

Le framework Spring

Module 4 – Spring Web

Objectifs

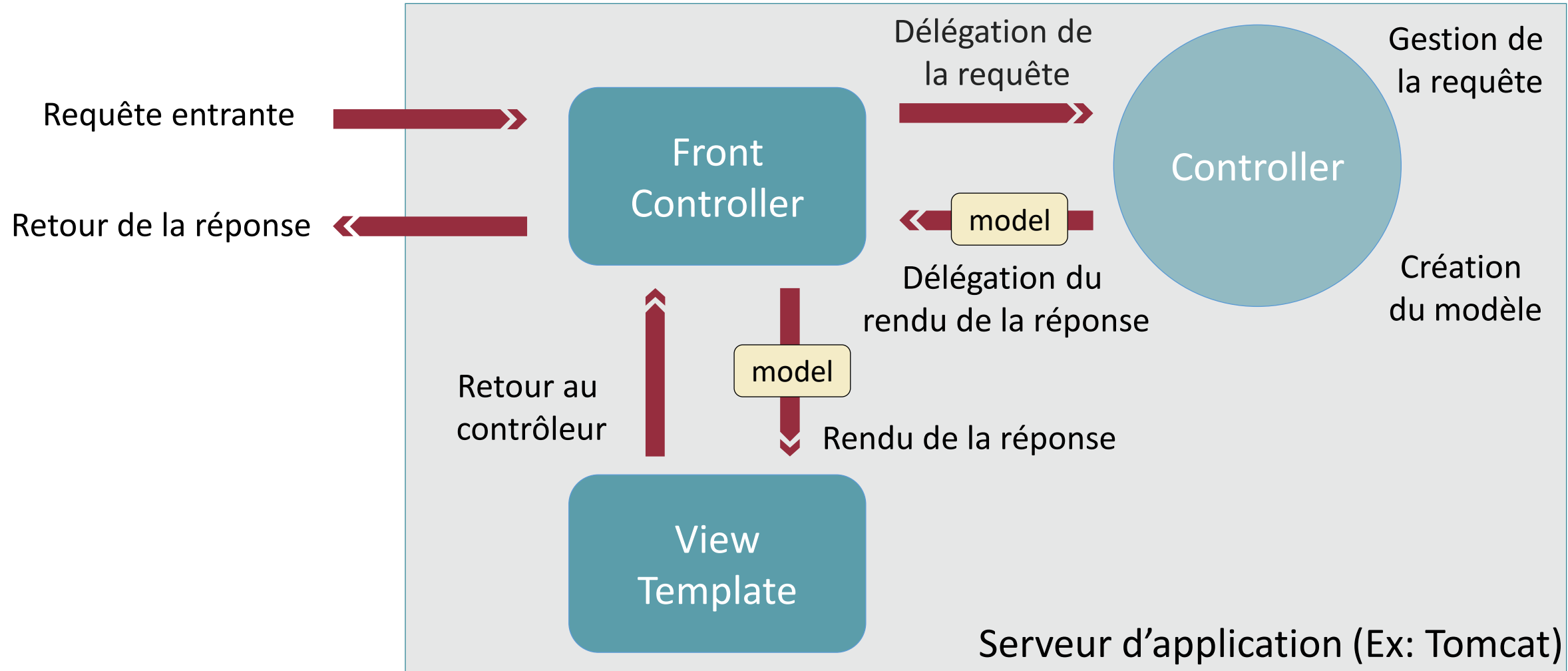
- Les bases
- Le moteur de template Thymeleaf
- Les contextes d'exécution
- Les formulaires
- La validation des données
- L'internationalisation

