



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**
NutriMenu



Presentado por Álvaro Manjón Vara
en Universidad de Burgos — 14 de febrero
de 2024

Tutores: Raúl Marticorena Sánchez y Antonio
Jesús Canepa Oneto



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. Raúl Marticorena Sánchez y D. Antonio Jesús Canepa Oneto, profesores del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Exponen:

Que el alumno D. Álvaro Manjón Vara, con DNI 71300527K, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado NutriMenu.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 14 de febrero de 2024

Vº. Bº. del Tutor:

D. Raúl Marticorena Sánchez

Vº. Bº. del co-tutor:

D. Antonio Jesús Canepa Oneto

Resumen

En este Trabajo de Fin de Grado se presenta la reestructuración, reimplementación y unificación de dos aplicaciones previamente existentes, cuyo objetivo es la generación de informes nutricionales teniendo como base los platos y menús ofrecidos en los diferentes centros de restauración de la Universidad de Burgos, así como la propia gestión de los centros y usuarios.

Durante el desarrollo del proyecto se ha llevado a cabo una remodelación completa de ambas aplicaciones, fusionándolas en una sola. Esta transformación ha implicado una modificación integral de la infraestructura, desde el despliegue hasta la lógica de negocio.

Descriptores

composición nutricional, aplicación web, infraestructura, Docker

Abstract

This Bachelor's Degree Final Project presents the restructuring, reimplementation, and unification of two previously existing applications. Its purpose is to generate nutritional reports based on the dishes and menus offered in the different restaurants at the University of Burgos, as well as to manage the restaurants themselves and their users.

Throughout the project development, a complete remodeling of both applications has been carried out, merging them into a single entity. This transformation has involved a comprehensive modification of the infrastructure, from deployment to business logic.

Keywords

nutritional composition, web application, infraestructure, Docker

Índice general

Índice general	iii
Índice de figuras	v
Índice de tablas	vi
Introducción	1
Objetivos del proyecto	3
2.1. Objetivos generales	3
2.2. Objetivos técnicos	3
Conceptos teóricos	5
3.1. HTTP	5
3.2. Aplicación web	6
3.3. Base de datos	6
3.4. API REST	6
3.5. Backend	7
3.6. Frontend	7
3.7. Contenerización	7
Técnicas y herramientas	9
4.1. Patrón de diseño Modelo-Vista-Controlador (MVC)	9
4.2. Infraestructura y despliegue	10
4.3. Lenguajes de programación	12
4.4. Frameworks y librerías	15
4.5. Sistema gestor de bases de datos	18

4.6. Herramientas de desarrollo	19
4.7. Herramientas de gestión	20
4.8. API	25
Aspectos relevantes del desarrollo del proyecto	27
5.1. Inicio del proyecto	27
5.2. Infraestructura de la aplicación	28
5.3. Lógica de negocio	30
5.4. Frontend	32
5.5. Fuente de datos nutricionales	35
5.6. Automatizaciones	36
Trabajos relacionados	37
Conclusiones y Líneas de trabajo futuras	39
7.1. Conclusiones	39
7.2. Líneas de trabajo futuras	39
Bibliografía	41

Índice de figuras

4.1.	Cada componente interactúa con los otros dos en el patrón MVC [9]	10
4.2.	Diagrama de cómo interactúan los elementos dentro de la arquitectura de Docker [11]	11
4.3.	Cada componente dispone de su propio estado, y los componentes padre son capaces de enviar propiedades a los componentes hijo [34]	16
4.4.	Los dos repositorios pueden llegar a estar en distintos puntos dentro de la misma rama [18]	22
4.5.	Cada commit va a ser hijo del commit anterior a este [18]	22
4.6.	A partir de un commit pueden surgir varias ramas distintas [18]	23
4.7.	Diagrama de una operación merge básica [18]	24
4.8.	Repositorio de NutriMenu en GitHub	24
5.1.	MySQL Workbench sólo es compatible con equipos de arquitectura x86_x64	29
5.2.	Modelo de base de datos de Mariya [33]	31
5.3.	Modelo de base de datos actual	32
5.4.	Aplicación renderizando sólo el contenido de los componentes padre	33
5.5.	Distintas tablas y gráficos usados en la aplicación	34
5.6.	Búsqueda de un alimento mediante el uso de la API de Nutritionix	35
5.7.	Listado de pasos ejecutados por la acción de GitHub Actions	36

Índice de tablas

Introducción

En la actualidad, cada vez se le da mayor importancia al hecho de llevar una alimentación saludable y equilibrada, puesto que es uno de los pilares clave para una vida longeva y libre de enfermedades. La Universidad de Burgos es consciente de esto, y es por ello por lo que forma parte de la Red Española de Universidades Promotoras de la Salud, y desde hace años se encuentra desarrollando el proyecto de Aula Campus Saludable, una sección que se encarga de promocionar hábitos saludables a todos los miembros de la Comunidad Universitaria.

Dentro de este área hay varios iniciativas y campañas que se han llevado a cabo, y uno de estos proyectos fue el del desarrollo de un conjunto de aplicaciones, cuyo objetivo era el de informar a los consumidores de los distintos centros de restauración de la Universidad de Burgos de los componentes nutricionales que contienen los platos que forman los menús disponibles. Este conjunto de aplicaciones también permitía la gestión de estos platos y menús, así como de los propios centros de restauración, sus empresas y sus usuarios.

La iniciativa para este nuevo proyecto es la de modernizar y mejorar este trabajo, para que su despliegue e implementación sean lo más sencillos posibles, así como intentar mejorar las aplicaciones webs lo máximo posible, manteniendo su propósito original.

Objetivos del proyecto

Los objetivos que se buscan con el desarrollo de este proyecto son los siguientes:

2.1. Objetivos generales

- Implementar una infraestructura que sea agnóstica al hardware en el que se ejecute.
- Separar la infraestructura en servicios bien definidos y aislados.
- Desarrollar un sistema de orquestación del despliegue y hacer que este sea sencillo y escalable.
- Buscar una forma de poder acceder a nuevos datos nutricionales sin tener que tenerlos almacenados en local de forma obligatoria.
- Unificar las dos aplicaciones web en una, mejorar su interfaz y experiencia de usuario para que sean más sencillas de usar y hacer que sean funcionales en dispositivos móviles.

2.2. Objetivos técnicos

- Contenerizar todos los servicios mediante Docker y Docker Compose.
- Reestructurar la lógica del modelo de datos para separarla de la aplicación web, manteniendo el patrón Modelo - Vista - Controlador (MVC).

- Desarrollar una API REST usando Spring Boot para que esta sea la vía por la que la aplicación interactúe con los datos.
- Implementar un flujo de integración continua mediante el uso de GitHub Actions para comprobar que la infraestructura es funcional en todo momento.
- Desarrollar una aplicación web con React que cumpla con todos los requisitos de las dos aplicaciones previas.
- Hacer un diseño responsive con Bootstrap, que funcione en cualquier tamaño de pantalla.
- Implementar la API externa de Nutritionix como fuente de composición nutricional de los alimentos.
- Hacer uso de GitHub para la gestión del control de versiones de este proyecto, así como la administración de las tareas y los sprints.

Conceptos teóricos

En este apartado se van a tratar de introducir algunos de los conceptos más relevantes al proyecto, para así poder tener una mejor comprensión de estos.

3.1. HTTP

HTTP (*Hypertext Transfer Protocol*) es un protocolo de la capa de aplicación diseñado para distribuir información entre equipos conectados a una red.

Se utiliza como base para la comunicación de datos en la World Wide Web (WWW), permitiendo la transferencia de hipertextos (archivos HTML), que son documentos unidos a través de enlaces.

Este protocolo **facilita la comunicación entre clientes y servidores**, ya que el cliente (por lo general, un navegador web) envía una solicitud HTTP al servidor, el cual responde con los recursos solicitados, como pueden ser una página web, imágenes, archivos... [8]

HTTP dispone de distintos métodos para realizar solicitudes, que son la forma en la que los clientes se comunican con los servidores. Algunos de los métodos más comunes son: **GET**, que se encarga de solicitar un recurso; **POST**, para enviar datos al servidor; y **PUT**, para actualizar un elemento. HTTP también soporta cabeceras en las solicitudes y respuestas, las cuales contienen información sobre el recurso o el estado.

3.2. Aplicación web

Una aplicación web es un programa o software que se ejecuta en un servidor web, en lugar de hacerlo localmente en el dispositivo del usuario, y a la que generalmente se accede mediante un navegador.

La principal característica de una aplicación web es su **capacidad para operar de manera independiente del sistema operativo o del dispositivo**, lo que permite a los usuarios acceder a la misma funcionalidad desde diversos dispositivos.

Además de esto, las aplicaciones web permiten desplegar cambios de forma instantánea, ya que el código no se encuentra descargado en el cliente, sino que este lo solicita al servidor cada vez que accede.

Las aplicaciones web modernas generalmente siguen el modelo de arquitectura cliente-servidor, donde el cliente (el navegador) interactúa con el servidor a través de solicitudes HTTP, permitiendo una experiencia de usuario interactiva y dinámica [31].

3.3. Base de datos

Una base de datos es un sistema organizado de almacenamiento de datos que permite la **recopilación, consulta, actualización y administración de información de manera eficiente**.

Las bases de datos están diseñadas para gestionar grandes volúmenes de datos de forma que se puedan realizar búsquedas, selecciones y operaciones sobre estos con gran rapidez y precisión.

Existen diferentes tipos de bases de datos, como las **relacionales**, que organizan la información en tablas relacionadas entre sí (y son el tipo de base de datos usada en este proyecto); las **no relacionales o NoSQL**, que permiten almacenar datos de forma más flexible; y las **bases de datos distribuidas**, que distribuyen los datos a través de múltiples ubicaciones para mejorar la accesibilidad y la escalabilidad [38].

3.4. API REST

Una API REST (*Representational State Transfer*) es un conjunto de principios de arquitectura que se utilizan para el diseño de interfaces de programación de aplicaciones (APIs) cuyo objetivo es el de permitir la comunicación entre sistemas en red.

