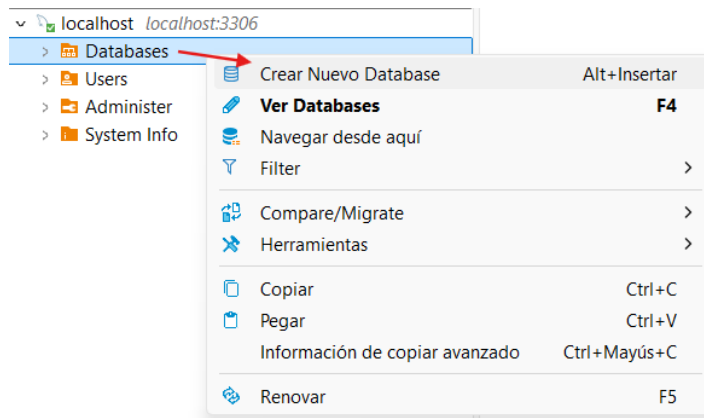


Configuraciones iniciales del entorno

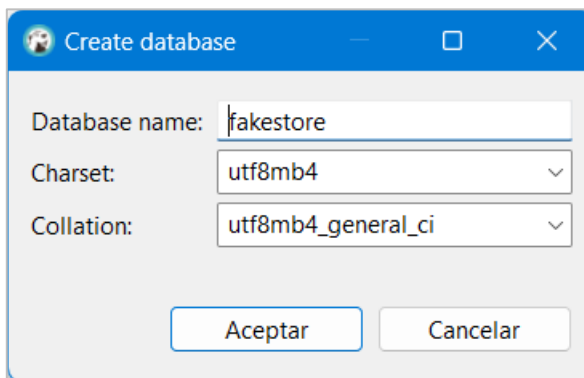
- Descarga de la plataforma virtual el respaldo y colócalo en una ubicación accesible.
- Abre **Laragon** e inicia el servidor de MySQL
- Abre **DBeaver** crea la sesión y conéctate a la instancia de Laragon

Restauración de base de datos

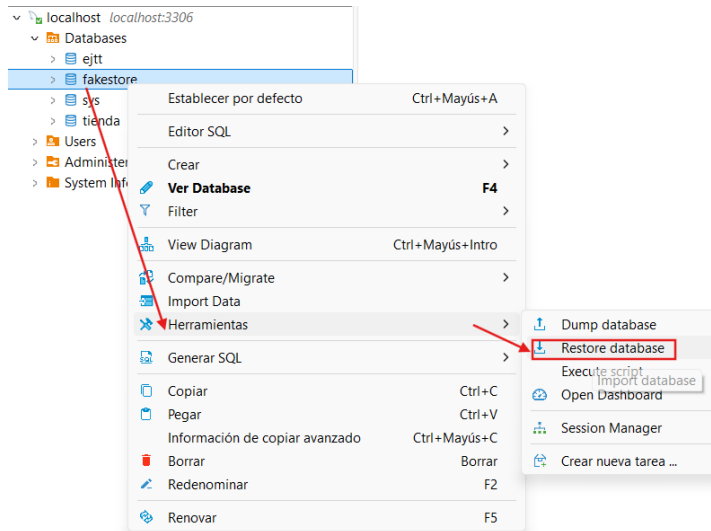
- Da clic derecho sobre la carpeta **Databases** y selecciona la opción **Crear Nuevo Database**.



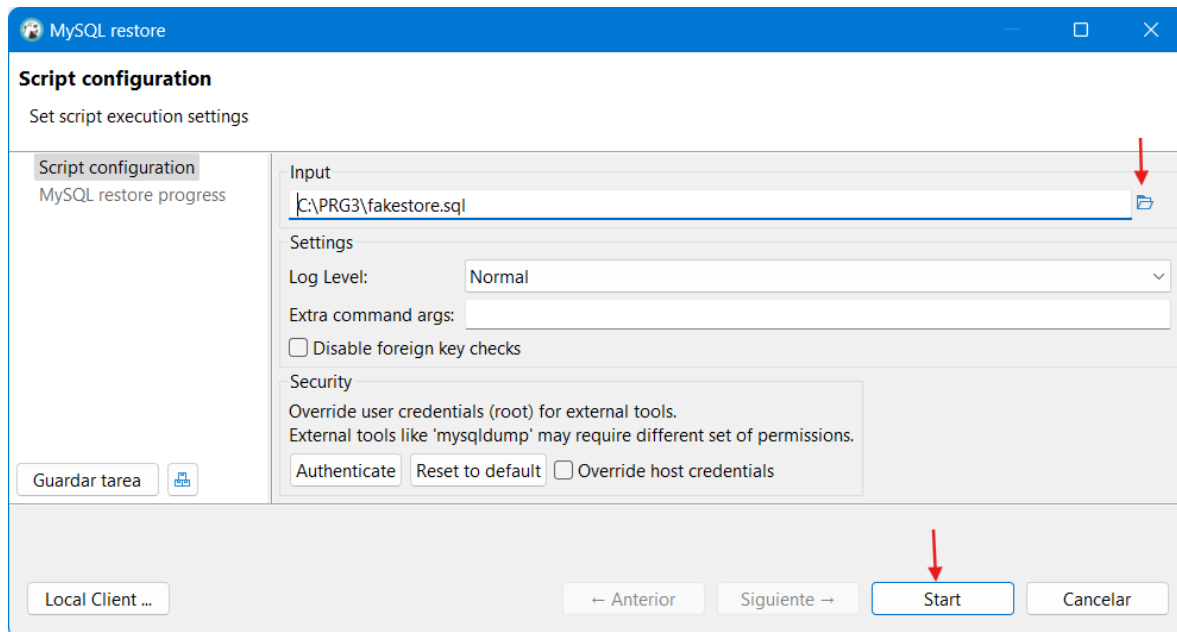
- En la ventana emergente coloca **fakestore** como nombre para la base de datos, selecciona **utf8mb4** como charset y **utf8mb4_general_ci** como collation.
- Luego da clic en aceptar



- Ahora da clic derecho sobre el nombre de la base de datos recién creada, selecciona **Herramientas** y luego **Restore database**



- En la ventana emergente utiliza el **botón de folder** para buscar la ubicación del respaldo que descargaste en el primer paso.



