Cyrus Edgren 101606991
Tutor: Wei Lai

5.1P

Task 1.

The lab was completed and the web application was created:

Welcome to Myuser Web Application!

Please select one of the following options

1. Add a new user
2. Display a user
3. Edit a user
4. Delete a user

Add a User

Please enter the user's details below

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

Submit

Cyrus Edgren 101606991
Tutor: Wei Lai

Task 2.

Various pages were created to facilitate the editing of a user.

selectUser.xhtml:

```
<h:head>
    <title>Select User</title>
</h:head>
<h:body>
    <h2>Please enter User Id: </h2>
    <h:form>
        <h:panelGrid>
            <h:outputText value='User Id: ' />
            <h:inputText id='userid' value="#{myuserManagedBean.userid}"
                         required="true"
                         requiredMessage="Plese enter a User Id"
                         size="6" />
        </h:panelGrid>
        <br />
        <h:commandButton id="submit" value="Submit"
                         action="#{myuserManagedBean.searchUser}" />
    </h:form>
</h:body>
```

Cyrus Edgren 101606991
Tutor: Wei Lai

updateUser.xhtml:

```
<h:body>
    <h1>
        Edit a User
    </h1>
    <h2>
        <p>Please enter the user's details below</p>
    </h2>
    <h3>
        <h:form>
            <h:inputHidden value="#{myuserManagedBean.userid}" />
            <h:panelGrid columns="2">
                <h:outputText value="User Id: "/>
                <h:outputText value="#{myuserManagedBean.userid}"/>
                <h:outputText value="Name: "/>
                <h:inputText id="name" value="#{myuserManagedBean.name}"
                             required="true"
                             requiredMessage="The name field cannot be empty!"
                             size="30"/>
                <h:outputText value="Password "/>
                <h:inputText id="password" value="#{myuserManagedBean.password}"
                             required="true"
                             requiredMessage="The password field cannot be empty!"
                             size="6"/>
                <h:outputText value="Confirm Password "/>
                <h:inputText id="cPassword" value="#{myuserManagedBean.cPassword}"
                             required="true"
                             requiredMessage="The confirm password field cannot be empty!"
                             size="6"/>
                <h:outputText value="Email: "/>
                <h:inputText id="email" value="#{myuserManagedBean.email}"
                             required="true"
                             requiredMessage="The email field cannot be empty!"
                             size="30" />
                <h:outputText value="Telephone: "/>
                <h:inputText id="phone" value="#{myuserManagedBean.phone}"
                             required="true"
                             requiredMessage="The telephone field cannot be empty!"
                             size="10" />
                <h:outputText value="Address: "/>
                <h:inputText id="address" value="#{myuserManagedBean.address}"
                             required="true"
                             requiredMessage="The email field cannot be empty!"
                             size="30" />
                <h:outputText value="Security Question: "/>
                <h:inputText id="secQn" value="#{myuserManagedBean.secQn}"
                             required="true"
                             requiredMessage="The security question field cannot be empty!"
                             size="60"/>
                <h:outputText value="Security Answer: "/>
                <h:inputText id="secAns" value="#{myuserManagedBean.secAns}"
                             required="true"
                             requiredMessage="The security answer field cannot be empty!"
                             size="60"/>
            </h:panelGrid>
            <p></p>
            <h:commandButton id="submit" value="Submit"
                             action="#{myuserManagedBean.updateUser}" />
        </h:form>
    </h3>
</h:body>
```

Cyrus Edgren 101606991
Tutor: Wei Lai

faces-config.xml:

```
<navigation-rule>
    <from-view-id>/addUser.xhtml</from-view-id>
    <navigation-case>
        <from-action>#{myuserManagedBean.addUser}</from-action>
        <from-outcome>success</from-outcome>
        <to-view-id>/addUserSuccess.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-action>#{myuserManagedBean.addUser}</from-action>
        <from-outcome>failure</from-outcome>
        <to-view-id>/addUserFailure.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
<navigation-rule>
    <from-view-id>/selectUser.xhtml</from-view-id>
    <navigation-case>
        <from-action>#{myuserManagedBean.searchUser}</from-action>
        <from-outcome>success</from-outcome>
        <to-view-id>/updateUser.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-action>#{myuserManagedBean.searchUser}</from-action>
        <from-outcome>failure</from-outcome>
        <to-view-id>/selectUserFailure.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
<navigation-rule>
    <from-view-id>/updateUser.xhtml</from-view-id>
    <navigation-case>
        <from-action>#{myuserManagedBean.updateUser}</from-action>
        <from-outcome>success</from-outcome>
        <to-view-id>/updateUserSuccess.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-action>#{myuserManagedBean.updateUser}</from-action>
        <from-outcome>failure</from-outcome>
        <to-view-id>/updateUserFailure.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

*** The failures and success pages were almost identical from the ones created in the lab, so for brevity they were excluded here.

Cyrus Edgren 101606991
Tutor: Wei Lai

Some function needed to be added to myuserManagedBean.java also to complete this task.

searchUser():

