Aplicación web para la medición de raciones en alimentos

Lorena Fernandez Elena
Grado de ingeniería informática

Gregorio Robles Martínez

Junio 2022

Agradecimientos
A mi **madre**,
por hacer posible que
cumpla mis sueños

y a mi **hermano pequeño**,
por sacarme siempre
una sonrisa en los
peores momentos.

FICHA DEL TRABAJO FINAL

Título del trabajo:	Aplicación web para la medición de raciones en alimentos
Nombre del autor:	Lorena Fernandez Elena
Nombre del consultor:	Gregorio Robles Martínez
Fecha de entrega:	06/2022
Área del Trabajo Final:	Desarrollo web
Titulación:	*Grado de Ingeniería Informática*

Resumen del Trabajo (máximo 250 palabras):

Hoy en día se calcula que hay aproximadamente 2 de cada 10 personas en España que están realizando una dieta [1] y, de este grupo, 8 de cada 10 fracasan en el seguimiento de la misma [2]. Asimismo, nos encontramos con que enfermedades como la diabetes, que requiere una dieta estricta, la padecen cerca del 8.3% de la población mundial.

Aunque hay muchos factores que pueden derivar en el abandono de una dieta, el no entender cómo aplicarla no debería ser uno de ellos. Con esta idea en mente, se plantea la creación de AMHRA: una aplicación web para la medición de raciones de hidratos de carbono en los alimentos.

Esta aplicación se plantea como una ayuda para todas aquellas personas que inician un proceso dietético con cualquier finalidad, pero con ciertos detalles de cara al usuario que sufre de diabetes. Con AMHRA, se busca reducir el proceso de aprendizaje, que actualmente requiere de largos cursos por parte de los tutores o la persona en cuestión donde se tienen que memorizar largas listas de equivalencias y/u otros conocimientos para garantizar la salud del aludido.

A nivel técnico, se ha apostado por el desarrollo de una API REST con java y basado en el *framework* Spring Boot y se ha optado por ReactJs para la parte de la interfaz del usuario. Asimismo, se ha priorizado la interacción intuitiva y el diseño “responsive” de cara a favorecer la comodidad del usuario para el que está pensado AMHRA.

Abstract (in English, 250 words or less):

Nowadays, it is estimated that there are approximately 2 out of 10 people in Spain who are following a diet and, among this group, 8 out of 10 fail to follow it. Likewise, we find that diseases such as diabetes, which requires a strict diet, are suffered by about 8.3% of the world population.

Although there are many factors that can lead to abandoning a diet, not understanding how to apply it should not be one of them. With this idea in mind, the creation of AMHRA has been proposed: a web application for measuring carbohydrate portions in food.

This application is conceived as an aid for all those people who start a dietary process with any purpose, but with certain details for the user who suffers from diabetes. With AMHRA, the aim is to reduce the learning process, which currently requires long courses by the tutors or the person in question where long lists of equivalences and/or other knowledge must be memorized to guarantee the health of the person in question.

At a technical level, it has opted for the development of a REST API with JAVA and based on the Spring Boot framework and ReactJS has been chosen for the user interface part. Likewise, intuitive interaction and "responsive" design have been prioritized in order to favor the comfort of the user for whom AMHRA is designed.

Palabras clave (entre 4 y 8):

Spring Boot, API REST, ReactJs, Alimentación, Equivalencias, Hidratos de Carbono, Dietas.

Índice

Lista de figuras

1. Introducción

1. Contexto y justificación del Trabajo

Hoy en día se calcula que 8 de cada 10 personas fracasan en el seguimiento de una dieta [2]. Aunque hay muchos factores que pueden derivar en esta decisión, el no entender cómo alimentarse o las cantidades de cada alimento a ingerir no debería ser uno de ellos. Con esta idea en mente, se plantea la creación de AMHRA: una aplicación web para la medición de raciones de hidratos de carbono en los alimentos

Actualmente, este proceso requiere de realizar ciertos cursos por parte de los tutores o la persona en cuestión donde se tienen que memorizar largas listas de equivalencias y/u otros conocimientos para garantizar la correcta alimentación del usuario. La otra alternativa es costearse a un nutricionista que esté pendiente del proceso dietético y te realice una guía personalizada. No obstante, esta opción no es posible para mucha gente cuyos recursos son limitados.

Mediante la implementación de AMHRA, se busca proporcionar una ayuda a cualquier proceso dietético para que su adaptación sea más amigable para los usuarios y reducir el número de personas que abandonan esas dietas en favor de formas de alimentación menos saludables.

2. Objetivos del Trabajo

Esta aplicación pretende facilitar la comprensión y uso de las raciones de hidratos de carbono de los alimentos para todas aquellas personas que por motivos de salud o personales requieran realizar estas mediciones, especialmente aquellas que tienen que convivir con la diabetes.

Dentro de este cómputo, se definen los siguientes subobjetivos:

- Disponer de una lista de alimentos con sus respectivas equivalencias. Dicha lista debe ser actualizable y permitir que el usuario añada nuevos alimentos no definidos.

- Priorizar que la aplicación sea intuitiva y que requiera un tiempo de aprendizaje menor a tener que memorizar dichas mediciones.

- Automatizar el proceso para reducir errores humanos.

- Proporcionar una forma de calcular las raciones y/o hidratos de carbono consumidos, así como crear menús personalizados.

3. Enfoque y método seguido

Este proyecto se ha enfocado desde la perspectiva de crear un producto nuevo con el fin de solventar una necesidad, que se define en los objetivos. Esto implica que la

persona responsable requiere de crear, informarse, configurar, documentar, desarrollar y probar todo aquello que implica el proyecto.

Por otro lado, el enfoque que se le ha dado al proyecto ha sido el de separar la lógica de la aplicación y la interfaz del usuario. Esto se ha logrado con la creación de dos aplicaciones:

- Un *backend* que actúa como una *API REST* para la creación, edición y, en general, manejo de los datos almacenados en la base de datos.
- Un *frontend* con un framework reactivo que se encarga de consumir el *backend.*

Esta decisión supone un reto a nivel personal ya que requiere de un uso extensivo de múltiples lenguajes de programación (JavaScript, Java, JSX) junto a HTML y la creación de un canal de comunicación efectivo entre ambas aplicaciones.

El método seguido para el correcto desarrollo de este proyecto ha consistido en generar un diagrama de Gantt generalizado sobre el cual se han ido definiendo subobjetivos según el ciclo. Como base para esta metodología, se ha tomado referencia “Scrum” [1] [2], pero adaptándola a un proyecto conformado por una única persona.

Este ciclo ha sido bastante variante debido al entorno personal, laboral y estudiantil sobre el que se ha tenido que basar. Inicialmente, se definió como un ciclo a revisar semanalmente con los progresos y próximos objetivos. No obstante, ante el cambio en mi entorno laboral, estos pasaron a revisarse de cada dos o tres semanas ya que el tiempo disponible para mis obligaciones universitarias se limitaba a los fines de semana.

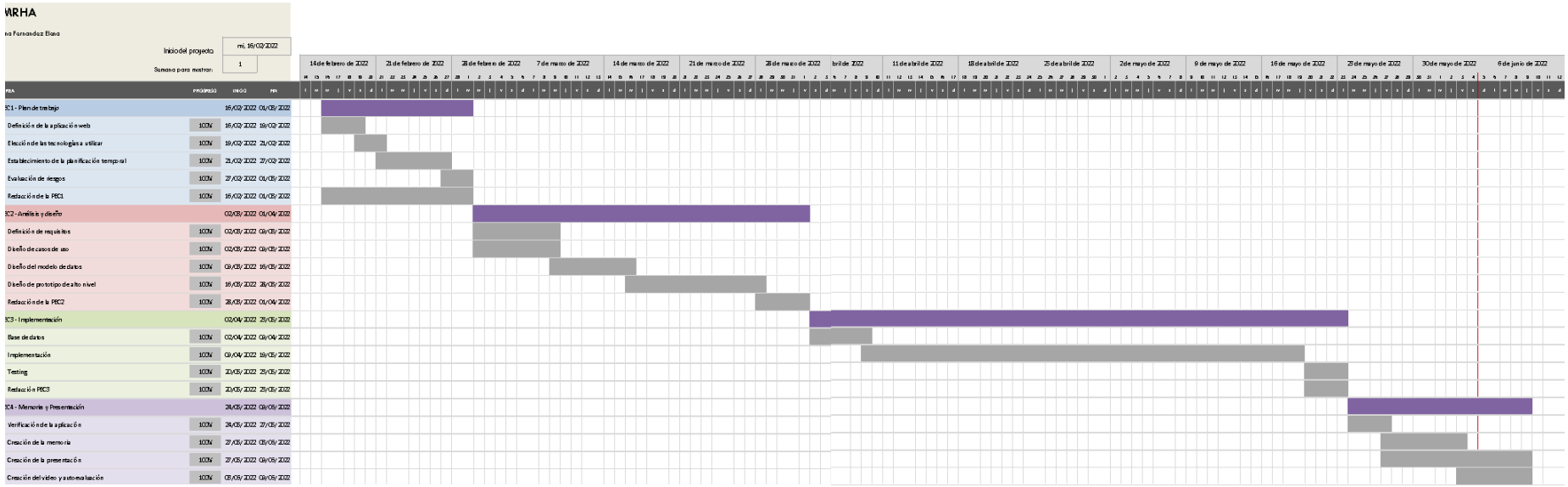
4. Planificación del Trabajo

La planificación del trabajo resulta un punto clave en cualquier proyecto independientemente del número de personas que conformen el equipo encargado de completarlo. En este caso, es especialmente decisivo ya que el tiempo y recursos del que se disponen es sumamente limitado.

Situados en este contexto, los diagramas de Gantt [3] son ayudas gráficas y visuales que se emplean para aspectos referentes a la de carga de trabajo y de operaciones que se producen en cualquier tipo de organización, sea productiva o social. Este diagrama debe ser actualizado de manera constante para lograr una utilización efectiva.

Si se tiene en cuenta la metodología por ciclos aplicada a este proyecto, el diagrama de Gantt es una opción efectiva, visual y fácilmente modificable, Inicialmente, se había escogido representar las tareas activas sin completar y las pendientes en color morado y las tareas completadas o parcialmente completadas en gris. Dado que esta se trata de la última versión, ya se muestran todas las tareas finalizadas.

Ilustración 1. Diagrama de Gantt. Planificación temporal



5. Breve sumario de productos obtenidos

A lo largo del transcurso del proyecto, se han empleado múltiples técnicas y tecnologías que han generado numerosos productos definidos en esta memoria. Para un mejor entendimiento de ellos, es necesario realizar una breve descripción de estos:

- Planificación: un diagrama de Gantt que muestra el plan de trabajo seguido en el transcurso del desarrollo.

- Prototipo: una maqueta no funcional de la aplicación. Representa de forma visualmente atractiva el concepto a seguir en la implementación de la interfaz.

- Memoria: Este documento técnico en el que se detalla todo aquello implicado con la creación del trabajo de final de grado.

- Presentación: Representación visual e informativa de la aplicación y sus funcionalidades.

- Video de presentación: Archivo audiovisual en el que se presenta este proyecto, así como sus funcionalidades de forma concisa.

- *Backend* de AMHRA: Aplicación desarrollada mediante el uso del framework Spring Boot y con la intención de conformar un *API REST*.

- *Frontend* de AMHRA: Aplicación desarrollada mediante el uso del framework reactivo ReactJs y con la intención de conformar la interfaz del usuario.

6. Breve descripción de los otros capítulos de la memoria

El desarrollo de AMHRA ha requerido de múltiples técnicas y tecnologías que resulta necesario documentar para entender el contexto general de las aplicaciones generadas. A continuación, se provee una breve introducción sobre los capítulos que conforman esta memoria:

- Tecnologías empleadas: Se identifican aquellas tecnologías y la razón de uso.

- Evaluación de riesgos: Análisis sobre los potenciales riesgos del proyecto y acciones necesarias para mitigarlos.

- Requisitos del proyecto: Cómputo usuarios destino y requisitos del proyecto y su especificación.

- Diseño: Conjunto de decisiones de diseño de AMHRA a nivel de interfaz como de lógica.

- Implementación: Información sobre la creación y estructura que siguen el *frontend* y el *backend*.

- Pruebas: Colección de pruebas realizadas para asegurar el funcionamiento correcto de la aplicación.

- Cambios respecto a la planificación inicial: Conjunto de cambios, riesgos y acciones que han acontecido durante el desarrollo del proyecto.

- Conclusiones: Reflexiones finales acerca del resultado del proyecto y posibles mejoras de cara al futuro.

2. Tecnologías empleadas

Definir las tecnologías a emplearse en el proyecto es un punto que debe plantearse al inicio del proceso de desarrollo, pues requiere de informarse encarecidamente sobre todos los aspectos que pueden acontecer ante la elección de unas u otras.

Asimismo, ha de estudiarse la compatibilidad entre unas y otras, posibles trabas a encontrarnos o si siquiera es posible desarrollar aquello que tenemos planeado en estas tecnologías [7].

AMHRA se propuso desde un inicio como una aplicación que combinara un *backend* que actúa como una *API REST* para la creación, edición y, en general, manejo de los datos almacenados en la base de datos y un *frontend* con un *framework* reactivo que se encargara de consumir el *backend.*

No obstante, esto supone estudiar las posibles tecnologías en la que esta idea pudiese llevarse a cabo, así como valorar el tiempo y recursos disponibles para poder adentrarse en el estudio de las mismas. Como resultado, se definieron las siguientes tecnologías y herramientas:

- La lógica de la aplicación será monitorizada por el *backend,* el cual ha sido implementado con el lenguaje de programación Java, el cual es uno de los más utilizados a nivel mundial.

  En conjunto con este lenguaje, se ha empleado el *framework* Spring Boot, uno de los más populares en Java y que facilita en gran medida en gran medida la configuración de la aplicación, aspecto que resulta potencialmente beneficioso si se tiene en cuenta el tiempo del que se dispone para implementar este proyecto. [8]

  Asimismo, se ha decidido emplear estas tecnologías debido a la gran cantidad de documentación de la que se dispone, hecho que facilita en gran medida la investigación y uso de las mismas.

- La interfaz del usuario será implementada en el *frontend,* el cual ha sido implementado mediante el lenguaje de programación JSX, una extensión de la sintaxis de JavaScript [9] cuya sintaxis recuerda ligeramente a HTML.

  En conjunto con este lenguaje, se ha empleado la biblioteca ReactJs, una de las bibliotecas de JavaScript más utilizadas por las facilidades que proporciona y las notables características que aumentan la productividad.

  En este contexto, se pretende hacer uso de las librerías de componentes MUI y PrimeReact para la personalización y diseño de la interfaz.

- El sistema operativo que se empleará será Windows10 [10] de 64 bits.

- El IDE empleado para el desarrollo del *backend* será Eclipse [11]. Su elección sobre otros IDE tiene un componente personal, ya que ha sido el que más he empleado a lo largo de la carrera.

- El editor de código empleado para desarrollar el *frontend* será Visual Studio Code. Esto se debe a que cuenta con múltiples extensiones que resultan especialmente útiles para desarrollar un programa en React. Por ejemplo, Simple React Snippets.

- La herramienta para la administración de bases de datos que se utiliza es DBeaver. Esto se debe a que se trata de una herramienta gratuita que soporta una gran cantidad de bases de datos, como PostgreSQL.

- El programa de prototipado que se ha de emplear es Justinmind. Esta herramienta nos proporciona la posibilidad de crear y diseñar prototipos de una aplicación interactivos y, por lo tanto, visualmente atractivos.

- La herramienta de creación de diagramas que se ha empleado es Visual Paradigm. Uno de los puntos positivos que tiene con respecto a la competencia es la rapidez y sencillez con la que puedes crear cualquier diagrama. Asimismo, puedes emplearlo directamente desde su web, sin necesidad de instalarlo localmente.

3. Evaluación de riesgos

La evaluación de riesgos trata de analizar todas aquellas situaciones imprevistas que podrían ocurrir en relación a nuestro proyecto y crear planes de contingencia en caso de que ocurran. Asimismo, resulta especialmente útil clasificar estos riesgos según la posibilidad de que ocurran y el impacto que tendrían en caso de ocurrir.

En nuestro caso, se han detectado diversos riesgos asociados que podrían ser potencialmente peligrosos para cumplir con los plazos de entrega o con la extensión de las funcionalidades a implementar. En la siguiente tabla se definen:

Ilustración 2. Tabla de riesgos

Riesgo	Descripción	Posibilidad	Impacto	Acción
R1	Limitación temporal de desarrollo	MEDIA	ALTO	A1
R2	Situaciones personales imprevistas	MEDIA	ALTO	A2
R3	Uso de nuevas tecnologías	MEDIA	MEDIO	A3
R4	Consecuencias derivadas de la falta de experiencia	MEDIA	ALTO	A2
R5	Problemas con el equipo de trabajo	BAJA	ALTO	A2

Tal como se ha dicho anteriormente, es importante determinar unas acciones que permitan contener los problemas que deriven de la ocurrencia de uno de los riesgos. En el contexto de nuestra aplicación, estas podrían ser:

Ilustración 3. Tabla de acciones

Acción	Descripción
A1	Dado que el tiempo para realizar el TFG es bastante limitado y la aplicación ha de desarrollarse individualmente, se han de fijar hitos y franjas temporales realistas y cumplibles que den cierto margen de tiempo, necesario si se requiere hacer algún cambio no concebido en el plan inicial.
A2	A la hora de definir los requisitos de la aplicación es necesario definir aquellos que son absolutamente obligatorios y aquello que pueden implementarse de forma opcional. Asimismo, dentro de estos dos grupos sería recomendable definir un orden de prioridad. De esta forma, será posible organizarse y definir un orden de acción en caso de imprevisto o problemas derivados de la falta de experiencia en el desarrollo de una aplicación completa individualmente.
A3	Ante el uso de nuevas tecnologías en el desarrollo con las que aún se está realizando un proceso de familiarización, pueden surgir problemas inesperados. Dependiendo de la gravedad de la situación, esto puede suponer un cambio completo de tecnologías empleadas o requerir más tiempo de desarrollo. Por lo tanto, se ha de intentar completar las tareas lo antes posible por si se requiriera de un margen extra de tiempo.

