7.1P – Securing Your Enterprise Application

Task 4 – Analysis Task

4.1 In the context of the employee's CRUD operations on their own record, why the system does not allow employee to perform the "C" and "D" operations? Justify your answers.

The system doesn't allow an employee to perform a create or delete option because it would allow them to change details that would otherwise be immutable by simply deleting their record and recreating

4.2 In the context of the employee's Review operation, what information can be reviewed? Justify your answer.

Empid, name, phone, address, email, bankaccountid, & salary.
These fields should be reviewable as they all directly relate to the employee.

Password is not shown as it could be seen by other employees in the same physical location.

Appgroup & active are not shown as they aren't relevant to the employee and partially reveal how the system is secured.

4.3 In the context of employee's Review operation (reviewing their own detail), the company decided to implement a DTO which excludes the password being sent to the client. Why password is excluded? Do you think that this is a good practice? Why or Why not? If not, propose an alternative and justify your choice.

The password is excluded because its unnecessary to display to the user, and risks someone in the same physical location seeing it.

I think this is a good practice because it removes a possible security risk.

4.4 In the context of the employee's Update operation,

a. **What information can be updated? Are these the same as those in 4.2 and 4.3 above? Why or Why not? Justify your answer.**

   Name, Phone, Address, email, password, bankaccountid.

   These can be changed because they directly relate to either the employee personally, or to their use of the application.

The password can be changed but not viewed, as its expected the employee memorises their password.

b. **What information cannot be updated? How would you avoid these data being updated by the employee "accidentally"?**

   Empid, appgroup, salary, active status.

   To avoid updates there would be a separate screen with only the fields we want the employee to be able to update.

4.5 In the context of the employee's Update operation, where should the actual change of the employee's information occur? Do you think this is a good practice? Why or Why not? Justify your answer.

On a separate employee only update page. I think this is good practice as it allows full control over what fields can be updated, and removes the possibility of bugs where the user can update fields only an admin should be able to that may occur using a shared ui.

4.6 In the context of employee's Update operation, the company decided to first display the details of a particular employee (if such employee exists after searching through the database via the employee's id) in the web browser so that the employee could enter the required information. Should the existing password be

a. **sent and displayed to the client?**

b. **sent to the client but not displayed?**

c. **not sent?**

What is your choice? Why or why not? Justify your answer. If your answer is (3), how would you implement the feature that allows the employee to change their own password?

C. There's no reason for the client to have a copy of the users current password. The managed bean on the server would contain it, and password updates would occur by having the user submit their old and new password and the managed bean would only allow it if the submitted old password matches the actual old password.

4.7 In the context of deleting employee's record, the company choose to accept the employee id as the input and then remove the employee record by setting the field "active" to false instead of removing the record from the database. Do you think that this is a good practice? Why or Why not? If not, propose an alternative and justify your choice.

This is more of a business choice as keeping a list of inactive users allows for collecting historical data about the business. This seems like an ok practice as I can't think of any way this could be abused.

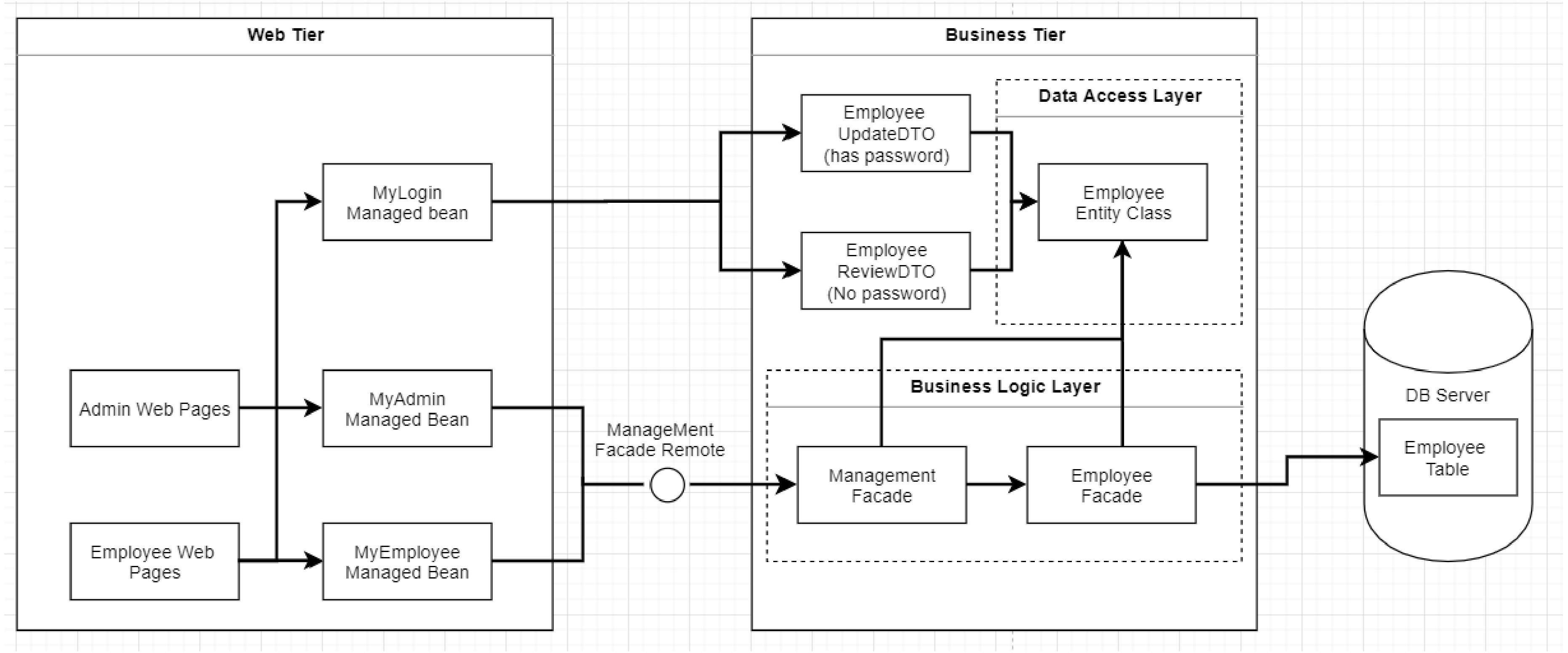
4.8 After reviewing the features provided by the application "ED-Secure", do you think that the application as is can provide the features listed in the Case Study Section above? Are there any deficiencies? Why and why not? What changes would you suggest to address all features listed?

The application needs the following changes:

- Separate user only jsf pages for employees to view and update their data
- New DTO class that omits password

Task 5

a. **Uml diagram**



b. Component Descriptions

- Admin Web Pages

The set of jsf webpages that provide the UI for the admin tasks to be completed in the application

- Employee web pages

JSF web pages in which the employee can navigate to review and update their details.

- Employee UpdateDTO

DTO specific for transferring updated details to the managed bean. This contains the password fields for updating user password

- Employee Review DTO

DTO for transferring employee review values from managed beans to the client. Does not contain a password field

- MyAdmin Managed Bean

Holds session data & provides functionality specific to admin pages.

- MyEmployee Managed Bean

Holds session data & provides functionality specific to employee pages. Notably different to the admin managed bean as it will lack methods for adding & deleting users, and will only retrieve data for the current user.

- Mylogin Managed Bean

Provides functionality for logging the user out of their login session.

