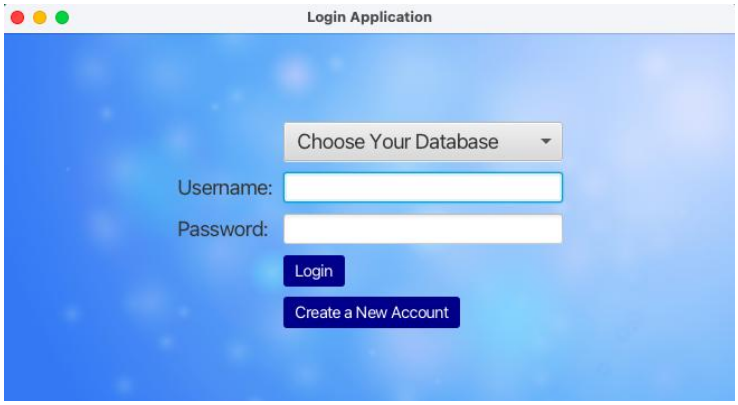
SWEN504 - Cloud and Security Report

Dan Luo

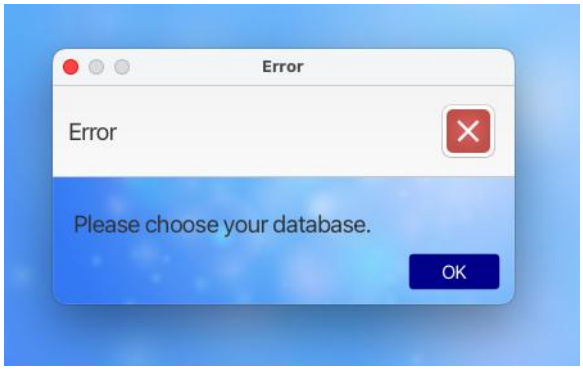
Part 1: Application's Functionality

1.1 Create Account

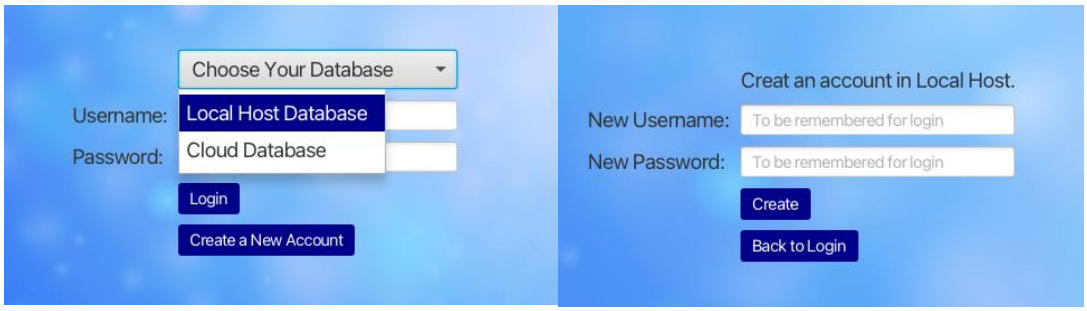


Step 1: Choose the database.

Before create a new account, users can choose which database they would like to create for. If not, an alert information will be presented.

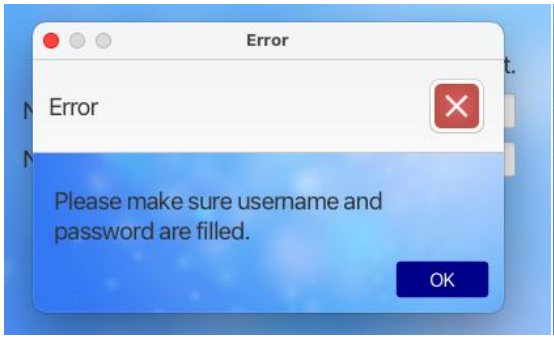


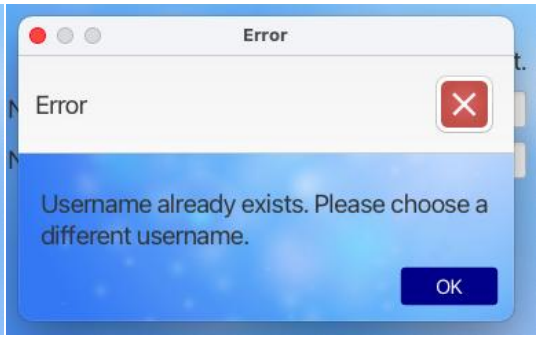
Step 2: Create an account in the chosen database.



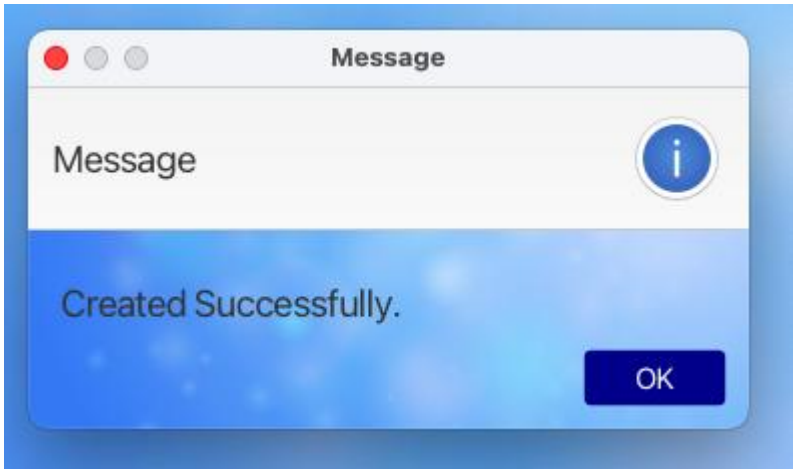
“Back to Login” button if user do not want to create an new account and will be back to the login page.

If username or password field is empty, an alert info will be presented. If username is already exist in the database, an alert info will be presented as well.





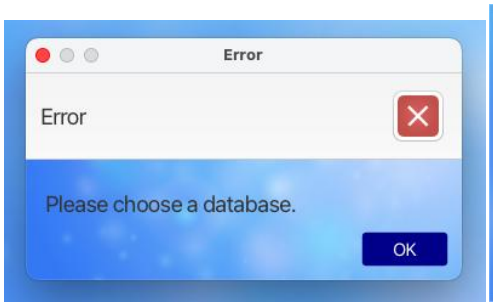
Otherwise, account can be created successfully and jump to login page.