4. Requisitos del proyecto

Previamente a la implementación de la aplicación, hay un paso sobre el que se definirán todas las funcionalidades y como afectarán a los usuarios que la empleen. Es por ello por lo que todo proyecto requiere de definir el público objetivo al que se está dirigiendo (*stakeholders*) [12] y los posibles intereses que podrían tener en el uso de nuestra plataforma (requisitos).

En nuestro caso, a partir de los objetivos establecidos en el apartado "Objetivos del trabajo", se plantean los *stakeholders* y los requisitos determinados por el interés de estos.

1. Identificación de usuarios

En el sistema se plantean dos tipos de usuarios (o stakeholders):

- Usuario no registrado. Se trata de un usuario que accede a nuestra plataforma, pero no tiene cuenta en la misma. Se le incentivará a registrarse para acceder a una mayor amalgama de funciones.

- Usuario registrado. Se trata de un usuario que ha accedido previamente a nuestra aplicación y se ha dado de alta en el sistema. Al acceder a nuestra plataforma, tendrá que iniciar sesión con sus credenciales.

2. Requisitos

Para poder ofrecer una visión más clara de los requisitos, se hará una división entre aquellos que se recogen para satisfaces las necesidades del usuario y aquellos que el sistema deberá cumplimentar para satisfacer las necesidades mencionadas anteriormente. Por lo tanto, se distinguirán entre requisitos funcionales y no funcionales, respectivamente.

Adicionalmente, cada requisito se agrupará según qué usuario es el objetivo y se valorará su prioridad tal como se ha indicado en el apartado de evaluación de riesgos.

Requisitos funcionales

Los requisitos funcionales según usuario son:

Ilustración 4. Tabla de requisitos funcionales de Usuario no Registrado

Usuario no registrado		
Identificador	**Descripción**	**Prioridad**
RF-UN01	Registrarse	Alta

Ilustración 5. Tabla de requisitos funcionales de Usuario Registrado

Usuario registrado		
Identificador	**Descripción**	**Prioridad**

RF-UR01	Añadir un alimento	Media
RF-UR02	Modificar un alimento creado por el usuario	Media
RF-UR03	Eliminar alimento creado por el usuario	Media
RF-UR04	Cerrar sesión	Alta
RF-UR05	Crear menú	Baja
RF-UR06	Modificar menú	Baja
RF-UR07	Eliminar menú	Baja
RF-UR08	Añadir racionalidad por comida	Baja
RF-UR09	Modificar racionalidad por comida	Baja
RF-UR10	Eliminar racionalidad por comida	Baja
RF-UR11	Iniciar sesión	Alta

Ilustración 6. Tabla de requisitos funcionales de todos los usuarios

Todos los usuarios		
Identificador	**Descripción**	**Prioridad**
RF-T01	Consultar la tabla de equivalencias	Alta
RF-T02	Filtrar la tabla de equivalencias	Alta
RF-T03	Consultar preguntas frecuentes	Alta
RF-T04	Cálculo de equivalencias	Alta

Requisitos no funcionales

Ilustración 7. Tabla de requisitos no funcionales

Identificador	**Descripción**	**Prioridad**
RNF - 01	El sistema debe poder utilizarse en los navegadores más utilizados	Media
RNF - 02	El sistema debe adaptarse a la pantalla del móvil.	Baja

3. Casos de uso

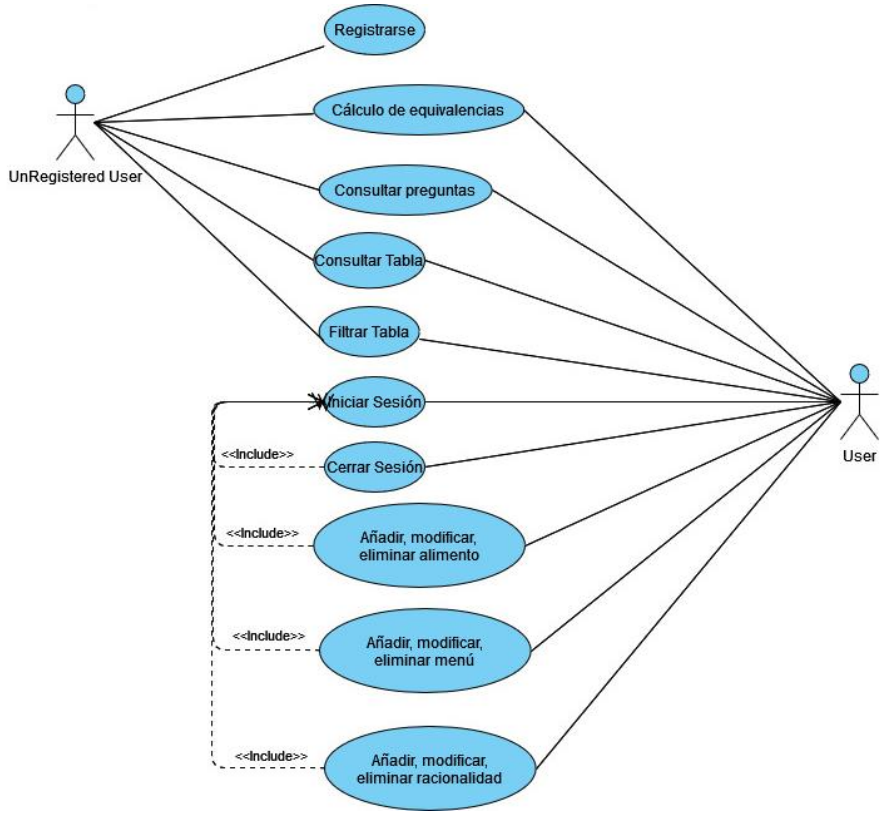
Los casos de uso [13] son una forma gráfica de visualizar los requisitos funcionales de una aplicación en el caso de un diagrama o con un documento textual donde se incluyen las especificaciones.

Estos casos permiten definir de forma clara las acciones a tomar para llegar a un determinado fin y nos proporcionan una primera vista de cómo debería implementarse cierto requisito.

En nuestro caso, se ha realizado un diagrama general donde se muestra que requisitos puede o no puede realizar cada usuario y la especificación textual de cada uno de ellos, representados en tablas [15].

Diagrama

Ilustración 8. Diagrama de casos de uso



Especificación

Ilustración 9. Especificación requisitos Registrarse

Identificador	RF-UN01
Caso de Uso	Registrarse
Stakeholder	Usuarios no registrados (UnRegistered User)
Precondición	Ninguna
Escenario principal de éxito	1. El usuario accede a la sección de registro 2. El sistema le muestra un formulario de registro 3. El usuario rellena el formulario y clica en registrarse 4. El sistema valida los datos del usuario como correctos. 5. El sistema da de alta al usuario. 6. El sistema notifica al usuario.
Escenarios alternativos	Paso 4. 1. Los valores introducidos por el usuario no son correctos (la cuenta ya existe, la contraseña no cumple con las condiciones…) a. El sistema notifica al usuario del error. b. Volvemos al paso 3.
Postcondición	El usuario es capaz de ver la tabla de equivalencias filtrada.

Ilustración 10. Especificación requisitos Añadir alimento

Identificador	RF-UR01
Caso de Uso	Añadir un alimento
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	1. El usuario accede a la sección de añadir alimento 2. El sistema muestra el formulario para añadir un alimento 3. El usuario rellena el formulario. 4. El sistema valida que los datos son correctos. 5. El sistema añade el alimento. 6. El sistema notifica al usuario.
Escenarios alternativos	Paso 4. 1. Los valores introducidos por el usuario no son correctos (El alimento ya existe) a. El sistema notifica al usuario y le ofrece acceder al formulario de modificación. i. El usuario decide añadir un alimento distinto. 1. Volvemos al paso 3. ii. El usuario decide modificar el alimento. 1. Vamos al caso “Modificar alimento”
Postcondición	El alimento introducido se ha añadido correctamente.

Ilustración 11. Especificación requisitos Modificar alimento

Identificador	RF-UR02
Caso de Uso	Modificar un alimento
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	1. El usuario accede a la sección de modificar alimento 2. El sistema muestra el formulario para modificar un alimento 3. El usuario rellena el formulario. 4. El sistema valida que los datos son correctos. 5. El sistema modificar el alimento. 6. El sistema notifica al usuario.
Escenarios alternativos	Paso 4. 2. Los valores introducidos por el usuario no son correctos (El alimento no existe) a. El sistema notifica al usuario y le ofrece acceder al formulario de añadir alimento. i. El usuario decide modificar un alimento distinto. 1. Volvemos al paso 3. ii. El usuario decide añadir el alimento. 1. Vamos al caso “Añadir alimento”
Postcondición	El alimento introducido se ha modificado correctamente.

Ilustración 12. Especificación requisitos Eliminar alimento

Identificador	RF-UR03
Caso de Uso	Eliminar un alimento
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	7. El usuario accede a la sección de eliminar alimento 8. El sistema muestra el formulario para eliminar un alimento 9. El usuario rellena el formulario. 10. El sistema valida que los datos son correctos. 11. El sistema elimina el alimento. 12. El sistema notifica al usuario.
Escenarios alternativos	Paso 4. 3. Los valores introducidos por el usuario no son correctos (El alimento no existe) a. El sistema notifica al usuario. b. Volvemos al paso 3.
Postcondición	El alimento introducido se ha añadido correctamente.

Ilustración 13. Especificación requisitos Cerrar Sesión

Identificador	RF-UR04
Caso de Uso	Cerrar sesión
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	1. El usuario clica sobre el botón de cerrar sesión 2. El sistema le pide que confirme su acción 3. El usuario confirma su acción 4. El sistema termina la sesión.
Escenarios alternativos	Paso 3. 1. El usuario cancela la acción. a. El sistema no cierra la sesión del usuario.
Postcondición	El usuario ha cerrado sesión correctamente.

Ilustración 14. Especificación requisitos Añadir menú

Identificador	RF-UR05
Caso de Uso	Añadir un menú
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	1. El usuario accede a la sección de añadir un menú 2. El sistema muestra el formulario para añadir un menú 3. El usuario rellena el formulario. 4. El sistema valida que los datos son correctos. 5. El sistema añade el menú. 6. El sistema notifica al usuario.
Escenarios alternativos	Paso 4. 4. Los valores introducidos por el usuario no son correctos (El menú ya existe) a. El sistema notifica al usuario y le ofrece acceder al formulario de modificación. i. El usuario decide añadir un menú distinto. 1. Volvemos al paso 3. ii. El usuario decide modificar el menú. 1. Vamos al caso “Modificar menú”
Postcondición	El menú introducido se ha añadido correctamente.

Ilustración 15. Especificación requisitos Modificar menú

Identificador	RF-UR06
Caso de Uso	Modificar un menú
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	1. El usuario accede a la sección de modificar menú 2. El sistema muestra el formulario para modificar un menú 3. El usuario rellena el formulario. 4. El sistema valida que los datos son correctos. 5. El sistema modificar el menú. 6. El sistema notifica al usuario.
Escenarios alternativos	Paso 4. 5. Los valores introducidos por el usuario no son correctos (El menú no existe) a. El sistema notifica al usuario y le ofrece acceder al formulario de añadir menú. i. El usuario decide modificar un menú distinto. 1. Volvemos al paso 3. ii. El usuario decide añadir el menú. 1. Vamos al caso “Añadir menú”
Postcondición	El menú introducido se ha modificado correctamente.

Ilustración 16. Especificación requisitos Eliminar menú

Identificador	RF-UR07
Caso de Uso	Eliminar un menú
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	1. El usuario accede a la sección de eliminar menú 2. El sistema muestra el formulario para eliminar un menú 3. El usuario rellena el formulario. 4. El sistema valida que los datos son correctos. 5. El sistema elimina el menú. 6. El sistema notifica al usuario.
Escenarios alternativos	Paso 4. 6. Los valores introducidos por el usuario no son correctos (El menú no existe) a. El sistema notifica al usuario. b. Volvemos al paso 3.
Postcondición	El menú introducido se ha añadido correctamente.

Ilustración 17. Especificación requisitos añadir racionalidad

Identificador	RF-UR08
Caso de Uso	Añadir racionalidad
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	1. El usuario accede a la sección de añadir racionalidad 2. El sistema muestra el formulario para añadir racionalidad 3. El usuario rellena el formulario. 4. El sistema valida que los datos son correctos. 5. El sistema añade la racionalidad. 6. El sistema notifica al usuario.
Escenarios alternativos	Paso 4. 7. Los valores introducidos por el usuario no son correctos (Ya existe una racionalidad) a. El sistema notifica al usuario y le ofrece acceder al formulario de modificación. i. El usuario decide modificar la racionalidad. 1. Vamos al caso “Modificar la racionalidad”
Postcondición	La racionalidad introducida se ha añadido correctamente.

Ilustración 18. Especificación requisitos Modificar racionalidad

Identificador	RF-UR09
Caso de Uso	Modificar racionalidad
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	1. El usuario accede a la sección de modificar racionalidad 2. El sistema muestra el formulario para modificar racionalidad 3. El usuario rellena el formulario. 4. El sistema valida que los datos son correctos. 5. El sistema modificar la racionalidad. 6. El sistema notifica al usuario.
Escenarios alternativos	Paso 4. 8. Los valores introducidos por el usuario no son correctos (No hay datos de racionalidad previos) a. El sistema notifica al usuario y le ofrece acceder al formulario de añadir racionalidad. i. El usuario decide añadir la racionalidad. 1. Vamos al caso “Añadir racionalidad”
Postcondición	La racionalidad introducida se ha modificado correctamente.

Ilustración 19. Especificación requisitos Eliminar racionalidad

Identificador	RF-UR10
Caso de Uso	Eliminar racionalidad
Stakeholder	Usuario registrado (User)
Precondición	Haber iniciado sesión en el sistema
Escenario principal de éxito	1. El usuario solicita eliminar la racionalidad 2. El sistema pide confirmación para eliminar la racionalidad 3. El usuario confirma la acción. 4. El sistema elimina la racionalidad. 5. El sistema notifica al usuario.
Escenarios alternativos	
Postcondición	La racionalidad se ha añadido correctamente.

Ilustración 20. Especificación requisitos Iniciar sesión

Identificador	RF-UR11
Caso de Uso	Iniciar sesión
Stakeholder	Usuario registrado (User)
Precondición	Ninguna
Escenario principal de éxito	1. El usuario accede a la sección de iniciar sesión 2. El sistema le muestra un formulario de inicio de sesión 3. El usuario rellena el formulario y clica en iniciar sesión. 4. El sistema valida los datos del usuario como correctos. 5. El sistema le da acceso al usuario a las funciones de registro.
Escenarios alternativos	Paso 4. 2. Los valores introducidos por el usuario no son correctos (la cuenta no existe, la contraseña no es correcta…) a. El sistema notifica al usuario del error. b. Volvemos al paso 3.
Postcondición	El usuario es capaz de iniciar sesión.

Ilustración 21. Especificación requisitos Consultar tabla equivalencias

Identificador	RF-T01
Caso de Uso	Consultar la tabla de equivalencias
Stakeholder	Todos los usuarios
Precondición	Ninguna
Escenario principal de éxito	1. El usuario accede a la página principal 2. El sistema muestra la tabla con sus respectivas equivalencias.
Escenarios alternativos	
Postcondición	El usuario es capaz de ver la tabla de equivalencias.

Ilustración 22. Especificación requisitos Filtrar tabla equivalencias

Identificador	RF-T02
Caso de Uso	Filtrar la tabla de equivalencias
Stakeholder	Todos los usuarios
Precondición	El usuario ha de haber accedido a la página principal
Escenario principal de éxito	1. El usuario selecciona los filtros que desea 2. El sistema muestra la tabla según los filtros seleccionados
Escenarios alternativos	
Postcondición	El usuario es capaz de ver la tabla de equivalencias filtrada.

Ilustración 23. Especificación requisitos Consulta preguntas frecuentes

Identificador	RF-T03
Caso de Uso	Consultar preguntas frecuentes
Stakeholder	Todos los usuarios
Precondición	Ninguna
Escenario principal de éxito	1. El usuario accede a la sección de “FAQS” 2. El sistema muestra las preguntas más frecuentes y sus respuestas
Escenarios alternativos	
Postcondición	El usuario es capaz de ver las preguntas más frecuentes.

Ilustración 24. Especificación requisitos Cálculo de equivalencias

Identificador	RF-T04
Caso de Uso	Cálculo de equivalencias
Stakeholder	Todos los usuarios
Precondición	Ninguna
Escenario principal de éxito	1. El usuario accede a la sección de “Cálculo de equivalencias” 2. El sistema muestra los formularios de equivalencias. 3. El usuario introduce los datos necesarios. 4. El sistema realiza el cálculo y lo muestra por pantalla.
Escenarios alternativos	
Postcondición	El usuario es capaz de ver las preguntas más frecuentes.

5. Diseño

La propuesta de diseño es una parte de la planificación del proyecto que permite proporcionar una idea bastante cercana a lo que será el producto final que se quiere obtener. Para poder proporcionar esta perspectiva se empleará un diagrama de clases, entidad – relación, de la arquitectura del sistema y un prototipo no funcional de alto nivel.

