



Java EE 6 Hands-on Lab
Java Server Faces 2.0 (JSF2) and
Contexts & Dependency Injection 1.0 (CDI)

Arun Gupta
blogs.sun.com/arungupta, @arungupta
Oracle Corporation

Table of Contents

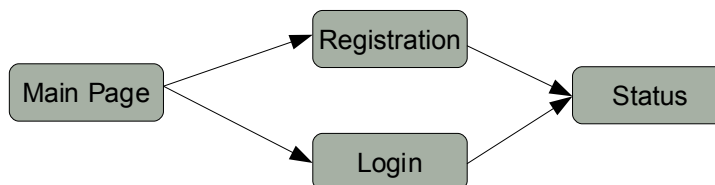
1.0 Introduction.....	3
2.0 Problem Statement.....	3
3.0 Build the template Web application.....	3
4.0 Define CDI “backing bean”s.....	5
5.0 Creating Facelets Template Page.....	7
6.0 Creating and Use “register” Facelets Client Page.....	8
6.1 Update “index” page to use the template.....	8
6.2 Create “register” page.....	9
6.3 Create “status” page	10
6.4 Register a new user.....	11
7.0 Create a Composite Component.....	11
8.0 Create and Use “login” Facelets Client Page.....	13
9.0 Add and use multiple implementations of “backing bean”	14
10.0 Conclusion.....	17
11.0 Troubleshooting.....	17
12.0 Possible Improvements.....	18
13.0 Completed Solutions.....	18

1.0 Introduction

This hands-on lab provide detailed instructions on how to use some of the Java Server Faces 2.0 and Contexts & Dependency Injection 1.0 features with NetBeans 6.9.1 and GlassFish Open Source Edition 3.0.1.

2.0 Problem Statement

This hands-on lab builds a User Registration and Authorization System (URAS) that allows new users to be registered and authorize them. The application consists of four pages and the flow between them is shown below.



A main page guides new users to register and existing users to login. The new users are asked for a user name and password are then shown a status message. The users trying to login are shown a status message indicating if their password matched with the expected.

JSF2 uses Facelets as the default viewing language. Each page will be defined as a Facelet and thus defined using XHTML and CSS only. All the pages will have same look-and-feel and this will be achieved using Facelets templating feature. The relevant data will be stored in a CDI “backing bean”. A welcome message is displayed after the users have registered. Different messages will be shown using different implementations of the bean and marked by CDI Qualifiers.

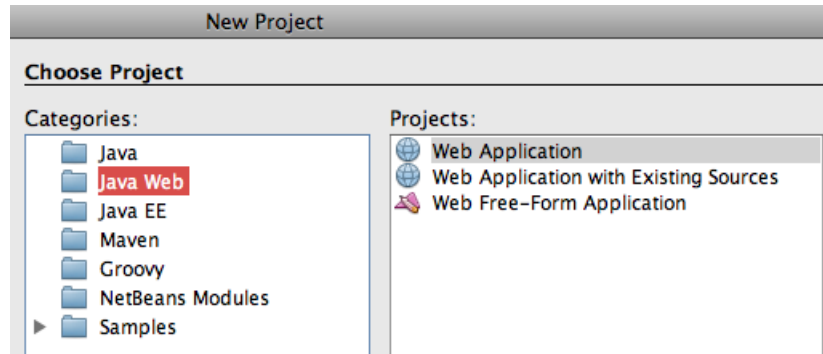
3.0 Build the template Web application

This step will build the template Web application and enable support for JSF and CDI during the project creation phase. The support for these technologies can be enabled after the initial project creation as well but is not covered here.

3.1 In NetBeans IDE, create a new Web application by selecting the “New Project” icon as shown below:



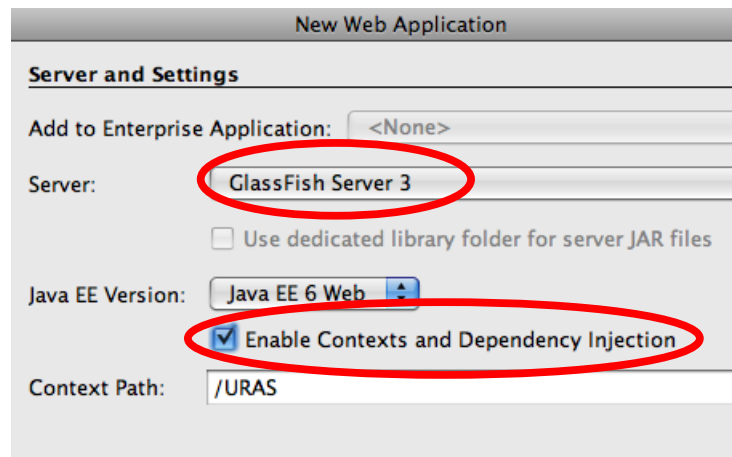
3.2 Choose “Java Web” as category and “Web Applications” as Projects as shown below:



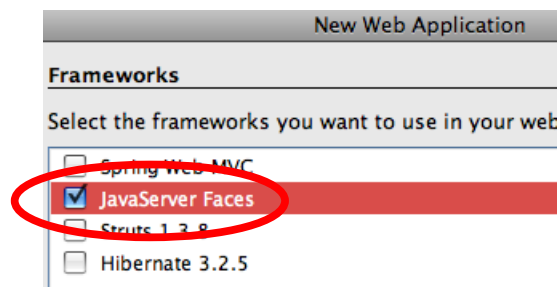
and click on “Next>”.

3.3 Specify the project name as “URAS” and click on “Next>”.

3.4 Choose the pre-configured “GlassFish Server 3” as the Server and click on “Enable Contexts and Dependency Injection” as shown below.



3.5 Click on “Next” and select “JavaServer Faces” as the framework:

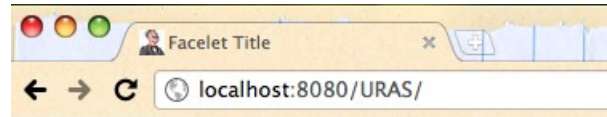


and click on “Finish”. Note the default libraries that come GlassFish are going to be used.

This generates a template Web project with:

- A “web.xml” with “/faces/*” URL pre-registered to be rendered by JSF runtime. This “web.xml” is not required if any JSF annotations are used in the project.
- “web.xml” also lists “faces/index.xhtml” as the welcome file.
- An empty “beans.xml” to enable CDI bean injection of POJOs.

3.6 Right-click on the project and select “Run”. This will start the chosen GlassFish server, deploy the Web application on the server, opens the default web browser, and displays “<http://localhost:8080/URAS/faces/index.xhtml>”. The default page looks like as shown here.



Hello from Facelets

Note, even though “/faces/index.xhtml” is not displayed in the URL window, “web.xml” lists “faces/index.xhtml” as the welcome file and so is displayed accordingly.

A display of this page ensures that the project is successfully created.

4.0 Define CDI “backing bean”s

This section will define CDI beans that will be used as the “backing bean” for the JSF pages and will store the business logic.

4.1 In NetBeans IDE, right-click on “Source Packages”, choose “New>”, “Java Class...”.

4.2 Enter the class name as “User”, package as “org.glassfish.samples”, and click on “Finish”. This bean will be used to store the username and password information for a request.

4.3 Add two private properties of the type “String”, name them “name” and “password”, and add getter/setters for each property. Add @Model annotation on this class. The @Model annotation is a pre-defined CDI stereotype that encapsulates @Named and @RequestScoped annotations as a design pattern. The class looks like:

```
import javax.enterprise.inject.Model;

@Model
public class User {
    private String name;
    private String password;

    // getters and setters
}
```

4.4 In the same package, create a new class “Status”. This bean will be used to store the registration and login status message for different users.

4.3 Change the generated class so that it looks like:

```
import javax.enterprise.inject.Model;

@Model
public class Status {
    String message;

    // getter and setter for "message"
    ...
}
```

This trivial class has one property to store the status message and is also using the `@Model` annotation.

Make sure to add accessor and mutator methods for the properties.

4.4 In the same package, create a new class and name it as “UserService”. Change the generated class so that it looks like:

```
import java.util.HashMap;
import java.util.Map;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.inject.Named;

@Named
@ApplicationScoped
public class UserService {
    private Map<String, String> credentials = new HashMap<String, String>();
    @Inject Status status;
    @Inject User user;

    public void register() {
        if (credentials.containsKey(user.getName())) {
            status.setMessage("User " + user.getName() + " already registered.");
            System.out.println(status.getMessage());
            return;
        }
        if (credentials.put(user.getName(), user.getPassword()) != null) {
            status.setMessage("User " + user.getName() + " not registered.");
            return;
        }
        status.setMessage("User " + user.getName() + " registered successfully.");
    }

    public void login() {
        if (credentials.containsKey(user.getName())
            && credentials.get(user.getName()).equals(user.getPassword())) {
            status.setMessage("User " + user.getName() + " logged in successfully.");
            return;
        }
    }
}
```

```

    }
    status.setMessage("User " + user.getName() + " login failed.");
}
}

```

This class contains two business methods “register” and “login”. The “register” method stores the registration data in a HashMap and “login” method matches the entered and expected password. The registration data is lost each time the application is redeployed as its stored in a Hashmap. However The Java Persistence API can be used here to persist the data in a database as well.

