JavaEE
Développement d'application web

Aix-Marseille Université

Gaël Guibon, Jean-Luc Massat, Omar Boucelma

2018-2019

Sommaire

Chapitre 1

Notions de bases et environnement

1.1 Notions de bases

1.1.1 Java et JavaEE

Vérification du niveau des étudiants en java standard
Topo sur l'intéret du JEE par rapport au java standard + distinction serveur / navigateur
Quelques rappels :

Java	**JavaEE**
Socle de base	Extension de Java
Programmes locaux	Programmes connectés
Entrée par méthode main()	Entrées par chaque servlet
HTML	**JavaEE**
Site web statique	Site web dynamique
Visible	Opaque et sécurisé

1.1.2 Le modèle MVC

Le principe Modèle Vue Controlleur (MVC) est une bonne pratique de programmation.

Modèle	**Vue**	**Controlleur**
Données	Affichage IHM	Actions / logique du code
Accès BDD	Adaptation de l'UI	Lien entre les éléments
Messages/contacts	Visualisation du message/contacts	Boutons "répondre à tous", envoyer emoji, ...

Exemple d'une répartition MVC :

De nombreux frameworks en différents langages sont désormais MVC :

- Java : Spring [1], Struts [2], Tapestry [3]
- Python : Django, Flask, Pyramid
- Javascript : Angular, ReactJS, EmberJS

1. http://spring.io/
2. http://struts.apache.org/
3. http://tapestry.apache.org/

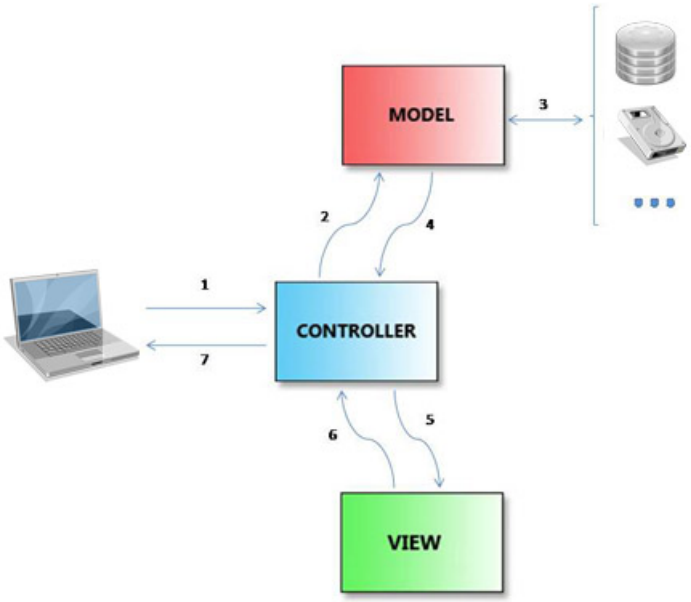


FIGURE 1.1 – Exemple : Messagerie SMS

1.2 Environnement

1.2.1 Installer Java8

Ouvrir un terminal et lancer les commandes suivantes :

1. sudo add-apt-repository ppa :webupd8team/java
2. sudo apt-get update -y
3. sudo apt-get install oracle-java8-installer
4. java -version

La dernière commande devrait vous afficher la version 1.8.x de Java.

1.2.2 Installer Maven

1. cd /opt/
2. wget http ://apache.crihan.fr/dist/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz
3. sudo tar -xvzf apache-maven-3.3.9-bin.tar.gz
4. sudo mv apache-maven-3.3.9 maven
5. sudo nano /etc/profile.d/mavenenv.sh
6. Dans ce fichier ajouter les lignes suivantes :

```
export M2_HOME=/opt/maven
export PATH=${M2_HOME}/bin:${PATH}
```

7. sudo chmod +x /etc/profile.d/mavenenv.sh
8. sudo source /etc/profile.d/mavenenv.sh
9. mvn –version

1.2.3 Installer l'IDE : Eclipse Oxygen

1. Télécharger Eclipse : https://www.eclipse.org/downloads/download.php?file=/oomph/epp/oxygen/R/eclipse-inst-linux64.tar.gz
2. Lancer l'installeur
3. Choisir Eclipse for JavaEE
4. Suivre l'installation avec les configurations par défaut

