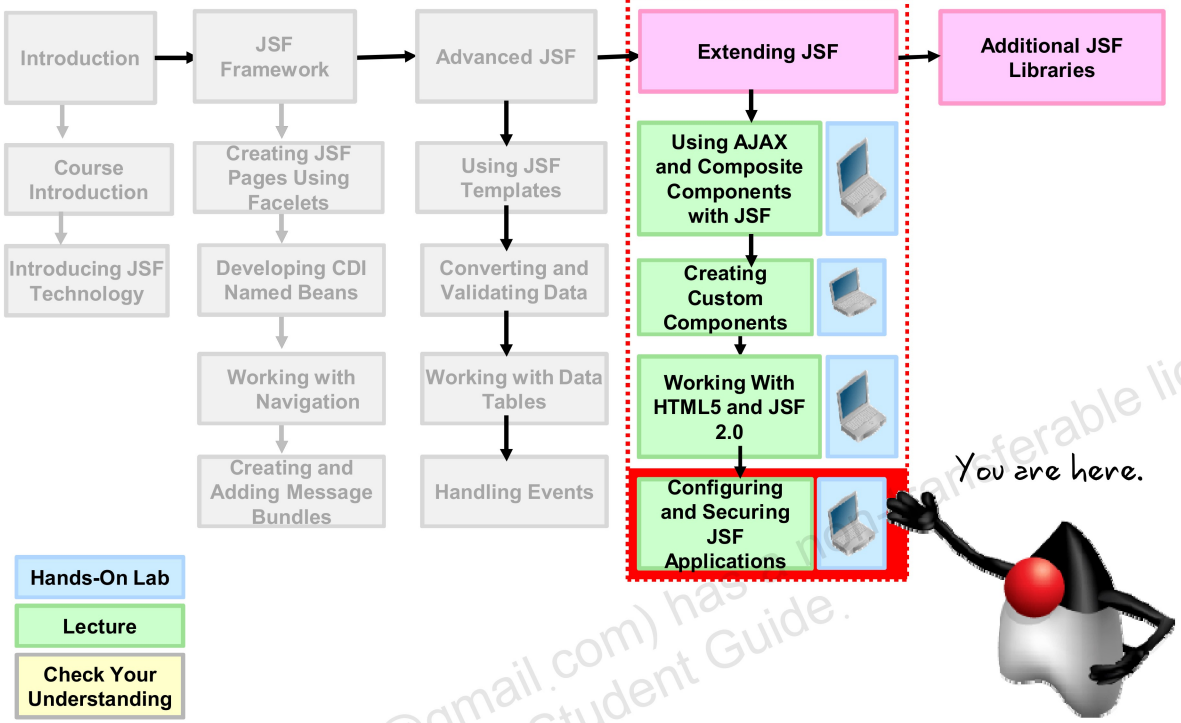
14

Configuring and Securing JSF Applications

Course Road Map



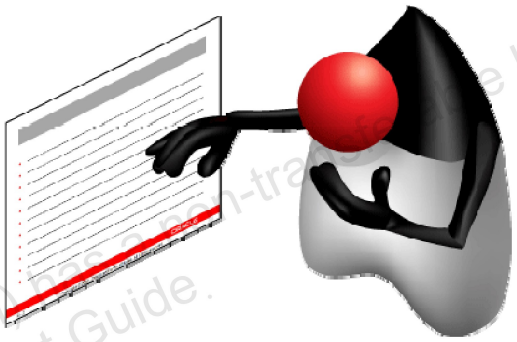
Objectives

After completing this lesson, you should be able to:

- Describe JSF Web application stages
- Configure the state maintenance method
- Describe the application configuration loading process
- Describe container managed security
- Declare user roles and responsibilities
- Configure security for JSF Web Applications
- Use the security API

Topics

- JSF Web application stages
- The state maintenance method
- The application configuration loading process
- The JSF implementation for a web container
- Container-managed security
- User roles and responsibilities
- Role-based security policies
- The security API
- Authentication configuration



Configuring the Project Stage

- Project stage is configured as a context parameter named `javax.faces.PROJECT_STAGE`.

```
1  <context-param>
2     <param-name>javax.faces.PROJECT_STAGE</param-name>
3     <param-value>Development</param-value>
4  </context-param>
```

- Allowed values of the parameters are: Production, Development, UnitTest, SystemTest, and Extension.
  - The Development (default) stage provides more output for better error reporting and debugging.
  - The Production stage provides better performance.
- The project stage parameter can also be configured through JNDI.

