



**École Doctorale en Sciences Economique, Gestion et Informatique**  
**Faculté des Sciences Économiques et de Gestion de Sfax**  
**Département Informatique**

## **Auditoire**

**M**astère Professionnel Audit et Sécurité Informatique

# **Architectures des Systèmes d'Information (JavaEE)**

**Enseignante**

**AÏDA KHEMAKHEM**

**Année Universitaire**

**2019 – 2020**

# Chapitre 3 :

## Controller – Model – View

**I. Architecture Spring Boot**

**II. Les principales annotations**

**III. Contrôleur : exemple**

**IV. Les paramètres de la méthode GET**

**V. Principe de fonctionnement MVC**

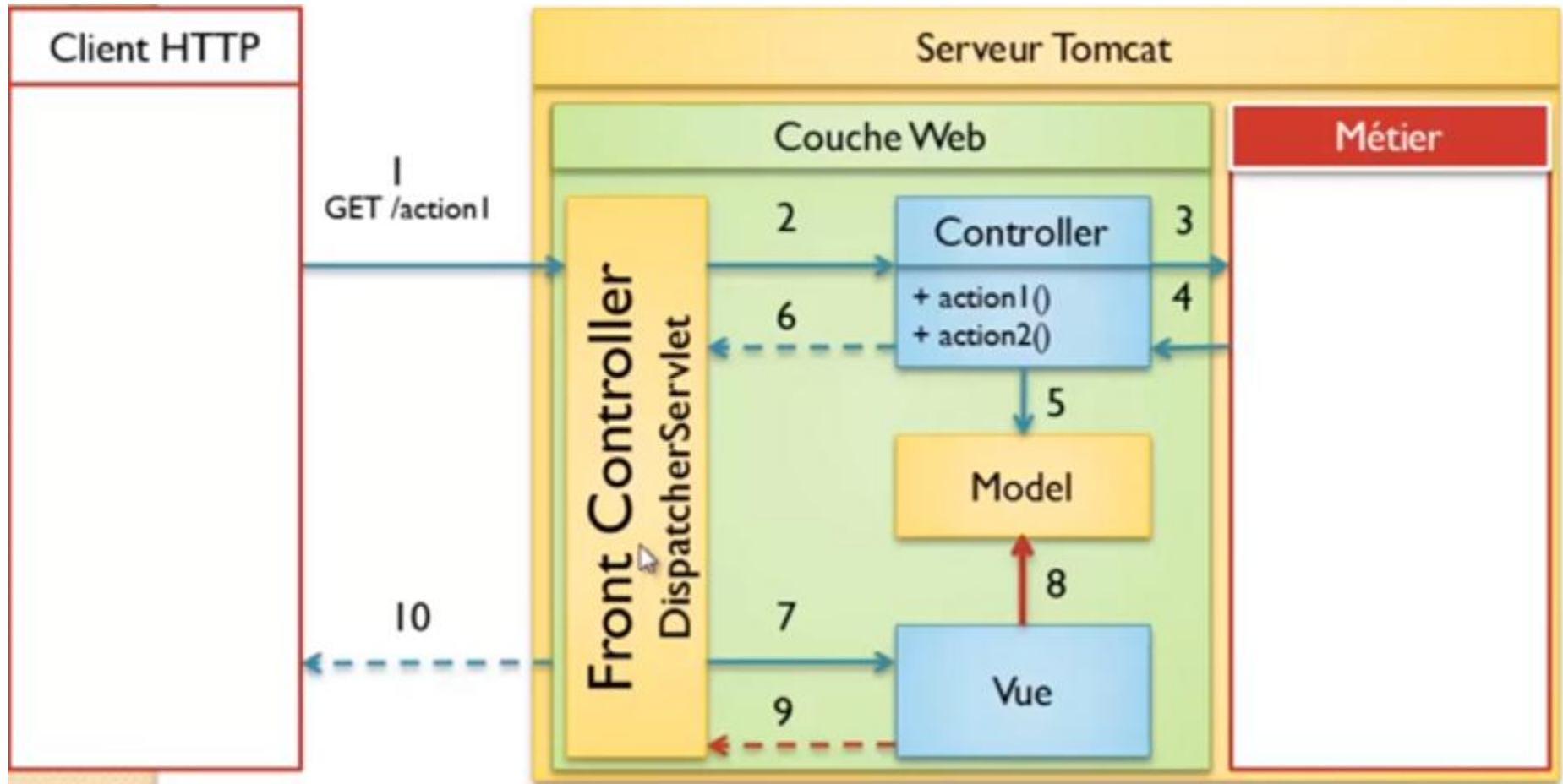
**VI. Le modèle**

**VII. View avec Thymeleaf**

**VIII. Méthode Get & POST**

**IX. Quelques balises HTML**

# I. Architecture Spring Boot



## I.1. Description des étapes (1/3)

1. Le clients fait une demande au contrôleur qui voit passer toutes les demandes des clients. C'est la porte d'entrée de l'application. Ce contrôleur est assuré par la servlet générique :

