



CSE Department – Faculty of Engineering - MSA

Spring 2025

GSE122 GSE122i COM265 PROGRAMMING 2

Course Project

Course Instructor: Dr. Ahmed El Anany

Due Date 9/MAY/2025 11:59 PM on E-learning**Discussion inside lecture 18/May till 23/May inside lab as per lab slot**

Student Name	Hesham Amr	Student ID	244875
Student Name	Omar Morsy	Student ID	243617
Student Name	Malak Hany	Student ID	241559
Student Name	Yasmin Sayed	Student ID	244173
Student Name	Hassanien Ala	Student ID	244163
TA Name	Eng. Dina Magdy Eng. Youmna Mohamed Eng. Mohamed Khaled Eng. Hussien Mostafa	Grade:	/

Diary Management System



Table of Contents

Project Overview	3
Objectives	3
Roles and Responsibilities	4
Algorithm and external libraries	5
GUI and Database Usage	6
Code explaining	7
Output and results	8
GitHub(optional)	9
References	10



Project Overview

This section describes the objectives , tools , responsibilities , technical concepts.

The Personal Diary Management System is a Java-based desktop application that enables users to securely manage personal diary entries. It supports user registration, login, password change, and CRUD operations (Create, Read, Update, Delete) for diary entries. Built using Java Swing for the GUI and MySQL for backend storage, the system ensures a seamless and responsive user experience. The project aims to enhance skills in object-oriented programming, GUI development, and database integration.



Objectives

This section describes the objectives of the project .

- ☐ Develop a secure and user-friendly application for managing diary tasks.
- ☐ Enable multi-user support with authentication mechanisms.
- ☐ Implement robust error handling for all operations.
- ☐ Provide GUI functionality for ease of use.
- ☐ Integrate MySQL database for persistent data storage.
- ☐ Use object-oriented programming concepts to build modular, maintainable code.



Roles and Responsibilities

This section describes the roles of each team member.

Report: Hesham Amr

Powerpoint: Hassanien Ala

Code & GitHub Management : Malak Hany

GUI code & Screenshots: Omar Morsy

Database Design, Code & References : Yasmin Sayed



Algorithm and external libraries

This section describes the algorithm and external libraries used in the project .

Must include a detailed description of them .

Algorithm Used:

- **Login/Registration:** Upon entering credentials, the system checks user existence in the MySQL database using SQL queries.
- **CRUD Operations:** Each diary task is associated with a user. Tasks are retrieved, added, edited, or deleted via SQL statements, and displayed in a GUI table.
- **Password Update:** Allows users to update their stored password with appropriate validation.

External Libraries:

- **Java Swing:** Used for creating the GUI elements (e.g., JFrame, JTextField, JTable).
- **JDBC:** Used for database connectivity to execute SQL commands between Java and MySQL.
- **MySQL:** The backend relational database used to store user credentials and diary data.



GUI and Database Usage

This section describes the GUI and Database Usage in the project .

Must include a detailed description of them and layout of the designed GUI and DataBase tables.

GUI:

- Built entirely using Java Swing.
- Pages include: Login, Registration, Main Menu (Add, Edit, Delete, View Entries), Change Password.
- Features color-coded buttons and validation prompts.

Database:

- **MySQL Tables:**
 - user(id, username, password)
 - diary(id, name, duration, address, date, time, details, user_id)
- Relationships:
 - One-to-many between user and diary (one user can have many diaries).

Sample GUI Layout:

- Login Page → Text fields and buttons for login/register.
- Main Menu → Form fields for diary data entry, JTable for diary listing, and control buttons.



Code explaining

The system is a personal diary management app where users can:

- Register and log in
- Add, edit, delete, and view personal diary entries
- Update their password

It uses:

- **Java Swing** for GUI
- **MySQL** for data storage
- **JDBC** for database connection

```
import java.awt.event.MouseAdapter; // Import the MouseAdapter class for handling mouse events
import java.awt.event.MouseEvent; // Import the MouseEvent class for identifying mouse actions
import java.sql.*; // Import all SQL classes to manage database connections, statements, and queries
import java.util.ArrayList; // Import ArrayList for dynamically storing objects like diary entries
import java.util.List; // Import List interface for abstraction over different list implementations
import javax.swing.*; // Import Swing components (GUI elements like JFrame, JPanel, JButton, etc.)
import javax.swing.table.DefaultTableModel; // Import the table model used for displaying entries in tables
import java.awt.*; // Import AWT classes for additional GUI support like layout and colors
```

The **DBConnection** class is responsible for establishing a connection to the MySQL database. It defines three constant variables: the database URL, the database user name (root), and an empty password. These constants are used by the `getConnection()` method to return a `Connection` object from the `JDBC DriverManager`. This approach simplifies the connection process because other classes don't need to repeat the database details—they just call this method when they need to execute SQL queries. This separation also makes the system more maintainable and secure by centralizing the sensitive configuration in one place.

```
// ----- DATABASE CONNECTION CLASS -----
// This class handles all operations related to database connectivity.
// It centralizes the configuration of DB URL, credentials, and provides a reusable method to connect.
class DBConnection {
    // URL for connecting to the MySQL database. It includes:
    // - Host: localhost
    // - Port: 3306 (default MySQL port)
    // - Database name: DiarySystem
    // - serverTimezone=UTC: resolves time zone errors between Java and MySQL
    private static final String URL = "jdbc:mysql://localhost:3306/DiarySystem?serverTimezone=UTC";
```




```
// Username to authenticate with the MySQL database
```

```
private static final String USER = "root";
```

```
// Password associated with the USER (empty string here for local testing)
```

```
private static final String PASS = "";
```

```
// This method returns an active Connection object to interact with the database.
```

```
// If connection fails, it throws an SQLException.
```

```
public static Connection getConnection() throws SQLException {  
    return DriverManager.getConnection(URL, USER, PASS); // Attempt to open DB connection  
}  
}
```

The **PasswordManager** class is used to handle user passwords in a flexible and modular way. Instead of storing a plain string directly in the User class, the password is wrapped in this class, allowing you to modify the way passwords are handled (such as adding hashing or encryption) without changing the structure of the User class itself. The PasswordManager has a constructor to initialize the password and provides getter and setter methods. This encapsulation is useful because it keeps the responsibility of managing passwords in one place and avoids mixing password logic with other unrelated user attributes.

```
// ----- PASSWORD MANAGER CLASS -----
```

```
// This class is designed to handle password logic separately for flexibility and future upgrades
```

```
// like encryption, validation, or rules enforcement.
```

```
class PasswordManager {  
    private String password; // Field to store the user's password
```

```
// Constructor that initializes the password field when object is created
```

```
public PasswordManager(String password) {  
    this.password = password;  
}
```

```
// Getter method that returns the current password value
```

```
public String getPassword() {  
    return password;  
}
```

```
// Setter method to update/change the password value
```

```
public void setPassword(String password) {  
    this.password = password;  
}  
}
```

The **User** class models each registered user of the system. It has three attributes: an ID (from the database), a username, and a PasswordManager object. These attributes are declared as final to make them immutable, except for the password, which can be updated through the PasswordManager. The class includes getters for all attributes and a



setter for the password, allowing the system to update the password securely. This class plays a key role during login, registration, and password change operations, and acts as the user's profile throughout the session.