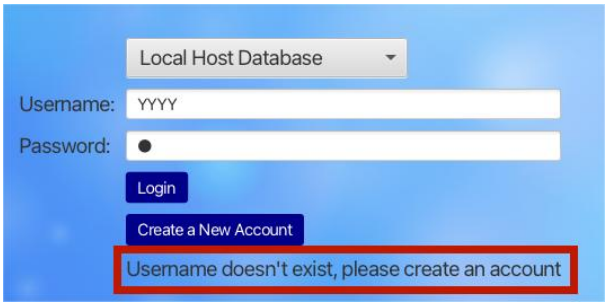


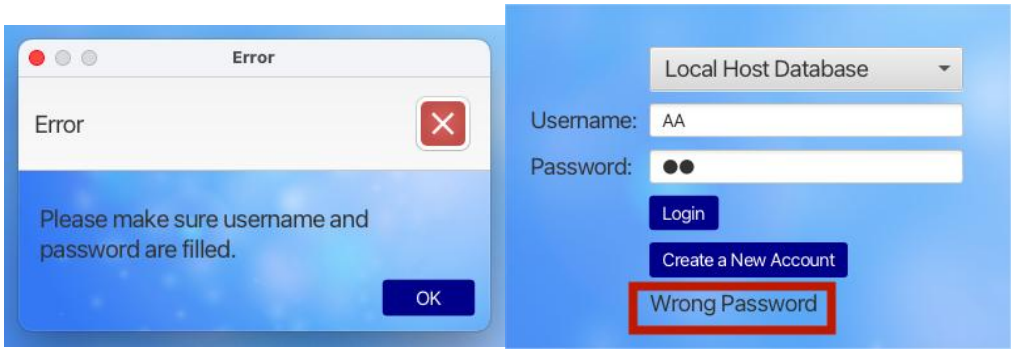
Every time a new user created, the user name and hashed password will be insert into the chosen database (table users, local host database as an example).

1.2 Login

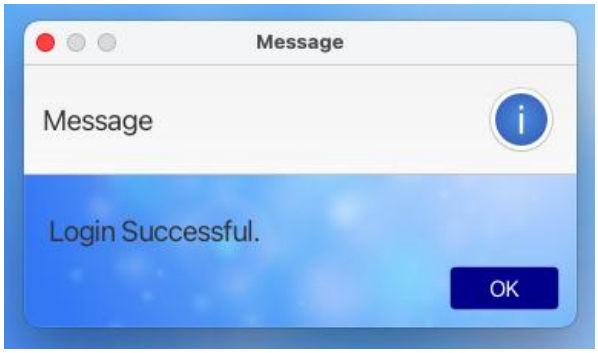
Some alert info are set for different unexpected situations.



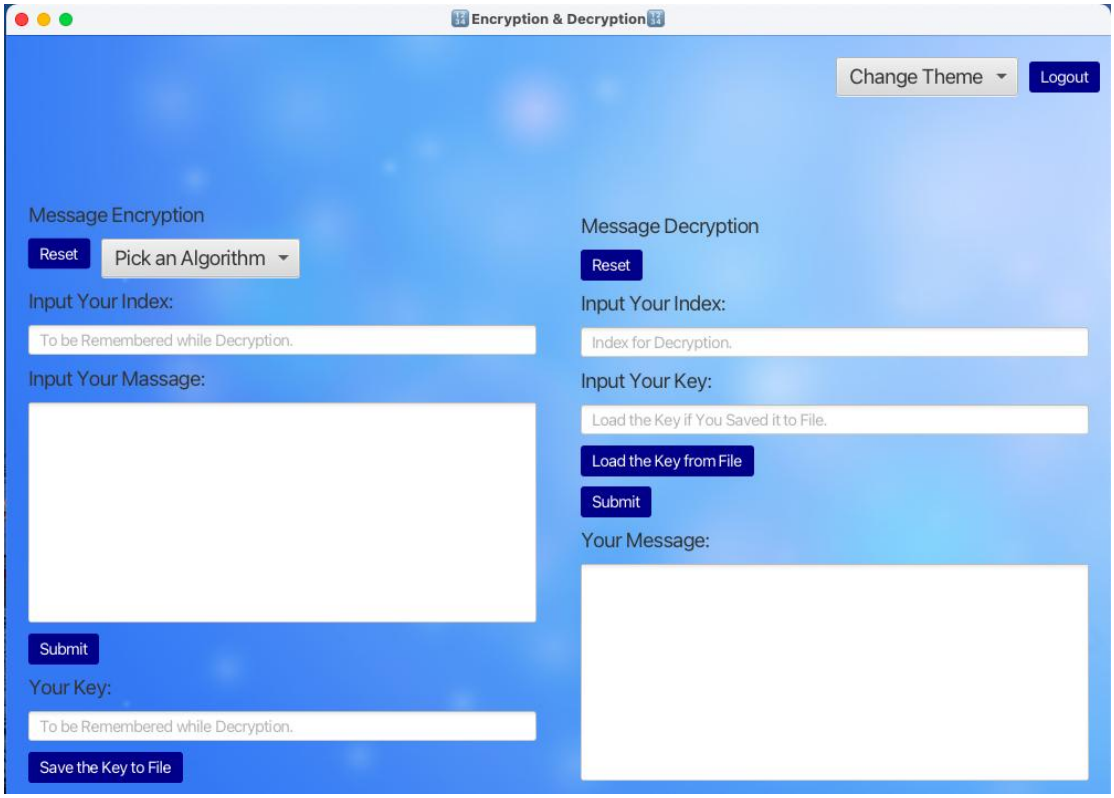




Successful login if username and password are matched. Then jump to the main page that can encrypt and decrypt messages.

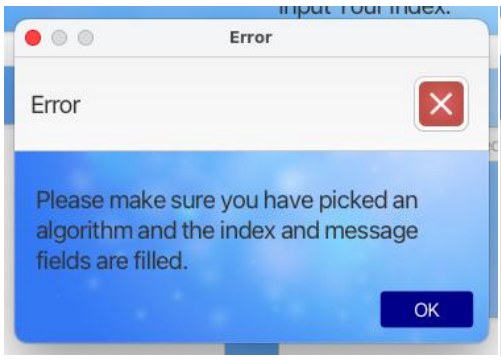


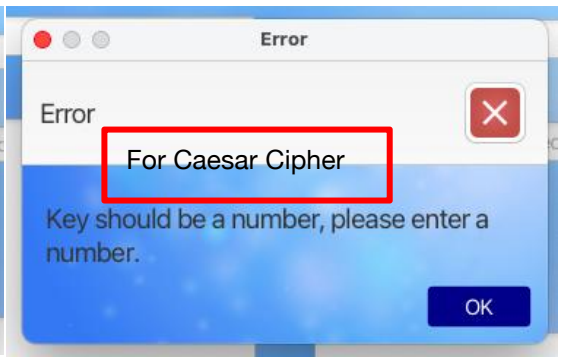
The main page as below. Left side for encryption and the right side for decryption. Theme chooser and logout button can be found on the top right.

