# Programmation EJB3 JMS sous Widfly 12.0

## Introduction

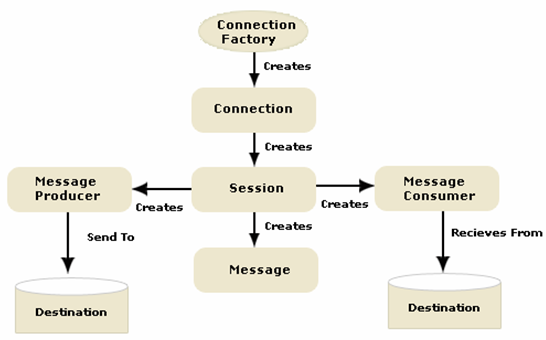
Dans ce sujet, vous allez découvrir la création d'un EJB3 JMS sous le serveur d’application Widfly 12.0

Java Message Service (JMS) API est un système de messages qui permet à une application, en utilisant les composants Java 2 Platform Enterprise Edition (J2EE), de créer, d’envoyer, de recevoir et de lire des messages. Cette API offre **une communication distribuée, synchrone/asynchrone fiable. Dans ce tp, nous allons**  consommer un message en mode **synchrone (receive)**.

JMS définit plusieurs entités :

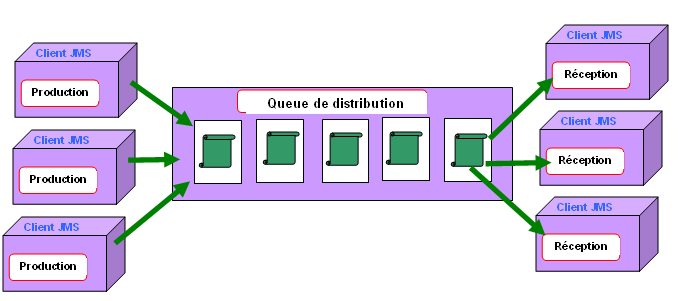
* un **provider JMS** : outil qui implémente l'API JMS pour échanger les messages : ce sont les « brokers » de messages
* un **client JMS** : composant écrit en java qui utilise JMS pour émettre et/ou recevoir des messages.
* un **message** : données échangées entre les composants

Le processus d’échange de messages peut être montré dans la figure suivante :



### Exemple d’application : Mode Publish/Souscribe

Le mode publication/abonnement repose sur le concept de sujets (Topic). Plusieurs producteurs peuvent placer les messages pour divers destinataires dans une queue. L'émetteur du message ne connaît pas les destinataires qui se sont abonnés



Ce tp vous présente une application qui permet à des clients de s’enregistrer à une manifestation. Les composants sont :

* File d’attente TOPIC sous le serveur WildFly 12.0 : InscriptionTopic
* Une base de données Mysql : lesInscriptions table inscription
* Une application Cliente qui va produire les informations à enregistrer
* Une application serveur qui va le réceptionner pour les insérer dans la base de données

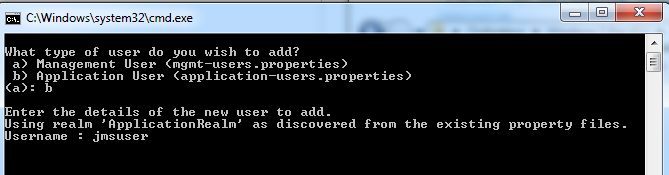
# Réalisation du TP

## Installation d’une queue sous le serveur Widfly 12.0

La création d’une queue sous JBOSS demande l’écriture d’un fichier XML qui définit la queue de publication.

|  |
| --- |
| On lance un client jboss-cli On se connecte au serveur    On exécute la commande suivante :  [standalone@localhost:9990 /]  **jms-topic add --topic-address=InscriptionTopic** **--entries=** **java:jboss/exported/topic/InscriptionTopic**    **On contrôle l’entré de notre queue topic dans le fichier standalone.xml**    Sous le serveur, on peut contrôler si la queue de type Topic est activée |

### Création d’un utilisateur : jmsuser/ jmsepul98! add-user



Ajoutez-le dans le groupe guest

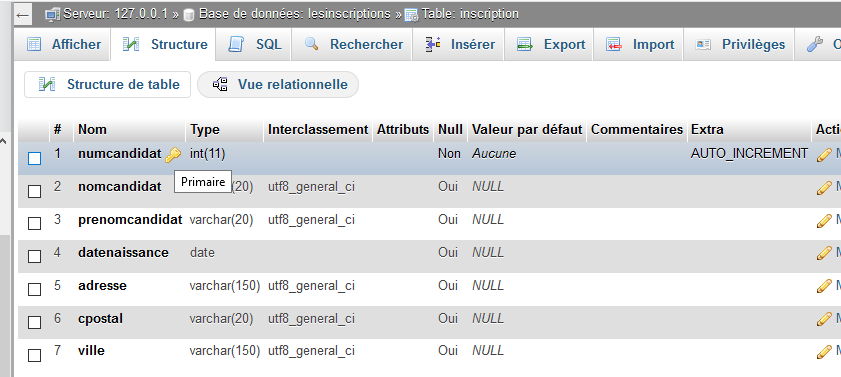
Le fichier application-roles sera mis à jour.

Si cet utilisateur a été créé dans le tp précédent : message et image en asynchrone, vous n’avez pas à le refaire.

## Base de données sous Mysql : lesinscriptions ;

Sous mysql, créez une nouvelle base de données nommée lesinscriptions avec une table nommée inscription.

Structure de la table inscription.

