JavaEE
Développement d'application web

Aix-Marseille Université

Gaël Guibon, Omar Boucelma

2017-2018

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 [2], Struts [3], Tapestry [4]
- Python : Django, Flask, Pyramid
- Javascript : Angular, ReactJS, EmberJS

1.2 Environnement

1.2.1 Installer Java8

Ouvrir un terminal et lancer les commandes suivantes :

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

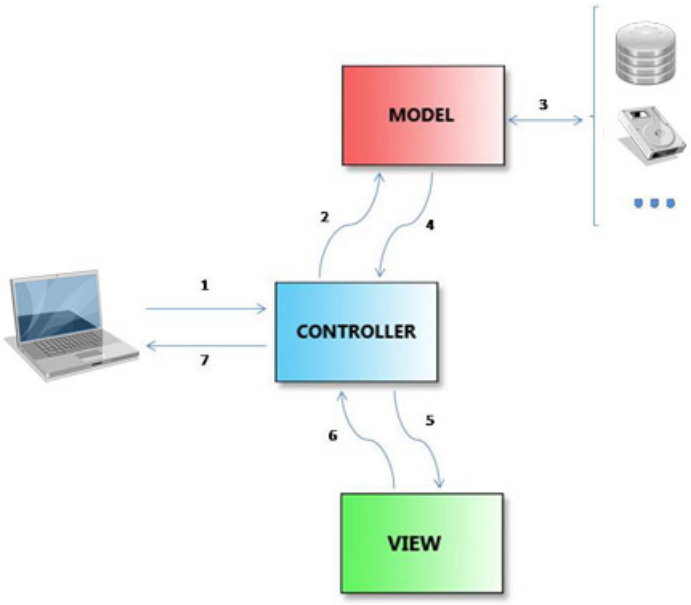

FIGURE 1.1 – Répartition MVC [1]

FIGURE 1.2 – Exemple : Messagerie SMS

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 ://www-eu.apache.org/dist/maven/maven-3/3.5.0/binaries/apache-maven-3.5.0-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
```

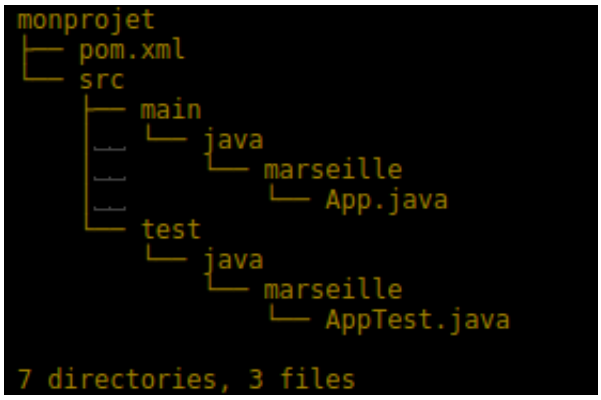

FIGURE 1.3 – Arborescence de base de maven

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 projet de base avec Git : git clone [URL]
2. Terminal : mvn package
3. Terminal : java -jar target/basicHelloWorld-jar-with-dependencies.jar OU double clic sur le jar
4. 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

Les contrôleurs - servlets

2.1 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()

2.2 Qu'est-ce qu'une servlet ?

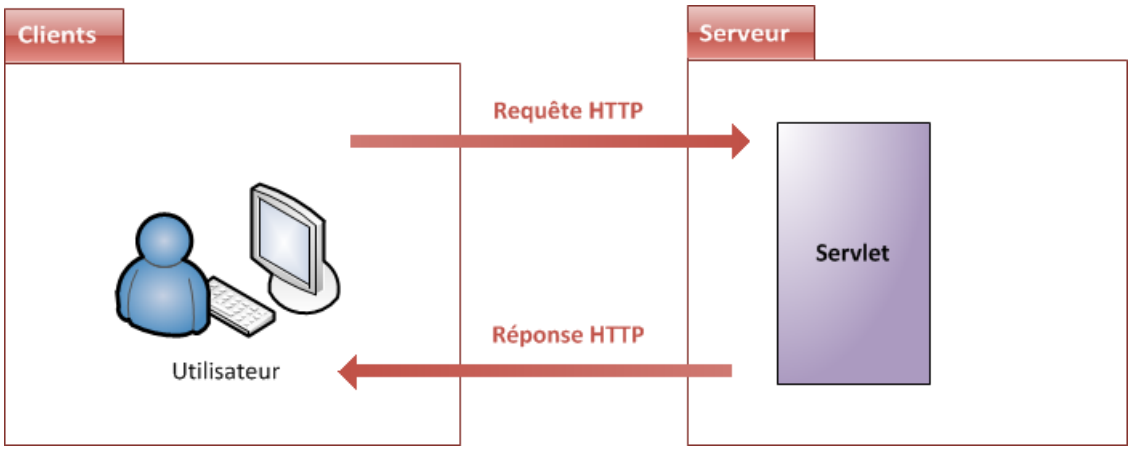

FIGURE 2.1 – Protocole HTTP et servlet [1]

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

2.3 Créer une servlet

2.3.1 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

2.3.2 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

2.3.3 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"

2.4 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

2.5 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 !

Chapitre 3

La vue - EL et JSTL

3.1 EL : Expression Language

3.1.1 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

3.1.2 Accéder aux paramètres et attributs

1. Créer une servlet nommée "ServletEL" avec url "/el" qui transfère vers "el.jsp"
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").

