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

## Documentación Técnica

Presentado por Amanda Pérez Olmos
en Universidad de Burgos — 11 de febrero de 2026
Tutor: Raúl Marticorena Sánchez

# Índice general

**Índice general** **i**

**Índice de figuras** **iii**

**Índice de tablas** **vi**

**Apéndice A Plan de Proyecto Software** **1**
A.1. Introducción . . . . . 1
A.2. Planificación temporal . . . . . 1
A.3. Estudio de viabilidad . . . . . 25

**Apéndice B Especificación de Requisitos** **31**
B.1. Introducción . . . . . 31
B.2. Objetivos generales . . . . . 31
B.3. Catálogo de requisitos . . . . . 32
B.4. Especificación de requisitos . . . . . 37

**Apéndice C Especificación de diseño** **59**
C.1. Introducción . . . . . 59
C.2. Diseño de datos . . . . . 59
C.3. Diseño arquitectónico . . . . . 66
C.4. Diseño procedimental . . . . . 73

**Apéndice D Documentación técnica de programación** **89**
D.1. Introducción . . . . . 89
D.2. Estructura de directorios . . . . . 89
D.3. Manual del programador . . . . . 94

D.4. Compilación, instalación y ejecución del proyecto . . . . . 99
D.5. Pruebas del sistema . . . . . 104

**Apéndice E Documentación de usuario 107**
E.1. Introducción . . . . . 107
E.2. Requisitos de usuarios . . . . . 107
E.3. Instalación . . . . . 108
E.4. Manual del usuario . . . . . 108

**Apéndice F Anexo de sostenibilización curricular 151**
F.1. Introducción . . . . . 151
F.2. ODS 3: Salud y bienestar . . . . . 151
F.3. ODS 8: Trabajo decente y crecimiento económico . . . . . 152
F.4. ODS 9: Industria, innovación e infraestructura . . . . . 152
F.5. ODS 12: Producción y consumo responsables . . . . . 152
F.6. ODS 13: Acción por el clima . . . . . 153
F.7. Conclusiones . . . . . 153

**Bibliografía 155**

# Índice de figuras

A.1. Gráfico *burndown* del Sprint 0 . . . . . 3
A.2. Gráfico *burndown* del Sprint 1 . . . . . 5
A.3. Gráfico *burndown* del Sprint 2 . . . . . 7
A.4. Gráfico *burndown* del Sprint 3 . . . . . 9
A.5. Gráfico *burndown* del Sprint 4 . . . . . 11
A.6. Gráfico *burndown* del Sprint 5 . . . . . 13
A.7. Gráfico *burndown* del Sprint 6 . . . . . 15
A.8. Gráfico *burndown* del Sprint 7 . . . . . 17
A.9. Gráfico *burndown* del Sprint 8 . . . . . 19
A.10. Gráfico *burndown* del Sprint 9 . . . . . 20
A.11. Gráfico *burndown* del Sprint 10 . . . . . 22
A.12. Gráfico *burndown* del Sprint 11 . . . . . 23
A.13. Gráfico *burndown* del Sprint 12 . . . . . 24
A.14. Gráfico *burnup* del proyecto . . . . . 24

B.1. Diagrama de Casos de Uso . . . . . 40

C.1. Esquema de la Base de Datos . . . . . 61
C.2. Diagrama relacional para módulo de usuarios y roles . . . . . 62
C.3. Diagrama relacional para módulo de alimentos y productos . . . 63
C.4. Diagrama relacional para módulo de reservas de mesas . . . . . 64
C.5. Diagrama relacional para módulo de pedidos . . . . . 65
C.6. Arquitectura general de *Orderly* . . . . . 67
C.7. Flujo de autenticación en *Spring Security* [14] . . . . . 70
C.8. Clase `OpenFoodFactsRegistry` con equivalencias entre campos de la API *Open Food Facts* y el modelo de nuestra aplicación . . 71
C.9. Diagrama de despliegue en entorno de desarrollo . . . . . 72
C.10. Diagrama de despliegue en entorno de producción . . . . . 73

C.11. Implementación del patrón *Factory* para la gestión polimórfica de pedidos y sus DTOs . . . 74
C.12. Ejemplo de métodos de dominio en entidades `Product`, `Food` y `Order` . . . 75
C.13. Ejemplos de métodos para consistencia bidireccional en entidades `Ingredient`, `Order` y `OrderItem` . . . 76
C.14. Diagrama de secuencia del inicio de sesión general . . . 77
C.15. Diagrama de secuencia del inicio de sesión en *Spring Boot* . . . 79
C.16. Diagrama de secuencia de la creación de una entidad . . . 81
C.17. Diagrama de secuencia de la creación de un pedido en *Spring Boot* 83
C.18. Diagrama de secuencia para la operación de añadir ingredientes a productos . . . 85
C.19. Diagrama de secuencia para la operación de añadir ítems a pedidos 86
C.20. Diagrama de secuencia para la búsqueda y creación de un alimento mediante *Open Food Facts* . . . 88

D.1. Módulos funcionales del *frontend* . . . 91
D.2. Estructura compartida entre los módulos funcionales de *React* . 92
D.3. Módulos funcionales del *backend* . . . 93
D.4. Estructura compartida entre los módulos funcionales de *Spring Boot* . . . 94
D.5. Contenedores *Docker* con los servicios de *Orderly* . . . 104

E.1. Página de bienvenida . . . 109
E.2. Página de registro de usuario . . . 109
E.3. Página de inicio de sesión . . . 110
E.4. Perfil de usuario . . . 111
E.5. Página de edición de perfil de usuario . . . 112
E.6. Barra de navegación para escritorio . . . 112
E.7. Sección izquierda de la barra de navegación . . . 113
E.8. Sección derecha de la barra de navegación . . . 114
E.9. Menú lateral de navegación para móviles . . . 114
E.10. *Breadcrumbs* de navegación . . . 115
E.11. Página de inicio tras registro, sin módulos de gestión . . . 115
E.12. Página de inicio con todos los módulos de gestión . . . 116
E.13. *Dashboard* de pedidos . . . 117
E.14. Modal de creación de un pedido de comedor . . . 117
E.15. Tabla con detalles del pedido seleccionado . . . 118
E.16. Lista de pedidos pendientes . . . 119
E.17. Filtro para excluir alérgenos en listado de productos . . . 119
E.18. Historial de pedidos . . . 120

E.19. Vista de detalles de un pedido desde historial . . . . . 121
E.20. Página de listado de productos . . . . . 122
E.21. Página de creación de productos . . . . . 122
E.22. Sección para la gestión de ingredientes de un producto . . . . . 123
E.23. Página de edición de productos . . . . . 124
E.24. Modal de eliminación de productos . . . . . 124
E.25. Página de detalles de un producto . . . . . 125
E.26. Gráfico nutricional de un producto . . . . . 125
E.27. Tabla nutricional de un producto con ingestas diarias de referencia 126
E.28. Lista detallada de ingredientes de un producto . . . . . 127
E.29. Página de listado de alimentos . . . . . 128
E.30. Modal de detalles de un alimento . . . . . 128
E.31. Página de creación de alimentos . . . . . 129
E.32. Introducción de valores nutricionales para un alimento . . . . . 130
E.33. Selección de alérgenos de un alimento . . . . . 130
E.34. Búsqueda de alimentos mediante API externa . . . . . 131
E.35. Página de edición de alimentos . . . . . 132
E.36. Modal de eliminación de alimentos . . . . . 133
E.37. Página de listado de reservas . . . . . 134
E.38. Página de creación de reservas . . . . . 134
E.39. Página de edición de reservas . . . . . 135
E.40. Modal de eliminación de reservas . . . . . 136
E.41. Página de listado de mesas . . . . . 137
E.42. Página de creación de mesas . . . . . 138
E.43. Página de edición de mesas . . . . . 138
E.44. Modal de eliminación de mesas . . . . . 139
E.45. Página de listado de usuarios . . . . . 140
E.46. Página de creación de usuarios . . . . . 141
E.47. Sección de asignación de roles para un usuario . . . . . 141
E.48. Página de edición de usuarios . . . . . 142
E.49. Modal de eliminación de usuarios . . . . . 143
E.50. Página de listado de roles . . . . . 144
E.51. Página de creación de roles . . . . . 144
E.52. Sección de asignación de permisos a un rol . . . . . 145
E.53. Página de edición de roles . . . . . 146
E.54. Modal de eliminación de roles . . . . . 146
E.55. Módulos de navegación para usuarios con rol `ROLE_CHEF` . . . . 147
E.56. Módulos de navegación para usuarios con rol `ROLE_WAITER_BAR` 148
E.57. Módulos de navegación para usuarios con rol `ROLE_WAITER_DINING` 149

# Índice de tablas

A.1. Equivalencia de *story points* y tiempo de desarrollo estimado . . 2
A.2. Resumen de costes económicos del proyecto . . . . . 28
A.3. Resumen de licencias de las herramientas y tecnologías utilizadas 29

B.1. Matriz de acciones según actores y módulos . . . . . 38
B.2. CU-1 Registrarse (*register*) . . . . . 41
B.3. CU-2 Iniciar sesión (*login*) . . . . . 41
B.4. CU-3 Cerrar sesión (*logout*) . . . . . 42
B.5. CU-4 Editar perfil propio . . . . . 43
B.6. CU-5 Crear entidad . . . . . 44
B.7. CU-5.1 Crear pedido . . . . . 45
B.8. CU-6 Editar entidad . . . . . 46
B.9. CU-7 Cambiar estado de entidad . . . . . 47
B.10. CU-8 Eliminar entidad . . . . . 48
B.11. CU-9 Listar entidades . . . . . 49
B.12. CU-10 Gestionar ingredientes de un producto . . . . . 50
B.13. CU-11 Gestionar líneas de pedido . . . . . 51
B.14. CU-12 Gestionar roles de un usuario . . . . . 53
B.15. CU-13 Buscar y crear alimento mediante *Open Food Facts* . . . 53
B.16. CU-14 Visualizar información nutricional y alérgenos . . . . . 55
B.17. CU-15 Filtrar productos excluyendo alérgenos . . . . . 56

*Apéndice* $A$

# Plan de Proyecto Software

## A.1. Introducción

Este apéndice detalla la estrategia de gestión que ha guiado el ciclo de vida del desarrollo. Para ello, se describe en primer lugar la organización temporal de las tareas y el seguimiento del progreso mediante metodologías ágiles. Asimismo, se incluye un análisis integral de la viabilidad del proyecto, evaluando tanto la inversión económica necesaria para su ejecución como el marco normativo y legal que regula el despliegue y uso de la aplicación.

## A.2. Planificación temporal

La ejecución del proyecto se ha articulado bajo un marco de trabajo basado en la metodología *Scrum*, empleando ciclos de desarrollo iterativos o *sprints*, una pila de producto organizada y un tablero *Kanban* para la gestión visual del flujo de trabajo. En este apartado se detalla la evolución del desarrollo a través de cada uno de los *sprints* realizados, especificando sus objetivos y analizando su progreso mediante gráficos *burn-down*. Los gráficos han sido generados mediante *Microsoft Excel*, imitando el formato de los generados por la herramienta *ZenHub*.

Para la gestión de las tareas (*issues*), se ha asignado a cada una un nivel de prioridad (Alta, Media o Baja) y una estimación de esfuerzo mediante *story points*. Aunque el *story point* es, por definición, una unidad de medida abstracta, para este proyecto la estimación se ha fundamentado principalmente en la carga temporal prevista. No obstante, dicho valor no es puramente cronológico, sino que se ha ponderado considerando también la

complejidad técnica del requisito y su relevancia dentro de la arquitectura global del sistema.

Siguiendo las prácticas habituales de las metodologías ágiles, los valores asignados corresponden a la secuencia de Fibonacci. En la tabla A.1 se detalla la equivalencia establecida entre los *story points* y la estimación temporal de referencia utilizada para la planificación de las tareas.

Tabla A.1: Equivalencia de *story points* y tiempo de desarrollo estimado

| ***Story Points* (Fibonacci)** | **Estimación temporal** |
|---|---|
| 1 | 30 mins |
| 2 | 1,5 horas |
| 3 | 3 horas |
| 5 | 6 horas |
| 8 | 12 horas |
| 13 | 1 día |
| 21 | 3 a 5 días |

## *Sprint* 0

Duración: 1 semana (18/09/2025 - 25/09/2025)

Este primer *sprint* abarca las tareas de investigación y definición inicial, enfocadas en determinar el alcance mínimo del proyecto, junto con las tecnologías y recursos a utilizar.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 0**.

### Objetivos

- **Creación del repositorio:** se configuró el repositorio de *GitHub* para el control de versiones. Utilizando *GitHub Projects*, se asoció un proyecto a dicho repositorio para la gestión y mantenimiento de las tareas de cada *sprint*.

- **Creación de los *epics*:** se definieron *issues* principales para estructurar el proyecto en áreas funcionales (desarrollo, *testing*, documentación) y agrupar tareas relacionadas, agilizando el acceso y el seguimiento del progreso.

- **Definición del *stack* tecnológico:** se eligió ***Spring Boot*** (con *Spring Security* y *Spring Data JPA*) para el *backend* y la lógica de negocio, junto con ***MySQL*** como base de datos. Para el *frontend*, se optó por ***Thymeleaf***, que facilita la generación de vistas HTML dinámicas desde el servidor. Finalmente, se seleccionó ***Docker*** para los contenedores de la aplicación y la base de datos, asegurando un despliegue portable y reproducible.
- **Estudio y evaluación de APIs nutricionales:** se realizó una comparativa entre varias APIs de alimentos. La API elegida fue *Nutritionix* [12]. Los criterios de elección y el resto de candidatas pueden verse en el *issue* #10.
- **Revisión de trabajos previos y TFGs relacionados:** se probó la aplicación *NutriMenu* [21], producto de un TFG anterior, revisando si alguno de sus módulos podría aprovecharse y documentando carencias.
- **Análisis de integración con plataformas de *delivery*:** se investigó acerca de los requisitos necesarios para obtener claves de acceso a las APIs de distintas plataformas de *delivery*. Se terminó por concluir que esta integración se pospondría para etapas más avanzadas del desarrollo, en caso de que se dispusiera del tiempo necesario.

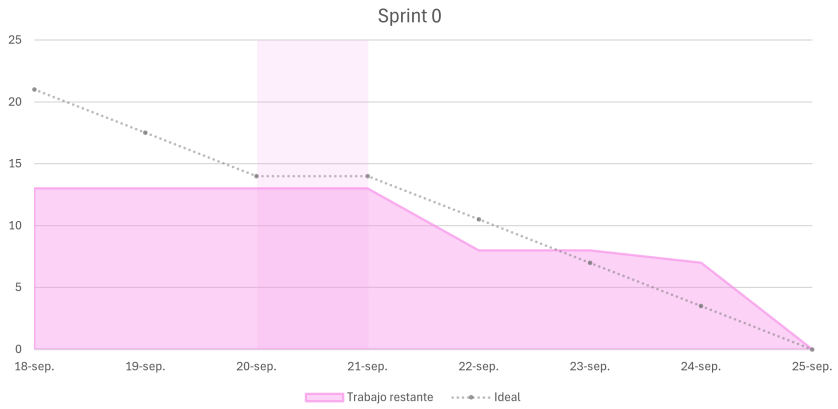


Figura A.1: Gráfico *burndown* del Sprint 0

### Revisión

Como la API de *Nutritionix* no proporciona directamente datos de alérgenos, se valoró la idea de **"*mapear*" los alérgenos** en base a los ingredientes,

mediante un modelo de reglas y una lista predefinida de alérgenos. Se planteó la posibilidad de que existiese algún recurso *online* que ofreciese esta información (para no tener que generarlo manualmente), por lo que se propuso investigar acerca de su disponibilidad.

Otro aspecto que se trató en relación con la API fue la gestión eficiente de las llamadas. Debido a que existe un límite mensual, se propuso la idea de **almacenar temporalmente** ("*cachear*") la **información de los productos** en nuestra base de datos local. De esta forma, si un usuario consultara un producto que ya hubiese sido buscado previamente, el sistema recuperaría los datos de nuestro almacén local en lugar de hacer una nueva solicitud a la API. Para asegurar que la información esté siempre actualizada, los datos almacenados podrían tener un **periodo de validez**. Una vez superado este tiempo, la aplicación forzaría una nueva llamada a la API para refrescar la información.

Se acordó, para el siguiente *sprint*, comenzar el desarrollo de la aplicación, empezando por la infraestructura de usuarios y roles.

## *Sprint* 1

Duración: 1 semana (25/09/2025 - 02/10/2025)

Este *sprint* se centró en crear la infraestructura inicial de la aplicación, añadiendo las dependencias y los archivos de configuración necesarios para levantar la aplicación *Spring Boot*. Una vez creada, se empezó a implementar la gestión de usuarios y roles, definiendo las entidades y métodos pertinentes para, como mínimo, realizar operaciones CRUD sobre ellos.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 1**.

### Objetivos

- **Creación del proyecto *Spring Boot*:** se configuró un proyecto base con dependencias como *Spring Security*, *Spring Data JPA* y *MySQL Driver*, entre otras.
- **Generación del `docker-compose.yml`:** se creó dicho archivo para orquestar los contenedores de *MySQL* y *Spring Boot*. La imagen de la aplicación *Spring Boot* se creó a partir de su propio `Dockerfile`. Se añadió un archivo de inicialización de la base de datos y se comprobó que ambos servicios podían comunicarse entre sí.

- **Creación de las entidades `User` y `Role`:** se definieron las entidades JPA y las tablas SQL del archivo de inicialización. Se estableció inicialmente que los usuarios solo puedan tener un rol. En iteraciones futuras, se agregará la funcionalidad de tener múltiples roles.
- **Implementación de operaciones CRUD sobre usuarios y roles:** se agregaron los repositorios, servicios y controladores necesarios para llevar a cabo estas funcionalidades.
- **Configuración de *Spring Security*:** se adaptaron las entidades `User` para poder autenticarse en la aplicación mediante *Spring Security* y controlar el acceso a cada *endpoint* según los roles.
- **Creación de formulario de *login* básico:** un inicio de sesión mediante usuario y contraseña, en el que se muestran mensajes descriptivos en caso de error.
- **Generación de pruebas de integración:** se generaron pruebas básicas de integración para el repositorio y servicio de usuarios. Sirvieron para familiarizarse con los tipos de pruebas y anotaciones de *Spring Boot*.

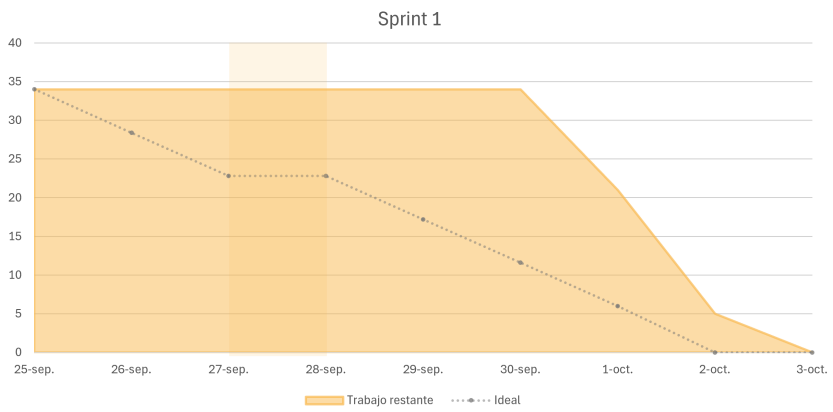


Figura A.2: Gráfico *burndown* del Sprint 1

### Revisión

Se acordó, para el siguiente *sprint*, seguir implementando los aspectos restantes relativos a la gestión de usuarios en la aplicación, para poder dar prácticamente por acabada esa funcionalidad. Se comentó también el

tema de la **internacionalización** de la aplicación, y cómo sería beneficioso intentar implementarlo desde el primer momento.

## *Sprint* 2

Duración: 1 semana (02/10/2025 - 09/10/2025)

Este *sprint* se enfocó en expandir la funcionalidad de gestión de usuarios mediante la creación de formularios, como el de registro, edición o mantenimiento de usuarios (por parte de un administrador). También se empezaron a escribir los apéndices A y B de la documentación.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 2**.

### Objetivos

- **Creación de formulario de registro de usuarios:** una pantalla que permite a nuevos usuarios crear su perfil en la aplicación (con `ROLE_USER` por defecto), en la que se muestran mensajes descriptivos en caso de error.
- **Creación de formulario de edición de usuarios:** una pantalla que permite editar el perfil de un usuario, disponible para el propio usuario y para administradores que quieran editar dicho usuario. Los campos mostrados dependen de la identidad y rol del *usuario editor*.
- **Implementación de módulo de gestión de usuarios para *admins*:** se creó una página que permite a los administradores de la aplicación ver un listado de todos los usuarios de la misma y poder realizar operaciones de gestión, como la creación, modificación y eliminación de dichos usuarios.
- **Comienzo de la documentación de *sprints* anteriores:** se redactó acerca de los *sprints* completados hasta entonces (véanse *Sprint* 0 y *Sprint* 1), para mantener un seguimiento claro de las tareas completadas y las decisiones tomadas durante el ciclo de vida del proyecto.
- **Primera definición de requisitos funcionales:** aunque estos requisitos ya se tenían presentes desde el comienzo del proyecto, se redactó una primera versión de estos en la documentación.

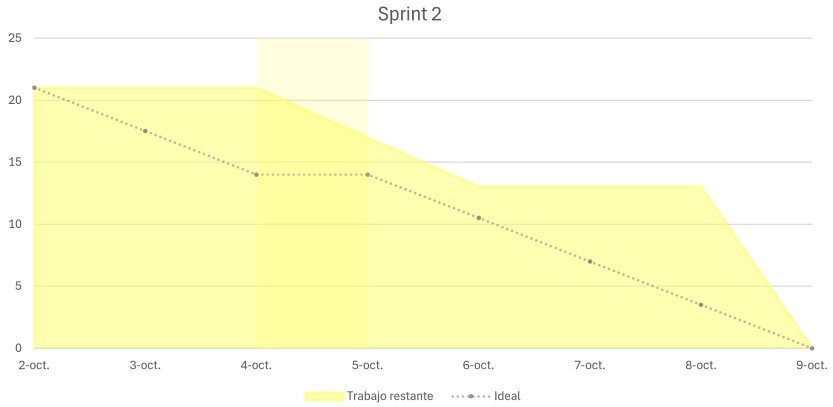


Figura A.3: Gráfico *burndown* del Sprint 2

### Revisión

Al haber mostrado el funcionamiento de los *endpoints* de la aplicación al tutor mediante *Postman*, se propuso la posible generación de una **colección de solicitudes** completa, para adjuntar como archivo complementario en cada *release*.

Se acordó, para el siguiente *sprint*, dejar a un lado la funcionalidad de usuarios y proseguir con la gestión de ingredientes y productos ofertados.

## *Sprint* 3

Duración: 2 semanas (09/10/2025 - 23/10/2025)

En este *sprint* se comenzó a desarrollar la gestión de los productos ofertados, tomando como referencia las clases existentes en *NutriMenu* [21]. También se finalizaron aspectos relacionados con la gestión de usuarios, que se consideraban necesarios de implementar lo antes posible. Inicialmente se planificó la duración para 1 semana, pero acabó alargándose debido a cuestiones personales.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 3**.

### Objetivos

- **Creación de las entidades `Food`, `Product` y `Recipe`:** se definieron las entidades JPA, basándose en los campos existentes en las clases de

*NutriMenu* [21] y agregando nuevos. Se simplificaron considerablemente el código y la lógica relacionada con dichas entidades. Las mejoras se explican con más detalle en el *pull request* #51.

- **Implementación de operaciones CRUD sobre alimentos y productos:** se agregaron los repositorios y servicios necesarios para llevar a cabo estas funcionalidades.

- **Cálculo dinámico de la información nutricional:** se simplificó enormemente la lógica original, logrando que los cálculos de información nutricional (ya sea para alimentos con cantidades particulares o para productos completos) sean dinámicos y **no se almacenen en la BBDD**. Las mejoras se explican con más detalle en el *pull request* #52.

- **Transformación de la relación `User-Role` a `@ManyToMany`:** se modificó la asignación de roles, para que los usuarios pudieran tener más de uno, lo cual permite la creación de roles y funcionalidades más específicas. Se garantiza que, como mínimo, un usuario registrado tendrá `ROLE_USER`.

- **Refactorización de los DTOs de `User`:** se redujo el número de DTOs para usuarios, fusionando todos en `UserRequestDTO` y manejando las restricciones según formulario mediante **grupos de validación** (*OnCreate*, *OnUpdate*, etc.).

- **Creación de un `UserDataInitializer`:** este componente crea roles y un usuario por defecto en la aplicación.

- **Definición de casos de uso para gestión de usuarios:** se redactó una primera versión de casos de uso para esta funcionalidad, teniendo en cuenta las acciones que pueden realizar los usuarios según sus roles.

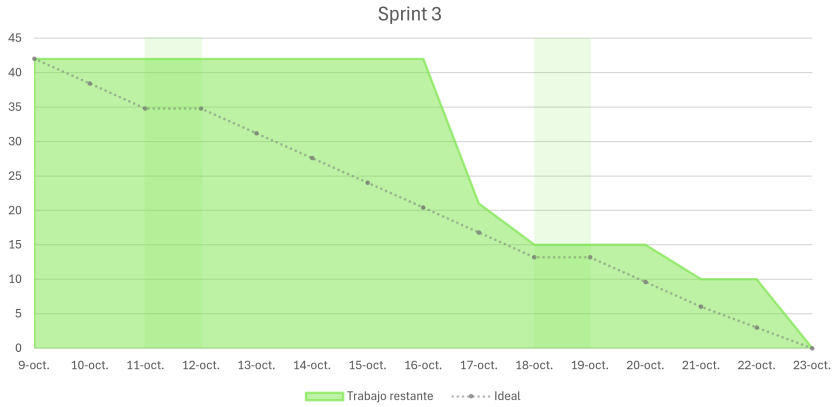


Figura A.4: Gráfico *burndown* del Sprint 3

### Revisión

Se acordó plasmar las mejoras respecto al TFG anterior en el apartado "*5. Aspectos relevantes del desarrollo del proyecto*", para reflejar el proceso de transformación seguido y las decisiones que llevaron a él.

Respecto a la arquitectura de la aplicación, se le planteó al tutor el **cambio de la arquitectura actual** monolítica, por una **arquitectura SPA** con *React*. Inicialmente, se había optado por usar *Thymeleaf* como motor de plantillas, y gestionar las vistas de la aplicación mediante controladores web. Sin embargo, la generación de las plantillas HTML y los controladores específicos para cada acción (p. ej. `UserEditController` o `UserManagementController`) resultaban un poco pesados y menos fáciles de escalar. Es por eso que se optó por la naturaleza reutilizable de los componentes de *React* para, en primera instancia, definir de forma más simple las tablas de gestión CRUD para cada entidad (y muchas otras mejoras).

Debido a que la idea del cambio de arquitectura surgió durante el desarrollo del *sprint*, hubo un *issue* que se pospuso para futuras iteraciones, relacionado con la **internacionalización** de la aplicación. Esto es debido a que la implementación de dicha funcionalidad cambia por completo dependiendo de la arquitectura. Por lo tanto, se pospuso para cuando se disponga de un *frontend* funcional.

Se acordó enfocar el siguiente *sprint* en la **migración a una arquitectura SPA con *React***, lo cual conlleva la refactorización de la aplicación *Spring Boot* para que actúe como API REST.

## *Sprint* 4

Duración: 2 semanas (23/10/2025 - 06/11/2025)

Este *sprint* se dedicó exclusivamente a la migración a *React*. Se refactorizó el proyecto de *Spring Boot* para convertirlo en API REST pura y se eliminaron las dependencias de *Thymeleaf*. Se construyó el nuevo proyecto de *React* y se reimplementaron las funcionalidades existentes hasta la fecha.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 4**.

### Objetivos

- **Investigación de mejores prácticas para la migración de arquitectura:** abarcó el aprendizaje de las bases de *React* y la selección de librerías útiles para el *frontend*. También se investigó sobre las prácticas estándar para refactorizar la API de *Spring Boot* y sobre los componentes necesarios para la comunicación entre ambos servicios.

- **Refactorización del *backend*:** se completó la migración completa a un diseño *RESTful*, eliminando toda la infraestructura web y reescribiendo los servicios y controladores para operar mediante DTOs y peticiones HTTP. Los cambios se explican con más detalle en el *pull request* #60.

- **Configuración de la autenticación y CORS:** se implementaron el servicio y controlador de autenticación (*login* y registro) en la API y se configuró CORS (*Cross-Origin Resource Sharing*) para permitir la comunicación entre la API y el *frontend* de *React*, estableciendo sesiones *stateless*.

- **Creación del proyecto y contenedor *Docker* para *React*:** para generar el proyecto inicial se utilizó la herramienta de construcción *Vite* [20]. Se creó el `Dockerfile` para la construcción del contenedor y se agregó el servicio a `docker-compose.yml`.

- **Implementación de la autenticación en *React*:** se creó un contexto de autenticación global para manejar el usuario de cada sesión. Se crearon las páginas para *login* y registro de usuarios.

- **Implementación de la gestión de usuarios en *React*:** se reimplementaron las funcionalidades existentes acerca del manejo de usuarios

y roles, junto con las interfaces asociadas. Las novedades se explican con más detalle en el *pull request* #74.

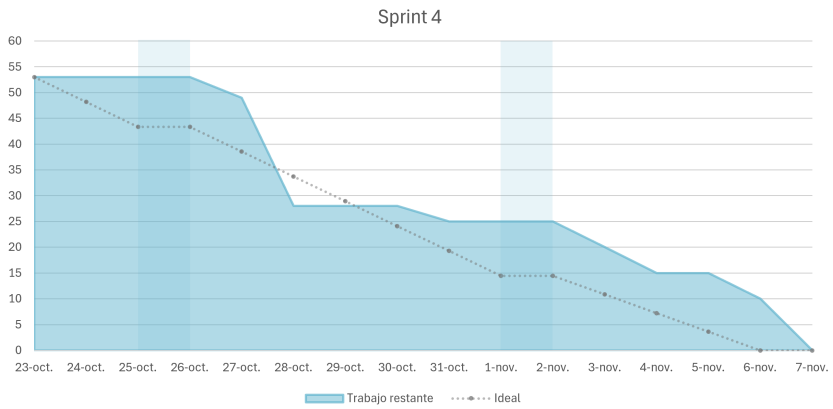


Figura A.5: Gráfico *burndown* del Sprint 4

### Revisión

Se acordó, para el siguiente *sprint*, continuar con la creación de páginas de gestión para las entidades restantes (gestión de productos ofertados). También se propuso ir avanzando con el resto de módulos que quedaban por implementar (pedidos, reservas de mesas), con el objetivo de ir cada vez centrando más el foco en el desarrollo *React* y poder dejar la API acabada en su primera versión.

## *Sprint* 5

Duración: 2 semanas (06/11/2025 - 20/11/2025)

En este *sprint* se implementaron en la API los módulos de gestión de reservas de mesas y gestión de pedidos, finalizando con ello la creación de entidades JPA y componentes *Spring Boot*. También se introdujo la granularidad de "permiso" a la gestión de roles y se internacionalizó la aplicación *React* para los idiomas inglés y español.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 5**.

### Objetivos

- **Creación de las entidades `DiningTable` y `Reservation`:** se definieron las entidades JPA para el módulo de gestión de reservas.
- **Implementación de operaciones CRUD sobre mesas y reservas:** se agregaron los repositorios, servicios y controladores necesarios para llevar a cabo estas funcionalidades. Los detalles de negocio se especifican en el *pull request* #93.
- **Creación de las entidades `Order`, `OrderItem`, `BarOrder` y `DiningOrder`:** aprovechando la existencia de varios "tipos" de pedidos, se implementó una jerarquía de **herencia**, la cual se detalla en el *pull request* #80.
- **Implementación de operaciones CRUD sobre pedidos:** se agregaron los repositorios, servicios y controladores necesarios para llevar a cabo estas funcionalidades. Se creó una factoría para elegir *mapper* según el tipo de pedido que llegase a la solicitud, sin que los componentes conozcan dicho tipo. El controlador general se utiliza para consultas sobre todas las entidades y gestión de ítems. Los controladores particulares se utilizan para consultas y operaciones CRUD por cada tipo, más eficientes.
- **Implementación de lógica de permisos para cada rol:** se creó un enumerado `Permission`, el cual alberga todos los permisos de la aplicación. A la entidad `Role` se le añadió una colección de permisos y se crearon métodos, *endpoints* y componentes *React* para manejarlos. Esto permite la creación de roles personalizados y un manejo más preciso de las acciones a realizar en la aplicación. Los cambios se explican con más detalle en los *pull requests* #81 y #96.
- **Internacionalización del *frontend React*:** se sustituyeron todas las cadenas *hardcodeadas* por referencias a archivos JSON con cadenas traducidas (*namespaces*). También se creó un componente para cambiar el idioma dinámicamente desde la aplicación.
- **Refactorización de `Recipe` a `Ingredient`:** la entidad `Recipe`, utilizada como tabla de unión entre `Food` y `Product`, pasó a denominarse `Ingredient`. También se eliminaron su repositorio y servicio particulares, trasladando la responsabilidad de su manejo a `Product`. Los cambios se explican con más detalle en el *pull request* #79.

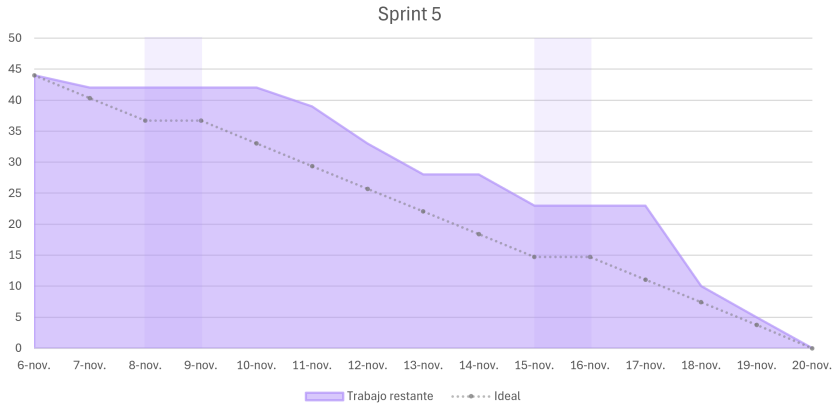


Figura A.6: Gráfico *burndown* del Sprint 5

### Revisión

Se acordó, para el siguiente *sprint*, generar una colección de **tests** para asegurar que la API REST funciona correctamente, antes de pasar el enfoque del desarrollo a *React*. Una vez probada, se deberían generar las interfaces de gestión para las entidades restantes, de manera que se dispusiera de una primera versión de la aplicación (aún sin mucha lógica incorporada, pero que permitiese gestionar las entidades involucradas).

## *Sprint* 6

Duración: 2 semanas (20/11/2025 - 04/12/2025)

Este *sprint* se utilizó para aplicar unas últimas mejoras a la API REST y desarrollar tests que comprueben el correcto funcionamiento de los servicios que ofrece. También se implementaron las interfaces de gestión restantes de las entidades añadidas en el anterior *sprint*.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 6**.

### Objetivos

- **Refactorización, limpieza y mejora de aspectos puntuales en la API:** se "estandarizó" la API *Spring Boot* para que las entidades y componentes fuesen uniformes, consistentes y que siguieran un patrón similar. Entre las modificaciones se encuentran la incorporación de

campos de auditoría (`createdAt` y `updatedAt`), la simplificación y eliminación de algunos métodos, la refactorización de los *mappers* para aprovechar mejor la lógica de conversiones, etc.

- **Creación de pruebas unitarias para *mappers*:** al tener lógica personalizada, se hacía necesario probar los *mappers* generados por *Mapstruct*, para comprobar que la conversión DTO ↔ Entidad funciona de la manera esperada. Como los comportamientos de los componentes involucrados son fáciles de simular, se utilizó *Mockito* para generar las pruebas, puesto que proporciona tests muy rápidos y aislamiento completo.

- **Creación de pruebas unitarias para servicios:** se probó la lógica de negocio (validaciones, excepciones lanzadas, métodos de consulta) de todas las entidades existentes en la API. Se utilizó *Mockito* por su rapidez y aislamiento.

- **Creación de pruebas de integración para servicios:** se crearon pruebas con dependencias reales utilizando `@SpringBootTest` y transaccionalidad, para probar la **persistencia**, relaciones entre entidades y conflictos que solo se dan en casos "reales". Estas pruebas de integración son más lentas puesto que necesitan cargar todo el contexto *Spring*. Al ser más costosas, solo se crearon en entidades más complejas o que "gestionan a otras" (normalmente, las **entidades propietarias** de la relación).

- **Implementación de la gestión de productos en *React*:** se crearon páginas y componentes para visualizar listados, crear, editar y eliminar alimentos y productos, junto con su información nutricional. Se reutilizaron los componentes y *layouts* de gestión creados anteriormente. Las novedades se explican con más detalle en el *pull request* #107.

- **Implementación de la gestión de reservas en *React*:** de igual forma, se crearon páginas y componentes para gestión básica con operaciones CRUD sobre mesas y reservas. Las novedades se explican con más detalle en el *pull request* #106.

- **Creación de *dashboard* para gestionar pedidos:** se creó una vista general que muestra tanto un listado de pedidos pendientes como un listado de productos ofertados. Se permite añadir productos al pedido actual haciendo *click* en ellos y se permite alternar entre pedidos de bar o comedor.

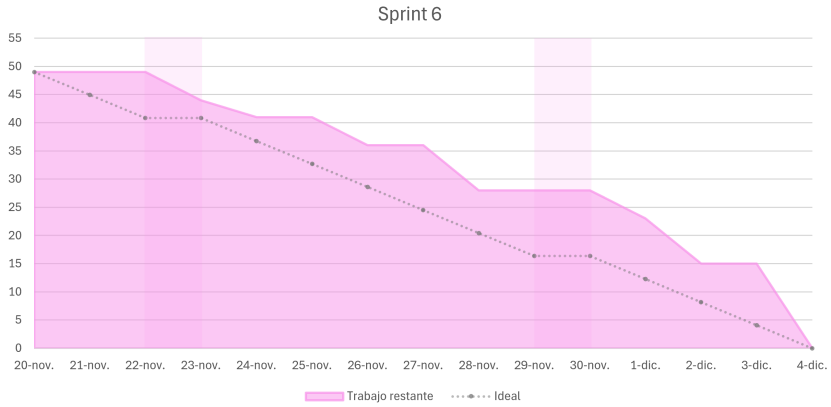


Figura A.7: Gráfico *burndown* del Sprint 6

### Revisión

Habiendo implementado las páginas de gestión principales y probado el comportamiento de la API, se acordó preparar la aplicación para su primer ***release***. Para ello, se propuso implementar los componentes faltantes para lograr una navegación fluida entre las páginas existentes (mediante un *navbar*) y asegurar el correcto manejo de excepciones y errores que puedan producirse. También deberán corregirse los errores pendientes del *dashboard* de pedidos.

Como añadidos futuros, para una siguiente versión de la API, se propuso la generación de **menús**. Actualmente se "añaden" los ítems individualmente al pedido, pero podría haber un conjunto de productos que constituya un menú, con un precio fijado y características particulares. Esto haría que fuese necesario añadir un campo de "tipo" a la entidad `Product`, para saber si se trata de un primer plato, postre, bebida, etc. También se propuso un filtrado por tipos o características de productos desde el *dashboard* de pedidos.

## *Sprint 7*

Duración: 2 semanas (04/12/2025 - 18/12/2025)

Así como el *sprint* anterior se dedicó mayoritariamente a solventar los errores de la API, este *sprint* se enfocó en resolver aspectos pendientes del *frontend* e implementar los componentes restantes de navegación, para lograr una **primera versión funcional** de la aplicación.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 7**.

### Objetivos

- **Implementación de una barra de navegación principal:** los componentes que la forman se explican con más detalle en el *pull request* #117.
- **Implementación de autenticación con sesiones de *Spring Security*:** se configuró la persistencia de la sesión para el *frontend React* mediante *cookies* `JSESSIONID`.
- **Implementación de formulario para crear/editar pedidos:** se añadieron al *dashboard* botones para crear y editar pedidos de distintos tipos. También se habilitó la eliminación de pedidos desde la interfaz.
- **Mejoras y resolución de errores pendientes:** cuestiones pendientes relacionadas con la internacionalización, estados de carga, manejo de errores y más. Las mejoras se explican con más detalle en el *pull request* #124.
- **Creación de *workflow* para ejecutar pruebas de la API:** se utilizó *GitHub Actions* para configurar la ejecución de los tests definidos para la API *Spring Boot* con cada *commit* realizado sobre el *backend*.
- **Redacción del apartado *C.2 Diseño de datos*:** se describieron las entidades y relaciones que conforman el modelo y se generaron diagramas EER.

