Fundamentals of Java Enterprise Components

Session: 15

Java Security

Objectives

- Describe the security mechanisms in Java
- Describe how to secure application containers
- Explain how to define user roles and groups
- Explain tasks associated with securing enterprise applications
- Explain how to secure Web applications
- Explain implementation of programmatic security

Introduction

Applications are deployed in containers.

Containers provide security as declarative security and programmatic security.

Declarative security is provided through deployment descriptors.

Programmatic security is provided by embedding security mechanisms in the application.

A security mechanism implements the following in an application:

- Prevents unauthorized access to application data
- Implements non-repudiation
- Ensures service continuity of the application

Characteristics of Security Mechanisms

Following are the characteristics of a security mechanism:

- Authentication
- Authorization
- Data integrity
- Confidentiality or data privacy
- Non-repudiation
- Quality of Service
- Auditing

Security Mechanisms

Following are the security mechanisms provided by Java EE:

- Java Authentication and Authorization Service (JAAS)
- Java Generic Security Services (JGSS)
- Java Cryptography Extension (JCE)
- Java Secure Sockets Extension (JSSE)
- Simple Authentication and Secure Layer (SASL)

The security mechanism can be defined at different layers of the application such as:

- Application layer
- Transport layer
- Message layer

Application Layer Security

Provided by the component container.

Application firewalls can be used to protect the communication among different application components.

Application layer security is responsible for enterprise applications.

Advantages and disadvantages of application layer security:

- The security mechanisms are fine grained with specific configuration settings
- Disadvantage of application layer security is that the security attributes are non-transferrable
- Application layer security is insufficient when there are multiple protocols used in the application

Transport Layer Security 1-2

Provided by the protocols used for transferring application data.

Commonly used protocol for transport layer security is secure HTTP with Secure Sockets Layer (SSL).

It is a point-to-point security mechanism used for authentication, message integrity and confidentiality on the network.

Following are different phases in Transport Layer Security:

- The communicating entities – client and server, agree upon a cryptographic algorithm
- The communicating entities agree upon a key
- Use symmetric cipher during message exchange
- Requires implementation of digital certificates when the communication is through SSL

Transport Layer Security 2-2

Following are the advantages and disadvantages:

- Simple mechanism applicable between two communicating entities
- Disadvantage is that it is tightly coupled with transport layer
- The security mechanism is applicable only between a pair of communicating entities not beyond that

Message Layer Security

Security information is sent in the message header.

The information is in the message header of the SOAP message.

Allows encryption of a part of the message.

Message level security can be implemented as both end-to-end security and point-to-point security.

Following are the advantages and disadvantages of message layer security:

- The mechanism stays with the message irrespective of the number of nodes through which the message traverses
- Security mechanism can be applied to only part of the message
- Independent of Transport layer protocol or application environment
- Adds additional overhead of message processing at each node

Container Security

Containers provide application security either declaratively or programmatically.

Declarative security mechanisms use annotations and deployment descriptors to declare the security mechanisms.

Annotations for specifying security mechanisms:

- Annotations provide declarative security mechanisms in the class files
- @RolesAllowed, @PermitAll, @DenyAll are some of the annotations used

Deployment descriptors for specifying security mechanisms:

- Using deployment descriptors prevents changing the application source code
- Deployment descriptors can also be used to provide structural information for each component of the application
- web.xml is the deployment descriptor for Web applications
- ejb-jar.xml is the deployment descriptor for EJB applications

Users, Groups, and Roles 1-3

Java applications define users, their respective roles, and user groups that are used to define access rights on application data.

The users are authenticated and then given access to protected resources. Following are the steps involved in authentication:

- An application developer would allow different users to access data by writing code which would prompt for username and password
- Application developer defines the security policy of the application through annotations or deployment descriptors
- The server administrator sets up groups of authorized users on the GlassFish server or any other Web server
- The application deployer maps different application security roles to users and user groups

Users, Groups, and Roles 2-3

- Any entity which tries to access application can be termed as a user.
- Users are authenticated through username and password.
- Multiple users can be grouped together into an user group.
- A security policy domain defined for a Web application is known as a realm.
- Each realm can have its own authentication scheme.
- The Java EE authentication service manages different realms on Web server.

Users, Groups, and Roles 3-3

A group of users in an application domain is a set of authenticated users who are defined by a set of common traits.

A Role defines access to a particular set of resources in an application.

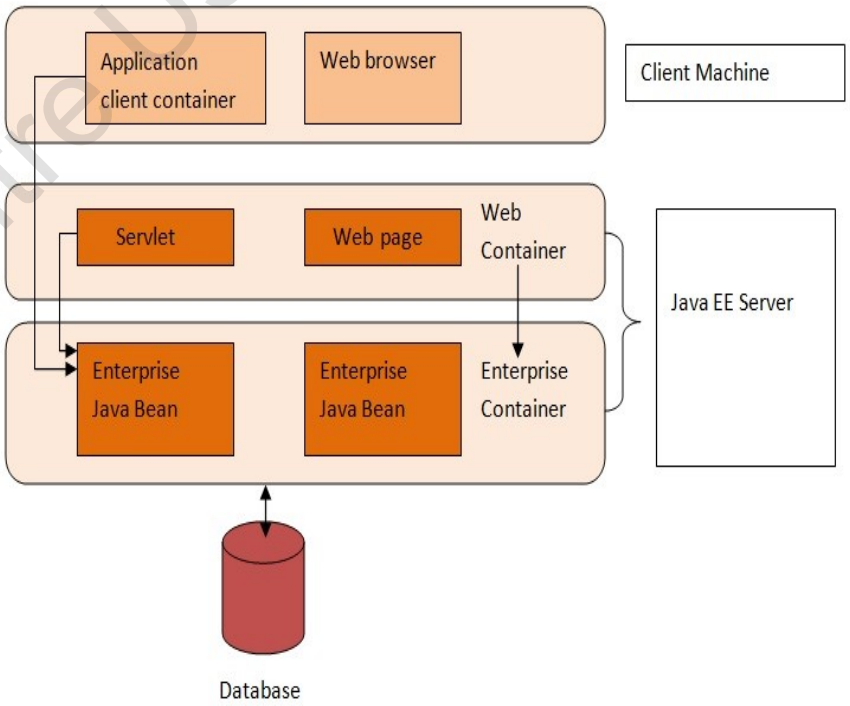
Following are terms associated with Java security mechanisms:

- Principal
- Security policy domain
- Security attributes
- Credentials

Basic Security Tasks for Enterprise Applications

Security of an enterprise application is implemented by system administrators, application developers, and deployers.

Following figure shows the enterprise application architecture:



Following are the tasks for defining the security of the application:

- Defining the database of users and grouping them into user groups
- Defining the method of identity propagation
- Configuring the Web server
- Annotating the classes and methods

Securing Enterprise Beans

Security mechanism can be defined declaratively and programmatically.

Declarative security is the preferred method of defining the security.

Application developer defines different types of users who may access application data.

Roles are defined based on the user definition in the application.

Based on the role assigned, the user is granted access to certain bean method.

