



Framework Spring

Couche contrôleur :

JSP et Servlets

TP n°3

—

LP DAWIN | Module JEE

Année 2022 – 2023

H. Berger | M.A. Tessier

Sommaire

SOMMAIRE.....	0
1 INTRODUCTION.....	0
2 AJOUTER LES LIENS SUR LES VETERINAIRES.....	1
2.1 EXERCICE.....	1
2.2 EXPLICATIONS :.....	2
3 AFFICHER LE DETAIL D'UN VETERINAIRE	3
3.1 EXPLICATIONS :.....	3
3.1.1 JSP ownerDetails.jsp et contrôleur ownerController	3
3.2 EXERCICE.....	4
3.3 EXPLICATIONS	6
3.3.1 JSP vetDetails.jsp.....	6
4 AJOUTER UN MEMO A UN VETERINAIRE.....	7
4.1 EXERCICE.....	7
4.2 EXPLICATIONS :.....	12
4.2.1 Classe MemoController	12
4.2.2 JSP createOrUpdateMemoForm.jsp	13
4.2.3 Classe MemoValidator.java.....	13
5 GERER LES OPERATIONS.....	14
5.1 EXERCICE.....	14

1 Introduction

Lors du second TP, la couche d'accès aux données (**DAO / repository**) et la couche de service ont été étudiées.

Nous allons étudier dans ce TP la couche Front dite de contrôleurs et de présentation.

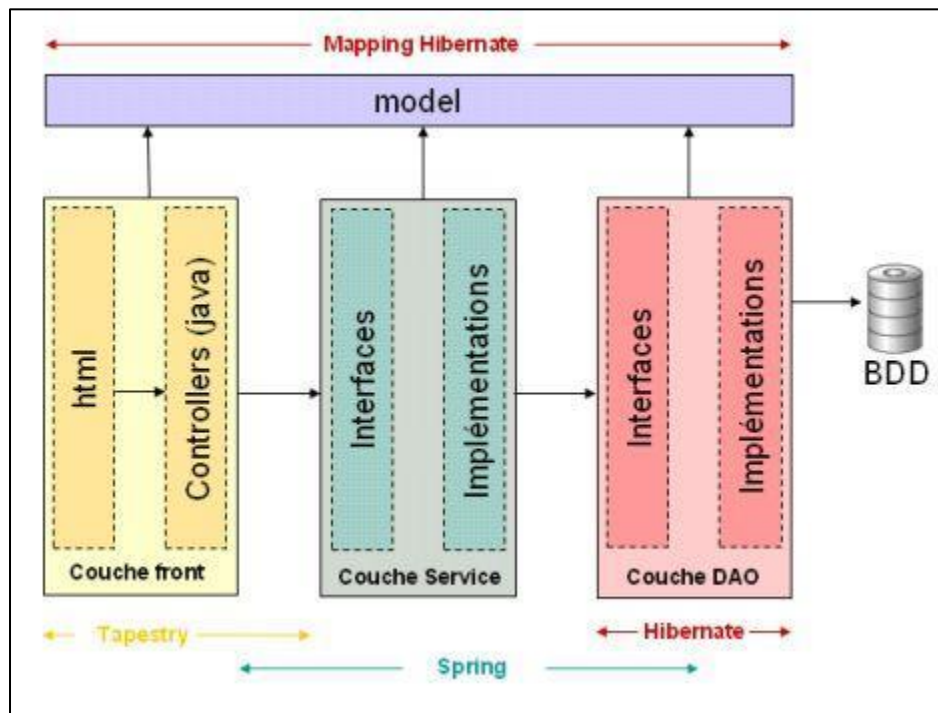
Cette couche est celle de plus haut niveau qui expose les informations au navigateur de l'utilisateur via des requêtes http.

La couche Front est composée de contrôleurs (basé sur des objets **servlet** java) et des vues : templates au format **JSP (Java Server Pages)**.

Les JSP sont définies par convention dans le répertoire WEB-INF (**src/main/webapp/WEB-INF**).


Les contrôleurs sont dans le package web (dans **src/main/java/org/springframework/samples/petclinic**).

