

The NetBeans E-commerce Tutorial - Securing the Application

Tutorial Contents

1. [Introduction](#)
2. [Designing the Application](#)
3. [Setting up the Development Environment](#)
4. [Designing the Data Model](#)
5. [Preparing the Page Views and Controller Servlet](#)
6. [Connecting the Application to the Database](#)
7. [Adding Entity Classes and Session Beans](#)
8. [Managing Sessions](#)
9. [Integrating Transactional Business Logic](#)
10. [Adding Language Support](#)
11. **Securing the Application**
 - [Examining the Project Snapshot](#)
 - [Setting up Form-Based Authentication](#)
 - [Creating Users, Groups and Roles](#)
 - [Configuring Secure Data Transport](#)
 - [See Also](#)
12. [Testing and Profiling](#)
13. [Conclusion](#)

This tutorial unit focuses on web application security. When securing web applications, there are two primary concerns that need to be addressed:

1. Preventing unauthorized users from gaining access to protected content.
2. Preventing protected content from being read while it is being transmitted.

The first concern, *access control*, is typically a two-step process that involves (1) determining whether a user is who he or she claims to be (i.e., *authentication*), and then (2) either granting or denying the user access to the requested resource (i.e., *authorization*). A simple and common way to implement access control for web applications is with a login form that enables the server to compare user credentials with a pre-existing list of authenticated users.

The second concern, protecting data while it is in transit, typically involves using Transport Layer Security (TLS), or its predecessor, Secure Sockets Layer (SSL), in order to encrypt any data communicated between the client and server.

Upon reviewing the Affable Bean staff's [list of requirements](#), we'll need to secure the application in the following ways:

- Set up a login form for the administration console that enables staff members access to the console's services, and blocks unauthorized users.
- Configure secure data transport for both the customer checkout process, and for any data transmitted to and from the administration console.



In order to implement the above, we'll take advantage of NetBeans' visual editor for the `web.xml` deployment descriptor. We'll also work in the GlassFish Administration Console to configure a "user group" that corresponds to Affable Bean staff members, and verify SSL support.

You can view a live demo of the application that you build in this tutorial: [NetBeans E-commerce Tutorial Demo Application](#).

Software or Resource	Version Required
NetBeans IDE	Java bundle, 6.8 or 6.9
Java Development Kit (JDK)	version 6
GlassFish server	v3 or Open Source Edition 3.0.1
MySQL database server	version 5.1
AffableBean project	snapshot 10



Notes:

- The NetBeans IDE requires the Java Development Kit (JDK) to run properly. If you do not have any of the resources listed above, the JDK should be the first item that you download and install.
- The NetBeans IDE Java Bundle includes Java Web and EE technologies, which are required for the application you build in this tutorial.

- The NetBeans IDE Java Bundle also includes the GlassFish server, which you require for this tutorial. You could [download the GlassFish server independently](#), but the version provided with the NetBeans download has the added benefit of being automatically registered with the IDE.
- You can follow this tutorial unit without having completed previous units. To do so, see the [setup instructions](#), which describe how to prepare the database and establish connectivity between the IDE, GlassFish, and MySQL.
- Java EE security is an expansive topic that spans well beyond the scope of this tutorial unit. In order to fully appreciate the range of implementation options that are available to you, refer to the [Java EE 6 Tutorial, Part VII: Security](#). This unit provides ample references to relevant sub-sections within the Java EE Tutorial.

Examining the Project Snapshot

The beginning state of the snapshot helps to illustrate the need for security in the application.

1. Open the [project snapshot](#) for this tutorial unit in the IDE. Click the Open Project () button and use the wizard to navigate to the location on your computer where you downloaded the project.
2. Run the project () to ensure that it is properly configured with your database and application server.

If you receive an error when running the project, revisit the [setup instructions](#), which describe how to prepare the database and establish connectivity between the IDE, GlassFish, and MySQL.

3. Test the application's functionality in your browser. This snapshot provides an implementation of the administration console, as specified in the [customer requirements](#). To examine the administration console, enter the following URL in your browser:


`http://localhost:8080/AffableBean/admin/`



The administration console enables you to view all customers and orders contained in the database. When you click either of the links in the left panel, the page will update to display a table listing customers or orders, depending on your choice. (The 'log out' link currently does not "log out" an authenticated user.)

Note: The customers and orders that you see displayed in the administration console are dependent on the data

stored in your database. You can create new records by stepping through the checkout process in the website. Alternatively, you can run the [affablebean_sample_data.sql](#) script on your affablebean database to have your data correspond to the records displayed in the following screenshots. (If you need help with this task, refer to step 2 in the [setup instructions](#).)



the affable bean

admin console

[view all customers](#)
[view all orders](#)
[log out](#)

customer id	name	email	phone
1	Charlie Pace	c.pace@youareeverybody.com	605434778
2	MC Hammer	hammer@hammertime.com	226884562
3	Karel Gott	gott@karelgott.com	224517995
4	Helena Vondráčková	h.vondrackova@seznam.cz	224517995
5	Sawyer Ford	sawyer.ford@gmail.com	204888845
6	Dalibor Janda	dalibor@dalibor.cz	728331184
7	Richard Genzer	r.genzer@nova.cz	737610775
8	Iveta Bartošová	i.bartosova@volny.cz	734556133
9	Jin-Soo Kwon	jin.kwon@hotmail.kr	606338909
10	Benjamin Linus	b.linus@lost.com	222756448
11	Leoš Mareš	mares@ferrari.it	608995383
12	John Locke	maninblack@lostpedia.com	413443727
13	Lucie Bílá	lucie@jampadampa.cz	733556813
14	Sayid Jarrah	sayid@gmail.com	602680793
15	Hugo Reyes	hurley@mrcluck.com	605449336

You can view details for each customer record by hovering your mouse and selecting an individual record.

