المدرسة العليا للتكنولوجيا - الصويرة
L'ÉCOLE SUPÉRIEURE DE TECHNOLOGIE – ESSAOUIRA
MARRAKECH
جامعة القاضي عياض
UNIVERSITE CADI AYYAD

Travail Pratique Java

CALCULATRICE | BOITE MAIL

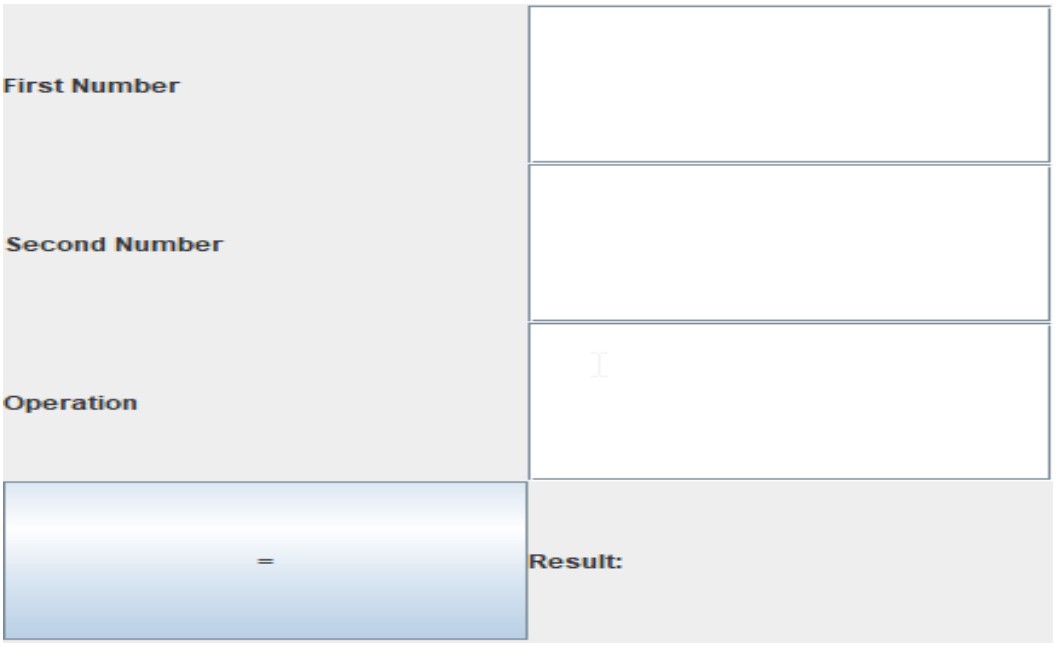
Mohamed EDDAHBY | 20 décembre 2023

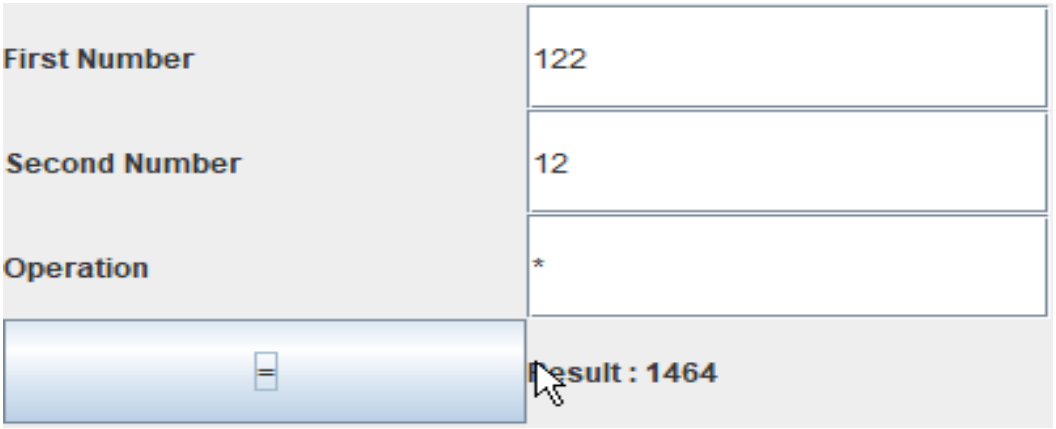
I. Calculatrice

Le but de ce travail pratique est mis en œuvre une interface graphique. Qui permet de calculer de nombre simple grâce à l'opération artistiques.
le calcul doit être dans un autre Project sous forme d'un server, pour faire une laissant entre ses deux projets.
le client envoyer le nombre a calculé est l'opération

Le server calculer les résultats.