**org.springframework.web.servlet.DispatcherServlet**

2. Le contrôleur principal [DispatcherServlet] fait exécuter l'action demandée par l'utilisateur par une classe implémentant l'interface :

**org.springframework.web.servlet.mvc.Controller**

3. Le contrôleur [Controller] traite une demande particulière de l'utilisateur. Pour ce faire, il peut avoir besoin de l'aide de la couche métier.

4. Après, les traitements dans le couche métier, le contrôleur reçoit le résultat sous forme d'objet

## I.1. Description des étapes (2/3)

**5.** Avant de générer la réponse au client, le contrôleur envoie les informations calculées au Model **M** de la vue (réponse à envoyer au client). Spring MVC fournit ce modèle sous forme d'un dictionnaire de type **java.util.Map**

**6.** Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :

- Une page d'erreurs si la demande n'a pu être traitée correctement
- Une page de confirmation sinon

Alors le contrôleur envoie à **DispatcherServlet** la vue qui peut être

- Une réponse simple sous forme de chaîne de caractères à afficher au client, c'est-à-dire passer directement à l'étape 10
- Une page html qui nécessite l'initialisation des parties dynamiques de la réponse, c'est-à-dire passer à l'étape 7

## I.1. Description des étapes (3/3)

7. Le contrôleur principal DispatcherServlet demande la vue choisie de s'afficher
8. Le générateur de vue utilise modèle **Map** préparé par le contrôleur pour initialiser les parties dynamiques de la réponse qu'il doit envoyer au client
9. La Vue envoie la réponse DispatcherServlet
10. Le contrôleur principal DispatcherServlet envoie la réponse au client. La forme exacte de cette réponse dépend du générateur de vue. Ce peut être un flux HTML, XML, PDF..

## II. Les principales annotations

- `@Controller` : cet annotation précède une classe contrôleur qui regroupe des actions (méthodes)
- `@RequestMapping(value = "/auteur")` rendent le nom de la vue qui doit être affichée. L'action est une combinaison d'une vue et d'un modèle construit par l'action pour cette vue qui fournit la réponse envoyée au client
  - `value = ""` : URL pour accéder à l'action
  - `method=RequestMethod.GET` :  
type de la requête (**GET, POST...**)
- `@ResponseBody` : cet annotation précède une méthode d'action, la valeur retournée est **une chaine envoyée** pour le client

### III. Contrôleur : exemple

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping(value = "/auteur") //optionnel Racine de l'URL

public class AuteurController {

    @RequestMapping(value = "/bonjourPar") //URL pour accéder à cet action
    @ResponseBody //si la réponse est une chaine (n'est pas un fichier)
    public String lireBonjourStatic( ) //type de retour est String ou void
    {return "Bonjour dans la FSEGS";}
}
```



# @RequestMapping: précède la classe ou la méthode

```
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;
```

```
@Controller
```

```
@RequestMapping(value = "/auteur")
```

```
public class AuteurController {
```

```
@RequestMapping(value = "/bonjour")
```

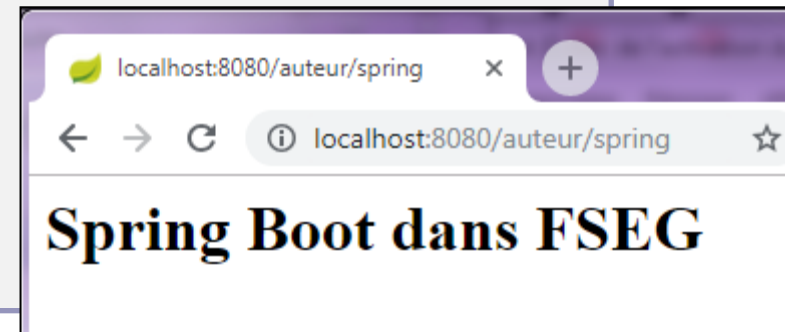
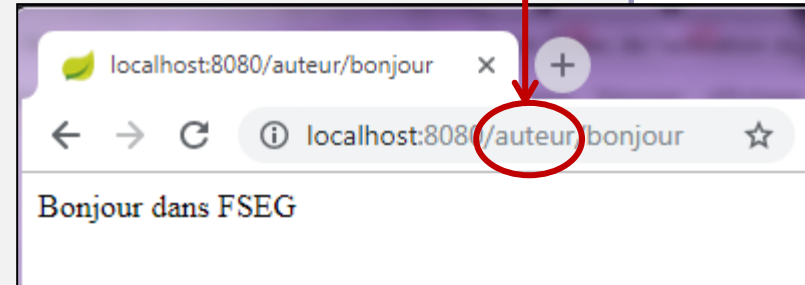
```
@ResponseBody
```