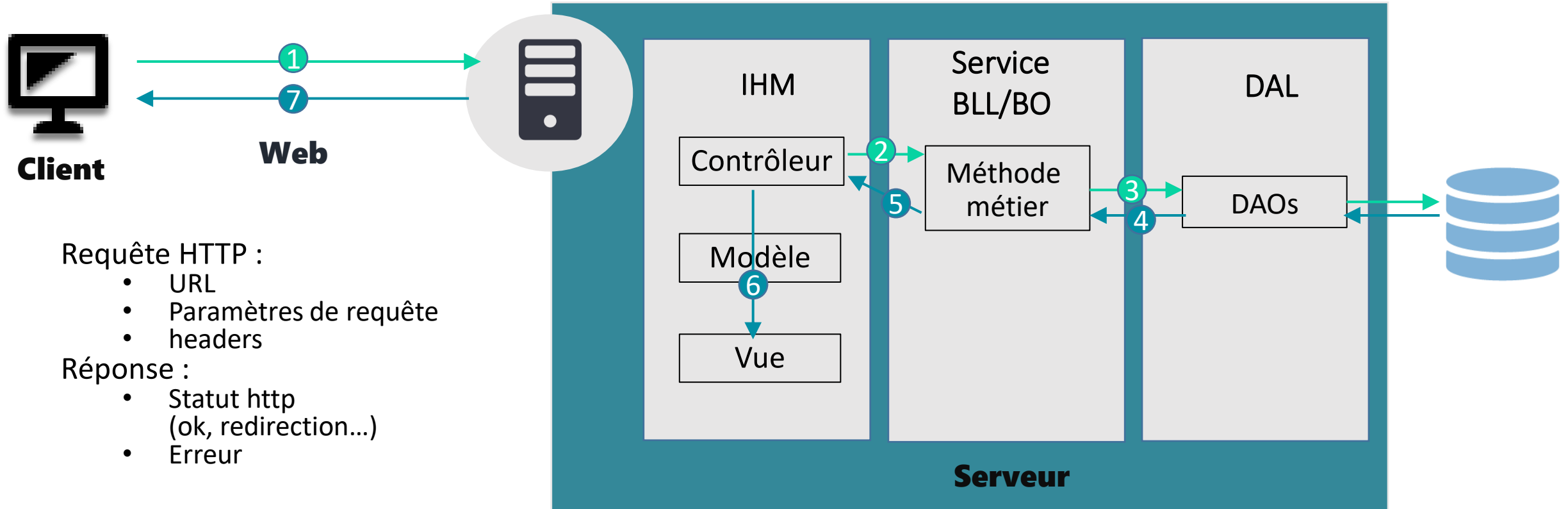
Spring Web – Les bases

Objectifs

- Le design pattern Modèle Vue Contrôleur et l'architecture Spring MVC
- Créer un projet Spring web et Thymeleaf avec Spring Boot
- Créer un contrôleur avec l'annotation @Controller
- Déclencher un traitement java depuis une URL
- Déléguer le traitement d'une requête à une vue
- Lire des paramètres et les variables http
- Utiliser le modèle (Model, @ModelAttribute)
- Gérer une redirection 302


- **MVC** est un design pattern destiné aux interfaces graphiques
- Il est composé de trois parties :
 - Le **contrôleur** qui contient la logique applicative et gère les actions utilisateurs
 - Le **modèle** qui contient les données d'affichage
 - La **vue** qui définit la structure et l'apparence de l'IHM





Ajout de Starters

- Pour le développement d'application Spring MVC :
 - spring-boot-starter-web
- Pour utiliser le moteur de template Thymeleaf :
 - Spring-boot-starter-thymeleaf
- Pour gérer le redéploiement automatique du serveur :
 - Spring-boot-devtools



```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  
    developmentOnly 'org.springframework.boot:spring-boot-devtools'  
}
```

build.gradle

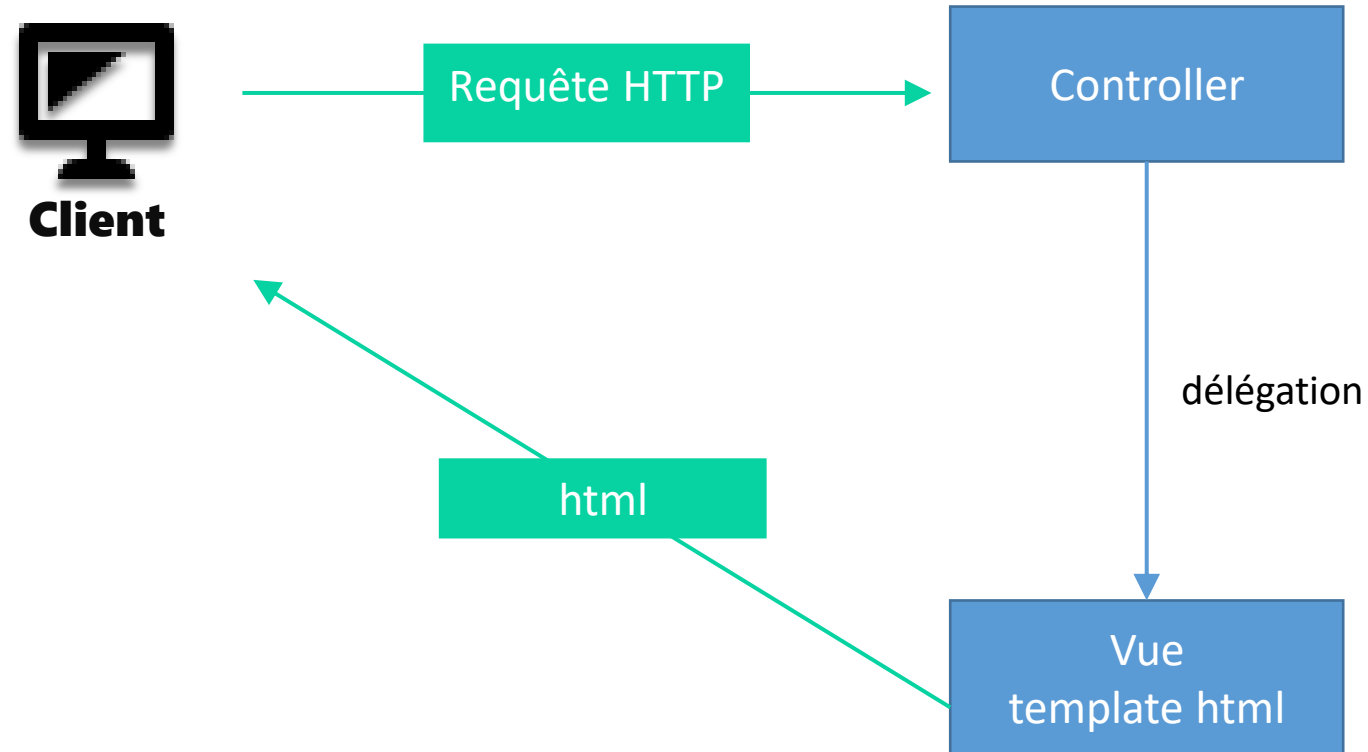

```
@SpringBootApplication  
@EnableAutoConfiguration  
@ComponentScan  
public @interface SpringBootApplication {
```