➢ L'interface





1. Coté Client :

➢ Les composant de l'interface graphique.

```
public Calculator() {
        l1 = new JLabel("First Number");
        l2 = new JLabel("Second Number");
        l3 = new JLabel("Operation");
        tx1 = new JTextField();
        tx2 = new JTextField();
        tx3 = new JTextField();
        bEquals = new JButton("=");

        panel = new JPanel();
        panel.setLayout(new GridLayout(4, 2));
        panel.add(l1);
        panel.add(tx1);
        panel.add(l2);
        panel.add(tx2);
        panel.add(l3);
        panel.add(tx3);
        panel.add(bEquals);
```

➢ La fonction qui invoque la fonction Sent to server

```
bEquals.addActionListener(e -> {
        CalculatorData calculatorData = new CalculatorData(
                Integer.parseInt(tx1.getText()),
                Integer.parseInt(tx2.getText()),
                tx3.getText());
        String result = sendToServer(calculatorData);

        resultLabel.setText("Result : " + result);
    });

```

➢ Fonction Send to server

```java
private String sendToServer(CalculatorData calculatorData) {
      try {
          Socket socket = new Socket("localhost", 9999);
          ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
          oos.writeObject(calculatorData);

          ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
          String result = (String) ois.readObject();

          oos.close();
          ois.close();
          socket.close();

          return result;
      } catch (Exception e) {
          e.printStackTrace();
          System.out.println("error !!!");
          return "";
      }
  }
```

2. Coté Server:

➢ Le coté server :

```java

    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(9999);
            System.out.println("Server waiting for client on port 9999...");

            while (true) {
                Socket socket = serverSocket.accept();
                ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
                CalculatorData calculatorData = (CalculatorData) ois.readObject();

                String result = calculatorData.Operation();

                ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
                oos.writeObject(result);

                ois.close();
                oos.close();
                socket.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

➢ La fonction qui calcule

```
1
2   public String Operation() {
3       String total;
4       switch (operation) {
5           case "+":
6               total = String.valueOf(value1 + value2);
7               break;
8           case "-":
9               total = String.valueOf(value1 - value2);
10              break;
11          case "%":
12              total = String.valueOf((float) value1 / value2);
13              break;
14          case "*":
15              total = String.valueOf(value1 * value2);
16              break;
17          default:
18              total = "";
19              break;
20      }
21      return total;
22  }
```

Cette fonction calculer les deux nombre réservoir dépend de l'opération fait dans la partie Client.

II. Boit Mail

Cette exercice sir a créé une boite mail pour affiche les emails pour chaque personne qui déjà enregistré dans la base de données.

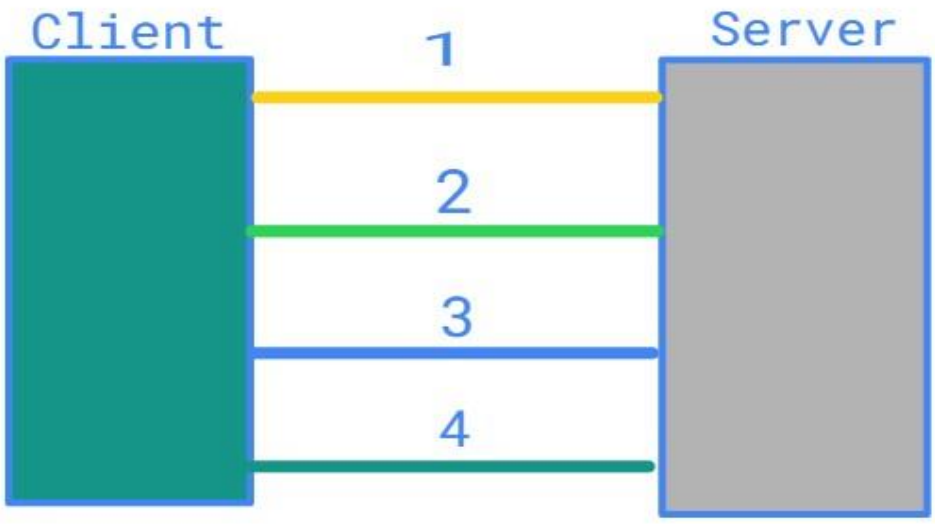
Les utilisateurs vont passer par une phase de autocitations (login) pour voir leur message et pour qu'il peut ajouter des autres.

Aux neveux techniques pour mettre en œuvre cette applications il faut utiliser plusieurs concepts de langage de programmation Java parmi eux :

- L'utilisation de JDBC (base de données pour sauvegarder les données des utilisateurs).
- L'utilisation des sockets est les threads pour que l'application soit flexible.
- L'utilisation des design pattern

Pour faciliter l'échange de donnée entre la partie client et la partie serveur j'ai choisi de la faire en quatre étapes.

Comme suivant :



- Le client envoyer un nombre a serveur pour l'informé de l'opération qu'il va faire
- Le serveur confirmer l'opération
- Le client envoyer les données
- Les serveurs recevoir les données de client elles les traités puis il les envois de neveux au client

➢ La table Personne

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	nom	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
3	prenom	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
4	password	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More

➢ La table Personne

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	titre	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
3	text	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
4	client_ID	int(11)			No	None			Change Drop More

1. Coté Client:

➢ La classe Personne