1. Diagrama de clases

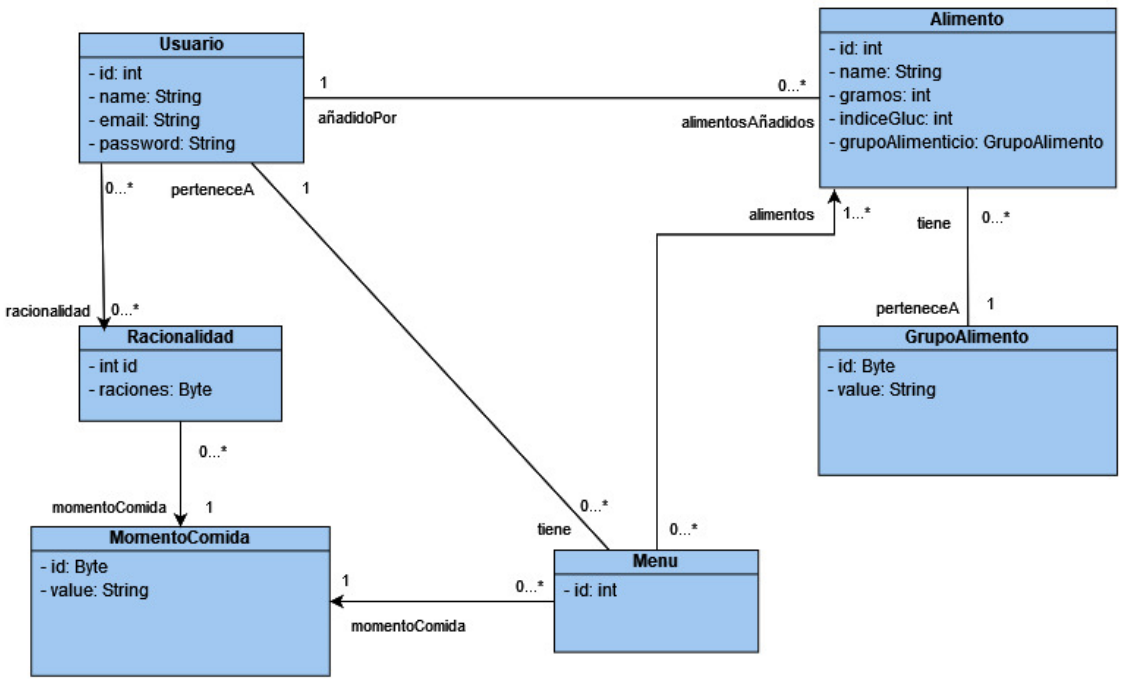
El diagrama de clases UML describe las clases del sistema, los atributos de cada uno y las relaciones entre sí para conformar un sistema completo.

En este caso, este diagrama nos proporcionará información de relevancia como el tipo de relación (esto definirá las etiquetas a usar) o cuál es el componente principal de la aplicación. Como es posible observar, alimento tiene cierta relevancia dentro del conjunto de clases y, por lo tanto, actuará como base sobre la aplicación.

Asimismo, otra entidad que resalta es usuario, el cual tendrá sus credenciales, una lista de alimentos añadidos, menús creados y racionalidades asignadas.

A su vez, cada racionalidad incluye un momento de la comida, que hace referencia a si se trata de una ración referente al desayuno, comida, cena… Y cada alimento presenta un grupo alimenticio referente a si pertenece a la agrupación de leches y derivados, frutas, verduras y hortalizas…

Ilustración 25. Diagrama de clases



2. Prototipo

Un prototipo es una maqueta no funcional donde se observan los primeros atisbos de lo que será la interfaz final del usuario. En este caso se ha decidido por optar por un prototipo de alto nivel, es decir, muestra únicamente cómo será visualmente la aplicación, pero no el comportamiento de esta. En él se representan las vistas más importantes de la plataforma.

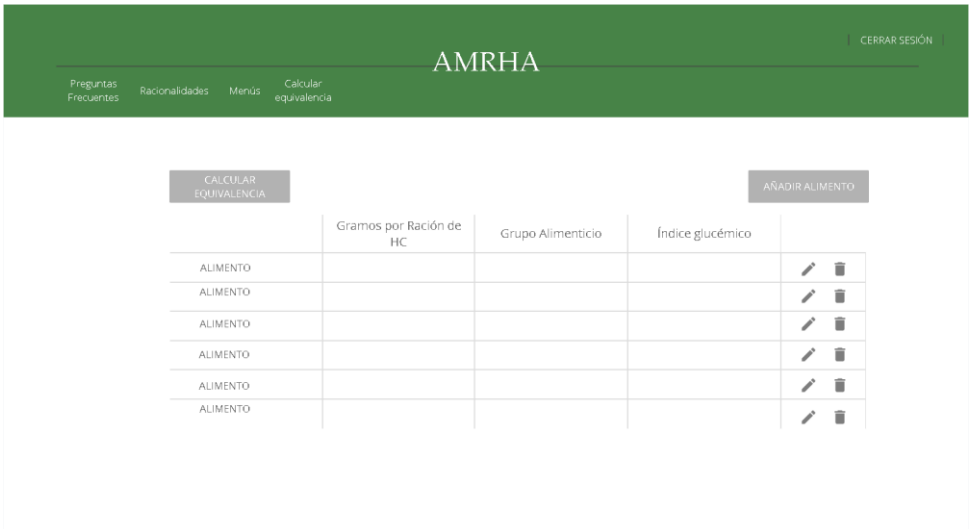
Para una mejor visualización del contenido se recomienda consultar el prototipo que se adjunta en formato HTML junto a esta parte de la memoria. No obstante, a continuación, se presentan capturas de las diferentes vistas (o componentes, en terminología de ReactJs).

Página Principal (Sin iniciar sesión)

Ilustración 26. Prototipo - Página principal no sesión

Página Principal (Sesión iniciada)

Ilustración 27. Página principal - Sesión iniciada

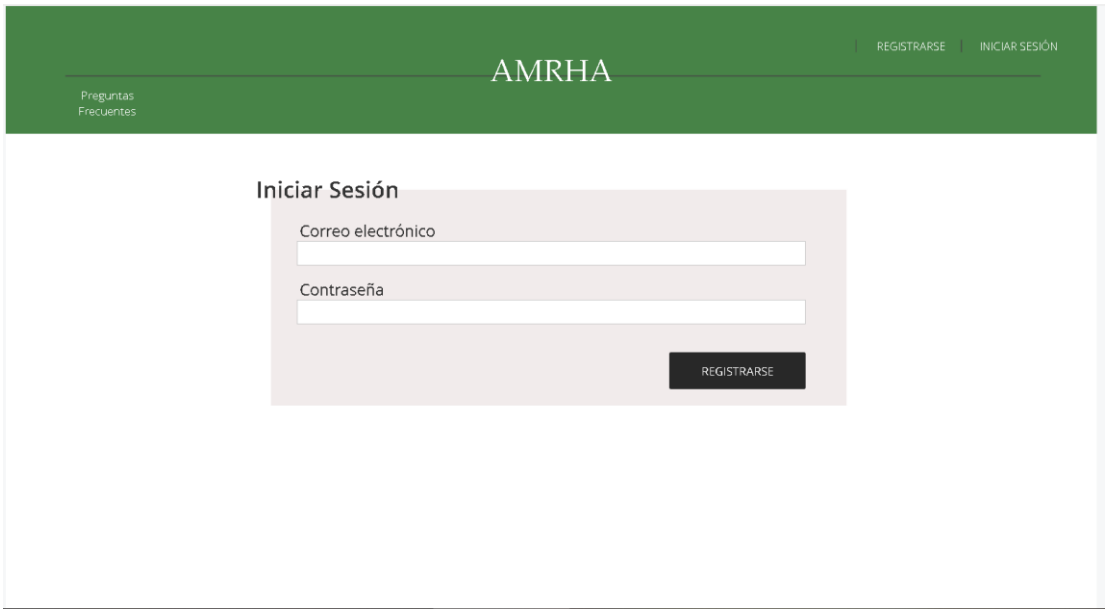


Formulario de registro

Ilustración 28. Prototipo - Formulario registro

Formulario de inicio de sesión

Ilustración 29. Prototipo - Formulario inicio sesión



Cálculo de equivalencias

Ilustración 30. Prototipo - Cálculo de equivalencias

Racionalidades

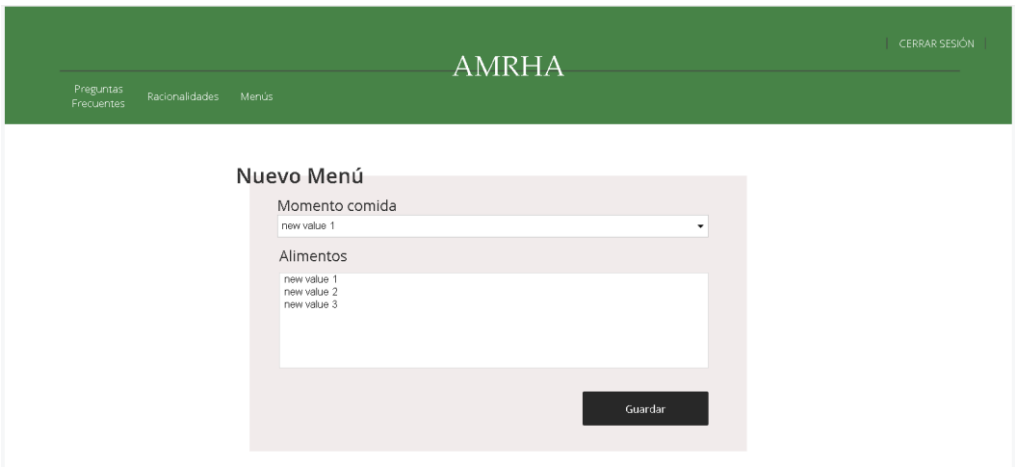
Ilustración 31. Prototipo - Racionalidades

Menús

Ilustración 32. Prototipo - Menús

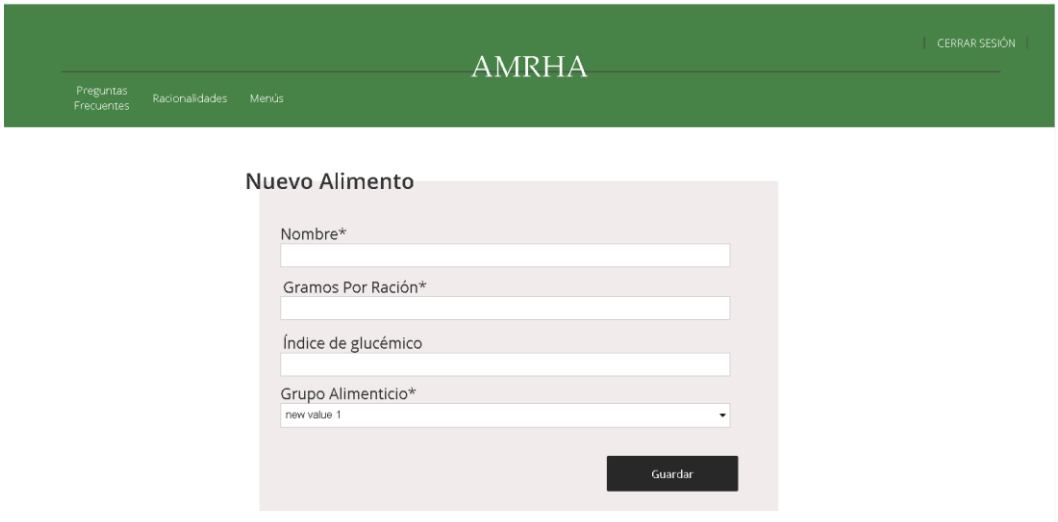
Añadir Menú

Ilustración 33. Prototipo - Añadir menú



Añadir Alimento

Ilustración 34. Prototipo - Añadir Alimento



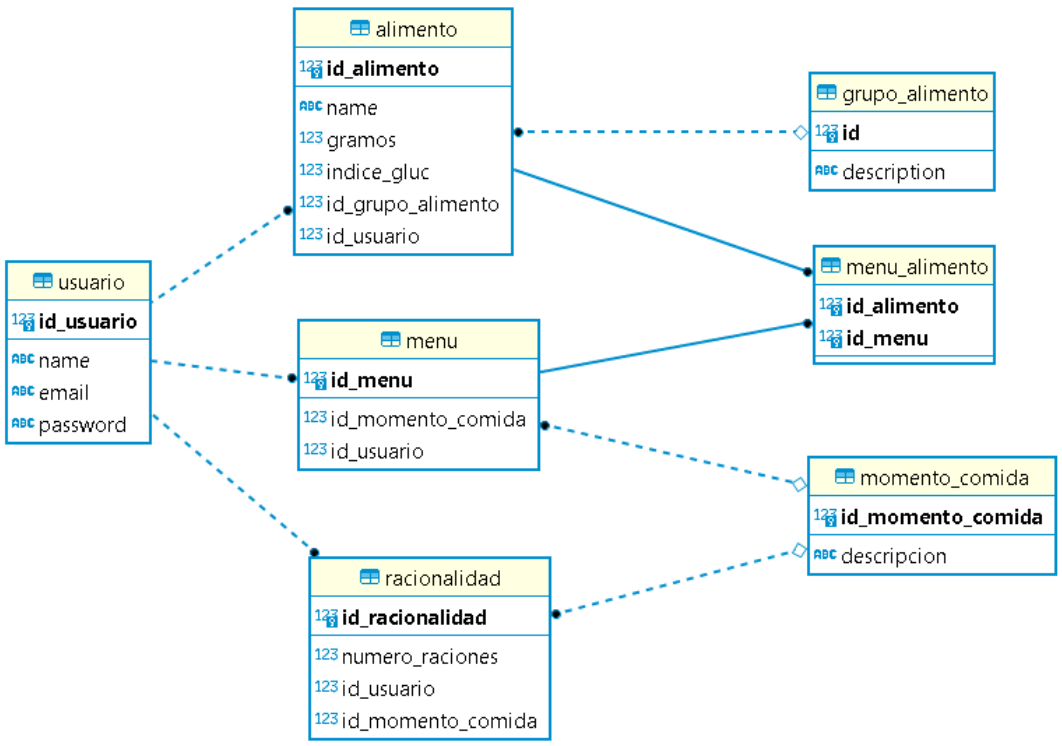
3. Diagrama Entidad-Relación

Un diagrama entidad – relación se plantea como una forma visual de observar las relaciones entre entidades, así como se relacionarán en la base de datos.

En este caso, se puede observar que nuestro sistema cuenta con 7 tablas, pero únicamente 6 de ellos son entidades. Esto se debe a que la tabla “menu_alimento” actúa como intermediaria en la relación muchos a muchos entre alimento y menú, demarcada con líneas continuas entre ambas entidades y esta tabla.

Otros aspectos que destacar de este tipo de diagrama son las líneas punteadas, que nos indican el resto de las relaciones entre tablas, y los atributos demarcados con negrita y con un icono de una llave, que se tratan de las claves primarias de cada entidad. Asimismo, se observa que el tipo del atributo está indicado con un icono delante de su nombre: “123” indica que es un atributo numérico y “ABC”, uno textual.

Ilustración 35. Diagrama entidad-relación



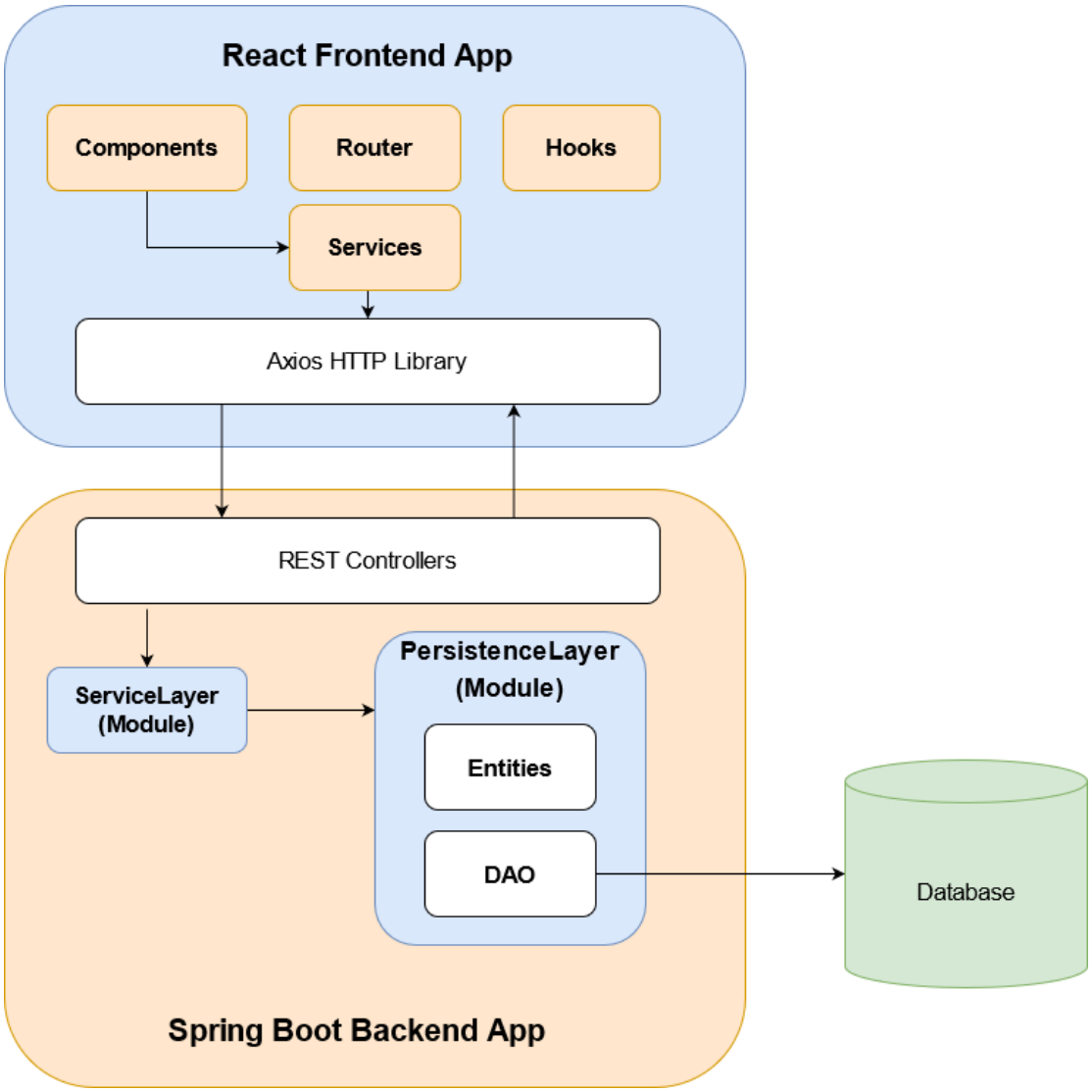
4. Diagrama de la arquitectura del sistema

Para una mejor comprensión de la arquitectura de la aplicación, se ha creado un diagrama de la arquitectura del sistema de alto nivel. En este se observa la clara división que se ha realizado entre el *backend* y el *frontend*, cada uno con sus propias tecnologías y *frameworks*.

