Spring-Boot.pdf

quiico_

Acceso a datos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

Spring Boot

2º DAM Curso 2024/2025

Asignatura: Acceso a datos

- **Profesor:** Manuel Mauri
- **Alumno:** Francisco José Alonso de Caso Ortiz Nº2

Índice

¿Qué es?

Spring Boot es un framework de Java basado en Spring que facilita la creación de aplicaciones web y microservicios al proporcionar configuración automática, un servidor embebido y una estructura modular.
Sirve también para crear Api's.

Estructura básica de un proyecto Spring Boot

Un proyecto básico tendrá una estructura similar a esta:

```
src
└── main
    └── java
        └── com
            └── ejemplo
                └── MiAplicacion.java
    └── resources
        └── application.properties
```

Crear la clase principal de Spring Boot

Dentro de src/main/java/com/ejemplo/MiAplicacion.java, crea la clase principal con la anotación @SpringBootApplication:

```java
package com.ejemplo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MiAplicacion {

    public static void main(String[] args) {
        SpringApplication.run(MiAplicacion.class, args);
    }
}
```

Configurar y personalizar la aplicación

En el archivo src/main/resources/application.properties puedes configurar diferentes aspectos de la aplicación, como el puerto del servidor:

```
server.port=8080
```

Añadir funcionalidades

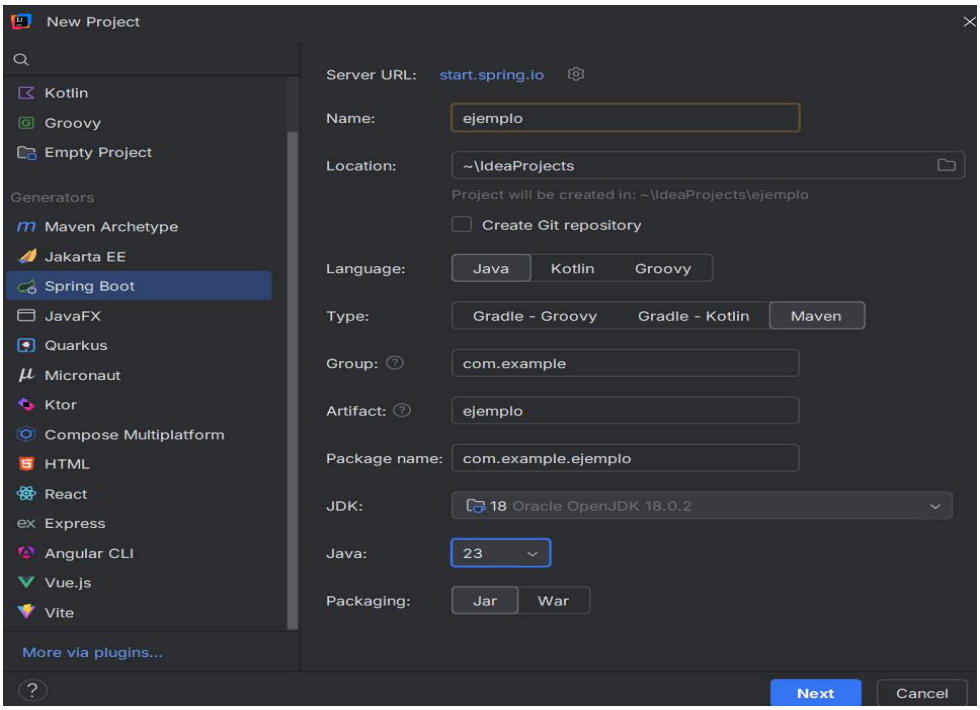
Para añadir funcionalidades, puedes crear controladores. Por ejemplo, un controlador simple que devuelva un saludo:

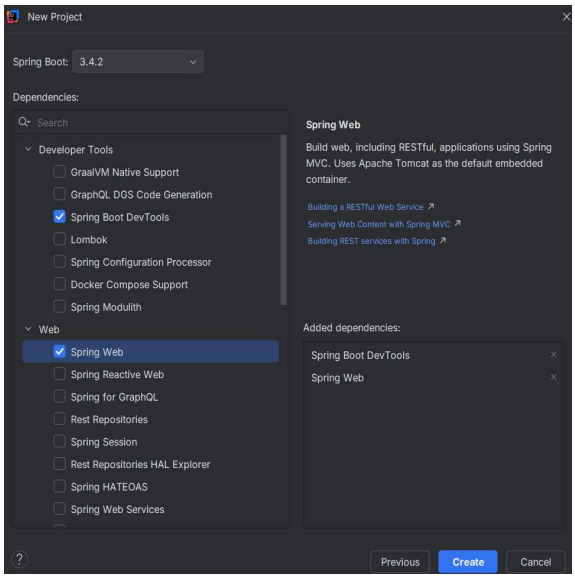
```java
package com.ejemplo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SaludoController {

    @GetMapping("/saludo")
    public String saludo() {
        return "¡Hola, mundo!";
    }
}
```