- Da clic en **start** para que inicie el proceso de restauración.
- Si todo marcha bien ya puedes cerrar esa ventana y si verificas tu base de datos ya tienes las tablas su respectiva data.

Creación del proyecto con Spring Initializr


- Ingresa a <https://start.spring.io> en tu navegador para crear el proyecto de Spring Boot
- Realiza las configuraciones tal como se muestra en la imagen, es importante ya que será la base de nuestro proyecto backend.
- **IMPORTANTE:** Debes sustituir “DS100125” por tu número de carnet

The screenshot shows the Spring Initializr web application interface. At the top, there's a navigation bar with the Spring logo and the text 'spring initializr'. Below this, the configuration options are organized into sections:

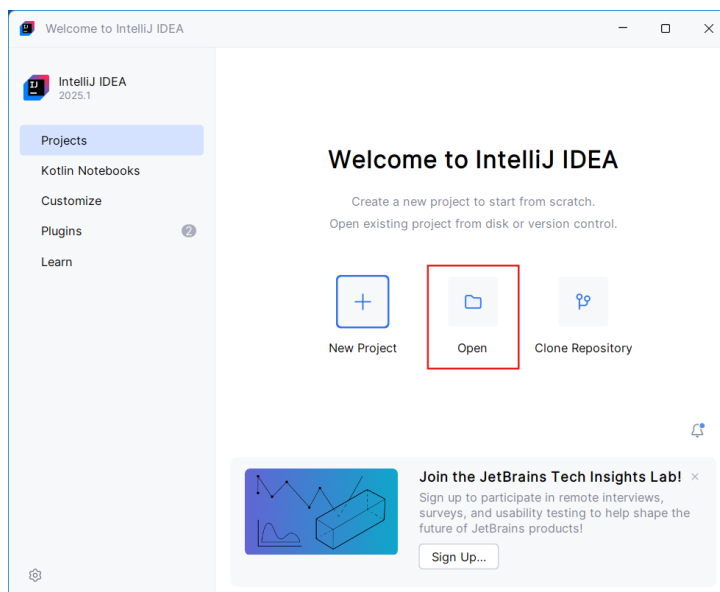
- Project:** Includes radio buttons for 'Gradle - Groovy', 'Gradle - Kotlin', 'Java' (selected), 'Kotlin', and 'Groovy'. There is also a 'Maven' option.
- Spring Boot:** Includes radio buttons for '3.5.0 (SNAPSHOT)', '3.5.0 (RC1)', '3.4.6 (SNAPSHOT)', '3.4.5' (selected), '3.3.12 (SNAPSHOT)', and '3.3.11'.
- Project Metadata:** Includes input fields for 'Group' (sv.edu.ufg), 'Artifact' (backend-DS100125), 'Name' (backend-DS100125), 'Description' (Backend para el proyecto ecommerce de PRG3), and 'Package name' (sv.edu.ufg.backend-DS100125).
- Packaging:** Includes radio buttons for 'Jar' (selected) and 'War'.
- Java:** Includes radio buttons for '24', '21' (selected), and '17'.

- Ahora agregaremos las dependencias necesarias para comenzar el proyecto.
- En el panel lateral de dependencias utiliza el buscador y agrega las siguientes:
 - Spring Web
 - Spring Data JPA
 - Lombok
 - MySQL Driver

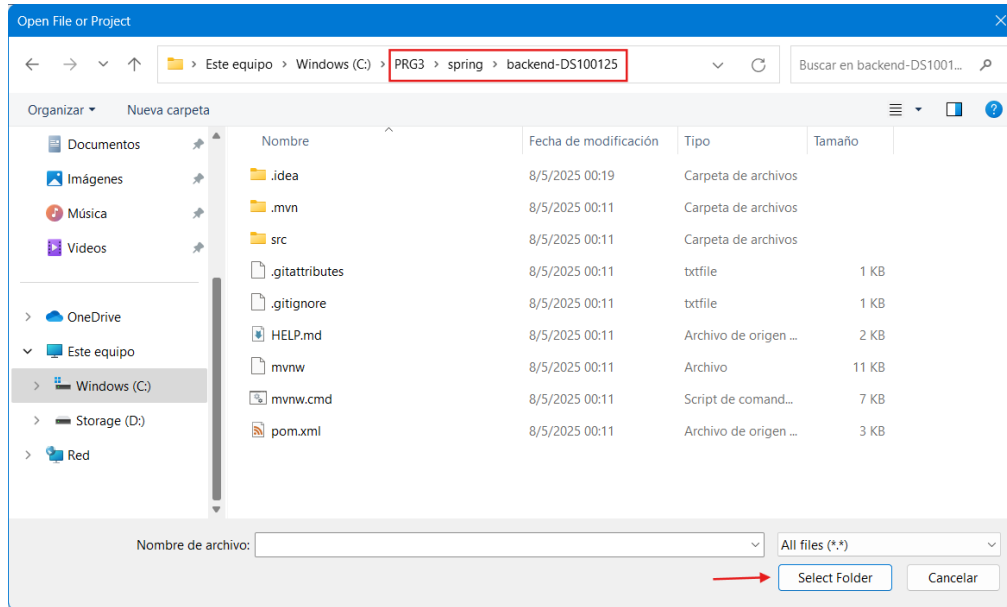
- Una vez agregadas da clic en el botón **Generate**
- Comenzará el proceso de descarga de tu proyecto de Spring Boot.

