**Case Study: Development of a User Authentication and Data Management Application Using Java and MongoDB**

**1. Problem Statement**

**In today's digital world, managing user data securely and efficiently is a critical requirement for many applications. A common challenge is creating user management systems with features such as authentication, registration, and data management. Many small to medium-sized applications require a simple yet robust solution that allows users to register, log in, and manage different types of data (e.g., notes, reminders, birthdays) in a secure and user-friendly manner.**

**Key Objectives:**

**Implement a login and registration system.**

**Provide a dashboard for managing user-specific data (e.g., notes, reminders).**

**Store user data securely using a NoSQL database (MongoDB).**

**Create a user-friendly GUI for easy interaction.**

**2. Approach to Solve the Problem**

**The solution proposed is a Java application that provides the following features:**

**User Authentication: Users can log in with a unique username and password.**

**User Registration: Users can sign up if they don't have an account.**

**Data Management: Users can manage personal data such as notes, diaries, birthdays, and reminders with options to add, edit, delete, and search data.**

**Database Integration: MongoDB is used as the database to store user credentials and data content.**

**3. Technology Stack**

**Java: A versatile programming language for developing the application's core logic and user interface.**

**Swing: Used to create the graphical user interface (GUI) for login, registration, dashboard, and data management.**

**MongoDB: A NoSQL database for storing and managing user credentials and content.**

**Maven: For managing project dependencies.**

**4. Detailed System Design**

**Login and Registration System:**

**Login Form: A user enters a username and password. The system checks if the credentials match a record in the MongoDB database. If they do, the user is authenticated and directed to the dashboard.**

**Sign-Up Form: New users can create an account by providing a username and password. The system checks if the username already exists in the database. If it does not, a new user is created in the database.**

**Dashboard:**

**After logging in, the user is presented with a dashboard featuring buttons to manage different types of data (e.g., Notebook, Diary, Birthdays, and Reminders).**

**Each section allows users to add, view, edit, and delete content.**

**The application uses MongoDB collections to store and retrieve the data.**

**Data Management:**

**Notebook: Stores and allows users to manage their notes.**

**Diary: Allows users to store diary entries.**

**Birthdays: Stores information about birthdays and allows adding/removing birthday reminders.**

**Reminders: Stores user-created reminders.**

**Each of these categories has a corresponding collection in MongoDB where data is stored. The system ensures that data is securely handled and provides the functionality to search for entries based on a search term.**

**5. Database Design**

**MongoDB will have the following collections:**

**users: Contains user credentials (username, password).**

**notebook: Stores notes with fields like content.**

**diary: Stores diary entries.**

**birthdays: Stores birthday reminders.**

**reminders: Stores user-created reminders.**

**Each collection stores data as documents, and MongoDB allows flexible and scalable data management. Each document has fields for storing relevant information (e.g., content, dates, etc.).**

**6. System Workflow**

**Login Process: When a user enters their username and password, the system checks the database for a match. If successful, the user is redirected to the dashboard; if not, an error message appears.**

**Sign-Up Process: When a user signs up, the system checks if the username already exists in the database. If it doesn’t, the user is registered; if it does, the user is prompted to choose a different username.**

**Dashboard Interaction: Once logged in, users can interact with their data (notes, diary entries, birthdays, reminders) by clicking on the relevant button on the dashboard. This opens a new window where users can add, edit, delete, or search for specific data.**

**7. Analysis of the Application**

**a. Security Considerations:**

**Password Storage: In a real-world application, storing passwords in plain text is a security risk. The application should ideally hash the passwords before storing them in the MongoDB database using libraries like BCrypt or PBKDF2 to ensure data protection.**

**Database Connection: The MongoDB URI should not be hardcoded in production applications. Environment variables or configuration files should be used to store sensitive information like the database URI.**

**b. Scalability:**

**MongoDB: MongoDB is a scalable, flexible NoSQL database, suitable for handling large amounts of data. The application can handle an increasing number of users and data by scaling the database horizontally (i.e., adding more nodes to the cluster).**

**User Management: The system can be extended to support more complex user roles, such as administrators or moderators.**

