



MÓDULO PROYECTO

CFGS Desarrollo de Aplicaciones
Multiplataforma
Informática y Comunicaciones

Gestión Empresarial: Facturación y Control de usuarios.

Tutor individual: M.^a Cristina Silván Pardo

Tutor colectivo: M.^a Cristina Silván Pardo

Año: 2023 - 2024

Fecha de presentación: 19/12/2024

Nombre y Apellidos: Ruslan Tejerina Zapico
Email: ruslan.tejzap@educa.jcyl.es

**Foto actual
del alumno**

Tabla de contenido

1	Identificación proyecto.....	3
2	Organización de la memoria.....	3
3	Descripción general del proyecto.....	3
3.1	Objetivos.....	3
3.2	Cuestiones metodológicas.....	4
3.3	Entorno de trabajo.....	5
4	Descripción general del producto.....	7
4.1	Visión general del sistema:.....	7
4.2	Descripción de métodos, técnicas o arquitecturas(m/t/a) utilizadas.....	10
4.3	Despliegue de la aplicación y puesta en marcha.....	19
5	Planificación y presupuesto.....	22
6	Documentación Técnica: análisis, diseño, implementación y pruebas.....	24
6.1	Análisis del sistema y especificación de requisitos.....	24
6.2	Diseño del sistema:.....	27
6.2.1	<i>Diseño de la Base de Datos.....</i>	27
6.2.2	Diseño API RESTful.....	29
6.2.3	Diseño de la Aplicación.....	30
6.3	Implementación:.....	31
6.3.1	Entorno de desarrollo.....	31
6.3.2	Estructura del código.....	33
6.3.3	Cuestiones de diseño e implementación reseñables.....	36
6.4	Pruebas.....	37
7	Manuales de usuario.....	40
7.1	Manual de usuario.....	40
7.2	Manual de instalación para la puesta en marcha de la aplicación.....	51
8	Conclusiones y posibles ampliaciones.....	54
9	Bibliografía.....	55

1 Descripción general del proyecto

1.1 *Objetivos*

Desarrollar una web de gestión completa de clientes y usuarios con roles administrativos diferenciados cada uno de ellos con su sesión personal, validaciones de los formularios, integración de logins con JWT: siglas de JSON Web Token, siendo un estándar abierto que me permite transmitir información de manera segura para cada usuario dependiendo del rol establecido.

La función principal de mi JWT es autenticar al usuario registrado, establecer diferentes roles y los permisos del usuario en función de su rol preestablecido, todo ello lo configuramos en mi código con Spring Security. Es una de las partes más complejas y sofisticadas del código, se detallará más adelante.

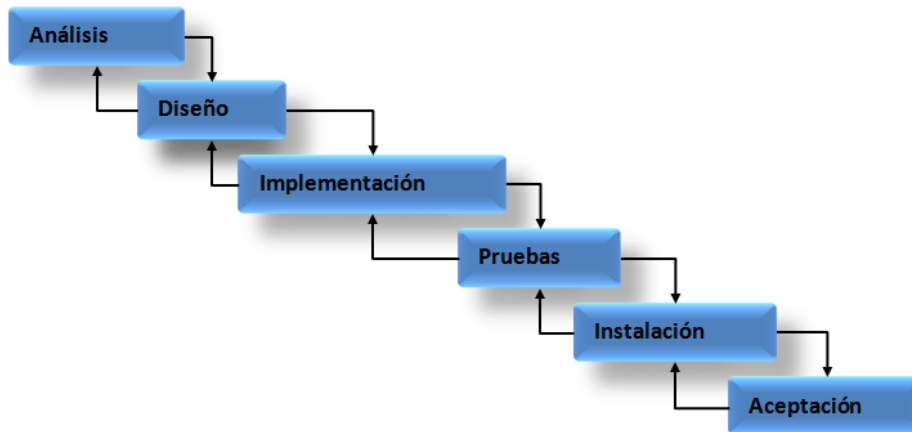
Para la parte de almacenamiento persistente de los datos he implemento el sistema de gestión de bases de datos relacional mediante el motor de datos de MySQL, estableciendo una API con Spring Boot 3, herramienta que nos ayuda a dar soporte para integrar JPA y Hibernate, permitiendo el mapeo objeto-relacional y la gestión de entidades, además mediante Spring Data JPA podemos realizar las operaciones CRUD y establecer configuraciones relativamente sencillas para la conexión a nuestra base de datos externa de MySQL.

La parte de la vista de la web con la que interactúa el usuario la desarrollo con el framework de Angular una arquitectura basada en componentes que facilita en cierto modo la creación de nuestra interfaz modular y reutilizable. Donde estableceremos el registro de los usuarios, entrada y salida de cada sesión, así como la parte de las acciones del CRUD siempre que el usuario tenga el rol administrador. Todo ello con los botones y validaciones de los formularios pertinentes.

El objetivo final es llegar a subir a producción mediante Amazon Web Service (AWS) todo nuestro proyecto web para no depender de conexiones a nivel local de nuestro ordenador y de ese modo llevar a cabo un proyecto real a pequeña escala con el objetivo de ver todos los pasos posibles en el camino de desarrollo emprendido.

1.2 Cuestiones metodológicas

Se ha seguido la metodología clásica: Ciclo de Vida en Cascada que se basa en una secuencia lineal de fases, donde el progreso fluye de forma descendente a través de las etapas del desarrollo y hay un enfoque en la documentación de tallada en cada fase. La imagen de abajo muestra claramente las etapas:



Ciclo de vida en cascada

Motivo de usar el Modelo en Cascada:

a. Simplicidad y Estructura:

- Es fácil de entender y gestionar.
- Cada fase tiene un objetivo claro.

b. Documentación Completa:

- Garantiza que todas las decisiones y requisitos estén bien documentados.

c. Adecuado para Proyectos Estables:

- Funciona bien en proyectos donde los requisitos no cambian significativamente.

1.3 Entorno de trabajo

El entorno de trabajo IDE utilizado para desarrollar el código en mi proyecto es el VSCode. Con dicho IDE conseguimos implementar todas las librerías y paquetes necesarios para poder trabajar con Spring para la parte de backend y con Angular para la parte del frontend. Al ser un IDE muy completo y potente, me permite desarrollar mis proyectos con gran agilidad y simplicidad y todo ello en un mismo entorno, lo cual supone una gran ventaja para trabajar con comodidad.

- **Tecnologías de desarrollo:**

Se ha seguido un modelo cliente-servidor.

- ✓ **Herramientas de software, lenguajes de programación y servidor:**

- **Spring Boot 3 con Java:**

Framework de desarrollo de aplicaciones en Java que nos ayuda a simplificar el proceso de creación aprovechando la inyección de dependencias y la programación orientada a aspectos. Aunque el aspecto más destacable de esta tecnología es el uso del patrón Modelo Vista Controlador (MVC) para crear aplicaciones de manera estructurada y eficiente pudiendo definir controladores, manejar solicitudes y enlazar datos.

- **Spring Security:**

Framework de seguridad personalizable para nuestra aplicación Java que nos permite gestionar la autenticación, la autorización y la protección contra las vulnerabilidades en el flujo de datos. En nuestro caso usamos JWT para asegurar los servicios REST de nuestra API a través de mecanismos de autenticación, asegurar que solo los usuarios autenticados puedan interactuar con ciertas funcionalidades CRUD y gestionar los roles de la aplicación que pueden ser muy variados.

- **Angular:**

Framework de desarrollo web que se utiliza para construir aplicaciones de una sola página (SPA). Se basa principalmente en TypeScript, una especie de super conjunto de JavaScript que nos permite crear aplicaciones escalables u mantenibles. Con un claro objetivo que es establecer un tipado estático lo que nos ayuda a detectar errores en tiempo de compilación en lugar del tiempo de ejecución. TypeScript soporta muchas de las características modernas de JavaScript (ES6/ES7), como clases, módulos y funciones flecha, lo que facilita la escritura de código limpio y mantenible. Esto realmente marca la diferencia a la hora de programar para conseguir el propósito anteriormente mencionado.

- **Node.js:**
 - **Servidor de desarrollo:** Node.js se utiliza para ejecutar el servidor de desarrollo de Angular, lo que permite a los desarrolladores ver los cambios en tiempo real mientras trabajan en su código. Muy útil la verdad para ver el resultado de nuestro código a la hora de ir implementándolo sobre la marcha.
 - **Gestión de dependencias:** Node.js incluye npm (Node Package Manager), que se utiliza para gestionar las dependencias del proyecto Angular. Esto facilita la instalación y actualización de bibliotecas y herramientas necesarias para el desarrollo. Nos permite ir actualizando el proyecto a diferentes versiones con cierta facilidad.
 - **Construcción y despliegue:** Herramientas como Angular CLI, que están basadas en Node.js, se utilizan para construir y desplegar aplicaciones Angular. Estas herramientas permiten compilar el código, optimizarlo y prepararlo para su despliegue en producción.
 - **Backend para APIs:** Node.js se puede utilizar junto con Angular para crear aplicaciones full-stack. En este caso, Node.js actúa como el backend, proporcionando APIs RESTful que la aplicación Angular puede consumir. Aunque en mi caso Spring Boot realiza o proporciona APIs RESTful que mi frontend acaba consumiendo.
- **Bootstrap:** es un framework popular de CSS que me facilita mucho el diseño de mi interfaz siendo uno de los puntos más difíciles al tener poco conocimiento y experiencia con la programación orientada a frontend.
- **Sweet Alert:** es una biblioteca de JavaScript que he empleado para crear alertas personalizadas y atractivas en mi aplicación web a la hora de avisar al usuario si ha llevado acabo con éxito la tarea o ha habido algún error en el proceso. Muy útil y visual a la hora de informar al usuario sobre el estado del proceso de la tarea.
- **MySQL:** es un sistema de gestión de bases de datos relacional que utiliza el lenguaje de consulta estructurado (SQL) para acceder y manipular datos. Nos ayuda a crear la persistencia de todos los datos que necesitamos mantener, acceder mediante consultas y listados.

2 Descripción general del producto

2.1 Visión general del sistema:

- **Límites del sistema:**

Es una aplicación completamente autónoma al 100%, independiente de otras APIs, servidores y webs. La aplicación se ha desarrollado dividiendo el proyecto en dos grandes bloques uno de Backend con Spring Boot que nos proporciona nuestra API RESTful interfaz que permite que mi sistema se comuniquen a través del protocolo HTTP mediante métodos estándar como GET, POST, PUT, DELETE para realizar las operaciones sobre recursos almacenados en la base de datos.

Mi API RESTful utiliza el formato JSON para intercambiar datos entre el cliente y el servidor. Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud. El servidor no almacena el estado del cliente entre solicitudes.

Spring Boot utiliza anotaciones que simplifican la definición de controladores, servicios, repositorios y entidades siendo los componentes claves a la hora de desarrollar mi API RESTful para mantener una claridad y mantenibilidad del código además nos ayuda a reutilizar la lógica de negocio y las operaciones de base de datos en diferentes partes de la aplicación sin duplicar código al tener los servicios y repositorios separados.

Otra ventaja es la flexibilidad al poder cambiar la implementación de un componente sin afectar a los demás.

El otro bloque principal del proyecto es la parte del frontend realizado con Angular framework de desarrollo web de código abierto creado por Google lo que me permite desarrollar aplicaciones dinámicas escalables. Arquitectura basada en componentes, clases desarrolladas en TypeScript que definen la lógica y la vista se lleva a cabo con HTML y mediante el Data Binding los datos fluyen desde el componente a la vista.

Angular incluye un módulo HTTPClient que me permite interactuar con mi API RESTful.

- **Funcionalidades básicas API RESTful:**

- **Gestión de clientes:**

- **Listado de todos los clientes:**

- Recupera una lista de todos los clientes almacenados en la base de datos.

- Método HTTP: GET

- Endpoint: /api/customers

- Respuesta: Devuelve los clientes en formato JSON.

- **Obtener un cliente por su apellido:**

- Método HTTP: GET

- Endpoint: /api/customers/{lastname}

- Respuesta: devuelve un cliente con todos sus datos.

- **Gestión de productos:**

- **Listado de todos los productos:**
Recupera una lista de todos los productos almacenados en la base de datos.
- Método HTTP: GET
- Endpoint: /api/products
- Respuesta: Devuelve los productos en formato JSON.

- **Crear un producto nuevo:**
Crea un nuevo producto del catálogo.
- Método HTTP: POST
- Endpoint: /api/products
- Respuesta: Devuelve los productos en formato JSON, confirmando que se ha creado con éxito.

- **Obtener un producto por su ID:**
Obtenemos un producto al facilitar su ID.
- Método HTTP: GET
- Endpoint: /api/products/{id}
- Respuesta: Devuelve el producto en formato JSON con sus datos.

