

TRABAJO DE FIN DE GRADO

**GRADO SUPERIOR EN DESARROLLO DE  
APLICACIONES MULTIPLATAFORMA**

**ARAKNA**

---

DESARROLLO DE APLICACIÓN CLIENTE Y SERVIDOR PARA WEB Y  
MÓVILES PARA EL GRUPO DE MÚSICA ARAKNA

---

AUTOR: VÍCTOR MARTÍN ROSA  
TUTORA: RAQUEL CERDÁ LOSA  
AÑO: 2020/2021

## ÍNDICE

Introducción, justificación y objetivos	página 2
Módulos formativos aplicados en el trabajo	página 3
Herramientas/Lenguajes utilizados	página 4
Fases del proyecto	página 5
Conclusiones y mejoras del proyecto	página 6
Bibliografía	página 7
Anexos	página 8
Manual de usuario	página 9
Manual de administrador	página 16
Manual de usuario - aplicación móvil	página 19
Prerrequisitos para la instalación del BackEnd del proyecto	página 23
Prerrequisitos para la instalación del FrontEnd del proyecto	página 25
Prerrequisitos para la instalación del FrontEnd de la app	página 26
Manual del programador	página 28

## Introducción, justificación y objetivos

Para este proyecto se ha planteado el aprendizaje e investigación como objetivo secundario, por lo que se ha realizado el backend del proyecto con Spring y Spring Security, y el frontend con Angular.

El objetivo principal es que tanto la aplicación móvil como la página web tengan sesiones, vistas y roles para que el usuario pueda registrarse, y que aquellos usuarios con permisos de administrador puedan gestionar múltiples aspectos de la página, cómo añadir o eliminar productos de la tienda. El contenido de la web se centra alrededor del grupo de música Arakna, con información acerca del grupo, redes sociales, conciertos y otros.

Adicionalmente, generar una aplicación Android, haciendo uso de las funcionalidades creadas en Angular, y el backend de Spring.

## Módulos formativos aplicados en el trabajo

- Programación - Spring y Spring Security
- Entornos de Desarrollo - Git
- Lenguaje de Marcas y Desarrollo de Interfaces - HTML / CSS / Angular
- Programación Multimedia y Dispositivos Móviles - Ionic

## Herramientas/Lenguajes utilizados

### Herramientas:

- IDEs: IntelliJ, Visual Studio Code, Android Studio
- XAMPP: Apache, MySQL

### Lenguajes:

- Java: Spring, Spring Security
- JavaScript
- TypeScript
- HTML
- CSS

## Fases del proyecto

Definir el alcance del proyecto: Establecer los objetivos del proyecto: Una página web con frontend diseñado en Angular, y backend diseñado en Spring con SpringSecurity, para así poder definir usuarios, roles y vistas en el frontend, con la incorporación de un carro de la compra funcional. Adicionalmente, creación de una aplicación móvil con Ionic, utilizando las funcionalidades de Angular y el mismo backend.

Definir base de datos: Diseñar la base de datos y sus tablas según las necesidades presentadas (user, order, product).

Crear el modelo entidad-relación en el backEnd: Creación de las diferentes clases para ajustarse a la base de datos.

Diseñar y crear el frontEnd: Diseño visual de la página principal de la web. Creación de clases en el FrontEnd.

Refactorizar estructura del proyecto para satisfacer las nuevas necesidades: Ajustar la base de datos a las necesidades generadas por SpringSecurity, se agregan las tablas hibernate\_sequence, order se divide en orders y order\_detail, user se divide en user, user\_info, user\_rol y rol. Se ajustan las clases y estructura del backend y del frontend.

Conectar SpringSecurity con el FrontEnd: Creación de endpoints en el backend y frontend para que la información viaje desde la web hasta la base de datos.

Crear todas las funcionalidades del carro de la compra: Funciones para crear usuarios e iniciar sesión. Crear, borrar, editar, ver y listar productos. Añadir y eliminar productos del carro de la compra. Pagar.

Corrección de errores: Corrección de errores diversos tanto en el backend como en el frontend.

Crear aplicación móvil: Instalación de Ionic y la generación de la aplicación Android.

## Conclusiones y mejoras del proyecto

Se han logrado la mayoría de objetivos propuestos, siendo como punto de mejora, la incorporación de más funcionalidades a la aplicación móvil. Adicionalmente se han adquirido nuevos conocimientos.

## Bibliografía

Frameworks

<https://ionicframework.com/>

<https://angular.io/>

<https://jwt.io/>

<https://capacitorjs.com/docs/getting-started/with-ionic>

<https://getbootstrap.com/>

Resources

<https://www.flaticon.com/>

<https://linearicons.com/free>

Tutoriales usados

<https://www2.deloitte.com/es/es/pages/technology/articles/Ionic-principales-framework-visuales.html>

<https://javadesde0.com/introduccion-a-los-beans/>

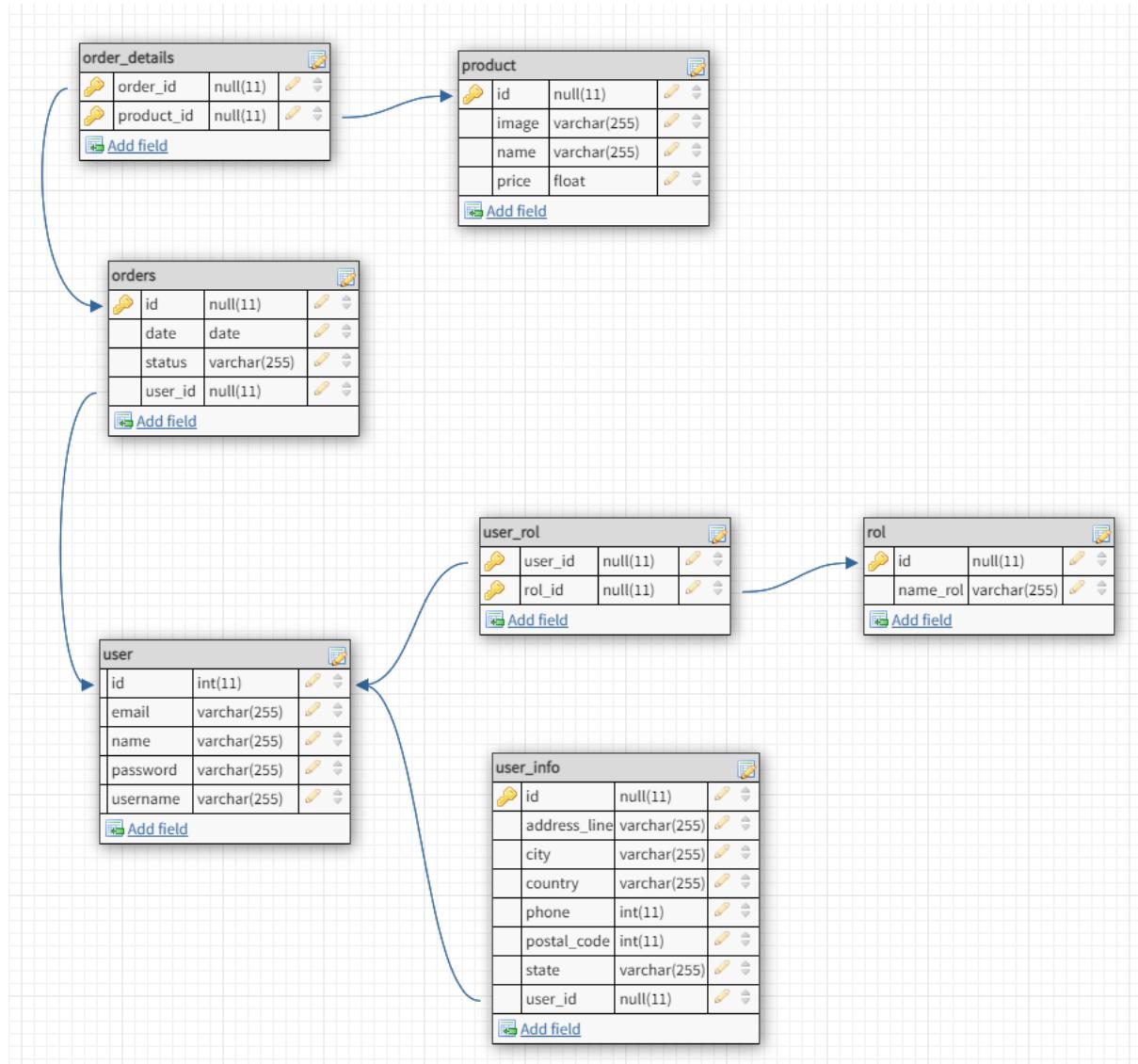
<https://blog.softtek.com/es/autenticando-apis-con-spring-y-jwt>

<https://www.baeldung.com/>

<https://desarrolloweb.com/manuales/manual-angular-2.html>

## Anexos

### Esquema de la base de datos



## Manual de usuario

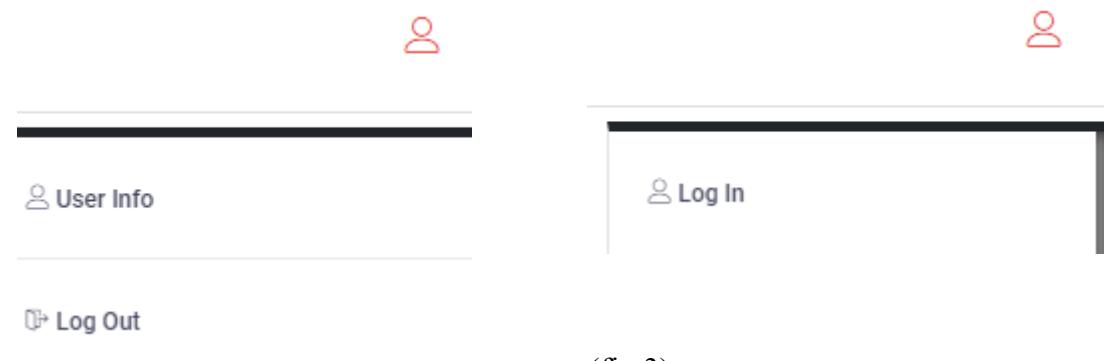
La página web está compuesta por una ventana principal donde se muestra diversa información sobre la banda, como próximos conciertos, últimos discos, videos destacados, noticias y una tienda, donde el usuario podrá adquirir productos como discos y entradas.

En la parte superior se encuentra un menú adherido que acompañará al usuario mientras se desplaza a la parte inferior de la página. Se pueden utilizar los elementos centrales para desplazarse más rápidamente a la sección deseada. (fig 1)



(fig 1)

En la parte izquierda del menú, se muestra un mensaje de bienvenida al usuario, así como un desplegable que permitirá ver y modificar la información personal del usuario, e iniciar o cerrar sesión; y un desplegable con el carro de la compra. (fig 2)(fig3)



(fig 2)

(fig 3)

El carro de compra dispone de varias funcionalidades, como eliminar el producto añadido, ver el precio total de los productos en el carro y realizar la compra. (fig 4)(fig 5)

(fig 4)

(fig 5)

- Primera sección: Un carrusel con información de las canciones del último álbum. (fig 6)
- Segunda sección: Lugar, fecha y hora de los próximos conciertos. (fig 7)
- Tercera sección: Últimos álbumes y sencillos, con códigos QR de Spotify y enlaces. (fig 8)
- Cuarta sección: Vídeos destacados. (fig 9)
- Quinta sección: Noticias y novedades, últimos eventos. (fig 10)
- Sexta sección: Tienda de discos y entradas para los conciertos. Para añadir un producto al carrito, el usuario debe deslizar el cursor por encima del producto, y se desplegará la opción “Add to Cart” (fig 11) (fig 12)

Antarctica - Album - 2019

## Kraken

Lore ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor  
ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation  
ullamco laboris nisi ut aliquip.



