Project Report

GROUP 5 – MESSAGING APPLICATION

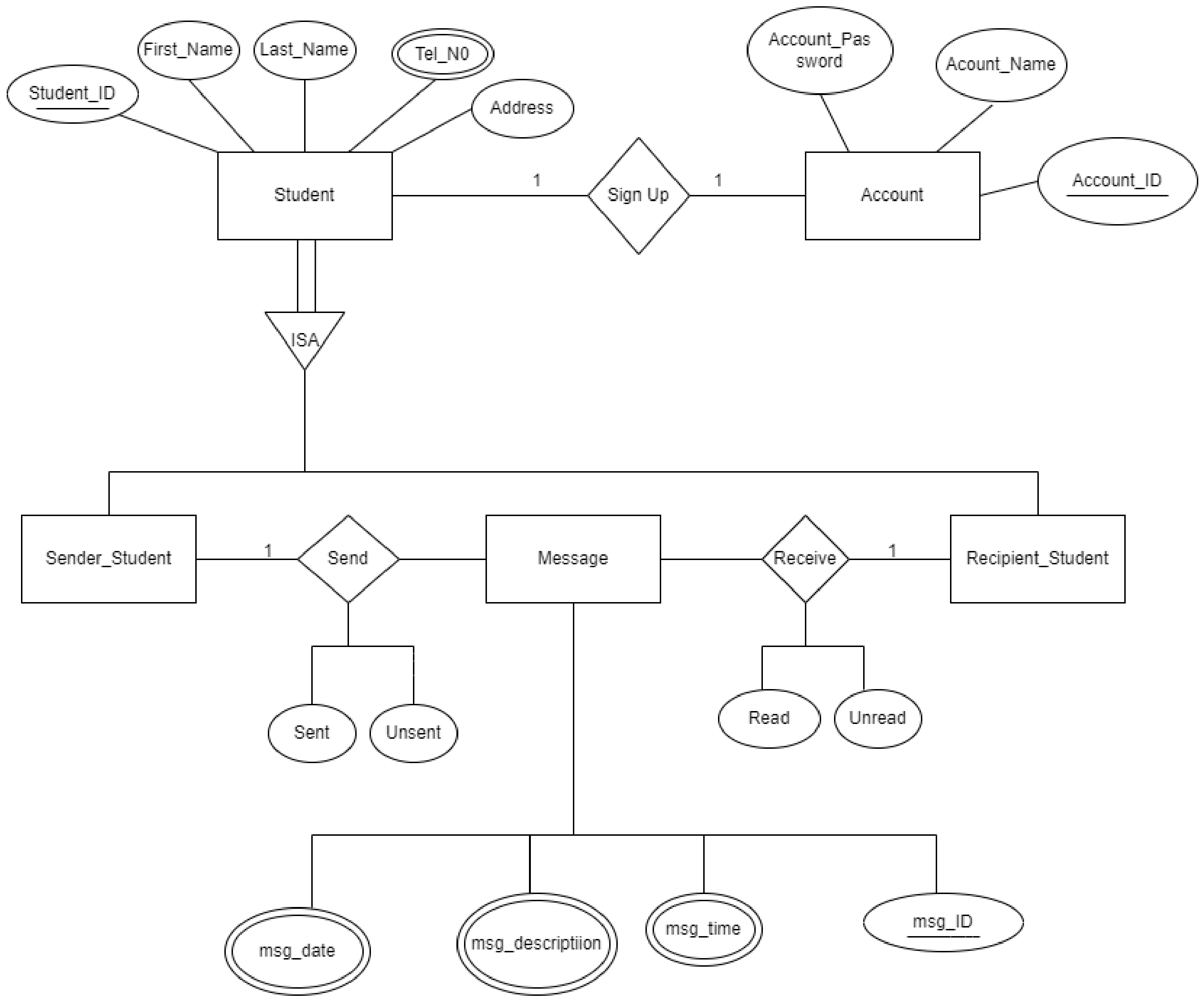
1- REQUIREMENT and ANALYSIS OF REQUIREMENTS

This phase was also carried out by the system architects.

- **Java API: Intellij**
  This is the IDE used in the development of our messaging application.
- **Scene builder**
  Allows you to easily layout JavaFX UI controls, charts, shapes and containers, so as to quickly prototype user interfaces.
- **Mysql**
  This is an open source relational database management system. Its stores the data of the system's user and allows the developer to manipulate this data.
- **MS Visio**
  Used to draw the ER diagram, process flow diagram.