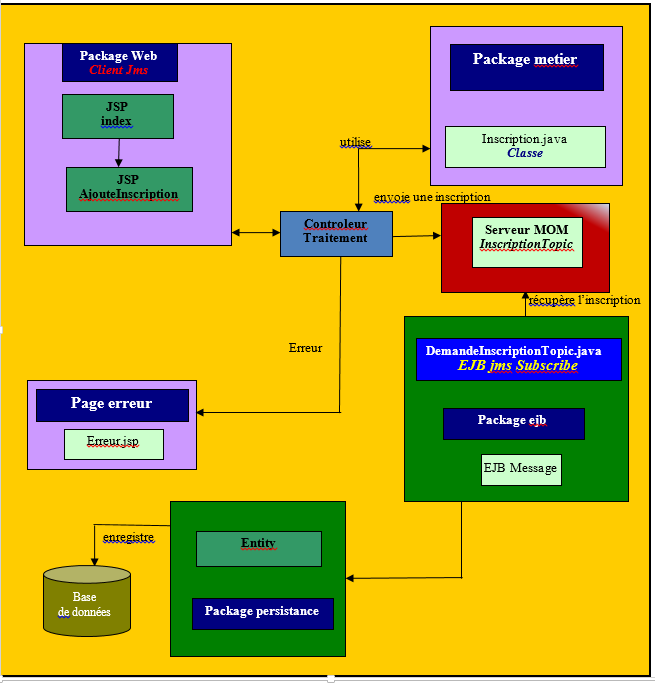


# Application Cliente

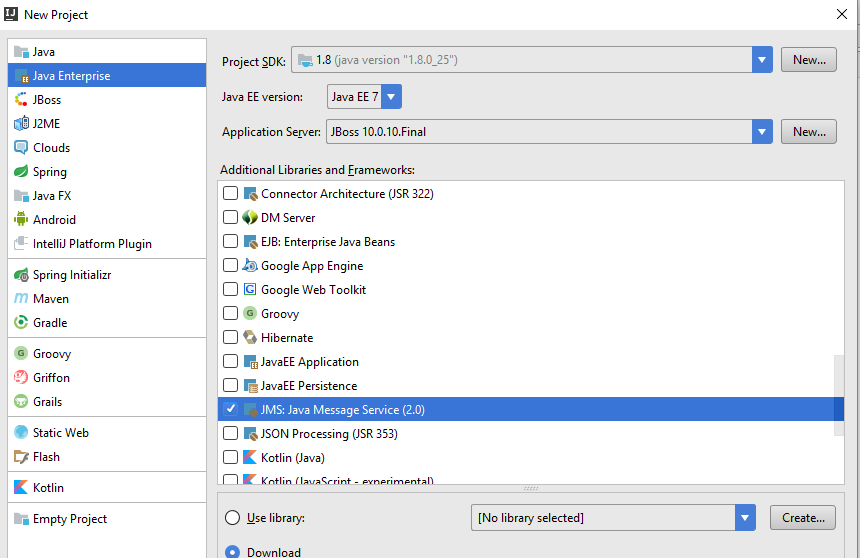
L’application cliente va saisir les informations d’un utilisateur qui veut s’enregistrer à la manifestation.

## Composants de l’application :

* + Architecture de l’application

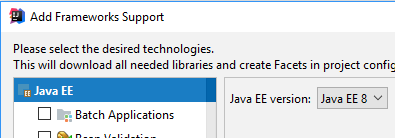


## Réalisation de l’application InscriptionTopic sous Intellij

Démarrez un nouveau projet  de type  **Java Entreprise / jms** 

Nom du projet : **AppClientInscription**

Ajouter ensuite les composants maven et JPA (persistance des données)



Voici le fichier pom.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>groupId</**groupId**>  
 <**artifactId**>InscriptionTopic</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**project.reporting.outputEncoding**>UTF-8</**project.reporting.outputEncoding**>  
 <**java.version**>1.8</**java.version**>  
 <**hibernate.version**>5.2.2.Final</**hibernate.version**>  
 </**properties**>  
  
 <**dependencies**>  
 *<!-- https://mvnrepository.com/artifact/javax.ejb/javax.ejb-api -->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>5.1.38</**version**>  
 </**dependency**>  
  
 *<!-- https://mvnrepository.com/artifact/javax.ejb/javax.ejb-api -->* <**dependency**>  
 <**groupId**>javax.ejb</**groupId**>  
 <**artifactId**>javax.ejb-api</**artifactId**>  
 <**version**>3.2.2</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.hibernate</**groupId**>  
 <**artifactId**>hibernate-core</**artifactId**>  
 <**version**>${hibernate.version}</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.hibernate</**groupId**>  
 <**artifactId**>hibernate-entitymanager</**artifactId**>  
 <**version**>${hibernate.version}</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
  
 *<!-- https://mvnrepository.com/artifact/javax.jms/javax.jms-api -->* <**dependency**>  
 <**groupId**>javax.jms</**groupId**>  
 <**artifactId**>javax.jms-api</**artifactId**>  
 <**version**>2.0.1</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>javax.persistence</**groupId**>  
 <**artifactId**>persistence-api</**artifactId**>  
 <**version**>1.0.2</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>javax</**groupId**>  
 <**artifactId**>javaee-api</**artifactId**>  
 <**version**>7.0</**version**>  
 </**dependency**>  
  
  
 </**dependencies**>  
</**project**>

Voici les packages à créer

|  |  |
| --- | --- |
|  | Dans le package metier  Vous avez deux classes :   * Inscription classe pour récupérer l’inscription     InscriptionEntity : classe générée en lien avec la BD |

#### Classe DemandeInscriptionTopic

On crée cette classe en ajoutant un EJB de type Message Driven Bean

**package** ejb;

**import** javax.ejb.ActivationConfigProperty;

**import** javax.ejb.MessageDriven;

**import** javax.jms.Message;

**import** javax.jms.MessageListener;

/\*\*

\* Message-Driven Bean implementation class for: InscriptionTopic

\*/

@MessageDriven(activationConfig = {

@ActivationConfigProperty(propertyName = "destination", propertyValue = "java:jboss/exported/topic/InscriptionJmsTopic"),

@ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic") }, mappedName = "InscriptionJmsTopic")

**public** **class** DemandeInscriptionTopic **implements** MessageListener {

/\*\*

\* Default constructor.

\*/

**public** DemandeInscriptionTopic() {

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@see** MessageListener#onMessage(Message)

\*/

**public** **void** onMessage(Message message) {

// **TODO** Auto-generated method stub

}

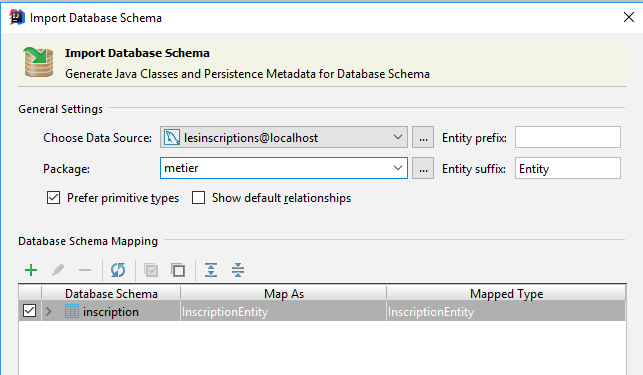
}

### Classe Inscription dans le package metier

On ajoute la classe inscription dans le package metier.

