

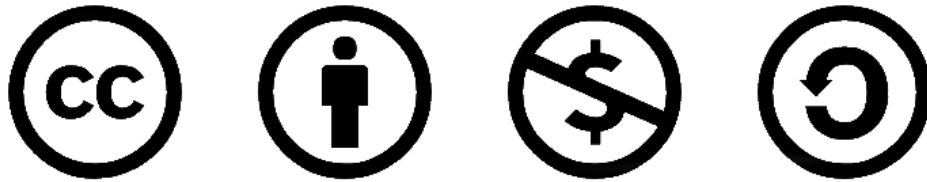


Shopflow

Sheng Ye
2º DAW
14/05/2025

Este trabajo se encuentra bajo la licencia de “**Creative Commons**”

[Attribution-NonCommercial-ShareAlike 4.0 International](https://creativecommons.org/licenses/by-nc-sa/4.0/)



CC BY-NC-SA 4.0

Eres libre de:

Compartir: Copiar y redistribuir el material en cualquier medio o formato

Adaptar: Remezclar, transformar y desarrollar el material

Índice

INTRODUCCIÓN	5
TEMPORIZACIÓN	7
PLANIFICACIÓN DE PROYECTO.....	8
PROBLEMAS ENCONTRADOS.....	9
IMPLEMENTACIÓN.....	11
ANÁLISIS	11
<i>Requerimientos Iniciales Y Actores de Sistema</i>	<i>11</i>
<i>Diagrama de Caso de Uso.....</i>	<i>12</i>
<i>Descripción de Cada Caso de Uso</i>	<i>12</i>
<i>Diagrama de Actividades.....</i>	<i>15</i>
DISEÑO.....	16
<i>Estructura de paquetes organizados por capas y Detalles</i>	<i>16</i>
<i>Diagrama de clases.....</i>	<i>33</i>
<i>Diagramas de secuencia de cada caso de uso</i>	<i>33</i>
<i>Diagrama de tablas</i>	<i>35</i>
MANUAL DE USUARIO.....	36
HERRAMIENTAS Y TECNOLOGÍAS EMPLEADAS.....	45
PROPUESTA DE MEJORA	50
VALORACIÓN PERSONAL	52
PUNTOS A DESCARTAR	54
BIBLIOGRAFÍA	56

Introducción

Para los dueños de pequeñas tiendas, tener el control del inventario, anotar las ventas y administrar cosas puede ser una tarea complicada. Muchas herramientas en el mercado son para grandes empresas, o tienen funciones de más, haciendo el proceso aún más complejo y menos intuitivo.

Por eso les presento Shopflow una aplicación fácil de usar y útil hecha para pequeñas tiendas que quieren manejar sus productos, compras y ventas rápido.

Con ShopFlow, los comerciantes pueden registrar cosas rápido, llevar un seguimiento claro de las ventas y ver datos al instante para tomar mejores decisiones. Todo esto se hace de manera fácil y perfecto para los que quieren mejorar su negocio.

Objetivos

El objetivo de este proyecto es aprender nuevos conocimientos y herramientas, así como profundizar en lo aprendido en clase. Se busca fortalecer el aprendizaje de forma autodidacta y ampliar la comprensión de los contenidos trabajados.

¿Por qué decidí hacer este proyecto?

Decidí desarrollar este proyecto porque quiero mejorar la gestión de nuestro bazar, especialmente en la organización de precios y el control de inventario. Los principales motivos son:

- **Organización de precios:** Necesitamos tener claro cual es el precio de los productos para poder gestionar la subida o bajada de precio.
- **Facilidad en la reposición de productos:** No tenemos claro muchas veces cuanto tenemos de cada producto y también queremos saber que cosas se están vendiendo mejor o peor.
- **Limitaciones de otras aplicaciones:** Las apps tienen varios problemas, como:
 - Muchos de anuncios que interrumpen la experiencia.
 - Interfaces con demasiados botones, lo que las hace complicadas de usar.
 - Modelos de pago que no se ajustan a nuestras necesidades.

Por ende, decidí crear mi versión de aplicación de gestión de inventario para que se adapte mejor a mis necesidades, me permita evitar el pago por el uso de otras aplicaciones y, además, poner en práctica lo aprendido.

Temporización

	Marzo																														Abril					
	Semana 1						Semana 2						Semana 3						Semana 4																	
		11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3	4	5	6								
Investigación Front																																				
Planificación de Proyecto																																				
Diseñar el Logo																																				
Diagrama de Casos de Uso																																				
Diagrama de Clases																																				
Diagrama de Tablas																																				
Redacción memoria																																				
Repaso de general de java																																				
Crear Base de Dato con MariaDB																																				
	Abril																														Mayo					
	Semana 5						Semana 6						Semana 7						Semana 8																	
	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4								
Desarrollo Crud de productos																																				
Desarrollo Crud de venta																																				
Desarrollo de login																																				
Redacción memoria																																				
	Mayo																																			
	Semana 9						Semana 10																													
	5	6	7	8	9	10	11	12	13	14	15	16	17	18																						
Desarrollo front-end																																				
Redacción memoria																																				

Planificación de Proyecto

La planificación lo he dividido en 3 parte principales la planificación (con lleva los diagramas e investigación), backend (la lógica de mi app) y el frontend (Lo que el usuario va tener acceso)

- Planificación (2 semanas)
 1. Investigación de que herramientas a usar y como usar
 2. Diseño de Caso de uso
 3. Diseño de Clases
 4. Diseño de Tablas
- BackEnd (4 semanas)
 1. Creación de la Base de Datos
 2. Creación de Spring boot con maven
 3. Creación de Crud Productos
 4. Creación de Crud Ventas
 5. Sistema de inicio de sesión
 6. Conexión con el Front
- FrontEnd (3 semanas)
 1. Creación base Angular
 2. Añadir Angular Material y Tailwind
 3. Creación de Componentes
 4. Conexión entre componentes
 5. Creación de guards para el login
 6. Crear app escritorio con electron

Problemas Encontrados

1. Lista de Detalle en ventas

En el código de mi app tengo en ventas una lista de detalle en ventas, el problema está en que ventas con jpa no tiene una conexión directa con detalles en la base de datos (no tiene un pk).

La solución lo encontré en la página web de [baeldung](#) que hay una opción en jpa “mapperBy” que te permite mapear sin necesidad de una conexión directa en la base de datos.

2. Método Lista detalle

Al planificar los métodos que iba a usar añadí un método, a lo largo del proyecto me di cuenta de que no era necesario poner un método este método en concreto. Solución: no poner el método.

3. Desarrollo de login

A la hora de desarrollar el login se me ocurrió hacerlo con Token, en el filtro (JwtFilter) tenía el problema de que no se pasaba (no detectaba) correctamente el rol del usuario la solución lo encontré con chatgpt.

El segundo problema fue en la autenticación, que la versión de JWT que usaba era más nueva que en los tutoriales que estaba viendo. La solución lo encontré en [StackOverflow](#), el parseClaimJws necesitaba estar builder primero, que en versiones antiguas no necesitaba.

Implementación

Análisis

Requerimientos Iniciales Y Actores de Sistema

Todos los usuarios (Usuarios)

- Sistema de login.
- Validar las credenciales del login
- Todos pueden ver todos los productos
- Buscar producto por código de barras.
- Comprobar que el producto existe si no saltar un error

Usuarios Registrados (Clientes)

- Añadir productos
- Borrar productos
- Editar productos
- Añadir una nueva venta con la lista de productos
- Ver las Ventas
- Seleccionar una venta y ver todos los productos detallados (cantidad y subtotal)
- Borrar una venta

Otros

- Hacer Pruebas unitarias
- Hacer el Fron End
- Crear app de escritorio con electron.