1.3 Comprendre Maven

1.3.1 Qu'est-ce que c'est ? ! !

Un outil d'automatisation :

— Très répandu en entreprise
— Pour la compilation
— Pour le déploiement
— Facilite le partage de code

Format d'usage :

```
usage: mvn [options] [<goal(s)>] [<phase(s)>]
```

1.3.2 Les éléments de base

```
<project>
  <modelVersion/> VERSION DE MON PROGRAMME
  <groupId/> ID DU GROUPE DE MON PROGRAMME : cnrs.amu.lsis ou com.usa.google ou autre
  <artifactId/>  ID DE MON PROGRAMME : monprojet ou minecraft ou autre
  <packaging/> TYPE DE PAQUET EN SORTIE : jar ou war ou autre
  <version/>  VERSION DE MON PROGRAMME : 0.0.9-beta ou 2.1.56
  <name/> NOM EN INTERNE
  <url/> URL DU PROGRAMME : par defaut mettre : http://maven.apache.org
  <dependencies/> DEPENDANCES DU PROGRAMME : jsp, guava, et autres librairies externes
  <build>  EXECUTION LORS DE LA COMPILATION DU PROGRAMME
    <plugins/> PLUGINS DIVERS
  </build>
  </project>
```

1.3.3 Architecture d'un projet maven

Générer un projet maven basique (java SE) :

```
mvn archetype:generate -DgroupId=test.project -DartifactId=superTest -DarchetypeArtifactId=maven-archetype-quickstart
      -DinteractiveMode=false
```

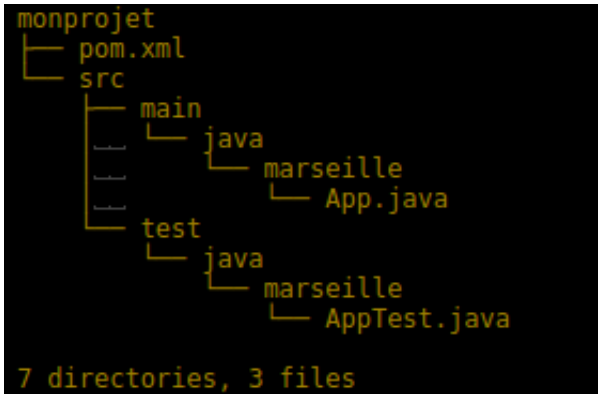


FIGURE 1.2 – Arborescence de base de maven

L'explication officielle de l'arborescence est présente ici : http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

1.3.4 Le fichier POM.xml

C'est le fichier central ! Celui qui permet de guider maven dans la compilation.
Un guide officiel du POM est présent à cette adresse : http://maven.apache.org/guides/introduction/introduction-to-the-pom.html

Les dépendances

```
<dependencies>
  <dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-core</artifactId>
    <version>${tomcat.version}</version>
  </dependency>
  .
  .
```

```
    .
</dependencies>
```

Les plugins

```
<plugins>
    <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.3</version>
        <configuration>
            <source>1.8</source>
            <target>1.8</target>
        </configuration>
    </plugin>
    .
    .
    .
</plugins>
```

1.3.5 Accéder aux dépendances et plugins

Aller chercher sur maven central : https://search.maven.org/

1.4 Hello World

1. Cloner le repository : git clone https ://github.com/gguibon/CoursesJavaEE.git
2. Aller dans le projet de base : cd basic_hello_world
3. Terminal : mvn package
4. Terminal : java -jar target/basicHelloWorld-jar-with-dependencies.jar OU double clic sur le jar
5. Ouvrir un navigateur à l'adresse localhost :8080

Astuce : La commande htop vous indique les processus en cours (sudo apt install htop)