```
public String afficherBonjour()  
{return "Bonjour dans FSEG";  
}
```

```
@RequestMapping(value = "/spring")
```

```
@ResponseBody
```

```
public String presenterSB()  
{return "<h1>Spring Boot dans FSEG</h1>";  
}
```



# Contrôleurs de Spring **sans** les ressources

```
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;
```

```
@Controller  
public class AuteurController {
```

```
@RequestMapping(value = "/bonjour")
```

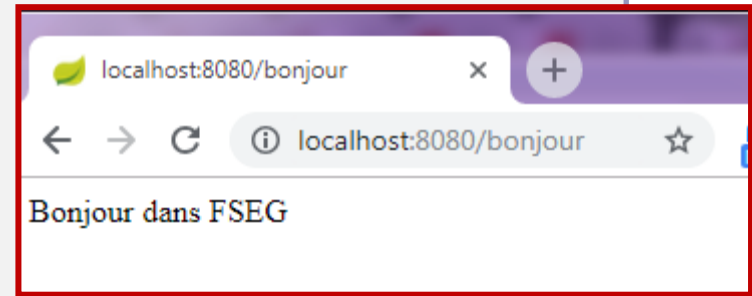
```
@ResponseBody
```

```
public String afficherBonjour()  
{return "Bonjour dans FSEG";  
}
```

```
@RequestMapping(value = "/spring")
```

```
@ResponseBody
```

```
public String presenterSB()  
{return "<h1>Spring Boot dans FSEG</h1>";  
}
```



# Contrôleurs de Spring **avec** les ressources

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
```

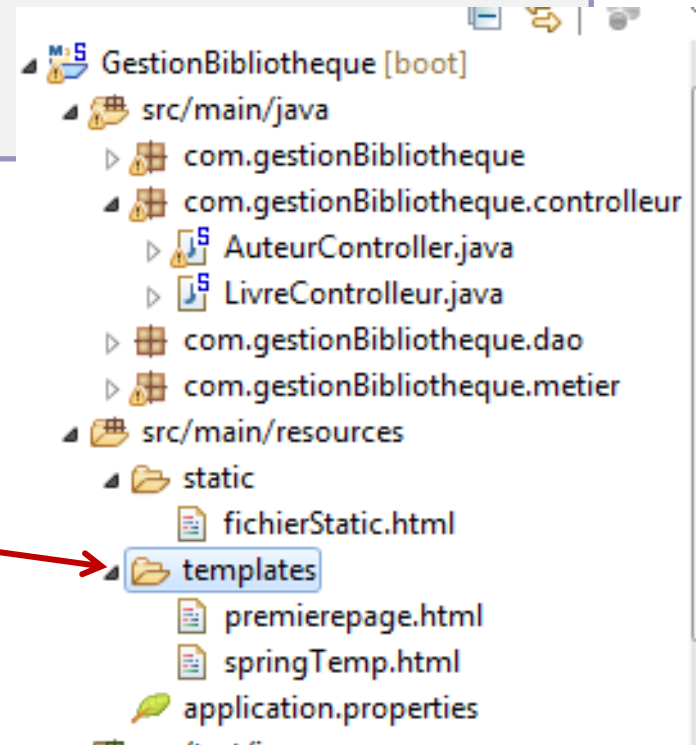
```
@Controller
public class AuteurController
{
```

```
@RequestMapping(value = "/springtemp")
public String lireSB()
{return "springTemp";}
```