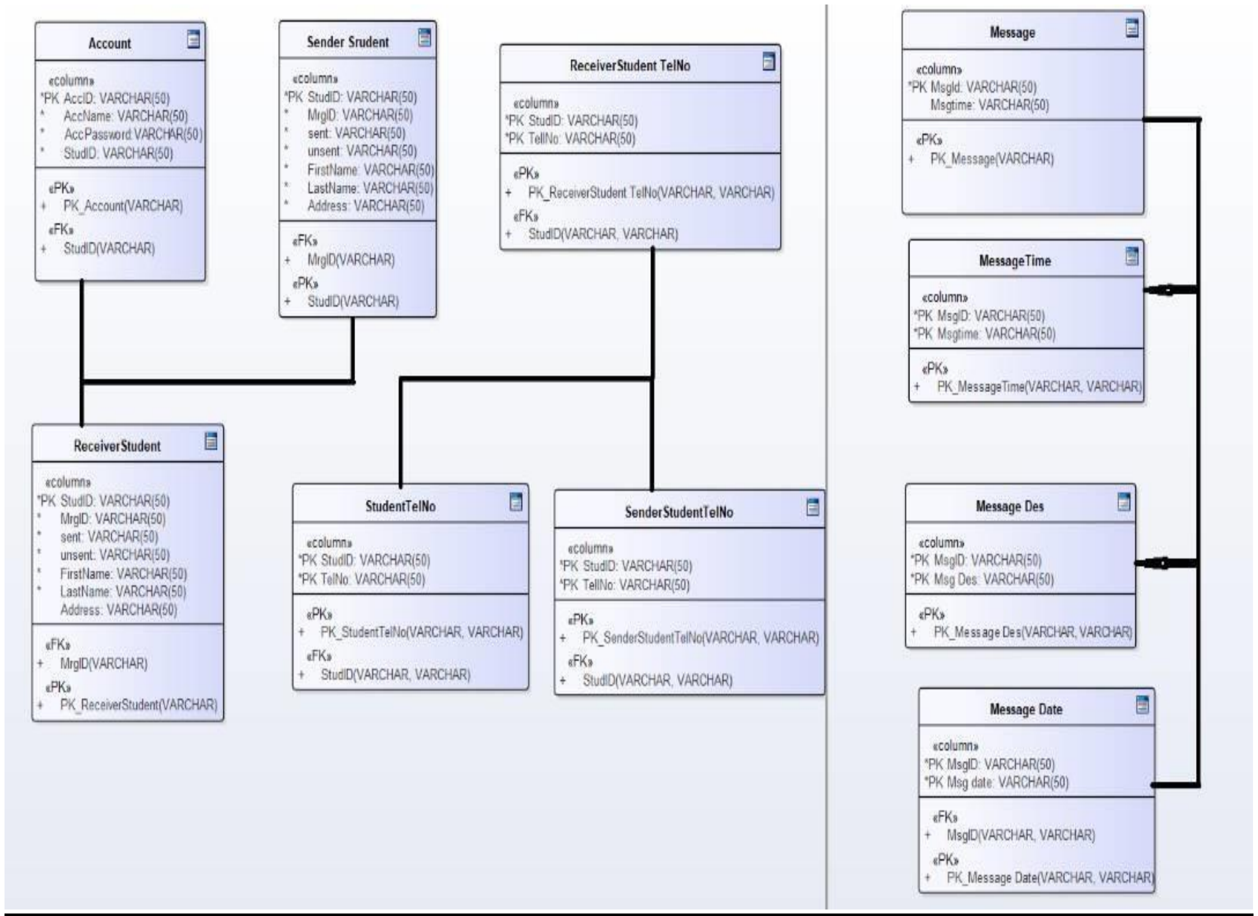
2- PLANNING

- **<u>Entity Relational Diagram (ER diagram) for the Messaging Application</u>**



<u>Fig 1: ER Diagram</u>

- **<u>Relational Database Model of the Messaging Application</u>**



<u>Fig 2: Relational Database Model</u>

- **Process Flow Diagram of the Messaging Application**

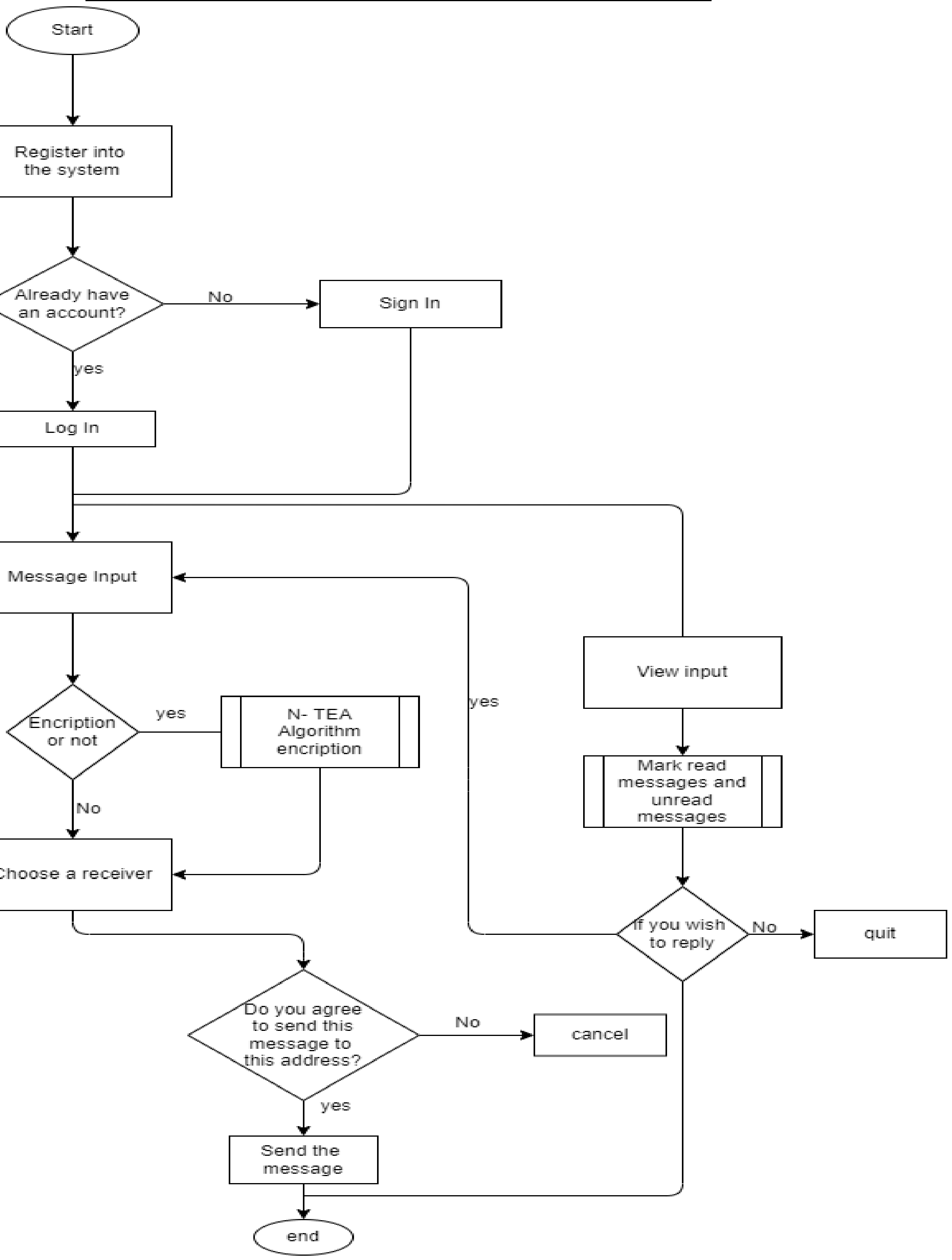
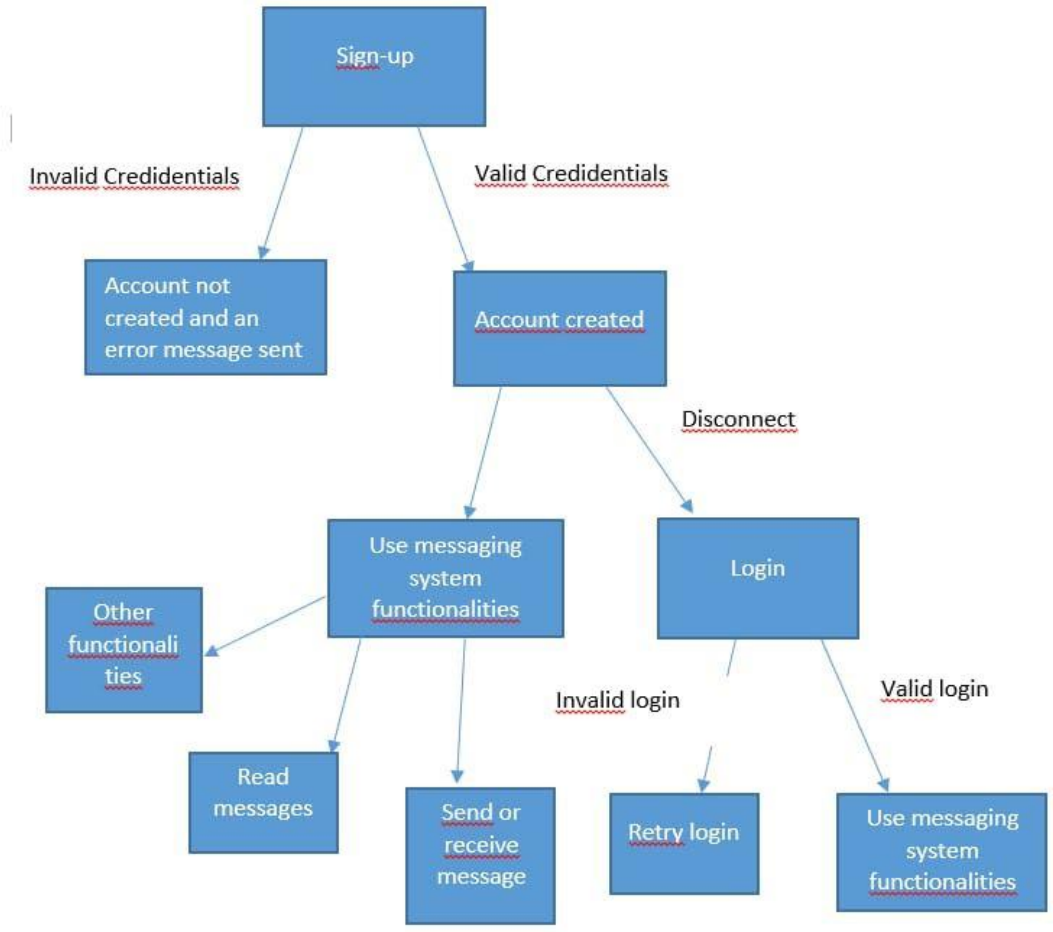


