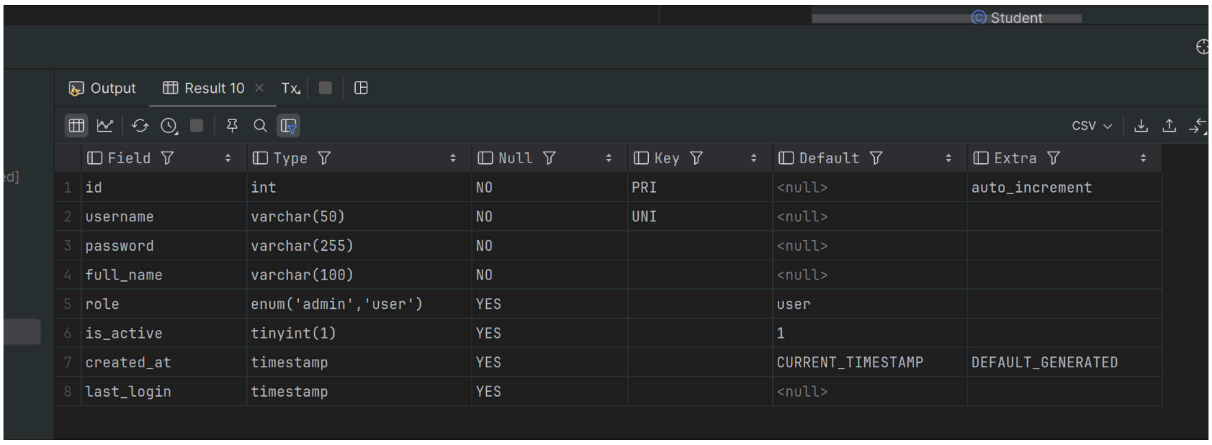
Task 1.1: Create Users Table (5 points)

Execute this SQL in your MySQL database:

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    role ENUM('admin', 'user') DEFAULT 'user',
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP NULL
);
```

Verification Query:

```
DESCRIBE users;
```



	Field	Type	Null	Key	Default	Extra
1	id	int	NO	PRI	<null>	auto_increment
2	username	varchar(50)	NO	UNI	<null>	
3	password	varchar(255)	NO		<null>	
4	full_name	varchar(100)	NO		<null>	
5	role	enum('admin','user')	YES		user	
6	is_active	tinyint(1)	YES		1	
7	created_at	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
8	last_login	timestamp	YES		<null>	

Task 1.2: Generate Hashed Passwords (5 points)

Create test file: src/util/PasswordHashGenerator.java

Steps:

1. Add BCrypt library to project
2. Create PasswordHashGenerator class
3. Run the main method
4. Copy generated hash

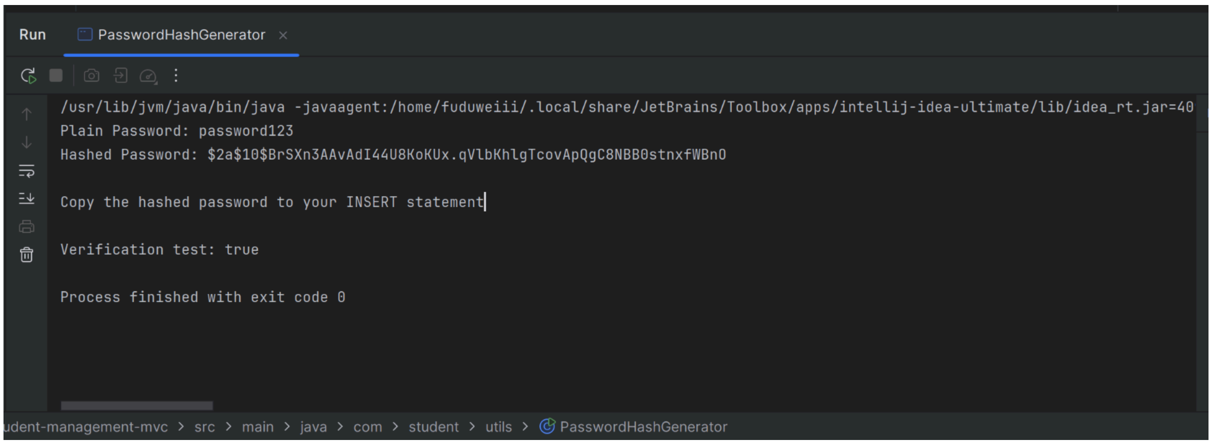
Evaluation Criteria:

Criteria	**Points**
BCrypt library added	2
Hash generated successfully	2
Verification works	1

Code for testing:

```java
package com.student.utils;

import org.mindrot.jbcrypt.BCrypt;

public class PasswordHashGenerator {

  public static void main(String[] args) {
      String plainPassword = "password123";

      // Generate hash
      String hashedPassword = BCrypt.hashpw(plainPassword, BCrypt.gensalt());

      System.out.println("Plain Password: " + plainPassword);
      System.out.println("Hashed Password: " + hashedPassword);
      System.out.println("\nCopy the hashed password to your INSERT statement");

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

Testing successfully



```
/usr/lib/jvm/java/bin/java -javaagent:/home/fuduweiii/.local/share/JetBrains/Toolbox/apps/intellij-idea-ultimate/lib/idea_rt.jar=40
Plain Password: password123
Hashed Password: $2a$10$BrSXn3AAvAdI44U8KoKUx.qVlbKhlgTcovApQgC8NBB0stnxfWBnO

Copy the hashed password to your INSERT statement

Verification test: true

Process finished with exit code 0
```

Run
PasswordHashGenerator
udent-management-mvc › src › main › java › com › student › utils › PasswordHashGenerator

