



Rapport de Mini-projet JAVA

Établissement : EMSI Casablanca

Site : Centre

Classe : 4IIR

Groupe : 4

Titre :

Application de Chat



Encadré par :

El Azhari Khadija

Réalisé par :

Haoud Soufiane

Année universitaire : 2022/2023

Remerciement :

Tout d'abord, nous adressons nos remerciements à notre professeur de l'élément de JAVA avancé Mme ELAzhari qui nous a énormément aidés à accumuler le maximum d'informations ainsi que ses conseils et ses recommandations lors des séances.

Résumé

Ce rapport a pour objectif de présenter une application de chat avancée développée en utilisant le langage de programmation Java. Il vise à fournir une analyse détaillée de l'application, en mettant en évidence ses fonctionnalités clés, son architecture et les défis rencontrés lors du processus de développement.

L'objectif principal de ce rapport est de démontrer la valeur de l'application de chat avancée en Java et de mettre en évidence les avantages qu'elle offre aux utilisateurs. Il mettra en évidence les fonctionnalités spécifiques qui rendent cette application supérieure aux alternatives existantes, en mettant l'accent sur son interface utilisateur conviviale, ses fonctionnalités avancées et sa sécurité renforcée.

Ce rapport abordera également les objectifs techniques du projet, tels que l'utilisation de bibliothèques et de frameworks Java, la mise en place de protocoles de cryptage et la prise en compte des performances et de la scalabilité de l'application.

En outre, le rapport explorera les perspectives d'amélioration et d'extension de l'application, en identifiant les fonctionnalités potentielles qui pourraient être ajoutées à l'avenir pour répondre aux besoins changeants des utilisateurs.

L'objectif global de ce rapport est de fournir une évaluation complète de l'application de chat avancée en Java, en mettant en évidence ses avantages, ses limites et ses perspectives futures. Il vise à informer les lecteurs sur les aspects techniques et fonctionnels de l'application, ainsi que sur ses implications pour les utilisateurs et les développeurs.

En conclusion, ce rapport se concentre sur l'objectif de présenter et d'évaluer une application de chat avancée en Java, en mettant en évidence ses caractéristiques distinctives et sa valeur ajoutée pour les utilisateurs.

Table des matières

| | |
|---|----|
| Introduction..... | 6 |
| Chapitre I: Cahier des charges fonctionnel..... | 6 |
| 1.1.Problématique | 6 |
| 1.2.Objectifs | 6 |
| Chapitre II: Programmation et solution logicielle | 7 |
| 1.1. IDE et langage utilisés | 7 |
| 1.2. Contrôleurs..... | 7 |
| 1.3. La partie serveur | 13 |
| 1.4. La partie main | 16 |
| 1.5. La partie main | 17 |
| Chapitre III: Bilan de projet et simulation | 18 |
| Conclusions et perspectives | 21 |

Table des figures

| | |
|---|----|
| figure 1 : Contrôleur de connexion..... | 8 |
| figure 2 : Contrôleur d'inscription..... | 10 |
| figure 3 : Contrôleur de l'interface de chat..... | 12 |
| figure 4 : La partie serveur..... | 15 |
| figure 5 : La partie main | 16 |
| figure 6 : Interface graphique de connexion..... | 17 |
| figure 7 : Interface graphique d'inscription..... | 17 |
| figure 8 : Interface graphique du chat..... | 18 |
| figure 9 : Démarrage du serveur | 18 |
| figure 10 : Connexion de deux utilisateurs..... | 19 |
| figure 11 : Connexion au serveur..... | 19 |
| figure 12 : Echange des messages | 19 |
| figure 13 : Nouveau utilisateur | 20 |
| figure 14 : Connexion au serveur pour l'utilisateur ajouté..... | 20 |
| figure 15 : échange de message avec un utilisateur spécifique | 21 |

Introduction :

Ce rapport présente une application de chat développée en utilisant le langage de programmation Java. Le chat est devenu un moyen de communication essentiel dans notre société moderne, et cette application vise à offrir une expérience de chat riche et conviviale aux utilisateurs.

L'objectif principal de ce projet était de créer une application de chat performante, en exploitant les fonctionnalités avancées offertes par Java. Pour cela, nous avons utilisé diverses bibliothèques et frameworks Java populaires, tels que JavaFX et Socket, afin de fournir une interface utilisateur intuitive et une communication en temps réel entre les utilisateurs.

Dans cette application, les utilisateurs ont la possibilité de créer un compte personnel et de se connecter en utilisant leurs identifiants. Une fois connectés, ils peuvent échanger des messages texte,

En outre, cette application de chat avancée offre une interface utilisateur élégante et conviviale. Nous avons utilisé JavaFX, une bibliothèque de développement d'interfaces graphiques, pour créer des fenêtres, des boutons et des champs de texte interactifs. Cela permet aux utilisateurs de naviguer facilement dans l'application et d'interagir de manière intuitive avec les fonctionnalités de chat.

Dans les sections suivantes de ce rapport, nous décrirons en détail l'architecture de l'application, les fonctionnalités implémentées, les défis rencontrés lors du développement et les améliorations potentielles qui pourraient être apportées à l'application.

En conclusion, cette application de chat avancée en Java offre une plateforme de communication moderne et sécurisée. Elle répond aux besoins croissants des utilisateurs en matière de chat en fournissant des fonctionnalités avancées telles que la messagerie instantanée, les appels audio/vidéo et la gestion des groupes. L'application offre également une expérience utilisateur agréable grâce à son interface intuitive et conviviale.

Chapitre I : Cahier des charges fonctionnel

1.1 : Problématique

Comment développer une application de chat en Java offrant une interface utilisateur intuitive, des fonctionnalités avancées et une sécurité robuste pour répondre aux exigences croissantes de communication instantanée ?

1.2 : Objectifs

Notre projet avait comme objectifs de :

- Communication en temps réel entre les utilisateurs
- Interface graphique simple

Chapitre II : Programmation et solution logicielle

1.1 : IDE et langage utilisés :

L'application de chat avancée a été développée en utilisant le langage de programmation Java et l'IDE (Integrated Development Environment) IntelliJ IDEA. Java est un langage polyvalent et largement utilisé, reconnu pour sa portabilité, sa robustesse et sa capacité à prendre en charge des applications complexes. IntelliJ IDEA est un environnement de développement populaire, offrant des fonctionnalités avancées telles que l'assistance au code, le débogage et la gestion de projets, ce qui en fait un choix courant parmi les développeurs Java.

