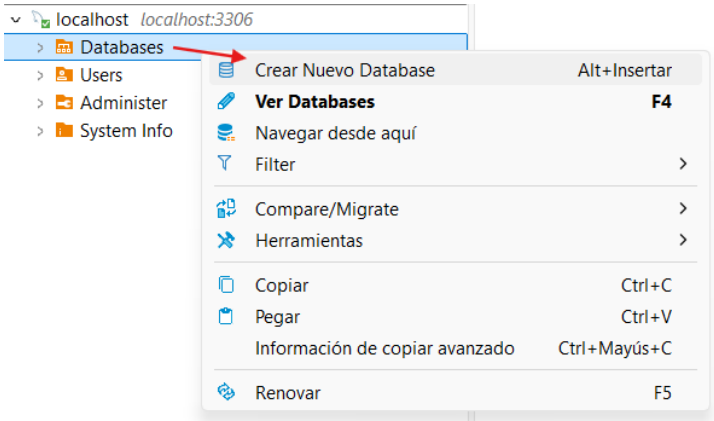
Configuraciones iniciales del entorno (Solo si estas trabajando en las maquinas del centro de cómputo)

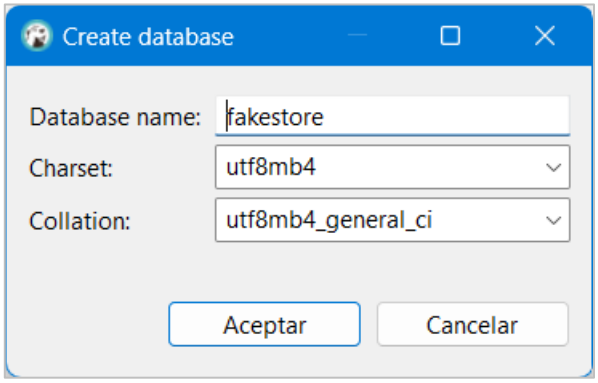
- Descarga de la plataforma virtual el respaldo de la base de datos y colócalo en una ubicación accesible.
- Abre **Laragon** e inicia el servidor de MySQL
- Abre **DBeaver** crea la sesión y conéctate a la instancia de Laragon

Restauración de base de datos

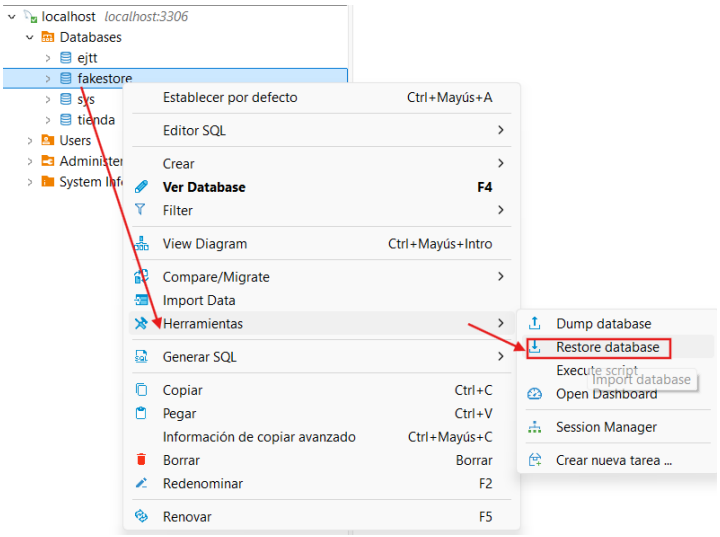
- Da clic derecho sobre la carpeta **Databases** y selecciona la opción **Crear Nuevo Database**.



