



## Módulo 3.-

### Acceso a Datos en Aplicaciones Back-End Java

- Spring Data JPA y JDBC Template.
- Configuración y conexión con bases de datos (PostgreSQL / MySQL).
- Repositorios y consultas personalizadas.
- Mapeo objeto-relacional (ORM) y entidades.
- Introducción a bases de datos NoSQL (MongoDB).



## Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

### **Objetivo general**

Comprender cómo integrar Spring con motores de bases de datos relacionales y NoSQL, aplicar el patrón repositorio y manejar entidades JPA.

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

○ ¿Por qué es clave el acceso a datos?

**¿Qué tienen en común una tienda online, una app de transporte y una fintech?**


Todas dependen de acceder, almacenar y consultar datos de manera eficiente y segura.

**¿Y qué rol cumple el Back-End Java en esto?**

- Orquestrar cómo se **persisten los datos** en bases de datos relacionales y NoSQL
- Aplicar **buenas prácticas** con herramientas modernas como Spring Data JPA
- Gestionar conexiones, consultas personalizadas y entidades de negocio



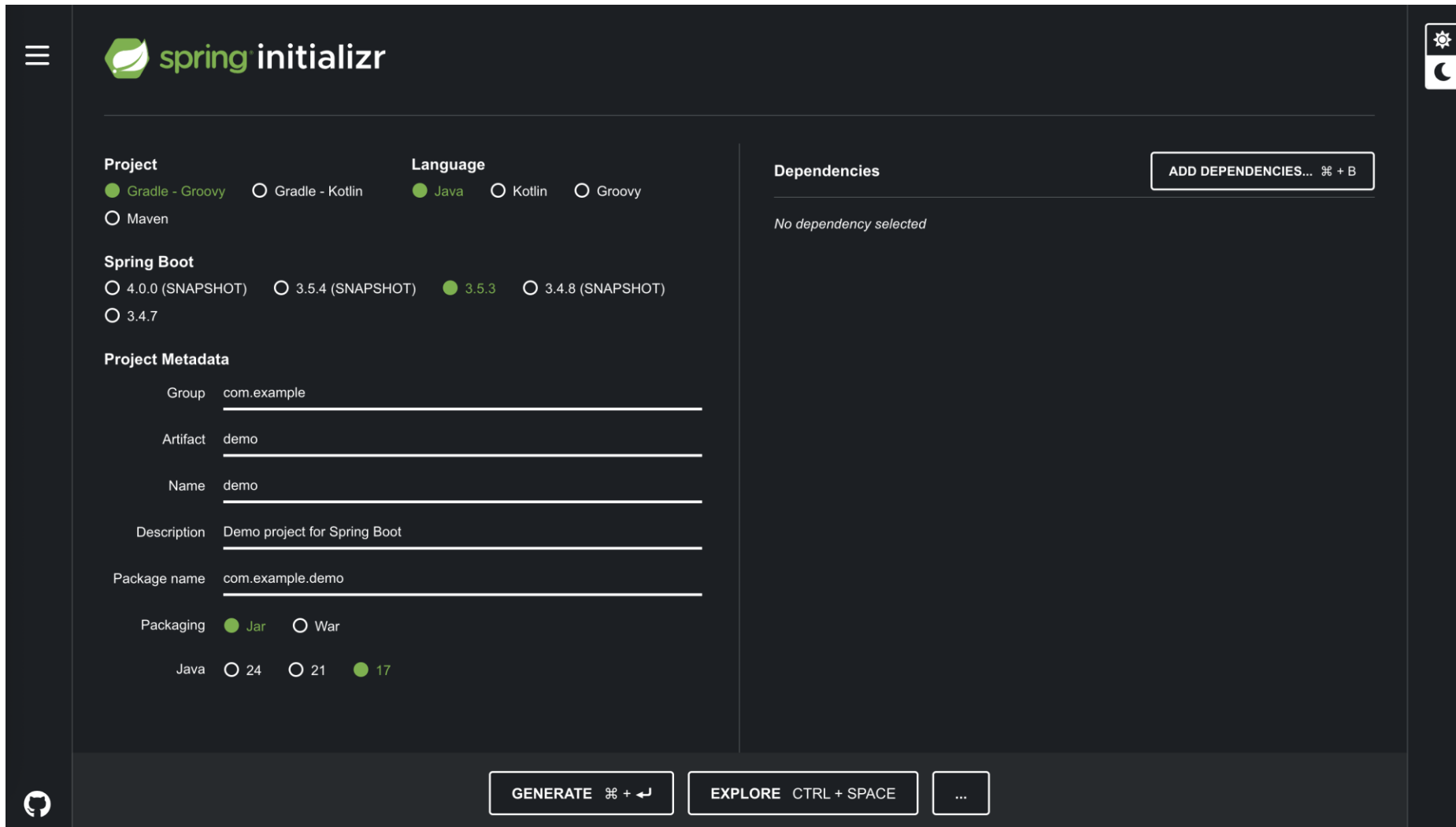
# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

 Este módulo te entrega las herramientas clave para eso:

- Elegir el mejor enfoque (JPA vs JDBC Template)
- Conectar con PostgreSQL, MySQL y MongoDB
- Crear repositorios y exponer tus datos con REST



# ¿Qué es [start.spring.io](https://start.spring.io)?



The screenshot shows the Spring Initializr web application interface. The header features the Spring logo and the text "spring initializr". The main content area is divided into two columns. The left column contains configuration options for the project, including the build tool (Gradle - Groovy, Gradle - Kotlin, Maven), the language (Java, Kotlin, Groovy), the Spring Boot version (4.0.0 (SNAPSHOT), 3.5.4 (SNAPSHOT), 3.5.3, 3.4.8 (SNAPSHOT), 3.4.7), and the project metadata (Group, Artifact, Name, Description, Package name). The right column is for dependencies, with a button to "ADD DEPENDENCIES... ⌘ + B". The bottom of the interface has three buttons: "GENERATE ⌘ + ↵", "EXPLORE CTRL + SPACE", and "...".

