

Tema 7

JAAS

Plataformas de Software Empresariales
Grado en Ingeniería Informática de Servicios y Aplicaciones
Curso 2021/2022

Introducción

▶ Objetivos

- Conocer el API para la autenticación y autorización en Java EE (JAAS)
- Crear un registro básico
- Crear un login básico
- Gestionar la aplicación para distintos roles

Introducción

- ▶ JAAS → Java Authentication and Authorization Service
- ▶ API básico para la seguridad en Java EE
- ▶ Nos proporciona mecanismos para gestionar la autenticación y la autorización de usuarios:
 - Conocer quién está ejecutando el código Java
 - Garantizar que quien ejecuta ese código tiene los permisos necesarios para hacerlo
- ▶ Vamos a utilizar distintos ficheros xml para configurar los permisos de seguridad
- ▶ La gestión de la seguridad la va a realizar directamente el servidor de aplicaciones (con un security realm)
 - Esto nos permite poder tener la misma seguridad para distintas aplicaciones empresariales de una manera centralizada y evitando duplicaciones

Introducción

- ▶ JAAS permite utilizar distintos mecanismo para almacenar las credenciales de los usuarios: bases de datos, ficheros, servidor LDAP, etc...
- ▶ En general, lo más directo es usar una base de datos que almacene usuarios y passwords
- ▶ Esto se conoce como **JDBC security realm de Payara**
- ▶ **Tenemos que utilizar dos tablas** para crear un security realm, la primera para almacenar las credenciales de usuario y la segunda para mapear usuarios con roles.
- ▶ **Los roles no están predefinidos**, así que podemos crear los roles que sean necesarios para nuestra aplicación

Proyecto de desarrollo

Vamos a incluir autenticación y autorización
de usuarios en la aplicación de cines

Proyecto de desarrollo

▶ Pasos:

- Crear las tablas de la base de datos
- Crear las clases entidad
- Configurar el fichero persistence.xml
- Construir la lógica de negocio
 - UserEJB y AuthenticationUtils
- Configurar el Security Realm de Glassfish
- Incluir información de roles y seguridad en glassfish-web.xml y web.xml
- Construir la interfaz de usuario
 - Crear página de registro (interfaz y beans correspondientes)
 - Crear página de login (interfaz y beans correspondientes)
 - Crear función de logout
 - Crear páginas privadas para cada rol
 - Actualizar template (para filtrar funcionalidad por rol, etc...)

Proyecto cines (vista general)

registro.xhtml

login.xhtml

template.xhtml

Menú renderizado según roles de usuarios

regok.xhtml

EmailValidator

```
public void validate {...}
```

RegisterView (backing bean)

```
private String name;
private String email;
private String password;
private String confirmPassword;

public void validatePassword(...) {
...
}
public String register() {
...
}
```

LoginView (backing bean)

```
private String email;
private String password;

public String login() {
...
}
public String logout() {
...
}
```

Otros

glassfish-web.xml
web.xml
persistence.xml

AuthenticationUtils

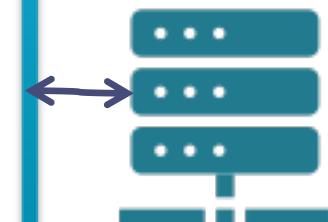
```
encodeSHA256
```

UserEJB

```
public Users
createUser(Users user) {
...
}
public Users
findByEmail(String email)
{
...
}
```

Entities

users.java
user_group.java



Glassfish Realm

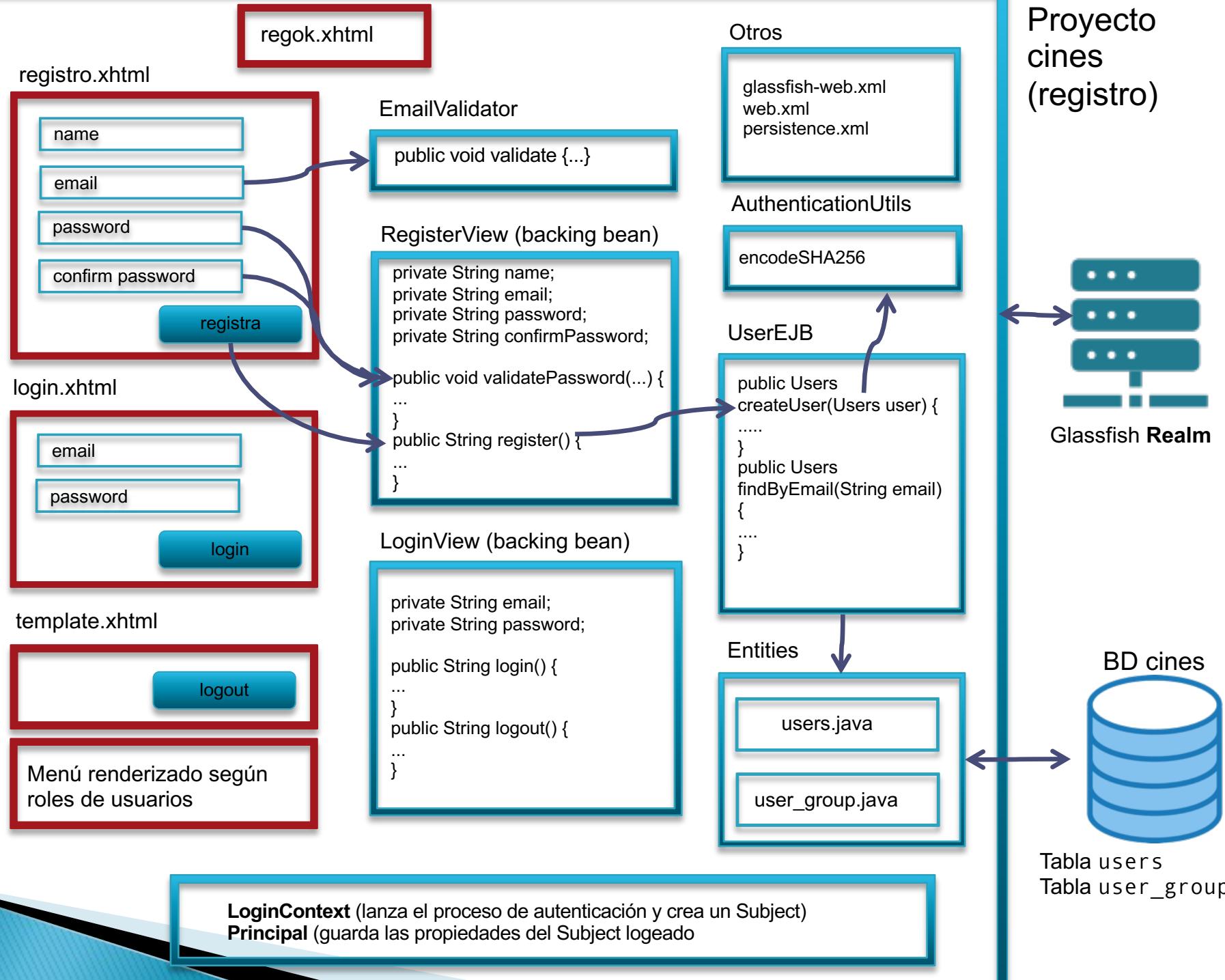
BD cines



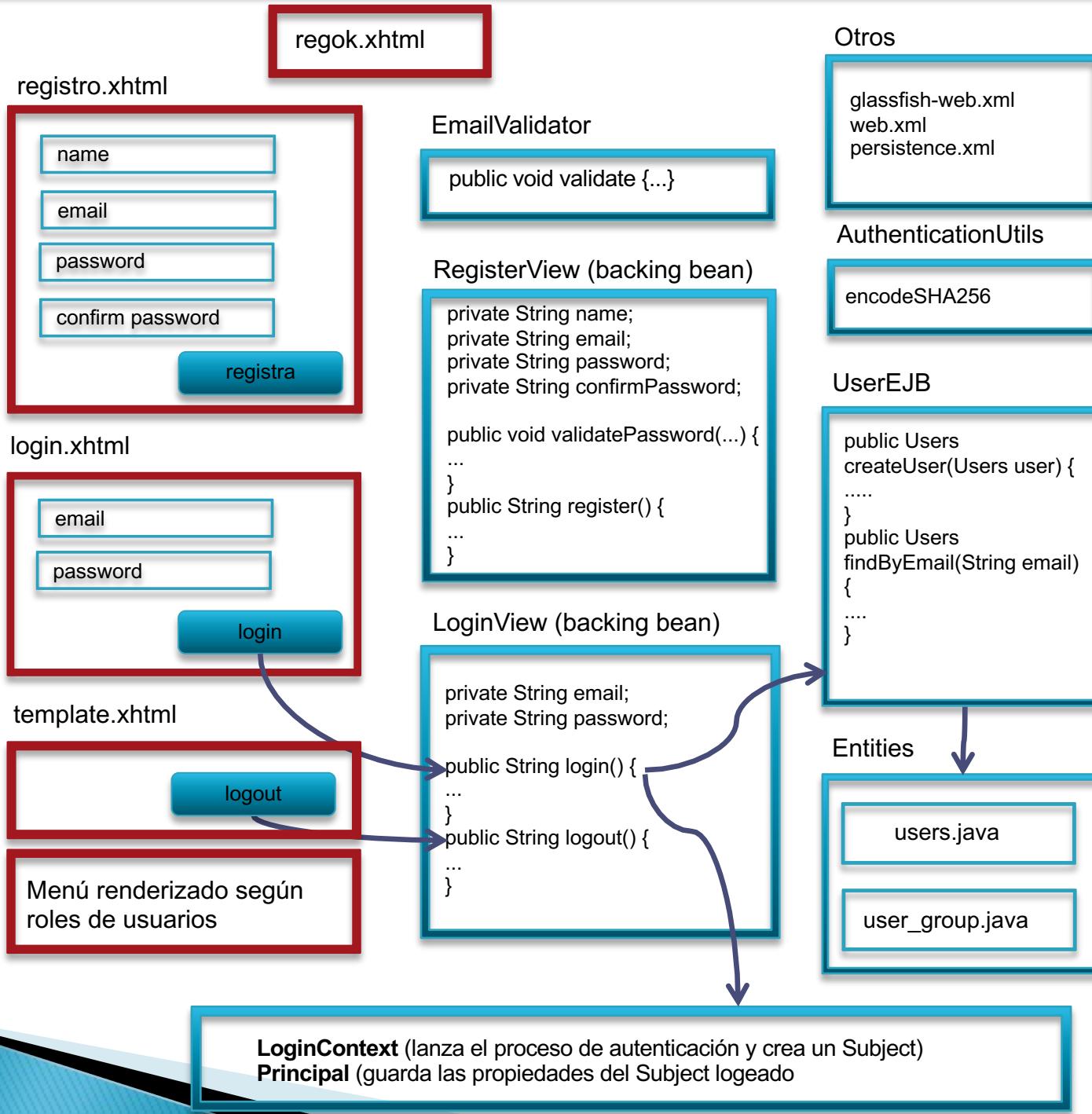
Tabla users
Tabla user_group

LoginContext (lanza el proceso de autenticación y crea un Subject)
Principal (guarda las propiedades del Subject logeado)

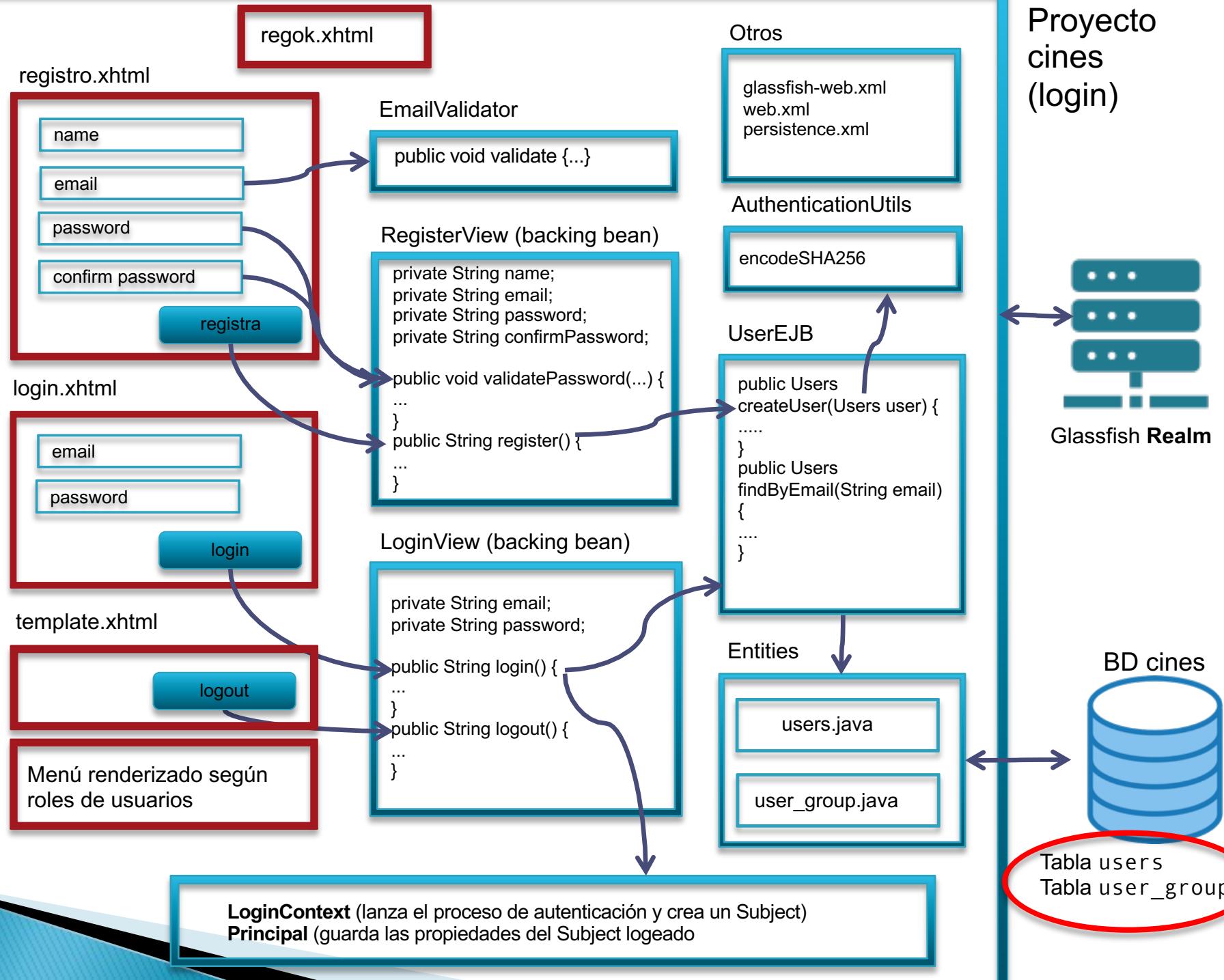
Proyecto cines (registro)



Proyecto cines (login)



Proyecto cines (login)



Crear tablas BD

- ▶ Necesitamos crear dos tablas en la base de datos, una con las credenciales de usuario y otra que mapee los usuarios con sus roles
- ▶ La tabla de usuario sólo tiene dos campos obligatorios: id de usuario y password
 - Pero podemos añadir más atributos si es necesario, en el ejemplo he añadido el nombre del usuario
- ▶ La tabla de grupos tiene dos campos obligatorios: id de usuario y grupo (rol)
 - Nosotros definimos los roles (no están predefinidos por defecto)
 - En el ejemplo consideraremos “users” y “admin” como roles
 - Los privilegios de cada rol se asignan posteriormente de manera programática

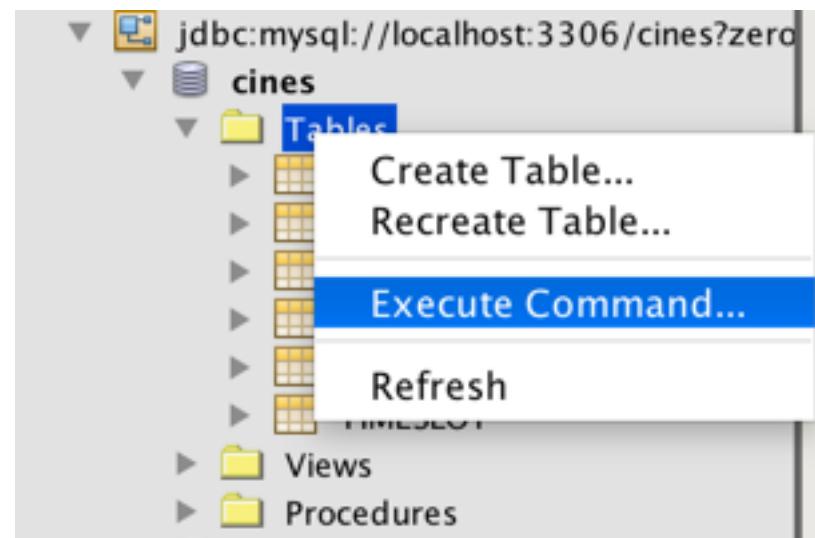
Crear tablas BD

- ▶ Creamos las tablas para usuarios y roles

```
CREATE TABLE users (
    email varchar(255) NOT NULL,
    password varchar(64) NOT NULL,
    name varchar(30) NOT NULL,
    PRIMARY KEY (email)
);
```

```
CREATE TABLE user_groups (
    email VARCHAR(255) NOT NULL,
    groupname VARCHAR(32) NOT NULL,
    PRIMARY KEY (email)
);
```