```
// servlet : donner un attribut a la requete
req.setAttribute("test", "youhou");

// jsp : acceder Ã  l'attribut
${ test }
```

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 Ã  valeur multiple
${ paramValues.XXX }

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

3.2 JSTL

3.2.1 Qu'est-ce que JSTL ?

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

3.2.2 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 Ã  50 -->
<c:if test = "${ maVar > 50 }">
    <p>La variable "maVar" est supérieure Ã  50 ! 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.

3.2.3 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" %>
```

3.2.4 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"
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 4

Le modèle - beans

4.1 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 ?

4.2 Exo : Agenda

4.2.1 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

4.2.2 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
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

4.2.3 Utiliser le bean

1. Afficher un tableau auto généré pour lequel chaque ligne est un rendez-vous, et chaque colonne une propriété
2. Par requête http, ajouter des rendez-vous et rafraîchir la page automatiquement

Chapitre 5

Créer le mockup d'une application de messagerie (clone de twitter)

5.1 User First ! !

— **L'utilisateur veut** voir les messages envoyés (beans + service + jstl).
— **L'utilisateur veut** qu'il soit indiqué le nom de l'auteur du message, sa date, et son contenu. (beans + jstl
— **L'utilisateur veut** pouvoir envoyer un message (param + service).
— **L'utilisateur veut** pouvoir voir son nouveau message sans avoir à recharger la page (request + servlet).
— **L'utilisateur veut** pouvoir connaître en temps réel le nombre de messages envoyés (service).
— **L'utilisateur veut** pouvoir supprimer un message (service).
— **L'utilisateur veut** pouvoir ne pas avoir besoin de sélectionner le champ pour pouvoir écrire et d'envoyer directement le message en appuyant sur "entrée" (jquery)
— **L'utilisateur veut** une belle interface.... (css + ajax + servlet)

5.2 Quelques ressources utiles

— **Bootstrap** : http://getbootstrap.com/
— **W3School** (html, css, javascripts memos et tutoriels) : https://www.w3schools.com/

Chapitre 6

Vue dynamique en AJAX (jquery)

6.1 Pourquoi AJAX ?

Jusque là lorsque nous voulions mettre à jour la vue, on devait recharger la page entière. Hors, cela entraîne plusieurs inconvénients :

- Peu ergonomique pour l'utilisateur
- Gros chargement à chaque fois (imaginez gmail qui se recharge entièrement à chaque fois...)
- Ce n'est plus une application, mais un simple site qu'on met à jour...