- **Gestión de usuarios:**

- **Listado de todos los usuarios:**
Recupera una lista de todos los usuarios almacenados en la base de datos.
- Método HTTP: GET
- Endpoint: /api/users
- Respuesta: Devuelve los usuarios en formato JSON con sus roles.

- **Obtener un usuario:**
Recupera un usuario de la base de datos por su id.
- Método HTTP: GET
- Endpoint: /api/users/{id}
- Respuesta: Devuelve un usuario con sus datos.

- **Crear un usuario nuevo:**
Crea un nuevo usuario con su rol.
- Método HTTP: POST
- Endpoint: /api/users
- Respuesta: Devuelve el usuario creado con todos sus campos.
- Autorización: Se requiere un token JWT válido en la cabecera.

- **Actualizar usuario:**
Modifica todos los campos de un usuario menos la contraseña, al ser un campo sensible por motivos de seguridad, para ello se requiere un endpoint más específico.
- Método HTTP: PUT
- Endpoint: /api/users/{id}
- Respuesta: Devuelve el usuario con los campos actualizados.
- Autorización: Se requiere un token JWT válido en la cabecera

- **Eliminar usuario:**
Elimina por completo al usuario por el id de la base de datos.
- Método HTTP: DELETE
- Endpoint: /api/users/{id}
- Respuesta: Devuelve el usuario con los campos actualizados.
- Autorización: Se requiere un token JWT válido en la cabecera

- Autenticación de usuarios con token JWT:

Se crea la clase *JWTAuthenticationFilter* que extiende de *UsernamePasswordAuthenticationFilter* lo que significa que intercepta solicitudes de inicio de sesión en la ruta por defecto /login y procesa las credenciales proporcionadas (nombre de usuario y contraseña). Este filtro es parte del flujo de autenticación de Spring Security y se utiliza para autenticar a los usuarios.

El filtro intercepta las solicitudes HTTP POST enviadas al endpoint de inicio de sesión (por defecto, /login).

Métodos principales en la autenticación de los usuarios:

- *attemptAuthentication*: este método se sobrescribe para extraer las credenciales del usuario (nombre de usuario y contraseña) de la solicitud HTTP y crear un token de autenticación. Luego, este token se pasa al *AuthenticationManager* para que realice la autenticación.
- *successfulAuthentication*: si las credenciales son correctas se ejecuta y realiza las siguientes funciones:
 - Obtiene el usuario autenticado y sus roles.
 - Crea el Token JWT:
 - Construye los claims (información adicional que se incluye en el token) como roles y el indicador de administrador.
 - Genera el token JWT usando:
 - El nombre del usuario (subject).
 - Los claims adicionales.
 - La clave secreta para firmar el token (SECRET_KEY).
 - Tiempo de expiración (1 hora).
 - Envía el Token al cliente:
 - Devuelve el token JWT en el encabezado Authorization y en el cuerpo de la respuesta como JSON.

En resumen, permite a los usuarios iniciar sesión en la aplicación enviando sus credenciales. Si las credenciales son válidas, genera un token JWT que el cliente puede usar para autenticar solicitudes posteriores. Estos métodos son fundamentales para el proceso de autenticación en aplicaciones que utilizan Spring Security, permitiendo una gestión segura y eficiente de las credenciales de los usuarios.

- **Usuarios:**

Existen dos tipos de usuarios definidos en el proyecto pudiendo establecer todos los roles que sean necesarios.

Usuarios con credenciales de administrador y usuarios regulares.

El usuario con el rol admin tiene la posibilidad de realizar operaciones CRUD, así como administrar la creación de otros usuarios y asignar los roles pertinentes.

El usuario regular solo podrá realizar las labores de consulta de datos y acceder al catálogo de productos, si agregamos más roles se podrán establecer diferentes accesos en función de las labores asignados a cada usuario.

El usuario administrador es el que tiene los permisos para crear las cuentas de otros usuarios asignándoles el nombre de usuario y la contraseña la cual se almacena en la base de datos encriptada.

Cuando los usuarios se acreditan en el login se lleva a cabo su autenticación mediante el JWT Token definido en el párrafo anterior.

2.2 Descripción de métodos, técnicas o arquitecturas(m/t/a) utilizadas.

✓ **Angular arquitectura basada en componentes:**

Cada parte de la interfaz de usuario se encapsula en componentes reutilizables.

En esta arquitectura se emplean componentes que se comunican mediante las rutas, la aplicación se segmenta en partes modulares y reutilizables.

Se puede observar los componentes que he generado en mi código del proyecto organizados por paquetes o directorios. En cada uno de ellos hay un fichero con extensión .ts (TypeScript) y un archivo .html (vista).

Además, mediante Servicios e inyección de dependencias se consigue compartir la lógica entre componentes, es decir que se pueden comunicar entre ellos.

Angular además proporciona el sistema de enrutamiento para manejar la navegación SPA lo que me permite definir como se cargan y muestran las diferentes vistas dentro de la misma página web, sin realizar recargas completas del navegador, esto lo gestionamos mediante el Router del framework.

Angular incluye un módulo para poder interactuar con mi API RESTful llamado HTTPClient que me permite consumir y mostrar datos dinámicamente en la web de la aplicación.

✓ **Descripción de los componentes en Angular:**

- *Modelo User*: es un modelo de datos que representa un usuario en la aplicación. Este modelo se utiliza para estructurar y tipar los datos relacionados con los usuarios. Esta clase se utiliza en varias partes del código, como en formularios de registro, en servicios que manejan la autenticación, autorización y en componentes que muestran información de los usuarios.
- El Componente Auth: es una parte crucial del flujo de autenticación de la aplicación, asegurando que los usuarios proporcionen la información necesaria antes de proceder con la autenticación.

Su plantilla (HTML) es una parte clave de la estructura del componente, se utiliza para mostrar los valores de las propiedades del componente en la vista, en este caso:

- Mostrar una interfaz de usuario para que los usuarios ingresen su nombre de usuario y contraseña.
 - Manejar la entrada del usuario mediante data binding que facilita la sincronización de datos entre el modelo y la vista.
 - Validar la entrada del usuario y mostrar mensajes de error si los campos requeridos no están completos.
 - Ejecutar la lógica de autenticación cuando el formulario es enviado.
- El componente NavBar con su plantilla HTML es responsable de gestionar la barra de navegación de la aplicación, mostrando u ocultando elementos basados en el estado de autenticación del usuario y proporcionando funcionalidad de cierre de sesión. Utiliza servicios inyectados para manejar la autenticación y la navegación dentro de la aplicación.
 - Mostrar una barra de navegación con enlaces a diferentes partes de la aplicación.
 - Gestionar el estado de autenticación del usuario, mostrando diferentes opciones de navegación basadas en si el usuario esta autenticado o no.
 - Proporcionar funcionalidad de cierre de sesión mediante el método handlerLogout() que cierra la sesión del usuario y lo redirige a la página de inicio de sesión. Para más detalles del método ver el código.

- El componente User con su plantilla HTML se encarga de gestionar la lógica relacionada con la visualización y manipulación de la lista de usuarios en la aplicación Angular.
 - Mostrar una lista de usuarios: Gestiona la visualización de una lista de usuarios, que puede ser pasada a través de la navegación obtenida del UserService.
 - Eliminar usuarios: Implementa el método `onRemoveUser(id: number)` para eliminar un usuario de la lista, emitiendo el id del usuario a través de un evento.
 - Seleccionar usuarios: Implementa el método `onSelectedUser(user: User)` para seleccionar un usuario y navegar a la página de edición del usuario.
 - Gestionar permisos de administrador: Utiliza el getter `admin` para determinar si el usuario actual es administrador, lo que permite mostrar u ocultar elementos en la vista basados en los permisos del usuario.

En el HTML asociado podemos ver el listado de usuarios con los botones para editar o eliminar en caso de que el rol del usuario sea admin, de lo contrario dichos botones no serán accesibles ni visibles.

- Componente `UserAppComponent`: es un componente principal que maneja varias funcionalidades relacionadas con la administración de usuarios y la autenticación en la aplicación. Lo desgloso brevemente, pero para más detalles revisar código., en especial el método `handlerLogin()` que tiene cierta complejidad de implementación.
 - Inicializar el Componente: en el constructor y en el método `ngOnInit()`, se llaman a varios métodos para inicializar el componente y manejar funcionalidades relacionadas con usuarios.
 - Manejar el Inicio de Sesión: implementa el método `handlerLogin()` para manejar el inicio de sesión del usuario:
 - Se suscribe a un evento para recibir las credenciales del usuario.
 - Llama al servicio de autenticación para verificar el usuario.
 - Procesa la respuesta de autenticación, descodifica el token y almacena la información del usuario autenticado en el `AuthService`.
 - Crear o actualizar Usuario: se encarga de crear insertar o editar usuarios, el mismo método dependiendo de si el usuario no tiene id o es 0, significa que debemos insertar nuevo usuario de lo contrario

actualiza los campos del usuario existente por su id. Se suscribe a un evento para para actualizar o crear usuarios en la base de datos. Todo ello asociado a los Swals de los Bootstrap que nos permiten emitir notificaciones con mensajes de éxito o error en la acción de crear o actualizar.

- Eliminar Usuarios: implementa e método `removeUser(id: number)` para quitar el usuario.
- Llama al servicio para eliminar el usuario de la base de datos.
- Actualiza la lista de usuarios en el componente.
- Navega para actualizar la vista de la lista de usuarios.
- Muestra notificaciones o alerts que saltan al usuario de éxito o cancelación usando el Swal.

Descripción de los servicios en Angular:

- Servicio `UserService`: es un servicio que proporciona métodos para interactuar con una API de backend para poder realizar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) relacionadas con usuarios. Aquí definimos la URL base del backend API para realizar las operaciones pertinentes.

Usamos los Observables para manejar las solicitudes HTTP de manera asíncrona y transformar los datos de la respuesta HTTP antes de emitirlos a sus suscriptores. Hay que recordar que los datos vienen en formato JSON desde el backend a través de las solicitudes de HTTP por ello hay que transformarlos en el tipo de dato manejable en nuestra aplicación Angular.

Explicación abreviada del flujo de datos:

- Método `findAll()`: realiza una solicitud HTTP GET para obtener todos los usuarios.
- Transforma la respuesta de la solicitud HTTP en un array de objetos `User`.
- Devuelve un Observable que emite el array de usuarios.
- Método `findById(id: number)`:
 - Realiza una solicitud HTTP GET para obtener un usuario específico por su ID.
 - Devuelve un Observable que emite el objeto `User` correspondiente al ID proporcionado.
- Servicio `SharingDataService`: es un servicio que se utiliza para compartir datos y eventos entre diferentes componentes de la aplicación. Este servicio se emplea para aquellos componentes que no tienen una relación directa entre padre-hijo.

- **Compartir Datos:**
 - El servicio tiene propiedades que almacenan datos que necesitan ser accesibles desde múltiples componentes.
 - Lo interesante es que los componentes pueden inyectar este servicio y acceder o modificar estos datos compartidos.
- **Emitir Eventos:**
 - El servicio utiliza EventEmitter de Angular para emitir eventos.
 - Los componentes pueden suscribirse a estos eventos para ser notificados cuando ocurran ciertos cambios o acciones.

En mi código he establecido propiedades privadas que son instancias de EventEmitter que usamos para poder emitir eventos específicos. Ver cada uno de ellos en el código. Después genero métodos getters que proporcionan acceso a los EventEmitter privados desde fuera de la clase. Mis componentes pueden suscribirse a estos eventos para ser notificados de cambios o acciones específicas, lo que me permite una comunicación eficiente y desacoplada entre componentes, lo que es realmente muy útil en mi código.

- **Servicio Auth.Service:** servicio que se encarga de manejar la autenticación de usuarios en la aplicación. Este servicio proporciona métodos para iniciar sesión, gestionar el estado de autenticación y almacenar información de autenticación en el almacenamiento de sesión (SessionStorage).

Descripción de los métodos del servicio AuthService:

- **LoginUser():** método de inicio de sesión.
 - Realiza una solicitud HTTP POST a la URL del puerto local 8080 del backend con las credenciales del usuario (usuario y contraseña)
 - Devuelve un Observable que emite la respuesta del servidor.
- **SetLogin:** método para establecer el estado de autenticación del usuario.
- **GetLogin:** método para obtener el estado de autenticación del usuario.
- **IsAdmin():** determina si el usuario es administrador
 - Utiliza el anterior getAdmin() para acceder al objeto login almacenado en el sessionStorage y devuelve el valor de la propiedad isAdmin del objeto login, true si es administrador y false si no lo es. Ver código para comprenderlo mejor, código comentado para facilitar la comprensión.
- **IsAuthenticated():** determina si el usuario actual está autenticado.
 - Accediendo a la propiedad isAuthenticated del objeto login almacenado en el SessionStorage.
- **Logout():** método que cierra la sesión del usuario eliminando el token y el objeto login tanto de la memoria del servicio como del sessionStorage.

