



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Aplicación web para la gestión
de negocios de restauración
con soporte de datos
nutricionales: Orderly**



Presentado por Amanda Pérez Olmos
en Universidad de Burgos — 11 de febrero
de 2026

Tutor: Raúl Marticorena Sánchez



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. Raúl Marticorena Sánchez, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Expone:

Que la alumna Dña. Amanda Pérez Olmos, con DNI 71365665T, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado “Aplicación web para la gestión de negocios de restauración con soporte de datos nutricionales: Orderly”.

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

En Burgos, 11 de febrero de 2026

V.^o B.^o del Tutor:

D. Raúl Marticorena Sánchez

Resumen

Orderly es una aplicación web integral diseñada para optimizar y centralizar la gestión operativa de locales de restauración (bares, restaurantes, cafeterías). Su objetivo principal es integrar en una única plataforma los flujos de trabajo fundamentales de este sector: la gestión de productos y su composición, el control de comandas (pedidos de bar y comedor), la administración de reservas de mesas y la gestión de usuarios y permisos del personal.

Una característica diferenciadora de *Orderly* es su integración nativa con **Open Food Facts**, una base de datos colaborativa de información nutricional. Esta funcionalidad permite enriquecer el catálogo de alimentos con datos nutricionales detallados, clasificaciones como el **Nutri-Score** y la identificación de alérgenos de forma semi-automática, respondiendo así a la creciente demanda de transparencia y opciones de alimentación saludable por parte de los consumidores.

El desarrollo se ha abordado con un fuerte énfasis en la calidad del *software*, la escalabilidad y la mantenibilidad. Se ha implementado una arquitectura moderna basada en **React** para el *frontend* y **Spring Boot** para el *backend* (API REST), utilizando contenedores **Docker** para garantizar un despliegue portable y reproducible. La metodología de trabajo ha seguido prácticas ágiles (*Scrum/Kanban*), con un riguroso control de versiones mediante **Git** y **GitHub**.

Descriptores

Gestión de restauración, información nutricional, Nutri-Score, alérgenos, aplicación web, React, Spring Boot, API REST, Open Food Facts, Docker, contenerización, desarrollo ágil.

Abstract

Orderly is a comprehensive web application designed to optimise and centralise the operational management of hospitality venues (bars, restaurants, cafés). Its main goal is to unify the fundamental workflows of this sector into a single platform: product and recipe management, order control (for both bar and dining areas), table reservation administration, and staff user and permission management.

A key distinguishing feature of *Orderly* is its native integration with ***Open Food Facts***, a collaborative database of nutritional information. This functionality enriches the food catalogue with detailed nutritional data, classifications such as ***Nutri-Score***, and semi-automatic allergen identification, addressing the growing consumer demand for transparency and healthy food options.

The development has been approached with a strong emphasis on software quality, scalability, and maintainability. A modern architecture based on ***React*** for the frontend and ***Spring Boot*** for the backend (REST API) has been implemented, using ***Docker*** containers to ensure portable and reproducible deployment. The workflow followed agile practices (*Scrum/Kanban*), with rigorous version control using ***Git*** and ***GitHub***.

Keywords

Restaurant management, nutritional information, Nutri-Score, allergens, web application, React, Spring Boot, REST API, Open Food Facts, Docker, containerization, agile development.

Índice general

Índice general	iii
Índice de figuras	v
Índice de tablas	vi
1. Introducción	1
2. Objetivos del proyecto	3
2.1. Objetivos generales	3
2.2. Objetivos técnicos	3
3. Conceptos teóricos	5
3.1. <i>Nutri-Score</i>	5
3.2. Clasificación NOVA	6
3.3. Regulación de alérgenos (Reglamento UE 1169/2011)	7
4. Técnicas y herramientas	9
4.1. Metodologías y técnicas de gestión	9
4.2. Patrón de arquitectura	11
4.3. Tecnologías y <i>frameworks</i> principales	12
4.4. Herramientas de desarrollo y documentación	18
5. Aspectos relevantes del desarrollo del proyecto	23
5.1. Inicio del proyecto	23
5.2. Desarrollo del <i>backend</i>	24
5.3. Desarrollo del <i>frontend</i>	29
5.4. Consumo de la API de datos nutricionales	32

5.5. Evaluación automatizada de la calidad del código	33
6. Trabajos relacionados	37
6.1. Floreant POS	37
6.2. MenuCalc	38
6.3. NutriMenu	38
6.4. Comparativa sobre las funcionalidades cubiertas	39
7. Conclusiones y Líneas de trabajo futuras	41
7.1. Conclusiones	41
7.2. Líneas de trabajo futuras	42
Bibliografía	45

Índice de figuras

3.1. Etiquetas A-E según <i>Nutri-Score</i>	5
3.2. Etiquetas 1-4 según clasificación NOVA	6
3.3. Catorce alérgenos de declaración obligatoria de la UE	7
4.1. Arquitectura de <i>Orderly</i>	11
4.2. Gráfico nutricional de un producto usando <i>Recharts</i>	15
4.3. Tablero Kanban de <i>GitHub Projects</i>	18
4.4. Comparativa de ramas entre <i>GitFlow</i> y <i>GitHub Flow</i> [42]	19
5.1. Modelo de base de datos de <i>NutriMenu</i> [56]	25
5.2. Modelo de base de datos actual	26
5.3. Diagrama de comunicación entre capas	28
5.4. Ejemplos de errores de la API que utilizan el DTO <code>ErrorResponse</code>	29
5.5. <i>Dashboard</i> de pedidos	31
5.6. Búsqueda de un alimento mediante la API <i>Open Food Facts</i>	32
5.7. Métricas de calidad de código generadas por <i>SonarCloud</i>	34

Índice de tablas

6.1. Comparativa de características entre aplicaciones	39
--	----

1. Introducción

El sector de la restauración es un entorno dinámico y complejo donde la eficiencia operativa es un factor clave para el éxito. Tradicionalmente, la gestión de estos establecimientos ha dependido de una combinación de herramientas dispersas (desde registros manuales hasta *software* especializado a menudo cerrado y poco adaptable) que no siempre se comunican entre sí. Esta fragmentación de procesos aumenta la complejidad operativa y el margen de error, dificultando el flujo de trabajo entre cocina y sala.

En este contexto, *Orderly* surge como un proyecto con el objetivo de desarrollar una solución web integral, moderna y centralizada. La aplicación busca ser un **punto de gestión unificado** que cubra los procesos esenciales de un local: desde la creación y composición de platos, pasando por el registro de comandas y la gestión de reservas, hasta la administración de los roles del equipo. Al integrar estas funcionalidades en una plataforma modular y extensible, se pretende no solo agilizar los flujos de trabajo, sino ofrecer una herramienta que se adapte con facilidad a distintos tipos de establecimientos y necesidades.

Un pilar diferenciador de este proyecto es su compromiso con la **transparencia nutricional**. La integración directa con la API de **Open Food Facts** permite a los establecimientos automatizar la consulta de ingredientes y alérgenos, facilitando la elaboración de fichas de productos precisas y verificadas. En una era donde la conciencia sobre la alimentación saludable es creciente, esta funcionalidad añade un valor significativo y contemporáneo a la herramienta.

Este documento constituye la **memoria** principal del proyecto, donde se detallan los objetivos, conceptos teóricos, técnicas empleadas y los aspectos más relevantes del desarrollo. La documentación técnica completa, que

incluye la planificación detallada del proyecto, la especificación de requisitos y diseño, así como los manuales de programador y usuario, se presenta en un conjunto de **anexos** independientes.

Todo el código fuente, la documentación y el histórico de desarrollo están disponibles públicamente en el repositorio de *GitHub* del proyecto:

<https://github.com/aperezolmos/Orderly>

El progreso del proyecto ha sido gestionado mediante la herramienta *GitHub Projects*, cuyo tablero y gráficas asociadas pueden visualizarse en:

<https://github.com/users/aperezolmos/projects/4>

El reporte de calidad del código realizado por *SonarCloud* puede visualizarse en:

https://sonarcloud.io/project/overview?id=aperezolmos_Orderly

2. Objetivos del proyecto

A continuación, se detallan los objetivos que han guiado el desarrollo del proyecto.

2.1. Objetivos generales

1. **Crear un sistema de gestión centralizado para locales de restauración** que unifique las operaciones críticas de gestión de productos, pedidos, reservas y usuarios en una única interfaz web.
2. **Integrar información nutricional externa** de manera automática, mediante el consumo de una API pública, para añadir valor diferencial y promover la transparencia alimentaria.
3. **Facilitar la identificación y gestión de alérgenos**, permitiendo asociarlos a los productos disponibles y ofrecer mecanismos de filtrado que ayuden al personal a identificar opciones seguras.
4. **Demostrar competencia en el ciclo completo de desarrollo de software**, desde el análisis y diseño hasta la implementación, pruebas, despliegue y documentación, aplicando los conocimientos adquiridos durante el grado.

2.2. Objetivos técnicos

1. **Implementar una arquitectura cliente-servidor desacoplada:** desarrollar un *frontend* dinámico con *React* que consuma una API REST construida con *Spring Boot*.

2. **Diseñar un modelo de datos relacional eficiente y mantenible:** utilizar *Spring Data JPA* sobre *MySQL* para definir un esquema de base de datos normalizado, incorporando un modelo de dominio enriquecido que encapsule lógica de negocio.
3. **Diseñar una jerarquía de pedidos escalable:** emplear estrategias de herencia en JPA que optimicen las consultas y faciliten la extensión del modelo.
4. **Separar el modelo interno de la interfaz expuesta:** estructurar la comunicación API mediante DTOs y *mappers* automatizados de *MapStruct* que aseguren conversiones eficientes.
5. **Implementar un sistema de seguridad robusto:** configurar *Spring Security* para gestionar la autenticación y una autorización granular, utilizando un modelo de roles y permisos que permita un control de acceso detallado a los recursos de la API.
6. **Asegurar la calidad y robustez del backend:** implementar un sistema completo de validación de entradas, manejo centralizado de excepciones y un conjunto de pruebas automatizadas (unitarias y de integración) para los componentes clave de la API (servicios, *mappers* y lógica de negocio).
7. **Establecer un flujo de Integración Continua (CI):** configurar un *workflow* con *GitHub Actions* que automatice la ejecución de pruebas e integre el análisis de código con *SonarCloud* para monitorizar métricas de calidad, seguridad y fiabilidad de forma continua.
8. **Construir una interfaz de usuario moderna, reactiva e internacionalizable:** desarrollar el *frontend* con componentes reutilizables, gestionar el estado de la aplicación de forma eficiente e implementar el soporte a varios idiomas (español/inglés).
9. **Garantizar la portabilidad y reproducibilidad del despliegue:** utilizar *Docker* para contenerizar todos los servicios de la aplicación (base de datos, *backend* y *frontend*), creando un entorno de ejecución consistente e independiente de la infraestructura subyacente.
10. **Aplicar metodologías ágiles y control de versiones:** gestionar el proyecto mediante *sprints*, utilizando *Git* con *GitHub Flow* para mantener un historial de cambios claro, trazable y semántico.