Por un lado, el *backend* se desarrolla empleando el *framework* de Spring, más específicamente utilizando Spring Boot, que nos permite centrarnos mayormente en el desarrollo. Esto se debe a que esta tecnología se encarga de la mayor parte de la configuración. Dado que se ha separado del *frontend*, su punto de entrada se realizará a través de un API REST.

Por otro lado, el *frontend* se desarrolla con la librería de código abierto de Javascript, ReactJs. Esta funciona con una metodología centrada en las "Single Page Applications", la cual se encargará de consumir la API REST.

Ilustración 36. Diagrama de la arquitectura



6. Implementación

Para implementar los requisitos, diagramas y componentes determinados en los apartados anteriores, se ha elegido generar una API REST con Spring Boot con un enfoque por capas, que se encarga del *backend* de la aplicación, y una app con ReactJs con Hooks para el *frontend*.

A continuación, se proporciona una documentación en detalle de las herramientas y tecnologías empleadas durante el desarrollo. Específicamente, se entrará en más detalles sobre la estructura, implementación y funcionamiento de las dos aplicaciones.

1. Spring Boot *Backend*

En primer lugar, se va a analizar cómo ha sido el inicio de la implementación del *backend* y cómo ha sido su proceso de construcción y desarrollo de este. Así como explicar las decisiones de diseño y la función de cada uno de los módulos que lo componen.

Los primeros pasos realizados para la creación y configuración del *backend* de AMHRA han consistido en generar un proyecto Maven padre con un empaquetado POM. Como parte más reseñable de este módulo raíz, se destaca el archivo POM, que se trata de un XML en las que se han definido las dependencias, configuraciones y otra información importante sobre el proyecto.

Este módulo a su vez es hijo del proyecto starter del *framework* de Spring. Esto se puede observar en mayor detalle en la siguiente imagen extraída del su POM.

Ilustración 37. pom

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchem
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.7</version>
    </parent>
    <groupId>com.tfg</groupId>
    <artifactId>amhra</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>pom</packaging>
    <name>Amhra</name>
    <description>Aplicación web para la medición de raciones en alimentos</description>

```

Tal como se puede observar, también actúa como identificador de nuestro proyecto y se proporciona una descripción sobre la funcionalidad de este, así como la información de la versión empleada.

A continuación, se han definido unas primeras dependencias que se considerarían útiles de cara al desarrollo de la aplicación. El siguiente árbol de dependencias generado con el comando “mvn dependency:tree” nos proporciona una vista detallada de las versiones y de las dependencias que se instalan en cascada junto a ellas:

Ilustración 38. Dependencias *backend*

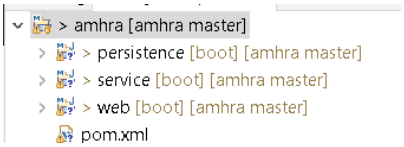
```
--- maven-dependency-plugin:3.2.0:tree (default-cli) @ web ---
[INFO] com.tfg:web:war:0.0.1-SNAPSHOT
[INFO] +- org.springframework.boot:spring-boot-starter-data-jpa:jar:2.6.7:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-aop:jar:2.6.7:compile
[INFO] |  |  +- org.springframework:spring-aop:jar:5.3.19:compile
[INFO] |  |  \- org.aspectj:aspectjweaver:jar:1.9.7:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-jdbc:jar:2.6.7:compile
[INFO] |  |  +- com.zaxxer:HikariCP:jar:4.0.3:compile
[INFO] |  |  \- org.springframework:spring-jdbc:jar:5.3.19:compile
[INFO] |  +- jakarta.transaction:jakarta.transaction-api:jar:1.3.3:compile
[INFO] |  +- jakarta.persistence:jakarta.persistence-api:jar:2.2.3:compile
[INFO] |  +- org.springframework.data:spring-data-jpa:jar:2.6.4:compile
[INFO] |  |  +- org.springframework:spring-orm:jar:5.3.19:compile
[INFO] |  |  \- org.springframework:spring-beans:jar:5.3.19:compile
[INFO] |  \- org.springframework:spring-aspects:jar:5.3.19:compile
[INFO] +- org.springframework.data:spring-data-elasticsearch:jar:4.3.4:compile
[INFO] |  +- org.springframework:spring-context:jar:5.3.19:compile
[INFO] |  |  \- org.springframework:spring-expression:jar:5.3.19:compile
[INFO] |  +- org.springframework:spring-tx:jar:5.3.19:compile
[INFO] |  +- org.springframework.data:spring-data-commons:jar:2.6.4:compile
[INFO] |  +- org.elasticsearch.client:transport:jar:7.15.2:compile
[INFO] |  |  +- org.elasticsearch:elasticsearch:jar:7.15.2:compile
[INFO] |  |  |  +- org.elasticsearch:elasticsearch-core:jar:7.15.2:compile
[INFO] |  |  |  +- org.elasticsearch:elasticsearch-secure-sm:jar:7.15.2:compile
[INFO] |  |  |  +- org.elasticsearch:elasticsearch-x-content:jar:7.15.2:compile
[INFO] |  |  |  |  +- com.fasterxml.jackson.dataformat:jackson-dataformat-smile:jar:2.13.2:compile
[INFO] |  |  |  |  +- com.fasterxml.jackson.dataformat:jackson-dataformat-yaml:jar:2.13.2:compile
[INFO] |  |  |  |  \- com.fasterxml.jackson.dataformat:jackson-dataformat-cbor:jar:2.13.2:compile
[INFO] |  |  |  +- org.elasticsearch:elasticsearch-geo:jar:7.15.2:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-core:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-analyzers-common:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-backward-codecs:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-grouping:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-highlighter:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-join:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-memory:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-misc:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-queries:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-queryparser:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-sandbox:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-spatial-extras:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-spatial3d:jar:8.9.0:compile
[INFO] |  |  |  +- org.apache.lucene:lucene-suggest:jar:8.9.0:compile
[INFO] |  |  |  +- org.elasticsearch:elasticsearch-cli:jar:7.15.2:compile
[INFO] |  |  |  |  \- net.sf.jopt-simple:jopt-simple:jar:5.0.2:compile
[INFO] |  |  |  +- com.carrotsearch:hppc:jar:0.8.1:compile
[INFO] |  |  |  +- org.lz4:lz4-java:jar:1.8.0:compile
[INFO] |  |  |  +- joda-time:joda-time:jar:2.10.10:compile
[INFO] |  |  |  +- com.tdunning:t-digest:jar:3.2:compile
[INFO] |  |  |  +- org.hdrhistogram:HdrHistogram:jar:2.1.9:compile
[INFO] |  |  |  +- org.apache.logging.log4j:log4j-api:jar:2.17.2:compile
[INFO] |  |  |  +- org.elasticsearch:jna:jar:5.7.0-1:compile
[INFO] |  |  |  \- org.elasticsearch:elasticsearch-plugin-classloader:jar:7.15.2:runtime
[INFO] |  |  +- org.elasticsearch.plugin:reindex-client:jar:7.15.2:compile
[INFO] |  |  |  \- org.elasticsearch:elasticsearch-ssl-config:jar:7.15.2:compile
[INFO] |  |  +- org.elasticsearch.plugin:lang-mustache-client:jar:7.15.2:compile
[INFO] |  |  |  \- com.github.spullara.mustache.java:compiler:jar:0.9.6:compile
[INFO] |  |  +- org.elasticsearch.plugin:percolator-client:jar:7.15.2:compile
[INFO] |  |  +- org.elasticsearch.plugin:parent-join-client:jar:7.15.2:compile
[INFO] |  |  \- org.elasticsearch.plugin:rank-eval-client:jar:7.15.2:compile
[INFO] |  +- org.elasticsearch.plugin:transport-netty4-client:jar:7.15.2:compile
[INFO] |  |  +- io.netty:netty-buffer:jar:4.1.76.Final:compile
[INFO] |  |  +- io.netty:netty-codec:jar:4.1.76.Final:compile
[INFO] |  |  +- io.netty:netty-codec-http:jar:4.1.76.Final:compile
[INFO] |  |  +- io.netty:netty-common:jar:4.1.76.Final:compile
[INFO] |  |  +- io.netty:netty-handler:jar:4.1.76.Final:compile
[INFO] |  |  +- io.netty:netty-resolver:jar:4.1.76.Final:compile
[INFO] |  |  \- io.netty:netty-transport:jar:4.1.76.Final:compile
[INFO] |  +- org.elasticsearch.client:elasticsearch-rest-high-level-client:jar:7.15.2:compile
[INFO] |  |  +- org.elasticsearch.client:elasticsearch-rest-client:jar:7.15.2:compile
[INFO] |  |  |  +- org.apache.httpcomponents:httpclient:jar:4.5.13:compile
[INFO] |  |  |  +- org.apache.httpcomponents:httpcore:jar:4.4.15:compile
[INFO] |  |  |  +- org.apache.httpcomponents:httpasyncclient:jar:4.1.5:compile
[INFO] |  |  |  +- org.apache.httpcomponents:httpcore-nio:jar:4.4.15:compile
```

```
[INFO] |  |  \- commons-codec:commons-codec:jar:1.15:compile
[INFO] |  +- org.elasticsearch.plugin:mapper-extras-client:jar:7.15.2:compile
[INFO] |  \- org.elasticsearch.plugin:aggs-matrix-stats-client:jar:7.15.2:compile
[INFO] +- com.fasterxml.jackson.core:jackson-databind:jar:2.13.2.1:compile
[INFO] |  \- com.fasterxml.jackson.core:jackson-annotations:jar:2.13.2:compile
[INFO] \- org.slf4j:slf4j-api:jar:1.7.36:compile
[INFO] +- org.springframework.boot:spring-boot-starter-test:jar:2.6.7:test
[INFO] |  +- org.springframework.boot:spring-boot-starter:jar:2.6.7:compile
[INFO] |  |  +- org.springframework.boot:spring-boot:jar:2.6.7:compile
[INFO] |  |  +- org.springframework.boot:spring-boot-autoconfigure:jar:2.6.7:compile
[INFO] |  |  +- org.springframework.boot:spring-boot-starter-logging:jar:2.6.7:compile
[INFO] |  |  |  +- ch.qos.logback:logback-classic:jar:1.2.11:compile
[INFO] |  |  |  |  \- ch.qos.logback:logback-core:jar:1.2.11:compile
[INFO] |  |  |  +- org.apache.logging.log4j:log4j-to-slf4j:jar:2.17.2:compile
[INFO] |  |  |  \- org.slf4j:jul-to-slf4j:jar:1.7.36:compile
[INFO] |  |  +- jakarta.annotation:jakarta.annotation-api:jar:1.3.5:compile
[INFO] |  |  \- org.yaml:snakeyaml:jar:1.29:compile
[INFO] |  +- org.springframework.boot:spring-boot-test:jar:2.6.7:test
[INFO] |  +- org.springframework.boot:spring-boot-test-autoconfigure:jar:2.6.7:test
[INFO] |  +- com.jayway.jsonpath:json-path:jar:2.6.0:test
[INFO] |  |  \- net.minidev:json-smart:jar:2.4.8:test
[INFO] |  |     \- net.minidev:accessors-smart:jar:2.4.8:test
[INFO] |  |        \- org.ow2.asm:asm:jar:9.1:test
[INFO] |  +- jakarta.xml.bind:jakarta.xml.bind-api:jar:2.3.3:compile
[INFO] |  |  \- jakarta.activation:jakarta.activation-api:jar:1.2.2:compile
[INFO] |  +- org.assertj:assertj-core:jar:3.21.0:test
[INFO] |  +- org.hamcrest:hamcrest:jar:2.2:test
[INFO] |  +- org.junit.jupiter:junit-jupiter:jar:5.8.2:test
[INFO] |  |  \- org.junit.jupiter:junit-jupiter-params:jar:5.8.2:test
[INFO] |  +- org.mockito:mockito-core:jar:4.0.0:test
[INFO] |  |  +- net.bytebuddy:byte-buddy-agent:jar:1.11.22:test
[INFO] |  |  \- org.objenesis:objenesis:jar:3.2:test
[INFO] |  +- org.mockito:mockito-junit-jupiter:jar:4.0.0:test
[INFO] |  +- org.skyscreamer:jsonassert:jar:1.5.0:test
[INFO] |  |  \- com.vaadin.external.google:android-json:jar:0.0.20131108.vaadin1:test
[INFO] |  +- org.springframework:spring-test:jar:5.3.19:test
[INFO] |  \- org.xmlunit:xmlunit-core:jar:2.8.4:test
[INFO] +- org.springframework.boot:spring-boot-starter-web:jar:2.6.7:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-json:jar:2.6.7:compile
[INFO] |  |  +- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:jar:2.13.2:compile
[INFO] |  |  +- com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.13.2:compile
[INFO] |  |  \- com.fasterxml.jackson.module:jackson-module-parameter-names:jar:2.13.2:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-tomcat:jar:2.6.7:compile
[INFO] |  |  +- org.apache.tomcat.embed:tomcat-embed-core:jar:9.0.62:compile
[INFO] |  |  +- org.apache.tomcat.embed:tomcat-embed-el:jar:9.0.62:compile
[INFO] |  |  \- org.apache.tomcat.embed:tomcat-embed-websocket:jar:9.0.62:compile
[INFO] |  +- org.springframework:spring-web:jar:5.3.19:compile
[INFO] |  \- org.springframework:spring-webmvc:jar:5.3.19:compile
[INFO] +- org.hibernate:hibernate-core:jar:5.6.3.Final:compile
[INFO] |  +- org.jboss.logging:jboss-logging:jar:3.4.3.Final:compile
[INFO] |  +- javax.persistence:javax.persistence-api:jar:2.2:compile
[INFO] |  +- net.bytebuddy:byte-buddy:jar:1.11.22:compile
[INFO] |  +- antlr:antlr:jar:2.7.7:compile
[INFO] |  +- org.jboss.spec.javax.transaction:jboss-transaction-api_1.2_spec:jar:1.1.1.Final:compile
[INFO] |  +- org.jboss:jandex:jar:2.2.3.Final:compile
[INFO] |  +- com.fasterxml:classmate:jar:1.5.1:compile
[INFO] |  +- javax.activation:javax.activation-api:jar:1.2.0:compile
[INFO] |  +- org.hibernate.common:hibernate-commons-annotations:jar:5.1.2.Final:compile
[INFO] |  +- javax.xml.bind:jaxb-api:jar:2.3.1:compile
[INFO] |  \- org.glassfish.jaxb:jaxb-runtime:jar:2.3.6:compile
[INFO] |     +- org.glassfish.jaxb:txw2:jar:2.3.6:compile
[INFO] |     +- com.sun.istack:istack-commons-runtime:jar:3.0.12:compile
[INFO] |     \- com.sun.activation:jakarta.activation:jar:1.2.2:runtime
[INFO] +- org.hibernate:hibernate-java8:jar:5.3.7.Final:compile
[INFO] +- org.springframework:spring-core:jar:5.3.18:compile
[INFO] |  \- org.springframework:spring-jcl:jar:5.3.19:compile
[INFO] +- javax.validation:validation-api:jar:2.0.1.Final:compile
[INFO] +- org.apache.commons:commons-lang3:jar:3.12.0:compile
[INFO] +- org.junit.jupiter:junit-jupiter-api:jar:5.8.2:test
[INFO] |  +- org.opentest4j:opentest4j:jar:1.2.0:test
[INFO] |  +- org.junit.platform:junit-platform-commons:jar:1.8.2:test
[INFO] |  \- org.apiguardian:apiguardian-api:jar:1.1.2:test
[INFO] +- org.junit.jupiter:junit-jupiter-engine:jar:5.8.2:test
[INFO] |  \- org.junit.platform:junit-platform-engine:jar:1.8.2:test
[INFO] +- org.postgresql:postgresql:jar:42.3.5:compile
```