1.2 : Contrôleurs

Dans cette partie, on va expliquer l'ensemble des contrôleurs et des notions utilisées dans le programme.

```
public class Scene1Controller {
    4 usages
    @FXML
    private TextField usernameId;
    2 usages
    @FXML
    private PasswordField passwordId;

    1 usage
    @FXML
    protected void onLogin() throws IOException, SQLException {

        Connection conn = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/users", user: "root", password: "");
        // Connected to database successfully...

        Statement stmt = conn.createStatement();
        String sql = "SELECT * FROM utilisateurs WHERE email=? AND password=?";
        PreparedStatement preparedStatement = conn.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1, usernameId.getText());
        preparedStatement.setString( parameterIndex: 2, passwordId.getText());

        ResultSet resultSet = preparedStatement.executeQuery();
    }
}
```

```

if (resultSet.next()) {
    Stage s=(Stage) usernameId.getScene().getWindow();
    //récupérer fxml de la deuxième scene
    FXMLLoader fx=new FXMLLoader(MainApp.class.getResource( name: "Scene2.fxml"));
    Scene sc2=new Scene(fx.load());
    //attacher la scene au stage
    s.setScene(sc2);
    stmt.close();
    conn.close();
}
else{
    Alert alert=new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Authentication Error");
    alert.setHeaderText("Username or password are not validated !");
    alert.setContentText("You can retry by changing the authentication information ");
    alert.show();
}
}

```

```

1 usage
@FXML
protected void getregistration() throws IOException {
    Stage s=(Stage) usernameId.getScene().getWindow();
    //récupérer fxml de la scene
    FXMLLoader fx=new FXMLLoader(MainApp.class.getResource( name: "scene3.fxml"));
    Scene sc3=new Scene(fx.load());
    //attacher la scene au stage
    s.setScene(sc3);
}
}

```

Figure 1 : Contrôleur de connexion

Ce code correspond à la classe "Scene1Controller" dans l'application. Cette classe est responsable de la gestion de la première scène de l'application de chat. Voici une explication du code :

- Les importations nécessaires sont effectuées pour les classes et les packages utilisés dans le code.
- La classe "Scene1Controller" est définie et elle correspond au contrôleur de la première scène de l'application.
- Les annotations @FXML sont utilisées pour lier les éléments de l'interface utilisateur définis dans le fichier FXML à des variables dans le code.
- La méthode "onLogin()" est appelée lorsque l'utilisateur clique sur le bouton de connexion. Cette méthode est responsable de la vérification des informations de connexion dans la base de données.

- Une connexion à la base de données est établie en utilisant JDBC (Java Database Connectivity). Les informations de connexion sont fournies en utilisant l'URL, le nom d'utilisateur et le mot de passe appropriés.
- Une requête SQL est préparée pour vérifier si les informations de connexion fournies correspondent à celles stockées dans la table "utilisateurs" de la base de données.
- Le résultat de la requête est vérifié à l'aide de la méthode "next()" de l'objet ResultSet. Si les informations de connexion sont valides, la scène 2 est chargée et affichée.
- Sinon, une boîte de dialogue d'erreur est affichée pour informer l'utilisateur que le nom d'utilisateur ou le mot de passe n'est pas valide.
- La méthode "getregistration()" est appelée lorsque l'utilisateur souhaite s'inscrire. Cette méthode charge et affiche la scène 3 de l'application.

Ce code illustre donc la gestion de la connexion et de l'authentification des utilisateurs, ainsi que la navigation entre les différentes scènes de l'application de chat en utilisant JavaFX.

```
public class Scene3Controller {
    3 usages
    @FXML
    private TextField usernameId;
    2 usages
    @FXML
    private PasswordField passwordId;

    1 usage
    @FXML
    protected void onRegister() throws IOException, SQLException {

        Connection conn = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/users", user: "root", password: "");
        // Connected to database successfully...

        Statement stmt = conn.createStatement();
        String sql = "INSERT INTO utilisateurs (email, password) " +
            "VALUES (?, ?)";
        PreparedStatement preparedStatement = conn.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1, usernameId.getText());
        preparedStatement.setString( parameterIndex: 2, passwordId.getText());
    }
}
```

```

int addedRows = preparedStatement.executeUpdate();

if (addedRows>0) {
    Stage s = (Stage) usernameId.getScene().getWindow();
    //récupérer fxml de la deuxième scene
    FXMLLoader fx = new FXMLLoader(MainApp.class.getResource( name: "Scene2.fxml"));
    Scene sc2 = new Scene(fx.load());
    //attacher la scene au stage
    s.setScene(sc2);
    stmt.close();
    conn.close();
} else {
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Authentication Error");
    alert.setHeaderText("Username or password are not validated !");
    alert.setContentText("You can retry by changing the authentication information ");
    alert.show();
}
}
}

```

Figure 2 : contrôleur d'inscription

Ce code correspond à la classe "Scene3Controller" dans l'application. Cette classe est responsable de la gestion de la troisième scène de l'application de chat, qui est dédiée au processus d'inscription des utilisateurs.