Diagrama de Caso de Uso

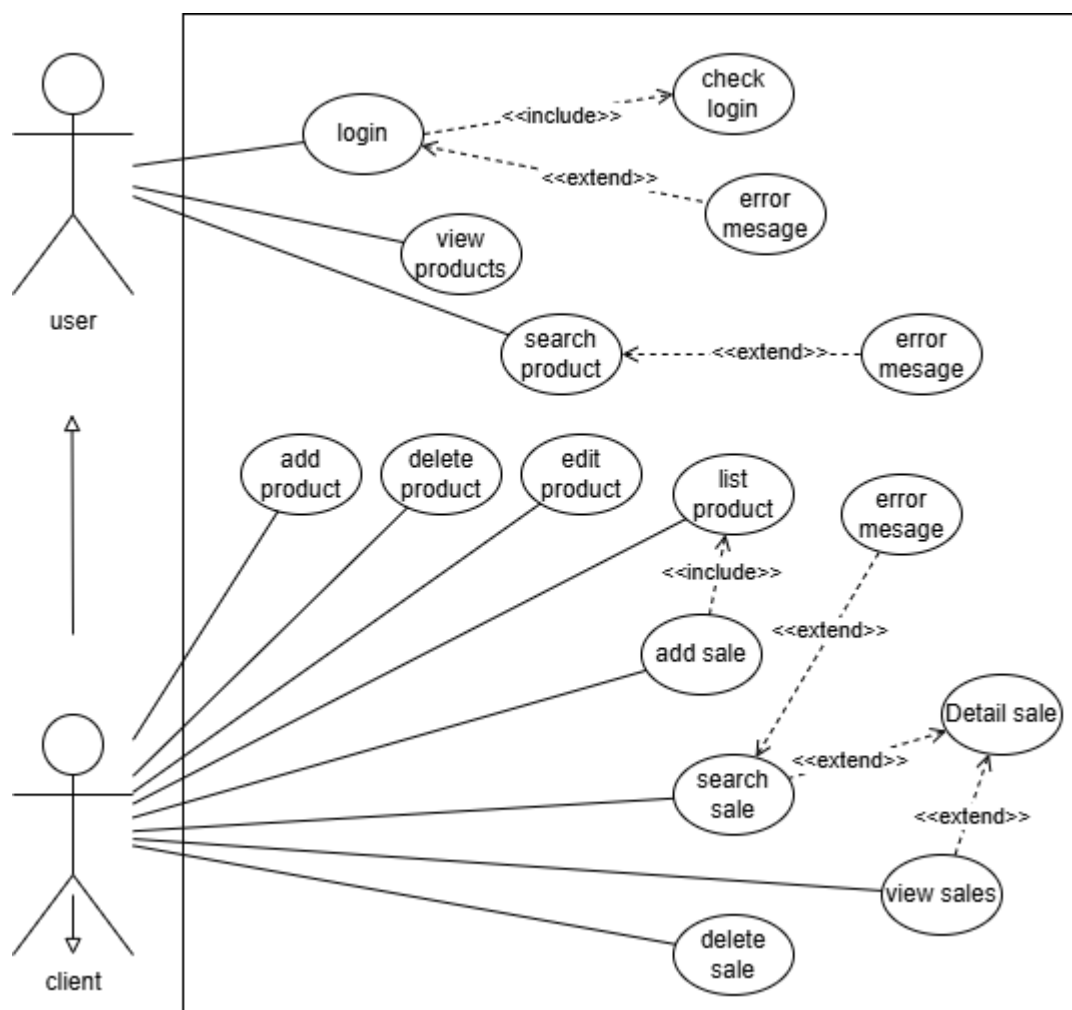


Diagrama de Caso de Uso con sus actores y metodos

Descripción de Cada Caso de Uso

Nombre	Login
Descripción	El usuario inicia sesión en el sistema con un usuario y contraseña.
Actor	Usuario no identificado (Usuarios)
Precondiciones	El usuario no debe estar identificado, pero sí registrado
Flujo Principal	1. El usuario pone su nombre de usuario y contraseña 2. El sistema valida los datos introducidos 3. El sistema identifica al usuario con su rol.
Flujo Alternativo	Devuelve una excepción

Nombre	View Product
Descripción	Muestra todos los productos
Actor	Todos los usuarios
Precondiciones	
Flujo Principal	El sistema devuelve el listado de todos los productos
Flujo Alternativo	Devuelve una excepción
Postcondición	El usuario iniciado de sesión podrá borrar o modificar el producto

Nombre	Search Product
Descripción	El sistema manda el producto en concreto por el código de barras
Actor	Todos los usuarios
Precondiciones	
Flujo Principal	<ol style="list-style-type: none"> 1. El usuario ingresa el código de barras 2. El sistema muestra los productos que coinciden
Flujo Alternativo	Devuelve una excepción
Postcondición	El usuario iniciado de sesión podrá borrar o modificar el producto

Nombre	Add Product
Descripción	Añadir un producto nuevo
Actor	Cliente
Precondiciones	El usuario debe haber iniciado sesión
Flujo Principal	<ol style="list-style-type: none"> 1. El usuario completa el formulario de nuevo producto. 2. El sistema valida y guarda el producto
Flujo Alternativo	Devuelve una excepción

Nombre	Edit Product
Descripción	El usuario modifica los detalles de un producto.
Actor	Cliente
Precondiciones	El usuario debe haber iniciado sesión
Flujo Principal	<ol style="list-style-type: none"> 1. El usuario selecciona un producto 2. Modifica la información 3. El sistema valida y guarda el producto
Flujo Alternativo	Devuelve una excepción

Nombre	Delete Product
Descripción	Elimina un producto existente.
Actor	Cliente
Precondiciones	El usuario debe haber iniciado sesión
Flujo Principal	<ol style="list-style-type: none"> 1. El usuario selecciona un producto 2. El sistema elimina el producto seleccionado
Flujo Alternativo	Devuelve una excepción

Nombre	List Product
Descripción	El usuario escoge una lista de productos que tendrá también la cantidad y un sub total.
Actor	Cliente
Precondiciones	El usuario debe haber iniciado sesión
Flujo Principal	<ol style="list-style-type: none"> 1. El usuario escoge una lista de productos 2. El sistema comprueba si hay stonk 3. Modifica el stonk de los productos 4. Valida los productos y guarda la lista detallada de los productos
Flujo Alternativo	Devuelve una excepción
Postcondición	El usuario iniciado de sesión podrá borrar o modificar los productos detallados

Nombre	Add Sale
Descripción	El usuario escoge una lista de productos detallados y guarda la venta
Actor	Cliente
Precondiciones	El usuario debe haber iniciado sesión y haber seleccionado una lista
Flujo Principal	<ol style="list-style-type: none"> 1. El usuario Selecciona los productos detallados 2. El sistema valida los datos introducidos 3. El sistema calcula el total y guarda
Flujo Alternativo	Devuelve una excepción

Nombre	Search Sale
Descripción	El usuario busca una venta registrada.
Actor	Cliente
Precondiciones	El usuario debe haber iniciado sesión
Flujo Principal	<ol style="list-style-type: none"> 1. El usuario ingresa la id 2. El sistema muestra las ventas que coinciden
Flujo Alternativo	Devuelve una excepción
Postcondición	El usuario iniciado de sesión podrá borrar o ver los productos detallados

Nombre	View Sale
Descripción	El usuario visualiza las ventas realizadas.
Actor	Cliente
Precondiciones	El usuario debe haber iniciado sesión
Flujo Principal	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de ver ventas 2. El sistema muestra todas las ventas
Flujo Alternativo	Devuelve una excepción
Postcondición	El usuario iniciado de sesión podrá borrar o ver los productos detallados

Nombre	Delete Sale
Descripción	El usuario elimina una venta registrada.
Actor	Cliente
Precondiciones	El usuario debe haber iniciado sesión
Flujo Principal	<ol style="list-style-type: none"> 1. El usuario selecciona una venta 2. El sistema elimina la venta
Flujo Alternativo	Devuelve una excepción

Diagrama de Actividades

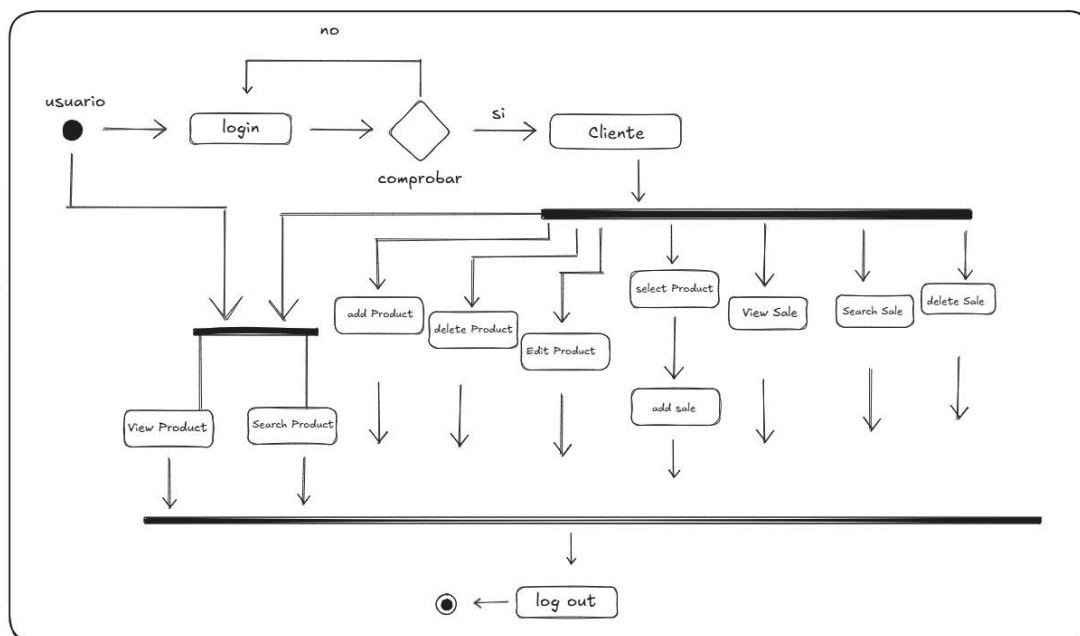


Diagrama de Actividades

El usuario puede iniciar sesión con usuario y contraseña, el sistema revisará que las claves (usuario y contraseña) sean correctas, si no le lleva de vuelta al inicio de sesión. Una vez iniciado de sesión el **cliente** puede hacer diversas operaciones que se podrían dividir en dos grupos: los de *producto* y *ventas*.