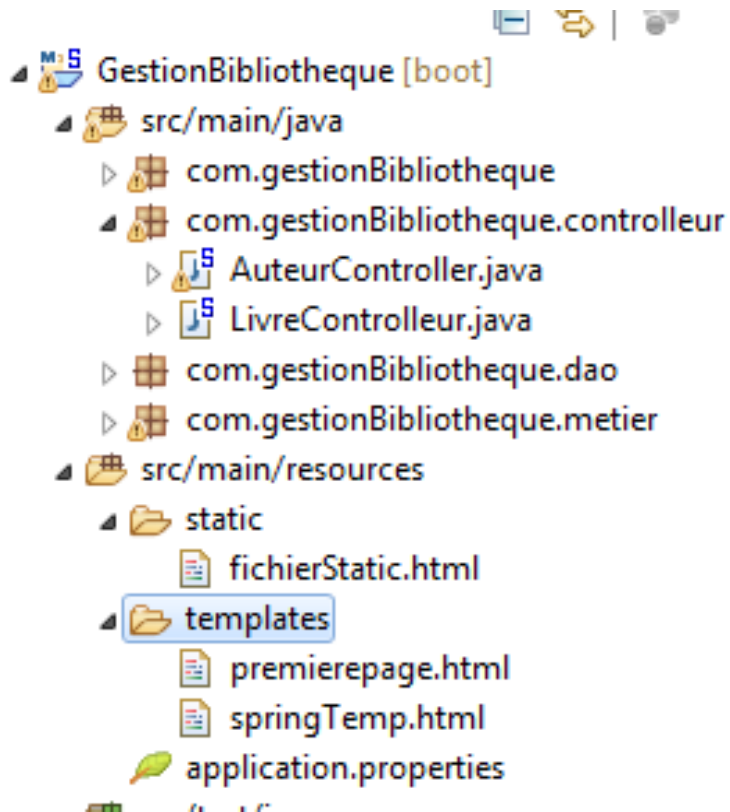
**Template : Spring Boot dans FSEG**

springTemp.html

```
<!DOCTYPE html>
<html><head><meta charset="ISO-8859-1">
<title>Insert title here</title></head>
  <body>
<h1>Template : Spring Boot dans FSEG</h1>
  </body></html>
```

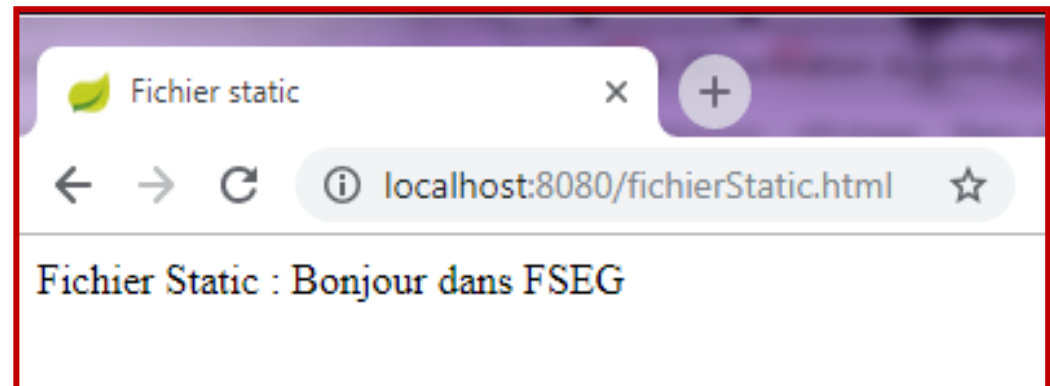


# Remarque : /ressources/**static**/xxx.html



## fichierStatic.html

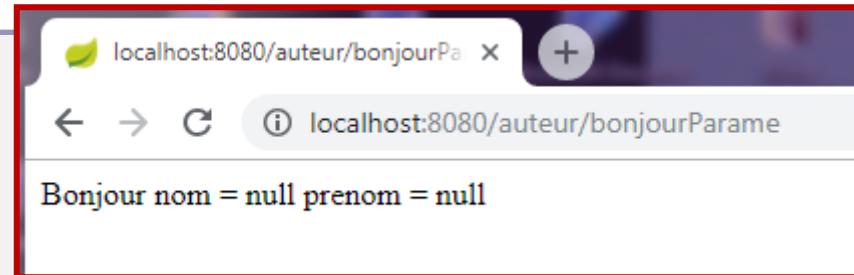
```
<!DOCTYPE html>
<html>
<head><meta charset="ISO-8859-1">
<title>Fichier static</title>
</head>
<body>
Fichier Static : Bonjour dans FSEG
</body>
</html>
```