- Les importations nécessaires sont effectuées pour les classes et les packages utilisés dans le code.
- La classe "Scene3Controller" est définie et elle correspond au contrôleur de la troisième scène de l'application.
- Les annotations @FXML sont utilisées pour lier les éléments de l'interface utilisateur définis dans le fichier FXML à des variables dans le code.
- La méthode "onRegister()" est appelée lorsque l'utilisateur clique sur le bouton d'inscription. Cette méthode est responsable de l'ajout des informations d'inscription de l'utilisateur à la base de données.
- Une connexion à la base de données est établie en utilisant JDBC (Java Database Connectivity). Les informations de connexion sont fournies en utilisant l'URL, le nom d'utilisateur et le mot de passe appropriés.
- Une requête SQL est préparée pour insérer les informations d'inscription fournies dans la table "utilisateurs" de la base de données.
- Les informations d'inscription sont extraites des champs de texte appropriés et sont définies comme paramètres de la requête préparée.
- La méthode "executeUpdate()" est appelée pour exécuter la requête d'insertion dans la base de données et renvoie le nombre de lignes ajoutées.
- Si au moins une ligne est ajoutée avec succès, la scène 2 est chargée et affichée.
- Sinon, une boîte de dialogue d'erreur est affichée pour informer l'utilisateur que le nom d'utilisateur ou le mot de passe n'a pas été validé.

```

public class Scene2Controller {

    2 usages
    @FXML
    private TextField MymsgID;
    2 usages
    @FXML
    private TextField PortID;
    2 usages
    @FXML
    private TextField HostID;
    2 usages
    @FXML
    private ListView testview;
    2 usages
    PrintWriter pw;
    1 usage
    @FXML
    protected void onconnect() throws IOException {
        String host=HostID.getText();
        int port=Integer.parseInt(PortID.getText());
        //socket
        Socket s=new Socket(host,port);
        InputStream is=s.getInputStream();//octet
        InputStreamReader isr=new InputStreamReader(is);//caractere
    }
}

```

```

        BufferedReader br=new BufferedReader(isr);
        OutputStream os=s.getOutputStream();//octet
        pw=new PrintWriter(os, autoFlush: true);
        new Thread()-> {
            while(true){
                try {
                    String reponse = br.readLine();
                    Platform.runLater()-> {
                        testview.getItems().add(reponse);
                    };
                }catch(IOException e){
                    e.printStackTrace();
                }
            }
        }).start();
    }

    1 usage
    @FXML
    public void onsubmit(){
        String message=MymsgID.getText();
        pw.println(message);
    }

```

Figure 3 : contrôleur de l'interface de chat

Ce code correspond à la classe "Scene2Controller" dans l'application. Cette classe est responsable de la gestion de la deuxième scène de l'application de chat, où les utilisateurs peuvent se connecter à un serveur et envoyer/recevoir des messages.

- La classe "Scene2Controller" est définie et elle correspond au contrôleur de la deuxième scène de l'application.
- Les annotations @FXML sont utilisées pour lier les éléments de l'interface utilisateur définis dans le fichier FXML à des variables dans le code.
- Les variables sont déclarées pour représenter les éléments de l'interface utilisateur, tels que les champs de texte (MymsgID, PortID, HostID) et la liste de visualisation (testview).
- Un objet PrintWriter (pw) est créé pour écrire des données dans le flux de sortie (OutputStream) du socket.
- La méthode "onconnect()" est appelée lorsque l'utilisateur clique sur le bouton de connexion. Cette méthode est responsable de l'établissement de la connexion avec le serveur.
- Les valeurs saisies par l'utilisateur dans les champs de texte (adresse hôte et port) sont extraites.

- Un objet Socket est créé pour établir une connexion avec le serveur en utilisant l'adresse hôte et le port fournis.
- Les flux d'entrée (InputStream) et de sortie (OutputStream) sont obtenus à partir du socket pour permettre la communication avec le serveur.
- Un objet BufferedReader est créé pour lire les données envoyées par le serveur à partir du flux d'entrée.
- Un thread est créé pour écouter en boucle les réponses du serveur. Les réponses sont lues ligne par ligne à l'aide de la méthode "readLine()" du BufferedReader, puis ajoutées à la liste de visualisation (testview) de manière sécurisée grâce à la méthode "Platform.runLater()".
- La méthode "onsubmit()" est appelée lorsque l'utilisateur souhaite soumettre un message. Le texte saisi dans le champ de texte (MymsgID) est extrait et envoyé au serveur en utilisant le PrintWriter (pw) pour écrire dans le flux de sortie.

1.3: La partie serveur :

```
public class ChatWithServer extends Thread{
    2 usages
    private int ClientNbre;
    2 usages
    private List<Communication> cliensconnectés=new ArrayList<>();

    public static void main(String[] args){

        new ChatWithServer().start();
    }
    @Override
    public void run(){
        try{
            ServerSocket ss=new ServerSocket( port: 1234);

            System.out.println("Le serveur essaie de démarrer ...");

            while(true){
                Socket s=ss.accept();
                ++ClientNbre;
                Communication NewCommunication=new Communication(s,ClientNbre);
                cliensconnectés.add(NewCommunication);
            }
        }
    }
}
```

```

        NewCommunication.start();
    }

} catch (IOException e) {
    throw new RuntimeException(e);
}
}

5 usages
public class Communication extends Thread {
    8 usages
    private Socket s;
    4 usages
    private int ClientNumber;
    1 usage
    Communication(Socket s, int ClientNumber) {
        this.s = s;
        this.ClientNumber = ClientNumber;
    }

    @Override
    public void run() {
        try {
            InputStream is = s.getInputStream(); // octet
            InputStreamReader isr = new InputStreamReader(is); // caractere
            BufferedReader br = new BufferedReader(isr);
            OutputStream os = s.getOutputStream(); // octet
            String Ip = s.getRemoteSocketAddress().toString();

            System.out.println("le client numero : " + ClientNumber + " et son IP : " + Ip);
            PrintWriter pw = new PrintWriter(os, true);
            pw.println("Vous etes le client " + ClientNumber);
            pw.println("Envoyer un message :");

            while (true) {
                String UserRequest = br.readLine();
                if (UserRequest.contains("=>")) {
                    String[] usermessage = UserRequest.split("=>");
                    if (usermessage.length == 2) {
                        String msg = usermessage[1];
                        int numeroClient = Integer.parseInt(usermessage[0]);
                        Send(msg, s, numeroClient);
                    }
                } else {
                    Send(UserRequest, s, -1);
                }
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```

2 usages
void Send(String UserRequest,Socket socket,int nbr) throws IOException {
    for(Communication client : cliensconnectés){
        if(client.s !=socket){
            if(client.ClientNumber==nbr || nbr==--1){
                PrintWriter pw=new PrintWriter(client.s.getOutputStream(), autoFlush: true);
                pw.println(UserRequest);
            }
        }
    }
}
}
}

