TP Spring n°4
Spring Web épisode 1

Nous allons maintenant utiliser le module Web de Spring afin de ne plus faire de l'affichage en console (enfin…) mais d'obtenir un résultat dans un navigateur. Nous allons lancer notre application Spring avec Spring Boot qui nous permet de monter rapidement un serveur en local sur le port 8080.

On testera donc notre application par l'adresse suivante : http://localhost:8080/

1° - Spring Boot

Spring Boot est fondamentalement une extension du framework Spring, qui élimine les configurations standard requises pour la configuration d'une application Spring.

Il est utilisé afin de créer des Micro Services. Un Micro Service est une architecture qui permet aux développeurs de développer et de déployer des services de manière indépendante. Chaque service en cours d'exécution a son propre processus, ce qui permet d'obtenir le modèle léger pour prendre en charge les applications d'entreprise.

Micro services offre les avantages suivants à ses développeurs :

- Déploiement facile
- Évolutivité simple
- Compatible avec les conteneurs
- Configuration minimale
- Moins de temps de production

Utilisation de Spring Boot :

Sur la déclaration de votre classe Application ajoutez la méthode suivante qui permet d'annoter cette application étant maintenant une application qui a vocation à être lancée par Spring Boot.

```
@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })
```

Votre main ne contient maintenant que la chose suivante (renommez votre main précédent) :

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

On va maintenant afficher un message lors du lancement de notre application dans un navigateur web.

Pour cela nous avons besoin de créer un contrôleur. On ajoute un nouveau package à notre projet : le package controller (à bien placer dans fr.votrenom.ecommerce) comme on a ajouté les packages model, service, exception (n'est-ce pas ?).

Dans ce package, ajoutez une classe HomeController. Ajoutez à la déclaration de cette classe l'annotation *@RestController*, qui permet d'identifier ce composant comme un contrôleur.

Ajoutez à ce contrôleur la méthode suivante permettant d'afficher le message suivant :

```
@GetMapping("/")
public String index() {
	return "Bienvenue sur mon super site";
}
```

Lancez votre application avec Spring Boot et lancez dans votre navigateur le localhost sur le port 8080. Vous devez voir le message apparaître.

Ajouter une méthode avec une nouvelle route */bonjour* :

```
@GetMapping("/bonjour")
public String printBonjour() {
	return "Ici on dit bonjour";
}
```

Testez cette route pour comprendre le fonctionnement et le principe de mapping d'une requête Get sur le protocole HTTP (rien à voir avec une méthode de type get en Java).

2° - La vue

Nous allons créer la vue au format jsp.

Les JSP (Java Server Pages) sont une technologie Java qui permet la génération de pages web dynamiques. La technologie JSP permet de séparer la présentation sous forme de code HTML et les traitements écrits en Java sous la forme de JavaBeans ou de servlets. Ceci est d'autant plus facile que les JSP définissent une syntaxe particulière permettant d'appeler un bean et d'insérer le résultat de son traitement dans la page HTML dynamiquement.

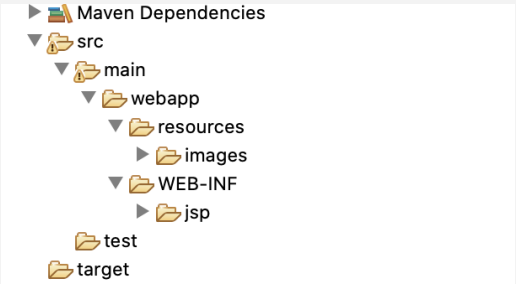
Ajouter le dossier webapp dans le dossier src/main, dans ce dossier ajoutez deux dossiers :

- WEB-INF
- resources

Vous pouvez ajouter dans le dossier *resources* un dossier *images*, afin d'ajouter des images dans vos pages.

Dans le dossier WEB-INF, ajoutez un dossier jsp afin d'y placer vos pages jsp.

Vous devez obtenir l'architecture suivante :



Nous allons maintenant faire un affichage de nos pages en jsp, avant cela il faut préciser l'emplacement de nos fichier jsp et le suffix.

Dans le fichier *application.properties* (qui doit être vide) ajouter les deux lignes suivantes :

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

Dans votre fichier pom.xml de votre configuration Maven ajoutez les dépendances suivantes :

```
<dependency>
	<groupId>org.apache.tomcat.embed</groupId>
	<artifactId>tomcat-embed-jasper</artifactId>
</dependency>
<dependency>
	<groupId>javax.servlet</groupId>
	<artifactId>javax.servlet-api</artifactId>
</dependency>
<dependency>
	<groupId>org.glassfish.web</groupId>
	<artifactId>jakarta.servlet.jsp.jstl</artifactId>
</dependency>
```

Nous allons maintenant pouvoir créer notre première page jsp. Pour cela créez un nouveau fichier dans votre dossier dédié *jsp* :
Clic droit sur jsp : New > Other > Web > JSP File
Nom du fichier home.jsp

Vous devez obtenir un fichier avec le code html suivant :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>

</body>
</html>
```

Ajoutez dans le *body* :

```
<h1>Ca marche !!!!!!!</h1>
```

2° - Le controller

C'est le controller qui va se charger de lancer la vue. Du coup la méthode index va retourner simplement *"home"* .
Et rappelez vous, vous aviez défini dans le fichier de configuration (*application.properties*) le suffix jsp et le prefix /WEB-INF/jsp/ : on obtient donc le chemin /WEB-INF/jsp/home.jsp qui est le bon chemin de notre fichier à afficher.

Sauf qu'un RestController retourne du texte mais ici on veut faire référence à la page. On modifie l'annotation par *@Controller* seulement qui permet d'obtenir le résultat souhaité :

```
@Controller
public class HomeController {

	@GetMapping("/")
	public String index() {
		System.out.println("/ : Home page");
		return "home";
	}
}
```

Relancer votre serveur et rafraîchissez la page sur le navigateur et vous devriez voir la page html s'afficher !

Vous pouvez créer une page web entièrement. J'ai pour ma part pris le code html de la page d'exemple suivant :

https://getbootstrap.com/docs/4.0/examples/album/#

Après 2-3 modification on obtient un résultat comme celui-ci :