

## ➔ Crear nuestra aplicación con Spring Initializr

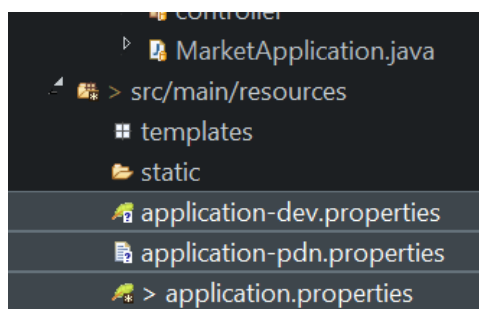
<https://start.spring.io/>

The screenshot shows the Spring Initializr web form with the following configuration:

- Project:** ☐ Maven Project, ☒ Gradle Project
- Language:** ☒ Java, ☐ Kotlin, ☐ Groovy
- Spring Boot:** ☐ 2.6.0 (SNAPSHOT), ☐ 2.6.0 (M2), ☐ 2.5.5 (SNAPSHOT), ☒ 2.5.4, ☐ 2.4.11 (SNAPSHOT), ☐ 2.4.10
- Project Metadata:**
  - Group: com.danicode
  - Artifact: market
  - Name: market
  - Description: API Springboot - security - jpa
  - Package name: com.danicode.market
  - Packaging: ☒ Jar, ☐ War
  - Java: ☐ 16, ☒ 11, ☐ 8
- Dependencies:** Spring Web (WEB) - Build web, including RESTful, a default embedded container.

## ➔ Configurar Spring Boot

Crear propiedades para desarrollar y para producción:



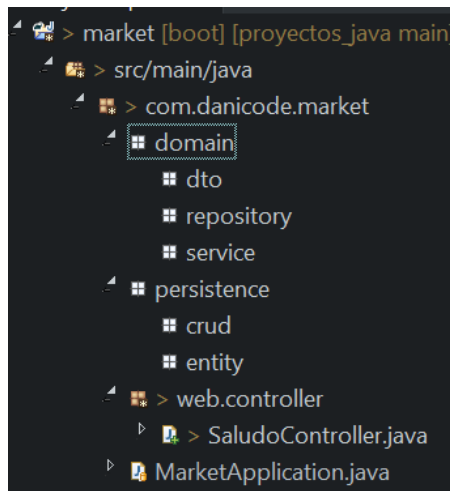
```
application.properties application-dev.properties
1 spring.profiles.active=dev
2 server.servlet.context-path=/market/api
```

```
application.properties application-dev.properties
1 server.port=8090
```

```
application.properties application-dev.properties application-pdn.properties
1 server.port=80
```

## → Crear la estructura del proyecto

# La estructura de nuestro proyecto



## → ¿Qué es JPA

- JPA es una especificación de Java (un estándar) para un framework ORM.
- Interactuar con las tablas de la base de datos en forma de objetos Java.
- Algunas de sus implementaciones son:
  - Hibernate
  - EclipseLink
  - TopLink
  - ObjectDB

## Anotaciones de JPA

- @Entity
- @Table
- @Column
- @Id & @EmbeddedId
- @GeneratedValue
- @OneToMany & @ManyToOne

## → Conocer qué es Spring Data

# Spring Data

- Es un proyecto que internamente contiene otros, nosotros usaremos el Spring Data JPA
- Optimización de tareas repetitivas
- Repositorios sin código con JpaRepository, CrudRepository & PagingAndSortingRepository
- Auditorías transparentes

Incluir Spring Data en el proyecto:

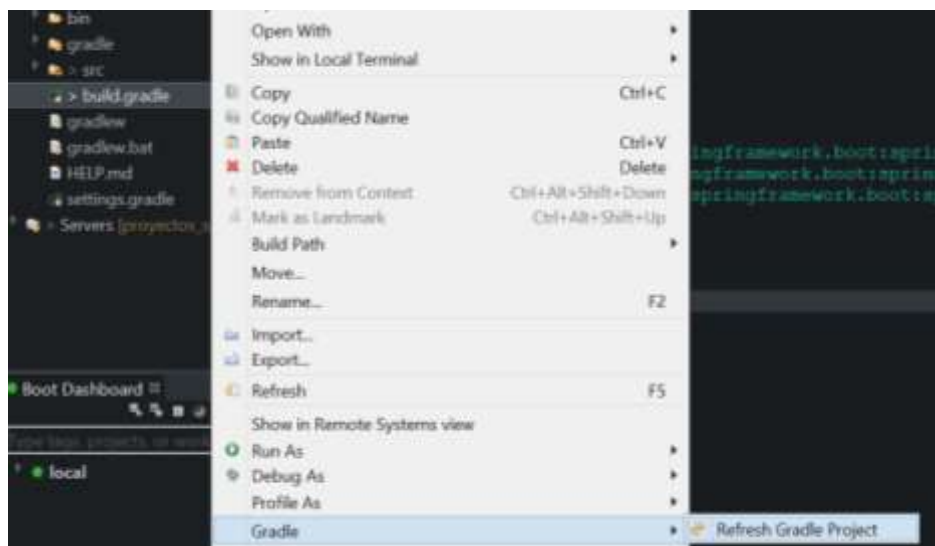
<https://mvnrepository.com/>

```
// https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa
implementation group: 'org.springframework.boot', name: 'spring-boot-starter-data-jpa',
version: '2.5.4'
```

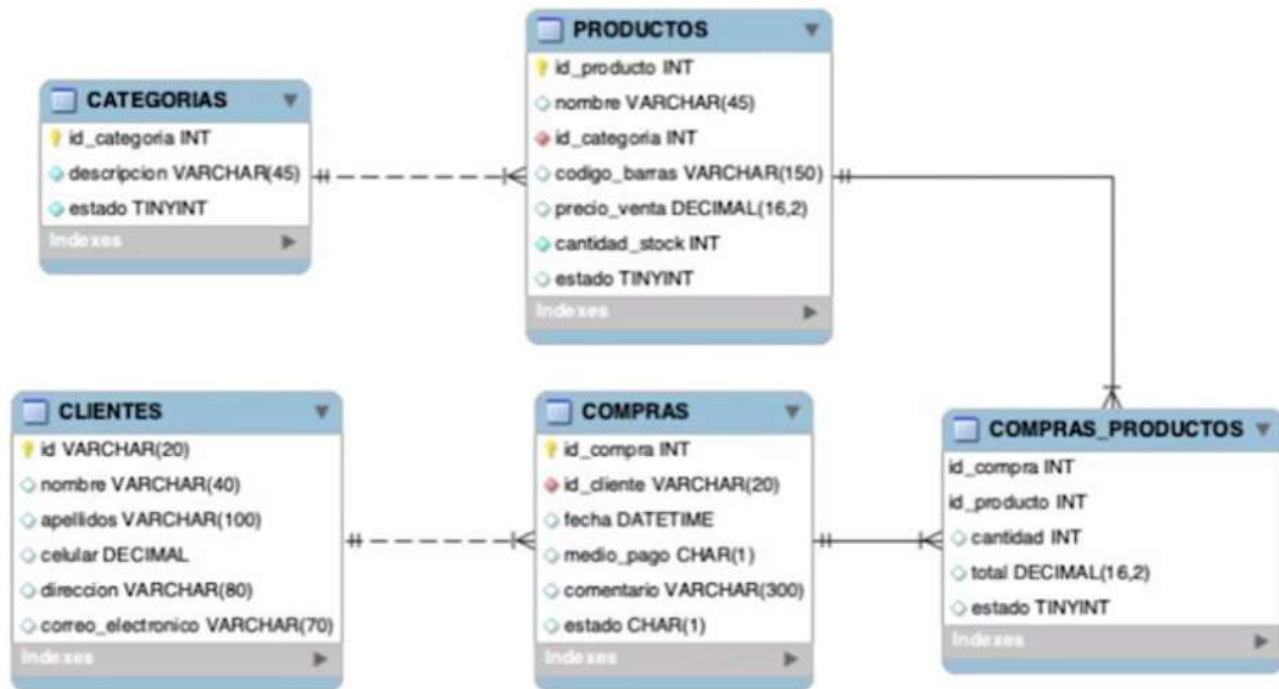
```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
```

```
build.gradle.kts
1 plugins {
2     id 'org.springframework.boot' version '2.5.4'
3     id 'io.spring.dependency-management' version '1.0.11.RELEASE'
4     id 'java'
5 }
6
7 group = 'com.danicode'
8 version = '0.0.1-SNAPSHOT'
9 sourceCompatibility = '11'
10
11 repositories {
12     mavenCentral()
13 }
14
15 dependencies {
16     implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
17     implementation 'org.springframework.boot:spring-boot-starter-web'
18     testImplementation 'org.springframework.boot:spring-boot-starter-test'
19 }
20
```

Refrescar gradle:



## ➔ Conectar la base de datos a nuestra aplicación



Conectar la BD a la aplicación (paquete solo en tiempo de ejecución:

```
runtimeOnly 'org.postgresql:postgresql'
```

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
    runtimeOnly 'org.postgresql:postgresql'  
}
```

En application-dev.properties y application-pdn.properties:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/market
```

```
spring.datasource.username=postgres
```

```
spring.datasource.password=admin
```

```
application.properties application-dev.properties  
1 server.port=8090  
2  
3 spring.datasource.url=jdbc:postgresql://localhost:5432/market  
4 spring.datasource.username=postgres  
5 spring.datasource.password=admin
```

➔Mapear las tablas como clases

```
@Entity
@Table(name = "productos")
public class Producto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_producto")
    private Integer idProducto; // Usar la clase Integer y no el primitivo int

    private String nombre;

    @Column(name = "id_categoria")
    private Integer idCategoria;

    @Column(name = "codigo_barras")
    private String codigoBarras;

    @Column(name = "precio_venta")
    private Double precioVenta;

    @Column(name = "cantidad_stock")
    private Integer cantidadStock;

    private Boolean estado;
```

```
@Entity
@Table(name = "compras")
public class Compra {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_compra")
    private Integer idCompra;

    @Column(name = "id_cliente")
    private Integer idCliente;

    private LocalDateTime fecha;

    @Column(name = "medio_pago")
    private String medioPago;

    private String comentario;

    private Boolean estado;
```

➔Crear Entity cuando su clave primaria es compuesta

La tabla compras\_productos tiene dos claves primarias:

compras\_productos

General Columns Advanced Constraints

Inherited from table(s) 

Select to inherit from

Columns

		Name	Data type
		id_compra	integer
		id_producto	integer

Para añadirlas, se debe crear una nueva clase que las contengan:

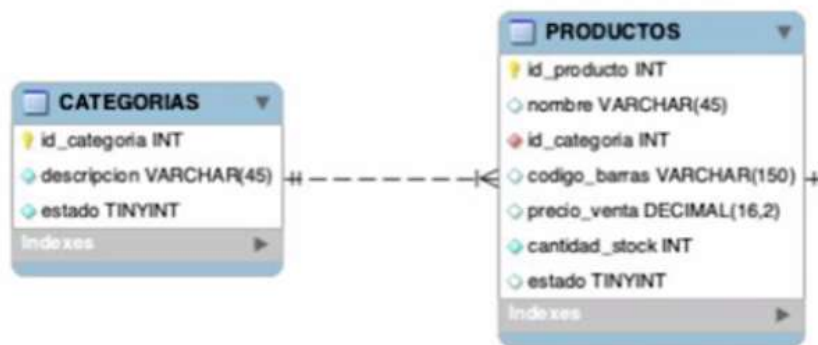
```
ComprasProducto.java  ComprasProductoPK.java
1 package com.danicode.market.persistence.entity;
2
3 import java.io.Serializable;
4
5 import javax.persistence.Column;
6 import javax.persistence.Embeddable;
7
8 @Embeddable
9 public class ComprasProductoPK implements Serializable {
10
11     private static final long serialVersionUID = 1L;
12
13     @Column(name = "id_compra")
14     private Integer idCompra;
15
16     @Column(name = "id_producto")
17     private Integer idProducto;
18
19     public Integer getIdCompra() {
20         return idCompra;
21     }
22
23     public void setIdCompra(Integer idCompra) {
24         this.idCompra = idCompra;
25     }
26
27     public Integer getIdProducto() {
28         return idProducto;
29     }
30
31     public void setIdProducto(Integer idProducto) {
32         this.idProducto = idProducto;
33     }
34 }
```

En la clase Comprasproducto:

```
ComprasProducto.java  ComprasProductoPK.java
1 package com.danicode.market.persistence.entity;
2
3 import javax.persistence.EmbeddedId;
4 import javax.persistence.Entity;
5 import javax.persistence.Table;
6
7 @Entity
8 @Table(name = "compras_productos")
9 public class ComprasProducto {
10
11     // Llave primaria compuesta
12     @EmbeddedId
13     private ComprasProductoPK id;
14
15     private Integer cantidad;
16
17     private Double total;
18
19     private Boolean estado;
20
21     public ComprasProductoPK getId() {
22         return id;
23     }
24
25     public void setId(ComprasProductoPK id) {
26         this.id = id;
27     }
28 }
```



## ➔ Mapear relaciones entre clases



```
@Entity
@Table(name = "productos")
public class Producto {

    • @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      @Column(name = "id_producto")
      private Integer idProducto; // Usar la clase Integer y no el primitivo int

      private String nombre;

    • @Column(name = "id_categoria")
      private Integer idCategoria;

    • @Column(name = "codigo_barras")
      private String codigoBarras;

    • @Column(name = "precio_venta")
      private Double precioVenta;

    • @Column(name = "cantidad_stock")
      private Integer cantidadStock;

      private Boolean estado;

    • /*
       * Varios productos estan en una misma categoria
       * insertable = false, updatable = false => No se crea ni actualiza en esta
       * relacion, sino a traves de la clase Categoria
       */
    • @ManyToOne
      @JoinColumn(name = "id_categoria", insertable = false, updatable = false)
      private Categoria categoria;
```

```
@Entity
@Table(name = "categorias")
public class Categoria {

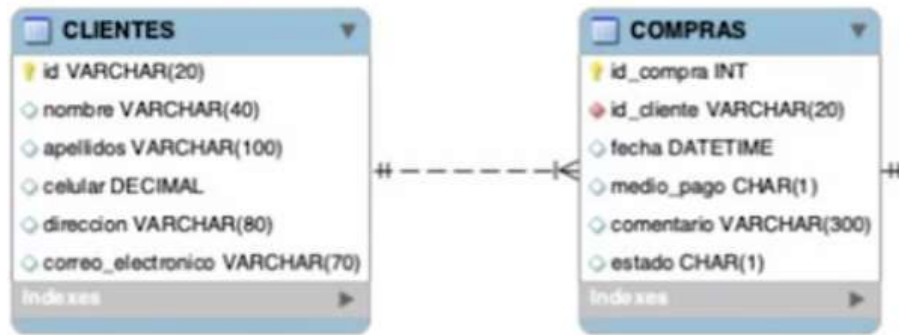
    • @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      @Column(name = "id_categoria")
      private Integer idCategoria;

      private String descripcion;

      private Boolean estado;

    • /*
       * Una categoria tiene varios productos
       */
    • @OneToMany(mappedBy = "categoria")
      private List<Producto> productos;
```

Luego:



```
Compra.java
1 package com.danicode.market.persistence.entity;
2
3 import java.time.LocalDateTime;
13
14 @Entity
15 @Table(name = "compras")
16 public class Compra {
17
18     @Id
19     @GeneratedValue(strategy = GenerationType.IDENTITY)
20     @Column(name = "id_compra")
21     private Integer idCompra;
22
23     @Column(name = "id_cliente")
24     private Integer idCliente;
25
26     private LocalDateTime fecha;
27
28     @Column(name = "medio_pago")
29     private String medioPago;
30
31     private String comentario;
32
33     private Boolean estado;
34
35     @ManyToOne
36     @JoinColumn(name = "id_cliente", insertable = false, updatable = false)
37     private Cliente cliente;
```

```
@Entity
@Table(name = "clientes")
public class Cliente {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String nombre;

    private String apellidos;

    private Long celular;

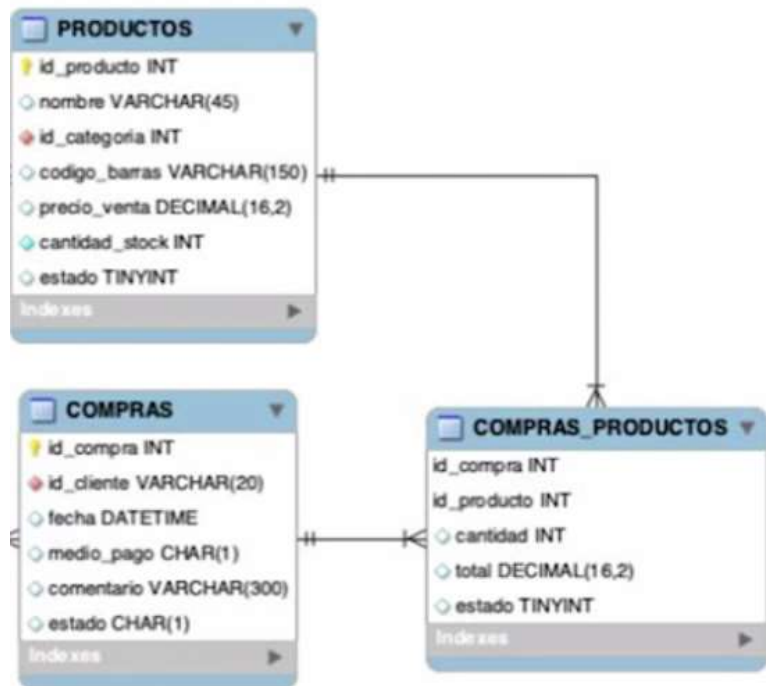
    private String direccion;

    @Column(name = "correo_electronico")
    private String correoElectronico;

    {
        @OneToMany(mappedBy = "cliente")
        private List<Compra> compras;
```



Luego:



```
@Entity
@Table(name = "compras_productos")
public class ComprasProducto {

    // Llave primaria compuesta
    @EmbeddedId
    private ComprasProductoPK id;

    private Integer cantidad;

    private Double total;

    private Boolean estado;

    @ManyToOne
    @JoinColumn(name = "id_compra", insertable = false, updatable = false)
    private Compra compra;

    @ManyToOne
    @JoinColumn(name = "id_producto", insertable = false, updatable = false)
    private Producto producto;
}
```

```
@Entity
@Table(name = "compras")
public class Compra {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_compra")
    private Integer idCompra;

    @Column(name = "id_cliente")
    private Integer idCliente;

    private LocalDateTime fecha;

    @Column(name = "medio_pago")
    private String medioPago;

    private String comentario;

    private Boolean estado;

    @ManyToOne
    @JoinColumn(name = "id_cliente", insertable = false, updatable = false)
    private Cliente cliente;

    @OneToMany(mappedBy = "producto")
    private List<ComprasProducto> productos;
}
```


La relación entre “productos” y “compras\_productos” no es necesaria, ya que no aporta información relevante.

➔ Usar la interface CrudRepository

## Spring Data Repositories

- Ahorrar un MONTÓN de código y tiempo de implementación
- Operaciones SIN CÓDIGO en la BD
- Repositorios de Spring Data
  - CrudRepository
  - PagingAndSortingRepository
  - JpaRepository

Creando “ProductoCrudRepository” en la carpeta “crud” capa de persistencia:



The screenshot shows an IDE with two panels. The left panel is the Project Explorer, showing a project structure with a 'crud' folder under 'persistence'. The right panel shows the code for 'ProductoCrudRepository.java'. The code defines a package, imports, and a public interface that extends 'CrudRepository<Producto, Integer>'.

```
1 package com.danicode.market.persistence.crud;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 import com.danicode.market.persistence.entity.Producto;
6
7 public interface ProductoCrudRepository extends CrudRepository<Producto, Integer> {
8
9 }
```

➔ Query Methods

## Uso de Query Methods

- En ocasiones, necesitamos consultas que el Repository de Spring Data no nos puede ofrecer.
- Los Query Methods proveen la posibilidad de generar consultas mediante el nombre de los métodos.
- Tienen la posibilidad de retornar Optional<T>

