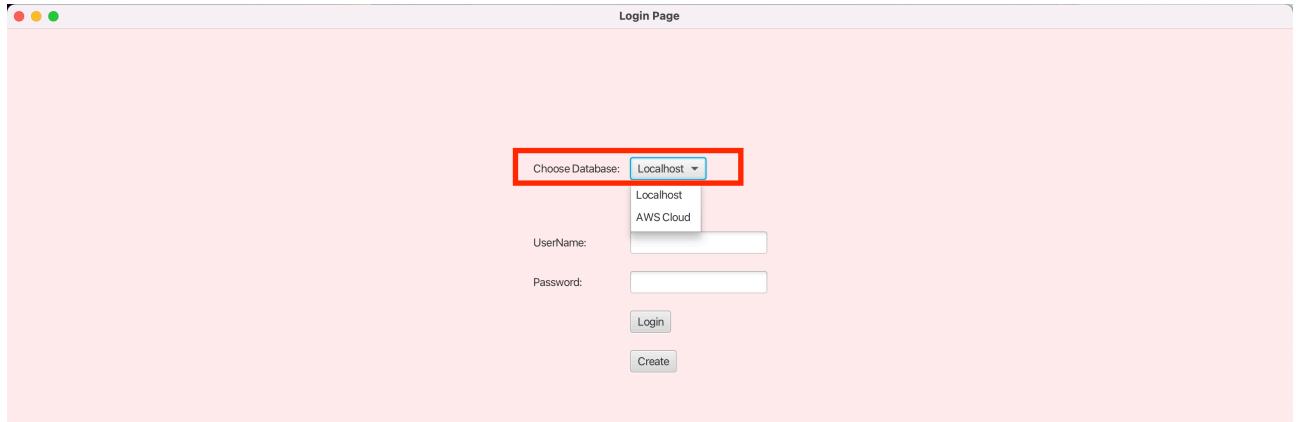


GitLink(<https://gitlab.ecs.vuw.ac.nz/wangming3/encryptdecryptapp>)

1. Login Page

1.1 Choose Database

User can choose Localhost Database or AWS Cloud Database to connect.



```
Label chooseDataLabel = new Label("Choose Database: ");
MenuItem localData = new MenuItem("Localhost");
MenuItem cloudData = new MenuItem("AWS Cloud");
dataMenuButton.getItems().addAll(localData,cloudData);
dataMenuButton.setText(chosenData);

localData.setOnAction(e ->{
    chosenData = localData.getText();
    dataMenuButton.setText(chosenData);
    JDBC_URL = "jdbc:mysql://localhost:3306/security";
    USERNAME = "root";
    PASSWORD = "";
});

cloudData.setOnAction(e ->{
    chosenData = cloudData.getText();
    dataMenuButton.setText(chosenData);
    JDBC_URL = "jdbc:mysql://myfirstdatabase.cf8sld5urrx1.ap-southeast-2.rds.amazonaws.com:3306/security";
    USERNAME = "admin";
    PASSWORD = "willawilla";
});
```

localhost Database

The screenshot shows the 'localhost / localhost / security' page in phpMyAdmin. The left sidebar shows databases like 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', and 'security'. The 'security' database is selected and has tables 'New', 'login', 'message', and 'settings'. The 'login' table is currently selected, showing 25 rows of data. The columns are 'id', 'userName', and 'password'.

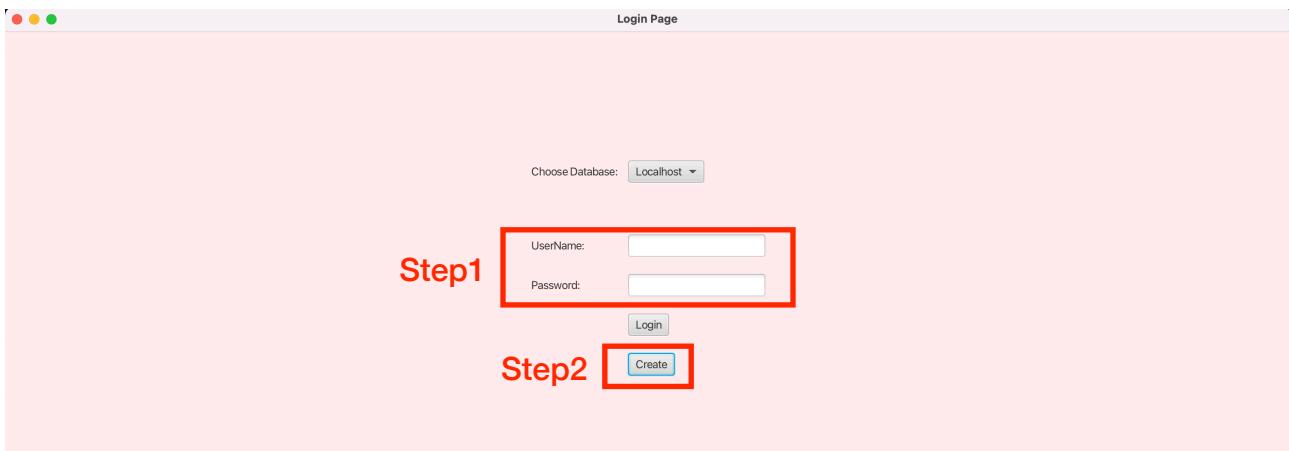
AWS Cloud Database

The screenshot shows the AWS RDS MySQL interface for the 'AWS-myfirstdatabase' database. The 'security' schema is selected, and the 'login' table is shown in the result grid. The table has columns 'id', 'userName', and 'password'. The result grid shows 5 rows of data. The 'Action Output' section at the bottom shows the SQL queries executed:

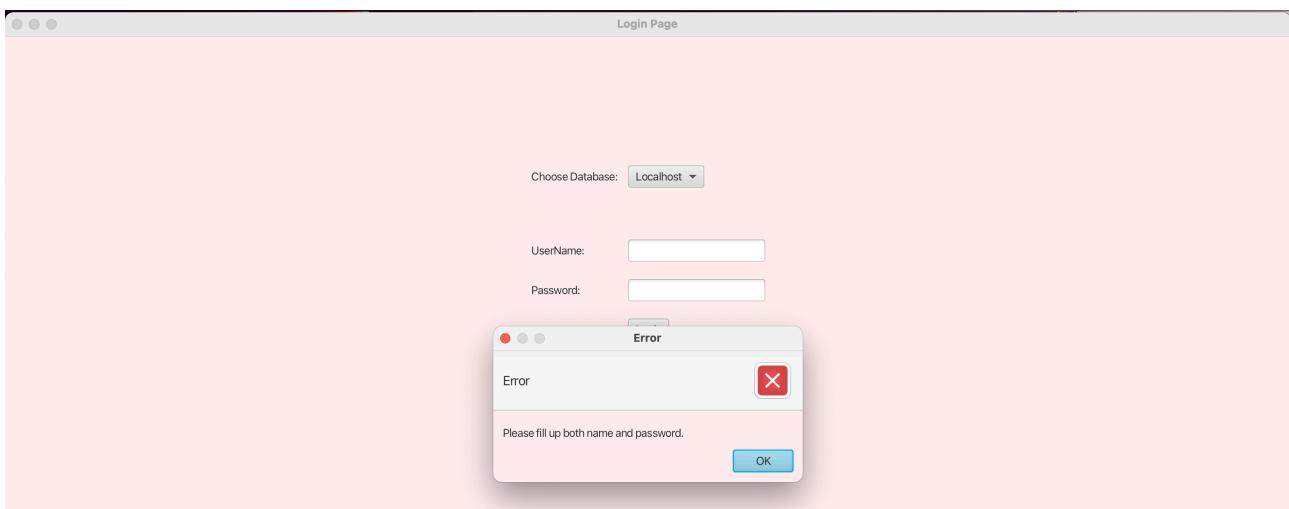
Time	Action	Response
13:02:23	SELECT * FROM security.login LIMIT 0, 1000	0 row(s) returned
13:02:41	SELECT * FROM security.settings LIMIT 0, 1000	3 row(s) returned
13:03:58	SELECT * FROM security.message LIMIT 0, 1000	3 row(s) returned
10:36:19	SELECT * FROM security.login LIMIT 0, 1000	5 row(s) returned