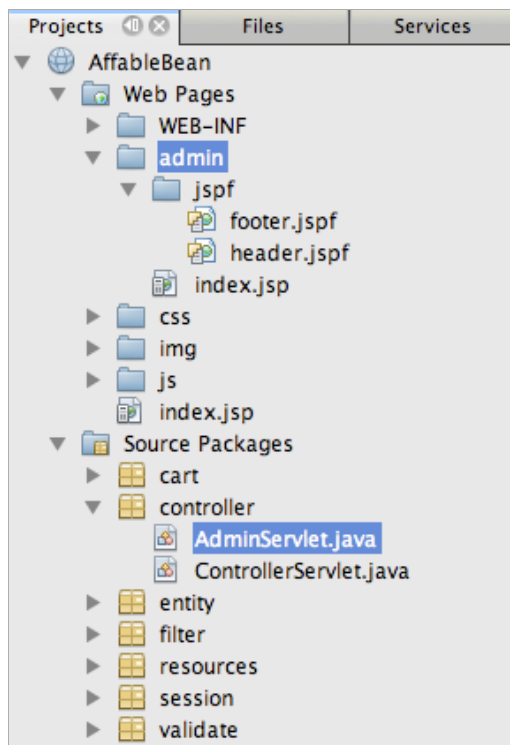
customers			
customer id	name	email	phone
1	Charlie Pace	c.pace@youareeverybody.com	605434778
2	MC Hammer	hammer@hammertime.com	226884562
3	Karel Gott	gott@karelgott.com	224517995
4	Helena Vondráčková	h.vondrackova@seznam.cz	224517995

Likewise, you can view an order summary for each customer either by selecting an order from the administration console's "orders" table, or by clicking the "view order summary" link in a "customer details" display.

customer details	
customer id:	3
name:	Karel Gott
email:	gott@karelgott.com
phone:	224517995
address:	Kostelni 83
city region:	7
credit card number:	3311332222444411
view order summary →	

Naturally, none of this information should be available to an anonymous site visitor. In the coming steps, you'll create login and error pages, so that when an unauthenticated user attempts to access the administration console, he or she will be directed to the login page. Upon successful login, the user is then redirected to the administration console's menu; upon login failure, the error page is displayed.

4. Examine the project snapshot in the Projects window.



This implementation of the administration console primarily relies on the following project resources:

- An **admin** directory within the project's webroot, which contains all page view files.
- An **AdminServlet**, contained in the **controller** package, which forwards requests to page views within the **admin** directory.

Also, the following files have been modified from the previous snapshot:

- **WEB-INF/web.xml**: Contains a new `<jsp-property-group>` that includes the header and footer fragments for page views contained in the **admin** directory.
- **css/affablebean.css**: Includes new style definitions for elements in the administration console

If you have been following the NetBeans E-commerce Tutorial sequentially, you'll find that there is nothing contained in the implementation for the administration console which hasn't already been covered in previous units. Essentially, the **AdminServlet** processes requests from the **admin/index.jsp** page, EJBs and entity classes are employed to retrieve information from the database, and the information is then forwarded back to the **admin/index.jsp** page to be displayed.

5. In the browser, return to the customer website by clicking the Affable Bean logo in the upper left corner of the web page. Step through the entire [business process flow](#) of the application and note that the checkout process is handled over a non-secure channel.

When customers reach the checkout page, they are expected to submit sensitive personal information in order to complete their orders. Part of your task in this tutorial unit is to ensure that this data is sent over a secure channel. Because the administration console also enables authenticated users to view customers' personal information, it too needs to be configured so that data is sent over the Internet securely.

Setting up Form-Based Authentication

In this section, you set up *form-based authentication* for the `AffableBean` administration console. Form-based authentication enables the server to authenticate users based on the credentials they enter into a login form. With these credentials, the server is able to make a decision on whether to grant the user access to protected resources. In order to implement this, you'll create login and error pages, and will rely on *declarative security* by entering security settings in the application's `web.xml` deployment descriptor.

Before you begin implementing a form-based authentication mechanism for the `AffableBean` application, the following background information is provided to help clarify the security terms relevant to our scenario.

- [Declarative and Programmatic Security](#)
- [Choosing an Authentication Mechanism](#)

Declarative and Programmatic Security

With *declarative security*, you specify all security settings for your application, including authentication requirements, access control, and security roles, using annotations and/or deployment descriptors. In other words, the security for your application is in a form that is external to the application, and relies on the mechanisms provided by the Java EE container for its management.

With *programmatic security*, your classes, entities, servlets, and page views manage security themselves. In this case, security logic is integrated directly into your application, and is used to handle authentication and authorization, and ensure that data is sent over a secure network protocol when necessary.

For the `AffableBean` application, we'll use declarative security by declaring all security information in the `web.xml` deployment descriptor.

For more information on declarative and programmatic security types, see the [Java EE 6 Tutorial: Overview of Web Application Security](#).

Choosing an Authentication Mechanism

An *authentication mechanism* is used to determine how a user gains access to restricted content. The Java EE platform supports various authentication mechanisms, such as *HTTP basic authentication*, *form-based authentication*, and *client authentication*. The authentication mechanism behind our login form will be *form-based authentication*. You'll learn what form-based authentication is when you begin [setting up the login form](#) for the `AffableBean` administration console below.

See the Java EE 6 Tutorial: [Specifying Authentication Mechanisms](#) for further information.

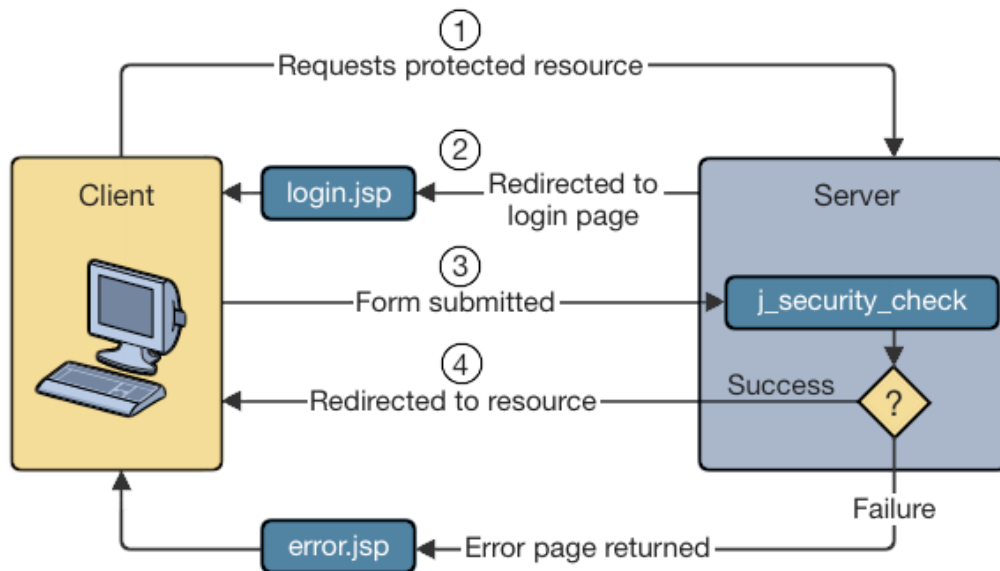
Form-based authentication has the advantage of enabling the developer to design the appearance of the login form so that it better suits the application which it belongs to. Our implementation for the form-based authentication mechanism can be divided into two steps. Begin by creating page views for the required login form and error message. Then add entries to the `web.xml` deployment descriptor to inform the servlet container that the application requires form-based authentication for access to the resources that comprise the administration console.