```
[INFO] |  \- org.checkerframework:checker-qual:jar:3.5.0:runtime
[INFO] \- com.fasterxml.jackson.core:jackson-core:jar:2.13.3:compile
```

El segundo paso ha consistido en construir la estructura general de nuestra aplicación con el objetivo de tener una arquitectura distribuida en capas. Para ello se han empleado módulos Maven hijos del proyecto que se ha comentado en anterioridad. Cada uno de ellos, tiene sus propias dependencias, configuración y clases; así como las referencias entre ellos. La estructura general que se ha formado es la siguiente:

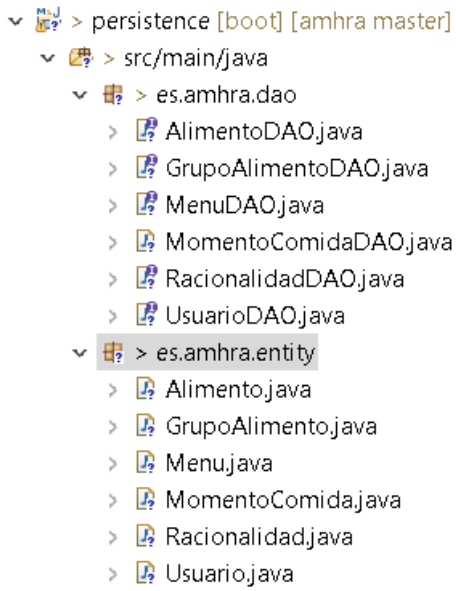
Ilustración 39. Proyecto base AMHRA



A continuación, se definen brevemente cada uno de los módulos y la función que tienen en el conjunto de la aplicación:

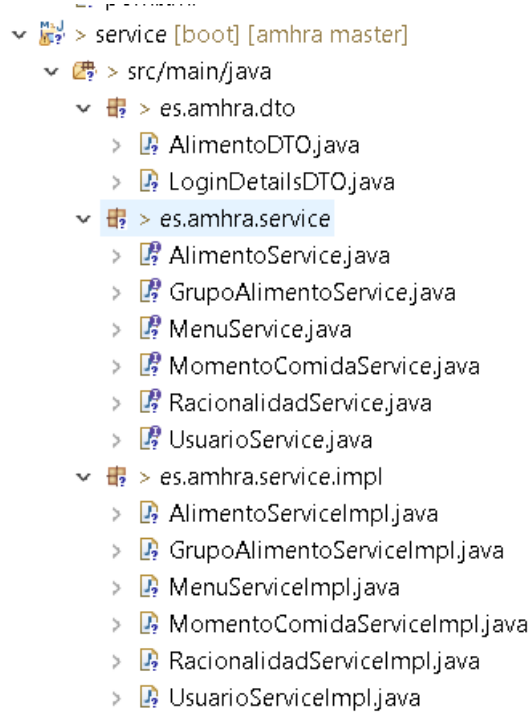
- Persistence: Hace referencia a la capa de persistencia en la que se almacenan las entidades, asociadas a cada una de las tablas de la base de datos, y los DAO, que se encargan de todas las operaciones en las que se interactúa con la base de datos.

Ilustración 40. Capa de persistencia AMHRA



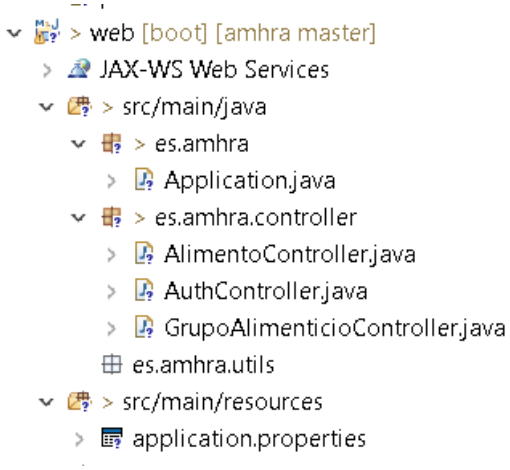
- Service: Esta capa actúa como mediadora entre la capa de persistencia y servicio y es donde se almacena la mayor parte de la lógica de la aplicación. Generalmente, también es donde almacenamos los DTO, objetos de transferencia entre capas, y los tratamos para realizar operaciones sobre ellos.

Ilustración 41. Capa de servicio AMHRA



- Web: En esta capa se almacenan los controladores REST de la aplicación, que se encargaran de gestionar las peticiones lanzadas por el *frontend*. También es donde se encuentra nuestra clase *main*, encargada de la ejecución del *backend* y las propiedades de acceso a nuestra base de datos.

Ilustración 42. Capa web AMHRA



Persistencia

Para poder trabajar con las entidades en la capa de persistencia se ha empleado Hibernate, que junto a las etiquetas de Spring Data JPA, conforman la mayor parte de la configuración de esta capa.

Por una parte, tenemos las entidades que son objetos persistentes que representan cada una de las tablas de nuestra base de datos. Por ejemplo, la siguiente entidad Alimento es la representación de la tabla alimento de la BBDD:

Ilustración 43. Ejemplo entidad Alimento

Alimento.java

```java

@Entity
@Table(name = "alimento")
public class Alimento implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_alimento")
    private Integer id;

    @Column(name = "name")
    private String name;

    @Column(name = "gramos")
    private Integer gramos;

    @Column(name = "indice_gluc")
    private Integer indiceGluc;

    @ManyToOne
    @JoinColumn(name = "id_grupo_alimento", referencedColumnName = "id")
    private GrupoAlimento grupoAlimenticio;

    @ManyToOne
    @JoinColumn(name = "id_usuario", referencedColumnName = "id_usuario")
    private Usuario usuario;

    public Alimento() {
    }

```

Esta es la entidad principal de nuestra aplicación y sobre la que más se trabajará a lo largo del desarrollo. Así pues, se procede a destacar algunos de los elementos más importantes, especialmente las etiquetas pertenecientes al paquete Jakarta Persistence, API para la gestión de la persistencia en el mapeo de objetos y relaciones.

- @Entity: Indica que esta clase se trata de una entidad.
- @Table: indica la tabla a la que hace referencia esta entidad.
- @Id: indica que este atributo es el identificador de la entidad/tabla.
- @GeneratedValue(Identity) implica la estrategia de generación automática de la clave primaria.
- @Column: Indica la columna a la que hace referencia este atributo
- @ManyToOne junto a @JoinColumn sirven para mapear la relación muchos a uno entre alimento y usuario/grupoAlimenticio.

Asimismo, la entidad tiene sus constructores, getter y setter que se han obviado en la imagen para facilitar la lectura y representación de esta.

Esta capa también se emplea el patrón de diseño DAO. De esta forma, se limita el acceso a los datos almacenados en la base de datos a los métodos almacenaos en estas clases. Normalmente, cuentan con una interfaz y la

implementación de la misma, pero en nuestro caso y, mediante el uso del repositorio JPARepository, una extensión de JPA específica para los repositorios. Gracias al uso de esta, se consigue reducir una gran cantidad de código que resulta repetitivo y poco legible para hacer operaciones CRUD.

Ilustración 44. Ejemplo DAO Alimento

Alimento.java | AlimentoDAO.java

```java
package es.amhra.dao;

import java.util.List;

@Repository
public interface AlimentoDAO extends JpaRepository<Alimento, Integer> {
    List<Alimento> findAllByUsuarioIn(Set<Usuario> usuarios);
    List<Alimento> findByGrupoAlimenticio(GrupoAlimento grupoAlimenticio);
    Alimento findByName(String name);

}
```

Service

Como se ha mencionado con anterioridad, esta capa se encarga de la mayor parte de la lógica de la aplicación y actúa como puente entre la capa de persistencia y la web. Para ello se implementan los servicios y sus implementaciones y los DTO, objetos que se encargan de almacenar y transportar los datos entre capas.

Las interfaces de los servicios son clases donde se declaran aquellos métodos que se van a desarrollar en la implementación de estos y es a partir de estas que el controlador accede a la mayor parte de la lógica del proyecto.

Ilustración 45. Ejemplo interfaz servicio Alimento

Alimento.java | AlimentoDAO.java | AlimentoService.java

```java
package es.amhra.service;

import java.util.List;

import es.amhra.dto.AlimentoDTO;
import es.amhra.entity.Alimento;

public interface AlimentoService {

    List<Alimento> getAllAlimentos();

    Alimento getAlimento(Integer id);

    Alimento updateAlimento(Alimento alimento);

    boolean deleteAlimento(int idAlimento, int idUsuario);

    Double getConversionGramosToRaciones(Integer idAlimento, Integer gramos);

    Double getConversionRacionesToGramos(Integer idAlimento, Integer raciones);

    List<Alimento> getAlimentos(int idUsuario, boolean checked);

    Alimento saveAlimento(AlimentoDTO alimentoDTO);

}
```

En las implementaciones de los servicios, tal como indica su nombre, se realiza la mayor parte del desarrollo de la aplicación. En estas clases se tratan los datos o información obtenida en el controlador mediante las peticiones y se procesa de la forma correspondiente. En nuestro caso, se ha querido mantener la lógica seguida tanto por las entidades y los controladores también y se ha creado un

servicio e implementación para las operaciones propias de cada una de las entidades. A continuación, se muestra como ejemplo aquella referente a Alimento.

Ilustración 46. Ejemplo implementación servicio Alimento

Alimento.java | AlimentoDAO.java | AlimentoService.java | AlimentoServiceImpl.java

```java
import es.amhra.entity.Alimento;
import es.amhra.entity.Usuario;
import es.amhra.service.AlimentoService;

@Service
public class AlimentoServiceImpl implements AlimentoService {

    @Autowired
    AlimentoDAO alimentoDAO;

    @Autowired
    UsuarioDAO usuarioDAO;


    @Override
    public List<Alimento> getAllAlimentos() {
        Set<Usuario> usuarios = findAdmin();
        if (usuarios != null) {
            return alimentoDAO.findAllByUsuarioIn(usuarios);
        }
        return null;
    }

    @Override
    public Alimento updateAlimento(Alimento alimento) {
        return alimentoDAO.save(alimento);
    }

    @Override
    public Alimento getAlimento(Integer id) {
        return alimentoDAO.findById(id).orElse(null);
    }

    @Override
    public boolean deleteAlimento(int idAlimento, int idUsuario) {
        Alimento alimento = alimentoDAO.findById(idAlimento).orElse(null);
        if (alimento != null && alimento.getUsuario().getId() == idUsuario)
```

Tal como se ha realizado con anterioridad, se analizan las etiquetas más destacas de la implementación:

- @Autowired: permite inyectar las dependencias necesarias.
- @Service: Indica que esta clase es un servicio.

Por otro lado, nos encontramos con los DTO, que suelen emplearse como objetos de transporte entre capas, sobre todo en el caso de aquellos objetos que nos interesa persistir. También nos permiten recoger datos del *frontend* que llegan con una estructura distinta a la que se espera, como en el siguiente caso donde se reciben los datos necesarios para guardar y actualizar un alimento creado por un usuario:

Ilustración 47. Ejemplo DTO Alimento

AlimentoDTO.java

```java
package es.amhra.dto;

import es.amhra.entity.GrupoAlimento;

public class AlimentoDTO {

    private String name;
    private Integer gramos;
    private Integer indiceGluc;
    private GrupoAlimento grupoAlimenticio;
    private Integer idUsuario;
    public AlimentoDTO() {
        super();
    }
    public AlimentoDTO(String name, Integer gramos, Integer indiceGluc, GrupoAlimento grupoAlimenticio,
            Integer idUsuario) {
        super();
        this.name = name;
        this.gramos = gramos;
        this.indiceGluc = indiceGluc;
        this.grupoAlimenticio = grupoAlimenticio;
        this.idUsuario = idUsuario;
    }

```

Web

Esta capa actúa como puente de acceso a nuestra aplicación ya que se encarga de recibir los datos del *frontend* y las peticiones que vienen con estos. En nuestro caso, nos encontramos ante cuatro métodos de petición HTTP:

- GET: El método GET es utilizado por aquellas peticiones cuyo fin es recuperar datos. Por ejemplo, es el caso de los métodos getAllAlimentos o getAlimentoById.

- POST: Las peticiones que emplean este método tienen generalmente el fin de cambiar el estado u ocasionar efectos secundarios en el servidor. Por ejemplo, es el caso del método saveAlimento.

- PUT: El método PUT se emplea cuando el objetivo es reemplazar las representaciones actuales de un recurso en el destino. Por ejemplo, es el caso de updateAlimento.

- DELETE: Las peticiones que emplean el método DELETE tienen como único objetivo borrar un recurso en específico. Es el caso de deleteAlimento();

En esta capa, a parte de los controladores que regulan el acceso a las peticiones, también tenemos el método *main* necesario para ejecutar la aplicación y el archivo recurso “application.properties” en el que se encuentra la configuración de nuestra base de datos.

Lo más destacable de nuestro método *main* es la etiqueta @SpringBootApplication, que realiza toda la configuración necesaria para poder ejecutar la aplicación. El proceso de limita a abrir el menú con *click* derecho sobre la clase Application y, al darle a run as Spring boot App, nuestra aplicación se desplegará en localhost:8080.

Ilustración 48. *Main* AMHRA

Application.java

```
1  package es.amhra;
2
3  import org.springframework.boot.SpringApplication;
5
6  @SpringBootApplication(scanBasePackages={"es.amhra"})
7  public class Application {
8      public static void main(String[] args) {
9      SpringApplication.run(Application.class, args);
10     }
11
12 }
13
```

La siguiente clase representa el controlador principal de nuestra aplicación. En ella se detallas las rutas mediante las que se podrá acceder a nuestra aplicación, el método de la petición y la respuesta y estado resultantes de cada una de ellas. Entre las etiquetas más relevantes, encontramos:

- @CrossOrigin: Determina que origines se permiten para realizar solicitudes HTTP de origen cruzado. En nuestro caso, localhost:3000 es donde se despliega el *frontend*.

- @RestController: Determina que esta clase se trata de un controlador REST.

- @RequestMapping: Determina la ruta de acceso a este controlador en concreto.

- @GetMapping, @PostMapping, @PutMapping y @RequestMapping(method.DELETE) determinan el método de la petición HTTP enviada desde el *frontend*.

A continuación, se muestra una parte de este controlador para poder observar dichas etiquetas:

Ilustración 49. Ejemplo alimento *controller*

Application.java | AlimentoController.java

```java
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import es.amhra.dto.AlimentoDTO;
import es.amhra.entity.Alimento;
import es.amhra.service.AlimentoService;

@CrossOrigin(origins = {"http://localhost:8080", "http://localhost:3000"})
@RestController
@RequestMapping("/api/v1/alimentos")
public class AlimentoController {

    @Autowired
    AlimentoService alimentoService;

    @GetMapping()
    public ResponseEntity<List<Alimento>> getAllAlimentos() {
        return ResponseEntity.ok(alimentoService.getAllAlimentos());
    }

    @GetMapping("loged")
    public ResponseEntity<List<Alimento>> getAllAlimentos(@RequestParam int idUsuario, @RequestParam boolean checked) {

        return ResponseEntity.ok( alimentoService.getAlimentos(idUsuario, checked));
    }

    @RequestMapping(value = "delete", method = RequestMethod.DELETE)
    @ResponseBody
    public ResponseEntity<String> deleteAlimento(@RequestParam Integer idAlimento, @RequestParam Integer idUsuario) {

        boolean borrado = alimentoService.deleteAlimento(idAlimento, idUsuario);
        if (borrado) {
            return ResponseEntity.ok("Alimento borrado correctamento");
        }
        return ResponseEntity.badRequest().build();

    }

```

2. ReactJs *Frontend*

El *frontend* se ha implementado con el uso de la librería de código abierto de Javascript, ReactJs. Esta ha sido desarrollada por Facebook con el objetivo de simplificar la creación de las “Single Page Applications” y facilitar la tarea de desarrollar interfaces de usuario. El paradigma que emplea es el de la programación orientada a componentes.

Adicionalmente, el enfoque que se le ha dado en el caso de AMHRA es con el uso Hooks. De esta forma, se puede extraer lógica de estado de un componente de tal forma que este pueda ser probado y vuelto a usar independientemente, guardar el estado y proporcionar un código visualmente más limpio y entendible. Gracias a este enfoque, se pueden reutilizar en multitud de componentes y/o funciones.

Los primeros pasos realizados para la creación y configuración del *frontend* de AMHRA han consistido en la instalación de Node.js, un entorno en tiempo de ejecución de código abierto basado en el lenguaje de programación JavaScript y en el motor V8 de Google.

De esta forma, tenemos acceso al uso de NPM (“Node Package Manager”). Este gestor de paquetes nos permite obtener cualquier librería de forma simple, lo cual nos permitirá agregar dependencias de rápidamente, distribuir paquetes y administrar eficazmente tanto los módulos como el proyecto a desarrollar en general. Por ejemplo, instalamos create-react-app con este gestor:

```
nmp install create-react-app
```

Para crear el proyecto base sobre el que se desarrollará el *frontend*, se emplea el siguiente comando:

```
npx create-react-app amhra-frontend
```

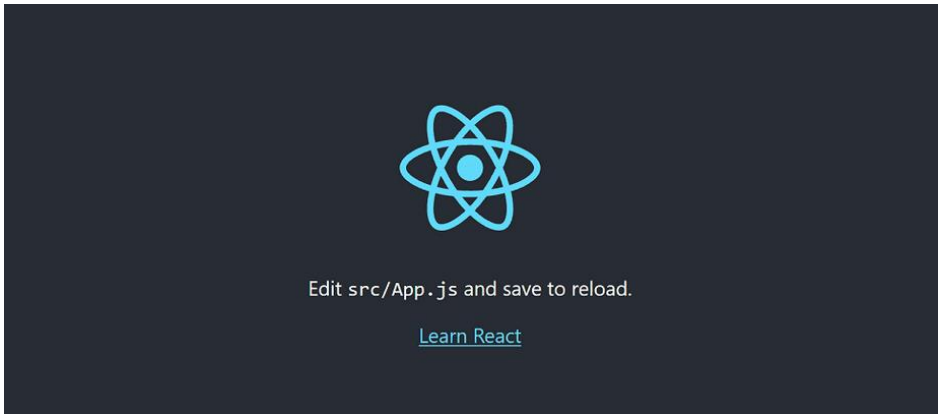
Entonces, se nos generará nuestra aplicación “amhra-frontend” con una estructura similar a esta:

Ilustración 50. Ejemplo proyecto base

En este punto ya se dispone de un proyecto capaz de desplegarse, pero solamente seremos capaces de ver el componente de bienvenida previamente creado por React. Por defecto, y mediante el uso del siguiente comando, el *frontend* se muestra en el localhost:3000.

