Crédit Mutuel ARKEA

"Un Groupe de bancassurance en mouvement"

Apache Thrift Bootcamp

Serveur d'application
Client HTML/JS

Pré-requis

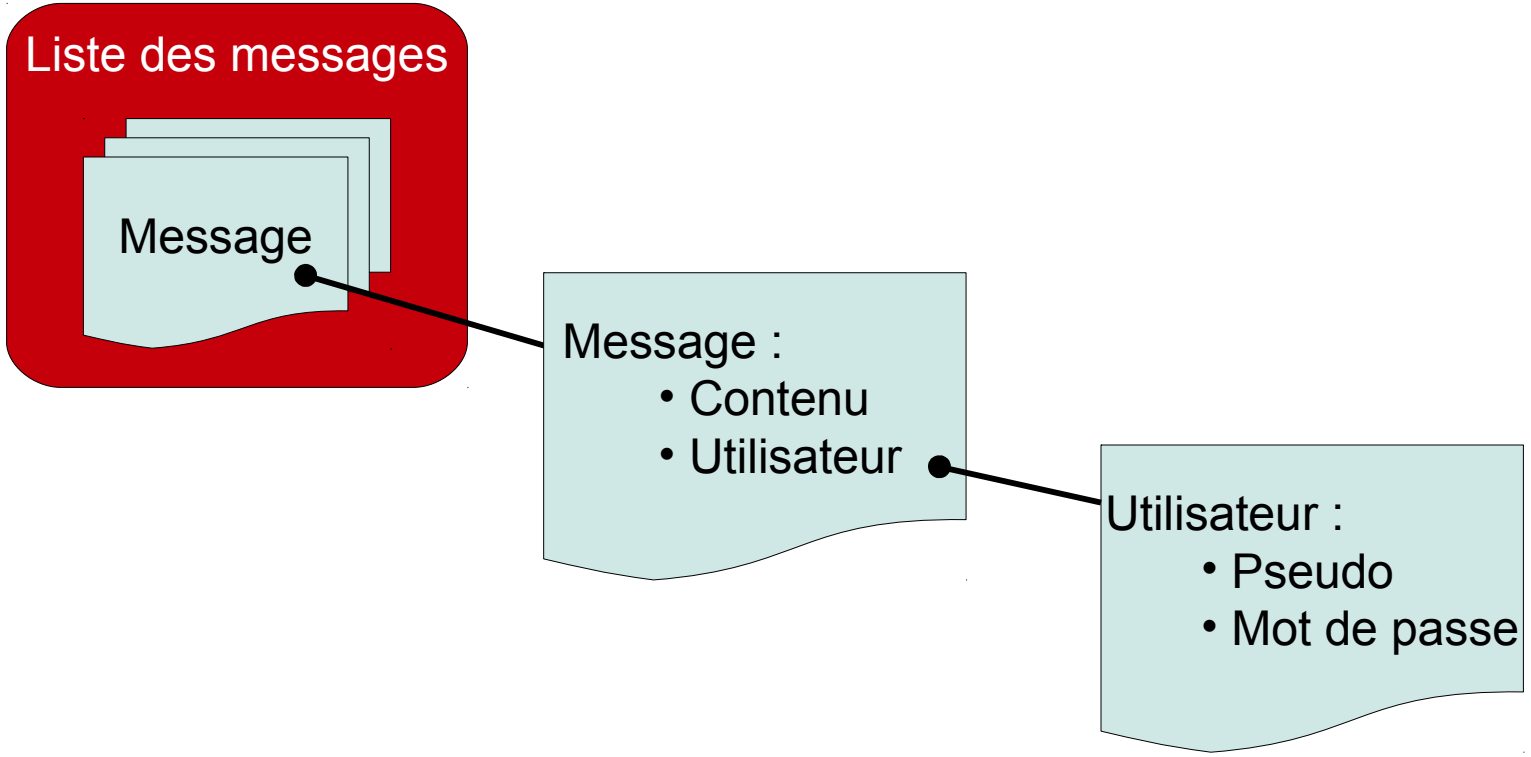
La liste ci-dessous de logiciels doit être installée et opérationnelle sur la machine des participants :

- Compilateur thrift
  http://thrift.apache.org/
- Libthrift java
- Serveur d'application Tomcat
  http://tomcat.apache.org/
- Eclipse IDE for Java EE Developers
  http://www.eclipse.org/