Securing an enterprise bean using declarative security:

- Developer defines the roles in the application and passes the information to the application deployer
- Deployer defines the security view of the application through a set of security roles and permissions to the users

Specifying Authorized Users by Declaring Security Roles 1-2

Permissions of access can be defined at the class level or at the method level.

Annotations like @DeclareRoles, @RolesAllowed, @PermitAll, and @DenyAll are used to declare access to bean methods and classes.

@DeclareRoles specifies the list of all the roles that will be used in the application.

@RolesAllowed specifies the roles that are allowed access to methods. For example:

```
@DeclareRoles({"Administrator", "Faculty", "Student"})
public class Calculator {
    @RolesAllowed("Administrator")
    public void setIncentive(int rate) {
        ...
    }
}
```

Specifying Authorized Users by Declaring Security Roles 2-2

@PermitAll annotation specifies that all the security roles can access the given bean method. Following code demonstrates the usage of @PermitAll annotation:

```
public class Hello {
                @PermitAll
                public void greeting(String name) {
                        System.out.println("Hello, " + name);
                }
              }
```

@DenyAll annotation specifies that no security role can access the current bean class or bean method. Following code demonstrates the usage of @DenyAll annotation:

```
public class Hello {
                @DenyAll
                public void getUserData() {
                        …..
                }
              }
```

Securing an Enterprise Bean Method Programmatically

javax.ejb.EJBContext provides methods to access security information about the user or entity who is invoking the enterprise bean method.

Following are the methods:

- `getCallerPrincipal()`
- `isCallerInRole()`

Following code demonstrates the usage of isCallerInRole() method:

```
@Resource Session Context x
if(x.isCallerRole(admin)==true){
System.out.println("Admin rights assigned");
}
else{
System.out.println(" No Admin rights");
        }
```

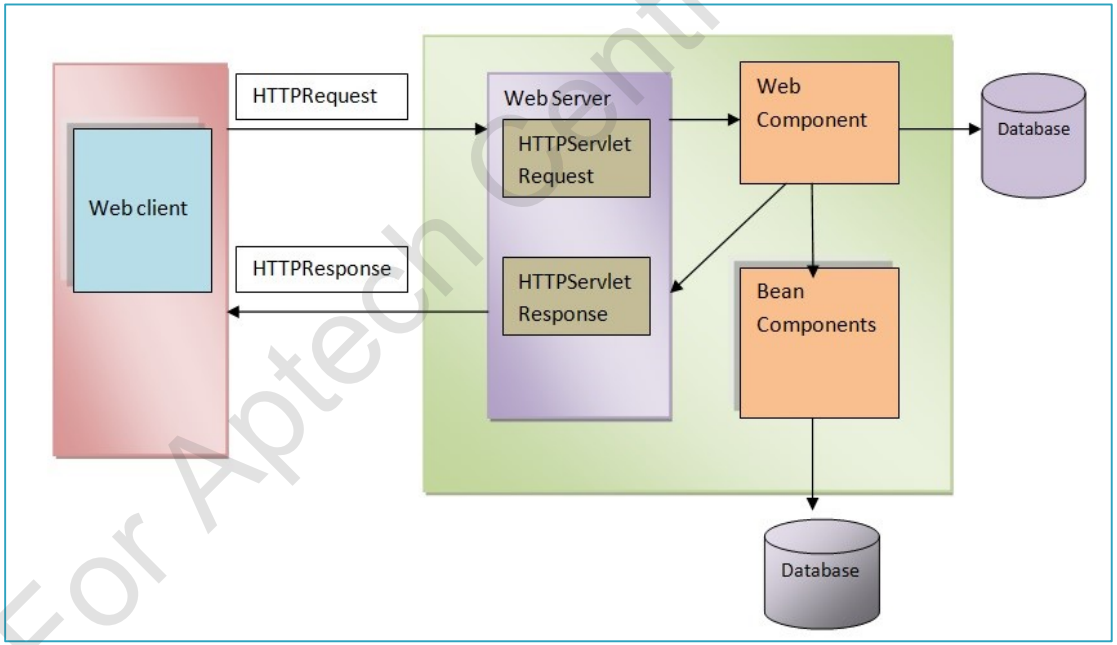
Propagating a Security Identity

- When an entity is allowed access to a bean class or method, the method may in turn call other methods.
- The security identity for the second method call or second level of access can be defined according to one of the following options:
  - The identity of the entity through which the user accessed the first method can be propagated to the second entity by default.
  - When the target enterprise bean expects a specific identity, then the expected identity is forwarded to the target enterprise bean.

Securing Web Applications 1-2

Web applications are accessed using Web browser and contain resources which are accessible by many users.

Following figure shows the layered architecture of Web applications:



Securing Web Applications 2-2

Web application security is implemented declaratively, programmatically and through message security.

- Declarative security declares the security mechanisms through annotations and deployment descriptors
- Programmatic security uses classes to implement the security mechanism
- Message security is implemented through security features such as digital signatures and encryption, which are transmitted as a part of the message in the SOAP message header

Specifying Security Constraints 1-3

A security constraint is defined in the deployment descriptor to define access privileges for a collection of resources, through security-constraint element.

The security-constraint element can have the following sub-elements:

- **web-resource –collection**: This element represents the collection of URLs which represent the resources of the application
- **auth-constraint**: This element is used to define the authorization
- **user-data-constraint**: This element defines how user data is protected while traversing over the network

If the Web application uses servlets, @HttpConstraint and @HttpMethodConstraint annotations can be used within the @ServletSecurity annotation to define the security constraint.

Specifying Security Constraints 2-3

Specifying Web Resource Collection - In order to specify a Web resource collection to be protected, following are the sub-elements used along with the element web-resource-collection:

- `web-resource-name`
- `url-pattern`
- `http-method`
- `http-method-omission`

Specifying the Authorization Constraint - The auth-constraint element has a sub element role-name which is used to specify the application roles that are authorized to access certain application resources.

Specifying a Secure Connection - The user-data-constraint has a transport-guarantee sub element, which has three levels of transport guarantee as follows:

- `CONFIDENTIAL`
- `INTEGRAL`
- `NONE`

Specifying Security Constraints 3-3

Following code demonstrates the definition of an authorization constraint:

```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>Results</web-resource-name>
                <url-pattern>/University/Results/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
        <role-name>Professor</role-name>
        </auth-constraint>
        <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>
```

Specifying Authentication Mechanisms 1-2

The authentication mechanism defines the access to the resources. Java EE platform supports the following authentication mechanisms:

- Basic authentication
- Form-based authentication
- Digest authentication
- Client authentication
- Mutual Authentication

Following is the sequence of operations in a basic authentication mechanism:

- The client tries to access a protected resource of the application
- The security mechanism prompts for username and password through a dialog box
- The client responds with the username and password in the dialog box
- The server verifies the username and password and grants/denies access

Specifying Authentication Mechanisms 2-2

Form-based authentication: The client trying to access a protected resource is redirected to a login page, which has an elaborate security mechanism. Following are the steps in Form-based authentication:

- The client tries to access a protected resource
- The server redirects the client to a login page/form requesting for the client details
- Based on the details filled in the login page, the client is granted access to the resource

Digest authentication: It follows the same steps as basic authentication. The exchange of username and password is as cryptographic hash.