- Management Façade

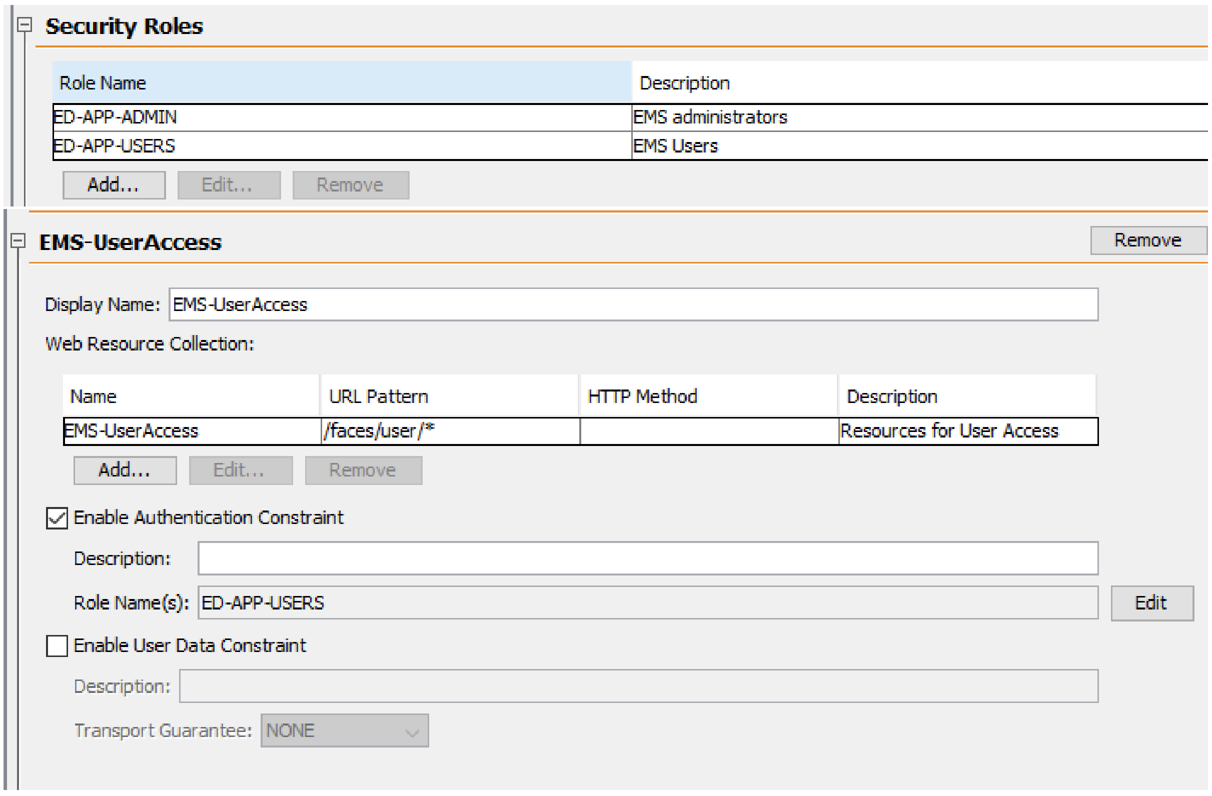
Provides data access methods to managed beans.

- Employee Façade

Provides data access methods for updating employees in the database.

Task 4. Programming

The first thing I did to secure the application was to secure a separate file set for the user group.



Security Roles

Role Name	Description
ED-APP-ADMIN	EMS administrators
ED-APP-USERS	EMS Users

EMS-UserAccess

Display Name: EMS-UserAccess

Web Resource Collection:

Name	URL Pattern	HTTP Method	Description
EMS-UserAccess	/faces/user/*		Resources for User Access

[✓] Enable Authentication Constraint

Description:

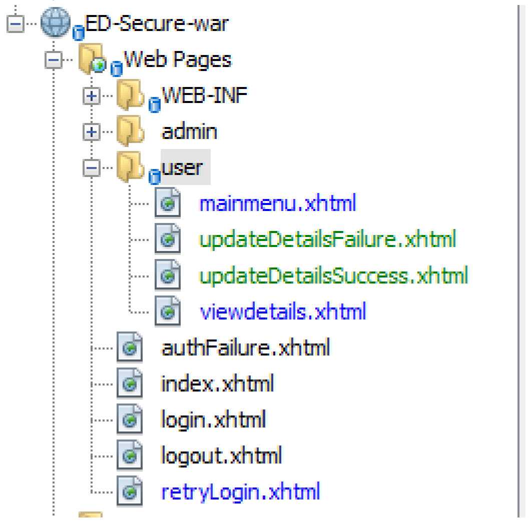
Role Name(s): ED-APP-USERS

[ ] Enable User Data Constraint

Description:

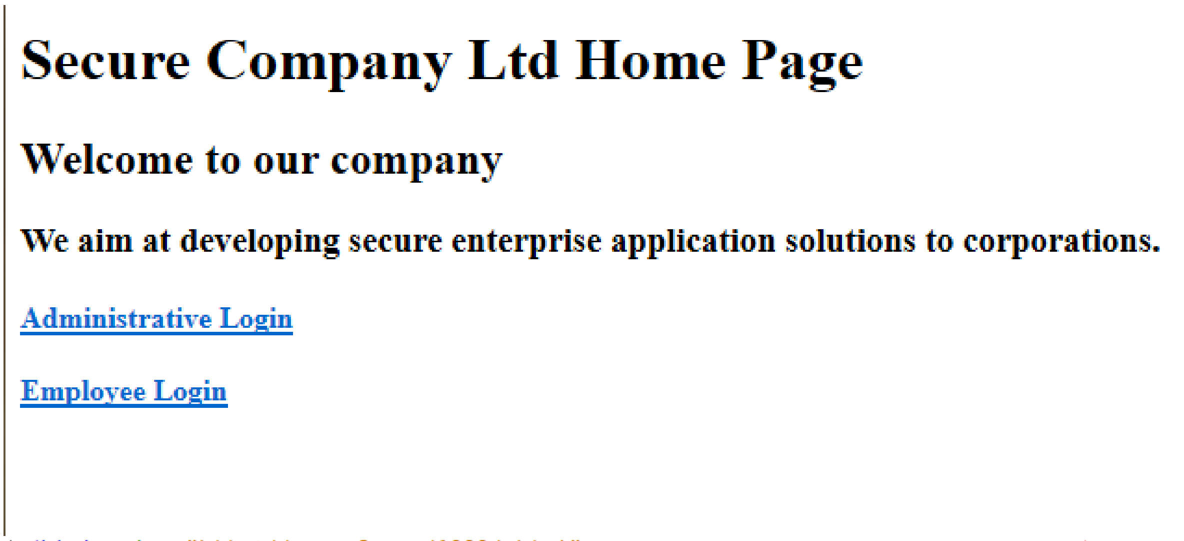
Transport Guarantee: NONE

Only users can access the set of files in the /faces/users/ folders.



```
ED-Secure-war
  Web Pages
    WEB-INF
    admin
    user
      mainmenu.xhtml
      updateDetailsFailure.xhtml
      updateDetailsSuccess.xhtml
      viewdetails.xhtml
    authFailure.xhtml
    index.xhtml
    login.xhtml
    logout.xhtml
    retryLogin.xhtml
```

The application uses a non-restricted index file so the user can choose between admin and employee login



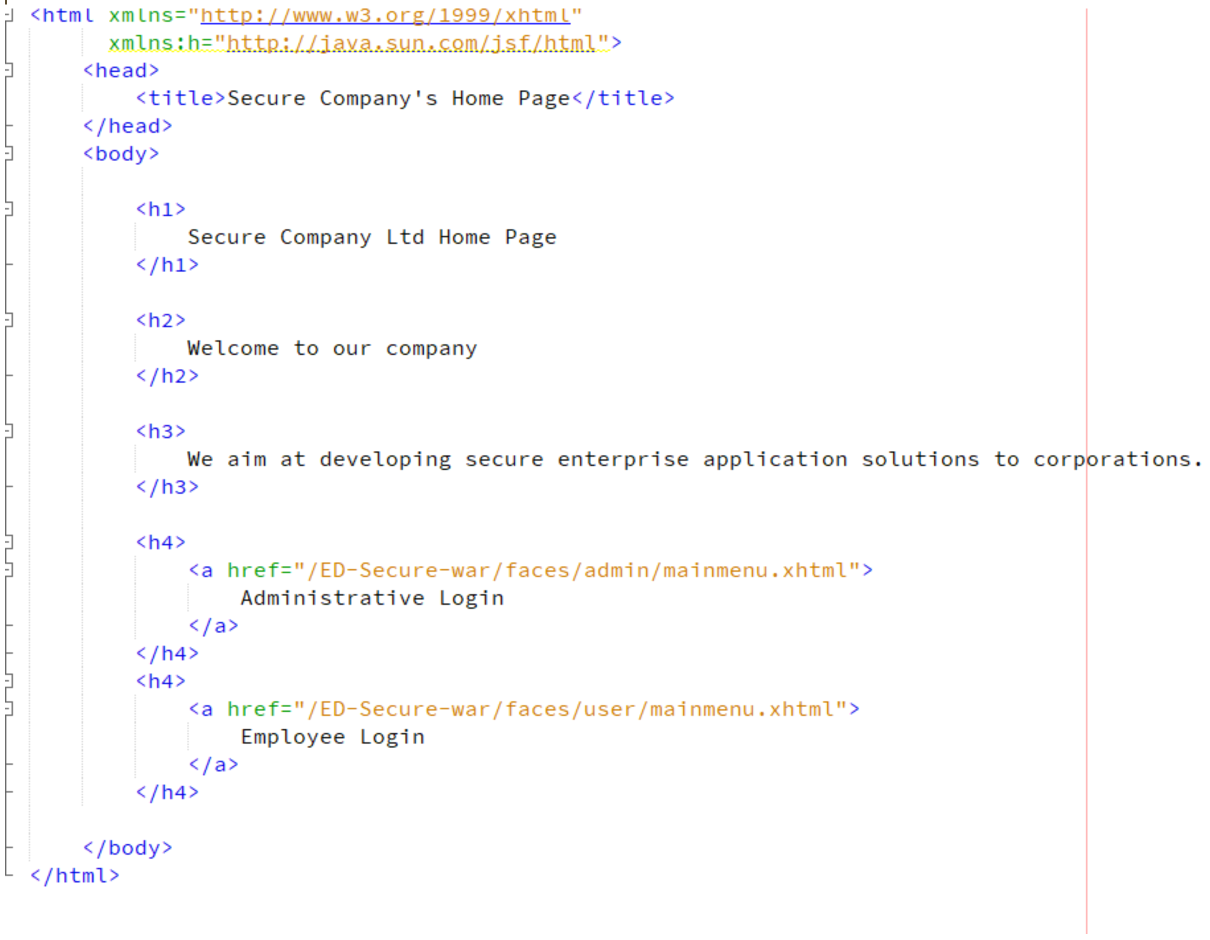
Secure Company Ltd Home Page

Welcome to our company

We aim at developing secure enterprise application solutions to corporations.

