

Tutorial

Spring

FCHAVEZ

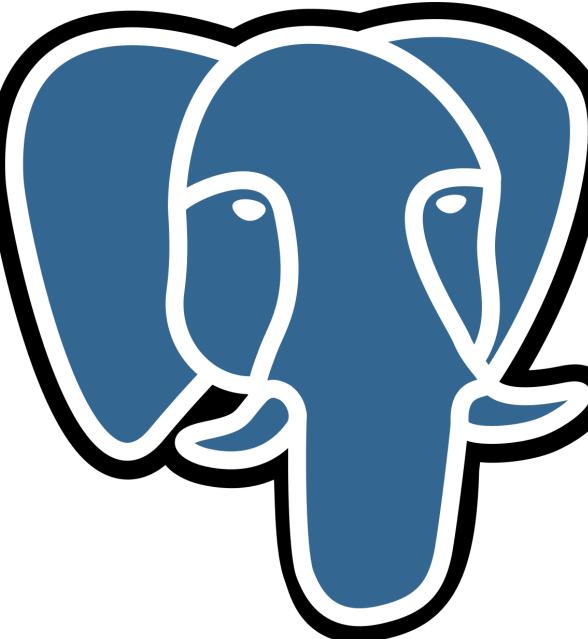
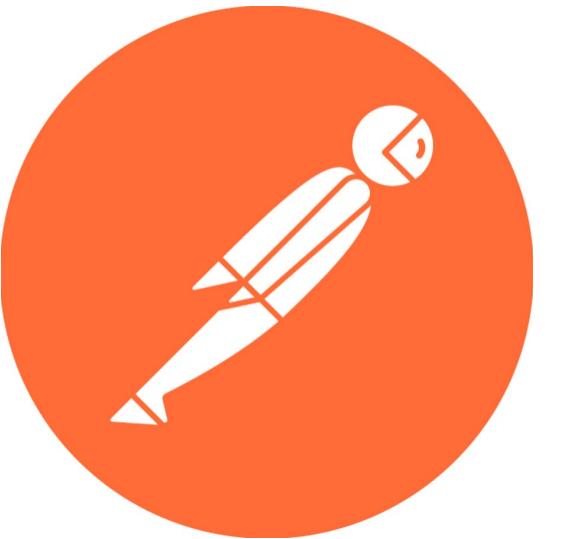
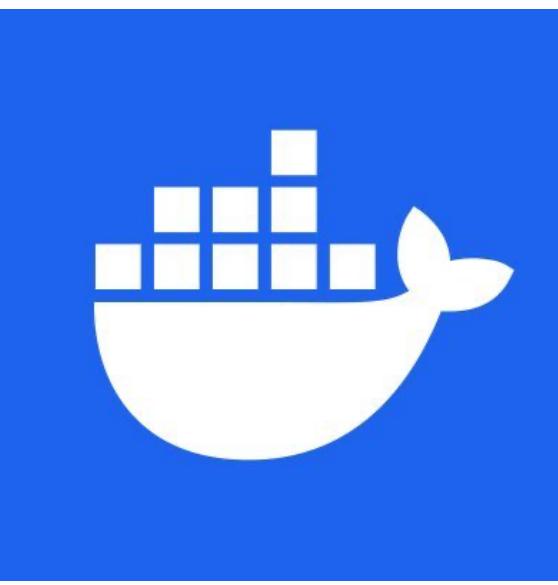
Servicio de usuarios

- Considerando a los usuarios como un recurso hacer una API para administrarlos



Ingredientes

- Spring framework
- IntelliJ IDEA
- Gradle
- Postman
- Docker
- Postgresql



Spring

Módulos a utilizar:



Spring Framework



Base de todos los proyectos



Spring Boot



Spring Data JPA



Spring Security

Spring

Módulos a utilizar:



Spring Framework



Spring Boot



Spring Data JPA



Spring Security



Aplicaciones autocontenidoas

Spring

Módulos a utilizar:



Spring Framework



Spring Boot



Spring Data JPA



Spring Security



Capa de persistencia

Spring

Módulos a utilizar:



Spring Framework



Spring Boot



Spring Data JPA

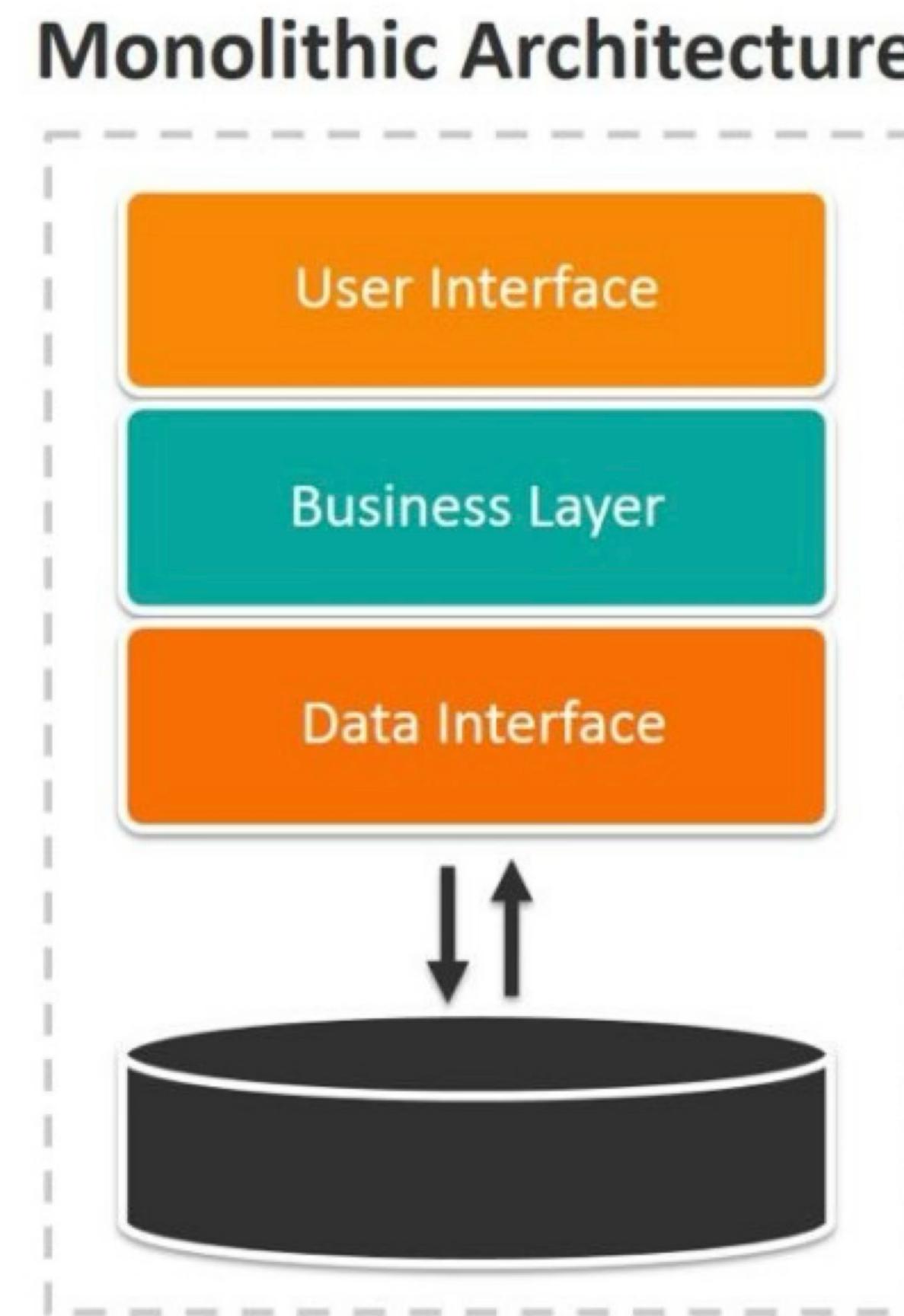


Spring Security



Gestión de seguridad

Spring



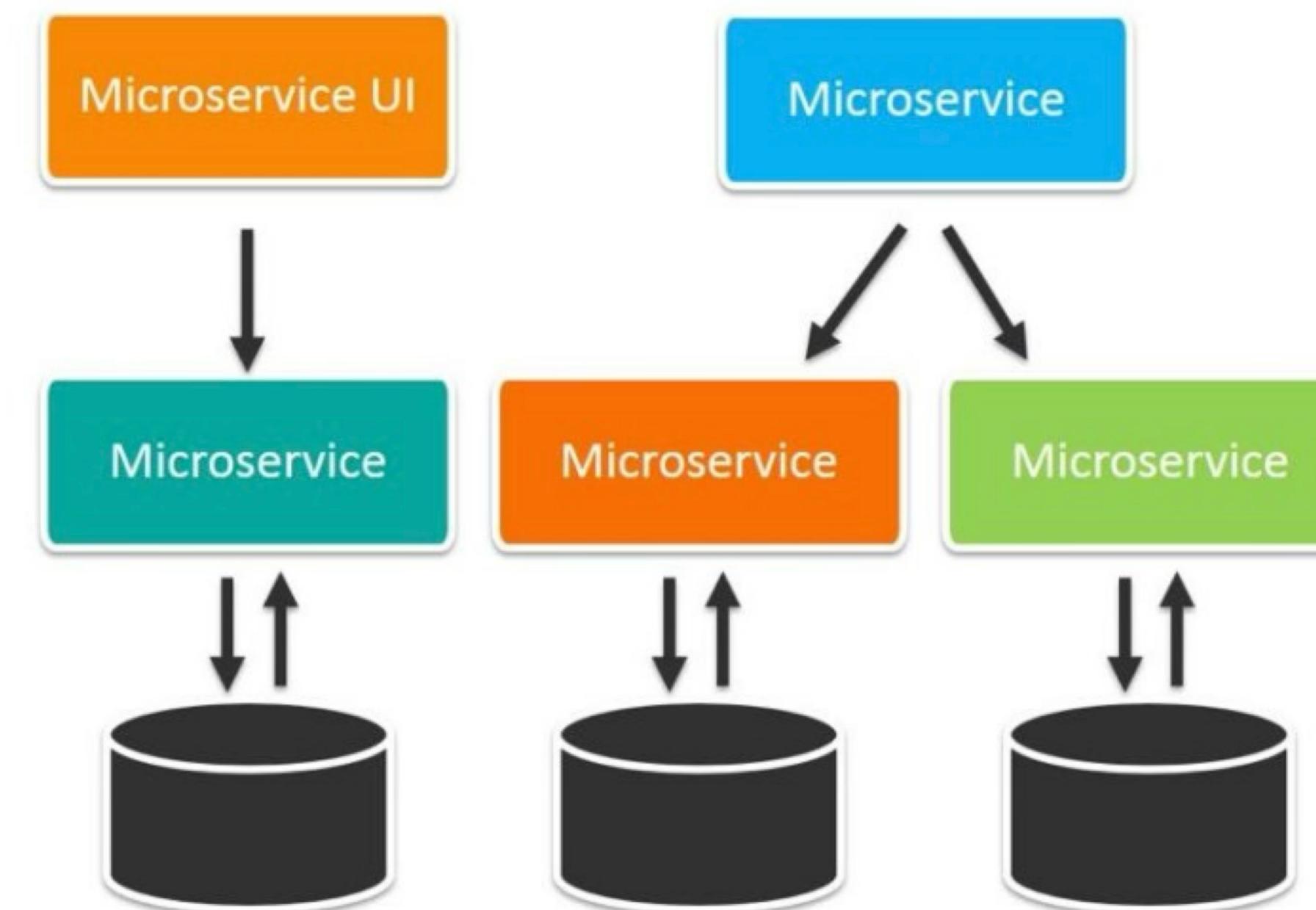
En lugar de dividir el funcionamiento completo de la aplicación en varias capas

Spring

Se divide el funcionamiento de la aplicación en un conjunto de servicios que interactúan entre si.

Cada servicio tiene su propio servidor de aplicaciones con su configuración particular, lo que aumenta la flexibilidad de la aplicación.

Microservices Architecture



Spring Boot

- Creación del proyecto



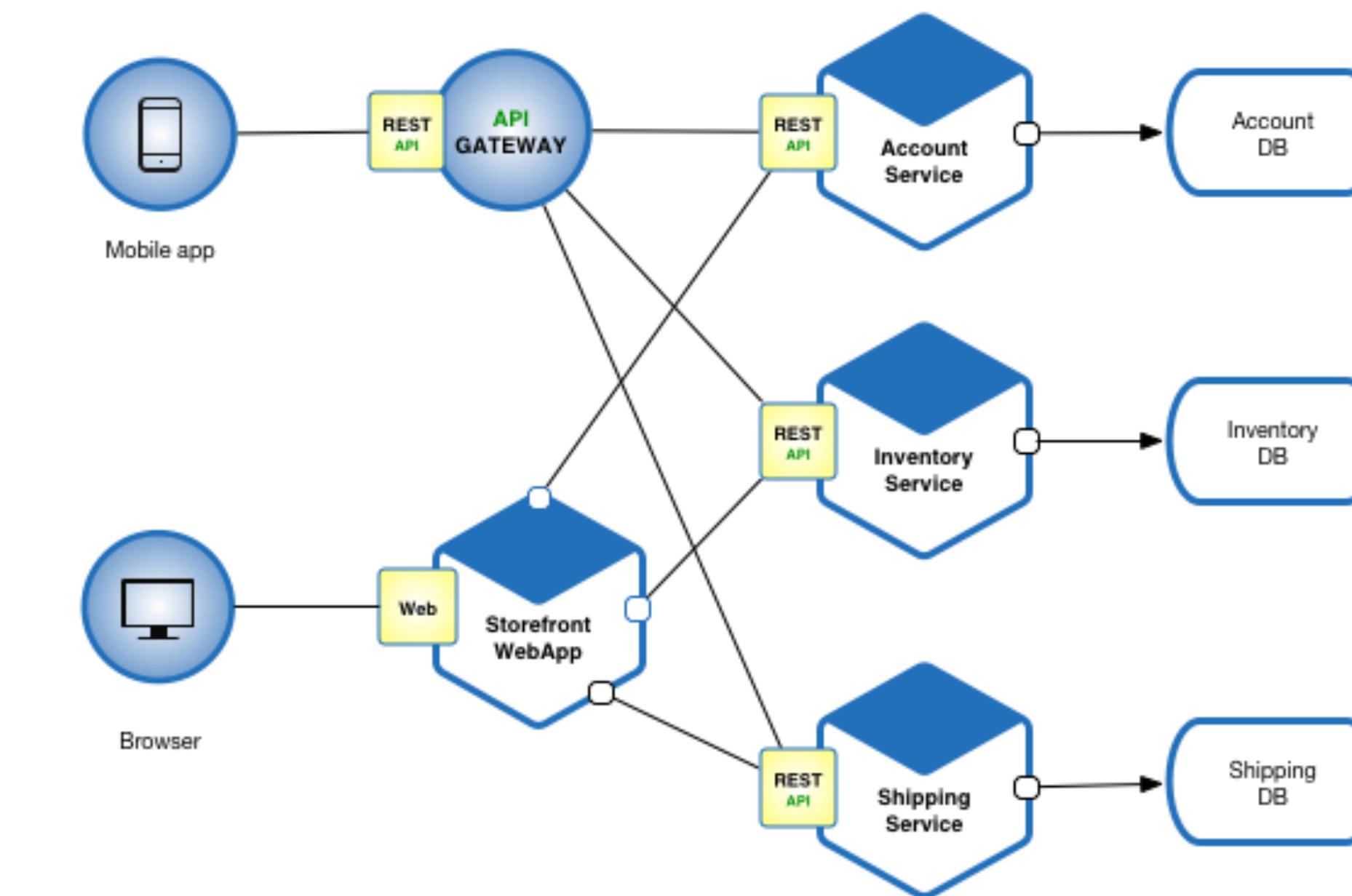
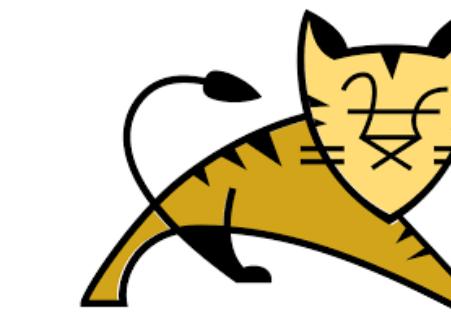
Spring Boot

- Creación del proyecto

Spring Boot facilita la creación y configuración de cada uno de los servidores así como el despliegue de la aplicación

Crea un servidor dentro de un contenedor y despliega la aplicación dentro del mismo

Nota: Por omisión el servidor creado es apache - Tomcat



Spring Boot

- Creación del proyecto



spring initializr

Spring Boot

- Creación del proyecto

The screenshot shows the Spring Initializr web interface at <https://start.spring.io/>. The page has a header with a back arrow, forward arrow, refresh button, and a search bar containing "start.spring.io". Below the header is a toolbar with various links like Yahoo!, Google Maps, YouTube, Wikipedia, Doctorado, Noticias, Populares, and a link to "Todos los favoritos". On the left, there's a sidebar with a menu icon and a clock icon. The main content area features the "spring initializr" logo. It includes sections for "Project" (Gradle - Groovy, Gradle - Kotlin, Maven), "Language" (Java, Kotlin, Groovy), "Spring Boot" (3.5.0 (SNAPSHOT), 3.5.0 (RC1), 3.4.6 (SNAPSHOT), 3.4.5, 3.3.12 (SNAPSHOT), 3.3.11), and "Project Metadata". A "Dependencies" section with a "No dependency selected" message and a "ADD DEPENDENCIES..." button is also present. To the right of the dependencies section is a small icon of a sun and moon.

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

Spring Boot

Llenar como se muestra:



- Project**
- Gradle - Groovy
 - Gradle - Kotlin
 - Maven

- Language**
- Java
 - Kotlin
 - Groovy

- Spring Boot**
- 4.0.1 (SNAPSHOT)
 - 4.0.0
 - 3.5.9 (SNAPSHOT)
 - 3.5.8
 - 3.4.13 (SNAPSHOT)
 - 3.4.12

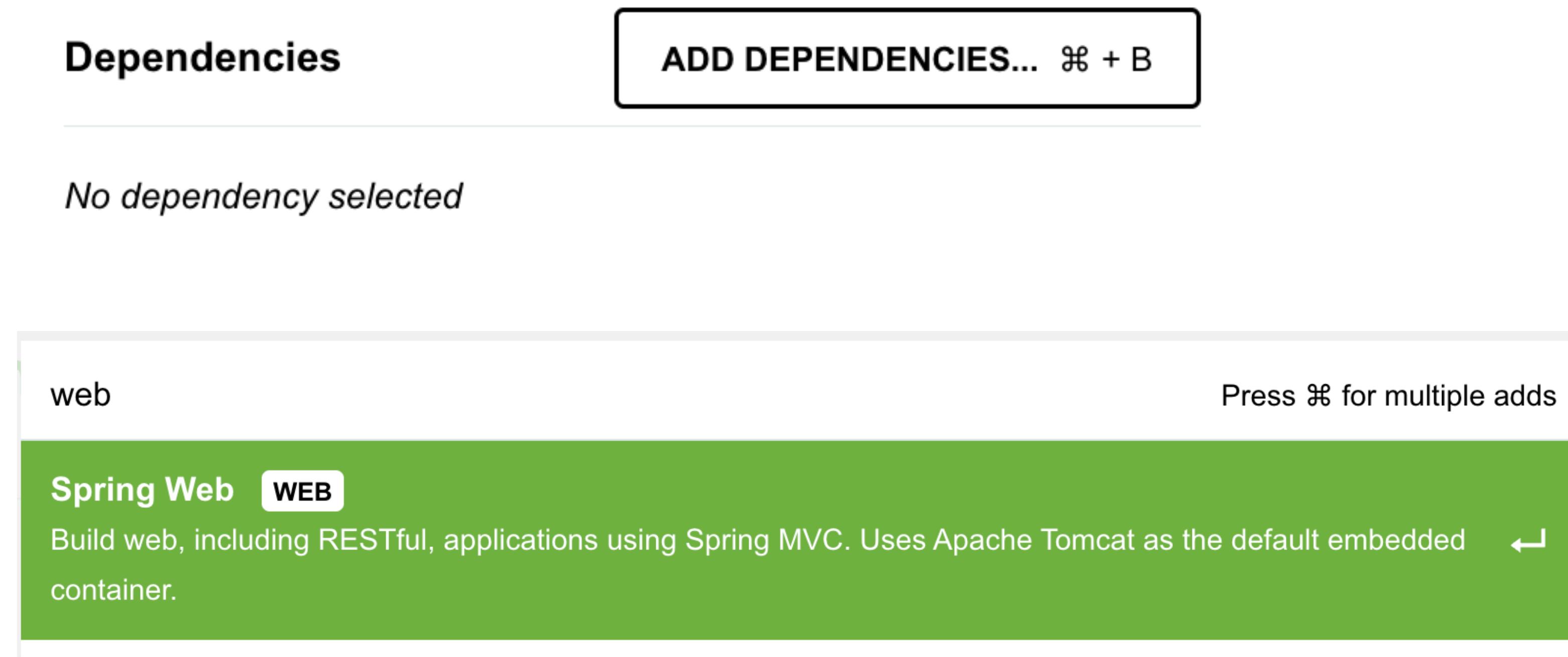
Versión estable

Project Metadata

Group	mx.uaemex.fi
Artifact	nrda
Name	nrda
Description	Administrador de usuarios
Package name	mx.uaemex.fi.nrda
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Configuration	<input checked="" type="radio"/> Properties <input type="radio"/> YAML
Java	<input type="radio"/> 25 <input checked="" type="radio"/> 21 <input type="radio"/> 17

Spring Boot

- Dependencias



Spring Boot

- Dependencias

Dependencies

ADD DEPENDENCIES... ⌘ + B

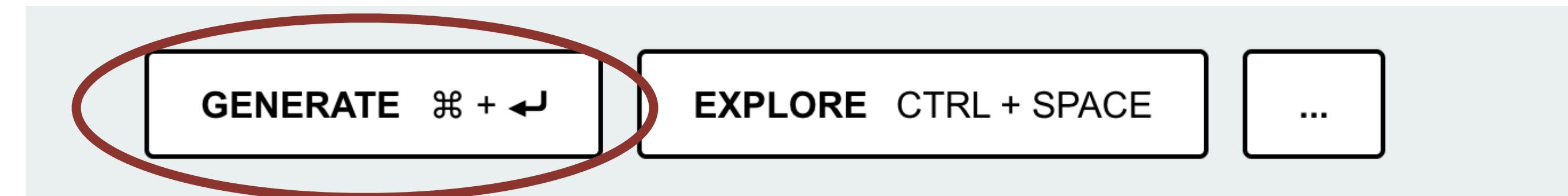
Spring Web WEB

Build web, including RESTful, applications using Spring MVC.