**c. User Interface:**

**Swing is an excellent choice for simple desktop applications, but if the application needs to scale or run on multiple platforms (mobile/web), transitioning to web frameworks (e.g., Spring Boot with React or Angular) would be ideal for greater flexibility and accessibility.**

**d. Error Handling:**

**The application should include more robust error handling mechanisms. For example, if the MongoDB server is down, users should be notified properly rather than seeing a generic error message.**

**e. User Experience:**

**The user interface should be intuitive, and it would be beneficial to add more features such as user profile management, password recovery, and settings for more personalization.**

**8. Conclusion**

**The development of this Java-based user authentication and data management application is a practical solution for applications requiring secure user login, registration, and data management features. By utilizing MongoDB for data storage, the system is scalable and flexible for future expansions. While this solution is effective for a basic desktop application, further improvements in security, error handling, and user interface design can enhance the system's reliability and usability.**

**9. Recommendations for Improvement:**

**Hash passwords to increase security.**

**Implement session management to keep users logged in after authentication.**

**Enhance the user interface by adopting modern frameworks.**

**Deploy the application as a web or mobile application for broader access.**

**This analysis can serve as a foundational study for anyone looking to develop a similar application in Java, focusing on user management, authentication, and data storage using MongoDB**

**Project Description:**

This project is a simple Java-based application that demonstrates user authentication using MongoDB as the database. It includes a login form, a sign-up form, and a dashboard with data management options. The application uses **Swing** for the user interface and **MongoDB** for storing and managing user data. The key features are:

* **Login Form**: Users can enter their credentials (username and password) to authenticate.
* **Sign-Up Form**: New users can create an account with a unique username and password.
* **Dashboard**: After logging in, users can access various features (notebook, diary, birthdays, reminders).
* **Data Management**: Users can manage items like notes, birthdays, and reminders with options to add, edit, delete, and search for entries.
* **MongoDB Integration**: Data is stored in MongoDB and is fetched for display in tables. The database is used for storing user credentials as well as the content for notebooks, diaries, birthdays, and reminders.

**User Manual:**

1. **Login**:
   * Launch the application.
   * Enter your username and password in the login form.
   * If the credentials are valid, you will be redirected to the dashboard.
   * If invalid, an error message will appear.
2. **Sign-Up**:
   * If you are a new user, click the "Sign Up" button on the login page.
   * Enter your username and password.
   * If successful, you will be redirected to the login page to log in.
3. **Dashboard**:
   * After logging in, you will have access to various sections such as Notebook, Diary, Birthdays, and Reminders.
   * Click on any section to manage the data.
4. **Data Management**:
   * **Search**: Enter a search term and click the search button to filter data.
   * **Add**: Use the "Save" button to add new data.
   * **Edit**: Select a row in the table and click "Edit" to modify the content.
   * **Delete**: Select a row and click "Delete" to remove an entry.

**Project Code:**

The project is divided into multiple classes for different functionalities:

1. **Main.java**:
   * Contains the login and sign-up forms.
   * Handles user authentication and redirects to the dashboard.
2. **MongoDBConnection.java**:
   * Establishes a connection to MongoDB.
   * Inserts documents into MongoDB collections.
3. **Dashboard.java**:
   * Displays the main dashboard with buttons to navigate to various sections (Notebook, Diary, etc.).
4. **DataWindow.java**:
   * Displays and manages data in tables for the selected category (e.g., notebook, diary, etc.).
   * Allows saving, editing, deleting, and searching data.

**Database Structure:**

* **MongoDB** is used to store the data:
  + **users** collection: Stores user credentials (username, password).
  + **notebook**, **diary**, **birthdays**, and **reminders** collections: Store data for respective categories (with fields like content).

**Dependencies:**

This project uses **MongoDB Java Driver** (version 4.3.4) for database interactions. It is managed using **Maven** for dependency management.

