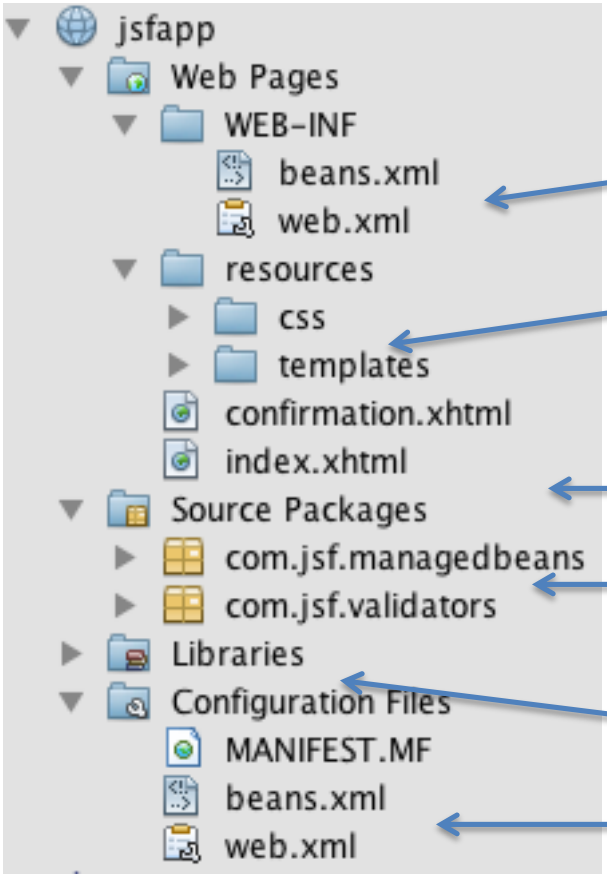
Labsheet 2 – Web tier: Creating a simple JSF application in Netbeans

Note: Netbeans has a range of tutorials and other useful resources covering all technologies on this module. These can be accessed via their learning trail for Java EE and web applications here: https://netbeans.org/kb/trails/java-ee.html

More information on Netbeans support for JSF including shortcuts can be found here: https://netbeans.org/kb/docs/web/jsf20-support.html

Typical folders in the web tier in Netbeans:



Pages within WEB-INF can NOT be accessed directly from a browser, but can be called from other web pages within the application.

Other resources like style sheets, images and templates

Pages that can be accessed directly from a browser.

Java programs associated with the web tier like backing beans and validators

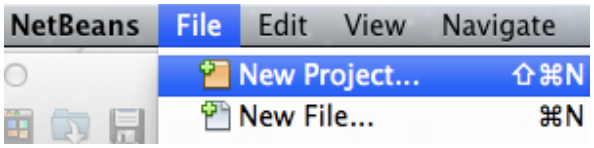
The LIBRARIES folder holds JAR files used by the application

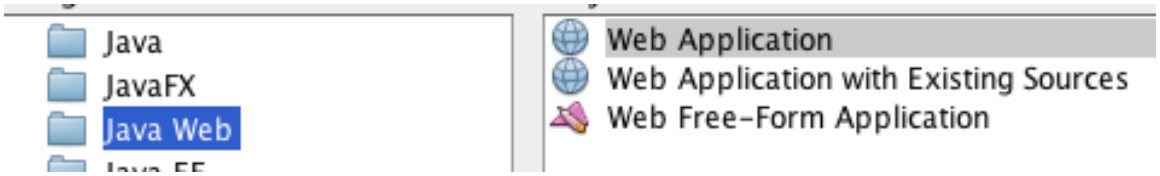
Configuration files generated by the JSF framework (note: these also appear above in WEB-INF)

The following steps will bring you through creating a simple application that accepts student registration details, validates the data, and echoes it back on the next screen.

