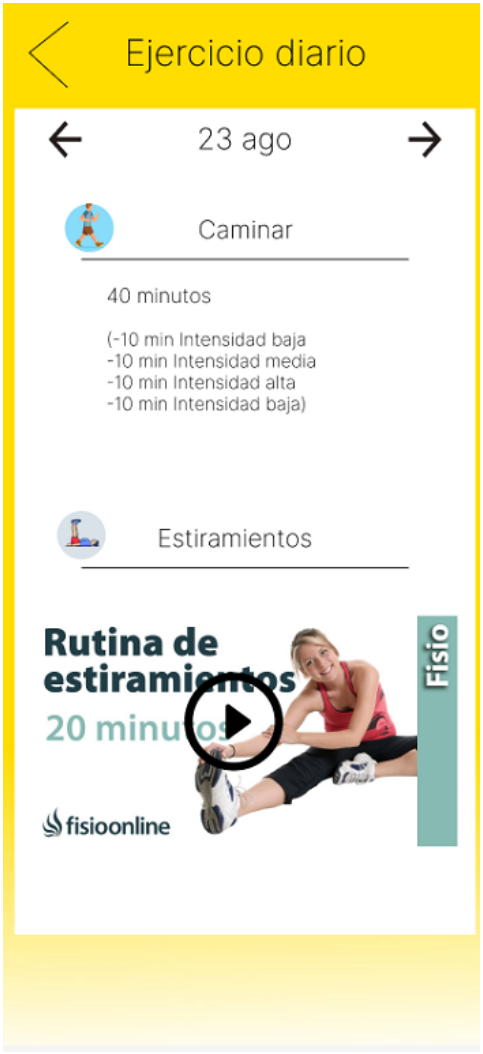


Figura 4.14.: Mockup: Ejercicio

La siguiente captura del mockup (Figura 4.15), muestra la página dedicada a la lista de la compra. En esta sección, los elementos de la lista se encuentran ordenados por fecha de uso, lo que facilita la organización y planificación de las compras. Cada elemento de la lista cuenta con una casilla de verificación que permite marcar si ya ha sido adquirido o no. Además, se muestra la fecha de uso correspondiente a cada elemento y la receta en la que será utilizado, brindando al usuario una referencia clara y útil para sus compras. Esta funcionalidad ayuda al usuario a mantenerse organizado y asegurarse de tener todos los ingredientes necesarios para seguir su plan de dieta de manera adecuada.

Figura 4.15.: Mockup: Lista de la compra

La última captura (Figura 4.16) del mockup presenta la sección de la pregunta diaria, donde se busca recopilar el feedback y la valoración del usuario respecto a su experiencia con la dieta y el ejercicio. Cada día, el usuario podrá responder una pregunta relacionada con su progreso, motivación o satisfacción con el plan de alimentación y rutina de ejercicio. Esta interacción permite al usuario expresar sus opiniones, sentimientos y cualquier dificultad que haya experimentado durante el proceso. El feedback recopilado a través de estas preguntas diarias será valioso para el enfermero encargado del tratamiento, ya que le brindará información importante sobre la efectividad del plan, la adherencia del paciente y posibles ajustes que se deban realizar. De esta manera, se fomenta una comunicación bidireccional entre el usuario y el profesional de la salud, promoviendo una atención personalizada y mejorando continuamente el programa de tratamiento.

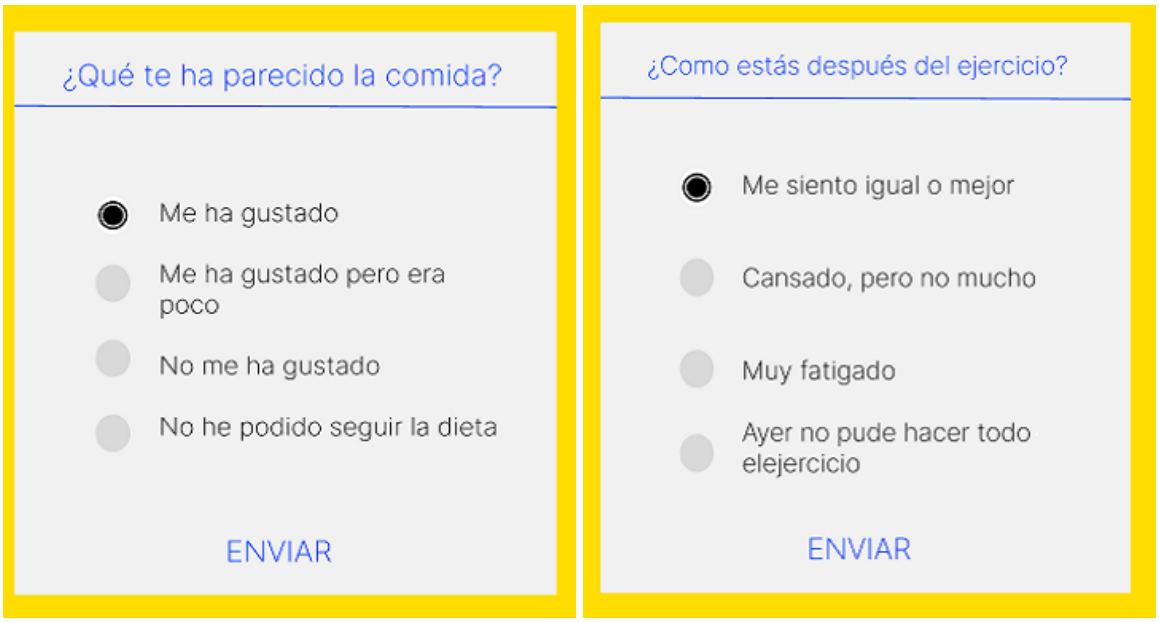


Figura 4.16.: Mockup: Preguntas al Usuario

Finalmente, con el objetivo de facilitar la comprensión para los pacientes, se decidió presentar las opciones de respuesta a nuestra pregunta diaria en forma de emoticonos en lugar de texto como se planteó en la Figura 4.16.

4.4. Fase de desarrollo

La fase de desarrollo del proyecto Food&Move es una etapa crucial en la creación de la aplicación móvil para pacientes. Durante esta fase, se llevó a cabo la programación y la construcción de la aplicación, siguiendo los requisitos y las especificaciones establecidas en las fases anteriores.

En primer lugar, se realizó un análisis detallado de los requisitos funcionales y no funcionales de la aplicación descritos en la sección 4.2. También se tuvieron en cuenta aspectos como la usabilidad, la seguridad y la compatibilidad con diferentes dispositivos móviles. Se trabajó con las tecnologías escogidas en la sección 4.2.4 que permiten un desarrollo eficiente y escalable, teniendo en cuenta las características y limitaciones de la plataforma móvil objetivo.

A continuación, se llevó a cabo la implementación de las funcionalidades de la aplicación. Esto implica la codificación de los diferentes módulos y componentes, el manejo de datos, la integración con la base de datos compartida con la plataforma web

de los enfermeros, y la creación de interfaces de usuario intuitivas y atractivas. Se trataron de seguir las mejores prácticas de desarrollo de software, como la modularidad, el uso de patrones de diseño y la realización de pruebas unitarias y de integración. Durante todo el proceso de desarrollo, se llevó a cabo un seguimiento y una revisión constante del código, con el objetivo de corregir posibles errores o problemas. Se realizaron pruebas exhaustivas en diferentes escenarios y se fueron aplicando ajustes según los resultados obtenidos. Además, fue necesaria la colaboración y la comunicación constante con el resto del equipo de desarrollo, para recoger sus comentarios y realizar mejoras iterativas en la aplicación.

El resultado del desarrollo de esta fase se encuentra detallado en la 5, donde se explica la aplicación en profundidad.

4.5. Fase de pruebas

La fase de pruebas es una etapa crucial en el proceso de desarrollo de la aplicación. En esta fase, se llevarán a cabo diferentes pruebas y evaluaciones con el objetivo de garantizar la calidad, el rendimiento y la usabilidad de la aplicación antes de su lanzamiento final. El primer paso en la fase de pruebas es la identificación y elaboración de casos de prueba. Estos casos de prueba van a tratar de representar todos los diferentes escenarios y situaciones que pueden ocurrir durante el uso de la aplicación. Se diseñan pruebas para verificar el correcto funcionamiento de las funcionalidades clave, la interacción con la base de datos y la integración con la plataforma web de los enfermeros. Además, se evaluará la respuesta de la aplicación ante situaciones inesperadas, como errores de conexión o situaciones de estrés.

Una vez definidos los casos de prueba, se procederá a realizar pruebas funcionales. Estas pruebas tienen como objetivo verificar que todas las funcionalidades de la aplicación se comporten como se esperaba y cumplan con los requisitos establecidos. Además de las pruebas funcionales, se realizarán pruebas de usabilidad. Estas pruebas se centran en evaluar la experiencia del usuario al interactuar con la aplicación.

Durante las pruebas, se registrarán y documentarán todos los errores, fallos o comportamientos inesperados que se encuentren. Estos registros permitirán realizar un seguimiento de los problemas identificados y llevar a cabo su posterior corrección y resolución. Es importante contar con un proceso de reporte de errores claro y es-

tructurado, para que el equipo de desarrollo pueda abordarlos de manera eficiente. Una vez finalizadas las pruebas, se realizará una revisión exhaustiva de los resultados obtenidos. Se analizarán los registros de errores y los resultados de las pruebas funcionales y de usabilidad. Con base en esta información, se realizarán los ajustes y mejoras necesarios en la aplicación. Estos cambios pueden incluir correcciones de errores, optimizaciones de rendimiento o ajustes en la interfaz de usuario.

Cabe destacar que esta fase de pruebas se divide en varias etapas, que incluyen las pruebas internas y las pruebas de compatibilidad. Las pruebas internas se enfocan en verificar la funcionalidad de la aplicación, identificar posibles errores o fallos en el código, y asegurar su correcto comportamiento. Durante esta etapa, se ejecutan pruebas unitarias para cada componente de la aplicación, comprobando que realicen sus funciones de manera adecuada y sin errores. Además, se realizan pruebas de integración, donde se verifican las interacciones entre los distintos módulos y componentes de la aplicación, asegurando su correcta comunicación y funcionamiento conjunto. Una vez completadas las pruebas internas, se procede al proceso de compilación y generación del paquete de distribución de la aplicación. Este paquete incluye todos los archivos necesarios para instalar y ejecutar la aplicación en dispositivos móviles. Durante este proceso, se realiza una compilación final de la aplicación, se optimiza su rendimiento y se configuran los ajustes necesarios para su correcta ejecución.

Posteriormente, se pasa a subir la app a los market, esto implica realizar un proceso de compilación y empaquetado para las plataformas objetivo, iOS y Android. Se generan los archivos de instalación correspondientes, los archivos .apk para Android y los archivos .ipa para iOS. Además, se configuran las cuentas de desarrollador necesarias en los respectivos mercados de aplicaciones, Google Play Store y App Store de Apple. Una vez esté la app en los market, se llevan a cabo las pruebas de compatibilidad, donde se verifica que la aplicación sea compatible con una amplia gama de dispositivos móviles y los sistemas operativos para los que esta diseñada. Esto implica probar la aplicación en diferentes modelos de dispositivos, tamaños de pantalla, resoluciones y versiones de los sistemas operativos. Esta etapa es crucial para asegurarse de que la aplicación se adapte y funcione correctamente en diferentes entornos móviles.

Además de las pruebas internas y de compatibilidad, es importante realizar pruebas de rendimiento, donde se evalúa el rendimiento de la aplicación en términos de velocidad, capacidad de respuesta y consumo de recursos. También se llevan a cabo

pruebas de seguridad, para detectar posibles vulnerabilidades y proteger los datos de los usuarios.

Una vez completadas todas las pruebas, se recopila el feedback y se realizan los ajustes necesarios para corregir errores y mejorar la experiencia de uso. Esta retroalimentación es esencial para garantizar que la aplicación cumpla con los requisitos establecidos y brinde una experiencia satisfactoria a los usuarios.

4.6. Fase final

La fase final del proyecto es el último paso en el proceso de desarrollo de la aplicación. En esta etapa, se llevan a cabo diversas actividades clave que culminan en el lanzamiento exitoso y la preparación para el mantenimiento futuro.

En primer lugar, se realiza un refinamiento del diseño de la aplicación. Esto implica realizar ajustes finales en la interfaz de usuario, mejorando la usabilidad y la experiencia del usuario, teniendo en cuenta las observaciones en las pruebas exhaustivas de la sección 4.5 y se implementan mejoras para garantizar un aspecto atractivo y una navegación fluida.

Una vez que la aplicación está completamente lista, se procede a la documentación necesaria para el mantenimiento futuro. Esto incluye la creación de manuales de usuario presentados en el Apéndice B, guías de instalación y configuración, si fueran necesarias, y documentación técnica para los desarrolladores que se encargarán del mantenimiento y las futuras actualizaciones de la aplicación. Esta documentación es esencial para garantizar una gestión efectiva y un proceso de mantenimiento sin problemas.

Finalmente, se realiza una revisión final de todos los aspectos del proyecto, asegurándose de que todos los requisitos y objetivos iniciales se hayan cumplido. Se realiza una evaluación exhaustiva para asegurar que la aplicación esté lista para su lanzamiento y que cumpla con los estándares de calidad establecidos. Cualquier problema o error identificado durante esta etapa se aborda y se realiza la corrección correspondiente.

CAPÍTULO 5

IMPLEMENTACIÓN DEL SISTEMA

"Stay Hungry,
Stay Foolish."
— Steve Jobs, *2005*.

En este capítulo, se presentará la implementación de la aplicación móvil desarrollada en el marco de este Trabajo de Fin de Grado. Se describirán los aspectos técnicos y metodológicos relacionados con la construcción de la app. Además, se abordarán

los desafíos encontrados durante la implementación y se detallarán las soluciones adoptadas. El objetivo principal de este capítulo es proporcionar una visión general de la implementación de la aplicación, destacando los aspectos relevantes que han permitido llevar a cabo con éxito el desarrollo de la misma.

En primer lugar, es necesario establecer algunos conceptos previos para comprender el proceso de construcción de la aplicación móvil en Angular. Posteriormente, se procederá a detallar cómo se ha introducido cada una de las funcionalidades en la aplicación. Se explicará de forma exhaustiva el desarrollo e implementación de los casos de uso identificados, desde la autenticación y gestión de sesiones hasta la consulta de dietas, ejercicios y otros elementos clave. Se describirá el proceso de integración de APIs externas, el manejo de datos y demás detalles relevantes. Además, se destacarán las decisiones de diseño y las técnicas utilizadas para optimizar el rendimiento y la usabilidad de la app.

A través de este enfoque gradual y detallado, se busca proporcionar una visión completa de la implementación de la aplicación, permitiendo a los lectores comprender tanto los aspectos conceptuales como las soluciones prácticas utilizadas para construir una aplicación móvil

5.1. Conceptos previos

En la implementación de la aplicación en Angular, es fundamental comprender algunos conceptos previos que ayudarán a familiarizarse con la estructura y los elementos clave del desarrollo. A continuación, se describirán brevemente algunos de estos conceptos.

5.1.1. Estructura de una app

El desarrollo de la aplicación en Angular requiere una estructura organizada de archivos que cumpla con las convenciones y buenas prácticas recomendadas. A continuación, se presenta una descripción de los archivos más relevantes y su utilidad en el proceso de desarrollo de la aplicación.

El árbol de archivos de la aplicación en Angular suele estar compuesto por los siguientes elementos, entre otros: [12]

- Carpeta "src": Esta carpeta es el corazón de la aplicación y contiene la mayor parte de los archivos relevantes para el desarrollo. En ella se encuentran los siguientes archivos y carpetas:
  - Carpeta "app": Esta carpeta contiene los archivos principales de la aplicación, incluyendo los componentes, servicios, modelos y otros elementos que definen su funcionamiento.
  - Archivo "app.module.ts": Este archivo es el punto de entrada de la aplicación y se encarga de la configuración global, la importación de módulos y la declaración de componentes.
  - Archivo "app.component.ts": Este archivo define el componente raíz de la aplicación, el cual es el primer componente que se carga al iniciar la aplicación.
  - Archivo "app.component.html": Este archivo contiene la plantilla HTML asociada al componente raíz y define la estructura de la aplicación.
  - Archivo "app.component.scss": Este archivo permite aplicar estilos específicos al componente raíz.
- Carpeta "assets": En esta carpeta se almacenan los recursos estáticos de la aplicación, como imágenes, iconos, archivos de configuración, entre otros.
- Carpeta "environments": Esta carpeta contiene los archivos de configuración específicos para cada entorno, como desarrollo y producción. Estos archivos permiten definir variables de entorno y ajustes específicos según el entorno de ejecución.
- Archivo "index.html": Este archivo es el punto de entrada de la aplicación y define la estructura HTML básica en la que se cargará la aplicación.

Para obtener una comprensión más detallada y precisa de la estructura de archivos en Angular, se recomienda consultar la documentación oficial de Angular. [13] Esta referencia proporciona información completa y actualizada sobre la estructura de archivos y el uso adecuado de cada uno de ellos.