Objectif : ChatRoom

- Création d'une application ChatRoom simple :
  - Enregistrement d'utilisateur
  - Visualiser les 20 derniers messages
  - Envoyer un message
- Implémentations :
  - Partie Serveur en Java
  - Implémentations clientes
    - dans des différents langages

Modèle



Fonctions métiers

- Lister les messages
- Envoyer un message
- Enregistrer un utilisateur

Modéle Thrift 1 - utilisateur.thrift

```
namespace java com.arkea.thrift.data.utilisateur

/**
* Structure utilisateur
**/
struct Utilisateur {

        /** pseudo Utilisateur */
        1:string pseudo,

        /** mot de passe */
        2:string motdepasse,
}
```

Modèle Thrift 2 - message.thrift

```
namespace java com.arkea.thrift.data.message

include "utilisateur.thrift"

/**
* Structure Message
**/
struct Message {

	/** contenu Message */
	1:string contenu,

	/** utilisateur ayant ecrit le Message */
	2:utilisateur.Utilisateur utilisateur,
}
```

Modèle Thrift 3 - chatRoomService.thrift

```
namespace java com.arkea.thrift.service.chatroom

include "message.thrift"
include "utilisateur.thrift"

/** Service ChatRoomService */
service ChatRoomService {

        /** Recupérer la liste des Messages */
        list<message.Message> getListeMessage()

        /** Envoyer un message */
        void envoyerMessage(1:message.Message message)

        /** Enregistrer un utilisateur */
        void enregistrerUtilisateur
                (1: utilisateur.Utilisateur utilisateur)

}
```

Génération Java

1. Dans un répertoire, créer les fichiers `user.thrift`, `message.thrift` et `chatRoomService.thrift`

2. Compiler le fichier `chatRoomService.thrift` via la commande :

```
thrift -v -r --gen java:beans,hashcode chatRoomService.thrift
```

- Les fichiers `user.thrift` et `message.thrift` sont compilés par récursivité
  - ils sont inclus dans le fichier service.
- Les classes générées sont dans le répertoire `gen-javabean`
  - avec l'arborescence spécifié dans la variable `namespace`.

Serveur d'application - Configuration `Server Runtime`

1. Dans Eclipse, aller dans le menu `Window > Preferences`
2. Ouvrir l'onglet `Server > Runtime Environments`
3. Cliquer sur `Add`
4. Sélectionner votre version de serveur d'application
   - Dans notre cas Apache Tomcat v7.0
5. Cliquer sur `Next` et choisir le répertoire d'installation via le bouton `Browse` puis cliquer sur `Finish`
6. Fermer la fenêtre des préférences en cliquant sur `OK`

Serveur d'application - Création projet

1. Créer un projet `Dynamic Web Project` sous Eclipse.
2. Nommer le `ChatRoomServeur`
3. Cliquer deux fois sur `Next`
4. Cocher la case `Generate web.xml deployment descriptor`
5. Cliquer sur `Finish`

Serveur d'application - Bibliothèques

1. Hors Eclipse, copier les bibliothèques dans le répertoire `ChatRoomServeur/WebContent/WEB-INF/lib`
   - `commons-lang.jar, httpclient.jar, httpcore.jar, slf4j-api.jar, slf4j-log4j12.jar, libthrift.jar, log4j.jar`
2. Dans Eclipse, click droit sur le projet `ChatRoomServeur` puis Refresh
3. Vérifier que les `.jar` sont bien présentes dans le répertoire `ChatRoomServeur/WebContent/WEB-INF/lib`

Serveur d'application – Bibliothèque `Server Runtime`

1. Dans Eclipse, click droit sur le projet `ChatRoomServeur` puis `Properties`
2. Cliquer sur `Java Build Path` et selectionner l'onglet `Libraries`
3. Cliquer sur `Add Librarie...` et sélectionner `Server Runtime`
4. Cliquer sur `Next`, sélectionner le serveur Tomcat précédemment configuré puis cliquer sur `Finish`
5. Fermer la fenêtre des préférences en cliquant sur `OK`

Serveur d'application - Création du `package`

1. Etendre `ChatRoomServeur > Java Resources`
2. Click droit sur `src` puis `New > Package`
3. Saisir `com.arkea.serveur.servlet`
4. Cliquer sur `Finish`
5. Refaire la même opération pour le package `com.arkea.serveur.processor`

Serveur d'application - Création des classes

1. Click droit sur le package `com.arkea.serveur.servlet`, puis `New > Class`
2. Saisir `ChatRoomServlet` puis cliquer sur Finish
3. Réitérer l'opération pour le package `com.arkea.serveur.processor` avec les classes `ChatRoomProcessor`