**PROJECT CODE**

**MAVEN.POM.MXL**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>

    <artifactId>demo</artifactId>

    <version>1.0-SNAPSHOT</version>

    <properties>

        <maven.compiler.source>17</maven.compiler.source>

        <maven.compiler.target>17</maven.compiler.target>

    </properties>

    <dependencies>

        <dependency>

            <groupId>org.mongodb</groupId>

            <artifactId>mongodb-driver-sync</artifactId>

            <version>4.3.4</version>

        </dependency>

    </dependencies>

</project>

**2 DATABASE CONNECTION**

**MONGODBCONNECTION .JAVA**

package com.example;

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

import org.bson.Document;

public class MongoDBConnection {

    private MongoClient *mongoClient*;

    private MongoDatabase *database*;

    public *MongoDBConnection*(String *uri*, String *dbName*) {

        mongoClient = *MongoClients*.*create*(uri);

        database = *mongoClient*.*getDatabase*(dbName);

    }

    public void *insertDocument*(String *collectionName*, Document *document*) {

        // Corrected line: Use MongoCollection<Document> instead of MongoDBConnection<Document>

        MongoCollection<Document> *collection* = *database*.*getCollection*(collectionName);

*collection*.*insertOne*(document);

*System*.*out*.*println*("Document inserted successfully");

    }

    public void *closeConnection*() {

*mongoClient*.*close*();

    }

    public static void *main*(String[] *args*) {

        String *uri* = "mongodb://localhost:27017"; // Replace with your MongoDB URI

        String *dbName* = "test"; // Replace with your database name

        MongoDBConnection *mongoDBConnection* = **new** *MongoDBConnection*(uri, dbName);

        Document *doc* = **new** *Document*("name", "John Doe")

                .*append*("age", 30)

                .*append*("city", "New York");

*mongoDBConnection*.*insertDocument*("users", doc);

*mongoDBConnection*.*closeConnection*();

    }

}

**MAIN.JAVA**

package com.example;

import javax.swing.*\**;

import javax.swing.table.DefaultTableModel;

import java.awt.*\**;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import com.mongodb.client.*\**;

import org.bson.Document;

public class Main extends JFrame implements ActionListener {

    private JTextField *usernameField*;

    private JPasswordField *passwordField*;

    private JButton *loginButton*, *signUpButton*;

    public *Main*() {

*setTitle*("Login Form");

*setSize*(400, 300);

*setDefaultCloseOperation*(*JFrame*.*EXIT\_ON\_CLOSE*);

*setLayout*(**new** *GridLayout*(4, 2));

        JLabel *usernameLabel* = **new** *JLabel*("Username:");

        usernameField = **new** *JTextField*();

        JLabel *passwordLabel* = **new** *JLabel*("Password:");

        passwordField = **new** *JPasswordField*();

        loginButton = **new** *JButton*("Login");

        signUpButton = **new** *JButton*("Sign Up");

*add*(usernameLabel);

*add*(usernameField);

*add*(passwordLabel);

*add*(passwordField);

*add*(loginButton);

*add*(signUpButton);

*loginButton*.*addActionListener*(*this*);

*signUpButton*.*addActionListener*(e -> {

**new** *SignUp*();

*dispose*();

        });

*setVisible*(true);

    }

    @Override

    public void *actionPerformed*(ActionEvent *e*) {

        String *username* = *usernameField*.*getText*();

        String *password* = **new** *String*(*passwordField*.*getPassword*());

**if** (*authenticateUser*(username, password)) {

*JOptionPane*.*showMessageDialog*(*this*, "Login Successful!");

**new** *Dashboard*();

*dispose*();

        } **else** {

*JOptionPane*.*showMessageDialog*(*this*, "Invalid credentials, please try again.");

        }

    }

    private boolean *authenticateUser*(String *username*, String *password*) {

        String *uri* = "mongodb://localhost:27017"; // Replace with your MongoDB URI

**try** (MongoClient *mongoClient* = *MongoClients*.*create*(uri)) {

            MongoDatabase *database* = *mongoClient*.*getDatabase*("user\_management");

            MongoCollection<Document> *collection* = *database*.*getCollection*("users");

            Document *user* = *collection*.*find*(**new** *Document*("username", username).*append*("password", password)).*first*();

**return** user != null;

        } **catch** (Exception *ex*) {

*ex*.*printStackTrace*();