Nous allons étudier dans ce TP ces 2 couches de l'architecture type du framework Spring :



Architecture en couche typique Spring

2 Ajouter les liens sur les vétérinaires



Name	Specialties
Helen Leary	radiology
Henry Stevens	radiology
James Carter	none
Linda Douglas	dentistry surgery
Rafael Ortega	surgery
Sharon Jenkins	none

Page de liste des vétérinaires

Voici les modifications nécessaires pour faire apparaître des liens sur la liste des vétérinaires.

On peut s'inspirer de la liste des propriétaires qui contient déjà des liens.

2.1 Exercice

Aller dans la JSP **vetList.jsp** (**src/main/webapp/WEB-INF/jsp/vets**)

Remplacer les lignes suivantes :

```
<td>
    <c:out value="${vet.firstName} ${vet.lastName}"/>
</td>
```

Par

```
<td>
    <spring:url value="/vets/{vetId}.html" var="vetUrl">
        <spring:param name="vetId" value="${vet.id}"/>
    </spring:url>
    <a href="${fn:escapeXml(vetUrl)}">
        <c:out value="${vet.firstName} ${vet.lastName}"/>
    </a>
</td>
```

Et ajouter le taglib suivant :

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

Compiler et exécuter pour tester.
Vérifier que les liens apparaissent bien sur les vétérinaires.
Utilisez le mode développeur de votre navigateur, inspectez l'élément HTML généré et faites le lien entre le modèle JSP et le code HTML généré.

2.2 Explications :

L'annotation **<spring:url ...>** a permis d'ajouter un lien sur chaque vétérinaire.

Ce lien pointe vers l'url « **/vets/{vetId}.html** » qui n'est pas gérée pour l'instant dans le contrôleur **VetController**.

Allez voir le code dans ce contrôleur **VetController**.

L'annotation **@Controller** a été ajoutée au-dessus du nom de la classe pour indiquer au framework qu'il s'agit d'un contrôleur.

Pour l'instant seule l'url **/vets** est mappée par ce contrôleur, voir l'annotation **@GetMapping(value = { "/vets.html" })**

C'est l'url qui permet d'obtenir la liste complète des Vétérinaires.

Cette servlet contrôleur contient un attribut **ClinicService** qui sera « *auto-injecté* » par le conteneur.

Les servlets contrôleur s'appuient sur la couche de services pour réaliser leurs traitements (c'est-à-dire recevoir et traiter une requête http et préparer une réponse http).

L'annotation **@Autowired** sur le constructeur de **VetController** indique que l'objet **ClinicService** est injecté par le framework Spring lors de l'instanciation du contrôleur.

C'est le framework qui gère le cycle de vie des objets

```
private final ClinicService clinicService;

@Autowired
public VetController(ClinicService clinicService) {
    this.clinicService = clinicService;
}
```

Observez la méthode **showVetList()** qui est lancée lorsque l'url **/vets** est mappée :

- Cette méthode crée un objet **Vets** (dans le modèle, classe plurielle de Vet qui contient une liste de **Vet**)
- Elle appelle la méthode **findVets()** du service **clinicService** (qui est « *auto injecté* »).
- Le résultat est stocké dans l'objet **vets**

- Cet objet **vets** est stocké dans un Model (objet utilisé par la JSP pour récupérer les données nécessaires à la page).
- Puis la méthode **showVetList** retourne la vue qui doit être renvoyée :
return "vets/vetList";
- Ceci a pour effet de lancer la page vetList.jsp.

3 Afficher le détail d'un vétérinaire

Nous allons maintenant ajouter le code nécessaire afin de rendre les liens des vétérinaires fonctionnels.

3.1 Explications :

De même que les liens des propriétaires (Owners) permettent d'afficher le détail de chaque propriétaire avec une JSP **ownerDetails.jsp**, nous allons faire la même chose pour les vétérinaires.

3.1.1 JSP ownerDetails.jsp et contrôleur ownerController

Allez observer **ownerDetails.jsp** (dans **src/main/webapp/WEB-INF/owners**) et **OwnerController.java** (dans le package web).

Observez la méthode suivante dans le contrôleur **ownerController** :

```
@GetMapping("/owners/{ownerId}")
public ModelAndView showOwner(@PathVariable("ownerId") int ownerId)
```

Cette méthode est appelée lorsque l'url **"/owners/{ownerId}"** est demandée (sur les liens des propriétaires dans **ownersList.jsp** obtenue à partir de l'onglet « **find owners** » dans le menu de l'application).