// ----- USER CLASS -----

// This class models a user entity in the diary system.

// It contains user-related information such as ID, username, and password (via PasswordManager).

```
class User {
    private final int id; // Unique integer ID assigned to the user by the database
    private final String username; // Username of the user used during login and registration
    private final PasswordManager passwordManager; // Object that manages password operations

    // Constructor that sets all user fields upon creation
    public User(int id, String username, String password) {
        this.id = id; // Assign user ID
        this.username = username; // Assign username
        this.passwordManager = new PasswordManager(password); // Initialize password manager with provided
password
    }

    // Getter for user ID
    public int getId() {
        return id;
    }

    // Getter for username
    public String getUsername() {
        return username;
    }

    // Getter that delegates to PasswordManager to retrieve the stored password
    public String getPassword() {
        return passwordManager.getPassword();
    }

    // Setter that delegates to PasswordManager to update the stored password
    public void setPassword(String password) {
        this.passwordManager.setPassword(password);
    }
}
```

The **Diary** class represents each diary entry made by a user. It stores information like the diary entry's ID, name, duration, address, date, time, details, and the user ID of the owner. This design supports the one-to-many relationship in the database where a single user can have multiple diary entries. Like the User class, it includes a constructor for setting all fields and getter/setter methods to retrieve and update entry details. This class is used heavily in displaying diary entries in the table and performing create, edit, or delete operations on specific entries.

// ----- DIARY ENTRY CLASS -----



// This class models a diary entry belonging to a user.

// It contains information such as task name, duration, location, date, and more.

```
class Diary {
```

```
    private final int id; // Unique ID of the diary entry assigned by the database
    private String name; // Name or title of the diary task (e.g., "Study Session")
    private String duration; // Duration of the task (e.g., "2 hours")
    private String address; // Location where the task took place (optional)
    private String date; // Date of the diary task in YYYY-MM-DD format
    private String time; // Time of the diary task in HH:MM:SS format
    private String details; // Additional notes or description provided by the user
    private final int userId; // ID of the user who owns this diary entry
```

// Constructor initializes all diary entry fields

```
public Diary(int id, String name, String duration, String address, String date, String time, String details, int userId) {
    this.id = id; // Assign diary ID
    this.name = name; // Set task name
    this.duration = duration; // Set duration of task
    this.address = address; // Set address or location of task
    this.date = date; // Set task date
    this.time = time; // Set task time
    this.details = details; // Set additional details/description
    this.userId = userId; // Associate diary with a specific user ID
}
```

// Getters - Retrieve values of diary fields

```
public int getId() { return id; }
public String getName() { return name; }
public String getDuration() { return duration; }
public String getAddress() { return address; }
public String getDate() { return date; }
public String getTime() { return time; }
public String getDetails() { return details; }
public int getUserId() { return userId; }
```

// Setters - Update values of diary fields (except id and userId which are final)

```
public void setName(String name) { this.name = name; }
public void setDuration(String duration) { this.duration = duration; }
public void setAddress(String address) { this.address = address; }
public void setDate(String date) { this.date = date; }
public void setTime(String time) { this.time = time; }
public void setDetails(String details) { this.details = details; }
```

```
}
```

The **DatabaseHandler** class handles all interactions with the MySQL database. It includes methods for retrieving a user by username, adding a new user, updating passwords, retrieving all diaries for a user, and performing CRUD



operations (Create, Read, Update, Delete) on diary entries. Each method establishes a new connection using the DBConnection class and executes a prepared SQL statement. It uses try-with-resources blocks to ensure that connections and statements are automatically closed, avoiding memory leaks. This class keeps all SQL operations centralized, making the code cleaner and easier to manage when changes are needed.

```
// ----- DATABASE HANDLER CLASS -----
```

```
// This class serves as a bridge between the application and the database.
```

```
// It provides methods to perform all CRUD operations (Create, Read, Update, Delete) on users and diary entries.
```

```
class DatabaseHandler {
```

```
    // Method to fetch a user from the database by their username
```

```
    // Used during login and to check for existing usernames during registration
```

```
    public User getUserByUsername(String username) throws SQLException {
```

```
        String sql = "SELECT * FROM user WHERE username=?"; // SQL statement to fetch a user record by username
```

```
        // Try-with-resources ensures automatic closing of connection and statement
```

```
        try (Connection conn = DBConnection.getConnection(); // Establish DB connection
```

```
            PreparedStatement stmt = conn.prepareStatement(sql)) { // Prepare statement to avoid SQL injection
```

```
            stmt.setString(1, username); // Bind the username to the query parameter
```

```
            ResultSet rs = stmt.executeQuery(); // Execute query and get result set
```

```
            if (rs.next()) { // If a matching user is found
```

```
                return new User(
```

```
                    rs.getInt("id"), // Extract ID from result set
```

```
                    rs.getString("username"), // Extract username
```

```
                    rs.getString("password") // Extract password
```

```
                );
```

```
            }
```

```
        }
```

```
        return null; // No user found with given username
```

```
    }
```

```
    // Method to add/register a new user to the database
```

```
    public void addUser(String username, String password) throws SQLException {
```

```
        String sql = "INSERT INTO user (username, password) VALUES (?, ?)"; // SQL statement to insert new user
```

```
        try (Connection conn = DBConnection.getConnection();
```

```
            PreparedStatement stmt = conn.prepareStatement(sql)) {
```

```
            stmt.setString(1, username); // Set username
```

```
            stmt.setString(2, password); // Set password
```

```
            stmt.executeUpdate(); // Execute update query to insert new record
```

```
        }
```

```
    }
```



// Method to update a user's password in the database

```
public void updatePassword(String username, String password) throws SQLException {  
    String sql = "UPDATE user SET password=? WHERE username=?"; // SQL statement to update password  
  
    try (Connection conn = DBConnection.getConnection();  
        PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
        stmt.setString(1, password); // Set new password  
        stmt.setString(2, username); // Identify which user to update using the username  
        stmt.executeUpdate(); // Execute the update  
    }  
}
```

// Method to retrieve all diary entries for a given user

```
public List<Diary> getUserDiaries(int userId) throws SQLException {  
    List<Diary> list = new ArrayList<>(); // List to hold all diary entries  
    String sql = "SELECT * FROM diary WHERE user_id=?"; // SQL to get all diary records for a user  
  
    try (Connection conn = DBConnection.getConnection();  
        PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
        stmt.setInt(1, userId); // Set user ID to filter diaries  
        ResultSet rs = stmt.executeQuery(); // Execute query  
  
        while (rs.next()) { // Iterate over result set  
            list.add(new Diary(  
                rs.getInt("id"),  
                rs.getString("name"),  
                rs.getString("duration"),  
                rs.getString("address"),  
                rs.getString("date"),  
                rs.getString("time"),  
                rs.getString("details"),  
                rs.getInt("user_id")  
            ));  
        }  
    }  
    return list; // Return list of diary entries  
}
```