Las APIs permiten la interacción entre aplicaciones cliente y servidor a través de protocolos y solicitudes, utilizando métodos HTTP como GET, POST, PUT y DELETE para realizar operaciones CRUD (creación, lectura, actualizado y eliminado) sobre recursos web identificados mediante URIs (*Uniform Resource Identifier*) [21].

Al ser interfaces, la mayor ventaja que ofrecen es que nos permiten acceder a los datos de una aplicación sin necesidad de conocer su modelo de datos interno, ni el cómo funciona su lógica de negocio.

3.5. Backend

El backend se refiere a la **parte del servidor de una aplicación**, la cual es responsable de la lógica de negocio, el almacenamiento de datos y la gestión de solicitudes que provienen del frontend [7].

Es la parte de la aplicación que no se ve de forma directa, pero que es crucial para el funcionamiento correcto de una aplicación, ya que es el núcleo de esta. Si nuestro proyecto no dispusiese de un backend la aplicación no podría almacenar ni consultar ningún dato, por lo que perdería totalmente el sentido.

3.6. Frontend

El frontend es la **parte de una aplicación web o móvil con la que el usuario interactúa directamente**. Es toda la parte visible, como la UI (User Interface), UX (User eXperience), el diseño escogido, los colores, las animaciones... [7]

El desarrollo frontend se enfoca principalmente en la experiencia del usuario, el diseño de interfaces atractivas y adaptables que se ajusten a diferentes dispositivos y tamaños de pantalla, y la implementación de la lógica de interacción en el lado del cliente.

3.7. Contenerización

Se conoce como contenerización a la tecnología de virtualización que permite ejecutar contenedores, entornos completamente aislados del resto de la máquina que los está ejecutando, y que contienen todo lo necesario (bibliotecas, código, archivos de configuración, dependencias...) para que se pueda ejecutar una aplicación en cualquier entorno.[5]

A diferencia de entornos clásicos de virtualización como las máquinas virtuales, en los que se emula el hardware por completo y se debe disponer de una instalación completa del sistema operativo para funcionar, en los contenedores se permite que **múltiples instancias comparten un mismo sistema operativo**, a pesar de estar usando espacios de ejecución distintos. Esto se consigue usando características de Linux como los *espacios de nombres del kernel* o los *cgroups* [20], puesto que los contenedores están basados en Linux, pero esto no impide su ejecución en distintas plataformas y sistemas operativos, incluido Windows.

Técnicas y herramientas

En este apartado voy a proceder a introducir y destacar las herramientas, metodologías y tecnologías que me han ayudado a poder desarrollar este proyecto:

4.1. Patrón de diseño Modelo-Vista-Controlador (MVC)

La función del patrón de diseño conocido como **Modelo-Vista-Controlador** es la de organizar de forma estructurada los componentes fundamentales de un determinado *sistema de software*, estableciendo relaciones entre ellos.

Como metodología de trabajo, fue propuesta por Trygve Reenskaug en 1979, basándose su utilidad en tres ideas fundamentales [1]:

- La organización y clasificación de la información que se aporta al sistema.
- La unificación y gestión de la lógica del sistema.
- La presentación de datos al usuario mediante una interfaz comprensible y asequible.

Los tres componentes fundamentales de este patrón tienen misiones diferenciadas sobre la forma en cómo tratan la información [9]:

- **Modelo:** Es el componente cuya misión es únicamente la de gestionar el contenido de una base de datos, incluida su manipulación y utilización. Es la parte que se va a encargar de la **gestión de los datos**.

- **Vista:** Es el componente que sirve para acceder al contenido de la base de datos y presentar los datos al usuario. Va a ser **la parte con la que va a interactuar el usuario**.
- **Controlador:** Este componente es el que pone en conexión el modelo y la vista, gestionando y procesando las instrucciones que se reciben para obtener un resultado. Su forma básica de trabajar consiste en acceder a la base de datos, manipularlos con una determinada finalidad y presentar los resultados de esa manipulación de los datos. Se va a encargar de gestionar **la lógica y la gestión de los datos**.

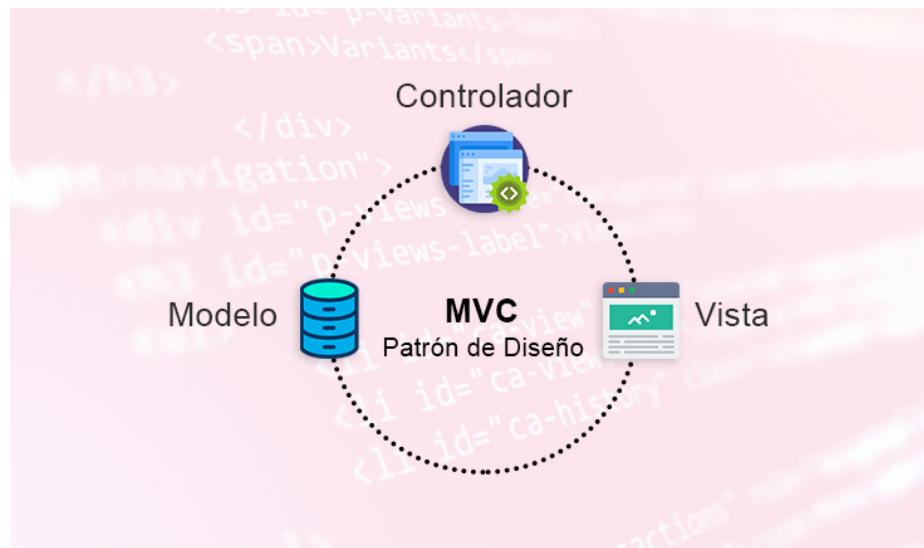


Figura 4.1: Cada componente interactúa con los otros dos en el patrón MVC [9]

El motivo por el cual he decidido utilizar este patrón es porque permite trabajar de una forma precisa y eficaz, haciendo que la estructura del sistema sea clara, ordenada y separada, lo que facilita su comprensión y posterior mantenimiento.

4.2. Infraestructura y despliegue

Docker

Docker es una plataforma de código abierto que permite el desarrollo, ejecución y distribución de aplicaciones en contenedores.

Arquitectura de Docker

La arquitectura de Docker es una arquitectura **cliente-servidor** [11]. El cliente de Docker, que es con lo que interactúa el usuario, habla con el *daemon* haciendo llamadas API REST, y este se encarga de gestionar las peticiones y los objetos de Docker, como son las imágenes, los contenedores, los volúmenes y las redes. Finalmente, para conseguir las imágenes, el *daemon* habla con *Docker Registry*, el repositorio que contiene todas las imágenes necesarias para el despliegue. Por defecto se usa **Docker Hub**, un repositorio público en el que cualquiera puede subir y descargar imágenes, pero se puede usar también un repositorio privado.

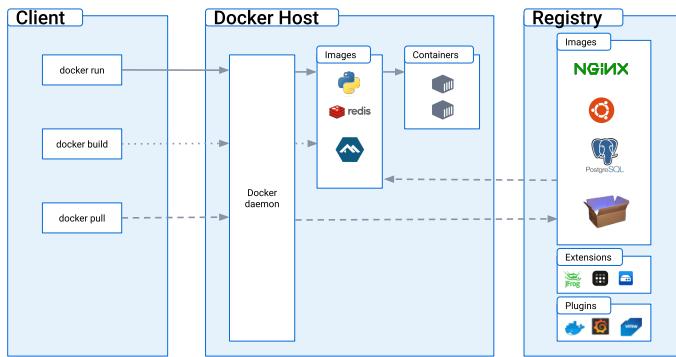


Figura 4.2: Diagrama de cómo interactúan los elementos dentro de la arquitectura de Docker [11]

Imágenes

Las imágenes son plantillas que contienen instrucciones y parámetros para crear contenedores de Docker. Se puede crear un contenedor usando directamente una imagen del *registry*, reutilizando imágenes ya existentes como base para crear nuestras propias imágenes, o incluso creando nuestras propias imágenes desde cero. Las imágenes se crean usando archivos *Dockerfile*, añadiendo en él las distintas dependencias, instrucciones y parámetros, y cada uno de estos elementos es una capa dentro de la imagen. De esta manera, cada vez que se modifica el *Dockerfile* y se vuelve a construir la imagen, realmente sólo se vuelven a construir las capas en las que han habido cambios, acelerando de esta forma el proceso de despliegue. [12]

Docker Compose

Docker Compose es una herramienta que nos permite definir, orquestrar, desplegar y escalar **aplicaciones Docker multi-contenedor**. [10] Para ello, se define un archivo *YAML* en el que se van a definir las configuraciones necesarias para los distintos servicios pertenecientes a la aplicación. Esto nos va a permitir establecer relaciones y dependencias entre distintos contenedores que forman parte de un mismo aplicativo, permitiendo así que varios contenedores formen parte de una misma red o tengan la capacidad de compartir volúmenes de almacenamiento, por poner unos ejemplos.

He decidido usar esta herramienta ya que facilita bastante la conexión e integración entre los distintos servicios que componen la aplicación, como son la base de datos, el backend, y el frontend. Además de esto, a la hora de poner la aplicación en producción, obtenemos varias ventajas respecto a un entorno tradicional:

- **Automatización de despliegues**, ya que una vez hayamos definido en el archivo de configuración de Docker Compose todas las dependencias, configuraciones y relaciones necesarias, con un sólo comando Docker Compose se va a encargar de crear y desplegar los contenedores, configurar los elementos necesarios, e iniciar los servicios.
- **Creación de entornos aislados y consistentes**, lo que nos permite desplegar entornos de desarrollo locales, así como entornos CI/CD de validación y pruebas, y que estos repliquen la infraestructura de producción, que también puede ser desplegada usando Docker Compose.
- **Escalabilidad de infraestructura**, puesto que con un sólo comando podemos aumentar o disminuir el número de instancias de las que dispone un servicio.

4.3. Lenguajes de programación

HTML

El lenguaje HTML (*HyperText Markup Language*) hoy en día es la base de internet, ya que es prácticamente imposible encontrar una página web que no tenga definido su contenido usando este lenguaje. Como **lenguaje de marcado** que es, utiliza marcas (etiquetas) que indican qué tipo de contenido se está representando en el navegador, lo que permite identificar y

estructurar mejor este contenido, así como ser más precisos posteriormente a la hora de dar estilo a este contenido [14].