Note

The development stage (the default) is the only stage where the `<ui:debug>` tag will work.

The PROJECT_STAGE parameter was introduced in JSF 2.0.

Configuring State-Saving Method

- The state-saving method is configured as a context parameter or named `javax.faces.STATE_SAVING_METHOD`.
- Allowed values are client and server.

```
1   <context-param>
2       <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
3       <param-value>client</param-value>
4    </context-param>
```


Saving application state on the server has the benefit that all the state information is stored on the server side. However, it has the drawback of requiring more server-side resources for maintaining state. If state is saved on the client (default), the state of the view and the UI object model are passed to the client in the form of hidden fields on the page.

JSF 1.2 performed full state saves, whereas JSF 2.0 can perform partial-page deltas.

Configuring State-Saving Method

- The state information that is saved on the client needs to be encrypted.
- The JSF 2.0 reference implementation (RI) automatically does this if the encryption password is specified as an environment resource named `ClientStateSavingPassword`.

```
1   <env-entry>
2      <env-entry-name>ClientStateSavingPassword</env-entry-name>
3      <env-entry-type>java.lang.String</env-entry-type>
4      <env-entry-value>password</env-entry-value>
5   </env-entry>
```


The state information that is saved on the client needs to be encrypted. Otherwise, this can lead to a security vulnerability where the full UI object model can be reconstructed. JSF 2.0 RI automatically does this if the encryption password is specified as an environment resource named `ClientStateSavingPassword`.

Application Configuration Files

- Multiple files can be applied to a JSF application.
- The search order of the files is as follows:
  - `/META-INF/faces-config.xml` in the web application library JAR files
  - `javax.faces.application.CONFIG_FILES` context initialization parameter; specifies one or more comma-delimited paths to configuration files
  - `/WEB-INF/faces-config.xml`

JSF technology provides a portable configuration format (as an XML document) for configuring resources. An application architect creates one or more files, called application configuration resource files, which use this format to register and configure objects and resources, and to define navigation rules. An application configuration resource file is usually called `faces-config.xml`.

The application configuration resource file must be valid against the schema located at `http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd`.

Controlling the Application Configuration Files Order

- The `<absolute-ordering>` and `<ordering>` elements can control the searching order.
- Absolute ordering:

```
1  <faces-config>
2    <name>myJSF</name>
3    <absolute-ordering>
4     <name>A</name>
5     <name>B</name>
6   </absolute-ordering>
7  </faces-config>
```

- Relative ordering:

```
1  <faces-config>
2   <name>myJSF</name>
3   <ordering>
4     <before><name>B</name></before>
5   </ordering>
6  </faces-config>
```

JSF Web Application Structure

A JSF web application using Facelets typically has the following directory structure:

```
$PROJECT_DIR
 +- /[xhtml pages]
 +- /resources
 +- /WEB-INF
        +- /classes
        +- /lib
        +- /web.xml
        +- /faces-config.xml
        +- /sun-web.xml   (if using GlassFish)
```


To run JSF 2.0 applications in containers that support the Servlet 2.5 and JavaServer Pages 2.1 specification (or later), the applications must be packaged in a WAR file as shown in the slide. Applications that conform to these specific requirements will execute across different container implementations.

