



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**NutriMenu
Documentación Técnica**



Presentado por Álvaro Manjón Vara
en Universidad de Burgos — 15 de febrero
de 2024

Tutores: Raúl Marticorena Sánchez y Antonio
Jesús Canepa Oneto

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	v
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	11
Apéndice B Especificación de Requisitos	17
B.1. Introducción	17
B.2. Objetivos generales	17
B.3. Catálogo de requisitos	18
B.4. Especificación de requisitos	18
Apéndice C Especificación de diseño	19
C.1. Introducción	19
C.2. Diseño de datos	19
C.3. Diseño procedimental	23
C.4. Diseño arquitectónico	58
Apéndice D Documentación técnica de programación	63
D.1. Introducción	63
D.2. Estructura de directorios	63
D.3. Manual del programador	65

D.4. Compilación, despliegue y ejecución del proyecto	75
D.5. Pruebas del sistema	86
Apéndice E Documentación de usuario	87
E.1. Introducción	87
E.2. Requisitos de usuarios	87
E.3. Instalación	87
E.4. Manual del usuario	87
Bibliografía	89

Índice de figuras

A.1. Tareas del Sprint 1	2
A.2. Tareas del Sprint 2	2
A.3. Tareas del Sprint 3	3
A.4. Tareas del Sprint 4	3
A.5. Tareas del Sprint 5	4
A.6. Tareas del Sprint 6	5
A.7. Tareas del Sprint 7	5
A.8. Tareas del Sprint 8	6
A.9. Tareas del Sprint 9	7
A.10.Tareas del Sprint 10	7
A.11.Tareas del Sprint 11	8
A.12.Tareas del Sprint 12	8
A.13.Tareas del Sprint 13	9
A.14.Tareas del Sprint 14	10
A.15.Tareas del Sprint 15	10
A.16.Cálculos estimados de una instancia Compute Engine	13
A.17.Información detallada sobre la licencia escogida para el proyecto	15
 C.1. Diagrama relacional del modelo de datos	22
C.2. Diagrama de inicio de sesión	23
C.3. Diagrama del proceso de contraseña olvidada	24
C.4. Diagrama de creación de una empresa o alimento a mano	25
C.5. Diagrama de creación de un local o usuario	26
C.6. Diagrama de creación de un alimento con Nutritionix	27
C.7. Diagrama de creación de un plato	28
C.8. Diagrama de creación de un menú	29
C.9. Diagrama de edición de un elemento	30
C.10.Diagrama de borrado de un elemento	31

C.11.Arquitectura de un proyecto Spring Boot [14]	59
C.12.Diagrama del patrón Contenedor/Presentacional en React [1]	60
C.13.Diagrama de la perforación de accesorios [16]	60
C.14.Diagrama del patrón Proveedor en React [16]	61
C.15.Diagrama de la infraestructura en Docker Compose	62
D.1. Descarga del repositorio del proyecto	64
D.2. Listado de todas las configuraciones de ejecución disponibles	72
D.3. Selección del tipo de configuración	72
D.4. Creación de la configuración del debugger remoto	73
D.5. Creación de un módulo para un servicio	74
D.6. Debug del servicio	75
D.7. Página de descarga de Docker Desktop	76
D.8. Instalación de Docker Desktop en Mac	76
D.9. Ventana principal de Docker Desktop	77
D.10.Creación de las imágenes de los contenedores Docker	80
D.11.Despliegue de los contenedores Docker	81
D.12.Ejecución multi-contenedor de la aplicación	82
D.13.Logs de la ejecución de un contenedor en Docker Desktop	82
D.14.Parada de los contenedores en ejecución	83
D.15.Contenedores parados pero no borrados en Docker Desktop	83
D.16.Parada y borrado de los contenedores en ejecución	84
D.17.Contenedores de Docker totalmente eliminados	84
D.18.Persistencia del volumen de la base de datos	85
D.19.Parada y borrado de contenedores y volúmenes existentes	85
D.20.Volumen de la base de datos totalmente eliminado	86

Índice de tablas

A.1. Herramientas y tecnologías usadas y sus licencias	16
C.1. Endpoint que devuelve empresas existentes	33
C.2. Endpoint que permite añadir una empresa	34
C.3. Endpoint que permite actualizar la información de una empresa	35
C.4. Endpoint que permite borrar una empresa	36
C.5. Endpoint que devuelve usuarios existentes	37
C.6. Endpoint que permite añadir un usuario	38
C.7. Endpoint que permite comprobar un inicio de sesión	39
C.8. Endpoint que permite actualizar la información de un usuario .	40
C.9. Endpoint que permite borrar un usuario	41
C.10. Endpoint que devuelve locales existentes	42
C.11. Endpoint que permite añadir un local	43
C.12. Endpoint que permite actualizar la información de un local .	44
C.13. Endpoint que permite eliminar un local	45
C.14. Endpoint que devuelve menús existentes	46
C.15. Endpoint que permite añadir menús	47
C.16. Endpoint que permite actualizar la información de un menú .	48
C.17. Endpoint que permite eliminar un menú	49
C.18. Endpoint que devuelve platos existentes	50
C.19. Endpoint que permite la creación de un plato	51
C.20. Endpoint que permite modificar un plato	52
C.21. Endpoint que permite borrar un plato	53
C.22. Endpoint que devuelve alimentos existentes	54
C.23. Endpoint que permite añadir un alimento	55
C.24. Endpoint que permite actualizar la información de un alimento	56
C.25. Endpoint que permite eliminar un alimento	57

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apartado se va a comenzar hablando sobre cómo se ha planificado el proyecto en el tiempo, cómo se han organizado las tareas en el tiempo y cuánto han durado finalmente.

Después se va a proceder a hablar sobre los distintos costes que conforman el proyecto, y cuánta sería la cantidad estimada a pagar en caso de que la aplicación estuviese funcionando durante un año.

Para acabar, se va a hablar de las distintas implicaciones legales que conllevan el uso de las tecnologías integradas en el proyecto, así como del tipo de licencia escogido para la aplicación.

Estos estudios son esenciales para entender cómo el proyecto se ha llevado a cabo y qué implicaciones tendría el ponerlo en marcha.

A.2. Planificación temporal

Sprint 1 (08/03 - 22/03)

Durante el sprint 1 del proyecto se llevó a cabo la configuración del repositorio, la asimilación de cómo funciona un TFG y qué partes lo componen, y es donde se estuvo leyendo y probando el TFG del que parte este proyecto, el TFG de Mariya Aleksandrova [13].

Durante este sprint se realizó la primera reunión con los tutores, en la que me facilitaron todos los materiales referentes al TFG anterior, y discutimos

sobre los posibles caminos a escoger para mejorar el proyecto existente. Al final se decidió comenzar por intentar dockerizar la aplicación existente.

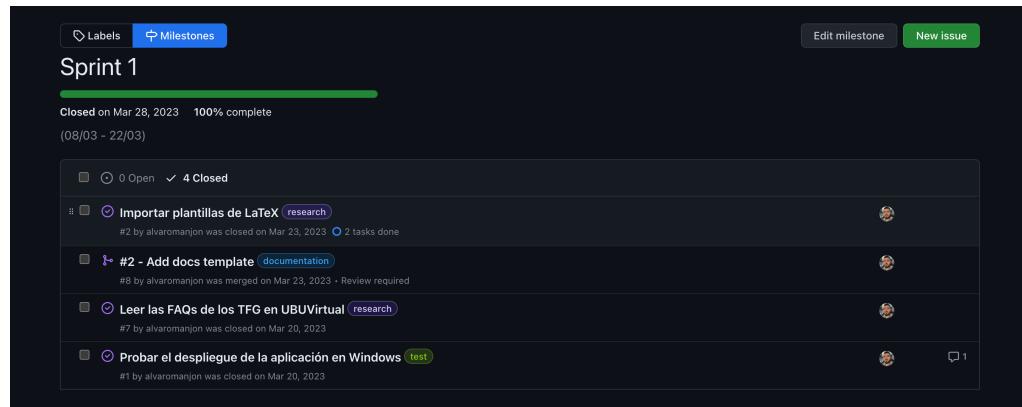


Figura A.1: Tareas del Sprint 1

Sprint 2 (23/03 - 12/04)

El segundo sprint lo dediqué a investigar y ampliar mis conocimientos de Docker y Docker Compose, ver cómo ambas tecnologías pueden funcionar juntas y cómo se podrían llegar a aplicar en este proyecto.

Durante este período intenté contenerizar las dos aplicaciones de las que partía el proyecto, pero tuve problemas a la hora de conectar las dos aplicaciones con la base de datos, así que terminé creando un proyecto nuevo de Spring Boot.

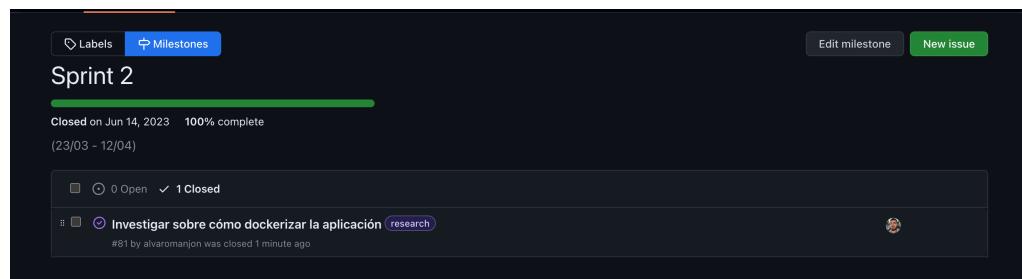


Figura A.2: Tareas del Sprint 2

Sprint 3 (13/04 - 26/04)

Durante este sprint se comenzó a plantear la idea de cambiar la fuente de datos nutricionales para pasar de BEDCA a un modelo más moderno, idealmente un modelo al que se pudiese acceder mediante el uso de una API REST. Después de darle varias vueltas, finalmente mi decisión fue usar la API de Nutritionix.

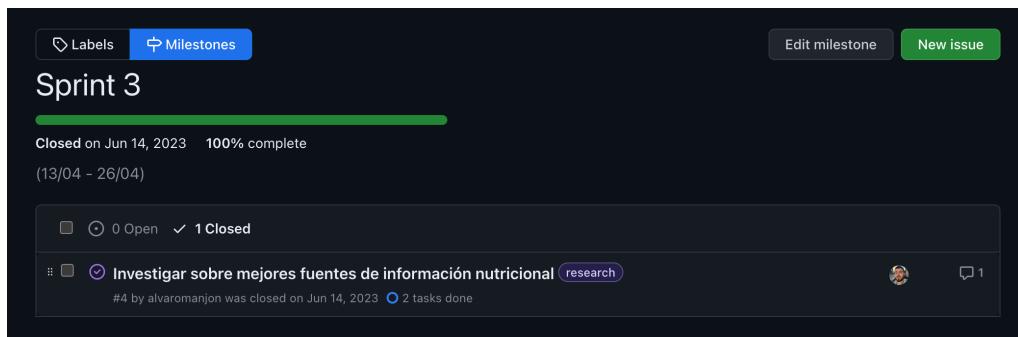


Figura A.3: Tareas del Sprint 3

Sprint 4 (27/04 - 17/05)

Este período de tiempo lo dedique a familiarizarme con LaTeX y a aprender cómo utilizarlo, ya que casi no tenía experiencia con este lenguaje. Fue también el momento en el que se decidió usar el propio repositorio como lugar para almacenar la memoria y los anexos, en vez de usar una herramienta externa como Overleaf. Fue aquí cuando descubrí y comencé a usar Texifier como herramienta para redactar estos documentos.

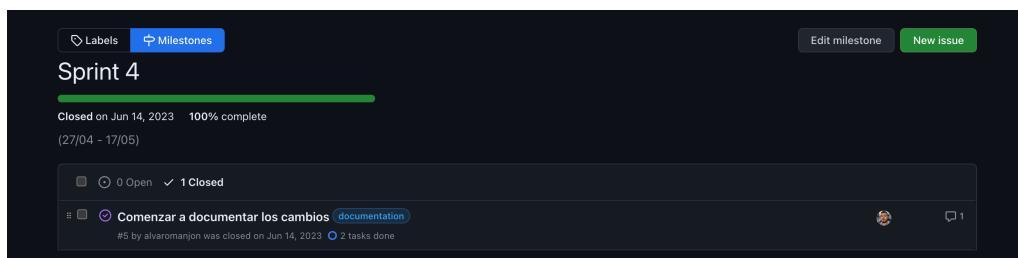


Figura A.4: Tareas del Sprint 4

Sprint 5 (18/05 - 01/06)

En este sprint fue cuando se terminó la primera iteración del despliegue de la infraestructura en Docker, creando un contenedor para la base de datos y otro para el proyecto de Spring Boot.

Es aquí donde se realizó toda la gestión del despliegue en Docker Compose, revisando que el orden de la ejecución de los contenedores fuese el correcto (ya que si la base de datos se ejecuta después que los proyectos de Spring Boot, la ejecución falla al no tener estos una base de datos a la que conectarse), así como la creación de los Dockerfiles y la configuración adecuada de sus parámetros.

Después de hacer varias pruebas y cambiar unas cuántas cosas, finalmente se consiguió desplegar un entorno en el que se pudiera realizar el desarrollo en local y los cambios se replicaran a los contenedores al momento.

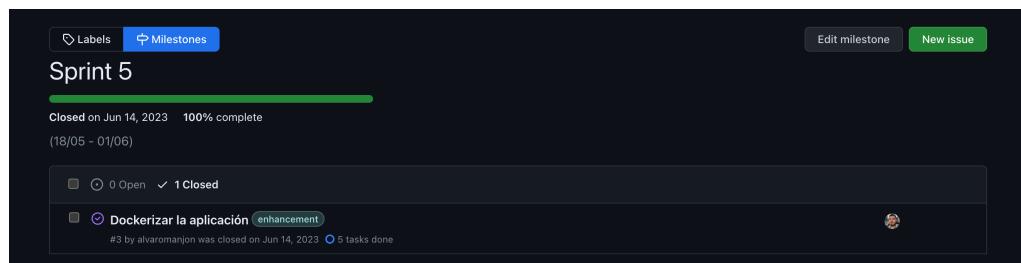


Figura A.5: Tareas del Sprint 5

Sprint 6 (02/06 - 15/06)

Este sprint se dedicó a desarrollar todos los cambios realizados en el sprint anterior en los anexos, así como a hablar sobre Docker en la memoria.

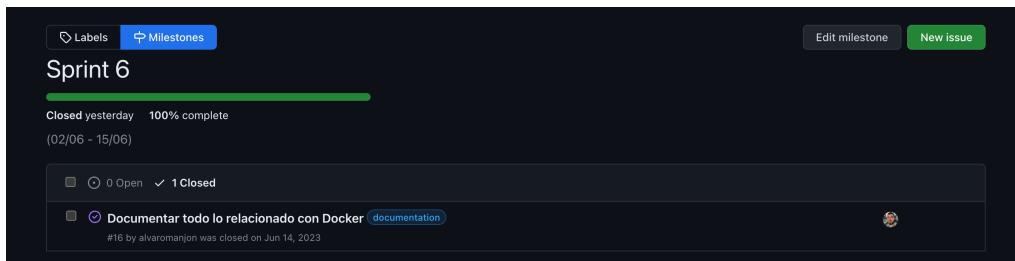


Figura A.6: Tareas del Sprint 6

Sprint 7 (16/06 - 22/06)

En este período de tiempo se realizó tanto toda la investigación referente a GitHub Actions como su implementación en el repositorio.

Inicialmente se intentó reutilizar la infraestructura ya implementada de Docker para esta tarea, pero no todos los elementos se traducían al 100 % a este tipo de despliegue, así que terminé desarrollando una infraestructura Docker ligeramente modificada.

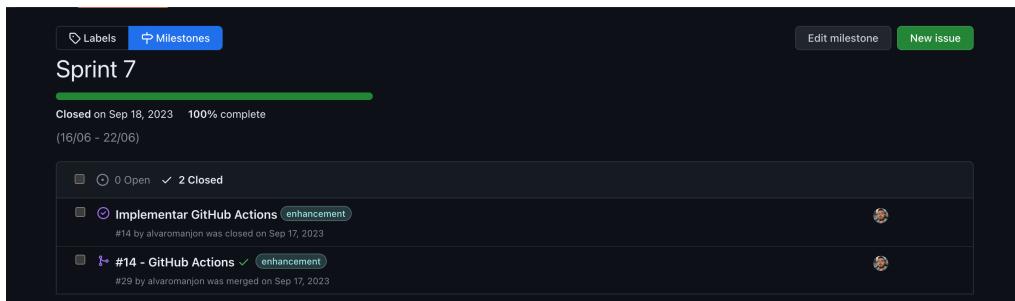


Figura A.7: Tareas del Sprint 7

Sprint 8 (07/09 - 14/09)

Fue en este sprint cuando se realizó toda la nueva implementación de la lógica de negocio, fusionando el backend de ambas aplicaciones en uno solo.

Me basé en el modelo de datos original para realizar esta nueva iteración, y la capa de negocio fue reescrita para que los datos actúasen de manera conjunta y no se encontrasen separados.

Por último se desarrolló la API en la capa del controlador, planteando qué endpoints iban a ser necesarios para que posteriormente la aplicación pudiese interactuar con los datos sin problemas.

Además de esto, también se realizaron algunas mejoras en la documentación de Docker para que quedase más claro cómo desplegar el proyecto desde 0, intentando cubrir todos los elementos necesarios.

Task	Status	Assignee	Merge Status
#19 - Improvement on Docker-related docs	Closed	avaromanjon	Merged
Mejorar documentación de la parte de contenerización	Closed	alvaromanjon	Closed
Crear una API REST	Closed	alvaromanjon	Closed
#20 - Creación de la API REST	Closed	alvaromanjon	Merged
Refactorizar los controllers de la API	Closed	alvaromanjon	Closed
#25 - Refactorización de los controllers de la API REST	Closed	alvaromanjon	Merged

Figura A.8: Tareas del Sprint 8

Sprint 9 (15/09 - 21/09)

En este sprint comenzó el desarrollo de la capa frontend, comenzando por crear un proyecto de React e integrar este proyecto dentro de la infraestructura Docker actual.

Fue en esta parte cuando se desarrollaron las primeras iteraciones de las vistas de gestión de todos los elementos, además de las vistas de inicio de sesión y contraseña olvidada. Aquí comencé a realizar mi primera implementación de la integración con la API de Nutritionix, pero no llegué a terminarla debido a que no conseguía sincronizar las múltiples llamadas a la API para que los datos se gestionasen correctamente.