**package** metier;  
  
  
**import** java.io.Serializable;  
**import** java.sql.Date;  
  
  
**public class** Inscription **implements** Serializable {  
 **private int numcandidat**;  
 **private** String **nomcandidat**;  
 **private** String **prenomcandidat**;  
 **private** Date **datenaissance**;  
 **private** String **adresse**;  
 **private** String **cpostal**;  
 **private** String **ville**;  
  
  
 **public int** getNumcandidat() {  
 **return numcandidat**;  
 }  
  
 **public void** setNumcandidat(**int** numcandidat) {  
 **this**.**numcandidat** = numcandidat;  
 }  
  
 **public** String getNomcandidat() {  
 **return nomcandidat**;  
 }  
  
 **public void** setNomcandidat(String nomcandidat) {  
 **this**.**nomcandidat** = nomcandidat;  
 }  
  
  
 **public** String getPrenomcandidat() {  
 **return prenomcandidat**;  
 }  
  
 **public void** setPrenomcandidat(String prenomcandidat) {  
 **this**.**prenomcandidat** = prenomcandidat;  
 }  
  
  
 **public** Date getDatenaissance() {  
 **return datenaissance**;  
 }  
  
 **public void** setDatenaissance(Date datenaissance) {  
 **this**.**datenaissance** = datenaissance;  
 }  
  
  
 **public** String getAdresse() {  
 **return adresse**;  
 }  
  
 **public void** setAdresse(String adresse) {  
 **this**.**adresse** = adresse;  
 }  
  
  
 **public** String getCpostal() {  
 **return cpostal**;  
 }  
  
 **public void** setCpostal(String cpostal) {  
 **this**.**cpostal** = cpostal;  
 }  
  
  
 **public** String getVille() {  
 **return ville**;  
 }  
  
 **public void** setVille(String ville) {  
 **this**.**ville** = ville;  
 }  
}

Génération de la classe InscriptionEntity



On ajoute la possibilité d’utiliser l’auto incrément de Mysql

*// Auto incrémlent pour Mysql*@Id @GeneratedValue(strategy=GenerationType.***IDENTITY***)  
**public int** getNumcandidat() {  
 **return numcandidat**;  
}

### EJB de type JMS

On complète la classe DemandeInscriptionTopic en ajoutant un code sur la méthode onMessage afin de récupérer un objet inscription pour le passer à la base de données.

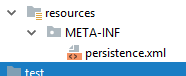
**package** ejb;  
  
**import** meserreurs.MonException;  
**import** metier.\*;  
  
**import** javax.annotation.Resource;  
**import** javax.ejb.ActivationConfigProperty;  
**import** javax.ejb.MessageDriven;  
**import** javax.ejb.MessageDrivenContext;  
**import** javax.jms.JMSException;  
**import** javax.jms.Message;  
**import** javax.jms.MessageListener;  
**import** javax.jms.ObjectMessage;  
**import** javax.naming.NamingException;  
**import** java.io.BufferedWriter;  
**import** java.io.FileNotFoundException;  
**import** java.io.FileWriter;  
**import** java.io.IOException;  
**import** java.text.SimpleDateFormat;  
**import** java.util.Date;  
  
**import** service.EnregistreInscription;  
  
*/\*\*  
 \* Message-Driven Bean implementation class for: DemandeInscriptionTopic  
 \*/  
// On se connecte à la file d'attente InscriptionTopic*@MessageDriven(activationConfig = {  
 @ActivationConfigProperty(propertyName = **"destination"**, propertyValue = **"java:jboss/exported/topic/InscriptionJmsTopic"**),  
 @ActivationConfigProperty(propertyName = **"destinationType"**, propertyValue = **"javax.jms.Topic"**)}, mappedName = **"InscriptionJmsTopic"**)  
**public class** DemandeInscriptionTopic **implements** MessageListener {  
  
 @Resource  
 **private** MessageDrivenContext **context**;  
  
 */\*  
 \* Default constructor.  
 \*/* **public** DemandeInscriptionTopic() {  
 *//* ***TODO Auto-generated constructor stub*** }

La méthode onMessage va récupérer les informations par l’intermédiaire de la classe Inscription et les transférer à la classe InscriptionEntity et on appelle la méthode insertionInscription de la classe Service.

*/\*\*  
 \** ***@see*** *MessageListener#onMessage(Message)  
 \*/* **public void** onMessage(Message message) {  
 *//* ***TODO Auto-generated method stub* boolean** ok = **false**;  
 *// On gère le message récupéré dans le topic* **try** {  
 *// On transforme le message en demande d'inscription* **if** (message != **null**) {  
 System.***out***.println(**"je suis là "**);  
 ObjectMessage objectMessage = (ObjectMessage) message;  
 Inscription uneInscription = (Inscription) objectMessage.getObject();  
 *// On insère cette demande d'inscription dans la base de données  
 // on s'assure que l'écriture ne se fera qu'une fois.* message = **null**;  
 **try** {  
 *// on construit un objet Entity* InscriptionEntity uneInsEntity = **new** InscriptionEntity();  
 *// on tansfère les données reçues dans l'objet Entity* uneInsEntity.setNomcandidat(uneInscription.getNomcandidat());  
 uneInsEntity.setPrenomcandidat(uneInscription.getPrenomcandidat());  
 uneInsEntity.setCpostal(uneInscription.getCpostal());  
 uneInsEntity.setVille(uneInscription.getVille());  
 uneInsEntity.setAdresse(uneInscription.getAdresse());  
 uneInsEntity.setDatenaissance(uneInscription.getDatenaissance());  
 EnregistreInscription uneE = **new** EnregistreInscription();  
 uneE.insertionInscription(uneInsEntity);  
 } **catch** (NamingException er) {  
 EcritureErreur(er.getMessage());  
 } **catch** (MonException e) {  
 EcritureErreur(e.getMessage());  
 } **catch** (Exception ex) {  
 EcritureErreur(ex.getMessage());  
 }  
 }  
  
 } **catch** (JMSException jmse) {  
 System.***out***.println(jmse.getMessage());  
 EcritureErreur(jmse.getMessage());  
 **context**.setRollbackOnly();  
 }  
 }  
  
 */\*\*  
 \* Permet d'enregistrer une erreur dans un fichier log  
 \*  
 \** ***@param message*** *Le message d'erreur  
 \*/* **public void** EcritureErreur(String message) {  
 BufferedWriter wr;  
 String nomf = **"erreurs.log"**;  
 Date madate = **new** Date();  
 SimpleDateFormat sdf = **new** SimpleDateFormat(**"dd/MM/yy hh:mm"**);  
 **try** {  
 *// On écrit à la fin du fichier* wr = **new** BufferedWriter(**new** FileWriter(nomf, **true**));  
 wr.newLine();  
 wr.write(sdf.format(madate));  
 wr.newLine();  
 wr.write(message);  
 wr.close();  
 } **catch** (FileNotFoundException ef) {  
 ;  
 } **catch** (IOException eio) {  
 ;  
 }  
 }  
}

### Classe Service

Pour insérer les données, on va utiliser un objet EntityManager.