The Status bean is injected using @Inject and the correct status message is set in the two business methods. The User bean is injected using @Inject as well. Note, the bean requesting injection does not need to be aware of the context and scope of the requested bean and the CDI runtime figures out the right bean and injects it appropriately.

The class is marked @ApplicationScoped so that the registration information stays until the app is un-deployed or re-deployed.

Save all the generated files.

5.0 Creating Facelets Template Page

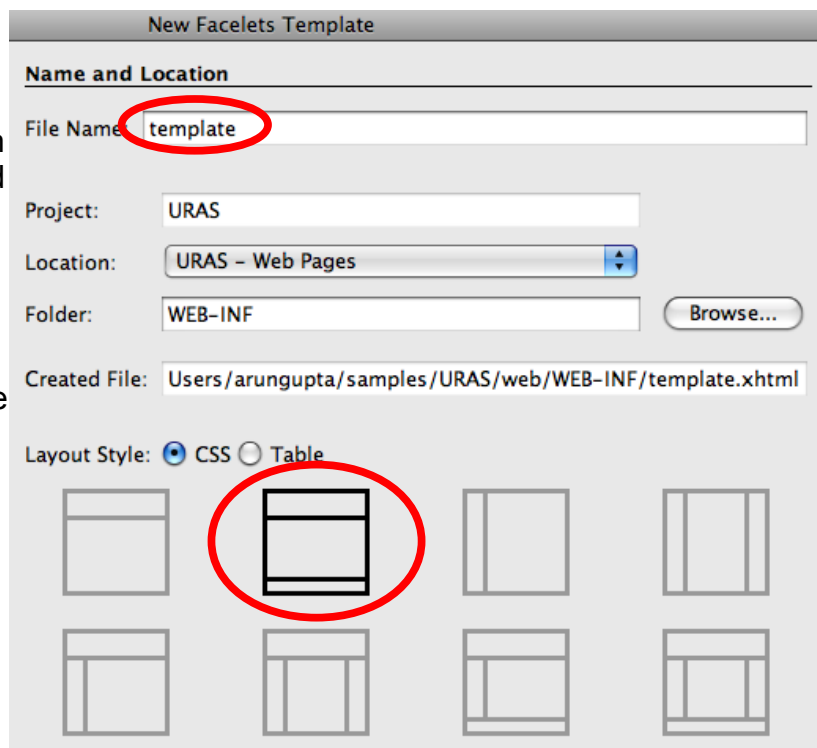
This section will create a Facelets template page that will be used to provide a consistent look-and-feel for other pages.

5.1 Right-click on the project, select “New>”, “Facelets template”.

5.2 Choose the file name as “template”, Folder as “WEB-INF” and select the pre-defined template that has a header, main page, and a footer as shown and click on “Finish”.

This generates “template.xhtml” in the WEB-INF folder and two stylesheets in “resources/css” folder. The “template.xhtml” page contains three <div>s with <ui:insert>s named “top”, “content”, and “bottom”.

Its a recommended practice to



keep the template pages in “WEB-INF” folder such that they are not accessible outside the web application.

5.3 In “template.xhtml”, replace the text “Top” (inside <ui:insert name="top">) with:

```
<h1>User Registration and Authorization System</h1>
```

and replace the text “Bottom” with (inside <ui:insert name="bottom">) with:

```
<center>Powered by GlassFish!</center>
```

The “top” and “bottom” <div>s will provide a consistent look-and-feel to other pages. The “content” <div> will be overridden in other pages to display the business components.

6.0 Creating and Use “register” Facelets Client Page

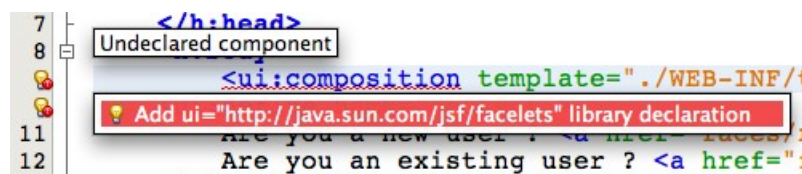
Now lets update existing “index.xhtml” and create new pages that use this template to provide a consistent look-and-feel. The client page inherits from the template page and overrides the required <div>s using <ui:define>. The sections not overridden are inherited from the template page.