 **backend-DS100125.zip**
16,3 KB • Hecho

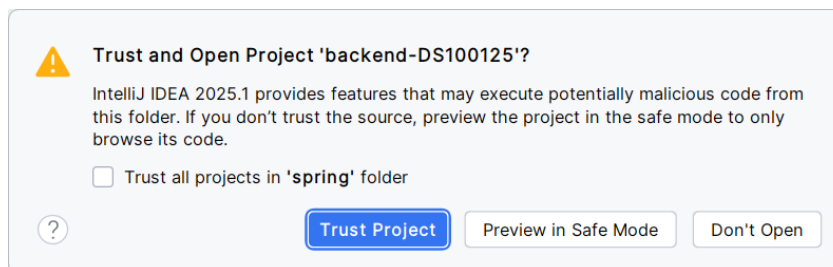
- Busca el archivo que descargaste, córtalo y pégalo en una ubicación accesible (de preferencia en C:/PRG3/spring)
- Descomprime el proyecto.
- Abre **IntelliJ Idea Community** y selecciona la opción Open (Abrir)



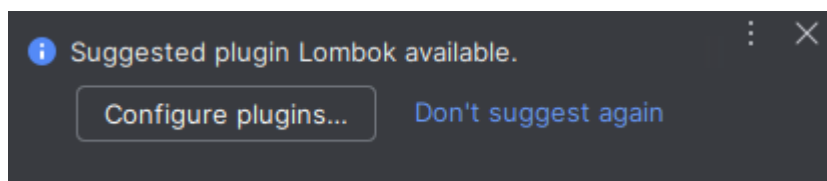
- En la ventana emergente busca la ubicación de la carpeta de tu proyecto



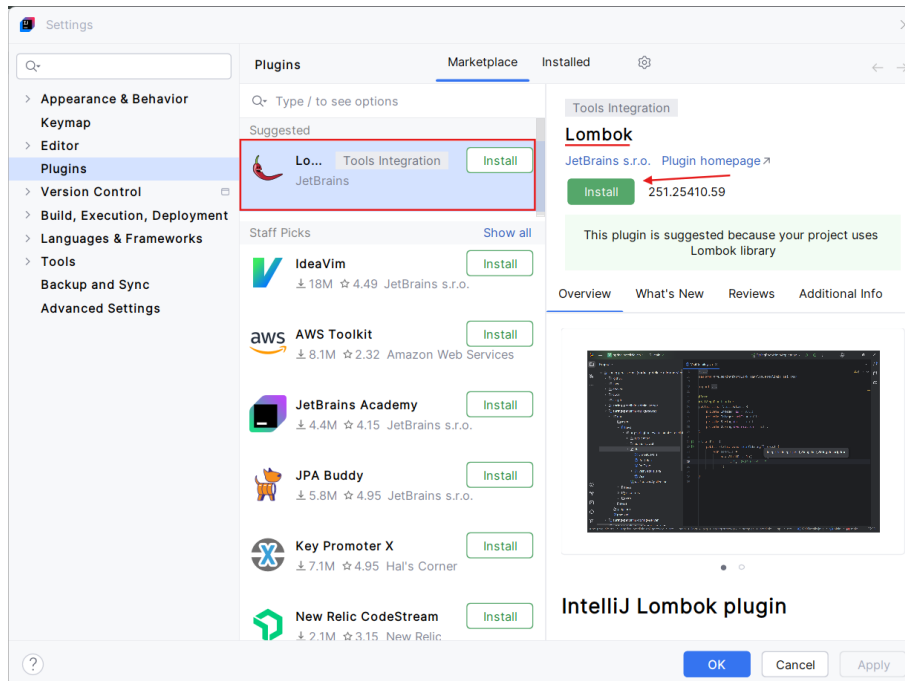
- Si te muestra un mensaje de confirmación de confianza en el autor del proyecto selecciona la opción **Trust Project**