L'apport d'AJAX :

- Permet de lancer une requête (GET/POST/HEAD) au serveur (servlet) tout en laissant la page active et disponible le temps de la requête.
- Avec javascript ou jQuery, une fois la réponse du serveur (servlet) obtenue, on peut mettre à jour l'élément concerné par la page.

6.2 Json comme format d'échange VUE-Servlet

Ajouter les dépendance :

```
<dependency>
        <groupId>javax.json</groupId>
        <artifactId>javax.json-api</artifactId>
        <version>1.1</version>
</dependency>

<dependency>
        <groupId>org.glassfish</groupId>
        <artifactId>javax.json</artifactId>
        <version>1.0.4</version>
</dependency>
```

Dans le Service, transformer la sortie en json :

```
JsonArrayBuilder arrayBuilder = Json.createArrayBuilder(); //Construit un Array d'objets json, fonctionne comme une
        liste
JsonObjectBuilder objectBuilder = Json.createObjectBuilder(); // Construit un objet json

objectBuilder.add("author", message.getAuthor()); // ajoute un champ a l'objet json
arrayBuilder.add(objectBuilder); // ajoute un objet a l array d'objets json
```

Construire le json final et le mettre en String vers la vue :

```
// On indique que le flux est du JSON
response.setContentType("application/json; charset=UTF-8");


// On renvoi le résultat
response.getWriter().write(ms.getMessagesJson().build().toString());

// C'est tout ! Pas besoin d'envoyer vers une jsp !
```

6.3 AJAX avec jQuery

1. Ajouter dépendance jquery dans la jsp

```
<script src="${pageContext.request.contextPath}/mon/chemin/vers/jquery.min.js"></script>
```

2. Demander tous les messages en AJAX à la servlet :

```
jQuery.ajax({
        type: 'GET',
        url: '/api', // url de la servlet qui renvoie du json
        data: {
                button : "lookup" // parametres que la servlet va recuperer
        },
        success: function(result){
                console.log(result); // visualiser le resultat dans la console du navigateur
        },
        error : function(){
                console.log('oops !');
        }
});
```

6.4 Exemple pour créer et mettre à jour le DOM en jQuery

— Création d'une balise html :

```
  var monDiv = jQuery('<div></div>').attr("class", "maClasse").attr('id','id123456').text('youhou');
  // ce qui donne : <div class='maClasse' id='id123456'>youhou</div>
```

— Ajout d'un élément à un autre :

```
  $('#monidentifiant').append(monDiv);
  // ceci va ajouter le contenu de monDiv Ã  l'element dont l'identifiant est "monidentifiant"
```

— Supprimer ou vider un élément :

```
  $('#monidentifiant').remove()
  // supprime l'element
  $('#monidentifiant').empty()
  // vide le contenu de l'element
```

6.5 Améliorer l'ergonomie : keypress et focus

L'utilisateur ne doit pas avoir à sans cesse vider la zone d'écriture du message (textarea ou div). Pour cela :

1. Au chargement, immédiatement entrer dans le champ de texte (méthode focus())
2. Une fois la requête d'ajout de message effectuée, vider le champ
3. Ajouter un listener sur les touches de claviers :

```
$('#monid').keypress(function(event){}
```

