Ibertrade
E-commerce Calzado
Alumnos: Jorge Gordillo Redondo, Alberto González Pintado y Linxuan Zhan

¿Qué es Ibertrade?

Ibertrade es una plataforma en línea especializada en la venta de calzado que se distingue por un enfoque innovador: sus vendedores visitan outlets y tiendas donde los precios de las zapatillas son más bajos, para luego ofrecer precios más competitivos a la web. Además de facilitar una gestión eficiente del inventario y permitir a los administradores editar usuarios directamente desde la página, Ibertrade cuenta con un exclusivo programa VIP. Este programa ofrece ofertas especiales a sus miembros VIP y acceso anticipado a las últimas novedades en calzado.

¿Qué tecnologías usamos?

Spring Boot: Es el framework principal sobre el que se construye la aplicación. Se utiliza en todo el proyecto para la configuración y ejecución de la aplicación.

- **Archivo:** application.properties

Thymeleaf: Se utiliza como motor de plantillas para generar el HTML dinámicamente en el lado del servidor.

- **Archivos:**

head.html
about.html
catalogue.html
end.html
r.html (Zapatilla)
cabecera.html
r.html (Usuario)
c.html (Zapatilla)

Spring Data JPA: Se utiliza para la persistencia de datos y la interacción con la base de datos.

- **Archivo:** ZapatillaController (aunque el uso de Spring Data JPA se infiere por la interacción con servicios que a su vez interactúan con repositorios, el código del controlador sugiere operaciones de persistencia).

MySQL: Sistema de gestión de base de datos.

- **Archivo:** application.properties (configuración de conexión a la base de datos).

HTML/CSS/JavaScript: Tecnologías estándar de desarrollo web utilizadas para la estructura, diseño y funcionalidad del lado del cliente.

- **Archivos:**

HTML: Todos los archivos .html mencionados anteriormente.
CSS: Se incluye dentro de los archivos HTML a través de etiquetas <style> y referencias a archivos CSS externos.

JavaScript: Se utiliza en varios archivos HTML para añadir interactividad y manejar eventos del lado del cliente, como en cabecera.html para la funcionalidad del carrito de compras.
Bootstrap: Framework de CSS utilizado para el diseño responsivo y estilizado de la aplicación web.

- **Archivo:** head.html (se incluye Bootstrap a través de un enlace CDN).

Spring Mail: Se utiliza para el envío de correos electrónicos desde la aplicación.

- **Archivo:** application.properties (configuración de Spring Mail).

Organización de las carpetas

La organización de las carpetas en el repositorio de TFG (Trabajo de Fin de Grado) sigue una estructura que es común en proyectos de desarrollo de software, especialmente aquellos que utilizan Spring Boot para el desarrollo de aplicaciones web. Esta estructura ayuda a mantener el código fuente bien organizado, facilita la navegación y el mantenimiento del código, y mejora la colaboración entre desarrolladores. A continuación, se describe el propósito de las principales carpetas:

src/main/java/org/tfg/spring/tfg: Esta carpeta contiene el código fuente de la aplicación. Está organizada siguiendo el paquete base org.tfg.spring.tfg, lo cual es una práctica común para evitar conflictos de nombres y organizar el código de acuerdo con su funcionalidad y dominio.

domain: Contiene las clases de dominio o entidades (Carrito, CarritoZapatillas, etc.) que representan las tablas de la base de datos y sus relaciones.
controller: Alberga los controladores (HomeController, CarritoController, etc.), que manejan las solicitudes HTTP, mapean las peticiones a los métodos correspondientes, y devuelven las respuestas al cliente.
service: Incluye los servicios (CarritoService, etc.) que contienen la lógica de negocio, interactúan con los repositorios para acceder a los datos y realizan operaciones.
repository: Contiene los repositorios que proporcionan métodos para realizar operaciones CRUD (crear, leer, actualizar, eliminar) en las entidades.
src/main/resources: Esta carpeta incluye recursos que no son código fuente, como archivos de configuración, plantillas de vistas y archivos estáticos.

templates: Contiene las plantillas de Thymeleaf (home, carrito, etc.) que se utilizan para generar las vistas HTML dinámicas.
static: Almacena archivos estáticos como CSS, JavaScript e imágenes, que se utilizan para el estilo y la funcionalidad del lado del cliente.
application.properties: Archivo de configuración de Spring Boot donde se definen propiedades específicas de la aplicación, como configuraciones de base de datos, puerto del servidor, etc.
src/test/java: Contiene los tests unitarios y de integración que ayudan a verificar que el código funciona como se espera.

La organización de las carpetas sigue el estándar de Maven y Spring Boot, lo que facilita la automatización de tareas como la compilación, el empaquetado, y la ejecución de pruebas. Además, esta estructura es ampliamente reconocida y utilizada en la industria del software, lo que hace que el proyecto sea más accesible para nuevos desarrolladores y mejora la eficiencia del desarrollo.

Casos de uso

Caso de Uso: Usuario Admin

Un usuario admin tiene la capacidad de establecer a otros usuarios como administradores o como usuarios VIP.

