Seam Tools Tutorial

Version: 4.2.0.Final-SNAPSHOT

Create a Seam Application

In this section you will learn how to create a Seam project in JBoss Developer Studio, how to start the server and what structure your project has after it is created.

1.1. Start Development Database

Before opening the JBoss Developer studio you need to download and start the *Workshop Database* [http://docs.jboss.org/tools/resources/GSG_database.zip] .

To start the database just run `./runDBServer.sh` or `runDBServer.bat` from the database directory.

The end result should be a console window that looks like:

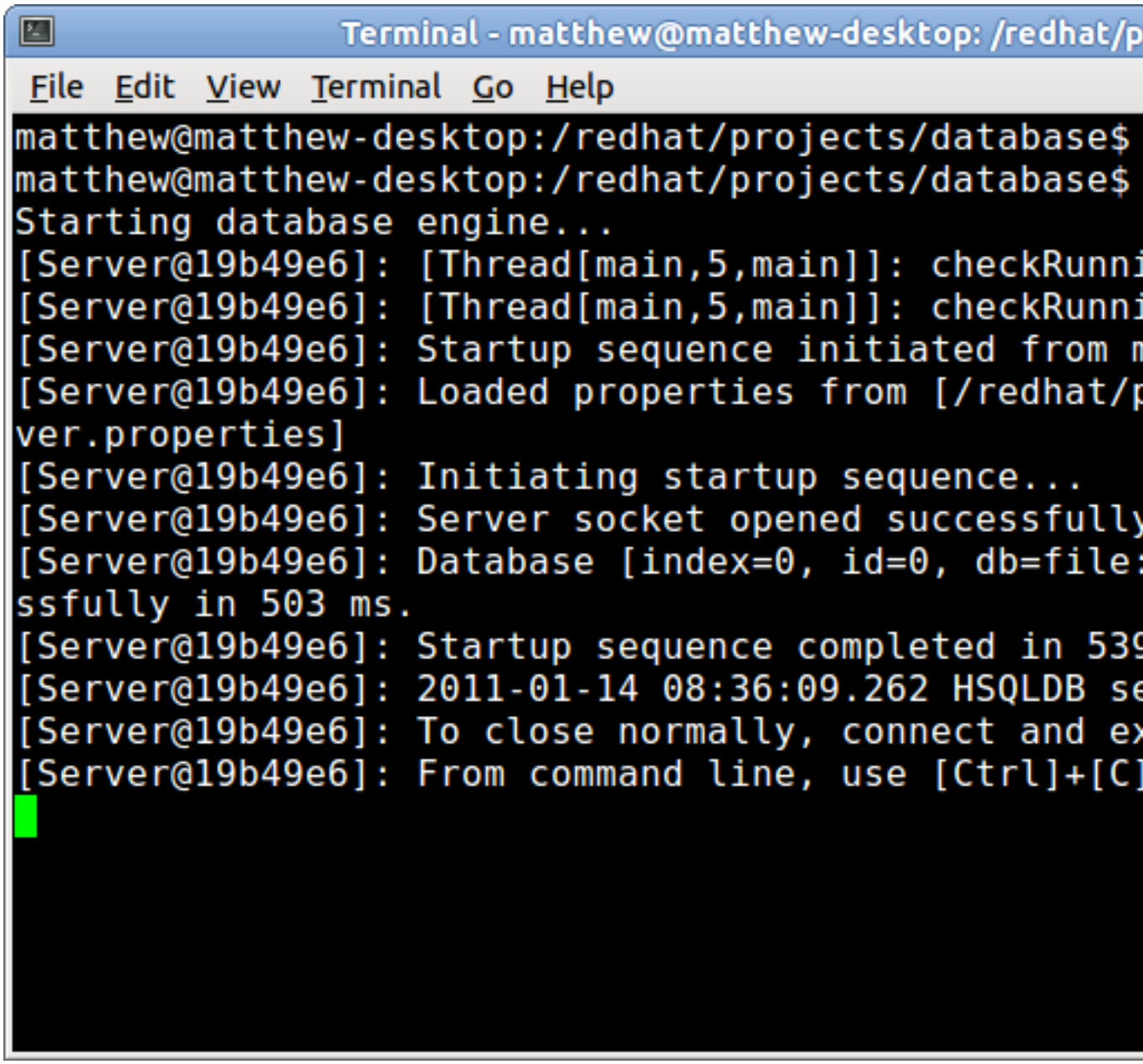

Figure 1.1. Starting the Database

Tip

You may need to set the `runDBServer.sh` executable flag with the following command:

```
chmod +x runDBServer.sh
```

1.2. Create and deploy Seam Web Project

Minimize the terminal window and run JBoss Developer Studio from Applications Menu or from the desktop icon.

First you will see the Workspace Launcher. Change the default workspace location if it's needed. Click the **OK** button.

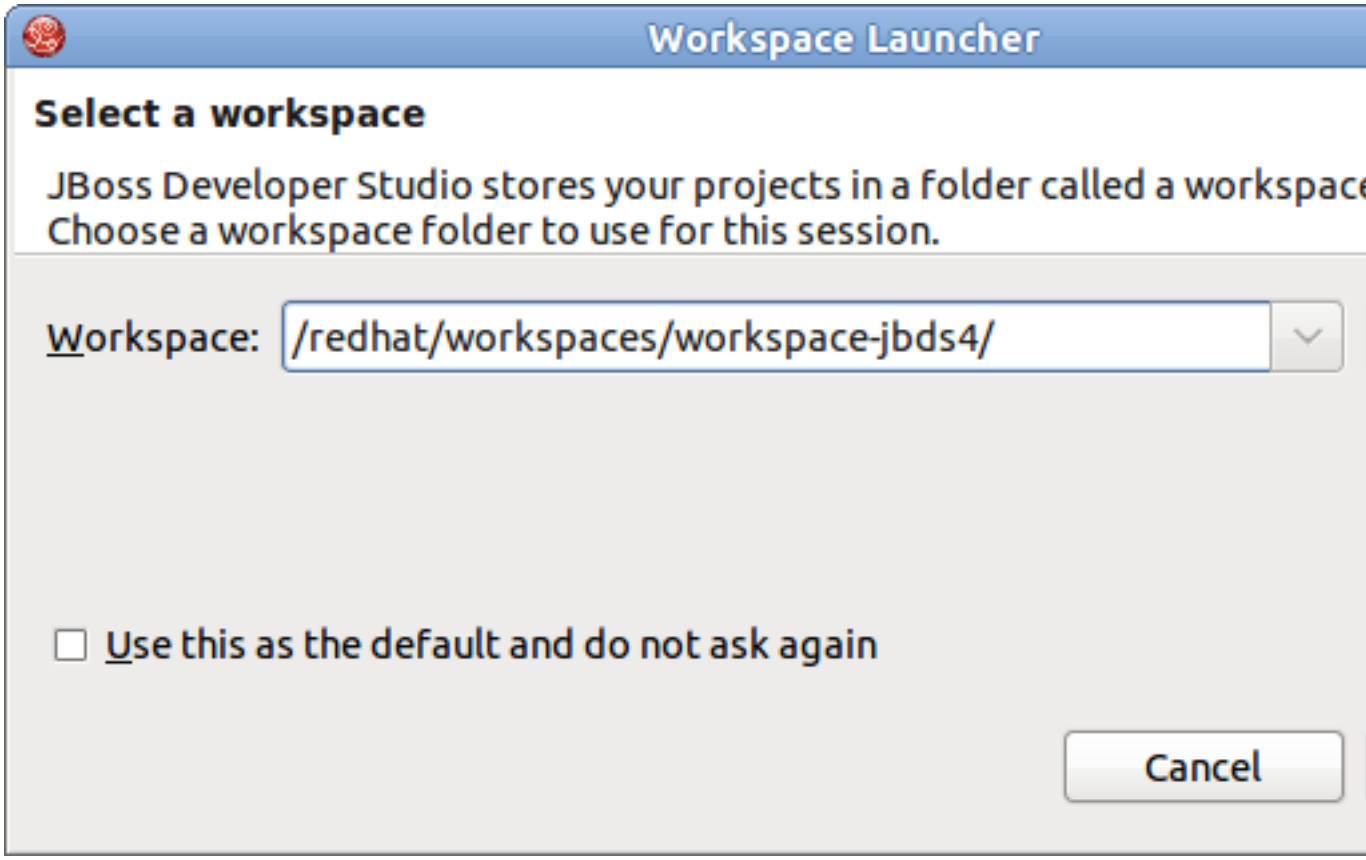

Figure 1.2. Workspace Launcher Dialog

After startup, you see the welcome page. Select **Create New...** icon and then press on **Create Seam Project** link.

The New Seam Project wizard is started. You need to enter a name (e.g., "workshop") and a location for your new project. The wizard has an option for selecting the actual Server (and not just WTP runtime) that will be used for the project. This allows the wizard to correctly identify where the destination folder for the required datasource and driver libraries.

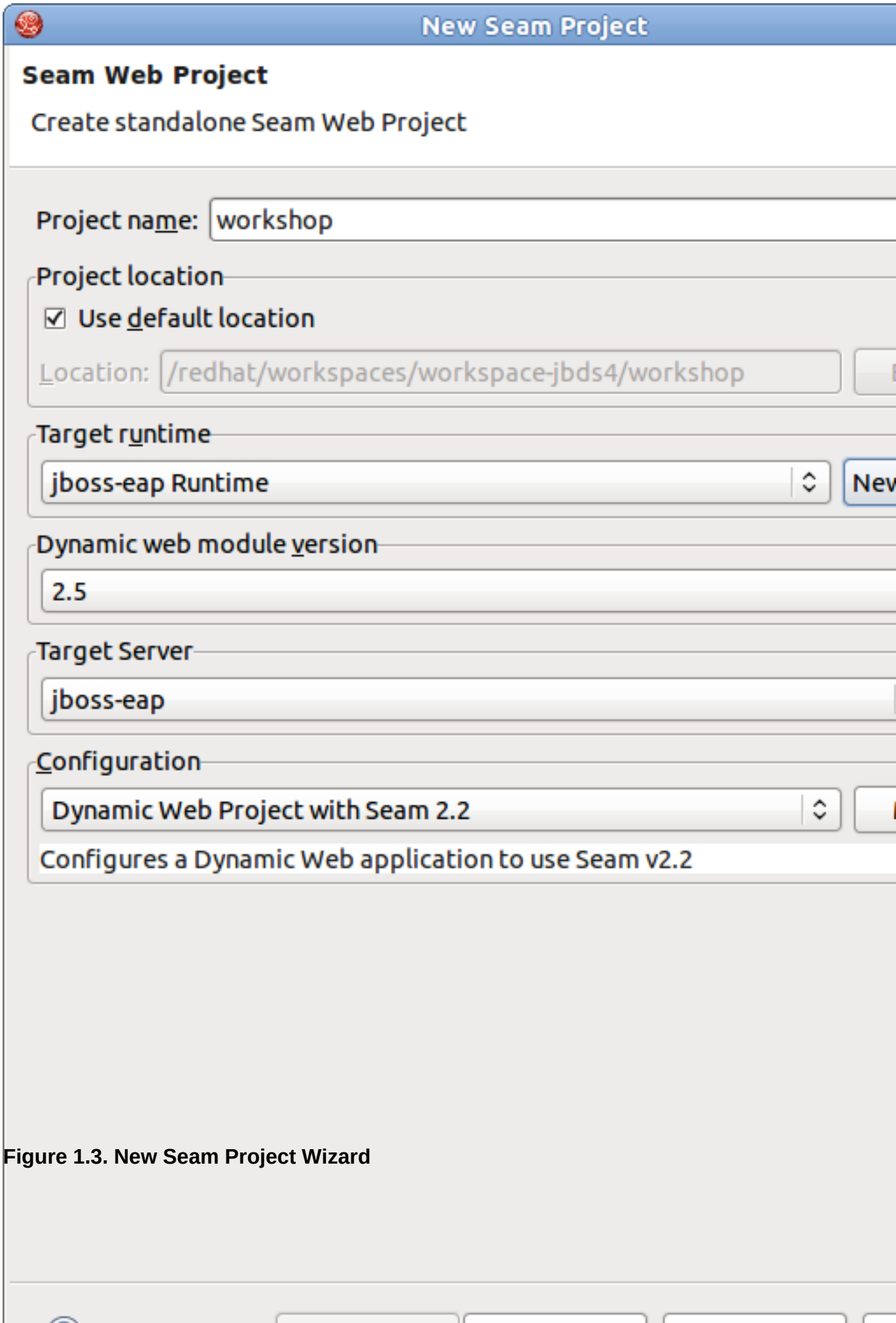

Figure 1.3. New Seam Project Wizard

All settings are already specified here, you can just modify the Configuration. Click on the **Modify...** button to configure your custom facet options:

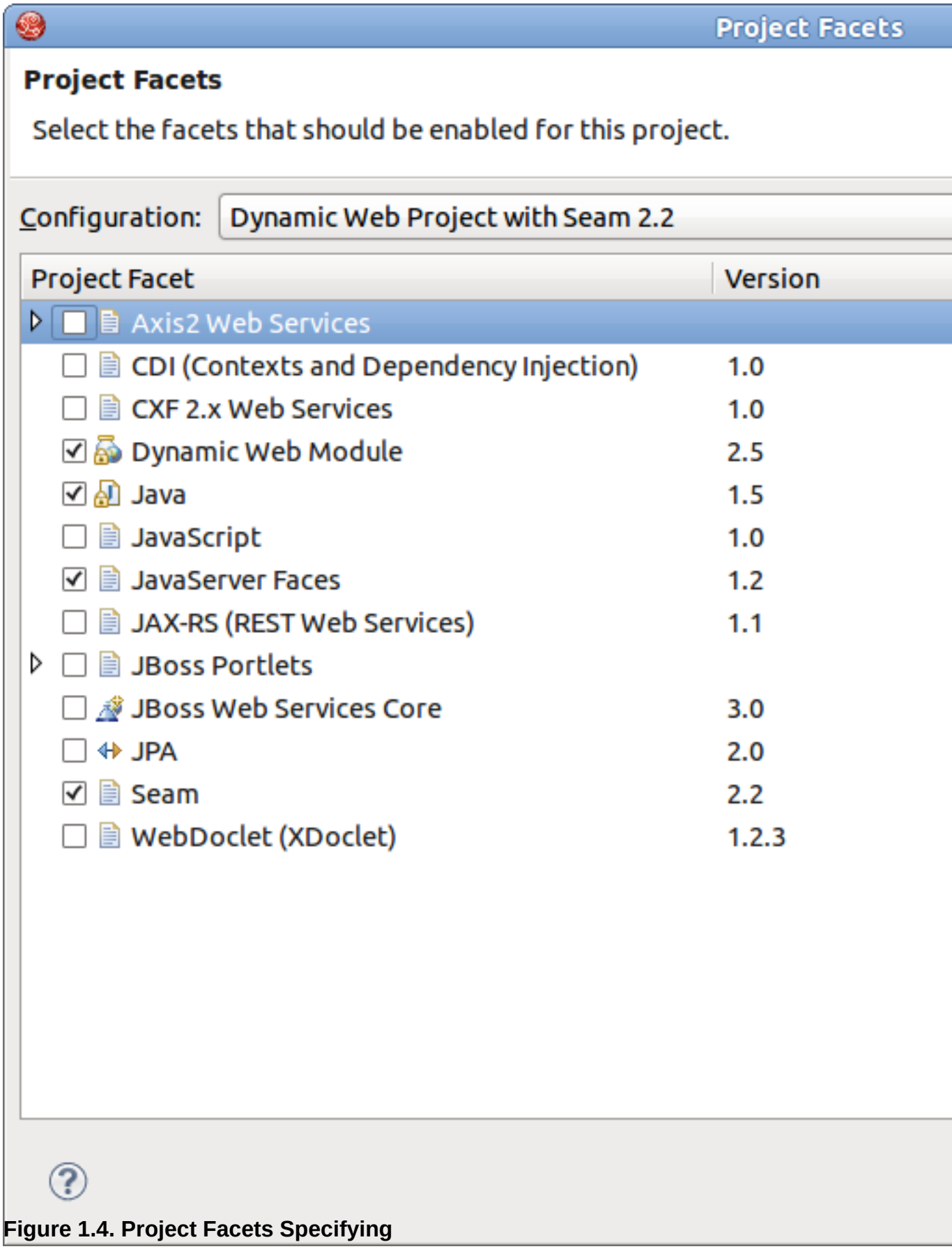

Figure 1.4. Project Facets Specifying

On the whole the dialog allows to select the "features" you want to use in your project. JBoss Developer Studio will then setup the appropriate tooling for your project. Since JBoss Seam integrates all popular Java EE frameworks, you can select any combination of technologies from the list. Here, for the default configuration, Dynamic Web Module, Java, JavaServer Faces (JSF), and Seam Facet are already selected for a typical database-driven web application. The default project facets should suffice.

In the Project Facets form you can also bring up server runtimes panel by clicking Runtimes tab on the right corner. This panel shows available server runtimes.

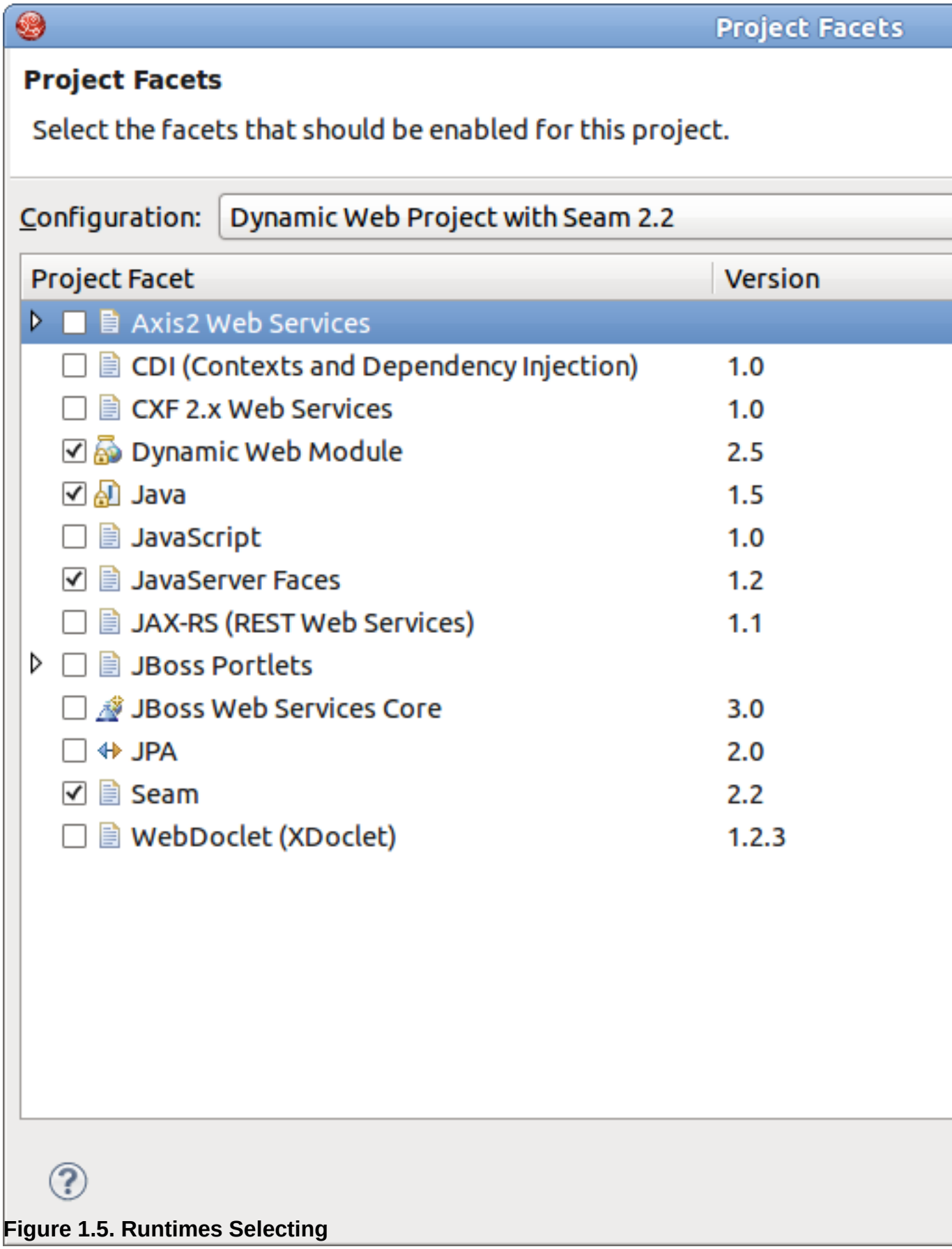

Figure 1.5. Runtimes Selecting

Click the **OK** and then the **Next** button to proceed to the next step.

A dynamic web application contains both web pages and Java code. The next wizard will ask you where you want to store Java files.

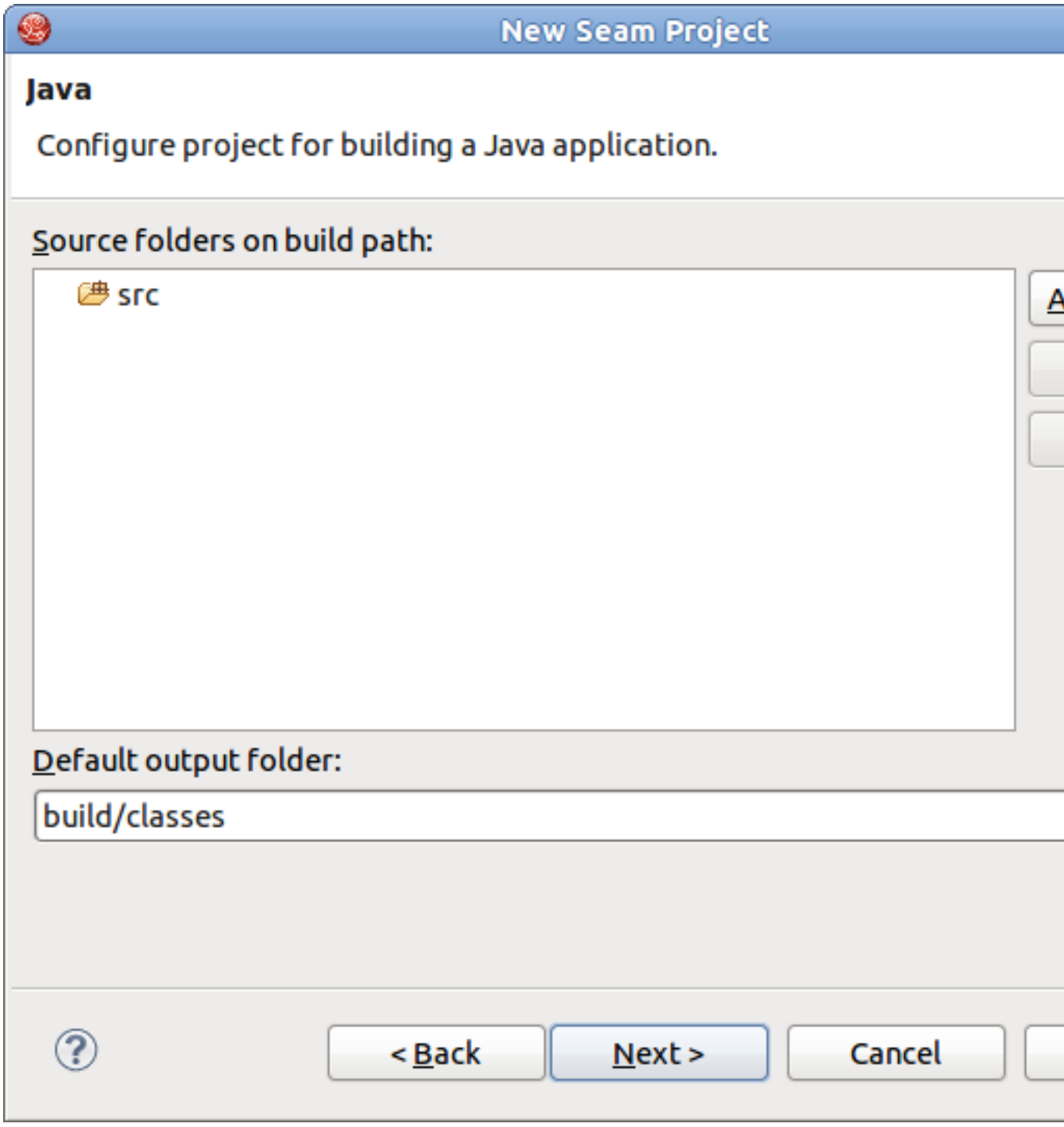

Figure 1.6. Java Build Path

Following page provides you Web Module Settings .You can just leave the default values or choose another folder.