4. Ajouter le message si la touche Entrée est appuyée (code 13)

```
event.keyCode //donne le code de l'evenement
```

Chapitre 7

Le stockage de données - ORM

7.1 Principe

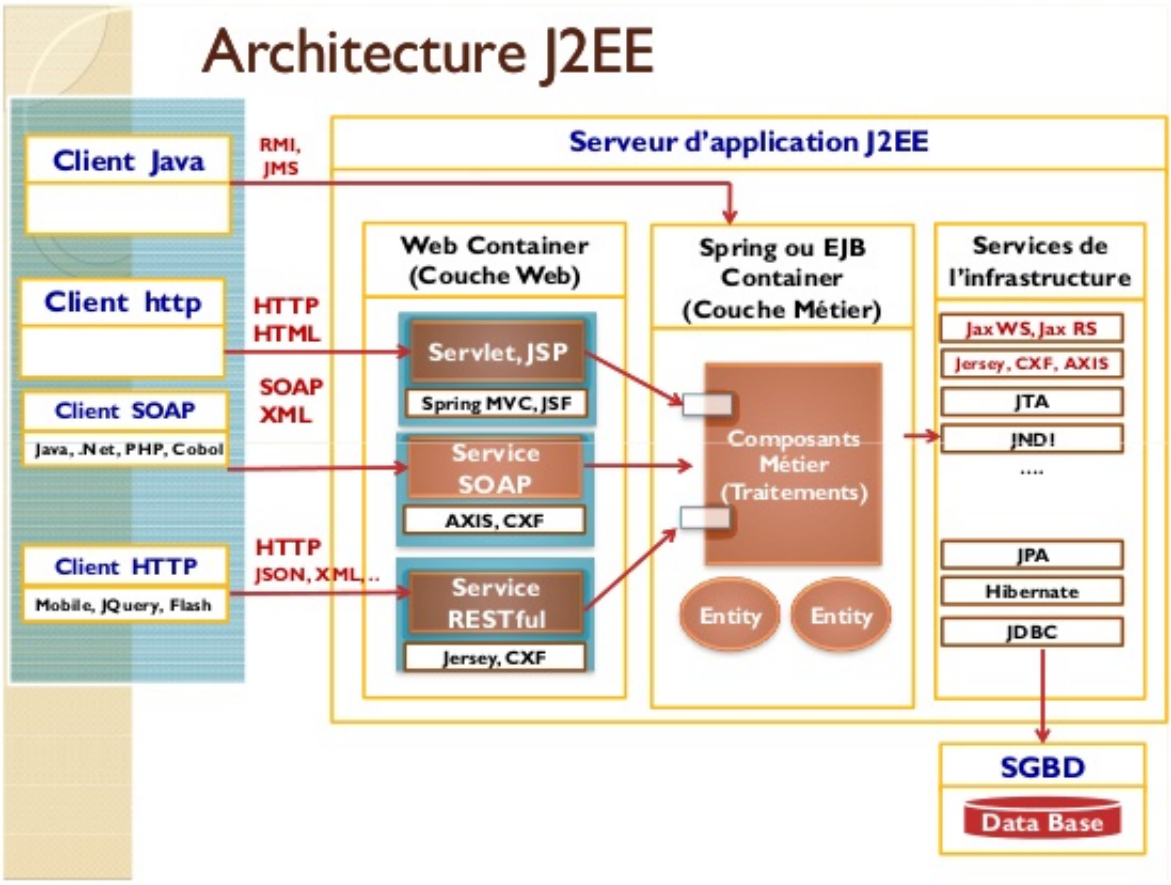

FIGURE 7.1 – Architecture finale d'un projet jee

Hiérarchie du Modèle : app > beans/service > ORM (Hibernate) > Base de données (Hsqldb)

- Permet de lier automatiquement les beans avec une base de données (HSQLDB, MySQL, etc.)
- De conserver ses données et d'encapsuler dans le service des opérations complexes SQL (vues, filtres, comptes utilisateurs, etc.)

Si vous avez bien respecté le modèle MVC, vous n'aurez pas besoin de toucher ni à la vue (jsp, css, js), ni aux controlleurs (servlets) ! !

7.2 Ressources utiles

http://hibernate.org/search/documentation/
http://www.tutorialspoint.com/hibernate/