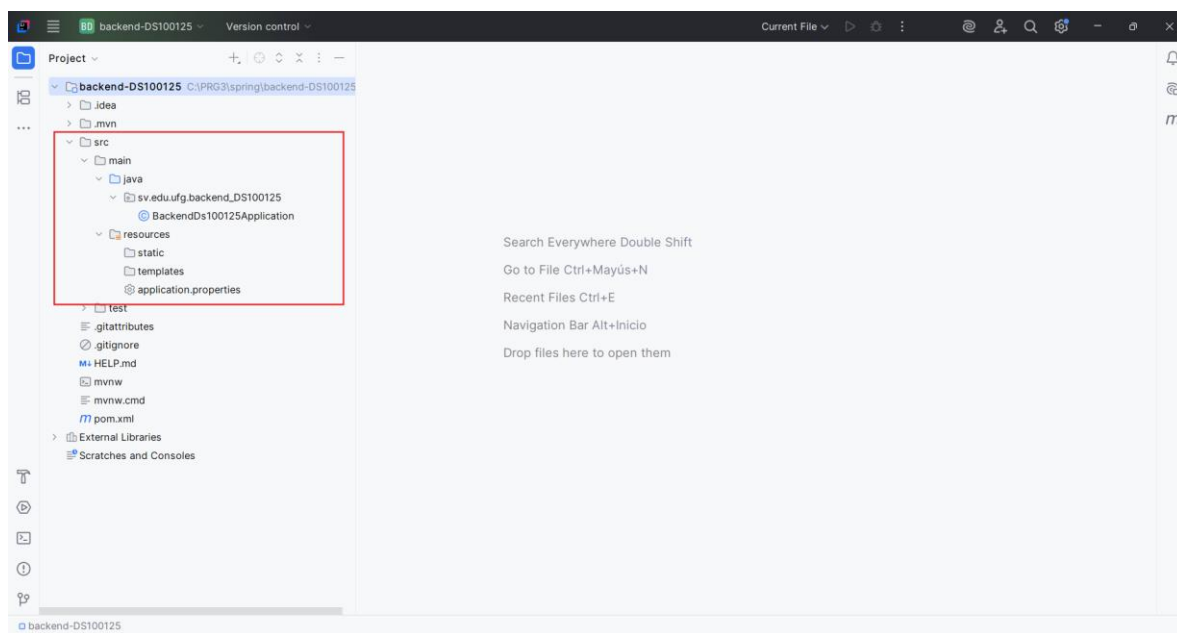
- Espera a que el proyecto sea reconstruido basado en las dependencias que seleccionamos en Spring Inicializar
- Si en el proceso de reconstrucción te muestra una advertencia de instalación del plugin de Lombok da clic en el botón **Configure plugins...**



- En la ventana emergente da clic sobre el botón **Install** para el plugin de **Lombok** y al finalizar la instalación clic en **Ok**



- Cuando el proceso de reconstrucción termine deberás poder acceder a los archivos de tu carpeta **src** en el panel lateral izquierdo.



Configuración de Application Properties

- Abre tu archivo **application.properties** que se encuentra en la carpeta **resources**.
- Ingresa las siguientes configuraciones (No es necesario que agregues las líneas de comentarios #).
- Recuerda sustituir donde dice DS100125 por tu número de carnet.

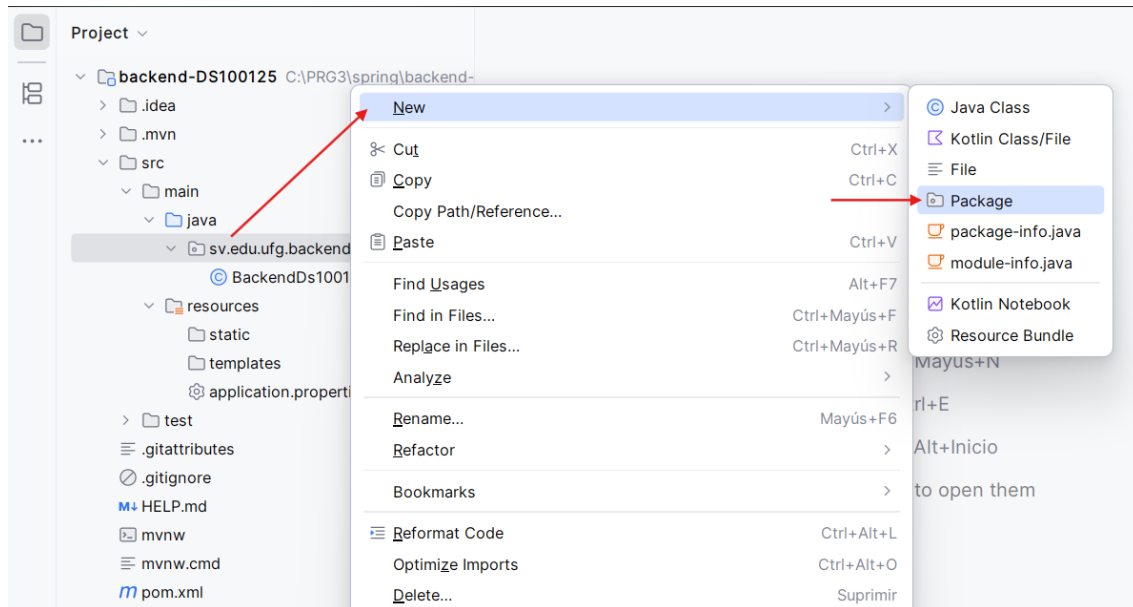
```
application.properties x
1  # Nombre de la aplicación Spring Boot
2  spring.application.name=backend-DS100125
3
4  # Configuración del DataSource para la conexión a la base de datos MySQL
5  spring.datasource.url=jdbc:mysql://localhost:3306/fakestore
6  spring.datasource.username=root
7  spring.datasource.password=
8
9  # Especificación del nombre de la clase del driver JDBC de MySQL
10 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
11
12 # Propiedades de Hibernate
13 # Dialecto de Hibernate para MySQL
14 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
15
16 # Estrategia de creación de tablas de Hibernate:
17 # - "update": Actualiza la estructura de la base de datos según las entidades
18 # definidas en el código. No elimina datos existentes.
19 spring.jpa.hibernate.ddl-auto=update
20
21 # Muestra las sentencias SQL generadas por Hibernate en la consola
22 spring.jpa.show-sql=true
```

- En el caso de que no se aprecien correctamente las líneas de configuración puedes basarte en el siguiente Gist:

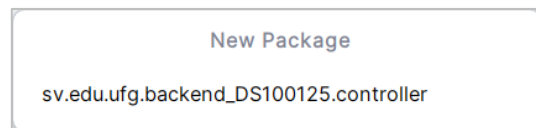
[GIST - application.properties](#)