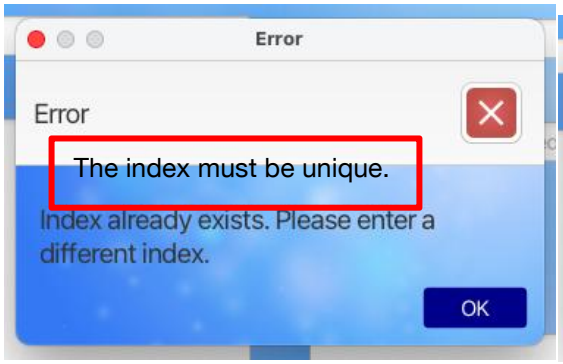


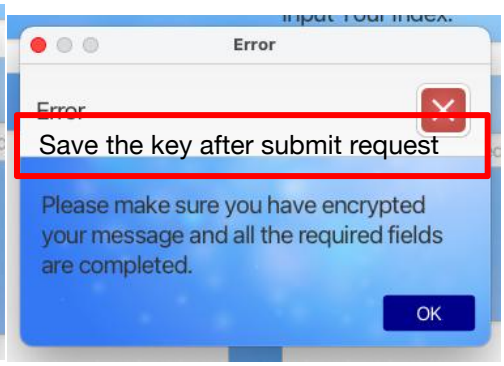
1.3 Encryption

Some alert info are set for different unexpected situations.

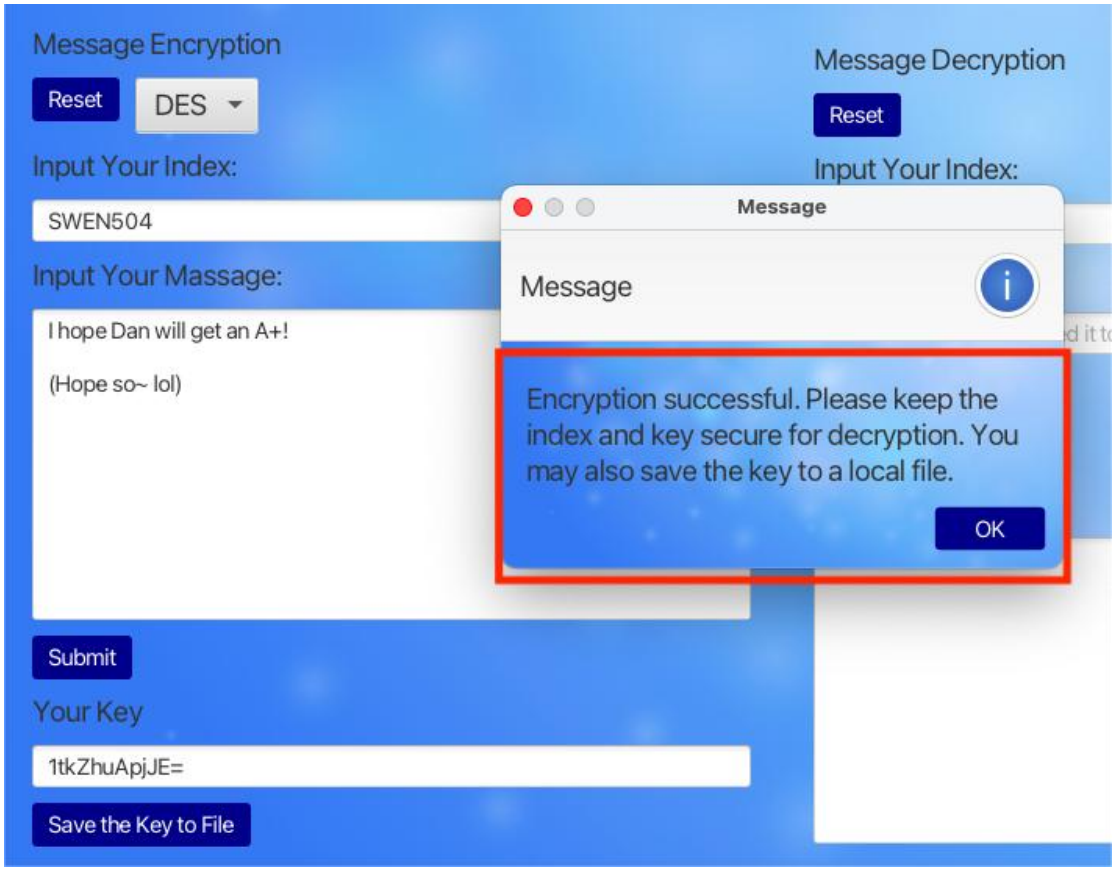




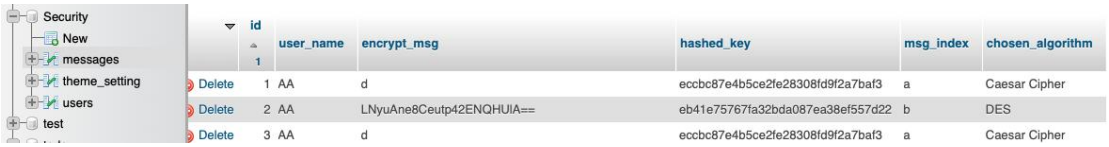




Encrypt successful if user input a unique index. A information will be presented:

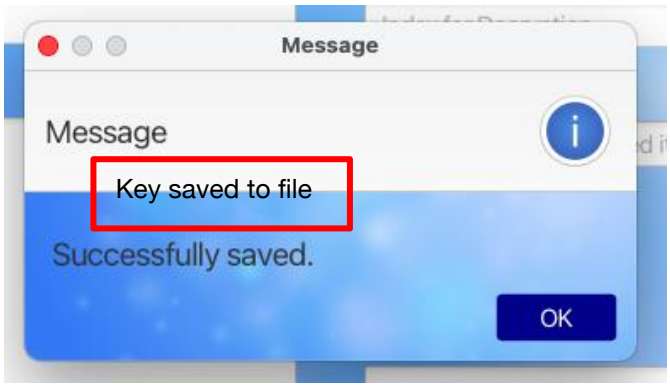


The username, encrypt message, hashed key, index and chosen algorithm will be insert into the chosen database automatically(table messages). To make sure the security, key will be hashed.