(fig6)

## Live



Madrid

April 1st - 20:00



Barcelona

May 4th - 18:00



Rome

May 30th - 19:00

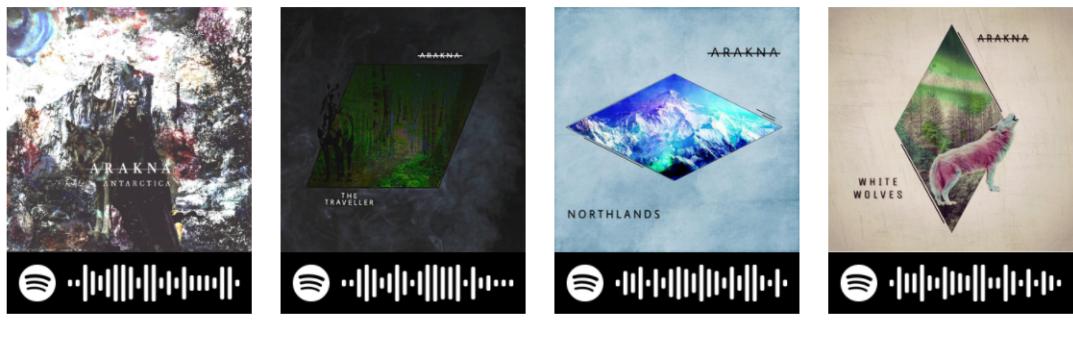


Milan

June 22nd - 21:00

(fig 7)

## Albums & Singles



Antarctica - Album - 2019

The Traveller - Single - 2018

Northlands - Single - 2018

White Wolves - Single - 2017

(fig 8)

## Videos



(fig 9)

## News



Concert #1

5th December 2021

Nemo Enim Ipsum Voluptatem Quia Voluptas Sit Aspernatur  
Aut Odit Aut Fugit, Sed Quia Consequuntur Magni Dolores  
Eos Qui Ratione Voluptatem Sequi Nesciunt....



Concert #2

5th December 2021

Nemo Enim Ipsum Voluptatem Quia Voluptas Sit Aspernatur  
Aut Odit Aut Fugit, Sed Quia Consequuntur Magni Dolores  
Eos Qui Ratione Voluptatem Sequi Nesciunt....



Concert #3

5th December 2021

Nemo Enim Ipsum Voluptatem Quia Voluptas Sit Aspernatur  
Aut Odit Aut Fugit, Sed Quia Consequuntur Magni Dolores  
Eos Qui Ratione Voluptatem Sequi Nesciunt....

(fig 10)

## Shop



Antarctica

20€



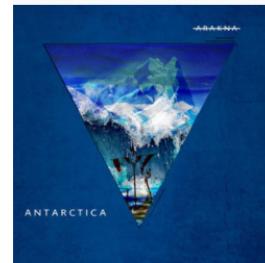
Toxic

15€



Above The Forest

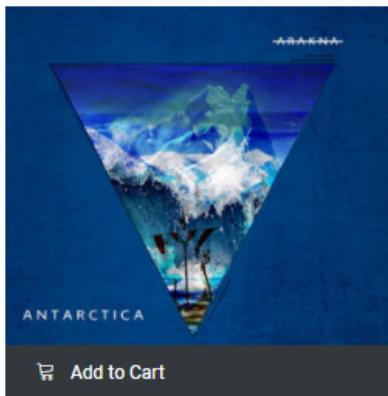
42€



Antarctica - Single

5€

(fig 11)



Antarctica - Single

5€

(fig 12)

### Inicio de sesión

Cuando el usuario pulsa el botón “Log in”, es redirigido a otra página, donde podrá iniciar sesión o registrarse. (fig 13)(fig 14)

<a href="#">Sign In</a>	<a href="#">Create Account</a>
Username	
<input type="text"/>	
Password	
<input type="password"/>	
<a href="#">Sign In</a>	

(fig 13)

<a href="#">Sign In</a>	<a href="#">Create Account</a>
Name	
<input type="text"/>	
Username	
<input type="text"/>	
Password	
<input type="password"/>	
Email	
<input type="text"/>	
<a href="#">Create Account</a>	

(fig 14)

## Detalles de usuario.

Para añadir o modificar información personal, el usuario deberá pulsar el botón “User Info”, ubicado en el desplegable de usuario en menú superior. Podrá añadir la información necesaria para enviar los pedidos con éxito: dirección, ciudad, país, teléfono, código postal y comunidad autónoma. (fig 15)

The figure shows a user interface for updating personal information. It consists of a vertical list of input fields with placeholder text, followed by two action buttons at the bottom.

- Address Line:** Placeholder: "Address Line".
- City:** Placeholder: "City".
- Country:** Placeholder: "Country".
- Phone:** Placeholder: "Phone".
- Postal Code:** Placeholder: "Postal Code".
- State:** Placeholder: "State".

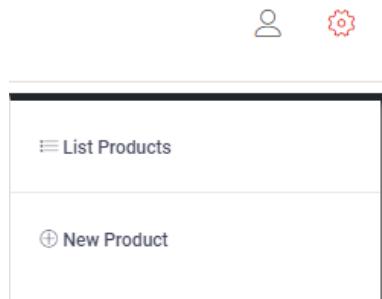
**Buttons:**

- Update Info:** A green button with a pen icon and the text "Update Info".
- Back:** A blue button with a left arrow icon and the text "Back".

(fig 15)

## Manual de administrador

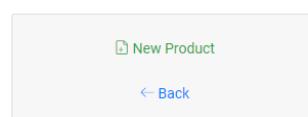
Los usuarios con permiso de administrador tienen acceso a funcionalidades adicionales. En el menú adherido, se mostrará un ícono adicional, un engranaje, el cual es un menú desplegable con las opciones de listar todos los productos y añadir un nuevo producto. (fig 16)



(fig 16)

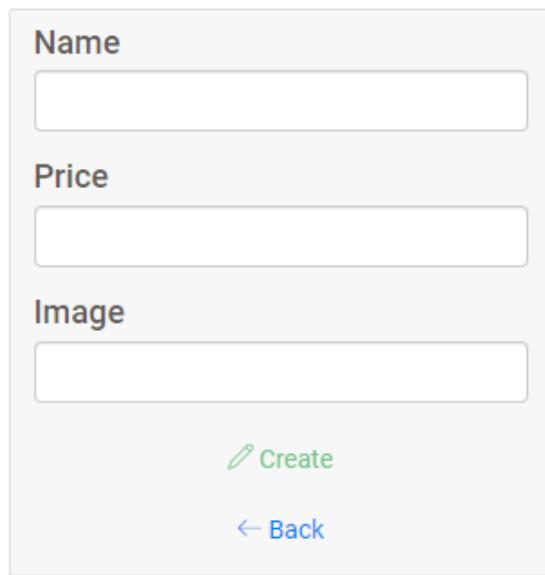
Listar productos: Esta opción redirige al usuario a una nueva ventana, donde podrá ver todos los productos, así como añadir uno nuevo, ver los detalles, editarlos, o borrarlos. (fig 17)

ID	Name	Price	Image	View	Edit	Delete
1	Antarctica	20€	Https://Drive.Google.Com/Uc?Export=View&Id=1iD_r3mUI5BI3j_YMwqc9REiA3BuT5Xvt	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
2	Toxic	15€	Https://Drive.Google.Com/Uc?Export=View&Id=11hCh_2os9HL5torWqWN7VdmfmuqbRAq	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
3	Above The Forest	42€	Https://Drive.Google.Com/Uc?Export=View&Id=1IVI9xLsKn_iu7W_UtY5NE6Wn2R3j5jrU	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
4	Antarctica - Single	5€	Https://Drive.Google.Com/Uc?Export=View&Id=1Ddj2LZ9mmpkCdJn3M5HA6wylo_-KqqIA	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
5	FN-KN	7€	Https://Drive.Google.Com/Uc?Export=View&Id=1v4Lz3msJKVmQB-EFpiFWrpqF9CZ0R53R	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
6	Northlands	13€	Https://Drive.Google.Com/Uc?Export=View&Id=105twK5JcgFnqae63fsDI7-3PiVw8Q956	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
7	The Sound Of The Rain	17€	Https://Drive.Google.Com/Uc?Export=View&Id=144I8vVbTkxZ4qYIAf2F8J-UCMLjd-And	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
8	The Traveller	4€	Https://Drive.Google.Com/Uc?Export=View&Id=1TYjq3MsQkICEmtpOpBozHCz0QJ8l612	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
9	White Wolves	11€	Https://Drive.Google.Com/Uc?Export=View&Id=1mh-SEIffMEVHX0AnbJ-BHKsLg5WhRZ-Z	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
10	Madrid Concert Ticket	5€	Https://Drive.Google.Com/Uc?Export=View&Id=1trQZJr1oDsT5eHXd-DNIHu6ywA4R5t2	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>



(fig 17)

Ventana para añadir un nuevo producto (fig 18)

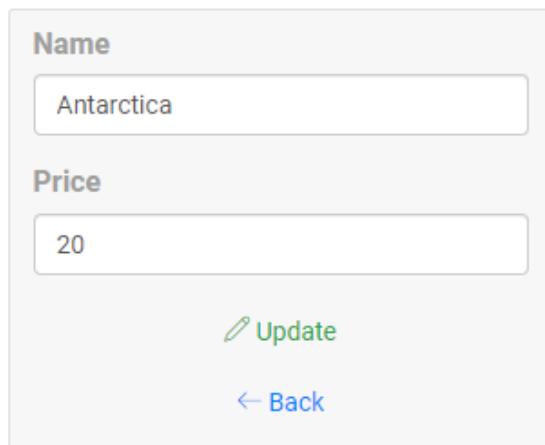


Formulario para añadir un nuevo producto:

- Name**: Campo de texto vacío.
- Price**: Campo de texto vacío.
- Image**: Campo de texto vacío.
- Create**: Botón verde con icono de lápiz y texto "Create".
- Back**: Botón azul con icono de flecha y texto "Back".

(fig 18)

Ventana para editar un producto (fig 19)

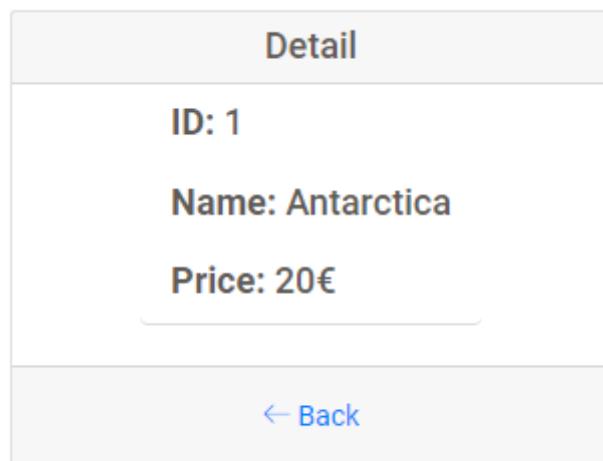


Formulario para editar un producto:

- Name**: Campo de texto con valor "Antarctica".
- Price**: Campo de texto con valor "20".
- Update**: Botón verde con icono de lápiz y texto "Update".
- Back**: Botón azul con icono de flecha y texto "Back".