3. Conceptos teóricos

3.1. *Nutri-Score*

El *Nutri-Score* es un sistema de etiquetado frontal que clasifica la calidad nutricional de los alimentos en una escala de cinco clases representadas por letras y colores, desde la “A” (verde) hasta la “E” (rojo), siendo “A” la mejor calidad nutricional [32]. Su propósito es ofrecer al consumidor una valoración rápida y comparativa del perfil nutricional de un producto, facilitando decisiones alimentarias informadas.

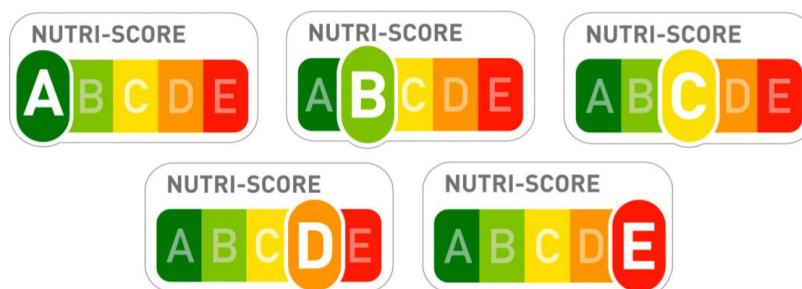


Figura 3.1: Etiquetas A-E según *Nutri-Score*

El resultado numérico (*score*) se obtiene por un sistema de puntos que combina “puntos negativos” y “puntos positivos”, calculados por 100 g o 100 ml del producto. Los puntos negativos penalizan contenidos elevados de energía (kJ), azúcares disponibles, ácidos grasos saturados y sodio, mientras que los puntos positivos recompensan la presencia de fruta/verdura/frutos secos/legumbres, fibra y proteínas. El *score* final se calcula restando los puntos positivos de los negativos y se asigna a una letra A–E mediante umbrales que pueden variar según la categoría del alimento (p. ej. bebidas,

quesos o grasas añadidas), existiendo adaptaciones específicas para dichas familias. Para el cálculo exacto y las adaptaciones más recientes, existen calculadoras y documentación técnica pública [33].

3.2. Clasificación NOVA

La clasificación NOVA organiza los alimentos según el grado y el propósito del procesamiento industrial, en lugar de centrarse únicamente en su composición nutricional.

NOVA pretende capturar efectos asociados al grado de procesamiento (por ejemplo patrones de consumo y riesgos poblacionales) y no reemplaza otros índices nutricionales. Por ejemplo, un producto ultraprocesado puede tener un perfil nutricional “mejor” según ciertos nutrientes, pero sigue perteneciendo al Grupo 4 por su naturaleza y propósito tecnológico.



Figura 3.2: Etiquetas 1-4 según clasificación NOVA

La clasificación consta de cuatro grupos principales [24]:

1. **Grupo 1 - Alimentos sin procesar o mínimamente procesados:** alimentos comestibles en su forma natural o sometidos a procesos simples (lavado, corte, pasteurización).
2. **Grupo 2 - Ingredientes culinarios procesados:** sustancias extraídas o refinadas de alimentos (aceites, mantequillas, harinas, azúcares) destinadas a cocinar o condimentar.
3. **Grupo 3 - Alimentos procesados:** productos sencillos elaborados con ingredientes del grupo 1 y 2 (conservas, quesos, panes simples).
4. **Grupo 4 - Alimentos ultraprocesados (UPF¹):** formulaciones industriales con numerosos ingredientes (azúcares, aceites refinados, aditivos, sustancias poco usadas en la cocina doméstica) y procesos tecnológicos que tienden a producir productos agradables al gusto, de larga duración y altamente transformados.

¹*Ultra-Processed Foods*

3.3. REGULACIÓN DE ALÉRGENOS (REGLAMENTO UE 1169/2011)

3.3. Regulación de alérgenos (Reglamento UE 1169/2011)

En la Unión Europea, la normativa sobre información alimentaria al consumidor (Reglamento (UE) nº 1169/2011 [47]) establece una lista de **14 alérgenos** cuya presencia en ingredientes debe identificarse de forma clara en el etiquetado de los alimentos destinados al consumidor final. Esta obligación se aplica tanto a los ingredientes utilizados como tal, como a los derivados que contengan alérgenos, y exige que los alérgenos sean fácilmente identificables en la lista de ingredientes (por ejemplo, destacándolos en negrita).



Figura 3.3: Catorce alérgenos de declaración obligatoria de la UE

La lista, en formato resumido, de ingredientes que deben ser declarados como alérgenos es:

1. Cereales que contienen gluten (p. ej. trigo, centeno, cebada, avena).
2. Crustáceos.
3. Huevos.
4. Pescado.
5. Cacahuete.
6. Soja.
7. Leche (incluida la lactosa).
8. Frutos de cáscara (nueces, almendras, avellanas, etc.).
9. Apio.
10. Mostaza.
11. Sésamo (semillas de sésamo).
12. Dióxido de azufre y sulfitos (si su concentración es > 10 mg/kg o 10 mg/l expresados como SO₂).
13. Altramuces.
14. Moluscos.

4. Técnicas y herramientas

En este capítulo se describen las metodologías y herramientas tecnológicas escogidas para el desarrollo del trabajo, detallando los aspectos principales y justificando los motivos de su selección.

4.1. Metodologías y técnicas de gestión

Scrum

Scrum [10] es un marco de trabajo ágil orientado a la gestión y desarrollo de proyectos complejos, especialmente en el ámbito del desarrollo de *software*. Se basa en un enfoque iterativo e incremental, en el que el trabajo se organiza en ciclos de duración fija denominados *sprints*. Durante cada *sprint* se selecciona un conjunto de requisitos priorizados que deben ser diseñados, implementados y validados.

Scrum define una serie de **roles**, **eventos** y **artefactos** que facilitan la planificación, la inspección continua del progreso y la adaptación a cambios en los requisitos, promoviendo la entrega frecuente de incrementos funcionales del producto.

En el marco de este trabajo, se adaptó esta metodología a las particularidades de un desarrollo académico individual. Para ello, se preservaron elementos fundamentales como la duración de los *sprints* (1-2 semanas) y las reuniones de revisión tras finalizar cada iteración, destinadas a evaluar los avances logrados y planificar las siguientes mejoras.

Kanban

Kanban [30] es una técnica de gestión visual del trabajo cuyo objetivo principal es optimizar el flujo de tareas y mejorar la eficiencia del proceso de desarrollo. Se basa en la representación gráfica del estado de las tareas mediante un tablero dividido en columnas que reflejan las distintas fases del proceso (p. ej. “*In progress*” o “*Done*”). En la figura 4.3 se puede observar el tablero Kanban utilizado durante el desarrollo de este proyecto.

Cada tarea se representa como un elemento que se desplaza entre estados a medida que avanza su ejecución. Kanban pone especial énfasis en la limitación del trabajo en curso (*Work In Progress*), la identificación de cuellos de botella y la mejora continua del proceso, sin imponer iteraciones temporales cerradas ni roles específicos.

Rich Domain Model

El *Rich Domain Model* o Modelo Enriquecido [48] es un enfoque de diseño propio de la arquitectura orientada a objetos y del *Domain-Driven Design* [12], en el que las entidades del dominio encapsulan tanto los datos como el comportamiento asociado a los mismos.

A diferencia de los **modelos anémicos**, en los que las entidades actúan únicamente como contenedores de información, un modelo de dominio enriquecido incluye lógica de negocio relevante dentro de las propias clases del dominio. Este enfoque favorece una mayor cohesión, una mejor representación del dominio del problema y una distribución más natural de las responsabilidades dentro del sistema.

Domain-Driven Design (DDD)

El *Domain-Driven Design* [12] es un enfoque de diseño de *software* orientado a la construcción de sistemas complejos a partir de un modelo de dominio rico y expresivo, alineado con el conocimiento del problema que se desea resolver. *DDD* propone centrar el diseño en el dominio del negocio, fomentando una comunicación constante entre expertos del dominio y desarrolladores, utilizando conceptos como modelos de dominio, agregados, entidades, objetos de valor y contextos delimitados (*bounded contexts*). El objetivo principal es lograr un diseño coherente, mantenable y adaptable a la evolución de los requisitos.

En el desarrollo de este proyecto se han adoptado algunos principios y conceptos inspirados en el *Domain-Driven Design*, sin aplicar la metodología

de forma estricta o completa. En particular, se ha priorizado una organización del código orientada a funcionalidades o dominios concretos del sistema (directorios `user`, `food`, `product`...), en lugar de una estructura clásica por capas (directorios `controller`, `service`, `repository`...). También se ha favorecido que las entidades del dominio encapsulen comportamientos y responsabilidades propias, especialmente en la gestión de sus relaciones. No obstante, dado el alcance y complejidad moderada del proyecto, no se ha llevado a cabo una implementación exhaustiva de todos los patrones y prácticas formales de *DDD*, optando por una aplicación selectiva y funcional, adaptada al contexto del trabajo.

4.2. Patrón de arquitectura

La aplicación sigue una arquitectura **Cliente-Servidor** con un *frontend* y un *backend* claramente desacoplados. El *frontend*, desarrollado en *React*, se estructura en componentes reutilizables, páginas y servicios modulares que consumen la API mediante un cliente centralizado.

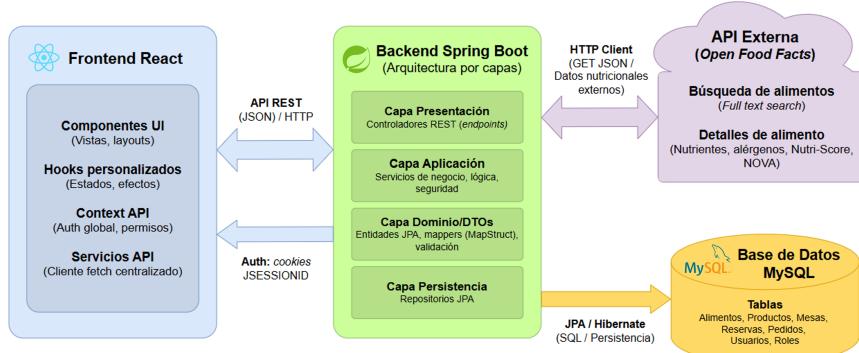


Figura 4.1: Arquitectura de *Orderly*

El *backend* implementa una **Arquitectura en Capas** (*Layered Architecture*) de cuatro niveles, la cual se puede visualizar en la figura 4.1:

1. **Presentación:** controladores REST finos, responsables de manejar las peticiones HTTP y las respuestas.
2. **Aplicación:** servicios encargados de la orquestación de operaciones y control transaccional.
3. **Dominio:** entidades JPA con comportamiento (modelo rico) que se relacionan entre sí.
4. **Persistencia:** repositorios JPA que abstraen *MySQL*.