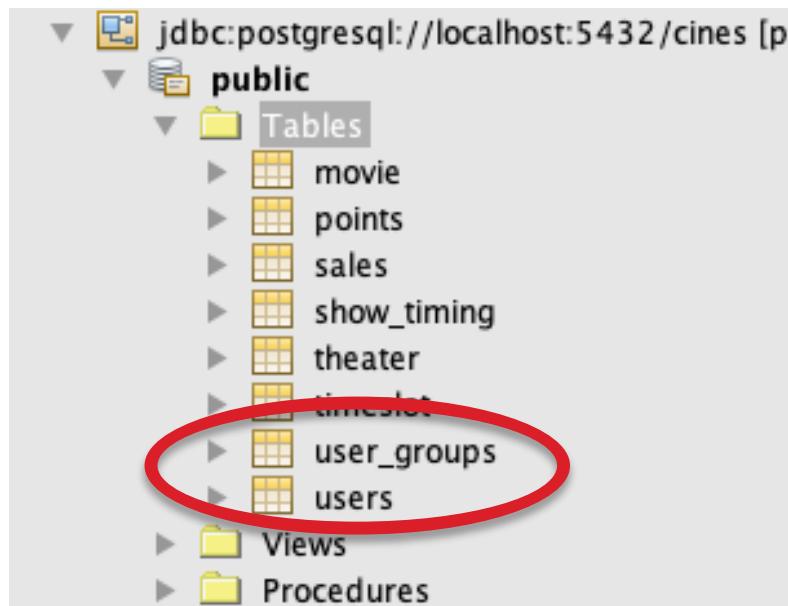
Crear tablas BD



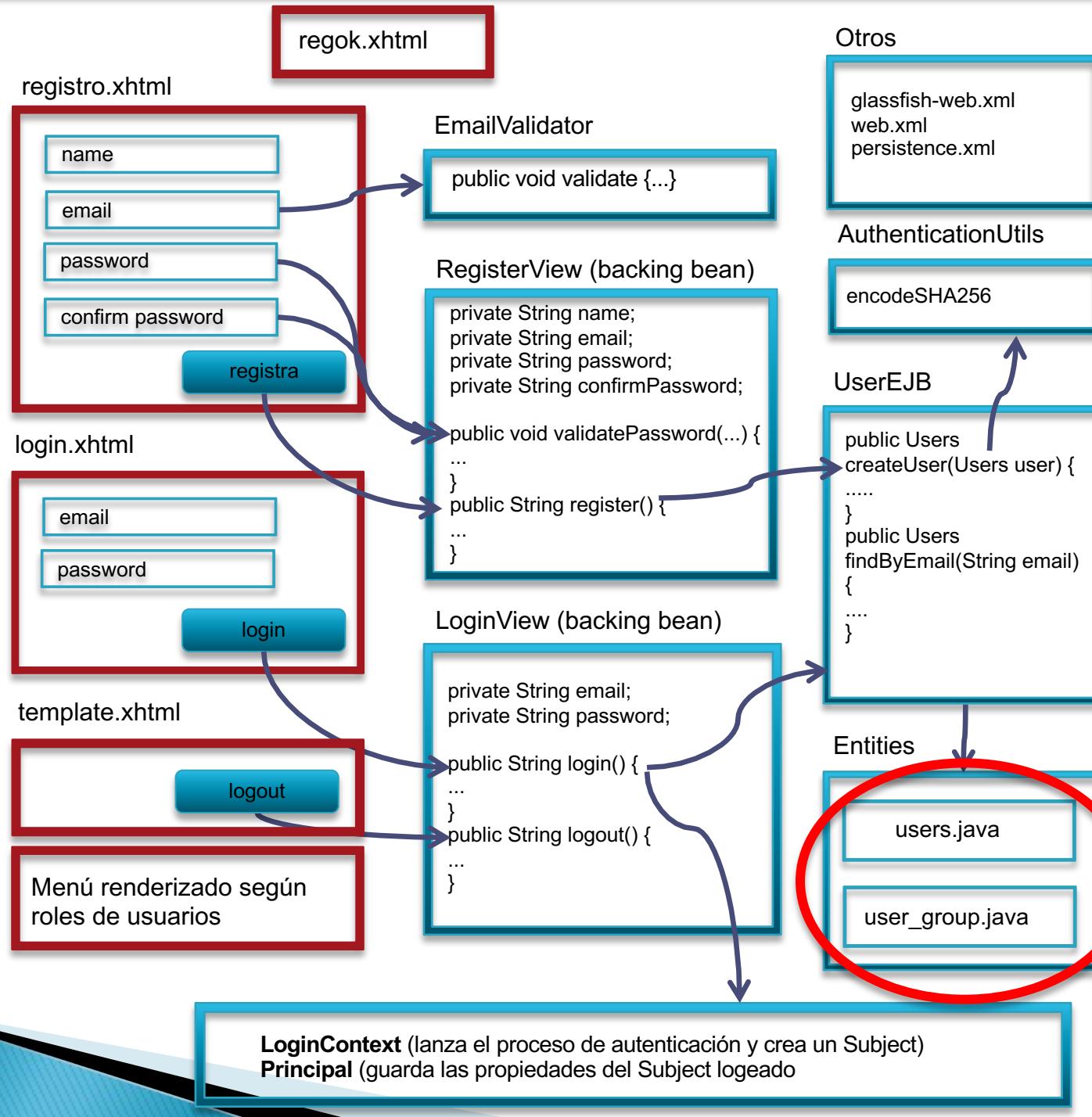
Crear tablas BD

```
Connection: jdbc:postgresql://localhost:5432/cines [postgres on ... ▾]  
1 CREATE TABLE users (  
2     email varchar(255) NOT NULL,  
3     password varchar(64) NOT NULL,  
4     name varchar(30) NOT NULL,  
5     PRIMARY KEY (email)  
6 );  
7  
8 CREATE TABLE user_groups (  
9     email VARCHAR(255) NOT NULL,  
10    groupname VARCHAR(32) NOT NULL,  
11    PRIMARY KEY (email)  
12 );  
13
```

Crear tablas BD

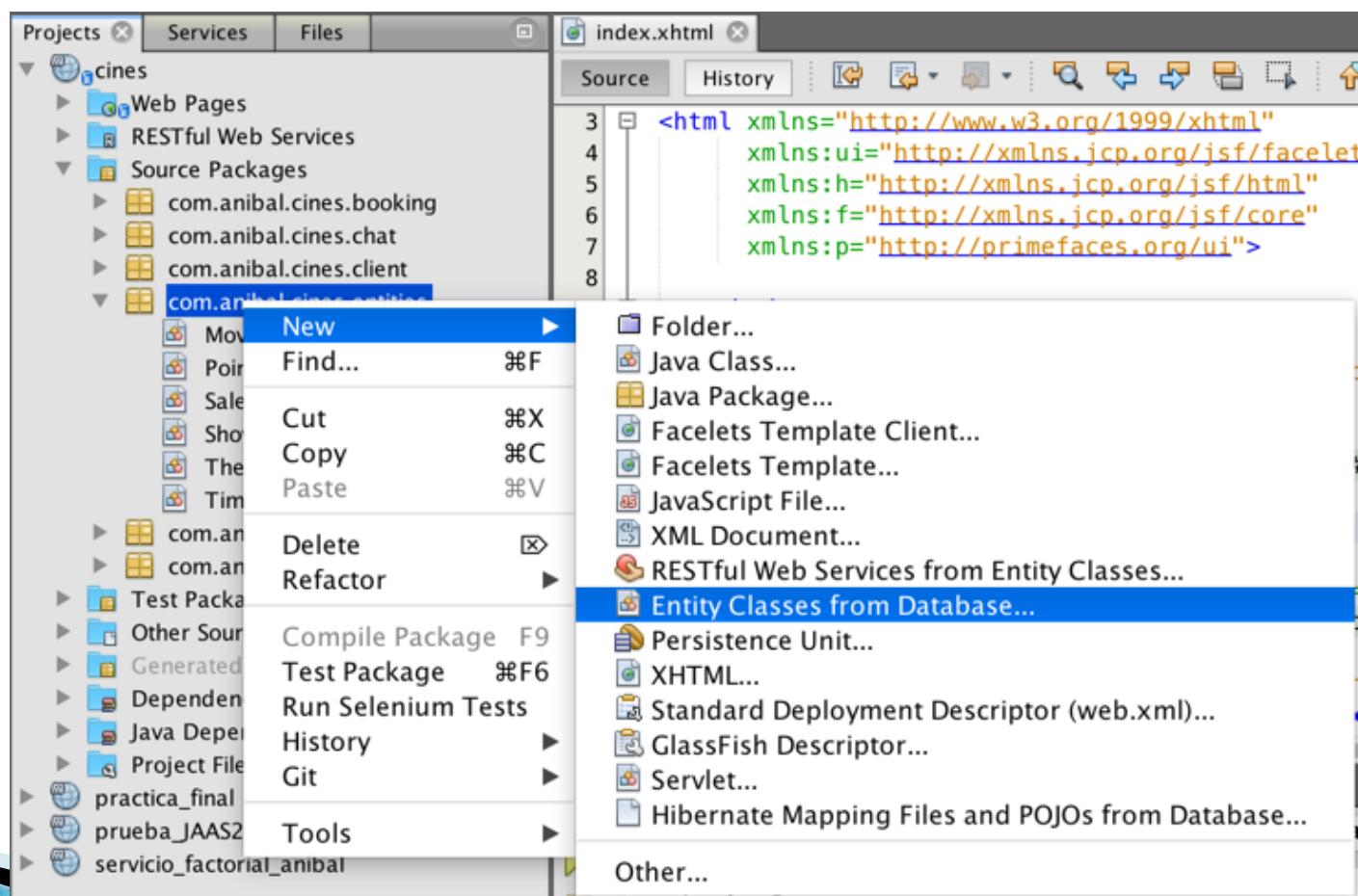


Proyecto cines (login)

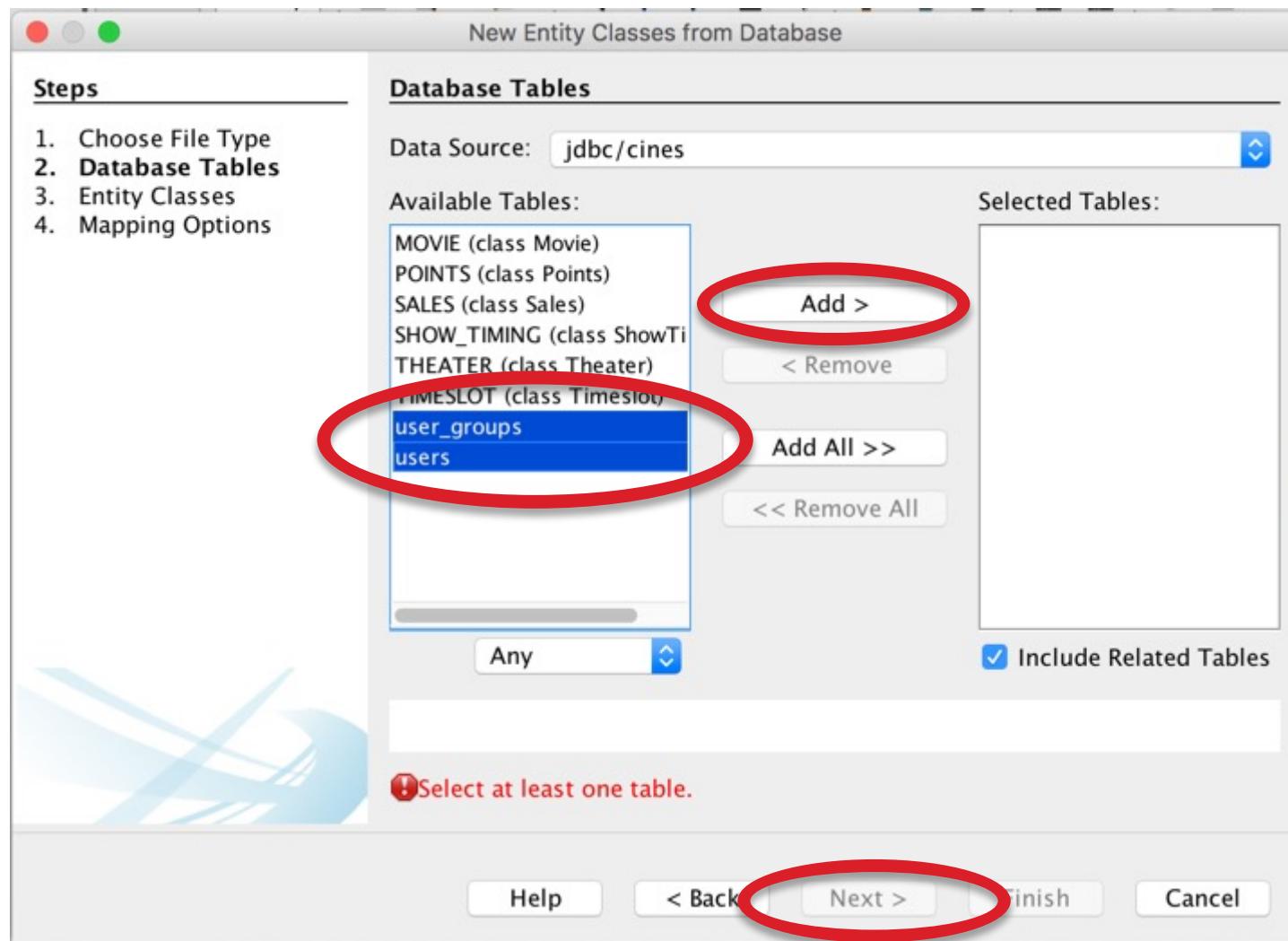


Crear clases entidad

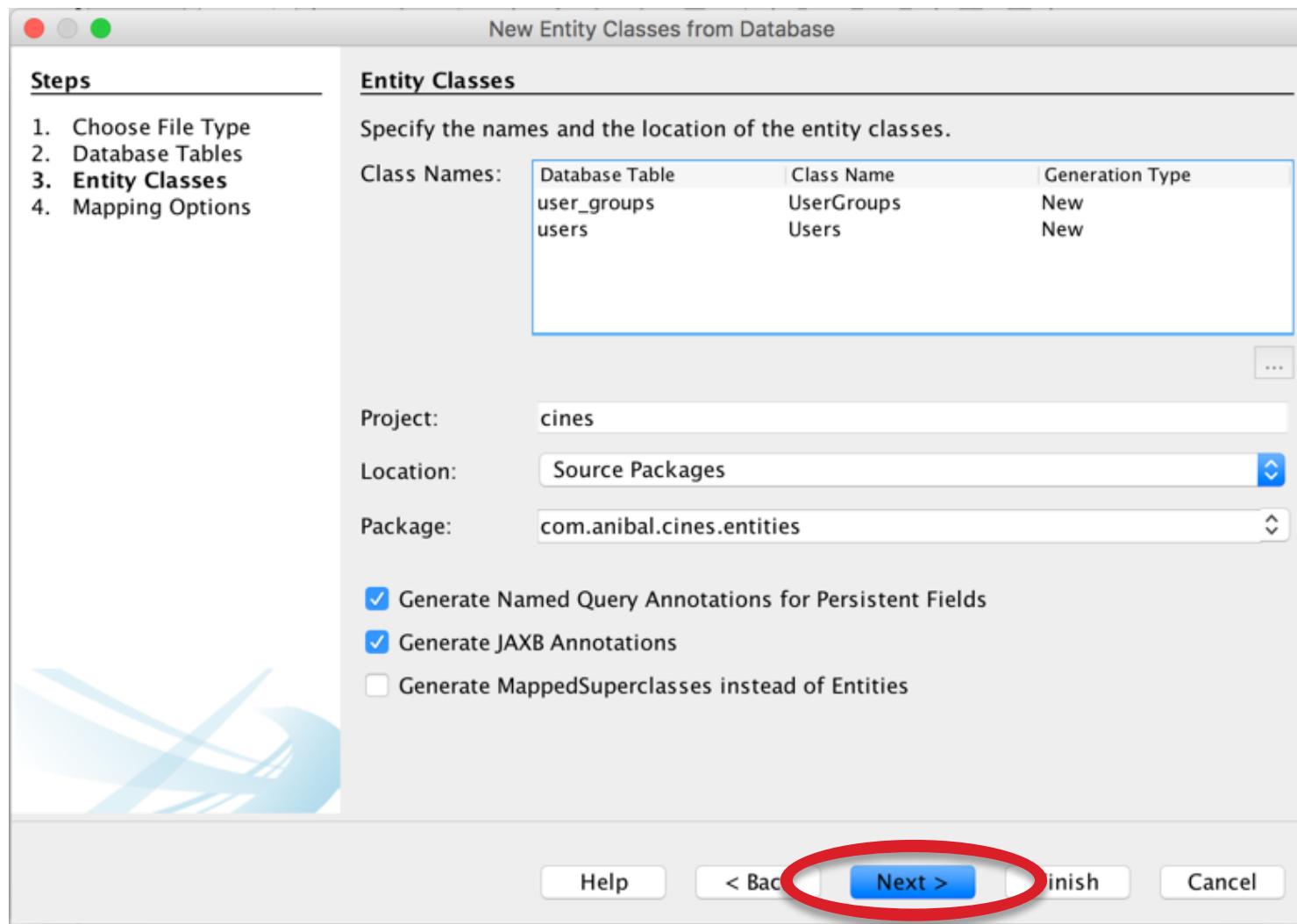
- ▶ Creamos automáticamente las clases entidad correspondientes a las tablas recién creadas