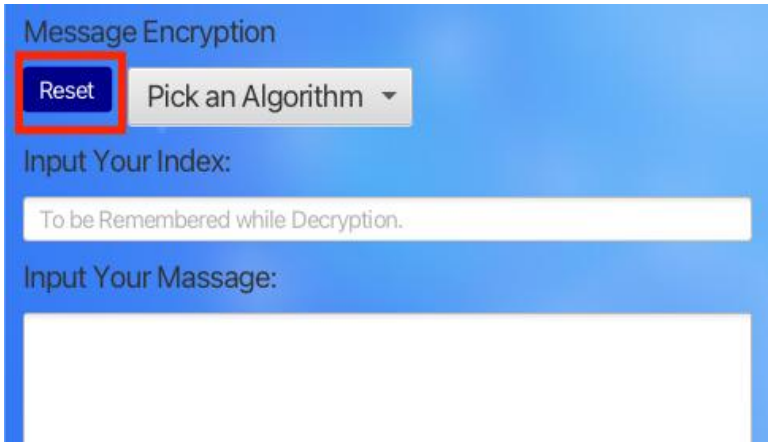
	id	user_name	encrypt_msg	hashed_key	msg_index	chosen_algorithm
Delete	1	AA	d	eccbc87e4b5ce2fe28308fd9f2a7baf3	a	Caesar Cipher
Delete	2	AA	LNyuAne8Ceutp42ENQHUIA==	eb41e75767fa32bda087ea38ef557d22	b	DES
Delete	3	AA	d	eccbc87e4b5ce2fe28308fd9f2a7baf3	a	Caesar Cipher

Users can choose whether to save the key to file or not. If they do not save the key, then the key should be remembered by the users themself.



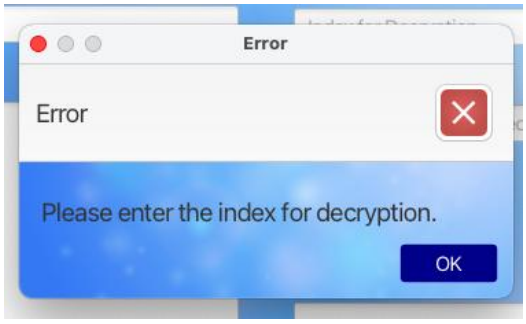
An file with encrypted key can be found on local.

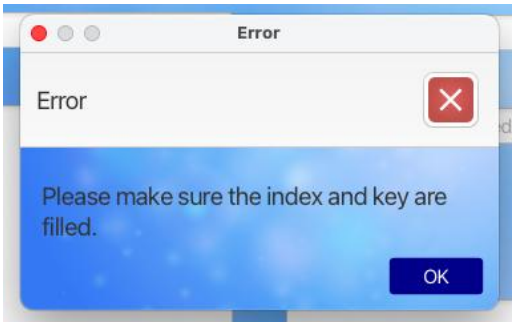
Click “Reset” button will clear all the text field, user-friendly repeat encryption.



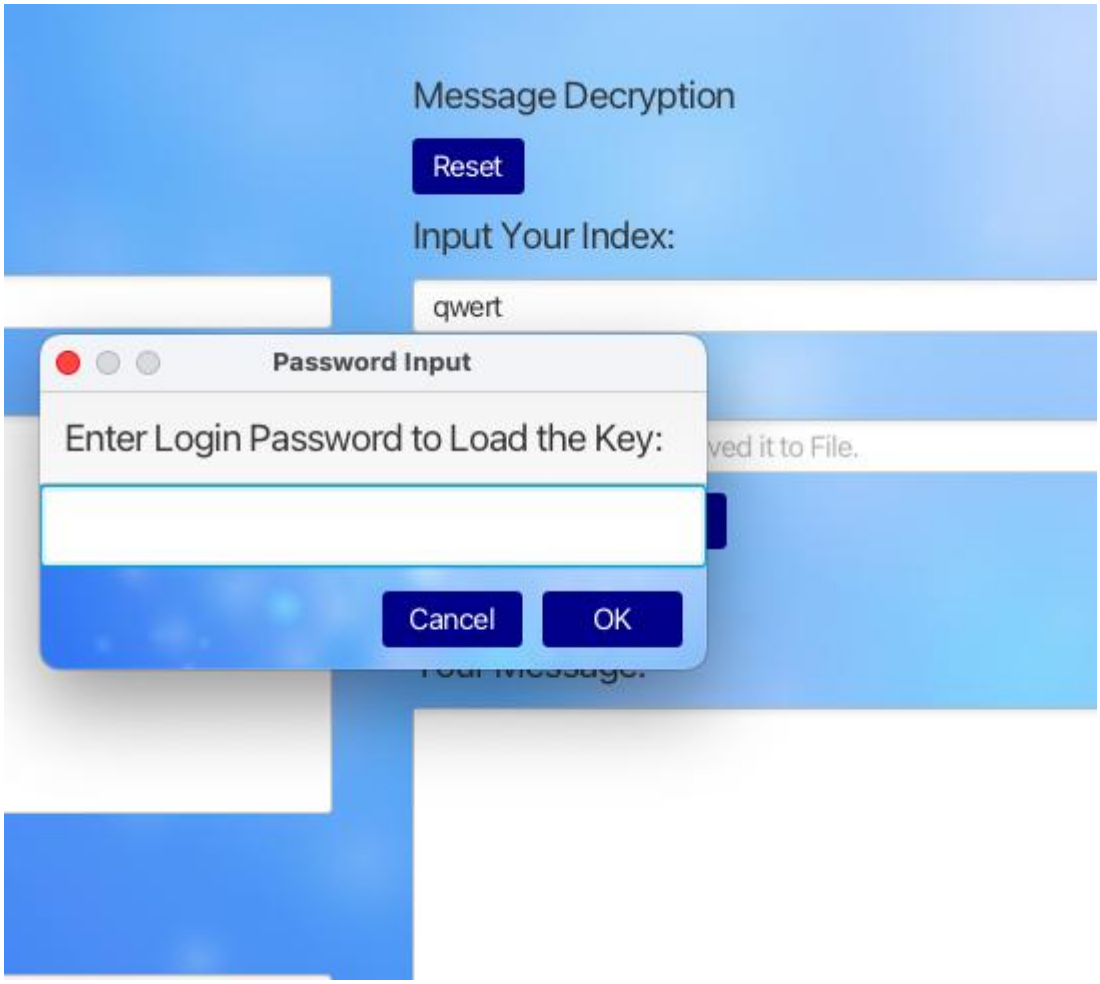
1.4 Decryption

Some alert info are set for different unexpected situations.