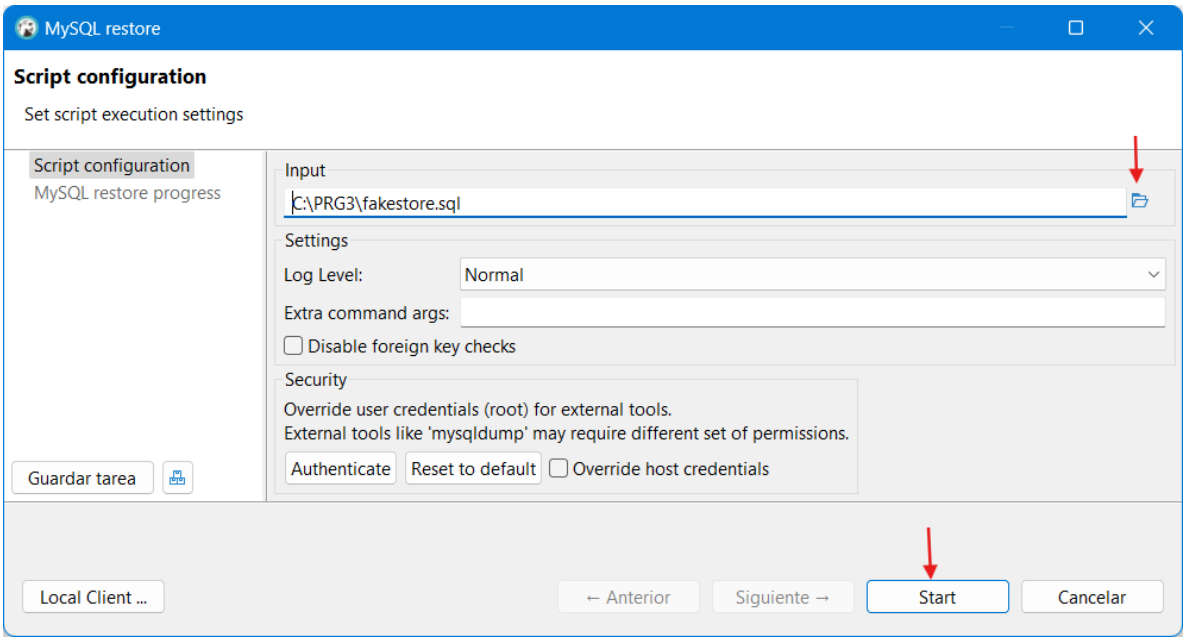
- En la ventana emergente coloca **fakestore** como nombre para la base de datos, selecciona **utf8mb4** como charset y **utf8mb4_general_ci** como collation.
- Luego da clic en aceptar



- 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.

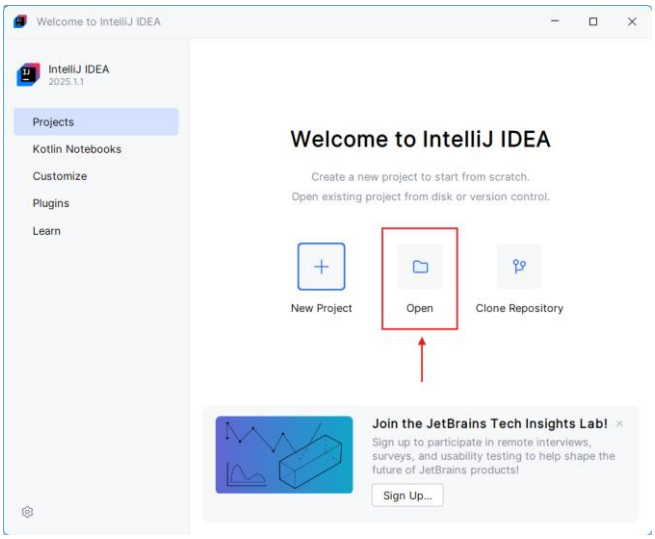


- Da clic en **start** para que inicie el proceso de restauración.
- Si todo marcha bien ya puedes cerrar esa ventana y si verificas tu base de datos ya tienes las tablas su respectiva data.

