**CST-235 Application Design and Implementation Guide**

**Objectives:**

1. Create a front end HTML form page and back end Servlet to process data.
2. Create a JSF pages, JSF Model, and backend JSF Controller to process data.

**Developer Notes and Best Practices:**

1. It is typical to map any files with an xhtml file extension and jsf file extension to the JSF Servlet.
2. At the top of all xhtml JSF pages there are a number of name space and tag declarations. This is how the JSF tag prefixes are established. The h tags are for all the core HTML tags. The ui tags are for all the Facelets tags. You should go thru the chapters in your required readings and make sure you understand all the JSF HTML and Facelet tags as they are key to building your JSF pages.
3. Any Java bean that is managed by the JSF runtime and referenced in any JSF pages needs to be have its class marked with the @ManagedBean annotation. A Managed Bean is instantiated and destroyed by the JSF runtime per a managed beans lifecycle rules.
4. Each Managed Bean should also define its scope. Unless you have a good reason to use a scope with a longer lifetime, such as the Application scope, Session scope, Conversation scope, or Custom scope, you should always mark the class with a @RequestScope or @ViewScope annotation.
5. Data binding (bi-directional) between JSF Form and a Managed Bean can be easily done by using the #{bean.property} syntax in your JSF page. Actions can also be specified using the same syntax except you reference a method name (with no parenthesis) rather than a property name.
6. Even though lots of code examples and books do not reinforce this concept, it is recommended that you separate your Object Model managed bean classes from your Controller managed bean classes. JSF is powerful and flexible enough so that you can use multiple beans on a page so you can bind your Object Model (data containers) managed beans to your page and then bind your Controllers action methods to appropriate JSF action buttons. This approach also adheres to a pure and cleaner implementation of the MVC design pattern. This approach also allows you to apply appropriate data annotations to your Object Model managed bean classes again supporting a cleaner implementation.
7. If you want to obtain a managed bean from with the method of a Controller simply use the evaluateExpressionGet() from the Faces Context and specify the name of the managed mean you wish to retrieve.
8. If you have a @RequestScope or @ViewScope that want extended and passed from a Controller class to another View (as part of the Controllers response) you can use the getRequestMap().put() methods from the Faces Context within the implementation of the Controller method.
9. Get in the habit of following good package naming and class naming conventions. Put all your Object Model managed bean classes in a packaged called *beans* or *models*. Put all your Controller managed bean classes in a package called *controllers*. You should also consider using Postfix names for your Object Model and Controller classes. End your Object Model class names with *Model* and end all your *Controller* class names with Controller. Also be aware that the Object Model and Controller class names and all other classes should be named using a noun from your business name. If your Object Model holds a list of models then make your class name plural.
10. One of the powerful features of JSF is that the standard defines a true UI component model that enables other developers to build reusable and redistributable UI components. You should research the available JSF Component Libraries that are available on the market. Some of the libraries are free and some are commercial with a license fee.

**Activity Directions:**

This assignment has 2 parts.

**Part 1: Design and Implement a Java Servlet Application**

Study the Servlet and JSP example code from the required readings for this Topic. Create a new Dynamic Web Project named assignment2a. Make sure you select the JBoss EAP Runtime as the target runtime.

Add a new Java Servlet to the project with the following requirements:

1. Java Servlet is in a package called 'assignment'
2. Java Servlet class name is TestServlet
3. Java Servlet URI at /TestServlet
4. Override the init() and destroy() methods.
5. Use the default generated HTTP Get Method (doGet()) code.
6. Print test messages to the console for the init(), doGet(), and destroy() methods.
7. Deploy the code and execute the application in the internal IDE browser. Take a screenshot.
8. Update the Servlet to read 2 HTTP parameters 'firstname' and 'lastname'. Process the HTTP parameters in the Servlet code and echo your name back to the user.
9. Deploy the code and execute the application specifying your name in the 2 HTTP parameters using the internal IDE browser. Your name should be displayed in the response. Take a screenshot.

Add a new JSP (using New JSP File template) to the project with the following requirements:

1. JSP name of TestResponse.jsp
2. Page Title of Test Response
3. Display the first name and last name in the body of the JSP by reading and displaying the 'firstname' and 'lastname' request attributes passed to the JSP.
4. Update the Servlet to pass the 2 HTTP parameters 'firstname' and 'lastname' as 2 request attributes 'firstname' and 'lastname' and forward the request to the 'TestResponse.jsp'.
5. Deploy the code and execute the application specifying your name in the 2 HTTP parameters using the internal IDE browser. Your name should be displayed in the response. Take a screenshot.

Add a new JSP (using New JSP File template) to the project with the following requirements:

1. JSP name of TestForm.jsp
2. Page Title of Test Form
3. Display a standard HTML form that renders a First Name label with an input form text control with an id of 'firstname', a Last Name label with an input form text control with an id of 'lastname', a Form Submit button, and where the Form uses a POST method to the action of ''TestServlet.
4. Update the Servlet to validate that the two request attributes 'firstname' and 'lastname' are not null or empty and forward the request to the 'TestError.jsp' if the data validation rules fail.
5. Deploy the code and execute the application specifying the TestForm.jsp using the internal IDE browser. Submit the form. The name entered in the form should be displayed in the response. Take a screenshot.

