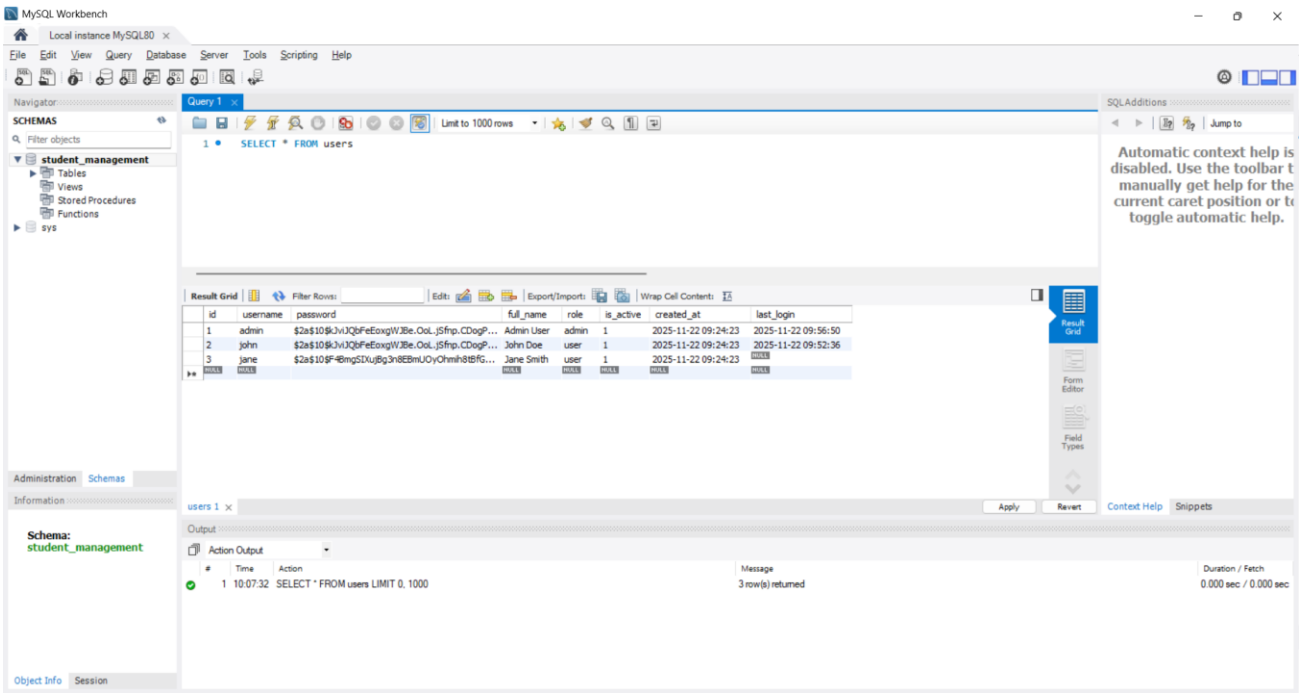
Web Application Development Lab 06
AUTHENTICATION & SESSION MANAGEMENT
Part A – INCLASS PRACTICE & EXERCISES

I. DATABASE & USER MODEL



II. USER MODEL & DAO

1. User model

```
package com.student.model;

import java.sql.Timestamp;

public class User {
    private int id;
    private String username;
    private String password;
    private String fullName;
    private String role;
```

```
private boolean isActive;
private Timestamp createdAt;
private Timestamp lastLogin;

// Constructors
public User() {
}

public User(String username, String password, String fullName, String role) {
    this.username = username;
    this.password = password;
    this.fullName = fullName;
    this.role = role;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
```

```
    this.username = username;
}

public String getPassword() {
    return password;
}

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

public String getFullName() {
    return fullName;
}

public void setFullName(String fullName) {
    this.fullName = fullName;
}

public String getRole() {
    return role;
}

public void setRole(String role) {
    this.role = role;
}

public boolean isActive() {
```

```
    return isActive;
}

public void setActive(boolean active) {
    isActive = active;
}

public Timestamp getCreatedAt() {
    return createdAt;
}

public void setCreatedAt(Timestamp createdAt) {
    this.createdAt = createdAt;
}

public Timestamp getLastLogin() {
    return lastLogin;
}

public void setLastLogin(Timestamp lastLogin) {
    this.lastLogin = lastLogin;
}

public boolean isAdmin() {
    return "admin".equalsIgnoreCase(this.role);
}

public boolean isUser() {
```

```
        return "user".equalsIgnoreCase(this.role);
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", fullName='" + fullName + '\'' +
                ", role='" + role + '\'' +
                ", isActive=" + isActive +
                '}';
    }
}
```