5.1.2. Archivos principales

En el desarrollo de una aplicación en Ionic, además de los archivos comunes de Angular, se utilizan otros elementos específicos de Ionic que ayudan a construir la interfaz de usuario y gestionar la lógica de la aplicación. A continuación, se describen algunos de los archivos principales utilizados en una aplicación Ionic [15]:

- Pages: Son componentes que representan las diferentes pantallas de la aplicación. Cada página está compuesta por tres archivos principales:
  - Un archivo TypeScript (.ts) que contiene la lógica de la página y define sus propiedades y métodos.
  - Un archivo HTML (.html) que define la estructura y el contenido de la página mediante etiquetas HTML y componentes de Ionic.
  - Un archivo de estilos (.scss) que permite aplicar estilos específicos a la página.
- Services: Los servicios son clases que encapsulan la lógica y la funcionalidad reutilizable de la aplicación. Estos archivos suelen tener una extensión .service.ts y se utilizan para realizar solicitudes a APIs, gestionar el estado de la aplicación, almacenar y compartir datos, entre otros.
- Models: Los modelos son interfaces o clases que definen la estructura de los datos utilizados en la aplicación. Estos archivos se utilizan para tipar los datos y proporcionar una estructura coherente en toda la aplicación.
- Components: Además de las "pages", Ionic permite la creación de componentes reutilizables que pueden ser utilizados en diferentes partes de la aplicación, aunque en esta app no han sido necesarios. Estos componentes suelen tener su propio archivo TypeScript, HTML y de estilos.
- Configuración: Ionic utiliza archivos de configuración para definir aspectos específicos de la aplicación, como los ajustes de enrutamiento, el tema visual, la configuración de plugins, entre otros. Estos archivos suelen estar ubicados en la carpeta "src/app"se pueden modificar según las necesidades del proyecto.

Para obtener una comprensión más detallada y precisa de la estructura de estos archivos en Ionic, es recomendable consultar la documentación oficial de Ionic para obtener información actualizada y detallada sobre la estructura de archivos y las mejores prácticas en el desarrollo de aplicaciones Ionic. [15]

5.2. *app.routing.module.ts*

El archivo 'app.routing.module.ts' es un archivo relevante ya que define las rutas de navegación de la aplicación utilizando el enrutador de Angular. En él, se importan los módulos necesarios y se definen una serie de rutas con sus respectivas configuraciones. Cada ruta tiene un path que indica la URL relativa a la que se accederá, y se utiliza la función *loadChildren* para cargar de manera perezosa los módulos asociados a cada ruta. Por ejemplo, la ruta 'inicio' carga el módulo *InicioPageModule* cuando se accede a la URL '/inicio'. De manera similar, otras rutas como 'login', 'comida', 'dieta', 'ejercicio', entre otras, también cargan sus respectivos módulos correspondientes.

Además de la definición de las rutas, se especifican los *canLoad guards* para cada ruta. Estos guards, como *AuthGuard* y *AutoLoginGuard*, son utilizados para controlar el acceso a las rutas en función de la autenticación y el inicio de sesión automático. [5] Por ejemplo, la ruta 'inicio' solo puede cargarse si el usuario está autenticado, por lo que se utiliza el guard *AuthGuard*. Finalmente, en el módulo *AppRoutingModule*, se importa el módulo *RouterModule* y se utiliza el método*forRoot* para configurar las rutas principales de la aplicación. También se especifica la estrategia de precarga de todos los módulos utilizando *PreloadAllModules*, lo que permite cargar de forma anticipada los módulos en segundo plano para mejorar el rendimiento de la aplicación. Este archivo permitirá la navegación entre páginas y que se cargue el login si el usuario no está autenticado y la página de inicio cuando lo esté.

```
1  import { NgModule } from '@angular/core';
2  import { PreloadAllModules, RouterModule, Routes } from '@angular/router';
3  import { AuthGuard } from './guards/auth.guard';
4  import { AutoLoginGuard } from './guards/auto-login.guard';

5  const routes: Routes = [

6    {
7      path: '',
8      redirectTo: 'login',
9      pathMatch: 'full'
10   },
11   {
12     path: 'inicio',
13     loadChildren: () => import('./pages/inicio/inicio.module').then( m => m.InicioPageModule),
14     canLoad: [AuthGuard]
15   },
16   {
17     path: 'login',
18     loadChildren: () => import('./pages/login/login.module').then( m => m.LoginPageModule),
19     canLoad: [AutoLoginGuard]
```

```
20    },
21    {
22      path: 'comida',
23      loadChildren: () => import('./pages/comida/comida.module').then( m => m.ComidaPageModule),
24      canLoad: [AuthGuard]
25    },
26    {
27      path: 'dieta',
28      loadChildren: () => import('./pages/dieta/dieta.module').then( m => m.DietaPageModule),
29      canLoad: [AuthGuard]
30    },
31    {
32      path: 'ejercicio',
33      loadChildren: () => import('./pages/ejercicio/ejercicio.module').then( m => m.EjercicioPageModule),
34      canLoad: [AuthGuard]
35    },
36    {
37      path: 'ejercicio2',
38      loadChildren: () => import('./pages/ejercicio2/ejercicio2.module').then( m =>
        ↪ m.Ejercicio2PageModule),
39      canLoad: [AuthGuard]
40    },
41    {
42      path: 'listacompra',
43      loadChildren: () => import('./pages/listacompra/listacompra.module').then( m =>
        ↪ m.ListacompraPageModule),
44      canLoad: [AuthGuard]
45    },
46    {
47      path: 'comida2',
48      loadChildren: () => import('./pages/comida2/comida2.module').then( m => m.Comida2PageModule),
49      canLoad: [AuthGuard]
50    },
51  ];

52  @NgModule({
53    imports: [
54      RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })
55    ],
56    exports: [RouterModule]
57  })
58  export class AppRoutingModule { }
```

Código fuente 5.1.: authentication.service.ts

5.3. Login

La funcionalidad de inicio de sesión en la aplicación móvil Food&Move permite a los usuarios acceder de forma segura a su cuenta personalizada. Al abrir la aplicación,

los usuarios son recibidos con una pantalla de inicio de sesión donde se les solicita ingresar su número de teléfono y contraseña. Una vez que se ingresan los datos (lo hará el enfermero/a para facilitar el uso de la app a los pacientes), se realiza una validación en el servidor backend para verificar la autenticidad de las credenciales proporcionadas. En caso de que las credenciales sean válidas, se redirige al usuario a la pantalla principal de la aplicación, donde puede acceder a todas las funcionalidades y recursos personalizados según su perfil. Además, se guarda una sesión activa para evitar la necesidad de volver a iniciar sesión en futuras aperturas de la aplicación, proporcionando una experiencia de uso más cómoda. En caso de que las credenciales no sean válidas o haya algún problema en el proceso de inicio de sesión, se muestra un mensaje de error adecuado en la pantalla de inicio de sesión, informando al usuario sobre la situación y brindando la oportunidad de corregir los datos ingresados.
En el archivo "authentication.service.ts", se define el servicio *AuthenticationService* que maneja la lógica de autenticación. Primero se importan las dependencias necesarias, como *HttpClient* [3] y los modelos de datos.

```
1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3  import { BehaviorSubject, Observable } from 'rxjs';
4  import { environment } from 'src/environments/environment';
5  import { AuthResponseModel } from '../../models/auth-response.model';
6  import { AuthRequestModel } from '../../models/auth-request.model';
7  import { PatientModel } from 'src/app/models/patient.model';
8  import { PatientsService } from '../patients.service';

9  @Injectable({
10   providedIn: 'root'
11 })
12 export class AuthenticationService {
13   private userLogged = new BehaviorSubject<PatientModel | null>(null);
14   public user$: Observable<PatientModel | null> = this.userLogged;

15   constructor(private http: HttpClient, private patientsService: PatientsService) {
16   }

17   // ...
18 }
```

Código fuente 5.2.: authentication.service.ts

```
1  export interface AuthRequestModel {
2      phone: string;
3      password: string;
4  }
```

Código fuente 5.3.: auth-request.model.ts

```
import { UserModel } from "./user.model";

export interface AuthResponseModel {
    token: string;
    user: UserModel;
}
```

Código fuente 5.4.: auth-response.model.ts

```
export interface UserModel {
    _id: string;
    email: string;
    password: string;
    isEmployee: boolean;
    phone:string;
}
```

Código fuente 5.5.: user.model.ts

En *AuthenticationService*, se define *userLogged* como un *BehaviorSubject* que almacena el estado del usuario autenticado. La propiedad user$ expone un Observable que se puede suscribir para recibir actualizaciones del estado del usuario.

A continuación, se muestra el método *getNewToken* que se encarga de enviar una solicitud POST [21] al servidor backend para obtener un token de autenticación:

```
getNewToken(user: AuthRequestModel): Observable<AuthResponseModel> {
  console.log(user);
  return this.http.post<AuthResponseModel>(`${environment.api}/auth/loginPatient`, user);
}
```

Código fuente 5.6.: Método getNewToken

En este fragmento, *getNewToken* recibe un objeto *AuthRequestModel* que contiene los datos de inicio de sesión del usuario. Luego, se realiza una solicitud HTTP POST [21] a la ruta «auth/loginPatient» del backend, utilizando el *HttpClient* para enviar los datos de inicio de sesión al servidor y obtener una respuesta que contiene el token de autenticación [25].

A continuación, se presenta el método *setSession* que se encarga de establecer la sesión del usuario:

```
setSession(): void {
  const myUser = this.userLogged.getValue();
  if (!myUser && this.isLogin()) {
    this.patientsService.getPatientByEmail(this.email!)
      .subscribe(
        res => {
          this.userLogged.next(res);
        },
        err => {
          console.log(err);
```

```
11            this.logout();
12          }
13        )
14    }
15  }
```

Código fuente 5.7.: Método setSession

En este fragmento, *setSession* verifica si el usuario está autenticado y si existe en la sesión. Si es así, se realiza una solicitud al servidor a través del servicio *PatientsService* para obtener la información completa del usuario. Una vez que se recibe la respuesta exitosa, se actualiza el estado del usuario autenticado mediante *userLogged.next(res)*.
El método *getPatientByEmail* del servicio *PatientService* se utiliza para obtener la información de un paciente a partir de su dirección de correo electrónico. El método toma un parámetro email, que es la dirección de correo electrónico del paciente que se desea buscar. Utilizando *HttpClient* [3] proporcionado por Angular, se envía una solicitud HTTP POST [21] al servidor. La URL utilizada se construye utilizando el valor de *environment.api*, que es la URL base de la API del servidor, junto con el segmento de ruta "/patients/lookUp". Esto indica al servidor que se desea buscar un paciente por su dirección de correo electrónico. Se crea un objeto en el cuerpo de la solicitud HTTP que contiene la propiedad email, cuyo valor es el correo electrónico proporcionado como parámetro. El servidor procesa la solicitud y busca en su base de datos el paciente que coincida con el correo electrónico proporcionado. El servidor devuelve la información del paciente como respuesta a la solicitud HTTP. El método *http.post* devuelve un observable del tipo Observable<PatientModel>, lo que significa que se espera una respuesta del servidor que será del tipo *PatientModel*, que es el modelo de datos utilizado para representar la información de un paciente.

```
1   getPatientByEmail (email: string): Observable<PatientModel> {
2     return this.http.post<PatientModel>(`${environment.api}/patients/lookUp`, {email});
3   }
```

Código fuente 5.8.: Método getPatientByEmail

Continuando, se presenta el método *login* que se invoca cuando se ha realizado con éxito el inicio de sesión:

```
1  login(authResponse: AuthResponseModel): void {
2    this.setToken(authResponse.token);
3    this.setEmail(authResponse.user.email);
4    this.setSession();
5  }
```

Código fuente 5.9.: authentication.service.ts

En este fragmento, *login* recibe un objeto *AuthResponseModel* que contiene el token de autenticación y el correo electrónico del usuario. Primero, se llama al método *setToken* para guardar el token en el almacenamiento local del navegador y luego se llama a *setEmail* para guardar el correo electrónico del usuario. A continuación, se invoca *setSession* para obtener la información completa del usuario y actualizar el estado del usuario autenticado.
Por último, se muestra el método *logout* que se utiliza para cerrar la sesión del usuario:

```
1 logout(): void {
2   localStorage.removeItem('token');
3   localStorage.removeItem('email');
4   this.userLogged.next(null);
5 }
```

Código fuente 5.10.: Método logout

En este fragmento, logout elimina el token y el correo electrónico almacenados en el almacenamiento local del navegador. Luego, establece el estado del usuario autenticado en *null* a través del BehaviorSubject *userLogged*.
Es importante tener en cuenta que este fragmento de código utiliza *localStorage* para almacenar el token y el correo electrónico en el navegador durante la fase de desarrollo porque el simulador lo permite. Sin embargo, al compilar la aplicación para dispositivos móviles, se deberá reemplazar localStorage por una solución de almacenamiento nativa como *NativeStorage* para mantener la sesión abierta correctamente en la aplicación compilada, ya que como se comentó en el capítulo 4, se busca que el enfermero/a inicie sesión al paciente al instalar la app y esta sesión se mantega abierta. La forma de importar y configurar adecuadamente el plugin NativeStorage en la aplicación se puede consultar en la documentación oficial del plugin. [17]
Se han expuesto algunos de los fragmentos de código clave relacionados con el inicio de sesión en la aplicación. Estos fragmentos se utilizan para realizar solicitudes al servidor, gestionar el estado de autenticación del usuario y almacenar la información necesaria en el almacenamiento.
El archivo "login.page.html"contiene el código HTML de la página de inicio de sesión. A continuación, se explica su estructura y elementos clave: [11]

- <ion-content>: Este elemento encapsula el contenido de la página y se utiliza para agregar desplazamiento vertical si el contenido excede el espacio disponible. [15]
- y <img>: Estos elementos se utilizan para mostrar el logotipo de la aplicación. El logotipo se encuentra en el directorio de activos (as-

sets/background/icon.png) y se muestra centrado en la página, este logotipo, como toda la iconografía, ha sido diseñado para la app.

- <form>: Este elemento engloba el formulario de inicio de sesión y utiliza la directiva (ngSubmit) para llamar al método login() en el componente cuando se envía el formulario.
- <ion-item> y <ion-input>: Estos elementos se utilizan para crear campos de entrada de texto. El primer campo utiliza formControlName="phone"para vincularlo al control correspondiente en el formulario, y el segundo campo utiliza formControlName="password". Esto permite la validación y obtención de los valores ingresados en el formulario. [15]
- <ion-button>: Este elemento representa el botón de inicio de sesión. Está deshabilitado cuando el formulario no es válido ([disabled]=iform.valid") y utiliza la directiva type="submit"para enviar el formulario cuando se hace clic en él. [15]

```
1  <ion-content>
2          <div style="text-align: center">
3          <img src="assets/background/icon.png" alt="logo" />
4      </div>
5          <form (ngSubmit)="login()" [formGroup]="form">
6                  <div class="input-group">
7                          <ion-item>
8                                  <ion-input type="phone" placeholder="Teléfono"
                                   ↪  formControlName="phone"></ion-input>
9                          </ion-item>
10                         <ion-item>
11                                 <ion-input type="password" placeholder="Password"
                                   ↪  formControlName="password"></ion-input>
12                         </ion-item>
13                 </div>
14                 <ion-button class="ion-margin-vertical"
15                 color="secondary"
16                 type="submit"
17                 expand="block"
18                 [disabled]="!form.valid">Iniciar Sesión</ion-button>
19         </form>
20 </ion-content>
```

Código fuente 5.11.: login.page.html

El archivo login.page.scss contiene estilos en formato SCSS (Sass) para personalizar la apariencia de la página de inicio de sesión. A continuación, se explica su estructura y las reglas de estilo utilizadas: [24]

- ion-content: Esta regla se utiliza para aplicar estilos al elemento <ion-content>. En este caso, se ajusta el espaciado vertical (–padding-top) y los márgenes laterales (–padding-start y –padding-end) para crear un diseño centrado en la página. [15]
- .input-group: Esta regla se utiliza para aplicar estilos al contenedor del grupo de entrada de texto. Se establece un fondo blanco (background: #fff), se aplica un borde redondeado (border-radius: 10px), se oculta el contenido que excede el tamaño del contenedor (overflow: hidden) y se agrega un margen inferior (margin-bottom: 15px).
- .errors: Esta regla se utiliza para aplicar estilos a los mensajes de error. En este caso, se establece un tamaño de fuente pequeño (font-size: small), se cambia el color del texto a blanco (color: #fff), se establece un fondo rojo (background: var(–ion-color-danger)), y se agrega un relleno en los lados y la parte superior e inferior (padding-left: 15px; padding-top: 5px; padding-bottom: 5px).

```
ion-content {
        --padding-top: 40%;
        --padding-start: 10%;
        --padding-end: 10%;
}

.input-group {
        background: #fff;
        border-radius: 10px;
        overflow: hidden;
        margin-bottom: 15px;
}

.errors {
        font-size: small;
        color: #fff;
        background: var(--ion-color-danger);
        padding-left: 15px;
        padding-top: 5px;
        padding-bottom: 5px;
}
```

Código fuente 5.12.: login.page.scss

La figura 5.1 muestra la pantalla de login donde se puede apreciar el resultado de los estilos y el HTML explicados previamente:

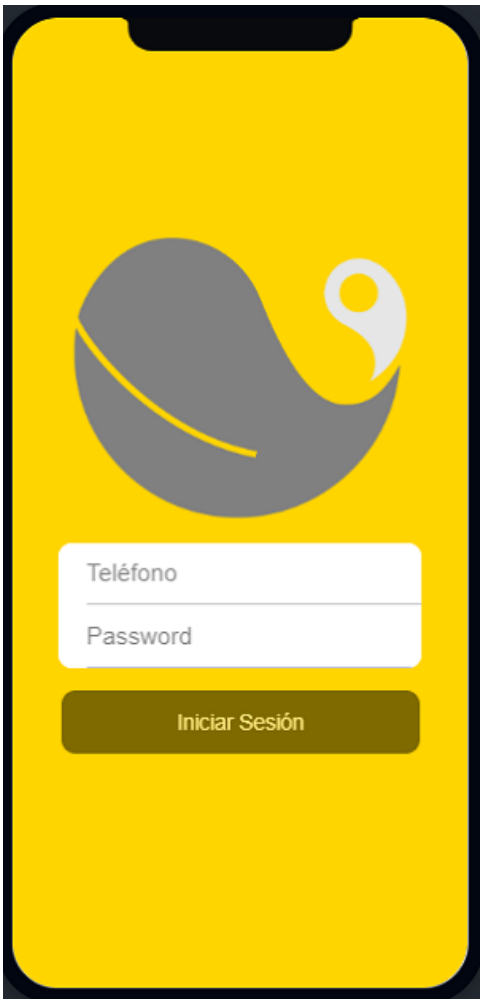


Figura 5.1.: Pantalla de login

5.4. Página de Inicio

La página de inicio de la aplicación, cuyos métodos se implementan en el archivo "inicio.page.ts", tiene como objetivo principal permitir a los usuarios elegir entre la página de datos con la comida y el ejercicio, además de en ella se implementan las preguntas diarias a los usuarios y se permite el logout entre otras cosas. A continuación, se proporciona una explicación más detallada de cada aspecto de esta página, junto con los fragmentos de código relevantes:
La página de inicio importa varios servicios y dependencias necesarios para su funcionamiento. Esto incluye el *AuthenticationService* para gestionar la autenticación del usuario, *Router* para la navegación entre páginas, *AlertController* para mostrar alertas y

el *FoodsService* y *MovesServiceService* para obtener y actualizar los datos relacionados con la comida y el ejercicio, respectivamente.

```
import { AuthenticationService } from './../../services/auth/authentication.service';
import { Router } from '@angular/router';
import { AlertController } from '@ionic/angular';
import { FoodsService } from 'src/app/services/foods.service';
import { LoaderService } from '../../services/loader.service';
import { MovesServiceService } from 'src/app/services/moves.service.service';
```

Código fuente 5.13.: Importaciones inicio.page.ts

La página de inicio define varias propiedades y variables que se utilizarán en su lógica. Estas incluyen *today*, que almacena la fecha actual, *startDate* y *endDate* para definir el rango de fechas, *todayfood* y *todaymove* para almacenar los datos de comida y ejercicio del día actual, *date* que representa el rango de fechas y *userId* para almacenar el ID del usuario autenticado. También se declara la propiedad *patient* que representa al paciente autenticado.

```
today: Date = new Date();
startDate: Date = ...
endDate: Date = ...
todayfood: FoodModel[] = [];
todaymove: MoveModel[] = [];
date: DateRangeModel = {
  startDate: this.startDate,
  endDate: this.endDate,
};
userId: string;
patient: PatientModel | null = null;
```

Código fuente 5.14.: Definiciones inicio.page.ts

El método *ngOnInit* se ejecuta al inicializar la página de inicio. Se suscribe al observable user$ del *AuthenticationService* para obtener la información del paciente autenticado. Cuando se recibe una respuesta exitosa, se almacena la información del paciente en la propiedad *patient* siguendo el modelo *PatientModel*.

```
ngOnInit() {
  this.authService.user$.subscribe(
    res => {
      this.patient = res;
    },
    err => {
      console.log(err);
    }
  );
}
```

Código fuente 5.15.: Método ngOnInit inicio.page.ts

El método *ionViewDidEnter* se ejecuta cuando la vista de la página de inicio se ha mostrado completamente. En este caso, se utiliza para realizar preguntas al usuario relacionadas con el ejercicio y la dieta. Estas preguntas se muestran a partir de una hora fijada y sólo si no se han realizado ese mismo día al paciente como se puede observar en la condición de la línea 4 del código. Se muestra una alerta utilizando el *AlertController* [16] que permite al usuario responder a la pregunta seleccionando una opción y luego las propiedades *done* y *rating* de las comidas y ejercicios valorados en cada pregunta se cargan y se actualizan con los métodos que se explican en la siguiente sección.

```
1  async ionViewDidEnter() {
2      const horaEspecifica = new Date();
3      horaEspecifica.setHours(19); //  pregunta a las 20:00
4      if (new Date() > horaEspecifica && !(await yaSeHizoLaPreguntaHoy(this.patient!._id))) { // Verifica
       ↪  si ya se hizo la pregunta hoy para el usuario actual
5        const alerta = await this.alertCtrl.create({
6          header: 'Entrenamiento',
7          message: '¿Te ha gustado el ejercicio o te ha parecido muy duro?',
8          buttons: [
9            {
10             text: '',
11             cssClass: 'glad',
12             handler: async () => {
13               this.loadmoveSetGlad();
14               await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la
                 ↪  pregunta
15             }
16           },
17           {
18             text: '',
19             cssClass: 'normal',
20             handler: async () => {
21               // Hacer algo si la respuesta es no
22               this.loadmoveSetNormal();
23               await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la
                 ↪  pregunta
24             }
25           },
26           {
27             text: '',
28             cssClass: 'sad',
29             handler: async () => {
30               this.loadmoveSetSad();
31               await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la
                 ↪  pregunta
32             }
33           }
34         ]
35       });
36       await alerta.present();
```

```
37        const alerta2 = await this.alertCtrl.create({
38          header: 'Ejercicio',
39          message: '¿Has podido realizar los ejercicios pautados?',
40          buttons: [
41            {
42              text: '',
43              cssClass: 'yes',
44              handler: async () => {
45                this.loadmoveSetDone(); //marco como done los ejercicios del día
46                await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la
                  ↪ pregunta
47              }
48            },
49            {
50              text: '',
51              cssClass: 'no',
52              handler: async () => {
53                // No hay que hacer nada si la respuesta es no porque done se inicializa como false
54                await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la
                  ↪ pregunta
55              }
56            }
57          ]
58        });
59        await alerta2.present();

60        const alerta3 = await this.alertCtrl.create({
61          header: 'Comida',
62          message: '¿Te ha gustado la comida?',
63          buttons: [
64          {
65          text: '',
66          cssClass:'glad',
67          handler: async () => {
68          // Hacer algo si la respuesta es glad
69          this.loadfoodSetGlad();
70          await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la pregunta
71          }
72          },
73          {
74          text: '',
75          cssClass: 'normal',
76          handler: async () => {
77          // Hacer algo si la respuesta es normal
78          this.loadfoodSetNormal();
79          await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la pregunta
80          }
81          },
82          {
83          text: '',
84          cssClass: 'sad',
85          handler: async () => {
86          // Hacer algo si la respuesta es sad
87          this.loadfoodSetSad();
88          await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la pregunta
```

```
89          }
90          }
91          ]
92          });

93          await alerta3.present();

94        const alerta4 = await this.alertCtrl.create({
95          header: 'Comida',
96          message: '¿Has podido realizar la dieta pautada?',
97          buttons: [
98            {
99              text: '',
100             cssClass: 'yes',
101             handler: async () => {
102               // Hacer algo si la respuesta es sí
103               await this.loadfoodSetDone(); //se carga y se pone a true la propiedad done
104               await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la
                  ↪ pregunta
105             }
106           },
107           {
108             text: '',
109             cssClass: 'no',
110             handler: async () => {
111               // si la respuesta es no, no es necesario hacer nada ya que done se inicializa como false
112               await guardarFechaPregunta(this.patient!._id); // Guardar la fecha en que se hizo la
                  ↪ pregunta
113             }
114           }
115         ]
116       });
117       await alerta4.present();
118     }
119   }
```

Código fuente 5.16.: Método ionViewDidEnter inicio.page.ts

La página de inicio implementa varios métodos para cargar y actualizar los datos de comida y ejercicio. Estos métodos utilizan el servicio *FoodsService* y *MoveService* para obtener y modificar los datos correspondientes. Aquí se muestran algunos ejemplos:

1. Método *loadfoodSetDone*: Este método carga los datos de comida del día actual y marca la propiedad 'done' como 'true' para cada registro de comida. Luego, se utiliza el método *updateFood* del servicio *FoodsService* para actualizar los registros en la base de datos. No es necesario un método *loadfoodSetUndone* que marque la propiedad como 'false' ya que la propiedad 'done' se inicializa como 'false'.

```
1   loadfoodSetDone(): void {
2     this.loaderService.isLoading.next(true);
3     this.foodsService.getFoodsByPatientAndDate(this.patient!._id, this.date)
4       .pipe(finalize(() => this.loaderService.isLoading.next(false)))
5       .subscribe(
```

```
6         res => {
7           res.forEach(food => {
8             var d = new Date(food.date.toString());
9             if (d.toDateString() == this.today.toDateString()) {
10              food.done = true;
11              this.foodsService.updateFood(food._id, food);
12              this.todayfood.push(food);
13            }
14          });
15        },
16        err => {
17          console.log(err.error.message);
18        }
19      );
20    }
```

Código fuente 5.17.: loadfoodSetDone

2. El método *getFoodsByPatientAndDate* del servicio *FoodsService* se utiliza para obtener una lista de alimentos (objetos *FoodModel*) asociados a un paciente específico y a un rango de fechas determinado. El método toma dos parámetros: ´id´ y ´date´. El parámetro ´id´ es el identificador único del paciente para el cual se desea obtener los alimentos. El parámetro ´date´ es un objeto de tipo *DateRangeModel* que contiene el rango de fechas en el que se desean buscar los alimentos. Utilizando el *HttpClient* proporcionado por Angular [3], se envía una solicitud HTTP POST al servidor [21]. La URL utilizada se construye utilizando el valor de environment.api, que es la URL base de la API del servidor, junto con el segmento de ruta ´/foods/findByPatient/$id´. Esto indica al servidor que se desea buscar los alimentos asociados al paciente con el identificador id. Se envía el objeto date como cuerpo de la solicitud HTTP. Este objeto contiene el rango de fechas en el que se desean buscar los alimentos. El servidor procesa la solicitud y busca en su base de datos los alimentos asociados al paciente y al rango de fechas especificado. El servidor devuelve la lista de alimentos como respuesta a la solicitud HTTP. El método http.post devuelve un observable del tipo Observable<FoodModel[]>, lo que significa que se espera una respuesta del servidor que será una lista de objetos *FoodModel*.

```
1   updateFood(id: string, food: FoodRequestModel) {
2     return this.http.patch<FoodModel>(`${environment.api}/foods/update/${id}`, food).subscribe(
3       res => console.log(""),
4       err => console.log("")
5     );
6   }
```

3. El método *updateFood* del servicio *FoodsService* se utiliza para actualizar la información de un alimento en la base de datos del servidor. El método toma dos parámetros: ´id´, que es el identificador único del alimento que se desea actualizar, y ´food´, que es

un objeto de tipo *FoodRequestModel* que contiene los datos actualizados del alimento. Utilizando el *HttpClient* proporcionado por Angular [3], se envía una solicitud HTTP PATCH al servidor [21]. La URL utilizada se construye utilizando el valor de environment.api, que es la URL base de la API del servidor, junto con el segmento de ruta '/foods/update/$id'. Esto indica al servidor que se desea actualizar el alimento con el ID específico. El objeto food se envía en el cuerpo de la solicitud HTTP como datos de actualización para el alimento. Se suscribe al observable devuelto por http.patch, lo que significa que se espera una respuesta del servidor.

```
1    updateFood(id: string, food: FoodRequestModel) {
2      return this.http.patch<FoodModel>(`${environment.api}/foods/update/${id}`, food).subscribe(
3        res => console.log(""),
4        err => console.log("")
5      );
6    }
```

Código fuente 5.18.: Método updateFood

4. Métodos *loadfoodSetGlad*, *loadfoodSetNormal* y *loadfoodSetSad*: Estos métodos tienen un funcionamiento similar al método *loadfoodSetDone*, pero en lugar de marcar la propiedad *done* como true, actualizan la propiedad *rating* del registro de comida con valores correspondientes a diferentes niveles de satisfacción (Glad, Normal, y Sad).

```
1  loadfoodSetGlad(): void {
2    // Código similar a loadfoodSetDone, pero actualizando la propiedad "rating"
3  }

4  loadfoodSetNormal(): void {
5    // Código similar a loadfoodSetDone, pero actualizando la propiedad "rating"
6  }

7  loadfoodSetSad(): void {
8    // Código similar a loadfoodSetDone, pero actualizando la propiedad "rating"
9  }
```

Código fuente 5.19.: Métodos carga y actualización de comidas/ejercicios

Estos son solo ejemplos de los métodos relacionados con la carga de datos de comida. En la página de inicio también se implementan métodos similares para la carga y actualización de datos de ejercicio utilizando los métodos equivalente del servicio *MovesService* llamados *getMovesByPatientAndDate* y *updateMove*.
La función *logout* en el código proporcionado se encarga de cerrar la sesión del usuario. Al hacer clic en el botón de cierre de sesión, se ejecuta la función *logout()*. La función crea una instancia de *AlertController* y configura un cuadro de diálogo de confirmación con un mensaje que pregunta al usuario si realmente quiere cerrar sesión y dos botones: *No* y *Sí*. Si el usuario selecciona *No*, se cancela la operación. Si el usuario selecciona *Sí*, se llama al método *logout*() del servicio de autenticación que se encarga de realizar las

acciones necesarias para cerrar la sesión del usuario. Luego, se redirige al usuario a la página de inicio de sesión utilizando el enrutador (this.router.navigate(['/login'])). Para que se presente el cuadro de diálogo de confirmación se utiliza el método *present*(). [16] [2]

```
  async logout(){

    let alert = this.alertCtrl.create({

      message: '¿Estás seguro de que quieres cerrar sesión? Para volver a acceder necesitarás tu usuario
      ↪  y contraseña',
      buttons: [
        {
          text: 'No',
          role: 'cancel',
          handler: () => {
            console.log('Cancel clicked');
          }
        },
        {
          text: 'Sí',
          handler: () => {
            this.authService.logout();
            this.router.navigate(['/login']);
          }
        }
      ]
    });
    (await alert).present()
  }
}
```

Código fuente 5.20.: Métodos carga y actualización de comidas/ejercicios

Como se comento en el capítulo 4 se va a mostrar una frase inspiradora al paciente en la pantalla de inicio. Para mostrar una frase aleatoria dentro de una colección se basa en la utilización de un array *Quotes* que contiene varias frases, se declara una variable *n* que almacenará un número aleatorio generado por la función *randomIntFromInterval* que recibe dos parámetros que representan el rango mínimo y máximo para generar el número aleatorio. La función *randomIntFromInterval* genera un número entero aleatorio dentro del rango especificado. Finalmente, el número aleatorio obtenido se asigna a la variable n y en el HTML de la página de inicio, se utiliza la interpolación de cadenas Quotes[n] para mostrar la frase aleatoria en la pantalla. [9]

```
  Quotes: string [] = [
    "Los buenos habitos son dificiles de adquirir pero son una buena compañía para siempre",
    "Saber comer es saber vivir",
    "Andar es el mejor ejercicio posible. Habitúate a andar muy lejos",
    "Come lo necesario, respira profundamente, vive con moderación, cultiva la alegría e interésate por
    ↪  la vida",
```

```
    "La vida empieza con una sonrisa y nada da más felicidad que sentirse saludable y lleno de energía",
    "Somos lo que comemos, pero lo que comemos nos puede ayudar a ser mucho más de lo que somos",
    "Cuanto más ejercicio hacemos, más podemos hacer",
    "El movimiento es una medicina para crear el cambio físico, emocional y mental",
  ];
  n : number = randomIntFromInterval(0, 7);

function randomIntFromInterval(min:number, max:number) { // min and max included
    return Math.floor(Math.random() * (max - min + 1) + min)
  }
```

Código fuente 5.21.: Frase pantalla inicial

También resulta interesante explicar algunas funciones que se han usado en métodos anteriormente detallados:

- La función *yaSeHizoLaPreguntaHoy* se utiliza para determinar si ya se ha realizado la pregunta al usuario en el día actual. Recibe el userId como parámetro y utiliza el Capacitor Storage [7] para obtener la fecha guardada correspondiente a ese usuario. Luego, compara la fecha guardada con la fecha actual y devuelve un valor booleano que indica si ya se hizo la pregunta hoy.
- La función *guardarFechaPregunta* se utiliza para guardar la fecha en que se realizó la pregunta al usuario. Recibe el userId como parámetro y utiliza el Capacitor Storage [7] para guardar la fecha actual correspondiente a ese usuario. La fecha se guarda utilizando el ID de usuario como parte de la clave.
- La función *scheduleDailyNotification* se encarga de programar una notificación diaria. Utiliza el plugin Capacitor Local Notifications [6] para programar una notificación que se mostrará todos los días a las 20:00 horas. La notificación servirá para recordar al usuario que responda sus preguntas diarias a la hora que están disponibles.

```
// Función para determinar si ya se hizo la pregunta hoy al usuario
async function yaSeHizoLaPreguntaHoy(userId: string) {
  const hoy = new Date().toISOString().slice(0, 10);
  const { value: fechaGuardada } = await Storage.get({ key: `fechaPregunta_${userId}` }); // Usa el ID de
  ↪  usuario como parte de la clave
  return fechaGuardada ? fechaGuardada === hoy : false;
}


// Función para guardar la fecha en que se hizo la pregunta
async function guardarFechaPregunta(userId: string) {
  const hoy = new Date().toISOString().slice(0, 10);
  await Storage.set({ key: `fechaPregunta_${userId}`, value: hoy }); // Usa el ID de usuario como parte
  ↪  de la clave
}
```

```
async function scheduleDailyNotification() {
  const notifs = await LocalNotifications.schedule({
    notifications: [
      {
        title: 'Recordatorio diario',
        body: 'Responda a las preguntas diarias',
        id: 1,
        schedule: { at: new Date(new Date().setHours(20, 0, 0, 0)) },
        sound: 'default',
        actionTypeId: '',
        extra: null,
      },
    ],
  });
  console.log('scheduled notifications', notifs);
}
```