```
npm start
```

Ilustración 51. Página de bienvenida React



Para poder empezar a personalizar este proyecto base, primero se eliminan algunos de los ficheros que venían por defecto, como los iconos o CSS que no requiramos reutilizar.

Posteriormente, se instalan las librerías de componentes de PrimeReact y MUI, a partir de las cuales se basarán los componentes que conformarán nuestra interfaz de usuario. Para ello, simplemente debemos aplicar el comando npm install o npm add, como se ha hecho con anterioridad.

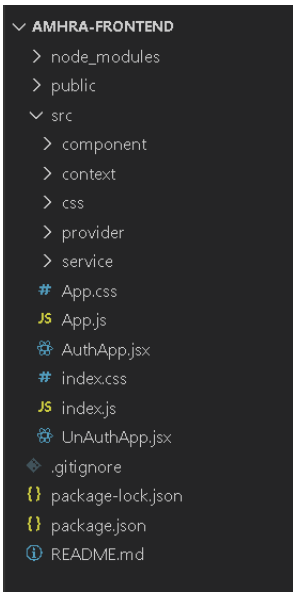
El siguiente paso consiste en definir el tema general de nuestra aplicación: fuentes, colores primarios y secundarios, modo oscuro o claro… Aunque, posteriormente, requeriremos componentes que se salgan de estos cánones, simplificará y reducirá la cantidad de hojas de estilo necesarias. Para esto empleamos el "Material-UI Theme Creator", copiamos las opciones que se generan y las pegamos en nuestro index.js y definimos un *ThemeProvider* que encapsule toda la aplicación:

Ilustración 52. index.js

```
src > JS index.js > ...
1   import React from 'react';
2   import ReactDOM from 'react-dom/client';
3   import './index.css';
4   import App from './App';
5   import { ThemeProvider, createTheme } from '@mui/material/styles';
6   import { BrowserRouter} from "react-router-dom";
7   import { UserProvider } from './provider/UserProvider';
8
9   let theme = createTheme({
10    palette: {
11      type: 'light',
12      primary: {
13        main: '#084309',
14      },
15      secondary: {
16        main: '#f50057',
17      },
18    },
19  });
20
21  const root = ReactDOM.createRoot(document.getElementById('root'));
22  root.render(
23    <React.StrictMode>
24      <ThemeProvider theme={theme}>
25      <UserProvider>
26        <BrowserRouter>
27          <App />
28        </BrowserRouter>
29      </UserProvider>
30      </ThemeProvider>
31
32    </React.StrictMode >
33  );
```

Una vez configurado el proyecto base y el tema que queremos que envuelva la interfaz, el siguiente paso es crear todos aquellos componentes, servicios y realizar las configuraciones oportunas para obtener nuestros objetivos. Tras realizar esto, contamos con el proyecto que se muestra a continuación, el cual se va a analizar con mayor profundidad por carpetas y ficheros:

Ilustración 53. *Frontend* AMHRA proyecto



Carpetas

Src

Esta carpeta es la raíz donde se encuentran todos los componentes organizados en distintas carpetas; pero también tiene index.js, que ya hemos visto anteriormente y actúa como punto de entrada al resto de la aplicación, y los componentes App, AuthApp y UnAuthApp.

App

Este componente se ha empleado para proporcionar acceso a unas partes de la aplicación u otras según si el usuario ha iniciado o no sesión en la aplicación. Tal como sus nombres indican, se renderiza AuthApp en caso de haber iniciado sesión y UnAuthApp en caso contrario.

Ilustración 54. App.js

```
src > JS App.js > ...
  1   import './App.css';
  2
  3   import "primereact/resources/themes/saga-green/theme.css";  //theme
  4   import "primereact/resources/primereact.min.css";                  //core css
  5   import "primeicons/primeicons.css";                                //icons
  6
  7
  8   import React, { useContext } from 'react';
  9   import AuthApp from './AuthApp';
 10   import UnAuthApp from './UnAuthApp';
 11   import { UserContext } from './context/UserContext';
 12
 13
 14   function App() {
 15     const { user } = useContext(UserContext);
 16     return user.auth ? <AuthApp /> : <UnAuthApp />;
 17
 18   }
 19
 20   export default App;
 21
```

AuthApp

Esta parte de la aplicación es la que cuenta con la mayor cantidad de funcionalidades, correspondientes a los requisitos del proyecto.

Ilustración 55. AuthApp.js

```
import React from 'react';
import {
  Routes,
  Route,
} from "react-router-dom";

import ListAlimentoComponent from './component/ListAlimentoComponent';
import CalculoComponent from './component/CalculoComponent';
import IndexComponent from './component/IndexComponent';
import EditAlimentoComponent from './component/EditAlimentoComponent';
import { AddAlimentoComponent } from './component/AddAlimentoComponent';

function AuthApp() {

  return (
    <div>
      <IndexComponent />
      <br></br>
      <Routes>
        <Route path="/" element={<ListAlimentoComponent />} />
        <Route path="alimentos" element={<ListAlimentoComponent />} />
        <Route path="calculo" element={<CalculoComponent />} />
        <Route path="edit" element={<EditAlimentoComponent />} />
        <Route path="add" element={<AddAlimentoComponent />} />
        <Route index element={<ListAlimentoComponent />} />
      </Routes>
    </div>

  );
}
```

Como se puede observar en el código, la navegación a través de todas estas funciones se controla con Routes y podremos acceder a la mayoría de ellas a través del componente IndexComponent. Aquellas funciones encargadas de editar, borrar y añadir un alimento solo se pueden acceder a través del icono de edición y borrado de la tabla de alimentos, en el caso de los primeros y a través del botón de añadir alimento en la misma.

Ilustración 56. IndexComponent (vista)

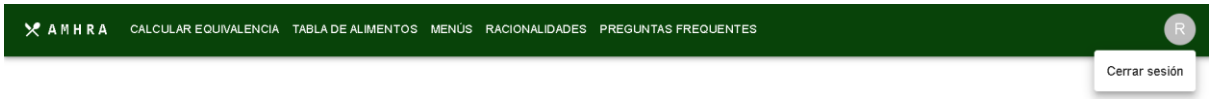
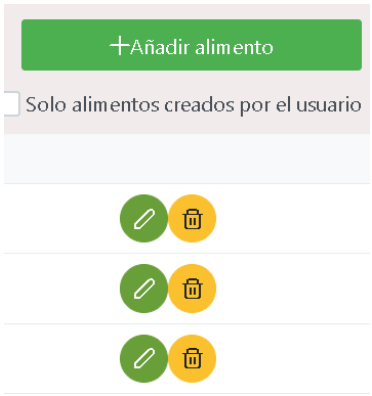


Ilustración 57. Botones añadir, eliminar, editar



UnAuthApp

Esta parte de la aplicación es la que cuenta con menos funciones, pero se añade la opción de registro e inicio de sesión:

Ilustración 58. UnAuthComponent

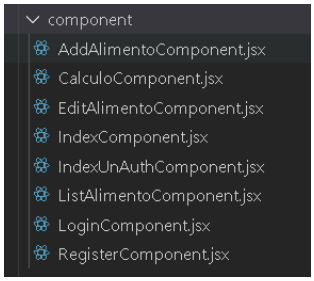
```
import ListAlimentoComponent from './component/ListAlimentoComponent';
import CalculoComponent from './component/CalculoComponent';
import { RegisterComponent } from './component/RegisterComponent';
import { LoginComponent } from './component/LoginComponent';
import IndexUnAuthComponent from './component/IndexUnAuthComponent';

function UnAuthApp() {

  return (
    <div>
      <IndexUnAuthComponent />
      <br></br>
      <Routes>
        <Route path="/" element={<ListAlimentoComponent />} />
        <Route path="alimentos" element={<ListAlimentoComponent />} />
        <Route path="calculo" element={<CalculoComponent />} />
        <Route path="register" element={<RegisterComponent />} />
        <Route path="login" element={<LoginComponent />} />
        <Route index element={<ListAlimentoComponent />} />
      </Routes>
    </div>