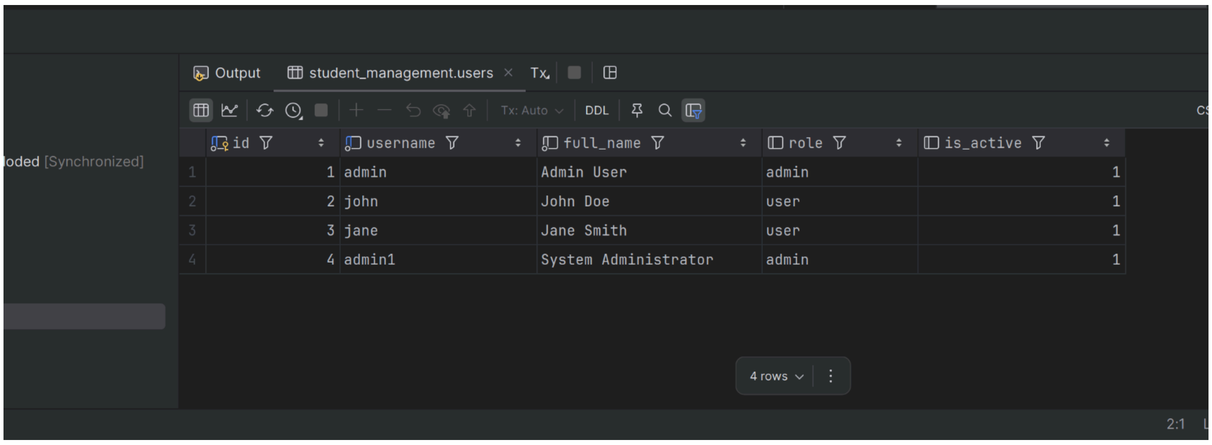
---

Task 1.3: Insert Test Users (5 points)

Use the hashed password from Task 1.2:

- Verify SQL: SELECT id, username, full_name, role, is_active FROM users;

Generate successfully



	id	username	full_name	role	is_active
1	1	admin	Admin User	admin	1
2	2	john	John Doe	user	1
3	3	jane	Jane Smith	user	1
4	4	admin1	System Administrator	admin	1

Task 2.1: Create User Model (7 points)

File: src/model/User.java

Requirements:

- Private attributes matching database columns
- Two constructors (no-arg and parameterized)
- All getters and setters
- Utility methods: isAdmin(), isUser()
- Override toString()

Evaluation Criteria:

Criteria	**Points**
All attributes declared correctly	2
Constructors implemented	2
Getters/setters complete	2
Utility methods implemented	1

```java
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;
```

```java
        this.role = role;
    }

    // Getters and Setters
    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;
    }

    // Utility methods
    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 +
                '}';
    }
}
```

- **All attributes declared correctly:** This criterion requires that the private fields (id, username, password, etc.) are declared using the correct Java data types (int, String, Timestamp) and the private access modifier. This step establishes the class structure and enforces encapsulation, ensuring the object's internal data state is protected from direct external manipulation.
- **Constructors implemented:** The class must include both a no-arg constructor (essential for reflection-based frameworks like ORMs) and a parameterized constructor (taking username, password, fullName, and role). These constructors ensure the object can be properly instantiated either by the framework when reading data or by the developer when creating a new record.
- **Getters/setters complete:** Getters (accessors) and setters (mutators) must be provided for every private attribute. This pair of public methods is the mechanism used to adhere to encapsulation, allowing controlled rea ding and writing of the private data. The getter for

the boolean field (isActive) must correctly follow the is[PropertyName] convention (e.g., isActive()).

- **Utility methods implemented:** This criterion requires the implementation of helper methods like isAdmin() and isUser(). These methods encapsulate simple business logic by checking the state of the role attribute. They improve code readability and maintainability, especially when using case-insensitive comparison (equalsIgnoreCase()) for robust role checking.

Task 2.2: Create UserDAO (8 points)

File: `src/dao/UserDAO.java`

Requirements:

- Database connection method
- `authenticate(username, password)` method
- `getUserById(id)` method
- `updateLastLogin(userId)` method
- Use BCrypt for password verification

```
package com.student.dao;


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:3307/student_management";

   private static final String DB_USER = "user1";

   private static final String DB_PASSWORD = "user1";


   // SQL Queries

   private static final String SQL_AUTHENTICATE =
```

```java
            "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
     * @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();
        }
```

```java
        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
     */
    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());
```

```java
            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 method - Generate hashed password

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

        // Generate hash for "password123"

        String plainPassword = "admin123";

                    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);


        System.out.println("------------------------------------");


        UserDAO userDAO = new UserDAO();

        User newUser = new User();

        newUser.setUsername("admin1");

        newUser.setPassword("admin123");

        newUser.setFullName("System Administrator");

        newUser.setRole("admin");

                   System.out.println("Attempting   to   create   user:   "   +
newUser.getUsername() + " with role: " + newUser.getRole());

        boolean isCreated = userDAO.createUser(newUser);

        if (isCreated) {

            System.out.println("✅ User created successfully!");


                                                    User     createdUser     =
userDAO.getUserByUsername(newUser.getUsername());

            if (createdUser != null) {
                System.out.println("Fetched User ID: " + createdUser.getId());
                System.out.println("Fetched User Role: " + createdUser.getRole());
            } else {
                System.out.println("❌ Error: Could not retrieve the created user.");
            }


        } else {
            System.out.println("❌ Failed to create user. Check the database connection and logs (SQLException).");
        }
    }
}
```