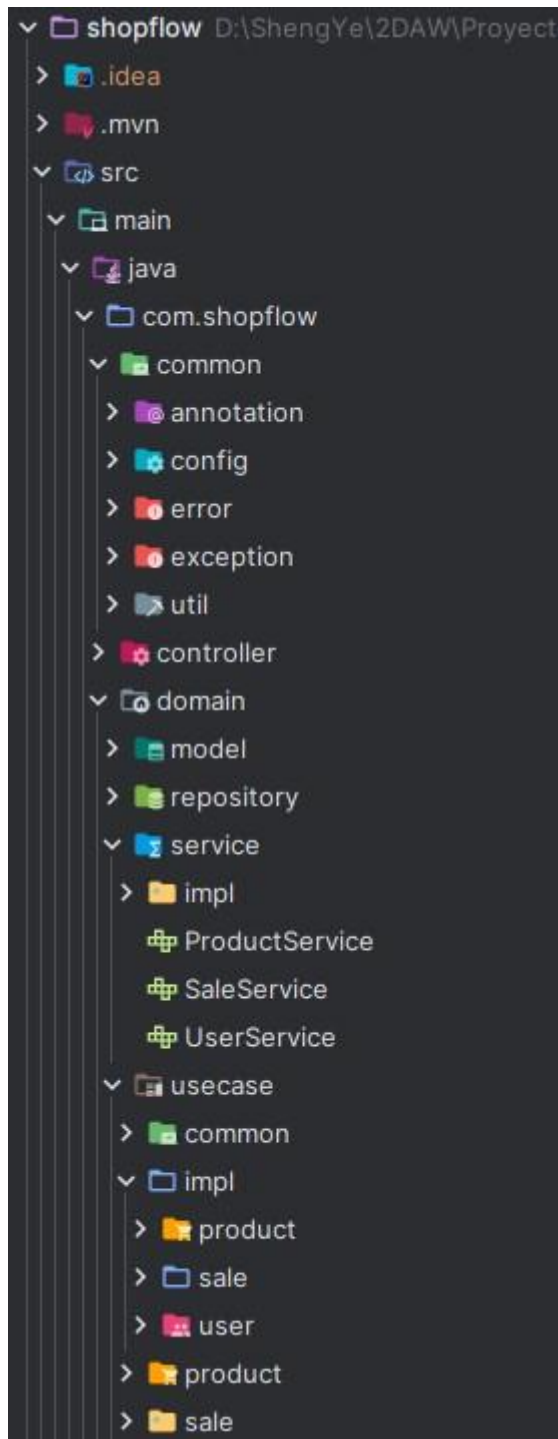
En productos hay dos métodos que funcionan, aunque no hayas iniciado sesión que sería siendo: Ver todos los productos y Buscar un producto.

El resto de métodos solo los puede usar el cliente como en la parte de producto: añadir, editar y borrar. Por parte de la venta cuando quieres registrar/añadir una venta tendrás que tener una lista detallada (aparte de producto tener cantidad y sub total) luego podrás añadir la venta.

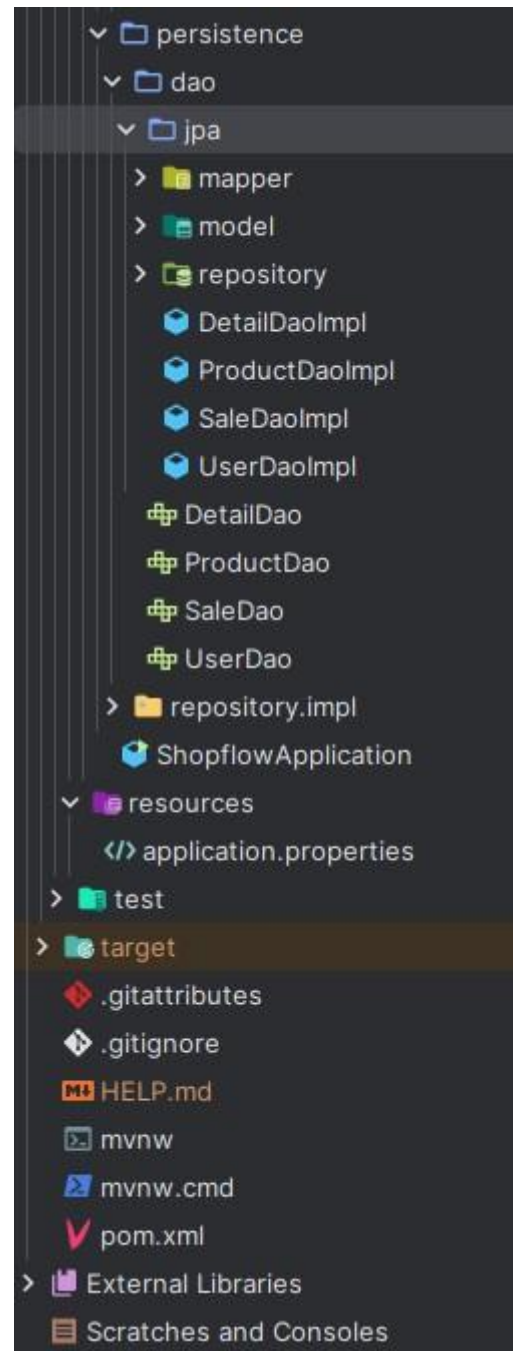
En cualquier momento el usuario registrado (cliente) se puede cerrar la sesión.

Diseño

Estructura de paquetes organizados por capas y Detalles



Carpets de spring 1



Carpets de spring 2

La estructura de capas que he utilizado para mi app es la que hemos estado practicando en clase (La **arquitectura por capas**) que su idea principal es dividir la aplicación en capas independientes, cada una con una responsabilidad específica.

src/: Carpeta principal donde está todo el código fuente de nuestra aplicación.

pom.xml: En él se define toda la información del proyecto, incluyendo sus dependencias, módulos, plugins y las configuraciones necesarias para compilar, probar y empaquetar la aplicación.

java/: Aquí van las clases, interfaces, servicios, controladores, etc., que forma la lógica de nuestra app

resource/: Aquí van los archivos de recurso que se usa junto con el código fuente.

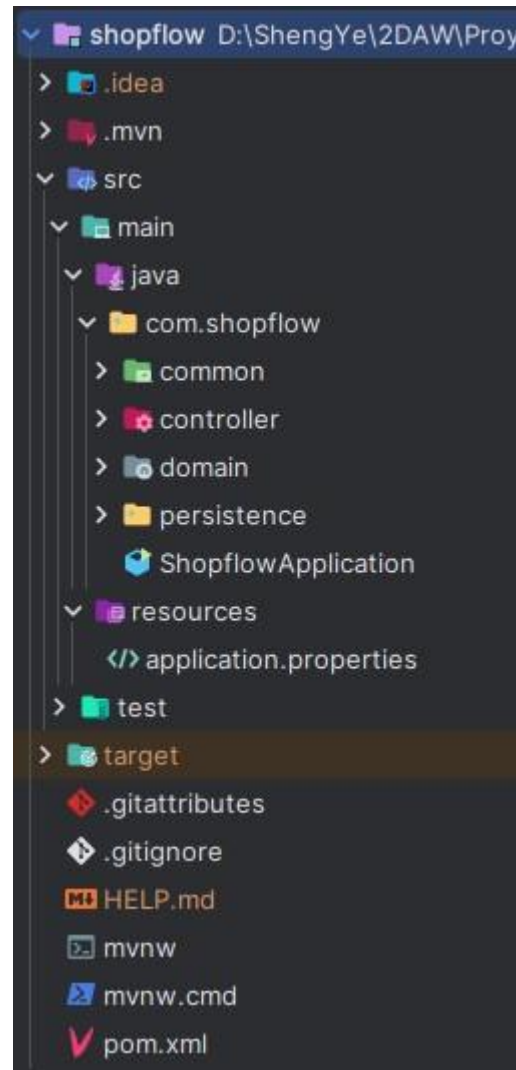
application.properties: este es un archivo de configuración donde contiene algunas de las configuraciones que va necesitar a lo largo del proyecto como la conexión a la base de datos.