La méthode **showOwner()** prend en paramètre la variable **ownerId** stockée dans l'url et retourne un objet de type **ModelAndView**.

Les objets **ModelAndView** sont des objets propres au framework Spring et dédiés pour la couche controller et contiennent à la fois les informations nécessaires à la page JSP (objet modèle) et l'url de la vue à afficher.

La méthode **findOwnerById()** est appelée sur l'objet **clinicService** (« auto-injecté »).

L'objet **owner** du modèle retourné par la méthode **findOwnerById()** est stocké dans l'objet **ModelAndView mav** retourné ainsi que l'url **"owners/ownerDetails"**.

C'est donc la page **ownerDetails.jsp** qui est appelée lorsque l'utilisateur clique sur un lien de propriétaire.

Nous allons implémenter le même comportement pour les vétérinaires.

3.2 Exercice

Dans **vetController** (dans **src/main/java/org/springframework/samples/petclinic/web**), ajouter la méthode suivante :

```
/**
 * Custom handler for displaying a vet.
 *
 * @param vetId the ID of the vet to display
 * @return a ModelAndView with the model attributes for the view
 */
@GetMapping("/vets/{vetId}")
public ModelAndView showVet(@PathVariable("vetId") int vetId) {
    ModelAndView mav = new ModelAndView("vets/vetDetails");
    mav.addObject(this.clinicService.findVetById(vetId));
    return mav;
}
```

Ajouter si nécessaire les imports suivants :

```
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.servlet.ModelAndView;
```

Observez le code de la méthode ci-dessus **showVet()** : il est très similaire à celui de **showOwner()** ci-dessus. Cette méthode est lancée lorsque l'url **"/vets/{vetId}"** est appelée (sur les liens des vétérinaires que nous avons ajoutés).

Elle retournera un objet **ModelAndView** contenant l'objet **vet** trouvé par la méthode **findVetById()** et l'url de la page JSP **vetDetails.jsp** qui va s'afficher.

Allez observer la méthode **findVetById()** de la classe **ClicServiceImpl** : elle fait appel à la méthode **findById()** de la classe **JpaVetRepositoryImpl**.

Observez la requête : c'est un objet **vet** qui va être récupéré mais pas seulement, le **fetch** permet de récupérer tous les objets Memos qui lui sont rattachés :

```
//class JpaVetRepositoryImpl.java
@Override
public Vet findById(int id) {
    // using 'join fetch' because a single query should load both owners and pets
    // using 'left join fetch' because it might happen that an owner does not have
    pets yet
    Query query = this.em.createQuery("SELECT vet FROM Vet vet left join fetch
    vet.memos WHERE vet.id =:id");
    query.setParameter("id", id);
    return (Vet) query.getSingleResult();
}
```

Maintenant ajouter la JSP **vetDetails.jsp** dans **src/main/webapp/WEB-INF/vets**.