```java
public class Personne implements Serializable {
    static  private final long serialVersionUID= 6L;
    private int id;
    private String nom ;
    private String prenom;
    private String password;
    public Personne(String nom, String prenom, String password) {
        this.nom = nom;
        this.prenom = prenom;
        this.password = password;
    }
```

➢ La classe Message

```java
public class Message implements Serializable {
static  private final long serialVersionUID= 6L;
    private int id ;
    private String titre ;
    private String message;
    private int clientID;

    public Message( String titre, String message, int clientID) {
        this.titre = titre;
        this.message = message;
        this.clientID = clientID;
    }

```

➢ La classe Client

Cette classe contient les sockets qui va communiquer avec le coté serveur aussi des composant graphique pour construire une interface graphique.

```java
public class Client extends JFrame {
    JMenuBar menubar;
    JMenu fileMenu, editMenu, helpMenu,authMenu;
    JMenuItem openItem, saveItem, cutItem, copyItem, pasteItem, aboutItem ,loginItem,registerItem ;
    JButton addButton, readButton, deleteButton, loginButton, registerButton;
    JLabel nomLabel, prenomLabel, passwordLabel;
    JTextField nomField, prenomField, passwordField;
    JTextArea messages;
    JList<String> emailList;
    DefaultListModel<String> emailListModel;
    JPanel mainPanel, buttonPanel, messagePanel, menuPanel;
    Personne personne  ;
    private static Personne  loggedInPerson;
    ObjectInputStream testing;
```

➢ Les fonctions de la classe Client

```java
public Client() {···
private void messagesList() {···
private void addMessage(String titre ,String messageText ) {···
private void addMessageDialog() {···
private void showLoginForm() {···
private void showRegisterForm() {···

Run | Debug
public static void main(String[] args) {
    new Client();
}
}
```

Chaque un de ces fonction se compose d'un socket et les éléments nécessaire pour envoyer est recevoir des données à partir de server.

On prend par exemple la fonction messageList qui permet de retourner une List des messages.

```java
    private void messagesList() {
        PrintWriter pn;
        ObjectInputStream testing;
        Socket socket;
        try {
            socket = new Socket("localhost", 9999);
            System.out.println("Connected ... ");
            pn = new PrintWriter(socket.getOutputStream(), true);
            pn.println("1");
            if(loggedInPerson != null){
            testing = new ObjectInputStream(socket.getInputStream());
            List<Message> message = (List<Message>) testing.readObject();
            System.out.println(message.size());
            StringBuilder messageText = new StringBuilder();
            for (Message m : message) {
                messageText.append(m.toString()).append("\n");
            }
                messages.setText(messageText.toString());
        }else {
                StringBuilder messageText = new StringBuilder();
                messages.setText(messageText.toString());
            }

    } catch (IOException | ClassNotFoundException ex) {
        throw new RuntimeException(ex);
    }
}

```

Cette fonction envoyer au début un nombre au serveur pour qu'il traiter ce cas dépond de ce nombre. Puis il attende que le serveur envoyer un objet. Cette Objet est une sérialisation d'une List des Object de types Message. Une fois il recoin cette objet il le traiter est le mais dans une composant graphique de type TextAea.

La même chose pour les autres fonctions

1. Coté Server:

➢ La classe DBConnection