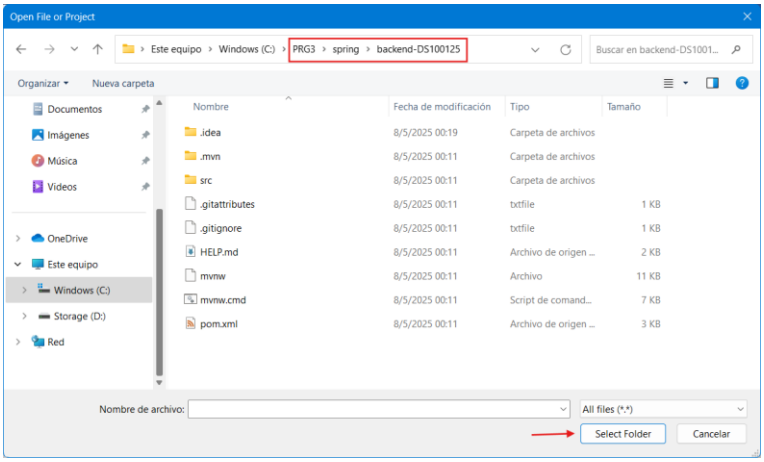
Modificación del proyecto back-end

- Vamos a utilizar el mismo proyecto que generamos la semana anterior.
  - Si estas trabajando en las maquinas del centro de cómputo descarga el respaldo de tu proyecto:
    - Busca el archivo que descargaste, córtalo y pégalo en una ubicación accesible (de preferencia en C:/PRG3/spring)
    - Descomprime el proyecto

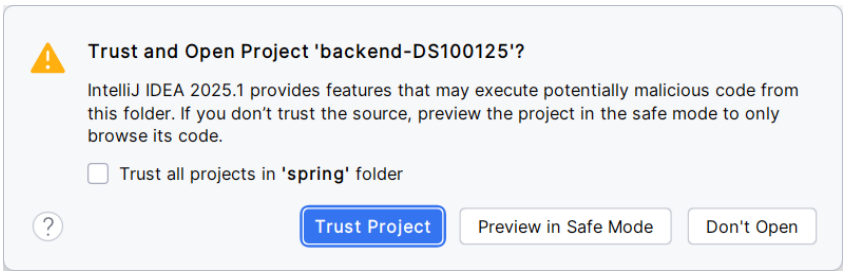
- Abre **IntelliJ Idea Community** y selecciona la opción Open.



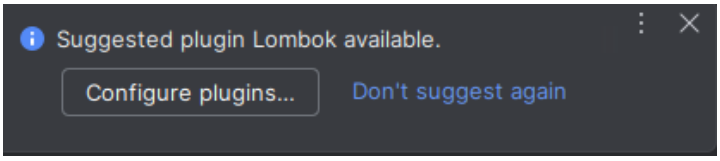
- Busca la ubicación de la carpeta de tu proyecto



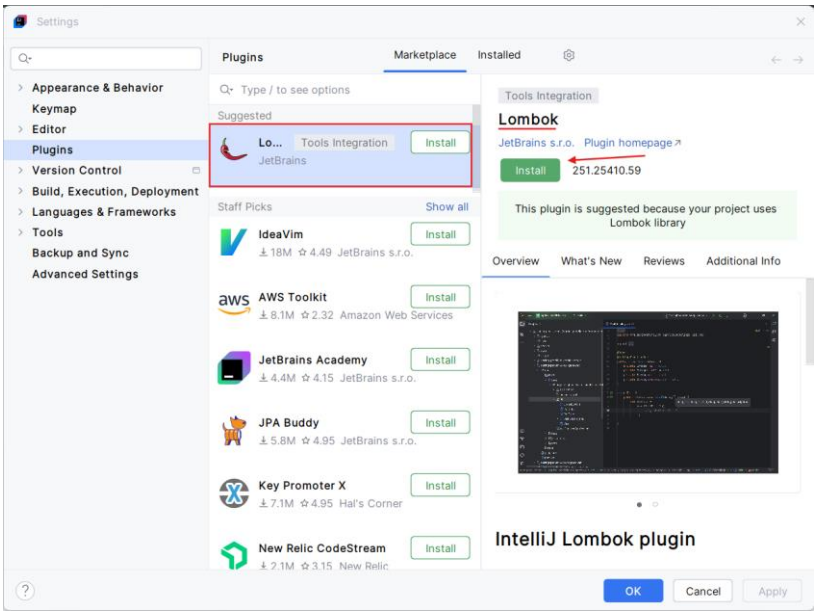
- Si te muestra un mensaje de confirmación de confianza en el autor del proyecto selecciona la opción **Trust Project**



