Service Remote Book

Ivan Graña Hermida
Grado en Ingeniería Informatica

Antonio Oller Arcas

26 de junio de 2023

Licencias alternativas

A) Creative Commons:

B) GNU Free Documentation License (GNU FDL)

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Service Remote Book: tu tienda de libros online
Nombre del autor:	Ivan Graña Hermida
Nombre del consultor:	Antonio Oller Arcas
Fecha de entrega (mm/aaaa):	06/2023
Área del Trabajo Final:	Java EE
Titulación:	*Grado en Ingeniera Informática*

Resumen del Trabajo (máximo 250 palabras):

El objetivo de este trabajo final de grado en Ingeniería Informática, es planificar y ejecutar un proyecto, basado en la creación de una tienda online de venta de libros. Además, se busca cumplir los objetivos marcados mediante el uso de herramientas y tecnologías punteras y, sobre todo, aplicar los conocimientos adquiridos durante el transcurso del grado.

Durante las fases de planificación, análisis y desarrollo, se han aplicado las competencias y conocimientos adquiridos en asignaturas cursas en el grado como por ejemplo: gestión de proyectos, ingeniería de requisitos, programación orientada a objetos, análisis y diseño con patrones, ingeniería de componentes distribuidos, entre otras.

La idea o elección de este proyecto, no es otra que el intentar unificar el medio más usado a día de hoy por las nuevas generaciones, con el mundo de la lectura y literatura de libros. Por ende, da como resultado la creación de una tienda online de venta de libros.

Abstract (in English, 250 words or less):

The objective of this final degree project in Computer Engineering is to plan and execute a project, based on the creation of an online store for selling books. Besides, it searches to achieve the objectives set through the use of tools and technologies on the top and, above all, to apply the knowledge acquired during the course of the degree.

During the planning, analysis and development phases, have been applied the skills and knowledge acquired in subjects studied in the degree such as project management, requirements engineering, object-oriented programming, analysis and design with patterns, component engineering distributed... among others.

I have chosen this project with the idea of trying to unify the most used medium today by the new generations, with the world of reading and literature.
The final product is the creation of an online store for selling books.

Palabras clave (entre 4 y 8):

J2EE, Spring Boot, Thymeleaf, postgreSQL, docker

Índice

1. Introducción ... 1
  1.1 Contexto y justificación del Trabajo ... 1
  1.2 Objetivos del Trabajo ... 2
  1.3 Enfoque y método seguido ... 2
  1.4 Planificación del Trabajo ... 3
  1.5 Breve sumario de productos obtenidos ... 4
  1.6 Breve descripción de los otros capítulos de la memoria ... 4
2. Análisis ... 5
  2.1 Introducción ... 5
  2.2 Toma de requisitos ... 5
  2.3 Requisitos funcionales ... 5
  2.4 Actores ... 6
  2.5 Diagrama de casos de uso ... 7
    2.6.1 Paquete usuario ... 9
    2.6.2 Paquete administrador ... 11
    2.6.3 Paquete cliente ... 15
3. Diseño ... 19
  3.1 Introducción ... 19
  3.2 Arquitectura ... 19
    3.2.1 Planteamiento ... 19
    3.2.2 Arquitectura de microservicios o hexagonal ... 19
    3.2.3 Arquitectura por capas o MVC ... 20
  3.3 Diagrama de clases ... 21
  3.4 Stack tecnológico ... 22
    3.4.1 Desarrollo ... 22
    3.4.1.1 Java JDK 17 ... 22
    3.4.1.2 Spring framework ... 23
    3.4.1.3 JPA (Java Persistence API) ... 23
    3.4.1.4 Hibernate ... 23
    3.4.1.5 PostgreSQL ... 24
    3.4.1.6 Lombok ... 24
    3.4.1.7 MapStructs ... 25
    3.4.1.8 Thymeleaf ... 25
    3.4.2 Sistemas ... 26
    3.4.2.1 Docker ... 26
    3.4.2.2 Adminer ... 26
    3.4.3 Otras herramientas ... 27
    3.4.3.1 IntellIJ IDEA ... 27
    3.4.3.2 Gitlab y Git ... 27
    3.4.3.3 Maven ... 28
    3.4.3.4 Stripe ... 28
  3.5 Arquitectura del proyecto ... 29
    3.5.1 Arquitectura hexagonal backend (API) ... 29
  3.6 Persistencia de datos ... 31
  3.7 Despliegue ... 33
  3.8 Seguridad ... 35
    3.8.1 Base de datos ... 35
    3.8.2 Backend (API) ... 36

4. Implementación 38
4.1 Introducción 38
4.2 Guías de estilo 38
4.3 Patrones SOLID 39
4.4 Patrones de diseño 40
4.5 Repositorio GitLab 41
4.5 Stripe payments 41
5. Manual de uso 43
5.1 Consideraciones previas 44
6. Puntos de mejora y/o pendientes 45
7. Conclusiones 46
8. Glosario 47
9. Bibliografía 48

Lista de figuras

Ilustración 1 - Ciclo de vida agile 2
Ilustración 2 - Planificación inicial 3
Ilustración 3 - Casos de uso del Usuario 7
Ilustración 4 - Casos de uso del Cliente 7
Ilustración 5 - Casos de uso del Administrador 8
Ilustración 6 - Modelo MVC 20
Ilustración 7 - Diagrama de clases del dominio 21
Ilustración 8 - Java 22
Ilustración 9 - Spring 23
Ilustración 10 - Hibernate 23
Ilustración 11 - PostgreSQL 24
Ilustración 12 - Lombok 24
Ilustración 13 - MapStruct 25
Ilustración 14 - Thymeleaf 25
Ilustración 16 - Adminer 26
Ilustración 15 - Docker 26
Ilustración 17 - IDE IntellIJ IDEA 27
Ilustración 18 - GitLab y Git Bash 27
Ilustración 19 - Maven 28
Ilustración 20 - Stripe (pasarela de pago) 28
Ilustración 21 - Arquitectura tecnológica 29
Ilustración 22 - Capas de Arquitectura hexagonal 29
Ilustración 23 - Componentes de cada capa de la arquitectura 30
Ilustración 24 - Diseño conceptual de la BD 31
Ilustración 25 - Entidades de la BD 32
Ilustración 26 - Componentes DB de la capa Infraestructura 32
Ilustración 27 - Diagrama Docker 33
Ilustración 28 - Fichero docker-compose.yml 34
Ilustración 29 - Credenciales DB application.properties 35
Ilustración 30 - Credenciales DB docker-compose.yml 35
Ilustración 31 - Package de configuración/seguridad 36
Ilustración 32 - Método de encriptación de contraseñas 36
Ilustración 33 - Método de seguridad de accesos 37
Ilustración 35 - Clase UserDTO 40
Ilustración 36 - Package aplicando el patrón mapper 40
Ilustración 34 - Clase donde están los Bean con el patrón Singleton 40
Ilustración 37 - Contenido del repositorio GitLab 41
Ilustración 38 - Dashboard de la integración de Stripe 41
Ilustración 39 - Claves de la integración de Stripe 42
Ilustración 40 - Resultado del git clone 43
Ilustración 41 - Despliegue en Docker Desktop 43

1. Introducción

1.1 Contexto y justificación del Trabajo

Este trabajo de fin de grado, el cual de ahora en adelante denominaremos por *TFG* utilizando sus siglas, ha sido desarrollado por el alumno Ivan Graña Hermida, cursando el grado de Ingeniera Informática en la UOC, mediante las metodologías online ofrecidas en el campus virtual.

El objetivo principal de dicho trabajo, no es otra que trasladar todos los conocimientos adquiridos durante estos años en un proyecto final, dando como resultado, la creación de una tienda virtual o e-commerce online de venta de libros.

La elección de este proyecto, viene enfocada por la nueva "cultura" de compra online que se está estableciendo en nuestra sociedad, donde cualquier tienda ahora "*vende de todo"*, y también por el alarmante descenso de la lectura en las nuevas generaciones.

Por este motivo, he decidido enfocar este proyecto juntando estos dos conceptos, la creación de una tienda online enfocada unicamente a la venta de libros, y que tenga un diseño minimalista, intuitivo y atractivo para las nuevas generaciones.

1.2 Objetivos del Trabajo

Los objetivos de este trabajo, no se reservan exclusivamente a la creación de un producto final, sino que también se establecen como objetivos el estudio y crecimiento de dicho producto de cara al futuro, donde se pueda expandir y añadir nuevas funcionalidades.

Teniendo como punto de nexo la creación de un proyecto donde se reuniesen todas las características y funcionalidades necesarias con una tienda Online.

También se han marcado unos objetivos personales, donde las metas finales eran ampliar los conocimientos en la arquitectura Java y las tecnologías empleadas en el área del desarrollo de Java EE, donde también se deben tener en consideración las capacidades personales de auto aprendizaje, control de tiempos de entrega, desarrollo e implementación de un proyecto completo realizado por una única persona, en este caso Ivan Graña Hermida.

1.3 Enfoque y método seguido

Con el fin de trabajar de forma ordenada, se planteo un enfoque y método de trabajo donde el objetivo principal era asegurar un MVP funcional, aplicando la metodología Agile en la cual se iban realizando pequeñas aportaciones de código, donde se fuesen añadiendo funcionalidades y características definidas en el MVP.

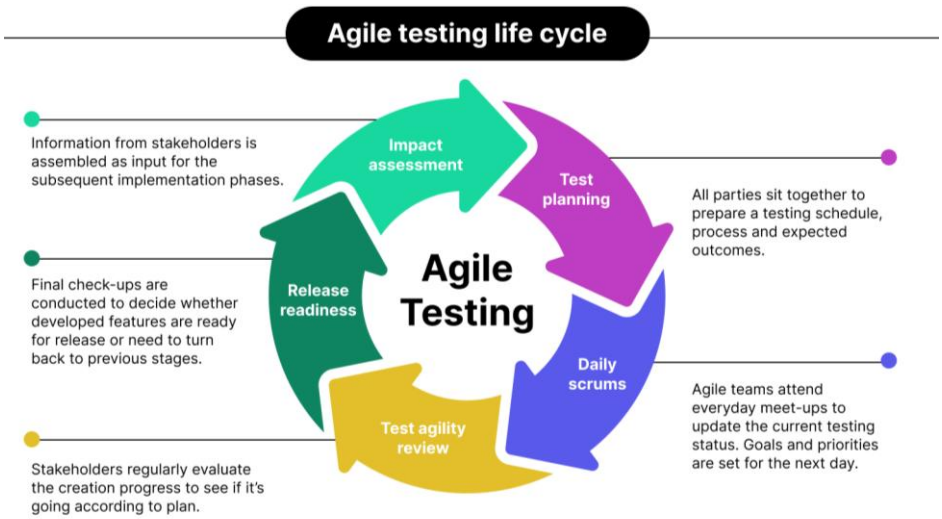

Ilustración 1 - Ciclo de vida agile

1.4 Planificación del Trabajo

La planificación del trabajo realizado y las respectivas ejecuciones de las diferentes fases, están condicionadas por el plan de trabajo de la propia asignatura, en este caso TFG – Java EE, y consta de las siguientes fechas.

Actividad	Plazo de entrega
PEC1 – Plan de trabajo	01/03/2023 -- 14/03/2023
PEC2 – Análisis y diseño	15/03/2023 -- 14/04/2023
PEC3 – Implementación	15/04/2023 -- 05/06/2023
Memoria y presentación	06/06/2023 -- 26/06/2023
Tribunal	03/07/2023 -- 09/07/2023

Teniendo en cuenta los plazos indicados en la tabla 1, se genera el diagrama de Gantt donde se especifican todas las tareas a cabo para actividad.

