



**Certified Tech
Developer**

The Ultimate Degree

Back end I

Exercitando o Padrão MVC (API) com o Professor

Objetivo

Em continuidade aos exercícios realizados no Playground, vamos criar um projeto com o Spring MVC denominado de Trainer seguindo as instruções.

- Exercício individual
- Complexidade: baixa 🔥

Instruções

1. Criar um projeto em <https://start.spring.io>

Lembre-se de atribuir um nome ao projeto e adicionar as dependências: Spring Web e Thymeleaf.

The Spring Initializr interface is shown with the following configuration:

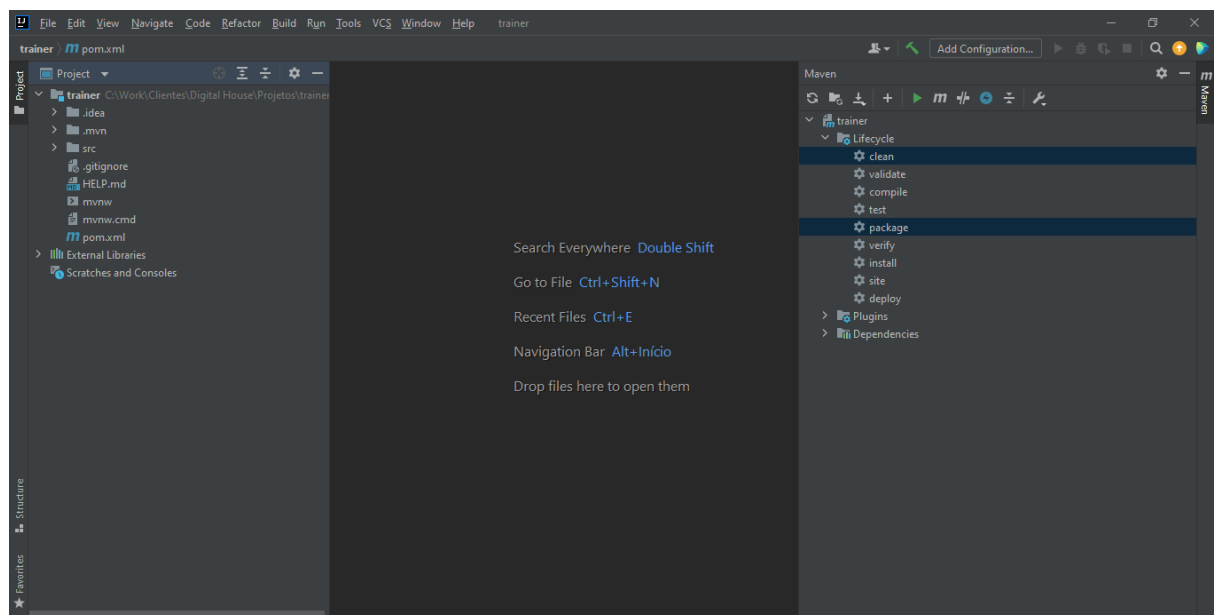
- Project:** ☒ Maven Project, ☐ Gradle Project
- Language:** ☒ Java, ☐ Kotlin, ☐ Groovy
- Spring Boot:** ☐ 2.6.0 (SNAPSHOT), ☐ 2.6.0 (M2), ☐ 2.5.6 (SNAPSHOT), ☒ 2.5.5
- Project Metadata:**
 - Group: com.example
 - Artifact: trainer
 - Name: trainer
 - Description: Demo project for Spring Boot
 - Package name: com.example.trainer
- Packaging:** ☒ Jar, ☐ War
- Java:** ☐ 17, ☒ 11, ☐ 8
- Dependencies:**
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Thymeleaf** (TEMPLATE ENGINES): A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

Buttons at the bottom: GENERATE (CTRL + G), EXPLORE (CTRL + SPACE), SHARE...

- Realizar o download do projeto e descompactá-lo para abrir no IntelliJ IDEA.

No IntelliJ, acesse o menu “File” > “New” > “Project from Existing Sources”;

- Na aba Maven, selecionar “clean” e “package”, e pressionar o “play” na aba superior do Maven.



- Criar um modelo, ou seja, uma classe de negócio Trainer.

```
package com.example.trainer.domain;

public class Trainer {
```

```

private String name;

public Trainer(String name) {
    this.name = name;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
}

```

5. Criar um package service dentro do projeto. Adicionar a interface TrainerService e sua implementação na classe TrainerServiceImpl.

```

package com.example.trainer.service;

import com.example.treinar.domain.Trainer;

import java.util.List;

public interface TrainerService {

    List<Trainer> listTrainer();
}

```

A anotação **@Service** sinaliza ao Spring que é um serviço. Podemos ver em **listTrainer** que estamos adicionando os dados manualmente. Em uma aplicação, devemos ir para nossa camada DAO para obtermos os dados de um banco. Por exemplo:

```

package com.example.trainer.service;

import com.example.trainer.domain.Trainer;
import java.util.Arrays;
import java.util.List;

@Service
public class TrainerServiceImpl implements TrainerService{
    @Override
    public List<Trainer> listTrainer() {
        return Arrays.asList(new Trainer("Marcos"), new New
Trainer("Ana"));
    }
}

```

6. Criar um controller no package controller.

```

import com.example.trainer.domain.Trainer;
import com.example.trainer.service.TrainerService;
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.Controller;

import java.util.List;

@Controller
@RequestMapping("trainer")
public class TrainerController {

    private final TrainerService trainerService;
}

```

```

@Autowired
public TrainerController(TrainerService trainerService) {
    this.trainerService = trainerService;
}

@GetMapping
public List<Trainer> getTrainer() {
    return trainerService.listTrainer();
}
}

```

Como podemos observar, a classe Controller se refere ao serviço (a model) e será transformado automaticamente em JSON, que será nossa visão. Isso acontece dentro da anotação **@GetMapping**. Dentro do Controller, devemos adicionar o **@Controller** para informar ao Spring que esse é o nosso controller e **@RequestMapping** para adicionar nossa URL, neste caso `"/trainer"`.

Veremos nas próximas aulas a anotação **@Autowired**, mas podemos mencionar que trata-se da conexão entre a model e o controller.

Agora podemos executar nosso servidor a partir do método main da classe TrainerApplication e acessar no navegador, por exemplo no Chrome, o endereço <http://localhost:8080/trainer>, devendo ser exibido nossa view, que neste caso é a resposta de uma API Rest: [{"name": "Marcos"}, {"name": "Ana"}].