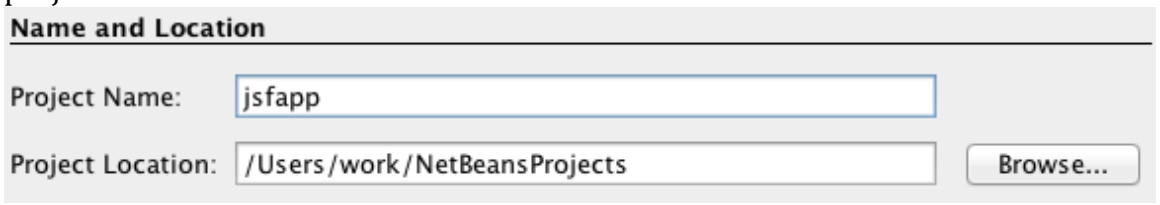
Labsheet 2 – Web tier: Creating a simple JSF application in Netbeans

1. In NETBEANS, create a new project.
2. Choose Java Web > Web Application and click next.

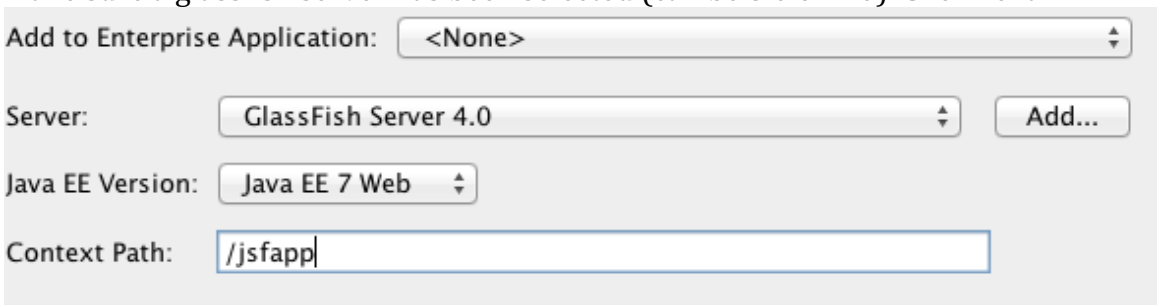




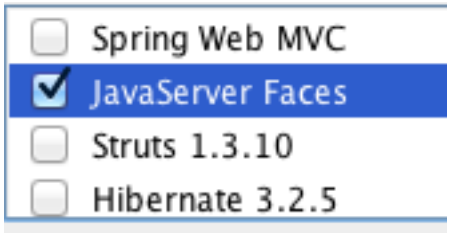
3. Call the project **jsfapp**, and choose a location on your machine to store the project. Click next.



4. Make sure a glassfish server has been selected (can be 3.0 or 4.0). Click next.



5. Select the JavaServer Faces framework, and leave everything else at the defaults. Click finish.



6. Take a look at what has been generated. The screen shot on page 1 above will explain the folders generated. The application will have one page called index.xhtml.
7. Right click on the application name a select RUN to run it. This should open your default browser, and display a page saying "Hello from Facelets".

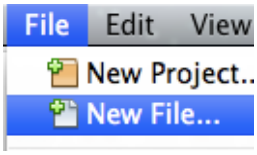
8. Adding a Form to the project.

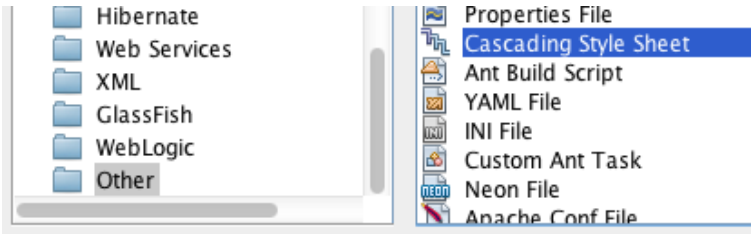
The code below is a JSF form to accept a course, firstname, last name, age and email. As you will see it is very similar to an XHTML form. Add the form to index.xhtml. See lecture notes for an explanation of the code.

```
<h:form id="studentForm">
    <h:panelGrid columns="3" columnClasses="rightalign,leftalign,leftalign">
     <h:outputLabel value="Course" for="course" />
     <h:selectOneMenu id="course" label="Course">
       <f:selectItem itemLabel="" itemValue=""/>
       <f:selectItem itemLabel="Computing" itemValue="Comp"/>
       <f:selectItem itemLabel="Digital Media" itemValue="DigM"/>
       <f:selectItem itemLabel="Engineering" itemValue="Eng"/>
     </h:selectOneMenu>
     <h:message for="course"/>

     <h:outputLabel value="First Name" for="firstName"/>
     <h:inputText id="firstName" label="First Name" required="true" />
     <h:message for="firstName" />

    <h:outputLabel value="Last Name" for="lastName"/>
     <h:inputText id="lastName" label="Last Name" required="true" />
     <h:message for="lastName" />

     <h:outputLabel value="Age" for="age"/>
    <h:inputText id="age" label="Age" size="2"/>
   <h:message for="age" />
```

```
        <h:outputLabel value="Email" for="email"/>
        <h:inputText id="email" label="Email" required="true" />
       <h:message for="email" />

        <h:panelGroup/>  <!--create an empty cell-->
        <h:commandButton id="register" value="Register" action="confirmation"/>
<!--static navigation, use name of page without xhtml extension-->
       </h:panelGrid>
</h:form>
```