1.2 Create Account

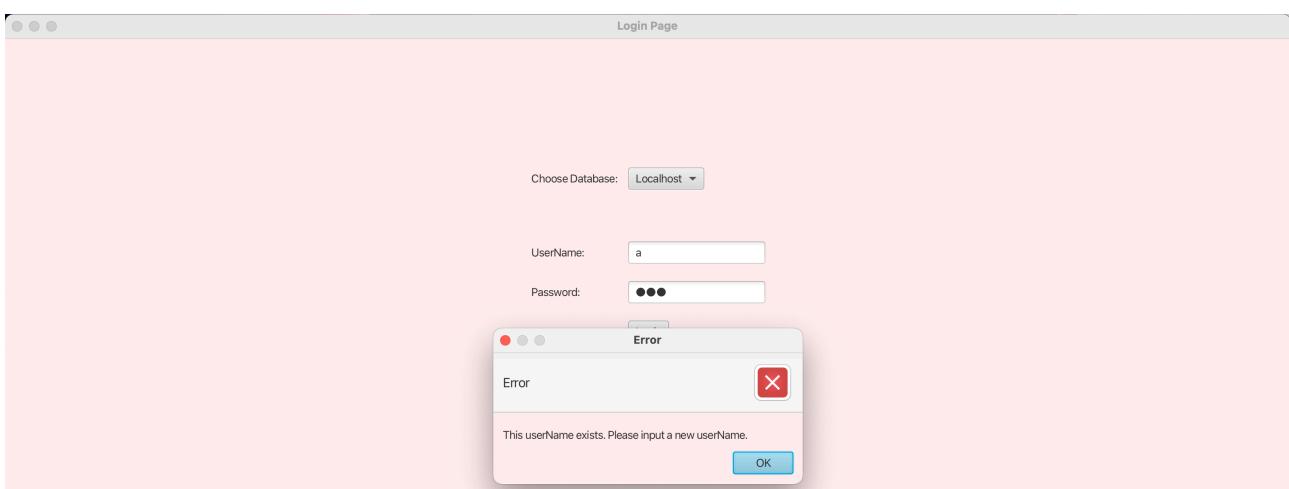
User can input UserName and Password, then click “Create” button to create a new account.



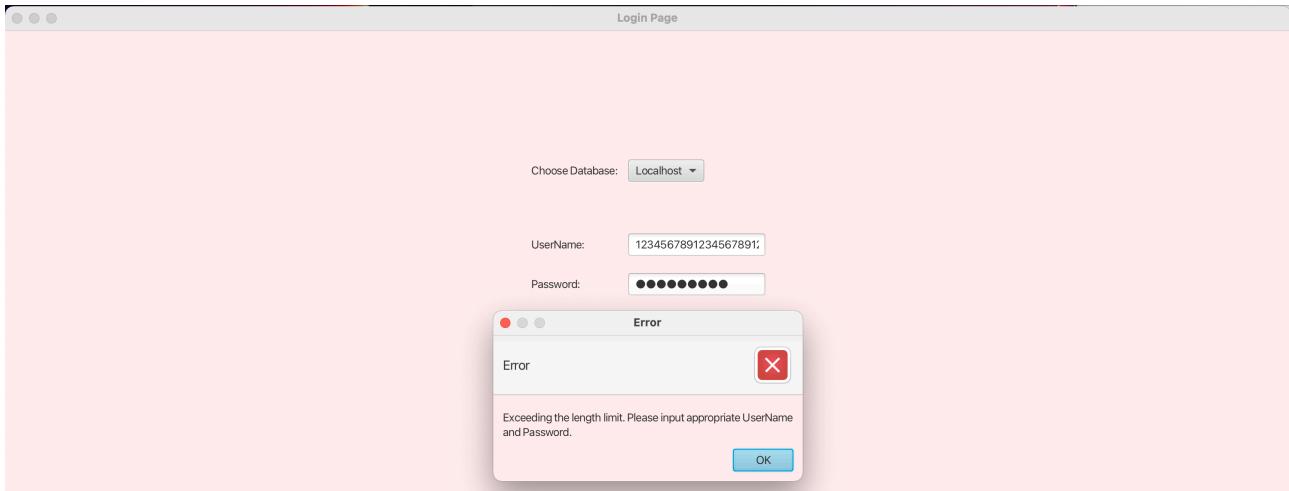
- (1). If either the username or password is empty, when click “Create”, will get an error “Please fill up both name and password.”



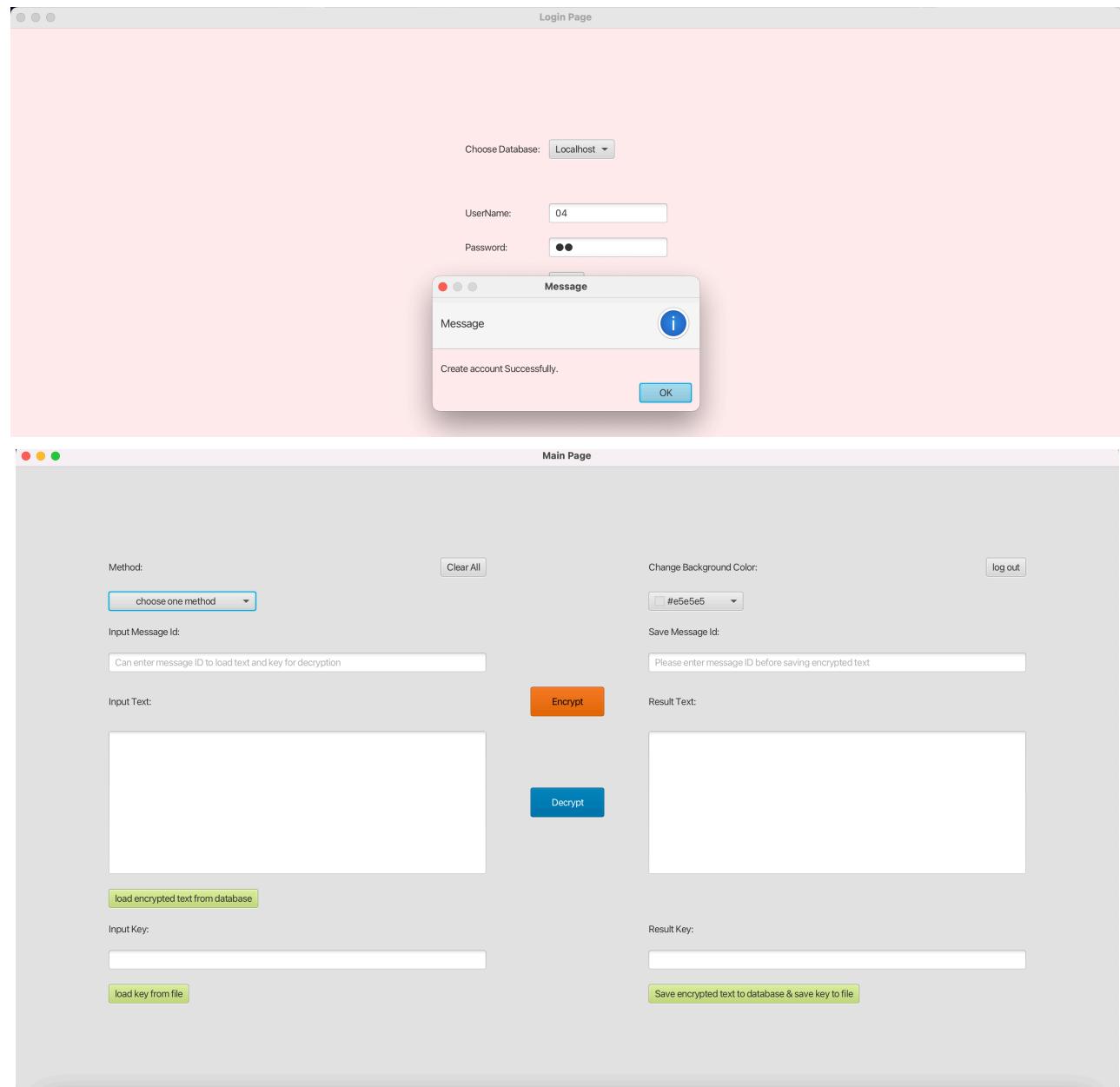
- (2).If the input username already exists in database, when click “Create”, will get an error "This userNmae exists. Please input a new userNmae."