2. User DAO

```
package com.student.dao;

/**
 *
 * @author Admin
 */
import com.student.model.User;
import org.mindrot.jbcrypt.BCrypt;

import java.sql.*;

public class UserDAO {
```

```
private static final String DB_URL = "jdbc:mysql://localhost:3306/student_management";
private static final String DB_USER = "lvp";
private static final String DB_PASSWORD = "050904";

// SQL Queries
private static final String SQL_AUTHENTICATE =
    "SELECT * FROM users WHERE username = ? AND is_active = TRUE";

private static final String SQL_UPDATE_LAST_LOGIN =
    "UPDATE users SET last_login = NOW() WHERE id = ?";

private static final String SQL_GET_BY_ID =
    "SELECT * FROM users WHERE id = ?";

private static final String SQL_GET_BY_USERNAME =
    "SELECT * FROM users WHERE username = ?";

private static final String SQL_INSERT =
    "INSERT INTO users (username, password, full_name, role) VALUES (?, ?, ?, ?)";

// Get database connection
private Connection getConnection() throws SQLException {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
    } catch (ClassNotFoundException e) {
        throw new SQLException("MySQL Driver not found", e);
```

```
    }
}

/**
 * Authenticate user with username and password
 * @param username
 * @param password
 * @return User object if authentication successful, null otherwise
 */
public User authenticate(String username, String password) {
    User user = null;

    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(SQL_AUTHENTICATE)) {

        pstmt.setString(1, username);

        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                String hashedPassword = rs.getString("password");

                // Verify password with BCrypt
                if (BCrypt.checkpw(password, hashedPassword)) {
                    user = mapResultSetToUser(rs);

                    // Update last login time
                    updateLastLogin(user.getId());
                }
```

```
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return user;
}

/**
 * Update user's last login timestamp
 */
private void updateLastLogin(int userId) {
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(SQL_UPDATE_LAST_LOGIN))
{

        pstmt.setInt(1, userId);
        pstmt.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * Get user by ID
 * @param id
```

```
 * @return
 */
public User getUserById(int id) {
    User user = null;

    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(SQL_GET_BY_ID)) {

        pstmt.setInt(1, id);

        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                user = mapResultSetToUser(rs);
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return user;
}

/**
 * Get user by username
 */
public User getUserByUsername(String username) {
    User user = null;
```

```
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(SQL_GET_BY_USERNAME)) {

        pstmt.setString(1, username);

        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                user = mapResultSetToUser(rs);
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return user;
}

/**
 * Create new user with hashed password
 */
public boolean createUser(User user) {
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(SQL_INSERT)) {

        // Hash password before storing
        String hashedPassword = BCrypt.hashpw(user.getPassword(), BCrypt.gensalt());
```

```
            pstmt.setString(1, user.getUsername());
            pstmt.setString(2, hashedPassword);
            pstmt.setString(3, user.getFullName());
            pstmt.setString(4, user.getRole());

            int rowsAffected = pstmt.executeUpdate();
            return rowsAffected > 0;

        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * Map ResultSet to User object
     */
    private User mapResultSetToUser(ResultSet rs) throws SQLException {
        User user = new User();
        user.setId(rs.getInt("id"));
        user.setUsername(rs.getString("username"));
        user.setPassword(rs.getString("password"));
        user.setFullName(rs.getString("full_name"));
        user.setRole(rs.getString("role"));
        user.setActive(rs.getBoolean("is_active"));
        user.setCreatedAt(rs.getTimestamp("created_at"));
        user.setLastLogin(rs.getTimestamp("last_login"));
```

```
        return user;
    }

    /**
     * Test
     */
    public static void main(String[] args) {
        // Generate hash for "password123"
        String plainPassword = "thisIsASecretPassword";
        String hashedPassword = BCrypt.hashpw(plainPassword, BCrypt.gensalt());
        System.out.println("Plain: " + plainPassword);
        System.out.println("Hashed: " + hashedPassword);

        // Test verification
        boolean matches = BCrypt.checkpw(plainPassword, hashedPassword);
        System.out.println("Verification: " + matches);
    }
}
```