4.3. Tecnologías y *frameworks* principales

Backend

Spring Boot

Spring Boot [2] es un *framework* del ecosistema *Spring* [38] diseñado para simplificar el desarrollo de aplicaciones *backend* basadas en Java. Proporciona configuración automática, un sistema de dependencias gestionado y un servidor embebido, lo que permite crear aplicaciones listas para producción con un esfuerzo de configuración mínimo. Está especialmente orientado al desarrollo de servicios web y APIs REST.

Spring Data JPA

Spring Data JPA [37] es un módulo de *Spring* [38] que facilita el acceso y la persistencia de datos mediante la especificación **JPA** (*Java Persistence API*).

JPA define un estándar para el mapeo objeto-relacional (ORM), permitiendo representar tablas de bases de datos como entidades Java. *Spring Data JPA* abstrae gran parte del código repetitivo asociado al acceso a datos, proporcionando repositorios y mecanismos automáticos de generación de consultas.

Spring Security

Spring Security [39] es un *framework* orientado a la gestión de la seguridad en aplicaciones Java. Proporciona mecanismos para la autenticación, autorización y protección frente a ataques comunes o accesos no autorizados. Se integra de forma nativa con *Spring Boot* y permite definir políticas de seguridad de manera flexible y extensible.

Spring Validation

Spring Validation [46] es un módulo que permite validar datos de entrada mediante anotaciones declarativas. Se basa en la especificación **Bean Validation** y se utiliza habitualmente para comprobar la validez de los datos recibidos en peticiones, asegurando el cumplimiento de restricciones como valores obligatorios, rangos numéricos o formatos específicos.

En nuestra aplicación, los datos que se validan con estas anotaciones son DTOs (*Data Transfer Objects*) de petición, es decir, los objetos que

toman los controladores como valor de entrada. De esta manera, se pueden definir restricciones rápidas sobre cada campo para liberar a los servicios de comprobaciones exhaustivas.

Lombok

Lombok [1] es una biblioteca que reduce la cantidad de código repetitivo (“boilerplate”) en aplicaciones Java mediante el uso de anotaciones. Permite generar automáticamente métodos comunes como *getters*, *setters*, constructores o métodos `equals` y `hashCode` durante el proceso de compilación, mejorando la legibilidad y mantenibilidad del código.

MapStruct

MapStruct [55] es una herramienta de mapeo de objetos que permite transformar de forma automática y segura objetos de un tipo a otro, como entidades y DTOs. Genera código en tiempo de compilación, lo que ofrece un alto rendimiento y evita errores en tiempo de ejecución, manteniendo una separación clara entre las distintas capas de la aplicación.

MySQL

MySQL [52] es un sistema de gestión de bases de datos relacional ampliamente utilizado en aplicaciones empresariales y web. Se caracteriza por su rendimiento, fiabilidad y compatibilidad con múltiples plataformas. En el contexto de aplicaciones *Spring Boot*, se emplea junto con el controlador JDBC correspondiente para permitir la comunicación entre la aplicación y la base de datos [36].

Frontend

React

React [13] es una biblioteca de JavaScript para la construcción de interfaces de usuario basadas en componentes reutilizables. Se centra en la creación de vistas declarativas y en la gestión eficiente del estado y la actualización del **DOM** (*Document Object Model*) mediante un modelo de renderizado reactivo.

Su elección se justifica frente a alternativas como *Thymeleaf* [43] por las necesidades específicas del proyecto: la gestión de un elevado número de entidades (productos, pedidos, reservas, usuarios, etc.) y la exigencia de una interfaz dinámica y reactiva. El paradigma basado en **componentes**

y la **arquitectura SPA** (*Single Page Application*) de *React* permiten un desarrollo más modular, mantenable y con mejor experiencia de usuario [53].

Pese a no haber utilizado *React* previamente, el análisis del proyecto de referencia *NutriMenu* [56] permitió evaluar su viabilidad y agilizar el aprendizaje inicial. Su adopción supuso, por lo tanto, un reto formativo para dominar una herramienta estándar en el sector.

Vite

Vite [50] es una herramienta de construcción y desarrollo *frontend* que proporciona un entorno de desarrollo rápido y optimizado. Utiliza un servidor de desarrollo basado en **módulos ES** y genera *builds* de producción altamente eficientes, reduciendo significativamente los tiempos de arranque y recarga.

Mantine

Mantine [21] es una biblioteca de componentes para *React* que proporciona elementos de interfaz modernos, accesibles y personalizables. Incluye componentes visuales, utilidades de estilo y *hooks* que facilitan el desarrollo de interfaces coherentes y funcionales.

React Router

React Router [14] es una biblioteca de enrutamiento para aplicaciones *React* que permite gestionar la navegación entre distintas vistas sin recargar la página. Facilita la creación de aplicaciones de una sola página mediante rutas declarativas y dinámicas.

i18next y react-i18next

i18next [15] es un *framework* de internacionalización que permite gestionar traducciones y contenido multilingüe en aplicaciones *frontend*. La librería *react-i18next* [16] proporciona integración específica con *React*, facilitando la adaptación dinámica de la interfaz al idioma seleccionado por el usuario.

Recharts

Recharts [31] es una biblioteca de visualización de datos para *React* basada en componentes reutilizables. Permite la creación de gráficos interactivos y personalizables a partir de datos estructurados, facilitando la representación visual de información compleja dentro de aplicaciones web.

Se ha utilizado esta librería para representar la información nutricional de los productos mediante gráficos interactivos, concretamente gráficos de tipo donut anidado. Estos gráficos permiten visualizar de forma intuitiva la distribución de los distintos valores nutricionales y mostrar información detallada al interactuar con ellos, como se observa en el ejemplo presente en la figura 4.2.



Figura 4.2: Gráfico nutricional de un producto usando *Recharts*

Zustand

Zustand [54] es una biblioteca ligera para la gestión del estado global en aplicaciones *React*. Se caracteriza por su simplicidad, bajo acoplamiento y uso de *hooks*, permitiendo compartir estado entre componentes sin la complejidad de soluciones más pesadas.

API de información nutricional: *Open Food Facts*

Open Food Facts (OFF) [11] es una base de datos abierta y colaborativa que ofrece información detallada sobre productos alimentarios de todo el mundo a través de una API REST. La información proporcionada incluye datos generales del producto, información nutricional estandarizada, listas de ingredientes y la identificación de alérgenos, lo que la convierte en una fuente especialmente útil para aplicaciones del ámbito de la restauración y la nutrición.

Las principales **ventajas** que ofrece *Open Food Facts* son las siguientes:

- Amplio catálogo de productos, con millones de alimentos registrados y disponibilidad en múltiples idiomas, lo que facilita su uso en aplicaciones multilingües como la nuestra.

- Búsqueda mediante texto libre (*full-text search*), que permite localizar productos a partir de su nombre, etiquetas, categorías o características nutricionales.
- Carácter abierto y acceso gratuito, sin restricciones estrictas en el número de peticiones a la API.
- Información detallada sobre alérgenos y métricas nutricionales. Este aspecto hace destacar a *OFF* respecto al resto de opciones y fue decisivo para escogerla como API nutricional.

No obstante, el uso de *Open Food Facts* también presenta algunas **limitaciones**:

- Presencia de ruido en las búsquedas, debido al gran volumen y heterogeneidad de los datos, lo que puede dificultar la localización de productos concretos.
- Ausencia de alimentos genéricos o no envasados, como frutas, verduras u otros productos comunes que no disponen de código de barras.
- Inconsistencias en la completitud de la información, ya que, al tratarse de una base de datos colaborativa, algunos productos presentan campos incompletos o ausentes.

***Testing* y calidad del software**

JUnit

JUnit [44] es un *framework* de pruebas unitarias para Java que permite verificar el correcto funcionamiento de componentes individuales del sistema. Facilita la automatización de pruebas y la detección temprana de errores durante el desarrollo.

Mockito

Mockito [45] es una biblioteca de apoyo para pruebas que permite crear objetos simulados (*mocks*) con el fin de aislar componentes y controlar su comportamiento durante la ejecución de las pruebas. Es especialmente útil para probar clases que dependen de otros componentes del sistema.

Spring Boot Test

Spring Boot Test es un módulo que proporciona soporte específico para la realización de pruebas en aplicaciones *Spring Boot*. Permite cargar contextos

de aplicación, simular peticiones HTTP y realizar pruebas de integración de forma controlada [40].

H2

H2 [18] es un sistema de base de datos relacional en memoria utilizado habitualmente en entornos de pruebas. Permite ejecutar pruebas de acceso a datos sin depender de una base de datos externa, garantizando rapidez y aislamiento en los tests.

Contenedores y despliegue

Docker

Docker [8] es una plataforma de contenerización que permite empaquetar aplicaciones junto con sus dependencias en contenedores ligeros y portables. Esto garantiza que la aplicación se ejecute de forma consistente en diferentes entornos, facilitando el despliegue y la reproducibilidad.

Docker Compose

Docker Compose [7] es una herramienta que permite definir y gestionar aplicaciones compuestas por múltiples contenedores mediante archivos de configuración declarativos. Facilita la orquestación de servicios, la definición de redes y volúmenes, y el despliegue conjunto de todos los componentes de una aplicación.

Automatizaciones

GitHub Actions

GitHub Actions [6] es una herramienta de automatización e integración continua integrada en *GitHub* que permite definir flujos de trabajo (*workflows*) basados en eventos del repositorio, como *pushes*, *pull requests* o cierres de *issues*. Mediante archivos de configuración declarativos, permite automatizar tareas como la ejecución de pruebas, el análisis de código, la construcción de aplicaciones o la ejecución de herramientas externas.

SonarCloud

SonarCloud (*SonarQube Cloud*) [34] es una plataforma de análisis estático de código basada en la nube que permite evaluar automáticamente la calidad y seguridad del *software*. Se integra en el flujo de integración continua para

detectar errores, vulnerabilidades y deuda técnica, proporcionando métricas detalladas sobre la mantenibilidad y la cobertura de pruebas del proyecto.

4.4. Herramientas de desarrollo y documentación

Control de versiones

GitHub

GitHub [4] es una plataforma de alojamiento de repositorios que se basa en el sistema de control de versiones distribuido *Git*. Permite gestionar el código fuente de un proyecto, registrar su evolución histórica y facilitar la colaboración mediante funcionalidades como repositorios remotos, gestión de ramas, control de incidencias (*issues*) y solicitudes de integración (*pull requests*). *Git*, como sistema subyacente, permite realizar un seguimiento detallado de los cambios y trabajar de forma descentralizada.

GitHub Projects

GitHub Projects [5] es una herramienta integrada en la plataforma *GitHub* orientada a la planificación y gestión del trabajo mediante tableros visuales. Permite organizar tareas a través de vistas tipo Kanban, listas o tablas, vinculando elementos como *issues* y *pull requests* procedentes de uno o varios repositorios.