(3).If the username or password exceeds the length limit, will get an error “Unsuccessfully”.



(4).If username and password both provided appropriately, and username is new, will get a message "Create account Successfully." Then Click "OK" button to load the default main page.



(5). Coding Logic in Create Function

Step 1: input username will be encrypted by caesarEncrypt Method(Key = 3) and password will be encrypted by hashEncrypt Method.

Step 2: check if the encrypted userName exists in database.

Step 3: if input `user_name` is empty or encrypted `user_name` exists, will show error alert.

Step 4: check if the encrypted userName and encrypted password is valid in length.

If not valid, show error alert.

If valid, the encrypted username and password will be inserted into database (security.login table). This ensures the userName and password stored in database have already been encrypted. HashEncrypted Password can't be decrypted, so this database will be more security.

At the same time, a default setting row will also be created for this new account (security.settings table)

```
private void createUser(Stage primaryStage) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);

        String userName = usernameField.getText();
        String secretUserName = method1.caesarEncrypt(userName, 3); // when create, use caesar method to encrypt username (key = 3)
        String password = passwordField.getText();
        String secretPassword = method1.hashEncrypt(password); //hashed password can't be decrypted, store hashed password in database more security. When log in, use method1.hashDecrypt(secretPassword) to get original password

        String checkUserNameSql = "SELECT * FROM login WHERE userName = ?";
        PreparedStatement checkStatement = connection.prepareStatement(checkUserNameSql);
        checkStatement.setString(1, secretUserName);
        ResultSet checkResult = checkStatement.executeQuery();

        if (userName.trim().isEmpty() || password.trim().isEmpty()) {
            Alert alert = new Alert(Alert.AlertType.ERROR, "Please fill up both name and password.");
            alert.initOwner(primaryStage);
            alert.showAndWait();
        } else if (checkResult.next()) {
            Alert alert = new Alert(Alert.AlertType.ERROR, "This user Name exists. Please input a new user Name.");
            alert.initOwner(primaryStage);
            alert.showAndWait();
        } else {
            String sql = "INSERT INTO login ('userName', 'password') VALUES (?, ?)";
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                preparedStatement.setString(1, secretUserName);
                preparedStatement.setString(2, secretPassword);
                try {
                    int rowsAffected = preparedStatement.executeUpdate();
                    if (rowsAffected > 0) {
                        Alert alert = new Alert(Alert.AlertType.INFORMATION, "Create account Successfully.");
                        alert.initOwner(primaryStage);
                        alert.showAndWait();
                        CreateUserSetting(secretUserName); // when create account successfully, create a related user setting row in settings table
                        mainPage(primaryStage, secretUserName);
                    }
                } catch (Exception e) {
                    Alert alert = new Alert(Alert.AlertType.ERROR, "Exceeding the length limit. Please input appropriate User Name and Password.");
                    alert.initOwner(primaryStage);
                    alert.showAndWait();
                }
                preparedStatement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        checkStatement.close();
        connection.close();
    } catch (Exception e) {
        System.out.println("createUser Error");
    }
}

public void createUserSetting(String userNameString) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
        String sql = "INSERT INTO settings ('setUserName', 'setBackColor', 'setDefaultMethod') VALUES (?, ?, ?)";
        PreparedStatement statement = connection.prepareStatement(sql);
        String color = "#0.9,0.9,0.9"; //default background color
        String method = "choose one method";
        statement.setString(1, userNameString);
        statement.setString(2, color);
        statement.setString(3, method);
        statement.executeUpdate();
        statement.close();
        connection.close();
    } catch (Exception e2) {
        System.out.println("createUserSetting Error");
    }
}
```

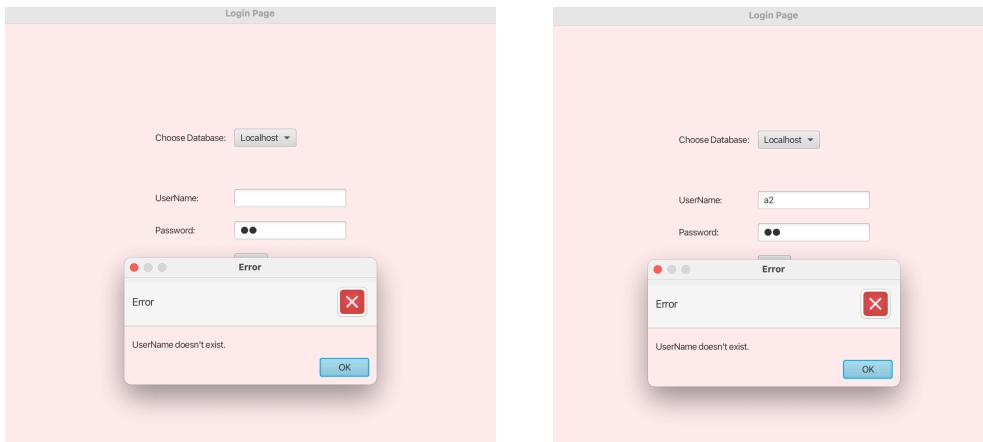
Server: localhost -> Database: security -> Table: login						Save to File
		Id	userNmae	password		
<input type="checkbox"/>	Edit	Copy	Delete	21 Y	c4ca4238a09b2820dc509a7f76549b	
<input checked="" type="checkbox"/>	Edit	Copy	Delete	22 Z	c81e728d9d4c2163607f899c14862c	
<input type="checkbox"/>	Edit	Copy	Delete	23 9	ecbc87e4b5ca2e2830bfdf2a7bf3	
<input type="checkbox"/>	Edit	Copy	Delete	24 1	a87f679a23fe71d9181a67b754212c	
<input type="checkbox"/>	Edit	Copy	Delete	28 R	c9f0895a79a91b591506197029367d	
<input type="checkbox"/>	Edit	Copy	Delete	38 7	a87f679a23fe71d9181a67b754212c	
<input type="checkbox"/>	Edit	Copy	Delete	39 8	e4db3fb7bce2345d77720d744318bd	
<input type="checkbox"/>	Edit	Copy	Delete	40 =	16790015d506195e069b7e1b2d	
<input type="checkbox"/>	Edit	Copy	Delete	41 2	cfc0208495d560e1656e7d9ff987674da	
<input type="checkbox"/>	Edit	Copy	Delete	42 J	03c7dca395b9182b07de0a2e30034	
<input type="checkbox"/>	Edit	Copy	Delete	43 5	c4a4238a09b2820dc509a7f76549b	
<input type="checkbox"/>	Edit	Copy	Delete	44 6	c81e728d9d4c2163607f899c14862c	
<input type="checkbox"/>	Edit	Copy	Delete	45 4	ecbc87e4b5ca2e2830bfdf2a7bf3	
<input type="checkbox"/>	Edit	Copy	Delete	47 YJ	f970e27671c0fe7587b857923b9d	
<input type="checkbox"/>	Edit	Copy	Delete	48 25	96a3b3cfdec2710470dab7645a2bd3	
<input type="checkbox"/>	Edit	Copy	Delete	49 26	a2e0ff6e2c2351b0e00029c9009242d	
<input type="checkbox"/>	Edit	Copy	Delete	50 24	a45ee7a7e88149af0d32b27905124d	
<input type="checkbox"/>	Edit	Copy	Delete	51 27	7d0665438e1b8dceb098c1e31ca0c01	

