Desarrollo asistido con GitHub Copilot – Spring Boot + JPA + H2

Objetivo del laboratorio

Que los participantes **usen GitHub Copilot de forma consciente y efectiva** para:

1. Extender un microservicio existente
2. Completar endpoints REST faltantes
3. Introducir lógica de negocio
4. Refactorizar hacia una capa `domain`
5. Crear **unit tests en Java** asistidos por Copilot

Contexto inicial (estado del repositorio)

El estudiante **clona un repositorio existente** que ya contiene:

- Spring Boot
- Spring MVC
- JPA + H2
- Gradle
- Packages existentes:
  - `entities`
  - `repositories`
  - `controllers`
- Un endpoint funcional:
  - `GET /api/products` → lista todos los productos

NO existen aún:

- Endpoint getById
- Endpoint delete
- Categorías
- Lógica de negocio
- Tests

PARTE 1 – Añadir endpoint: Obtener Producto por ID

Objetivo técnico

Agregar:

`GET /api/products/{id}`

con manejo correcto de:

- Producto encontrado → `200 OK`
- Producto no encontrado → `404 Not Found`

Prompt sugerido para Copilot Chat

```
Add a GET endpoint to retrieve a Product by id using Spring
MVC.
Use ProductRepository.
If the product does not exist, return 404.
Follow REST best practices.
```

PARTE 2 – Añadir endpoint: Eliminar Producto

Objetivo técnico

Agregar:

`DELETE /api/products/{id}`

Prompt sugerido

```
Add a DELETE endpoint to remove a Product by id.
If the product does not exist, return 404.
If deleted successfully, return 204 No Content.
```

PARTE 3 – Crear CRUD de Categorías

Objetivo técnico

Agregar soporte completo para **Categorías**:

- Entity
- Repository
- Controller
- CRUD básico

Nuevos elementos esperados

```
entities/Category.java
repositories/CategoryRepository.java
controllers/CategoryController.java
```

```
Create a JPA entity called Category with:
- id (Long, autogenerated)
- name (String, required, max 100)
Map it to a table called categories.
```

```
Create a REST controller for Category.
Base path: /api/categories
Include basic CRUD operations using Spring MVC and JPA.
```

PARTE 4 – Añadir lógica de negocio en un endpoint

Objetivo técnico

Agregar **regla de negocio**, por ejemplo:

> No permitir eliminar un producto si está marcado como `active = true`

Actividad

Modificar el endpoint `DELETE /products/{id}`

Prompt sugerido

```
Add business logic to prevent deleting a product if it is
active.
If active, return HTTP 409 Conflict with a meaningful message.
```

PARTE 5 – Mover lógica de negocio al package `domain`

Objetivo técnico

Introducir **separación de responsabilidades**:

```
domain/
 └── ProductService.java
```

Actividad

1. Crear package `domain`
2. Mover la lógica de negocio del controller

```
Refactor the business logic from the controller into a
ProductService
inside a domain package.
The controller should delegate responsibility to the service.
```

PARTE 6 – Crear Unit Tests con Java

Objetivo técnico

Crear **tests unitarios** para:

- `ProductService`
- `ProductController` (opcional según tiempo)

Test de servicio

Prompt sugerido

```
Create unit tests for ProductService using JUnit 5 and
Mockito.
Mock the ProductRepository.
Cover:
- delete inactive product
- prevent deleting active product
```

Test de controller (opcional)

```
Create a WebMvcTest for ProductController.
Mock ProductService.
Test GET product by id:
- when found return 200
- when not found return 404
```

Validaciones clave

- No usar contexto completo (`@SpringBootTest`) innecesariamente
- Uso correcto de mocks
- Tests legibles (Copilot a veces genera ruido)