ASI-2
J2E
Step By Step

2015

Ce Tp a pour objectif de concevoir une application cross technologie permettant d'appréhender la fonction et l'usage de chaque technologie.
Cette deuxième partie vous permettra de mettre en œuvre une application à base de web service, d'EJB et de JMS permettant de délivrer une authentification

Part II J2E-M2M

Table des matières

1. Introduction

Ce Tp déroule en 3 étapes comme suit :

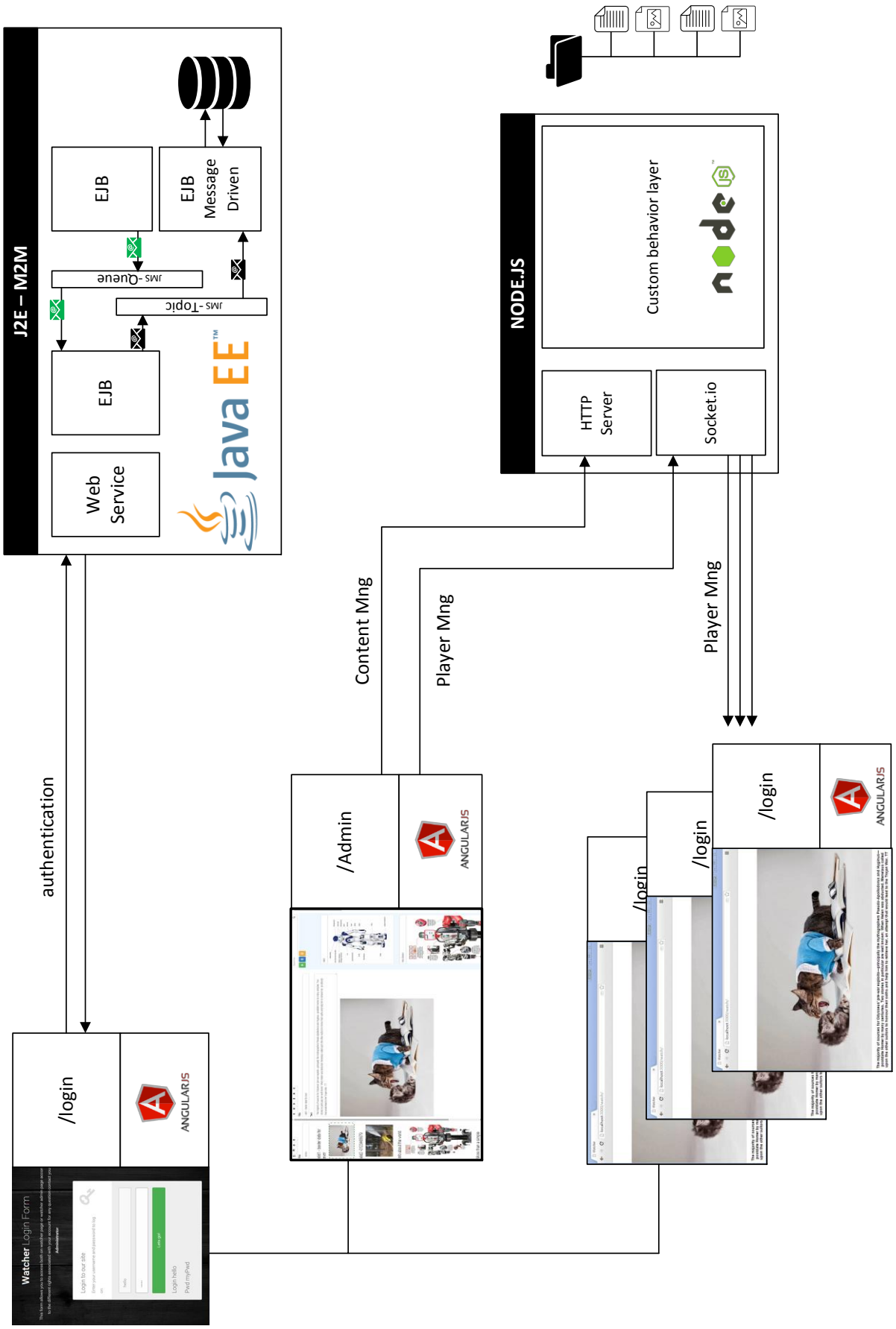
- **Step 1** : Création d'un point d'entrée, d'un web service pour la demande d'authentification
- **Step 2** :Création d'un EJB permettant d'envoyer cette demande sur un bus de communication
- **Step 3** :Création d'un EJB Message driven permettant de recevoir les demandes d'authentification.
- **Step 4** : Création d'un EJB permettant de renvoyer un réponse sur un 2ème bus de communication
- **Step 5** Création d'un EJB entité permettant de récupérer les informations de la base de données

CHAQUE ETAPE et SOUS ETATE DEVRA ETRE VALIDEE PAR UN ENSEIGNANT AVANT DE PASSER A LA SUIVANTE.