También se documentaron todos los endpoints desarrollados para la API en los anexos, así como también se arregló un fallo que hacía que a veces la base de datos arrancase antes que el resto de servicios. Para solucionar esto se implementó un healthcheck en la base de datos, que se encarga de

A.2. Planificación temporal

7

comprobar si el servicio ha arrancado completamente, para que cuando así sea se avise al resto de servicios y continúen con su inicio.

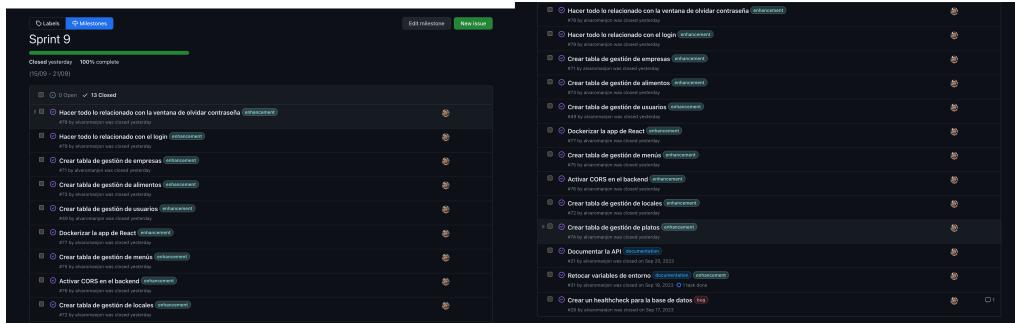


Figura A.9: Tareas del Sprint 9

Sprint 10 (21/12 - 08/01)

Dentro de este sprint se realizó la redacción completa del apartado Técnicas y herramientas de la memoria.

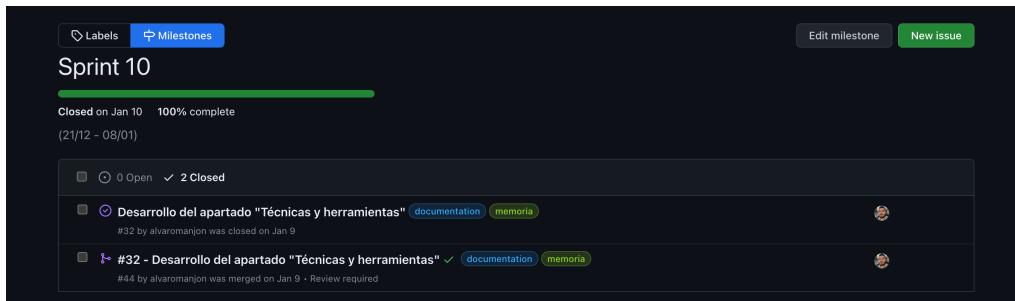


Figura A.10: Tareas del Sprint 10

Sprint 11 (09/01 - 16/01)

En este período se crearon las vistas de creación de usuarios, locales y empresas, así como toda su lógica subyacente para que estas funcionasen.

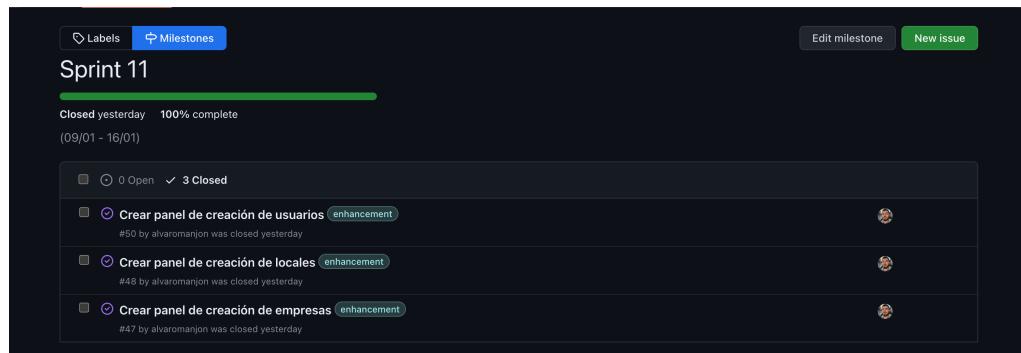


Figura A.11: Tareas del Sprint 11

Sprint 12 (17/01 - 30/01)

Este período lo dediqué mayoritariamente a seguir formándome sobre React, ya que era la tecnología con la que menos experiencia contaba, y había algunas partes del código con las que estaba teniendo complicaciones para conseguir que realizasen lo que tenía en mente.

También descubrí que la herramienta de compilación que estaba usando, Create React App, ya no era la opción recomendada por la comunidad de desarrolladores, ya que su ritmo de mantenimiento era bastante lento y sus paquetes se estaban quedando desactualizados, así que decidí hacer la migración a Vite. Esto se notó bastante en el rendimiento, ya que de repente todo funcionaba mucho más rápido, y dejé de tener problemas a la hora de instalar algunas dependencias.

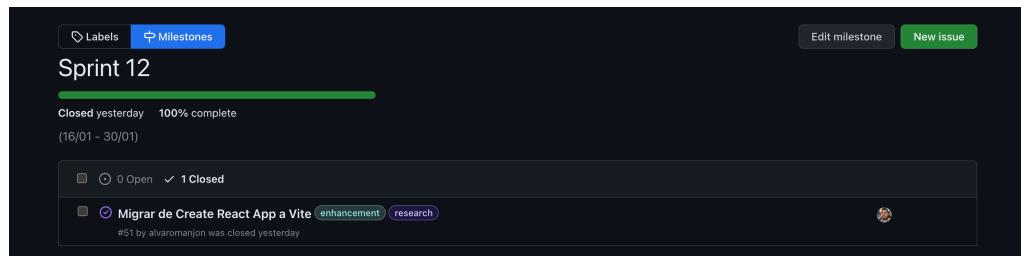


Figura A.12: Tareas del Sprint 12

Sprint 13 (31/01 - 05/02)

En este sprint conseguí terminar con la integración de la API de Nutritionix, lo que me supuso inicialmente un gran reto, y terminé de desarrollar toda la funcionalidad de las vistas de creación de alimentos.

También desarrollé la vista de creación de platos por completo, aprovechando lo desarrollado en las vistas de creación de alimentos para integrar estas aquí también.

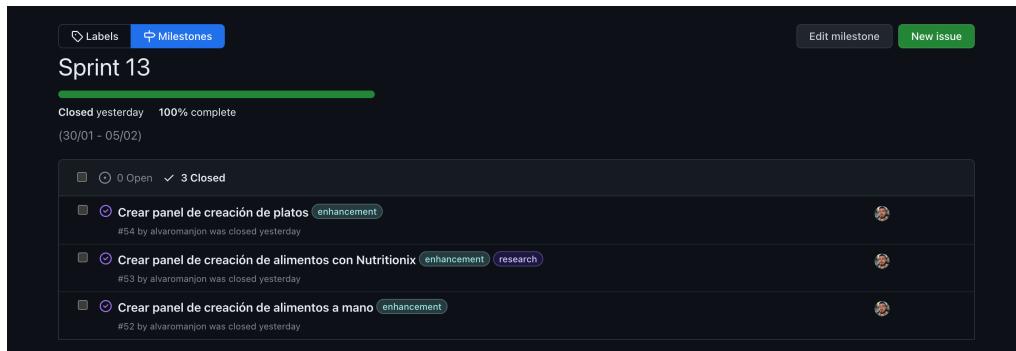


Figura A.13: Tareas del Sprint 13

Sprint 14 (06/02 - 12/02)

Durante este sprint se ha originado la vista de creación de menús, se han terminado de implementar todas las acciones relacionadas a eliminar elementos en los apartados de gestión, y se ha desarrollado todo lo relacionado a la parte de consumidores.

También se ha avanzado en el desarrollo de la memoria, terminando los apartados de Introducción, Objetivos del proyecto, y Aspectos relevantes del desarrollo del proyecto.

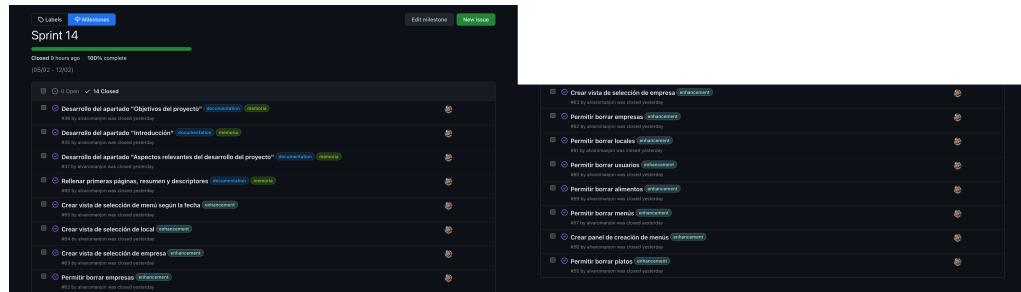


Figura A.14: Tareas del Sprint 14

Sprint 15 (13/02 - 16/02)

En este período se han realizado las últimas tareas por completar en la aplicación, añadiendo las vistas de edición de elementos e implementando un conjunto de datos de prueba para que la aplicación parte de unos datos en su primera ejecución.

Se han terminado de desarrollar los apartados de Conceptos teóricos, Trabajos relacionados y Conclusiones y líneas de trabajo futuras en la memoria, además de revisar unas mejoras pendientes en el apartado de Técnicas y herramientas.

En los anexos se han terminado todos los apartados, ya que había algunos pendientes de revisión y otros que todavía no se habían completado.

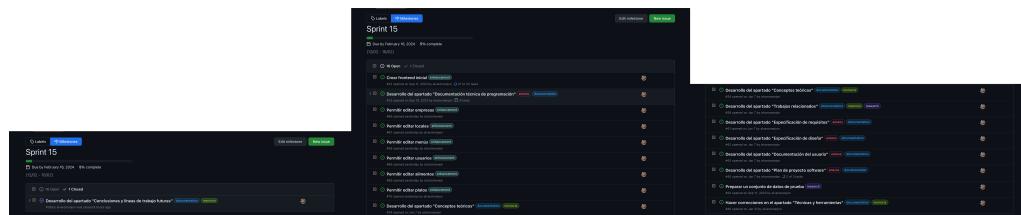


Figura A.15: Tareas del Sprint 15

A.3. Estudio de viabilidad

Viabilidad económica

Para elaborar un estudio de viabilidad económica de esta aplicación es necesario estimar los costes asociados a su desarrollo y mantenimiento, considerando el tiempo invertido en su realización. Se asume que el proyecto podría tener potencial para escalar o comercializarse en el futuro.

Todos los costes que se van a considerar a continuación llevan incluido el IVA del 21 %, haciéndose su desglose al final del análisis económico.

Costes de recursos humanos

Dado que el desarrollo del proyecto ha supuesto unas 300 horas efectivas a lo largo de 10 meses, se calcula el coste como si se estuviese contratando a un desarrollador externo, lo que da una idea del valor de mercado del trabajo realizado.

Para el coste por hora de un desarrollador de software, se considera un coste promedio de 50 €/hora (costes indirectos incluidos) para un desarrollador con las habilidades necesarias, por lo que para las 300 horas de trabajo indicadas, resulta un coste de 15 000 €.

$$300 \text{ horas} * 50 \text{ €/hora} = 15\,000 \text{ €}$$

Costes de hardware

El costo estimado de un equipo informático de desarrollo es de 2000 €, al que se le considera una vida útil de 3 años. Considerando que cada año tiene 1820 horas laborales, supone en total una vida útil para el ordenador de 5460 horas laborales.

El coste por hora de ordenador resulta ser de:

$$2000 \text{ €} / 5460 \text{ horas} = 0,37 \text{ €/hora}$$

Teniendo en cuenta las 300 h de trabajo utilizadas para el desarrollo del proyecto, resulta un coste de 111€:

$$300 \text{ horas} * 0,37 \text{ €/hora} = 111 \text{ €}$$

Costes de software y herramientas

Se ha intentado utilizar software gratuito u open source para el desarrollo del proyecto en la medida de lo posible, sin embargo hay ciertas herramientas que requieren de una licencia o que lo requerirían una vez puesto en producción:

- **Texifier**, la aplicación utilizada para redactar los anexos y la memoria, tiene un coste de 34,99 €. Suponiendo que la licencia puede llegar a durar unos 3 años, hasta que salga una actualización mayor que requiera volver a pagar para actualizar, y que la aplicación ha sido usada durante un tercio de las horas dedicadas al proyecto, el coste sería:

$$34,99 \text{ €} / 5460 \text{ horas} = 0,0064 \text{ €/hora}$$

$$0,0064 \text{ €} * 100 \text{ horas} = 0,64 \text{ €}$$

- **La API de Nutritionix** es gratuita cuando se utiliza para un desarrollo, ya que tiene un límite de 2 usuarios como máximo. Sin embargo, una vez puesta en producción, se requeriría una licencia que soporte un mayor número de usuarios. Al ser una aplicación gratuita, el coste sería necesario negociarlo con la empresa, como indican en [las FAQs de su web](#).

Por lo tanto, a falta de saber el precio del uso de Nutritionix, el coste quedaría en 0,64 € en gastos de software y herramientas.

Coste total del desarrollo

El coste total del desarrollo de este proyecto sería la suma de todos los costes mencionados anteriormente:

$$15\,000 \text{ €} + 111 \text{ €} + 0,64 \text{ €} = 15\,111,64 \text{ €}$$

Si además de esto queremos calcular el costo del proyecto con un año de operación, deberíamos de añadir los costes de infraestructura y mantenimiento:

Costes de infraestructura

Una vez desarrollada la aplicación, esta requerirá de una plataforma donde ser alojada para poder ser accesible.

Para hacer estas estimaciones, se ha supuesto que toda la infraestructura estaría alojada en una máquina virtual Compute Engine en Google Cloud, así que se ha hecho una configuración estimada de lo que podría ser una instancia adecuada para este proyecto (figura A.16).

El coste mensual sería de unos \$ 172,88, lo que convertido a euros a día de hoy sería unos 161,28 €. Por lo tanto, el coste anual sería de:

$$161,28 \text{ €} * 12 \text{ meses} = 1935,36 \text{ €}$$

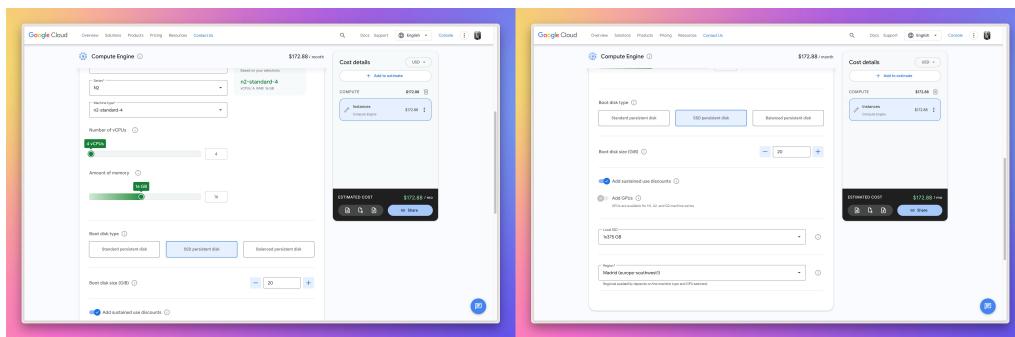


Figura A.16: Cálculos estimados de una instancia Compute Engine

Costes de mantenimiento

Las actualizaciones, corrección de errores, etc. pueden estimarse en términos de horas al año. Se estima que se necesitarán 50 horas al año al mismo coste que el tenido en cuenta para el desarrollador, lo que supone un coste anual de 2500 €:

$$50 \text{ horas/año} * 50 \text{ €/hora} = 2500 \text{ €/año}$$

Coste total del proyecto con un año de operación y mantenimiento

Sumando los costes de desarrollo y los de infraestructura y mantenimiento se obtiene el coste total del proyecto:

$$15\,111,64 \text{ €} + 1935,36 \text{ €} + 2500 \text{ €} = 19\,547 \text{ €}$$

Los cuales se desglosarían en:

- **Coste líquido:** 15 442,13 €
- **IVA (21 %):** 4104,87 €

Aunque el coste inicial del proyecto es significativo principalmente debido a los costes de desarrollo, los costes de infraestructura y mantenimiento son relativamente bajos, por lo que el proyecto podría ser económicamente viable si se implementa en más empresas y se gestiona eficientemente.

Viabilidad legal

Para realizar el estudio de viabilidad legal se van a listar todas las licencias involucradas en este proyecto en la tabla A.1.

La gran mayoría de licencias de los productos escogidos son open source, y todos son compatibles con la licencia escogida para este proyecto (figura A.17), la **GPL 3.0**. Para escoger la licencia se ha hecho uso de <https://choosealicense.com/>, una web que te permite escoger una licencia en función del propósito del proyecto.

Como este es un proyecto en el que probablemente termine trabajando e iterando más gente, la licencia GPL 3.0 es la más adecuada para este caso, ya que permite que se haga prácticamente de todo con el proyecto excepto distribuir el código de forma privada [2].

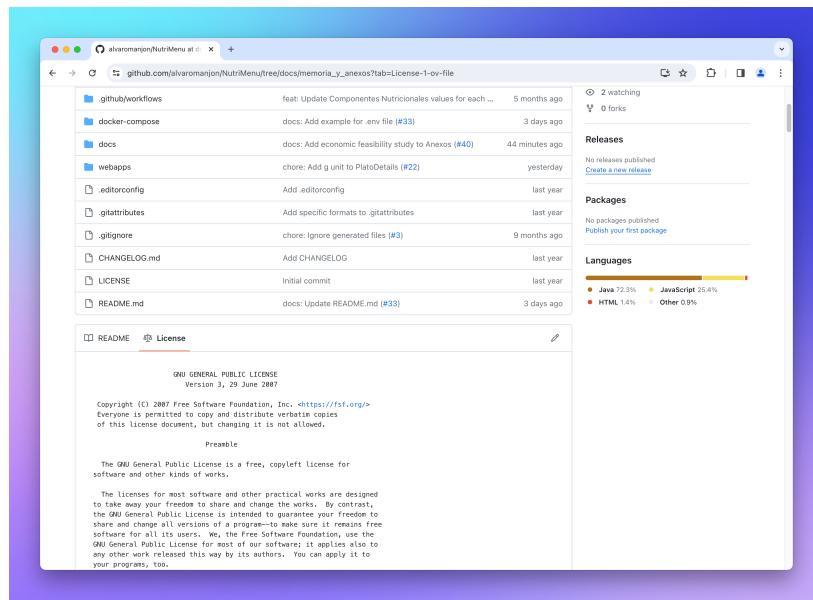


Figura A.17: Información detallada sobre la licencia escogida para el proyecto

Herramienta usada	Licencia
Docker (Community Edition)	Apache License 2.0
Docker Compose	Apache License 2.0
MySQL	GNU-2.0
Spring Boot	Apache License 2.0
Java (Eclipse Temurin)	Eclipse Public License (EPL) 2.0
React	MIT License
Vite	MIT License
React Router	MIT License
Bootstrap	MIT License
React Bootstrap	MIT License
AG React Data Grid (Community Edition)	MIT License
AG React Charts (Community Edition)	MIT License
JavaScript	Depende de la implementación del navegador, en caso del motor de Google (V8) es una mezcla de la MIT License y BSD License
Nutritionix	Propietaria, es una licencia de uso no exclusiva, revocable, no sublicenciable y no transferible
GitHub	Propietaria

Tabla A.1: Herramientas y tecnologías usadas y sus licencias

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apéndice se van a identificar y discutir los distintos objetivos que debe cumplir el proyecto, y los requisitos funcionales y no funcionales necesarios para que estos objetivos se lleguen a cumplir.