Código fuente 5.22.: Funciones auxiliares inicio.page.ts

El archivo HTML muestra la estructura y los elementos visuales de la página de inicio. En el código HTML, se encuentran los siguientes elementos:

- <ion-button>: Se utiliza para representar botones. En este caso, se muestra un botón de «Cerrar Sesión» con un ícono de salida. Al hacer clic en el botón, se ejecuta la función logout().
- <ion-text>: Se utiliza para mostrar texto en la página. En este caso, se muestra la cita aleatoria utilizando la interpolación de cadenas (Quotes[n]).
- <ion-button>: Se utilizan dos botones para representar las opciones de «Comida» y «Ejercicio». Al hacer clic en cada botón, se redirige al usuario a las respectivas páginas comida yejercicio.

```
<ion-content>
  <ion-button class="cerrarSesion" fill="clear" (click)="logout()" color="secondary">
    <ion-icon slot="icon-only" name="log-out-outline"></ion-icon>
  </ion-button>
  <ion-text class="texto">
    <h2>"{{ Quotes[n]}}"</h2>
  </ion-text>

  <!-- Botón comida -->
  <ion-button class="botonPrincipal" fill="clear" size="large" shape="round" [routerLink]="['/comida']">
    <img src="../../../assets/icon/menu.png">
  </ion-button>
  <ion-text class="texto2">COMIDA</ion-text>

  <!-- Botón ejercicio -->
  <ion-button class="botonPrincipal" fill="clear" size="large" shape="round"
  ↪  [routerLink]="['/ejercicio']">
    <img src="../../../assets/icon/move.png">
  </ion-button>
```

```
17    <ion-text class="texto2">EJERCICIO</ion-text>
18  </ion-content>
```

Código fuente 5.23.: inicio.page.html

El archivo SCSS (inicio.page.scss) contiene los estilos personalizados para la página de inicio. A continuación, se muestra una explicación de los estilos aplicados. Los estilos personalizados se aplican a los elementos HTML en la página de inicio. Por ejemplo, .botonPrincipal establece el tamaño de fuente y el color del texto en los botones principales. .texto establece el estilo del texto de la cita. .cerrarSesion define los márgenes y el relleno para el botón de «Cerrar Sesión». .alert-button-inner define los estilos para los botones en las alertas.

```
1   .botonPrincipal {
2     color: black;
3     font-size: 70px;
4     align-self: center;
5     width: 100%;
6   }

7   .texto {
8     text-align: center;
9     font-size: 70px;
10    font-family: Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
11    margin-top: 0%;
12  }

13  .texto2 {
14    text-align: center;
15    display: block;
16    margin: 0 auto;
17    font-family: Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
18  }

19  .cerrarSesion {
20    margin-left: 85%;
21    margin-top: 5%;
22    fill: clear;
23  }

24  .alert-button-inner {
25    display: flex;
26    justify-content: center;
27    align-items: center;
28    height: 100%;
29    width: 100%;
30    padding: 0;
31  }
```

Código fuente 5.24.: inicio.page.scss

A continuación, en la Figura 5.2 se muestra una imagen de la pantalla de inicio de la aplicación. Esta imagen captura cómo se ven aplicados los estilos y el HTML explicados previamente. Además, es importante destacar que los iconos han sido diseñados específicamente para la aplicación, aportando el aspecto visual deseado. En esta imagen, se puede apreciar el resultado estético y la interfaz de usuario, que trata de ser amigable para el usuario.

Figura 5.2.: Página de Inicio

Además de la imagen de la pantalla de inicio, a continuación en las figuras 5.3, 5.4 y 5.5 se presentan otras capturas de pantalla que nos permiten apreciar el estilo aplicado en las preguntas diarias y el diálogo de cierre de sesión. Estas imágenes complementan la comprensión de la estética general de la aplicación en las funcionalidades explicadas en esta sección. En las preguntas diarias, también se han diseñado los iconos con las opciones de respuesta. Estas vistas siguen el enfoque en la usabilidad y la experiencia del usuario, ofreciendo una navegación intuitiva en toda la aplicación.

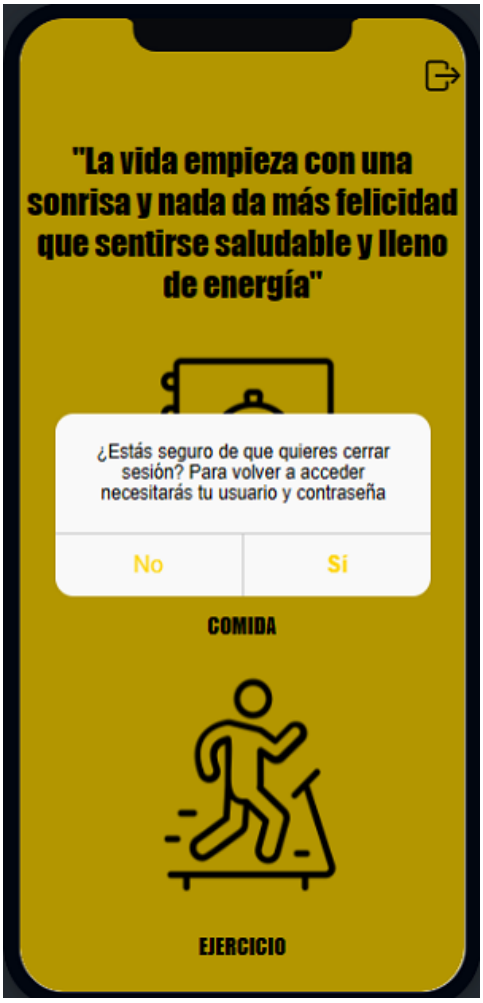


Figura 5.3.: Dialogo cierre de sesión

Figura 5.4.: Pregunta diaria

Figura 5.5.: Pregunta diaria

5.5. Página de Menú de Comida

La página del menú de «Comida» (ComidaPage) es un componente Angular que muestra opciones relacionadas con la comida al seleccionar el botón de comida en la página de inicio de la app. En el archivo TypeScript comida.page.ts, se importa la clase *Router* de '@angular/router' para permitir la navegación entre páginas. En el método *gotoInicio()*, se utiliza *this.router.navigate(['/inicio'])* para redirigir al usuario a la página de inicio cuando se hace clic en el botón de retroceso.

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
@Component({
  selector: 'app-comida',
  templateUrl: './comida.page.html',
  styleUrls: ['./comida.page.scss'],
})
export class ComidaPage implements OnInit {

  constructor(private router:Router) { }

  ngOnInit() {
  }

  gotoInicio(){
      this.router.navigate(['/inicio'])
  }
}
```

Código fuente 5.25.: comida.page.ts

En el archivo HTML, el contenido de la página se encuentra dentro de <ion-content>. Se utilizan varios elementos <ion-button>para permitir elegir las diferentes opciones relacionadas con la comida. El botón «Dieta Semanal» utiliza la directiva [routerLink]="['/dieta']"para enlazar a la ruta /dieta cuando se hace clic en él y ocurre lo mismo con el botón "Lista de la Compra". El botón de retroceso tiene la directiva (click)="gotoInicio()"para llamar al método *gotoInicio*() cuando se hace clic en él para redirigir a la página de inicio.

```
<ion-content>
  <br><br>
  <!-- Botón back -->
  <ion-button (click)="gotoInicio()" fill="clear" color="secondary">
    <ion-icon slot="icon-only" name="chevron-back-outline"></ion-icon>
    ATRÁS
  </ion-button>

  <!-- Botón Dieta Semanal -->
  <ion-button class="botonPrincipal" fill="clear" size="large" shape="round" [routerLink]="['/dieta']">
    <img src="../../../assets/icon/diet.png">
```

```
11    </ion-button>
12    <ion-text class="texto2">DIETA<br><br></ion-text>

13    <!-- Botón Lista de la compra -->
14    <ion-button class="botonPrincipal" fill="clear" color="secondary" expand="full" size="large"
      ↪  shape="round" [routerLink]="['/listacompra']">
15      <img src="../../../assets/icon/list.png">
16    </ion-button>
17    <ion-text class="texto2">LISTA DE LA COMPRA</ion-text>
18  </ion-content>
```

Código fuente 5.26.: comida.page.html

En el archivo SCSS, se aplican estilos a los elementos de la página. La clase .botonPrincipal se utiliza para estilizar los botones principales de la página de comida. Se establecen propiedades como el color (color: black), el tamaño de fuente (font-size: 80px), la alineación (align-self: center) y el ancho. La clase .texto2 se utiliza para estilizar el texto en la página, estableciendo la alineación, el display, el margen y la fuente.

```
1   .botonPrincipal {
2     color: black;
3     font-size: 80px;
4     align-self: center;
5     width: 100%;
6   }

7   .texto2 {
8     text-align: center;
9     display: block;
10    margin: 0 auto;
11    font-family: Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
12  }
```

Código fuente 5.27.: comida.page.scss

La figura 5.6 muestra la pantalla de comida donde se puede apreciar el resultado de los estilos y el HTML explicados previamente:

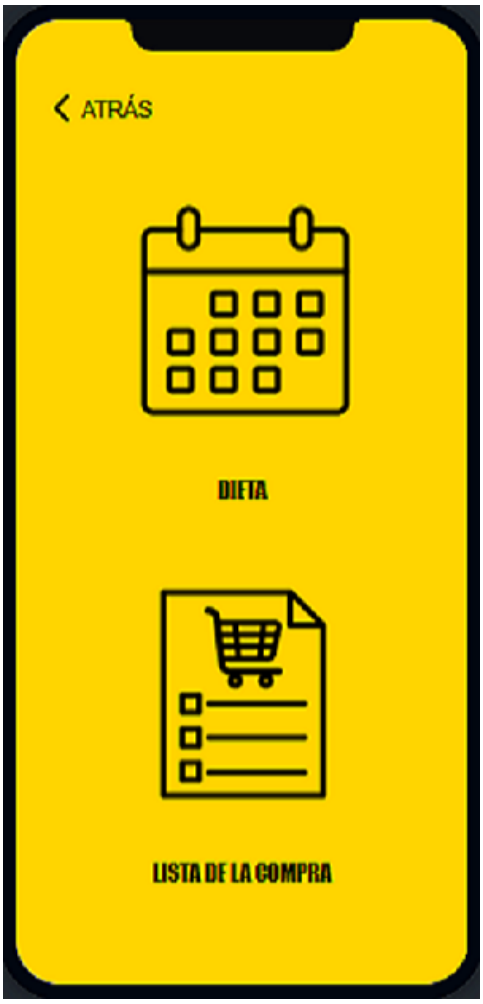


Figura 5.6.: Pantalla de Menú de comida

5.6. Página de Dieta Semanal

La página de "Dieta Semanal"(DietaPage) es un componente Angular que muestra al paciente las comidas de diferentes días.
En el archivo TypeScript "dieta.page.ts", se define la propiedad *selectedSegment*, que almacena el valor del segmento seleccionado para moverse entre los distintos días, y *segmentHidden*, un objeto que indica qué segmentos están ocultos. El método *moveSegment(val: number)* se utiliza para cambiar entre segmentos y mostrar las comidas correspondientes al día del segmento seleccionado. Luego se definen varias propiedades relacionadas con las fechas, como *startDate*, *endDate* y *dia1* o *dia2*, que representan las fechas de los diferentes días que se mostrarán en la página. También se definen los

arrays que contendrán los objetos con las comidas siguiento el formato definido en *FoodModel*.

```
  public showBackButton: boolean = true;
  public showForwardButton: boolean = true;

  selectedSegment: string = 'button2';
  segmentHidden = {
    button1: true,
    button2: false,
    button3: true,
    button4: true,
    button5: true,
    button6: true,
    button7: true
  };

  moveSegment(val: number) {
    const segments = document.querySelectorAll('ion-segment-button');
    const selectedIndex = Array.from(segments).findIndex(
      (segment: any) => segment.value === this.selectedSegment
    );
    const maxIndex = segments.length - 1;
    const newIndex = selectedIndex + val;
    if (newIndex > 0 && newIndex < maxIndex) {
      this.selectedSegment = segments[newIndex].value;
      segments[newIndex].scrollIntoView({
        behavior: 'smooth',
        block: 'center'
      });
      this.showBackButton = true;
      this.showForwardButton = newIndex < maxIndex
    } else if (newIndex == 0) {
      this.selectedSegment = segments[0].value;
      segments[0].scrollIntoView({
        behavior: 'smooth',
        block: 'center'
      });
      this.showBackButton = false;
      this.showForwardButton = true;
    } else if (newIndex == maxIndex) {
      this.selectedSegment = segments[maxIndex].value;
      segments[maxIndex].scrollIntoView({
        behavior: 'smooth',
        block: 'center'
      });
      this.showBackButton = true;
      this.showForwardButton = false;
    }
  }
```

```
  patient: PatientModel | null = null;
```

```
47    id : PatientModel | null = null;

48    addDays = (date: Date, days: number): Date => {
49      let result = new Date(date);
50      result.setDate(result.getDate() + days);
51      return result;
52    };
53    subtractDays = (date: Date, days: number): Date => {
54      let result = new Date(date);
55      result.setDate(result.getDate() - days);
56      return result;
57    };



58    dia2: Date = new Date('03-12-2023'); //Definimos las fechas de de cada día que se mostrará
59    startDate: Date = this.subtractDays(this.dia2,2);
60    dia1: Date = this.subtractDays(this.dia2,1);
61    dia3: Date = this.addDays(this.dia2, 1);
62    dia4: Date = this.addDays(this.dia2, 2);
63    dia5: Date = this.addDays(this.dia2, 3);
64    dia6: Date = this.addDays(this.dia2, 4);
65    endDate: Date = this.addDays(this.dia2, 5);

66    date: DateRangeModel = {
67      startDate: this.startDate,
68      endDate :this.endDate,
69    };

70    desday1: FoodModel[] = []; //Arrays que contendrán los desayunos de cada dia
71    desday2: FoodModel[] = [];
72    desday3: FoodModel[] = [];
73    desday4: FoodModel[] = [];
74    desday5: FoodModel[] = [];
75    desday6: FoodModel[] = [];
76    desday7: FoodModel[] = [];

77    comday1: FoodModel[] = []; //Arrays que contendrán las comidas de cada dia
78    comday2: FoodModel[] = [];
79    comday3: FoodModel[] = [];
80    comday4: FoodModel[] = [];
81    comday5: FoodModel[] = [];
82    comday6: FoodModel[] = [];
83    comday7: FoodModel[] = [];

84    merday1: FoodModel[] = []; //Arrays que contendrán las meriendas de cada dia
85    merday2: FoodModel[] = [];
86    merday3: FoodModel[] = [];
87    merday4: FoodModel[] = [];
88    merday5: FoodModel[] = [];
89    merday6: FoodModel[] = [];
90    merday7: FoodModel[] = [];

91    cenday1: FoodModel[] = []; //Array que contendrán las cenas de cada dia
```

```
92    cenday2: FoodModel[] = [];
93    cenday3: FoodModel[] = [];
94    cenday4: FoodModel[] = [];
95    cenday5: FoodModel[] = [];
96    cenday6: FoodModel[] = [];
97    cenday7: FoodModel[] = [];
```

Código fuente 5.28.: Definición de propiedades y variables en dieta.page.ts

En el método *ngOnInit()*, se obtiene el usuario actual haciendo uso del *authService*. Luego, se llama al método *loadFoods()* para cargar las comidas del paciente. El método *loadFoods()* utiliza el servicio *foodsService* para obtener las comidas según el paciente y fecha (las comidas serán objetos que seguirán el modelo definido en *food.model.ts*). Se suscribe a la respuesta y, para cada objeto, se comprueba la fecha y la comida (desayuno, comida, merienda o cena) y se asigna al array correspondiente a las comidas de cada día (desday1, comday1, etc.). Además se utilizan los métodos *sortFood()* y *getPointsDish*() para ordenar los platos de las comidas de cada día (Primer plato, segundo plato, postre).

También se define método *gotoInicio*() que se utiliza para navegar a la página de inicio cuando se hace clic en el botón «ATRÁS» como en el resto de páginas donde se usa.

Por último, el método *gotoComida2*(item: any) se utiliza para navegar al usuario a la página de detalles del ejercicio cuando se hace clic en una comida. El objeto 'item' se pasa como parámetro utilizando el servicio *navExtras* [10] y contendrá el objeto con la comida del cuál se desean consultar los detalles.

