**Short Assignment/In-Class Activity #6 – Security in Enterprise Java**

**Short Assignment 6a: Add security to an Enterprise Java application.**

Objective:

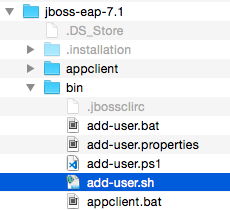
1. Design and implement standards based Login Form, Login Error, and configuration to secure an existing Enterprise Java application.

Activity Directions:

Study the Security example code from the reading for this Topic. This activity will add code to the previous activity.

Add and update the following components in the project:

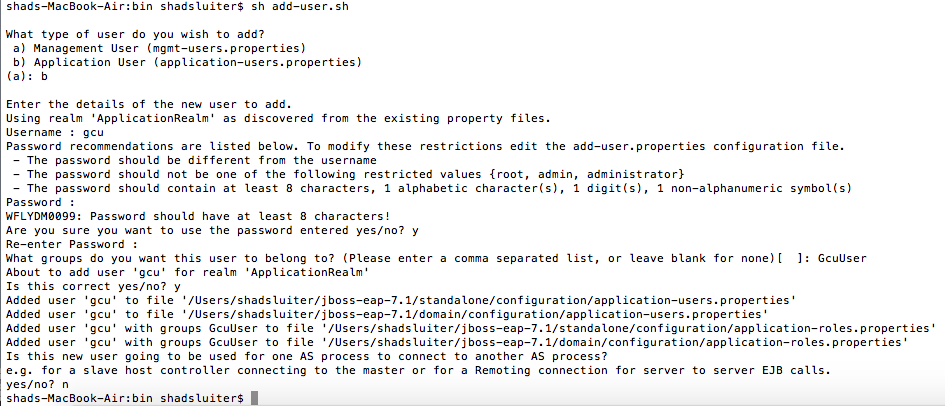
* JBoss Configuration:
  + Navigate to the *bin* directory where JBoss EAP is installed. For Macintosh, the folder is //users/username/jboss-eap-7.1



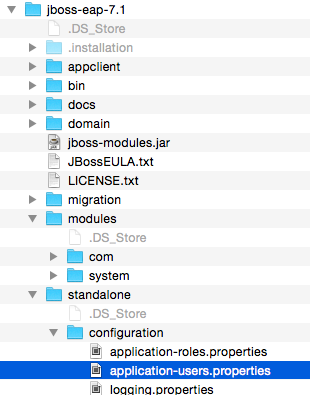
* + Locate the *add-user* script.
  + Run the add-user script (for Mac or Linux users you may need to run chmod 777 over the file to make it executable).



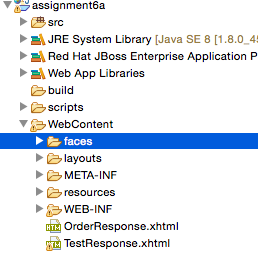
* + When prompted select the Application User type of user, enter a desired username (gcu), enter a desired password (lopes), enter GcuUser for the group, and answer no for connection to another process.



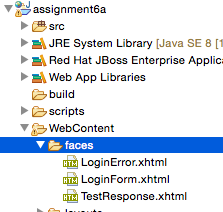
* + This will create a username and password you can now log into the assignment application. NOTE: the users are created in a file application-users.properties located in the JBoss EAP standalone/configuration directory.



* JSF Login Form v2:
  + Create a folder named *faces* under the WebContent folder.



* + Copy and paste the TestForm.xhtml to LoginForm.xhtml in the *faces* folder.



* + Change the content section to the following. Note, DO NOT use the JSF form tags but use the standard HTML form tags. Inspect the important **action** and **id** attributes in the form as these are standard in an enterprise application for FORM based authentication.

