Mise en place du poste de dev :

1. Installer WampServer du dossier « outils », démarrer puis jouer tous les scripts du dossier « scripts bdd »
2. Les répertoires :

* eclipse : contient toutes les sources et configuration eclipse pour du développement jee
* outils JEE : contient les sources des outils utilisés dans eclipse (maven, server apache tomcat, jdk, etc.)
* workspace : dossier de travail eclipse contenant également les sources du projet

1. Configuration du projet :

* Sous eclipse, ‘File -> new ->Maven Project ->next -> choisir l’archetype: maven-archetype-webapp -> configurer le projet -> Finish
* **Configuration du pom.xml**

On ajoute les dependances junit ( test unitaire), javax.servlet-api (pour la gestion des servlets via un fichier de conf), eclipselink, javax.persistence, mysql-connector-java ( liaison avec des bdd), springframework (Architecture spring MVC), jta ( java transaction api pour la gestion des transactions bdd), javax.mail pour l’envoi de mail, slf4j ( pour les logs), etc.

* **Conf bdd :**

Src/main/resources/META-INF/persistence.xml

* **Conf mvc :**
* Src/main/webapp/WEB-INF/web.xml contenant de la conf spring côté servlet-dispatcher ( gestion via des controllers des pages web)
* Src/main/web app/WEB-INF/dispatcher-servlet.xml -> configuration des accès bdd via des entity manager, conf du répertoires contenant les pages web et conf du package contenant les controllers.

**Architecture :**

1. **Source Java** :

* com.holySearch.bean -> package regroupant tous les objets java dits entités décrivant les tables en bdd via la techno hibernate jpa. Le lien est effectué via les annotations spring du genre : @Entity, @Table(name = ‘’nom\_table\_en\_bdd’’), etc.
* Com. holySearch.controller -> il s’agit du package contenant le controller (Main Controller) qui se charge lui d’effectuer la gestion des actions et redirection entre les pages web.
* Com. holySearch.dao -> package des classes d’accès à la bdd (DAO : data access object), chaque class définira un entityManager (manager d’entités).
* Com. holySearch.forms -> package des objets java de type formulaire qui seront utilisés par les pages web.
* Com. holySearch.transfert.object -> package d’objets utilisés uniquement au niveau de la couche service (TO : Transfert Object).
* Com. holySearch.mapper -> package contenant la classe MapperUtils qui fera le mapping entre les objets Forms utilisés niveau controller et les objets TO utilisés côté service.
* Com. holySearch.mail -> packqge d’envoi des mails de création de compte ou de réinitialisation du mot de passe.

**Cas pratique : Page de connexion**

* 1ere étape : création de la page index.jsp dans le dossier src/main/webapp/WEB-INF/vues/jsp
* On inclut les librairies jstl(gestion d’objet java côté jsp) et spring
* On inclut les divers css bootstrap, css personnalisé, etc.
* On inclut les divers javascript
* On créer le code web correspondant avec du bootstrap pour que le site sot responsive.
* On ajoute le bloc pour le formulaire de connexion :

<form :form modelAttribute=”connexionForm” method=”post” action=”connexion”>

…..

<inputtype=*"submit"* value=*"Connexion"* />

</form:form>

* 2eme etape : Création de l’objet Form représentant le formulaire de connexion

On créé la classe ConnexionForm.java dans le package com. holySearch.forms avec les annotations @NotEmpty qui feront que spring vérifiera que les login et password ne soit jamais vides.

**package** com.holySearch.forms;

**import** org.hibernate.validator.constraints.NotEmpty;

**public** **class** ConnexionForm {

@NotEmpty

**private** String login;

@NotEmpty

**private** String password;

**public** **void** setLogin(String login) {

**this**.login = login;

}

**public** String getLogin() {

**return** login;

}

/\*\*

\* **@return** the password

\*/

**public** String getPassword() {

**return** password;

}

/\*\*

\* **@param** password the password to set

\*/

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

* 2eme etape : Ajout de la méthode de gestion de l’action action=”connexion” dans la classe MainController.java
* value : /connexion vient de action=”connexion” dans la jsp
* method = RequestMethod.***POST*** vient demethod=”post” dans la jsp
* **public** String checkExistence(@Valid @ModelAttribute(value = "connexionForm") **final** ConnexionForm pConnexionForm,

**final** BindingResult pBindingResult, **final** ModelMap pModel) **throws** UnsupportedEncodingException

fera le lien avec l’objet java transmit par la jsp (modelAttribute=”connexionForm”) et le transformera en un objet de la classe ConnexionForm avec une validation des annotations du type @NotEmpty.

Le paramètre pModel de type ModelMap est celui qui permettra depuis le controller de transmettre un objet java à la jsp suivante.

* Return redirect -> redirige vers la page de nom search.jsp ou index.jsp selon si l’utilisateur a les droits de connexion ou non.

@RequestMapping(value = "/connexion", method = RequestMethod.***POST***)

**public** String checkExistence(@Valid @ModelAttribute(value = "connexionForm") **final** ConnexionForm pConnexionForm,

**final** BindingResult pBindingResult, **final** ModelMap pModel) **throws** UnsupportedEncodingException {

String redirect = "index";

**if** (mUserService.userBeanExist(pConnexionForm.getLogin(), pConnexionForm.getPassword())) {

pModel.addAttribute("identifiant", pConnexionForm.getLogin());

redirect = "search";

}

**return** redirect;

}

3eme etape : Créer le service qui fera UserService qui xposera des méthodes de vérification du couple (identifiant, mot de passe), de création d’un compte utilisateur, de réinitialisation du mot de passe.

Cette classe ne manipulera que des objets de type TO et sera liée au DAO userBeanDAO pour effectuer des actions en bdd via l’entité User.

Une lecture plus approfondie du code source permettra de faire la lumière sur certaines zones d’ombre qui persistent.