- **Arquitectura en Capas en una API RESTful con Spring Boot:**

Beneficios de la arquitectura en capas:

- a. Claridad y Mantenibilidad:
 - Controladores: Manejan las solicitudes HTTP y devuelven las respuestas adecuadas. Esto hace que el código sea más fácil de entender y mantener.
 - Servicios: Contienen la lógica de negocio. Al separar la lógica de negocio de los controladores, se facilita la prueba y el mantenimiento del código.
 - Repositorios: Gestionan la interacción con la base de datos. Esto permite cambiar la implementación de la persistencia sin afectar la lógica de negocio.
 - Entidades: Representan los datos y sus relaciones. Facilitan la gestión de los datos y su mapeo a la base de datos.
- b. Reutilización de Código:
 - Al tener servicios y repositorios separados, puedes reutilizar la lógica de negocio y las operaciones de base de datos en diferentes partes de la aplicación sin duplicar código.
- c. Facilidad de Pruebas:
 - La separación de responsabilidades permite realizar pruebas unitarias y de integración de manera más efectiva. Puedes probar cada componente de forma aislada, lo que facilita la detección y corrección de errores.
- d. Escalabilidad:
 - La arquitectura modular facilita la escalabilidad de la aplicación. Puedes añadir nuevas funcionalidades o modificar las existentes sin afectar otras partes del sistema.
- e. Flexibilidad:
 - Permite cambiar la implementación de un componente sin afectar a los demás. Por ejemplo, puedes cambiar la base de datos o la lógica de negocio sin necesidad de modificar los controladores.

Capas implementadas en mi API RESTful:

- **Capa de Entidad (Model):**

- **Ubicación:** en el paquete o directorio entities.
- **Propósito:**
 - Mapear las clases de dominio a las tablas de la base de datos.
 - Definir los atributos de la clase y su comportamiento en la base de datos.
- **Finalidad de las entidades en mi proyecto:**
 - Mapear datos a la base de datos con validaciones.
 - Establecer relaciones entre entidades
- **Anotaciones JPA y Hibernate más relevantes:**
 - **@Entity:**
 - Indica que la clase es una entidad de la base de datos, lo que significa que se mapea a una tabla.

- **@Table(name = "users"):**
 - Mapea la clase a la tabla users en la base de datos.
- **@Id:**
 - Indica que el campo id es la clave primaria.
- **@GeneratedValue(strategy = GenerationType.IDENTITY):**
 - La clave primaria id se generará automáticamente en la base de datos.
- **@ManyToMany:**
 - Define una relación de muchos a muchos entre User y Role. Un usuario puede tener varios roles, y un rol puede ser asignado a varios usuarios.
- **@JoinTable:**
 - Especifica cómo se mapea la relación de muchos a muchos, indicando que hay una tabla intermedia users_roles, esta tabla hace de unión entre la tabla usuarios y roles mediante dos claves forráneas.
- **@JoinColumn:**
 - Mapea las claves forráneas en la tabla intermedia.
- **@UniqueConstraint:**
 - Asegura que la combinación de user_id y role_id sea única en la tabla intermedia users_roles.
- **@JsonIgnoreProperties:**
 - Previene la recursividad infinita cuando se convierte a JSON (por ejemplo, para evitar que se serialicen propiedades no necesarias).
- **@NotEmpty, @NotBlank, @NotNull, @Email, @Size(min = 4, max = 20), @Min(100000000), @Max(999999999):**
 - Estas anotaciones validan los campos antes de ser almacenados en la base de datos. Aseguran que los datos ingresados sean válidos (por ejemplo, el correo electrónico debe ser válido, el nombre no puede estar vacío, el teléfono debe tener un valor en el rango permitido, etc.).

- **Capa de Repositorio (Repository):**

- **Ubicación:** en los paquetes o directorios llamados repositories
- **Propósito:**
 - Proporcionar acceso a la base de datos mediante operaciones **CRUD** (Create, Read, Update, Delete).
 - Permite definir métodos personalizados para búsquedas o consultas específicas.
- **Características:**
 - Utiliza interfaces como CrudRepository.
 - La implementación de los métodos se genera automáticamente por Spring Data JPA.

- **Capa de Servicio (Service):**

- **Ubicación:** en el paquete o directorio services.
- **Propósito:**
 - Contener la lógica de negocio.
 - Actuar como una capa intermedia entre el repositorio y el controlador.
- **Características:**
 - Definir una interfaz que actúa como contrato para los métodos del servicio.
 - Usar la anotación `@Service` para marcar la clase como un servicio.
 - Inyección de dependencias preferentemente por constructor para asegurar inmutabilidad.
 - Controlar transacciones con `@Transactional`.
- **Finalidad principal:**
 - Definir reglas y procesos de negocio (cálculos, encriptación de contraseñas, gestión de roles, etc.).
 - Asegurar que las operaciones de la base de datos sean seguras y atómicas.
- **Anotaciones principales:**
 - **@Service:**
 - Indica que la clase contiene la lógica de negocio.
 - Permite que Spring gestione la clase y la inyecte donde sea necesario.
 - **@Transactional:**
 - Se usa para **gestionar transacciones** de manera automática.
 - Garantiza que las operaciones dentro del servicio se realicen de forma atómica (si una falla, se revierte todo).
 - Puede aplicarse a nivel de clase o método.
 - **@Autowired:**
 - Inyecta automáticamente **dependencias** necesarias (como repositorios u otros servicios).

- **Capa de Controladores (Controller):**

- **Ubicación:** en el paquete o directorio controllers.
- **Propósito:**
 - Manejar solicitudes HTTP y enviar respuestas al cliente.
 - Implementar operaciones **CRUD** siguiendo el patrón RESTful.
- **Finalidad principal:**
 - Manejar solicitudes del cliente y coordinar con el servicio.
 - Validar datos entrantes.
 - Proveer respuestas HTTP claras y estructuradas.
 - Generar documentación interactiva con Swagger.
- **Anotaciones principales:**
 - **@RestController:**
 - Esta anotación le indica a Spring que la clase es un **controlador REST**. Esto significa que el controlador maneja las solicitudes HTTP y devuelve respuestas en formato **JSON** (u otros formatos si es necesario).
 - **@RequestMapping("/api/users")**
 - Establece la URL base para las rutas de este controlador. Es decir, todos los métodos en este controlador estarán accesibles bajo la ruta /api/users. Por ejemplo, GET /api/users para obtener la lista de usuarios.
 - **@CrossOrigin(origins = {"http://localhost:4200"})**
 - Habilita las solicitudes **CORS** (Cross-Origin Resource Sharing) desde un dominio específico (en este caso, http://localhost:4200, que es comúnmente el puerto donde se ejecuta mi aplicación Angular). Esto es útil cuando el cliente (el frontend) y el servidor (el backend) están en dominios diferentes.
 - **@Valid**
 - Spring realiza automáticamente una validación del objeto antes de invocar el método del controlador. Esta validación se hace con las reglas definidas en las anotaciones de validación en la clase del objeto (como @NotNull, @Size, etc.). Si alguno de los campos no cumple con los criterios de validación, los errores se almacenan en el objeto BindingResult.

- **@Tag(name = "Usuarios", description = "API para gestionar los usuarios")**
 - Esto se utiliza para describir y categorizar el controlador en la documentación generada por **Swagger** (una herramienta de documentación de APIs). En este caso, se está etiquetando al controlador con el nombre "Usuarios", que describe que esta API se ocupa de la gestión de usuarios.

El propósito de esta capa es manejar las operaciones relacionadas con los usuarios en la aplicación de una manera estructurada, utilizando el patrón RESTful estableciendo los métodos de operaciones CRUD.

2.3 Despliegue de la aplicación y puesta en marcha

Instrucciones por pasos para llevar a cabo la puesta en marcha de la aplicación.

Nota: Tener instalado el motor de base de datos de MySQL donde se almacenan los datos en las tablas de forma persistente, será la fuente de acceso de nuestra API para recoger los datos almacenados en cada una de las tablas, configurado en el puerto localhost: 3306 para el servidor db_backend_users.

Paso 1:

- Disponer del entorno de trabajo Visual Studio Code.
- Angular depende de Node.js para gestionar las dependencias y ejecutar Scripts por lo que debemos descargar e instalar el Node.js previamente, simplemente vamos al sitio oficial, escogemos la versión estable y los instalamos siguiendo los pasos que nos indica.

Verificar la instalación con los comandos **node -v** y **npm -v**

- Disponer de Angular CLI: una interfaz de línea de comandos que facilita la creación, desarrollo y mantenimiento de proyectos Angular.

Instalar Angular CLI desde la página oficial, entrando en la documentación oficial y siguiendo los pasos indicados en el apartado de instalación, que básicamente es usar el siguiente comando:

npm install -g @angular/cli

Confirmamos que la instalación se ha llevado a cabo con éxito con este comando: **ng versión**

Paso 2:

- Usar los comandos de git para clonar los repositorios que vamos a necesitar:

Clonamos los repositorios desde GitHub tanto del proyecto de Angular como de Spring Boot para tener la parte web y tener la API RESTful incluso desde el propio IDE de VS Code. Clonando los repositorios nos aseguramos de traer todas las dependencias y librerías configuradas en el pom.xml.

Una vez dentro de la carpeta del proyecto, recomendable instalar las dependencias de Angular. Esto se hace a través de npm: **npm install**

- Ejecutamos el proyecto en el puerto 4200.

Para ejecutar el proyecto usamos el comando por defecto: **ng serve**

- Accede a la aplicación de Angular:

Una vez que el servidor esta en marcha, abre el navegador y ve a la siguiente URL:

<http://localhost:4200/login>

Si todo ha ido correctamente deberíamos ver la aplicación web funcionando.

- Recomendación de extensiones para tener en VS Code para Angular: instalando simplemente desde el Marketplace del propio IDE:

Angular Language Service: proporciona el autocompletado, errores y sugerencias de código en tiempo real.

Angular Snippets: ayuda a crear componentes, directivas, servicios, pipes ... bastante útil para facilitar la composición de nuestro proyecto en Angular.

Paso 3:

- Preparar nuestro IDE VS Code para trabajar con Spring Boot y poder usar la API RESTful.
 - o Disponer de Java Development Kit (JDK) recomendable la versión 21 que es con la que trabaja mi API.
 - o Extensiones para VS Code que se pueden encontrar en el propio IDE en el apartado de Marketplace Ctrl+Shift+X
 - **Spring Boot Extension Pack:** Incluye soporte para Spring Boot, Java, Spring Initializr y herramientas relacionadas.
 - **Java Extension Pack:** Proporciona soporte completo para Java.
 - **Spring Boot Dashboard:** nos permite tener un panel con el que poder arrancar la API sin necesidad de usar comandos, además de hacer un seguimiento en tiempo real de cada uno de nuestros endpoints.

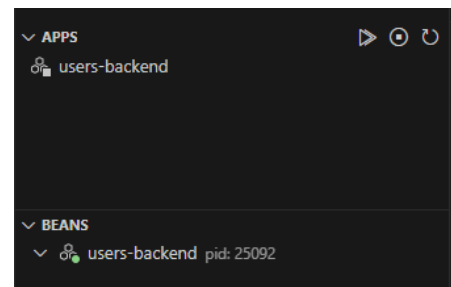
Paso 4:

- Arrancar la API RESTFul:

Nos aseguramos de tener libre el puerto del localhost: 8080 libre de lo contrario debemos cambiarlo en el fichero del pom.xml.

Después abrimos el proyecto de Spring Boot en el IDE de Visual Studio Code y nos aseguramos de estar en la ruta de la carpeta del proyecto antes de proceder a ejecutar el programa.

- Usar Spring Boot Dashboard:
 - Abre la vista de Spring Boot Dashboard en VS Code (en el lateral izquierdo).
 - Detectará automáticamente las aplicaciones Spring Boot en tu proyecto.
 - Haz clic en el botón **play** (▶) para ejecutar tu aplicación.



- Verificar la ejecución:

Una vez que la API esta corriendo abrimos el navegador y accede a esta URL:

<http://localhost:8080/api/users>

Recomendable ir a esta documentación de Swagger para poder interactuar directamente con la API:

<http://localhost:8080/swagger-ui/index.html#/>

3 Planificación y presupuesto

Semana 1-2: Preparación y Configuración

1. Análisis de Requisitos

- Identificar las necesidades del cliente y modelar las funcionalidades.
- Crear diagramas de flujo y diseño inicial.

2. Configuración del Entorno

- Configurar Spring Boot (backend) y Angular (frontend).
- Crear la base de datos en MySQL y definir el esquema inicial.
- Configurar herramientas: Git y CI/CD.