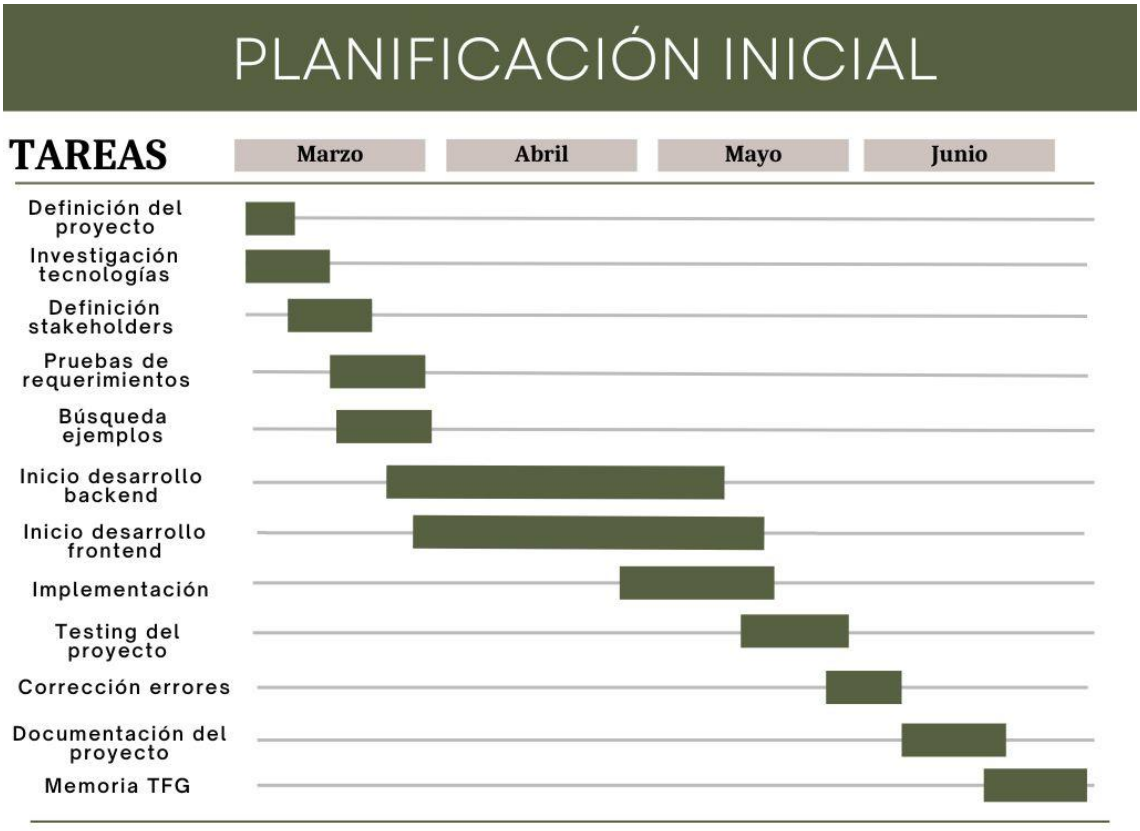

Ilustración 2 - Planificación inicial

1.5 Breve sumario de productos obtenidos

Aunque el objetivo era crear un producto final, conformado por varios micro servicios en una arquitectura hexagonal, que incluyera servicio de mensajería, avisos de stock, test unitarios, etc.… pero, el tiempo disponible para el desarrollo del producto únicamente me ha permitido crear un MVP funcional, sin las funcionalidades anteriormente mencionadas.

Este MVP está compuesto de la aplicación del backend, las vistas del frontend y la infraestructura necesaria para su virtualización mediante el uso de contenedores Docker.

Además, el producto final viene acompañado por otros dos productos obtenidos: la memoria final y el video de la presentación.

1.6 Breve descripción de los otros capítulos de la memoria

El presente documento está compuesto por los siguientes capítulos:

1. Análisis
2. Diseño
3. Implementación
4. Puntos de mejora y/o pendientes
5. Documentación de uso
6. Valoración final

2. Análisis

2.1 Introducción

En esta sección, en la que se aborda el análisis de la aplicación, se abordarán aspectos como: identificar los principales stakeholders y sus respectivos casos de uso, utilizando el análisis orientado a objetos.

2.2 Toma de requisitos

Para especificar los requerimientos que debía tener la aplicación, y detectar los posibles stakeholders que la iban a utilizar, se llevo a cabo un análisis en algunas de las plataformas ya existentes, para detectar dichos conceptos, llegando a la conclusión que los principales requisitos y stakeholders serían los definidos y argumentados en los siguientes sub apartados.

2.3 Requisitos funcionales

- Poder acceder a la aplicación mediante un método de autenticación (obtener un ID de sesión): el usuario ha de acceder mediante las credenciales {email y password} y se le devuelve un ID de sesión.
- Los datos sensibles como las contraseñas y los métodos de pagos (ley de protección de datos): se guardarán en la base de datos con un cifrado.
- Gestión del catalogo: esta acción se centra en la creación, modificación y gestión de los libros y su stock. Este requisito implica ser administrador mediante el rol de administrador en el sistema.
- Poder realizar búsquedas en el catalogo por nombre, autor o categoría del libro.
- Poder acceder a la ficha/detalle de un libro y ver su información:
    - Datos generales (autor, categoría, descripción)
    - Unidades disponibles (stock)
    - Precio
- Si el usuario esta registro, añadiendo el requisito anterior, la opción de añadir el libro al carrito de la compra.

- Gestión del perfil de usuario: esta acción se centra en la parte del usuario donde puede ver su información, actualizarla y ver sus pedidos realizados.
- Gestión del carrito de la compra: esta acción se centra en la parte del usuario donde pueda ver el carrito de la compra creado, ver los libros añadidos, eliminarlos y confirmar el pedido.

2.4 Actores

- Administrador
- Usuario u usuario no registrado,
- Cliente u usuario registrado.

Por este motivo, se ha decidido que la aplicación funcionará mediante roles para diferenciar las funcionalidades a mostrar.

Por lo tanto, tenemos:

Stakeholders	
Administrador	Se encarga de dar de alta los libros, gestionarlos y organizar el stock de la tienda.
Usuario	Puede buscar libros y ver sus detalles.
Cliente	Puede buscar libros, ver sus detalles, añadirlos al carrito, realizar pedidos, ver su perfil, actualizar, ver sus pedidos realizados y ver las facturas de sus pedidos.

2.5 Diagrama de casos de uso

Se generan los casos de uso, para cada uno de los actores, extraídos del análisis de requisitos y funcionalidades de aplicaciones existentes.