Administrative Login

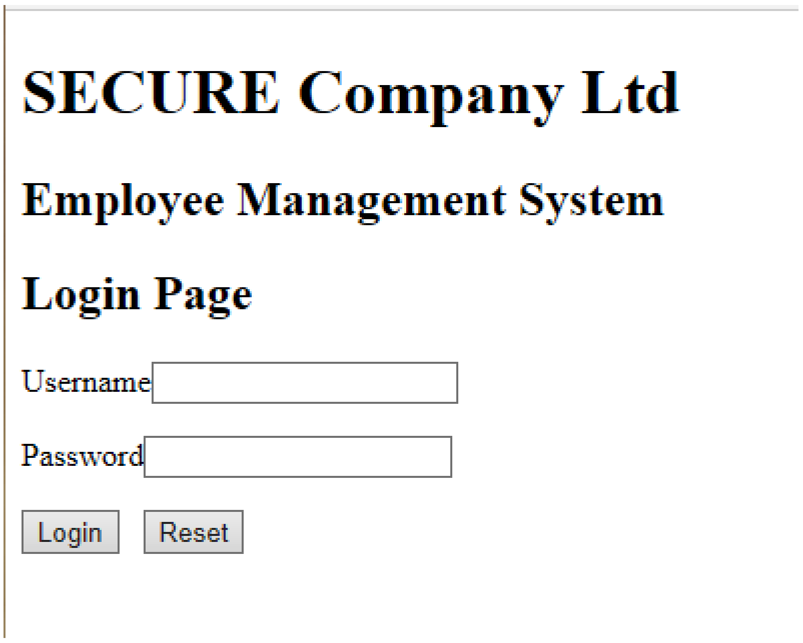
Employee Login



```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <title>Secure Company's Home Page</title>
    </head>
    <body>

        <h1>
            Secure Company Ltd Home Page
        </h1>

        <h2>
            Welcome to our company
        </h2>

        <h3>
            We aim at developing secure enterprise application solutions to corporations.
        </h3>

        <h4>
            <a href="/ED-Secure-war/faces/admin/mainmenu.xhtml">
                Administrative Login
            </a>
        </h4>
        <h4>
            <a href="/ED-Secure-war/faces/user/mainmenu.xhtml">
                Employee Login
            </a>
        </h4>

    </body>
</html>
```

The links direct to restricted resources so the user is forced to login, which essentially gets us the two login screen functionality

When you select employee login the user is redirected to the java ee login screen


SECURE Company Ltd

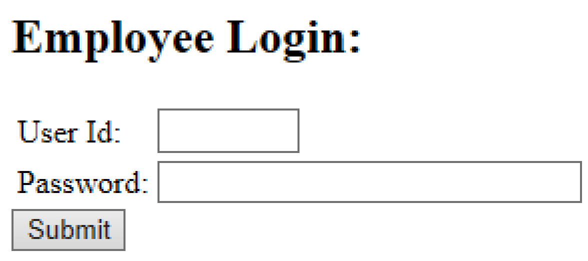
Employee Management System

Login Page

Username

Password

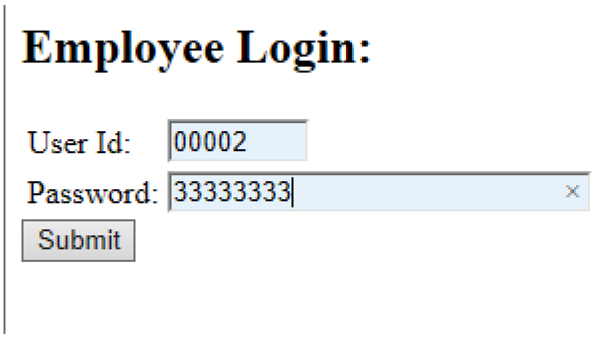
Because this tutorial doesn't set up the app to work with the database for java ee authentication the java authentication simply leads to another login page:


Employee Login:

User Id:

Password:

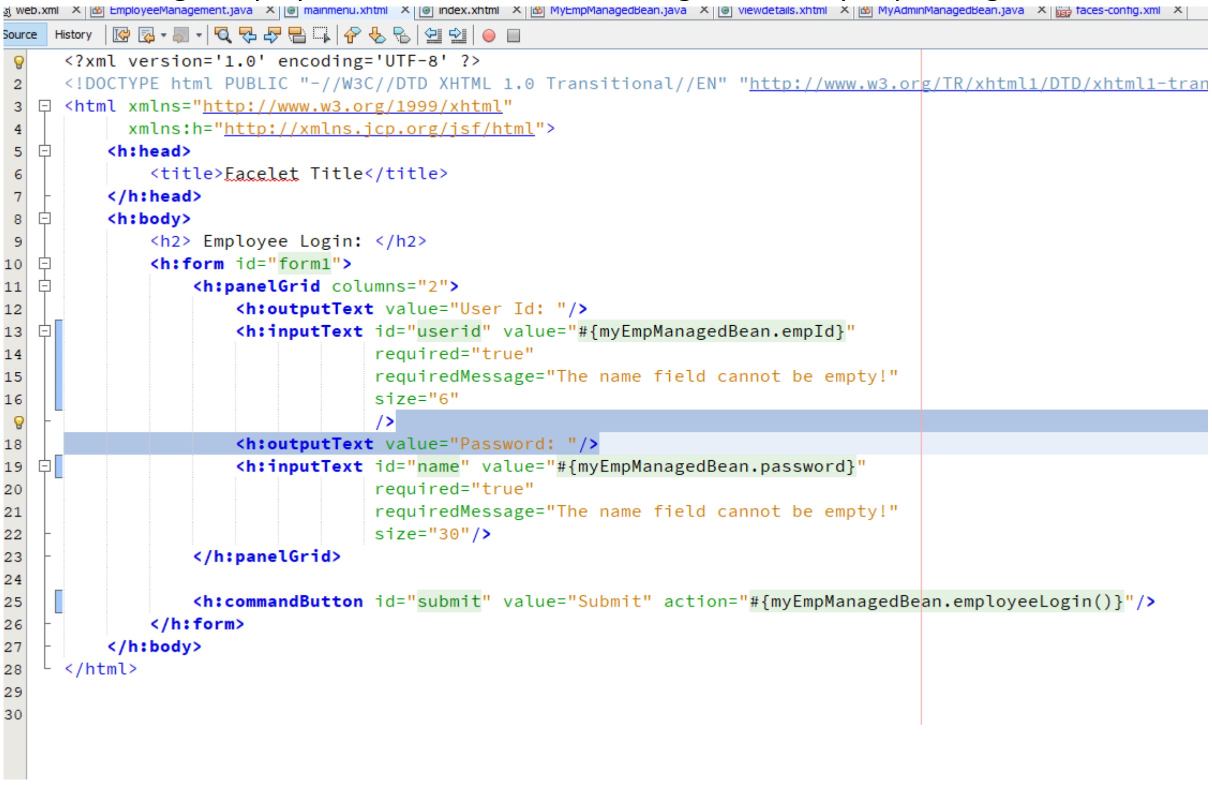
Here we put the employee no and their password to login:


Employee Login:

User Id: 00002

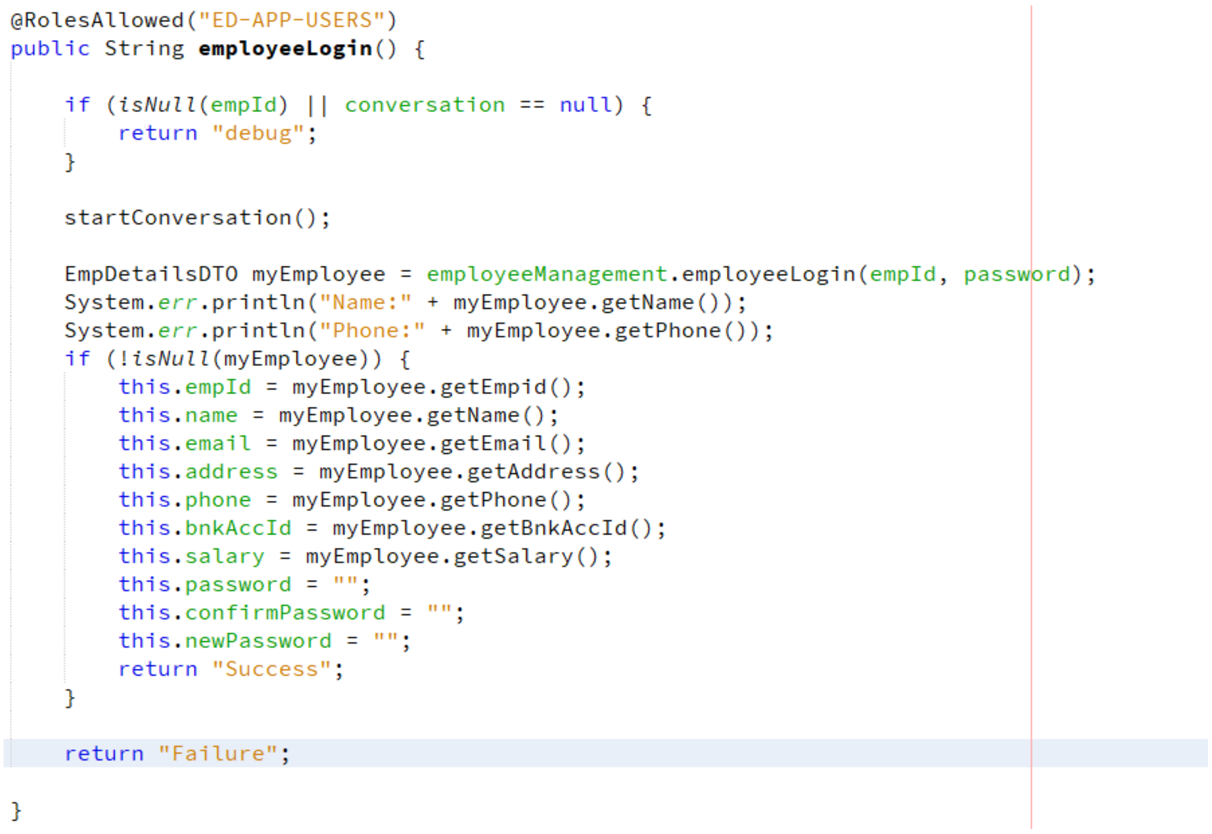
Password: 33333333

This calls a loginEmployee method in our new managed bean: myEmpManagedBean



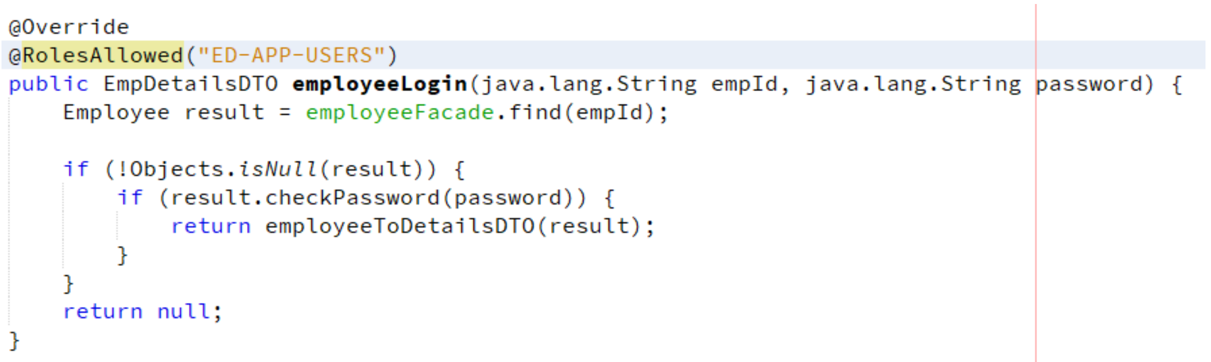
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-tran
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h2> Employee Login: </h2>
        <h:form id="form1">
            <h:panelGrid columns="2">
                <h:outputText value="User Id: "/>
                <h:inputText id="userid" value="#{myEmpManagedBean.empId}"
                             required="true"
                             requiredMessage="The name field cannot be empty!"
                             size="6"
                             />
                <h:outputText value="Password: "/>
                <h:inputText id="name" value="#{myEmpManagedBean.password}"
                             required="true"
                             requiredMessage="The name field cannot be empty!"
                             size="30"/>
            </h:panelGrid>

            <h:commandButton id="submit" value="Submit" action="#{myEmpManagedBean.employeeLogin()}"/>
        </h:form>
    </h:body>
</html>
```

This method asks the Employee management database for an employee record that matches BOTH the employee id and password:



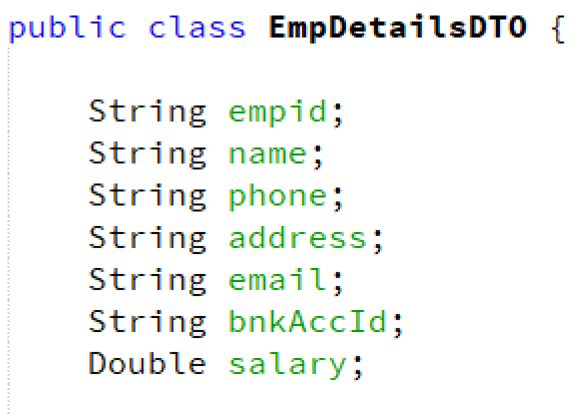
```
@RolesAllowed("ED-APP-USERS")
public String employeeLogin() {

    if (isNull(empId) || conversation == null) {
        return "debug";
    }

    startConversation();

    EmpDetailsDTO myEmployee = employeeManagement.employeeLogin(empId, password);
    System.err.println("Name:" + myEmployee.getName());
    System.err.println("Phone:" + myEmployee.getPhone());
    if (!isNull(myEmployee)) {
        this.empId = myEmployee.getEmpid();
        this.name = myEmployee.getName();
        this.email = myEmployee.getEmail();
        this.address = myEmployee.getAddress();
        this.phone = myEmployee.getPhone();
        this.bnkAccId = myEmployee.getBnkAccId();
        this.salary = myEmployee.getSalary();
        this.password = "";
        this.confirmPassword = "";
        this.newPassword = "";
        return "Success";
    }

    return "Failure";

}
```

This is the method in the management façade.



```java
@Override
@RolesAllowed("ED-APP-USERS")
public EmpDetailsDTO employeeLogin(java.lang.String empId, java.lang.String password) {
    Employee result = employeeFacade.find(empId);

    if (!Objects.isNull(result)) {
        if (result.checkPassword(password)) {
            return employeeToDetailsDTO(result);
        }
    }
    return null;
}
```

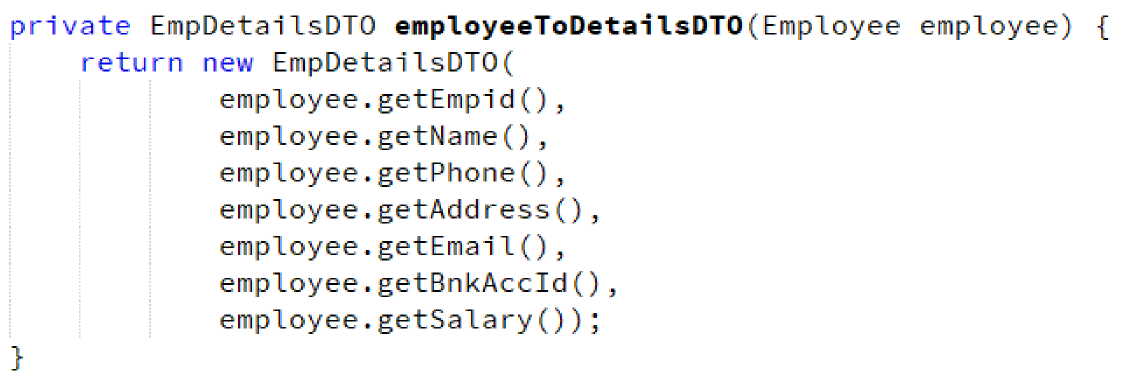
It checks that the employee exists, then it uses the employee checkpassword method to make sure the password is correct, then it converts the full employee object to one of the new DTO's: employeeDetailsDTO:



```java
public class EmpDetailsDTO {

    String empid;
    String name;
    String phone;
    String address;
    String email;
    String bnkAccId;
    Double salary;
```

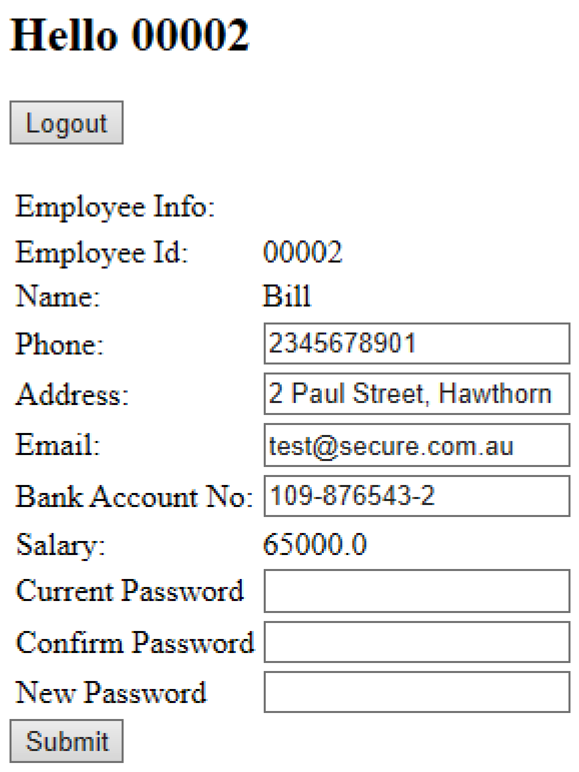
This DTO omits the password, user group, and active fields so that no secure data is leaked to the frontend.

The following method does the conversion



```java
private EmpDetailsDTO employeeToDetailsDTO(Employee employee) {
    return new EmpDetailsDTO(
            employee.getEmpid(),
            employee.getName(),
            employee.getPhone(),
            employee.getAddress(),
            employee.getEmail(),
            employee.getBnkAccId(),
            employee.getSalary());
}
```

Once this is returned the values are assigned to the bean and we get this page:



Hello 00002

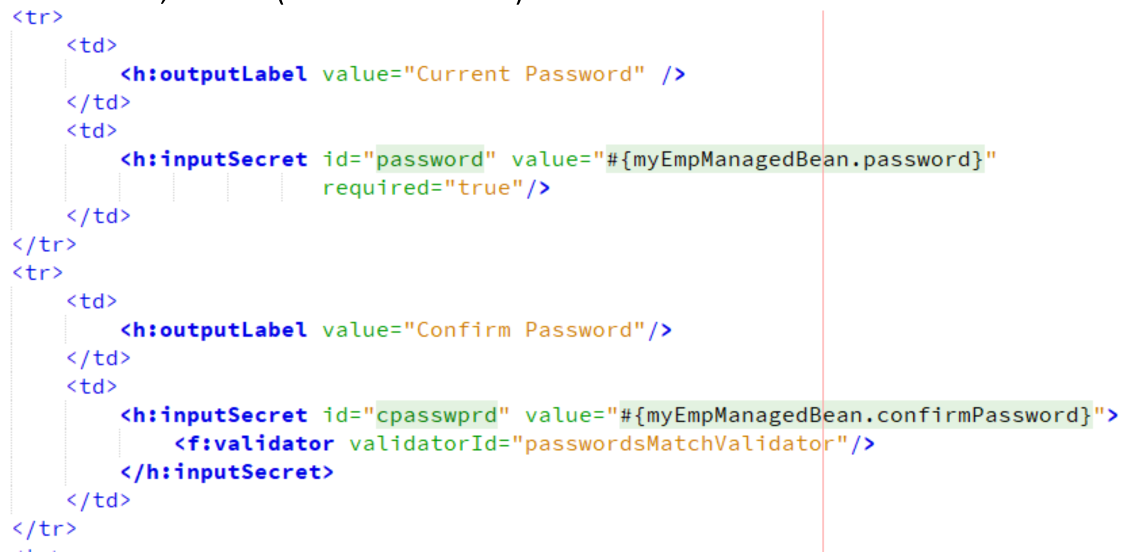
Logout

Employee Info:	
Employee Id:	00002
Name:	Bill
Phone:	2345678901
Address:	2 Paul Street, Hawthorn
Email:	test@secure.com.au
Bank Account No:	109-876543-2
Salary:	65000.0
Current Password	
Confirm Password	
New Password	

Submit

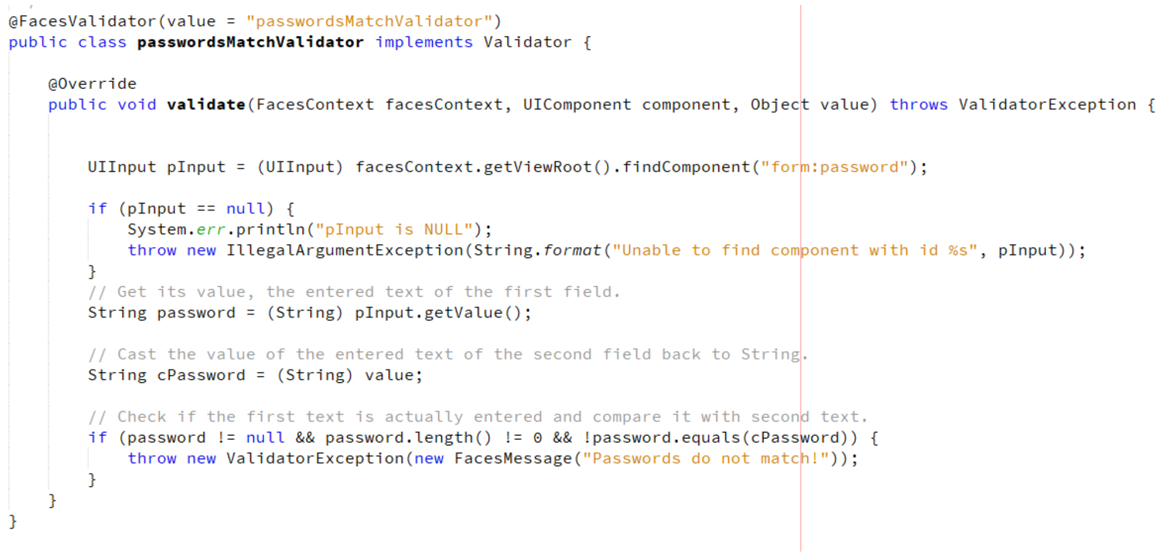
Because I’m lazy all detail changes are done here.
To submit the data the user is required to enter their current password and then re-enter it in the confirm password field.

This is done via the required=”true” properties, and by reusing the password validator I made for 5.1, or 5.2C (I don’t remember)



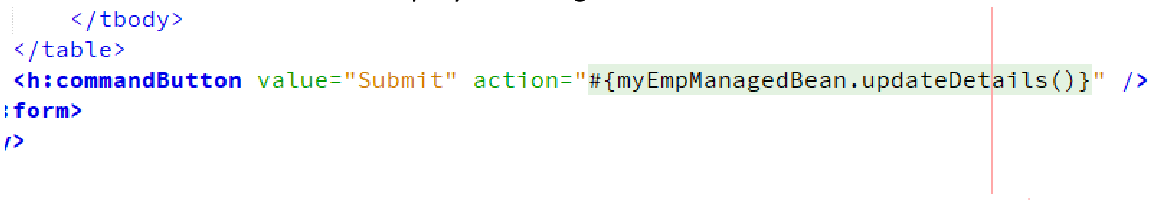
```
<tr>
    <td>
        <h:outputLabel value="Current Password" />
    </td>
    <td>
        <h:inputSecret id="passwprd" value="#{myEmpManagedBean.password}"
                       required="true"/>
    </td>
</tr>
<tr>
    <td>
        <h:outputLabel value="Confirm Password"/>
    </td>
    <td>
        <h:inputSecret id="cpasswprd" value="#{myEmpManagedBean.confirmPassword}">
            <f:validator validatorId="passwordsMatchValidator"/>
        </h:inputSecret>
    </td>
</tr>
```

The validator:

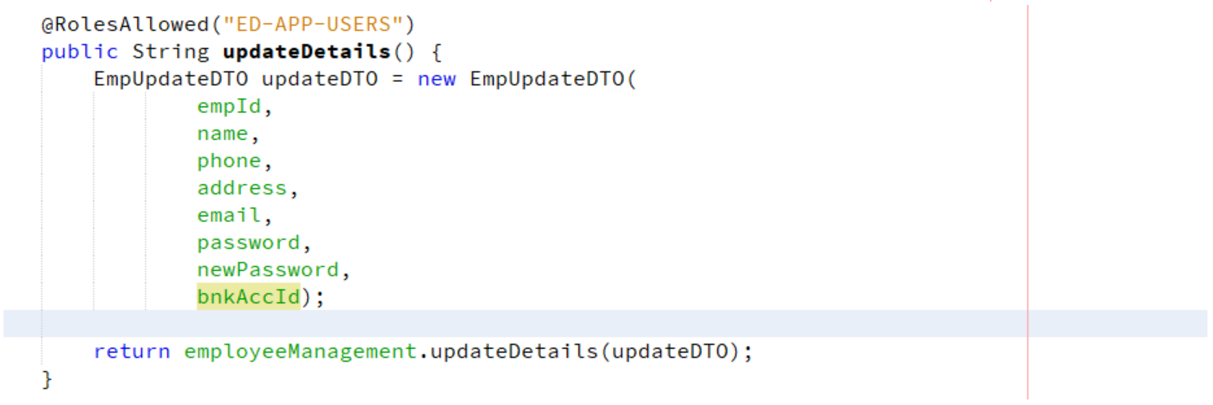