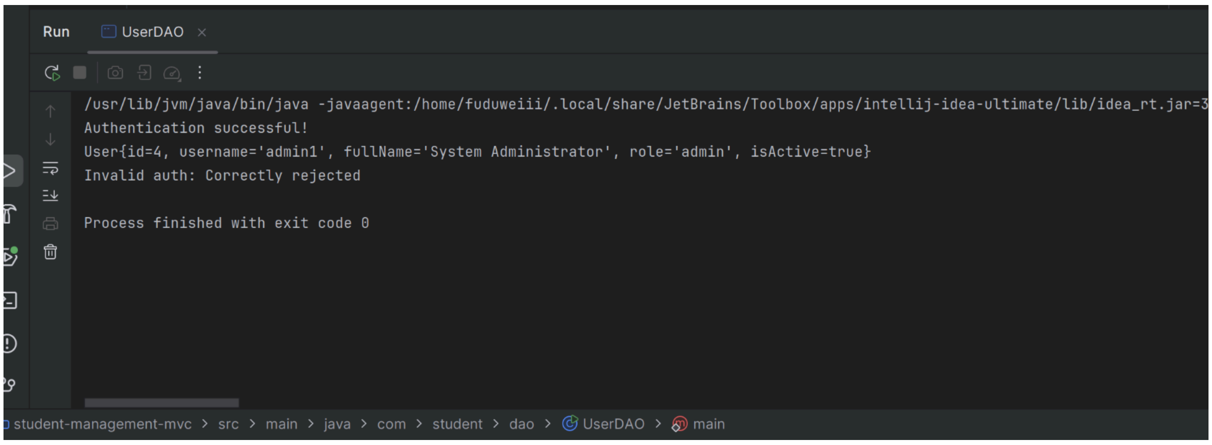
- **Database Connection Method (getConnection):** The getConnection() method is a private utility designed to abstract the necessary JDBC setup. It begins by attempting to load the MySQL driver (Class.forName("com.mysql.cj.jdbc.Driver")). Once the driver is registered, it uses DriverManager.getConnection() with the hardcoded connection URL, username, and password. This central method ensures that all Data Access Object (DAO) methods have a consistent and reliable way to establish a connection to the database.
- **authenticate(username, password):**
  - The authenticate method is the secure gateway for user login. The flow is as follows:
  - A database query is executed to retrieve the user record based on the provided username, ensuring the account is active (is_active = TRUE).
  - If a record is found, the stored hashed password is retrieved from the database.
  - The plaintext password provided by the user is verified against the stored hash using BCrypt.checkpw(password, hashedPassword).
  - If the verification succeeds, the ResultSet is mapped to a User object, and the private updateLastLogin method is immediately called to record the new login time.
  - The successfully authenticated User object is returned; otherwise, null is returned.
- **Secure Password Handling (BCrypt)**
  - Security is paramount, and the UserDAO correctly integrates BCrypt for robust password management:
    - Verification: In authenticate, BCrypt.checkpw() is used. This method performs a comparison between the user's plain-text input and the stored BCrypt hash, handling the salt and hashing process internally. This is the only secure way to verify passwords without ever storing or handling the plaintext password.

    - Hashing (Storage): While not explicitly required, the createUser method correctly demonstrates the use of BCrypt.hashpw(user.getPassword(), BCrypt.gensalt()) to generate a new secure hash before storing a new user's password in the database.
- **Retrieval and Timestamp Update:** The getUserById(id) method provides standard functionality for retrieving a User object by its unique ID, which is often used after authentication to refresh user data or verify permissions. The updateLastLogin(userId) method, which is marked as private, is integrated into the successful execution path of authenticate. It uses a simple UPDATE SQL statement with NOW() to automatically record the current server time as the user's last login, ensuring tracking of user activity without requiring manual time setting in the business logic.

Testing code

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

   UserDAO dao = new UserDAO();


   // Test authentication

   User user = dao.authenticate("admin1", "admin123");

   if (user != null) {

       System.out.println("Authentication successful!");

       System.out.println(user);

   } else {

       System.out.println("Authentication failed!");

   }


   // Test with wrong password

   User invalidUser = dao.authenticate("admin1", "wrongpassword");

    System.out.println("Invalid auth: " + (invalidUser == null ? "Correctly rejected" : "ERROR!"));

}
```

Testing successfully


Run
UserDAO
student-management-mvc > src > main > java > com > student > dao > UserDAO > main

```
/usr/lib/jvm/java/bin/java -javaagent:/home/fuduweiii/.local/share/JetBrains/Toolbox/apps/intellij-idea-ultimate/lib/idea_rt.jar=3
Authentication successful!
User{id=4, username='admin1', fullName='System Administrator', role='admin', isActive=true}
Invalid auth: Correctly rejected

Process finished with exit code 0
```

Task 3.1: Create Login Controller (10 points)

File: `src/controller/LoginController.java`

Requirements:

- Handle GET: Display login page
- Handle POST: Process login form
- Create session on successful login
- Redirect based on user role
- Display error messages

```java
package com.student.controller;

import com.student.dao.UserDAO;
import com.student.model.User;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.io.IOException;

@WebServlet("/login")
public class LoginController extends HttpServlet {

   private UserDAO userDAO;

   @Override
   public void init() {
       userDAO = new UserDAO();
   }

   /**
    * Display login page
    */
   @Override
     protected  void  doGet(HttpServletRequest  request,  HttpServletResponse
response)
           throws ServletException, IOException {

       // If already logged in, redirect to dashboard
       HttpSession session = request.getSession(false);
       if (session != null && session.getAttribute("user") != null) {
           response.sendRedirect("dashboard");
           return;
       }

       // Show login page
```

```java
            request.getRequestDispatcher("/views/login.jsp").forward(request,
response);
    }

    /**
     * Process login form
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String rememberMe = request.getParameter("remember");