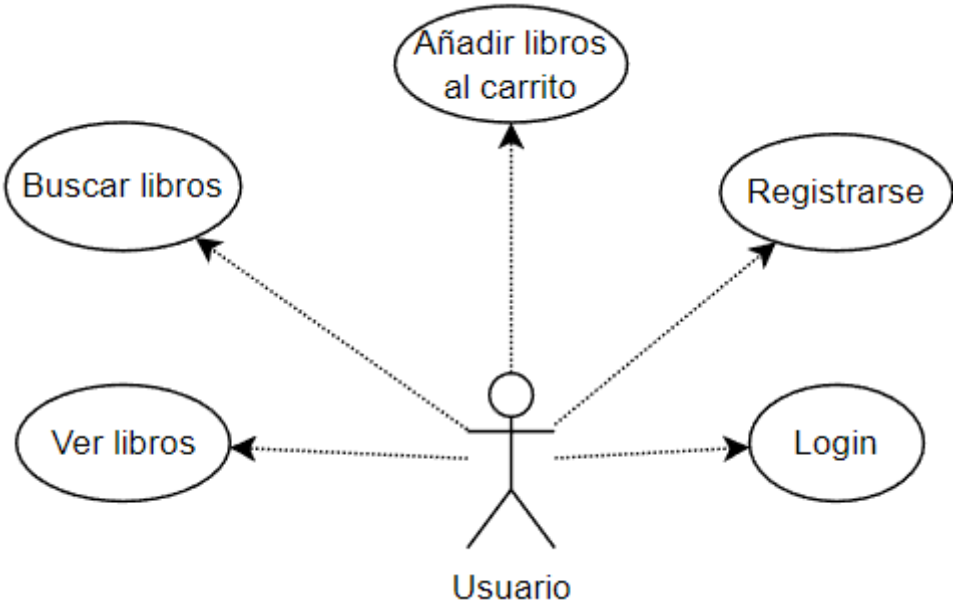

Ilustración 3 - Casos de uso del Usuario

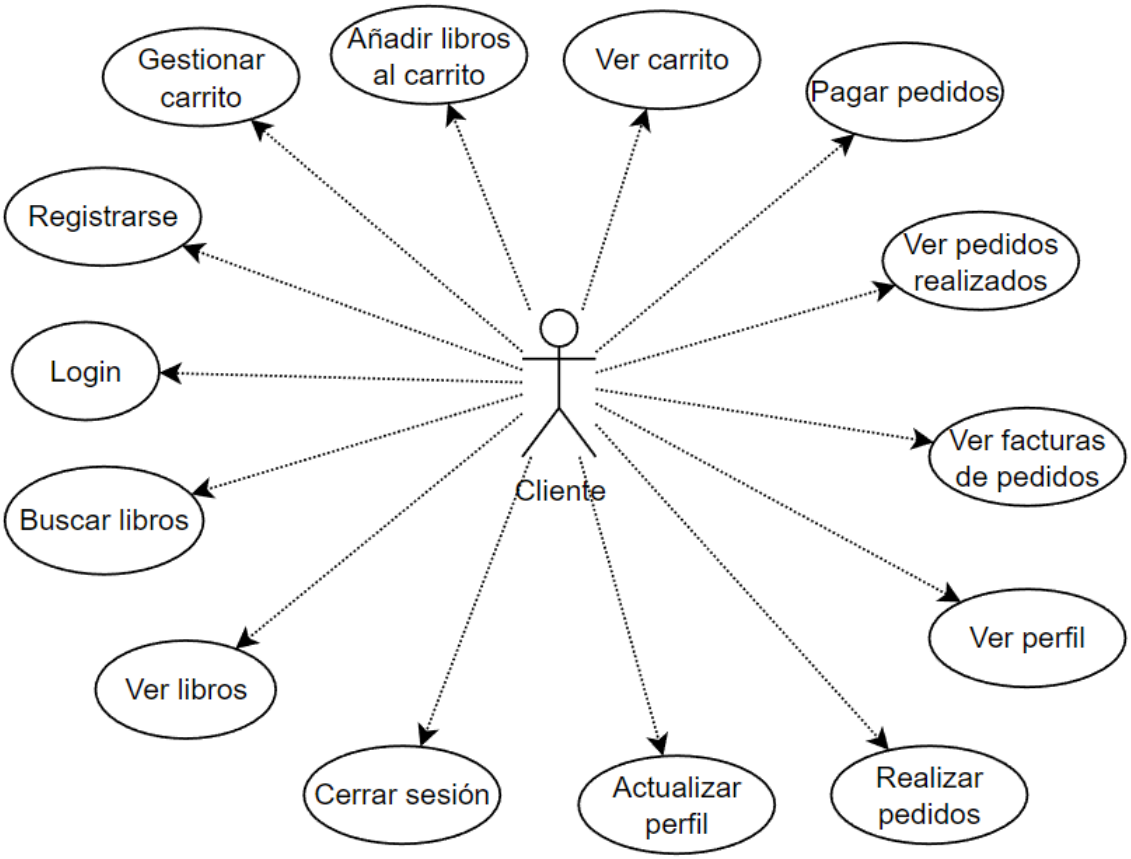

Ilustración 4 - Casos de uso del Cliente

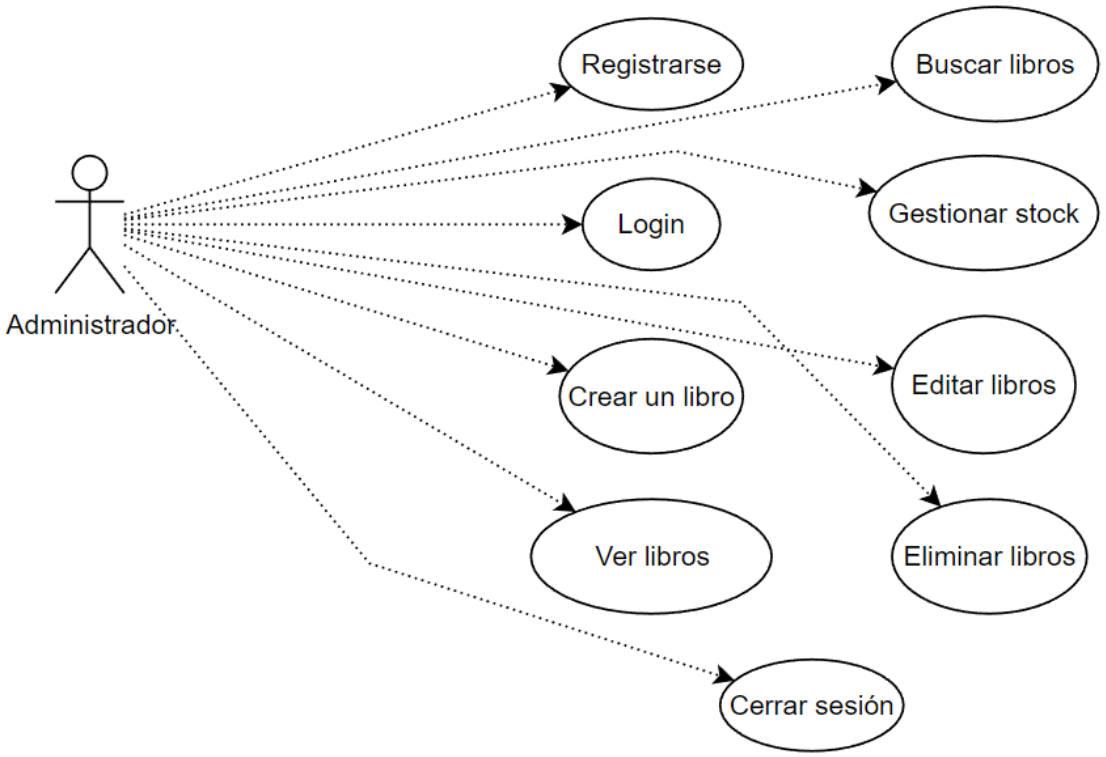

Ilustración 5 - Casos de uso del Administrador

2.6 Fichas de casos de uso

A partir de los anteriores diagramas, se desgranan los casos de uso por actor, excluyendo el primer caso de uso CU1 – identificarse en el sistema, que actuará de precondición para el resto de casos en administrador y cliente.

Caso de uso	**CU1 – Identificarse en el sistema**	
Descripción	El usuario debe identificarse en la aplicación, para acceder a sus funcionalidades.	
Actores	Administrador, Cliente	
Precondición	El usuario no se ha identificado previamente.	
Resultado	El usuario obtiene acceso a sus funcionalidades según el rol especificado.	
Escenario principal	**Paso**	**Descripción**
	1	La aplicación le solicita al usuario su nombre de usuario y su contraseña
	2	El usuario introduce sus credenciales
	3	La aplicación valida las credenciales y redirige al usuario a su vista correspondiente
Escenario alternativo	3.1	El sistema comprueba que las credenciales no corresponden con ningún usuario. Se muestra un mensaje por pantalla y se vuelve al paso 2.

2.6.1 Paquete usuario

A continuación se detallan los casos de uso pertenecientes al usuario.

Caso de uso	**CU2 – Ver libros**	
Descripción	El usuario es capaz de poder visualizar el catalogo disponible en la tienda.	
Actores	Usuario	
Precondición	No existe precondición.	
Resultado	El usuario visibiliza los libros disponibles.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario accede a la página principal.
	2	El usuario visualiza el catalogo.
	3	El usuario accede al detalle de un libro

Caso de uso	**CU3 – Buscar libros**	
Descripción	El usuario utiliza el buscador de la aplicación para buscar un libro en concreto, ya sea por nombre del libro, autor o categoría del mismo.	
Actores	Usuario	
Precondición	El usuario debe saber que libro desea buscar	
Resultado	El usuario visualiza el libro a buscar	
Escenario principal	**Paso**	**Descripción**
	1	El usuario accede a la página principal.
	2	El usuario busca el libro.
	3	El sistema devuelve el libro buscado.
	4	El usuario accede al CU-2 Ver libro
Escenario alternativo	3.1	El libro buscado no se encuentra en el catalogo, y el sistema muestra todos los libros disponibles

Caso de uso	CU4 – Añadir libros al carrito	
Descripción	El usuario añade un libro al carrito de la compra "*temporal"*.	
Actores	Usuario	
Precondición	El usuario ha encontrado el libro que sea comprar.	
Resultado	El usuario añade el libro y las unidades al carrito.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario está en la vista del detalle del libro.
	2	Selecciona cuantas unidades quiere y lo añade al carrito
	3	El libro se ha añadido al carrito de la compra "*temporal"*
Escenario alternativo	2.1	El libro buscado, no tiene stock en ese momento. El sistema muestra un mensaje por pantalla y no deja añadirlo al carrito.

Caso de uso	CU5 – Registrarse	
Descripción	El usuario desea validar su carrito de la compra y/o desea registrarse por su propia iniciativa	
Actores	Usuario	
Precondición	El usuario desea validar su carrito de la compra.	
Resultado	El usuario se registra correctamente en el sistema.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario accede a la página principal.
	2	El usuario hace *click* en el botón de registrarse
	3	El sistema le devuelve la vista con el formulario de registro
	4	El usuario se registra correctamente.
Escenario alternativo	3.1	El usuario introduce datos erróneos, y el sistema le avisa con un mensaje por pantalla.

Con los casos de uso 1 y 5, el Usuario se convierte en Cliente.

2.6.2 Paquete administrador

A continuación se detallan los casos de uso pertenecientes al administrador.

Caso de uso	**CU6 – Crear/Añadir nuevo libro**	
Descripción	El administrador añade un nuevo libro al catalogo disponible	
Actores	Administrador	
Precondición	Esta acción solo la puede realizar el Administrador.	
Resultado	El administrador registra un nuevo libro en el sistema.	
Escenario principal	**Paso**	**Descripción**
	1	El administrador accede a su vista personalizada.
	2	El administrador hace *click* en el botón de Inventario → añadir nuevo libro
	3	El sistema le devuelve la vista con el formulario de creación del libro.
	4	El sistema guarda correctamente los datos introducidos.
Escenario alternativo	3.1	El administrador introduce datos erróneos, y el sistema le avisa con un mensaje por pantalla.

Caso de uso	**CU7 – Ver libros creados por el administrador**	
Descripción	El administrador visualiza los libros creados por él.	
Actores	Administrador	
Precondición	El administrador debe haber creado como mínimo 1 libro.	
Resultado	El administrador visualiza sus libros	
Escenario principal	**Paso**	**Descripción**
	1	El administrador va su vista personalizada
	2	Accede a la página de inicio
	3	Visualiza los libros y puede acceder a sus detalles
	4	El administrador también puede realizar la misma acción desde el botón de Inventario.
Escenario alternativo	3.1	Si no existe ningún libro creado por él, el sistema no le mostrará nada.

Caso de uso	CU7 – Gestionar stock/Inventario	
Descripción	El administrador visualiza los libros creados por él, y quiere gestionar el stock del catalogo.	
Actores	Administrador	
Precondición	El administrador debe haber creado como mínimo 1 libro.	
Resultado	El administrador gestiona el stock, añadiendo nuevos libros, editándolos, añadiendo nuevas unidades disponibles en stock.	
Escenario principal	Paso	Descripción
	1	El administrador va su vista personalizada
	2	Accede a la opción de Inventario
	3	El sistema le carga la vista del Inventario
	4	El administrador selecciona una de las opciones disponibles.
Escenario alternativo	3.1	Si no existe ningún libro creado por él, el sistema no le mostrará nada, y no podrá realizar ninguna acción.

Caso de uso	CU7.1 – Gestionar Inventario – Añadir nuevas unidades	
Descripción	El administrador visualiza los libros creados por él, y quiere gestionar el stock del catalogo.	
Actores	Administrador	
Precondición	El administrador debe haber creado como mínimo 1 libro.	
Resultado	El administrador añade nueva unidades disponibles a la venta de un libro en concreto.	
Escenario principal	Paso	Descripción
	1	El administrador va su vista personalizada
	2	Accede a la opción de Inventario
	3	El sistema le carga la vista del Inventario
	4	El administrador selecciona el libro para gestionar su stock.
	5	El sistema le carga la vista para añadir las nuevas unidades.
	6	El administrador introduce la cantidad de unidades más un mensaje para el registro del stock.
	7	El sistema guarda las nuevas unidades al sistema y las mostrará en la vista del detalle del libro.
Escenario alternativo	3.1	Si el administrador no introduce un número valido, el sistema le mostrará un mensaje por pantalla.

Caso de uso	CU7.2 – Gestionar Inventario – Editar un libro	
Descripción	El administrador visualiza los libros creados por él, y quiere gestionar el stock del catalogo.	
Actores	Administrador	
Precondición	El administrador debe haber creado como mínimo 1 libro.	
Resultado	El administrador actualiza la información de un libro.	
Escenario principal	**Paso**	**Descripción**
	1	El administrador va su vista personalizada
	2	Accede a la opción de Inventario
	3	El sistema le carga la vista del Inventario
	4	El administrador selecciona el libro que quiere actualizar.
	5	El sistema le carga la vista con el formulario y los datos actuales del libro seleccionado.
	6	El administrador actualiza los campos que él requiera oportunos.
	7	El sistema guarda la nueva información y la actualiza en la base de datos.
Escenario alternativo	3.1	Si el administrador no introduce nuevos datos, el sistema no realizará ninguna acción.

Caso de uso	CU7.3 – Gestionar Inventario – Eliminar un libro	
Descripción	El administrador desea eliminar un libro creado por el, del catalogo.	
Actores	Administrador	
Precondición	El administrador debe haber creado como mínimo 1 libro.	
Resultado	El administrador elimina el libro seleccionado	
Escenario principal	**Paso**	**Descripción**
	1	El administrador va su vista personalizada
	2	Accede a la opción de Inventario
	3	El sistema le carga la vista del Inventario
	4	El administrador selecciona el libro a eliminar.
	5	El administrador hace "*click"* en el botón de eliminar
	6	El sistema le muestra por pantalla un mensaje de confirmación de la acción.
	7	El administrador confirma la acción.
	8	El sistema borra el libro, y redirige al administrador a la vista del Inventario
Escenario alternativo	3.1	Si no existe ningún libro creado por él, el sistema no le mostrará ninguna acción a realizar.

