TP : Formulaire de création d'événements avec Spring Boot et Thymeleaf

Table of Contents

1. Création d'un formulaire HTML

Créez une page `creation.html` pour permettre la création d'un événement.

Le formulaire doit contenir les champs suivants : - **nom** - **lieu** - **date**

Il doit également comporter un bouton de validation.

L'action liée au formulaire doit être `/mesevenements/creation`.

Pour cela, utilisez l'attribut `th:action` comme suit :

```
<form th:action="@{/mesevenements/creation}" method="post">
```

L'action du formulaire doit utiliser la méthode `POST`.

Modifiez l'annotation de la méthode `ajouter` dans le contrôleur :

```
@PostMapping("/creation")
public String ajouter(
        @RequestParam String nom,
        @RequestParam String lieu,
        @RequestParam String date,
        Model model
) {
    int identifiant = gestion.enregistrer(nom, lieu, date);
    model.addAttribute("message", "Événement ajouté avec succès !");
    model.addAttribute("identifiant", identifiant);
    return "resume";
```

```
}
```

Commentez temporairement la méthode `ajouterViaModel` dans le contrôleur.

Ajoutez une méthode `gotoCreation` dans le contrôleur, qui renvoie vers la page `creation.html`. Elle doit être annotée avec `@GetMapping("/creationForm")`.

Lancez l'application, puis accédez à l'URL suivante dans votre navigateur : http://localhost:8080/mesevenements/creationForm

Testez ensuite la création d'un événement via le formulaire et le bouton de validation.

Bravo ! Vous venez de créer un formulaire de création d'événement avec Spring Boot et Thymeleaf.

2. Passage à l'utilisation d'un DTO

Commentez la méthode `ajouter` du contrôleur.

Décommentez la méthode `ajouterViaModel`, et annotez-la avec `@PostMapping("/creation")`.

Il est impossible d'avoir deux méthodes avec exactement la même annotation `@PostMapping` et le même chemin dans un même contrôleur.

Modifiez l'entête du formulaire dans `creation.html` pour qu'il utilise la méthode `POST` et l'action `/mesevenements/creation` en ajoutant l'attribut `th:object="evenementDTO"`. Notez que l'attribut `th:object` est utilisé pour lier le formulaire à un objet de type `EvenementDTO` que nous retrouvons dans le contrôleur dans la fonction ajouterViaModel (`@ModelAttribute @ModelAttribute EvenementDTO evenement`). En résumé, le formulaire va remplir un DTO `EvenementDTO` avec les valeurs saisies dans le formulaire. Mais pour ceci il a besoin de savoir quel objet il doit remplir.

Nous allons donc ajouter une redirection vers la page de création d'événement dans le contrôleur de la façon suivante :

```
@GetMapping("/creationform")
  public String gotoCreationForm(Model model) {
      model.addAttribute("evenementDTO", new EvenementDTO());
      return "creation";
  }
```

Notez ici que nous ajoutons un objet `EvenementDTO` vide au modèle, ce qui permet à Thymeleaf de lier les champs du formulaire à cet objet.

Relancez l'application, puis testez à nouveau la création d'un événement.

Cette nouvelle version permet de mieux gérer les erreurs, ce qui sera traité dans l'exercice suivant.

3. Validation des données avec un DTO

Ajoutez la dépendance suivante dans votre fichier `pom.xml` :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Cette dépendance vous permet d'utiliser des annotations de validation dans vos classes métier (ou DTO).

Dans la méthode `ajouterViaModel`, ajoutez l'annotation `@Valid` juste avant `@ModelAttribute`.

```
public String ajouterViaModel(@Valid @ModelAttribute EvenementDTO dto, BindingResult
result)
```

Lors de l'utilisation de la validation, soit vous nommez la variable associée au formulaire `@ModelAttribute EvenementDTO evenementDTO`, soit vous utilisez l'annotation `@ModelAttribute("evenementDTO") EvenementDTO nomQuelconque` dans le contrôleur. Le risque est que si vous utilisez un nom différent non précisé par l'annotation, Thymeleaf perd la référence aux erreurs stockées dans le `BindingResult`.

Cela permet d'activer la validation automatique des contraintes définies dans le DTO.

Dans la classe `EvenementDTO`, ajoutez les annotations de validation suivantes :

- Le champ **nom** doit être renseigné (`@NotBlank`) et ne doit pas dépasser 50 caractères (`@Size(max = 50)`).
- Le champ **lieu** doit respecter les mêmes contraintes.
- Le champ **date** doit être renseigné et ne doit pas être vide.

Il n'existe pas d'annotation standard pour valider qu'une date respecte le format `yyyy-MM-dd` (ISO-8601). Une annotation personnalisée avec un validateur est nécessaire pour cela, mais ne sera pas abordée ici.

Pour plus d'informations sur les annotations de validation, vous pouvez consulter la documentation officielle : https://beanvalidation.org/2.0/spec/#builtinconstraints

Chaque annotation peut être configurée avec un attribut `message` pour personnaliser le message d'erreur.

Testez à nouveau la création d'un événement via le formulaire.

L'annotation `@Valid @ModelAttribute` permet de déclencher la validation. Si des erreurs sont détectées, elles sont stockées dans un objet de type `BindingResult`, qui doit être placé immédiatement après le paramètre annoté dans la signature de la méthode.

```
public String ajouterViaModel(@Valid @ModelAttribute EvenementDTO evenementDTO,
BindingResult result)
```

Pour vérifier s'il y a des erreurs, utilisez la méthode `hasErrors()` sur l'objet `BindingResult`.

Si des erreurs sont présentes, retournez la vue `creation`.

Sinon, enregistrez l'événement et retournez la vue `resume`.

4. Affichage des erreurs dans la vue

Ajoutez dans la vue `creation.html` le bloc suivant pour afficher les messages d'erreur de validation :

```
<div th:if="${#fields.hasErrors()}" class="error-box">
    <p><strong>Des erreurs ont été détectées :</strong></p>
    <ul>
        <!-- Erreurs globales -->
        <li th:each="err : ${#fields.globalErrors()}" th:text="${err}"></li>

        <!-- Erreurs de champ -->
        <li th:each="err : ${#fields.allErrors()}" th:text="${err.defaultMessage}"
></li>
    </ul>
</div>
```

Ajoutez également le style suivant dans la page pour mettre les erreurs en évidence :

```
<style>
    .error-box {
        border: 1px solid red;
        background-color: #fdd;
        color: darkred;
        padding: 10px;
        margin-bottom: 15px;
    }
</style>
```

Testez à nouveau la création d'un événement en remplissant des champs invalides (vides ou trop longs, par exemple).

La magie de la validation s'opère : les erreurs sont détectées, retournées à la vue,

> et affichées automatiquement.