1. [Create Pages for Login and Login Failure](#)
2. [Add Security Entries to the Deployment Descriptor](#)

Create Pages for Login and Login Failure

In form-based authentication, the process of authentication and authorization is shown in the following four steps:

1. The client sends a request to the server for a protected resource.
2. The server recognizes that a protected resource has been requested, and returns the login page to the client.
3. The client sends username and password credentials using the provided form.
4. The server processes the credentials, and if an authorized user is identified the protected resource is returned, otherwise the error page is returned.



For more information on form-based authentication, see the Java EE 6 Tutorial: [Form-Based Authentication](#).

The `j_security_check` keyword represents the destination in the servlet container that handles authentication and authorization. When implementing the HTML login form, you apply it as the value for the form's `action` attribute. You also apply the `"j_username"` and `"j_password"` keywords, as in the following template:

```
<form action="j_security_check" method=post>

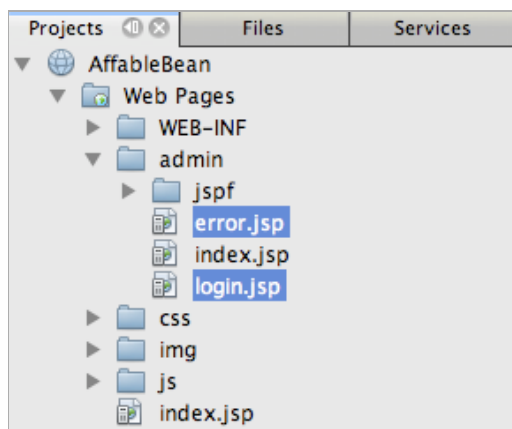
  <p>username: <input type="text" name="j_username"></p>

  <p>password: <input type="password" name="j_password"></p>

  <p><input type="submit" value="submit"></p>
</form>
```

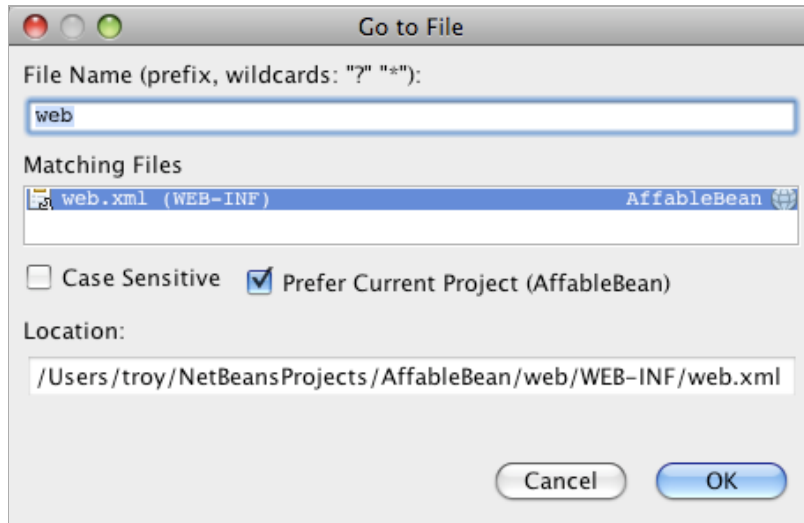
Perform the following steps.

1. In the Projects window, right-click the `admin` folder node and choose **New > JSP**.
2. Name the file `login`, then click **Finish**. The new `login.jsp` file is created and opens in the editor.
3. Repeat the previous two steps to create a new `error.jsp` file. In the New JSP wizard, name the file `error`. When you finish, you'll have two new files listed in the Projects window.



4. Open the project's web deployment descriptor. Press **Alt-Shift-O** (**Ctrl-Shift-O** on Mac) and in the Go to File dialog, type

'web', then click OK.



5. In the editor, scroll to the bottom of the `web.xml` file and note the `<jsp-property-group>` entry created for JSP pages in the administration console. Add the new login and error JSP pages as `<url-pattern>` entries. (Changes in **bold**.)

```
<jsp-property-group>
  <description>JSP configuration for the admin console</description>
  <url-pattern>/admin/index.jsp</url-pattern>
  <url-pattern>/admin/login.jsp</url-pattern>
  <url-pattern>/admin/error.jsp</url-pattern>
  <include-prelude>/admin/jspf/header.jspf</include-prelude>
  <include-coda>/admin/jspf/footer.jspf</include-coda>
</jsp-property-group>
```

This step ensures that when these two pages are returned to a client, they will be prepended and appended with the defined `header.jspf` and `footer.jspf` fragments, respectively.

You can equally configure the `<jsp-property-group>` entry from the `web.xml`'s visual editor. Click the Pages tab along the top of the editor, and enter the URL patterns into the respective JSP Property Group.

6. Press `Ctrl-Tab` to switch to the `login.jsp` file in the editor. Delete the entire template contents for the file, then enter the following HTML form.

```
<form action="j_security_check" method=post>
  <div id="loginBox">
    <p><strong>username:</strong>
      <input type="text" size="20" name="j_username"></p>

    <p><strong>password:</strong>
      <input type="password" size="20" name="j_password"></p>

    <p><input type="submit" value="submit"></p>
  </div>
</form>
```

Note that the HTML form is based on the [template provided above](#). Here, you use the `"j_security_check"` keyword as the value for the form's `action` attribute, and the `"j_username"` and `"j_password"` keywords as the values for the `name` attribute of the username and password text fields. The style of the form is implemented by encapsulating the form widgets within a `<div>` element, then defining a set of rules for the `loginBox` ID in `affablebean.css`.

