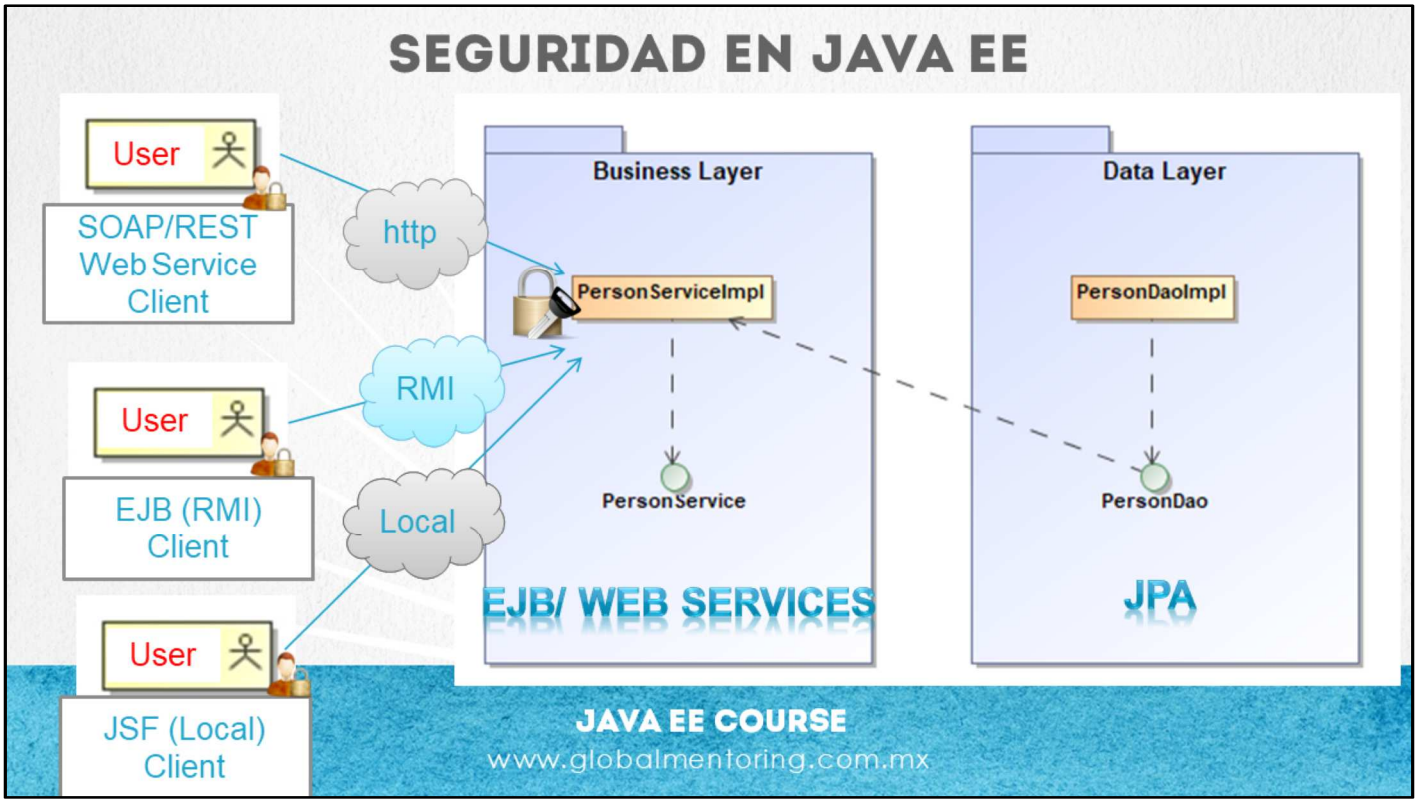
Hello, Ubaldo Acosta greets you again. I hope you're ready to start with this lesson.

We are going to study the topic of Security in Java EE.

Are you ready? Let's go!



One of the crucial issues when developing Java business applications is the security issue.

Whether our application is consulted through a Web interface using Servlets or JSF, we expose the functionality using Web Services SOAP or REST, or that directly a Swing Client has access to our EJB, it is necessary to establish the mechanisms that guarantee security of the system information.