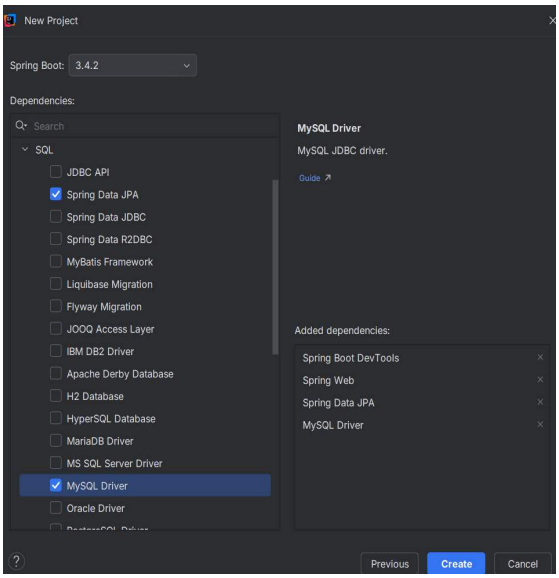
Resumen

- Spring Boot facilita la creación de aplicaciones Java.
- Utiliza Spring Initializr para generar un proyecto básico.
- La clase principal debe estar anotada con @SpringBootApplication.
- Los controladores se definen con @RestController para manejar las rutas HTTP.

Crear un nuevo proyecto Spring Boot







Create

Google Gemini: Plan Pro a 0€ durante 1 año.

Tu ventaja por ser estudiante

Entra en wlh.es/estudiacongemini

Consigue la oferta

Domina cualquier tema con el Aprendizaje Guiado de Google Gemini.

Oferta válida hasta el 9 de diciembre de 2025

Después, 21,99€/mes. 18+. Los resultados/la compatibilidad del dispositivo varían. Comprobar la exactitud de las respuestas. Se aplican restricciones de almacenamiento y de usuario. Se requiere una cuenta de Google. Consulta los términos y condiciones.

Application.properties

```
spring.application.name=Spring1
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/Spring1
spring.datasource.username=root
spring.datasource.password=rootpassword  Typo: In word 'rootpassword'.
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#spring.jpa.show-sql: true
```

Record class

En Java, una Record class es un tipo de clase introducido en Java 14 (como preview) y estabilizado en Java 16. Se usa para representar datos de manera inmutable, reduciendo el código repetitivo asociado a los POJOs.

```
public record Persona(String nombre, int edad) {}

}
```

Esto automáticamente genera:

- Constructor
- Métodos toString(), equals(), hashCode()
- Accesores (nombre() y edad())

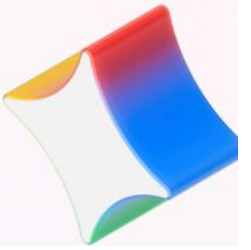
Es ideal para estructuras de datos simples y modelos inmutables.

¿Qué es un POJO?

POJO (Plain Old Java Object) es un objeto Java simple que no depende de ninguna biblioteca o framework específico. Solo contiene atributos y métodos getter/setter, sin lógica compleja. Se usa para representar datos de forma sencilla.

GreetingController

Un Greeting Controller en Spring Boot es un controlador que maneja solicitudes HTTP y devuelve una respuesta, generalmente un mensaje de saludo. Se implementa usando la anotación @RestController y define endpoints con @GetMapping.




```java
package com.ejemplo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class GreetingController {

    @GetMapping("/greeting")
    public String greeting(@RequestParam(value = "name", defaultValue =
"Mundo") String name) {
        return "¡Hola, " + name + "!";
    }
}

        return "¡Hola, mundo!";
    }
}
```

@restController

@RestController es una anotación de Spring Boot que combina @Controller y @ResponseBody, indicando que la clase manejará solicitudes HTTP y devolverá directamente el resultado en formato JSON o XML en lugar de renderizar una vista.

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api")
public class MiControlador {

    @GetMapping("/saludo")
    public String saludo() {
        return "¡Hola, mundo!";
    }
}
```

Explicación:

- @RestController: Indica que la clase es un controlador REST.
- @RequestMapping("/api"): Define la ruta base del controlador.
- @GetMapping("/saludo"): Maneja solicitudes GET en la ruta /api/saludo, devolviendo directamente el texto "¡Hola, mundo!".

@getMapping

@GetMapping es una anotación de Spring Framework que se usa en controladores para manejar solicitudes HTTP GET. Se utiliza para mapear una URL a un método específico dentro de un controlador en aplicaciones Spring Boot.

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HolaMundoController {

    @GetMapping("/saludo")
    public String saludo(@RequestParam(value = "nombre", defaultValue =
"Mundo") String nombre) {
        return "¡Hola, " + nombre + "!";
    }
}
```

Explicación:

- @RestController: Define que esta clase es un controlador REST.
- @GetMapping("/saludo"): Asocia el método a la URL /saludo.
- @RequestParam: Permite recibir un parámetro opcional (nombre) en la solicitud.