Authentication testing:

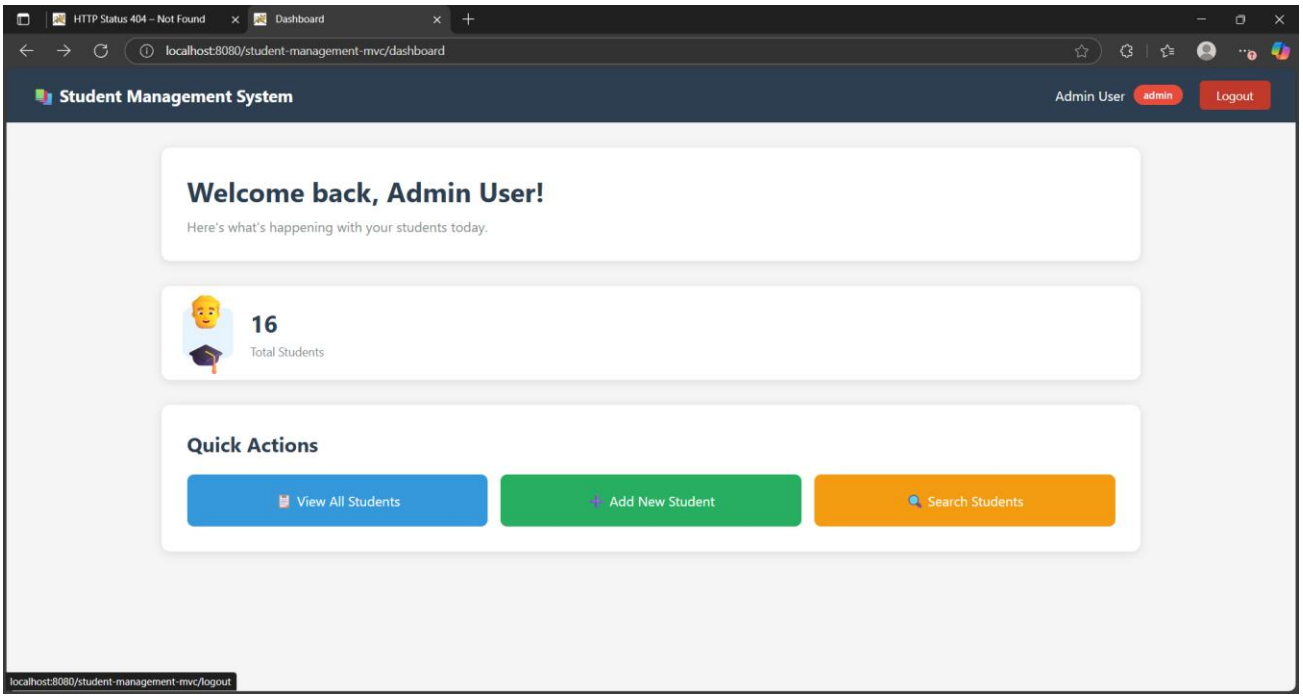
```
--- exec:3.1.0:exec (default-cli) @ student-management-mvc ---
Authentication successful!
User{id=1, username='admin', fullName='Admin User', role='admin', isActive=true}
Invalid auth: Correctly rejected
------------------------------------------------------------------------
```