Semana 3-4: Backend - Spring Boot

- Implementación del sistema de autenticación (JWT).
- Desarrollo del módulo de gestión de usuarios.
- Desarrollo de endpoints para el catálogo de productos.
- Desarrollo del módulo de facturación y lógica de negocio.
- Pruebas unitarias y de integración para el backend.

Semana 5-6: Frontend - Angular

- Crear el diseño de las interfaces de usuario (UI/UX).
- Implementar el sistema de autenticación y rutas protegidas.
- Crear páginas para el catálogo de productos (listado, detalles, CRUD).
- Crear páginas para la gestión de facturas (crear, ver, editar).
- Pruebas funcionales en el frontend.

Semana 7-8: Integración, Pruebas y Despliegue

- Integrar el frontend y backend.
- Realizar pruebas de aceptación con el cliente.
- Optimizar rendimiento (consultas, carga de la interfaz).
- Desplegar la aplicación en un servidor (AWS).

3. Recursos Humanos

Roles Necesarios

- Desarrollador Backend (Spring Boot): 1 persona, responsable del desarrollo del backend y la lógica del negocio.
- Desarrollador Frontend (Angular): 1 persona, encargado de implementar la interfaz de usuario y las interacciones.
- Diseñador UI/UX (opcional): 1 persona, para crear un diseño atractivo y funcional.

4. Costos del Proyecto

Estimación de Tiempo

- Total: 8 semanas (2 meses).
- Horas por semana: 40 horas por desarrollador.
- Total horas del proyecto: ~320 horas por rol.

Tarifas por Hora (Referenciales)

- Desarrollador Backend: 25-50 €/hora.
- Desarrollador Frontend: 20-40 €/hora.
- Diseñador UI/UX (opcional): 20-30 €/hora.

Presupuesto Estimado

Rol	Horas Estimadas	Tarifa Promedio	Costo Total
Desarrollador Backend	160	35 €	5,600 €
Desarrollador Frontend	160	30 €	4,800 €
Subtotal			10,400 €
Diseñador UI/UX (opcional)	40	25 €	1,000 €
Total (con extras)			11,400 €

5. Costos de Infraestructura

Servidor de Despliegue

AWS : 45 €/mes dependiendo de la configuración.

4 Documentación Técnica: análisis, diseño, implementación y pruebas.

4.1 *Analiss del sistema y especificación de requisitos*

- Requisitos funcionales:
 - **Gestión de Usuarios**
 - El sistema debe permitir la creación de usuarios.
 - El sistema debe permitir la actualización de usuarios.
 - El sistema debe permitir la eliminación de usuarios.
 - El sistema debe permitir la consulta de usuarios por su ID.
 - El sistema debe permitir la consulta de todos los usuarios.
 - **Roles de Usuario**
 - El sistema debe asignar roles a los usuarios.
 - El sistema debe permitir la asignación del rol "ROLE_USER" a todos los usuarios.
 - El sistema debe permitir la asignación del rol "ROLE_ADMIN" a los usuarios con permisos de administrador, los cuales tendrán posibilidad de gestionar a otros usuarios.
 - **Validación de Datos**
 - El sistema debe validar que el nombre del usuario no esté vacío y no contenga solo espacios en blanco.
 - El sistema debe validar que el apellido del usuario no esté vacío.
 - El sistema debe validar que el correo electrónico del usuario sea válido.
 - El sistema debe validar que el nombre de usuario tenga entre 4 y 20 caracteres.
 - El sistema debe validar que el teléfono del usuario sea un número de 9 dígitos.

- **Seguridad**
 - El sistema debe encriptar la contraseña del usuario antes de guardarla en la base de datos.
 - El sistema debe permitir que el campo "admin" solo sea considerado durante la deserialización de datos JSON.
- **Persistencia de Datos**
 - El sistema debe mapear la entidad Usuario, Cliente y Producto a la tablas correspondientes en la base de datos.
 - El sistema debe ignorar ciertos campos de la entidad Usuario, Cliente y Producto durante las operaciones de persistencia.
- **API de Gestión Usuarios y Clientes.**
 - El sistema debe proporcionar una API para la gestión de usuarios que permita las operaciones de creación, actualización, eliminación y consulta.
- Requisitos no funcionales implementados con **Spring Security**
 - **Autenticación de Usuarios**
 - **Gestión de Credenciales:** autenticar usuarios utilizando diferentes métodos, como autenticación en memoria, autenticación basada en base de datos, autenticación LDAP, OAuth2, etc.
 - **Encriptación de Contraseñas:** Las contraseñas de los usuarios se encriptan utilizando algoritmos seguros como BCrypt, asegurando que las contraseñas no se almacenen en texto plano.
 - **Autorización de Usuarios**
 - **Roles y Permisos:** Spring Security permite definir roles y permisos para los usuarios. Restrinjo el acceso a ciertas partes de mi aplicación basándome en los roles asignados a los usuarios.
 - **Control de Acceso:** Configurar reglas de acceso a nivel de URL, métodos y recursos, asegurando que solo los usuarios autorizados puedan acceder a ciertas funcionalidades, para manipular los datos almacenado en la base de datos.
 - **Gestión de Sesiones**
 - **Control de Sesiones:** Establecer la posibilidad de gestionar las sesiones de los usuarios, incluyendo la configuración de tiempos de expiración y la prevención de sesiones concurrentes.

- **Recordar Sesión:** Funcionalidad de "recordar sesión" que permite a los usuarios permanecer autenticados durante un período prolongado sin necesidad de volver a iniciar sesión.
- **Integración con OAuth2**
 - Autenticación: el usuario se tiene que autenticar en el sistema mediante un usuario y contraseña.
 - Obtención del Token (Bearer Token): genera y obtener un bearer token si todo el proceso ha ido correctamente del servidor al cliente.
 - Uso del Token: el cliente incluye este token de autorización de solicitudes HTTP para acceder a la API protegida así como al mismo tiempo a la base de datos con los datos sensibles.
- Requisito no funcional implementado con **sessionStorage** en Angular de cara a mejorar el rendimiento de la aplicación web.
 - **Descripción:**
 - La aplicación debe utilizar sessionStorage para almacenar datos temporales del usuario con el fin de reducir las solicitudes al servidor y mejorar el tiempo de respuesta de la interfaz de usuario.
 - **Justificación:**
 - El uso de sessionStorage permite almacenar datos en el lado del cliente durante la sesión del navegador, lo que reduce la carga en el servidor y mejora la experiencia del usuario al proporcionar tiempos de carga más rápidos. Importante tener en cuenta este aspecto para lograr una aplicación de calidad.
 - **Criterios de Aceptación:**
 - Los datos que no requieren persistencia más allá de la sesión actual del usuario deben ser almacenados en sessionStorage.
 - La aplicación debe ser capaz de recuperar y utilizar los datos almacenados en sessionStorage de manera eficiente.
 - La implementación debe asegurar que los datos almacenados en sessionStorage sean eliminados al finalizar la sesión del navegador.
 - La utilización de sessionStorage no debe comprometer la seguridad de los datos sensibles del usuario.

- **Medición de Rendimiento:**

- La reducción en el número de solicitudes al servidor debe ser al menos del 20% en comparación con la versión sin sessionStorage.
- El tiempo de carga de las páginas debe mejorar en al menos un 15% en comparación con la versión sin sessionStorage.

4.2 Diseño del sistema:

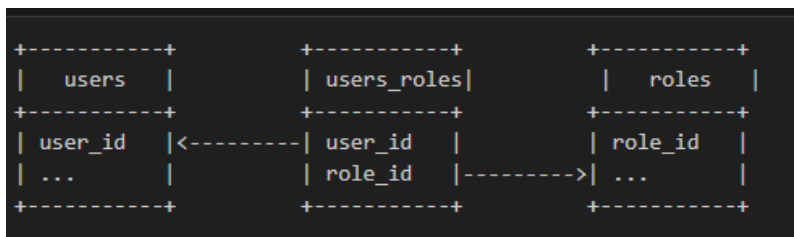
- ✓ **Arquitectura del sistema:**

- Estructura cliente-servidor:
 - Angular: maneja la interfaz del usuario y se comunica con la APIs REST.
 - Spring Boot maneja la lógica de negocio y las conexiones a la base de datos.

4.2.1 Diseño de la Base de Datos

La tabla de los roles de usuarios actúa como una tabla de unión que conecta usuarios y roles.

En una relación de ManyToMany se necesitan dos claves foráneas para poder establecer la conexión entre dos tablas principales que en mi caso son usuarios y roles. Un usuario puede tener más de un rol y un rol puede tener más de un usuario. Esto también nos permite generar múltiples roles para los distintos usuarios y establecer diferentes credenciales de acceso en función del rol que desempeña cada usuario.



- En la tabla de roles de usuario user_id es una clave foránea que referencia a la clave primaria user_id en la tabla usuarios.
- En la tabla roles de usuario el role_id es una clave foránea que referencia a la clave primaria role_id en la tabla de roles.

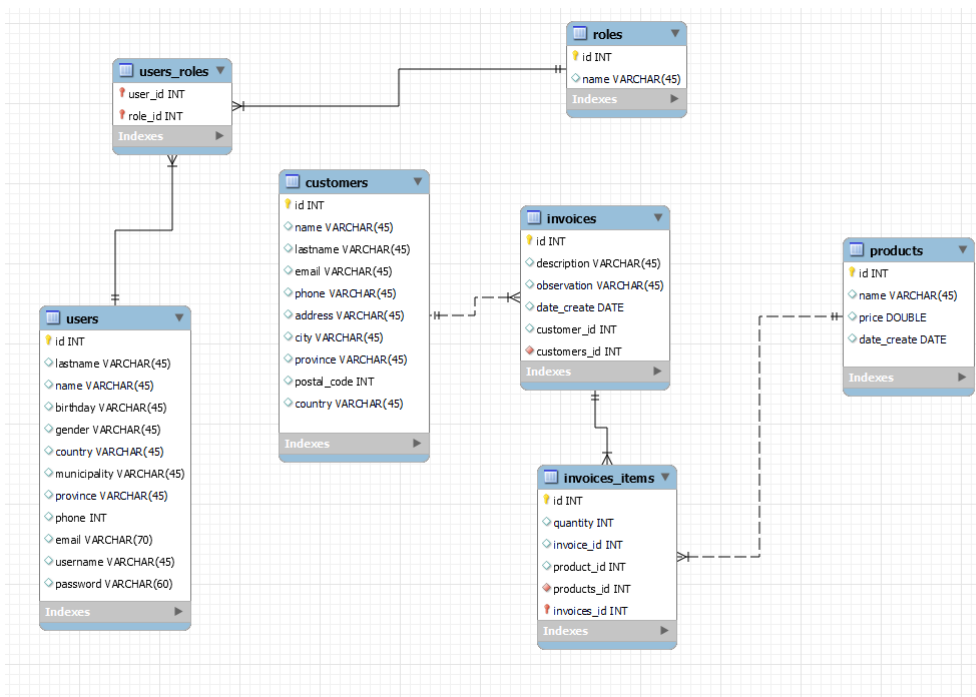
Cada registro en la tabla de unión de roles de usuario conecta un usuario con un rol específico, permitiendo que un usuario tenga múltiples roles y un rol sea asignado a múltiples usuarios. Como se puede apreciar en la imagen gráficamente.

• Tabla users :	
user_id	username
-----	-----
1	Alice
2	Bob

• Tabla roles :	
role_id	role_name
-----	-----
1	Admin
2	User

• Tabla users_roles (tabla de unión):	
user_id	role_id
-----	-----
1	1
1	2
2	2

El resto de del diseño de la base de datos es más simple, Un cliente puede tener más de una factura relación OneToMany. Cada línea de producto en la factura representa un ítem que se ha vendido, con su respectiva cantidad, precio unitario, impuestos aplicables y otros detalles. Una factura puede tener por lo tanto varios ítems relación de OneToMany y finalmente el producto tiene diferentes ítems que se refieren en este caso a la cantidad de un producto vendido. Un producto tiene una relación de OneToMany con los ítems que representan su cantidad.



4.2.2 Diseño API RESTful

- ✓ Formato Swagger UI: <http://localhost:8080/swagger-ui.html>

Swagger UI es una interfaz gráfica interactiva que he generado para visualizar la documentación de la API en un navegador web. Gracias a esta interfaz, podemos ver todos los endpoints disponibles en la API, la descripción de cada uno, los parámetros necesarios, los tipos de respuestas y también probar las peticiones directamente desde la interfaz.