*JOptionPane*.*showMessageDialog*(*this*, "Error connecting to the database.");

**return** false;

        }

    }

    public static void *main*(String[] *args*) {

**new** *Main*();

    }

    // Inner SignUp Class

    class SignUp extends JFrame implements ActionListener {

        private JTextField *usernameField*;

        private JPasswordField *passwordField*;

        private JButton *signUpButton*, *backButton*;

        public *SignUp*() {

*setTitle*("Sign Up Form");

*setSize*(400, 300);

*setDefaultCloseOperation*(*JFrame*.*EXIT\_ON\_CLOSE*);

*setLayout*(**new** *GridLayout*(4, 2));

            JLabel *usernameLabel* = **new** *JLabel*("Username:");

            usernameField = **new** *JTextField*();

            JLabel *passwordLabel* = **new** *JLabel*("Password:");

            passwordField = **new** *JPasswordField*();

            signUpButton = **new** *JButton*("Sign Up");

            backButton = **new** *JButton*("Back to Login");

*add*(usernameLabel);

*add*(usernameField);

*add*(passwordLabel);

*add*(passwordField);

*add*(signUpButton);

*add*(backButton);

*signUpButton*.*addActionListener*(*this*);

*backButton*.*addActionListener*(e -> {

**new** *Main*();

*dispose*();

            });

*setVisible*(true);

        }

        @Override

        public void *actionPerformed*(ActionEvent *e*) {

            String *username* = *usernameField*.*getText*();

            String *password* = **new** *String*(*passwordField*.*getPassword*());

**if** (*username*.*isEmpty*() || *password*.*isEmpty*()) {

*JOptionPane*.*showMessageDialog*(*this*, "Username and Password cannot be empty.");

**return**;

            }

            String *uri* = "mongodb://localhost:27017"; // Replace with your MongoDB URI

**try** (MongoClient *mongoClient* = *MongoClients*.*create*(uri)) {

                MongoDatabase *database* = *mongoClient*.*getDatabase*("user\_management");

                MongoCollection<Document> *collection* = *database*.*getCollection*("users");

                Document *existingUser* = *collection*.*find*(**new** *Document*("username", username)).*first*();

**if** (existingUser != null) {

*JOptionPane*.*showMessageDialog*(*this*, "Username already exists. Please choose another.");

**return**;

                }

                Document *newUser* = **new** *Document*("username", username).*append*("password", password);

*collection*.*insertOne*(newUser);

*JOptionPane*.*showMessageDialog*(*this*, "Sign-Up Successful! Redirecting to Login...");

**new** *Main*();

*dispose*();

            } **catch** (Exception *ex*) {

*ex*.*printStackTrace*();

*JOptionPane*.*showMessageDialog*(*this*, "Error occurred while signing up. Please try again.");

            }

        }

    }

    // Inner Dashboard Class

    class Dashboard extends JFrame {

        public *Dashboard*() {

*setTitle*("Dashboard");

*setSize*(400, 400);

*setDefaultCloseOperation*(*JFrame*.*EXIT\_ON\_CLOSE*);

*setLayout*(**new** *GridLayout*(2, 2));

            JButton *notebookButton* = **new** *JButton*("Notebook");

            JButton *diaryButton* = **new** *JButton*("Diary");

            JButton *birthdaysButton* = **new** *JButton*("Birthdays");

            JButton *remindersButton* = **new** *JButton*("Reminders");

*add*(notebookButton);

*add*(diaryButton);

*add*(birthdaysButton);

*add*(remindersButton);

*notebookButton*.*addActionListener*(e -> **new** *DataWindow*("notebook"));

*diaryButton*.*addActionListener*(e -> **new** *DataWindow*("diary"));

*birthdaysButton*.*addActionListener*(e -> **new** *DataWindow*("birthdays"));

*remindersButton*.*addActionListener*(e -> **new** *DataWindow*("reminders"));

*setVisible*(true);

