Swinburne University of Technology

Creating Secure and Scalable Software (S1 2020)

Doubtfire Submission

05 Pass_Task 5.1P

Submitted By:
Thomas Wright
101470604
2020/03/27 12:46

Tutor:
Naurin Afrin

March 27, 2020

5.1

1. Lab 5

Questions:

1. *What if you run the "ED-JEE-DTO-war" project instead?*

The window opens and throws an error: HTTP Status 404 - /index.xhtml Not Found in ExternalContext as a Resource.

2. *What if you open a browser and type the url as http://localhost:8080/ED-JEE-DTO/faces/mainmenu.xhtml?*

The browser throws a 404 page not found error.

3. *What if the url http://localhost:8080/ED-JEE-DTO-war/mainmenu.xhtml?*

The page partially loads. Only the list of hyperlinks loads, the title does not.

4. *What if the url is http://localhost:8080/ED-JEE-DTO-war/faces/mainmenu.xhtml?*

The page fully loads and behaves as expected

2. Programming & Test Cases:

1. The user searches their id

From the main menu 'Add a new user' the following page is displayed to allow the user to search for their record.

Case 1: The id field is left blank

An error is displayed and the user cannot continue

Find User

Please Enter your user id to continue

User Id: Submit

- The userid field cannot be empty!

In the event a valid ID Is entered, the search method is done via the following piece of code:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Search for User Id</title>
    </h:head>
    <h:body>
        <h1> Find User </h1>

        <h:form>
            <h2> Please Enter your user id to continue </h2>
            <h:outputText value="User Id: "/>
            <h:inputText id="userid" value="#{myuserManagedBean.userid}"
                         required="true"
                         requiredMessage="The userid field cannot be empty!"
                         size="6"/>
            <h:commandButton id="submit" value="Submit"
                             action="#{myuserManagedBean.findUser}" />
        </h:form>
    </h:body>
</html>
```

```
// Search for the user & store in flash scope for retrieval
public String findUser() {
    if (isValidUserid(userid)) {
        MyuserDTO myUserDTO = myuserFacade.getRecord(userid);

        if (myUserDTO == null) {
            myUserDTO = new MyuserDTO(userid, "", "", "", "", "", "", "");
        }

        Flash fScope = FacesContext.getCurrentInstance().getExternalContext().getFlash();
        fScope.setKeepMessages(true);
        fScope.put("myuserDTO", myUserDTO);

        return "success";
    }
    return "failure";
}
```

The form calls the findUser method on its submission & redirects the user back to the page if the given ID is not valid. Because the bean is only scoped to exist for a single HTTP request the retrieved user is stored in flash scope to be retrieved in the next page.

On the add user page the following html loads the user data into the form

```
!>

<h:form>
    <f:event type="preRenderView" listener="#{myuserManagedBean.loadFromFlash()}"></f:event>
    <h:panelGrid columns="2">
        <h:outputText value="User Id: "/>
```

The <f:event> tag is a listener for an event to occur so that It can trigger a method. The preRenderView type attribute means that it will call its method on the initial loading of the page before the html is rendered.

```
public void loadFromFlash() {
    Flash fStore = FacesContext.getCurrentInstance().getExternalContext().getFlash();
    MyuserDTO storedUser = (MyuserDTO) fStore.get("myuserDTO");

    if (storedUser != null) {
        //(userid, name, password, email, phone, address, secQn, secAns)
        userid = storedUser.getUserid();
        name = storedUser.getName();
        password = storedUser.getPassword();
        email = storedUser.getEmail();
        phone = storedUser.getPhone();
        address = storedUser.getAddress();
        secQn = storedUser.getSecQn();
        secAns = storedUser.getSecAns();
    } else {
        try {
            FacesContext.getCurrentInstance().getExternalContext().redirect("findUser.xhtml");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Which accesses the flash scope & retrieves the userDTO object stored by the findUser method. If the user attempts to directly navigate to the addUser.xhtml page they will be redirected to the findUser page, as there will be no DTO within flash scope and the else with the redirect call will be made.

Case 2: A valid id is entered, however the user does not already exist

The add user form is submitted with all blank fields.