**Project**

☒ Gradle - Groovy ☐ Gradle - Kotlin ☐ Maven

**Language**

☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**

☐ 4.0.0 (SNAPSHOT) ☐ 3.5.4 (SNAPSHOT) ☒ 3.5.3 ☐ 3.4.8 (SNAPSHOT) ☐ 3.4.7

**Project Metadata**

Group

Artifact

Name

Description

Package name

**Packaging**

☒ Jar ☐ War

**Java**

☐ 24 ☐ 21 ☒ 17

**Dependencies**

No dependency selected

Es un **generador de proyectos Spring Boot** que te permite crear de forma rápida y sencilla una estructura de proyecto lista para comenzar a programar. Es como un **asistente interactivo** que configura por ti lo que normalmente harías a mano: estructura de carpetas, dependencias, configuración de Maven o Gradle, y clases base.

# ¿Qué es [start.spring.io](https://start.spring.io)?

¿Qué puedes configurar ahí?

Al ingresar a la página, puedes elegir:

## 1. Proyecto

- **Maven o Gradle:** el sistema de construcción.

## 2. Lenguaje

- **Java, Kotlin, o Groovy.**

## 3. Versión de Spring Boot

- Te sugiere la versión más estable (puedes elegir otras si lo necesitas).

## 4. Metadatos del proyecto

- **Group:** como un paquete base (por ejemplo, `cl.ingridgonzalez`).
- **Artifact:** nombre del proyecto (por ejemplo, `api-ventas`).
- **Name, Description, Package name:** opcional, aunque útil si estás creando un proyecto más formal.
- **Packaging:** jar o war (usa jar si no es para un servidor externo tipo Tomcat).
- **Java version:** 17 es la más común ahora, aunque puedes elegir según tu entorno.



# ¿Qué es [start.spring.io](https://start.spring.io)?

## 5. Dependencias

Aquí agregas los "módulos" que quieres usar en tu proyecto, como:

Dependencia	¿Para qué sirve?
Spring Web	Para crear controladores REST o MVC
Spring Data MongoDB	Para conectarte con MongoDB usando repositorios
Lombok	Elimina código repetitivo como getters/setters
Spring Boot DevTools	Recarga automática mientras desarrollas
Spring Security	Para autenticación y autorización
Spring Boot Actuator	Para monitorear el estado del app



# ¿Qué es [start.spring.io](https://start.spring.io)?

¿Qué te entrega?

Cuando haces clic en "**Generate**", se descarga un archivo .zip con:

- Proyecto ya estructurado con carpetas `src/main/java` y `resources`.
- Archivo `pom.xml` o `build.gradle` listo.
- Clase principal con `@SpringBootApplication`.
- Archivos de configuración (`application.properties` o `.yaml`).
- Dependencias ya definidas.
- Listo para abrir en VS Code, IntelliJ, Eclipse, etc.





# ¿Qué es [start.spring.io](https://start.spring.io)?

¿Cómo lo usas?

- Entrás a <https://start.spring.io>
- Configuras el proyecto a tu medida
- Descargas el .zip
- Lo abres en tu editor Java favorito (VS Code, IntelliJ, etc.)
- Comienzas a programar inmediatamente



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

### ¿Qué son Spring Data JPA y JDBC Template?

#### Spring Data JPA

Es una abstracción de alto nivel sobre **JPA (Java Persistence API)** desarrollada por Spring, que permite acceder y manipular datos de bases de datos relacionales usando **interfaces y anotaciones**, sin necesidad de escribir SQL manualmente.

🔧 Características clave:

- CRUD automático con solo definir una interfaz.
- Consultas personalizadas con `@Query`.
- Soporte para paginación, ordenamiento y relaciones entre entidades.



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

### ¿Qué son Spring Data JPA y JDBC Template?

#### JDBC Template

Es una herramienta de **bajo nivel** provista por Spring Framework que simplifica la ejecución de SQL puro usando JDBC. Proporciona una forma más limpia de trabajar con `Connection`, `PreparedStatement`, `ResultSet`, etc.

Características clave:

- Máximo control sobre SQL.
- Requiere escribir queries manualmente.
- Ideal para operaciones complejas o tuning fino.



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

Diferencias entre JPA (vía Spring Data) y JDBC Template

Característica	Spring Data JPA	JDBC Template
Nivel de abstracción	Alto (basado en entidades y ORM)	Bajo (manipula SQL y ResultSet directamente)
Requiere mapeo de entidades	Sí	No necesariamente
Consultas	Derivadas del nombre del método o anotaciones	SQL escrito a mano
Curva de aprendizaje	Más amigable	Más control, pero más código
Casos de uso recomendados	CRUD, relaciones complejas, modelo de dominio	Operaciones de alto rendimiento o SQL puro

✓ Para la mayoría de los proyectos Java modernos, **Spring Data JPA es la primera opción.**



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ☐ Spring Data JPA y JDBC Template

**¿Cuándo usar uno u otro?**

Situación	¿Cuál conviene?
Necesitas control total del SQL (joins complicados, tuning)	JDBC Template
Quieres construir rápido una app CRUD típica	Spring Data JPA
Debes trabajar con vistas, stored procedures o DB legacy	JDBC Template
Estás modelando tu dominio como entidades	Spring Data JPA



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

### Buenas prácticas

- En proyectos grandes: **puedes combinar ambos**.
- Si usas JPA, **monitoriza el SQL generado** (usa `spring.jpa.show-sql=true`).
- En consultas complejas: evalúa si usar `@Query` con JPQL o cambiar a JDBC Template.



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

✚ Anotaciones de JPA más comunes

### @Entity

Marca una clase Java como una entidad que se mapeará a una tabla.

```
@Entity public class Producto {  
    ...  
}
```

### @Id

Define el campo que actúa como **clave primaria**.

```
@Id  
private Long id;
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

@GeneratedValue

Especifica cómo se generará el valor del ID (auto-incremental, UUID, etc.)

```
@Id
```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
private Long id;
```

@Table y @Column

Se usan para especificar el nombre real de la tabla o columna en la BD (opcional si coinciden).

```
@Table(name = "productos")
```

```
@Column(name = "nombre_producto")
```





# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

### Repositorios en Spring Data

Spring Data proporciona interfaces listas para usar:

`CrudRepository<T, ID>`

- Operaciones básicas: `save`, `findById`, `findAll`, `delete`, etc.