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

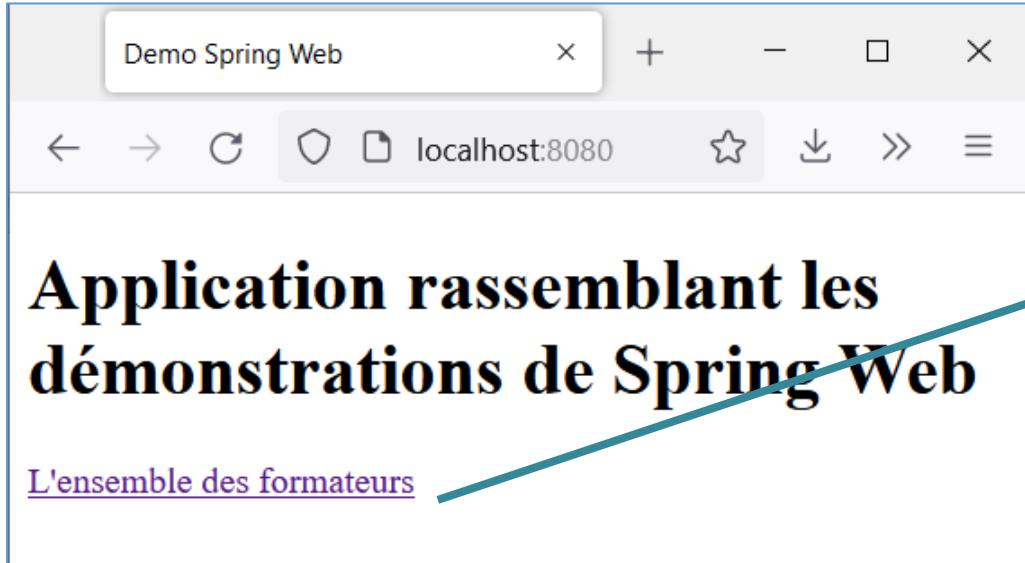
Exécutable directement sur
le tomcat intégré!

- C'est un Bean Spring
 - Défini avec l'annotation `@Controller`
- Sert à traiter les requêtes HTTP
- Permet l'interaction avec le modèle
- Délègue le traitement de la requête à une vue ou renvoie la réponse
- Les requêtes peuvent être mappées au niveau de la classe ou des méthodes
 - `@RequestMapping`
 - `@GetMapping`, `@PostMapping`, ...

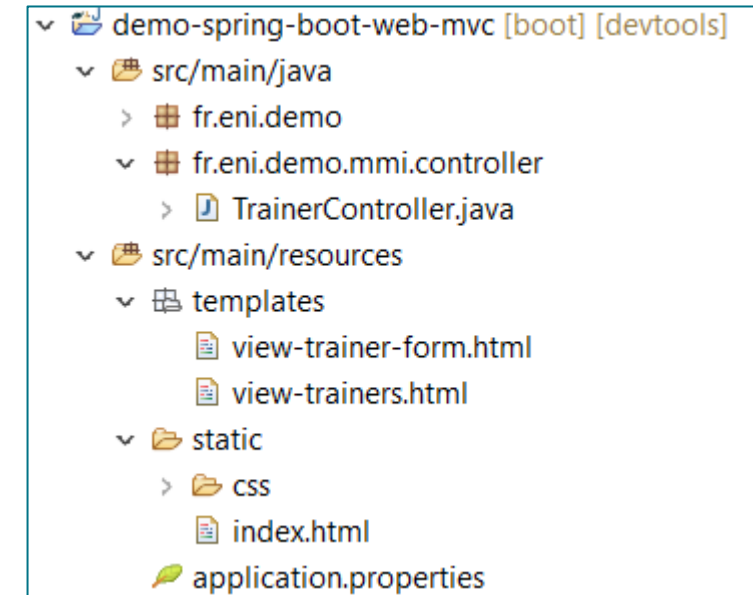
Déléguer le traitement de la requête à une vue



Déléguer le traitement d'une requête à une vue

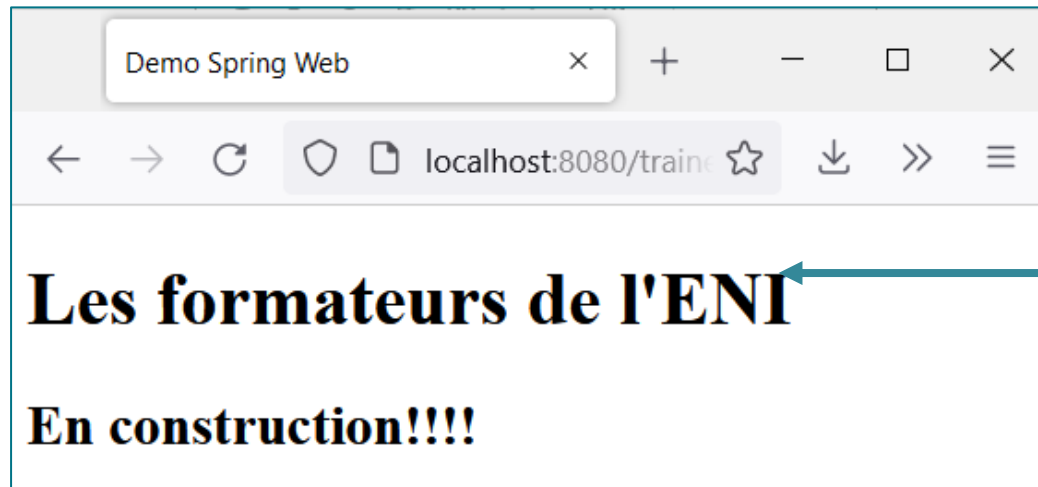


```
@Controller
public class TrainerController {
    @GetMapping("/trainers")
    public String allTrainers() {
        return « view-trainers»;
    }
}
```



view-trainers.html

```
<!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>
```