	Browse	Structure	SQL	Search	Insert	Export	Import	Privileges	Operations	More
Edit	Copy	Delete	setid	setUserName	setBackgroundColor			setDefaultMethod		
<input type="checkbox"/>	Edit	Copy	Delete	1 Y	0.80000011920929,0.9019607901573181,1,0			DES		
<input type="checkbox"/>	Edit	Copy	Delete	2 R	0.7019608020782471,0.9019607901573181,0.9019607901... Caesar					
<input type="checkbox"/>	Edit	Copy	Delete	4 Z	1.0,0.80000011920929,0.7019608020782471			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	5 9	0.7019608020782471,0.9019607901573181,0.7019608020... DES					
<input type="checkbox"/>	Edit	Copy	Delete	6 1	1.0,0.80000011920929,0.400000059604645			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	9 7	0.80000011920929,1.0,1,0			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	10 8	0.9490196108818054,0.9490196108818054,0.949019610... choose one method					
<input type="checkbox"/>	Edit	Copy	Delete	11 =	0.9,0.9,0.9			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	12 2	0.9,0.9,0.9			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	13 J	1.0,0.9019607901573181,0.80000011920929			Caesar		
<input type="checkbox"/>	Edit	Copy	Delete	14 5	0.9,0.9,0.9			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	15 6	0.5,0.5,0.5			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	16 4	0.80000011920929,0.9019607901573181,1,0			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	18 YJ	0.9,0.9,0.9			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	19 25	0.9,0.9,0.9			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	20 26	0.9,0.9,0.9			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	21 24	0.9,0.9,0.9			choose one method		
<input type="checkbox"/>	Edit	Copy	Delete	22 27	0.9,0.9,0.9			choose one method		

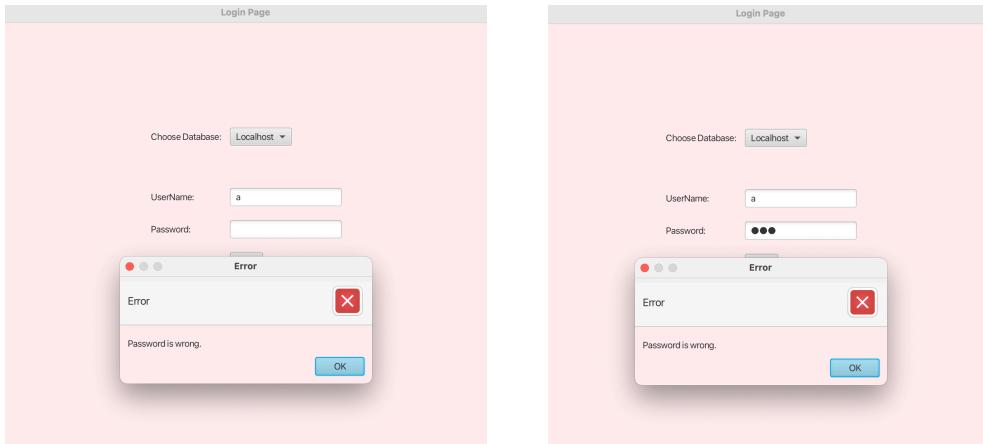
1.3 Log in

User input UserName and Password, then click “Login” button to load the user’s main page.
(Can use userName = a, password = 1 for testing)

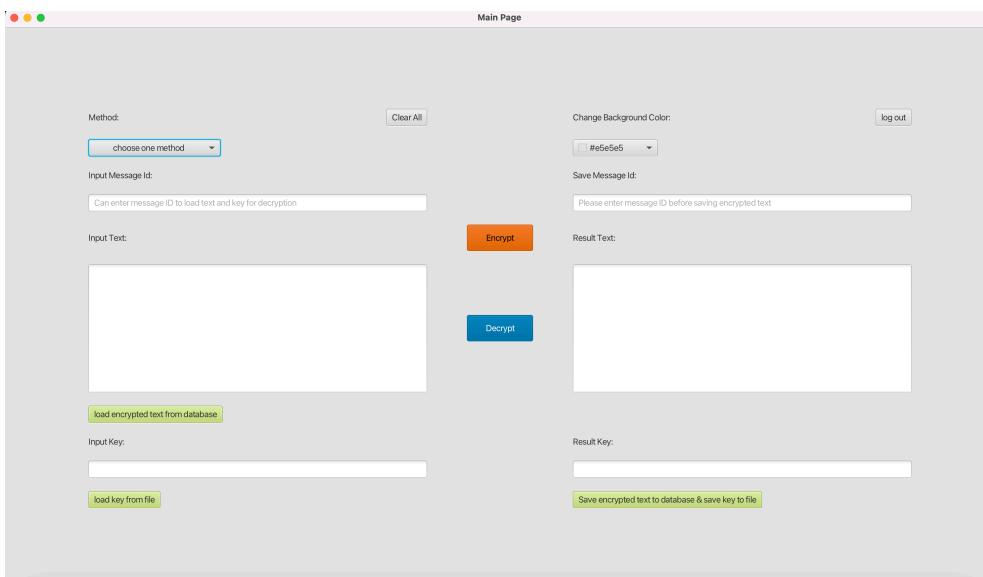
- (1). If username is empty or not exist, will get an error “UserName doesn’t exist.”



- (2). If username exists, password is empty or not correct, will get an error “Password is wrong.”



- (3). If username and password match, will automatically load the main page with this user’s setting.



(4). Coding Logic in Login Function

Step 1: input userName will be encrypted by caesarEncrypt method(Key=3) and input password will be encrypted by hashEncrypt method.

Step 2: check if the encrypted userName exists in database.

Step 3: If encrypted userName doesn't exist, show error.

If encrypted userName exists, check if encrypted password match.

If encrypted password doesn't match, show error.

If encrypted password matches, will log in successfully, load this user's main page

```
public void login(Stage primaryStage) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
        Statement statement = connection.createStatement();

        String userName = usernameField.getText();
        String secretUserName = method1.caesarEncrypt(userName, 3);
        String password = passwordField.getText();
        String secretPassword = method1.hashEncrypt(password);

Step 1
        String sql = "SELECT * FROM login where userName = '" + secretUserName + "'";
        ResultSet resultSet = statement.executeQuery(sql);

        if (resultSet.next()) {
            String hashPassword = resultSet.getString("password");
            if (secretPassword.equals(hashPassword)) {
                MainPage(primaryStage, secretUserName);
            } else {
                Alert alert = new Alert(Alert.AlertType.ERROR, "Password is wrong.");
                alert.initOwner(primaryStage);
                alert.showAndWait();
            }
        } else {
            Alert alert = new Alert(Alert.AlertType.ERROR, "UserName doesn't exist.");
            alert.initOwner(primaryStage);
            alert.showAndWait();
        }

        resultSet.close();
        statement.close();
        connection.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

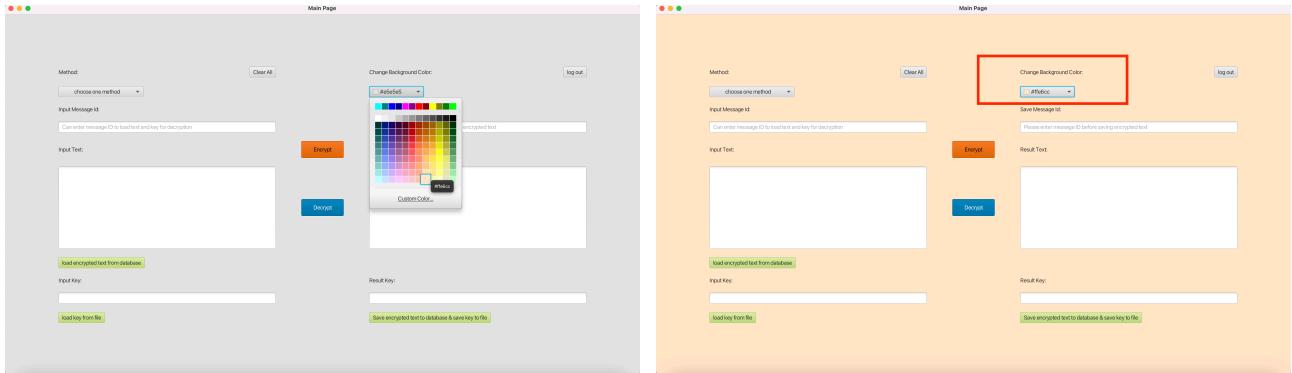
2.Main Page

2.1 setting Colour

Use colorPicker to change the background colour, the choose colour will be updated in database(security.settings table).

If user log out and log in next time, will automatically load this user's previous stored colour from database as background colour. And in colorPicker section, chosen colour will also follow the stored colour.

Each user account can save its own colour.



