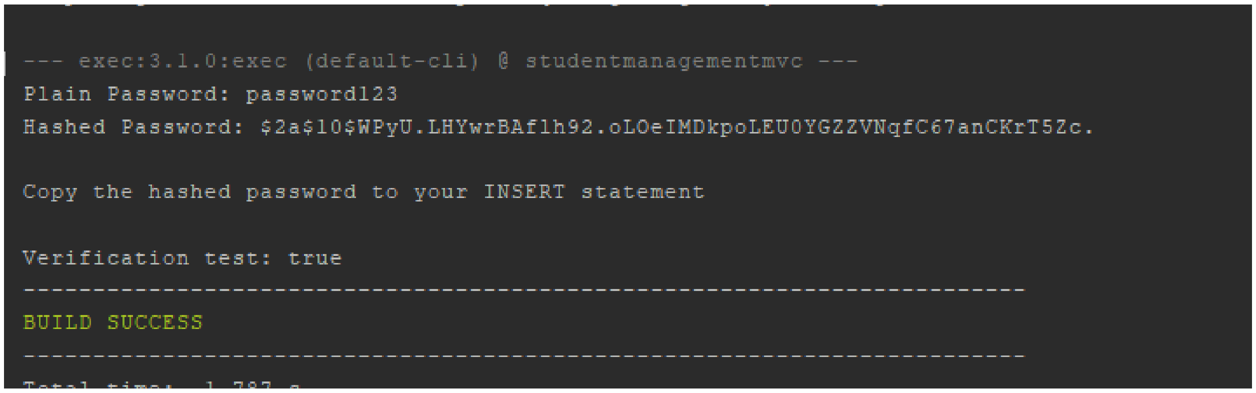
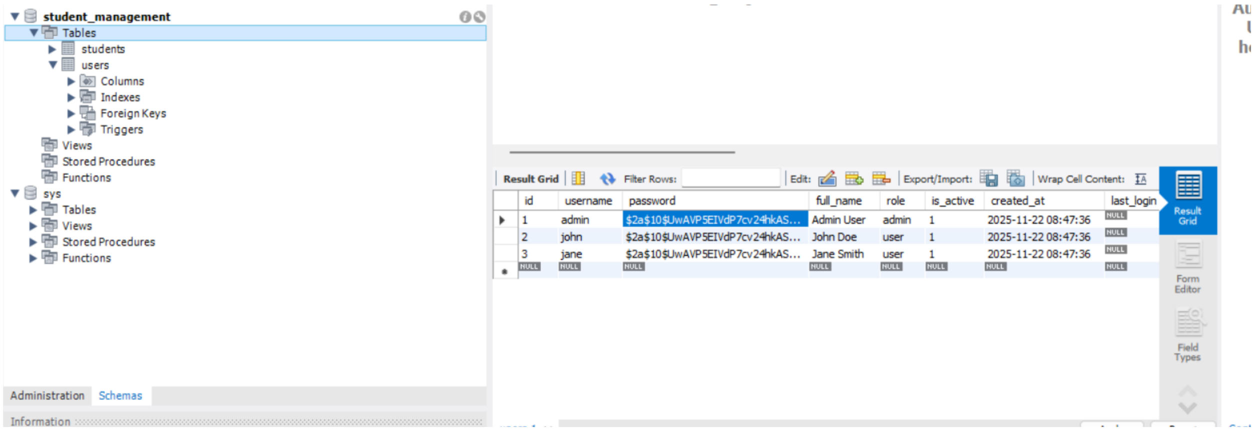
Exercise 1:

Hashed Code:



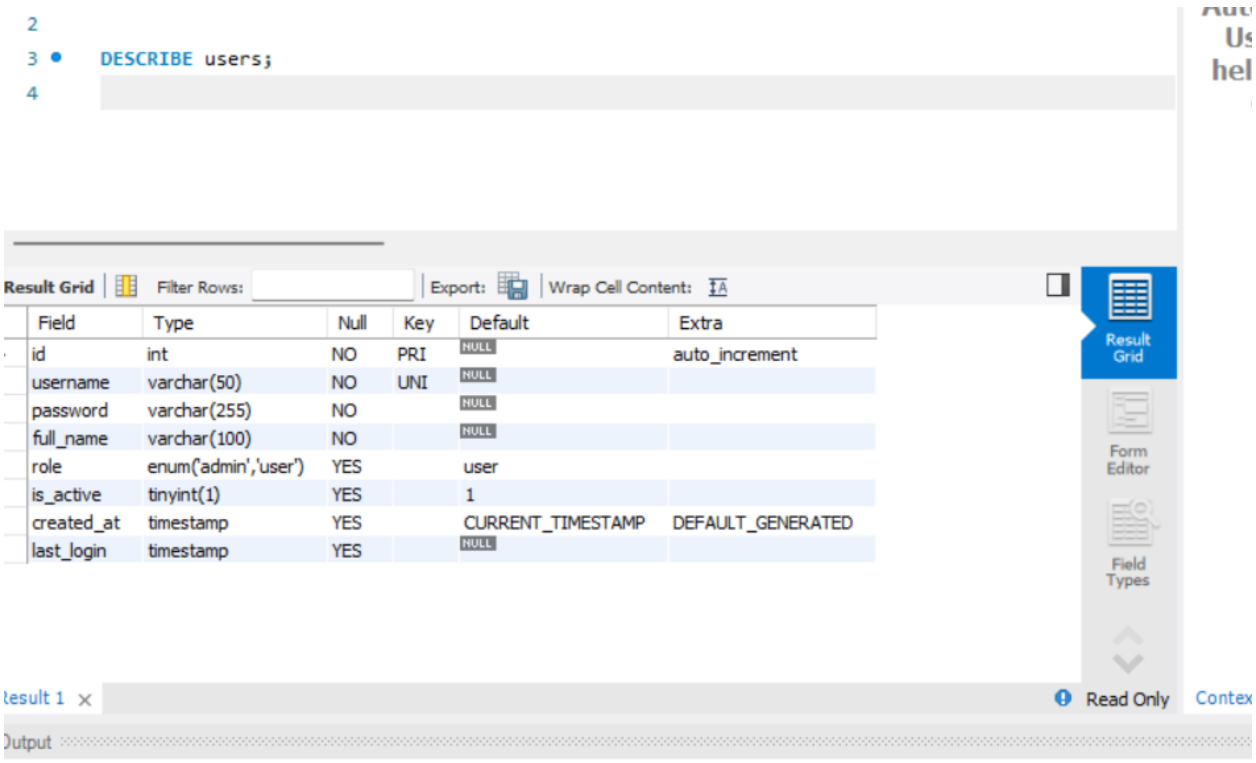
```
--- exec:3.1.0:exec (default-cli) @ studentmanagementmvc ---
Plain Password: password123
Hashed Password: $2a$10$WPyU.LHYwrBAflh92.oLOeIMDkpoLEU0YGZZVNqfC67anCKrT5Zc.

Copy the hashed password to your INSERT statement

Verification test: true
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

Generated Table:



id	username	password	full_name	role	is_active	created_at	last_login
1	admin	$2a$10$UwAVP5EIVdP7cv24hkAS...	Admin User	admin	1	2025-11-22 08:47:36	NULL
2	john	$2a$10$UwAVP5EIVdP7cv24hkAS...	John Doe	user	1	2025-11-22 08:47:36	NULL
3	jane	$2a$10$UwAVP5EIVdP7cv24hkAS...	Jane Smith	user	1	2025-11-22 08:47:36	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Verification Query:



```sql
DESCRIBE users;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
username	varchar(50)	NO	UNI	NULL	
password	varchar(255)	NO		NULL	
full_name	varchar(100)	NO		NULL	
role	enum('admin','user')	YES		user	
is_active	tinyint(1)	YES		1	
created_at	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
last_login	timestamp	YES		NULL	