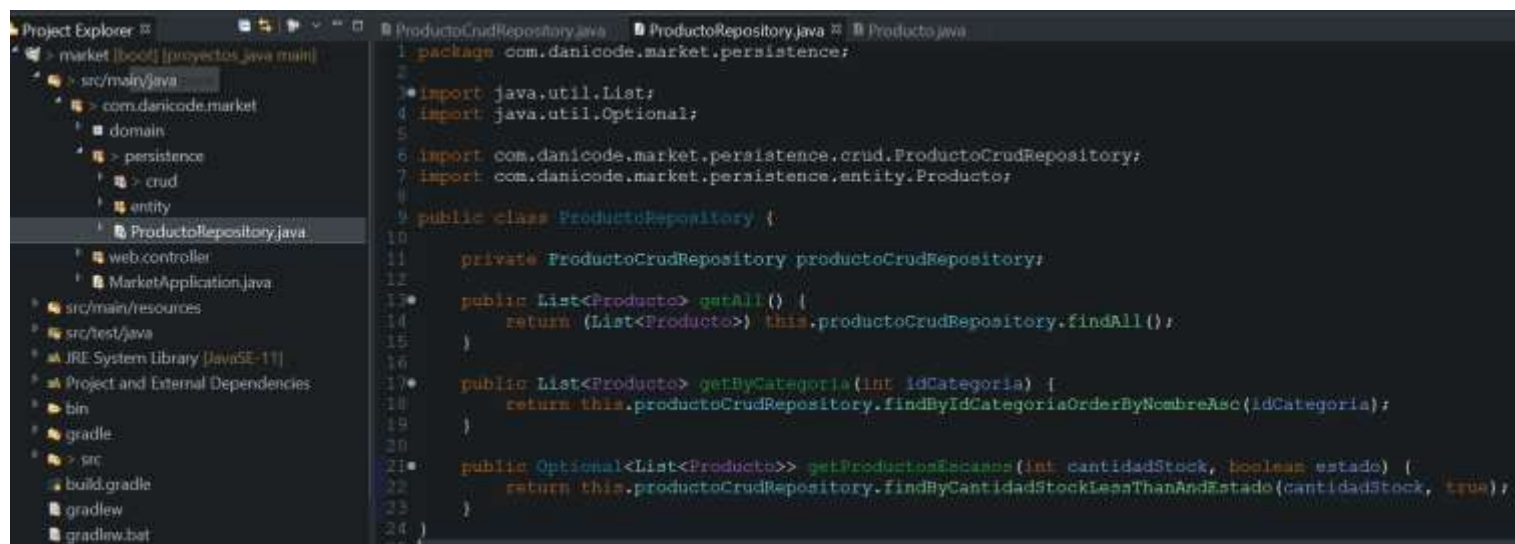
<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods>



The screenshot shows the same IDE as before, but now the code for 'ProductoCrudRepository.java' includes two query methods: 'findByIdCategoriaOrderByNombreAsc' and 'findByCantidadStockLessThanAndEstado'.

```
1 package com.danicode.market.persistence.crud;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.springframework.data.repository.CrudRepository;
7
8 import com.danicode.market.persistence.entity.Producto;
9
10 public interface ProductoCrudRepository extends CrudRepository<Producto, Integer> {
11     // Productos por categoria
12     public List<Producto> findByIdCategoriaOrderByNombreAsc(int idCategoria);
13
14     public Optional<List<Producto>> findByCantidadStockLessThanAndEstado(int cantidadStock, boolean estado);
15 }
16
```

Creando clase ProductoRepository:

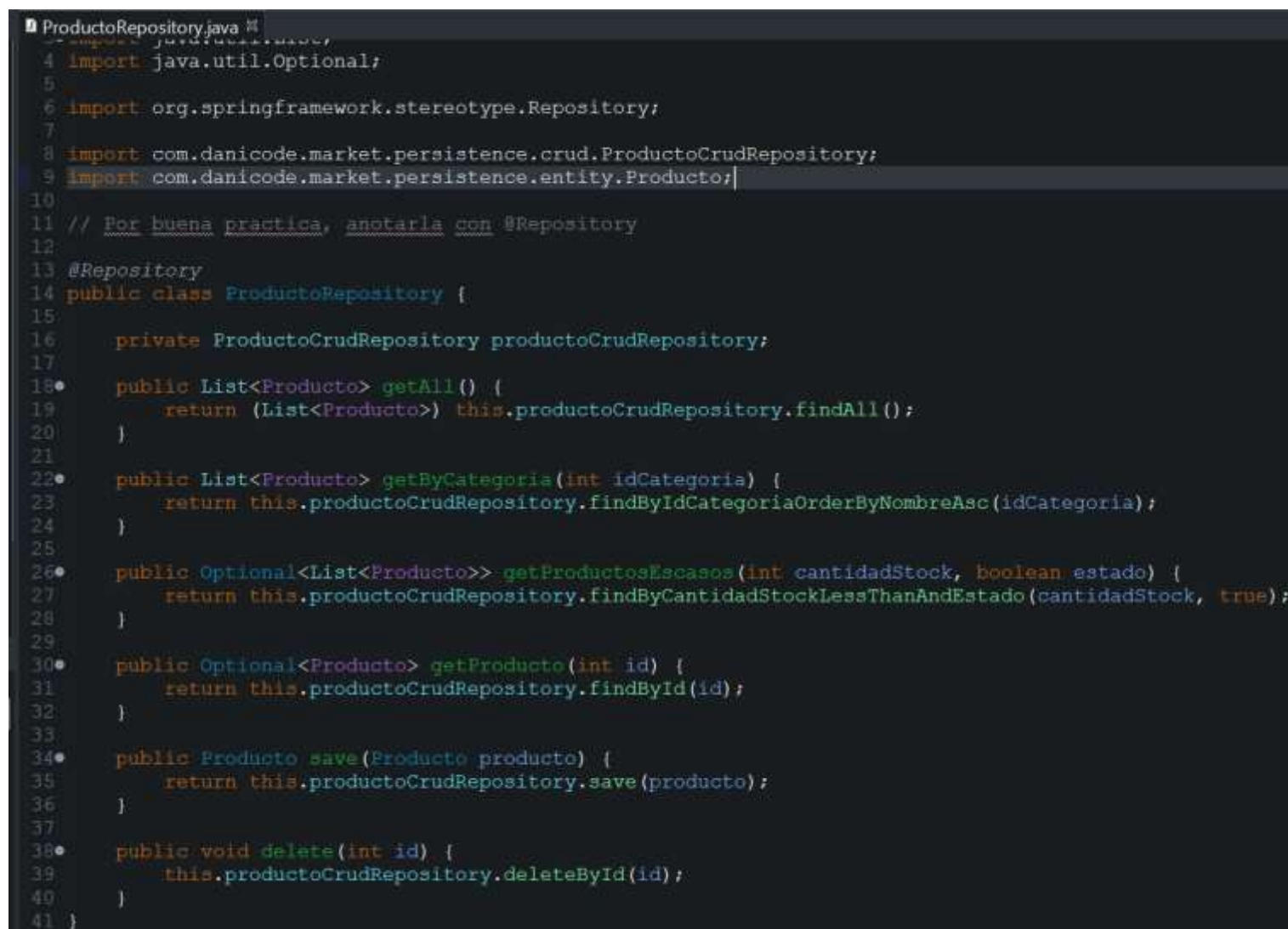


```
1 package com.danicode.market.persistence;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import com.danicode.market.persistence.crud.ProductoCrudRepository;
7 import com.danicode.market.persistence.entity.Producto;
8
9 public class ProductoRepository {
10
11     private ProductoCrudRepository productoCrudRepository;
12
13     public List<Producto> getAll() {
14         return (List<Producto>) this.productoCrudRepository.findAll();
15     }
16
17     public List<Producto> getByCategoria(int idCategoria) {
18         return this.productoCrudRepository.findByIdCategoriaOrderByNombreAsc(idCategoria);
19     }
20
21     public Optional<List<Producto>> getProductosEscasos(int cantidadStock, boolean estado) {
22         return this.productoCrudRepository.findByCantidadStockLessThanAndEstado(cantidadStock, true);
23     }
24 }
```

## ➔ Implementar la anotación @Repository

Anotación que se encarga de interactuar con la BD.

Clase ProductoRepository:

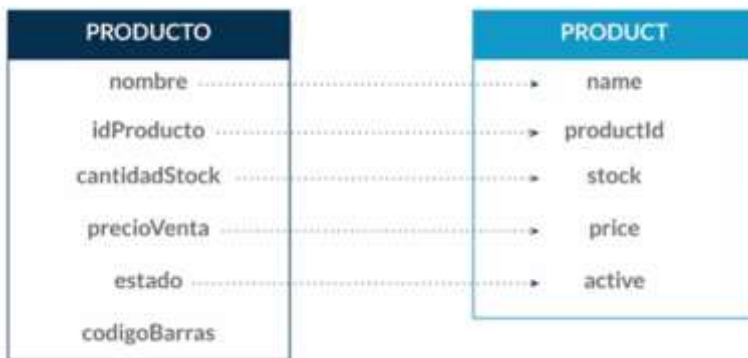


```
1 package com.danicode.market.persistence;
2
3 import java.util.Optional;
4
5 import org.springframework.stereotype.Repository;
6
7 import com.danicode.market.persistence.crud.ProductoCrudRepository;
8 import com.danicode.market.persistence.entity.Producto;
9
10 // Por buena practica, anotarla con @Repository
11
12 @Repository
13 public class ProductoRepository {
14
15     private ProductoCrudRepository productoCrudRepository;
16
17     public List<Producto> getAll() {
18         return (List<Producto>) this.productoCrudRepository.findAll();
19     }
20
21     public List<Producto> getByCategoria(int idCategoria) {
22         return this.productoCrudRepository.findByIdCategoriaOrderByNombreAsc(idCategoria);
23     }
24
25     public Optional<List<Producto>> getProductosEscasos(int cantidadStock, boolean estado) {
26         return this.productoCrudRepository.findByCantidadStockLessThanAndEstado(cantidadStock, true);
27     }
28
29     public Optional<Producto> getProducto(int id) {
30         return this.productoCrudRepository.findById(id);
31     }
32
33     public Producto save(Producto producto) {
34         return this.productoCrudRepository.save(producto);
35     }
36
37     public void delete(int id) {
38         this.productoCrudRepository.deleteById(id);
39     }
40 }
41 }
```

## ➔ Patrón Data Mapper y qué resuelve

Traducción de clases:

### En qué consiste el patrón Data Mapper



### ¿Y esto en qué nos ayuda? Logramos varias cosas...

- No exponer la base de datos en el API
- Desacoplar nuestra API a una base de datos puntual
- No tener campos innecesarios en el API
- Sin mezclar idiomas en el dominio

