GUÍA CONTROLADORES

1. ¿Qué es un Controlador en Spring Boot?

Un controlador es una clase en la que definimos métodos que se ejecutan cuando un usuario realiza una acción, como hacer clic en un enlace o enviar un formulario. En Spring Boot, utilizamos anotaciones para vincular una URL (por ejemplo, /bienvenida) a un método que responderá a esa solicitud.

Ejemplo de un Controlador Sencillo

Este es un ejemplo básico de un controlador que responde a la solicitud de un saludo:

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

// La anotación @Controller indica que esta clase es un controlador.
@Controller
public class MiPrimerControlador {

   // La anotación @GetMapping vincula la ruta /bienvenida con este método.
   @GetMapping("/bienvenida")
    // @ResponseBody significa que el valor devuelto (el mensaje) será el cuerpo de la respuesta HTTP.
   @ResponseBody
   public String mostrarBienvenida() {
      // Este método simplemente devuelve un mensaje de bienvenida.
      return "¡Bienvenido a Spring Boot!";
   }
}
```

Explicación del Código:

@Controller: Le dice a Spring Boot que esta clase es un controlador, es decir, se encargará de manejar solicitudes web.

@GetMapping("/bienvenida"): Esta anotación mapea (vincula) la URL /bienvenida a este método. Cuando el usuario visita esta URL, Spring ejecuta el método mostrarBienvenida().

@ResponseBody: Le indica a Spring que el valor devuelto (en este caso, el texto "¡Bienvenido a Spring Boot!") será enviado como respuesta directa al navegador.

return "¡Bienvenido a Spring Boot!": Este es el mensaje que se enviará como respuesta cuando el usuario visite la URL /bienvenida.

2. Configuración Básica del Proyecto

Paso 1: Configurar el Proyecto Spring Boot en Replit

Crear un Proyecto Maven: Ve a la plataforma Replit, crea un nuevo proyecto Maven y nómbralo "MiPrimerControlador".

Añadir el pom.xml: En Maven, usamos el archivo pom.xml para gestionar las dependencias (librerías que necesita el proyecto).

Aquí tienes un ejemplo básico de pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://www.w3.org/2001/XMLSchema-instance
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>controlador-basico</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>controlador-basico</name>

  <properties>
```

```
        <java.version>17</java.version> <!-- Especificamos que estamos usando Java 17
-->
    </properties>

    <dependencies>
        <!-- Dependencia de Spring Boot para aplicaciones web -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

3. Crear el Controlador Paso a Paso

Paso 1: Crear la Clase Principal

La clase principal en una aplicación Spring Boot es donde el programa arranca. Esta clase debe incluir el método main() y estar anotada con @SpringBootApplication.

Clase Principal:

```
package com.example.controladorbasico;

import org.springframework.boot.SpringApplication;
```

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

// @SpringBootApplication marca el punto de inicio de la aplicación Spring Boot.
@SpringBootApplication
public class ControladorBasicoApplication {

    // El método main() es el punto de entrada de la aplicación.
    public static void main(String[] args) {
        // SpringApplication.run() inicia la aplicación Spring Boot.
        SpringApplication.run(ControladorBasicoApplication.class, args);
    }
}
```

Paso 2: Crear el Controlador

Crear la Clase del Controlador:

Crea una nueva clase llamada MiPrimerControlador.java en el paquete com.example.controladorbasico.

Escribe el Código del Controlador: Copia el siguiente código y colócalo en la clase:

```
package com.example.controladorbasico;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

// @Controller indica que esta clase maneja solicitudes HTTP.
@Controller
public class MiPrimerControlador {

    // @GetMapping mapea la URL "/hola" a este método.
    @GetMapping("/hola")
```

```
    // @ResponseBody le dice a Spring que devuelva el valor directamente en el cuerpo de la respuesta.
    @ResponseBody
    public String decirHola() {
        // Este método devuelve el texto "¡Hola, Mundo!" como respuesta.
        return "¡Hola, Mundo!";
    }
}
```

4. Anotaciones Importantes en los Controladores

A lo largo de la práctica, utilizaremos algunas anotaciones clave para manejar diferentes tipos de solicitudes HTTP. Aquí te explicamos las más importantes:

- 1. @GetMapping

  Se utiliza para manejar solicitudes GET (cuando un usuario solicita ver una página o un recurso).

  Mapea la URL a un método en el controlador.

Ejemplo:

```
@GetMapping("/bienvenida")
@ResponseBody
public String mostrarBienvenida() {
    return "¡Bienvenido a la aplicación!";
}
```

- 2. @PostMapping

  Se usa para manejar solicitudes POST (cuando un usuario envía datos, como en un formulario).

  Similar a @GetMapping, pero para enviar datos al servidor.

Ejemplo:

```
@PostMapping("/enviar")
@ResponseBody
public String enviarDatos(@RequestParam("nombre") String nombre) {
```

```
    return "Datos recibidos: " + nombre;
}
```

- 3. @RequestParam

  Captura parámetros que vienen en la URL o en un formulario.

Ejemplo:

```
@GetMapping("/saludo")
@ResponseBody
public String saludoPersonalizado(@RequestParam(name = "nombre", defaultValue =
"Invitado") String nombre) {
    return "Hola, " + nombre + "!";
}
```

En este caso, si la URL es /saludo?nombre=Juan, el controlador devolverá "Hola, Juan!".

Si no se proporciona un nombre, devolverá "Hola, Invitado!".

5. Ejercicio de Práctica

Objetivo:

Crear dos controladores adicionales que respondan a las siguientes solicitudes:

- Controlador 1: Ruta /productos. Devuelve una lista de productos ficticios.

  Respuesta esperada: "Producto 1: Computadora, Producto 2: Smartphone".
- Controlador 2: Ruta /fecha. Devuelve la fecha actual en formato dd/MM/yyyy.

  Pista: Usa la clase java.time.LocalDate.

Sugerencia de Solución:

Controlador 1:

```
@GetMapping("/productos")
@ResponseBody
public String listarProductos() {
    return "Producto 1: Computadora, Producto 2: Smartphone";
}
```

Controlador 2:

```
@GetMapping("/fecha")
@ResponseBody
public String mostrarFecha() {
   // LocalDate.now() devuelve la fecha actual.
   return "Fecha actual: " + java.time.LocalDate.now();
}
```