Code Explaination:

```java
String plainPassword = "password123";

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

    System.out.println("Plain Password: " + plainPassword);
    System.out.println("Hashed Password: " + hashedPassword);
```

Uses Bcrypt.hashpw to generate a hash of password123, then print them out

Exercise 2:

Usermodel:

```java
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;

  }


  // 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 +
                '}';
    }
}
```

Defines user and its attributes, set getters, setters and other utilities

UserDAO:

```
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 (?, ?, ?, ?)";
```

Defines the queries for SQL

```
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;
    }
```

Verifies the User password by checking the hashed password

Then return the authenticated user if the password is valid

```
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();
    }
}
```

Store the last login of the user by setting it in the userID

```
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;
    }
```

Searches the User's name and ID based on the either the username or ID. Then returns the username and ID

```
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;
        }
```

```
    }
```

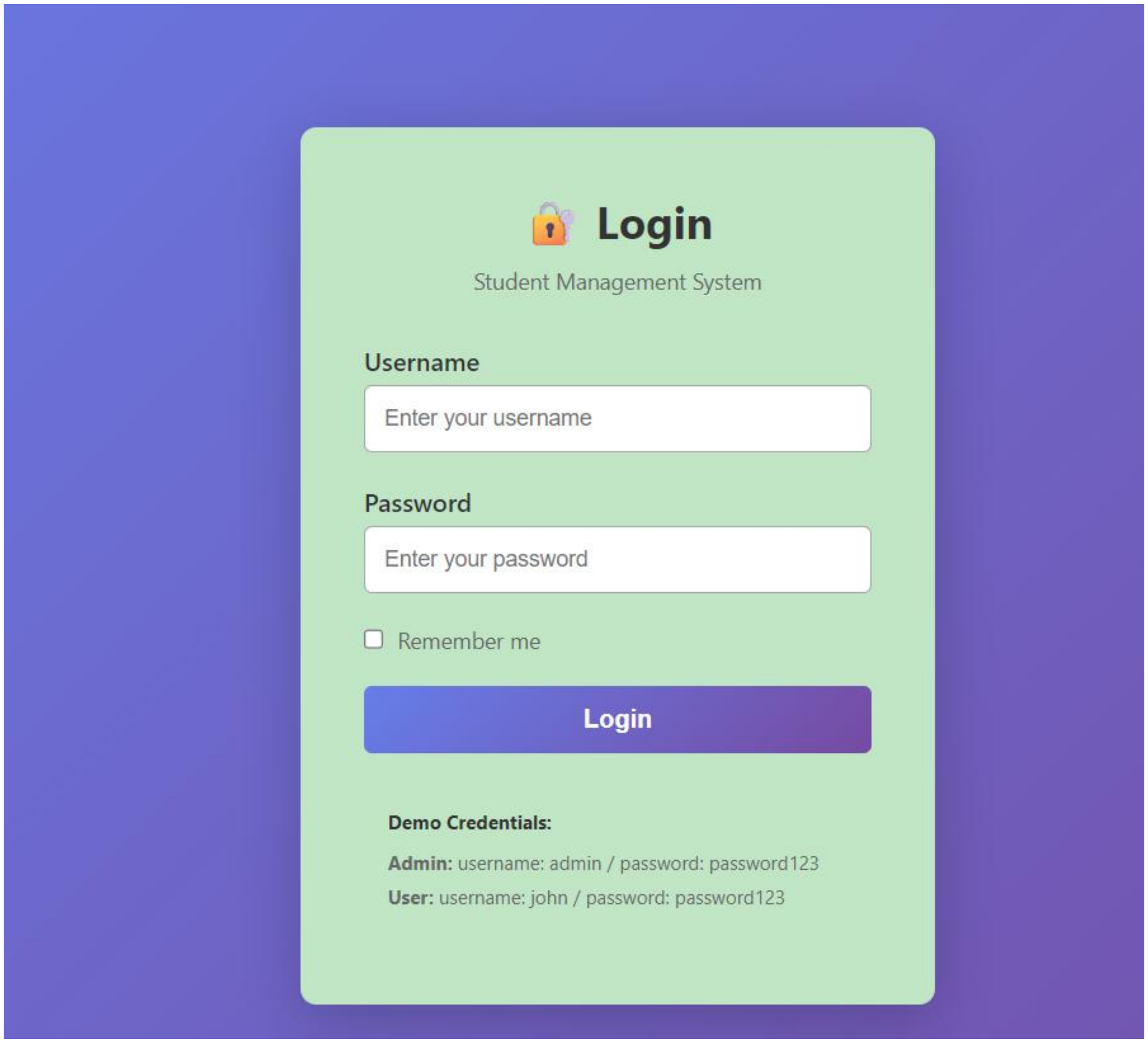
Create a brand new user based on inputs, setting each value in a prepared statement based on input. Then enter the new values into the database by pstmt.executeUpdate(). Return if more than 0 rows were affected to see if insertion was successful. Try connections to prevent resource leaks

```
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;
    }
```