        }

    }

    // Inner DataWindow Class

    class DataWindow extends JFrame {

        private final String *collectionName*;

        private final DefaultTableModel *tableModel*;

        private final JTable *table*;

        private JTextField *searchField*;

        private JButton *searchButton*;

        public *DataWindow*(String *collectionName*) {

*this*.*collectionName* = collectionName;

*setTitle*(*collectionName*.*substring*(0, 1).*toUpperCase*() + *collectionName*.*substring*(1));

*setSize*(600, 400);

*setDefaultCloseOperation*(*JFrame*.*DISPOSE\_ON\_CLOSE*);

            tableModel = **new** *DefaultTableModel*(**new** String[]{"ID", "Content"}, 0);

            table = **new** *JTable*(tableModel);

*loadData*("");

            JScrollPane *scrollPane* = **new** *JScrollPane*(table);

            // Search panel with text field and search button

            JPanel *searchPanel* = **new** *JPanel*();

            searchField = **new** *JTextField*(20);

            searchButton = **new** *JButton*("Search");

*searchButton*.*addActionListener*(e -> *searchData*());

*searchPanel*.*add*(**new** *JLabel*("Search: "));

*searchPanel*.*add*(searchField);

*searchPanel*.*add*(searchButton);

            JButton *saveButton* = **new** *JButton*("Save");

            JButton *editButton* = **new** *JButton*("Edit");

            JButton *deleteButton* = **new** *JButton*("Delete");

*saveButton*.*addActionListener*(*this***::**saveData);

*editButton*.*addActionListener*(*this***::**editData);

*deleteButton*.*addActionListener*(*this***::**deleteData);

            JPanel *buttonPanel* = **new** *JPanel*();

*buttonPanel*.*add*(saveButton);

*buttonPanel*.*add*(editButton);

*buttonPanel*.*add*(deleteButton);

*add*(searchPanel, *BorderLayout*.*NORTH*);

*add*(scrollPane, *BorderLayout*.*CENTER*);

*add*(buttonPanel, *BorderLayout*.*SOUTH*);

*setVisible*(true);

        }

        private void *loadData*(String *searchTerm*) {

            String *uri* = "mongodb://localhost:27017";

**try** (MongoClient *mongoClient* = *MongoClients*.*create*(uri)) {

                MongoDatabase *database* = *mongoClient*.*getDatabase*("user\_management");

                MongoCollection<Document> *collection* = *database*.*getCollection*(collectionName);

                // Adjust the query to filter based on the search term

                Document *query* = **new** *Document*();

**if** (!*searchTerm*.*isEmpty*()) {

*query*.*append*("content", **new** *Document*("$regex", searchTerm).*append*("$options", "i"));

                }

*tableModel*.*setRowCount*(0); // Clear existing rows

**for** (Document *doc* **:** *collection*.*find*(query)) {

*tableModel*.*addRow*(**new** Object[]{*doc*.*getObjectId*("\_id"), *doc*.*getString*("content")});

                }

                // Alert when the number of items exceeds the limit for Birthdays and Reminders

**if** (*collectionName*.*equals*("birthdays") || *collectionName*.*equals*("reminders")) {

                    long *count* = *collection*.*countDocuments*();

**if** (count >= 5) {

*JOptionPane*.*showMessageDialog*(*this*, "You have reached the limit of saved items in " + collectionName + ".");

                    }

                }

            } **catch** (Exception *e*) {

*e*.*printStackTrace*();

*JOptionPane*.*showMessageDialog*(*this*, "Error loading data.");

            }

        }

        private void *searchData*() {

            String *searchTerm* = *searchField*.*getText*();

*loadData*(searchTerm); // Reload data based on the search term

        }

        private void *saveData*(ActionEvent *e*) {

            String *content* = *JOptionPane*.*showInputDialog*(*this*, "Enter content:");

**if** (content == null || *content*.*trim*().*isEmpty*()) **return**;

            String *uri* = "mongodb://localhost:27017";

**try** (MongoClient *mongoClient* = *MongoClients*.*create*(uri)) {

                MongoDatabase *database* = *mongoClient*.*getDatabase*("user\_management");

                MongoCollection<Document> *collection* = *database*.*getCollection*(collectionName);

