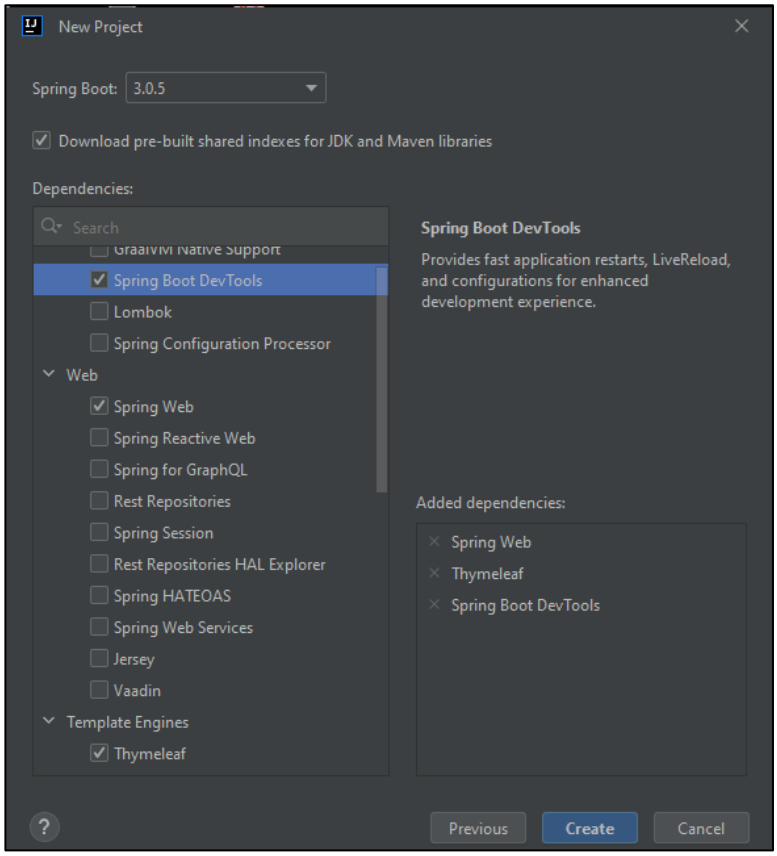
Serveur Web avec Spring boot

Introduction

Plutôt que de coder entièrement le serveur web en Java, il est possible d'utiliser un framework qui prépare le code et simplifie la mise en place du serveur. C'est ce que nous allons faire dans ce TP.

Nouveau projet web avec Spring boot

Ouvrez *Anaconda* et lancez le *CMD exe Prompt*. Le terminal est ouvert.
Cochez les dépendances *Spring Boot Dev Tools, Thymeleaf et Spring Web*.



Il génère tout le projet et un fichier qui ressemble à ça :

```java
package com.example.web_spring;

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

@SpringBootApplication
public class WebSpringApplication {

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

}
```

Créez une classe **WebController** qui contiendra les méthodes pour gérer les requêtes web. Dans cette classe, créez deux méthodes annotées avec **@RequestMapping** pour gérer les requêtes pour les URL **/** et **/about**. Ces méthodes renvoient le nom de la page HTML correspondante à l'aide d'un objet **Model.**

```
package com.example.web_spring;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class WebController {

    @RequestMapping("/")
    public String home(Model model) {
        return "index";
    }

    @RequestMapping("/about")
    public String about(Model model) {
        return "about";
    }
}
```

Créez deux fichiers HTML dans le dossier **src/main/resources/templates**: **index.html** et **about.html**.

Remarque : Pour que les pages HTML soient servies correctement, vous devez vous assurer que les noms des fichiers HTML placés dans le dossier **src/main/resources/templates** correspondent aux noms donnés dans les méthodes du **WebController,** sans l'extension **.html**.

Lancez l'application en exécutant la classe qui contient le **main** générée par Spring Boot. Vous pouvez maintenant accéder à la page d'accueil à l'URL **http://localhost:8080/** et à la page "à propos" à l'URL **http://localhost:8080/about**.

Les liens du site web

La navigation entre les pages se fait de façons très simple : il suffit de réutiliser le url données dans les **RequestMapping** dans les pages HTML..

Avec notre exemple, voici les liens qui peuvent être mis en place :

```
<a href="/about"> A propos </a>
```

```
<a href="/">Index</a>
```

Pour les ressources appelées de la page web, comme css et images, il suffit de les placer dans le répertoire **/static**.

Traitement du formulaire

La gestion d'un formulaire web se fait en 2 étapes sur un serveur web. Dans un premier temps il faut l'afficher, puis il faut récolter les données saisies lorsque le formulaire est soumis.

Servir le formulaire

Soit le formulaire suivant, **form.html** :

```
<form method="post" action="/submit">
  <label for="name">Name:</label>
  <input type="text" name="name" id="name">

  <label for="email">Email:</label>
  <input type="email" name="email" id="email">

  <input type="submit" value="Submit">
</form>
```

Pour que les données soient envoyées au serveur, il faut que tous les champs de saisie doivent être placés dans une balise **form**, dont l'attribut **method = « post »,** qui est la méthode d'envoi, est **action= « url_page_réponse_a_charger »**, qui est la page à charger lorsque le formulaire est soumis. Enfin, il faut ou bouton de soumission qui déclenche l'envoie des données : **<input type="submit" value="Submit">**.

Côté serveur, il suffit de le servir le formulaire comme une autre page web :

```
@RequestMapping("/form")
public String endPointForm(Model model) {
    return "form";
}
```

Traiter le formulaire soumis

Vous pouvez ensuite ajouter une méthode dans votre contrôleur qui prendra en charge la soumission de ce formulaire :

```
public String submitForm(@RequestParam("name") String name,
                         @RequestParam("email") String email,
                         Model model) {

    // Ajout des données dans le modèle
    model.addAttribute("name", name);
    model.addAttribute("email", email);

    // Retourne la vue qui affichera les données du formulaire
    return "result";
}
```

Dans cet exemple, nous utilisons l'annotation **@PostMapping** pour indiquer à Spring que cette méthode doit être appelée lors de la soumission du formulaire. Nous utilisons également les annotations **@RequestParam** pour récupérer les valeurs des champs **name** et **email** du formulaire. Ces valeurs sont ensuite utilisées pour faire tout traitement nécessaire sur les données du formulaire, puis elles sont ajoutées au modèle à l'aide de la méthode **addAttribute**. Enfin, la méthode retourne le nom de la vue (**form_result**) qui affichera les données du formulaire.

Pour compléter cet exemple, les valeurs ajoutées au modèle, **name** et **email** sont ajoutées au modèle, puis la vue **result.html** est chargée et les valeurs sont affichées grâce aux expressions Thymeleaf.

Dans votre vue **result.html**, vous pouvez afficher ces valeurs en utilisant des expressions du template **Thymeleaf** :

```
<body>
<h1>Form Result</h1>
<p>Thank you for submitting the form!</p>
<p>Your name is: <span th:text="${name}"></span></p>
<p>Your email is: <span th:text="${email}"></span></p>
</body>
```

Dans cet exemple, nous utilisons les expressions Thymeleaf **th:text** pour afficher les valeurs de **name** et **email** dans les balises ****. Les expressions Thymeleaf sont entourées de **${}** pour indiquer à Thymeleaf qu'il s'agit d'une expression à évaluer.