Uses Apache Tomcat as the default embedded container.

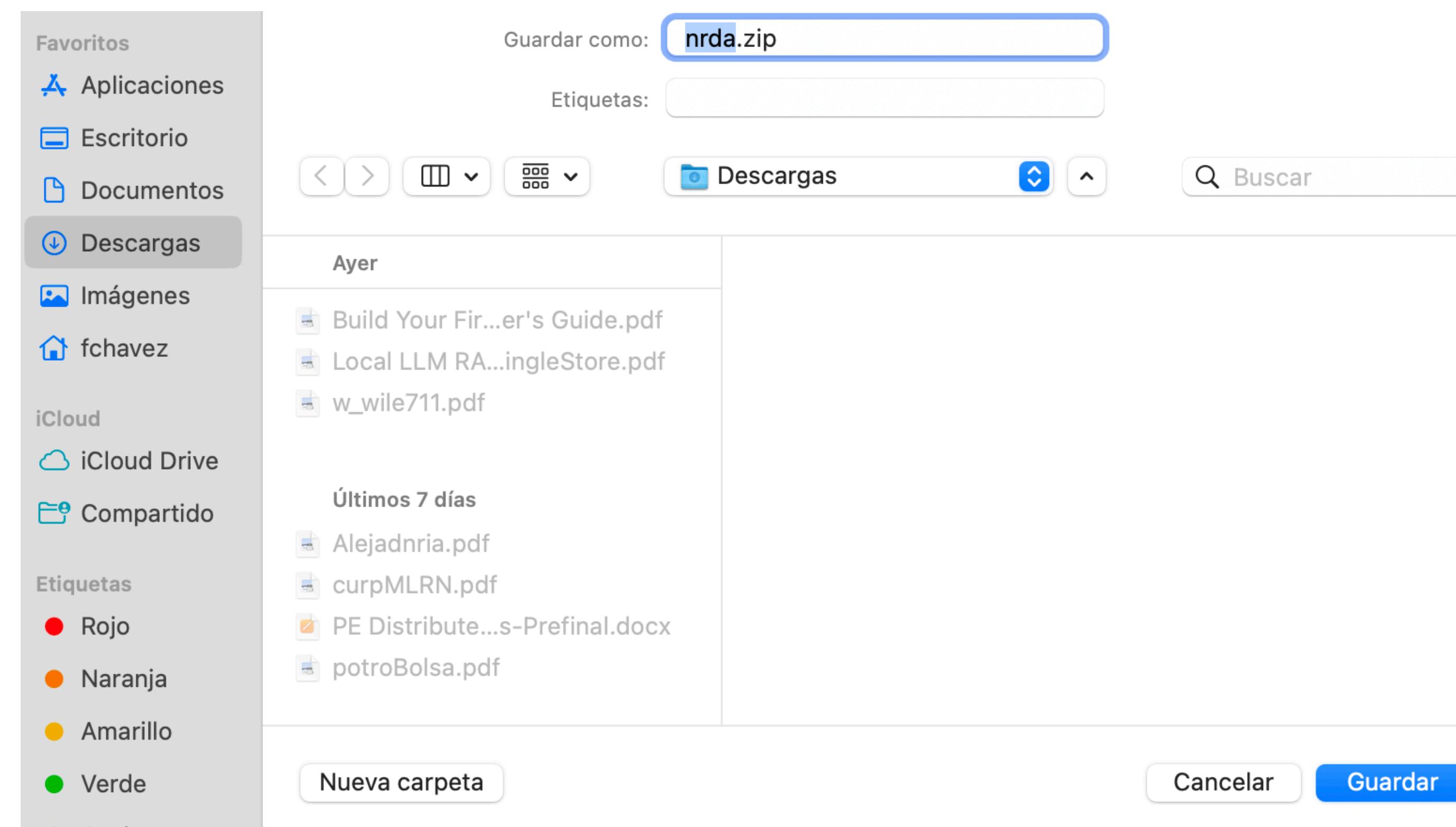
Spring Boot

- Generar



Spring Boot

- Generar



Spring Boot

- Descompactas



¡Proyecto creado de forma exitosa!

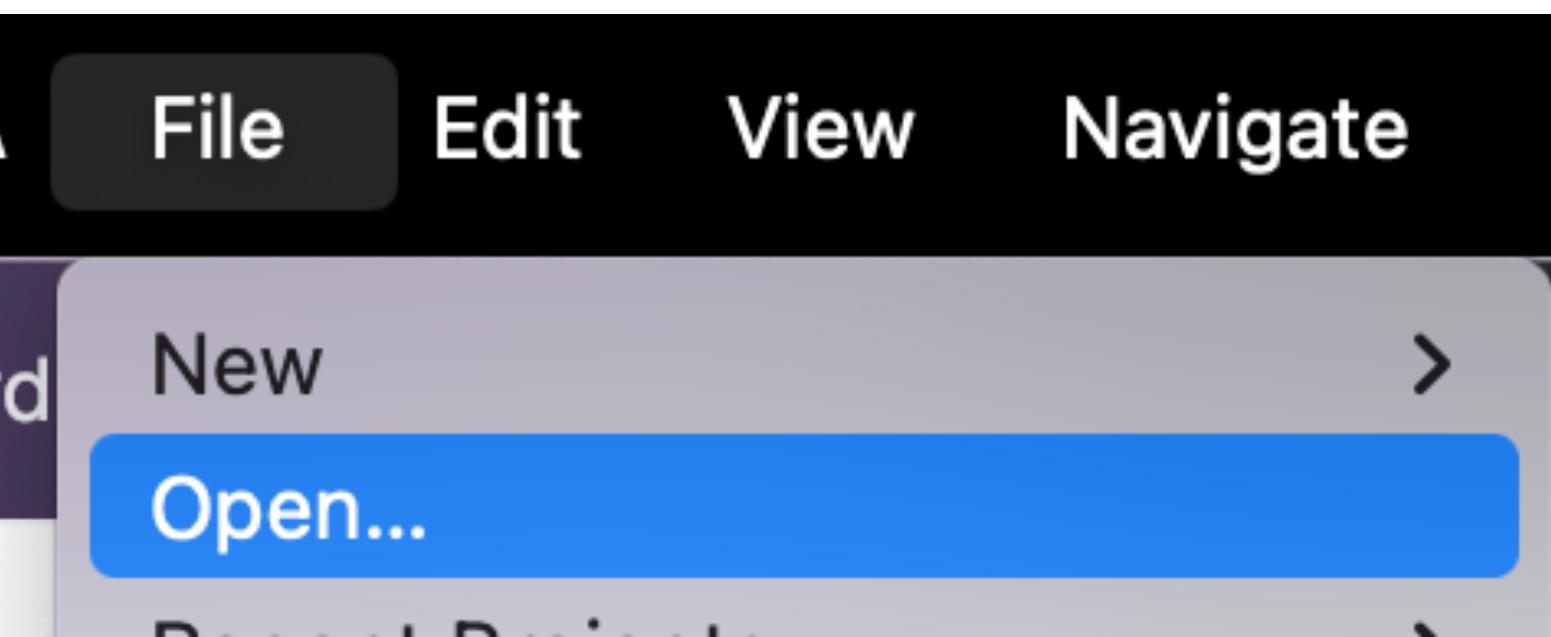
Spring Boot

- Programación del proyecto



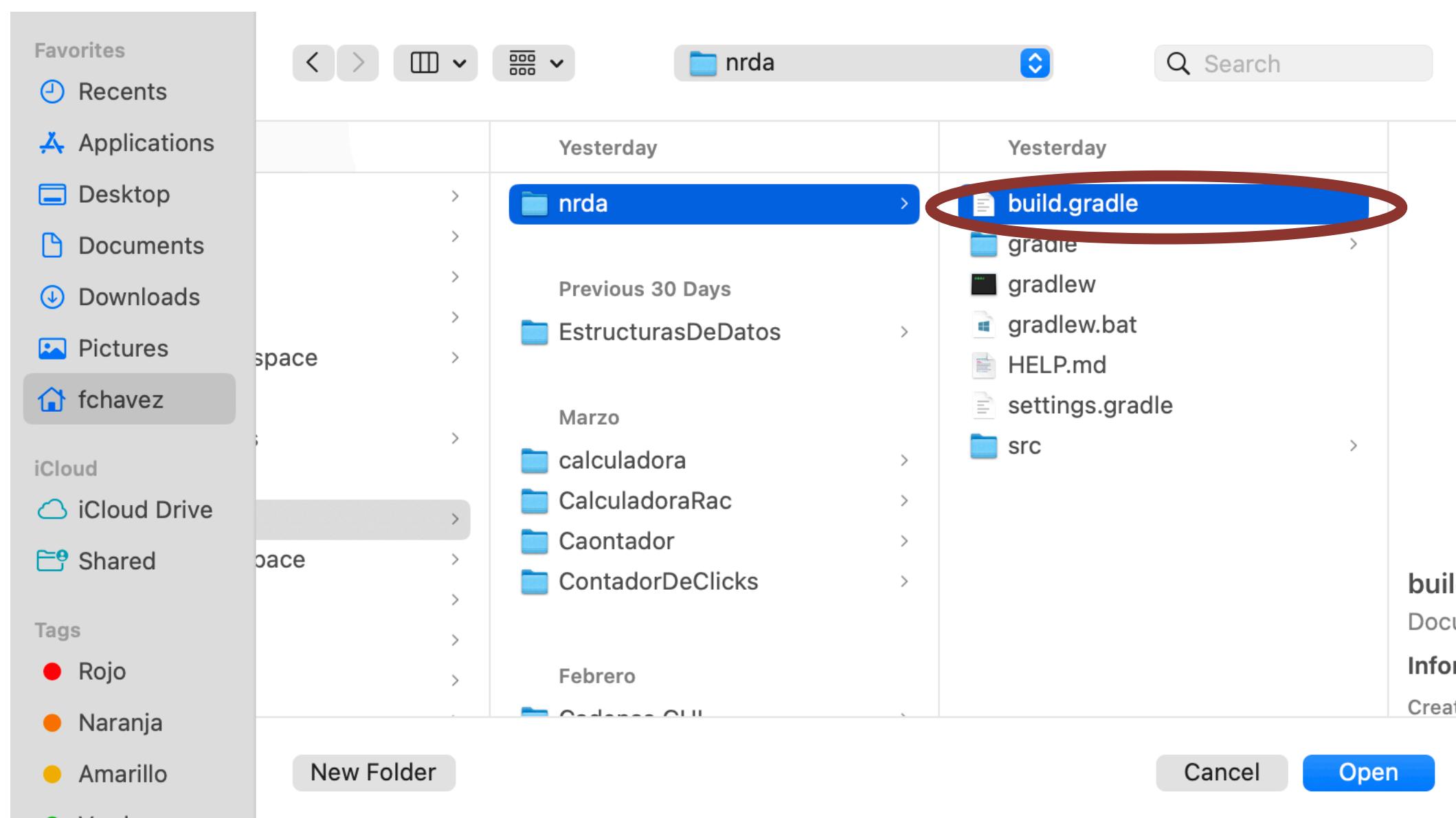
Spring Boot

- Abrir el proyecto con el IDE



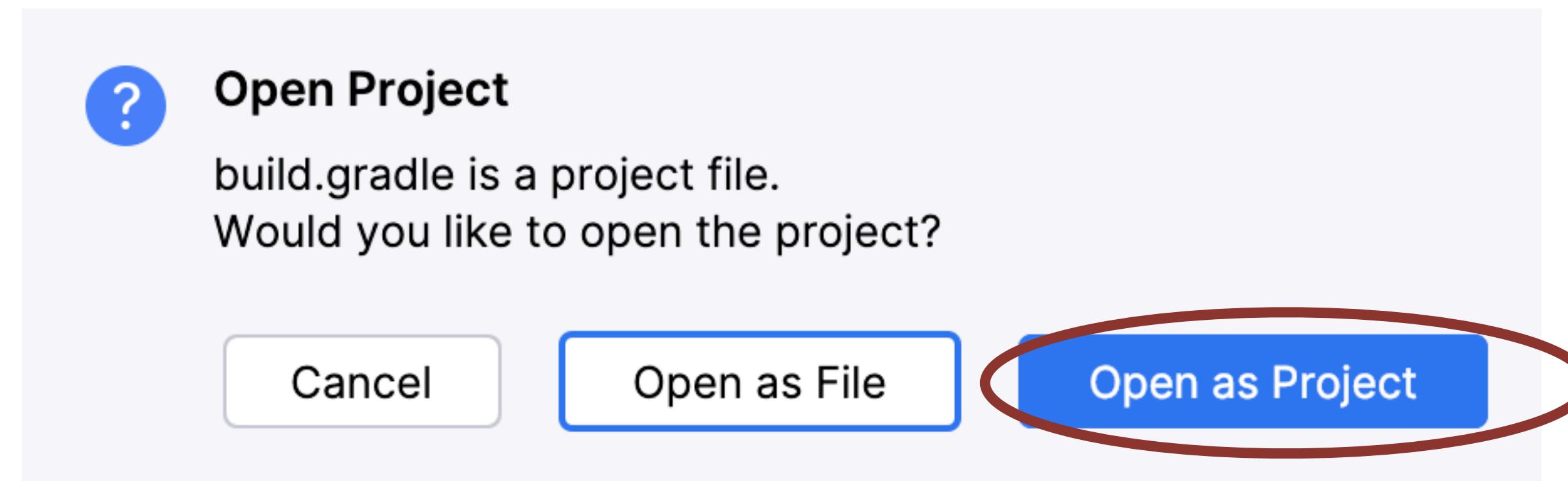
Spring Boot

- Abrir el proyecto con el IDE



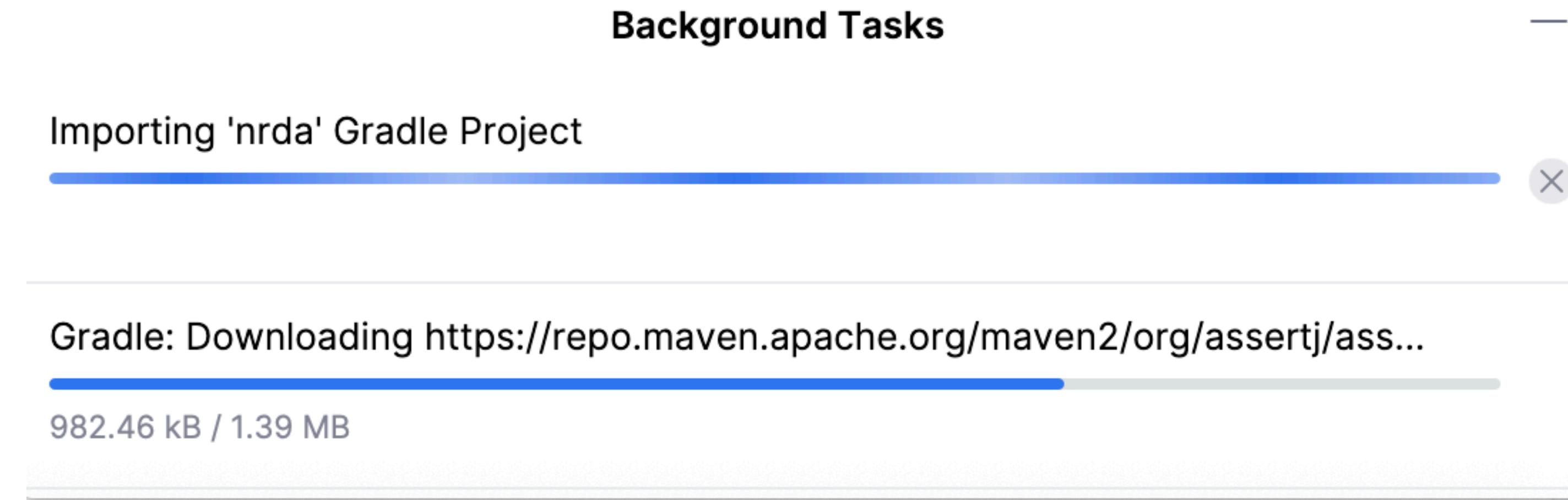
Spring Boot

- Abrir el proyecto con el IDE



Spring Boot

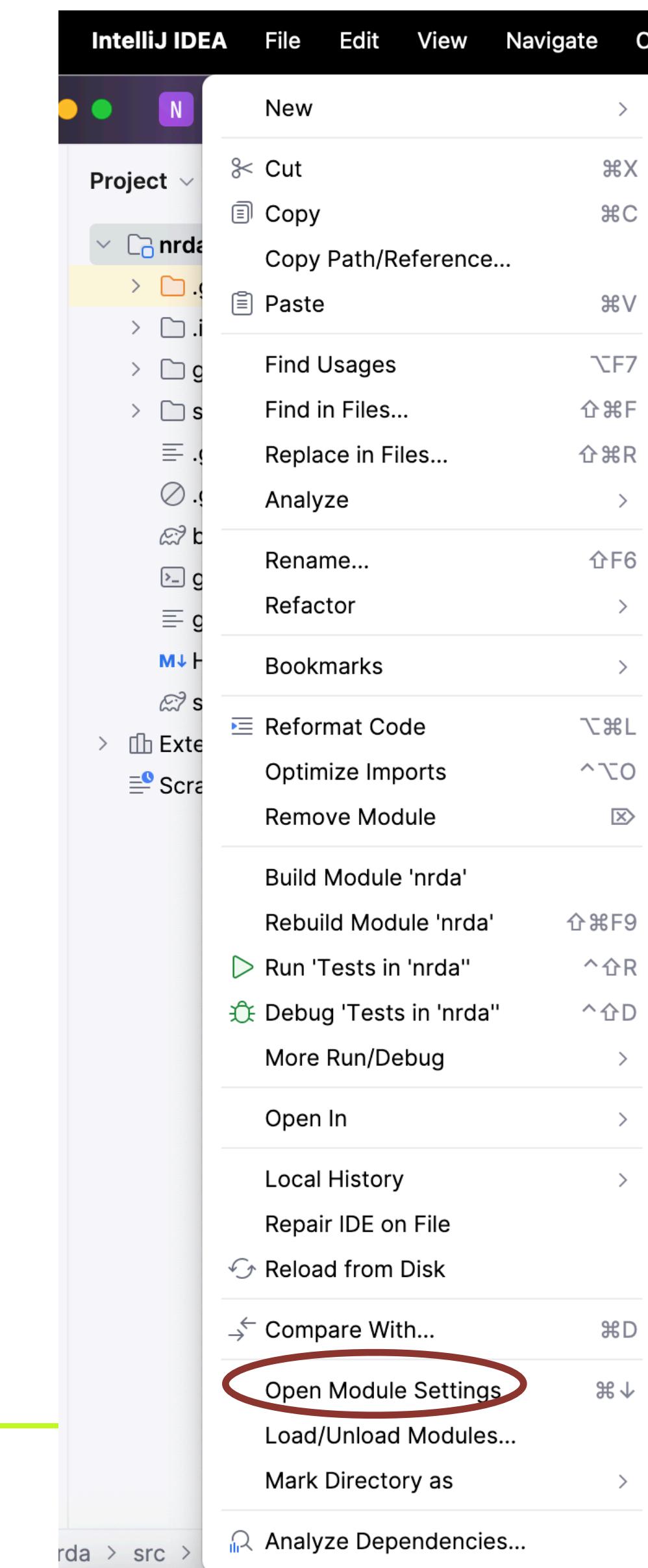
- Tomará un tiempo que el IDE descargue dependencias