<div align="center">

<form  method="post" action="j\_security\_check">

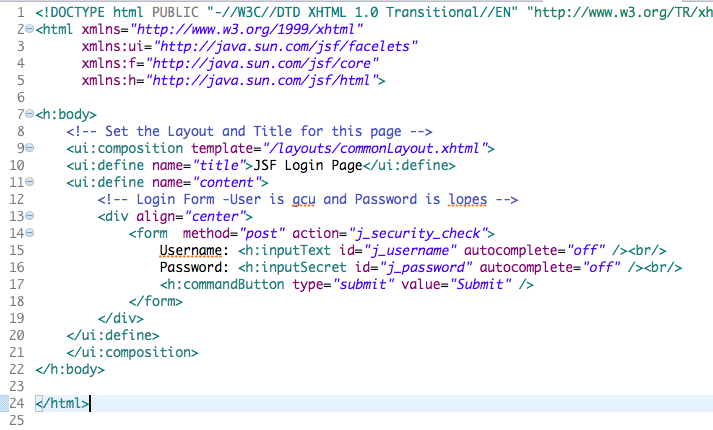
Username: <h:inputText id="j\_username" autocomplete="off" /><br/>

Password: <h:inputSecret id="j\_password" autocomplete="off" /><br/>

<h:commandButton type="submit" value="Submit" />

</form>

</div>



* + Create a new JSF page LoginError.xhtml in the *faces* folder. Insert the following into the context section of the page. Inspect the code.

<div align="center">

<p>Please enter a user name or password that is authorized to access this

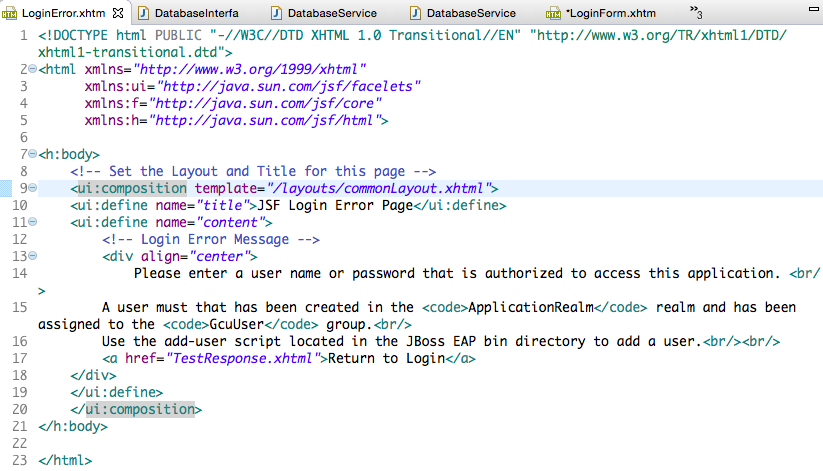
        application. For this application, this means a user that has been

        created in the <code>file</code> realm and has been assigned to the

    <em>group</em> of <code>GcuUser</code>.</p>

    <p><a href="TestResponse.xhtml">Return to Login</a></p>

</div>



* Security Configuration
  + Open web.xml.
  + Add the following security configuration below the JAX-WS configuration section.
  + Inspect the configuration. Note, that the FORM authentication is setup using the ApplicationRealm which is the available security realm built into the JBoss server. Also note the URI’s that are protected and the URI’s that are not projected.