// Method to add a new diary entry into the database

```
public void addDiary(Diary d) throws SQLException {
```



```
String sql = "INSERT INTO diary (name, duration, address, date, time, details, user_id) VALUES (?, ?, ?, ?, ?, ?, ?)";
```

```
try (Connection conn = DBConnection.getConnection();
    PreparedStatement stmt = conn.prepareStatement(sql)) {

    stmt.setString(1, d.getName());
    stmt.setString(2, d.getDuration());
    stmt.setString(3, d.getAddress());
    stmt.setString(4, d.getDate());
    stmt.setString(5, d.getTime());
    stmt.setString(6, d.getDetails());
    stmt.setInt(7, d.getUserId());
    stmt.executeUpdate(); // Execute insert operation
}
}
```

// Method to update an existing diary entry

```
public void updateDiary(Diary d) throws SQLException {
    String sql = "UPDATE diary SET name=?, duration=?, address=?, date=?, time=?, details=? WHERE id=?";

    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, d.getName());
        stmt.setString(2, d.getDuration());
        stmt.setString(3, d.getAddress());
        stmt.setString(4, d.getDate());
        stmt.setString(5, d.getTime());
        stmt.setString(6, d.getDetails());
        stmt.setInt(7, d.getId());
        stmt.executeUpdate(); // Execute update query
    }
}
```

// Method to delete a diary entry by its ID

```
public void deleteDiary(int id) throws SQLException {
    String sql = "DELETE FROM diary WHERE id=?"; // SQL statement to delete a diary by ID

    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, id); // Specify ID of the diary to delete
        stmt.executeUpdate(); // Execute delete operation
    }
}
```



```
}  
}  
}
```

The **LoginManager** class manages all user authentication features like logging in, registering a new account, and changing a password. It relies on the DatabaseHandler to perform these actions. For login, it retrieves the user and checks if the entered password matches the one in the database. For registration, it ensures the username doesn't already exist before adding a new record. And for changing the password, it calls the DatabaseHandler's method to update the password field. This class acts as a middle layer between the user interface and the database logic, focusing specifically on account security.

```
// ----- LOGIN MANAGER CLASS -----
```

```
// This class manages user authentication: logging in, registering a new user, and changing passwords.
```

```
class LoginManager {
```

```
    // Composition: LoginManager uses an instance of DatabaseHandler to perform DB operations
```

```
    private final DatabaseHandler dbHandler = new DatabaseHandler();
```

```
    // Method to authenticate user login
```

```
    public User login(String username, String password) throws Exception {
```

```
        // Call DatabaseHandler to retrieve a user by their username
```

```
        User user = dbHandler.getUserByUsername(username);
```

```
        // If the user exists and the password matches, return the user object
```

```
        if (user != null && user.getPassword().equals(password)) {
```

```
            return user;
```

```
        } else {
```

```
            // If no match, return null to indicate login failure
```

```
            return null;
```

```
        }
```

```
    }
```

```
    // Method to register a new user
```

```
    public boolean register(String username, String password) throws Exception {
```

```
        // Check if the username is already taken
```

```
        if (dbHandler.getUserByUsername(username) != null) {
```

```
            return false; // Username exists - cannot register again
```

```
        } else {
```

```
            // If username is free, add new user to database
```

```
            dbHandler.addUser(username, password);
```

```
            return true;
```

```
        }
```

```
    }
```




```
// Method to change an existing user's password
```

```
public void changePassword(String username, String newPassword) throws Exception {  
    dbHandler.updatePassword(username, newPassword); // Call DatabaseHandler to update password  
}  
}
```

The **RecordManager** class deals with managing diary entries for users. It provides four main functions: viewing all entries, adding a new entry, updating an existing entry, and deleting a selected entry. Like the LoginManager, it uses DatabaseHandler internally but focuses on diary data rather than user accounts. This separation of concerns keeps the code organized and modular. The RecordManager is primarily used by the GUI to manage records based on user actions such as clicking “Add”, “Edit”, or “Delete”.

```
// ----- RECORD MANAGER CLASS -----
```

```
// This class manages diary records for a user: viewing, adding, editing, deleting  
class RecordManager {
```

```
    // Uses an instance of DatabaseHandler to interact with the database  
    private final DatabaseHandler dbHandler = new DatabaseHandler();
```

```
    // Retrieve all diary entries for a specific user
```

```
    public List<Diary> viewRecord(int userId) throws Exception {  
        return dbHandler.getUserDiaries(userId); // Delegate to DatabaseHandler  
    }
```

```
    // Add a new diary entry to the database
```

```
    public void addRecord(Diary d) throws Exception {  
        dbHandler.addDiary(d); // Use DatabaseHandler to add the diary  
    }
```

```
    // Update an existing diary record
```

```
    public void updateRecord(Diary d) throws Exception {  
        dbHandler.updateDiary(d); // Use DatabaseHandler to update the diary  
    }
```

```
    // Delete a diary record by its ID
```

```
    public void deleteRecord(int id) throws Exception {  
        dbHandler.deleteDiary(id); // Use DatabaseHandler to delete the diary  
    }  
}
```

The **LoginManager** class handles all user account operations such as login, registration, and password changes. It uses an internal DatabaseHandler object to communicate with the database. The login() method checks if the username exists and if the entered password matches the stored one, returning a User object if successful. The register() method prevents duplicate accounts by checking if the username already exists before adding a new user. The



changePassword() method updates the user's password in the database. This class keeps all authentication logic in one place, making it easy to manage and extend in the future.

```
// ----- LOGIN MANAGER CLASS -----
```

```
// This class manages user authentication: logging in, registering a new user, and changing passwords.
```

```
class LoginManager {
```

```
    // Composition: LoginManager uses an instance of DatabaseHandler to perform DB operations
```

```
    private final DatabaseHandler dbHandler = new DatabaseHandler();
```

```
    // Method to authenticate user login
```

```
    public User login(String username, String password) throws Exception {
```

```
        // Call DatabaseHandler to retrieve a user by their username
```

```
        User user = dbHandler.getUserByUsername(username);
```

```
        // If the user exists and the password matches, return the user object
```

```
        if (user != null && user.getPassword().equals(password)) {
```

```
            return user;
```

```
        } else {
```

```
            // If no match, return null to indicate login failure
```

```
            return null;
```

```
        }
```

```
    }
```

```
    // Method to register a new user
```

```
    public boolean register(String username, String password) throws Exception {
```

```
        // Check if the username is already taken
```

```
        if (dbHandler.getUserByUsername(username) != null) {
```

```
            return false; // Username exists - cannot register again
```

```
        } else {
```

```
            // If username is free, add new user to database
```

```
            dbHandler.addUser(username, password);
```

```
            return true;
```

```
        }
```

```
    }
```

```
    // Method to change an existing user's password
```

```
    public void changePassword(String username, String newPassword) throws Exception {
```

```
        dbHandler.updatePassword(username, newPassword); // Call DatabaseHandler to update password
```

```
    }
```

```
}
```

The **RecordManager** class is the diary-entry controller. Holding a single DatabaseHandler instance, it exposes four straightforward methods: viewRecord(int userId) retrieves all diary entries that belong to a specific user; addRecord(Diary d) inserts a new entry; updateRecord(Diary d) edits an existing one; and deleteRecord(int id) removes an entry by its database ID. Each call simply forwards the request to the corresponding CRUD method in