7.3 Migrer l'application Twitter pour de l'ORM

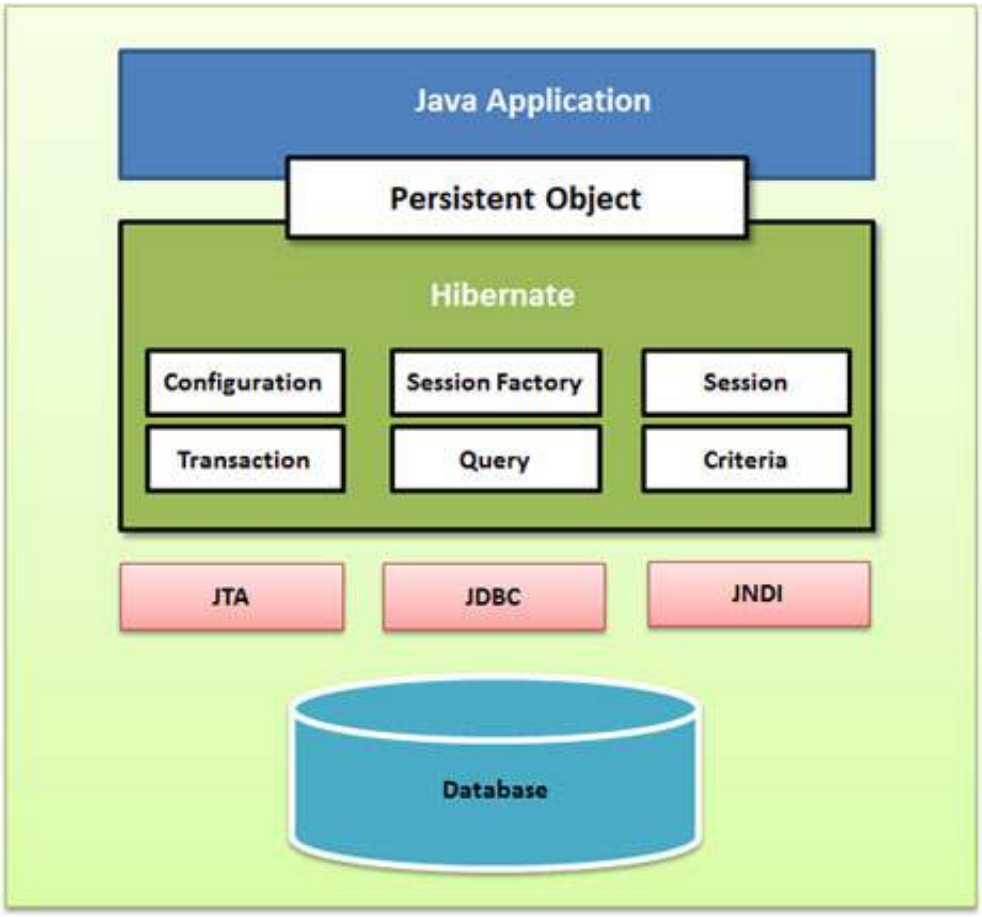

FIGURE 7.2 – Architecture finale d'un projet jee

7.3.1 Configurer Hibernate avec HSQLDB

Il d'abord commencer par configurer tout le système. Contraignant mais nécessaire.

1. Créer une sauvegarde de votre projet Twitter-like
2. Dans le pom.xml, ajouter la dépendance "hibernate-core" version "4.3.5.Final" et "hsqldb"
3. Dans resources, créer un fichier nommé "hibernate.cfg.xml" avec le contenu suivant :

```
  <?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
```

```
    <session-factory>
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.url">jdbc:hsqldb:mabdd/mabdd</property>
        <property name="connection.username">user</property>
        <property name="connection.password"></property>

        <property name="connection.pool_size">1</property>

        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
        <!-- Creer le contexte de la session -->
        <property name="current_session_context_class">thread</property>

        <property name="hibernate.cache.use_second_level_cache">false</property>
        <property name="hibernate.cache.use_query_cache">false</property>

        <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>

        <!-- Debug mode qui permet de voir toute requete SQL effectuee -->
        <property name="show_sql">true</property>


    </session-factory>
</hibernate-configuration>
```

4. Créer un fichier de configuration du cache dans "resources/" nommé "ehcache.xml" et contenant ceci :

```
 <ehcache>
    <diskStore path="java.io.tmpdir"/>
    <defaultCache
        maxElementsInMemory="1000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="true"
        />
    <cache name="Message2"
        maxElementsInMemory="1000"
        eternal="false"
        timeToIdleSeconds="3600"
        overflowToDisk="false"
        />
</ehcache>
```

7.3.2 Configurer le Mapping entre vos beans et la base de données

Là il s'agit d'une configuration très dépendance de vos données et ressources.