```
1   loadFoods (): void {
2       this.loaderService.isLoading.next(true);
3       this.foodsService.getFoodsByPatientAndDate(this.patient!._id, this.date)
4       .pipe(finalize(() => this.loaderService.isLoading.next(false)))
5       .subscribe(
6         res => {
7           res.forEach(food => {
8             var d = new Date(food.date.toString());
9             //Incluyo los platos en el array correspondiente a su día y comida
10            if (d.toDateString() == this.dia1.toDateString()) {
11              if (food.meal == Meal.Desayuno) {
12                this.desday1.push(food);
13              } else if (food.meal == Meal.Almuerzo) {
14                this.comday1.push(food);
15              } else if (food.meal == Meal.Cena) {
16                this.cenday1.push(food);
17              } else if (food.meal == Meal.Merienda) {
18                this.merday1.push(food);
19              }
20            }
21            else if (d.toDateString() == this.dia2.toDateString()) {
22              if (food.meal == Meal.Desayuno) {
```

```
23              this.desday2.push(food);
24            } else if (food.meal == Meal.Almuerzo) {
25              this.comday2.push(food);
26            } else if (food.meal == Meal.Cena) {
27              this.cenday2.push(food);
28            } else if (food.meal == Meal.Merienda) {
29              this.merday2.push(food);
30            }
31          }
32          else if (d.toDateString() == this.dia3.toDateString()) {
33            if (food.meal == Meal.Desayuno) {
34              this.desday3.push(food);
35            } else if (food.meal == Meal.Almuerzo) {
36              this.comday3.push(food);
37            } else if (food.meal == Meal.Cena) {
38              this.cenday3.push(food);
39            } else if (food.meal == Meal.Merienda) {
40              this.merday3.push(food);
41            }

42          }
43          else if (d.toDateString() == this.dia4.toDateString()) {
44            if (food.meal == Meal.Desayuno) {
45              this.desday4.push(food);
46            } else if (food.meal == Meal.Almuerzo) {
47              this.comday4.push(food);
48            } else if (food.meal == Meal.Cena) {
49              this.cenday4.push(food);
50            } else if (food.meal == Meal.Merienda) {
51              this.merday4.push(food);
52            }
53          }
54          else if (d.toDateString() == this.dia5.toDateString()) {
55            if (food.meal == Meal.Desayuno) {
56              this.desday5.push(food);
57            } else if (food.meal == Meal.Almuerzo) {
58              this.comday5.push(food);
59            } else if (food.meal == Meal.Cena) {
60              this.cenday5.push(food);
61            } else if (food.meal == Meal.Merienda) {
62              this.merday5.push(food);
63            }
64          }
65          else if (d.toDateString() == this.dia6.toDateString()) {
66            if (food.meal == Meal.Desayuno) {
67              this.desday6.push(food);
68            } else if (food.meal == Meal.Almuerzo) {
69              this.comday6.push(food);
70            } else if (food.meal == Meal.Cena) {
71              this.cenday6.push(food);
72            } else if (food.meal == Meal.Merienda) {
73              this.merday6.push(food);
74            }

75          }
```

```
76            else if (d.toDateString() == this.endDate.toDateString()) {
77              if (food.meal == Meal.Desayuno) {
78                this.desday7.push(food);
79              } else if (food.meal == Meal.Almuerzo) {
80                this.comday7.push(food);
81              } else if (food.meal == Meal.Cena) {
82                this.cenday7.push(food);
83              } else if (food.meal == Meal.Merienda) {
84                this.merday7.push(food);
85              }
86            }

87          });

88          //Ordeno los platos
89          this.desday1.sort((a,b) => this.sortFood(a,b));
90          this.comday1.sort((a,b) => this.sortFood(a,b));
91          this.merday1.sort((a,b) => this.sortFood(a,b));
92          this.cenday1.sort((a,b) => this.sortFood(a,b));

93          this.desday2.sort((a,b) => this.sortFood(a,b));
94          this.comday2.sort((a,b) => this.sortFood(a,b));
95          this.merday2.sort((a,b) => this.sortFood(a,b));
96          this.cenday2.sort((a,b) => this.sortFood(a,b));

97          this.desday3.sort((a,b) => this.sortFood(a,b));
98          this.comday3.sort((a,b) => this.sortFood(a,b));
99          this.merday3.sort((a,b) => this.sortFood(a,b));
100         this.cenday3.sort((a,b) => this.sortFood(a,b));

101         this.desday4.sort((a,b) => this.sortFood(a,b));
102         this.comday4.sort((a,b) => this.sortFood(a,b));
103         this.merday4.sort((a,b) => this.sortFood(a,b));
104         this.cenday4.sort((a,b) => this.sortFood(a,b));

105         this.desday5.sort((a,b) => this.sortFood(a,b));
106         this.comday5.sort((a,b) => this.sortFood(a,b));
107         this.merday5.sort((a,b) => this.sortFood(a,b));
108         this.cenday5.sort((a,b) => this.sortFood(a,b));

109         this.desday6.sort((a,b) => this.sortFood(a,b));
110         this.comday6.sort((a,b) => this.sortFood(a,b));
111         this.merday6.sort((a,b) => this.sortFood(a,b));
112         this.cenday6.sort((a,b) => this.sortFood(a,b));

113         this.desday7.sort((a,b) => this.sortFood(a,b));
114         this.comday7.sort((a,b) => this.sortFood(a,b));
115         this.merday7.sort((a,b) => this.sortFood(a,b));
116         this.cenday7.sort((a,b) => this.sortFood(a,b));
117       },
118       err => {
119         console.log(err.error.message);
120       }
121     );
122   }
```

```
123   sortFood (a: FoodModel, b: FoodModel): number {
124     const pa: number = this.getPointsDish(a.dish);
125     const pb: number = this.getPointsDish(b.dish);
126     if (pa < pb) {
127       return -1;
128     } else if (pa > pb) {
129       return 1;
130     } else {
131       return 0;
132     }
133   }

134   getPointsDish (dish: Dish): number {
135     switch(dish) {
136       case Dish.Principal: return 0;
137       case Dish.Primero: return 1;
138       case Dish.Segundo: return 2;
139       case Dish.Postre: return 3;
140       default: return 4;
141     }
142   }
143   gotoComida(){
144     this.router.navigate(['/comida'])
145   }

146   gotoComida2(item: any){
147     console.log(item);
148     this.navExtras.setExtras(item);
149     this.router.navigate(['/comida2']);
150   }
151 }
```

Código fuente 5.29.: Métodos usados en dieta.page.ts

El método que se utiliza para cargar las comidas del paciente, llamado *getFoodsByPatientAndDate*, se encuentra en el servicio *FoodService*. Toma dos parámetros: ´id´, que representa el identificador único del paciente, y ´date´, que es un objeto 'DateRangeModel' que contiene la fecha de inicio y la fecha de fin del rango. El método realiza una solicitud HTTP POST al backend con url ´$environment.api/foods/findByPatient/$id´ para buscar las comidas del paciente en el backend. El objeto date se envía como parte del cuerpo de la solicitud. El servidor procesará la solicitud y devolverá un array de objetos 'FoodModel' que contienen las comidas. El método devuelve un objeto Observable que emite la respuesta del servidor, que en este caso es un array de objetos 'FoodModel'. Esto permite suscribirse al resultado y recibir las comidas cuando la respuesta del servidor esté disponible.

Es importante destacar que este método utiliza http.post para realizar una solicitud POST en lugar de GET, ya que se está enviando un objeto en el cuerpo de la solicitud

(date). Esto indica que la solicitud está enviando información adicional al servidor para filtrar las comidas por fecha. [21]

```
getFoodsByPatientAndDate(id: string, date: DateRangeModel): Observable<FoodModel[]> {
    return this.http.post<FoodModel[]>(`${environment.api}/foods/findByPatient/${id}`, date);
  }
```

Código fuente 5.30.: getFoodsByPatientAndDate()

En el archivo HTML, se diseña la interfaz de usuario de la página de ejercicios. El encabezado contiene un segmento que muestra las fechas de los diferentes días y botones de navegación para moverse entre ellos. En el contenido, se utiliza la directiva *ngIf para mostrar el contenido de las comidas del día seleccionado y *ngFor para iterar sobre los arrays de comidas (desday2, comday2, etc.) y mostrar un botón para cada comida asignada al paciente en ese día. Se utiliza otra directiva *ngIf para comprobar que haya alguna comida asignado a ese día y si no es así, se muestra un texto indicandolo.

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-button *ngIf="showBackButton" (click)="moveSegment(-1)">
        <ion-icon name="chevron-back"></ion-icon>
      </ion-button>
    </ion-buttons>
    <ion-segment [(ngModel)]="selectedSegment">
      <ion-segment-button id="button1" value="button1" size="large" [hidden]="selectedSegment !==
      ↪  'button1'">{{ dia1 | date:'EEEE dd MMMM' }}</ion-segment-button>
      <ion-segment-button id="button2" value="button2" size="large" [hidden]="selectedSegment !==
      ↪  'button2'">{{ dia2 | date:'EEEE dd MMMM' }}</ion-segment-button>
      <ion-segment-button id="button3" value="button3" size="large" [hidden]="selectedSegment !==
      ↪  'button3'">{{ dia3 | date:'EEEE dd MMMM' }}</ion-segment-button>
      <ion-segment-button id="button4" value="button4" size="large" [hidden]="selectedSegment !==
      ↪  'button4'">{{ dia4 | date:'EEEE dd MMMM' }}</ion-segment-button>
      <ion-segment-button id="button5" value="button5" size="large" [hidden]="selectedSegment !==
      ↪  'button5'">{{ dia5 | date:'EEEE dd MMMM' }}</ion-segment-button>
      <ion-segment-button id="button6" value="button6" size="large" [hidden]="selectedSegment !==
      ↪  'button6'">{{ dia6 | date:'EEEE dd MMMM' }}</ion-segment-button>
      <ion-segment-button id="button7" value="button7" size="large" [hidden]="selectedSegment !==
      ↪  'button7'">{{ endDate | date:'EEEE dd MMMM' }}</ion-segment-button>
    </ion-segment>

    <ion-buttons slot="end">
      <ion-button *ngIf="showForwardButton && selectedSegment !== null" (click)="moveSegment(1)">
        <ion-icon name="chevron-forward"></ion-icon>
      </ion-button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>


<ion-content>
```

```
25    <br><br>
26    <!--Botón back-->
27    <ion-button fill = "clear" (click)="gotoComida()" color = "secondary">
28      <ion-icon slot="icon-only" name="chevron-back-outline"> </ion-icon>
29      Atrás
30    </ion-button>
31    <div *ngIf="selectedSegment == 'button1'">
32      <!--Día 1-->
33      <ion-card>
34        <ion-card-header>
35          <ion-card-subtitle>
36            <ion-label>{{dia1 | date:'EEEE'}}</ion-label>
37          </ion-card-subtitle>
38        </ion-card-header>
39        <ion-card-content>
40          <!-- Condición para comprobar si hay componentes en cada lista -->
41          <ng-container *ngIf="desday1.length > 0 || comday1.length > 0 || merday1.length > 0 ||
          ↪  cenday1.length > 0; else noComidas">
42            <ion-list>
43              <ion-list-header [ngClass]="{'ion-list-header-gray': desday1.length == 0}">
44                <ion-img class="imagen" src="../../../assets/icon/breakfast.png"></ion-img>
45                <ion-label class="comida">Desayuno</ion-label>
46              </ion-list-header><br>
47              <ion-item *ngFor="let item of desday1">
48                <ion-button class="boton" size="small" (click)="gotoComida2(item)">
49                  {{item.title}}
50                </ion-button>
51              </ion-item>
52            </ion-list>
53            <ion-list>
54              <ion-list-header [ngClass]="{'ion-list-header-gray': comday1.length == 0}">
55                <ion-img class="imagen" src="../../../assets/icon/lunch.png"></ion-img>
56                <ion-label class="comida">Comida</ion-label>
57              </ion-list-header><br>
58              <ion-item *ngFor="let item of comday1">
59                <ion-button class="boton" size="small" (click)="gotoComida2(item)">
60                  {{item.title}}
61                </ion-button>
62              </ion-item>
63            </ion-list>
64            <ion-list>
65              <ion-list-header [ngClass]="{'ion-list-header-gray': merday1.length == 0}">
66                <ion-img class="imagen" src="../../../assets/icon/lunch.png"></ion-img>
67                <ion-label class="comida">Merienda</ion-label>
68              </ion-list-header ><br>
69              <ion-item *ngFor="let item of merday1">
70                <ion-button class="boton" size="small" (click)="gotoComida2(item)">
71                  {{item.title}}
72                </ion-button>
73              </ion-item>
74            </ion-list>
75            <ion-list>
76              <ion-list-header [ngClass]="{'ion-list-header-gray': cenday1.length == 0}">
77                <ion-img class="imagen" src="../../../assets/icon/dinner.png"></ion-img>
78                <ion-label class="comida">Cena</ion-label>
```

```
79              </ion-list-header><br>
80              <ion-item *ngFor="let item of cenday1">
81                <ion-button class="boton" size="small" (click)="gotoComida2(item)">
82                  {{item.title}}
83                </ion-button>
84              </ion-item>
85            </ion-list>
86          </ng-container>
87      <!-- Mensaje a mostrar cuando no hay componentes -->
88      <ng-template #noComidas>
89        <div style="font-weight:bold; text-align:center">
90              <br><br><br><br><br><br>Aún no se ha subido ningún ejercicio<br><br><br><br><br><br>
91        </div>
92      </ng-template>
93        </ion-card-content>
94      </ion-card>
95    </div>
```

Código fuente 5.31.: dieta.page.html

En el archivo SCSS, se aplican estilos a los elementos de la página. Los estilos más destacables son los del encabezado del *ion-list*. *ion-list-header* se aplica a los encabezados de las listas que tienen comidas disponibles para un día específico. El estilo define un fondo con un gradiente amarillo una sombra sutil y un radio de borde redondeado. *.ion-list-header-gray* se aplica a los encabezados de las listas que no tienen comidas disponibles para una comida específica. El estilo define un fondo con un gradiente de gris que da una sensación de deshabilitado, esto indica visualmente que no hay comidas asignadas para ese día y ayuda a distinguirlo.

```
1   ion-list-header {
2     background: linear-gradient(to bottom right, #f3f2a8, #f5f109);
3     box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.3);
4     color: black;
5     border-radius: 30px;
6     text-align: center;
7     display: block;
8     margin: 0 auto;
9   }

10  .ion-list-header-gray {
11    background: linear-gradient(to bottom right, #e1e1e1, #a8a8a8);
12    box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.3);
13    color: black;
14    border-radius: 30px;
15  }
```

Código fuente 5.32.: Estilos encabezados de ion-list dieta.page.scss

La figura 5.7 muestra la pantalla con los ejercicios semanales donde se puede apreciar el resultado de los estilos y el HTML aplicados (el scrollbar no aparece en los dispositivos móviles).

Figura 5.7.: Pantalla de Dieta Semanal

5.7. Página de Comida

La página de comida se utiliza para mostrar los detalles de una comida específica y sus contenidos multimedia. Al inicio del archivo 'comida2.page.ts', se importan los módulos y servicios necesarios y se declaran las variables necesarias. Una variable *comida* de tipo *FoodModel* que almacena los datos de la comida seleccionada. Luego, se declara una variable *pdfUrl* para almacenar la URL segura del archivo adjunto (PDF) del ejercicio.
En el método *ngOnInit()*, que se ejecuta al inicializar la página, se obtienen los datos del ejercicio seleccionado utilizando el servicio *DataService* y el método *getExtras()* [10]. Estos datos se almacenan en la variable *comida*. A continuación, se verifica si la comida tiene un archivo adjunto. Si es así, se utiliza el servicio *FoodService* y el método *getAttachmentbyId()* para obtener el archivo adjunto. La respuesta se asigna a la variable *attachment*. Después de obtener el archivo adjunto, se genera una URL segura utilizando el servicio *DomSanitizer* y el método *bypassSecurityTrustResourceUrl*(). [1] Esta URL se almacena en la variable *pdfUrl* para mostrar el archivo adjunto (PDF) en la página.

```
1   ngOnInit() {
2     this.comida = this.navExtras.getExtras();
3     if (this.comida.attachment){
4       this.foodservice.getAttachmentbyId(this.comida.attachment).subscribe(
5         res => {
6           this.attachment = res;
7           const url  = `${environment.api}/files/attachment/${encodeURI(this.attachment.filename)}`;
8           this.pdfUrl = this.sanitizer.bypassSecurityTrustResourceUrl(url);
9         },
10        err => {
11          console.log(err);
12        }
13      );
14    }
15  }
```

Código fuente 5.33.: Método ngOnInit() de comida2.page.ts

La función *getAttachmentbyId*(idAttachment: string) del *FoodService* se utiliza para obtener los detalles de un archivo adjunto específico a partir de su ID de adjunto. Toma el ID del adjunto como argumento, representado por *idAttachment*. Utiliza el servicio *HttpClient* para realizar una solicitud GET al servidor para obtener los detalles del adjunto.[3] [21] La URL de la solicitud GET se construye utilizando el valor de *environment.api* (que representa la URL base de la API) concatenada con /attachments/findOne/ y el ID del adjunto. El tipo de respuesta esperado se especifica

como *AttachmentModel*. La función devuelve un Observable que emitirá la respuesta del servidor una vez que se complete la solicitud.

```
1  getAttachmentbyId(idAttachment: string){
2    return this.http.get<AttachmentModel>(`${environment.api}/attachments/findOne/${idAttachment}`);
3  }
```

Código fuente 5.34.: getAttachmentbyId()

A continuación se van a explicar otros métodos definidos en el archivo 'comida.page.ts'

- *getSafeUrl*(video: string): Esta función se utiliza para obtener una URL segura para incrustar un video en la página. Toma una URL de video como argumento y devuelve una URL segura que se puede utilizar en un elemento <iframe> del HTML para mostrar el video. Primero, se recorta y se almacena la URL proporcionada en la variable url, eliminando cualquier espacio en blanco alrededor. Luego, se verifica si la URL incluye «youtube.com» o «youtu.be». Si es así, se llama a la función getVideoId(url) para extraer el ID del video. A continuación, se construye una URL segura utilizando el ID del video extraído y el formato específico de la URL de YouTube para videos incrustados. Esta URL segura se devuelve. Si la URL no es de YouTube, se utiliza el servicio *DomSanitizer* para obtener una URL segura utilizando el método *bypassSecurityTrustResourceUrl*(url). [1]
- *getVideoId*(url: string): Esta función se utiliza en la función *getSafeUrl*(video: string) para extraer el ID del video de la URL proporcionada. La función toma una URL de video como argumento y devuelve el ID del video como una cadena. Primero, se inicializa la variable *videoId* como una cadena vacía. Luego, se verifica si la URL incluye «youtube.com». Si es así, se divide la URL utilizando el delimitador «v=» y se toma la segunda parte dividida (url.split('v=')[1]), que corresponde al ID del video. Si la URL incluye «youtu.be», se divide la URL utilizando el delimitador «.be/» y se toma la segunda parte dividida (url.split('.be/')[1]), que también corresponde al ID del video. Si el ID del video contiene caracteres adicionales, como un parámetro de consulta («&»), se divide nuevamente para eliminar esos caracteres (videoId.split('&')[0]). Finalmente, se devuelve el ID del video como una cadena.

