# TP JEE (1)
# Développement Web en Java

Dans ce TP, nous commencerons la programmation JEE par le premier niveau d'une application JEE : l'application web.


FIGURE 1 – Architecture multi-niveaux avec Java EE

Ce premier niveau d'application est constitué de deux éléments clé :
- Les JSP (JavaServer Pages) sont des fichiers qui peuvent mélanger Java, HTML et JavaScript. Ces fichiers ont pour but de présenter les données au client. Une JSP est un fichier qui sera transformé en Servlet puis compilé par le conteneur de Servlets.
- Les Servlets sont de simples classes Java qui héritent de la classe HttpServlet et doivent implémenter deux méthodes : *doPost(...)* , *doGet(...)*. Ces deux méthodes répondent aux deux opérations principales du protocole HTTP : GET et POST.

Dans ce TP, nous allons créer notre première application JEE.

Le but de ce TP est de créer un petit projet Web permettant de créer un contact, de l'afficher, puis, de le modifier. Pour cela, nous allons créer deux fichier JSP responsables de l'affichage d'un contact et de son édition (ViewContact.jsp et EditContact.jsp). Chacune de ces JSP communiquera avec une classe Java qui aura le rôle de contrôleur : le Servlet (ContactServlet.java).

# Table des matières

# 1 Création du projet avec Maven

### Récupérer le projet squelette

1. Récupérez le projet (vide) ContactWeb.

```
hg clone ssh://your_login@info-ssh1.iut.u-bordeaux1.fr//net/
   Bibliotheque/JEE/2011-2012/skull-tp-war/
```

2. Générez un projet Eclipse et ouvrez-le.
3. Déployez le projet sur Tomcat et testez la servlet HelloWorld.

# 2 La partie présentation

## 2.1 Web statique

### Création d'un fichier HTML/ JavaScript

1. Ajoutez un fichier HTML (index.html) dans le dossier src/main/webapp. Editez-le afin d'afficher une image de votre choix, ainsi qu'un lien nommé *Ajouter Contact* vers l'url *./EditContact.jsp*


FIGURE 2 – Lien entre index.html et la JSP EditContact que l'on créera plus tard.

2. Ajoutez du code Javascript pour afficher la date du jour.
3. Re-déployez un nouveau WAR de votre application vers Tomcat.
4. Visitez la page HTML que vous venez de créer avec votre navigateur.

## 2.2 Les JSP (Java Server Pages)

Les JSP (Java Server Pages) correspondent à la partie vue (web) d'une application JEE. Ces pages sont dynamiques et peuvent contenir du code Java. Les JSP sont converties en classes Java par le conteneur de Servlets (par exemple Tomcat).

Il existe plusieurs solutions pour écrire du code Java dans une JSP. Basiquement, il suffit de le placer entre les balises <% et %>. On parle de scriptlet. Pour évaluer une expression (variable Java par exemple) il faut par contre utiliser la balise<%=.

```
<%@ page language="java" contentType="text/html; charset=ISO
    -8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import="java.util.List" %>
<%@ page import="java.util.ArrayList" %>
...
<%
        List<Integer> entiers = new ArrayList<Integer>();
        entiers.add(1);entiers.add(2);entiers.add(3);
        for(Integer i : entiers)
        {
%>
                <p>Numero : <%= i %></p>
<%
        }
%>
```

FIGURE 3 – Utlisation de code java dans une page JSP.

### Une première JSP

1. Créez un nouveau fichier JSP dans /src/main/webapp : TestJSP.jsp.
2. En vous inspirant du code source 3, créez une liste de chaînes de caractères dans un premier temps, puis affichez leurs valeurs.
3. Dans une JSP, comment utiliser les conditions (if, else, ...) ?
4. Re-déployez et testez dans votre navigateur.

### La JSP EditContact

1. Créez un nouveau fichier JSP dans /src/main/webapp que l'on appellera EditContact.jsp.
2. Modifiez cette JSP afin qu'elle affiche un formulaire permettant d'insérer un nom, un numéro de téléphone, et une date de naissance.
3. Modifiez la JSP pour que le champ *date de naissance* soit par défaut égal à la date du jour (utilisez Javascript).
4. Testez dans votre navigateur.

