



Certified Tech Developer

The Ultimate Degree

Back End I

Ejercitación Patrón MVC (API) - Práctica con Docente

Objetivo

Continuando con las ejercitaciones realizadas en la clase asincrónica, vamos a crear un proyecto llamado Entrenador en Spring MVC siguiendo las instrucciones.

- Ejercicio individual
- Complejidad: baja 🔥

Instrucciones

1- Crear un proyecto desde <https://start.spring.io>.

✓ Recordá poner el nombre del proyecto y agregar en dependencias Spring Web.



The screenshot shows the Spring Initializr web application interface. On the left, there is a sidebar with a hamburger menu icon and social media icons for GitHub and Twitter. The main content area is divided into three sections: Project, Language, and Dependencies. The Project section includes radio buttons for Maven Project (selected) and Gradle Project. The Language section includes radio buttons for Java (selected), Kotlin, and Groovy. The Spring Boot section includes radio buttons for various versions, with 2.5.3 selected. The Project Metadata section includes input fields for Group (com.example), Artifact (entrenador), Name (entrenador), Description (Demo project for Spring Boot), and Package name (com.example.entrenador). The Packaging section includes radio buttons for Jar (selected) and War. The Java section includes radio buttons for versions 16, 11 (selected), and 8. The Dependencies section includes a button to add dependencies and a description for Spring Web. At the bottom, there are three buttons: GENERATE (with a keyboard shortcut), EXPLORE (with a keyboard shortcut), and SHARE...

Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (M1) ☐ 2.5.4 (SNAPSHOT) ☒ 2.5.3 ☐ 2.4.10 (SNAPSHOT) ☐ 2.4.9

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 16 ☒ 11 ☐ 8

Dependencies [ADD DEPENDENCIES... ⌘ + B](#)

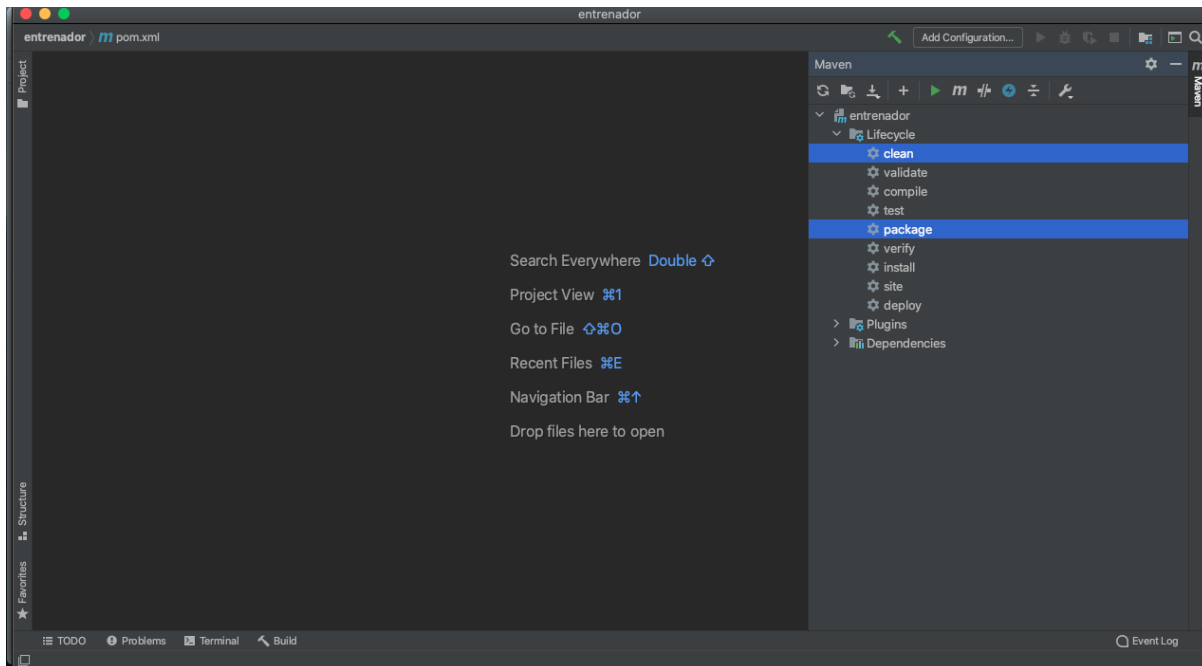
Spring Web **WEB**

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

[GENERATE ⌘ + ↵](#) [EXPLORE CTRL + SPACE](#) [SHARE...](#)

2- Generar el proyecto y descomprimir el archivo .zip para abrirlo en IntelliJ IDEA.
En IntelliJ IDEA elegimos "New -> existing source".

3- En la solapa de Maven hacer clean package y presionar "Play" en la solapa de arriba de Maven.



4- Crear un **modelo**, es decir, una clase de negocio Entrenador.

```
package com.example.entrenador.model;  
  
public class Entrenador {  
  
    private String nombre;  
  
    public Entrenador(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {
```



```
        this.nombre = nombre;  
    }  
}
```

5- Crear un paquete service dentro del proyecto. Agregar la interface **EntrenadorService** y su clase que la implementa: **EntrenadorServiceImpl**.

```
package com.example.entrenador.service;  
  
import com.example.entrenador.model.Entrenador;  
  
import java.util.List;  
  
public interface IEntrenadorService {  
  
    List<Entrenador> listaEntrenador();  
}
```

La annotation **@Service** le dice a Spring que es un servicio.

Vemos cómo en listaEntrenador estamos agregando de manera manual los datos. En una aplicación debemos ir a nuestra capa de DAO para devolverlo desde una base de datos. Por ejemplo:

```
package com.example.entrenador.service;  
  
import com.example.entrenador.model.Entrenador;  
  
import java.util.Arrays;  
import java.util.List;  
  
@Service
```



```
public class EntrenadorService implements IEntrenadorService{  
    @Override  
    public List<Entrenador> listaEntrenador() {  
        return Arrays.asList(new Entrenador("Pibe"), new  
Entrenador("Roman"));  
    }  
}
```

6- Crear un controller en el paquete Controller.

```
import com.example.entrenador.model.Entrenador;  
import com.example.entrenador.service.IEntrenadorService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("entrenador")  
public class EntrenadorController {  
  
    private final IEntrenadorService entrenadorService;  
  
    @Autowired  
    public EntrenadorController(IEntrenadorService entrenadorService) {  
        this.entrenadorService = entrenadorService;  
    }  
}
```



```
@GetMapping
public List<Entrenador> getEntrenador() {
    return entrenadorService.listaEntrenador();
}
```

Como vemos, la clase Controller hace referencia a service (el modelo) y después automáticamente lo transforma en JSON, que sería nuestra vista. Esto pasa dentro de **@GetMapping** annotation. Dentro del Controller debemos agregar **@Controller** para decirle a Spring que este es nuestro controller y **@RequestMapping** para agregar nuestra URL, en este caso /entrenador.

La annotation **@Autowired** la vamos a ver en las próximas clases, pero podemos adelantar que es la conexión entre el modelo y el controller.

Ahora, si corremos nuestro servidor desde el main de la clase EntrenadorApplication y vamos al navegador (por ejemplo Chrome) y ponemos **http://localhost:8080/entrenador**, obtenemos nuestra vista que en este caso es la respuesta a una API Rest: [{"nombre":"Pibe"}, {"nombre":"Roman"}].