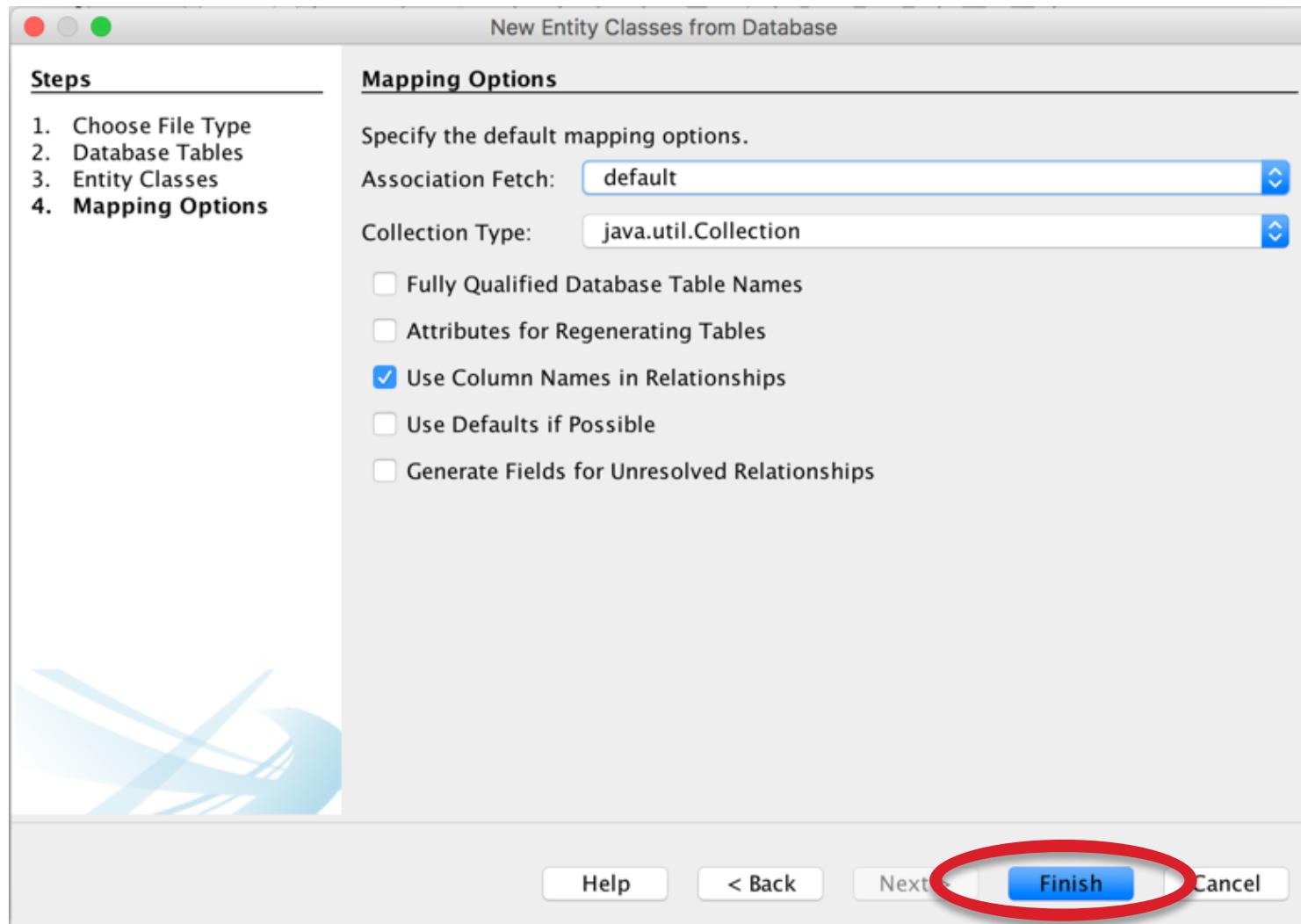
Crear clases entidad



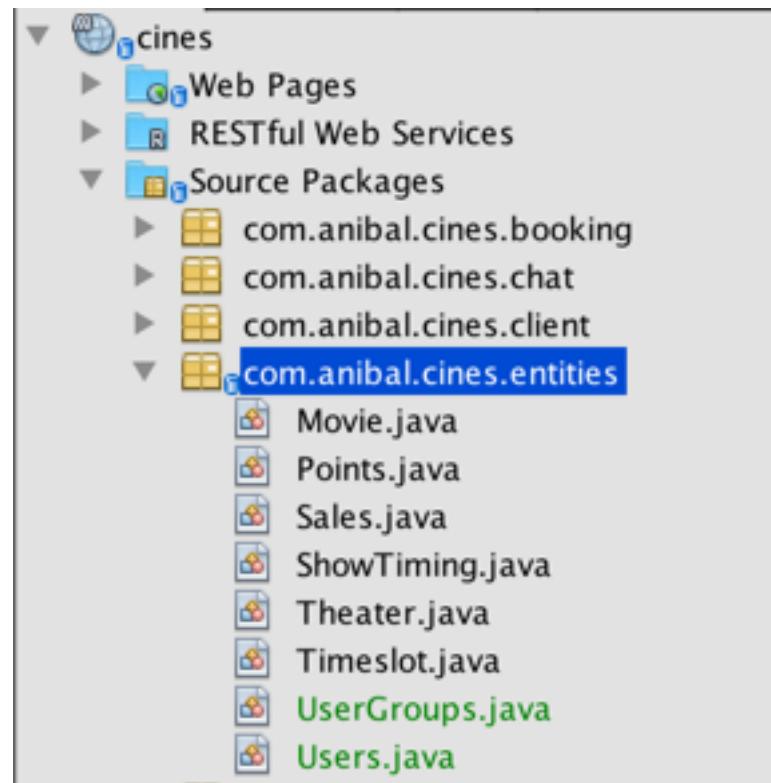
Crear clases entidad



Crear clases entidad



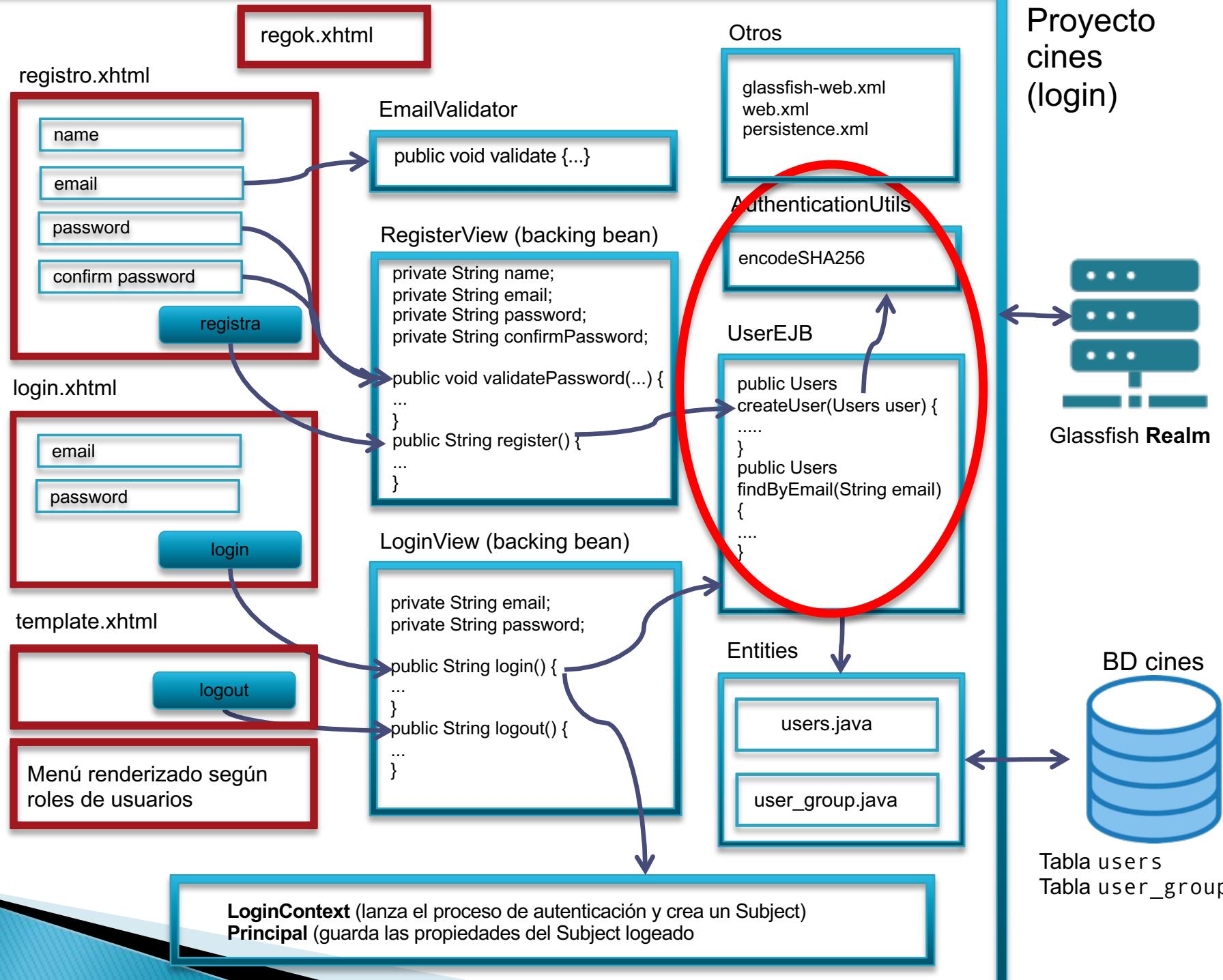
Crear clases entidad



Configurar persistence.xml

- ▶ En este fichero se especifican los atributos de la capa de persistencia
- ▶ En el Tema 3 vimos como crear un recurso JDBC con un pool de conexiones en Glassfish... si lo hemos creado de esa manera, aquí no tendríamos que hacer nada... de lo contrario (si lo hemos hecho automáticamente en local) tendríamos que crear la “persistence-unit” a mano y configurar las características correspondientes

Proyecto cines (login)

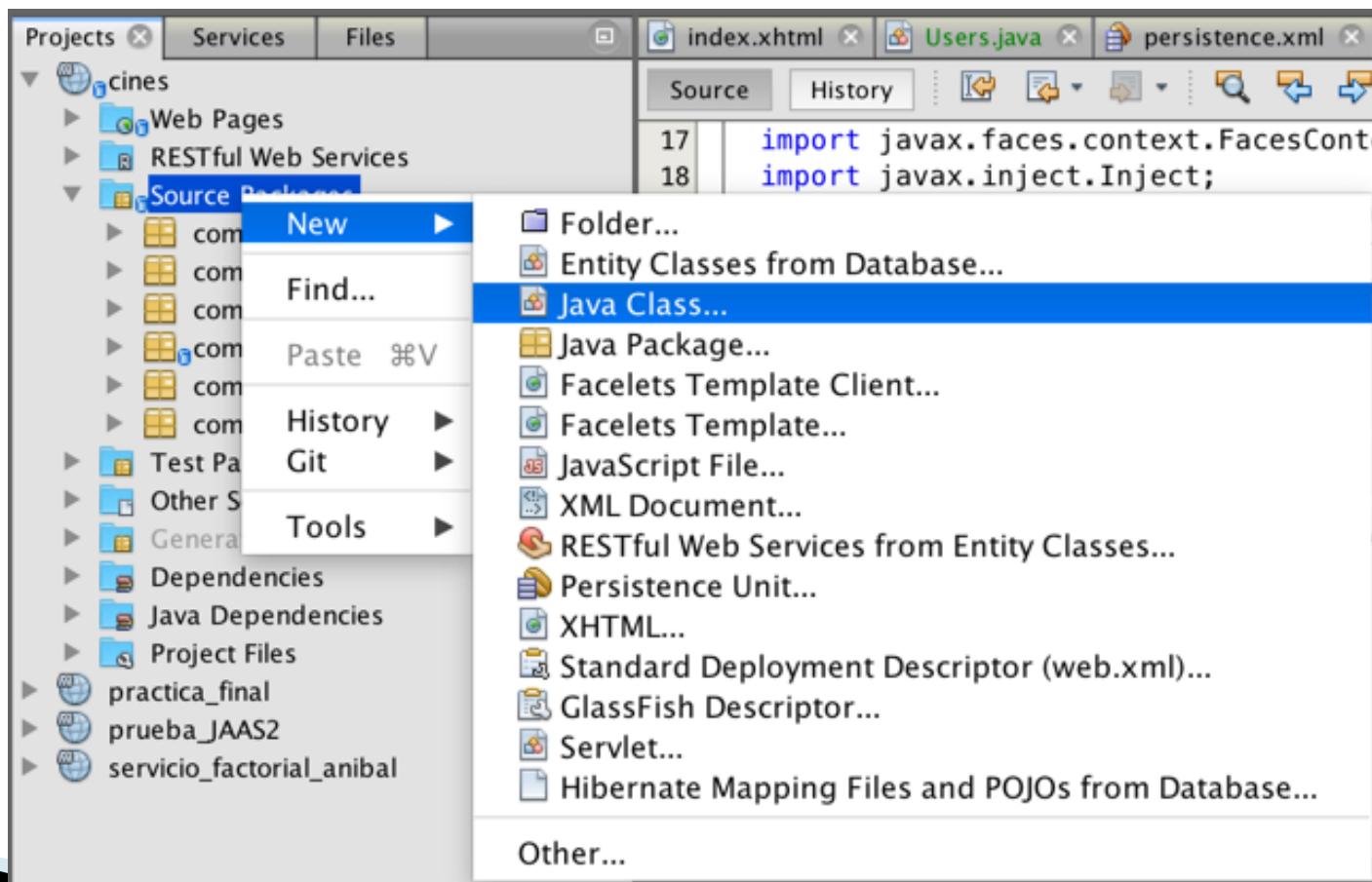


Construir la lógica de negocio

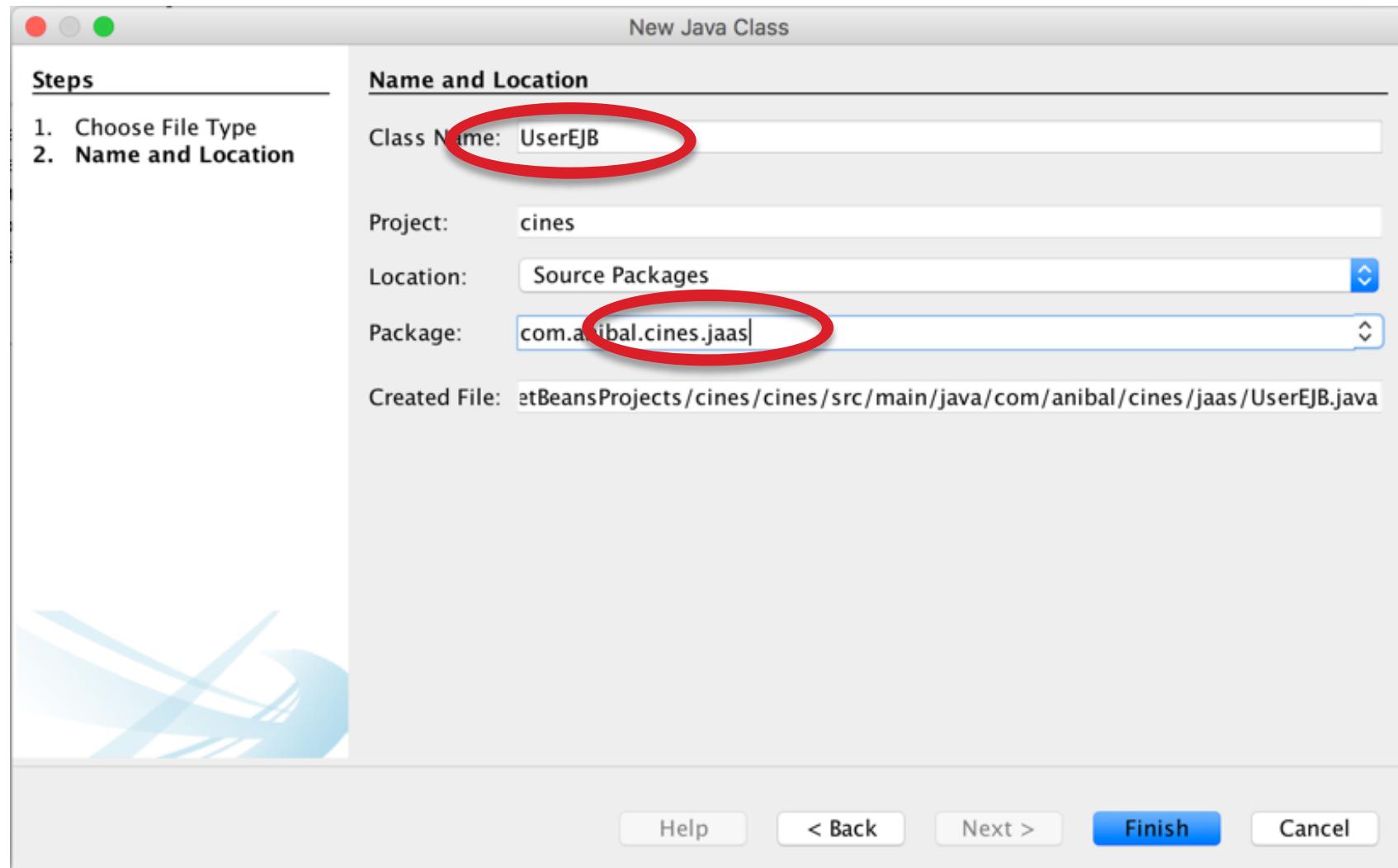
- ▶ El siguiente paso será crear el EJB que se encargue de insertar los usuarios en la base de datos (cuando se registre un nuevo usuario) y de buscarlos (cuando un usuario haga login)
- ▶ Para ello vamos a crear dos clases:
 - Una clase (UserEJB) con los métodos para insertar usuarios y buscarlos
 - Una clase (AuthenticationUtils) para convertir las contraseñas a SHA256