Form layout is defined by a panel grid, which references entries in a style sheet. Create folders resources/css under Web Pages. Create a style sheet, call it styles.css, and save it under resources/css. (Folder names are in lower case; don't forget the 's' in 'resource**s'.**)





Put the following class definitions in the style sheet:

```
root {
   display: block;
}
.leftalign {
   text-align: left;
}
.rightalign {
   text-align: right;
}
.centered {
   margin-left:auto;
   margin-right:auto;
}
```

```
.centered-text {
   text-align:center;
}
.rightalign-bold {
   font-weight: bold;
   text-align:right;
}
```

Place the following between the head tags in index.xhtml to link the page to the style sheet:

```
<h:outputStylesheet library="css" name="style.css"/>
```

Note: outputStylesheet renders a style file located in Web Pages/resources/css; it is convert <link type="text/css" rel="stylesheet" href="/JavaServerFaces/faces/javax.faces.resource/style.css?ln=css" />.
If you have a typo in folder names it will not find the css file.

9. Run the application again to view the form

---

10. Adding a backing bean.

To echo this information on another page, the information entered by the user must be stored somewhere. This is done by associating the input texts with attributes defined in a backing bean. To create a backing bean, you need to add a Java program to the project.

Under File > New File select JavaServer Faces, and JSF Managed Bean. Click next.

Call the bean RegistrationBean, and place it in a package called jsf.managedbeans, Make sure the scope is **request**. Leave other defaults and click finish.

In the bean, create variables for each input as follows:

```
private String course;
private String firstName;
```

private String lastName;
private Integer age;
private String email;

Add a **get** and **set** method for each attributes as follows:
Under the declaration statements for the five attributes, **right click > select import code > select getter and setter** methods, and click on all attributes.

11. Linking the backing bean to the JSF form.

    Return to index.xhml. Each attribute in the backing bean must now be linked to its corresponding input on the XHTML form. Below is the code to link up firstname. **Complete this for all inputs:**

    <h:inputText id="firstName" label="First Name" required="true" **value="#{registrationBean.firstName}"**/>

    For the attribute course, add the value parameter to the selectOneMenu tag.

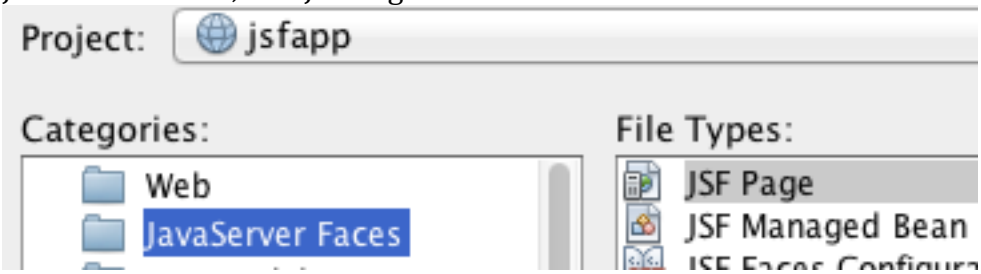
    Note: the name of the bean is the same as the name of the class you created, with the first letter changed to lower case.

    Doing this assigns a data type to each input as defined in the backing bean.
    Run the application again, and try entering text for the age.

---

12. CREATE OUTPUT PAGE

The final part of this simple JSF application is a confirmation page that echoes the contents from the first page. Under File > New File select JavaServer Faces, and JSF Page. Click next.



