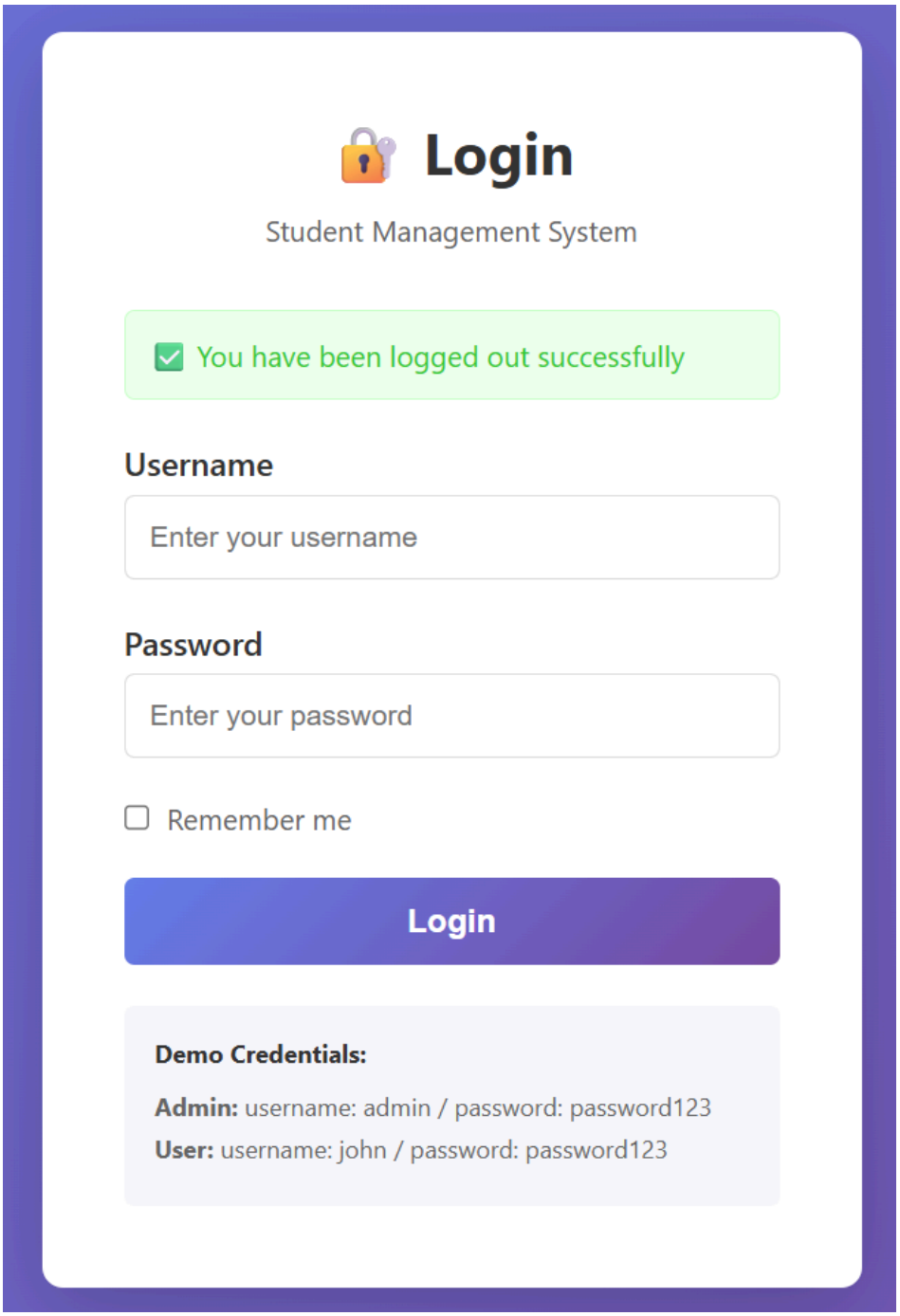
Trần Phúc Quý - ITITDK23027

Lab 6



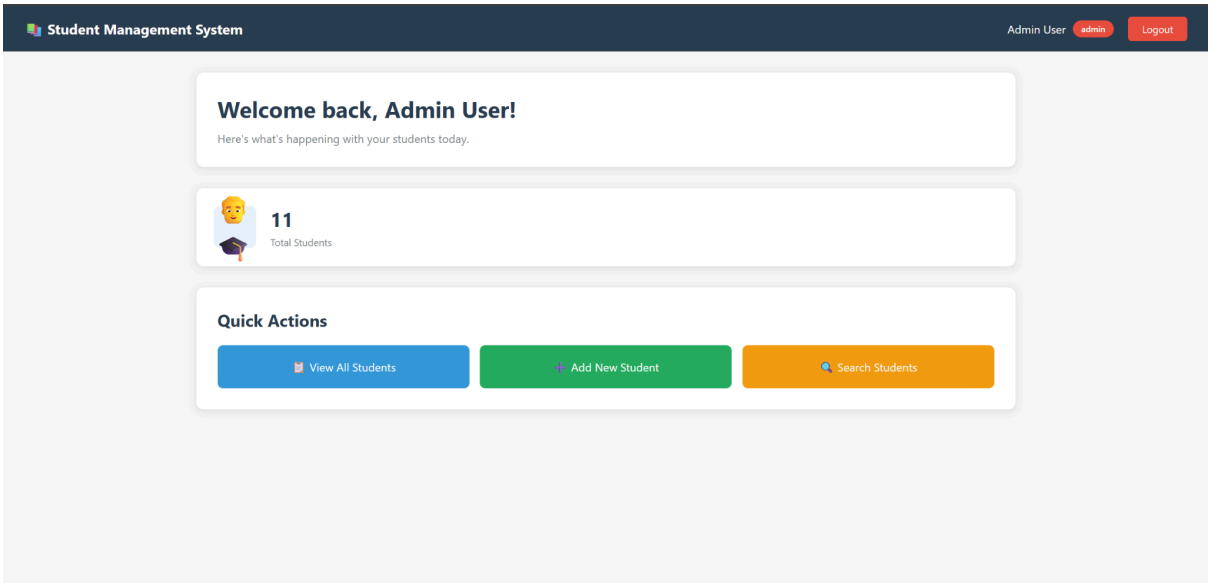


Figure 1 – Login form, Admin page

When the user opens the login page (URL /login), the browser sends a GET request to the server. The request first passes through AuthFilter. Because /login is defined as a public URL, the filter does not require a session and simply forwards the request down the chain. The request then reaches LoginController.doGet.

Inside doGet, the controller usually checks whether a user is already logged in by looking for a user attribute in the session. If a user is found, it can redirect directly to the dashboard or student list instead of showing the login page again. If no user is logged in, LoginController forwards the request to login.jsp, which produces the HTML form you see in this screenshot.

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

Figure 2 – Logincontroller And Userdao

When the user submits the login form, the browser sends a POST /login request with username and password. Again, the request passes through AuthFilter, but since /login is public, it is allowed through without a session check. The request arrives at LoginController.doPost.

In doPost, the controller reads the username and password from the request and then calls UserDAO.authenticate(username, password). The authenticate method queries the users table by username. If a user row is found, it compares the raw password from the form with the hashed password stored in the database using BCrypt.checkpw. If the comparison matches, a User object is returned; if not, null is returned.

If UserDAO.authenticate returns null, the controller sets an error message and forwards back to login.jsp so the user can try again. If it returns a valid user, the controller invalidates any existing session, creates a new session, stores user, role and fullName in the session, and then redirects. Typically, ADMIN users are redirected to /dashboard, and normal USER accounts are redirected to /student?action=list. This new session is the basis for all later security checks.

```java
@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);
    }
}
```

```java
@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);