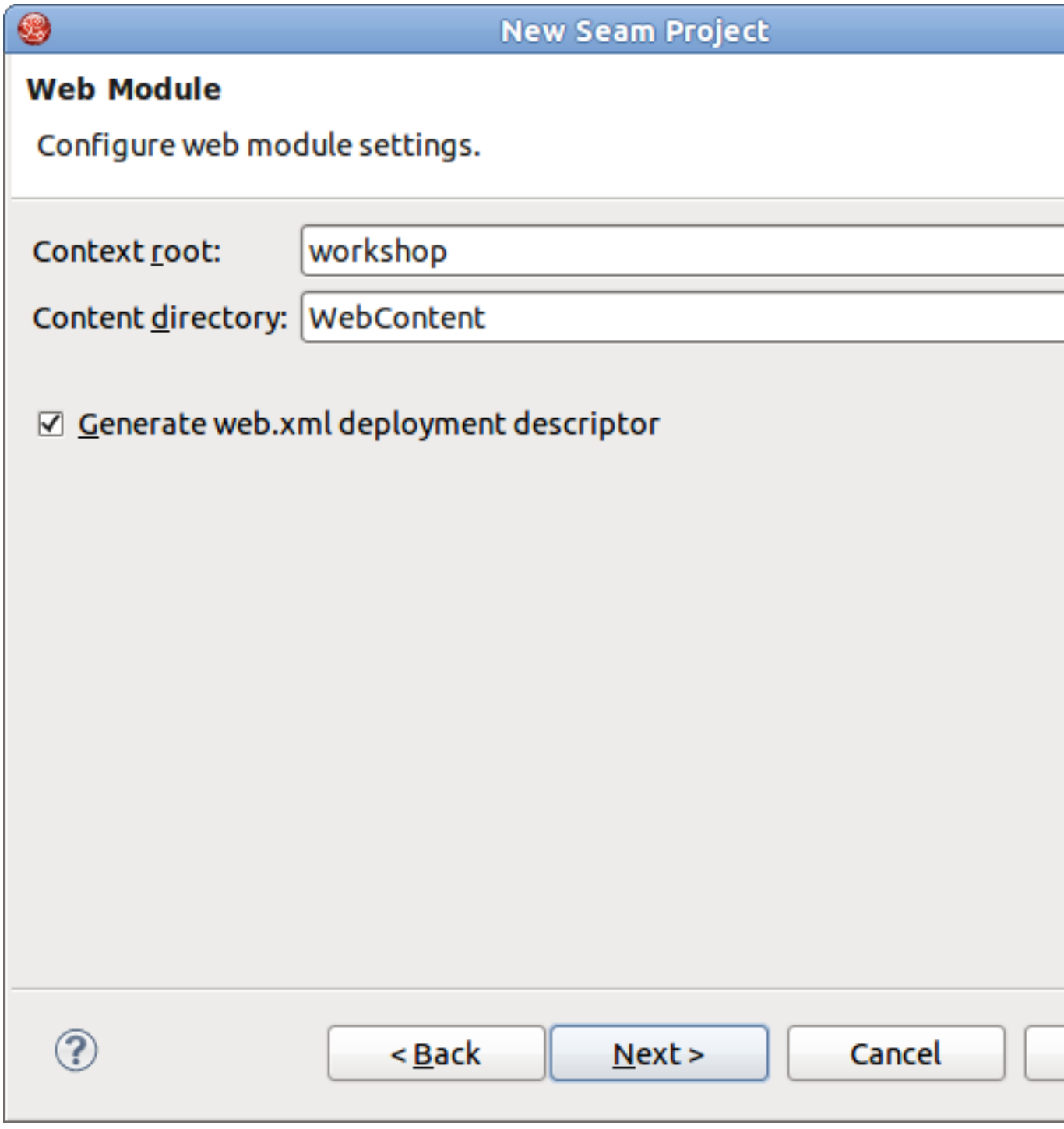

Figure 1.7. Web Module Settings

On the next form, you will be able to select where those library JARs come from. The easiest is just to select the JARs provided by the JBoss AS runtime associated with this project. That is why it is important to chose the right JBoss AS 4.2 runtime in the project setup window.

- Select *Library Provided by Target Runtime* as Type of JSF Implementation Library. We will use the JSF implementation that comes with JBoss server.

- Click the **Next** button

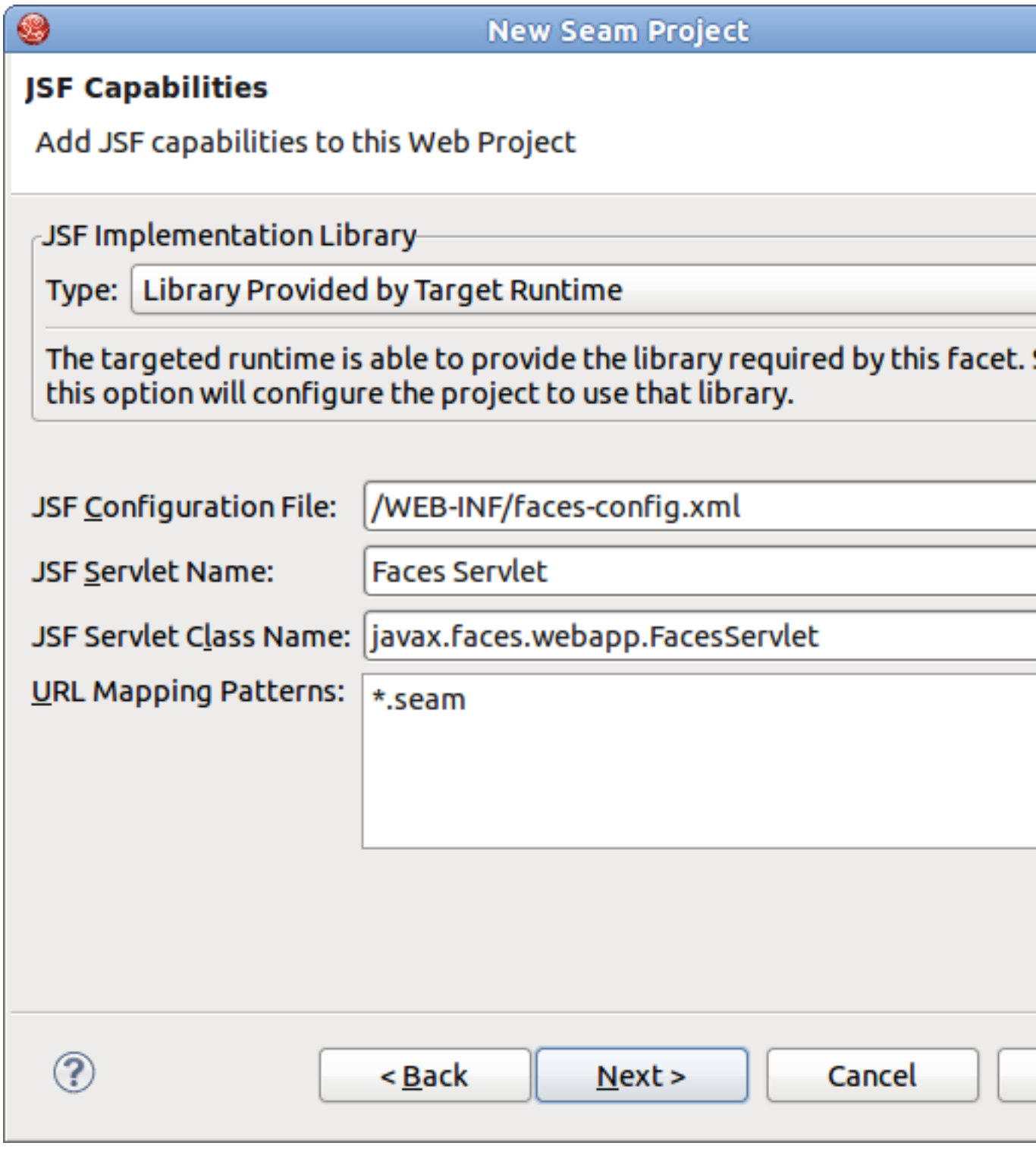

Figure 1.8. JSF Capabilities Adding

Next wizard step needs more settings that previous. Let's start with General section.

Leave the default Seam runtime and check a WAR deployment.

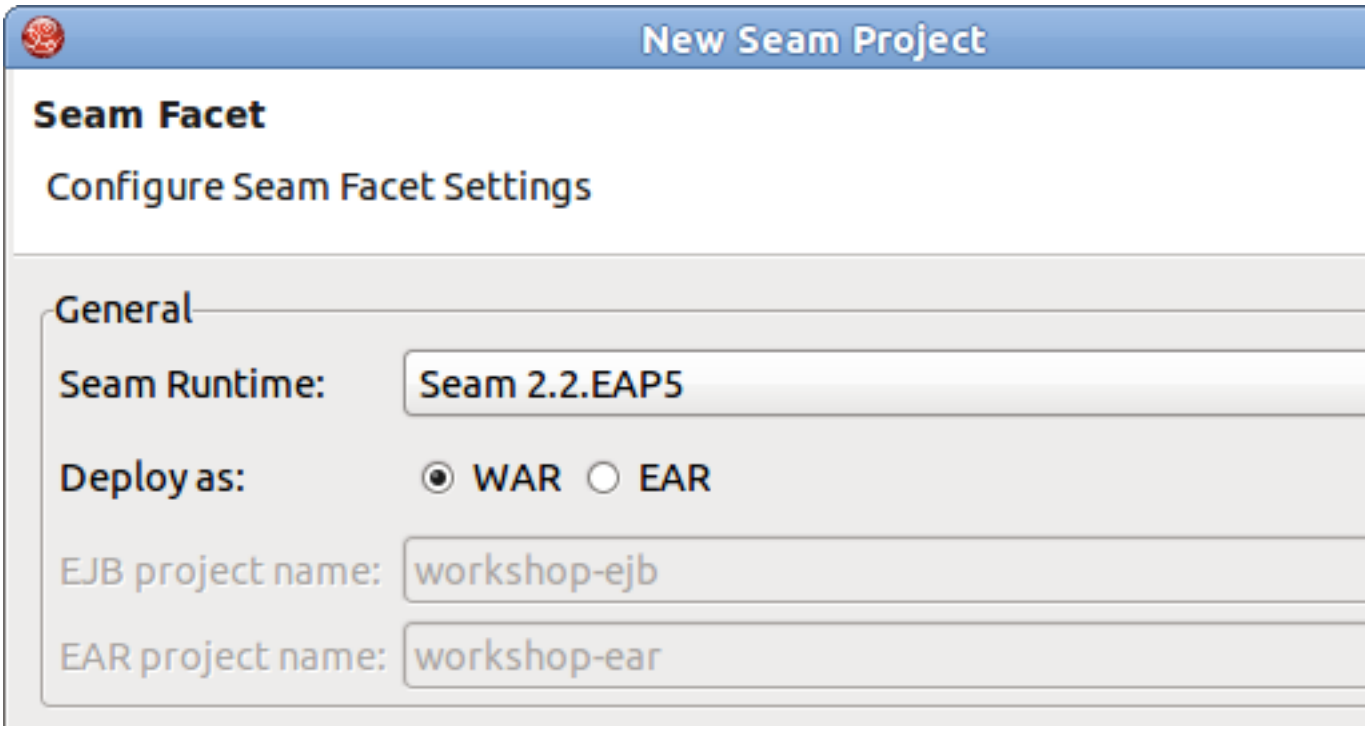

Figure 1.9. Seam Facet Setting

The Database section is a little tricky. The Connection Profile needs to be edited so that the new project works properly with the external HSQLDB server. By default the project wizard tries to use the JBoss embedded HSQLDB, but the tutorial uses an external database to replicate a more real world development scenario. Click on the **Edit** button to modify the Connection Profile.

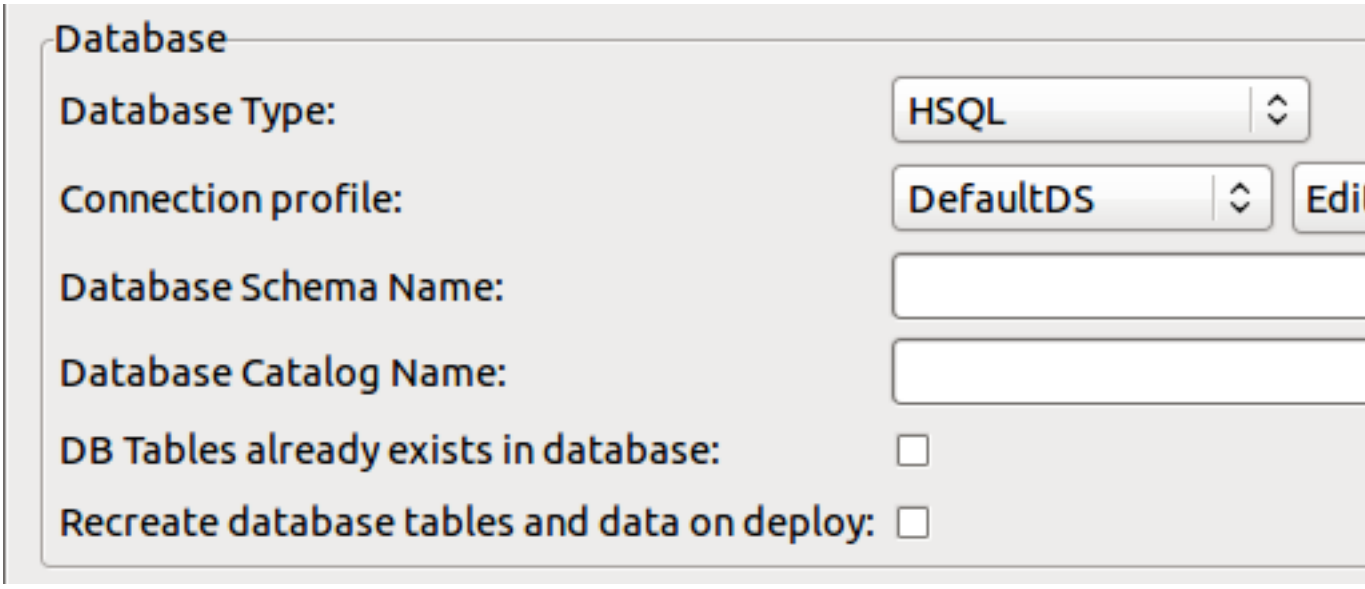

Figure 1.10. DataBase Setting

Select HSQLDB Profile Properties. Make sure the Database location is set to *hsql://localhost:1701*

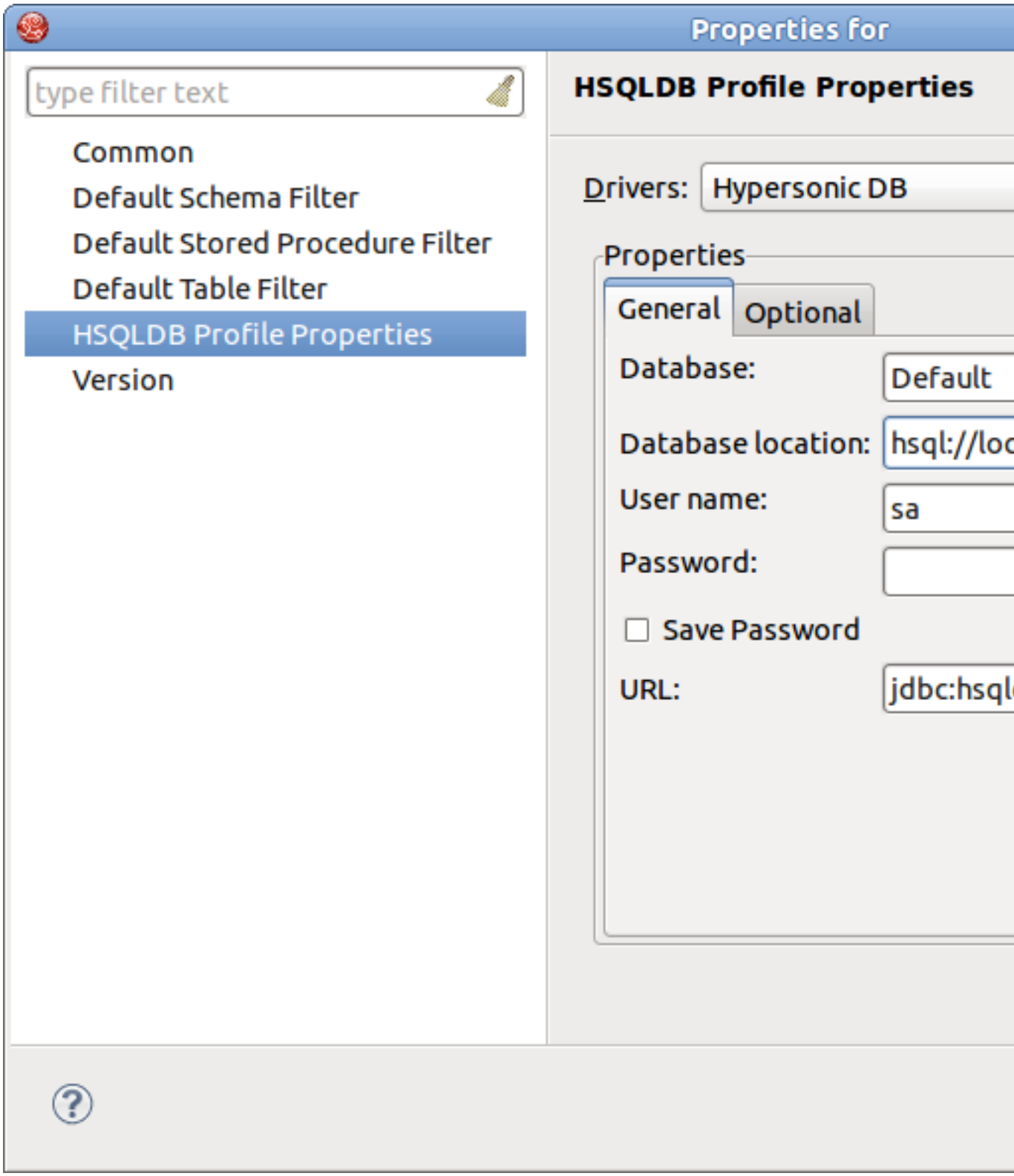

Figure 1.11. JDBC Connection Properties

Click the **Test Connection** button. At this point it probably won't work. This happens if the HSQL JDBC driver is not exactly the same. This can be solved by modifying the HSQLDB database driver settings. To modify the settings, click the **Edit Driver Definition Driver** button.

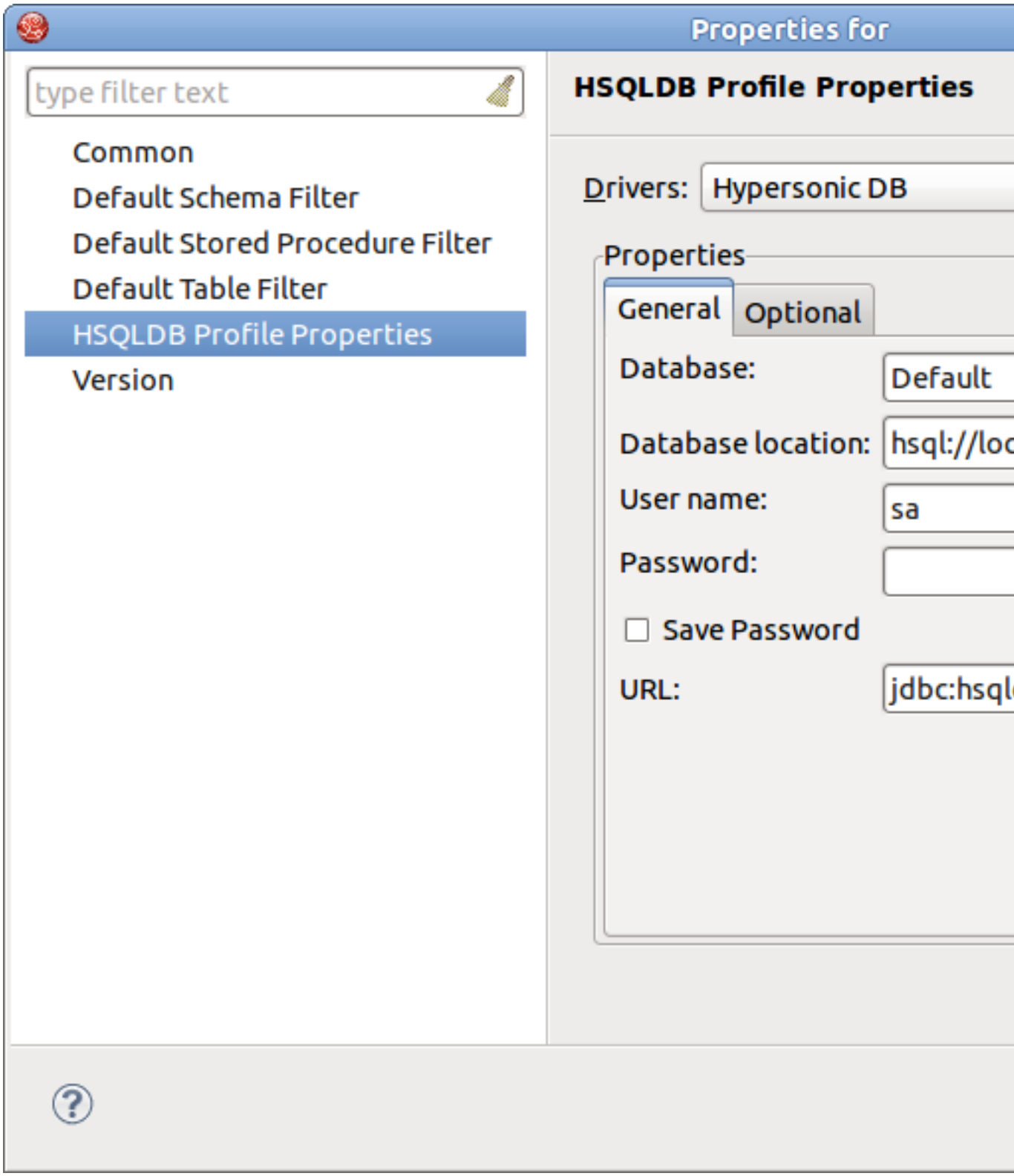

Figure 1.12. Driver Details

The proper Driver JAR File should be listed under Jar List. Select the `hsqldb.jar` file found in the `jbdevstudio/jboss-eap/jboss-as/common/lib/` directory and click the **OK** button.