1.5 Sources utiles pour compléter

Documentation officielle de maven : http://maven.apache.org/guides/index.html
Tutoriel officiel : https://docs.oracle.com/javaee/7/tutorial/
Documentations et API officielle : http://docs.oracle.com/javaee/7/index.html
Tutoriel Site du Zero : https://openclassrooms.com/courses/creez-votre-application-web-avec-java-ee
Mémento et exemples d'annotations Servlet 3.0 : http://www.codejava.net/java-ee/servlet/servlet-annotations-

Chapitre 2

Le modèle - beans

2.1 Qu'est-ce que c'est ?

Une manière de stocker les données. Un est :
- Réutilisable ! (objet java)
- Persistant ! (serialization)
- Paramétrable ! (propriétés)

Il respecte donc plusieurs règles :
- C'est une classe publique.
- Implémente Serializable pour pouvoir être sauvegardé.
- AUCUN champ public ! (getter et setters)
- Possède un constructeur par défaut public, sans paramètres ?

2.2 Exo : Agenda

2.2.1 Créer le bean Rendezvous

Créer un bean (un objet java) Rendezvous.java.

1. Ajouter les propriétés "duree"(int), "personnes"(liste de noms), "lieu"(string) et "type"(string)
2. Ajouter des getter et setter pour chaque propriété
3. Ajouter une méthode qui retourne le nombre de personnes

2.2.2 Créer le service RendezvousService

Un service est une classe qui va ouvoir être instancier et gérer un ou plusieurs beans, ainsi que leurs relations.

1. Créer un service RendezvousService.java

```
@Service //annotation qui indique àspring que c'est un service
public class RendezvousService { }
```

2. Ajouter une liste de rendezvous
3. Ajouter un constructeur qui initialise des rendez-vous par défauts
4. Ajouter une méthode d'ajout de rendez-vous
5. Ajouter une méthode qui récupère le nombre de rendez-vous
6. Ajouter une méthode qui récupère les rendez-vous en fonction de plusieurs types de rendez-vous différents

2.2.3 Utiliser le bean

1. Afficher un tableau auto généré pour lequel chaque ligne est un rendez-vous, et chaque colonne une propriété (dans terminal/log). De cette manière :

```
  public void visualizeRdvs() {
    List<String> lines = new ArrayList<String>();
    for (Rendezvous rdv : rendezvousList) lines.add(rdv.toString());
    log.info(String.format("VISUALIZATION\n===== BDD CONTENT ROWS =====\n\%s",
          Joiner.on("\n=====\n").join(lines)) );
  }
```

2. Tester l'ajout de chaque fonctionnalité en visualisant au fur et à mesure (manuellement ou non)

Pour rappel, les logs se font ainsi :

```
private static final Logger log = LoggerFactory.getLogger(Application.class);

log.info("Le log fonctionne =) !")
```

Chapitre 3

Le stockage de données - DAO et JDBC Template

3.1 Exo : Stockage de l'agenda

3.1.1 Créer l'interface RendezvousDAO.java

1. Spécifier une méthode de sauvegarde d'un nouveau rendez-vous (ajout) comme ceci :

```
public interface RendezvousDAO {
  void add(Rendezvous rdv);
}
```

2. De la même façon, spécifier une méthode de mise à jour d'un rendez-vous
3. Spécifier une méthode de suppression d'un rendez-vous
4. Spécifier une méthode de requête d'un rendez-vous en fonction de son type
5. Spécifier une méthode de requête des rendez-vous (qui retourne donc une liste de rendez-vous)

```
  List<Rendezvous> findByType( String type );
```

6. Spécifier une méthode de requête des rendez-vous en fonction de leurs types

3.1.2 Créer l'implémentation RendezvousDAOImpl.java

Une classe d'implémentation d'un DAO ressemble à ceci :

```
@Transactional // évite d'avoir àgérer les transactions
@Repository // indique àspring qu'il s'agit d'un accès BDD
public class RendezvousDAOImpl implements RendezvousDAO {
    @Autowired // injection de dépendance pour un jdbctemplate global récupérant ses paramètres dans
         src/main/resources/application.properties
  JdbcTemplate jt;
...
}
```