Construir la lógica de negocio – UserEJB

- ▶ Creamos una nueva clase llamada “UserEJB” y lo metemos dentro de un nuevo paquete llamado “jaas”



Construir la lógica de negocio – UserEJB



Construir la lógica de negocio – UserEJB

```
@Stateless  
public class UserEJB {  
  
    @PersistenceContext  
    private EntityManager em;  
  
    public Users createUser(Users user) {  
        try {  
            user.setPassword(AuthenticationUtils.encodeSHA256(user.getPassword()));  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        UserGroups group = new UserGroups();  
        group.setEmail(user.getEmail());  
        group.setGroupname("users");  
        em.persist(user);  
        em.persist(group);  
  
        return user;  
    }  
}
```

Construir la lógica de negocio – UserEJB

```
public Users findByEmail(String email) {  
    TypedQuery<Users> query = em.createNamedQuery("Users.findByEmail",  
Users.class);  
    query.setParameter("email", email);  
    Users user = null;  
    try {  
        user = query.getSingleResult();  
    } catch (Exception e) {  
  
    }  
    return user;  
}
```

Construir la lógica de negocio – UserEJB

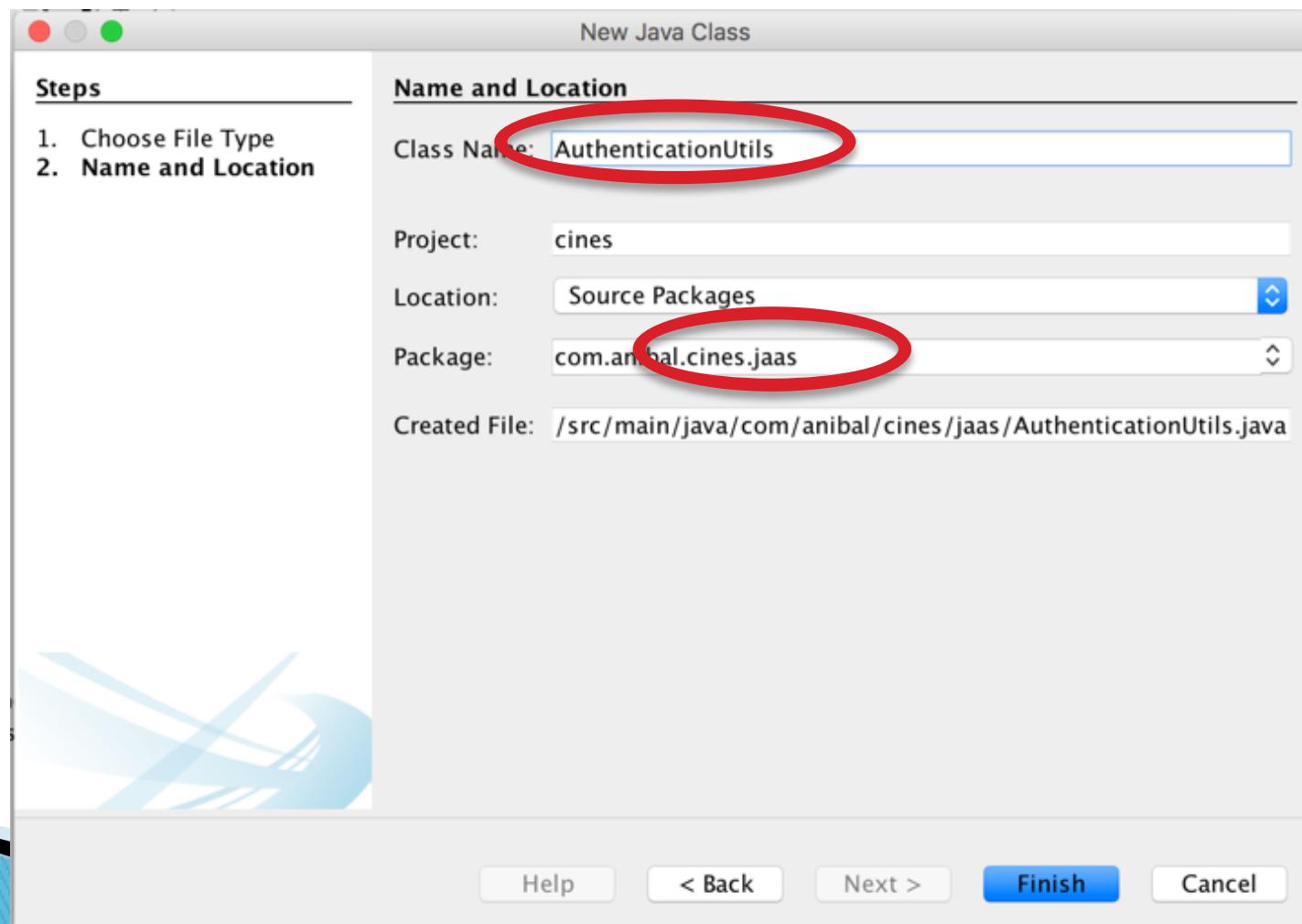
- ▶ `@Stateless`
 - EJB sin estado
- ▶ `@PersistenceContext`
`private EntityManager em;`
 - Inyectamos el contexto de persistencia para hacer llamadas a la base de datos (crear usuario y buscar usuario)
- ▶ `public Users createUser(Users user) {`
 - Método para crear un usuario
- ▶ `user.setPassword(AuthenticationUtils.encodeSHA256(user.getPassword()));`
 - Codificamos el password introducido por el usuario con SHA256

Construir la lógica de negocio – UserEJB

- ▶ `UserGroups group = new UserGroups();`
 - Creamos un nuevo objeto de la entidad UserGroups para introducir en la base de datos el id/rol del usuario creado
- ▶ `group.setGroupname("users");`
 - Ponemos al usuario recién creado el rol “users”
 - Aquí pensad cómo habría que hacerlo para la práctica final
- ▶ `TypedQuery<Users> query = em.createNamedQuery("Users.findByEmail", Users.class);`
 - Devolverá un usuario (si existe en la base de datos) o NULL
 - Por esto es importante poner un try/catch, ya que la llamada a getSingleResult() lanzará una excepción
 - El NULL lo trataremos más adelante en el bean correspondiente

Construir la lógica de negocio – AuthenticationUtils

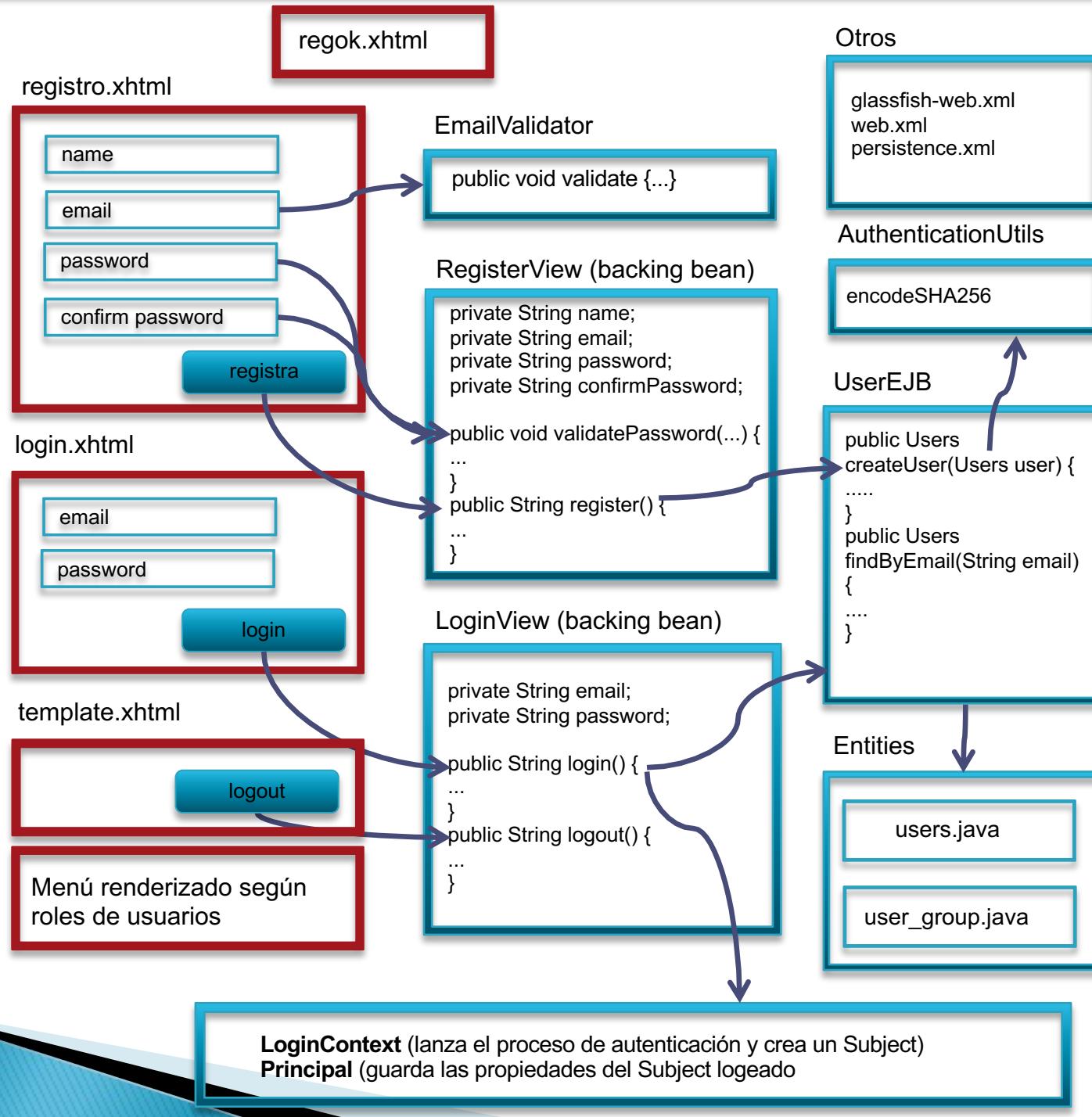
- ▶ Creamos una nueva clase llamada “AuthenticationUtils” y lo metemos dentro del paquete “jaas”



Construir la lógica de negocio – AuthenticationUtils

```
public static String encodeSHA256(String password) throws  
UnsupportedEncodingException, NoSuchAlgorithmException {  
    MessageDigest md = MessageDigest.getInstance("SHA-256");  
    md.update(password.getBytes("UTF-8"));  
    byte[] digest = md.digest();  
    return DatatypeConverter.printBase64Binary(digest);  
}
```

Proyecto cines (login)



Configurar el Security Realm de Glassfish

- ▶ Para que JAAS funcione será necesario crear un “realm” de seguridad en Glassfish
- ▶ Glassfish utiliza el realm para automatizar todo el proceso de autenticación/autorización, como por ejemplo saber en qué tablas de la base de datos está la información de los usuarios/roles o la codificación a utilizar
- ▶ Para ello tendremos que ir a la consola de administración (localhost:4848)
- ▶ En Configurations/server-config/Security/Realms pulsamos en “New...”

Configurar el Security Realm de Glassfish

User: admin Domain: domain1 Server: localhost

 Payara® SERVER

server-config

- Admin Service
- Availability Service
- Batch
- Connector Service
- Data Grid
- EJB Container
- HealthCheck
- HTTP Service
- JVM Settings
- Java Message Service
- Logger Settings
- MicroProfile
- Monitoring
- Network Config
- Notification
- ORB
- Request Tracing
- Security
- Admin Audit
- Realms

Realms

Create, modify, or delete security (authentication) realms.

Configuration Name: server-config

Realms (3)

| Select | Name | Class Name |
|--------------------------|-------------|---|
| <input type="checkbox"/> | admin-realm | com.sun.enterprise.security.auth.realm.file.FileRealm |
| <input type="checkbox"/> | certificate | com.sun.enterprise.security.auth.realm.certificate.CertificateRealm |
| <input type="checkbox"/> | file | com.sun.enterprise.security.auth.realm.file.FileRealm |

New...

The 'New...' button in the Realms table header is circled in red.

Configurar el Security Realm de Glassfish

User: admin Domain: domain1 Server: localhost

 Payara® SERVER

New Realm

Create a new security (authentication) realm. Valid realm types are PAM, OSGi, File, Certificate, LDAP, JDBC, Digest, Oracle Solaris, and Custom.

* Indicates required field

Configuration Name: server-config

Name: * jdbc-realm

Class Name: com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm

Choose a realm class name from the drop-down list or specify a custom class

Properties specific to this Class

JAAS Context: * jdbcRealm

Identifier for the login module to use for this realm

JNDI: * jdbc/cines

JNDI name of the JDBC resource used by this realm

User Table: * users

The 'Name' field (containing 'jdbc-realm') and the 'User Table' field (containing 'users') are circled in red.

Configurar el Security Realm de Glassfish

User: admin Domain: domain1 Server: localhost [Home](#)

 Payara® SERVER

server-config

- Admin Service
- Availability Service
- Batch
- Connector Service
- Data Grid
- EJB Container
- HealthCheck
- HTTP Service
- JVM Settings
- Java Message Service
- Logger Settings
- MicroProfile
- Monitoring
- Network Config
- Notification
- ORB
- Request Tracing
- Security
- Admin Audit
- Realms
- admin-realm
- certificate
- ...

User Table: * Name of the database table that contains the list of authorized users for this realm

User Name Column: * Name of the column in the user table that contains the list of user names

Password Column: * Name of the column in the user table that contains the user passwords

Group Table: * Name of the database table that contains the list of groups for this realm

Group Table User Name Column: Name of the column in the user group table that contains the list of groups for this realm

Group Name Column: * Name of the column in the group table that contains the list of group names

Assign Groups: Comma separated list of group names

Database User: Specify the database user name in the realm instead of the JDBC connection pool

Database Password:

A red circle highlights the "User Table" field.