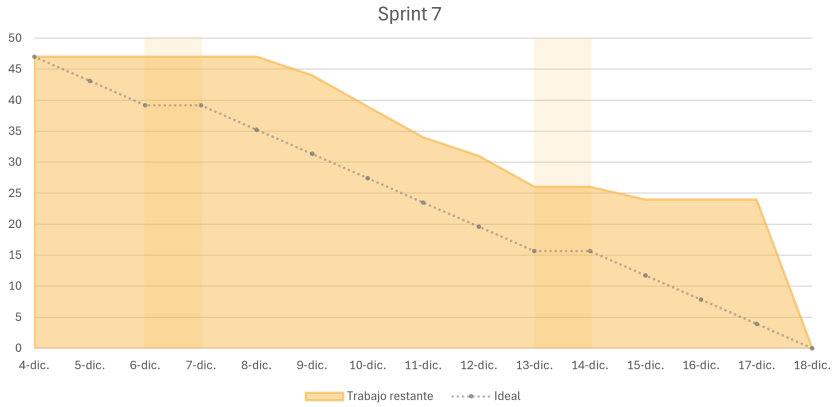


Figura A.8: Gráfico *burndown* del Sprint 7

### Revisión

Tras el visto bueno del tutor, se lanzó la **v0.8.0-beta** de la aplicación.

Para la siguiente versión de la aplicación, se acordó incluir la **sección nutricional** y **comunicación con la API externa** *Open Food Facts*. Esto incluye, como mínimo, la búsqueda y creación de alimentos a partir de la API, la extracción de información nutricional de los productos y la visualización de gráficos para mostrar dicha información de manera más amigable para el usuario.

También se planteó la posible inclusión del ***Nutri-Score*** como medida adicional y más reconocible para los clientes. Esto requeriría documentarse acerca de cómo se calcula dicha medida o buscar herramientas gratuitas que realicen este cálculo y adaptar nuestros datos para dicha operación.

## *Sprint* 8

Duración: 3 semanas (18/12/2025 - 08/01/2026)

En este *sprint* se redactaron los apartados principales de la memoria y se integró la conexión con la API nutricional *Open Food Facts* (*OFF*), permitiendo la creación y persistencia de un alimento (*Food*) a partir de su búsqueda en la API externa.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 8**.

**Objetivos**

- **Investigación acerca de los datos devueltos por *OFF*:** para poder adaptar los campos devueltos por la API *OFF*, se hizo necesario revisar su documentación para saber qué campos podrían ser aprovechados por nuestro modelo y cómo se podían transformar.
- **Búsqueda de alimentos de *OFF* desde *React*:** se implementó un componente de búsqueda (por nombre o información relacionada) en la página de creación de un alimento, permitiendo realizar peticiones a la API externa y visualizar un listado paginado con los resultados devueltos. Se explica con más detalle en el *pull request* #138.
- **Creación de alimentos de *OFF* desde Spring Boot:** se implementaron mecanismos para adaptar los datos de la API externa al modelo del *backend* y posteriormente persistir el alimento obtenido. Se explica con más detalle en el *pull request* #138.
- **Creación de gráficos y tablas nutricionales:** los macronutrientes de un producto ahora se visualizan en forma de gráfico de donut, también con la opción de visualizarlos en tablas junto a ingestas de referencia.
- **Redacción de apartados principales de la memoria:** se redactaron casi por completo los apartados "*3. Conceptos teóricos*", "*4. Técnicas y herramientas*" y "*5. Aspectos relevantes del desarrollo del proyecto*". Queda pendiente la redacción de algunos subapartados menores y la revisión por parte del tutor.

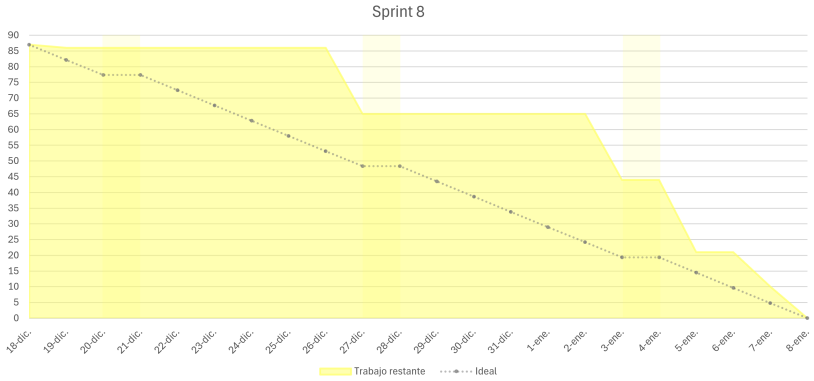


Figura A.9: Gráfico *burndown* del Sprint 8

### Revisión

Con la sección nutricional mayormente implementada (a falta de alérgenos y medidas nutricionales puntuales), se acordó dar por **finalizada la implementación de funcionalidades** de la aplicación. Los *sprints* posteriores se dedicarán a la corrección de errores, modificación de interfaces y mejoras menores, así como la redacción de la documentación y anexos.

En esta fase final del desarrollo se acordó volver a *sprints* de 1 semana, para agilizar las revisiones. El siguiente *sprint* tendría como objetivo principal la finalización y revisión de la **memoria** principal del trabajo.

## *Sprint* 9

Duración: 1 semana (08/01/2026 - 15/01/2026)

En este *sprint* se finalizó la redacción de la memoria en su primera versión (a falta de conclusiones y aspectos puntuales) y se redactaron de forma más estructurada los requisitos y casos de uso de la aplicación. También se implementaron aspectos pendientes del módulo de pedidos, como los cambios de estado y la visualización de un historial.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 9**.

### Objetivos

- **Implementación de cambios de estado para pedidos:** se añadió un botón en el *dashboard* para permitir cambiar el estado de los pedidos,

de forma que puedan completarse, cancelarse o avanzar en su ciclo de vida (*"pendiente"*, *"en proceso"*, etc.).

- **Creación de historial de pedidos:** se añadió una página para visualizar el registro de todos los pedidos creados en la aplicación, permitiendo filtrar por tipo de pedido. Se diferencia del resto de páginas de listado en que no permite la edición o borrado de pedidos (para mantener la transparencia).

- **Corrección de estados de carga en *dashboard*:** se mejoró el manejo de estado de carga en el *dashboard* de pedidos, puesto que anteriormente los componentes se sincronizaban mal al cargar y no mostraban *feedback* del progreso al usuario.

- **Redacción de apartados restantes de la memoria:** se redactaron el *abstract* y la introducción, junto con los apartados "*2. Objetivos del proyecto*" y "*6. Trabajos relacionados*". Esto dejó pendiente la redacción de conclusiones y algunos aspectos relevantes.

- **Redacción estructurada de requisitos y casos de uso:** anteriormente se habían listado los requisitos funcionales de la aplicación y se habían descrito algunos CU del módulo de usuarios. Se reformularon y describieron mejor los requisitos y se completaron los casos de uso, para incluir a todos los módulos de la aplicación.

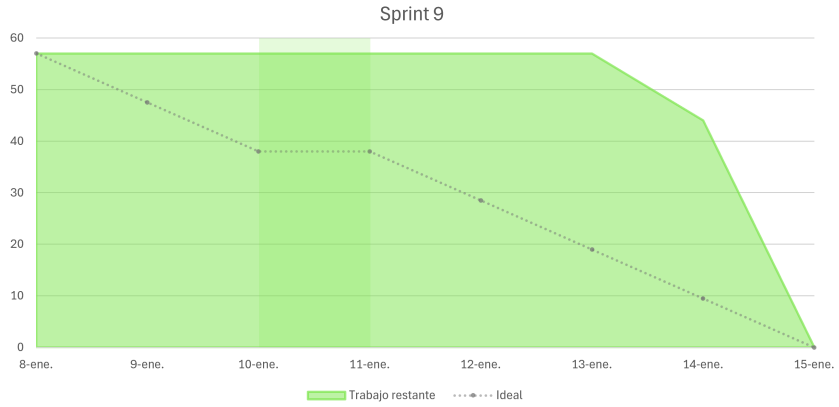


Figura A.10: Gráfico *burndown* del Sprint 9

### Revisión

Se revisaron junto al tutor la memoria y anexos escritos hasta el momento, y se comentaron aspectos a añadir en la documentación y aspectos de usabilidad de la aplicación.

Se acordó, para el siguiente *sprint*, incluir la mención de **alérgenos** y **métricas nutricionales** en alimentos y productos, que era un aspecto pendiente mencionado en el *Sprint* 8. También corregir, en la medida de lo posible, aspectos relacionados con la visualización de módulos funcionales según permisos (actualmente en el *frontend* solo se considera al usuario administrador).

## *Sprint* 10

Duración: 1 semana (15/01/2026 - 22/01/2026)

En este *sprint* se añadieron campos para alérgenos y métricas nutricionales a las entidades de alimentos y productos, junto con su integración y visualización en el *frontend*. También se modificaron los componentes de navegación de *React*, para que la visualización de los módulos dependiera de los permisos que posee el usuario autenticado.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 10**.

### Objetivos

- **Inclusión de alérgenos y métricas nutricionales:** se implementó el manejo de listas de alérgenos (para alimentos y productos) y métricas nutricionales (*Nutri-Score* y grupo NOVA). Estos campos se obtienen de la API *Open Food Facts* y, en caso de los alérgenos, también pueden añadirse manualmente. Se explica con más detalle en el *pull request* #151.

- **Restricción de acceso en *frontend* mediante permisos:** se refactorizaron los componentes de navegación de *React* para que el acceso a rutas y las acciones a realizar estuvieran restringidos por los permisos del usuario autenticado, en vez de comprobar estáticamente su rol. Se explica con más detalle en el *pull request* #152.

- **Corrección de sesiones no persistentes:** se solucionaron los errores en la comunicación *React-Spring Boot*, que hacían que las *cookies*

`JSESSIONID` no se almacenaran, lo cual causaba que el usuario perdiese su sesión al recargar la página.

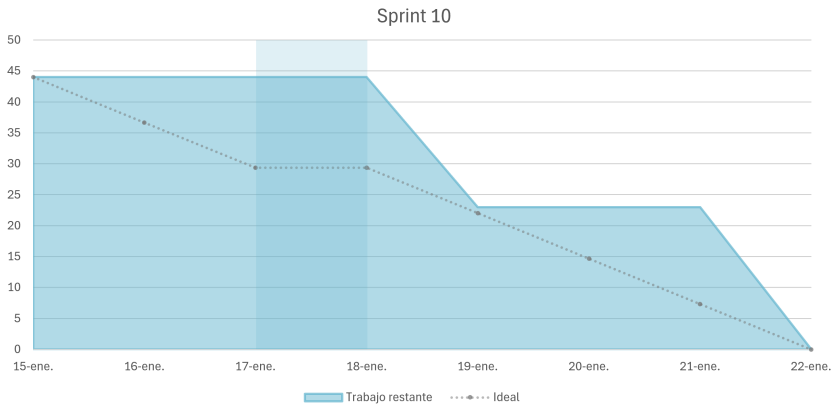


Figura A.11: Gráfico *burndown* del Sprint 10

### Revisión

Se detectó que las búsquedas a *Open Food Facts* generaban errores si el navegador utilizado era *Mozilla Firefox*. Se optó por **trasladar las peticiones de búsqueda** a *Spring Boot*, desacoplando por completo el *frontend* de la implementación de la API externa y evitando problemas de CORS.

Se acordó seguir corrigiendo errores y desajustes en la interfaz, con el objetivo de preparar la aplicación para su segundo ***release***. También se acordó integrar ***SonarCloud*** para disponer de métricas sobre la calidad del código actual y posibles errores que no hayan sido detectados.

## *Sprint* 11

Duración: 1 semana (22/01/2026 - 29/01/2026)

Este *sprint* se dedicó a la corrección y mejora de aspectos pendientes, como cambios de estado para mesas y reservas, filtrado de alérgenos en el *dashboard* de pedidos y modificación de la interfaz de la aplicación, para así lograr una **segunda versión funcional** de la aplicación.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 11**.

### Objetivos

- **Añadir mejoras, correcciones y finalizar aspectos pendientes de la aplicación:** los componentes modificados se explican con más detalle en el *pull request* #157.
- **Integración con *SonarCloud*:** se agregó la comunicación con *SonarCloud* en el *workflow* existente de *GitHub Actions*, que ejecutaba los tests de *Spring Boot*.

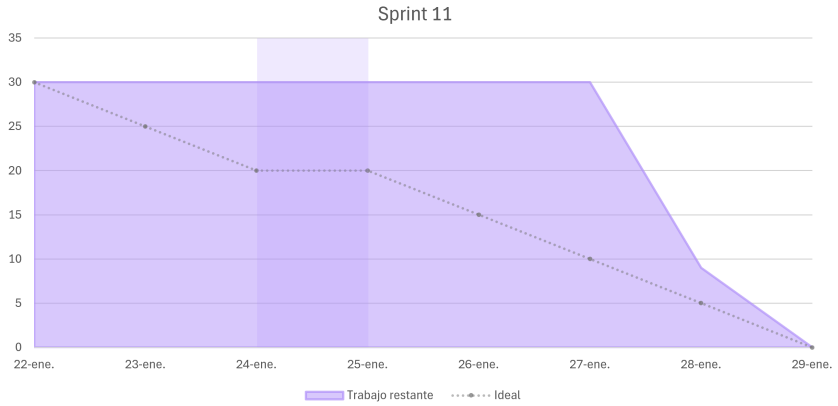


Figura A.12: Gráfico *burndown* del Sprint 11

### Revisión

Tras el visto bueno del tutor, se lanzó la **v0.9.0-beta** de la aplicación.

## *Sprint* 12

Duración: 2 semanas (29/01/2026 - 12/02/2026)

Este *sprint* final se basó en la finalización de la redacción de los anexos y en la corrección de aspectos de la aplicación, así como en mejoras de diseño en la interfaz. También se generaron **datos por defecto** en las tablas de la base de datos, para así contar con registros existentes en el arranque de la aplicación y facilitar su uso.

Los *issues* creados para cumplir los objetivos definidos se pueden ver en: **Sprint 12**.

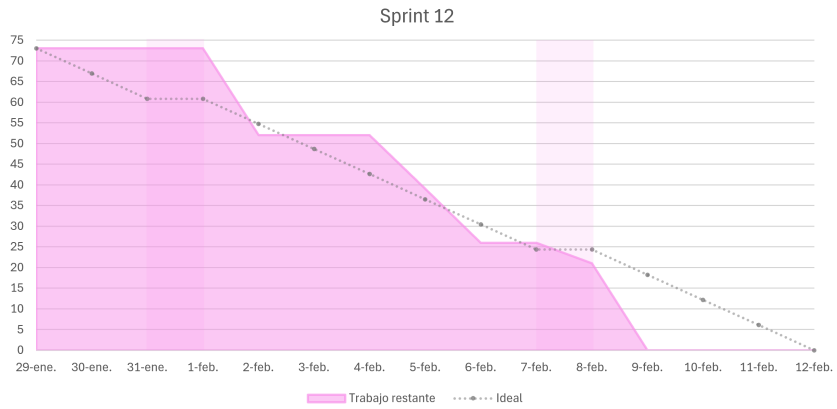


Figura A.13: Gráfico *burndown* del Sprint 12

## Gráfico *Burn-up*

El gráfico presentado en la figura A.14 permite visualizar de manera dinámica el progreso del trabajo completado (línea morada) frente al alcance total planificado (línea verde). El eje vertical se corresponde con el número de tareas (*issues*) agregadas al tablero Kanban de *GitHub Projects*.

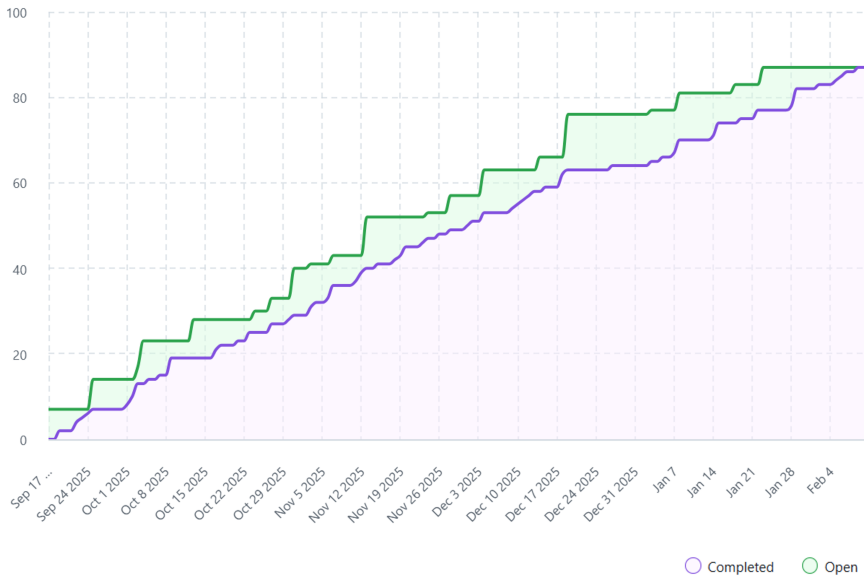


Figura A.14: Gráfico *burnup* del proyecto

En la medida de lo posible, se ha procurado que la creación y asignación de nuevas tareas a *sprints* se sitúe al **comienzo** (o también a mitad, en

caso de *sprints* de 2 semanas) de cada *sprint*. Tras los picos causados por el incremento en el número de *issues*, se puede observar que, en la mayoría de casos, la línea de alcance teórico permanece prácticamente **constante**, lo que refleja que el trabajo se ejecutó conforme a lo planificado **sin añadir carga adicional** de forma imprevista.

# A.3. Estudio de viabilidad

## Viabilidad económica

El análisis de la viabilidad económica tiene como objetivo estimar los costes que supondría el desarrollo, despliegue y puesta en marcha del proyecto *Orderly* en un entorno empresarial real. Para realizar este cálculo, se han considerado los precios de mercado actuales en España y se ha supuesto un escenario de desarrollo de **5 meses** de duración con una jornada laboral completa (**8 horas** diarias).

### Costes de personal

El desarrollo ha sido realizado íntegramente por una ingeniera informática con un perfil *Junior*. Para el cálculo, se ha tomado como referencia un salario bruto anual promedio para este perfil en España de 21.000 € [7].

Para obtener el coste real para la empresa, es necesario sumar al salario bruto los costes de la Seguridad Social a cargo de la empresa (contingencias comunes, desempleo, formación, FOGASA, etc.), que se estiman aproximadamente en un 32 % del salario bruto.

Los cálculos mensuales y totales para los 5 meses de desarrollo son los siguientes:

1. Salario mensual bruto:

$$\frac{21.000\,€}{12\,\text{meses}} = 1.750,00\,€/\text{mes}$$

2. Coste Seguridad Social (32 %):

$$1.750,00\,€ \times 0,32 = 560,00\,€/\text{mes}$$

3. Coste mensual total para la empresa:

$$1.750,00\,€ + 560,00\,€ = 2.310,00\,€/\text{mes}$$

4. Coste total del proyecto (5 meses):

$$2.310,00\,€ \times 5 = 11.550,00\,€$$

### Costes de *hardware*

Para el desarrollo de la aplicación se ha utilizado un equipo informático personal con las siguientes características:

- **Modelo:** *HP Laptop 15s-fq2xxx*
- **Procesador:** *11th Gen Intel Core i5-1135G7*
- **RAM:** 8GB
- **Precio de mercado estimado:** 600,00 €

El coste imputable al proyecto se calcula mediante la amortización del equipo durante el tiempo de uso (5 meses), estimando una vida útil estándar para equipos informáticos de 4 años (48 meses).

1. Amortización mensual:

$$\frac{600,00\,€}{48\,\text{meses}} = 12,50\,€/\text{mes}$$

2. Coste total *hardware* (5 meses):

$$12,50\,€ \times 5 = 62,50\,€$$

### Costes de *software*

El proyecto *Orderly* se ha desarrollado priorizando el uso de tecnologías de código abierto y herramientas gratuitas, lo que reduce considerablemente los costes de licencias. El stack tecnológico incluye:

- **Entorno de desarrollo:** *Visual Studio Code* (Gratuito).
- ***Backend***: Java (OpenJDK) y *Spring Boot* (*Open Source - Apache 2.0*).
- *Frontend*: *React* y *Vite* (*Open Source* - MIT).
- **Base de datos:** *MySQL Community Edition* (*Open Source* - GPL).
- **Contenerización:** *Docker Desktop* (versión personal/educativa gratuita).
- **APIs externas:** *Open Food Facts* (acceso gratuito para uso no comercial).
- **Otras herramientas:** *Git*, *Postman*, *SonarCloud* (plan gratuito para proyectos públicos).

El único coste de software imputable corresponde a la licencia del **sistema operativo** *Microsoft Windows 11 Home*, con precio aproximado de 145,00€. Al igual que el *hardware*, se amortiza en 4 años.

1. Amortización mensual:

$$\frac{145,00\,€}{48\,\text{meses}} \approx 3,02\,€/\text{mes}$$

2. Coste total *software* (5 meses):

$$3,02\,€ \times 5 = 15,10\,€$$

### Costes de infraestructura y despliegue

Aunque el proyecto se ha desarrollado en un entorno local utilizando *Docker*, para una puesta en **producción** real sería necesario alojar los contenedores (*React* + *Spring Boot* + *MySQL*) en un servicio de computación en la nube (VPS[1]).

Una opción económica sería ***Hetzner CX22*** (2 vCPU, 4 GB RAM), cuya tarifa representativa es 3,79 €/mes (plan económico de *Hetzner* [8]).

- Cálculo *hosting* para 5 meses:

$$3,79 \times 5 = 18,95\ €$$

- Adicionales:
  - **Dominio** `.es` (estimación anual $\approx 12\,€ \rightarrow$ coste por 5 meses $\approx 12 \times \frac{5}{12} = 5\,€$).
  - **Certificado SSL**: *Let's Encrypt* (gratuito), asumido como 0 €.

El presupuesto total sería aproximadamente $18,95 + 5 = 23,95\,€$.

### Resumen de costes

En la tabla A.2 se presenta el desglose final de los costes estimados para el desarrollo de *Orderly*, incluyendo el Impuesto sobre el Valor Añadido (IVA) del 21 % vigente en España.

[1] *Virtual Private Server*

Tabla A.2: Resumen de costes económicos del proyecto

| **Concepto** | **Coste estimado (€)** |
|---|---:|
| Recursos Humanos (5 meses) | 11.550,00 |
| *Hardware* (Amortización) | 62,50 |
| *Software* (Sistema Operativo) | 15,10 |
| Infraestructura (*Cloud*) | 23,95 |
| **Total Neto** | **11.651,55** |
| IVA (21 %) | 2.446,83 |
| **TOTAL PROYECTO** | **14.098,38** |

## Viabilidad legal

El desarrollo de *Orderly* se ha apoyado en el uso de diversas bibliotecas, *frameworks* y herramientas de terceros. Para garantizar la viabilidad legal del proyecto, es fundamental analizar las licencias de estos componentes y asegurar que la licencia final escogida para el *software* sea compatible con las obligaciones heredadas de las tecnologías utilizadas.

En la tabla A.3 se presenta un resumen de las licencias de las principales tecnologías y librerías integradas en el proyecto, tanto del *backend* (*Spring Boot*) como del *frontend* (*React*).

La gran mayoría de las librerías utilizadas (*React*, *Mantine*, *Lombok*, etc.) se distribuyen bajo la licencia ***MIT***, la cual es altamente permisiva. Otras, como *Spring Boot* o *Docker*, utilizan ***Apache 2.0***, que también permite el uso comercial y la modificación del *software*, siendo compatible con licencias permisivas. Respecto a *Open Food Facts*, la licencia ***ODbL*** aplica sobre los datos, permitiendo su uso siempre que se atribuya la fuente, lo cual no impide licenciar el código de la aplicación bajo otros términos.

### Selección de licencia para el *software*

Considerando la naturaleza de las dependencias utilizadas y el objetivo académico del proyecto, se ha optado por liberar el código fuente de *Orderly* bajo la licencia ***MIT License***.

Esta decisión se justifica por los siguientes motivos:

- **Compatibilidad:** es compatible con la mayoría de las librerías utilizadas en el *stack* tecnológico (*React*, *Spring Boot*, etc.).

- **Permisividad:** otorga una gran libertad a futuros desarrolladores o usuarios, permitiendo usar, copiar, modificar, fusionar, publicar y distribuir el *software* sin restricciones complejas, siempre que se mantenga el aviso de derechos de autor original.
- **Fomento de la colaboración:** al ser una licencia estándar en el ecosistema *open source*, facilita que otros estudiantes o desarrolladores puedan reutilizar el código para futuros Trabajos de Fin de Grado o proyectos derivados.

### Selección de licencia para la documentación

Para la memoria y la documentación técnica del proyecto (incluidos los presentes anexos), se ha seleccionado la licencia ***Creative Commons Attribution 4.0 International*** (**CC BY 4.0**).

Esta licencia permite compartir y adaptar el material para cualquier propósito, incluso comercial, con la única condición de otorgar el crédito apropiado al autor original. Se considera la opción más adecuada para un trabajo académico, ya que fomenta la divulgación del conocimiento y permite que la documentación sirva de referencia para futuros alumnos, protegiendo al mismo tiempo la autoría intelectual del trabajo.

Tabla A.3: Resumen de licencias de las herramientas y tecnologías utilizadas

| **Herramienta / Librería** | **Uso en el proyecto** | **Licencia** |
|---|---|---|
| *Spring Boot* | *Framework del backend* | Apache License 2.0 |
| *React* | Biblioteca del *frontend* | MIT License |
| *MySQL* | Sistema de Base de Datos | GPL v2 |
| *Docker* | Despliegue y contenerización | Apache License 2.0 |
| *Vite* | Empaquetado y entorno *frontend* | MIT License |
| *Mantine* | Componentes de UI | MIT License |
| *Lombok* | Utilidad para reducir código Java | MIT License |
| *MapStruct* | Mapeo de objetos (DTOs) | Apache License 2.0 |
| *Zustand* | Gestión de estado global | MIT License |
| *i18next* | Internacionalización | MIT License |
| *Recharts* | Visualización de gráficos | MIT License |
| *Open Food Facts* | Fuente de datos nutricionales | ODbL |

*Apéndice* $B$

# Especificación de Requisitos

## B.1. Introducción

Este apéndice detalla la base técnica y funcional del proyecto, estableciendo el marco de trabajo necesario para el desarrollo del *software*. En las siguientes secciones se describen tanto las metas estratégicas como el comportamiento esperado del sistema, proporcionando una hoja de ruta clara sobre cómo la aplicación debe interactuar con sus usuarios.

## B.2. Objetivos generales

Los objetivos generales que definen el propósito y alcance principal de la aplicación son los siguientes:

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.

## B.3. Catálogo de requisitos

A continuación se detallan las funcionalidades que la aplicación debe cumplir para satisfacer los objetivos generales.

### Requisitos funcionales

- **RF-1 Gestión de alimentos:** la aplicación debe permitir la gestión de alimentos.
  - **RF-1.1 Crear alimento:** la aplicación debe permitir crear un alimento, aportando su nombre, porción en gramos e información nutricional asociada.
    - **RF-1.1.1 Crear alimento manualmente:** el usuario debe poder crear un alimento introduciendo manualmente sus campos.
    - **RF-1.1.2 Crear alimento mediante API externa:** el usuario debe poder crear un alimento mediante su búsqueda en una API nutricional externa.
  - **RF-1.2 Editar alimento:** el usuario debe poder editar la información asociada a un alimento existente.
  - **RF-1.3 Eliminar alimento:** el usuario debe poder eliminar un alimento existente.
  - **RF-1.4 Calcular información nutricional:** la aplicación debe calcular los valores nutricionales de un alimento en base a su cantidad en gramos.
  - **RF-1.5 Listar alimentos:** el usuario debe poder listar todos los alimentos presentes en la aplicación.
- **RF-2 Gestión de productos:** la aplicación debe permitir la gestión de productos.
  - **RF-2.1 Crear producto:** el usuario debe poder crear un producto (plato, ración), aportando su nombre y precio unitario.
  - **RF-2.2 Editar producto:** el usuario debe poder editar la información asociada a un producto existente.
  - **RF-2.3 Eliminar producto:** el usuario debe poder eliminar un producto existente.

- **RF-2.4 Gestionar ingredientes de un producto:** el usuario debe poder gestionar los ingredientes de un producto.
  - **RF-2.4.1 Agregar ingredientes:** el usuario debe poder agregar ingredientes (alimentos) a un producto.
  - **RF-2.4.2 Quitar ingredientes:** el usuario debe poder retirarle ingredientes a un producto.
  - **RF-2.4.3 Modificar porciones:** el usuario debe poder modificar la cantidad (en gramos) de un ingrediente perteneciente a un producto.
- **RF-2.5 Calcular información nutricional:** la aplicación debe calcular los valores nutricionales totales del producto mediante el sumatorio proporcional de sus ingredientes, basándose en la cantidad de cada uno de ellos presente en la composición final del producto.
- **RF-2.6 Ver productos:** el usuario debe poder visualizar toda la información relativa a un producto, incluyendo su información nutricional e ingredientes.
- **RF-2.7 Listar productos:** el usuario debe poder listar todos los productos presentes en la aplicación.

- **RF-3 Gestión de mesas:** la aplicación debe permitir la gestión de mesas del local.
  - **RF-3.1 Crear mesa:** el usuario debe poder crear una mesa, aportando nombre y capacidad máxima de comensales.
  - **RF-3.2 Editar mesa:** el usuario debe poder editar la información asociada a una mesa existente.
  - **RF-3.3 Desactivar mesa:** el usuario debe poder marcar una mesa como "inactiva", para realizar un borrado lógico sin perder el historial de reservas.
  - **RF-3.4 Eliminar mesa:** el usuario debe poder eliminar una mesa existente.
  - **RF-3.5 Listar mesas:** el usuario debe poder listar todas las mesas registradas en la aplicación.

- **RF-4 Gestión de reservas:** la aplicación debe permitir la gestión de reservas de mesas del local.
  - **RF-4.1 Crear reserva:** el usuario debe poder reservar una mesa para una fecha concreta.

- **RF-4.2 Editar reserva:** el usuario debe poder editar la información asociada a una reserva existente, como los datos del cliente, la fecha o el estado de la reserva.
- **RF-4.3 Eliminar reserva:** el usuario debe poder eliminar una reserva existente.
- **RF-4.4 Listar reservas:** el usuario debe poder listar todas las reservas registradas en la aplicación.

- **RF-5 Gestión de pedidos:** la aplicación debe permitir la gestión de pedidos (comandas).
  - **RF-5.1 Crear pedido:** el usuario debe poder crear un pedido para la barra o el comedor.
  - **RF-5.2 Editar pedido:** el usuario debe poder editar la información asociada a un pedido existente, como el estado del pedido.
  - **RF-5.3 Eliminar pedido:** el usuario debe poder eliminar un pedido existente que no haya sido finalizado (por pago o cancelación).
  - **RF-5.4 Gestionar líneas de pedido:** el usuario debe poder gestionar los ítems que conforman un pedido.
    - **RF-5.4.1 Agregar productos:** el usuario debe poder agregar productos a un pedido.
    - **RF-5.4.2 Quitar productos:** el usuario debe poder retirar productos de un pedido.
    - **RF-5.4.3 Modificar cantidades:** el usuario debe poder modificar la cantidad unitaria de un producto que se incluye en el pedido.
  - **RF-5.5 Calcular precio total:** la aplicación debe calcular el precio total de un pedido, basándose en los precios unitarios y cantidades de los productos que lo forman.
  - **RF-5.6 Ver pedidos:** el usuario debe poder visualizar toda la información relativa a un pedido.
  - **RF-5.7 Listar pedidos:** el usuario debe poder visualizar un listado de todos los pedidos registrados en la aplicación.
    - **RF-5.7.1 Listar pedidos en curso:** el usuario debe poder visualizar los pedidos que no hayan sido finalizados.

- **RF-6 Gestión de roles:** la aplicación debe permitir la gestión de roles y permisos.
  - **RF-6.1 Crear rol:** el usuario debe poder crear un rol asociado a una serie de permisos.
  - **RF-6.2 Editar rol:** el usuario debe poder editar la información asociada a un rol existente, incluyendo los permisos asociados.
  - **RF-6.3 Eliminar rol** el usuario debe poder eliminar un rol existente.
  - **RF-6.4 Listar roles:** el usuario debe poder listar todos los roles existentes en la aplicación.
- **RF-7 Gestión de usuarios:** la aplicación debe permitir la gestión de usuarios.
  - **RF-7.1 Crear usuario:** la aplicación debe permitir crear un usuario, aportando un *username* y una contraseña.
  - **RF-7.2 Editar usuario:** la aplicación debe permitir editar la información asociada a un usuario existente.
    - **RF-7.2.1 Editar perfil propio:** el usuario debe poder editar su información asociada.
    - **RF-7.2.2 Editar perfil ajeno:** el usuario con permisos suficientes debe poder editar la información asociada a otros usuarios, incluyendo sus roles.
  - **RF-7.3 Eliminar usuario:** la aplicación debe permitir eliminar un usuario existente.
  - **RF-7.4 Gestionar roles de un usuario:** la aplicación debe permitir la gestión de los roles de un usuario.
    - **RF-7.4.1 Agregar roles:** el usuario con permisos suficientes debe poder agregar roles a un usuario, incluido él mismo.
    - **RF-7.4.2 Quitar roles:** el usuario con permisos suficientes debe poder quitarle roles a un usuario, incluido él mismo.
  - **RF-7.5 Ver perfil propio:** el usuario debe poder visualizar toda la información relativa a sí mismo, incluyendo sus roles.
  - **RF-7.6 Listar usuarios:** la aplicación debe permitir listar todos los usuarios registrados en ella.
- **RF-8 Autenticación de usuarios:** la aplicación debe permitir acceder a los usuarios al sistema mediante su *username* y contraseña.

- **RF-9 Búsqueda de alimentos mediante API *Open Food Facts*:** la aplicación debe permitir la búsqueda de alimentos presentes en la base de datos de *Open Food Facts*.
- **RF-10 Gráficas y tablas nutricionales:** la aplicación debe permitir la visualización de gráficas y tablas que informen al usuario de la proporción de componentes nutricionales de un producto.
- **RF-11 Detección de alérgenos en alimentos y productos:** la aplicación debe mostrar los alérgenos presentes en cada alimento y producto.

## Requisitos no funcionales

- **RNF-1 Rendimiento y escalabilidad:** la aplicación debe mantener tiempos de respuesta adecuados bajo carga operativa normal y picos moderados de uso.
- **RNF-2 Disponibilidad:** el sistema debe estar operativo durante el horario de apertura típico de un local de restauración.
- **RNF-3 Seguridad y protección de datos:** el sistema debe garantizar la confidencialidad, integridad y autenticación de los datos, especialmente los de usuarios y transacciones.
- **RNF-4 Usabilidad y experiencia de usuario (UX):** la interfaz debe ser intuitiva, eficiente y requerir un entrenamiento mínimo para el personal. Debe existir *feedback* visual claro para acciones (éxito, error, advertencia).
- **RNF-5 Mantenibilidad y extensibilidad:** el código debe estar estructurado para facilitar su mantenimiento, corrección de errores y adición de nuevas funcionalidades.
- **RNF-6 Interoperabilidad y consumo de API externa:** la integración con una API externa de terceros (RF-1.1.2, RF-9) debe ser robusta y manejar adecuadamente los fallos de la fuente externa.
- **RNF-7 Internacionalización (i18n):** la aplicación debe soportar al menos dos idiomas (español e inglés).
- **RNF-8 Portabilidad y despliegue:** la aplicación debe poder desplegarse de forma sencilla y reproducible.

- **RNF-9 Fiabilidad de los datos:** la integridad y consistencia de los datos, especialmente en relaciones bidireccionales, debe estar garantizada.

## B.4. Especificación de requisitos

Este apartado define el comportamiento esperado del sistema, transformando las necesidades del negocio en una estructura técnica detallada para delimitar su alcance funcional. La figura B.1 ofrece una visión global de estas interacciones, sirviendo como base para comprender los diversos flujos de la aplicación.

Debido a la naturaleza modular de la aplicación, todas las entidades del sistema comparten un flujo operativo prácticamente idéntico en sus operaciones básicas de mantenimiento. Por este motivo, y con el fin de evitar una redundancia excesiva en la documentación, para las operaciones CRUD sobre entidades se presentan tablas detalladas únicamente para los **casos de uso generales** (del CU-5 a CU-9). Solo se han detallado casos particulares cuando presentan variaciones significativas, como la creación de pedidos (CU-5.1), que requiere diferenciar entre pedidos de barra y comedor. No obstante, en el diagrama de casos de uso (figura B.1) se han modelado todas las **especializaciones** correspondientes para ofrecer una visión completa del sistema.

### Actores

Los actores representan los roles funcionales básicos que interactúan con el sistema. En este proyecto se han identificado dos actores principales que cubren las necesidades de análisis de casos de uso, aunque la aplicación está diseñada para soportar una granularidad mayor mediante el sistema de **roles y permisos** descrito en los requisitos funcionales (RF-6, RF-7).

Los actores principales identificados son los siguientes:

- **Administrador:** usuario con control total sobre los módulos de la aplicación. Tiene acceso completo a todas las funcionalidades y es responsable de la configuración inicial del sistema, gestión de usuarios, asignación de permisos y supervisión global.

- **Empleado:** usuario registrado que forma parte del personal del local de restauración. Sus capacidades dentro del sistema están determinadas

por los **permisos específicos** que el Administrador le haya asignado, pudiendo abarcar desde un acceso mínimo (solo gestión de su perfil) hasta capacidades amplias en áreas específicas.

Es importante destacar que la aplicación implementa un **sistema de control de acceso basado en roles (*RBAC*)**[1] [10], el cual permite crear configuraciones personalizadas que materialicen roles funcionales más específicos, como por ejemplo *"Cocinero"* o *"Camarero de barra"*. Para simplificar el diagrama B.1, no se modelan como actores separados, pero son completamente realizables mediante el módulo de gestión de roles mencionado. Se profundiza más en esta característica del sistema en el apartado *Seguridad: Modelo RBAC y gestión de acceso*.

### Capacidades por actor

La siguiente tabla resume las capacidades básicas de cada actor, donde se observa la diferencia fundamental entre ambos:

Tabla B.1: Matriz de acciones según actores y módulos

| **Módulo** | **Acción del módulo** | **Empleado** | **Admin.** |
|---|---|---|---|
| Autenticación | Iniciar sesión | ✓ | ✓ |
| | Ver perfil | ✓ | ✓ |
| | Editar perfil | ✓ | ✓ |
| Alimentos | Crear alimento | ? | ✓ |
| | Consultar/Ver listado | ? | ✓ |
| | Editar | ? | ✓ |
| | Eliminar | ? | ✓ |
| Productos | Crear producto | ? | ✓ |
| | Consultar/Ver listado | ? | ✓ |
| | Editar | ? | ✓ |
| | Eliminar | ? | ✓ |
| Mesas | Crear mesa | ? | ✓ |
| | Consultar/Ver listado | ? | ✓ |
| | Editar | ? | ✓ |
| | Eliminar | ? | ✓ |

*continúa en la página siguiente*

[1] *Role-Based Access Control*

*continúa desde la página anterior*

| **Módulo** | **Acción del módulo** | **Empleado** | **Admin.** |
|---|---|---|---|
| Reservas | Crear reserva | ? | ✓ |
| | Consultar/Ver listado | ? | ✓ |
| | Editar | ? | ✓ |
| | Eliminar | ? | ✓ |
| Pedidos | Crear pedido | ? | ✓ |
| | Consultar/Ver listado | ? | ✓ |
| | Editar | ? | ✓ |
| | Eliminar | ? | ✓ |
| Usuarios | Crear usuario | ? | ✓ |
| | Consultar/Ver listado | ? | ✓ |
| | Editar | ? | ✓ |
| | Eliminar | ? | ✓ |
| Roles | Crear rol | ? | ✓ |
| | Consultar/Ver listado | ? | ✓ |
| | Editar | ? | ✓ |
| | Eliminar | ? | ✓ |

Como se ve en la tabla B.1, el administrador posee acceso completo (✓) a todas las funcionalidades del sistema sin restricciones. En cuanto a los empleados, la única funcionalidad garantizada que tienen es la **gestión de su propio perfil** (cambiar información del perfil y contraseña o ver sus datos). El símbolo de interrogación (**?**) en el resto de filas indica que la capacidad de poder realizar dicha acción **depende de los permisos específicos** asignados a ese usuario.