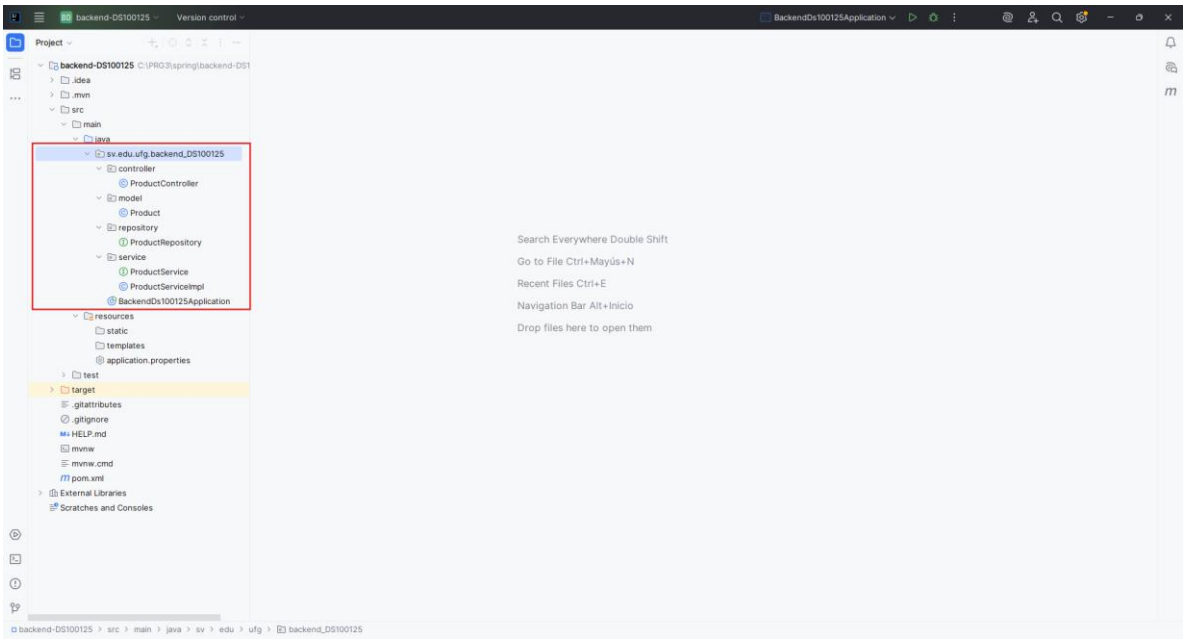
- Espera a que el proyecto sea reconstruido basado en las dependencias configuradas en el archivo **pom.xml**
- Si en el proceso de reconstrucción te muestra una advertencia de instalación del plugin de Lombok da clic en el botón **Configure plugins...**



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

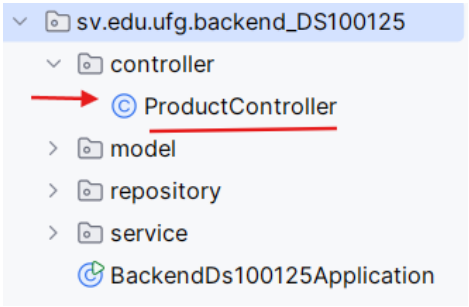


- Cuando el proceso de reconstrucción termine deberás poder acceder a los archivos de tu carpeta **src** en el panel lateral izquierdo.