Under file name, call the page **confirmation**, and click finish. This will add a second XHTML page to the application.

13. The code below gives the code to echo the course name, structured again within a panel grid. Add outputs for the remaining attributes within the panel grid tags.

```
<h:head>
    <title>Confirmation Page</title>
    <h:outputStylesheet library="css" name="styles.css"/>
  </h:head>
  <h:body>
    <h3>Confirmation Page</h3>

    <h:panelGrid columns="2" columnClasses="rightalign-bold,normal" >

    <h:outputText value="Course:" />
    <h:outputText value="#{registrationBean.course}" />

<!-- PUT THE REST OF THE CODE HERE, TO ECHO NAME, AGE AND EMAIL
ADDRESS  -->

</h:panelGrid>
  </h:body>
</html>
```

14. **Run the application again** and click the 'register button' on page 1 to navigate to page 2.
    Note: The command button on index.xhtml already has **action** set to '**confirmation'** and so will navigate to 'confirmation.xhtml'

---

15. Adding form validation.

There is already some validation on the form – firstname and lastname are mandatory inputs (required=true), and age must be numeric. Run the application to test these if you haven't already done so.

We are going to add two additional validations to the form, a built in validator for **age**, and a custom validator for **email**.

Open index.xhtml, and validate that age is between 16 and 99 as follows.
Make sure the <h:inputText> tag is closed AFTER <f:validate…>

```
<h:inputText id="age" label="Age" size="2"
value="#{registrationBean.age}">
        <f:validateLongRange minimum="18" maximum="99"/>
</h:inputText>
```

Run the application, and enter an age of 15 to test the validation.

16. To validate the email address, we will use custom validation, implemented in a Java program. Before created the java program, reference it from the email input text in the XHTML form as follows:

```
<h:inputText id="email" label="Email" required="true"
value="#{registrationBean.email}">
            <f:validator validatorId="emailValidator"/>
  </h:inputText>
```

17. Under file > New file > Java > Java class, create a new Java class called EmailValidator, and store it in the package jsf.validators

The class must implement javax.faces.validator.Validator and have one method called validate(). The code is below to validate email address, as explained in lectures notes.

```
package jsf.validators;


import java.util.regex.Matcher;

import java.util.regex.Pattern;

import javax.faces.application.FacesMessage;

import javax.faces.component.UIComponent;

import javax.faces.component.html.HtmlInputText;

import javax.faces.context.FacesContext;

import javax.faces.validator.FacesValidator;

import javax.faces.validator.Validator;

import javax.faces.validator.ValidatorException;


@FacesValidator("emailValidator")

public class EmailValidator implements Validator {

@Override

  public void validate(FacesContext facesContext,

      UIComponent uIComponent, Object value) throws

      ValidatorException {
```

```
    Pattern pattern = Pattern.compile("[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-
z]{2,4}");
    Matcher matcher = pattern.matcher(
        (CharSequence) value);
    HtmlInputText htmlInputText =
        (HtmlInputText) uIComponent;
    String label;

    if (htmlInputText.getLabel() == null
        ||
htmlInputText.getLabel().trim().equals("")) {
      label = htmlInputText.getId();
    } else {
      label = htmlInputText.getLabel();
    }

    if (!matcher.matches()) {
      FacesMessage facesMessage =
          new FacesMessage(label
          + ": not a valid email address");

      throw new ValidatorException(facesMessage);
    }
  }
}
```

Regular expression that the email address must match.

Get the name of the input text's label to add to the error message

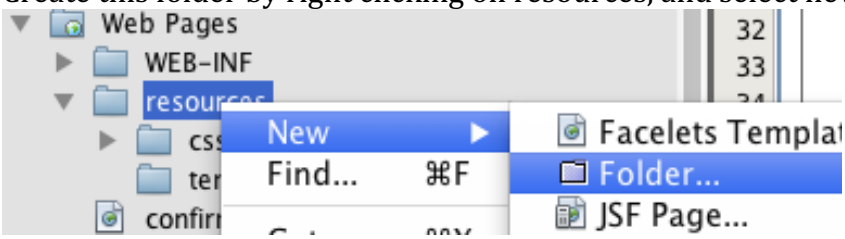
When creating the matcher object above, it was initialised with regular expression and the email address entered. This method compares them.