First line: - Testing a valid authentication

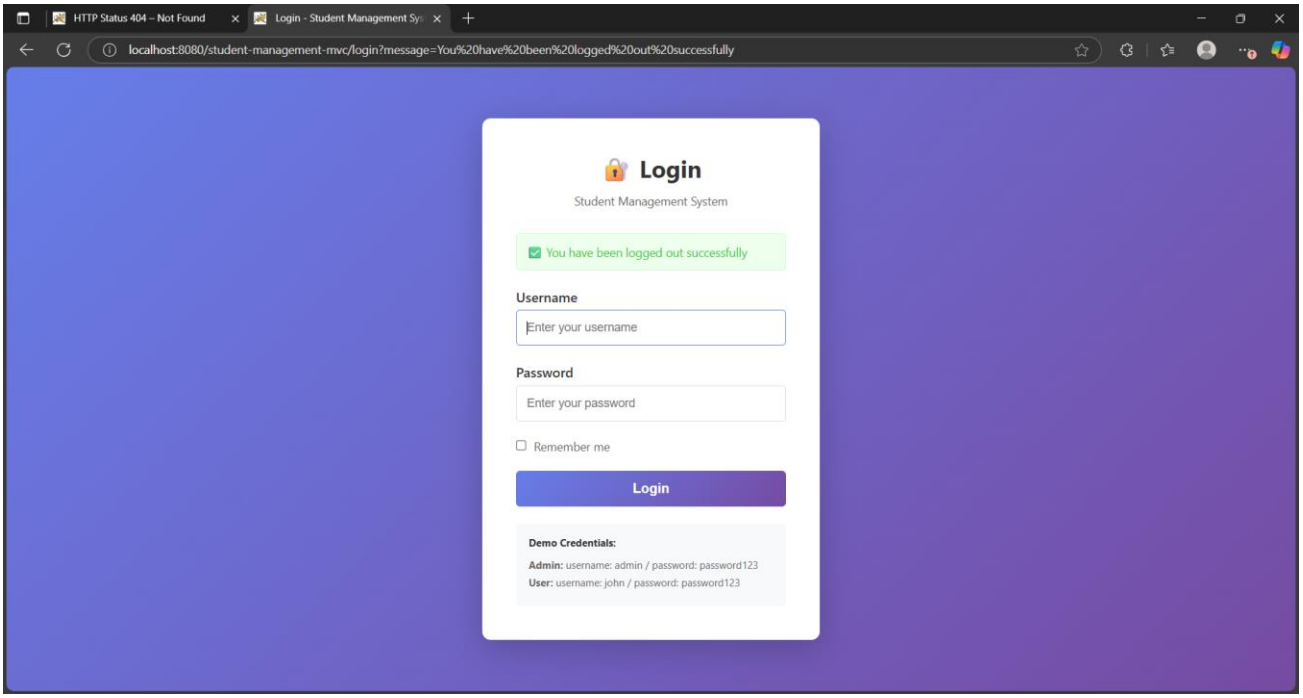
Second line: - Printing the user information

