EXERCISE 5: AUTHENTICATION FILTER (12 points)

Estimated Time: 45 minutes

Task 5.1: Create AuthFilter (12 points)

File: src/filter/AuthFilter.java

Requirements:

- Check if user is logged in for protected pages
- Allow public URLs (login, logout, static resources)
- Redirect to login if not authenticated
- Use @WebFilter annotation

Testing:

1. Deploy application
2. Try accessing /student without login → Should redirect to login
3. Login successfully
4. Access /student → Should work
5. Access static files (CSS, images) → Should work without login

Final Code

```
package com.student.filter;



import jakarta.servlet.*;

import jakarta.servlet.annotation.WebFilter;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import jakarta.servlet.http.HttpSession;

import java.io.IOException;



/**

* Authentication Filter - Checks if user is logged in

* Protects all pages except login and public resources
```

```
*/

@WebFilter(filterName = "AuthFilter", urlPatterns = {"/*"})

public class AuthFilter implements Filter {



   // Public URLs that don't require authentication

   private static final String[] PUBLIC_URLS = {

           "/login",

           "/logout",

           ".css",

           ".js",

           ".png",

           ".jpg",

           ".jpeg",

           ".gif"

   };



   @Override

   public void init(FilterConfig filterConfig) throws ServletException {

       System.out.println("AuthFilter initialized");

   }



   @Override

   public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)

           throws IOException, ServletException {
```

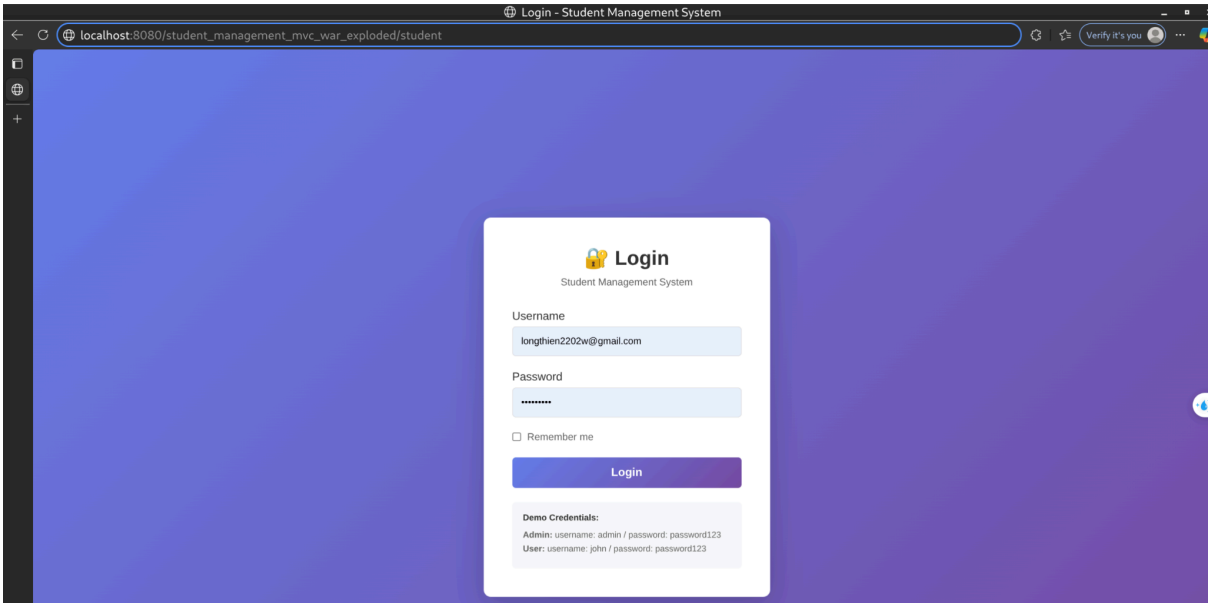
```
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        String requestURI = httpRequest.getRequestURI();
        String contextPath = httpRequest.getContextPath();
        String path = requestURI.substring(contextPath.length());

        // Check if this is a public URL
        if (isPublicUrl(path)) {
            // Allow access to public URLs
            chain.doFilter(request, response);
            return;
        }

        // Check if user is logged in
        HttpSession session = httpRequest.getSession(false);
        boolean isLoggedIn = (session != null &&
session.getAttribute("user") != null);

        if (isLoggedIn) {
            // User is logged in, allow access
            chain.doFilter(request, response);
        } else {
            // User not logged in, redirect to login
            String loginURL = contextPath + "/login";
            httpResponse.sendRedirect(loginURL);
```

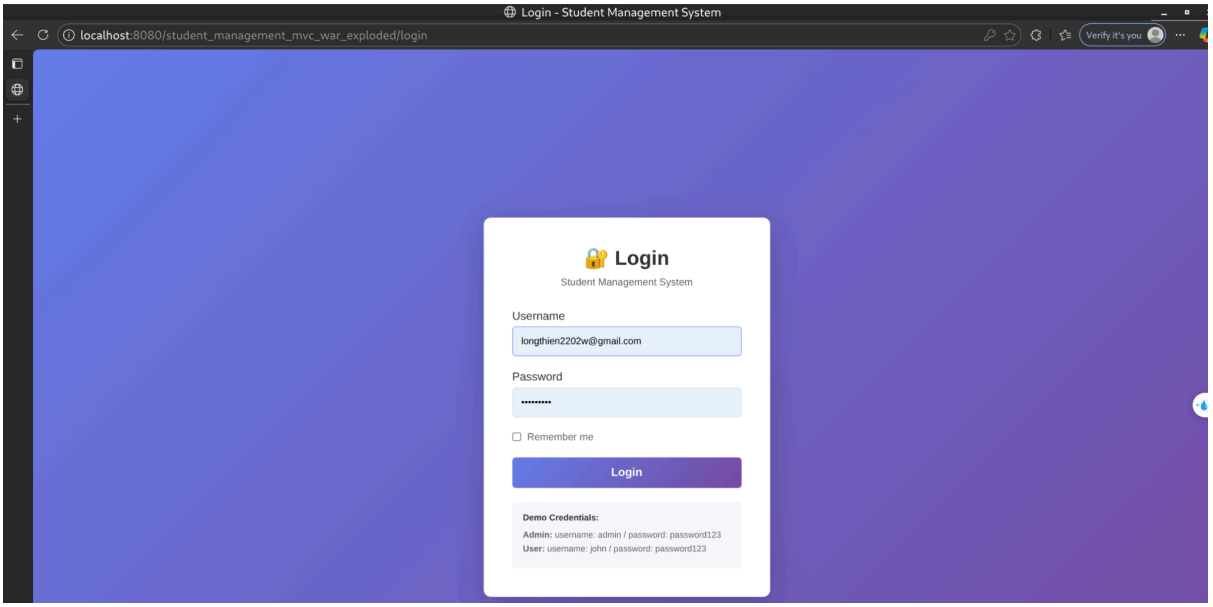
```
        }
    }

    @Override
    public void destroy() {
        System.out.println("AuthFilter destroyed");
    }

    /**
     * Check if URL is public (doesn't require authentication)
     */
    private boolean isPublicUrl(String path) {
        for (String publicUrl : PUBLIC_URLS) {
            if (path.contains(publicUrl)) {
                return true;
            }
        }
        return false;
    }
}
```