B.2. Objetivos generales

El objetivo del proyecto es el de disponer de una aplicación que permita mostrar a los consumidores de un centro de restauración los distintos menús que se ofrecen, que los usuarios puedan escoger el que deseen y que, dentro de él, a la hora de elegir los platos que van a consumir, puedan ver en tiempo real el cálculo de un informe nutricional en función de sus elecciones.

Para escoger el menú que desean, los consumidores primero deben seleccionar una fecha para ver los menús disponibles en ese día.

Inicialmente el proyecto está planteado para ser usado por la Universidad de Burgos, pero se va a disponer de la opción de crear nuevas empresas, por lo que podría ser implementado en otros lugares. Cada empresa tiene una URL única, en la cual se muestran los distintos locales de esa empresa, por lo que esta URL podría ser la facilitada a los usuarios finales.

Además de esto, también se ofrece un panel de gestión en el que el personal de los centros pueda administrar los distintos recursos, y este panel ofrece distintas funciones de administración en función del rol que cumpla el personal en la empresa.

Existen tres roles diferenciados:

- **Administrador:** Este tipo de usuario va a tener acceso para manejar los recursos globales de la aplicación, que son las empresas, los locales, los usuarios y los alimentos. Va a tener permisos para crear, editar o borrar cualquiera de estos 4 recursos. También va a tener posibilidad de ver los informes nutricionales de los alimentos. Es el único tipo de usuario que no va a estar vinculado a una empresa.
- **Editor:** Este tipo de usuario va a tener acceso a los recursos vinculados a su empresa (exceptuando los locales), es decir, va a tener acceso para gestionar los menús y los platos, pudiendo crear nuevos elementos, editarlos o borrarlos. Además de esto, dentro del panel de creación de platos va a tener la posibilidad de crear nuevos alimentos, pero no va a poder editarlos ni borrarlos. También va a poder visualizar los informes nutricionales de los menús, los platos y los alimentos.
- **Camarero:** Este tipo de usuario sólo va a tener acceso a gestionar los menús, pudiendo crear, editar y borrar los distintos menús de su empresa. Va a poder visualizar los informes nutricionales de los menús.

Para el acceso al panel de gestión, existe una vista de inicio de sesión en la que el usuario debe indicar su nombre de usuario y contraseña.

En caso de que el usuario haya olvidado su contraseña, hay un panel de contraseña olvidada en el que el usuario debe introducir su nombre de usuario, y a continuación, la nueva contraseña deseada.

B.3. Catálogo de requisitos

B.4. Especificación de requisitos

Apéndice C

Especificación de diseño

C.1. Introducción

Este apartado provee un estudio que permite entender mejor cómo se integran todas las piezas que conforman el proyecto. En los siguientes puntos se va a proceder a analizar cómo se ha planteado el modelo de datos, se van a explorar las distintas rutas que existen para interactuar con los datos y cómo se llevan a cabo, y se va a analizar la arquitectura de todos los servicios, con el objetivo de que leyendo esta sección se sea capaz de tener una idea bastante acertada de cómo se comporta y comunica la aplicación.

C.2. Diseño de datos

En esta sección se va a presentar el modelo de datos usado en la aplicación (figura C.1), y se van a describir las distintas relaciones entre las entidades que lo forman:

- **Empresa:** Contiene toda la información referente a las empresas registradas en la aplicación. Esta entidad puede estar relacionada con uno o varios locales, menús, platos y usuarios.
- **Usuario:** Dispone de toda la información referente a los usuarios, como su información de acceso y una enumeración que indica el rol al que pertenecen. Los roles a tomar pueden ser:

- **Administrador:** Permite el acceso al panel de administración, que se encarga de la gestión de empresas, locales, usuarios y alimentos. No pertenecen a ninguna empresa.
- **Editor:** Son miembros de una empresa, y se permite el acceso al panel de editores, que se encarga de la gestión de menús y platos.
- **Camarero:** Son miembros de una empresa, y se permite el acceso al panel de camareros, que se encarga de la gestión de menús.

Los usuarios con roles de Editor o Camarero van a tener una relación varios a uno con una empresa (realmente los administradores también, solo que el valor de id de empresa va a ser nulo).

- **Local:** Esta entidad contiene toda la información relacionada con los locales. Los locales sólo puede pertenecer a una empresa, debido a su relación varios a uno. Además, existe una relación de varios a varios con los menús.
- **Menú:** Es la entidad que contiene toda la información relacionada con los menús. Está directamente relacionada con las empresas, ya que varios menús pueden pertenecer a una sola empresa. Además existe una relación de varios a varios con los locales, y otra de este mismo tipo con los platos.
- **Plato:** Esta entidad dispone de toda la información referente a los platos, y dispone de una enumeración que indica qué tipo de plato es cada objeto de tipo Plato. Tiene una relación de varios a uno con una empresa, y además de esto tiene dos relaciones varios a varios: una correspondiente a los menús, y otra a los alimentos.
- **Alimento:** Es la entidad que almacena la información básica de los alimentos, como el grupo de alimento al que pertenece, los gramos por ración que representan los componentes nutricionales, y su nombre y descripción. Esta entidad tiene una relación de varios a varios con los platos, y además tiene una relación uno a uno con los componentes nutricionales, ya que cada alimento dispone de una instancia de componentes.
- **Componentes nutricionales:** Esta entidad contiene toda la información relevante a los componentes nutricionales de un alimento. Tiene dos relaciones uno a uno, una con vitaminas y otra con minerales, ya que estas 3 entidades juntas forman el junta de información referente a un alimento. También forma parte de una relación uno a varios con

la entidad Plato _Alimento, puesto que es la entidad intermedia que contiene información adicional sobre la relación varios a varios entre Plato y Alimento.

- **Vitaminas:** Es la instancia que contiene toda la información sobre las vitaminas de un alimento, y tiene una relación uno a uno con componentes nutricionales.
- **Minerales:** Es la instancia que contiene toda la información sobre los minerales de un alimento, y tiene una relación uno a uno con componentes nutricionales.
- **Plato_Alimento:** Esta relación es la que permite que varios platos puedan pertenecer a varios alimentos, y viceversa. Esta entidad intermedia existe para almacenar los gramos por ración escogidos para un alimento en un plato, y calcular y almacenar los valores de los componentes nutricionales proporcionales a esa cantidad.
- **Menu_Local:** Esta relación es la que permite que varios locales puedan pertenecer a varios menús, y viceversa. Esto es esencial para que no haga falta crear menús duplicados por cada local, y que estos puedan ser reutilizados tantas veces como sea necesario.
- **Menu_Plato:** Esta relación es la que permite que varios menús puedan pertenecer a varios platos, y viceversa. Esto permite que un plato pueda estar en varios menús, y no tengan que existir entidades duplicadas.

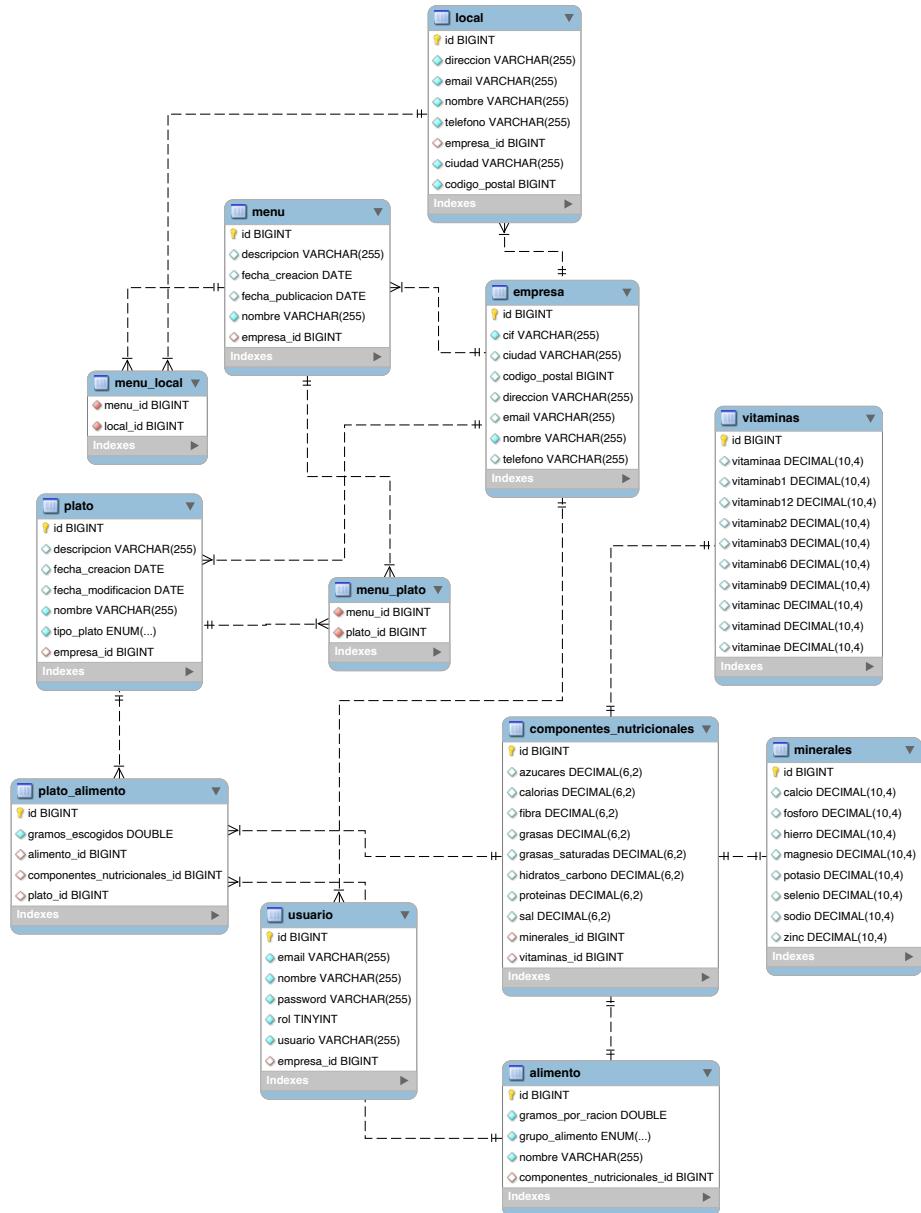


Figura C.1: Diagrama relacional del modelo de datos

C.3. Diseño procedimental

Diagramas de secuencia

A continuación se van a presentar las interacciones entre el usuario, la aplicación web y la API REST mediante el uso de diagramas de secuencia, ya que esto va a permitir entender mejor el flujo de ejecución de las distintas tareas que se pueden ejecutar en la aplicación.