Spring Boot

- Revisemos

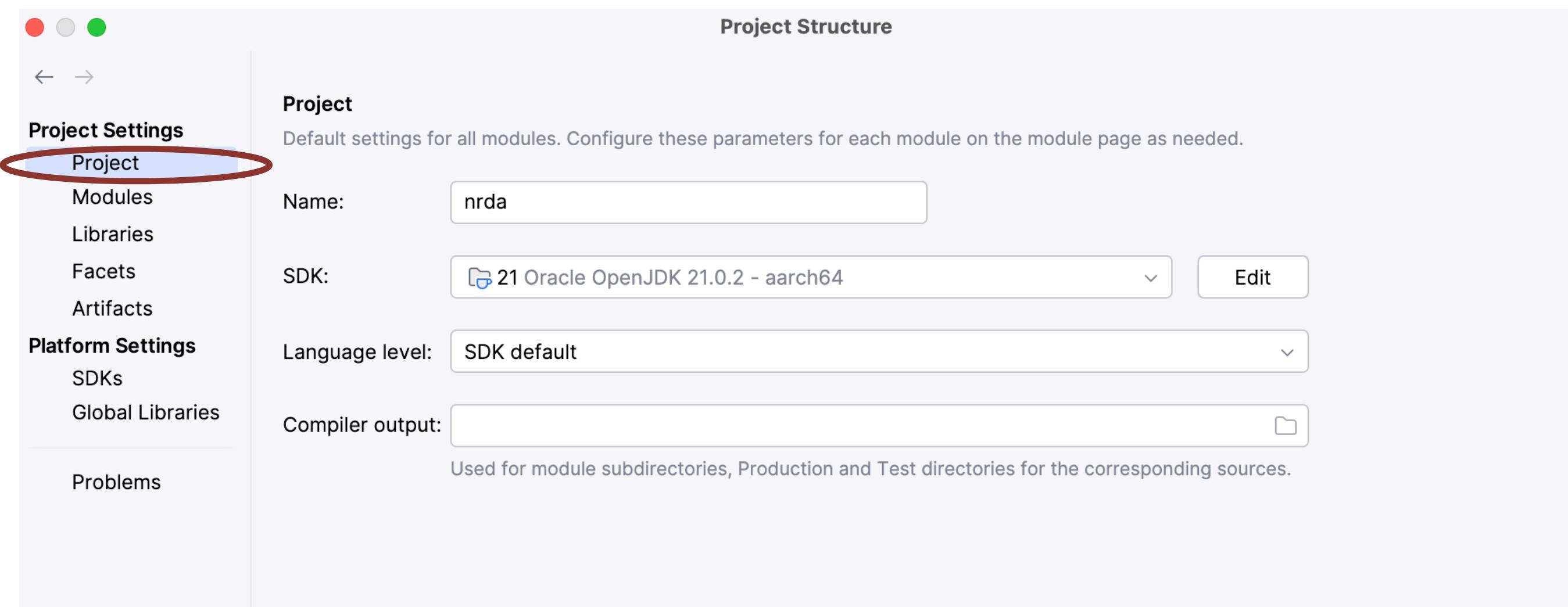
Click derecho sobre el proyecto



Spring Boot

- Revisemos

Click derecho sobre el proyecto



Spring Boot

- Revisemos

← →

Project

Default settings for all modules. Configure these parameters for each module on the module page as needed.

Project Settings

- Project
- Modules
- Libraries
- Facets
- Artifacts

Platform Settings

- SDKs
- Global Libraries

Name: nrda

SDK: 23 Oracle OpenJDK 23.0.2 - aarch64

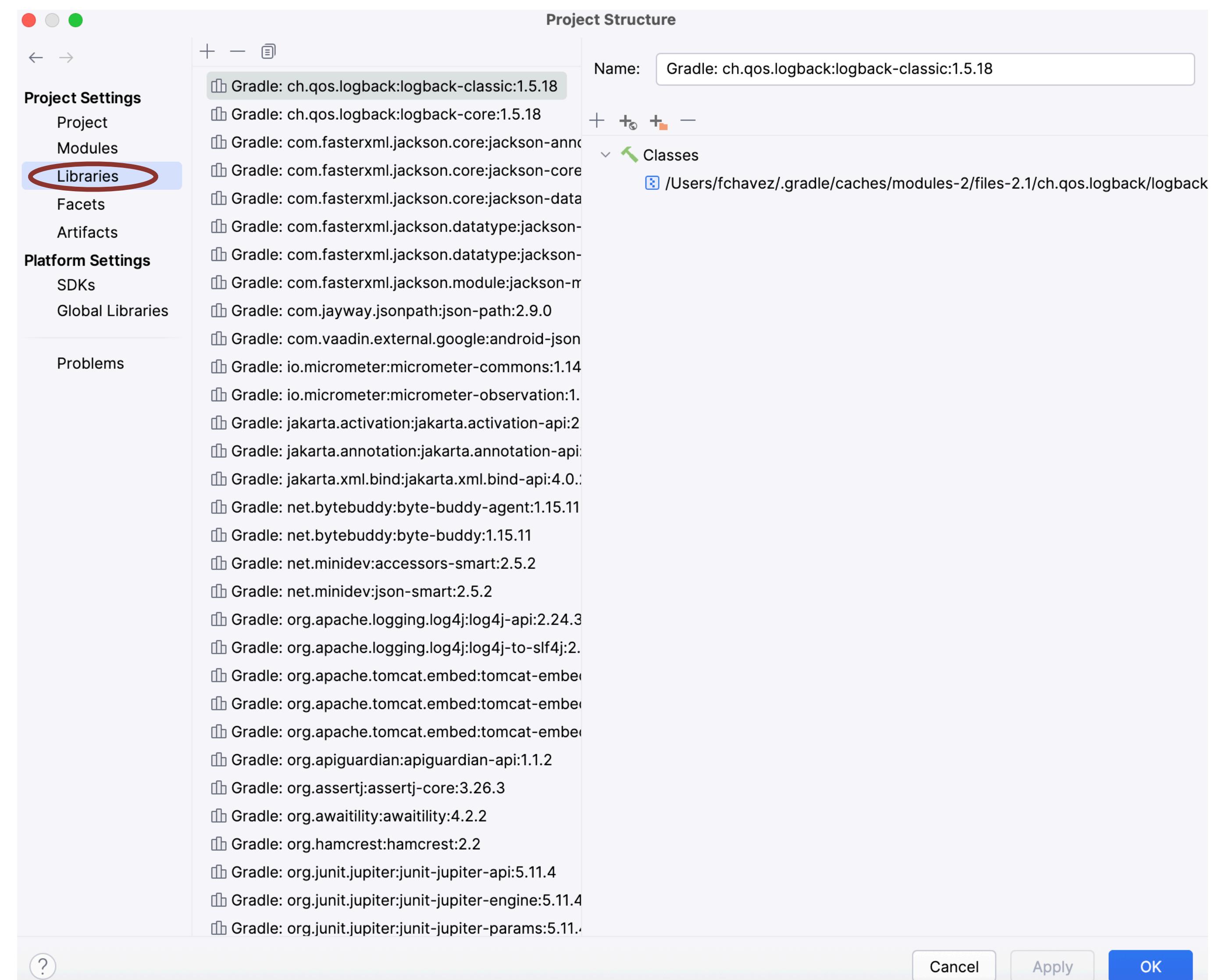
Language level: 21 - Record patterns, pattern matching for switch

Compiler output:

Used for module subdirectories, Production and Test directories for the corresponding sources.

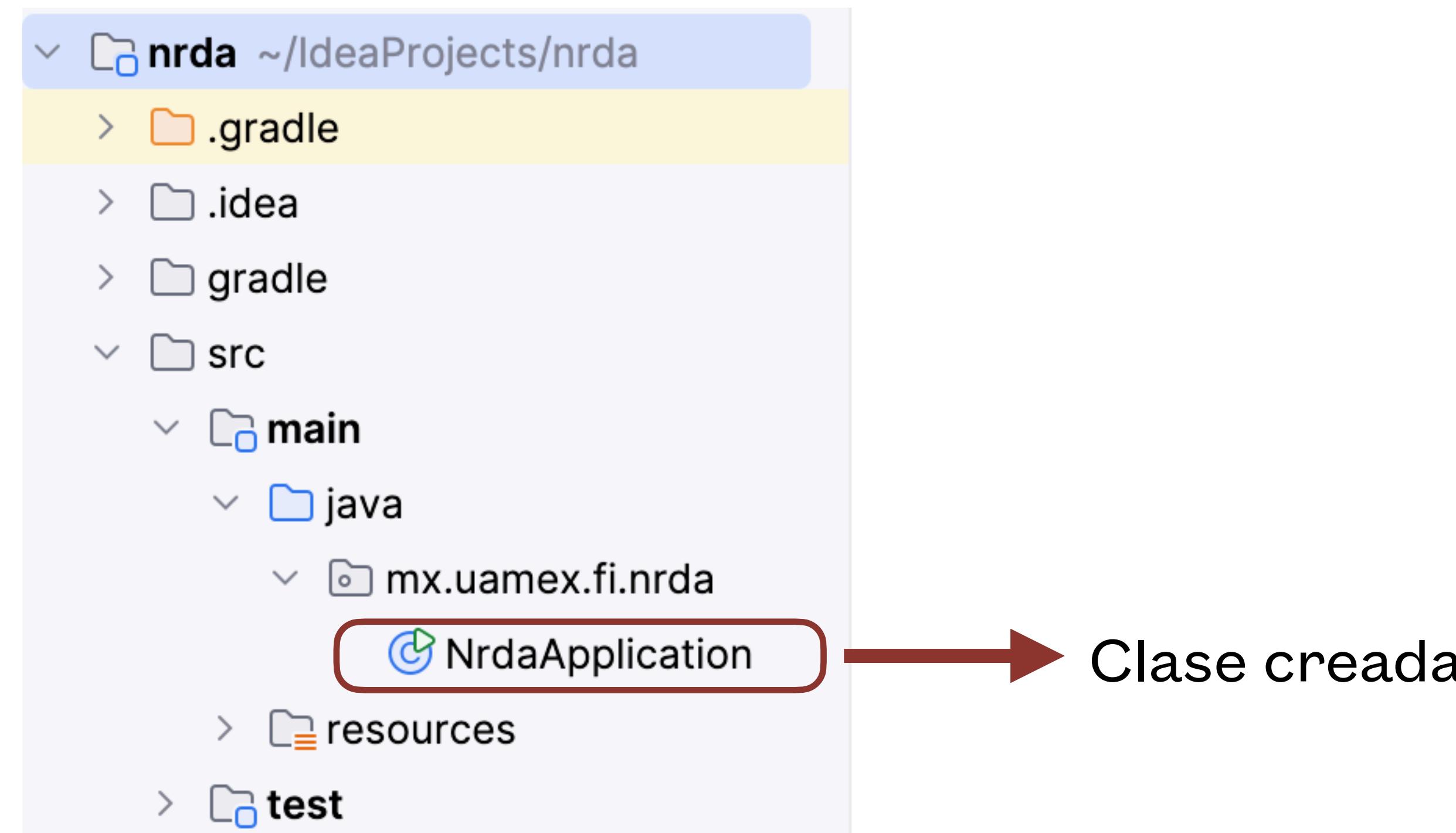
Spring Boot

- Dependencias descargadas



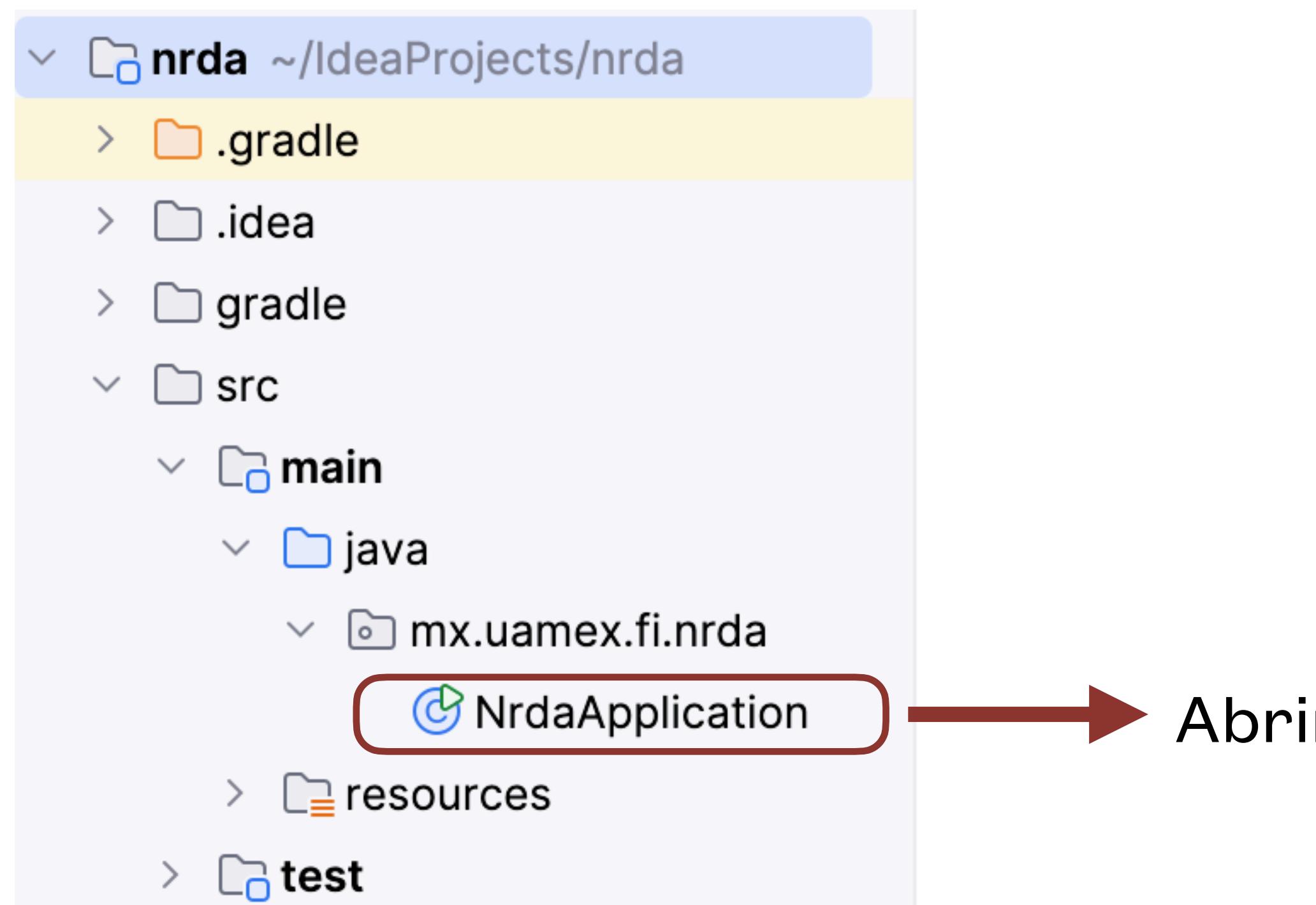
Spring Boot

- Estructura del proyecto



Spring Boot

- Estructura del proyecto



Spring Boot

NrdaApplication.java

```
1 package mx.uamex.fi.nrda;  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  
6 @SpringBootApplication  
7 public class NrdaApplication {  
8  
9     public static void main(String[] args) {  
10         SpringApplication.run(NrdaApplication.class, args);  
11     }  
12 }
```



Anotación que marca esta clase como la principal de la aplicación auto-contenida

Spring Boot

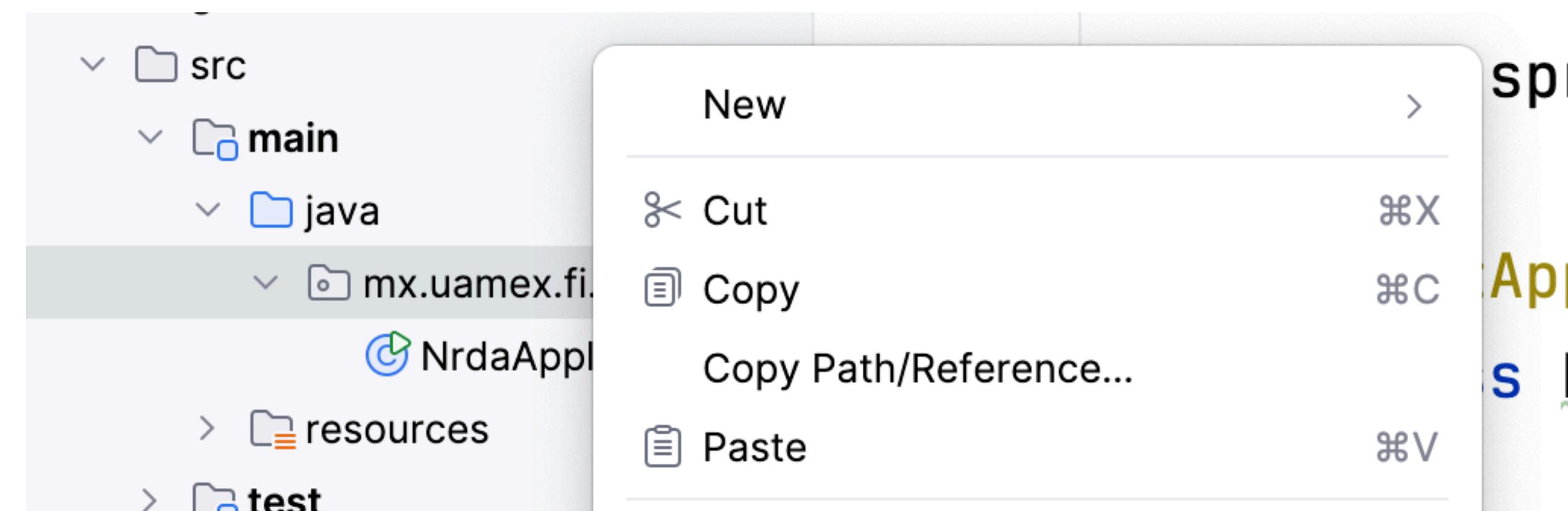
Spring está basado en MVC por lo que para que la aplicación haga algo, necesitamos un ...