```java
@FacesValidator(value = "passwordsMatchValidator")
public class passwordsMatchValidator implements Validator {

    @Override
    public void validate(FacesContext facesContext, UIComponent component, Object value) throws ValidatorException {

        UIInput pInput = (UIInput) facesContext.getViewRoot().findComponent("form:password");

        if (pInput == null) {
            System.err.println("pInput is NULL");
            throw new IllegalArgumentException(String.format("Unable to find component with id %s", pInput));
        }
        // Get its value, the entered text of the first field.
        String password = (String) pInput.getValue();

        // Cast the value of the entered text of the second field back to String.
        String cPassword = (String) value;

        // Check if the first text is actually entered and compare it with second text.
        if (password != null && password.length() != 0 && !password.equals(cPassword)) {
            throw new ValidatorException(new FacesMessage("Passwords do not match!"));
        }
    }
}
```

The way it works is the user clicks submit, and then the employee managed bean packages the data and sends it to the EmployeeManagementFacade for validation:

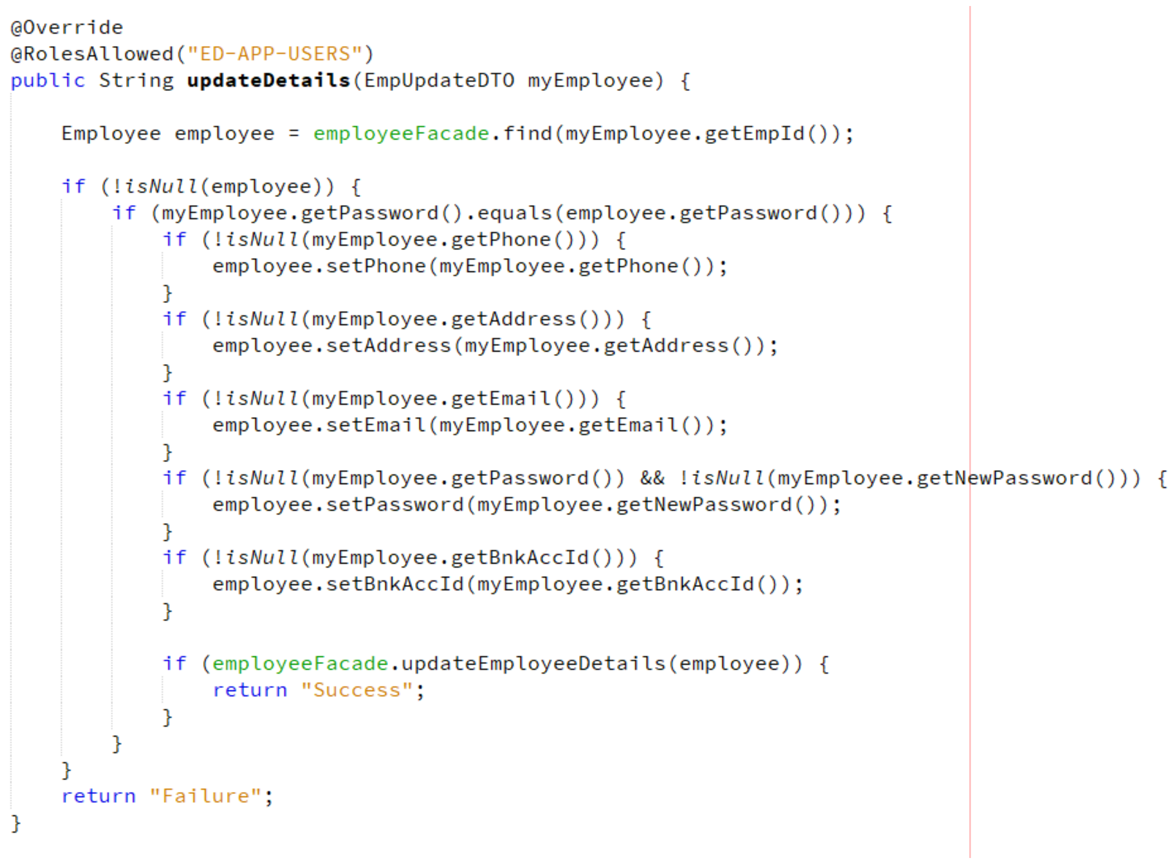


```xml
    </tbody>
</table>
<h:commandButton value="Submit" action="#{myEmpManagedBean.updateDetails()}" />
form>
/>
```



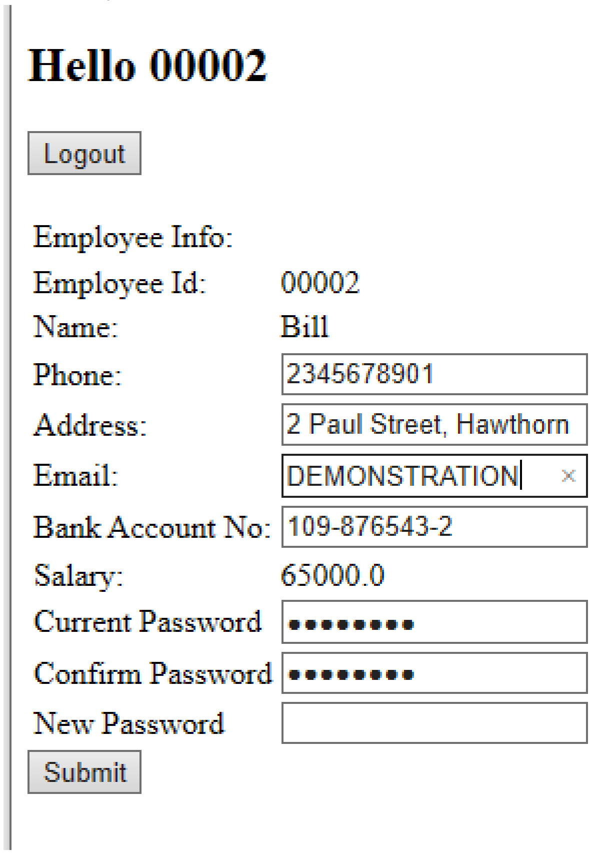
```java
@RolesAllowed("ED-APP-USERS")
public String updateDetails() {
    EmpUpdateDTO updateDTO = new EmpUpdateDTO(
            empId,
            name,
            phone,
            address,
            email,
            password,
            newPassword,
            bnkAccId);

    return employeeManagement.updateDetails(updateDTO);
}
```

So, how can I use one method to update ALL the data at once? What if fields are null? Well, this first checks that the password is valid (no need to confirm cPassword is, as it necessarily matches the password) and then only updates fields that are NOT null.