5. Remplacez le code Javascript par du code Java.
6. Quelle est la différence entre le code Java et le code Javascript ?

# 3 Contrôleurs, Servlets

Une servlet est une classe Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP. Ces données sont le plus généralement présentées au format HTML, mais elles peuvent également l'être au format XML, ou tout autre format destiné aux navigateurs web. Les servlets utilisent l'API Java Servlet (package javax.servlet).

Une servlet s'exécute dynamiquement sur le serveur web et permet l'extension des fonctions de ce dernier, typiquement : accès à des bases de données, transactions d'e-commerce, etc. Une servlet peut être chargée automatiquement lors du démarrage du serveur web ou lors de la première requête du client. Une fois chargées, les servlets restent actives dans l'attente d'autres requêtes du client.

L'utilisation de servlets se fait par le biais d'un conteneur de servlets côté serveur. Celui-ci constitue l'environnement d'exécution de la servlet et lui permet de persister entre les requêtes des clients. L'API définit les relations entre le conteneur et la servlet. Le conteneur reçoit la requête du client, et sélectionne la servlet qui aura à la traiter. Le conteneur fournit également tout un ensemble de services standards pour simplifier la gestion des requêtes et des sessions.

## 3.1 Les Servlets

**Créer une nouvelle servlet avec Eclipse**

1. En vous inspirant de la classe *HelloWorld*, ajoutez une nouvelle servlet au projet : ContactServlet.
2. À quoi sert l'annotation *@WebServlet*, les méthodes *doGet* et *doPost* ?
3. Le protocole HTTP définit 4 types d'opérations : GET, POST, DELETE, PUT. Quel est l'utilité de PUT et DELETE ? Comment les implémenter avec une Servlet ?

**Test de la nouvelle servlet**

Un objet de type *PrintWriter* peut être récupéré depuis l'objet *response* afin d'écrire des chaînes de caractères dans la réponse d'une servlet.

1. Créez une classe Contact (name, email, phone, birthday).

2. Modifiez la méthode *doGet(...)* afin d'afficher les informations d'une instance d'un contact.
3. Testez votre toute nouvelle Servlet.

## 3.2 Communication entre une *Servlet* et une *JSP*

### Transférer la requête vers une JSP

Il est possible, grâce à l'objet *RequestDispatcher* accessible via la méthode *request.getRequestDispatcher* de l'objet *request*, de transférer la requête et la réponse de la servlet vers une *JSP*.

1. Modifiez la méthode *doGet* afin de rediriger l'utilisateur vers la JSP *EditContact*.

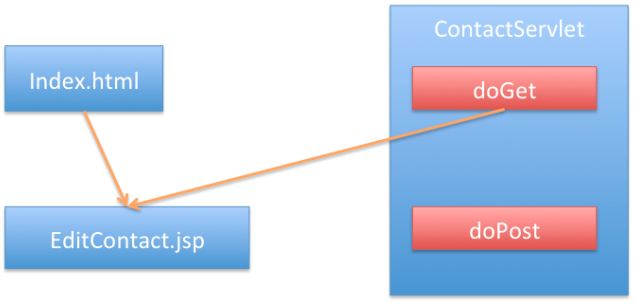


FIGURE 4 – Transfert de la requête *doGet* vers la JSP *EditContact*

2. Testez.
3. Modifiez la méthode *doPost* afin de récupérer les paramètres du formulaire. On pourra utiliser la méthode *getParameter* de l'objet *request*.
4. Il est possible de passer des paramètres à une JSP en utilisant la méthode *request.setAttribute(String e, Object o)*. Avec e le nom du paramètre et o sa valeur. Puis, du côté JSP on pourra récupérer ce paramètre en utilisant la méthode *request.getAttribute(String e)*. Utilisez ces méthodes afin d'afficher le contact créé par la méthode *doPost(...)* dans une nouvelle *JSP ViewContact.jsp*.