Controller

Spring Boot

Click derecho sobre el proyecto



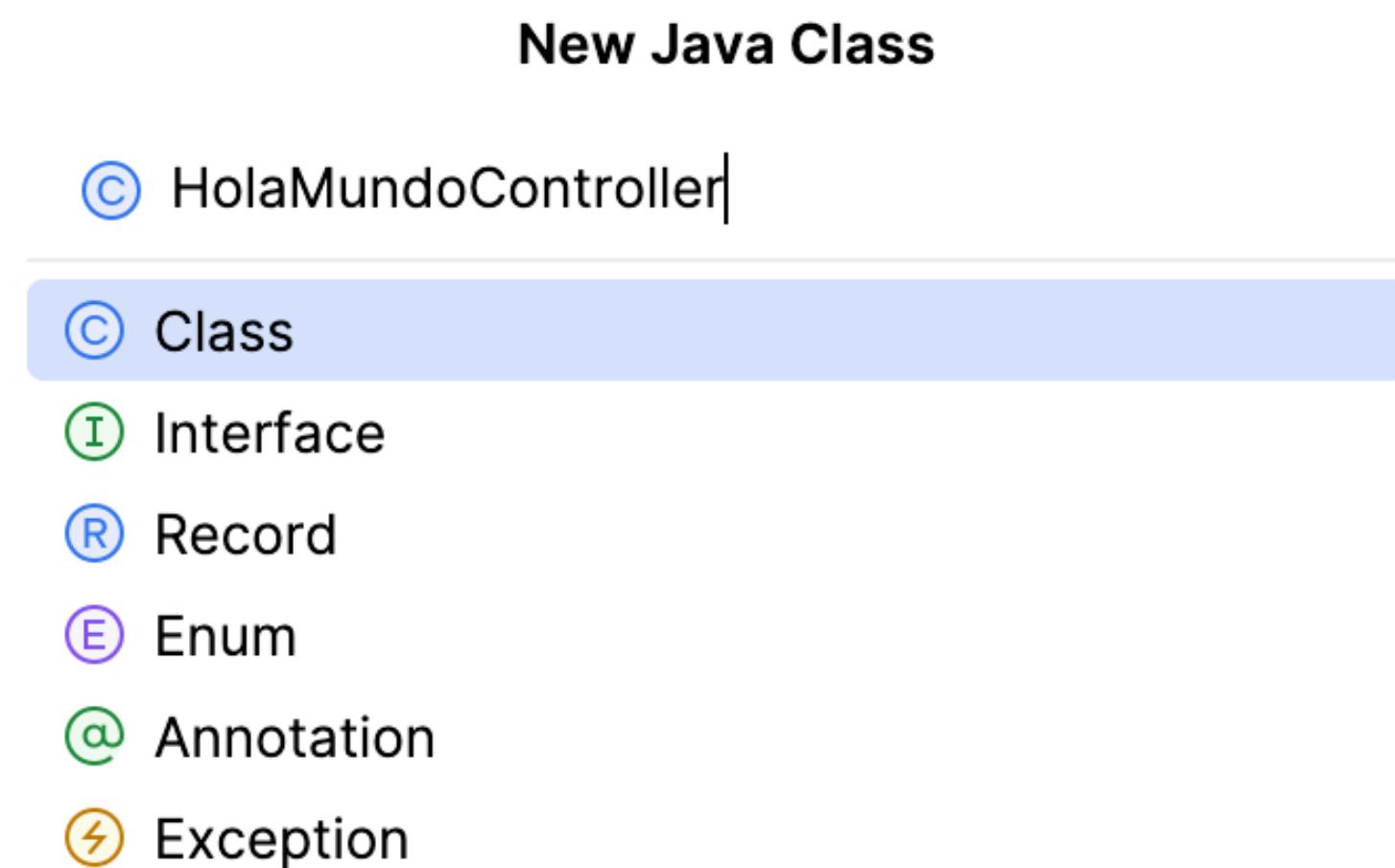
Spring Boot

Click derecho sobre el proyecto



Spring Boot

Click derecho sobre el proyecto

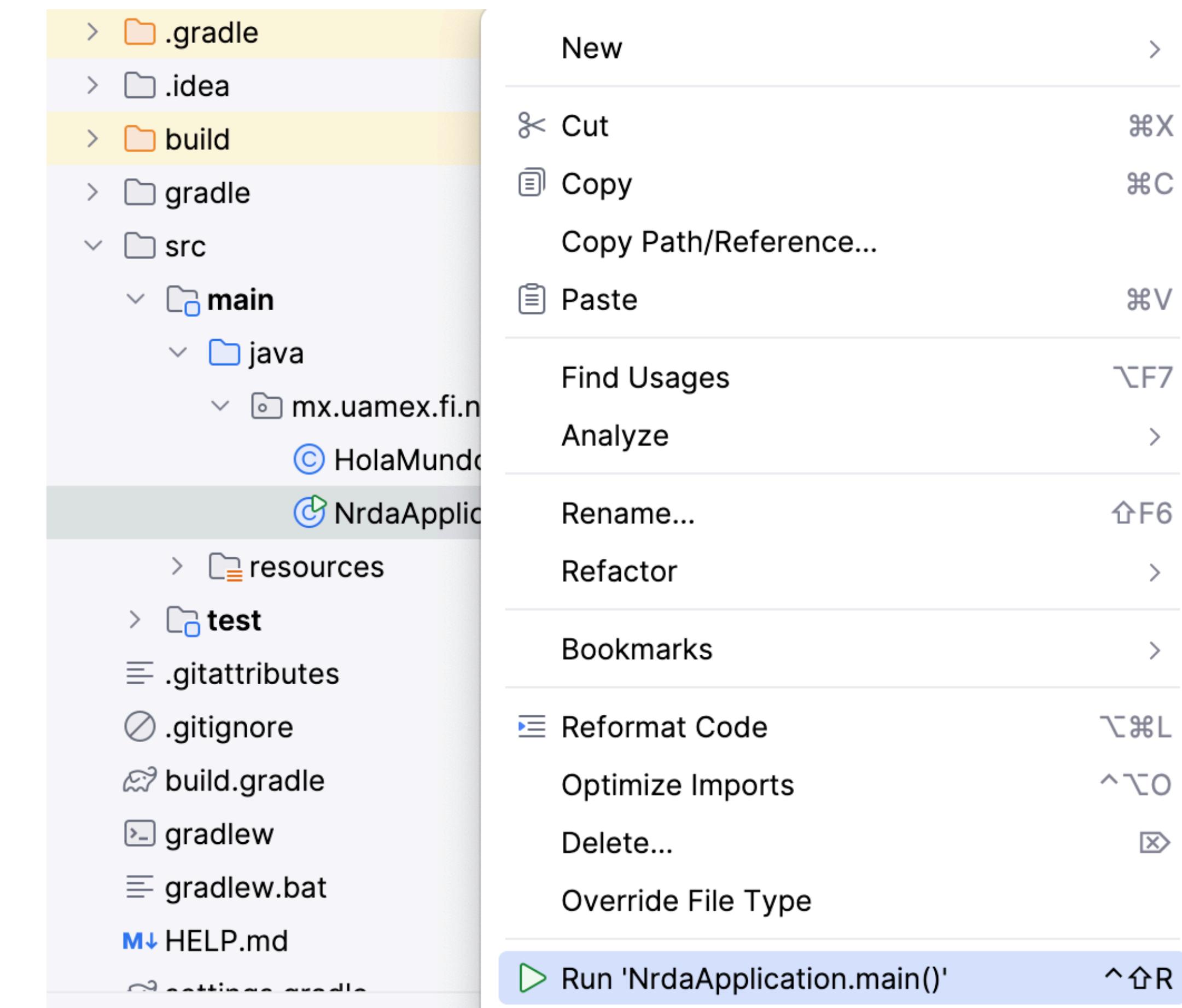


Spring Boot

```
1 package mx.uamex.fi.nrda;  
2  
3 import org.springframework.web.bind.annotation.GetMapping;  
4 import org.springframework.web.bind.annotation.RequestMapping;  
5 import org.springframework.web.bind.annotation.RestController;  
6  
7 @RestController  
8 @RequestMapping("/saluda")  
9 public class HolaMundoController {  
10     @GetMapping("/hola")  
11     public String holaMundo() {  
12         return "Hola, Mundo";  
13     }  
14 }  
15 }
```

Spring Boot

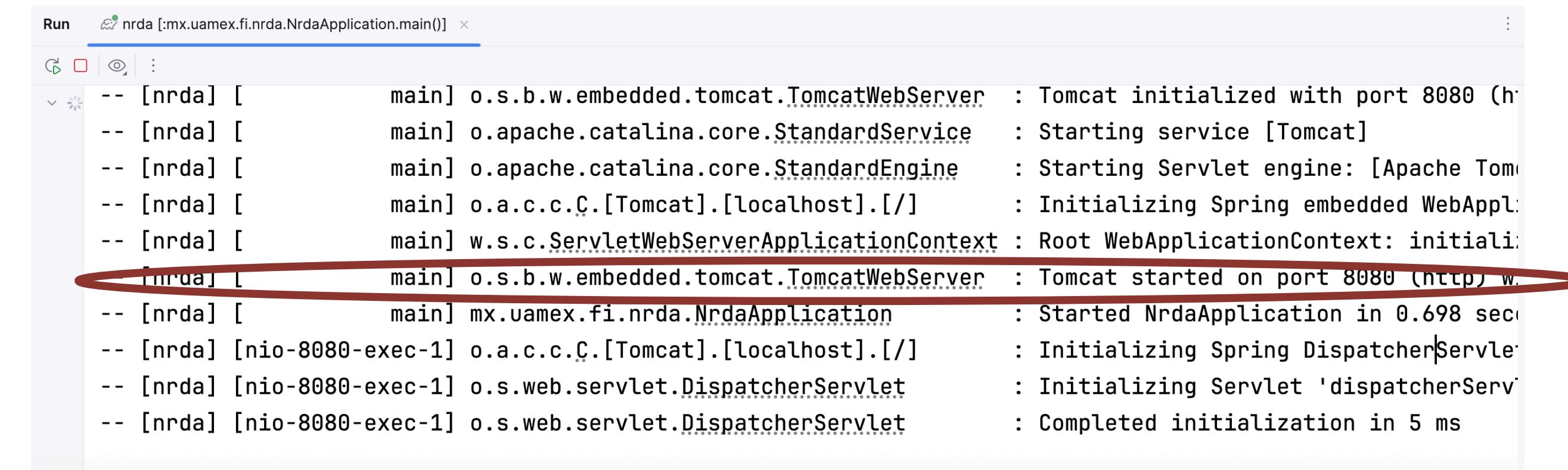
Ejecutar: Click derecho sobre



Spring Boot

- **¿Qué pasa cuando ejecutamos la aplicación?**

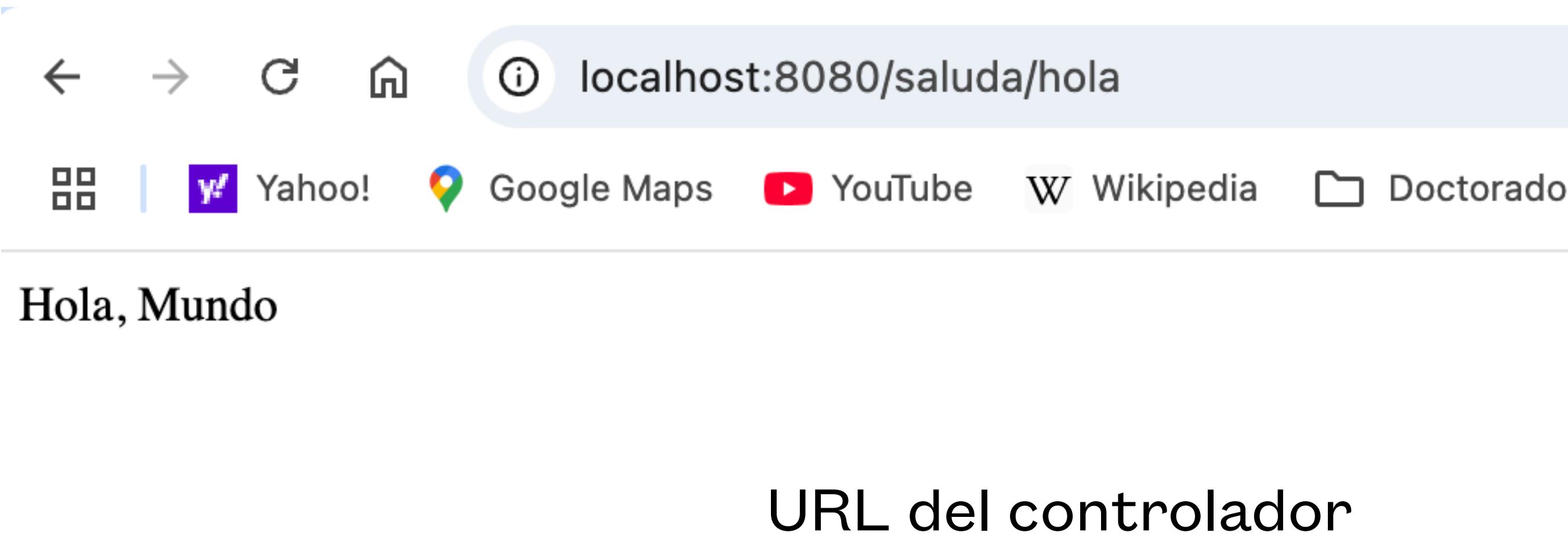
- Spring, construye e instancia un servidor web (tomcat)
- Programa y compila una aplicación web
- Despliega la aplicación web en el servidor creado



```
Run nrda [:mx.uamex.fi.nrda.NrdaApplication.main()] ×  
-- [nrda] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http://localhost:8080)  
-- [nrda] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]  
-- [nrda] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.43]  
-- [nrda] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationEnvironment in container [Tomcat]  
-- [nrda] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialized at [2023-09-11T14:11:41.144+00:00]  
-- [nrda] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http://localhost:8080) with context path /  
-- [nrda] [main] mx.uamex.fi.nrda.NrdaApplication : Started NrdaApplication in 0.698 seconds (JVM running for 0.709)  
-- [nrda] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet  
-- [nrda] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'  
-- [nrda] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 5 ms
```

Spring Boot

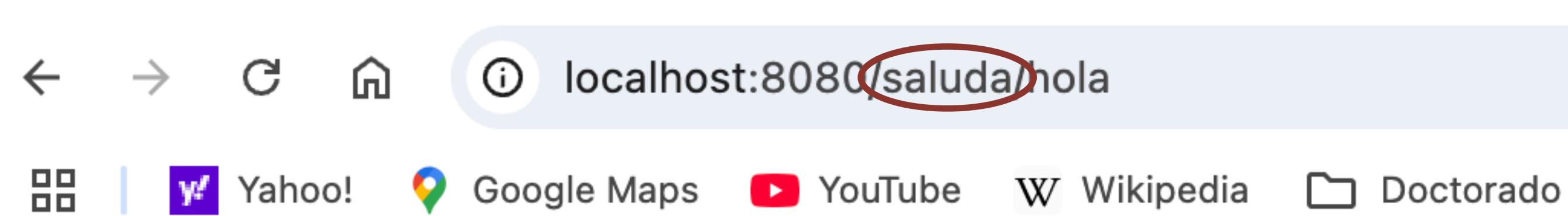
- Solicitamos el url que programamos



```
@RestController no usages
@RequestMapping("/saluda")
public class HolaMundoController {
    @GetMapping("/hola") no usages
    public String holaMundo() {
        return "Hola, Mundo";
    }
}
```

Spring Boot

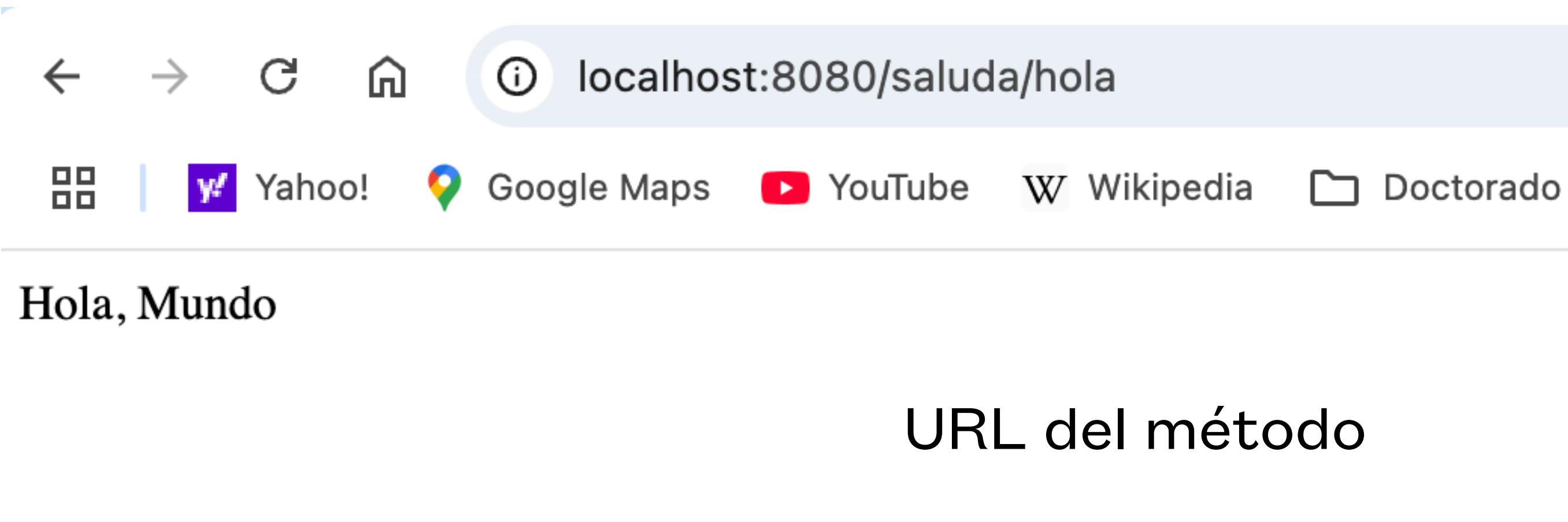
- Solicitamos el url que programamos



```
@RestController no usages
@RequestMapping("/saluda")
public class HolaMundoController {
    @GetMapping("/hola") no usages
    public String holaMundo() {
        return "Hola, Mundo";
    }
}
```

Spring Boot

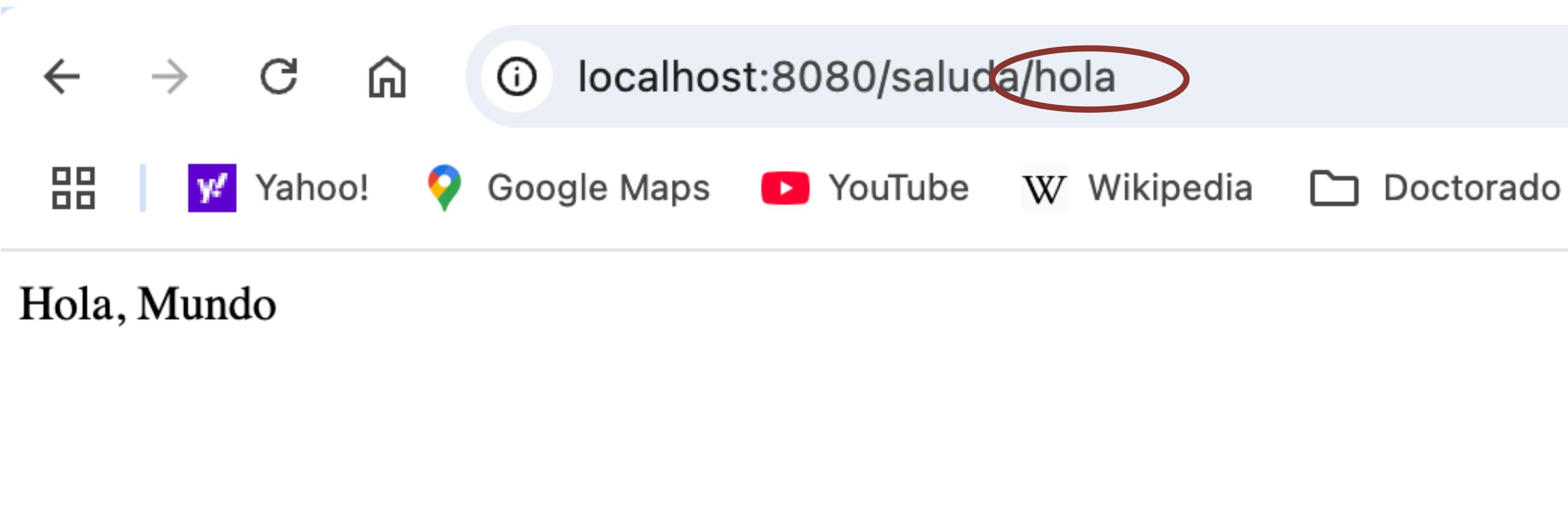
- Solicitamos el url que programamos



```
@RestController no usages
@RequestMapping("/saluda")
public class HolaMundoController {
    @GetMapping("/hola") no usages
    public String holaMundo() {
        return "Hola, Mundo";
    }
}
```

Spring Boot

- Solicitamos el url que programamos



```
@RestController no usages
@RequestMapping("/saluda")
public class HolaMundoController {
    @GetMapping("/hola") no usages
    public String holaMundo() {
        return "Hola, Mundo";
    }
}
```