*<?***xml version="1.0" encoding="UTF-8"***?>*<**persistence xmlns="http://java.sun.com/xml/ns/persistence" version="2.0"**>  
 <**persistence-unit name="PInscription"**>  
 <**provider**>org.hibernate.jpa.HibernatePersistenceProvider</**provider**>  
 <**class**>metier.InscriptionEntity</**class**>  
 <**properties**>  
 <**property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/lesinscriptions"**/>  
 <**property name="hibernate.connection.driver\_class" value="com.mysql.jdbc.Driver"**/>  
 <**property name="hibernate.connection.username" value="userepul"**/>  
 <**property name="hibernate.connection.password" value="epul"**/>  
 <**property name="hibernate.archive.autodetection" value="class"**/>  
 <**property name="hibernate.show\_sql" value="true"**/>  
 <**property name="hibernate.format\_sql" value="true"**/>  
 <**property name="hbm2ddl.auto" value="update"**/>  
 </**properties**>  
 </**persistence-unit**>  
</**persistence**>

#### EntityManager

On doit créer un objet de type EntityManager dans une classe qui n’appartient pas à un bean session, donc on ne peut pas utiliser l’injection de contrôle avec les annotations.

**package** service;  
  
**import** meserreurs.MonException;  
**import** javax.persistence.\*;  
**import** metier.InscriptionEntity;  
  
**public class** EnregistreInscription {  
  
 *// on décvlare un EntityManager* **private** EntityManagerFactory **factory**;  
 **private** EntityManager **entityManager**;  
  
 **public void** insertionInscription(InscriptionEntity uneI) **throws** Exception, MonException {  
  
 *// On instancie l'entity Manager* **factory** = Persistence.*createEntityManagerFactory*(**"PInscription"**);  
 **entityManager** = **factory**.createEntityManager();

La suite de la méthode est classique, il faut créer une transaction et la valider.

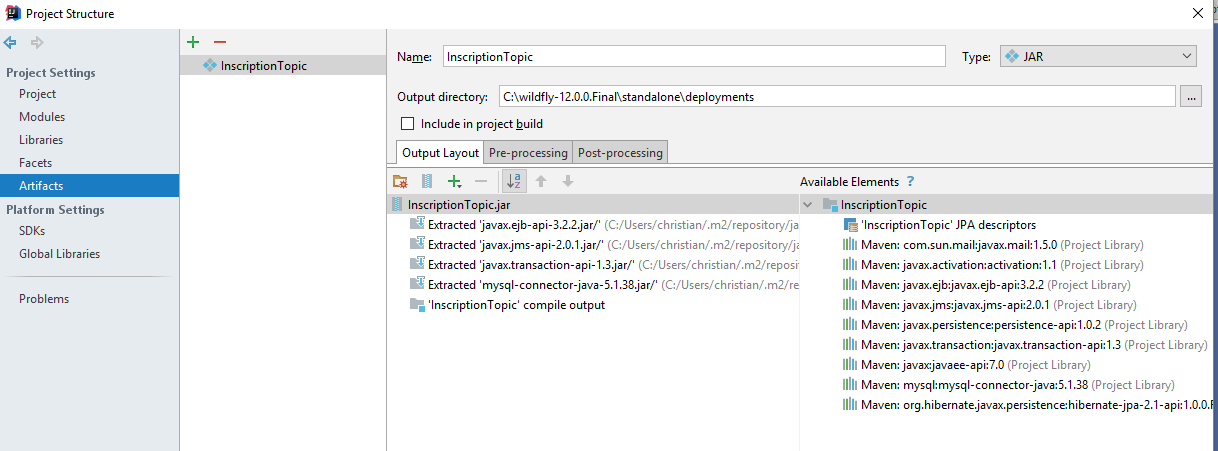
**try** {  
  
 **if** (!**entityManager**.contains(uneI))  
 {  
 *// On démarre une transaction* **entityManager**.getTransaction().begin();  
 **entityManager**.persist(uneI);  
 **entityManager**.flush();  
 *// on valide la transacition* **entityManager**.getTransaction().commit();  
 }  
 **entityManager**.close();  
  
 } **catch** (EntityNotFoundException h) {  
 **new** MonException(**"Erreur d'insertion"**, h.getMessage());  
 } **catch** (Exception e) {  
 **new** MonException(**"Erreur d'insertion"**, e.getMessage());  
 }  
 }  
}

#### jboss-ejb-client.properties

Il faut rajouter ce fichier pour se connecter à la file JMS

**endpoint.name**=**client-endpoint  
remote.connectionprovider.create.options.org.xnio.Options.SSL\_ENABLED**=**false  
remote.connections**=**default  
remote.connection.default.host**=**localhost  
remote.connection.default.port**=**8080  
remote.connection.default.connect.options.org.xnio.Options.SASL\_POLICY\_NOANONYMOUS**=**false  
remote.connection.default.username**=**jmsuser  
remote.connection.default.password**=**jmsepul98!**

La partie EJB de notre application est à présent terminée. Vous pouvez le déployer sous Wildfly



#### Déploiement de l’EJB DemandeInscriptions

Vous devez rajouter dans le fichier standlone.xml la configuration pour un message MDB (en gras). Personnellent, j’ai utilisé la version standloneFull.xml qui contient l’ensemble des possibilités du serveur.