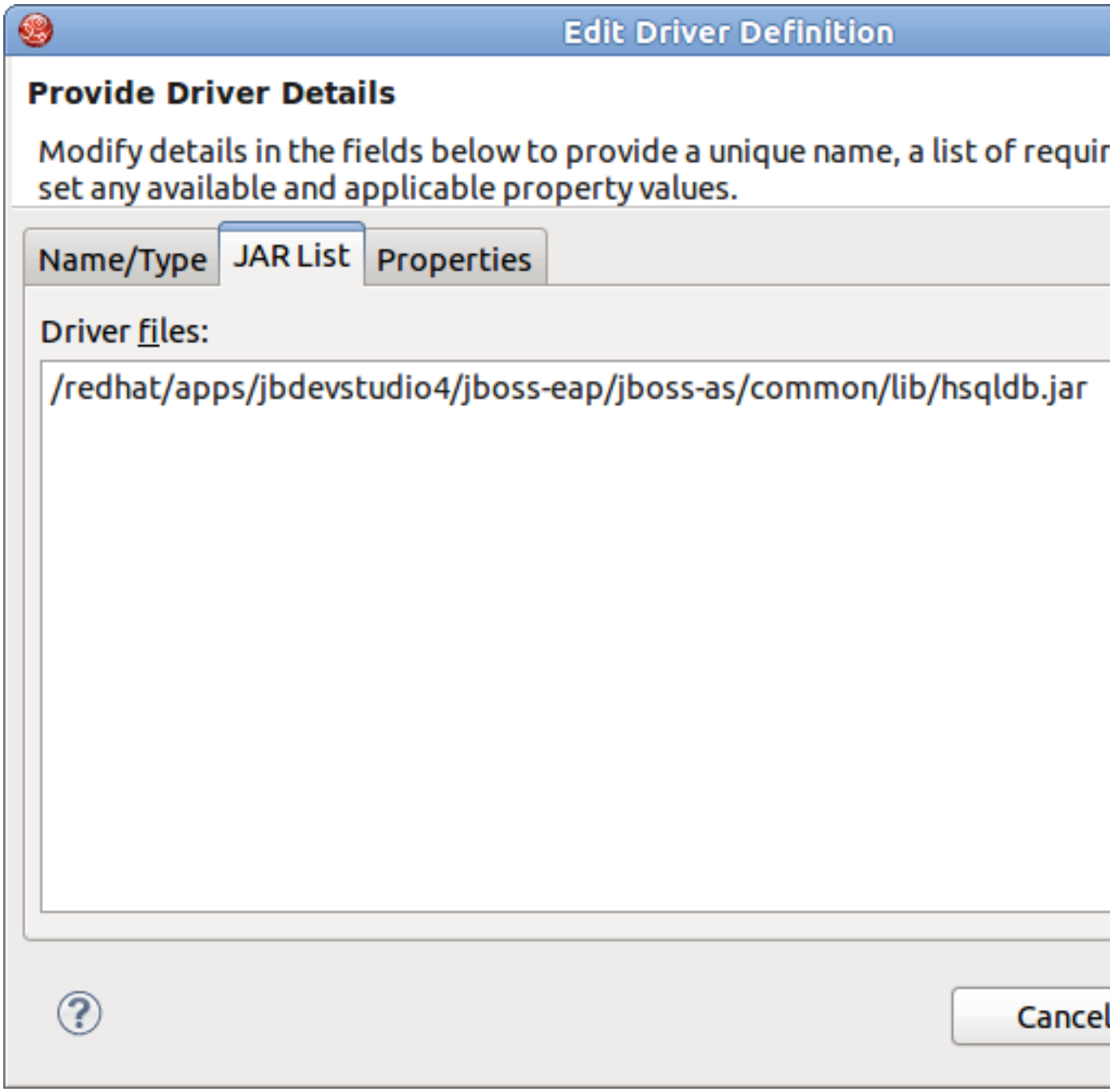

Figure 1.13. Driver Details

Now, the **Test Connection** should succeed. After testing the connection, click the **OK** button.

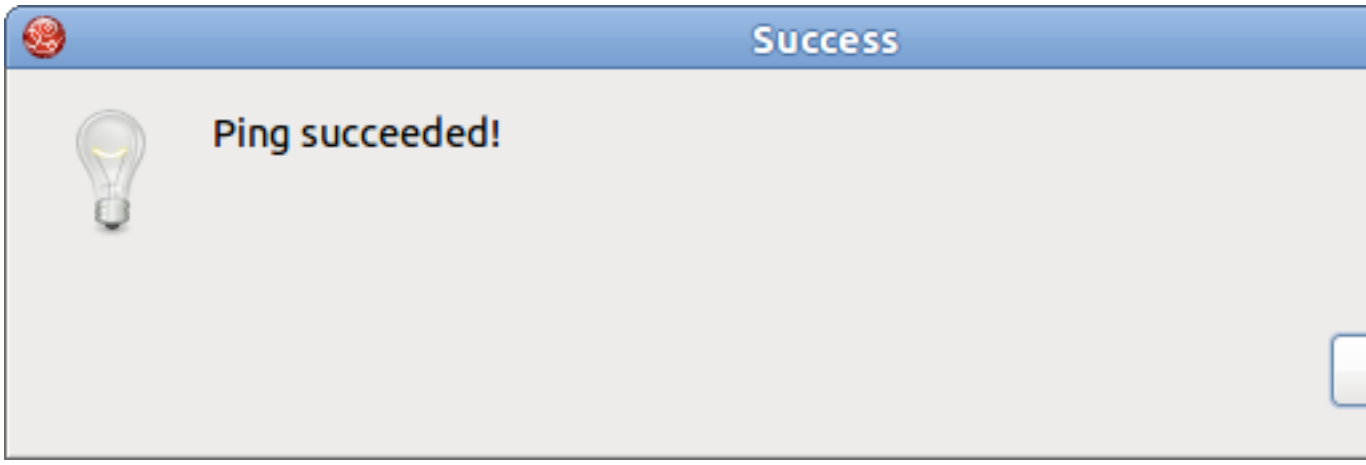

Figure 1.14. Connection Testing

You can leave the Code Generation section as is. It refers to Java packages in which the generated code will be placed.

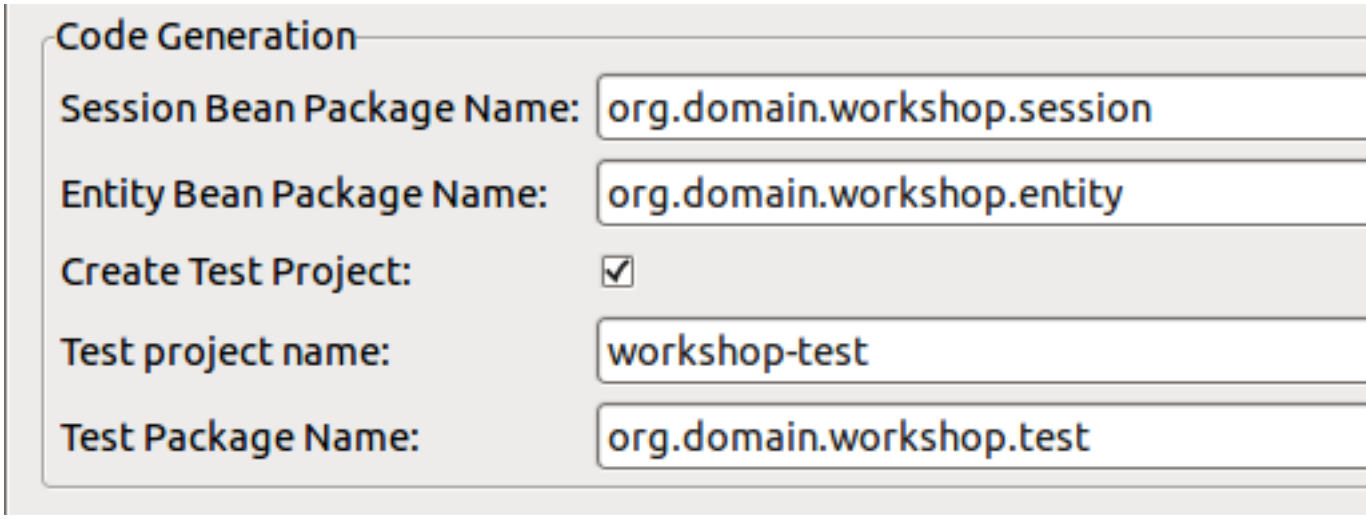

Figure 1.15. Code Generation Setting

Tip:

If you want to name your web project "MyProject-war" note that the Test project name should not be "MyProject-war-test", it should be "MyProject-test".

Click on **Finish** button. Now, there should be a new Seam project called “workshop” listed in the Package Explorer view.

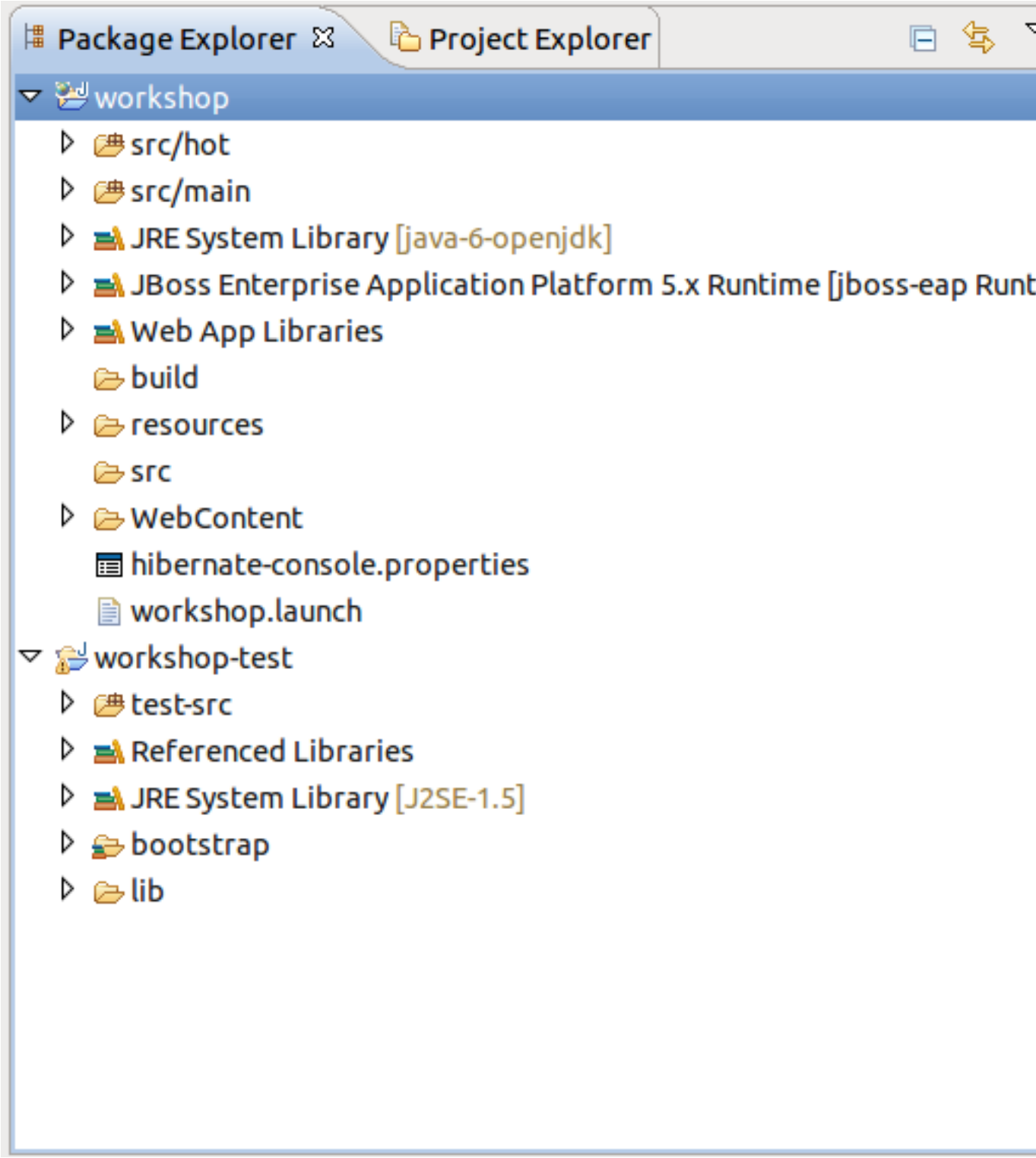

Figure 1.16. "workshop" Project in the Package Explorer

1.3. Start JBoss Application Server

Start the server by clicking on the Start the server icon () in the Servers view.

Then run the project by selecting the project then selecting **Run As...** → **Run on Server**.

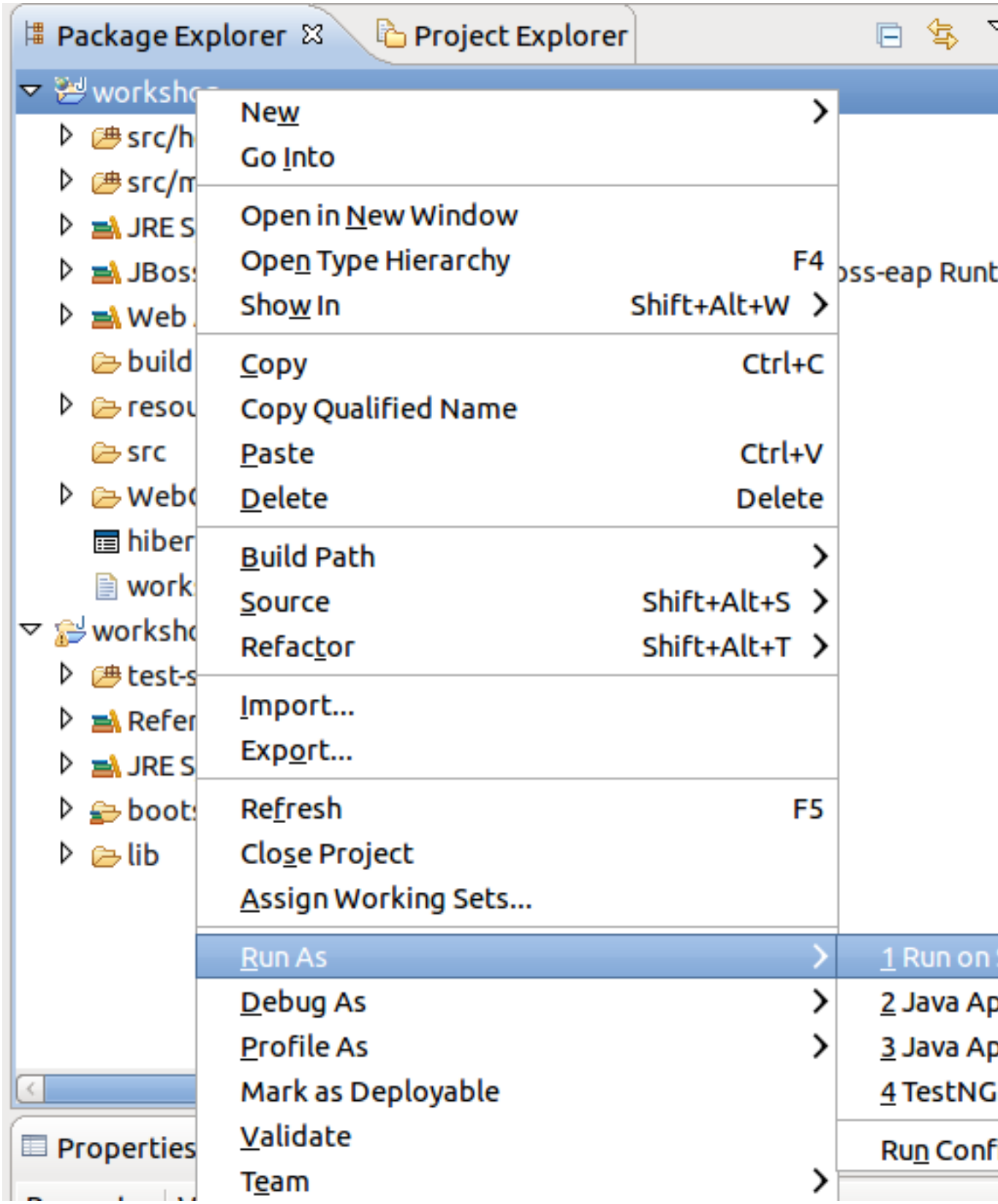

Figure 1.17. "workshop" Run As

Select the server you want to run the project on, and click the **Finish** button.

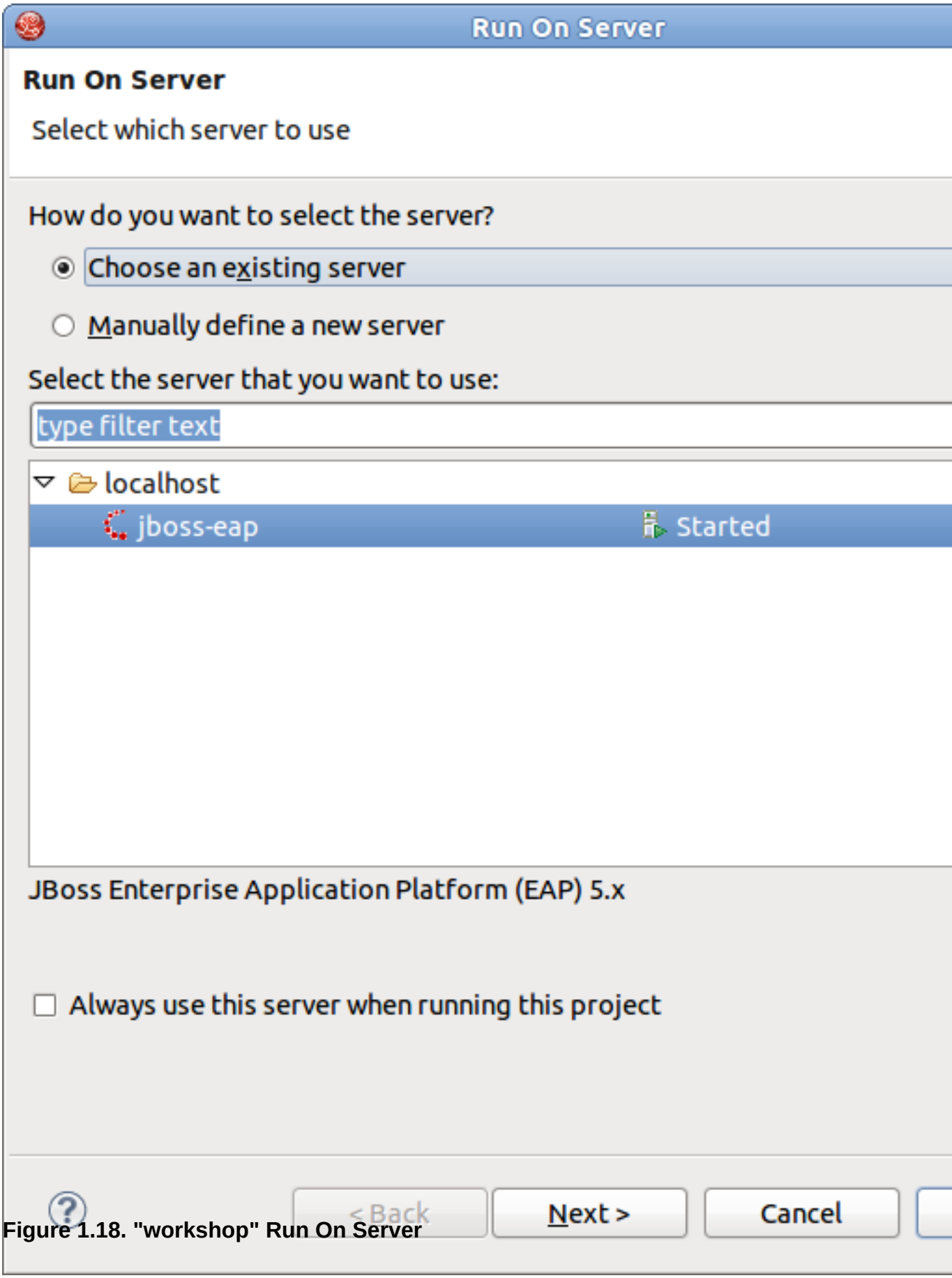

Figure 1.18. "workshop" Run On Server

Note:

If the project does not show up, then you can use a normal browser and use *http://localhost:8080/workshop/home.seam* as the URL.

Your project looks like this:

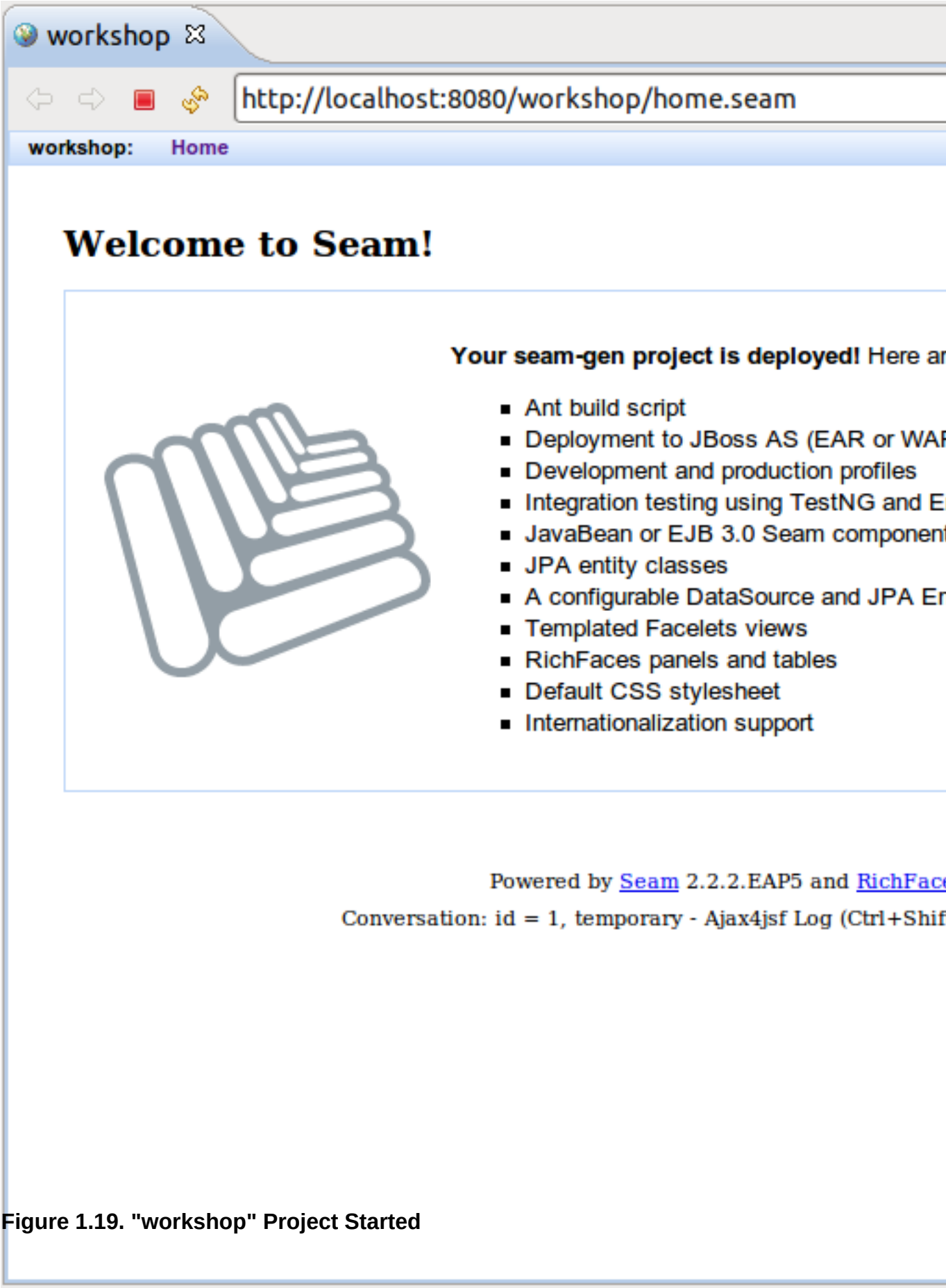

Figure 1.19. "workshop" Project Started

1.4. Workshop Project Code Overview

Now let's examine the project and its structure. Go back to the Package Explorer view in JBoss Developer Studio.

It seems like it's not much for a project but this shell application contains a login screen with default login logic, a menu template that can be further modified, and other layout templates.