  );
}

export default UnAuthApp;
```

Ilustración 59. UnAuthIndexComponent

Components

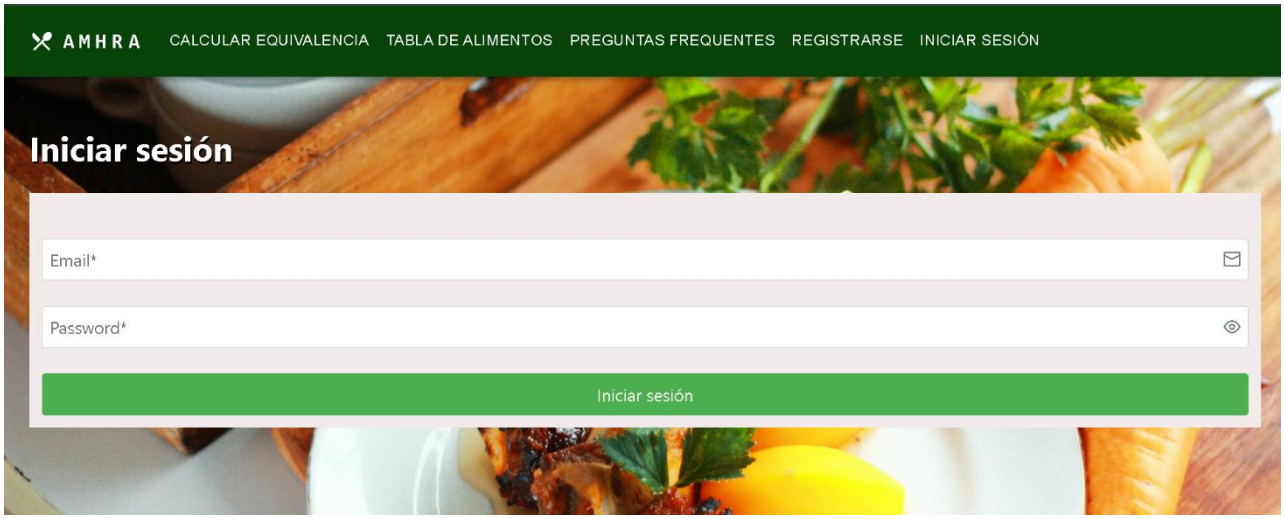
Esta carpeta actúa como contenedor de todas aquellos componentes que representan las opciones de la aplicación y son las encargadas de contener el código necesario para el correcto funcionamiento de la interfaz.

Ilustración 60. Carpeta Component



Por ejemplo, el componente “LoginComponent” , hace referencia a la siguiente parte del *frontend*:

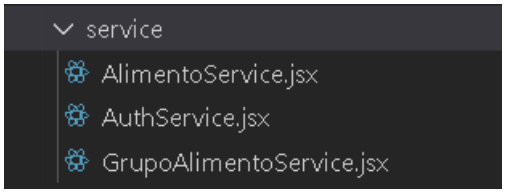
Ilustración 61. Iniciar sesión vista



Service

Aquí se almacenan todos aquellos archivos que actúan como servicio, y, por lo tanto, los encargados de gestionar las peticiones de la aplicación al *backend*.

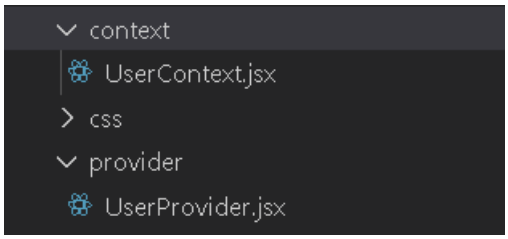
Ilustración 62. Carpeta service



Context y provider

Estas dos carpetas almacenan aquellos ficheros encargados de mantener la sesión del usuario en el contexto.

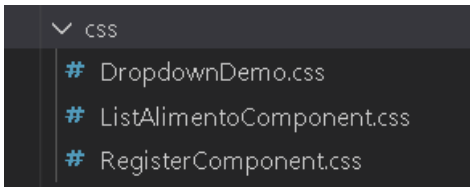
Ilustración 63. Carpetas context y provider



CSS

Aquí se incluyen todas las hojas de estilo empleadas en la aplicación.

Ilustración 64. Carpeta CSS



Diseño responsive

Uno de los requisitos no funcionales de la aplicación era que se adaptara a las pantallas de distintos tamaños, especialmente aquellas más pequeñas como las de móvil.

Para poder ofrecer esta opción, se han configurado los componentes de MUI y PrimeReact para que se mostraran de forma distinta según la necesidad. A continuación, se presentan algunos ejemplos de cómo se ve en una pantalla de ordenador y una pantalla típica de móvil.

Menú

- Móvil

Ilustración 65. Menú vista móvil

- Ordenador

Ilustración 66. Menú vista ordenador

Tabla de alimentos

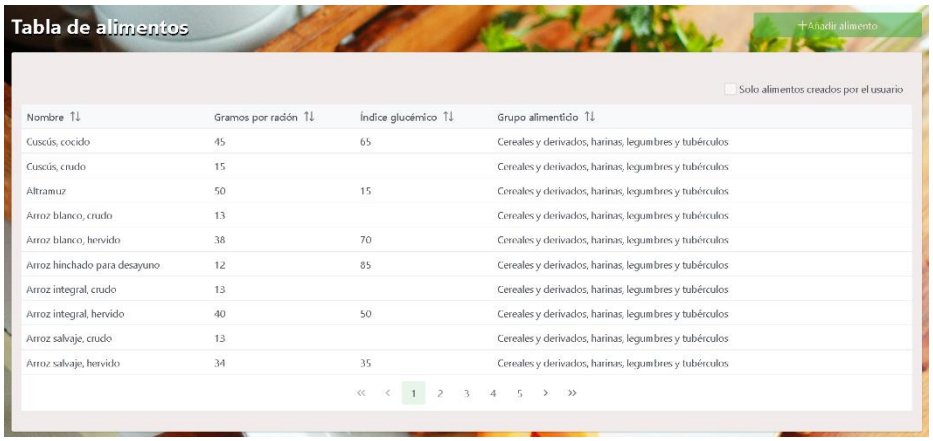
Además de adaptar la tabla a la pantalla, esta también cuenta con su contenido paginado. De esta forma, el usuario no tendrá que desplazarse infinitamente por la pantalla en caso de escalar y que aumente notablemente el número de alimentos.

- Móvil

Ilustración 67. Tabla alimento vista móvil

- Ordenador

Ilustración 68. Tabla alimento vista ordenador



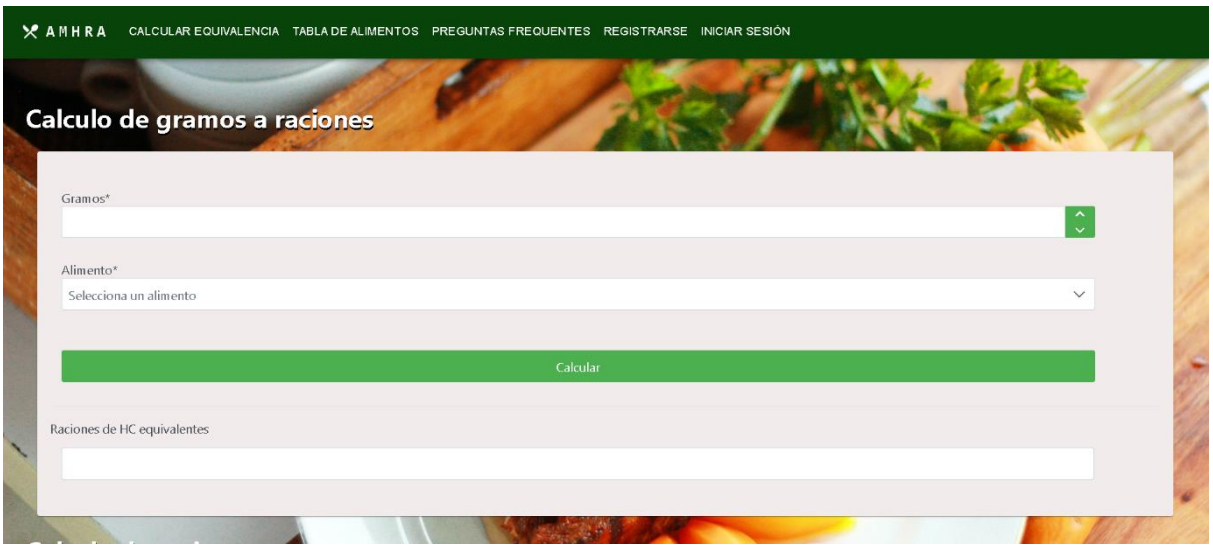
Consulta equivalencias

- Móvil

Ilustración 69. Equivalencia móvil

- Ordenador

Ilustración 70. Equivalencia ordenador



3. Base de datos

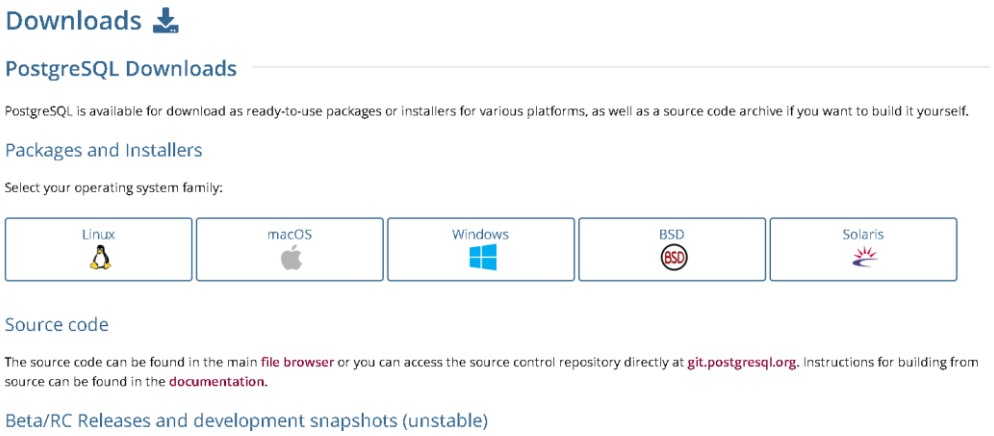
La base de datos son los cimientos de nuestra aplicación pues es donde se van a almacenar todos los datos y los cambios acontecidos en estos. En ella, las entidades se asocian con tablas y los atributos como columnas.

En nuestro caso, se ha escogido el uso de PostgreSQL como base de datos y, por lo tanto, se ha procedido a crear la conexión y la base de datos pertinente.

El primer paso ha consistido en descargar e instalar la última versión de PostgreSQL para Windows desde la página oficial. En este proceso de

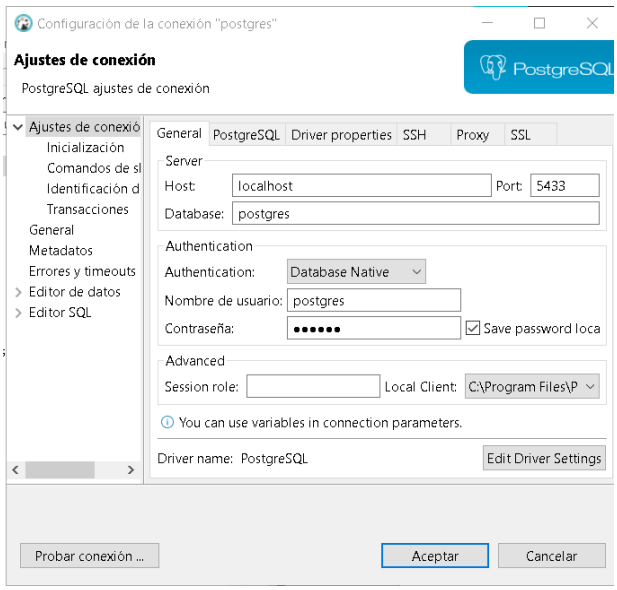
instalación se han definido el directorio, la contraseña para el superusuario “postgres” y el puerto sobre el que el server debe escuchar.

Ilustración 71. Página de descarga PostgreSQL



Posteriormente, se ha utilizado PGAdmin para crear la base de datos y se ha conectado en DBeaver para la correcta creación de las tablas, columnas y restricciones.

Ilustración 72. Conexión a la base de datos Dbeaver



El último paso ha requerido poblar la base de datos con los datos de los alimentos, grupos, momentos del día y, en general, todas las tablas. En el caso de usuarios, se ha definido un usuario administrador para asignar todos aquellos alimentos que han de venir por defecto en la base de datos. También se ha creado un usuario de prueba "aaa" que será sobre el cuál se realizarán las pruebas oportunas.

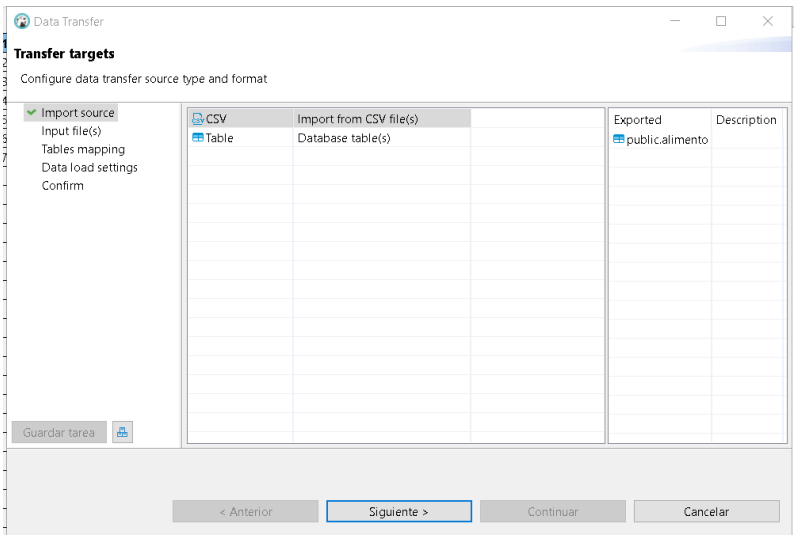
Ilustración 73. Usuarios de prueba

1	admin	admin@gmail.com	123456
2	aaa	a@gmail.com	1234

En cuanto a los alimentos, los datos se han importado siguiendo estos pasos:

1. Mediante Excel, importamos todas las tablas de alimentos disponibles en "Fundación para la Diabetes"
2. Convertimos los datos para adaptarnos al formato CSV.
3. Importamos los datos en base de datos con el uso de la opción *import* dándole *click* derecho sobre la tabla.

Ilustración 74. Interfaz de importar datos



7. Pruebas

En este apartado se definen las pruebas no automatizadas realizadas sobre la aplicación. La decisión de no implementar pruebas automatizadas ha residido en la limitación temporal con la que se cuenta para implementar y probar este proyecto. Tras valorar el escenario en conjunto, se ha decidido priorizar la implementación de los requisitos funcionales y no funcionales por encima de dichas pruebas.

Aun así, se ha tratado de especificar en detalle cada una de las pruebas que se realizan a lo largo del proceso de desarrollo. Estas vienen detalladas según el identificador del requisito al que hacen referencia.

1. Pruebas no automatizadas

Requisitos funcionales

Ilustración 75. Tabla de pruebas de requisitos funcionales de Usuario no Registrado

Usuario no registrado			
Identificador	**Descripción**	**Resultado esperado**	**Estado**
RF-UN01	Registrarse con nombre, email y contraseña correctos.	Se da de alta al usuario con los datos introducidos.	**OK**
RF-UN01	Registrarse con algún campo obligatorio sin introducir	El formulario muestra un error pidiendo que se introduzcan.	**OK**
RF-UN01	Registrarse con algún campo en un formato incorrecto	El formulario muestra un error con el formato adecuado.	**OK**
RF-UN01	Registrarse con una cuenta ya existente.	Se muestra un error.	**OK**

Ilustración 76. Tabla de pruebas de requisitos funcionales de Usuario Registrado

Usuario registrado			
Identificador	**Descripción**	**Resultado esperado**	
RF-UR01	Añadir un alimento nuevo con todos los campos	Se guarda el alimento con los datos introducidos.	**OK**
RF-UR01	Añadir un alimento nuevo con todos los campos, menos índice glucémico	Se guarda el alimento con los datos introducidos.	**OK**
RF-UR01	Añadir un alimento nuevo con algún campo obligatorio faltante	El formulario muestra un error pidiendo que se introduzcan.	**OK**
RF-UR02	Modificar un alimento creado por el usuario con todos los campos obligatorios introducidos	El alimento se actualiza correctamente.	**OK**

RF-UR02	Modificar un alimento creado por el administrador	La opción no es posible ya que solo se muestran los botones de editar y eliminar para alimentos creados por el usuario.	**OK**
RF-UR02	Modificar un alimento creado por el usuario con algún campo obligatorio faltante.	El formulario muestra un error pidiendo que se introduzcan.	**OK**
RF-UR03	Eliminar alimento sin iniciar sesión	La opción no es posible ya que se controla que aparezca solamente al haber iniciado sesión.	**OK**
RF-UR03	Eliminar alimento creado por el usuario	Se elimina correctamente el alimento de la base de datos.	**OK**
RF-UR03	Eliminar alimento creado por el administrador	La opción no es posible ya que solo se muestran los botones de editar y eliminar para alimentos creados por el usuario.	**OK**
RF-UR04	Cerrar sesión tras haber iniciado sesión con anterioridad	Se cierra sesión correctamente y se eliminan los datos de dicha sesión. Se redirecciona a la página principal.	**OK**
RF-UR04	Cerrar sesión sin haber iniciado sesión con anterioridad.	La opción no es posible ya que se controla que aparezca solamente al haber iniciado sesión.	**OK**
RF-UR05	Crear menú con todos los datos obligatorios	Se guarda el menú correctamente	**OK**
RF-UR05	Crear menú con algún dato obligatorio faltante	Se muestra un error en el que se indica que no dispone de los permisos necesarios.	**OK**
RF-UR06	Modificar menú con todos los datos obligatorios	Se actualizan los datos del menú correctamente.	**OK**
RF-UR06	Modificar menú con algún dato obligatorio faltante	No es posible, el único dato a modificar que se muestra es la lista de alimentos, que se recuperan de la base de datos, y el usuario puede desear no seleccionar ninguno.	**OK**
RF-UR06	Modificar menú que no existe	La opción no es posible ya que la opción solo se muestra para los menús ya existentes.	**OK**
RF-UR07	Eliminar menú sin haber iniciado sesión	La opción no es posible ya que se controla que	**OK**

		aparezca solamente al haber iniciado sesión.	
RF-UR07	Eliminar menú del usuario con sesión iniciada	El menú se elimina correctamente	**OK**
RF-UR08	Añadir racionalidad por comida con todos los datos introducidos	Se añade la racionalidad correctamente.	**OK**
RF-UR08	Añadir racionalidad por comida con algún dato faltante	Se muestra un error pidiendo que se introduzcan los datos.	**OK**
RF-UR09	Modificar racionalidad por comida con todos los datos introducidos	Se actualiza la racionalidad correctamente.	**OK**
RF-UR09	Modificar racionalidad por comida con algún dato faltante	Por defecto, el único campo modificable son el número de raciones y se establece el 0 como valor predeterminado. Por lo tanto, aunque esté vacío se establecerá a 0.	**OK**
RF-UR10	Eliminar racionalidad por comida con sesión iniciada	Se elimina la racionalidad correctamente.	**OK**
RF-UR10	Eliminar racionalidad por comida sin iniciar sesión	La opción no es posible ya que se controla que aparezca solamente al haber iniciado sesión.	**OK**
RF-UR11	Iniciar sesión con email y contraseña correctos.	Se inicia sesión y se guarda la sesión correctamente con los datos introducidos.	**OK**
RF-UR11	Iniciar sesión con algún campo obligatorio sin introducir	El formulario muestra un error pidiendo que se introduzcan.	**OK**
RF-UR11	Iniciar sesión con algún campo incorrecto.	El formulario muestra un error.	**OK**

Ilustración 77. Tabla de pruebas de requisitos funcionales de todos los usuarios

Todos los usuarios			
Identificador	**Descripción**	**Resultado esperado**	
RF-T01	Consultar la tabla de alimentos sin haber iniciado sesión	Se muestra correctamente la lista de forma paginada, solamente mostrando alimentos introducidos por el administrador.	**OK**
RF-T01	Consultar la tabla de alimentos con sesión iniciada	Se muestra correctamente la lista de forma paginada, mostrando alimentos introducidos por el	**OK**

		administrador y el propio usuario.	
RF-T02	Filtrar tabla por alimentos creados por el usuario.	Se muestran correctamente los alimentos creados por el usuario.	**OK**
RF-T02	Ordenar tabla alfabéticamente por nombre del alimento.	Se muestran correctamente los alimentos ordenados.	**OK**
RF-T02	Ordenar tabla de menor a mayor o viceversa por gramos del alimento.	Se muestran correctamente los alimentos ordenados.	**OK**
RF-T02	Ordenar tabla de menor a mayor o viceversa por índice de glucosa del alimento.	Se muestran correctamente los alimentos ordenados.	**OK**
RF-T02	Ordenar tabla alfabéticamente por grupo alimenticio del alimento.	Se muestran correctamente los alimentos ordenados.	**OK**
RF-T03	Consultar preguntas frecuentes	Se muestran correctamente las preguntas y respuestas frecuentes ordenadas en pestañas.	**OK**
RF-T04	Cálculo de gramos dados el alimento y las raciones	Se muestra correctamente el resultado (gramos) con los decimales establecidos.	**OK**
RF-T04	Cálculo de raciones dados el alimento y los gramos	Se muestra correctamente el resultado (raciones) con los decimales establecidos.	**OK**
RF-T04	Cálculo de raciones con un alimento inexistente.	Controlado con un *select* con todos los alimentos disponibles en el que se puede hacer una búsqueda por nombre.	**OK**
RF-T04	Cálculo de raciones sin introducir los gramos.	Se muestra un error indicando que se introduzcan los datos faltantes.	**OK**
RF-T04	Cálculo de gramos sin introducir las raciones.	Se muestra un error indicando que se introduzcan los datos faltantes.	**OK**

Requisitos no funcionales

Ilustración 78. Tabla de pruebas de requisitos no funcionales

Identificador	Descripción	Resultado esperado	
RNF - 01	El sistema debe poder utilizarse en los navegadores más utilizados	El sistema se puede visualizar correctamente en Google Chrome, Mozilla Firefox y Opera.	OK
RNF - 02	El sistema debe adaptarse a la pantalla del móvil.	Se han adaptado a la pantalla aquellos elementos más problemáticos como el menú, la lista de alimentos, cálculo de equivalencias…	OK

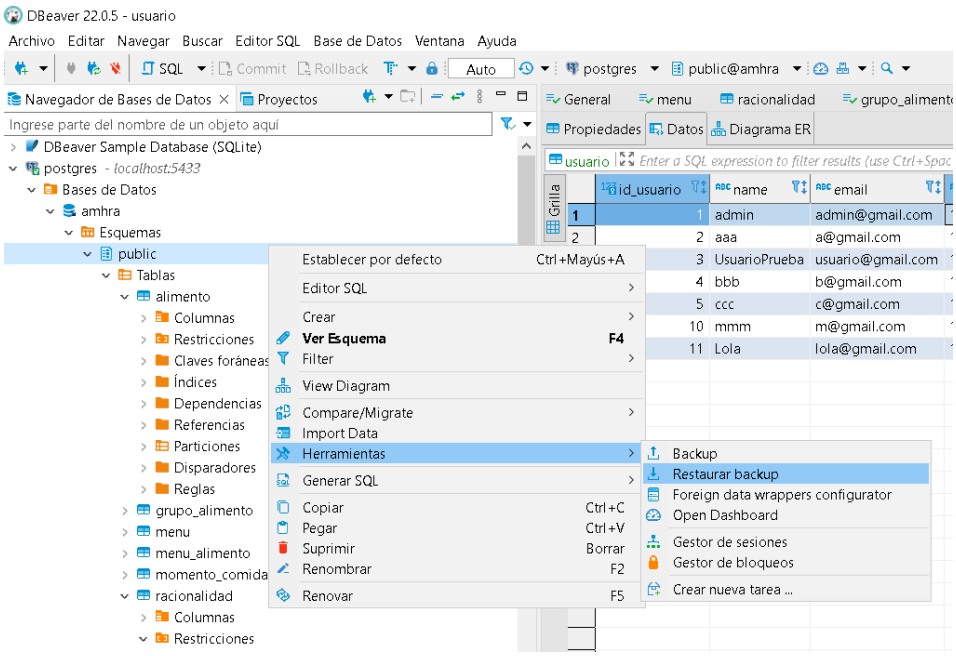
2. Manual de pruebas

El manual de pruebas se proporciona para poder reproducir las pruebas realizadas para asegurar el correcto funcionamiento del proyecto. En ella se proporcionarán los pasos requeridos para la puesta en marcha del *frontend*, el *backend* y la base de datos.

Base de datos

El primer paso consistirá en restaurar la base de datos a partir del script que se proporciona junto a este proyecto. Para ello, se utiliza la opción restaurar *backup* tal como se indica en la siguiente imagen:

Ilustración 79. Restaurar backup



Para comprobar que se ha restaurado la base de datos correctamente, recomendamos hacer clic sobre “public” y comparar el diagrama entidad – relación con el proporcionado en el apartado 3.

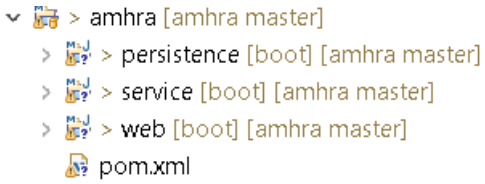
Backend

Las pruebas del *backend* consisten en probar con el uso de una herramienta como Postman que los puntos de acceso hacen su función correctamente y devuelven aquello que se espera.

El primer paso para poder probarlo consiste en importar el proyecto “amhra” en el IDE de su elección. En nuestro caso se emplea Eclipse. Para ello, seleccionamos en el menú superior “File > Import >Existing Maven Projects” y seleccionamos el directorio donde tenemos guardado nuestro proyecto.

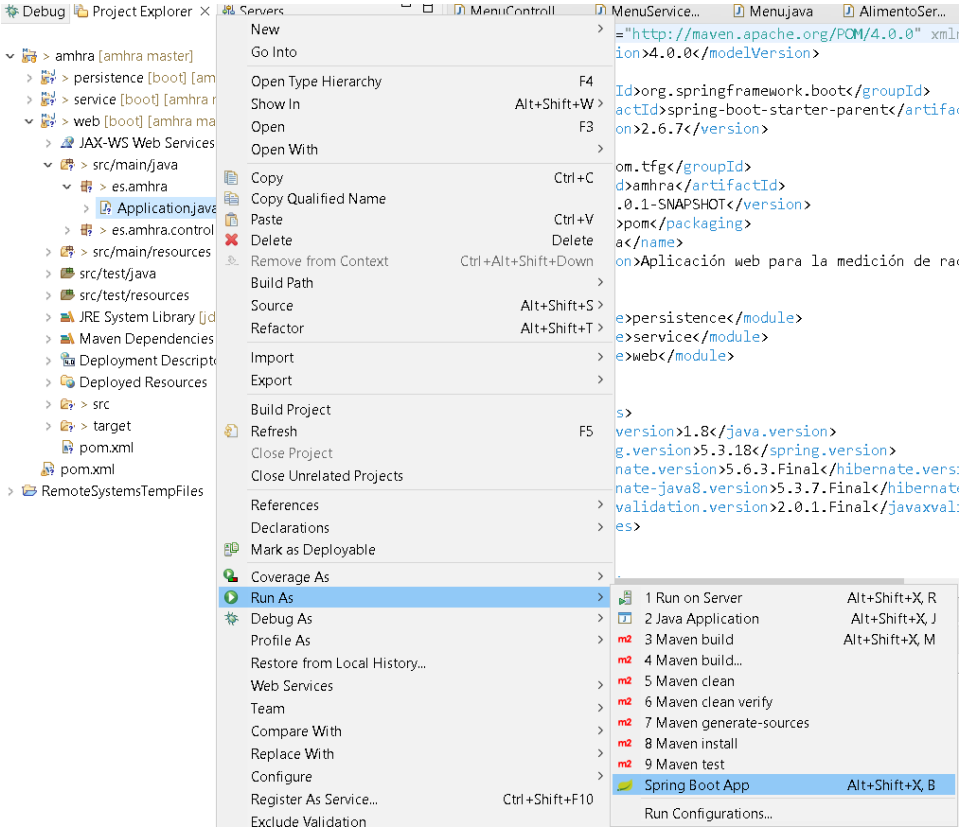
En este punto, y si desplegamos nuestro proyecto, debemos tener algo así:

Ilustración 80. Estructura AMHRA import



Para poder ejecutar nuestra aplicación desplegamos la carpeta web y buscamos el paquete raíz “es.amhra”, donde estará almacenado nuestro ejecutable “Application.java”. Le damos click derecho > Run As > Spring Boot App.

Ilustración 81. Ejecución backend



Entonces tendremos la aplicación funcionando en el puerto 8080 y podremos hacer nuestras primeras comprobaciones con Postman. Los puntos de acceso a la aplicación son:

- Alimento: “`/api/v1/alimentos`”
  - Recuperar todos los alimentos: GET “”
  - Recuperar todos los alimentos tras iniciar sesión: GET “`/loged/`”
    - QueryParams: idUsuario, checked.