```java
@Override
@RolesAllowed("ED-APP-USERS")
public String updateDetails(EmpUpdateDTO myEmployee) {

    Employee employee = employeeFacade.find(myEmployee.getEmpId());

    if (!isNull(employee)) {
        if (myEmployee.getPassword().equals(employee.getPassword())) {
            if (!isNull(myEmployee.getPhone())) {
                employee.setPhone(myEmployee.getPhone());
            }
            if (!isNull(myEmployee.getAddress())) {
                employee.setAddress(myEmployee.getAddress());
            }
            if (!isNull(myEmployee.getEmail())) {
                employee.setEmail(myEmployee.getEmail());
            }
            if (!isNull(myEmployee.getPassword()) && !isNull(myEmployee.getNewPassword())) {
                employee.setPassword(myEmployee.getNewPassword());
            }
            if (!isNull(myEmployee.getBnkAccId())) {
                employee.setBnkAccId(myEmployee.getBnkAccId());
            }

            if (employeeFacade.updateEmployeeDetails(employee)) {
                return "Success";
            }
        }
    }
    return "Failure";
}
```

This means if you leave any fields blank it wont update.
So we update the values:



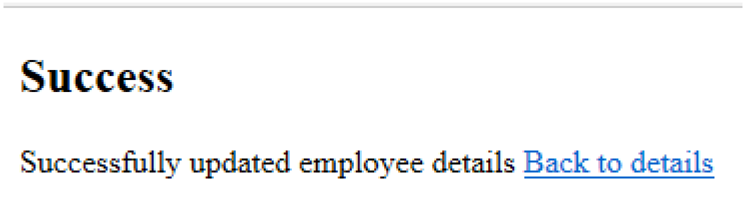
Hello 00002

Logout

Employee Info:	
Employee Id:	00002
Name:	Bill
Phone:	2345678901
Address:	2 Paul Street, Hawthorn
Email:	DEMONSTRATION
Bank Account No:	109-876543-2
Salary:	65000.0
Current Password	••••••••
Confirm Password	••••••••
New Password	

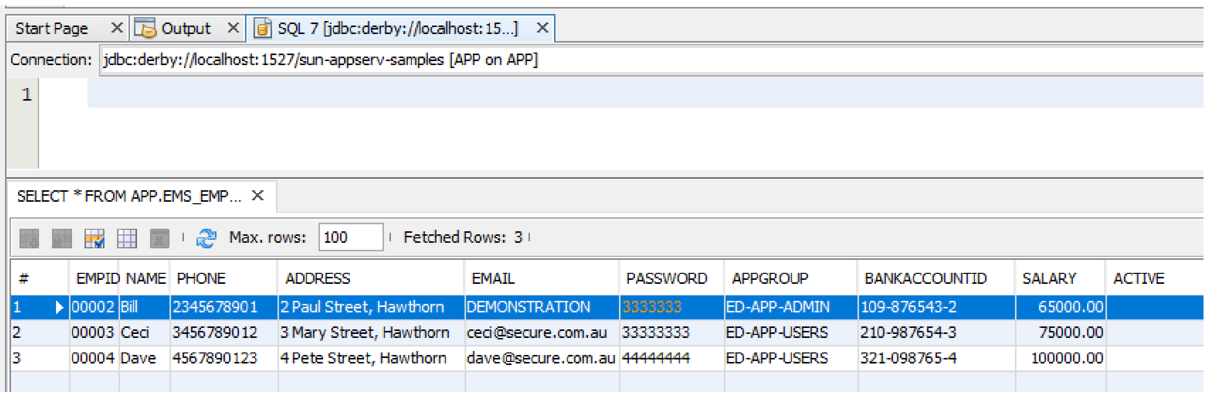
Submit

Then submit:



Success

Successfully updated employee details Back to details



Start Page | Output | SQL 7 [jdbc:derby://localhost:15...]

Connection: jdbc:derby://localhost:1527/sun-appserv-samples [APP on APP]

SELECT * FROM APP.EMS_EMP...

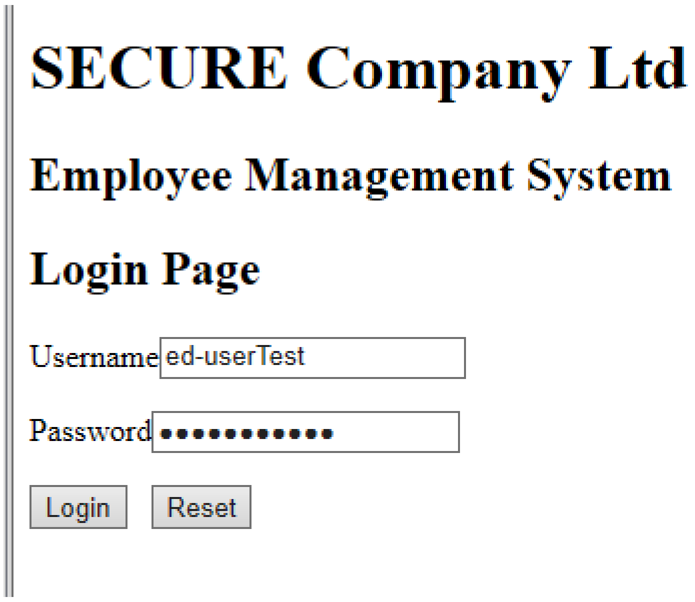
Max. rows: 100 Fetched Rows: 3

#	EMPID	NAME	PHONE	ADDRESS	EMAIL	PASSWORD	APPGROUP	BANKACCOUNTID	SALARY	ACTIVE
1	00002	Bill	2345678901	2 Paul Street, Hawthorn	DEMONSTRATION	3333333	ED-APP-ADMIN	109-876543-2	65000.00	
2	00003	Ceci	3456789012	3 Mary Street, Hawthorn	ceci@secure.com.au	33333333	ED-APP-USERS	210-987654-3	75000.00	
3	00004	Dave	4567890123	4 Pete Street, Hawthorn	dave@secure.com.au	44444444	ED-APP-USERS	321-098765-4	100000.00	

Here we see its updated

5. Testing:

Java EE secure login requires valid credentials



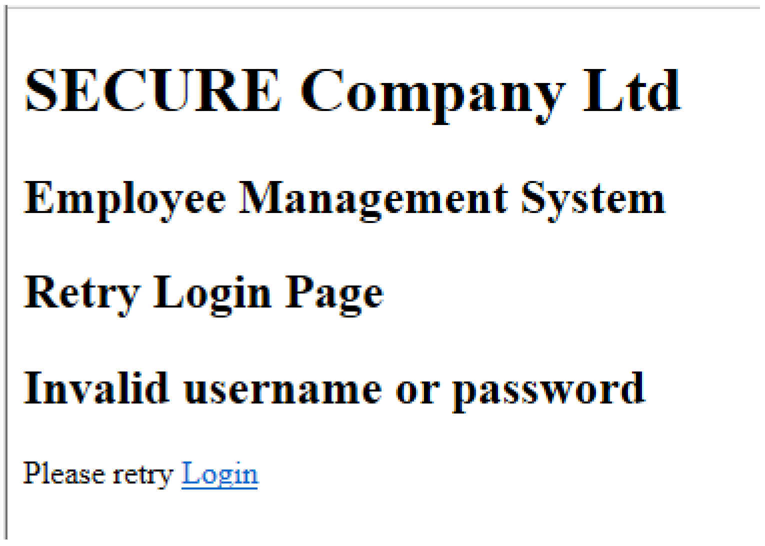
SECURE Company Ltd

Employee Management System

Login Page

Username ed-userTest

Password ●●●●●●●●●●●●



SECURE Company Ltd

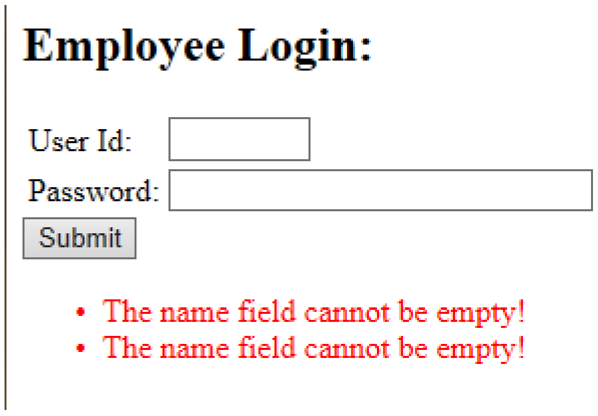
Employee Management System

Retry Login Page

Invalid username or password

Please retry Login

Employee login requires both fields:



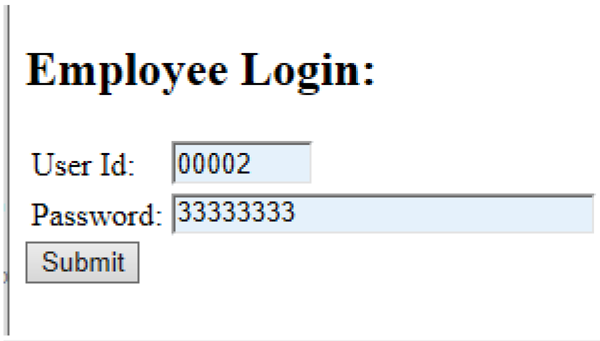
Employee Login:

User Id:

Password:

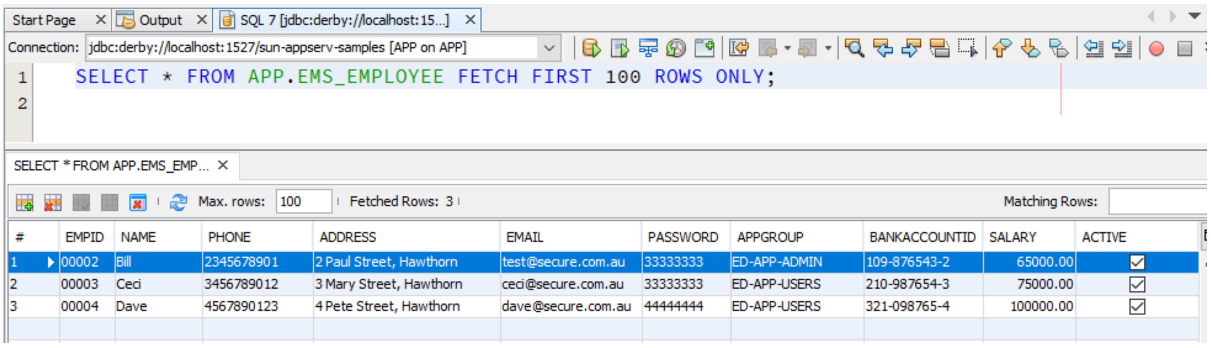
- The name field cannot be empty!
- The name field cannot be empty!

Employee login must match an emoloyee in db



Employee Login:

User Id:	00002
Password:	33333333



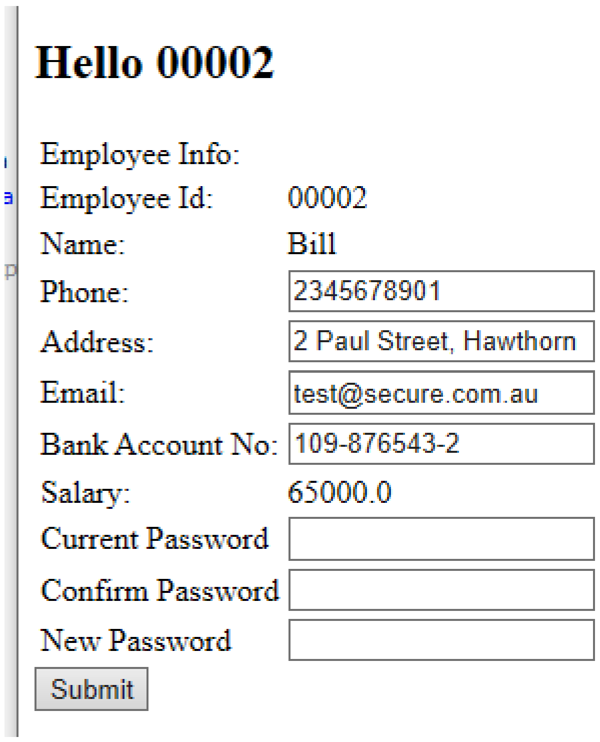
Connection: jdbc:derby://localhost:1527/sun-appserv-samples [APP on APP]