Serveur d'application - Ajout des classes thrift générées

1. Hors Eclipse, copier les classes thrifts générées du répertoire `gen-javabean` au répertoire `src` du projet `ChatRoomServeur`
2. Dans Eclipse, click droit sur le projet `ChatRoomServeur` puis `Refresh`
3. Puis vérifier que les packages ainsi que les classes thrifts sont désormais visible dans le répertoire `src`

Serveur d'application - `ChatRoomProcessor`

1. Ouvrir la classe `ChatRoomProcessor` dans l'editeur
2. Ajouter l'import : `import com.arkea.thrift.service.chatroom.*;`
3. Implémenter l'interface `ChatRoomService.Iface`,
   - Ajouter les méthodes obligatoires
4. Dans la méthode `enregistrerUtilisateur()`
   - Remplacer le `// TODO Auto-generated method stub` par `System.out.println(utilisateur.getPseudo());`
5. Dans la méthode `envoyerMessage()`
   - Remplacer le `// TODO Auto-generated method stub` par `System.out.println(message.getContenu());`

Serveur d'application - `ChatRoomServlet` (1/2)

Cette servlet permet de faire le mapping entre le flux http entrant dans le serveur d'application et le service thrift que nous avons créé. Le format d'échange des données est le format JSON.

1. Ouvrir la classe `ChatRoomServlet` dans l'editeur

2. Ajouter les imports thrifts suivants :

```
import org.apache.thrift.protocol.TJSONProtocol;
import org.apache.thrift.server.TServlet;
import com.arkea.serveur.processor.ChatRoomProcessor;
import com.arkea.thrift.service.chatroom.ChatRoomService;
```

3. Hériter de la classe `TServlet`

Serveur d'application - `ChatRoomServlet` (2/2)

4. Ajouter le constructeur :

```
public ChatRoomServlet() {
  super(
    new ChatRoomService.Processor<ChatRoomService.Iface>(
      new ChatRoomProcessor()),
    new TJSONProtocol.Factory());
}
```

5. Ajouter un `serialVersionUID`

6. Sauvegarder

Serveur d'application - Configuration servlet `web.xml`

1. Etendre `WebContent/WEB-INF`
2. Ouvrir le fichier `web.xml`
3. Après la ligne `</welcome-file-list>`, ajouter les lignes

```
<servlet>
      <servlet-name>ChatRoomServlet</servlet-name>
      <servlet-
     class>com.arkea.serveur.servlet.ChatRoomServlet</servlet-
     class>
</servlet>

<servlet-mapping>
      <servlet-name>ChatRoomServlet</servlet-name>
      <url-pattern>/chatroom/*</url-pattern>
</servlet-mapping>
```

4. Sauvegarder le tout

Serveur d'application - Déploiement

1. Sous Eclipse, ajouter la vue `Server` via le menu `Windows > Show View > Servers`

2. Dans la nouvelle vue, faire un click droit et `New > Server`

5. Cliquer sur `Next` dans la nouvelle boite de dialogue

7. Cliquer sur `ChatRoomServer > Add > Finish`

9. Click droit sur le nouveau serveur puis `Debug`

Le serveur se lance, les traces sont disponibles dans la vue console

Application Cliente JS - Présentation

Création d'une application cliente utilisant le JavaScript.

- Cette application sera hébergée sur un serveur d'application utilisant le même nom de domaine que le service `ChatRoomServeur`
  - dans notre cas, `localhost`
  - pour éviter les alertes de *cross-scripting* dans les navigateurs.

Application Cliente JS - Création projet

1. Créer un projet `Dynamic Web Project` sous Eclipse.
2. Nommer le `ChatRoomJs`
3. Cliquer sur `Finish`
4. Dans le répertoire `WebContent`, créer un répertoire `js`

Application Cliente JS - Génération JS

Dans le répertoire qui contient les fichiers thrift, exécuter la commande de génération JS suivante :

```
thrift -v -r --gen js:jquery chatRoomService.thrift
```

La génération des JS sera compatible avec le modèle JQuery parce qu'à l'heure actuelle il s'agit de la librairie la plus commune.