  <!-- Security Configuration -->

  <security-constraint>

        <display-name>GCU Test Application Security Constraints</display-name>

        <web-resource-collection>

            <web-resource-name>Protected</web-resource-name>

    <description>Protect all pages in Web Root</description>

            <url-pattern>/\*</url-pattern>

            <http-method>GET</http-method>

            <http-method>POST</http-method>

        </web-resource-collection>

        <auth-constraint>

            <description/>

            <role-name>GcuUser</role-name>

        </auth-constraint>

   </security-constraint>

   <security-constraint>

  <web-resource-collection>

    <web-resource-name>Public</web-resource-name>

    <description>Do not Protect Web Services and JSF Resources</description>

    <url-pattern>/rest/\*</url-pattern>

    <url-pattern>/OrdersSoapService/\*</url-pattern>

   <url-pattern>/javax.faces.resource/\*</url-pattern>

   </web-resource-collection>

</security-constraint>

    <login-config>

        <auth-method>FORM</auth-method>

        <realm-name>ApplicationRealm</realm-name>

        <form-login-config>

            <form-login-page>/faces/LoginForm.xhtml</form-login-page>

            <form-error-page>/faces/LoginError.xhtml</form-error-page>

        </form-login-config>

    </login-config>

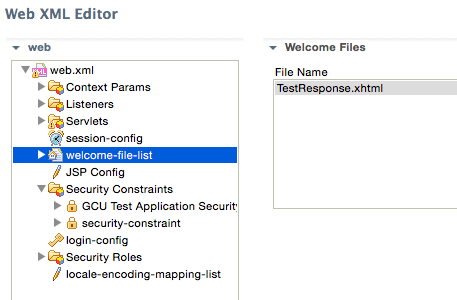
   <security-role>

        <description/>

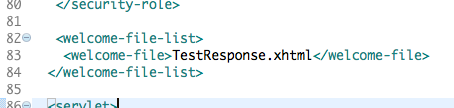
        <role-name>GcuUser</role-name>

   </security-role>

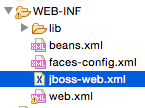
* + Change the welcome-file to TestResponse.xhtml (to force the root to go to a secure page and display the Login Page).
  + Tree view mode…



Source XML Mode…



* Security Configuration (in jboss-web.xml):
  + Create a new file named *jboss-web.xml* in the WEB-INF directory.



* + Add the following configuration to the file to have the application run with in the JBoss *other* security domain which is built into the JBoss server:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE jboss-web>

<jboss-web>

  <security-domain>other</security-domain>

</jboss-web>

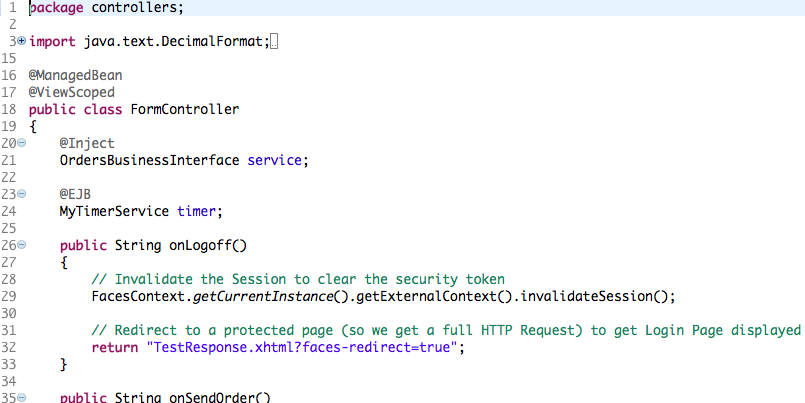
* Update the Form Controller:
  + Remove the *TestForm.xhtml* page from the project (not needed any longer).
  + Remove the *onSubmit()* method from the Controller (not needed any longer).
  + Add a new public method *onLogoff()* that returns a string. Add the following code to the implementation. Inspect the code.