Otro de los valores fundamentales del lenguaje HTML es el **hipertexto**, que es lo que permite enlazar fácilmente contenidos, ya sea dentro de la misma página web o con otras distintas. Esto permite mejorar la forma de acceso a la información e interconectar el conocimiento, algo muy importante puesto que este es uno de los pilares básicos de internet.

Un archivo HTML sólo contiene (o debería contener, según las buenas prácticas [26]) **información referente a la estructura y el contenido de la página web**, pero no información relacionada con el cómo se va a ver estructurada esta información o cómo se van a procesar los datos y se van a interactuar con ellos, puesto que eso es tarea de otros lenguajes, como CSS y JavaScript. Esto hace que, si visualizamos un archivo HTML desde el navegador que no contiene referencias a ningún otro archivo de otro lenguaje, veamos la información mostrada de forma cruda, sin ningún tipo de estilo ni distribución aplicadas mas allá que los que vienen por defecto en los navegadores.

CSS

Como complemento óptimo del lenguaje HTML, el lenguaje CSS (*Cascading Style Sheets*) define **la estructura y los estilos utilizados en la presentación** de documentos escritos con HTML.

CSS no es un lenguaje de programación propiamente dicho ni tampoco un lenguaje de marcado, sino que es lo que se ha dado en llamar un **lenguaje de hojas de estilo**. Su funcionalidad principal es la de definir y mejorar la presentación de páginas web creadas con HTML, creando estilos y formatos que les son de aplicación [13].

De la misma forma que HTML, CSS es uno de los lenguajes más usados hoy día en el diseño de los estilos de páginas web, ya sea de forma pura o mediante el uso de distintos frameworks que facilitan su uso.

JavaScript

La funcionalidad principal del lenguaje JavaScript es la de mejorar la experiencia de los usuarios de páginas web, haciendo que las páginas dejen de ser estáticas para pasar a ser **interactivas y dinámicas**, lo que indudablemente las hace más atractivas e incrementa exponencialmente su funcionalidad [30].

A diferencia de HTML, que es un lenguaje de marcado; y CSS, que es un lenguaje de hojas de estilo; JavaScript es un lenguaje de programación propiamente dicho con **compilación just-in-time**, es decir, compilación en tiempo de ejecución [15]. Este lenguaje consta de todo lo que podríamos esperar a la hora de escribir código, desde funciones y estructuras de lógica hasta la implementación de programación orientada a objetos.

Al ser un lenguaje de programación con funcionalidad completa, no sólo se usa JavaScript para añadir funciones a las páginas web, sino que hoy en día hay aplicaciones y entornos construidos en JavaScript, como son **Node.js**, **Visual Studio Code**, o **Adobe Acrobat**. JavaScript también puede ser usado tanto en el **lado del cliente**, para añadir interactividad a la web; o en el **lado del servidor**, para manejar la lógica de la aplicación y modificar los datos de esta.

Java

Java es un **lenguaje de programación orientado a objetos basado en C y C++**, buscando ser similar a estos lenguajes pero haciéndolo más universal, ya que la idea de este lenguaje es que **se pueda compilar una vez y funcionar en cualquier sistema**, gracias a la implementación de la máquina virtual de Java.

Este lenguaje es uno de los más populares y utilizados hoy en día, ya que podemos encontrarlo en prácticamente cualquier campo y ámbito, gracias a su compatibilidad, robustez, y gran cantidad de librerías y *frameworks* de terceros. [29]

En mi caso he usado este lenguaje para el desarrollo del *backend*, puesto que al ser un lenguaje que ya conozco y con el que tengo experiencia, sabía que el usar un *framework* como **Spring Boot**, así como distintas librerías como **JPA**, no iba a suponer un problema, puesto que el ecosistema Java es de los más extensos y robustos que hay ahora mismo, por lo que no voy a tener problema a la hora de buscar documentación o investigar cómo añadir una funcionalidad que me interese.

4.4. Frameworks y librerías

Spring y Spring Boot

Spring Boot es una herramienta que **simplifica y agiliza el desarrollo de aplicaciones web basadas en Spring**, un framework de código abierto para Java.

Spring es un **framework que facilita el desarrollo de aplicaciones en Java**, añadiendo soporte para el patrón MVC, inyección de dependencias, inversión de control, y demás herramientas esenciales. Sin embargo, las aplicaciones basadas en Spring son difíciles de configurar, así que el objetivo de Spring Boot es proporcionar un conjunto de herramientas para crear rápidamente y de forma fácil **aplicaciones Spring mantenibles y sencillas de entender**. [32]

Antiguamente, cuando queríamos desarrollar un proyecto en Spring, los pasos a seguir eran:

1. Crear un proyecto en Maven o Gradle y seleccionar y añadir al proyecto las dependencias necesarias.
2. Configurar la aplicación mediante ficheros XML o anotaciones complejas.
3. Desplegar la aplicación en el tipo de servidor que escogiésemos.

Sin embargo, Spring Boot se encarga de reducir significativamente la complejidad de este proceso, ofreciendo configuraciones automáticas que podemos desplegar prácticamente sin tener que tocar nada, integración con servidores HTTP, mejor gestión de las dependencias... [39]

Son todas estas ventajas las que me han hecho decantarme por usar esta tecnología para desarrollar el *backend* de la aplicación, ya que me permite usar toda la potencia que ofrece Spring sin necesidad de desarrollos ni configuraciones complejas que puedan entorpecer mi proyecto.

React

React es una librería de JavaScript y TypeScript creada por Meta para el desarrollo de aplicaciones web y móviles. Esta librería **permite construir interfaces de usuario a partir de componentes independientes y reutilizables**, y esto hace que el desarrollo se facilite bastante, ya que ahorra

repetir código y hace que la interfaz sea modular, lo que le da bastante escalabilidad y ayuda a la hora de añadir nuevas implementaciones.

Los componentes interactúan entre ellos de tal forma que el flujo de los datos es unidireccional, puesto que se crea un **árbol de componentes** en el que el estado se transfiere del componente principal hacia los componentes secundarios. Esto hace que sea más fácil rastrear y depurar el estado de la aplicación [24].

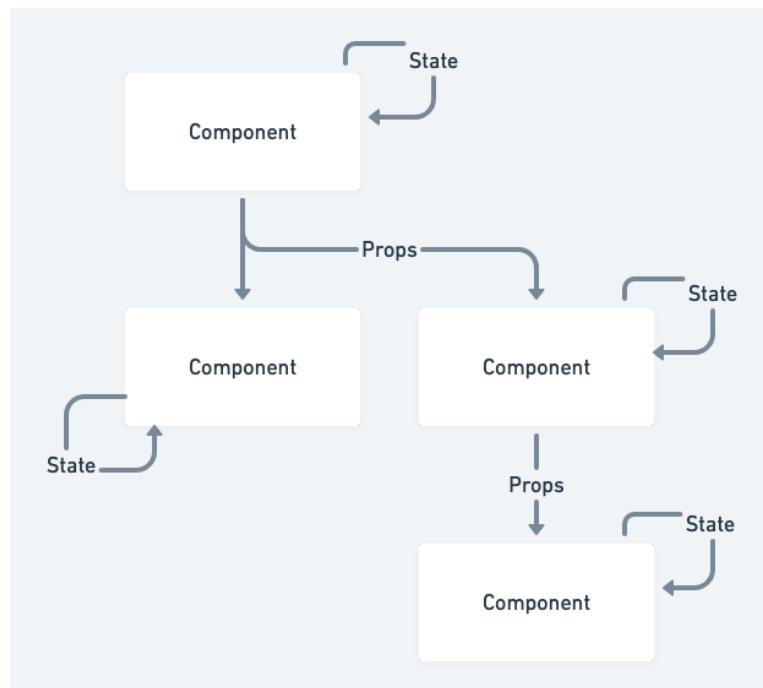


Figura 4.3: Cada componente dispone de su propio estado, y los componentes padre son capaces de enviar propiedades a los componentes hijo [34]

Además de este elemento, **React dispone también de un Virtual DOM** que ayuda a mejorar el rendimiento de las aplicaciones web. En lugar de actualizar directamente el DOM (*Document Object Model*) del navegador cada vez que cambia el estado de la aplicación, React crea una representación virtual del DOM y sólo actualiza las partes que han cambiado. Esto hace que se reduzca la carga en el navegador y mejore la velocidad de la aplicación [17].

React utiliza JSX (*JavaScript XML*), una **extensión de la sintaxis de JavaScript que se utiliza para escribir código HTML dentro de React**, permitiéndonos insertar y evaluar expresiones de JavaScript dentro

del código HTML, algo esencial a la hora de crear las vistas de nuestra aplicación [16].

A parte de todas estas funciones, React también está dotado de librerías y componentes adicionales, como **React Router**, que da *soporte de enrutado* a React para poder crear URLs personalizadas; y **React Context**, que *dota a nuestras aplicaciones React de un estado global compartido* entre los distintos componentes. Esto ha hecho que React haya terminado siendo mi elección a la hora de desarrollar el *frontend* de la aplicación.

JPA

JPA (*Java Persistence API*) es una especificación de Java que nos permite integrar en nuestro código Java **conexiones e interacciones con bases de datos, persistencia de objetos y mapeo de estos con un modelo relacional**.

Esto nos permite no tener que escribir *queries SQL* directamente dentro de nuestro código, ya que una vez establezcamos con anotaciones las relaciones entre el modelo relacional de la base de datos que estemos usando y las clases de nuestro proyecto, vamos a poder usar una implementación concreta de JPA, como Hibernate o en mi caso el propio Spring Boot (puesto que JPA no deja de ser una especificación, no una implementación), para interactuar con los datos mediante el uso de interfaces y llamadas a métodos [2].

Bootstrap

Bootstrap es un *framework CSS*, inicialmente desarrollado por Twitter y después convertido en un proyecto de código abierto, usado para **facilitar el desarrollo de interfaces responsive para aplicaciones web y sitios móviles** [35].