Estas dos funciones trabajan juntas para obtener una URL segura y válida para incrustar videos de YouTube en la página de detalles de comida.

```
1  getSafeUrl(video: string) {
2      let url: string = video.trim();
3      if (url.includes('youtube.com') || url.includes('youtu.be')) {
4        let videoId: string = this.getVideoId(url);
5        return 'https://www.youtube.com/embed/' + videoId;
```

```
    } else {
      return this.sanitizer.bypassSecurityTrustResourceUrl(url);
    }
  }

  getVideoId(url: string): string {
    let videoId: string = '';
    if (url.includes('youtube.com')) {
      videoId = url.split('v=')[1];
    } else if (url.includes('youtu.be')) {
      videoId = url.split('.be/')[1];
    }
    if (videoId.includes('&')) {
      videoId = videoId.split('&')[0];
    }
    return videoId;
  }
```

Código fuente 5.35.: getSafeUrl() y getVideoId()

También se define el método *gotoDieta*() que se utiliza para navegar hacia la página anterior, que en este caso sería la página de la dieta semanal.

El archivo HTML representa la estructura de una página en Ionic que muestra los detalles de una comida específica. La página comienza con un botón de retroceso que permite al usuario volver a la página anterior. Luego, se muestra un contenedor de tarjeta que contiene el título de la comida en su cabecera. El contenido de la tarjeta incluye la descripción de la comida, los comentarios (si los hay), la lista de ingredientes de la receta, un archivo PDF con la receta (si está disponible), los enlaces relacionados con la receta y los videos asociados (si existen).

Si se proporciona un archivo PDF, se muestra utilizando un *iframe* que carga y muestra el contenido del archivo. Los enlaces relacionados se muestran como elementos de ancla, lo que permite al usuario hacer clic en ellos para abrir enlaces externos en una nueva pestaña o ventana del navegador. [22]

Finalmente, los videos asociados se muestran utilizando *iframes*. Cada video se carga y muestra dentro de un iframe, lo que permite al paciente reproducir y ver los videos relacionados con la comida. [22]

```
<ion-content>
  <br><ion-button fill = "clear" (click)="gotoDieta()" color = "secondary">
    <ion-icon slot="icon-only" name="chevron-back-outline"></ion-icon>
    ATRÁS
  </ion-button>
  <ion-card>
    <ion-card-header>
      <ion-card-title style="text-align: center">{{comida.title}}</ion-card-title>
    </ion-card-header>
```

```
10      <ion-card-content>
11        <br>
12        <ion-text class="dish">{{comida.dish}}</ion-text>
13        <br>
14        <ion-text class="texto" *ngIf=comida.description><strong class="titulo">
          ↪ Descripción:</strong>{{comida.description}}<br></ion-text>
15        <ion-text class="texto" *ngIf=comida.comments><strong class="titulo">
          ↪ Comentarios:</strong>{{comida.comments}}<br></ion-text>
16        <ion-text class="texto" *ngIf="comida.ingredients.length != 0"><strong class="titulo">
          ↪ Ingredientes: </strong>
17          <p *ngFor="let ing of comida.ingredients">{{ing.name}}
18            <span *ngIf="ing.quantity != 0">, {{ing.quantity}} {{ing.unit}}</span>
19          </p>
20          <br>
21        </ion-text>
22        <ion-text [innerHTML]="pdfUrl" *ngIf="pdfUrl">
23          <iframe [src]="pdfUrl" width="100%" height="250px"></iframe>
24          <br>
25        </ion-text>

26        <ion-text class="texto" *ngIf="comida.links.length != 0">
27          <strong class="titulo"> Links de recetas:</strong>
28          <a *ngFor="let link of comida.links" target="_blank" href="{{link}}">
29            {{link}}
30            <br><br>
31          </a>
32        </ion-text>

33        <div *ngIf="comida.videos.length != 0">
34          <div *ngFor="let video of comida.videos">
35            <iframe width="100%" height="200" [src]="getSafeUrl(video) | safe" frameborder="0"
              ↪ allow="autoplay; encrypted-media" allowfullscreen></iframe>
36            <br>
37          </div>
38        </div>
39      </ion-card-content>
40    </ion-card>
41  </ion-content>
```

Código fuente 5.36.: comida2.page.html

La figura 5.8 muestra la pantalla con los contenido relacionados con una comida donde se puede apreciar el resultado de los estilos aplicados en el archivo scss y el HTML explicado (el scrollbar no aparece en los dispositivos móviles).

Figura 5.8.: Detalles de la comida

5.8. Página de Lista de la compra

La página de la lista de la compra es una vista que muestra los ingredientes necesarios para cada día de la semana al paciente.
El archivo 'listacompra.page.ts' se encarga de la lógica de la página. Se importan los módulos y servicios necesarios para el funcionamiento de la página y contiene propiedades y métodos para manejar los ingredientes y las interacciones en la página. En el método *ngOnInit*(), se hace uso del servicio de autenticación importado para obtener la información del paciente logueado y carga los ingredientes necesarios llamando al método *loadIngredients*(). El método *loadIngredients*() recupera los ingredientes necesarios para cada día de la semana utilizando el servicio *FoodsService*. Luego, clasifica los ingredientes en los arrays correspondientes a cada día de la semana. El método

check(idFood: string, nameIngredient: string) maneja el evento de hacer clic en la casilla de verificación de un ingrediente para marcarlo como adquirido y llama al método *checkIngredient*() del servicio *FoodsService* para marcar el ingrediente como comprado.

```
ngOnInit() {
   this.authService.user$.subscribe(
     res => {
       this.patient = res;
       this.loadIngredients();
     },
     err => {
       console.log(err);
       this.snackerService.showFailed("No se ha encontrado al paciente");
     }
   );
 }

 loadIngredients (): void {
   this.loaderService.isLoading.next(true);
   this.foodsService.findIngredients(this.patient!._id, this.date)
   .pipe(finalize(() => this.loaderService.isLoading.next(false)))
   .subscribe(
     res => {
       res.forEach(ingredient => {
         var d = new Date(ingredient.date.toString());
         //Incluyo los ingredientes en el array correspondiente a su día
         if (d.toDateString() == this.startDate.toDateString()) {
           this.Ingday1.push(ingredient);
         }

         else if (d.toDateString() == this.dia2.toDateString()) {
           this.Ingday2.push(ingredient);
         }

         else if (d.toDateString() == this.dia3.toDateString()) {
           this.Ingday3.push(ingredient);
         }

         else if (d.toDateString() == this.dia4.toDateString()) {
           this.Ingday4.push(ingredient);
         }

         else if (d.toDateString() == this.dia5.toDateString()) {
           this.Ingday5.push(ingredient);
         }

         else if (d.toDateString() == this.dia6.toDateString()) {
           this.Ingday6.push(ingredient);
         }

         else if (d.toDateString() == this.endDate.toDateString()) {
           this.Ingday7.push(ingredient);

         }
```

```
43          });
44        },
45        err => {
46          console.log(err.error.message);
47        }
48      );
49    }

50    check(idFood: string, nameIngredient: string){
51      this.foodsService.checkIngredient(idFood,nameIngredient);
52    }
```

Código fuente 5.37.: Métodos de listacompra.page.ts

El servicio *FoodService* contiene dos métodos relevantes que se han usado en la página de la lista de la compra: *findIngredients*() y *checkIngredient*(). A continuación, se explica cada uno de ellos:

1. Método *findIngredients*(idPatient: string, date: DateRangeModel) Este método se utiliza para obtener los ingredientes necesarios para un paciente específico en un rango de fechas determinado. Recibe como parámetros el ID del paciente (*idPatient*) y un objeto *DateRangeModel* que contiene la fecha de inicio y la fecha de fin. Realiza una solicitud HTTP POST al servidor utilizando la URL $environment.api/foods/findIngredients/$idPatient y envía el objeto *date* como cuerpo de la solicitud. [21] El servidor procesa la solicitud y devuelve un array de objetos IngredientModel que representan los ingredientes encontrados. El método retorna un Observable que emite el array de ingredientes.
2. Método *checkIngredient*(idFood: string, nameIngredient: string) Este método se utiliza para marcar un ingrediente como comprado. Recibe como parámetros el ID del alimento (*idFood*) y el nombre del ingrediente (*nameIngredient*). Realiza una solicitud HTTP GET al servidor utilizando la URL $environment.api/foods/checkIngredient/$idFood/$nameIngredient. [21] El servidor procesa la solicitud y realiza la acción correspondiente para marcar el ingrediente como verificado.

```
1   findIngredients(idPatient: string, date: DateRangeModel) {
2     return this.http.post<IngredientModel[]>(`${environment.api}/foods/findIngredients/${idPatient}`,
      ↪ date);
3   }

4   checkIngredient(idFood: string, nameIngredient: string) {
5
      ↪ this.http.get<FoodModel>(`${environment.api}/foods/checkIngredient/${idFood}/${nameIngredient}`).subscribe(
6       res => console.log(""),
7       err => console.log("")
```

```
    );
  }
```

Código fuente 5.38.: findIngredients() y checkIngredient()

En el archivo 'listacompra.page.ts' también se define el método *gotoComida()* que redirige al usuario a la página de comida al hacer clic en el botón «ATRÁS».

La estructura HTML de la página incluye un encabezado y una serie de tarjetas que representan los ingredientes necesarios para cada día de la semana. El encabezado contiene un botón de retroceso que permite al usuario regresar a la página anterior llamando al método *gotoComida*(). Se utiliza la directiva *ngFor para iterar sobre los ingredientes de cada día y generar las tarjetas correspondientes. Las tarjetas muestran el nombre del ingrediente, la cantidad y la unidad en caso de que haya una cantidad definida. Cada tarjeta incluye una casilla de verificación que refleja si el ingrediente se ha adquirido o no. La etiqueta <ion-label> muestra el nombre del día de la semana en formato legible para cada tarjeta permitiendo saber qué día se usará.

```
<ion-button fill = "clear" (click)="gotoComida()" color = "secondary">
  <ion-icon slot="icon-only" name="chevron-back-outline"></ion-icon>
  ATRÁS
</ion-button>

<ion-card *ngFor="let item of Ingday1" class="ion-margin">
  <ion-card-header>
    <ion-card-subtitle>
      <ion-label class="ingrediente">{{ item.name }}</ion-label>
    </ion-card-subtitle>
  </ion-card-header>
  <ion-card-content>
    <ion-text *ngIf="item.quantity!=0"><b>{{item.quantity}}  {{item.unit}}</b></ion-text>
    <ion-checkbox [checked]="item.isChecked" class="alignme" (click)="check(item.food, item.name)">
    ↪  </ion-checkbox>
      <ion-label  style="font-size: x-small;" class="alignme1">
       <br> {{item.date | date:'EEEE'}}
      </ion-label>
  </ion-card-content>
</ion-card>

<!-- Ingredientes del resto de días-->
```

Código fuente 5.39.: listacompra.page.html

Se aplican estilos personalizados para el encabezado, las tarjetas y los elementos de la lista de la compra que se definen en el archivo 'listacompra.page.scss'. La figura 5.9 muestra la pantalla de comida donde se puede apreciar el resultado final de la página (el scrollbar no aparece en los dispositivos móviles):

Figura 5.9.: Pantalla de lista de la compra

5.9. Página de Ejercicios Semanales

La página de «Ejercicio» (EjercicioPage) es un componente Angular que muestra al paciente los ejercicios pautados en diferentes días.
En el archivo TypeScript 'ejercicio.page.ts', se define la propiedad *selectedSegment*, que almacena el valor del segmento seleccionado para moverse entre los distintos días, y *segmentHidden*, un objeto que indica qué segmentos están ocultos. El método *moveSegment*(val: number) se utiliza para cambiar entre segmentos y mostrar los ejercicios correspondientes al día del segmento seleccionado. Luego se definen varias propiedades relacionadas con las fechas, como *startDate*, *endDate* y *dia1* o *dia2*, que representan las fechas de los diferentes días que se mostrarán en la página.

```
selectedSegment: string = 'button2';
 segmentHidden = {
   button1: true,
   button2: false,
   button3: true,
   button4: true,
   button5: true,
   button6: true,
   button7: true
 };

 public showBackButton: boolean = true;
 public showForwardButton: boolean = true;

 moveSegment(val: number) {
   const segments = document.querySelectorAll('ion-segment-button');
   const selectedIndex = Array.from(segments).findIndex(
     (segment: any) => segment.value === this.selectedSegment
   );
   const maxIndex = segments.length - 1;
   const newIndex = selectedIndex + val;
   if (newIndex > 0 && newIndex < maxIndex) {
     this.selectedSegment = segments[newIndex].value;
     segments[newIndex].scrollIntoView({
       behavior: 'smooth',
       block: 'center'
     });
     this.showBackButton = true;
     this.showForwardButton = newIndex < maxIndex
   } else if (newIndex == 0) {
     this.selectedSegment = segments[0].value;
     segments[0].scrollIntoView({
       behavior: 'smooth',
       block: 'center'
     });
     this.showBackButton = false;
     this.showForwardButton = true;
   } else if (newIndex == maxIndex) {
```

```
37        this.selectedSegment = segments[maxIndex].value;
38        segments[maxIndex].scrollIntoView({
39          behavior: 'smooth',
40          block: 'center'
41        });
42        this.showBackButton = true;
43        this.showForwardButton = false;
44      }
45    }
46    patient: PatientModel | null = null;
47    id : PatientModel | null = null;

48    addDays = (date: Date, days: number): Date => {
49      let result = new Date(date);
50      result.setDate(result.getDate() + days);
51      return result;
52    };
53    subtractDays = (date: Date, days: number): Date => {
54      let result = new Date(date);
55      result.setDate(result.getDate() - days);
56      return result;
57    };


58    dia2: Date = new Date(); //Definimos las fechas de de cada día que se mostrará
59    startDate: Date = this.subtractDays(this.dia2,2);
60    dia1: Date = this.subtractDays(this.dia2,1);
61    dia3: Date = this.addDays(this.dia2, 1);
62    dia4: Date = this.addDays(this.dia2, 2);
63    dia5: Date = this.addDays(this.dia2, 3);
64    dia6: Date = this.addDays(this.dia2, 4);
65    endDate: Date = this.addDays(this.dia2, 5);


66    date: DateRangeModel = {
67      startDate: this.startDate,
68      endDate :this.endDate,
69    };

70    moveday1: MoveModel[] = []; //Arrays que contendrán los ejercicios de cada dia
71    moveday2: MoveModel[] = [];
72    moveday3: MoveModel[] = [];
73    moveday4: MoveModel[] = [];
74    moveday5: MoveModel[] = [];
75    moveday6: MoveModel[] = [];
76    moveday7: MoveModel[] = [];
```

Código fuente 5.40.: Definición de propiedades y variables en ejercicio.page.ts

En el método *ngOnInit*(), se obtiene el usuario actual haciendo uso del *authService*. Luego, se llama al método *loadMoves*() para cargar los ejercicios del paciente. El método *loadMoves*() utiliza el servicio *movesService* para obtener los ejercicios según el paciente y fecha (los ejercicios serán objetos que seguirán el modelo definido en *move.model.ts*) .