Add a User

Please enter the user's details below

User Id:	**12345**
Name:	
Password	
Confirm Password	
Email:	
Telephone:	
Address:	
Security Question:	
Security Answer:	

Submit

Case 3: The user enters an id for an existing record

The add user form is displayed with all info except the confirm password pre-filled

Add a User

Please enter the user's details below

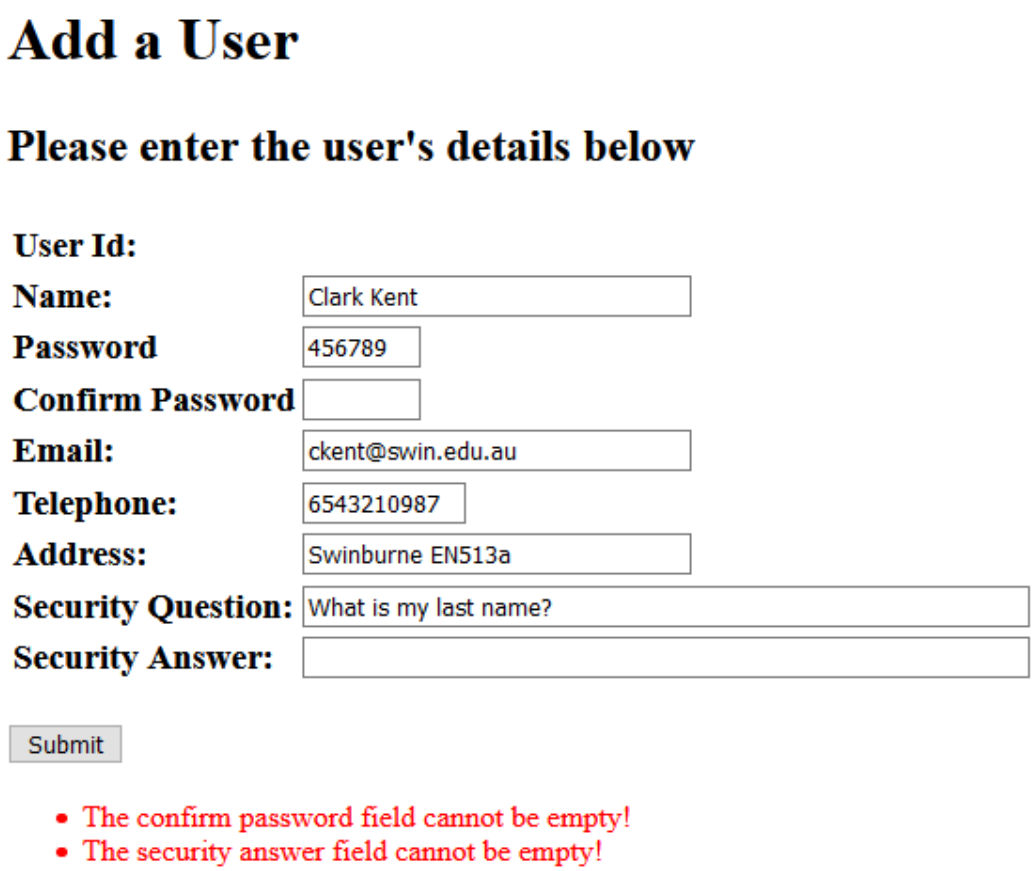
User Id:	**000004**
Name:	Clark Kent
Password	456789
Confirm Password	
Email:	ckent@swin.edu.au
Telephone:	6543210987
Address:	Swinburne EN513a
Security Question:	What is my last name?
Security Answer:	Kent

Submit

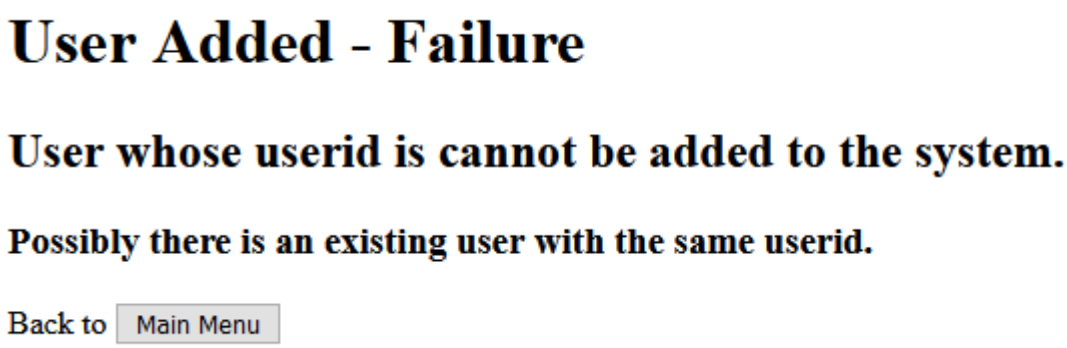
2. **The user is able to update their info.**