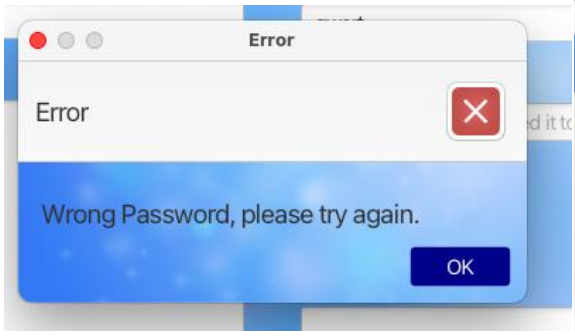


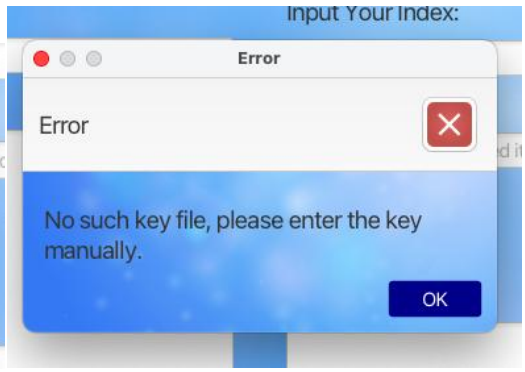


User can decrypt the message by enter the index and key. If the key is stored in local file, “Load the Key from File” button can help user to obtain the key. But it require the login password.

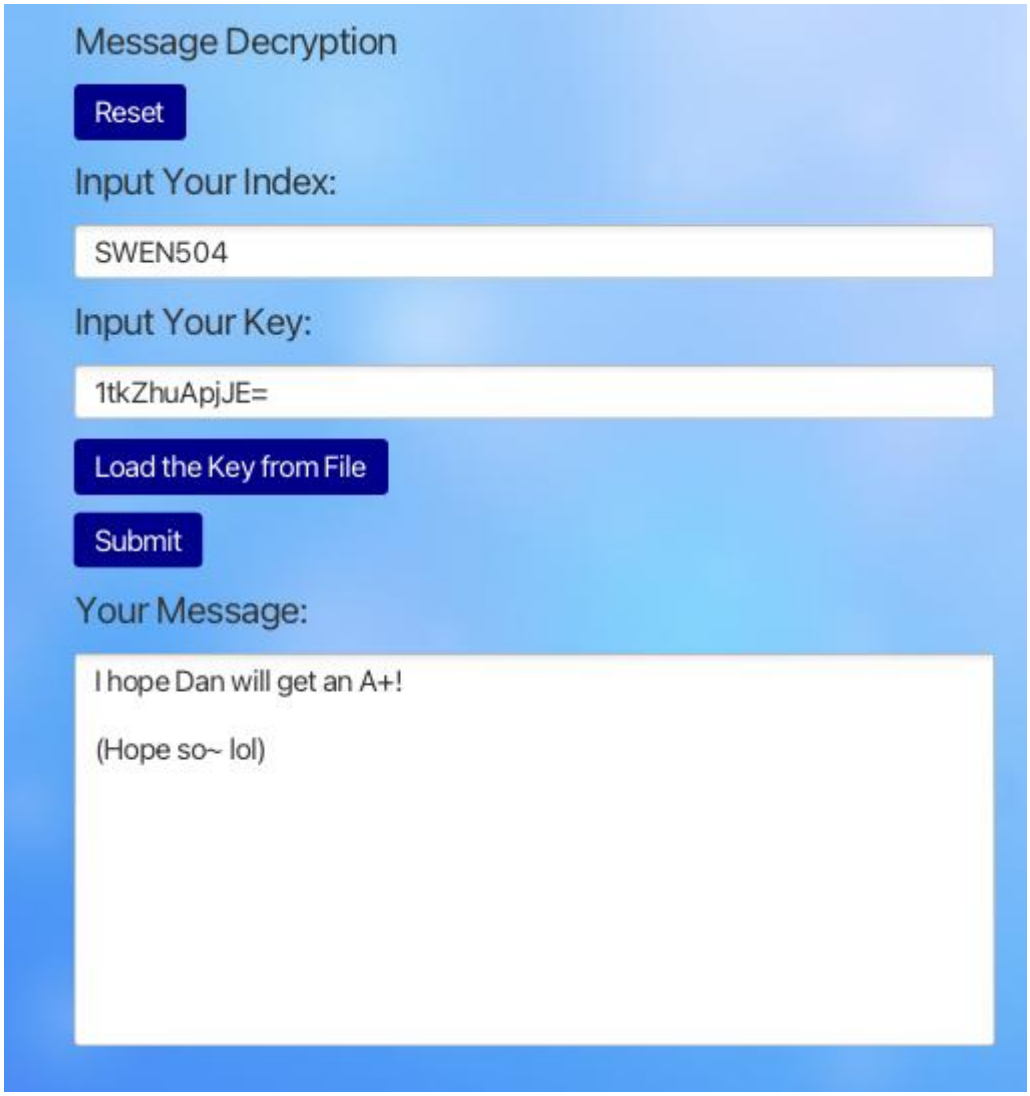


If the login password is wrong or the key is not saved on local file, error info will be presented:



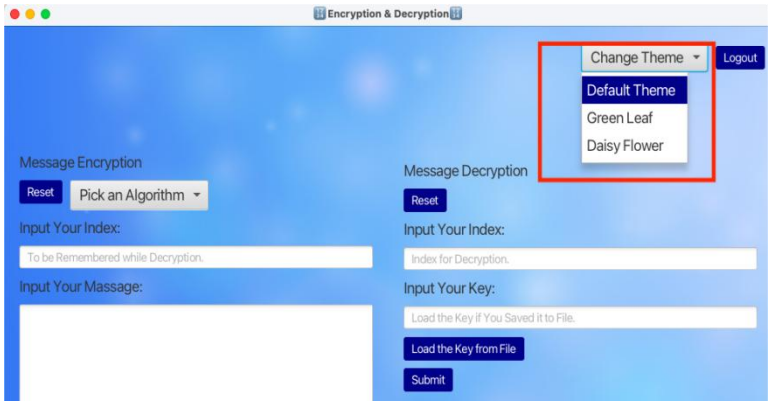


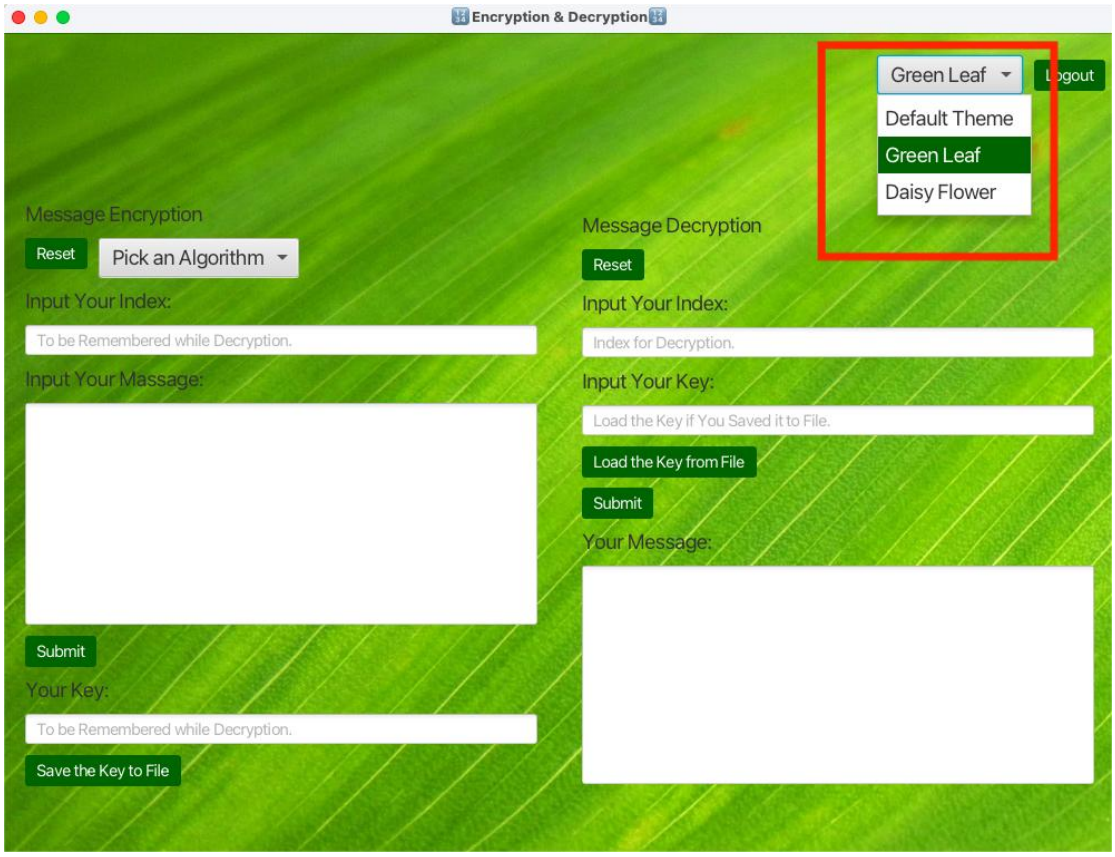
Hit the submit button, if the index and key are matched, the original message will be appeared on the message text area.

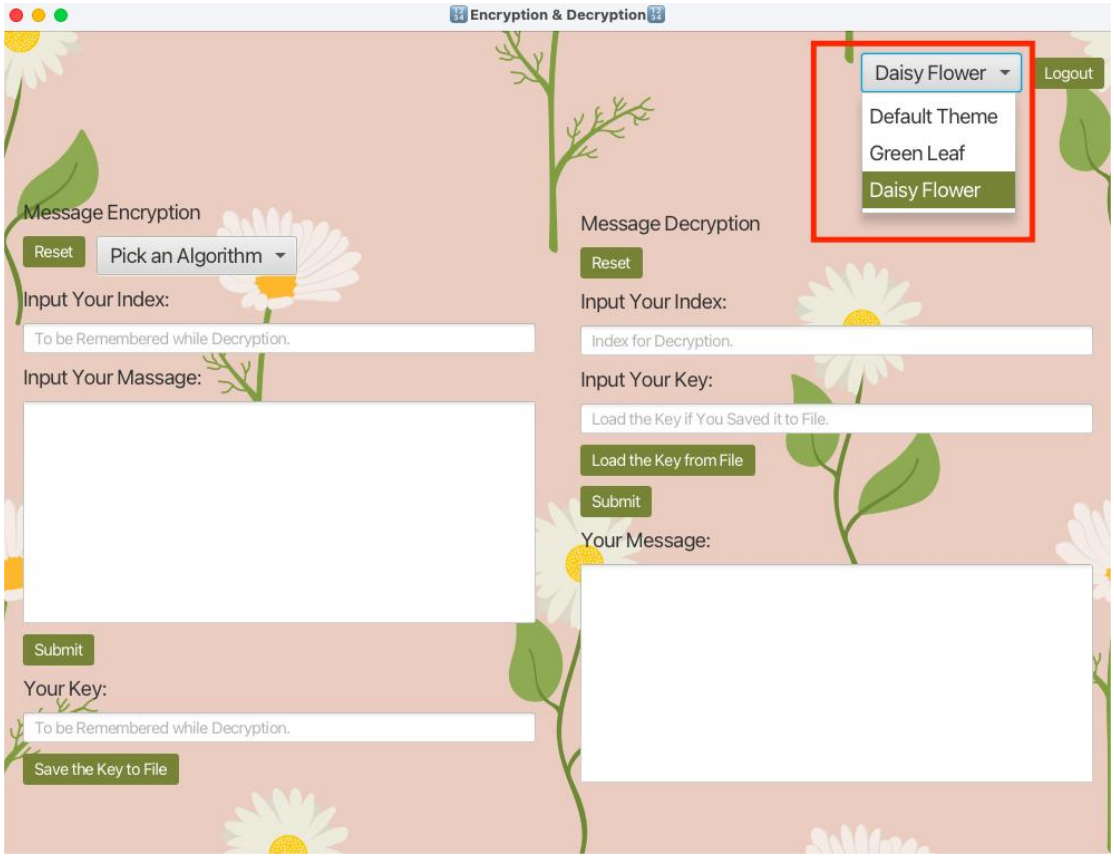


1.5 Change Theme

Users can choose three different themes. When create a new account, username and the default theme will insert into the chosen database(table theme_setting). By choose different themes, chosen one will be saved to their database. When next login, the main page will be presented the last theme they have chosen (if not chosen any, then show the default theme).