1. Créer une méthode de sauvegarde d'un nouveau rendez-vous (ajout) avec requête SQL de cette façon :

```
   @Override
  public void add(Rendezvous rdv) {
     String sql = "INSERT INTO rendezvous(duree,lieu,type,personnes) values(?,?,?,?)";
     jt.update(sql, rdv.getDuree(), rdv.getLieu(), rdv.getType(), rdv.getPersonnes());
  }
```

2. De la même façon, créer une méthode de mise à jour d'un rendez-vous avec requête SQL
3. Créer une méthode de suppression d'un rendez-vous avec requête SQL

4. Créer une méthode de requête d'un rendez-vous en fonction de son id unique comme ceci :

```
  @Override
  public Rendezvous findById(Integer id) {
    String sql = "SELECT * FROM rendezvous WHERE id = ?";
    // mapper de lignes des éléments récupérés, automatique liés au bean Rendezvous grâce à
        BeanPropertyRowMapper<T>(T.class)
    RowMapper<Rendezvous> rowMapper = new BeanPropertyRowMapper<Rendezvous>(Rendezvous.class);
    // jdbcTemplate (jt) possède une méthode pour récupérer un seul objet : queryForObject.
    return jt.queryForObject(sql, rowMapper, id);
  }
```

5. Créer une méthode de requête des rendez-vous avec requête SQL avec row Mapper personnalisé :

```
class RDVRowMapper implements RowMapper<Rendezvous>
{
    @Override
    public Rendezvous mapRow(ResultSet rs, int rowNum) throws SQLException {
        Rendezvous rdv = new Rendezvous();
        rdv.setId(rs.getInt("id"));
        rdv.setDuree(rs.getInt("duree"));
        rdv.setLieu(rs.getString("lieu"));
        rdv.setType(rs.getString("type"));
        rdv.setPersonnes(rs.getString("personnes"));
        return rdv;
    }
}
```

6. Créer une méthode de requête des rendez-vous en fonction de leurs types avec requête SQL avec le rowMapper de votre choix.

3.1.3 Modifier le service Rendez-vous

— Adapter RendezvousService pour qu'il utilise le DAO. Par exemple :

```
  public Rendezvous getRDVByID(int id) {
    return rdvdao.findById(id);
  }
```

— Tester les fonctionnalités et visualiser les résultats et changements par terminal/logs

3.1.4 *Optionnel* Ajouter d'autres données

Un agenda a plus que de simples rendez-vous :
— Ajouter des rappels (anniversaires, jours fériés, etc.)
— Ajouter des dates butoir ayant une méthode pour une booléenne propriété *done*

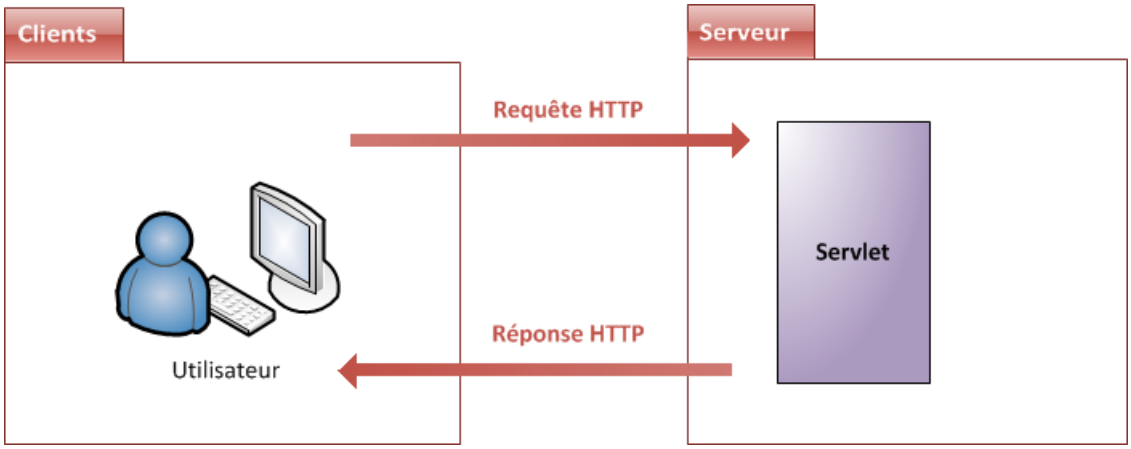
Chapitre 4

Les contrôleurs - servlets

4.1 Méthodes HTTP

GET	POST	HEAD
Pour ressources web	Pour envoi de fichiers	Pour méta-informations web
Répétée	Ponctuelle	Répétée
Taille limitée	Taille non limitée	Taille limitée
doGet()	doPost()	doHead()