Installing and Updating the JSF Implementation

- Not all Java EE web containers and application servers have built-in support for JSF 2.0.
- The reference implementation of JSF 2.0 (Mojarra) is hosted at `http://javaserverfaces.java.net`.
  - Mojarra can be configured for most Java EE web containers and application servers.
- Additional JSF implementations are available:
  - Apache MyFaces: `http://myfaces.apache.org`

Quiz

The default stage for a JSF application that provides better error reporting and debugging is:

a. Production
b. SystemTest
c. UnitTest
d. Development

Answer: d

The value is declared in the `faces-config.xml` file through `javax.faces.PROJECT_STAGE`.

Quiz

The disadvantage of using the following state-saving declaration is:

```
<context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>server</param-value>
 </context-param>
```

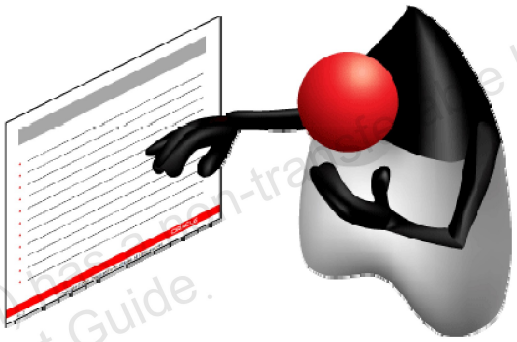
a. Data security
b. Resource usage
c. Performance
d. Code maintenance

Answer: b

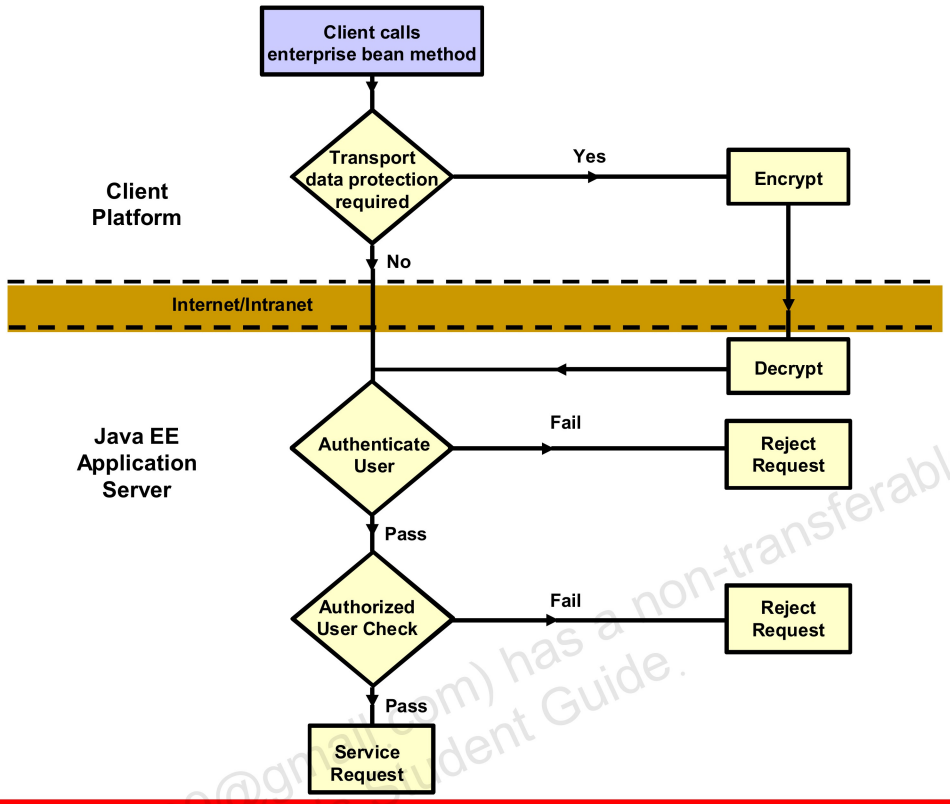
Server-side state management is not the default for this reason.