1. Dans le fichier "hibernate.cfg.xml", ajouter le(s) mapping(s) de votre/vos bean(s) dans la balise <session-factory> :

```
<mapping class ="beanpackage.MonSuperBean" resource="MonSuperBean.hbm.xml"/>
```

2. Créer un fichier "MonBean.hbm.xml" dans le dossier "resources"
3. Y indiquer les informations relatives au bean et à la stratégie voulue dans la base de données :

```
  <?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate␣Mapping␣DTD␣3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class entity-name="beans.MaClasseDuBean" name="beans.MaClasseDuBean" table="TABLEDUBEAN"
         dynamic-update="false" dynamic-insert="false">
        <cache usage="read-write"/>

    </class>
</hibernate-mapping>
```

4. Dans ce même fichier, remplir la balise <class> avec les informations des propriétés du bean :
   — Indiquer un ID auto incrémenté :

```
<id name="id" column="ID" type="java.lang.Long">
        <generator class="increment"></generator>
    </id>
```

   — Indiquer pour chaque propriété ses valeurs et types. Exemple :

```
<property name="nom" type="java.lang.String" update="true" insert="true" access="property" column
    ="nom"/>
```

7.3.3 Créer une classe utilitaire pour la base de données

Cette classe va permettre de lancer la base de données et de s'y connecter. D'y effectuer toutes sortes d'opérations, etc.

1. Créer une classe "HibernateUtil" dans le package dédié au Modèle
   — Le constructeur doit contenir l'ouverture de la session :

```
Configuration configuration = new Configuration();
configuration.configure("hibernate.cfg.xml");
StandardServiceRegistryBuilder ssrb = new StandardServiceRegistryBuilder().applySettings(configuration.
    getProperties());
SessionFactory sessionFactory = configuration.buildSessionFactory(ssrb.build());
session = sessionFactory.openSession();
```

   — Et le démarrage de la base de données HSQDB :

```
Class.forName("org.hsqldb.jdbcDriver");
System.out.println("Driver Loaded.");
String url = "jdbc:hsqldb:mabdd/mabdd";

Connection conn = DriverManager.getConnection(url, "sa", "");
System.out.println("Got Connection.");
st = conn.createStatement();
```

   — La classe doit posséder un getter pour la session
   — La classe doit posséder une méthode d'exécution de commande SQL (méthode "executeUpdate" de l'objet Statement)

7.3.4 Lier le service à la base de données

Enfin, le service doit être adapté pour ne plus accéder aux données en mémoire mais directement dans la base de données.
Le service doit pouvoir se passer de la liste, ou HashMap, de messages. **Attention ! !** Chaque opération d'écriture ou de lecture est considérée comme une **Transaction**. Une transaction doit toujours être commencée, puis commitée, et enfin la session doit être cleanée :

```
session.beginTransaction();
//MON_OPERATION
session.getTransaction().commit();
session.flush();
```

Suivant ce principe, le service doit :

1. Créer une table si elle n'existe pas par commande SQL (dans le constructeur)
2. Avoir une méthode de lecture des messages en récupérant une liste d'objets directement depuis la base de données :

```
List<MonBean> listeDeBeans = new ArrayList<MonBean>(session.createQuery("from beans.MonBean").list
    ());
```

3. Avoir une méthode d'ajout de message qui écrit dans la base de données :

```
session.save(monBean);
```

4. Avoir une méthode de suppression de message, grâce à son identifiant dans la base de données (méthode "delete" de l'objet Session)

Vous pouvez ENFIN tester l'application !

7.4 Améliorer l'application grâce aux données

7.4.1 Login simple

Ajouter une page de connection.
Exemple de page de connection bootstrap simple : https://www.w3schools.com/bootstrap/bootstrap_forms.asp.