Third line: - Testing an invalid authentication

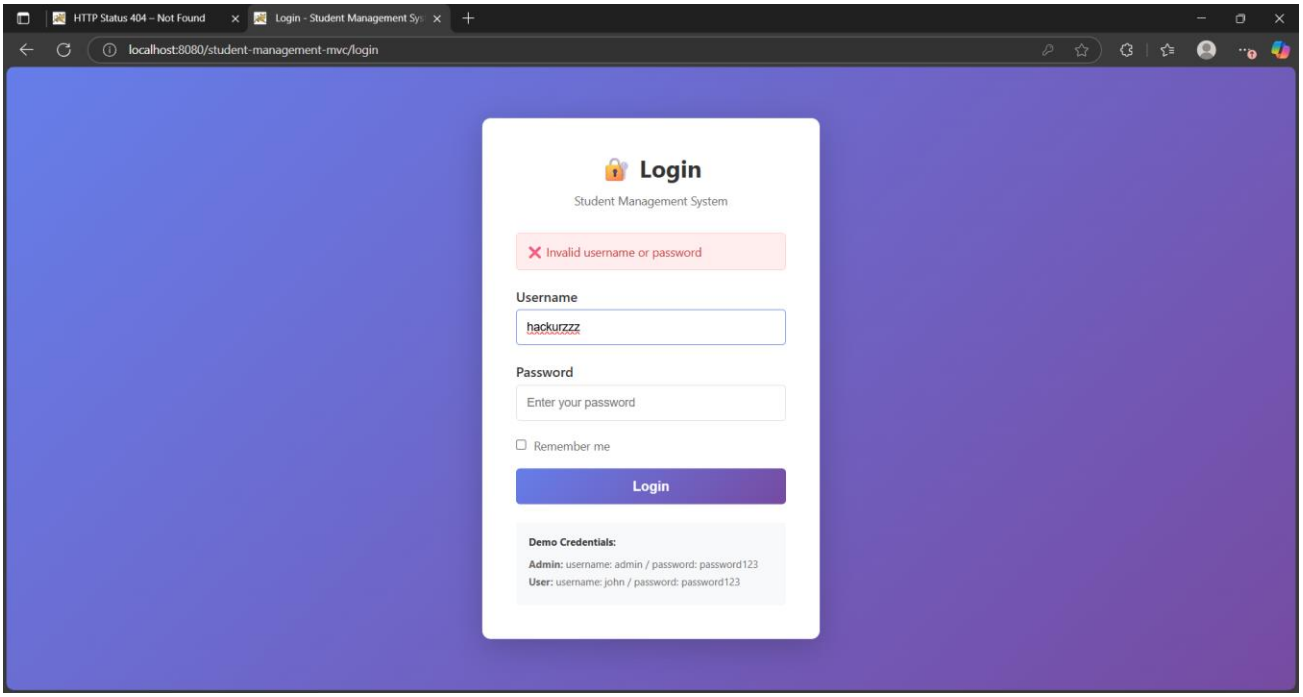
III. LOGIN/LOGOUT CONTROLLERS



Logging out



Logging out successfully



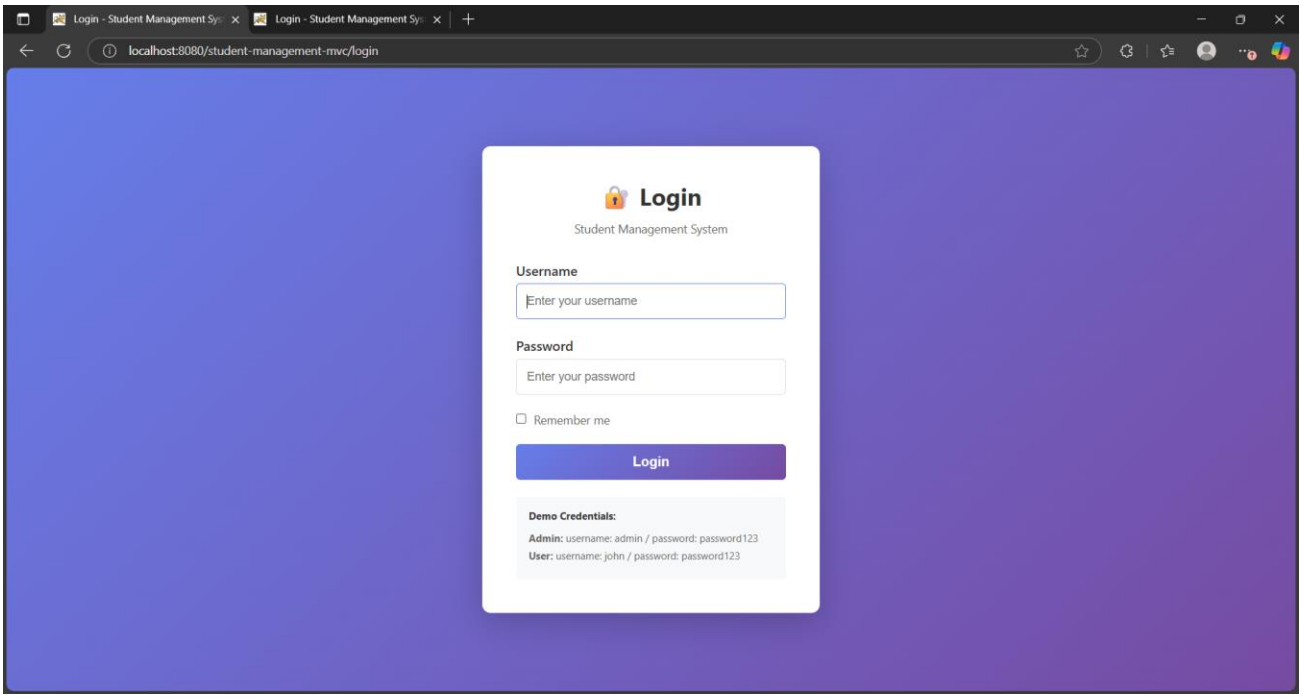
Error message

*Issue:

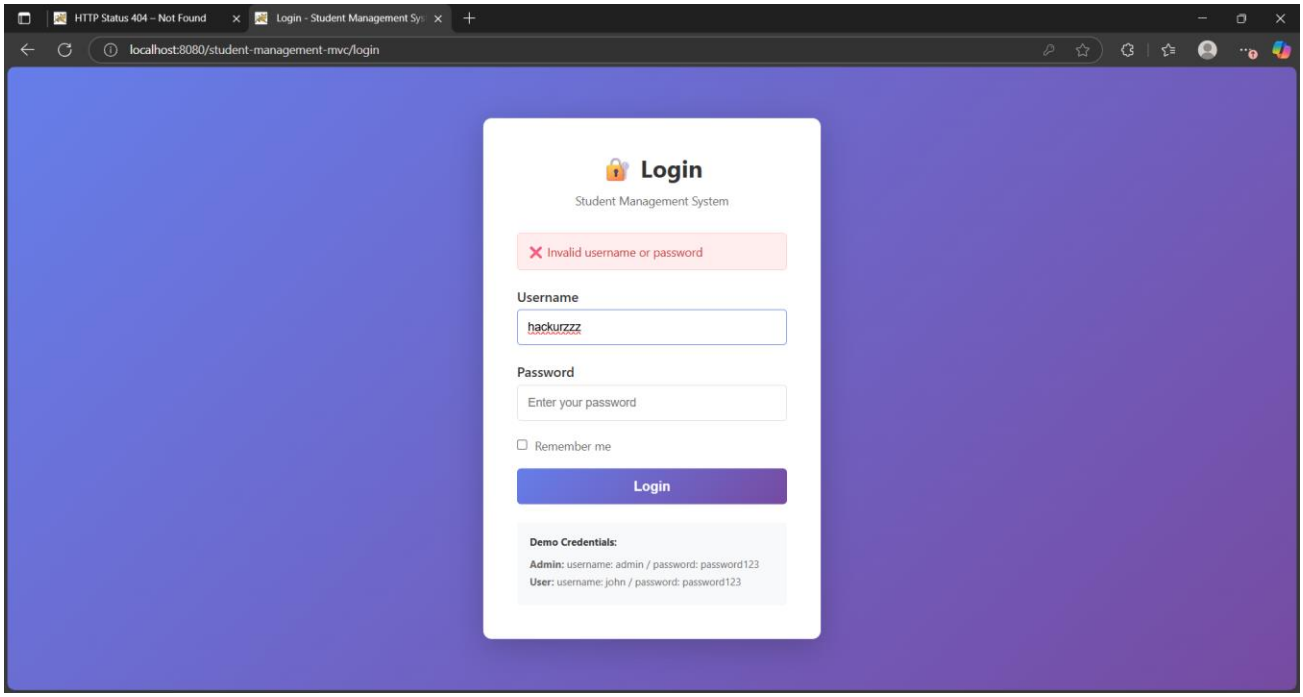
- Login also takes space ‘ ‘ into account when validating users, even though trim() is applied [BUG]. (This could be that the string is not padded with traditional white space)