1. Intercept: The filter catches every HTTP request (/*) coming into the server.
2. Identify: It calculates the specific resource path (e.g., /student or /login).
3. Whitelist Check: It checks if the path contains public keywords (Login page, CSS, JS, Images).
    - YES (Public): The request is allowed through immediately.
    - NO (Protected): The code proceeds to the login check.
4. Login Check:
    - It attempts to retrieve the User object from the Session.
    - User Found: The user is logged in. The request is allowed through (chain.doFilter).

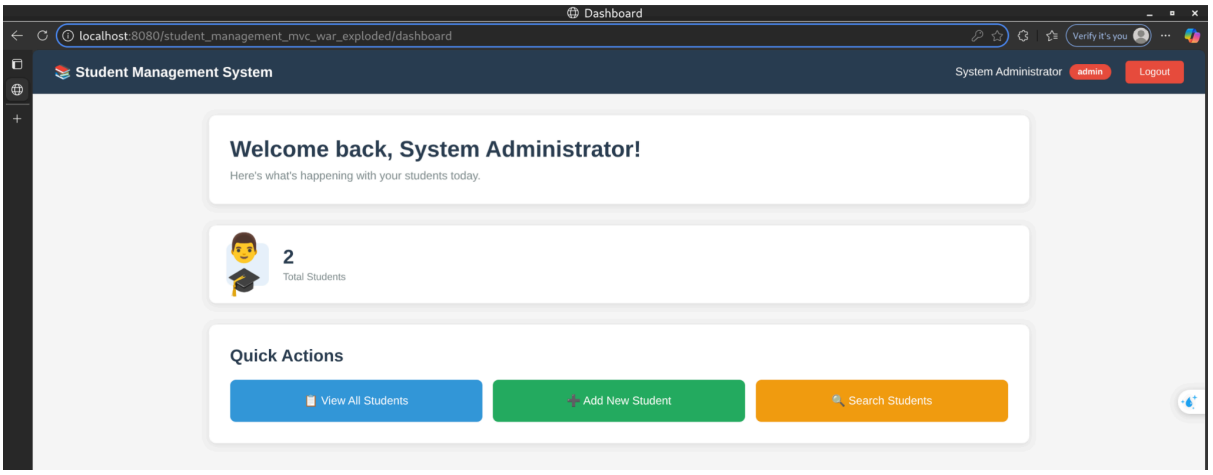
- User Not Found: The request is blocked, and the user is forced to redirect to the /login page.

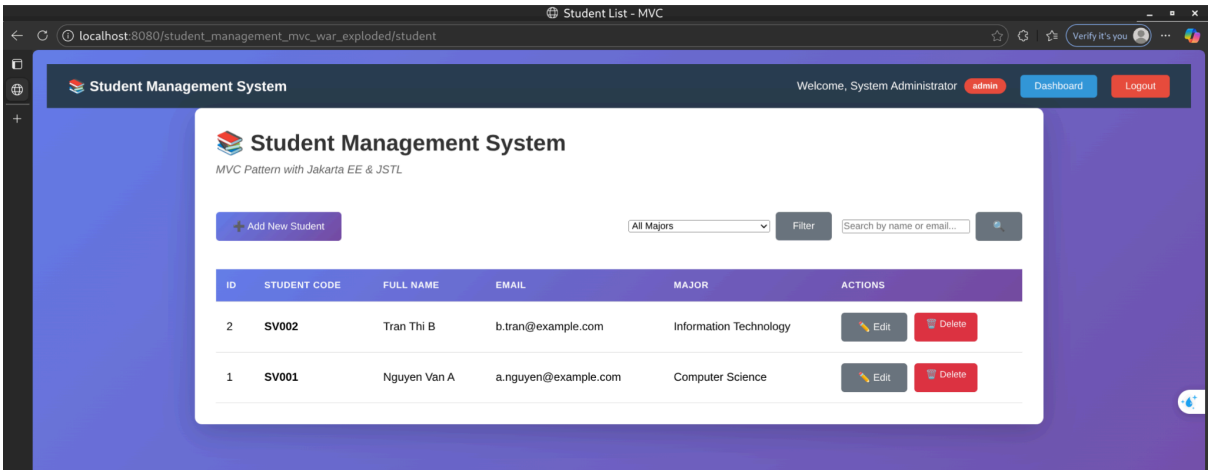
Change the url to /student, get redirect to login





Login successfully, change the url to /student and get redirect to student





EXERCISE 6: ADMIN AUTHORIZATION FILTER (10 points)

Estimated Time: 40 minutes

Task 6.1: Create AdminFilter (10 points)

File: src/filter/AdminFilter.java

Requirements:

- Check if user has admin role for admin actions
- Block non-admin users from creating/editing/deleting
- Show error message on denial

Testing:

1. Login as admin → Try edit/delete → Should work
2. Logout
3. Login as regular user → Try edit/delete → Should be blocked
4. Try direct URL: /student?action=delete&id=1 → Should be blocked

Final Code

```
package com.student.filter;


import com.student.model.User;


import jakarta.servlet.*;
import jakarta.servlet.annotation.WebFilter;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.io.IOException;


/**
* Admin Filter - Checks if user has admin role
```

```
* Protects admin-only pages
*/
@WebFilter(filterName = "AdminFilter", urlPatterns = {"/student"})
public class AdminFilter implements Filter {


  // Admin-only actions
  private static final String[] ADMIN_ACTIONS = {
          "new",
          "insert",
          "edit",
          "update",
          "delete"
  };


  @Override
  public void init(FilterConfig filterConfig) throws ServletException {
      System.out.println("AdminFilter initialized");
  }


  @Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
          throws IOException, ServletException {


      HttpServletRequest httpRequest = (HttpServletRequest) request;
      HttpServletResponse httpResponse = (HttpServletResponse) response;
```

```
String action = httpRequest.getParameter("action");

// Check if this action requires admin role
if (isAdminAction(action)) {
    HttpSession session = httpRequest.getSession(false);

    if (session != null) {
        User user = (User) session.getAttribute("user");

        if (user != null && user.isAdmin()) {
            // User is admin, allow access
            chain.doFilter(request, response);
        } else {
            // User is not admin, deny access
            httpResponse.sendRedirect(httpRequest.getContextPath() +
                    "/student?action=list&error=Access denied. Admin privileges required.");
        }
    } else {
        // No session, redirect to login
        httpResponse.sendRedirect(httpRequest.getContextPath() + "/login");
    }
} else {
    // Not an admin action, allow access
    chain.doFilter(request, response);
```