Este framework proporciona plantillas de elementos tales como *botones, formularios, tablas, carruseles...* ya estilizados pero personalizables, lo que me ha permitido ahorrarme bastante tiempo a la hora de dar estilo a los distintos elementos HTML de la web, aparte que **hace que no sea necesario disponer de conocimientos extensos de CSS** para conseguir un diseño agradable, adaptable y funcional.

Además de esto, Bootstrap **usa el sistema CSS Grid para colocar los elementos**, lo que ayuda bastante a posicionar los elementos donde nosotros queramos.

Bootstrap se puede integrar en prácticamente cualquier librería o framework, pero en el caso de React además existe una librería llamada **React Bootstrap**, que lo que hace es integrar Bootstrap dentro de los componentes y el enfoque usado en React, por lo que lo hace aún más simple [6].

4.5. Sistema gestor de bases de datos

MySQL

MySQL es un sistema RDBMS (*Relational Database Management System*) gratuito (aunque dispone de una licencia empresarial) y de código abierto, y es **uno de los sistemas de administración y gestión de bases de datos más populares y utilizados**, motivo por el que lo he escogido como herramienta de trabajo.

Su funcionamiento se basa en las **bases de datos relacionales**, en las que los datos se organizan y almacenan en tablas que pueden relacionarse entre sí; y en el **modelo cliente-servidor**, en el que el servidor es el encargado de gestionar las bases de datos, y el cliente interactúa con el servidor para solicitar información, modificarla, insertarla o eliminarla mediante el uso del lenguaje SQL, permitiendo así que múltiples clientes puedan interactuar con una base de datos centralizada. [37]

Algunas de las ventajas que me han hecho decantarme por este sistema son:

- Es un proyecto de código abierto, activo y con gran soporte de la comunidad, lo que significa que va a disponer de prácticamente cualquier característica que necesite, y cualquier error va a ser solucionado lo más pronto posible.
- Este sistema es conocido por su rendimiento y eficiencia, por lo que va a poder soportar las cargas de mi aplicación sin problema alguno.
- Es un sistema altamente escalable, por lo que si en algún momento la aplicación creciese y necesitase de más potencia de procesamiento o de opciones de clusterización, sé que con este gestor no tendría ningún problema.

4.6. Herramientas de desarrollo

Visual Studio Code

Visual Studio Code es un **editor de código fuente gratuito y de código abierto** desarrollado por Microsoft y disponible para macOS, Windows y Linux, así como recientemente también [en versión web](#).

Al ser una herramienta bastante enfocada en el desarrollo web (aunque está adaptada para el desarrollo de cualquier lenguaje), puesto que la propia aplicación está desarrollada en JavaScript, y además dispone de gran soporte en cuanto a extensiones y personalización por parte de la comunidad de desarrolladores, es una herramienta perfecta para un desarrollo como el de este proyecto, ya que dispone de un montón de herramientas que pueden agilizar y facilitar el trabajo. En mi caso esto ha sido especialmente útil a la hora de desarrollar el **frontend**, puesto que gracias al alto número de extensiones que existen en el *marketplace* de Visual Studio Code he encontrado bastantes integraciones con React, como [Simple React Snippets](#), una extensión que permite usar abreviaciones para insertar bloques de código comunes en el desarrollo de React.

Como ya he mencionado, este IDE dispone de un montón de opciones a la hora de personalizar el entorno de programación, lo cuál sumado a las propias características de la aplicación (integración con *Git*, autocompletado y resaltado de sintaxis con *IntelliSense*, herramientas de debugging...) la convierten en una aplicación muy versátil [25].

IntelliJ IDEA

Otro IDE que he usado para poder llevar a cabo este trabajo es IntelliJ IDEA, **un editor que permite escribir código Java de forma eficaz y rápida**.

Gracias a ser un editor de código principalmente enfocado en el lenguaje Java, presenta **características e integraciones que de normal no encontraríamos en otros editores** (aunque en editores como Visual Studio Code las podríamos llegar a asemejar mediante el uso de extensiones, pero en este editor tenemos la ventaja de que ya encontramos todo esto por defecto), como integración con los paquetes Maven, detección automática de errores, autocompletado inteligente del código... [22]

Esta aplicación dispone de una versión gratuita y otra de pago, pero al ser estudiante he tenido acceso a la versión completa sin coste adicional, disponiendo de todas las funcionalidades que ofrece esta herramienta. [23]

Postman

Postman es una herramienta diseñada para **construir, probar y usar APIs** (*Application Programming Interfaces*).

Esta herramienta soporta la **creación de colecciones de APIs**, lo cual es bastante útil a la hora de probar una API y sus distintos *endpoints*, ya que sólo necesitas guardarlos una vez para poder hacer después todas las pruebas que deseas.

Postman también permite **hacer solicitudes mediante cualquier método HTTP**, y tiene la capacidad de **ejecutar pruebas automatizadas**, lo cuál puede llegar a ser bastante útil en un desarrollo como este, en el que por un lado se dispone de una API propia (el *backend*) y por otro lado se está usando una API externa (*Nutritionix*) para obtener la información nutricional [28].

4.7. Herramientas de gestión

Metodología Scrum

Para el desarrollo de este proyecto he seguido la metodología Scrum, que es una forma de trabajo que permite **gran agilidad, flexibilidad y eficiencia en la gestión de proyectos de software** [4].

Un proceso Scrum se ejecuta en base a la realización de sesiones de trabajo (*sprints*) que suelen tener dos semanas de duración. Al finalizar cada sprint, se realiza la entrega específica que se ha definido y marcado inicialmente.

Previamente al inicio del sprint, se realiza **una reunión de planificación** dirigida por el responsable del proyecto (*Scrum Master*), que en este caso serían los tutores del TFG, en la que se marca al *Equipo de Desarrollo*, es decir, a mí, **los objetivos a cumplir durante el sprint**, la metodología de trabajo a utilizar, o se analizan y solventan las dudas y obstáculos que puedan presentarse durante el mismo.

Durante el periodo que dura el sprint, **se pueden realizar breves reuniones diarias**. En ellas se analizan los obstáculos surgidos en el día anterior y se sincronizan las actividades a realizar en ese día.

Al finalizar el sprint, **se analiza si el trabajo realizado es óptimo y se ajusta a los objetivos** planteados inicialmente. Para ello se realiza una reunión retrospectiva en la que se analiza qué ha funcionado o no, y qué es mejorable.

La utilidad de la metodología Scrum radica en varios aspectos destacables:

- **Es flexible** y permite adaptarse a los cambios.
- Es transparente y **fomenta la comunicación entre los miembros del equipo**, lo que además redunda en una mayor colaboración entre ellos.
- Al final de cada sprint, **el producto final resulta mejorado e incrementado gracias a un proceso de retroalimentación**, lo que permite su mejora continua.

Git

Git es una herramienta inicialmente diseñada por *Linus Torvalds*, la cual permite el **control de versiones** de un proyecto de software. Este sistema de control gestiona los cambios y los diferentes estados y versiones por los que pasa el código de un proyecto, por lo que es una herramienta esencial para el desarrollo de cualquier proyecto, ya sea individual o colaborativo. [3]

La base de Git son **los repositorios**, el lugar donde *se va a almacenar todo el código y el histórico de cambios* que se ha producido sobre este.

Cada desarrollador va a disponer de su propio **repositorio local**, que no es más que *su copia local del código y sus cambios*, sumado a los cambios que este mismo desarrollador haya realizado.

Además de esto, va a haber un **repositorio central**, que va a ser *el lugar donde se unifican todos los cambios* que realicen los distintos desarrolladores. Los repositorios locales van a ser un *clon* de este repositorio principal. Como veremos en el siguiente punto, en el caso de este proyecto el repositorio central se trata de GitHub.

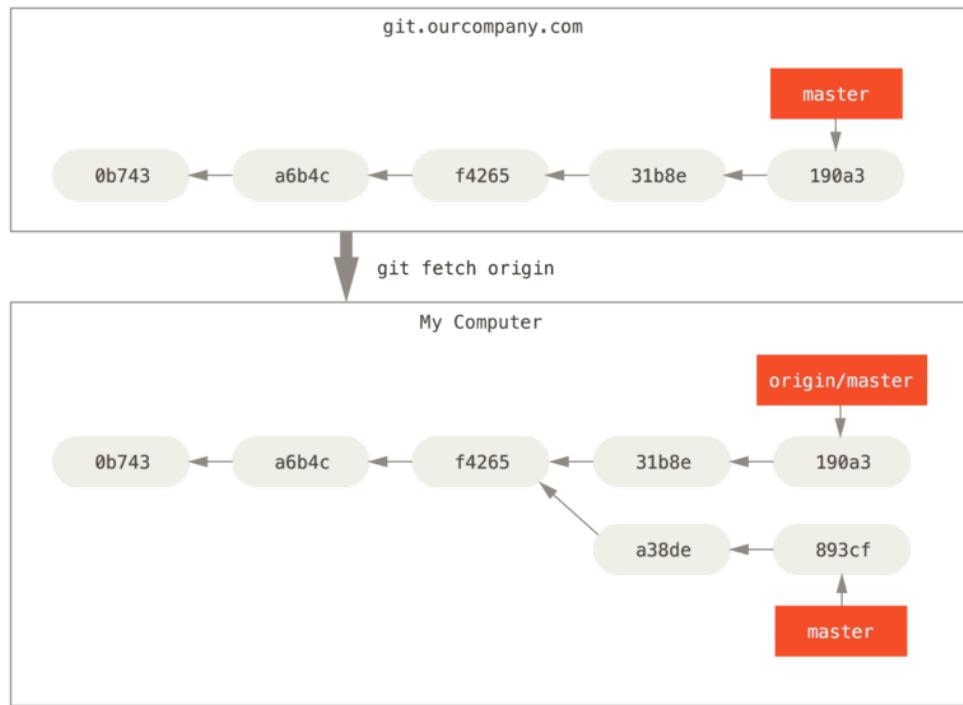


Figura 4.4: Los dos repositorios pueden llegar a estar en distintos puntos dentro de la misma rama [18]

Los desarrolladores van a realizar los cambios e implementaciones pertinentes sobre el código, y una vez estén satisfechos con estos, van a realizar lo que se denomina como **commit**, *una instantánea del código* en ese preciso momento.