IV. VIEWS & DASHBOARD

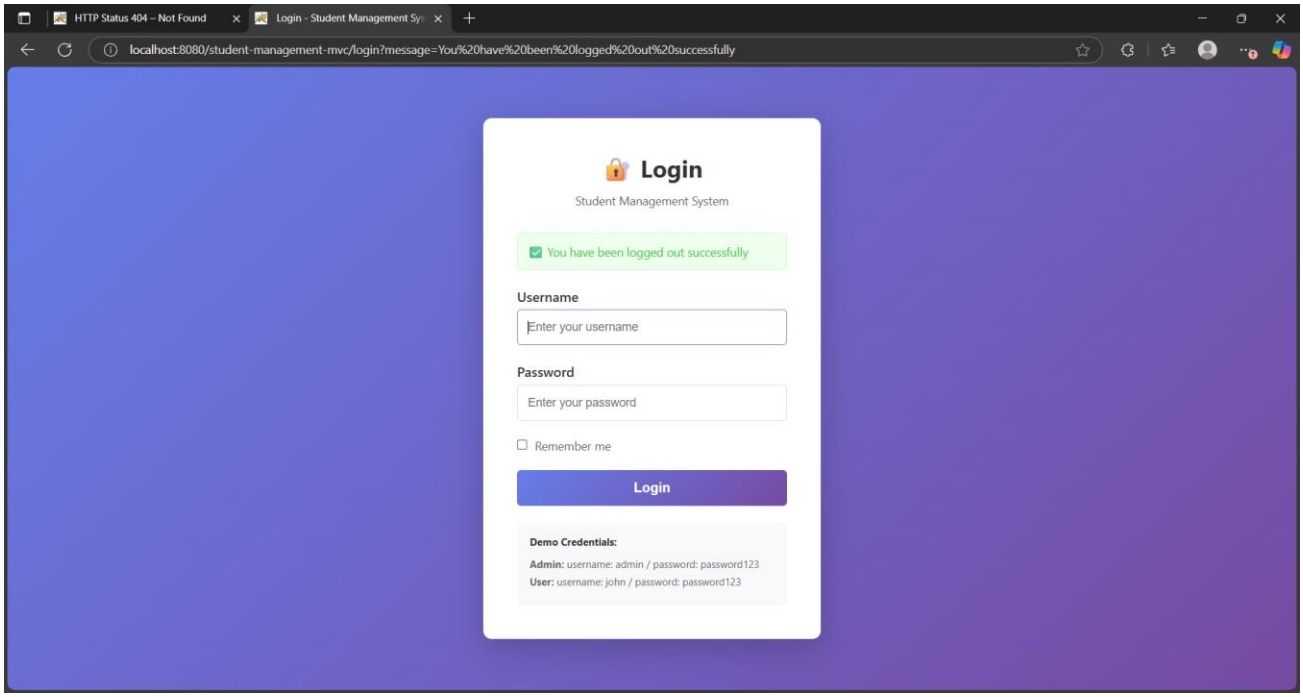
1. Login



Default login page

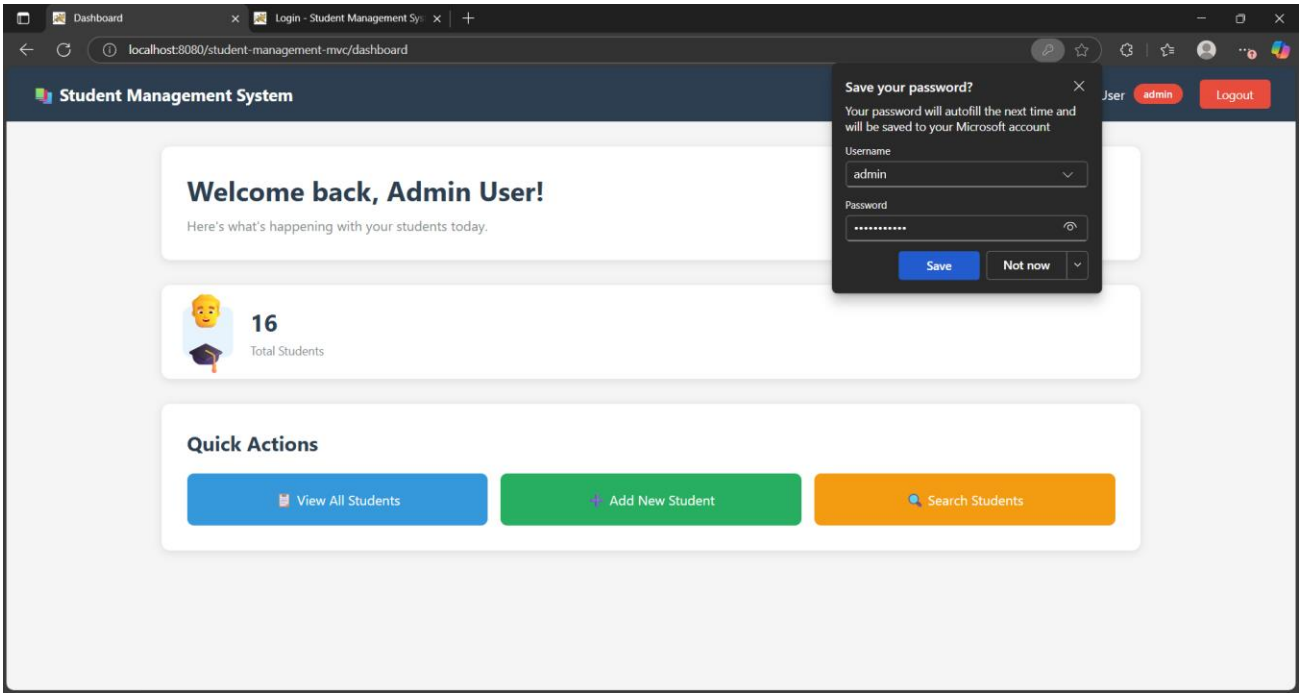