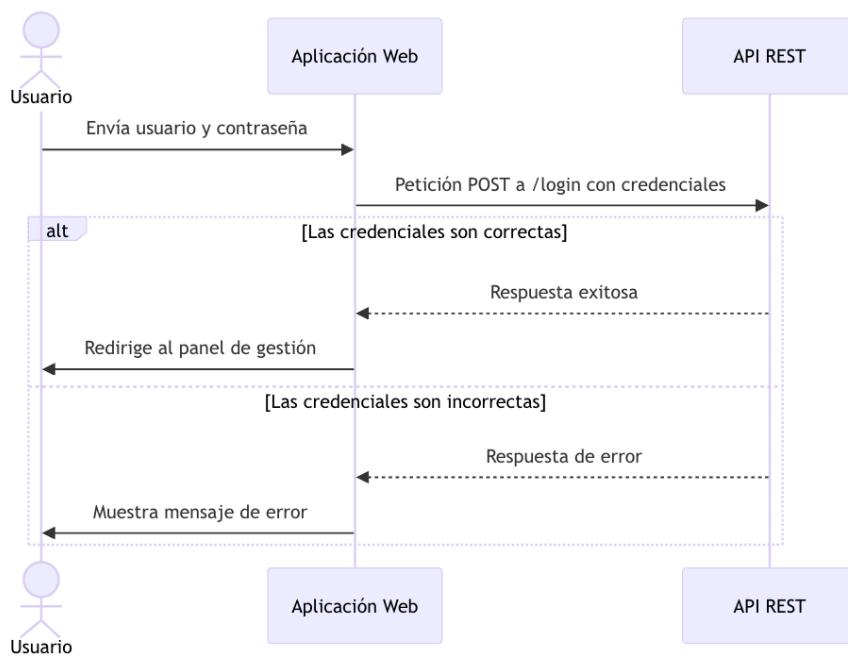


Figura C.2: Diagrama de inicio de sesión

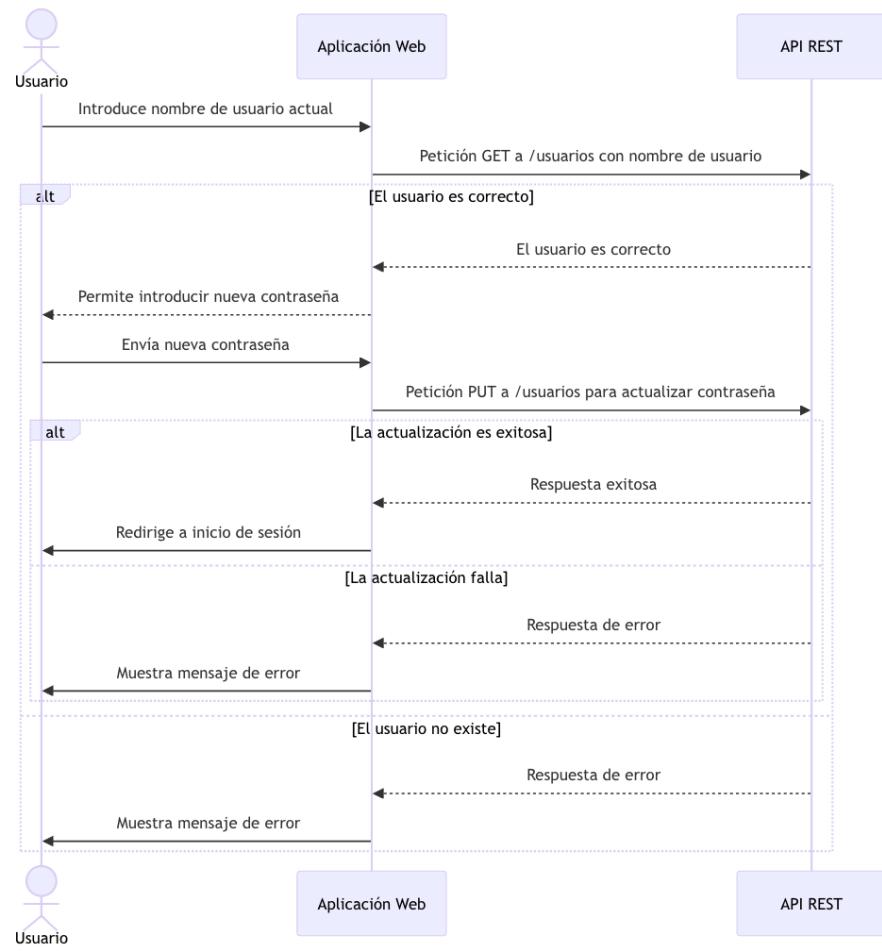


Figura C.3: Diagrama del proceso de contraseña olvidada

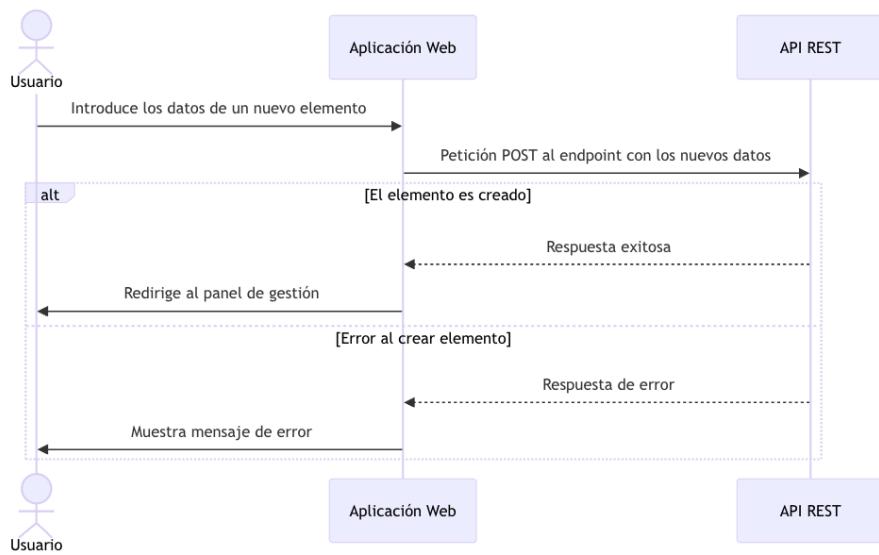


Figura C.4: Diagrama de creación de una empresa o alimento a mano

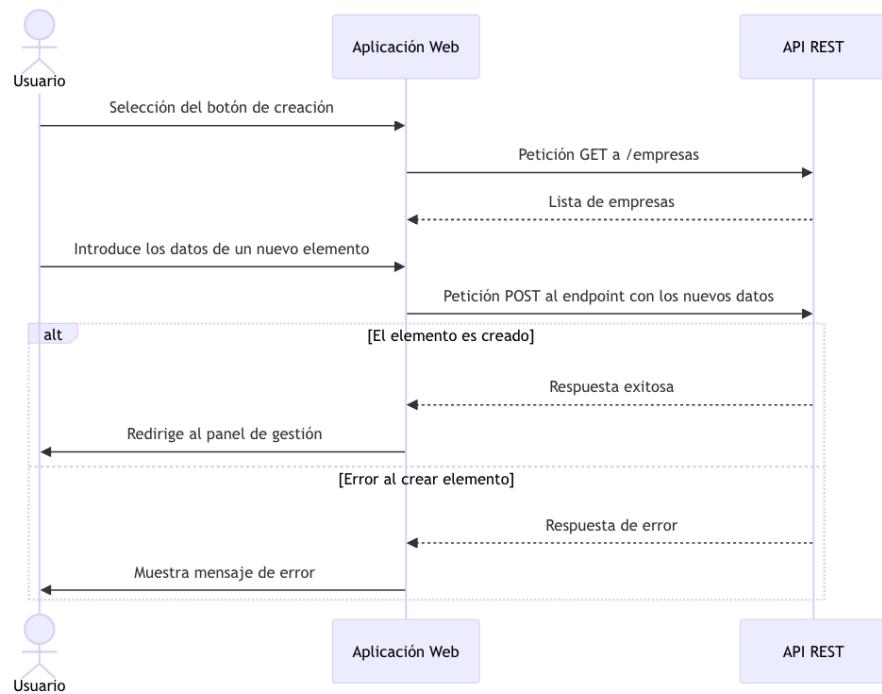


Figura C.5: Diagrama de creación de un local o usuario

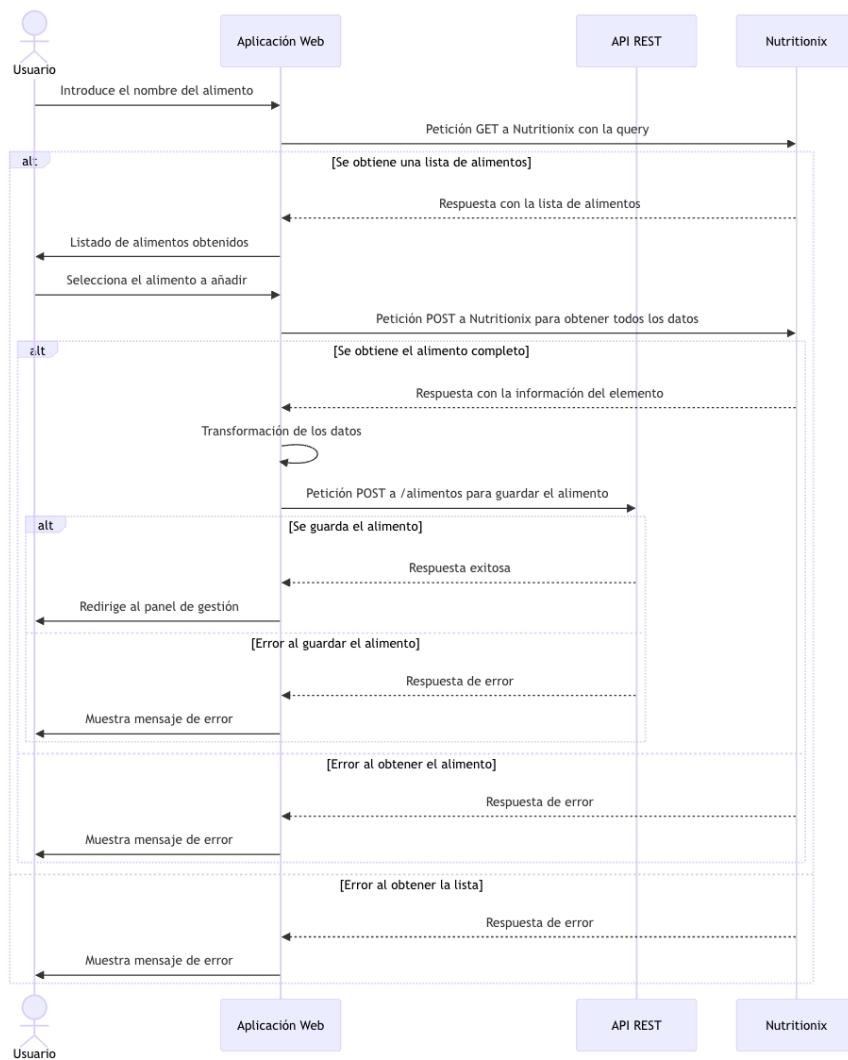


Figura C.6: Diagrama de creación de un alimento con Nutritionix

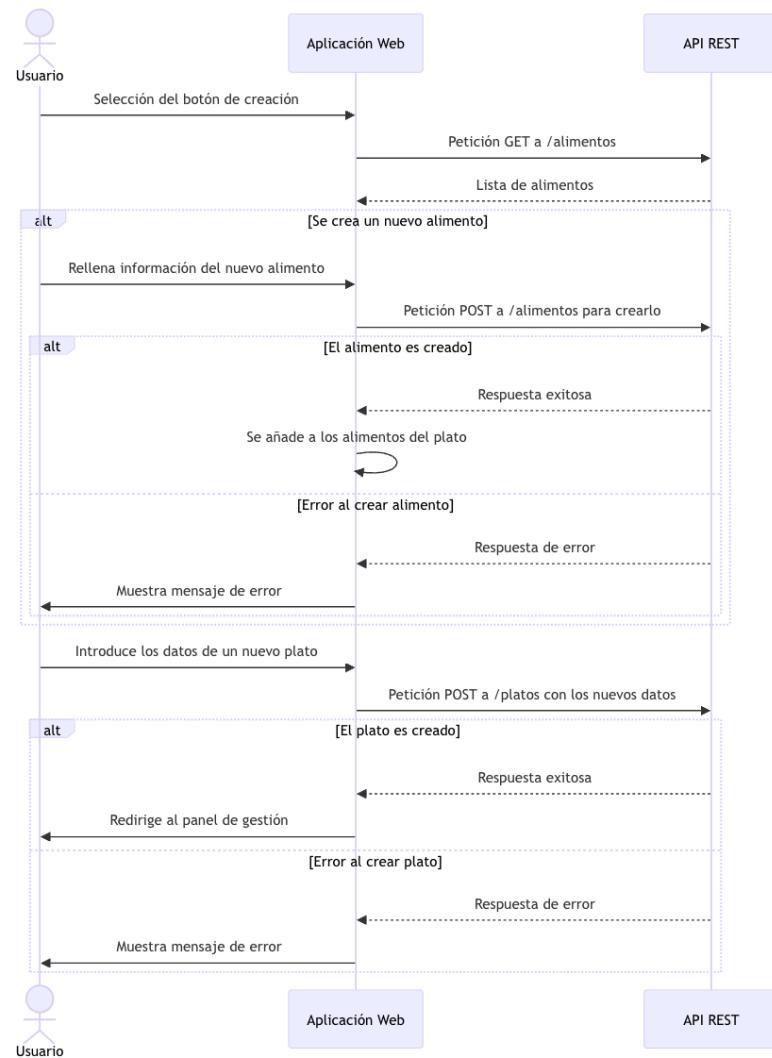


Figura C.7: Diagrama de creación de un plato

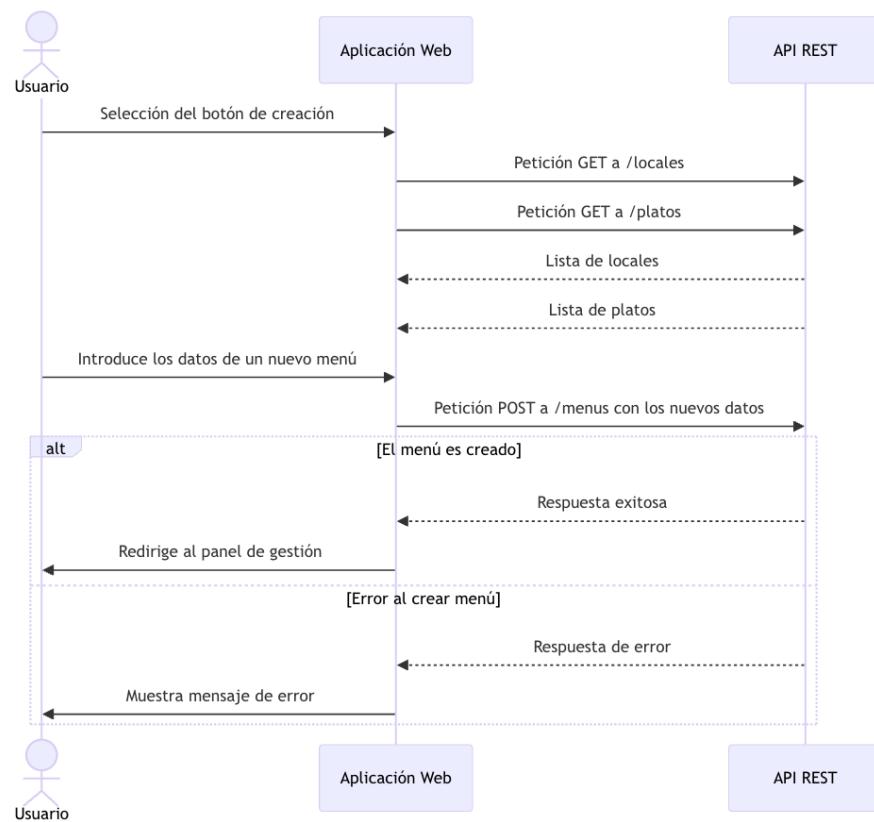


Figura C.8: Diagrama de creación de un menú

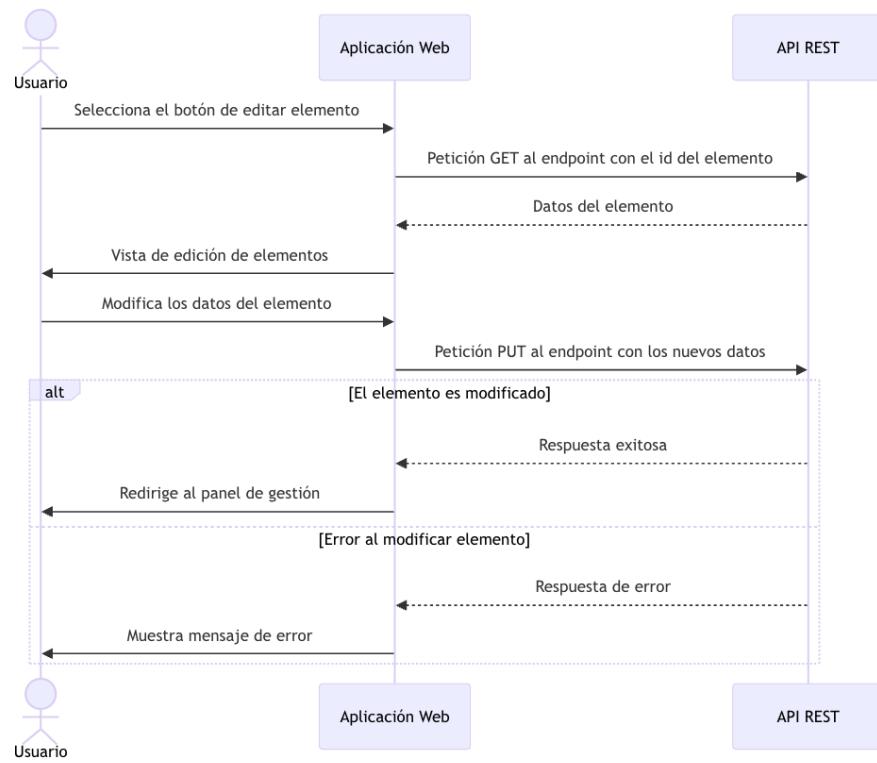


Figura C.9: Diagrama de edición de un elemento

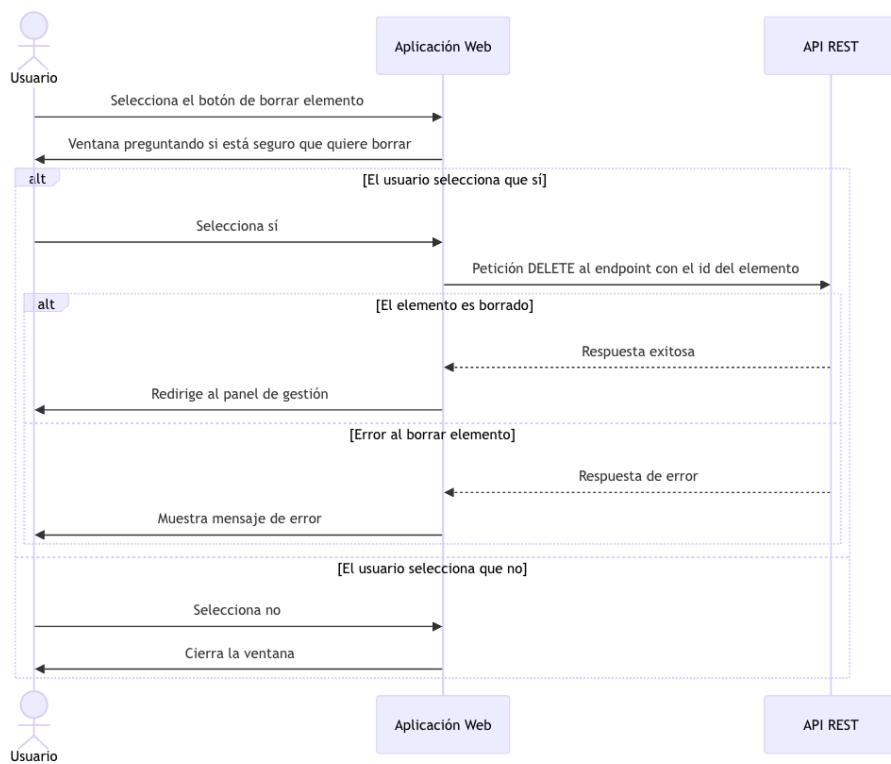


Figura C.10: Diagrama de borrado de un elemento

Diseño de la API REST

Uno de los grandes cambios que se han realizado en este proyecto respecto a la iteración anterior es el desarrollo de una API REST completa para trabajar e interactuar con los datos. Esta API nos permite abstraernos de la base de datos, ya que todas las operaciones CRUD en la aplicación se van a realizar atacando directamente a esta.

A continuación se van a detallar y documentar cada uno de los *endpoints* disponibles, para poder trabajar con ellos cómodamente y entender mejor las posibilidades que ofrecen.

get	/empresas
<i>Devuelve todas las empresas existentes</i>	
Parameter	
id_empresa	Devuelve la empresa con el ID correspondiente
nombre	Devuelve todas las empresas cuyo nombre contengan el valor pasado por parámetro
cif	Devuelve la empresa con el CIF correspondiente
Response	application/json
200	ok
<pre>{ "id": 13, "nombre": "Universidad de Burgos", "email": "contacto@ubu.es", "direccion": "Calle Don Juan de Austria, 1", "ciudad": "Burgos", "codigoPostal": 9001, "telefono": "947129482", "cif": "15544225C" }</pre>	
400	Bad Request: EntityDoesntExistException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": ["ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe una empresa con id 3", "path": "/empresas" }</pre>	

Tabla C.1: Endpoint que devuelve empresas existentes

post	/empresas
<i>Añade una nueva empresa</i>	
Parameter	
<i>No hay parámetros</i>	
Body	application/json
{ "nombre": "Vips", "email": "empresa@vips.com", "direccion": "Puerta del Sol, Madrid", "telefono": "900123123", "cif": "123456787" }	
Response	application/json
200 ok	{ "id": 3, "nombre": "Vips", "email": "empresa@vips.com", "direccion": "Puerta del Sol, Madrid", "telefono": "900123123", "cif": "123456787" }

Tabla C.2: Endpoint que permite añadir una empresa

put	/empresas
<i>Actualiza la información de una empresa ya existente</i>	
Parameter	
id_empresa	Modifica cualquier valor de la empresa con el ID correspondiente
Body application/json	
<pre>{ "nombre": "UBU" }</pre>	
Response application/json	
200	ok
<pre>{ "id": 13, "nombre": "Universidad de Burgos", "email": "contacto@ubu.es", "direccion": "Calle Don Juan de Austria, 1", "ciudad": "Burgos", "codigoPostal": 9001, "telefono": 947129482, "cif": "15544225C" }</pre>	
400	Bad Request: EntityDoesntExistException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe una empresa con id 3", "path": "/empresas" }</pre>	

Tabla C.3: Endpoint que permite actualizar la información de una empresa

delete	/empresas
<i>Elimina una empresa de la base de datos</i>	
Parameter	
id_empresa	Elimina la empresa con el ID correspondiente
Response	application/json
200	ok
Empresa con id eliminada correctamente	
400	Bad Request: EntityDoesntExistException
{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe una empresa con id 3", "path": "/empresas" }	

Tabla C.4: Endpoint que permite borrar una empresa

get	/usuarios
<i>Devuelve todos los usuarios existentes</i>	
Parameter	
id_usuario	Devuelve el usuario con el ID correspondiente
id_empresa	Devuelve todos los usuarios de la empresa con el ID correspondiente
usuario	Devuelve el usuario cuyo nombre de usuario sea el pasado por parámetro
email	Devuelve el usuario cuyo email sea el pasado por parámetro
rol	Devuelve todos los usuarios con un rol determinado
Response	application/json
200	ok
<pre>[{"id": 1, "usuario": "camarero", "password": "test", "nombre": "Administrador", "email": "camarero@gmail.com", "rol": "CAMARERO", "empresa": {...}}]</pre>	
400	Bad Request: EntityDoesntExistException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un usuario con id 3", "path": "/usuarios" }</pre>	

Tabla C.5: Endpoint que devuelve usuarios existentes

post	/usuarios
<i>Añade un nuevo usuario</i>	
Parameter	
id_empresa	ID de la empresa a la que va a pertenecer el usuario (en caso de que su rol lo indique)
Body application/json	
<pre>{ "usuario": "administrador", "password": "admin123", "nombre": "Administrador", "email": "administrador@gmail.com", "rol": "ADMINISTRADOR" }</pre>	
Response application/json	
200	ok
<pre>{ "id": 2, "usuario": "administrador", "password": "admin123", "nombre": "Administrador", "email": "administrador@gmail.com", "rol": "ADMINISTRADOR", "empresa": null }</pre>	
400 Bad Request: EntityDoesntExistException	
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe una empresa con id 3", "path": "/usuarios" }</pre>	

Tabla C.6: Endpoint que permite añadir un usuario

post	/login <i>Comprueba un inicio de sesión</i>
Body	application/json
	<pre>{ "usuario": "administrador", "password": "admin123" }</pre>
Response	application/json
200 ok	<pre>{ "id": 2, "usuario": "administrador", "password": "admin123", "nombre": "Administrador", "email": "administrador@gmail.com", "rol": "ADMINISTRADOR", "empresa": null }</pre>
400 Bad Request: EntityDoesntExistException	<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un usuario con id 3", "path": "/usuarios" }</pre>
401 Bad Request: PasswordNotCorrectException	<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "La contraseña para el ... es incorrecta", "path": "/usuarios" }</pre>