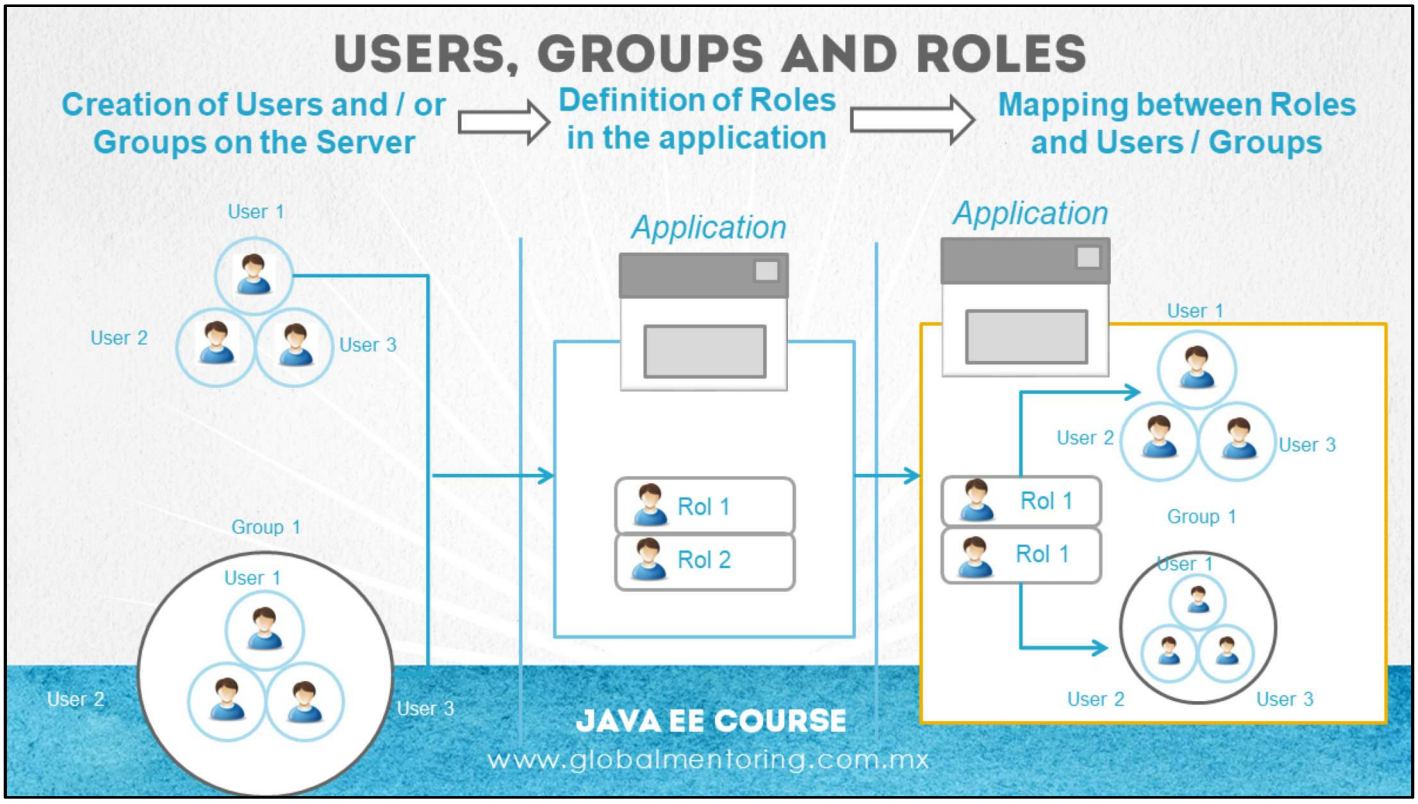
Security must be secured from the point of view of the view and the business layer, because a user who can not view a page does not mean that he can not execute a business layer method.

One of the benefits of using EJB is the simplicity and flexibility to add security in the execution of your methods. Similar to the handling of transactions, the Security management we have the Declarative Security and the Programmatic Security. We will review each of them in the following sheets.

Two concepts are basic regarding the security of a system. The concepts of Authentication and Authorization:

Authentication: Authentication is the process of verifying the identity of the user. The most common is to identify ourselves before the system by means of a username and password. These data are known as credentials. More advanced forms of authentication are through biometric authentication, such as: fingerprints, iris and retina patterns, detection of physical features such as the face, etc. Authentication must be executed before authorization.

Authorization: Once the user has authenticated in the system, the next step is to verify the user's permissions to execute certain system functionality. In such a way, that regardless of whether a user has authenticated correctly, they will not necessarily have the authorization to execute the desired functionality in the system. These two concepts are linked to the management of users, groups and roles, which we will review below.



Users, Groups and Roles are 3 inter related concepts that form the basis of the security scheme in a business system.

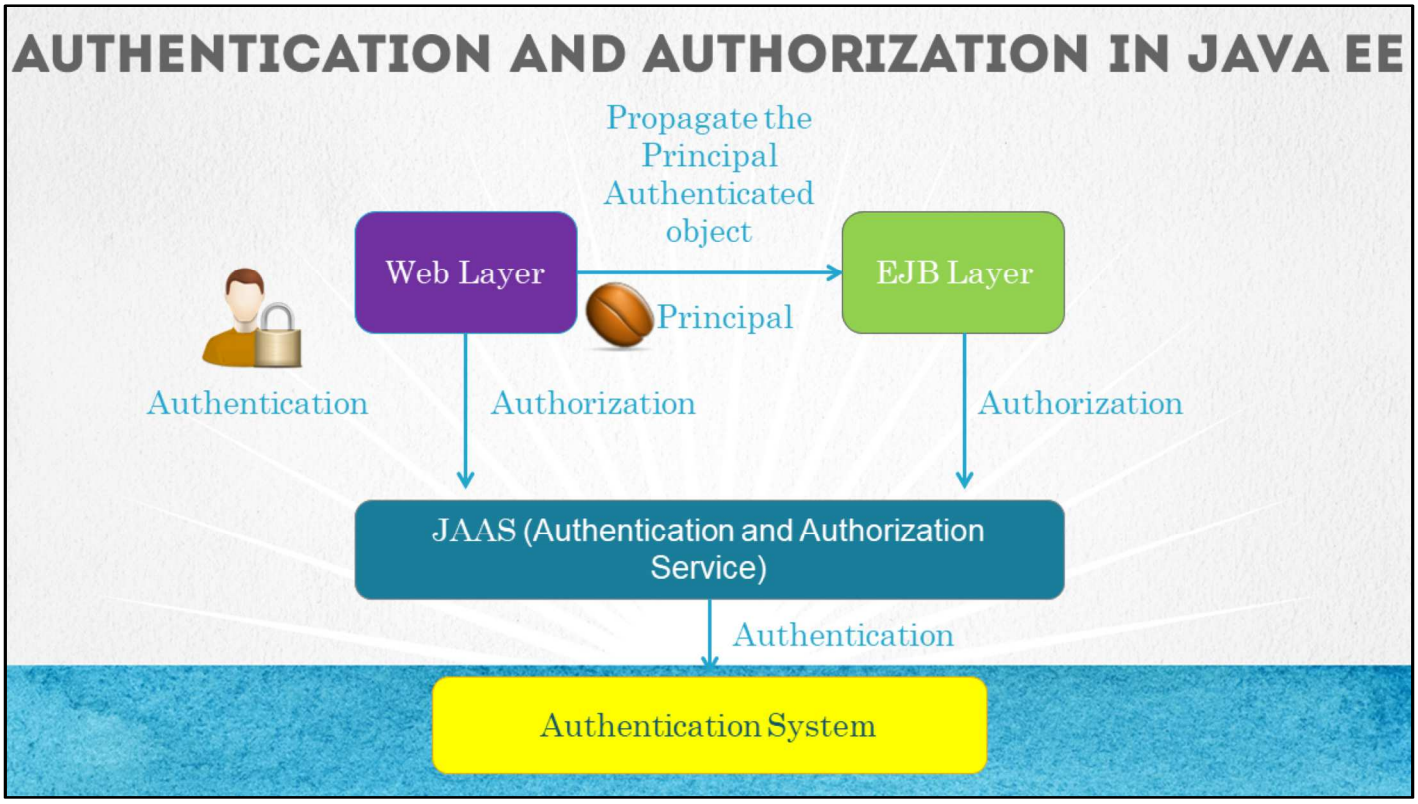
To simplify administration, users are divided into groups. The groups are a logical partition for the identification of users, who can access different resources. For example, an application can have a group of "Administrators" made up of users with permission to delete records.