`JpaRepository<T, ID>`

- Extiende `CrudRepository` y agrega funcionalidades más avanzadas: paginación, ordenamiento, etc.

```
public interface ProductoRepository extends JpaRepository<Producto, Long> {  
    List<Producto> findByNombre(String nombre);  
}
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

🧠 Consultas derivadas vs. consultas personalizadas

- ◆ Derivadas del nombre del método

Spring genera la consulta automáticamente:

```
List<Producto> findByCategoria(String categoria);
```

Internamente genera algo como:

```
SELECT * FROM producto WHERE categoria = ?
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

- ◆ Personalizadas con `@Query`

Para más control o SQL más complejo:

```
@Query("SELECT p FROM Producto p WHERE p.precio > :min")  
List<Producto> productosCaros(@Param("min") BigDecimal precioMinimo);
```

También puedes usar SQL nativo:

```
@Query(value = "SELECT * FROM productos WHERE nombre LIKE %:nombre%",  
nativeQuery = true)  
List<Producto> buscarPorNombre(@Param("nombre") String nombre);
```





# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java



☐ Spring Data JPA y JDBC Template

**Ejercicios**









**Project**

☐ Gradle - Groovy

☐ Gradle - Kotlin

☒ **Maven**

**Language**

☒ **Java**

☐ Kotlin

☐ Groovy

**Spring Boot**

☐ 4.0.0 (SNAPSHOT)

☐ 3.5.4 (SNAPSHOT)

☒ **3.5.3**

☐ 3.4.8 (SNAPSHOT)

☐ 3.4.7

**Project Metadata**

Group

com.isolis

Artifact

api-productos

Name

api-productos

Description

Demo CRUD con Spring Data JPA

Package name

com.isolis.api-productos

Packaging

☒ **Jar**

☐ War

Java

☐ 24

☐ 21

☒ **17**

**Dependencies**

ADD DEPENDENCIES... ⌘ + B

**Spring Web**

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Data JPA**

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

**MySQL Driver**

SQL

MySQL JDBC driver.

**Spring Boot DevTools**

DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

**Lombok**


DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

GENERATE ⌘ + ↵

EXPLORE CTRL + SPACE

...



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

Ir a Github y descargar : [spring\\_crud\\_examples/jpa](https://github.com/Joselota/Relatorias/tree/main/spring_crud_examples/jpa)

[https://github.com/Joselota/Relatorias/tree/main/spring\\_crud\\_examples/jpa](https://github.com/Joselota/Relatorias/tree/main/spring_crud_examples/jpa)

```
src/
├─ main/
│   ├── java/
│   │   └─ cl/ingridgonzalez/apiproductos/
│   │       ├── ApiProductosApplication.java
│   │       ├── controller/
│   │       ├── model/
│   │       └─ repository/
│   └─ resources/
│       └─ application.properties
pom.xml
```

Relatorias / [spring\\_crud\\_examples](#) / [jpa](#) /

 **Joselota** Agrega nuevos programas

Name	Last commit
..	
Producto.java	Agrega
ProductoJpaController.java	Agrega
ProductoRepository.java	Agrega

No olvidar configurar: Configurar `application.properties`

```
spring.application.name=api-productos
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/productos_db?serverTimezone=UTC
```

```
spring.datasource.username=root
```

```
spring.datasource.password=tu_clave # La misma que usaste en el terminal
```



```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```




```
# Hibernate
```

```
spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.format_sql=true
```





**Project**

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ **Java** ☐ Kotlin ☐ Groovy

☒ **Maven**

**Spring Boot**

☐ 4.0.0 (SNAPSHOT) ☐ 3.5.4 (SNAPSHOT) ☒ **3.5.3** ☐ 3.4.8 (SNAPSHOT)

☐ 3.4.7

**Project Metadata**

Group

com.isolis

Artifact

api-productos-jdbc

Name

api-productos-jdbc

Description

CRUD básico con JDBC Template

Package name

com.isolis.api-productos-jdbc

Packaging

☒ **Jar** ☐ War

Java

☐ 24 ☐ 21 ☒ **17**

**Dependencies**

ADD DEPENDENCIES... ⌘ + B

**Spring Web** **WEB**

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Boot DevTools** **DEVELOPER TOOLS**

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

**Spring Data JDBC** **SQL**

Persist data in SQL stores with plain JDBC using Spring Data.

**MySQL Driver** **SQL**

MySQL JDBC driver.

GENERATE ⌘ + ↵

EXPLORE CTRL + SPACE

...

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



```
src/  
└─ main/  
    └─ java/  
        └─ com/isolis/api_productos_jdbc/  
            ├── ApiProductosJdbcApplication.java  
            ├── controller/  
            │   └── ProductoJdbcController.java  
            ├── dao/  
            │   └── ProductoJdbcDAO.java  
            └── model/  
                └── Producto.java  
└─ resources/  
    └─ application.properties  
pom.xml
```

Ir a Github y descargar : [spring\\_crud\\_examples/jdbc](https://github.com/Joselota/Relatorias/tree/main/spring_crud_examples/jdbc)

[https://github.com/Joselota/Relatorias/tree/main/spring\\_crud\\_examples/jdbc](https://github.com/Joselota/Relatorias/tree/main/spring_crud_examples/jdbc)



No olvidar configurar: Configurar `application.properties`

`spring.application.name=api-productos-jdbc`

`spring.datasource.url=jdbc:mysql://localhost:3306/productos_db?serverTimezone=UTC`

`spring.datasource.username=root`

`spring.datasource.password=admin`

`spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver`

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA y JDBC Template

¿Cuál elegir?

Escenario	¿Qué usar?	¿Por qué?
App CRUD con entidades relacionadas	● JPA	Mapea automáticamente, rápido de construir
Consultas complejas con joins o subqueries	● JDBC Template	Mayor control del SQL
Necesitas paginación y ordenamiento	● JPA (Pageable)	Ya está integrado
Quieres máxima performance y tuning	● JDBC Template	Tú defines el SQL
Prototipado rápido	● JPA	Menos código, más productividad
Reportes con filtros dinámicos	● JDBC Template	Eficiencia + personalización

- **Spring Data JPA** es como usar Excel con macros automatizadas. Tú defines qué necesitas, y el sistema se encarga del “cómo”.
- **JdbcTemplate** es como usar SQL puro en una base de datos: tú controlas todo, pero debes hacer todo tú.



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Configuración y conexión con bases de datos

### ¿Cómo se conecta Spring Boot con MySQL?

Una aplicación necesita acceder a la base de datos para:

- Guardar y consultar información (ej. productos, usuarios)
- Ejecutar operaciones CRUD desde los controladores
- Persistir los cambios de forma segura

### ¿Qué archivo se usa?

Spring Boot utiliza el archivo `application.properties` o `application.yml` para definir:

- Dirección del servidor de base de datos
- Usuario y contraseña
- Driver JDBC compatible con el motor (MySQL, PostgreSQL, etc.)



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Configuración y conexión con bases de datos

### 1. Configuración del datasource en application.properties

El archivo application.properties (o application.yml) es el **centro de configuración** de un proyecto Spring Boot.

Para conectar una base de datos PostgreSQL local, se agregan los siguientes parámetros:

```
# Configuración de conexión
spring.datasource.url=jdbc:postgresql://localhost:5432/mibase
spring.datasource.username=postgres
spring.datasource.password=admin

# Driver
spring.datasource.driver-class-name=org.postgresql.Driver
```

Para MySQL sería:

```
spring.datasource.url=jdbc:mysql://localhost:3306/mibase?serverTimezone=UTC
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Configuración y conexión con bases de datos

### 2. Dependencias JDBC en pom.xml (o en start.spring.io)

Para que Spring Boot pueda conectarse a la BD, debes incluir el *driver JDBC* correspondiente.

#### PostgreSQL:

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
</dependency>
```

#### MySQL:

```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
</dependency>
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Configuración y conexión con bases de datos

### 3. ¿Cómo crea y actualiza tablas Spring Boot automáticamente?

Spring Boot usa Hibernate por debajo para gestionar el ORM (mapear clases a tablas). Podemos controlar cómo maneja las tablas con esta propiedad:

```
# application.properties  
spring.jpa.hibernate.ddl-auto=update
```

Modos comunes:

Modo	¿Qué hace?	¿Cuándo usarlo?
none	No hace nada	Producción con esquema fijo
validate	Verifica pero no crea	Producción segura
update	Crea/actualiza columnas	Desarrollo ágil
create	Borra y crea desde cero	Pruebas temporales
create-drop	Igual a create, pero borra al cerrar	Pruebas unitarias

✓ **En producción**, se recomienda **validate**.

⚠ **create** y **create-drop** son útiles solo para pruebas.



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Configuración y conexión con bases de datos

### 4. Visualizar las consultas generadas

Para fines didácticos y de depuración, puedes mostrar el SQL que Hibernate está ejecutando:

```
#application.properties
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Esto imprime algo como:

```
select producto0_.id as id1_0_,
       producto0_.nombre as nombre2_0_
from producto producto0_;
```





# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Configuración y conexión con bases de datos

### 5. Verificar la conexión a la base de datos

Cuando ejecutas tu app, en la consola verás:

HikariPool-1 - Starting...

HikariPool-1 - Start completed.

- Si hay errores de conexión, verás `Connection refused`, `invalid password`, o `relation does not exist`.

Puedes hacer pruebas básicas con `psql`, DBeaver o pgAdmin para confirmar que la BD está funcionando correctamente.



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Repositorios y consultas personalizadas.

Un Repository es una interfaz que nos permite acceder a datos sin necesidad de escribir código SQL. Spring Data JPA genera automáticamente las implementaciones.

Ejemplo:

```
public interface ProductoRepository extends CrudRepository<Producto, Long> {}
```

Interfaz	Descripción
CrudRepository	Operaciones básicas CRUD
JpaRepository	Extiende CrudRepository + paginación/sorting

```
public interface ProductoRepository extends JpaRepository<Producto, Long> {  
    List<Producto> findByNombre(String nombre);  
    List<Producto> findByPrecioGreaterThan(Double precio);  
}
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Repositorios y consultas personalizadas.

### Consultas personalizadas

Existen **tres formas principales** de personalizar consultas en Spring Data JPA:

Tipo	Ejemplo	Comentario breve
Query derivada	<code>findByNombre(String nombre)</code>	Spring infiere la consulta desde el nombre
@Query (JPQL)	<code>@Query("SELECT p FROM Producto p WHERE p.precio &gt; :precio")</code>	Más flexible y legible
Consulta nativa	<code>@Query(value = "SELECT * FROM productos WHERE precio &gt; ?", nativeQuery = true)</code>	Cuando necesitas SQL puro



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Repositorios y consultas personalizadas.

Ejemplo con @Query

```
public interface ProductoRepository extends JpaRepository<Producto, Long> {  
  
    // Consulta derivada  
    List<Producto> findByNombreContaining(String fragmento);  
  
    // JPQL  
    @Query("SELECT p FROM Producto p WHERE p.precio > :precio")  
    List<Producto> productosCaros(@Param("precio") double precio);  
  
    // SQL nativa  
    @Query(value = "SELECT * FROM producto WHERE precio < ?1", nativeQuery = true)  
    List<Producto> productosBaratos(double precioMax);  
}
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Mapeo objeto-relacional (ORM) y entidades.

### **Definición:**

**ORM (Object-Relational Mapping)** es una técnica que permite mapear objetos del lenguaje de programación (como clases en Java) a tablas de una base de datos relacional.



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Mapeo objeto-relacional (ORM) y entidades.

### ¿Qué es el ORM?

- **Object-Relational Mapping:** técnica para convertir datos entre sistemas de tipos incompatibles (Java <--> SQL).
- Permite trabajar con **objetos** en vez de escribir queries SQL manuales.
- JPA (Java Persistence API) es la especificación estándar; Hibernate es la implementación más común (y la usada por Spring Boot).



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Mapeo objeto-relacional (ORM) y entidades.

### Anotaciones clave de JPA

```
@Entity                // Marca la clase como entidad
@Table(name = "clientes") // (opcional) Nombre de la tabla

public class Cliente {

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

    @Column(nullable = false, length = 100)
    private String nombre;

    private String correo;
}
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

○ Mapeo objeto-relacional (ORM) y entidades.

Anotación	Descripción
@Entity	Define que la clase es persistente.
@Table	Opcional. Define nombre de la tabla.
@Id	Indica el campo de clave primaria.
@GeneratedValue	Cómo se genera el ID.
@Column	Configura opciones específicas de columna.





# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Mapeo objeto-relacional (ORM) y entidades.