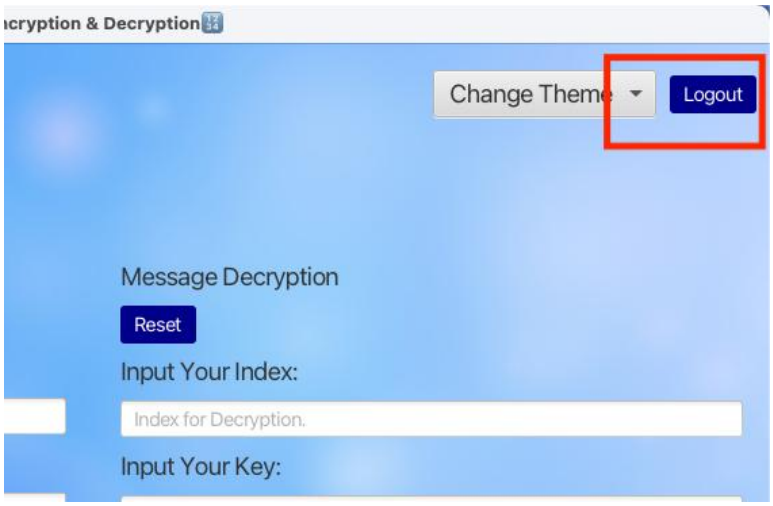






1.6 Logout

Hit the “Logout” button will direct to login page.



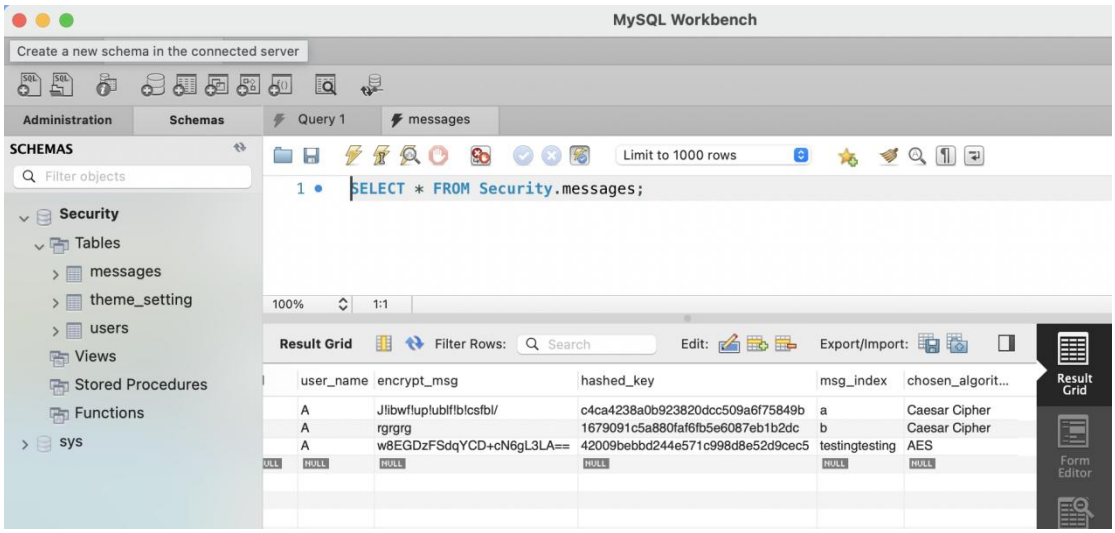
1.7 Database: Local Host and AWS

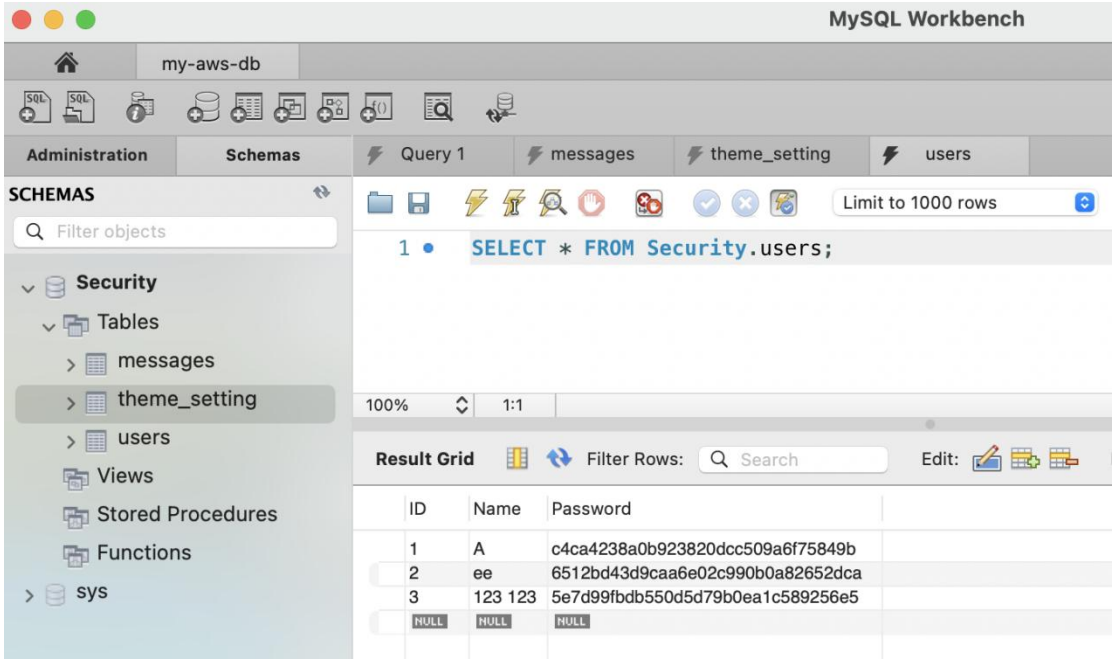
Both Local Host and AWS database are applied in this application.