Fig 3: Process flow diagram

- **Decision tree for messaging application**



3- DESIGN

- **Student Login Page:** Here students who already have an account in the system are allowed enter into the system after a verification check, using their Username and password. The information entered is then compared with that found in the database, if it correctly matches that in the database then the student is granted access, otherwise the student won’t be granted access till the proper credentials are entered.

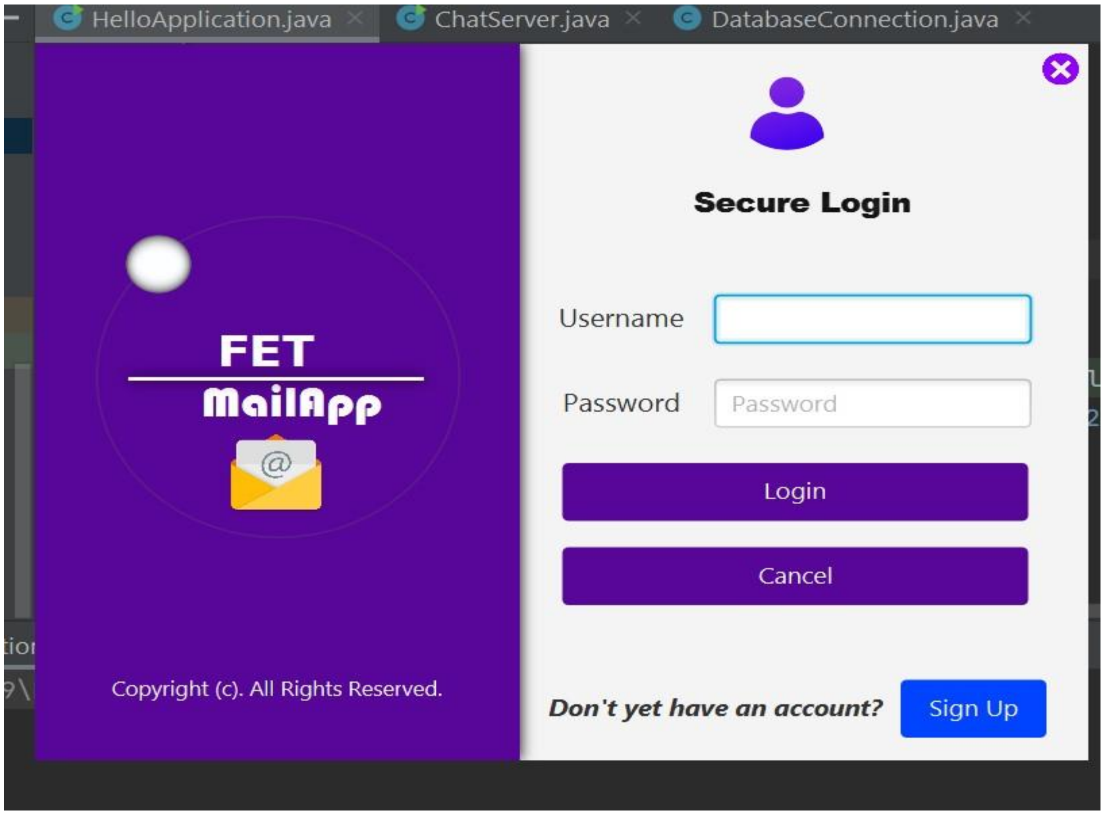
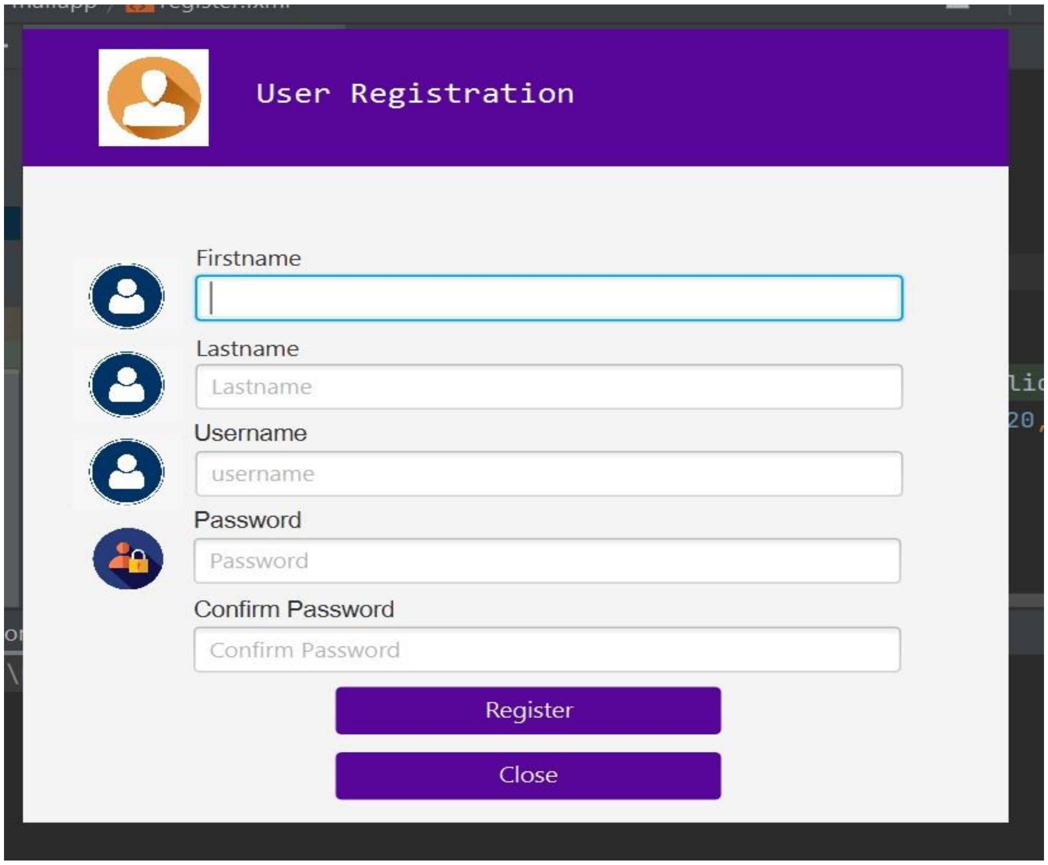