<subsystem xmlns="urn:jboss:domain:ejb3:4.0">

<session-bean>

<stateless>

<bean-instance-pool-ref pool-name="slsb-strict-max-pool"/>

</stateless>

<stateful default-access-timeout="5000" cache-ref="simple" passivation-disabled-cache-ref="simple"/>

<singleton default-access-timeout="5000"/>

</session-bean>

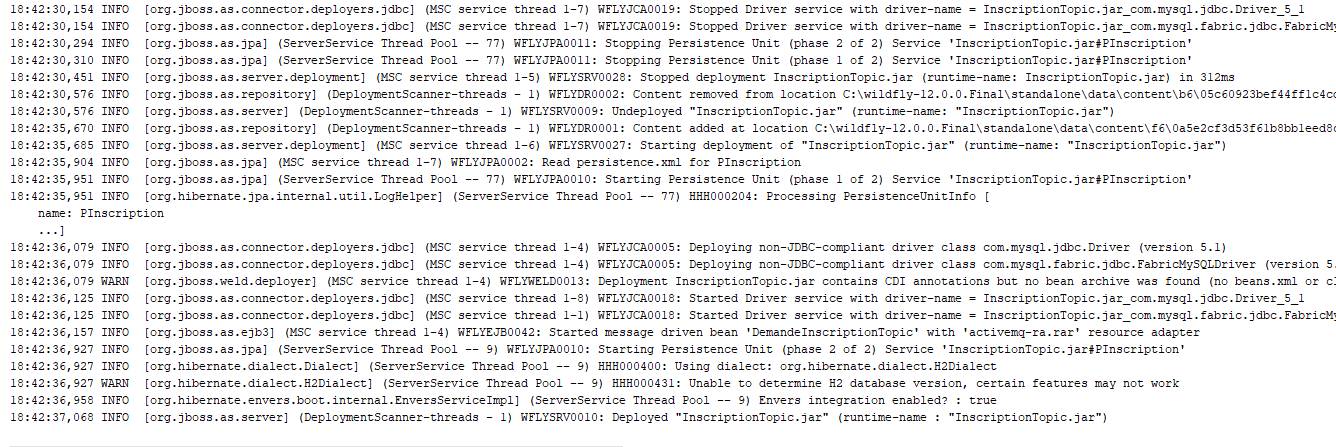
**<mdb>**

**<resource-adapter-ref resource-adapter-name="${ejb.resource-adapter-name:activemq-ra.rar}"/>**

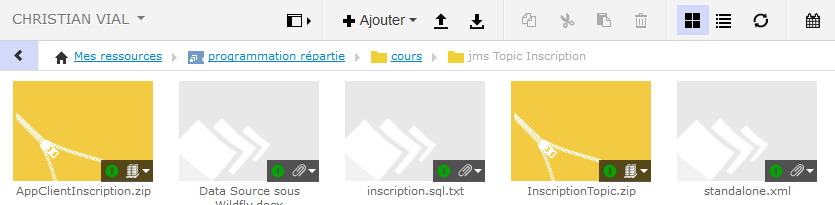
**<bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>**

**</mdb>**

Wildfly accepte votre EJB.



Vous pouvez récupérer l’EJB sous SPIRAL

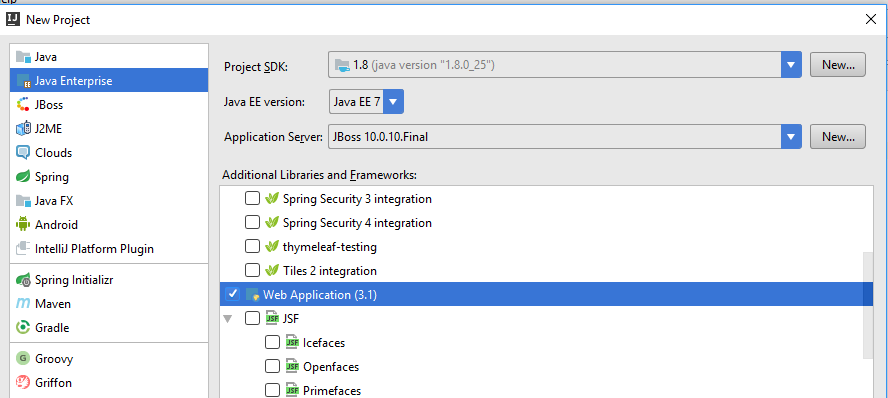


## Partie Cliente : production de nos messages

La partie cliente se compose

* de pages JSP
* d’un contrôleur
* d’une classe métier Inscription

Démarrez une nouvelle application de type Web Application



Ajoutez le composant maven et le framework bootstrap

Voici l’architecture de l’application avec le fichier pom.xml qui utilise peu de dépendances.

|  |  |
| --- | --- |
|  | *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**groupId**>groupId</**groupId**>  <**artifactId**>AppClientInscription</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  <**dependencies**>  *<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->* <**dependency**>  <**groupId**>javax.servlet</**groupId**>  <**artifactId**>jstl</**artifactId**>  <**version**>1.2</**version**>  </**dependency**>   </**dependencies**> </**project**> |

**Remarque** :

Les packages metier et meserreurs sont dupliqués entre l’EJB et l’application war.

Les pages index.jsp et AjoutInscription.jsp sont simples à écrire. La page AjoutInscription affiche un formulaire et appelle la servlet Controleur pour envoyer l’inscription à la file d’attente : InscriptionTopic

### Code la servlet Controleur

On déclare les packages à importer et les références sur notre file d’attente nommée InscriptionTopic

**package** controle;  
  
**import** java.io.IOException;  
**import** javax.jms.\*;  
**import** javax.servlet.ServletException;  
**import** javax.servlet.annotation.WebServlet;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** javax.servlet.ServletRequest;  
**import** javax.servlet.ServletResponse;  
**import** java.text.SimpleDateFormat;  
  
  
**import** java.sql.Date;  
  
**import** javax.annotation.Resource;  
**import** meserreurs.MonException;  
**import** metier.Inscription;  
  
**import** javax.naming.NamingException;  
  
*/\*\*  
 \* Servlet implementation class Traitement  
 \*/*@WebServlet(**"/Controleur"**)  
**public class** Controleur **extends** HttpServlet {  
  
 **private static final long *serialVersionUID*** = 10L;  
 **private static final** String ***ACTION\_TYPE*** = **"action"**;  
 **private static final** String ***AJOUTER\_INSCRIPTION*** = **"ajouteInscription"**;  
 **private static final** String ***ENVOI\_INSCRIPTION*** = **"envoiInscription"**;  
 **private static final** String ***RETOUR\_ACCUEIL*** = **"Retour"**;  
  
 */\*\*  
 \** ***@see*** *HttpServlet#HttpServlet()  
 \*/* @Resource(lookup = **"java:jboss/exported/topic/InscriptionJmsTopic"**)  
 **private** Topic **topic**;  
 *// On accède à l'EJB* @Resource(mappedName = **"java:/ConnectionFactory"**)  
 **private** TopicConnectionFactory **cf**;  
  