Creación de estructura de paquetes

- Da clic derecho sobre el nombre del paquete principal de tu proyecto `sv.edu.ufg.backend_DS100125`
- Selecciona **New** del menú emergente y luego **Package**



- Agrégale la palabra **controller** en la ventanita emergente y da enter

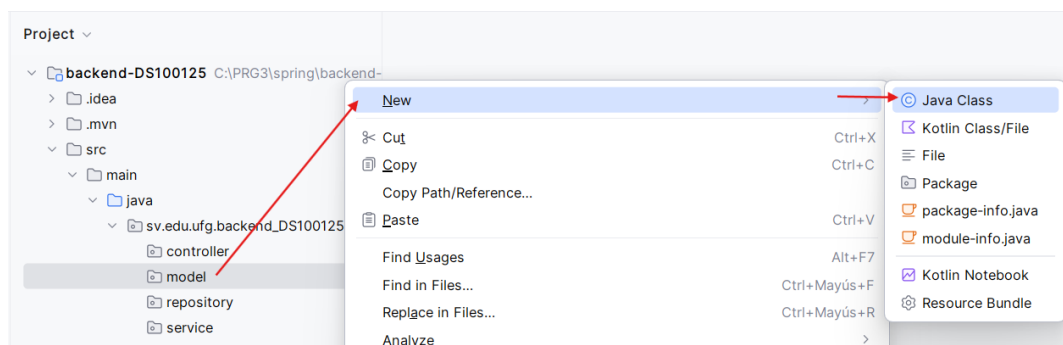


- Repite el mismo proceso y crea los paquetes:

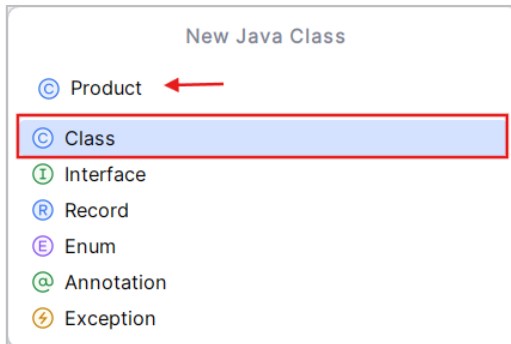
- model
- repository
- service

Creación de la capa de modelo (Entidad)

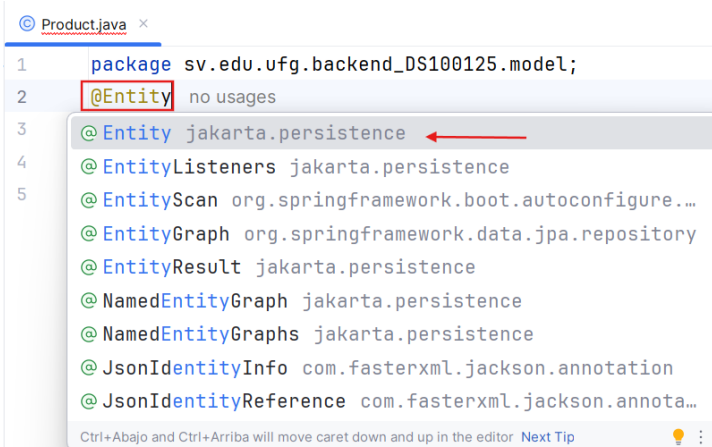
- Da clic derecho sobre el paquete **model** y agrega una nueva clase



- Colócale como nombre **Product**



- Primero que todo, agrega la anotación **@Entity** al inicio de la clase y asegúrate de importarla de `jakarta.persistence.Entity`;

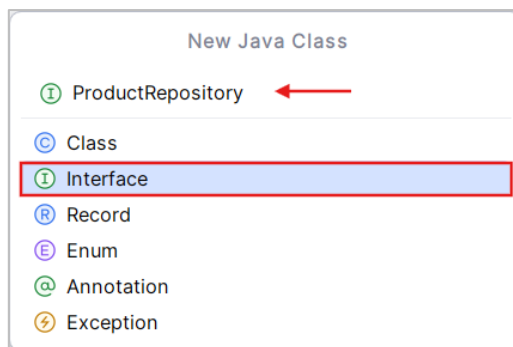
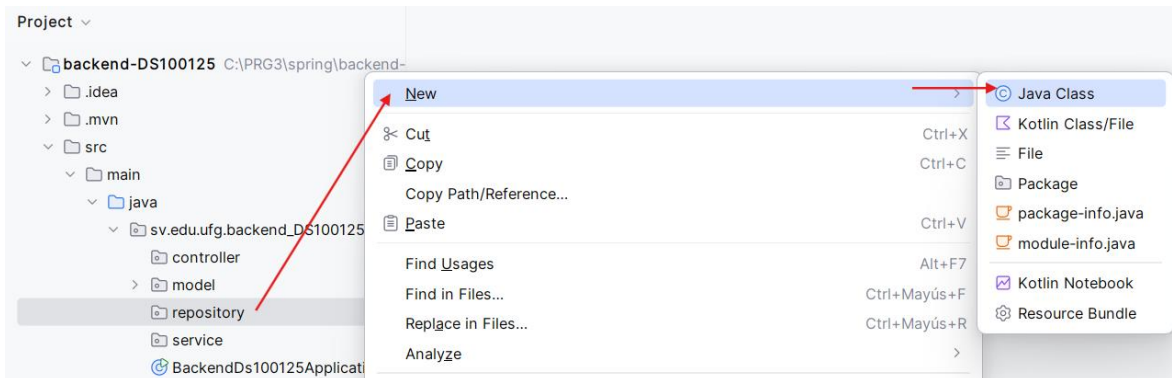