Caso de uso	**CU8 – Cerrar sesión**	
Descripción	El administrador quiere cerrar su sesión en el sistema.	
Actores	Administrador	
Precondición	El administrador debe estar previamente *logeado* en el sistema	
Resultado	El administrador cierra su sesión, y el sistema libera la sesión establecida.	
Escenario principal	**Paso**	**Descripción**
	1	El administrador va su vista personalizada
	2	Accede a la barra de navegación
	3	Hace *"click"* en la opción de Cerrar sesión
	4	El sistema cierra la sesión.
Escenario alternativo	3.1	Si el administrador no tiene una sesión iniciada en el sistema, este no podrá cerrarla.

2.6.3 Paquete cliente

A continuación se detallan los casos de uso pertenecientes al cliente.

Caso de uso	**CU9 – Ver perfil**	
Descripción	El usuario debe poder ver su perfil y la información rellenada en el formulario de registro.	
Actores	Usuario registrado (Cliente)	
Precondición	El usuario debe estar previamente registrado y *logeado* en el sistema	
Resultado	El usuario puede ver su información en la vista de perfil.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario va su vista personalizada
	2	Accede a la barra de navegación
	3	Hace *"click"* en la opción de Mi perfil
	4	El sistema carga la vista con todos sus datos
	5	El usuario ve sus datos cargados
Escenario alternativo	2.1	Si el usuario no tiene una sesión iniciada en el sistema, este no podrá actualizar su perfil.

Caso de uso	**CU10 – Actualizar perfil**	
Descripción	El usuario debe poder actualizar su información de su perfil.	
Actores	Usuario registrado (Cliente)	
Precondición	El usuario debe estar previamente registrado y *logeado* en el sistema y CU9.	
Resultado	El usuario actualiza la información deseada con respecto a su perfil.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario va su vista personalizada
	2	Accede a la barra de navegación
	3	Hace *"click"* en la opción de Mi perfil
	4	El sistema carga la vista con todos sus datos
	5	El usuario hace "*click"* en el botón de Actualizar mi perfil
	6	El sistema le carga la vista personalizada, recuperando todos los campos y sus valores.
	7	El usuario actualiza los campos que el crea oportunos
	8	El sistema actualiza los datos, mostrando un mensaje por pantalla.

Escenario alternativo	2.1	Si el usuario no tiene una sesión iniciada en el sistema, este no podrá actualizar su perfil.
	8.1	Si el usuario finalmente no quiere actualizar los campos, el sistema no realizará dicha operación.
Caso de uso	**CU11 – Ver carrito**	
Descripción	El usuario debe poder ver el estado actual del carrito de la compra.	
Actores	Usuario registrado (Cliente)	
Precondición	El usuario debe estar previamente registrado y *logeado* en el sistema, y en el carrito debe haber libros añadidos.	
Resultado	El usuario ver la cantidad de libros que tiene en el carrito pendientes de realizar un pedido.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario va su vista personalizada
	2	Accede a la barra de navegación
	3	Hace *"click"* en la opción de Carrito
	4	El sistema carga la vista con todos los datos.
	5	El usuario ve la información de los libros añadidos en el carrito y el botón de realizar pedido.
Escenario alternativo	2.1	Si el usuario no tiene una sesión iniciada en el sistema, este no podrá ver su carrito.
	5.1	Si el usuario no tiene ningún libro añadido en el carrito, el sistema cargará la vista vacía.

Caso de uso	**CU12 – Gestionar carrito**	
Descripción	El usuario debe poder ver el estado actual del carrito de la compra y gestionarlo, pudiendo añadir o quitar libros.	
Actores	Usuario registrado (Cliente)	
Precondición	El usuario debe estar previamente registrado y *logeado* en el sistema, y en el carrito debe haber libros añadidos.	
Resultado	El usuario ver la cantidad de libros que tiene en el carrito pendientes de realizar un pedido.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario va su vista personalizada
	2	Accede a la barra de navegación
	3	Hace *"click"* en la opción de Carrito
	4	El sistema carga la vista con todos los datos.
	5	El usuario ve la información de los libros añadidos en el carrito y el botón de realizar pedido.
	6	El usuario añade o quitar unidades del carrito.
Escenario alternativo	2.1	Si el usuario no tiene una sesión iniciada en el sistema, este no podrá ver su carrito.
	5.1	Si el usuario no tiene ningún libro añadido en el carrito, el sistema cargará la vista vacía.

Caso de uso	CU13 – Confirmar pedido	
Descripción	El usuario debe poder confirmar el pedido que tiene en el carrito de la compra.	
Actores	Usuario registrado (Cliente)	
Precondición	El usuario debe estar previamente registrado y *logeado* en el sistema, y en el carrito debe haber libros añadidos.	
Resultado	El usuario confirma que quiere realizar el pedido de todos los libros que tiene actualmente en el carrito de la compra.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario va su vista personalizada
	2	Accede a la barra de navegación
	3	Hace *"click"* en la opción de Carrito
	4	El sistema carga la vista con todos los datos.
	5	El usuario ve la información de los libros añadidos en el carrito y hace "*click"* en el botón de tramitar pedido.
	6	El sistema carga la siguiente vista personalizada al usuario.
Escenario alternativo	2.1	Si el usuario no tiene una sesión iniciada en el sistema, este no podrá ver su carrito.
	5.1	Si el usuario no tiene ningún libro añadido en el carrito, el sistema cargará la vista vacía.

Caso de uso	CU14 – Tramitar pedido	
Descripción	El usuario debe poder realizar el pago del pedido mediante la pasarela de pago y el método de pago seleccionado.	
Actores	Usuario registrado (Cliente)	
Precondición	El usuario debe estar previamente registrado y *logeado* en el sistema, y CU13.	
Resultado	El usuario confirma que quiere realizar el pedido, y realiza el pago mediante la pasarela de pagos.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario está en la vista confirmación del pedido y hace *"click"* en el botón de Tramitar pedido.
	2	El sistema carga la vista de la pasarela de pagos.
	3	El usuario ve la vista, con la información del pedido y debe seleccionar el método de pago e introducir los datos bancarios.
	4	El sistema procesa el pago.
	5	El sistema muestra una vista personalizada conforme el pago se ha realizado correctamente y redirige al usuario a su vista personalizada de usuario.
Escenario alternativo	2.1	Si el usuario no tiene una sesión iniciada en el sistema, no podrá tramitar el pedido.

	3.1	Si el usuario introduce datos bancarios incorrectos, el sistema le mostrará un aviso
	4.1	Si se produce algún error con la validación del pago, el sistema anula el pago y muestra un mensaje por pantalla.

Caso de uso	**CU15 – Ver pedidos**	
Descripción	El usuario debe poder visualizar el listado de pedidos realizados	
Actores	Usuario registrado (Cliente)	
Precondición	El usuario debe estar previamente registrado y *logeado* en el sistema, y debe haber realizado como mínimo 1 pedido.	
Resultado	El usuario ve el listado de pedidos realizados.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario va su vista personalizada
	2	Accede a la barra de navegación
	3	Hace *"click"* en la opción de Mis pedidos
	4	El sistema carga la vista con todos los datos.
	5	El usuario ve la información de los pedidos realizados.
Escenario alternativo	2.1	Si el usuario no tiene una sesión iniciada en el sistema, este no podrá visualizar sus pedidos.
	5.1	Si el usuario no tiene ningún pedido realizado, el sistema no le mostrará nada.

Caso de uso	**CU15 – Ver factura**	
Descripción	El usuario debe poder visualizar y descargarse la factura del pedido seleccionado	
Actores	Usuario registrado (Cliente)	
Precondición	El usuario debe estar previamente registrado y *logeado* en el sistema, y debe haber realizado como mínimo 1 pedido.	
Resultado	El usuario ve el listado de pedidos realizados, selecciona un pedido en concreto y el sistema le muestra la factura emitida.	
Escenario principal	**Paso**	**Descripción**
	1	El usuario va su vista personalizada
	2	Accede a la barra de navegación
	3	Hace *"click"* en la opción de Mis pedidos
	4	El sistema carga la vista con todos los datos.
	5	El usuario ve la información de los pedidos realizados.
	6	El usuario selecciona un pedido en concreto, y hace *"click"* en el botón de Ver factura
	7	El sistema le abre una nueva pestaña en el navegador con la factura cargada.
Escenario alternativo	2.1	Si el usuario no tiene una sesión iniciada en el sistema, este no podrá ver sus pedidos.

3. Diseño

3.1 Introducción

En esta sección, se detalla el diseño que será aplicado y explicado en la fase de implementación, con las decisiones tomadas a la hora de escoger la arquitectura y el stack tecnológico.

3.2 Arquitectura

3.2.1 Planteamiento

Según lo explicado en la fase de análisis, se puede observar que la aplicación debe ofrecer vistas personalizadas definidas por el tipo de rol del usuario, y dichos roles tendrán diferentes funcionalidades disponibles.

Por este motivo, el resultado obtenido en dicha fase es la de un sistema planteado por capas de iteración entre el backend y el frontend, conjuntamente con las consultas a la base de datos.

3.2.2 Arquitectura de microservicios o hexagonal

La arquitectura de microservicios nos permite aplicar una división de una aplicación en múltiples aplicaciones más pequeñas, donde cada una es independiente de la otra y teniendo cohesionadas sus funcionalidades y responsabilidades.

Para poder aplicar dicha arquitectura, se definen y diseñan dos 2 microservicios

- ms-infra: su responsabilidad es levantar mediante el fichero docker-compose.yml los servicios necesarios para el correcto funcionamiento de la aplicación.

- tfg-bookstore: la responsabilidad de este microservicio no es otra que contener toda la lógica de negocio, aplicando la arquitectura hexagonal en la parte del backend y las vistas relacionadas con el frontend.

3.2.3 Arquitectura por capas o MVC

El modelo de arquitectura seleccionado para la implementación de las aplicaciones spring boot, es el modelo por capas o MVC. Concretamente, el modelo contendrá las siguientes capas:

- Controlador: esta capa es la encargada de gestionar las peticiones HTTP mediante las interfaces REST, para posteriormente, trasladar la petición a la capa de servicio.
- Servicio: esta capa es la encargada de gestionar e implementar toda la lógica de negocio.
- Repositorio: capa encargada de gestionar las operaciones de persistencia sobre los objetos de la capa de dominio.
- Vistas: capa encargada de gestionar y visualizar toda la información que se solicite mediante las peticiones HTTP mediante la capa controlador.