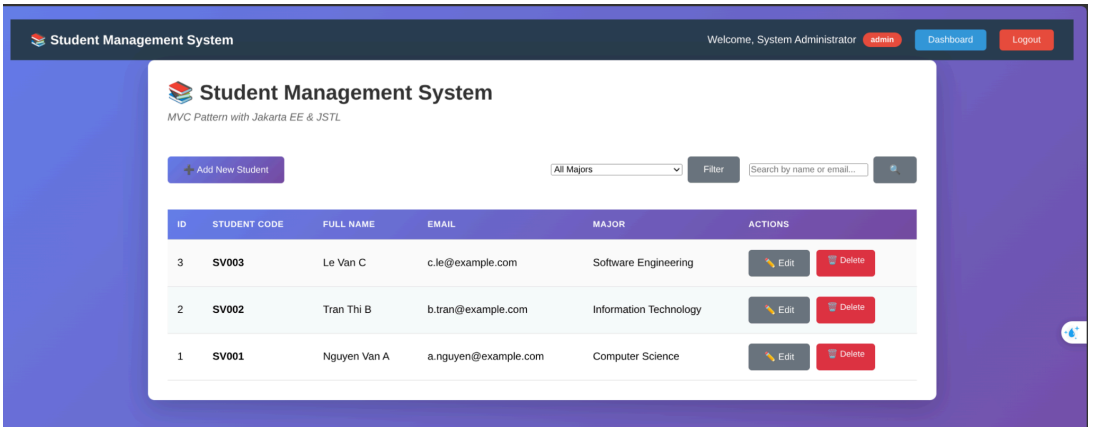
```
    }
}

@Override
public void destroy() {
    System.out.println("AdminFilter destroyed");
}

/**
 * Check if action requires admin role
 */
private boolean isAdminAction(String action) {
    if (action == null) return false;

    for (String adminAction : ADMIN_ACTIONS) {
        if (adminAction.equals(action)) {
            return true;
        }
    }
    return false;
}
}
```

1. Intercept: The filter catches any HTTP request going to the URL /student.
2. Identify: It extracts the action parameter from the request (e.g., ?action=delete).
3. Evaluate: It checks if the action is in the restricted list (new, insert, edit, update, delete).

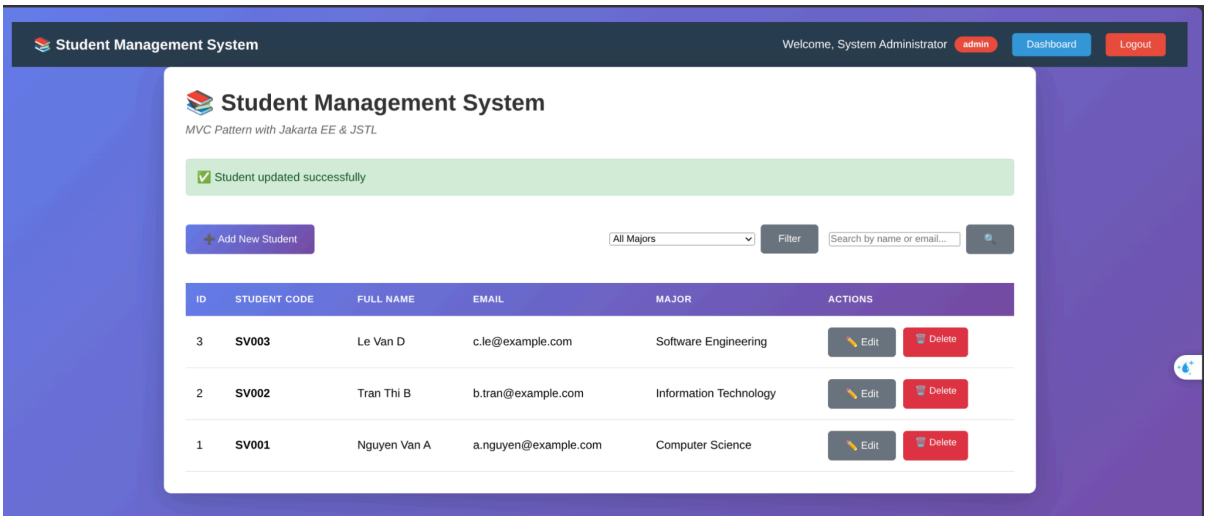
- NO (Safe Action): The request is allowed through immediately (chain.doFilter).
- YES (Restricted Action): The code proceeds to the security check.

4. Security Check:
   - If No Session: The user is redirected to the /login page.
   - If Session Exists: It retrieves the User object.
     - User is Admin: The request is allowed through (chain.doFilter).
     - User is NOT Admin: The request is blocked, and the user is redirected to the student list with an "Access Denied" error.