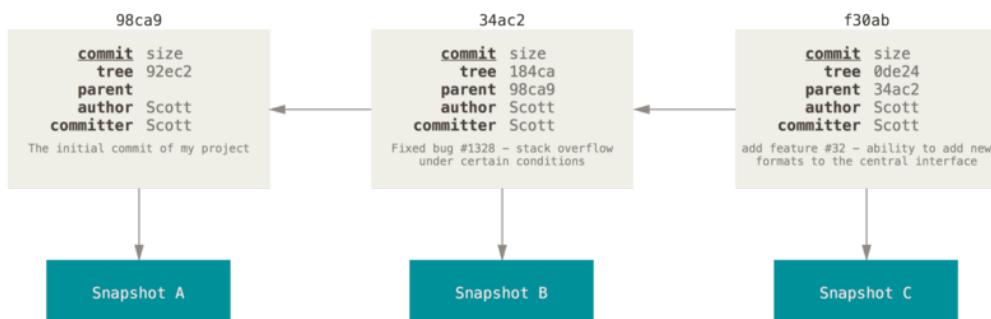


Figura 4.5: Cada commit va a ser hijo del commit anterior a este [18]

Para poder realizar modificaciones en el código sin pisar el desarrollo de otros compañeros, ni modificar el código principal de la aplicación, se da uso de lo que se denomina como **ramas o branches**, que son *versiones independientes del código en un estado concreto* (para seleccionar este momento indicamos de qué *commit* partimos). Las ramas son un elemento fundamental a la hora de realizar implementaciones, experimentar con el código o simplemente hacer cambios, y son uno de los puntos clave en las buenas prácticas del desarrollo de software.

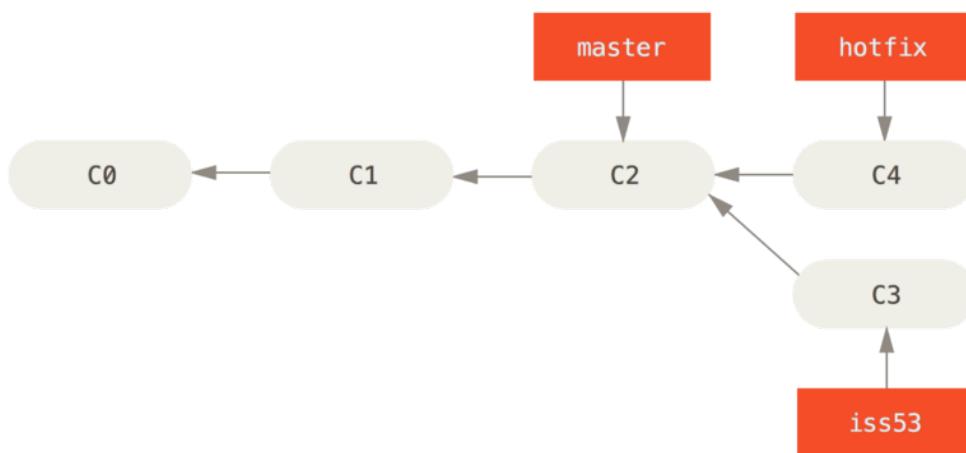


Figura 4.6: A partir de un commit pueden surgir varias ramas distintas [18]

Una vez nuestros cambios estén listos, todo funcione correctamente y tengamos nuestros *commits* hechos, podemos subir el código de nuestro repositorio local al repositorio central haciendo ***push***. De la misma forma, si nuestros compañeros han estado realizando cambios y queremos descargarlos a nuestro repositorio local, podemos hacerlo mediante ***pull***.

En caso de que hayamos terminado con el trabajo de una rama concreta y queramos unificar estas modificaciones a la rama principal, usaremos la operación ***merge***, que se encargará de fusionar e integrar los cambios a la rama principal. [18]

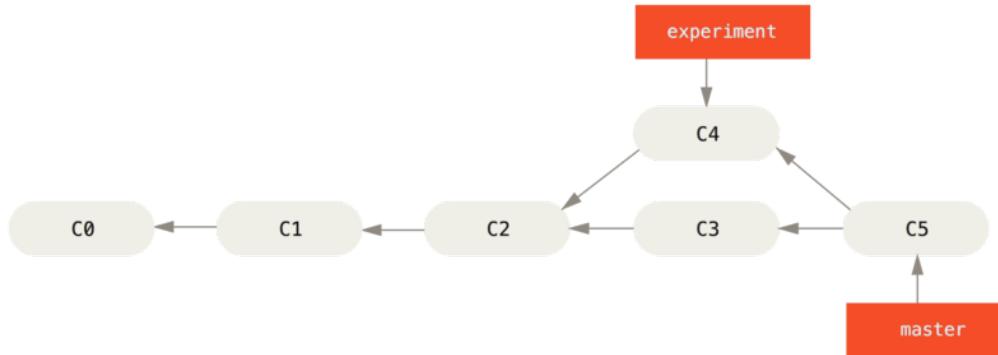


Figura 4.7: Diagrama de una operación merge básica [18]

GitHub

GitHub es una plataforma web de Microsoft que permite **alojar el código de proyectos software que usen el sistema de control de versiones Git**, gestionar estos proyectos y sus tareas, y facilitar la colaboración entre usuarios.

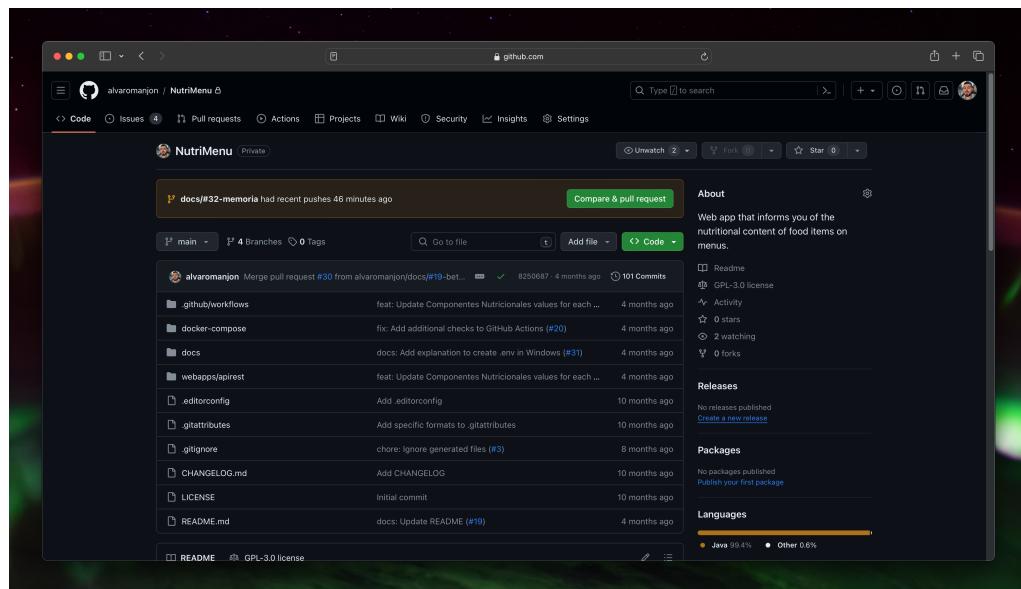


Figura 4.8: Repositorio de NutriMenu en GitHub

Como GitHub actúa como un repositorio basado en Git, mediante GitHub se puede examinar todo el historial de versiones por las que ha pasado el código, volver hacia atrás en el tiempo y analizar la evolución de este. Además

de esto, también permite hacer gestión de las ramas de los repositorios, con herramientas como las ***pull requests***, que permiten hacer solicitudes de *merge* a los administradores de los proyectos, siendo esto un pilar fundamental en la comunidad de código abierto.

A parte de las funciones relacionadas con Git, GitHub también ofrece herramientas para gestionar tareas, como son los ***issues***, e incluso integración con la *metodología Scrum* mediante el uso de ***sprints***, lo que ha hecho que haya terminado siendo mi herramienta de elección a la hora de gestionar este proyecto, tanto a nivel de código como de gestión de trabajo y tareas [36].

GitHub Actions

GitHub también ofrece **GitHub Actions**, una **plataforma de automatización que permite crear flujos de *testing* personalizados**. Esto puede ser configurado para hacer que estas pruebas se ejecuten cada vez que hagamos un *commit*, una *pull request*... [19]

En caso de mi proyecto, cada vez que hago un *commit* se ejecutan unas pruebas de *build* y *ejecución* para asegurarme de que todo funciona correctamente, lo cuál me ha ayudado a evitar errores que podía haber llegado a pasar por alto.

4.8. API

Nutritionix

Nutritionix es una de las **plataformas de información nutricional más grandes del mundo**, disponiendo de información verificada de más de 600.000 alimentos [27].

La base de datos de la que dispone este servicio es usada por una gran cantidad de aplicaciones de salud y *fitness*, y además de esto el servicio está dotado de **bases de datos regionales y especializadas** con los alimentos de cada lugar, lo cuál ha sido un punto muy a tener en cuenta, ya que al final el público base de este proyecto es el español, y no tiene sentido usar datos de una fuente de información en la que los alimentos van a venir en otro idioma o no va a disponer de los alimentos de esa zona.

La API pública de la que disponen tiene opciones bastante competitivas si la comparamos con el resto de competidores en el mercado, especialmente

el nivel gratuito, así que creo que es la opción perfecta para un proyecto de este nivel.

Aspectos relevantes del desarrollo del proyecto

En este apartado se recogerán los aspectos de mayor relevancia que han surgido durante la realización de este proyecto, con el objetivo de comprender mejor el desarrollo y los desafíos encontrados a lo largo del camino.

5.1. Inicio del proyecto

Este proyecto, propuesto por mis tutores, surge como continuidad de otro Trabajo de Final de Grado, el realizado por Mariya Aleksandrova Stroyanova, en el que trabajó para crear una versión web y ampliar las funcionalidades de otro Trabajo de Final de Grado, el realizado por Joseba Fernando Moisén, que consistía en una aplicación de escritorio de la que se podían obtener los valores nutricionales de los alimentos de la base de datos de BEDCA.