<https://mapstruct.org/>

<https://mapstruct.org/documentation/installation/>

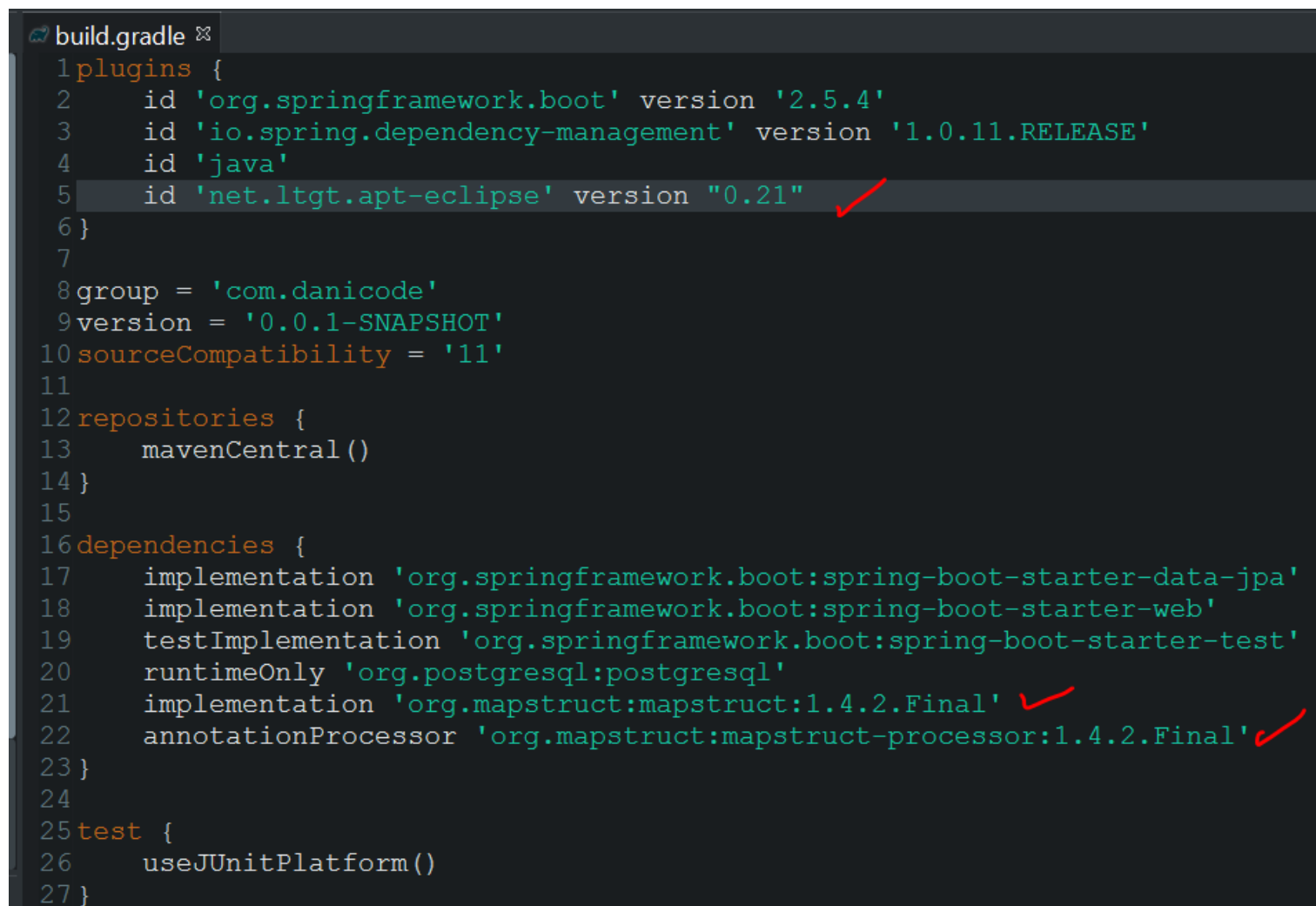
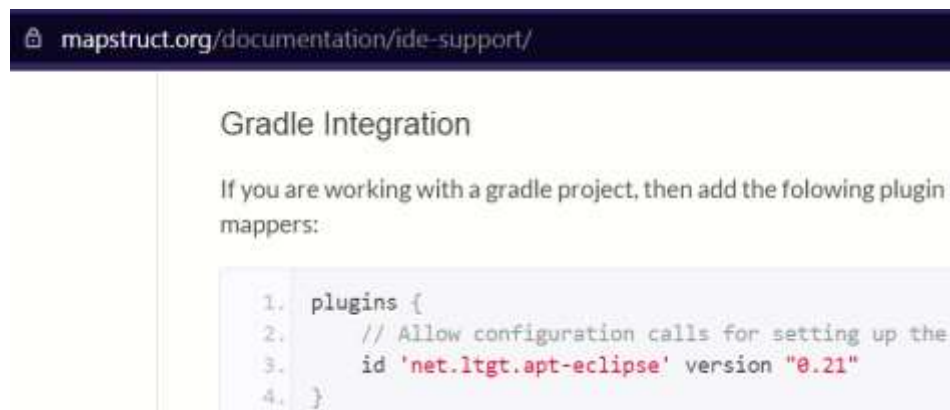


```
14
15 dependencies {
16     implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
17     implementation 'org.springframework.boot:spring-boot-starter-web'
18     testImplementation 'org.springframework.boot:spring-boot-starter-test'
19     runtimeOnly 'org.postgresql:postgresql'
20     implementation 'org.mapstruct:mapstruct:1.4.2.Final' {
21         annotationProcessor 'org.mapstruct:mapstruct-processor:1.4.2.Final' \
22     }
23 }
```

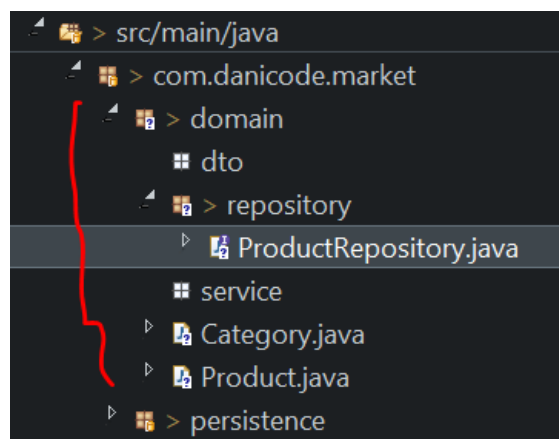


Instalar plugin para autocompletar las estructuras de mapStruct para eclipse:

<https://mapstruct.org/documentation/ide-support/>



En el paquete "domain":



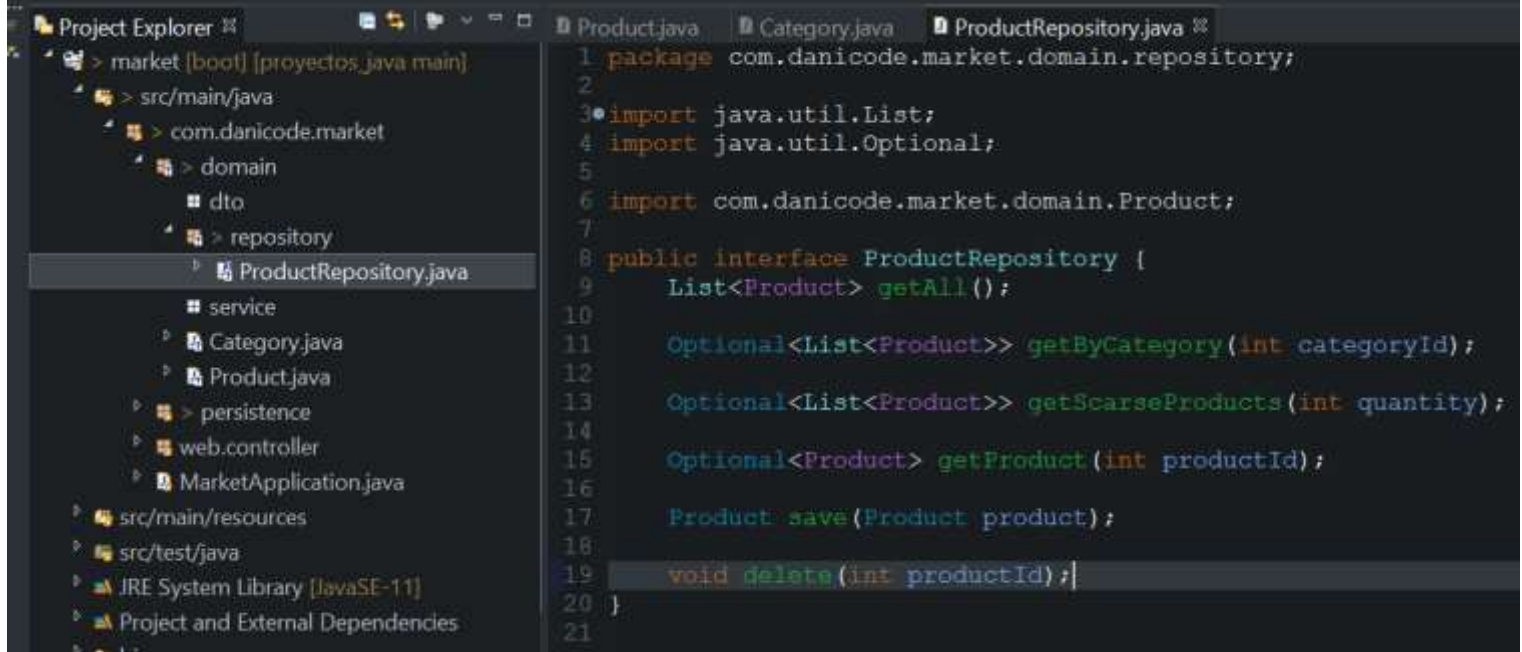
Product.java Category.java ProductRepository.java

```
1 package com.danicode.market.domain;
2
3 public class Category {
4     private int categoryId;
5     private String category;
6     private boolean active;
7
8     public int getCategoryId() {
9         return categoryId;
10    }
11
12    public void setCategoryId(int categoryId) {
13        this.categoryId = categoryId;
14    }
15
16    public String getCategory() {
17        return category;
18    }
19
20    public void setCategory(String category) {
21        this.category = category;
22    }
23
24    public boolean isActive() {
25        return active;
26    }
27
28    public void setActive(boolean active) {
29        this.active = active;
30    }
31 }
```

Product.java Category.java ProductRepository.java

```
4 * Clase que se relaciona con la entidad Producto sin codigo de barras
5 * Clase que usaremos en la respuesta de la API
6 */
7 public class Product {
8     private int productId;
9     private String name;
10    private int categoryId;
11    private double price;
12    private int stock;
13    private boolean active;
14    private Category category;
15
16    public int getProductId() {
17        return productId;
18    }
19
20    public void setProductId(int productId) {
21        this.productId = productId;
22    }
23
24    public String getName() {
25        return name;
26    }
27
28    public void setName(String name) {
29        this.name = name;
30    }
31
32    public int getCategoryId() {
33        return categoryId;
34    }
35
36    public void setCategoryId(int categoryId) {
37        this.categoryId = categoryId;
38    }
39
40    public double getPrice() {
41        return price;
42    }
43 }
```