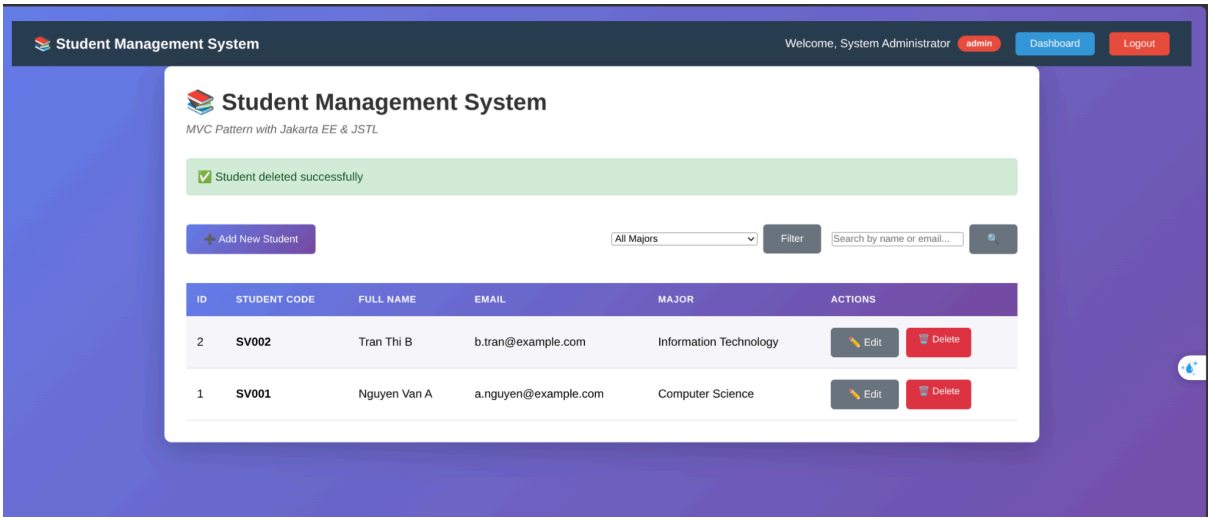
Login as admin



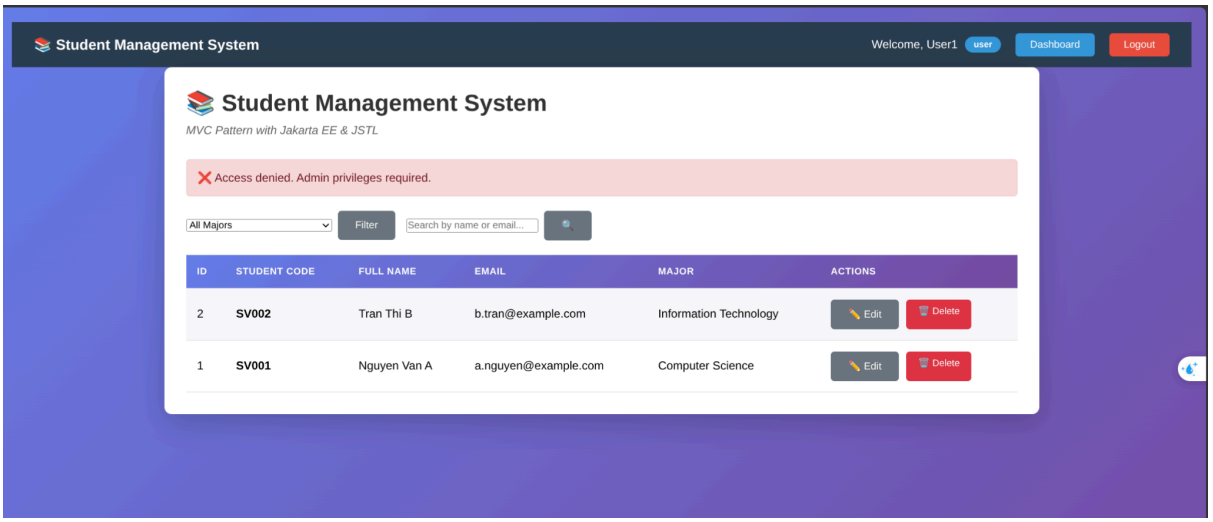
Edit student successfully



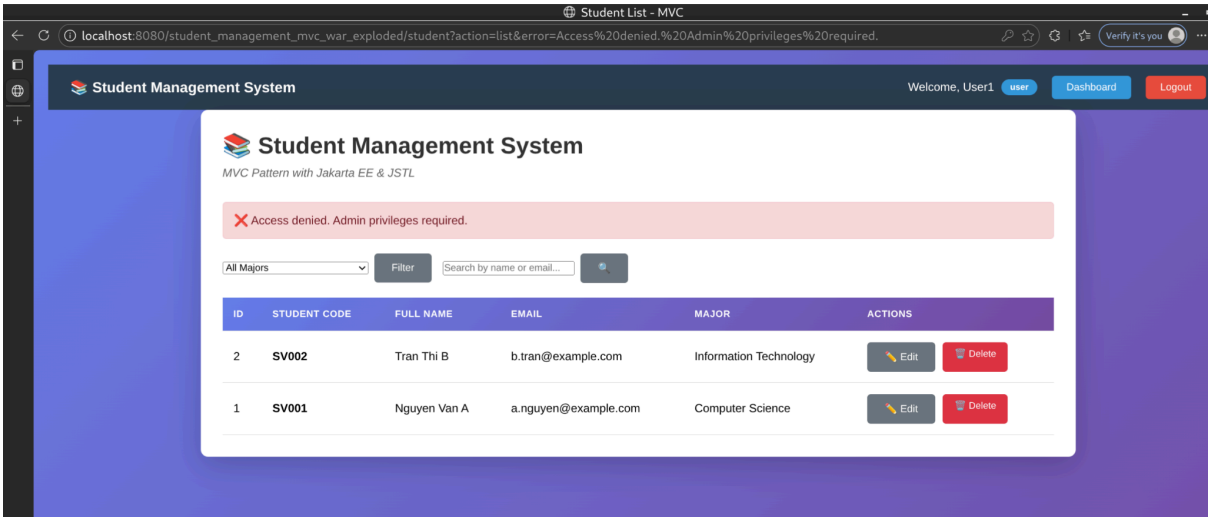
Delete Student Successfully



Login as regular user and action edit and delete will be block



Try direct URL: /student?action=delete&id=1 and get blocked



EXERCISE 7: ROLE-BASED UI (10 points)

Estimated Time: 35 minutes

Task 7.1: Update Student List View (10 points)

File: WebContent/views/student-list.jsp

Requirements:

- Add navigation bar with user info
- Show role badge
- Hide "Add Student" button for non-admins
- Hide Edit/Delete buttons for non-admins
- Show error messages from AdminFilter

Final code

```
<body>

<div class="navbar">

   <h2>📚 Student Management System</h2>

   <div class="navbar-right">

       <div class="user-info">

           <span>Welcome, ${sessionScope.fullName}</span>

           <span class="role-badge role-${sessionScope.role}">

               ${sessionScope.role}

           </span>

       </div>

       <a href="dashboard" class="btn-nav">Dashboard</a>

       <a href="logout" class="btn-logout">Logout</a>

   </div>

</div>


<div class="container">

   <h1>📚 Student Management System</h1>

   <p class="subtitle">MVC Pattern with Jakarta EE & JSTL</p>
```

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

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

    <!-- Toolbar -->
    <div class="toolbar">
        <c:if test="${sessionScope.role eq 'admin'}">
            <div style="margin: 20px 0;">
                <a href="student?action=new" class="btn btn-primary">➕ Add New Student</a>
            </div>
        </c:if>

        <div style="display: flex; gap: 15px; flex-wrap: wrap;">
            <div class="filter-box">
                <form action="student" method="get">
                    <input type="hidden" name="action" value="filter">
```

```
                    <select name="major" class="filter-select"
onchange="this.form.submit()">

