POLYTECH ANGERS

TP 4A - Java Entreprise Edition.
JSP (java Server Pages) et MVC (Modèle vue controleur)

Nicolas Delanoue et Sylvain Joyeau

Le but de ce TP est que vous vous familiarisiez avec la notion de page jsp et de mettre en place le patron de conception Modèle-Vue-Controleur.

Exercice 1 (Premières jsp)

1. Créer une page jsp nommée `q1.jsp` de sorte qu'elle génère le fichier html suivant. On pourra par exemple utiliser la librairie `java.util.Date` pour afficher la date actuelle.

```
<!DOCTYPE html>
<html>
 <head> <meta charset="UTF-8"> <title>Quand et où ? </title> </head>
 <body>
   <p>
      Nous sommes le  Sat Feb 20 18:17:56 CET 2021 <br>
      Le nom du serveur sur lequel vous êtes connecté : localhost <br>
   </p>
 </body>
 </html>
```

2. Créer une page jsp nommée `q2_aleatoire.jsp` qui génère un nombre aléatoire à chaque appel. La sortie générée pourra être la suivante :

```
<!DOCTYPE html>
<html>
 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>Ne pas oublier l'import de java.util.Random ;-)</title>
 </head>
 <body>
    Le nombre aléatoire généré est 12. <br>
    <a href="/TP3_JSP/question2/q02_aleatoire.jsp">Recommencer !</a>
 </body>
</html>
```

3. Dans la page `q2_aleatoire.jsp`, utiliser l'instruction `request.getRequestURI()` pour compléter le pointeur de la balise `<a href>`.