Si accedes a http://localhost:8080/saludo?nombre=Francisco, recibirás:

➡ ¡Hola, Francisco!

Si no envías el parámetro, la respuesta será:

➡ ¡Hola, Mundo!

@RequestParam

@RequestParam es una anotación de Spring Boot que se usa en controladores para extraer parámetros de la URL en solicitudes HTTP. Se emplea en métodos de controladores para recibir valores de consulta (query parameters).

```java
@GetMapping("/saludo")
public String saludo(@RequestParam String nombre) {
    return "Hola, " + nombre;
}
```

Si el cliente envía GET /saludo?nombre=Francisco, la respuesta será "Hola, Francisco".

@RequestMapping

@RequestMapping es una anotación en Spring que se usa para mapear solicitudes HTTP a métodos específicos en un controlador. Se puede aplicar a métodos o clases y permite definir la ruta (URL), el tipo de solicitud (GET, POST, etc.) y otros parámetros como los parámetros de la solicitud.

```java
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MiControlador {

    @RequestMapping("/saludo")
    public String saludo() {
        return "¡Hola, mundo!";
    }
}
```

Este método se ejecutará cuando se haga una solicitud GET a /saludo.

Resumen:

- @RequestMapping se usa para asociar una URL a un método.
- Puede definir el tipo de solicitud HTTP y otras configuraciones.

@Autowired

@Autowired es una anotación de Spring que permite inyectar automáticamente dependencias en una clase, eliminando la necesidad de instanciarlas manualmente.

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UsuarioService {

    @Autowired
    private UsuarioRepository usuarioRepository;

    public void guardarUsuario(Usuario usuario) {
        usuarioRepository.save(usuario);
    }
}
```

¿Qué hace?

- Spring detecta la dependencia (UsuarioRepository) y la inyecta automáticamente.
- Facilita la gestión de componentes sin necesidad de new.
- Funciona con clases gestionadas por Spring (@Service, @Repository, @Component).

@PostMapping

@PostMapping es una anotación en Spring que se utiliza para mapear una solicitud HTTP POST a un método específico de un controlador. Generalmente, se usa para enviar datos desde el cliente al servidor, como al crear o actualizar un recurso.

```
@PostMapping("/usuario")
public Optional <Usuario> crearUsuario(@RequestBody Usuario usuario) {
    return usuarioRepository.save(usuario);
}
```

Optional → Puede ser opcional, puede existir o no el Usuario

@RequestBody

@RequestBody es una anotación en Spring que se usa para vincular el cuerpo de una solicitud HTTP a un objeto Java. Permite que los datos enviados en el cuerpo de la solicitud (generalmente en formato JSON o XML) sean automáticamente deserializados y convertidos en un objeto Java que el controlador puede manejar.

En este caso, los datos enviados en el cuerpo de la solicitud (por ejemplo, un JSON con la información de un usuario) se convierten en un objeto Usuario para ser procesado dentro del método del controlador.

CrudRepository

CrudRepository es una interfaz de Spring Data que proporciona operaciones CRUD (Create, Read, Update, Delete) para interactuar con bases de datos de forma sencilla.

Características principales:

- Forma parte de Spring Data JPA.
- Permite gestionar entidades sin escribir SQL manualmente.
- Proporciona métodos como save(), findById(), findAll(), deleteById(), etc.

Ejemplo de uso

Clase Usuario	Usuario usando CrudRepository

```java
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private String email;

    // Getters y Setters
}
```

```java
import org.springframework.data.repository.CrudRepository;

public interface UsuarioRepository extends CrudRepository<Usuario, Long>
{
}
```

Uso en un servicio o controlador:

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

@RestController
@RequestMapping("/usuarios")
public class UsuarioController {

    @Autowired
    private UsuarioRepository usuarioRepository;

    @PostMapping
    public Usuario crearUsuario(@RequestBody Usuario usuario) {
        return usuarioRepository.save(usuario);
    }

    @GetMapping("/{id}")
    public Optional<Usuario> obtenerUsuario(@PathVariable Long id) {
        return usuarioRepository.findById(id);
    }

    @DeleteMapping("/{id}")
    public void eliminarUsuario(@PathVariable Long id) {
        usuarioRepository.deleteById(id);
    }
}
```

findAll()

findAll() es un método de la interfaz CrudRepository en Spring Data JPA que recupera todas las entidades de una tabla en la base de datos.

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/usuarios")
public class UsuarioController {

    @Autowired
    private UsuarioRepository usuarioRepository;

    @GetMapping
    public Iterable<Usuario> listarUsuarios() {
        return usuarioRepository.findAll();
    }
}
```

¿Qué devuelve findAll()?

Retorna un Iterable<T>, donde T es la entidad (en este caso, Usuario).
Se puede recorrer con un for o convertir en una lista (

```
List<Usuario> usuarios = (List<Usuario>) usuarioRepository.findAll();
```

).