The method takes **one record** from a SQL query result.Reads each column using rs.getX(...).Fills the User object's properties.Returns the completed User.

Exercise 3:

Login Page:



Demo Credentials:

Admin: username: admin / password: password123

User: username: john / password: password123

Code explanation:

```
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);
    }
```

This doGet handles requests to the login page. It performs session checking to avoid letting logged-in users see the login page again.

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

Get the username, password and whether the page remembers the two

```
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;
      }
```

Checks if the username and password are empty, returns an error if either one is empty.

```
HttpSession oldSession = request.getSession(false);
if (oldSession != null) {
   oldSession.invalidate();
}
```

If an attacker somehow obtains a session ID before the user logs in, invalidating the old session ensures they cannot hijack a valid authenticated session.

Create a fresh session and store user details

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

Now the user is “logged in,” because future requests can retrieve this session and check authentication.

```
if (user.isAdmin()) {
   response.sendRedirect("dashboard");
} else {
   response.sendRedirect("student?action=list");
}
```

Admin users → dashboard

Student/normal users → student list page

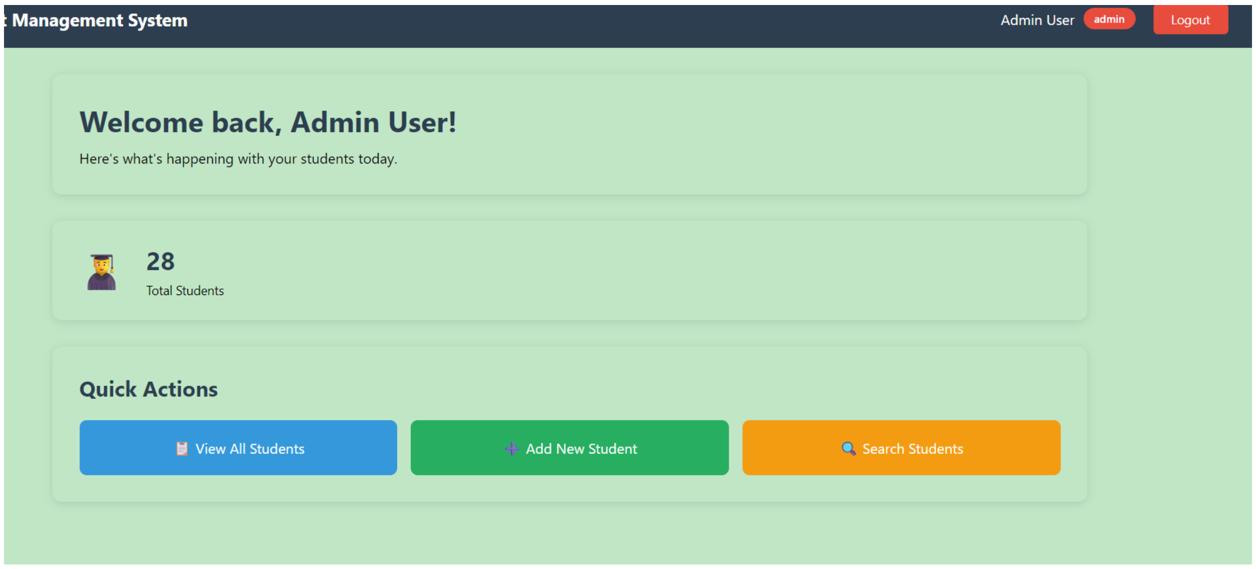
```
} else {
   request.setAttribute("error", "Invalid username or password");
   request.setAttribute("username", username);
   request.getRequestDispatcher("/views/login.jsp").forward(request, response);
}
```