Spring Boot

- Programación exitosa



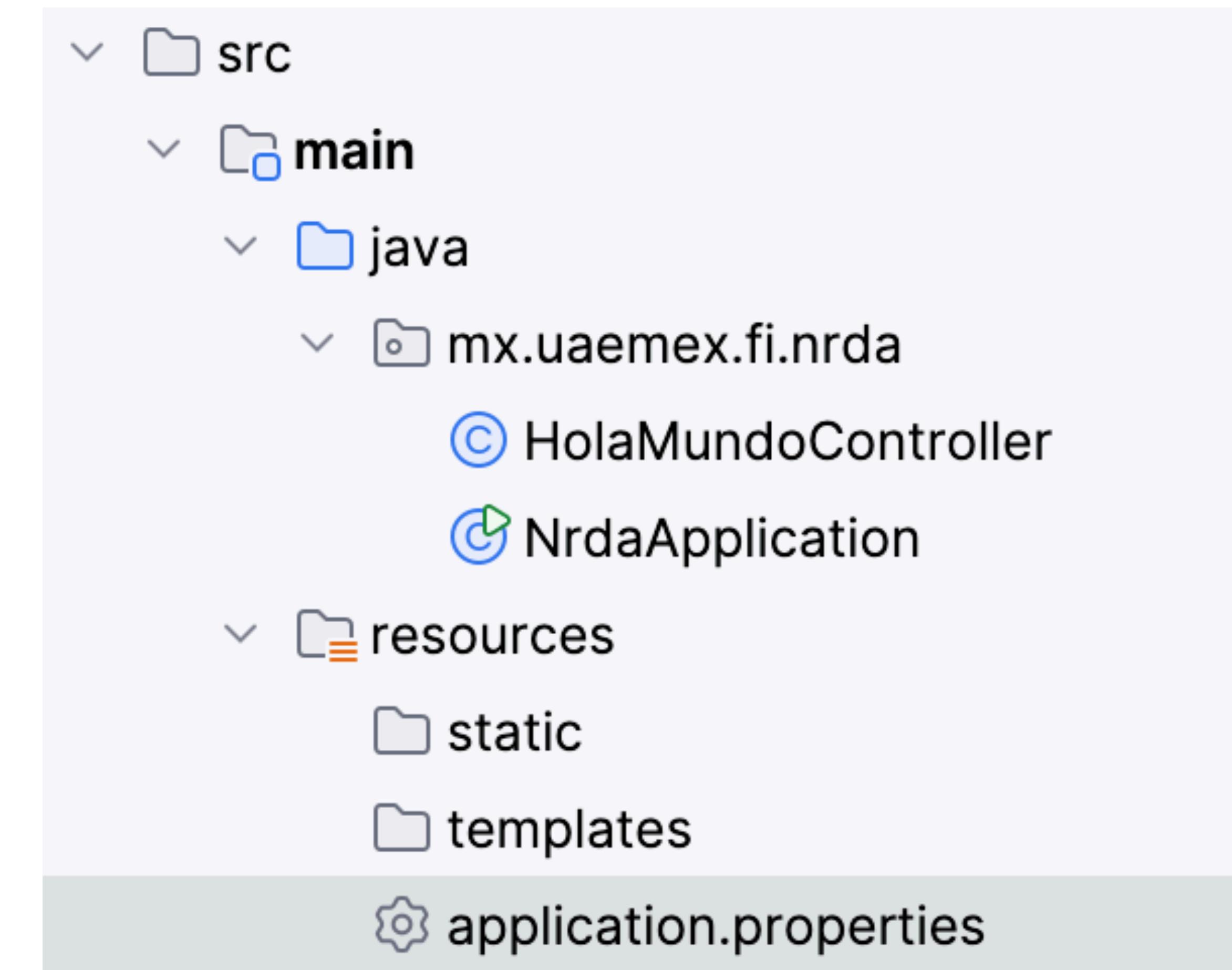
Spring Boot

- Configuración



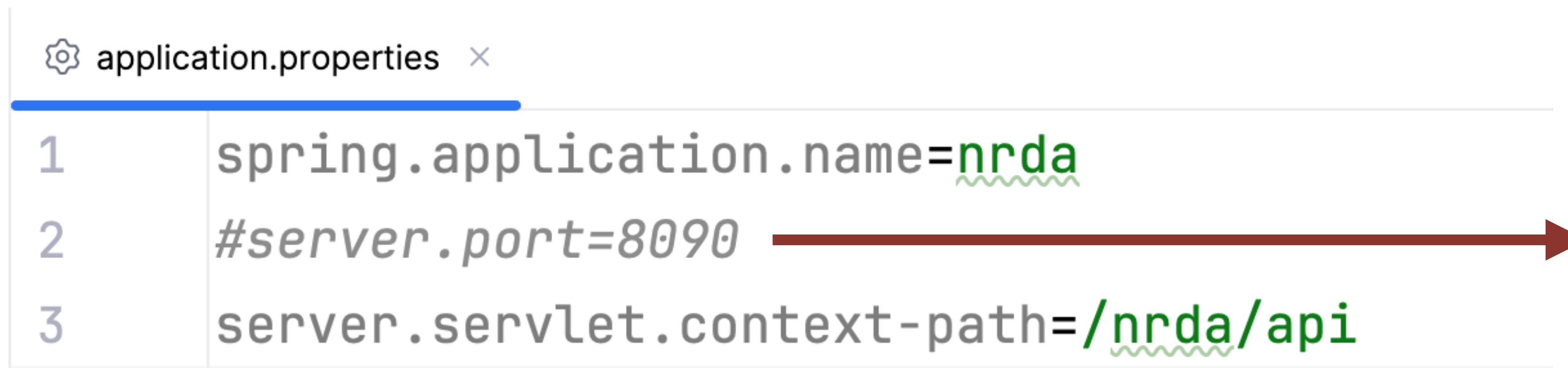
Spring Boot

- Archivo “***application.properties***



Spring Boot

- Archivo “***application.properties***”
 - Sirve para configurar el servidor y/o la aplicación



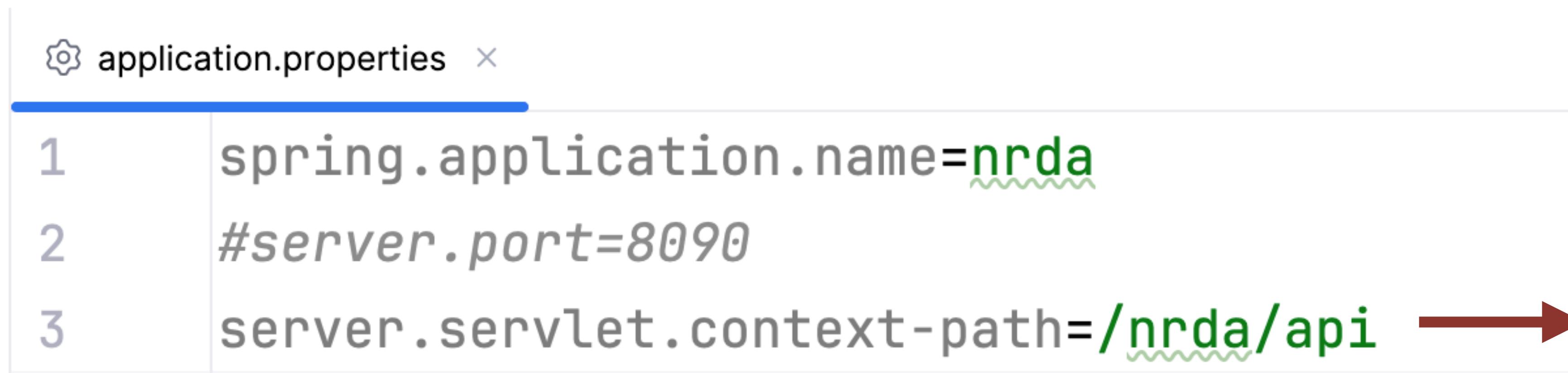
```
application.properties
```

1	spring.application.name=nrda
2	#server.port=8090
3	server.servlet.context-path=/nrda/api

Si no estuviera comentada, cambiaría el puerto

Spring Boot

- **Archivo “*application.properties*”**
 - Sirve para configurar el servidor y/o la aplicación



The screenshot shows a code editor window with the file "application.properties" open. The file contains the following configuration:

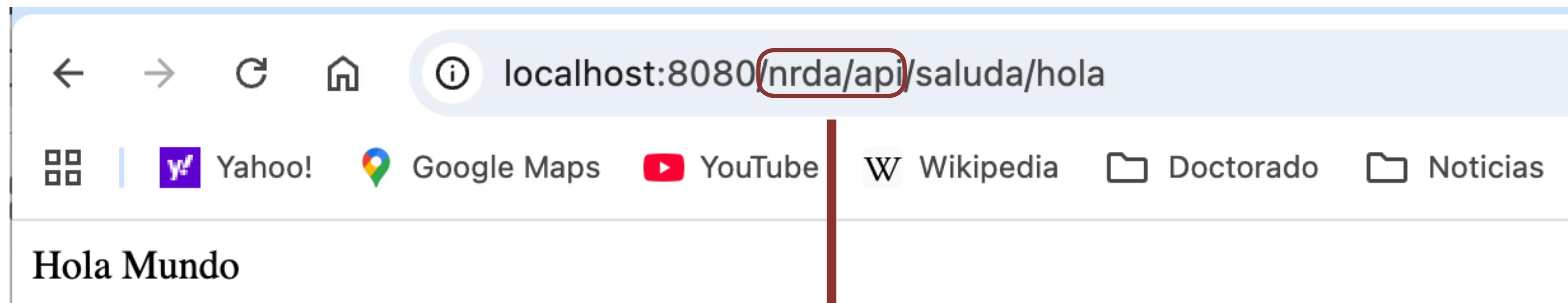
```
1 spring.application.name=nrda
2 #server.port=8090
3 server.servlet.context-path=/nrda/api
```

A red arrow points from the text "server.servlet.context-path=/nrda/api" to the explanatory text "Modifica el URL de la aplicación".

Modifica el URL
de la aplicación

Spring Boot

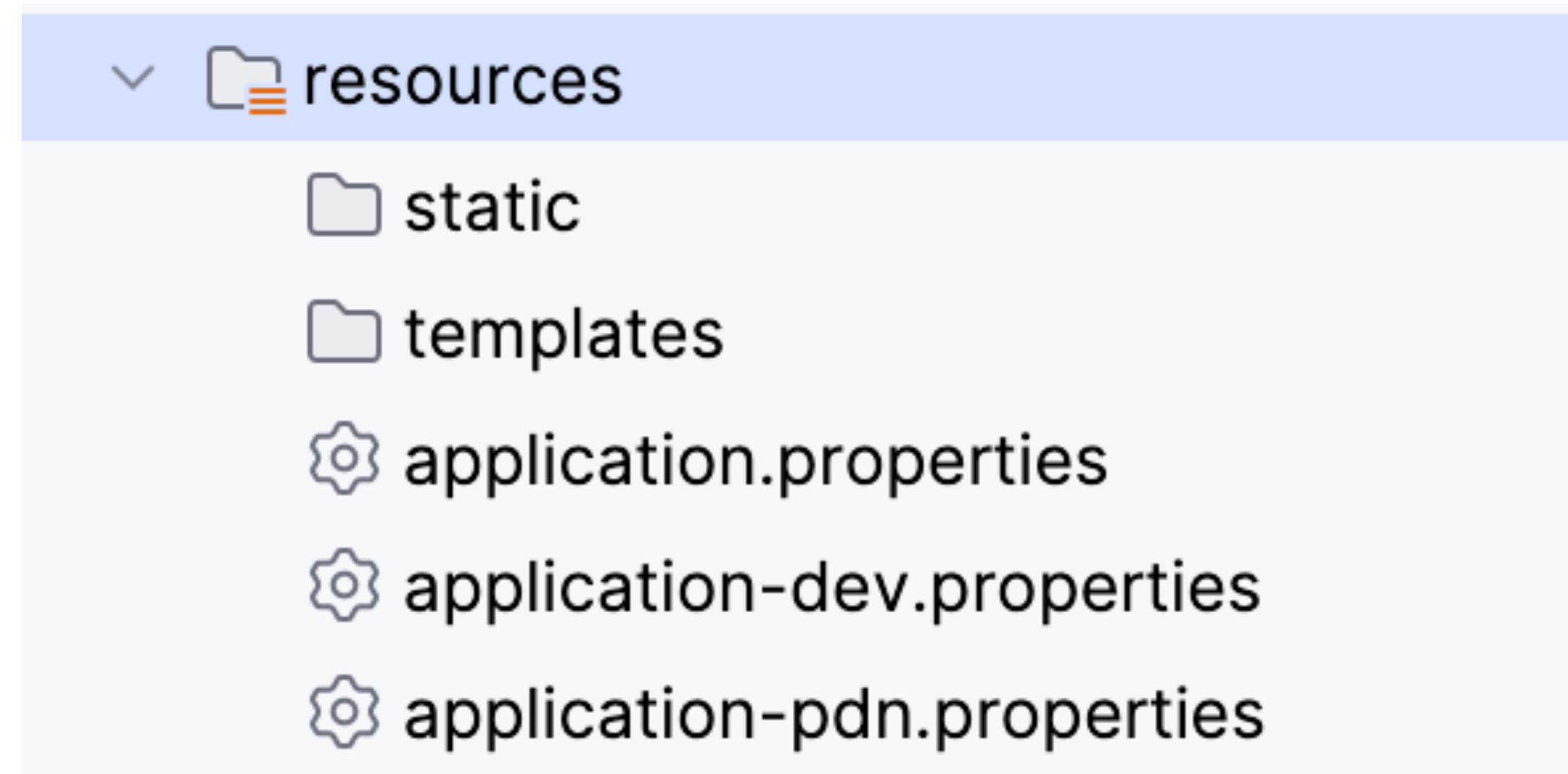
- Probamos las nuevas rutas



Ruta de la aplicación

Spring Boot

- Puedo configurar varios ambientes



Los nombres de los archivos son importantes, -dev y -pdn son los nombres con los que Spring identifica los ambientes

Spring Boot

- Puedo configurar varios ambientes



Spring Boot

- Puedo configurar varios ambientes

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

application.properties



Señalar el ambiente de trabajo

```
application-dev.properties
1 server.port=8080
```

application-dev.properties

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

application-pdn.properties

Spring Boot

- Puedo configurar varios ambientes

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

application.properties

```
application-dev.properties
1 server.port=8080
```

application-dev.properties

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

application-pdn.properties



Configuración para el ambiente
de desarrollo (dev)

Spring Boot

- Puedo configurar varios ambientes

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

application.properties

```
application-dev.properties
1 server.port=8080
```

application-dev.properties

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

application-pdn.properties



Configuración para el ambiente de producción (pdn)

Spring Boot

Listo, ya sabes configurar un proyecto de Spring



Spring Boot

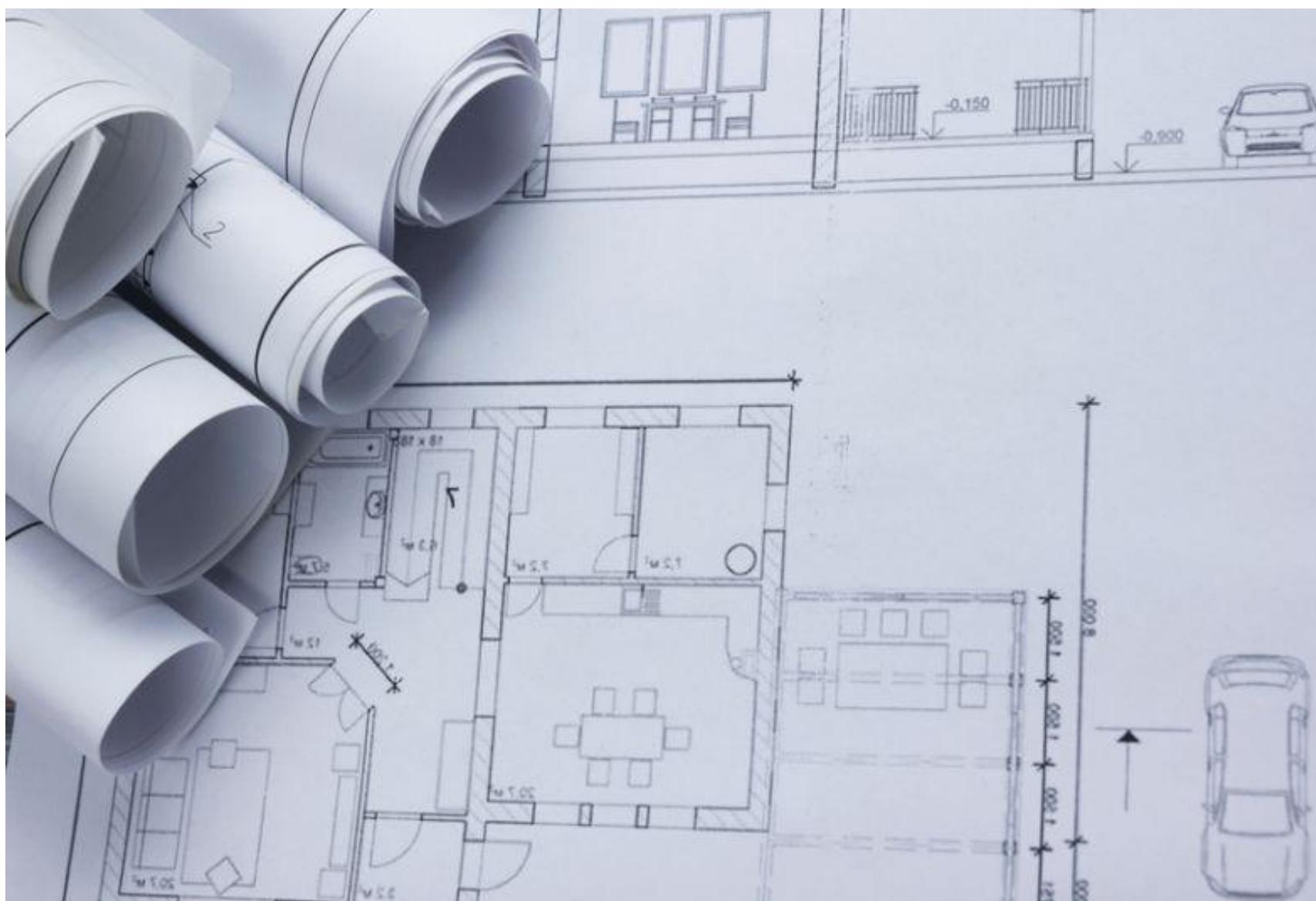
- Estructura del proyecto



Spring Boot

- **Arquitectura para la API**

Arquitectura por capas orientada al dominio



Spring Boot

- **Primera capa**

Dominio



Domain Layer

Spring Boot

- **Primera capa**

Compuesta por:

A) Objetos del dominio: objetos que forman parte del contexto de nuestra aplicación

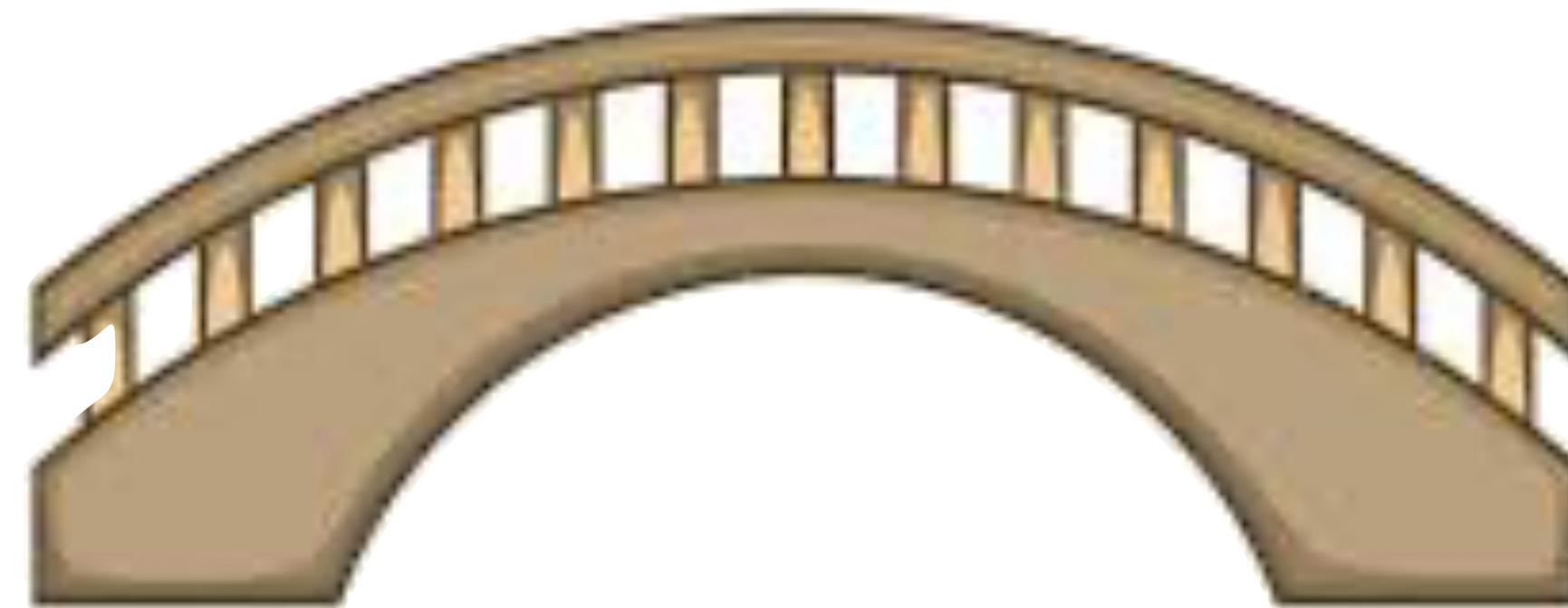


Spring Boot

- **Primera capa**

Compuesta por:

B) Servicios: puente entre los controladores y los repositorios



Spring Boot

- **Primera capa**

Compuesta por:

C) Especificación de los repositorios: interfaces que definen los repositorios



Spring Boot

- Segunda capa

WEB



Web Layer

Spring Boot

- **Segunda capa**

La capa WEB contiene los controladores tipo REST



Spring Boot

- **Tercera capa**

Capa de persistencia



Persistence Layer

Spring Boot

- **Tercera capa**

Capa de persistencia, la que tiene el trabajo de interactuar con la base de datos. Utilizará como guía las interfaces del domino y creará las entidades (clases que representan los registros de las base de datos).