Configurar el Security Realm de Glassfish

User: admin Domain: domain1 Server: localhost [Home](#)

 Payara® SERVER

server-config

- Admin Service
- Availability Service
- Batch
- Connector Service
- Data Grid
- EJB Container
- HealthCheck
- HTTP Service
- JVM Settings
- Java Message Service
 - Logger Settings
 - MicroProfile
 - Monitoring
 - Network Config
 - Notification
 - ORB
 - Request Tracing
- Security
 - Admin Audit
- Realms
 - admin-realm
 - certificate
 - file

Database User: [REDACTED] Specify the database user name in the realm instead of the JDBC connection pool

Database Password: [REDACTED] Specify the database password in the realm instead of the JDBC connection pool

Digest Algorithm: SHA-256 Digest algorithm (default is SHA-256; note that the default was MD5 in GlassFish versions prior to 3.1)

Encoding: Base64 Encoding (allowed values are Hex and Base64)

Charset: [REDACTED] Character set for the digest algorithm

Login module

Register Login Module: Update JAAS configuration in login.conf (when using non-default JAAS context of standard realm, or custom realm)

Additional Properties (0)

Add Property Delete Properties

| Select | Name | Value | Description |
|-----------------|------|-------|-------------|
| No items found. | | | |

Configurar el Security Realm de Glassfish

User: admin Domain: domain1 Server: localhost

Home About... Help Online Help Enable Asadmin Recorder

payara® SERVER

server-config

- Admin Service
- Availability Service
- Batch
- Connector Service
- Data Grid
- EJB Container
- HealthCheck
- HTTP Service
- JVM Settings
- Java Message Service
- Logger Settings
- MicroProfile
- Monitoring
- Network Config
- Notification
- ORB
- Request Tracing
- Security
 - Admin Audit
 - Realms
 - admin-realm
 - certificate

Database Password: [REDACTED]
Specify the database password in the realm instead of the JDBC connection pool

Digest Algorithm: SHA-256
Digest algorithm (default is SHA-256); note that the default was MD5 in GlassFish versions prior to 3.1

Encoding: Base64
Encoding (allowed values are Hex and Base64)

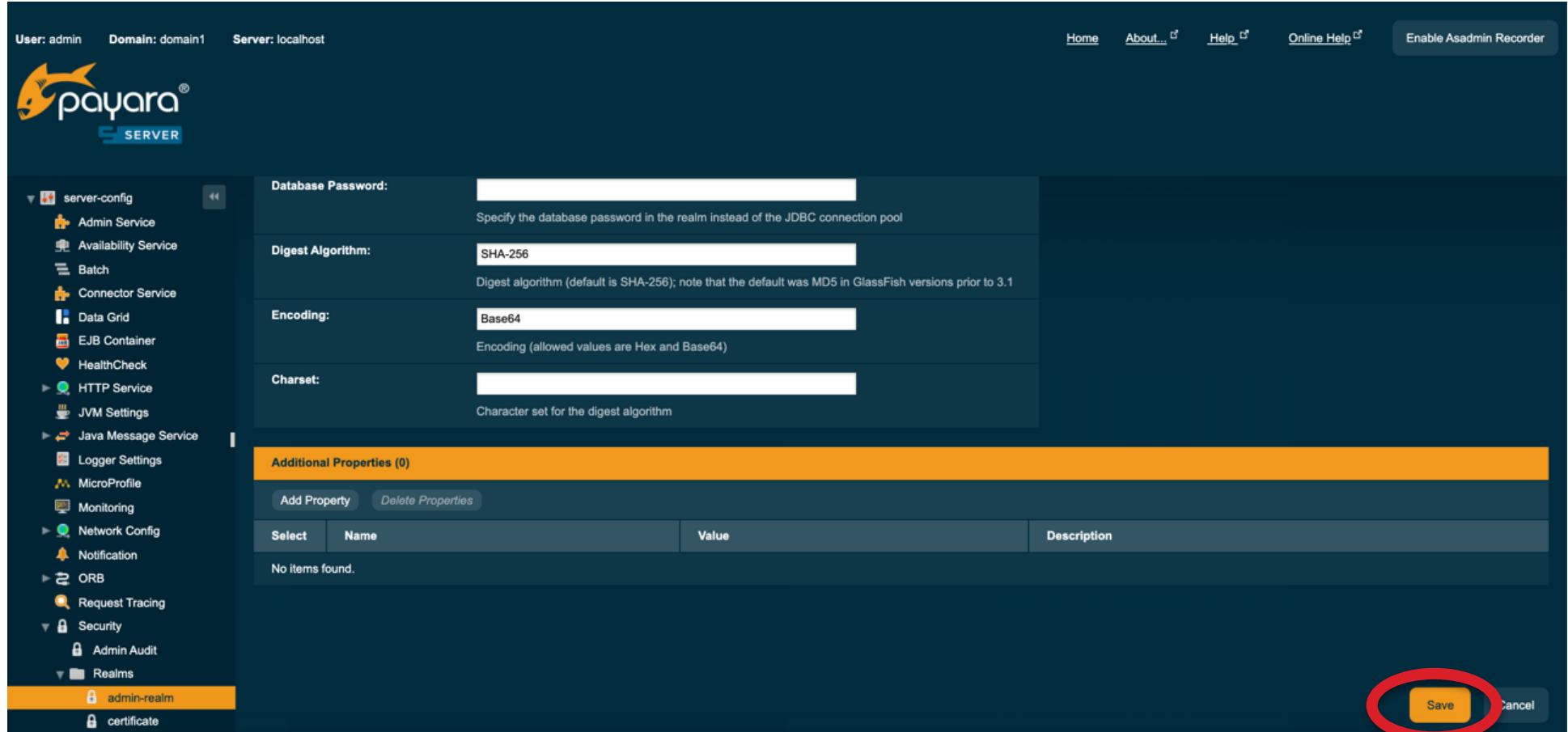
Charset: [REDACTED]
Character set for the digest algorithm

Additional Properties (0)

Add Property Delete Properties

| Select | Name | Value | Description |
|-----------------|------|-------|-------------|
| No items found. | | | |

Save Cancel



Configurar el Security Realm de Glassfish

User: admin Domain: domain1 Server: localhost

 Payara® SERVER

server-config

- Admin Service
- Availability Service
- Batch
- Connector Service
- Data Grid
- EJB Container
- HealthCheck
- HTTP Service
- JVM Settings
- Java Message Service
- Logger Settings
- MicroProfile
- Monitoring
- Network Config
- Notification
- ORB
- Request Tracing
- Security
 - Admin Audit

Realms

Create, modify, or delete security (authentication) realms.

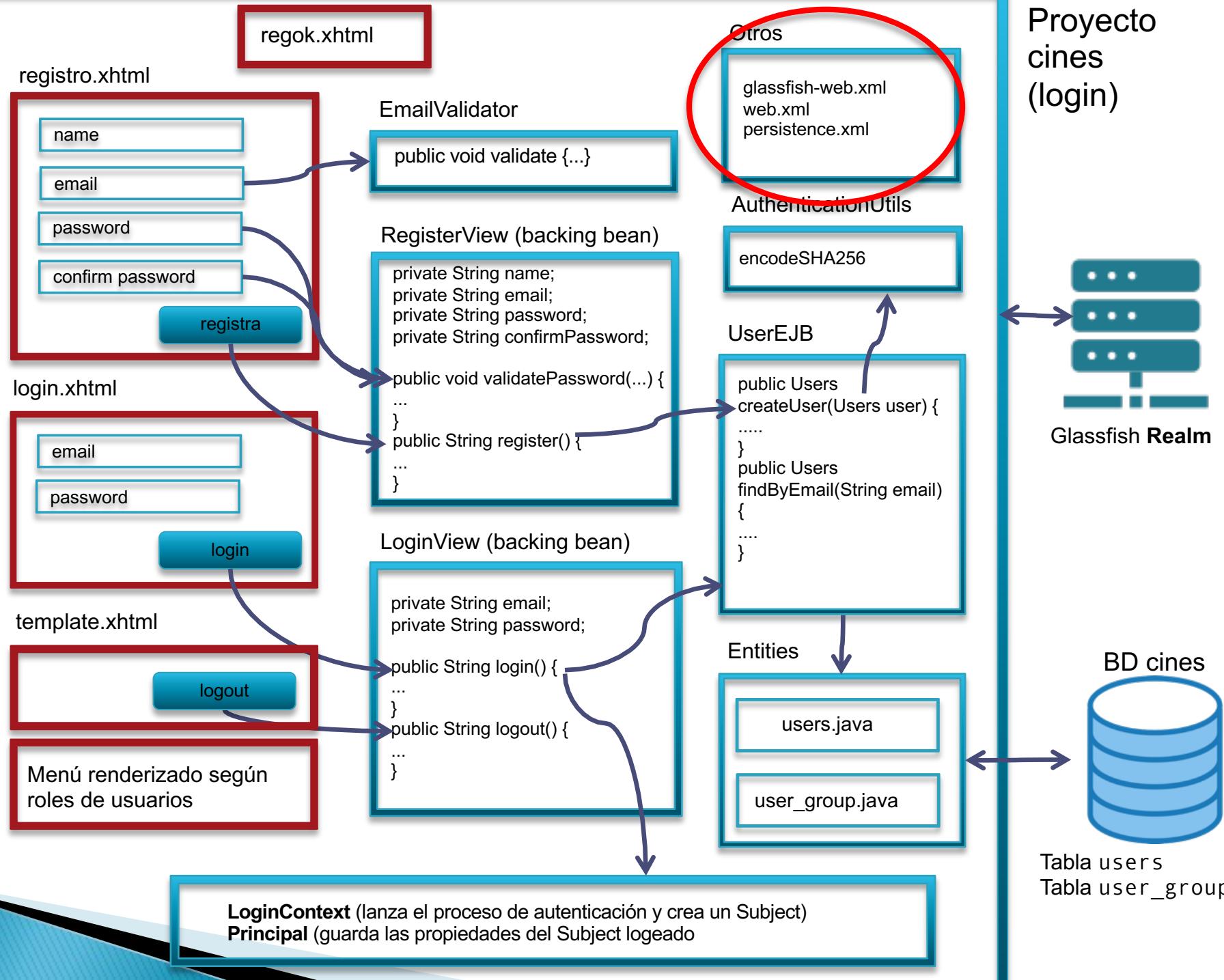
Configuration Name: server-config

Realms (4)

| Select | Name | Class Name |
|--------|-------------|---|
| | admin-realm | com.sun.enterprise.security.auth.realm.file.FileRealm |
| | certificate | com.sun.enterprise.security.auth.realm.certificate.CertificateRealm |
| | file | com.sun.enterprise.security.auth.realm.file.FileRealm |
| | jdbc-realm | com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm |

The "jdbc-realm" row is circled with a red oval.

Proyecto cines (login)



Incluir información de roles y seguridad en glassfish-web.xml y web.xml

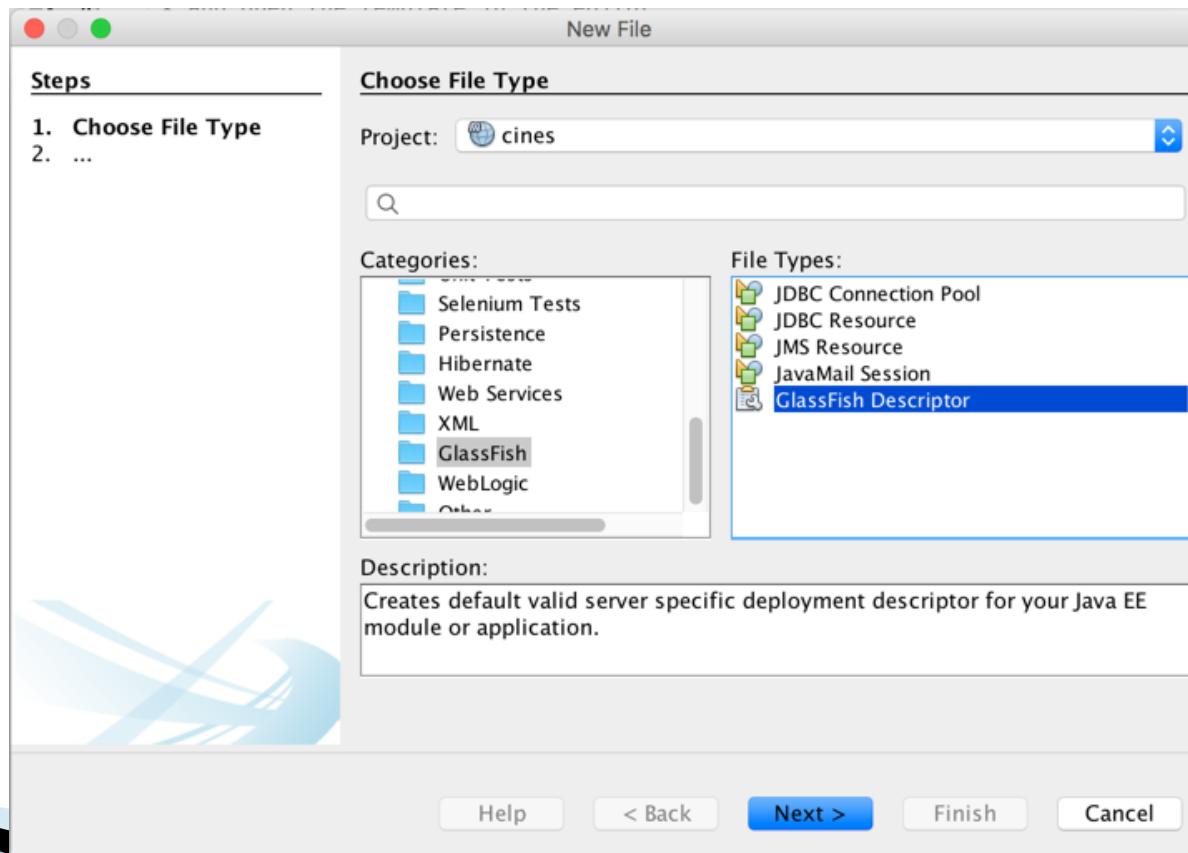
- ▶ El archivo glassfish-web.xml (antes llamado sun-web.xml) y el fichero web.xml son necesario para configurar el mapeo de los roles de usuarios así como las distintas opciones de seguridad deseadas (como el nombre del realm a utilizar, navegación, restricciones de acceso para los distintos roles, etc...)

glassfish-web.xml

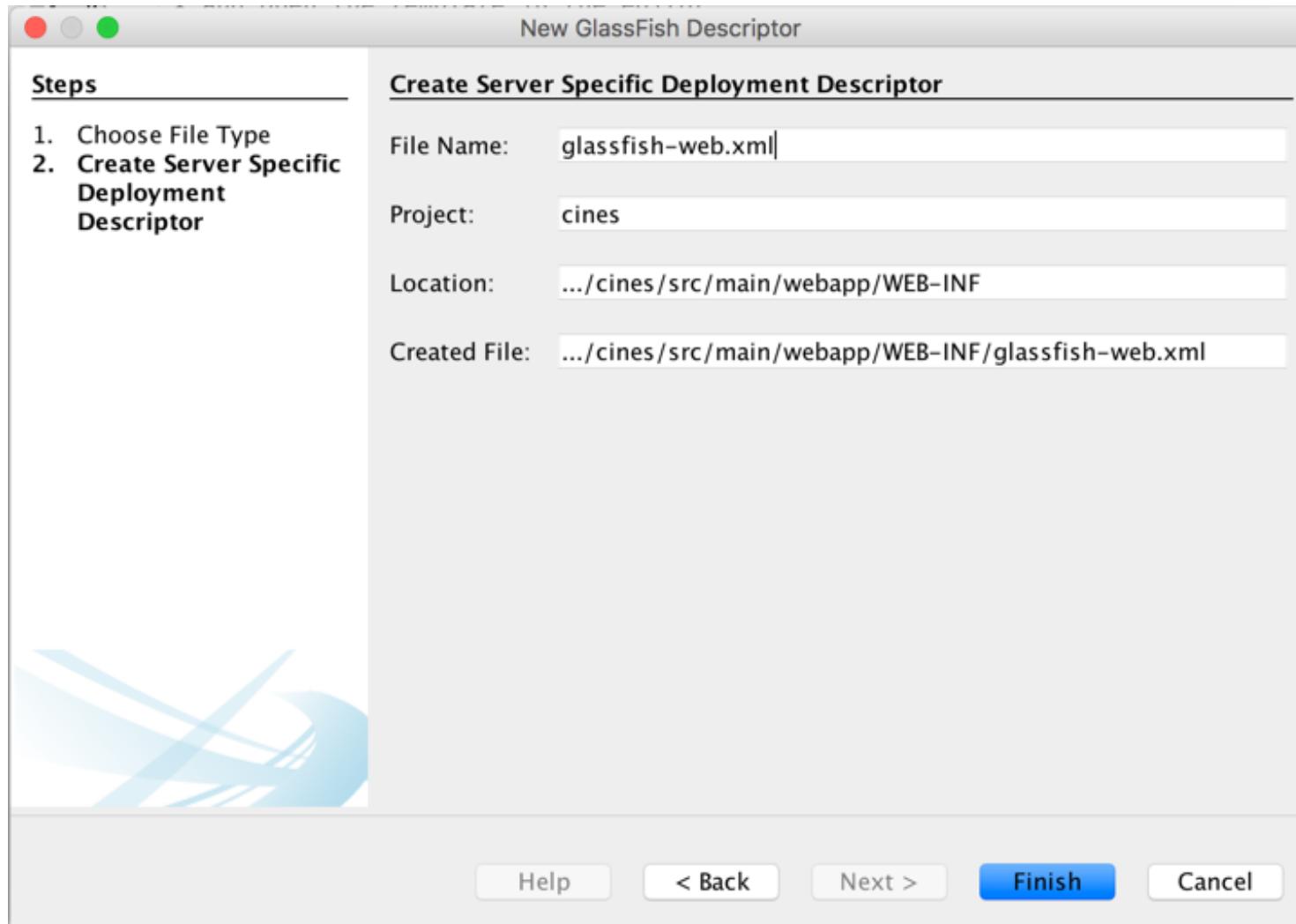
- ▶ Este fichero se utiliza para configurar aplicaciones de web en Java EE (war)
 - Ej: seguridad, cachés, servlets, mensajes, etc... se configuran en este fichero
- ▶ Hasta ahora no lo hemos necesitado, pero para JAAS necesitamos definir en este fichero el mapeo de los distintos roles en la aplicación empresarial (es la manera que tiene JAAS de controlar la autorización)
- ▶ Primero tendremos que crear el fichero glassfish-web.xml
- ▶ Después configuramos los roles de seguridad

glassfish-web.xml

- Para generar el fichero glassfish-web.xml, pulsamos el botón derecho sobre WEB-INF, después New/Other... y ahí buscamos GlassFish/GlassFish Descriptor



glassfish-web.xml



Aunque en este punto os de un error de Deployment Descriptor configuration not found, no pasa nada, podéis continuar sin problema y os creará el fichero.