7. Press Ctrl-Tab and switch to the `error.jsp` file in the editor. Delete the entire template contents for the file, then enter the following.

```
<div id="loginBox">

    <p class="error">Invalid username or password.</p>

    <p>Return to <strong><a href="login.jsp">admin login</a></strong>.</p>

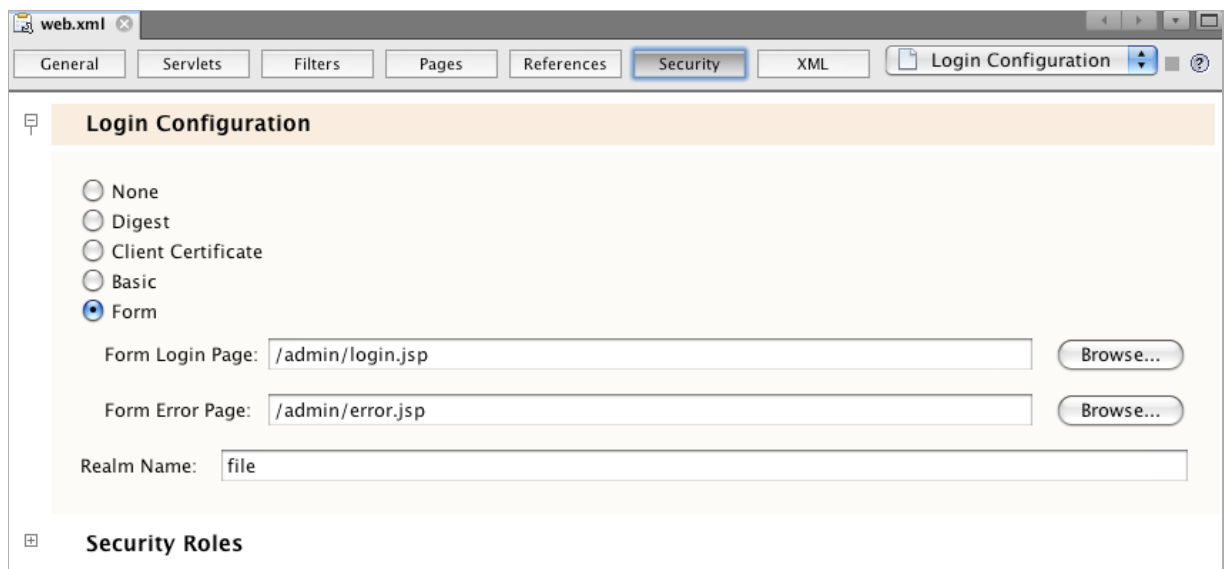
</div>
```

The above content includes a simple message indicating that login has failed, and provides a link that allows the user to return to the login form.

Add Security Entries to the Deployment Descriptor

In order to instruct the servlet container that form-based authentication is to be used, you add entries to the `web.xml` deployment descriptor. This is essentially a three-step process, which can be followed by specifying settings under the three headings in the `web.xml` file's Security tab. These are: (1) Login Configuration, (2) Security Roles, and (3) Security Constraints.

1. Open the project's `web.xml` file in the editor. (If it is already opened, you can press Ctrl-Tab and select it.)
2. Click the Security tab along the top of the editor. The IDE's visual editor enables you to specify security settings under the Security tab.
3. Expand the Login Configuration heading, select Form, then enter the following details:
 - **Form Login Page:** `/admin/login.jsp`
 - **Form Error Page:** `/admin/error.jsp`
 - **Realm Name:** `file`



The screenshot shows the IDE's Security tab for the `web.xml` file. The 'Security' tab is selected, and the 'Login Configuration' section is expanded. Under 'Login Configuration', the 'Form' radio button is selected. The 'Form Login Page' field is set to `/admin/login.jsp`, the 'Form Error Page' field is set to `/admin/error.jsp`, and the 'Realm Name' field is set to `file`. There are 'Browse...' buttons next to the login and error page fields. Below the 'Login Configuration' section, the 'Security Roles' section is visible but collapsed.

4. Click the XML tab along the top of the editor and verify the changes made to the deployment descriptor. The following entry has been added to the bottom of the file:

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>file</realm-name>
    <form-login-config>
        <form-login-page>/admin/login.jsp</form-login-page>
```



```

        <form-error-page>/admin/error.jsp</form-error-page>
    </form-login-config>
</login-config>

```

This entry informs the servlet container that form-based authentication is used, the realm named `file` should be checked for user credentials, and specifies the whereabouts of the login and error pages.

- Click the Security tab again, then expand the Security Roles heading and click Add.
- In the Add Security Role dialog, type in `affableBeanAdmin` for the role name, then click OK. The new role entry is added beneath Security Roles.
- Click the XML tab to examine how the file has been affected. Note that the following entry has been added:

```

<security-role>
  <description/>
  <role-name>affableBeanAdmin</role-name>
</security-role>

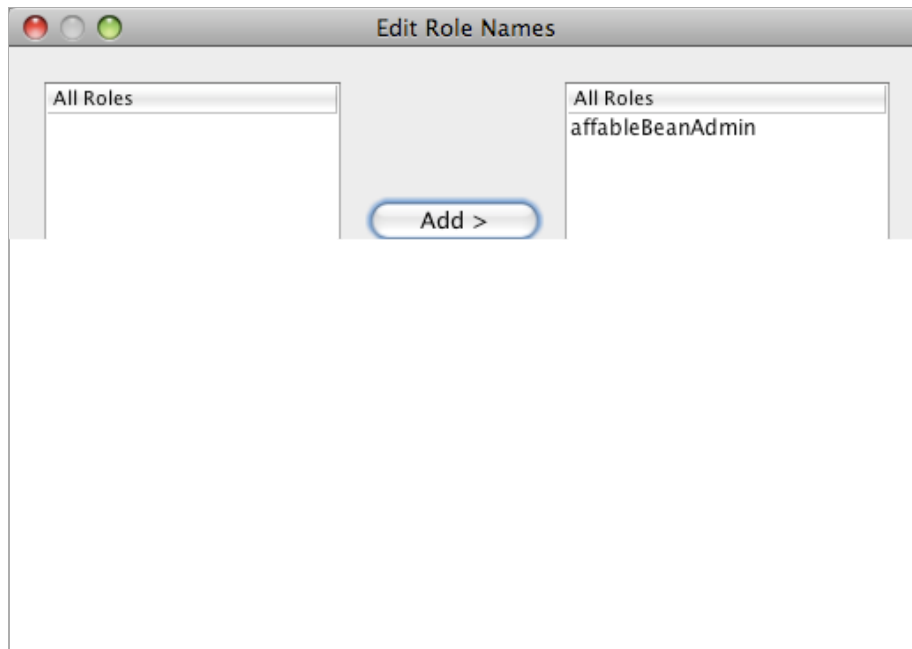
```

Here we've specified the name of a security role used with the application. We'll need to associate this role with the protected resources that define the administration console (under the Security Constraints heading below), and later we'll [create this role on the GlassFish server](#).

- Click the Security tab again, then click the Add Security Constraint button.
- Type in Admin for the Display Name, then under Web Resource Collection click the Add button. Enter the following details, then when you are finished, click OK.

- Resource Name:** Affable Bean Administration
- URL Pattern(s):** /admin/*
- HTTP Method(s):** All HTTP Methods

- Under the new Admin security constraint, select the Enable Authentication Constraint option and click the Edit button next to the Role Name(s) text field.
- In the dialog that displays, select the `affableBeanAdmin` role in the left column, then click Add. The role is moved to the right column.



12. Click OK. The role is added to the Role Name(s) text field.

+

Login Configuration

+

Security Roles

+

Security Constraints

Add Security Constraint

Admin

Remove

Display Name:

Admin

Web Resource Collection:

Name	URL Pattern	HTTP Method	Description
Affable Bean Administration	/admin/*		

Add...

Edit...

Remove

☒ Enable Authentication Constraint

Description:

Role Name(s):

affableBeanAdmin

Edit

☐ Enable User Data Constraint

Description:

Transport Guarantee:

NONE

13. Click the XML tab to examine how the file has been affected. Note that the following entry has been added:


```
<security-constraint>
  <display-name>Admin</display-name>
  <web-resource-collection>
    <web-resource-name>Affable Bean Administration</web-resource-name>
    <description/>
```

```

        <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <description/>
        <role-name>affableBeanAdmin</role-name>
    </auth-constraint>
</security-constraint>

```

In these previous six steps, you've created a security constraint that specifies which resources need to be protected, and identifies the role(s) that are granted access to them. Since the administration console implementation is essentially everything contained within the application's `admin` folder, you use a wildcard (*). Although you've specified that all HTTP methods should be protected, you could have equally selected just GET and POST, since these are the only two that are handled by the `AdminServlet`. As previously mentioned, the `affableBeanAdmin` role that we declared still needs to be created on the GlassFish server.

14. Run the project () to examine how the application now handles access to the administration console.
15. When the application opens in the browser, attempt to access the administration console by entering the following URL into the browser's address bar:

```
http://localhost:8080/AffableBean/admin/
```

When you attempt to access the administration console, the login page is now presented.



the affable bean

admin console

username:

password:

16. Click the 'submit' button to attempt login. You see the error page displayed.



Setting up Users, Groups and Roles

Much of our security implementation is dependent on configuration between the application and the GlassFish server we are using. This involves setting up *users*, *groups*, and *roles* between the two, and using one of the preconfigured security policy domains, or *realms*, on the server. Start by reading some background information relevant to our scenario, then proceed by configuring users, groups and roles between the application and the GlassFish server.

- [Understanding Users, Groups, and Roles](#)
- [Understanding Realms on the GlassFish Server](#)

Understanding Users, Groups, and Roles

A *user* is a unique identity recognized by the server. You define users on the server so that it can be able to determine who should have access to protected resources. You can optionally cluster users together into a *group*, which can be understood as a set of authenticated users. In order to specify which users and/or groups have access to protected resources, you create *roles*. As stated in the Java EE 6 Tutorial,

A role is an abstract name for the permission to access a particular set of resources in an application. A role can be compared to a key that can open a lock. Many people might have a copy of the key. The lock doesn't care who you are, only that you have the right key.

The role that a user or group is assigned to is what specifically allows the server to determine whether protected resources can be accessed. Users and groups can be assigned to multiple roles. As will be demonstrated below, you accomplish this by defining the role in the application, then mapping it to users and groups on the server.

The relationship between users, groups, and roles, and the process in which you establish them in the application and on the server, is presented in the following diagram.

For more information on groups, users, and roles, see [Working with Realms, Users, Groups, and Roles](#) in the Java EE 6 Tutorial.

Understanding Realms on the GlassFish Server

When you define users and groups on the server, you do so by entering details into a security policy domain, otherwise known as a *realm*. A realm protects user credentials (e.g., user names and passwords) through an authentication scheme. For example, user credentials can be stored in a local text file, or maintained in a certificate database.

The GlassFish server provides three preconfigured realms by default. These are the `file`, `admin-realm`, and `certificate` realms. Briefly, the `file` realm stores user credentials in a local text file named `keyfile`. The `admin-realm` also stores credentials in a local text file, and is reserved for server administrator users. The `certificate` realm, the server stores user credentials in a certificate database.


When defining users, groups and roles for the `AffableBean` administration console, we'll use the server's preconfigured `file` realm.

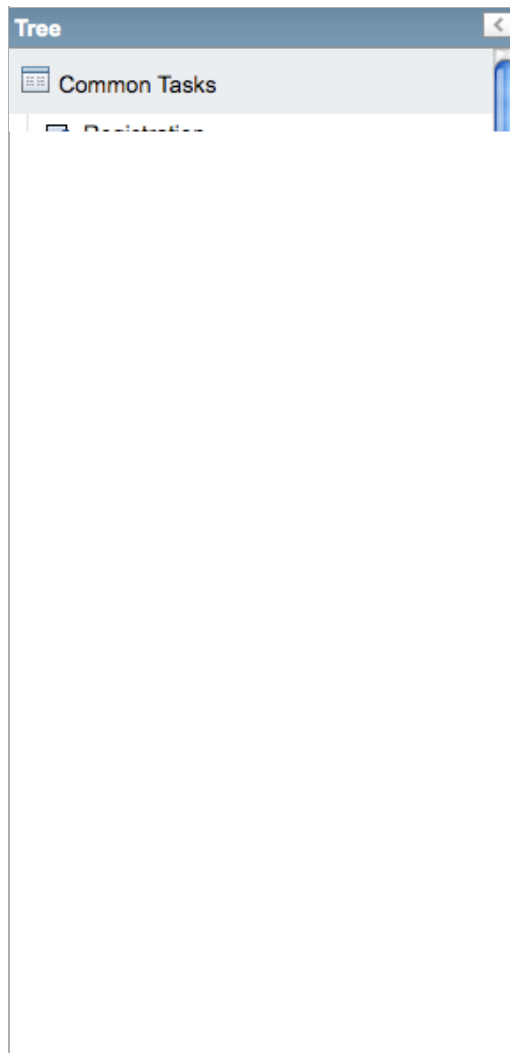
In order to set up users, groups and roles to satisfy the form-based authentication mechanism we've created, perform the following three steps corresponding to the [diagram above](#).

1. [Create Users and/or Groups on the Server](#)
2. [Define Roles in the Application](#)
3. [Map Roles to Users and/or Groups](#)

Create Users and/or Groups on the Server

In this step, we'll use the GlassFish Administration Console to create a user named `nbuser` within the preexisting `file` security realm. We'll also assign the new `nbuser` to a *group* that we'll create called `affableBeanAdmin`.

1. Open the Services window (Ctrl-5; ⌘-5 on Mac) and expand the Servers node so that the GlassFish server node is visible.
2. Ensure that the GlassFish server is running. If the server is running, a small green arrow is displayed next to the GlassFish icon (). If you need to start it, right-click the server node and choose Start.
3. Right-click the GlassFish server node and choose View Admin Console. The login form for the GlassFish Administration Console opens in a browser.
4. Log into the Administration Console by typing `admin / adminadmin` for the username / password.
5. In the Tree which displays in the left column of the Administration Console, expand the Configuration > Security > Realms nodes, then click the `file` realm.



6. In the main panel of the GlassFish Administration Console, under Edit Realm, click the Manage Users button.
7. Under File Users, click the New button.
8. Under New File Realm User, enter the following details:
 - **User ID:** nbuser
 - **Group List:** affableBeanAdmin
 - **New Password:** secret
 - **Confirm New Password:** secret

Here, we are creating a user for the `file` security realm, which we've randomly named `nbuser`. We have also assigned the new user to a randomly named `affableBeanAdmin` group. Remember the `secret` password you set, as you will require it to later log into the `AffableBean` administration console.

9. Click OK. The new `nbuser` user is now listed under File Users in the GlassFish Administration Console.

Optionally close the browser window for the GlassFish Administration Console, or leave it open for the time being. You will need to return to the Administration Console in the [Map Roles to Users and/or Groups](#) step below.

Define Roles in the Application

By "defining roles in the application," you specify which roles have access to EJB session beans, servlets, and/or specific methods that they contain. You can accomplish this declaratively by creating entries in the deployment descriptor, or using annotations. For the `AffableBean` administration console, we've actually already completed this step when we [added the](#)

affableBeanAdmin [role to the security constraint](#) that we created when implementing form-based authentication. However, in more complicated scenarios you may have multiple roles, each with varying degrees of access. In such cases, implementation requires a more fine-grained access control.

The Java EE 6 API includes various security annotations that you can use in place of the XML entries you add to deployment descriptors. The availability of annotations primarily aims to offer ease of development and flexibility when coding. One common method is to use annotations within classes, but override them when necessary using deployment descriptors.

- [Using Security Annotations in Servlets](#)
- [Using Security Annotations in EJBs](#)

Using Security Annotations in Servlets

The following table lists some of the annotations available to you when applying roles to servlets.

Servlet 3.0 Security Annotations (specified in JSR 315)

<code>@ServletSecurity</code>	Used to specify security constraints to be enforced by a Servlet container on HTTP protocol messages.
<code>@HttpConstraint</code>	Used within the <code>ServletSecurity</code> annotation to represent the security constraints to be applied to all HTTP protocol methods.

If we wanted to apply the Servlet 3.0 annotations to declare the `affableBeanAdmin` role on the `AdminServlet`, we could do so as follows. (Changes in **bold**.)