Fig 4: Login page of the messaging application

- **Student Registration Page (Sign up):** Here, students visiting the application for the first time do not have an account within the system. Hence will have to sign so as to have their information registered in the system's database. Once student has successfully registered, he/she can now login into the system using the valid credentials.


User Registration

Firstname

Lastname

Lastname

Username

username

Password

Password

Confirm Password

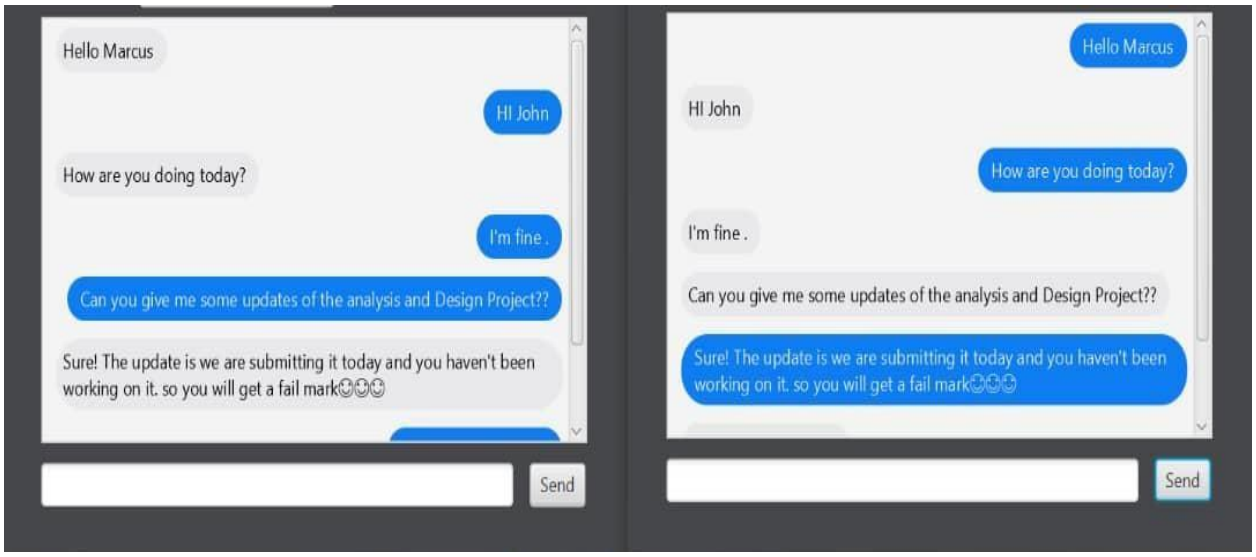
Confirm Password

Register

Close

Fig 5: Sign up page of the messaging application

- **Student Inbox:** Here, we can find the messages sent and received between two students who are logged in to the system. The sender's message is in blue while, the receivers message is in grey.


Hello Marcus

HI John

How are you doing today?

I'm fine .

Can you give me some updates of the analysis and Design Project??

Sure! The update is we are submitting it today and you haven't been working on it. so you will get a fail mark😊😊😊

Send

Hello Marcus

HI John

How are you doing today?

I'm fine .

Can you give me some updates of the analysis and Design Project??

Sure! The update is we are submitting it today and you haven't been working on it. so you will get a fail mark😊😊😊

Send

Fig 6: Student inbox of the messaging application

- **Student Dashboard:**

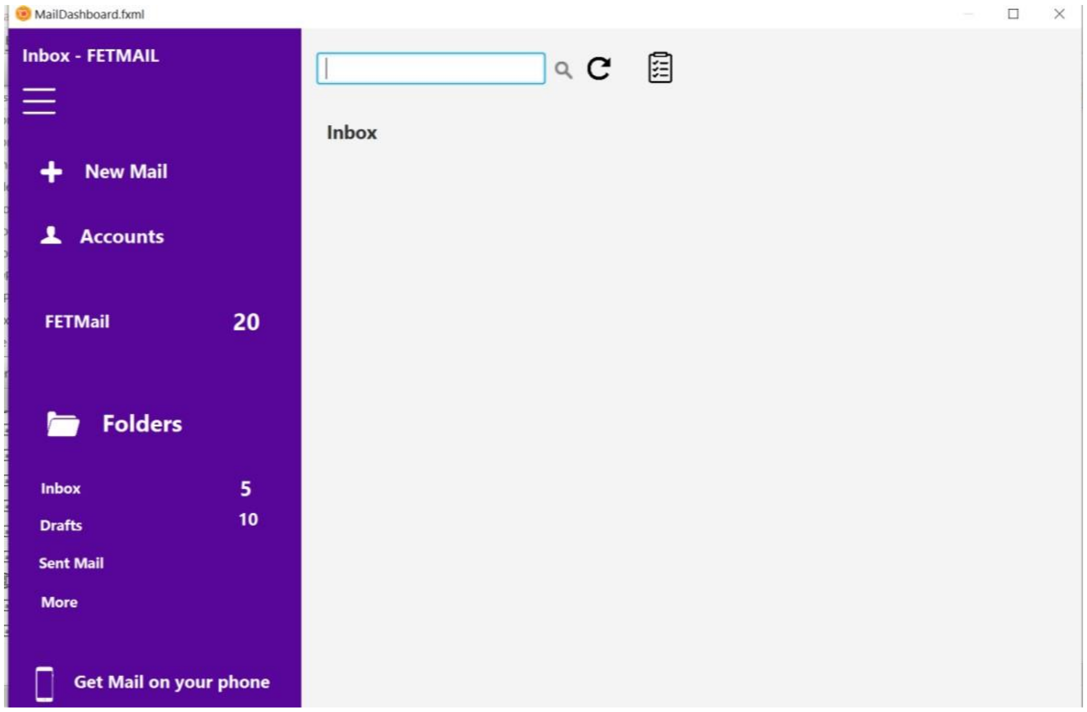
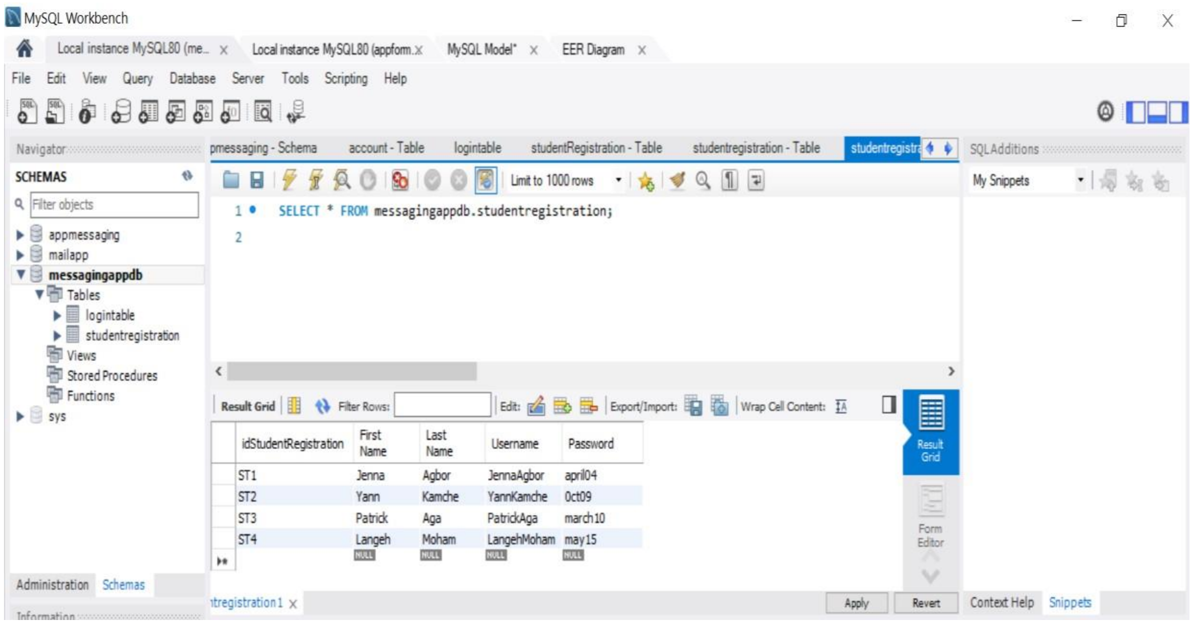