Client authentication: It is done through the public key of client. Requires public key cryptography mechanism.

Mutual authentication: Implemented through mutual exchange of secret information, there are following two variants of it:

- Certificate based authentication
- Username-password based authentication

Using Programmatic Security Through Web Applications

Following tasks should be implemented through code when security of the Web applications is programmatically implemented:

- Authenticating users
- Checking caller identity

Authenticating users – HttpServletRequest has the following methods to authenticate users:

- `authenticate()`
- `login()`
- `logout()`

Checking caller identity programmatically – Following methods are provided by Servlets 3.0 to check caller identity:

- `getRemoteUser()`
- `getUserInRole()`
- `getUserPrincipal()`

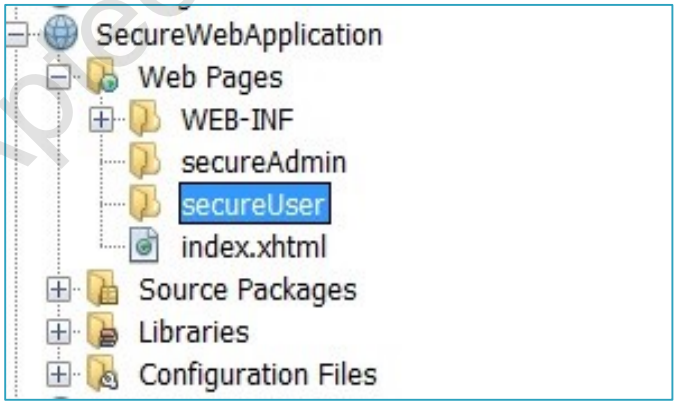
Configuring Declarative Security 1-17

A Web application can be configured with different security options. The developer can give access to certain Web pages to a specific role of the application.

To begin with, create a Web application named, SecureWebApplication in the NetBeans IDE. Select JSF from the frameworks while creating the application.

Define two different security domains in the Web Pages folder of the application

The hierarchy of the folders in the application is shown in the following figure:



Configuring Declarative Security 2-17

Create Web pages within the secureAdmin and secureUser directories.

The Web pages created are named, secureAdminPage.html and secureUserPage.html respectively.

Following code demonstrates the html code of the secureAdminPage:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Admin page</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width">
    </head>
    <body>
        <h1>Secure admin page</h1>
    </body>
</html>
```

Configuring Declarative Security 3-17

Following code demonstrates the HTML code of the secureUserPage:

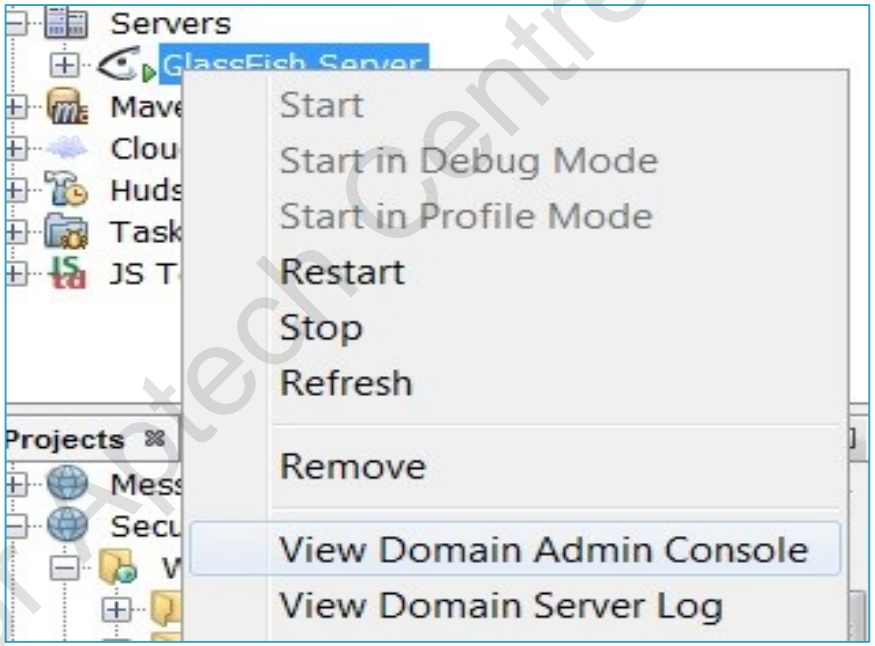
```
<html>
    <head>
        <title>Secure User page</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width">
    </head>
    <body>
        <h1>Secure user page</h1>
    </body>
</html>
```

Add the following code to the body section of index.xhtml:

```
<p> Access to secure Admin page <a
href="secureAdmin/secureAdminPage.html">here!</a></p>
<p>Access to secure User page <a
href="secureUser/secureUserPage.html" >here!</a></p>
```

Configuring Declarative Security 4-17

The roles of the application should be defined on the Web server. In order to define the users and roles on the application, access the Domain Admin Console as shown in following figure:

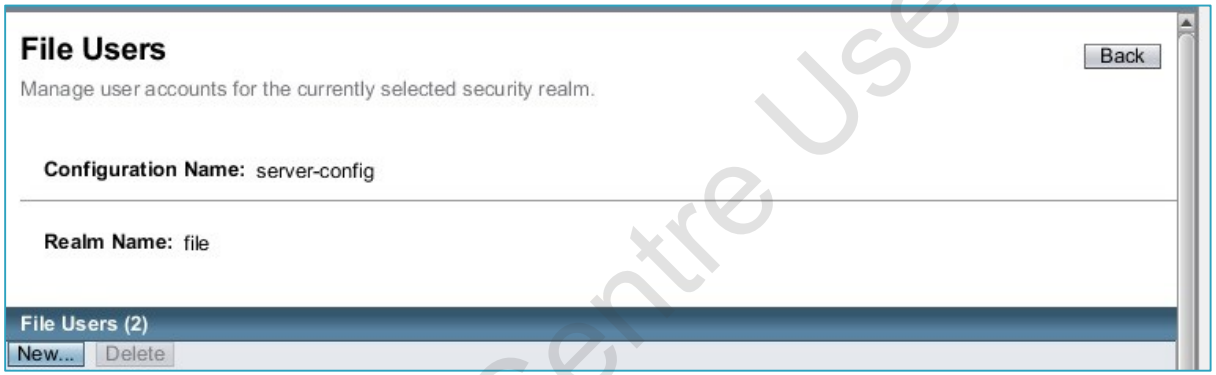


Configuring Declarative Security 5-17

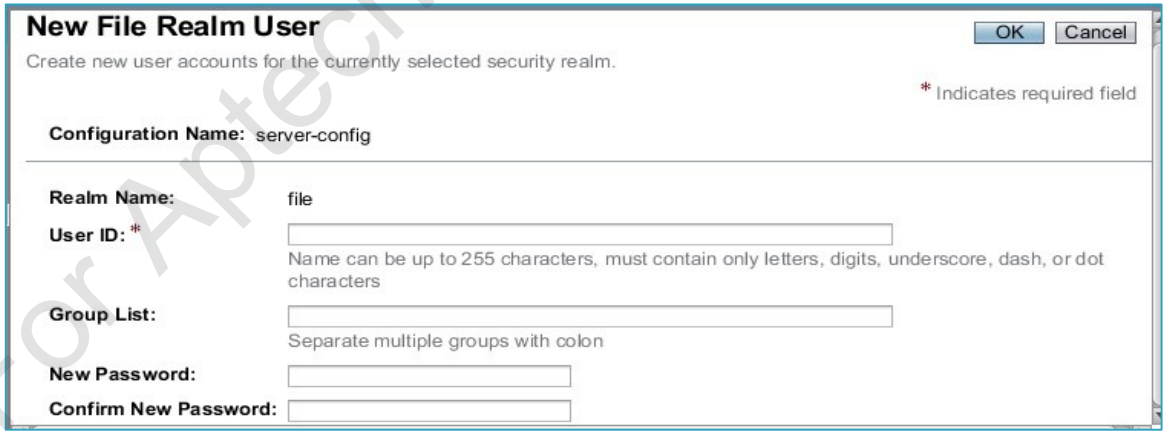
- In order to define the roles for the intended files in the application, follow the given path.
- Configurations → server-config → Security → Realms → file. 'File' is selected as security is defined at the file level in the application. The selection appears as shown in the following figure:

Configuring Declarative Security 6-17

Click the Manage Users button in the server configuration screen. This will lead to an interface where users for the application can be defined as shown in the following figure:



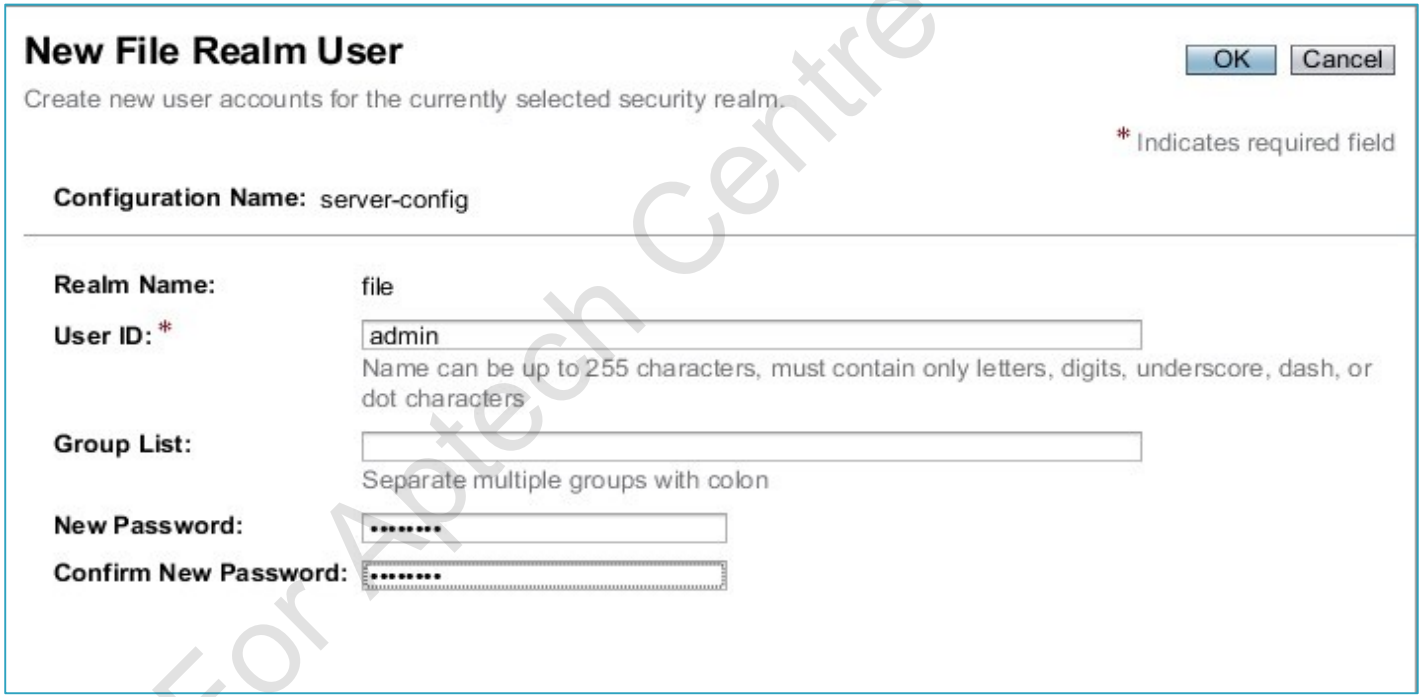
Click New. The New File Realm User screen is displayed as shown in the following figure:



Configuring Declarative Security 7-17

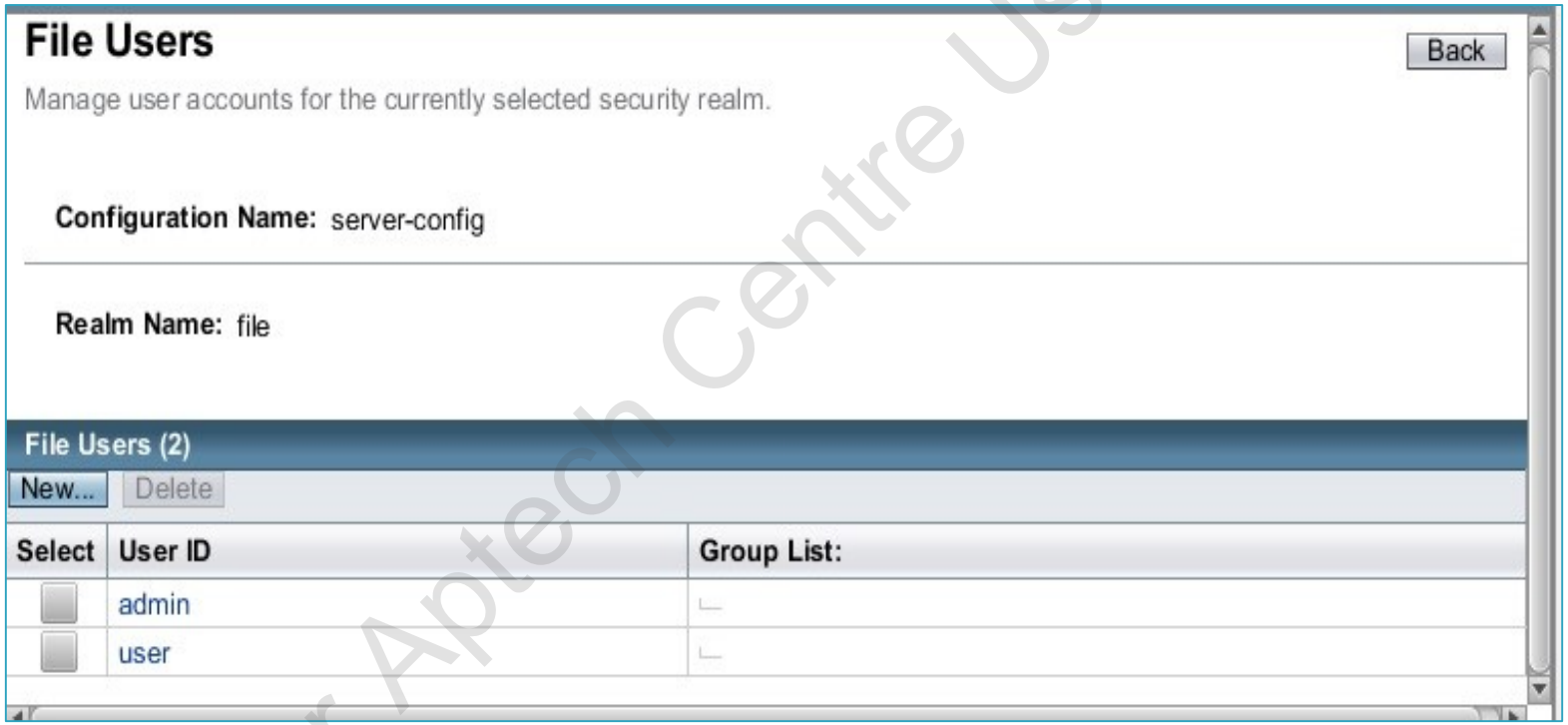
Create two users, admin and user (specify the names in the User ID box) as per the application requirement. Provide password as admin123 for admin and user123 for user.

Following figure demonstrates creation of the admin user:



Configuring Declarative Security 8-17

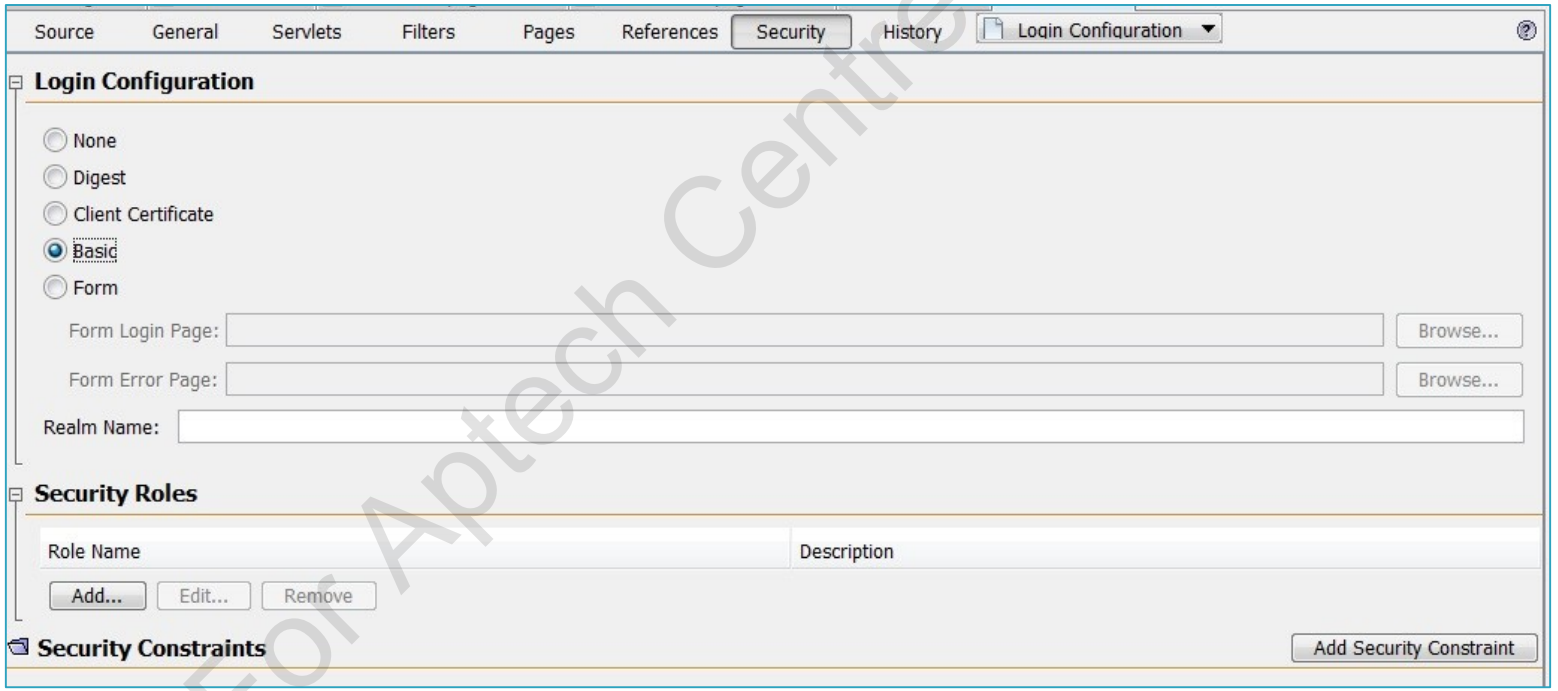
Once the users are created, they will appear in the user table as shown in following figure:



Configuring Declarative Security 9-17

Once the users are defined, the developer has to define the authentication mechanism to login and access the resources. This is done in the application deployment descriptor web.xml.

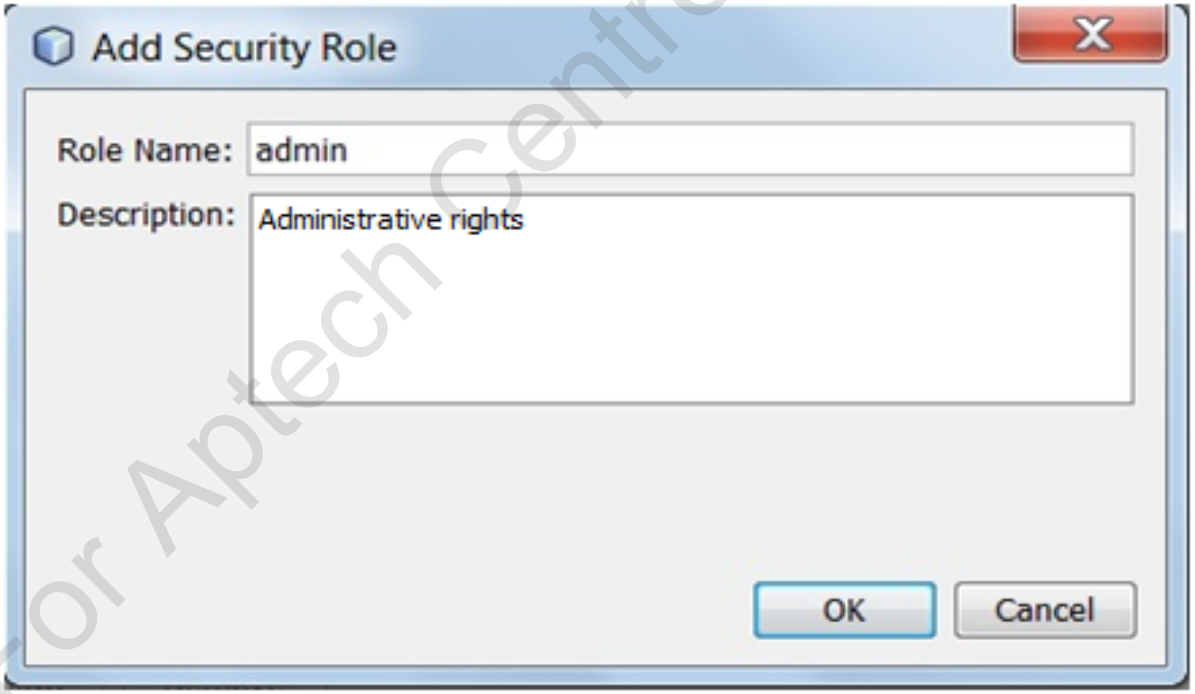
Click the Security tab in the deployment descriptor as shown in following figure:



Configuring Declarative Security 10-17

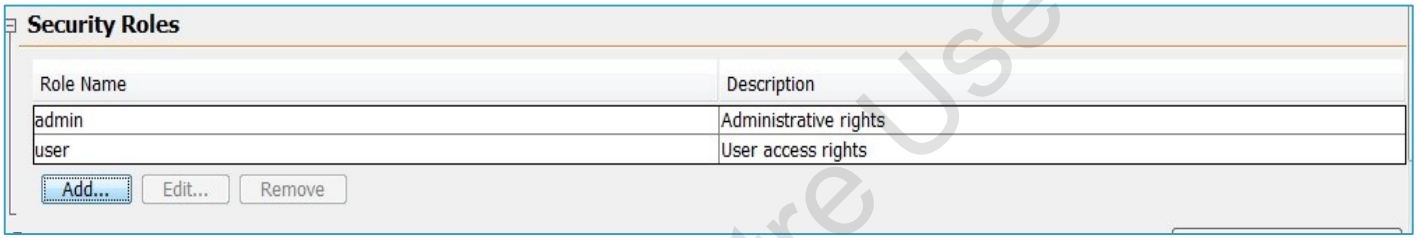
Define the login mechanism in the Login Configuration section; here the Basic login mechanism is selected.

Add security roles to the application by clicking Add in the Security Roles section. It will lead to the screen as shown in the following figure:

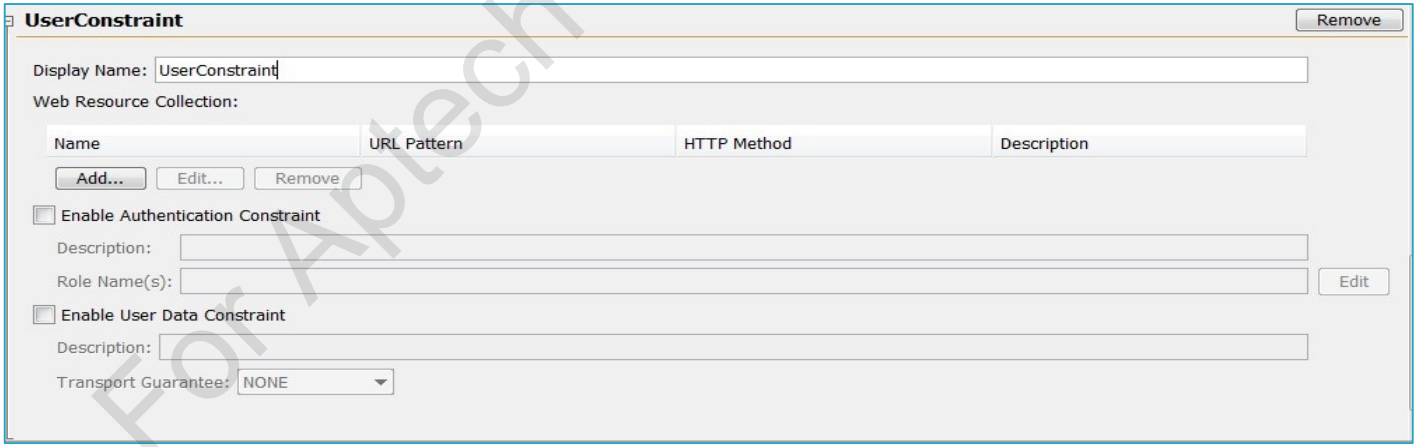