```

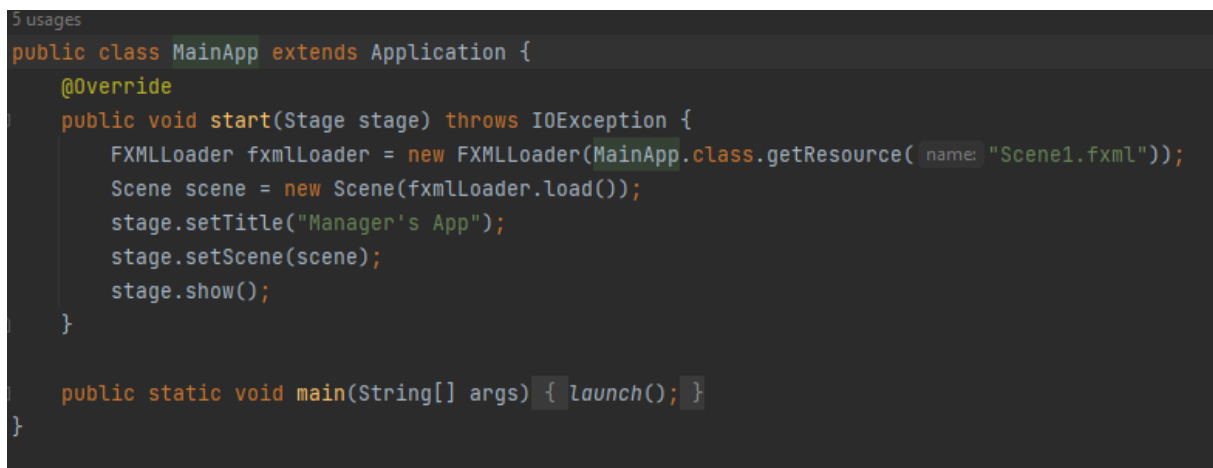
Figure 4 : la partie serveur

Ce code correspond à la classe "ChatWithServer" dans l'application. Cette classe est responsable de la mise en place du serveur de chat qui accepte les connexions des clients et gère la communication entre eux.

- Les importations nécessaires sont effectuées pour les classes utilisées dans le code.
- La classe "ChatWithServer" étend la classe Thread pour permettre l'exécution asynchrone du serveur.
- La variable "ClientNbre" est utilisée pour suivre le nombre de clients connectés.
- La liste "cliensconnectés" est utilisée pour stocker les instances de la classe "Communication" représentant les connexions avec les clients.
- La méthode "main()" est utilisée pour démarrer le serveur en créant une instance de "ChatWithServer" et en appelant la méthode "start()" pour exécuter le serveur dans un thread séparé.
- La méthode "run()" est exécutée lorsque le serveur démarre. Un objet ServerSocket est créé sur le port 1234 pour écouter les connexions entrantes.
- Le serveur entre dans une boucle infinie où il accepte les connexions des clients. Chaque fois qu'un client se connecte, une nouvelle instance de la classe "Communication" est créée pour gérer la communication avec ce client. Cette instance est ajoutée à la liste "cliensconnectés" et démarre dans un thread séparé.
- La classe interne "Communication" représente la communication avec un client spécifique.
- La méthode "run()" de la classe "Communication" est exécutée lorsque la communication avec le client démarre. Les flux d'entrée (InputStream) et de sortie (OutputStream) sont obtenus à partir du socket pour permettre la communication avec le client.
- Le numéro du client et son adresse IP sont affichés.
- Un objet PrintWriter est créé pour écrire des données dans le flux de sortie (OutputStream) du socket, et un message d'accueil est envoyé au client.

- Le serveur entre ensuite dans une boucle où il lit les demandes du client à l'aide de la méthode "readLine()" du BufferedReader. Les demandes peuvent être des messages à envoyer à tous les clients ou des messages privés à un client spécifique.
- Si la demande contient "=>", cela signifie qu'il s'agit d'un message privé. Le message est extrait et envoyé au client approprié à l'aide de la méthode "Send()".
- Sinon, le message est envoyé à tous les clients en utilisant la méthode "Send()" avec le paramètre -1.
- La méthode "Send()" est responsable de l'envoi du message aux clients appropriés. Elle parcourt la liste des clients connectés et envoie le message à tous les clients sauf à celui qui l'a envoyé.
- Le code utilise des flux d'entrée/sortie et des objets PrintWriter pour la communication avec les clients via les sockets.

1.4: La partie main :



```

5 usages
public class MainApp extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(MainApp.class.getResource("Scene1.fxml"));
        Scene scene = new Scene(fxmlLoader.load());
        stage.setTitle("Manager's App");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) { launch(); }
}