Tabla C.7: Endpoint que permite comprobar un inicio de sesión

put	/usuarios
<i>Actualiza la información de un usuario ya existente</i>	
Parameter	
id_usuario	Modifica cualquier valor del usuario con el ID correspondiente
Body application/json	
<pre>{ "nombre": "Juan Sanchez" }</pre>	
Response application/json	
200	ok
<pre>{ "id": 2, "usuario": "administrador", "password": "admin123", "nombre": "Juan Sanchez", "email": "administrador@gmail.com", "rol": "ADMINISTRADOR", "empresa": null }</pre>	
400	Bad Request: EntityDoesntExistException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un usuario con id 3", "path": "/usuarios" }</pre>	

Tabla C.8: Endpoint que permite actualizar la información de un usuario

delete	/usuarios
<i>Elimina un usuario de la base de datos</i>	
Parameter	
id_usuario	Elimina el usuario con el ID correspondiente
Response	application/json
200	ok
Usuario con id eliminado correctamente	
400	Bad Request: EntityDoesntExistException
{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un usuario con id 3", "path": "/usuarios" }	

Tabla C.9: Endpoint que permite borrar un usuario

get	/locales
<i>Devuelve todos los locales existentes</i>	
Parameter	
id_local	Devuelve el local con el ID correspondiente
id_empresa	Devuelve todos los locales correspondientes a la empresa con el ID correspondiente
nombre	Devuelve todos los locales cuyo nombre contengan el valor pasado por parámetro
email	Devuelve el local con el email correspondiente
telefono	Devuelve el local con el teléfono correspondiente
Response	application/json
200	ok
<pre>{ "id": 1, "nombre": "Escuela Politecnica", "email": "eps@ubu.es", "direccion": "Avenida Cantabria", "telefono": "947123123", "empresa": { "id": 1, ... } }</pre>	
400	Bad Request: EntityDoesntExistException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe una empresa con id 3", "path": "/locales" }</pre>	

Tabla C.10: Endpoint que devuelve locales existentes

post	/locales
<i>Añade un nuevo local</i>	
Parameter	
id_empresa	Indica a qué empresa pertenece el local
Body	application/json
<pre>{ "nombre": "Cafeteria EPS", "email": "eps@uba.es", "direccion": "Avenida Cantabria", "telefono": "947123954" }</pre>	
Response	
200	ok
<pre>{ "id": 3, "nombre": "Cafeteria EPS", "email": "eps@uba.es", "direccion": "Avenida Cantabria", "telefono": "947123954", "empresa": { "id": 1, ... } }</pre>	
400	Bad Request: EntityDoesntExistException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": ["ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe una empresa con id 3", "path": "/locales"] }</pre>	

Tabla C.11: Endpoint que permite añadir un local

put	/locales
<i>Actualiza la información de un local ya existente</i>	
Parameter	
id_local	Modifica cualquier valor del local con el ID correspondiente
Body	application/json
	<pre>{ "nombre": "Cafeteria Economicas" }</pre>
Response	application/json
200	ok
	<pre>{ "id": 3, "nombre": "Cafeteria Economicas", "email": "eps@ubu.es", "direccion": "Avenida Cantabria", "telefono": "947123954", "empresa": { "id": 1, ... } }</pre>
400	Bad Request: EntityDoesntExistsException
	<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un local con id 3", "path": "/locales" }</pre>

Tabla C.12: Endpoint que permite actualizar la información de un local

delete	/locales
<i>Elimina un local de la base de datos</i>	
Parameter	
id_local	Elimina el local con el ID correspondiente
Response	application/json
200	ok
Local con id eliminado correctamente	
400	Bad Request: EntityDoesntExistException
{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un local con id 3", "path": "/locales" }	

Tabla C.13: Endpoint que permite eliminar un local

get	/menus
<i>Devuelve todos los menús existentes</i>	
Parameter	
id_menu	Devuelve el menú con el ID correspondiente
id_local	Devuelve todos los menús del local con el ID correspondiente
id_empresa	Devuelve todos los menús de la empresa con el ID correspondiente
Response	
application/json	
200	ok
<pre>{ "id": 1, "nombre": "Menu del martes", "descripcion": "Menu del martes", "fechaCreacion": "2023-09-19", "fechaPublicacion": "2023-09-19", "locales": [...], "platos": [...] }</pre>	
400	Bad Request: EntityDoesntExistException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un menu con id 3", "path": "/menus" }</pre>	
400	Bad Request: LackOfParametersException
No se ha especificado ningun parametro de busqueda	

Tabla C.14: Endpoint que devuelve menús existentes

post	/menus
<i>Añade un nuevo menú</i>	
Parameter	
id_empresa	Indica a qué empresa pertenece el menú
Body	application/json
	<pre>{ "nombre": "Menu del martes", "descripcion": "Menu del martes", "fechaPublicacion": "2023-09-19", "platos": [{ "id": 3 }], "locales": [{ "id": 3 }] }</pre>
Response	application/json
200	ok
	<pre>{ "id": 1, "nombre": "Menu del martes", ... }</pre>
400	Bad Request: EntityDoesntExistException
	<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe una empresa con id 3", "path": "/menus" }</pre>

Tabla C.15: Endpoint que permite añadir menús

put	/menus <i>Actualiza la información de un menú ya existente</i>
Parameter	
id_menu	Modifica cualquier valor del menú con el ID correspondiente Body application/json
<pre>{ "nombre": "Menu del miercoles" }</pre>	
Response application/json	
200 ok	<pre>{ "id": 1, "nombre": "Menu del miercoles", "descripcion": "Menu del martes", "fechaCreacion": "2023-09-19", "fechaPublicacion": "2023-09-19", "locales": [...], "platos": [...] }</pre>
400 Bad Request: EntityDoesntExistsException	<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un menu con id 3", "path": "/menus" }</pre>

Tabla C.16: Endpoint que permite actualizar la información de un menú

delete	/menus
<i>Elimina un menú de la base de datos</i>	
Parameter	
id_menu	Elimina el menú con el ID correspondiente
Response	application/json
200	ok
Menu con id eliminado correctamente	
400	Bad Request: EntityDoesntExistException
{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un menu con id 3", "path": "/menus" }	

Tabla C.17: Endpoint que permite eliminar un menú

get	/platos
<i>Devuelve todos los platos existentes</i>	
Parameter	
id_plato	Devuelve el plato con el ID correspondiente
id_empresa	Devuelve todos los platos de la empresa con el ID correspondiente
tipo_plato	Devuelve todos los platos de un tipo determinado en una empresa (para poder realizar esta búsqueda es necesario pasar id_empresa también)
Response	application/json
200	ok
<pre>[{ "id": 1, "tipoPlato": "PRIMER_PLATO", "nombre": "Kevin Bacon", "descripcion": "Hamburguesa rellena de bacon", "fechaCreacion": "2023-09-19", "fechaModificacion": "2023-09-19", "alimentos": [...] }]</pre>	
400	Bad Request: EntityDoesntExistsException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un plato con id 3", "path": "/platos" }</pre>	
400	Bad Request: LackOfParametersException
No se ha especificado ningun parametro de busqueda	

Tabla C.18: Endpoint que devuelve platos existentes

post	/platos
<i>Añade un nuevo plato</i>	
Parameter	
id_empresa	Indica a qué empresa pertenece el plato
Body	application/json
	<pre>{ "nombre": "Paella", "descripcion": "Paella valenciana", "tipoPlato": "PRIMER_PLATO", "alimentos": [{ "id": 2, "cantidad": 150 }] }</pre>
Response	application/json
200	ok
	<pre>{ "id": 4, ... }</pre>
400	Bad Request: EntityDoesntExistException
	<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe una empresa con id 3", "path": "/platos" }</pre>

Tabla C.19: Endpoint que permite la creación de un plato

put	/platos	<i>Actualiza la información de un plato ya existente</i>
Parameter		
id_plato	Modifica cualquier valor del plato con el ID correspondiente	
Body		application/json
	<pre>{ "nombre": "Paella valenciana" }</pre>	
Response		
200	ok	application/json
	<pre>{ "id": 4, "tipoPlato": "PRIMER_PLATO", "nombre": "Paella valenciana", ... }</pre>	
400	Bad Request: EntityDoesntExistException	
	<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un plato con id 3", "path": "/platos" }</pre>	

Tabla C.20: Endpoint que permite modificar un plato

delete	/platos
<i>Elimina un plato de la base de datos</i>	
Parameter	
id_plato	Elimina el plato con el ID correspondiente
Response	application/json
200	ok
Plato con id eliminado correctamente	
400	Bad Request: EntityDoesntExistException
{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un plato con id 3", "path": "/platos" }	

Tabla C.21: Endpoint que permite borrar un plato

get	/alimentos
<i>Devuelve todos los alimentos existentes</i>	
Parameter	
id_alimento	Devuelve el alimento con el ID correspondiente
nombre	Devuelve todos los alimentos que contienen ese nombre
tipo_alimento	Devuelve todos los alimentos de un tipo determinado
Response	
application/json	
200	ok
<pre>[{ "id": 1, "nombre": "Tequenos", "grupoAlimento": "CEREALES", "gramosPorRacion": 125.0, "componentesNutricionales": { ... } }, ...]</pre>	
400	Bad Request: EntityDoesntExistException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un alimento con id 3", "path": "/alimentos" }</pre>	

Tabla C.22: Endpoint que devuelve alimentos existentes

post	/alimentos
<i>Añade un nuevo alimento</i>	
Body	application/json
	<pre>{ "nombre": "Arroz", "grupoAlimento": "CEREALES", "gramosPorRacion": 125, "componentesNutricionales": { ... } }</pre>
Response	application/json
200	ok
	<pre>{ "id": 2, "nombre": "Arroz", "grupoAlimento": "CEREALES", "gramosPorRacion": 125.0, "componentesNutricionales": { ... } }</pre>

Tabla C.23: Endpoint que permite añadir un alimento

put	/alimentos
<i>Actualiza la información de un alimento ya existente</i>	
Parameter	
id_alimento	Modifica cualquier valor del alimento con el ID correspondiente
Body application/json	
<pre>{ "nombre": "Teques" }</pre>	
Response application/json	
200	ok
<pre>{ "id": 1, "nombre": "Teques", "grupoAlimento": "CEREALES", "gramosPorRacion": 125.0, "componentesNutricionales": { ... } }</pre>	
400	Bad Request: EntityDoesntExistException
<pre>{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un alimento con id 3", "path": "/alimentos" }</pre>	

Tabla C.24: Endpoint que permite actualizar la información de un alimento

delete	/alimentos
<i>Elimina un alimento de la base de datos</i>	
Parameter	
id_alimento	Elimina el alimento con el ID correspondiente
Response	application/json
200	ok
Alimento con id eliminado correctamente	
400	Bad Request: EntityDoesntExistException
{ "timestamp": "2023-09-19T17:18:27.228+00:00", "status": 400, "error": "Bad Request", "trace": "ooo.alvar.nutrimenu.apirest.excepciones...", "message": "No existe un alimento con id 3", "path": "/alimentos" }	

Tabla C.25: Endpoint que permite eliminar un alimento

C.4. Diseño arquitectónico

En esta sección se va a analizar cómo está construida la aplicación desde una perspectiva estructural y de alto nivel. Para ello, se van a analizar los distintos patrones de diseño aplicados, así como la estructura de la infraestructura planteada.

Backend

Para el desarrollo del backend se ha aplicado el patrón Modelo-Vista-Controlador, como ya se ha descrito en la memoria. A continuación se van a desarrollar las distintas capas que conforman cada entidad en Spring Boot (figura C.11), con el fin de entender cómo se traduce el patrón MVC a este modelo:

- **Modelo:** Corresponde a la definición como tal de cada entidad, es decir, la estructura de datos por la que está compuesta y sus relaciones con otras entidades. Los modelos se van a mapear a las tablas de la base de datos a través del uso de JPA y anotaciones.
- **Repositorio:** Esta capa permite abstraer la lógica de acceso a datos.

Para crear un repositorio, se define una interfaz que extiende *CrudRepository*, otra interfaz que ofrece métodos ya predefinidos de acceso a datos, para evitar tener que escribir consultas SQL explícitas. En esta interfaz se van a indicar los métodos que van a ser necesarios para acceder a la base de datos.

En caso de que algún método requiera de una operación no contenida en *CrudRepository*, también se pueden indicar las *queries* SQL de forma explícita mediante el uso de JPQL.

- **Servicio:** Esta es la capa que contiene la lógica de negocio de la aplicación.

Se encuentra entre la capa de repositorio y la de controlador, puesto que va a ser la encargada de realizar las operaciones complejas de gestión de datos mediante el uso de los repositorios, para luego proveer estos métodos al controlador.

- **Controlador:** La capa de controlador es la encargada de definir, gestionar y proveer de una forma de acceso a los datos, que en nuestro caso son los *endpoints* de la API.

Esta capa interactúa con los distintos métodos que provee la capa de servicio para que cada endpoint realice las operaciones correspondientes, y sea capaz de presentar los datos adecuados.

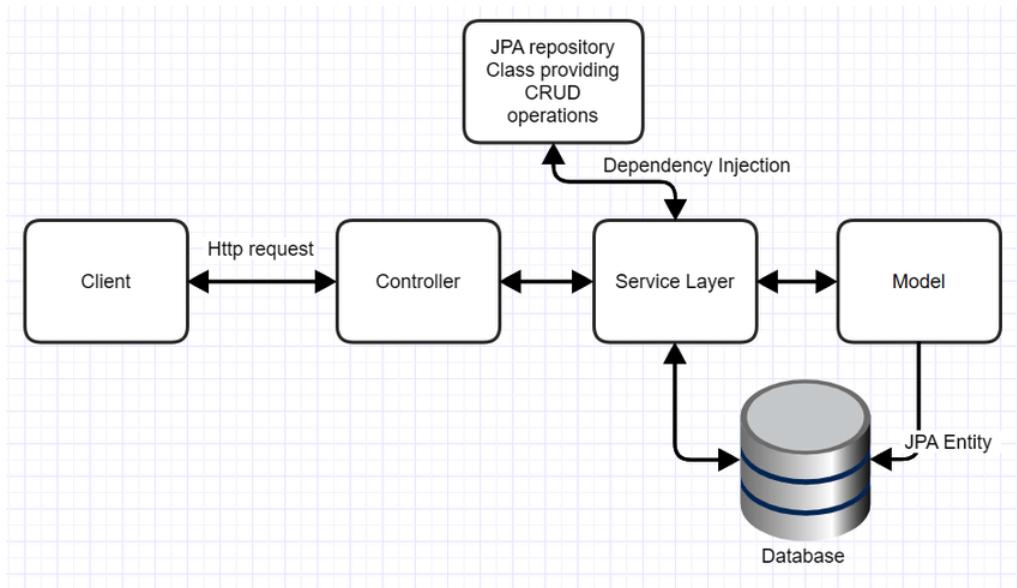


Figura C.11: Arquitectura de un proyecto Spring Boot [14]

Frontend

Para el desarrollo del frontend, las distintas piezas que conforman la aplicación se han implementado siguiendo distintos patrones:

- **Patrón Contenedor/Presentacional:** Este patrón se encarga de dividir la lógica de negocio y el estado de la aplicación de la presentación de los datos (figura C.12), lo cual favorece a separar responsabilidades y hacer un código más reusable y mantenible.

Esta es una estrategia que se ha seguido bastante a lo largo del desarrollo de la aplicación, especialmente en las partes encargadas de crear elementos.

Un componente padre (**Contenedor**) se encarga del manejo de los datos, mientras que los componentes hijos (**Presentacionales**) son los encargados de presentar la interfaz al usuario.

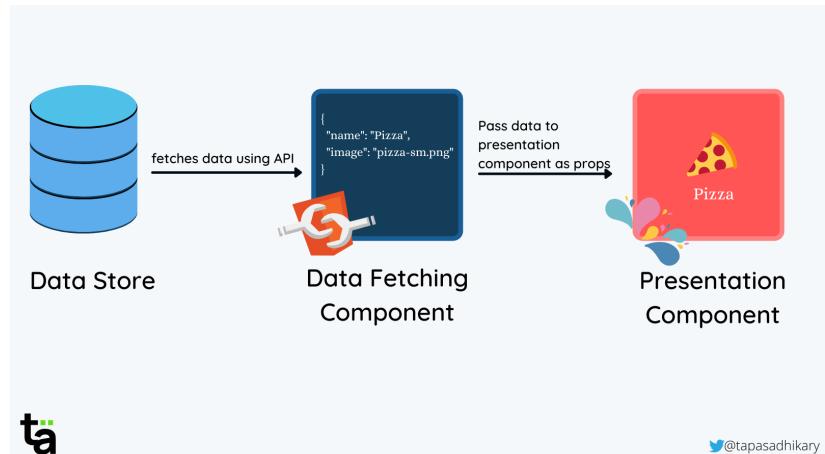


Figura C.12: Diagrama del patrón Contenedor/Presentacional en React [1]

- **Patrón Proveedor:** El patrón Proveedor se usa en React para pasar datos a lo largo del árbol de componentes sin tener que hacerlo de forma explícita, ya que se acabaría produciendo lo que se denomina como *perforación de accesorios* (figura C.13), que es el enviar props de un elemento de alto nivel a otro de muy bajo nivel, pasando este prop por todos los elementos intermedios.

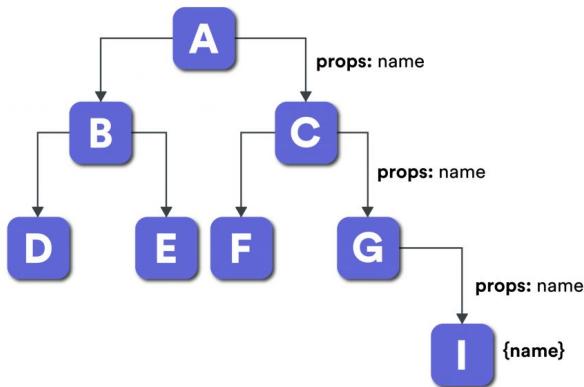


Figura C.13: Diagrama de la perforación de accesorios [16]

Para esto, lo que se hace es envolver al árbol de componentes de un componente Proveedor (figura C.14), que va a ser el que va a almacenar los valores que queremos que estén disponibles.

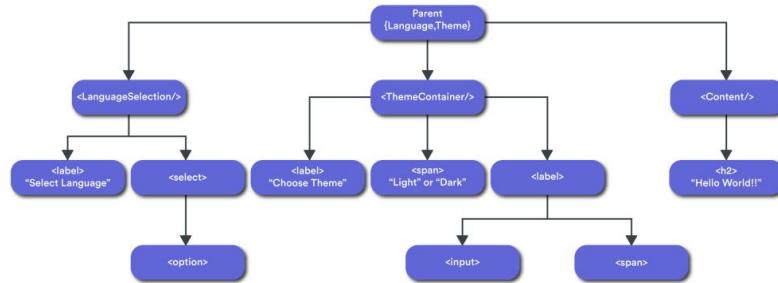


Figura C.14: Diagrama del patrón Proveedor en React [16]

En caso de este proyecto esto se utiliza para almacenar la información referente al usuario que ha iniciado sesión.