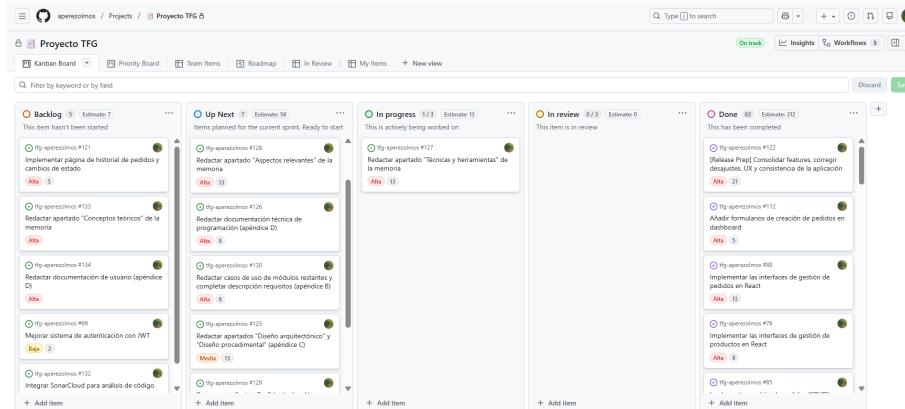


Figura 4.3: Tablero Kanban de *GitHub Projects*

GitHub Projects añade una capa de gestión independiente del repositorio, facilitando la priorización, el seguimiento del progreso y la visualización del estado del proyecto. Además, incorpora mecanismos de automatización (*workflows*) que permiten realizar acciones automáticas en función de eventos, como actualizar el estado de una tarea al cerrarse un *issue* o al cambiar su posición dentro del tablero. La herramienta también ofrece métricas y gráficos, como diagramas de progreso (*burn-up*), que ayudan a analizar la evolución del trabajo a lo largo del tiempo.

GitHub Flow

GitHub Flow es un flujo de trabajo ligero para la gestión de ramas en proyectos controlados con *Git*. Se basa en una rama principal (*main*) estable y en la creación de ramas independientes (*/feature/x*) para el desarrollo de nuevas funcionalidades, correcciones o mejoras (en nuestro caso, se ha creado una rama por cada *issue*). Una vez completado el trabajo en una rama, ésta se integra nuevamente en la rama principal mediante un proceso de revisión y fusión.

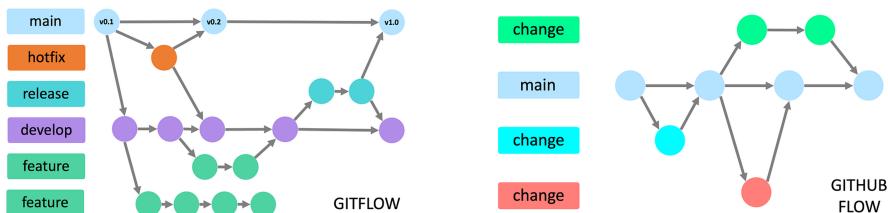


Figura 4.4: Comparativa de ramas entre *GitFlow* y *GitHub Flow* [42]

A diferencia de metodologías más complejas como *Git Flow*, que define múltiples ramas permanentes (p. ej. *develop*, *release* o *hotfix*), *GitHub Flow* reduce la complejidad del proceso y resulta especialmente adecuado para proyectos de tamaño reducido o desarrollo individual, donde se prioriza la simplicidad y la trazabilidad de los cambios.

Conventional Commits

Conventional Commits [3] es una convención para la redacción de mensajes de *commit* que define una estructura estandarizada basada en un prefijo que indica el tipo de cambio realizado (p. ej. *feat*, *fix*, *refactor*). Este enfoque mejora la legibilidad del historial de cambios y facilita la comprensión de la evolución del proyecto.

La estructura de un *commit* según esta convención es la siguiente:

```
<tipo>[alcance (opcional)]: <descripción>
```

Entorno de desarrollo

Visual Studio Code

Visual Studio Code [51] es un editor de código fuente ligero y multiplataforma ampliamente utilizado en el desarrollo de *software*. Destaca su amplio ecosistema de extensiones que permiten adaptarlo a distintos lenguajes y tecnologías. También resulta muy útil su integración nativa con *Git*, que permite gestionar el control de versiones directamente desde la interfaz del editor.

Para este proyecto se ha hecho uso del *Extension Pack for Java*, un conjunto de herramientas que proporciona soporte avanzado para el lenguaje. Incluye autocompletado inteligente, navegación por el código, refactorización, ejecución y depuración de aplicaciones, así como integración con herramientas de construcción y pruebas.

A pesar de que existen entornos específicos para el desarrollo con Java, como *IntelliJ IDEA* [17], se optó por *VS Code* debido a la familiaridad con su flujo de trabajo tras su uso continuado durante el grado. Gracias al mencionado sistema de extensiones, este editor ofrece una experiencia equiparable en potencia y funcionalidad.

Postman

Postman [28] es una herramienta de desarrollo que permite diseñar, ejecutar y analizar peticiones HTTP a servicios web, especialmente APIs REST. Facilita el envío de solicitudes con distintos métodos (**GET**, **POST**, **PUT**, **DELETE**, etc.), la configuración de cabeceras y cuerpos de petición, y la visualización detallada de las respuestas devueltas por el servidor.

Postman se utiliza habitualmente para verificar el comportamiento de los *endpoints*, validar respuestas y apoyar el proceso de desarrollo y depuración de APIs. En este proyecto, se ha usado principalmente para validar la lógica de la API desarrollada con *Spring Boot*, así como para testear las respuestas provenientes de las APIs nutricionales externas, facilitando así su posterior integración en el sistema.

Documentación

LaTeX

\LaTeX [19] es un sistema de composición tipográfica orientado a la elaboración de documentos técnicos y científicos. Permite separar el contenido del formato, facilitando la creación de documentos estructurados y coherentes, especialmente adecuados para trabajos académicos extensos.

TeXstudio

TeXstudio [41] es un entorno de desarrollo integrado para \LaTeX que proporciona herramientas como edición asistida, autocompletado de comandos, compilación integrada y visualización del documento resultante. Facilita la redacción y revisión de documentos complejos en \LaTeX de forma local.

La herramienta escogida inicialmente fue *Overleaf* [26], con el fin de aprovechar sus funciones de colaboración y revisión remota. Sin embargo, a medida que aumentaba la extensión y complejidad de la documentación, se terminó excediendo el *compile timeout* permitido en su [plan gratuito](#). Es por ello que se trasladó el flujo de trabajo al entorno local mediante, manteniendo la documentación sincronizada tanto en el repositorio de *Github* como en la plataforma de *Overleaf*. De este modo, el tutor podría disponer siempre del documento más reciente para su revisión.

MiKTeX

MiKTeX [23] es una distribución ligera de \LaTeX que incluye los paquetes y herramientas necesarios para compilar documentos. Se caracteriza por su sistema de gestión de paquetes bajo demanda, lo que permite instalar únicamente los componentes necesarios, reduciendo el tamaño de la instalación. Esta particularidad motivó la elección de MiKTeX frente a otras distribuciones como TeXLive [20], la cual realiza instalaciones completas de forma predeterminada.

5. Aspectos relevantes del desarrollo del proyecto

Este capítulo recoge los aspectos más importantes del desarrollo del proyecto. Engloba la descripción de los componentes implementados, la justificación de las decisiones tomadas y la diferenciación de las fases del desarrollo.

5.1. Inicio del proyecto

La idea del proyecto surgió a partir de mi familiaridad con el flujo de trabajo en locales de restauración y el potencial de escalabilidad detectado, permitiendo futuras ampliaciones como la gestión de inventario o el análisis de costes, entre otros. Asimismo, el desarrollo se planteó como una oportunidad para consolidar las competencias adquiridas durante la carrera y profundizar en tecnologías modernas de alta demanda en el mercado actual.

En la primera reunión con el tutor, se le explicaron los objetivos generales y las funcionalidades mínimas que se pretendía que abarcara la aplicación. Se acordó entonces introducir aspectos que contribuyeran a la diferenciación de la aplicación respecto de herramientas comunes de gestión. Fue ahí donde el tutor propuso tomar como referencia un Trabajo de Fin de Grado anterior, elaborado por Álvaro Manjón Vara. En dicho proyecto se desarrolló una aplicación web de nombre *NutriMenu* [56], dirigida a los centros de restauración de la Universidad de Burgos, cuya principal funcionalidad era la creación de menús con información nutricional de cada plato, obtenida a partir de una API externa.

Se dio la circunstancia de que el *stack* tecnológico sugerido al tutor coincidía en gran medida con el del TFG mencionado. Aún así, no se optó por utilizar como base dicha aplicación, sino que se partió de un desarrollo desde cero, puesto que tanto el modelo de datos como la lógica de negocio iban a verse modificados significativamente y nuevos módulos de gestión iban a ser añadidos.

5.2. Desarrollo del *backend*

Funcionalidades principales y módulos

La API REST desarrollada cubre los principales procesos de gestión de un local de restauración, agrupados en los siguientes módulos funcionales:

- **Gestión de pedidos (comandas):** soporta la creación y seguimiento de pedidos realizados en el local, diferenciando entre pedidos de barra y de comedor mediante una jerarquía de herencia.
- **Gestión de productos:** permite la creación, edición, consulta y eliminación de productos ofertados, así como la gestión de sus ingredientes y el cálculo dinámico de su información nutricional.
- **Gestión de alimentos e ingredientes:** facilita la administración de alimentos y su información nutricional, permitiendo su uso como ingredientes en productos.
- **Gestión de reservas de mesas:** permite la reserva y administración de mesas, incluyendo la gestión de disponibilidad y la prevención de solapamientos.
- **Gestión de mesas:** administra las mesas disponibles en el local, su capacidad y estado.
- **Gestión de usuarios:** administra las cuentas de acceso al sistema, permitiendo la asignación de roles específicos y el control de los datos personales del personal.
- **Gestión de roles:** define los perfiles de acceso mediante agrupaciones de permisos granulares, garantizando que cada usuario solo acceda a las funcionalidades autorizadas según su responsabilidad.

Cada módulo está implementado siguiendo el patrón de capas descrito en la sección [4.2 Patrón de arquitectura](#), lo que facilita la mantenibilidad y la extensibilidad del sistema.

Modelo de datos y relaciones

El modelo de datos, específicamente en su módulo de alimentos, toma como referencia el esquema de *NutriMenu* [56], el cual puede visualizarse por completo en la figura 5.1. Se utiliza una base de datos ***MySQL*** para asegurar la consistencia relacional de la información. La persistencia se gestiona mediante repositorios que extienden de **JpaRepository**, lo que permite aprovechar las capacidades de *Spring Data JPA* para la consulta y manipulación de datos. Al igual que en el proyecto de referencia, se emplea un **enfoque híbrido**, el cual combina el almacenamiento local de alimentos creados manualmente con la obtención de datos externos provenientes de la API de *Open Food Facts* [11].