- Agrega los atributos de clase con sus respectivas anotaciones, recuerda que estos atributos deben nombrarse de la misma manera que los campos en tu tabla products.
- Asegúrate de que se realicen las importaciones de Lombok

```
1 package sv.edu.ufg.backend_DS100125.model;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7 import lombok.AllArgsConstructor;
8 import lombok.Data;
9 import lombok.NoArgsConstructor;
10
11 @Data
12 @NoArgsConstructor
13 @AllArgsConstructor
14 @Entity
15 public class Product {
16
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Integer id; // Identificador de la tabla
20
21     // Otros atributos
22     private String title;
23     private Double price;
24     private String description;
25     private String category;
26     private String image;
27     private Double rating_rate;
28     private Integer rating_count;
29 }
```

Creación del repositorio

- Da clic derecho sobre el paquete **repository** y crea una nueva interface **ProductRepository**



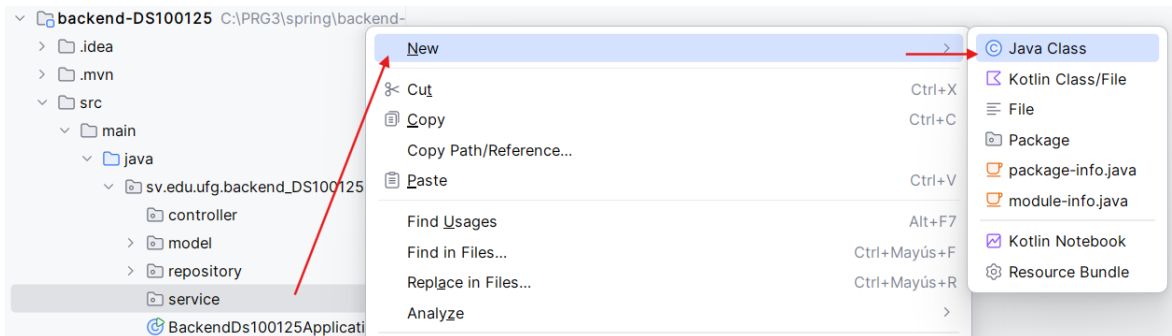
- Has que la interface herede de **JpaRepository**, agrégle la entidad **Product** y el tipo de datos **Integer** como parámetros

```
1 package sv.edu.ufg.backend_DS100125.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import sv.edu.ufg.backend_DS100125.model.Product;
5
6 public interface ProductRepository extends JpaRepository<Product, Integer> {
7 }
```

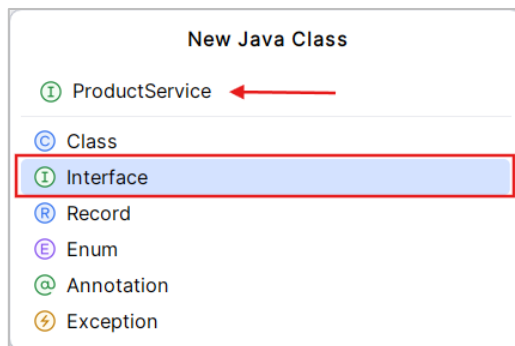
- **Integer** viene del tipo de datos del campo id de la entidad.
- Asegúrate de realizar los imports necesarios.

Creación del servicio

- Da clic derecho sobre el paquete **service** y selecciona la opción para crear una nueva clase



- Colócale como nombre **ProductService** y selecciona **Interface** como tipo de clase a crear



- En esta interface crearemos las definiciones de los métodos que utilizaremos más adelante pero no sus implementaciones.

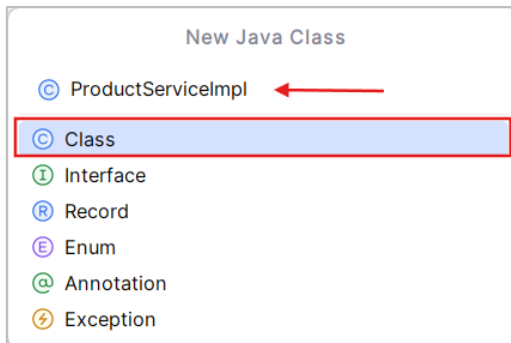
```

1 package sv.edu.ufg.backend_DS100125.service;
2
3 import sv.edu.ufg.backend_DS100125.model.Product;
4
5 import java.util.List;
6
7 public interface ProductService {
8     List<Product> getAllProducts();
9     Product getProductById(Integer id);
10    Product saveProduct(Product product);
11    void deleteProduct(Integer id);
12 }

```

- Ahora crearemos una clase para implementar esos métodos.

- Da clic derecho sobre el paquete **service** y crea una nueva clase, colócale como nombre **ProductServiceImpl**.