```java
public class DBConnection {
    private static Connection conn;

    static {
        try {
            //TODO: JNDI methode ...!
        /* Context context = new InitialContext();
            DataSource dataSource = (dataSource) context.lookup("com.mysql.cj.jdbc.Driver");
            conn = dataSource.getConnection("jdbc:mysql://localhost/Cabinier","root", ""));*/
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql://localhost/BoitDialog", "root", "");
            System.out.println("Connected !");
        } catch (SQLException e) {
            System.out.println("SQL problems");
        } catch (ClassNotFoundException e) {
            System.out.println("Driver not exist");
        }
    }

    public static Connection getConnection() {
        return conn;
    }
}
```

Cette classe permet la connexion avec la base de données on appliquant le design pattern Singleton pour garantir qu'on aura une seul connexion dans notre projet.

➢ L'interface Gestion

```java
import java.util.List;

public interface Gestion<T>{
        public T getT(String id );
        public List<T> getAllT(int id);
        public void addT(T t);
        public void editT(T t , int id);
        public void deleteT(int id );
}

```

Cette Interface va être implémenté par des classes se qui facilite l'implémentation de ces classes en appliquant les design pattern DAO

➢ Classe Server

```
public class Server {
    static Personne loggedInPerson ;
    Run | Debug
    public static void main(String[] args) {···

    private static void handleClient(Socket socket ) {···
    }
```

Cette Classe possède deux fonction la fonction appellent (main) et une fonction handleClient qui permet de traiter les données recevoir.

On verra par détails chaque fonction et leur traitement.

➢ La fonction main

```
1  public static void main(String[] args) {
2      try {
3          ServerSocket serverSocket = new ServerSocket(9999);
4          System.out.println("Server waiting for client on port 9999...");
5          while (true) {
6              Socket socket = serverSocket.accept();
7              new Thread(() -> handleClient(socket)).start();
8          }
9      } catch (Exception e) {
10         e.printStackTrace();
11     }
12 }
```

Cette fonction permet de crée un thread pour chaque socket vient du client.

Cette fonction chaque fois une Cockett client se connecter avec e server il appelé la fonction handelClient.

➢ La fonction handleClient

```java
private static void handleClient(Socket socket ) {
    MessageImplementation cl = new MessageImplementation();
    ClientImplementation client = new ClientImplementation();
    ClientSide.Personne p = null;
    List<Personne> lp = new ArrayList<>();
    PrintWriter pn;
    ClientSide.Message message = null;
    List<Message> ms ;
    BufferedReader bff;
    PrintWriter ptn;
    BufferedReader bf;
    ObjectOutputStream outputDataO;
    ObjectInputStream inputDataO;

```

Cette fonction contient plusieurs éléments qu'ils vont être utilisé par la suite.

```java
try {
                bf = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                String requestType =  bf.readLine();

       if("3".equals(requestType)) {
           ptn = new PrintWriter(socket.getOutputStream(), true);
           ptn.println("done...");
           inputDataO = new ObjectInputStream((socket.getInputStream()));
               Personne userName=  (Personne) inputDataO.readObject();
               ClientSide.Personne pp = client.login(userName.getNom(), userName.getPassword());
           if(pp != null){
               outputDataO = new ObjectOutputStream(socket.getOutputStream());
               outputDataO.writeObject(pp);
               loggedInPerson = pp;
           }else{
               pn = new PrintWriter(socket.getOutputStream(), true);
               pn.println("false");
               }
               inputDataO.close();
               bf.close();
       }
```

Comme j'ai déjà mentionné le client envoi un code pour spécifier l'opération il va faire pour le server interagir dépend de code envoyer.
par exemple dans cette cas le server va revoit le nombre 3, puis il va envoyer une confirmation qu'il attende le client à envoyer les données. Par la suit il les traités et il les retourner au client.

Au moment de traitement la fonctions handleClient appelé des fonctions des classes qui implémente l'interface Gestion et qui sont utilisé l'instance de connexion qui déjà créer au début.

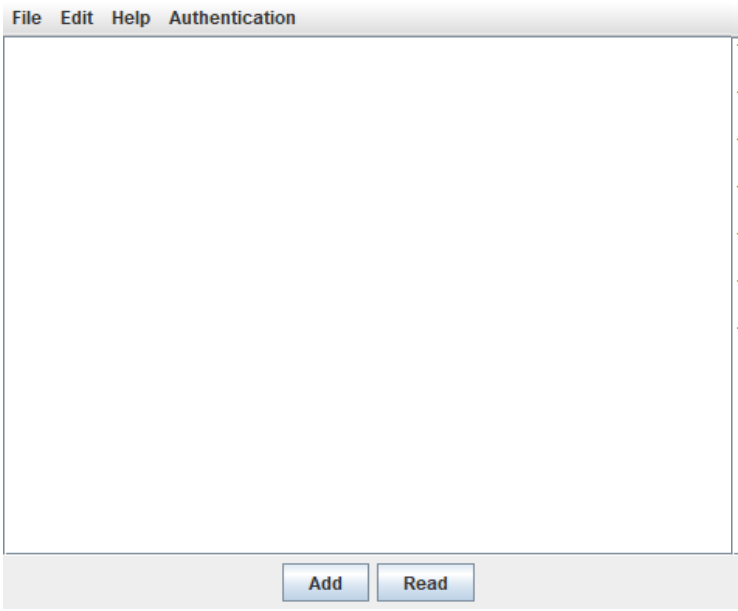
➢ La suite de fonction handleClient