Le mapping des requêtes

- Spring permet d'utiliser 2 stratégies pour le passage de valeurs dans requête HTTP
 - Paramètres de requêtes HTTP (@RequestParam)
 - Variables de l'URI (@PathVariable)

- Comprendre la différence d'utilisation :

http://localhost:8080/trainers/10/trainings?tag=Spring

- 10 est une variable, récupérable avec @PathVariable
- tag=Spring est un paramètre, récupérable avec @RequestParam

- Le paramètre peut être :

- Dans l'URL directement, derrière le ?

```
<a href="trainers/detail?email=abaille@campus-eni.fr">
```

- Au travers d'un formulaire (attribut « name » des champs)

```
<input type="text" name="email"/>
```

- @RequestParam sur le paramètre de la méthode du contrôleur permet de le récupérer

```
public String detailTrainer(  
    @RequestParam(name = "email", required = false, defaultValue = "coach@campus-eni.fr") String emailTrainer) {
```

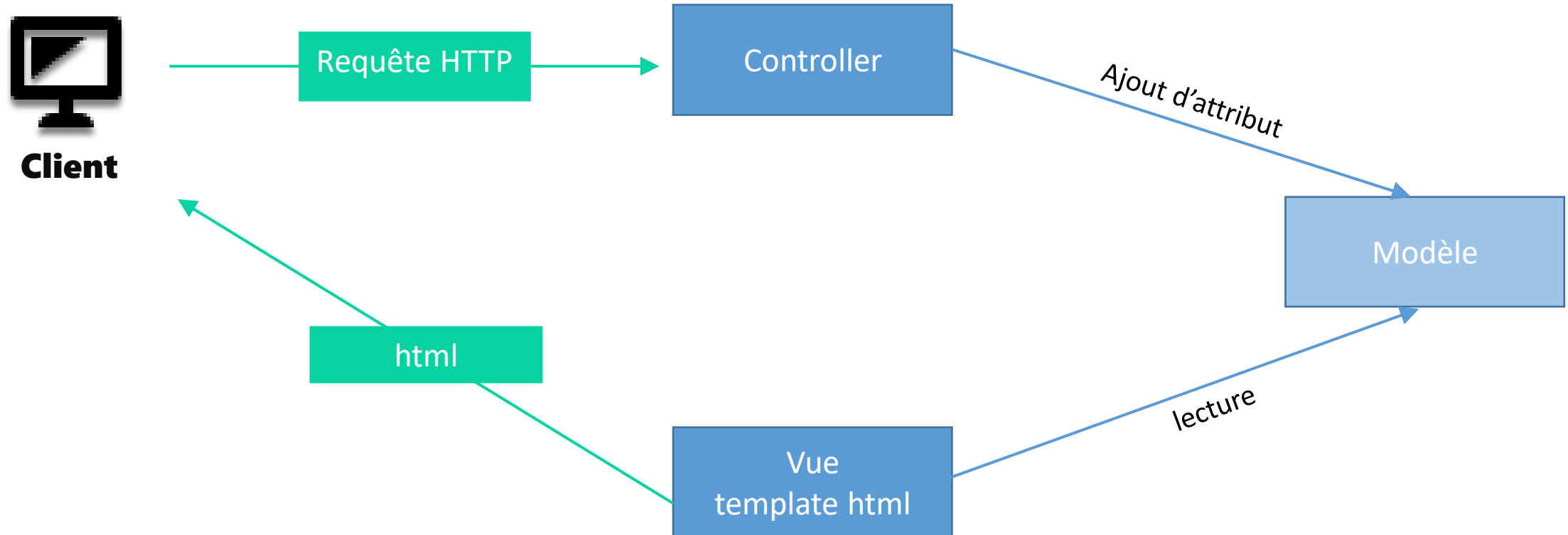
- Le membre d'annotation « required » permet d'indiquer si le paramètre http est optionnel ou non
 - Quand le paramètre est optionnel on peut définir une valeur par défaut

- La variable est déclarée dans l'URI directement
 - Sans encodage ``
- Côté contrôleur :
 - Définir la variable à l'emplacement attendu dans l'URL avec la syntaxe `{nomVariable}`
 - Injecter la valeur avec l'annotation `@PathVariable`

```
@GetMapping("/detail/variable/{email}")  
public String detailTrainer2(@PathVariable(name = "email") String emailTrainer) {
```