Topics

- JSF Web application stages
- The state maintenance method
- The application configuration loading process
- The JSF implementation for a web container
- Container-managed security
- User roles and responsibilities
- Role-based security policies
- The security API
- Authentication configuration



Security Requirements of an Application



Most applications have security requirements. The application usually needs to be able to identify the user, decide the operations that the user is allowed to perform, and maintain the confidentiality and integrity of the data that is in transit.

The diagram in the slide illustrates the runtime security interventions performed by the Java EE infrastructure when responding to a method invocation by a client.

Security Interventions

- Transport Data Protection
  - It ensures the confidentiality and integrity of the information that is transported across the network.
  - Systems use a combination of certificate exchange and encryption to achieve this security.
- Caller Identification and Authentication
  - It is the process of verifying the identity of users and ensuring that they are who they say they are.
  - Most systems use password challenges, client certificates, or digests to perform this task.
- Access Control Implementation for Resources
  - It determines who is allowed to do what.
  - Authorization is the process of allocating permissions to authenticated users.

Java EE Platform Security Model

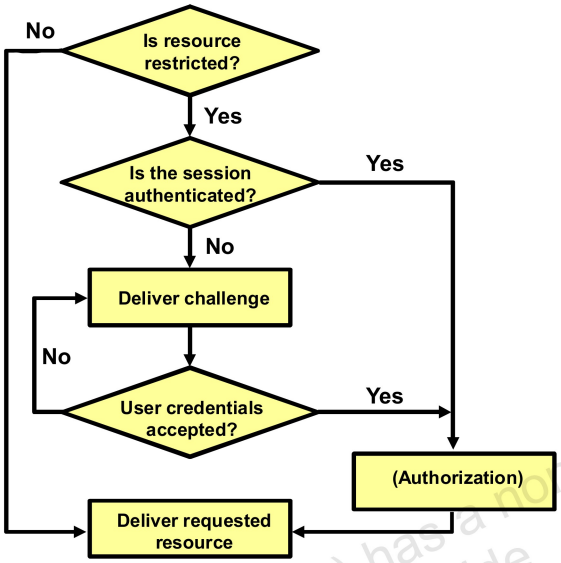
The primary goal of the Java EE platform is to:

- Relieve the application developer from the details of security mechanisms
- Facilitate secure deployment of an application in diverse environments
- Enable the use of the Java EE security model to increase development efficiency and portability
- The Java EE security model:
  - Contains no reference to the real security infrastructure
  - Can use declarative or programmatic authorization controls
  - Maps the security policy to the target security domain
  - Provides end-to-end security

The following list summarizes the characteristics of Java EE security:

- Contains no reference to the real security infrastructure. This makes the application portable.
- Can use declarative or programmatic authorization controls
- Maps the security policy to the target security domain. This is performed in an application server–specific way.
- Provides end-to-end security. End-to-end security means that if security credentials are gathered in one part of the application, they become available to the other parts. For example, if a user is using a web browser to interact with a servlet-based application and the servlet application calls enterprise beans, all the components (servlets and enterprise beans) that are invoked in a particular user interaction see the same user.

Authentication in the Web Tier



The figure in the slide shows the authentication process in the web tier. Web containers typically use lazy authentication. Lazy authentication means that the user is only asked to authenticate if the requested resource, such as a servlet or JSP component, is actually subject to a security restriction.

Authentication completes the following basic sequence:

- If the resource is not restricted, the resource is delivered without challenge.
- If the resource is restricted, the container gets the session ID from the request and checks in the session store to determine whether the user is already authenticated.
- If the user is authenticated, there is no need to authenticate again.
- The container checks whether the user has permission to request the specified resource.
- If the user is not authenticated, instead of delivering the requested resource, the container delivers an authentication challenge. The challenge very often consists of a request to the user to supply a username and password.
- If the username and password are acceptable, the container moves on to the authorization step.

Web-Tier Authentication Methods

- **HTTP Basic:** The web browser prompts the user for a username and password, and supplies this information in the request header.
- **Client Certificate:** The client presents the user's digital certificate in response to a challenge from the server.
- **Form-based:** The developer controls the look and feel of the authentication process by supplying HTML forms.
- **Programmatic:** Java EE 6 includes programmatic servlet security, which adds the ability to force a challenge in-code or to collect authentication information in a custom way.