Before executing any operation in the system, the application checks if the user is a member of the group and allows or denies the execution of the requested method / resource. The management of groups allows to assign the common functions of the users to a single entity, and in this way facilitate the administration of each individual.

A role, is a concept very similar to a group. However, as shown in the figure, it is a concept more linked to the application that is being developed. In contrast, a group is defined at the organizational level, for example, in an Active Directory of Microsoft or an LDAP (Lightweight Directory Access Protocol). A role allows to relate the groups of an organization in entities that understand and process the application correctly.

In this way, the roles are an abstraction of the groups focused on the Java application server. A mapping between roles and groups is commonly carried out. If the names of the groups and roles are equivalent, most application servers can perform the mapping automatically, otherwise the server must be configured to perform this task.

In case the names of the groups and roles do not match, you must specify the mapping of roles-groups to the Java server. This can be done through configuration of the same server or through configuration files. For example, in the case of GlassFish, the file glassfish-web.xml must be added. For more information on the mapping of groups and roles, you can consult the following link:
https://javaee.github.io/tutorial/security-webtier001.html
https://javaee.github.io/glassfish/documentation
http://docs.oracle.com/cd/E18930_01/html/821-2417/beaql.html
http://docs.oracle.com/cd/E18930_01/html/821-2418/beabg.html#scrolltoc



Security in Java EE is based largely on the JAAS (Authentication and Authorization Service) API. As the name implies, this API is responsible for the process of user authentication and resource authorization of Java systems, especially focused on adding security in the Web layer and the business layer where the EJBs are located.

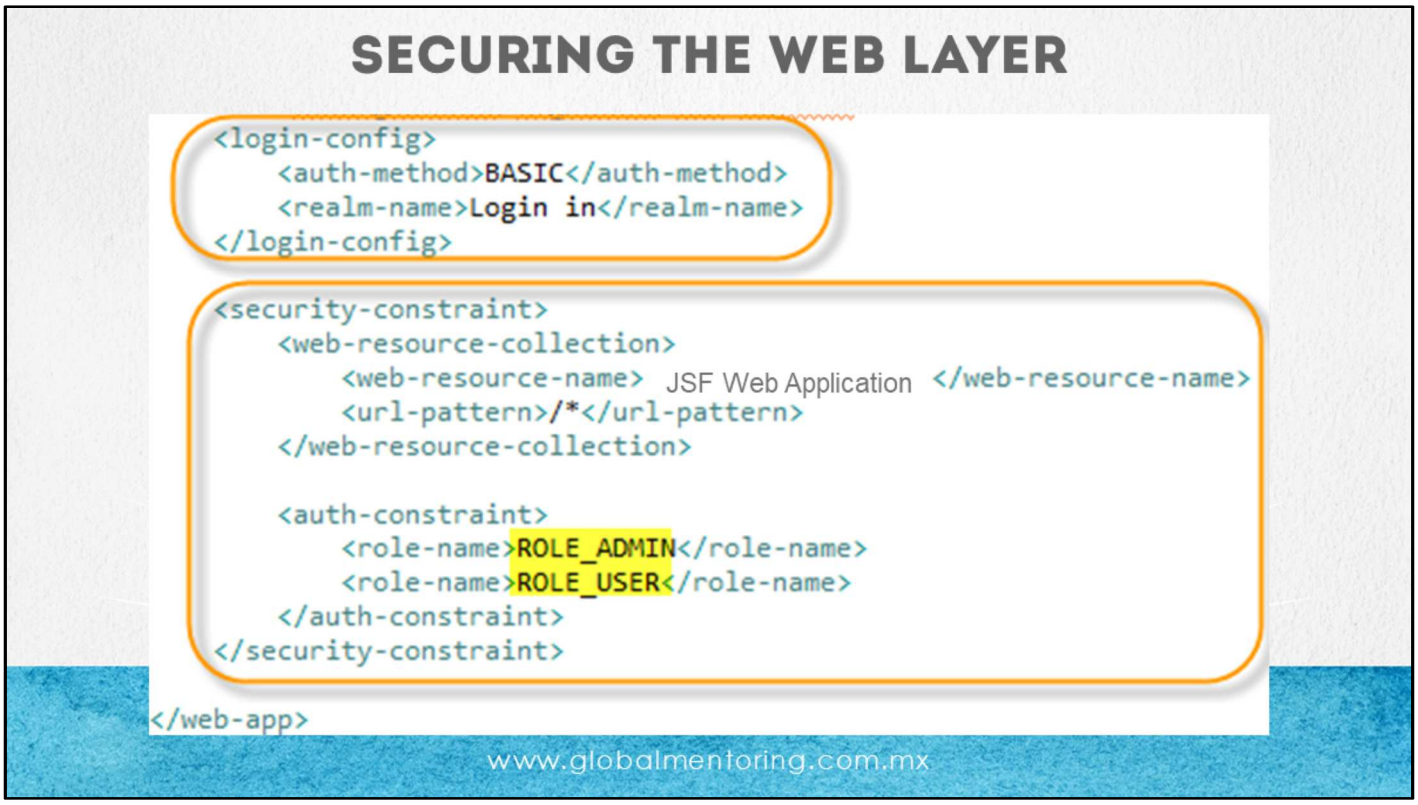
Once the user of the system is authenticated, the authentication context is propagated through the different layers of a business application, whenever possible. This is done so that you avoid the authentication process for each of the layers.