## ➔ Orientar nuestra API al dominio con MapStruct

Creando paquete “mapper” con la interfaz “CategoryMapper” dentro de “persistence”

```
package com.danicode.market.persistence.mapper;

import org.mapstruct.InheritInverseConfiguration;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.Mappings;

import com.danicode.market.domain.Category;
import com.danicode.market.persistence.entity.Categoria;

@Mapper(componentModel = "spring")
public interface CategoryMapper {

    // De Categoria a Category
    @Mappings({ @Mapping(source = "idCategoria", target = "categoryId"),
        @Mapping(source = "descripcion", target = "category"), @Mapping(source = "estado",
target = "active") })
    Category toCategory(Categoria categoria);

    // De Category a Categoria sin mapear productos
    @InheritInverseConfiguration
    @Mapping(target = "productos", ignore = true)
    Categoria toCategoria(Category category);
}
```

```
CategoryMapper.java
1 package com.danicode.market.persistence.mapper;
2
3 import org.mapstruct.InheritInverseConfiguration;
4 import org.mapstruct.Mapper;
5 import org.mapstruct.Mapping;
6 import org.mapstruct.Mappings;
7
8 import com.danicode.market.domain.Category;
9 import com.danicode.market.persistence.entity.Categoria;
10
11 @Mapper(componentModel = "spring")
12 public interface CategoryMapper {
13
14     // De Categoria a Category
15     @Mappings({ @Mapping(source = "idCategoria", target = "categoryId"),
16         @Mapping(source = "descripcion", target = "category"), @Mapping(source = "estado", target = "active") })
17     Category toCategory(Categoria categoria);
18
19     // De Category a Categoria sin mapear productos
20     @InheritInverseConfiguration
21     @Mapping(target = "productos", ignore = true)
22     Categoria toCategoria(Category category);
23 }
```

En Product:

```
package com.danicode.market.persistence.mapper;

import java.util.List;

import org.mapstruct.InheritInverseConfiguration;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.Mappings;

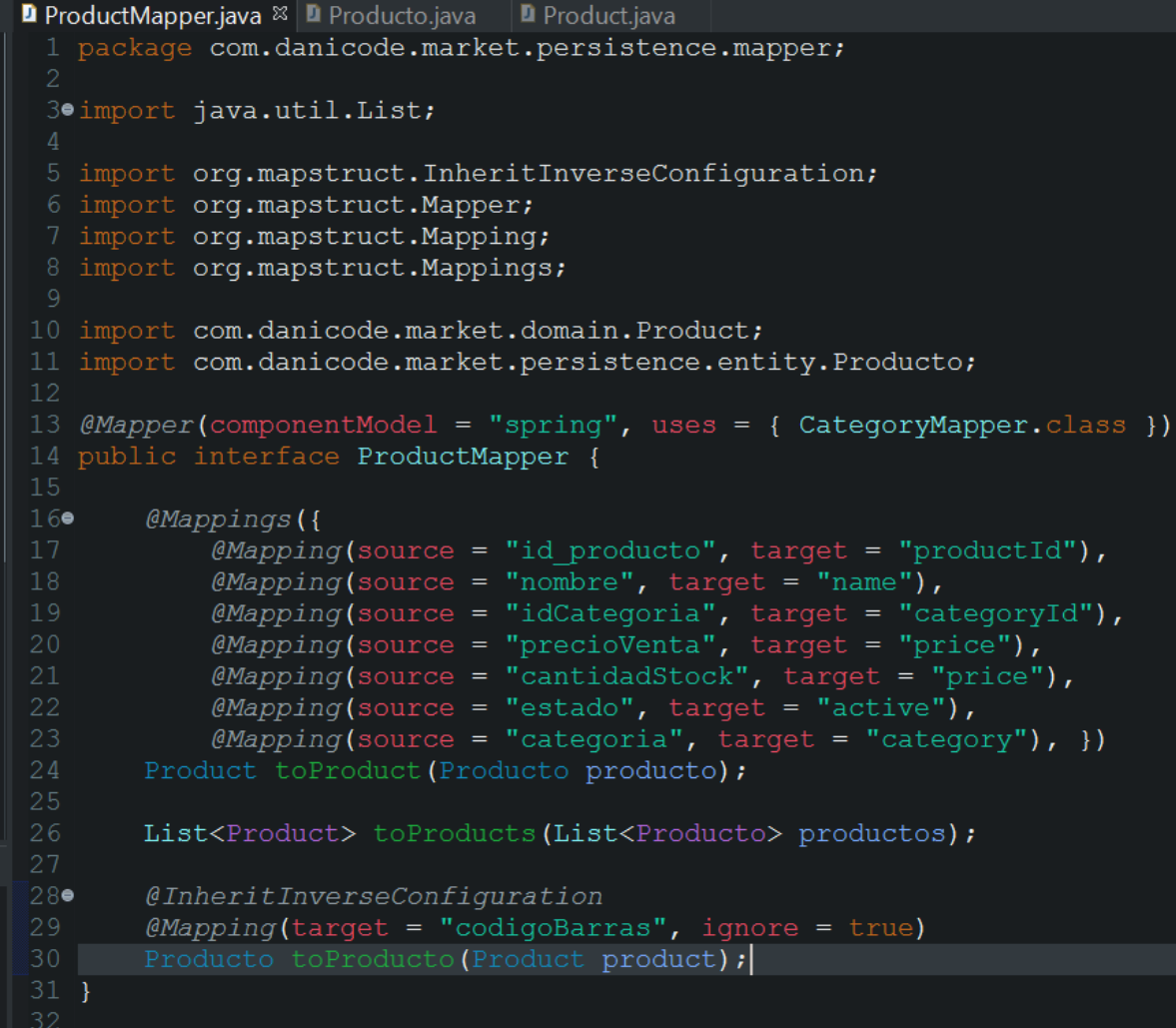
import com.danicode.market.domain.Product;
import com.danicode.market.persistence.entity.Producto;

@Mapper(componentModel = "spring", uses = {CategoryMapper.class})
public interface ProductMapper {

    @Mappings({@Mapping(source = "idProducto", target = "productId"), @Mapping(source = "nombre", target = "name"),
        @Mapping(source = "idCategoria", target = "categoryId"), @Mapping(source = "precioVenta", target = "price"),
        @Mapping(source = "cantidadStock", target = "stock"), @Mapping(source = "estado", target = "active"),
        @Mapping(source = "categoria", target = "category"),})
    Product toProduct(Producto producto);

    List<Product> toProducts(List<Producto> productos);

    @InheritInverseConfiguration
    @Mapping(target = "codigoBarras", ignore = true)
    Producto toProducto(Product product);
}
```



```
ProductMapper.java  Producto.java  Product.java
1 package com.danicode.market.persistence.mapper;
2
3 import java.util.List;
4
5 import org.mapstruct.InheritInverseConfiguration;
6 import org.mapstruct.Mapper;
7 import org.mapstruct.Mapping;
8 import org.mapstruct.Mappings;
9
10 import com.danicode.market.domain.Product;
11 import com.danicode.market.persistence.entity.Producto;
12
13 @Mapper(componentModel = "spring", uses = { CategoryMapper.class })
14 public interface ProductMapper {
15
16     @Mappings({
17         @Mapping(source = "id_producto", target = "productId"),
18         @Mapping(source = "nombre", target = "name"),
19         @Mapping(source = "idCategoria", target = "categoryId"),
20         @Mapping(source = "precioVenta", target = "price"),
21         @Mapping(source = "cantidadStock", target = "price"),
22         @Mapping(source = "estado", target = "active"),
23         @Mapping(source = "categoria", target = "category"), })
24     Product toProduct(Producto producto);
25
26     List<Product> toProducts(List<Producto> productos);
27
28     @InheritInverseConfiguration
29     @Mapping(target = "codigoBarras", ignore = true)
30     Producto toProducto(Product product);
31 }
32
```

➔Orientar nuestro repositorio a términos del dominio

Clase ProductoRepository:

```

1 package com.danicode.market.persistence;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.springframework.stereotype.Repository;
7
8 import com.danicode.market.domain.Product;
9 import com.danicode.market.domain.repository.ProductRepository;
10 import com.danicode.market.persistence.crud.ProductoCrudRepository;
11 import com.danicode.market.persistence.entity.Producto;
12 import com.danicode.market.persistence.mapper.ProductMapper;
13
14 // Por buena practica, anotarla con @Repository
15
16 @Repository
17 public class ProductoRepository implements ProductRepository {
18
19     private ProductoCrudRepository productoCrudRepository;
20     private ProductMapper mapper;
21
22     @Override
23     public List<Product> getAll() {
24         List<Producto> productos = (List<Producto>) this.productoCrudRepository.findAll();
25         return this.mapper.toProducts(productos);
26     }
27
28     @Override
29     public void delete(int productId) {
30         this.productoCrudRepository.deleteById(productId);
31     }
32
33     @Override
34     public Optional<List<Product>> getByCategory(int categoryId) {
35         List<Producto> productos = this.productoCrudRepository.findByIdCategoriaOrderByNombreAsc(categoryId);
36         return Optional.of(this.mapper.toProducts(productos));
37     }
38
39     @Override
40     public Optional<List<Product>> getScarseProducts(int quantity) {
41         Optional<List<Producto>> productos = this.productoCrudRepository.findByCantidadStockLessThanAndEstado(quantity,
42             true);
43         return productos.map(producto -> mapper.toProducts(producto));
44     }
45
46     @Override
47     public Optional<Product> getProduct(int productId) {
48         return this.productoCrudRepository.findById(productId).map(producto -> mapper.toProduct(producto));
49     }
50
51     @Override
52     public Product save(Product product) {
53         Producto producto = this.mapper.toProducto(product);
54         return this.mapper.toProduct(this.productoCrudRepository.save(producto));
55     }
56
57 }
58 }

```

➔ Inyección de dependencias

## Inyección de dependencias

- Principios S.O.L.I.D
- Inyección de dependencias (DI)
- Inversión de Control (IoC)
- Spring y @Autowired



```

1  import java.util.Optional;
2
3  in]
4
5
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.stereotype.Repository;
8
9  import com.danicode.market.domain.Product;
10 import com.danicode.market.domain.repository.ProductRepository;
11 import com.danicode.market.persistence.crud.ProductoCrudRepository;
12 import com.danicode.market.persistence.entity.Producto;
13 import com.danicode.market.persistence.mapper.ProductMapper;
14
15 // Por buena practica, anotarla con @Repository
16
17 @Repository
18 public class ProductoRepository implements ProductRepository {
19
20     @Autowired
21     private ProductoCrudRepository productoCrudRepository;
22
23     @Autowired
24     private ProductMapper mapper;
25
26     @Override
27     public List<Product> getAll() {
28         List<Producto> productos = (List<Producto>) this.productoCrudRepository.findAll();
29         return this.mapper.toProducts(productos);
30     }