common/: En esta carpeta contiene código que se puede reutilizar en todo el proyecto.

controller/: En esta carpeta se gestiona las solicitudes o peticiones que viene del usuario y devolviendo respuesta del sistema

domain/: Es la carpeta donde está la lógica de negocio, entidades, etc.

persistence/: Esta carpeta es la encargada de almacenar y recuperar datos de una base de datos o servicio, o ambas, dependiendo de lo que este buscando el programador. Nosotros solo a la base de datos



Carpeta principal de las capas

```
spring.application.name=shopflow

spring.datasource.url=jdbc:mariadb://localhost:3307/shop
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

app.base.url=http://localhost:8080
app.pageSize.default=10
```

configuración application.properties

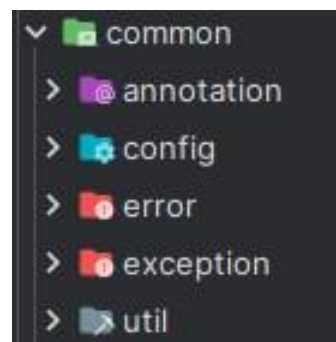
annotation/: aquí es donde se ponen las anotaciones personalizadas.

config/: en esta carpeta agrupa todas las clases relacionadas con la configuración general del sistema

error/: en esta carpeta se encarga de gestionar los errores y excepciones que aparecen en nuestro proyecto

exception/: contiene las excepciones personalizadas que representan errores específicos del dominio de la aplicación.

util/: contiene clases que se puede reutilizables que proporcionan funcionalidades de apoyo a diferentes partes del sistema.



Carpetas de Common

Util: JwtUtil

```
@Component 4 usages Shengye
public class JwtUtil {

    private static final SecretKey SECRET = Keys.secretKeyFor(SignatureAlgorithm.HS256); 2 usages

    public String generateToken(UserDetails user){ 1 usage Shengye
        return Jwts.builder()
            .setSubject(user.getUsername())
            .claim("rol", user.getAuthorities())
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60)) // 1 hora
            .signWith(SignatureAlgorithm.HS256, SECRET)
            .compact();
    }

    public Claims obtainClaims(String token) { 3 usages Shengye
        try {
            return Jwts.parser() JwtParserBuilder
                .setSigningKey(SECRET)
                .build() JwtParser
                .parseClaimsJws(token) Jws<Claims>
                .getBody();
        } catch (Exception e) {
            return null;
        }
    }

    public boolean validateToken(String token, String username) { 1 usage Shengye
        final String extractedUsername = extractUsername(token);
        if (extractedUsername == null) {
            return false;
        }
        return (extractedUsername.equals(username) && !isTokenExpired(token));
    }
}
```

JwtUtil 1

```
public String extractUsername( String token) { 2 usages  Shengye
    try {
        return obtainClaims(token).getSubject();
    } catch (Exception e) {
        return null;
    }
}

public List<String> extractRoles(String token) { 1 usage  Shengye
    List<Map<String, Object>> roles = obtainClaims(token).get(s: "rol", List.class);
    return roles.stream() Stream<Map<...>>
        .map( Map<String, Object> role → (String) role.get("authority")) Stream<String>
        .collect(Collectors.toList());
}
}
```

JwtUtil 2

En esta clase se encuentran las utilidades para gestionar operaciones relacionadas al token JWT por ejemplo crear, comprobar, sacar datos, etc. del token.

Config JwtFilter

```
@Component 1 usage  Shengye
public class JwtFilter extends OncePerRequestFilter {

    @Autowired 3 usages
    private JwtUtil jwtUtil;

    @Override no usages  Shengye
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
        final String authHeader = request.getHeader(s: "Authorization");

        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }

        final String token = authHeader.substring(beginIndex: 7);

        if (SecurityContextHolder.getContext().getAuthentication() == null) {
            String username = jwtUtil.extractUsername(token);
            if (jwtUtil.validateToken(token, username)) {
                List<String> role = jwtUtil.extractRoles(token);

                List<GrantedAuthority> authorities = role.stream() Stream<String>
                    .map(SimpleGrantedAuthority::new) Stream<SimpleGrantedAuthority>
                    .collect(Collectors.toList());

                UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                    username,
                    credentials: null,
                    authorities
                );

                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            } else {
                response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
                response.getWriter().write(s: "Token expirado o inválido");
                return;
            }
        }

        filterChain.doFilter(request, response);
    }
}
```

JwtFilter

Esta clase es el filtro de seguridad de la app, que afecta a los endpoints configurados en el securityConfig.

Config SecurityConfig

```
@Configuration no usages Shengye
@EnableWebSecurity
public class SecurityConfig {

    @Autowired 1 usage
    private JwtFilter jwtFilter;

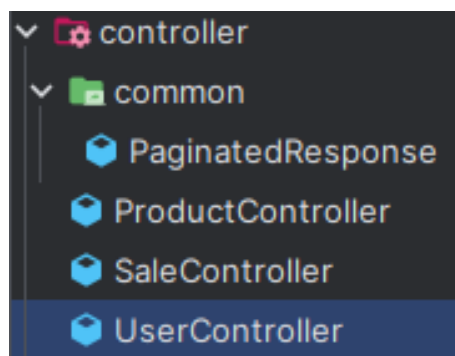
    @Bean no usages Shengye
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(...patterns: "/api/auth/**").permitAll()
                .requestMatchers(
                    ...patterns: "/v3/api-docs/**",
                    "/swagger-ui/**",
                    "/swagger-ui.html"
                ).permitAll()
                .requestMatchers(HttpMethod.GET, ...patterns: "/api/products/**").permitAll()
                .requestMatchers(HttpMethod.POST, ...patterns: "/api/products").hasRole("CLIENT")
                .requestMatchers(HttpMethod.PUT, ...patterns: "/api/products/**").hasRole("CLIENT")
                .requestMatchers(HttpMethod.DELETE, ...patterns: "/api/products/**").hasRole("CLIENT")
                .requestMatchers(HttpMethod.GET, ...patterns: "/api/sales/**").hasRole("CLIENT")
                .anyRequest().authenticated()
            )
            .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
}
```

SecurityConfig

Esta clase se encarga definir como se tiene que proteger la aplicación (Pertenece a spring security).

controller/: contiene las clases que manejan las solicitudes HTTP. Se encargan de recibir peticiones del cliente, llamar a los servicios correspondientes y devolver respuestas.

common/: en esta carpeta se guarda una utilidad que puede usar todos los controladores.



Carpeta Controller

UserController

```
@RestController 1 usage Shengye
@RequestMapping(UserController.URL)
@CrossOrigin(origins = "http://localhost:4200")
public class UserController {

    public static final String URL = "/api/auth"; 1 usage

    @Autowired 1 usage
    private UserLoadUseCase userLoadUseCase;

    @Autowired 1 usage
    private JwtUtil jwtUtil;

    @PostMapping("/login") no usages Shengye
    public ResponseEntity<> login(@RequestBody Users loginRequest){
        UserDetails user = userLoadUseCase.loadUserByUsername(loginRequest.getUsername());
        if (user == null || !user.getPassword().equals(loginRequest.getPassword())) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Credenciales inválidas");
        }
        String token = jwtUtil.generateToken(user);
        return ResponseEntity.ok(Collections.singletonMap("token", token));
    }
}
```

UserController

En este controlador lo que hace es cuando recibe una petición post por la ruta `/api/auth/login`⁽¹⁾ recibe los datos de user⁽²⁾ (usuario y contraseña) de parte del cliente. Manda a buscar los datos del usuario a los casos de uso, cuando lo recibe comprueba que esta el usuario y coincide la contraseña insertada con la de la base de datos. Si todo está bien devuelve una respuesta con un token con los datos del usuario, si la contraseña no es correcta o no existe el usuario devuelve una respuesta “Credenciales invalidas”.