JSP vetDetails :

```

<%@ page session="false" trimDirectiveWhitespaces="true" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="petclinic" tagdir="/WEB-INF/tags" %>

<petclinic:layout pageName="vets">

    <h2>Vet Information</h2>

    <table class="table table-striped">
        <tr>
            <th>Name</th>
            <td><b><c:out value="${vet.firstName} ${vet.lastName}"/></b></td>
        </tr>
    </table>

    <spring:url value="{vetId}/memos/new.html" var="addUrl">
        <spring:param name="vetId" value="{vet.id}"/>
    </spring:url>
    <a href="{fn:escapeXml(addUrl)}" class="btn btn-default">Add New Memo</a>

    <br/>
    <br/>
    <br/>
    <h2>Memos</h2>

    <table class="table table-striped">

        <tr>
            <td valign="top">
                <table class="table-condensed">
                    <thead>
                        <tr>
                            <th>Memo Date</th>
                            <th>Description</th>
                        </tr>
                    </thead>
                    <c:forEach var="memo" items="${vet.memos}">
                        <tr>
                            <td><petclinic:localDate date="{memo.date}"
pattern="yyyy-MM-dd"/></td>
                            <td><c:out value="{memo.description}"/></td>
                        </tr>
                    </c:forEach>
                </table>
            </td>
        </tr>
    </table>

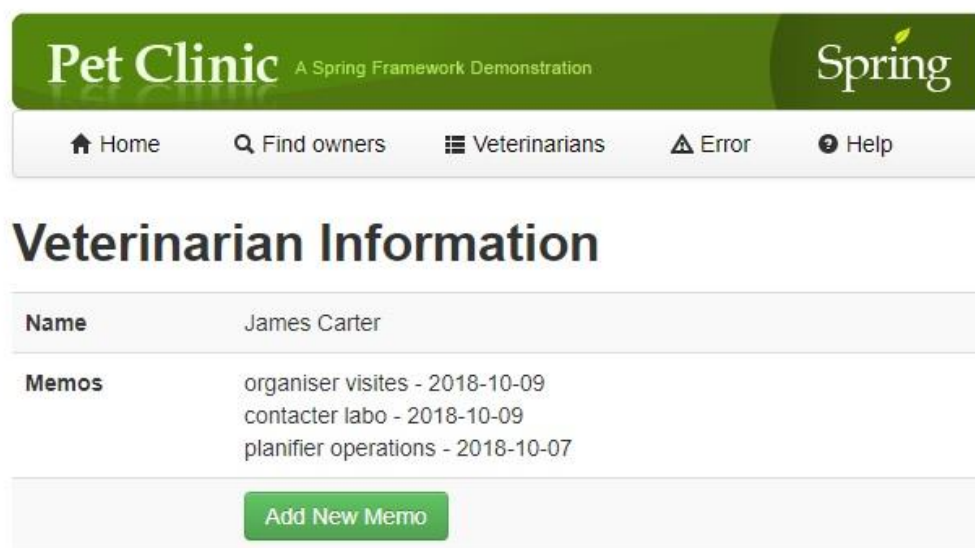
```



```
</table>
</petclinic:layout>
```

Compiler et exécuter pour tester.

Vérifier que les liens des vétérinaires permettent bien d'afficher le détail de chaque vétérinaire, avec leur nom, leur prénom et la liste de leurs mémos.



Page de détails d'un vétérinaire incluant la liste des mémos

Si vous cliquez sur le bouton « **Add New Memo** », vous obtenez une page erreur car l'ajout de Memo n'est pas encore géré.

3.3 Explications

3.3.1 JSP vetDetails.jsp

Observez le code de la JSP **vetDetails.jsp**.

Elle affiche dans un tableau html, sur la 1^{ère} ligne de ce tableau, les noms et prénoms du vétérinaire puis dans un 2^{ème} tableau la liste de ses mémos.

Les balises JSTL **<c:out ...>** (pour afficher un élément) et **<c:forEach ...>** (pour faire une boucle de parcours **foreach** des memos de l'objet **vet**) sont utilisées.

L'objet **vet** du modèle qui avait été récupéré par le contrôleur (et stocké dans un **ModelAndView**) est utilisé. Il est possible d'accéder directement à ses différents attributs (**\${vet.firstName}**, **\${vet.lastName}**, **\${vet.memos}**).

La date de chaque objet memo s'affiche en tant que chaîne de caractère dans un format spécifié via un pattern (**<joda:format value="{memo.date}" pattern="yyyy-MM-dd"/>**)

```
<table class="table-condensed">
  <thead>
    <tr>
      <th>Memo Date</th>
      <th>Description</th>
    </tr>
  </thead>
  <c:forEach var="memo" items="{vet.memos}">
    <tr>
      <td><joda:format date="{memo.date}" pattern="yyyy-MM-dd"/></td>
      <td><c:out value="{memo.description}"/></td>
    </tr>
  </c:forEach>
</table>
```

Un bouton « Ajouter Memo » a été placé également dans la page JSP **vetDetails.jsp** :

```
<spring:url value="{vetId}/memos/new.html" var="addUrl">
  <spring:param name="vetId" value="{vet.id}"/>
</spring:url>
<a href="{fn:escapeXml(addUrl)}" class="btn btn-default">Add New Memo</a>
```

On voit que ce bouton va pointer vers l'url "**{vetId}/memos/new.html**".

Cette url est gérée dans la servlet contrôleur **MemoController** que nous allons ajouter par la suite. Le paramètre **vetId** va être remplacé par la valeur "**{vet.id}**" récupérée dans le bean **vet** (**<spring:param name="vetId" value="{vet.id}"/>**).