**Part 2: Design and Implement a JSF Application**

Study the JSF example code from the required readings for this Topic. Create a new Dynamic Web Project named assignment2b. Make sure you select the JBoss EAP Runtime as the target runtime. Right click on the project to display the project properties. In the project properties select the Project Facets section. Add the JavaServer Faces facet to the project.

Add a JSF model and JSF page as new components to the project with the following requirements:

*JSF Model as a Managed Bean:*

1. New Java class
2. Java class in a package 'beans'
3. Java class name is User.
4. Java class is marked with the @ManagedBean annotation and @ViewScoped annotation (make sure to import these annotations from the javax.faces.beans package).
5. Java class has a firstName and lastName properties of type String with getter/setters. These 2 properties should be initialized to your name using a default constructor.

*JSF Response Page:*

1. New XHTML Page named TestResponse.xhml, located in WebContent folder, using Blank JSF Page Template.
2. Page title of JSF Response Page.
3. Display the first name and last name properties of the JSF model in the body of the JSF using JSF model binding display syntax (#{model.property}).

*JSF Configuration:*

1. The web.xml Faces Servlet has a URL pattern mapping for \*.jsf and \*.xhtml.

Deploy the code and invoke the TestResponse.jsf page using the internal IDE browser. Your name should be displayed in the response. Take a screenshot.

Add a JSF controller and JSF page as new components to the project with the following requirements:

*JSF Controller as a Managed Bean:*

1. New Java class
2. Java class in a package 'controllers'.
3. Java class name is FormController.
4. Java class is marked with the @ManagedBean annotation and @ViewScoped (make sure to import these annotations from the javax.faces.beans package).
5. Java class has an onSubmit(User user) method that gets the User Managed Bean as a method argument, sets an attribute 'user' with the User Managed Bean in the FacesContext Request Map, and returns as a String the name of the Test Response View (i.e. TestResponse.xhtml).

*JSF Login Page:*

1. New XHTML Page named TestForm.xhml, located in WebContent folder, using Blank JSF Page Template.
2. Page title of JSF Form Page.
3. Display a JSF form that renders a First Name label with a JSF input form text control with a value that is bound to the User Managed Beans 'firstname' property #{user.firstname}, a Last Name label with a JSF input form text control that is bound to the User Managed Beans 'lastname' property #{user.lastname}, and a JSF command button whose action invokes the FormController Controller onSubmit(User user) method using the property #{formController.onSubmit(user)} and with a value attribute set to "Submit".

Deploy the code and execute the application specifying the TestForm.jsf using the internal IDE browser. Submit using the Submit button. The name entered in the form should be displayed in the response. Take a screenshot.

*JSF Flash Scope:*

1. Update the TestForm.xhml to add a new JSF command button whose action invokes the FormController Controller onFlash(User user) method using the property #{formController.onFlash(user)} and with a value attribute set to "Flash".
2. Add a new method to the FormController Controller with an onFlash(User user) method that gets the User Managed Bean as a method argument, sets an attribute 'user' with the User Managed Bean in the FacesContext Request Map, and return as a String the name of the Test Response View with the redirect option (i.e. TestResponse.xhtml?faces-redirect=true).
3. Deploy and run the application. Submit the form using the Flash button in a browser with Developer Tools enabled and validate that the Controller is generating a validate HTTP 302 redirect request. Take a screenshot of the HTTP 302 redirect from the browsers Developer Tools. Also validate the TestResponse.xhtml does not maintain the User Model between the Controller and View Request.
4. Update the onFlash() method to set an attribute 'user' with the User Managed Bean FacesContext Flash Map (i.e. change the call to getRequestMap() with getFlash()).
5. Copy the View TestResponse.xhtml to a new View TestResponse2.xhtml and update the onFlash() to return to the new View. Update the EL in the TestResponse2.xhtml to prefix each EL expression with a 'flash' (i.e. #{flash.user.firstName} #{flash.user.lastName}).

Deploy the code and execute the application specifying the TestForm.jsf using the internal IDE browser. Submit the form using the Flash button. The name entered in the form should be displayed in the response. Take a screenshot. Validate the TestResponse.xhtml does maintain the User Model between the Controller and View Request.

**Deliverables:**

A “Project Report,” containing the following:

1. A coversheet with the name of class, assignment, date, and your name
2. The GitHub URL
3. Brief theory of operation explaining the approach to design and implementation, including a detailed list of all classes, methods, variables, and pages
4. Project report with all screenshots showing execution of applications

**Submissions:**

Submit code to GitHub.

Submit all code as a zip file to LoudCloud.

Submit Project Report to LoudCloud.