Application Cliente JS - Ajout des fichiers au projet

1. Copier l'ensemble des fichiers JS du répetoire `gen-js` dans les répertoire `WebContent/js` du projet `ChatRoomJs`
2. Ajouter également au répertoire la lib `thrift.js` présent dans le projet thrift dans le répertoire /lib/js
3. Ajouter également la dernière version de JQuery (http://jquery.com/)
4. Dans Eclipse, click droit sur le projet `ChatRoomJs` puis Refresh
5. Puis vérifier que les fichiers JS sont désormais visible dans le répertoire `WebContent/js`

Application Cliente JS - chatroom.html

1. Sous éclipse, click droit sur `WebContent` du projet `ChatRoomJs` créer puis `New > File`

2. Saisir `chatroom.html`, puis `Finish`

3. Y ajouter le squelette ci-dessous puis sauvegarder :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
      "http://www.w3.org/TR/REC-html40/strict.dtd">
<HTML>
<HEAD>
<TITLE>ChatRoom Service</TITLE>
</HEAD>
<BODY>
         <a href="enregistrer.html">Enregistrer une utilisateur</a><br>
         <a href="envoyer.html">Envoyer un message</a><br>
         <a href="message.html">Lister les messages</a><br>
</BODY>
</HTML>
```

Application Cliente JS - Fichiers services (1/2)

1. Créer les fichiers `enregistrer.html`, `message.html` et `envoyer.html` comme `chatroom.html`

2. Ajouter le squelette suivant :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
      "http://www.w3.org/TR/REC-html40/strict.dtd">
<HTML>
<HEAD>
<TITLE>Fonction</TITLE>
</HEAD>
<BODY>
         <div></div>
         <br>
         <a href="chatroom.html">retour</a>
</BODY>
</HTML>
```

Application Cliente JS - Fichiers services (2/2)

3. Ajouter les includes javascript suivants entre les balises `head` :

```
<script src="./js/thrift.js" type="text/javascript"></script>
<script src="./js/jquery-1.7.2.min.js" type="text/javascript"></script>
<script src="./js/utilisateur_types.js" type="text/javascript"></script>
<script src="./js/message_types.js" type="text/javascript"></script>
<script src="./js/ChatRoomService.js" type="text/javascript"></script>
<script src="./js/chatRoomService_types.js" type="text/javascript"></script>
```

4. Sauvegarder

Application Cliente JS – `enregistrer.html` (1/2)

1. Ajouter le code HTML suivant dans la div :

```
<p>
        Pseudo : <input id="pseudo" type="text"/><br>
        Mot de passe : <input id="motdepasse" type="text"/><br>
        <input type="button" value="Enregistrer"
               onclick="javascript:enregistrer()" /><br>
</p>
```

Application Cliente JS – `enregistrer.html` (2/2)

2. Ajouter le fonction JavaScript suivante entre les balises `head` et sauvegarder :

```
<script type="text/javascript">
  function enregistrer() {
    var transport = new
        Thrift.Transport("http://localhost:8080/ChatRoomServeur/chatroom/");
    var protocol  = new Thrift.Protocol(transport);
    var client    = new ChatRoomServiceClient(protocol);

    var utilisateur = new Utilisateur();
    utilisateur.pseudo = $("#pseudo").val();
    utilisateur.motdepasse = $("#motdepasse").val();

    try {
        client.enregistrerUtilisateur(utilisateur);
    } catch(e){
        alert(e);
    }
  }
</script>
```

Application Cliente JS - Déploiement

1. Sous Eclipse, dans la vue `Servers`, faire un click droit sur le `Server > Add and Remove`...
2. Sélectionner `ChatRoomJs` puis `Add` puis `Finish`
3. Lancer ou redémarrer le serveur par click droit sur le serveur puis `Server > Restart in Debug`

   Le serveur se lance, les traces sont disponibles dans la vue console

Application Cliente JS - Test

1. Ouvrir un navigateur web, saisir l'URL `http://localhost:8080/ChatRoomJs/chatroom.html`
2. Cliquer sur le lien Enregistrer un utilisateur
3. Saisir un pseudo et un mot de passe puis cliquer sur Envoyer
4. Aller sous Eclipse, dans la vue `Console`, le pseudo doit y être affiché

Application Serveur - Enregistrer un utilisateur

1. Sous Eclipse, ouvrir la classe `ChatRoomProcessor` et ajouter l'import de `Hashtable`

```
import java.util.Hashtable;
```

2. Ajouter la variable globale suivant :

```
private Hashtable<String, Utilisateur> lstUtilisateurs =
                                        new Hashtable<String, Utilisateur>();
```

3. Modifier le code de `enregistrerUtilisateur()` en le remplaçant par :

```
if(!lstUtilisateurs.containsKey(utilisateur.getPseudo())){
    lstUtilisateurs.put(utilisateur.getPseudo(), utilisateur);
}
```