El Trabajo de Mariya, que ha sido el punto de partida, consta de dos aplicaciones web: una encargada de la generación y gestión de empresas, locales, usuarios, menús y platos, que sería la parte que controlarían las empresas encargadas de la gestión de las cafeterías de las distintas facultades de la Universidad de Burgos; y otra aplicación que sería a la que tendrían acceso los clientes, en la cual se pueden realizar informes nutricionales a partir de los menús escogidos por el usuario.

La propuesta para este Trabajo de Final de Grado ha sido la de mejorar estas aplicaciones web, tanto a nivel de infraestructura y servicios, como a nivel de interactividad y usabilidad, procediendo a realizar un rediseño completo de ellas y haciéndolas mucho más amigables a los dispositivos

móviles, ya que es el lugar desde donde van a acceder la mayoría de usuarios que visiten la cafetería.

5.2. Infraestructura de la aplicación

Inicialmente, las aplicaciones web parten de un modelo de despliegue de forma manual y en local, ya que requieren de la instalación de varios programas, como MySQL Community, para poder lanzar la base de datos; el SDK de Java, para poder ejecutar el código; así como Eclipse o un IDE similar, para poder lanzar los proyectos de Spring Boot en los que están contenidas las aplicaciones. Esto, como se puede observar en el apartado de Documentación técnica de programación en los anexos de Mariya [33] es un proceso bastante laborioso, que requiere de la preparación y configuración de un entorno específico para esta tarea, y que no dota a la infraestructura de ningún tipo de escalabilidad ni posibilidad de automatización a la hora de su despliegue.

Además de esto, este modelo de infraestructura es dependiente del sistema operativo y la arquitectura de hardware usadas, lo cual puede suponer un gran inconveniente, como ha terminado siendo el caso, ya que mi equipo personal es un MacBook Pro con procesador Apple Silicon, el cual está basado en una arquitectura ARM. Esto ha supuesto un gran problema a la hora de realizar el despliegue inicial de las aplicaciones, ya que el despliegue está pensado para ser realizado desde un equipo con sistema operativo Windows.

Primero se intentó solucionar este problema mediante el uso de una máquina virtual, puesto que existen versiones tanto de Windows 10 como de Windows 11 para equipos con arquitectura ARM (cosa no exclusiva de los procesadores de Apple, ya que los propios equipos de Microsoft, los portátiles Surface, actualmente también están dotados de procesadores ARM). Sin embargo, el gran inconveniente surgió a la hora de instalar MySQL, ya que nada más abrir el instalador se obtuvo el siguiente error:

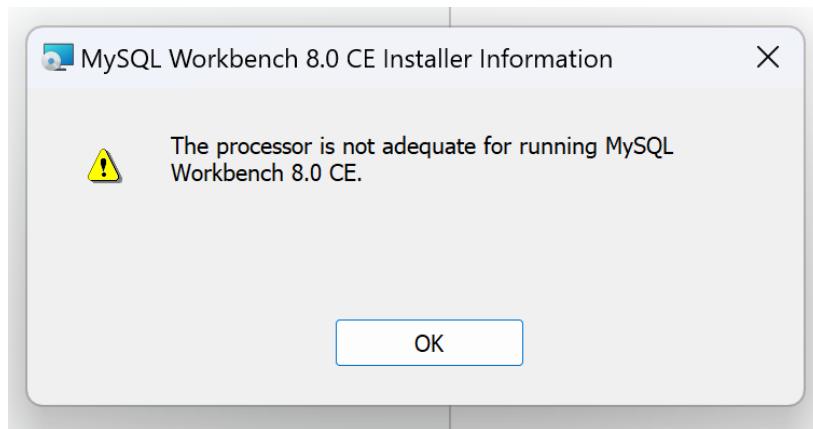


Figura 5.1: MySQL Workbench sólo es compatible con equipos de arquitectura x86_x64

Esto, unido a la motivación de la búsqueda de un despliegue rápido, escalable y moderno, ha terminado en la decisión de realizar un aislamiento completo de los distintos servicios (frontend, backend y base de datos) y contenerizar estos servicios mediante el uso de **Docker** y **Docker Compose**, para poder orquestarlos y gestionarlos de forma conjunta. Esto trae numerosas ventajas al proyecto:

- **La infraestructura pasa a ser totalmente independiente del sistema operativo y la arquitectura de procesador**, ya que Docker funciona con todos los sistemas y arquitecturas principales.
- **El despliegue se vuelve mucho más rápido y sencillo**, ya que en prácticamente un minuto y con tan sólo dos comandos puedes estar ejecutando la aplicación con todos sus servicios activos.
- **Este tipo de infraestructura permite la creación de distintos entornos** (desarrollo, producción, testing) consistentes y sencillos de implementar, ya que permite asegurarse de que la aplicación se va a ejecutar de la misma forma en todos ellos.
- **Cada contenedor se encuentra aislado del resto**, con sus propios recursos y sistemas de archivos, por lo que mejora bastante la seguridad y asegura que un problema en uno de ellos no va a afectar ni canibalizar al resto.

Además de esto, puesto que ambas aplicaciones trabajan con el mismo conjunto de datos y no existe realmente un grado de separación necesario como para mantener dos proyectos, también se ha decidido unificar las dos aplicaciones web en una, debido a que esto facilita bastante su desarrollo, elimina código repetido y mantiene mayor homogeneidad.

5.3. Lógica de negocio

En el proyecto original, ambas aplicaciones siguen el patrón **Modelo-Vista-Controlador (MVC)** en todo su desarrollo, tanto para el manejo de datos como la interfaz, ya que todo forma parte de un mismo monolito. Sin embargo, como en este proyecto se ha decidido hacer una separación total de los servicios, el enfoque adoptado ha sido el de **API-Driven Development**, en el que primero se implementa esta lógica de negocio siguiendo el patrón MVC, y en la capa del Controlador se implementa una API REST que proporciona los endpoints necesarios para que el frontend interactúe con esta.

Esto ha permitido abstraer a la aplicación web de tener que interactuar directamente con la base de datos, lo que ahorra el uso de queries SQL complejas y permite prescindir de APIs como JDBC, ya que se va a trabajar directamente con objetos JSON mucho más amigables de manejar y representar en la capa frontend.

Para la implementación de este servicio se ha mantenido el uso de **Spring Boot**, al igual que en el proyecto original, lo que ha permitido tener el modelo original como referencia a la hora de realizar esta reimplementación.

Modelo de datos

Puesto que tanto el proyecto de Mariya como el actual siguen el modelo MVC, las capas del Modelo de datos se asemejan, ya que en ambos se mantiene un modelo puramente relacional. Sin embargo, esta transformación en la capa del Controlador, así como los cambios realizados a la hora de obtener la información de los alimentos, han hecho que haya varias diferencias entre los dos:

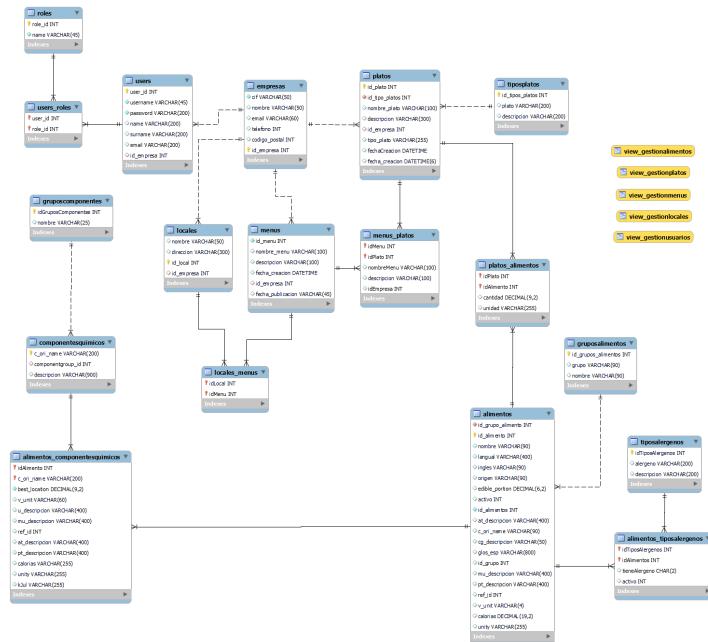


Figura 5.2: Modelo de base de datos de Mariya [33]

Como los datos van a ser accedidos y manipulados directamente desde la API, se ha eliminado la necesidad de tener vistas en la base de datos, ya que los propios endpoints son los encargados de mostrar sólo la información que nos interese, permitiendo además mayor personalización y granularidad en los resultados obtenidos.

Además de esto, como en la sección de Alimentos se ha pasado de un modelo en el que los datos sólo se podían almacenar de forma local, usando los datos de BEDCA, a un modelo híbrido que permite tanto el almacenamiento local como la interacción con la API de Nutritionix, esta parte del modelo de datos se ha rediseñado completamente, intentando hacerla más simple y fácil de entender.

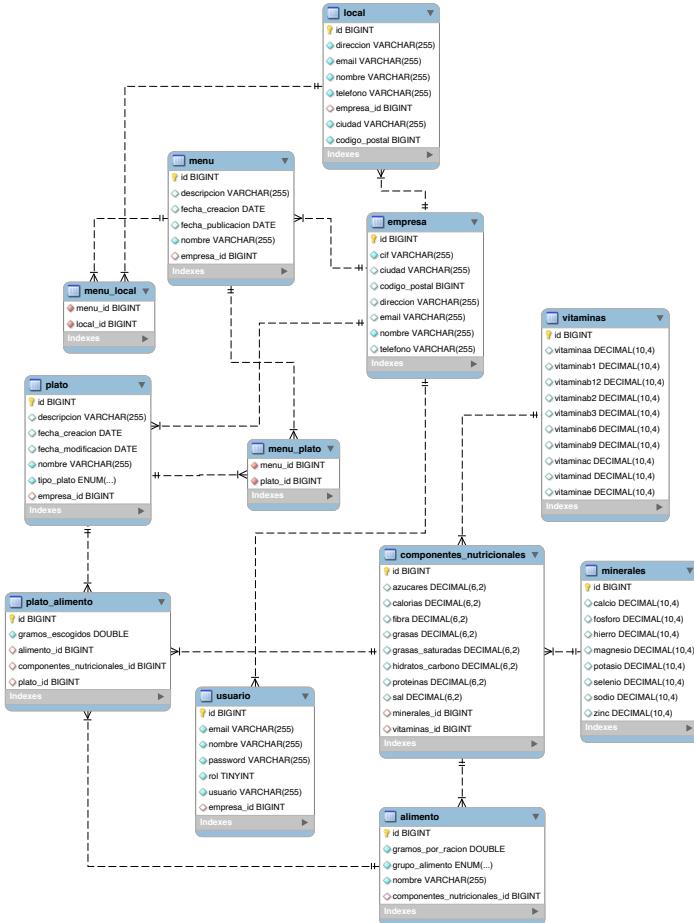


Figura 5.3: Modelo de base de datos actual

Para la implementación de la base de datos se ha mantenido **MySQL**, ya que al seguir manteniendo un modelo de bases de datos relacional no he visto necesidad de modificarlo.