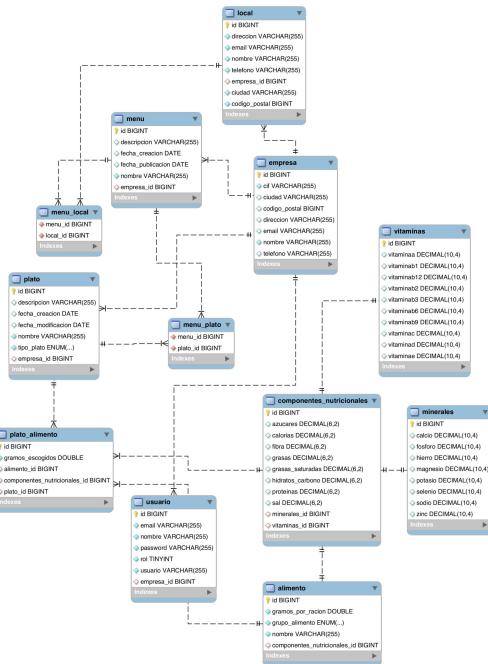


Figura 5.1: Modelo de base de datos de *NutriMenu* [56]

A pesar de compartir el planteamiento conceptual con el proyecto de referencia, se han introducido **mejoras** significativas en la arquitectura de datos y la lógica de negocio para optimizar el **rendimiento** y la **mantenibilidad** del sistema. Las modificaciones más relevantes son:

- **Simplificación mediante objetos embebidos:** a diferencia de *NutriMenu*, que gestionaba la información nutricional (vitaminas, minerales, etc.) como entidades independientes, en este trabajo se han

implementado como objetos embebidos (`@Embeddable`). Esto elimina la necesidad de tablas adicionales y relaciones complejas (*joins*), permitiendo que toda la información nutricional se persista en una única tabla de alimentos, lo que mejora la eficiencia de las operaciones en la base de datos.

- **Cálculo dinámico de información nutricional y consistencia:** se ha sustituido la actualización manual de valores nutricionales por un sistema de cálculo dinámico. Mientras que en el proyecto anterior los cálculos eran verbosos y se duplicaban en la base de datos, este modelo utiliza el principio “*Single Source of Truth*” [29], recalculando los valores en tiempo real a partir del alimento original. Esto asegura que cualquier cambio en un ingrediente se refleje automáticamente en todos los productos asociados.
 - **Inclusión de información de alérgenos y métricas nutricionales por alimento:** se ha enriquecido el modelo mediante la integración de campos específicos para alérgenos y clasificaciones nutricionales estándar como *Nutri-Score* y NOVA. Al aprovechar los datos proporcionados por la API de *Open Food Facts*, el sistema ofrece una visión más profunda sobre la calidad y seguridad de los alimentos.

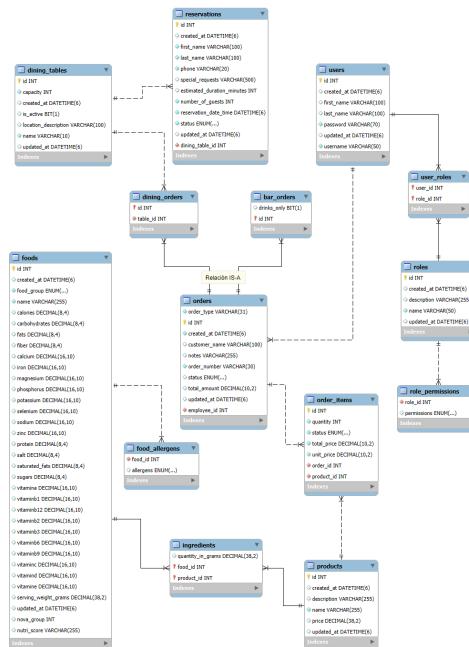


Figura 5.2: Modelo de base de datos actual

Para una descripción técnica más detallada sobre la implementación de estas entidades y la lógica de negocio aplicada, se recomienda consultar los *pull requests* #51 y #52 en el repositorio del proyecto.

Lógica de negocio y servicios

La lógica de negocio se encapsula en servicios específicos para cada módulo, que se encargan de validar las operaciones, gestionar las transacciones y aplicar las reglas del dominio. Por ejemplo:

- El servicio de **productos** calcula la información nutricional total de cada producto en función de sus ingredientes, utilizando métodos de dominio y operaciones sobre objetos de tipo `NutritionInfo`.
- El servicio de **reservas** valida la disponibilidad de mesas y previene solapamientos mediante la comprobación de conflictos de horario antes de confirmar una reserva.
- El servicio de **pedidos** gestiona de forma transparente la creación y actualización de comandas, independientemente del tipo de pedido. Esto se logra mediante una jerarquía de entidades (`Order`, `BarOrder`, `DiningOrder`) y DTOs con herencia, combinada con una factoría de *mappers* que selecciona dinámicamente la estrategia de conversión apropiada. Además, cada tipo de pedido dispone de un servicio y controlador específicos para consultas especializadas sobre las tablas hijas, optimizando el rendimiento al usar la estrategia *joined inheritance*.

La separación de responsabilidades permite mantener la lógica de negocio aislada de los puntos de entrada de la API, facilitando la reutilización y el testeo de los componentes. Este diseño ha favorecido la implementación de **controladores finos** (*thin controllers*), esto es, controladores que se mantienen ligeros al delegar toda la complejidad y las reglas de dominio en los servicios correspondientes.

Comunicación entre capas: DTOs y *mappers*

La comunicación entre las distintas capas de la aplicación se realiza mediante objetos de transferencia de datos, diferenciando entre DTOs de **petición** (`RequestDTO`) y de **respuesta** (`ResponseDTO`). Esta distinción permite adaptar los datos expuestos por la API a las necesidades de cada

operación, facilitando la validación de los datos recibidos (ver apartado *Manejo de excepciones y validaciones*) y ocultando detalles internos o sensibles. Por ejemplo, la contraseña de un usuario, aunque se almacena encriptada, no se proporciona en el DTO de respuesta.

La conversión entre entidades y DTOs se realiza mediante *mappers* implementados con *MapStruct* (ver diagrama de comunicación en la figura 5.3), lo que permite definir reglas de mapeo declarativas y reducir la cantidad de código repetitivo. En casos complejos, como la jerarquía de pedidos, se emplea una factoría de *mappers* para aislar la lógica de conversión y facilitar la extensión a nuevos tipos de pedidos.

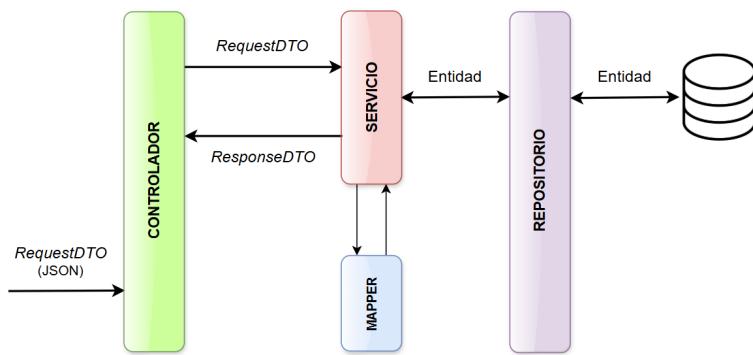


Figura 5.3: Diagrama de comunicación entre capas

Seguridad y control de acceso

La seguridad y el control de acceso en el *backend* se gestionan mediante *Spring Security*, implementando un sistema de autenticación basado en sesiones HTTP y **autorización granular por roles y permisos** (ver *pull request #81*). Los *endpoints* de la API están protegidos mediante anotaciones `@PreAuthorize`, que verifican los permisos necesarios para cada operación, garantizando que solo los usuarios autorizados puedan acceder a los recursos sensibles.

La gestión de la **autenticación** se centraliza en una capa de servicios que valida las credenciales y, tras un inicio de sesión o registro exitoso, vincula el contexto de seguridad a la sesión del usuario. Dicha sesión se mantiene de forma **persistente** en el *backend* mediante el componente `HttpSessionSecurityContextRepository`, permitiendo que el sistema identifique al usuario en peticiones sucesivas a través de una *cookie* de sesión (`JSESSIONID`). Para la representación de usuarios en el sistema de seguridad,

se utiliza la clase `CustomUserDetails`, que adapta la entidad `User` y expone sus roles y permisos como *authorities* de *Spring Security*, permitiendo así un control de acceso detallado a los recursos.

Manejo de excepciones y validaciones

El *backend* implementa un **sistema centralizado de manejo de excepciones** mediante el componente `GlobalExceptionHandler`, que captura y gestiona los errores más frecuentes (entidades no encontradas, violaciones de integridad, errores de validación, etc.). Los errores se devuelven en un formato estructurado que facilita su tratamiento en el *frontend*.



Figura 5.4: Ejemplos de errores de la API que utilizan el DTO `ErrorResponse`

Las validaciones de los datos de entrada se realizan tanto a nivel de DTOs, utilizando anotaciones estándar de *Java Validation* (`@NotNull`, `@Size`, `@Min`, etc.), como a nivel de lógica de negocio en los servicios.

5.3. Desarrollo del *frontend*

El desarrollo de la interfaz de usuario comenzó inicialmente con *Thymeleaf*, integrado en el proyecto *Spring Boot*, aprovechando la experiencia adquirida durante la carrera, particularmente en la asignatura de *Sistemas Distribuidos*. Sin embargo, tras una primera fase de exploración, se migró la capa de presentación a ***React***, como se justifica en la sección [4.3 Tecnologías y frameworks principales](#).

La navegación entre las distintas secciones de la aplicación se gestiona mediante ***React Router***, permitiendo una experiencia de usuario fluida y sin recargas de página. Se han definido **rutas protegidas** para aquellas

secciones que requieren autenticación o permisos específicos, utilizando el componente `ProtectedRoute`.

Arquitectura de componentes y experiencia de usuario

Para sentar las bases visuales y acelerar el desarrollo, se utilizó la biblioteca ***Mantine***. A diferencia de otros marcos de diseño rígidos, *Mantine* proporciona componentes de bajo nivel altamente **personalizables** y accesibles (como formularios o modales), lo que ha permitido diseñar una identidad visual propia sin renunciar a la robustez de elementos probados. Esta flexibilidad facilitó la creación de una interfaz limpia y **responsiva**, con soporte nativo para temas claros y oscuros. Sobre esta base de componentes customizados, se desarrollaron estructuras como `MainLayout`, que agrupa elementos comunes como la barra de navegación, y `FormLayout`, un esquema unificado para los formularios de creación y edición de todas las entidades.

Asimismo, para garantizar una experiencia consistente ante la naturaleza **asíncrona** de las peticiones a la API, la interfaz debe responder adecuadamente ante posibles demoras o errores. Por ello, se integró de forma sistemática un sistema de *feedback* que informa al usuario en todo momento: **indicadores de carga** (`LoadingOverlay`) durante las operaciones, mensajes informativos cuando no hay datos para mostrar, alertas en formularios y notificaciones de éxito o error (mediante `@mantine/notifications`) con mensajes contextualizados.

Gestión de datos y operaciones sobre entidades

La gestión de las múltiples entidades del sistema se abordó mediante un patrón modular y repetible que **desacopla** la lógica de acceso a datos, la presentación en pantalla y la validación de la información introducida por el usuario.