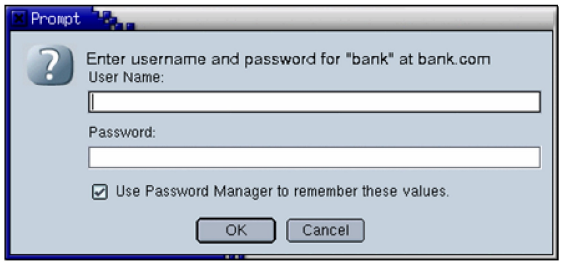
ORACLE

Selecting the Authentication Type

The Java EE specification recognizes three types of authentication in the web tier. Each has its own configuration requirements.

- **HTTP Basic:** Select this authentication method by specifying it in the `web.xml` deployment descriptor. You can also use the IDE to perform this task.
- **Client Certificate:** Select this authentication method by specifying it in the `web.xml`. You must also configure the web container's certificate database and allocate appropriate trust levels, but this process is specific to the particular application server.
- **Form-based:** Select this authentication method by specifying it in the `web.xml`. In addition, you must create a login page and an error page in the HTML or JSP software. These are the pages that are presented by the web container in response to a request that requires authentication. The pages can have any names. Use `web.xml` to specify the names to the container.
- **Programmatic:** The Servlet 3.0 specification (Java EE 6) allows username and password collection using application-specific methods.

Basic Authentication



Configuring authentication:

- The `login-config` element of the `web.xml` deployment descriptor can be used to configure the user authentication:

```
1<login-config>
2   <auth-method>BASIC</auth-method>
3   <realm-name>default</realm-name>
4</login-config>
```


The figure in the slide shows a basic authentication dialog box from a web browser.

It is a common misconception that basic authentication is less secure than the other types of authentication.

In fact, both basic and form-based methods of authentication supply the credentials in plaintext, unless a secure channel is established first.

The real problem with basic authentication is that after the user has logged in, it is difficult to log out.

The reason for this problem is that after the browser has collected the username and password, the browser can supply this information whenever the server asks for them.

So the removal of the user's authenticated status at the application server only has the effect of making the browser resupply the credentials and log the user right back in again.

There are solutions to this problem, but they are not entirely portable. Form-based or programmatic authentication is usually the preferred method when the client is a web browser.

You tell the container which kind of authentication you are using by placing a <login-config> element inside of the <web-app> element in the web.xml file

Note that the implementation of authentication is application server–dependent.

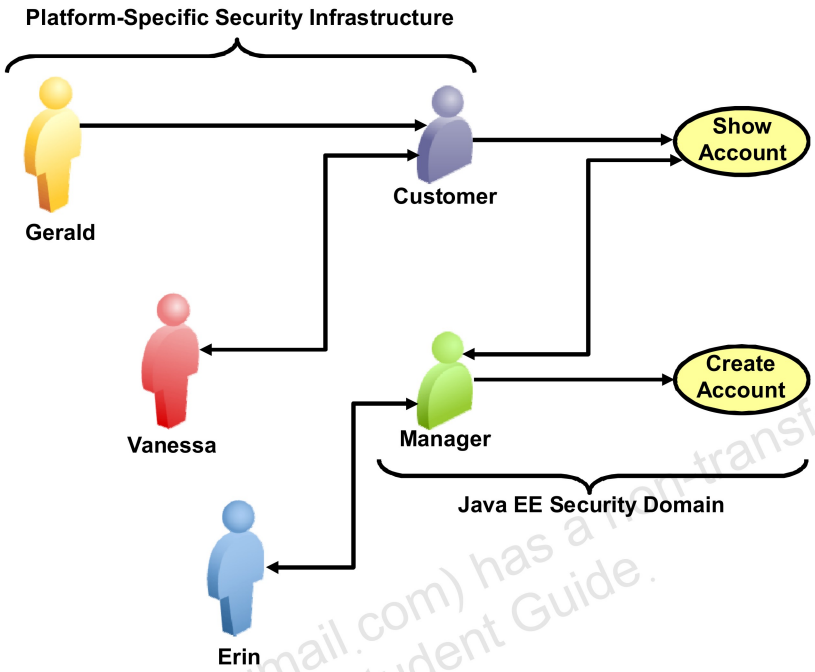
A realm is an opaque string interpreted by the server to identify a data store for resolving username and password information. The implementation of the realm concept is not standard and varies from container to container.