                Document *doc* = **new** *Document*("content", content);

*collection*.*insertOne*(doc);

*tableModel*.*addRow*(**new** Object[]{*doc*.*getObjectId*("\_id"), content});

                // Alert when the number of items exceeds the limit for Birthdays and Reminders

**if** (*collectionName*.*equals*("birthdays") || *collectionName*.*equals*("reminders")) {

                    long *count* = *collection*.*countDocuments*();

**if** (count >= 5) {

*JOptionPane*.*showMessageDialog*(*this*, "You have reached the limit of saved items in " + collectionName + ".");

                    }

                }

            } **catch** (Exception *ex*) {

*ex*.*printStackTrace*();

*JOptionPane*.*showMessageDialog*(*this*, "Error saving data.");

            }

        }

        private void *editData*(ActionEvent *e*) {

            int *selectedRow* = *table*.*getSelectedRow*();

**if** (selectedRow == -1) {

*JOptionPane*.*showMessageDialog*(*this*, "Please select a row to edit.");

**return**;

            }

            String *content* = *JOptionPane*.*showInputDialog*(*this*, "Edit content:", *table*.*getValueAt*(selectedRow, 1));

**if** (content == null || *content*.*trim*().*isEmpty*()) **return**;

            Object *id* = *table*.*getValueAt*(selectedRow, 0);

            String *uri* = "mongodb://localhost:27017";

**try** (MongoClient *mongoClient* = *MongoClients*.*create*(uri)) {

                MongoDatabase *database* = *mongoClient*.*getDatabase*("user\_management");

                MongoCollection<Document> *collection* = *database*.*getCollection*(collectionName);

*collection*.*updateOne*(**new** *Document*("\_id", id), **new** *Document*("$set", **new** *Document*("content", content)));

*tableModel*.*setValueAt*(content, selectedRow, 1);

            } **catch** (Exception *ex*) {

*ex*.*printStackTrace*();

*JOptionPane*.*showMessageDialog*(*this*, "Error editing data.");

            }

        }

        private void *deleteData*(ActionEvent *e*) {

            int *selectedRow* = *table*.*getSelectedRow*();

**if** (selectedRow == -1) {

*JOptionPane*.*showMessageDialog*(*this*, "Please select a row to delete.");

**return**;

            }

            Object *id* = *table*.*getValueAt*(selectedRow, 0);

            String *uri* = "mongodb://localhost:27017";

**try** (MongoClient *mongoClient* = *MongoClients*.*create*(uri)) {

                MongoDatabase *database* = *mongoClient*.*getDatabase*("user\_management");

                MongoCollection<Document> *collection* = *database*.*getCollection*(collectionName);

*collection*.*deleteOne*(**new** *Document*("\_id", id));

*tableModel*.*removeRow*(selectedRow);

            } **catch** (Exception *ex*) {

*ex*.*printStackTrace*();

*JOptionPane*.*showMessageDialog*(*this*, "Error deleting data.");