18. Run the application, and try an invalid email addresses.

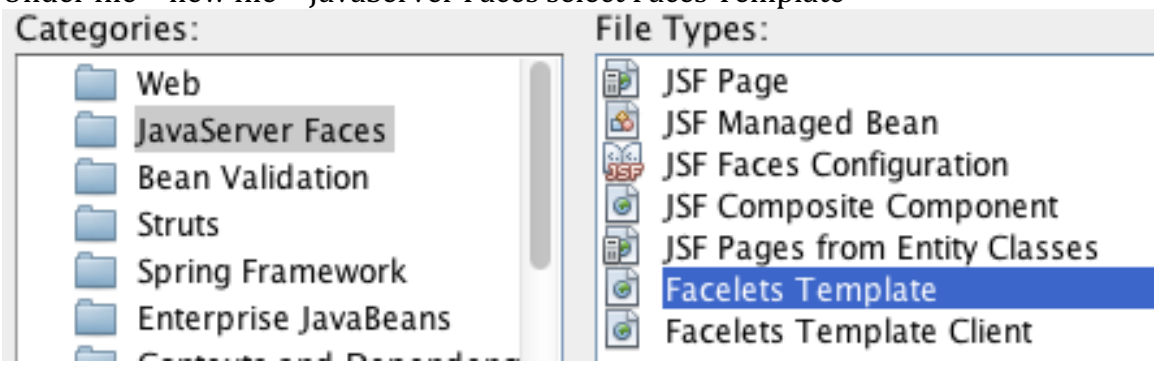
Labsheet 2 – Web tier: Creating a simple JSF application in Netbeans

19. The final thing we will look at is **Facelets Templates** to give a consistent look and feel to all pages in the application. Create a folder called **templates** under **Web Pages / resources**.

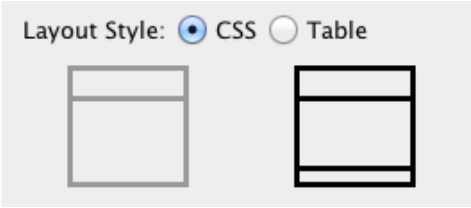
    The template file itself should not be visible to users of the application, and so will be stored in the folder called **templates** within the **resources** folder. Create this folder by right clicking on resources, and select new > folder.



    Under file > new file > JavaServer Faces select Faces Template



    Call the template lab2Template, and store it in the **resources/templates** folder. There are eight predefined layouts to chose from. Choose the 2nd one, which has a header and footer, and click finish.



    You now have a file divided into a top, content and bottom. Give the top section a title like 'College Registration', and in the bottom section include the text: 'created by .....' (giving your name).

```
<div id="top">
        <ui:insert name="top"><h1>College Registration</h1></ui:insert>
    </div>
.....
<div id="bottom">
        <ui:insert name="bottom">create by Geraldine Gray</ui:insert>
    </div>
```

    To use the template, rather than creating a JSF page, you need to create Facelets Template Client pages. Under file > new file > JavaServer Faces

select Faces Template Client. Call the new page **confirmation2**. Under template, navigate to the template you have created, and click finish.

20. Open **confirmation2**.xhml and take a look at the code. As you can see the three sections have been included, nested within a <ui:composition> tag. Where a section is included in a template clients file, it will override the content in the original template. If a section is deleted, then content will be taken from the template itself. Right click on **confirmation2**.xhtml to run just this page. You will see the words top, content and bottom included, but not the header and footer from your original template.

Return to the **confirmation2.**xhtml code, and delete the top section, i.e. delete the three lines below and run the file again. You should now see the top of the page is taken from the template.

```
<ui:define name="top">
    top
</ui:define>
```

21. Finally, change **confirmation2**.xhtml so that the top and bottom are taken from mytemplate.xhtml, but the content section has the code from **confirmation**.xhtml, i.e. echoing the contents of the registration page (copy and paste your code from confirmation.xhtml). Replace the action setting on index.xhtml to navigate to **confirmation2**.xhml, and run the application again.

Note: at run time, if the application cannot find the template's style sheets, try the following paths in your template file:

<h:outputStylesheet name="./resources/css/default.css"/>

<h:outputStylesheet name="./resources/css/tableLayout.css"/>