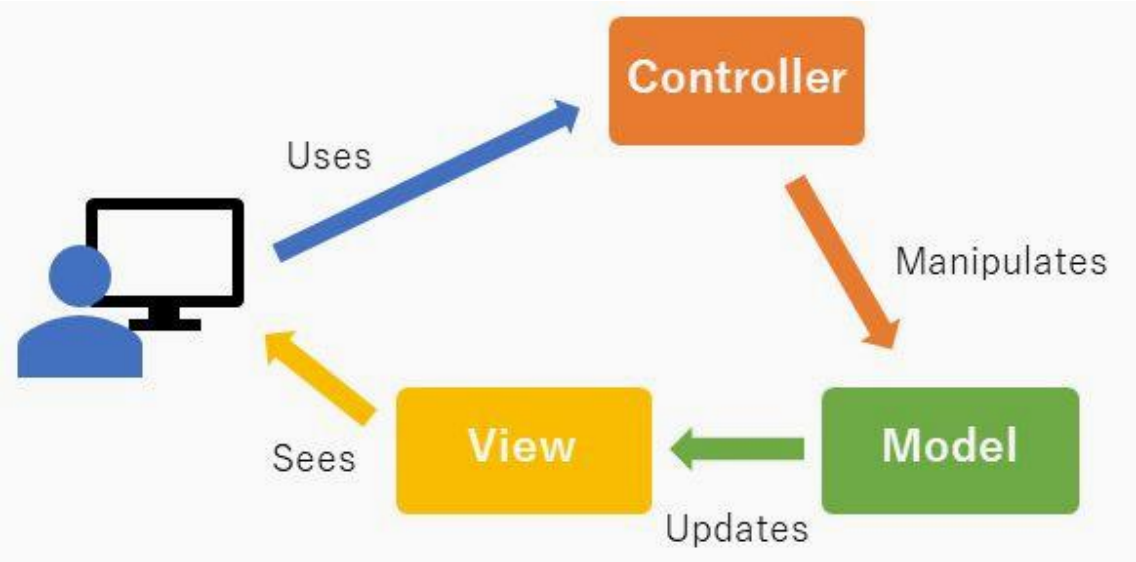

Ilustración 6 - Modelo MVC

3.3 Diagrama de clases

El conjunto de clases principales, se encuentran en la capa del dominio (modelo), que vendrá definido por el siguiente diagrama:

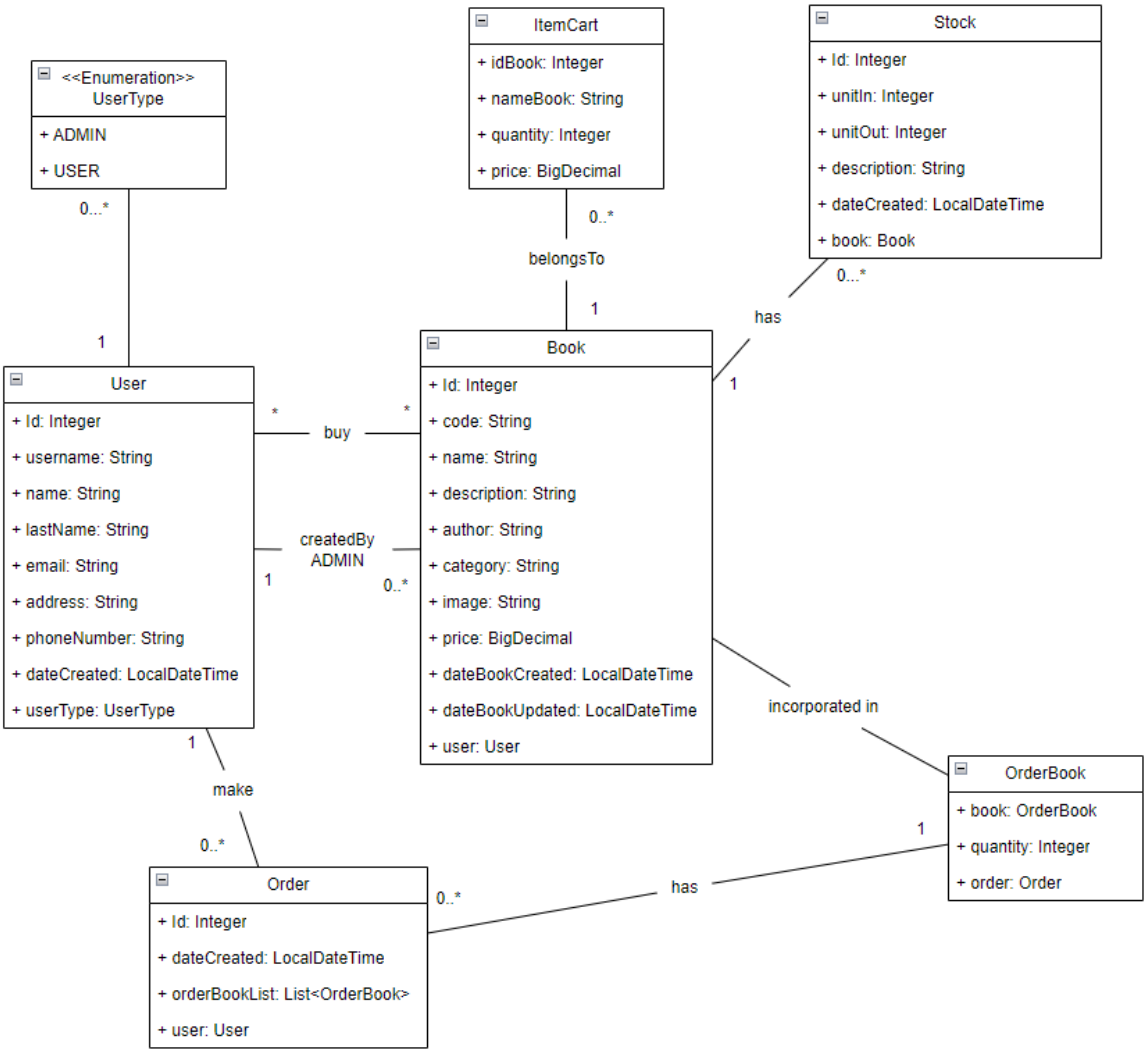

Ilustración 7 - Diagrama de clases del dominio

3.4 Stack tecnológico

En los siguientes subapartados se nombrarán y detallarán cada una de las tecnologías empleadas para llevar a cabo el proyecto.

Comentar que todas las herramientas utilizadas son de código/uso libre.

Para tener una mejor organización en el esquema tecnológico, se van a dividir en diferentes ámbitos:

- Desarrollo
- Sistemas
- Otras herramientas

3.4.1 Desarrollo

En este apartado, se detallaran las tecnologías utilizadas para el propio desarrollo del proyecto.

3.4.1.1 Java JDK 17

Java es un lenguaje de programación, de alto nivel, multiplataforma, orienta a objetos y basado en clases. Ha sido una opción popular entre los desarrolladores durante más de dos décadas, con millones de aplicaciones Java en uso en la actualidad. Fue desarrollado por Sun Microsystems en el año 1995.

Ilustración 8 - Java

3.4.1.2 Spring framework

Framework aplicativo más popular para Java empresarial, nos permite crear código de alto rendimiento, liviano y reutilizable. Ya que su finalidad es estandarizar, agilizar, manejar y resolver los problemas que puedan ir surgiendo en la etapa de desarrollo.

Ilustración 9 - Spring

3.4.1.3 JPA (Java Persistence API)

Es la API interna de Java EE para el manejo de datos relacionales en el desarrollo de aplicaciones Java. Su objetivo no es otro que el convertir las entidades del Dominio en instrucciones traducibles para la base de datos.

3.4.1.4 Hibernate

Hibernate es una herramienta de mapeo objeto-relacional (ORM) bajo licencia GNU LGPL para Java, que facilita el mapeo de atributos en una base de datos tradicional, y el modelo de objetos de un aplicación mediante archivos declarativos o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Ilustración 10 - Hibernate

3.4.1.5 PostgreSQL

Es un sistema de bases de datos de código abierto, altamente estable, que proporciona soporte a diferentes funciones de SQL, como claves foráneas, subconsultas, disparadores y diferentes tipos y funciones definidas por el usuario.

Ilustración 11 - PostgreSQL

3.4.1.6 Lombok

Es una biblioteca para Java que nos ofrece bastantes funcionalidades, ya que su objetivo es facilitarnos el desarrollo de nuestro código y así nos evita a nosotros desarrollar, métodos como getters/setters, equals, constructores y nos facilita la creación de variables y constantes.

Ilustración 12 - Lombok

3.4.1.7 MapStructs

Es una librería que nos va a permitir hacer mapeo de objetos sin tener que escribir todo el código a mano, simplemente con una interfaz, en la que le vamos a indicar el objeto de entrada y el objeto de salida.

Ilustración 13 - MapStruct

3.4.1.8 Thymeleaf

Es un motor de plantillas Java, que nos permite mostrar información en páginas html de forma sencilla. Se utiliza en proyectos web spring con arquitectura MVC. También se utiliza para crear plantillas bajo los estándares de XML/XHTML/HTML5.

Ilustración 14 - Thymeleaf

3.4.2 Sistemas

En este apartado, se detallaran las tecnologías utilizadas para el propio en el ámbito de sistemas y operaciones externas a la programación.

3.4.2.1 Docker

Es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución.

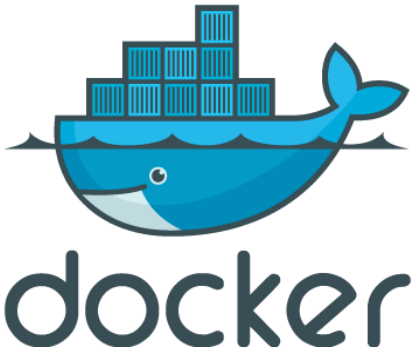

Ilustración 15 - Docker

3.4.2.2 Adminer

Es una herramienta de gestión de bases de datos de código abierto, gratuita y basado en PHP, es súper simple de implementarlo solo se tiene que configurar la URL hacía el puerto de escucha.

También admite la gestión de otras bases de datos como postgreSQL, SQLite, MS SQL, Oracle, SimpleDB, Elasticsearch,MongoDB y Firebird.

Ilustración 16 - Adminer

3.4.3 Otras herramientas

En este apartado, se detallaran las herramientas *auxiliares* utilizadas para el proyecto, con el propósito de emplear herramientas punteras aplicando los estándares del mercado laboral.

3.4.3.1 IntellIJ IDEA

Entorno de desarrollo integrado (IDE) para aplicaciones escritas en Java, Groovy o Kotlin. El fabricante es JetBrains, y dispone de dos versiones, en este caso se ha empleado la versión Community.

Ilustración 17 - IDE IntellIJ IDEA

3.4.3.2 Gitlab y Git

GitLab es una plataforma Git y DevOps basada en la nube que ayuda a los desarrolladores a supervisar, probar y desplegar su código.
Ofrece un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de ser un gestor de repositorios, ofrece generación de wikis y un sistema de seguimiento de errores.

También se ha utilizado la consola de git bash para poder hacer las operaciones git al repositorio GitLab.

Ilustración 18 - GitLab y Git Bash

3.4.3.3 Maven

Es una potente herramienta de gestión de proyectos que se utiliza para gestión de dependencias, como herramienta de compilación e incluso como herramienta de documentación. Es de código abierto y gratuito.

Conjuntamente con las tecnologías anteriormente mencionadas en puntos anteriores, se utiliza el fichero pom.xml. Es la unidad básica de trabajo en Maven es el llamado Modelo de Objetos de Proyecto conocido simplemente como POM (de sus siglas en inglés: Project Object Model).
Se trata de un archivo XML llamado pom.xml que se encuentra por defecto en la raíz de los proyectos y que contiene toda la información del proyecto: su configuración, sus dependencias, etc...

Ilustración 19 - Maven

3.4.3.4 Stripe

Stripe es un sistema de pago online diseñado para integrarlo directamente en la página web de una tienda online. Es decir, es un sistema de pago al estilo de PayPal, pero con la diferencia de que no envía al comprador a otra web externa para finalizar el pago, sino que el formulario de pago está dentro de la propia web.