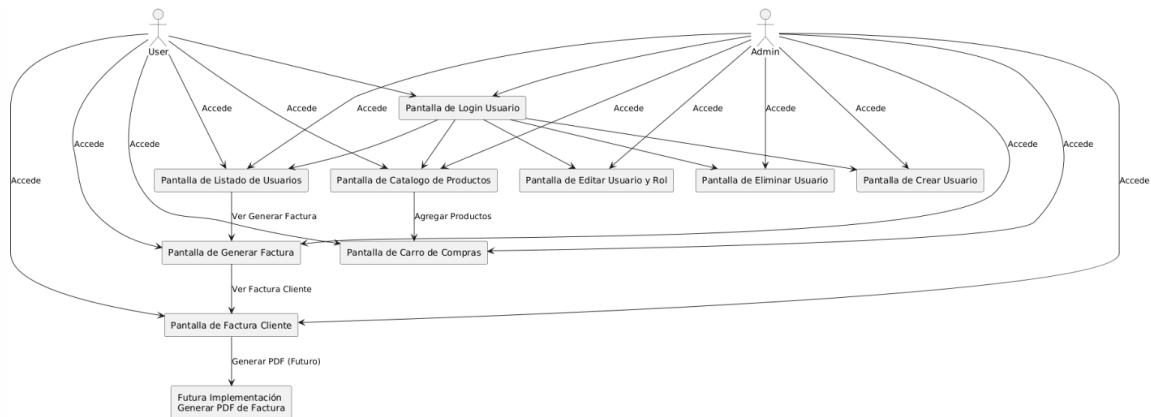
- Interactividad: podemos hacer pruebas de los endpoints directamente desde la interfaz sin necesidad de usar herramientas externas como Postman.
- Visualización clara: nos muestra de manera ordenada todos los endpoints de mi API, sus descripciones, parámetros y posibles respuestas.
- Documentación en tiempo real: al realizar cambios en los controladores o servicios, Swagger UI se actualiza automáticamente, reflejando dichos cambios.

- ✓ Formato OpenAPI JSON: <http://localhost:8080/v3/api-docs/users>

El formato **OpenAPI JSON** proporciona la documentación de la API en formato JSON, conforme a la especificación OpenAPI 3.0. Este formato es útil para la integración automática con otras herramientas, como generar clientes de API o para usar en sistemas de validación o pruebas automáticas.

- Interoperabilidad: Este formato es ampliamente soportado por diversas herramientas que pueden consumirlo, como Postman, Swagger Codegen, o herramientas de validación.
- Automatización: Si necesitas generar clientes API o servidores automáticos en otros lenguajes, el archivo OpenAPI JSON es la fuente de verdad para ello.
- Documentación exhaustiva: Al ser un formato estándar y formal, proporciona una documentación muy detallada sobre la API, ideal para ser procesada por otras herramientas.

4.2.3 Diseño de la Aplicación.



Flujo de autenticación:

- El usuario ingresa su nombre de usuario y contraseña.
- El frontend envía una solicitud de autenticación al backend.
- El backend valida las credenciales.
- Si las credenciales son correctas, el backend genera un token JWT.
- El token JWT se envía al frontend, que lo guarda para futuras solicitudes.

Flujo de creación de un nuevo usuario:

- El administrador accede a la interfaz para crear un nuevo usuario.
- El formulario de creación de usuario en el frontend envía los datos al backend.
- El backend valida los datos y guarda el nuevo usuario en la base de datos.
- El backend responde con el estado de la operación (usuario creado exitosamente o error).
- El frontend muestra el mensaje adecuado al usuario (confirmación de éxito o mensaje de error).

Flujo de consulta de usuarios:

- El usuario o administrador solicita la lista de usuarios a través del frontend.
- El frontend envía una solicitud GET al backend.
- El backend consulta la base de datos y obtiene los usuarios.

- El backend responde con los datos de los usuarios en formato JSON.
- El frontend presenta los datos en una tabla o lista.

4.3 Implementación:

4.3.1 Entorno de desarrollo.

Se ha empleado el IDE de Visual Studio Code, que proporcionan un entorno de desarrollo integrado y un editor de código fuente potente. Además, se emplea GitHub como plataforma de control de versiones, facilitando la colaboración entre desarrolladores, el seguimiento de cambios y la integración continua del código.

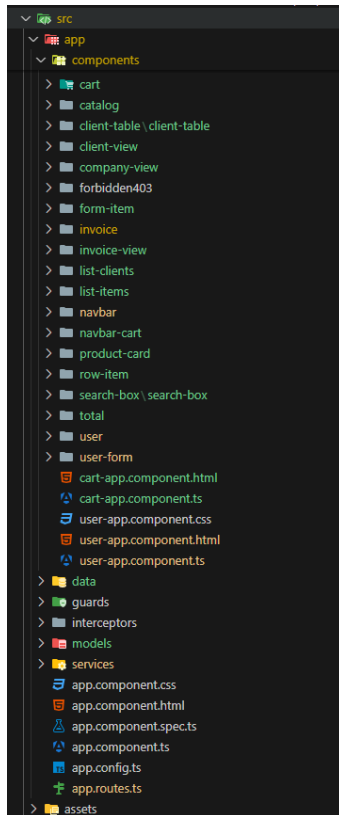
Los frameworks, librerías y extensiones que se han implementado para llevar a cabo con éxito el desarrollo del proyecto son las siguientes:

- **Spring Boot extensión Pack:** es un conjunto de extensiones para VS Code que facilita el desarrollo de aplicaciones con Spring Boot. Este paquete combina varias herramientas en una sola instalación, optimizando la experiencia de desarrollo para quienes usan Spring Framework y Spring Boot.
- **Spring Web:** es un módulo del framework Spring diseñado para construir aplicaciones web. Es una parte clave del ecosistema Spring y proporciona las herramientas y funcionalidades necesarias para manejar solicitudes HTTP, gestionar rutas, crear controladores y desarrollar APIs RESTful o aplicaciones web basadas en MVC.
- **Spring Data JPA:** es un módulo del ecosistema Spring Data que simplifica el acceso y manejo de datos en aplicaciones Java, utilizando el estándar Java Persistence API (JPA). Proporciona una abstracción sobre JPA para facilitar las operaciones de persistencia en bases de datos relacionales, reduciendo la cantidad de código necesario para realizar consultas y operaciones CRUD (Crear, Leer, Actualizar, Eliminar).
- **Spring Boot DevTools:** es una herramienta diseñada para mejorar la productividad del desarrollo al proporcionar características útiles durante la fase de desarrollo, como la recarga automática de la aplicación, deshabilitación de la caché en plantillas, y un reinicio rápido del servidor. Detecta automáticamente los cambios en los archivos de código fuente y reinicia la aplicación sin necesidad de hacerlo manualmente. Permite conectarse a la aplicación de forma remota para depuración en vivo. Usando la técnica de reinicio de clase, la herramienta reinicia únicamente las clases necesarias, lo que reduce el tiempo de reinicio en comparación con un reinicio completo.

- **Extension Pack Java:** es un conjunto de extensiones para VS Code diseñado para proporcionar un entorno completo de desarrollo en Java. Este paquete simplifica la configuración y mejora la experiencia de desarrollo al integrar herramientas esenciales para codificar, depurar y ejecutar proyectos Java, tanto para aplicaciones simples como para proyectos más complejos como los basados en Spring Boot.
- **MySQL Driver SQL:** Este driver implementa la especificación JDBC, lo que te permite ejecutar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en MySQL desde tu aplicación. Lo más importante facilitar la conexión con la base de datos desde mi aplicación Java.
- **Spring Security:** framework para manejar la autenticación, la autorización y protección de las rutas HTTPS de la aplicación, así como integración con proveedores de autenticación mediante OAuth2.
- **Springdoc Open Api:** es una biblioteca de código abierto que facilita la integración de OpenAPI 3.0 con aplicaciones basadas en Spring Boot. OpenAPI (anteriormente conocido como Swagger) es un estándar para describir, consumir y visualizar APIs RESTful.
Springdoc OpenAPI genera automáticamente la documentación de tu API REST a partir de las anotaciones y configuraciones que utilices en tu aplicación Spring Boot. Esta documentación se presenta en una interfaz interactiva que permite a los usuarios explorar y probar las APIs sin necesidad de escribir código adicional.
- **Angular Language Service:** herramienta de desarrollo que ofrece funcionalidades como autocompletado, resaltado de errores y navegación de código. Imprescindible para aprovechar al máximo las capacidades del framework de Angular.
- **Angular Snippets:** herramientas útil para generar rápidamente estructuras comunes de Angular, como componentes, servicios, directivas, módulos, templates HTML, rutas...

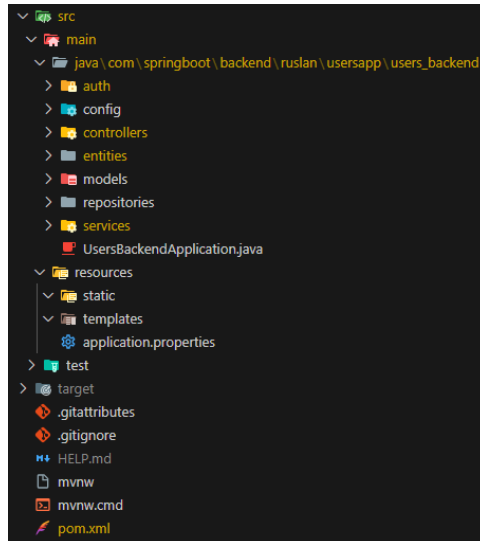
4.3.2 Estructura del código.

- Estructura de un proyecto Angular:



- Principales elementos del proyecto:
 - Componentes:
 - Piezas fundamentales del proyecto. Responsables de controlar una porción de la interfaz de usuario (UI) y se compone de tres elementos clave: **HTML** (vista), **CSS** (estilos), y **TypeScript** (lógica).
 - Data:
 - Contiene datos que necesitamos compartir entre varias partes de la aplicación.
 - Guards:
 - Verifica si el usuario está autenticado, si el token ha expirado y si el usuario tiene permisos de administrador antes de permitir el acceso a una ruta protegida. Si alguna de estas condiciones no se cumple, el usuario es redirigido a la página de inicio de sesión o a la página de acceso prohibido según corresponda.
 - Models:
 - Define las interfaces o clases de modelos de datos para estructurar la información de un objeto.
 - Interceptors:
 - Manejan errores, ayudan a configurar encabezados personalizados y agregan tokens de autenticación.

- Services:
 - Realiza solicitudes HTTP a la API, maneja la lógica de negocio y gestiona el estado compartido entre componentes.
- *Estructura del proyecto de Spring Boot: APIRESTFul:*



- Principales elementos del proyecto:
 - Auth:
 - Manejan la autenticación de usuarios, establece filtros de seguridad para establecer permisos de acceso a las distintas partes de los endpoints de la API.
 - Config
 - Establecen la configuración de la documentación de Swagger.
 - Controllers:
 - Contienen los controladores que gestionan las solicitudes HTTP (API REST)
 - Entities:
 - Mapean las propiedades de las entidades del dominio directamente a las tablas de la base de datos mediante Hibernate.
 - Models:
 - Contienen los DTOs estructuras utilizadas en las solicitudes o respuestas de la API.
 - Repositories:
 - Contiene las interfaces para acceder a la base de datos usando Spring Data JPA.
 - Services:
 - Contienen métodos que contienen la lógica para interactuar con los repositorios y procesar datos.

Librerías que se han configurado en el archivo pom.xml para crear y validar JSON Web Tokens:

- **JSON WebToken:** se utilizan para crear y validar JSON Web Tokens (JWT) en aplicaciones Java. JWT es un estándar abierto (RFC 7519) que define una forma compacta y autónoma para transmitir información de forma segura entre las partes como un objeto JSON.
 - Dependencias para trabajar con JWTs dentro del archivo pom.xml de mi proyecto:
 - **jjwt-api:** Define la API para trabajar con JWTs, pero no implementa la lógica de fondo. Se utiliza para la interfaz de usuario y la interacción básica con tokens.
 - **jjwt-impl:** Contiene las implementaciones reales de las funciones para crear, firmar, y verificar JWT. Esta librería hace el trabajo pesado.
 - **jjwt-jackson:** Proporciona funcionalidad de serialización/deserialización usando Jackson para convertir entre JSON y objetos Java, lo que es necesario para procesar los datos dentro de un JWT.

En conjunto, estas tres dependencias te permiten:

- a) **Crear** un JWT con las claims (información).
- b) **Firmar** el JWT con un algoritmo y una clave secreta.
- c) **Verificar** la autenticidad del JWT.
- d) **Leer** las claims y los datos del JWT en formato JSON.

Este conjunto es comúnmente usado en aplicaciones que implementan autenticación basada en JWT, como aplicaciones de API RESTful.

4.3.3 Cuestiones de diseño e implementación reseñables.

- En el código de Angular cabe destacar el uso de **sessionStorage**: una API de almacenamiento web proporcionada por los navegadores que permite almacenar datos de manera local en el navegador del usuario. Dichos datos almacenados solo persisten durante la sesión de navegación, lo que significa que se eliminan cuando la pestaña del navegador se cierra.