        // Validate input
        if (username == null || username.trim().isEmpty() ||
                password == null || password.trim().isEmpty()) {

                    request.setAttribute("error", "Username and password are
required");
            request.getRequestDispatcher("/views/login.jsp").forward(request,
response);
            return;
        }

        // Authenticate user
        User user = userDAO.authenticate(username, password);

        if (user != null) {
            // Authentication successful

            // Invalidate old session (prevent session fixation)
            HttpSession oldSession = request.getSession(false);
            if (oldSession != null) {
                oldSession.invalidate();
            }

            // Create new session
            HttpSession session = request.getSession(true);
            session.setAttribute("user", user);
            session.setAttribute("role", user.getRole());
            session.setAttribute("fullName", user.getFullName());

            // Set session timeout (30 minutes)
            session.setMaxInactiveInterval(30 * 60);

            // Handle "Remember Me" (optional - cookie implementation)
            if ("on".equals(rememberMe)) {
                // TODO: Implement remember me functionality with cookie
            }
            // Redirect based on role
            if (user.isAdmin()) {
                response.sendRedirect("dashboard");
            } else {
                response.sendRedirect("student?action=list");
            }

        } else {
            // Authentication failed
            request.setAttribute("error", "Invalid username or password");
              request.setAttribute("username", username); // Keep username in
form
             request.getRequestDispatcher("/views/login.jsp").forward(request,
response);
        }
    }
}
```

1. Handling HTTP GET Request (doGet): displaying the login form.

- Pre-check: It first checks if a valid session and user attribute already exist (session != null && session.getAttribute("user") != null).
- Redirect if Logged In: If the user is already authenticated, they are immediately redirected to the dashboard to prevent them from seeing the login form again.
- Display Login: If no valid session is found, the request is forwarded to the /views/login.jsp page.

2. Handling HTTP POST Request (doPost): processes the submission of the login form and handles the core authentication logic.

1. Input Retrieval and Validation: Parameters for username and password are retrieved. If either is null or empty after trimming, an error message is set, and the user is forwarded back to the login page.
2. Authentication: The provided credentials are passed to userDAO.authenticate(username, password).
3. Success Path:
   - The old session is invalidated (oldSession.invalidate()) to mitigate session fixation attacks.
   - A new session is created (request.getSession(true)).
   - Key user data (user, role, fullName) is stored in the new session.
   - A session timeout is set (setMaxInactiveInterval(30 * 60)).
   - Role-Based Redirection: If user.isAdmin() is true, the user is sent to dashboard. Otherwise, the user is sent to the standard user entry point (student?action=list).
4. Failure Path:
   - If authentication fails (user == null), an "Invalid username or password" error message is set.
   - The entered username is retained in a request attribute to re-populate the form.
   - The user is forwarded back to /views/login.jsp.

Task 3.2: Create Logout Controller (5 points)

File: `src/controller/LogoutController.java`

Requirements:

- Invalidate session
- Redirect to login with message

```
package com.student.controller;


import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import jakarta.servlet.http.HttpSession;

import java.io.IOException;


@WebServlet("/logout")

public class LogoutController extends HttpServlet {


   @Override

   protected void doGet(HttpServletRequest request, HttpServletResponse
response)

           throws ServletException, IOException {


       // Get current session

       HttpSession session = request.getSession(false);
```

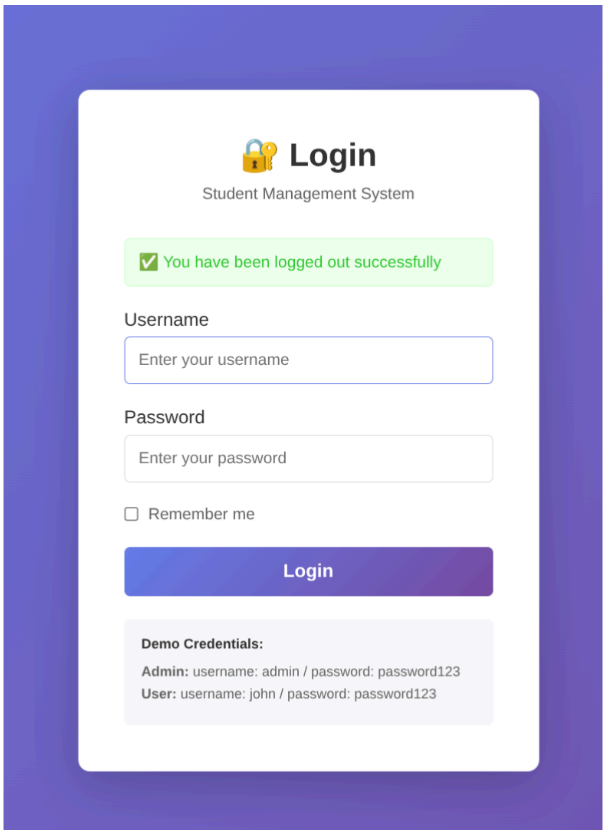
```
        if (session != null) {

            // Invalidate session

            session.invalidate();

        }



        // Redirect to login page with message

        response.sendRedirect("login?message=You have been logged out
successfully");

    }



    @Override

    protected void doPost(HttpServletRequest request, HttpServletResponse
response)

            throws ServletException, IOException {

        doGet(request, response);