4.2 Qu'est-ce qu'une servlet ?



Une servlet est :

- une classe java
- le lien central de l'application : requêtes, réponses, etc.
- un accès aux méthodes HTTP

4.3 Créer une servlet

Ici nous allons créer une servlet de la manière la plus classique qu'il soit. Nous verrons après cela comment avoir une servlet simplifiée en Spring.

4.3.1 Créer la servlet vide

Imports nécessaires :

```
import javax.servlet.http.HttpServlet;
```

1. Créer une classe src/mainjava/servlet/HelloServlet.java qui étend la classe HttpServlet
2. Sérialiser la classe

4.3.2 Déclarer la servlet à l'aide d'annotation

Imports nécessaires :

```
import javax.servlet.annotation.WebServlet;
```

1. Utiliser l'annotation @WebServlet
2. Par argument d'annotation, nommer la servlet "MaServlet"
3. Par argument d'annotation, lier la servlet aux urls "/hello" et "/bonjour"

Annotations possibles : https://docs.oracle.com/javaee/7/api/ section javax.servlet.annotation
- **@WebServlet** : pour créer une servlet et la déclarer
- **@WebFilter** : pour filtrer des catégories ("/admin/login"), des types de fichiers ("doc, txt, jpg, etc."),
- **@WebInitParam** : pour associer @WebServlet et @WebFilter
- **@WebListener** : pour créer un listener (exemple : listerner sur l'application pour savoir elle s'est arrêtée)
- **@HandlesTypes** : pour déclarer les classes
- **@MultipartConfig** : pour l'upload de fichier(s)
- **@HttpConstraint** : pour les contraintes de sécurité (SSL/TLS par exemple)
- **@HttpMethodConstraint** : pour les contraintes de sécurité sur les méthodes GET POST HEAD
- **@ServletSecurity** : pour les contraintes de sécurité sur toute la servlet

4.3.3 Remplir la servlet

Imports nécessaires :

```
import javax.servlet.http.HttpServletRequest; // pour req
import javax.servlet.http.HttpServletResponse; // pour resp
import javax.servlet.ServletOutputStream; // pour acceder au stream de sortie de la servlet
```

1. Créer une méthode doGet() ayant deux arguments : "req" de type HttpServletRequest et "resp" de type HttpServletResponse
2. Afficher la réponse (resp.getOutputStream() ;) dans le stream de sortie de la servlet (ServletOutputStream)
3. Ecrire "Bonjour le monde" dans le stream de sortie de la servlet
4. Fermer le stream
5. Dans un navigateur aller à "localhost :8080/bonjour"

4.4 Lier une servlet à une jsp (MVC)

1. Créer une nouvelle jsp basique nommée seconde.jsp dans src/main/resources/webapp
2. Ecrire un code html basique dans cette jsp

```
  <html>
<body>
   <h2>This is a superb text</h2>
   <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
```

```
    eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
    minim veniam, quis nostrud exercitation ullamco laboris nisi ut
    aliquip ex ea commodo consequat. Duis aute irure dolor in
    reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
    pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
    culpa qui officia deserunt mollit anim id est laborum.</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
    eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
    minim veniam, quis nostrud exercitation ullamco laboris nisi ut
    aliquip ex ea commodo consequat. Duis aute irure dolor in
    reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
    pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
    culpa qui officia deserunt mollit anim id est laborum.</p>
</body>
</html>
```

3. Créer une nouvelle servlet nommée "servletMVC" et comme url "/lorem"
4. Au lieu d'écrire directement dans la sortie de la servlet, transférer la requête et la réponse à une jsp existante

```
this.getServletContext().getRequestDispatcher( "/[NOM_DE_LA_JSP].jsp" ).forward( [REQUETE], [REPONSE] );
```