```
SELECT * FROM APP.EMS_EMPLOYEE FETCH FIRST 100 ROWS ONLY;
```

SELECT * FROM APP.EMS_EMP...

Max. rows: 100 Fetched Rows: 3 Matching Rows:

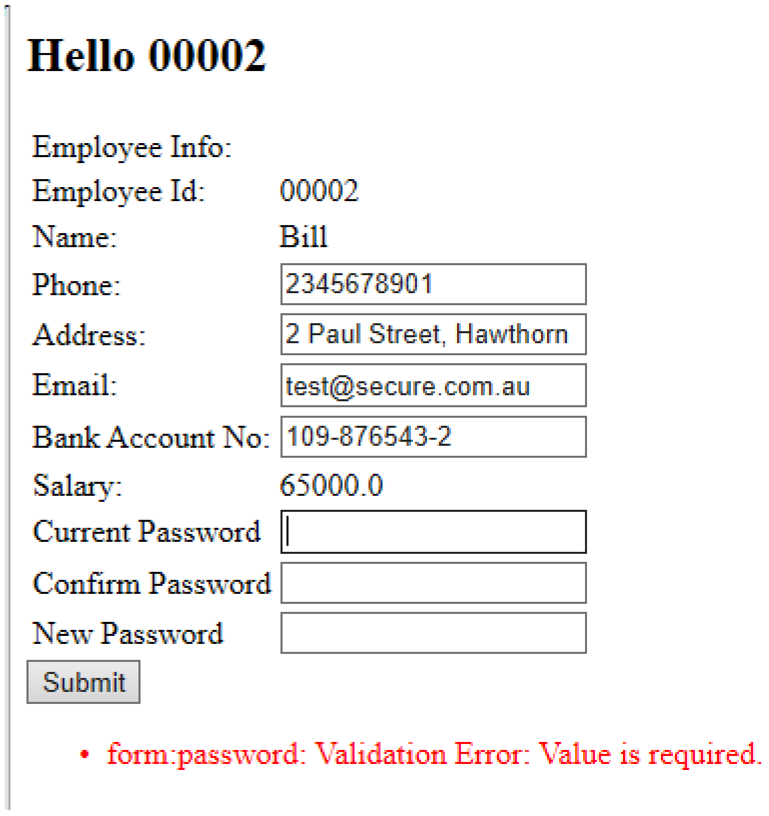
#	EMPID	NAME	PHONE	ADDRESS	EMAIL	PASSWORD	APPGROUP	BANKACCOUNTID	SALARY	ACTIVE
1	00002	Bill	2345678901	2 Paul Street, Hawthorn	test@secure.com.au	33333333	ED-APP-ADMIN	109-876543-2	65000.00	✓
2	00003	Ceci	3456789012	3 Mary Street, Hawthorn	ceci@secure.com.au	33333333	ED-APP-USERS	210-987654-3	75000.00	✓
3	00004	Dave	4567890123	4 Pete Street, Hawthorn	dave@secure.com.au	44444444	ED-APP-USERS	321-098765-4	100000.00	✓



Hello 00002

Employee Info:	
Employee Id:	00002
Name:	Bill
Phone:	2345678901
Address:	2 Paul Street, Hawthorn
Email:	test@secure.com.au
Bank Account No:	109-876543-2
Salary:	65000.0
Current Password	
Confirm Password	
New Password	

Password is required to submit updated data

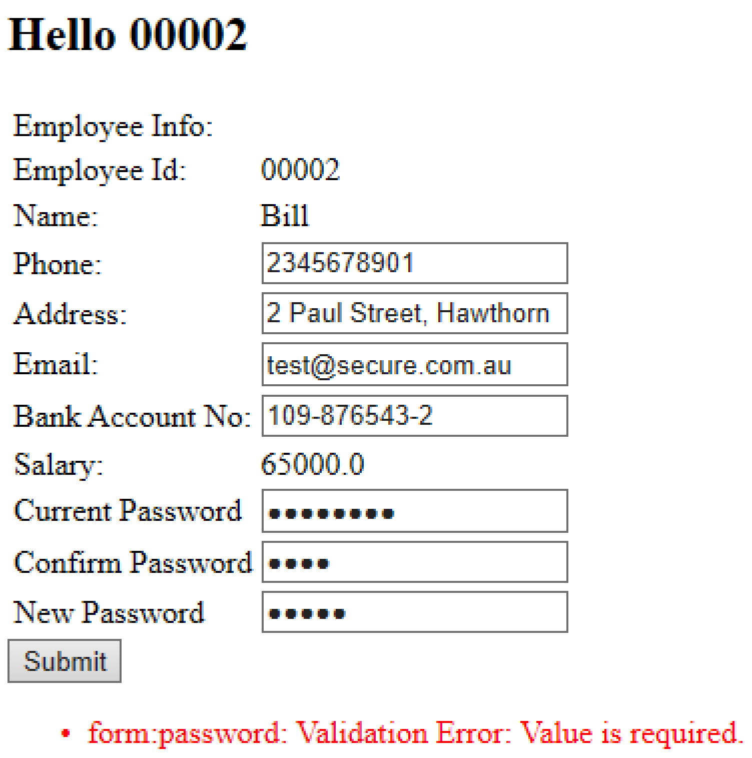


Hello 00002

Employee Info:	
Employee Id:	00002
Name:	Bill
Phone:	2345678901
Address:	2 Paul Street, Hawthorn
Email:	test@secure.com.au
Bank Account No:	109-876543-2
Salary:	65000.0
Current Password	
Confirm Password	
New Password	

- form:password: Validation Error: Value is required.

Passwords must match to submit updated data



Hello 00002

Employee Info:	
Employee Id:	00002
Name:	Bill
Phone:	2345678901
Address:	2 Paul Street, Hawthorn
Email:	test@secure.com.au
Bank Account No:	109-876543-2
Salary:	65000.0
Current Password	●●●●●●●●●
Confirm Password	●●●●
New Password	●●●●●

Submit

- form:password: Validation Error: Value is required.