    }

}
```

Designed to handle a simple HTTP GET request, which is the standard mechanism for initiating a logout link.

- **Retrieve Session Safely:** The doGet method starts by attempting to retrieve the existing session using HttpSession session = request.getSession(false);. Using false ensures that if no session exists, one is not created, preventing unnecessary overhead.
- **Session Invalidation:**
    - An if (session != null) check ensures a session exists before proceeding.
    - The core action is session.invalidate(), which immediately removes the session and all its stored attributes (like the user object and role) from the server memory. The user is now effectively logged out.
- **Redirect with Feedback:**
    - The final step is to redirect the user back to the login page using response.sendRedirect("login?message=You have been logged out successfully").
    - The use of a query parameter allows the login page (login.jsp) to easily detect and display a user-friendly confirmation message, providing important feedback to the user that the logout process was successful.

- **Handling POST:** The doPost method is included as a safeguard to ensure that if a form submission somehow hits the logout URL, it defaults to the doGet logic, ensuring consistency in the logout process.

Testing logout successfully



🔐 Login

Student Management System

✅ You have been logged out successfully

Username

Enter your username

Password

Enter your password

☐ Remember me

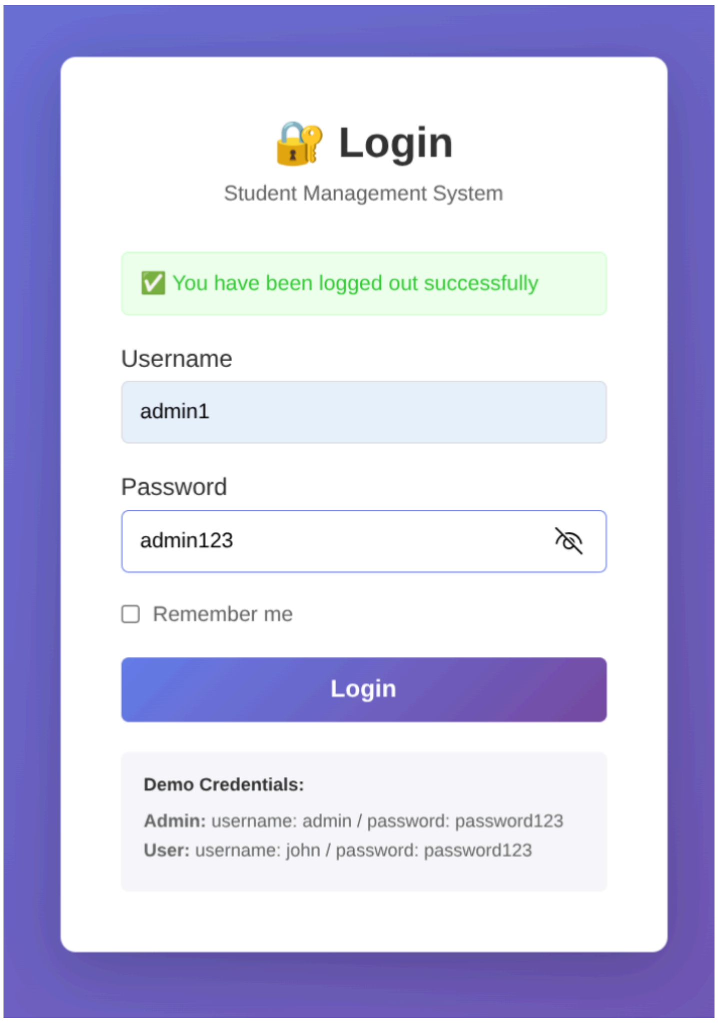
Login

Demo Credentials:

Admin: username: admin / password: password123

User: username: john / password: password123

Testing login with user name is “admin1” and password is “admin123” successfully



🔐 Login

Student Management System

✅ You have been logged out successfully

Username: admin1

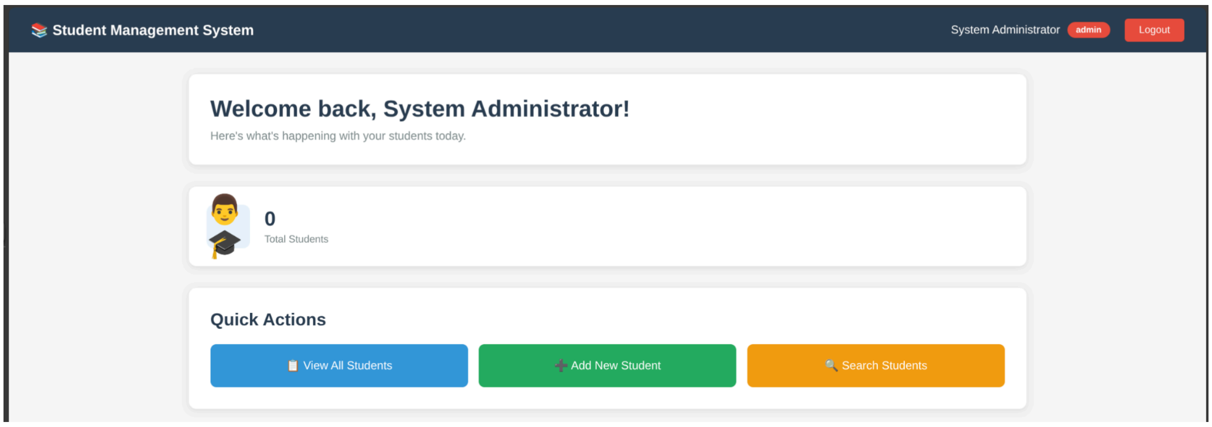
Password: admin123

☐ Remember me

Demo Credentials:

Admin: username: admin / password: password123

User: username: john / password: password123



📚 Student Management System — System Administrator admin

Welcome back, System Administrator!

Here's what's happening with your students today.

0 Total Students

Quick Actions

Task 4.1: Create Login View (8 points)

File: `WebContent/views/login.jsp`

Requirements:

- Professional design
- Display error messages
- Display success messages (from logout)
- Remember username on error
- Demo credentials shown

```
<%@     page     language="java"     contentType="text/html;     charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Login - Student Management System</title>
   <style>
       * {
           margin: 0;
           padding: 0;
           box-sizing: border-box;
       }

       body {
           font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
           background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
           min-height: 100vh;
           display: flex;
           justify-content: center;
           align-items: center;
       }

