

# Le mapping des requêtes

## Démonstration 1 du module 4

Les objectifs de cette démonstration sont

- de coder un contrôleur Spring MVC
- et d'expliquer le mapping entre les requêtes http reçues par le serveur d'application et les méthodes du contrôleur

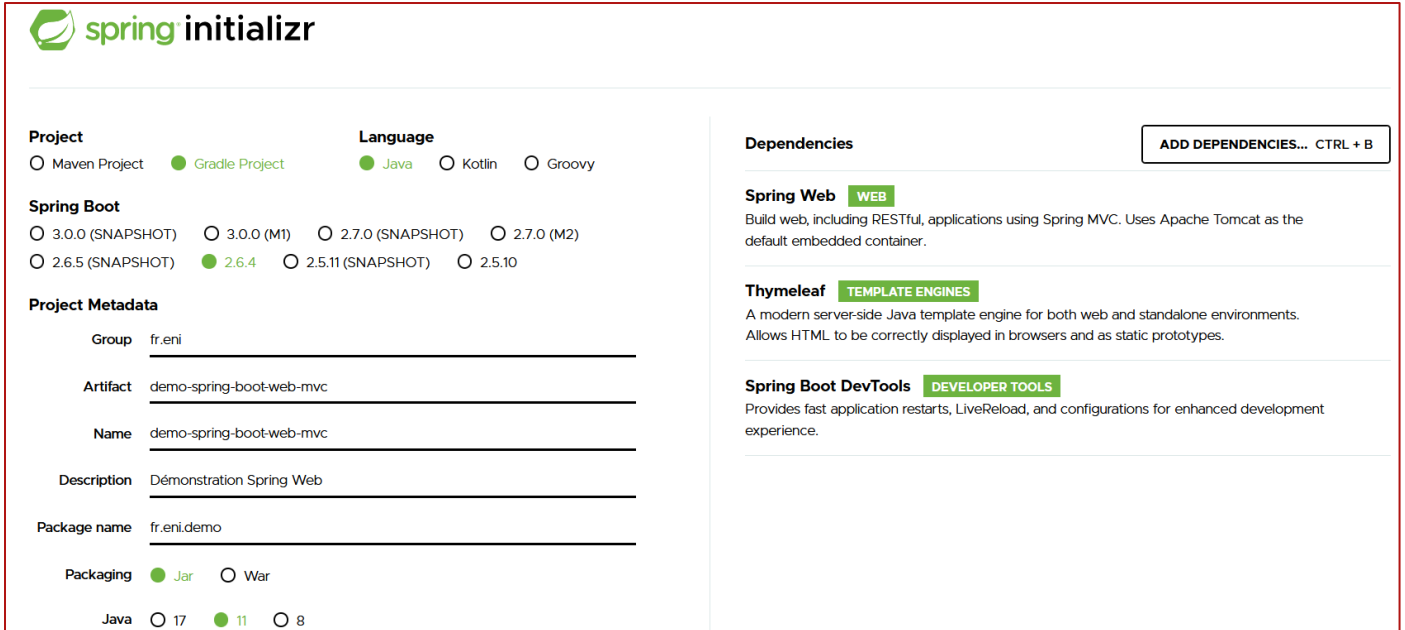
## Contexte

- Les démonstrations seront créées sur une application représentant une partie du cœur de métier de l'ENI Ecole.
  - Nous allons intégrer la couche présentation associée à la couche métier vu précédemment.
  - Pour le moment, nous nous concentrons uniquement sur la couche présentation

## Déroulement

### Création d'un nouveau projet avec Spring Boot

- Aller sur le site : <https://start.spring.io/>
- Sélectionner :
  - Gradle Project (Gradle est le gestionnaire de dépendances que nous utilisons)
  - Java 11
  - Prendre la version la plus récente de Spring Boot GA
  - Group : fr.eni
  - Artifact : demo-spring-boot-web-mvc
  - Package name : fr.eni.demo
- Le livrable sera créé sous la forme d'un .jar et non d'un .war ce qui facilite le déploiement
- Ajouter les librairies : Spring Web, Thymeleaf et Spring Boot Dev Tools ( pour le rechargement automatique du serveur suite à des changements de code)



**Project**  
☐ Maven Project ☒ Gradle Project  
**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M1) ☐ 2.7.0 (SNAPSHOT) ☐ 2.7.0 (M2)  
☐ 2.6.5 (SNAPSHOT) ☒ 2.6.4 ☐ 2.5.11 (SNAPSHOT) ☐ 2.5.10

**Project Metadata**

Group   
 Artifact   
 Name   
 Description   
 Package name   
 Packaging ☒ Jar ☐ War  
 Java ☐ 17 ☒ 11 ☐ 8