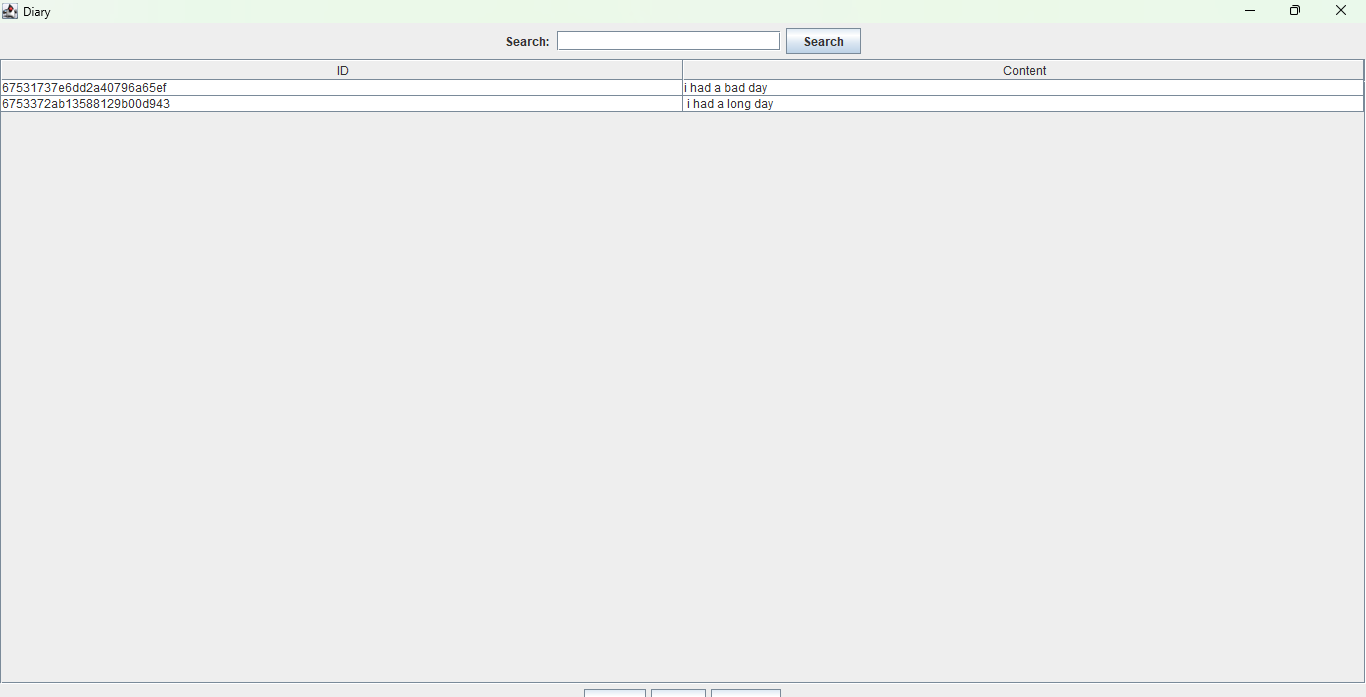
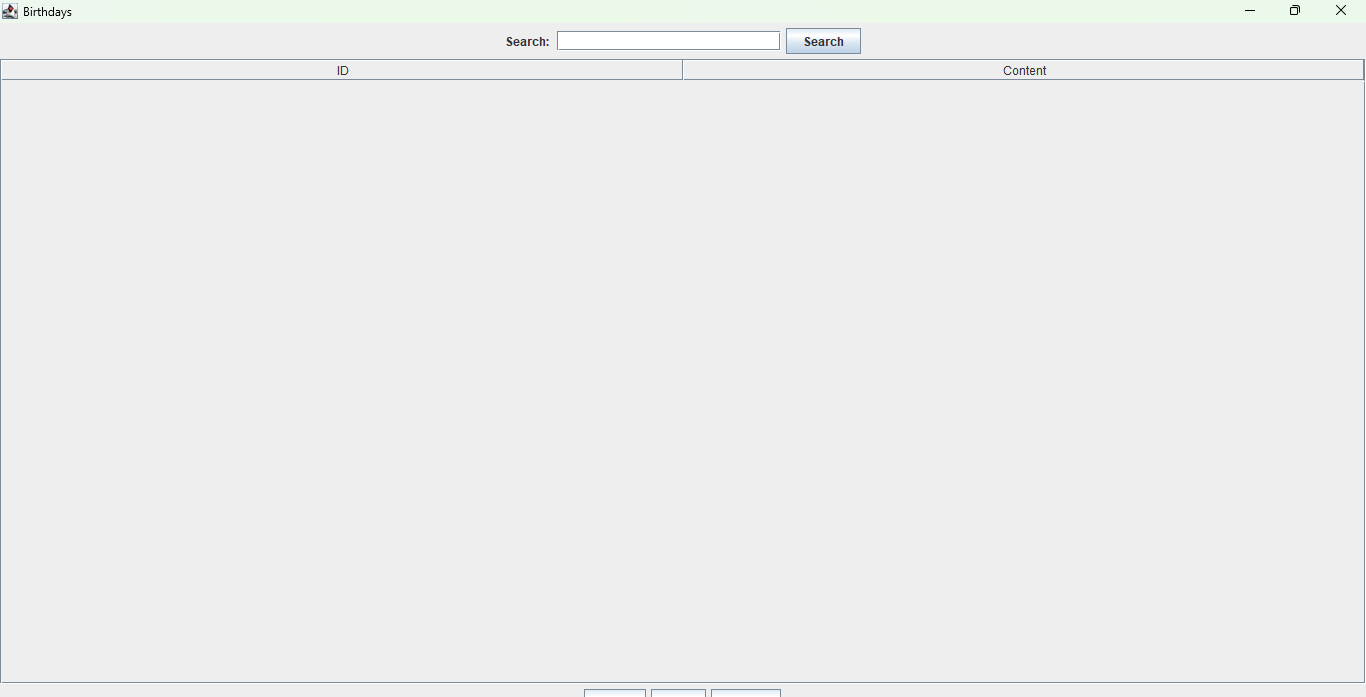