La **comunicación con el backend** se canaliza a través de un **cliente API centralizado**, que se encarga de realizar todas las peticiones HTTP a la API REST y gestionar de manera uniforme los errores y la autenticación. Para cada dominio, se crearon **hooks** personalizados (por ejemplo, `useUsers`, `useProducts`) que encapsulan todas las operaciones CRUD, gestionando internamente los **estados** de carga, error y datos. Esta abstracción permitió que los componentes de UI se centraran exclusivamente en la presentación, consumiendo los `hooks` sin preocuparse de los detalles de la comunicación.

Respecto a la gestión de **formularios**, se utilizó el sistema `useForm` de *Mantine*, el cual permite definir validaciones personalizadas, gestionar el estado de los campos y mostrar mensajes de error de forma clara.

Para la **gestión del estado** general se siguió un principio de simplicidad: la mayoría de las entidades manejan su información de forma local mediante sus **hooks**. Sin embargo, se hizo una excepción para el *dashboard* de pedidos (figura 5.5), una de las funcionalidades centrales y más distintivas de la aplicación. Dado que este módulo requiere visualizar y actualizar en tiempo real el estado de los pedidos pendientes (permitiendo añadir productos, modificar cantidades y cambiar estados), se optó por un **store** global basado en **Zustand**. Esta decisión permite que varios componentes reaccionen simultáneamente a los cambios, ofreciendo una experiencia fluida y en tiempo real sin recargas de página.

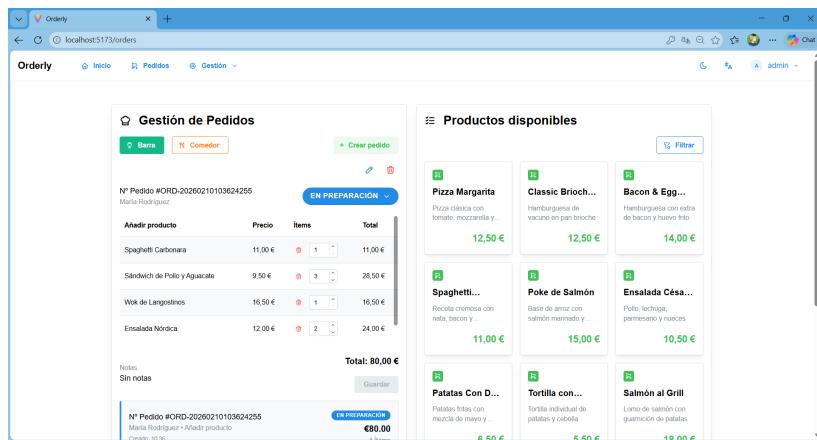


Figura 5.5: *Dashboard* de pedidos

Internacionalización

La aplicación es completamente multilingüe, permitiendo alternar entre español e inglés de forma instantánea. Para ello se ha utilizado **react-i18next**, organizando los textos en archivos de recursos por módulo (*namespaces*) y facilitando la traducción y ampliación futura a otros idiomas.

La aplicación también permite la búsqueda de **alimentos** de distintos idiomas y países. Como se ha indicado previamente, esto es gracias a que la API nutricional *Open Food Facts* permite la búsqueda por texto libre, es decir, no requiere que se introduzca un índice exacto o un nombre de producto en un idioma particular, puesto que contiene información de miles de productos etiquetados de cada país.

5.4. Consumo de la API de datos nutricionales

La elección de una API que proporcionase información de alimentos y datos nutricionales se llevó a cabo mediante una comparativa que se encuentra detallada en el *issue #10*. La API seleccionada fue **Nutritionix** [25], la cual también fue elegida en el proyecto *NutriMenu* [56].

Sin embargo, a 19 de noviembre de 2025, recibí una notificación oficial vía correo electrónico por parte de *Nutritionix*, en la cual se comunicaba que el plan de acceso gratuito dejaría de estar operativo [49], invalidando así su uso para el proyecto.

Se optó entonces por la segunda opción más completa de la comparativa: **Open Food Facts (OFF)** [11]. Esta decisión permitió la inclusión de métricas nutricionales (*Nutri-Score* y clasificación NOVA) por alimento e información sobre **alérgenos**, las cuales no estaban disponibles en la mayoría de planes gratuitos del resto de APIs.

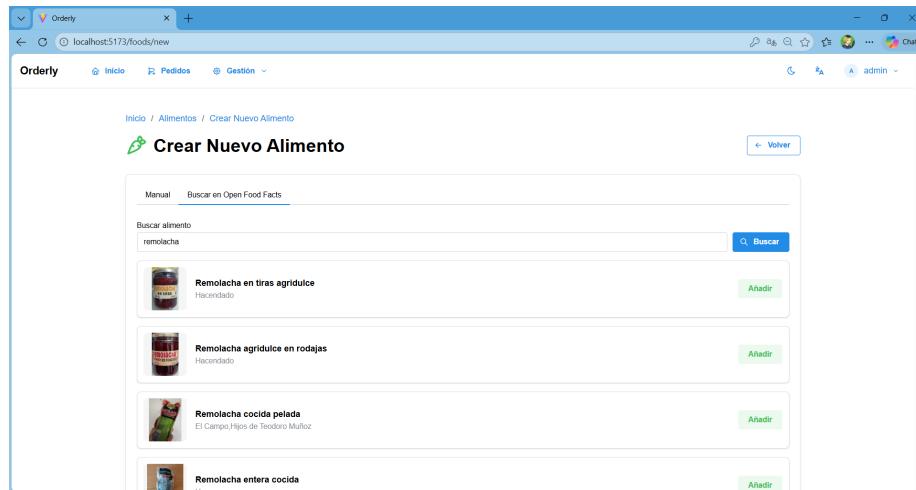


Figura 5.6: Búsqueda de un alimento mediante la API *Open Food Facts*

La comunicación con *OFF* se basa en una **primera fase** de búsqueda y selección de un alimento (figura 5.6), complementada por una **segunda fase** de petición de información detallada de dicho alimento y posterior adaptación al modelo de datos. Las peticiones a la API externa para ambas operaciones se realizan desde *Spring Boot*. De esta manera, el *frontend* queda **desacoplado** de la implementación de la API nutricional, favoreciendo el mantenimiento y el **intercambio** (o adición) de **fuentes de datos** nutricionales. También

5.5. EVALUACIÓN AUTOMATIZADA DE LA CALIDAD DEL CÓDIGO

se evitan restricciones de CORS (*Cross-Origin Resource Sharing* [9]) al realizar las peticiones desde el servidor en lugar del cliente, garantizando la **compatibilidad entre navegadores** y centralizando la gestión de las cabeceras de red.

De igual manera, para facilitar el proceso de adaptación de los datos importados, las equivalencias entre campos del modelo y datos de *OFF* se han centralizado en el archivo `OpenFoodFactsRegistry`, mientras que toda la lógica de conversión se sitúa en un *mapper* específico (`OpenFoodFactsMapper`).

Dado que *Open Food Facts* ofrece una cantidad muy elevada de información por alimento, fue necesario realizar un proceso de **selección** y adaptación de los campos más relevantes para el modelo de datos de la aplicación. Para cada nutriente, la API ofrece distintas representaciones (por ración o *serving*, por 100 gramos...), lo que introduce cierta **heterogeneidad** en términos de disponibilidad y formato. Finalmente, se optó por utilizar de forma sistemática los valores normalizados por **100 gramos**, ya que estos presentan una mayor disponibilidad y fiabilidad al tratarse de información obligatoria en el etiquetado nutricional, según el Reglamento (UE) nº 1169/2011 [47]. Esta elección permite simplificar el procesamiento de los datos, reducir la necesidad de conversiones y validaciones adicionales (por ejemplo, consultar las unidades), y resulta adecuada teniendo en cuenta que los valores nutricionales se **recalculan** posteriormente en función de la cantidad de alimento empleada en cada producto.

5.5. Evaluación automatizada de la calidad del código

Para garantizar la robustez, mantenibilidad y calidad técnica de la solución desarrollada, se ha implementado un **flujo** de integración continua (CI) que automatiza el análisis del código fuente de forma integral. Esta infraestructura permite **monitorizar** de manera constante el estado del proyecto, asegurando que el desarrollo de nuevas funcionalidades no comprometa la integridad ni la calidad del *software* existente.

La implementación de este sistema se ha articulado mediante el uso de *Github Actions* como **orquestador** de flujos de trabajo y *SonarCloud* como plataforma de **análisis estático**. La configuración permite un análisis modular que abarca tanto el *frontend* como el *backend*, centralizando los resultados en un **único panel** de control (figura 5.7).

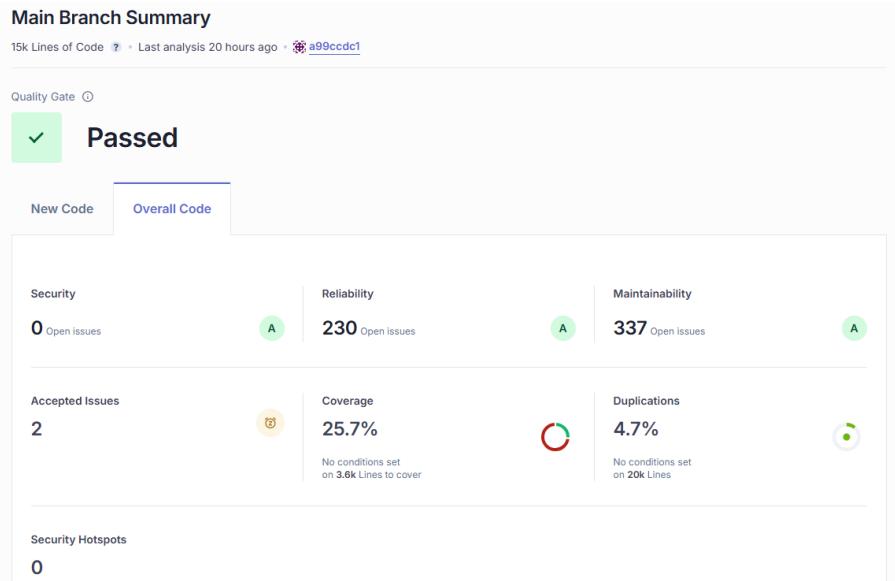


Figura 5.7: Métricas de calidad de código generadas por *SonarCloud*

Cada vez que se sincronizan **cambios** en el repositorio, se dispara automáticamente un proceso (*workflow*) que prepara el entorno y realiza las siguientes acciones:

- **En el backend:** se compila la aplicación y se ejecuta la batería de **pruebas** unitarias y de integración mediante *Maven*. Durante este proceso, se utiliza el componente *JaCoCo* para generar informes de **cobertura** de código, que son enviados a *SonarCloud* para su evaluación.
- **En el frontend:** aunque esta parte no dispone actualmente de pruebas automatizadas, se integra en el flujo de análisis para evaluar la calidad del código escrito, detectando malas prácticas o posibles errores de lógica en *JavaScript/React*.