### ✿ Relaciones entre entidades

Breve mención si se desea ampliar:

```
@OneToMany(mappedBy = "cliente") private List pedidos;
```

### 💻 Ejercicios prácticos:

- Crear una clase Producto mapeada con:
  - `@Entity`, `@Id`, `@Column` y `@GeneratedValue`.
  - Campos: `id`, `nombre`, `precio`, `stock`.
- Generar la tabla automáticamente con `spring.jpa.hibernate.ddl-auto=update`.
- Observar en consola cómo Hibernate genera el SQL para crear la tabla.



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Mapeo objeto-relacional (ORM) y entidades.

### Correspondencia Clase ↔ Tabla en la Base de Datos

¿Cómo se traduce una entidad Java a una tabla SQL?

En Java (@Entity)	En MySQL (tabla generada)
@Entity	Tabla con el mismo nombre (o definido con @Table)
private Long id; con @Id, @GeneratedValue	Columna id BIGINT AUTO_INCREMENT
private String nombre;	Columna nombre VARCHAR(255)
private Double precio;	Columna precio DOUBLE



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

○ Mapeo objeto-relacional (ORM) y entidades.

**Ejemplo:**

```
@Entity
public class Producto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nombre;
    private Double precio;
}
```

```
CREATE TABLE producto (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255),
    precio DOUBLE
);
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Mapeo objeto-relacional (ORM) y entidades.

### Relaciones entre Entidades (Mapeo ORM)

¿Cómo se modelan relaciones en JPA?

Tipo de relación	Anotación JPA	Ejemplo Java	Ejemplo SQL
Uno a muchos	@OneToMany	Una categoría tiene muchos productos	Clave foránea en tabla producto
Muchos a uno	@ManyToOne	Muchos productos tienen una categoría	producto.categoria_id → categoria.id
Uno a uno	@OneToOne	Un usuario tiene un perfil	Una FK única
Muchos a muchos	@ManyToMany	Cursos y estudiantes	Tabla intermedia (curso_estudiante)



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

○ Mapeo objeto-relacional (ORM) y entidades.

**Ejemplo práctico: Producto → Categoría**

```
@Entity
public class Producto {
    @ManyToOne
    @JoinColumn(name = "categoria_id")
    private Categoria categoria;
}
```

```
ALTER TABLE producto ADD categoria_id BIGINT;
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### ¿Qué es NoSQL?

Cuando se utiliza este término es para referirse a cualquier base de datos que no es relacional, porque la manera en la cual los datos se almacenan es esencialmente distinta a como almacenarían en una típica base de datos relacional. Es decir, que no se almacenan en tablas relacionales. Las bases de datos NoSQL aparecieron a finales de los 2.000, cuando los costos de almacenamiento bajaron radicalmente.

Las bases de datos NoSQL permiten almacenar grandes cantidades de información sin estructura lo cual otorga gran potencial y gran flexibilidad.

- BD de documentos -> json
- BD clave – valor
- BD en grafo -> nodos y arcos ( nodos : entidades)
- BD orientadas a objetos

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Introducción a bases de datos NoSQL(MongoDB)

### ¿Qué es MongoDB?

Es la base de datos más popular en este momento.

Su versión community es completamente gratuita y de código abierto.

Además, es una base de datos de documentos diseñada para aumentar la productividad de los desarrolladores y para soportar una **escalabilidad**.

MongoDB es una base de datos documental, donde cada registro es un documento que no es más que una estructura de datos compuesta por pares campo valor.

Los documentos son similares a objetos json e incluso los valores de los campos puedes incluir otros documentos arrays.

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### ¿Qué es MongoDB?

#### Ventajas

- Mapeo directo
- Objetos anidados

MongoDB almacena los documentos en colecciones que son equivalentes a las tablas en las bases de datos relacionales.



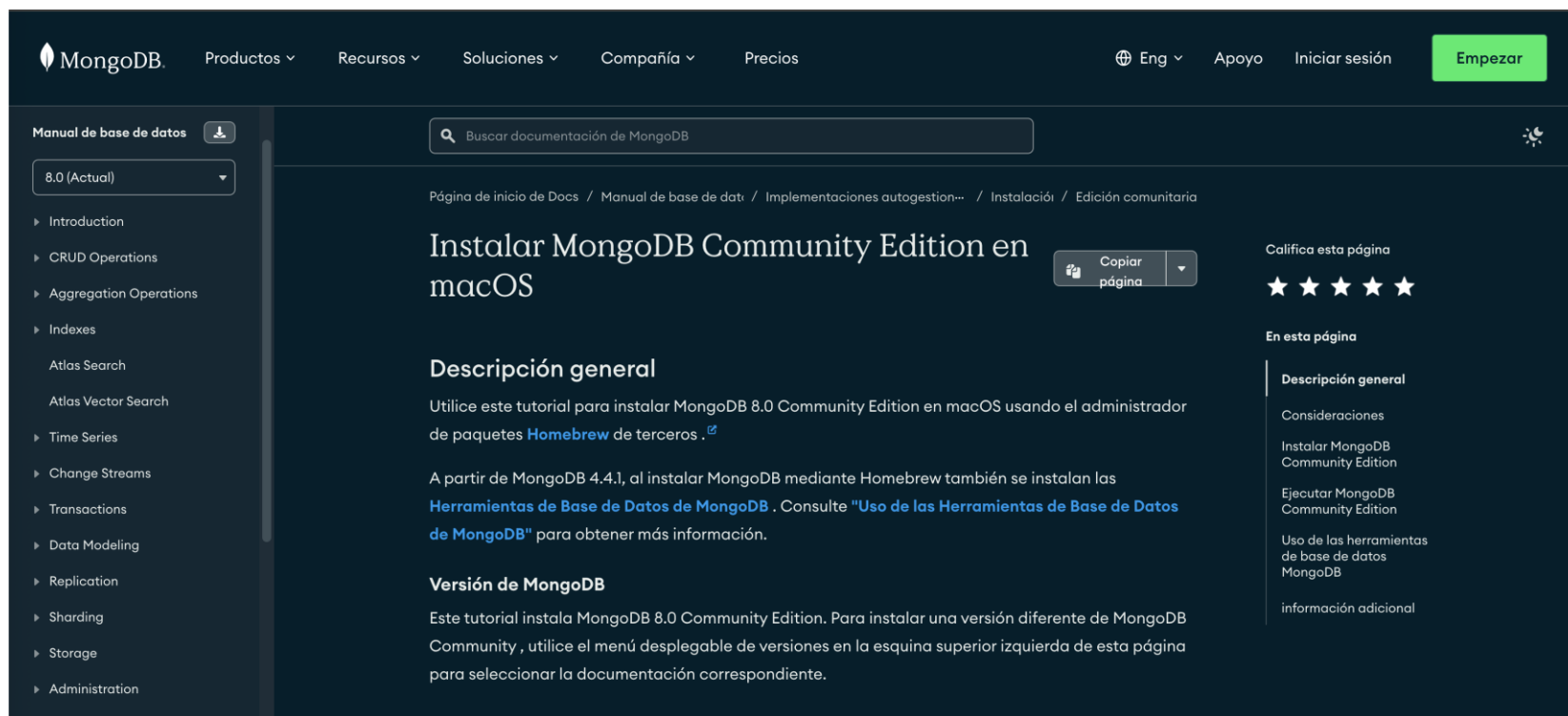
# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### ¿Cómo instalar MongoDB?