(fig 19)

Ventana para ver los detalles del producto (fig 20)



(fig 20)

Los administradores también pueden editar o eliminar productos directamente desde la página principal, utilizando los botones “edit” y “delete” que aparecen bajo cada producto. (fig 21)



Antarctica - Single

5€

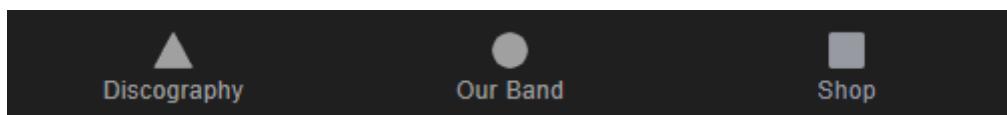
Edit

Delete

(fig 21)

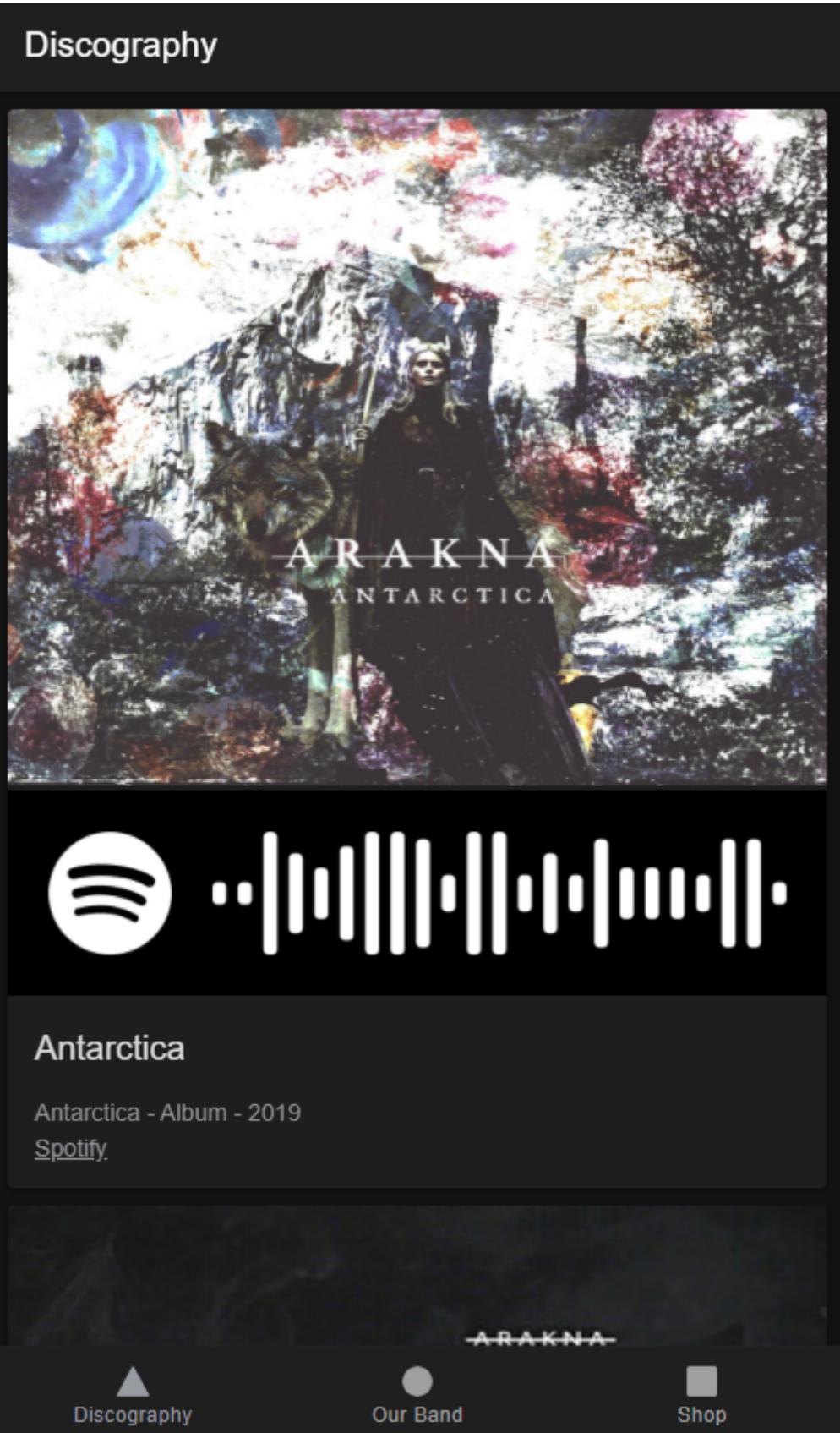
## Manual de usuario - aplicación móvil

La aplicación móvil está compuesta por tres ventanas. El usuario podrá navegar a través de ellas utilizando la barra inferior. (fig 22)



(fig 22)

- Primera ventana - Discografía. En esta ventana, el usuario puede ver la discografía de la banda organizada por álbumes, así como los códigos QR de Spotify y los hipervínculos.(fig 23)
- Segunda ventana - La banda. En esta ventana se muestra a los integrantes de la banda, así como sus instrumentos y los perfiles de la red social Instagram.(fig 24)
- Tercera ventana - Tienda. En esta ventana, el usuario puede ver un catálogo de los productos que se encuentran en la tienda de la página web.(fig 25)



(fig 23)

## Our Band



### About us

A METAL band from Badajoz, Spain

@javidroide: Guitar/Keys

@juanex\_\_: Bass

@jisusfrank\_22: Guitar

@3blllvck: Vocals/Guitar

@juanma\_teo : Drums

[Twitch](#)



[Discography](#)

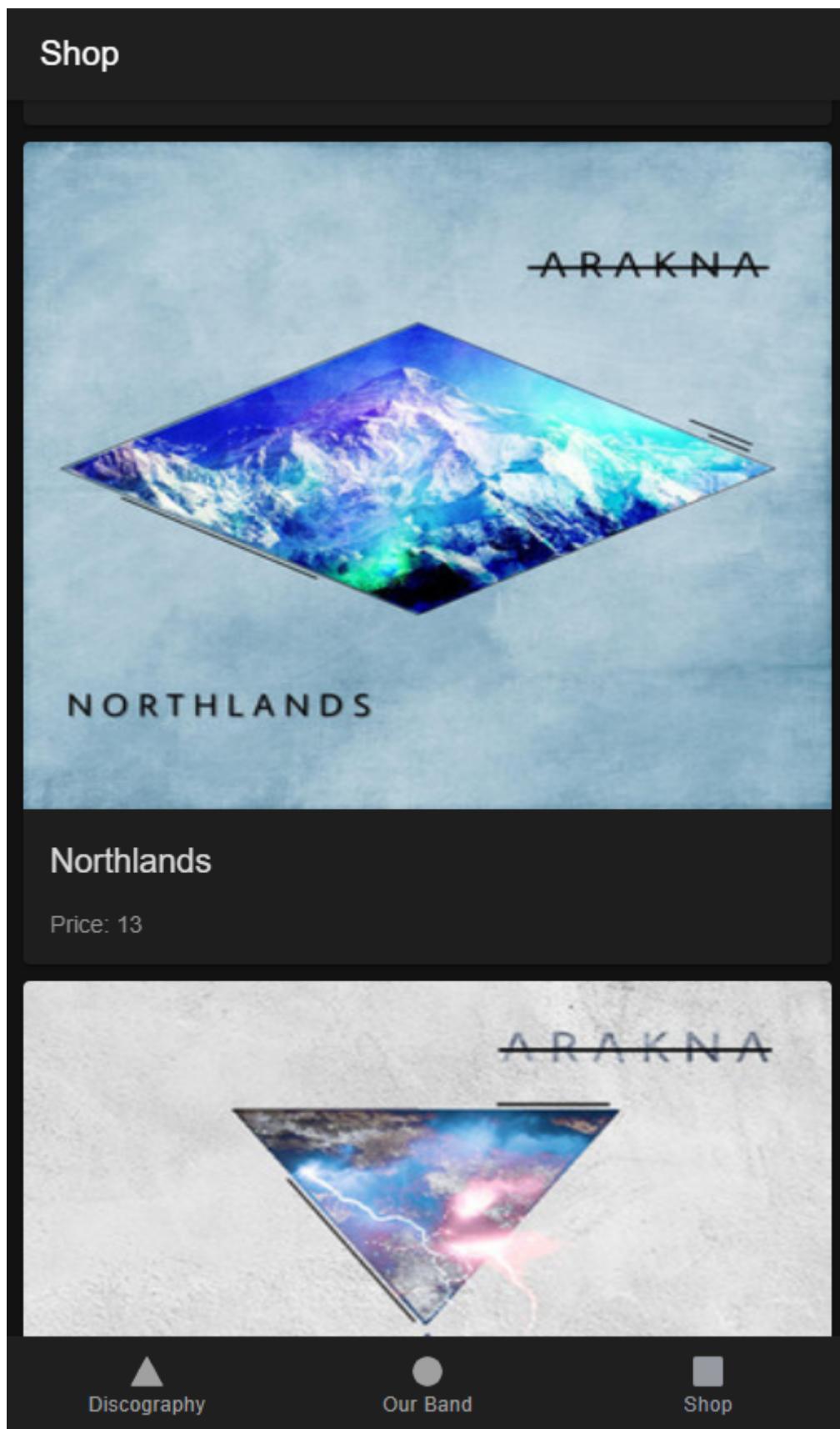


[Our Band](#)



[Shop](#)

(fig 24)

