Partie 2

Architecture Web

Application de gestion de location de véhicules

Exercice 1 : Configurer le projet en Webapp

1. Dans src/main créer un dossier webapp.
2. Copiez le dossier “webapp” contenu dans l’archive “webapp.zip” dans le dossier webapp de votre projet.

Exercice 2 : Mise en place de la première Servlet

Nous avons besoin de nouvelles dépendances à ajouter à notre fichier pom.xml pour la suite du projet.

```
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>javax.servlet-api</artifactId>
   <version>4.0.1</version>
   <scope>provided</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.annotation/javax.annotation-api -->
<dependency>
   <groupId>javax.annotation</groupId>
   <artifactId>javax.annotation-api</artifactId>
   <version>1.3.2</version>
</dependency>
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>jstl</artifactId>
   <version>1.2</version>
</dependency>
```

Il faut aussi ajouter dans le pom sous la version :
<packaging>war</packaging>

En dessous du plugin Exec, dans le pom.xml ajoutez le Tomcat embarqué :

```
 <plugin>
        <groupId>org.apache.tomcat.maven</groupId>
```

```
        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.2</version>
</plugin>
```

qu'on lance avec :

mvn tomcat7:run

Maintenant dans le package **com.epf.rentmanager.ui.servlets**, vous écrirez vos classes `Servlet` (ces dernières joueront le rôle de contrôleurs).

Créez une Servlet `HomeServlet` qui pointera vers "/home". Vous pouvez faire le mapping de la servlet en utilisant l'annotation `@WebServlet("/home")`. Lors de la création de votre Servlet n'oubliez pas de la faire hériter de `HttpServlet`, de plus n'oubliez d'y ajouter la méthode `doGet` pour traiter les requêtes HTTP.

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      // TODO
}
```

Utilisez la méthode `forward` de la classe `RequestDispatcher` afin d'afficher la vue **home.jsp**.

Compilez le projet

Si vous avez une erreur concernant la JSTL, localisez le jar dans votre repository maven local et copiez le dans le répertoir WEB-INF/lib de votre projet. Puis relancez le serveur. Faites de même pour le jar de h2.

Exercice 3 : Affichage des voitures présentes dans la base de données

Créez une Servlet `VehicleListServlet` dont le rôle sera d'afficher un tableau contenant la liste des véhicules présents dans la base de données.

En vous servant de la taglib core présentée en annexe, modifiez la page `WEB-INF/views/vehicle/list.jsp` pour que le tableau présente les données récupérées par la Servlet.

Exercice 4 : Insérer un véhicule

Créez une Servlet `VehicleCreateServlet` dont le rôle sera d'afficher un tableau contenant la liste des véhicules présents dans la base de données.

Dans la méthode `doGet`, récupérez et affichez la jsp contenant le formulaire de création de véhicule (`WEB-INF/views/vehicle/create.jsp`).

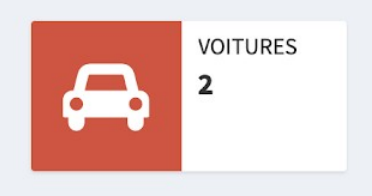
Dans la méthode `doPost`, traitez l'envoi du formulaire afin d'insérer un véhicule dans la base de données.

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    // affichage du formulaire
}
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    // traitement du formulaire (appel à la méthode de sauvegarde)
}
```

Exercice 5 : Affichage du nombre de véhicules présents dans la base

Dans la classe `VehicleDao`, ajouter une méthode `public int count()` permettant de récupérer la nombre de véhicules présents dans la base de données. Modifier la classe `VehicleService` pour qu'elle expose la méthode `count()`de la DAO. On pourra se référer à l'annexe de la première séance pour la requêtes SQL.

Mettre à jour la Servlet `HomeServlet` et la vue `WEB-INF/views/home.jsp` pour afficher cette valeur dans le rectangle approprié.



Annexes

A. Utiliser la taglib *core*

Pour manipuler les éléments fournis par la taglib *core* dans une JSP, il est nécessaire de placer le code suivant en début de fichier (avant la balise <html>) :

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

Cela permet d'importer la taglib et de lui associer un raccourci (ici « c »).
Une fois l'import effectué, vous serez en mesure d'utiliser les balises de type `<c:machin>`.

Faire des boucles foreach

On considère que l'attribut items est une liste d'éléments qui a été ajoutée à la requête transmise à la JSP à l'aide du code `request.setAttribute("items", uneListeDElements);`

```
<c:forEach items="${items}" var="item">
...
</c:forEach>
```

Chaque élément de la liste `items` sera considéré comme un élément `item` qui pourra être manipulé à l'intérieur de la boucle, avec les Expression Languages par exemple.

Faire des if

On considère que l'attribut value a été ajouté à la requête transmise à la JSP à l'aide du code `request.setAttribute("value", uneValeur);`

```
<c:if test="${value == false}">
...
</c:if>
```

Si le résultat du test est évalué à vrai, alors le bloc de code contenu entre les balises sera exécuté.