```
if  ("2".equals(requestType)){
                ptn = new PrintWriter(socket.getOutputStream(), true);
                ptn.println("done ... ! ");
                inputDataO = new ObjectInputStream(socket.getInputStream());
                p= (ClientSide.Personne) inputDataO.readObject();
                client.addT(p);
                pn = new PrintWriter(socket.getOutputStream(), true);
                pn.println("true");
                inputDataO.close();
                bf.close();
        }if  ("1".equals(requestType)){
                System.out.println("hello from here ");
                ms = new ArrayList<>();
                ms = cl.getAllT(loggedInPerson.getId());
                ms = cl.getAllT(loggedInPerson.getId());
                outputDataO = new ObjectOutputStream(socket.getOutputStream());
                outputDataO.writeObject(ms);
                outputDataO.flush();
                outputDataO.close();
        }
```

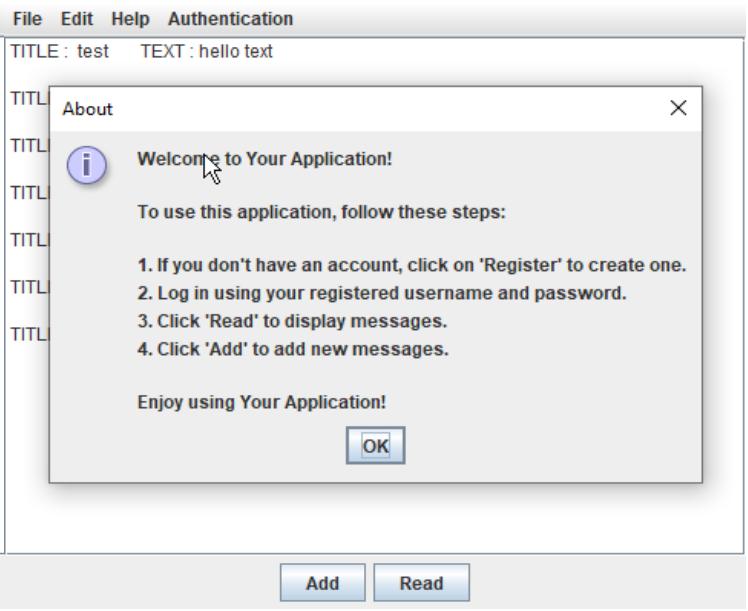
➢ La suite de fonction handleClient