```

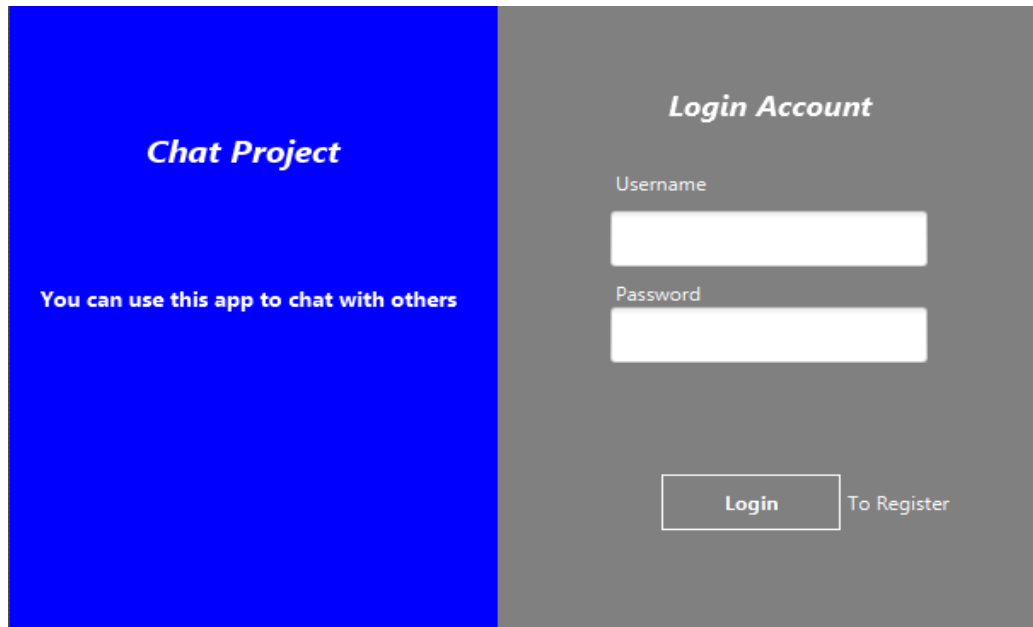
Figure 5 : la partie main

Si la valeur lue Ce code correspond à la classe principale "MainApp" de l'application. Cette classe étend la classe "Application" de JavaFX et est responsable du démarrage de l'application et de l'affichage de la première scène.

- Les importations nécessaires sont effectuées pour les classes et les packages utilisés dans le code.
- La classe "MainApp" étend la classe "Application" de JavaFX.
- La méthode "start()" est remplacée pour définir le point d'entrée de l'application.
- À l'intérieur de la méthode "start()", un objet FXMLLoader est créé pour charger le fichier FXML de la première scène ("Scene1.fxml").
- Une scène est créée en utilisant le contenu chargé à partir du fichier FXML.
- Un objet Stage est créé et la scène est définie comme contenu principal de la fenêtre.
- Le titre de la fenêtre est défini comme "Manager's App".
- Enfin, la fenêtre est affichée en utilisant la méthode "show()" de l'objet Stage.
- La méthode "main()" est utilisée pour démarrer l'application en appelant la méthode statique "launch()" de la classe "Application".

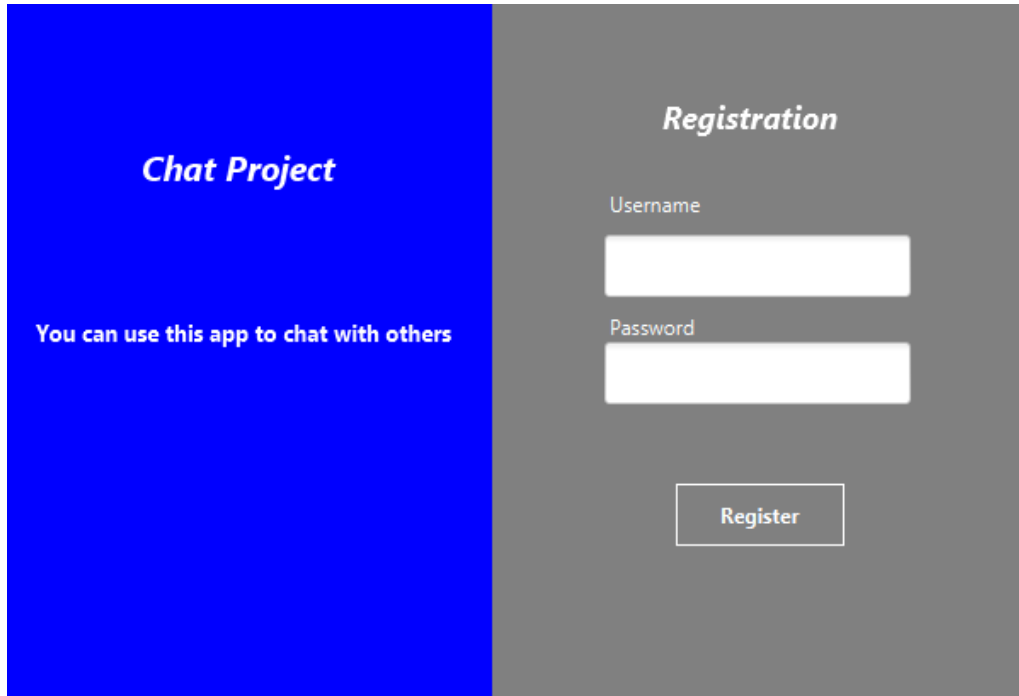
Ce code assure le démarrage de l'application et l'affichage de la première scène ("Scene1.fxml") qui constitue le point de départ de l'application de chat en JavaFX.

1.5: Les interfaces graphiques :



The image shows a JavaFX login interface. It is split into two vertical panels. The left panel has a solid blue background and contains the text "Chat Project" in white, italicized font, and "You can use this app to chat with others" in a smaller white font below it. The right panel has a gray background and is titled "Login Account" in white, italicized font. It contains two white text input fields labeled "Username" and "Password" in gray text. Below the fields are two buttons: "Login" and "To Register", both with white text on a gray background.

Figure 6 : interface graphique de connexion.



The image shows a JavaFX registration interface. It is split into two vertical panels. The left panel has a solid blue background and contains the text "Chat Project" in white, italicized font, and "You can use this app to chat with others" in a smaller white font below it. The right panel has a gray background and is titled "Registration" in white, italicized font. It contains two white text input fields labeled "Username" and "Password" in gray text. Below the fields is a single button labeled "Register" with white text on a gray background.

Figure 7 : interface graphique d'inscription.