La principal ganancia de este enfoque es la obtención de un “*Quality Gate*” o **umbral de calidad**. Con cada subida de código, el sistema evalúa automáticamente la presencia de errores, vulnerabilidades, duplicidades y “deuda técnica” (*code smells*). Esto permite una detección temprana de problemas que, de otra forma, podrían pasar desapercibidos hasta etapas más avanzadas del despliegue.

5.5. EVALUACIÓN AUTOMATIZADA DE LA CALIDAD DEL CÓDIGO³⁵

En la figura 5.7 se muestra el panel de métricas obtenido tras el análisis de la aplicación. En él se puede observar cómo el sistema califica distintos aspectos del *software*, como el grado de fiabilidad, mantenibilidad o diversas vulnerabilidades. En general, se obtuvo una calificación **A** para el conjunto de métricas evaluadas, lo que indica que el código cumple con los estándares de **buenas prácticas** y que carece de errores lógicos críticos o vulnerabilidades que comprometan su ejecución.

Respecto a la **cobertura** del código, los umbrales de calidad por defecto de *SonarCloud* exigen que sea superior al 80 % para superar el *Quality Gate*. Como la cobertura de los tests no era un objetivo principal del desarrollo del proyecto, se optó por **excluir** esta métrica de dicho umbral, mediante la definición de un *Quality Gate propio*. Esto se hizo posible adquiriendo el periodo de prueba del plan *Team* de *SonarCloud* [35].

6. Trabajos relacionados

En el ecosistema de *software* relacionado con la restauración se observan, de forma general, dos líneas predominantes:

1. **Sistemas de punto de venta (TPV²/POS³)** y gestión de locales, orientados a la operativa de sala, caja e integración con *hardware*.
2. **Herramientas y servicios especializados en análisis nutricional**, cálculo de información nutricional y etiquetado de menús.

Con el objetivo de situar este trabajo en el panorama descrito, se describirá a continuación una aplicación representativa de cada rama. Posteriormente se mostrará una tabla comparativa (ver tabla 6.1) entre las funcionalidades características de las aplicaciones presentadas y las de *Orderly*.

6.1. Floreant POS

Floreant POS [27] es un sistema *POS* de código abierto orientado a restaurantes, cafeterías y establecimientos de hostelería. Entre sus funcionalidades destacadas se encuentran la gestión de comandas y mesas (*table management*), impresión y enrutamiento a cocina (*kitchen printer / kitchen display*), soporte para pantallas táctiles, manejo de impuestos, gestión de inventario y empleados, funcionalidades de toma de pedidos para distintos modos de servicio (*dine-in, take-away, delivery*), y capacidades de reporting/estadísticas.

²Terminal Punto de Venta

³*Point of Sale*

Adicionalmente, *Floreant* está diseñado para funcionar en entornos locales (modo *offline*), admite integración con periféricos típicos de TPV (impresoras de tickets, cajón, etc.) y dispone de *plugins* o extensiones para funcionalidades añadidas como inventario más avanzado.

6.2. MenuCalc

MenuCalc [22] es una plataforma *SaaS (Software as a Service)* especializada en análisis nutricional para la restauración y el sector alimentario. Sus funcionalidades principales son: biblioteca/patrón extensivo de ingredientes (base de datos con cientos de miles de ingredientes), análisis nutricional de recetas y menús, etiquetado y generación de información para cumplimiento normativo (por ejemplo, etiquetado y requisitos de información al consumidor), marcado automático de alérgenos, categorización de recetas y funcionalidades de publicación (*SmartMenu* u opciones para mostrar información al comensal).

MenuCalc está orientado a ofrecer análisis rápidos y conformes a normativa (especialmente dirigido a mercados con requisitos de etiquetado) y es una solución comercial que se integra con flujos de trabajo de restauración en entornos profesionales.

6.3. NutriMenu

La aplicación *NutriMenu* [56] fue desarrollada por el alumno Álvaro Manjón Vara como Trabajo de Fin de Grado, con el objetivo de generar informes nutricionales para los menús ofrecidos en los centros de restauración de la Universidad de Burgos. Su funcionalidad principal se centra en el ciclo de vida de los menús: creación de alimentos (ingredientes) a partir de la API de *Nutritionix* [25], composición de platos, construcción de menús a partir de esos platos y, finalmente, la generación de informes nutricionales detallados para los consumidores.

Mientras que *NutriMenu* es una solución especializada en la transparencia nutricional para entornos multi-centro, *Orderly* evoluciona el concepto hacia una **plataforma de gestión operativa integral**. Incorpora flujos de negocio esenciales (ventas, reservas), un modelo de seguridad más sofisticado, información de alérgenos e internacionalización, posicionándose como un prototipo más cercano a un sistema de gestión moderno para el sector de la restauración.

6.4. COMPARATIVA SOBRE LAS FUNCIONALIDADES CUBIERTAS

6.4. Comparativa sobre las funcionalidades cubiertas

Tabla 6.1: Comparativa de características entre aplicaciones

Funcionalidad	NutriMenu	Floreant	MenuCalc	Orderly
Gestión de alimentos	Sí	Parcial	Sí	Sí
Importación API	Sí	No	Sí	Sí
Cálculo nutricional	Sí	No	Sí	Sí
Gestión de pedidos	No	Sí	No	Sí
Gestión de reservas	No	Sí	No	Sí
Integración TPV	No	Sí	No	No
Cumplimiento fiscal	No	Parcial	Sí	No
Roles y usuarios	Parcial	Sí	Sí	Sí
Despliegue	Docker	Local	SaaS	Docker

El **valor añadido** de *Orderly* radica en **combinar** los dos dominios mencionados al inicio: la **gestión operativa** (productos, pedidos, reservas, usuarios) y la incorporación automatizada de **información nutricional** y detección de alérgenos mediante consumo de APIs públicas, en una plataforma moderna, modular y fácilmente desplegable mediante contenedores.

Este enfoque híbrido facilita la replicabilidad y la extensión (por ejemplo, integraciones con bases de datos comerciales o con soluciones TPV) y permite atender tanto las necesidades operativas de un local como la creciente demanda de transparencia nutricional por parte de los consumidores. Para la importación automática de datos nutricionales pueden usarse fuentes públicas como *Open Food Facts*, la actual implementación, o alternativas comerciales (*Spoonacular*, etc.), según precisión y garantías que se requieran en el escenario de despliegue.

7. Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

El desarrollo de este Trabajo de Fin de Grado me ha permitido abordar de forma práctica el ciclo completo de desarrollo de una aplicación web, desde el análisis y diseño hasta la implementación, despliegue y documentación. Considero que los **objetivos** planteados al inicio del proyecto se han cumplido satisfactoriamente, ya que se ha obtenido una aplicación funcional y coherente para la gestión de locales de restauración, capaz de centralizar procesos clave como la gestión de productos, pedidos, reservas, mesas y usuarios. Además, la arquitectura diseñada sienta una **base sólida** para futuras ampliaciones y mejoras.

A nivel técnico, el proyecto ha supuesto una importante oportunidad de **aprendizaje**, permitiéndome afianzar conceptos generales de ingeniería del *software* y del ciclo de vida del desarrollo adquiridos a lo largo del grado. Destaca especialmente el desarrollo del *frontend* con *React*, tecnología que no había utilizado previamente y que me ha permitido adquirir competencias en el desarrollo de interfaces web modernas. En el *backend*, se ha dado una gran importancia al diseño del modelo de datos y de la lógica de negocio, priorizando la mantenibilidad y la escalabilidad de los componentes. Además, el desarrollo del proyecto ha requerido una planificación cuidadosa del trabajo, fomentando una correcta distribución y priorización de tareas a lo largo del tiempo.

En este contexto, la implementación de un sistema flexible de roles y **permisos** aporta un valor significativo a la aplicación, ya que permite adaptarse

a **escenarios reales** donde los puestos de trabajo y sus responsabilidades son cambiantes y específicos, ofreciendo al usuario la capacidad de definir y gestionar sus propios perfiles de acceso según sus necesidades.

Otro aspecto diferencial de la aplicación es la **integración de la API** de *Open Food Facts*, la cual ha permitido enriquecer los productos con información nutricional, alérgenos y métricas como *Nutri-Score* y NOVA. Este aspecto, unido a mi interés personal y experiencia previa en el sector de la restauración, ha hecho que el desarrollo del proyecto resultara especialmente motivador y satisfactorio, consolidando tanto conocimientos técnicos como habilidades de planificación y organización del trabajo.

7.2. Líneas de trabajo futuras

Como se ha mencionado anteriormente, *Orderly* nace con una arquitectura diseñada para la **escalabilidad**. Aunque la versión actual cumple con los objetivos de gestión básicos, el sistema se ha concebido como una **base** sobre la cual se pueden implementar múltiples módulos adicionales. A continuación, se detallarán las principales vías de expansión propuestas.

Evolución del modelo y gestión de productos

- **Diferenciación de tipos de producto:** clasificación detallada de ítems (entrantes, platos principales, bebidas, postres) para mejorar la organización de la carta.
- **Venta por peso y unidades de medida:** soporte para precios por kilogramo o litro, permitiendo una gestión más precisa de productos a granel.
- **Escandallo de costes:** implementación de una herramienta para calcular el coste de producción de cada plato a partir de sus ingredientes y determinar márgenes de beneficio óptimos.

Módulo visual de sala y reservas

- **Mapa dinámico del comedor:** sustitución de los listados de mesas por una interfaz gráfica interactiva que represente la disposición física del local.

- **Gestión avanzada de reservas:**

- Asignación de reservas directamente sobre el mapa visual.
- Historial de ocupación por mesa y tramos horarios.
- Sistema de avisos y recordatorios automáticos para clientes.

Experiencia del cliente y salud

- **Carta digital interactiva:** expansión de la interfaz para uso de clientes, permitiendo la consulta de platos y su información nutricional en tiempo real.
- **Generador de menús dinámicos:** creación de menús configurables (ej. menú del día, fin de semana) con reportes nutricionales agregados.
- **Recomendaciones saludables:** implementación de un motor de sugerencias basado en los perfiles nutricionales obtenidos de la API externa.

Optimización operativa y analítica

- **Panel de visualización para cocina:** interfaz específica para el personal de cocina que permita gestionar el estado de los pedidos (en preparación, listo para servir) mediante actualizaciones en tiempo real.
- **Estadísticas:** creación de un *dashboard* analítico para monitorizar los productos más vendidos, horas de mayor afluencia y rendimiento económico.
- **Gestión de inventario automática:** descuento automático de existencias tras cada pedido e integración de alertas de *stock* bajo.

Mejoras técnicas y de accesibilidad

- **Seguridad y recuperación:** implementación de un flujo de recuperación de contraseñas mediante envío de credenciales temporales al correo electrónico.
- **Integración con servicios de terceros:** exploración de conexiones con plataformas de *delivery* e integración de pasarelas de pago *online*.
- **Calidad de software:** ampliación de la cobertura de pruebas unitarias e integración de pruebas de extremo a extremo (E2E) para el *frontend*.