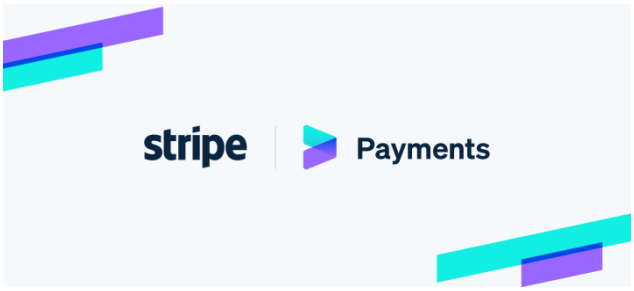

Ilustración 20 - Stripe (pasarela de pago)

3.5 Arquitectura del proyecto

Tras explicar detalladamente el diseño y stack tecnológico empleado para el desarrollo del proyecto, se muestra el diagrama completo de la arquitectura tecnológica.

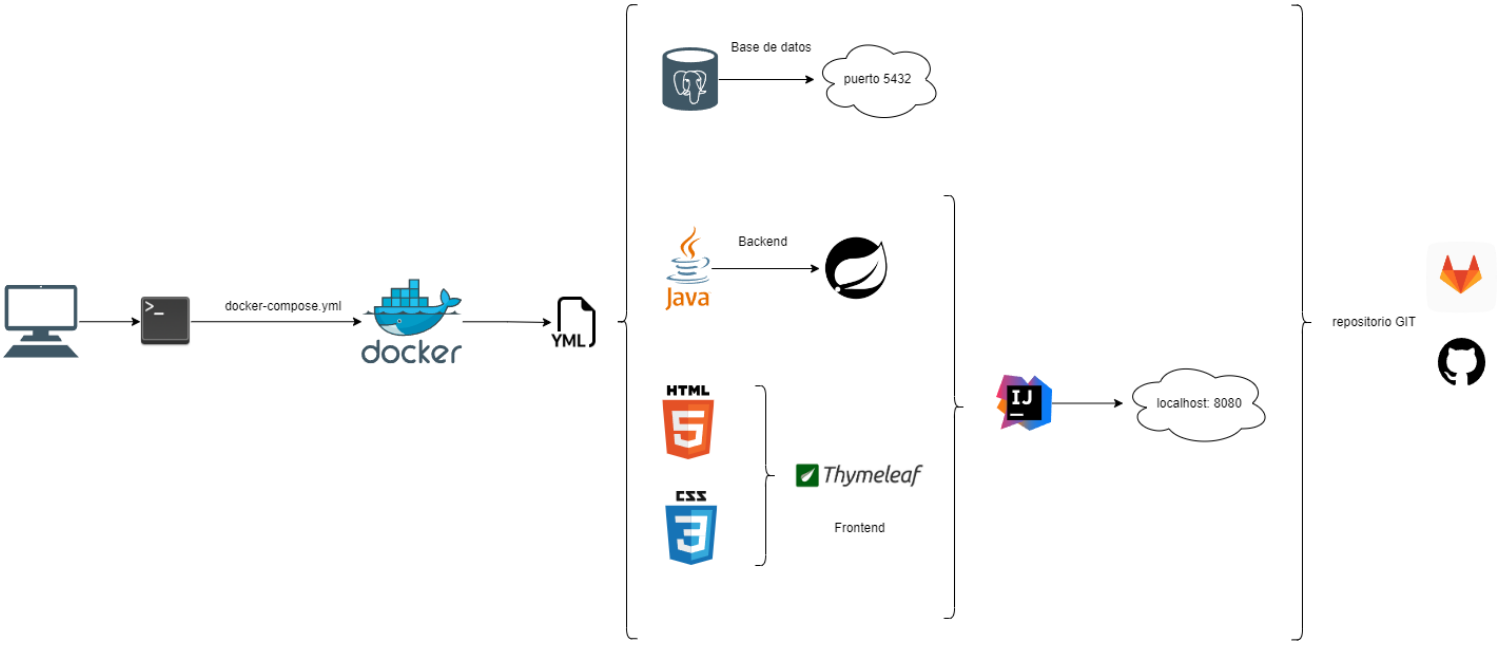

Ilustración 21 - Arquitectura tecnológica

3.5.1 Arquitectura hexagonal backend (API)

Para construir el backend enfocado en microservicios, se realizo un estudio de las diferentes arquitecturas existentes en cuanto a desarrollo, aprovechando los conocimientos adquiridos en asignaturas como Sistemas Distribuidos o Ing. del software de componentes y sistemas distribuidos, se llego a la conclusión que la forma más ordenada y limpia de construir el backend, era el construirlo bajo el paraguas de una arquitectura hexagonal.

Teniendo como puntos claro, la división del sistema en tres capas:

- Infraestructura
- Dominio
- Aplicación

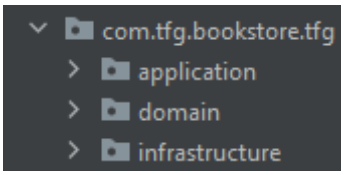

Ilustración 22 - Capas de Arquitectura hexagonal

Dentro de cada una de estas capas, se crean los diferentes componentes de la lógica de negocio, donde cada una de dichas capas se encarga exclusivamente de una funcionalidad en concreto, dando como resultado un bajo acoplamiento y una alta cohesión entre las capas.

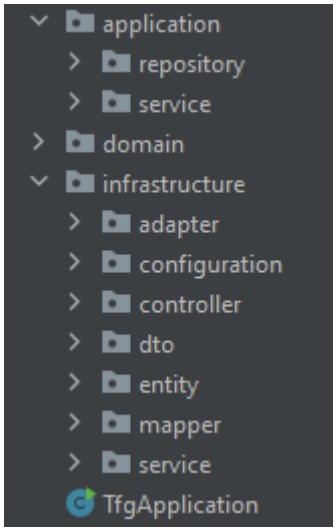

Ilustración 23 - Componentes de cada capa de la arquitectura

En el apartado de Implementación, se harán mención a los patrones SOLID y patrones de diseño utilizados.

3.6 Persistencia de datos

PostgreSQL será el encargado de la persistencia de los datos del sistema. El diseño conceptual de muestra en la *Ilustración 7 – Diagrama de clases del dominio,* pero al emplear una arquitectura hexagonal, no todas las clases de dominio serán finalmente entidades relacionales en la base de datos, por lo tanto, el esquema de base de datos es:

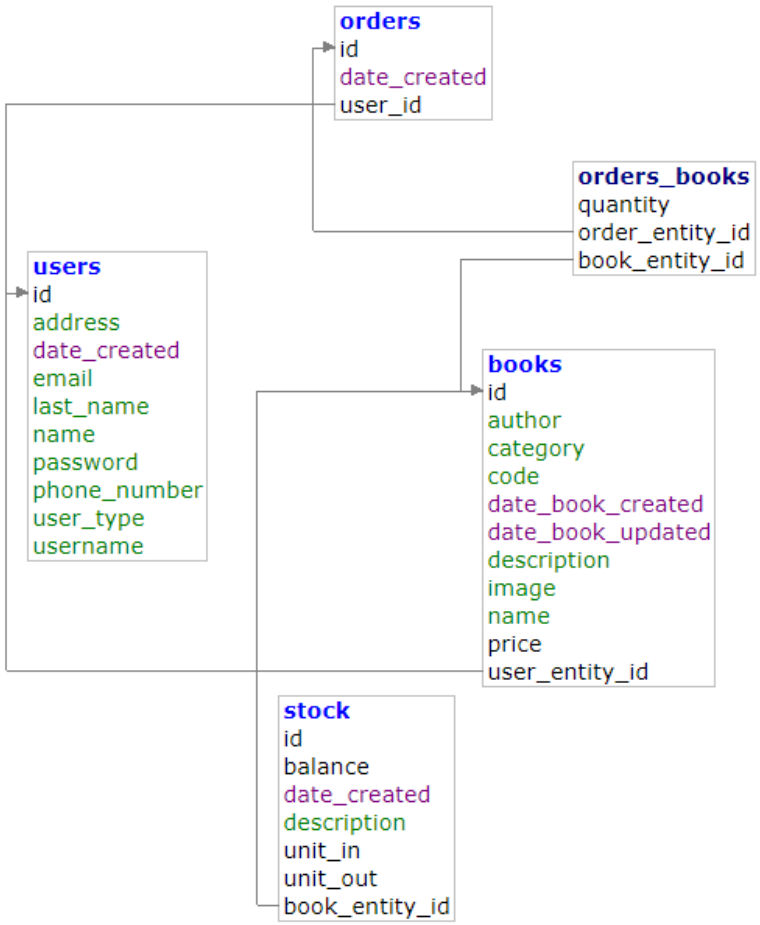

Ilustración 24 - Diseño conceptual de la BD

En cuanto al apartado de persistencia en el backend, se tiene configurado en la capa de infraestructura mediante el package de Entity, donde mediante las anotaciones pertinentes (@Entity) de Spring, se define la relación entre la API y la BD.

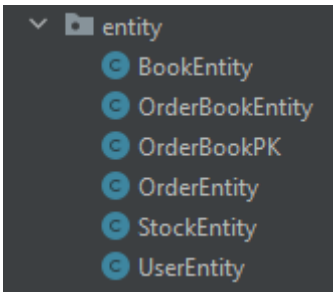

Ilustración 25 - Entidades de la BD

También comentar, que en el backend se tiene persistencia de los datos mediante el uso de JPA y Hibernate, empleando los componentes Repository y sus Implementations, donde las interfaces Repository extienden y sobrescriben los métodos de CrudRepository. Permitiendo el empleo de la encapsulación de las consultas (queries) mediante el uso de dichos métodos.

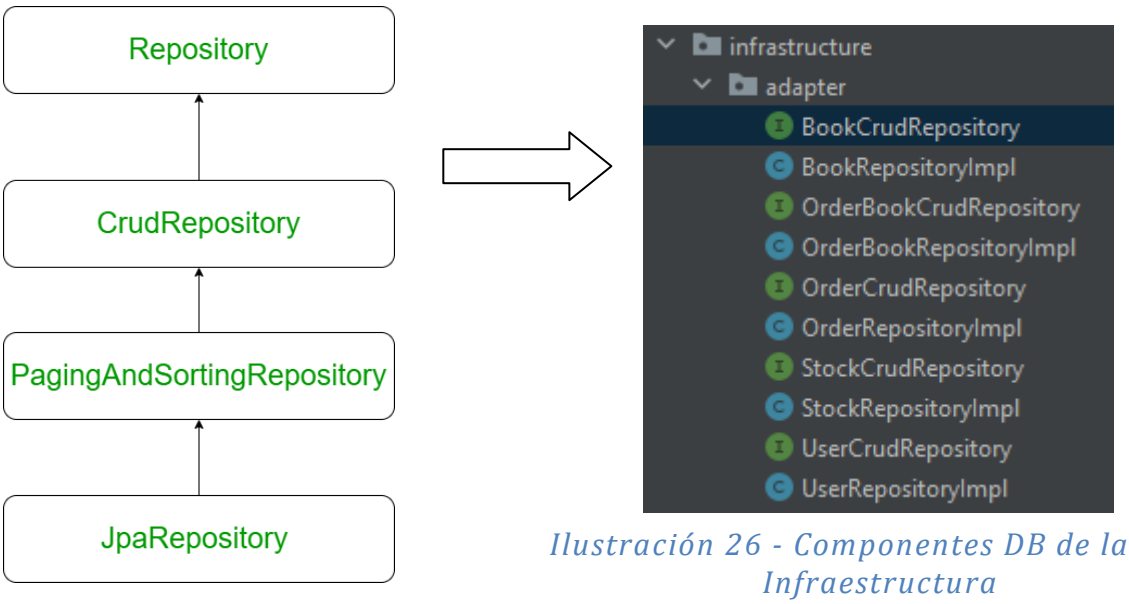

Ilustración 26 - Componentes DB de la capa Infraestructura

3.7 Despliegue

El despliegue de la aplicación (proyecto) se debe realizar mediante la virtualización de los diferentes componentes, utilizando Docker como herramienta DevOps para encapsular todos los componentes y tenerlos en un mismo entorno.