Infraestructura

La infraestructura de la aplicación consta de 3 servicios, que son el *frontend*, el *backend* (API REST) y la base de datos, los cuales se encuentran conectados entre ellos (figura C.15).

Los puertos por los que se van a conectar los distintos elementos se van a especificar en el archivo de variables, aunque mis puertos recomendados son los que se van a indicar a continuación:

- El usuario se conecta al frontend mediante el puerto **3000**.
- El frontend hace llamadas a la API REST mediante el puerto **8080**.
- El backend se conecta a la base de datos mediante el puerto **3306**.

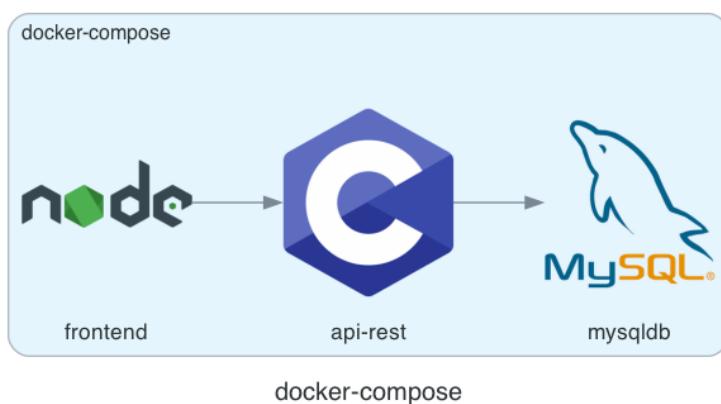


Figura C.15: Diagrama de la infraestructura en Docker Compose

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

El proyecto se encuentra alojado en un repositorio de GitHub, el cuál contiene tanto esta documentación, como las dos aplicaciones principales (API REST y frontend), así como toda la infraestructura necesaria para ser desplegadas.

La URL del repositorio es la siguiente:

<https://github.com/alvaromanjon/NutriMenu>

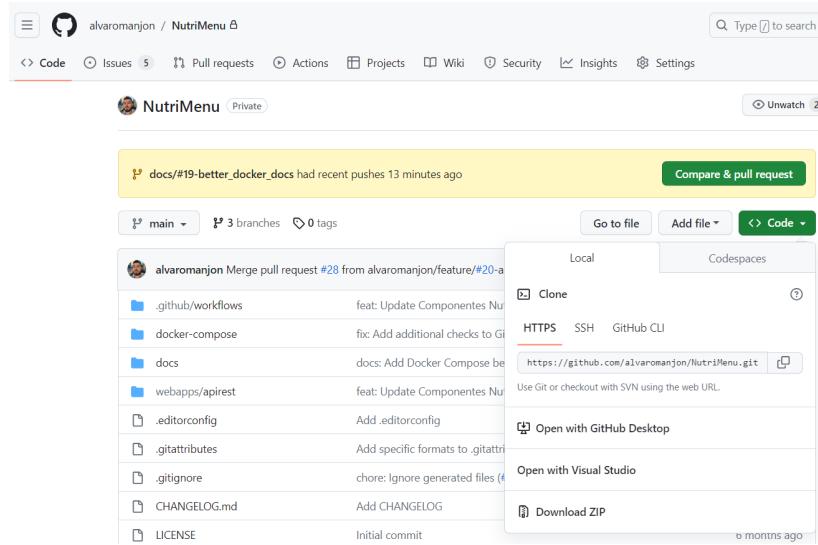
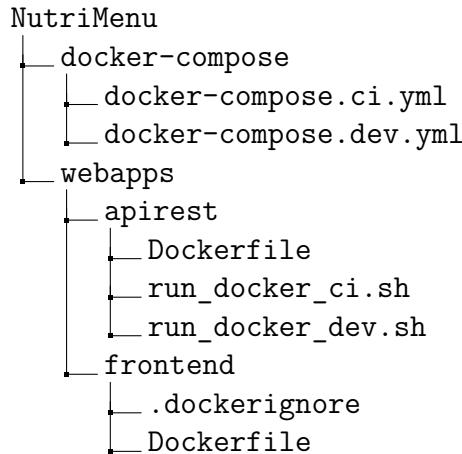


Figura D.1: Descarga del repositorio del proyecto

Podemos descargar el repositorio desde el propio GitHub, haciendo click en el botón *Code > Download ZIP*, o clonando el repositorio escribiendo en una línea de comandos:

```
git clone https://github.com/alvaromanjon/NutriMenu.git
```

Estructura de los archivos de Docker



D.3. Manual del programador

Dockerfiles

API REST

```
FROM eclipse-temurin:17
RUN apt update && apt -y upgrade
RUN apt install -y inotify-tools dos2unix
ENV HOME=/app
RUN mkdir -p $HOME
WORKDIR $HOME
```

A la hora de desplegar la parte de *backend* disponemos para ello de un **Dockerfile**, que va a ser el encargado de definir cómo se va a crear la imagen de Docker, y qué parámetros y dependencias va a necesitar. Es el equivalente al proceso al que todos estamos acostumbrados de crear una máquina virtual, instalar el sistema operativo escogido y configurar todas las dependencias y paquetes necesarios, solo que en este caso todo esto podemos ahorrárnoslo (tanto en esfuerzo como en tiempo, ya que el tiempo de despliegue de una imagen de Docker es de apenas un par de minutos en la mayoría de los casos) escribiendo en este archivo la configuración necesaria que necesitamos para nuestro caso de uso.

En este caso me he basado en otra imagen ya existente, *eclipse-temurin* ([Docker Hub](#)), ya que contiene tanto el JDK como el JRE de Java, además de una instalación mínima de **Linux Ubuntu** con todas las dependencias ya instaladas y configuradas. He especificado que quiero usar la etiqueta **17**, puesto que quiero usar Java 17 para este proyecto, ya que es la última versión LTS (*Long Term Support*) disponible en el momento [11]. Existen muchas implementaciones del JDK de Java [4], pero en este caso he usado Eclipse Temurin principalmente por su soporte a arquitecturas *ARMv7* y *ARM64/v8*, lo que permite poder ejecutar los contenedores en plataformas como Apple Silicon, o incluso en mini-computadores como una Raspberry Pi.

Aunque la imagen descarga automáticamente la última versión disponible de [Docker Hub](#), por si acaso cada vez que construyamos la imagen vamos a comprobar si hay actualizaciones disponibles para cualquier paquete del sistema operativo del que dispone la imagen. Además de esto, también se van a instalar como dependencias *inotify-tools*, una herramienta para monitorizar cambios en el código; y *dos2unix*, que se encarga de cambiar

el tipo de final de línea de **CRLF** (Windows) a **LF** (Unix) [12]. Indicamos también que, dentro de la imagen, el directorio que vamos a usar para trabajar es `/app`, puesto que esto nos permite que cualquier comando que ejecutemos a continuación se realice dentro de ese directorio [6].

Frontend

```
FROM node:alpine
ENV HOME=/app
RUN mkdir -p $HOME
WORKDIR $HOME
COPY package*.json ./
RUN npm install
EXPOSE 3000
CMD ["npm", "start"]
```

En la parte del *frontend* también disponemos de otro **Dockerfile**, en este caso con la imagen oficial de **Node**, ya que es el entorno de ejecución que hay debajo de React. He usado la etiqueta **alpine** (Docker Hub), ya que **Alpine** es una distribución de Linux orientada a ser lo más ligera posible, reduciendo el número de paquetes al mínimo y sustituyendo las herramientas GNU por **BusyBox** [17], un ejecutable que es capaz de emularlas.

Al igual que en el otro Dockerfile que acabamos de ver, he configurado el entorno de ejecución en el directorio `/app` del contenedor. A continuación he indicado que quiero que se copien los archivos que empiezan por `package` y acaban en `.json` al contenedor, ya que en un proyecto React son los archivos que indican todas las dependencias que necesita el proyecto para poder ejecutarse. Sobre el qué directorio va a usar Docker para coger estos archivos, lo veremos después en el archivo de configuración de **Docker Compose**, puesto que al construir la imagen con la etiqueta `build`: indicaremos el directorio al que puede acceder Docker durante la construcción de la imagen con el atributo `context`: [5].

Por último, la imagen va a instalar todas las dependencias del proyecto necesarias con `npm install`, se va a abrir el puerto `3000` del contenedor, ya que es el utilizado por React para poder acceder a la aplicación, y se va a iniciar la aplicación con el comando `npm start`.

Docker Compose

Dentro de los archivos de configuración de Docker Compose, situados en la carpeta `docker-compose` del directorio raíz del proyecto, tenemos dos archivos distintos:

- `docker-compose.ci.yml`: Es el encargado de configurar los contenedores que se van a desplegar en el **entorno CI/CD configurado**, es decir, *Github Actions*. Es prácticamente idéntico al entorno de desarrollo, tan sólo cambian las referencias a los scripts de arranque, como veremos a continuación.
- `docker-compose.dev.yml`: Es el **entorno de desarrollo** utilizado durante la creación de esta aplicación, ya que está configurado de tal forma que permite interactuar con la aplicación y aplicar y compilar los cambios en el código en tiempo real, sin necesidad de reiniciar los contenedores, usando *LiveReload* [9].

Frontend

```
frontend:  
  build:  
    context: ../webapps/frontend  
  restart: always  
  ports:  
    - ${FRONTEND_DOCKER_PORT}:3000  
  networks:  
    - nutrimenu-net  
  env_file:  
    - .env  
  depends_on:  
    - api-rest  
  volumes:  
    - ../webapps/frontend:/app:rw  
    - /app/node_modules  
  stdin_open: true
```

Dentro de nuestro archivo de configuración `docker-compose.dev.yml` el primer servicio con el que nos encontramos es el correspondiente al contenedor encargado de desplegar la parte del *frontend*. Esta parte no aparece dentro de `docker-compose.ci.yml`, ya que GitHub Actions sólo está configurado para la parte del *backend* y la base de datos.

Dentro de este archivo de configuración podemos destacar algunos elementos:

- Como hemos mencionado anteriormente en el apartado de los *Dockerfile* [D.3](#), la etiqueta `context`: en `build:` indica el **directorio al que vamos a dar acceso a Docker a la hora de construir la imagen**, que en este caso es `webapps/frontend`. Los dos puntos antes del directorio significan que ese directorio se encuentra fuera de la carpeta actual, `docker-compose`.
- El contenedor está configurado para **reiniciarse de forma automática** en caso de fallo [\[10\]](#).
- El contenedor **va a estar conectado a una red** creada también en el fichero de Docker Compose, la cual veremos a continuación.
- Se va a usar el fichero de **variables de entorno** descrito en el apartado [D.4](#) para definir las distintas variables especificadas en este archivo.
- El contenedor va a esperar a que el contenedor de la **API REST** esté en estado *ready* para arrancar [\[7\]](#).
- **Se van a crear dos volúmenes**, uno encargado de **mapear los archivos de la aplicación en local** (`./webapps/frontend`) **con el directorio de trabajo dentro del contenedor** (`/app`), indicando que este directorio va a ser tanto de lectura como de escritura; y otro volumen específico para `node_modules`, **la carpeta que contiene todas las dependencias descargadas**. Este primer volumen es lo que nos permite poder realizar modificaciones sin tener que reiniciar servicios ni volver a construir imágenes, mientras que el segundo volumen se monta debido a que, cuando montamos este primer volumen, el contenido que hubiera en nuestro directorio de trabajo del contenedor (`/app`) se sobreescribe por el del directorio mapeado, así que con este segundo volumen montado podemos recuperar la carpeta `node_modules` generada al crear la imagen con el comando `npm install`, puesto que sin esta carpeta **no podemos ejecutar la aplicación** (a no ser que hubiéramos ejecutado la app en local anteriormente y ya tuviésemos una carpeta `node_modules`) [\[15\]](#).
- **Se mantiene una entrada abierta para el contenedor**, lo que nos puede servir para poder instalar paquetes usando `npm`.

Backend

```

api-rest:
  build:
    context: ../webapps/apirest
  restart: always
  ports:
    - ${API_WEB_DOCKER_PORT}:${API_WEB_LOCAL_PORT}
    - ${API_RELOAD_DOCKER_PORT}:${API_RELOAD_LOCAL_PORT}
    - ${API_DEBUG_DOCKER_PORT}:${API_DEBUG_LOCAL_PORT}
  networks:
    - nutrimenu-net
  env_file:
    - .env
  depends_on:
    mysqlDb:
      condition: service_healthy
  working_dir: /app
  command: sh run_docker_dev.sh
  volumes:
    - ../webapps/apirest:/app
    - .m2:/root/.m2

```

Para la parte del *backend* muchos de los elementos del Dockerfile son similares a los vistos en la parte del *frontend*, así que simplemente destacaré aquellos que varían respecto a este:

- Se van a **configurar los puertos** descritos en el apartado [D.4](#).
- A la hora de depender del contenedor de la base de datos para arrancar, he especificado la condición **service_healthy**, que hace que no sólo el contenedor de la API espere a que el de la base de datos esté en estado *ready*, sino que además **va a ejecutar un *healthcheck* para comprobar que la base de datos se encuentra accesible**. Esto es **MUY importante** para el correcto funcionamiento de la app, ya que sin este *healthcheck*, se puede dar el caso de que la API REST arranque antes que la base de datos, causando un error en la ejecución.
- Nada más arrancar **va a ejecutar el comando `run_docker_dev.sh`**. Este comando lo que hace es levantar el servicio de *backend* y estar a su vez a la espera de cambios en los ficheros del código fuente para, en caso de que suceda, recompilar el código de forma transparente. Este

apartado es distinto en el archivo `docker-compose.ci.yml`, puesto que hace referencia a `run_docker_ci.sh` que simplemente compila la aplicación y la ejecuta.

- Se van a conectar dos volúmenes al contenedor: uno va a ser **el que contiene el código del servicio** (ya que si simplemente copiáramos el código al contenedor, cada vez que hubiera cambios en el código habría que volver a desplegar los contenedores, y de esta forma podemos realizar cambios en tiempo real); y otro va a ser **el volumen encargado de almacenar las dependencias de Maven**, para no tener que descargarlas cada vez que ejecutemos el código.

Base de datos

```
mysql ldb:
  image: "mysql:8.0"
  restart: always
  ports:
    - ${DB_DOCKER_PORT}:${DB_LOCAL_PORT}
  networks:
    - nutrimenu-net
  env_file:
    - .env
  volumes:
    - db_data:/var/lib/mysql
  healthcheck:
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u",
           "root", "-p${MYSQL_ROOT_PASSWORD}"]
    interval: 15s
    timeout: 5s
    start_period: 10s
    retries: 3
```

La base de datos también va a disponer de su propio contenedor, con ciertos parámetros similares a los de los servicios del *backend*. Podemos destacar de aquí algunos elementos:

- Vamos a usar la **imagen oficial de MySQL** ([Docker Hub](#)), concretamente la versión 8 (con los últimos parches).
- Se va a conectar un volumen a la base de datos, que no es más que **el volumen encargado de almacenar todos los datos** de la base de

datos. Esto es para no tener que inicializar la base de datos cada vez que la despleguemos, y mantener así persistencia entre sesiones.

- Se ha definido el *healthcheck* del que hemos hablado en el apartado del *backend*, que lanza un comando para comprobar que el servicio esté totalmente arrancado. Este comando se va a lanzar un máximo de 3 veces en un intervalo de 15 segundos entre una y otra, siendo la primera 10 segundos después de que arranque el contenedor, y con un timeout de 5 segundos.

Volúmenes

```
volumes:  
  db_data:
```

Es el volumen creado para la base de datos, ya que a diferencia de los volúmenes creados para el *backend*, en los que realmente hemos conectado dos carpetas de nuestro equipo a los contenedores, este volumen se crea dentro de Docker.

Redes

```
networks:  
  nutrimenu-net:
```

Esta es la red creada para conectar todos los contenedores y que no existan problemas de conectividad entre ellos.

Configuración del debugger remoto

Como veremos en el apartado D.4, se van a configurar distintos puertos para el servicio de *backend*, y uno de estos va a ser el puerto de *debug*. Como vamos a ejecutar la aplicación usando Docker Compose y no compilando el código directamente desde nuestro IDE, por defecto no podemos hacer debug como haríamos con cualquier código compilado localmente.

Sin embargo, con los pasos que vamos a ver a continuación [12], **vamos a ser capaces de conectarnos a nuestro entorno contenerizado y realizar debug del código como lo haríamos de forma normal**. En este caso voy a demostrar cómo funcionaría en IntelliJ IDEA, pero mientras que el IDE disponga de un *debugger remoto* y sigamos correctamente las

instrucciones del desarrollador del IDE para usarlo, no debería de haber ningún problema.