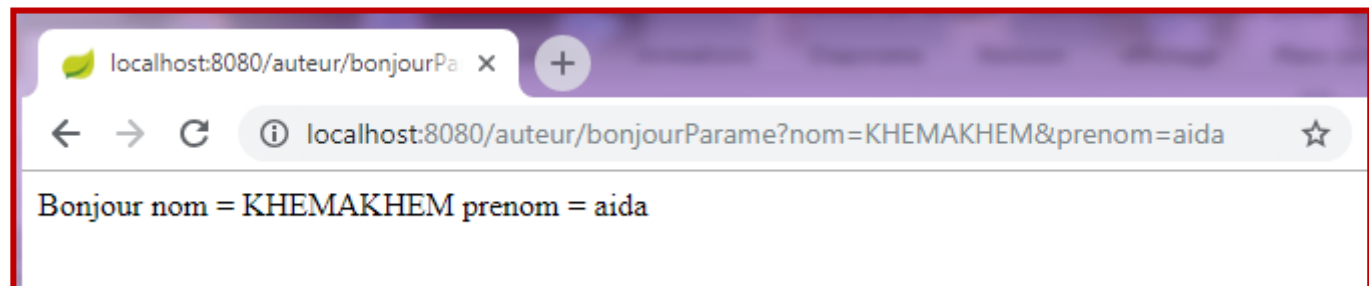
# IV. Les paramètres de la méthode GET

```
@Controller
@RequestMapping(value = "/auteur")

public class AuteurController {
    @RequestMapping(value = "/bonjourPar")
    @ResponseBody
    public String lireBonjourStatic(String nom, String prenom)
    {return "Bonjour nom = "+nom+" prenom = "+prenom;
    }
}
```



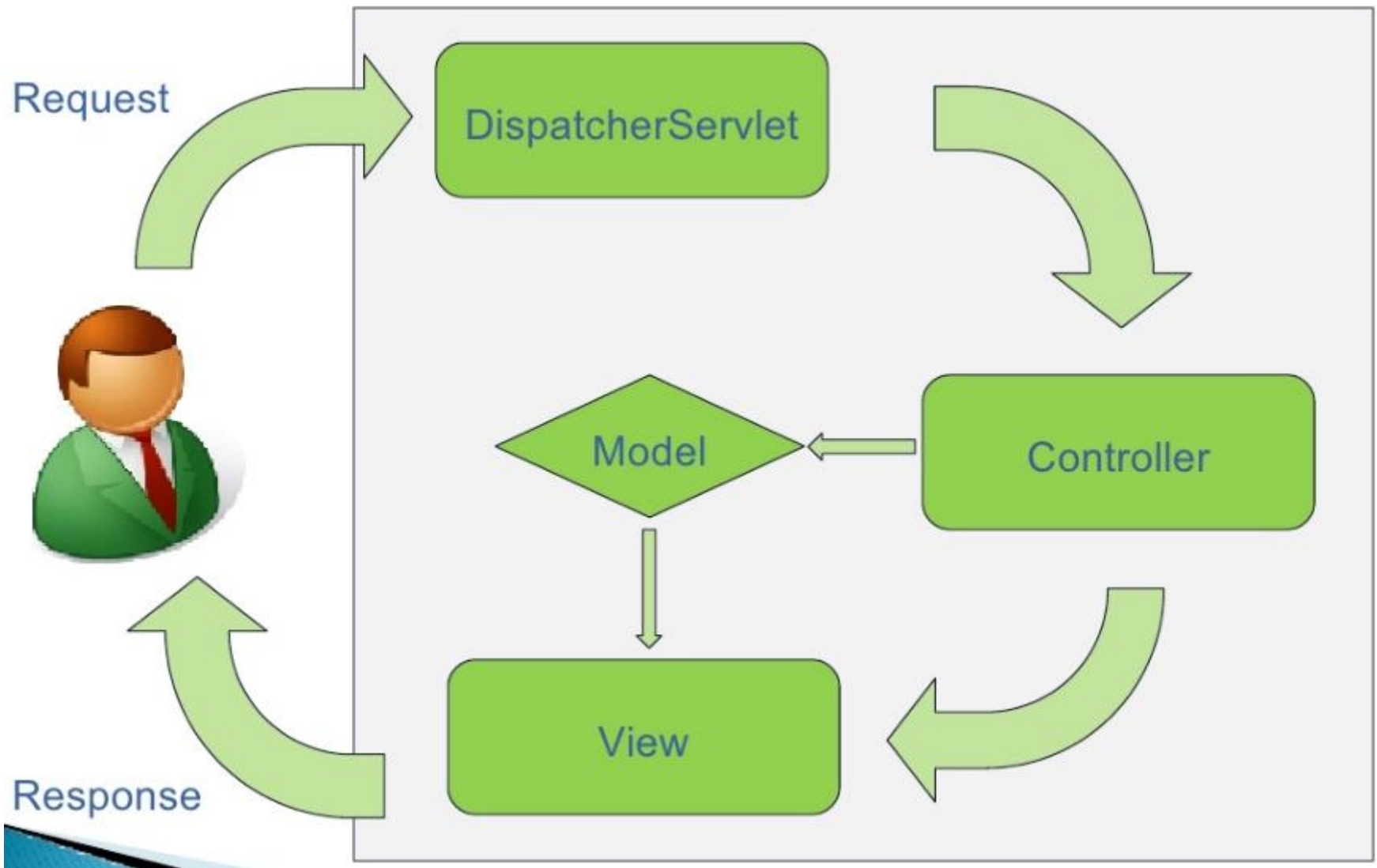
localhost:8080/auteur/bonjourParame?**nom**=KHEMAKHEM**&prenom**=aida