L'évaluation prendra en compte votre compréhension de l'étape, la mise en œuvre et les bonnes pratiques de programmation.

2. Présentation générale

2.1.Présentation du projet dans sa globalité

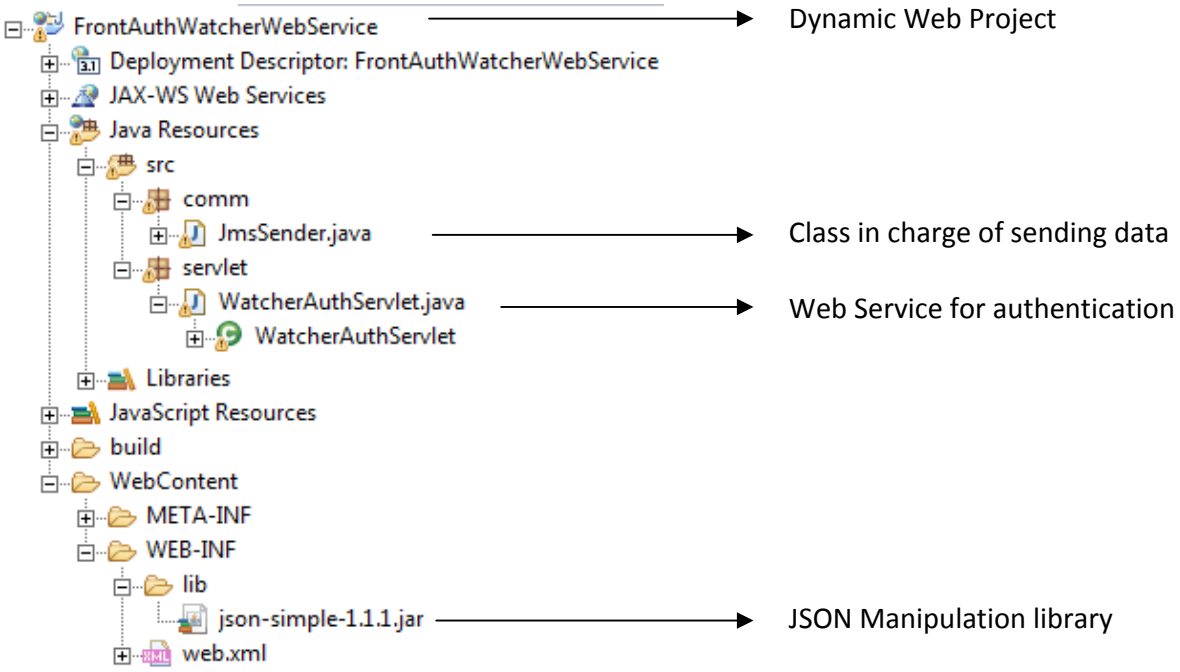


3. STEP 1 : Web service d'authentification

Objectif : Réalisation d'un web service, manipulation de données au format JSON

3.1. Création d'un projet web dynamique

3.1.1. L'objectif est de créer une structure de programmation comme suit :



3.2. Créer WatcherAuthServlet.java

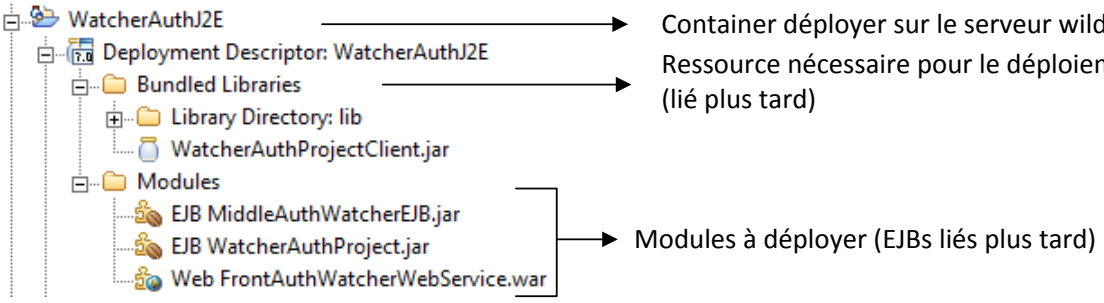
3.2.1. Créer une servlet WatcherAuthServlet.java possédant les caractéristiques suivantes :