Sitio oficial: <https://www.mongodb.com/es>

<https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/>



The screenshot displays the MongoDB documentation website. The top navigation bar includes links for 'Productos', 'Recursos', 'Soluciones', 'Compañía', and 'Precios', along with a language selector set to 'Eng', 'Apoyo', 'Iniciar sesión', and a green 'Empezar' button. A left sidebar shows the 'Manual de base de datos' with a dropdown for '8.0 (Actual)' and a list of topics including Introduction, CRUD Operations, Aggregation Operations, Indexes, Atlas Search, Atlas Vector Search, Time Series, Change Streams, Transactions, Data Modeling, Replication, Sharding, Storage, and Administration. The main content area is titled 'Instalar MongoDB Community Edition en macOS' and includes a 'Copiar página' button. Below the title is a 'Descripción general' section with text about using Homebrew for installation. A 'Versión de MongoDB' section follows, explaining how to select the correct documentation version. On the right, there is a 'Califica esta página' section with five stars and a list of related topics under 'En esta página'.

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### ¿Cómo instalar MongoDB?

#### Mac

- Verifica que tienes Homebrew instalado: `> brew -v`
- Agrega el repositorio de MongoDB a Homebrew: `> brew tap mongodb/brew`
- Instala MongoDB Community Edition: `> brew install mongodb-community@7.0`
- Iniciar MongoDB como servicio: `> brew services start mongodb-community@7.0`
- Verifica que MongoDB esté corriendo: `> mongo`

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### ¿Cómo instalar MongoDB?

#### Windows

- Descargar MongoDB: <https://www.mongodb.com/try/download/community>
- Ejecutar el instalador

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Introducción a bases de datos NoSQL(MongoDB)

### Interactuando con MongoDB

Acción	Comando
Mostrar bases de datos	show dbs
Crear o cambiar a una base de datos	use miBaseDeDatos
Mostrar colecciones dentro de la base	show collections
Insertar un documento en una colección	db.clientes.insertOne({ nombre: "Ingrid", pais: "Chile" })
Consultar documentos en una colección	db.clientes.find()

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Introducción a bases de datos NoSQL(MongoDB)

### Ejercicios en MongoDB

Acción	Descripción	Comando
1. Ver bases de datos	Lista todas las bases existentes	show dbs
2. Crear o cambiar base	Cambia a una base (la crea si no existe)	use tienda
3. Ver colecciones	Muestra las colecciones de la base actual	show collections

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### Ejercicios en MongoDB

Insertar un documento:

```
>> db.productos.insertOne({ nombre: "Laptop", precio: 1200, stock: 15 })
```

Insertar varios documentos:

```
>> db.productos.insertMany([ { nombre: "Mouse", precio: 25, stock: 100 }, { nombre: "Teclado", precio: 45, stock: 75 } ])
```

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### Ejercicios en MongoDB

Ver todos los documentos:

```
>> db.productos.find()
```

Filtro por nombre:

```
>> db.productos.find({ nombre: "Mouse" })
```

Solo mostrar nombre y precio:

```
>> db.productos.find({}, { nombre: 1, precio: 1, _id: 0 })
```

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### Ejercicios en MongoDB

Actualizar un campo:

```
>> db.productos.updateOne({ nombre: "Mouse" }, { $set: { precio: 30 } })
```

Aumentar stock:

```
> db.productos.updateOne({ nombre: "Teclado" }, { $inc: { stock: 10 } })
```



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### Ejercicios en MongoDB: Eliminar documentos

Eliminar uno:

```
> db.productos.deleteOne({ nombre: "Teclado" })
```

Eliminar todos:

```
> db.productos.deleteMany({})
```

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Introducción a bases de datos NoSQL(MongoDB)

### Ejercicios en MongoDB: Eliminar documentos

Ver la base actual

```
> db
```

Contar cuántos documentos hay:

```
> db.productos.countDocuments()
```

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### Parte 1: Consultas personalizadas en Spring Data JPA

#### ◆ 1. Consultas derivadas vs personalizadas

Spring Data permite dos formas de crear consultas:

##### **A. Consultas derivadas del nombre del método**

Spring las interpreta automáticamente:

```
List<Producto> findByCategoria(String categoria);  
List<Producto> findByPrecioGreaterThan(Double precio);  
List<Producto> findByNombreContaining(String fragmento);
```

Ventajas:

- Simples
- Buen rendimiento
- Cero SQL



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## ○ Introducción a bases de datos NoSQL(MongoDB)

Diferencias clave con modelos relacionales:

Característica	Relacional (JPA)	Documental (MongoDB)
Estructura	Tablas y relaciones	Documentos anidados (JSON)
Esquema	Fijo (con validación)	Flexible (dinámico)
Relacionamiento	JOINS	Subdocumentos, referencias
Ideal para	Datos estructurados	Datos semiestructurados



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Spring Data JPA vs Spring Data MongoDB

