Le modèle

Démonstration 3 du module 4

L'objectif de cette démonstration est d'utiliser le modèle de Spring web MVC

Contexte

- Intégration de la couche BO, BLL et DAL de la démonstration du module 2 « demo-spring-bean-application »
- Vous pouvez les récupérer dans le répertoire ressources si vous n'aviez pas réalisé la démonstration complètement

Compléter le projet précédent

- Copier les packages et classes :
  - fr.eni.demo.bo.Trainer
  - fr.eni.demo.dal.TrainerDAO
  - fr.eni.demo.dal.mock.TrainerDAOMock
  - fr.eni.demo.bll.TrainerService
  - fr.eni.demo.bll.TrainerServiceImpl
- Réactiver les annotations @Service, @Respository et @Autorwired

Déroulement

Ajouter un comportement dans la couche BLL

- Ajout du comportement métier qui permet de récupérer un formateur depuis son email qui est unique.

1. **Modification de l'interface TrainerService**

- Ajouter la méthode : Trainer findByEmail(String emailTrainer)

2. **Modification de la classe TrainerServiceImpl**

- Ajouter la méthode : Trainer findByEmail(String emailTrainer)

```
@Override
public Trainer findByEmail(String emailTrainer) {
	return trainerDAO.read(emailTrainer);
}
```

Création du modèle dans le contrôleur

1. **Injecter TrainerService dans le constructeur de la classe TrainerController**

- Création d'un attribut de type TrainerService
- Création du constructeur avec paramètre TrainerService
- Ajouter l'annotation @Autowired pour l'injection

```
…
import org.springframework.beans.factory.annotation.Autowired;
…
import fr.eni.demo.bll.TrainerService;
import fr.eni.demo.bo.Trainer;

//Injection du TrainerService
private TrainerService trainerService;

@Autowired
public TrainerController(TrainerService trainerService) {
	this.trainerService = trainerService;
}
```

2. **Compléter la méthode detailTrainer de la classe TrainerController**

- Spring permet d'injecter en paramètre des méthodes des contrôleurs tous les types de données nécessaires au traitement http.
- Dans la cas présent, il est nécessaire de se passer un objet de type org.springframework.ui.Model pour transmettre un modèle à la vue
- Dans la méthode, on peut alors ajouter au modèle une instance de la classe Trainer initialisée par la couche BLL.
- Le traitement de la requête est ensuite délégué à la vue du formulaire.