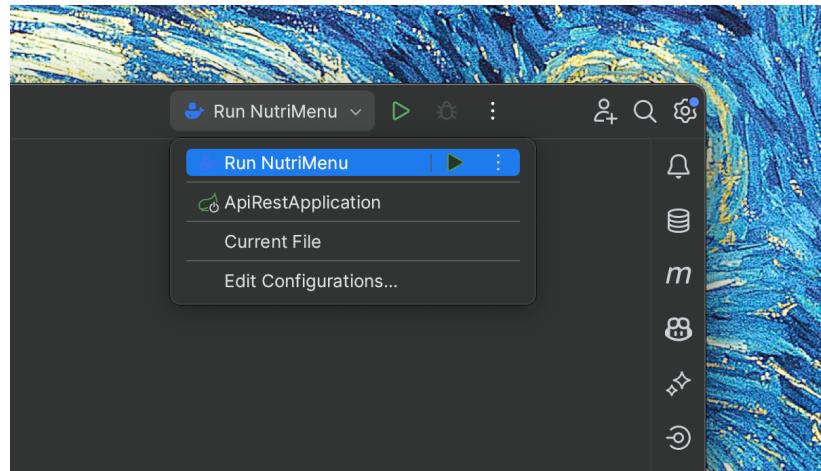


Figura D.2: Listado de todas las configuraciones de ejecución disponibles

Para comenzar, debemos de abrir el proyecto en IntelliJ y en la parte superior, al lado de los botones de Ejecución y Debug, tenemos un selector donde podemos seleccionar la configuración que deseemos. Debemos hacer click en *Edit Configurations....*

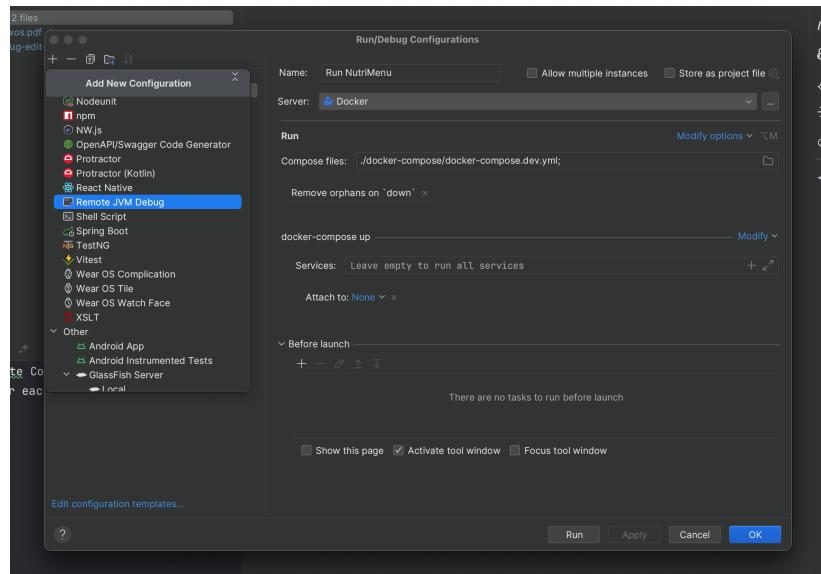


Figura D.3: Selección del tipo de configuración

Debemos hacer click en el icono con un + para añadir una nueva configuración, y seleccionar dentro del desplegable *Remote JVM Debug*.

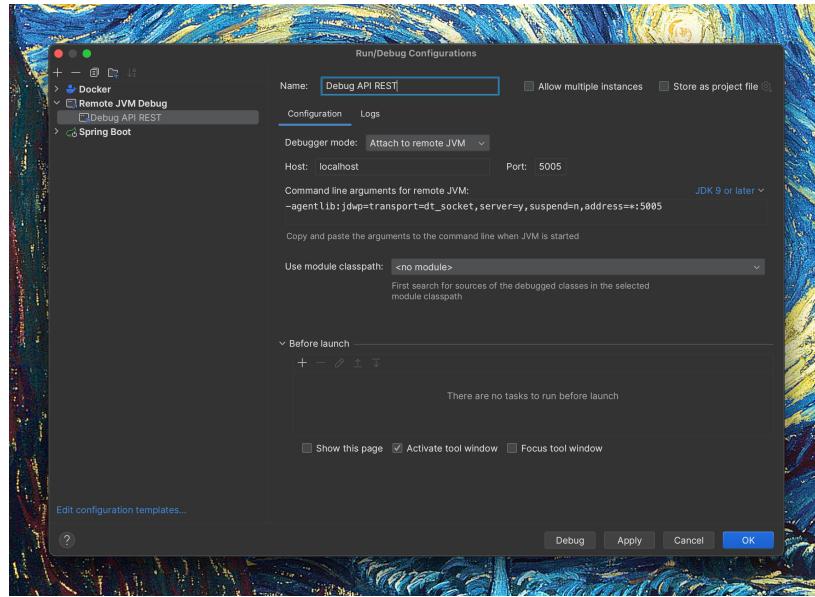


Figura D.4: Creación de la configuración del debugger remoto

A continuación daremos nombre a la configuración, indicaremos el puerto correspondiente al servicio que estemos configurando, y seleccionaremos el módulo de dicho servicio.

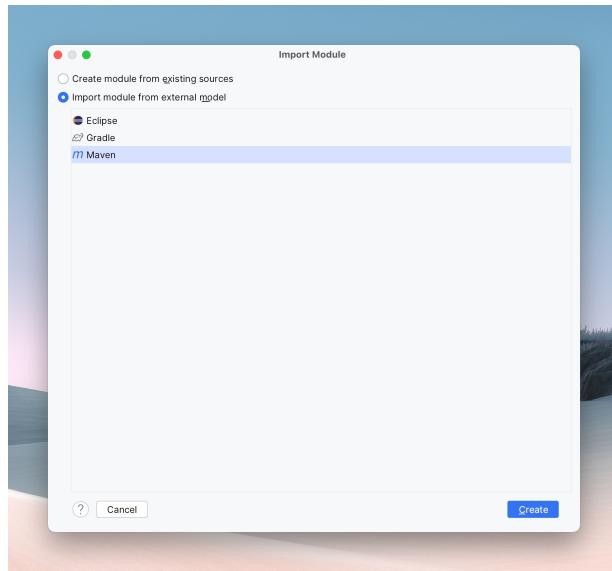


Figura D.5: Creación de un módulo para un servicio

En caso de que el módulo correspondiente no aparezca, deberemos seleccionar en la barra de herramientas *File > New > Module from Existing Sources...*, seleccionar el directorio */webapps/apirest*, y escoger Maven dentro de *Import module from external model*. Si sí que nos aparece el módulo podemos ignorar este paso.

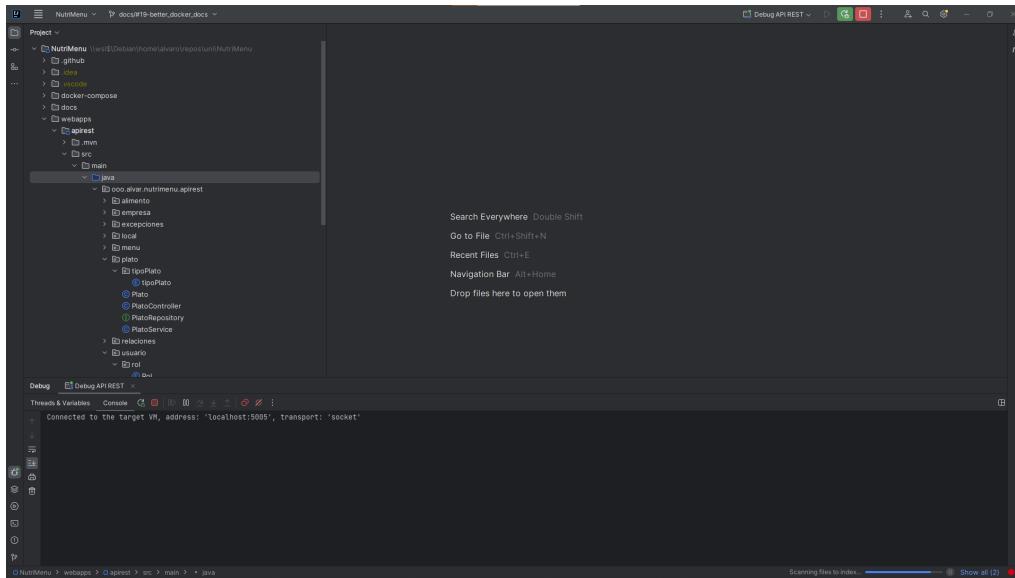


Figura D.6: Debug del servicio

Una vez hayamos completado estos pasos, podemos guardar la configuración y ejecutar el proyecto (asegurándonos de que la configuración deseada está seleccionada) desde el botón de Debug, como haríamos de normal. Si todo ha ido bien, el IDE nos devolverá en la consola el mensaje *Connected to the target VM, address: 'localhost:5005', transport: 'socket'*.

D.4. Compilación, despliegue y ejecución del proyecto

Para realizar la compilación, despliegue y ejecución del proyecto es imprescindible disponer de **Docker CLI** y **Docker Compose** instalados en el equipo. La mejor forma de instalarlos es mediante **Docker Desktop**, ya que se encarga de instalar en el equipo el *daemon* de Docker, Docker CLI, Docker Compose, y demás dependencias y herramientas que no vamos a utilizar para este proyecto, pero que nos pueden ayudar en el futuro. Docker Desktop es compatible con Windows, macOS y Linux, y soporta tanto arquitecturas x86 como ARM64 (Apple Silicon).

Instalación de Docker Desktop

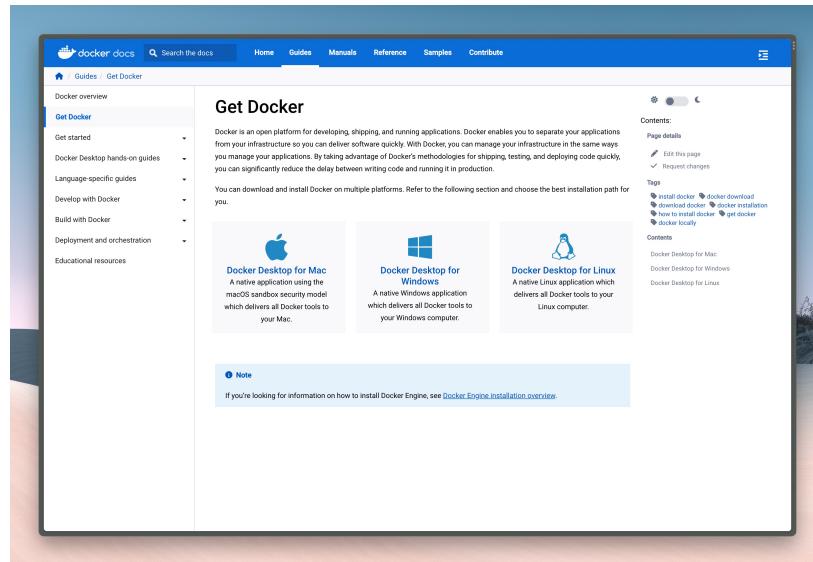


Figura D.7: Página de descarga de Docker Desktop

Para realizar la instalación de Docker Desktop tan sólo debemos dirigirnos a <https://docs.docker.com/get-docker/>, seleccionar la plataforma deseada, y descargar el instalador.

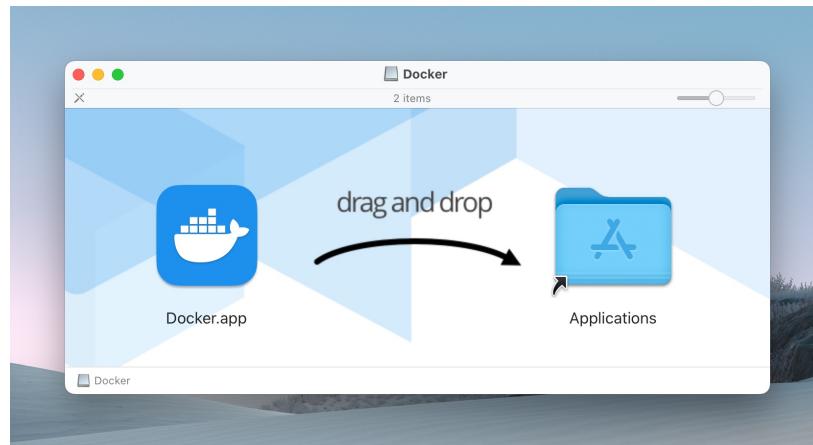


Figura D.8: Instalación de Docker Desktop en Mac

Una vez ejecutemos el instalador (en caso de macOS simplemente se debe arrastrar la aplicación a la carpeta de Aplicaciones), y hayamos seguido todos los pasos hasta finalizar la instalación, nos encontraremos con el panel principal de Docker Desktop.

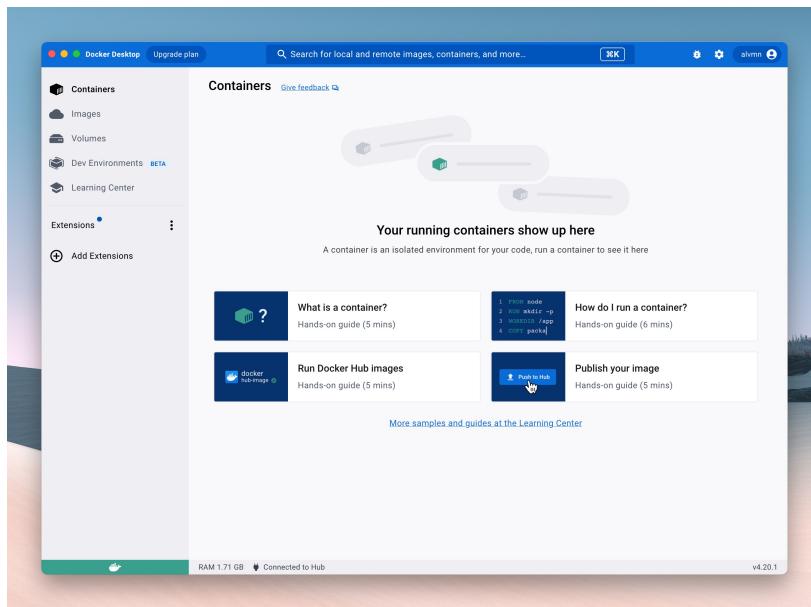


Figura D.9: Ventana principal de Docker Desktop

En esta ventana podemos ver todos los contenedores que están ejecutándose actualmente en el sistema, así como las imágenes descargadas y los volúmenes creados.

Preparación del entorno de desarrollo

Para hacer más sencillo el cambio de valores en distintos parámetros de la aplicación, como el mapeado de puertos o la gestión de secretos, y facilitar así el despliegue de nuevos entornos o la escalabilidad de los servicios, estos valores se han externalizado en un sólo archivo `.env`, que va a ser un archivo que simplemente va a contener las variables junto a sus valores.

Al ser un fichero que contiene secretos, siguiendo lo que dictan las buenas prácticas no se sincronizará con el repositorio, ya que si se tratase de un entorno de producción o el código se hiciera público, cualquier usuario podría obtener acceso a los datos de los usuarios de la aplicación.

Por lo tanto, antes de desplegar los contenedores, para que la aplicación funcione correctamente **se debe crear este archivo `.env` dentro del directorio `/docker-compose`**, y se deben de definir las siguientes variables dentro de él:

```
# Variables de la API REST
```

```

API_WEB_DOCKER_PORT=8080
API_WEB_LOCAL_PORT=8080
API_RELOAD_DOCKER_PORT=35729
API_RELOAD_LOCAL_PORT=35729
API_DEBUG_DOCKER_PORT=5005
API_DEBUG_LOCAL_PORT=5005

# Variables del frontend
FRONTEND_DOCKER_PORT=3000
CHOKIDAR_USEPOLLING=true

# Variables de la base de datos
DATABASE_HOST=mysqlDb
MYSQL_DATABASE=nutri_db
MYSQL_USER=dbadmin
MYSQL_PASSWORD=<MySQL password> # Introduce la password que
desees para el usuario dbadmin
MYSQL_ROOT_PASSWORD=<MySQL root password> # Introduce la password
que deseas para el usuario root
DB_DOCKER_PORT=3306
DB_LOCAL_PORT=3306

```

NOTA: Para crear un documento oculto en Windows, como un fichero .env, podemos usar **NotePad++**. Para ello, abriremos el programa, escribiremos el contenido del fichero .env, daremos a **Save > Save As**, seleccionaremos en tipo de fichero **All files**, y nombraremos el fichero como .env [8].

Los valores pueden ser modificados si es necesario, pero se recomienda usar los valores dados (**es necesario dar los valores que queramos en las variables MYSQL_PASSWORD y MYSQL_ROOT_PASSWORD**). Con estos valores lo que estamos indicando es:

- Los puertos en los que se van a ejecutar cada uno de los servicios dentro de los contenedores (**LOCAL_PORT**) y el puerto desde el cuál vamos a acceder al servicio correspondiente (**DOCKER_PORT**), que en este caso no hay necesidad de diferenciarlos, y en este ejemplo son el **8080** para la API REST, el **3000** para el frontend y el **3306** para la base de datos.
- El puerto en el que va a estar escuchando el plugin *LiveReload* de Spring, que es el encargado de recargar el servidor en caliente cada vez

que se produzcan cambios en el código, y no tener así que recompilar todo el código cada vez que cambiemos algo. En este caso el puerto va a ser el **35729**.

- El puerto que se va a usar para permitir conectar un debugger remoto a la aplicación, y poder así hacer debug desde el IDE que usemos para el desarrollo. El puerto va a ser el **5005** para la API REST.
- Configuración adicional de la parte del frontend, como la variable `CHOKIDAR_USEPOLLING` que indica que usemos el mecanismo *CHOKIDAR* para observar cambios en los archivos fuente, para así poder aplicar los cambios de forma automática y realizar una recarga de la página de forma instantánea. Es necesario usar este mecanismo en aplicaciones React que se encuentren desplegadas en VMs o dockerizadas, como es este caso [3].
- Todas las variables correspondientes a la base de datos, como el **nombre de la instancia**, **nombre de la base de datos**, y **credenciales de los usuarios de MySQL**.