       .login-container {
           background: white;
           padding: 40px;
           border-radius: 10px;
           box-shadow: 0 10px 40px rgba(0,0,0,0.2);
           width: 100%;
           max-width: 400px;
       }

       .login-header {
           text-align: center;
           margin-bottom: 30px;
       }
```

```css
.login-header h1 {
    color: #333;
    font-size: 28px;
    margin-bottom: 10px;
}

.login-header p {
    color: #666;
    font-size: 14px;
}

.form-group {
    margin-bottom: 20px;
}

.form-group label {
    display: block;
    margin-bottom: 5px;
    color: #333;
    font-weight: 500;
}

.form-group input[type="text"],
.form-group input[type="password"] {
    width: 100%;
    padding: 12px;
    border: 1px solid #ddd;
    border-radius: 5px;
    font-size: 14px;
    transition: border-color 0.3s;
}

.form-group input:focus {
    outline: none;
    border-color: #667eea;
}

.remember-me {
    display: flex;
    align-items: center;
    margin-bottom: 20px;
}

.remember-me input {
    margin-right: 8px;
}

.remember-me label {
    color: #666;
    font-size: 14px;
}
```

```css
    .btn-login {
        width: 100%;
        padding: 12px;
        background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
        color: white;
        border: none;
        border-radius: 5px;
        font-size: 16px;
        font-weight: 600;
        cursor: pointer;
        transition: transform 0.2s;
    }

    .btn-login:hover {
        transform: translateY(-2px);
    }

    .alert {
        padding: 12px;
        border-radius: 5px;
        margin-bottom: 20px;
        font-size: 14px;
    }

    .alert-error {
        background: #fee;
        color: #c33;
        border: 1px solid #fcc;
    }

    .alert-success {
        background: #efe;
        color: #3c3;
        border: 1px solid #cfc;
    }

    .demo-credentials {
        margin-top: 20px;
        padding: 15px;
        background: #f8f9fa;
        border-radius: 5px;
        font-size: 12px;
    }

    .demo-credentials h4 {
        margin-bottom: 10px;
        color: #333;
    }

    .demo-credentials p {
        margin: 5px 0;
        color: #666;
    }
```

```
   </style>
</head>
<body>
<div class="login-container">
   <div class="login-header">
       <h1>🔐 Login</h1>
       <p>Student Management System</p>
   </div>

   <!-- Error Message -->
   <c:if test="${not empty error}">
       <div class="alert alert-error">
           ❌ ${error}
       </div>
   </c:if>

   <!-- Success Message -->
   <c:if test="${not empty param.message}">
       <div class="alert alert-success">
           ✅ ${param.message}
       </div>
   </c:if>

   <!-- Login Form -->
   <form action="login" method="post">
       <div class="form-group">
           <label for="username">Username</label>
           <input type="text"
                  id="username"
                  name="username"
                  value="${username}"
                  placeholder="Enter your username"
                  required
                  autofocus>
       </div>

       <div class="form-group">
           <label for="password">Password</label>
           <input type="password"
                  id="password"
                  name="password"
                  placeholder="Enter your password"
                  required>
       </div>

       <div class="remember-me">
           <input type="checkbox" id="remember" name="remember">
           <label for="remember">Remember me</label>
       </div>

       <button type="submit" class="btn-login">Login</button>
   </form>
   <!-- Demo Credentials -->
   <div class="demo-credentials">
       <h4>Demo Credentials:</h4>
       <p><strong>Admin:</strong> username: admin / password: password123</p>
       <p><strong>User:</strong> username: john / password: password123</p>
   </div>
</div>
</body>
</html>
```

- **Aesthetics and Structure:** The page utilizes a minimal but effective CSS style block to ensure the login form is centered and contained within a well-shadowed, responsive card . The use of a simple linear-gradient background provides visual depth, and the mobile-first design (max-width: 400px) ensures good usability on all screen sizes.
- **Message Handling:** The view uses the JSTL <c:if> tag for conditional rendering of feedback messages.
    - **Error Display:** It checks the Request Scope for the error attribute (set by LoginController on failed login) and displays it in a red alert-error box.
    - **Success Display:** It checks the URL's parameter map (param.message) for the message parameter (set by LogoutController on successful logout) and displays it in a green alert-success box.
- **Form Persistence:** The HTML input for the username includes the dynamic value attribute value="${username}". Since the LoginController places the submitted username back into the Request Scope on authentication failure, this ensures that the user does not have to re-type their username after a failed login attempt, which is a key usability feature.
- **Security and Context:** The form is correctly directed to action="login" with method="post", ensuring credentials are sent securely and processed by the LoginController. A dedicated demo-credentials panel provides necessary context for testers or new users to access the system immediately.

Task 4.2: Create Dashboard (7 points)

File: `src/controller/DashboardController.java`

Requirements:

- Navigation bar with user info and logout
- Welcome message
- Statistics cards
- Quick action buttons
- Role-based UI elements

Dashboard controller

```
package com.student.controller;

import com.student.dao.StudentDAO;
import com.student.model.User;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.io.IOException;

@WebServlet("/dashboard")
public class DashboardController extends HttpServlet {

   private StudentDAO studentDAO;

   @Override
   public void init() {
       studentDAO = new StudentDAO();
   }

   @Override
      protected  void  doGet(HttpServletRequest  request,  HttpServletResponse
response)
           throws ServletException, IOException {

       // Get user from session
       HttpSession session = request.getSession(false);
       if (session == null || session.getAttribute("user") == null) {
           response.sendRedirect("login");
           return;
       }

       User user = (User) session.getAttribute("user");