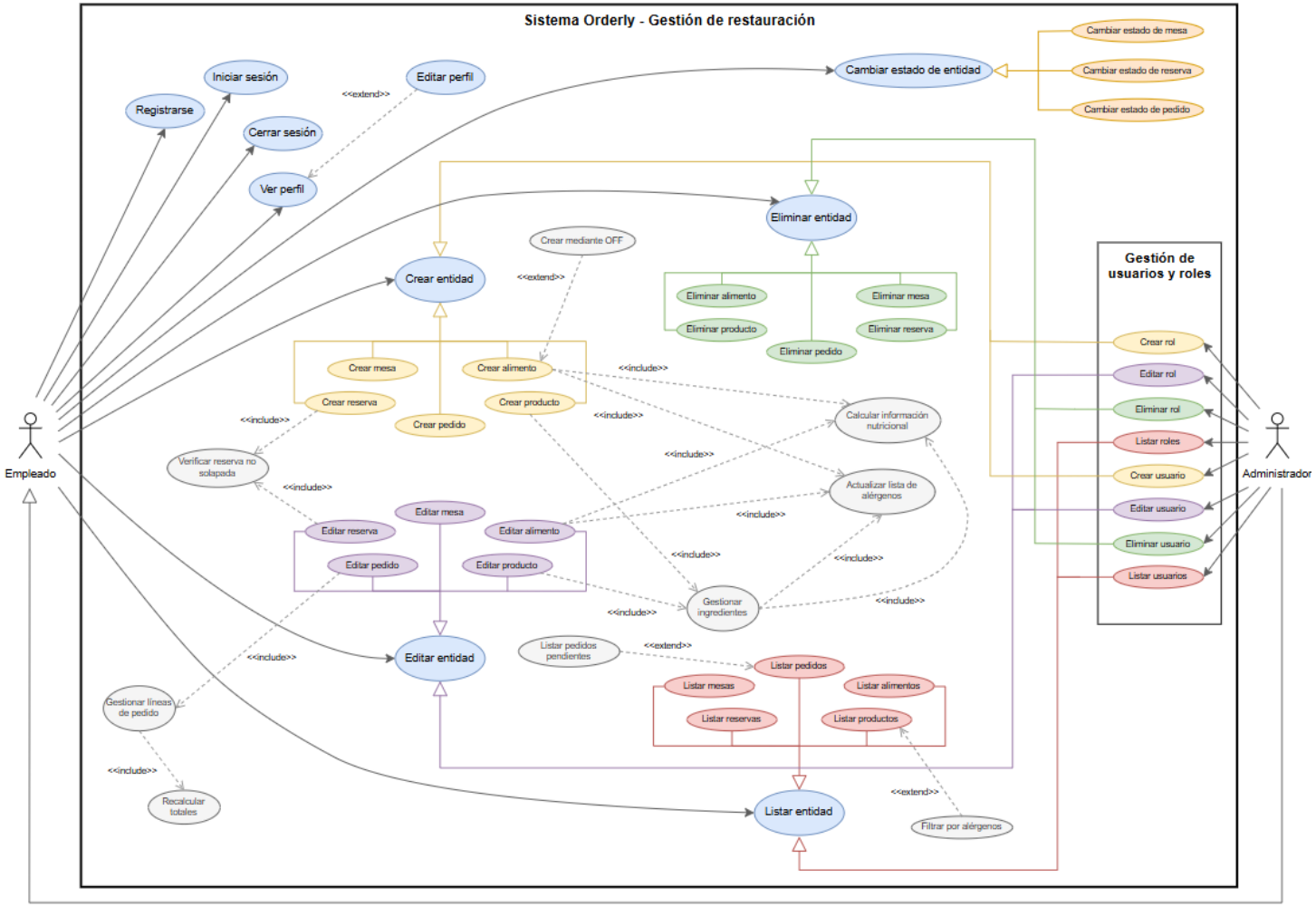


Figura B.1: Diagrama de Casos de Uso

## Casos de uso

Tabla B.2: CU-1 Registrarse (*register*)

| **CU-1** | **Registrarse (*register*)** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-7.1, RF-8 |
| **Actor** | Visitante sin cuenta en el sistema. |
| **Descripción** | Visitante crea una nueva cuenta de usuario. |
| **Precondición** | El usuario no está autenticado. |
| **Acciones** | 1. Hacer *click* en el botón *Registrarse*.<br>2. Rellenar el formulario de registro (como mínimo, el nombre de usuario y la contraseña).<br>3. El sistema valida los campos.<br>*a*) Si la validación es exitosa, el sistema crea el usuario y le añade el rol por defecto. Se inicia sesión automáticamente con el nuevo usuario registrado, redirigiendo a la página de inicio.<br>*b*) Si la validación falla, se muestra un mensaje con la descripción del error. |
| **Postcondición** | Nuevo usuario persistido en la BBDD con el rol por defecto. |
| **Excepciones** | ▪ El *username* ya existe.<br>▪ La confirmación de contraseña no coincide con la contraseña introducida.<br>▪ Alguno de los campos supera la longitud máxima.<br>▪ No se ha rellenado el campo de *username* o el de contraseña. |
| **Importancia** | Alta |

Tabla B.3: CU-2 Iniciar sesión (*login*)

| **CU-2** | **Iniciar sesión (*login*)** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-2** | **Iniciar sesión (*login*)** |
|---|---|
| **Requisitos asociados** | RF-8 |
| **Actor** | Empleado |
| **Descripción** | Usuario se autentica y accede a la aplicación. |
| **Precondición** | ▪ El usuario tiene una cuenta existente.<br>▪ El usuario no está autenticado. |
| **Acciones** | 1. Hacer *click* en el botón *Iniciar sesión*.<br>2. Introducir *username* y contraseña.<br>3. El sistema valida las credenciales.<br>*a*) Si la validación es exitosa, el sistema redirige al usuario a la página de inicio.<br>*b*) Si la validación falla, se muestra un mensaje con la descripción del error. |
| **Postcondición** | ▪ Usuario autenticado y sesión activa.<br>▪ Acceso a vistas según roles. |
| **Excepciones** | Credenciales incorrectas o vacías. |
| **Importancia** | Alta |

Tabla B.4: CU-3 Cerrar sesión (*logout*)

| **CU-3** | **Cerrar sesión (*logout*)** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-8 |
| **Actor** | Empleado |
| **Descripción** | Usuario termina su sesión y sale de la aplicación. |
| **Precondición** | El usuario está autenticado. |
| **Acciones** | 1. Hacer *click* en el botón *Cerrar sesión* de la barra de navegación.<br>2. El sistema invalida la sesión y redirige a la página de inicio. |
| **Postcondición** | Sesión inválida. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-3** | **Cerrar sesión (*logout*)** |
|---|---|
| **Excepciones** | - |
| **Importancia** | Baja |

Tabla B.5: CU-4 Editar perfil propio

| **CU-4** | **Editar perfil propio** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-7.2.1, RF-7.5 |
| **Actor** | Empleado |
| **Descripción** | Usuario edita la información de su perfil. |
| **Precondición** | El usuario está autenticado. |
| **Acciones** | 1. En la vista de perfil del usuario, hacer *click* en el botón *Editar perfil*.<br>2. Modificar campos permitidos.<br>*a*) Solo podrá editar sus roles si tiene los permisos necesarios.<br>*b*) Si se quiere editar la contraseña, se deberá introducir la contraseña actual.<br>3. El sistema valida y persiste los cambios.<br>▪ Si se ha modificado el *username*, se cerrará sesión para que el usuario vuelva a ingresar con las nuevas credenciales. |
| **Postcondición** | Campos del usuario actualizados. |
| **Excepciones** | ▪ El *username* ya existe.<br>▪ La confirmación de contraseña no coincide con la contraseña introducida.<br>▪ La contraseña introducida como "*contraseña actual*" no coincide con la real.<br>▪ Alguno de los campos supera la longitud máxima. |
| **Importancia** | Media |

Tabla B.6: CU-5 Crear entidad

| **CU-5** | **Crear entidad** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-1.1, RF-2.1, RF-3.1, RF-4.1, RF-5.1, RF-6.1, RF-7.1 |
| **Actor** | Empleado con permiso `CREATE` sobre la entidad correspondiente. |
| **Descripción** | Usuario crea una nueva instancia de una entidad del sistema (Alimento, Producto, Mesa, Reserva, Pedido, Usuario o Rol). |
| **Precondición** | ▪ El usuario está autenticado.<br>▪ El usuario tiene permiso para crear la entidad específica. |
| **Acciones** | 1. Hacer *click* en el botón de creación.<br>2. El sistema muestra un formulario modal o página con los campos requeridos.<br>3. Rellenar los campos del formulario.<br>4. Hacer *click* en el botón de guardado.<br>5. El sistema valida todos los campos del formulario.<br>*a*) Si la validación es exitosa, persiste la entidad, cierra el formulario y actualiza la vista.<br>*b*) Si la validación falla, muestra mensajes de error junto a los campos afectados. |
| **Postcondición** | ▪ Nueva entidad persistida en la base de datos.<br>▪ La vista se actualiza para reflejar el nuevo elemento.<br>▪ El usuario recibe confirmación visual de la creación exitosa. |
| **Excepciones** | ▪ Campos obligatorios vacíos.<br>▪ Validación de formato fallida (ej: longitud excedida, número negativo).<br>▪ Violación de unicidad (ej: nombre duplicado).<br>▪ Error de conexión con el servidor.<br>▪ Pérdida de sesión durante la operación. |
| **Importancia** | Alta |

Tabla B.7: CU-5.1 Crear pedido

| **CU-5.1** | **Crear pedido** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-5.1 |
| **Actor** | Empleado con permisos `CREATE` sobre pedidos. |
| **Descripción** | Usuario crea un nuevo pedido de barra o comedor. |
| **Precondición** | ▪ El usuario está autenticado.<br>▪ El usuario tiene el permiso general para crear pedidos.<br>▪ El usuario tiene el permiso para crear pedidos del tipo específico que quiere crear. |
| **Acciones** | 1. Desde el *dashboard* de pedidos, situarse en la vista del tipo de pedido que se quiera crear ("barra" o "comedor").<br>2. Hacer *click* en el botón de creación.<br>3. El sistema muestra un formulario modal con los campos requeridos.<br>*a*) Para los pedidos de **barra**, se podrá elegir si el pedido es solo de bebidas.<br>*b*) Para los pedidos de **comedor**, será necesario asociar una mesa al nuevo pedido.<br>4. Rellenar los campos del formulario.<br>*a*) Si no se especifica número de pedido, se genera uno automáticamente.<br>*b*) Si se especifica, se comprobará que el número introducido es único en el sistema.<br>5. Hacer *click* en el botón de guardado.<br>6. El sistema valida todos los campos del formulario.<br>*a*) Si la validación es exitosa, persiste la entidad, cierra el formulario y actualiza la vista.<br>*b*) Si la validación falla, muestra mensajes de error junto a los campos afectados. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-5.1** | **Crear pedido** |
|---|---|
| **Postcondición** | ▪ Nuevo pedido persistido en la base de datos.<br>▪ La vista se actualiza para reflejar el nuevo pedido.<br>• Se visualiza el nuevo pedido en la tabla de detalles, vacío (sin ítems), con precio total a 0€ y estado `PENDING`.<br>• Se añade el nuevo pedido al final de la lista de pedidos pendientes.<br>▪ El usuario recibe confirmación visual de la creación exitosa. |
| **Excepciones** | ▪ Número de pedido ya existente en el sistema (si se especifica).<br>▪ Mesa no asociada (solo en pedidos de comedor). |
| **Importancia** | Alta |

Tabla B.8: CU-6 Editar entidad

| **CU-6** | **Editar entidad** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-1.2, RF-2.2, RF-3.2, RF-4.2, RF-5.2, RF-6.2, RF-7.2 |
| **Actor** | Usuario con permiso `EDIT` sobre la entidad correspondiente. |
| **Descripción** | Usuario modifica los datos de una entidad existente. |
| **Precondición** | ▪ El usuario está autenticado.<br>▪ El usuario tiene permiso para editar la entidad específica.<br>▪ La entidad a editar existe y es editable.<br>▪ Está visualizando la lista o detalle de la entidad. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-6** | **Editar entidad** |
|---|---|
| **Acciones** | 1. Hacer *click* en el botón *Editar* de la entidad deseada (desde lista o vista detalle).<br>2. El sistema muestra formulario pre-rellenado con los datos actuales.<br>3. Modificar los campos deseados.<br>4. Hacer *click* en el botón de guardado.<br>5. El sistema valida los campos modificados.<br>*a*) Si la validación es exitosa, persiste los cambios y actualiza la vista.<br>*b*) Si falla, muestra mensajes de error sin cerrar el formulario. |
| **Postcondición** | ▪ Datos de la entidad actualizados en la base de datos.<br>▪ Vista actualizada con los nuevos valores.<br>▪ Confirmación visual de la edición exitosa. |
| **Excepciones** | - |
| **Importancia** | Alta |

Tabla B.9: CU-7 Cambiar estado de entidad

| **CU-7** | **Cambiar estado de entidad** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-3.3, RF-4.2, RF-5.2 |
| **Actor** | Empleado con permiso `EDIT` sobre la entidad correspondiente. |
| **Descripción** | Usuario cambia el estado de una entidad existente (Mesa, Reserva o Pedido). |
| **Precondición** | ▪ El usuario está autenticado.<br>▪ El usuario tiene permiso para editar la entidad específica.<br>▪ La entidad a editar existe y es editable.<br>▪ Está visualizando la lista o detalle de la entidad. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-7** | **Cambiar estado de entidad** |
|---|---|
| **Acciones** | 1. Hacer *click* en el botón de estado de la entidad deseada.<br>2. Se desplegará un menú con los estados posibles para esa entidad.<br>3. Seleccionar un estado nuevo.<br>4. El sistema valida la transición de estado.<br>*a*) Si la la transición es válida, persiste los cambios y actualiza la vista.<br>*b*) Si falla, muestra un mensaje de error. |
| **Postcondición** | ▪ Estado de la entidad actualizado en la base de datos.<br>▪ Vista actualizada con el nuevo estado.<br>▪ Confirmación visual de la edición exitosa.<br>▪ **Solo pedidos**: en caso de haber cancelado o completado el pedido, dicho pedido desaparecerá de la lista de pedidos pendientes del *dashboard* y solo podrá visualizarse desde el historial. |
| **Excepciones** | ▪ La transición del estado anterior al nuevo no está permitida. Por ejemplo, intentar cambiar el estado de una reserva completada o cancelada. |
| **Importancia** | Baja |

Tabla B.10: CU-8 Eliminar entidad

| **CU-8** | **Eliminar entidad** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-1.3, RF-2.3, RF-3.3, RF-4.3, RF-5.3, RF-6.3, RF-7.3 |
| **Actor** | Usuario con permiso `DELETE` sobre la entidad correspondiente. |
| **Descripción** | Usuario elimina una entidad del sistema, aplicando borrado lógico o físico según la política de la entidad. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-8** | **Eliminar entidad** |
|---|---|
| **Precondición** | ▪ El usuario está autenticado.<br>▪ El usuario tiene permiso para eliminar la entidad específica.<br>▪ La entidad a eliminar existe.<br>▪ Está visualizando la lista o detalle de la entidad. |
| **Acciones** | 1. Hacer *click* en el botón *Eliminar* de la entidad deseada.<br>2. El sistema muestra diálogo de confirmación con:<br>▪ Nombre/identificador de la entidad.<br>▪ Advertencia sobre consecuencias (dependiendo de entidad).<br>3. Confirmar eliminación. |
| **Postcondición** | ▪ Vista actualizada (elemento desaparece).<br>▪ Confirmación visual de la eliminación exitosa. |
| **Excepciones** | ▪ Entidad tiene dependencias que impiden eliminación (ej: alimento usado en producto). |
| **Importancia** | Media |

Tabla B.11: CU-9 Listar entidades

| **CU-9** | **Listar entidades** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-1.5, RF-2.7, RF-3.5, RF-4.4, RF-5.7, RF-6.4, RF-7.6 |
| **Actor** | Usuario con permiso `VIEW_LIST` sobre la entidad correspondiente. |
| **Descripción** | Usuario visualiza una lista paginada de todas las instancias de una entidad. |
| **Precondición** | ▪ El usuario está autenticado.<br>▪ El usuario tiene permiso para listar la entidad específica. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-9** | **Listar entidades** |
|---|---|
| **Acciones** | 1. Navegar a la sección de la entidad deseada en el menú principal o barra de navegación.<br>2. El sistema carga y muestra la lista de elementos. |
| **Postcondición** | ▪ Vista actualizada con la lista de elementos. |
| **Excepciones** | ▪ No hay elementos para mostrar (lista vacía).<br>▪ Error al cargar datos del servidor. |
| **Importancia** | Alta |

Tabla B.12: CU-10 Gestionar ingredientes de un producto

| **CU-10** | **Gestionar ingredientes de un producto** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-2.4.1, RF-2.4.2, RF-2.4.3, RF-2.5 |
| **Actor** | Empleado con permisos `PRODUCT_EDIT_INGREDIENTS` y `PRODUCT_CREATE` o `PRODUCT_EDIT`. |
| **Descripción** | Usuario agrega, quita o modifica cantidades de ingredientes que conforman un producto. |
| **Precondición** | ▪ Usuario autenticado con permiso para crear o editar productos.<br>▪ El usuario tiene permiso para editar los ingredientes de un producto.<br>▪ Está en el formulario de creación o edición de producto. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-10** | **Gestionar ingredientes de un producto** |
|---|---|
| **Acciones** | 1. Hacer *click* en la sección *Ingredientes* dentro del formulario. Desde esa sección pueden llevarse a cabo tres operaciones:<br>*a*) **Añadir ingrediente**: en la barra de búsqueda, introducir el nombre de un alimento que quiera añadirse como ingrediente y pulsar en *Añadir*.<br>*b*) **Editar porciones**: pulsar en el botón para editar cantidades e introducir una cantidad en gramos.<br>*c*) **Quitar ingrediente**: en la tabla de ingredientes, pulsar el botón de borrar del alimento que quiera eliminarse de la lista. |
| **Postcondición** | ▪ Lista de ingredientes del producto actualizada.<br>▪ Información nutricional del producto recalculada automáticamente, en base a los ingredientes que lo forman y su cantidad en gramos dentro del producto.<br>▪ Lista de alérgenos del producto actualizada, incluyendo los alérgenos particulares de todos sus ingredientes. |
| **Excepciones** | - |
| **Importancia** | Alta |

Tabla B.13: CU-11 Gestionar líneas de pedido

| **CU-11** | **Gestionar líneas de pedido** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-5.4.1, RF-5.4.2, RF-5.4.3, RF-5.5 |
| **Actor** | Empleado con permisos `EDIT` sobre pedidos y `PRODUCT_VIEW_LIST`. |
| **Descripción** | Usuario agrega, quita o modifica cantidades de ítems (productos) que pertenecen a un pedido. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-11** | **Gestionar líneas de pedido** |
|---|---|
| **Precondición** | ▪ Usuario autenticado con permiso para editar pedidos.<br>▪ El usuario tiene permiso para visualizar la lista de productos.<br>▪ Está en el *dashboard* (panel) de pedidos. |
| **Acciones** | 1. Seleccionar el pedido a editar desde la lista de pedidos pendientes (sección izquierda).<br>2. En la lista de productos ofertados (sección derecha), buscar el producto a añadir y hacer *click* sobre él.<br>3. Se añadirá ese producto al pedido con una cantidad de 1. Una vez añadido, pueden llevarse a cabo las siguientes acciones desde la tabla de detalles del pedido:<br>*a*) **Editar cantidad**: introducir la cantidad unitaria del producto en su línea correspondiente (ej: *"3 raciones de Estofado de ternera"*) y pulsar el botón de *Guardar*.<br>*b*) **Eliminar línea de pedido**: pulsar el botón de borrar de la línea que quiera eliminarse de la lista. |
| **Postcondición** | ▪ Lista de ítems del pedido actualizada.<br>▪ Subtotales (por línea de pedido) y precio total del pedido recalculados automáticamente, en función a los precios de los productos que lo forman y sus cantidades unitarias. |
| **Excepciones** | ▪ Si se editan las cantidades de los productos pero no se pulsa el botón de *Guardar*, no se persisten ni se actualiza el importe total.<br>• Cuando se recargue la página o se seleccione otro pedido, se perderán. |
| **Importancia** | Alta |

Tabla B.14: CU-12 Gestionar roles de un usuario

| **CU-12** | **Gestionar roles de un usuario** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-7.4.1, RF-7.4.2 |
| **Actor** | Administrador |
| **Descripción** | Usuario asigna o quita roles a otro usuario. |
| **Precondición** | ▪ Usuario autenticado con permiso para editar los roles de un usuario.<br>▪ Está en el formulario de creación o edición de un usuario. |
| **Acciones** | 1. Hacer *click* en la sección *Gestión de roles* dentro del formulario.<br>2. Se visualizarán 2 columnas: *"Roles asignados"* y *"Roles disponibles"*.<br>3. Hacer *click* en las *pills* que contienen el nombre de cada rol, para alternarlos entre columnas.<br>4. Hacer *click* en el botón de guardado. |
| **Postcondición** | ▪ Roles del usuario actualizados.<br>▪ El usuario editado sigue teniendo el rol por defecto `ROLE_USER`, incluso si en el formulario se le ha intentado retirar. |
| **Excepciones** | - |
| **Importancia** | Alta |

Tabla B.15: CU-13 Buscar y crear alimento mediante *Open Food Facts*

| **CU-13** | **Buscar y crear alimento mediante *Open Food Facts*** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-1.1.2, RF-9 |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-13** | **Buscar y crear alimento mediante *Open Food Facts*** |
|---|---|
| **Actor** | Empleado con permiso `FOOD_CREATE`. |
| **Descripción** | Usuario busca e importa información nutricional de alimentos desde la API pública *Open Food Facts*. |
| **Precondición** | ▪ Usuario autenticado con permiso para crear alimentos.<br>▪ Está en el formulario de creación de alimento.<br>▪ Conexión a Internet activa. |
| **Acciones** | 1. Hacer *click* en la sección *Buscar en Open Food Facts* dentro del formulario.<br>2. Introducir término de búsqueda (ej: *"Arroz basmati"*).<br>3. El sistema consulta la API externa y muestra resultados.<br>4. Seleccionar un resultado de la lista.<br>5. El sistema importa automáticamente:<br>  ▪ Nombre del alimento.<br>  ▪ Información nutricional (calorías, proteínas, grasas, carbohidratos, etc.).<br>  ▪ Lista de alérgenos detectados.<br>  ▪ Métricas nutricionales (*Nutri-Score*, grupo NOVA). |
| **Postcondición** | ▪ Alimento persistido en el sistema, con campos pre-rellenados mediante datos externos.<br>▪ Alérgenos automáticamente asociados al alimento. |
| **Excepciones** | ▪ Ya existe un alimento con ese nombre en el sistema (alimento ya importado).<br>▪ API externa no disponible o con *timeout*.<br>▪ No se encuentran resultados para la búsqueda.<br>▪ Datos incompletos o mal estructurados en la respuesta.<br>▪ Pérdida de conexión durante la búsqueda. |
| **Importancia** | Media |

Tabla B.16: CU-14 Visualizar información nutricional y alérgenos

| **CU-14** | **Visualizar información nutricional y alérgenos** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-2.6, RF-10, RF-11 |
| **Actor** | Empleado con permiso `PRODUCT_VIEW_LIST`. |
| **Descripción** | Usuario consulta la información nutricional detallada y los alérgenos de un producto, presentada en formatos tabulares y gráficos. |
| **Precondición** | ▪ Usuario autenticado.<br>▪ Está en la vista de listado de productos.<br>▪ Producto/alimento objetivo existe.<br>▪ **Para gráficas:** El producto tiene información nutricional calculada. |
| **Acciones** | 1. Hacer *click* en el nombre del producto.<br>2. El sistema redirigirá a vista de detalle.<br>3. En la sección nutricional, visualizar:<br>*a*) Gráfico (*donut chart*) de macronutrientes.<br>*b*) Tablas para macronutrientes, vitaminas y minerales.<br>*c*) Lista de alérgenos detectados con iconos visuales.<br>*d*) Lista de ingredientes del producto. Para cada ingrediente se muestra:<br>▪ Nombre del alimento.<br>▪ Porción en gramos dentro del producto.<br>▪ Lista de alérgenos particulares.<br>▪ Métricas nutricionales (*Nutri-Score*, grupo NOVA) con etiquetas visuales. |
| **Postcondición** | ▪ Información nutricional y de alérgenos presentada claramente.<br>▪ Gráficas renderizadas y actualizadas. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-14** | **Visualizar información nutricional y alérgenos** |
|---|---|
| **Excepciones** | ▪ Información nutricional incompleta o no calculada.<br>▪ Producto sin ingredientes definidos (valores nutricionales nulos).<br>▪ Error al renderizar gráficas (datos mal formados). |
| **Particularidades** | ▪ **Productos:** Valores calculados a partir de ingredientes.<br>▪ **Alimentos:** Valores directos o importados de API.<br>▪ **Alérgenos:** Herencia automática de ingredientes a productos. |
| **Importancia** | Media |

Tabla B.17: CU-15 Filtrar productos excluyendo alérgenos

| **CU-15** | **Filtrar productos excluyendo alérgenos** |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Amanda Pérez Olmos |
| **Requisitos asociados** | RF-2.7, RF-11 |
| **Actor** | Empleado con permiso `PRODUCT_VIEW_LIST`. |
| **Descripción** | Usuario selecciona uno o varios alérgenos para excluir de la lista de productos ofertados en el *dashboard* de pedidos. |
| **Precondición** | ▪ Usuario autenticado.<br>▪ Está en el *dashboard* (panel) de pedidos. |

*continúa en la página siguiente*

*continúa desde la página anterior*

| **CU-15** | **Filtrar productos excluyendo alérgenos** |
|---|---|
| **Acciones** | 1. En la sección derecha (productos ofertados), hacer *click* en el botón *Filtrar*.<br>2. Se abrirá un *pop-up* con una barra de búsqueda y una lista de los alérgenos que pueden excluirse.<br>▪ Se puede buscar el alérgeno por nombre o hacer *scroll* hasta encontrarlo.<br>3. Seleccionar uno o varios alérgenos para excluir y pulsar el botón *Filtrar* (el interno del *pop-up*). |
| **Postcondición** | ▪ Se muestran los productos que no contienen ninguno de los alérgenos seleccionados.<br>▪ En el botón *Filtrar* aparece el número de alérgenos que se están excluyendo. |
| **Excepciones** | ▪ Si se hace *click* fuera del *pop-up* sin pulsar el botón *Filtrar*, no se llevará a cabo la operación de filtrado. |
| **Importancia** | Baja |

*Apéndice* $C$

# Especificación de diseño

## C.1. Introducción

Una vez consolidado el marco de requisitos, este apéndice establece el puente entre la fase de análisis y la implementación real de la plataforma de restauración. El objetivo es proyectar una solución técnica coherente que garantice la integridad de la información, la estabilidad de la infraestructura y la correcta ejecución de los procesos del sistema. Esta etapa resulta fundamental para asegurar que la construcción del *software* responda de manera eficiente a los objetivos planteados, proporcionando una estructura organizada que facilite tanto su desarrollo como su escalabilidad futura.

## C.2. Diseño de datos

En esta sección se describe la estructura lógica de la información que sustenta la plataforma de restauración. Se procede a describir las entidades involucradas y las relaciones entre ellas.

Los diagramas que se mostrarán a continuación han sido generados mediante la herramienta *"Reverse Engineer"* de *MySQL Workbench*. Cabe señalar que esta funcionalidad **no soporta** la representación de **relaciones IS-A**, presentes en la jerarquía de herencia de pedidos (figuras C.1 y C.5). Adicionalmente, las representa con cardinalidad 1:N, lo cual no se corresponde con la cardinalidad real de las herencias con estrategia *JOINED*, pero de nuevo, la herramienta no sabe diferenciar entre 1:1 y 1:N [13].

## Modelo general

A continuación, se procede a listar las **entidades** que conforman la aplicación y a describir brevemente su propósito. La figura C.1 muestra el esquema completo de la base de datos. Las **relaciones** entre entidades y detalles particulares se describirán en el apartado *Modelos detallados*.

- **Usuario (`User`):** se trata de un empleado que utiliza la aplicación. Dispone de credenciales para ingresar en ella y tiene roles que determinan su comportamiento dentro del sistema.
- **Rol (`Role`):** agrupa un conjunto de permisos (`Permission`). Controla las acciones y el acceso a los recursos que un usuario puede tener en la aplicación.
  - El enumerado `Permission` define todas las acciones que pueden ser autorizadas en el sistema. Los permisos se organizan por áreas funcionales (pedidos, mesas, reservas, etc.).
- **Alimento (`Food`):** representa un componente o ingrediente base (p. ej. *"Lechuga"*, *"Pollo"*), con una información nutricional asociada.
- **Producto (`Product`):** se trata de un plato de comida o producto ofertado (p. ej. *"Ensalada César"*) en el local, el cual se compone de varios alimentos y tiene un precio asociado.
- **Ingrediente (`Ingredient`):** representa la relación *muchos a muchos* entre `Food` y `Product`. Contiene un atributo que determina la cantidad (en gramos) de un alimento que está presente en un producto determinado.
- **Mesa (`DiningTable`):** representa una mesa física del restaurante, que puede reservarse y desde la que se puede realizar un pedido.
- **Reserva (`Reservation`):** contiene información sobre la reserva de una mesa de comedor, los datos del huésped que la realizó y la fecha, entre otros.
- **Pedido (`Order`):** se trata de una comanda que se le hace a un empleado, esto es, uno o varios productos que se piden para ser consumidos y que suman un precio total. Se trata de una entidad "padre" que alberga los campos comunes entre tipos de pedidos. Se verá con mayor detalle en la sección *Modelo para gestión de pedidos*.

- **Pedido de bar (`BarOrder`):** es un pedido que se realiza en la barra del bar y puede estar compuesto solo por bebidas (por lo que no sería necesario prepararlo en la cocina).

- **Pedido de comedor (`DiningOrder`):** es una comanda que se realiza desde el comedor, por lo que tendrá una mesa asociada a la que llevar los platos.

- **Línea de pedido (`OrderItem`):** representa un ítem de cada pedido, compuesto por un producto y una cantidad de dicho producto. Constituye un porcentaje del precio total del pedido.

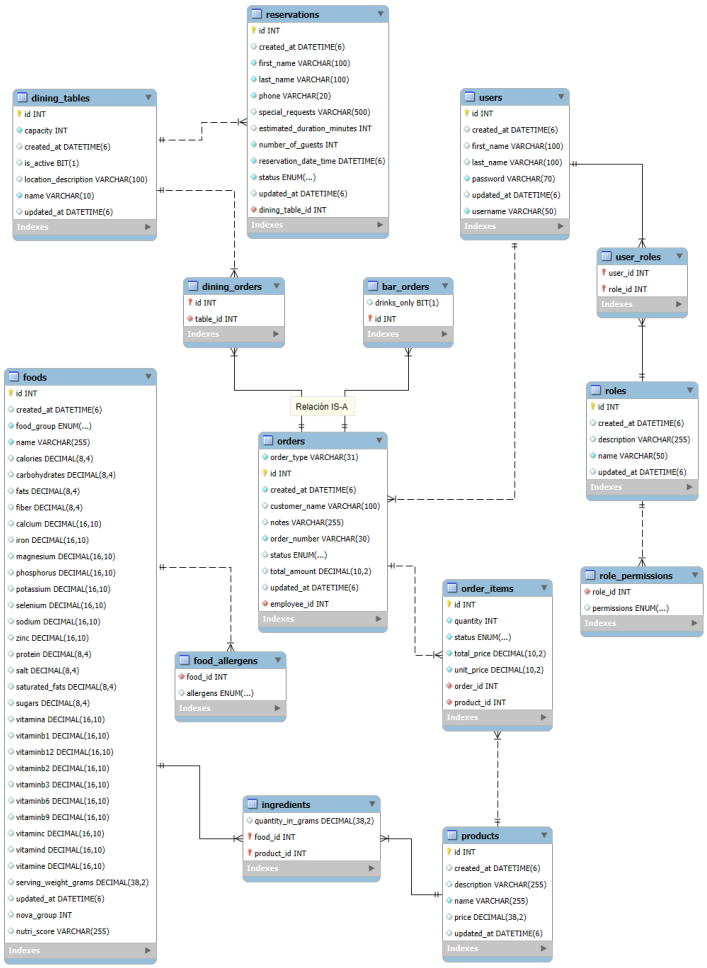


Figura C.1: Esquema de la Base de Datos

Cabe destacar que todas las entidades disponen de campos de auditoría (`createdAt` y `updatedAt`) para rastrear automáticamente la creación y última modificación de cada registro. Adicionalmente, la presencia de **relaciones bidireccionales** entre entidades hace necesaria la creación de métodos que aseguren la **consistencia a ambos lados** de la relación, puesto que JPA no lo hace automáticamente. Este aspecto se verá con más detalle en el apartado *C.4 Diseño procedimental*.

## Modelos detallados

Para facilitar la comprensión del modelo de datos, se presentan distintos diagramas relacionales que agrupan los módulos funcionales según su interdependencia.

### Modelo para gestión de usuarios y roles

Este módulo, cuyo esquema se detalla en la figura C.2, controla el acceso al sistema y los permisos del personal del restaurante.

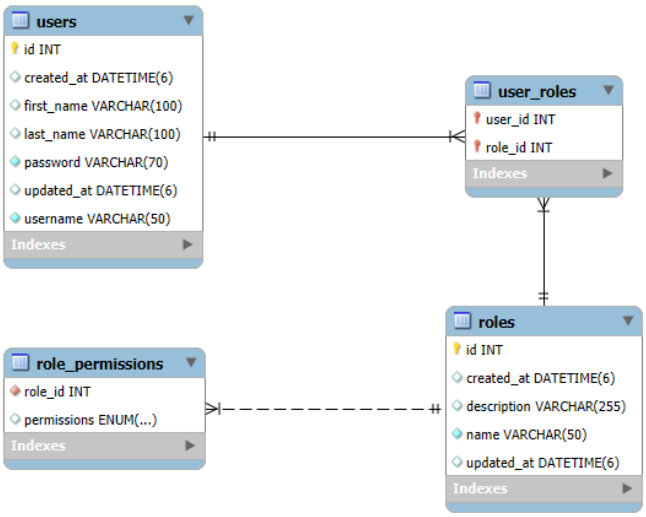


Figura C.2: Diagrama relacional para módulo de usuarios y roles

Las relaciones involucradas son:

- `User` ↔ `Role`: relación `@ManyToMany` (N:M) bidireccional, lo que implica que un usuario puede tener varios roles y que un rol puede pertenecer a varios usuarios. Esta relación se implementa mediante la

tabla `user_roles`, la cual **no es una entidad JPA**, puesto que solo almacena claves foráneas y no tiene lógica de negocio.

- `Role` → `Permission`: la lista de permisos de un rol es una colección de elementos (`@ElementCollection`) del enumerado `Permission` (1:N), lo que se traduce en una tabla separada `role_permissions`. Esta tabla **no es una entidad JPA** porque solo sirve para almacenar relaciones simples sin atributos extra.

### Modelo para gestión de alimentos y productos

Este módulo, cuyo esquema se detalla en la figura C.3, gestiona el catálogo de productos del restaurante y su composición en ingredientes, permitiendo calcular información nutricional automáticamente.

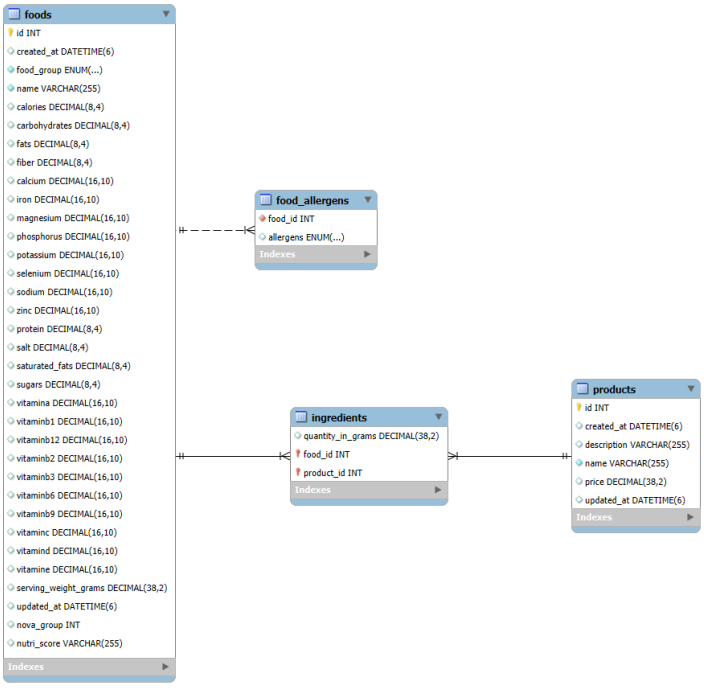


Figura C.3: Diagrama relacional para módulo de alimentos y productos

Las relaciones involucradas son:

- `Product` ↔ `Ingredient` ↔ `Food`: relación `@ManyToMany` (N:M) implementada como dos relaciones (1:N) bidireccionales a través de la entidad intermedia `Ingredient`, la cual registra la cantidad en gramos.

- La entidad `Ingredient` posee una **clave compuesta y derivada** (`@EmbeddedId`), la cual se forma mediante las claves de `Food` y `Product`.

- `Food` → `Allergen`: la lista de alérgenos de un alimento es una colección de elementos (`@ElementCollection`) del enumerado `Allergen` (1:N), lo que se traduce en una tabla separada `food_allergens`. Esta tabla **no es una entidad JPA** porque solo sirve para almacenar relaciones simples sin atributos extra.

Respecto a la **información nutricional** que almacena un alimento, cabe destacar el uso de **objetos embebidos** (`@Embeddable`) para almacenar los datos en una sola tabla `foods`, pero manteniendo una separación entre clases: `NutritionInfo`, `Minerals` y `Vitamins`. Esto permite la reutilización, encapsulación y cálculo de operaciones matemáticas particulares. Se explica con más detalle en los *pull requests* #51 y #52 del repositorio del proyecto.

### Modelo para gestión de reservas de mesas

Este módulo, cuyo esquema se detalla en la figura C.4, permite gestionar las mesas del comedor y asignar reservas para fechas futuras.

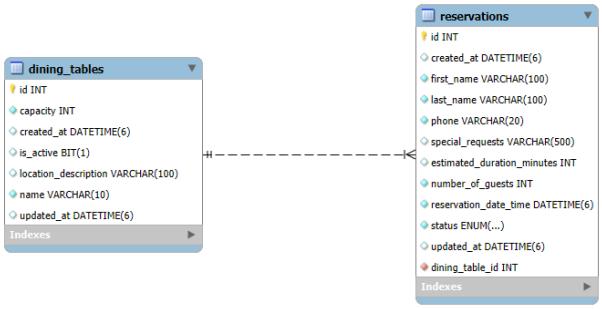


Figura C.4: Diagrama relacional para módulo de reservas de mesas

Las relaciones involucradas son:

- `Reservation` → `DiningTable`: relación `@ManyToOne` (N:1), la cual define que una reserva debe llevar una mesa asociada y que una mesa puede tener múltiples reservas en distintas fechas. La **unidireccionalidad** desde `Reservation` hacia `DiningTable` implica que una mesa **no conoce** su historial de reservas, puesto que sobrecargaría demasiado la entidad.

En este módulo también hay presencia de **clases embebidas** para almacenar detalles de una reserva `ReservationDetails`, (como el número de comensales, fecha y hora) e información del cliente `GuestInfo` (como el nombre y teléfono de contacto) en distintas clases, manteniendo la tabla unificada `reservations`.

### Modelo para gestión de pedidos

En este módulo, cuyo esquema se detalla en la figura C.5, se anotan las comandas realizadas al restaurante, que pueden hacerse desde la barra del bar o desde el comedor. Una comanda se compone de una serie de productos, cada uno con su cantidad y precio de venta asociados, que se manejan en conjunto y tienen un precio total.

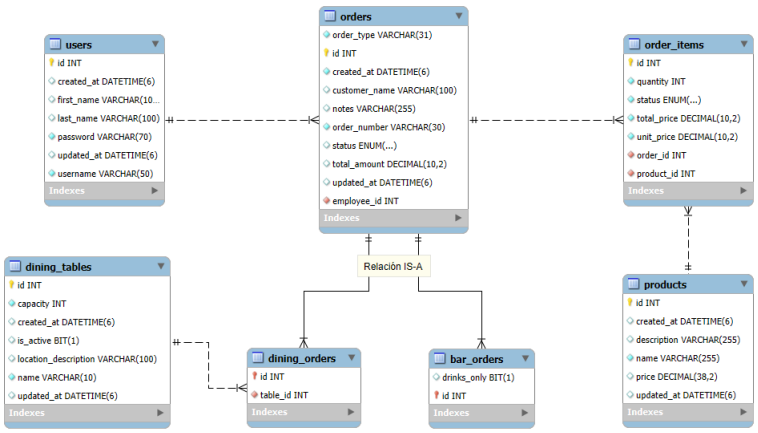


Figura C.5: Diagrama relacional para módulo de pedidos

Las relaciones involucradas son:

- `Order` → `User`: relación `@ManyToOne` (N:1) unidireccional, puesto que un pedido será creado o manejado por un empleado. La **unidireccionalidad** desde `Order` hacia `User` implica que los usuarios **no conocerán** (al menos, en primera instancia o sin necesidad de consultas adicionales) los pedidos asociados a ellos.
- `Order` → `OrderItem`: relación `@OneToMany` (1:N), puesto que un pedido contendrá múltiples ítems. En *Spring Boot* se definió como bidireccional, para que los ítems pudieran notificar al pedido padre que recalculase su precio total. Sin embargo, en la práctica funcionará como **unidireccional**, puesto que los ítems no consultarán ni gestionarán en ningún momento el pedido al que pertenecen.