```
@WebServlet(name = "AdminServlet",
            urlPatterns = {"/admin/",
                          "/admin/viewOrders",
                          "/admin/viewCustomers",
                          "/admin/customerRecord",
                          "/admin/orderRecord",
                          "/admin/logout"})
@ServletSecurity( @HttpConstraint(rolesAllowed = {"affableBeanAdmin"}) )
public class AdminServlet extends HttpServlet { ... }
```

In this case, we could then remove the corresponding entry in the `web.xml` deployment descriptor. (Removed content displayed as ~~strike-through~~ text.)

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>file</realm-name>
  <form-login-config>
    <form-login-page>/admin/login.jsp</form-login-page>
    <form-error-page>/admin/error.jsp</form-error-page>
  </form-login-config>
</login-config>

<del>
  <security-constraint>
    <display-name>Admin</display-name>
    <web-resource-collection>
      <web-resource-name>Affable Bean Administration</web-resource-name>
      <description/>
      <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <description/>
      <role-name>affableBeanAdmin</role-name>
    </auth-constraint>
  </security-constraint>
</del>
```

```

<security-role>
  <description/>
  <role-name>affableBeanAdmin</role-name>
</security-role>

```

Using Security Annotations in EJBs

The following table lists some of the annotations available to you when applying roles to EJBs.

EJB Security Annotations (specified in JSR 250)

<code>@DeclareRoles</code>	Used by application to declare roles. It can be specified on a class.
<code>@RolesAllowed</code>	Specifies the list of roles permitted to access method(s) in an application.

To demonstrate the use of EJB security annotations, we'll apply the `@RolesAllowed` annotation to a method that should only be called when a user has been identified as belonging to the `affableBeanAdmin` role.

1. Reexamine the [snapshot implementation for the AffableBean administration console](#). Note that in the `CustomerOrderFacade` session bean, a new `findByCustomer` method enables the `AdminServlet` to access a specified `Customer`.
2. Open the `CustomerOrderFacade` bean in the editor, then add the `@RolesAllowed` annotation to the `findByCustomer` method.

```

@RolesAllowed("affableBeanAdmin")
public CustomerOrder findByCustomer(Object customer) { ... }

```

3. Press `Ctrl-Shift-I` (`%-Shift-I` on Mac) to fix imports. An import statement for `javax.annotation.security.RolesAllowed` is added to the top of the class.

The `findByCustomer` method is only called by the `AdminServlet`, which is previously authenticated into the `affableBeanAdmin` role using our implementation of form-based authentication. The use of the `@RolesAllowed` annotation here is not strictly necessary - its application simply guarantees that the method can only be called by a user who has been authenticated in the `affableBeanAdmin` role.

Map Roles to Users and/or Groups

We have so far accomplished the following:

- Defined the `affableBeanAdmin` role for our form-based authentication mechanism (either in the `web.xml` deployment descriptor, or as an annotation in the `AdminServlet`).
- Created a user named `nbuser` on the GlassFish server, and associated it with a group named `affableBeanAdmin`.

It is no coincidence that the group and role names are the same. While it is not necessary that these names be identical, this makes sense if we are only creating one-to-one matching between roles and groups. In more complicated scenarios, you can map users and groups to multiple roles providing access to different resources. In such cases, you would give unique names to groups and roles.