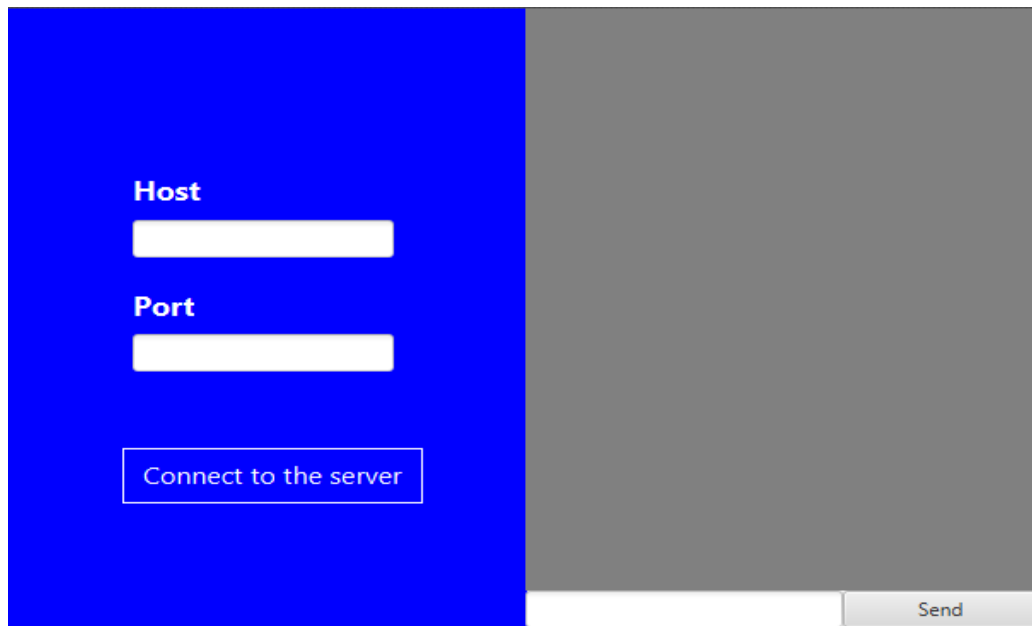


Figure 8 : interface graphique du chat

Ces trois interfaces sont manipulées à l'aide de SceneBuilder.

Chapitre III : Bilan de projet et simulation

On passe à la simulation qui va nous montrer comment fonctionne l'application

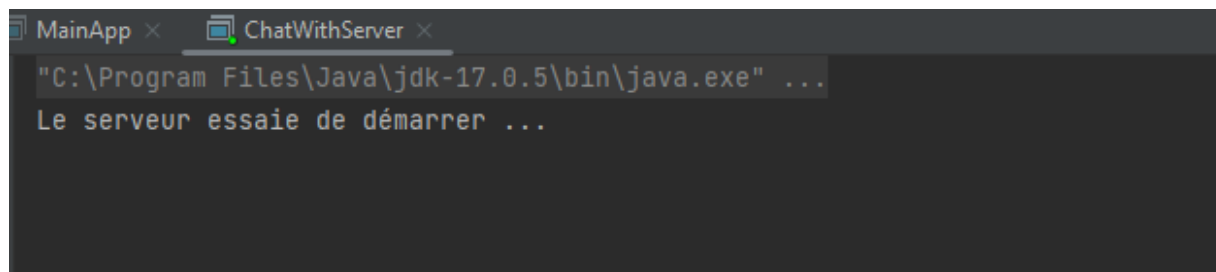


Figure 9 : démarrage du serveur

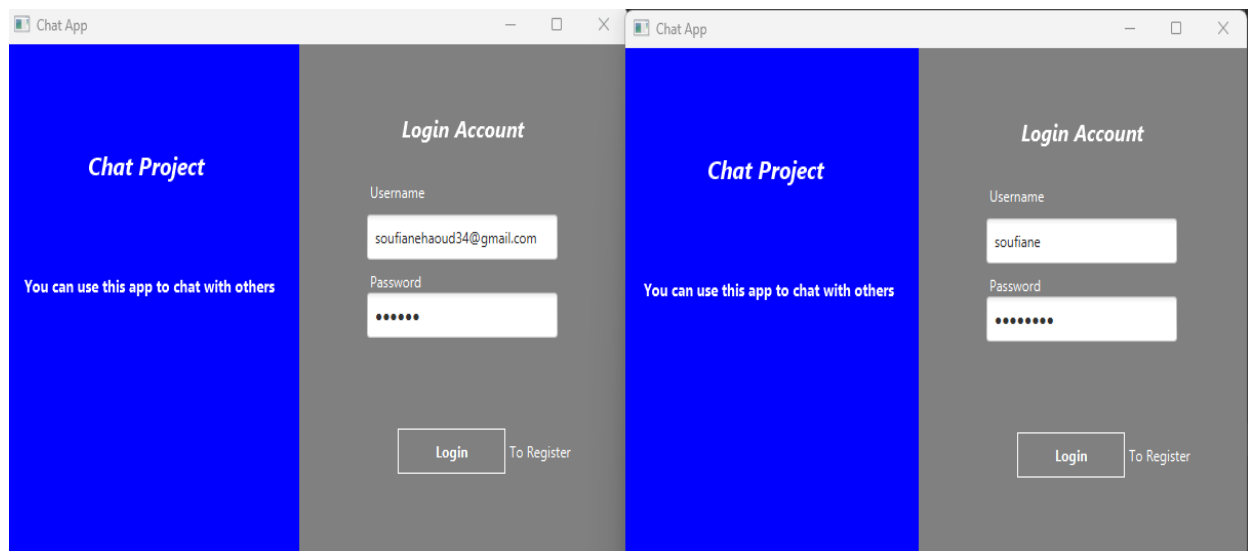


Figure 10 : connexion de deux utilisateurs

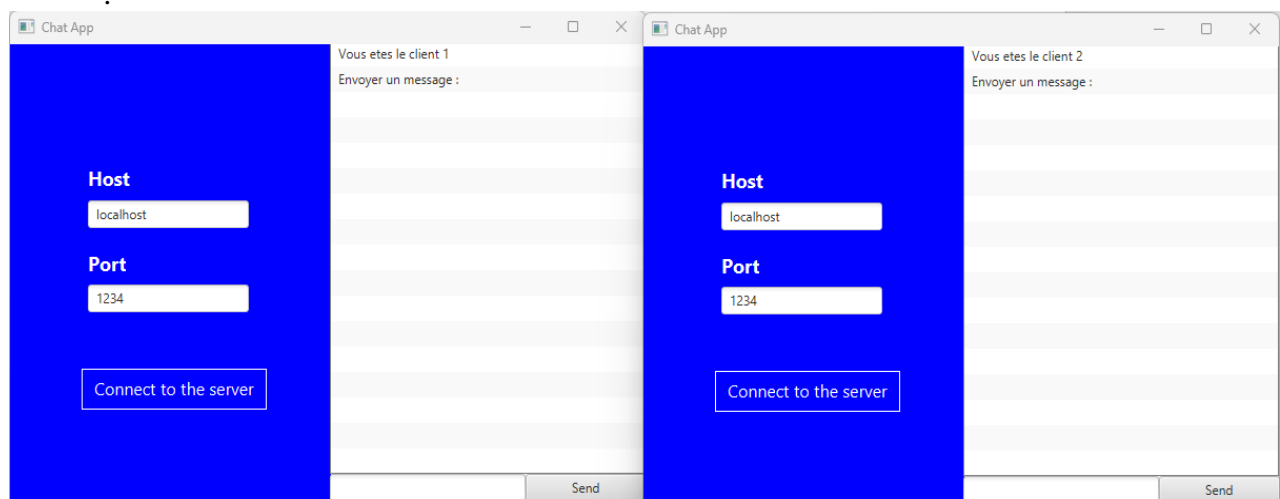


Figure 11 : connexion au serveur.