```
// save this user's set color to database
public void saveColorToDatabase(String userName, Color c) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);

        String updateColorSql = "UPDATE settings SET setBackColor = ? WHERE setUserName = ? ";
        PreparedStatement updateStatement = connection.prepareStatement(updateColorSql);
        String colorString = c.getRed() + "," + c.getGreen() + "," + c.getBlue();
        System.out.println("print color :" + colorString);
        updateStatement.setString(1, colorString);
        updateStatement.setString(2, userName);
        updateStatement.executeUpdate();

        updateStatement.close();
        connection.close();
    } catch (Exception e2) {
        System.out.println("saveColorToDatabase Error");
    }
}

// load user's set color from database
public void loadColorFromDatabase(String userName, GridPane grid) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
        Statement statement = connection.createStatement();
        String sql = "SELECT setBackColor FROM settings WHERE setUserName = '" + userName + "'";
        ResultSet resultSet = statement.executeQuery(sql);
        if (resultSet.next()) {
            String colorString = resultSet.getString("setBackColor");
            String colors[] = colorString.split(",");
            Color newColor = new Color(Double.parseDouble(colors[0]), Double.parseDouble(colors[1]),
                Double.parseDouble(colors[2]), 1);
            grid.setBackground(new Background(new BackgroundFill(newColor, null, null)));
            chosenColor = newColor;
        }
    } catch (Exception e2) {
        System.out.println("loadColorFromDatabase Error");
    }
}
```

2.2 setting method

User can choose one method(Caesar, DES, AES) from the menu, the input key field will show different prompt text by the choose method.

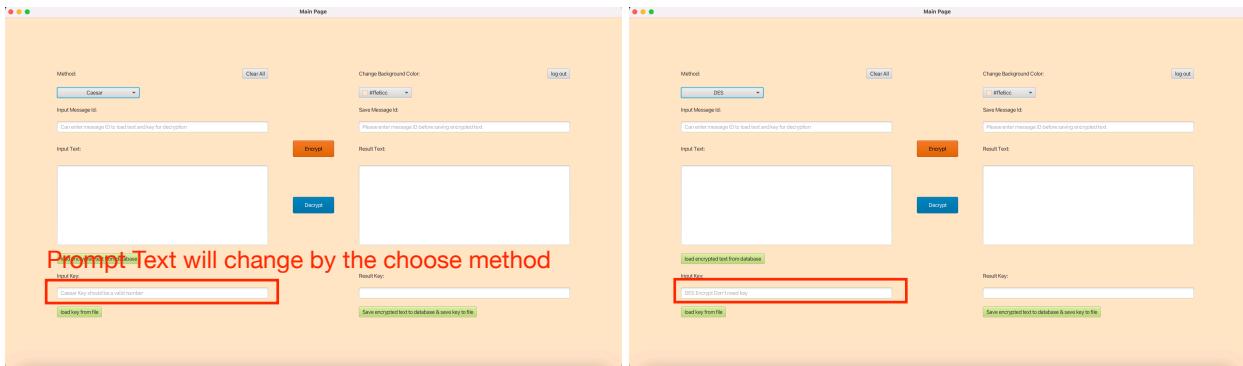
If choose Caesar, the input key field will show “Caesar Key should be a valid number”.

If choose DES, the input key field will show “DES Encrypt don’t need key”.

If choose AES, the input key field will show “AES Encrypt don’t need key”.

And at the same time, the choose method will be updated in database(security.settings table).

If user log out and log in again, will automatically load this user’s stored method from database. Each user account can save its own prefer method.



```
// save this user's set method to database
public void saveMethodToDatabase(String userName, String methodString) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);

        String updateColorSql = "UPDATE settings SET setDefaultMethod = ? WHERE setUserName = ? ";
        PreparedStatement updateStatement = connection.prepareStatement(updateColorSql);
        updateStatement.setString(1, methodString);
        updateStatement.setString(2, userName);
        updateStatement.executeUpdate();

        updateStatement.close();
        connection.close();
    } catch (Exception e2) {
        System.out.println("saveMethodToDatabase Error");
    }
}