- `OrderItem` → `Product`: relación `@ManyToOne` (N:1) unidireccional, debido a que cada ítem referencia a una cantidad determinada de un producto ofertado en el catálogo. La **unidireccionalidad** desde `OrderItem` hacia `Product` implica que los productos **no conocerán** (al menos, en primera instancia o sin necesidad de consultas adicionales) los pedidos a los que han sido añadidos.

- `DiningOrder` → `DiningTable`: relación `@ManyToOne` (N:1), puesto que un pedido de comedor debe estar asociado a la mesa donde se servirá. Una mesa tendrá un historial de pedidos pasados, pero debido a que se trata de una relación **unidireccional**, no conocerá dicho historial (esto sobrecargaría demasiado la entidad).

El aspecto más característico de este módulo es la presencia de una **jerarquía de herencia**, para manejar los distintos tipos de pedidos (bar o comedor) desde un enfoque organizado y escalable. El uso de herencia permite **atributos específicos** por tipo, mientras comparte **datos comunes** eficientemente. La estrategia de herencia utilizada es ***JOINED*** y los motivos de su elección, así como una descripción más detallada de los tipos de pedido, se pueden encontrar en el *pull request* #80.

## C.3. Diseño arquitectónico

En esta sección se define la estructura global del sistema, la organización de sus componentes y las reglas que rigen la comunicación entre ellos. Para este proyecto, se ha optado por una arquitectura moderna que prioriza la modularidad, alta cohesión y bajo acoplamiento. La estructura busca garantizar la escalabilidad del sistema y facilitar el mantenimiento independiente de sus componentes fundamentales.

### Arquitectura de la aplicación

El sistema implementa un modelo de arquitectura **cliente-servidor desacoplada**. En este esquema, el *frontend* (*React*) y el *backend* (*Spring Boot*) se constituyen como sistemas independientes que interactúan exclusivamente a través de una interfaz de programación de aplicaciones (**API**) de tipo **REST** (*Representational State Transfer*).

La infraestructura se apoya en la ***contenerización*** mediante *Docker*, donde cada servicio se despliega en un entorno aislado. El intercambio de información se realiza sobre el protocolo HTTP, empleando **JSON** como

formato de serialización de datos. Esta separación física y lógica permite que ambos extremos evolucionen tecnológicamente de forma autónoma, facilitando tareas de despliegue continuo y escalabilidad horizontal.

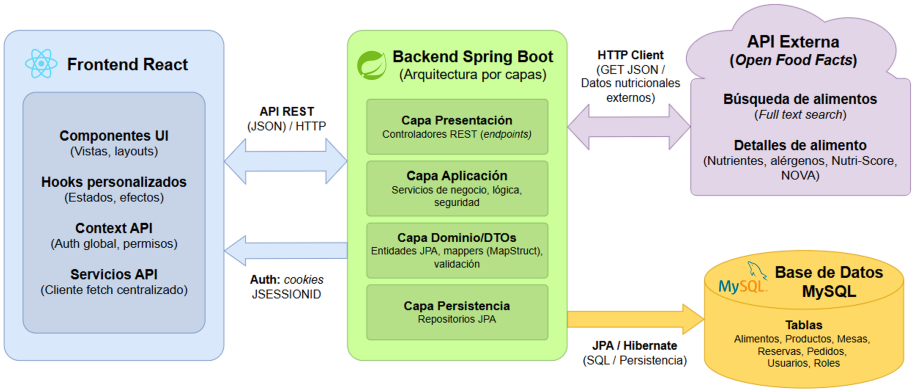


Figura C.6: Arquitectura general de *Orderly*

La figura C.6 muestra un diagrama que ilustra la arquitectura general de la aplicación. A continuación se explicarán con más detalle los componentes principales que constituyen cada servicio.

### Arquitectura del *backend*

El *backend*, desarrollado bajo el ecosistema de *Spring Boot*, sigue un patrón de **arquitectura por capas** (*Layered Architecture*). Esta organización jerárquica permite una clara separación de responsabilidades, facilitando la testabilidad y la reutilización de componentes. Las capas se definen de la siguiente manera:

1. **Capa de Presentación (Controladores):** implementada mediante controladores REST, se encarga de la exposición de recursos (*endpoints* a los que el cliente puede llamar) y la gestión de los contratos de la API. Su responsabilidad se limita a la interceptación de peticiones, la delegación del trabajo a los servicios y la generación de respuestas HTTP adecuadas.

2. **Capa de Aplicación (Servicios):** constituye el núcleo de la lógica de negocio del sistema. Los servicios actúan como orquestadores de las operaciones, gestionando la transaccionalidad, las reglas de dominio y la integración con otros servicios.

3. **Capa de Dominio (Entidades):** define el modelo de persistencia del sistema mediante entidades JPA (*Java Persistence API*). Estas clases representan el esquema de datos y las relaciones de negocio en la base de datos *MySQL*, como ya se ha detallado en el apartado *C.2 Diseño de datos*.

4. **Capa de Persistencia (Repositorios):** utiliza *Spring Data JPA* [16] para abstraer el acceso a los datos. Los repositorios facilitan la realización de operaciones CRUD y consultas complejas, de forma desacoplada del motor de base de datos específico.

### Arquitectura del *frontend*

El *frontend* se ha diseñado bajo el paradigma de **arquitectura basada en componentes** de *React*, priorizando el principio de separación de intereses (***Separation of Concerns*** - *SoC*). La lógica se distribuye para evitar componentes monolíticos y facilitar la mantenibilidad:

- **Componentes de interfaz (UI):** son las piezas visuales (botones, formularios, tablas). Su responsabilidad se limita a mostrar la información y capturar las interacciones del usuario, delegando cualquier lógica compleja a otros elementos.

- ***Hooks* personalizados:** se utilizan para encapsular la lógica de estado (estados de carga, errores) y los efectos secundarios ocasionados por las llamadas asíncronas. Esta abstracción permite reutilizar comportamientos complejos a lo largo de toda la aplicación y ofrece un punto **centralizado** de comunicación con los servicios específicos de cada módulo.

- **Servicios de API:** actúan como una capa de abstracción sobre el cliente de transporte (`fetch`). **Centralizan** las llamadas a la API de *Spring Boot*, de modo que cualquier cambio en los *endpoints* o en la estructura de las peticiones solo requiera modificaciones en un único punto del código.

- **Gestión de estado global:** para el manejo de datos transversales a la aplicación, como la sesión del usuario o sus permisos asociados, se utilizan contextos que gestionan el estado global (***Context API*** de *React* [1, 11]). Esto evita el fenómeno de *prop drilling* (paso excesivo de propiedades entre niveles), permitiendo que cualquier componente acceda al contexto de autenticación (`AuthContext`) de forma eficiente.

## Seguridad: Modelo RBAC y gestión de acceso

La seguridad del sistema se ha diseñado bajo un modelo de **control de acceso basado en roles** (*RBAC*) [10] de **granularidad fina**. A diferencia de modelos más simples donde el rol determina directamente el acceso, en esta arquitectura los roles actúan como **contenedores** lógicos de permisos individuales. Estos permisos, definidos en el enumerado `Permission`, representan acciones atómicas sobre las entidades del dominio (ej. `FOOD_CREATE`, `ORDER_EDIT`, `USER_VIEW_LIST`).

Esta aproximación permite una gestión altamente flexible y adaptable a la estructura organizativa de un local de restauración. Dado que un usuario puede tener asignados **múltiples roles** simultáneamente, el sistema evalúa de forma aditiva el conjunto total de permisos, permitiendo el acceso si al menos uno de los roles del usuario posee el permiso requerido. Esto facilita la creación de **perfiles profesionales** específicos, como por ejemplo:

- **Camarero:** con permisos para gestionar pedidos y consultar reservas.
- **Cocinero:** con permisos para gestionar alimentos y visualizar pedidos pendientes de cocina.

Algunos ejemplos de configuraciones de permisos que materializan puestos de trabajo comunes se encuentran en el apartado *Ejemplos de creación de roles*.

### Implementación en el *backend* y propagación al *frontend*

En la capa de aplicación, la seguridad se articula mediante *Spring Security*. Se ha implementado un servicio personalizado (`CustomUserDetailsService`) que carga el perfil del usuario y transforma tanto los roles como sus permisos asociados en una colección de `GrantedAuthority`[1]. La protección de los recursos se realiza de forma declarativa mediante la anotación `@PreAuthorize` en los métodos de los **controladores**, verificando que el usuario autenticado posea la autoridad específica necesaria antes de ejecutar la lógica de negocio.

[1] Las autoridades (*"authorities"*) son la unidad de medida que *Spring Security* utiliza para entender y validar los permisos de un usuario.

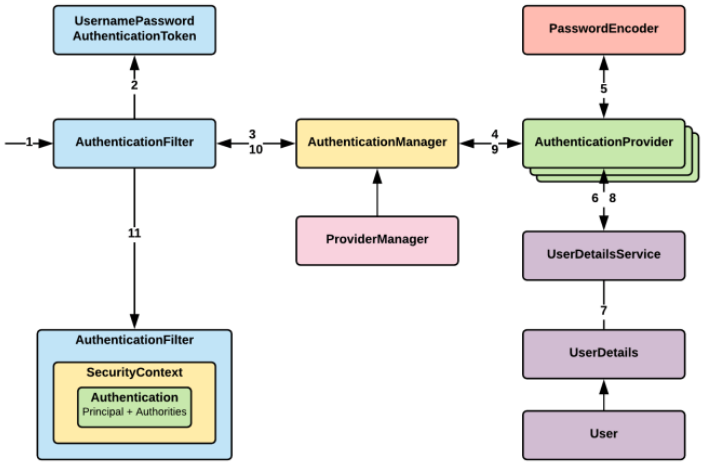


Figura C.7: Flujo de autenticación en *Spring Security* [14]

Para garantizar una experiencia de usuario coherente, el estado de autenticación y el catálogo de permisos se propagan al *frontend*. El `AuthContext` de *React* actúa como el núcleo de decisión en el cliente, permitiendo el **renderizado condicional de componentes** de la interfaz de usuario. De este modo, elementos críticos como botones de edición o opciones de administración **se ocultan preventivamente** si el usuario no cuenta con la autorización pertinente, mejorando la usabilidad y reforzando la seguridad de la aplicación.

### Gestión de Sesiones y autenticación

El sistema emplea una estrategia de autenticación basada en sesión HTTP, gestionada de forma transparente por *Spring Security*. Tras una validación exitosa de credenciales, el **servidor establece una sesión** y emite una *cookie* `JSESSIONID`. Esta *cookie*, configurada con los atributos de seguridad adecuados, es gestionada automáticamente por el navegador en las peticiones subsiguientes. El *frontend* detecta este estado de sesión para mantener la **persistencia del usuario entre recargas** de página, permitiendo un flujo de trabajo ininterrumpido.

## Desacoplamiento de integraciones con APIs nutricionales externas

Una cuestión a la que se le ha dado gran importancia es el **aislamiento** de dependencias externas. La aplicación se integra con la API de *Open Food*

*Facts* para el enriquecimiento de datos nutricionales, pero esta **integración se gestiona exclusivamente en el *backend***, permaneciendo totalmente opaca para el *frontend*, como se puede visualizar en la figura C.6.

El diseño se apoya en un módulo independiente que encapsula la lógica de comunicación, mapeo y transformación de datos externos (directorio `/foods/external`). Mediante el uso de la clase `OpenFoodFactsRegistry` (figura C.8), se establece una fuente única de verdad (***Single Source of Truth***) para las **equivalencias** entre los códigos de la API externa y los tipos internos de la aplicación (como alérgenos o grupos de alimentos). Este desacoplamiento ofrece dos **ventajas** críticas:

- **Mantenibilidad:** cualquier cambio en la estructura o en los puntos de acceso de la API externa solo requiere modificaciones en el módulo dedicado del *backend*.
- **Extensibilidad:** facilita la **sustitución** o adición de **nuevos proveedores** de datos nutricionales con un impacto mínimo en el resto del sistema, ya que el **contrato** de datos que recibe el *frontend* permanece **inalterado**. Esta cuestión se explora más a fondo en el apartado *Integrar una nueva API nutricional*.

```java
1  package es.ubu.inf.tfg.food.external.openFoodFacts.dto.mapper;
2
3  import es.ubu.inf.tfg.food.classification.type.Allergen;
4
5  import lombok.experimental.UtilityClass;
6
7  import java.util.Map;
8
9  @UtilityClass
10 public class OpenFoodFactsRegistry {
11
12     public final Map<String, Allergen> ALLERGEN_CODES = Map.ofEntries(
13         Map.entry(k: "en:gluten", Allergen.GLUTEN),
14         Map.entry(k: "en:crustaceans", Allergen.CRUSTACEANS),
15         Map.entry(k: "en:eggs", Allergen.EGGS),
16         Map.entry(k: "en:fish", Allergen.FISH),
17         Map.entry(k: "en:peanuts", Allergen.PEANUTS),
18         Map.entry(k: "en:soybeans", Allergen.SOYBEANS),
19         Map.entry(k: "en:milk", Allergen.MILK),
20         Map.entry(k: "en:nuts", Allergen.NUTS),
21         Map.entry(k: "en:celery", Allergen.CELERY),
22         Map.entry(k: "en:mustard", Allergen.MUSTARD),
23         Map.entry(k: "en:sesame-seeds", Allergen.SESAME_SEEDS),
24         Map.entry(k: "en:sulphur-dioxide-and-sulphites", Allergen.SULPHUR_DIOXIDE_SULPHITES),
25         Map.entry(k: "en:lupin", Allergen.LUPIN),
26         Map.entry(k: "en:molluscs", Allergen.MOLLUSCS)
27     );
28
```

```java
29     @UtilityClass
30     public class Nutriments {
31         public static final String CALORIES = "energy-kcal_100g";
32         public static final String CARBS = "carbohydrates_100g";
33         public static final String FAT = "fat_100g";
34         public static final String FIBER = "fiber_100g";
35         public static final String PROTEIN = "proteins_100g";
36         public static final String SALT = "salt_100g";
37         public static final String SATURATED_FAT = "saturated-fat_100g";
38         public static final String SUGARS = "sugars_100g";
39
40         // Minerals
41         public static final String CALCIUM = "calcium_100g";
42         public static final String IRON = "iron_100g";
43         public static final String MAGNESIUM = "magnesium_100g";
44         public static final String PHOSPHORUS = "phosphorus_100g";
45         public static final String POTASSIUM = "potassium_100g";
46         public static final String SELENIUM = "selenium_100g";
47         public static final String SODIUM = "sodium_100g";
48         public static final String ZINC = "zinc_100g";
49
50         // Vitamins
51         public static final String VIT_A = "vitamin-a_100g";
52         public static final String VIT_B1 = "vitamin-b1_100g";
53         public static final String VIT_B2 = "vitamin-b2_100g";
54         public static final String VIT_B3 = "vitamin-pp_100g";
55         public static final String VIT_B6 = "vitamin-b6_100g";
56         public static final String VIT_B9 = "vitamin-b9_100g";
57         public static final String VIT_B12 = "vitamin-b12_100g";
58         public static final String VIT_C = "vitamin-c_100g";
59         public static final String VIT_D = "vitamin-d_100g";
60         public static final String VIT_E = "vitamin-e_100g";
61     }
62 }
```

Figura C.8: Clase `OpenFoodFactsRegistry` con equivalencias entre campos de la API *Open Food Facts* y el modelo de nuestra aplicación

## Infraestructura y despliegue

La infraestructura de la aplicación se gestiona mediante *Docker*, empleando ***Docker Compose*** para la orquestación de los servicios. El sistema se

descompone en **tres contenedores independientes**: el entorno de ejecución para el *frontend* (*React/Vite*), el servidor de aplicaciones (*Spring Boot*) y el motor de base de datos (*MySQL*).

Durante la fase de **desarrollo**, la infraestructura se configura para maximizar la productividad del programador. Como se observa en el diagrama C.9, se emplean **volúmenes de datos** específicos en el servicio *frontend* para habilitar el *Hot Module Replacement* (HMR).

Esta configuración permite que los cambios en el código fuente se reflejen instantáneamente en el navegador sin necesidad de reconstruir las imágenes. Los detalles técnicos sobre el funcionamiento de estos volúmenes y su relación con el flujo de trabajo se detallan en el apartado *Frontend: React + Vite con HMR*.

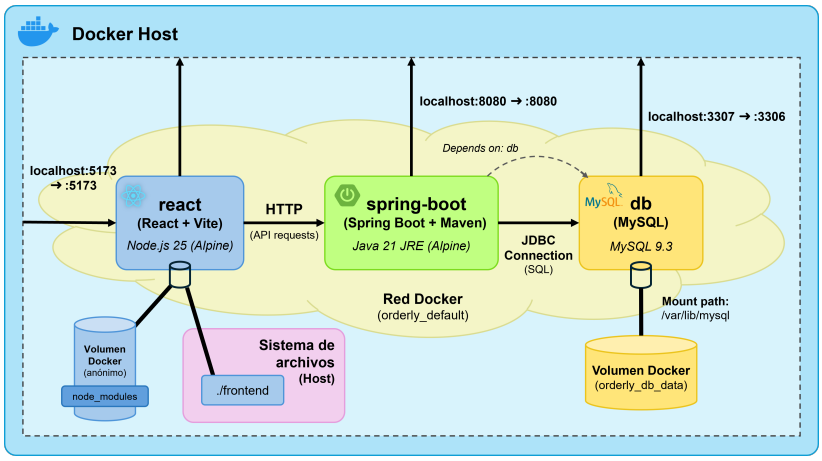


Figura C.9: Diagrama de despliegue en entorno de desarrollo

Para el entorno de **producción**, la arquitectura se simplificaría para garantizar la portabilidad y el rendimiento. En este escenario (diagrama C.10):

- **Eliminación de dependencias locales:** se prescindiría de los volúmenes del *frontend*. El código fuente y las dependencias quedarían embebidos estáticamente dentro de la imagen del contenedor, eliminando la dependencia con el sistema de archivos del anfitrión.
- **Persistencia de datos:** se mantendría el volumen persistente `orderly_db_data` para el servicio *MySQL*. Esto asegura que la información de la base de datos sea independiente del ciclo de vida del contenedor.

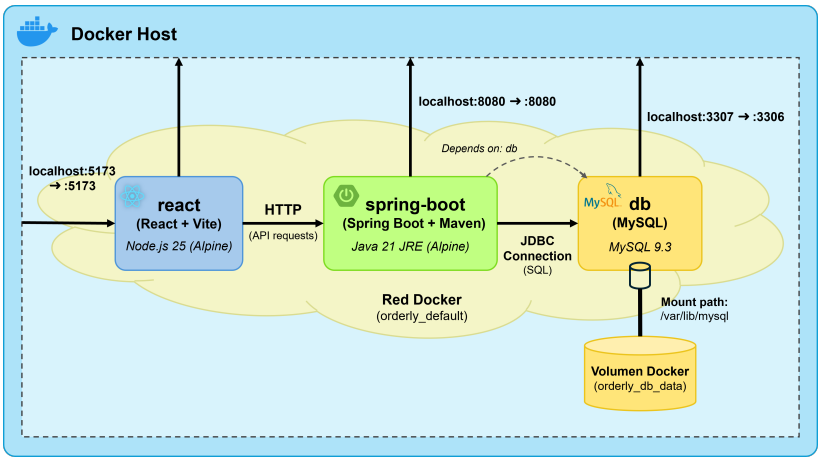


Figura C.10: Diagrama de despliegue en entorno de producción

# C.4. Diseño procedimental

Esta sección trata de la transformación de los modelos arquitectónicos en especificaciones técnicas detalladas, definiendo la lógica interna y el comportamiento de los componentes del sistema. El diseño procedimental permite establecer la secuencia exacta de pasos y el flujo de control necesarios para resolver cada tarea, asegurando que la transición hacia la implementación sea coherente.

## Patrones de diseño utilizados

Para resolver problemas recurrentes del *software* y facilitar el mantenimiento, se han implementado los siguientes patrones:

- **Patrón *Builder*:** se ha integrado de forma extensiva mediante la anotación `@Builder` de *Lombok*, tanto en entidades como en DTOs. Este patrón facilita la instanciación de objetos complejos (como `Food` o `Product`), mejorando significativamente la legibilidad del código y evitando constructores con largas listas de parámetros que pueden inducir a error.

- **Patrón *Factory* (`OrderMapperFactory`):** dado que el sistema gestiona distintos tipos de pedidos (`BarOrder` y `DiningOrder`) con requerimientos de datos específicos, se ha implementado una factoría de mapeo (figura C.11). La clase `OrderMapperFactory` actúa como

un *"dispatcher"* que decide en tiempo de ejecución qué estrategia de mapeo (*mapper*) aplicar basándose en el tipo de DTO o entidad.

- Podría considerarse una combinación del patrón *factory* y el patrón ***strategy***.
- **Ventaja:** desacopla el servicio `OrderService` de las implementaciones concretas de cada pedido.

- **Patrón *Adapter*:** se utiliza en la clase `CustomUserDetails`. Esta clase actúa como un adaptador que permite que nuestra entidad de dominio `User` cumpla con el contrato de la interfaz `UserDetails` requerida por *Spring Security*. Gracias a esto, el sistema de seguridad puede interactuar con el modelo de usuarios propio sin necesidad de modificar la estructura de la base de datos.

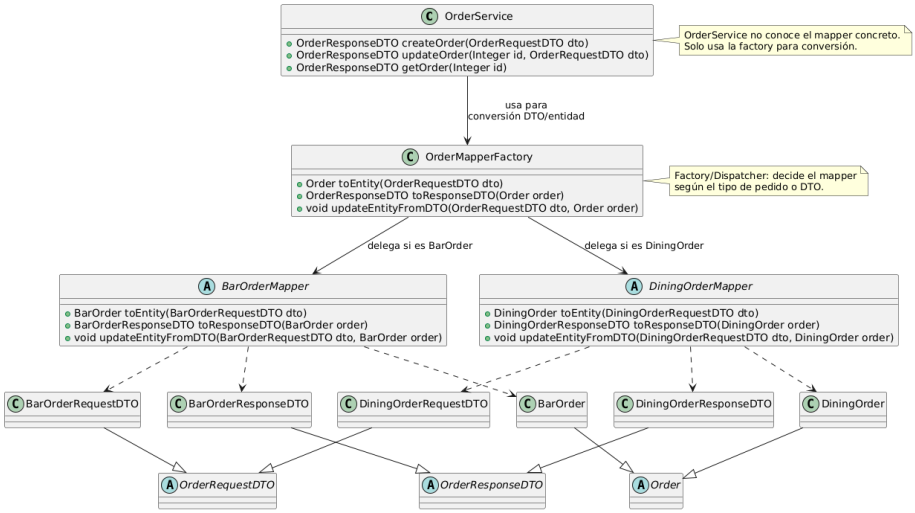


Figura C.11: Implementación del patrón *Factory* para la gestión polimórfica de pedidos y sus DTOs

## Gestión de consistencia y lógica de dominio

A diferencia de las arquitecturas con modelos anémicos, donde las entidades son meros contenedores de datos, en este proyecto se ha optado por un *Rich Domain Model* (**Modelo de Dominio Enriquecido**).

**Lógica de dominio encapsulada** Las entidades asumen la responsabilidad de sus propios cálculos y estados (figura C.12). Por ejemplo, la entidad `Product` es capaz de calcular automáticamente su perfil nutricional y alérgenos agregando la información de sus ingredientes. Del mismo modo, la entidad `Order` gestiona sus líneas de pedido y actualiza su importe total de forma autónoma.

```
25  public class Product {
77      public AllergenInfo calculateTotalAllergens() {
78          AllergenInfo total = AllergenInfo.builder().build();
79          for (Ingredient ingredient : ingredients) {
80              if (ingredient.getFood() != null && ingredient.getFood().getAllergenInfo() != null) {
81                  total = total.addAll(ingredient.getFood().getAllergenInfo());
82              }
83          }
84          return total;
85      }
86
87      public NutritionInfo calculateTotalNutrition() {
88          NutritionInfo total = NutritionInfo.builder().build();
89          for (Ingredient ingredient : ingredients) {
90              NutritionInfo ingredientNutrition = ingredient.calculateNutritionInfo();
91              total = total.add(ingredientNutrition);
92          }
93          return total;
94      }
95  }
```

```
30  public abstract class Order {
117     public void calculateTotalAmount() {
118         if (items != null) {
119             this.totalAmount = items.stream()
120                     .map(OrderItem::getTotalPrice)
121                     .filter(Objects::nonNull)
122                     .reduce(BigDecimal.ZERO, BigDecimal::add);
123         }
124     }
```

```
27  public class Food {
82      public NutritionInfo getNutritionInfoForQuantity(BigDecimal quantityInGrams) {
83
84          if ((quantityInGrams == null || quantityInGrams.compareTo(BigDecimal.ZERO) <= 0) ||
85              (this.servingWeightGrams == null || this.servingWeightGrams.compareTo(BigDecimal.ZERO) <= 0)) {
86              return NutritionInfo.builder().build();
87          }
88          BigDecimal factor = quantityInGrams.divide(this.servingWeightGrams);
89          return this.nutritionInfo.multiplyByFactor(factor);
90      }
91  }
```

Figura C.12: Ejemplo de métodos de dominio en entidades `Product`, `Food` y `Order`

**Consistencia bidireccional** Para mantener la integridad referencial en memoria, se han implementado **métodos auxiliares** de sincronización en las relaciones bidireccionales. Por ejemplo, en la relación entre `Order` e `OrderItem`, los métodos `addItem` y `removeItem` aseguran que, al vincular un ítem a un pedido, el ítem también "conozca" a qué pedido pertenece, evitando estados inconsistentes antes de la persistencia (figura C.13).

```java
19  public class Ingredient {
59      public void setProduct(Product product) {
60          if (this.product != null) {
61              this.product.getIngredients().remove(this);
62          }
63          this.product = product;
64          if (product != null && !product.getIngredients().contains(this)) {
65              product.getIngredients().add(this);
66          }
67      }
68
69      public void setFood(Food food) {
70          if (this.food != null) {
71              this.food.getUsages().remove(this);
72          }
73          this.food = food;
74          if (food != null && !food.getUsages().contains(this)) {
75              food.getUsages().add(this);
76          }
77      }
78
79      public void remove() {
80          if (this.product != null) {
81              this.product.getIngredients().remove(this);
82              this.product = null;
83          }
84          if (this.food != null) {
85              this.food.getUsages().remove(this);
86              this.food = null;
87          }
88      }
```

```java
22  public class OrderItem {
72      public void setQuantity(Integer newQuantity) {
73          if (newQuantity != null && newQuantity > 0) {
74              this.quantity = newQuantity;
75              calculateTotalPrice();
76
77              // Notify parent order to recalculate the total amount
78              if (order != null) {
79                  order.calculateTotalAmount();
80              }
81          }
82      }
83
84      public void setOrder(Order order) {
85          if (this.order != null) {
86              this.order.getItems().remove(this);
87          }
88          this.order = order;
89          if (order != null && !order.getItems().contains(this)) {
90              order.getItems().add(this);
91          }
92      }
```

```java
30  public abstract class Order {
95      public void addItem(OrderItem item) {
96          if (item != null && !items.contains(item)) {
97              items.add(item);
98              item.setOrder(this);
99              calculateTotalAmount();
100         }
101     }
102
103     public void removeItem(OrderItem item) {
104         if (item != null && items.remove(item)) {
105             item.setOrder(order: null);
106             calculateTotalAmount();
107         }
108     }
109
110     public void clearItems() {
111         List<OrderItem> itemsToRemove = new ArrayList<>(items);
112         for (OrderItem item : itemsToRemove) {
113             removeItem(item);
114         }
```

Figura C.13: Ejemplos de métodos para consistencia bidireccional en entidades `Ingredient`, `Order` y `OrderItem`

## Flujos principales de la aplicación

A continuación se detalla el funcionamiento de las principales operaciones realizadas en la aplicación, junto a los componentes responsables de llevarlas a cabo.

En la mayoría de los casos, los diagramas de secuencia (creados mediante la herramienta *PlantUML*) presentados serán de **alto nivel**, esto es, señalarán al servicio (*React*, *Spring Boot* o *MySQL*) responsable de realizar la acción. Sin embargo, para casos en los que resulte útil conocer la implementación interna dentro de un servicio, se detallarán también las interacciones de **bajo nivel** entre componentes de dicho servicio, como es el caso de las figuras C.15 y C.17.

### A) Iniciar sesión

1. Empleado introduce su nombre de usuario y contraseña en el formulario de inicio de sesión de la aplicación web.

2. *Frontend React* recibe los datos del formulario y realiza una petición `HTTP POST` al *endpoint* `/api/auth/login` del *backend Spring Boot*.

3. *API Spring Boot* recibe la petición y procesa las credenciales:

   *a*) Si las credenciales son correctas:

      1) El *backend* autentica al usuario y crea una sesión.
      2) Devuelve una respuesta con los datos del usuario y la *cookie* `JSESSIONID` (identificador de sesión) en la cabecera.

   *b*) Si las credenciales son incorrectas:

      1) El *backend* devuelve un error indicando que las credenciales no son válidas.

4. *Frontend React* recibe la respuesta:

   *a*) Si el inicio de sesión es exitoso, almacena los datos del usuario en el contexto y la *cookie* de sesión se guarda automáticamente en el navegador.

   *b*) Si hay error, muestra un mensaje de error al usuario.

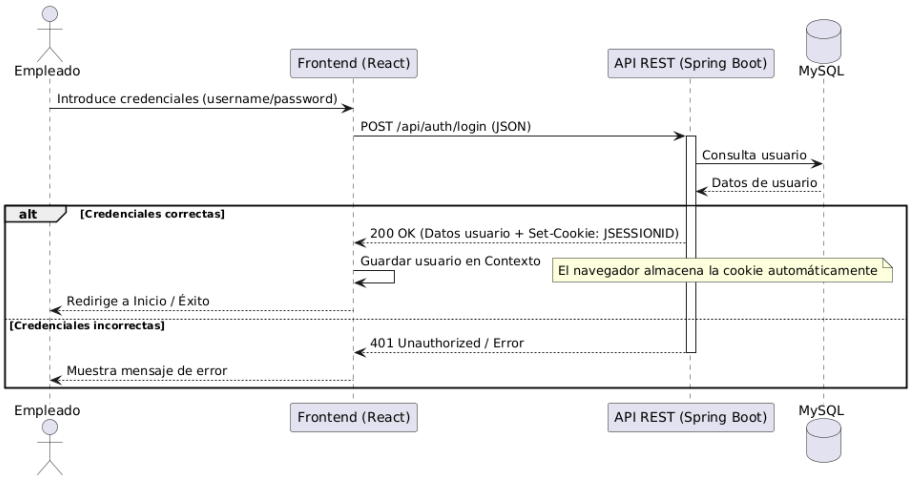


Figura C.14: Diagrama de secuencia del inicio de sesión general

### A.1) Iniciar sesión (detalles de *Spring Boot*)

1. El controlador `AuthController` recibe la petición `POST /api/auth/login` con las credenciales.
2. El controlador llama al método `login` de `AuthService`, pasando el DTO con las credenciales y los objetos `HttpServletRequest` y `HttpServletResponse`.
3. `AuthService` utiliza el `AuthenticationManager` de *Spring Security* para autenticar al usuario:
   *a*) Crea un objeto `UsernamePasswordAuthenticationToken` con el usuario y contraseña.
   *b*) Llama a `authenticationManager.authenticate(...)`.
4. El `AuthenticationManager` delega la autenticación a la implementación de `UserDetailsService` (en este caso, `CustomUserDetailsService`):
   *a*) Busca el usuario en la base de datos mediante `UserRepository`.
   *b*) Si el usuario existe, construye un objeto `CustomUserDetails` con los datos del usuario.
   *c*) Verifica que la contraseña proporcionada coincida con la almacenada (usando el `PasswordEncoder`).
5. Si la autenticación es exitosa:
   *a*) Se establece el contexto de seguridad en `SecurityContextHolder`.
   *b*) Se guarda el contexto de seguridad en la sesión HTTP mediante `securityContextRepository.saveContext(...)`, lo que genera la *cookie* `JSESSIONID`.
   *c*) Se devuelve el DTO del usuario al controlador.
6. Si la autenticación falla:
   *a*) Se lanza una excepción (`InvalidCredentialsException`) y se devuelve un error al *frontend*.
7. El controlador responde al *frontend* con los datos del usuario (o el error).

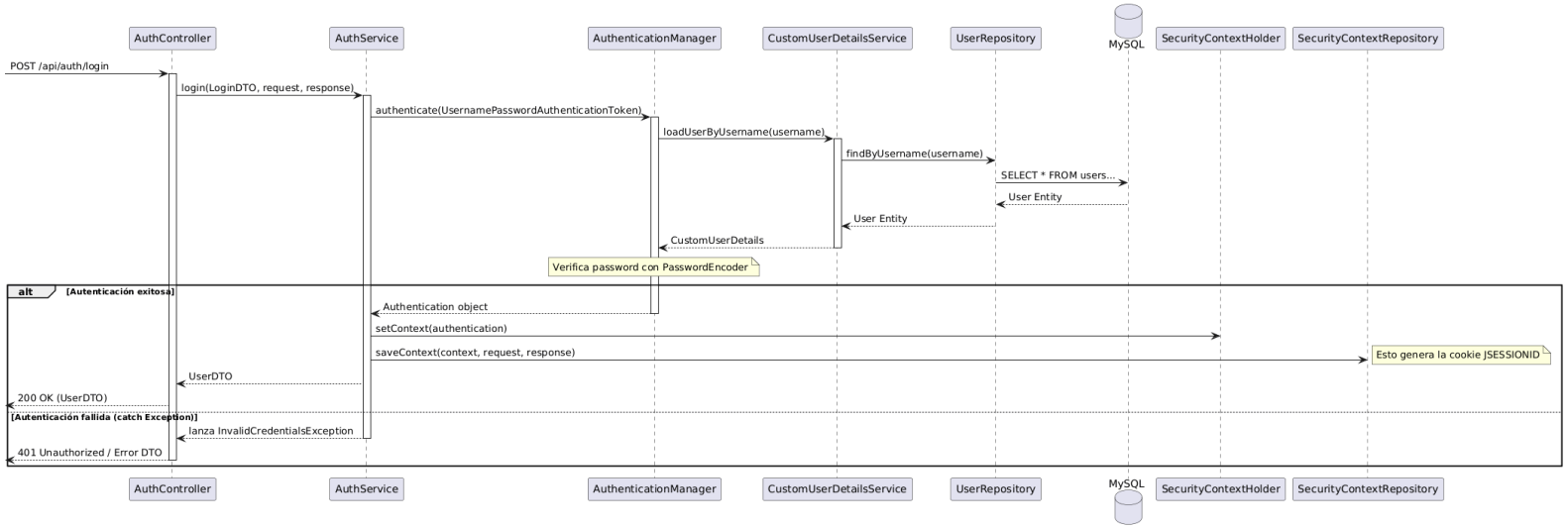


Figura C.15: Diagrama de secuencia del inicio de sesión en *Spring Boot*

### B) Crear una entidad

1. Empleado accede al formulario de creación en la aplicación *React* y lo rellena.
2. *Frontend React* valida los campos localmente (formulario):
   *a*) Si los datos no son válidos, muestra errores y no permite enviar el formulario.
3. (Opcional) *Frontend React* realiza una comprobación de unicidad llamando al *endpoint* de verificación en *Spring Boot*:
   *a*) Si el valor no es único, muestra error en el formulario.
4. Empleado envía el formulario con los datos.
5. *Frontend React* realiza una petición `HTTP POST` al *endpoint* de creación correspondiente en *Spring Boot* (por ejemplo, `/api/users` o `/api/tables`).
6. *API Spring Boot* recibe la petición y valida los datos del DTO:
   *a*) Si los datos no son válidos, responde con error de validación (400).
   *b*) Si el usuario no tiene permisos, responde con error de autorización (403).
   *c*) Si hay conflicto de unicidad, responde con error de conflicto (409).
7. Si todo es correcto, *Spring Boot* crea la entidad y la persiste en la base de datos *MySQL*.
8. *Spring Boot* responde al *frontend React* con los datos de la entidad creada.
9. *Frontend React* muestra mensaje de éxito o error según la respuesta.

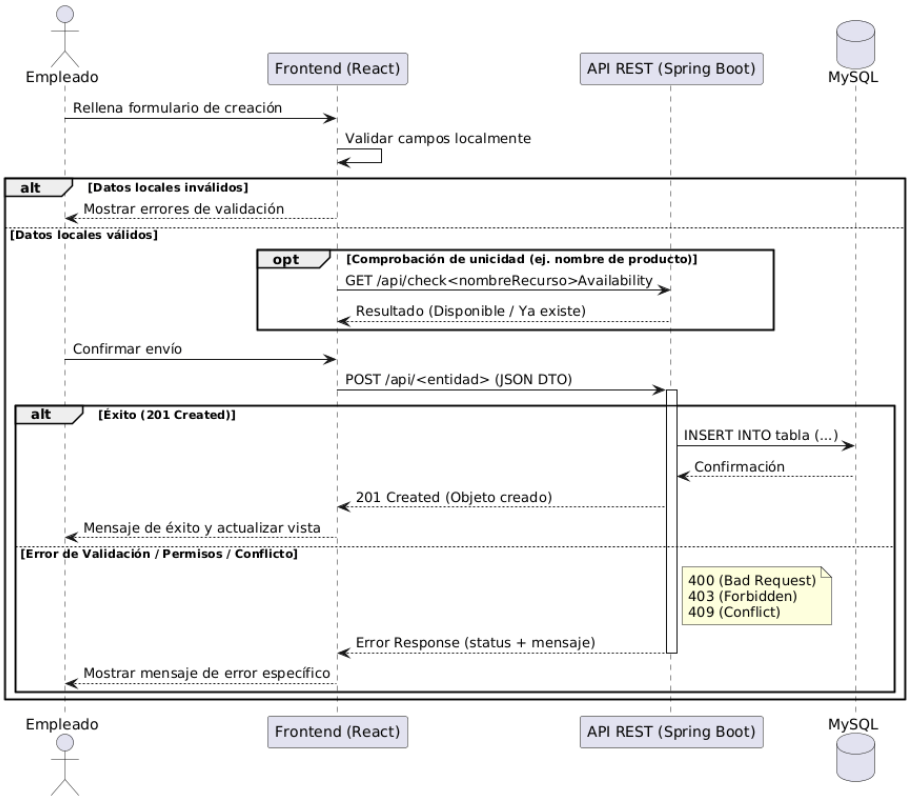


Figura C.16: Diagrama de secuencia de la creación de una entidad

## B.1) Crear un pedido (detalles de *Spring Boot*)

1. El controlador (`BarOrderController`, o `DiningOrderController`) recibe la petición `POST` con el DTO de pedido.
2. El controlador verifica los permisos del usuario (anotaciones `@PreAuthorize`):
   *a*) Si el usuario no tiene permisos, lanza excepción de acceso denegado (403).
3. El controlador llama al método `create` de `OrderService`, pasando el DTO.
4. `OrderService` delega la transformación del DTO en entidad a `OrderMapperFactory`:
   *a*) `OrderMapperFactory` selecciona el *mapper* adecuado según el tipo de pedido (`BarOrderMapper` o `DiningOrderMapper`).
   *b*) El *mapper* transforma el DTO en la entidad correspondiente (`BarOrder` o `DiningOrder`).

5. `OrderService` asocia los ítems al pedido y calcula el importe total.
6. `OrderService` persiste la entidad en la base de datos mediante el repositorio (`OrderRepository`).
7. `OrderService` transforma la entidad persistida en el DTO de respuesta usando `OrderMapperFactory`.
8. El controlador responde al *frontend* con el DTO de respuesta.
9. Si ocurre algún error (validación, permisos, unicidad, etc.), se lanza la excepción correspondiente y el controlador la gestiona mediante el `GlobalExceptionHandler`.