 *// Session établie avec le serveur* **private** TopicSession **session** = **null**;  
  
 *// Le client utilise un Producteur de messsage pour envoyer une demande de  
 // formation* **private** TopicPublisher **producer**;  
  
 */\*\**

Le corps du contrôleur est standard, il faut cependant penser à formater la date pour Mysql

*/\*\*  
 \* Constructeur par défaut de la classe  
 \*/***public** Controleur() {  
 **super**();  
}  
  
*/\*\*  
 \*   
 \*/***protected void** doGet(HttpServletRequest request, HttpServletResponse response)  
 **throws** ServletException, IOException {  
 **try** {  
 TraiteRequete(request, response);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
}  
  
*/\*\*  
 \** ***@see*** *HttpServlet#doPost(HttpServletRequest request, HttpServletResponse  
 \* response)  
 \*/***protected void** doPost(HttpServletRequest request, HttpServletResponse response)  
 **throws** ServletException, IOException {  
 *//* ***TODO Auto-generated method stub* try** {  
 TraiteRequete(request, response);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
}  
  
*/\*\*  
 \* Procédure principale de démarrage  
 \*/***public void** TraiteRequete(ServletRequest request, ServletResponse response) **throws** ServletException, IOException {  
 *// On récupère l'action* String actionName = request.getParameter(***ACTION\_TYPE***);  
  
 *// Si on veut afficher l'ensemble des demandes d'inscription* **if** (***AJOUTER\_INSCRIPTION***.equals(actionName)) {  
  
 request.getRequestDispatcher(**"AjouteInscription.jsp"**).forward(request, response);  
  
 } **else if** (***RETOUR\_ACCUEIL***.equals(actionName)) {  
 **this**.getServletContext().getRequestDispatcher(**"/index.jsp"**).include(request, response);  
 }  
  
 **else if** (***ENVOI\_INSCRIPTION***.equals(actionName))  
 {  
 SimpleDateFormat sdf = **new** SimpleDateFormat(**"dd/MM/yyyy"**);  
 response.setContentType(**"text/html;charset=UTF-8"**);  
 *// On récupère les informations sisies* String nom = request.getParameter(**"nom"**);  
 String prenom = request.getParameter(**"prenom"**);  
  
 **if** ((nom != **null**) && (prenom != **null**)) {  
 **try** {  
 *// On récupère la valeur des autres champs saisis par  
 // l'utilisateur  
 // on transfome la date  
 // au format Mysql java.sql.Date* String datenaissance = request.getParameter(**"datenaissance"**);  
 java.util.Date initDate = **new** SimpleDateFormat(**"dd/MM/yyyy"**).parse(datenaissance);  
 SimpleDateFormat formatter = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  
 String parsedDate = formatter.format(initDate);  
 initDate= formatter.parse(parsedDate);  
 Date uneDate = **new** Date(initDate.getTime());  
  
 String adresse = request.getParameter(**"adresse"**);  
 String cpostal = request.getParameter(**"cpostal"**);  
 String ville = request.getParameter(**"ville"**);  
  
 *// On crée une demande d'inscription avec ces valeurs* Inscription unedemande = **new** Inscription();  
 unedemande.setNomcandidat(nom);  
 unedemande.setPrenomcandidat(prenom);  
 unedemande.setDatenaissance(uneDate);  
 unedemande.setAdresse(adresse);  
 unedemande.setCpostal(cpostal);  
 unedemande.setVille(ville);  
  
 *// On envoie cette demande d'inscription dans le topic* **boolean** ok = envoi(unedemande);  
 **if** (ok)  
 *// On retourne àla page d'accueil* **this**.getServletContext().getRequestDispatcher(**"/index.jsp"**).include(request, response);  
 **else** {  
 **this**.getServletContext().getRequestDispatcher(**"/Erreur.jsp"**).include(request, response);  
 }  
 }**catch** (MonException m) {  
 *// On passe l'erreur à la page JSP* request.setAttribute(**"MesErreurs"**, m.getMessage());  
 request.getRequestDispatcher(**"PostMessage.jsp"**).forward(request, response);  
 }**catch** (Exception e) {  
 *// On passe l'erreur à la page JSP* System.***out***.println(**"Erreur client :"** + e.getMessage());  
 request.setAttribute(**"MesErreurs"**, e.getMessage());  
 request.getRequestDispatcher(**"PostMessage.jsp"**).forward(request, response);  
 }  
 }  
 }  
}

Lorsque l’objet Inscription est instancié, il faut l’envoyer à la file d’attente avec le code suivant :

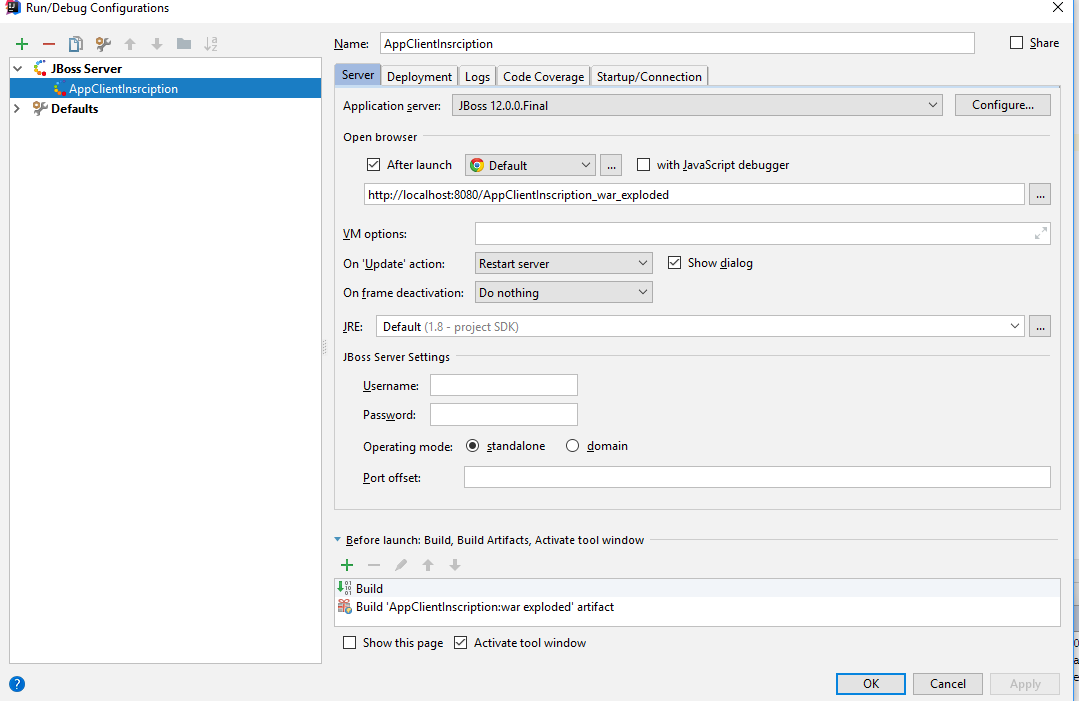
* création d’une factory
* ouverture d’une session
* connexion
* publication du message