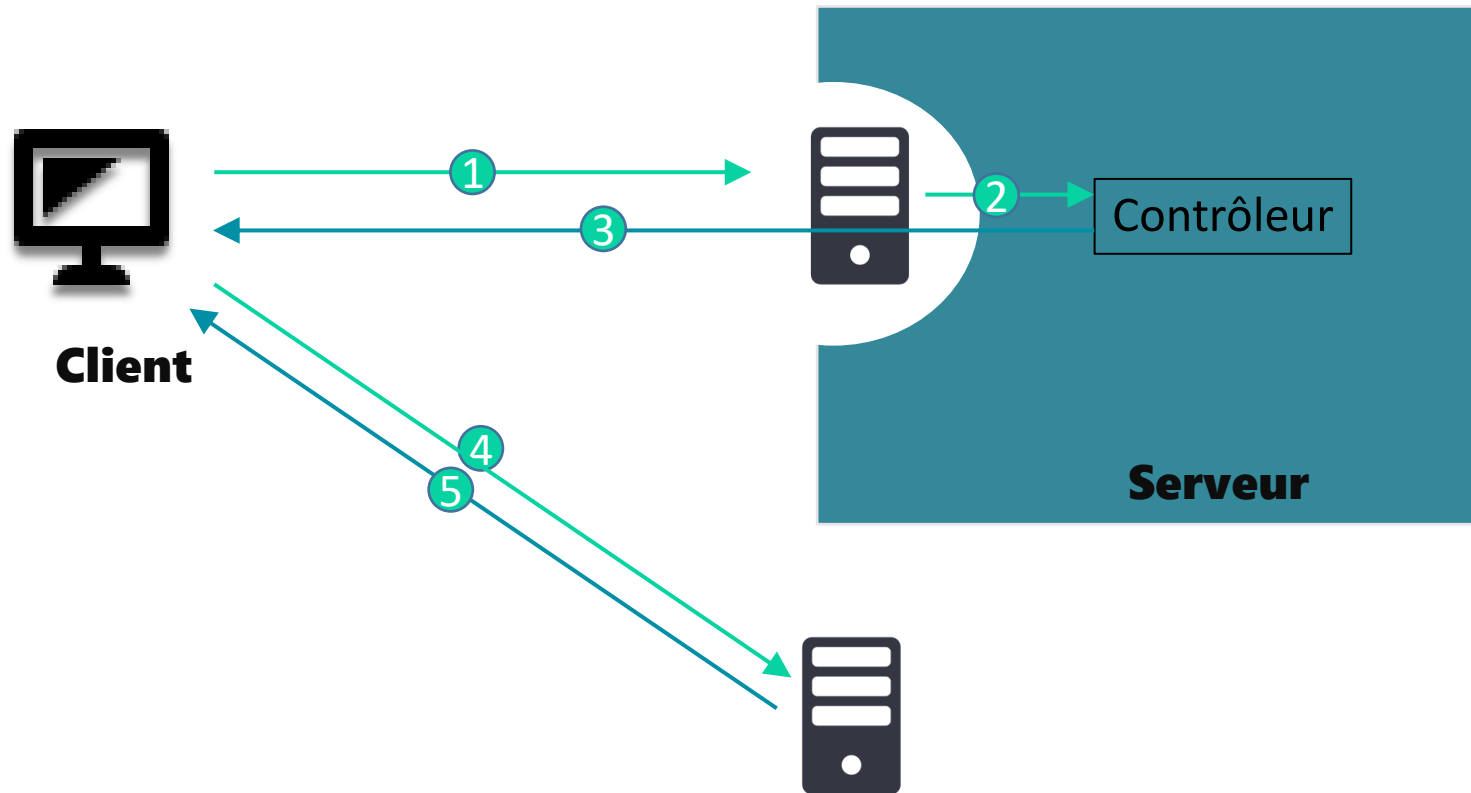

Les paramètres et variables http



- Côté contrôleur :
 - Injecter le modèle dans la méthode en ajoutant un paramètre de type `org.springframework.ui.Model`
 - Les données sont stockées dans le modèle sous forme d'attributs (clé - valeur)
- Côté vue :
 - Lire les données du modèle suivant la technologie de vue choisie
 - Nous utiliserons Thymeleaf



Utiliser le modèle



- Au lieu de déléguer le traitement d'une requête à une vue, le contrôleur peut renvoyer directement une réponse de type redirection (statut HTTP 302)
 - Change l'url et la requête HTTP courante sur le client
 - Dirige l'utilisateur à l'extérieur de la Webapp
 - Inconvénient : 2 allers-retours sont nécessaires pour afficher la page

- Pour faire une redirection, il faut que la méthode mappée renvoie le mot clé redirect: suivi de l'url cible
- Syntaxe : « redirect:/url_de_redirection »

```
return "redirect:/index.html";
```



Redirection

Introduction au moteur de template Thymeleaf

Objectifs

- Qu'est ce que Thymeleaf
- Comment intégrer Thymeleaf dans un projet
- Créer une vue Thymeleaf
- Intégrer les instructions de Thymeleaf

- Un moteur de modèle (template engine)
- Permet d'intégrer des données dans un modèle de document pour générer :
 - page web, email, fichier csv, pdf ou autre
- Adapté aux applications web :
 - Alternative aux JSP
 - Indépendant du framework Spring
 - Ne fait qu'ajouter ses propres attributs html, ce qui permet de visualiser le rendu d'une page sans exécuter l'application



www.thymeleaf.org

- Nécessite la dépendance Thymeleaf
- Pour Gradle ajouter la dépendance comme suit :

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  
    ...  
}
```

Comment utiliser Thymeleaf dans une vue

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
...
</head>
<body>
...
</body>
</html>
```

template.html

Il suffit d'ajouter le namespace Thymeleaf !

