Golfier Maxime, Majeune-Bazillou Nicolas, Lathaud Quentin, Philippe Benoît

Scénario

Le scénario est un complément à la javadoc, pour plus d'informations sur les méthodes, veuillez vous y référer. :)

PARTIE MODELE

```java
public interface ICallback extends Remote {

    // Une méthode permettant d'afficher un message envoyé, écrire l'implémentation de cet interface.
    void dispMsg(String message) throws RemoteException;

}
```

```java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CallbackImpl extends UnicastRemoteObject implements ICallback
{

    public CallbackImpl() throws RemoteException
    {
            super();
    }

    @Override
    public void dispMsg(String message)
    {
        System.out.println(message);
    }



}
```

Sur cette page nous avons la partie effectué par le client. C'est une méthode d'affichage très légère.

```java
public interface IMsn extends Remote {

    //Une méthode permettant à un client de s'enregistrer en envoyant son nom ainsi que la référence de son objet callback.
    void addUser(String name, ICallback client) throws RemoteException;

    //Une méthode permettant à un client de savoir si un autre client est en ligne (déjà enregistrer) grâce à un nom.
    boolean isConnected (String name) throws RemoteException;

    //Une méthode permettant d'obtenir la référence d'un objet callback d'un client en ligne grâce à son nom.
    ICallback obtainRef (String name) throws RemoteException;

    //Une méthode permettant à un client de se déconnecter en envoyant son nom.
    void logOut (String name) throws RemoteException;

    //Une méthode permettant à un client d'avoir la map retournant tous les utilisateurs
    Map<String, ICallback> getMap () throws RemoteException;

}
```

```java
public class MsnImpl extends UnicastRemoteObject implements IMsn{

    private Map<String, ICallback> map = new HashMap<>();


    public MsnImpl () throws RemoteException
    {
        super();
    }

    @Override
    public void addUser(String name, ICallback client) throws RemoteException {
        if (!map.containsKey(name)){
            map.put(name,client);
        }
        else {
            System.out.println("Contient déjà l'élement");
        }
    }
```

```java
    @Override
    public boolean isConnected(String name) throws RemoteException {
        if (map.entrySet().stream().anyMatch((entry) -> (entry.getKey().equals(name)))) {
            return true;
        }
        return false;
    }

    @Override
    public ICallback obtainRef(String name) throws RemoteException {
        for (Map.Entry<String,ICallback> entry : map.entrySet()) {
            if (entry.getKey().equals(name)) {
                return entry.getValue();
            }
        }
        return null;
    }
```

```java
@Override
public void logOut(String name) throws RemoteException {
    Iterator<Map.Entry<String,ICallback>> it;
    Map.Entry<String,ICallback> entry;
    it = map.entrySet().iterator();
    while (it.hasNext())
    {
        entry = it.next();
        if (entry.getKey().equals(name)) {
            map.remove(entry.getKey());
        }
    }
}

@Override
public Map<String, ICallback> getMap() throws RemoteException {
    return map;
}
```

Nous avons ici les méthodes du serveur, c'est celle qui nous permet la gestion utilisateur

PARTIE VIEW/CONTROLLER

```java
public class Communaute {

    private final ObservableList<User> lesTravauxObservables = FXCollections.observableArrayList();

    private final ListProperty<User> lesTravaux = new SimpleListProperty<>(lesTravauxObservables);
        public ObservableList<User> getLesTravaux() {return lesTravaux.get();}
        public ReadOnlyListProperty<User> lesTravauxProperty() {return lesTravaux;}

    public Communaute(){

    }
}
```

La communauté seront tous les utilisateurs présents dans la listview, on utilise donc une observablelist pour l'affichage.

```java
@FXML
protected void onValidateButton (ActionEvent event) {
    String name = txt.getText();
    IMsn mn = null;
    ICallback cb = null;
    try {
        Registry rg =LocateRegistry.getRegistry("localhost",6767);
        mn = (IMsn) rg.lookup("Manager");
    } catch (NotBoundException | RemoteException e) {
        showMessage(Alert.AlertType.ERROR, null, "Erreur a l'accès du serveur, message d'erreur : \n\n" + e);
    }
    try {
        cb = new CallbackImpl();
    } catch (RemoteException e) {
        System.out.println("Erreur de création, message d'erreur : \n\n" + e);
    }
    try {
        utilisateur = new User(name,mn,cb);
        utilisateur.enregistreServeur();
        launch();
    } catch (Exception e) {
        showMessage(Alert.AlertType.WARNING, null, "Erreur lors de l'authentification, message d'erreur : \n\n" + e);
    }
}
```

```java
@FXML
protected void deconnexion (ActionEvent event) {
    IMsn mn = null;
    try {
        Registry rg =LocateRegistry.getRegistry("localhost",6767);
        mn = (IMsn) rg.lookup("Manager");
    } catch (NotBoundException | RemoteException e) {
        showMessage(Alert.AlertType.ERROR, null, "Erreur a l'accès du serveur, message d'erreur : \n\n" + e);
    }
    try {
        utilisateur.seDeconnecte();
        showMessage(Alert.AlertType.INFORMATION, null, utilisateur.getName()+ " est déconnecté avec succès !\n On vo
    } catch (Exception e) {
        showMessage(Alert.AlertType.WARNING, null, "Erreur lors de la deconnexion, message d'erreur : \n\n" + e);
    }
}
```

Ce sont les méthodes qui permettent de gérer lorsque le client clique sur un bouton ou effectue une action.