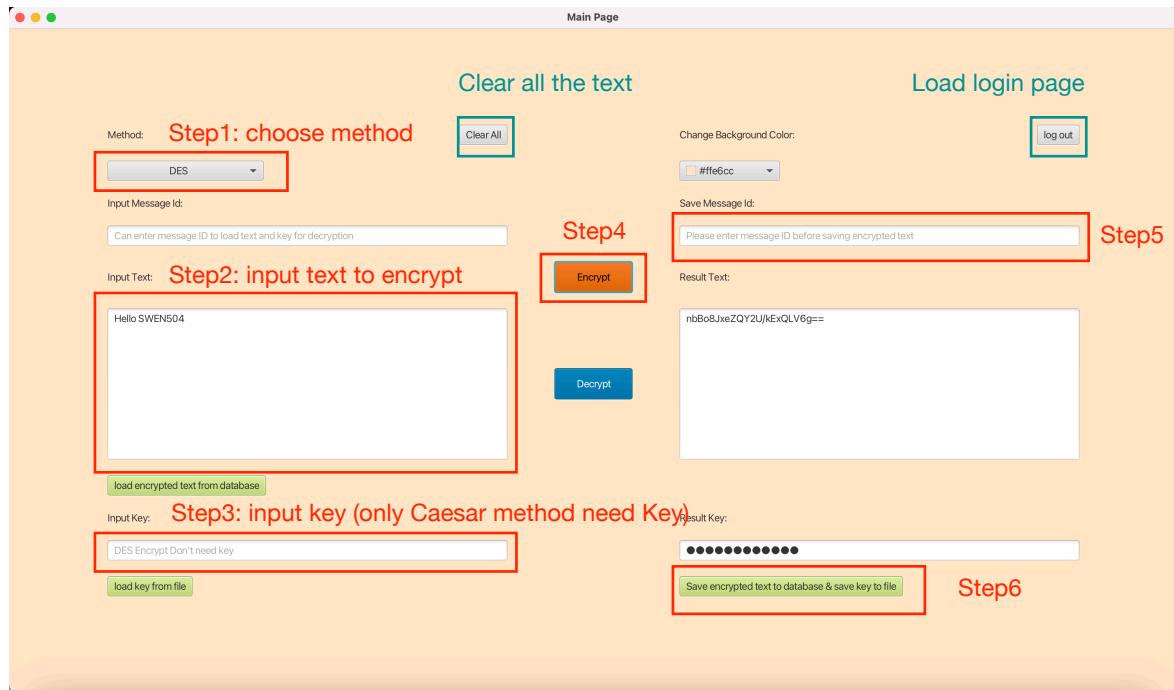
// load user's set method from database
public void loadMethodFromDatabase(String userName) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
        Statement statement = connection.createStatement();
        String sql = "SELECT setDefaultMethod FROM settings WHERE setUserName = '" + userName + "'";
        ResultSet resultSet = statement.executeQuery(sql);
        if (resultSet.next()) {
            String methodString = resultSet.getString("setDefaultMethod");
            chosenMethod = methodString;
        }
        statement.close();
        connection.close();
    } catch (Exception e2) {
        System.out.println("loadMethodFromDatabase Error");
    }
}
```

2.3 Encrypt Function

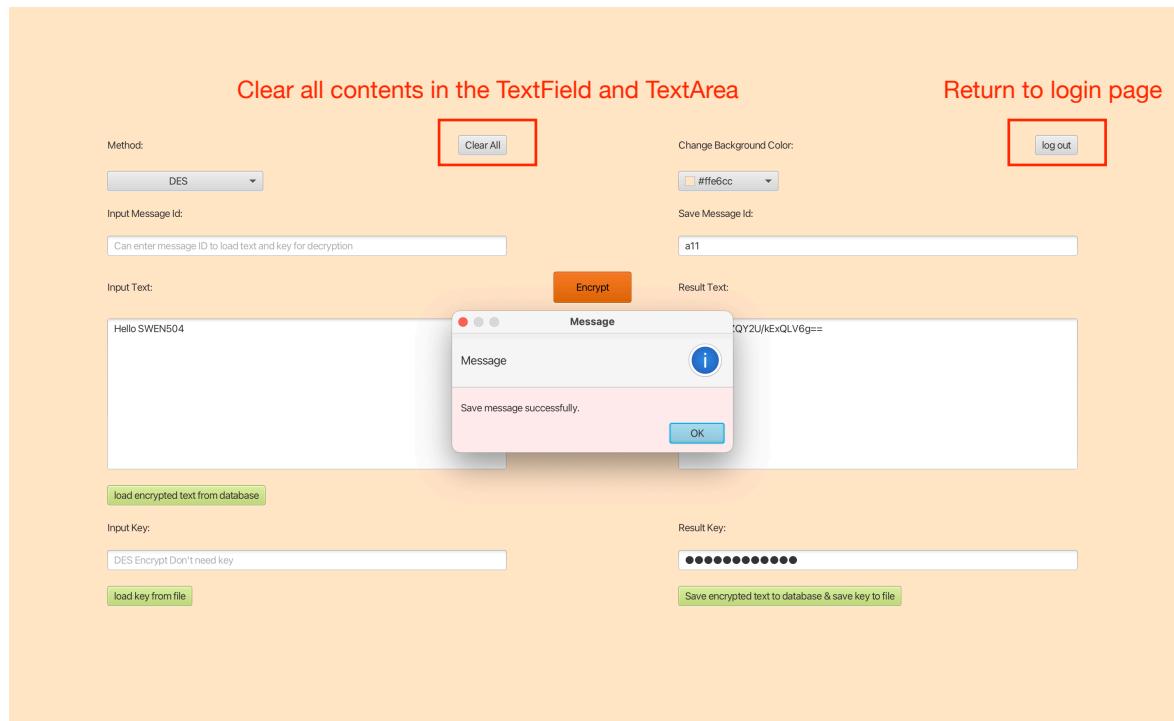
Follow from step1 to step4, can encrypt text.

The encrypted text and the key will show in the result text and result key.

If user wants to save the encrypted message, add step5 and step6. The encrypted text will be saved to database, key will be encrypted by master key (using DES method) and then stored in local file.



If the input save message id doesn't exist in this user's account, get an message show "Save message successfully".



Error Conditions:

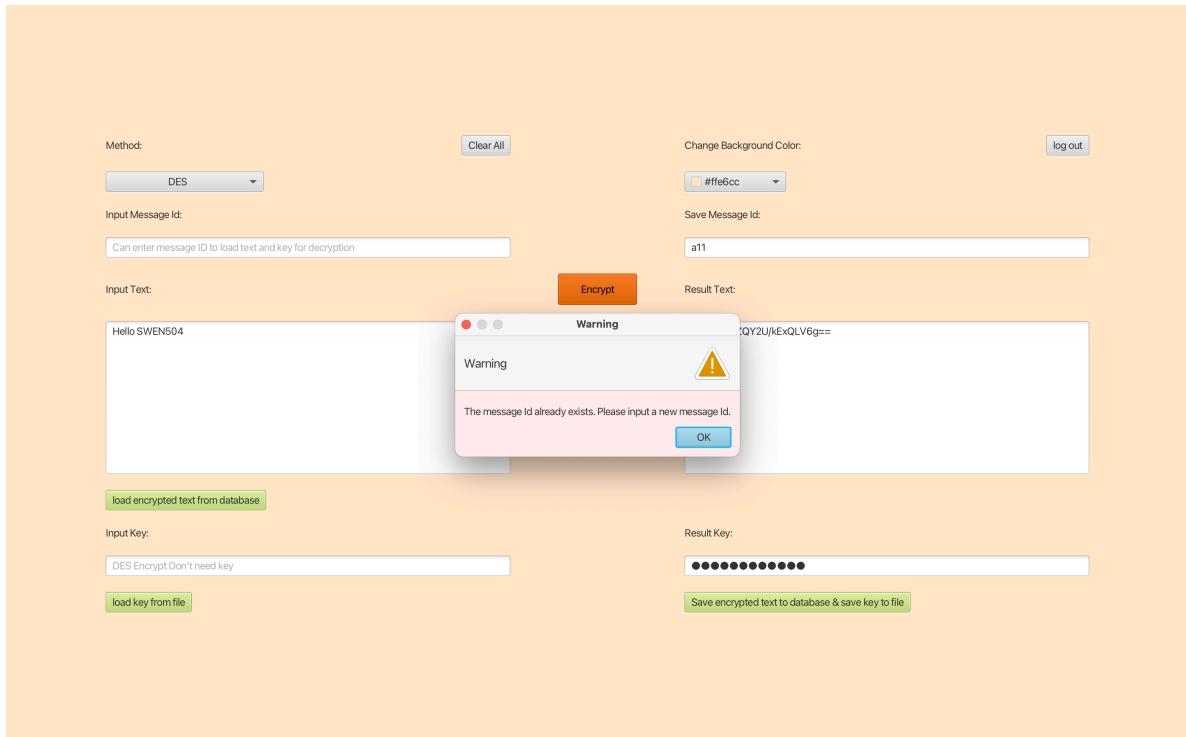
(1). If after choose method, user only input key and click “Encrypt” button, will get a prompt text in the input text show “Please input text”

The screenshot shows the application's main interface. At the top left, there is a dropdown menu labeled "Method" with "Caesar" selected. To its right are buttons for "Clear All" and "log out". On the right side, there are buttons for "Change Background Color" (set to #ffe6cc) and "Save Message Id". Below these are input fields for "Input Message Id" and "Result Text". In the center-left area, there is a large text input field labeled "Input Text" containing the placeholder "Please input text", which is highlighted with a red border. To the right of this are two buttons: "Encrypt" (orange) and "Decrypt" (blue). Below the "Input Text" field are buttons for "load encrypted text from database" and "load key from file". In the bottom-left corner, there is an "Input Key" field containing two dots ("••") and a button "Input Key".

(2). If user choose Caesar method to encrypt text, but the input key field is empty or not number, will clear the input key and show a prompt text.

This screenshot shows the same application interface as the previous one, but with different input values. The "Input Key" field now contains two dots ("••"), which is highlighted with a red border. The "Encrypt" button is orange and has a blue glow around it, indicating it is active. The "Decrypt" button is blue. The rest of the interface elements are identical to the first screenshot, including the dropdown menu, background color selection, and message ID fields.

(3).If the save message id already exists in this user's account, will get an warning.



Coding Logic in Encrypt Function