       // Get statistics
       int totalStudents = studentDAO.getTotalStudents();

       // Set attributes
       request.setAttribute("totalStudents", totalStudents);
               request.setAttribute("welcomeMessage",  "Welcome  back,  "  +
user.getFullName() + "!");

       // Forward to dashboard
        request.getRequestDispatcher("/views/dashboard.jsp").forward(request,
response);
   }
}
```

Dashboard

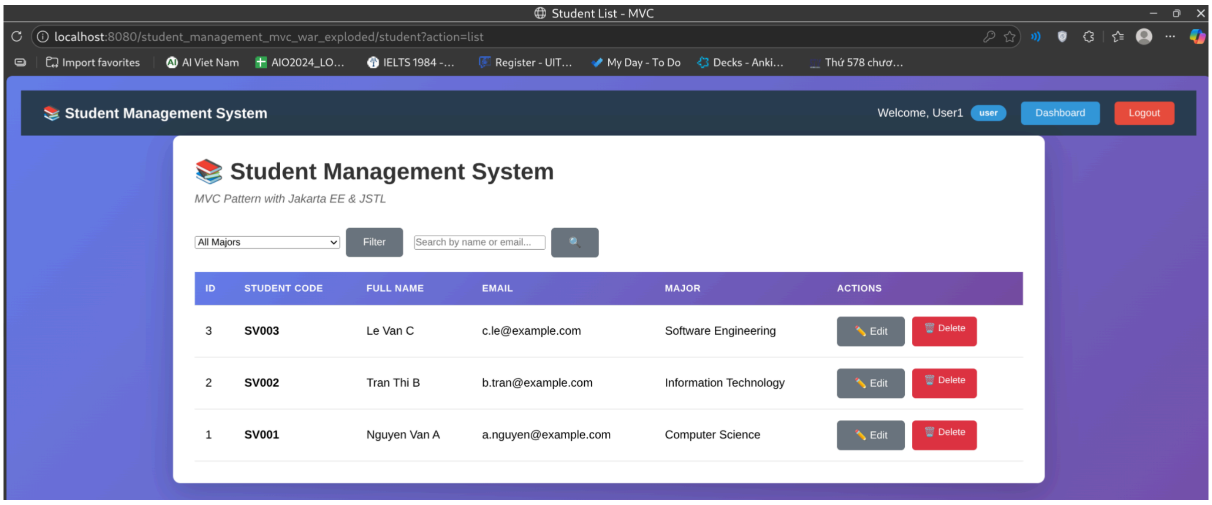
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>Dashboard</title>
   <style>
       * {
           margin: 0;
           padding: 0;
           box-sizing: border-box;
       }

       body {
           font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
           background: #f5f5f5;
       }

       .navbar {
           background: #2c3e50;
           color: white;
           padding: 15px 30px;
           display: flex;
           justify-content: space-between;
           align-items: center;
       }

       .navbar h2 {
           font-size: 20px;
       }

       .navbar-right {
           display: flex;
           align-items: center;
           gap: 20px;
       }

       .user-info {
           display: flex;
           align-items: center;
           gap: 10px;
       }

       .role-badge {
           padding: 4px 12px;
           border-radius: 12px;
           font-size: 12px;
           font-weight: 600;
       }
```

```
        .role-admin {
            background: #e74c3c;
        }

        .role-user {
            background: #3498db;
        }

        .btn-logout {
            padding: 8px 20px;
            background: #e74c3c;
            color: white;
            text-decoration: none;
            border-radius: 5px;
            font-size: 14px;
            transition: background 0.3s;
        }

        .btn-logout:hover {
            background: #c0392b;
        }

        .container {
            max-width: 1200px;
            margin: 30px auto;
            padding: 0 20px;
        }

        .welcome-card {
            background: white;
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
            margin-bottom: 30px;
        }

        .welcome-card h1 {
            color: #2c3e50;
            margin-bottom: 10px;
        }

        .welcome-card p {
            color: #7f8c8d;
        }

        .stats-grid {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
            gap: 20px;
            margin-bottom: 30px;
        }

        .stat-card {
```

```
        background: white;
        padding: 25px;
        border-radius: 10px;
        box-shadow: 0 2px 10px rgba(0,0,0,0.1);
        display: flex;
        align-items: center;
        gap: 20px;
    }

    .stat-icon {
        font-size: 40px;
        width: 60px;
        height: 60px;
        display: flex;
        align-items: center;
        justify-content: center;
        border-radius: 10px;
    }

    .stat-icon-students {
        background: #e8f4fd;
    }

    .stat-content h3 {
        font-size: 28px;
        color: #2c3e50;
        margin-bottom: 5px;
    }

    .stat-content p {
        color: #7f8c8d;
        font-size: 14px;
    }

    .quick-actions {
        background: white;
        padding: 30px;
        border-radius: 10px;
        box-shadow: 0 2px 10px rgba(0,0,0,0.1);
    }

    .quick-actions h2 {
        color: #2c3e50;
        margin-bottom: 20px;
    }

    .action-grid {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
        gap: 15px;
    }

    .action-btn {
            padding: 20px;
            background: #3498db;
            color: white;
            text-decoration: none;
            border-radius: 8px;
            text-align: center;
            transition: all 0.3s;
            display: block;
        }

        .action-btn:hover {
            background: #2980b9;
            transform: translateY(-2px);
        }

        .action-btn-primary {
            background: #3498db;
        }

        .action-btn-success {
            background: #27ae60;
        }

        .action-btn-warning {
            background: #f39c12;
        }
    </style>
</head>
<body>
<!-- Navigation Bar -->
<div class="navbar">
    <h2>📚 Student Management System</h2>
    <div class="navbar-right">
        <div class="user-info">
            <span>${sessionScope.fullName}</span>
            <span class="role-badge role-${sessionScope.role}">
                ${sessionScope.role}
            </span>
        </div>
        <a href="logout" class="btn-logout">Logout</a>
    </div>
</div>

<!-- Main Content -->
<div class="container">
    <!-- Welcome Card -->
    <div class="welcome-card">
        <h1>${welcomeMessage}</h1>
        <p>Here's what's happening with your students today.</p>
    </div>

    <!-- Statistics -->
    <div class="stats-grid">
```

```
        <div class="stat-card">
            <div class="stat-icon stat-icon-students">
                👨‍🎓
            </div>
            <div class="stat-content">
                <h3>${totalStudents}</h3>
                <p>Total Students</p>
            </div>
        </div>
    </div>

    <!-- Quick Actions -->
    <div class="quick-actions">
        <h2>Quick Actions</h2>
        <div class="action-grid">
                            <a  href="student?action=list"  class="action-btn
action-btn-primary">
                📋 View All Students
            </a>

            <c:if test="${sessionScope.role eq 'admin'}">
                              <a  href="student?action=new"  class="action-btn
action-btn-success">
                    ➕ Add New Student
                </a>
            </c:if>

                          <a  href="student?action=search"  class="action-btn
action-btn-warning">
                🔍 Search Students
            </a>
        </div>
    </div>
</div>
</body>
</html>
```