It's important to note that the business logic will reside in the `src/hot` folder, by default. And, the package naming conventions that were used in New Seam project wizard could have been changed to something different from *org.domain.workshop.session.* Also, notice that there is a default `Authenticator.java` file. This is where custom security logic can be added. Seam has a nice declarative security model that we will explore in more detail later on. The `src/main` folder is a model directory. It stores the project's JPA entity beans.

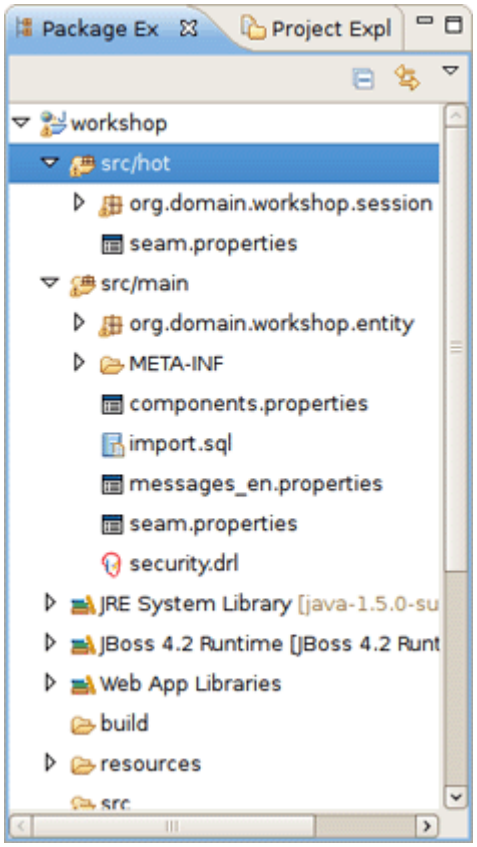

Figure 1.20. Project Structure

The view tier of the application is also important. Seam uses facelets and there is a built-in facelets GUI editor that includes nice WYSIWYG and component drag/drop functionality. Try this out by opening `home.xhtml` from `WebContent` folder.

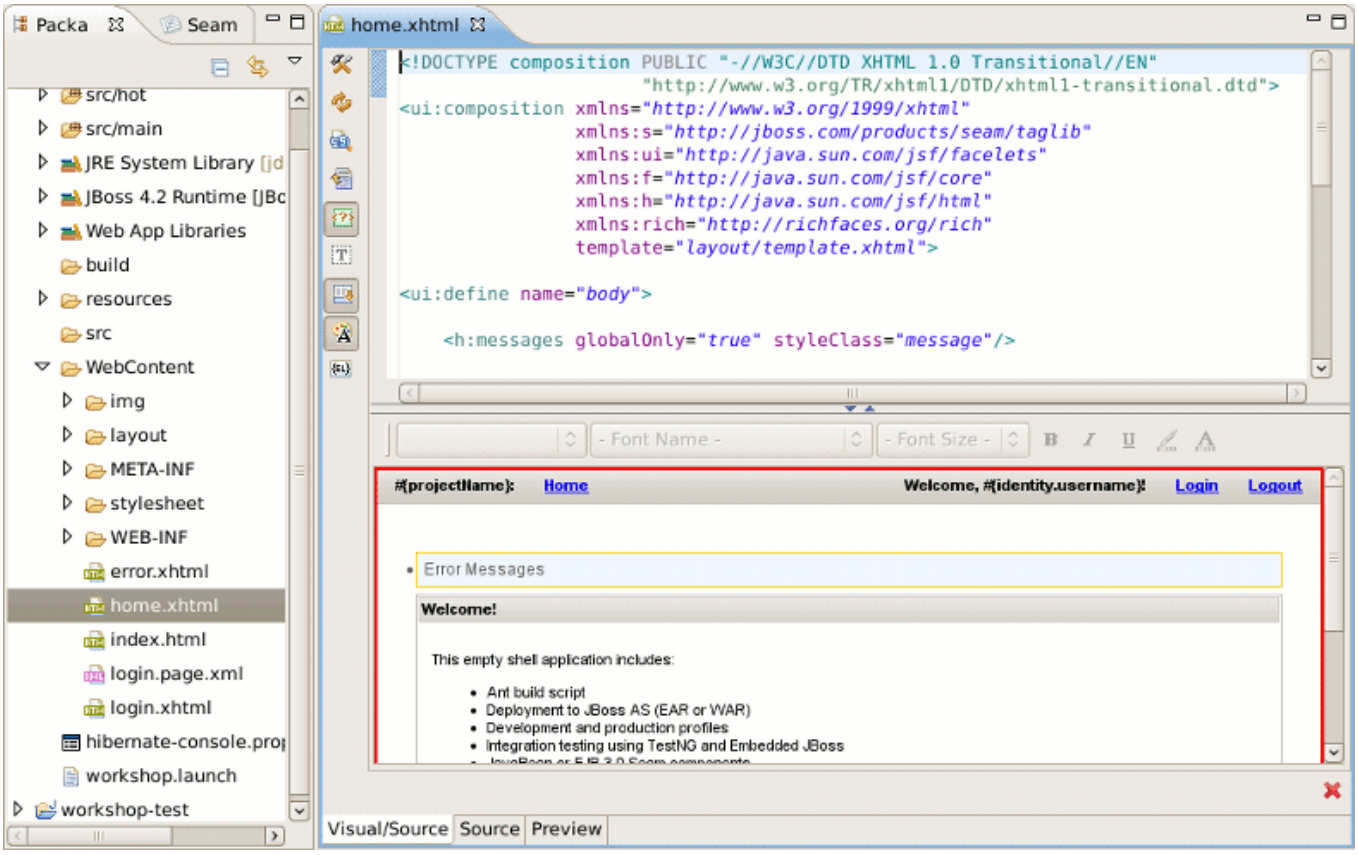

Figure 1.21. Facelets GUI Editor

Notice that the templates reside in the `WebContent/layout` folder. There is a stylesheet in the `WebContent/stylesheet` folder. There is also a login and default error page. The Facelet editor will be explored in more detail later in the lab.

The project already has a datasource that was created via the Seam project wizard database settings. All of the Seam specific configuration files and JAR dependencies are included and located in their proper locations. On last noteworthy line item is related to the build script. There isn't a build script because the Eclipse WTP (Web Tools Project) plugin is used to publish web application changes. As you can see, JBoss Developer Studio is removing a great deal of complexity from the enterprise Java project setup and deployment process. The end result is the developer is writing code, not spending time trying to figure out how to get a decent development environment and project build process.

Seam Action Development

Now it’s time to write some code. The good news is that JBoss Developer Studio can also help out in this respect. In this section, we will create a new Seam Action POJO and facelet with some custom business logic and some GUI changes.

2.1. Create a New Seam Action

Go to main menu bar and click on **File** → **New** → **New Seam Action** to start the New Seam Action wizard.

Specify a Seam component name (e.g., "myAction"). The other properties will be auto-completed for you so there is no need to change them. Click on the **Finish** button.

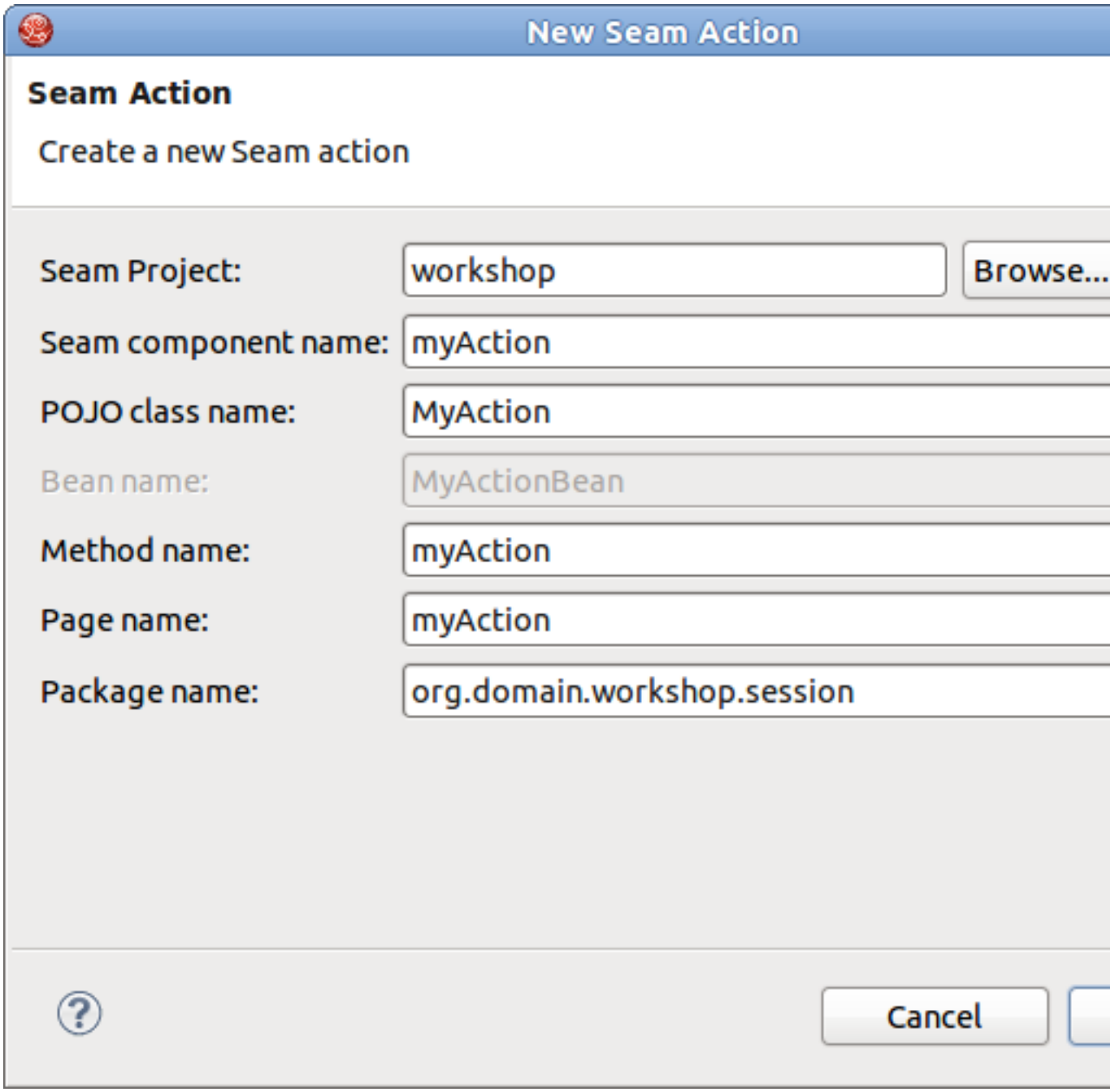

Figure 2.1. New Seam Action Wizard

Now, open the `MyAction.java` file and replace the "myAction" method with this logic:

```
public void myAction() {
Calendar cal = Calendar.getInstance();
log.info("myAction.myAction() action called");
statusMessages.add("MyAction Executed on:" + cal.getTime());
}
```

You also need to import the java.util.Calendar class by clicking **CTRL**+**Shift**+**O**.

2.2. Test Seam Action

The new action can be tested by browsing the workshop-test project. JBoss Developer Studio has already created a TestNG test case for you.

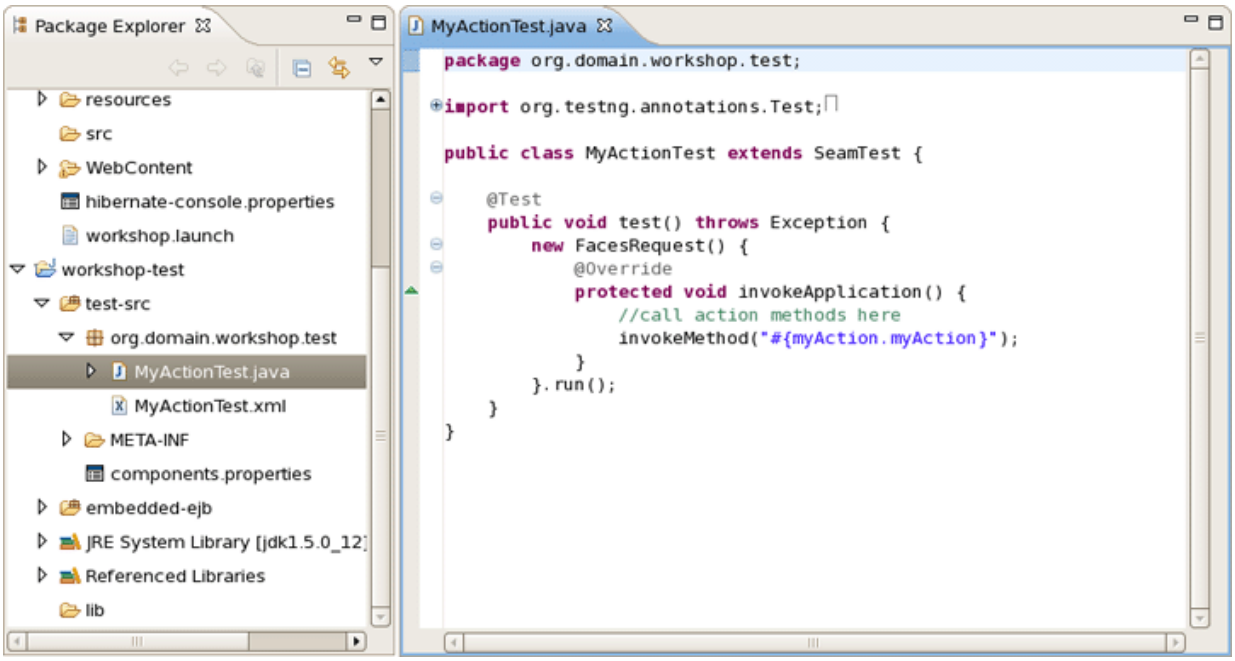

Figure 2.2. "workshop-test" Project

Tip

You may have to refresh the project to see the new files.

The test case simulates a Seam method execution for the MyAction.myAction() logic.

To run the test case, right click on `MyActionTest.xml` and click **Run As** → **TestNG Suite** or use the **Run As...** toolbar shortcut as shown below.

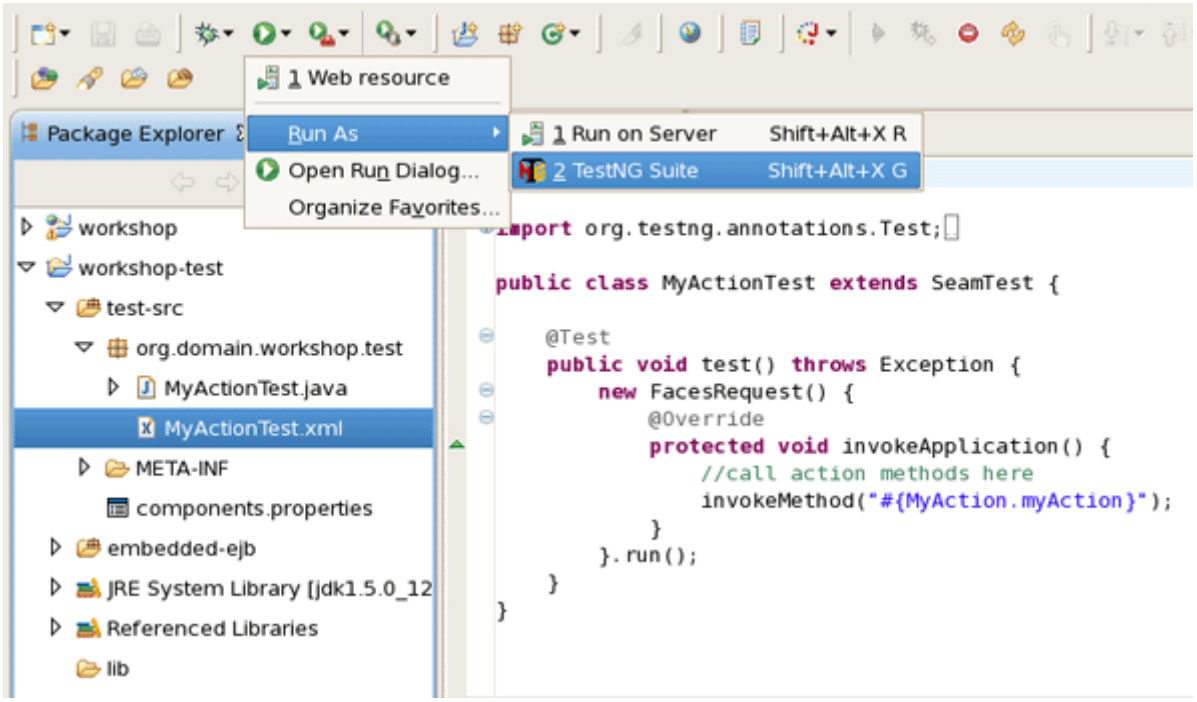

Figure 2.3. TestNG Running

With any luck, the test case will pass. Look at the TestNG view.

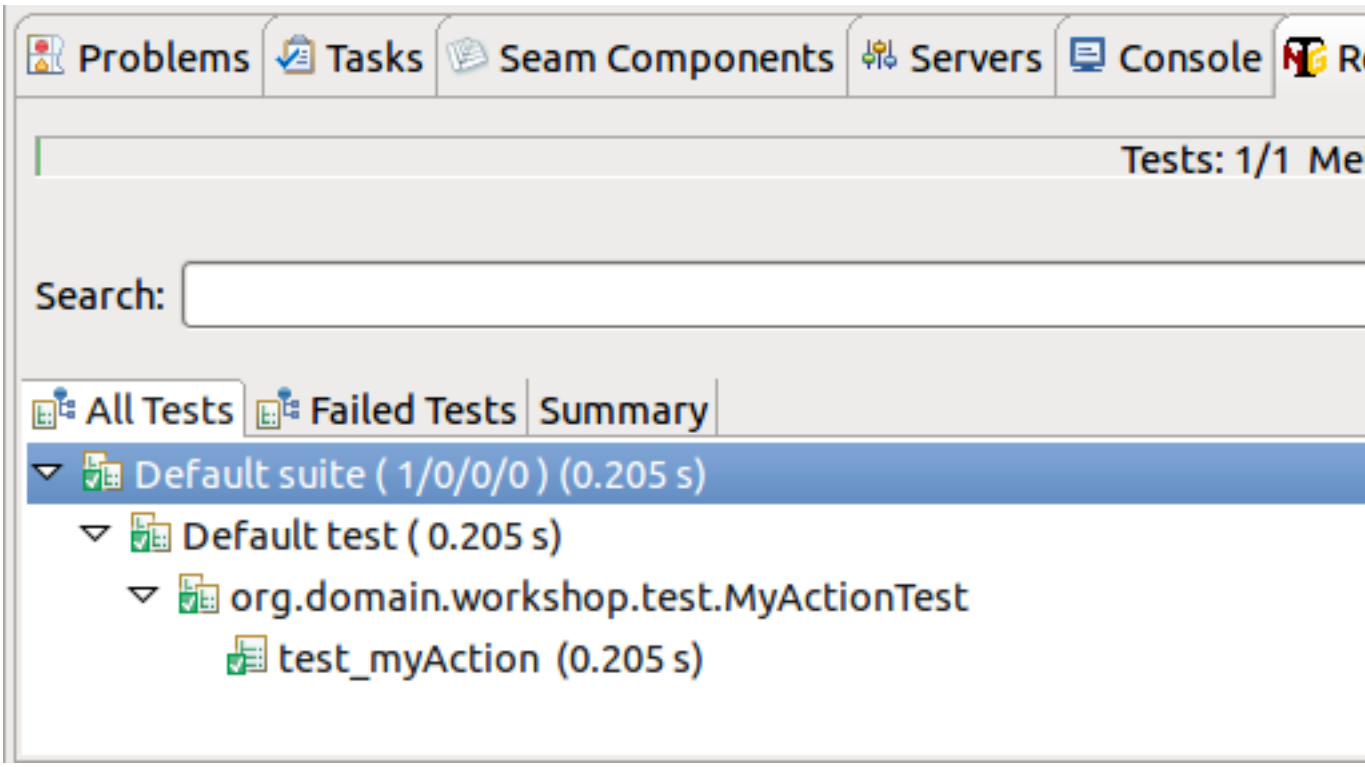

Figure 2.4. TestNG Results

Now, it's safe to test the new Seam Action in a web browser. The fastest way to do that is to right click on `myAction.xhtml` and use **Run As...** → **Run On Server** which will show the appropriate URL in the browser. Alternatively you can manually enter *http://localhost:8080/workshop/myAction.seam* into a browser.

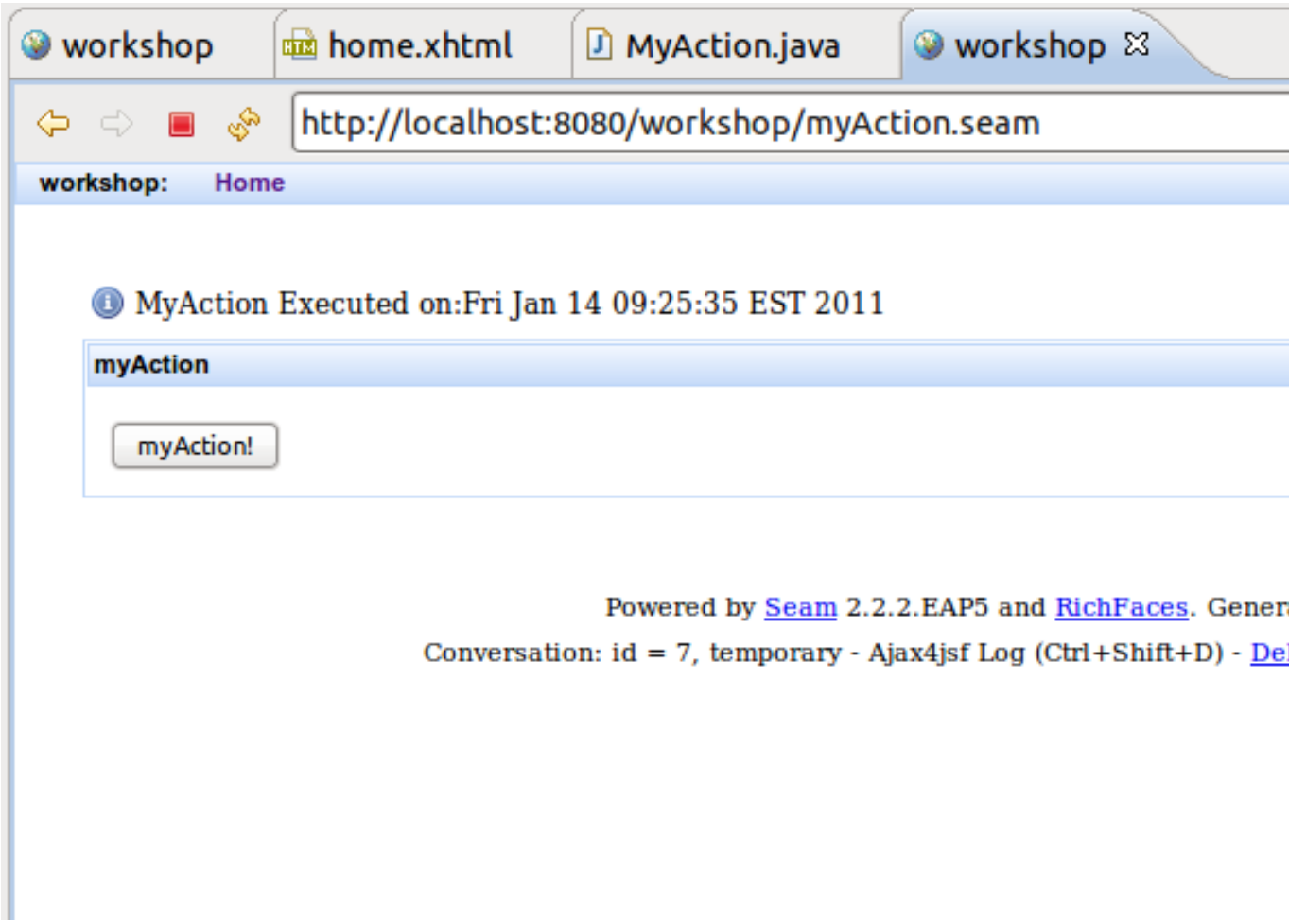

Figure 2.5. Seam Action in a Web Browser

2.3. Modify Seam Action User Interface

Browse to *http://localhost:8080/workshop/myAction.seam* and click on the **myAction** button. This executes the "myAction" method. This looks pretty good, but we could make this page look a little better.