Case 4: The user leaves a field blank when updating

The page displays an error and prevents the submission of the form



3. The user submits a form with an invalid email



Whether the user is redirected to addUserSuccess or addUserFailure is not only dependant on whether the information can be inserted into the database, but also whether the user can be notified via email.

```
public String addUser() {
    // Prevents an invalid form submission from erasing data
    saveToFlash();

    if (checkForm()) {
        if (notifyUser()) {
            MyuserDTO myuserDTO = new MyuserDTO(userid, name, password, email, phone, address, secQn, secAns);
            if (myuserFacade.getRecord(userid) != null) {
                if (myuserFacade.updateRecord(myuserDTO)) {
                    return "success";
                }
            } else {
                System.err.println("New user");
                if (myuserFacade.createRecord(myuserDTO)) {
                    return "success";
                }
            }
        }
    }
    return "failure";
}
```

If the form has all its fields set, and it is able to notify the user then it will attempt to either create a new record or update an existing one. If all 3 conditions are met then the user is redirected to the addUserFailure.xtml page.

The following code attempts to send the user an email

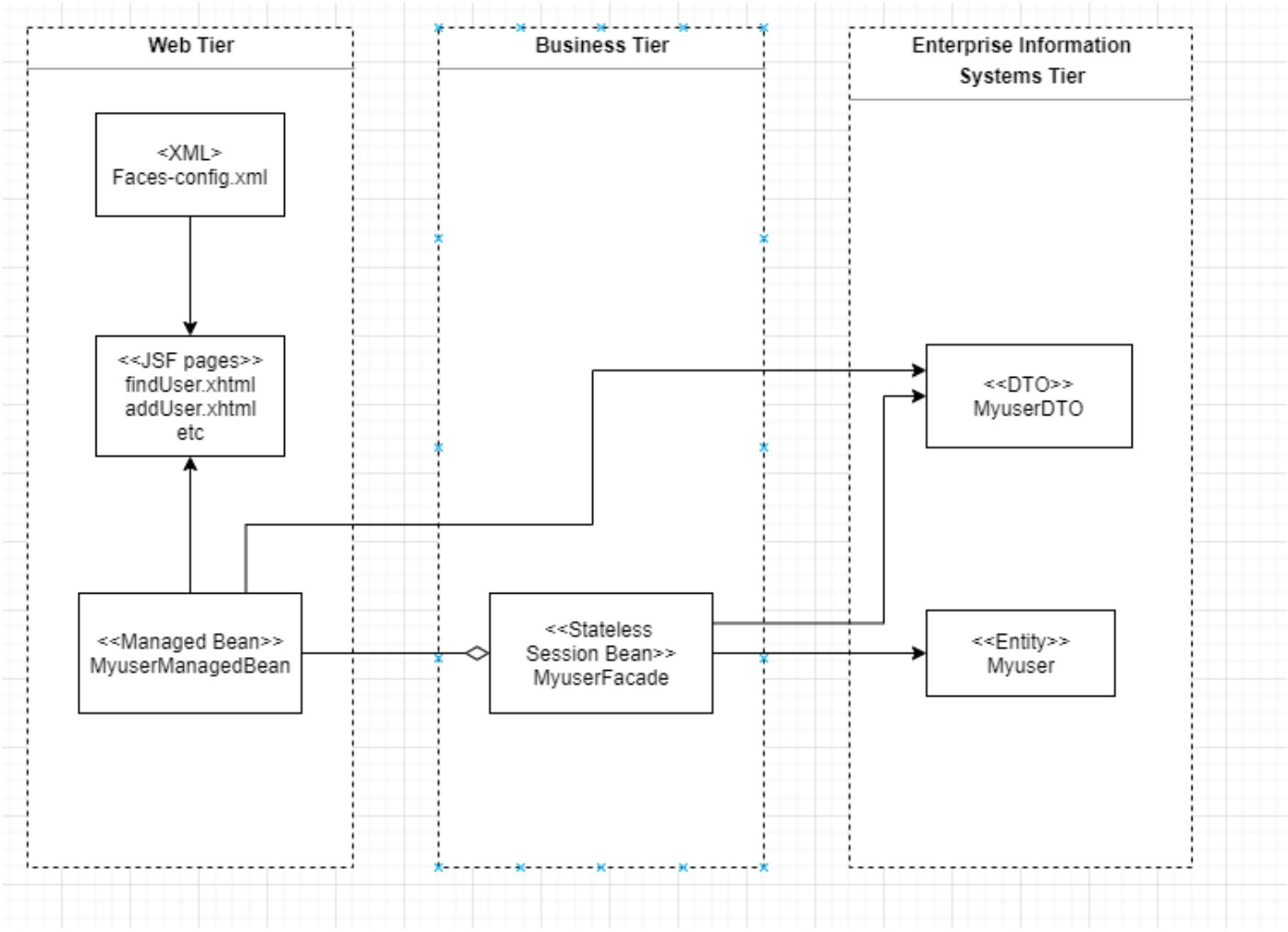
```
private boolean notifyUser() {
    String smtpServer = "smtp-mail.outlook.com";
    String from = "101470604@student.swin.edu.au";
    String to = email;
    String subject = "User info changed";
    String body = "Hi " + name + ",\nYour user information has been updated in our records"
            + "\nIf this was not performed by you please contact us at xyz@student.swin,edu.au,"
            + "\nRegard, 5.1P Development Team\n";
    String emailUser = from;
    String password = "";
    try {
        Properties props = System.getProperties();
        // -- Attaching to default Session, or we could start a new one --
        props.put("mail.smtp.host", smtpServer);
        props.put("mail.smtp.port", 587);
        props.put("mail.smtp.auth", true);
        props.put("mail.smtp.starttls.enable", true);
        // -- prepare a password authenticator --
        MyAuthenticator myPA = new MyAuthenticator(emailUser, password); // see MyAuthenticator class
        // get a session
        Session session = Session.getInstance(props, myPA);
        // -- Create a new message --
        Message msg = new MimeMessage(session);

        // -- Set the FROM and TO fields --
        msg.setFrom(new InternetAddress(from));
        msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse(to, false));
        // -- Set the subject and body text --
        msg.setSubject(subject);
        msg.setText(body);
        // -- Set some other header information --
        msg.setHeader("X-Mailer", "Outlook");
        msg.setSentDate(new Date());
        // -- Send the message --
        Transport.send(msg);
        Transport.send(msg, emailUser, password);
        System.out.println("Message sent OK.");
        return true;
    } catch (Exception ex) {
        ex.printStackTrace();
        return false;
    }
}
```