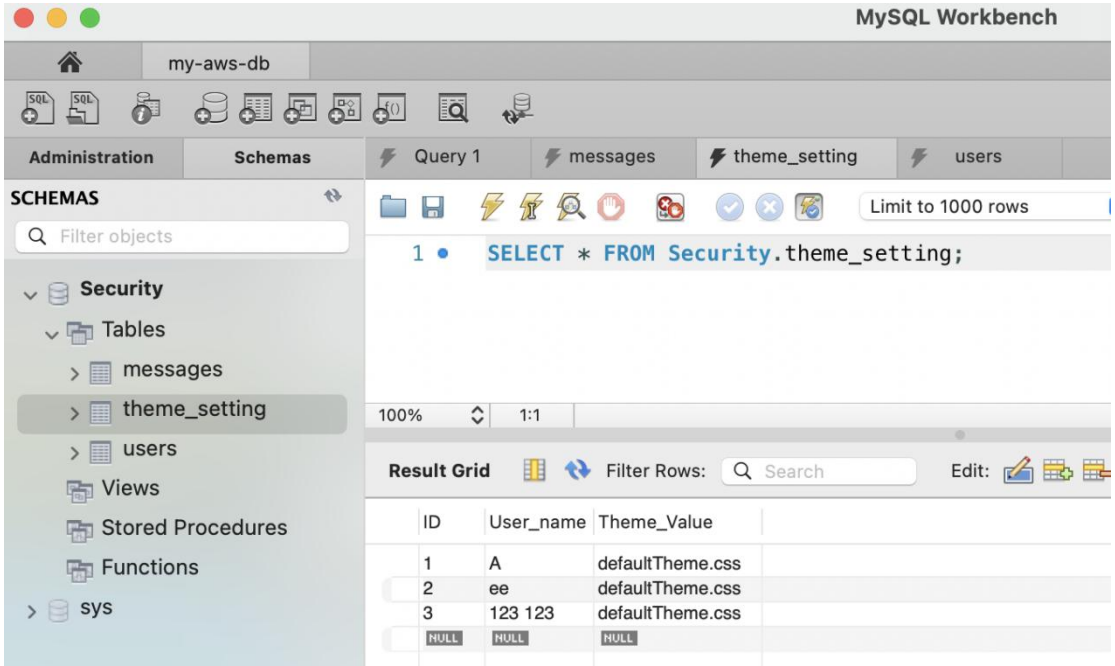
```java
// choose different database
dbChooserBtn = new MenuButton("Choose Your Database");
GridPane.setConstraints(dbChooserBtn, 1, 0);
dbChooserBtn.setMinWidth(250);
MenuItem localDb = new MenuItem("Local Host Database");
MenuItem cloudDb = new MenuItem("Cloud Database");
dbChooserBtn.getItems().addAll(localDb, cloudDb);

localDb.setOnAction(event -> {
    dbChooserBtn.setText("Local Host Database");
    JDBC_URL = "jdbc:mysql://127.0.0.1:3306/Security";
    USERNAME = "root";
    PASSWORD = "";
    chosenDB = "Local Host";
});
cloudDb.setOnAction(event -> {
    dbChooserBtn.setText("Cloud Database");
    JDBC_URL = "jdbc:mysql://my-aws-db.c[illegible]3306/Security";
    USERNAME = "admin";
    PASSWORD = "adminisdan";
    chosenDB = "Cloud";
});
```

MySQL Workbench:




MySQL Workbench
my-aws-db
Administration
Schemas
Query 1
messages
theme_setting
users
SCHEMAS
Filter objects
Security
Tables
messages
theme_setting
users
Views
Stored Procedures
Functions
sys
Limit to 1000 rows
1 SELECT * FROM Security.users;
100%
1:1
Result Grid
Filter Rows:
Search
Edit:
ID Name Password
1 A c4ca4238a0b923820dcc509a6f75849b
2 ee 6512bd43d9caa6e02c990b0a82652dca
3 123 123 5e7d99fbdb550d5d79b0ea1c589256e5
NULL NULL NULL


MySQL Workbench
my-aws-db
Administration
Schemas
Query 1
messages
theme_setting
users
SCHEMAS
Filter objects
Security
Tables
messages
theme_setting
users
Views
Stored Procedures
Functions
sys
Limit to 1000 rows
1 SELECT * FROM Security.theme_setting;
100%
1:1
Result Grid
Filter Rows:
Search
Edit:
ID User_name Theme_Value
1 A defaultTheme.css
2 ee defaultTheme.css
3 123 123 defaultTheme.css
NULL NULL NULL

Part 2: Individual Reflection

2.1 Learning

There are three things that I gained the most in this week: 1. learning to use JDBC to connect to a database; 2. understanding the concept of cloud databases; and 3. thinking deeply about data security. The database connection is like breathing life into a programme, no longer is it a application of one-off operations. And data security made me think more about the internal logic and usability of the application. Next I will describe a few of my design decisions about store the key.

2.2 Design Decisions

The first one is the login password. At the very beginning, I directly encrypted the user's password using DES, and saved the encrypted password and key in the database at the same time. This significantly deviates from security principles. Therefore, I implemented the hashed password method in the program to securely store the user's password after hashing, rather than storing the key along with it.

```
private void createAccountMethod(Stage primaryStage) {

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

        String newUserName = usernameField.getText();
        String newPassword = passwordField.getText();
        String encryptPassword = algorithm.hashPassword(newPassword);
```

The second is the master key. Until today, my master key (used to encrypt the user's key stored in a local file) the was hard coded in the code. At the last minute, I decided to store it in a more secure way. In the current version, the master key is stored in a separate file, config.properties. It won't be the most secure method, but it's a big improvement in security over hard coded.

```
public String getMasterKey() {
    Properties properties = new Properties();
    try (InputStream input = getClass().getClassLoader().getResourceAsStream("config.properties")) {
        properties.load(input);
    } catch (IOException e) {
        e.printStackTrace();
    }
    String masterK = properties.getProperty("masterKey");
    return masterK;
}
```

The last one is how the user load key from local file. Despite the encryption of the key in the local file, I would raise concerns about its security if it can be extracted with a simple click of a button. In designing the "Load Key from File" button, I took into consideration that the user needs to enter the login password to load the key from local file, which can provide an extra layer of protection.

```java
// load key from file
loadKeyBtn.setOnAction(event -> {
    String decryptIndex = decryptIndexField.getText();
    masterKey = algorithm.getMasterKey();
    if (decryptIndex != null && !decryptIndex.isEmpty()) {
        String pw = getPasswordFromUser(primaryStage);
        Boolean validated = validatePassword(userName, pw);
        if (validated) {
            try {
                String loadedKey = algorithm.loadKeyFile(decryptIndex);
                String decryptLoadedKey = algorithm.AESDecrypt(masterKey, loadedKey);
                decryptKeyField.setText(decryptLoadedKey);
            } catch (Exception e) {
                algorithm.showErrorInfo(primaryStage, "No such key file, please enter the key manually.");
                e.printStackTrace();
                return;
            }
        } else {
            algorithm.showErrorInfo(primaryStage, "Wrong Password, please try again.");
        }
    } else {
        algorithm.showErrorInfo(primaryStage, "Please enter the index for decryption.");
    }

});
```

In general, the design decisions were mainly to add a few extra layers of security. While the application of these methods isn't perfect, it does play into the enhancement over my initial idea.

2.3 Conclusion

Overall, this assignment has boosted my confidence and provided me with a better understanding of programming. I recognize that patience is key to furthering my learning. A special thanks to Ali, Michael, and Buddhima for their invaluable help!