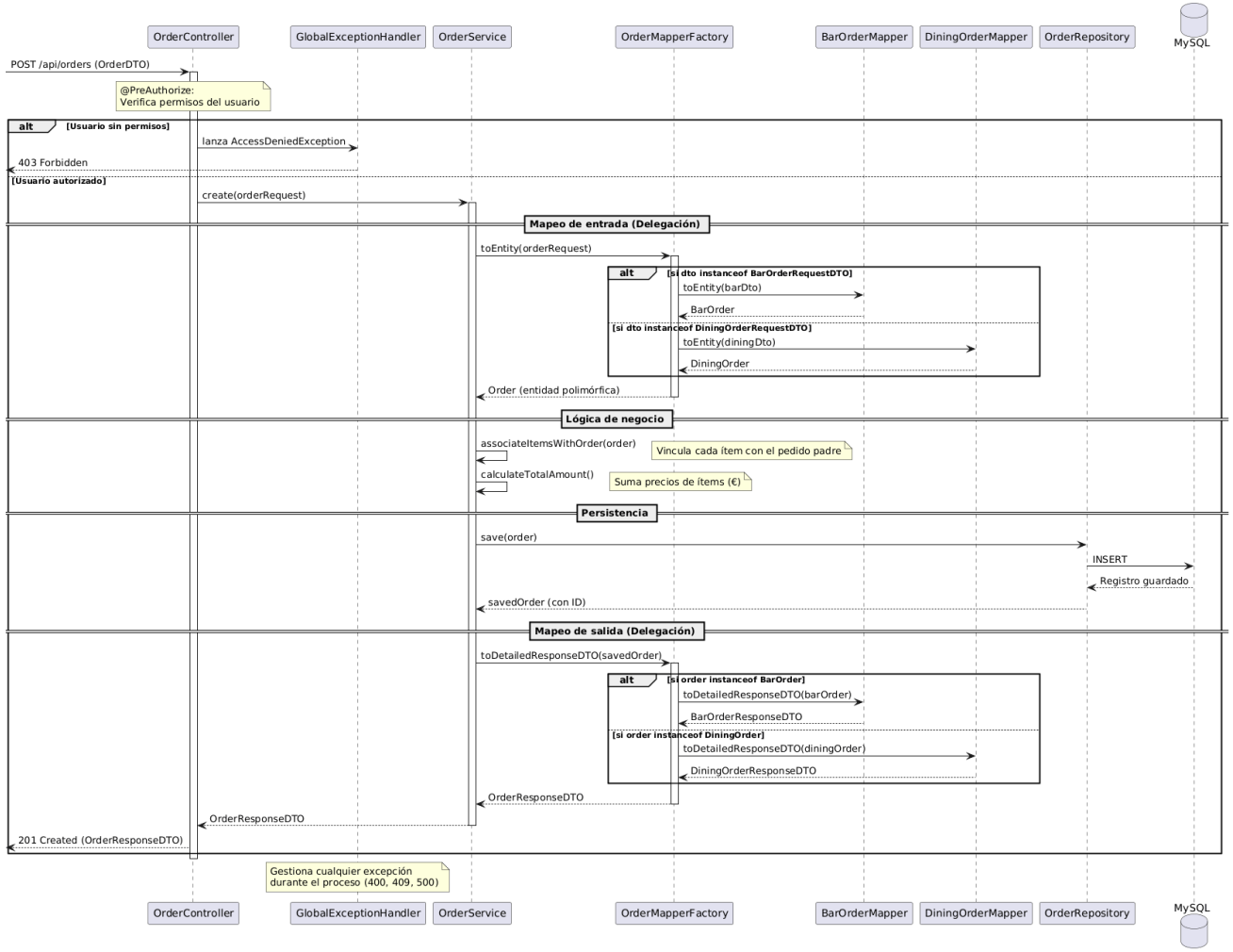


Figura C.17: Diagrama de secuencia de la creación de un pedido en *Spring Boot*

### C) Añadir ingredientes a un producto

1. Empleado accede a la pantalla de edición/creación de producto en *React*.
2. *Frontend React* muestra el formulario y permite buscar alimentos por nombre (petición a *Spring Boot*).
3. Empleado busca un alimento y selecciona uno de los resultados.
4. *Frontend React* añade el alimento como ingrediente al producto (opcionalmente permite modificar la cantidad en gramos).
5. Empleado puede modificar la cantidad de cada ingrediente o eliminar ingredientes.
6. Al guardar el producto:
   a) *Frontend React* valida los campos y envía la lista de ingredientes (con cantidades) junto al resto de datos al *endpoint* de *Spring Boot*.
   b) *Spring Boot* valida los datos y verifica permisos:
      1) Si el usuario no tiene permisos, responde 403.
      2) Si hay errores de validación, responde 400.
   c) *Spring Boot* crea o actualiza el producto y sus ingredientes en la base de datos.
   d) *Spring Boot* recalcula automáticamente los nutrientes (suma de los nutrientes de cada ingrediente en proporción a la cantidad en gramos de ese ingrediente en el producto) y alérgenos totales (unión de conjuntos entre listas de alérgenos de los ingredientes) del producto .
   e) *Spring Boot* responde a *React* con el DTO del producto, incluyendo los nutrientes y alérgenos calculados.
7. *Frontend React* muestra el producto actualizado y los mensajes de éxito o error.

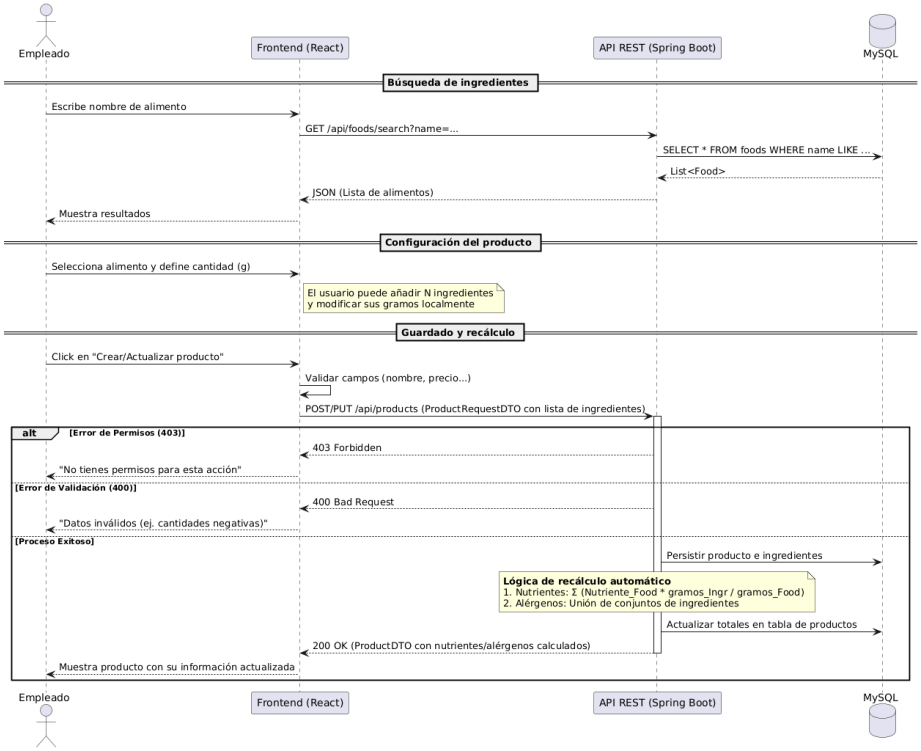


Figura C.18: Diagrama de secuencia para la operación de añadir ingredientes a productos

## D) Añadir ítems a un pedido

1. Empleado accede al *dashboard* de pedidos en *React*.
2. *Frontend React* realiza una petición a *Spring Boot* para obtener la lista de productos disponibles.
3. Empleado selecciona productos para añadirlos como ítems al pedido.
4. *Frontend React* añade los ítems al pedido (opcionalmente permite modificar la cantidad de cada ítem).
   *a*) Cada vez que se añade un ítem, se persiste y se actualiza la vista.
5. Empleado puede modificar la cantidad de cada ítem o eliminar ítems.
6. Al guardar el pedido (opción disponible cuando se han modificado las cantidades de los ítems):
   *a*) *Frontend React* valida los campos y envía la lista de ítems (producto + cantidad) junto al resto de datos al *endpoint* de *Spring Boot*.

b) *Spring Boot* valida los datos y verifica permisos:

1) Si el usuario no tiene permisos, responde 403.
2) Si hay errores de validación, responde 400.

c) *Spring Boot* actualiza el pedido y sus ítems en la base de datos.

d) *Spring Boot* recalcula automáticamente el precio total del pedido (sumando los subtotales de cada ítem: cantidad × precio unitario del producto).

e) *Spring Boot* responde a *React* con el DTO del pedido, incluyendo el total calculado.

7. *Frontend React* muestra el pedido actualizado y los mensajes de éxito o error.

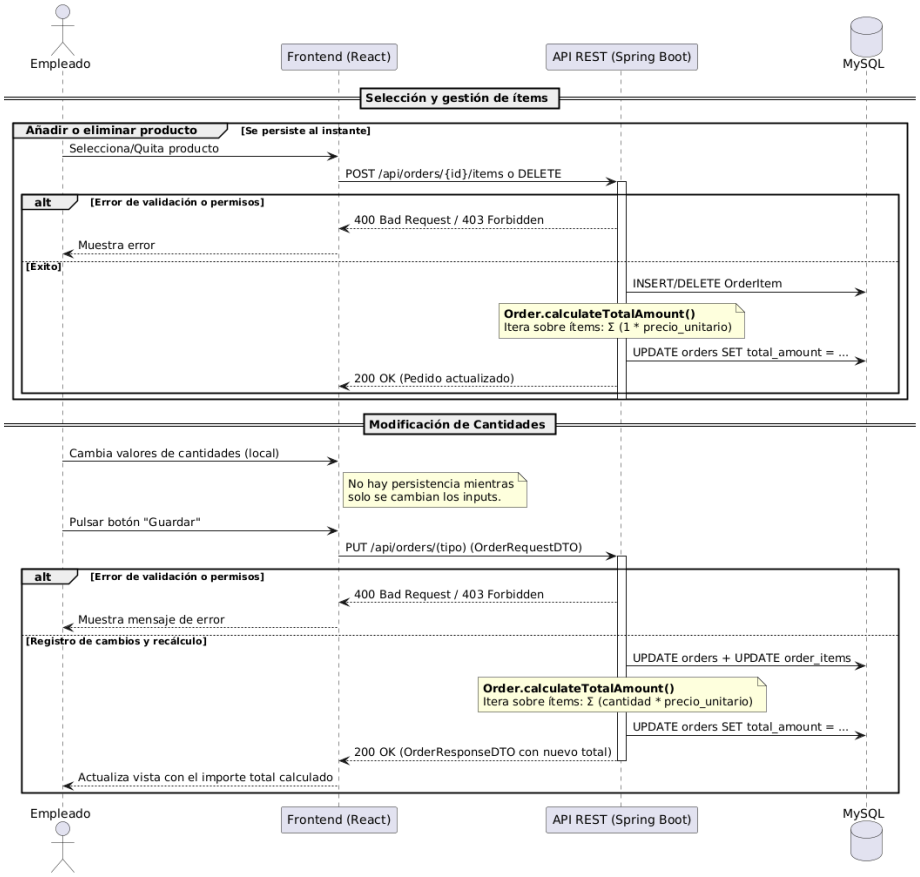


Figura C.19: Diagrama de secuencia para la operación de añadir ítems a pedidos

### E) Búsqueda y creación de alimentos mediante API Externa (*Open Food Facts*)

1. Empleado accede a la sección de creación de alimentos en la aplicación *React* y selecciona la pestaña de búsqueda externa (*OFF*).
2. Empleado introduce un texto en la barra de búsqueda (puede ser nombre, marca, etiqueta, etc.).
3. *Frontend React* envía el texto de búsqueda a *Spring Boot* mediante una petición al *endpoint* `/api/foods/external/search`.
4. *Spring Boot* recibe la petición y realiza una llamada `HTTP` al *endpoint* de búsqueda de *OFF* (`/cgi/search.pl`), solicitando solo los campos mínimos necesarios (nombre, imagen, *barcode*...).
5. *Spring Boot* recibe la respuesta de *OFF* (lista de productos) y la reenvía directamente a *React*, para evitar retardos.
6. *Frontend React* muestra el listado de resultados al usuario, permitiendo seleccionar uno.
7. Cuando el empleado selecciona un alimento, *React* envía el código de barras del producto seleccionado a *Spring Boot* mediante el endpoint `/api/foods/external/{barcode}`.
8. *Spring Boot* realiza una nueva petición a *OFF*, esta vez al *endpoint* de detalles del producto (`/api/v2/product`), solicitando toda la información relevante (nutrientes, alérgenos, métricas, etc.).
9. *Spring Boot* adapta los datos recibidos de *OFF* a su modelo interno de alimento (`Food`) y persiste el nuevo alimento en la base de datos.
10. *Spring Boot* responde a *React* con el DTO del alimento creado.
11. *Frontend React* muestra un mensaje de éxito al usuario, informando de la creación.

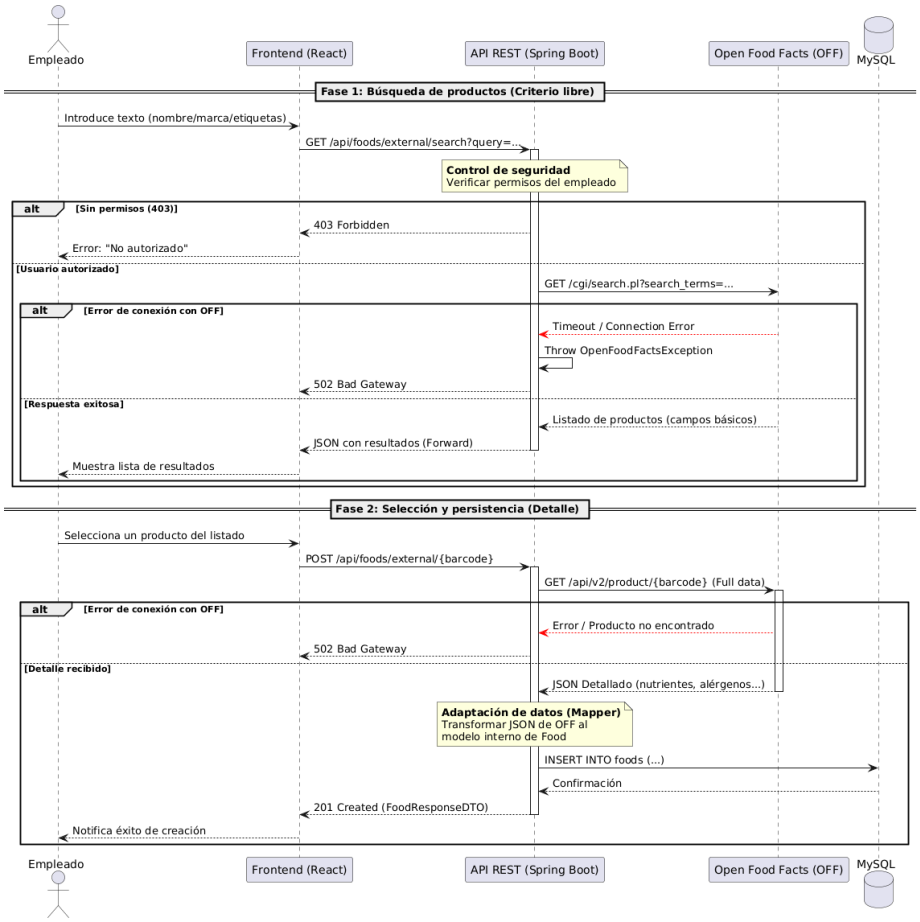


Figura C.20: Diagrama de secuencia para la búsqueda y creación de un alimento mediante *Open Food Facts*

*Apéndice D*

# Documentación técnica de programación

## D.1. Introducción

Este apéndice ofrece el contexto necesario para comprender la implementación de la aplicación, desde su base estructural hasta su capacidad de expansión. El contenido está diseñado para facilitar tanto la puesta en marcha del sistema como su evolución técnica, aprovechando una arquitectura **modular** que prioriza la **escalabilidad**. A través de las siguientes secciones, se establece el marco de trabajo para navegar por la lógica del proyecto, configurar su entorno y añadir nuevas funcionalidades de manera coherente con el diseño original.

## D.2. Estructura de directorios

En este apartado se detalla la organización del proyecto, el cual sigue una estructura modular que **separa** claramente las responsabilidades del **cliente** (*frontend*) y del **servidor** (*backend*), facilitando así el mantenimiento y la escalabilidad del *software*.

### Organización del repositorio y directorio de aplicación

A continuación, se presenta la jerarquía de archivos en la raíz del repositorio, donde se gestionan la documentación, el código principal y los flujos de integración continua.

```
/
├── .github/workflows -> Automatización de flujos CI/CD
│   └── sonar-cloud.yml -> Tests de API y reporte a SonarCloud
├── app/ -> Código fuente y orquestación de servicios
├── docs/ -> Documentación técnica y manuales
├── .gitattributes -> Configuración de atributos de Git
├── .gitignore -> Exclusiones del control de versiones
└── sonar-project.properties -> Configuración para análisis CI monorepo
```

El directorio `app` contiene los servicios que componen el ecosistema de *Orderly* y los archivos necesarios para su orquestación.

```
app/
├── frontend/ -> Cliente web (React + Vite)
│   ├── public/ -> Activos estáticos (iconos SVG alérgenos)
│   ├── src/ -> Componentes y lógica de interfaz
│   ├── Dockerfile -> Definición del contenedor de frontend
│   ├── package.json -> Dependencias y scripts de Node.js
│   └── vite.config.js -> Configuración del servidor Vite
├── spring-boot/ -> Servidor API REST (Spring Boot)
│   ├── .mvn/wrapper/ -> Gestión de versión local de Maven
│   ├── src/ -> Código fuente
│   │   ├── main/ -> Lógica de negocio y controladores
│   │   └── test/ -> Pruebas de la API REST
│   ├── Dockerfile -> Build multi-etapa para el backend
│   ├── mvnw -> Ejecutable de Maven Wrapper
│   ├── mvnw.cmd -> Ejecutable de Maven Wrapper
│   └── pom.xml -> Gestión de dependencias de Java
├── .env.example -> Plantilla de variables de entorno
└── docker-compose.yml -> Orquestación de servicios y red
```

## Desglose del directorio `frontend`

A continuación se muestran los directorios y archivos principales contenidos en la carpeta `frontend/src`. El directorio `common` contiene recursos utilizados globalmente por el resto de módulos, como los *layouts* de las páginas de gestión o componentes de navegación (`Navbar`, `Sidebar`, etc.).

El directorio `i18n` alberga toda la lógica relativa a la **internacionalización** de la aplicación. Contiene un directorio por cada idioma soportado (inglés y español). Cada directorio se compone de ***namespaces***, esto es, archivos JSON que albergan los *strings* de las traducciones. Siguiendo el

patrón modular y teniendo en cuenta la posibilidad futura de cargas dinámicas, se ha creado un *namespace* por cada módulo, junto a uno común para traducciones recurrentes.

```
frontend/src/
├── common/ -> Recursos compartidos y componentes transversales
│   ├── components/ -> Componentes UI reutilizables
│   │   ├── feedback/ -> Componentes de estado (carga, errores)
│   │   └── home/ -> Elementos visuales de la página de bienvenida
│   ├── hooks/ -> Lógica de estado común
│   ├── layouts/ -> Estructuras de página (ej: MainLayout)
│   └── pages/ -> Vistas generales (MainPage)
├── context/ -> Gestión de estado global (auth, permisos)
│   └── AuthContext.jsx -> Lógica de sesión y control de acceso
├── i18n/locales/ -> Traducciones multi-idioma
│   ├── en/ -> Diccionarios en inglés
│   ├── es/ -> Diccionarios en español
│   └── index.js -> Configuración de i18next y detección de idioma
├── modules/ -> Funcionalidades de negocio (Pedidos, Productos...)
├── services/backend/ -> Comunicación con la API REST
├── utils/ -> Constantes y funciones de ayuda (navegación, permisos)
├── App.jsx -> Componente raíz de la aplicación
├── AppRouter.jsx -> Definición de rutas públicas y protegidas
└── main.jsx -> Punto de entrada y configuración de Mantine/Providers
```

El directorio `modules` contiene todo lo relativo al funcionamiento de las páginas de gestión y se organiza en carpetas por módulo (figura D.1). En general, todos los módulos comparten la misma estructura de carpetas básica, la cual se puede visualizar en la figura D.2.

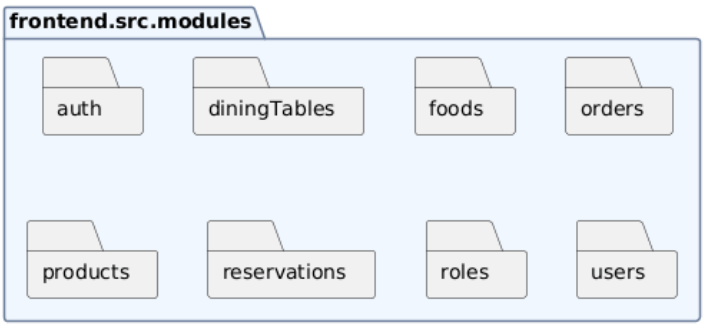


Figura D.1: Módulos funcionales del *frontend*

- `components`: incluye los componentes de interfaz reutilizables del módulo.
- `hooks`: cada módulo dispone de un *hook* principal (`use<Entidad>.js`) que centraliza la lógica de comunicación con el *backend*.
- `pages`: se implementan al menos tres páginas: `ListPage`, `CreatePage` y `EditPage`.

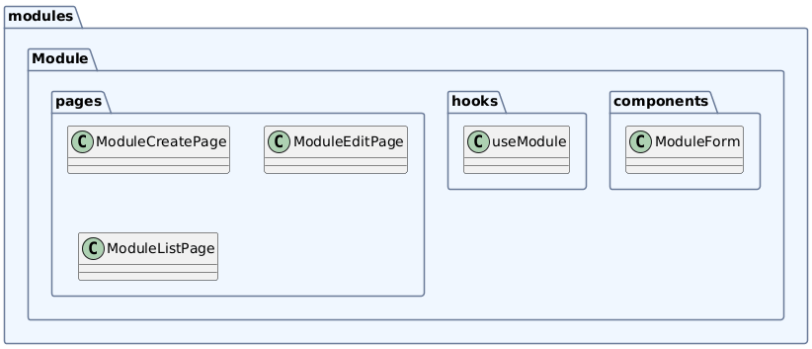


Figura D.2: Estructura compartida entre los módulos funcionales de *React*

Más allá de dicha estructura homogénea, cada módulo suele implementar componentes **propios**, particulares para su funcionamiento. Los directorios `auth` y `orders` se alejan un poco de la estructura general, pero mantienen las bases compartidas.

## Desglose del directorio `spring-boot`

La API REST *Spring Boot* también tiene una organización en carpetas según *features*, frente a la organización clásica según el tipo de componente (`controller`, `service`, `repository`).

Con diferencia respecto del *frontend*, en este caso algunos directorios de entidades contienen a otros (figura D.3). La estructura del dominio sigue un enfoque inspirado en el patrón de **agregados** del diseño guiado por dominios (*Domain-Driven Design*), donde ciertas entidades actúan como raíces (*aggregate roots*) que aglutinan y **gestionan el ciclo de vida** de otras entidades dependientes [6].

Como se ha mencionado, esta jerarquía se refleja en la organización de paquetes, donde las entidades propietarias de la relación (como `User` o `Product`) contienen a sus entidades asociadas (`Role`, `Ingredient`) para

garantizar la integridad de los datos y centralizar la lógica de negocio, asegurando que el acceso y la modificación de los elementos hijos, en lo que a la **relación** se refiere, se realice siempre a través de su entidad principal.

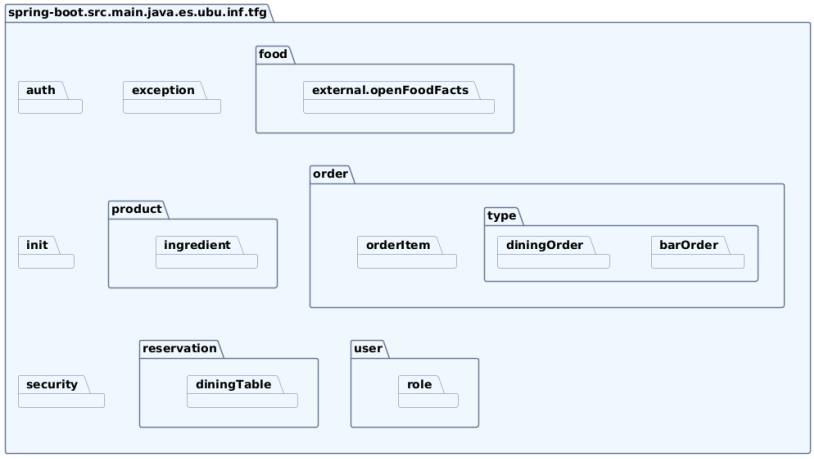


Figura D.3: Módulos funcionales del *backend*

El directorio `exception` centraliza la **gestión de errores** mediante un manejador global que captura excepciones específicas (credenciales inválidas, recursos en uso, etc.). Su propósito es **transformar** fallos internos en **respuestas estructuradas** y consistentes para el cliente, asegurando que cada error devuelva un código de estado HTTP adecuado y un mensaje legible.

Por otro lado, el directorio `security` contiene la **infraestructura de seguridad** basada en *Spring Security*, gestionando la autenticación y autorización de la API. Incluye la configuración de **filtros** de acceso, la definición de usuarios personalizados y servicios específicos para verificar permisos, garantizando que solo los usuarios autenticados y con los permisos adecuados puedan acceder a los **recursos protegidos**.

Finalmente, el directorio `init` contiene los archivos de **inicialización** de la base de datos con registros por defecto, la cual puede activarse o no según una variable de entorno, como se detalla en el apartado *Inicialización de la BBDD con datos de prueba*.

El resto de directorios (exceptuando `auth`), se corresponden con los **módulos** de gestión de la aplicación (alimentos, productos, reservas, etc.). Dentro de cada uno se sitúan sus componentes **específicos**, como archivos de objetos embebidos o tipos enumerados, presentes en los campos de las entidades.

Dejando a un lado las particularidades de cada entidad, todos los directorios de módulos comparten una **estructura** mínima de archivos y directorios, la cual se puede visualizar en la figura D.4.

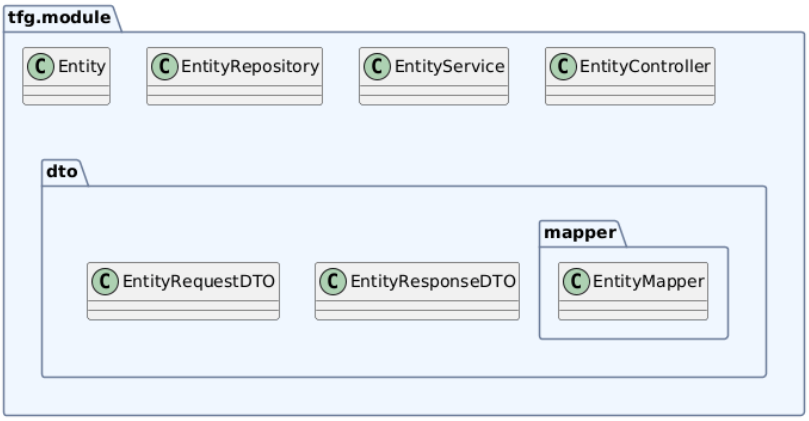


Figura D.4: Estructura compartida entre los módulos funcionales de *Spring Boot*

## D.3. Manual del programador

Esta sección tiene como objetivo profundizar en los aspectos técnicos y las decisiones de diseño que permiten el funcionamiento, la mantenibilidad y la extensibilidad de *Orderly*. Mientras que en el apartado *D.4 Compilación, instalación y ejecución del proyecto* se detallan los pasos operativos para desplegar el sistema, en esta sección se analiza la arquitectura de *software* desde el punto de vista del desarrollo y la infraestructura.

### Estrategias de construcción (*build*)

#### *Backend*: *Multi-stage Build* en *Spring Boot*

Para el servicio de *Spring Boot* se ha implementado una estrategia de construcción en múltiples etapas (*multi-stage build*). Esta técnica es una buena práctica en ingeniería de *software* que divide el `Dockerfile` en dos fases diferenciadas [4]:

1. **Etapa de construcción:** utiliza una imagen ligera de *Maven* con el JDK 21 para descargar las dependencias, compilar el código fuente y generar el archivo ejecutable (`.jar`).

2. **Etapa de ejecución:** utiliza una imagen de JRE (*Java Runtime Environment*) sobre *Alpine Linux*, **mucho más ligera**, a la que solo se copia el archivo `.jar` generado en la fase anterior.

**Dockerfile (Spring Boot)**

```
# Construcción
FROM maven:3.9.11-eclipse-temurin-21 AS build

WORKDIR /spring-boot
COPY pom.xml ./
RUN mvn dependency:go-offline
COPY src ./src
RUN mvn clean package -DskipTests

# Ejecución
FROM eclipse-temurin:21-jre-alpine

WORKDIR /app
COPY -from=build /spring-boot/target/*.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Esta separación **reduce** drásticamente el **tamaño final de la imagen** de producción (al no incluir herramientas de compilación ni el código fuente original) y aumenta la seguridad, al reducir la superficie de ataque del contenedor.

### ***Frontend*: *React* + *Vite* con *HMR***

El servicio de *frontend* aprovecha las capacidades de *Vite* para ofrecer una experiencia de **desarrollo** ágil mediante *HMR* (***Hot Module Replacement***). A diferencia del *Hot Reload* tradicional, que reinicia la aplicación completa ante cualquier cambio, el *HMR* permite **sustituir** exclusivamente el **módulo** modificado en tiempo de **ejecución**. Esto permite al programador visualizar cambios en la interfaz de manera instantánea sin perder el estado actual de la aplicación (por ejemplo, manteniendo modales abiertos o formularios rellenos) [15].

La viabilidad técnica de esta función dentro de un entorno contenerizado se apoya en una gestión específica de **volúmenes** en *Docker Compose*:

a) Se **vincula** el directorio **local** `./frontend` con la ruta `/app` del contenedor (*bind mount*). Esto permite que cualquier edición realizada en el IDE del anfitrión se refleje de inmediato en el sistema de archivos del contenedor.

b) Se ha definido un **volumen específico** (anónimo) para la ruta `/app/node_modules`. Su propósito es **aislar las dependencias** instaladas dentro de la imagen (optimizadas para el entorno *Linux* del contenedor) de las que pudieran existir en la máquina local. Esto **evita conflictos** de binarios entre distintos sistemas operativos y garantiza que el proceso de construcción sea robusto y eficiente.

**YML - Docker Compose (React)**

```yml
react:
  build:
    context: ./frontend
    dockerfile: Dockerfile
  container_name: react
  restart: always
  env_file: .env
  ports:
    - "${REACT_LOCAL_PORT}:${REACT_DOCKER_PORT}"
  depends_on:
    - spring-boot
  volumes:
    - ./frontend:/app
    - /app/node_modules
```

## Configuración centralizada y portabilidad

La portabilidad de *Orderly* se apoya en el uso del archivo de variables de entorno `.env`. El sistema ha sido diseñado para evitar el *"hardcoding"* (escritura rígida en código) de valores sensibles o dependientes del entorno, como puertos, credenciales o roles.

Un ejemplo de archivo `.env`, para ilustrar las variables de las que se hablará a continuación, se puede encontrar en el apartado *Configuración del entorno*.

### Gestión de puertos

El archivo `.env` diferencia entre los puertos internos del contenedor (`DOCKER_PORT`) y los puertos accesibles desde la máquina local (`LOCAL_PORT`).

- Los **puertos de *Docker*** son fijos y corresponden a la configuración estándar del servicio (ej. `8080` para *Spring*).
- Los **puertos locales** son los que el usuario **podrá modificar** si tiene conflictos de red en su equipo o si simplemente quiere asignar otros puertos distintos a los proporcionados por defecto.

Esta configuración está **centralizada**: cuando el programador **cambia el puerto** en el `.env`, el cambio se propaga automáticamente (sería necesario reconstruir los contenedores). Por ejemplo, en el ***backend***, la clase `SecurityConfig.java` lee este valor para configurar las reglas de **CORS**, permitiendo que las peticiones del *frontend* sean aceptadas sin importar en qué puerto se esté ejecutando este último. Del mismo modo, `vite.config.js` utiliza estas variables para levantar el servidor de desarrollo en el puerto deseado.

### Roles y datos de usuario por defecto

Para facilitar la flexibilidad en la gestión de permisos, los nombres de los roles por defecto (`ROLE_ADMIN`, `ROLE_USER`) y las credenciales del administrador inicial se **inyectan** mediante variables de entorno. La clase `UserDataInitializer.java` utiliza estas variables durante el **arranque** para verificar la existencia del usuario administrador y **crearlo si fuera necesario**, permitiendo cambiar el acceso por defecto sin modificar una sola línea de código Java.

### Inicialización de la BBDD con datos de prueba

La inicialización de los roles y el usuario mencionados en el apartado anterior se da **en todo momento**, durante cada arranque del contenedor `spring-boot`.

Adicionalmente, se han añadido múltiples **archivos** de inicialización para el **resto de entidades** del sistema (alimentos, productos, mesas, etc.), con el objetivo de tener un conjunto de **datos de prueba** siempre disponible. Dichos archivos se localizan en el directorio `init` de *Spring Boot* y tienen la particularidad de poder ser **activados** o desactivados mediante la **variable** de entorno `APP_INIT_DEMO_DATA`:

- Si su valor es `true`, se ejecutarán todos los archivos de inicialización, rellenando las tablas con múltiples registros de prueba.
    - El **orden de ejecución** viene dado por la anotación `@Order`, de manera que las entidades no propietarias son inicializadas **antes** que las entidades que dependen de otras.
    - En caso de ya existir registros en una tabla, se **salta** su inicialización y se procede con el siguiente *script*.
- Si su valor es `false`, ninguno de los archivos se ejecutará. Solo se dispondrá de los **roles y usuario por defecto** descritos en *Roles y datos de usuario por defecto*, los cuales son necesarios para el correcto funcionamiento del sistema.

### Consistencia horaria (*Timezone*)

Un aspecto crítico en la gestión de **reservas** y **auditoría** de entidades (en particular, de pedidos) es la precisión de las fechas. Por defecto, los contenedores suelen operar en tiempo UTC, lo que causaba **desajustes** en campos como `createdAt` o en las horas de reserva respecto a la hora local de Madrid (UTC+1/+2).

Para solucionar esto, se ha introducido la variable `APP_TIMEZONE`. Al establecerla en el archivo `.env` (ej. *Europe/Madrid*), *Docker* configura la variable de entorno `TZ` en el sistema operativo del contenedor. Esto garantiza que tanto la máquina virtual de Java como el servidor *MySQL* operen bajo el **mismo contexto horario que el usuario**, evitando inconsistencias de datos y errores en la lógica de negocio temporal.

## Integrar una nueva API nutricional

Como ya se introdujo en el apartado *Desacoplamiento de integraciones con APIs nutricionales externas*, la arquitectura del módulo de alimentos ha sido diseñada para maximizar el desacoplamiento entre la lógica de negocio interna y las fuentes externas de datos nutricionales.

Los pasos para integrar una nueva API nutricional, desde un enfoque más orientado al código, son:

1. **Crear el subdirectorio de integración:** ubicar la nueva API en `food/external/<nombreAPI>`, implementando los servicios de comunicación y los *mappers* para transformar los datos externos al modelo interno (`FoodRequestDTO`).

2. **Adaptar el controlador:** modificar `FoodController` para enrutar las peticiones a la nueva API según corresponda, manteniendo la abstracción y evitando referencias directas a la implementación.

3. **Actualizar el *frontend*:** si se desea permitir la selección entre varias APIs, adaptar los *hooks* y componentes para aceptar un parámetro de selección, pero sin acoplarlos a la lógica interna de cada API.

   - Los archivos de traducción relativos al módulo de alimentos (`i18n/<idioma>/foods.js`) también deberían actualizarse para reflejar la nueva API a usar (o la convivencia de varias).

4. **Pruebas y validación:** verificar que la integración no afecta al resto del sistema y que la convivencia entre APIs es transparente para el usuario y el desarrollador.

Este diseño modular y desacoplado facilita enormemente la evolución del sistema, permitiendo la sustitución, ampliación o **coexistencia** de múltiples fuentes externas sin comprometer la estabilidad ni la mantenibilidad del código.

## D.4. Compilación, instalación y ejecución del proyecto

Este apartado detalla los pasos necesarios para preparar el entorno, configurar las variables necesarias y poner en marcha la aplicación *Orderly*. Gracias al uso de la tecnología de **contenedores**, este proceso se ha simplificado para garantizar que la aplicación se comporte de manera idéntica **independientemente del sistema operativo** anfitrión.

### Requisitos de *software*

Para la correcta ejecución del sistema, es imprescindible disponer de las herramientas de virtualización de *Docker*. En concreto, se requiere:

- ***Docker Engine* y *Docker Compose*:** motor de ejecución de contenedores y herramienta para la definición y gestión de aplicaciones multi-contenedor [2].

- ***Docker Desktop*:** se recomienda el uso de esta interfaz gráfica (disponible para *Windows*, *macOS* y *Linux*) ya que integra tanto el motor

de *Docker* como *Docker Compose* en un único paquete [3]. Su utilidad reside en facilitar la monitorización de los estados de los contenedores, la gestión de volúmenes de datos y la visualización de *logs* de los servicios (base de datos, *backend* y *frontend*) de manera centralizada y sencilla.

- **Git:** sistema de control de versiones necesario para la descarga (clonación) del código fuente desde el repositorio remoto y la gestión de las diferentes versiones del proyecto [5].

## Obtención del código fuente

El código fuente del proyecto se encuentra alojado en un repositorio de *GitHub*. Para obtener una copia local, se debe clonar el repositorio mediante el siguiente comando en la terminal:

**Terminal**

```
git clone https://github.com/aperezolmos/Orderly.git
```

Si se quiere descargar el código de una versión específica, se puede acceder a la sección *Releases* en el repositorio de *GitHub* y descargar manualmente el código fuente de la versión escogida, siendo **v1.0.0** la versión más reciente.

Una vez finalizada la descarga, se debe acceder al directorio del proyecto que contiene la aplicación:

**Terminal**

```
cd Orderly/app
```

## Configuración del entorno

La aplicación utiliza variables de entorno para gestionar aspectos críticos como las credenciales de la base de datos, los puertos de red y la comunicación entre servicios. Por razones de seguridad y buenas prácticas, el archivo que contiene estos valores reales (`.env`) **no se incluye** en el repositorio de código.

En su lugar, se proporciona un archivo de **plantilla** denominado `.env.example`. Para configurar la aplicación, el usuario debe crear una copia de este archivo y renombrarla como `.env`. Esto puede realizarse manualmente o mediante el siguiente comando:

**Terminal**

```
cp .env.example .env
```

A continuación, se debe editar el archivo `.env` recién creado. Aunque los valores pueden ajustarse según las necesidades de red del sistema anfitrión, se recomienda la siguiente **configuración** para una **ejecución estándar**:

**Plaintext**

```
# =====================
# Database
# =====================
MYSQL_ROOT_PASSWORD=root
MYSQL_DATABASE=orderly_db
MYSQL_USER=user
MYSQL_PASSWORD=user
MYSQL_LOCAL_PORT=3307
MYSQL_DOCKER_PORT=3306

# =====================
# REST API (Spring Boot)
# =====================
SPRING_LOCAL_PORT=8080
SPRING_DOCKER_PORT=8080

# Application default users configuration
DEFAULT_ADMIN_USERNAME=admin
DEFAULT_ADMIN_PASSWORD=admin
DEFAULT_ADMIN_ROLE=ROLE_ADMIN
DEFAULT_USER_ROLE=ROLE_USER

# Application default users configuration
APP_INIT_DEMO_DATA=true


# =====================
# Frontend (React)
# =====================
REACT_LOCAL_PORT=5173
REACT_DOCKER_PORT=5173
```

```
VITE_API_BASE_URL=http://localhost:${SPRING_LOCAL_PORT}/api
REACT_URL=http://localhost:${REACT_LOCAL_PORT}


# =====================
# Time Zone
# =====================
APP_TIMEZONE=Europe/Madrid
```

## Permisos de acceso a archivos (*File Sharing*)

Antes de proceder con la ejecución, es importante tener en cuenta que, dependiendo de la configuración de *Docker Desktop* y del sistema operativo anfitrión, puede ser necesario **autorizar explícitamente el acceso** de *Docker* a la carpeta del proyecto.

Si al intentar levantar los contenedores se produce un **error** de tipo ***Mounts denied*** o problemas de permisos al montar los volúmenes, se deben seguir estos pasos para solucionar el conflicto de *File Sharing* [9, 17]:

1. Abrir la interfaz de ***Docker Desktop***.
2. Acceder a `Settings > Resources > File sharing`.
3. Añadir la **ruta** del directorio donde se ha clonado el proyecto *Orderly* mediante el botón +.
4. Hacer clic en ***Apply & Restart*** para aplicar los cambios.

**Nota técnica:** este paso es generalmente innecesario en sistemas *Windows 11* que utilicen el motor de ***WSL 2***, ya que la gestión de permisos y recursos se realiza de forma automática a través del subsistema de *Linux*. No obstante, es un requisito común en instalaciones que utilicen el *backend* de ***Hyper-V*** o en sistemas ***macOS***.

## Ejecución de la aplicación

Una vez configurado el archivo de entorno, el despliegue de los servicios es automático. El flujo de ejecución se rige por el archivo **`docker-compose.yml`**, el cual coordina la lectura de los `Dockerfile` específicos para el *backend*

(*Spring Boot*) y el *frontend* (*React*), además de levantar una instancia de *MySQL*.

Para iniciar el proceso de construcción de imágenes y levantado de contenedores, se debe ejecutar el siguiente comando dentro del directorio `app`:

**Terminal**