Le fichier est suffixé par l'extension .html
et doit être placé dans le dossier templates

- La plupart des instructions Thymeleaf sont ajoutées en tant qu'attribut dans les balises et utilisent le préfixe défini par le namespace (th)
 - Pour que les attributs soient valides pour HTML5 : il faut les préfixer par **data-** et utiliser le **séparateur « - »** au lieu du « : »
- Les instructions peuvent utiliser le Spring Expression language (Spel)
- Le résultat de l'expression est affecté au contenu de la balise
- Pour plus d'informations :
<https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html>

- Les expressions langages vont permettre de récupérer de l'information à l'extérieur de la vue et l'incorporer dans la page web générée :

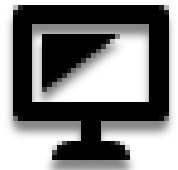
| Objectif | Syntaxe | Exemple(s) |
|--|------------------------|--|
| Expressions variables (accès aux données du modèle, beans...) | <code>\${ ... }</code> | <code>\${'nom : ' + personne.nom}</code> <code>\${@myBean.doSomething()}</code> |
| URL | <code>@{ ... }</code> | <code>@{/order/details(orderId=\${o.id})}</code> |
| Messages | <code>#{ ... }</code> | <code>#{home.welcome}</code> |
| Expressions de sélection – permet un accès direct aux membres d'un objet | <code>*{ ... }</code> | <code><body th:object="\${order}"></code> <code>99</code> (Est equivalent à <code>\${order.id}</code>) |



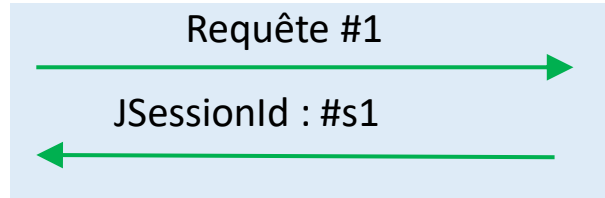
Thymeleaf

TP – Filmothèque - Partie 02

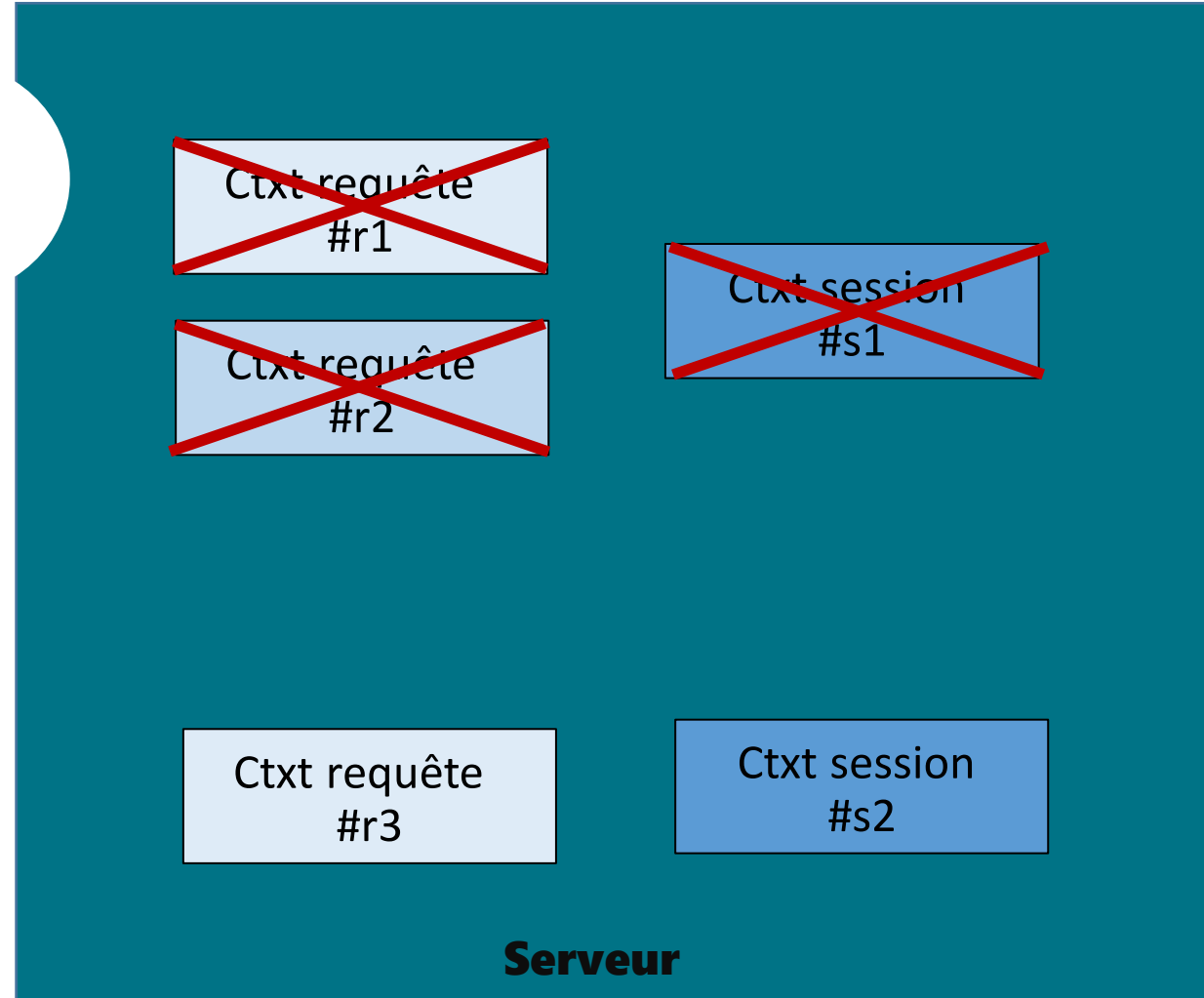
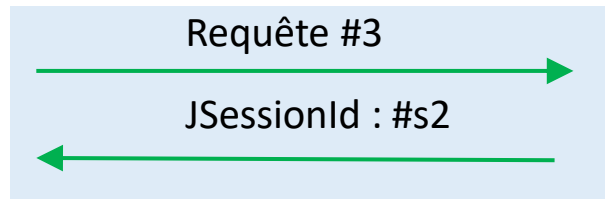
Contextes d'exécution