Spring Boot

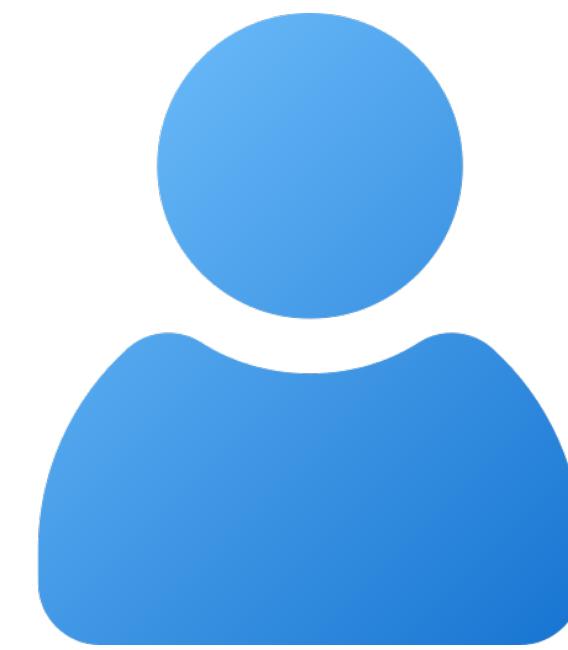
- Flujo



El usuario ejecuta un controlador (REST) de la capa WEB

Spring Boot

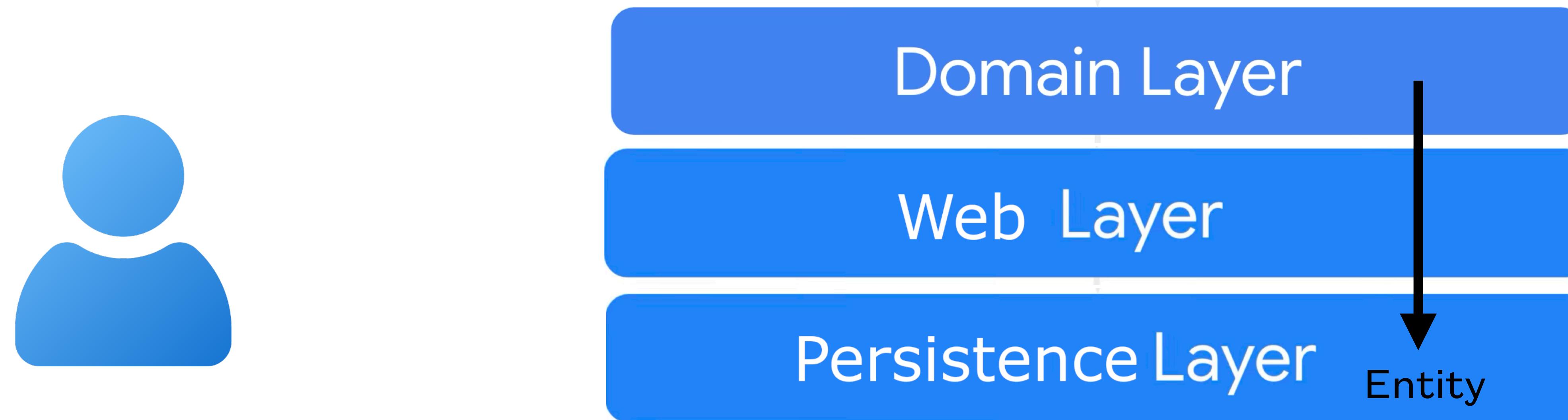
- Flujo



El controlador (REST) ejecuta un servicio de la capa del dominio

Spring Boot

- Flujo



El servicio interactúa con una o más entidades de la capa de persistencia

Spring Boot

- Flujo



El servicio da información a la capa WEB

Spring Boot

- Flujo



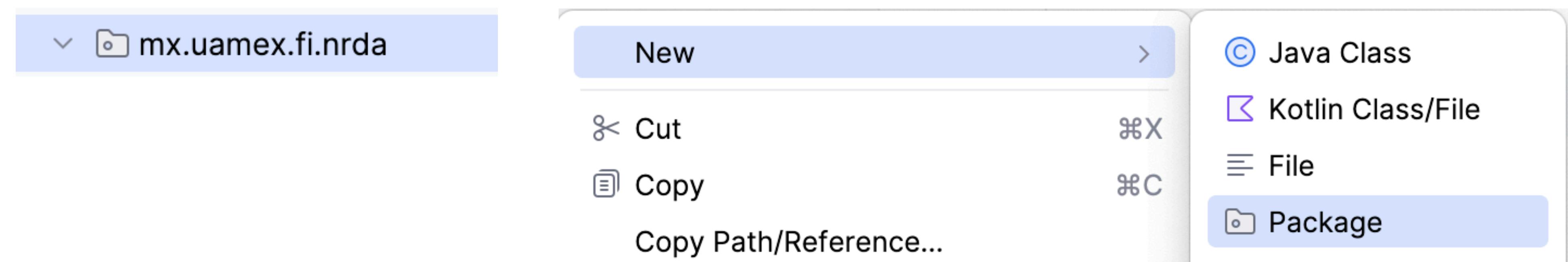
La capa WEB procesa la información y responde al usuario

Spring Boot

- **Creación de la infraestructura**

Creamos el paquete domain

Click derecho sobre el paquete de nuestro proyecto



Spring Boot

- **Creación de la infraestructura**

Creamos el paquete domain

Nombramos

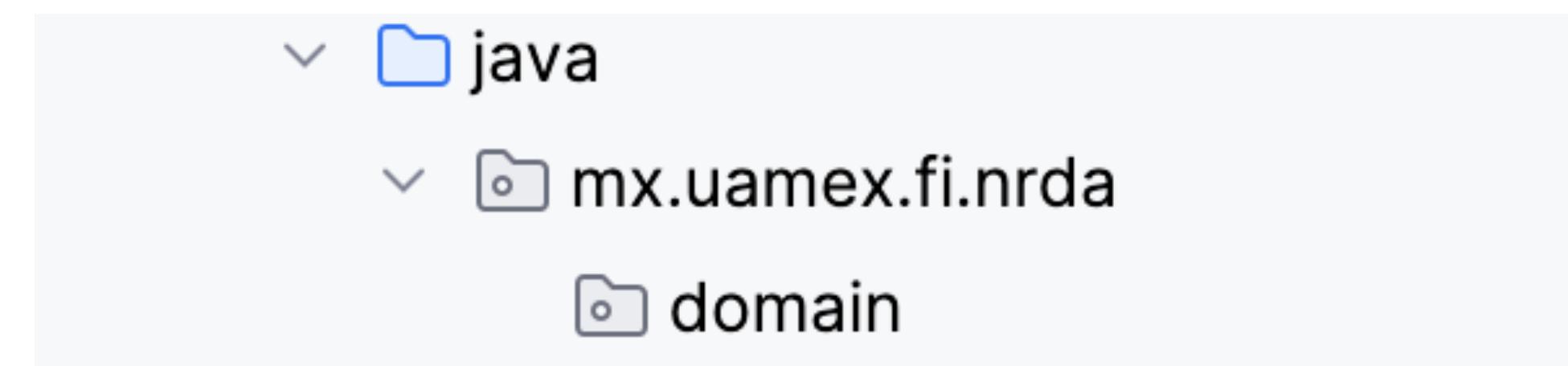
New Package
mx.uamex.fi.nrda.domain|

Spring Boot

- **Creación de la infraestructura**

Creamos el paquete domain

Verificamos



Spring Boot

- **Creación de la infraestructura**

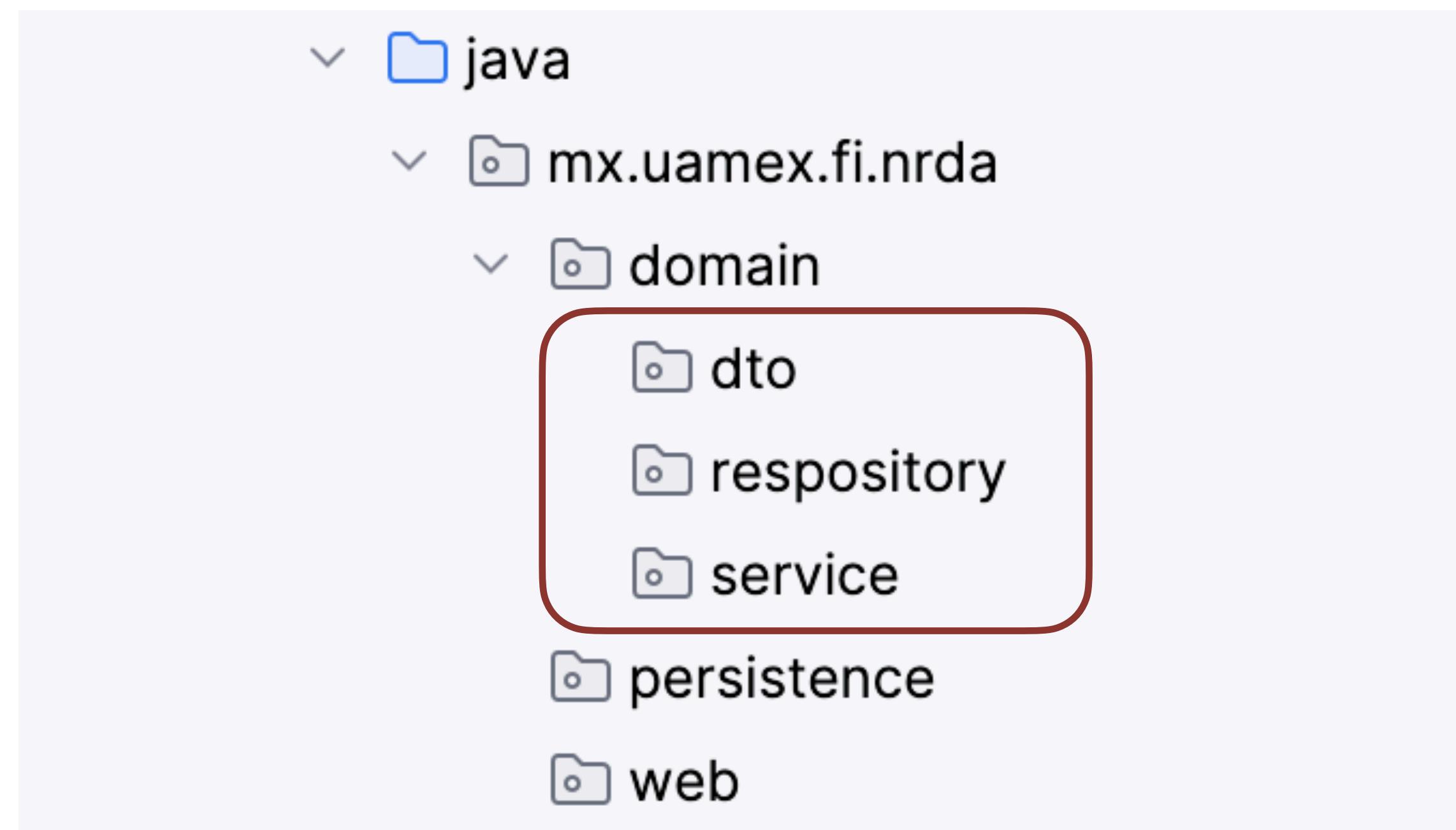
De forma análoga creamos los paquetes web y persistence