Set an error message

Keep the user's input so the username field isn't blank after failure

Forward back to login JSP (no redirect, because you want to show the error)

Succesful Login:


Management System

Admin User admin Logout

Welcome back, Admin User!

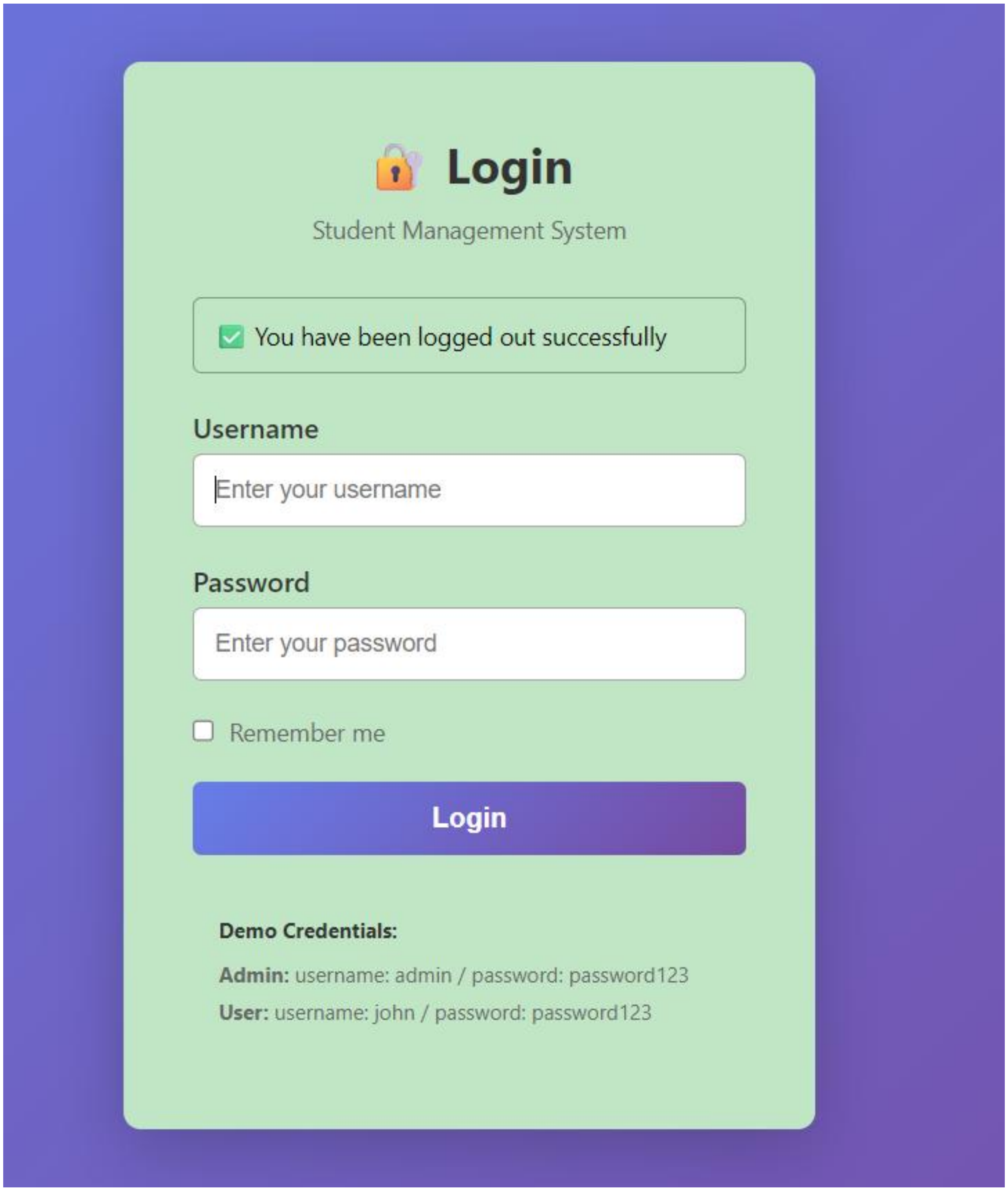
Here's what's happening with your students today.