Modificación del controlador

- En el paquete **controller**, abre la clase de nombre **ProductController**

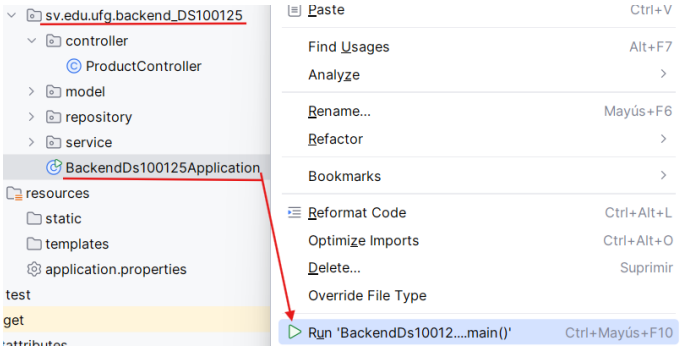


- Adiciona las implementaciones de los métodos configurados en tu servicio, el cual debe estar agregado por medio de la inyección de dependencias de la anotación **@Autowired**
- Agrega un método anotado con **@PostMapping** para poder *crear un nuevo registro* en la tabla de productos. Este método debe solicitar por parámetro la data para crear el producto con la anotación **@RequestBody** y retornará un JSON con la información del producto registrado.
- Agrega un método anotado con **@PutMapping("/{id}"),** que permitirá la *edición de un registro* de la tabla de productos y retornará un JSON con la información del producto modificado.
- También agrega un método para *eliminar un registro* de la tabla de productos, este método debe estar anotado con **@DeleteMapping("/{id}")**

```java
package sv.edu.ufg.backend_DS100125.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import sv.edu.ufg.backend_DS100125.model.Product;
import sv.edu.ufg.backend_DS100125.service.ProductService;

import java.util.List;

@RestController
@RequestMapping("/api/product")
public class ProductController {
    @Autowired
    private ProductService productService;

    @GetMapping
    public List<Product> getAllProducts() {
        System.out.println("Obteniendo todos los productos");
        return productService.getAllProducts();
    }
    @GetMapping("/{id}")
    public Product getProductById(@PathVariable Integer id) {
        return productService.getProductById(id);
    }

    @PostMapping
    public Product createProducto(@RequestBody Product producto){
        return productService.saveProduct(producto);
    }

    @PutMapping("/{id}")
    public Product updateProducto(@PathVariable Integer id, @RequestBody Product producto){
        producto.setId(id);
        return productService.saveProduct(producto);
    }

    @DeleteMapping("/{id}")
    public void deleteProducto(@PathVariable Integer id){
        productService.deleteProduct(id);
    }
}
```

Ejecutando nuestro servicio REST

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



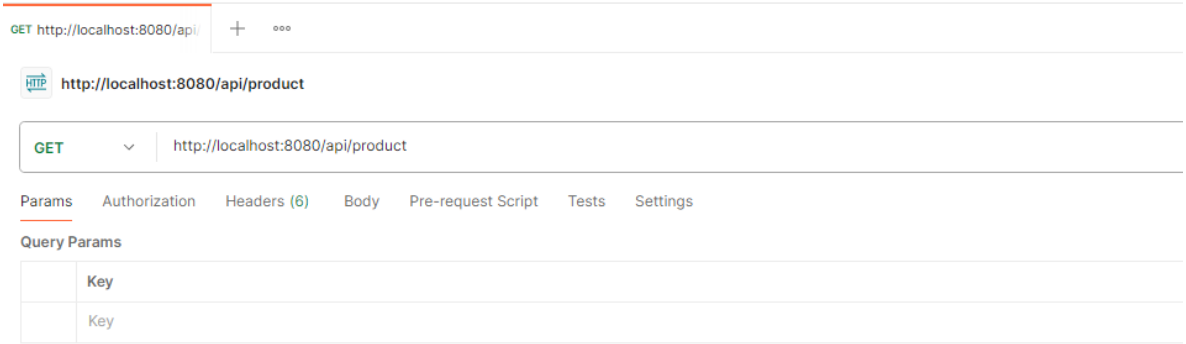
- Si has realizado correctamente la guía en este punto tu servidor debe levantar y publicar tu servicio REST

Verifica el funcionamiento del servicio

- Utiliza **Postman** y comprueba el funcionamiento de cada uno de los endpoints con los que ahora cuenta tu API Rest.