                        <option value="">All Majors</option>

                        <option value="Computer Science" ${currentMajor ==
'Computer Science' ? 'selected' : ''}>Computer Science</option>

                        <option value="Information Technology"
${currentMajor == 'Information Technology' ? 'selected' : ''}>Information
Technology</option>

                        <option value="Software Engineering" ${currentMajor
== 'Software Engineering' ? 'selected' : ''}>Software Engineering</option>

                        <option value="Business Administration"
${currentMajor == 'Business Administration' ? 'selected' : ''}>Business
Administration</option>

                    </select>

                    <button type="submit" class="btn
btn-secondary">Filter</button>

                    <c:if test="${not empty currentMajor}">

                        <a href="student?action=list" class="btn
btn-outline">Clear</a>

                    </c:if>

                </form>

            </div>

            <div class="search-box">

                <form action="student" method="get">

                    <input type="hidden" name="action" value="search">

                    <input type="text"

                           name="keyword"

                           class="search-input"

                           value="${keyword}"

                           placeholder="Search by name or email...">

                    <button type="submit" class="btn
btn-secondary">🔍</button>

                    <c:if test="${not empty keyword}">
```

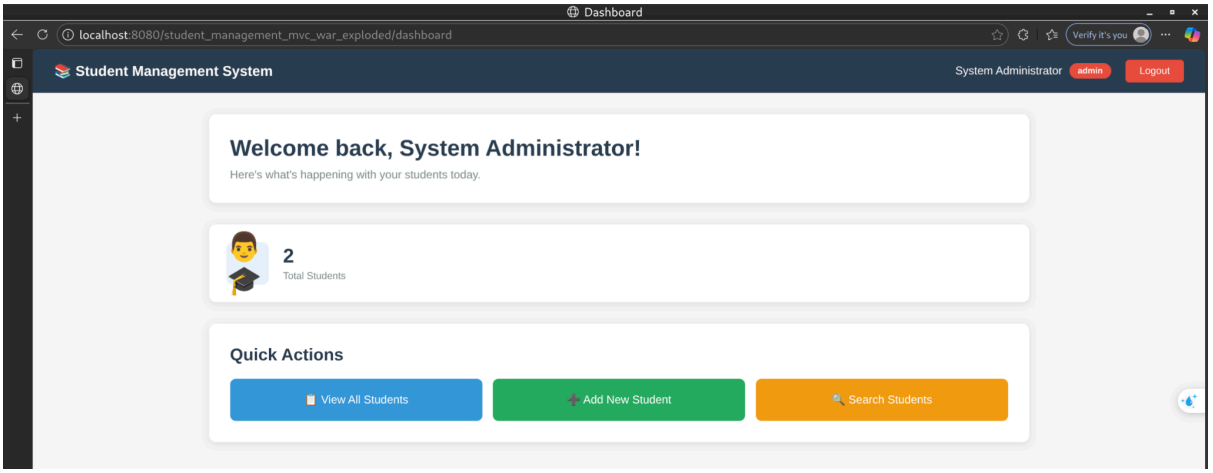
```
                        <a href="student?action=list" class="btn
btn-outline">Clear</a>

                    </c:if>

                </form>

            </div>

        </div>

    </div>


    <!-- Search Feedback Message -->

    <c:if test="${not empty keyword}">

        <div class="message info">

            Search results for: <strong>${keyword}</strong>

        </div>

    </c:if>


    <!-- Student Table -->

    <c:choose>

        <c:when test="${not empty students}">

            <table>

                <thead>

                <tr>

                    <!-- ID Column Sort -->

                    <th>

                        <c:set var="newOrder" value="${sortBy == 'id' &&
order == 'asc' ? 'desc' : 'asc'}" />

                        <a
href="student?action=sort&sortBy=id&order=${newOrder}">

                            ID <c:if test="${sortBy == 'id'}">${order ==
'asc' ? '▲' : '▼'}</c:if>

                        </a>
```

```
                    </th>



                    <!-- Code Column Sort -->

                    <th>

                        <c:set var="newOrder" value="${sortBy ==
'student_code' && order == 'asc' ? 'desc' : 'asc'}" />

                        <a
href="student?action=sort&sortBy=student_code&order=${newOrder}">

                            Student Code <c:if test="${sortBy ==
'student_code'}">${order == 'asc' ? '▲' : '▼'}</c:if>

                        </a>

                    </th>



                    <!-- Name Column Sort -->

                    <th>

                        <c:set var="newOrder" value="${sortBy == 'full_name'
&& order == 'asc' ? 'desc' : 'asc'}" />

                        <a
href="student?action=sort&sortBy=full_name&order=${newOrder}">

                            Full Name <c:if test="${sortBy ==
'full_name'}">${order == 'asc' ? '▲' : '▼'}</c:if>

                        </a>

                    </th>



                    <!-- Email Column Sort -->

                    <th>

                        <c:set var="newOrder" value="${sortBy == 'email' &&
order == 'asc' ? 'desc' : 'asc'}" />

                        <a
href="student?action=sort&sortBy=email&order=${newOrder}">

                            Email <c:if test="${sortBy == 'email'}">${order
== 'asc' ? '▲' : '▼'}</c:if>
```

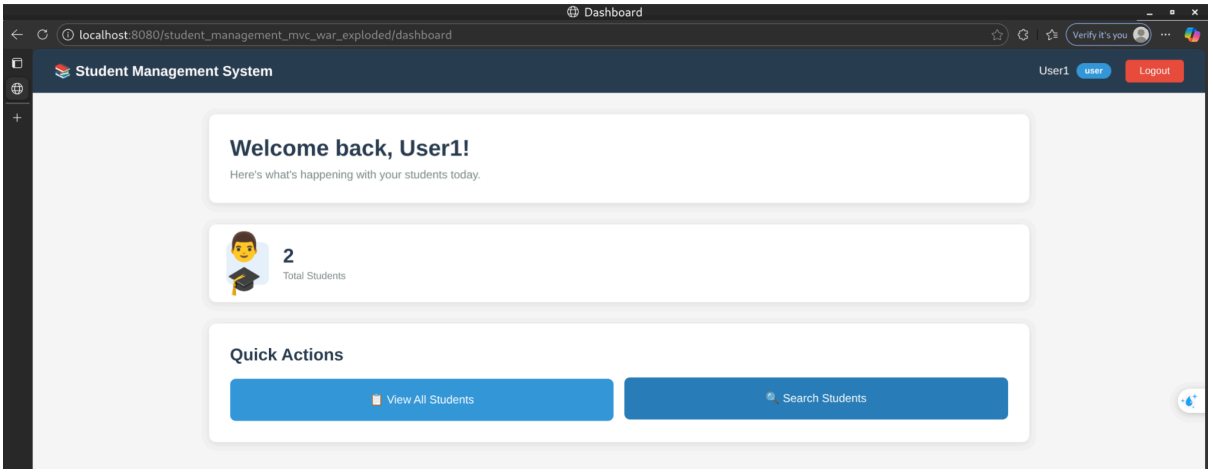
```
                        </a>
                    </th>