4. Créer une page jsp nommée `PageAvecErreur.jsp` contenant le code suivant :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
         pageEncoding="UTF-8"%>
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="UTF-8">
<title>Division par zero !</title>
</head>
<body>
 <%
  if(request.getParameter("attention") == null)
    {response.getWriter().append("Cette page fonctionne !<br>"); }
  else
    {int i = 1/0;}
  %>
</body>
</html>
```

5. Analyser le code de cette page jsp afin de proposer une url faisant en sorte que cette page jsp génère une exception ?
6. Ajouter un attribut `import` à la directive `page` de la page `PageAvecErreur.jsp` de sorte que lors d'une exception, une page nommée `error.jsp` prenne en charge son affichage.
7. Créer la page jsp `error.jsp`.

Exercice 2
L'idée de cet exercice est de coupler une servlet à une page jsp afin de découpler le traitement de l'affichage.

Plus précisément, la page `vue.jsp` sera responsable de l'affichage alors la servlet `ToUpperCase` s'occupera du traitement. Les échanges sont illustrés par la figure 1.

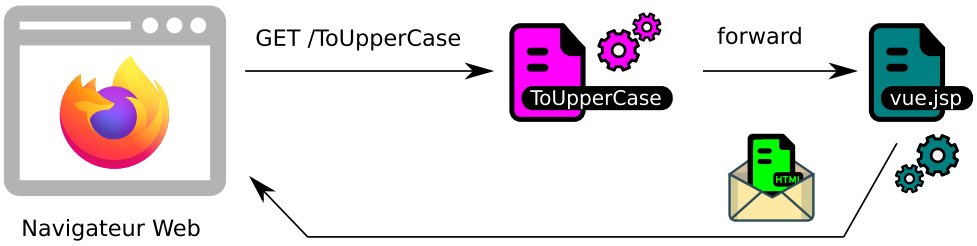

FIGURE 1 – Communications pour l'application à développer.

1. Créer une nouvelle Application Web Dynamique nommée `Exercice2`.
2. Créer une servlet nommée `ToUpperCase` qui
   (a) récupère la valeur associée à un paramètre `"nom"` passée lors d'une requête http get,
   (b) produit une chaine de caractères en mettant en majuscule ce nom, dans une variable `NomEnMajuscule` de type `String`,
   (c) ajoute cette variable, sous forme d'attribut nommé `AttributNomEnMajuscule`, à l'objet `request`,
   (d) transmet automatiquement le traitement de la requête à la page jsp `vue.jsp`.

3. Créer la page `vue.jsp`, dans le répertoire `/WEB-INF` de votre projet, de sorte qu'elle affiche un message du type : `Bonjour ALAN TURING`.
4. Dans cet exercice, la transmission entre la servlet et la page jsp est réalisée grâce à l'ajout d'un attribut à l'objet `request`, est ce que le client peut voir cet échange ?
5. Peut-on accéder directement à la page `vue.jsp` depuis le navigateur ?
6. Par quel autre mécanisme aurions-nous pu transmettre la variable `NomEnMajuscule` entre la servlet et la page jsp sans que le client ne puisse la voir ? Quelles différences avec un attribut de requête ?

Exercice 3

Pour cet exercice, on dispose d'une application Web qui permet de créer une commande de livraison de petits déjeuners dans un hôtel. Dans cet exercice, on analyse un code de mauvaise qualité avant de le modifier dans l'exercice 4.

1. Télécharger l'application web via :
   `http://perso-laris.univ-angers.fr/~delanoue/polytech/jee/TP3_ex3.zip`.
2. Importer puis déployer cette application dans votre serveur Tomcat.
3. Cette application est fonctionnelle mais souffre de nombreux défauts de conception. Lesquels ?
4. Identifier dans le code les vérifications sur les champs des formulaires qui empêchent la création d'une commande de petit déjeuner.

Exercice 4 (Refactoring - mise en place d'un design pattern MVC)
L'objectif est de modifier le code afin que cette application respecte le patron de conception MVC (tant en gardant les fonctionnalités). A la fin de cet exercice, l'application aura le comportement illustré par la figure 2 :

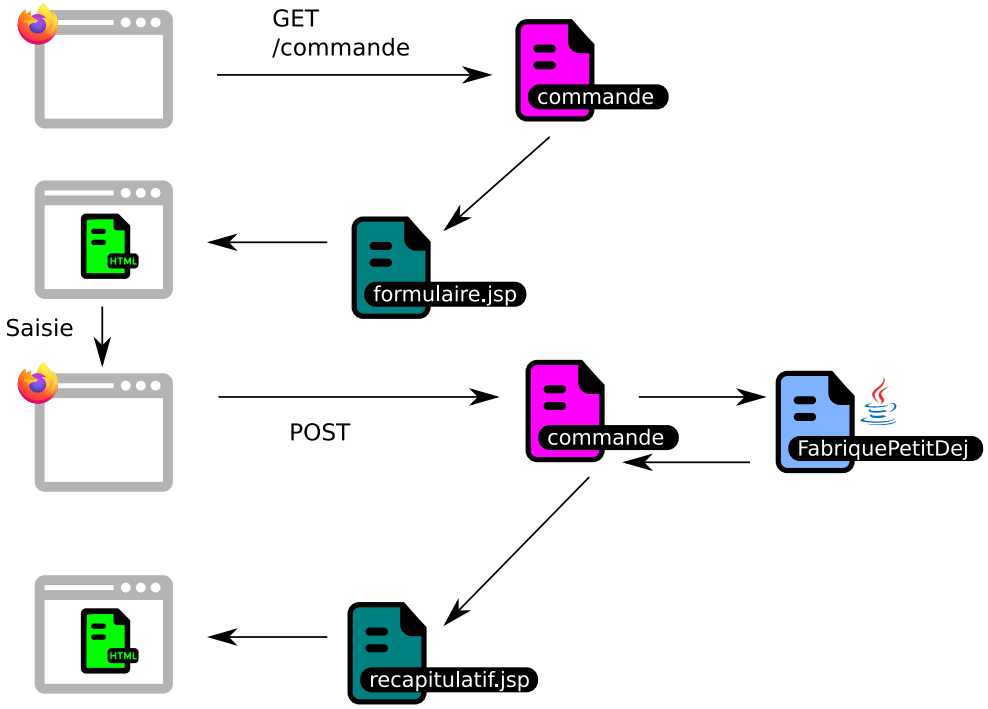

FIGURE 2 – Plan d'exécution.

1. Créer une nouvelle application web.
2. Créer une page jsp nommée `formulaire.jsp` qui affiche un formulaire identique à celui de l'application de l'exercice précédent.
3. Placer le fichier `formulaire.jsp` dans le répertoire `/WEB-INF`.
4. Créer une servlet `commande` et modifier la méthode `doGet()` de sorte qu'elle déclenche l'affichage de la page jsp `formulaire.jsp`.
5. Créer une classe métier nommée `FabriquePetitDej`. A terme, cette classe gérera le formulaire de création d'une commande et fabriquera un objet de la classe `PetitDejeuner`. La classe métier `FabriquePetitDej` pourra reprendre les champs du formulaire comme premiers attributs :
   - `private static final String CHAMP_CHAMBRE = "chambre";`
   - `private static final String CHAMP_HEURE = "heure";`
   - `private static final String CHAMP_NB_REPAS = "nombreDeRepas";`
   - `private static final String CHAMP_COMMENTAIRES = "commentaires";`
6. Ajouter à la classe `FabriquePetitDej` la méthode suivante :

   ```
   PetitDejeuner construitPetitDejeuner( HttpServletRequest request  )
   ```

   Coder le contenu de cette méthode de sorte qu'elle crée un objet de la classe `PetitDejeuner` à partir des données contenues dans l'objet `request`.

7. Modifier la méthode `doPost()` de la servlet `commande` en y insérant les deux lignes suivantes :