Error message

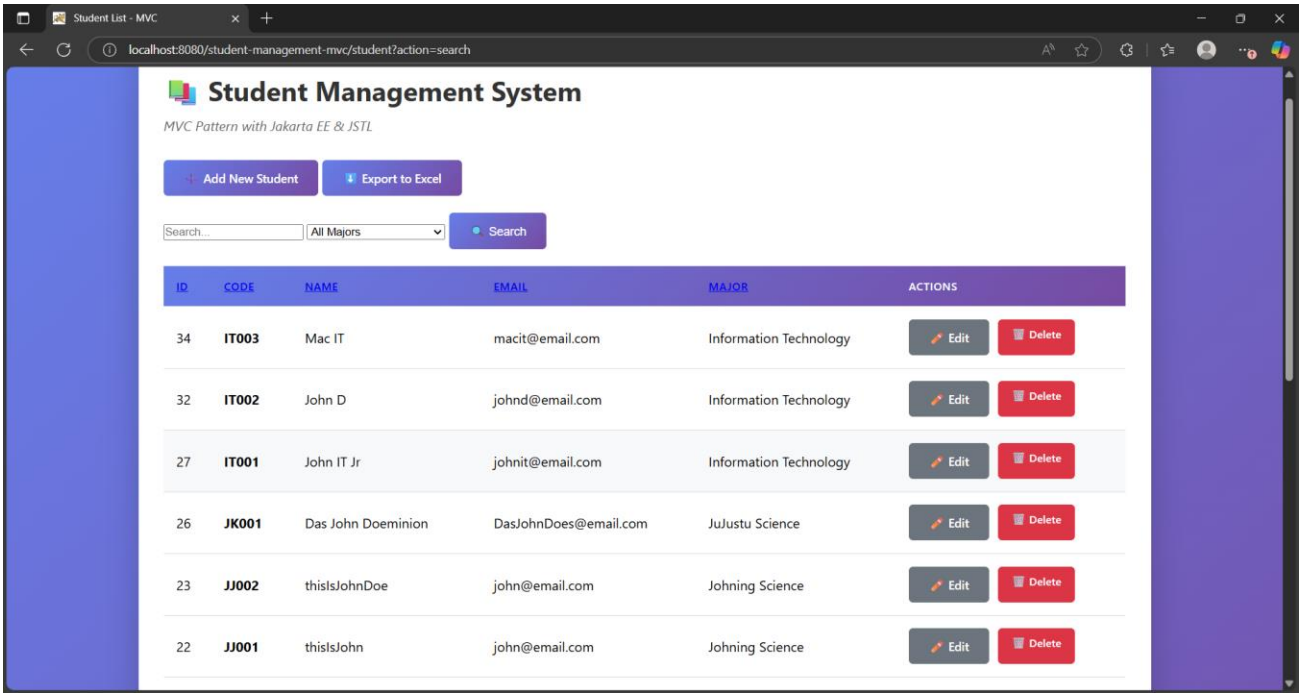


Logs out successfully

2. Dashboard



Default dashboard



Quick actions take you to /student