DatabaseHandler, so no SQL ever appears in the GUI layer. By centralizing diary logic here, the system cleanly separates “what to do” (GUI buttons) from “how it’s stored” (SQL operations), simplifying maintenance and future upgrades.

```
// ----- RECORD MANAGER CLASS -----  
// This class manages diary records for a user: viewing, adding, editing, deleting  
class RecordManager {  
  
    // Uses an instance of DatabaseHandler to interact with the database  
    private final DatabaseHandler dbHandler = new DatabaseHandler();  
  
    // Retrieve all diary entries for a specific user  
    public List<Diary> viewRecord(int userId) throws Exception {  
        return dbHandler.getUserDiaries(userId); // Delegate to DatabaseHandler  
    }  
  
    // Add a new diary entry to the database  
    public void addRecord(Diary d) throws Exception {  
        dbHandler.addDiary(d); // Use DatabaseHandler to add the diary  
    }  
  
    // Update an existing diary record  
    public void updateRecord(Diary d) throws Exception {  
        dbHandler.updateDiary(d); // Use DatabaseHandler to update the diary  
    }  
  
    // Delete a diary record by its ID  
    public void deleteRecord(int id) throws Exception {  
        dbHandler.deleteDiary(id); // Use DatabaseHandler to delete the diary  
    }  
}
```

The **DiaryGUI** class handles the entire graphical user interface of the application using Java Swing. When the program starts, it shows a login screen implemented through the inner LoginFrame class. Users can enter their credentials or register. If login is successful, the user is directed to the MainMenuFrame, which displays form fields for diary entry and a table for listing all added entries. The interface includes buttons to add, edit, delete entries, or change the password. The GUI components respond to user actions with listeners and update the display accordingly. The GUI also ensures validation of input fields, such as making sure task name, date, and time are not left empty before saving an entry.

```
// ----- DIARY GUI ENTRY POINT CLASS -----  
// This class initializes the GUI of the diary system and controls the user interface flow.  
class DiaryGUI {  
  
    // Create instances of login and record managers to handle authentication and diary operations
```



```
private final LoginManager loginManager = new LoginManager();  
private final RecordManager recordManager = new RecordManager();
```

```
// Holds the current logged-in user object
```

```
private User currentUser;
```

```
// Method to show the login screen when the app starts
```

```
public void showLogin() {  
    new LoginFrame(); // Create and display the login window  
}
```

LoginFrame, an inner class of DiaryGUI, builds the application's first screen—the login and registration window—entirely with Java Swing. Its constructor sets up a small JFrame, arranges username and password fields in a 3 × 2 grid, and adds **Login** and **Register** buttons. Event listeners on those buttons call private helpers doLogin() and doRegister(), which validate that both fields are non-empty, then delegate the actual authentication or account-creation work to the outer class's LoginManager. Successful login stores the returned User in currentUser, shows a welcome dialog, disposes of the login window, and launches the MainMenuFrame; failures trigger an error message. By packaging all UI widgets, validation, and button logic in a compact inner class, LoginFrame keeps the login workflow self-contained and prevents it from cluttering other parts of the GUI code.

```
// ----- INNER CLASS: Login Frame -----
```

```
// This class builds the login and registration screen GUI
```

```
class LoginFrame extends JFrame {
```

```
    // UI components for username and password input fields
```

```
    JTextField usernameField = new JTextField(15);
```

```
    JPasswordField passwordField = new JPasswordField(15);
```

```
    // Constructor initializes the login window layout
```

```
    public LoginFrame() {  
        setTitle("Login - Diary System"); // Set title of the login window  
        setDefaultCloseOperation(EXIT_ON_CLOSE); // Close app on window close  
        setSize(350, 200); // Set window size  
        setLocationRelativeTo(null); // Center window on the screen
```

```
    // Create panel with grid layout for form fields and buttons
```

```
    JPanel panel = new JPanel(new GridLayout(3, 2, 5, 5));
```

```
    // Add UI labels and fields to the panel
```

```
    panel.add(new JLabel("Username:"));
```

```
    panel.add(usernameField);
```

```
    panel.add(new JLabel("Password:"));
```

```
    panel.add(passwordField);
```

```
    // Create login and register buttons
```

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



```
JButton regButton = new JButton("Register");

// Add buttons to panel
panel.add(loginButton);
panel.add(regButton);

// Add panel to the JFrame
add(panel);

// Add action listeners to buttons to trigger login and registration logic
loginButton.addActionListener(e -> doLogin());
regButton.addActionListener(e -> doRegister());

setVisible(true); // Show the login window
}

// Helper method to validate that both fields are not empty
private boolean validateLoginFields(String username, String password) {
    // If either field is empty, show an error message and return false
    if (username == null || username.trim().isEmpty() || password == null || password.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "Username and password cannot be empty.");
        return false;
    } else {
        return true; // Fields are valid
    }
}

// Logic for login button click
private void doLogin() {
    String username = usernameField.getText().trim(); // Get input username
    String password = new String(passwordField.getPassword()); // Get password securely

    // Validate inputs
    if (!validateLoginFields(username, password)) return;

    try {
        User u = loginManager.login(username, password); // Try to log in
        if (u != null) {
            currentUser = u; // Set logged-in user
            JOptionPane.showMessageDialog(this, "Welcome, " + username + "!");
            dispose(); // Close login window
            new MainMenuFrame(); // Open main menu window
        } else {
            JOptionPane.showMessageDialog(this, "Login failed! Wrong credentials.");
        }
    }
}
```



```
}  
} catch (Exception ex) {  
    JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());  
}  
}  
  
// Logic for registration button click  
private void doRegister() {  
    String username = usernameField.getText().trim();  
    String password = new String(passwordField.getPassword());  
  
    // Validate inputs  
    if (!validateLoginFields(username, password)) return;  
  
    try {  
        if (loginManager.register(username, password)) {  
            JOptionPane.showMessageDialog(this, "Registration successful! Login now.");  
        } else {  
            JOptionPane.showMessageDialog(this, "Username already exists.");  
        }  
    } catch (Exception ex) {  
        JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());  
    }  
}
```

The **Main** class is the entry point of the application. It includes a `main()` method that sets the GUI look and feel to match the system's default (for a more native look) and launches the application by showing the login window. It uses `SwingUtilities.invokeLater()` to ensure that the GUI is started on the correct thread. This class is intentionally kept small to follow best practices—its only role is to start the application cleanly and reliably.

// ----- MAIN MENU FRAME CLASS -----

// This class represents the main user interface frame shown after a user logs in.

// It allows the user to manage diary entries: add, edit, delete, view, and also change their password.