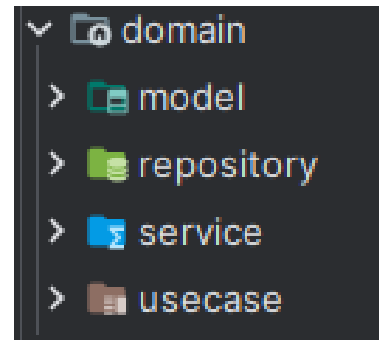
- **@CrossOrigin()**: indica las direcciones desde las que puede recibir peticiones, con esto lo podemos conectar con el front-end.
- (1) **@RequestMapping()**: indica que para llamar a este controlador tiene que empezar por la ruta que tenga dentro del paréntesis, el **@PostMapping()**: indica que solo atiende a los que llamen por post y tengan ruta principal mas la ruta que hay entre paréntesis del post (pasa lo mismo cuando quieres hacer Get, Put, Delete, etc.).
- (2) Sabemos los datos que vamos a recibir gracias al **@RequestBody** dentro del método.

model/: Dentro de esta carpeta se guarda los datos del negocio de nuestra aplicación.

repository/: En esta carpeta se encuentra solo las interfaces del repositorio.

service/: En esta carpeta es donde se implementan reglas de negocio, llamadas a repositorios, validaciones, etc.

usecase/: contiene clases que representan una acción o proceso de negocio concreto.



Carpeta Domain

Model Detail

```
public class Detail { 20 usages  🧑 Shengye
    private int quantity; 2 usages
    private BigDecimal subtotal; 2 usages
    private Product products; 2 usages

    public int getQuantity() { 4 usages  🧑 Shengye
        return quantity;
    }

    public void setQuantity(int quantity) { 2 usages  🧑 Shengye
        this.quantity = quantity;
    }

    public BigDecimal getSubtotal() { 1 usage  🧑 Shengye
        return subtotal;
    }

    public void setSubtotal(BigDecimal subtotal) { 2 usages  🧑 Shengye
        this.subtotal = subtotal;
    }

    public Product getProducts() { 3 usages  🧑 Shengye
        return products;
    }

    public void setProducts(Product products) { 2 usages  🧑 Shengye
        this.products = products;
    }

    public BigDecimal calculateSubtotal(int quantity, BigDecimal price) {
        return price.multiply(new BigDecimal(quantity));
    }
}
```

Detail

Model Product

```
public class Product {  🐞 Shengye
    private Integer id;  2 usages
    private String barCode;  2 usages
    private String name;  2 usages
    private BigDecimal price_sell;  2 usages
    private BigDecimal price_buy;  2 usages
    private int stock;  2 usages

    public Integer getId() {  🐞 Shengye
        return id;
    }

    public void setId(Integer id) {  🐞 Shengye
        this.id = id;
    }

    public String getBarCode() { return barCode; }

    public void setBarCode(String barCode) { this.barCode = barCode; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public BigDecimal getPrice_sell() { return price_sell; }

    public void setPrice_sell(BigDecimal price_sell) { this.price_sell = price_sell; }

    public BigDecimal getPrice_buy() { return price_buy; }

    public void setPrice_buy(BigDecimal price_buy) { this.price_buy = price_buy; }

    public int getStock() { return stock; }

    public void setStock(int stock) { this.stock = stock; }
}
```

Product

Model Sale

```
public class Sale {  🧑 Shengye
    private Integer id;  2 usages
    private LocalDateTime date;  2 usages
    private BigDecimal total;  2 usages
    private List<Detail> details;  2 usages

    public Integer getId() { return id; }

    💡 public void setId(Integer id) { this.id = id; }

    public LocalDateTime getDate() { return date; }

    public void setDate(LocalDateTime date) { this.date = date; }

    public BigDecimal getTotal() { return total; }

    public void setTotal(BigDecimal total) { this.total = total; }

    public List<Detail> getDetails() { return details; }

    public void setDetails(List<Detail> details) { this.details = details; }

    public BigDecimal calculateTotal(List<BigDecimal> subtotalList) {  1 usage
        BigDecimal total = BigDecimal.ZERO;
        for (BigDecimal subtotal : subtotalList) {
            total = total.add(subtotal);
        }
        return total;
    }
}
```

Sale

Model Users

```
public class Users { 21 usages  👤 Shengye
    private String username; 4 usages
    private String password; 4 usages
    private String rol; 3 usages

    public Users() { 1 usage  👤 Shengye
    }

    public Users(String username, String password, String rol) { no usages  👤
        this.username = username;
        this.password = password;
        this.rol = rol;
    }

    public Users(String username, String password) { no usages  👤 Shengye
        this.username = username;
        this.password = password;
    }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }

    public String getRol() { return rol; }

    public void setRol(String rol) { this.rol = rol; }
}
```

Users

UseCase UserLoadUseCase

```
@DomainUseCase 2 usages 🧑 Shengye
@DomainTransactional
public class UserLoadUseCase implements UserDetailsService {

    @Autowired 1 usage
    private UserService userService;

    @Override 1 usage 🧑 Shengye
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Users user = userService.findByUserName(username)
            .orElseThrow(() -> new UsernameNotFoundException("Usuario no encontrado"));
        return new User(user.getUsername(), user.getPassword(), getAuthorities(user));
    }

    private Collection<? extends GrantedAuthority> getAuthorities(Users usuario) { 1 usage 🧑 Shengye
        return List.of(new SimpleGrantedAuthority("ROLE_" + usuario.getRol()));
    }
}
```

UserLoadUseCase

Los casos de uso reciben los datos del controlador, hace la lógica y se lo mandan al servicio. Si tiene datos se lo devuelve al controlador.

Service UserService

```
@DomainService no usages 🧑 Shengye
public class UserServiceImpl implements UserService {

    @Autowired 1 usage
    private UserRepository userRepository;

    @Override 1 usage 🧑 Shengye
    public Optional<Users> findByUserName(String username) {
        return userRepository.findByUserName(username);
    }
}
```

UserService

El servicio recibe los datos del caso de uso y lo envía al repositorio. Si tiene datos se lo devuelve al servicio

dao/: En esta carpeta sus clases se encarga de conectar con la base de datos.

repository/: Esta carpeta se encarga de dirigir los datos diferentes conexiones a base de datos, servicios o cache. (en nuestro caso solo a la base de datos)

Repository UserRepository

```
@Repository no usages Shengye
public class UserRepositoryImpl implements UserRepository {

    @Autowired 1 usage
    private UserDao userDao;

    @Override 1 usage Shengye
    public Optional<Users> findByName(String username) { return userDao.findByName(username); }
```

UserRepository

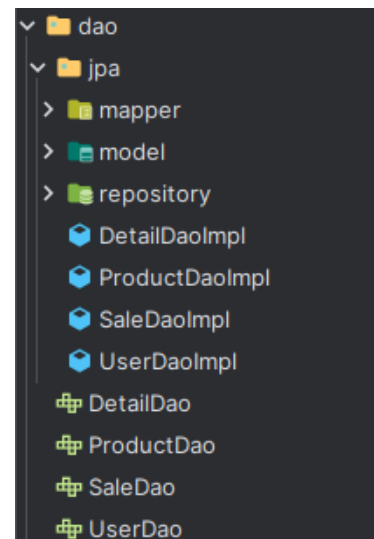
Recibe los datos del servicio y se lo manda a la conexión correspondiente, en este caso dao.

jpa/: Esta carpeta tiene las clases con conexión a la base de datos con JPA.

mapper/: En esta carpeta esta los mapeadores que transforma las entidades de la base de datos a los de la aplicación

model/: En esta carpeta esta las entidades de la base de datos.

repository/: En esta carpeta esta las interfaces de jpa repository que nos permite acceder a la base de datos sin tener que escribir SQL.



Carpeta Dao

Jpa UserDao

```
@Component no usages Shengye
public class UserDaoImpl implements UserDao {

    @Autowired 1 usage
    private UserJpaRepository userJpaRepository;

    @Override 1 usage Shengye
    public Optional<Users> findByUsername(String username) {
        return userJpaRepository
            .findByUsername(username)
            .map(UserJpaMapper.INSTANCE::toUser);
    }
}
```

UserDao

En el Dao si estas buscando datos (un find) harás el mapeo después de hacer la consulta, pero si estas insertando (save) haces el mapeo antes de la consulta.

Mapper UserJpaMapper

```
public class UserJpaMapper { 4 usages Shengye
    public static UserJpaMapper INSTANCE = new UserJpaMapper(); 1 usage

    public Users toUser(UserEntity userEntity) { 1 usage Shengye
        Users user = new Users();
        user.setUsername(userEntity.getUsername());
        user.setPassword(userEntity.getPassword());
        user.setRol(getRol(userEntity.getRol()));
        return user;
    }

    private String getRol(Integer rol) { 1 usage Shengye
        switch (rol) {
            case 1:
                return "CLIENT";
            default:
                return "UNKNOWN";
        }
    }
}
```

UserJpaMapper

El mapeador transforma los datos de la aplicación a la de la BBDD y viceversa.