```

➔ Implementar la anotación @Service

Project Explorer

- market [boot] [proyectos\_java main]
  - src/main/java
    - com.danicode.market
      - domain
        - dto
        - repository
        - service
          - ProductService.java
      - Category.java
      - Product.java
      - persistence
      - web.controller
      - MarketApplication.java
    - src/main/resources
    - src/test/java
    - JRE System Library [JavaSE-11]
    - Project and External Dependencies
    - bin
    - gradle
      - src
        - build.gradle
        - gradlew
        - gradlew.bat
        - HELP.md
        - settings.gradle
    - Servers [proyectos\_spring main]

```

3  import java.util.List;
4  import java.util.Optional;
5
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.stereotype.Service;
8
9  import com.danicode.market.domain.Product;
10 import com.danicode.market.domain.repository.ProductRepository;
11
12 @Service
13 public class ProductService {
14
15     @Autowired
16     private ProductRepository productRepository;
17
18     public List<Product> getAll() {
19         return this.productRepository.getAll();
20     }
21
22     public Optional<Product> getProduct(int productId) {
23         return this.productRepository.getProduct(productId);
24     }
25
26     public Optional<List<Product>> getByCategory(int categoryId) {
27         return this.productRepository.getByCategory(categoryId);
28     }
29
30     public Product save(Product product) {
31         return this.productRepository.save(product);
32     }
33
34     public boolean delete(int productId) {
35         return getProduct(productId).map(product -> {
36             this.productRepository.delete(productId);
37             return true;
38         }).orElse(false);
39     }
40 }

```

## ➔ Implementar la anotación @RestController

# ¿Qué anotaciones usaremos?

- Nuestra API se expone por @RestController
- Los métodos se exponen con @GetMapping, @PostMapping ó @DeleteMapping

Creando controlador ProductoController en “web.controller”:

```
package com.danicode.market.web.controller;

//http://localhost:8090/market/api/products/**

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.danicode.market.domain.Product;
import com.danicode.market.domain.service.ProductService;

@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping("/all")
    public List<Product> getAll() {
        return this.productService.getAll();
    }

    @GetMapping("/{id}")
    public Optional<Product> getProduct(@PathVariable("id") int productId) {
        return this.productService.getProduct(productId);
    }

    @GetMapping("/category/{categoryId}")
    public Optional<List<Product>> getCategory(@PathVariable("categoryId") int categoryId) {
        return this.productService.getCategory(categoryId);
    }

    @PostMapping("/save")
    public Product save(@RequestBody Product product) {
        return this.productService.save(product);
    }

    @DeleteMapping("/delete/{id}")
    public boolean delete(@PathVariable("id") int productId) {
        return this.productService.delete(productId);
    }
}
```

# ResponseEntity

- ¿Qué es y en qué nos ayuda?
- HttpStatus

```
package com.danicode.market.web.controller;

//http://localhost:8090/market/api/products/**

import java.util.List;
import java.util.Optional;

import org.apache.coyote.Response;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.danicode.market.domain.Product;
import com.danicode.market.domain.service.ProductService;

@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping("/all")
    public ResponseEntity<List<Product>> getAll() {
        return new ResponseEntity<>(this.productService.getAll(), HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Product> getProduct(@PathVariable("id") int productId) {
        return this.productService.getProduct(productId)
            .map(product -> new ResponseEntity<>(product, HttpStatus.OK))
            .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @GetMapping("/category/{categoryId}")
    public ResponseEntity<List<Product>> getByCategory(@PathVariable("categoryId") int
categoryId) {
        return this.productService.getByCategory(categoryId)
            .map(products -> new ResponseEntity<>(products, HttpStatus.OK))
            .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @PostMapping("/save")
    public ResponseEntity<Product> save(@RequestBody Product product) {
        return new ResponseEntity<>(this.productService.save(product), HttpStatus.CREATED);
    }

    @DeleteMapping("/delete/{id}")
    public ResponseEntity delete(@PathVariable("id") int productId) {
        if (this.productService.delete(productId)) {
            return new ResponseEntity(HttpStatus.OK);
        }
    }
}
```

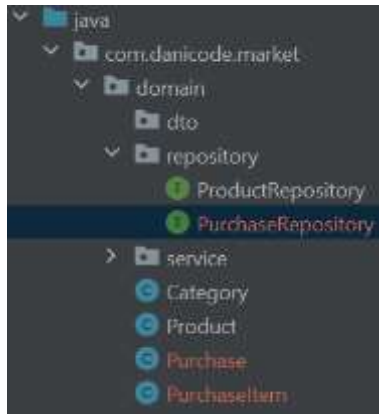


```

    }
    return new ResponseEntity(HttpStatus.NOT_FOUND);
}
}

```

## ➔ Crear el dominio de compras



### Clase PurchaseItem (Item de la compra)

```

package com.danicode.market.domain;

public class PurchaseItem {
    private int productId;
    private int quantity;
    private double total;
    private boolean active;

    public int getProductId() {
        return productId;
    }

    public void setProductId(int productId) {
        this.productId = productId;
    }

    public int getQuantity() {
        return quantity;
    }

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

    public double getTotal() {
        return total;
    }

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

    public boolean isActive() {
        return active;
    }

    public void setActive(boolean active) {
        this.active = active;
    }
}

```

### Clase Purchase:

```

package com.danicode.market.domain;

import java.time.LocalDateTime;
import java.util.List;

```

```
public class Purchase {
    private int purchaseId;
    private String clientId;
    private LocalDateTime date;
    private String paymentMethod;
    private String comment;
    private String state;
    private List<PurchaseItem> items;

    public int getPurchaseId() {
        return purchaseId;
    }

    public void setPurchaseId(int purchaseId) {
        this.purchaseId = purchaseId;
    }

    public String getClientId() {
        return clientId;
    }

    public void setClientId(String clientId) {
        this.clientId = clientId;
    }

    public LocalDateTime getDate() {
        return date;
    }

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

    public String getPaymentMethod() {
        return paymentMethod;
    }

    public void setPaymentMethod(String paymentMethod) {
        this.paymentMethod = paymentMethod;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public List<PurchaseItem> getItem() {
        return items;
    }

    public void setItem(List<PurchaseItem> items) {
        this.items = items;
    }
}
```

## PurchaseRepository:

```
package com.danicode.market.domain.repository;

import com.danicode.market.domain.Purchase;

import java.util.List;
import java.util.Optional;

public interface PurchaseRepository {
    List<Purchase> getAll();

    Optional<List<Purchase>> getByClient(String clientId);

    Purchase save(Purchase purchase);
}
```

## ➔Mapear el dominio de compras

Crear mapper PurchaseItemMapper:

```
package com.danicode.market.persistence.mapper;

import com.danicode.market.domain.PurchaseItem;
import com.danicode.market.persistence.entity.ComprasProducto;
import org.mapstruct.InheritInverseConfiguration;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.Mappings;

@Mapper(componentModel = "spring", uses = {ProductMapper.class})
public interface PurchaseItemMapper {

    // El atributo total esta en ambas clases, no es necesario mapearlo
    @Mappings({
        @Mapping(source = "id.idProducto", target = "productId"),
        @Mapping(source = "cantidad", target = "quantity"),
        @Mapping(source = "total", target = "total"),
        @Mapping(source = "estado", target = "active")
    })
    PurchaseItem toPurchaseItem(ComprasProducto producto);

    /*
     * Conversion contraria
     * Se debe mapear todos los atributos, si no se usan, mapearlos con
     * ignore = true (en ComprasProducto, ignorar "compra", "producto") y
     * dentro de "ComprasProductoPK", ignorar "idCompra"
     */
    @InheritInverseConfiguration
    @Mappings({
        @Mapping(target = "compra", ignore = true),
        @Mapping(target = "producto", ignore = true),
        @Mapping(target = "id.idCompra", ignore = true),
    })
    ComprasProducto toComprasProducto(PurchaseItem item);
}
```

Crear PurchaseMapper:

```
package com.danicode.market.persistence.mapper;

import com.danicode.market.domain.Purchase;
import com.danicode.market.persistence.entity.Compra;
import org.mapstruct.InheritInverseConfiguration;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.Mappings;

import java.util.List;

@Mapper(componentModel = "spring", uses = {PurchaseItemMapper.class})
```

```

public interface PurchaseMapper {

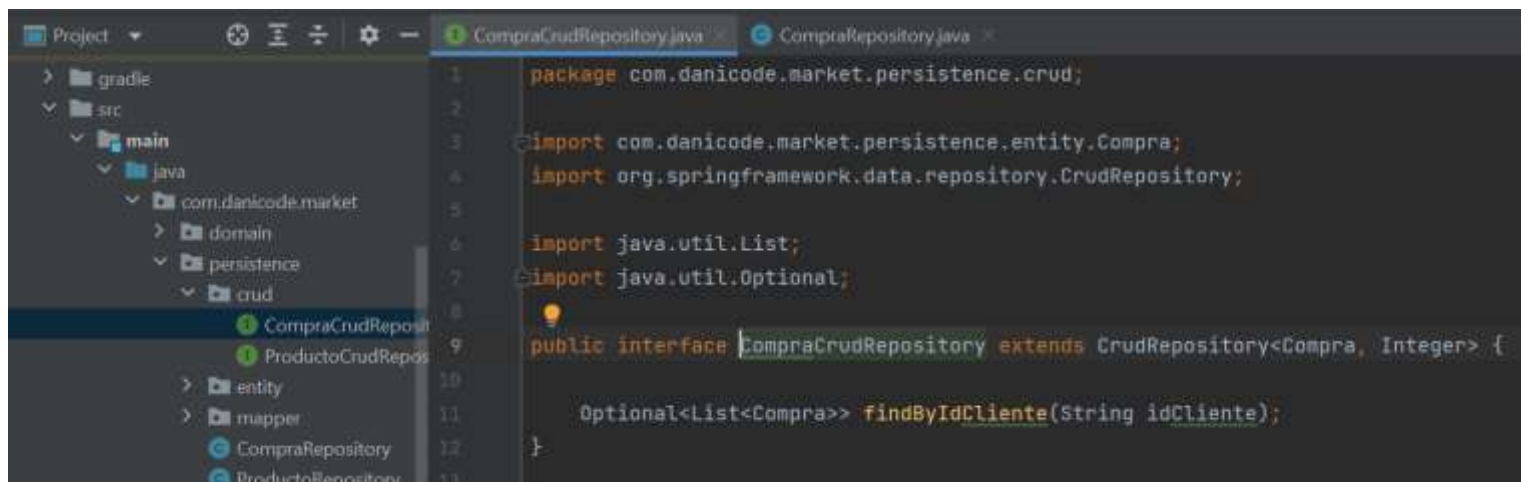
    @Mappings({
        @Mapping(source = "idCompra", target = "purchaseId"),
        @Mapping(source = "idCliente", target = "clientId"),
        @Mapping(source = "fecha", target = "date"),
        @Mapping(source = "medioPago", target = "paymentMethod"),
        @Mapping(source = "comentario", target = "comment"),
        @Mapping(source = "estado", target = "state"),
        @Mapping(source = "productos", target = "items"),
    })
    Purchase toPurchase(Compra compra);

    List<Purchase> toPurchases(List<Compra> compras);

    // Invertir mapeo => Ignorar el cliente
    @InheritInverseConfiguration
    @Mapping(target = "cliente", ignore = true)
    Compra toCompra(Purchase purchase);
}