```
docker-compose up --build -d
```

- El *flag* `--build` asegura que las imágenes se construyan con la **última versión** del código fuente disponible.
- El flag `-d` (*detached mode*) permite que los contenedores se ejecuten en segundo plano, liberando la terminal para seguir introduciendo comandos mientras la aplicación permanece activa.

*Docker* se encargará de descargar las imágenes base, instalar las dependencias necesarias (***Maven*** para *Java* y ***NPM*** para *Node.js*) y configurar la **red interna** para que los tres servicios puedan comunicarse entre sí.

En caso de no poder llevar a cabo la ejecución del comando debido a un **error** *Mounts denied*, se recomienda seguir los pasos indicados en el apartado *Permisos de acceso a archivos (File Sharing)*.

Por otro lado, si el servicio `db` mostrara un estado *Unhealthy*, esto indicaría que el *healthcheck* ha **fallado** al no recibir una respuesta válida de *MySQL* en el tiempo previsto. En este caso, se debe editar el archivo `docker-compose.yml` y aumentar los valores de `start_period` (tiempo de gracia inicial) o `retries` (número de reintentos), permitiendo así que el motor de base de datos complete su inicialización en equipos con menor rendimiento. Se trata de un escenario poco probable debido a los márgenes de tiempo holgados que se han definido originalmente, aunque factible en sistemas con recursos de *hardware* muy limitados.

## Acceso al sistema

Tras finalizar el proceso de arranque (el cual puede demorarse unos minutos en la primera ejecución mientras se descargan las dependencias), la aplicación estará disponible para su uso. Si abrimos *Docker Desktop*, podrán visualizarse los tres contenedores mediante los que se despliega la aplicación (figura D.5). Haciendo *click* sobre el nombre de cada contenedor, podrán visualizarse sus *logs*, si se quisieran comprobar los detalles del arranque.

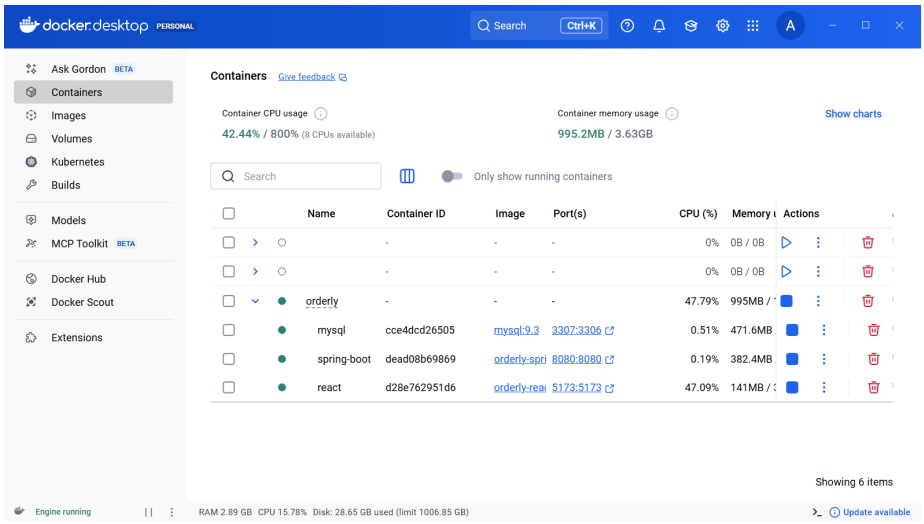


Figura D.5: Contenedores *Docker* con los servicios de *Orderly*

Teniendo en cuenta la configuración por defecto del archivo `.env`, se puede acceder a la interfaz de usuario a través de cualquier navegador web moderno en la siguiente dirección:

**URL de la aplicación:** `http://localhost:5173`

Asimismo, la API REST y la base de datos estarán escuchando en los puertos `8080` y `3307` respectivamente, permitiendo su inspección técnica si fuera necesario.

## D.5. Pruebas del sistema

Para garantizar la calidad y fiabilidad de la API REST desarrollada en *Spring Boot*, se han implementado diferentes tipos de pruebas automatizadas, las cuales se sitúan en el directorio `spring-boot/src/test`. Es importante destacar que el alcance de las pruebas se ha limitado exclusivamente al *backend*. No se han realizado pruebas sobre el *frontend* (*React*), ya que el objetivo principal era asegurar la **robustez** de la **lógica** de negocio y la **persistencia** de datos en la API.

Se han desarrollado tanto pruebas unitarias como pruebas de integración, siguiendo buenas prácticas de desarrollo en aplicaciones empresariales con *Spring Boot*:

- **Pruebas unitarias:** se han creado tests para los servicios y *mappers* de todas las entidades principales del sistema (alimentos, productos, mesas, reservas, pedidos, usuarios, roles). Estas pruebas verifican el correcto funcionamiento de la lógica interna de cada clase, **simulando dependencias** mediante *mocks* (***Mockito***), y comprobando que los métodos devuelven los resultados esperados ante distintos escenarios.

- **Pruebas de integración**: para las entidades que **gestionan relaciones** y flujos críticos (por ejemplo, usuarios sobre roles, productos sobre ingredientes, pedidos sobre ítems), se han implementado tests de integración. Estas pruebas se ejecutan sobre una base de datos en memoria (**H2**) y validan que los procesos completos funcionen correctamente, incluyendo la **persistencia** y **consistencia** de las relaciones entre entidades. Así, se comprueba que los cambios realizados en una entidad se reflejan adecuadamente en las entidades relacionadas.

El *backend* cuenta con una **cobertura** de pruebas del 50 %. Aunque no se han cubierto todos los flujos y casos posibles de la API REST, sí se han probado los escenarios más relevantes y **críticos** para el correcto funcionamiento del sistema, como la gestión de **relaciones**, la validación de datos y los procesos de negocio principales. La cobertura no era un requisito prioritario, pero se ha procurado asegurar la fiabilidad de los módulos más importantes.

## Ejecución de las pruebas

Las pruebas se pueden ejecutar fácilmente mediante *Maven*, situándonos en el directorio `app/spring-boot` y utilizando el siguiente comando en el terminal:

**Terminal**