User Roles and Responsibilities

A role is an abstraction of a set of user authorization privileges.

- Users in the same role have broadly similar rights and responsibilities.
- The role structure of the security model in the Java EE platform is flat, not hierarchical.
- Individual users can, and often will, occupy more than one role.
- There is some correspondence between a role and a group in many security infrastructures, but the mapping of real users or groups to roles is platform specific.

Role-Based Java EE Security Model



Roles, Actors, and Use Cases

If you are familiar with UML modeling, you may find it helpful to think of a role as an actor in a use-case model.

- An actor is not a user. Rather, an actor is a collection of users with similar responsibilities.
- Each actor interacts with one or more use cases. A use case is a self-contained unit of capabilities within the application. It follows that the actor must have some access rights over that use case.

In the figure in the slide:

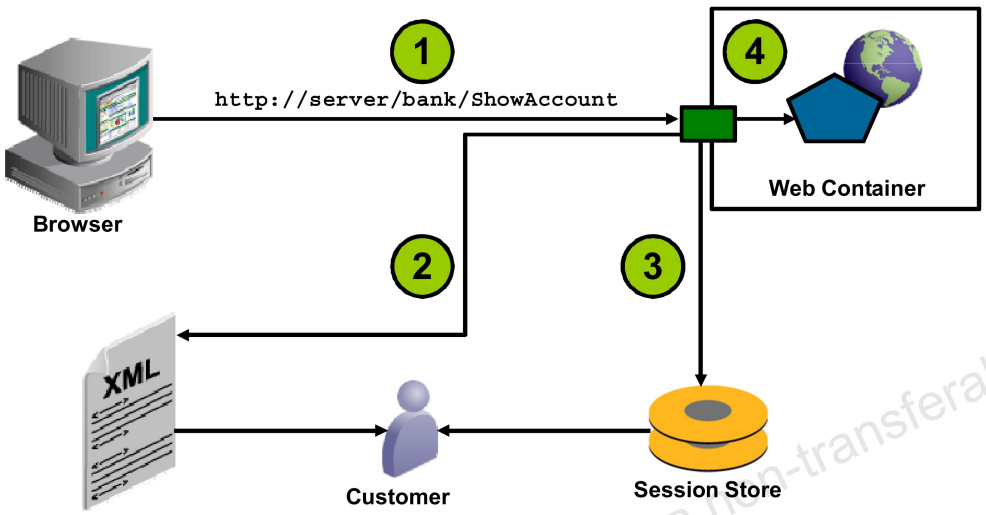
- The Customer role groups the rights and responsibilities that are shared by customers of the application, which includes the right to see the contents of their bank accounts.
- The Manager role groups the rights of managers, which includes the right to create new bank accounts, as well as to view existing bank accounts.