28

Total Students

Quick Actions

Logout:



🔐 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

Code Explaination:

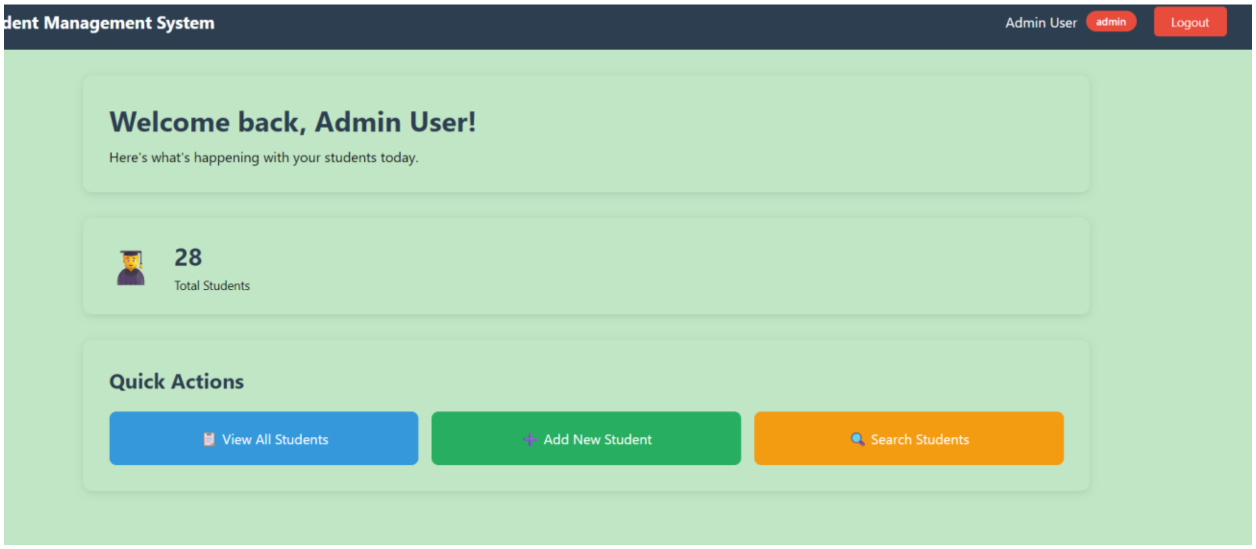
```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
```

```
        // 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);
    }
```

Gets the current session and then validate it. Invalid if the session is null. Then send to the login page

Exercise 4:

Dashboard:



Welcome back, Admin User!

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

28

Total Students

Quick Actions

Code Explaination:

```
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);
  }
}
```

DashboardController extends on HttpServlet. Then it gets the session and validates whether it is empty or not, in which case it redirects the user to the login page. Then it get the total number of students and set it in the request as well as a welcome back message including the user’s full name.