```
public String searchUser() {
    String result = "failure";

    if (isValidUserid(userid)) {
        MyuserDTO myuser = myuserFacade.getRecord(userid);
        if (myuser != null) {
            result = "success";
        }
    }

    return result;
}
```

updateUser():

```
public String updateUser() {
    String result = "failure";
    if (isValidUserid(userid) && isValidName(name)
            && isValidPassword(password) && isValidPassword(cPassword)
            && isValidEmail(email) && isValidPhone(phone)
            && isValidAddress(address) && isValidSecQn(secQn)
            && isValidSecAns(secAns) && password.equals(cPassword)) {
        MyuserDTO myuserDTO = new MyuserDTO(userid, name,
                password, email, phone, address, secQn, secAns);
        if (myuserFacade.updateRecord(myuserDTO)) {
            result = "success";
            sendEmail();
        }
    }
    return result;
}
```

Cyrus Edgren 101606991
Tutor: Wei Lai

sendEmail():

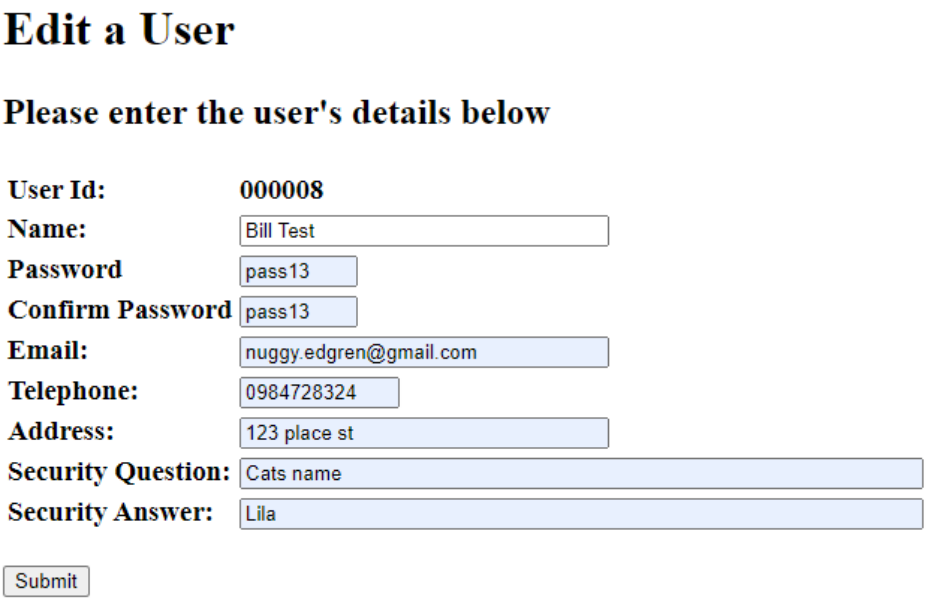
```
private void sendEmail() {
    String smtpServer = "smtp.gmail.com";
    String from = "nuggy.edgren@gmail.com";
    System.out.println(email);
    String to = email;
    String subject = "Testing from gmail";
    String body = "Hi " + name + ",\nYour account info has been changed!\n"
            + "In case this is not done by you, please contact us immediately at xyz@swin.com\nRegards,\nCyrus\n";
    String emailUser = from;
    String password = "[redacted]";
    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", "Gmail");
        msg.setSentDate(new Date());
        // -- Send the message --
        //Transport.send(msg);
        Transport.send(msg, emailUser, password);
        System.out.println("Message sent OK.");
    } catch (Exception ex) {
        ex.printStackTrace();
    }