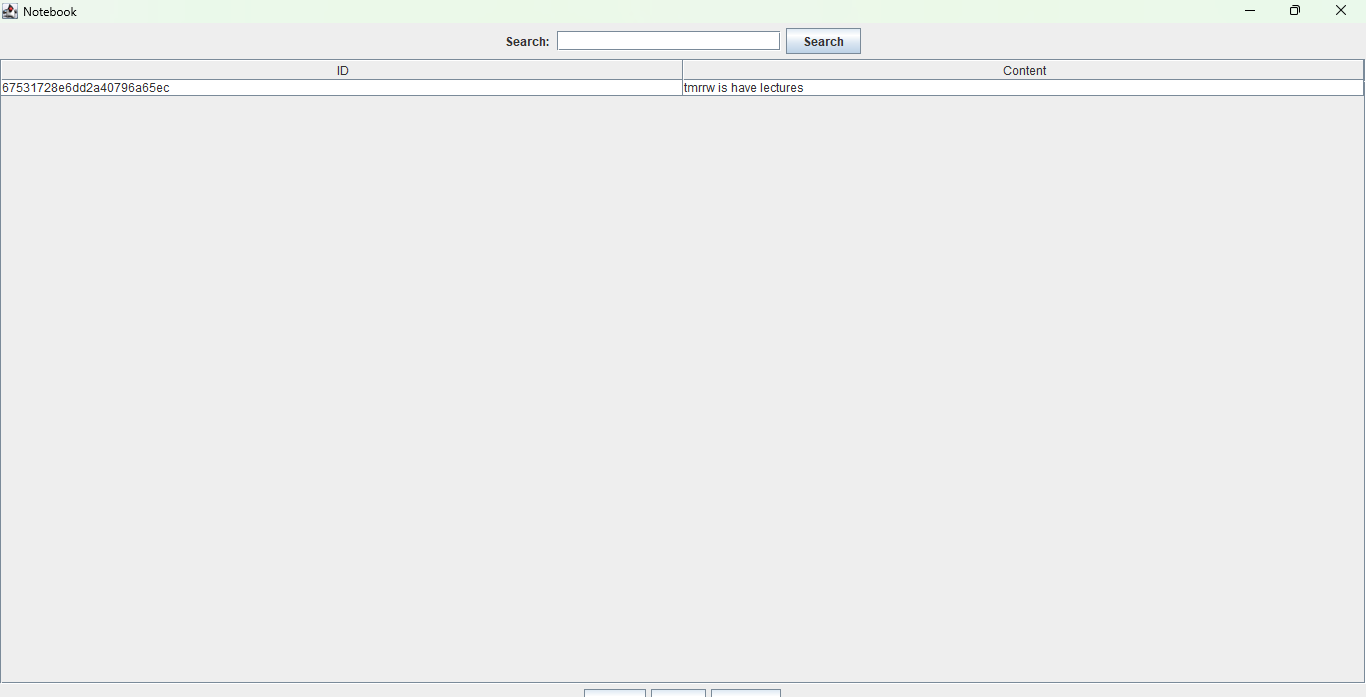
            }

        }

    }

}

**SCREENSHORTS OF THE INTERFENCE**

****