Fig 7: Dashboard of messaging application

4- IMPLENETAION AND CODING

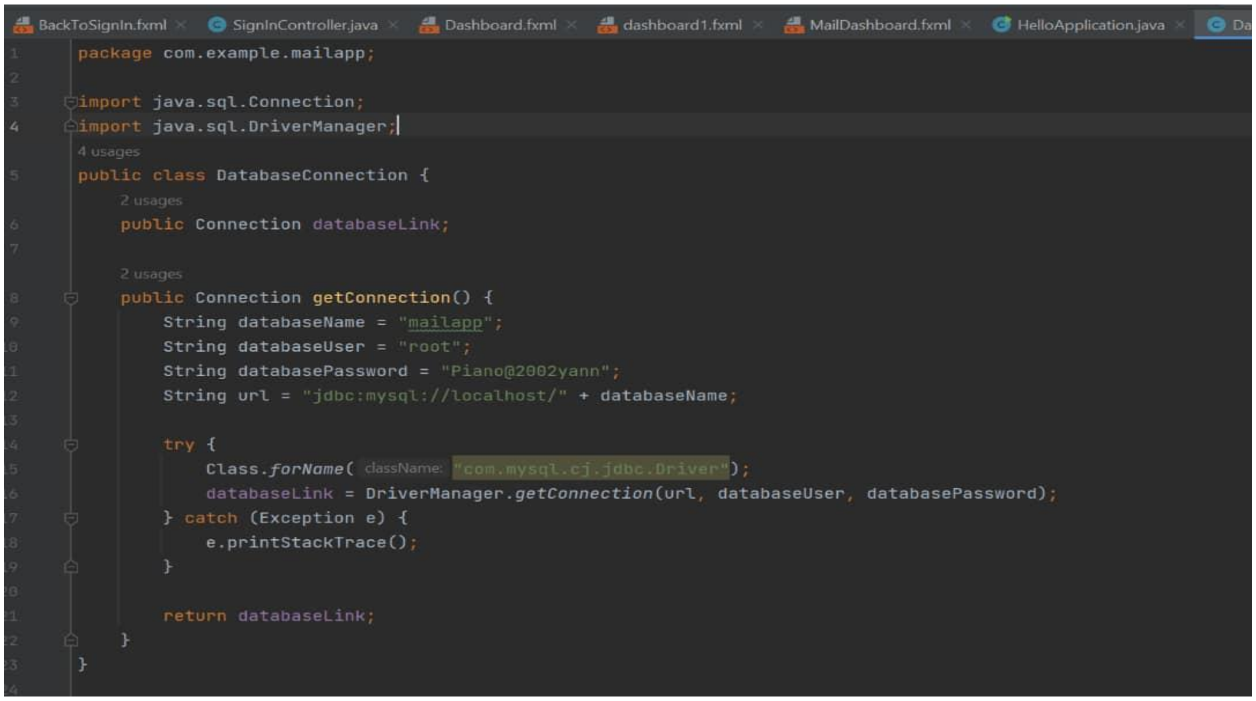
- **Student Database:** In the figure below, we have the system's database which holds the information of students who have accounts in the system. The data found in this database is compared to that which the student inserts during log in, if the data is the same then the student is granted entry into his/her account.



```sql
SELECT * FROM messagingappdb.studentregistration;
```

idStudentRegistration	First Name	Last Name	Username	Password
ST1	Jenna	Agbor	JennaAgbor	april04
ST2	Yann	Kamche	YannKamche	Oct09
ST3	Patrick	Aga	PatrickAga	march10
ST4	Langeh	Moham	LangehMoham	may15
	NULL	NULL	NULL	NULL