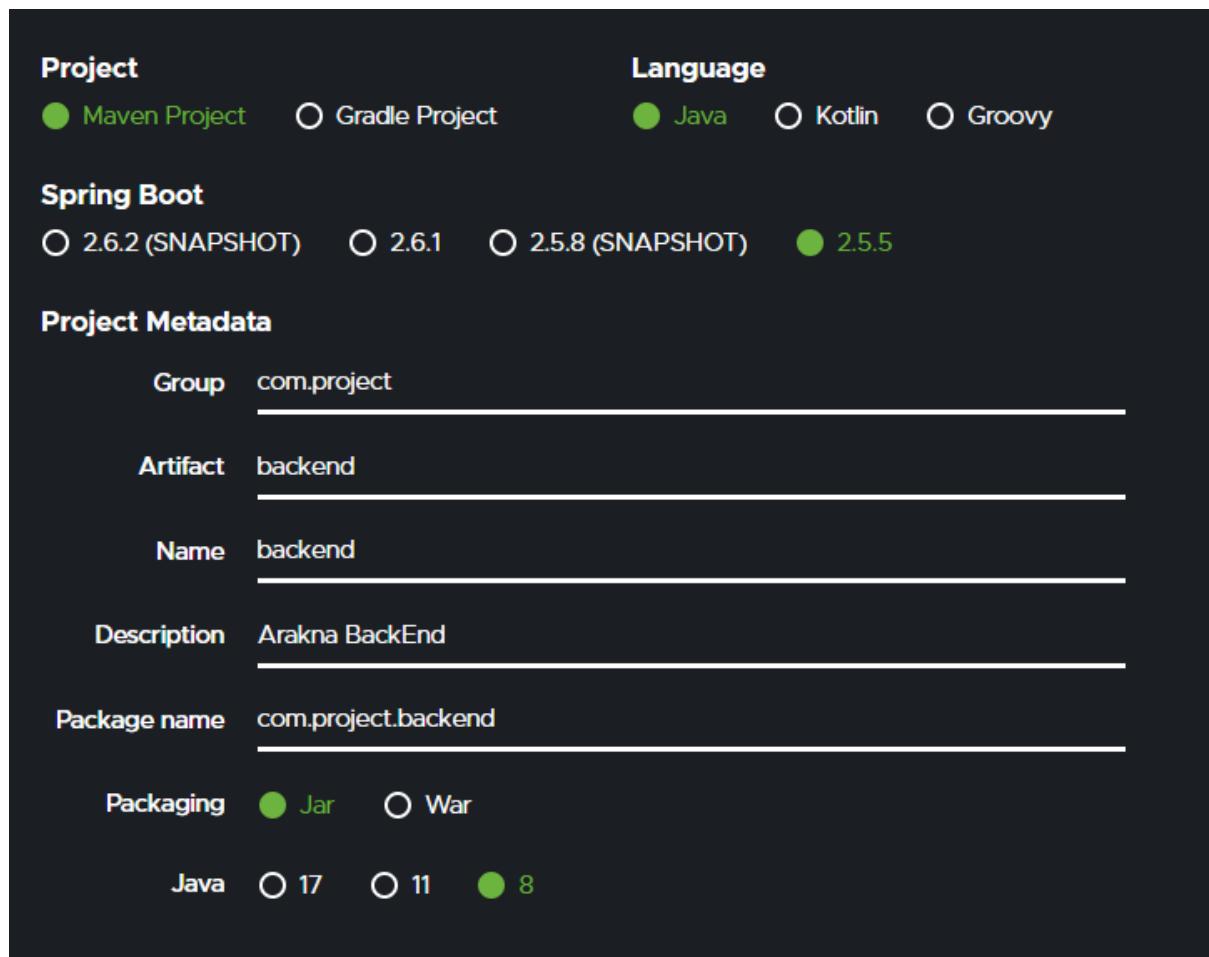


(fig 25)

## Prerrequisitos para la instalación del BackEnd del proyecto

- JDK 8
- IDE - IntelliJ
- Gestor de dependencias - Maven
- Spring y SpringSecurity con sus respectivas dependencias - Versión 2.5.5
- XAMPP - Apache + MySQL y fichero .sql con tablas y contenido.

Para generar un nuevo proyecto con SpringBoot, se ha utilizado el SpringInitializr, a través de la web <https://start.spring.io/> (fig 26), y se han agregado las dependencias requeridas. (fig 27)



(fig 26)

The screenshot shows a dark-themed interface for managing Spring Boot dependencies. At the top, there's a header with the word "Dependencies" and a button labeled "ADD DEPENDENCIES... CTRL + B". Below the header, there are five main sections, each with a title, a category tag, and a brief description:

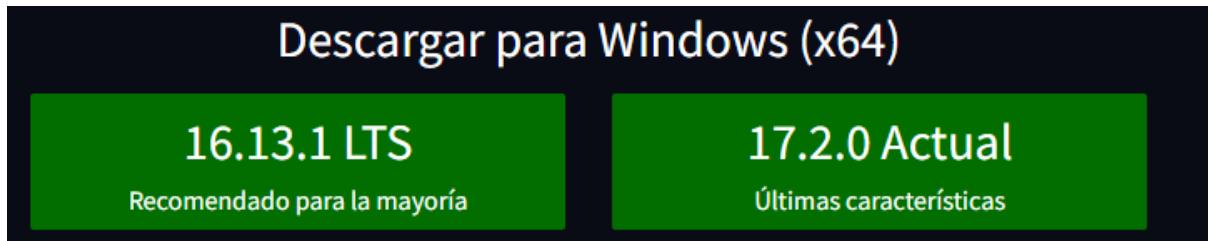
- Spring Boot DevTools** [DEVELOPER TOOLS]: Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- Spring Security** [SECURITY]: Highly customizable authentication and access-control framework for Spring applications.
- Spring Data JPA** [SQL]: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- Validation** [I/O]: Bean Validation with Hibernate validator.
- Spring Web Services** [WEB]: Facilitates contract-first SOAP development. Allows for the creation of flexible web services using one of the many ways to manipulate XML payloads.

(fig 27)

## Prerrequisitos para la instalación del FrontEnd del proyecto

- Angular - versión 13.0.4
- Node.js

Node.js es el entorno de ejecución para javaScript. Es requerido instalarlo. Es posible descargarlo a través de la web <https://nodejs.org/es/> (fig 28)



(fig 28)

Es necesario instalar el framework de Angular para poder trabajar.

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

Tras la instalación global, es necesario generar las dependencias necesarias dentro de la carpeta del proyecto, con el comando:

```
> npm install
```

Adicionalmente, se instala el componente ngx-toastr, que permite crear toasts.

```
> npm install ngx-toastr --save
```

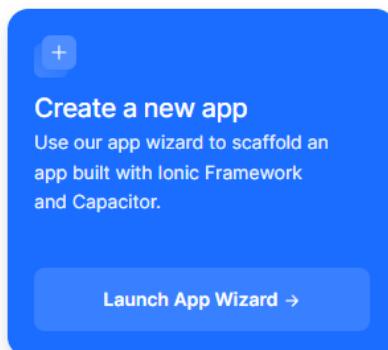
Por último, se arranca el servicio.

```
> ng serve
```

## Prerrequisitos para la instalación del FrontEnd de la app

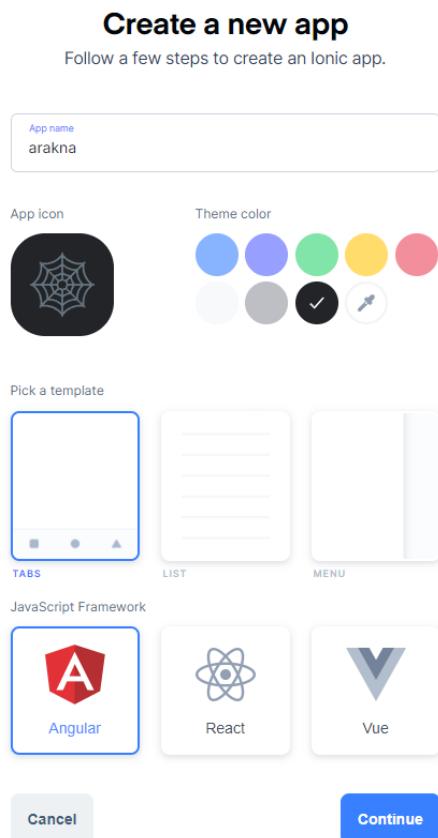
Es necesario instalar el framework de Ionic para poder trabajar. Este framework trabaja con la base de Angular, por lo que las funcionalidades de Angular pueden ser utilizadas en Ionic.

Para generar una aplicación, se crea a través de la web de Ionic: <https://ionicframework.com/> (fig 29)



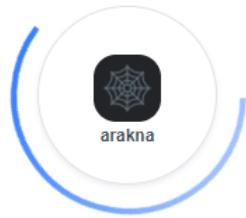
(fig 29)

Al igual que al crear una nueva aplicación con Android Studio, Ionic ofrece plantillas. (fig 30)



(fig 30)

Tras compilar y generar la aplicación, es necesario copiarla al directorio local de destino donde se arrancará el servicio de Ionic. (fig 31)



## Compiling your app

Hang tight for just a moment.

Generating app...

(fig 31)

Es necesario instalar el framework de Angular para poder trabajar.

```
> npm install -g @ionic/cli cordova-res
```

Tras la instalación global, es necesario generar las dependencias necesarias dentro de la carpeta del proyecto, con el comando:

```
> npm install
```

Por último, se arranca el servicio.

```
> ng serve
```

Para generar la aplicación de Android, es necesario ejecutar el comando:

```
> ionic capacitor build android --release
```

Para exportar la APK, se puede copiar la app-debug.apk desde la ruta relativa “android\app\build\outputs\apk\debug” o se puede generar una Signed APK desde Android Studio.

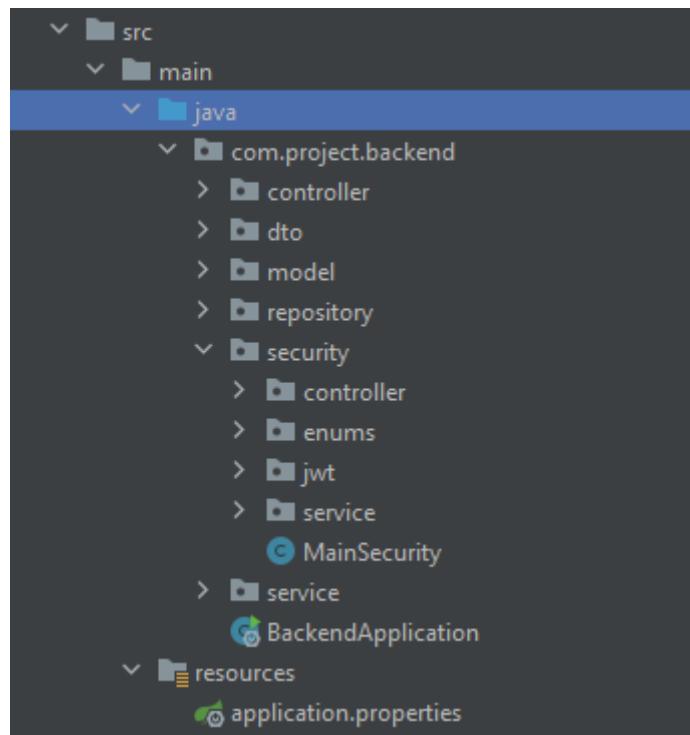
## Manual del Programador

El proyecto consta de tres partes fundamentales. Un backend desarrollado en Spring Security el cual cuenta con JWT (JSON Web Token) para la validación de usuarios y el intercambio correcto de datos. El frontend se compone de dos aplicaciones basadas en node js, un entorno en tiempo de ejecución multiplataforma de código abierto basado en el lenguaje de programación javaScript asíncrono. Este frontend se divide en aplicación web y aplicación móvil, la aplicación web está desarrollada con Angular y plantillas de Bootstrap, mientras que la aplicación móvil está desarrollada en Ionic, un framework del mismo tipo para aplicaciones móviles.

### Backend

El lenguaje utilizado es en su totalidad java, en la aplicación servidor está implementada una api-rest, la cual gestiona todas las peticiones que llegan de las aplicaciones del frontend que maneja el usuario. Para el manejo de la base de datos se utiliza el framework Spring Boot, para gestionar la seguridad se utiliza Spring Security y JWT como se ha mencionado anteriormente.