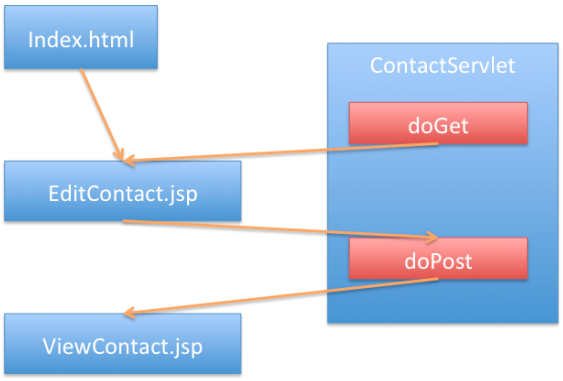


FIGURE 5 – Suite d'appels JSP et Servlet afin d'ajouter un contact

5. Testez.

### Les JavaBeans

Les *JavaBeans* (à ne pas confondre avec les Enterprise JavaBeans, EJB, que nous verrons plus tard) sont de simples classes Java mais qui sont écrites en suivant une spécification bien particulière :

- Elle doivent implémenter l'interface *Serializable*.
- Chaque attribut doit avoir un getter et un setter de la forme *getXxx* et *setXxx* où xxx est le nom de l'attribut. Par exemple, l'attribut "name" doit avoir un setter *setName(String name)* et un getter *String getName()*.

1. Modifiez la classe *Contact* afin qu'elle soit conforme à la spécification JavaBeans.

Il est possible d'utiliser des JavaBeans dans les JSP grâce aux tags *jsp:useBean*, *jsp:setProperty* et *jsp:getProperty*.

```
<jsp:useBean id="contact" class="iut.jee.Contact" scope="request
    ">
        <jsp:setProperty name="contact" property="*" />
</jsp:useBean>

<form action="./viewcontact" method="post">
        <input type="text" value="${contact.name}" name="name" />
            <input type="text"
                value="${contact.email}" name="email" />
</form>
```

1. En vous inspirant du code précédent, modifiez la JSP *EditContact* afin d'utiliser le composant JavaBean *Contact*. Testez.

2. À quoi sert l'attribut *scope* de la balise *jsp:useBean* ?
3. À quoi sert la balise *jsp:setProperty* ?

Il est possible d'ajouter des variables en session. On peut récupérer cette session depuis l'objet *request* et la méthode *getSession*. Une fois la session obtenue on peut utiliser la méthode *putValue* pour y ajouter des variables, ou la méthode *getValue* pour récupérer des informations.

1. Modifiez la JSP *ViewContact.jsp* pour que, en plus d'afficher un contact, elle propose un lien pour le modifier.

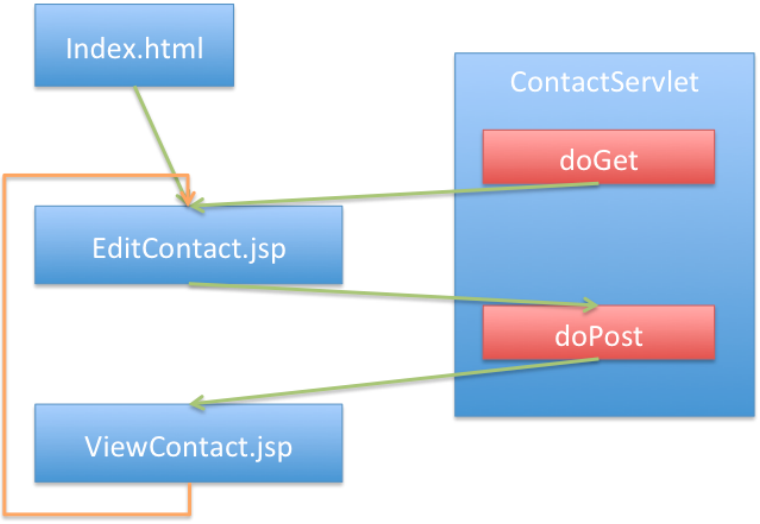


FIGURE 6 – Modifier un contact depuis *ViewContact*

2. Utilisez la session pour afficher les informations du contact à éditer dans la JSP *EditContact.jsp*.

# 4 Simulation d'une base de donnée

1. Ajoutez un attribut de type *HashMap<String, Contact>* (nommé *contacts*) dans *ContactServlet*.
2. Dans le constructeur de *ContactServlet* ajoutez quelques contacts dans cette *HashMap*.
3. Modifiez la méthode *doPost* afin que le nouveau contact soit ajouté à la *HashMap*.
4. Créez une JSP (*ViewAllContacts.jsp*) permettant d'afficher le nom de l'intégralité des contacts présents dans la HashMap.
5. Modifiez la méthode *doGet* afin de gérer plusieurs types d'action :