// The diary entries are displayed in a table, and input fields are provided for entry details.

class MainMenuFrame extends JFrame {

// Input fields for diary entry details

private final JTextField tfTaskName = new JTextField(); // Task name or title input

private final JTextField tfAddress = new JTextField(); // Location/address input

private final JTextField tfDuration = new JTextField(); // Duration of task input

private final JTextField tfDate = new JTextField(); // Date of task input, format: YYYY-MM-DD

private final JTextField tfTime = new JTextField(); // Time of task input, format: HH:MM:SS

// Multi-line text area for additional details about the task



```
private final JTextArea taDetails = new JTextArea(3, 20);
```

```
// Table to display diary entries in tabular form
```

```
private final JTable table;
```

```
private final DefaultTableModel tableModel; // Model backing the table data
```

```
// Action buttons for user commands
```

```
private final JButton btnAdd = new JButton("Add"); // Add new entry
```

```
private final JButton btnEdit = new JButton("Edit"); // Edit selected entry
```

```
private final JButton btnDelete = new JButton("Delete"); // Delete selected entry
```

```
private final JButton btnLogout = new JButton("Logout"); // Logout from the app
```

```
private final JButton btnPwd = new JButton("Change Password"); // Change user password
```

```
// List of diary entries currently loaded from the database for display and manipulation
```

```
private List<Diary> loadedDiaries;
```

```
// Constructor to initialize the GUI components and layout
```

```
public MainMenuFrame() {
```

```
    setTitle("Diary Management System - User: " + currentUser.getUsername()); // Show current username in title bar
```

```
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Exit app on close
```

```
    setSize(900, 600); // Set window size
```

```
    setLocationRelativeTo(null); // Center window on screen
```

```
// Create a panel with grid layout to hold form input labels and fields
```

```
JPanel formPanel = new JPanel(new GridLayout(6, 2, 5, 5));
```

```
formPanel.add(new JLabel("Task Name:")); formPanel.add(tfTaskName);
```

```
formPanel.add(new JLabel("Address:")); formPanel.add(tfAddress);
```

```
formPanel.add(new JLabel("Duration:")); formPanel.add(tfDuration);
```

```
formPanel.add(new JLabel("Date (YYYY-MM-DD):")); formPanel.add(tfDate);
```

```
formPanel.add(new JLabel("Time (HH:MM:SS):")); formPanel.add(tfTime);
```

```
formPanel.add(new JLabel("Details:")); formPanel.add(new JScrollPane(taDetails)); // Scroll pane for multiline text area
```

```
// Apply custom styling to buttons (colors, flat look)
```

```
styleButtons();
```

```
// Panel to contain the buttons horizontally
```

```
JPanel buttonPanel = new JPanel();
```

```
buttonPanel.add(btnAdd);
```

```
buttonPanel.add(btnEdit);
```

```
buttonPanel.add(btnDelete);
```

```
buttonPanel.add(btnPwd);
```

```
buttonPanel.add(btnLogout);
```



```
// Column headers for the table representing diary entry attributes
String[] columns = {"Task Name", "Address", "Duration", "Date", "Time", "Details"};

// Initialize the table model with column headers and no editable cells
tableModel = new DefaultTableModel(columns, 0) {
    public boolean isCellEditable(int row, int col) {
        return false; // Disallow direct editing inside the table cells
    }
};

// Create the table with the above model and allow only one row selection at a time
table = new JTable(tableModel);
table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
JScrollPane tableScroll = new JScrollPane(table); // Add scrolling to the table

// Top panel combines form inputs and buttons vertically
JPanel topPanel = new JPanel(new BorderLayout());
topPanel.add(formPanel, BorderLayout.CENTER); // Form inputs in center
topPanel.add(buttonPanel, BorderLayout.SOUTH); // Buttons below the form

// Main frame uses BorderLayout: form+buttons on top, table filling the rest
setLayout(new BorderLayout(10, 10));
add(topPanel, BorderLayout.NORTH);
add(tableScroll, BorderLayout.CENTER);

// Set button action listeners to respond to user clicks
btnAdd.addActionListener(e -> addEntry());
btnEdit.addActionListener(e -> editEntry());
btnDelete.addActionListener(e -> deleteEntry());
btnLogout.addActionListener(e -> logout());
btnPwd.addActionListener(e -> changePassword());

// When a table row is clicked, load that diary entry's data into input fields for easy editing
table.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        int row = table.getSelectedRow();
        // Validate row selection and that diaries are loaded
        if (row >= 0 && loadedDiaries != null && row < loadedDiaries.size()) {
            Diary d = loadedDiaries.get(row);
            // Populate input fields with selected diary entry details
            tfTaskName.setText(d.getName());
            tfAddress.setText(d.getAddress());
            tfDuration.setText(d.getDuration());
        }
    }
});
```




```
tfDate.setText(d.getDate());
tfTime.setText(d.getTime());
taDetails.setText(d.getDetails());
    }
}
});
```

```
loadEntries(); // Load diary entries from DB for the current user into the table
setVisible(true); // Make the window visible
}
```

// Apply custom colors and styles to all action buttons for better UX

```
private void styleButtons() {
    JButton[] buttons = {btnAdd, btnEdit, btnDelete, btnPwd, btnLogout};
    Color[] colors = {
        new Color(76, 175, 80), // Green for Add
        new Color(33, 150, 243), // Blue for Edit
        new Color(244, 67, 54), // Red for Delete
        new Color(255, 193, 7), // Yellow for Change Password
        new Color(158, 158, 158) // Grey for Logout
    };

    for (int i = 0; i < buttons.length; i++) {
        buttons[i].setOpaque(true);
        buttons[i].setBackground(colors[i]);
        buttons[i].setForeground(i == 3 ? Color.BLACK : Color.WHITE); // Black text on yellow button
        buttons[i].setFocusPainted(false);
        buttons[i].setBorderPainted(false);
    }
}
```

// Fetch user's diary entries from the database and populate the table

```
private void loadEntries() {
    try {
        loadedDiaries = recordManager.viewRecord(currentUser.getId());
        tableModel.setRowCount(0); // Clear existing rows before adding fresh data
        // Add each diary entry as a new row in the table
        for (Diary d : loadedDiaries) {
            tableModel.addRow(new Object[]{
                d.getName(), d.getAddress(), d.getDuration(),
                d.getDate(), d.getTime(), d.getDetails()
            });
        }
    } catch (Exception ex) {
```




```
JOptionPane.showMessageDialog(this, "Failed to load entries:\n" + ex.getMessage());  
}  
}
```

// Clear all input fields to prepare for a new entry or after updates

```
private void clearForm() {  
    tfTaskName.setText("");  
    tfAddress.setText("");  
    tfDuration.setText("");  
    tfDate.setText("");  
    tfTime.setText("");  
    taDetails.setText("");  
}
```

// Check that required fields (task name, date, time) are filled before allowing add/edit

```
private boolean validateEntryFields() {  
    if (tfTaskName.getText().trim().isEmpty() || tfDate.getText().trim().isEmpty() ||  
    tfTime.getText().trim().isEmpty()) {  
        JOptionPane.showMessageDialog(this, "Task Name, Date, and Time are required.");  
        return false;  
    }  
    return true;  
}
```

// Add a new diary entry based on input field values

```
private void addEntry() {  
    if (!validateEntryFields()) return; // Validate inputs first  
    try {  
        // Create new Diary object with user inputs and current user ID  
        Diary d = new Diary(0, tfTaskName.getText(), tfDuration.getText(), tfAddress.getText(),  
            tfDate.getText(), tfTime.getText(), taDetails.getText(), currentUser.getId());  
        recordManager.addRecord(d); // Add to DB  
        JOptionPane.showMessageDialog(this, "Entry added!");  
        clearForm(); // Clear inputs for next entry  
        loadEntries(); // Refresh table to show new entry  
    } catch (Exception ex) {  
        JOptionPane.showMessageDialog(this, "Failed to add entry:\n" + ex.getMessage());  
    }  
}
```