Once the user has authenticated, the JAAS security API creates an object known as Principal. This object is propagated between the layers in order to no longer request user authentication again.

As shown in the figure, a user can be authenticated through a Web application, in turn the Web layer retrieves the information and authenticates the user, creating the Main object in case of a successful authentication.

The Main object is associated with one or more roles. The security system reviews for each action, either in the Web layer or in the layer of EJBs, that you have the permissions to execute said resource.

The Main object is sent transparently between the Web and EJB layers as necessary.



Security in the Web layer is done through the web.xml descriptor file. There are different types of authentication which are HTTP Basic, HTTP DIGEST, HTTPS Client-Cert and FORM based. The detailed study of each of these settings is beyond the scope of this course, however in the exercises we will use the most common methods, with the aim of becoming familiar with this configuration.

As we observe in the code of the figure, the first element that we add is <login-config>, which allows us to specify the way in which the Web container will retrieve the information in order to authenticate the users.

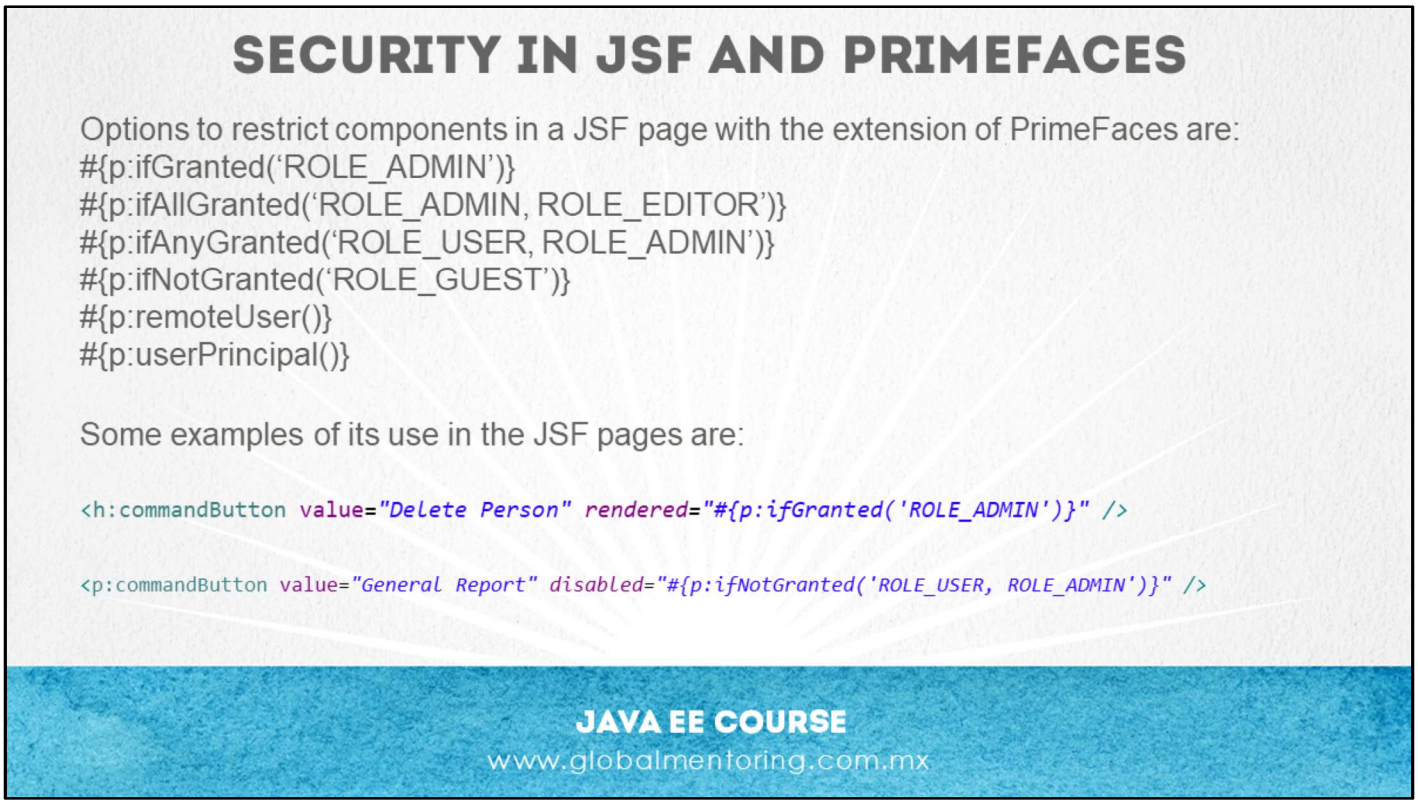
Later we have the element <security-constraint>, which will allow us to specify the resources (URL) to which we will add security, as well as the Roles that will participate in the security review. It is possible to specify even the type of HTTP Method that will be allowed (eg GET, POST, PUT, DELETE, etc).