glassfish-web.xml

- ▶ Añadimos los roles de seguridad:

```
<security-role-mapping>
    <role-name>users</role-name>
    <group-name>users</group-name>
    <group-name>admin</group-name>
</security-role-mapping>
<security-role-mapping>
    <role-name>admin</role-name>
    <group-name>admin</group-name>
</security-role-mapping>
<parameter-encoding default-charset="UTF-8"/>
```

- ▶ Estamos estableciendo dos roles “users” y “admin” y decimos que a los componentes de users se puede acceder desde los usuarios con rol users y con rol admin y a los componentes de admin solamente los usuarios con rol admin
- ▶ Lo de parameter-encoding es para evitar un warning continua que aparece por la distinta codificación entre Glassfish y JSF

glassfish-web.xml

The screenshot shows a Java IDE interface with the tab bar at the top containing General, Servlets, Security, Web Services, Messaging, Environment, XML, and History. The XML tab is currently selected. Below the tabs is a code editor window displaying the `glassfish-web.xml` file. The code is color-coded for syntax highlighting, with tags in blue and attributes in green. The XML structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Servlet 3.0//E
<glassfish-web-app error-url=">
  <security-role-mapping>
    <role-name>users</role-name>
    <group-name>users</group-name>
    <group-name>admin</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>admin</role-name>
    <group-name>admin</group-name>
  </security-role-mapping>
  <class-loader delegate="true"/>
  <jsp-config>
    <property name="keepgenerated" value="true">
      <description>Keep a copy of the generated servlet class' java code.</description>
    </property>
  </jsp-config>
</glassfish-web-app>
```

web.xml

- ▶ En el fichero web.xml se especifican las reglas de seguridad
 - Timeout: tiempo que se mantiene abierta la sesión del usuario. Una vez pasado este tiempo, el usuario tiene que hacer login de nuevo. El valor se da en minutos. Por defecto es igual a 30.
 - Fichero de bienvenida
 - login-config:
 - auth-method: método de autenticación. El nuestro será FORM ya que utilizamos un formulario web para introducir el nombre de usuario y su password (hay otros, ej: BASIC,...)
 - realm-name: nombre del realm a utilizar configurado en GlassFish
 - form-login-page/form-error-page: ficheros xhtml con el formulario de login y con el xhtml mostrado en caso de error (usaremos el mismo y mostraremos los usuarios con los elementos de Primefaces).
 - security-role: tenemos dos roles, así que tenemos que definir dos componentes de este tipo, uno para cada rol

web.xml

- ▶ En el fichero web.xml se especifican las reglas de seguridad
 - security-constraint: restricciones de seguridad para cada rol.
Definimos que las páginas para los usuarios con rol user (user y admin) estén en la carpeta user, y las páginas para los usuarios con rol admin (admin) están en la carpeta admin.
 - En resumen, tenemos que añadir lo siguiente:

web.xml

```
...
</welcome-file-list>

<!-- SECURITY-->
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>jdbc-realm</realm-name>
    <form-login-config>
        <form-login-page>/faces/login.xhtml</form-login-page>
        <form-error-page>/faces/login.xhtml</form-error-page>
    </form-login-config>
</login-config>

<security-role>
    <role-name>users</role-name>
</security-role>

<security-role>
    <role-name>admin</role-name>
</security-role>
```

web.xml

```
<security-constraint>
    <display-name>Restricted to users</display-name>
    <web-resource-collection>
        <web-resource-name>Restricted Access</web-resource-name>
        <url-pattern>/faces/users/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>users</role-name>
    </auth-constraint>
</security-constraint>

<security-constraint>
    <display-name>Restricted to admin</display-name>
    <web-resource-collection>
        <web-resource-name>Restricted Access</web-resource-name>
        <url-pattern>/faces/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>
</web-app>
```

Construir la interfaz de usuario

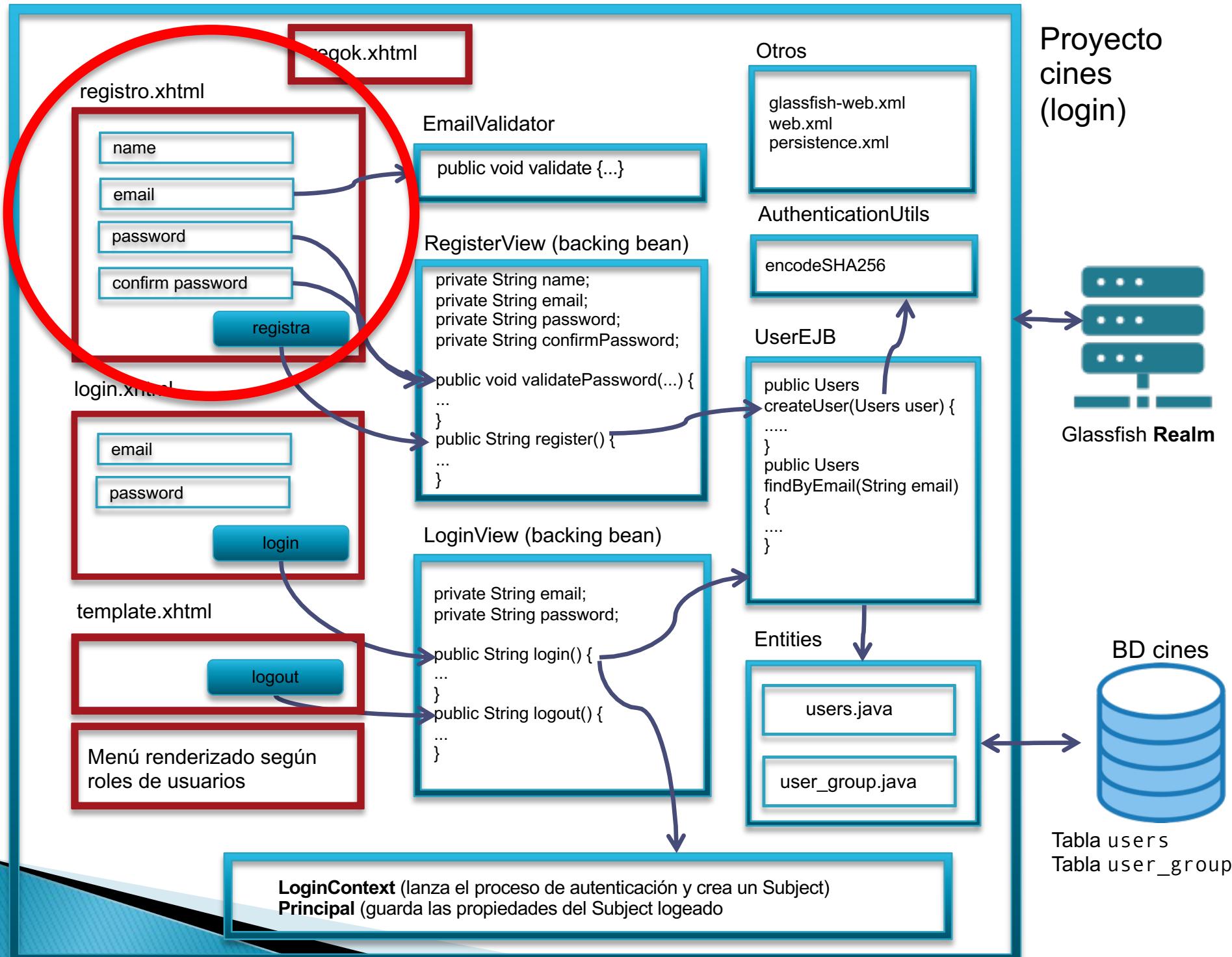
► Construir la interfaz de usuario

- Crear página de registro (interfaz y beans correspondientes)
- Crear página de login (interfaz y beans correspondientes)
- Crear función de logout
- Crear páginas privadas para cada rol
- Actualizar template

Crear página de registro

- ▶ Para crear la página que gestione el registro de usuarios necesitaremos crear varios elementos:
 - Una página (facelets template client) con el formulario de registro de usuario (pediremos el nombre del usuario, email, password y confirmación de password)
 - Un backing bean que guarde la información del usuario y que realice el registro
 - Validadores: que la dirección de email tenga el formato adecuado, que los passwords son iguales, que el usuario no existe (email único), que el password tenga un determinado número de caracteres en mayúsculas, dígitos, etc...
 - Una página (facelets template client) para mostrar en caso de que el registro se realice correctamente

Proyecto cines (login)



Crear página de registro

Formulario de registro

Crear una nueva cuenta

| | |
|-----------------------|---------------------|
| Nombre: | Anibal Bregon |
| E-Mail: | anibal@infor.uva.es |
| Contraseña: | |
| Confirmar contraseña: | |

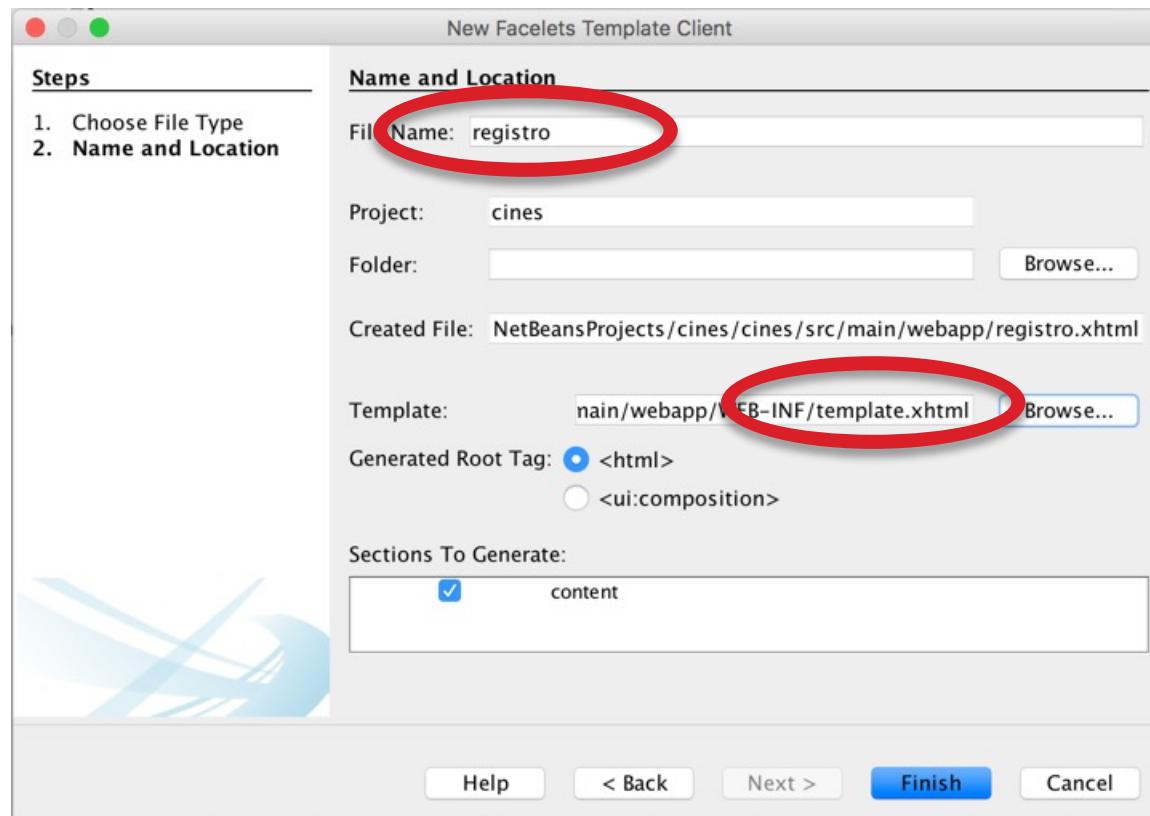
 **Regístrate!**

 Good

Crear página de registro

Formulario de registro

- ▶ Creamos una nueva página (Facelets Template Client) que cuelgue de “Web Pages” llamada “registro” y añadimos lo siguiente...



Crear página de registro

Formulario de registro

```
<h:form>
    <f:event listener="#{registerView.validatePassword}" type="postValidate" />

    <p:panel header="Crear una nueva cuenta">
        <h:panelGrid id="grid" columns="3">

            <h:outputLabel for="name" value="Nombre:" style="font-weight:bold"/>
            <p:inputText id="name" value="#{registerView.name}" required="true"
requiredMessage="Por favor, introduce tu nombre." maxlength="30"/>
            <p:message for="name"/>

            <h:outputLabel for="email" value="E-Mail:" style="font-weight:bold"/>
            <p:inputText id="email" value="#{registerView.email}" required="true"
requiredMessage="Por favor, introduce tu email.">
                <f:validator validatorId="emailValidator" />
            </p:inputText>
            <p:message for="email"/>
        </h:panelGrid>
    </p:panel>
</h:form>
```

Crear página de registro

Formulario de registro

```
<h:outputLabel for="password" value="Contraseña:" style="font-weight:bold"/>
<p:password id="password" value="#{registerView.password}" feedback="true"
    required="true" requiredMessage="Por favor, introduce tu contraseña."
    validatorMessage="La contraseña debe contener al menos un carácter en minúsculas,
    un carácter en mayúscula, un número y su longitud debe estar entre 6 y 20
caracteres.">
    <f:validateRegex pattern="((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,20})" />
</p:password>
<p:message for="password"/>

<h:outputLabel for="confirmpassword" value="Confirmar contraseña:" style="font-weight:bold"/>
<p:password id="confirmpassword" feedback="true"
    value="#{registerView.confirmPassword}" required="true"
    requiredMessage="Por favor, confirma tu contraseña."/>
<p:message for="confirmpassword"/>