  - Eliminar alimento: DELETE “`/delete`”
    - QueryParams: idAlimento, idUsuario.
  - Guardar o actualizar alimento: POST “`/save`”
    - Body: AlimentoDTO
  - Conversión gramos a raciones: GET “`/conversionGramos`”
    - QueryParams: idAlimento, gramos.
  - Conversión raciones a gramos: GET `"/conversionRaciones"`
    - QueryParams: idAlimento, raciones.
- Auth: “`/api/v1/auth`”
  - Registrar usuario: POST “”
    - Body: UsuarioDTO
  - Iniciar sesión: GET “”
    - QueryParams: email, password.
- Grupo Alimenticio: “`/api/v1/grupos`”
  - Recuperar todos los grupos: GET “”
- Menú: “`/api/v1/menus`”
  - Recuperar resumen de todos los menús: GET “”
    - QueryParams: idUsuario.
  - Recuperar menú: GET “`/menu`”
    - QueryParams: idMenu.
  - Eliminar menú: DELETE “`/delete`”
    - QueryParams: idMenu, idUsuario.
  - Guardar o actualizar menú: POST “`/save`”
    - Body: MenuDTO
- Grupo Alimenticio: “`/api/v1/momentos`”
  - Recuperar todos los momentos: GET “”

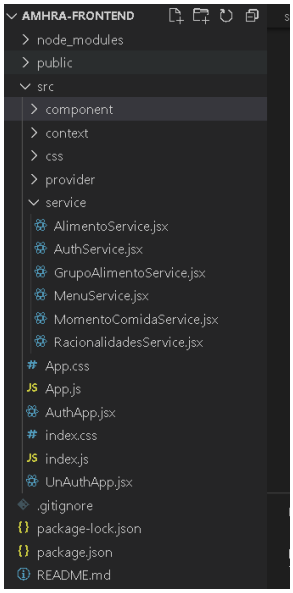
- Racionalidades: “`/api/v1/racionalidades`”
    - Recuperar todas las racionalidades: GET “”
        - QueryParams: idUsuario.
    - Eliminar racionalidad: DELETE “`/delete`”
        - QueryParams: idRacionalidad, idUsuario.
    - Guardar o actualizar alimento: POST “`/save`”
        - Body: RacionalidadDTO

Frontend

Las pruebas del *frontend* consisten en probar mediante el uso de la interfaz que todas las funcionalidades se comportan como corresponden. Estas pruebas están detalladas en pruebas no automatizadas

El primer paso para poder probarlo consiste en importar el proyecto “amhra-frontend” en el IDE de su elección. En nuestro caso se emplea Visual Studio Code. Para ello, seleccionamos en el menú superior Archivo > Abrir carpeta y seleccionamos el directorio donde tenemos guardado nuestro proyecto. Nos debe quedar la siguiente estructura:

Ilustración 82. Estructura prueba AMHRA *frontend*



Posteriormente, en el menú superior seleccionamos Terminal > Nuevo Terminal e introducimos el comando “npm start” sin las comillas. Este comando ejecutará nuestra aplicación en el puerto 3000 y nos abrirá una nueva pestaña en nuestro navegador predeterminado con la raíz del proyecto.

Cabe destacar que para comprobar que funciona correctamente, es necesario tener en funcionamiento el *backend*. Por lo tanto, se deben seguir los pasos definidos en el apartado “Backend”

8. Cambios respecto a la planificación inicial

Debido a diversos problemas personales, se ha tenido que realizar una revalorización exhaustiva del plan inicial. La situación acontecida entra dentro de la evaluación de riesgos, específicamente los aparados:

- R2 – Situaciones personales imprevistas: A lo largo del mes y medio en el que transcurre la PEC3 han ocurrido diversas situaciones que han sido consecuencia del azar como una nueva situación laboral en la que solo puedo dedicar los fines de semana a mis asignaturas universitarias y el resultar enferma durante uno de ellos. Estas situaciones han sido dos de las más decisivas en cuanto a la decisión de revalorizar el productor final.

- R4 - Consecuencias derivadas de la falta de experiencia. Debido a la falta de experiencia en Javascript y React, he encontrado numerosos problemas que han requerido bastantes horas de investigación y pruebas que no fueron contempladas en la planificación inicial a esta extensión.

- R5 - Problemas con el equipo de trabajo. Pese a ser la probabilidad más remota de entre las contempladas, mi ordenador personal se estropeó, perdí parte del trabajo ya realizado y lo tuve que mandar a la garantía ocasionando otro atraso con respecto a la planificación inicial.

Tal como se valoró en el apartado “Evaluación de riesgos”, se ha aplicado la acción A2 en la que se ha seguido el orden de prioridad de los requisitos hasta llegar al punto definido en las “Pruebas”, las cuales han sufrido un drástico recorte de tiempo de los 23 días iniciales a los 2 actuales. Asimismo, el inicio de sesión y registro se han simplificado (evitando comprobaciones de seguridad complejas).

9. Conclusiones

El realizar un proyecto desde la nada hasta el punto definido en esta memoria ha sido un reto a nivel personal, ya que resulta más complicado de lo que parece a primera vista.

Algunas competencias como la capacidad de planificación y organización son sumamente importantes para poder llevar a cabo el proyecto, ya que te puedes encontrar con multitud de situaciones que requieran que saques lo mejor de ti para llevar el proyecto adelante.

No obstante, y pese a las múltiples trabas que me he ido encontrando, ha resultado un proceso sumamente satisfactorio ya que te permite aplicar todos los conocimientos que has ido adquiriendo a lo largo de la carrera e incluso adquirir nuevos a partir de ellos.

Personalmente, creo que AMHRA tiene mucho potencial de expansión y capacidad de mejora, sobre todo en la parte de la interfaz de usuario. Esto se debe a que es la primera vez que trabajo a esta escala en un proyecto de ReactJs. La única experiencia previa que tenía consistía en un breve tiempo que estuve durante las prácticas de empresa usando esta tecnología, ya que a posteriori se priorizo otro proyecto en Java.

Críticamente, y viendo mi situación personal actual, tal vez hubiese apuntado a otro tipo de aplicación si hubiese sabido que el tiempo que podría dedicarle al estudio se reduciría considerablemente respecto a las horas consideradas inicialmente. Por ese lado, hay varios aspectos de la aplicación que me hubiese gustado pulir y que ahora definimos como líneas de trabajo futuro a explotar:

- El sistema de registro e inicio de sesión: Actualmente, este sistema es una simple representación y no pretende ser parte de un producto final enfocado al uso instantáneo por parte del usuario.

  En este punto, simplemente trata de simular un sistema que requeriría de conexiones más seguras y donde se cifrara los datos del usuario. Por ejemplo, este proceso podría realizarse mediante Spring Security, un *framework* dedicado a el control de acceso y la autentificación que es lo suficientemente personalizable para realizar dicha tarea en nuestra aplicación.

  La principal razón para implementar este tipo de inicio de sesión y registro más sencillo es ofrecer una mirada general sobre las funciones de las que dispondría un usuario registrado y aquellas que no.

- La interfaz podría mejorarse en gran medida y hacerla visualmente más atractiva. Pese a que se ha seguido la estética y esquema de colores establecido en el prototipo, considero que otra gama de colores podría darle un aspecto más moderno.

- Añadir fotografías a la comida o iconos era uno de los requisitos que, pese a no llegar a redactar, tenía esperanza de poder añadir. Personalmente, creo que este detalle podría cambiar por completo la experiencia del usuario y le aportaría mayor gratificación a la exploración.

- Ampliar el apartado de menús para que se pueda establecer la comida a ingerir para toda la semana. Así como implementar un sistema de recomendación automática de menús según tus racionalidades.

- Añadir filtros más complejos, principalmente a la tabla de alimentos. Actualmente estos filtros permiten buscar a la hora de añadir un alimento a un menú, añadir un grupo alimenticio a un alimento, ver solamente los alimentos creados por el usuario, ordenar alfabéticamente o de menor a mayor… No obstante, sería adecuado proporcionar un cuadro de búsqueda similar al de añadir un alimento en un menú, pero en la tabla de alimentos sobre cada columna.

Aun pese a estos puntos que han quedado pendientes de explotar, considero que la aplicación desarrollada cumple con los objetivos con los que se planteó crearla. Para que esto fuese posible, considero que la metodología por ciclos basada en “Scrum” ha sido de gran utilidad y ha permitido ampliar y reducir la extensión de la aplicación según el tiempo y recursos disponibles en la valoración del ciclo.

Tal y como hay ciertas líneas de trabajo a futuro por llevar a cabo, también existen ciertas partes de la aplicación que resultan positivamente satisfactorias el haberlas estudiado y desarrollado. Entre ellas, destacaría que AMHRA es bastante amigable de cara a cualquier usuario.

Por un lado, está el hecho de haber priorizado que el diseño se adaptara a cualquier tipo de pantalla, permitiendo así acceder desde cualquier tipo de dispositivo y navegador.

Por el otro lado, al limitar las opciones del usuario para que pueda errar lo mínimo posible, se pretende reducir la sensación de frustración e insatisfacción durante el uso de AMHRA. Algunos ejemplos de este tipo de conciencia en el desarrollo son la aparición de los botones de editado y borrado de alimentos solamente en aquellos propios y cuando has iniciado sesión.

Finalmente, y para concluir con esta memoria, este proyecto me ha servido para valorar el potencial que tiene el trabajo en equipo frente a cualquier aplicación web que se plantee. Si realizo una comparación sobre mi situación actual en el trabajo o con el proyecto realizado en la asignatura “Proyecto de desarrollo de software”, considero que la mejora no es solamente a nivel productivo, sino también a nivel moral y de aprendizaje.

Pese a que a veces es difícil coordinarse con los compañeros, considero que permite consultar con más libertad cuando tienes algún error que te está

costando solucionar o permite valorar en conjunto si algo es más correcto de una forma u otra.

Aun con todo esto considero que la realización de este trabajo de final de grado me ha permitido reforzar mis conocimientos adquiridos a través de estos años universitarios, así como adquirir confianza en mis habilidades.

10. Glosario

Framework: Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar [16].

React: Biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. []

Spring Boot: *framework* basado en Java de código abierto. Está desarrollado por Pivotal Team y se utiliza para crear aplicaciones Spring independientes y listas para producción [17]

API: Conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas. [18]

Frontend: Parte visual de la aplicación donde reside la interfaz del usuario. También sirve de conexión entre el usuario y la lógica.

Backend: Parte no visible de la aplicación donde reside toda la lógica de la aplicación.

JSON: Formato de intercambio de datos.

AMHRA: Aplicación web para la medición de raciones en alimentos.

Hook: Funciones que te permiten conectar el estado de React y el ciclo de vida desde componentes funcionales. [20]

Scrum: Metodología ágil orientada a trabajar colaborativamente con el objetivo de realizar entregas parciales, incrementales y regulares.

REST: Interfaz para conectar varios sistemas basados en el protocolo HTTP que sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos como XML y JSON [19].

Responsive: Diseño web caracterizado por su adaptabilidad de la interfaz a la pantalla del dispositivo del usuario.

11. Bibliografía

[1] El 19% de los españoles se puso a dieta el año pasado y el 30% cuenta las calorías. (s.f.). Recuperado el 29 de mayo de 2022, de Heraldo: https://www.heraldo.es/noticias/sociedad/2020/05/22/el-19-de-los-espanoles-se-puso-a-dieta-el-ano-pasado-y-el-30-cuenta-las-calorias-1376298.html

[2] El 81% de los españoles fracasa al hacer una dieta. (s.f.). Recuperado el 29 de mayo de 2022, de El Mundo: https://www.elmundo.es/salud/2014/02/12/52fb7d5022601d7f228b4570.html

[3] Una de cada cinco personas en España se pone a dieta con mucha frecuencia; son un 5% más que hace un año. (s.f.). Recuperado el 29 de mayo de 2022, de 20minutos: https://www.20minutos.es/noticia/4271199/0/una-de-cada-cinco-personas-en-espana-se-pone-a-dieta-con-mucha-frecuencia/

[4] Trigás Gallego, M. (2012). Metodología scrum.

[5] Rodríguez, C., & Dorado, R. (2015). ¿Por qué implementar Scrum? *Revista Ontare*, *3*(1), 125-144.

[6] Hinojosa, M. A. (2003). Diagrama de Gantt. *Producción, procesos y operaciones*.

[7] How to choose a Technology Stack for Web Application Development? (s.f.). Recuperado el 20 de febrero de 2022, de Geeks for Geeks: https://www.geeksforgeeks.org/how-to-choose-a-technology-stack-for-web-application-development/

[8] KeepCoding, R. (21 de enero de 2022). *Los 5 mejores frameworks Full Stack*. Recuperado el 3 de junio de 2022, de KeepCoding: https://keepcoding.io/blog/los-5-mejores-frameworks-full-stack/

[9] Presentando JSX. (s.f.). Recuperado el 03 de junio de 2022, de ReactJS: https://es.reactjs.org/docs/introducing-jsx.html

[10] Microsoft Windows. (s.f.). Recuperado el 03 de junio de 2022, de Wikipedia: https://es.wikipedia.org/wiki/Microsoft_Windows

[11] Eclipse IDE. (s.f.). Recuperado el 03 de junio de 2022, de Eclipse: https://www.eclipse.org/downloads/packages/

[12] Luis Matamala Belinchón. (s.f.). *¿Qué son los Stakeholders y cómo se gestionan en los proyectos?* Recuperado el 04 de junio de 2022, de IMF: https://blogs.imf-formacion.com/blog/mba/que-son-los-stakeholders/

[13] Definición de casos de uso. (s.f.). Recuperado el 04 de junio de 2022, de IBM: https://www.ibm.com/docs/es/elm/6.0.3?topic=requirements-defining-use-cases

[14] Jacobson, I., P. Jonsson, M. Christerson and G. Overgaard, Ingeniería de Software Orientada a Objetos – Un acercamiento a través de los casos de uso. Addison Wesley Longman, Upper Saddle River, N.J., 1992.

[15] Guía para la redacción de casos de uso. (s.f.). Recuperado el 04 de junio de 2022, de Junta de Andalucia: https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416

[16] Framework. (04 de junio de 2022). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Framework

[17] Spring Boot - Introduction. (05 de junio de 2022). Obtenido de Tutorials Point: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

[18] API: qué es y para qué sirve. (s.f.). Recuperado el 05 de junio de 2022, de Xataka: https://www.xataka.com/basics/api-que-sirve

[19] *¿Qué es REST? Conoce su potencia*. (s.f.). Recuperado el 05 de junio de 2022, de Open Webinars: https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/

[20] Un vistazo a los Hooks. (s.f.). Recuperado el 05 de junio de 2022, de Reactjs: https://es.reactjs.org/docs/hooks-overview.html