

@ModelAttribute et Gestion d'objet d'un formulaire

Démonstration 8 du module 4

Les objectifs de cette démonstration sont

- Utilisation @ModelAttribute sur une méthode et un paramètre de méthode
- Utilisation d'une instance d'objet pour gérer les formulaires

Contexte

Compléter le projet précédent

- Le fichier de CSS : demo-form.css doit être mis à jour
- Ajouter

```
.flex-outer>li select {  
    flex: 1 0 220px;  
}
```

Déroulement

1. Injection de la liste des cours avec @ModelAttribute

- Supprimer l'appel de la méthode loadCourses dans la méthode detailTrainer de TrainerController.
- Supprimer la méthode loadCourses de la classe TrainerController
- Mettre en commentaire l'annotation @SessionAttributes
- La remplacer par la méthode suivante :

```
@ModelAttribute("allCourses")  
public List<Course> getCourses(){  
    System.out.println("getCourses");  
    return courseService.findAll();  
}
```

- Exécution de l'application

- La liste des cours est toujours présent à chaque fois qu'on revient sur le détail d'un formateur
- L'inconvénient, c'est que dès qu'il y a appel d'une méthode mappée du contrôleur, la méthode est elle-même appelée et donc rechargement de la liste des cours.
 - Pour éviter cela, il suffit de réactiver l'annotation `@SessionAttributes` du contrôleur. Spring appellera la méthode seulement si la donnée n'est pas en session
 - En utilisant l'annotation `@SessionAttributes` et la méthode annotée `@ModelAttribute`, il n'est plus nécessaire de tester si les données sont déjà en session. Spring le fait pour nous.

2. Utilisation d'un objet pour le formulaire de view-trainer-form.html

- Dans la méthode `detailTrainer`, nous injectons déjà un objet `Trainer` dans le modèle.
 - Donc côté contrôleur ; le travail est déjà fait.
- Côté vue, il faut modifier le formulaire en utilisant :
 - `data-th-object`
 - `data-th-field` à la place de `data-th-value` et `name`
 - `data-th-action` à la place d'action

```
<form data-th-action="@{/trainers}" data-th-object="${trainer}" method="post">
  <h1>Détail du formateur</h1>
  <ul class="flex-outer">
    <li><label for="inputFirstN">Prénom : </label>
      <input type="text" id="inputFirstN" required data-th-field="*{firstName}" />
    </li>
    <li><label for="inputLastN">Nom : </label>
      <input type="text" id="inputLastN" required data-th-field="*{lastName}" />
    </li>
    <li><label for="inputEmail">Email : </label>
      <input type="text" id="inputEmail" required data-th-field="*{email}" />
    </li>
    <li><label for="tableCourses">Cours : </label>
      <ul>
        <li data-th-each="current : ${trainer.lstCourses}" data-th-text="${current.title}"></li>
      </ul>
    </li>
    <li><button type="submit">Enregistrer</button></li>
  </ul>
</form>
```

- L'affichage du formulaire est opérationnelle avec ce nouveau code

3. Utilisation @ModelAttribute sur le post du formulaire

- Supprimer la méthode createOrUpdateTrainer de TrainerController
- Récupération dans le contrôleur sur le post du formulaire de l'objet « trainer »
 - Création de la méthode updateTrainer dans TrainerController
 - Spring lui injecte l'objet du formulaire (@ModelAttribute)
 - Elle utilise le service pour mettre à jour le formateur
 - Et redirige vers l'url « /trainers » pour recharger la vue des formateurs.

```
@PostMapping
public String updateTrainer(@ModelAttribute("trainer")Trainer trainer) {
    System.out.println(trainer) ;
    trainerService.update(trainer) ;
    return «redirect:/trainers»;
}
```

- Tout est opérationnel maintenant, il est possible de tester la mise à jour d'un formateur

Exécution :

- Modifier le nom ou le prénom d'un formateur :

- Enregistrer, la trace dans la console doit correspondre :

```
Trainer [firstName=Anne, lastName=Baille, email=abaille@campus-eni.fr]
```

- Puis dans la vue de tous les formateurs, les informations sont aussi à jour :

Index	Prénom	Nom	Action
0	Anne	Baille	Détail
1	Stéphane	Gobin	Détail
2	Julien	Trillard	Détail