// Edit currently selected diary entry with updated input values

```
private void editEntry() {  
    int row = table.getSelectedRow();  
    // Validate that a row is selected
```



```
if (row < 0 || loadedDiaries == null || row >= loadedDiaries.size()) {
    JOptionPane.showMessageDialog(this, "Select a row to edit.");
    return;
}
if (!validateEntryFields()) return; // Validate inputs

try {
    Diary d = loadedDiaries.get(row);
    // Update diary fields with new input values
    d.setName(tfTaskName.getText());
    d.setAddress(tfAddress.getText());
    d.setDuration(tfDuration.getText());
    d.setDate(tfDate.getText());
    d.setTime(tfTime.getText());
    d.setDetails(taDetails.getText());

    recordManager.updateRecord(d); // Save updates to DB
    JOptionPane.showMessageDialog(this, "Entry updated!");
    clearForm();
    loadEntries();
} catch (Exception ex) {
    JOptionPane.showMessageDialog(this, "Failed to edit entry:\n" + ex.getMessage());
}

// Delete the selected diary entry from database and table
private void deleteEntry() {
    int row = table.getSelectedRow();
    if (row < 0 || loadedDiaries == null || row >= loadedDiaries.size()) {
        JOptionPane.showMessageDialog(this, "Select a row to delete.");
        return;
    }
    try {
        Diary d = loadedDiaries.get(row);
        recordManager.deleteRecord(d.getId()); // Delete from DB
        JOptionPane.showMessageDialog(this, "Entry deleted!");
        clearForm();
        loadEntries(); // Refresh table
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Failed to delete entry:\n" + ex.getMessage());
    }
}

// Log the user out: close this frame, clear user session, show login screen again
```



```
private void logout() {
    dispose();      // Close current window
    currentUser = null; // Clear user session info
    showLogin();    // Launch login window
}

// Show prompt to enter a new password and update it in database if valid
private void changePassword() {
    String newPwd = JOptionPane.showInputDialog(this, "Enter new password:");
    // Check if user entered a non-empty password
    if (newPwd != null && !newPwd.trim().isEmpty()) {
        try {
            loginManager.changePassword(currentUser.getUsername(), newPwd); // Update password in DB
            JOptionPane.showMessageDialog(this, "Password changed.");
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
        }
    }
}
```

Finally, the **main()** method in the Main class is the program's launch point.

It first applies the system look-and-feel to give Swing components a native appearance, then schedules GUI creation on the Event-Dispatch Thread using `SwingUtilities.invokeLater`.

Inside that runnable, it simply instantiates `DiaryGUI` and calls `showLogin()`, bringing up the login window and transferring control to the user.

```
// ----- MAIN METHOD TO START APPLICATION -----
// Entry point of the diary application.
// Sets the system look and feel, then opens the login GUI.
public class Main {
    public static void main(String[] args) {
        try {
            // Use native OS look and feel for better UI consistency
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception ignored) {}

        // Run the GUI creation on the Swing event dispatch thread for thread safety
        SwingUtilities.invokeLater(() -> new DiaryGUI().showLogin());
    }
}
```



Output and results

Must include the screenshots of the running program for every case with detailed explanation.

Adding data entry

Diary Management System - User: Malak Hany

Task Name: Programming Trial 1

Address: D:/

Duration: 5 min

Date (YYYY-MM-DD): 2025-05-16

Time (HH:MM:SS): 12:48:53

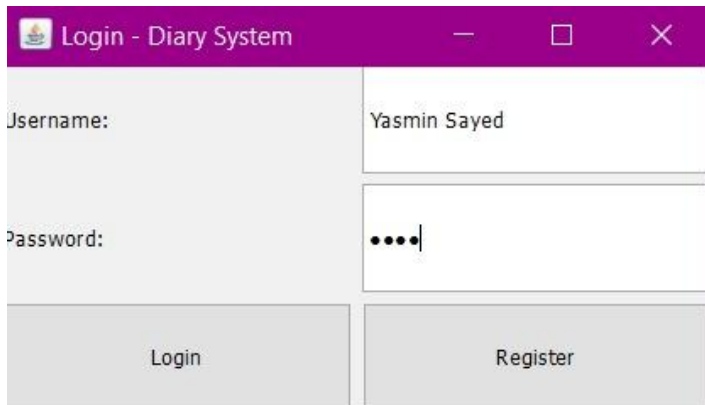
Details: Run 1st Trial

Add Edit Delete Change Password Logout

Task Name	Address	Duration	Date	Time	Details
Programming Trial 1	D:/	5 min	2025-05-16	12:48:53	Run 1st Trial



Registering account



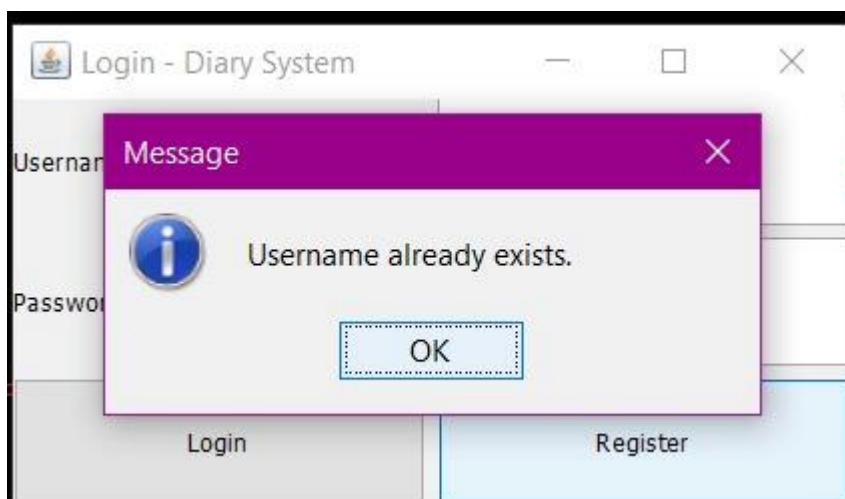
Login - Diary System

Username: Yasmin Sayed

Password:

Login Register

If its already registered



Login - Diary System

Username:

Password:

Login Register

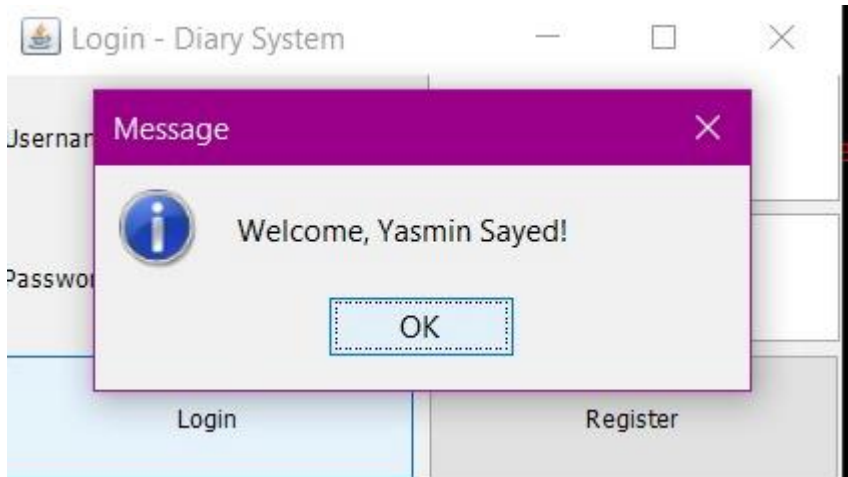
Message

Username already exists.