# V. Principe de fonctionnement M-V-C

- Les **contrôleurs** et les **vues** partagent les objets stockés dans le modèle
- Dans les actions du **Controller** qui utilisent le **Model**, il faut :
  - Ajouter un paramètre (**model**) de type **Model**
  - Affecter dans ce paramètre des attributs :  
**model.addAttribute("age", 12) ;**  
**model.addAttribute("liv", new Livre("22RRR", "Java", 1999)) ;**
- Dans les vues qui utilisent le Model, il faut
  - Utiliser les attributs affectés en utilisant \$ avec nom de attribut :  
`<span th:text="{age}">Age sans thymeleaf</span>`  
`<span th:text="{liv.titre}"> </span>`

# Controller – Model –View



## VI. Le modèle

- Un modèle de type [Model] est une sorte de dictionnaire d'éléments de type **java.util.Map** sous forme de <String, Object>

- Pour ajouter une entrée dans ce dictionnaire :

- Utiliser la méthode **model.addAttribute(String, Object)** :
- Spécifier la clé comme premier paramètre de type String
- Spécifier l'objet (ou la valeur) associé à cette clé
- Exemple :

```
model.addAttribute("liv", new Livre("22RRR", "Java", 1999));
```

➤ La clé [liv] associée à une valeur de type [Livre]

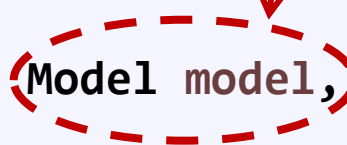


# Exemple d'utilisation de Model dans Controller

```
...
import org.springframework.ui.Model;

@Controller
@RequestMapping(value = "/auteur")

public class AuteurController {
    @RequestMapping(value = "/bonjourModel")
    @ResponseBody
    public String bonjourModel(Model model, String nom, String prenom)
    {
        model.addAttribute("nom", nom);
        model.addAttribute("prenom", prenom);
        model.addAttribute("liv", new Livre("22RRR", "Java", 1999));
        return "nom = "+nom+"\n prenom = "+prenom+"\n modele
        =" +model.toString();
    }
}
```



## VII. View avec Thymeleaf

- **Thymeleaf** est un **Template Engine** de **Java XML/XHTML /HTML5** qui peut travailler à la fois dans des environnements Web (Servlet) et celui de non Web
- Il est mieux adapté pour diffuser **XHTML / HTML5** sur **View** (View Layer) des applications **Web** basées sur **MVC**
- Il peut traiter n'importe quel fichier **XML** même dans des environnements hors ligne (offline).
- Il fournit une intégration complète de **Spring Framework**.