Eficiencia técnica y rendimiento del sistema

- **Estrategias de almacenamiento en caché (*Caching*):**
 - **Caché de servicios externos:** implementar una capa de caché para almacenar los resultados de la API nutricional, reduciendo el tiempo de respuesta y el consumo de cuota de la API externa (en caso de integrar una nueva API que tenga límite de peticiones).
 - **Caché en el cliente:** optimizar la comunicación entre *React* y *Spring Boot*, evitando peticiones redundantes al *backend* cuando los datos no han variado.
- **Sistema de refresco automático de información nutricional:** establecer un mecanismo basado en un tiempo de vida (*Time-To-Live* o *TTL*) para los datos de los alimentos. Una vez superado dicho umbral, el sistema realizaría una petición automática de actualización a la API para garantizar que la información mostrada sea siempre vigente y precisa.

En definitiva, *Orderly* se posiciona como una solución versátil que puede **evolucionar** desde una herramienta interna de gestión hasta convertirse en una plataforma integral que conecte a trabajadores y clientes bajo un **ecosistema digital común**.

Bibliografía

- [1] Baeldung. Introduction to Project Lombok. <https://www.baeldung.com/intro-to-project-lombok>, 2025. [Internet; Accedido 27-12-2025].
- [2] Esteban Canle. ¿Qué es Spring Boot y para qué sirve? <https://www.tokioschool.com/noticias/spring-boot/>, 2025. [Internet; Accedido 26-12-2025].
- [3] Conventional Commits. Conventional Commits. <https://www.conventionalcommits.org/en/v1.0.0/>, 2025. [Internet; Accedido 27-12-2025].
- [4] Documentación de GitHub. Acerca de GitHub y Git. <https://docs.github.com/es/get-started/start-your-journey/about-github-and-git>, 2025. [Internet; Accedido 27-12-2025].
- [5] Documentación de GitHub. Acerca de Projects. <https://docs.github.com/es/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects>, 2025. [Internet; Accedido 27-12-2025].
- [6] Documentación de GitHub. Entender las GitHub Actions. <https://docs.github.com/es/actions/get-started/understanding-github-actions>, 2026. [Internet; Accedido 09-01-2026].
- [7] Docker. Docker Compose. <https://docs.docker.com/compose/>, 2025. [Internet; Accedido 27-12-2025].
- [8] Docker. What is Docker? <https://docs.docker.com/get-started/docker-overview/>, 2025. [Internet; Accedido 27-12-2025].

- [9] MDN Web Docs. Intercambio de recursos de origen cruzado (CORS) - HTTP. <https://developer.mozilla.org/es/docs/Web/HTTP/Guides/CORS>, 2026. [Internet; Accedido 06-02-2026].
- [10] Claire Drumond. ¿Qué es scrum? Desglose del marco de trabajo ágil. <https://www.atlassian.com/es/agile/scrum>, 2025. [Internet; Accedido 26-12-2025].
- [11] Open Food Facts. Open Food Facts. <https://es.openfoodfacts.org/>, 2025. [Internet; Accedido 29-12-2025].
- [12] GeeksforGeeks. Domain-Driven Design (DDD). <https://www.geeksforgeeks.org/system-design/domain-driven-design-ddd/>, 2025. [Internet; Accedido 26-12-2025].
- [13] GeeksforGeeks. React Introduction. <https://www.geeksforgeeks.org/reactjs/reactjs-introduction/>, 2025. [Internet; Accedido 27-12-2025].
- [14] GeeksforGeeks. React Router. <https://www.geeksforgeeks.org/reactjs/reactjs-router/>, 2025. [Internet; Accedido 27-12-2025].
- [15] i18next Documentation. Introduction. <https://www.i18next.com/>, 2025. [Internet; Accedido 27-12-2025].
- [16] React i18next Documentation. Introduction. <https://react.i18next.com/>, 2025. [Internet; Accedido 27-12-2025].
- [17] IntelliJ IDEA. IntelliJ IDEA overview. <https://www.jetbrains.com/help/idea/discover-intellij-idea.html>, 2025. [Internet; Accedido 30-12-2025].
- [18] Akshata Kanaje. Setting Up H2 for Testing in Spring Boot application. <https://medium.com/@akshatakanaje08/setting-up-h2-for-testing-in-spring-boot-application-7f016220a475>, 2024. [Internet; Accedido 30-12-2025].
- [19] LaTeX. LaTeX - A document preparation system. <https://www.latex-project.org/>, 2025. [Internet; Accedido 27-12-2025].
- [20] TeX Live. The TeX Live Guide - 2025. <https://tug.org/texlive/doc/texlive-en/texlive-en.html>, 2025. [Internet; Accedido 09-01-2026].

- [21] Mantine. Getting started. <https://mantine.dev/getting-started/>, 2025. [Internet; Accedido 27-12-2025].
- [22] MenuCalc. Nutrition Analysis Software | MenuCalc. <https://menucalc.com/>, 2026. [Internet; Accedido 14-01-2026].
- [23] MiKTeX. Home. <https://miktex.org/>, 2025. [Internet; Accedido 27-12-2025].
- [24] Carlos Augusto Monteiro, Geoffrey Cannon, Mark Lawrence, Maria Laura da Costa Louzada, and Priscila Pereira Machado. Ultra-processed foods, diet quality, and health using the NOVA classification system. Technical report, Food and Agriculture Organization of the United Nations (FAO), Rome, 2019.
- [25] Nutritionix. Nutritionix. <https://www.nutritionix.com/>, 2025. [Internet; Accedido 29-12-2025].
- [26] Overleaf. About us. <https://www.overleaf.com/about>, 2025. [Internet; Accedido 27-12-2025].
- [27] Floreant POS. Floreant POS - Open Source Free Point of Sale. <https://floreant.org/>, 2026. [Internet; Accedido 14-01-2026].
- [28] Postman. Postman API Platform - Build, Test & Manage. <https://www.postman.com/product/>, 2025. [Internet; Accedido 27-12-2025].
- [29] Ed Putans. Single Source of truth and applying it software development. <https://edunceputans.medium.com/single-source-of-truth-and-problems-with-implication-in-an-organisation-588883492133>, 2018. [Internet; Accedido 30-12-2025].
- [30] Dan Radigan. Kanban. <https://www.atlassian.com/es/agile/kanban>, 2025. [Internet; Accedido 26-12-2025].
- [31] Recharts. Recharts. <https://recharts.github.io/>, 2026. [Internet; Accedido 11-01-2026].
- [32] Santé publique France. Nutri-Score: Questions & Answers. Technical report, Santé publique France, March 2025. English version dated 17th of March 2025.
- [33] Santé publique France. Nutri-Score: Questions & Answers. Section: Calculation methods for the updated algorithm (Nutri-Score 2023). Technical report, Santé publique France, March 2025. English version dated 17th of March 2025. pp. 26–32.

- [34] SonarSource. Online Code Review as a Service Tool, SonarQube Cloud (Formerly SonarCloud). <https://www.sonarsource.com/products/sonarqube/cloud/>, 2026. [Internet; Accedido 06-02-2026].
- [35] SonarSource. Subscription plans | SonarQube Cloud | Sonar Documentation. <https://docs.sonarsource.com/sonarqube-cloud/administering-sonarcloud/managing-subscription/subscription-plans>, 2026. [Internet; Accedido 10-02-2026].
- [36] Spring. Getting Started | Accessing data with MySQL. <https://spring.io/guides/gs/accessing-data-mysql>, 2025. [Internet; Accedido 27-12-2025].
- [37] Spring. Spring Data JPA. <https://spring.io/projects/spring-data-jpa>, 2025. [Internet; Accedido 26-12-2025].
- [38] Spring. Spring Framework. <https://spring.io/projects/spring-framework>, 2025. [Internet; Accedido 26-12-2025].
- [39] Spring. Spring Security. <https://spring.io/projects/spring-security>, 2025. [Internet; Accedido 26-12-2025].
- [40] Spring Framework Team. Spring Framework Reference Documentation. <https://docs.spring.io/spring-framework/docs/4.3.11.RELEASE/spring-framework-reference/htmlsingle/#integration-testing>, 2025. [Internet; Accedido 30-12-2025].
- [41] TeXstudio. TeXstudio - A LaTeX editor. <https://www.texstudio.org/>, 2025. [Internet; Accedido 27-12-2025].
- [42] Yan Min Thwin. Understanding GitHub Flow and Git Flow. <https://medium.com/@yanminthwin/understanding-github-flow-and-git-flow-957bc6e12220>, 2023. [Internet; Accedido 27-12-2025].
- [43] Thymeleaf. Tutorial: Using Thymeleaf. <https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html>, 2024. [Internet; Accedido 09-01-2026].
- [44] TutorialsPoint. JUnit - Overview. https://www.tutorialspoint.com/junit/junit_overview.htm, 2025. [Internet; Accedido 30-12-2025].

- [45] TutorialsPoint. Mockito - Overview. https://www.tutorialspoint.com/mockito/mockito_overview.htm, 2025. [Internet; Accedido 30-12-2025].
- [46] Alejandro Ugarte. Validation in Spring Boot. <https://www.baeldung.com/spring-boot-bean-validation>, 2019. [Internet; Accedido 27-12-2025].
- [47] Unión Europea. Reglamento (UE) n.º 1169/2011 del Parlamento Europeo y del Consejo, de 25 de octubre de 2011, sobre la información alimentaria facilitada al consumidor, 2011. Anexo II: Sustancias o productos que causan alergias o intolerancias.
- [48] Alejandro Urrestarazu. Modelos Anémicos vs. Modelos Enriquecidos. <https://memobackend.com.ar/2024-06-11-modelos-anemicos-enriquecidos/>, 2024. [Internet; Accedido 26-12-2025].
- [49] u/WatermelonWOseeds. Getting ERROR 401 Unauthorized when using Nutritionix Track API. Reddit (r/learnpython) - https://www.reddit.com/r/learnpython/comments/1oj0pys/getting_error_401_unauthorized_when_using/, 2025. [Internet; Accedido 28-12-2025].
- [50] Vite. Introducción. <https://es.vite.dev/guide/>, 2025. [Internet; Accedido 27-12-2025].
- [51] VSCode. Documentation for Visual Studio Code. <https://code.visualstudio.com/docs>, 2025. [Internet; Accedido 27-12-2025].
- [52] Wikipedia. MySQL - Wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/MySQL>, 2025. [Internet; Accedido 27-12-2025].
- [53] Wikipedia. Single-page application - Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Single-page_application, 2026. [Internet; Accedido 06-02-2026].
- [54] Zustand. Introduction. <https://zustand.docs.pmnd.rs/getting-started/introduction>, 2025. [Internet; Accedido 27-12-2025].
- [55] Cecilio Álvarez Caules. Java Mapping con MapStruct y anotaciones. <https://www.arquitecturajava.com/java-mapping-con-mapstruct-y-anotaciones/>, 2025. [Internet; Accedido 27-12-2025].

- [56] Álvaro Manjón Vara. NutriMenu: Plataforma unificada para la gestión nutricional en centros de restauración. Trabajo Fin de Grado, Universidad de Burgos, febrero 2024. Grado en Ingeniería Informática.