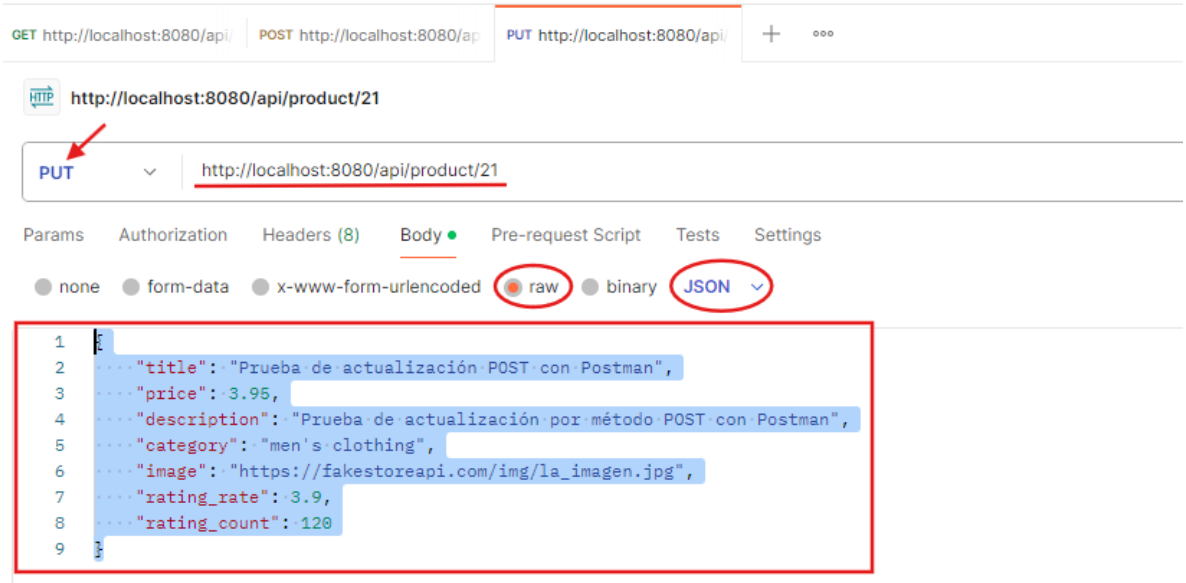
Recuerda que la URL base es: **localhost:8080/api/product**

- Para probar la petición **GET** puedes basarte en la siguiente configuración:

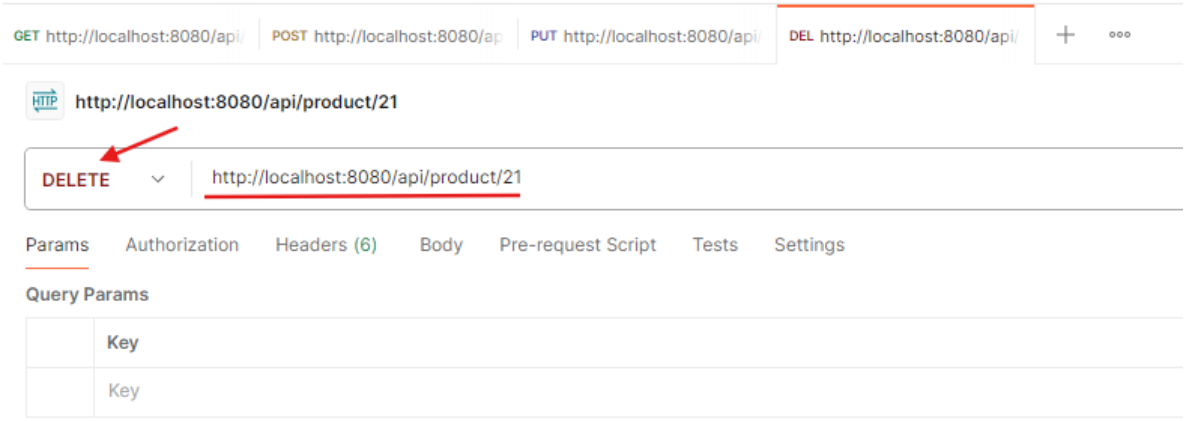


- Para probar la petición **POST** puedes basarte en la siguiente configuración:

- Para probar la petición **PUT** puedes basarte en la siguiente configuración:



- Para probar la petición **DELETE** puedes basarte en la siguiente configuración:



Modificación del proyecto con Angular

Solo si estas trabajando en las maquinas del centro de cómputo:

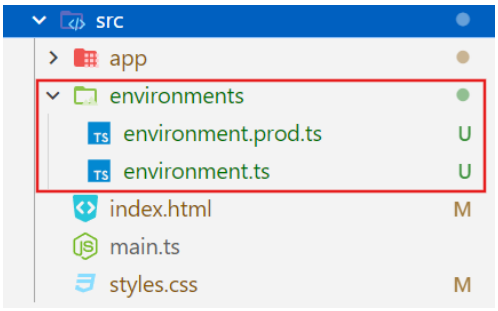
- Descarga tu proyecto del laboratorio 3 (el front-end hecho con Angular)
- Colócalo en una ubicación accesible (Puedes dejarlo incluso en la misma carpeta donde esta tu back-end, ejemplo: C:/PRG3/spring)
- Reconstruye tu carpeta node_modules de ser necesario (**npm i**)

Todos:

- Abre tu proyecto front-end (Angular) con VSCode
- Utiliza el generador de Angular para crear un nuevo archivo de interfaces (Ya sabes cómo hacerlo)
- Dentro del archivo que acabas de generar agrega la estructura de la interface basada en la estructura de campos de tu tabla de productos.
- Debe quedar de la siguiente forma:

```
1  export interface Interfaces{
2  }
3
4  export interface Product{
5      id: number;
6      title: string;
7      price: number;
8      description: string;
9      category: string;
10     image: string;
11     rating_rate: number;
12     rating_count: number;
13 }
```

- Crea un directorio de archivos de entorno en app/src/ colócale como nombre environments, dentro crea dos archivos uno con nombre environment.ts y el otro como environment.prod.ts, debería quedarte de esta forma:



- Modifica tu archivo app/src/environments/environment.ts, agregando la siguiente configuración:

```
export const environment = {
  production: false,
  debugMode: true,
  baseUrl: "http://localhost:8080/api/products",
  fakeUrl: "https://fakestoreapi.com/products/categories"
};

```

- Modifica tu archivo app/src/environments/environment.prod.ts, agregando la siguiente configuración:

```
export const environment = {
  production: true,
  debugMode: false,
  baseUrl: "http://localhost:8080/api/products",
  fakeUrl: "https://fakestoreapi.com/products/categories"
};

```

- Modifica tu archivo app/src/services/**api.service.ts** agregando los métodos según para comunicarse con el back-end, debe quedarte como el siguiente código:

- Asegúrate de que se agreguen los imports necesarios en la parte superior.

```typescript
import { HttpClient } from '@angular/common/http';
import { Injectable, inject } from '@angular/core';
import { Observable } from 'rxjs';
import { Product } from '../interfaces/interfaces';
import { environment } from '../environments/environment';

@Injectable({
    providedIn: 'root'
})
export class ApiService {
    private http = inject(HttpClient)

    constructor() { }

    getCategories() {
        return this.http.get(environment.fakeUrl)
    }

    getAllProducts(): Observable<Product[]> {
        return this.http.get<Product[]>(environment.baseUrl)
    }

    getProductById(id: number): Observable<Product> {
        return this.http.get<Product>(`${environment.baseUrl}/${id}`)
    }

    createProduct(product: Product): Observable<Product> {
        return this.http.post<Product>(environment.baseUrl, product)
    }

    updateProduct(id: number, product: Product): Observable<Product> {
        return this.http.put<Product>(`${environment.baseUrl}/${id}`, product)
    }

    deleteProduct(id: number): Observable<any> {
        return this.http.delete(`${environment.baseUrl}/${id}`)
    }
}

```

- Modifica tu archivo app/src/pages/**main.component.ts** agregando los métodos según los cambios realizados al service

```typescript
import { Component, OnInit, inject } from '@angular/core';
import { ApiService } from '../../services/api.service';
import { TitleCasePipe } from '@angular/common';
import { Product } from '../../interfaces/interfaces';

@Component({
  selector: 'app-main',
  standalone: true,
  imports: [ TitleCasePipe ],
  templateUrl: './main.component.html',
  styleUrl: './main.component.css'
})
export default class MainComponent implements OnInit {

  title = 'Productos'
  categories: any = []
  products: Product[] = []

  private apiService = inject(ApiService)

  ngOnInit(): void {
      this.loadCategories()
      this.loadProducts()
  }

  loadCategories(){
    this.apiService.getCategories().subscribe({
      next: (res: any) => {
        this.categories = res
      },
      error: (err: any) => {}
    })
  }

  loadProducts(){
    this.apiService.getProducts().subscribe({
      next: (res) => {
        this.products = res
      },
      error: (err: any) => {}
    })
  }
}

```

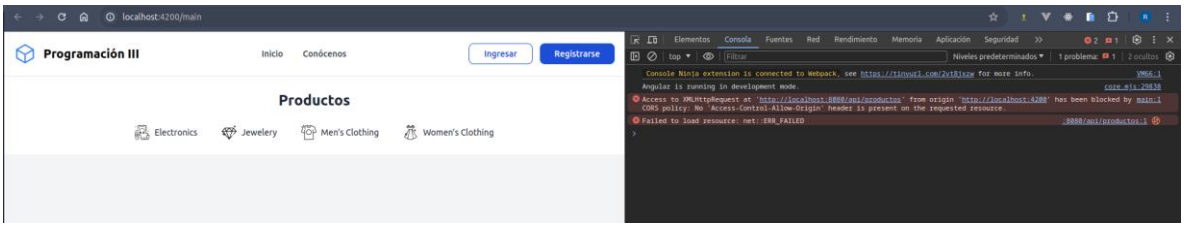
- Modifica tu archivo app/src/pages/**main.component.html** para no perder mucho tiempo en el diseño y la asignación de las reglas de tailwind, puedes basarte en el siguiente Gist:

HTML del componente Main

- Guarda todos los cambios ejecuta tu aplicacion front-end con: **ng s -o**

¿Notas algo raro?
¿Te cargan los productos desde tu back-end?

- Inspecciona tu aplicación desde el navegador y encontrarás cual es la causa:



Para corregir este comportamiento es necesario agregar en el back-end los permisos para que acepte peticiones desde ubicaciones externas al **localhost** o por puertos como el **4200**

- Regresa al controlador de tu back-end y agrega la anotación **@CrossOrigin** y asegúrate de importarlo del paquete: **org.springframework.web.bind.annotation.*;**

- Guarda los cambios, reinicia tu servicio rest (back-end)

```
1  @RestController
2  @RequestMapping("/api/product")
3  @CrossOrigin
4  public class ProductController {
```

- Ahora regresa a tu proyecto front-end
- Recarga el navegador y verifica si ahora te muestra los productos, en el caso de que te genere error revisa los bindings de las propiedades en tu archivo main.component.html
- Si lograste hacer que te muestre los productos ya tienes conectada tu aplicación FullStack (back-end y front-end)

- 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.**