                    <!-- Major Column Sort -->
                    <th>
                        <c:set var="newOrder" value="${sortBy == 'major' &&
order == 'asc' ? 'desc' : 'asc'}" />
                        <a
href="student?action=sort&sortBy=major&order=${newOrder}">
                            Major <c:if test="${sortBy == 'major'}">${order
== 'asc' ? '▲' : '▼'}</c:if>
                        </a>
                    </th>

                    <th>Actions</th>
                </tr>
                </thead>
                <tbody>
                <c:forEach var="student" items="${students}">
                    <tr>
                        <td>${student.id}</td>
                        <td><strong>${student.studentCode}</strong></td>
                        <td>${student.fullName}</td>
                        <td>${student.email}</td>
                        <td>${student.major}</td>
                        <td>
                            <div class="actions">
                                <a
href="student?action=edit&id=${student.id}" class="btn btn-secondary">
                                    ✏️ Edit
```

```
                                </a>

                                <a
href="student?action=delete&id=${student.id}"

                                   class="btn btn-danger"

                                   onclick="return confirm('Are you sure you
want to delete this student?')">

                                    🗑️ Delete

                                </a>

                            </div>

                        </td>

                    </tr>

                </c:forEach>

                </tbody>

            </table>

        </c:when>

        <c:otherwise>

            <div class="empty-state">

                <div class="empty-state-icon">📭</div>

                <h3>No students found</h3>

                <c:if test="${not empty currentMajor}">

                    <p>Current filter: <strong>${currentMajor}</strong>. <a
href="student?action=list">Clear filter</a></p>

                </c:if>

            </div>

        </c:otherwise>

    </c:choose>

</div>

</body>

</html>
```

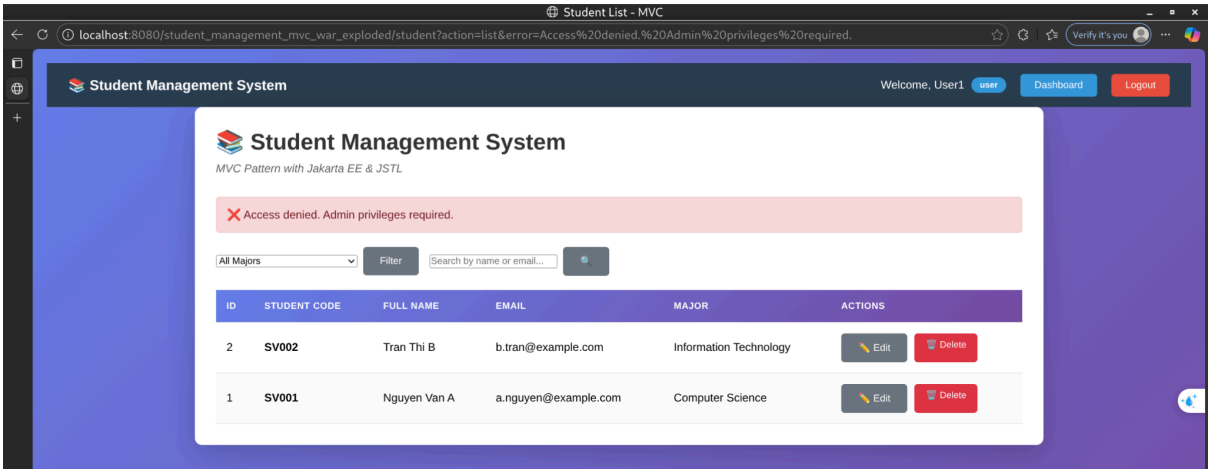
Login as admin (Dashboard)



Login as normal user (Hide add student button on dashboard)



Admin filter work successfully (normal user cannot edit or remove student)



EXERCISE 8: CHANGE PASSWORD (8 points) - Optional

Task 8.1: Implement Change Password Feature

Requirements:

- New page: `change-password.jsp`
- Validate current password
- Validate new password (minimum 8 characters)
- Confirm new password matches
- Hash and update password

1. Controller: `src/controller/ChangePasswordController.java`

```
package com.student.controller;

import com.student.dao.UserDAO;
import com.student.model.User;
import org.mindrot.jbcrypt.BCrypt;

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("/change-password")
public class ChangePasswordController extends HttpServlet {

   private UserDAO userDAO;

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

   protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
       HttpSession session = request.getSession(false);
       if (session == null || session.getAttribute("user") == null) {
           response.sendRedirect("login");
           return;
       }

request.getRequestDispatcher("views/change-password.jsp").forward(request,
response);
   }
```

```
   protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
       HttpSession session = request.getSession(false);
       if (session == null || session.getAttribute("user") == null) {
           response.sendRedirect("login");
           return;
       }

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

       String currentPassword = request.getParameter("currentPassword");
       String newPassword = request.getParameter("newPassword");
       String confirmPassword = request.getParameter("confirmPassword");

       String errorMessage = null;

       if (currentPassword == null || newPassword == null || confirmPassword
== null) {
           errorMessage = "All fields are required";
       } else if (newPassword.length() < 8) {
           errorMessage = "New password must be at least 8 characters";
       } else if (!newPassword.equals(confirmPassword)) {
           errorMessage = "New password and confirmation do not match";
       } else if (!BCrypt.checkpw(currentPassword, user.getPassword())) {
           errorMessage = "Current password is incorrect";
       }

       if (errorMessage != null) {
           request.setAttribute("error", errorMessage);

request.getRequestDispatcher("views/change-password.jsp").forward(request,
response);
           return;
       }

       String hashedNewPassword = BCrypt.hashpw(newPassword,
BCrypt.gensalt());

       if (userDAO.updatePassword(user.getId(), hashedNewPassword)) {
           user.setPassword(hashedNewPassword);
           session.setAttribute("user", user);
           request.setAttribute("success", "Password changed successfully!");
       } else {
           request.setAttribute("error", "Database error occurred, please try
again.");
       }


request.getRequestDispatcher("views/change-password.jsp").forward(request,
response);
   }
}
```

2. View: `WebContent/views/change-password.jsp`

```
<%@ 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>Change Password</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;
   }

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

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

   .header h1 {
     color: #333;
     font-size: 24px;
     margin-bottom: 10px;
   }