Se suscribe a la respuesta y, para cada ejercicio, se comprueba la fecha y se asigna al array correspondiente a los ejercicios de cada día (*moveday1*, *moveday2*, etc.).
También se define método *gotoInicio*() que se utiliza para navegar al usuario a la página de inicio cuando se hace clic en el botón «ATRÁS» como en el resto de páginas donde se usa.
El método *gotoEjercicio2*(item: any) se utiliza para navegar al usuario a la página de detalles del ejercicio cuando se hace clic en un botón de ejercicio. El objeto 'item' se pasa como parámetro utilizando el servicio *navExtras* y contendrá el objeto con el ejercicio del cuál se desean consultar los detalles. [10]

```
loadMoves (): void {
    this.loaderService.isLoading.next(true);
    this.movesService.getMovesByPatientAndDate(this.patient!._id, this.date)
    .pipe(finalize(() => this.loaderService.isLoading.next(false)))
    .subscribe(
      res => {
        res.forEach(move => {
          var d = new Date(move.date.toString());
          //Incluyo los platos en el array correspondiente a su día y comida
          if (d.toDateString() == this.dia1.toDateString()) {
              this.moveday1.push(move);
          }
          else if (d.toDateString() == this.dia2.toDateString()) {
            this.moveday2.push(move);
          }
          else if (d.toDateString() == this.dia3.toDateString()) {
            this.moveday3.push(move);
          }
          else if (d.toDateString() == this.dia4.toDateString()) {
            this.moveday4.push(move);
          }
          else if (d.toDateString() == this.dia5.toDateString()) {
            this.moveday5.push(move);
          }
          else if (d.toDateString() == this.dia6.toDateString()) {
            this.moveday6.push(move);
          }
          else if (d.toDateString() == this.endDate.toDateString()) {
            this.moveday7.push(move);
          }
        });


      },
      err => {
        console.log(err.error.message);
      }
    );
  }

  gotoEjercicio2(item: any){
```

```
39      console.log(item);
40      this.navExtras.setExtras(item);
41      this.router.navigate(['/ejercicio2']);
42    }
43  }
```

Código fuente 5.41.: Métodos usados en ejercicio.page.ts

El método que se utiliza para cargar las comidas del paciente, llamado *getMovesByPatientAndDate*, se encuentra en el servicio *MovesServiceService*. Toma dos parámetros: 'id', que representa el identificador único del paciente, y 'date', que es un objeto *DateRangeModel* que contiene la fecha de inicio y la fecha de fin del rango. El método realiza una solicitud HTTP POST al backend con url ´$environment.api/moves/findByPatient/$id´ para buscar los ejercicios del paciente en el backend. [21] El objeto date se envía como parte del cuerpo de la solicitud. El servidor procesará la solicitud y devolverá un array de objetos *MoveModel* que contienen los ejercicos. El método devuelve un objeto Observable que emite la respuesta del servidor, que en este caso es un array de objetos *MoveModel*. Esto permite suscribirse al resultado y recibir los ejercicios cuando la respuesta del servidor esté disponible.
Es importante destacar que este método utiliza http.post para realizar una solicitud POST en lugar de GET, ya que se está enviando un objeto en el cuerpo de la solicitud (date). Esto indica que la solicitud está enviando información adicional al servidor para filtrar los movimientos por fecha.[21]

```
1   getMovesByPatientAndDate (id: string, date: DateRangeModel): Observable<MoveModel[]> {
2      return this.http.post<MoveModel[]>(`${environment.api}/moves/findByPatient/${id}`,  date);
3    }
```

Código fuente 5.42.: getMovesByPatientAndDate

En el archivo HTML, se diseña la interfaz de usuario de la página de ejercicios. El encabezado contiene un segmento que muestra las fechas de los diferentes días y botones de navegación para moverse entre ellos. En el contenido, se utiliza la directiva *ngIf para mostrar el contenido de los ejercicios del día seleccionado y *ngFor para iterar sobre los arrays de ejercicios (*moveday1*, *moveday2*, etc.) y mostrar un botón para cada ejercicio asignado al paciente en ese día. Se utiliza otra directiva *ngIf para comprobar que haya algún ejercicio asignado a ese día y si no es así, se muestra un texto indicandolo.

```
1   <ion-header>
2       <ion-toolbar>
3       <ion-buttons slot="start">
4         <ion-button *ngIf="showBackButton" (click)="moveSegment(-1)">
5           <ion-icon name="chevron-back"></ion-icon>
6         </ion-button>
```

```
7       </ion-buttons>
8       <ion-segment [(ngModel)]="selectedSegment">
9         <ion-segment-button id="button1" value="button1" [hidden]="selectedSegment !== 'button1'">{{ dia1 |
          ↪ date:'EEEE dd MMMM' }}</ion-segment-button>
10        <ion-segment-button id="button2" value="button2" [hidden]="selectedSegment !== 'button2'">{{ dia2 |
          ↪ date:'EEEE dd MMMM' }}</ion-segment-button>
11        <ion-segment-button id="button3" value="button3" [hidden]="selectedSegment !== 'button3'">{{ dia3 |
          ↪ date:'EEEE dd MMMM' }}</ion-segment-button>
12        <ion-segment-button id="button4" value="button4" [hidden]="selectedSegment !== 'button4'">{{ dia4 |
          ↪ date:'EEEE dd MMMM' }}</ion-segment-button>
13        <ion-segment-button id="button5" value="button5" [hidden]="selectedSegment !== 'button5'">{{ dia5 |
          ↪ date:'EEEE dd MMMM' }}</ion-segment-button>
14        <ion-segment-button id="button6" value="button6" [hidden]="selectedSegment !== 'button6'">{{ dia6 |
          ↪ date:'EEEE dd MMMM' }}</ion-segment-button>
15        <ion-segment-button id="button7" value="button7" [hidden]="selectedSegment !== 'button7'">{{
          ↪ endDate | date:'EEEE dd MMMM' }}</ion-segment-button>
16      </ion-segment>
17      <ion-buttons slot="end">
18        <ion-button *ngIf="showForwardButton && selectedSegment !== null" (click)="moveSegment(1)">
19          <ion-icon name="chevron-forward"></ion-icon>
20        </ion-button>
21      </ion-buttons>
22    </ion-toolbar>
23  </ion-header>

24  <ion-content>
25    <div *ngIf="selectedSegment == 'button1'">
26      <!--Día 1-->
27      <ion-card >
28        <ion-card-header>
29          <ion-card-subtitle>
30            <ion-label>{{dia1 | date:'EEEE'}}</ion-label>
31          </ion-card-subtitle>
32        </ion-card-header>

33        <ion-card-content>
34          <ion-list>
35            <ion-list-header [ngClass]="{'ion-list-header-gray': moveday1.length == 0}">
36              <ion-label class="ejercicios" >EJERCICIOS</ion-label>
37            </ion-list-header>
38            <br>
39            <ion-item *ngFor="let item of moveday1">
40              <ion-button class="boton" size="default" (click)="gotoEjercicio2(item)">
41                {{item.title}}
42              </ion-button>
43            </ion-item>
44            <div *ngIf="moveday1.length == 0" style="font-weight:bold; text-align:center">
45              <br><br><br><br><br>Aún no se ha subido ningún ejercicio<br><br><br><br><br><br>
46            </div>
47          </ion-list>
48        </ion-card-content>
49      </ion-card>
50    </div>
51    <!-- Contenido del resto de días -->
52  </ion-content>
```

Código fuente 5.43.: ejercicio.page.html

En el archivo SCSS, se aplican estilos a los elementos de la página. Los estilos más destacables son los del encabezado del *ion-list*. *ion-list-header* se aplica a los encabezados de las listas que tienen ejercicios disponibles para un día específico. El estilo define un fondo con un gradiente amarillo una sombra sutil y un radio de borde redondeado. *.ion-list-header-gray* se aplica a los encabezados de las listas que no tienen ejercicios disponibles para un día específico. El estilo define un fondo con un gradiente de gris que da una sensación de deshabilitado, esto indica visualmente que no hay ejercicios disponibles para ese día y ayuda a distinguirlo de los días con ejercicios.

```
1   ion-list-header {
2     background: linear-gradient(to bottom right, #f3f2a8, #f5f109);
3     box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.3);
4     color: black;
5     border-radius: 30px;
6     text-align: center;
7     display: block;
8     margin: 0 auto;
9   }

10  .ion-list-header-gray {
11    background: linear-gradient(to bottom right, #e1e1e1, #a8a8a8);
12    box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.3);
13    color: black;
14    border-radius: 30px;
15  }
```

Código fuente 5.44.: Estilos encabezados de ion-list ejercicio.page.scss

Las figuras 5.10 y 5.11 muestran la pantalla con los ejercicios semanales donde se puede apreciar el resultado de los estilos y el HTML aplicados.

Figura 5.10.: Ejercicos Semanales

Figura 5.11.: Ejercicos Semanales sin asignar

5.10. Página de Ejercicio

La siguiente página se utiliza para mostrar los detalles de un ejercicio específico y sus contenidos multimedia. Al inicio del archivo 'ejercicio2.page.ts', se importan los módulos y servicios necesarios y se declaran las variables necesarias. Una variable *move* de tipo *MoveModel* que almacena los datos del ejercicio seleccionado. Luego, se declara una variable *pdfUrl* para almacenar la URL segura del archivo adjunto (PDF) del ejercicio.

En el método *ngOnInit*(), que se ejecuta al inicializar la página, se obtienen los datos del ejercicio seleccionado utilizando el servicio *DataService* y el método *getExtras*().[10] Estos datos se almacenan en la variable *move*. A continuación, se verifica si el ejercicio tiene un archivo adjunto. Si es así, se utiliza el servicio *MovesServiceService* y el método *getAttachmentbyId*() para obtener el archivo adjunto. La respuesta se asigna a la variable *attachment*. Después de obtener el archivo adjunto, se genera una URL segura utilizando el servicio *DomSanitizer* y el método *bypassSecurityTrustResourceUrl*().[1] Esta URL se almacena en la variable pdfUrl para mostrar el archivo adjunto (PDF) en la página.

```
1   ngOnInit() {
2     this.move = this.navExtras.getExtras();
3     if (this.move.attachment){
```

```
4       this.moveservice.getAttachmentbyId(this.move.attachment).subscribe(
5         res => {
6           this.attachment = res;
7           const url  = `${environment.api}/files/attachment/${encodeURI(this.attachment.filename)}`;
8           this.pdfUrl = this.sanitizer.bypassSecurityTrustResourceUrl(url);
9         },
10        err => {
11          console.log(err);
12        }
13      );
14    }
15  }
```

Código fuente 5.45.: Método ngOnInit() de ejercicio2.page.ts

La función *getAttachmentbyId*(idAttachment: string) del *MoveService* se utiliza para obtener los detalles de un archivo adjunto específico a partir de su ID de adjunto. Toma el ID del adjunto como argumento, representado por *idAttachment*. Utiliza el servicio *HttpClient* para realizar una solicitud GET al servidor para obtener los detalles del adjunto. [3][21] La URL de la solicitud GET se construye utilizando el valor de *environment.api* (que representa la URL base de la API) concatenada con /attachments/findOne/ y el ID del adjunto. El tipo de respuesta esperado se especifica como *AttachmentModel*. La función devuelve un Observable que emitirá la respuesta del servidor una vez que se complete la solicitud.

```
1   getAttachmentbyId(idAttachment: string){
2     return this.http.get<AttachmentModel>(`${environment.api}/attachments/findOne/${idAttachment}`);
3   }
```

Código fuente 5.46.: getAttachmentbyId()

A continuación se van a explicar otros métodos definidos en el archivo 'ejercicio2.page.ts'

- *getSafeUrl*(video: string): Esta función se utiliza para obtener una URL segura para incrustar un video en la página. Toma una URL de video como argumento y devuelve una URL segura que se puede utilizar en un elemento <iframe> del HTML para mostrar el video. Primero, se recorta y se almacena la URL proporcionada en la variable url, eliminando cualquier espacio en blanco alrededor. Luego, se verifica si la URL incluye «youtube.com» o «youtu.be». Si es así, se llama a la función *getVideoId*(url) para extraer el ID del video. A continuación, se construye una URL segura utilizando el ID del video extraído y el formato específico de la URL de YouTube para videos incrustados. Esta URL segura se devuelve. Si la URL no es de YouTube, se utiliza el servicio *DomSanitizer* para obtener una URL segura utilizando el método *bypassSecurityTrustResourceUrl*(url). [1]

- *getVideoId*(url: string): Esta función se utiliza en la función *getSafeUrl*(video: string) para extraer el ID del video de la URL proporcionada. La función toma una URL de video como argumento y devuelve el ID del video como una cadena. Primero, se inicializa la variable videoId como una cadena vacía. Luego, se verifica si la URL incluye «youtube.com». Si es así, se divide la URL utilizando el delimitador «v=» y se toma la segunda parte dividida (url.split('v=')[1]), que corresponde al ID del video. Si la URL incluye «youtu.be», se divide la URL utilizando el delimitador «.be/» y se toma la segunda parte dividida (url.split('.be/')[1]), que también corresponde al ID del video. Si el ID del video contiene caracteres adicionales, como un parámetro de consulta («&»), se divide nuevamente para eliminar esos caracteres (videoId.split('&')[0]). Finalmente, se devuelve el ID del video como una cadena.

Estas dos funciones trabajan juntas para obtener una URL segura y válida para incrustar videos de YouTube en la página de detalles del ejercicio.

```
getSafeUrl(video: string) {
    let url: string = video.trim();
    if (url.includes('youtube.com') || url.includes('youtu.be')) {
      let videoId: string = this.getVideoId(url);
      return 'https://www.youtube.com/embed/' + videoId;
    } else {
      return this.sanitizer.bypassSecurityTrustResourceUrl(url);
    }
  }

  getVideoId(url: string): string {
    let videoId: string = '';
    if (url.includes('youtube.com')) {
      videoId = url.split('v=')[1];
    } else if (url.includes('youtu.be')) {
      videoId = url.split('.be/')[1];
    }
    if (videoId.includes('&')) {
      videoId = videoId.split('&')[0];
    }
    return videoId;
  }
```

Código fuente 5.47.: getSafeUrl() y getVideoId()

También se define el método *gotoEjercicio*() que se utiliza para navegar hacia la página anterior, que en este caso sería la lista de ejercicios.

El archivo HTML representa la estructura de una página en Ionic que muestra los detalles de un ejercicio específico. La página comienza con un botón de retroceso que permite al usuario volver a la página anterior. Luego, se muestra un contenedor de

tarjeta que contiene el título del ejercicio en su cabecera. El contenido de la tarjeta incluye la descripción del ejercicio, los comentarios (si los hay), un archivo PDF (si está disponible), los enlaces relacionados con el ejercicio y los videos asociados (si existen). Si se proporciona un archivo PDF, se muestra utilizando un *iframe* que carga y muestra el contenido del archivo. Los enlaces relacionados se muestran como elementos de ancla, lo que permite al usuario hacer clic en ellos para abrir enlaces externos en una nueva pestaña o ventana del navegador. [22]
Finalmente, los videos asociados se muestran utilizando *iframes*. Cada video se carga y muestra dentro de un iframe, lo que permite al paciente reproducir y ver los videos relacionados con el ejercicio. [22]

```
<ion-content>
  <br>
  <ion-button fill = "clear" (click)="gotoEjercicio()" color = "secondary">
    <ion-icon slot="icon-only" name="chevron-back-outline"></ion-icon>
    ATRÁS
  </ion-button>
  <ion-card>
    <ion-card-header>
      <ion-card-title style="text-align: center">{{move.title}}</ion-card-title>
    </ion-card-header>

    <ion-card-content>
      <ion-text class="texto" *ngIf=move.description><strong class="titulo"> Descripción:</strong>
      ↪ {{move.description}}<br></ion-text>
      <ion-text class="texto" *ngIf=move.comments><strong class="titulo"> Comentarios:</strong>
      ↪ {{move.comments}}<br> <br></ion-text>
      <ion-text [innerHTML]="pdfUrl" *ngIf="pdfUrl">
        <iframe [src]="pdfUrl" width="100%" height="250px"></iframe>
        <br>
      </ion-text>
      <ion-text class="texto" *ngIf="move.links.length != 0">
        <strong class="titulo"> Links:</strong>
        <a *ngFor="let link of move.links" target="_blank" href="{{link}}">
          {{link}}
          <br><br>
        </a>
      </ion-text>
      <div *ngIf="move.videos.length != 0">
        <div *ngFor="let video of move.videos">
          <iframe width="100%" height="200" [src]="getSafeUrl(video) | safe" frameborder="0"
          ↪ allow="autoplay; encrypted-media" allowfullscreen></iframe>
          <br>
        </div>
      </div>
    </ion-card-content>
  </ion-card>
</ion-content>
```

Código fuente 5.48.: ejercicio2.page.html

La figura 5.12 muestra la pantalla con los contenido relacionados con un ejercicio donde se puede apreciar el resultado de los estilos aplicados en el archivo scss y el HTML explicados.

Figura 5.12.: Detalles de ejercicio

CAPÍTULO 6

PRESUPUESTO

"La felicidad se alcanza cuando
lo que uno piensa, lo que uno dice y
lo que uno hace están en armonía."
— Mahatma Gandhi, *1869-1948*.

El presupuesto aproximado para llevar a cabo el proyecto Food & Move dependerá de varios factores, como la duración del proyecto y las tarifas del equipo de desarrollo. A continuación, se presenta una estimación considerando algunos roles clave en el proyecto:

- **Desarrollador de la aplicación móvil:** Se necesita un desarrollador especializado en el desarrollo de aplicaciones móviles para crear la app de Food & Move. Considerando un precio por hora de 30 euros y 90 horas de trabajo, el cálculo sería: 90 horas * 30 euros/hora = 2,700 euros.

- **Desarrollador web y backend:** Para el desarrollo del sitio web y la implementación del backend, se requiere un desarrollador web con experiencia en tecnologías como HTML, CSS, JavaScript y un lenguaje de programación para el backend (en este caso, Node.js). Considerando un precio por hora de 30 euros y 90 horas de trabajo, el cálculo sería: 90 horas * 30 euros/hora = 2,700 euros.

- **Director de proyecto:** El proyecto Food & Move necesita un director de proyecto que se encargue de la gestión general del proyecto, la coordinación del equipo y la supervisión del progreso. Tomemos un precio por hora de 40 euros y unas 30 horas de trabajo, el cálculo sería: 30 horas * 40 euros/hora = 1,200 euros.

- **Diseñador gráfico:** Para el aspecto visual de la web y la aplicación, se requiere un diseñador gráfico especializado en la creación de iconografía y diseño de interfaces de usuario atractivas.Suponiendo un precio por hora de 30 euros y unas 30 horas de trabajo, el cálculo sería: 30 horas * 30 euros/hora = 900 euros.

El total es de 2,700 euros (desarrollador de la aplicación móvil) + 2,700 euros (desarrollador web y backend) + 1,200 euros (director de proyecto) + 900 euros (diseñador gráfico) = 7,500 euros. Es importante tener en cuenta que estos costos son solo estimaciones aproximadas y pueden variar significativamente. Además, el presupuesto total del proyecto también deberá tener en cuenta otros gastos, como infraestructura tecnológica, servicios de alojamiento y mantenimiento continuo.

CAPÍTULO 7

Conclusiones

"Sólo podemos ver
una corta distancia por delante,
pero podemos ver
muchas cosas que hay que hacer"
— Alan Turing, *1912-1954*.

Durante el desarrollo de este Trabajo de Fin de Grado, he tenido la oportunidad de adquirir valiosos aprendizajes y enfrentar desafíos significativos, lo que ha contribuido a mi crecimiento personal y profesional. A lo largo del proyecto, me he sumergido en el fascinante mundo de la programación y he aprendido a desarrollar una aplicación de manera full stack, abarcando tanto el frontend como parte del backend. Estos conocimientos me han permitido comprender la complejidad y la importancia de cada componente en el proceso de creación de una aplicación.