Esta implementada en el servicio auth.service.ts donde la utilizo para almacenar y recuperar el estado de autenticación del usuario y el token de autenticación, de ese modo consigo que las credenciales estén activas aunque se recargue el navegador sin necesidad de volver a iniciar la sesión de nuevo, además, aunque naveguemos a diferentes partes de la aplicación la sesión se mantiene activa al menos hasta que se cierre el navegador, proporcionando una experiencia de usuario más fluida y consistente.

Características destacables del sessionStorage:

- Persistencia del estado de autenticación:
Mantiene la sesión activa incluso si la página se recarga, sin su implementación toda la información de autenticación almacenada en la memoria del servicio se perdería al recargar la página.
- Seguridad:
Almacena los datos solo durante la sesión de navegación cuando la pestaña o el navegador se cierra los datos se eliminan automáticamente lo que proporciona una capa adicional de seguridad.
Acceso controlado al almacenar el token de autenticación y el estado de inicio de sesión en sessionStorage lo que asegura que solo el código del lado del cliente en la misma sesión del navegador puede acceder a estos datos.
- Mejora la experiencia del usuario:
Los usuarios pueden navegar por la aplicación sin interrupciones, incluso si recargan la página o abren nuevas pestañas dentro de la misma sesión.
Los usuarios no necesitan volver a iniciar sesión cada vez que recargan la página, lo que mejora la experiencia de usuario y reduce la fricción en el uso de la aplicación.

4.4 Pruebas.

- Validación de las respuestas de la API.

Se ha desarrollado un plan de pruebas de las respuestas de la API para verificar que cumplen con las expectativas. Comprobar el estado HTTP, validar datos específicos en la respuesta, como valores en el cuerpo JSON, si cumple con los campos establecidos en la respuesta JSON, si la respuesta está vacía, comprobar el tiempo de respuesta, asegurar que ciertos datos no estén duplicados... A continuación, se exponen una serie de test implementados en los scripts de los endpoints del Postman.

DESCRIPCION DE LA PRUEBA	CONDICIONES PREVIAS	DATOS QUE SE ESPERAN	PASOS PARA EJECUTAR	RESULTADO FINAL
Asegurar que devuelve un listado de usuarios de la base de datos mediante el endpoint GET: URL localhost:8080 a la que se envía una respuesta HTTP para recuperar los datos del servidor.	Que el servidor proporcione una respuesta a la solicitud	Un respuesta 200 OK	Se establece el test en script en el endpoint GET: http://localhost:8080/api/users	PASSED
Verificar que el cuerpo de la respuesta no este vacía	Que las propiedades estén correctamente establecidas en el objeto con sus datos establecidos en la BBDD	Se espera la devolución completa de un listado de usuarios en formato JSON con todos los campos y datos.	Se establece el test en script en el endpoint GET: http://localhost:8080/api/users	PASSED

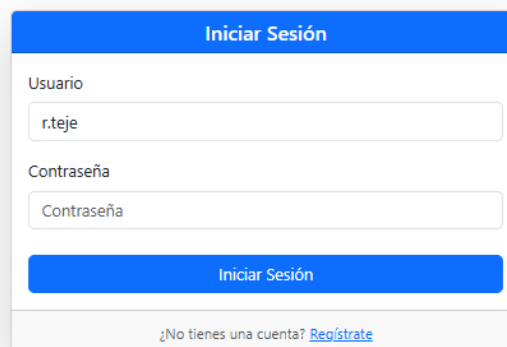
Validar que se devuelven las propiedades requeridas de los usuarios como id, name, lastname, birthday, username, email, phone, password y sus roles.	Los objetos contengan todas las propiedades requeridas	Se espera la devolución de todas las propiedades en el cuerpo JSON	Se establece el test en script en el endpoint GET: http://localhost:8080/api/users	PASSED
Verificar que los nombres de los username son únicos	Al crear nuevos usuarios verificar que los nombres no estén duplicados	Se espera que no devuelva nombres de usuarios repetidos	Se establece el test en script en el endpoint GET: http://localhost:8080/api/users	PASSED
Verificar que los correos son únicos	Al crear nuevos usuarios validar correos únicos	Se espera que no devuelva correos duplicados	Se establece el test en script en el endpoint GET: http://localhost:8080/api/users	PASSED
Verificar que el tiempo de respuesta		Se espera un tiempo de respuesta menor a 2 segundos	Se establece el test en script en el endpoint GET: http://localhost:8080/api/users	PASSED
POST: Verificar que la respuesta tenga un estado 201 (OK)	Tener en el body del Postman el objeto JSON con las propiedades y los datos requeridos	Al lanzar el endpoint de la API se espera el estado de la respuesta 201	Se establece el test en script en el endpoint POST: http://localhost:8080/api/users	
Verificar que el cuerpo de la respuesta no este vacía	Crear un objeto JSON con todas sus propiedades requeridas y sus datos	Se espera la devolución completa de de nuestro objeto usuario con sus campos y datos creados.	Se establece el test en script en el endpoint POST: http://localhost:8080/api/users	PASSED

Comprobar que el formato del email sea válido, que no contenga dos .. consecutivos que terminen en .es o .com	Crear un objeto usuario con el formato de correo establecido	Devuelve un objeto con el correo valido	Se establece el test en script en el endpoint POST: http://localhost:8080/api/users	PASSED
Validar el dominio de correo @educa.jcyl.es	Se crea un objeto con el dominio @educa.jcyl.es	Devuelve un objeto con el dominio valido	Se establece el test en script en el endpoint POST: {{url}}/api/users	PASSED
GET: Búsqueda de usuario por ID	Introducir el ID del usuario a buscar en la url del endpoint	Devuelve un usuario con el ID de la consulta	Se establece el test en script en el endpoint GET: {{url}}/api/users/{{ID}}	PASSED
PUT: Editar usuario	Introducir el teléfono con 9 dígitos	Devuelve un usuario editado con la propiedad teléfono con 9 digitos	Se establece el test en script en el endpoint PUT: {{url}}/api/users/{{ID}}	PASSED
DELETE: Comprobar que devuelve el estado 204	Se introduce el ID del usuario a borrar por la URL	Que el servidor proporcione una respuesta de estado 204 al borrar con éxito al usuario	Se establece el test en script en el endpoint DELETE: {{url}}/api/users/{{ID}}	PASSED

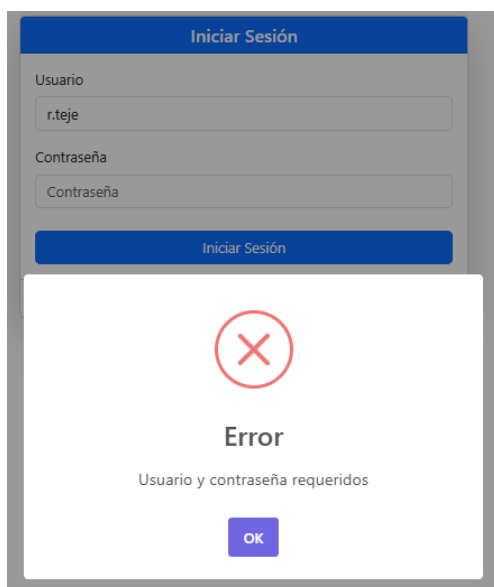
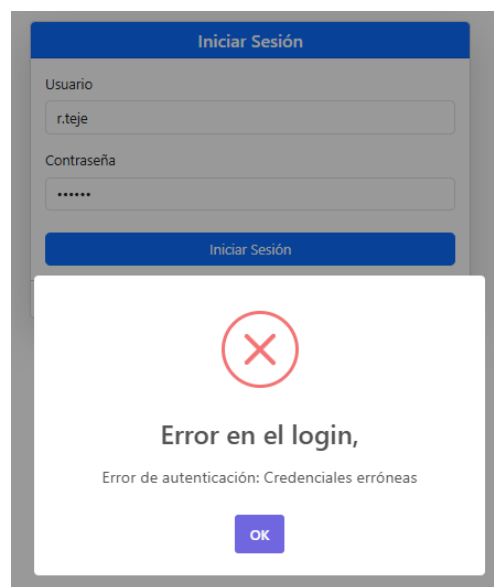
5 Manuales de usuario

5.1 Manual de usuario






Inicio de sesión del usuario, en la pantalla inicial de la aplicación se solicitan las credenciales del nombre del usuario y la contraseña.

The image shows a login form titled 'Iniciar Sesión'. It has two input fields: 'Usuario' with the text 'r.teje' and 'Contraseña' with the text 'Contraseña'. Below the fields is a blue 'Iniciar Sesión' button. At the bottom, there is a link that says '¿No tienes una cuenta? [Regístrate](#)'.

Si las credenciales del usuario son incorrectas salta un mensaje de aviso de información indicando que las credenciales no son válidas, también si el usuario no introduce alguno de los campos solicitados salta otro aviso indicando que los campos son requeridos:

The image shows the login form with a modal error message displayed. The message has a red 'X' icon and the text 'Error' followed by 'Usuario y contraseña requeridos'. There is an 'OK' button at the bottom.The image shows the login form with a modal error message displayed. The message has a red 'X' icon and the text 'Error en el login,' followed by 'Error de autenticación: Credenciales erróneas'. There is an 'OK' button at the bottom.

Una vez que entramos con las credenciales de un usuario administrador se nos presenta la siguiente pantalla: donde observamos que el usuario admin puede realizar las operaciones de gestión sobre otros usuarios del sistema. Tiene credenciales para crear nuevos usuarios, editar sus campos e incluso eliminar un usuario del sistema.






ClientSphere Usuarios  Nuevo Usuario  Factura cliente  Catálogo  r.teje  Logout

Listado de usuarios!


[Crear usuario](#)

id	Apellidos	Nombre	Fecha nacimiento	Género	País	Municipio	Provincia	Teléfono	Email	Usuario	Eliminar	Modificar
20	Muñoz Santos	Roberto	15/10/1980	Masculino	España	Valladolid	Valladolid	983697854	ruslan.tejzap@educajcy.es	r.teje	Borrar	Editar
21	García Marquez	Manuel	10/10/1975	Masculino	España	Sevilla	Sevilla	983659874	o.sanchez@mail.com	o.sanchez	Borrar	Editar
24	Huertas Marquez	Sandra	25/06/1981	Femenino	España	Valladolid	Valladolid	687485235	s.huertas@mail.com	s.huertas	Borrar	Editar
25	Arribas	Hugo Exteberria	15/03/1982	Masculino	España	Valladolid	Valladolid	983659874	h.arribas@mail.com	arribas	Borrar	Editar
41	García	marcos	15/03/1982	Masculino	España	Huelva	Huelva	983659874	prueba1@mail.com	prueba1	Borrar	Editar
44	Tejerina	Ruslan	15/03/1987	Masculino	España	Valladolid	Valladolid	983659874	r.teje@mail.com	r.teje	Borrar	Editar
45	Lopez Marquez	Juan	15/03/1980	Masculino	España	Soria	Soria	983659874	l.marquez@mail.com	j.lopez	Borrar	Editar
46	Lopez	Marcos	20/12/1984	Masculino	España	Valladolid	Valladolid	986547895	lpez@mail.com	lopez	Borrar	Editar

Si entramos en el apartado de crear nuevos usuarios observamos que disponemos de un formulario a rellenar con los siguientes campos:

ClientSphere Usuarios  Nuevo Usuario  Factura cliente  Catálogo  r.teje  Logout

Formulario Crear Usuario

Lastname
 Name
 Fecha nacimiento
 Género
 País
 Municipality
 Province
 Phone
 Email
 Username
 Password
☐  Admin

Todos los campos son requeridos al generar un nuevo usuario en el sistema, además se da la opción de otorgar el rol de administrador en el caso que así lo requiera.

ClientSphere

Formulario Crear Usuario

Lastrame

El campo lastrame no debe estar vacío

Name

El campo name no debe estar vacío

Fecha nacimiento

El campo birthday no debe estar vacío

Género

El campo gender no debe estar vacío

País

El campo country no debe estar vacío

Municipality

El campo municipality no debe estar vacío

Province

El campo province no debe estar vacío

Phone

El campo phone no debe ser nulo

Email

El campo email no debe estar vacío

Username

El campo username no debe estar vacío

Password

El campo password no debe estar vacío

Admin

Crear

Limpiar

Se validan que todos los campos estén completos, que el nombre de usuario no se repita y el teléfono debe contener 9 dígitos. Si todo se rellena correctamente debería mostrar el usuario en la BBDD e indicar en la siguiente pantalla el mensaje de:

ClientSphere

Usuarios

Nuevo Usuario

Factura cliente

Catálogo

r.teje Logout

Listado de usuarios!