```
// Préparation de la fabrique
FabriquePetitDej fabrique = new FabriquePetitDej();
// Traitement de la requête et récupération du bean en résultant
PetitDejeuner petitDej = fabrique.construitPetitDejeuner(request);
// Ajout du bean à l'objet request pour Dispatcher vers recapitulatif.jsp
request.setAttribute("attributPetitDej", petitDej );
```

8. Ajouter un `getRequestDispatcher` à la fin de cette méthode `doPost` afin d'automatiquement orienter le voyageur vers `recapitulatif.jsp` (comme illustré en bas de la figure 2.
9. Créer une page jsp `recapitulatif.jsp` qui présente les attributs du petit déjeuner validé et souhaite bonne nuit au voyageur.
10. Sur la page `recapitulatif.jsp`, vous avez utilisé, au choix, l'une des trois syntaxes suivantes :

    (a)
    ```
    <%
    PetitDejeuner petitDej;
    petitDej = (PetitDejeuner) request.getAttribute("attributPetitDej");
    %>
    <%= petitDej  %>
    ```

    (b) `<%= request.getAttribute("attributPetitDej") %>`

    (c) `${requestScope.attributPetitDej}`

    Laquelle préférez-vous ? Discuter.

Exercice 5 (Refactoring - mise en place d'un design pattern MVC)
A la fin de cet exercice, cette nouvelle application aura le comportement suivant :

- en cas d'erreur lors de la validation (champs manquants ou erronés), vous devrez faire retourner l'utilisateur au formulaire de saisie en
  - lui réaffichant les données qu'il a saisies,
  - précisant un message signalant les erreurs sur chaque champ qui pose problème ;
- en cas de succès, vous devrez envoyer l'utilisateur vers la page qui affiche le récapitulatif.

Pour arriver à cet objectif, nous allons mettre en place des tests de validations dans l'objet métier `fabriquePetitDej` afin d'accéder à la page `recapitulatif.jsp` uniquement dans le cas uniquement si les contraintes d'horaires et de numéro de chambres sont satisfaites. Lors d'une contrainte non satisfaite, le formulaire sera automatiquement rempli avec les données précédemment renseignées et complétées par des messages expliquant les erreurs. Un exemple d'exécution est illustré par la figure 3.

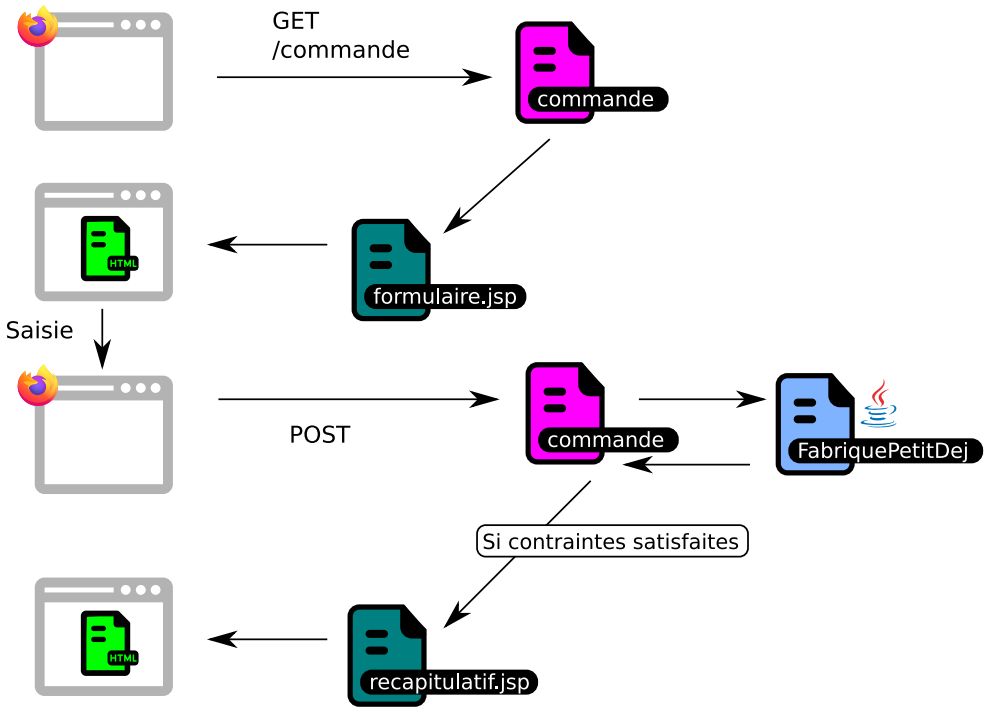

FIGURE 3 – Illustration de la suite d'appels.

1. Créer une nouvelle application web afin de bien distinguer cet exercice du précédent.
2. Ajouter à la classe `fabriquePetitDej` la méthode `validationChambre(int chambre)` dont le code est le suivant :