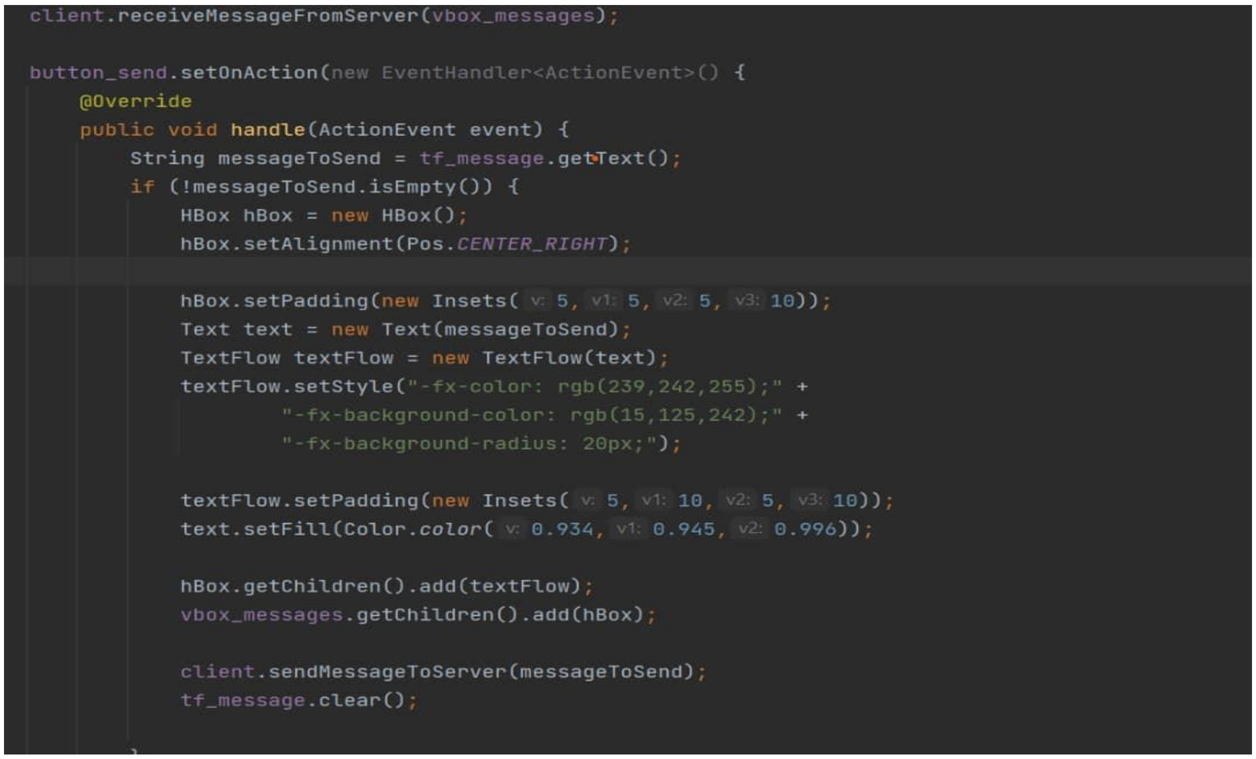
Fig 7: Messaging application database



```java
package com.example.mailapp;

import java.sql.Connection;
import java.sql.DriverManager;
4 usages
public class DatabaseConnection {
    2 usages
    public Connection databaseLink;

    2 usages
    public Connection getConnection() {
        String databaseName = "mailapp";
        String databaseUser = "root";
        String databasePassword = "Piano@2002yann";
        String url = "jdbc:mysql://localhost/" + databaseName;

        try {
            Class.forName( className: "com.mysql.cj.jdbc.Driver");
            databaseLink = DriverManager.getConnection(url, databaseUser, databasePassword);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return databaseLink;
    }
}
```