*/\*\*  
 \* Permet de publier une demande d'inscription dans le topic  
 \*   
 \** ***@param uneDemande*** *\* La demande d'inscription � publier  
 \** ***@return*** *\** ***@throws*** *Exception  
 \*/* **boolean** envoi(Inscription uneDemande) **throws** Exception {  
  
 **boolean** ok = **true**;  
 TopicConnection connection = **null**;  
  
 **try** {  
  
 *// On crée la connexion JMS , session, producteur et message;  
 /\*  
 \* connection = connectionFactory.createConnection(  
 \* System.getProperty("username", DEFAULT\_USERNAME),  
 \* System.getProperty("password", DEFAULT\_PASSWORD));  
 \*/  
  
 // Création Connection et Session JMS* connection = **cf**.createTopicConnection();  
 **session** = connection.createTopicSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  
  
 *// On crée le producteur utilisé pour envoyer un message* **producer** = **session**.createPublisher(**topic**);  
 *// On lance la connection* connection.start();  
 ObjectMessage message = **session**.createObjectMessage();  
 message.setObject(uneDemande);  
 *// On publie le message* **producer**.publish(message);  
 **producer**.close();  
 **session**.close();  
 connection.close();  
 } **catch** (JMSException j) {  
 **new** MonException(j.getMessage());  
 ok=**false**;  
 }**catch** (Exception e) {  
 **new** MonException(e.getMessage());  
 ok=**false**;  
 }  
 **return** ok;  
 }  
}

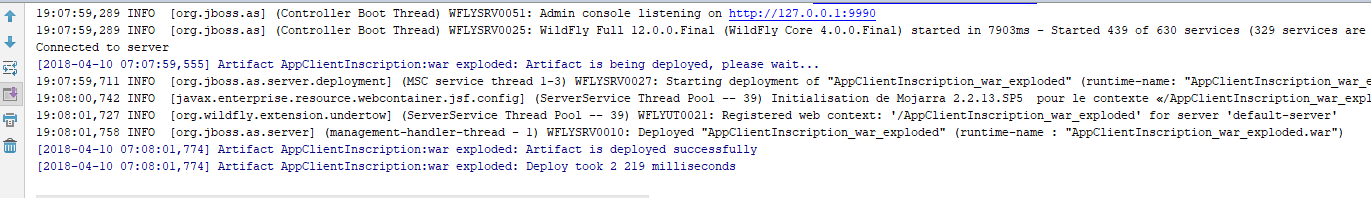
Vous pouvez récupérer la partie Cliente sous Spiral.

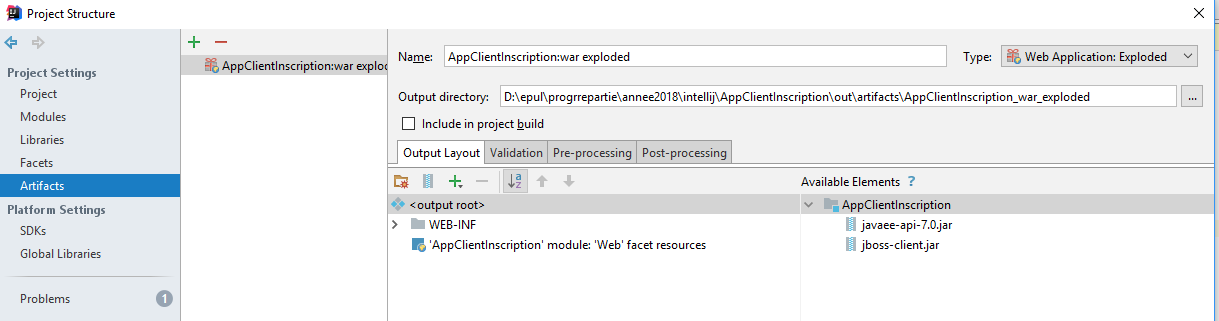
### Déploiement de l’application

Il faut choisir le serveur WildFly 12



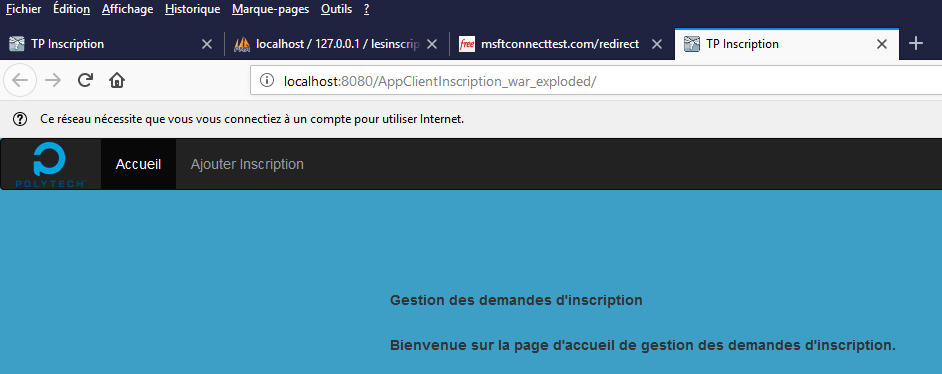
On crée l’artifact et on lance l’application