The type of authentication shown in the figure is of the BASIC type, which the Web browser will display a generic pop-up requesting the user and password to be able to authenticate to the system. Another method very commonly used is the method based on an HTML form (FORM based). This configuration has the advantage that it is possible to customize the authentication page that is displayed to the user.

Subsequently the real-name. A realm is an abstraction of the application server, where the security policies of the system are specified. A realm contains a collection of users, which may or may not be associated with a group. A realm is configured using the administration tools of the application server in question. In our exercise we will use a real based on a file, however there are realms based on JDBC, LDAP, and we can add customized realms, since sometimes we will use external systems to perform the authentication process.

The <security-constraint> element allows us to specify one or more URL collections that we want to add security. By means of a url-pattern, we specify the pages to be secured. Finally with the <auth-constraint> element we specify the allowed roles to execute the specified resource.

In this way we have added both the concept of Authentication and the concept of Authorization for a Web application. The issue of how to secure an EJB will be discussed later. For a more detailed study of the types of authentication you can consult the following link:
https://javaee.github.io/tutorial/security-intro001.html



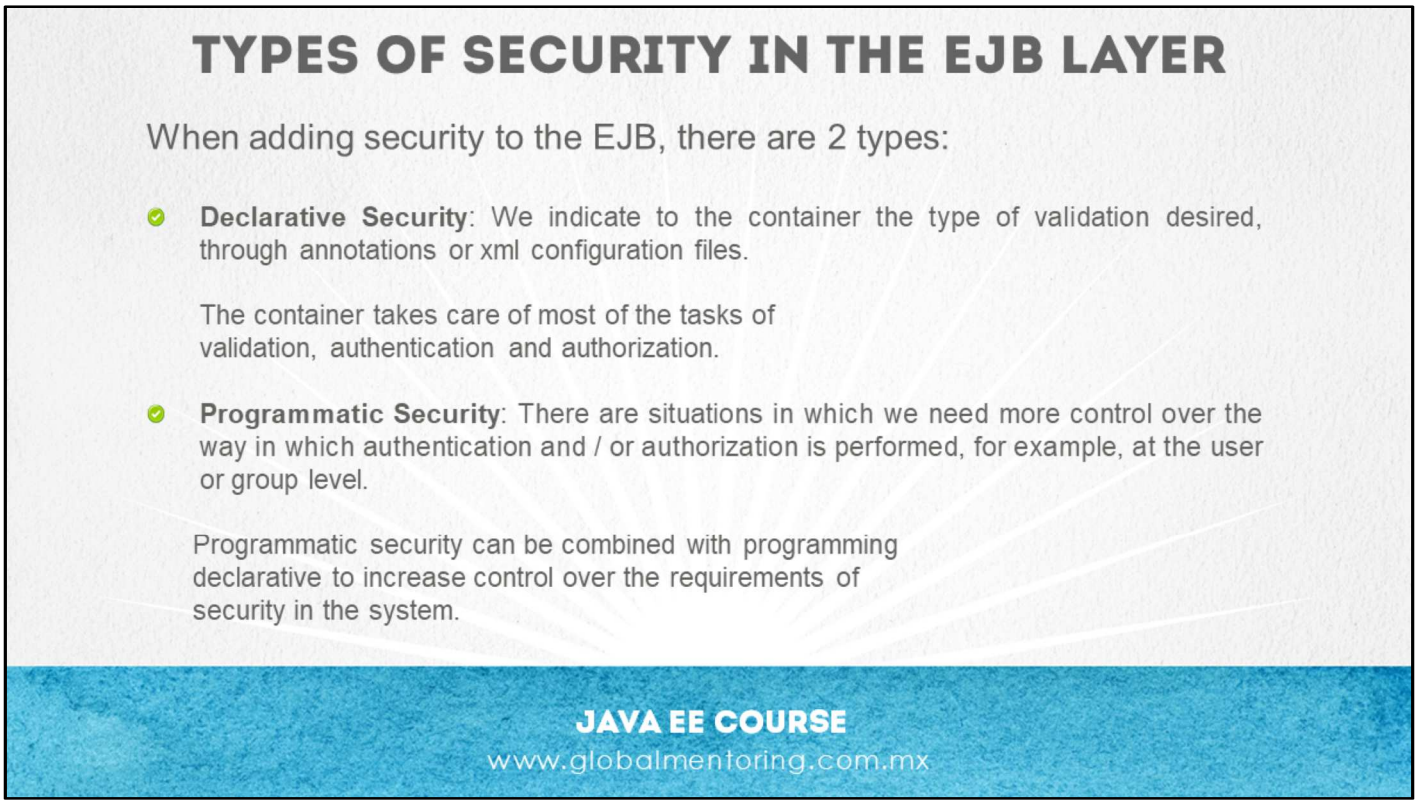
By authenticating us correctly through the Web application, JSF and in particular the extension of PrimeFaces, has added several elements that allow us to easily add security to our JSF pages.