</h:panelGrid>
<p:commandButton action="#{registerView.register}" update="grid" value="Regístrate!" icon="ui-
icon-pencil"/>
</p:panel>
</h:form>
```

Crear página de registro

Formulario de registro

- ▶ <f:event listener="#{registerView.validatePassword}" type="postValidate" />
 - Registrar un PostValidateEvent – se llama justo antes de hacer el post
 - **Lo vamos a utilizar para validar los passwords introducidos por el usuario antes de enviarlo al servidor**
- ▶ <p:inputText id="name" value="#{registerView.name}" required="true" requiredMessage="Por favor, introduce tu nombre." maxlength="30"/>
 - En registerView, además de tener el método de validación de passwords, vamos a guardar toda la información introducida por el usuario.

Crear página de registro

Formulario de registro

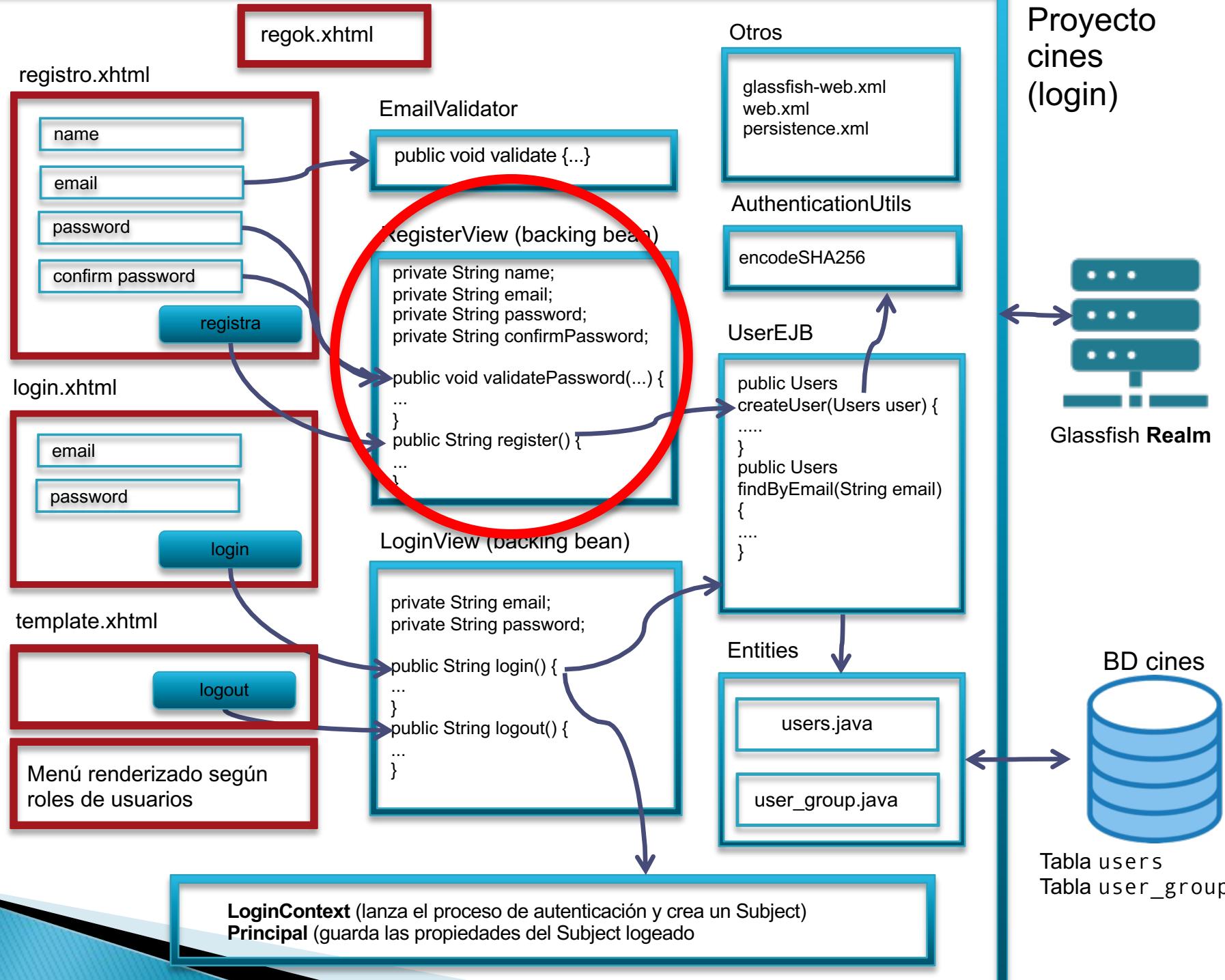
- ▶ <f:validator validatorId="emailValidator" />
 - Llamamos al validador emailValidator para comprobar, antes de enviarlo al servidor, que el email introducido por el usuario cumple un formato determinado
 - Más adelante veremos como crear este validador personalizado
- ▶ <p:password ...>
 - Utilizamos p:password en lugar de p:inputText para las contraseñas, así nos ocultará el texto que vayamos introduciendo
 - Con la opción feedback="true" hacemos que primefaces muestre una indicación sobre la fortaleza de la contraseña introducida

Crear página de registro

Formulario de registro

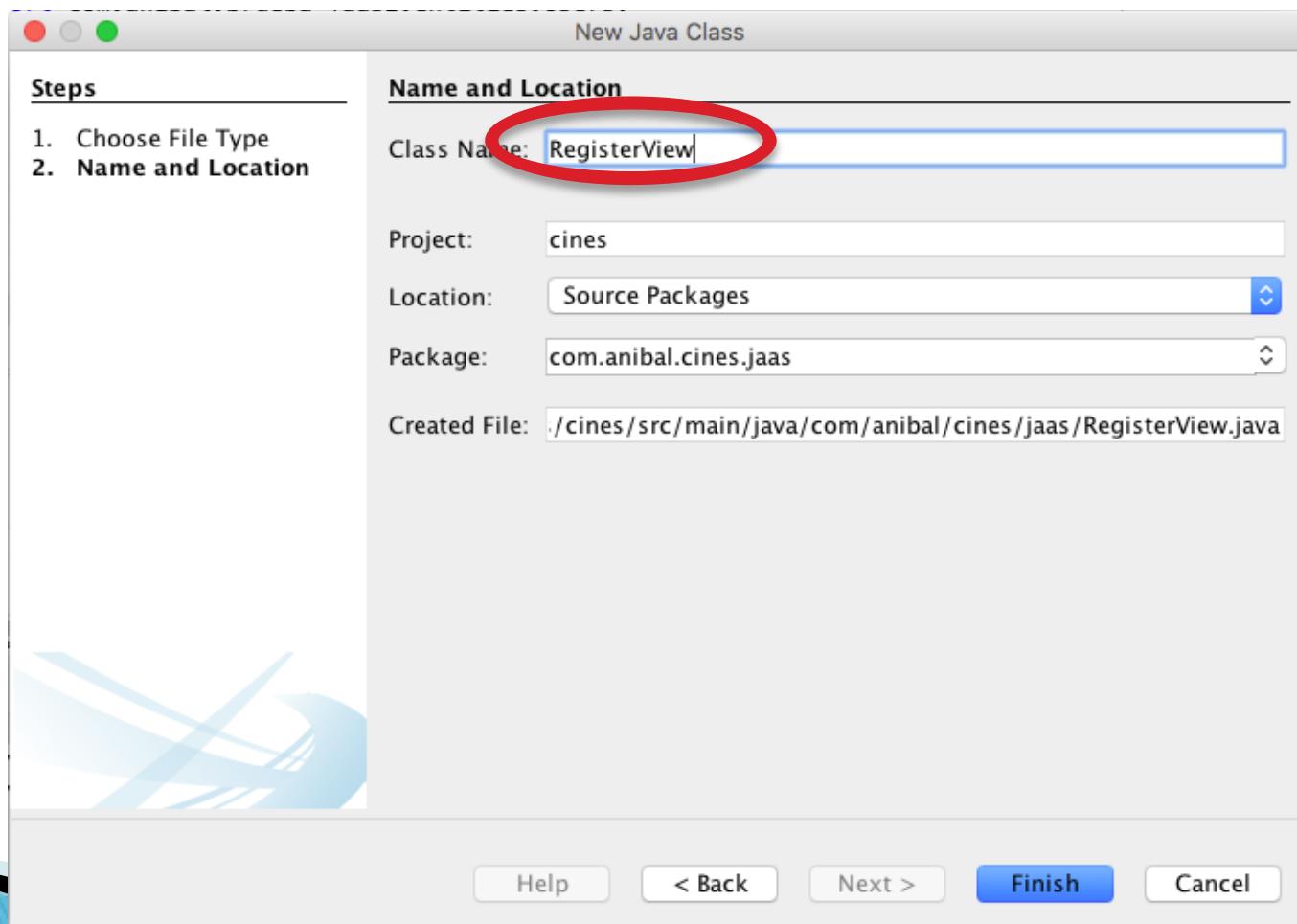
- ▶ <f:validateRegex pattern="((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,20})" />
 - Validador para el formato de la contraseña... al menos un número (\d), una minúscula, una mayúscula y un número de caracteres entre 6 y 20
- ▶ <p:commandButton action="#{registerView.register}" update="grid" value="Regístrate!" icon="ui-icon-pencil"/>
 - Cuando pulsamos el botón de registrar se ejecuta la función register de registerView
 - Esta acción devolverá un string al action que se corresponde con la página web a la que navegar a continuación

Proyecto cines (login)



Crear página de registro BackingBean RegisterView

- ▶ Crear una nueva clase java llamada “RegisterView” dentro del paquete jaas



Crear página de registro

BackingBean RegisterView

- ▶ Añadimos las variables que van a guardar la información de registro introducida por el usuario:
 - name, email, password, confirmPassword
- ▶ Generamos getter y setter
- ▶ Añadimos las anotaciones @Named y @SessionScoped
 - **(cuidado con los import que usamos aquí!!! – de javax.enterprise)**
- ▶ Implementamos la interfaz Serializable
- ▶ Inyectamos UserEJB ya que es la capa que contiene la lógica de negocio para consultar el email y crear usuarios en la base de datos:

```
@Inject  
private UserEJB userEJB;
```

```
private String name;  
private String email;  
private String password;  
private String confirmPassword;
```

Crear página de registro BackingBean RegisterView

- ▶ Por último creamos los métodos para validar el password introducido por el usuario y para registrar el usuario

Crear página de registro

BackingBean RegisterView

```
public void validatePassword(ComponentSystemEvent event) {
    FacesContext facesContext = FacesContext.getCurrentInstance();
    UIComponent components = event.getComponent();

    UIInput uiInputPassword = (UIInput) components.findComponent("password");
    String password = uiInputPassword.getLocalValue() == null ? "" : uiInputPassword.getLocalValue().toString();

    UIInput uiInputConfirmPassword = (UIInput) components.findComponent("confirmpassword");
    String confirmPassword = uiInputConfirmPassword.getLocalValue() == null ? "" :
    uiInputConfirmPassword.getLocalValue().toString();

    if (password.isEmpty() || confirmPassword.isEmpty()) {
        return;
    }
    if (!password.equals(confirmPassword)) {
        FacesMessage msg = new FacesMessage("Las contraseñas no coinciden");
        msg.setSeverity(FacesMessage.SEVERITY_ERROR);
        facesContext.addMessage(uiInputPassword.getClientId(), msg);
        facesContext.renderResponse();
    }

    UIInput uiInputEmail = (UIInput) components.findComponent("email");
    String email = uiInputEmail.getLocalValue() == null ? "" : uiInputEmail.getLocalValue().toString();

    if (userEJB.findByEmail(email) != null) {
        FacesMessage msg = new FacesMessage("Ya existe un usuario con ese email");
        msg.setSeverity(FacesMessage.SEVERITY_ERROR);
        facesContext.addMessage(uiInputPassword.getClientId(), msg);
        facesContext.renderResponse();
    }
}
```

Crear página de registro

BackingBean RegisterView

- ▶ `FacesContext facesContext = FacesContext.getCurrentInstance();`
 - Guardamos la instancia actual del contexto de JSF desde el que se llama a esta función
 - Lo necesitaremos para poder escribir mensajes de error en la interfaz de usuario en el caso de que fallen las validaciones
- ▶ `UIComponent components = event.getComponent();`
 - En components guardamos la referencia a los componentes de la interfaz de usuario
 - Lo necesitaremos para ser capaz de coger los valores que ha introducido el usuario en los inputText
- ▶ `UIInput uiInputPassword = (UIInput) components.findComponent("password");`
 - Buscamos el componente de la interfaz de usuario que tenga el id password
- ▶ `String password = uiInputPassword.getLocalValue() == null ? "" : uiInputPassword.getLocalValue().toString();`
 - Y guardamos el valor introducido por el usuario (si no está vacío) como un String

Crear página de registro

BackingBean RegisterView

- ▶ `if (!password.equals(confirmPassword)) {`
 - Si los passwords introducidos por el usuario son distintos, tendremos que mostrar un mensaje de error
- ▶ `FacesMessage msg = new FacesMessage("Las contraseñas no coinciden");`
 - Creamos un nuevo mensaje de JSF
- ▶ `msg.setSeverity(FacesMessage.SEVERITY_ERROR);`
 - Hacemos que el mensaje tenga formato y color como un mensaje de tipo error
- ▶ `facesContext.addMessage(uiInputPassword.getClientId(), msg);`
`facesContext.renderResponse();`
 - Mostramos el mensaje al usuario

Crear página de registro

BackingBean RegisterView

- ▶ if (userEJB.findByEmail(email) != null) {
 - Si las contraseñas que ha introducido el usuario no son nulas, son iguales, y cumplen la validación de formato establecida, entonces tenemos que mirar que el usuario no esté ya registrado en la aplicación
 - Para ello llamamos a findByEmail de la lógica de negocio (userEJB) y si el usuario ya existe mostraremos un mensaje de error indicándolo como para el caso en el que los passwords eran distintos

Crear página de registro

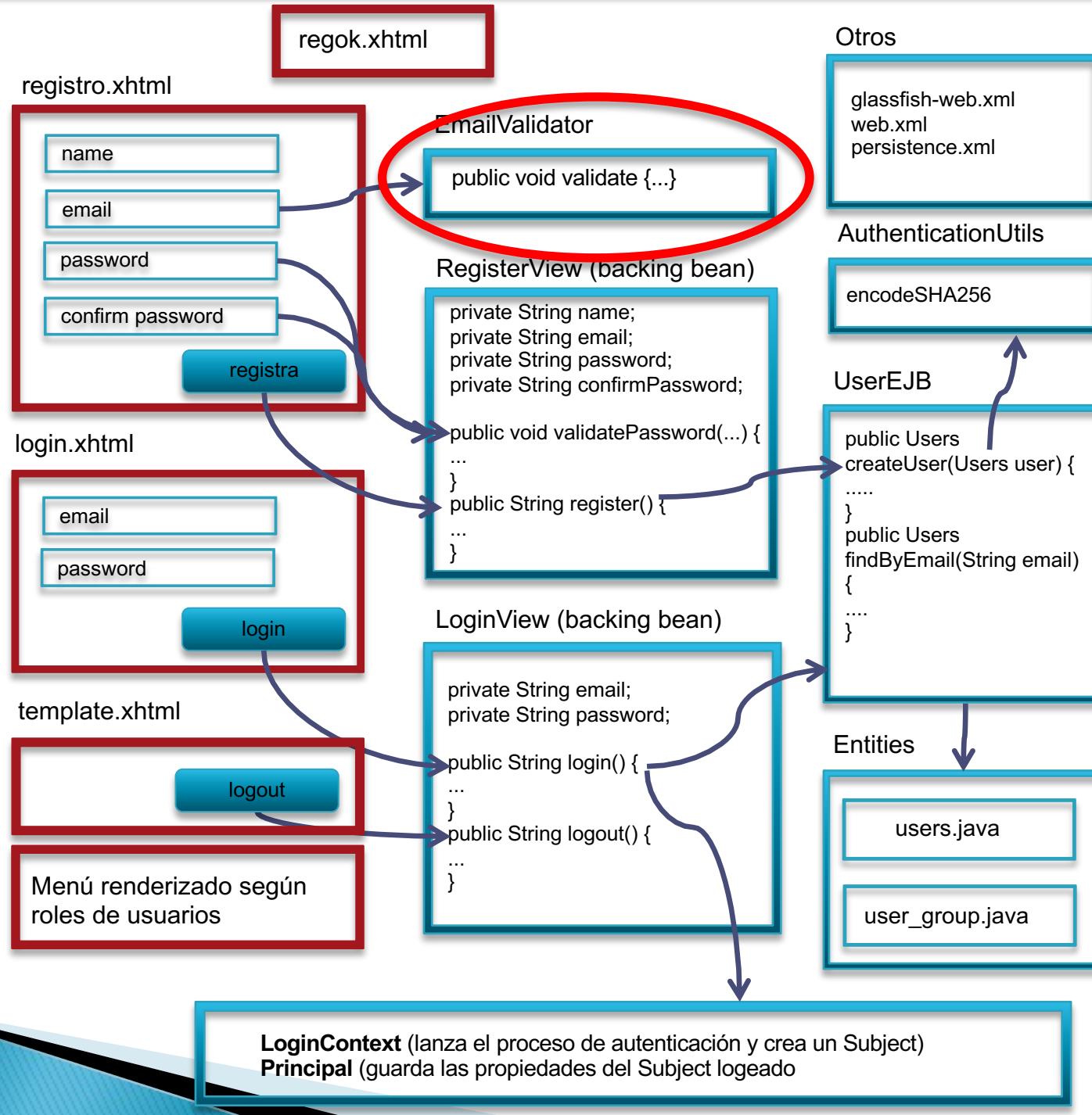
BackingBean RegisterView