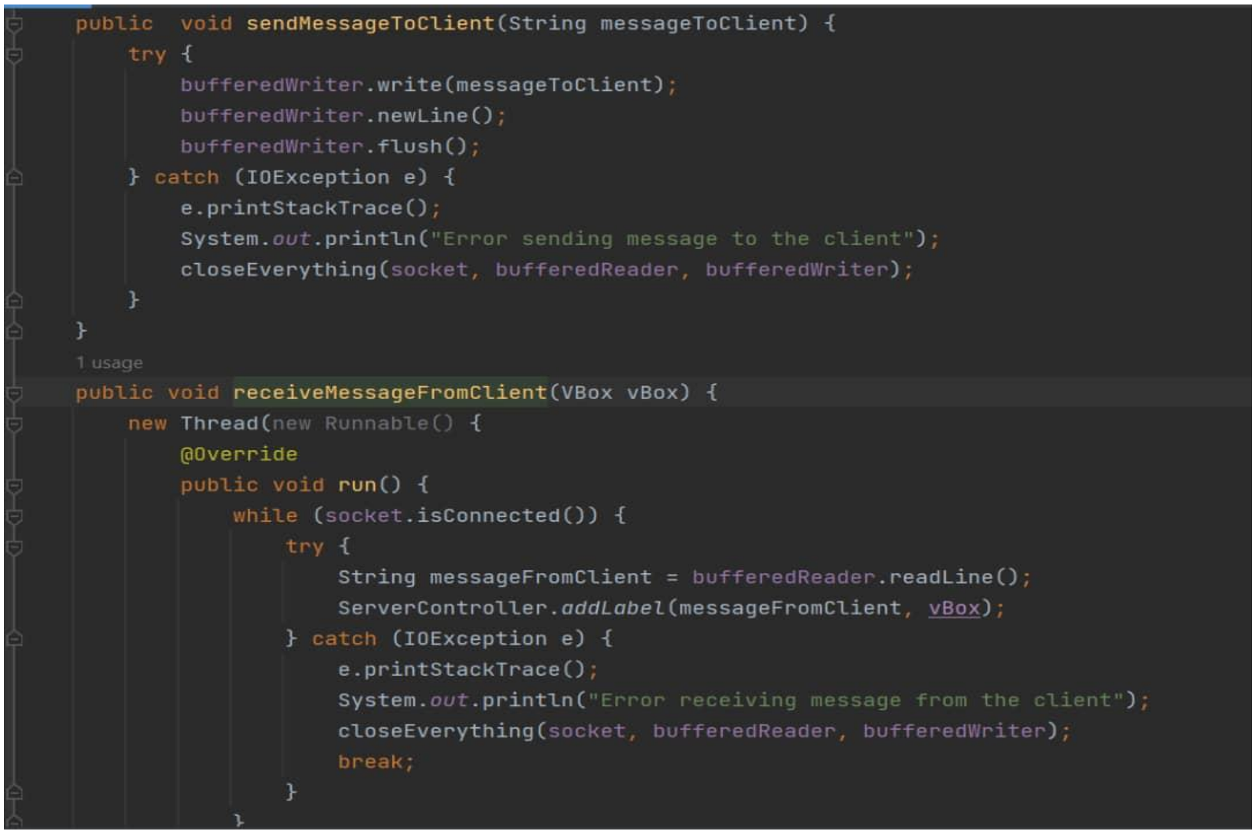
Fig 6: Database Connection code

- **Sending and Receiving message:**



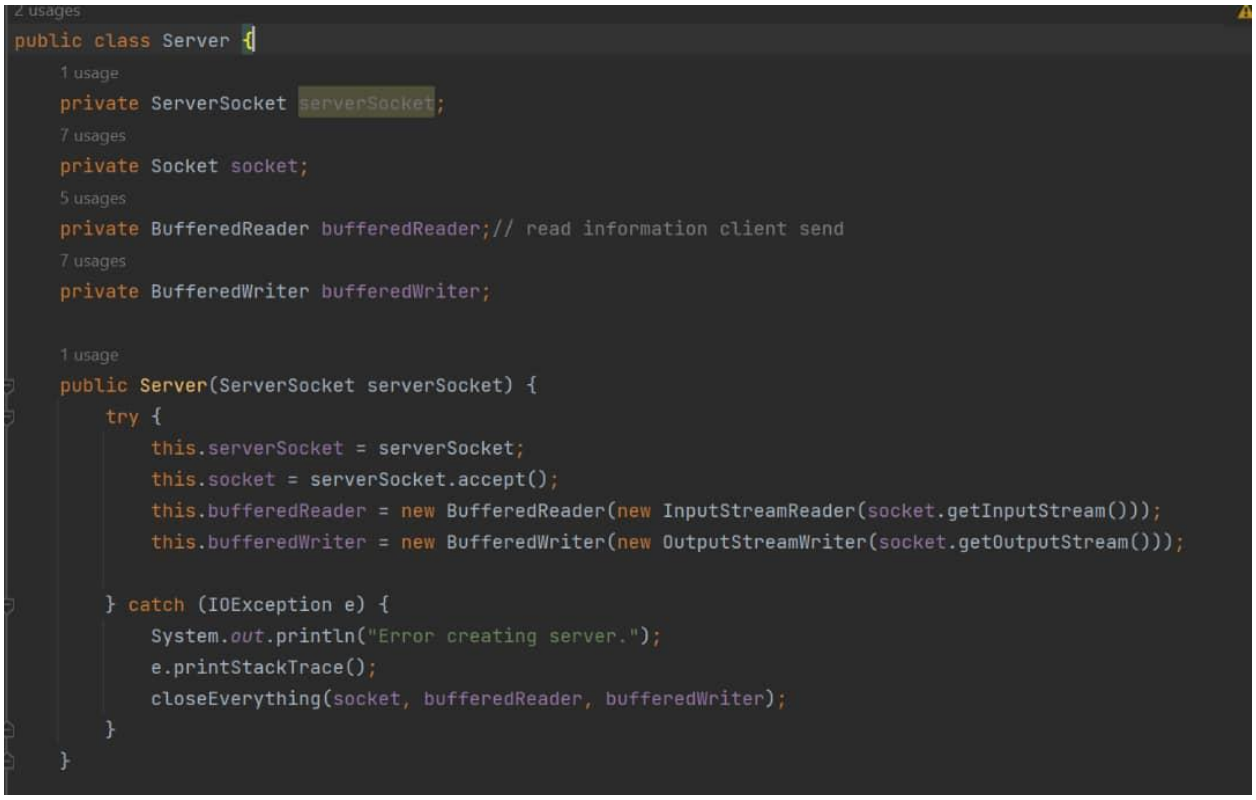
```
client.receiveMessageFromServer(vbox_messages);

button_send.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        String messageToSend = tf_message.getText();
        if (!messageToSend.isEmpty()) {
            HBox hBox = new HBox();
            hBox.setAlignment(Pos.CENTER_RIGHT);

            hBox.setPadding(new Insets( v: 5, v1: 5, v2: 5, v3: 10));
            Text text = new Text(messageToSend);
            TextFlow textFlow = new TextFlow(text);
            textFlow.setStyle("-fx-color: rgb(239,242,255);" +
                    "-fx-background-color: rgb(15,125,242);" +
                    "-fx-background-radius: 20px;");

            textFlow.setPadding(new Insets( v: 5, v1: 10, v2: 5, v3: 10));
            text.setFill(Color.color( v: 0.934, v1: 0.945, v2: 0.996));

            hBox.getChildren().add(textFlow);
            vbox_messages.getChildren().add(hBox);

            client.sendMessageToServer(messageToSend);
            tf_message.clear();
```

Fig 7: Code for sending mails



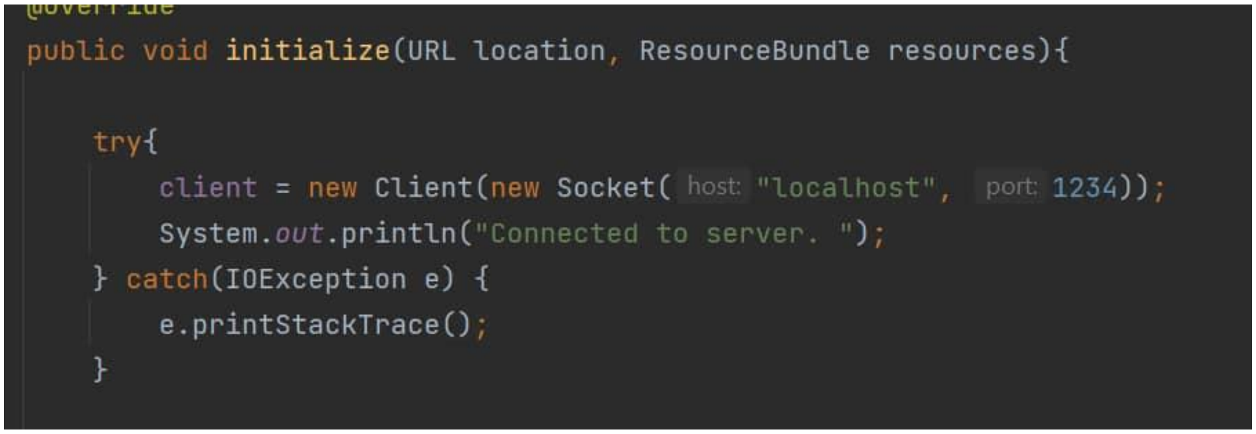
```
    public  void sendMessageToClient(String messageToClient) {
        try {
            bufferedWriter.write(messageToClient);
            bufferedWriter.newLine();
            bufferedWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("Error sending message to the client");
            closeEverything(socket, bufferedReader, bufferedWriter);
        }
    }
    1 usage
    public void receiveMessageFromClient(VBox vBox) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                while (socket.isConnected()) {
                    try {
                        String messageFromClient = bufferedReader.readLine();
                        ServerController.addLabel(messageFromClient, vBox);
                    } catch (IOException e) {
                        e.printStackTrace();
                        System.out.println("Error receiving message from the client");
                        closeEverything(socket, bufferedReader, bufferedWriter);
                        break;
                    }
```

Fig 8: Code for receiving mails



```
2 usages
public class Server {
    1 usage
    private ServerSocket serverSocket;
    7 usages
    private Socket socket;
    5 usages
    private BufferedReader bufferedReader;// read information client send
    7 usages
    private BufferedWriter bufferedWriter;

    1 usage
    public Server(ServerSocket serverSocket) {
        try {
            this.serverSocket = serverSocket;
            this.socket = serverSocket.accept();
            this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            this.bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

        } catch (IOException e) {
            System.out.println("Error creating server.");
            e.printStackTrace();
            closeEverything(socket, bufferedReader, bufferedWriter);
        }
    }
```