```

## ➔ Crear el repositorio de compras



Implementar:

```

package com.danicode.market.persistence;

import com.danicode.market.domain.Purchase;
import com.danicode.market.domain.repository.PurchaseRepository;
import com.danicode.market.persistence.crud.CompraCrudRepository;
import com.danicode.market.persistence.entity.Compra;
import com.danicode.market.persistence.mapper.PurchaseMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository
public class CompraRepository implements PurchaseRepository {

    @Autowired
    private CompraCrudRepository compraCrudRepository;

    @Autowired
    private PurchaseMapper purchaseMapper;

    @Override
    public List<Purchase> getAll() {

        return purchaseMapper.toPurchases((List<Compra>) this.compraCrudRepository.findAll());
    }

    @Override
    public Optional<List<Purchase>> getByClient(String clientId) {

```

```

        return this.compraCrudRepository.findByIdCliente(clientId)
            .map(compras -> purchaseMapper.toPurchases(compras));
    }

    @Override
    public Purchase save(Purchase purchase) {
        Compra compra = purchaseMapper.toCompra(purchase);
        // Guardado en cascada (Compra conoce los productos y los productos
        // conocen a que compra pertenece) => en el atributo productos de la
        // clase Compra: cascade = {CascadeType.ALL}
        compra.getProductos().forEach(producto -> producto.setCompra(compra));
        return purchaseMapper.toPurchase(this.compraCrudRepository.save(compra));
    }
}

```

Creando el servicio en el dominio:

```

package com.danicode.market.domain.service;

import com.danicode.market.domain.Purchase;
import com.danicode.market.domain.repository.PurchaseRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class PurchaseService {

    @Autowired
    private PurchaseRepository purchaseRepository;

    public List<Purchase> getAll() {
        return this.purchaseRepository.getAll();
    }

    public Optional<List<Purchase>> getByClient(String clientId) {
        return this.purchaseRepository.getByClient(clientId);
    }

    public Purchase save(Purchase purchase) {
        return purchaseRepository.save(purchase);
    }
}

```

Controlador:

```

package com.danicode.market.web.controller;

import com.danicode.market.domain.Purchase;
import com.danicode.market.domain.service.PurchaseService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/purchases")
public class PurchaseController {

    @Autowired
    private PurchaseService purchaseService;

    @GetMapping("/all")
    public ResponseEntity<List<Purchase>> getAll() {
        return new ResponseEntity<>(purchaseService.getAll(), HttpStatus.OK);
    }
}

```

```

@GetMapping("/client/{idClient}")
public ResponseEntity<List<Purchase>> getByClient(@PathVariable("idClient") String
clientId) {
    return purchaseService.getByClient(clientId).map(
        purchases -> new ResponseEntity<>(purchases, HttpStatus.OK)
    ).orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
}

@PostMapping("/save")
public ResponseEntity<Purchase> save(@RequestBody Purchase purchase) {
    return new ResponseEntity<>(purchaseService.save(purchase), HttpStatus.CREATED);
}
}

```

## ➔ Documentar nuestra API con Swagger

### ¿Por qué documentar nuestra API?

- Le agregaremos una capa de entendimiento
- Es más fácil de usar
- Es más profesional
- Quien consuma tendrá información oficial y de primera mano



Incluyendo las dependencias:

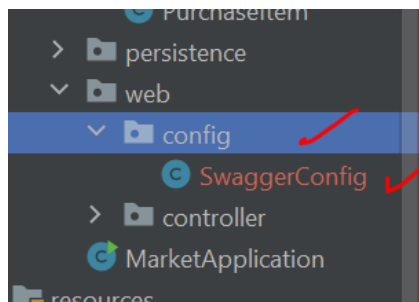
```

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    implementation 'org.mapstruct:mapstruct:1.3.1.Final'
    annotationProcessor 'org.mapstruct:mapstruct-processor:1.3.1.Final'
    runtimeOnly 'org.postgresql:postgresql'
    implementation 'io.springfox:springfox-swagger2:2.9.2'
    implementation 'io.springfox:springfox-swagger-ui:2.9.2'
}

```

La primera es para usar la dependencia y la otra es para generar una interfaz gráfica con las urls.

Crear la configuración “config/SwaggerConfig” en la carpeta web :



Se puede ver la interfaz en: <http://localhost:8090/market/api/swagger-ui.html>

En el controlador:

```

package com.danicode.market.web.controller;

//http://localhost:8090/market/api/products/**

```



```

import java.util.List;
import java.util.Optional;

import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.danicode.market.domain.Product;
import com.danicode.market.domain.service.ProductService;

@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping("/all")
    @ApiOperation("Get all supermarket products")
    @ApiResponse(code = 200, message = "OK")
    public ResponseEntity<List<Product>> getAll() {
        return new ResponseEntity<>(this.productService.getAll(), HttpStatus.OK);
    }

    @GetMapping("/{id}")
    @ApiOperation("Search a product with ID")
    @ApiResponses({
        @ApiResponse(code = 200, message = "OK"),
        @ApiResponse(code = 404, message = "Product not found"),
    })
    public ResponseEntity<Product> getProduct(@ApiParam(value = "ID of product", required =
true, example = "7") @PathVariable("id") int productId) {
        return this.productService.getProduct(productId)
            .map(product -> new ResponseEntity<>(product, HttpStatus.OK))
            .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @GetMapping("/category/{categoryId}")
    public ResponseEntity<List<Product>> getByCategory(@PathVariable("categoryId") int
categoryId) {
        return this.productService.getByCategory(categoryId)
            .map(products -> new ResponseEntity<>(products, HttpStatus.OK))
            .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @PostMapping("/save")
    public ResponseEntity<Product> save(@RequestBody Product product) {
        return new ResponseEntity<>(this.productService.save(product), HttpStatus.CREATED);
    }

    @DeleteMapping("/delete/{id}")
    public ResponseEntity delete(@PathVariable("id") int productId) {
        if (this.productService.delete(productId)) {
            return new ResponseEntity(HttpStatus.OK);
        }
        return new ResponseEntity(HttpStatus.NOT_FOUND);
    }
}

```

## ➔ Configurar la seguridad de nuestra API con Spring Security

Descargar dependencia de repositorio maven:



### Spring Boot Starter Security » 2.5.4

Starter for using Spring Security

License	Apache 2.0
Organization	Pivotal Software, Inc.
HomePage	<a href="https://spring.io/projects/spring-boot">https://spring.io/projects/spring-boot</a>
Date	(Aug 19, 2021)
Files	<a href="#">pom (2 KB)</a> <a href="#">jar (4 KB)</a> <a href="#">View All</a>
Repositories	Central
Used By	1,399 artifacts

Maven	Gradle	Gradle (Short)	Gradle (Kotlin)	SBT	Ivy	Grape	Leiningen	Buildr
<pre>// https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-security implementation group: 'org.springframework.boot', name: 'spring-boot-starter-security', version: '2.5.4'</pre>								

En el archivo “build.gradle”:

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    implementation 'org.mapstruct:mapstruct:1.3.1.Final'
    annotationProcessor 'org.mapstruct:mapstruct-processor:1.3.1.Final'
    runtimeOnly 'org.postgresql:postgresql'
    implementation 'io.springfox:springfox-swagger2:2.9.2'
    implementation 'io.springfox:springfox-swagger-ui:2.9.2'
    implementation 'org.springframework.boot:spring-boot-starter-security'
}
```

Creando servicio para ingresar al API con nuestras credenciales (DanicodeUserDetailsService):

```
package com.danicode.market.domain.service;

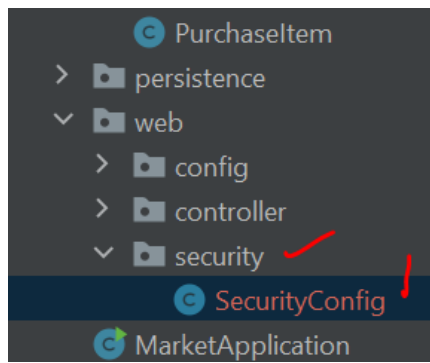
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.ArrayList;

@Service
public class DanicodeUserDetailsService implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // User("usuario", "password", Tipo de Errores)
        return new User("danicode", "{noop}123456", new ArrayList<>());
    }
}
```

Crear el paquete “security”, crear la clase “SecurityConfig” dentro de “web”:



La clase tendrá la anotación @EnableWebSecurity.

```
package com.danicode.market.web.security;

import com.danicode.market.domain.service.DanicodeUserDetailsService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    // Usar las credenciales creadas en la clase DanicodeUserDetailsService
    // Sobrecribir configure(AuthenticationManagerBuilder auth)
    @Autowired
    private DanicodeUserDetailsService danicodeUserDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(danicodeUserDetailsService);
    }
}
```

➔ Generar un JWT

## Qué es un JWT

- Estándar de código abierto basado en JSON para crear tokens de seguridad
- La autenticación viaja en el header de la petición:

**Authorization: Bearer <token>**

**1** eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjMNTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDAifQ.XbPflHMI6arZ3Y922BhjWgQzWXcXNrzo0gtVhfEd2o**3**

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

```

HMACSHA256(
  BASE64URL(header)
  .
  BASE64URL(payload) ,
  secret)

```



## JSON Web Token Support For The JVM

License	Apache 2.0
Categories	JWT Libraries
HomePage	<a href="https://github.com/jwtk/jjwt">https://github.com/jwtk/jjwt</a>
Date	(Jul 05, 2018)
Files	jar (110 KB) View All
Repositories	Central Spring Lib M Spring Lib Release
Used By	611 artifacts

## Leiningen

```
// https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt
implementation group: 'io.jsonwebtoken', name: 'jjwt', version: '0.9.1'
```

Creando clase JWTUtil, dentro de security, que se encargará de generar el JWT

```
package com.danicode.market.web.security;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.security.core.userdetails.UserDetails;