- Url : /WatcherAuth
- doPost fonction qui traitera l'information d'auth (JSON format {login :*jdoe*,pwd :*jdoepwd*})
- Les informations reçues seront transformées en un objet UserModel (lastName,surName,login,pwd,role)
- Affichée l'objet crée (System.ou.println)
- Renvoyer une réponse à la requête au format JSON ({login : user.getLogin(),validAuth :true,role : 'ADMIN'})

4. STEP 2 : EJB Sender EJB Receiver

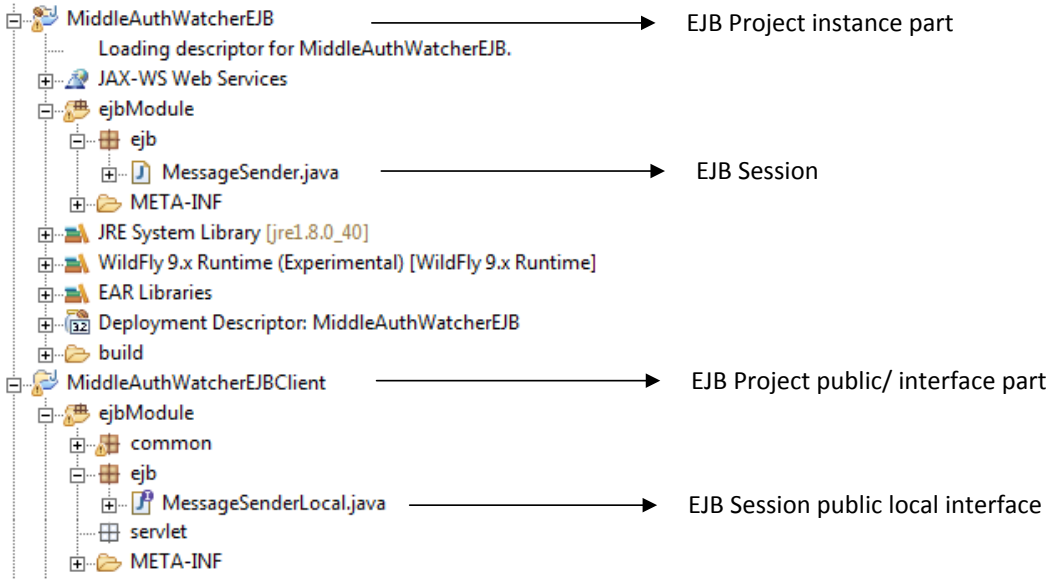
Objectif : Création et Usage d'EJB, utilisation de bus de données JMS

4.1.Création d'un composant Entreprise Application Project comme suit :



4.2.Création d'un EJB Session Sender

4.2.1.Créer un projet EJB comme suit :



4.2.2.A quoi sert la partie publique du projet ? En quoi est –elle indispensable ?

4.2.3.Créer un EJB session MessageSender de type StateLess

4.2.4.Pourquoi est-il préférable de coché la case générer interface ?

4.2.5. Que représente Business Interface Remote ?

4.2.6.Que représente Business Interface Local ?

4.2.7.A l'aide des références ci-dessous compléter l'EJB session MessageSender compléter la classe afin qu'elle puisse envoyer des messages dans le bus de données `"java:/jms/watcherAuthJMS")`

http://www.oracle.com/technetwork/articles/java/jms20-1947669.html
https://docs.oracle.com/javaee/7/api/javax/jms/JMSContext.html

```java
package ejb;

import javax.annotation.Resource;
import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.inject.Inject;
import javax.jms.JMSContext;
import javax.jms.JMSException;
import javax.jms.ObjectMessage;
import javax.jms.Topic;
import common.UserModel;

@Stateless
@LocalBean
public class MessageSender implements MessageSenderLocal {

	@Inject
	JMSContext context;

	@Resource(mappedName = "java:/jms/watcherAuthJMS")
	Topic topic;

	public void sendMessage(String message) {
		// TODO
	}

	public void sendMessage(UserModel user) {
		// TODO

	}
}
```

4.2.8.Que Signifie @Stateless ?

4.2.9.Que Signifie @LocalBean ?