Spring Boot

- **Creación de la infraestructura**

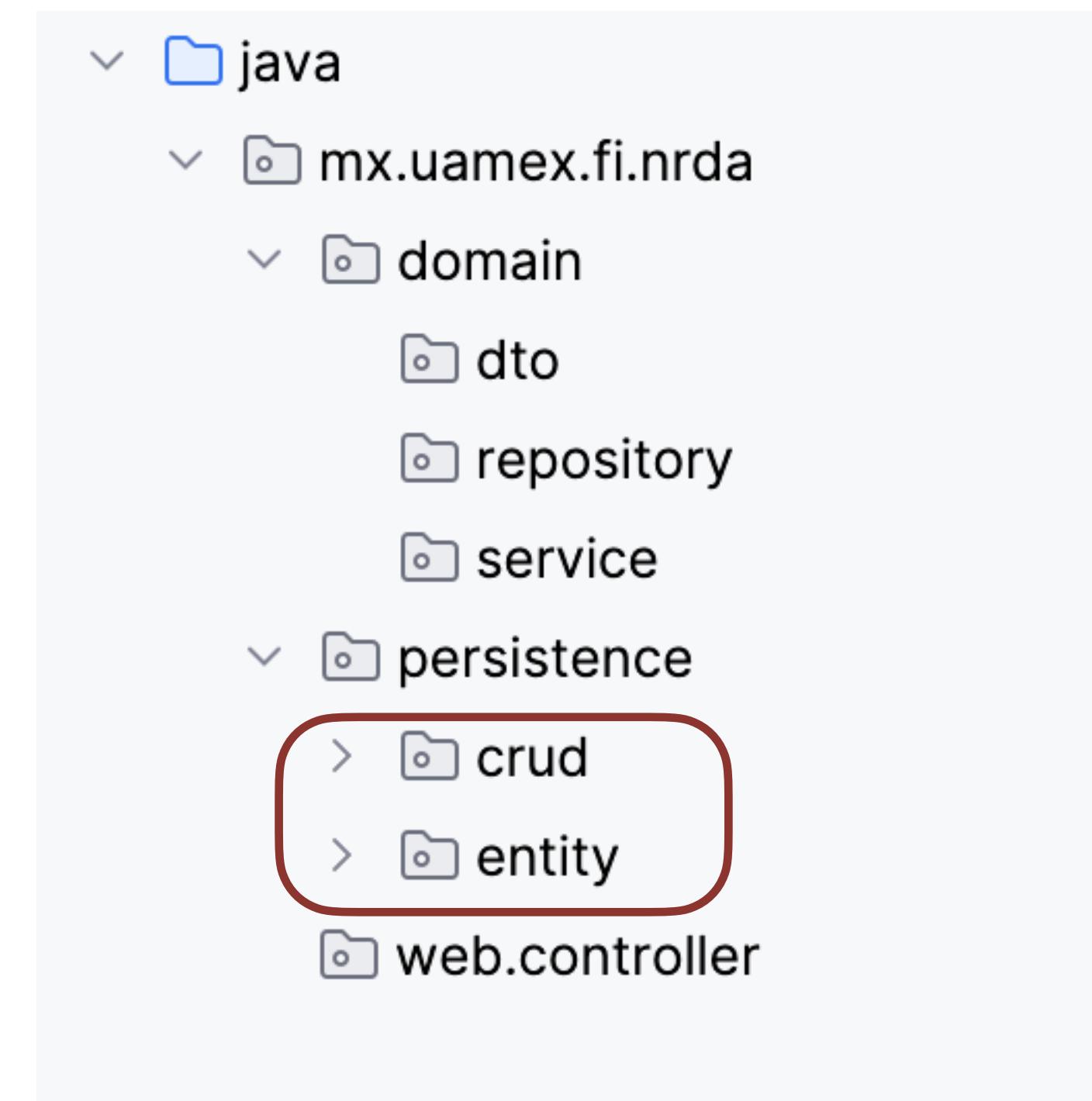
Sub-paquetes de domain



Spring Boot

- **Creación de la infraestructura**

Sub-paquetes de persistence

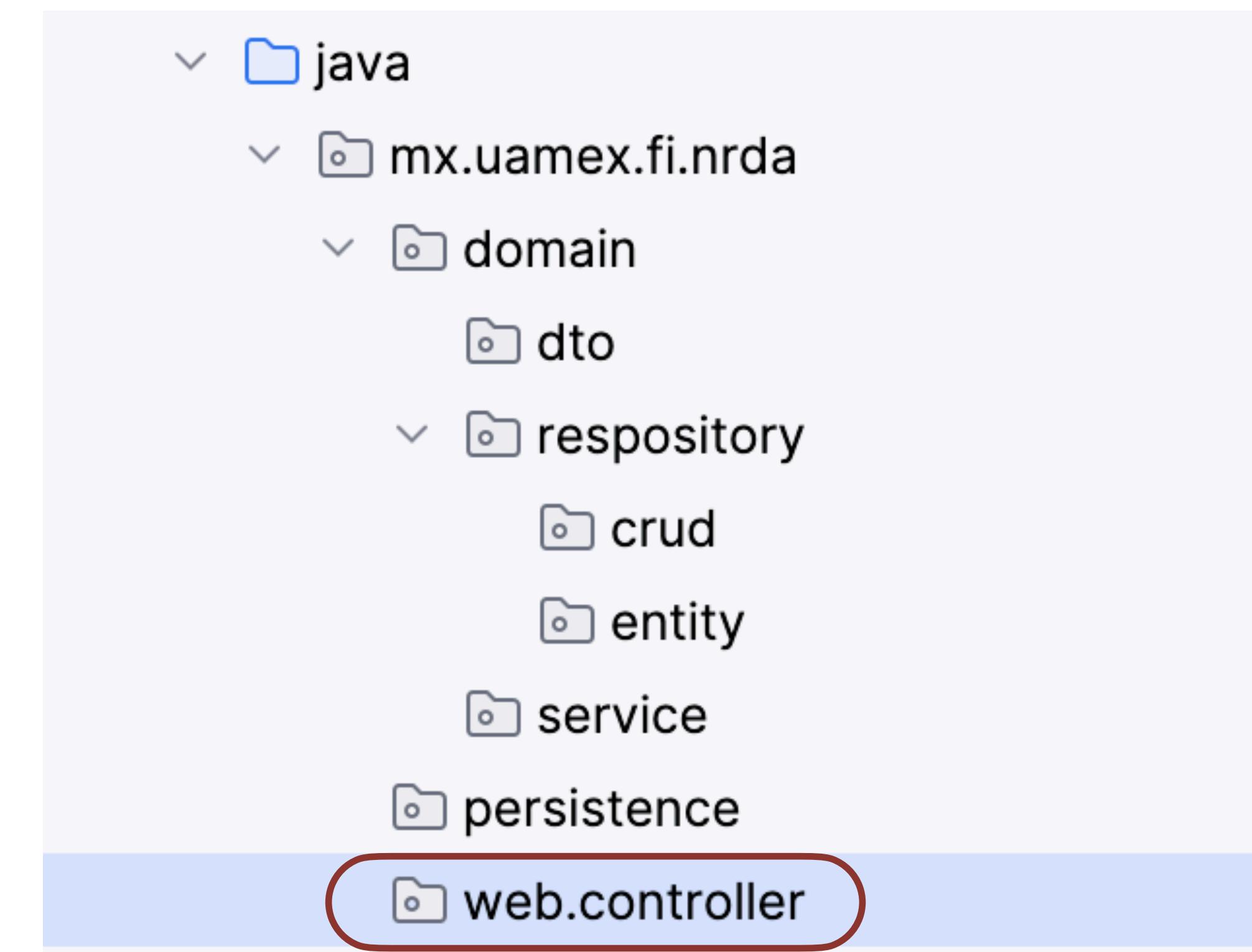


Spring Boot

- **Creación de la infraestructura**

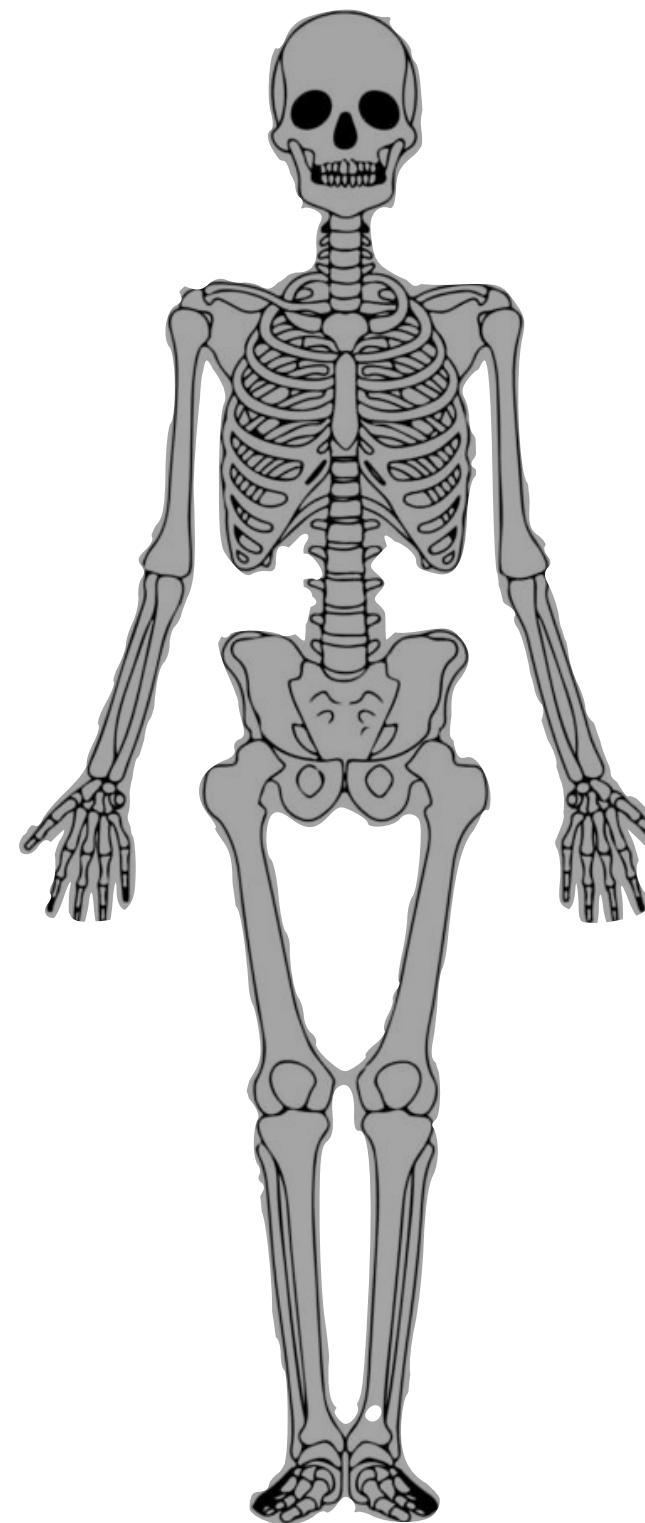
Sub-paquetes de web

Nota: controller está dentro de web, pero así lo presenta el IDE



Spring Boot

- Estructura lista



¡Pongamos carnita al
proyecto!

Spring Boot

- Pero antes de empezar con el código, estudiaremos un par de herramientas:
 - JPA
 - Spring Data



Spring Boot

- **JPA**

Java Persistence API - Especificación para mapear datos entre objetos y bases de datos relacionales

OMR - Mapper

Spring Boot

- **JPA**

JPA es la especificación, herramientas populares que la implementan son:



HIBERNATE

TopLink

eclipselink



ObjectDB

Spring Boot

- **JPA**

Estas herramientas escriben el código Java para vincular los registros de las tablas con objetos Java



Siéntate papito, en lo que nosotros trabajamos con la base de datos

Spring Boot

- **JPA**

Como la implementación de JPA hace el código, JPA utiliza anotaciones para especificar la forma de mapear entre las tablas y los objetos

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

Spring Boot

- **JPA**

La anotación `@Entity` especifica que la clase representa una tabla de la base de datos



Spring Boot

- **JPA**

La anotación `@Table` especifica la tabla de la base de datos con la que se mapearán los objetos de la clase

	CustomerId	FirstName	LastName	DateCreated	City
1	1	Homer	Simpson	13/06/2014 3:33:37 PM	Bartlet
2	2	Peter	Griffin	13/06/2014 9:09:56 PM	Bartlet
3	3	Stewie	Griffin	13/06/2014 9:16:07 PM	Bartlet
4	4	Brian	Griffin	13/06/2014 9:16:36 PM	Bartlet
5	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	Bartlet
6	6	Philip	Fry	13/06/2014 9:17:02 PM	Bartlet
7	7	Amy	Wong	13/06/2014 9:22:05 PM	Bartlet
8	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	Bartlet
9	9	Marge	Simpson	13/06/2014 9:22:37 PM	Bartlet
10	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	Bartlet
11	11	Turanga	Leela	13/06/2014 9:23:37 PM	Bartlet
*		(New)		15/06/2014 9:00:01 PM	

Spring Boot

- **JPA**

La anotación `@Column` especifica la columna de la tabla de la base de datos con la que se mapeará el atributo de la clase

Customers					
	CustomerId	FirstName	LastName	DateCreated	ClientType
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)				15/06/2014 9:00:01 PM

Nota: Esta anotación es opcional, sólo se usa cuando el nombre del atributo difiere del nombre de la columna en la tabla

Spring Boot

- **JPA**

La anotación `@Id` especifica que ese atributo se mapea con la llave primaria de la tabla



Spring Boot

- **JPA**

La anotación `@EmbeddedId` para mapear llaves primarias compuestas



Spring Boot

- **JPA**

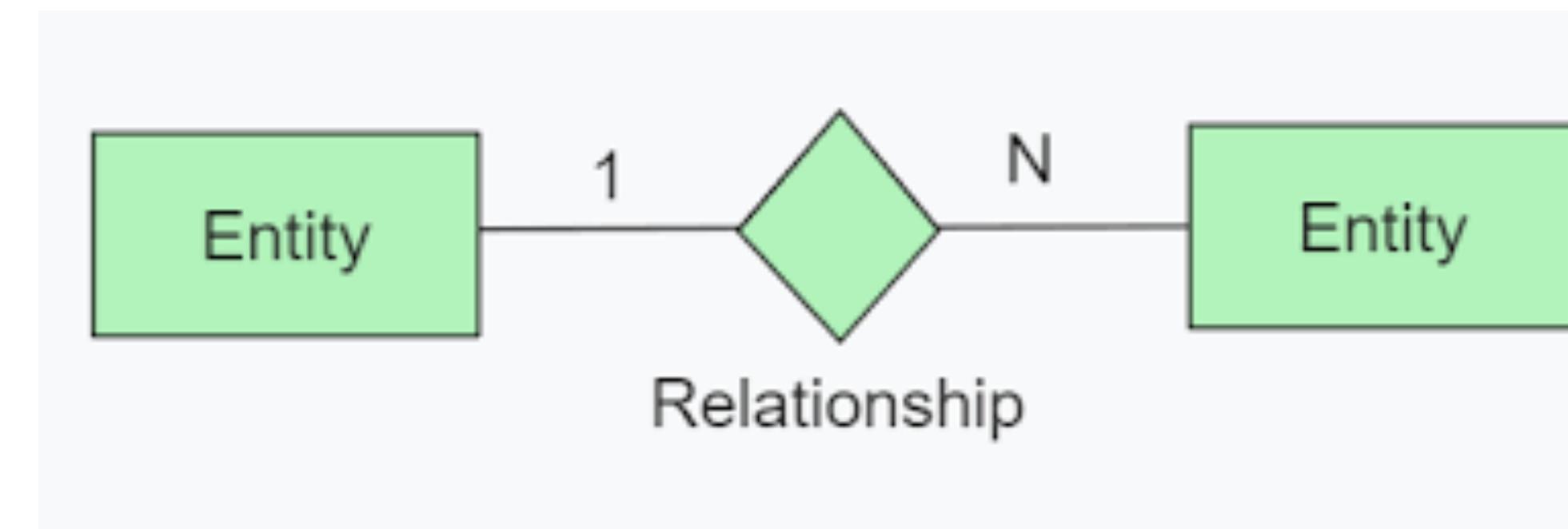
La anotación `@GeneratedValue` llaves primarias autogeneradas



Spring Boot

- **JPA**

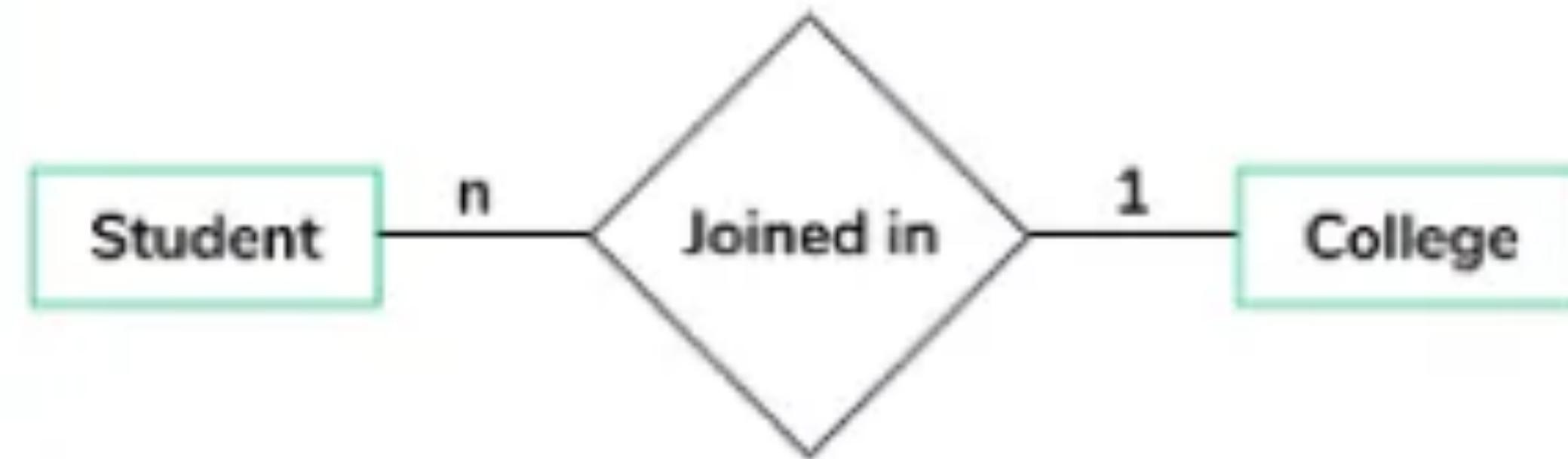
La anotación `@OneToOne` para marcar relaciones “*uno a uno*”



Spring Boot

- **JPA**

La anotación `@ManyToOne` para marcar relaciones “*muchos a uno*”



Spring Boot

- **Spring Data**

Herramienta que usa JPA para explotarla con las funcionalidades de Spring framework tiene implementaciones para bases de datos relacionales, orientadas a documentos, archivos, etc



Spring Boot

- **Spring Data**

Su tarea principal es optimizar tareas repetitivas, utilizando repositorios. Adicionalmente posee auditorías para verificar si las operaciones se ejecutaron en la base de datos.

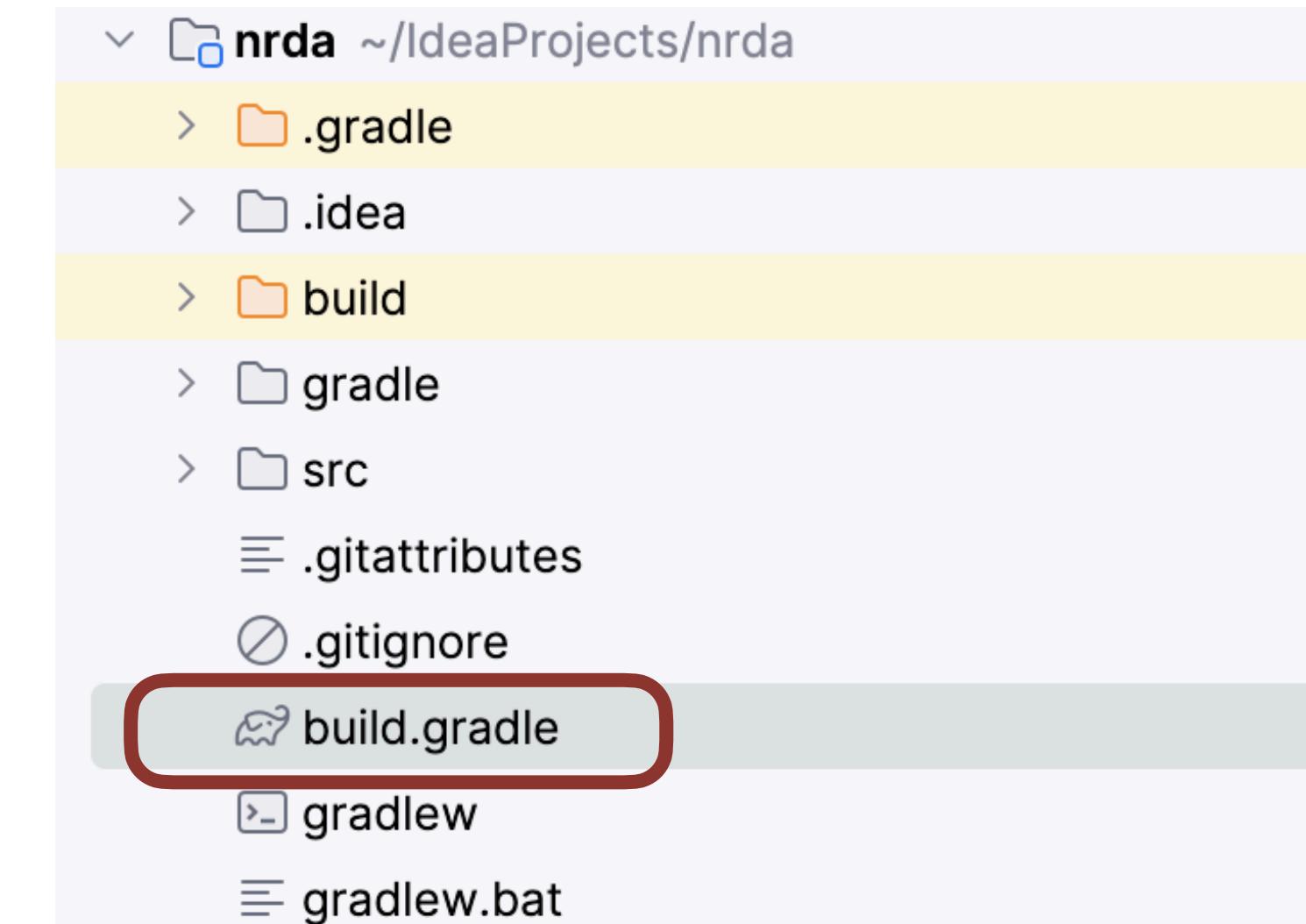


Spring Boot

- **Spring Data**

Necesitamos incluir la versión adecuada de Spring Data en nuestro proyecto

Esto se especifica en el archivo:



Spring Boot

- **Spring Data**

Eso se especifica en la sección dependencies del archivo build.gradle

```
build.gradle (nrda) ...
19
20 > dependencies {
21     implementation 'org.springframework.boot:spring-boot-starter-web'
22     testImplementation 'org.springframework.boot:spring-boot-starter-test'
23     testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
24 }
25
26 > tasks.named('test') { Task it ->
27     useJUnitPlatform()
28 }
29
```

Spring Boot

- **Spring Data**

Agregar esta línea

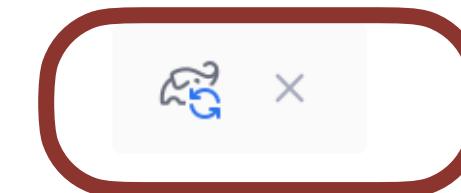
```
mavenCentral()  
}  
  
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
```

Spring Boot

- **Spring Data**

Dar click en este botón para cargar cambios en Gradle

```
mavenCentral()  
}  
  
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'  
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
```



Spring Boot

- **Spring Data**

Crear una fuente de datos (DataSource)

Paso (1): Agregar dependencias

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

Spring Boot

- **Spring Data**

Crear una fuente de datos (DataSource)

Paso (1): Agregar dependencias, en el archivo build.gradle

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



Agregar
dependencia

Spring Boot

- **Spring Data**

Crear una fuente de datos (DataSource)

Paso (1): Agregar dependencias, recargar Gradle

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



Actualizar
Gradle

Spring Boot

- **Spring Data**

Crear una fuente de datos (DataSource)

Paso (2): Configurar la fuente de datos

```
application-dev.properties
1 server.port=8080
2 # Configuracion de la fuente de datos
3 spring.datasource.url=jdbc:postgresql://localhost:5432/nrda
4 spring.datasource.username=postgres
5 spring.datasource.password=PennyLane
```

Datos de conexión

Spring Boot

- **Spring Data**

Crear una fuente de datos (DataSource)



Nota: Ejecuta el servidor para verificar que todo va bien

Spring Boot

- **Spring Data**

La base debe de existir y el servidor estar en ejecución



Spring Boot

- **Spring Data**

```
2025-11-25T18:16:27.782-06:00 INFO 70147 --- [nrda] [           main]
org.hibernate.orm.connections.pooling : HHH10001005: Database info:
```

Database JDBC URL [jdbc:postgresql://localhost:5432/nrda]

Database driver: PostgreSQL JDBC Driver

Database dialect: PostgreSQLDialect

Database version: 15.7

Default catalog/schema: nrda/public

Spring Boot

- **Spring Data**

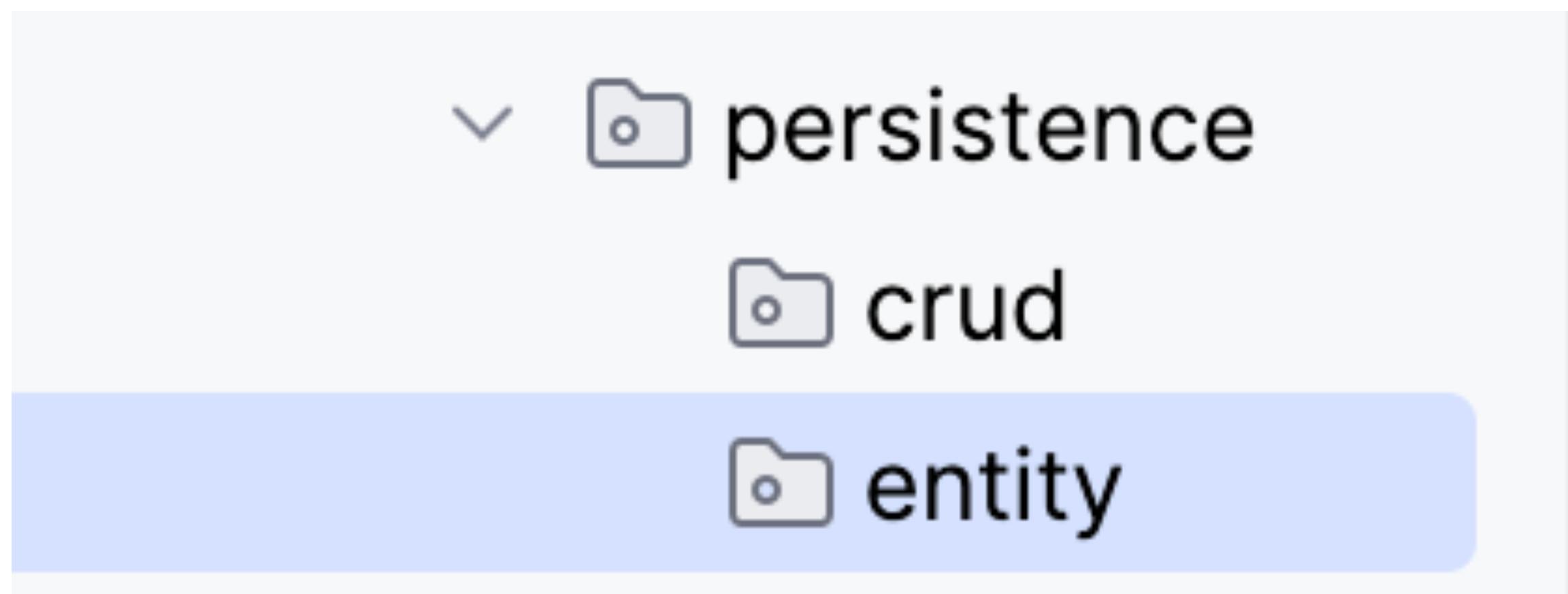
Creación de Entidades



Spring Boot

- **Usuario**

En el paquete entity crear la clase Usuario



Spring Boot

- Usuario

```
package mx.uamex.fi.nrda.domain.repository.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "usuarios")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String login;
    @Column(name = "password")
    private String contrasena;
    private String email;
```