5. Aller à localhost :8080/lorem et voyez votre résultat en version MVC

4.5 Rediriger en fonction des paramètres de l'URL

1. Créer une nouvelle servlet nommée "ServletCondition" avec url "/condition"
2. Lire le paramètre "prenom" donné en requête url "/condition ?prenom=John"

```
// l'ensemble des parametres est une Map
Map<String, String[]> parameters = req.getParameterMap();

// il est possible d'acceder directement a un parametre
String paramPrenom = req.getParameter("MON_PARAMETRE");
```

3. Si le paramètre "prenom" est présent et non vide, transférer vers "seconde.jsp". Si non, transférer vers "index.jsp"

Cette solution ne respecte pas le modèle MVC !

4.6 Créer une servlet en Spring (AKA controller)

L'approche est beaucoup plus simple, une classe avec l'annotation correspondante devient une servlet :

1. Créer une nouvelle classe "MonControleur" qui sera votre controleur

```
@Controller // annotation qui indique qu'il s'agit d'un controleur (une servlet est crée dans l'ombre)
public class MonControleur {
    @Autowired
   ServletContext context; //injection du servlet context si besoin est pour obtenir la session en cours

   @Autowired
   MonService ms; // injection d'un service pour pouvoir l'appeler et ainsi faire agir la logique de l'app. Par
        exemple avec ms.getAllItems()
...
}
```

2. Dans cette classe, créer une méthode qui possède un mapping par annotation. Cette méthode retourne le nom de la page .jsp à afficher (plus simple que getRequestDispatcher() !)

```
@RequestMapping(value = "/", method = RequestMethod.GET) // indique que cette méthode du contrôleur (qui est une
     servlet cachée par Spring) gère la page racine "/" via la méthode GET. Cela aurait pu être "/bonjour" pour
     gérer localhost/bonjour
    public String add(){
    ...
    return "nom_de_ma_jsp_sans_extension" // ou par exemple return "index" (pour index.jsp)
    }
```

3. Tester que votre appli vous renvoie bien là où vous voulez, selon l'url fournie et la méthode fournie.

Chapitre 5

La vue - EL et JSTL

Nous utilisons ici principalement EL et JSTL combinés dans les jsp. Le langage des jsp est donc évité pour une raison simple : il outrepasse la séparation entre données, vues et modèles en permettant de mettre du java directement dans la jsp. Il n'y a aucune raison, jamais, jamais de faire cela. EL et JSTL sont donc une bonne pratique, plus simple, plus moderne (nombreux sont les framework de vue à s'en être inspiré : AngularJS, Moustache, VueJS, etc.).

5.1 EL : Expression Language

5.1.1 Qu'est-ce que l'EL ?

C'est un langage permettant de connecter facilement le contenu java et l'affichage HTML via une jsp.
Les EL permettent notamment de :

- Conditionner l'affichage
- Faire des calculs simples
- Accéder aux données / méthodes / paramètres de requête

5.1.2 Accéder aux paramètres et attributs de la requête

1. Créer une méthode nommée "ServletEL" avec url "/el" qui transfère vers "el.jsp" - ou par Spring, créer une méthode d'un contrôleur renvoyant gérant "/el" et retournant "el"
2. Créer une jsp nommée "el.jsp" et y placer le HTML suivant

```
 <!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Test des expressions EL</title>
</head>
<body>
   <p>Je m'appelle ${ A REMPLIR }</p>
</body>
</html>
```

3. Afficher la valeur du paramètre "prenom". ("Je m'appelle XXX")
4. En fonction de la valeur du paramètre "pays" afficher la nationalité à l'aide d'un attribut ("Je m'appelle XXX et je suis de nationalité XXX").

```
// donner un attribut àla requête (servlet clasique)
req.setAttribute("test", "youhou"); // pour une servlet classique

// donner un attribut àla requête (contrôleur Spring)
@RequestMapping(value = "/el", method = RequestMethod.GET)
public String add(HttpServletRequest request, Map<String, Object> model){
model.put("test", "youhou"); // model est donc une map qui représente la requête et ses informations
...
}

// jsp : acceder àl'attribut
```

```
${ test } // cela affichera "youhou"
```