Crear usuario

id	Apellidos	Nombre	Fecha nacimiento	Género	País	Municipio	Provincia	Teléfono	Email	Usuario	Eliminar	Modificar
85	Gomez Arribas	Marta	24/08/1975	Femenino	España	Segovia	Segovia	984569852	marta.gomez@educajcyl.es	marta	Borrar	Editar

✓

Agregado nuevo usuario!

Usuario guardado con éxito!

OK

En la BBDD debe aparecer registrada la nueva usuaria con la contraseña encriptada por seguridad al final de la lista.

ender	country	municipality	province	phone	email	username	password
sculino	España	Huelva	Huelva	983659874	prueba1@mail.com	prueba1	\$2a\$10\$R0pmNOE.z4mwTjkdCTzf.Vclbcs8dayy4ANaH0bbMqkT94PBHnm
sculino	España	Valladolid	Valladolid	983659874	r.teje@mail.com	r.teje	\$2a\$10\$64Izvg3hEA6NY/fuzpbaOOmqePjb8fGNMSJCXIBBqLdf7zk5m2
sculino	España	Soria	Soria	983659874	l.marquez@mail.com	j.lopez	\$2a\$10\$TKgyCjh2p61PE4y.vEx8S.pZRAPY40fJLKeHIZj0ZSxo4gAvVx96
sculino	España	Valladolid	Valladolid	986547895	lopez@mail.com	lopez	\$2a\$10\$FNdLluzHQcCmGVLJRbVS.OhG.cdNwYJ0sJY9aLRMBj58MhPd6Fr7a
sculino	España	Valladolid	Valladolid	983659874	hola@mail.com	pepe	\$2a\$10\$Qem89pHtH37NkVY89vJ2.odtDbswRc7Ksedcm.RA1TCK5UjmgGO
sculino	España	Valladolid	Valladolid	983659874	hola1@mail.com	pepe1	\$2a\$10\$ui30HFVxyia2mUGDg5Aile183WLYaNBcy62uOC9/nSCJf.rs0JIS
sculino	España	Valladolid	Valladolid	983659874	oscargonzalez@mail.com	oscar23	\$2a\$10\$SxExI0LmbR88YQaZLJEQuYpEGewFxWv1ACkdyTelSS5S2PBWtnq
sculino	España	Valladolid	Valladolid	983659874	jo.doe@company.es	hgfhggbv	\$2a\$10\$2d.a9LxY4H9/vOWdyHlVQlorA6zPeOvZUskmozOgMSrCya2zsmY
sculino	España	Valladolid	Valladolid	983659874	john.doe@company.es	sdftrfbfdg	\$2a\$10\$4ZTM/SuLkS40pHDLR/dKCOymEyBdZ8i8Qa73jTwZjBjYzB.r9.
sculino	España	Valladolid	Valladolid	983659874	john@domain.es	qwerty	\$2a\$10\$jrV2oU5yBPof7Qo93A4.grkQSQQRnHixAxxj/5f6mfdgEHLuey
sculino	España	Valladolid	Valladolid	983659874	jodsfnn@domain.es	sdfsdf	\$2a\$10\$oeowbY/ZnkYz4jJkq6c.4vFqfyPOnFG6Mhz/3eFQugPDTHQg59y
sculino	España	Valladolid	Valladolid	983659874	jodpwmhnn@domain.es	rutru	\$2a\$10\$djYpqKJREvntpdBqElv2cefQnpxtuVHE8BbLk0vv7onZAhGZyk9y
sculino	España	Valladolid	Valladolid	983659874	jod.pwmhnn@domain.es	tyuuytt	\$2a\$10\$ZymwGBCV97hskG6Y0cDOaE5cfwJvwYjTManXHkV3a88xDOHt
menino	España	Segovia	Segovia	984569852	marta.gomez@educa.jcyl.es	marta	\$2a\$10\$4evlJpftGVIWJ4m0PrXhuSaU92GvjehyOE6ZDVQ7NZ1jVma/P2y

Si por el motivo que fuera necesitásemos cambiar alguno de los datos de uno o varios campos, entramos en editar y procedemos a modificar, como detalle importante por motivos de seguridad no existe la posibilidad de modificar ni el nombre de usuario ni la contraseña en este formulario. Se facilita un formulario con los datos ya volcados para evitar que por descuido se cambien otros datos que no son necesarios de editar, se da la opción al usuario de limpiar el formulario en caso de que lo necesite:

Formulario Editar Usuario

Gomez Arribas

Marta

24/08/1975

Femenino

España

Segovia

Segovia

984569852

marta.gomez@educa.jcyl.es

marta

☐
Admin

Actualizar


Limpiar

Una vez cambiados los campos se actualizan los datos en la BBDD y muestra al usuario la siguiente pantalla de confirmación:

Listado de usuarios!

[Crear usuario](#)

id	Apellidos	Nombre	Fecha nacimiento	Género	País	Municipio	Provincia	Teléfono	Email	Usuario	Eliminar	Modificar
85	Gomez Arribas	Marta	24/08/1975	Femenino	España	Segovia	Segovia	984569852	marta.gomez@gmail.es	marta	Borrar	Editar



Actualizado!

Usuario editado con éxito!


[OK](#)

Si procedemos a eliminar a nuestro usuario nos salta un aviso de confirmación antes de proceder a eliminarlo definitivamente dando la opción de cancelarlo, si no deseamos eliminarlo finalmente.

Listado de usuarios!

[Crear usuario](#)

id	Apellidos	Nombre	Fecha nacimiento	Género	País	Municipio	Provincia	Teléfono	Email	Usuario	Eliminar	Modificar
85	Gomez Arribas	Marta	24/08/1975	Femenino	España	Segovia	Segovia	984569852	marta.gomez@gmail.es	marta	Borrar	Editar

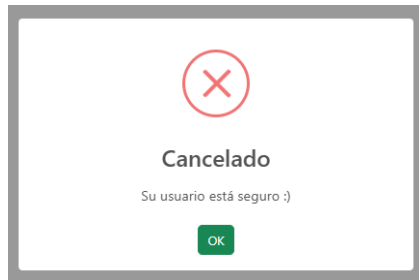


Desea eliminar realmente?

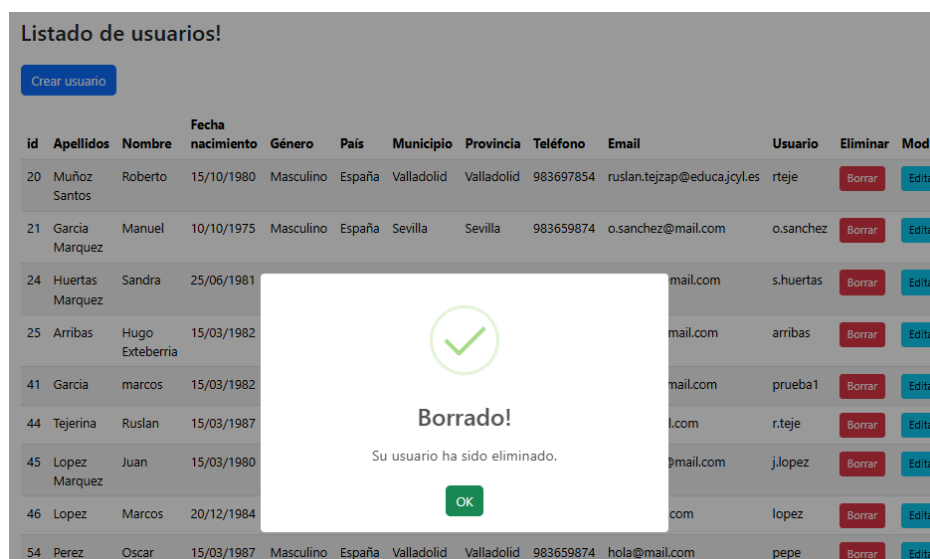
Cuidado el usuario será eliminado del sistema

[No, cancelar!](#)
[Sí, eliminar!](#)

Si optamos por cancelar salta este otro aviso.



Si confirmamos el borrado nos traslada a la pantalla donde se listan los usuarios confirmando el borrado:




Si entramos como usuario sin permisos del administrador no se nos permite realizar operaciones de crear o modificar a otros usuarios. Directamente la aplicación nos oculta esas opciones en el navegador, solo nos lista a otros usuarios en el

Listado de usuarios!

id	Apellidos	Nombre	Fecha nacimiento	Género	País	Municipio	Provincia	Teléfono	Email
20	Muñoz Santos	Roberto	15/10/1980	Masculino	España	Valladolid	Valladolid	983697854	ruslan.tejzap@educa.jcyl.es
21	Garcia Marquez	Manuel	10/10/1975	Masculino	España	Sevilla	Sevilla	983659874	o.sanchez@mail.com
24	Huertas Marquez	Sandra	25/06/1981	Femenino	España	Valladolid	Valladolid	687485235	s.huertas@mail.com
25	Arribas	Hugo Exteberria	15/03/1982	Masculino	España	Valladolid	Valladolid	983659874	h.arribas@mail.com
41	Garcia	marcos	15/03/1982	Masculino	España	Huelva	Huelva	983659874	prueba1@mail.com
44	Tejerina	Ruslan	15/03/1987	Masculino	España	Valladolid	Valladolid	983659874	r.teje@mail.com
45	Lopez Marquez	Juan	15/03/1980	Masculino	España	Soria	Soria	983659874	l.marquez@mail.com
46	Lopez	Marcos	20/12/1984	Masculino	España	Valladolid	Valladolid	986547895	lpez@mail.com
54	Perez	Oscar	15/03/1987	Masculino	España	Valladolid	Valladolid	983659874	hola@mail.com
55	Perez	Oscar	15/03/1987	Masculino	España	Valladolid	Valladolid	983659874	hola1@mail.com
56	Gonzalez	Mario	15/03/1987	Masculino	España	Valladolid	Valladolid	983659874	oscardgonzalez@mail.com
67	Gonzalez	Mario	15/03/1987	Masculino	España	Valladolid	Valladolid	983659874	jo.doe@company.es

Si se intentase acceder a través de la url: <http://localhost:4200/users/create> sin las credenciales de administrador para crear nuevos usuarios nos redirige a la página que nos indica de que no tenemos autorización, la ruta esta protegida con JWT (JSON Web Token), es un mecanismo de autenticación y autorización para que solo los usuarios autorizados puedan acceder al recurso. El token contiene: identidad del usuario, los roles de permisos, tiempo de expiración y una información firmada digitalmente para evitar manipulaciones.

Acceso denegado

 No tiene autorización de acceso a esta página.

Por favor, [inicie sesión](#) con una cuenta autorizada.

En la sección de factura cliente, el usuario tiene acceso al listado de clientes de la empresa con los datos de cada uno de ellos, además dispone de un buscador que localiza a los clientes por el apellido y muestra los datos de un cliente en particular. Tiene la posibilidad de generar los detalles de la factura agregando productos o servicios, precios y cantidad, todo ello genera un total que posteriormente trasladamos a la hora de generar la factura para un cliente en formato pdf.


Factura

Detalles de la Factura

Factura n°: 1234

Nombre: Productos tecnológicos

Información de la Empresa



ClientSphere

Correo: miempresa@example.com

Teléfono: 123456789

Dirección: Calle Falsa 123

Ciudad: Madrid

Código postal: 28001

País: España

Fecha de creación: 15/12/2024

Número fiscal: 123456789

Descripción: Empresa dedicada a la fabricación y distribución de productos tecnológicos.