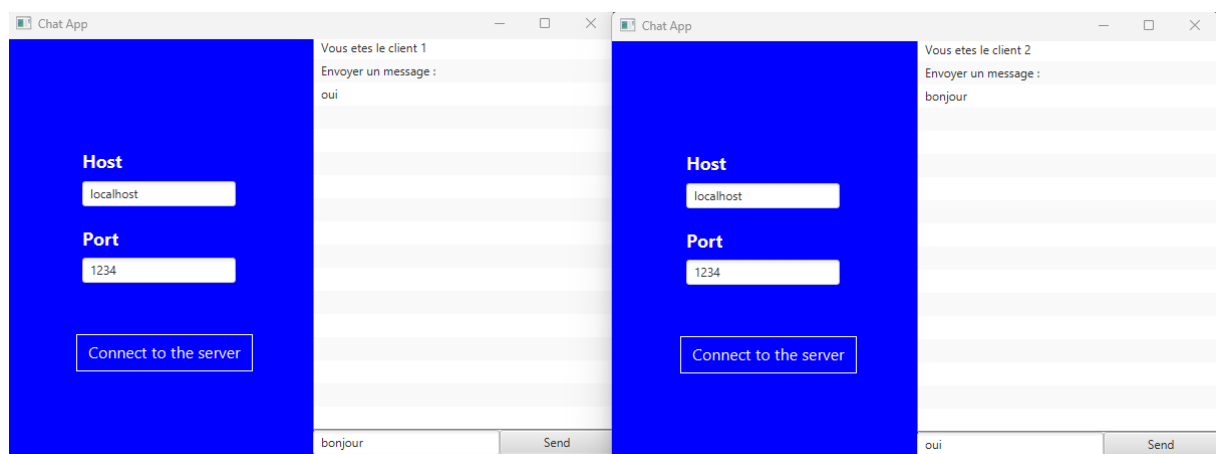


Figure 12 : échange de messages.

On ajoute un autre utilisateur :