Aspecto	Spring Data JPA	Spring Data MongoDB
Tipo de base de datos	Relacional (PostgreSQL, MySQL, etc.)	NoSQL orientada a documentos (MongoDB)
Anotación principal de entidad	@Entity	@Document(collection = "nombre")
Identificador primario	@Id (de javax.persistence)	@Id (de org.springframework.data.annotation)
Mapeo tabla/campo	@Table, @Column	No es necesario definir campos
Relaciones	@OneToMany, @ManyToOne, etc.	Se modelan como referencias o embebidos manualmente
Repositorio base	JpaRepository<T, ID>	MongoRepository<T, ID>
Derivación de métodos	findByNombre, findByEdadGreaterThan	Igual
Consultas personalizadas	@Query("SELECT p FROM Producto p...")	@Query("{ nombre : ?0 }")
Transacciones	Soportadas vía @Transactional	Limitadas en Mongo (solo en clusters replicados)
Requiere esquema?	✅ Sí (tablas, relaciones)	❌ No (esquema flexible)
Persistencia	SQL / ORM con Hibernate	BSON (documentos)

# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### Spring Data MongoDB: configuración básica

En tu pom.xml:

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-  
mongodb</artifactId>  
</dependency>
```

En application.properties:

```
spring.data.mongodb.uri=mongodb://localhost:27017/demo  
spring.data.mongodb.database=demo
```

Puedes usar [MongoDB Compass](#) o Docker para ejecutarlo localmente.



# Módulo 3.- Acceso a Datos en Aplicaciones Back-End Java

## Introducción a bases de datos NoSQL(MongoDB)

### Anotaciones clave en MongoDB

Anotación	Descripción
@Document	Define una clase como documento de MongoDB
@Id	Campo que actúa como clave primaria
@Field	Personaliza el nombre del campo en el documento

Ejemplo:

```
@Document(collection = "clientes")
public class Cliente {
    @Id
    private String id;

    @Field("nombre_completo")
    private String nombre;

    private String correo;
}
```



# Ejercicio: Sprint boot + MongoDB

Ejercicio:

Crear una app Spring Boot que se conecte a MongoDB y realice operaciones CRUD sobre una colección (por ejemplo, productos).







### Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Java ☐ Kotlin ☐ Groovy  
☒ Maven

### Spring Boot

☐ 4.0.0 (SNAPSHOT) ☐ 3.5.4 (SNAPSHOT) ☒ 3.5.3 ☐ 3.4.8 (SNAPSHOT)  
☐ 3.4.7

### Project Metadata

Group   
Artifact   
Name   
Description   
Package name   
Packaging ☒ Jar ☐ War  
Java ☐ 24 ☐ 21 ☒ 17

### Language

### Dependencies

ADD DEPENDENCIES... ⌘ + B

#### Spring Web Services WEB

Facilitates contract-first SOAP development. Allows for the creation of flexible web services using one of the many ways to manipulate XML payloads.

#### Spring Data MongoDB NOSQL

Store data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.

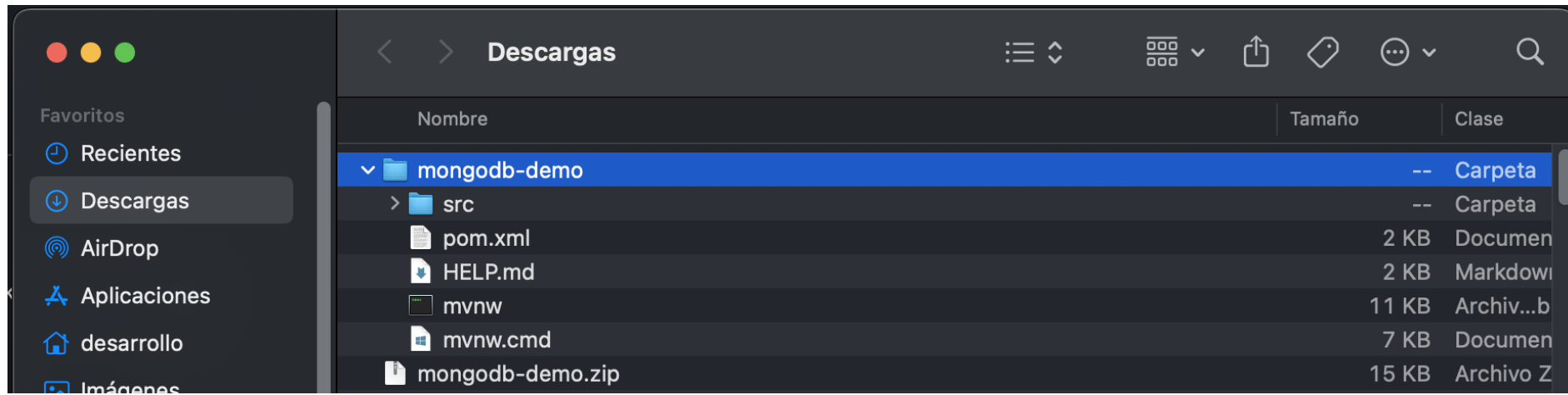
#### Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

#### Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.





## Clase: Controller.java

```
package com.isolis.mongodb_demo.controller;
import com.isolis.mongodb_demo.model.Producto;
import com.isolis.mongodb_demo.repository.ProductoRepository;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/productos")
public class ProductoController {

    private final ProductoRepository productoRepo;

    public ProductoController(ProductoRepository productoRepo) {
        this.productoRepo = productoRepo;
    }

    @GetMapping
    public List<Producto> getAll() {
        return productoRepo.findAll();
    }

    @PostMapping
    public Producto create(@RequestBody Producto producto) {
        return productoRepo.save(producto);
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable String id) {
        productoRepo.deleteById(id);
    }

    // 🆕 Buscar por nombre
    @GetMapping("/buscar")
    public List<Producto> buscarPorNombre(@RequestParam String nombre) {
        return productoRepo.findByNombre(nombre);
    }
}
```



```
package com.isolis.mongodb_demo.model;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
@Document(collection = "productos")
public class Producto {
    @Id
    private String id;
    private String nombre;
    private double precio;
    private int stock;

    public Producto() {
    }
    public Producto(String id, String nombre, double precio, int stock) {
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.stock = stock;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public double getPrecio() {
        return precio;
    }
    public void setPrecio(double precio) {
        this.precio = precio;
    }
    public int getStock() {
        return stock;
    }
    public void setStock(int stock) {
        this.stock = stock;
    }
}
```

Clase: Producto.java



## Clase: ProductoRepository.java