La fonction "**{fn:escapeXml(addUrl)}**" permet de remplacer les caractères spéciaux de l'url (exemple : les caractères accentués) par leur code html (exemple : **´** ; pour un « **é** »).

4 Ajouter un Memo à un vétérinaire

4.1 Exercice

Nous allons implémenter l'ajout de Memo.

Ajouter dans **src/main/java/org/springframework/samples/petclinic/web**, les 2 classes suivantes :

- MemoController.java
- MemoValidator.java

Classe MemoController.java

```
package org.springframework.samples.petclinic.web;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.samples.petclinic.model.Memo;
import org.springframework.samples.petclinic.model.Owner;
import org.springframework.samples.petclinic.model.Pet;
import org.springframework.samples.petclinic.model.PetType;
import org.springframework.samples.petclinic.model.Vet;
import org.springframework.samples.petclinic.service.ClinicService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.util.StringUtils;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.Collection;

@Controller
@RequestMapping("/vets/{vetId}")
public class MemoController {

    private static final String VIEWS_MEMOS_CREATE_OR_UPDATE_FORM =
"memos/createOrUpdateMemoForm";
    private final ClinicService clinicService;

    @Autowired
    public MemoController(ClinicService clinicService) {
        this.clinicService = clinicService;
    }

    @ModelAttribute("vet")
    public Vet findVet(@PathVariable("vetId") int vetId) {
        return this.clinicService.findVetById(vetId);
    }

    @InitBinder("vet")
    public void initVetBinder(WebDataBinder dataBinder) {
        dataBinder.setDisallowedFields("id");
    }

    @InitBinder("memo")
    public void initMemoBinder(WebDataBinder dataBinder) {
        dataBinder.setValidator(new MemoValidator());
    }

    @GetMapping(value = "/memos/new.html")
    public String initCreationForm(Vet vet, ModelMap model) {
        Memo memo= new Memo();
        vet.addMemo(memo);
        model.put("memo", memo);
        return VIEWS_MEMOS_CREATE_OR_UPDATE_FORM;
    }
}
```

```

    }

    @PostMapping(value = "/memos/new.html")
    public String processCreationForm(Vet vet, @Valid Memo memo, BindingResult
result, ModelMap model) {

        if (result.hasErrors()) {

            model.put("memo", memo);
            return VIEWS_MEMOS_CREATE_OR_UPDATE_FORM;
        } else {

            vet.addMemo(memo);
            this.clinicService.saveMemo(memo);
            return "redirect:/vets/{vetId}";
        }
    }
}

```

Classe MemoValidator.java

```

package org.springframework.samples.petclinic.web;

import org.springframework.samples.petclinic.model.Memo;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

public class MemoValidator implements Validator {

    private static final String REQUIRED = "required";

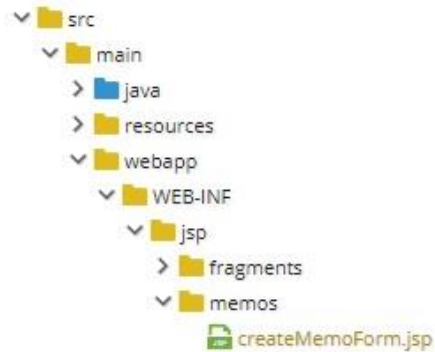
    @Override
    public void validate(Object obj, Errors errors) {
        Memo memo = (Memo) obj;
        String description = memo.getDescription();
        // description validation
        if (!StringUtils.hasLength(description)) {
            errors.rejectValue("description", REQUIRED, REQUIRED);
        }
    }

    /**
     * This Validator validates *just* Memo instances
     */
    @Override
    public boolean supports(Class<?> clazz) {
        return Memo.class.isAssignableFrom(clazz);
    }
}

```

Dans le répertoire **src/main/webapp/WEB-INF/jsp**, ajouter le répertoire memos.