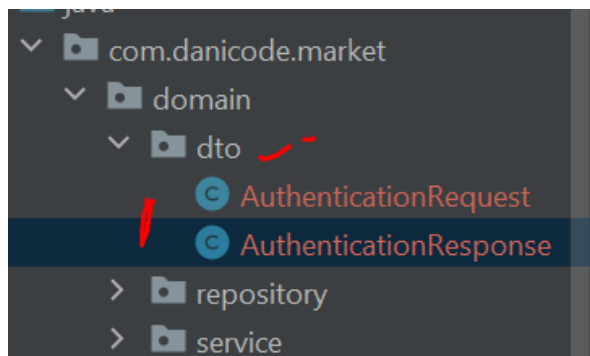
import java.util.Date;

public class JWTUtil {

    private static final String KEY = "danicode";

    public String generateToken(UserDetails userDetails) {
        // setIssuedAt > Fecha de creacion del token
        // setExpiration > 10 horas
        return Jwts.builder().setSubject(userDetails.getUsername())
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10))
            .signWith(SignatureAlgorithm.HS256, KEY)
            .compact();
    }
}
```

Crear las clases AuthenticationRequest y AuthenticationResponse dentro del paquete “dto” para usarlos en el controlador que manejará el JWT.



JWTUtil.java × AuthenticationRequest.java × AuthenticationResponse.java ×

```
1 package com.danicode.market.domain.dto;
2
3 public class AuthenticationRequest {
4     private String username;
5     private String password;
6
7     public String getUsername() {
8         return username;
9     }
10
11     public void setUsername(String username) {
12         this.username = username;
13     }
14
15     public String getPassword() {
16         return password;
17     }
18
19     public void setPassword(String password) {
20         this.password = password;
21     }
22 }
```

JWTUtil.java × AuthenticationRequest.java × AuthenticationResponse.java ×

```
1 package com.danicode.market.domain.dto;
2
3 public class AuthenticationResponse {
4     private String jwt;
5
6     public AuthenticationResponse(String jwt) {
7         this.jwt = jwt;
8     }
9
10    public String getJwt() {
11        return jwt;
12    }
13
14    public void setJwt(String jwt) {
15        this.jwt = jwt;
16    }
17 }
```



## ➔ Autenticación con JWT

Creando controlador “AuthController”:

```
package com.danicode.market.web.controller;

import com.danicode.market.domain.dto.AuthenticationRequest;
import com.danicode.market.domain.dto.AuthenticationResponse;
import com.danicode.market.domain.service.DanicodeUserDetailsService;
import com.danicode.market.web.security.JWTUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private DanicodeUserDetailsService danicodeUserDetailsService;

    @Autowired
    private JWTUtil jwtUtil;

    @PostMapping("/authenticate")
    public ResponseEntity<AuthenticationResponse> createToken(@RequestBody
AuthenticationRequest request) {
        try {
            // Verificar credenciales
            this.authenticationManager
                .authenticate(new
UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword()));
            // Detalles del usuario
            UserDetails userDetails =
this.danicodeUserDetailsService.loadUserByUsername(request.getUsername());
            // Generar token
            String jwt = this.jwtUtil.generateToken(userDetails);

            return new ResponseEntity<>(new AuthenticationResponse(jwt), HttpStatus.OK);
        } catch (BadCredentialsException e) {
            return new ResponseEntity<>(HttpStatus.FORBIDDEN);
        }
    }
}
```

Indicar que todas las peticiones se hagan con “authenticate” (usuarios autenticados). En la configuración, “SecurityConfig”:

```
package com.danicode.market.web.security;

import com.danicode.market.domain.service.DanicodeUserDetailsService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
```

```

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    // Usar las credenciales creadas en la clase DanicodeUserDetailsService
    // Sobrecribir configure(AuthenticationManagerBuilder auth)
    @Autowired
    private DanicodeUserDetailsService danicodeUserDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(danicodeUserDetailsService);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // PERMITIR Todas las peticiones que terminen en authenticate
        // Las otras peticiones, necesitan autenticarse => anyRequest().authenticated()
        http.csrf().disable().authorizeRequests()
            .antMatchers("/**/authenticate").permitAll()
            .anyRequest().authenticated();
    }

    // Incluir AuthenticationManager con anotación @Bean para inyectarlo y usarlo
    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}

```

## ➔ Autorización con JWT

Validando token en la clase JWTUtil:

```

package com.danicode.market.web.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
public class JWTUtil {

    private static final String KEY = "danicode";

    public String generateToken(UserDetails userDetails) {
        // setIssuedAt > Fecha de creacion del token
        // setExpiration > 10 horas
        return Jwts.builder().setSubject(userDetails.getUsername())
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10))
            .signWith(SignatureAlgorithm.HS256, KEY)
            .compact();
    }

    public boolean validateToken(String token, UserDetails userDetails) {
        // Verificar el usuario y que token no haya expirado
        return (userDetails.getUsername().equals(extractUsername(token))) &&
(!isTokenExpired(token));
    }

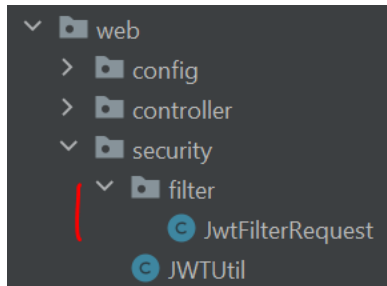
    public String extractUsername(String token) {
        return getClaims(token).getSubject();
    }

    public boolean isTokenExpired(String token) {
        return getClaims(token).getExpiration().before(new Date());
    }
}

```

```
// claims => objetos del jwt
private Claims getClaims(String token) {
    return Jwts.parser().setSigningKey(KEY).parseClaimsJws(token).getBody();
}
}
```

Creando un filtro para las peticiones (peticiones con cabecera “Authentication” y valor enviado “Bearer [TOKEN]”):



```
package com.danicode.market.web.security.filter;

import com.danicode.market.domain.service.DanicodeUserDetailsService;
import com.danicode.market.web.security.JWTUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.security.web.authentication.WebAuthenticationDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Component
public class JwtFilterRequest extends OncePerRequestFilter {

    @Autowired
    private JWTUtil jwtUtil;

    @Autowired
    private DanicodeUserDetailsService danicodeUserDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
    IOException {
        // Verificar el encabezado (que sea un token) => Bearer TOKEN
        String authorizationHeader = request.getHeader("Authorization");
        if (authorizationHeader != null && authorizationHeader.startsWith("Bearer")) {
            String jwt = authorizationHeader.substring(7);
            // Verificar usuario del jwt
            String username = this.jwtUtil.extractUsername(jwt);
            if (username != null && SecurityContextHolder.getContext().getAuthentication() ==
null) {
                UserDetails userDetails =
this.danicodeUserDetailsService.loadUserByUsername(username);
                if (this.jwtUtil.validateToken(jwt, userDetails)) {
                    UsernamePasswordAuthenticationToken authToken =
                        new UsernamePasswordAuthenticationToken(userDetails,
                            null, userDetails.getAuthorities());
                    authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                    SecurityContextHolder.getContext().setAuthentication(authToken);
                }
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

```

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

```

Modificando la configuración de seguridad:

```

package com.danicode.market.web.security;

import com.danicode.market.domain.service.DanicodeUserDetailsService;
import com.danicode.market.web.security.filter.JwtFilterRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    // Usar las credenciales creadas en la clase DanicodeUserDetailsService
    // Sobreescribir configure(AuthenticationManagerBuilder auth)
    @Autowired
    private DanicodeUserDetailsService danicodeUserDetailsService;

    @Autowired
    private JwtFilterRequest jwtFilterRequest;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(danicodeUserDetailsService);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // PERMITIR Todas las peticiones que terminen en authenticate
        // Las otras peticiones, necesitan autenticarse => anyRequest().authenticated()
        http.csrf().disable().authorizeRequests()
            .antMatchers("/**/authenticate").permitAll()
            .anyRequest().authenticated()
            .and()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        http.addFilterBefore(this.jwtFilterRequest,
            UsernamePasswordAuthenticationFilter.class);
    }

    // Incluir AuthenticationManager con anotación @Bean para inyectarlo y usarlo
    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}

```

➔ Desplegar nuestra API desde la ventana de comandos

```
java -jar platzi-market-1.0.jar
```

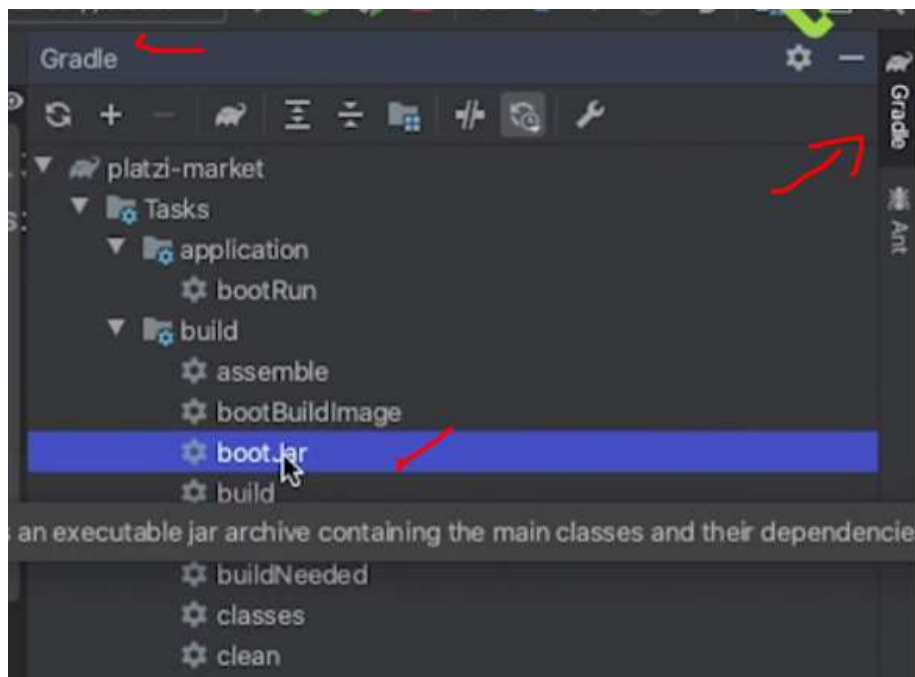
### Algunas propiedades adicionales

- -Xmx2048m
- -Dspring.profiles.active=pdn
- -Dserver.port=88

Cambiar la versión a 1.0 en el build.gradle

```
group = 'com.platzi'
version = '1.0' ✓
sourceCompatibility = '11'
```

Generar boot.jar:



En la terminal, en la carpeta del proyecto (usando las propiedades de producción):

```
$$> java -jar -Dspring.profiles.active=pdn build/libs/market-1.0.jar
```