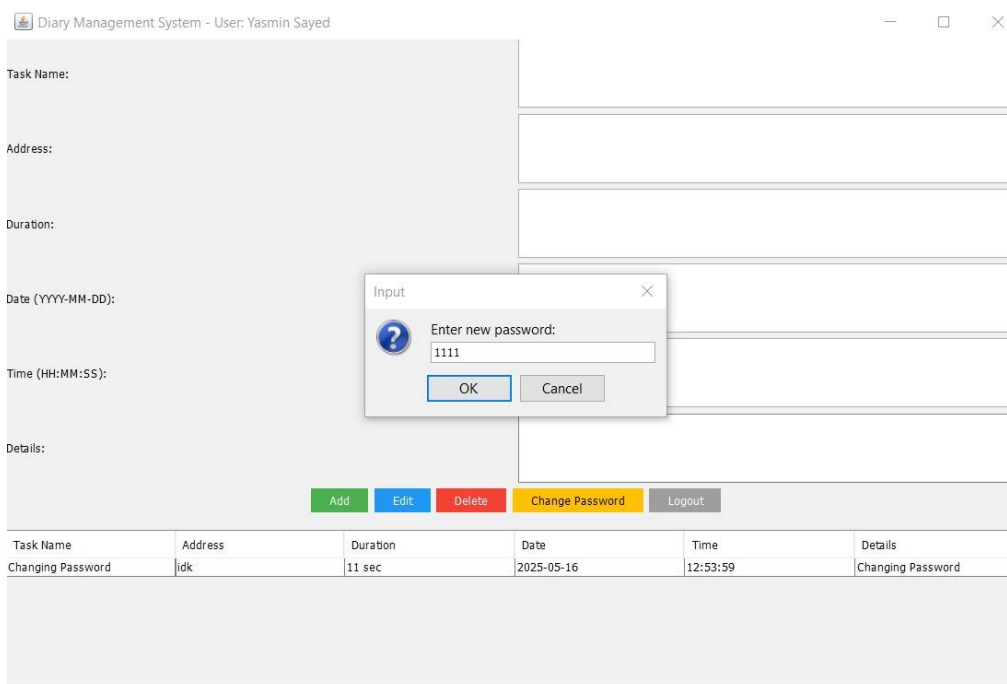
OK



Logging in



Changing password





Before deleting

Diary Management System - User: Yasmin Sayed

Task Name:

Address:

Duration:

Date (YYYY-MM-DD):

Time (HH:MM:SS):

Details:

Add Edit Delete Change Password Logout

Task Name	Address	Duration	Date	Time	Details
Changing Password	ldk	11 sec	2025-05-16	12:53:59	Changing Password
Deleting entry	sbh	20 sec	2025-05-16	12:54:15	Deleting one of the entries

Message when deleting

Diary Management System - User: Yasmin Sayed

Task Name:

Address:

Duration:

Date (YYYY-MM-DD):

Time (HH:MM:SS):

Details:

Add Edit Delete Change Password Logout

Deleting entry

sbh

20 sec

Deleting one of the entries

Message

Entry deleted!

OK

Task Name	Address	Duration	Date	Time	Details
Changing Password	ldk	11 sec	2025-05-16	12:53:59	Changing Password
Deleting entry	sbh	20 sec	2025-05-16	12:54:15	Deleting one of the entries



After deleting

Diary Management System - User: Yasmin Sayed

Task Name:

Address:

Duration:

Date (YYYY-MM-DD):

Time (HH:MM:SS):

Details:

Add

Edit

Delete

Change Password

Logout

Task Name	Address	Duration	Date	Time	Details
Changing Password	ldk	11 sec	2025-05-16	12:53:59	Changing Password

Editing

Diary Management System - User: Yasmin Sayed

Task Name:

Address:

Duration:

Date (YYYY-MM-DD):

Time (HH:MM:SS):

Details:

Add

Edit

Delete

Change Password

Logout

Task Name	Address	Duration	Date	Time	Details
Changing Password	ldk	11 sec	2025-05-16	12:53:59	Changing Password

After editing

Diary Management System - User: Yasmin Sayed

Task Name:

Address:

Duration:

Date (YYYY-MM-DD):

Time (HH:MM:SS):

Details:

[Add](#) [Edit](#) [Delete](#) [Change Password](#) [Logout](#)

Task Name	Address	Duration	Date	Time	Details
Changing Password	Edited	11 sec	2025-05-16	12:53:59	Changing Password

Database after adding accounts

phpMyAdmin

Server: 127.0.0.1 » Database: diarysystem » Table: diary

Showing rows 0 - 2 (3 total, Query took 0.0004 seconds)

`SELECT * FROM `diary``

☐ Profiling [\[Edit inline \]](#) [\[Edit \]](#) [\[Explain SQL \]](#) [\[Create PHP code \]](#) [\[Refresh \]](#)

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	id	name	duration	address	date	time	details	user_id
<input type="checkbox"/> Edit Copy Delete	2	Programming Trial 1	5 min	D/	2025-05-16	12:48:53	Run 1st Trial	2
<input type="checkbox"/> Edit Copy Delete	3	Changing Password	11 sec	Edited	2025-05-16	12:53:59	Changing Password	3

☐ Check all | With selected: [Edit](#) [Copy](#) [Delete](#) [Export](#)

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Query results operations

[Print](#) [Copy to clipboard](#) [Export](#) [Display chart](#) [Create view](#)