Role Mapping

Mapping application roles to the principal accounts or groups that exist within the application server is done with a vendor deployment descriptor file(s). The `WEB-INF/weblogic.xml` file for web components in WebLogic Server is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app xmlns="http://xmlns.oracle.com/weblogic/weblogic-web-
app" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd
http://xmlns.oracle.com/weblogic/weblogic-web-app
http://xmlns.oracle.com/weblogic/weblogic-web-app/1.0/weblogic-web-
app.xsd">
        <security-role-assignment>
         <role-name>user</role-name>
         <principal-name>customer</principal-name>
     </security-role-assignment>
</weblogic-web-app>
```

Role-Based Authorization in the Web Tier



The figure in the slide shows the procedure that the web container uses to check the authorization of each request.

The authorization procedure in the web tier involves the following steps:

1. The browser makes a request for a specific URL.
2. The URL of the request is matched against the patterns in the deployment descriptor that are subject to security restrictions and to security annotations placed in servlet classes.
3. If the URL is restricted, the server obtains the user's role assignments from the session store and the underlying security infrastructure.
4. The server checks the user's allocated roles against the roles allowed for the resource. If any of the user's roles are in the allowed set, the request is allowed, and the web component is invoked.

Configuring Authorization

The `security-constraint` element of `web.xml` can be used to define which security roles can access which URL pattern–based web resource collection:

```
1   <security-constraint>
2      <web-resource-collection>
3         <web-resource-name>Protected Area</web-resource-name>
4         <url-pattern>/admin/*</url-pattern>
5      </web-resource-collection>
6      <auth-constraint>
7         <role-name>admin</role-name>
8       </auth-constraint>
9    </security-constraint>
```

- The role name `admin` must be defined in a security role element in the `web.xml` deployment descriptor:

```
1   <security-role>
2      <role-name>admin</role-name>
3   </security-role>
```

web.xml Security Constraints

URL patterns can be restricted via `web.xml`.

```
<security-constraint>
    <display-name>MembersOnly</display-name>
    <web-resource-collection>
        <web-resource-name>secret-page</web-resource-name>
        <url-pattern>/faces/membersonly.xhtml</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>member</role-name>
    </auth-constraint>
</security-constraint>
```


Servlet Security Annotations

Java EE 6 and Servlet 3.0 introduced the `@ServletSecurity` annotation that can be used in place of `web.xml` when restricting HTTP access to a servlet.

Using the Security API

The security API is only available to applications that use container-managed authentication. The API provides two basic facilities that you can use to determine identity and role allocation.

	Method Calls to Determine User or Client Identity	Method Calls to Determine User Role Allocations
Web Tier	`getUserPrincipal`	`isUserInRole`
EJB Tier	`getCallerPrincipal`	`isCallerInRole`

Web-Tier Security API

```
String user = request.getUserPrincipal().getName();
String message =
"Welcome to the on-line bank, " + user;

if (request.isUserInRole("manager")) {
//... show manager menu
}
```


The web container makes the security context of the current user available with the `HttpServletRequest` object that is passed to the `service` method.

The code snippet in the slide shows the use of the `getUserPrincipal` method to determine the user or caller identity.

The `getUserPrincipal` method returns a `Principal` object, which encapsulates the user identity. The `Principal.getName` method renders a textual form of that identity, which will be the username if authentication is HTTP basic or form-based.

The following code snippet shows the use of the `isUserInRole` method to determine whether the user has been assigned to the manager role. A servlet might complete this method to construct a display that was appropriate for a particular user.

```
if (request.isUserInRole("manager")) {
  //... show manager menu
}
```

Configuring Authentication in the Web Tier

Web-tier authentication is a complex topic. This course addresses only:

- Selecting the authentication type
- Creating a login page for form-based authentication
- Using Programmatic Logins

Selecting Basic Authentication

The web deployment descriptor can be used to specify the authentication type.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name/>
    </login-config>
</web-app>
```

HTTP Basic is the default method in the Servlet 3.0 specification.

The code example in the slide shows how to select the authentication type in Java EE 6 with a `web.xml` deployment descriptor.

HTTP Error Pages

If a basic authentication challenge fails, the user will be sent a HTTP 403 Forbidden response. You can customize the result.

```
<error-page>
    <error-code>403</error-code>
    <location>/notallowed.xhtml</location>
</error-page>
```


Error Pages

Any HTTP error code can be customized in this fashion such as a 404. Error pages cannot be JSF pages.