Student dashboard


Student List - MVC

localhost:8080/student_management_mvc_war_exploded/student?action=list

Student Management System — Welcome, User1 user — Dashboard — Logout

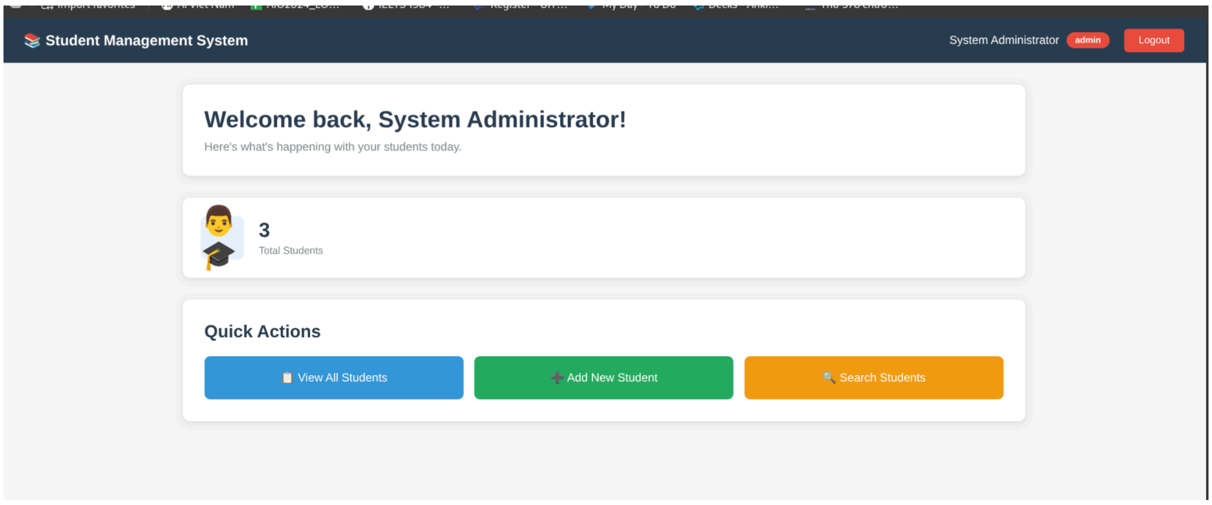
Student Management System

MVC Pattern with Jakarta EE & JSTL

All Majors — Filter — Search by name or email...

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
3	SV003	Le Van C	c.le@example.com	Software Engineering	Edit Delete
2	SV002	Tran Thi B	b.tran@example.com	Information Technology	Edit Delete
1	SV001	Nguyen Van A	a.nguyen@example.com	Computer Science	Edit Delete

Admin dashboard


Student Management System — System Administrator admin — Logout

Welcome back, System Administrator!

Here's what's happening with your students today.

3

Total Students

Quick Actions

View All Students — Add New Student — Search Students

- The login.jsp serves as the user interface for authentication, utilizing JSP Standard Tag Library (JSTL) and CSS to meet all functional and aesthetic requirements.
- **Aesthetics and Structure**: The page employs a minimal yet effective internal CSS style block to create a professional user experience. This styling ensures the login form is centered and contained within a well-shadowed, responsive card . The use of a simple linear-gradient background provides visual depth, and the mobile-first design philosophy, achieved with max-width: 400px, ensures good usability across various screen sizes.
- **Message Handling:** The view leverages the JSTL <c:if> tag for the conditional rendering of feedback messages, making the display logic clean and concise.
    - Error Display: It checks the Request Scope for the ${error} attribute (which the LoginController sets upon authentication or input failure) and displays it prominently in a red alert-error box.
    - Success Display: It checks the URL's parameter map using ${param.message} (which the LogoutController sets upon successful logout) and displays it in a green alert-success box, offering crucial feedback to the user.
- **Form Persistence:** To enhance user experience, the HTML input for the username includes the dynamic value attribute value="${username}". This is a key usability feature because if

the LoginController forwards the request back to the login page after an authentication failure, it places the submitted username back into the Request Scope, ensuring the user does not have to re-type their username.

- **Security and Context:** The form element is correctly configured to post to the action="login" with the method="post" attribute, ensuring credentials are sent securely and processed by the LoginController. Finally, a dedicated demo-credentials panel is included to provide necessary context for immediate testing or use of the application.