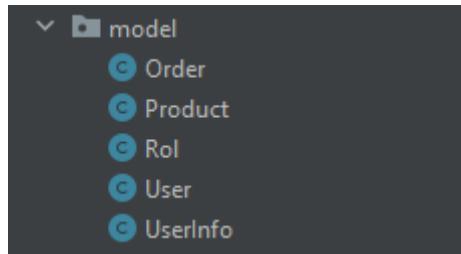
La estructura del proyecto es la siguiente:



- **Controller** contiene los restControllers, donde se encuentran los endpoints a los que accede el frontend. Se han definido OrderController, ProductController y UserInfoController. Dentro del paquete security también existe un paquete controller, este contiene el AuthController, el cual gestiona el login de usuarios, y por este motivo se encuentra en el paquete de security.
- **Dto** contiene los data transfer objects los cuales se utilizan para transportar la información del backend al frontend y viceversa.
- **Model** contiene el modelo de la base de datos, donde se definen todas las entidades que existen en la base de datos con sus correspondientes relaciones para la posible implementación de datos en la base de datos.
- **Repository** contiene los repositorios de la aplicación.
- **Security** contiene todos los paquetes y clases que gestionan la seguridad de la aplicación, tanto los roles como el control del login. Dentro contiene un paquete jwt en el que se ubican las clases relacionadas con el token de autenticación. También se encuentra un paquete service el cual contiene los servicios relacionados con usuarios y roles. La clase MainSecurity es la clase de configuración de Spring Security.
- **Service** contiene los servicios necesarios para hacer consultas e inserciones en la base de datos.

Esquema de entidades:

Cada clase equivale a una tabla en base de datos, para indicarlo, se utiliza la anotación @Entity y @Table para indicar el nombre de la tabla.



Se hará un recorrido por todas aquellas entidades que contengan anotaciones de Spring para explicar cada una de las anotaciones.

```
@Entity  
 @Table(name = "orders")  
 public class Order implements Serializable {  
     @Id  
     @GeneratedValue(strategy = GenerationType.AUTO)  
     private Integer id;  
     @DateTimeFormat(pattern = "yyyy-MM-dd")  
     @Temporal(TemporalType.DATE)  
     private Date date;  
     private String status;  
     @ManyToMany(fetch = FetchType.EAGER)  
     @JoinTable(name = "order_details", joinColumns = @JoinColumn(name = "order_id"),  
               inverseJoinColumns = @JoinColumn(name = "product_id"))  
     private List<Product> products;  
  
     @ManyToOne  
     @JoinColumn(name="user_id", nullable=false)  
     private User user;
```

- La anotación `@Id` en conjunto con `@GeneratedValue` se utilizan para indicar qué campo servirá de id y qué tipo de generación tendrá.
- La anotación `@Temporal` acompañada de `@DateTimeFormat` se utiliza para el tipo date así como para especificar su patrón.
- La anotación `@ManyToMany` indica una relación de tipo n:m, viene acompañada de una variable fetch la cual especifica como se hará la actualización. Seguido de esta anotación se encuentra `@JoinTable`, en esta anotación se especifica la tabla de union y sus correspondientes columnas.
- La anotación `@ManyToOne` indica una relación tipo muchos a uno, en la que con `@JoinColumn` se indica la clave foránea.
- La anotación `@NotNull` indica que el valor no puede ser nulo y `@Enumerated` que es un tipo enumerado.

```
    private int id;  
    @NotNull  
    @Enumerated(EnumType.STRING)  
    private NameRol nameRol;
```

Consultas realizadas a base de datos:

```
public interface ProductRepository extends JpaRepository<Product, Integer> {
    Optional<Product> findByName(String nombre);
    boolean existsByName(String nombre);
}
```

```
@Repository
public interface RolRepository extends JpaRepository<Rol, Integer> {
    Optional<Rol> findByNameRol(NameRol nameRol);
}
```

```
@Repository
public interface UsersRepository extends JpaRepository<User, Integer> {
    Optional<User> findByUsername(String nombreUsuario);
    boolean existsByUsername(String nombreUsuario);
    boolean existsByEmail(String email);
```

```
@Repository
public interface UserInfoRepository extends JpaRepository<UserInfo, Integer> {
    @Query(" SELECT u FROM UserInfo u WHERE u.user.id = (SELECT id FROM User WHERE username = ?1)")
    UserInfo findByUser(String user);
}
```

Gracias al framework, no es necesario montar las queries, en el caso de que se quiera implementar una query personalizada se utiliza `@Query` como se puede observar en una de las imágenes.

Los endpoints de los diferentes RestController:

Para indicar que es un RestController se utiliza la anotación @RestController y con @RequestMapping se indica la ruta, mediante esta ruta se pueden pasar parámetros con “{}”, también se pueden observar los servicios usados.

```
@RestController
@RequestMapping("/order")
@CrossOrigin(origins = "*")
public class OrderController {

    @Autowired
    OrdersServiceImpl ordersService;

    @Autowired
    UserService userService;
```

### order/create

Recibe un OrderDto del front con su usuario y hace su pedido guardando en la base de datos, si en el dto no viene un usuario, no se guardará el pedido.

```
@PostMapping("/create")
public ResponseEntity<?> create(@RequestBody OrderDto orderDto){
    if(orderDto.getUser()==null)
        return new ResponseEntity(new Message(mensaje: "You are not logged in"), HttpStatus.BAD_REQUEST);
    if(orderDto.getProducts()==null)
        return new ResponseEntity(new Message(mensaje: "There are no products"), HttpStatus.BAD_REQUEST);
    User user = userService.getByUsername(orderDto.getUser()).get();
    Order order = new Order(orderDto.getDate(), orderDto.getStatus(), orderDto.getProducts(), user);
    ordersService.save(order);
    return new ResponseEntity(new Message(mensaje: "Order created"), HttpStatus.OK);
}
```

## product/list

Retorna una lista de los productos de la base de datos

```
@GetMapping("/list")
public ResponseEntity<List<Product>> list(){
    List<Product> list = productServiceImpl.list();
    return new ResponseEntity(list, HttpStatus.OK);
}
```

## product/detail

Retorna los detalles de un producto, existen dos versiones, una que busca por id y otra que realiza la búsqueda por nombre

```
@GetMapping("/detail/{id}")
public ResponseEntity<Product> getById(@PathVariable("id") int id){
    if(!productServiceImpl.existsById(id))
        return new ResponseEntity(new Message( mensaje: "Does not exist"), HttpStatus.NOT_FOUND);
    Product product = productServiceImpl.getOne(id).get();
    return new ResponseEntity(product, HttpStatus.OK);
}
```

## product/create

Crea un producto que viene de un dto del front, el rol Admin es requerido.

```
@PreAuthorize("hasRole('ADMIN')")
@PostMapping("/create")
public ResponseEntity<?> create(@RequestBody ProductDto productDto){
    if(StringUtils.isBlank(productDto.getName()))
        return new ResponseEntity(new Message( mensaje: "Name is required"), HttpStatus.BAD_REQUEST);
    if(productDto.getPrice()==null || productDto.getPrice()<0 )
        return new ResponseEntity(new Message( mensaje: "Price should be greater than 0"), HttpStatus.BAD_REQUEST);
    if(productServiceImpl.existsByName(productDto.getName()))
        return new ResponseEntity(new Message( mensaje: "Name already exists"), HttpStatus.BAD_REQUEST);
    Product product = new Product(productDto.getName(), productDto.getPrice(), productDto.getImage());
    productServiceImpl.save(product);
    return new ResponseEntity(new Message( mensaje: "Product created"), HttpStatus.OK);
}
```

## product/update/{id}

Realiza un update por id, el cual viene dado por variable y sustituye los valores por los del dto que llega del front, el rol Admin es requerido.

```
@PreAuthorize("hasRole('ADMIN')")
@PutMapping("update/{id}")
public ResponseEntity<?> update(@PathVariable("id") int id, @RequestBody ProductDto productDto){
    if(!productServiceImpl.existsById(id))
        return new ResponseEntity(new Message( mensaje: "Does not exist"), HttpStatus.NOT_FOUND);
    if(productServiceImpl.existsByName(productDto.getName()) && productServiceImpl.getByName(productDto.getName()).getId() != id)
        return new ResponseEntity(new Message( mensaje: "Name already exists"), HttpStatus.BAD_REQUEST);
    if(StringUtils.isBlank(productDto.getName()))
        return new ResponseEntity(new Message( mensaje: "Name is required"), HttpStatus.BAD_REQUEST);
    if(productDto.getPrice()==null || productDto.getPrice()<0 )
        return new ResponseEntity(new Message( mensaje: "Price should be greater than 0"), HttpStatus.BAD_REQUEST);

    Product product = productServiceImpl.getOne(id).get();
    product.setName(productDto.getName());
    product.setPrice(productDto.getPrice());
    productServiceImpl.save(product);
    return new ResponseEntity(new Message( mensaje: "Product updated"), HttpStatus.OK);
}
```

## product/delete/{id}

Realiza un delete por id de producto.

```
@PreAuthorize("hasRole('ADMIN')")
@DeleteMapping("delete/{id}")
public ResponseEntity<?> delete(@PathVariable("id") int id){
    if(!productServiceImpl.existsById(id))
        return new ResponseEntity(new Message( mensaje: "Does not exist"), HttpStatus.NOT_FOUND);
    productServiceImpl.delete(id);
    return new ResponseEntity(new Message( mensaje: "Product deleted"), HttpStatus.OK);
}
```

## userinfo/detailname/{username}

Obtiene la información asignada al usuario pasado por variable, si este no tiene, devuelve un toast, solicitando que añada los datos.

```
@GetMapping("detailname/{username}")
public ResponseEntity<UserInfo> get(@PathVariable("username") String username){
    UserInfo userInfo = userInfoService.getInfoByUsername(username);
    if(userInfo==null)
        return new ResponseEntity(new Message( mensaje: "Introduce data"), HttpStatus.NOT_FOUND);
    userInfo = userInfoService.getInfoByUsername(username);
    return new ResponseEntity(userInfo, HttpStatus.OK);
}
```

## userinfo/update/{username}

Realiza un update de la información de usuario, solo introduciendo los campos que vienen en el dto. Si el usuario no tiene información, se crea una nueva línea en la BDD.

```
@PutMapping("update/{username}")
public ResponseEntity<?> update(@PathVariable("username") String username, @RequestBody UserInfoDto userInfoDto){
    User user = userService.getByUsername(username).get();
    UserInfo userInfo = userInfoService.getInfoByUsername(username);
    if (userInfo==null)
        userInfo = new UserInfo();
    if (userInfoDto.getAddressLine()!=null&&!userInfoDto.getAddressLine().isEmpty())
        userInfo.setAddressLine(userInfoDto.getAddressLine());
    if (userInfoDto.getCity()!=null&&!userInfoDto.getCity().isEmpty())
        userInfo.setCity(userInfoDto.getCity());
    if (userInfoDto.getPhone()!=null)
        userInfo.setPhone(userInfoDto.getPhone());
    if (userInfoDto.getState()!=null)
        userInfo.setState(userInfoDto.getState());
    if (userInfoDto.getCountry()!=null&&!userInfoDto.getCountry().isEmpty())
        userInfo.setCountry(userInfoDto.getCountry());
    if (userInfoDto.getPostalCode()!=null)
        userInfo.setPostalCode(userInfoDto.getPostalCode());
    if (userInfo.getUser()==null)
        userInfo.setUser(user);
    userInfoService.save(userInfo);
    return new ResponseEntity(new Message( mensaje: "Updated"), HttpStatus.OK);
}
```