In such a way that we can enable or disable functionality, even at the level of components, buttons, links, tables, etc., depending on the role of the user that has been authenticated.

The options that we have available with the extension of PrimeFaces are the following:
#{p:ifGranted('ROLE_ADMIN')}
#{p:ifAllGranted('ROLE_ADMIN, ROLE_EDITOR')}
#{p:ifAnyGranted('ROLE_USER, ROLE_ADMIN')}
#{p:ifNotGranted('ROLE_GUEST')}
#{p:remoteUser()}
#{p:userPrincipal()}

Some examples of its use are:

```
<h:commandButton value="Delete Person" rendered="#{p:ifGranted('ROLE_ADMIN')}" />
<p:commandButton value="General Report" disabled="#{p:ifNotGranted('ROLE_USER, ROLE_ADMIN')}" />
```



Similar to the handling of transactions, in the subject of security it is also possible to handle Declarative Security and Programmatic Security.

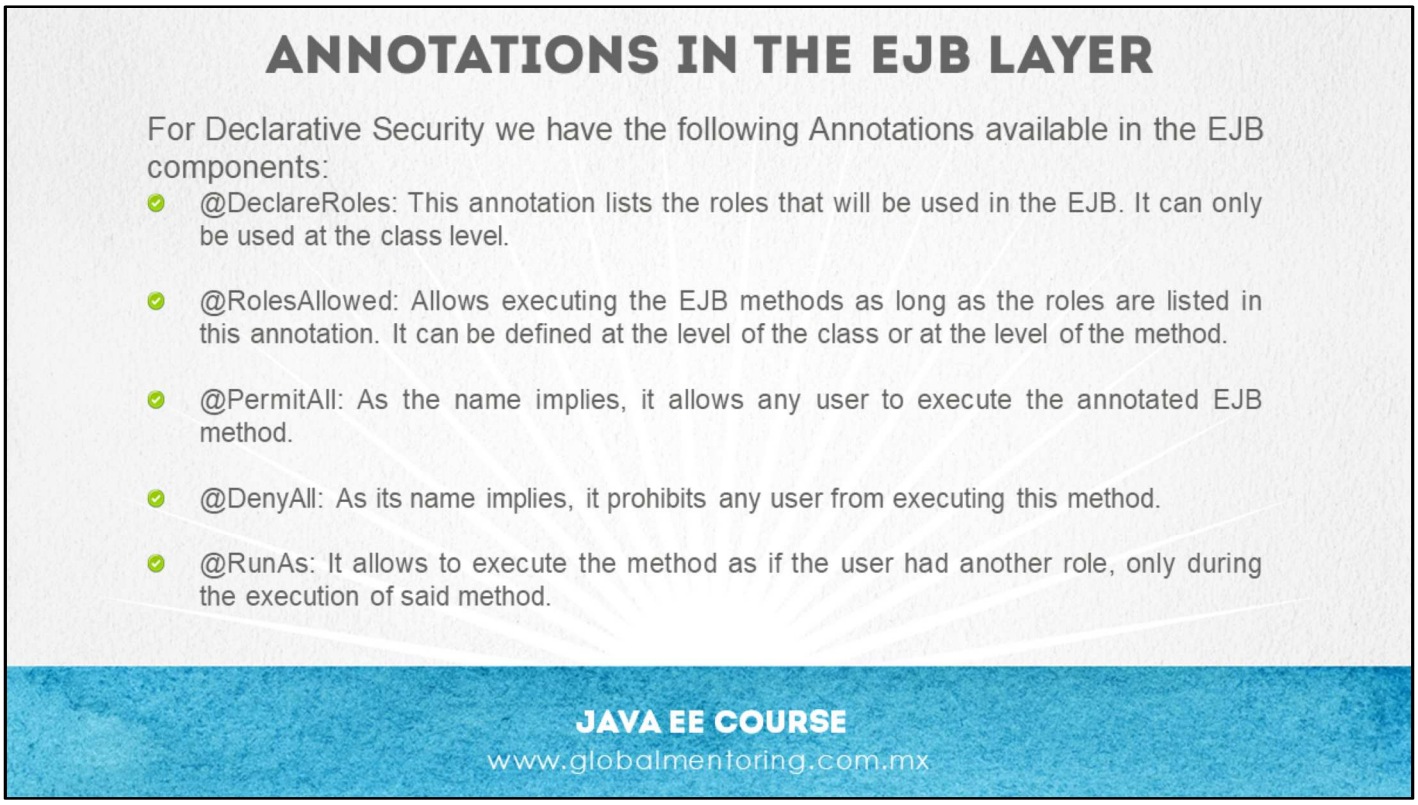
Declarative Security: We indicate to the container the type of validation desired, through annotations or xml configuration files. The container takes over most of the validation, authentication and authorization tasks.

In the case of declarative security, it is possible to apply it at the level of the class or at the level of each method. The container verifies the assigned role of the previously authenticated user, and if it has the requested role, then it allows executing the respective EJB method.