6.1 Update “index” page to use the template

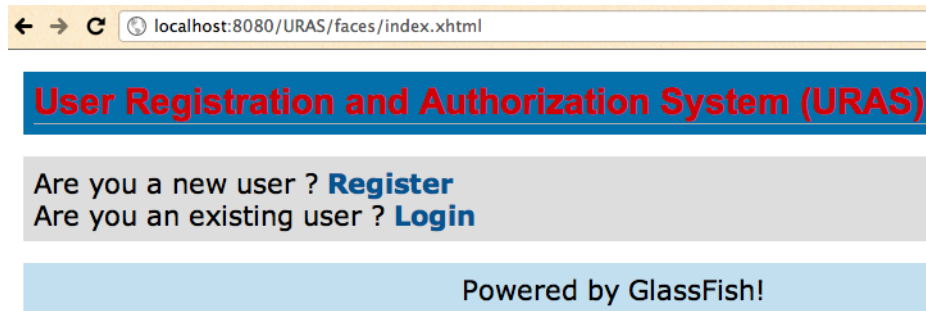
6.1.1 Replace “Hello from Facelets” text in <h:body> element as shown below:

```
<ui:composition template="./WEB-INF/template.xhtml">
  <ui:define name="content">
    Are you a new user ? <a href="faces/register.xhtml">Register</a> <br/>
    Are you an existing user ? <a href="faces/login.xhtml">Login</a>
  </ui:define>
</ui:composition>
```

Note that this page has <ui:composition template='./WEB-INF/template.xhtml'> indicating that this page is inheriting from the template page created earlier. This element only has <ui:define> with name “content” so the other elements, namely “top” and “bottom” are inherited from the template page. Fix the imports by right-clicking on the yellow bulb as shown:



Save the page and the updated page looks like:

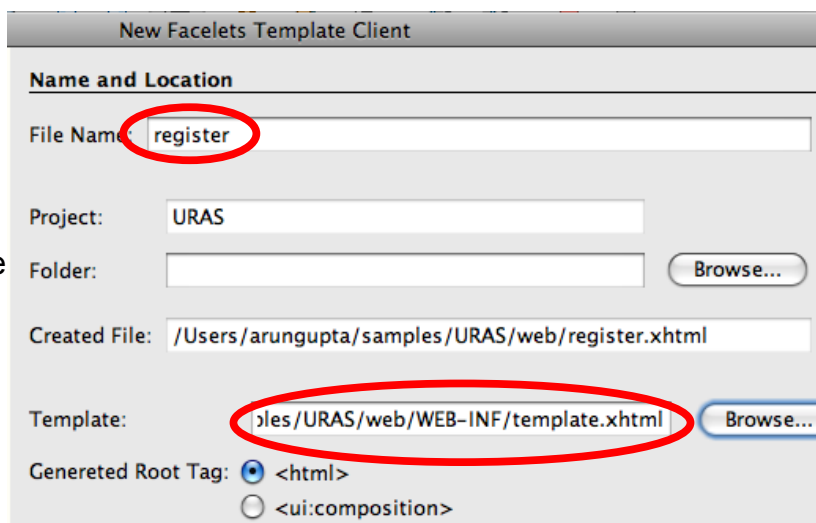


6.2 Create “register” page

6.2.1 Right-click on “Web Pages”, select “Java Server Faces” in Categories and “Facelets Template Client” in File Types. Click on “Next>”.

6.2.2 Enter the file name as “register”, choose the Template after click on “Browse...” button and selecting “WEB-INF/template.xhtml” as shown and click on “Finish”.

Note that the generated page has `<ui:composition template='../WEB-INF/template.xhtml'>` indicating that this page is inheriting from the template page created earlier. It has three `<ui:define>` elements with the exact same name as in the template. This allows specific sections of the template page to be overridden. The sections that are not overridden are inherited from the template.



6.2.3 Delete `<ui:define>` element with name “top” and “bottom” as these sections are already defined in the template.

6.2.4 Replace the text “content” (inside `<ui:define name="content">`) with:

```
<h2>Register New User</h2>
<h:form>
  <h:panelGrid columns="3">
```

```

<h:outputText value="Name:" />
<h:inputText value="#{user.name}" id="name"/>
<h:message for="name" style="color: red" />
<h:outputText value="Password:" />
<h:inputText value="#{user.password}" id="password"/>
<h:message for="password" style="color: red" />
</h:panelGrid>
<h:commandButton actionListener="#{userService.register}" action="status"
value="submit" />
</h:form>

```

As earlier, resolve the import by clicking on the yellow bulb in the left bar. This is an HTML form with 3 columns, accepts the username and password, binds them to the “name” and “password” property of the “user” bean. The form contains a command button with the value of action as “show”. Following the default navigation rules defined by the JSF2 specification, clicking on this button requires a “status.xhtml” file in the same directory. The value of attribute “actionListener” identifies the business method that will be invoked, “register” method of the UserService bean in this case.