4. Sauvegarder

Modéle Thrift 4 - `exception.thrift` (1/3)

Thrift propose une gestion d'erreur dans son implémentation que nous allons mettre en œuvre

1. Création d'une structure exception

```
namespace java com.arkea.thrift.data.exception

/**
* Exception pour le service ChatRoom
**/

exception ServiceException {

             /** identifiant erreur */
            1:string id,

            /** message erreur */
            2:string message,
}
```

Modéle Thrift 4 - `exception.thrift` (2/3)

2. Modifier le service `chatroomService.thrift` en ajoutant l'include du fichier `exception.thrit` et en ajoutant l'exception à la déclaration des méthodes

```
include "exception.thrift"

/** Service ChatRoomService */
service ChatRoomService {

        /** Recupérer la liste des Messages */
        list<message.Message> getListeMessage()
                        throws (1:exception.ServiceException se)

        /** Envoyer un message */
        void envoyerMessage(1:message.Message message)
                        throws (1:exception.ServiceException se)

        /** Enregistrer un utilisateur */
        void enregistrerUtilisateur (1: utilisateur.Utilisateur utilisateur)
                        throws (1:exception.ServiceException se)
}
```

Modéle Thrift 4 - `exception.thrift` (3/3)

3. Créer le fichier `exception.thrift` et modifier `chatroomService.thrift`

4. Régénérer toutes les fichiers thrift en Java et JS comme vu précédemment

5. Mettre a jour les deux projets sous Eclipse en suivant la même démarche que lors de la première génération

6. Dans la classe `ChatRoomProcessor.java` ajouter l'import

```
import com.arkea.thrift.data.exception.ServiceException
```

7. Ajouter l'exception `ServiceException` à la signature des méthodes

Application Serveur - Gestion des exceptions

1. Sous Eclipse, ouvrir la classe `ChatRoomProcessor`
2. Modifier le code de `enregistrerUtilisateur()` en ajoutant le cas ou le pseudo de l'utilisateur existe déjà, une exception est levée :

```
if(!lstUtilisateurs.containsKey(utilisateur.getPseudo())) {
    lstUtilisateurs.put(utilisateur.getPseudo(), utilisateur);
} else {
    throw new ServiceException("err1","Pseudo déjà existant !");
}
```

3. Sauvegarder

Application Cliente JS - chatroom.html

1. Ajouter l'inclusion suivante aux fichiers enregistrer.html, message.html et envoyer.html:

```
<script src="./js/exception_types.js" type="text/javascript"></script>
```

2. Sauvegarder, redémarrer le serveur d'application et effectuer un test en envoyant deux fois le même pseudo

3. L'exception doit apparaître dans une boite de dialogue

Application Serveur - Gestion des messages (1/3)

1. Sous Eclipse, ouvrir la classe `ChatRoomProcessor` et ajouter l'import de `LinkedList`

```
import java.util.LinkedList;
```

2. Ajouter la variable globale suivant :

```
private LinkedList<Message> lstMessages = new LinkedList<Message>();
```

3. Modifier le code de `getListeMessage()` en le remplaçant par :

```
return lstMessages;
```

4. Ajouter la méthode privée d'ajout de message, cette dernière est synchronisée pour éviter les soucis d'écriture via le multithread du serveur d'application

Application Serveur - Gestion des messages (2/3)

4. Ajouter la méthode privée d'ajout de message, cette dernière est synchronisée pour éviter les soucis d'écriture

```
private synchronized void addMessage(Message message)
            throws ServiceException {
  try {
    if (message.getUtilisateur().getMotdepasse().equals(
          lstUtilisateurs.get(message.getUtilisateur().
          getPseudo()).getMotdepasse())) {
      if (lstMessages.size() == 20) {
        lstMessages.remove(0);
      }
      lstMessages.add(message);
    } else {
      throw new ServiceException("err2",
        "Identité non valide lors d'un ajout de message !");
    }
  } catch(Exception e) {
    throw new ServiceException("err3",
      "Erreur technique lors d'un ajout de message !");
  }
}
```

Application Serveur - Gestion des messages (3/3)

5. Modifier le code de `envoyerMessage()` en le remplaçant par :

```
addMessage(message);
```

6. Sauvegarder

Application Cliente JS – `envoyer.html` (1/2)