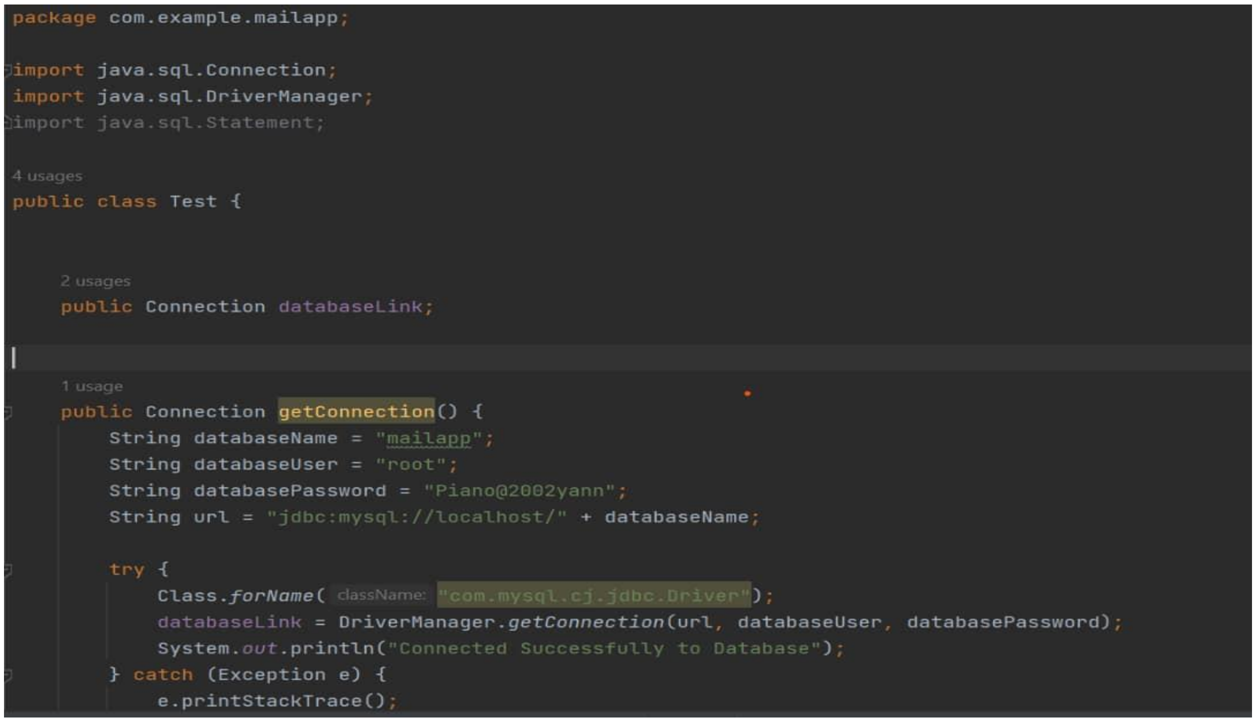
Fig 9: Code for creating the server



```
@Override
public void initialize(URL location, ResourceBundle resources){

    try{
        client = new Client(new Socket( host: "localhost",  port: 1234));
        System.out.println("Connected to server. ");
    } catch(IOException e) {
        e.printStackTrace();
    }
```

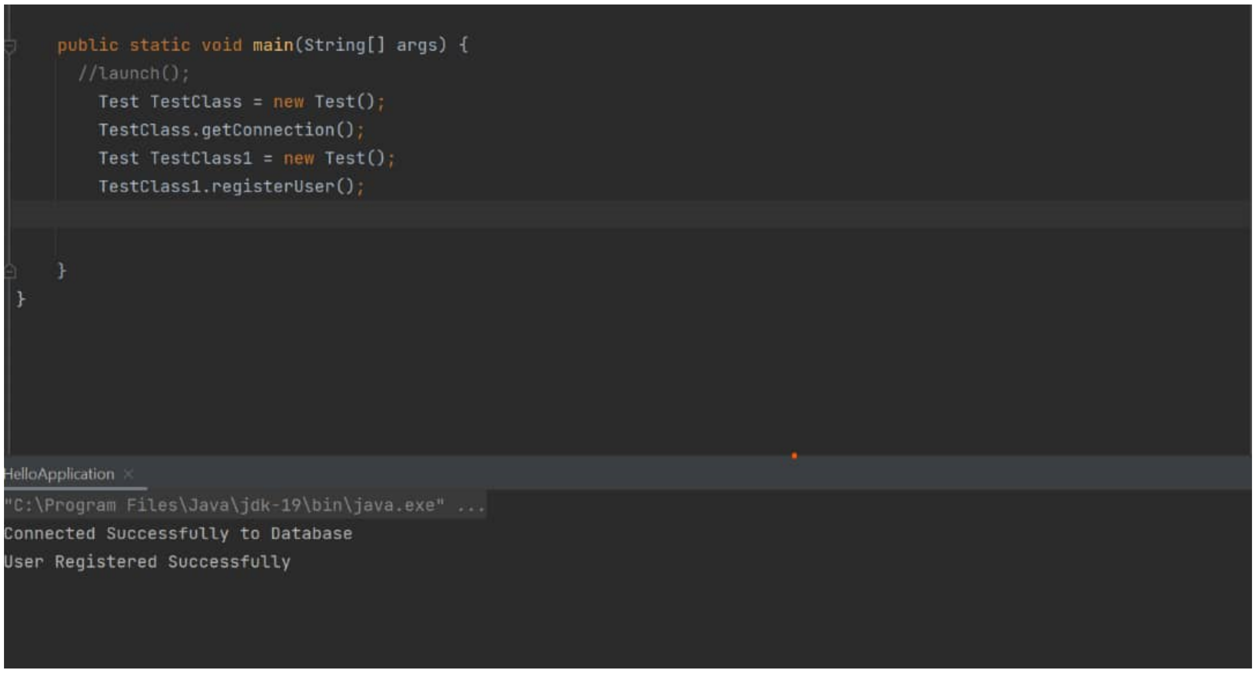
Fig 10: Port implementation

5- **Testing:** Here the testers wrote codes to find out whether the software is working according to the requirements.



```
package com.example.mailapp;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

4 usages
public class Test {


    2 usages
    public Connection databaseLink;


    1 usage
    public Connection getConnection() {
        String databaseName = "mailapp";
        String databaseUser = "root";
        String databasePassword = "Piano@2002yann";
        String url = "jdbc:mysql://localhost/" + databaseName;

        try {
            Class.forName( className: "com.mysql.cj.jdbc.Driver");
            databaseLink = DriverManager.getConnection(url, databaseUser, databasePassword);
            System.out.println("Connected Successfully to Database");
        } catch (Exception e) {
            e.printStackTrace();
```

Fig 11: Test code



```
    public static void main(String[] args) {
      //launch();
        Test TestClass = new Test();
        TestClass.getConnection();
        Test TestClass1 = new Test();
        TestClass1.registerUser();


    }
}
```

HelloApplication

```
"C:\Program Files\Java\jdk-19\bin\java.exe" ...
Connected Successfully to Database
User Registered Successfully
```

Fig 12: Test implementation

7- CONCLUSION

In conclusion, the email client software hereby mentioned is fully functional with the following capabilities:

- ✓ User registration
- ✓ User login
- ✓ Sending mail
- ✓ Receiving mail

With the incremental model being used, essential modules (functionalities) where implemented first and other functionalities will be implemented with progress of the project.