- Chaque utilisateur (nouveau bean User, nouvelle table USER) doit posséder : identifiant et mot de passe
- L'identifiant/nom doit être la clef primaire :

```
// dans User.hbm.xml
<id name="name" column="name" type="java.lang.String"><generator class="assigned"/></id>
```

- Chaque utilisateur peut se connecter avec identifiant et mot de passe (UserService)
- L'information de connection doit être conservée dans la session :

```
import javax.servlet.http.HttpSession;
.
.
HttpSession session = request.getSession(); // creer/recuperer la session
session.getAttribute("user"); // recuperer une info dans la session
session.setAttribute("user", "JonSnow"); // ajouter une info dans la session
```

Gérer l'aiguillage en fonction de la session en cours :

- Si l'utilisateur arrive sur la page d'accueil sans session –> renvoyer vers la page de login
- Si l'utilisateur arrive sur la page de login avec une session –> renvoyer vers la page d'accueil et afficher son nom
- Ajouter un bouton de déconnection qui renvoie vers la page d'accueil.

Tester l'application avec plusieurs comptes simultanés.

7.4.2 Fonction "j'aime"

Ajouter la fonction "j'aime"/"like" pour chaque message.

- Un like est lié à son utilisateur et au message : plusieurs approches possibles (BDD, vue+session)
- Les personnes ayant aimé un message sont visibles en "hover" du message concerné.

Chapitre 8

Améliorer l'ergonomie

Pour que l'application soit plus conforme aux standards d'ergonomie actuels, il faut veiller à plusieurs choses.

1. Le champ de texte doit être focused (.focus()) automatiquement.
2. La touche Entrée doit permettre d'envoyer un message
3. Lors de l'envoi de message, le champ de texte doit se vider et s'auto focus.
4. Les derniers messages doivent être en haut (.prepend()).
5. Un nouveau message envoyé par l'utilisateur X doit immédiatement être visible en AJAX par l'utilisateur Y.

```
// javascript
setInterval();
```

Pour le Jar exécutable Le navigateur par défaut doit pouvoir se lancer dès l'ouverture du programme.

```
// pur java
Desktop desktop = Desktop.getDesktop();
desktop.browse(URI);
```

Pour le jar exécutable Un bouton de fermeture du programme doit être disponible pour facilement libérer le port.

```
Main.hibernateUtil.quit();
System.exit(0);
```

Chapitre 9

Déployer l'application sur le LAN

9.1 Adapter HSQLDB

Actuellement l'accès à la base de données ne peut provenir que d'une source. Il faut adapter hsql en utilisant sa version serveur.
S'aider de la documentation ! http://hsqldb.org/doc/guide/ch01.html

- Changer l'url d'hsqldb dans le main et hibernate.cfg.xml
- Lancer la base de données en mode serveur.
- Se connecter et visualiser la base de données en double cliquant sur hsqldb.jar (il est possible enregistrer les settings)

9.2 Adapter le pom et les urls

1. Adapter le pom pour déployer un .war
2. Modifier le root "/" dans les forwards d'urls.
3. Vérifier la gestion des transactions dans les services (begin, commit, flush)
4. Compiler et packager le war

9.3 Déployer le war dans un tomcat standalone

1. Installer tomcat8 : sudo apt-get install tomcat8
2. Copier le .war dans le dossier webapps de tomcat (souvent /var/lib/tomcat8/webapps)
3. Vérifier que la base de donnée en mode server est lancée
4. Tester l'application à partir d'un autre appareil sur le même réseau (LAN).

Créez une page d'inscription et testez l'applications des autres.

9.4 Ouvrir l'application au monde

Le déploiement effectué ne concerne que le LAN. Pour l'ouvrir au reste du monde il faut configurer un transfert d'adresse vers une adresse ip fixe, sur le routeur ou le serveur apache.
Pour des raisons de temps (cela sort du cadre de ce cours) et de sécurité nous n'allons pas le faire ici. Sachez toutefois que si vous hébergez votre application chez un fournisseur, alors elle marchera de la même manière que sur le LAN.

Chapitre 10

[WIP] Framework Spring