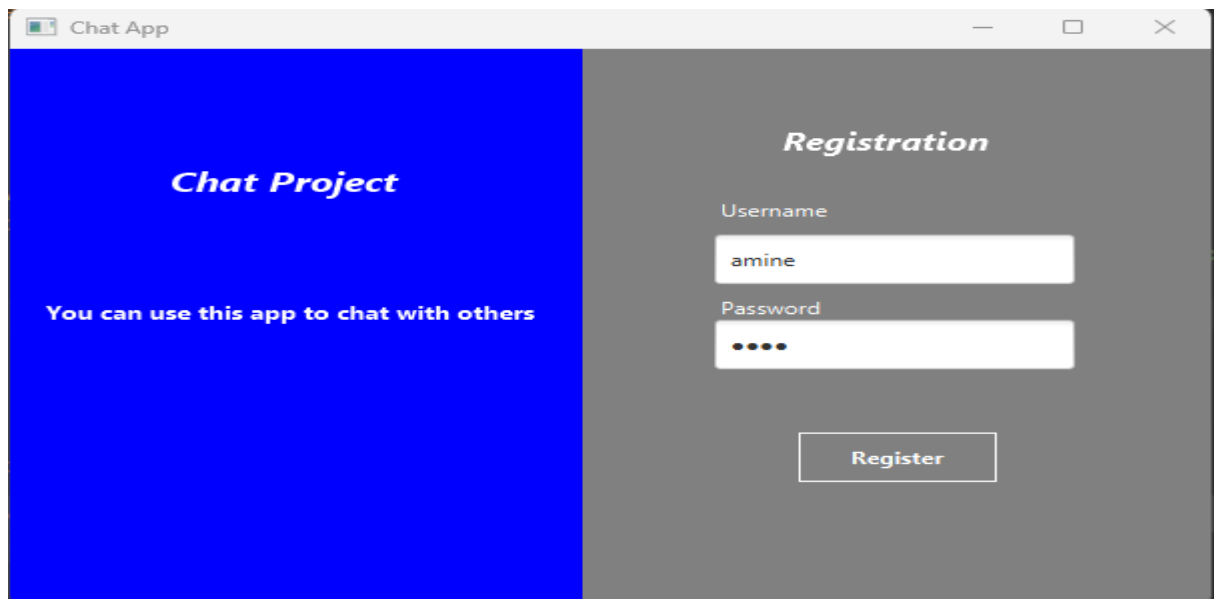


Figure 13 : nouveau utilisateur

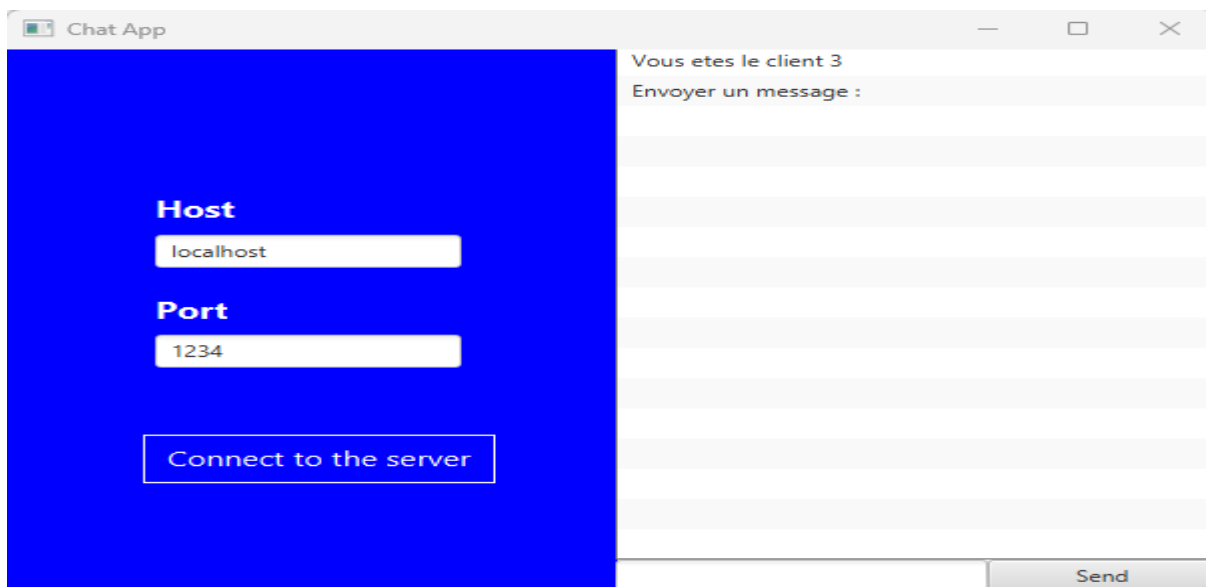


Figure 14 : connexion au serveur pour l'utilisateur ajouté

On envoie un message à un utilisateur spécifique :

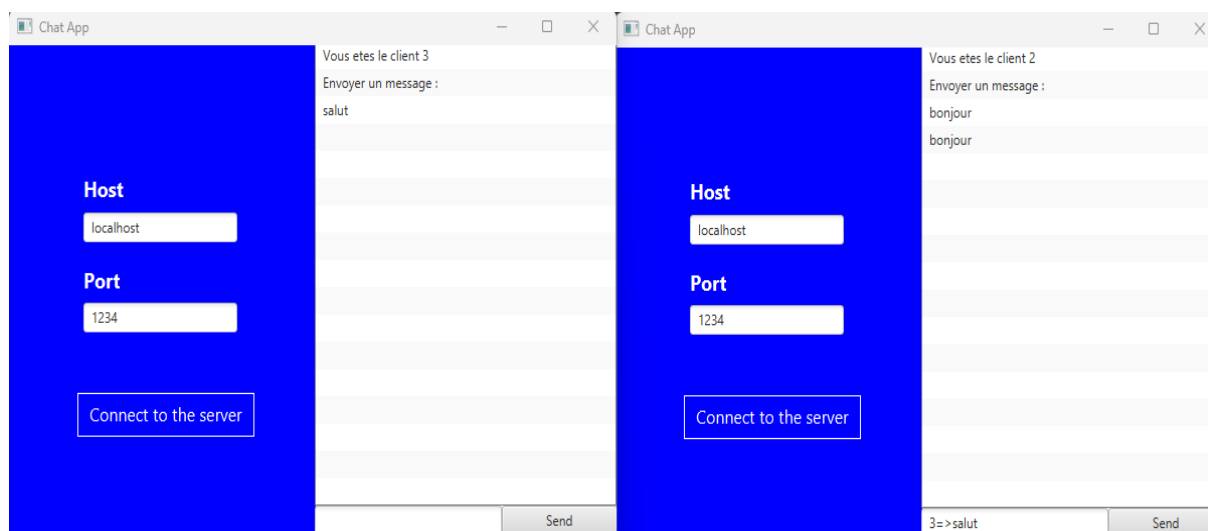


Figure 15 : échange de message avec un utilisateur spécifique

Conclusions et perspectives

Dans ce projet, nous avons développé une application de chat avancée en Java en utilisant JavaFX pour l'interface utilisateur et les fonctionnalités de communication en réseau. L'application permet aux utilisateurs de se connecter, de s'inscrire, d'envoyer des messages publics et privés, et de visualiser les messages dans une interface conviviale.

Nous avons implémenté les fonctionnalités essentielles telles que l'authentification des utilisateurs, la gestion des connexions, l'envoi et la réception des messages, et la gestion des erreurs. L'application est conçue pour être utilisée avec un serveur central qui permet aux clients de communiquer entre eux.

Bien que l'application de chat développée soit fonctionnelle, il existe encore des possibilités d'amélioration et de développement de nouvelles fonctionnalités. Voici quelques perspectives possibles pour le projet :

1. Amélioration de l'interface utilisateur : Ajouter des fonctionnalités d'amélioration de l'expérience utilisateur telles que la possibilité d'envoyer des émoticônes, d'envoyer des fichiers, de personnaliser l'apparence, etc.
2. Implémentation d'une fonctionnalité de chat de groupe : Permettre aux utilisateurs de créer des groupes de discussion et d'inviter d'autres utilisateurs à rejoindre le groupe.
3. Sécurité et confidentialité : Renforcer la sécurité de l'application en implémentant des mécanismes de chiffrement et d'authentification plus robustes pour protéger les communications et les données des utilisateurs.
4. Historique des messages : Ajouter la possibilité de stocker et d'afficher l'historique des messages pour permettre aux utilisateurs de voir les conversations précédentes.
5. Fonctionnalités de notification : Implémenter des notifications en temps réel pour informer les utilisateurs des nouveaux messages ou des activités dans l'application.
6. Déploiement sur le cloud : Adapter l'application pour qu'elle puisse être déployée sur une infrastructure de cloud computing, ce qui permettrait une évolutivité et une disponibilité accrues.

École Marocaine des Sciences De l'Ingénieur de Casablanca

Nom : Haoud
Prénom : Soufiane

Titre du mini-

Projet :

« Application de chat »

Résumé :

Ce rapport montre la réalisation d'une application de chat performante, en exploitant les fonctionnalités avancées offertes par Java. Pour cela, nous avons utilisé diverses bibliothèques et frameworks Java populaires, tels que JavaFX et Socket,

Mots clés : JavaFx , serveur , FXML ,socket