Client 1



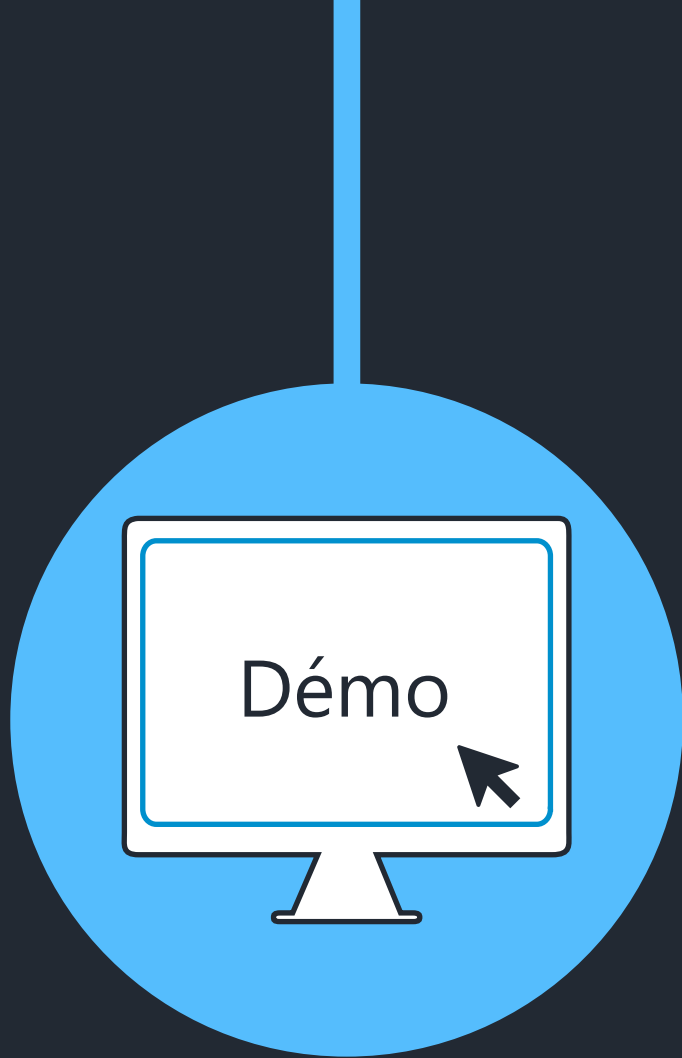
Client 2



- @SessionAttributes
 - Permet de définir une liste d'attribut qui doivent être gérés en session
 - Injecter cette liste sur tous les contrôleurs qui les affichent
- @SessionScope
 - Ajouté à la définition d'un bean, et dans le contexte d'une Webapp
- HttpSession
 - Utiliser la gestion des sessions de Http
- ATTENTION :
 - Spring charge le contexte en Session au chargement de la vue
 - HttpSession, ajoute immédiate l'attribut



Contexte de session par Spring



Différences entre HttpSession et @SessionAttributes

- Il est possible de manipuler le contexte de l'application
- ServletContext
 - Injecter dans le contrôleur
- @ApplicationScope
 - Ajouté à la définition d'un bean dans une classe de configuration

TP – Filmothèque – Partie 03

Formulaire

@ModelAttribute : 2 manières de manipuler le modèle

- 1) Porté par un paramètre de méthode mappée, @ModelAttribute permet d'injecter un unique attribut du modèle
 - Plutôt que tout le modèle (Model)
- 2) Porté par une méthode, il permet d'initialiser un attribut du modèle avant l'appel de la méthode mappée

- Gestion des données d'un formulaire directement à partir d'un objet
- Thymeleaf propose
 - l'instruction data-th-object pour référencer un objet du modèle
 - L'instruction data-th-field permet de faire référence aux membres de l'instance courante
- Côté contrôleur, il y a 2 choses à faire :
 - Initialiser les données du formulaire
 - Récupérer les données saisies par l'utilisateur



@ModelAttribute et Gestion
d'objet d'un formulaire

Les objets métiers ont souvent des associations entre eux.

- Côté contrôleur, cela impose un objet de formulaire plus complexe
- Côté vue, une liste de données à afficher en continue
- Exemple : Nos formateurs et leurs cours
 - Actuellement séparation de la mise à jour (attributs directs et association)
- Sans les Converter, il n'est pas possible de gérer le tout en un objet de formulaire
 - Ni d'afficher facilement la liste des données d'origine

Converter, permet de gérer l'affichage et l'injection de l'association

- Définir un bean de Spring qui implémente

`Converter<Class<?> sourceType, Class<?> targetType>`

- Redéfinir la méthode

`<T> T convert(Object source, Class<T> targetType);`

Spring l'appelle automatiquement

- Un attribut de Model qui correspond au type
- Ou sur un bean qui retourne des objets de ce type



Utiliser un Converter

TP – Filmothèque - Partie 04

Validation des données

- Pourquoi valider les données ?
 - Pour l'intégrité des données
 - Les données reçues doivent correspondre au format attendu pour permettre leur traitement et leur stockage
 - Pour la sécurité
 - Exemple : Commander un nombre d'article négatif pourrait permettre de créditer son solde bancaire !