6.2.5 Save the files and open “<http://localhost:8080/URAS/faces/register.xhtml>” in a browser window, it looks like as shown below:

Notice the header and footer sections are inherited from the template page.

6.3 Create “status” page

6.3.1 Create a new Facelets client page and name it “status” following the steps defined above. Make sure to choose the template page defined earlier and remove the “top” and “bottom” <div>s.

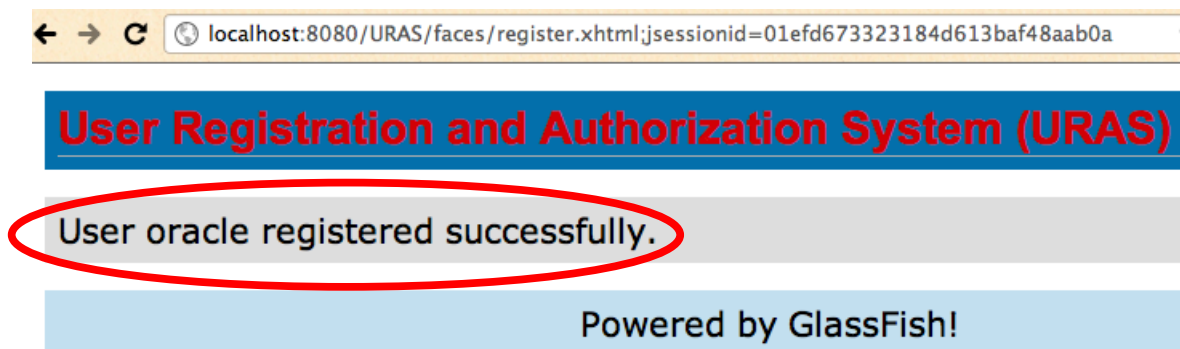
6.3.2 Change the text “content” (inside <ui:define name="content">) with:

```
#{status.message}
```

This page will display the status after the user has registered.

6.4 Register a new user

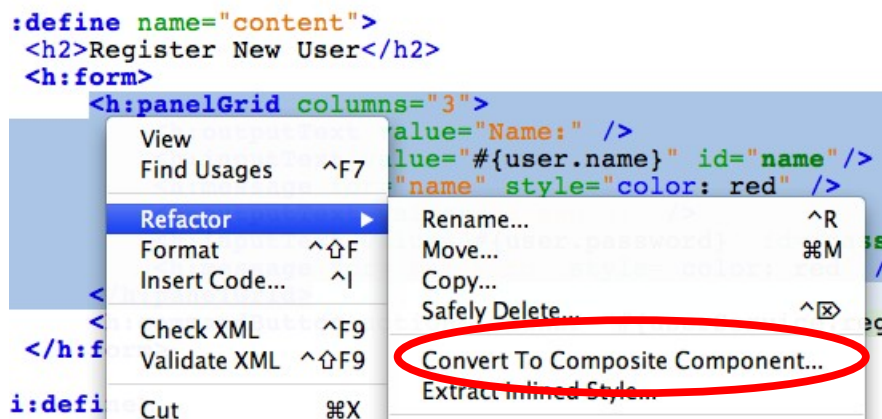
6.4.1 Enter the username as “oracle”, password as “glassfish”, and click on “submit” and the result is shown as:



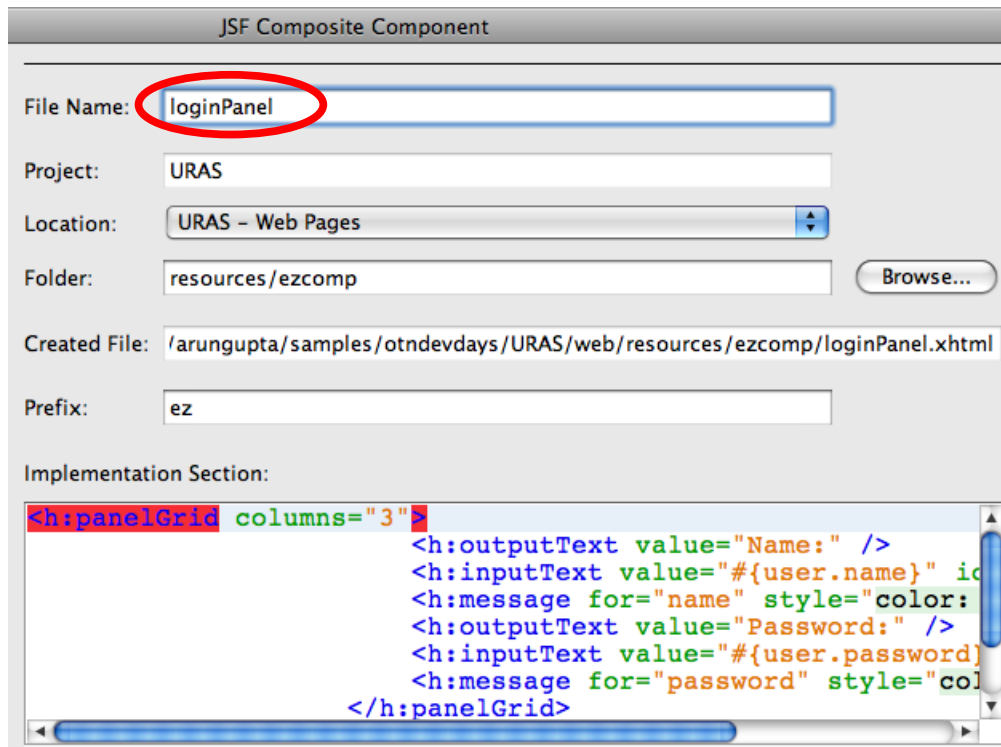
7.0 Create a Composite Component

The composite component allows a code fragment to be abstracted, parameterize, and reused in other places. Lets create a composite component in the “register” page.

7.1 In the “register” page, select the code fragment inside and inclusive of the <h:panelGrid> element, right-click, and select “Convert to Composite Component” as shown.



Change the File Name to “loginPanel” in the wizard as shown:



and click on “Finish”. This moves the selected fragment to “resources/ezcomp/loginPanel.xhtml”, declares a new namespace as “xmlns:ez=“<http://java.sun.com/jsf/composite/ezcomp>” in the “register” page, and replaces the selected code fragment with “<ez:loginPanel/>”. The updated page looks like:

```
<?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-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ez="http://java.sun.com/jsf/composite/ezcomp">

  <body>

    <ui:composition template="./WEB-INF/template.xhtml">

      <ui:define name="content">
        <h2>Register New User</h2>
        <h:form>
          <ez:loginPanel/>
          <h:commandButton actionListener="#{userService.register}"
            action="status" value="submit"/>
        </h:form>
      </ui:define>
    </ui:composition>
  </body>
</html>
```

Just refresh "<http://localhost:8080/URAS/faces/register.xhtml>" and go through the steps outlined in 6.4 to register a new user and ensure that the page is working as expected.

8.0 Create and Use “login” Facelets Client Page

8.1 Create a new Facelets client page and name it “login” following the steps defined above. Make sure to choose the template defined earlier and remove the “top” and “bottom” <div>s.

8.2 Change the text “content” (inside <ui:define name="content">) with:

```
<h2>Login</h2>
<h:form>
    <ez:loginPanel/>
    <h:commandButton actionListener="#{userService.login}"
        action="status" value="submit"/>
</h:form>
```

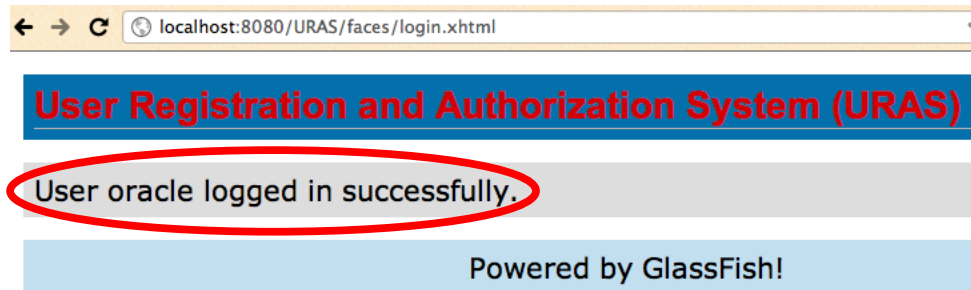
Fix the import by right-clicking on the yellow bulb and manually add the following namespace declaration “xmlns:ez=”<http://java.sun.com/jsf/composite/ezcomp>” in the page. This will use the composite component defined earlier and invokes the “login” method of UserService when the submit button is clicked. This method, as explained earlier, checks the username and password credentials.

The benefit of defining composite component is that only a single line of code, instead of the entire code fragment, can be used now for duplicating the functionality. Note that in this case <h:commandButton> is not included in the composite component but it can easily be included as explained in http://blogs.sun.com/arungupta/entry/totd_147_java_server_faces.