JpaModel DetailEntity

```
@Entity 14 usages Shengye
@Table(name = "detail_sale")
public class DetailEntity {
    @Id 2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private int quantity; 2 usages
    private BigDecimal subtotal; 2 usages
    @ManyToOne(fetch = FetchType.LAZY) 2 usages
    @JoinColumn(name = "id_product")
    private ProductEntity product;
    @ManyToOne(fetch = FetchType.LAZY) 2 usages
    @JoinColumn(name = "id_sale")
    private SaleEntity sale;

    public Integer getId() { return id; }

    public void setId(Integer id) { this.id = id; }

    public int getQuantity() { return quantity; }

    public void setQuantity(int quantity) { this.quantity = quantity; }

    public BigDecimal getSubtotal() { return subtotal; }

    public void setSubtotal(BigDecimal subtotal) { this.subtotal = subtotal; }

    public ProductEntity getProduct() { return product; }

    public void setProduct(ProductEntity product) { Shengye
        this.product = product;
    }

    public SaleEntity getSale() { return sale; }

    public void setSale(SaleEntity sale) { this.sale = sale; }
}
```

DetailEntity

- **@Entity**: Indica que los datos lo va tratar JPA.
- **@Table**: En este se pone el nombre de la tabla.
- **@Id y @GeneratedValue()**: indica clave primaria y la estrategia de generación.
- **@JoinColumn**: indica la conexión con otra tabla de la base de datos.
- **@ManyToOne()**: Indica el tipo de conexión.

JpaModel ProductEntity

```
@Entity 11 usages Shengye
@Table(name = "products")
public class ProductEntity {
    @Id 2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(name = "bar_code") 2 usages
    private String barCode;
    private String name; 2 usages
    private BigDecimal price_sell; 2 usages
    private BigDecimal price_buy; 2 usages
    private int stock; 2 usages

    public Integer getId() { return id; }

    public void setId(Integer id) { this.id = id; }

    public String getBarCode() { return barCode; }

    public void setBarCode(String barCode) { this.barCode = barCode; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public BigDecimal getPrice_sell() { return price_sell; }

    public void setPrice_sell(BigDecimal price_sell) { this.price_sell = price_sell; }

    public BigDecimal getPrice_buy() { return price_buy; }

    public void setPrice_buy(BigDecimal price_buy) { this.price_buy = price_buy; }

    public int getStock() { return stock; }

    public void setStock(int stock) { this.stock = stock; }
}
```

ProductEntity

- **@Column:** Indica el nombre de la columna de la base de datos, se pone cuando el nombre no coincide.

JpaModel SaleEntity

```
@Entity 16 usages  Shengye
@Table(name = "sales")
public class SaleEntity {
    @Id 2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private LocalDateTime date; 2 usages
    private BigDecimal total; 2 usages
    @OneToMany(mappedBy = "sale", cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.LAZY)
    private List<DetailEntity> details;

    public Integer getId() { return id; }

    public void setId(Integer id) { this.id = id; }

    public LocalDateTime getDate() { return date; }

    public void setDate(LocalDateTime date) { this.date = date; }

    public BigDecimal getTotal() { return total; }

    public void setTotal(BigDecimal total) { this.total = total; }

    public List<DetailEntity> getDetails() { return details; }

    public void setDetails(List<DetailEntity> details) { this.details = details; }
}
```

SaleEntity

- **@OneToMany()**: Indica otro tipo de conexión con la base de datos, el atributo **mappedBy** indica que se papea en otra tabla y le indicas la tabla, el atributo **cascade** indica que los cambios que le haga al padre también le afecta a el y **fetch** indica cuando se carga los hijos.

JpaModel UserEntity

```
@Entity 5 usages 🧑 Shengye
@Table(name = "users")
public class UserEntity {
    @Id no usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String username; 2 usages
    private String password; 2 usages
    private Integer rol; 2 usages

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return password; }

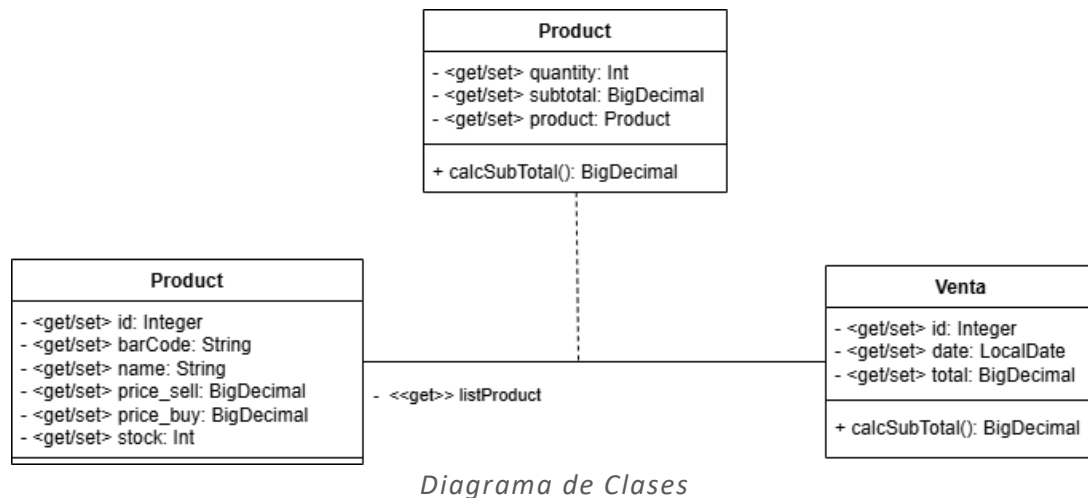
    public void setPassword(String password) { this.password = password; }

    public Integer getRol() { return rol; }

    public void setRol(Integer rol) { this.rol = rol; }
}
```

UserEntity

Diagrama de clases



Diagramas de secuencia de cada caso de uso

- Inicio de sesión
 1. Ingresar Usuario y contraseña.
 2. Verificar credenciales.
 - a. Credenciales correcto, Acceso concedido.
 - b. Credenciales incorrecto, denegar acceso y mostrar error.
 3. Fin
- Ver Producto
 1. Solicitar lista de Producto
 2. Consultar productos de la base de datos.
 3. Mostrar los productos al Usuario
 4. Fin
- Buscar Producto
 1. Ingresar código de barras.
 2. Consulta la base de datos.
 - a. Mostrar datos.
 - b. Mostrar mensaje de error.
 3. Fin
- Borrar Producto
 1. Comprobar si ha iniciado sesión.
 2. Seleccionar producto a eliminar
 3. Comprobar que existe el producto, si no mensaje de error.
 4. Confirmar eliminación.
 5. Eliminar Producto
 6. Fin

- Editar Producto
 1. Comprobar si ha iniciado sesión.
 2. Seleccionar producto.
 3. Modificar los datos.
 4. Validar los datos.
 - a. Guardar los datos modificados.
 - b. Mostrar mensaje de error.
 5. Fin
- Añadir Producto
 1. Comprobar si ha iniciado sesión.
 2. Ingresar los datos de producto.
 3. Validar datos.
 - a. Guardar datos.
 - b. Mensaje de error.
 4. Fin.
- Lista de Productos
 1. Comprobar si ha iniciado sesión.
 2. Buscar producto
 3. Cargar sub datos
 4. Mostrar lista
 5. Fin
- Añadir Venta
 1. Comprobar si ha iniciado sesión.
 2. Lista de Productos.
 3. Validar información.
 - a. Guardar nueva venta
 - b. Mostrar mensaje de error
 4. Fin
- Ver Venta
 1. Comprobar si ha iniciado sesión.
 2. Solicitar lista de ventas
 3. Ver en la base de datos
 4. Mostrar ventas
 5. Fin
- Buscar Venta
 1. Comprobar si ha iniciado sesión.
 2. Ingresar datos necesarios.
 3. Consulta la base de datos.
 - a. Mostrar datos.
 - b. Mostrar mensaje de error.
 4. Fin
- Borrar Venta
 1. Comprobar si ha iniciado sesión.
 2. Seleccionar producto a eliminar
 3. Comprobar que existe el producto, si no mensaje de error.
 4. Confirmar eliminación.
 5. Eliminar Producto
 6. Fin