5.4. Frontend

Este es el servicio que más cambios ha sufrido, puesto que prácticamente ha sido recreado desde 0. Para el desarrollo de esta aplicación web se ha usado la librería **React**, debido a que actualmente es de las opciones más populares en el desarrollo frontend, y su modelo basado en Componentes me pareció muy buena solución para el desarrollo de este proyecto, puesto que prima bastante la reutilización y la modularidad.

Sin embargo, como React no dispone de forma nativa de ningún sistema de enrutamiento, puesto que su modelo principal es el de Single Page Applications (aplicaciones que reescriben el contenido mostrado de forma dinámica, en vez de recargar la página completa), el proyecto da bastante uso a otra librería pensada para ser usada junto a React, que es **React Router**. Esta librería ha permitido gestionar las distintas partes de la aplicación sin ningún problema, pues al ser una aplicación con distintas secciones bien diferenciadas, siguiendo el modelo de las SPA sería mucho más complicado de separar y limitaría bastante las posibilidades.

Además de esto, **React Router** incluye otras funciones bastante aprovechadas en este proyecto, como los **Outlets**, que permiten crear un componente plantilla y que los componentes hijos se rendericen dentro de ese componente padre (usados en varias partes, como por ejemplo en los layouts de la barra de navegación y los paneles de gestión de los elementos, como se puede ver en la figura 5.4); o los **loaders**, que son funciones que permiten realizar llamadas asíncronas a una API antes de que se renderice el componente que va a usar esos datos (usados cada vez que se muestra información sobre un alimento, plato o menú), lo cual acelera bastante la carga.

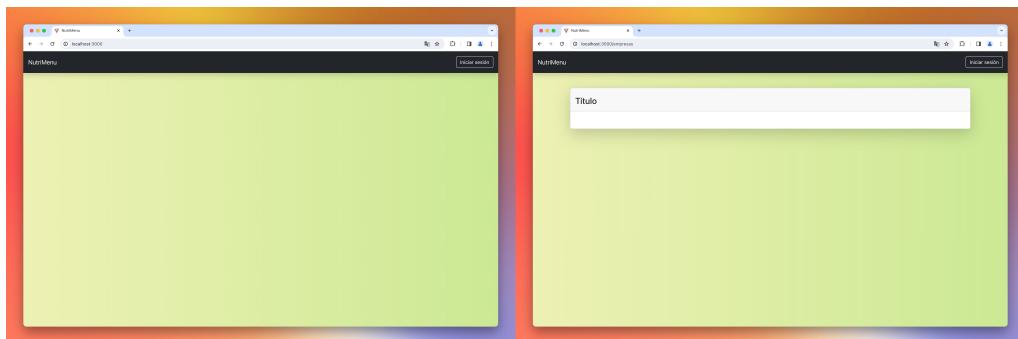


Figura 5.4: Aplicación renderizando sólo el contenido de los componentes padre

El desarrollo se ha realizado mediante el uso de **JavaScript**, lo que ha supuesto posiblemente el mayor reto de todo el proyecto, pues previo a la realización de este trabajo no había tenido ninguna experiencia con este lenguaje.

Como he usado React, a la hora de escribir el código referente a la interfaz he usado **JSX**, una extensión de JavaScript que permite integrar tanto HTML como JavaScript como si fueran un sólo lenguaje.

Para la construcción y compilación del frontend inicialmente se comenzó usando **Create React App**, ya que hasta hace poco era la opción recomendada por los propios desarrolladores de React, pero puesto que es un proyecto que ha dejado de ser mantenido, y los paquetes que contiene están quedando desactualizados, finalmente se ha cambiado por **Vite**, una de las herramientas de compilación de JavaScript más eficientes de la actualidad. Esta herramienta ofrece distintas optimizaciones a la hora de ejecutar y compilar nuestro código que hacen que la aplicación web sea lo más rápida posible, lo que se nota bastante al usarla, ya que todo es prácticamente instantáneo.

En el estilado de la aplicación se ha trabajado principalmente con **Bootstrap**, ya que anteriormente ya había realizado algún proyecto dando uso de este framework y tenía algo de experiencia, y ha sido de bastante ayuda al crear una interfaz responsive y visualmente agradable, adaptada tanto a pantallas de dispositivos móviles como a pantallas de ordenador.

Existe una librería llamada **React Bootstrap** que integra todos los elementos de Bootstrap dentro de React, para trabajar con ellos como si fueran componentes de React, por lo que me he ahorrado el uso de bastantes clases de CSS.

Por último, para la representación de las tablas y los gráficos en las vistas de información nutricional (ejemplo en la figura 5.5), he usado las librerías **AG React Data Grid** y **AG React Charts** respectivamente, debido a que requieren de muy poco código para conseguir los resultados buscados, y me permiten actualizar los datos de forma dinámica, algo que necesitaba para poder cambiar los datos nada más se seleccione un plato distinto en la vista del informe nutricional de un menú.

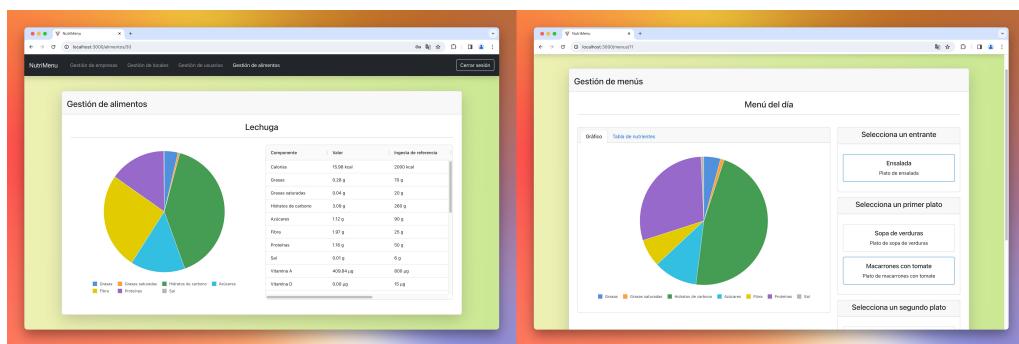


Figura 5.5: Distintas tablas y gráficos usados en la aplicación

5.5. Fuente de datos nutricionales

Las aplicaciones originales hacían uso de la base de datos de BEDCA, la Base de Datos Española de Composición de Alimentos, para obtener toda la información relacionada con los alimentos y sus componentes nutricionales.

El gran inconveniente de esto es que esta base de datos no dispone de un punto de acceso al que consultar los datos mediante una API o similar, así que era necesario mantener la base de datos en local. Esto, además de que impide obtener la última información nutricional disponible, puede suponer un costo en almacenamiento y rendimiento importante, ya que todos los alimentos y su información deben estar guardados en el servidor obligatoriamente.

Para solucionar esto, se ha decidido dar uso de la **API de Nutritionix**, que dispone de una base de datos específica para alimentos en castellano. Esto permite almacenar sólo los alimentos que vayan a ser usados, mientras que el resto quedan disponibles con tan sólo realizar una búsqueda en la aplicación (como se ve en la figura 5.6). Además, para suplir aquellos alimentos que pueden no estar presentes en esta nueva base de datos, también se ha dado la opción de crear los alimentos a mano, para que en caso de que sea necesario un alimento esto no sea un factor limitante.

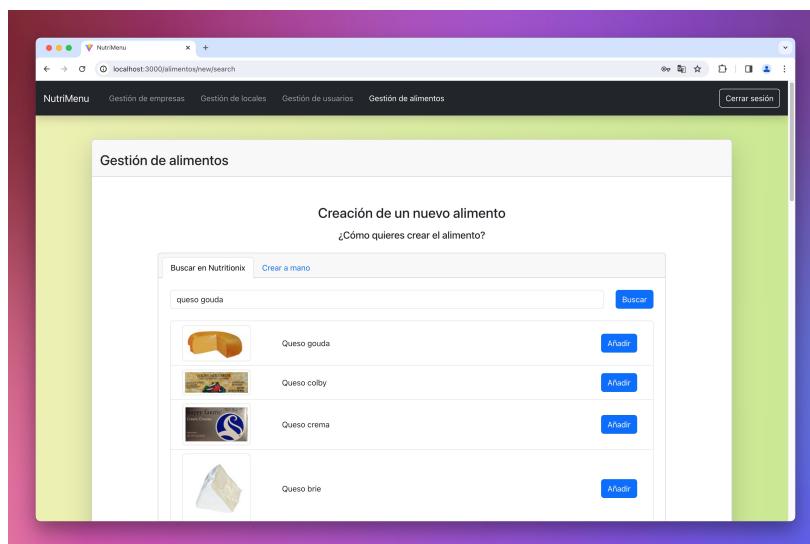


Figura 5.6: Búsqueda de un alimento mediante el uso de la API de Nutritionix

El mayor inconveniente que he encontrado con el uso de esta API es el hecho de que no contiene información nutricional sobre alérgenos, quedando esto como objeto de mejora del proyecto.