Configuring Declarative Security 11-17

Following figure shows the state of the security roles table after the roles are added with corresponding description:

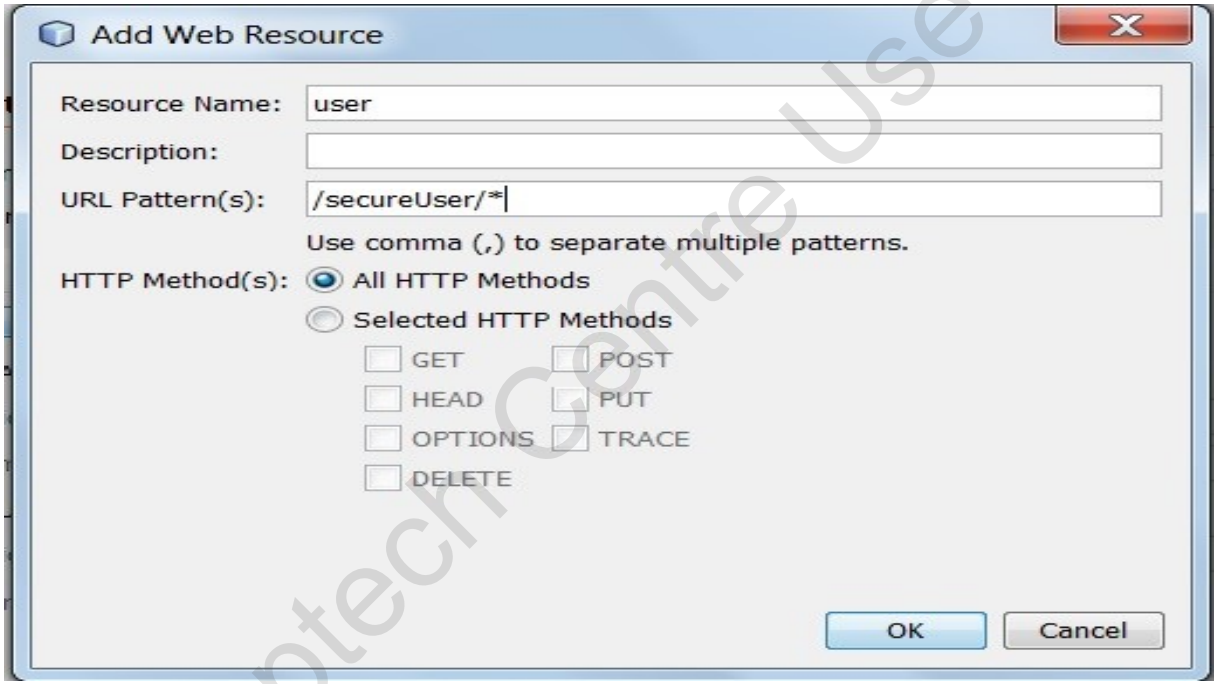


Once the roles are added, define the security constraints in the deployment descriptor by clicking Add Security Constraint. It leads to a screen as shown in the following figure:



Configuring Declarative Security 12-17

Specify UserConstraint in the Display Name box and click Add. This will lead to the screen as shown in the following figure:



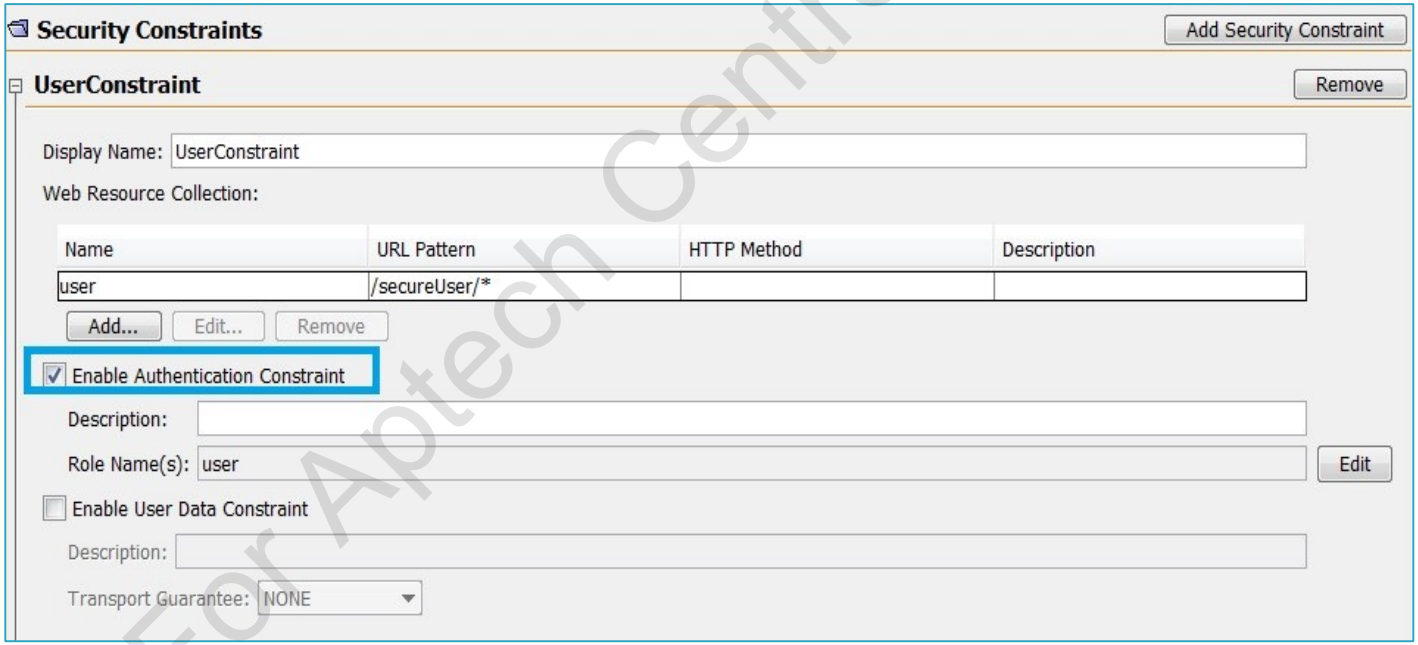
In this wizard, map the user to the folder which the user can access, i.e all the files in the folder secureUser according to the given URL pattern. Click OK. The user to resource mapping will be displayed in the Web Resource Collection section. Similarly, add the AdminConstraint for admin with respect to the secureAdmin folder.

Configuring Declarative Security 13-17

Select the checkbox 'Enable Authentication Constraint' while defining the UserConstraint as well as the AdminConstraint.

Click Edit to set the Role Name as user for UserConstraint and admin for AdminConstraint.