// Invalidate the Session to clear the security token

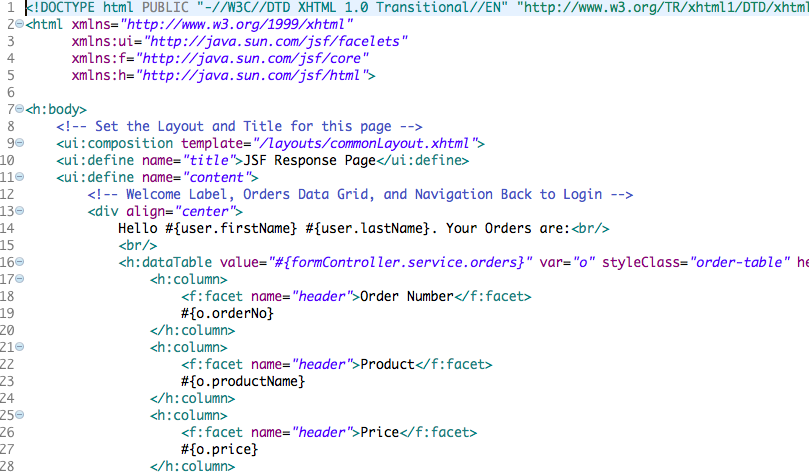
FacesContext.getCurrentInstance().getExternalContext().invalidateSession();

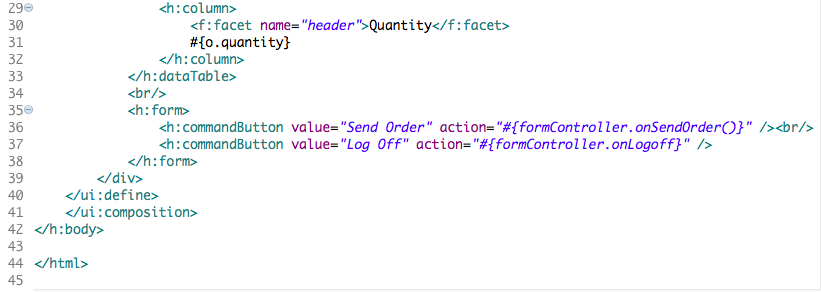
// Redirect to a protected page (so we get a full HTTP Request) to get Login Page

return "TestResponse.xhtml?faces-redirect=true";



* Update the Test Response Page:
  + Update the Log Out button label to *Log Out* and set the action for the command button to the Form Controller *onLogoff()* method.
  + Change the ‘Hello’ message so it only renders the *User* Bean FirstName property (and make the same change to the *OrderResponse* page.





* Update the User Managed Bean:
  + Add a new public method named *init()* that returns void. Mark this method with a @PostConstruct annotation.



* + Add the following code to initialize the User bean from the security principle. Inspect the code.

// Get the logged in Principle

Principal principle= FacesContext.getCurrentInstance().getExternalContext().getUserPrincipal();

if(principle == null)

{

setFirstName("Unknown");

setLastName("");

}

else

{

setFirstName(principle.getName());

setLastName("");

}

Deploy the code and execute the application specifying the TestResponse.jsf (or go to the root) using the internal IDE browser. The Login page should be displayed. Login with valid credentials (that you created). Take a screenshot. When first accessing the Test Response page (remember it is in a protected URI) the Login page should be displayed. You should be able to login with the credentials you created to access the Test Response page. Login with invalid credentials. Take a screen shot. The Login Error page should be displayed. Login with valid credentials and click the Log out button. The Login page should be displayed.

Developer Notes & Best Practices:

* If you are securing the pages in your web application make sure to configure this properly so that any URI’s for your REST or SOAP based services do not get inadvertently caught in your security framework for your web application.
* Unless warranted always use a standards based and security framework implementation provided by your Enterprise Java Application Server. DO NOT implement your own security framework.
* Be aware of security policies when working with Personal Identifying Information (PII) data or Payment Card Industry (PCI) data for data at rest or in flight.
* Unless warranted you should always deploy your application to utilize secure protocols such as HTTPS.

Deliverables and Submission:

1. A Project report containing the following:
   * Cover sheet with name of class, assignment, date, and your name.
   * BitBucket URL.
   * Brief theory of operation explaining the approach to design and implementation, including a detailed list of all classes, methods, variables, and pages.
   * Project report with all screenshots showing execution of applications.
2. Upload code to BitBucket
3. Upload code as a zip file to LoudCloud
4. Upload Project report to LoudCloud.

This assignment utilizes a rubric. Please review the rubric prior to beginning the assignment to become familiar with the expectations for successful completion.