4.2.10. Que permet de faire l'annotation @Inject ?

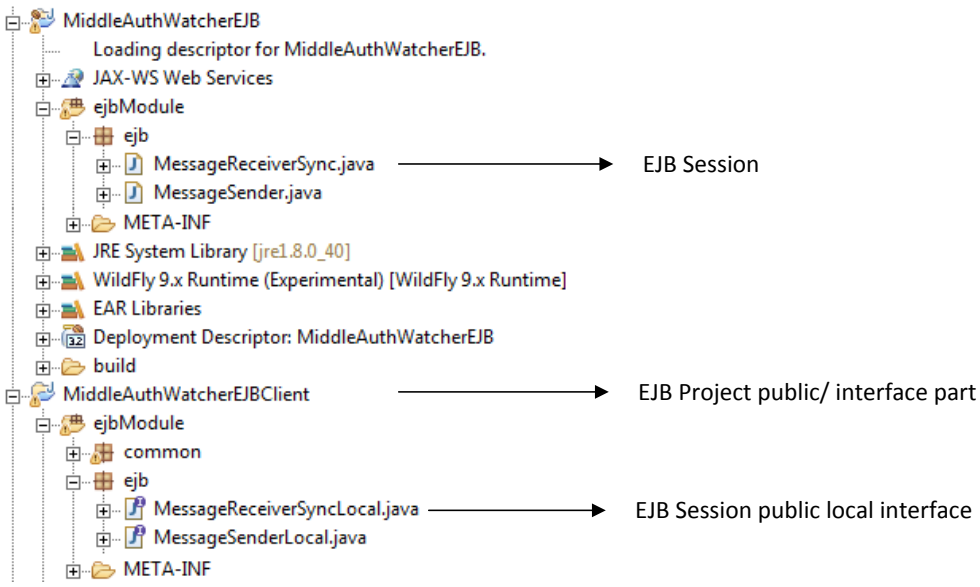
4.2.11. Que permet de faire l'annotation @Ressource?

4.2.12. Quelle est la différence entre un Topic et une Queue ?

FAIRE VALIDER L'ETAT D'AVANCEMENT

4.3.Création d'un EJB Session Receiver

4.3.1.Modifier le projet EJB préalablement créé afin d'obtenir la structure suivante



4.3.2.A l'aide des références ci-dessous compléter l'EJB session MessageReceiverSync compléter la classe afin qu'elle puisse recevoir des messages dans le bus de données
https://docs.oracle.com/javaee/7/tutorial/jms-concepts.htm#BNCDQ
https://docs.oracle.com/javaee/7/tutorial/jms-examples002.htm#BNCFA

```
package ejb;

import javax.annotation.Resource;
import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.inject.Inject;
import javax.jms.JMSConsumer;
import javax.jms.JMSContext;
import javax.jms.Queue;

import common.UserModel;

@Stateless
@LocalBean
public class MessageReceiverSync implements MessageReceiverSyncLocal {

	// TODO get jms context

	// TODO associate queue from "java:/jms/queue/watcherqueue"

	 public UserModel receiveMessage() {

	// TODO create a consumer

	//TODO Wait 1s incoming message (UserModel)

		return message;
	 }
}
```

4.4.Associer les EJB Session Sender and Receiver au Web Service

4.4.1.Modifier the web service WatcherAuthServlet afin d'utiliser les EJB précédemment créé

```
	...

@WebServlet("/WatcherAuth")
public class WatcherAuthServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	// private JmsSender sender;

	@EJB
	MessageSenderLocal sender;

	@EJB
	MessageReceiverSyncLocal receiver;

	...
```

4.4.2.Pourquoi est-ce que les objets appelés sont « Local » ?

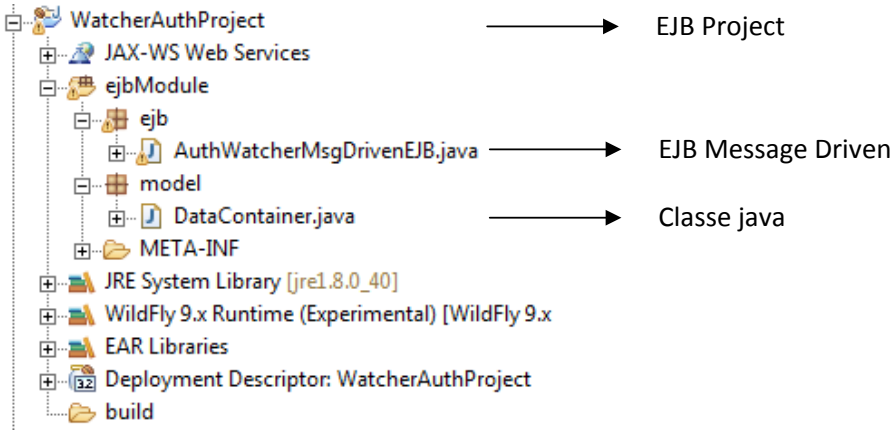
4.4.3.Quelle(s) modification(s) dans le projet doit-on effectuer afin de pouvoir utiliser MessageSenderLocal et MessageReceiverSyncLocal ?