## VII.1. Comment utiliser Thymeleaf

- Tous les fichiers **HTML** doivent déclarer l'utilisation de **Thymeleaf Namespace**

```
<!-- Thymeleaf Namespace -->  
<html xmlns:th="http://www.thymeleaf.org">
```

- Les fichiers **HTML** doivent être adaptés aux normes de **XML**, toutes les balises doivent être ouvertes et fermées
- Dans les fichiers modèles (Template file), il y a des **Thymeleaf Marker** (des marqueurs de Thymeleaf) qui sont des instructions aidant les données de processus de **Thymeleaf Engine**

Thymeleaf  
Marker

```
<tr th:each = "person : ${persons}">  
  <td th:utext="${person.firstName}">...</td>  
  <td th:utext="${person.lastName}">...</td>  
</tr>
```

- Les expressions variables ( **`${...}`** ) sont Spring EL et exécutent sur les attributs de modèle
- les expressions de lien ( **`@{...}`** ) réécrivent les URL
- les expressions astérisques ( **`*{...}`** ) sont exécutées sur le bean de sauvegarde du formulaire
- et les expressions hachées ( **`#{...}`** ) sont destinées à l'internationalisation

## VII.3. Exemple d'utilisation de Model dans une Vue

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title th:text="Livre">Livre sans Thymeleaf</title>
<meta http-equiv="Content-Type"
      content="text/html; charset=UTF-8" />
</head>

<body>
<div> <label>Nom :</label>
  <span th:text="{nom}">Nom sans thymeleaf</span>
</div>
<div> <label>Prénom:</label>
  <span th:text="{prenom}">Prénom sans thymeleaf</span>
</div>

<div> <label>Livre :</label>
  <span th:text="{liv.id}">Id sans thymeleaf</span>
  <span th:text="{liv.titre}"> Titre sans thymeleaf</span>
</div>
</body>
</html>
```

Voir page  
8

## VII.3. Exemple d'utilisation de Model dans une Vue (suite)

```
<table align=center>
<thead><tr>
<th>ISBN    </th>
<th>Titre    </th>
<th>Année    </th>
</tr></thead>
```

```
<tbody>
<tr th:each="e:${LivresRech}">
<td th:text="${e.isbn}"></td>
<td th:text="${e.tite}"></td>
<td th:text="${e.annee}"></td>
</tr>
</tbody>
</table>
```

Boucle for each pour parcourir une collection

Une collection dans le Model

## VIII. Méthode Get & POST

- Il y a deux méthodes pour transmettre des données à un serveur Web : GET et POST.

Y-a-t-il une méthode meilleure que l'autre ?

- Pour envoyer les valeurs d'un formulaire avec la méthode GET, le navigateur affiche dans son champ **Adresse** l'URL demandée sous la forme **URL?param1=val1&param2=val2&....**
- On peut voir cela comme un avantage ou un inconvénient :
  - ☺ un avantage si on veut permettre à l'utilisateur de placer cette URL paramétrée dans ses liens favoris ;
  - ☹ un inconvénient si on ne souhaite pas que l'utilisateur ait accès à certaines informations du formulaire tels, par exemple, les champs cachés.

Nous utiliserons quasi exclusivement la méthode POST dans les formulaires de saisies et la méthode GET pour les recherches

# Exemple de formulaire avec la méthode GET