To handle the Declarative Security is very common that we use annotations, which we will study in the following sheet.

Programmatic Security: There are situations in which we need more control over the way in which authentication and / or authorization is performed, for example, at the user or group level. Programmatic security can be combined with declarative programming to increase control over security requirements in the system.

Programmatic security helps us when we have more detailed requirements, which would be complicated or impossible to achieve with Declarative Security. The good news is that we can combine Declarative Security in conjunction with Programmatic Security, so that in cases where we need to perform more detailed validations, including for each user, we can combine the power of both security options.



For Declarative Security we have the following Annotations available in the EJB components:

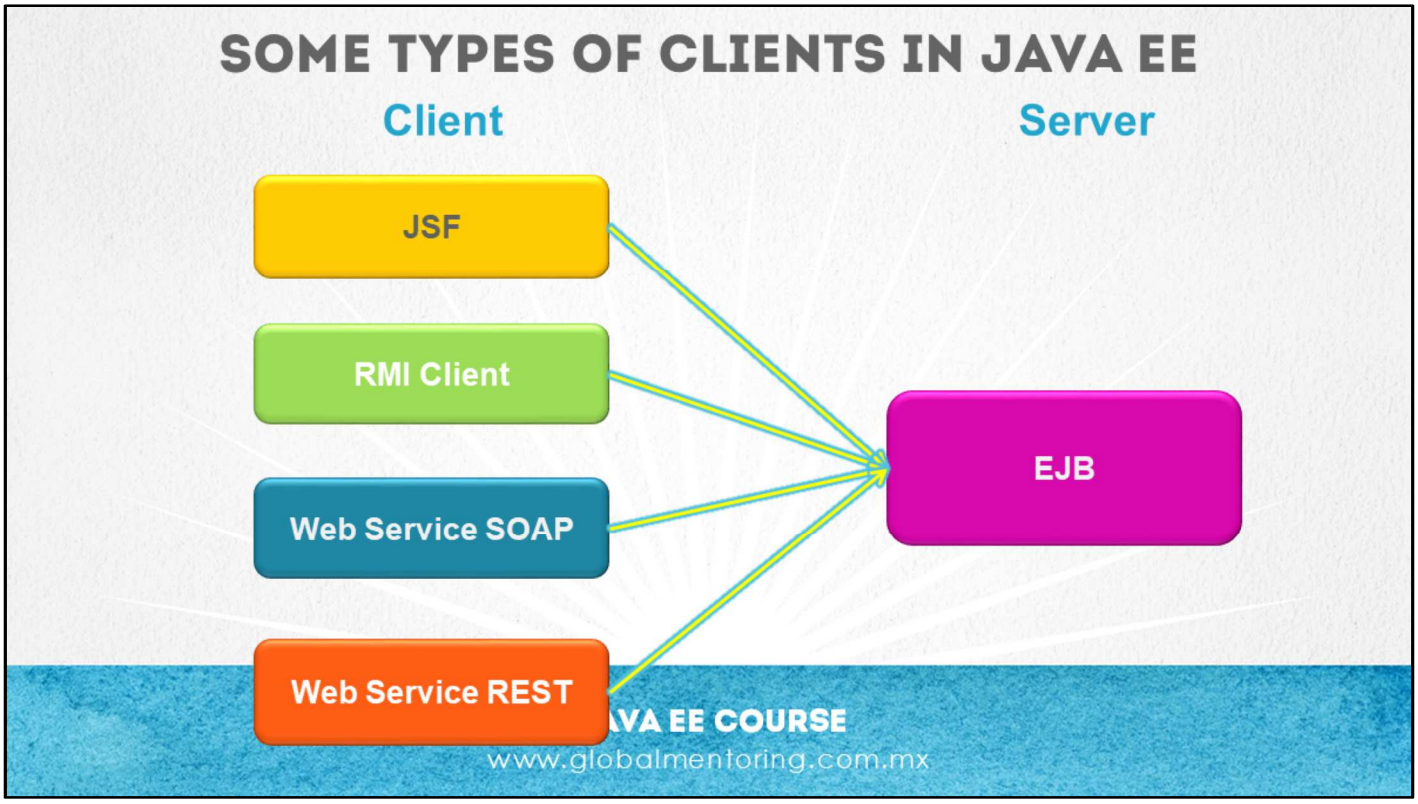
- @DeclareRoles: This annotation lists the roles that will be used in the EJB. It can only be used at the class level. If this annotation is not provided, the container will search the roles defined by the @RolesAllowed annotation and build a list of roles. It is recommended to add it because in some containers, when not adding it, we must add extra configuration to detect the roles we are handling. If the EJB extends from another class, the list of roles is concatenated, that is, they are also included.

- @RolesAllowed: Allows executing the EJB methods as long as the roles are listed in this annotation. It can be defined at the level of the class or at the level of the method. When it is defined at the class level, we are indicating to the container that the listed roles can execute any EJB method. If we add this annotation at the method level, we overwrite any behavior defined at the class level. However, we must be careful when combining roles at the class level and method level to avoid that mixing or overwriting of roles becomes a problem.

- @PermitAll: As the name implies, it allows any user to execute the annotated EJB method. This annotation should be used with caution, as it can be a safety hole if it is misused.

- @DenyAll: As its name implies, it prohibits any user from executing this method. This annotation causes the method to be unusable, however to disable certain functionality of the system may be a possible solution.