```

encryptBtn.setOnAction(e -> {
    String inputTextString = inputTextField.getText();
    String inputKeyString = inputKeyField.getText();

    if (inputTextString.trim().isEmpty()) {
        inputTextField.setPromptText("Please input text");
    } else if (chosenMethod.equals("Caesar")) {
        try {
            int key = Integer.parseInt(inputKeyString);
            String decryptString = method1.caesarEncrypt(inputTextString, key);
            resultText.setText(decryptString);
            resultKey.setText(String.valueOf(key));
        } catch (NumberFormatException f) {
            inputKeyField.clear();
            inputKeyField.setPromptText("Please input a valid number");
            f.printStackTrace();
        }
    } else if (chosenMethod.equals("DES")) {
        try {
            DES des = new DES();
            SecretKey secretKey = des.getSecretkey();
            String secretKeyString = Base64.getEncoder().encodeToString(secretKey.getEncoded());
            String decryptString = method1.desEncrypt(secretKey, inputTextString);
            resultText.setText(decryptString);
            resultKey.setText(secretKeyString);
        } catch (NumberFormatException | NoSuchAlgorithmException f) {
            f.printStackTrace();
        }
    } else if (chosenMethod.equals("AES")) {
        try {
            AES aes = new AES();
            SecretKey secretKey = aes.getSecretkey();
            String secretKeyString = Base64.getEncoder().encodeToString(secretKey.getEncoded());
            String decryptString = method1.aesEncrypt(secretKey, inputTextString);
            resultText.setText(decryptString);
            resultKey.setText(secretKeyString);
        } catch (NoSuchAlgorithmException f) {
            f.printStackTrace();
        }
    }
});
```

Coding Logic in Save Encrypted Text Function

```

saveMessageButton.setOnAction(e -> {
    String message = resultText.getText();
    String key = resultKey.getText();
    String secretKey = method1.hashEncrypt(key);
    String messageIdString = saveMessageIdField.getText();
    String methodName = chosenMethod;

    String saveKeyString = resultKey.getText();
    byte[] secretKeyBytes = method1.desEncrypt(desMasterKey, saveKeyString);

    if (!messageIdString.trim().isEmpty()) {
        saveMessageById(primaryStage, userName, message, secretKey, messageIdString, methodName, secretKeyBytes);
    } else {
        Alert alert = new Alert(Alert.AlertType.ERROR, "Please input save id.");
        alert.initOwner(primaryStage);
        alert.showAndWait();
    }
});

public void saveMessageById(Stage primaryStage, String userName, String message, String key, String idString,
String methodName, byte[] secretKeyB) {
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);

    //first check if this message id already exists in this user's message row.
    String checkMessageIdSql = "SELECT * FROM message WHERE loginUserName = ? AND messageId = ? ";
    PreparedStatement checkMessageIdStatement = connection.prepareStatement(checkMessageIdSql);
    checkMessageIdStatement.setString(1, userName);
    checkMessageIdStatement.setString(2, idString);
    ResultSet checkMessageIdResult = checkMessageIdStatement.executeQuery();

    if (checkMessageIdResult.next()) {
        Alert alert = new Alert(Alert.AlertType.WARNING, "The message Id already exists. Please input a new message Id.");
        alert.initOwner(primaryStage);
        alert.showAndWait();
    } else {
        String updateMessageSql = "INSERT INTO message (`methodName`, `encryptText`, `hashKey`, `loginUserName`, `messageId`) VALUES (?, ?, ?, ?, ?)";
        PreparedStatement updateStatement = connection.prepareStatement(updateMessageSql);
        updateStatement.setString(1, methodName);
        updateStatement.setString(2, message);
        updateStatement.setString(3, key);
        updateStatement.setString(4, userName);
        updateStatement.setString(5, idString);
        int rowsAffected = updateStatement.executeUpdate();

        if (rowsAffected > 0) {
            String saveFileName = userName + idString + ".txt";
            method1.saveKeyFile(saveFileName, secretKeyB);
            Alert alert = new Alert(Alert.AlertType.INFORMATION, "Save message successfully.");
            alert.initOwner(primaryStage);
            alert.showAndWait();
        } else {
            Alert alert = new Alert(Alert.AlertType.ERROR, "Unsuccessfully.");
            alert.initOwner(primaryStage);
            alert.showAndWait();
        }
        updateStatement.close();
    }
    checkMessageIdStatement.close();
    connection.close();
} catch (Exception e2) {
    System.out.println("saveMessageToDatabaseById Error");
}
}

```

Save encrypted text to database

Save encrypted key to local file

The screenshot shows the phpMyAdmin interface for the 'message' table. The table has columns: loginUserName, encryptText, hashKey, id, methodName, and messageId. The highlighted row (row 27) contains the following values:

loginUserName	encryptText	hashKey	id	methodName	messageId
ntbBo8.lxeZQY2U/kExQLVbg==	566874e6d3de7c186334e5f5de117361		23	DES	a11

Saved message example

2.4 Decrypt Function

Follow from step1 to step4, can decrypt text.

The decrypted text and the key will show in the result text and result key.

Method: Step1: choose correct method

Input Message Id:

Input Text: Step2: input text to decrypt

Input Key: Step3: input key

Change Background Color:

Save Message Id:

Encrypt

Result Text: Hello SWEN504

Result Key:

load encrypted text from database

load key from file

Save encrypted text to database & save key to file

log out

If user wants to use the encrypted message that already stored in database and file, can choose load button, the stored decrypt method, stored encrypted text and key will show automatically.

Show Automatically by load

Method: DES

Input Message Id: a11

Input Text: nbBo8JxeZQY2U/kExQLV6g==

Input Key: *****

Change Background Color:

Save Message Id:

Encrypt

Result Text: Hello SWEN504

Result Key:

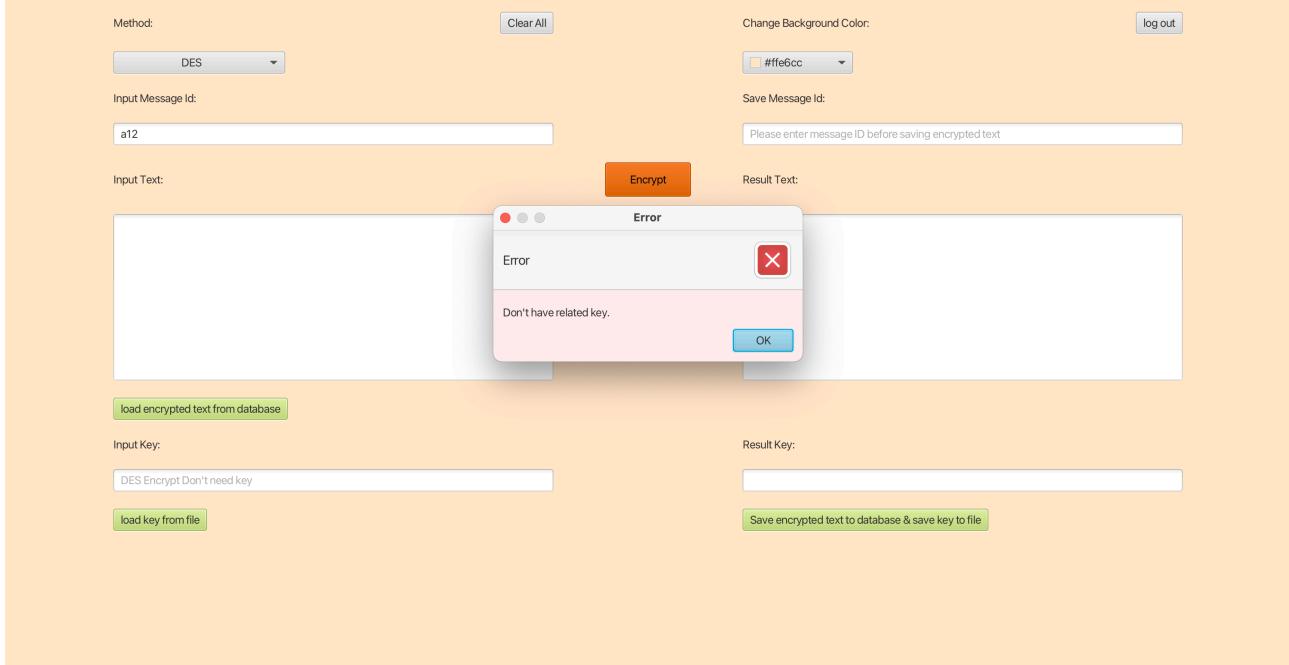
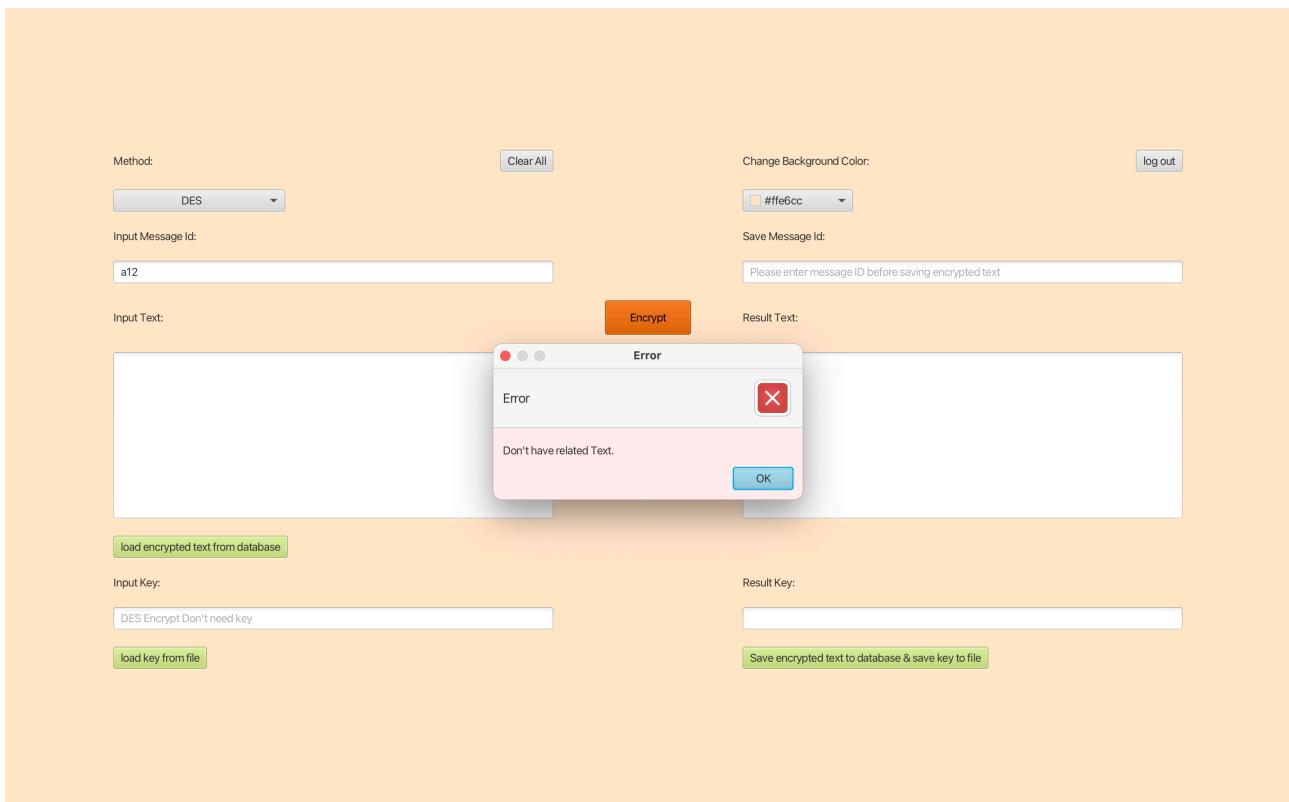
load encrypted text from database

load key from file

Save encrypted text to database & save key to file

Step1 Step2 Step3 Step4

If the save message id doesn't exist in this user's account, when load text and load key, will get an warning.



Coding Logic in Decrypt Function

