
TP2 – Développement d’une application Web avec Spring MVC

Objectifs pédagogiques

À l’issue de ce TP, l’étudiant sera capable de :

- Expliquer l’architecture **Spring MVC**
- Configurer un `DispatcherServlet`
- Implémenter des contrôleurs Spring
- Mapper des requêtes HTTP avec `@RequestMapping`
- Passer des données du contrôleur à la vue
- Comprendre le rôle du modèle (`Model`)
- Structurer une application Web Spring

Prérequis

- Avoir réalisé le **TP1 (IoC & DI)**
- Java 8+
- Maven
- Notions HTTP (GET / POST)
- Serveur Servlet (Tomcat)

Contexte du TP

On développe une **application Web de gestion de messages** :

- L’utilisateur accède à une page Web
- Il saisit un message
- Le serveur traite la requête

- Le message est affiché dynamiquement

👉 L’objectif est de **comprendre le flux MVC**, pas le design graphique.

Partie 1 – Architecture Spring MVC

1.1 Rappel MVC

Spring MVC repose sur :

- **Model** : données métier
- **View** : JSP / Thymeleaf
- **Controller** : logique de traitement des requêtes

Le cœur du framework est :

👉 **`DispatcherServlet`**

1.2 Flux d’exécution

1. Le client envoie une requête HTTP
2. `DispatcherServlet` reçoit la requête
3. Le contrôleur est sélectionné
4. Le contrôleur traite la requête
5. Le modèle est préparé
6. La vue est résolue
7. La réponse HTML est renvoyée

Partie 2 – Configuration Spring MVC (XML)

2.1 Configuration du `web.xml`

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
```

```
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

Questions

1. Quel est le rôle du `DispatcherServlet` ?
2. Pourquoi `/` et non `/*` ?

2.2 Configuration Spring MVC

Créer `dispatcher-servlet.xml`

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="com.tp.mvc"/>
    <mvc:annotation-driven/>

    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

</beans>
```

Partie 3 – Premier contrôleur Spring MVC

3.1 Création du contrôleur

```
@Controller
public class HomeController {

    @RequestMapping("/home")
    public String home() {
```

```
        return "home";
    }
}
```

Analyse

- `@Controller` → composant MVC
- `/home` → URL
- `"home"` → nom logique de la vue

3.2 Création de la vue JSP

`/WEB-INF/views/home.jsp`

```
<html>
<body>
    <h2>Bienvenue dans Spring MVC</h2>
</body>
</html>
```

3.3 Test

Accéder à :

`http://localhost:8080/app/home`

Questions

1. Pourquoi la JSP est-elle dans `/WEB-INF` ?
2. Qui choisit la vue finale ?

Partie 4 – Passage de données (Model)

4.1 Contrôleur avec modèle

```
@Controller
public class MessageController {

    @RequestMapping("/message")
    public String message(Model model) {
        model.addAttribute("msg", "Bonjour Spring MVC");
        return "message";
    }
```

```
}
```

4.2 Vue JSP

```
<html>
<body>
    <h2>${msg}</h2>
</body>
</html>
```

Partie 5 – Formulaire et requêtes POST

5.1 Formulaire JSP

```
<form action="send" method="post">
    <input type="text" name="content"/>
    <input type="submit"/>
</form>
```

5.2 Contrôleur POST

```
@Controller
public class FormController {

    @PostMapping("/send")
    public String send(@RequestParam String content, Model model) {
        model.addAttribute("msg", content);
        return "result";
    }
}
```

5.3 Vue résultat

```
<h2>Message reçu : ${msg}</h2>
```

Partie 6 – Intégration avec le métier (IoC)

6.1 Service métier

```
@Service
public class MessageService {

    public String process(String msg) {
        return msg.toUpperCase();
    }
}
```

6.2 Injection dans le contrôleur

```
@Controller
public class FormController {

    private final MessageService service;

    @Autowired
    public FormController(MessageService service) {
        this.service = service;
    }

    @PostMapping("/send")
    public String send(@RequestParam String content, Model model) {
        model.addAttribute("msg", service.process(content));
        return "result";
    }
}
```

Partie 7 – Bonnes pratiques Spring MVC

1. Contrôleur = orchestration, pas logique métier
2. Logique métier → `@Service`
3. Injection par constructeur
4. URLs claires et REST-friendly
5. JSP protégées dans `/WEB-INF`

Partie 8 – Questions de réflexion

1. Quel est le rôle exact du `DispatcherServlet` ?
2. Pourquoi Spring MVC est-il basé sur Front Controller ?
3. Où s'applique l'IoC dans Spring MVC ?
4. Différence entre `@Controller` et `@Service` ?

Livrables attendus

- Application fonctionnelle
- Code structuré (MVC)
- Schéma d'architecture Spring MVC
- Réponses aux questions

Extensions possibles (optionnelles)

- Migration vers **Spring Boot**
- Remplacement JSP par **Thymeleaf**
- Ajout de validation (`@Valid`)
- Introduction au **REST (`@RestController`)**