```
mvn test
```

Esto ejecutará tanto las pruebas unitarias como las de integración, generando un **informe de cobertura** *Jacoco* en el directorio `target`. Además, al utilizar *Spring Boot* y H2, las pruebas de integración se ejecutan en un **entorno aislado**, sin afectar a la base de datos real.

*Apéndice* $E$

# Documentación de usuario

## E.1. Introducción

Este apéndice constituye la guía técnica y funcional destinada al usuario final de *Orderly*. En las secciones siguientes se describen los requisitos previos para el acceso a la plataforma, el proceso de puesta en marcha y el manual de operación de las distintas herramientas de gestión de restauración que ofrece la aplicación.

## E.2. Requisitos de usuarios

Para garantizar una experiencia de uso fluida y el correcto funcionamiento de todas las características de la interfaz, el usuario debe cumplir con los siguientes requisitos mínimos:

- **Conectividad:** disponer de acceso a la red local (en caso de despliegue privado) o conexión a Internet. La comunicación con la API externa requiere de conexión a Internet.

- ***Software*:** un navegador web moderno y actualizado (por ejemplo, *Google Chrome*, *Mozilla Firefox*, *Microsoft Edge* o *Safari*).

- ***Hardware*:** Al tratarse de una aplicación con diseño adaptativo (*"responsive"*), es posible acceder desde ordenadores de sobremesa, portátiles, tabletas o dispositivos móviles con resolución de pantalla suficiente para la visualización de datos.

## E.3. Instalación

Desde la perspectiva del usuario final, *Orderly* **no requiere** de ninguna instalación de *software* local ni configuración de bases de datos, ya que se trata de una aplicación web centralizada.

Se presupone que el sistema ha sido previamente **desplegado** en un servidor o entorno de ejecución siguiendo los pasos técnicos detallados en el apartado *D.4 Compilación, instalación y ejecución del proyecto*. Una vez que los contenedores del sistema están activos, el usuario solo deberá:

1. Abrir el navegador web.
2. Introducir la dirección URL proporcionada por el administrador (por defecto, `http://localhost:5173` en entornos de ejecución local).
3. Iniciar sesión con las credenciales correspondientes.

## E.4. Manual del usuario

A continuación se describen las funcionalidades que pueden llevarse a cabo en la aplicación, junto a los componentes y pantallas que las sostienen.

### Página de bienvenida

Al acceder a la aplicación, la primera vista que se visualiza es la página de bienvenida (figura E.1). Se trata de una **carta de presentación** de *Orderly*, la cual expone las funcionalidades que ofrece la aplicación. También contiene un botón *Ver Documentación*, que redirige al repositorio de *GitHub*.

Se puede acceder al inicio de sesión tanto desde la barra de navegación como desde el botón *Comenzar*. Suponiendo el acceso de un nuevo usuario, empezaremos pulsando el botón de *Registrarse*.

Figura E.1: Página de bienvenida

## Registro de usuario

La pantalla de registro permite a un nuevo usuario generar sus credenciales. Como mínimo, un usuario deberá introducir un **nombre de usuario** y una **contraseña** para enviar el formulario. La contraseña debe confirmarse y ambas deben coincidir (figura E.2).

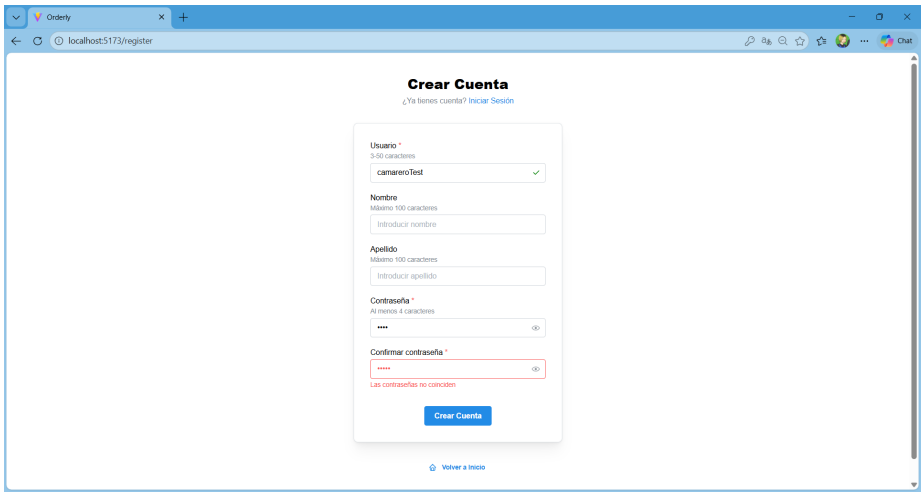


Figura E.2: Página de registro de usuario

Al introducir el nombre de usuario, el sistema valida que dicho nombre **no esté ya en uso**, es decir, que ya exista otro usuario con dicho *username*.

- En caso de que el nombre de usuario esté en uso, se mostrará una ✗ y no se permitirá el envío de formulario.
- Si el nombre de usuario no está en uso, se mostrará un ✓ y se podrá enviar el formulario, siempre que se hayan rellenado todos los campos requeridos.

Una vez pulsado el botón de creación, se redirigirá al usuario a la página de inicio, que inicialmente aparecerá vacía debido a que el usuario no tendrá asignados los permisos necesarios, como se detalla en *Página de Inicio (Home)*. Las únicas acciones que puede realizar un usuario recién registrado, al que aún no se le han asignado permisos, son visualizar y editar su perfil.

## Inicio de sesión

En caso de que el usuario ya tenga una cuenta existente, lo único que debe hacer es introducir su nombre de usuario y contraseña para acceder. Si el nombre de usuario introducido no existe o la contraseña no se corresponde con la de dicho usuario, se mostrará un error por credenciales incorrectas (figura E.3).

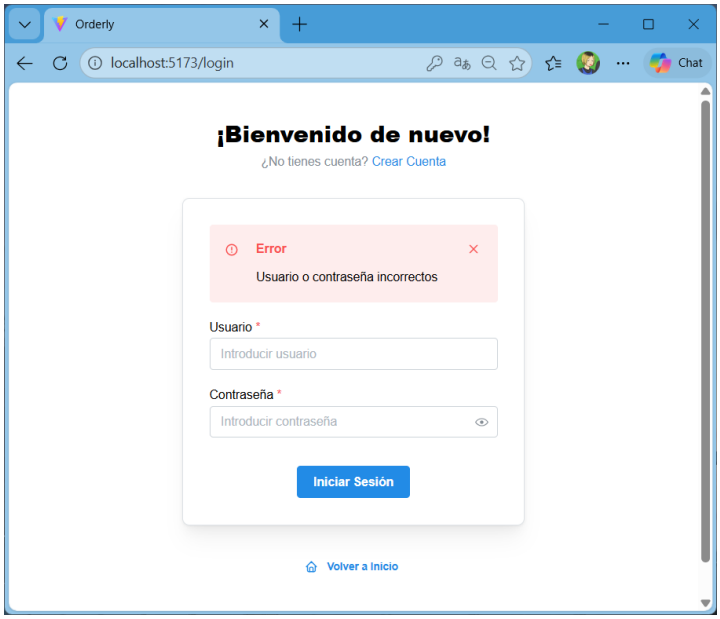


Figura E.3: Página de inicio de sesión

Al iniciar sesión con credenciales válidas, se redirigirá al usuario a la *Página de Inicio (Home)*.

## Perfil de usuario

En esta página (E.4) se muestra toda la información asociada al usuario autenticado, incluyendo sus roles y fechas de creación y última modificación.

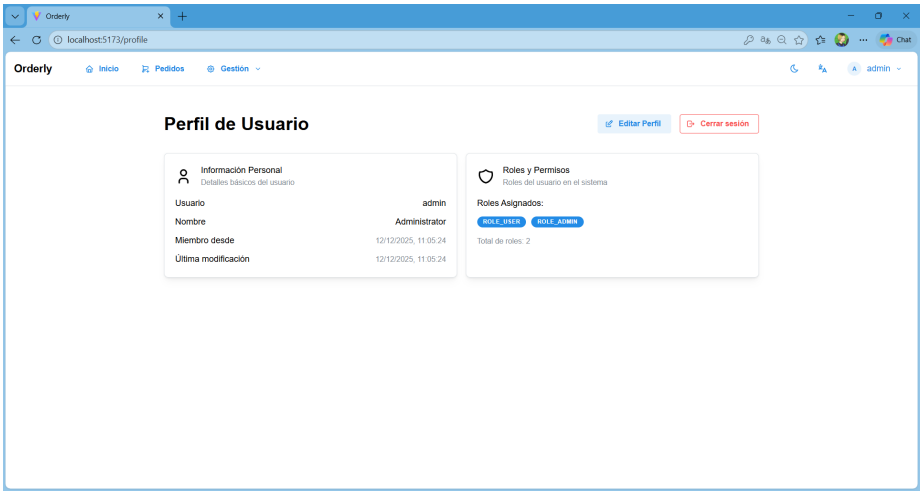


Figura E.4: Perfil de usuario

Las acciones que pueden realizarse desde esta página son *Editar perfil de usuario* y *Cerrar sesión*.

## Editar perfil de usuario

Accediendo a la página de edición (figura E.5) desde el perfil de un usuario, se pueden editar sus campos, con algunas particularidades:

- Si se quiere editar el nombre de usuario, se volverá a realizar la validación de unicidad para comprobar que el nuevo *username* no pertenece ya a otro usuario.
    - Si se modifica el nombre de usuario, se **cerrará sesión** tras enviar el formulario y se deberá volver a ingresar con las nuevas credenciales.
- Se puede editar el perfil sin necesidad de modificar o aportar la contraseña, basta con dejar dicho campo vacío. Sin embargo, en caso de que se quiera modificar, deberán aportarse:
    - Contraseña anterior del usuario.
    - Confirmación de la nueva contraseña.

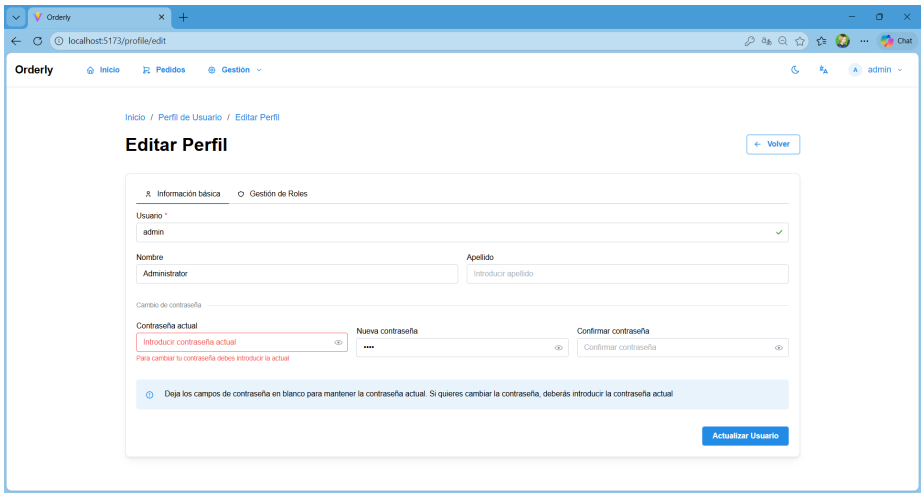


Figura E.5: Página de edición de perfil de usuario

Dependiendo de si el usuario tiene permisos de edición de roles, podrá visualizar una **sección** dentro del formulario que le permitirá añadirse o quitarse roles. Esta funcionalidad se explicará con más detalle en el apartado *Crear usuarios*.

## Cerrar sesión

Se puede salir de la sesión actual en todo momento desde la **barra de navegación** (se explicará con más detalle en el siguiente apartado *Navegación por la aplicación*). También se puede cerrar sesión desde el *Perfil de usuario*, haciendo *click* en el botón asociado a dicha acción.

## Navegación por la aplicación

Antes de comenzar a describir las páginas a las que se puede acceder en la aplicación, es necesario exponer cómo se puede navegar por ella.

Figura E.6: Barra de navegación para escritorio

### Barra de navegación principal (escritorio)

Por defecto, *Orderly* dispone de una barra de navegación superior (figura E.6), la cual se compone de dos secciones.

**Sección izquierda: botones de acceso a los módulos de gestión**

Esta sección contiene dos botones de acceso rápido a páginas principales (*Inicio* y *Panel de pedidos*), pero lo más destacable es su **menú desplegable** de módulos de gestión (figura E.7). Al pulsar o hacer *hover* sobre dicho componente, se desplegará un menú anidado para acceder a las páginas de gestión de los distintos módulos de la aplicación (alimentos, productos, reservas, etc.).

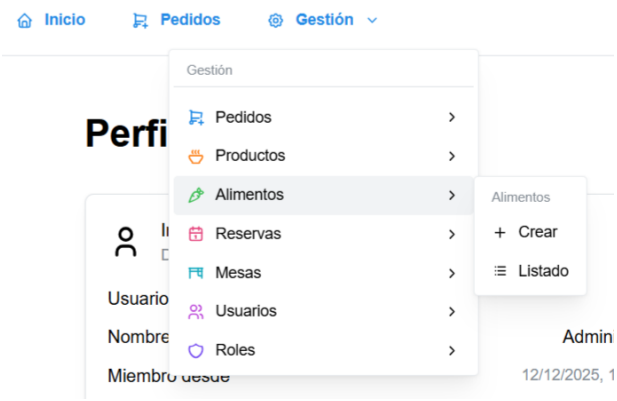


Figura E.7: Sección izquierda de la barra de navegación

La cantidad de **módulos** y **páginas** a mostrar en este menú viene dada por los **permisos** que posea el usuario autenticado. En la sección *Ejemplos de creación de roles* se visualiza con más detalle cómo se modifican los componentes de navegación en función de los permisos que posean los roles del usuario.

**Sección derecha: botones de tema, idioma y autenticación**

*Orderly* permite el **cambio de tema** claro a oscuro y viceversa. Basta con pulsar el botón (o si nos encontramos en el modo oscuro) para cambiar la paleta de colores.

La aplicación también soporta el **cambio de idioma**. *Orderly* está disponible tanto en **inglés** como en **español**. Por defecto, al acceder a la aplicación, el idioma se establece en función del idioma del **navegador** del usuario, pero puede cambiarse en cualquier momento desde el icono , el cual desplegará un **menú** para seleccionar el nuevo idioma.

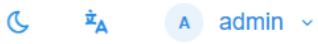


Figura E.8: Sección derecha de la barra de navegación

Finalmente, se muestra el *username* del usuario autenticado y, haciendo *click* sobre él, se despliega un menú que permite navegar al perfil del usuario o cerrar sesión.

### Menú lateral (móvil)

*Orderly* está diseñada para adaptarse a distintos tamaños de pantalla. Es por eso por lo que, en **tamaños reducidos** (como pantallas móviles), la barra de navegación principal se transforma en un menú lateral (figura E.9), accesible desde el icono . Los menús y acciones a realizar coinciden con los presentados en el apartado anterior.

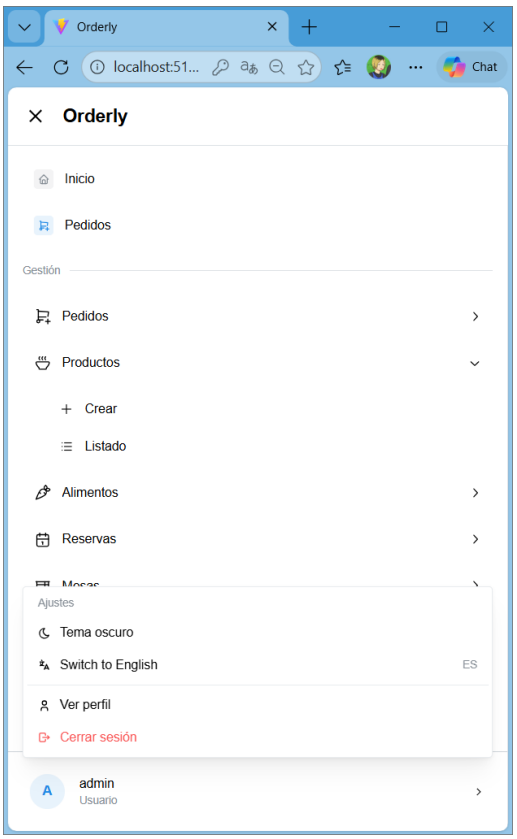


Figura E.9: Menú lateral de navegación para móviles

### *Breadcrumbs*

En las páginas de gestión que se verán a continuación, hay otro elemento de navegación presente: los *breadcrumbs*. Se trata de **enlaces de navegación** que muestran al usuario la **ruta jerárquica** desde la página principal (inicio) hasta la sección actual (figura E.10). Son una forma más rápida de acceder a secciones del mismo módulo y ayudan al usuario a comprender mejor la estructura de la aplicación.

Inicio / Productos / Crear Nuevo Producto

Figura E.10: *Breadcrumbs* de navegación

## Página de Inicio (*Home*)

Se trata de la vista general a la que se redirige a los usuarios tras el registro o inicio de sesión. Esta página contiene **accesos rápidos** a las páginas de listado (o *dashboard*, en caso de los pedidos) de los distintos módulos de la aplicación.

La **visualización** de estos accesos rápidos depende de si el usuario autenticado tiene o no **permisos** para visualizar listados de cada módulo. De hecho, un usuario que acabe de registrarse **no visualizará ningún módulo**. En su lugar, se le informará de que debe contactar con un administrador para que se le asignen los permisos apropiados (figura E.11).

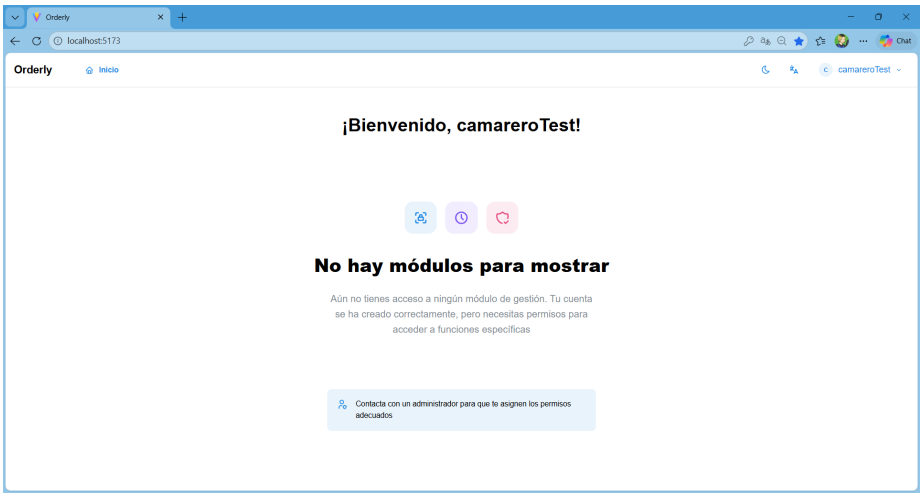


Figura E.11: Página de inicio tras registro, sin módulos de gestión

La vista general para un usuario con todos los permisos se visualiza en la figura E.12. A continuación, se detallarán las **páginas de gestión** asociadas a cada módulo y los comportamientos que albergan, siguiendo el orden establecido en los menús de navegación.

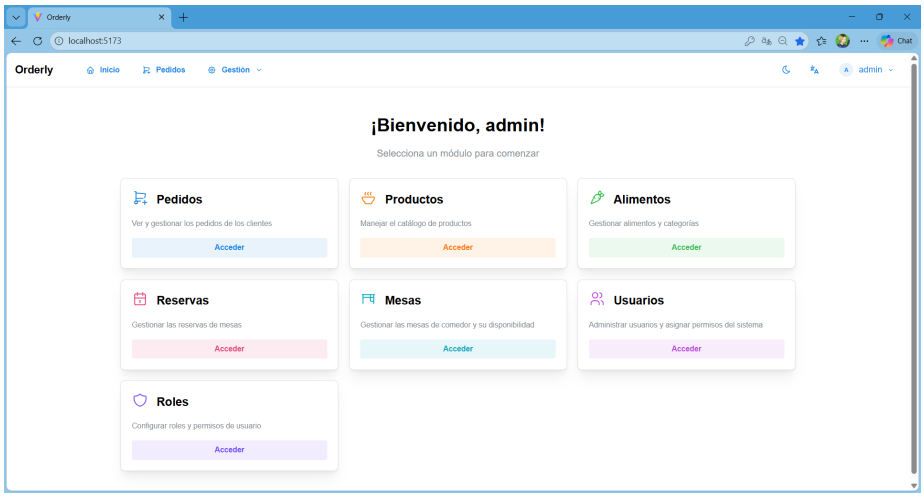


Figura E.12: Página de inicio con todos los módulos de gestión

## Gestión de pedidos

La gestión de pedidos incluye la creación de distintos tipos de comandas (barra o comedor), la modificación de los ítems que los forman y el progreso en el ciclo de vida de un pedido, mediante la modificación de su estado.

El módulo de pedidos cuenta con páginas propias y **diferentes** a las del resto de módulos. Mientras que, generalmente, los módulos disponen de páginas de creación, edición y listado, los pedidos gestionan todo su ciclo de vida desde un *dashboard* (panel) centralizado. Por motivos de **auditoría**, también existe un historial de pedidos, en el que no estará habilitada la edición ni eliminación de pedidos registrados.

### Dashboard (panel)

En esta página (figura E.13) se centralizan todas las operaciones sobre pedidos. Se divide en **dos secciones**: el listado de pedidos pendientes y el listado de productos ofertados.

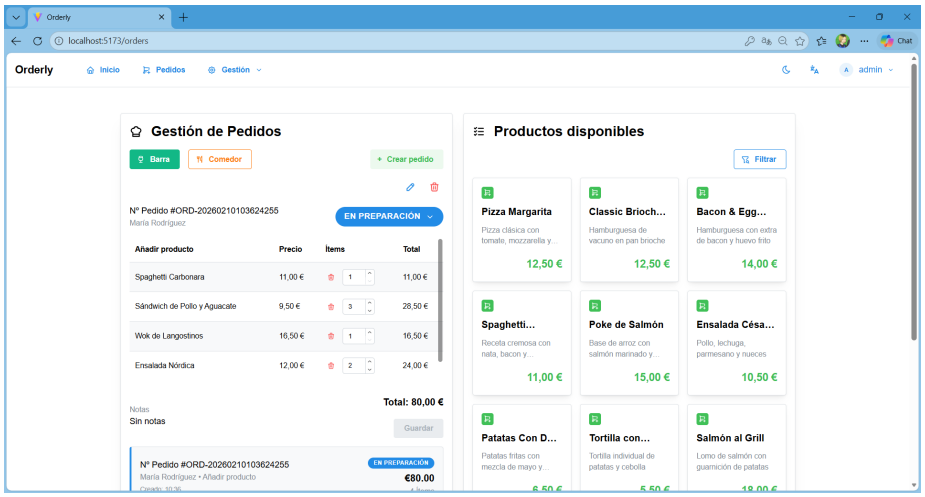


Figura E.13: *Dashboard* de pedidos

La **sección de pedidos pendientes** (izquierda) cuenta con **dos vistas** disponibles: pedidos de **barra** y pedidos de **comedor**. Pulsando los botones asociados, se puede alternar entre visualizar los pedidos de un tipo u otro. El botón de *Crear pedido* también se ve afectado por la vista en la que se sitúe el usuario (figura E.14). Es decir, si se quiere **crear**, por ejemplo, un pedido de comedor, el usuario deberá situarse en la vista *Comedor* antes de pulsar el botón de creación.

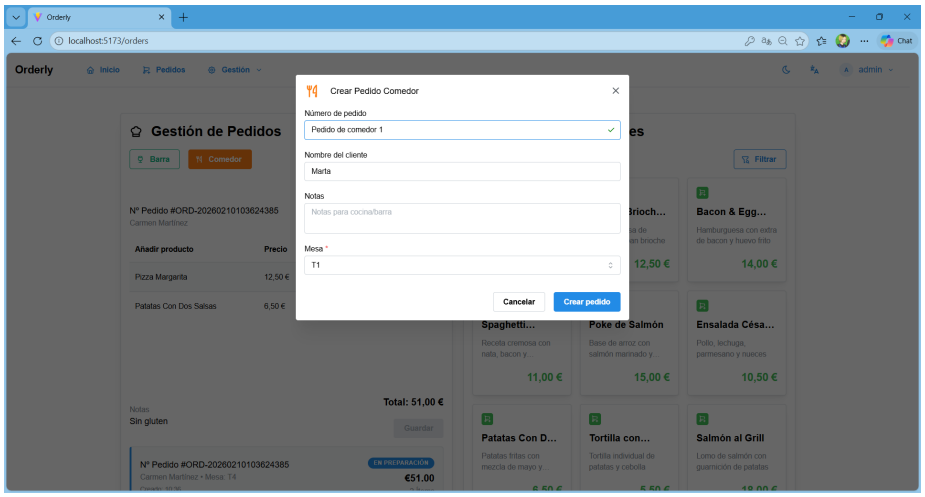


Figura E.14: Modal de creación de un pedido de comedor

En dicha sección, se visualiza una tabla (figura E.15) que muestra la información asociada al **pedido actual** (el seleccionado en ese momento) y permite **editar** su información asociada (número de pedido, nombre del cliente, notas) o **eliminarlo**.

Para **añadir ítems a un producto**, basta con hacer *click* en el producto que se quiera agregar, buscándolo en el listado de la **sección de productos ofertados** (derecha) del *dashboard*.

- Una vez añadido, se puede **modificar la cantidad unitaria** de dicho producto dentro del pedido o **eliminar** el ítem del pedido.
- El botón *Guardar* se **habilita** cuando las cantidades de los ítems se modifican, y **debe pulsarse** para actualizar el precio total y conservar la información del pedido. De lo contrario, dichas modificaciones se perderán al cambiar el pedido seleccionado o salir de la página.

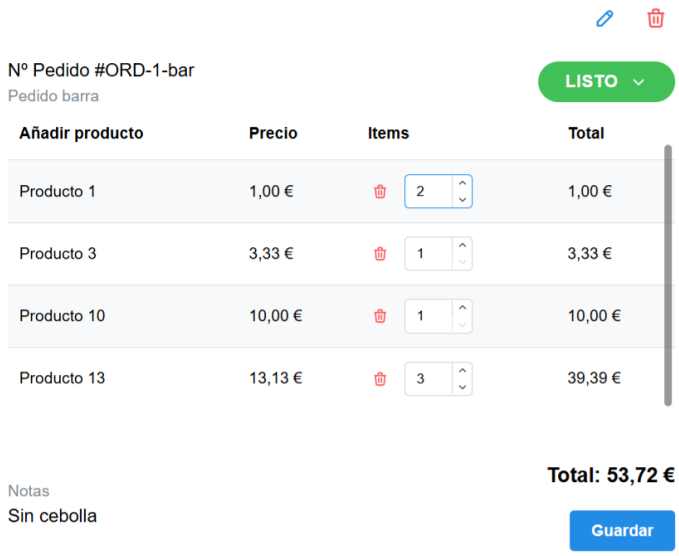


Figura E.15: Tabla con detalles del pedido seleccionado

Desde esta tabla también puede **modificarse el estado** del pedido seleccionado, desde el botón que lo contiene. Si se pulsa dicho botón, se despliega un menú con los estados posibles. Es **importante** recalcar que desde el *dashboard* **solo** se gestionan pedidos con estados *"pendiente"*, *"en progreso"*, *"listo"* o *"servido"*. Una vez se marque un pedido como *"pagado"* o *"cancelado"*, el pedido **desaparecerá** de la lista de pedidos del *dashboard* y solo se tendrá acceso a su información desde el *Historial de pedidos*.

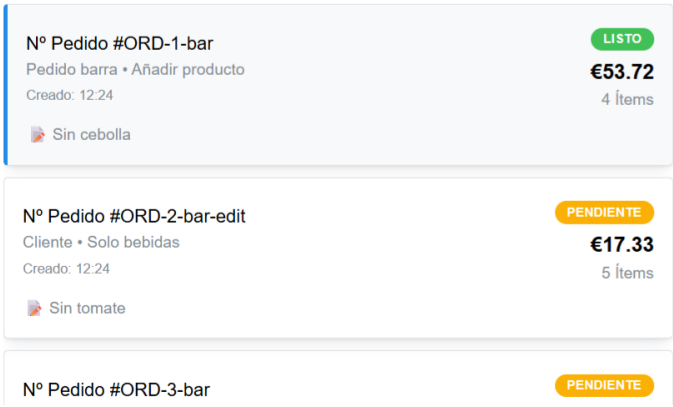


Figura E.16: Lista de pedidos pendientes

Respecto a dicha lista de pedidos (figura E.16), desde aquí pueden visualizarse todos los pedidos activos de la aplicación, y su visualización también depende de la vista (bar o comedor) en la que se encuentre el usuario. Pulsando sobre uno de los elementos de la lista, se establece dicho elemento como el "pedido actual" y pasa a visualizarse en la tabla descrita anteriormente.

Finalmente, ya se ha mencionado la **sección de productos ofertados** al explicar cómo añadir ítems a un producto. Se trata de un listado paginado con **todos** los productos registrados en la aplicación. Dicho listado puede filtrarse para **excluir ciertos alérgenos**. Tan solo hace falta hacer *click* en el botón *Filtrar* (figura E.17), seleccionar uno o varios alérgenos para excluir y pulsar en el botón *Filtrar* del cuadro emergente.

Figura E.17: Filtro para excluir alérgenos en listado de productos

Desde la barra de búsqueda se pueden **buscar** alérgenos por nombre y **deseleccionar todos** a la vez, si se quisiera dejar de filtrar.

### Historial de pedidos

En esta página se puede acceder al **registro completo** (figura E.18) de todos los pedidos realizados en la aplicación (excluyendo los eliminados desde el *dashboard*), ordenados por fecha de modificación más reciente.

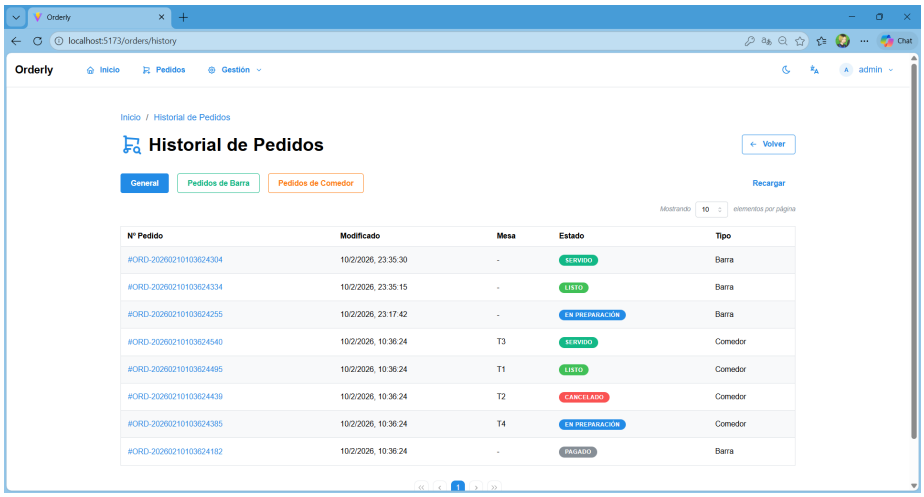


Figura E.18: Historial de pedidos

Contiene botones para visualizar todos los pedidos o aquellos de un tipo concreto. Si se pulsa sobre el **número de pedido** de alguno de los registros, se abrirá una ventana emergente con los **detalles** de dicho pedido, incluyendo una tabla con su precio, ítems y cantidades finales (figura E.19). No se podrán realizar modificaciones a ninguno de los pedidos desde esta vista.

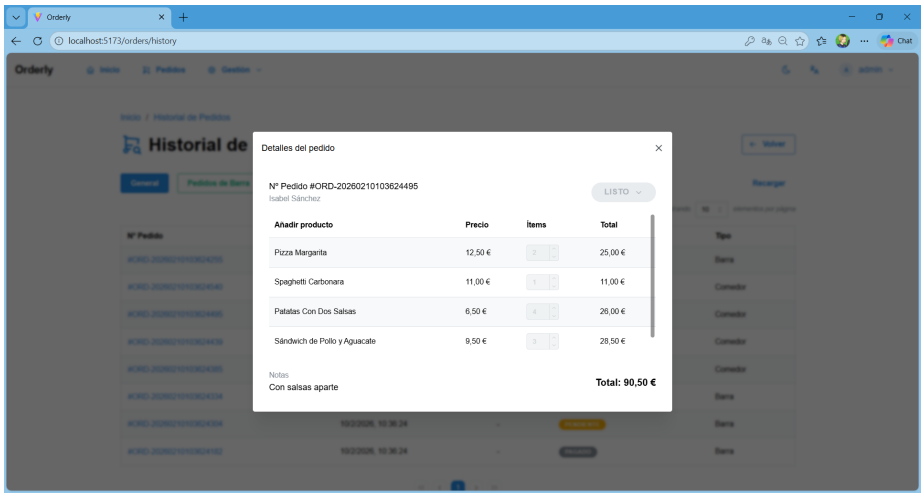


Figura E.19: Vista de detalles de un pedido desde historial

## Gestión de productos

La gestión de productos abarca la creación, edición y borrado de los platos (raciones) ofertados en el negocio de restauración. También incluye la visualización de **información nutricional** y la lista de **alérgenos** de cada producto, extraídas a partir de los ingredientes que lo forman.

### Listar productos

La página de listado (figura E.20) muestra una tabla con todos los productos almacenados en la base de datos de la aplicación, ordenados por ID. Se muestran varios campos relevantes como el precio del producto. Desde esta página se pueden realizar varias acciones como:

- Pulsar el botón de *Nuevo Producto*, que redirigirá a la página de *Crear productos*.
- Pulsar el botón de en la fila de un producto particular, que redirigirá a la página de *Editar productos*.
- Pulsar el botón de en la fila de un producto particular, que abrirá un cuadro de diálogo para *Eliminar productos*.
- Pulsar el nombre de un producto, que redirigirá a la página de *Ver detalles de un producto*.

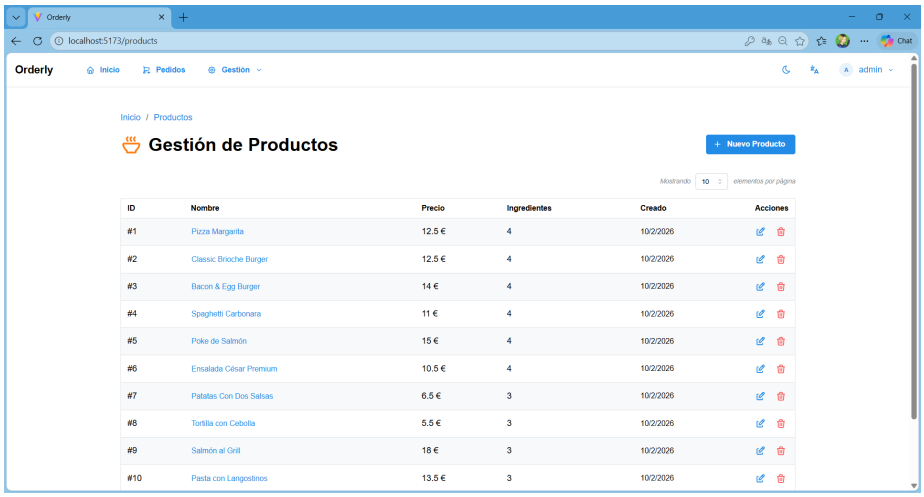


Figura E.20: Página de listado de productos

## Crear productos

La página de creación de un producto (figura E.21) contiene un formulario para rellenar la información básica asociada a dicho producto. Como mínimo, se le debe asignar un **nombre** y un **precio** al producto (puede contener decimales).

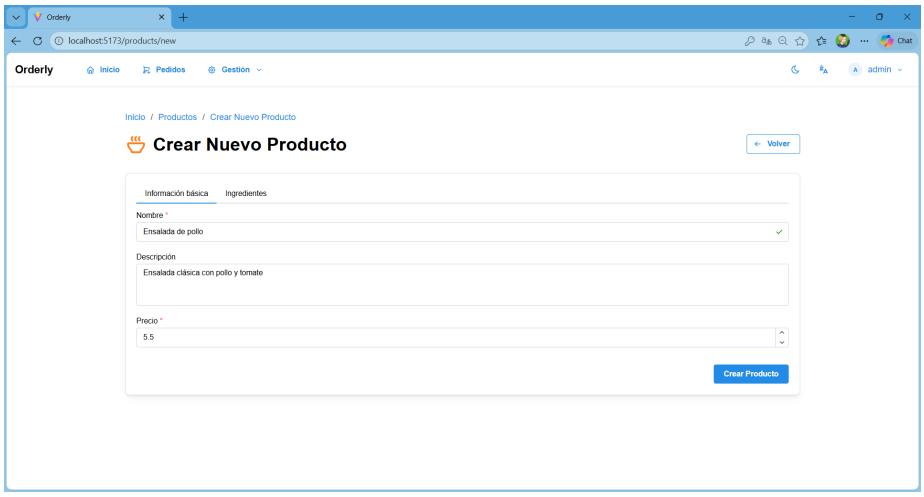


Figura E.21: Página de creación de productos

El nombre del producto introducido debe ser **único** (no debe coincidir con ninguno de los productos almacenados en la base de datos). Si se introduce

un nombre que ya está en uso, se informa al usuario y no se permite enviar el formulario hasta que se corrija.

Aparte del formulario inicial, hay otra sección dedicada para **gestionar los ingredientes** de ese producto (figura E.22). Dicha sección contiene una barra de búsqueda para **buscar por nombre** los ingredientes (alimentos) que se quieran añadir al producto.

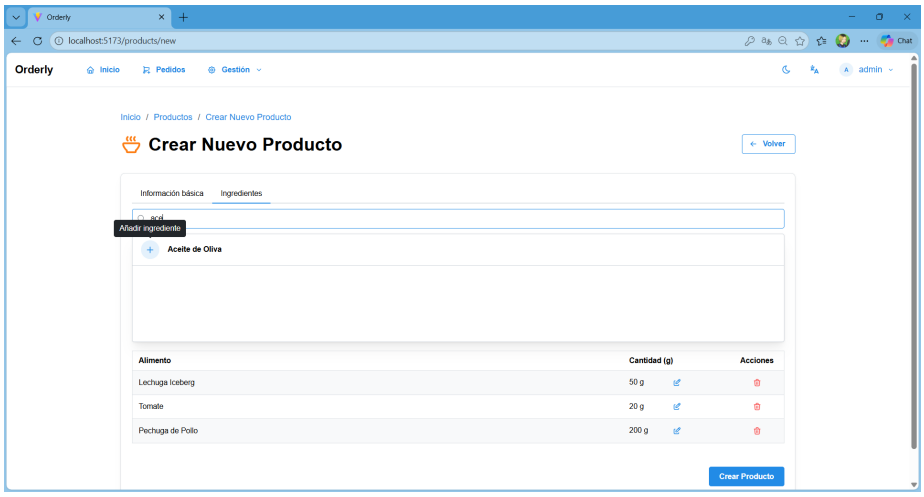


Figura E.22: Sección para la gestión de ingredientes de un producto

Una vez se añade un ingrediente, se puede **editar la cantidad** en gramos de dicho ingrediente en el producto. Esta porción del alimento determinará qué nutrientes individuales aporta el ingrediente al conjunto completo de nutrientes del producto. También se pueden **quitar ingredientes** de la tabla.

### Editar productos

La página de edición de un producto (figura E.23) permite realizar todas las acciones descritas en el apartado anterior.

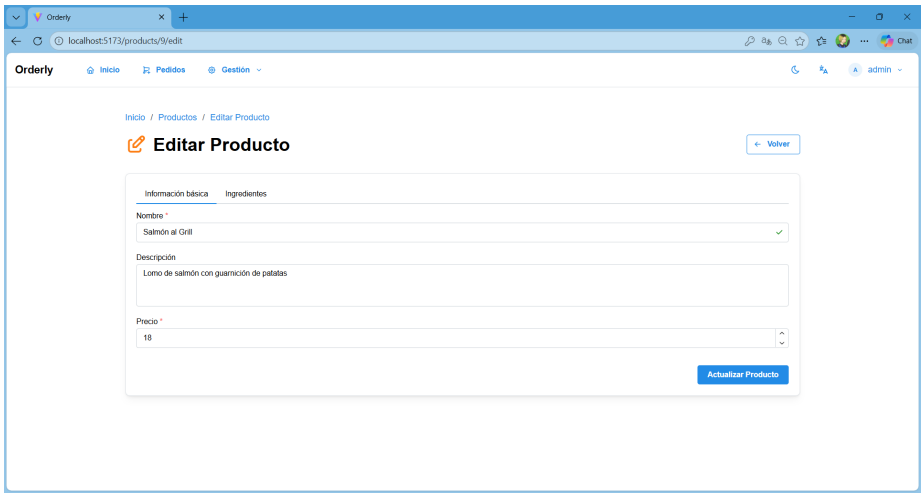


Figura E.23: Página de edición de productos

### Eliminar productos

Para eliminar un producto, basta con situarse en la pantalla de listado y pulsar el botón del producto que quiera eliminarse. Se abrirá entonces un cuadro de diálogo (figura E.24) pidiendo **confirmación** de la operación.

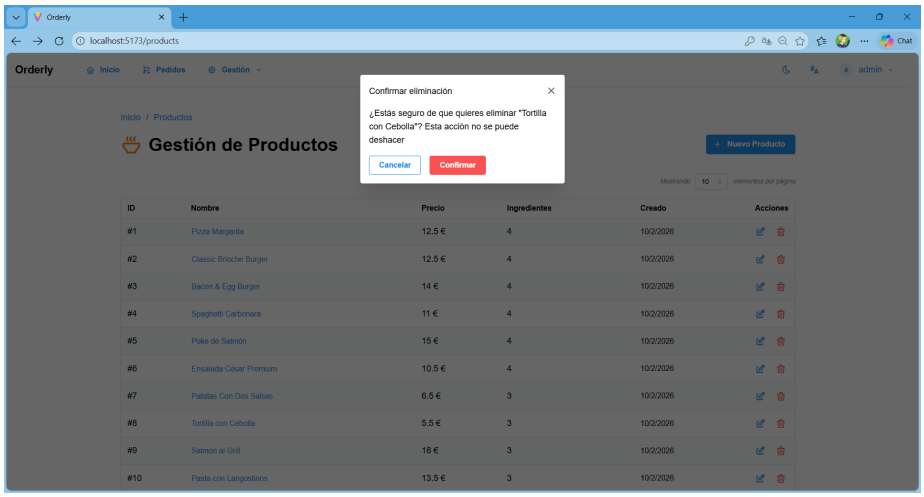


Figura E.24: Modal de eliminación de productos

Si la operación de borrado es exitosa, se notificará al usuario del éxito en el borrado y se actualizará el listado. Si no es exitosa, se informará al usuario del error mediante una notificación.

### Ver detalles de un producto

La página de información detallada de un producto (figura E.25) contiene su listado de ingredientes y una sección para ver la información nutricional total del producto. En la parte superior de la página, se visualiza la información básica del producto (nombre, descripción, precio) y una lista desplegable de **alérgenos**. Los alérgenos de un producto son la unión del conjunto de alérgenos que aportan los ingredientes de dicho producto.

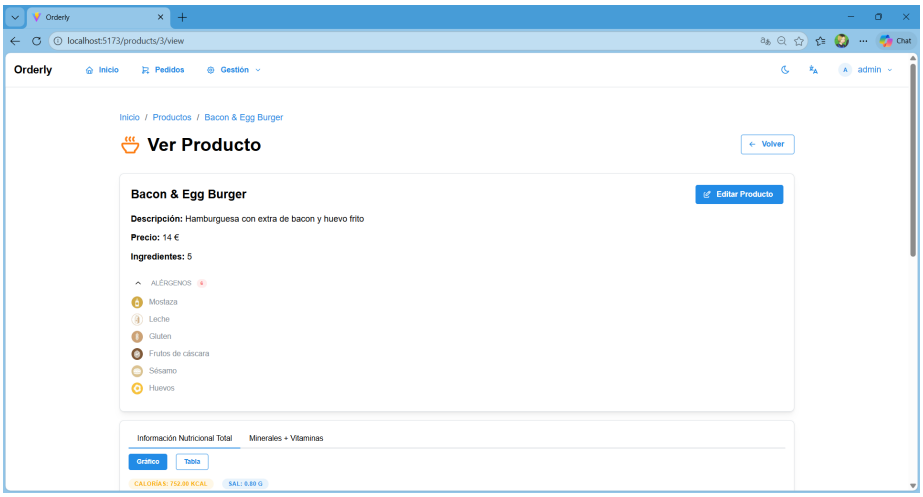


Figura E.25: Página de detalles de un producto

Figura E.26: Gráfico nutricional de un producto

La siguiente sección a visualizar presenta en su vista inicial un **gráfico** de tipo donut, con los **macronutrientes** totales del producto (figura E.26).

Se trata de un gráfico anidado, puesto que algunos nutrientes representan una **porción** de otros más generales (por ejemplo, las *grasas saturadas* son una parte de las *grasas* totales del producto). Estos datos también pueden visualizarse en formato **tabla**, pulsando el botón pertinente para alternar la vista (figura E.27).

Adicionalmente, en una sección secundaria, se muestran tablas con valores de **minerales** y **vitaminas** del producto. Tanto estas tablas como la relativa a los macronutrientes contienen una columna de valores con **ingestas diarias** de referencia.

- Si en una fila el valor real representa más del **80 %** de la ingesta de referencia, esta se sombreará de color **amarillo** suave.
- Si en una fila se **supera** el valor de referencia, esta se sombreará de color **rojo** suave.

Información Nutricional Total | Minerales + Vitaminas

Gráfico | Tabla

| Información Nutricional | Valor | Referencia |
|---|---|---|
| Calorías | 752 kcal | 2000 kcal |
| Carbohidratos | 45 g | 260 g |
| Azúcares | 8 g | 25 g |
| Proteína | 30.3 g | 60 g |
| Grasas | 72.8 g | 70 g |
| Grasas Saturadas | 17 g | 20 g |
| Fibra | 0 g | 25 g |
| Sal | 0.8 g | 5 g |

Figura E.27: Tabla nutricional de un producto con ingestas diarias de referencia

Al final de la página se visualiza una lista con todos los ingredientes del producto y sus cantidades en gramos para dicho producto (figura E.28). Por cada ingrediente se visualizan:

- Un listado desplegable de los **alérgenos** pertenecientes a dicho alimento.
- **Etiquetas** nutricionales para mostrar el ***Nutri-Score*** y **grupo NOVA** de dicho alimento. Si el alimento no dispone de dichas métricas, aparecerán en **gris**.

Figura E.28: Lista detallada de ingredientes de un producto

## Gestión de alimentos (ingredientes)

En *Orderly* denominamos "alimentos" a los **ingredientes que componen un producto** puesto a la venta. Por ejemplo, un producto *"Ensaladilla rusa"* estaría compuesto de varios alimentos: *"Patata"*, *"Pimiento"*, *"Atún"*, etc. **No es correcto** asociar *alimento=materia prima*, puesto que los alimentos pueden ser también artículos **comerciales** o **envasados**, con un código de barras y etiquetas nutricionales asociadas.

La gestión de alimentos abarca la creación, edición y borrado de los ingredientes que se usarán en los productos ofertados en el negocio.

### Listar alimentos

La página de listado (figura E.29) muestra una tabla con todos los alimentos almacenados en la base de datos de la aplicación, ordenados por ID. Se muestran varios campos relevantes como el grupo alimenticio (*"Verduras"*, *"Cereales"*). Desde esta página se pueden realizar varias acciones como:

- Pulsar el botón de *Nuevo Alimento*, que redirigirá a la página de *Crear alimentos manualmente*.
- Pulsar el botón de en la fila de un alimento particular, que redirigirá a la página de *Editar alimentos*.
- Pulsar el botón de en la fila de un alimento particular, que abrirá un cuadro de diálogo para *Eliminar alimentos*.
- Pulsar el nombre de un alimento, que abrirá una ventana emergente con **información detallad**a de dicho alimento (figura E.30), incluyendo

su lista de alérgenos y métricas nutricionales estandarizadas como el ***Nutri-Score*** y ***grupo NOVA***[1].

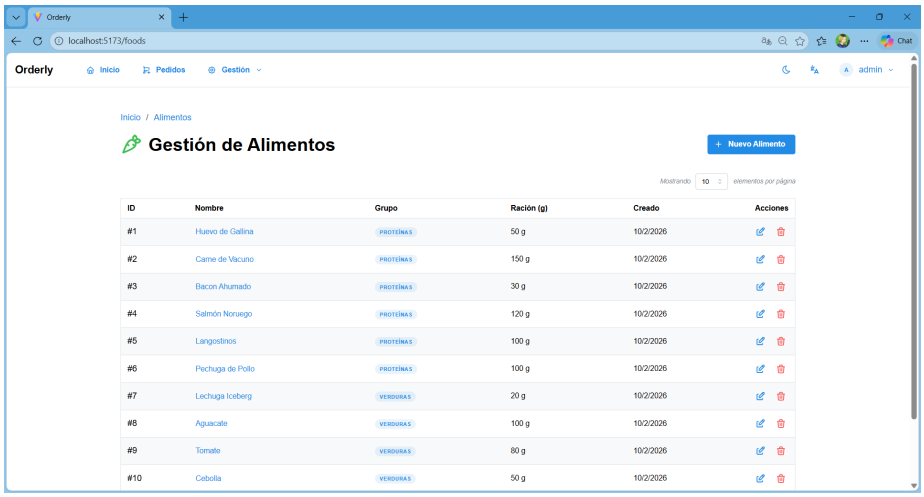


Figura E.29: Página de listado de alimentos

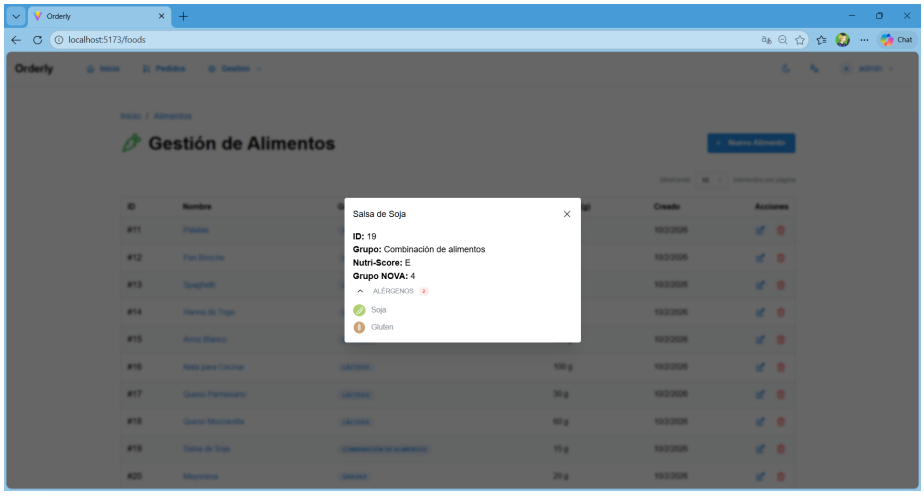


Figura E.30: Modal de detalles de un alimento

[1]Estas métricas **solo** estarán disponibles para aquellos alimentos comerciales que se hayan importado desde una **API externa**.

### Crear alimentos manualmente

Se pueden crear alimentos de forma manual o mediante una API externa. En este apartado, se detallará la creación manual de un alimento. El siguiente apartado hablará de cómo *Importar productos mediante API externa*.

La página de creación manual de un alimento (figura E.31) contiene un formulario para rellenar la información básica asociada a dicho alimento. Como mínimo, se le debe asignar un **nombre**, un **grupo** alimenticio y una **cantidad** (en gramos) de lo que supone una ración de dicho alimento.

El nombre introducido debe ser **único** (no debe coincidir con ninguno de los nombres almacenados en la base de datos). Si se introduce un nombre que ya está en uso, se informa al usuario y no se permite enviar el formulario hasta que se corrija.

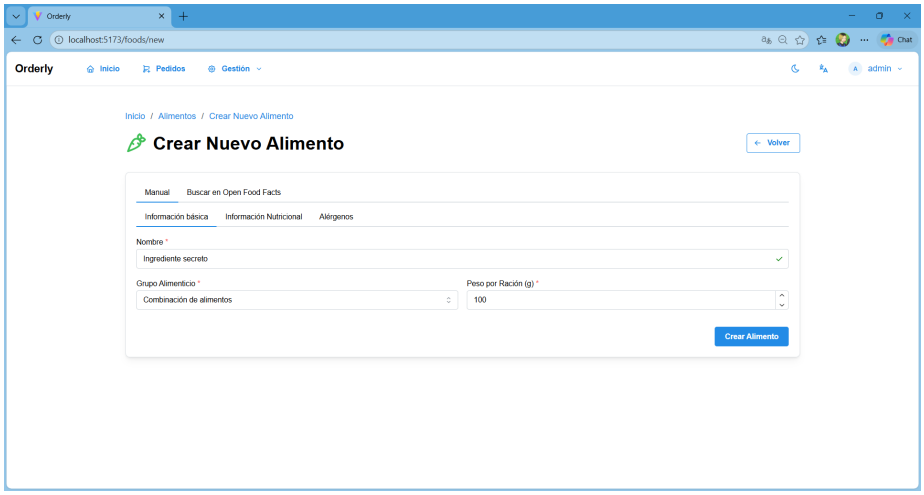


Figura E.31: Página de creación de alimentos

Aparte del formulario inicial, hay otra sección dedicada para **gestionar la información nutricional** de dicho alimento (figura E.32). En ella se pueden introducir valores para los **macronutrientes** (*"carbohidratos"*, *"proteínas"*, etc.) o las **vitaminas** y **minerales** del alimento, para la cantidad (porción) de alimento escogida. Añadir valores de información nutricional a los alimentos sirve para que se puedan obtener **informes más detallados** de los productos que los contienen y obtener gráficos nutricionales más precisos.

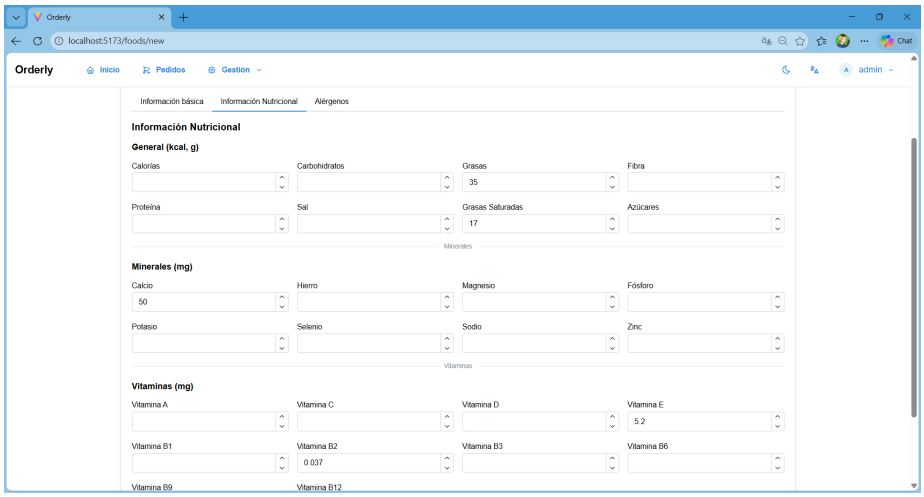


Figura E.32: Introducción de valores nutricionales para un alimento

Finalmente, el formulario de creación manual dispone de otra sección dedicada para **gestionar los alérgenos** asociados a ese alimento (figura E.33). Dicha sección contiene una lista con todos los alérgenos reconocidos en la aplicación, los cuales se corresponden con los **14 alérgenos de declaración obligatoria de la UE** [19].

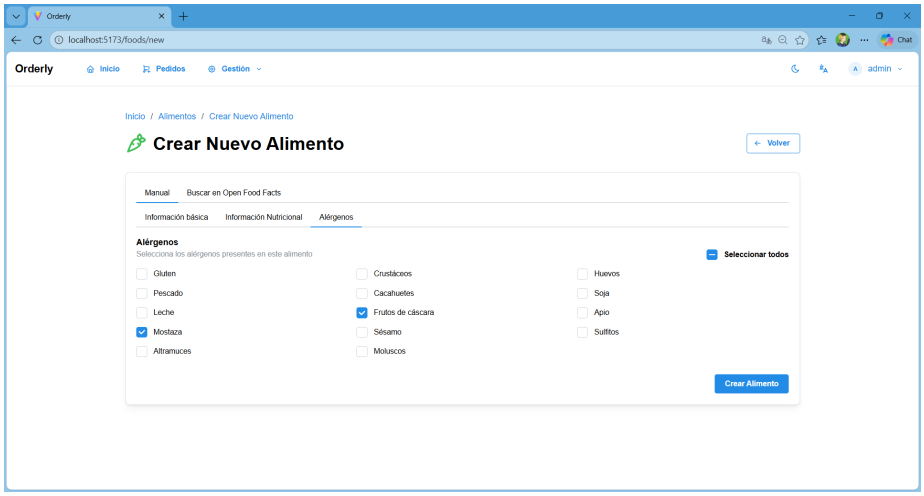


Figura E.33: Selección de alérgenos de un alimento

La selección de los alérgenos se hace mediante *checkboxes*, marcando o desmarcando las opciones. En la parte superior hay una opción para seleccionar o deseleccionar **todos** los alérgenos.

Cuando se le agregan alérgenos a un alimento, todos los productos que contengan dicho alimento **heredarán** su lista de alérgenos (y la lista de alérgenos del resto de ingredientes que forman el producto).

### Importar productos mediante API externa

Crear un alimento a partir de su **búsqueda** en una API externa es muy sencillo. Tan solo hay que situarse en la página de **creación** del alimento y acceder a la sección *Buscar en Open Food Facts*.

Se visualizará entonces una barra de búsqueda, en la que se podrá introducir el nombre o información asociada al producto. La búsqueda en *Open Food Facts* permite encontrar productos a partir de su **nombre**, pero también se pueden introducir otros aspectos como su **marca** o **etiquetas** (ej. introducir *"zumo de naranja"* no devuelve solo alimentos llamados estrictamente así, sino cualquier alimento que entre en esa **categoría**).

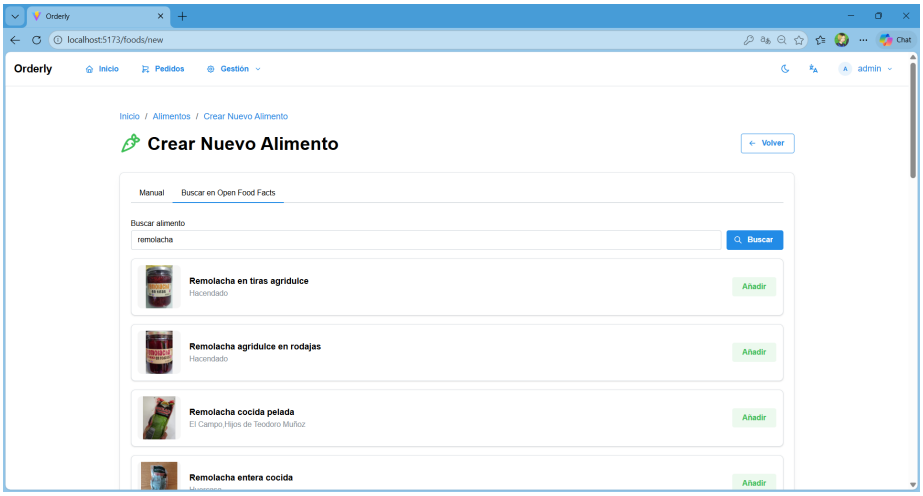


Figura E.34: Búsqueda de alimentos mediante API externa

Una vez se introduzca el texto, se debe pulsar el botón de búsqueda y esperar a los resultados. Los alimentos devueltos se visualizarán en varias páginas y dispondrán de un botón de *Añadir* (figura E.34). Crear un alimento es tan simple como **pulsar** dicho botón y esperar la notificación de **confirmación** de creación.

Al crear un alimento mediante la API externa, se importan sus valores nutricionales, lista de alérgenos, métricas nutricionales, etc. Si se quisieran hacer **modificaciones** sobre estos campos, siempre pueden realizarse desde

la página de *Editar alimentos* (exceptuando las métricas *Nutri-Score* y grupo NOVA).

### Editar alimentos

La página de edición de un alimento (figura E.35) permite realizar todas las acciones descritas en el apartado de creación manual de un alimento.

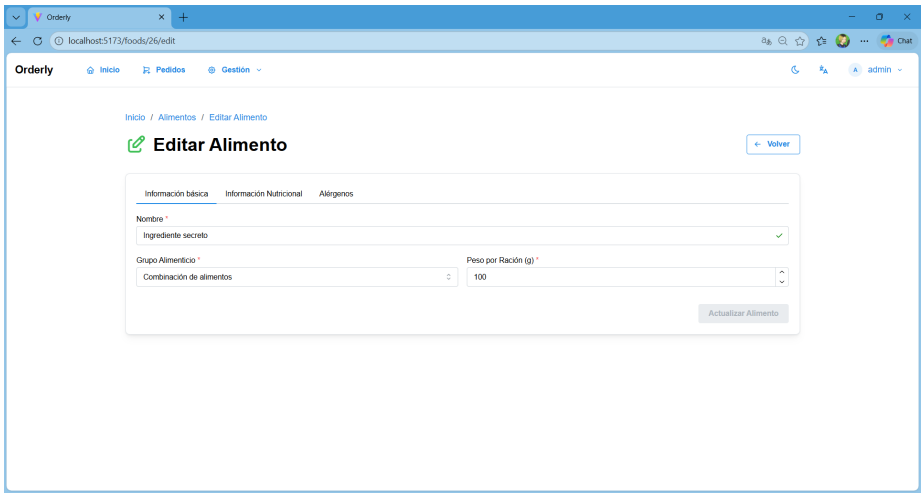


Figura E.35: Página de edición de alimentos