    }
}
```

Figure 3 – Authfilter And Adminfilter

For all other URLs, the request first goes through AuthFilter. In the code, this filter checks the request path. If the path is a public resource (login, logout, CSS, JS, images), it continues without any login requirement. For any other path, it looks up the user attribute in the session. If there is no user, it redirects the browser to /login, so controllers and JSPs behind protected URLs are never executed by anonymous visitors.

After AuthFilter, requests to /student pass through AdminFilter. In the code, AdminFilter reads the action parameter from the request. Actions like list, search, sort, and filter are considered read-only, so they are allowed for any logged-in user. Actions such as new, insert, edit, update, and delete are considered admin-only. For these actions, the filter reads the role from the session. If role is ADMIN, it lets the request continue to StudentController. If the

role is USER, the filter blocks the request, usually by redirecting to student?action=list with an access denied message. This ensures that user accounts cannot perform write operations even if they manually type the URL.

After a request passes through the filters, it reaches the appropriate controller. For example, /student?action=list is handled by StudentController. In the code, StudentController reads the action parameter, and for list it calls a method such as listStudents, which then uses studentDAO.getAllStudents().

Inside StudentDAO, the code opens a database connection, creates a PreparedStatement with a SELECT query, executes it, and maps each row of the ResultSet into a Student object. It returns a list of students to the controller. The controller then stores this list in the request with setAttribute and forwards to student-list.jsp.

The JSP file uses JSTL to loop over the list and render an HTML table, which you see in the student list screenshot. It can also read sessionScope.role to show or hide the “Add”, “Edit”, and “Delete” buttons. Even if those buttons are hidden in the UI, the real protection still comes from AdminFilter and the controller logic. If a USER somehow triggers an admin URL, the filters and server-side checks will stop the operation.