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

@GetMapping("/detail")
public String detailTrainer(
	@RequestParam(name = "email", required = false,
			defaultValue = "coach@campus-eni.fr") String emailTrainer, Model model) {
	System.out.println("Le paramètre - " + emailTrainer);
	Trainer trainer = trainerService.findByEmail(emailTrainer);
	// Ajout de l'instance dans le modèle
	model.addAttribute("trainer", trainer);
	return "view-trainer-form";
}
```

3. **Compléter le template view-trainer-form.html**

- Mise en place de l'utilisation du moteur de template Thymeleaf.
  - `<html xmlns:th="http://www.thymeleaf.org">`
- Nous le verrons plus en détail ; dans la suite du chapitre.
  - Utilisation de ses tags pour manipuler l'objet Trainer mis dans le modèle de Spring
- `data-th-value="${trainer.firstName}"` ;
  - *trainer* permet de récupérer l'attribut du modèle, le nom correspond à la clef dans

l'attribut

- Une fois l'objet récupéré, il est possible d'accéder à ses attributs : *trainer.firstName*
- Et data-th-value permet de préciser sur quel attribut HTML du composant, il faut injecter la donnée. Dans le cas présent sur l'attribut value du champ input.

- Voici le code de tout le template :

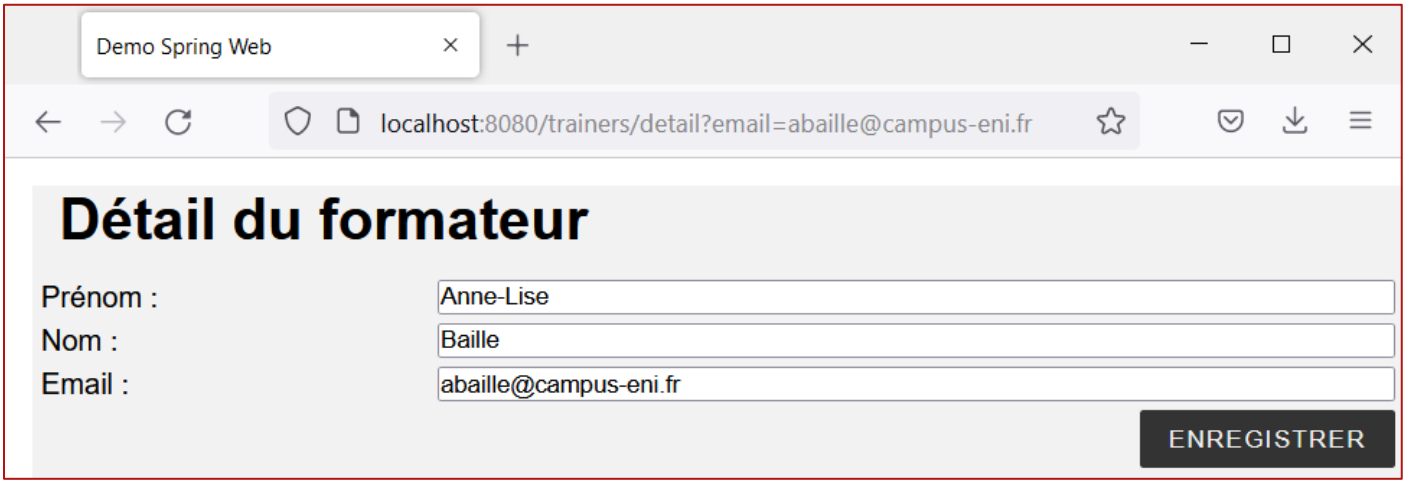
```
<!DOCTYPE html>
<!-- Ajout du moteur de template Thymeleaf -->
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Demo Spring Web</title>
<link rel="stylesheet" href="/css/demo-form.css">
</head>
<body>
	<form action="/trainers" method="post">
	<h1>Détail du formateur </h1>
		<ul class="flex-outer">
		<li>
			<label for="inputFirstN">Prénom : </label>
			<input type="text" name="firstName" id="inputFirstN" required
			data-th-value="${trainer.firstName}"/>
		</li>
		<li>
			<label for="inputLastN">Nom : </label>
			<input type="text" name="lastName" id="inputLastN" required
			data-th-value="${trainer.lastName}"/>
		</li>
		 <li>
			<label for="inputEmail">Email : </label>
			<input type="text" name="email" id="inputEmail" required
			data-th-value="${trainer.email}"/>
		</li>
		<li>
			<button type="submit">Enregistrer</button>
		</li>
		</ul>
	</form>
</body>
</html>
```

- Le fait d'avoir ajouté ce code, va permettre d'afficher les données du formateur sélectionné.

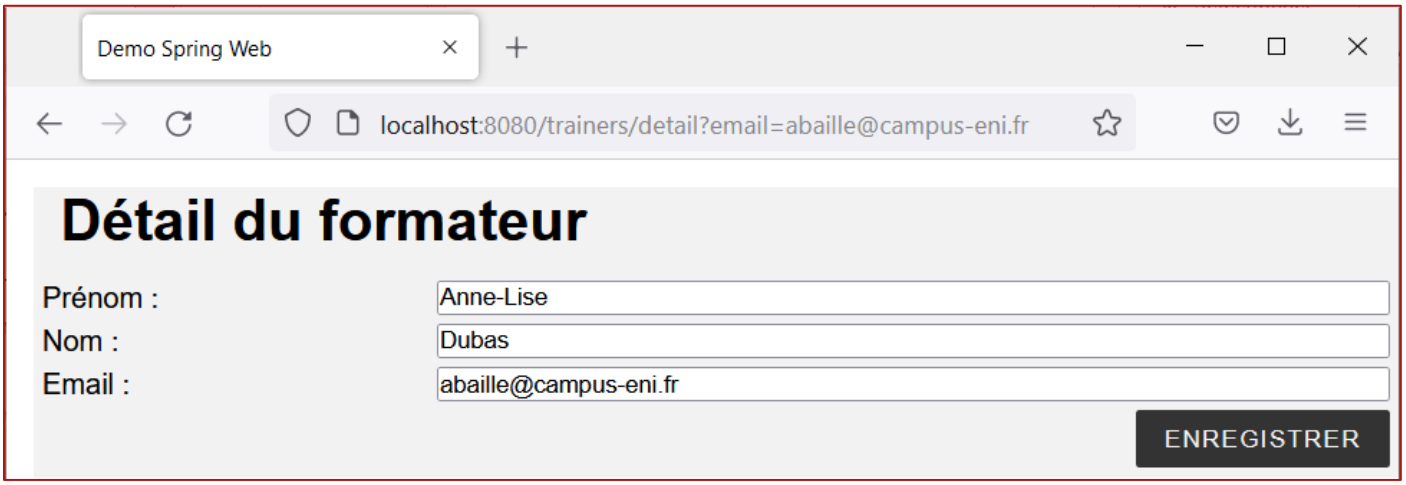
4. **Exécuter l'application, en cliquant sur le premier lien :**

```
<a href="trainers/detail?email=abaille@campus-eni.fr">Accéder au détail du formateur</a><br>
```

- voici la vue, avec les données du formateur affichées par défaut :



- Modifier le nom du formateur :



- Cliquer sur le bouton de soumission.
- Voici les traces de la console :

```
Les paramètres
Email - abaille@campus-eni.fr
FirstName - Anne-Lise
LastName - Dubas
```

- Les données sont toujours retournées vers le contrôleur au travers des paramètres de la requête.