8.3 Open

[“http://localhost:8080/URAS/faces/login.xhtml”](http://localhost:8080/URAS/faces/login.xhtml) in a browser window. Enter the values “oracle” and “glassfish” as shown.

Clicking on the “submit” button shows the following status page:



Notice, all the pages are using same header and footer and there by providing a consistent look-and-feel for the application. If your application has been deployed since you first registered this user, you may have to register again as the credentials are lost during each re-deploy.

9.0 Add and use multiple implementations of “backing bean”

CDI allows multiple types of a bean to be injected in a type-safe way. This means that no String-based identifiers, which are malleable, are used for injection and instead all the information is derived from the Java object model.

Lets change our Status bean to delegate the message handling from another CDI managed bean. Actually this new bean will be an interface with couple of slightly different implementations. This part of the hands-on lab will show how different implementations can be injected and finally see how different status messages are shown based upon the injected bean.

9.1 In the Project Explorer, right-click on the project, select “New”, “Java Interface...”, and specify the name as “StatusMessage” in “org.glassfish.samples” package.

9.3 Add the following two methods to this interface:

```
public String getMessage();
public void setMessage(String message);
```

9.4 In the same package, create a new class “SimpleStatusMessage” class and make it implement the newly created interface. The update definition looks like:

```
public class SimpleStatusMessage implements StatusMessage {
    protected String message;

    @Override
    public String getMessage() {
        return message;
    }

    @Override
    public void setMessage(String message) {
```

```

        this.message = message;
    }
}

```

9.5 In “Status” class, inject “StatusMessage” bean and use that for getting and setting the message by changing the getter/setter methods. The updated class looks like as shown below:

```

@Model
public class Status {

    //    String message;

    @Inject StatusMessage statusMessage;

    public String getMessage() {
        return statusMessage.getMessage();
    }

    public void setMessage(String message) {
        statusMessage.setMessage(message);
    }
}

```

Note that the “message” variable is commented as getting/setting the message is now delegated to the “StatusMessage” bean.

Fix the imports, saving all the files automatically deploys the project, and lets verify that everything works as earlier. So try registering and logging in a user and ensure there is no change in the output.

9.6 Add a new class “SmileyStatusMessage” to the package “org.glassfish.samples”. This class will add a smiley to each status message. Change the class definition as shown below:

```

import javax.enterprise.inject.Model;

@Model
public class SmileyStatusMessage extends SimpleStatusMessage {
    @Override
    public String getMessage() {
        return message + " :-)";
    }
}

```

Notice, this class is also marked with @Model. However the CDI runtime will now get confused because there are two implementations of the “StatusMessage” interface. The CDI runtime will throw the following error:

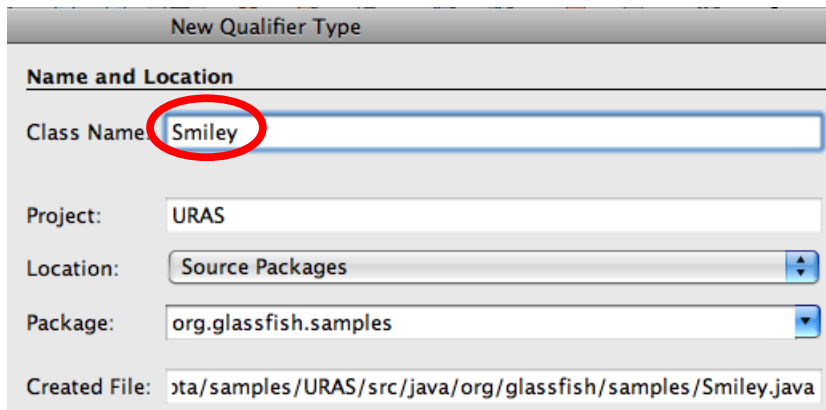
```

Injection point has ambiguous dependencies.

```

and this can be shown in the Output window of the IDE. Lets add a CDI qualifier to clearly mark one of the implementations.

9.7 Right-click on the package “org.glassfish.samples”, select “New>”, “Other...”, “Context and Dependency Injection”, “Qualifier Type”, click on “Next>”, and type the class name as “Smiley” as shown



and click on “Finish”.