Despliegue del entorno de desarrollo

```

~/Developer/git/NutriMenu/docker-compose git:(feature/#22-frontend) (28.812s)
$ docker compose -f docker-compose.dev.yml build
--> CACHED [api-rest 2/5] RUN apt update && apt -y upgrade          0.0s
--> CACHED [api-rest 3/5] RUN apt install -y inotify-tools dos2unix    0.0s
--> CACHED [api-rest 4/5] RUN mkdir -p /app                           0.0s
--> CACHED [api-rest 5/5] WORKDIR /app                             0.0s
--> [api-rest] exporting to image                                     0.0s
--> => exporting layers                                         0.0s
--> => writing image sha256:517ccf2235641b7329af65f7eba2a10346b3601285d8dc131737e 0.0s
--> => naming to docker.io/library/nutrimenu-dev-api-rest           0.0s
--> [frontend internal] load .dockignore                            0.0s
--> => transferring context: 88B                                    0.0s
--> [frontend internal] load build definition from Dockerfile     0.0s
--> => transferring dockerfile: 173B                                0.0s
--> [frontend internal] load metadata for docker.io/library/node:alpine   1.5s
--> [frontend auth] library/node:pull token for registry-1.docker.io  0.0s
--> [frontend 1/5] FROM docker.io/library/node:alpine@sha256:d75175d449921d06250a 0.0s
--> [frontend internal] load build context                         0.0s
--> => transferring context: 710.79kB                            0.0s
--> => writing image sha256:67b7f928f3f5108e1dd34bd97bc52dc8c1bc37ea711778b32d81d 0.0s
--> => naming to docker.io/library/nutrimenu-dev-frontend          0.0s

~/Developer/git/NutriMenu/docker-compose git:(feature/#22-frontend)±1

```

Figura D.10: Creación de las imágenes de los contenedores Docker

Una vez definidas las variables de entorno, podemos probar que todo funciona correctamente construyendo las imágenes de los contenedores de los servicios. Para ello, abriremos una consola y accederemos al directorio `/docker-compose`, (**TODOS los comandos que veremos a continuación deben ejecutarse dentro del directorio `docker-compose`**), donde ejecutaremos el siguiente comando:

```
docker compose -f docker-compose.dev.yml build
```

```

~/Developer/git/NutriMenu/docker-compose git:(feature/#22-frontend) (28.812s)
docker compose -f docker-compose.dev.yml build
=> => transferring context: 88B 0.0s
=> [frontend internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 173B 0.0s
=> [frontend internal] load metadata for docker.io/library/node:alpine 1.5s
=> [frontend auth] library/node:pull token for registry-1.docker.io 0.0s
=> [frontend 1/5] FROM docker.io/library/node:alpine@sha256:d75175d449921d06250a 0.0s
=> [frontend internal] load build context 0.0s
=> => transferring context: 710.79kB 0.0s
=> CACHED [frontend 2/5] RUN mkdir -p /app 0.0s
=> CACHED [frontend 3/5] WORKDIR /app 0.0s
=> [frontend 4/5] COPY package*.json ./ 0.0s
=> [frontend 5/5] RUN npm install 21.6s
=> [frontend] exporting to image 2.2s
=> => exporting layers 2.2s
=> => writing image sha256:67b7f928f3f5108e1dd34bd97bc52dc8c1bc37ea711778b32d81d 0.0s
=> => naming to docker.io/library/nutrimenu-dev-frontend 0.0s

~/Developer/git/NutriMenu/docker-compose git:(feature/#22-frontend) ±1 (21.619s)
docker compose -f docker-compose.dev.yml up -d

[+] Running 3/3
✓ Container nutrimenu-dev-mysqldb-1  Heal... 21.1s
✓ Container nutrimenu-dev-api-rest-1  Sta... 21.3s
✓ Container nutrimenu-dev-frontend-1  Sta... 11.0s

~/Developer/git/NutriMenu/docker-compose git:(feature/#22-frontend)±2

```

Figura D.11: Despliegue de los contenedores Docker

Este proceso de construcción de imágenes sólo es necesario realizarlo la primera vez que ejecutemos el proyecto, o en caso de realizar cambios en los Dockerfiles, ya que estaríamos cambiando la estructura de las imágenes. Como tanto la capa de *frontend* como la de *backend* disponen de mecanismos para recargarse automáticamente en caso de cambios en el código, no hace falta reiniciar los contenedores cada vez que realicemos cambios en las aplicaciones.

Una vez haya terminado de construir las imágenes, para desplegar los contenedores y levantar el servicio ejecutaremos el siguiente comando:

```
docker compose -f docker-compose.dev.yml up -d
```

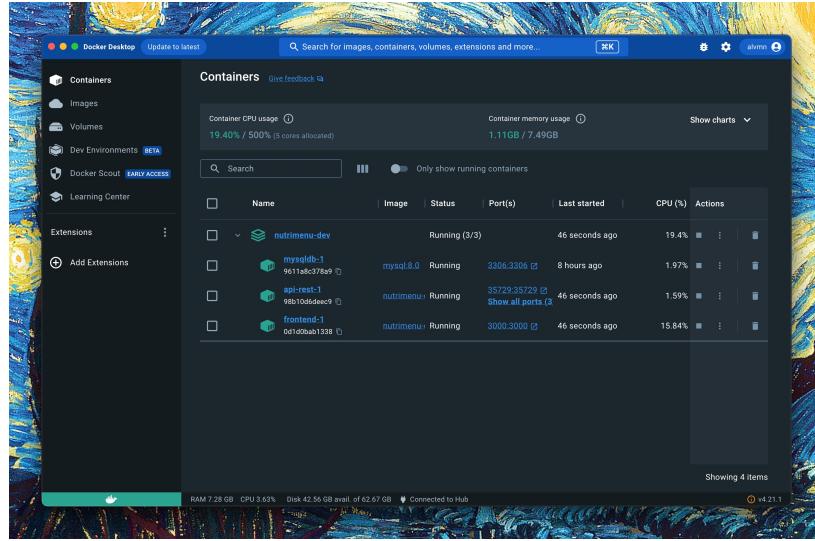


Figura D.12: Ejecución multi-contenedor de la aplicación

Si todo ha ido bien, podremos ver en Docker Desktop los contenedores corriendo con estado *Running*, y ya podremos acceder a la aplicación como haríamos con cualquier tipo de despliegue.

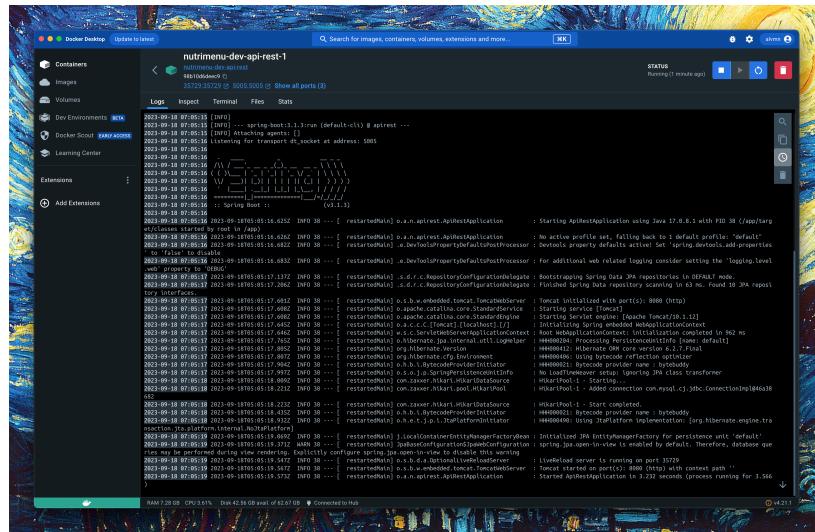


Figura D.13: Logs de la ejecución de un contenedor en Docker Desktop

Para asegurarnos de que no hay ningún tipo de problema, podemos seleccionar los 3 puntos verticales situados a la derecha de cada contenedor,

y hacer click en *View details*, ya que esto nos permite poder ver el log de cada servicio en tiempo real.

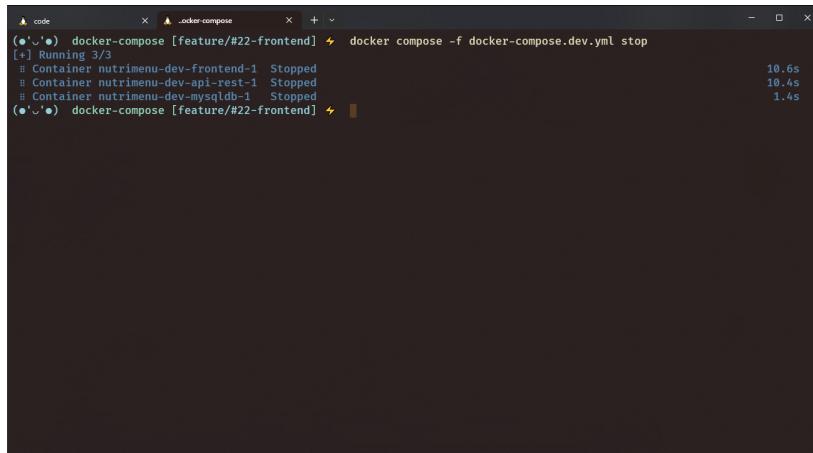


Figura D.14: Parada de los contenedores en ejecución

En caso de querer parar la ejecución de los contenedores, podemos hacerlo con el siguiente comando:

```
docker compose -f docker-compose.dev.yml stop
```

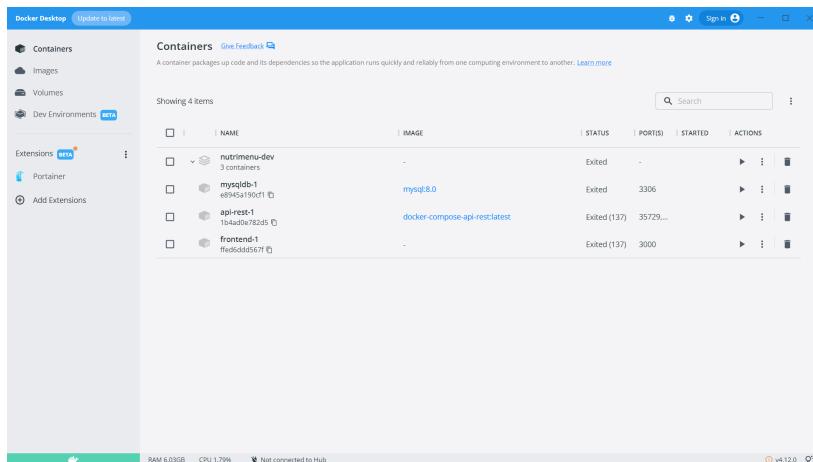


Figura D.15: Contenedores parados pero no borrados en Docker Desktop

Este comando lo que va a hacer es parar los servicios, pero no borrarlos, pudiendo reanudar el servicio otra vez en cualquier momento con `docker compose -f docker-compose.dev.yml up -d`.

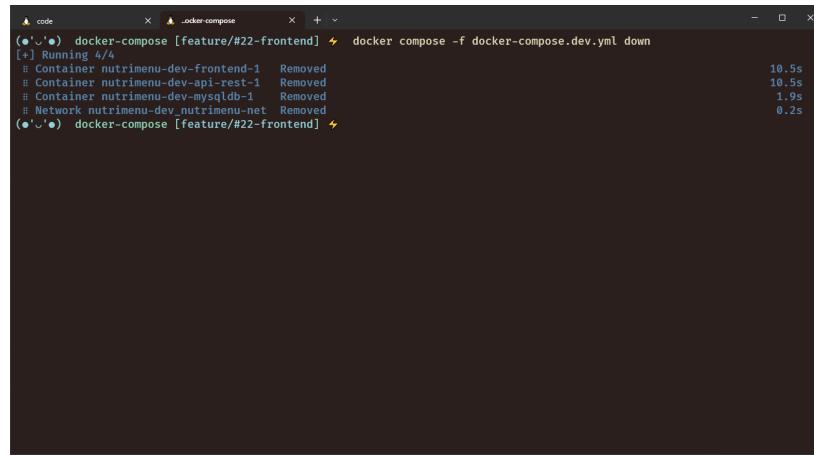


Figura D.16: Parada y borrado de los contenedores en ejecución

En caso de querer borrar los contenedores, en vez de usar el comando que hemos usado arriba para parar los servicios, usariamos el siguiente comando, que para y elimina los contenedores:

```
docker compose -f docker-compose.dev.yml down
```

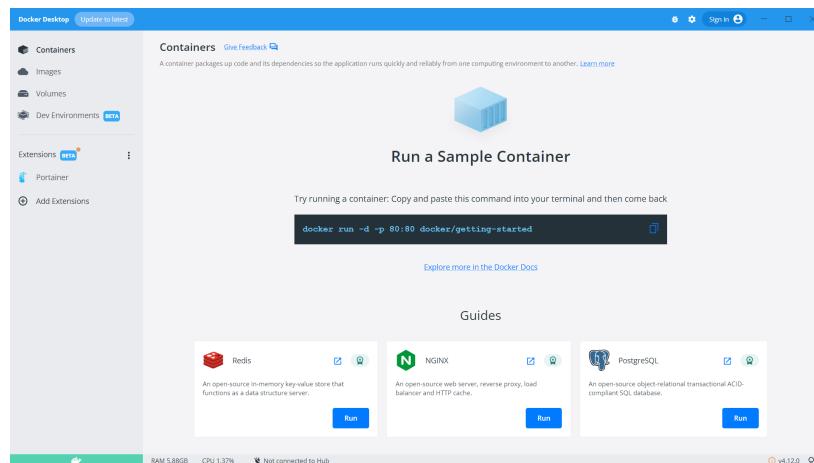


Figura D.17: Contenedores de Docker totalmente eliminados

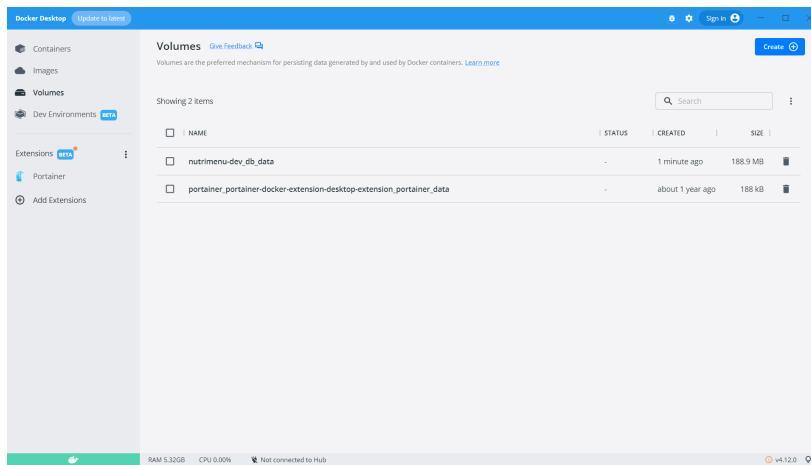


Figura D.18: Persistencia del volumen de la base de datos

En el caso de nuestra aplicación, como el código se monta desde nuestro equipo en todos los servicios como un volumen externo, y la base de datos se encuentra en un volumen de Docker, borrar los contenedores no supondría ninguna pérdida de datos, así que realmente no habría diferencia respecto a simplemente parar los servicios, al menos a nivel de datos. Si reconstruyésemos las imágenes entonces sí que tendría más sentido borrar los contenedores primero, ya que así la próxima vez que los desplegásemos lo harían con la nueva imagen.

```
(⎈ | 'code') code      ✘ .docker-compose      x + ~
(⎈ | 'code') docker-compose [feature/#22-frontend] ✘ docker compose -f docker-compose.dev.yml down --volumes
[+] Running 5/5
  ⚡ Container nutrimenu-dev-frontend-1   Removed
  ⚡ Container nutrimenu-dev-api-rest-1   Removed
  ⚡ Container nutrimenu-dev-mysqldb-1    Removed
  ⚡ Volume nutrimenu-dev_db_data         Removed
  ⚡ Network nutrimenu-dev_nutrimenu-net Removed
(⎈ | 'code') docker-compose [feature/#22-frontend] ✘
```

Figura D.19: Parada y borrado de contenedores y volúmenes existentes

En caso de querer borrar los contenedores y los volúmenes asociados (en nuestro caso, el volumen de la base de datos), deberíamos de ejecutar el siguiente comando:

```
docker compose -f docker-compose.dev.yml down --volumes
```

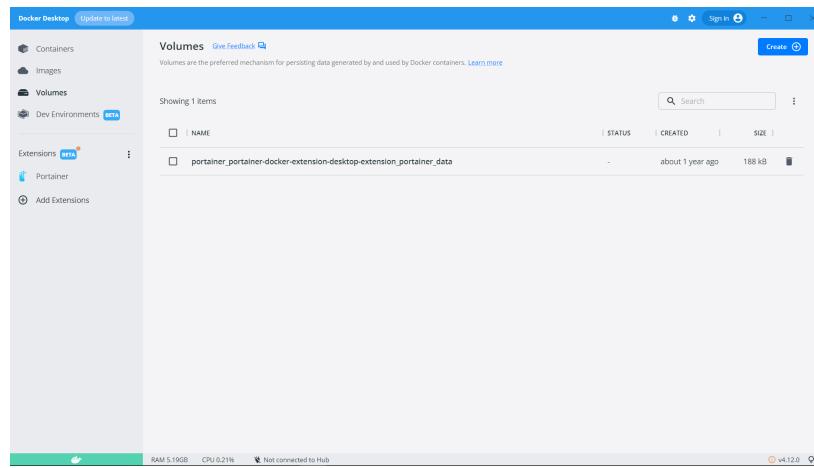


Figura D.20: Volumen de la base de datos totalmente eliminado

Hay que tener en mente que ejecutar este comando **hará que se pierdan todos los datos de la base de datos**.

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

- E.1. Introducción**
- E.2. Requisitos de usuarios**
- E.3. Instalación**
- E.4. Manual del usuario**

Bibliografía

- [1] Tapas Adhikary. Understanding The Container Component Pattern With React Hooks. <https://blog.openreplay.com/understanding-the-container-component-pattern-with-react-hooks/>, Jan 2022.
- [2] Choose an open source license. GNU General Public License v3.0. <https://choosealicense.com/licenses/gpl-3.0/>, 2024.
- [3] Create React App. Advanced Configuration. <https://create-react-app.dev/docs/advanced-configuration/>, 2023.
- [4] Erik Costlow and Simon Ritter. Foojay Podcast no. 4: Why So Many JDKs? <https://foojay.io/today/foojay-podcast-4/>, Oct 2021.
- [5] Docker Docs. Build context. <https://docs.docker.com/build/building/context/>, 2023.
- [6] Docker Docs. Dockerfile reference - WORKDIR. <https://docs.docker.com/engine/reference/builder/#workdir>, 2023.
- [7] Docker Docs. Services top-level element - depends_on. https://docs.docker.com/compose/compose-file/05-services/#depends_on, 2023.
- [8] Ayush Gupta. Stack Overflow: How to save .env file in windows? <https://stackoverflow.com/questions/48770643/how-to-save-env-file-in-windows>, Feb 2018.
- [9] Nam Ha Minh. Spring Boot auto reload changes using LiveReload and DevTools. <https://www.codejava.net/frameworks/spring-boot/spring-boot-auto-reload-changes-using-livereload-and-devtools>, Feb 2020.

- [10] Christian Jaimes. Docker Compose Restart Policies. <https://www.baeldung.com/ops/docker-compose-restart-policies>, Oct 2022.
- [11] Java. JDK Releases. <https://www.java.com/releases/>, 2023.
- [12] Vibhor Mahajan. Developing Spring Boot Applications in Docker locally. <https://medium.com/trantor-inc/developing-spring-boot-applications-in-docker-locally-4ec922f4cb45>, Sep 2021.
- [13] Mariya Aleksandrova Stroyanova. TFG del Grado en Ingeniería Informática - Generador de informes nutricionales de la restauración en centros universitarios. [Documento PDF](#), Feb 2022.
- [14] S. R. Swamy. Review on Spring Boot and Spring Webflux for Reactive Web Development. https://www.researchgate.net/publication/341151097_Review_on_Spring_Boot_and_Spring_Webflux_for_Reactive_Web_Development, 2020.
- [15] takacsmark. Stack Overflow: Why does docker create empty node_modules and how to avoid it? <https://stackoverflow.com/questions/54269442/why-does-docker-create-empty-node-modules-and-how-to-avoid-it>, Jan 2019.
- [16] Gobinda Thakur. Provider Pattern with React Context API. <https://flexiple.com/react/provider-pattern-with-react-context-api>, Nov 2023.
- [17] Wikipedia. BusyBox. <https://en.wikipedia.org/wiki/BusyBox>, 2023.