5.6. Automatizaciones

Como la infraestructura de la aplicación dispone de varios servicios independientes, es posible que en algún momento realice un cambio en uno de ellos y sin darme cuenta esto afecte a otro distinto, haciendo que algún elemento no funcione como debería. Es por ello que he realizado una implementación de integración continua mediante el uso de **GitHub Actions**, aprovechando que el despliegue se realiza con Docker y Docker Compose.

De lo que se encarga esta automatización es que, cada vez que se hace un push de algún commit al repositorio de GitHub, se ejecutan una serie de pasos (como podemos ver en la figura 5.7) para desplegar los contenedores de Docker en una máquina nueva, y le da un minuto desde que todos los contenedores están corriendo para que cada servicio devuelva un *healthcheck* indicando que todo está funcionando correctamente. En caso de que alguno de los servicios no devuelva este mensaje, el propio GitHub se encarga de enviar un mensaje indicando que algo ha fallado, además de indicarlo en el propio commit.

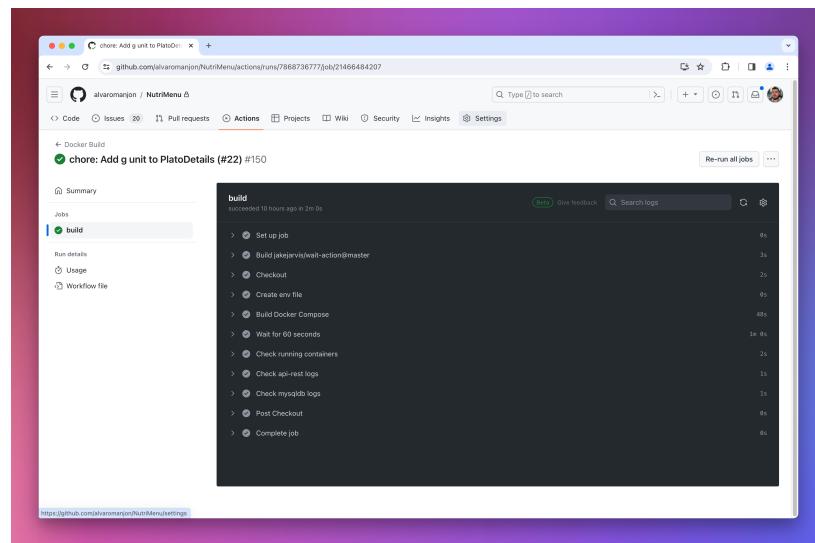


Figura 5.7: Listado de pasos ejecutados por la acción de GitHub Actions

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

Este proyecto me ha permitido ganar mucha experiencia, pues al final he terminado realizando un desarrollo completo en el que he tenido que pasar por todas las etapas, desde el plantear cómo estructurar los datos, a pensar cómo representarlos, para acabar en investigar de qué forma hacer que esta aplicación esté disponible de la forma más eficiente posible. Me ha gustado rotar por todos los roles, lo cual me ha proporcionado una visión más amplia sobre el funcionamiento general y la responsabilidad de cada parte.

Antes de este proyecto tenía poca experiencia en el ámbito del desarrollo web, especialmente en lo relacionado con el frontend, y esto me ha hecho crecer como desarrollador y tener una idea más clara de hacia dónde quiero orientar mi carrera profesional en el futuro.

Además, el desarrollar un trabajo estrechamente relacionado con la nutrición me ha hecho más consciente de su importancia para nuestra salud y de cómo las pequeñas decisiones diarias pueden tener un gran impacto a largo plazo.

7.2. Líneas de trabajo futuras

- Implementación de una fuente de datos de alérgenos, para incrementar la información disponible sobre cada alimento.

- Implementación de algún método que permita mostrar los alimentos con valores nutricionales sin llenar como nulos, no como 0, para evitar confusiones a la hora de ver los informes.
- Securización de la API mediante el uso de tókenes, para impedir el acceso a ciertos *endpoints* dependiendo del tipo de usuario que los consulte.
- Mejorar la gestión de las sesiones en la aplicación web, para que se almacenen de forma segura y cifrada, y tengan un *timeout* que al superarse haga que se cierre la sesión.
- Implementación de distintos tests, como unitarios y de rendimiento tanto en el frontend como en el backend.
- Añadir bebidas como nuevo tipo de platos a la hora de componer los menús.
- Añadir paginación, filtrado y distintas opciones de ordenación en las tablas de gestión de los distintos elementos.
- Añadir soporte de localización a la aplicación, para que pueda ser mostrada en otros idiomas.
- Añadir funcionalidades de accesibilidad a la aplicación, para que sea más sencilla de usar por todos los usuarios

Bibliografía

- [1] José María Aguilar. ¿Qué es el patrón MVC en programación y por qué es útil? <https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx>, Oct 2019.
- [2] José Manuel Alarcón. La API de persistencia de Java: ¿Qué es JPA? - JPA vs Hibernate vs EclipseLink vs Spring JPA. <https://www.campusmvp.es/recursos/post/la-api-de-persistencia-de-java-que-es-jpa-jpa-vs-hibernate-vs-eclipselink-vs-spring-jpa.aspx>, Mar 2021.
- [3] Atlassian. Qué es Git. <https://www.atlassian.com/es/git/tutorials/what-is-git>, 2023.
- [4] Atlassian. Qué es scrum y cómo empezar. <https://www.atlassian.com/es/agile/scrum>, 2023.
- [5] Microsoft Azure. ¿Qué es un contenedor? <https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-a-container/>, 2023.
- [6] React Bootstrap. React Bootstrap. <https://react-bootstrap.netlify.app/>, 2023.
- [7] Nicole Chapaval. Qué es Frontend y Backend: características, diferencias y ejemplos. <https://platzi.com/blog/que-es-frontend-y-backend/>, 2018.

- [8] Cloudflare. ¿Qué es HTTP? <https://www.cloudflare.com/es-es/learning/ddos/glossary/hypertext-transfer-protocol-http/>, 2024.
- [9] Easy App Code. Patrón de diseño MVC. ¿Qué es y cómo puedo utilizarlo? <https://www.easyappcode.com/patron-de-diseno-mvc-que-es-y-como-puedo-utilizarlo>, Sept 2020.
- [10] Docker Docs. Docker Compose overview. <https://docs.docker.com/compose/>, 2023.
- [11] Docker Docs. Docker overview - Docker architecture. <https://docs.docker.com/get-started/overview/#docker-architecture>, 2023.
- [12] Docker Docs. Docker overview - Docker objects. <https://docs.docker.com/get-started/overview/#docker-objects>, 2023.
- [13] MDN Web Docs. CSS. <https://developer.mozilla.org/es/docs/Web/css>, 2023.
- [14] MDN Web Docs. HTML: Lenguaje de etiquetas de hipertexto. <https://developer.mozilla.org/es/docs/Web/HTML>, 2023.
- [15] MDN Web Docs. JavaScript. <https://developer.mozilla.org/es/docs/Web/JavaScript>, 2023.
- [16] React Docs. Escribir marcado con JSX. <https://es.react.dev/learn/writing-markup-with-jsx>, 2023.
- [17] Geeks for Geeks. React Virtual DOM. <https://www.geeksforgeeks.org/reactjs-virtual-dom/>, 2023.
- [18] Git. Git Basics. <https://git-scm.com/book/en/v2>, 2023.
- [19] GitHub. Documentación de GitHub Actions. <https://docs.github.com/es/actions>, 2023.
- [20] Sascha Grunert. Demystifying Containers - Part I: Kernel Space. <https://medium.com/@saschagrunert/demystifying-containers-part-i-kernel-space-2c53d6979504>, Mar 2019.
- [21] Red Hat. ¿Qué es una API REST? <https://www.redhat.com/es/topics/api/what-is-a-rest-api>, 2024.
- [22] JetBrains. IntelliJ IDEA. <https://www.jetbrains.com/es-es/idea/>, 2023.

- [23] JetBrains. Licencias educativas gratuitas. <https://www.jetbrains.com/es-es/community/education/#students>, 2023.
- [24] Kinsta. ¿Qué es React.js? Un Vistazo a la Popular Biblioteca de JavaScript. <https://kinsta.com/es/base-de-conocimiento/que-es-react-js/>, 2023.
- [25] Microsoft. Visual Studio Code. <https://code.visualstudio.com/>, 2023.
- [26] Midudev. Curso COMPLETO de HTML GRATIS desde cero: SEO, semántica y más. <https://www.youtube.com/watch?v=3nYLTiY5skU>, Oct 2023.
- [27] Nutritionix. What is Nutritionix. <https://www.nutritionix.com>, 2023.
- [28] Postman. What is Postman? <https://www.postman.com/product/what-is-postman/>, 2023.
- [29] Amazon Web Services. ¿Qué es Java? <https://aws.amazon.com/es/what-is/java/>, 2023.
- [30] Amazon Web Services. ¿Qué es JavaScript? <https://aws.amazon.com/es/what-is/javascript/>, 2023.
- [31] Amazon Web Services. ¿Qué es una aplicación web? <https://aws.amazon.com/es/what-is/web-application/>, 2024.
- [32] Spring. Spring Boot. <https://spring.io/projects/spring-boot/>, 2023.
- [33] Mariya Aleksandrova Stroyanova. TFG del Grado en Ingeniería Informática - Generador de informes nutricionales de la restauración en centros universitarios - Documentación Técnica. Documento PDF, Feb 2022.
- [34] Jessica Verduzco. Aprendiendo React JS. <https://blog.atomikod.com/2020/12/22/aprendiendo-react-js-primeros-pasos-los-conceptos-clave/>, Dic 2020.
- [35] W3Schools. What is Bootstrap? https://www.w3schools.com/whatis/whatis_bootstrap.asp, 2023.
- [36] Wikipedia. GitHub. <https://en.wikipedia.org/wiki/GitHub>, 2023.

- [37] Wikipedia. MySQL. <https://en.wikipedia.org/wiki/MySQL>, 2023.
- [38] Wikipedia. Base de datos. https://es.wikipedia.org/wiki/Base_de_datos, 2024.
- [39] Cecilio Álvarez Caules. ¿Qué es Spring Boot? <https://www.arquitecturajava.com/que-es-spring-boot/>, Oct 2023.