Open *`WebContent/myAction.xhtml`* in JBoss Developer Studio to use the nice facelets editor.

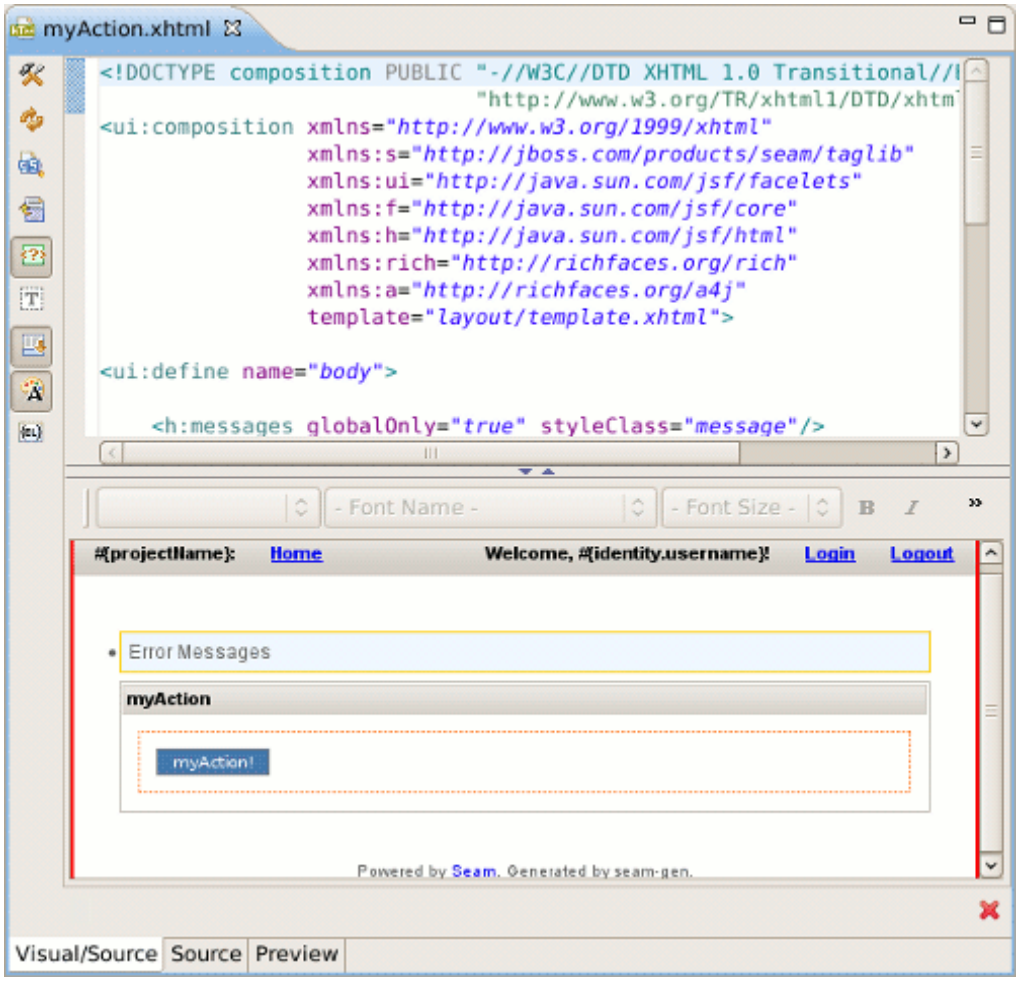

Figure 2.6. Open Seam Action with Editor

Right click on the "myAction!" button in the visual part of editor and select <h:commandButton> Attributes.

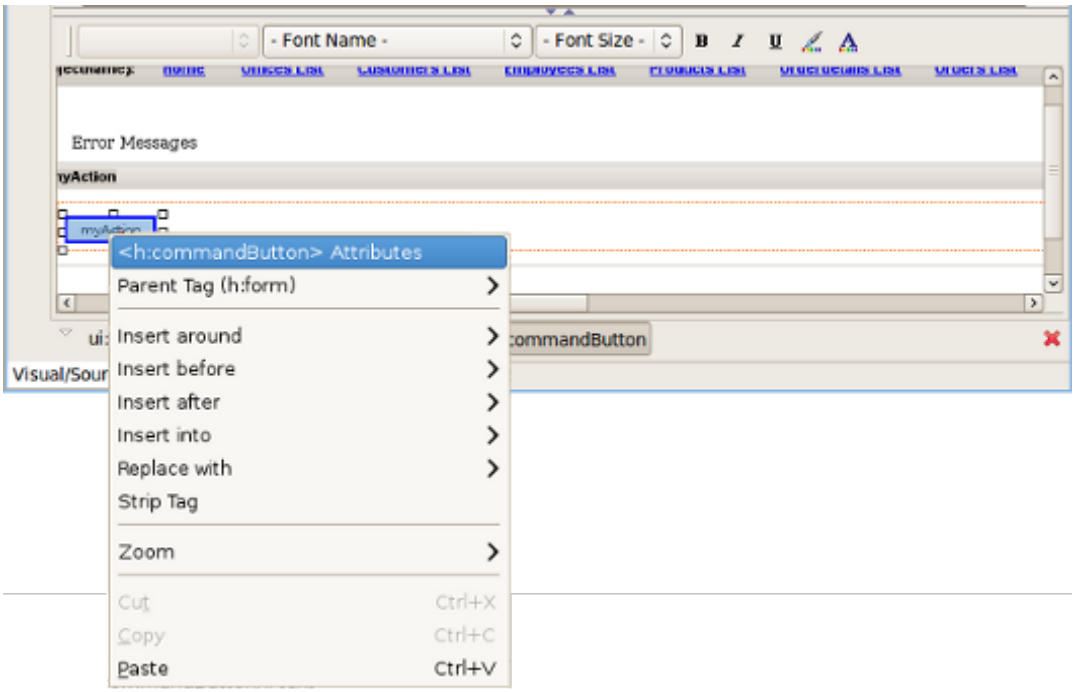

Figure 2.7. Seam Action Editing

Change the value of the button to something different. If desired, you can change any other text on the page. Then, type **CTRL**+**S** to save the facelet.

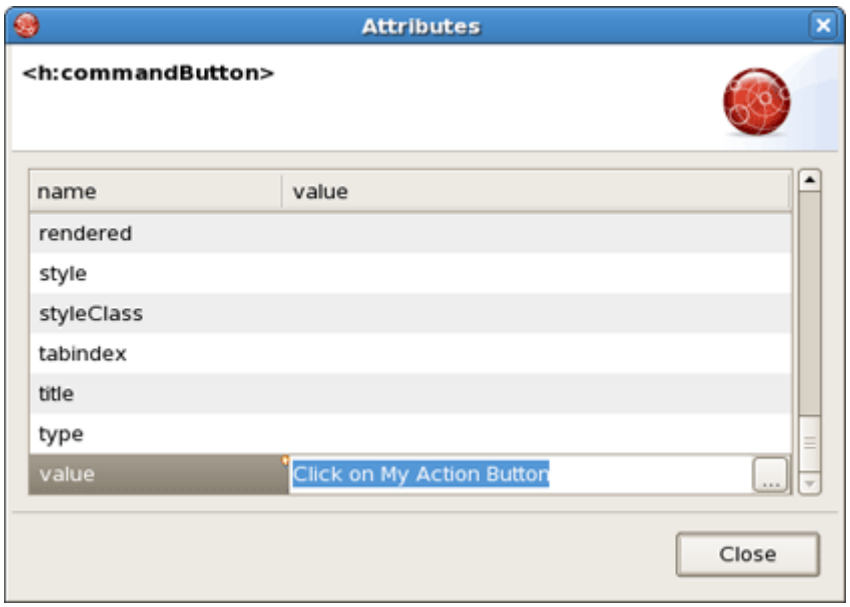

Figure 2.8. Attributes Dialog

Refresh *http://localhost:8080/workshop/myAction.seam* and now you should see your changes.

Notice that you did not have to publish the application. JBoss Developer Studio auto-published it for you.

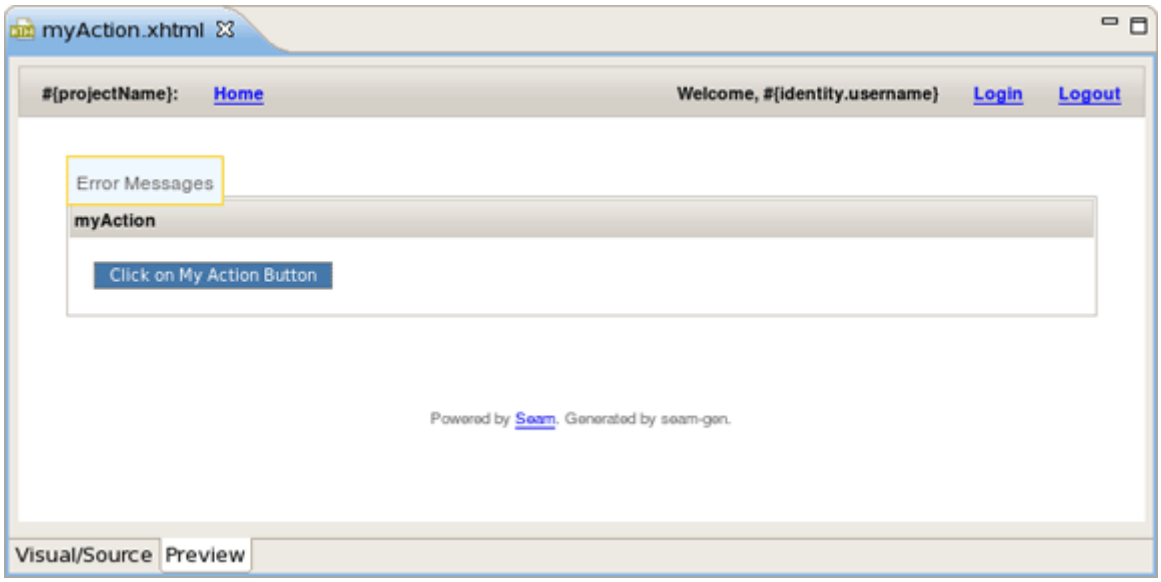

Figure 2.9. Seam Action Is Modified

Declarative Security

In this section you will see how easy it is to secure the facelets and facelet components in Seam. Let's go ahead and secure the action button, then we will secure the entire page.

3.1. Edit Login Authentication Logic

There is a class called `Authenticator.java`. The login page will execute the Authenticator.authenticate() method by default, so we'll start by viewing the authentication logic.

Open `Authenticator.java` in JBoss Developer Studio and you will see that it contains the authenticate() method with this code:

```
public boolean authenticate()
    {
        log.info("authenticating {0}", credentials.getUsername());
        //write your authentication logic here,
        //return true if the authentication was
        //successful, false otherwise
        if ("admin".equals(credentials.getUsername()))
        {
            identity.addRole("admin");
            return true;
        }
        return false;
    }
```

3.2. Secure Seam Page Component

Open `myAction.xhtml` and add a new secured command button:

```
<h:commandButton id="myActionSecured"
value="Secured Action Button"
action="#{myAction.myAction}"
rendered="#{s:hasRole('admin')}"/>
```

Refresh *http://localhost:8080/workshop/myAction.seam* If you are not logged in you will only see one button. If you are logged in, there will be two buttons.

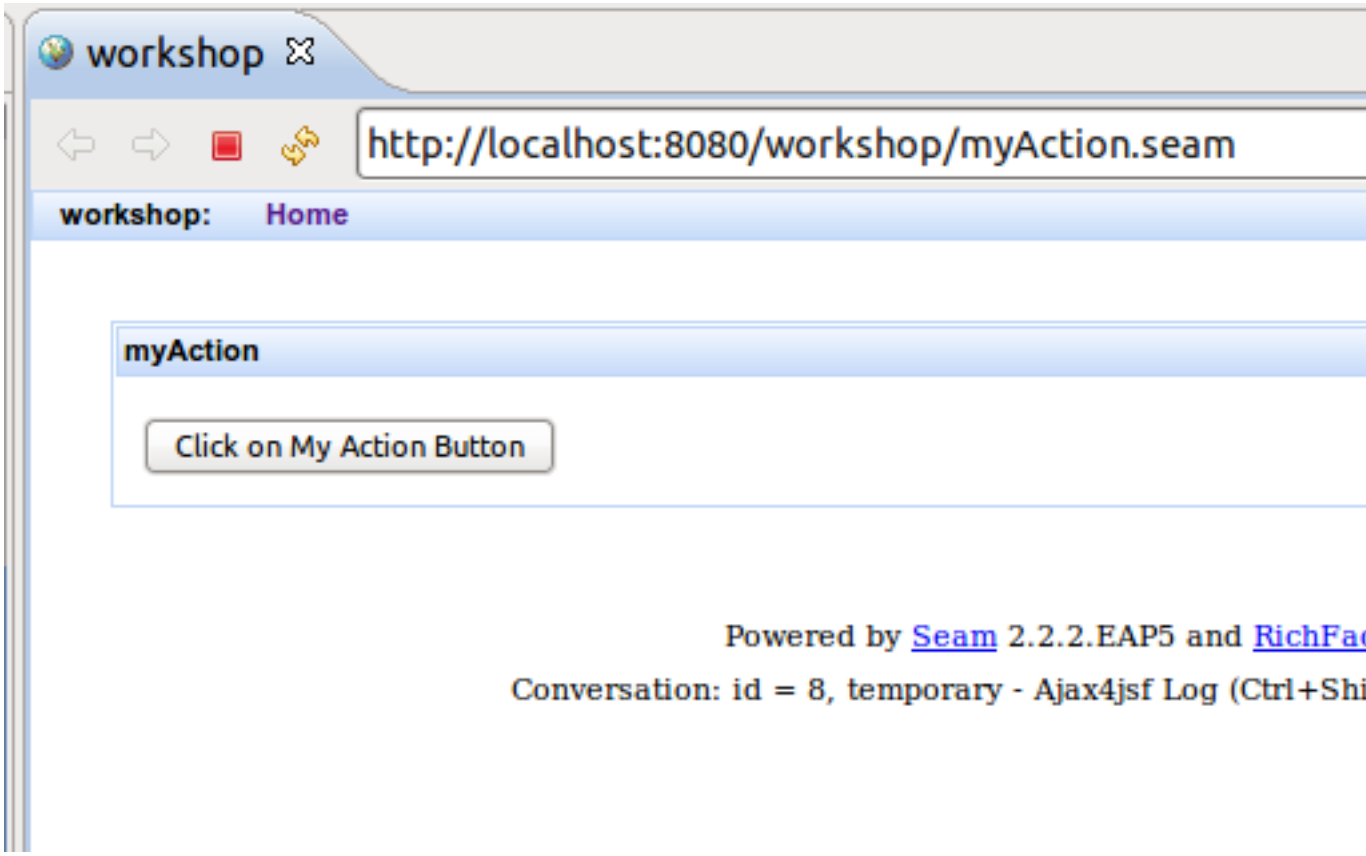

Figure 3.1. One Button on a Page

The secured button is not visible because the user isn't logged in as "admin".

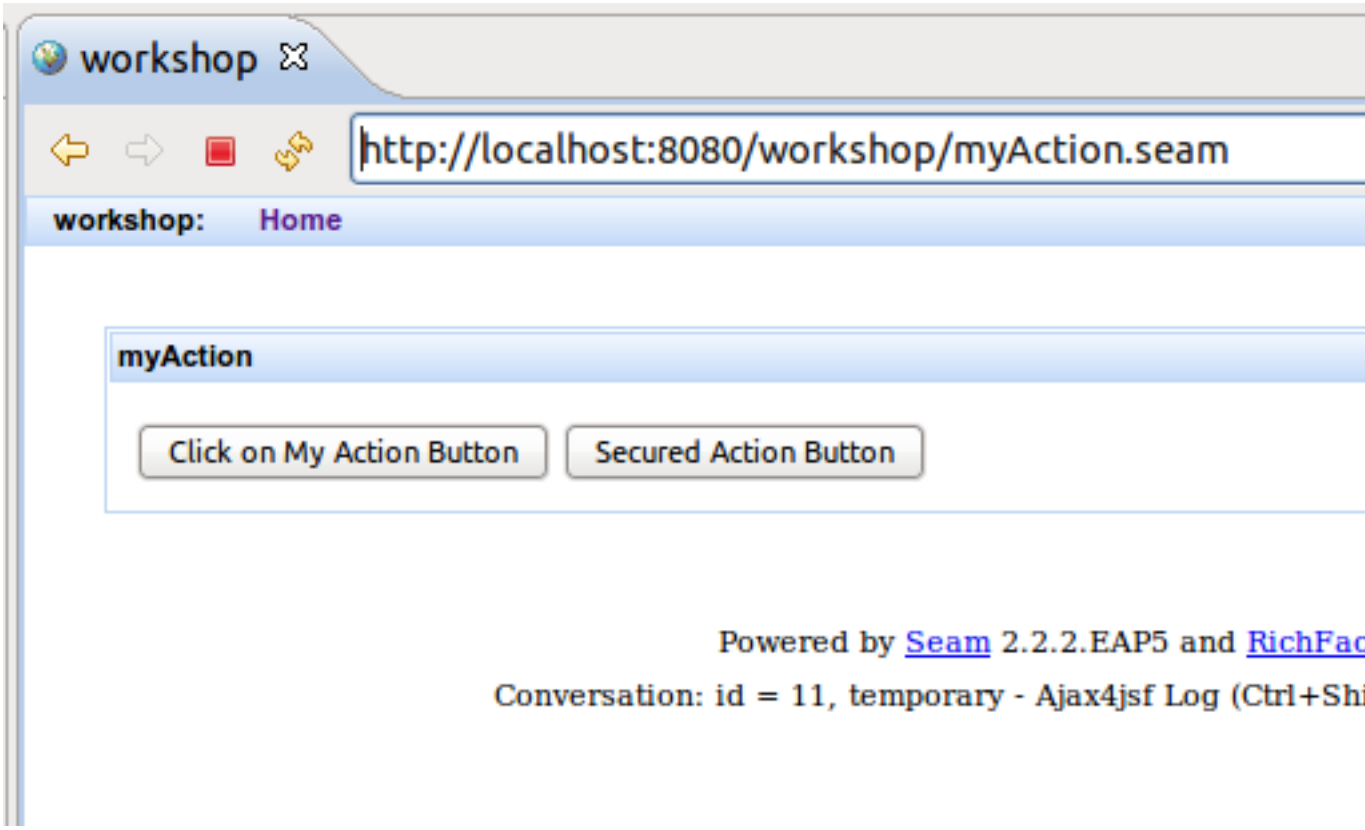

Figure 3.2. Secured Button is Visible

The user is logged in as "admin". Securing components is easy but securing pages is pretty simple as well.

Open *`WebContent/WEB-INF/pages.xml`*. Then add this markup directly underneath the <pages> element:

```
<page view-id="/myAction.xhtml" login-required="true"/>
```

Refresh *http://localhost:8080/workshop/myAction.seam* If you are not logged in you will get bounced back to the login page.

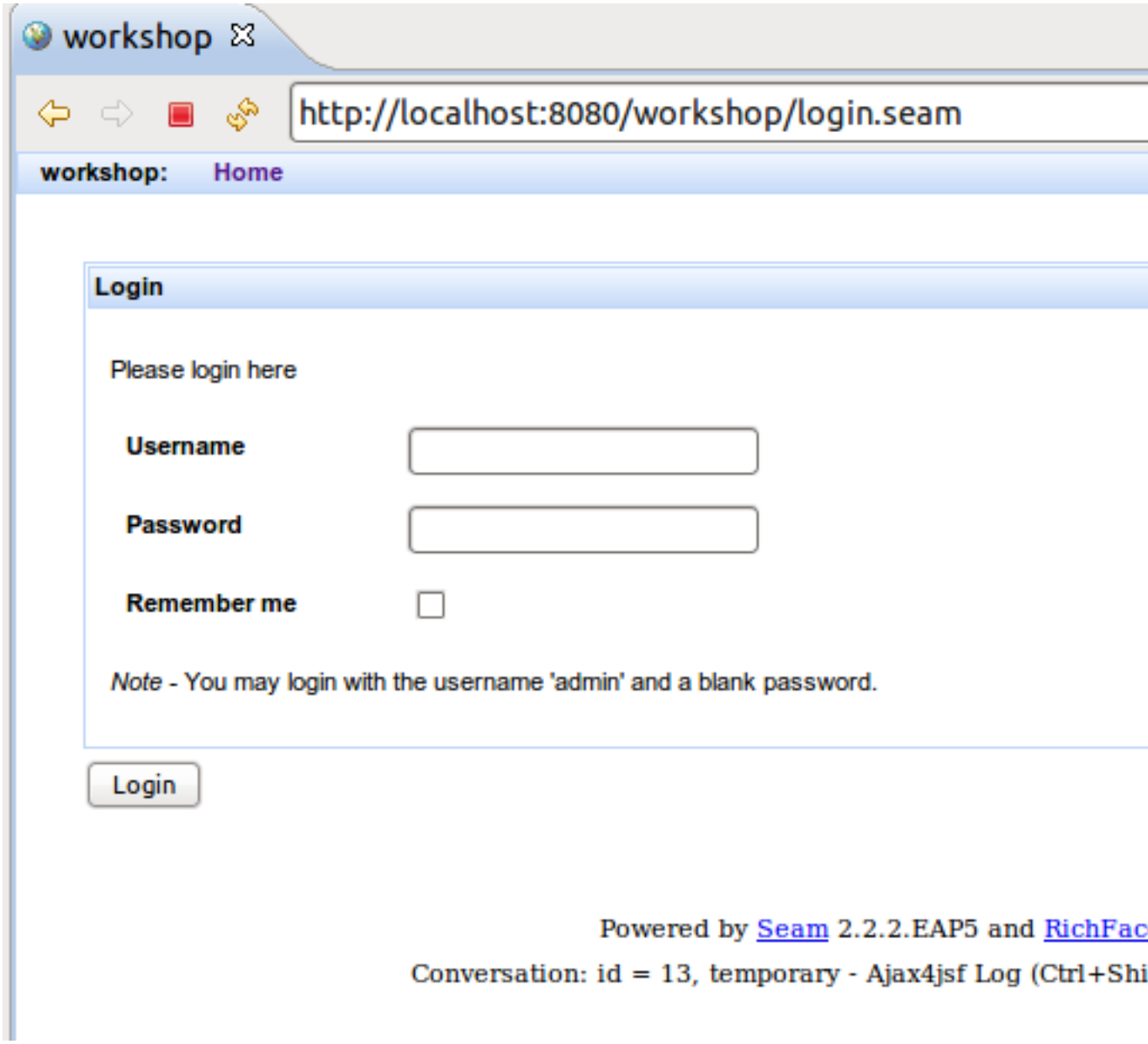

Figure 3.3. Login Page

Thus, if you enter login credentials for the "admin" user, you will be re-directed to the secured page and secured component. If you enter different login credentials, page access will be granted, but the secured component will not be displayed.

Congratulations! You have secured your new action both at the facelet component and page level. You also added custom authentication logic to the login action.

Browsing Workshop Database

In this section you get to know how to use the workshop database that was started at the beginning of the lab.

4.1. Database Connectivity Setup

The workshop data can be browsed inside of JBoss Developer Studio.

To open the Data Source Explorer, click on **Window** → **Open Perspective** → **Other** → **Database Development**.

In the Data Source Explorer, expand the Databases node and select the Default database. Right click on it, select **Connect** from the context menu.