Esto permite tener todo el contenido más centralizado, y operativo para su desarrollo, implementación y despliegue en diferentes sistemas. Ya que el único requerimiento sería tener Docker instalado en el sistema.

Posteriormente, se tendría que realizar un clone de los repositorios GitLab que conforman los diferentes componentes, e iniciar el despliegue en el sistema.

El conjunto de componentes que conforman el proyecto TFG, se definen en las ilustraciones 22 y 23.

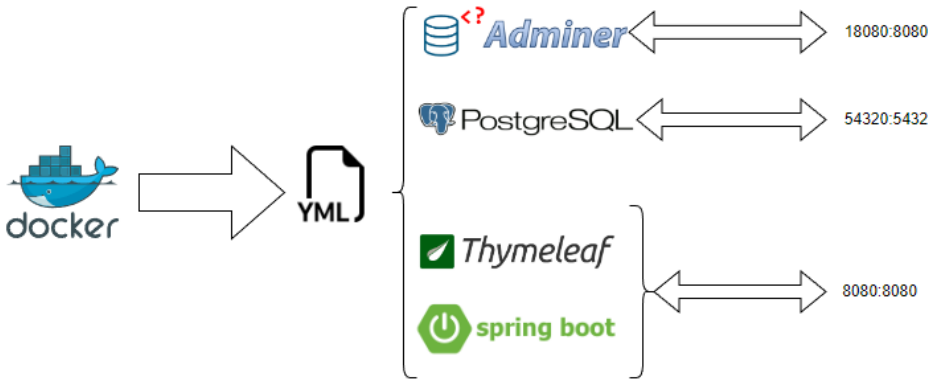

Ilustración 27 - Diagrama Docker

En el apartado de Manual de uso, se mostrarán ilustraciones de los contenedores en funcionamiento, con sus puertos dedicados y expuestos al tráfico de peticiones HTTP.

```yaml
services:
  db:
    image: postgres
    restart: always
    volumes:
      - postgres:/var/lib/postgresql/testdb
    ports:
      - 54320:5432
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    healthcheck:
      test:
        [
          "CMD",
          "pg_isready",
          "-q",
          "-d",
          "${POSTGRES_DB:-postgres}",
          "-U",
          "${POSTGRES_USER:-postgres}"
        ]
      timeout: 45s
      interval: 10s
      retries: 10
    networks:
      - net

  adminer:
    image: adminer
    platform: linux/amd64
    restart: always
    ports:
      - 18080:8080
    networks:
      - net

  backend:
    build:
      context: ../tfg-bookstore_2.0
      dockerfile: Dockerfile
    ports:
      - 8080:8080
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/postgres
      - SPRING_DATASOURCE_USERNAME=postgres
      - SPRING_DATASOURCE_PASSWORD=postgres
    depends_on:
      db:
        condition: service_healthy
    networks:
      - net
volumes:
  postgres:

networks:
  net:
```

Ilustración 28 - Fichero docker-compose.yml

3.8 Seguridad

En este apartado, se especificarán y detallarán los aspectos relativos a la seguridad de los diferentes componentes del proyecto.

3.8.1 Base de datos

Referente a la base de datos, únicamente es accesible mediante la gestión conjunta de la interfaz UI Adminer, la cual tiene configurada un puerto dedicado y puerto expuesto, a la par que se han definido unas credenciales de conexión mediante el conector JDBC entre la API y la BD.

Para acceder a la gestión de la BD mediante Adminer, las credenciales de acceso están definidas tanto en el fichero application.properties del backend como en el servicio db del fichero docker-compose.yml

```
spring.datasource.url=jdbc:postgresql://localhost:54320/postgres
spring.datasource.username=postgres
spring.datasource.password=postgres
```

Ilustración 29 - Credenciales DB application.properties

```
environment:
  - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/postgres
  - SPRING_DATASOURCE_USERNAME=postgres
  - SPRING_DATASOURCE_PASSWORD=postgres
depends_on:
  db:
    condition: service_healthy
```

Ilustración 30 - Credenciales DB docker-compose.yml

3.8.2 Backend (API)

En cuanto a la implementación de seguridad referente al backend, se tiene dentro de la capa de Infraestructura un package destinado íntegramente a ello *Configuration,* donde las clases pertenecientes albergan en su *interior* el código de seguridad.

Toda esta implementación de seguridad, se consigue gracias a la utilización de las dependencias que nos ofrece Spring Security.

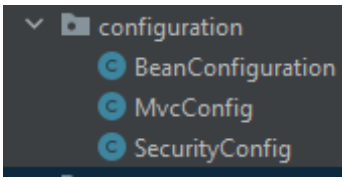

Ilustración 31 - Package de configuración/seguridad

Dentro de esta configuración, concretamente en la clase SecurityConfig, podemos encontrar métodos específicos para ciertos aspectos de seguridad, como por ejemplo:

- Encriptación de las contraseñas de los usuarios en la base de datos.

```
//Método para encriptar las contraseñas introducidas tanto en el registro como en el login
@Bean
public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Ilustración 32 - Método de encriptación de contraseñas

- Configuración de acceso a las distintas URL's accesibles mediante los puertos expuestos por los diferentes Controllers.

```
/*
 * Configuración de acceso por URL y con las credenciales comprobadas del usuario
 * damos acceso a los diferentes controladores de la aplicación
 * mediante el uso de los roles de usuario
 *
 * También indicamos a Spring Security que tenemos un login propio para comprobar las credenciales
 * y si es OK, entonces hace la redirección al método GetMapping("/access) de LoginController
 */

/*
 * con la anotación anyRequest().permitAll()
 * indicamos que las URL de mas bajo nivel, serán visibles para los usuarios que no estén registrados
 */