Uno de los mayores aprendizajes ha sido la familiarización con diversos lenguajes de programación y la estructura de una app. Desde el inicio del proyecto, me enfrenté al reto de aprender lenguajes nuevos y explorar sus funcionalidades. Este proceso de aprendizaje me ha brindado una valiosa experiencia para enfrentar futuros proyectos.

Además de los aspectos técnicos, he tenido la oportunidad de experimentar el trabajo en equipo, colaborando con personas con distintos ritmos y prioridades. Esta experiencia ha sido fundamental para desarrollar habilidades de comunicación efectiva, adaptabilidad y flexibilidad, aprendiendo que la colaboración es esencial para alcanzar los objetivos planteados.

El camino no estuvo exento de desafíos, y uno de los más significativos fue el hecho de asumir la responsabilidad de aprender nuevos lenguajes de programación por

mi cuenta. Sin embargo, el deseo de superar estos obstáculos y lograr el éxito en el proyecto me impulsó a perseverar y a desarrollar la confianza en mis habilidades como programador.

La satisfacción personal que he experimentado a lo largo de este TFG ha sido una gran motivación. Saber que mi trabajo está dirigido a facilitar y mejorar la calidad de vida de personas que enfrentan problemas de salud mental mediante la promoción de hábitos saludables, es algo que me llena de orgullo y me impulsó a elegir y a continuar este proyecto.

Si bien la aplicación Food & Move ha alcanzado un nivel satisfactorio de funcionalidad y usabilidad, existen diversas oportunidades para continuar mejorando y ampliando su alcance. Una línea de trabajo futuro que se considera prioritaria y ya está en marcha es la optimización del espacio de almacenamiento en la aplicación.

Actualmente, la asignación de nuevas dietas a los pacientes cada semana ocupa espacio significativo en el almacenamiento de la aplicación. Para abordar este desafío, se propone implementar una funcionalidad donde el enfermero o profesional de la salud pueda asignar una dieta específica al paciente, y esta misma dieta se mantenga hasta que sea modificada. Al evitar la creación y almacenamiento repetido de dietas, se logrará una considerable reducción en el consumo de espacio y se mejorará la eficiencia de la aplicación.

APÉNDICE A

CÓDIGO FUENTE

A continuación, se proporcionan los enlaces a los repositorios de GitHub donde se encuentra el código completo del backend y del frontend de la aplicación Food & Move:

- Backend: https://github.com/alvarovaq/food-move-backend

  Descripción: Este repositorio contiene el código fuente completo del backend de la aplicación Food & Move. El backend está desarrollado utilizando tecnologías como Node.js y MongoDB para gestionar la lógica del servidor y la base de datos.

- Frontend: https://github.com/marcos-novoa/F-M-app

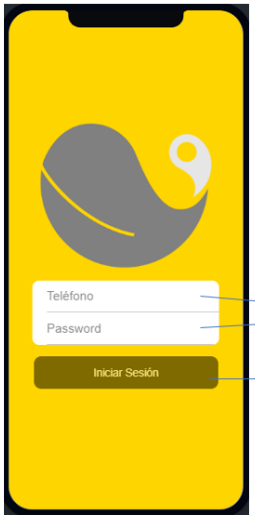
  Descripción: En este repositorio se encuentra el código fuente completo del frontend de la aplicación Food & Move. El frontend está desarrollado utilizando Ionic, Angular, HTML, CSS y otras tecnologías relacionadas para crear la interfaz de usuario interactiva y atractiva.

Food & Move es un proyecto en constante desarrollo, por lo que el código no es estático y puede haber diferencias con el código explicado a lo largo del trabajo.

APÉNDICE B

MANUAL DE USUARIO

Como parte de este proyecto, se ha elaborado un manual de usuario simplificado que brinda una guía clara y concisa sobre el uso de la aplicación, permitiendo a los usuarios entender fácilmente la funcionalidad de cada componente en cada página y saber cómo aprovechar al máximo todas sus características. El manual de usuario será de gran utilidad para los usuarios, ya que les permitirá explorar y utilizar la aplicación de manera eficiente y efectiva desde el inicio. A continuación, se presenta dicho manual de usuario para facilitar su comprensión y uso.



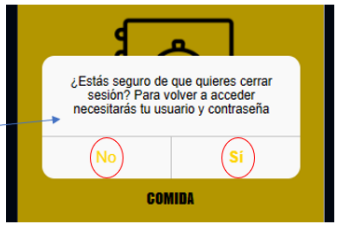
La pantalla de Log in será utilizada únicamente por el profesional sanitario

Cuadros para insertar los datos de Inicio de sesión del paciente

Botón de Inicio de Sesión

Botón de cierre de sesión

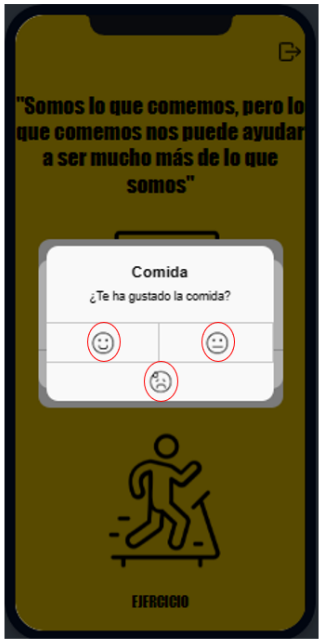
Se pedirá confirmar que se desea cerra la sesión

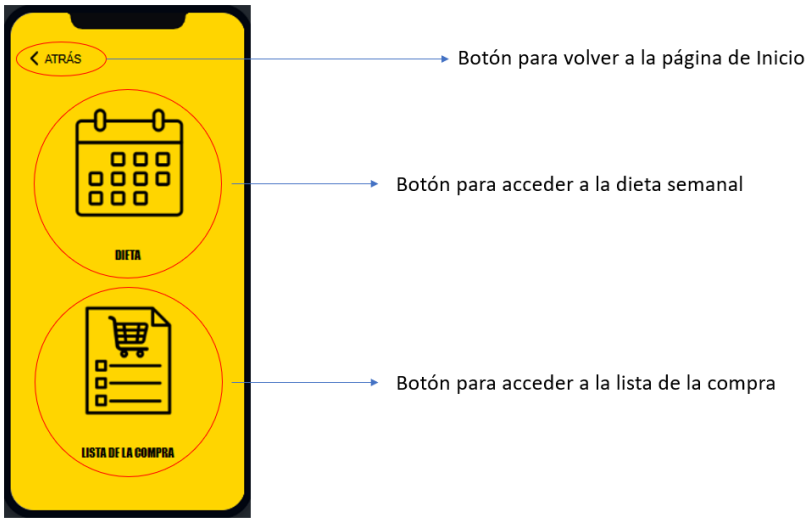


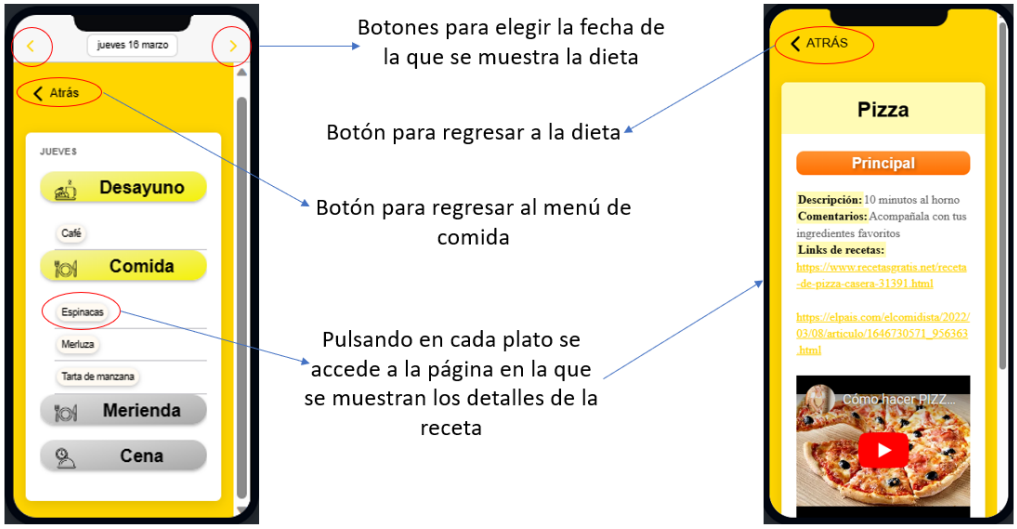
Botón para acceder a la información de las comidas

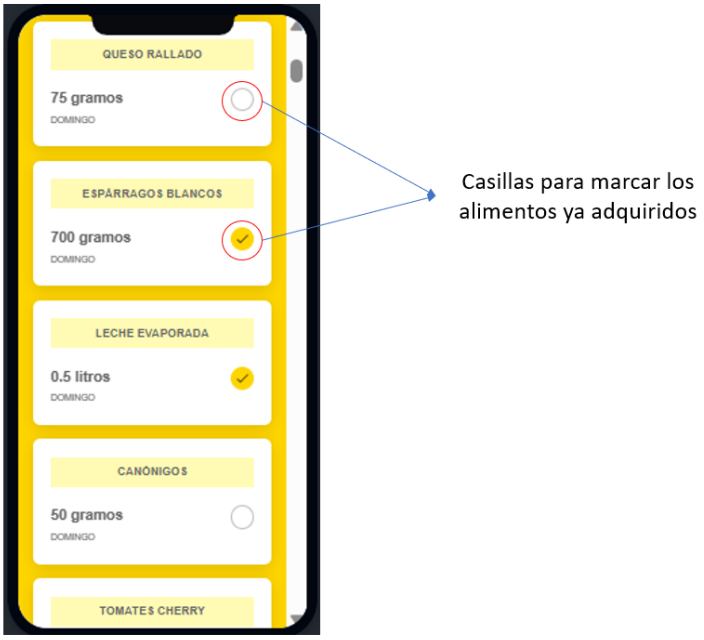
Botón para acceder a la información de los ejercicios

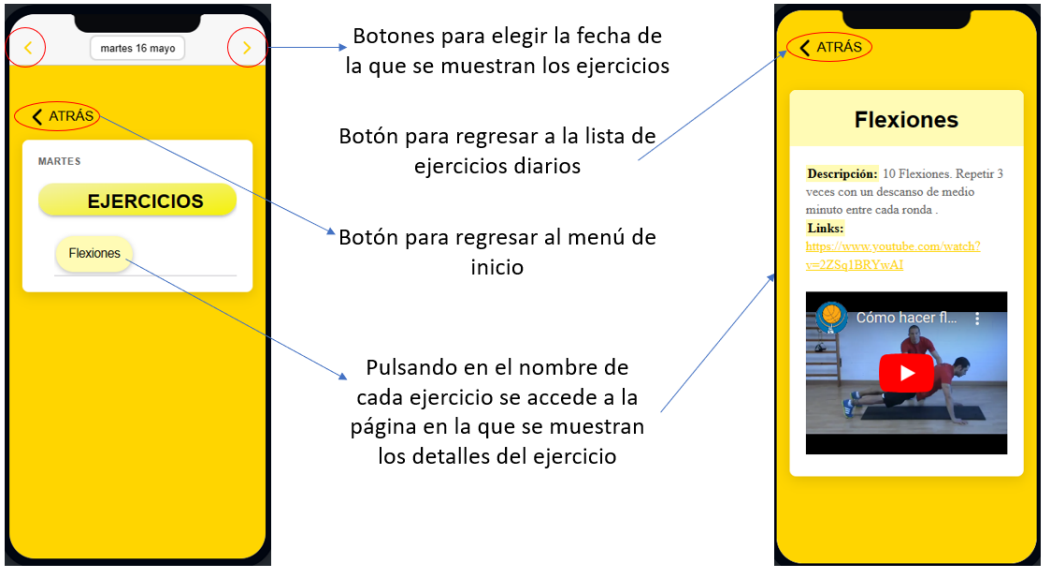
Botones para responder a las preguntas que aparecerán (una vez al día) en la página de Inicio






jueves 16 marzo
Atrás
JUEVES
Desayuno
Café
Comida
Espinacas
Merluza
Tarta de manzana
Merienda
Cena
Botones para elegir la fecha de la que se muestra la dieta
Botón para regresar a la dieta
Botón para regresar al menú de comida
Pulsando en cada plato se accede a la página en la que se muestran los detalles de la receta
ATRÁS
Pizza
Principal
Descripción: 10 minutos al horno
Comentarios: Acompañala con tus ingredientes favoritos
Links de recetas:
https://www.recetasgratis.net/receta-de-pizza-casera-31391.html
https://elpais.com/elcomidista/2022/03/08/articulo/1646730571_956363.html
Cómo hacer PIZZ...


QUESO RALLADO
75 gramos
DOMINGO
ESPÁRRAGOS BLANCOS
700 gramos
DOMINGO
LECHE EVAPORADA
0.5 litros
DOMINGO
CANÓNIGOS
50 gramos
DOMINGO
TOMATES CHERRY
Casillas para marcar los alimentos ya adquiridos


martes 16 mayo
ATRÁS
MARTES
EJERCICIOS
Flexiones
Botones para elegir la fecha de la que se muestran los ejercicios
Botón para regresar a la lista de ejercicios diarios
Botón para regresar al menú de inicio
Pulsando en el nombre de cada ejercicio se accede a la página en la que se muestran los detalles del ejercicio
ATRÁS
Flexiones
Descripción: 10 Flexiones. Repetir 3 veces con un descanso de medio minuto entre cada ronda .
Links:
https://www.youtube.com/watch?v=2ZSq1BRYwAI
Cómo hacer fl...

ÍNDICE DE FIGURAS

Índice de código fuente

BIBLIOGRAFÍA

[1] Angular: *Angular - domsanitizer api*. https://angular.io/api/platform-browser/DomSanitizer. Accedido el 6 de Marzo de 2023.

[2] Angular: *Angular - router guide*. https://angular.io/guide/router. Accedido el 26 de Noviembre de 2023.

[3] Angular: *Httpclient*. https://angular.io/api/common/http/HttpClient. Accedido el 20 de febrero de 2023.

[4] Astah: *Documentación oficial de astah*. https://astah.net/support/astah-pro/user-guide/, Accedido el 18 de mayo de 2023.

[5] Binary Coffe: *Angular - Guards*. https://binarycoffee.dev/post/guards-en-angular-como-funcionan. Accedido el 3 de febrero de 2023.

[6] Capacitor: *Capacitor local notifications*. https://capacitorjs.com/docs/apis/local-notifications. Accedido el 9 de marzo de 2023.

[7] Capacitor: *Capacitor storage*. https://capacitorjs.com/docs/apis/storage. Accedido el 8 de marzo de 2023.

[8] Kristina Chodorow y Michael Dirolf: *MongoDB: The Definitive Guide*. O'Reilly Media, 2013.

[9] Stack Overflow Community: *Stack overflow - generate random number between two numbers in javascript*. https://stackoverflow.com/questions/4959975/generate-random-number-between-two-numbers-in-javascript. Accedido el 24 de Febrero de 2023.

[10] Stack Overflow Community: *Stack overflow - ionic 4, angular 7: Passing object data to another page*. https://stackoverflow.com/questions/54304481/ionic-4-angular-7-passing-object-data-to-another-page. Accedido el 28

de Febrero de 2023.

[11] Jon Duckett: *HTML and CSS: Design and Build Websites*. Wiley, 2011.

[12] Adam Freeman y Steven Robson: *Angular Development with TypeScript*. The Pragmatic Programmers, 2019.

[13] Google: *Angular*. `https://angular.io/`. Accedido el 10 de marzo de 2023.

[14] Hampton Herman y Chris Meyer: *Sass: Up and Running*. O'Reilly Media, Sebastopol, CA, 2018, ISBN 978-1-491-94461-6.

[15] Ionic: *Documentación oficial de ionic*. `https://ionicframework.com/docs`. Accedido el 12 de marzo de 2023.

[16] Ionic: *Ionic framework - alert api documentation*. `https://ionicframework.com/docs/api/alert`. Accedido el 26 de Febrero de 2023.

[17] Ionic: *Nativestorage*. `https://ionicframework.com/docs/native/native-storage`. Accedido el 5 de junio de 2023.

[18] Joyce Justin y Joseph Jude: *Learn Ionic 2 - Develop Multi-platform Mobile Apps*. Packt Publishing, 2017.

[19] Nancy Kwallek, Cara M Lewis y Ann S Robbins: *Effects of office interior color on workers' mood and productivity*. Perceptual and Motor Skills, 66(1):123–128, 1988.

[20] MongoDB: *Documentación oficial de mongodb*. `https://docs.mongodb.com/`. Accedido el 14 de marzo de 2023.

[21] Mozilla Developer Network: *Http methods*. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods`. Accedido el 18 de febrero de 2023.

[22] Mozilla Developer Network: *Mdn web docs - iframe (html element)*. `https://developer.mozilla.org/es/docs/Web/HTML/Element/iframe`, 2023. Accedido el 8 de Marzo de 2023.

[23] Kyle Plugge, Peter Hawkins y Tim Membrey: *MongoDB in Action*. Manning Publications, 2019.

[24] Sass: *Documentación oficial de scss*. `https://sass-lang.com/documentation`, Accedido el 22 de marzo de 2023.

[25] John Smith: *Token-Based Authentication: A Comprehensive Guide*. PublisherXYZ,

2021, ISBN 978-1234567890.

[26] Jeremy Wilken: *Ionic in Action*. Manning Publications, 2016.

[27] Sani Yusuf: *Mastering Ionic: Build cutting-edge apps using UI Components, Angular services, Firebase, and more*. Packt Publishing, 2017.