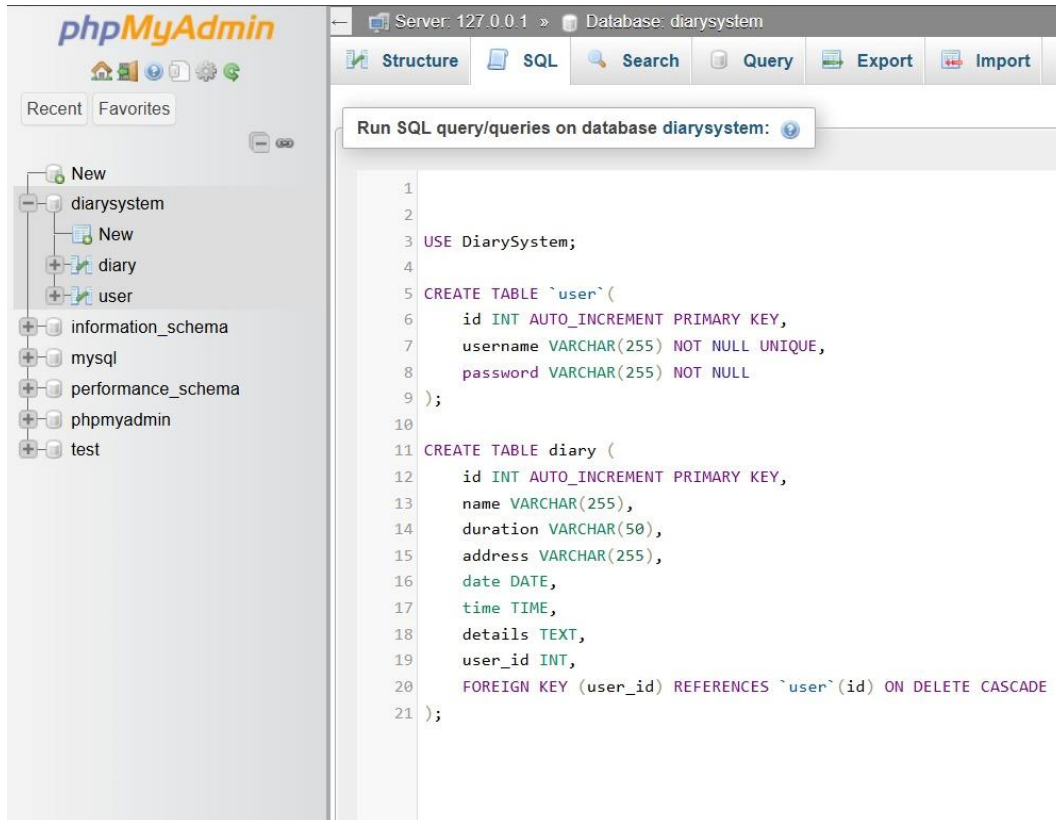
Bookmark this SQL query

Label: ☐ Let every user access this bookmark

[Bookmark this SQL query](#)



Database Query



The screenshot shows the phpMyAdmin interface. The left sidebar displays the database structure, including 'diarysystem' and its sub-databases 'diary' and 'user'. The main area shows the SQL query editor for the 'diarysystem' database. The query is as follows:

```
1  
2  
3 USE DiarySystem;  
4  
5 CREATE TABLE `user`(  
6     id INT AUTO_INCREMENT PRIMARY KEY,  
7     username VARCHAR(255) NOT NULL UNIQUE,  
8     password VARCHAR(255) NOT NULL  
9 );  
10  
11 CREATE TABLE diary (  
12     id INT AUTO_INCREMENT PRIMARY KEY,  
13     name VARCHAR(255),  
14     duration VARCHAR(50),  
15     address VARCHAR(255),  
16     date DATE,  
17     time TIME,  
18     details TEXT,  
19     user_id INT,  
20     FOREIGN KEY (user_id) REFERENCES `user` (id) ON DELETE CASCADE  
21 );
```



GitHub

<https://github.com/Malak-s-organization/DiarySystemManagementt.git>

```
MINGW64:/c:/Users/Me/Desktop/Repo/DiarySystemManagement
me@DESKTOP-I3VH6V2 MINGW64 ~
$ cd Desktop
me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop
$ cd Repo
me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo
$ git clone https://github.com/malakhany419/DiaryManagementSystem.git
fatal: Too many arguments.

usage: git clone [<options>] [--] <repo> [<dir>]

  -v, --[no-]verbose          be more verbose
  -q, --[no-]quiet            be more quiet
  --[no-]progress             force progress reporting
  --[no-]reject-shallow       don't clone shallow repository
  -n, --no-checkout           don't create a checkout
  --checkout                  opposite of --no-checkout
  --[no-]bare                 create a bare repository
  --[no-]mirror               create a mirror repository (implies --bare)
  -l, --[no-]local            to clone from a local repository
  --no-hardlinks              don't use local hardlinks, always copy
  --hardlinks                 opposite of --no-hardlinks
  -s, --[no-]shared           setup as shared repository
  --[no-]recurse-submodules[=<pathspec>]
                              initialize submodules in the clone
  --[no-]recursive ...       alias of --recurse-submodules
  -j, --[no-]jobs <n>         number of submodules cloned in parallel
  --[no-]template <template-directory>
                              directory from which templates will be used
  --[no-]reference <repo>     reference repository
  --[no-]reference-if-able <repo>
                              reference repository
  --[no-]dissociate           use --reference only while cloning
  -o, --[no-]origin <name>    use <name> instead of 'origin' to track upstream
  -b, --[no-]branch <branch> checkout <branch> instead of the remote's HEAD
  --[no-]revision <rev>      clone single revision <rev> and check out
  -u, --[no-]upload-pack <path>
                              path to git-upload-pack on the remote
  --[no-]depth <depth>       create a shallow clone of that depth
  --[no-]shallow-since <time>
                              create a shallow clone since a specific time
  --[no-]shallow-exclude <ref>
                              deepen history of shallow clone, excluding ref
  --[no-]single-branch        clone only one branch, HEAD or --branch
```



```
MINGW64/c/Users/Me/Desktop/Repo/DiarySystemManagement
--[no]-tags           clone tags, and make later fetches not to follow them
--[no]-shallow-submodules
                        any cloned submodules will be shallow
--[no]-separate-git-dir <gitdir>
                        separate git dir from working tree
--[no]-ref-format <format>
                        specify the reference format to use
-c, --[no]-config <key-value>
                        set config inside the new repository
--[no]-server-option <server-specific>
                        option to transmit
-4, --ipv4             use IPv4 addresses only
-6, --ipv6             use IPv6 addresses only
--[no]-filter <args>   object filtering
--[no]-also-filter-submodules
                        apply partial clone filters to submodules
--[no]-remote-submodules
                        any cloned submodules will use their remote-tracking branch
--[no]-sparse          initialize sparse-checkout file to include only files at root
--[no]-bundle-uri <uri>
                        a URI for downloading bundles before fetching from origin remote
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo
$ git clone https://github.com/malakany429/DiarySystemManagement.git
Cloning into 'DiarySystemManagement'...
fatal: unable to access 'https://github.com/malakany429/DiarySystemManagement.git/': Could not resolve host: github
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo
$ cd DiarySystemManagement
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  Main.java

nothing added to commit but untracked files present (use "git add" to track)
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git add Main.java
warning: in the working copy of 'Main.java', LF will be replaced by CRLF the next time Git touches it
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.
```

```
MINGW64/c/Users/Me/Desktop/Repo/DiarySystemManagement
```

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   Main.java
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git commit -m "uploaded the src code for the project."
Author identity unknown
```

*** Please tell me who you are.

Run

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'Me@DESKTOP-I3VH6V2.(none)')

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git config --global user.email "malak.hany2@msa.edu.eg"
error: key does not contain a section: global
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git config --global user.email "malak.hany2@msa.edu.eg"
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git config --global user.name "malakany419"
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git commit -m "uploaded the src code for the project."
[main 7c90f30] uploaded the src code for the project.
 1 file changed, 599 insertions(+)
 create mode 100644 Main.java
```

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
```

nothing to commit, working tree clean

```
Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
```




```
your branch is ahead of origin/main by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean

Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 5.91 KiB | 1.48 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/malakhany419/DiarySystemManagement.git
  48280eb..7c90f30  main -> main
branch 'main' set up to track 'origin/main'.

Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

Me@DESKTOP-I3VH6V2 MINGW64 ~/Desktop/Repo/DiarySystemManagement (main)
$ _
```

References

1. <https://www.surfsidemedia.in/post/developing-a-personal-diary-app-in-java>