```
private void validationChambre( int chambre ) throws Exception {
  if (( chambre < 100 ) || ( 200 < chambre ))
     {
       throw new Exception( "Le numéro de chambre doit
                             être contenu entre 100 et 200." );
     }
  }
```

3. Ajouter à la classe `fabriquePetitDej` et à partir du modèle de la question précédente, une méthode `private void validationHeure(String heure)`.
4. Ajouter à la classe `fabriquePetitDej`, l'attribut suivant

   ```
   private Map<String, String> erreurs = new HashMap<String, String>();
   ```

   Cet objet sera utilisé pour stocker les messages d'erreur à retourner au formulaire. Lors de la question suivante, on pourra utiliser comme clé pour cette `Map` une des constantes `CHAMP_CHAMBRE`, `CHAMP_HEURE`, ...
5. Modifier la méthode `construitPetitDejeuner()` avec des `try catch` de sorte que
   — on complète les attributs de l'objet `petitDej` un par un après avoir tester leur validité,
   — l'objet `erreurs` contienne des messages appropriées à destination de l'utilisateur.
6. Ajouter à la classe `fabriquePetitDej` un attribut `public boolean succesCreation`, puis compléter la fin de la méthode `construitPetitDejeuner()` avec le code suivant :

   ```
   succesCreation = erreurs.isEmpty()
   ```
7. Modifier la servlet `commande` afin qu'elle redirige vers la page `formulaire.jsp` ou bien vers `recapitulatif.jsp` en fonction du succès de l'exécution de la méthode `construitPetitDejeuner()`.
8. Compléter le formulaire de sorte qu'il puisse afficher les messages d'erreurs à coté des champs qui ont posé problème.

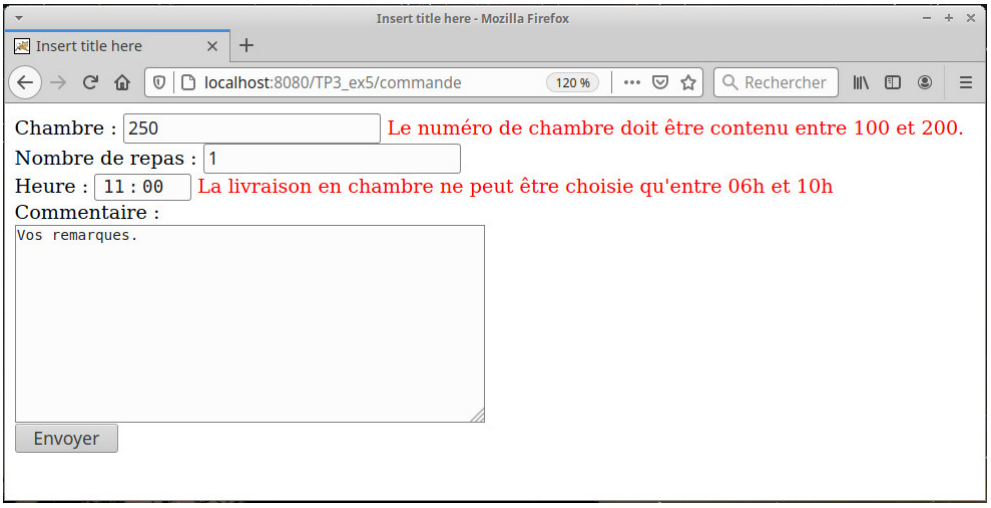

FIGURE 4 – Exemple d'affichage en cas d'erreur.