### Eliminar alimentos

Para eliminar un alimento, basta con situarse en la pantalla de listado y pulsar el botón del alimento que quiera eliminarse. Se abrirá entonces un cuadro de diálogo (figura E.36) pidiendo **confirmación** de la operación.

Si la operación de borrado es exitosa, se notificará al usuario del éxito en el borrado y se actualizará el listado. Si no es exitosa, se informará al usuario del error mediante una notificación.

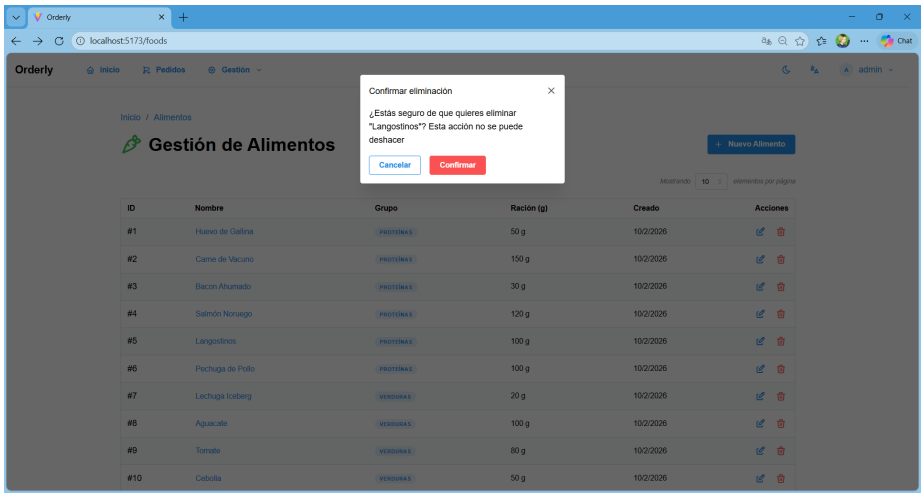


Figura E.36: Modal de eliminación de alimentos

## Gestión de reservas

La gestión de reservas abarca la creación, edición y eliminación de reservas sobre mesas de comedor, realizadas para fechas concretas.

### Listar reservas

La página de listado (figura E.37) muestra una tabla con todas las reservas registradas en la aplicación, ordenadas por ID. Se muestran varios campos relevantes como la fecha y hora de la reserva y el número de comensales. Desde esta página se pueden realizar varias acciones como:

- Pulsar el botón de *Nueva Reserva*, que redirigirá a la página de *Crear reservas*.
- Pulsar el botón de en la fila de una reserva particular, que redirigirá a la página de *Editar reservas*.
- Pulsar el botón de en la fila de una reserva particular, que abrirá un cuadro de diálogo para *Eliminar reservas*.
- Si se tienen los permisos necesarios para editar reservas, se puede pulsar en el botón de su **estado** para modificarlo (ej: *"cancelada"*, *"pagada"*).

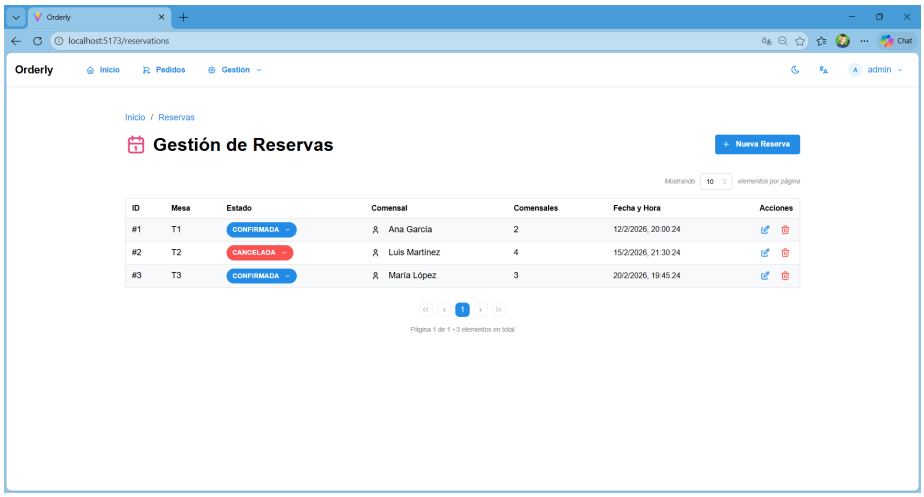


Figura E.37: Página de listado de reservas

## Crear reservas

La página de creación de una reserva (figura E.38) contiene un formulario para rellenar la información asociada a dicha reserva. Aparte del **nombre** de la mesa y la **fecha y hora** de reserva, es necesario aportar el número de **comensales** que acudirán a comer y la **información de contacto** del cliente que realizó la reserva. Si no se aporta una duración estimada, se le asignan 2 horas por defecto.

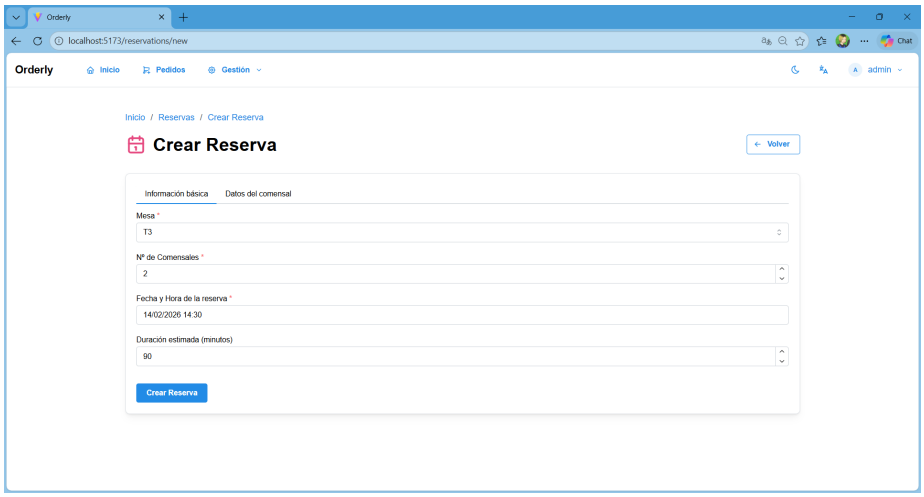


Figura E.38: Página de creación de reservas

A la hora de enviar el formulario, se comprueba si, para la fecha y hora anotadas, la mesa seleccionada está disponible. En caso de que haya **solapamiento** con otra reserva existente, se notificará del error para que el usuario cambie la fecha o la mesa asociada.

### Editar reservas

La página de edición de una reserva (figura E.39) permite realizar todas las acciones descritas en el apartado anterior.

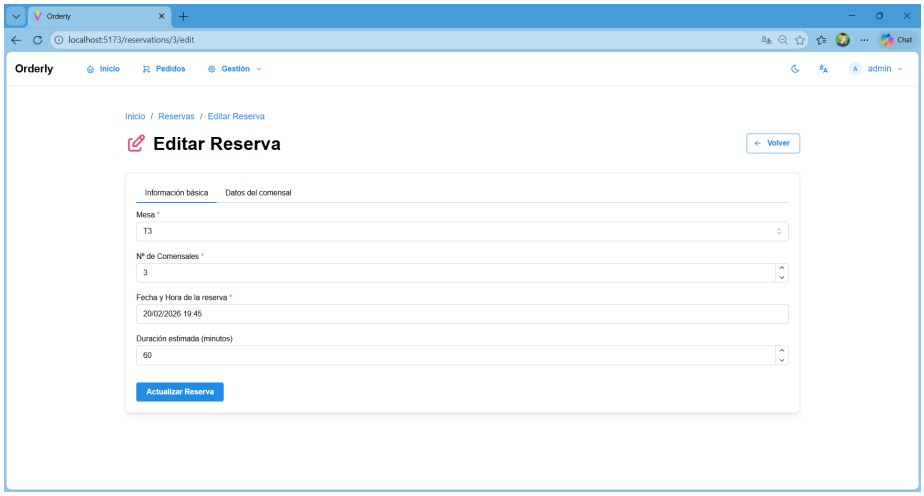


Figura E.39: Página de edición de reservas

### Eliminar reservas

Para eliminar una reserva, basta con situarse en la pantalla de listado y pulsar el botón 🗑 de la reserva que quiera eliminarse. Se abrirá entonces un cuadro de diálogo (figura E.40) pidiendo **confirmación** de la operación.

Si la operación de borrado es exitosa, se notificará al usuario del éxito en el borrado y se actualizará el listado. Si no es exitosa, se informará al usuario del error mediante una notificación.

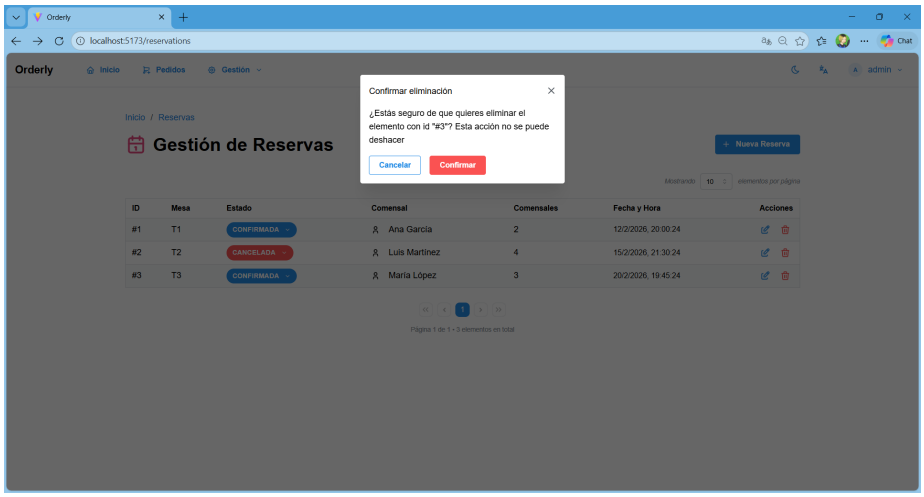


Figura E.40: Modal de eliminación de reservas

## Gestión de mesas

La gestión de mesas abarca la creación, edición y eliminación de registros que hacen referencia a las mesas del comedor, presentes físicamente en el local.

### Listar mesas

La página de listado (figura E.41) muestra una tabla con todas las mesas almacenadas en la base de datos de la aplicación, ordenadas por ID. Se muestran varios campos relevantes como la capacidad de cada mesa (número de comensales que acepta) y su estado (activa o inactiva). Desde esta página se pueden realizar varias acciones como:

- Pulsar el botón de *Nueva Mesa*, que redirigirá a la página de *Crear mesas*.
- Pulsar el botón de en la fila de una mesa particular, que redirigirá a la página de *Editar mesas*.
- Pulsar el botón de en la fila de una mesa particular, que abrirá un cuadro de diálogo para *Eliminar mesas*.
- Si se tienen los permisos necesarios para editar mesas, se puede pulsar en el botón de su **estado** para marcar una mesa como *"activa"* o "inactiva".

- Marcar una mesa como *"inactiva"* supone un **borrado lógico** de la entidad, aunque siempre puede volver a activarse.
- Al dejar inactiva una mesa, ésta ya **no aparecerá** en los listados de mesas *disponibles* en los formularios de **reservas** y **pedidos de comedor**. Es decir, esa mesa ya no podrá reservarse ni se le podrán asignar nuevos pedidos de comedor.

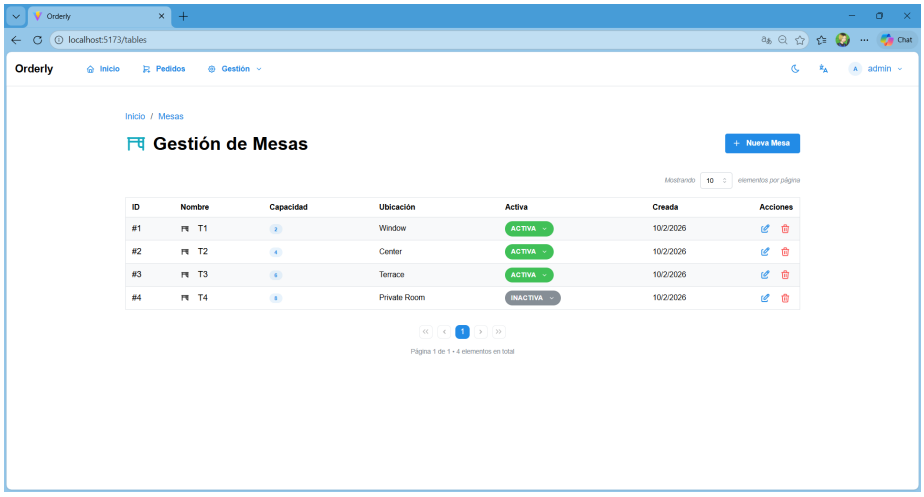


Figura E.41: Página de listado de mesas

### Crear mesas

La página de creación de una mesa (figura E.42) contiene un formulario para rellenar la información básica asociada a dicha mesa. Como mínimo, se le debe asignar un **nombre** y un número máximo de comensales (**capacidad**) que pueden sentarse en dicha mesa.

El nombre introducido debe ser **único** (no debe coincidir con ninguno de los nombres almacenados en la base de datos). Si se introduce un nombre que ya está en uso, se informa al usuario y no se permite enviar el formulario hasta que se corrija.

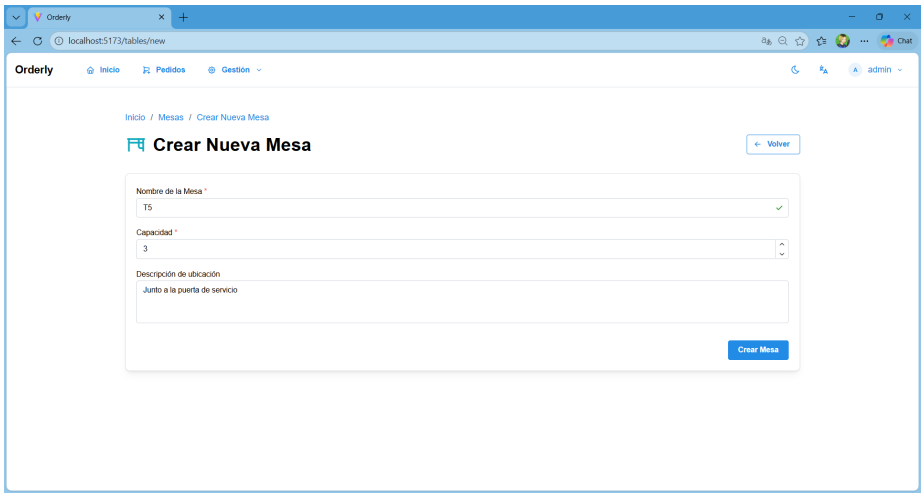


Figura E.42: Página de creación de mesas

### Editar mesas

La página de edición de una mesa (figura E.43) permite realizar todas las acciones descritas en el apartado anterior.

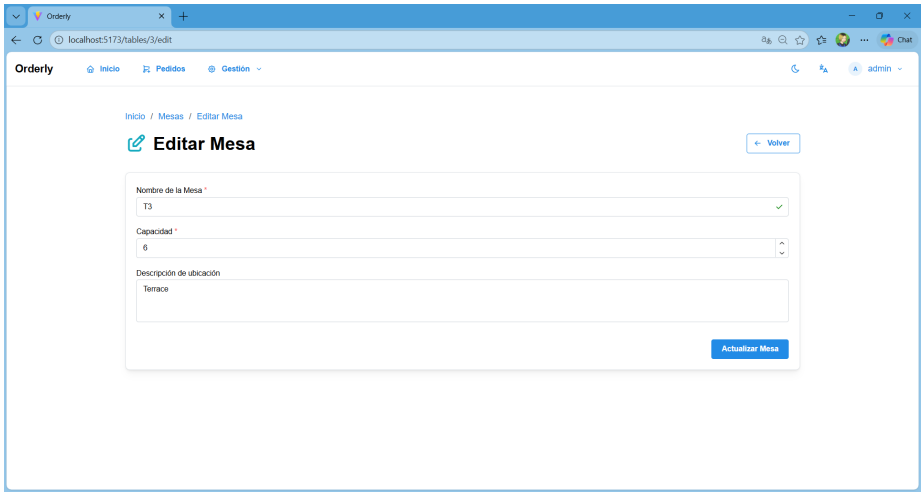


Figura E.43: Página de edición de mesas

### Eliminar mesas

**Nota:** se recomienda **no** eliminar los registros de las mesas, con el objetivo de no perder el historial de reservas asociado a ellas (de hecho, si

una mesa tiene reservas asociadas, saltará un error al intentar eliminarla). En su lugar, se recomienda **marcar** simplemente la mesa **como *"inactiva"***, de manera que no pueda ser reservada ni asignada a los pedidos de comedor, pero se mantenga su historial de reservas.

En caso de que la mesa no tenga reservas asociadas y sí se desee eliminarla, basta con situarse en la pantalla de listado y pulsar el botón de la mesa que quiera eliminarse. Se abrirá entonces un cuadro de diálogo (figura E.44) pidiendo **confirmación** de la operación.

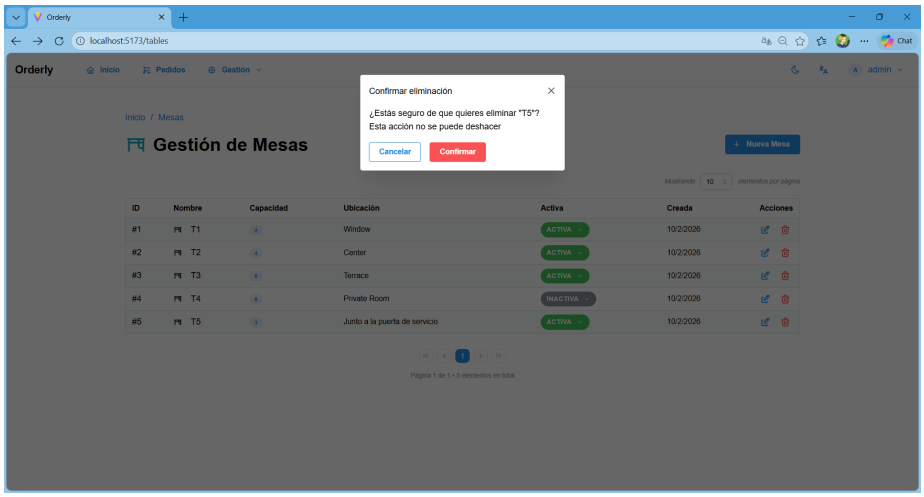


Figura E.44: Modal de eliminación de mesas

Si la operación de borrado es exitosa, se notificará al usuario del éxito en el borrado y se actualizará el listado. Si no es exitosa, se informará al usuario del error mediante una notificación.

## Gestión de usuarios

La gestión de usuarios abarca la creación, edición y eliminación de los usuarios que pueden acceder a la aplicación. También incluye la asignación de roles a dichos usuarios, controlando las acciones que pueden realizar en el sistema.

### Listar usuarios

La página de listado (figura E.45) muestra una tabla con todos los usuarios registrados en la aplicación, ordenados por ID. Se muestran varios

campos relevantes como el nombre completo del usuario o sus roles. Desde esta página se pueden realizar varias acciones como:

- Pulsar el botón de *Nuevo Usuario*, que redirigirá a la página de *Crear usuarios*.
- Pulsar el botón de en la fila de un usuario particular, que redirigirá a la página de *Editar usuarios*.
- Pulsar el botón de en la fila de un usuario particular, que abrirá un cuadro de diálogo para *Eliminar usuarios*.

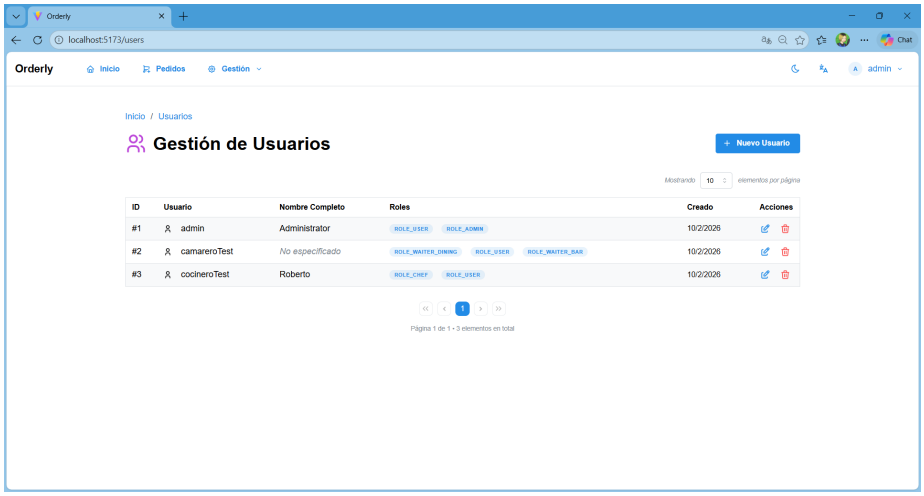


Figura E.45: Página de listado de usuarios

### Crear usuarios

La página de creación de un usuario (figura E.46) contiene un formulario para rellenar la información básica asociada a dicho usuario. Como mínimo, se le debe asignar un **nombre de usuario** y una **contraseña**, que se deberá confirmar.

El nombre de usuario introducido debe ser **único** (no debe coincidir con ninguno de los *usernames* almacenados en la base de datos). Si se introduce un nombre que ya está en uso, se informa al usuario y no se permite enviar el formulario hasta que se corrija.

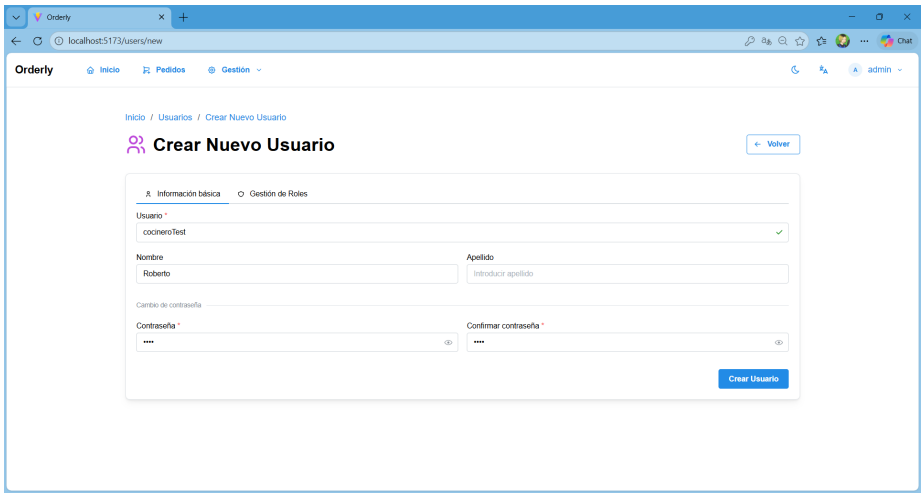


Figura E.46: Página de creación de usuarios

Aparte del formulario inicial, hay otra sección dedicada para **gestionar los roles** de ese usuario, la cual solo es **visible** para aquellos usuarios que tengan permisos de edición de roles. Dicha sección contiene dos columnas: una que contiene los roles ya asignados al usuario y otra con los roles que aún se le pueden asignar (figura E.47). Para **asignar o desasignar** un rol, basta con **pulsar** en la *píldora* (botón redondeado) que contiene dicho rol.

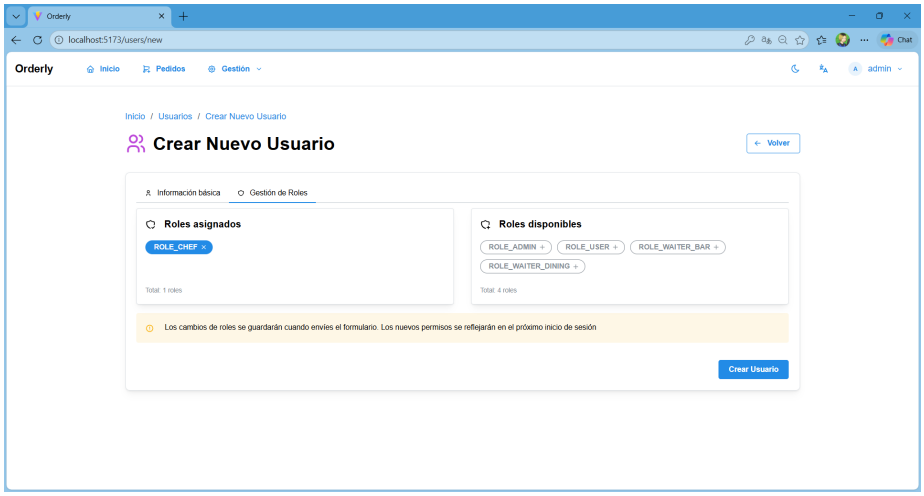


Figura E.47: Sección de asignación de roles para un usuario

Como nota adicional, aunque a un usuario se le **elimine el rol por defecto** (`ROLE_USER`), se le asignará de nuevo tras enviar el formulario. Este

rol estará **siempre presente** en todos los usuarios, puesto que representa los **permisos mínimos posibles** para un empleado.

### Editar usuarios

La página de edición de un usuario (figura E.48) permite realizar todas las acciones descritas en el apartado anterior, introduciendo algunos comportamientos nuevos:

- Para editar un usuario, no es necesario aportar su contraseña. Sin embargo, en caso de querer modificarla, se deberá confirmar la nueva contraseña.
- Los cambios que se hagan sobre los roles de un usuario, se verán reflejados la próxima vez que ese usuario inicie sesión.
- En caso de que un usuario con permisos se quiera editar a sí mismo, se le redirigirá a la página de *Editar perfil de usuario*. Si tiene permisos de edición de roles, visualizará la sección correspondiente para editarlos.

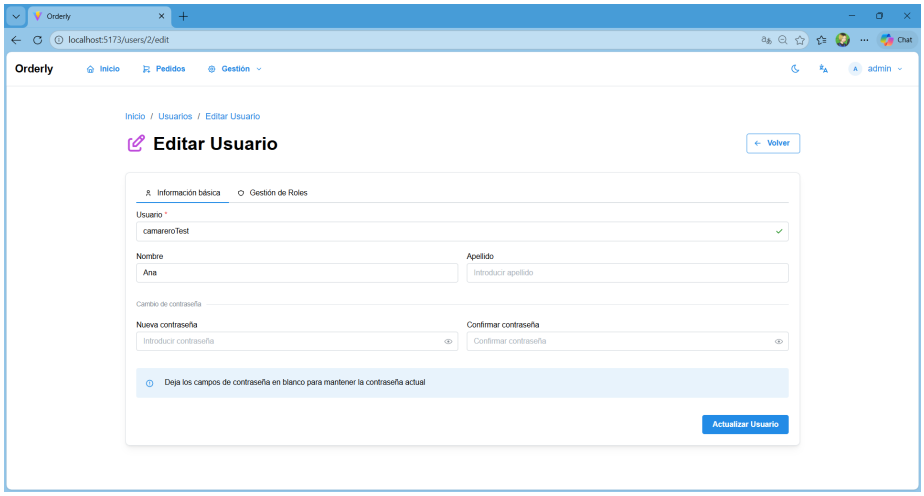


Figura E.48: Página de edición de usuarios

### Eliminar usuarios

Para eliminar un usuario, basta con situarse en la pantalla de listado y pulsar el botón del usuario que quiera eliminarse. Se abrirá entonces un cuadro de diálogo (figura E.49) pidiendo **confirmación** de la operación. Se

visualizará un mensaje de aviso en caso de que el usuario a eliminar tenga rol de administrador.

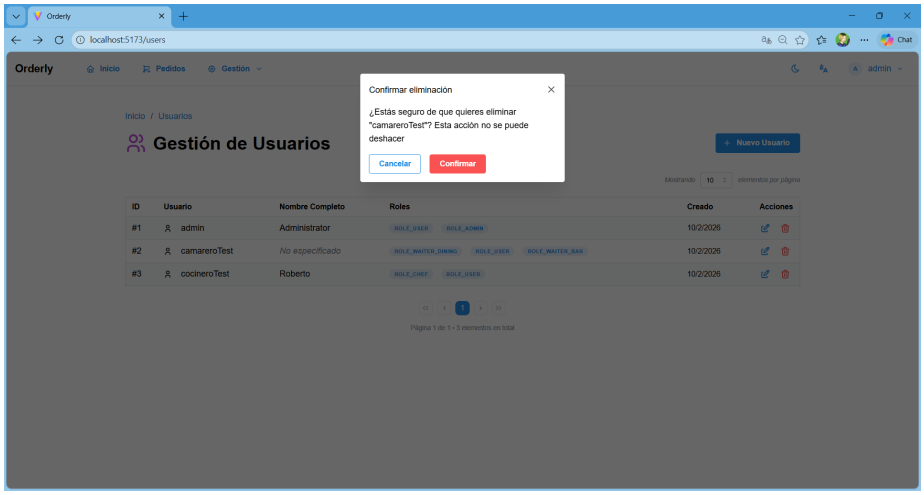


Figura E.49: Modal de eliminación de usuarios

Si la operación de borrado es exitosa, se notificará al usuario del éxito en el borrado y se actualizará el listado. Si no es exitosa, se informará al usuario del error mediante una notificación.

## Gestión de roles

La gestión de roles abarca la creación, edición y eliminación de los roles que se le pueden asignar a los usuarios del sistema. Los roles actúan como "contenedores" de permisos, de manera que engloban conjuntos de acciones que pueden ser realizadas en la aplicación.

### Listar roles

La página de listado (figura E.50) muestra una tabla con todos los roles almacenados en la base de datos de la aplicación, ordenados por ID. Se muestran varios campos relevantes como número de usuarios asignados a cada rol. Desde esta página se pueden realizar varias acciones como:

- Pulsar el botón de *Nuevo Rol*, que redirigirá a la página de *Crear roles*.
- Pulsar el botón de en la fila de un rol particular, que redirigirá a la página de *Editar roles*.

- Pulsar el botón de 🗑 en la fila de un rol particular, que abrirá un cuadro de diálogo para *Eliminar roles*.

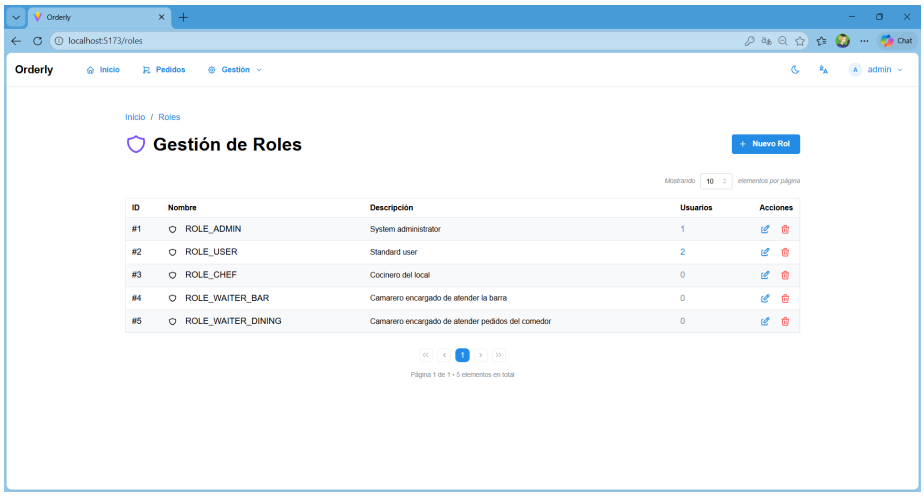


Figura E.50: Página de listado de roles

## Crear roles

La página de creación de un rol (figura E.51) contiene un formulario para rellenar la información básica asociada a dicho usuario. Como mínimo, se le debe asignar un **nombre** al rol.

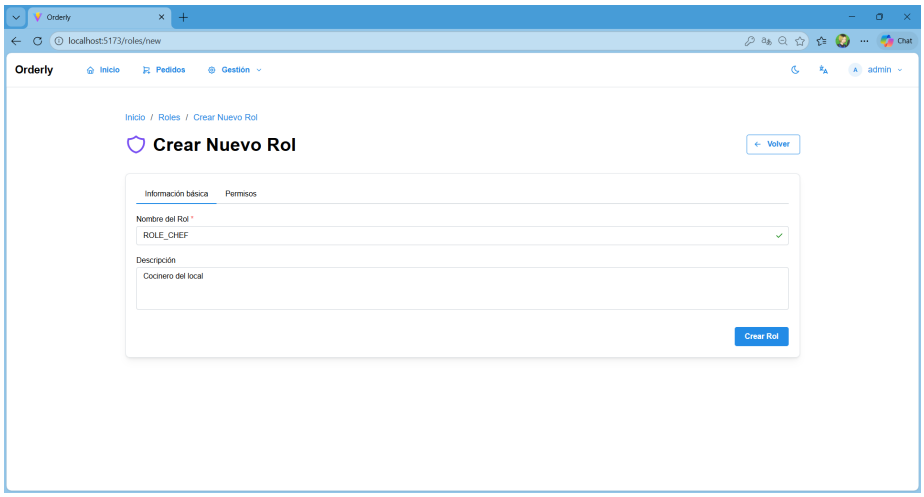


Figura E.51: Página de creación de roles

El nombre introducido debe ser **único** (no debe coincidir con ninguno de los nombres almacenados en la base de datos). Si se introduce un nombre que ya está en uso, se informa al usuario y no se permite enviar el formulario hasta que se corrija.

Aparte del formulario inicial, hay otra sección dedicada para **gestionar los permisos** asociados a ese rol (figura E.52). Dicha sección contiene una lista con todos los permisos disponibles de la aplicación, los cuales representan acciones a realizar (*"ver alimentos"*, *"editar usuarios"*, *"eliminar reservas"*, etc.).

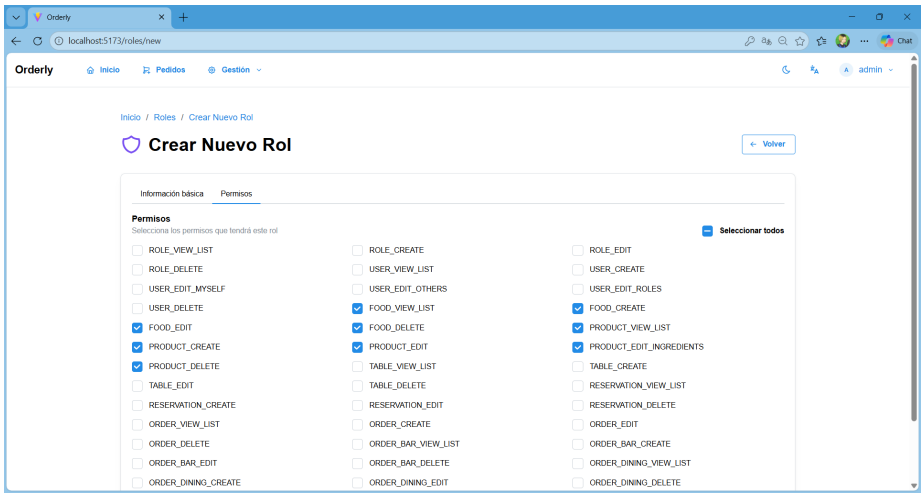


Figura E.52: Sección de asignación de permisos a un rol

La selección de los permisos se hace mediante *checkboxes*, marcando o desmarcando las opciones. En la parte superior hay una opción para seleccionar o deseleccionar **todos** los permisos.

En el apartado *Ejemplos de creación de roles* se incluyen **ejemplos de combinaciones** de permisos que se le pueden asignar a roles relacionados con el entorno de la **restauración**.

### Editar roles

La página de edición de un rol (figura E.53) permite realizar todas las acciones descritas en el apartado anterior.

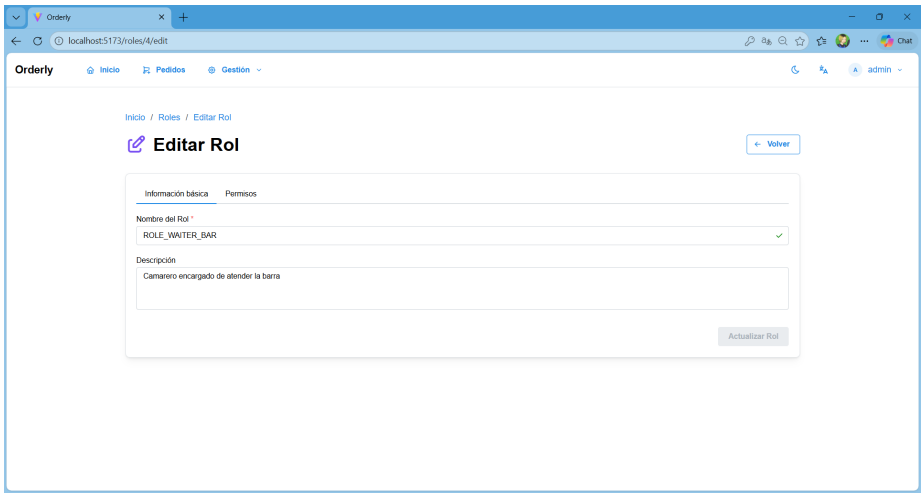


Figura E.53: Página de edición de roles

### Eliminar roles

Para eliminar un rol, basta con situarse en la pantalla de listado y pulsar el botón del rol que quiera eliminarse. Se abrirá entonces un cuadro de diálogo (figura E.54) pidiendo **confirmación** de la operación.

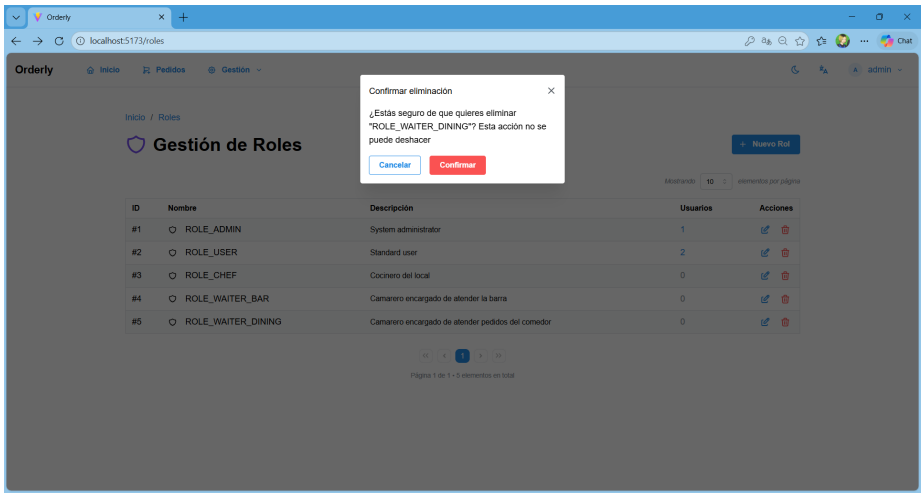


Figura E.54: Modal de eliminación de roles

Si la operación de borrado es exitosa, se notificará al usuario del éxito en el borrado y se actualizará el listado. Si no es exitosa, se informará al usuario del error mediante una notificación.

## Ejemplos de creación de roles adecuados a perfiles laborales de restauración

Como ya se ha ido introduciendo en los apartados anteriores, el **sistema de roles y permisos** de *Orderly* es muy **flexible** y permite **configuraciones personalizadas** que se ajusten a perfiles laborales más específicos. A continuación se detallan algunas combinaciones de permisos que reflejan **posibles ocupaciones laborales** en un negocio de restauración y cómo se reflejarían dichos permisos en la navegación de la aplicación.

### Cocinero

Podría interesar tener un `ROLE_CHEF` (o similar) en la aplicación, que se encargue exclusivamente del manejo de los **alimentos** y **productos**. Los usuarios con este rol podrían configurar los platos ofertados y modificar los ingredientes y sus cantidades, a medida que creen nuevas recetas o ajusten las proporciones.

Los **permisos** contenidos en este rol serían:

- `FOOD_VIEW_LIST`
- `FOOD_CREATE`
- `FOOD_EDIT`
- `FOOD_DELETE`
- `PRODUCT_VIEW_LIST`
- `PRODUCT_CREATE`
- `PRODUCT_EDIT`
- `PRODUCT_EDIT_INGREDIENTS`
- `PRODUCT_DELETE`

Los módulos de navegación disponibles, para usuarios que solo dispongan de este rol, serían los que se visualizan en la figura E.55.

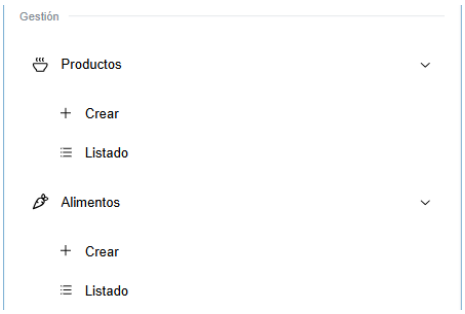


Figura E.55: Módulos de navegación para usuarios con rol `ROLE_CHEF`

### Camarero (de sala o barra)

Un usuario con `ROLE_WAITER_BAR` (o similar) sería aquel que cuenta con permisos exclusivos para la gestión de **pedidos en la barra** del local. Sus funciones principales incluirían la apertura de nuevos pedidos para la barra, la modificación de los ítems de dichos pedidos y la actualización de sus estados conforme al progreso del servicio. A pesar de solo gestionar pedidos, también necesitaría acceso al listado de **productos** ofertados, para añadirlos como ítems de un pedido.

Los **permisos** contenidos en este rol serían:

- `PRODUCT_VIEW_LIST`
- `ORDER_VIEW_LIST`
- `ORDER_BAR_VIEW_LIST`
- `ORDER_BAR_CREATE`
- `ORDER_BAR_EDIT`
- `ORDER_BAR_DELETE`

**Nota técnica:** el permiso `ORDER_VIEW_LIST` es **necesario** para la visualización de las páginas de historial y *dashboard*. Se debe asignar **siempre** que se quieran realizar operaciones sobre pedidos, aunque solo se realicen sobre un tipo de pedido. De lo contrario, dichas páginas de gestión no se visualizarían en los módulos de navegación.

Los módulos de navegación disponibles, para usuarios que solo dispongan de este rol, serían los que se visualizan en la figura E.56.

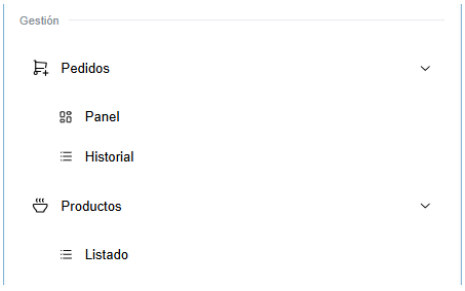


Figura E.56: Módulos de navegación para usuarios con rol `ROLE_WAITER_BAR`

En contraposición, un usuario con `ROLE_WAITER_DINING` (o similar) sería aquel que cuenta con permisos exclusivos para la gestión de **pedidos en el comedor**. Sus funciones principales incluirían la apertura de nuevos pedidos para la toma de comandas, la modificación de los ítems de dichos pedidos y la actualización de sus estados conforme al progreso del servicio en las mesas. También se le podría asignar la responsabilidad de gestionar las **reservas** de las mesas de comedor, sin entrar en la modificación de la disposición de la sala.

Los **permisos** contenidos en este rol serían:

- `PRODUCT_VIEW_LIST`
- `RESERVATION_VIEW_LIST`
- `RESERVATION_CREATE`
- `RESERVATION_EDIT`
- `RESERVATION_DELETE`
- `ORDER_VIEW_LIST`
- `ORDER_DINING_VIEW_LIST`
- `ORDER_DINING_CREATE`
- `ORDER_DINING_EDIT`
- `ORDER_DINING_DELETE`

Los módulos de navegación disponibles, para usuarios que solo dispongan de este rol, serían los que se visualizan en la figura E.57.

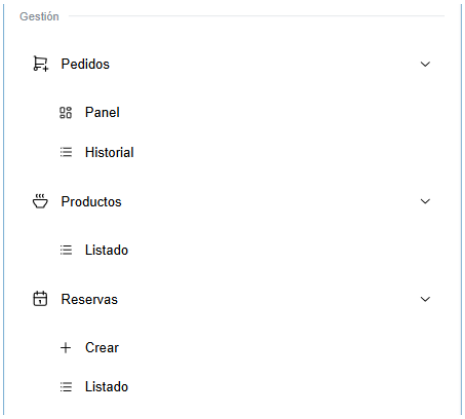


Figura E.57: Módulos de navegación para usuarios con rol `ROLE_WAITER_DINING`

También es común encontrar camareros que trabajan simultáneamente en el bar y comedor. Debido a que *Orderly* soporta la **asignación de múltiples roles** a un usuario, esto no es problema. En vez de crear un rol específico que contenga los permisos para ambos tipos de pedido, bastaría con asignarle los roles `ROLE_WAITER_BAR` y `ROLE_WAITER_DINING`. De esta manera el usuario podrá realizar acciones sobre cualquier tipo de pedido, sin necesidad de generar roles adicionales.

### Consideraciones finales sobre la coherencia de permisos

Si bien la plataforma *Orderly* cuenta con un sistema de permisos dinámico y granular que permite una personalización total, se recomienda aplicar un criterio de **coherencia funcional** al asignar permisos a los distintos roles.

La flexibilidad del sistema permite separar acciones de forma estricta, pero para garantizar una **operatividad fluida**, las capacidades otorgadas deben ser **consecuentes entre sí**. Por ejemplo, si se concede a un usuario

el permiso para crear **nuevos productos**, lo lógico, en términos de flujo de trabajo, es que también posea permisos para **editarlos**, evitando así bloqueos administrativos innecesarios. Del mismo modo, al otorgar facultades de **edición** sobre, por ejemplo, **roles de terceros**, es fundamental asegurar que el usuario cuente con **permisos de visualización** sobre el listado completo de roles. De lo contrario, se generarían conflictos en la interfaz al intentar gestionar información que el sistema, por jerarquía de acceso, no permitiría cargar.

También se debe tener en cuenta el aspecto mencionado en la creación de roles que gestionen **pedidos**, referente a la **obligatoriedad** del permiso `ORDER_VIEW_LIST` para poder **visualizar** las páginas de historial y *dashboard* de pedidos. La inclusión de estos permisos "generales" (frente a los particulares de cada tipo de pedido) se fundamenta en la creación de una capa de abstracción que centraliza la visibilidad **global** y facilita la gestión de roles transversales, garantizando la escalabilidad del sistema ante la futura incorporación de nuevos tipos de servicio sin fragmentar la lógica de acceso principal.

*Apéndice* $F$

# Anexo de sostenibilización curricular

## F.1. Introducción

Este apéndice tiene como objetivo reflexionar sobre los aspectos de sostenibilidad abordados en el Trabajo de Fin de Grado y analizar cómo el desarrollo de la aplicación web *Orderly* se alinea con los Objetivos de Desarrollo Sostenible (ODS) definidos por la Agenda 2030 de las Naciones Unidas [18]. Esta reflexión se realiza desde una perspectiva tanto técnica como social, atendiendo a las competencias de sostenibilidad adquiridas durante la formación académica y aplicadas de manera práctica en el proyecto. A continuación, se detallan los ODS que más se alinean con el propósito de *Orderly*.

## F.2. ODS 3: Salud y bienestar

La aplicación permite consultar **información nutricional** detallada de los alimentos, facilitando el acceso a datos que ayudan a tomar decisiones alimentarias más informadas. La visualización de indicadores como el ***Nutri-Score*** y el **grupo NOVA**, junto con tablas y gráficos de nutrientes, contribuye a promover hábitos de consumo más saludables.

Adicionalmente, la posibilidad de **filtrar alérgenos** resulta especialmente relevante para personas con intolerancias o alergias alimentarias, mejorando la seguridad alimentaria y reduciendo riesgos para la salud.

## F.3. ODS 8: Trabajo decente y crecimiento económico

*Orderly* busca **optimizar y simplificar** la gestión diaria de un local de restauración mediante la digitalización de procesos que, tradicionalmente, se realizan de forma manual o dispersa. La **centralización** de la información reduce la carga administrativa del personal, mejora la organización del trabajo y disminuye errores operativos. Esto puede traducirse en un entorno laboral más eficiente y menos estresante, favoreciendo **condiciones de trabajo más dignas y productivas**, especialmente en un sector caracterizado por ritmos elevados y alta rotación de personal.

Por otro lado, *Orderly* representa una oportunidad de **transformación digital** para pequeños negocios. Ayudar a que un **restaurante local** pueda dar el salto de las notas en papel a una gestión web robusta contribuye a su competitividad y, por tanto, a la estabilidad económica del negocio a largo plazo.

## F.4. ODS 9: Industria, innovación e infraestructura

El desarrollo de la aplicación se apoya en tecnologías modernas y ampliamente utilizadas en la industria del *software*, como *React*, *Spring Boot*, *APIs REST* y contenedores *Docker*. La arquitectura modular y el uso de estándares abiertos favorecen la **escalabilidad**, el mantenimiento y la **reutilización** del sistema. Además, la **integración de servicios externos** de datos abiertos, como *Open Food Facts*, refleja un enfoque innovador que promueve el aprovechamiento de infraestructuras digitales existentes y el **acceso abierto** a la información.

## F.5. ODS 12: Producción y consumo responsables

Aunque *Orderly* actualmente no gestiona el *stock* de productos, su labor en la digitalización de la operativa diaria tiene un impacto directo en el consumo de recursos.

El beneficio más evidente es la **transición** hacia un entorno **"sin papel"**. En muchos restaurantes, el ciclo de vida de un *ticket* de comanda es de

apenas unos minutos antes de acabar en la papelera. Multiplicando esto por cientos de servicios al año, el **ahorro de papel** que supone gestionar comandas y reservas de forma digital es significativo.

Además, al proporcionar información sobre el **grupo NOVA** (grado de procesamiento) y el perfil nutricional de los alimentos, la aplicación fomenta de forma indirecta un consumo más responsable. Un restaurante que tiene visibilidad sobre la calidad nutricional de sus materias primas está mejor posicionado para tomar **decisiones de compra más éticas y saludables**, reduciendo la dependencia de productos industriales que suelen generar más **residuos de envasado** y procesos productivos menos sostenibles.

## F.6. ODS 13: Acción por el clima

Desde una perspectiva técnica y medioambiental, el uso de tecnologías de virtualización mediante contenedores *Docker* contribuye a una **gestión más eficiente** de los **recursos computacionales**, alineándose de forma transversal con este objetivo.

La contenerización permite un despliegue más ligero y controlado de la aplicación, optimizando el uso de infraestructuras y **reduciendo el consumo innecesario** de recursos en comparación con soluciones menos eficientes. Aunque el impacto ambiental directo es limitado en el contexto de un TFG, el enfoque adoptado refleja una conciencia sobre la importancia de la eficiencia tecnológica.

## F.7. Conclusiones

Aunque *Orderly* es principalmente una aplicación de gestión para el sector de la restauración, su diseño y funcionalidades permiten alinearla con diversos Objetivos de Desarrollo Sostenible, especialmente aquellos relacionados con la salud, el bienestar, el trabajo digno, la innovación tecnológica y el consumo responsable.

El desarrollo de este trabajo demuestra que es posible integrar criterios de sostenibilidad en proyectos tecnológicos de alcance moderado, reforzando la idea de que el desarrollo de *software* puede y debe contribuir de forma positiva a la sociedad.

# Bibliografía


[1] 4geeks. Contexto de Autenticación con React: Gestionando el Estado del Usuario en Componentes. https://4geeks.com/es/lesson/contexto-de-autenticacion-con-react, 2026. [Internet; Accedido 02-02-2026].

[2] Docker. Docker Compose. https://docs.docker.com/compose/, 2025. [Internet; Accedido 27-12-2025].

[3] Docker. Docker Desktop. https://docs.docker.com/desktop/, 2026. [Internet; Accedido 03-02-2026].

[4] Docker. Multi-stage builds. https://docs.docker.com/get-started/docker-concepts/building-images/multi-stage-builds/, 2026. [Internet; Accedido 05-02-2026].

[5] Git Documentation. Git. https://git-scm.com/docs/git, 2026. [Internet; Accedido 03-02-2026].

[6] GeeksforGeeks. Domain-Driven Design (DDD). https://www.geeksforgeeks.org/system-design/domain-driven-design-ddd/, 2025. [Internet; Accedido 26-12-2025].

[7] Glassdoor. Sueldo: Programador Junior (Febrero, 2026). https://www.glassdoor.es/Sueldos/programador-junior-sueldo-SRCH_KO0,18.htm, 2026. [Internet; Accedido 05-02-2026].

[8] Hetzner. Hetzner introduces new shared vCPU cloud servers. https://www.hetzner.com/pressroom/new-cx-plans/, 2024. [Internet; Accedido 05-02-2026].

[9] Microsoft Learn. Troubleshoot Docker client errors on Windows - Visual Studio. https://learn.microsoft.com/en-us/troubleshoot/developer/visualstudio/ide/troubleshooting-docker-errors, 2025. [Internet; Accedido 09-02-2026].

[10] Gregg Lindemulder. What Is Role-Based Access Control (RBAC)? https://www.ibm.com/think/topics/rbac, 2026. [Internet; Accedido 05-02-2026].

[11] Luis Llamas. How to Use the Context API in React. https://www.luisllamas.es/en/react-context-api/a, 2026. [Internet; Accedido 02-02-2026].

[12] Nutritionix. Nutritionix. https://www.nutritionix.com/, 2025. [Internet; Accedido 29-12-2025].

[13] Stack Overflow. After reverse Engineer MySQL Workbench change 1:1 relationship to 1:n. https://stackoverflow.com/questions/33417257/after-reverse-engineer-mysql-workbench-change-11-relationship-to-1n, 2019. [Internet; Accedido 01-02-2026].

[14] Manh Phan. The mechanism of Spring security. https://ducmanhphan.github.io/2019-02-09-The-mechanism-of-spring-security/, 2019. [Internet; Accedido 01-02-2026].

[15] React Router. Hot Module Replacement. https://reactrouter.com/explanation/hot-module-replacement, 2026. [Internet; Accedido 09-02-2026].

[16] Spring. Spring Data JPA. https://spring.io/projects/spring-data-jpa, 2025. [Internet; Accedido 26-12-2025].

[17] Steve S. (steves801). Mounts denied: The path is not shared to the host; when it is. https://forums.docker.com/t/mounts-denied-the-path-is-not-shared-to-the-host-when-it-is/142364, 2024. [Internet; Accedido 09-02-2026].

[18] Naciones Unidas. Objetivos y metas de desarrollo sostenible. https://www.un.org/sustainabledevelopment/es/sustainable-development-goals/, 2015. [Internet; Accedido 05-02-2026].

[19] 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.

[20] Vite. Introducción. https://es.vite.dev/guide/, 2025. [Internet; Accedido 27-12-2025].

[21] 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.