– Si le paramètre *action* est égal à *viewAll*, rediriger vers *ViewAllContacts* en insérant dans la requête l'attribut *contacts*.

```
/contact?action=viewAll
```

– Si le paramètre *action* est égal à *view*, récupérer le paramètre *name*, et rediriger vers la JSP *ViewContact* avec en requête l'attribut de type Contact correspondant au nom passé en paramètre.

```
/contact?action=view&name=Vincent
```

– Si le paramètre *action* est égal à *add* rediriger vers la JSP *EditContact*.

```
/contact?action=add
```

– Si le paramètre *action* est égal à *edit* rediriger vers la JSP *EditContact*, avec en requête l'attribut de type Contact correspondant au nom passé en paramètre.

```
/contact?action=edit&name=Vincent
```

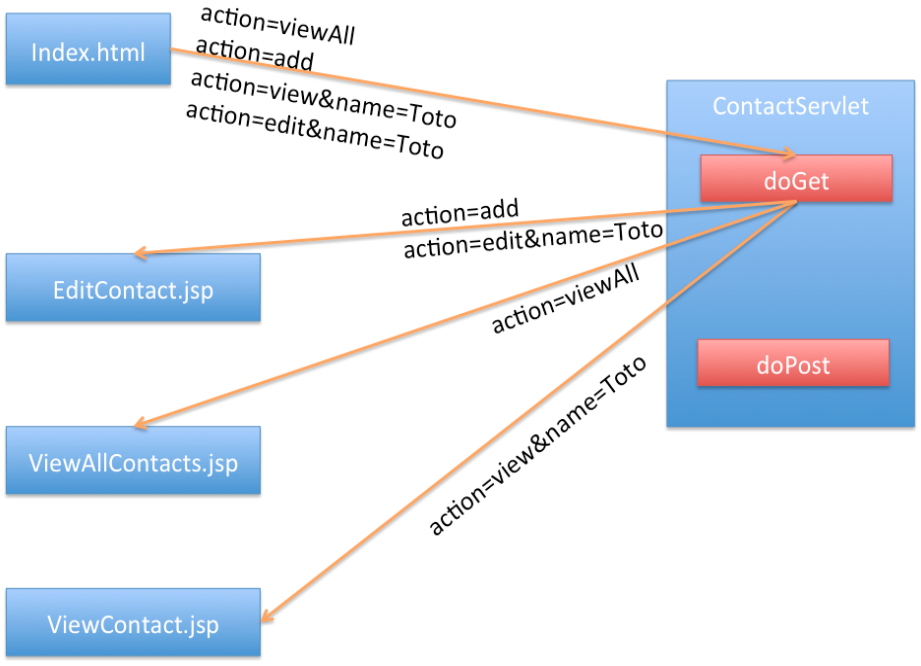


FIGURE 7 – Il est possible de donner des paramètres à *doGet* afin de spécifier une action. On se rapproche un tout petit peu de MVC2.

6. Modifiez votre fichier index.html pour proposer un lien permettant d'afficher tous les contacts.
7. Modifiez la JSP ViewAllContacts.jsp et ajoutez un lien à chaque contact pour soit l'éditer, soit voir toutes ses informations. Ajouter aussi un lien permettant d'ajouter un nouveau contact.

8. Modifiez la méthode *doPost* de la Servlet *ContactServlet* pour qu'elle redirige vers la JSP *ViewAllContacts*.

# 5 La culture Java EE (ou : avoir une bonne note à l'exam)

À l'aide de votre navigateur favori, répondez aux questions suivantes :

1. Qu'est-ce qu'un conteneur de servlets ? Citez un exemple.
2. Qu'est-ce qu'un fichier WAR ?
3. Qu'est-ce que le fichier *web.xml* ? À quoi sert-il ? Pourquoi n'en n'avons nous pas besoin dans ce TP ?
4. Quelle est la différence entre une JSP et une *Servlet* ?
5. Qu'est-ce qu'un *JavaBean* ?
6. Quels sont les attributs accessibles en Java depuis une *JSP* ?
7. À quoi sert la session HTTP ?
8. Qu'est-ce que Struts et Spring MVC ?
9. Quelle est la différence entre MVC et MVC2 ? Quels sont les avantages et inconvénients de l'un par rapport à l'autre ?
10. La question troll : Java vs PHP ? Et Ruby On Rails ?