In order to map the `affableBeanAdmin` role to the `affableBeanAdmin` group, you have a choice of performing one of two actions. You can either create a `<security-role-mapping>` entry in GlassFish' `sun-web.xml` deployment descriptor. (In the Projects window, `sun-web.xml` is located within the project's Configuration Files). This would look as follows:


```

<security-role-mapping>
  <role-name>affableBeanAdmin</role-name>
  <group-name>affableBeanAdmin</group-name>
</security-role-mapping>

```


This action explicitly maps the `affableBeanAdmin` role to the `affableBeanAdmin` group. Otherwise, you can enable GlassFish' Default Principal To Role Mapping service so that roles are automatically assigned to groups of the same name.


The following steps demonstrate how to enable the Default Principal To Role Mapping service in the GlassFish Administration Console.

1. Open the Services window (Ctrl-5; ⌘-5 on Mac) and expand the Servers node so that the GlassFish server node is visible.
2. Ensure that the GlassFish server is running. If the server is running, a small green arrow is displayed next to the GlassFish icon (). If you need to start it, right-click the server node and choose Start.
3. Right-click the GlassFish server node and choose View Admin Console. The login form for the GlassFish Administration Console opens in a browser.
4. Log into the Administration Console by typing `admin / adminadmin` for the username / password.
5. In the Tree which displays in the left column of the Administration Console, expand the Configuration node, then click the Security node.
6. In the main panel of the Administration Console, select the Default Principal To Role Mapping option.

The Java EE 6 Tutorial defines the term *principal* as, "An entity that can be authenticated by an authentication protocol in a security service that is deployed in an enterprise. A principal is identified by using a principal name and authenticated by using authentication data." See [Working with Realms, Users, Groups, and Roles: Some Other Terminology](#) for more information.

7. Click the Save button.

At this stage, you have taken the necessary steps to enable you to log into the `AffableBean` administration console using the `nbuser / secret` username / password combination that you set earlier.

8. Run the project (). When the application opens in the browser, attempt to access the administration console by entering the following URL into the browser's address bar:

```
http://localhost:8080/AffableBean/admin/
```

9. When the login page displays, enter the username and password you set earlier in the GlassFish Administration Console (`nbuser / secret`), then click 'submit'.

Using form-based authentication, the server authenticates the client using the username and password credentials sent from the form. Because the `nbuser` belongs to the `affableBeanAdmin` group, and that group is associated with the `affableBeanAdmin` role, access is granted to the administration console.

10. Click the 'log out' link provided in the administration console. The `nbuser` is logged out of the administration console, and you are returned to the login page.

The `AdminServlet` handles the `"/logout"` URL pattern by invalidating the user session:

```
// if logout is requested
if (userPath.equals("/admin/logout")) {
    session = request.getSession();
    session.invalidate(); // terminate session
    response.sendRedirect("/AffableBean/admin/");
    return;
}
```

Calling `invalidate()` terminates the user session. As a consequence, the authenticated user is dissociated from the active session and would need to login in again in order to access protected resources.

Configuring Secure Data Transport

There are two instances in the `AffableBean` application that require a secure connection when data is transmitted over the Internet. The first is when a user initiates the checkout process. On the checkout page, a user must fill in his or her personal details to complete an order. This sensitive data must be protected while it is sent to the server. The second instance occurs when a user logs into the administration console, as the console is used to access sensitive data, i.e., customer and order details.

Secure data transport is typically implemented using Transport Layer Security (TLS) or Secure Sockets Layer (SSL). HTTP is applied on top of the TLS/SSL protocol to provide both encrypted communication and secure identification of the server. The combination of HTTP with TLS or SSL results in an HTTPS connection, which can readily be identified in a browser's address bar (e.g., **https://**).

The GlassFish server has a secure (HTTPS) service enabled by default. This service uses a self-signed digital certificate, which is adequate for development purposes. Your production server however would require a certificate signed by a trusted third-party Certificate Authority (CA), such as [VeriSign](#) or [Thawte](#).

You can find the generated certificate in: `<gf-install-dir>/glassfish/domains/domain1/config/keystore.jks`

Begin this section by verifying that GlassFish' HTTPS service is enabled. Then configure the application so that a secure HTTPS connection is applied to the checkout process and administration console.

- [Verify HTTPS Support on the Server](#)
- [Configure Secure Connection in the Application](#)

Verify HTTPS Support on the Server

1. Open the Services window (Ctrl-5; ⌘-5 on Mac) and expand the Servers node so that the GlassFish server node is visible.
2. Ensure that the GlassFish server is running. If the server is running, a small green arrow is displayed next to the GlassFish icon (). If you need to start it, right-click the server node and choose Start.
3. Switch to your browser and type the following URL into the browser's address bar:

```
https://localhost:8181/
```

The browser displays a warning, indicating that the server is presenting you with a self-signed certificate. In Firefox for example, the warning looks as follows.


•

4. Enable your browser to accept the self-signed certificate. With Firefox, click the Add Exception button displayed in the warning. The following pane displays, allowing you to view the certificate.

•

Click Confirm Security Exception. A secure connection is established on port 8181, and your local development server, GlassFish, is then able to display the following page.

•

Aside from the HTTPS protocol displayed in the browser's address bar, Firefox indicates that a secure connection is established with the blue background behind `localhost` in the address bar. Also, a lock () icon displays in the lower right corner of the browser. You can click the lock icon for secure pages to review certificate details.

The following optional steps demonstrate how you can identify this security support in the GlassFish Administration Console.

5. Open the GlassFish Administration Console in the browser. (Either type `'http://localhost:4848/'` in your browser, or click the 'go to the Administration Console' link in the GlassFish server's welcome page, as displayed in the [image above](#).)
6. In the Tree which displays in the left column of the Administration Console, expand the Configuration > Network Config nodes, then click the Network Listeners node.

The main panel displays the three network listeners enabled by default on the GlassFish server. `http-listener-2`, which has been configured to listen over port 8181, is the network listener used for secure connections.

For more information on network listeners, see the Oracle GlassFish Server 3.0.1 Administration Guide: [About HTTP Network Listeners](#).

- Under the Name column, click the link for `http-listener-2`. In the main panel, note that the Security checkbox is selected.

- Click the SSL tab. Note that TLS is selected. In the lower portion of the SSL panel, you see the Cipher Suites that are available for the connection. As stated in the Oracle GlassFish Server 3.0.1 Administration Guide, [Chapter 11: Administering System Security](#),

A cipher is a cryptographic algorithm used for encryption or decryption. SSL and TLS protocols support a variety of ciphers used to authenticate the server and client to each other, transmit certificates, and establish session keys. Some ciphers are stronger and more secure than others. Clients and servers can support different cipher suites. During a secure connection, the client and the server agree to use the strongest cipher that they both have enabled for communication, so it is usually sufficient to enable all ciphers.