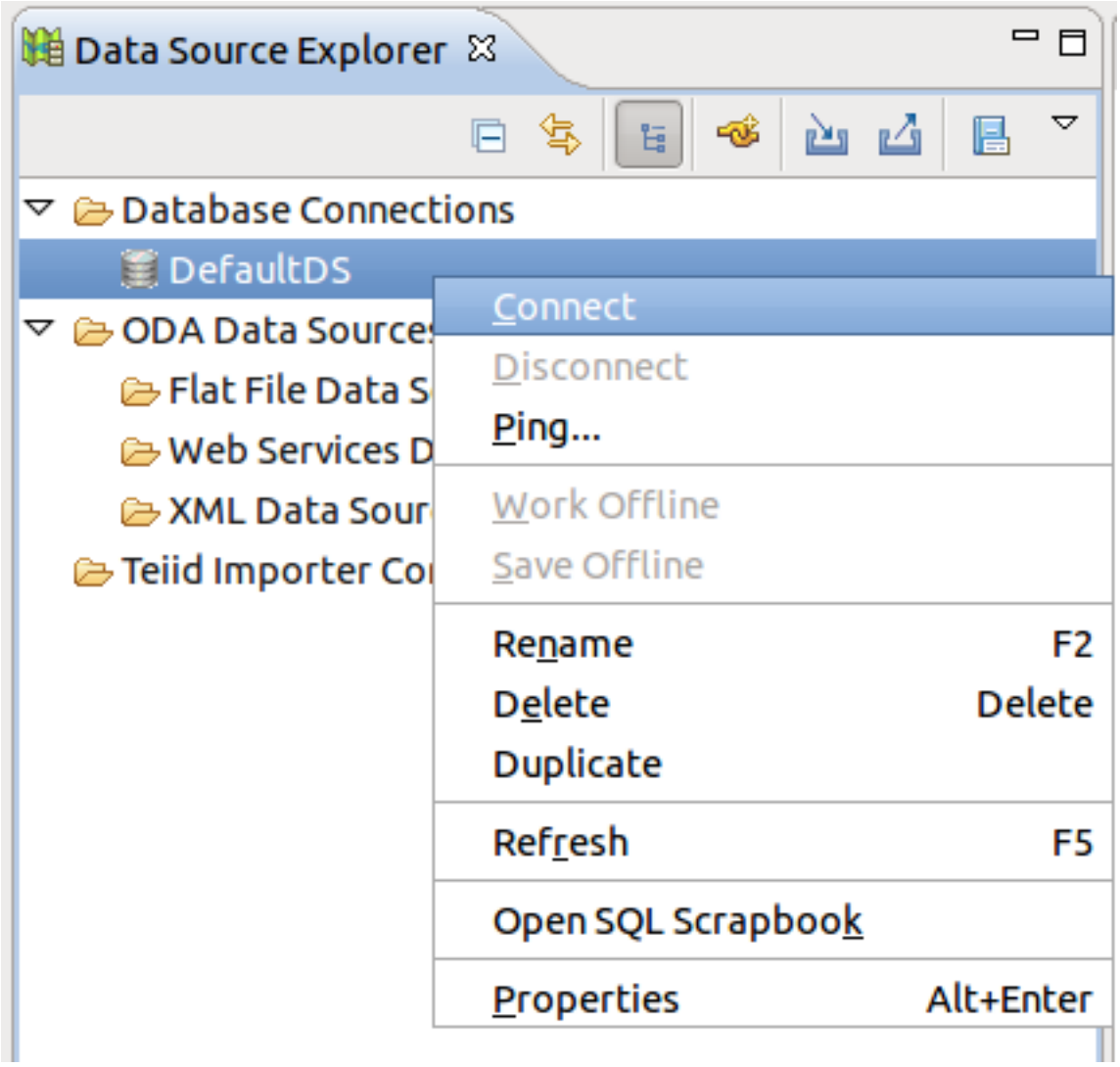

Figure 4.1. Data Source Explorer

4.2. Browse Workshop Database

Then in the current view, drill down to the CUSTOMERS table.

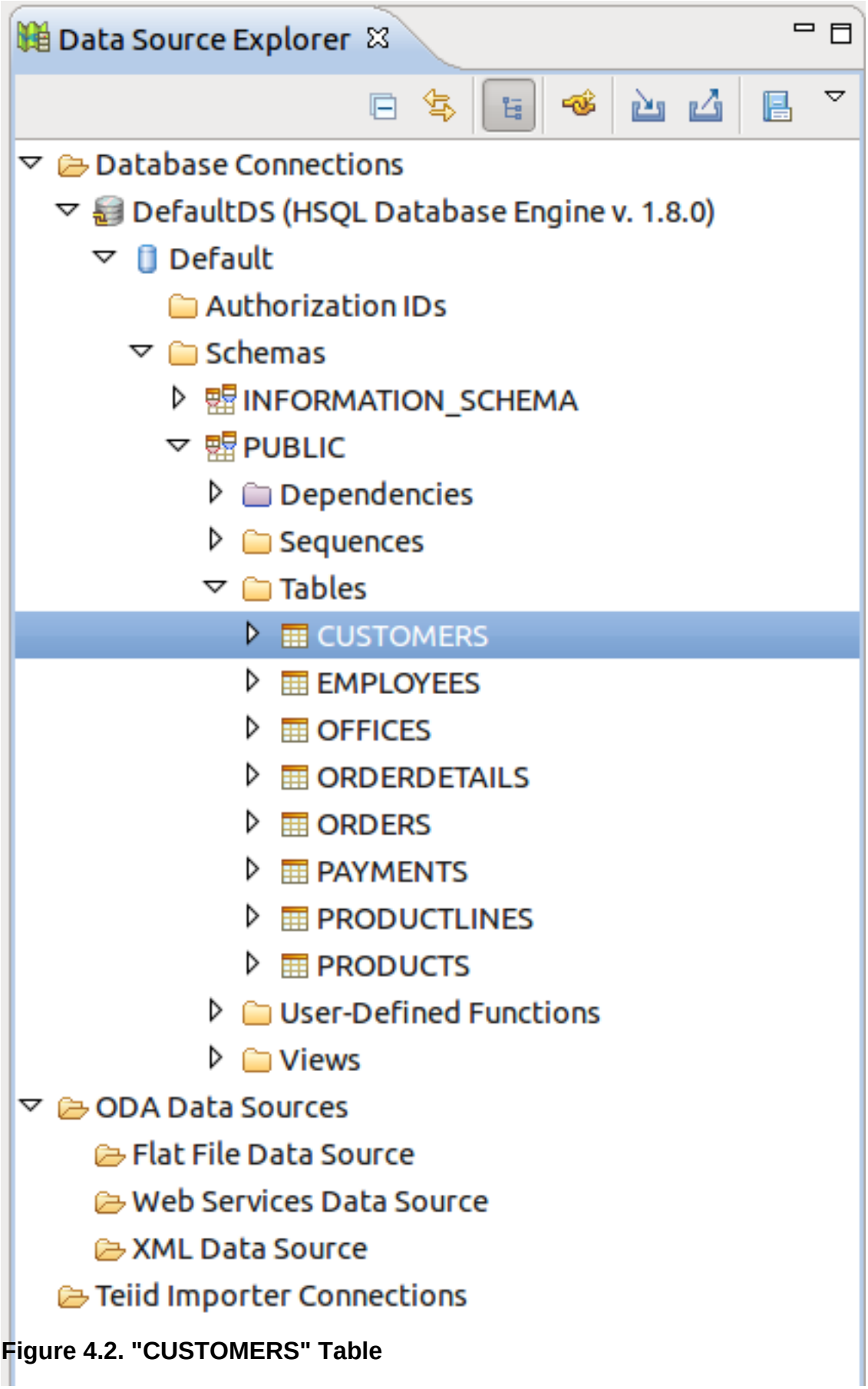

Figure 4.2. "CUSTOMERS" Table

Right click on CUSTOMERS, select **Data** → **Sample Contents** to view the data in the table.

There should be a SQL Results view on the workbench, but it could be hidden. Click on the "Result1" tab in the right side and you should see the data in the CUSTOMERS table.

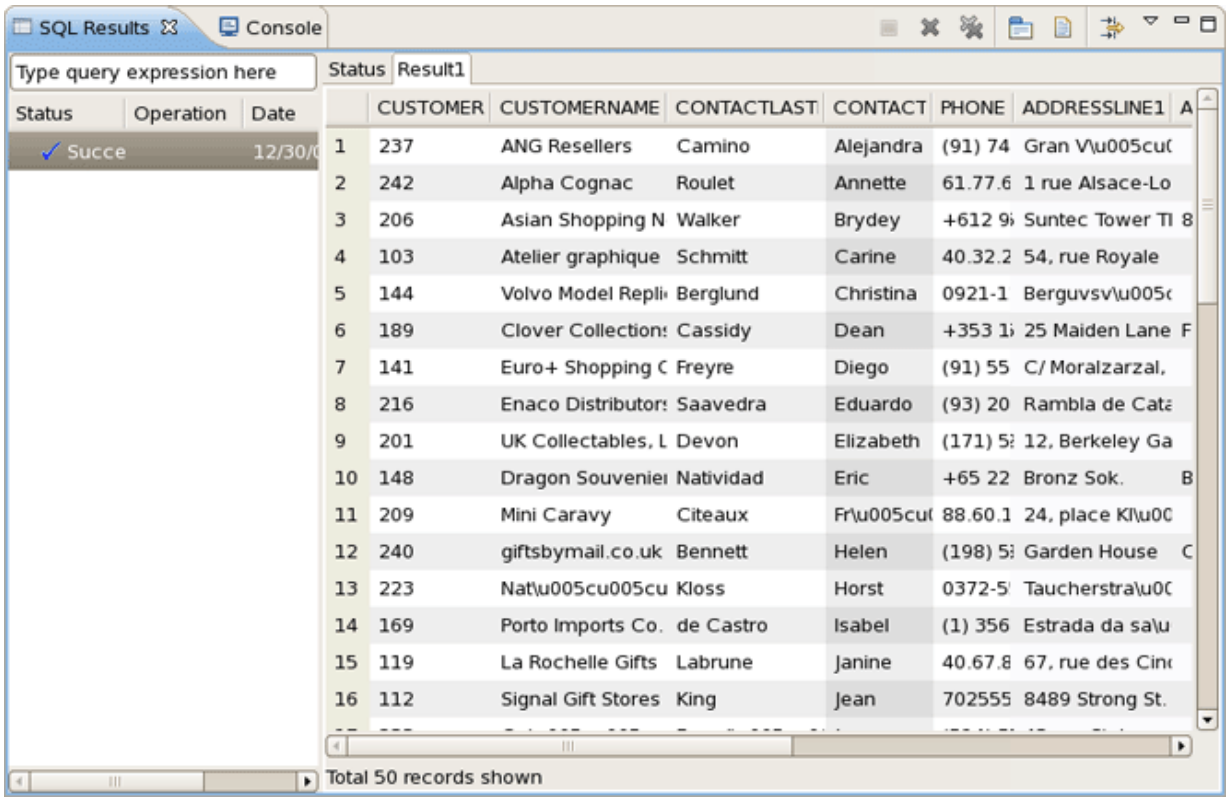

Figure 4.3. SQL Results View

Note:

If you can't find the SQL Results view tab, click on **Window** → **Show View** → **Other** → **SQL Development** → **SQL Results**.

Congratulations! You just connected to the workshop database and queried the content using Database Explorer tools.

Database Programming

Now, it’s time to reverse engineer the workshop database into a fully functioning Seam CRUD (Create Read Update Delete) application.

5.1. Reverse Engineer CRUD from a Running Database

In JBoss Developer Studio, switch to the Seam perspective, and then right-click the project and select **New** → **Seam Generate Entities**.

The "workshop" project in the Seam Generate Entities wizard will be selected automatically. There is no need to change something more, click the **Next** button to proceed to the next step.

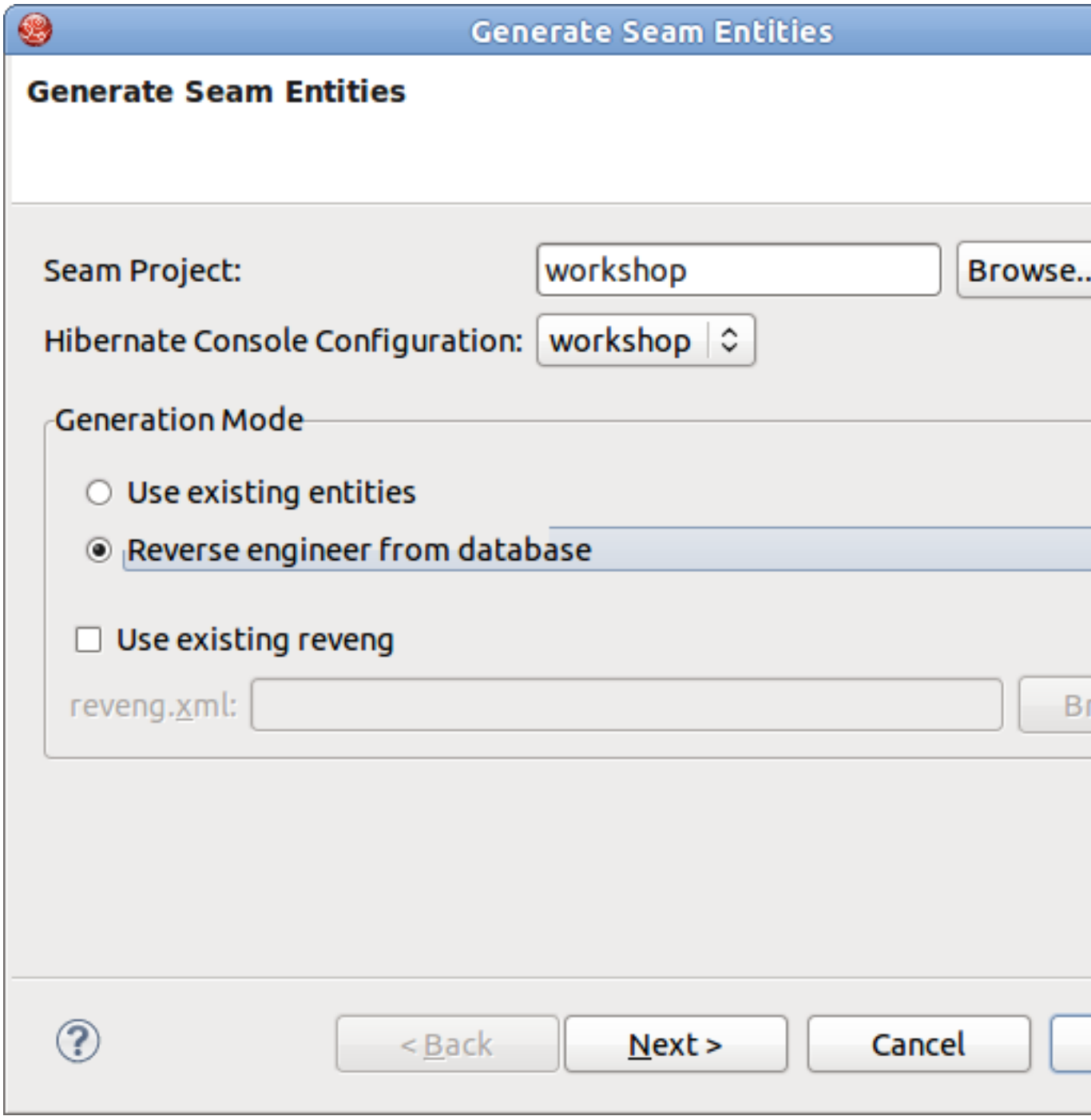

Figure 5.1. Generate Seam Entities

On the next page use the **Refresh** button to display the database, then click the **Include** button to include all the tables from the database, and finally click the **Finish** button.

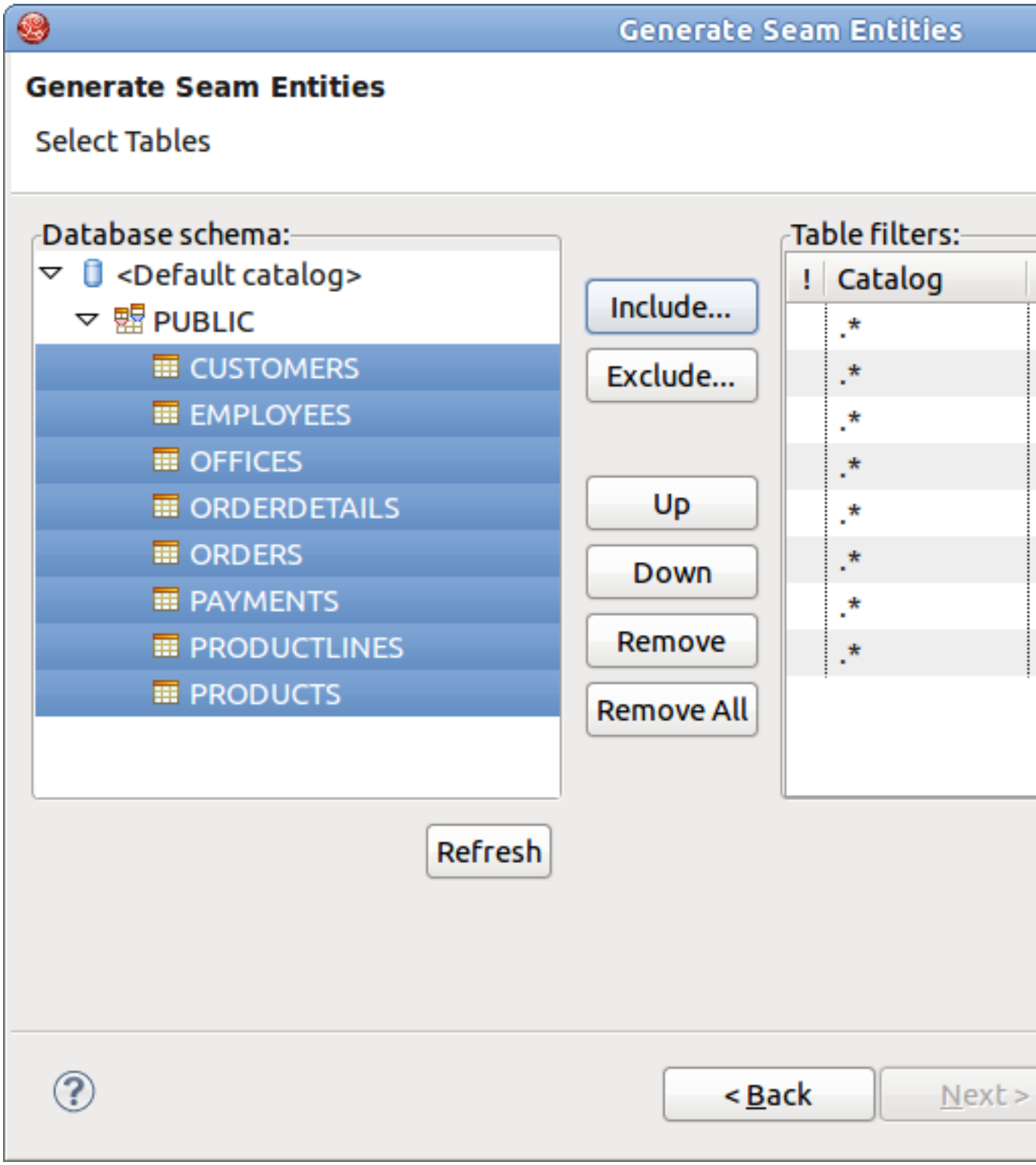

Figure 5.2. Selecting Tables

After running the Generate Entities action, you will see new *org.domain.workshop.entity* classes. These classes represent insert/update/delete/query logic.

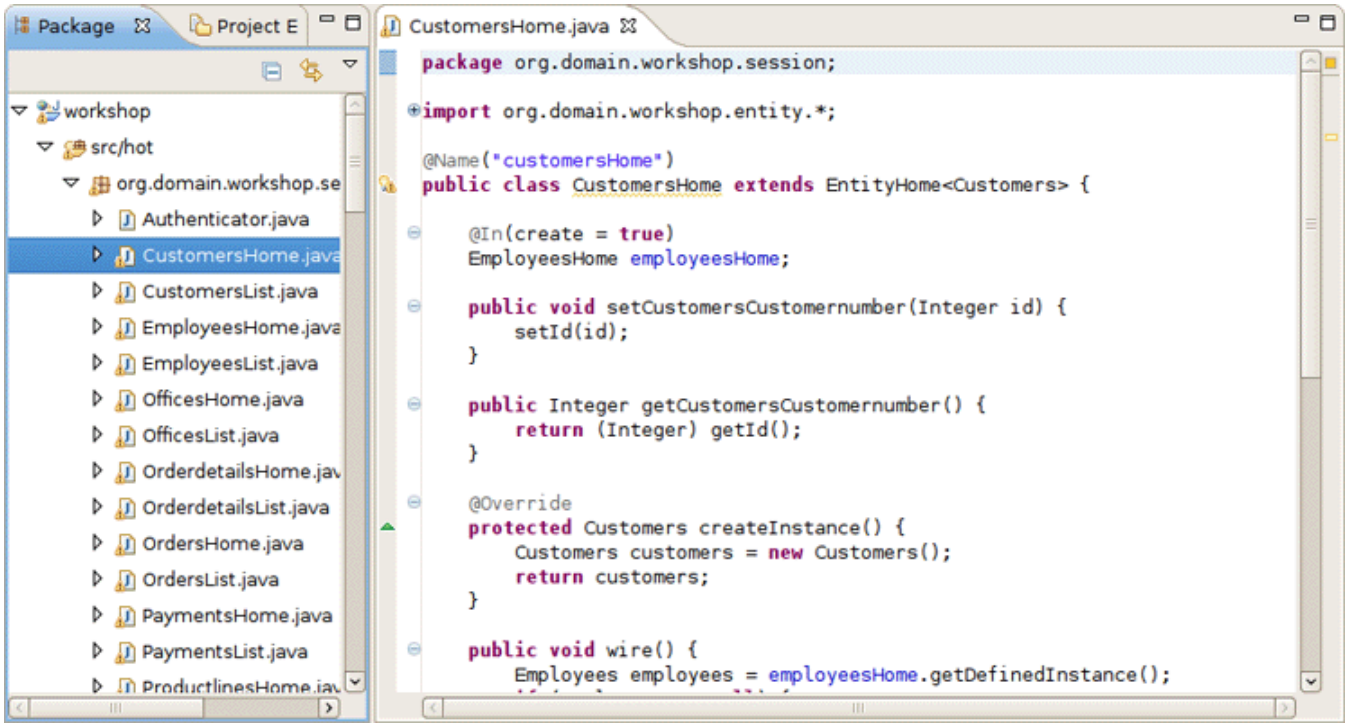

Figure 5.3. org.domain.workshop.entity Classes

There is also the *org.domain.workshop.entity* package that contains the JPA classes. These are the entity beans that are mapped to database tables. Note that you can use Seam refactoring tools with Seam components. Read more about it in *Seam refactoring tools chapter* [http://download.jboss.org/jbosstools/nightly-docs/en/seam/html_single/index.html#seam_refactoring] of Seam Dev Tools Reference Guide.

Last, but not least, there are facelets for all of the CRUD screens. The best way to get a feel for the generated code is to open a browser and play around with the application. Go to *http://localhost:8080/workshop* and insert/update/delete/query a few records. There is quite a bit of AJAX in this application, but we will explore that later on in the lab. For now, take note of the page tabs, required field logic and data table sorting in the list pages.

Tip

If you see the error `java.lang.ClassNotFoundException: org.jboss.seam.servlet.SeamListener` in the console output from the Application Server, you may need to copy the `jboss-seam.jar` file from the `lib` subdirectory in the Seam library (which can be downloaded from *here* [http://seamframework.org/Seam2/Seam2DistributionDownloads]) into the `/server/default/deploy/workshop.war/WEB-INF/lib/` subdirectory in your Application Server (where "default" refers to the server profile that you are using).