```
package com.isolis.mongodb_demo.repository;

import com.isolis.mongodb_demo.model.Producto;

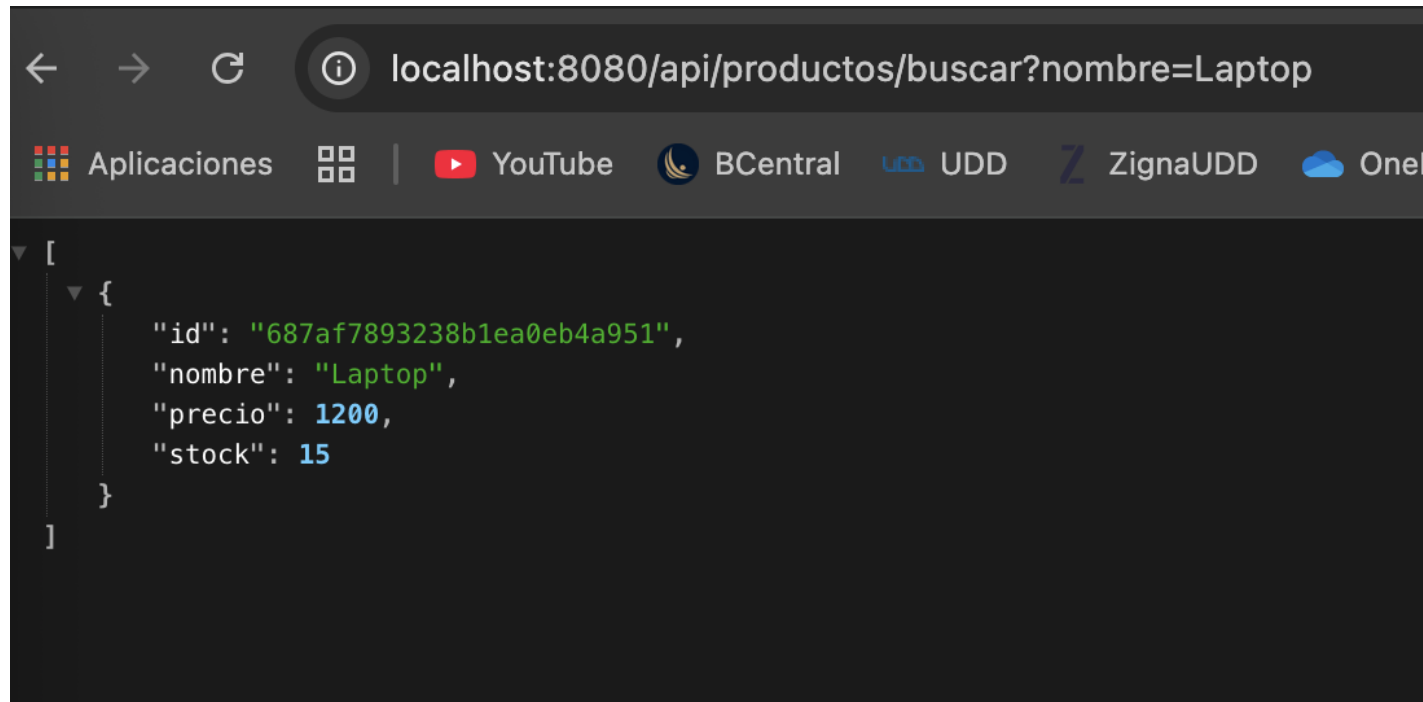
import java.util.List;

import org.springframework.data.mongodb.repository.MongoRepository;

public interface ProductoRepository extends MongoRepository<Producto, String> {
    // Puedes agregar métodos como: List<Producto> findByNombre(String nombre);
    List<Producto> findByNombre(String nombre);
    List<Producto> findByPrecioLessThan(double precio);
    List<Producto> findByStockGreaterThan(int stock);
}
```



Resultado:

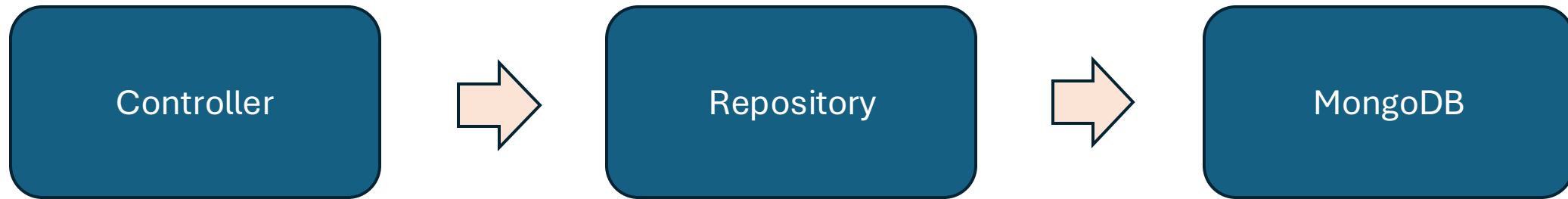


A screenshot of a web browser window. The address bar shows the URL `localhost:8080/api/productos/buscar?nombre=Laptop`. Below the address bar, there is a tab bar with several tabs: 'Aplicaciones', 'YouTube', 'BCentral', 'UDD', 'ZignaUDD', and 'One'. The main content area of the browser displays a JSON response in a dark-themed editor. The JSON is a single-element array containing an object with the following properties: `"id": "687af7893238b1ea0eb4a951"`, `"nombre": "Laptop"`, `"precio": 1200`, and `"stock": 15`.

```
[
  {
    "id": "687af7893238b1ea0eb4a951",
    "nombre": "Laptop",
    "precio": 1200,
    "stock": 15
  }
]
```



## Diagrama de capas



## CRUD en MongoDB con Spring Data

Operación	Método HTTP	Ruta API	Descripción	Código en MongoRepository
Create	POST	/clientes	Crear un nuevo cliente	repository.save(cliente)
Read (todos)	GET	/clientes	Obtener todos los clientes	repository.findAll()
Read (por ID)	GET	/clientes/{id}	Obtener cliente por ID	repository.findById(id)
Update	PUT	/clientes/{id}	Actualizar cliente existente	repository.save(cliente) (con ID existente)
Delete	DELETE	/clientes/{id}	Eliminar cliente por ID	repository.deleteById(id)