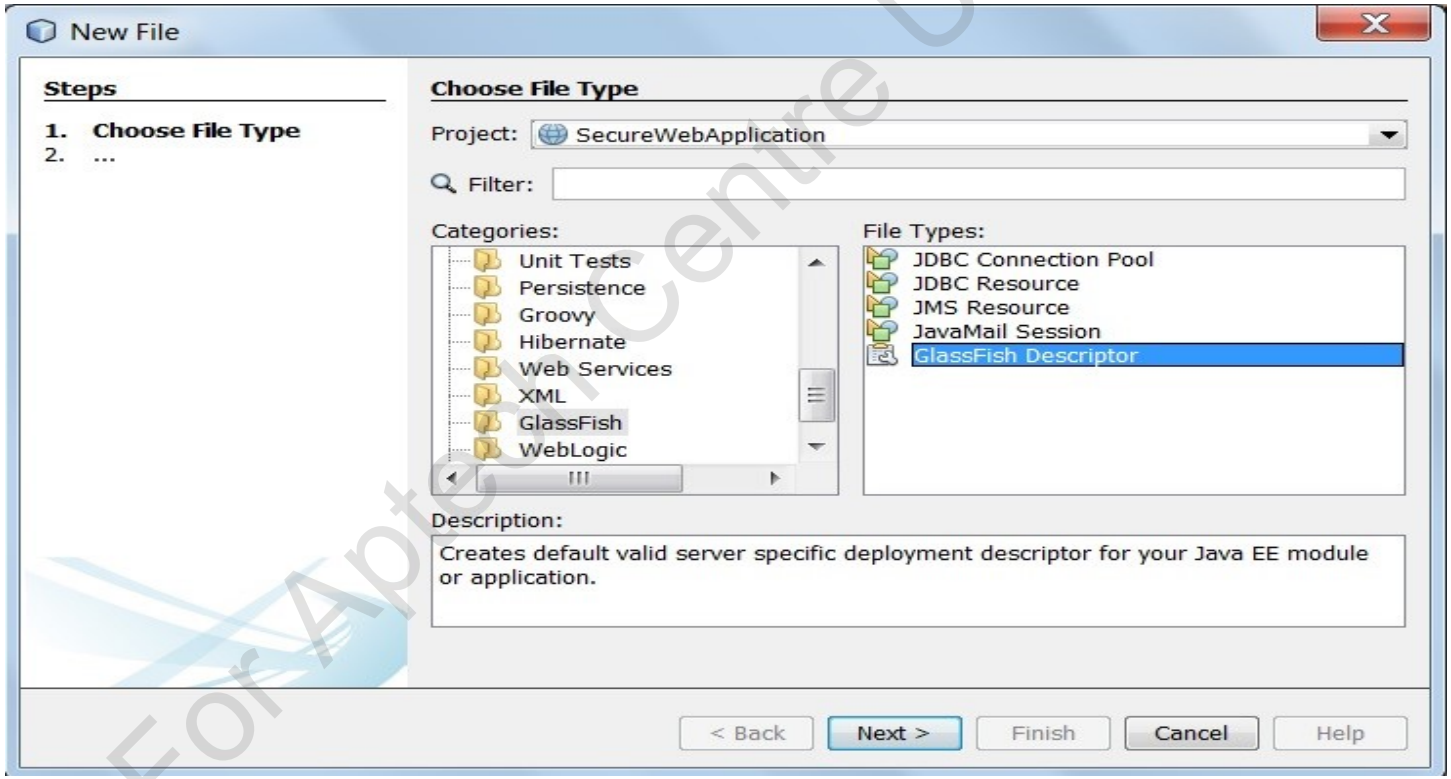
Following figure shows the final state of the User Constraint after all the values are configured:



Configuring Declarative Security 14-17

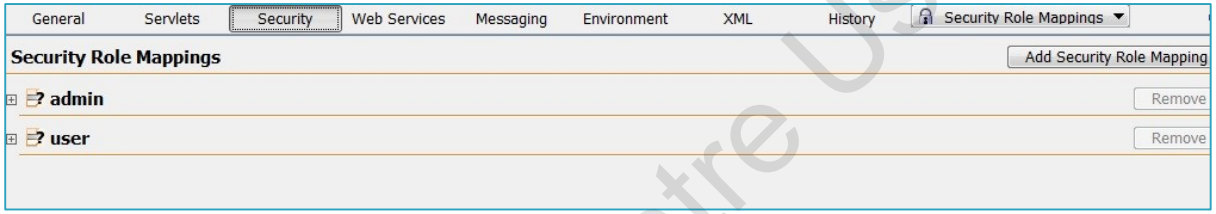
Once the application's deployment descriptor is configured, configure the Web server deployment descriptor. In case of GlassFish server, it is glassfish-web.xml.

If the deployment descriptor is not already present in the application, right-click the project and select New → Other → GlassFish → GlassFish Descriptor as shown in the following figure:

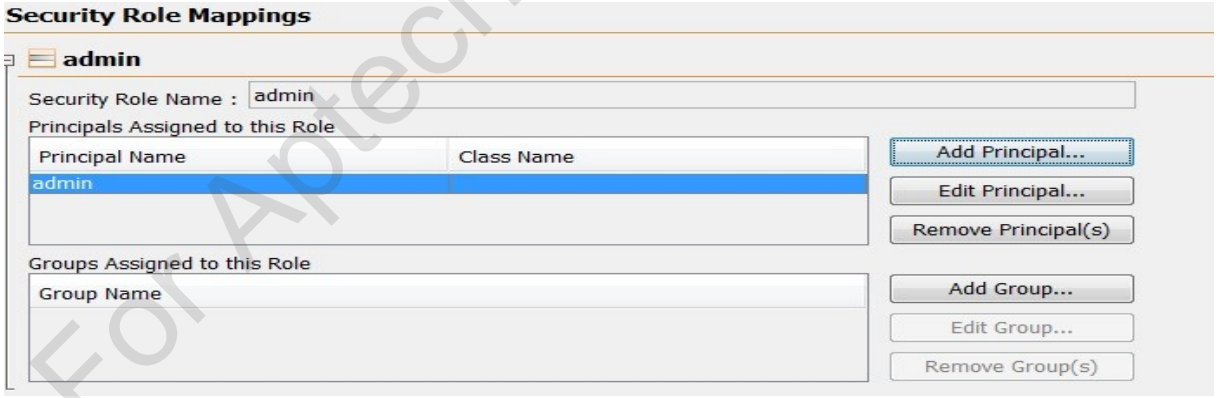


Configuring Declarative Security 15-17

Once the Web server deployment descriptor is created, open it and click the Security tab. This will lead to the screen as shown in the following figure. The roles created earlier in the server's Admin Console are seen listed here.

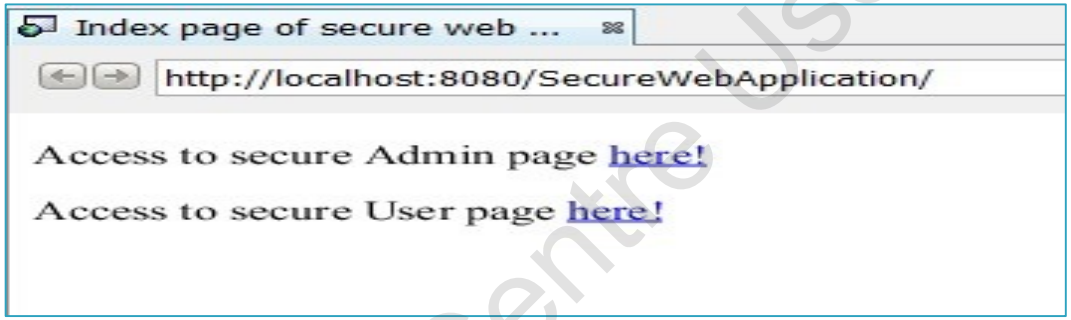


On expanding the security roles, the screen shown in the following figure appears. Click Add Principal to add users to the role. For admin role, add 'admin' as the Principal. Similarly, add 'user' as Principal for the user role.

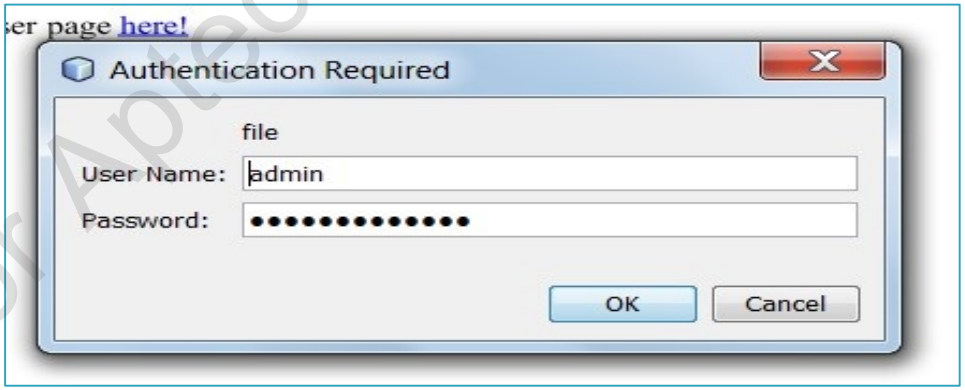


Configuring Declarative Security 16-17

After all the configurations are completed, deploy and run the application. The application execution will lead to the index page as shown in the following figure:



Click the hyperlink. It will prompt for user name and password as shown in the following figure:



Configuring Declarative Security 17-17

On providing the appropriate credentials, it will lead to the Web page as shown in the following figure:

Summary

- Security mechanisms in both enterprise and Web applications are specified both declaratively and programmatically.
- Security mechanisms are declaratively specified through annotations and deployment descriptors.
- Programmatically security mechanisms are specified through Java security APIs such as JAAS.
- Security mechanisms in Web applications are implemented at three levels: application level security, transport level security, and message level security.
- Methods of HttpServletRequest interface are used to programmatically define security mechanisms for Web applications.
- The security roles and mapping can be done on the application deployment descriptor and Web server deployment descriptor.