Tip

If you see the error `Could not instantiate Seam component: org.jboss.seam.security.ruleBasedPermissionResolver`, copy the `mvel2.jar` file from the Seam library to the same destination directory mentioned in the tip above.

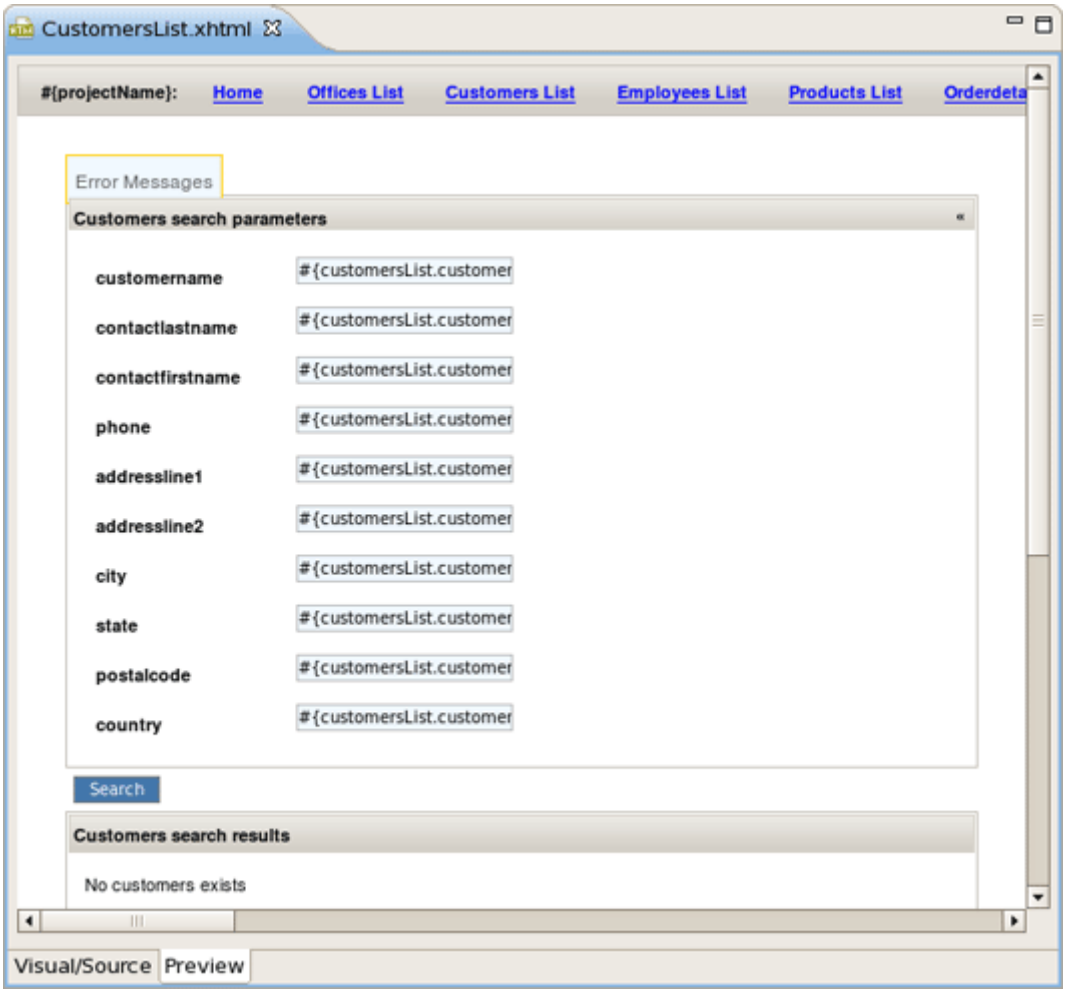

Figure 5.4. CustomersList.xhtml in the Editor

Congratulations! You now have a fully functioning CRUD application that is AJAX enabled.

5.2. Use Hibernate Tools to Query Data via JPA

Now, it's time to write some JPA queries using the Hibernate perspective in JBoss Developer Studio.

In the upper right corner of the workbench there is a small icon (see the figure below), click on it and select **Hibernate**.

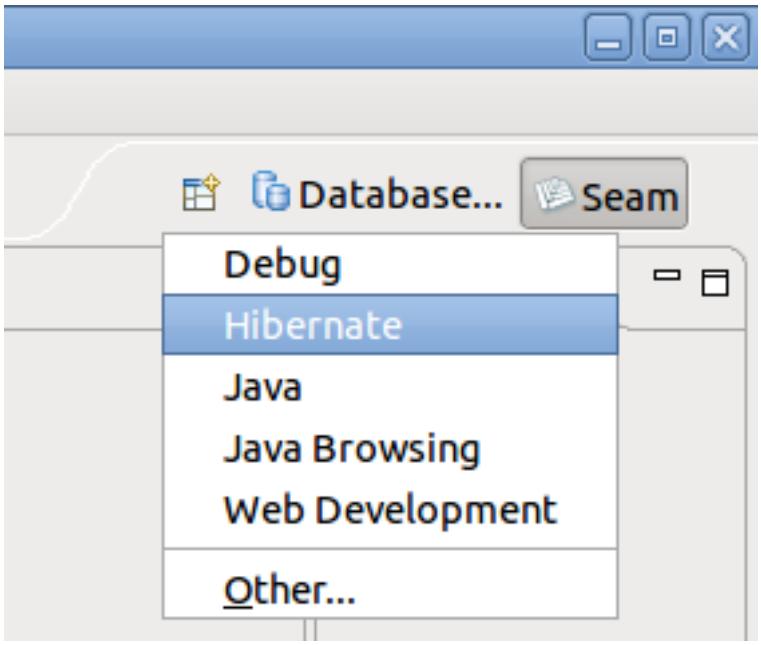

Figure 5.5. Hibernate Perspective

Look at the Hibernate Configurations view. In the "workshop" project, drill down on the Session Factory and notice that the JPA entities/attributes are listed in a nice tree view.

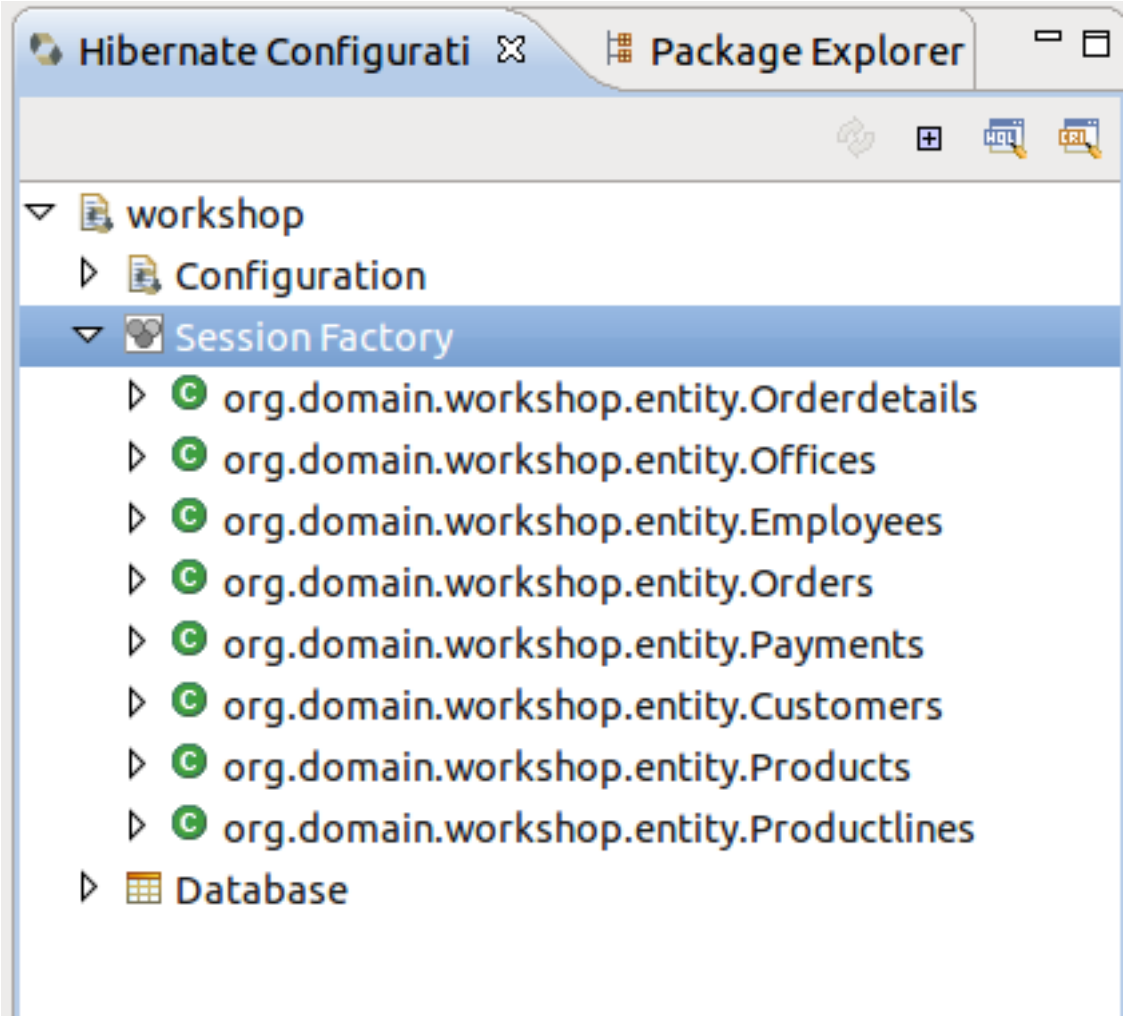

Figure 5.6. Hibernate Configurations View

Right click on the Session Factory and select **HQL Editor**. This will open a JPA query scratch pad window.

Write your query and click on the "Hibernate Dynamic SQL Preview" tab. You should see the SQL that will be executed if this JPA query is run.

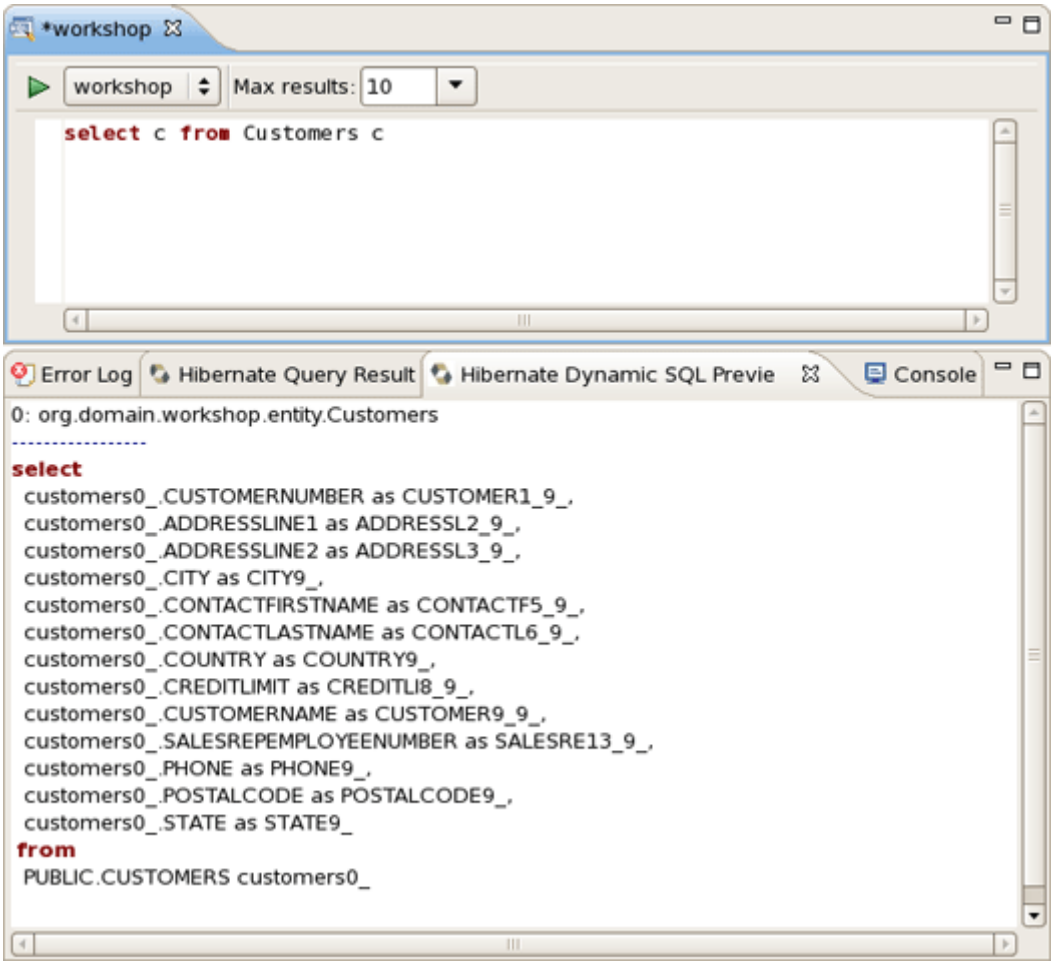

Figure 5.7. JPA Query Editor

Run the query by clicking on the green run icon.

The results are listed in the "Hibernate Query Result" view. There is a "Properties" tab in the workbench that can be used to see a specific JPA result. These results represent the JPA objects because our query did not specify column names.

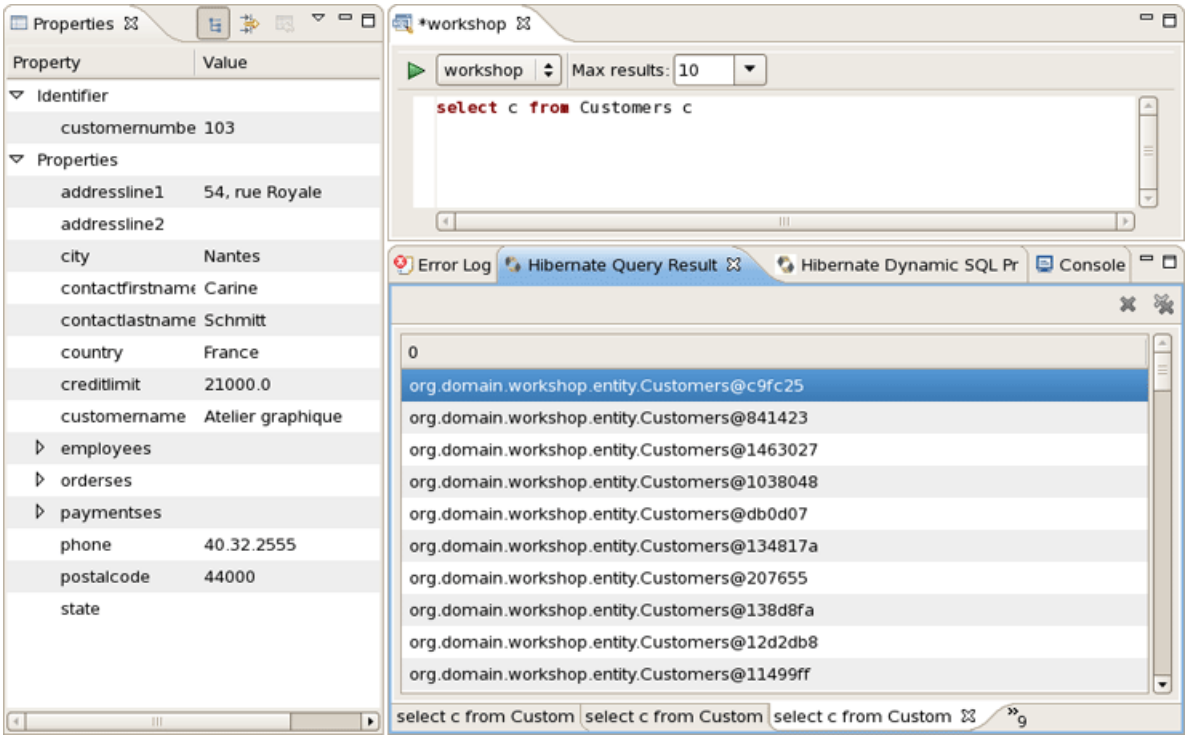

Figure 5.8. Hibernate Query Result View

The query can be refined, and take note that there is nice code completion in the JPA query editor.

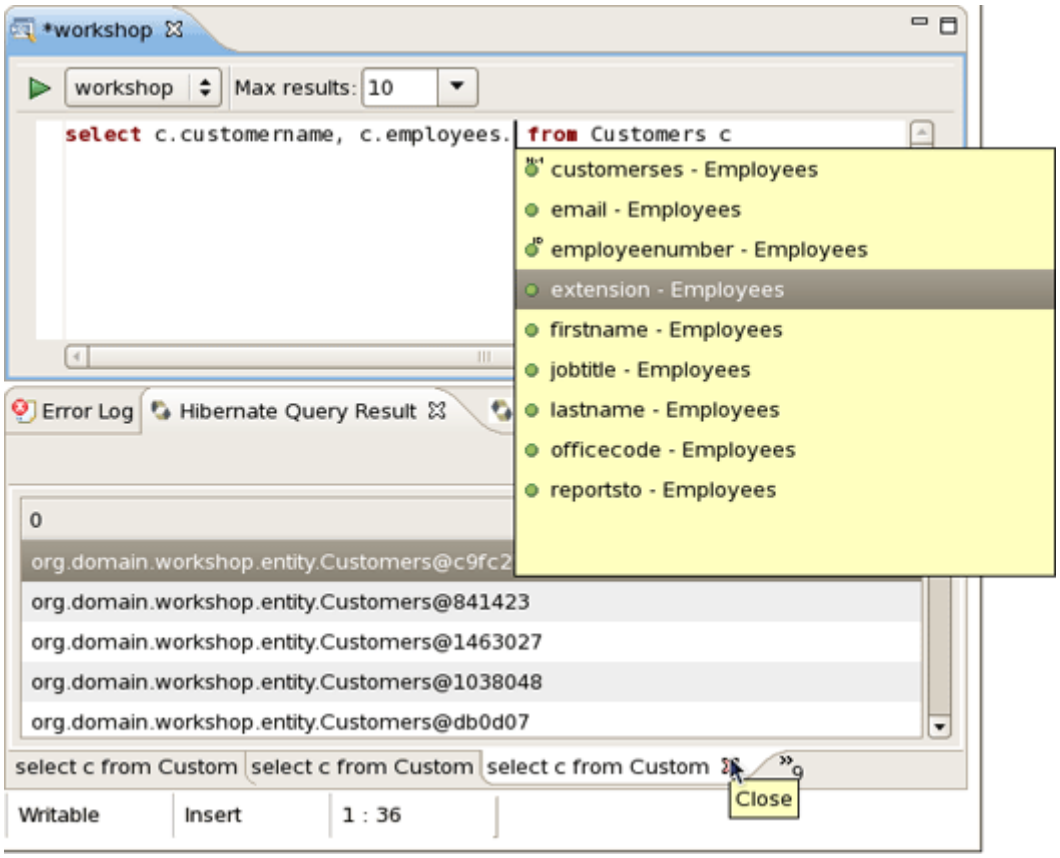

Figure 5.9. Code Completion

A refined query will return results that are more ResultSet oriented. Notice the join logic that JPA supports.

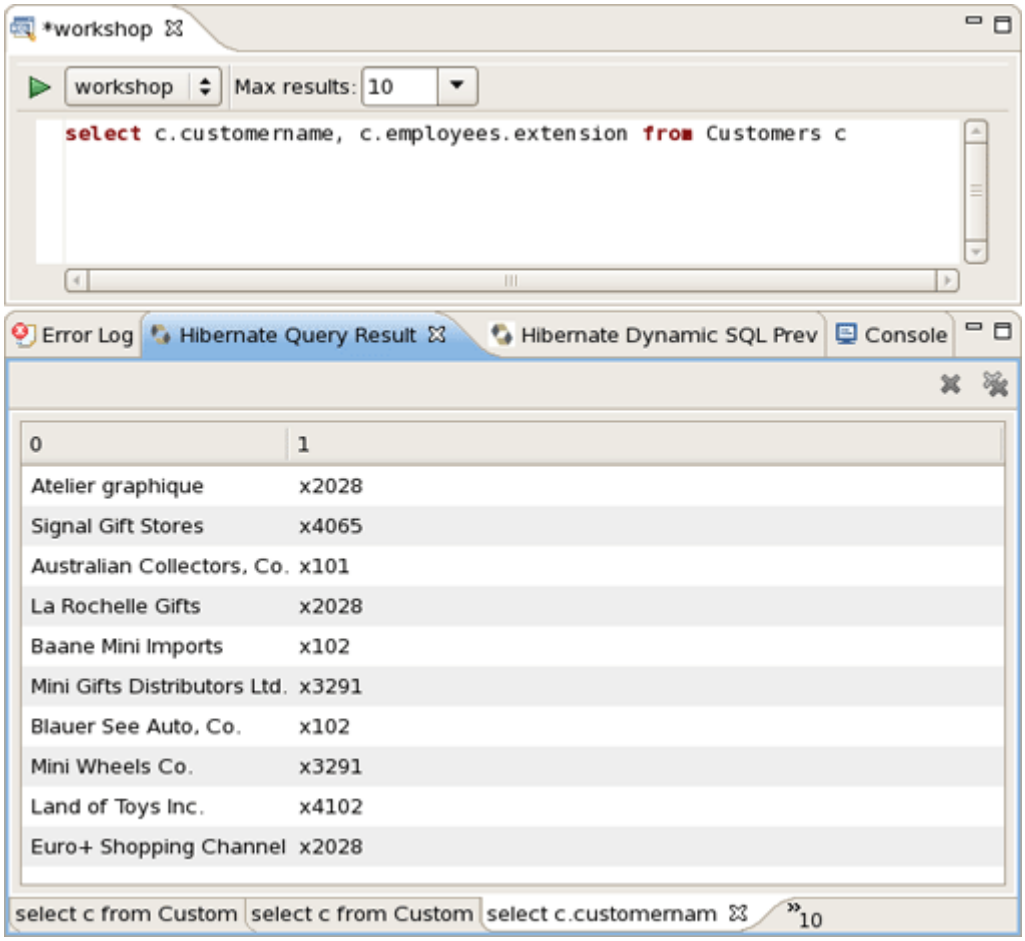

Figure 5.10. The Hibernate Query Result

There was no need to specify an Employees table in the from part of the JPA query because JPA supports reference traversal via Java class attribute references. Not only are JPA and HQL queries fully supported, but Criteria based queries can also be written in the Criteria Editor. You should spend some time tinkering with different queries and possibly Criteria based queries, even though the instructions are not provided in this lab.

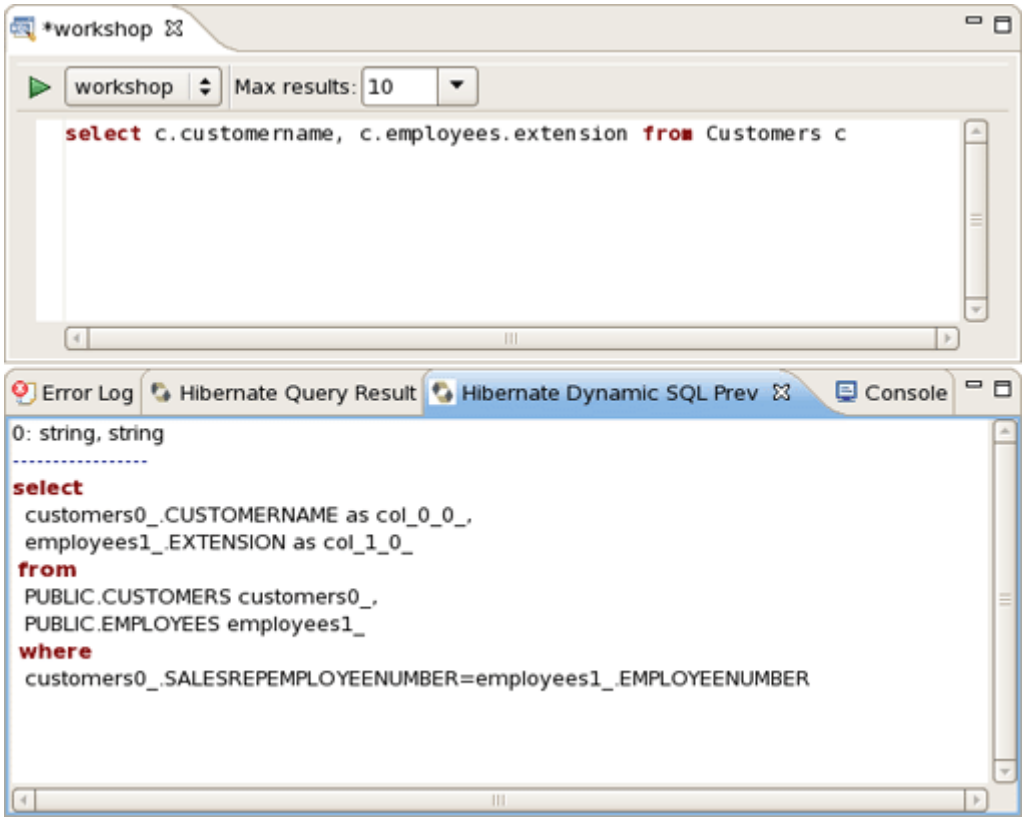

Figure 5.11. Criteria Editor

5.3. Use Hibernate Tools to visualize the Data Model

Now, it's time to view the data model for the workshop database.