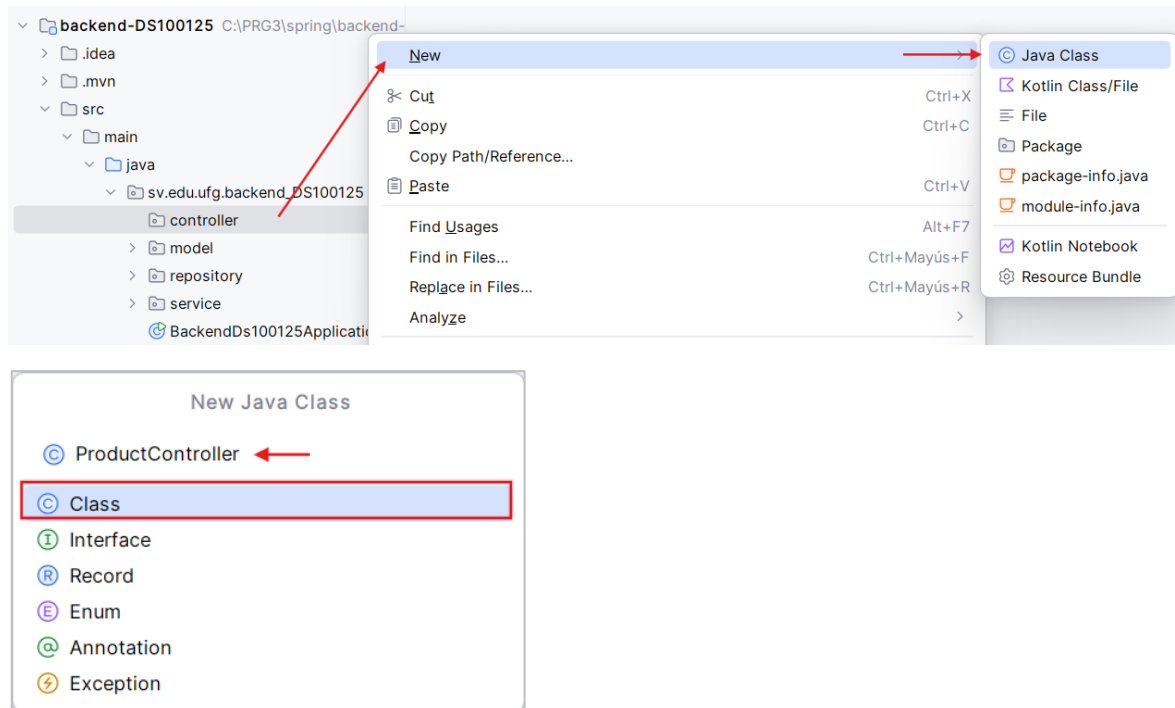


- Esta clase debe de estar anotada con **@Service**
- Debe de implementar de **ProductService**
- Asegúrate que colocas la anotación **@Autowired** para hacer la inyección de dependencias del servicio
- Se deben sobrecargar los métodos de ProductService utilizando la anotación **@Override**
- Asegúrate que los nombres de los métodos coinciden con las definiciones en ProductService
- Verifica los imports en la parte superior

```
1 package sv.edu.ufg.backend_DS100125.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import sv.edu.ufg.backend_DS100125.model.Product;
6 import sv.edu.ufg.backend_DS100125.repository.ProductRepository;
7
8 import java.util.List;
9 import java.util.Optional;
10
11 @Service
12 public class ProductServiceImpl implements ProductService {
13     @Autowired
14     private ProductRepository productRepository;
15
16     @Override
17     public List<Product> getAllProducts() {
18         return productRepository.findAll();
19     }
20
21     @Override
22     public Product getProductById(Integer id) {
23         Optional<Product> productOptional = productRepository.findById(id);
24         return productOptional.orElse(null);
25     }
26
27     @Override
28     public Product saveProduct (Product product) {
29         return productRepository.save(product);
30     }
31
32     @Override
33     public void deleteProduct(Integer id) {
34         productRepository.deleteById(id);
35     }
36 }
```

Creación del controlador

- Da clic derecho sobre el paquete **controller** y crea una nueva clase de nombre **ProductController**



- Es primordial que agregues las anotaciones siguientes al inicio de la clase
 - `@RestController`
 - `@RequestMapping("/api/product")`
- Asegúrate que se realicen los imports respectivos para cada anotación
- Luego coloca las implementaciones de los métodos configurados en tu servicio, el cual debe estar agregado por medio de la inyección de dependencias de la anotación `@Autowired`

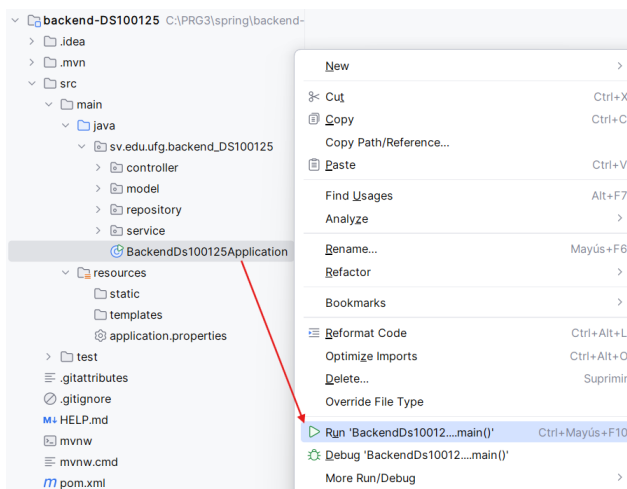
```

1  package sv.edu.ufg.backend_DS100125.controller;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.web.bind.annotation.GetMapping;
5  import org.springframework.web.bind.annotation.PathVariable;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RestController;
8  import sv.edu.ufg.backend_DS100125.model.Product;
9  import sv.edu.ufg.backend_DS100125.service.ProductService;
10 import java.util.List;
11
12 @RestController
13 @RequestMapping("/api/product")
14 public class ProductController {
15     @Autowired
16     private ProductService productService;
17
18     @GetMapping
19     public List<Product> getAllProducts() {
20         return productService.getAllProducts();
21     }
22     @GetMapping("/{id}")
23     public Product getProductById(@PathVariable Integer id) {
24         return productService.getProductById(id);
25     }
26 }

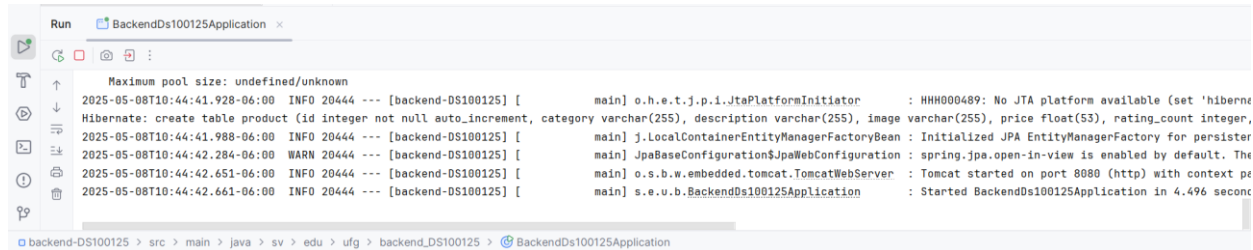
```