Ajouter la JSP **createOrUpdateMemoForm.jsp**



JSP createOrUpdateMemoForm

```
<%@ page session="false" trimDirectiveWhitespaces="true" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="petclinic" tagdir="/WEB-INF/tags" %>

<petclinic:layout pageName="memos">
  <jsp:attribute name="customScript">
    <script>
      $(function () {
        $("#date").datepicker({dateFormat: 'yy/mm/dd'});
      });
    </script>
  </jsp:attribute>
  <jsp:body>
    <h2><c:if test="${memo['new']}">New </c:if>Memo</h2>

    <b>Vet</b>
    <table class="table table-striped">
      <thead>
        <tr>
          <th>First Name</th>
          <th>Last Name</th>
        </tr>
      </thead>
      <tr>
        <td><c:out value="${memo.vet.firstName}"/></td>
        <td><c:out value="${memo.vet.lastName}"/></td>
      </tr>
    </table>

    <form:form modelAttribute="memo" class="form-horizontal">
      <div class="form-group has-feedback">
        <petclinic:inputField label="Date" name="date"/>
        <petclinic:inputField label="Description" name="description"/>
      </div>
```

```
<div class="form-group">
  <div class="col-sm-offset-2 col-sm-10">

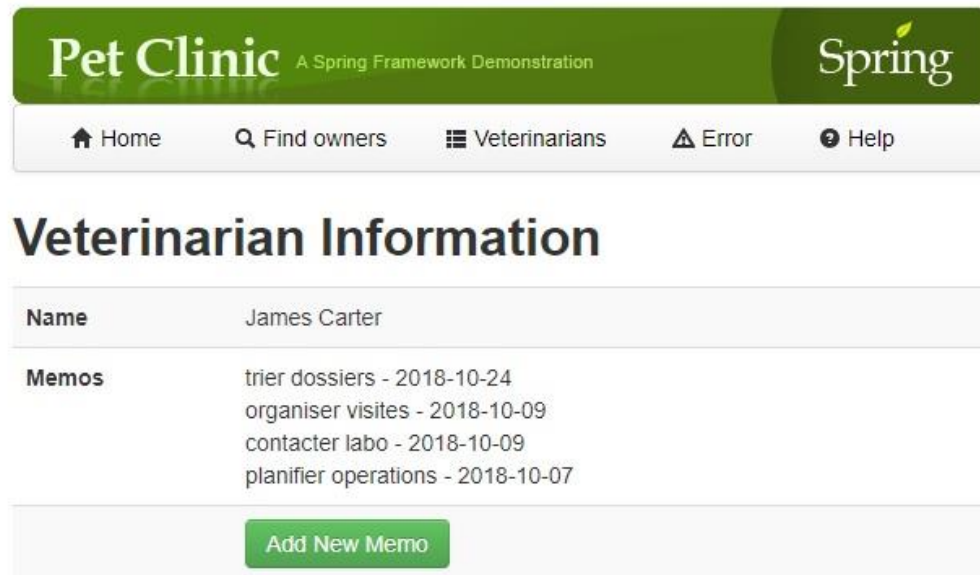
    <button class="btn btn-default" type="submit">Add Memo</button>
  </div>
</div>
</form:form>

</jsp:body>
</petclinic:layout>
```

Compiler et exécuter pour tester.
Vérifier que l'ajout de Memo à un vétérinaire est fonctionnel.

The screenshot shows the 'Pet Clinic' application interface. At the top is a green header with the 'Pet Clinic' logo and 'A Spring Framework Demonstration' text, followed by the 'Spring' logo. Below the header is a navigation bar with links: Home, Find owners, Veterinarians, Error, and Help. The main content area is titled 'New Memo'. It features a form with the following fields: 'Vet' (pre-filled with 'James Carter'), 'Description' (containing the text 'trier dossiers'), and a date field (containing '2018/10/24'). At the bottom of the form is an 'Add Memo' button.

Page d'ajout de mémo



Page de détails d'un vétérinaire incluant la liste des mémos

4.2 Explications :

4.2.1 Classe MemoController

Observez la classe controller **MemoController** :

La méthode **initCreationForm()** est appelée lorsque l'url **"/vets/{vetId}/memos/new"** est demandée avec la méthode GET (depuis le bouton « **ajouter Memo** » de la page **vetDetails.jsp**).

```
@GetMapping(value = "/memos/new")
public String initCreationForm(Vet vet, ModelMap model) {
    Memo memo= new Memo();
    vet.addMemo(memo);
    model.put("memo", memo);
    return VIEWS_MEMOS_CREATE_OR_UPDATE_FORM;
}
```

Cette méthode récupère l'objet vétérinaire passé en paramètre.