FAIRE VALIDER L'ETAT D'AVANCEMENT

5. STEP 3 : EJB Message driven

Objectif : Création et Usage d'EJB Message Driven, communication avec une base de données

5.1.Créer un projet EJB comme suit :



5.1.1.Pourquoi ce projet n'a-t'il pas besoin de partie publique ?

5.1.2.Qu'est ce qu'un EJB message driven ?

5.2.Compléter l'EJB Message Driven comme suit :

```
package ejb;

import java.util.Date;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;
import javax.jms.TextMessage;
import common.Role;
import common.UserModel;
import model.DataContainer;

@MessageDriven(
      activationConfig = {
            @ActivationConfigProperty(
                  propertyName = "destinationType",
                  propertyValue = "javax.jms.Topic"),
            @ActivationConfigProperty(
                  propertyName = "destination",
                  propertyValue = "java:/jms/watcherAuthJMS")
            })
public class AuthWatcherMsgDrivenEJB implements MessageListener {
      private DataContainer dataContainer;

            public AuthWatcherMsgDrivenEJB() {
      dataContainer=new DataContainer();
    }

    public void onMessage(Message message) {

      }
}
```

5.3.Envoi de Réponse à l'aide d'un EJB Session

Afin de permettre à notre EJB message driven de répondre à notre Web Service une nouvel EJB Session sera créer qui enverra une réponse dans un bus de message de type Queue

5.3.1.Pour cela dans votre projet d'ejb MiddleAuthWatcherEJB ajouter un nouvelle EJB session MessageSenderQueue comme suit :

```
package ejb;

import javax.annotation.Resource;
import javax.ejb.Stateless;
import javax.inject.Inject;
import javax.jms.JMSContext;
import javax.jms.JMSException;
import javax.jms.ObjectMessage;
import javax.jms.Queue;

import common.UserModel;

/**
 * Session Bean implementation class MessageSenderQueue
 */
@Stateless
public class MessageSenderQueue implements MessageSenderQueueLocal {

	@Inject
	JMSContext context;

	@Resource(mappedName = "java:/jms/queue/watcherqueue")
	Queue queue;

	public void sendMessage(String message) {
		context.createProducer().send(queue, message);
	}

	public void sendMessage(UserModel user) {
		try {
			ObjectMessage message = context.createObjectMessage();
			message.setObject(user);
			context.createProducer().send(queue, user);
		} catch (JMSException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}
```

5.3.2.Une fois votre EJB créé, vous pouvez l'utiliser dans l'EJB Message Driven comme suit :

```
...
public class AuthWatcherMsgDrivenEJB implements MessageListener {
	private DataContainer dataContainer;

	@EJB MessageSenderQueueLocal sender;


	public AuthWatcherMsgDrivenEJB() {
	dataContainer=new DataContainer();
    }
    public void onMessage(Message message) {
         try {
              if (message instanceof TextMessage) {
                  System.out.println("Topic: I received a TextMessage at "
```

```
                        + new Date());
                TextMessage msg = (TextMessage) message;
                System.out.println("Message is : " + msg.getText());

            } else if (message instanceof ObjectMessage) {
                System.out.println("Topic: I received an ObjectMessage at "
                        + new Date());
                ObjectMessage msg = (ObjectMessage) message;

                if( msg.getObject() instanceof UserModel){
                    UserModel user=(UserModel)msg.getObject();

                     System.out.println("User Details: ");
                     System.out.println("login:"+user.getLogin());
                     System.out.println("pwd:"+user.getPwd());

                    Role currentTestRole=dataContainer.checkUser(user);
                     if( Role.NONE==currentTestRole){
                           sender.sendMessage(user);
                     }else{
                           user.setRole(currentTestRole);
                           sender.sendMessage(user);
                     }
                }
            } else {
                System.out.println("Not valid message for this Queue MDB");
            }
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

5.3.3. Compléter la classe DataContainer afin qu'elle stocke des utilisateurs (UserModel) et vérifie la validité de ces derniers (checkUser).

FAIRE VALIDER L'ETAT D'AVANCEMENT

5.4. Stockage des Users dans une base de données

5.4.1. Utiliser les EJB entity afin de pouvoir récupérer les UserModel contenu dans une base de données.

FAIRE VALIDER L'ETAT D'AVANCEMENT