```
<form method="get" action="chercherLivres">
<table align=center>
<tr><td>Titre </td>
      <td><input type="text" name="titre"></input></td>
      <td><input type="submit" value="Chercher"></input></td>
</tr>
</table>
</form>
```

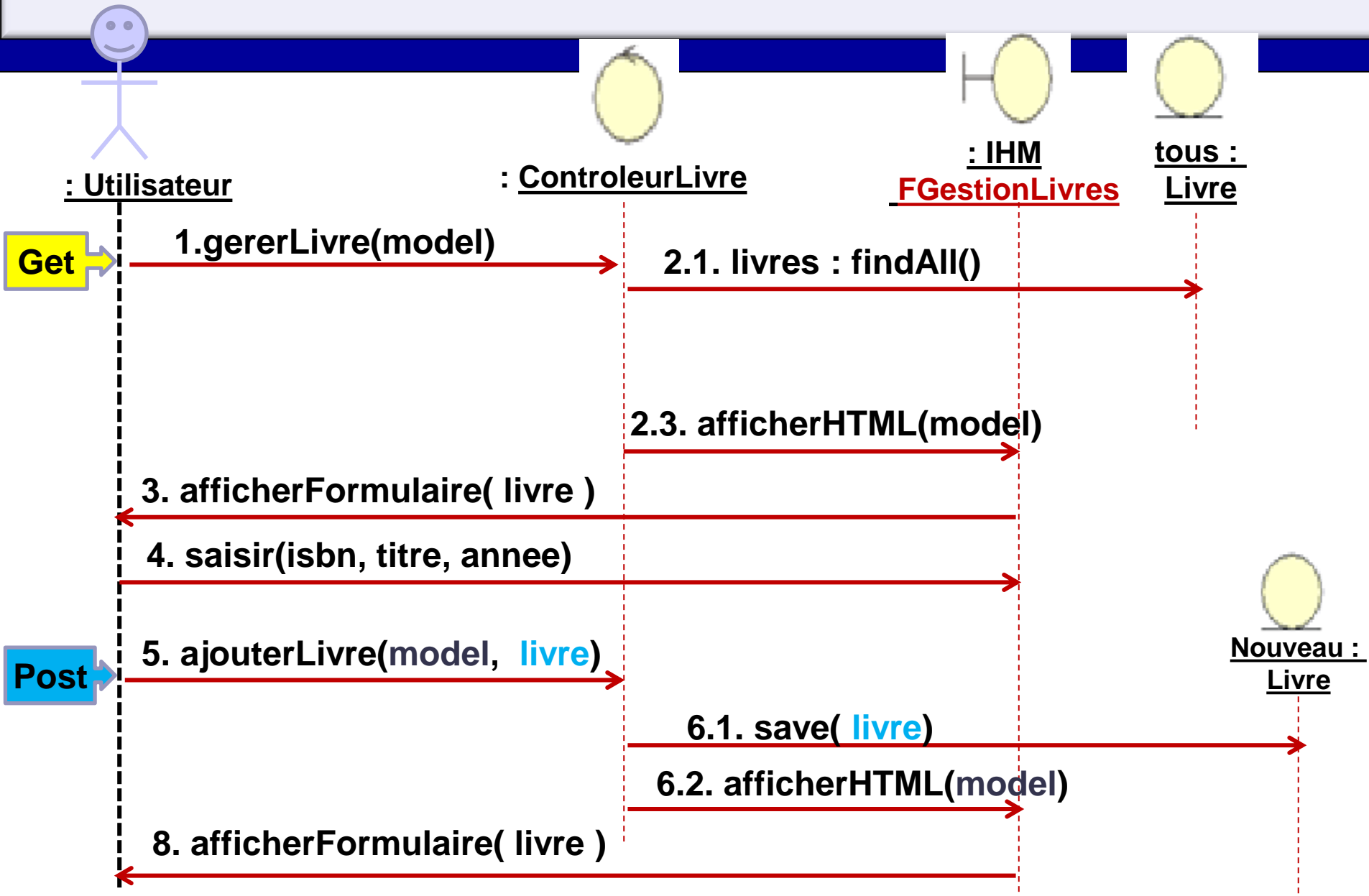
Nom de l'action  
dans le contrôleur

Nom du paramètre  
de l'action

Le click sur le bouton Chercher déclenche la requête  
**localhost:8080/chercherLivres?titre=**



# Création d'un formulaire



# Exemple de formulaire avec la méthode POST

```
<form method="post" action="ajouterLivre">
<table align="center">
<tr><td> ISBN : </td>
      <td><input type="text" name="isbn"/>
      </td></tr>
<tr><td><u>Titre</u></td>
      <td><input type="text" name="titre"></input></td>
      <td><input type="submit" value="Chercher"></input></td></tr>
<tr><td> Année :</td>
      <td><input type="number" name="annee"/></td></tr>
</table>
<div align=center><input type="submit" value="Ajouter"></input>
</div>
</form>
```

Nom de l'action dans  
le contrôleur

Noms des attributs du  
paramètre de l'action de  
type Livre

Le click sur le bouton **Ajouter** déclenche la requête  
**localhost:8080/ajouterLivre**

# IX. Quelques balises HTML

## formulaire

```
<form method="post" action="...">
```

## champ de saisie

```
<input type="text" name="txtSaisie" size="20" value="qqs mots" />
```

## champ de saisie cachee

```
<input type="password" name="txtMdp" size="20" value="unMotDePasse" />
```

## champ de saisie multilignes

```
<textarea rows="2" name="areaSaisie" cols="20">
```

```
ligne1
```

```
ligne2
```

```
ligne3
```

```
</textarea>
```

# Quelques balises HTML

## boutons radio

```
<input type="radio" value="Oui" name="R1" />Oui  
<input type="radio" value="non" name="R1" checked="checked" />Non
```

## cases a cocher

```
<input type="checkbox" name="C1" value="un" />1  
<input type="checkbox" name="C2" value="deux" checked="checked" />2  
<input type="checkbox" name="C3" value="trois" />3
```

## Combo

```
<select size="1" name="cmbValeurs">  
  <option value="1">choix1</option>  
  <option selected="selected" value="2">choix2</option>  
  <option value="3">choix3</option>  
</select>
```