5. Afficher plusieurs prénoms à partir de l'unique paramètre "prénoms". ("Mes prénoms sont XXX et XXX")

```
// donner plusieurs valeurs a un seul parametre
/el?prenoms=XXX&prenoms=XXX

// acceder au parametre àvaleur multiple
${ paramValues.XXX }

// acceder a une de ces valeurs
${ paramValues.XXX[index] }
```

5.2 JSTL

5.2.1 Qu'est-ce que JSTL ?

JSTL est une librairie regroupant plusieurs fonctions à utiliser dans la Vue. Permet de mieux respecter le modèle MVC !

5.2.2 Les principales commandes

Les balises core

Super mémo complet ici –> https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm
Quelques exemples :

```
<%-- Affiche Hello World -->
<c:out value="Hello World">

<%-- Declare une variable nommee "maVar" avec valeur 250 et une portee sur la session -->
<c:set var = "maVar" scope = "session" value = "${50+200}"/>

<%-- Affiche le contenu de la variable "maVar" (i.e. 250) -->
<c:out value="${ maVar }">

<%-- Affiche la balise <p> si "maVar" est superieure à50 -->
<c:if test = "${ maVar > 50 }">
   <p>La variable "maVar" est supérieure à50 ! Wahou!/><p>
</c:if>

<%-- Boucle for sur un indice -->
<c:forEach var = "i" begin = "1" end = "5">
   Item <c:out value = "${i}"/><p>
</c:forEach>

<%-- Iteration sur les elements d'une liste -->
<c:forEach items="${ maListe }" var="element">
     <c:out value="${ element }" />
</c:forEach>
```

Les fonctions JSTL

- fn :replace()
- fn :length()
- fn :join()
- etc.

5.2.3 Configurer JSTL

- Ajouter la librairie jstl en dépendance du projet maven (pom.xml) : https://mvnrepository.com/artifact/jstl/jstl/1.2
- Créer une nouvelle jsp nommée jstl.jsp et y ajouter tout en haut la ligne suivante :

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

5.2.4 Un affichage plus intelligent

1. Ajouter les lignes suivantes à jstl.jsp :

```
<html>
<head>
<meta charset="utf-8" />
<title>Test de JSTL</title>
</head>
<body>
   <c:out value="<p>Bonjour inconnu(e)</p>" />
</body>
</html>
```

2. Créer une servlet simple nommée "ServletJSTL" avec url "/jstl", ou par Spring un contrôleur avec RequestMapping et return adéquats.
3. Afficher "Bonjour inconnu(e)" s'il n'y a pas de paramètre "prenom" donné.
4. Afficher le drapeau du pays donné en paramètre "pays"

C'est bien plus propre et respecte mieux le MVC !