Selecting Form Authentication

Form-based authentication requires you to specify two URLs.

```
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>
            /faces/authmethodform.xhtml
         </form-login-page>
         <form-error-page>
            /badlogin.xhtml
         </form-error-page>
     </form-login-config>
</login-config>
```

Creating an HTML Login Page for Form-Based Authentication

Minimal section of HTML:

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

    <-- layout code not shown for clarity -->

    Username:
    <input name="j_username"/><br/>
    Password:
    <input name="j_password" type="password"/><br/>

    <input type="submit" name="submit" value="Log in"/>

</form>
```


The login page can have any appearance or layout, but the names of the form elements must comply with the specification.

You usually create login and error pages that match the look and feel of the rest of the application.

The use of a JSP component or a servlet for the login page allows it to contain dynamic content, if necessary.

The `action` attribute should be `j_security_check`, and the username and password fields should have the names `j_username` and `j_password`, respectively. It is these names that signal to the web container that the authentication data is being supplied.

Programmatic Logins

The most flexible login mechanism is the new Java EE 6 `HttpServletRequest` login method.

```
ExternalContext externalContext =
    FacesContext.getCurrentInstance()
        .getExternalContext();
HttpServletRequest request = (HttpServletRequest)
externalContext.getRequest();
```

If you need a `HttpServletRequest` in a JSF managed bean

```
try {
    request.login(userName, password);
    request.logout();
} catch (ServletException ex) { }
```

A `ServletException` is thrown when either login or logout fails

Quiz

Select three common security concepts.

a. Scalability
b. Authorization
c. Authentication
d. Confidentiality

Answer: b, c, d

Quiz

Roles are tied to which of the following?

a. Systems and users
b. Users and privileges
c. Systems and databases
d. Privileges and systems

Answer: b

Quiz

Web-tier components configure their security settings in which file.

a. `application.xml`
b. `ejb-jar.xml`
c. `web.xml`

Answer: c

Quiz

Identify the version of enterprise Java that first allowed the `@ServletSecurity` annotation to be used in a Servlet.

a. Java EE 4
b. Java EE 5
c. Java EE 6

Answer: c

Quiz

Review the following code snippet and select the correct options:

```
1   <login-config>
2      <auth-method>FORM</auth-method>
3      <realm-name>ldap</realm-name>
4      <form-login-config>
5         <form-login-page>/faces/login.xhtml</form-login-page>
6         <form-error-page>/faces/error.xhtml</form-error-page>
7      </form-login-config>
8   </login-config>
```

a. Configuring authentication in `web.xml`
b. Configuring authorization in `application.xml`
c. Configuring Form-based authentication
d. Code for a login page

Answer: a, c

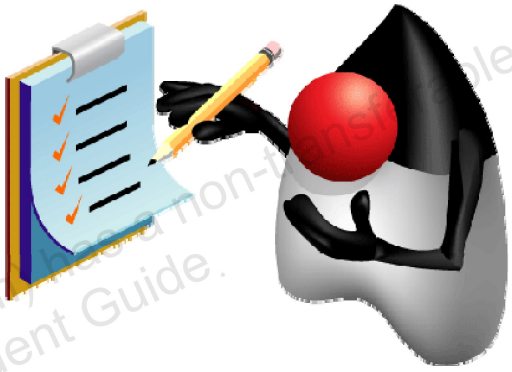
Additional Resources

Topic	Website
Security Tutorial	http://docs.oracle.com/javaee/6/tutorial/doc/gijrp.html
WebLogic 12c Security Realms	http://docs.oracle.com/cd/E24329_01/web.1211/e24484/realm_chap.htm
Securing Web Applications	http://docs.oracle.com/cd/E24329_01/web.1211/e24485/thin_client.htm#i1033642

Summary

In this lesson, you should have learned to:

- Configure JSF web applications in the web container
- Configure security for JSF web applications



Practice 14: Overview

This lesson covers the following practices:

- 14-1: Creating roles, users, groups, and a web-tier security policy
- 14-2: Using the `HttpServletRequest` `login` and `logout` methods