@Entity



Marca a los objetos como entidades

Spring Boot

- Usuario

```
package mx.uamex.fi.nrda.domain.repository.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "usuarios")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String login;
    @Column(name = "password")
    private String contrasena;
    private String email;
```

Como la clase y la tabla
difieren en nombre, señalamos
el nombre de la tabla

Spring Boot

- Usuario

```
package mx.uamex.fi.nrda.domain.repository.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String login;
    @Column(name = "password")
    private String contrasena;
    private String email;
```

Este atributo carga con la llave primaria

Spring Boot

- Usuario

El id es manejado de forma automática por la base

```
package mx.uamex.fi.nrda.domain.repository.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "usuarios")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String login;
    @Column(name = "password")
    private String contrasena;
    private String email;
```



Spring Boot

- Usuario

No es necesario
especificar la columna
porque tiene el mismo
nombre que el atributo

```
package mx.uamex.fi.nrda.domain.repository.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "usuarios")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String login;
    @Column(name = "password")
    private String contrasena;
    private String email;
```

Spring Boot

- Usuario

No es necesario especificar la columna porque tiene el mismo nombre que el atributo

```
package mx.uamex.fi.nrda.domain.repository.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "usuarios")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String login; ←
    @Column(name = "password")
    private String contrasena;
    private String email;
```

Spring Boot

- Usuario

```
package mx.uamex.fi.nrda.domain.repository.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "usuarios")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String login;
    @Column(name = "password") →
    private String contrasena;
    private String email;
```

Es necesario nombrar la columna a la que se asocia el atributo porque NO tienen el mismo nombre

Spring Boot

- Usuario

```
package mx.uamex.fi.nrda.domain.repository.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "usuarios")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String login;
    @Column(name = "password")
    private String contrasena;
    private String email;
```

No es necesario especificar la columna porque tiene el mismo nombre que el atributo

Spring Boot

- Usuario

- Agregar métodos Get y Set

Spring Boot

- **Spring Data**

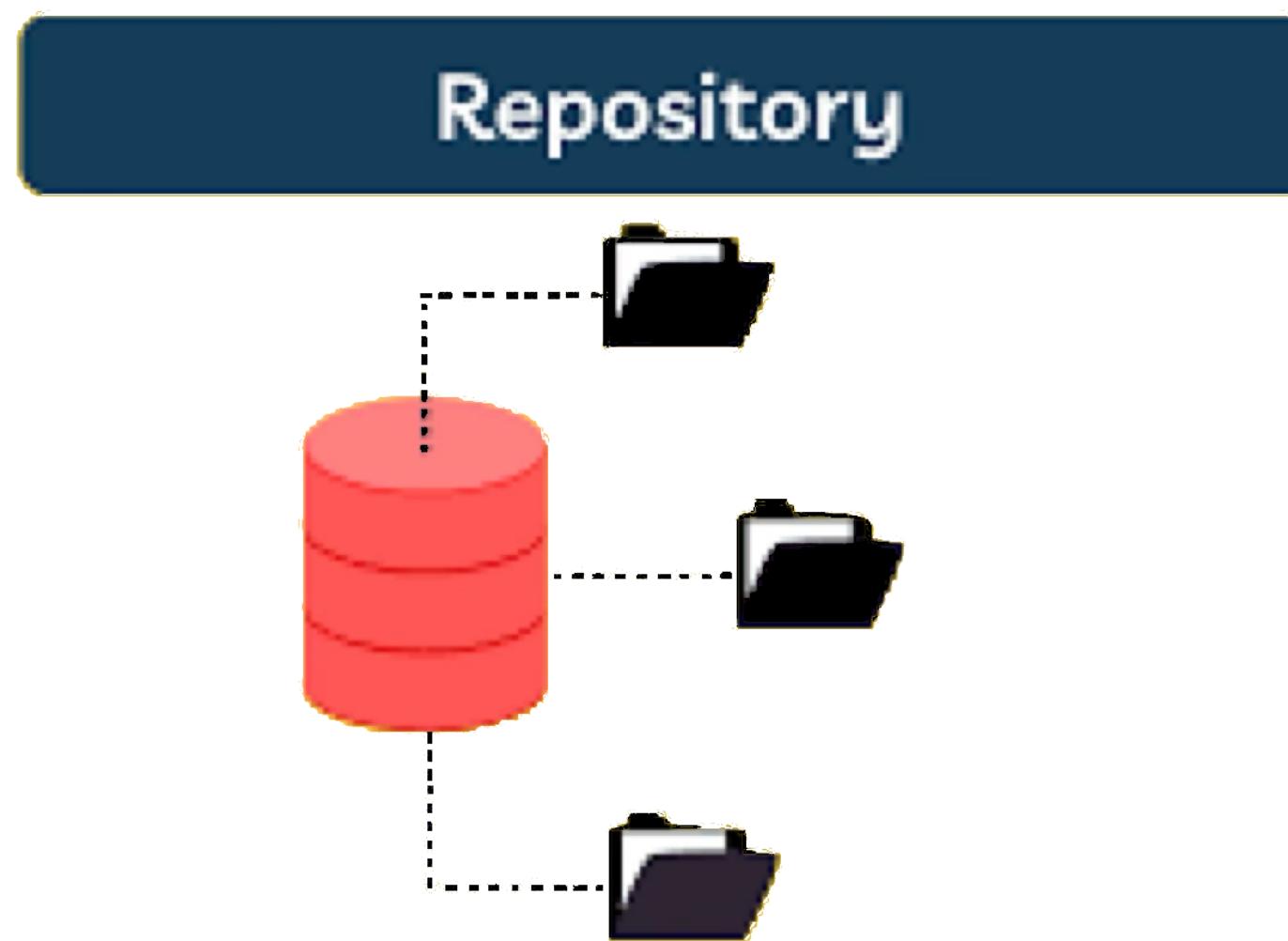
Como sólo tenemos una entidad podemos decorar la creación de entidades



Spring Boot

- **Spring Data**

Creación de Repositorios



Spring Boot

• Spring Data

Los repositorios de Spring Data crean de forma automática métodos de explotación de la base de datos. Este subproyecto de Spring cuenta con tres tipos de repositorios:

- CrudRepository
- PagingAndSortingRepository
- JPA Repository

Spring Boot

- **Spring Data**

CrudRepository

Hace las operaciones CRUD

Spring Boot

- **Spring Data**

PagingAndSortingRepository

Hace las operaciones CRUD y además tareas de paginación y ordenamiento

Spring Boot

- **Spring Data**

- JPARepository

- Hace todas las operaciones de los otros dos y tareas específicas de JPA (Flush por ejemplo).

Spring Boot

- **Spring Data**

CrudRepository

En el proyecto utilizaremos un CrudRepository. Primer paso, en el paquete CRUD crear una interfaz que defina los métodos a utilizar.



Spring Boot

- **Spring Data**

```
package mx.uamex.fi.nrda.domain.repository.crud;  
  
import mx.uamex.fi.nrda.domain.repository.entity.Usuario;  
import org.springframework.data.repository.CrudRepository;  
  
public interface UsuarioCrudRepository extends CrudRepository<Usuario, Integer> {  
}
```

Spring Boot

- **Spring Data**

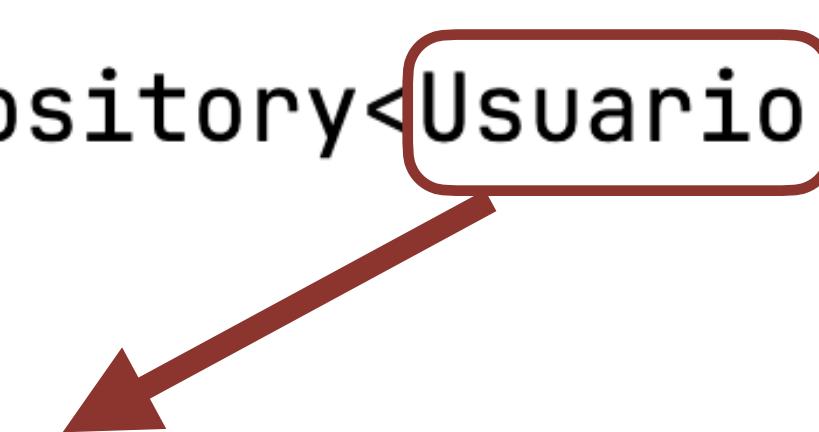
```
package mx.uamex.fi.nrda.domain.repository.crud;  
  
import mx.uamex.fi.nrda.domain.repository.entity.Usuario;  
import org.springframework.data.repository.CrudRepository;  
  
public interface UsuarioCrudRepository extends CrudRepository<Usuario, Integer> {  
}
```

Spring Boot

- **Spring Data**

```
package mx.uamex.fi.nrda.domain.repository.crud;  
  
import mx.uamex.fi.nrda.domain.repository.entity.Usuario;  
import org.springframework.data.repository.CrudRepository;  
  
public interface UsuarioCrudRepository extends CrudRepository<Usuario, Integer> {  
}
```

Entidad para la que deseamos el Repositorio

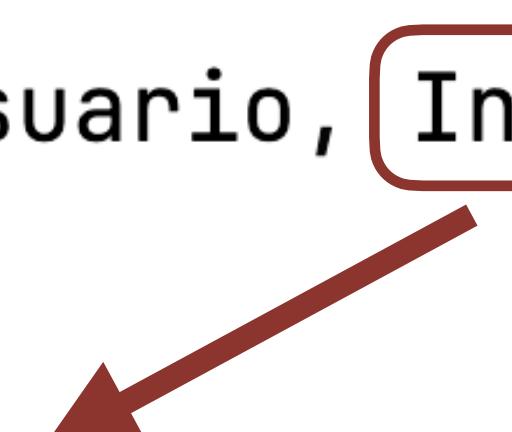


Spring Boot

- **Spring Data**

```
package mx.uamex.fi.nrda.domain.repository.crud;  
  
import mx.uamex.fi.nrda.domain.repository.entity.Usuario;  
import org.springframework.data.repository.CrudRepository;  
  
public interface UsuarioCrudRepository extends CrudRepository<Usuario, Integer> {  
}
```

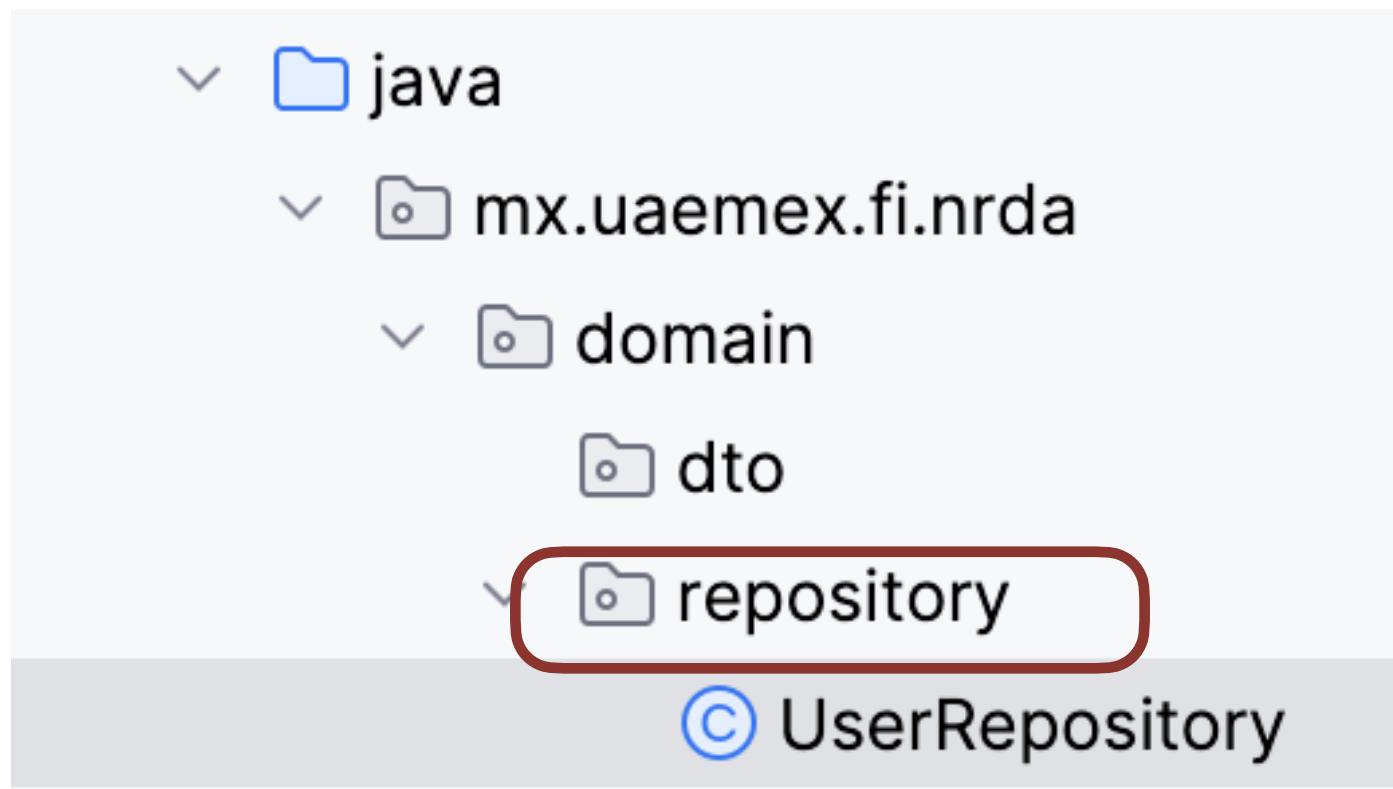
Tipo del atributo Id (primary key)



Spring Boot

• Spring Data

Listo, sabiendo esto Spring hace el código de acceso a base de datos por ti, sólo necesitas una clase que utilice el tipo de datos que creará Spring.



```
import mx.uaemex.fi.nrda.persistence.crud.UsuarioCrudRepository;  
import mx.uaemex.fi.nrda.persistence.entity.Usuario;  
  
import java.util.List;  
  
public class UserRepository { no usages  
    private UsuarioCrudRepository repository; 1 usage  
  
    public List<Usuario> getAll() { no usages  
        return (List<Usuario>) repository.findAll();  
    }  
}
```

Spring Boot

• Spring Data

El CrudRepository tiene los siguientes métodos:

- save
- saveAll
- findById
- ExistsById
- findAll
- findAllById
- count
- delete
- deleteById
- deleteAllById
- deleteAll
- deleteAll

Spring Boot

• Spring Data

Esto es lo que en automático puede hacer, pero ¿cómo hago métodos de base de datos, fuera de estos?

- save
- saveAll
- findById
- ExistsById
- findAll
- findAllById
- count
- delete
- deleteById
- deleteAllById
- deleteAll
- deleteAll

Spring Boot

- Spring Data



Spring Boot

- **Spring Data**

```
package mx.uamex.fi.nrda.domain.respository.crud;  
  
import mx.uamex.fi.nrda.domain.respository.entity.Usuario;  
import org.springframework.data.repository.CrudRepository;  
  
public interface UsuarioCrudRepository extends CrudRepository<Usuario, Integer> {  
    Usuario findByLogin(String login);  
}  
→ Incluimos el método necesario
```

Spring Boot

- **Spring Data**

QueryMethods

Spring usa una nomenclatura de nombres para poder especificar lo que deseamos, al comenzar con `find` Spring sabe que se trata de una consulta, el `By` debe ir seguido de uno de los atributos de la entidad, con esto sabe que ese atributo será usando en el `where` de la consulta.

De esta manera el método `findByLogin` consulta el usuario cuyo login sea igual al recibido como parámetro.

Spring Boot

- **Spring Data**

QueryMethods

Si no se desea seguir la nomenclatura de nombres o si la consulta es tan específica que no se pueda construir a través de la nomenclatura Spring permite escribir la consulta en SQL mediante una anotación.

El método de buscar a un Usuario por medio de su login pudo haberse escrito como:

```
@Query(value="select Usuario from usuarios where login = ?", nativeQuery=true)
Usuario getByLogin(String login);
```

Spring Boot

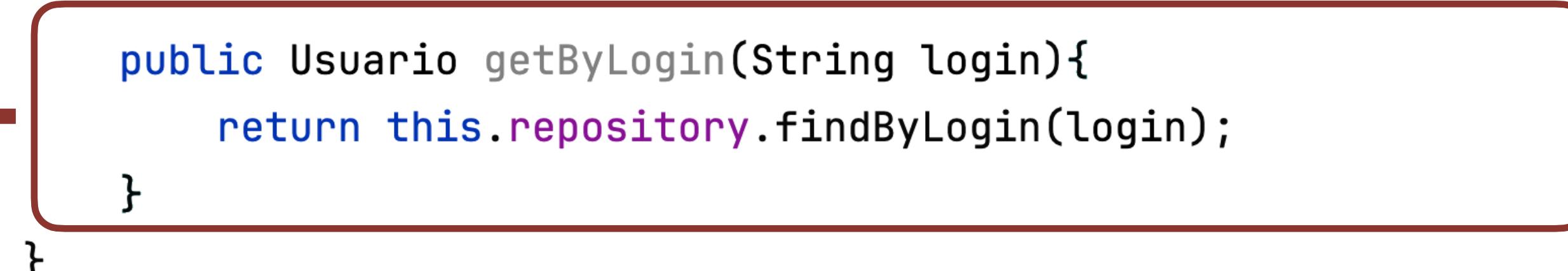
- Spring Data

QueryMethods

Como cambio la interfaz la clase debe cambiar para poder aprovechar el nuevo método

```
public class UsuarioRepository {  
    private UsuarioCrudRepository repository;  
  
    public List<Usuario> getAll(){  
        return (List<Usuario>) repository.findAll();  
    }  
  
    public Usuario getByLogin(String login){  
        return this.repository.findByLogin(login);  
    }  
}
```

Nuevo Método



A red rounded rectangle callout box surrounds the implementation of the `getByLogin` method. An arrow points from the text "Nuevo Método" to the top-left corner of this box.

```
public Usuario getByLogin(String login){  
    return this.repository.findByLogin(login);  
}
```

Spring Boot

- Spring Data

QueryMethods

Terminemos el repositorio

```
@Repository  
public class UsuarioRepository {  
    private UsuarioCrudRepository repository;  
  
    public List<Usuario> getAll(){  
        return (List<Usuario>) repository.findAll();  
    }  
    public Usuario getByLogin(String login){  
        return this.repository.findByLogin(login);  
    }  
    public Usuario save(Usuario usuario){  
        return this.repository.save(usuario);  
    }  
    public void delete(Usuario usuario){  
        this.repository.delete(usuario);  
    }  
    public void deleteById(Integer id){  
        this.repository.deleteById(id);  
    }  
    public Usuario getById(Integer id){  
        return this.repository.findById(id).get();  
    }  
}
```

Lo marcamos como Repositorio

Spring Boot

- Spring Data

QueryMethods

Terminemos el repositorio

Exponemos los métodos
del CRUD en la clase
para poder utilizarlos



```
@Repository
public class UsuarioRepository {
    private UsuarioCrudRepository repository;

    public List<Usuario> getAll(){
        return (List<Usuario>) repository.findAll();
    }

    public Usuario getByLogin(String login){
        return this.repository.findByLogin(login);
    }

    public Usuario save(Usuario usuario){
        return this.repository.save(usuario);
    }

    public void delete(Usuario usuario){
        this.repository.delete(usuario);
    }

    public void deleteById(Integer id){
        this.repository.deleteById(id);
    }

    public Usuario getById(Integer id){
        return this.repository.findById(id).get();
    }
}
```