```
decryptBtn.setOnAction(e -> {
    String inputTextString = inputTextField.getText();
    String inputKeyString = inputKeyField.getText();

    if (inputTextString.trim().isEmpty()) {
        inputTextField.setPromptText("Please input text");
    } else if (chosenMethod.equals("Caesar")) {
        try {
            int key = Integer.parseInt(inputKeyField.getText());
            try {
                String encryptString = method1.caesarDecrypt(inputTextString, key);
                resultText.setText(encryptString);
            } catch (Exception e2) {
                e2.printStackTrace();
            }
        } catch (Exception f) {
            inputKeyField.clear();
            inputKeyField.setPromptText("Please input a valid number");
            f.printStackTrace();
        }
    } else if (chosenMethod.equals("DES")) {
        try {
            String decryptString = method1.desDecrypt(inputKeyString, inputTextString);
            resultText.setText(decryptString);
        } catch (Exception f) {
            f.printStackTrace();
        }
    } else if (chosenMethod.equals("AES")) {
        try {
            String decryptString = method1.aesDecrypt(inputKeyString, inputTextString);
            resultText.setText(decryptString);
        } catch (Exception f) {
            f.printStackTrace();
        }
    }
});
```

Coding Logic in Loading Encrypted Text and Key

```
loadTextButton.setOnAction(e -> {
    inputTextField.clear();
    inputTextField.setPromptText("");
    // get the correct key name by userName and input id
    String inputIdText = inputMessageIdField.getText();
    String textString = getMessageTextById(primaryStage, userName, inputIdText);
    inputTextField.setText(textString);
});

loadKeyButton.setOnAction(e -> {
    inputKeyField.clear();
    inputTextField.setPromptText("");

    // get the correct key name by userName and input id
    String inputIdText = inputMessageIdField.getText();
    String idString = getKeyFileIndexById(primaryStage, userName, inputIdText);

    if (!idString.trim().isEmpty()) {
        // read and decrypt key
        String file = userName + idString + ".txt";
        String loadKey = "";
        try {
            loadKey = method1.loadKeyFile(file);
        } catch (Exception f) {
            f.printStackTrace();
        }
        String keyString = method1.desDecrypt(desMasterKey, loadKey);
        System.out.println("2key string " + keyString);
        inputKeyField.setText(keyString);
        if (!chosenMethod.isEmpty()) {
            methodMenuButton.setText(chosenMethod);
        }
    } else {
        System.out.println("The key file doesn't exist");
    }
});
```

```

public String getMessageTextById(Stage primaryStage, String userName, String inputId) {
    String getMessageText = "";
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);

        String checkSql = "SELECT * FROM message WHERE loginUserName = ? AND messageId = ? ";
        PreparedStatement checkStatement = connection.prepareStatement(checkSql);
        checkStatement.setString(1, userName);
        checkStatement.setString(2, inputId);
        ResultSet checkResult = checkStatement.executeQuery();
        if (checkResult.next()) {
            getMessageText = checkResult.getString("encryptText");
            chosenMethod = checkResult.getString("methodName");
        } else {
            Alert alert = new Alert(Alert.AlertType.ERROR, "Don't have related Text.");
            alert.initOwner(primaryStage);
            alert.showAndWait();
        }
        checkStatement.close();
        connection.close();
    } catch (Exception e2) {
        System.out.println("getMessageTextById Error");
    }
    return getMessageText;
}

public String getKeyFileIndexById(Stage primaryStage, String userName, String inputId) {
    String keyFileIndex = "";
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);

        String checkSql = "SELECT * FROM message WHERE loginUserName = ? AND messageId = ? ";
        PreparedStatement checkStatement = connection.prepareStatement(checkSql);
        checkStatement.setString(1, userName);
        checkStatement.setString(2, inputId);
        ResultSet checkResult = checkStatement.executeQuery();
        if (checkResult.next()) {
            keyFileIndex = checkResult.getString("messageId");
            // change the default method when call
            chosenMethod = checkResult.getString("methodName");
        } else {
            Alert alert = new Alert(Alert.AlertType.ERROR, "Don't have related key.");
            alert.initOwner(primaryStage);
            alert.showAndWait();
        }
        checkStatement.close();
        connection.close();
    } catch (Exception e2) {
        System.out.println("checkMessageId Error");
    }
    return keyFileIndex;
}

```