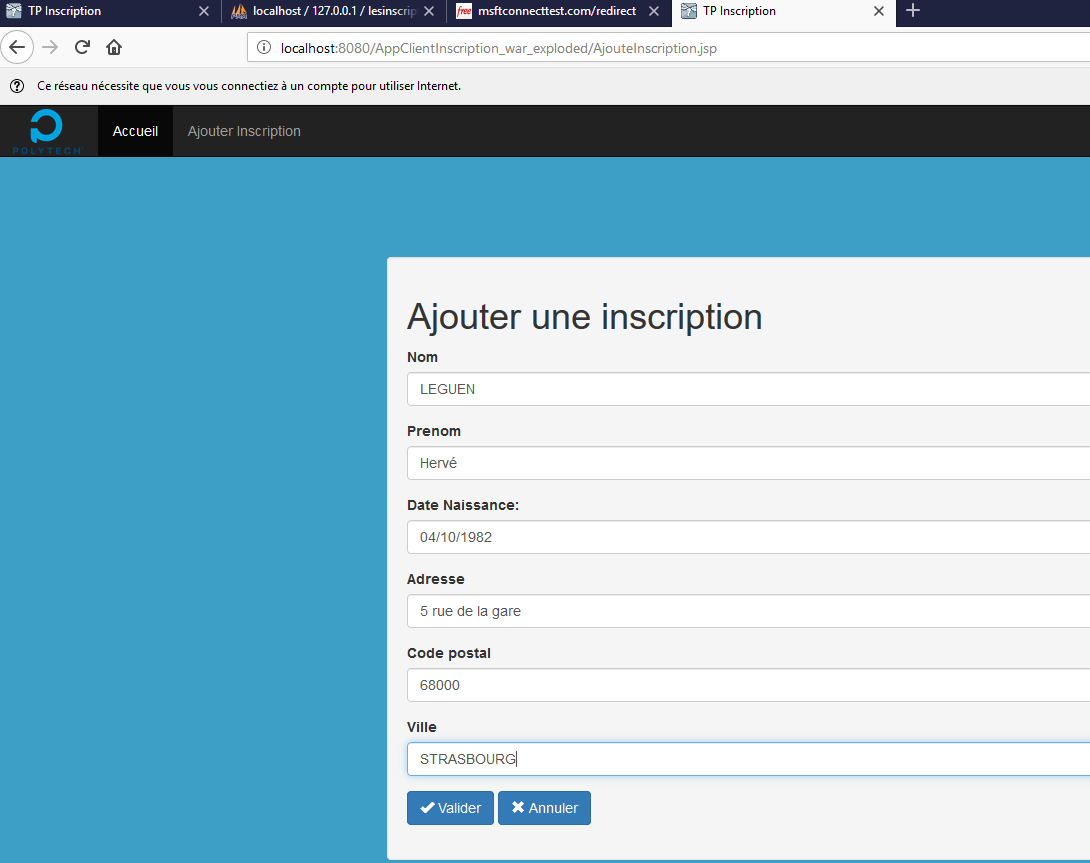




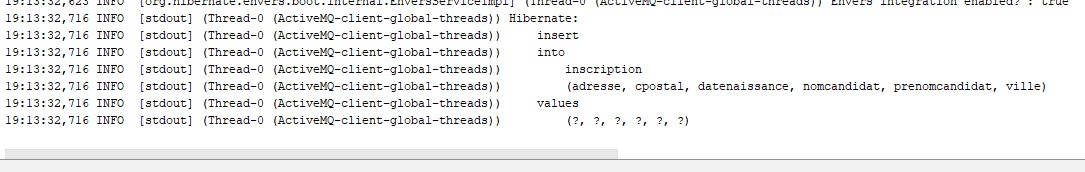
### Fonctionnement de l’application

Cette application est écrite en utilisant le framework Bootstrap





Sur la validation, on peut voir la requête SQL transmise



Contrôle au niveau de la base de données



# Gestion des Erreurs

On doit intercepter les erreurs dans :

* la connexion à la base et sa gestion (lecture, modification, ajout..)

Les erreurs sont interceptées et stockées dans la classe Monexception. Cette classe est instanciée si une erreur survient.

Il n’est pas possible d’afficher les erreurs générées par l’EJB JMS avec les pages de l’application Cliente. Il s’agit en fait de deux processus différents qui ne peuvent pas communiquer. La solution proposée consiste à écrire les erreurs dans un fichier **erreurs.log**.

Pour les erreurs qui surviennent dans la page JSP, il est possible de faire afficher les erreurs avec une fonction javascript.

## Affichage des erreurs générées dans les pages de l’application cliente

Il faut définir un champ caché qui recevra l’éventuelle erreur en provenance de la servlet Controleur

**public** **void** service(ServletRequest request, ServletResponse response)

**throws** ServletException, IOException {

………

**catch** (MonException e) {

// On passe l'erreur à la page JSP

System.*out*.println("Erreur client :"+ e.getMessage()) ;

request.setAttribute("MesErreurs", e.getMessage());

request.getRequestDispatcher("Resultat.jsp").forward(request, response);

}

**catch** (Exception e) {

System.*out*.println("Erreur client Exception :"+ e.getMessage()) ;

request.setAttribute("MesErreurs", e.getMessage());

request.getRequestDispatcher("Resultat.jsp").forward(request, response);

}Cette erreur arrive dans le champ caché de la page

<input type =*"hidden"* name=*"uneErreur"* value=*"*${MesErreurs}*"* id =*"id\_erreur"* >

Il faut à présent l’afficher par une fonction javascript :

**function** Chargement()

{

**var** obj = document.getElementById("id\_erreur");

**if** (obj.value!='')

alert('Erreur signalée : "'+obj.value+"'");

}

</script>

Cette fonction est activée au lancement de la page

<BODY onLoad=*"Chargement();"* >

### Gestion des erreurs générées dans l’EJB JMS

On les enregistre dans un fichier erreurs.log du répertoire jboss../bin

**private** **void** EcritureErreur ( String message)

{

BufferedWriter wr ;

String rep=System.*getProperty*("user.dir");

String nomf ="erreurs.log";

java.util.Date madate = **new** java.util.Date();

SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/yy hh:mm");

**try**

{

wr = **new** BufferedWriter(**new** FileWriter(nomf));

wr.write( sdf.format(madate));

wr.newLine();

wr.write(message);

wr.close();

}

**catch** (FileNotFoundException ef) {

;

} **catch** (IOException eio) {

;

}

}

**public** **void** onMessage(Message message) {

…..

**catch** (MonException e)

{

EcritureErreur (e.getMessage());

}

}

**catch (JMSException jmse) {**

**EcritureErreur (jmse.getMessage());**

**context.setRollbackOnly();**

**}**

}

### Simulation d’une erreur  et enregistrement dans le fichier

On modifie le mode passe pour se connecter à la base de données.

On obtient :