## user/new

Creación de usuarios

```
@PostMapping("new")
public ResponseEntity<?> nuevo(@Valid @RequestBody NewUser newUser, BindingResult bindingResult){
    if(bindingResult.hasErrors())
        return new ResponseEntity(new Message( mensaje: "campos mal puestos o email inválido"), HttpStatus.BAD_REQUEST);
    if(userService.existsByUsername(newUser.getUsername()))
        return new ResponseEntity(new Message( mensaje: "ese nombre ya existe"), HttpStatus.BAD_REQUEST);
    if(userService.existsByEmail(newUser.getEmail()))
        return new ResponseEntity(new Message( mensaje: "ese email ya existe"), HttpStatus.BAD_REQUEST);
    User user =
        new User(newUser.getName(), newUser.getUsername(), newUser.getEmail(),
                passwordEncoder.encode(newUser.getPassword()));
    Set<Role> roles = new HashSet<>();
    roles.add(rolService.getByRolNombre(NameRol.ROLE_USER).get());
    if(newUser.getRoles().contains("admin"))
        roles.add(rolService.getByRolNombre(NameRol.ROLE_ADMIN).get());
    user.setRoles(roles);
    userService.save(user);
    return new ResponseEntity(new Message( mensaje: "usuario guardado"), HttpStatus.CREATED);
}
```

## user/login

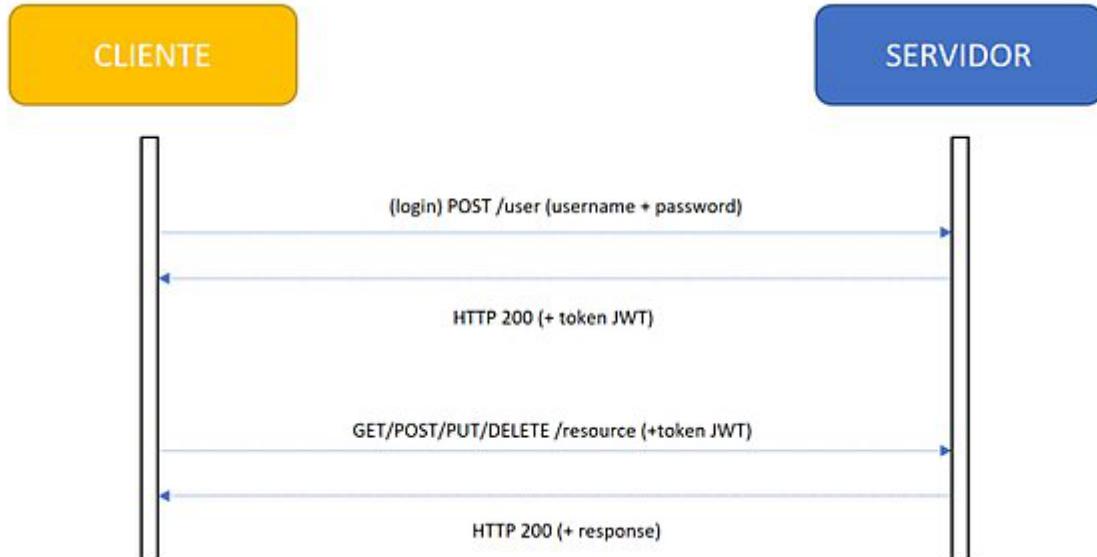
Autenticación de usuarios, aquí se realiza el login. Se puede observar el uso del servicio que crea un JWT que se encarga de mantener los permisos y asegurar la comunicación entre back y front.

```
@PostMapping(value="/Login")
public ResponseEntity<JwtDto> login(@Valid @RequestBody UserLogin userLogin, BindingResult bindingResult){
    if(bindingResult.hasErrors())
        return new ResponseEntity(new Message("campos mal puestos"), HttpStatus.BAD_REQUEST);
    Authentication authentication =
        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(userLogin.getUsername(), userLogin.getPassword()));
    SecurityContextHolder.getContext().setAuthentication(authentication);
    String jwt = jwtProvider.generateToken(authentication);
    UserDetails userDetails = (UserDetails)authentication.getPrincipal();
    JwtDto jwtDto = new JwtDto(jwt, userDetails.getUsername(), userDetails.getAuthorities());
    return new ResponseEntity(jwtDto, HttpStatus.OK);
}
```

## Explicación de Security y de JWT:

Para restringir el acceso a ciertas funciones que requieren administrador y para mantener los permisos y la conexión entre back y front se utiliza tanto JWT como security, en el siguiente apartado se explica su implementación en el proyecto.

## Funcionamiento de JWT:



Para poder usar los endpoints de nuestra aplicación, el cliente debe tener un token el cual se obtiene mediante el login.

```
@Component
public class JwtProvider {
    private final static Logger logger = LoggerFactory.getLogger(JwtProvider.class);

    @Value("secret")
    private String secret;

    @Value("36000")
    private int expiration;

    public String generateToken(Authentication authentication){
        PrincipalUser principalUser = (PrincipalUser) authentication.getPrincipal();
        return Jwts.builder().setSubject(principalUser.getUsername())
            .setIssuedAt(new Date())
            .setExpiration(new Date(new Date().getTime() + expiration * 1000))
            .signWith(SignatureAlgorithm.HS512, secret)
            .compact();
    }
}
```

En provider se genera el token, además de la utilización de la anotación @Component, el cual registra un bean dentro del framework para así poder hacer uso de él. En la misma clase existe un método para recuperar el usuario por el token y para validar el token.

```
public String getNombreUsuarioFromToken(String token){
    return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().getSubject();
}

public boolean validateToken(String token){
    try {
        Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
        return true;
    }catch (MalformedJwtException e){
        logger.error("token mal formado");
    }catch (UnsupportedJwtException e){
        logger.error("token no soportado");
    }catch (ExpiredJwtException e){
        logger.error("token expirado");
    }catch (IllegalArgumentException e){
        logger.error("token vacio");
    }catch (SignatureException e){
        logger.error("fail en la firma");
    }
    return false;
}
```

En la clase JwtEntryPoint se implementa la interfaz AuthenticationEntryPoint que implementa el método commence, el cual se encarga de gestionar los encabezados para comprobar que están autenticados.

```
@Component
public class JwtEntryPoint implements AuthenticationEntryPoint {

    private final static Logger logger = LoggerFactory.getLogger(JwtEntryPoint.class);

    @Override
    public void commence(HttpServletRequest req, HttpServletResponse res, AuthenticationException e) throws IOException, ServletException {
        logger.error("fail en el método commence");
        res.sendError(HttpStatus.SC_UNAUTHORIZED, msg: "no autorizado");
    }
}
```

En el filter se consigue el usuario si el token ha sido validado. Se utiliza la clase UserDetails, que es la clase de usuarios de Spring Security para poder hacer un loadByUsername. El usuario se carga en usuarioPrincipal el cual tiene unos authorities que le permiten acceder o no a los distintos puntos.

```
public class JwtTokenFilter extends OncePerRequestFilter {

    private final static Logger logger = LoggerFactory.getLogger(JwtTokenFilter.class);

    @Autowired
    JwtProvider jwtProvider;

    @Autowired
    UserDetailsServiceImpl userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain filterChain) throws ServletException, IOException {
        try {
            String token = getToken(req);
            if(token != null && jwtProvider.validateToken(token)){
                String nombreUsuario = jwtProvider.getNombreUsuarioFromToken(token);
                UserDetails userDetails = userDetailsService.loadUserByUsername(nombreUsuario);

                UsernamePasswordAuthenticationToken auth =
                    new UsernamePasswordAuthenticationToken(userDetails, credentials: null, userDetails.getAuthorities());
                SecurityContextHolder.getContext().setAuthentication(auth);
            }
        } catch (Exception e){
            logger.error("fail en el método doFilter " + e.getMessage());
        }
        filterChain.doFilter(req, res);
    }

    private String getToken(HttpServletRequest request){
        String header = request.getHeader( name: "Authorization");
        if(header != null && header.startsWith("Bearer"))
            return header.replace( target: "Bearer ", replacement: "");
        return null;
    }
}
```

Clase de configuración de seguridad, se puede observar cómo se usan los servicios de userDetail(usuario de Spring Security) y el jwt. También se declaran los beans que se utilizarán, todos los beans son reutilizables y serializables.

Método con las opciones de configuración de Security, se declaran las URL que son accesibles por todos y cuando es necesario que se autentifique.

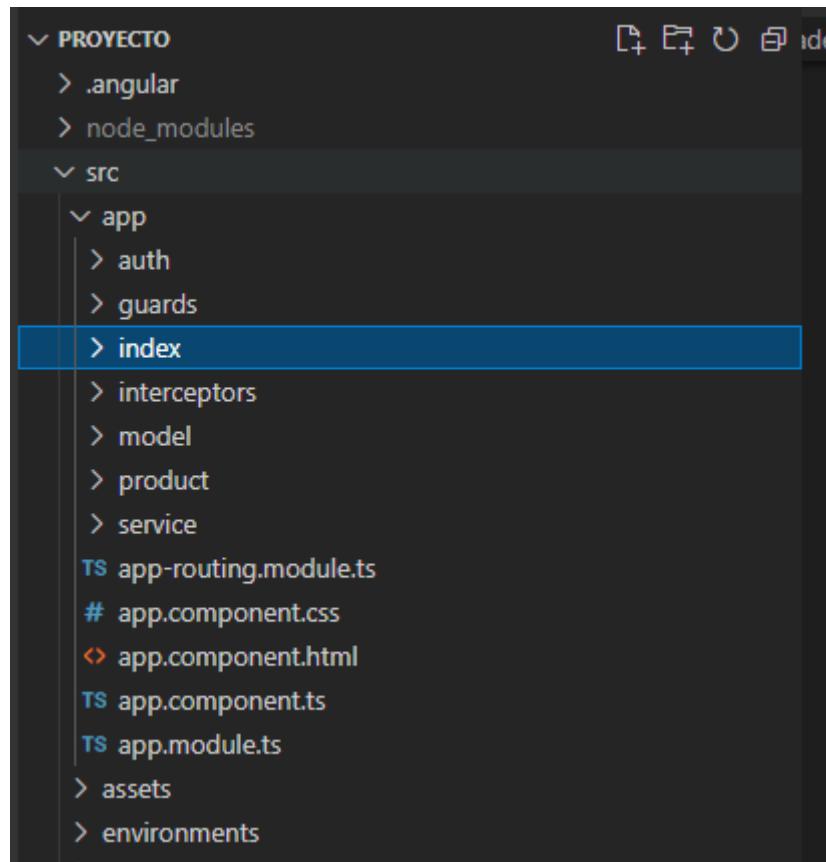
```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
    http.cors().and().csrf().disable()  
        .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry  
            .antMatchers( ...antPatterns: "/auth/**").permitAll()  
            .anyRequest().authenticated()  
        .and() HttpSecurity  
            .exceptionHandling().authenticationEntryPoint(jwtEntryPoint) ExceptionHandlingConfigurer<HttpSecurity>  
            .and() HttpSecurity  
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);  
    http.addFilterBefore(jwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);  
}
```

## Frontend

### ANGULAR

Las aplicaciones angular se componen de componentes, los cuales tienen un HTML con un archivo typeScript asociado que controla su funcionalidad.

La estructura del proyecto es la siguiente:



Las carpetas que contienen componentes son:

- Auth: Componentes de login, registro y información de usuario
- Index: Componente de la página inicial, en la cual se llama toda la funcionalidad
- Product: Componentes asociados a la creación y modificación de productos de la tienda
- Model: Objetos de todas las entidades usadas, funcionan como un DTO para intercambiar información con el backend.
- Service: Servicios que realizan las peticiones al backend en el frontend.