1. Ajouter le code HTML suivant dans la div :

```
<p>
        Pseudo : <input id="pseudo" type="text"/><br>
        Mot de passe : <input id="motdepasse" type="text"/><br>
        Message : <input id="message" type="text" size="100"/><br>
        <input type="button" value="Envoyer"
               onclick="javascript:envoyer()" />
</p>
```

2. Ajouter la fonction javascript ci-dessous dans les balises HEAD puis sauvegarder :

Application Cliente JS – `envoyer.html` (2/2)

```
<script type="text/javascript">
function envoyer() {

  var transport = new Thrift.Transport(
                    "http://localhost:8080/ChatRoomServeur/chatroom/");
  var protocol = new Thrift.Protocol(transport);
  var client = new ChatRoomServiceClient(protocol);

  var message = new Message();
  var utilisateur = new Utilisateur();

  utilisateur.pseudo = $("#pseudo").val();
  utilisateur.motdepasse = $("#motdepasse").val();
  message.utilisateur = utilisateur;
  message.contenu = $("#message").val();

  try {
    client.envoyerMessage(message);
  } catch(e){
    alert(e);
  }
}
</script>
```

Application Cliente JS – `message.html` (1/2)

1. Ajouter le code HTML suivant dans la div :

```
<div id="lstMessage"/><br/>
<input type="button" value="Rafraichir"
                onclick="javascript:rafraichir()" />
```

Application Cliente JS - `message.html` (2/2)

2. Ajouter la fonction JavaScript ci-dessous entre les balises `head` puis sauvegarder :

```
<script type="text/javascript">
function rafraichir() {
  var transport = new
    Thrift.Transport("http://localhost:8080/ChatRoomServeur/chatroom/");
  var protocol = new Thrift.Protocol(transport);
  var client = new ChatRoomServiceClient(protocol);

  try {
    var lstMessage = client.getListeMessage();
  } catch(e){
    alert(e);
  }
  $("#lstMessage").empty();
  for(i=0; i<lstMessage.length;i++) {
    $("#lstMessage").append(lstMessage[i].utilisateur.pseudo +
      " : " + lstMessage[i].contenu + " <br>");
  }
}
</script>
```

Application Cliente JS - Déploiement

1. Redémarrer le serveur d'application
2. Tester les fonctionnalités du service via les pages web.

Modéle Thrift 5 - chatroomService.thrift

1. Modification d'un service pour illustrer la continuité dans la compatibilité

```
/** Service ChatRoomService */
service ChatRoomService {

  /** Recupérer la liste des Messages */
  list<message.Message> getListeMessage()
                        throws (1:exception.ServiceException se)

  /** Envoyer un message */
  bool envoyerMessage(1:message.Message message)
                        throws (1:exception.ServiceException se)

  /** Enregistrer un utilisateur */
  bool enregistrerUtilisateur (1: utilisateur.Utilisateur utilisateur)
                        throws (1:exception.ServiceException se)

}
```

Application Serveur – Ajout de `boolean` retour (1/3)

1. Régénérer tous les fichiers thrift uniquement en Java
2. Mettre a jour le projet `ChatRoomServeur` sous Eclipse en suivant la même démarche que lors de la première génération
3. Désormais la classe `ChatRoomServeur` comporte la nouvelle signature de méthode :

```
@Override
public boolean envoyerMessage() throws ServiceException, TException {
        // TODO Auto-generated method stub
        return false;
}
```

Application Serveur – Ajout de `boolean` retour (2/3)

4. Modifier le code de `envoyerMessage()` :

```
@Override
public boolean envoyerMessage() throws ServiceException, TException {
        addMessage(message);
        return true;
}
```

Application Serveur – Ajout de `boolean` retour (3/3)

4. Faire les mêmes modifications avec `enregistrerUtilisateur()` :

```
public boolean enregistrerUtilisateur(Utilisateur utilisateur)
                          throws ServiceException {

  if(!lstUtilisateurs.containsKey(utilisateur.getPseudo())){
    lstUtilisateurs.put(utilisateur.getPseudo(), utilisateur);
    System.out.println(utilisateur.getPseudo() + " enregistré");
  } else {
    throw new ServiceException("err1","Pseudo déjà existant !");
  }
  return true ;
}
```

5. Sauvegarder la classe

Application Serveur – Test

1. Relancer le serveur d'application.

2. Vérifier à travers des tests que le client JS fonctionne normalement alors qu'il n'a pas été mis à jour

   - Des pop-ups d'avertissement apparaissent pour indiquer que la version a été mise à jour
   - Ceci est dû à notre gestion d'erreurs qui attrape toutes les exceptions techniques et fonctionnelles en JavaScript