Información del Cliente

#	4
Nombre	Roberto
Apellidos	Garcia Marquez
Email	user123@domain.com
Teléfono	983569874
Dirección	Avenida Falsa, 123
Ciudad	Valladolid
Provincia	Valladolid
Código Postal	44789
País	España

Descargar Factura

Limpiar Cliente

Lista de Clientes

id	Nombre	Apellido	Correo	Telefono	Dirección	Ciudad	Provincia	Código Postal	País
1	Manuel	Lopez Gutierrez	user123@domain.com	983356298	Avenida Falsa, 23	Barcelona	Barcelona	49023	España
2	Carlos	Marcos	user123@domain.com	983258974	Avenida Falsa, 123	Sevilla	Sevilla	48653	España
3	Manuel	Perez	user123@gmail.com	983258969	Avenida Falsa, 123	Malaga	Malaga	47256	España
4	Roberto	Garcia Marquez	user123@domain.com	983569874	Avenida Falsa, 123	Valladolid	Valladolid	44789	España

Detalle de la factura

Id	Producto	Cantidad	Precio	Eliminar
1	Tarjeta Gráfica SLIM 16GB GDDR6X DLSS3	1	1246	Borrar
2	Placa Base MSI MAG X670E TOMAHAWK WIFI	2	314	Borrar
3	Memoria RAM DDR4 3200 MHz 16GB 2x8GB CL16	5	40	Borrar

Agregar Item

Total

Importe:

2,074.00 €

El usuario tiene la opción de generar y descargar la factura en formato PDF con todos los detalles del cliente, los productos, precios y el total adeudado además de mostrar los detalles del pago:

Factura

ClientSphere

Referencia: #FACT-001

Fecha: 19-12-2024

Nº: Factura: 1234

Facturado a:

Roberto
García Marquez
user123@domain.com
983569874

Dirección de envío:

Avenida Falsa, 123
44789
Valladolid
Valladolid
España

Detalles de la empresa

Nº Fiscal: 123456789
Clintersphere
Calle Falsa 123
Madrid, Madrid, 28001
España

Total adeudado

2509.54 €

Fecha de vencimiento: 28-01-2025

Items	Descripción	Cantidad	Precio	Total
Tarjeta Gráfica	SLIM 16GB GDDR6X DLSS3	1	1246 €	1246 €
Placa Base	MSI MAG X670E TOMAHAWK WIFI	2	314 €	628 €
Memoria RAM	Kingston FURY Beast DDR4 3200 MHz 16GB 2x8GB CL16	5	40 €	200 €
Móvil	Apple iPhone 16 128GB	1	1000 €	1000 €
Subtotal				2074.00 €
Impuesto Total (21%)				435.54 €
Total				2509.54 €

Detalles del pago

Métodos de pago disponibles: Transferencia bancaria, tarjeta de crédito, PayPal.

Datos de cuenta bancaria:

Número de cuenta: 1234 5678 90 1234567890

IBAN: ESXX XXXX XXXX XXXX XXXX XXXX



Beneficiario: ClientSphere

Plazo de pago: antes del 28/01/2025.













Referencia de pago: **FACT-00123-ABCDEF**

Número de factura: FACT-001


El último apartado es el catalogo de los productos disponibles con su descripción y su precio que un usuario podrá ir agregando a la cesta tantas vez que lo requiera.

Productos  Carro Compra (0) 

Catálogo Productos


Teclado Mecánico 210 €  Gaming RGB Inalámbrico ROG RX Red QWERTY Español Negro Agregar al carro	Portatil 2000 €  Intel Core i9-14900HX/32GB/1TB SSD/RTX 4070/18 Agregar al carro	Placa base 160 €  MSI PRO B650-S WIFI DDR5 de doble canal 7200+MHz Agregar al carro
Memoria interna 110 €  Kingston FURY Beast DDR5 6000MHz Agregar al carro	Tarjeta de video 1134 €  Gigabyte GeForce RTX 4080, diseñada para optimizar y medir la latencia del sistema Agregar al carro	Ordenador Sobremesa 699 €  E1 12TC-033ES Intel Core i5-12400F/16GB/1TB SSD/RTX 3060 Agregar al carro
Móvil 799 €  5G 16/512GB Flowy Emerald Libre Agregar al carro	Auriculares Inalámbricos 90 €  Cancelación de Ruido Adaptativa Negros Agregar al carro	Monitor de ordenador 115 €  MP273AW 27" LED IPS FullHD 100Hz Agregar al carro
Silla oficina 80 €  Silla de Oficina Blanca Agregar al carro	Mesa oficina 160 €  Mesa de Oficina Negra Agregar al carro	Ratón inalámbrico 30 €  Marathon Mouse Ratón Inalámbrico Agregar al carro

Según los va añadiendo a la cesta automáticamente se suman las cantidades, los precios y nos muestra el total de los productos en la cesta, además si nos hemos equivocado a la hora de seleccionar un producto tiene la opción de quitar el producto de la cesta y los precios se vuelven a recalcular.

Carro Compra (4) 

Listado de Productos

Producto	Precio	Cantidad	Total	Eliminar
Teclado Mecánico	210	1	210	eliminar
Ordenador Sobremesa	699	1	699	eliminar
Móvil	799	1	799	eliminar
Mesa oficina	160	1	160	eliminar
			Total	1868 €




Shopping Cart


Nuevo producto agregado al carro!

[OK](#)

Antes de quitar el producto de la cesta se nos muestra un mensaje previo dando la opción de rectificar:

Listado de Productos 

Producto	Precio	Cantidad	Total	Eliminar
Teclado Mecánico	210	1	210	<button>eliminar</button>
Ordenador Sobremesa	699	1	699	<button>eliminar</button>
Móvil	799	1	799	<button>eliminar</button>
Mesa oficina	160	1	160	<button>eliminar</button>
			Total	1868 €




Esta seguro que desea eliminar?


Cuidado el item se eliminara del carro de compras!

Si, eliminar!


Cancel

Una vez eliminado nos confirma que se ha llevado a cabo con éxito:

Carro Compra (3) 

Listado de Productos 

Producto	Precio	Cantidad	Total	Eliminar
Teclado Mecánico	210	1	210	<button>eliminar</button>
Móvil	799	1	799	<button>eliminar</button>
Mesa oficina	160	1	160	<button>eliminar</button>
			Total	1169 €



Eliminado!

Se ha eliminado el item del carrito de compras.

OK

5.2 Manual de instalación para la puesta en marcha de la aplicación

Requisitos Previos

1. Tener instalado el motor de base de datos **MySQL**:
 - Asegúrate de que está configurado en el puerto localhost:3306.
 - Crea la base de datos necesaria para el proyecto: db_backend_users.
2. Contar con el entorno de trabajo **Visual Studio Code** instalado.
3. Tener permisos de administrador para instalar herramientas y dependencias adicionales.

Paso 1: Configuración del Entorno de Angular

1. Descargar e instalar Node.js:

- Descarga la versión estable de **Node.js** desde su [sitio oficial](#).
- Sigue los pasos de instalación proporcionados por el instalador.
- Verifica la instalación ejecutando los siguientes comandos en tu terminal:

```
node -v  
npm -v
```

Estos comandos deben mostrar las versiones instaladas de Node.js y npm.

2. Instalar Angular CLI:

- Usa el siguiente comando para instalar Angular CLI globalmente:

```
npm install -g @angular/cli
```

- Verifica que la instalación fue exitosa ejecutando:

```
ng version
```

Paso 2: Clonar Repositorios y Configurar Dependencias

1. Clonar los repositorios del proyecto desde GitHub:
 - Usa los comandos de git para clonar tanto el repositorio de Angular (parte web) como el de Spring Boot (API RESTful).

```
git clone <URL-repositorio-Angular>  
git clone <URL-repositorio-SpringBoot>
```

2. Instalar dependencias en el proyecto de Angular:

- Navega a la carpeta del proyecto de Angular y ejecuta el siguiente comando:

```
npm install
```

3. Ejecutar el proyecto de Angular:

- Usa el siguiente comando para iniciar el servidor de desarrollo en el puerto predeterminado 4200:

```
ng serve
```

- Una vez que el servidor está corriendo, abre tu navegador y ve a la siguiente URL:

```
http://localhost:4200/login
```

Si todo está configurado correctamente, deberías ver la aplicación web funcionando.

Extensiones Recomendadas para VS Code (Angular):

- **Angular Language Service:** Autocompletado, sugerencias y errores en tiempo real.
- **Angular Snippets:** Atajos para crear componentes, directivas, servicios, entre otros.

Paso 3: Configurar el Entorno de Spring Boot

1. Instalar **Java Development Kit (JDK)**:

- Descarga e instala el JDK desde el [sitio oficial de Oracle](#).
- Recomendado: Versión 21 (la misma utilizada por la API del proyecto).

Verifica la instalación ejecutando: **java -version**

2. Instalar extensiones para VS Code:

- Abre el Marketplace de VS Code (Ctrl+Shift+X) y busca las siguientes extensiones:
 - **Spring Boot Extension Pack:** Incluye soporte para Spring Boot, Spring Initializr y herramientas relacionadas.
 - **Java Extension Pack:** Proporciona soporte completo para Java.

- **Spring Boot Dashboard:** Permite gestionar la API y visualizar los endpoints en tiempo real.

Paso 4: Arrancar la API RESTful de Spring Boot

1. Asegurarse de que el puerto 8080 esté libre:
 - Si el puerto está ocupado, cambia la configuración en el archivo `application.properties` o `pom.xml` del proyecto.
2. Abrir el proyecto en VS Code:
 - Asegúrate de estar en la carpeta del proyecto antes de proceder a la ejecución.
3. Ejecutar la API usando Spring Boot Dashboard:
 - En el panel lateral izquierdo de VS Code, abre la vista **Spring Boot Dashboard**.
 - El dashboard detectará automáticamente las aplicaciones Spring Boot disponibles.
 - Haz clic en el botón **play** (▶) para iniciar la aplicación.
4. Verificar la ejecución:
 - Una vez que la API está corriendo, abre tu navegador y accede a:
<http://localhost:8080/api/users>
5. Interactuar con la API mediante Swagger:
 - Abre la documentación interactiva de la API en Swagger:
<http://localhost:8080/swagger-ui/index.html#/>

Notas Finales

- **Base de Datos:** Asegúrate de que la base de datos MySQL esté configurada y que las credenciales en el archivo `application.properties` coincidan con las de tu entorno.
- **Resolución de Problemas:**
 - Si algo no funciona correctamente, revisa los logs en la consola de VS Code para identificar errores.
 - Verifica que todas las dependencias estén instaladas correctamente.
- **Pruebas Finales:**
 - Realiza pruebas tanto en la aplicación Angular como en los endpoints de la API para asegurarte de que todo funcione como se espera.

6 Conclusiones y posibles ampliaciones

El proyecto surge como respuesta a un análisis del mercado laboral, donde se identificó una creciente demanda de habilidades en tecnologías y frameworks como **Java**, **Spring Boot** y **Angular**. Esta tendencia, sumada a la experiencia adquirida durante mis prácticas profesionales, motivó la idea de crear una aplicación que no solo consolidara mis conocimientos en estas herramientas, sino que también me permitiera enfrentar los desafíos reales del desarrollo de software moderno.

La decisión de emplear **Angular** en el frontend fue un reto significativo, dado que no contaba con una base sólida en esta tecnología. Sin embargo, este desafío se convirtió en una oportunidad para experimentar y superar la **curva de aprendizaje pronunciada** que implica trabajar con Angular, dedicando tiempo y recursos para construir una interfaz funcional y coherente.

El propósito central del proyecto era desarrollar una aplicación **totalmente autónoma**, evitando depender de APIs externas, bases de datos preexistentes o servicios de terceros. Esto no solo me permitió comprender y participar en todas las fases del desarrollo, desde la concepción de la idea hasta su implementación, sino que también me ayudó a valorar el esfuerzo y la planificación necesarios para construir un proyecto independiente.

Además, la conexión directa con mis prácticas fue un factor clave en la decisión de llevar a cabo este proyecto. Esta experiencia combinada sirvió como puente entre mi formación académica y los requisitos del mercado laboral, proporcionándome una visión práctica y aplicable de las tecnologías más demandadas en la industria actual.

En definitiva, el proyecto se plantea como un ejercicio de aprendizaje integral que combina el desarrollo de competencias técnicas, el enfoque en tendencias del mercado y la voluntad de enfrentar desafíos tecnológicos para construir una aplicación sólida y moderna.

Como futuras ampliaciones de mi aplicación, me gustaría implementar la posibilidad de filtrar los productos del catálogo por categorías y rangos de precios. Además, me gustaría

que los productos seleccionados por el cliente desde el catálogo y añadidos al carrito de compra se asocien automáticamente al cliente en la factura generada, la cual se exporta en formato PDF.

7 Bibliografía

Curso de Angular y Spring Boot desarrollado durante las prácticas en la empresa **Hiberus**.

Tutora de prácticas. (s.f.). Ayuda ofrecida durante las prácticas.

OpenAI. (2023). ChatGPT. <https://chatgpt.com>

Stack Overflow. (s.f.). <https://stackoverflow.com/>

Spring Framework Reference Documentation <https://docs.spring.io/spring-framework/reference/index.html>

Spring.io Guides <https://spring.io/guides>

VVAA, *The Complete Guide to Angular*. Nate Murray, Felipe Coury, Ari Lerner, y Carlos Taborda

SESHADRIV, Shyam. *Angular Up & Running*

FREEMAN, Adam, *Pro Angular*

HUSSAIN, Asim, *Angular: From Theory to Practice*

VVAA, *Learning Angular*. Brad Green y Shyam Seshadri

BSAAGHERI, Hashem, *Mastering Angular*.

WILKEN, Jeremy, *Angular in Action*.

Spring Boot <https://dev.to/weder96/spring-boot-everything-you-need-to-know-and-what-nobody-told-you-o4j>

Javatpoint - Spring Boot Tutorial <https://www.javatpoint.com/spring-boot-tutorial>

Documentación Oficial de Angular <https://angular.dev/overview>