Chapitre 6

Une application de vente

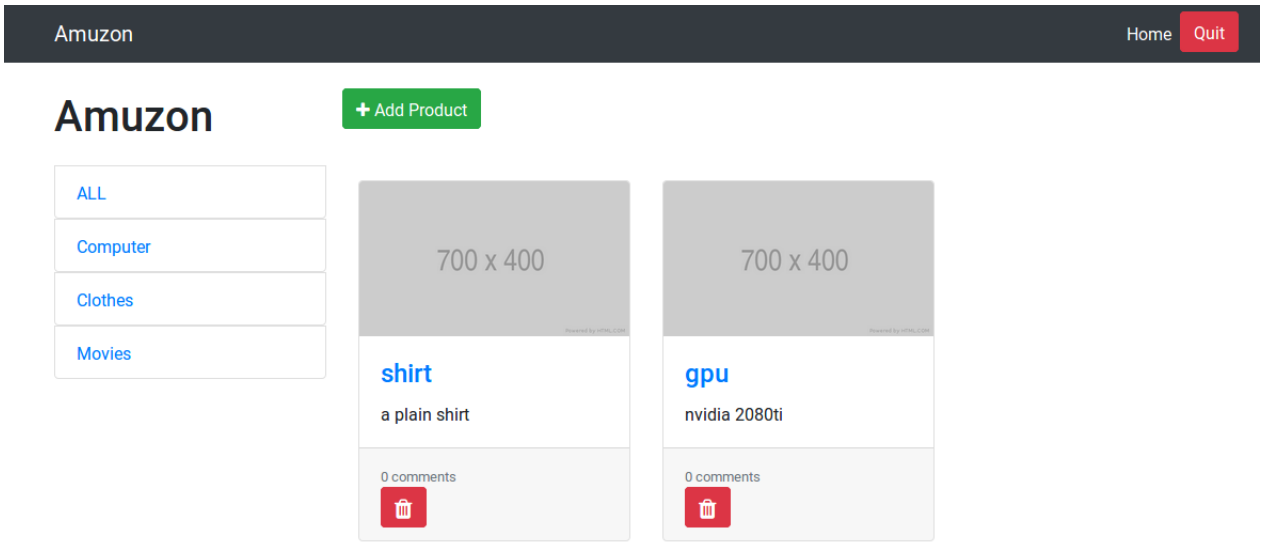


FIGURE 6.1 – Interface de l'applciation de vente

Vous connaissez :

1. La représentation des données (Bean)
2. Ses traitements (Service)
3. Sa persistance (DAO, JDBC, BDD, SQL)
4. L'aiguillage des pages (Contrôleurs, Servlets)
5. Et enfin l'affiche final (JSP, EL, JSTL)

Vous avez donc désormais le nécessaire pour faire une application ! Suivez donc les étapes.

1. Partez du squelette spring avec toutes les dépendances et les vues préparées mis à disposition à cette adresse (à venir sur github)
2. Créez un bean Product qui possède : category, title, description
3. Créez l'implémentation de l'interface suivante :

```
public interface ProductDAO {
  void add(Product product);
  void update(Product product);
  void delete(Product product);
  List<Product> findAll();
```

```
    Product findById( Integer id );
    List<Product> findByCategory( String category );
  }
```

4. Créez l'implémentation du service suivant :

```
  public interface ProductService {
    void addProduct(Product product);
    Product getProduct(int id);
    void removeProduct(int id);
    List<Product> getProducts();
    List<Product> getCategoryProducts(String category);
  }
```

5. Créer un contrôleur pour la racine qui renvoie vers la page principale en donnant la liste des produits en attribut comme ceci :

```
@Controller
public class IndexController {

  @Autowired
  private HttpSession httpSession;

  @Autowired
  ServletContext context;

  @Autowired
  ProductService ps;

   @GetMapping("/") // raccourci pour @RequestMapping( value = "/", method = RequestMethod.GET)
   public String index(Map<String, Object> model) {
     String sessionUser= (String) httpSession.getAttribute("user");
     model.put("products", ps.getProducts() );

     System.out.println("session user = " + sessionUser);
     return "homepage";
   }
}
```

6. Dans ProductController, créer une méthode ("/add") qui ajoute un produit et renvoie en attribut la liste des produits à "homepage"
7. Dans ProductController, créer une méthode ("/remove") qui supprime un produit et renvoie la liste des produits à "homepage"
8. Dans ProductController, créer une méthode ("/category") qui filtre les produits par catégorie et renvoie cette liste à "homepage"
9. Dans la de homepage.jsp, afficher tous les produits avec leurs titre, description
10. Dans la de homepage.jsp, pour chaque produit ajoute un bouton de suppression tel que :

```
  <form action="${pageContext.request.contextPath}/remove" method="POST">
    <input type="hidden" value="${product.id}" name="productId" />
    <button name="buttonRemove" type="submit" class="btn btn-danger btn-xs" >
      <i class="fa fa-trash fa-lg" aria-hidden="true"></i>
    </button>
  </form>
```

11. Dans le modal , adapter le formulaire pour rendre l'ajout de produit effectif
12. Tester les fonctionnalités, l'application est prête !