```
if("4".equals(requestType)){
                ptn = new PrintWriter(socket.getOutputStream(), true);
                ptn.println("done ... ! ");
                inputDataO = new ObjectInputStream(socket.getInputStream());
                Message messageR = (Message) inputDataO.readObject();
                cl.addT(messageR);
                pn = new PrintWriter(socket.getOutputStream(), true);
                pn.println("true");
                inputDataO.close();
                bf.close();
            }socket.close();
            } catch (IOException | ClassNotFoundException ex) {
            throw new RuntimeException(ex);
         }
    }
    }
```

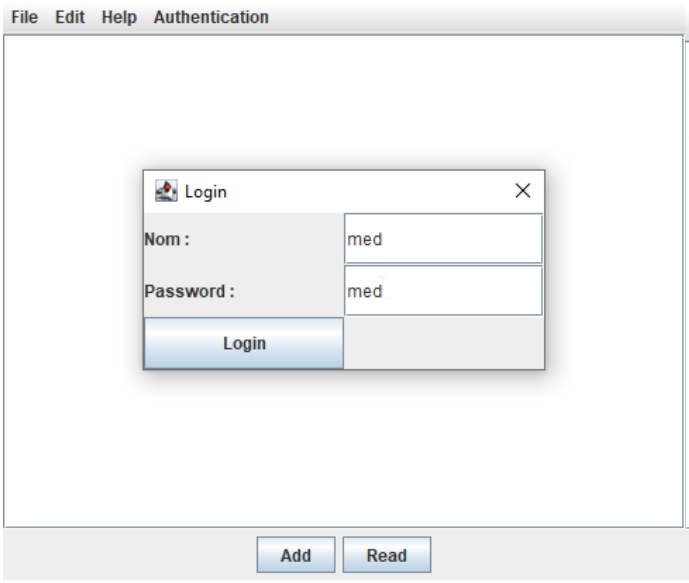
Interface graphique



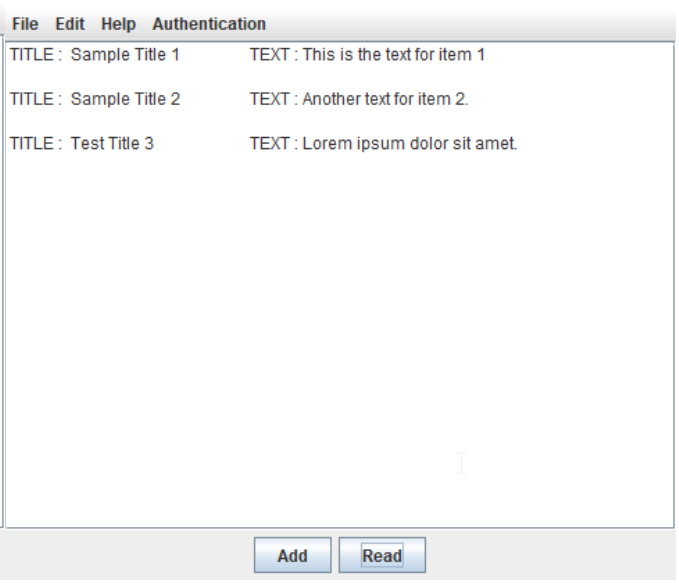
About



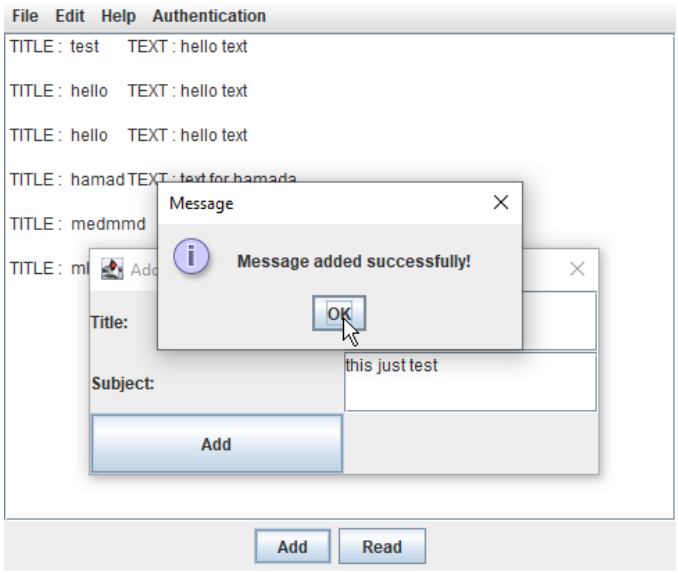
Login

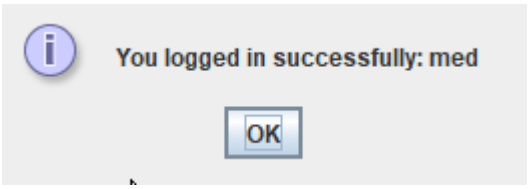


Les messages



Ajouter un message





Conclusion : :

En conclusion, le développement de l'interface graphique connectée au serveur pour la création de la boîte mail a été réalisé avec succès.

. Les choix techniques, tels que les design pattern, les threads, les et sockets ont été cruciaux pour atteindre les performances optimales attendues.