Elle instancie un objet de type **Memo** qu'elle ajoute à l'objet **vet** (ceci aura pour effet également de lier l'objet **vet** à l'objet **memo** → voir la méthode **addMemo()** de la classe **Vet**), pour préparer l'ajout d'un mémo.

Cet objet **memo** est stocké dans un objet **Model** qui sera utilisé par la page JSP pour récupérer les informations demandées.

Puis la méthode retourne une url sous forme de chaine de caractères :

```
return VIEWS_MEMOS_CREATE_OR_UPDATE_FORM
```

```
...
// (voir la constante)
private static final String VIEWS_MEMOS_CREATE_OR_UPDATE_FORM =
"memos/createOrUpdateMemoForm";
```

C'est donc la page JSP **createOrUpdateMemoForm** qui va être appelée.

4.2.2 JSP createOrUpdateMemoForm.jsp

Observez la JSP **createOrUpdateMemoForm.jsp**

Elle contient un formulaire permettant d'ajouter un nouveau **Memo** en saisissant une description et une date.

Le bouton **submit** va demander l'url « **/vets/{vetId}/memos/new** » mais avec la méthode http **POST** cette fois-ci.

Retournons dans **MemoController.java** :

Cette fois-ci, c'est la méthode **processCreationForm()** qui est appelée

```
@PostMapping(value = "/memos/new")
public String processCreationForm(Vet vet, @Valid Memo memo, BindingResult
result, ModelMap model) {

    if (result.hasErrors()) {

        model.put("memo", memo);
        return VIEWS_MEMOS_CREATE_OR_UPDATE_FORM;
    } else {

        vet.addMemo(memo);
        this.clinicService.saveMemo(memo);
        return "redirect:/vets/{vetId}";
    }
}
```

On remarque que cette méthode utilise un objet **MemoValidator()** pour valider les données saisies dans le formulaire avec l'annotation **@Valid**.

S'il y a des erreurs après validation, l'url « **memos/createMemoForm** » est retournée et la page **createOrUpdateMemoForm.jsp** est donc relancée à nouveau (avec gestion des messages d'erreurs → essayez de tester dans l'application en laissant le champ description du mémo vide).

Si il n'y a pas d'erreur, on est redirigé vers la page affichant le détail du vétérinaire (**return "redirect:/vets/{vetId}";**)

4.2.3 Classe MemoValidator.java

Observez le code de la classe **MemoValidator.java**

Cette classe permet de valider les données saisies dans le formulaire : là on va juste vérifier que le champ description n'est pas nul.

Elle contient l'annotation **@Validator** pour indiquer au framework qu'il s'agit d'une classe de validation de données.

Vous pouvez regarder également la classe **PetValidator.java** (dans web) qui est plus complète au niveau de la validation des données.

5 Gérer les Opérations

5.1 Exercice

Vous allez procéder de la même manière, par étapes pour ajouter la gestion des Opérations.

1. Pour ce faire, un onglet « pets » va être ajouté dans le menu horizontal de l'application.
2. Il permettra d'afficher la liste complète des animaux domestiques (sous forme de liens) avec leur nom, et leur date de naissance, ainsi que le nom et le prénom du propriétaire.
3. En cliquant sur un animal domestique, on aura une page `petDetails.jsp` rappelant les informations de l'animal (nom, date de naissance), ainsi que la liste des opérations de cet animal.
4. Un bouton « ajouter opération » permettra d'accéder à une page `vetsOperationList.jsp` contenant la liste des vétérinaires sous forme de liens (les informations de l'animal seront à nouveau rappelées sur cette page).
5. En cliquant sur un vétérinaire, une nouvelle page `createOrUpdateOperationForm.jsp` s'affichera. Celle-ci appellera les informations de l'animal et celles du vétérinaire sélectionné. Elle contiendra un formulaire permettant de saisir la description et la date de l'opération.
6. Après validation, l'utilisateur sera redirigé vers la même page s'il y a des erreurs et vers la page `petDetails.jsp` si il n'y a pas d'erreurs.

Respectez les étapes ci-dessus.

Re-testez bien l'application pour valider que l'étape courante est bien fonctionnelle avant de passer à la suivante.