**Dependencies** ADD DEPENDENCIES... CTRL + B

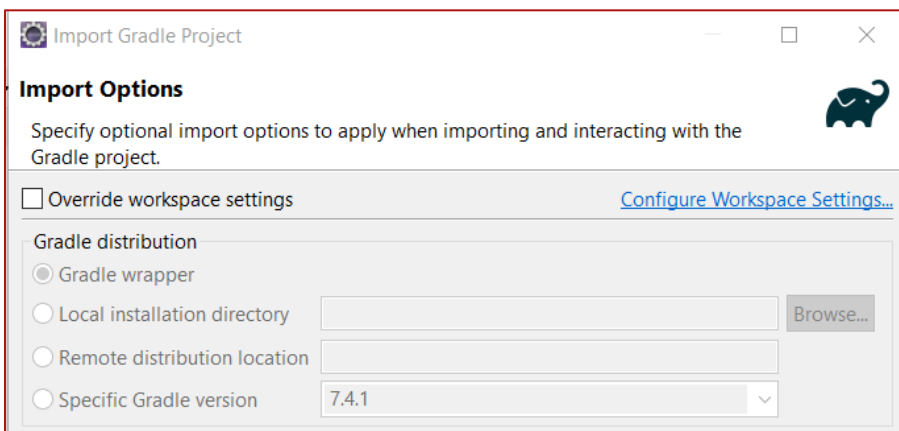
**Spring Web** WEB  
 Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**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.

**Spring Boot DevTools** DEVELOPER TOOLS  
 Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

## Intégration du projet dans Eclipse

- Copier le zip dans le répertoire du Workspace d'Eclipse et dézipper le.
- Faire « Import », sélectionner « Existing Gradle Project » dans Eclipse
  - Sélectionner le projet dans le Workspace
  - Vérifier la configuration de Gradle en 7.4.1 :



**Import Gradle Project**

**Import Options**  
 Specify optional import options to apply when importing and interacting with the Gradle project.

☐ Override workspace settings [Configure Workspace Settings...](#)

**Gradle distribution**

☒ Gradle wrapper

☐ Local installation directory  Browse...

☐ Remote distribution location

☐ Specific Gradle version  ▼

- Et valider

## Page index.html

- Comme dans toute application Web, il est possible avec Spring de créer une page index qui sera chargée par défaut au lancement de l'application.
  - Celle-ci ne dépend pas d'un contrôleur
  - Elle va nous permettre de gérer des liens vers les différents contrôleurs et leurs templates

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Demo Spring Web</title>
</head>
<body>
    <h1>Application rassemblant les démonstrations de Spring Web</h1>

    <nav>
        <a href="trainers">L'ensemble des formateurs</a>
    </nav>
</body>
</html>
```

- Exécuter l'application, l'url par défaut <http://localhost:8080> arrive sur la page d'index.



- Le clic sur le lien hypertexte « L'ensemble des formateurs » renvoie une erreur 404, page not found.
  - Effectivement, il nous reste à définir ce mapping !

## Création du contrôleur TrainerController

- Création du sous package : fr.eni.demo.mmi.controller

Remarque : MMI = Man Machine Interface

- C'est la version anglaise de
- IHM = Interface Homme-Machine

- L'annotation @Controller permet de définir la classe comme un bean Spring de type Controller
- @RequestMapping,
  - Peut se placer sur la classe annotée @Controller pour préciser une URL racine pour toutes les méthodes de la classe
  - peut être déclarée sur les méthodes Java pour gérer le mapping entre elles et une URL
    - Il faut préciser dans ses paramètres : l'url et la méthode du protocole http (get, post, put, delete, ...)

- Exemple :

`@RequestMapping(value = "/get/{id}", method = RequestMethod.GET)`

- Pour le mapping des méthodes de la classe, il y a une nouvelle approche.
  - C'est une solution plus directe et plus simple : `@GetMapping` ou `@PostMapping`, ... (toutes les méthodes du protocole http sont représentées)
    - Le nom de la méthode du protocole http ; est précisé par l'annotation et il suffit d'intégrer l'url en paramètre
    - Exemple : `@GetMapping(value = "/get/{id}")`
- Les 2 solutions sont viables et peuvent être utilisées.
  - Pour la simplification de code, nous utiliserons la nouvelle génération.
- Le retour de la méthode indique le nom de la vue recherchée (template).

```
package fr.eni.demo.mmi.controller;

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

//@Controller --> permet de définir la classe comme un bean Spring de type Controller
@Controller
public class TrainerController {
    @GetMapping("/trainers")
    public String allTrainers() {
        System.out.println("Nous chargerons la liste des formateurs dans une autre démonstration");
        return "view-trainers";
    }
}
```

## Création du template view-trainers.html

- Dans le dossier src/main/resources/templates, créer le fichier html suivant :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Demo Spring Web</title>
</head>
<body>
    <h1>Les formateurs de l'ENI</h1>
    <h2>En construction!!!!</h2>
</body>
</html>
```

- Retester le comportement du lien hypertexte, voici le résultat attendu :