Diagrama de tablas

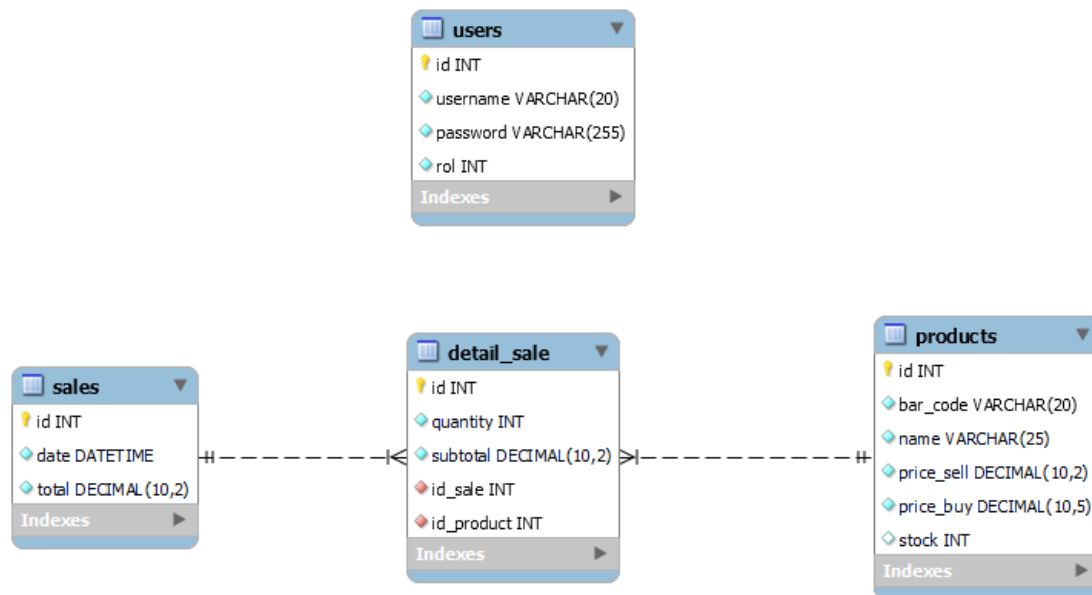


Diagrama de Tablas / Base de Datos

La base de datos esta dividida en dos partes una para los usuarios y la otra para las ventas y los productos.

Usuarios encargada de guardar el nombre de usuario, contraseña encriptada desde el front y su rol en formato numérico (El api se encarga de transformar el numero en texto).

Los productos y las ventas tienen una relación de muchas a muchas (N:M), al ser una relación N:M se crea una tabla en medio para conectarlos.

Manual de usuario

- BackEnd

Producto-Controller

GET
/api/products

Parameters
Try it out

Name	Description
page	Default value : 1
integer(\$int32) (query)	<input type="text" value="1"/>
size	
integer(\$int32) (query)	<input type="text" value="size"/>

Responses

Code	Description	Links
200	OK	No links
Media type <input type="text" value="*/*"/> Controls Accept header.		
Example Value Schema		
<pre>{ "data": [{ "id": 1073741824, "barCode": "string", "name": "string", "price_sell": 0, "price_buy": 0, "stock": 1073741824 }], "total": 1073741824, "currentPage": 1073741824, "pageSize": 1073741824, "next": "string", "previous": "string" }</pre>		
404	Not Found	No links
Media type <input type="text" value="*/*"/>		
Example Value Schema		
<pre>{ "error": "string", "message": "string" }</pre>		

Producto-Controller

GET
/api/products/{codeBar}

Parameters
Try it out

Name	Description
codeBar * required string (path)	<input type="text" value="codeBar"/>

Responses

Code	Description	Links
200	OK Media type <input type="text" value="*/"/> Controls Accept header. Example Value Schema <pre>{ "id": 1073741824, "barCode": "string", "name": "string", "price_sell": 0, "price_buy": 0, "stock": 1073741824 }</pre>	No links
404	Not Found Media type <input type="text" value="*/"/> Example Value Schema <pre>{ "error": "string", "message": "string" }</pre>	No links

Producto-Controller

POST

/api/products

^

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "id": 1073741824,
  "barCode": "string",
  "name": "string",
  "price_sell": 0,
  "price_buy": 0,
  "stock": 1073741824
}
```

Responses

Code	Description	Links
200	OK	No links
404	Not Found	No links

Media type

/

Example Value | Schema

```
{
  "error": "string",
  "message": "string"
}
```

Producto-Controller

PUT `/api/products/{codeBar}`

Parameters

Try it out

Name	Description
codeBar * required	
string (path)	<input type="text" value="codeBar"/>

Request body required

application/json

Example Value | Schema

```
{
  "id": 1073741824,
  "barCode": "string",
  "name": "string",
  "price_sell": 0,
  "price_buy": 0,
  "stock": 1073741824
}
```

Responses

Code	Description	Links
200	OK	No links
404	Not Found	No links

Media type

Example Value | Schema

```
{
  "error": "string",
  "message": "string"
}
```

Venta-Controller

GET
/api/sales

Parameters
Try it out

Name	Description
page	Default value : 1
integer(\$int32) (query)	1
size	
integer(\$int32) (query)	size

Responses

Code	Description	Links
200	OK	No links
	Media type <div>*/*</div> Controls Accept header. Example Value Schema <div> <pre>{ "data": [{ "id": 1073741824, "date": "2025-05-11T17:45:20.585Z", "total": 0, "details": [{ "quantity": 1073741824, "subtotal": 0, "products": { "id": 1073741824, "barCode": "string", "name": "string", "price_sell": 0, "price_buy": 0, "stock": 1073741824 } }] }], "total": 1073741824, "currentPage": 1073741824, "pageSize": 1073741824, "next": "string", "previous": "string" }</pre> </div>	
404	Not Found	No links
	Media type <div>*/*</div> Example Value Schema <div> <pre>{ "error": "string", "message": "string" }</pre> </div>	

Venta-Controller

GET
/api/sales/{id}

Parameters
Try it out

Name	Description
id * required integer(\$int32) (path)	id

Responses

Code	Description	Links
200	OK	No links
	Media type <div>*/*</div> Controls Accept header. <div>Example Value Schema</div> <pre>{ "id": 1073741824, "date": "2025-05-11T19:37:50.982Z", "total": 0, "details": [{ "quantity": 1073741824, "subtotal": 0, "products": { "id": 1073741824, "barCode": "string", "name": "string", "price_sell": 0, "price_buy": 0, "stock": 1073741824 } }] }</pre>	
404	Not Found	No links
	Media type <div>*/*</div> <div>Example Value Schema</div> <pre>{ "error": "string", "message": "string" }</pre>	

Venta-Controller

POST `/api/sales`

Parameters Try it out

No parameters

Request body required application/json

Example Value | **Schema**

```
{
  "id": 1073741824,
  "date": "2025-05-11T19:37:50.979Z",
  "total": 0,
  "details": [
    {
      "quantity": 1073741824,
      "subtotal": 0,
      "products": {
        "id": 1073741824,
        "barCode": "string",
        "name": "string",
        "price_sell": 0,
        "price_buy": 0,
        "stock": 1073741824
      }
    }
  ]
}
```

Responses


Code	Description	Links
200	OK	No links
404	Not Found	No links

Media type
/

Example Value | **Schema**

```
{
  "error": "string",
  "message": "string"
}
```

Venta-Controller

DELETE /api/sales/{id} 

Parameters Try it out

Name	Description
id * required integer(\$int32) (path)	<input type="text" value="id"/>

Responses

Code	Description	Links
200	OK	No links
404	Not Found	No links

Media type

Example Value | Schema

```
{  
  "error": "string",  
  "message": "string"  
}
```

User-Controller

POST /api/auth/login ^

Parameters Try it out

No parameters

Request body required application/json

Example Value | **Schema**