- @RunAs: It allows to execute the method as if the user had another role, only during the execution of said method. This allows a user with a role with less privileges, to assign more privileges, due to the role change, only for the execution of the method in question.



When we talk about a business application, we can have different clients interested in obtaining information from our system. This information is commonly exposed through the business layer or EJB.

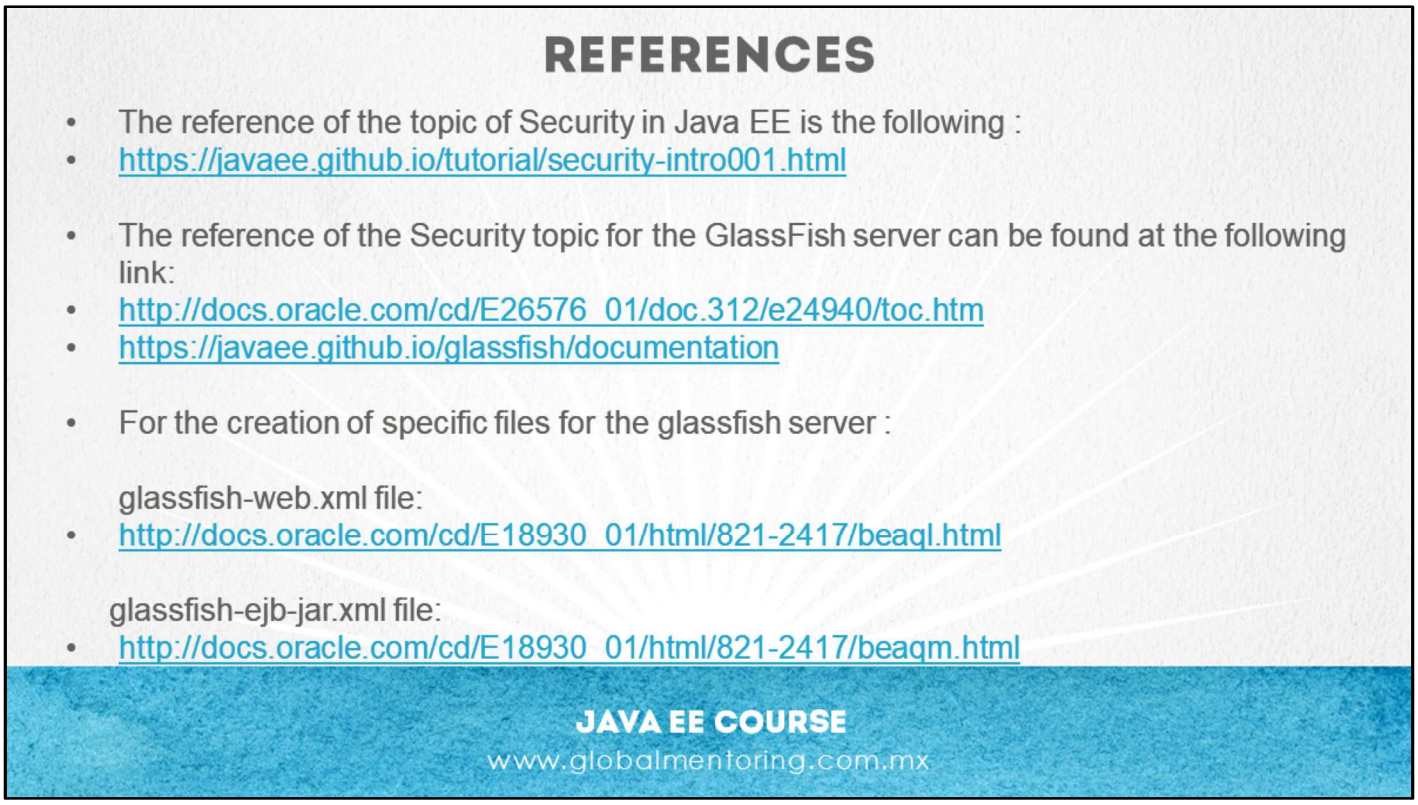
As we can see in the figure, we can have Web client, RMI, SOAP Web Service, REST Web Service, among others.

Unfortunately the authentication mechanisms between different clients today are not standard, and many times we still depend on the implementation of the Java server that we are using to be able to communicate each client and be able to authenticate successfully with the Java server.

In the exercise we will perform in this lesson we will include the authentication of each of the clients shown in the figure, since each one has different mechanisms, even in some cases it is necessary to use classes provided by the application server that we are using, for what the solution may depend on the application server in question.

The case where there are more configuration steps is the Web Client, since in turn, as we have seen, it works as an authentication point of the system, therefore the propagation of the Main object between the Web layer and the EJB layer works from Automatic way when using security in Java EE.

As a conclusion, we can observe the way in which different technologies and types of clients are forced to identify themselves before the system before being able to access any resource that has added security. This allows to create robust, flexible, easier to maintain and of course secure business applications.



- The reference of the topic of Security in Java EE is the following :
- https://javaee.github.io/tutorial/security-intro001.html

- The reference of the Security topic for the GlassFish server can be found at the following link:
- http://docs.oracle.com/cd/E26576_01/doc.312/e24940/toc.htm
- https://javaee.github.io/glassfish/documentation

- For the creation of specific files for the glassfish server :

glassfish-web.xml file:

- http://docs.oracle.com/cd/E18930_01/html/821-2417/beaql.html

glassfish-ejb-jar.xml file:

- http://docs.oracle.com/cd/E18930_01/html/821-2417/beaqm.html