Ejecutando nuestro servicio REST

- Da clic derecho sobre tu clase principal y selecciona la opción **Run** del menú emergente



- Si has realizado correctamente los pasos de la guía, en este punto tu servidor debe levantar y publicar tu servicio REST

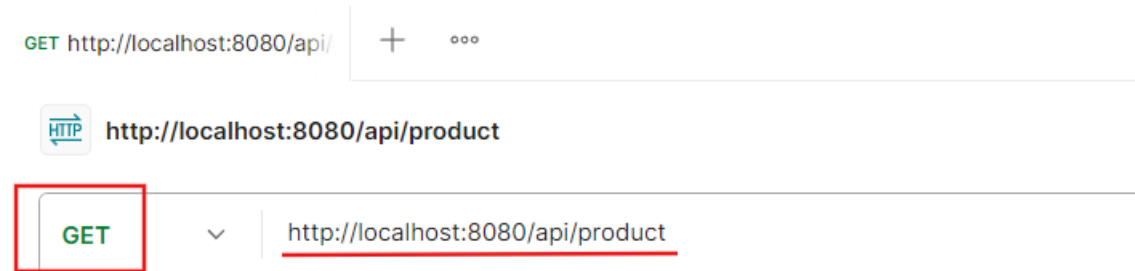


```

Maximum pool size: undefined/unknown
2025-05-08T10:44:41.928-06:00 INFO 20444 --- [backend-DS100125] [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate
Hibernate: create table product (id integer not null auto_increment, category varchar(255), description varchar(255), image varchar(255), price float(53), rating_count integer,
2025-05-08T10:44:41.988-06:00 INFO 20444 --- [backend-DS100125] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persisten
2025-05-08T10:44:42.284-06:00 WARN 20444 --- [backend-DS100125] [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. The
2025-05-08T10:44:42.651-06:00 INFO 20444 --- [backend-DS100125] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context pe
2025-05-08T10:44:42.661-06:00 INFO 20444 --- [backend-DS100125] [main] s.e.u.b.BackendDs100125Application : Started BackendDs100125Application in 4.496 second
  
```

Verifica el funcionamiento del servicio

- Utiliza Postman, puedes descargarlo de <https://www.postman.com/downloads/>
- La instalación es sencilla, sigue los pasos del asistente
- Una vez instalado abre Postman y crea una nueva solicitud HTTP usando GET



- Da clic sobre el botón **Send**
- Verifica que funcione tanto el end-point **product** como **product/1**

- <http://localhost:8080/api/product>

Home Workspaces Explore Search Postman Sign In Create Account

GET http://localhost:8080/api/ +

http://localhost:8080/api/product Save </>

GET http://localhost:8080/api/product Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Bulk Edit
Key	Value	

Body Cookies Headers (5) Test Results Status: 200 OK Time: 24 ms Size: 10.6 KB Save Response

Pretty Raw Preview Visualize JSON Bulk Edit

```

1 {
2   {
3     "id": 1,
4     "title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",
5     "price": 109.95,
6     "description": "Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday",
7     "category": "men's clothing",
8     "image": "https://fakestoreapi.com/img/81fPKd-2AYL_AC_SL1500_.jpg",
9     "rating_rate": 3.9,
10    "rating_count": 120
11  },
12  {
13    "id": 2,
14    "title": "Mens Casual Premium Slim Fit T-Shirts",
15    "price": 22.3,
16    "description": "Slim-fitting style, contrast raglan long sleeve, three-button henley placket, light weight & soft fabric for breathable and comfortable wearing. And Solid stitched shirts with round neck made for durability and a great fit for casual fashion wear and diehard baseball fans. The Henley style round neckline includes a three-button placket.",
17  }
18 }
  
```

Console Not connected to a Postman account

- <http://localhost:8080/api/product/1>

Home Workspaces Explore Search Postman Sign In Create Account

GET http://localhost:8080/api/ +

http://localhost:8080/api/product/1 Save </>

GET http://localhost:8080/api/product/1 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Bulk Edit
Key	Value	

Body Cookies Headers (5) Test Results Status: 200 OK Time: 9 ms Size: 531 B Save Response

Pretty Raw Preview Visualize JSON Bulk Edit

```

1 {
2   "id": 1,
3   "title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",
4   "price": 109.95,
5   "description": "Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday",
6   "category": "men's clothing",
7   "image": "https://fakestoreapi.com/img/81fPKd-2AYL_AC_SL1500_.jpg",
8   "rating_rate": 3.9,
9   "rating_count": 120
10 }
  
```

Console Not connected to a Postman account



- Para la entrega debes comprimir **todos** los archivos y carpetas del proyecto generado con spring boot
- **No olvides respaldar este proyecto ya que sobre el mismo continuaras trabajando en las prácticas posteriores.**