This code is almost entirely copy pasted from the 5.1 task pdf.

4. Design Justification

4.1 Architecture Diagram



Faces-config references the JSF pages & defines the navigation rules between the pages.

The JSF pages contain a reference to MyuserManagedBean and access its properties to fill out data on the web pages.

MyuserManagedBean holds a reference to MyuserFacade internally. It uses MyuserFacade to perform database operations to retrieve data to be held in its properties and to persist data entered by the user.

MyuserManagedBean has an association relationship with MyuserDTO as it has to create MysuserDTO's to persist the data & must retrieve them in the same form from MyuserFacade.

MyuserFacade acts as the DAO and is the only class that uses Myuser. It uses the Myuser objects returned by the entity manager to alter the database & returns the readonly MyuserDTO's in its methods so that the classes calling its methods cannot alter the data themselves.

4.2Roles

Faces-Config is responsible for defining the navigation between the JSF pages

The JSF pages define the html and behaviour of the web pages.

MyuserManagedBean provides data to the JSF pages, and a means for the JSF pages to send & retrieve data to the database.

MyuserFacade is the data access object and is responsible for defining database operations.

MyuserDTO is a readonly representation of the data retrieved from the database. It is also used as the data transfer object to be sent over a network between Java EE components.

Myuser is the entity class that defines the Myuser table in the database. Myuser objects also alter existing entries in the database.

4.3 Web Page Decisions

The only major design decision I have made is the addition of the findUser page. In this page there is a form with a single text input that takes a userId. The reasoning behind this was simply to get the webpage functioning as quickly as possible.

If I were to design the website to be secure I would separate this page into a login page in which a password would also be required, and a create an account page which would automatically generate a userId for the user to use. As it stands the website is extremely insecure as any users information can be accessed so long as you know the users id.

5. Security concerns

5.1 Which email address should be notified?

The users initial email address before the changes were made should be notified. This should be done so that in the event that an intruder attempts to update the users information to use the intruders email, rather than the intruder getting the notification instead the user will receive a email notification to their email address.

5.2 What changes are required to make this happen

When the addUser.xhtml page loads the managed bean will have to store the users initial email address. When the user submits the changes the initial email address will be used to notify the user.