## app.module

Se indican todos los componentes usados de la aplicación.

```
@NgModule({
  declarations: [
    AppComponent,
    ListProductComponent,
    DetailProductComponent,
    NewProductComponent,
    EditProductComponent,
    LoginComponent,
    RegisterComponent,
    IndexComponent,
    UserInfoComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    ToastrModule.forRoot(),
    HttpClientModule,
    FormsModule
  ],
  providers: [interceptorProvider],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

## app.component

Componente principal, mediante router podemos hacer que su contenido cambie en función de la ruta.

```
src > app > app.component.html > router-outlet
1   <router-outlet></router-outlet>
```

## app-routing

Se definen todas las rutas y los permisos necesarios para cargarlos

```
const routes: Routes = [
  { path: '', component: IndexComponent },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'userinfo', component: UserInfoComponent, canActivate: [guard], data: { expectedRol: ['admin', 'user'] } },
  { path: 'list', component: ListProductComponent, canActivate: [guard], data: { expectedRol: ['admin', 'user'] } },
  { path: 'detail/:id', component: DetailProductComponent, canActivate: [guard], data: { expectedRol: ['admin', 'user'] } },
  { path: 'new', component: NewProductComponent, canActivate: [guard], data: { expectedRol: ['admin'] } },
  { path: 'edit/:id', component: EditProductComponent, canActivate: [guard], data: { expectedRol: ['admin'] } },
  { path: '**', redirectTo: '', pathMatch: 'full' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

## prod-interceptor.service

Con este servicio se intercepta el token del encabezado

```
@Injectable({
  providedIn: 'root'
})
export class ProdInterceptorService implements HttpInterceptor {

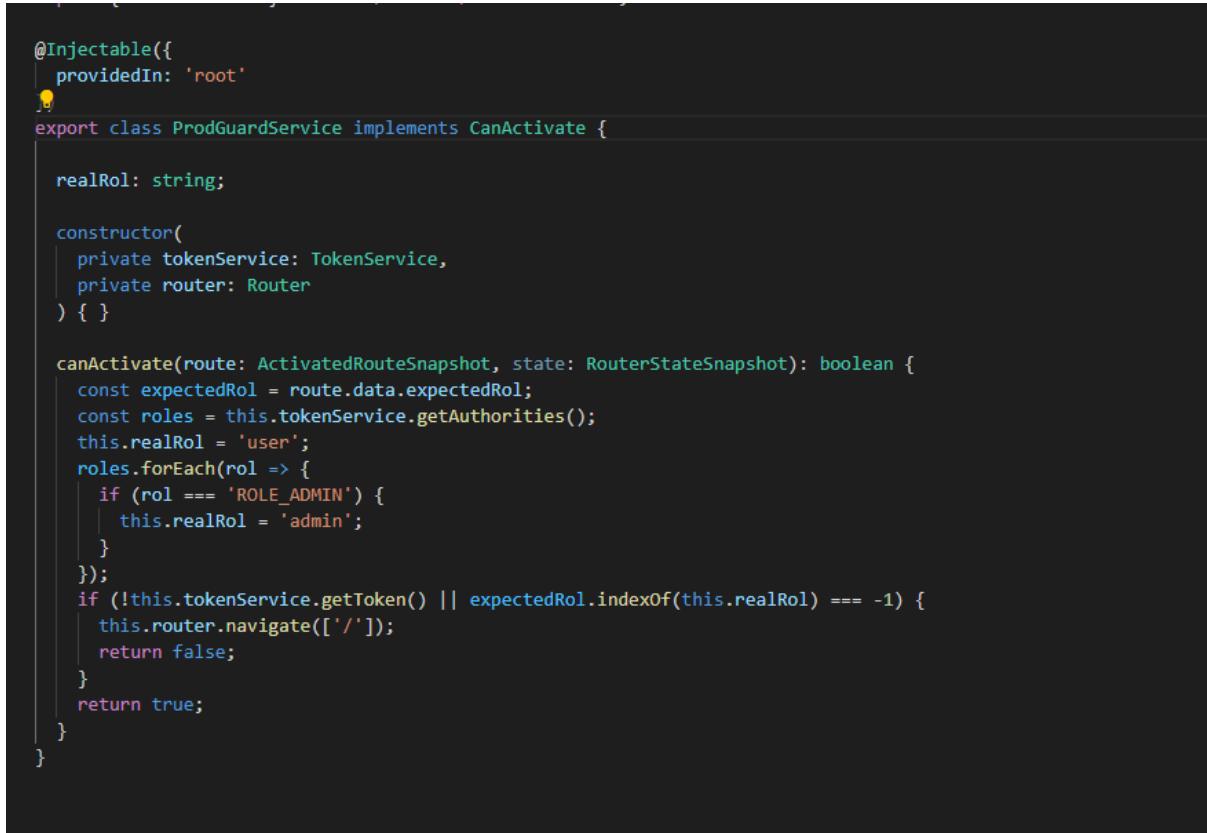
  constructor(private tokenService: TokenService) { }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    let intReq = req;
    const token = this.tokenService.getToken();
    if (token != null) {
      intReq = req.clone({ headers: req.headers.set('Authorization', 'Bearer ' + token)});
    }
    return next.handle(intReq);
  }
}

export const interceptorProvider = [{provide: HTTP_INTERCEPTORS, useClass: ProdInterceptorService, multi: true}];
```

## prod-guard.service

En esta clase se convierten los roles de Security para que en el routing modules sea posible identificar quién tiene acceso.



```
@Injectable({
  providedIn: 'root'
})
export class ProdGuardService implements CanActivate {

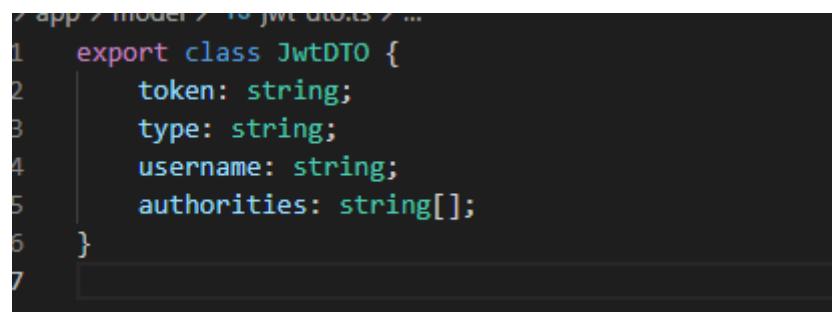
  realRol: string;

  constructor(
    private tokenService: TokenService,
    private router: Router
  ) { }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    const expectedRol = route.data.expectedRol;
    const roles = this.tokenService.getAuthorities();
    this.realRol = 'user';
    roles.forEach(rol => {
      if (rol === 'ROLE_ADMIN') {
        this.realRol = 'admin';
      }
    });
    if (!this.tokenService.getToken() || expectedRol.indexOf(this.realRol) === -1) {
      this.router.navigate(['/']);
      return false;
    }
    return true;
  }
}
```

## jwtDto

El dto que trae el token del back.



```
app > model > jwt.dto.ts > ...
1  export class JwtDTO {
2    token: string;
3    type: string;
4    username: string;
5    authorities: string[];
6  }
7
```

Constantes que se guardarán en sesión con la información del usuario logueado y su token.

```
const TOKEN_KEY = 'AuthToken';
const USERNAME_KEY = 'AuthUserName';
const AUTHORITIES_KEY = 'AuthAuthorities';
```

## TokenService

Servicio que maneja la información del token y lo guarda en sesión para su posterior uso.

```
@Injectable({
  providedIn: 'root'
})
export class TokenService {

  roles: Array<string> = [];

  constructor() { }

  public setToken(token: string): void {
    window.sessionStorage.removeItem(TOKEN_KEY);
    window.sessionStorage.setItem(TOKEN_KEY, token);
  }

  public getToken(): string {
    return sessionStorage.getItem(TOKEN_KEY);
  }

  public setUserName(username: string): void {
    window.sessionStorage.removeItem(USERNAME_KEY);
    window.sessionStorage.setItem(USERNAME_KEY, username);
  }

  public getUserName(): string {
    return sessionStorage.getItem(USERNAME_KEY);
  }

  public setAuthorities(authorities: string[]): void {
    window.sessionStorage.removeItem(AUTHORITIES_KEY);
    window.sessionStorage.setItem(AUTHORITIES_KEY, JSON.stringify(authorities));
  }

  public getAuthorities(): string[] {
    this.roles = [];
    if (sessionStorage.getItem(AUTHORITIES_KEY)) {
      JSON.parse(sessionStorage.getItem(AUTHORITIES_KEY)).forEach(authority => {
        this.roles.push(authority.authority);
      });
    }
    return this.roles;
  }
}
```

El resto de servicios son de conexión con el backend, por lo que su funcionalidad es similar. Solo se distinguen en el tipo de dato que envían o reciben según sea get, post o put. Aquí se observa un ejemplo de servicio que trae y envía datos:

Se realiza a través de la clase httpClient que es la que hace las peticiones, los métodos deben ser de tipo observable dado a que se debe esperar respuesta.

```
export class ProductService {  
  ProductURL = 'http://localhost:8080/product/';  
  
  constructor(private httpClient: HttpClient) { }  
  
  public list(): Observable<Product[]> {  
    return this.httpClient.get<Product[]>(this.ProductURL + 'list');  
  }  
  
  public detail(id: number): Observable<Product> {  
    return this.httpClient.get<Product>(this.ProductURL + `detail/${id}`);  
  }  
  
  public detailName(name: string): Observable<Product> {  
    return this.httpClient.get<Product>(this.ProductURL + `detailname/${name}`);  
  }  
  
  public save(Product: Product): Observable<any> {  
    return this.httpClient.post<any>(this.ProductURL + 'create', Product);  
  }  
  
  public update(id: number, Product: Product): Observable<any> {  
    return this.httpClient.put<any>(this.ProductURL + `update/${id}`, Product);  
  }  
  
  public delete(id: number): Observable<any> {  
    return this.httpClient.delete<any>(this.ProductURL + `delete/${id}`);  
  }  
}
```

En la funcionalidad de los componentes se encuentra:

### Index.component.ts

Se declaran las variables que se cargarán en la vista, así como la lista de los productos y el carrito. Además se importan los servicios usados para recuperar y enviar los datos. También se implementa la interfaz OnInit para tener el método que se ejecutará con la carga del componente.

```
  ,
export class IndexComponent implements OnInit {

  isLoggedIn = false;
  username = '';
  products: Product[] = [];
  cartProducts: Product[] = [];
  roles: string[];
  isAdmin = false;
  total: number;
  order : Order;
  euro: boolean = false;

  constructor(private productService: ProductService,
    private orderService: OrderService,
    private toastr: ToastrService,
    private tokenService: TokenService
  ) { }
```

Se comprueba que tenga la sesión iniciada, así como sus permisos, y se cargan los productos.

```
ngOnInit() {
  if (this.tokenService.getToken()) {
    this.isLoggedIn = true;
    this.username = this.tokenService.getUserName();
  } else {
    this.isLoggedIn = false;
    this.username = '';
  }
  if (this.tokenService.getToken()) {
    this.isLoggedIn = true;
  } else {
    this.isLoggedIn = false;
  }

  this.loadProducts((property) IndexComponent.tokenService: TokenService) {
    this.roles = this.tokenService.getAuthorities();
    this.roles.forEach(rol => {
      if (rol === 'ROLE_ADMIN') {
        this.isAdmin = true;
      }
    });
  }
}
```