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

   .form-group label {
     display: block;
```

```
    margin-bottom: 5px;
    color: #333;
    font-weight: 500;
}

.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;
}

.btn-submit {
    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-submit: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;
```

```
    }

    .back-link {
      display: block;
      text-align: center;
      margin-top: 20px;
      color: #666;
      text-decoration: none;
      font-size: 14px;
    }

    .back-link:hover {
      color: #667eea;
    }
  </style>
</head>
<body>
<div class="container">
  <div class="header">
    <h1>Change Password</h1>
  </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 success}">
    <div class="alert alert-success">
      ✅ ${success}
    </div>
  </c:if>

  <form action="change-password" method="post">
    <div class="form-group">
      <label>Current Password</label>
      <input type="password" name="currentPassword" required>
    </div>

    <div class="form-group">
      <label>New Password (min 8 chars)</label>
      <input type="password" name="newPassword" required>
    </div>

    <div class="form-group">
      <label>Confirm New Password</label>
      <input type="password" name="confirmPassword" required>
    </div>

    <button type="submit" class="btn-submit">Update Password</button>
```

```
 </form>

 <a href="dashboard" class="back-link">← Back to Dashboard</a>
</div>
</body>
</html>
```

3. DAO Method: Add to `UserDAO.java`

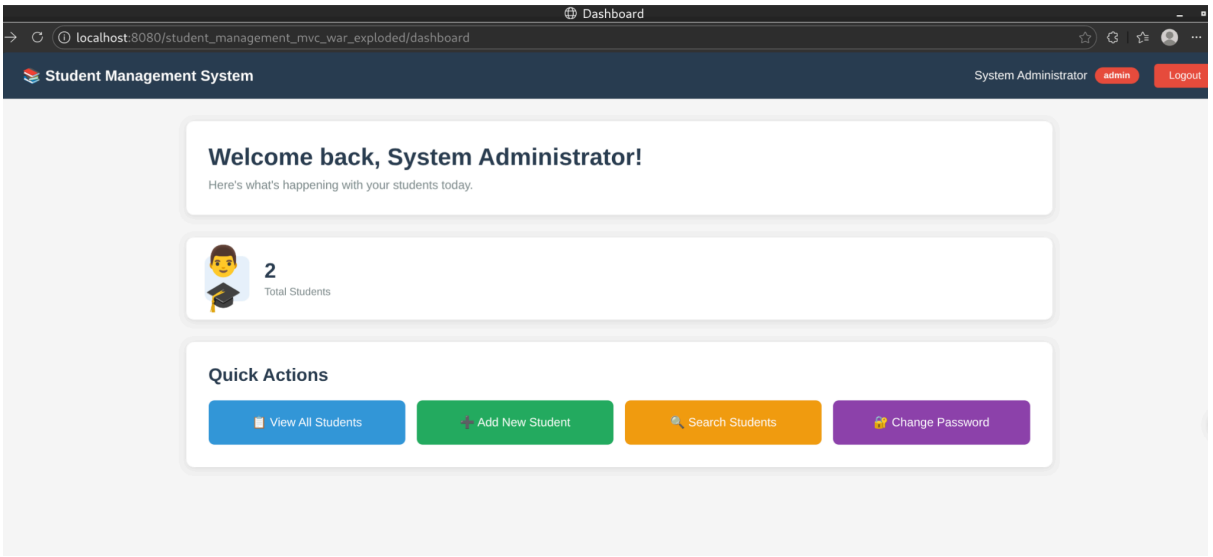
```
public boolean updatePassword(int userId, String newHashedPassword) {
   try (Connection conn = getConnection();
        PreparedStatement pstmt =
conn.prepareStatement(SQL_UPDATE_PASSWORD)) {

       pstmt.setString(1, newHashedPassword);
       pstmt.setInt(2, userId);

       return pstmt.executeUpdate() > 0;

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

Code flow:

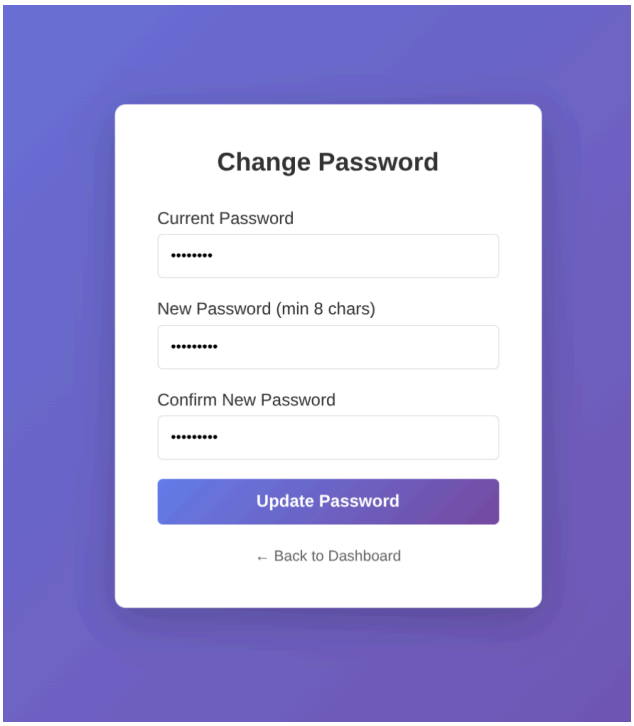
1. Intercept (Request Handling):
    - The ChangePasswordController (mapped to /change-password) catches the HTTP POST request when the user clicks "Update Password".
    - It retrieves the current session using request.getSession(false).
2. Security Check (Session Validation):
    - If No Session/User: The code checks if session is null or user attribute is missing. If true, the user is immediately redirected to the /login page.
    - If Session Exists: The code retrieves the current User object from the session to identify *who* is trying to change the password.
3. Input Validation (Data Integrity):
    - The controller extracts three parameters: currentPassword, newPassword, and confirmPassword.
    - Basic Checks: It validates that fields are not empty and newPassword is at least 8 characters.
    - Matching Check: It verifies that newPassword equals confirmPassword.
4. Cryptographic Verification (Current Password):
    - Before allowing a change, the system must prove the user is the owner.
    - It uses BCrypt.checkpw(currentPassword, user.getPassword()).
    - It compares the plain text password entered in the form against the hashed password stored in the User object.
5. Execution (Hash & Update):