```
{
  "username": "string",
  "password": "string",
  "rol": "string"
}
```

Responses

Code	Description	Links
200	OK	No links
	<div>Media type */*</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{}</pre></div>	
404	Not Found	No links
	<div>Media type */*</div> <div>Example Value Schema</div> <div><pre>{ "error": "string", "message": "string" }</pre></div>	

Herramientas y Tecnologías empleadas



He decido usar MariaDB como sistema de gestión de bases de datos porque es una DDBB relacional, lo que permite organizar y consultar los datos de forma estructurada y eficiente. Además, es gratuita y de código abierto.



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 gratuita. [\[doc\]](#)



Java es un lenguaje de programación de propósito general, orientado a objetos y ampliamente utilizado en el desarrollo de aplicaciones. Se caracteriza por su filosofía de “escribir una vez, ejecutar en cualquier lugar” (Write Once, Run Anywhere), gracias a que el código Java se compila en bytecode, que puede ejecutarse en cualquier sistema que tenga una Máquina Virtual de Java (JVM). Versión de java 21. [\[doc\]](#)

Spring Boot

Spring Boot es un framework de código abierto cuyo objetivo es facilitar el desarrollo de aplicaciones basadas en el ecosistema de Spring. Ha sido diseñado para simplificar la configuración y el despliegue de aplicaciones, ayudando a los desarrolladores a centrarse menos en las tareas tediosas de configuración. [\[doc\]](#)

Spring Security

Spring Security es un framework que se centra en proporcionar autenticación y autorización a aplicaciones Java. Como todos los proyectos Spring, el verdadero poder de Spring Security reside en su facilidad de extensión para cumplir con requisitos personalizados. [\[doc\]](#)

Spring Data JPA

La conexión a la base de datos he utilizado spring data jpa por que facilita la implementación de repositorios basados en JPA. Facilita la creación de aplicaciones basadas en Spring que utilizan tecnologías de acceso a datos. [\[doc\]](#)



JWT (Json Web Token) es un estándar abierto (RFC 7519) que define un método compacto y autocontenido para la transmisión segura de información entre partes codificadas como un objeto JSON. Esta información puede verificarse y ser fiable porque está firmada digitalmente. [\[doc\]](#) Versión de JWT 0.12.6



Swagger™

Supported by SMARTBEAR

Swagger es una herramienta de código abierto para documentar, probar y visualizar API RESTFull. Utiliza la especificación OpenAPI para describir la estructura de la API, generando automáticamente documentación detallada y una interfaz web interactiva (Swagger UI) para explorar y probar los endpoints de la API. También permite generar código cliente en varios lenguajes de programación y validar la especificación OpenAPI para garantizar su cumplimiento con los estándares.



git



GitHub

Herramientas esenciales para el control de versiones y la colaboración en proyectos de desarrollo de software, permitiendo un seguimiento preciso de los cambios y la gestión eficiente del código Fuente.



POSTMAN

Una plataforma completa para probar y documentar APIs, ofreciendo un entorno de desarrollo intuitivo y amigable para probar solicitudes HTTP.



IntelliJ IDEA

Un potente entorno de desarrollo integrado (IDE) que ofrece características avanzadas y una excelente integración con tecnologías de desarrollo de Java y otros lenguajes de programación. Utilizado para el desarrollo de la parte de Back.



draw.io



EXCALIDRAW

Sketch hand-drawn like diagrams

Draw.io es una herramienta en línea gratuita y de código abierto utilizada para crear diagramas de forma visual.

Exalidraw también es una herramienta de pizarra virtual en línea gratuita para crear bocetos, ilustraciones y diagramas.

Propuesta de mejora

Por motivos de tiempo y mala organización no se ha terminado el proyecto de lo que me gustaría.

1. Lo primero sería terminar el front-end (Se ha empezado con lo más básico) para atraer a mas usuario. Las herramientas que se pueden utilizar son: Angular CLI, Angular Material (para las ventanas emergentes) y junto a Bootstrap para el diseño, al final decidí utilizar bootstrap porque me era más familiar y podría ir más rápido.
2. Tener un mapeador en el controlador para enviar solo los datos necesarios para el usuario ej.: en las ventas le envía una lista de ventas con todos los datos incluyendo productos detallados.
3. Añadir tablas extra para mejorar la búsqueda y su posición física en la tienda física ej.: tener la tabla de categoría para separar los distintos tipos de producto, el otro podría ser la dirección donde guarda en que pasillo y estante está el producto.
4. Tener la aplicación en un Docker para poder desplegarla automáticamente.
5. Tener una cache para poder responder al usuario si esta viendo los mismos datos.
6. Añadir un apartado en ventas que se pueda ver los productos más vendido en cierto tiempo y también poner ver los productos que tengan la cantidad por debajo de un numero para avisar señalizar que hay pocas unidades.
7. Crear la app de escritorio usando Electron que sirve para crear aplicaciones de escritorio usando JavaScript, HTML y CSS. Incrustando Chromium y Node.js dentro del mismo, Electron le permite mantener una base de código JavaScript y crear aplicaciones multiplataforma que funcionan en distintos S.O.
8. Conexión con dispositivos externos tipo impresoras térmicas para poder imprimir recibos, escáner de código de barras (la configuración seria si está el usuario iniciado sesión cuando escanea el código automáticamente lo añade a las lista para posteríos crear la venta).

Valoración personal

Este proyecto ha sido una muy buena experiencia tanto a nivel técnico como personal. A pesar de no haber terminado todo lo propuesto en la planificación y en el tiempo asignado.

Una de las cosas más importante que he aprendido es la necesidad de ser autodidacta. En el desarrollo del proyecto, he tenido dificultades que no habíamos visto en clase y me ha tocado buscar soluciones por mi mismo. Esto ha contribuido al hecho de que haya mejorado la interpretación de la documentación técnica y a un mejor conocimiento de la capacidad de resolver problemas.

También me ha sido útil para reforzar las bases como programación. A pesar de que no he terminado todo lo que me había propuesto, he sido capaz de llevar a la práctica muchas de las cosas vistas en clase. Ver como las cosas en clase en un proyecto real me ha ayudado a reforzar lo aprendido.

Respecto a la organización, al no seguir el horario eso perjudico el avance del proyecto. Subestimé el tiempo que necesitaría para ciertas tareas y mi horario personal. Esto hecho hizo que me diera cuenta de lo importante que es organizar y cumplir con el horario diseñado para evitar dejar cosas a medias.

Al final, aunque no pude entregar un proyecto finalizado como se había planificado, he aprendido la importancia de la organización y la actitud auto formativa. Todo eso hace que me sienta motivado para seguir mejorando, pero también en próximos proyectos éste a la altura con una mejor preparación y disciplina.

Puntos a descartar

Arquitectura y Tecnología: se ha implementado la arquitectura por capas, lo que ha permitido organizar el código de forma clara y modular. Al usar tecnologías y herramientas conocidas también ha facilitado el trabajo.

Seguridad: también me gustaría destacar el sistema de seguridad usando las librerías spring security y JWT (JSON WEB TOKEN). Teniendo la posibilidad de añadir mas roles a cada usuario, cada endpoint tiene seguridad específica sin afectar a los demás y no necesita una seguridad genérica, lo que permite escalabilidad al proyecto.

Gracias a todo lo anterior y más se ha hecho posible desarrollar una aplicación funcional, clara y con potencial de crecimiento. Aunque el proyecto no se ha completado del todo, la base técnica es sólida y está preparada para futuras mejoras.

Bibliografía

Licencia:

<https://creativecommons.org/>

Diagramas:

https://joapuiib.github.io/daw-ed/apunts/06_uml/010_casos_us

https://joapuiib.github.io/daw-ed/apunts/03_classes/010_classes

https://joapuiib.github.io/daw-ed/apunts/03_classes/010_classes

<https://guayahack.co/images/diagrama-de-clases.png>

Spring:

Tutorial Udemy de la empresa

<https://www.baeldung.com/persistence-with-spring-series>

<https://www.baeldung.com/jpa-joincolumn-vs-mappedby>

<https://medium.com/@burakkocakeu/in-spring-data-jpa-onetomany-what-are-these-fields-mappedby-fetch-cascade-and-orphanremoval-2655f4027c4f>

<https://github.com/cesguiro>

<https://stackoverflow.com/questions/58269386/how-to-get-the-id-of-saved-object-by-jpa>

<https://docs.spring.io/spring-security/reference/servlet/configuration/java.html>

<https://spring.io/guides/gs/securing-web>

<https://stackoverflow.com/questions/76973793/what-to-put-instead-of-parseclaimsjs-for-my-jwtgenerator-for-my-spring-securi>

<https://medium.com/somos-pragma/gu%C3%ADa-de-implementaci%C3%B3n-jwt-para-la-autenticaci%C3%B3n-en-java-db47b04eda54>

<https://blog.softtek.com/es/autenticando-apis-con-spring-y-jwt>

<https://adictosaltrabajo.com/2020/05/21/introduccion-a-spring-security/>

Memoria:

Documentos de ejemplo de antiguos compañeros

<https://swagger.io/tools/open-source/getting-started/>

<https://github.com/swagger-api/swagger-core>

<https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api>

<https://mvnrepository.com/artifact/org.springframework.openapi-starter-webmvc-ui/2.8.6>

<https://www.youtube.com/watch?v=DSOi2Hs80Co>

<https://github.com/swagger-api/swagger-core>

<https://mariadb.org/about/>

<https://www.ibm.com/es-es/topics/java-spring-boot>

<https://www.tokioschool.com/noticias/spring-boot/>

<https://aws.amazon.com/es/what-is/java/>

<https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-java-programming-language>

<https://swagger.io/docs/open-source-tools/swagger-codegen/codegen-v3/versioning/>