Método de carga de los productos y el método que realiza el pedido con los productos metidos en el array cardProducts.

```
loadProducts(): void {
  this.productService.list().subscribe(
    data => {
      this.products = data;
    },
    err => {
      console.log(err);
    }
  );
}

buy(): void {
  this.order = new Order(new Date(), "Processing", this.cartProducts, this.username);
  this.orderService.save(this.order).subscribe(
    data => {
      this.toastr.success('Compra efectuada', 'OK', {
        timeOut: 3000, positionClass: 'toast-top-center'
      });
    },
    );
  this.cartProducts = [];
}
```

Métodos que gestionan el carrito y la variable cardProducts que es la que se enviará al backend para hacer el pedido con los productos seleccionados.

```
addToCart(cartProducts:Product): void {
  this.cartProducts.push(cartProducts);
  this.totalCart();
  this.euro = true;
}

deleteFromCart(cartProducts:Product): void {
  this.cartProducts = this.cartProducts.filter(obj => obj !== cartProducts);
  this.totalCart();
}

totalCart() {
  this.total = 0;
  for(var i = 0; i < this.cartProducts.length; i++) {
    this.total = this.total + this.cartProducts[i].price;
  }
}
```

En la vista podemos hacer que se carguen elementos en función de variables de typescript.

```
</li>
<li class="dropdown" *ngIf="isAdmin">
  <a data-toggle="dropdown" >
```

Con ngFor es posible hacer que se repitan elementos en función de una variable, así se imprimen por pantalla todos los productos, siendo así reactivo a la información que proviene del typeScript. El carrito también funciona con ngFor.

```
<div class="col-md-3 col-sm-4" *ngFor="let product of products">
  <div class="single-new-arrival">
    <div class="single-new-arrival-bg">
      
      <div class="new-arrival-cart" (click)="addToCart(product)">
        <p>
          <span class="lnr lnr-cart"></span>
          <a href="#">add <span>to </span> cart</a>
        </p>
      </div>
    </div>
    <h4><a>{{product.name}}</a></h4>
    <p class="arrival-product-price">{{product.price}}€</p>
  </div>

```

## LoginComponent

En este método se recoge el usuario en el dto loginUser que se construye con variables introducidas en la vista (this.username y password). Hace una petición al back y después mediante el tokenService setea los valores del token de autenticación. Si ha tenido éxito lanza un toast con un mensaje y se realiza una redirección para que vuelva a cargar la página.

```
export class LoginComponent implements OnInit {

  isLoggedIn = false;
  isLoginFail = false;
  loginUser: LoginUser;
  username: string;
  password: string;
  roles: string[] = [];
  errMsj: string;

  constructor(
    private tokenService: TokenService,
    private authService: AuthService,
    private router: Router,
    private toastr: ToastrService
  ) { }
```

```
onLogin(): void {
  this.loginUser = new LoginUser(this.username, this.password);
  this.authService.login(this.loginUser).subscribe(
    data => {
      this.isLoggedIn = true;

      this.tokenService.setToken(data.token);
      this.tokenService.setUserName(data.username);
      this.tokenService.setAuthorities(data.authorities);
      this.roles = data.authorities;
      this.toastr.success('Welcome ' + data.username, 'OK', {
        timeOut: 3000, positionClass: 'toast-top-center'
      });
      window.location.href = "http://localhost:4200";
    },
    err => {
      this.isLoggedIn = false;
      this.errMsj = err.error.message;
      this.toastr.error(this.errMsj, 'Fail', {
        timeOut: 3000, positionClass: 'toast-top-center',
      });
    }
  );
}
```

## RegisterComponent

Es similar al login pero esta vez se crea un dto diferente y la petición es para la creación de un nuevo usuario.

```
onRegister(): void {
  this.newUser = new NewUser(this.name, this.username, this.email, this.password);
  this.authService.new(this.newUser).subscribe(
    data => {
      this.toastr.success('Account Created', 'OK', {
        timeOut: 3000, positionClass: 'toast-top-center'
      });

      this.router.navigate(['/login']);
    },
    err => {
      this.errMsg = err.error.mensaje;
      this.toastr.error(this.errMsg, 'Fail', {
        timeOut: 3000, positionClass: 'toast-top-center',
      });
    }
  );
}
```

## InfoComponent

Se comprueba el usuario y trae su información del backend, si no hay, muestra un toast.

```
ngOnInit() {
  if (this.tokenService.getToken()) {
    this.isLoggedIn = true;
    this.username = this.tokenService.getUserName();
  } else {
    this.isLoggedIn = false;
    this.username = '';
  }

  this.userInfoService.get(this.username).subscribe(
    data => {
      this.phone = data.phone;
      this.addressLine = data.addressLine;
      this.city = data.city;
      this.country= data.country;
      this.state = data.state;
      this.postalCode = data.postalCode;
    },
    err => {
      this.toastr.error(err.error.mensaje, 'Fail', {
        timeOut: 3000, positionClass: 'toast-top-center',
      });
    }
  );
}
```

Hace una actualización de la información de usuario. El dto es UserInfo

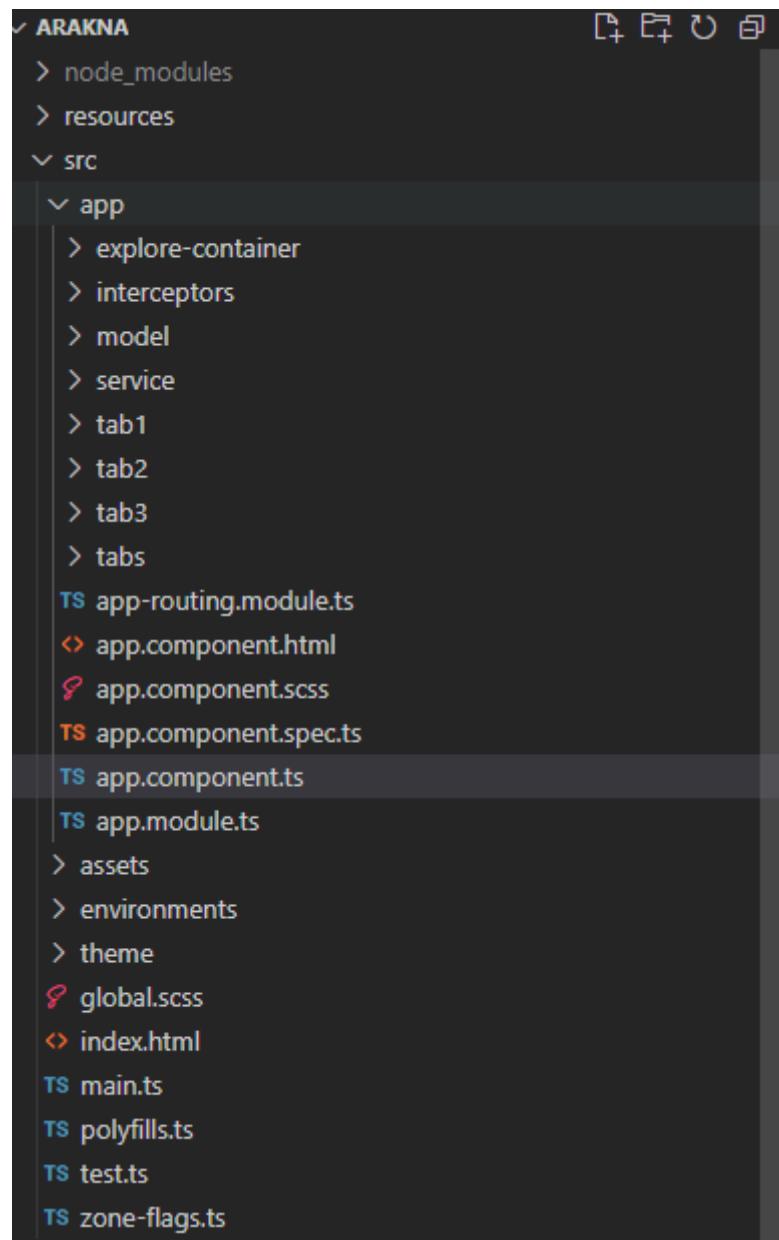
```
onUpdate(): void {
  const userInfo = new UserInfo(this.phone, this.addressLine, this.city, this.state, this.postalCode, this.country);
  this.userInfoService.update(this.username,userInfo).subscribe(
    data => {
      this.toastr.success('Product Created', 'OK', {
        timeOut: 3000, positionClass: 'toast-top-center'
      });
      this.router.navigate(['/']);
    },
    err => {
      this.toastr.error(err.error.mensaje, 'Fail', {
        timeOut: 3000, positionClass: 'toast-top-center',
      });
    }
  );
}
```

## ProductComponent

Los componentes de producto funcionan de una manera muy similar, hace peticiones para obtener y enviar productos y los mete en html mediante typeScript con ngFor.

## IONIC

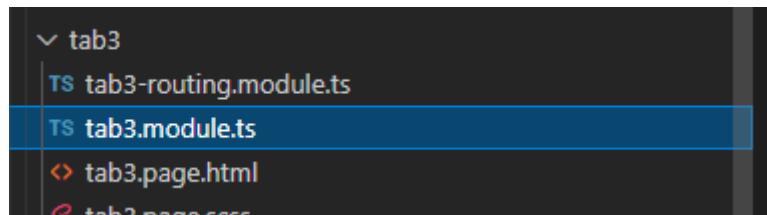
Ionic es un framework similar a Angular que también contiene, de la misma manera, componentes y typeScript. De esta manera la mayor parte de la funcionalidad de angular se puede llevar a Ionic. La aplicación Ionic usa los mismos servicios y autentificación que la aplicación de Angular, pero en el caso de la app móvil solo se podrá ver un catálogo de productos que proceden del backend, no se podrá realizar compras.



El gran cambio con respecto a Angular es la manera en la que trata la vista.

```
<ion-content [fullscreen]="true" (click)="loadProducts()" >
  <ion-header collapse="condense">
    <ion-toolbar>
      <ion-title size="large">Shop </ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content fullscreen>
    <ion-card *ngFor="let product of products">
      
      <ion-card-header>
        <ion-card-title>{{product.name}}</ion-card-title>
      </ion-card-header>
      <ion-card-content>
        Price: {{product.price}}
      </ion-card-content>
    </ion-card>
  </ion-content>
</ion-content>
```

Los elementos deben ser de tipo ion para que sean tratados como vista de móvil. Además, cada componente que vaya a cargar debe tener su propio módulo para que se genere correctamente.



Funcionalidad en la app de ionic:

```
)  
export class Tab3Page implements OnInit {  
  products: Product[] = [];  
  
  constructor(private productService: ProductService)  
  {}  
  
  ngOnInit(): void {  
    this.loadProducts();  
  }  
  loadProducts(): void {  
    this.productService.list().subscribe(  
      data => {  
        this.products = data;  
      },  
      err => {  
        console.log(err);  
      }  
    );  
  }  
}
```

Vista reactiva en la app:

Como se puede apreciar se realiza de la misma manera que en el proyecto web pero con elementos de tipo ion.

```
<ion-header [translucent]="true">
  <ion-toolbar>
    <ion-title>
      Shop
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content [fullscreen]="true" (click)="loadProducts()">
  <ion-header collapse="condense">
    <ion-toolbar>
      <ion-title size="large">Shop </ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content fullscreen>
    <ion-card *ngFor="let product of products">
      
      <ion-card-header>
        <ion-card-title>{{product.name}}</ion-card-title>
      </ion-card-header>
      <ion-card-content>
        Price: {{product.price}}
      </ion-card-content>
    </ion-card>
  </ion-content>
</ion-content>
```