At this stage, you have an understanding of how the GlassFish server supports secure connections out-of-the-box. Naturally, you could set up your own network listener, have it listen on a port other than 8181, enable SSL 3 instead of TLS (or both), or generate and sign your own digital certificates using Java's `keytool` management utility. You can find instructions on how to accomplish all of these tasks from the following resources:

- The Java EE 6 Tutorial, [Establishing a Secure Connection Using SSL](#)
- Oracle GlassFish Server 3.0.1 Administration Guide, [Chapter 11: Administering System Security](#)
- Oracle GlassFish Server 3.0.1 Administration Guide, [Chapter 16: Administering Internet Connectivity](#)

Configure Secure Connection in the Application

This example demonstrates how to specify a secure connection using both XML in the web deployment descriptor, as well as Servlet 3.0 annotations directly in a servlet. You begin by creating an `<security-constraint>` entry in `web.xml` for the customer checkout process. Then, to create a secure connection for access to the administration console, you specify a `TransportGuarantee` constraint for the `@HttpConstraint` annotation in the `AdminServlet`.

- Open the project's `web.xml` file in the editor. (If it is already opened, you can press Ctrl-Tab and select it.)
- Click the Security tab along the top of the editor, then click the Add Security Constraint button.
- Type in Checkout for the Display Name, then under Web Resource Collection click the Add button. Enter the following details, then when you are finished, click OK.

- Resource Name:** Checkout
- URL Pattern(s):** /checkout
- HTTP Method(s):** Selected HTTP Methods (GET)

Note: Recall that the `/checkout` URL pattern is handled by the `ControllerServlet`'s `doGet` method, and forwards the user to the checkout page.

- Under the new Checkout security constraint, select the Enable User Data Constraint option, then in the Transport Guarantee drop-down, select `CONFIDENTIAL`.

When you choose `CONFIDENTIAL` as a security constraint, you are instructing the server to encrypt data using TLS/SSL so that it cannot be read while in transit. For more information, see the Java EE 6 Tutorial, [Specifying a](#)

Secure Connection.

- Click the XML tab along the top of the editor. Note that the following `<security-constraint>` entry has been added.

```
<security-constraint>
  <display-name>Checkout</display-name>
  <web-resource-collection>
    <web-resource-name>Checkout</web-resource-name>
    <url-pattern>/checkout</url-pattern>
    <http-method>GET</http-method>
  </web-resource-collection>
  <user-data-constraint>
    <description/>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Configuration for the customer checkout process is now complete. To ensure that a secure connection is applied for access to the administration console, simply specify that any requests handled by the `AdminServlet` are transmitted over a secure channel.

- Open the `AdminServlet`. Press Alt-Shift-O (Ctrl-Shift-O on Mac) and in the Go to File dialog, type 'admin', then click OK.
- Use the `@HttpConstraint` annotation's `transportGuarantee` element to specify a `CONFIDENTIAL` security constraint. Make the following change (in **bold**).

```
@WebServlet(name = "AdminServlet",
            urlPatterns = {"/admin/",
                           "/admin/viewOrders",
                           "/admin/viewCustomers",
                           "/admin/customerRecord",
                           "/admin/orderRecord",
                           "/admin/logout"})
@WebServletSecurity(
    @HttpConstraint(transportGuarantee = TransportGuarantee.CONFIDENTIAL,
                    rolesAllowed = {"affableBeanAdmin"})
)
public class AdminServlet extends HttpServlet { ... }
```

- Press Ctrl-Shift-I (⌘-Shift-I on Mac) to fix imports. An import statement for `javax.servlet.annotation.ServletSecurity.TransportGuarantee` is added to the top of the class.
- Run the project (🟢) to examine the application's behavior in a browser.
- In the browser, step through the `AffableBean` website by selecting a product category and adding several items to your shopping cart. Then click the 'proceed to checkout' button. The website now automatically switches to a secure channel when presenting the checkout page. You see the HTTPS protocol displayed in the browser's address bar, and the port is changed to 8181.

Also, in Firefox, note the lock () icon displayed in the lower right corner of the browser.

- Investigate security for the administration console. Type in the following URL into the browser's address bar:

```
http://localhost:8080/AffableBean/admin/
```

The website now automatically switches to a secure channel when presenting the checkout page. You see the HTTPS protocol displayed in the browser's address bar, and the port is changed to 8181.

Note: You may wonder at this point how it is possible to switch from a secure connection back to a normal, unsecured one. This practice however is not recommended. The [Java EE 6 Tutorial](#) explains as follows:

If you are using sessions, after you switch to SSL you should never accept any further requests for that session that are non-SSL. For example, a shopping site might not use SSL until the checkout page, and then it might switch to using SSL to accept your card number. After switching to SSL, you should stop listening to non-SSL requests for this session. The reason for this practice is that the session ID itself was not encrypted on the earlier communications. This is not so bad when you're only doing your shopping, but after the credit card information is stored in the session, you don't want a bad guy trying to fake the purchase transaction against your credit card. This practice could be easily implemented using a filter.

You have now successfully secured the `AffableBean` application according to the defined customer requirements. You've set up a login form for the administration console to authorize or deny access based on user credentials, and you configured the application and server to create a secure connection for access to the administration console, as well as the customer checkout process.

You can compare your work with the [completed AffableBean project](#). The completed project includes the security implementation demonstrated in this unit, and also provides a basic implementation for web page error customization, such as when a request for a nonexistent resource is made, and the server returns an HTTP 404 'Not Found' error message.

[Send Us Your Feedback](#)

See Also

NetBeans Resources

- [Securing a Web Application](#)
- [Introduction to Java EE Technology](#)
- [Getting Started with Java EE Applications](#)
- [Keyboard Shortcuts & Code Templates Card](#)
- [Java EE & Java Web Learning Trail](#)

External Resources

- [The Java EE 6 Tutorial, Chapter 24: Introduction to Security in the Java EE Platform](#)
- [The Java EE 6 Tutorial, Chapter 25: Getting Started Securing Web Applications](#)
- [The Java EE 6 Tutorial, Chapter 26: Getting Started Securing Enterprise Applications](#)
- [Oracle GlassFish Server 3.0.1 Administration Guide](#)
- [Security Annotations and Authorization in GlassFish and the Java EE 5 SDK](#)
- [Java EE 6: Application Security Enhancements](#)
- [Getting Started with Java EE Security \[RefCard\]](#)
- [HTTP Secure \[Wikipedia\]](#)
- [Public key certificate \[Wikipedia\]](#)