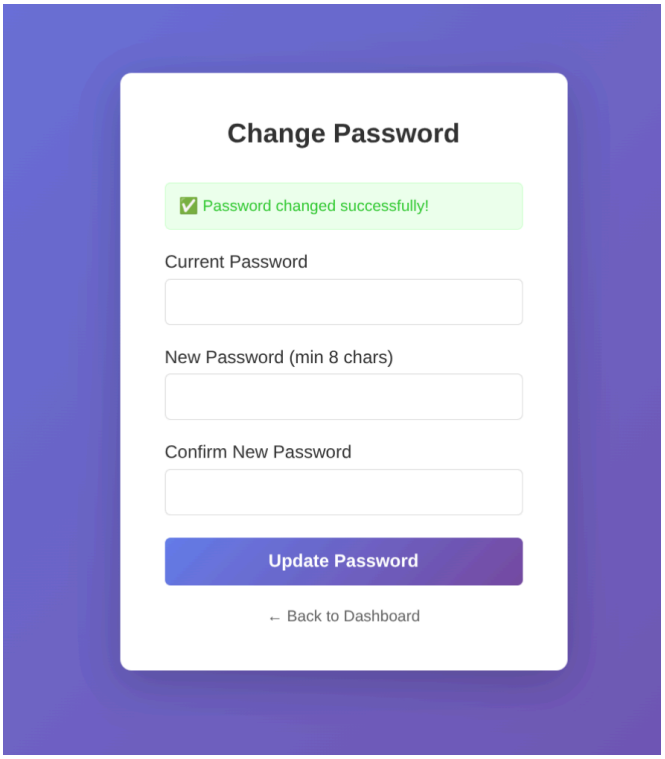
    - Hashing: If all validations pass, the system generates a new salt and hashes the newPassword using BCrypt.hashpw().
    - Database Update: It calls userDAO.updatePassword(userId, newHashedPassword) to save the new hash into the database.
    - Session Update: The User object in the current session is updated with the new password so the user remains logged in without issues.
6. Response:
    - Success: A success message is sent to the JSP, and the user stays on the page.
    - Failure: If any validation fails (wrong current password, short password, etc.), an error message is forwarded back to change-password.jsp to alert the user.

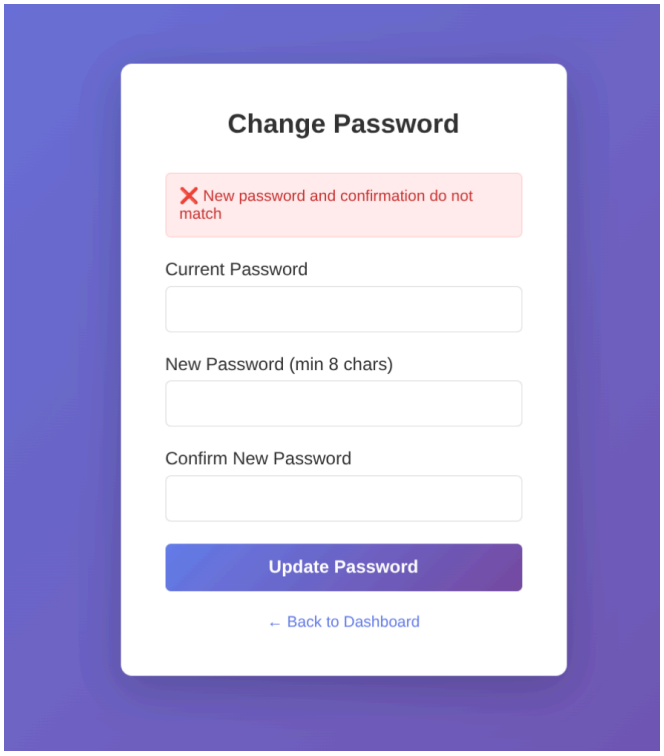
Change password button added into the dashboard

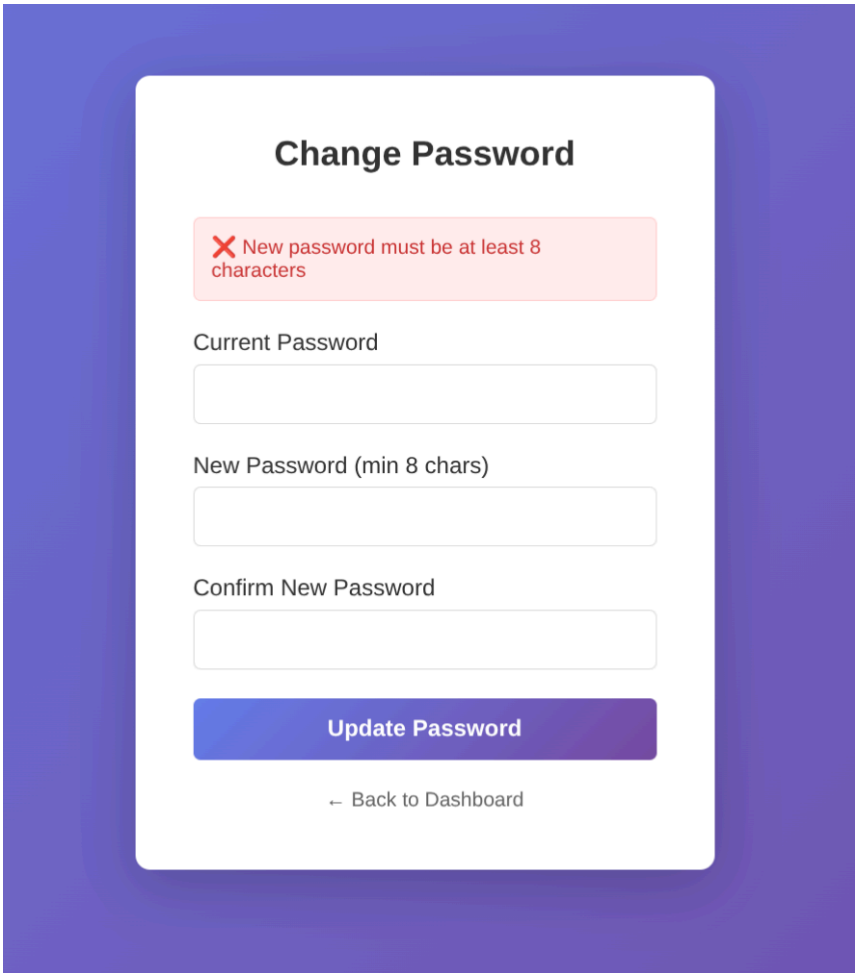


Change password UI




Change Password
✅ Password changed successfully!
Current Password
New Password (min 8 chars)
Confirm New Password
Update Password
← Back to Dashboard


Change Password
❌ New password and confirmation do not match
Current Password
New Password (min 8 chars)
Confirm New Password
Update Password
← Back to Dashboard


Change Password
❌ New password must be at least 8 characters
Current Password
New Password (min 8 chars)
Confirm New Password
Update Password
← Back to Dashboard