}
```

Cyrus Edgren 101606991
Tutor: Wei Lai

Task 3.

When the user presses the ‘edit user’ button from the main menu they can search for a user to edit:

If successful and the user is found the page direct to a form to update the found user:

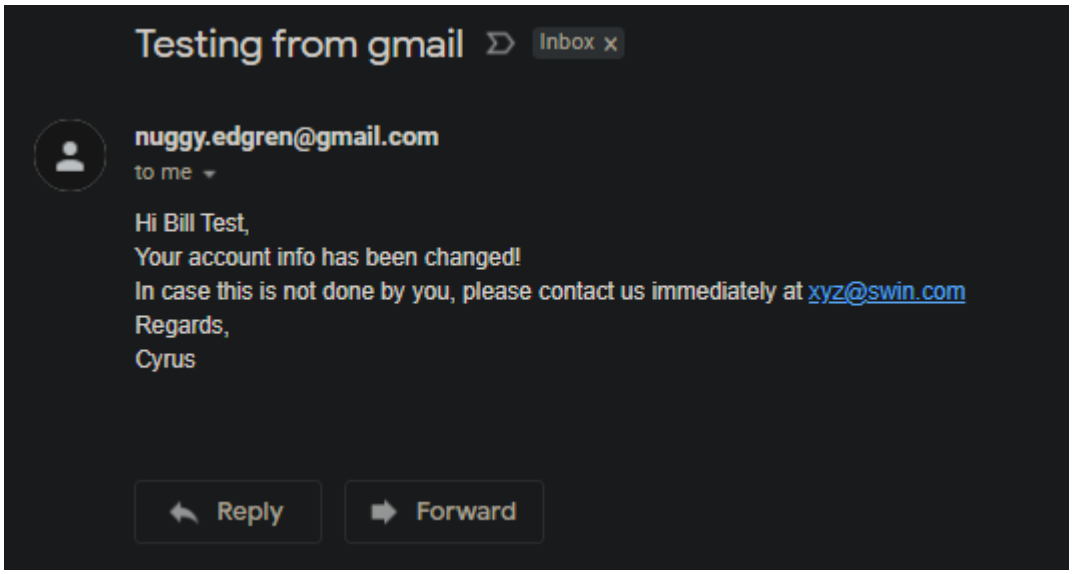


When this is submitted, if successful the user is notified via the web page as well as an email:

User edit - Success

User whose userid is 000008 has been edited.

Back to Main Menu **!**



If either of these steps fail, the relevant page is shown:

User Select - Failure

User whose userid is 000009 cannot be found.

Back to Main Menu

User edit - Failure

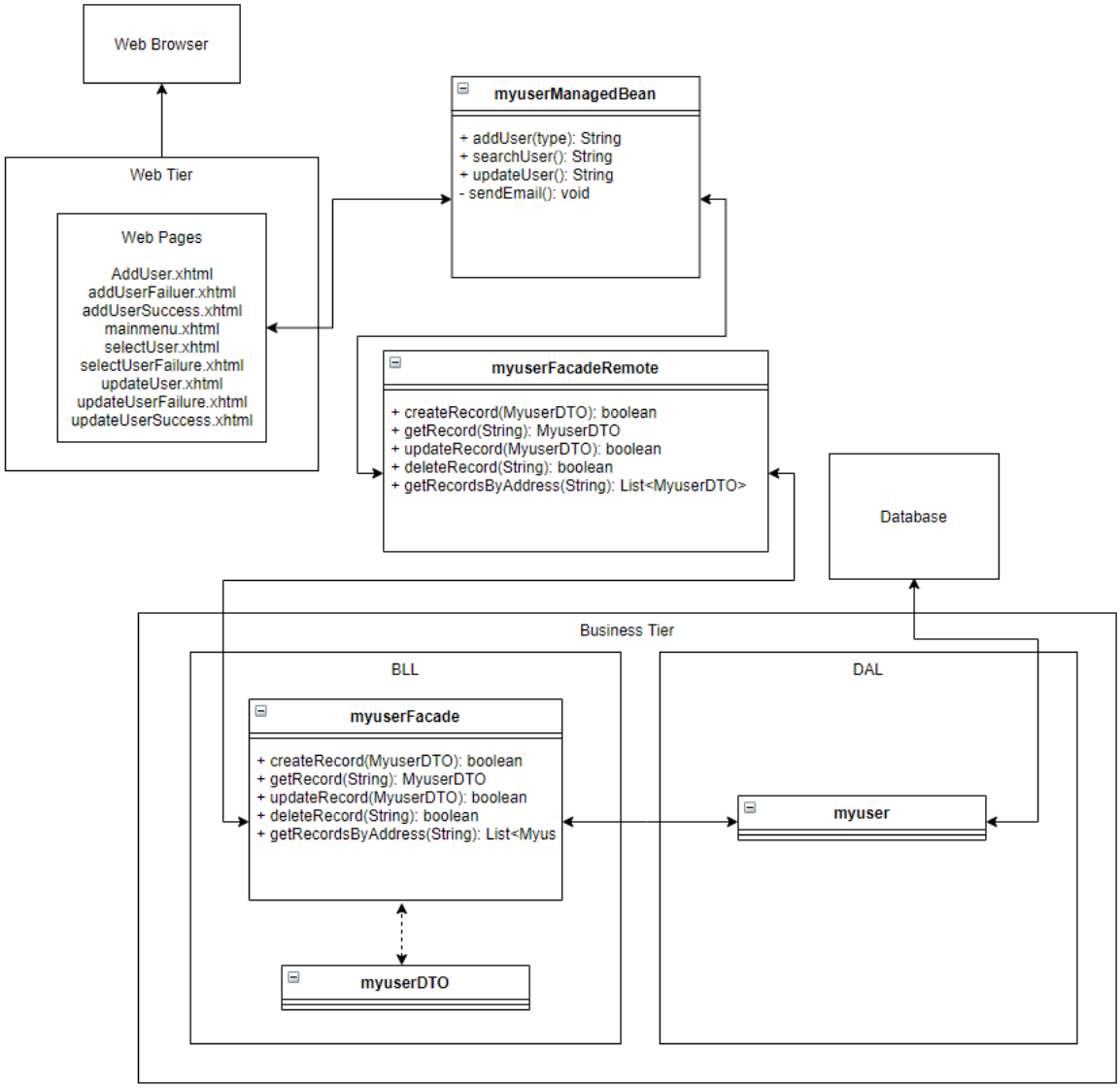
User whose userid is 000008 cannot be edited.

Back to Main Menu

Cyrus Edgren 101606991
Tutor: Wei Lai

Task 4

4.1



Cyrus Edgren 101606991
Tutor: Wei Lai

4.2

myuserManagedBean (Java Bean)

myuserManagedBean is the bean that handles tasks from the web interface and uses myuserFacadeRemote to interact with the database.

myuserFacade (Stateless Session Bean)

myuserFacade is the stateless session bean in charge of performing business logic with the DTO (this involves CRUD operations) and changing the data into the DAO for use with the database.

myuserFacadeRemote (Remote Interface for Stateless Session Bean)

myuserFacadeRemote is the remote interface that describes what the myuserFacade bean is able to do. Other beans or classes can use the interface to perform tasks with the related bean.

myuserDTO (DTO)

myuserDTO is the data transfer object which is created whenever user data needs to be stored in an object and used or manipulated. Does not directly manipulate the database, it just holds the data that is passed around by beans.

myuser (DAO / Entity Class)

myuser is the data access object in charge of taking data and making queries to change data in the database as well as making queries to retrieve data..

Task 5

5.1

When the user's details have been changed I think it would be best to alert both the old and new emails of the change, this way if the change was not intended the user can take action or if it was they then know the new email was entered correctly. The largest improvement to the security of account changes would be to add two factor authentication to the login process. This could be done with a separate device or an email based solution. With protection like this the chance of an account being able to be hack and so having details changed by a third party is made much less likely. Another simpler solution would be to have a confirmation email sent to the old email address of the user and the changes would not be made unless a link was clicked or code entered from the confirmation email. While this is more secure as this would require the user's email to also be compromised, this also introduces the problem of if the user no longer has access to the old email then the changing of account details would not be possible.

5.2

For the changes in the actual program, two factor authentication would probably be done more in the login/authentication part of the system which was not actually created in this task. The email confirmation could be done in the myuserManagedBean when you would send the email alerting the user to changes, instead send a confirmation email that contains a link that then triggers the rest of the database change process.