Validation des données

Exemples de contraintes proposées

| Contrainte | Signification (sur l'élément) |
|-------------------|---|
| @AssertTrue | Doit être à true |
| @AssertFalse | Doit être à false |
| @Min, @DecimalMin | Doit être supérieur à ... |
| @Max, @DecimalMax | Doit être inférieur à ... |
| @Digits | Définit le nombre de chiffres |
| @Size | Doit être entre deux tailles |
| @Null | Doit être nul |
| @NotNull | Doit être non nul |
| @NotEmpty | Ne peut pas être nul ni vide |
| @NotBlank | Ne peut pas être nul ni vide après trim() |

| Contrainte | Signification (sur l'élément) |
|------------|-------------------------------|
| @Pattern | Doit respecter une RegExp |
| @Email | Possède le format email |
| @Past | Doit être dans le passé |
| @Future | Doit être dans le futur |

- Quelques contraintes fournies spécifiquement par Hibernate Validator :

| Contrainte | Signification (sur l'élément) |
|-------------------|---|
| @CreditCardNumber | Représente un numéro de carte de crédit |
| @URL | Représente une URL valide |
| @Range | Doit être dans l'intervalle |

- Ajouter le starter : spring-boot-starter-validation
- Placer les annotations sur les objets métiers
- Déclencher la validation des données dans le Contrôleur :
 - Appel de méthode validant le formulaire : `@Valid`
 - Ajout du paramètre de type `BindingResult`
 - Ce paramètre doit suivre le paramètre annoté `@Valid`
- Affichage des erreurs sur la vue
 - Soit dans un bloc général
 - Soit sur chaque champ

- Les messages d'erreurs sont déterminés comme suit :
 - Message d'erreur défini par défaut par l'annotation
 - Attribut de l'annotation
 - Exemple `@NotBlank(message = "Le prenom est obligatoire")`
 - Dans la vue et en relation avec les **fichiers propriétés**
- Le message le plus spécifique est pris en compte



Validation d'un formulaire

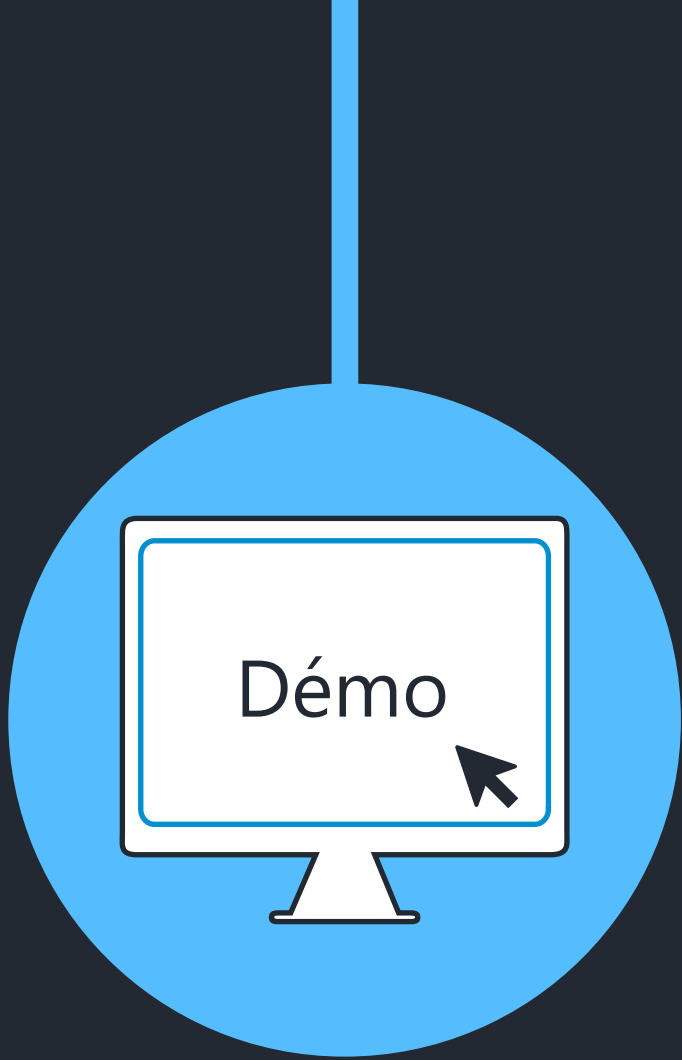
TP – Filmothèque - Partie 05

Internationalisation

Mise en place de l'internationalisation sur les messages

- Ajout de fichier de properties par langues
 - `messages.properties`, `messages_fr.properties`, ...
- Dans la configuration générale : `application.properties`
 - Ajout de l'encodage en UTF-8
 - `spring.messages.encoding=UTF-8`
- Au niveau des vues, utilisation de
 - `data-th-text="#{clef_dans_messages}"`

- Changement de langues :
 - Définition d'une classe de configuration implémentant `WebMvcConfigurer` et annotée `@Configuration`
 - Définir un bean `LocaleResolver` pour gérer la langue par défaut
 - Définir un bean `LocaleChangeInterceptor`; pour intercepter le paramètre de changement de langue (exemple : `language`)
 - Redéfinir la méthode `addInterceptors` pour enregistrer le bean précédent
 - Sur les vues, ajouter aux URL le paramètre de langue et la Locale de la langue (exemple : `?language=en`)



Internationalisation

TP – Filmothèque – Partie 06