This generates the CDI qualifier as:

```
@Qualifier
@Retention(RUNTIME)
@Target({METHOD, FIELD, PARAMETER, TYPE})
public @interface Smiley {
}
```

9.8 Add the annotation “@Smiley” on the class “SmileyStatusMessage” as shown below:

```
@Model
@Smiley
public class SmileyStatusMessage extends SimpleStatusMessage {
```

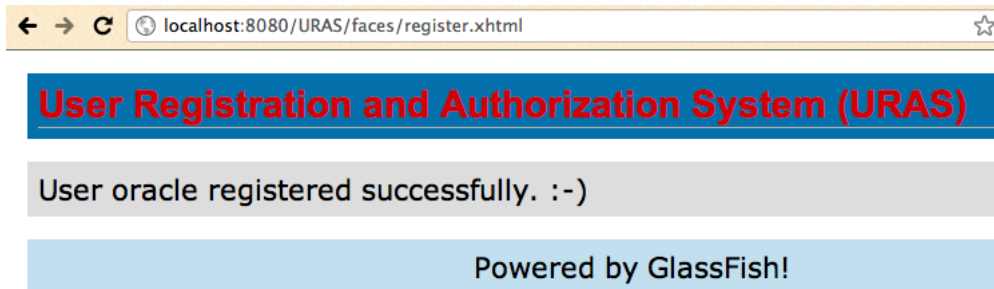
Now the deployment error shown above disappears as the CDI runtime knows that there is only one default implementation of “StatusMessage” interface and so it will inject that. As always, just save the file to deploy the project.

Now the usual register and login flows will work as described above.

9.10 Edit Status class and change the injection of “StatusMessage” bean as:

```
@Inject @Smiley StatusMessage status;
```


This provides enough information to the CDI runtime to inject a unique implementation of the StatusMessage interface. Now registering a user shows the following status message:



Notice the additional “:-)” at the end of the message.

10.0 Conclusion

This hands-on lab created a simple application demonstrating several features of Java Server Faces 2.0 and Context & Dependency Injection 1.0, namely:

- Facelets as viewing language for JSF2
- JSF2 Composite Components
- CDI Typesafe Bean injection
- CDI Stereotype
- CDI Qualifiers

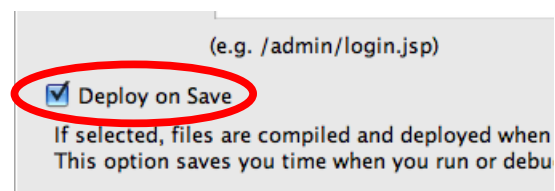
Hopefully this has raised your interest enough in trying out Java EE 6 applications using GlassFish and NetBeans.

Send us feedback at users@glassfish.dev.java.net.

11.0 Troubleshooting

11.1 The project is getting deployed to GlassFish every time a file is saved. How can I enable/disable that feature ?

This feature can be enabled/disable per project basis from the Properties window. Right-click on the project, select “Properties”, choose “Run” categories and select/unselect the checkbox “Deploy on Save” to enable/disable this feature.



11.2 The pages are not displaying the “top” and “bottom” sections.

Make sure the URL pattern includes “/faces/*.xhtml”. This ensures that the page is rendered by JSF and all the components are displayed as expected.

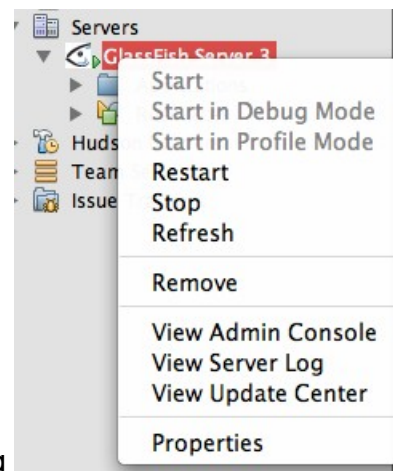
11.3 No status messages are displayed.

Make sure there is an empty “beans.xml” in the “WEB-INF” folder. This is required for CDI injection to work properly.

11.4 How can I start/stop/restart GlassFish from within the IDE ?

In the “Services” tab, right-click on “GlassFish Server 3”, and all the commands to start, stop, and restart are available from the pop-up menu.

The server log can be viewed by clicking on “View Server Log” and web-based administration console can also be seen by clicking on “View Admin Console”.



11.5 The message “User oracle already registered” or something similar is displayed after you edit/save a file and try to register a user. Isn't the Hashtable supposed to clear out with each re-deployment ?

This may happen because the project did not get re-deployed, for some reason, after the file was saved. Deploy the project explicitly by right-clicking on the project and select “Deploy”.

12.0 Possible Improvements

Do you feel motivated and would like to try some enhancements to the application ? Here is one:

12.1 Add links to “login” and “register” page in the header for easy navigation.

13.0 Completed Solutions

The completed solution is available as a NetBeans project in the home directory of the VirtualBox instance by the directory name “URAS”. Open the project in NetBeans, browse

through the source code, and enjoy!

The completed solution can also be downloaded from
<http://blogs.sun.com/arungupta/resource/URAS.zip>.