Establecer a un usuario como administrador:

Para hacer a un usuario administrador, se utiliza el método setAdmin de UsuarioService.

usuarioService.setAdmin("nombreUsuario");

Establecer a un usuario como VIP:

Aunque el usuario sea admin, también puede tener la capacidad de hacer a otros usuarios

VIP utilizando el método setVip.

usuarioService.setVip("nombreUsuario");

Caso de Uso: Usuario VIP

Un usuario VIP no tiene capacidades administrativas pero puede tener beneficios o características especiales dentro de la aplicación, como descuentos o acceso a contenido exclusivo. Sin embargo, en el código fuente proporcionado, la distinción de comportamiento para usuarios VIP no está explícitamente definida más allá de la asignación de la propiedad VIP. Por lo tanto, el caso de uso se centra en cómo un usuario se convierte en VIP.

Ser marcado como VIP por un admin:

Un usuario se convierte en VIP cuando un administrador llama al método setVip con su nombre.

usuarioService.setVip("nombreUsuario");

Caso de Uso: Usuario Regular

Un usuario que no es ni admin ni VIP puede realizar las siguientes acciones básicas:

Registrarse en la plataforma:

Un nuevo usuario puede registrarse utilizando el método save de UsuarioService.

usuarioService.save("nombre", "dni", "mail", "contraseña");
Iniciar sesión:

Un usuario puede iniciar sesión mediante el método login.

Usuario usuario = usuarioService.login("nombre", "contraseña");
Actualizar su información:

Un usuario puede actualizar su información personal a través del método update de UsuarioService.

usuarioService.update(idUsuario, "nombreActualizado", "dniActualizado", "mailActualizado");
Eliminar su cuenta:

Un usuario puede eliminar su cuenta utilizando el método delete.

usuarioService.delete(idUsuario);

Caso de Uso: Compra de Usuario Regular

Agregar productos al carrito: El usuario puede agregar productos (en este caso, zapatillas) a su carrito. Esto se maneja a través del método updateSaveCrarito de la clase CarritoService. Este método verifica si ya existe un carrito para el usuario; si es así, agrega el producto al carrito existente. Si no, crea un nuevo carrito y agrega el producto.

public Carrito updateSaveCrarito(ZapatillaCantidad zapatillaCantidad, HttpSession s){
// Implementación
}

Finalizar la compra: Una vez que el usuario ha agregado todos los productos deseados al carrito, puede proceder a finalizar la compra. Esto se realiza a través del método finalizarCompra en CarritoService, el cual es invocado por el método finalizarCompra en CarritoController. Este proceso verifica el stock de cada producto en el carrito y, si hay suficiente stock, reduce el stock y marca el carrito como comprado.

```
@GetMapping("/carritos/finalize")
public void finalizarCompra(HttpSession s) {
// Implementación en CarritoController
}

public void finalizarCompra(HttpSession s){
// Implementación en CarritoService
}
```

Enviar confirmación de compra: Después de finalizar la compra, si el usuario está autenticado, se envía un correo electrónico de confirmación de la venta. Esto se maneja en el mismo método finalizarCompra en CarritoController, donde se llama a mailService.sendSaleConfirmEmail(usuario) si el usuario no es null.

```
if (usuario != null) {
mailService.sendSaleConfirmEmail(usuario);
}
```

Cancelar la compra: Si el usuario decide cancelar la compra antes de finalizarla, puede hacerlo a través del método cancelarCompra en CarritoService. Este método elimina los productos del carrito y luego elimina el carrito.

```
public void cancelarCompra( List<Long> carritoZapatillasId,Long carritoId,HttpSession s){
// Implementación
}
```

Visualizar el carrito: El usuario puede ver los productos agregados al carrito antes de finalizar la compra. Esto se maneja a través del método findCarritoByUsuarioId en CarritoService, que es utilizado por el método getResumen en CompraController para mostrar un resumen del carrito.

```
@GetMapping("/comprar")
public String getResumen(HttpSession s, ModelMap m) {
// Implementación en CompraController
}
```

Casos de uso de un usuario regular tras registrarse

Login: Un usuario puede iniciar sesión utilizando su nombre y contraseña. Esto se maneja a través del método login en UsuarioService.

```
public Usuario login(String nombre, String contraseña) throws Exception {
Usuario usuario = usuarioRepository.getByNombre(nombre);
if (usuario == null) {
throw new Exception("El Usuario " + nombre + " no existe");
}
if (!passwordEncoder.matches(contraseña, usuario.getContraseña())) {
throw new Exception("La contraseña para el Usuario " + nombre + " es incorrecta");
}

return usuario;
}
```

Actualizar Información: Un usuario puede actualizar su información personal como nombre, DNI y correo electrónico. Esto se realiza a través del método update en UsuarioService y se accede a través de la ruta /usuario/u en UsuarioController.

```
public void update(Long idUsuario, String nombre, String dni, String mail) {
Usuario usuario = usuarioRepository.findById(idUsuario).orElse(null);
if (usuario != null) {
usuario.setNombre(nombre);
usuario.setDni(dni);
usuario.setMail(mail);
usuarioRepository.save(usuario);
}
}

public void delete(Long idUsuario) {
usuarioRepository.deleteById(idUsuario);
carritoRepository.deleteByUsuarioId(idUsuario);
}
```

Navegar por el Catálogo: Un usuario regular puede navegar por el catálogo de productos. Esto se asume como parte de la funcionalidad general de la aplicación.

Agregar Productos al Carrito: Un usuario puede agregar productos a su carrito de compras. La gestión del carrito de compras se asocia con el usuario a través de la relación @OneToMany con la entidad Carrito en Usuario.

```
@OneToMany(mappedBy = "usuario", cascade = CascadeType.ALL, orphanRemoval = true)
private List<Carrito> carritos;
```

Realizar Compras: Gestión de carritos y el proceso de checkout.