In the Hibernate Configurations view, select "workshop" project and expand the Configuration node. Select the Customers entity, right click on it, choose **Mapping Diagram**.

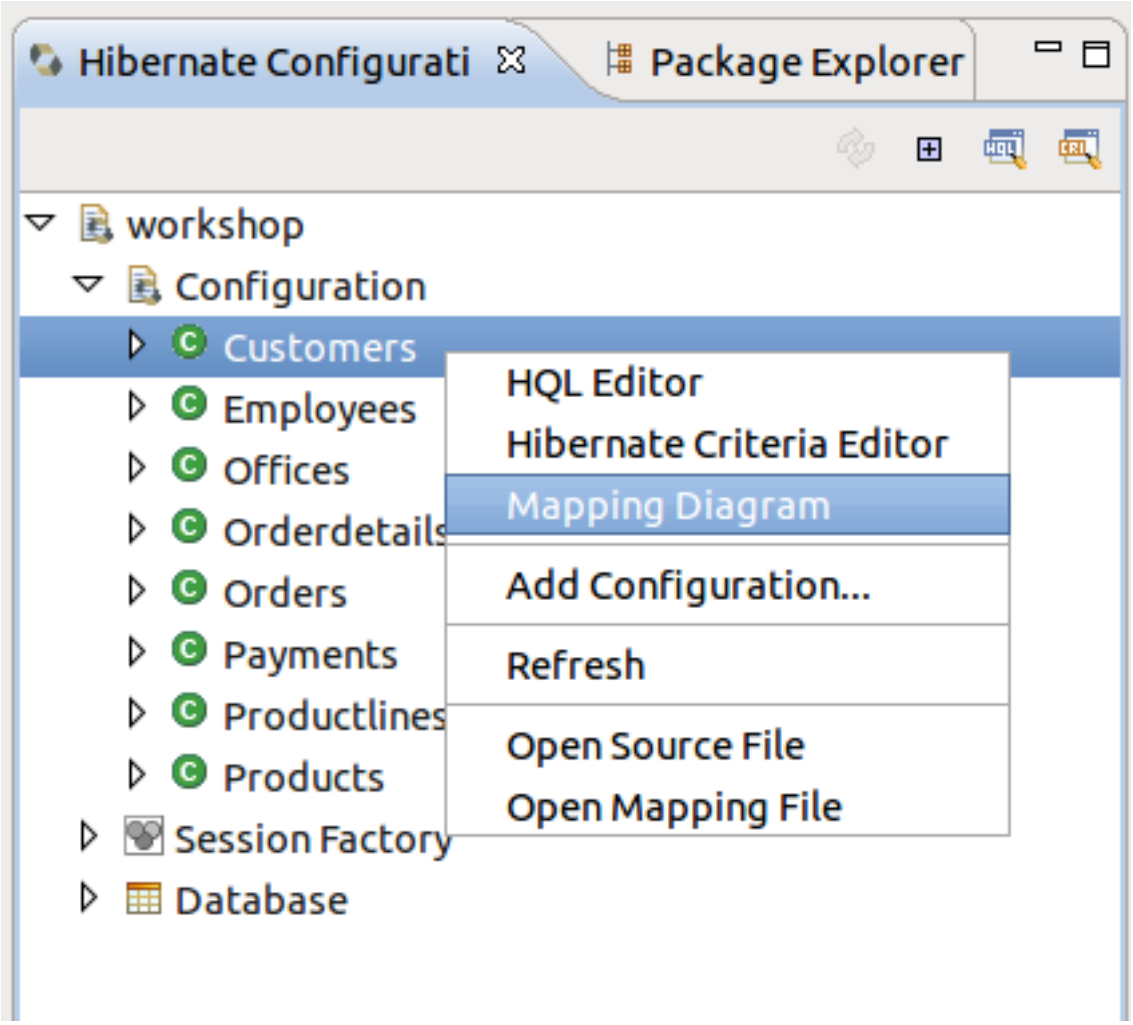

Figure 5.12. Mapping Diagram Opening

You see a Diagram tab for the CUSTOMERS table and any tables that have FK references. This is a handy way to view the data model and JPA mappings. Now, you've got access to something that the Erwin Data Modeler can't do.

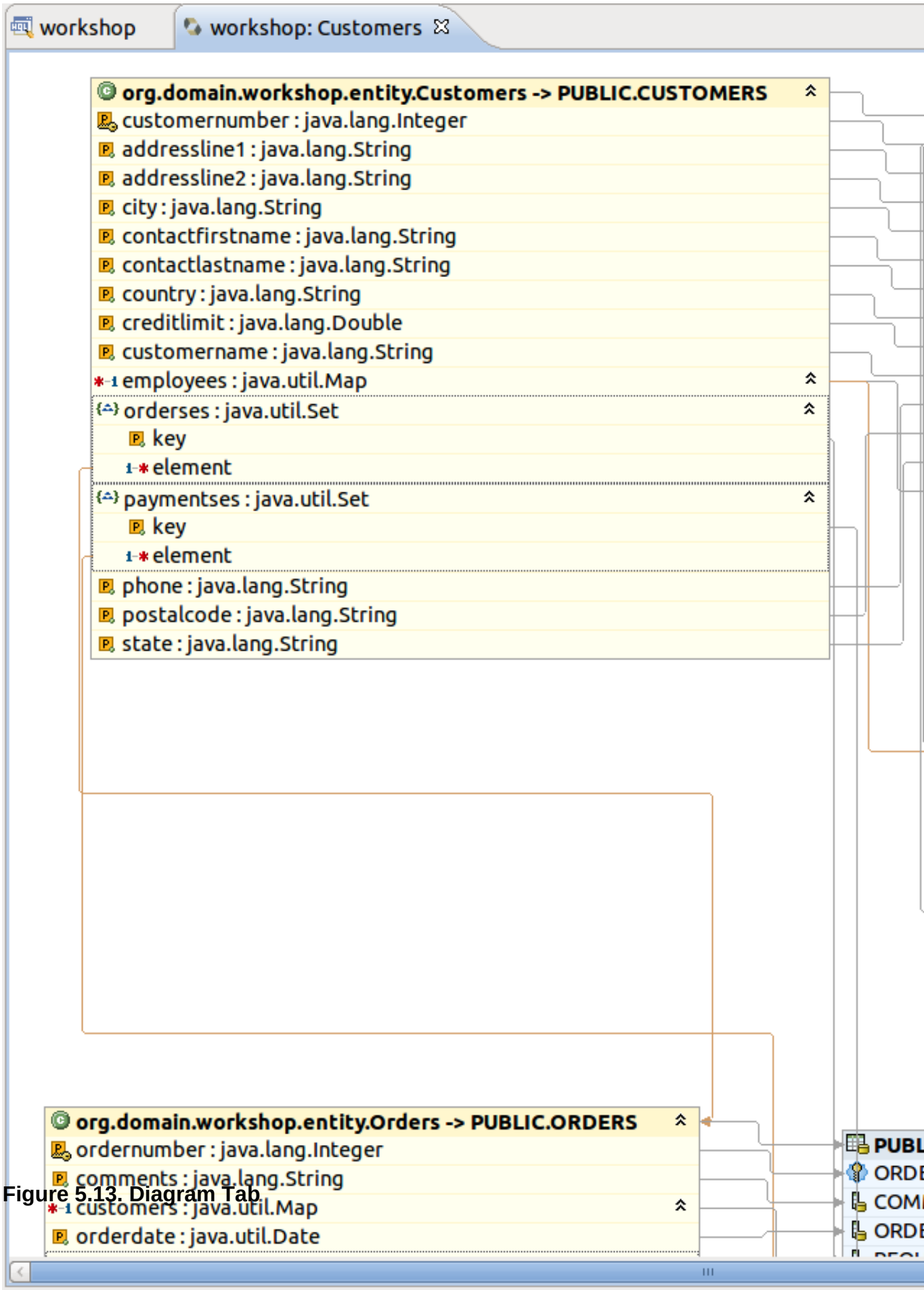

Figure 5.13. Diagram Tab

Rich Components

This lab will conclude with one last AJAX twist. In this section we add a RichFaces inputNumberSlider to the Order Details edit screen.

6.1. Add a Richfaces component to the CRUD Application

Switch to Seam perspective, and open `WebContent/OrderdetailsEdit.xhtml` in JBoss Developer Studio.

Change the form field values using the visual editor. Seam has generated the form field names that match the database column names. This is not ideal for business users.

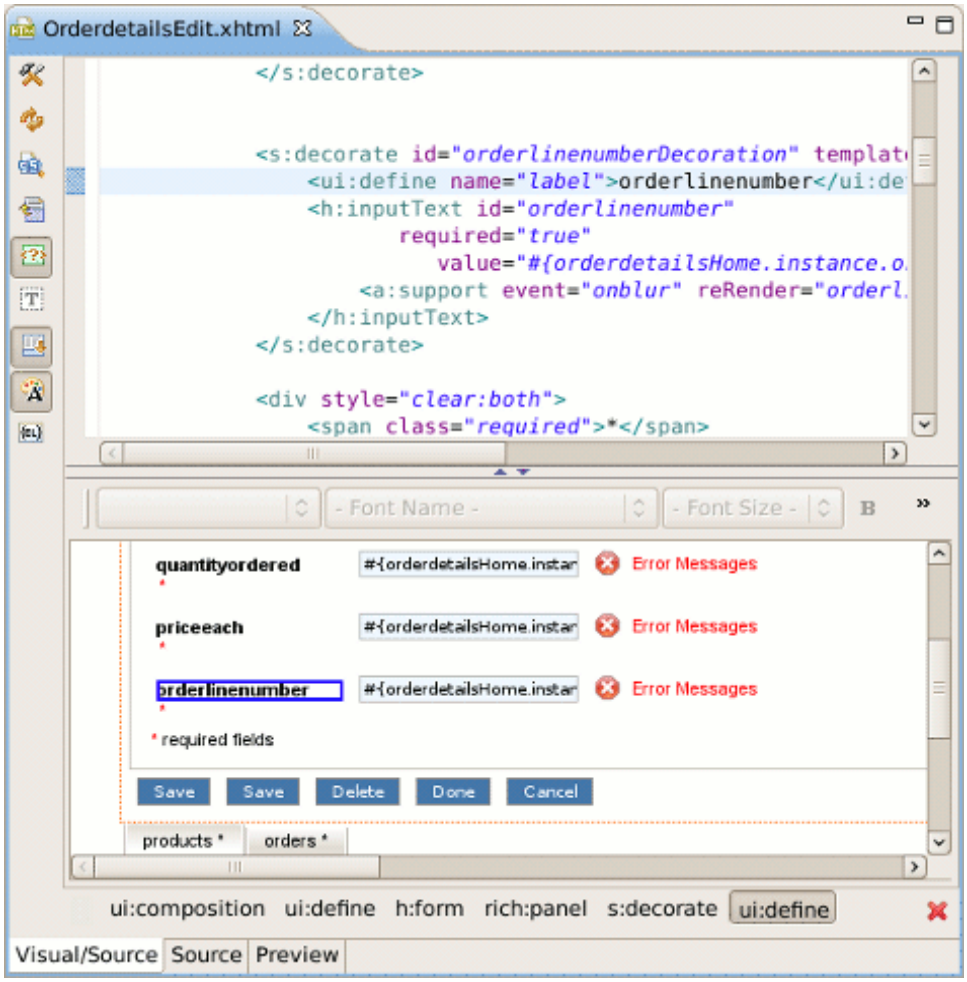

Figure 6.1. Form Fields Editing

Also, replace the QTY Ordered input field with a inputNumberSlider. You can use the JBoss Developer Studio palette or right click on the form and insert the RichFaces component.

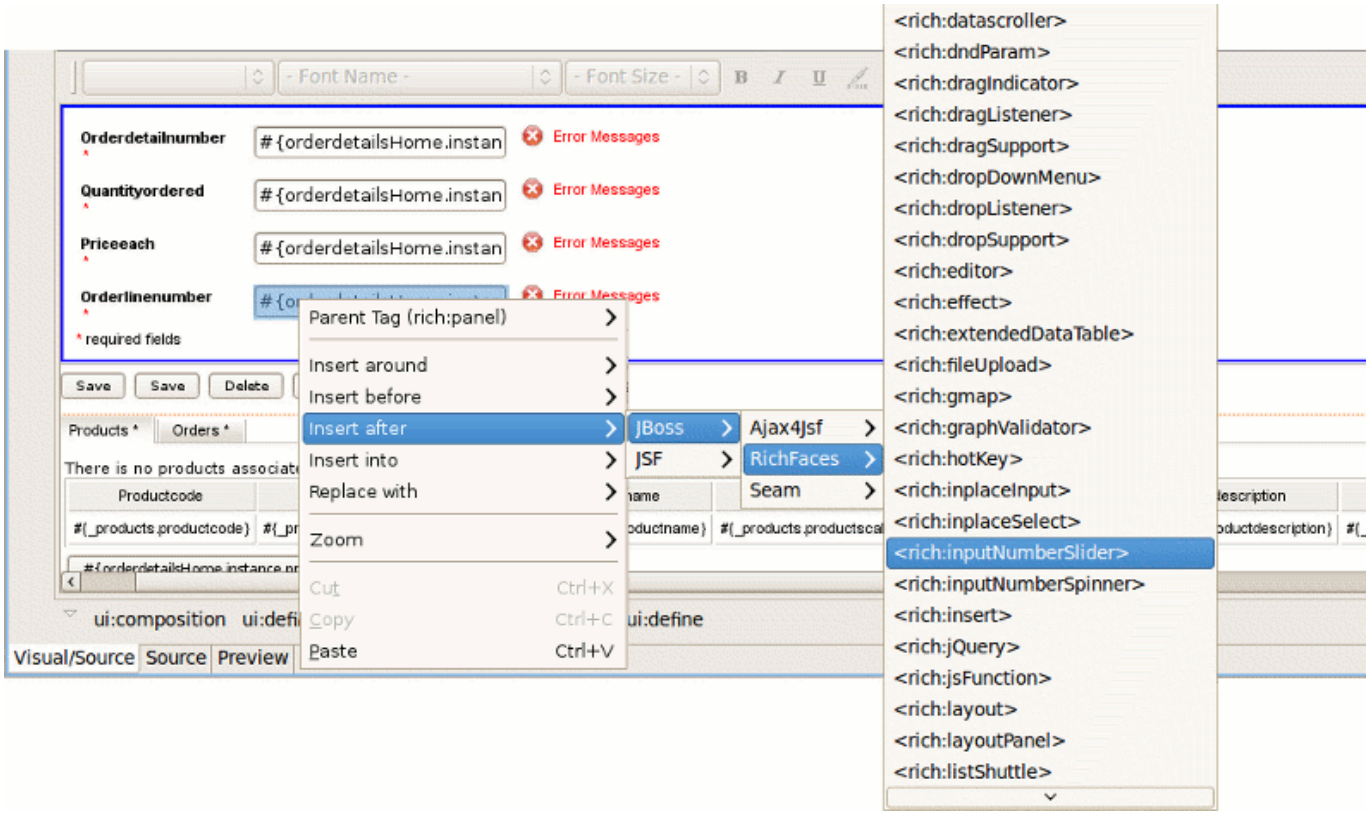

Figure 6.2. Insert RichFaces Component from Context Menu

One the last option is to use the source view and manually copy the inputNumberSlider markup listed below:

```
<rich:inputNumberSlider id="quantityOrdered" required="true"
value="#{orderdetailsHome.instance.quantityordered}"/>
```

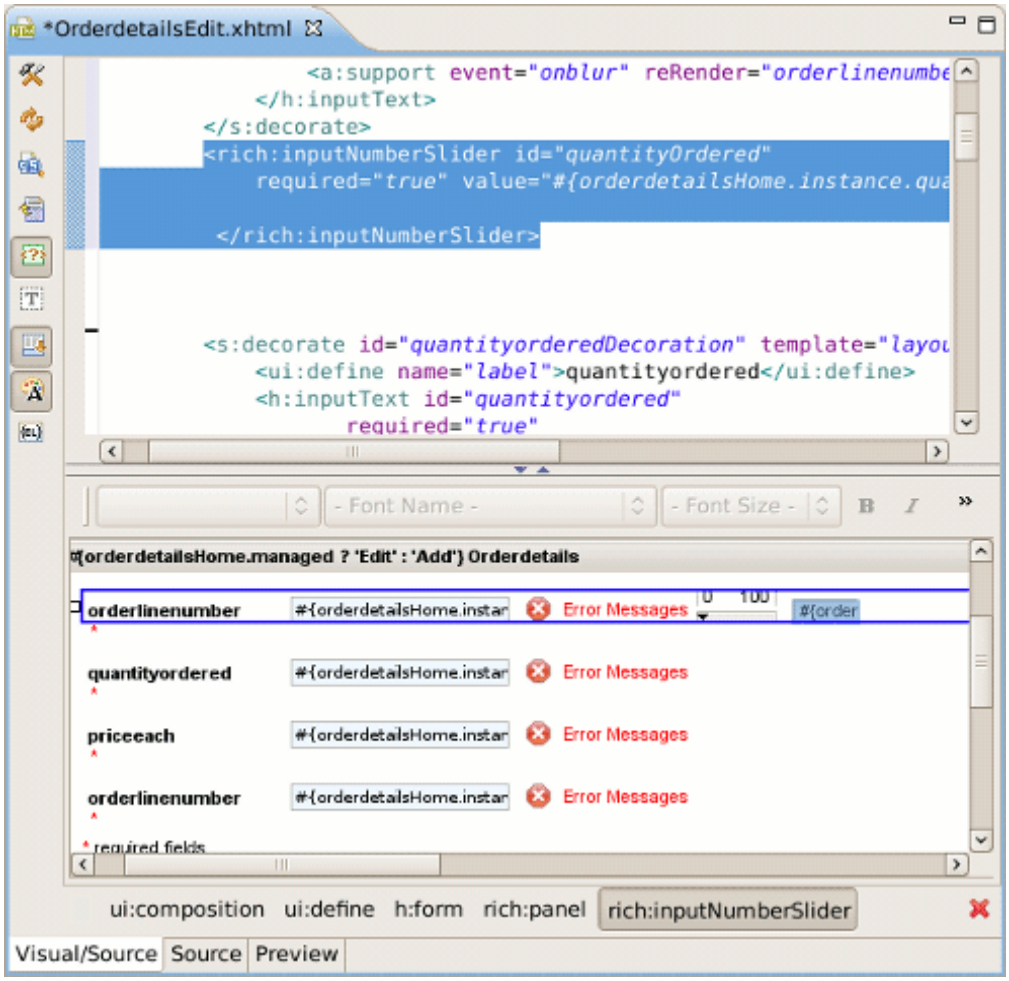

Figure 6.3. Manually copying Source Code

The end result is an edit page that has better form labels and a new RichFaces control.

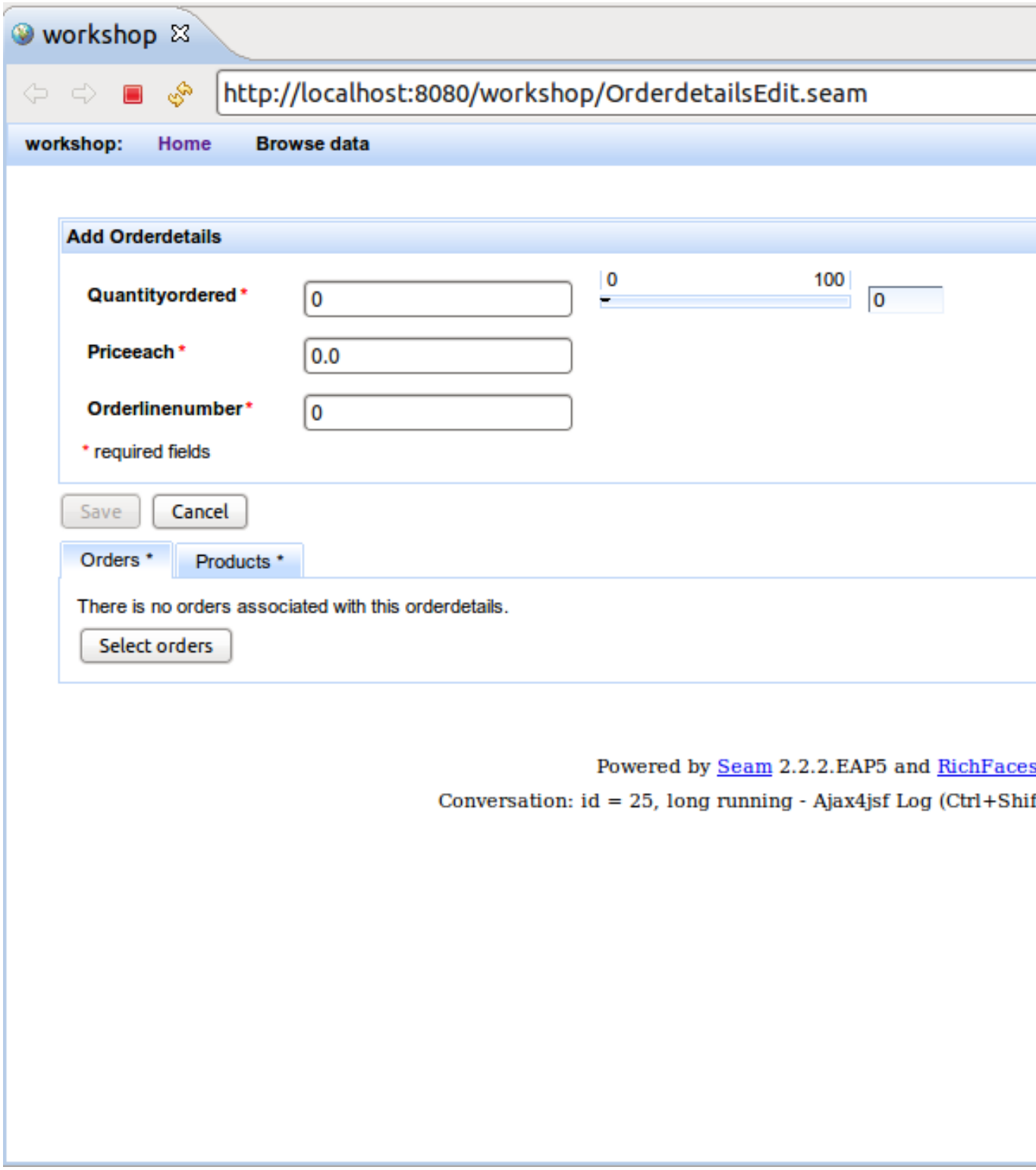

Figure 6.4. The Result Page

Congratulations! You have completed the JBoss Developer Studio lab.