@Bean
public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.csrf().disable().authorizeHttpRequests() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegistry
            .requestMatchers( ...patterns: "/admin/**").hasRole("ADMIN")
            .requestMatchers( ...patterns: "/user/**").hasRole("USER")
            .anyRequest().permitAll()
            .and() HttpSecurity
                .formLogin() FormLoginConfigurer<HttpSecurity>
                .loginPage("/login")
                .defaultSuccessUrl("/login/access")
            .and() HttpSecurity
                .logout() LogoutConfigurer<HttpSecurity>
                .logoutSuccessUrl("/")
            .deleteCookies( ...cookieNamesToClear: "JSESSIONID");
    return httpSecurity.build();
}
```

Ilustración 33 - Método de seguridad de accesos

4. Implementación

4.1 Introducción

En esta sección, se detalla todo el proceso de implementación del proyecto en base a lo estudiado y argumentado en las fases anteriormente explicadas, análisis y diseño.

También se realizará una explicación de conceptos externos aplicados al proyecto, como estándares, conceptos y códigos de buenas prácticas dentro del ámbito de la programación, que han ayudado a tener una visión más clara y simple, dando como resultado un código simple, fácil de leer y entender.

4.2 Guías de estilo

Durante toda la fase de desarrollo y implementación del código en los diferentes componentes que conforman el proyecto, se han tenido en cuenta las guías de estilo establecidas por la comunidad de cada uno de los lenguajes.

Como el proyecto está compuesto por varios componentes, los cuales a su vez están programados en diferentes lenguajes, se han aplicado las guías pertinentes a cada uno.

- Backend:
    - Java: Google Java Style Guide
- Frontend:
    - HTML5 + CSS: Google HTML/CSS Style Guide
    - JavaScript: Google JavaScript Style Guide

4.3 Patrones SOLID

Para este apartado, se ha tenido muy en cuenta todo lo aprendido durante el transcurso de la asignatura Análisis y diseño de patrones de este mismo grado.

Partiendo como base la documentación ofrecida en la propia asignatura y sus prácticas entregables, siendo un buen punto de partida aplicable al código del proyecto.

Es por eso, que se han aplicado ciertos patrones durante las fases de desarrollo e implementación, tales como:

- Principio abierto/cerrado.
- Bajo acoplamiento.
- Alta cohesión.
- Principio de única responsabilidad

4.4 Patrones de diseño

Siguiendo con los conocimientos obtenidos en la asignatura anteriormente nombre, durante la fase de desarrollo e implementación se tuvo muy en cuenta el empleo de patrones de diseño, siendo algunos de ellos:

- Modelo Vista Controlador → MVC
- Singleton: los objetos @Bean de Spring se construyen bajo este patrón de diseño.

src/main/java/com/tfg/bookstore/tfg/infrastructure/configuration/BeanConfiguration.java · main · UOC TFG / TFG BookStore · GitLab

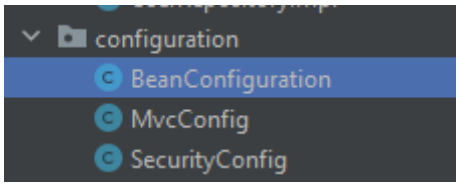

Ilustración 34 - Clase donde están los Bean con el patrón Singleton

- Patron DTO: se ha utilizado para serializar las clases del dominio y la entidad entre las diferentes capas, en este caso bajo UserDTO.

src/main/java/com/tfg/bookstore/tfg/infrastructure/dto · main · UOC TFG / TFG BookStore · GitLab

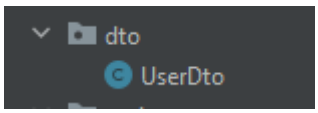

Ilustración 35 - Clase UserDTO

- Patrón mapper: se ha empleado para mapear los atributos entre las clases de dominio y sus respectivas entidades.

src/main/java/com/tfg/bookstore/tfg/infrastructure/mapper · main · UOC TFG / TFG BookStore · GitLab

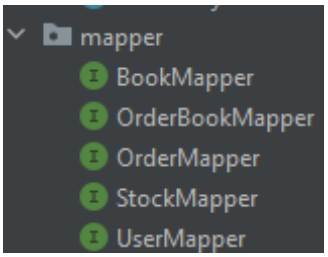

Ilustración 36 - Package aplicando el patrón mapper

4.5 Repositorio GitLab

El código de los componentes y el resto de archivos que forman parte del proyecto, se pueden encontrar en el repositorio de Gitlab en forma de grupo de trabajo.

- https://gitlab.com/uoc-tfg

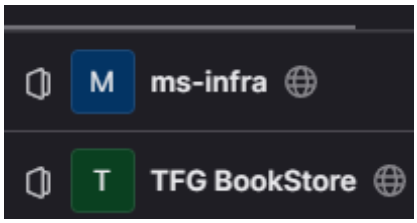

Ilustración 37 - Contenido del repositorio GitLab

4.5 Stripe payments

Debido a la temática elegida para la creación del proyecto, se pensó, analizo y estudio, cual de todas las de API's de terceros era la mejor opción para ofrecer la funcionalidad de pagos online en la aplicación.

Finalmente, se escogió realizar la integración e implementación de la API de Stripe, mediante la funcionalidad de Checkout payment que tienen disponible.

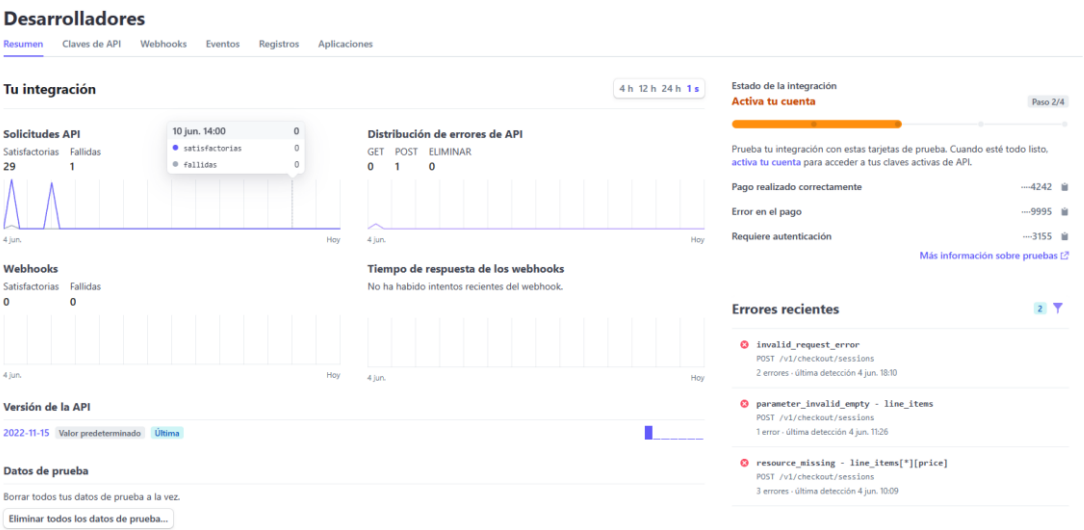

Ilustración 38 - Dashboard de la integración de Stripe

Para realizar la integración e implementación de la API de Stripe al código ya existente en el backend, se tuvieron que vincular las claves públicas y privadas para que se estableciera la comunicación mediante las llamadas de los Controllers.

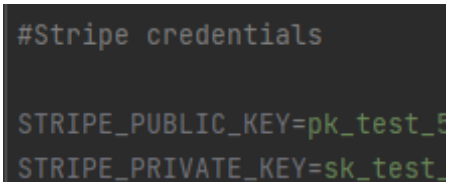

Ilustración 39 - Claves de la integración de Stripe

Todo lo referente a la implementación de funcionalidades, se puede consultar en la Ilustración del fichero pom.xml, donde se muestran las dependencias de cada funcionalidad añadida al proyecto → Ver apartado Anexo

5. Manual de uso

En este apartado, se hará una explicación detallada de la puesta en marcha del proyecto para poder realizar las pruebas pertinentes, y ver el completo funcionamiento de todos los componentes que lo forman.

1. Realizar un git clone de ambos proyectos disponibles en el repositorio de GitLab.
    - UOC TFG · GitLab
2. Acceder a las carpetas donde se han clonado los proyectos

Ilustración 40 - Resultado del git clone

3. Copiarnos la URL de acceso a la raíz de la carpeta ms-infra.
4. Abrir una terminal CMD y realizar los siguientes pasos:
    - cd + la ruta anteriormente copiada
    - docker-compose up –build

➢ *Con este paso, levantaremos los servicios del fichero docker-compose.yml creando la infraestructura necesaria para el correcto funcionamiento del proyecto, teniendo como resultado:*

➢

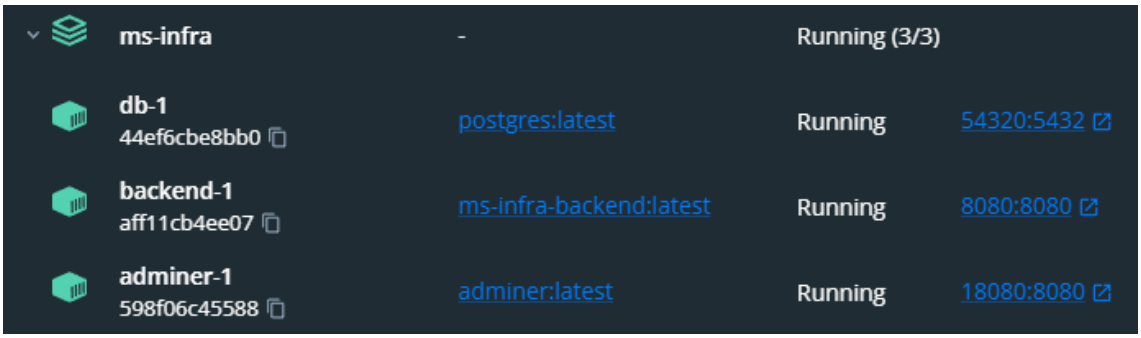

Ilustración 41 - Despliegue en Docker Desktop

5.1 Consideraciones previas

El registro siempre hace un INSERT en la tabla User del UserType_USER, por lo que habrá que modificar 1 usuario para que sea UserType_ADMIN y poder crear un catalogo de pruebas.

En el apartado del checkout de Stripe, se utiliza su API en el entorno SandBox/TEST, por lo tanto los datos bancarios son de test.

- El nº de la targeta ha utilizar es: 4242 4242 4242
- El resto de datos pueden ser inventados, pero cumpliendo con el formato correcto

Para cualquier aclaración de la puesta en marcha del proyecto, se tienen creados dos ficheros Readme.md, los cuales contienen un seguido de instrucciones e consideraciones previas para el inicio del proyecto.

- README.md · main · UOC TFG / ms-infra · GitLab
- README.md · main · UOC TFG / TFG BookStore · GitLab

6. Puntos de mejora y/o pendientes

En el siguiente apartado, se enumerarán los puntos de mejora o tareas pendientes de realizar de este trabajo.

- Inclusión de pruebas unitarias.
- Implementación de más herramientas DevOps, tales como:
  - Grafana
  - RabbitMQ
  - SonarQube
- Creación de un microservicio exclusivo para la gestión del Administrador.
- Implementación de un microservicio dedicado al envío de correo electrónico mediante SMTP, para poder reenviar las facturas emitidas o el *reseteo* de las credenciales de los usuarios.

7. Conclusiones

Estoy muy satisfecho con el resultado final obtenido, ya que es funcional y abierto al desarrollo de nuevas funcionalidades gracias a los diferentes métodos seguidos durante el desarrollo del mismo.

Con la realización de este trabajo, he podido afianzar los conocimientos del paradigma de la programación orientada a objetos que se ha ido trabajando durante el transcurso del grado.

También me gustaría comentar, que durante las fases de análisis y diseño, he podido aplicar todos los conocimientos adquiridos y utilizar los recursos disponibles durante la realización de varias asignaturas aplicables a la Ingeniería del Software, tales como:

- Ingeniería de requisitos, Ingeniería del Software, Análisis y Diseño de patrones Programación Orienta a Objetos, Proyecto de software, etc.

Además, el seguir el proceso de creación y ejecución de un proyecto de forma individual, llevado a cabo de manera ordenada y con unas fechas que cumplir, ha sido todo un reto personal, dando como resultado la entrega de este trabajo de final de grado.

Para concluir, me gustaría poder expresar mi satisfacción personal y de orgullo, por haber logrado llegar hasta este punto donde se concluye la memoria de este proyecto, y por lo tanto, la conclusión de una etapa de aprendizaje y superación personal.

8. Glosario

- *Reseteo*: generar unas nuevas credenciales de acceso, concretamente la contraseña.
- DevOps: conjunto de prácticas que combinan el desarrollo y las operaciones IT.
- Stakeholder: personas interesada.
- Container: unidad de software referida al uso específico de la herramienta Docker.
- DTO: (Data Transfer Object) objeto simple y plano que representa una unidad y que se encarga del transporte de las diferentes entidades entre las capas de la arquitectura del proyecto.
- Microservicio: unidad que indica los servicios que forman parte de una aplicación construida sobre la arquitectura de microservicios.
- MVC: (Modelo Vista Controlador) es un patrón de arquitectura de software que separa la lógica de negocio en tres componentes: (modelo – vista – controlador)
- API: (Application Programming Interface) es un componente intermediario entre dos sistemas, que permite que una aplicación se comunique con otra y pida datos o acciones específicas.
- Stack tecnológico: listado de todos los servicios tecnológicos, sistemas, herramientas y componentes utilizados para crear y ejecutar una aplicación.
- *Virtualización*: tecnología que se puede usar para crear representaciones virtuales de servidores, almacenamiento, redes y otras máquinas físicas.

9. Bibliografía

- https://www.w3schools.com/html/default.asp
- https://www.w3schools.com/bootstrap4/default.asp
- https://getbootstrap.com/docs/5.3/getting-started/introduction/
- https://getbootstrap.com/docs/4.0/examples/
- https://mdbootstrap.com/
- https://www.baeldung.com/java-stripe-api
- https://stripe.com/docs/payments/checkout
- https://checkout.stripe.dev/
- https://medium.com/javarevisited/lets-develop-an-ecommerce-application-from-scratch-using-java-and-spring-6dfac6ce5a9f
- https://www.baeldung.com/spring-angular-ecommerce
- https://github.com/masasdani/paypal-springboot
- https://www.baeldung.com/spring-boot-postgresql-docker
- https://pramodshehan.medium.com/simple-rest-api-with-springboot-postgres-and-docker-d15071908b8a
- https://openwebinars.net/blog/como-configurar-cors-en-mi-api-rest-con-spring-boot/
- https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
- Para consultas relacionadas con errores en las fases de implementación, he utilizado https://stackoverflow.com/ y las diferentes búsquedas de Google.
- Para la aplicación de los patrones SOLID y de diseño, se ha empleado los recursos de aprendizaje de la asignatura *Análisis y Diseño de patrones*.
- Para la creación de los modelos de casos de uso, se ha empleado el disponible en la asignatura *Ingeniería de Requisitos*.
- Para la redacción de la memoria, se ha empleado el recurso de aprendizaje disponible Redacción de textos científico-técnicos.

10. Anexo

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.6</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.tfg.bookstore</groupId>
    <artifactId>tfg</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>tfg</name>
    <description>UOC Java EE TFG</description>
    <properties>
        <java.version>17</java.version>
        <version.mapstruct>1.5.2.Final</version.mapstruct>
        <version.lombok>1.18.24 </version.lombok>
        <version.mapstruct-lombok>0.2.0 </version.mapstruct-lombok>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Añadimos las dependencias para poder generar las facturas en PDF -->
        <dependency>
            <groupId>com.github.librepdf</groupId>
            <artifactId>openpdf</artifactId>
            <version>1.3.30</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<!-- Añadimos las depencias de MapStruct -->
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>${version.mapstruct}</version>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>${version.lombok}</version>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok-mapstruct-binding</artifactId>
    <version>${version.mapstruct-lombok}</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

```
        <!-- Dependencia para la validación de los campos backend del registro de usuarios -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <!-- Añadimos dependencia para la integración con stripe -->
        <dependency>
            <groupId>com.stripe</groupId>
            <artifactId>stripe-java</artifactId>
            <version>22.22.0</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <!--Plugin Maven para poder dockerizar el proyecto -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <!--Plugin compiler Maven para poder utilizar los MapStructs y Lombok -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.10.1</version>
                <configuration>
                    <annotationProcessorPaths>
                        <path>
                            <groupId>org.mapstruct</groupId>
                            <artifactId>mapstruct-processor</artifactId>
                            <version>${version.mapstruct}</version>
                        </path>
                        <path>
                            <groupId>org.projectlombok </groupId>
                            <artifactId>lombok</artifactId>
                            <version>${version.lombok}</version>
                        </path>
                                <path>
                                    <groupId>org.projectlombok </groupId>
                                    <artifactId>lombok-mapstruct-binding</artifactId>
                                    <version>${version.mapstruct-lombok}</version>
                                </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```