```
public String register() {  
    Users user = new Users(email, password, name);  
    userEJB.createUser(user);  
    System.out.println("Nuevo usuario creado con e-mail: " + email + " y nombre:  
" + name);  
    return "regok";  
}
```

Crear página de registro BackingBean RegisterView

- ▶ `Users user = new Users(email, password, name);`
 - Creamos un objeto de la clase (entidad) usuario con los datos de registro
 - Lo hacemos así como ya hemos visto en ejemplos anteriores: creamos un objeto de la clase entidad y luego llamamos a persist (en este caso lo haremos persistente en la clase userEJB que es la que guarda toda la lógica de negocio)
- ▶ `userEJB.createUser(user);`
 - Llamamos a la clase createUser del EJB para crear el usuario (también tendrá que crear una entrada en la tabla grupos y codificar la contraseña... como estas son opciones de la lógica de negocio, la creación del usuario la hacemos en el EJB y no en el BackingBean)
- ▶ `return "regok";`
 - Devolvemos un string que es el nombre de la página a la que redirigirse cuando se realiza el registro
 - Este string es el que interpreta el action en registro.xhtml
 - Aquí podríamos haberlo hecho más complejo, por ejemplo gestionando si el registro se ha hecho correctamente o no, y que nos mande a distintas páginas dependiendo del resultado

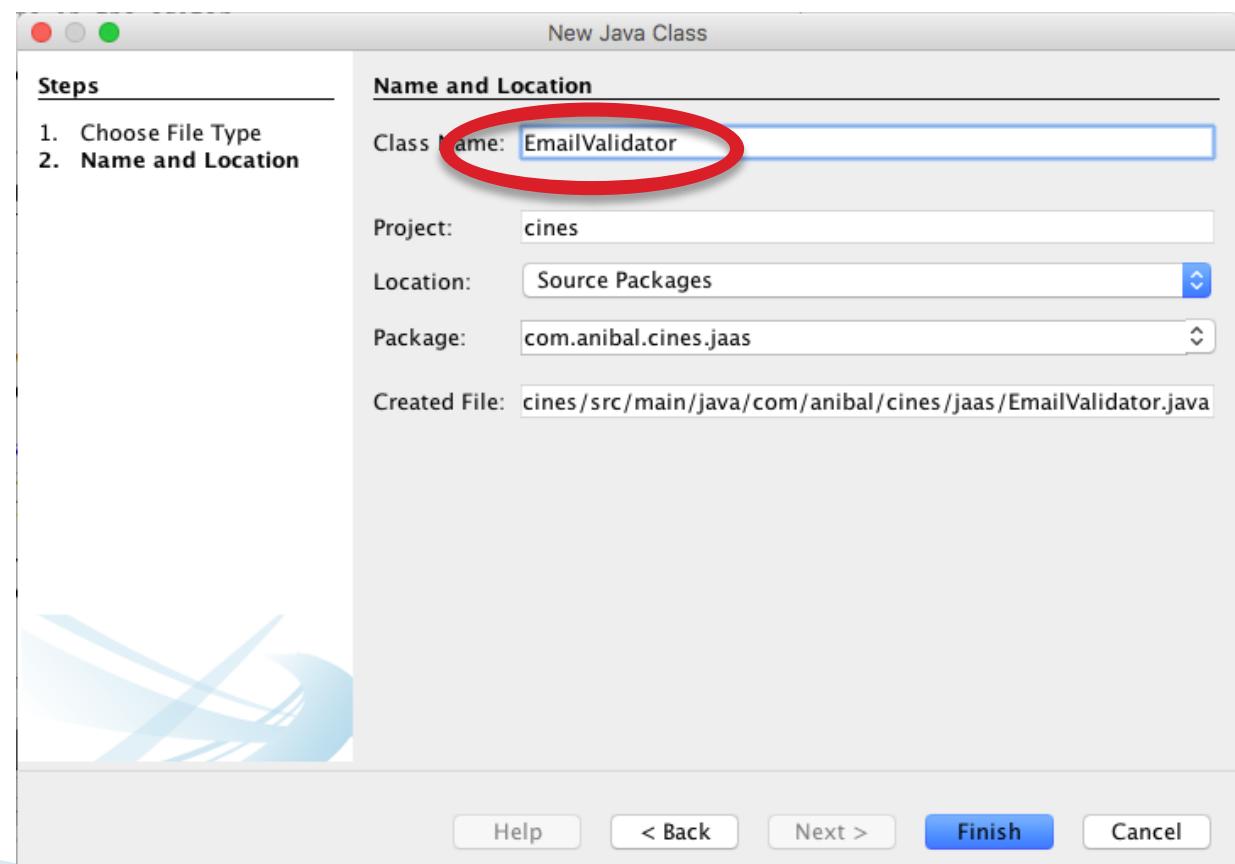
Proyecto cines (login)



Crear página de registro

Validador del formato del email

- ▶ Crear una nueva clase java llamada “EmailValidator” dentro del paquete jaas
- ▶ Con esto también veremos como se hacen validadores distintos de los que vienen por defecto en JSFs



Crear página de registro

Validador del formato del email

```
@FacesValidator("emailValidator")
public class EmailValidator implements Validator {

    private static final String EMAIL_PATTERN = "^[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";
    private Pattern pattern;

    public EmailValidator() {
        pattern = Pattern.compile(EMAIL_PATTERN);
    }

    @Override
    public void validate(FacesContext context, UIComponent component, Object value) throws ValidatorException {
        if (value == null) {
            return;
        }

        if (!pattern.matcher(value.toString()).matches()) {
            throw new ValidatorException(new FacesMessage(FacesMessage.SEVERITY_ERROR, "Error de
validación", value + " no es un email válido."));
        }
    }
}
```

(cuidado con los import que usamos
aquí!!! – de javax.faces (excepto el
de Pattern))

Crear página de registro

Validador del formato del email

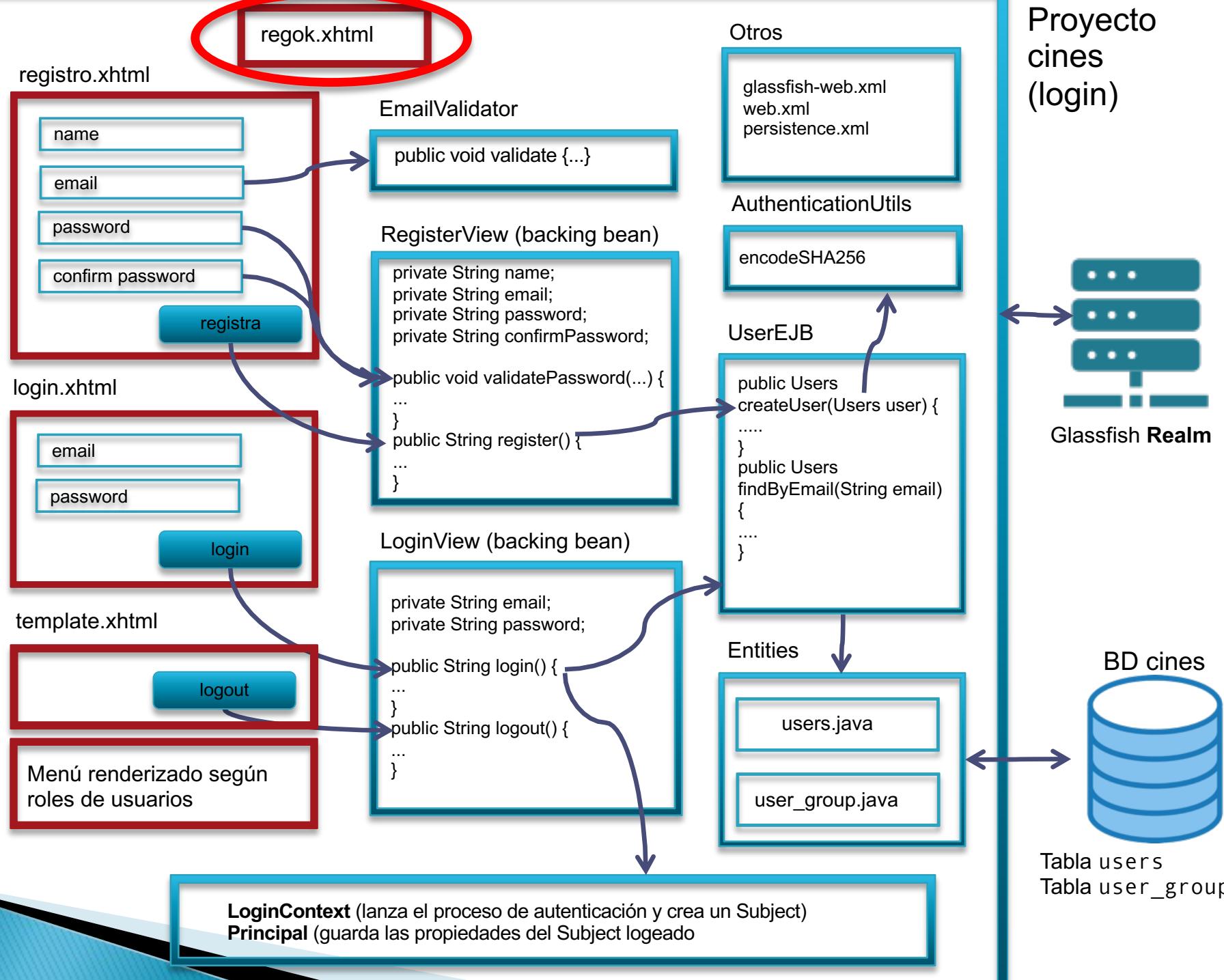
- ▶ `@FacesValidator("emailValidator")`
 - Anotación para validadores de JSF
- ▶ `public class EmailValidator implements Validator {`
 - Implementamos la interfaz Validator
- ▶ `private static final String EMAIL_PATTERN = "^[A-Za-z0-9-]+(\\.[A-Za-z0-9-]+)*@[A-Za-z0-9-]+(\\.[A-Za-z0-9-]+)*(\\.[A-Za-z]{2,})$";`
 - Patrón de expresión regular para los emails:
 - ^ – que haga match al principio de la línea
 - \$ – que haga match al final de la línea
 - + – 1 o más de lo anterior
 - * – 0 o más de lo anterior
 - \\. – punto (.)
 - {2,} – ocurre 2 veces o más

Crear página de registro

Validador del formato del email

- ▶ if (!pattern.matcher(value.toString()).matches()) {
 - Comprueba si el patrón introducido por el usuario se ajusta al patrón definido en pattern (que se construye utilizando la constante EMAIL_PATTERN)
 - Si no se ajusta, muestra un mensaje de error

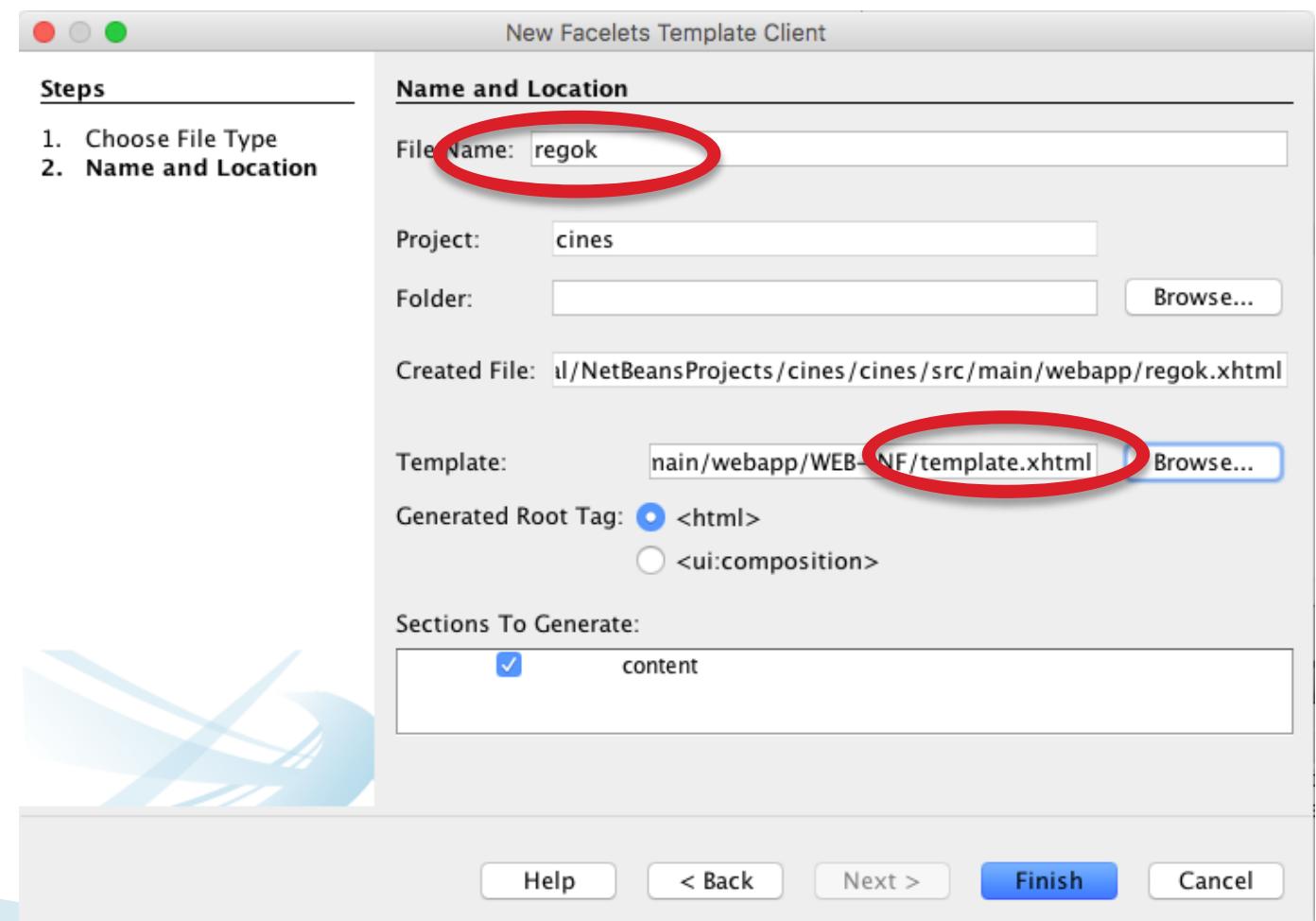
Proyecto cines (login)



Crear página de registro

Página de registro correcto

- ▶ Creamos una nueva página (Facelets Template Client) que cuelgue de “Web Pages” llamada “regok” y añadimos lo siguiente...



Crear página de registro

Página de registro correcto

```
<h3>Tu cuenta de usuario ha sido creada satisfactoriamente!</h3>
<br/><br/>
<p:link value="Accede con tu nueva cuenta" outcome="login" />
```

Construir la interfaz de usuario

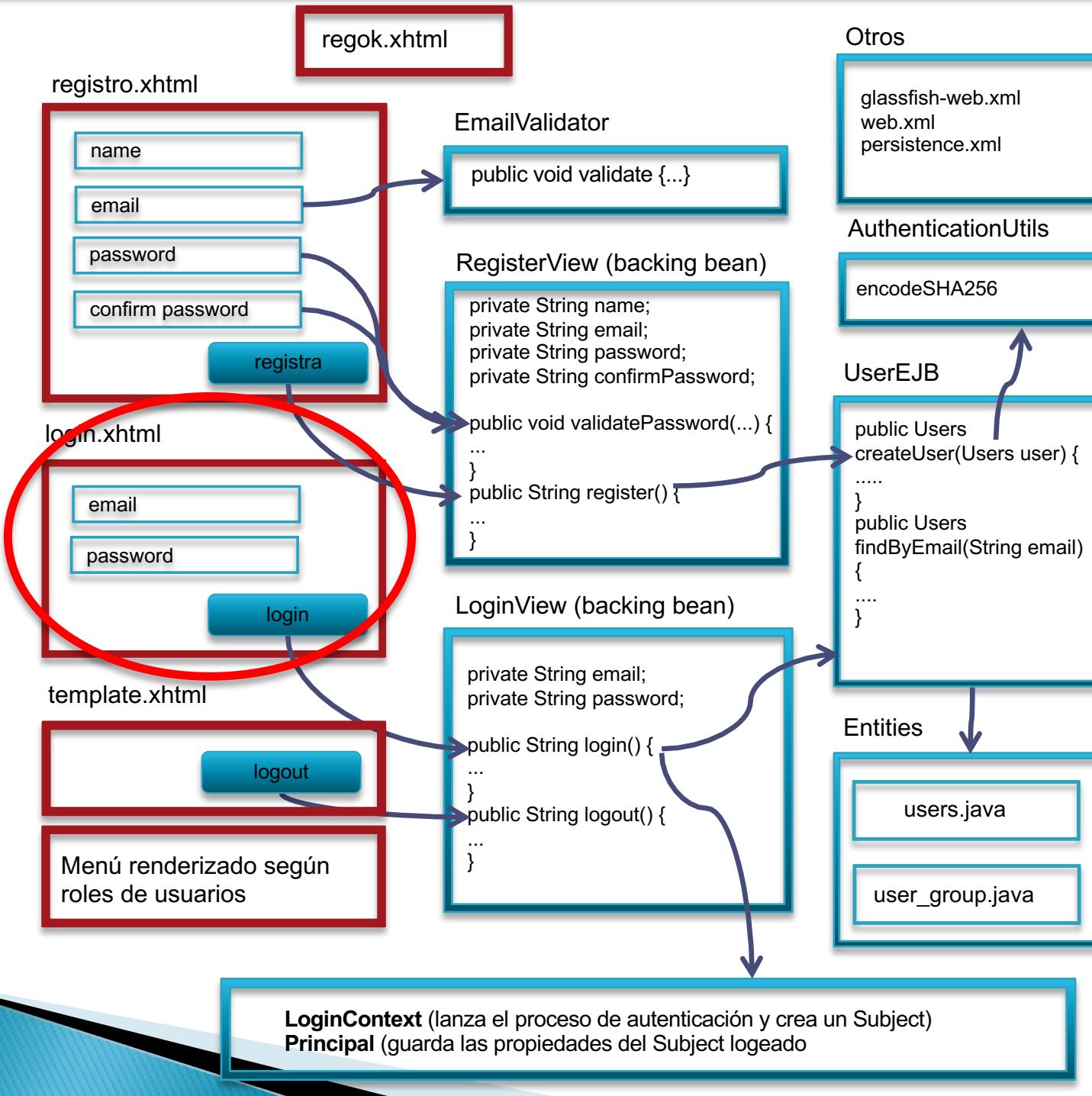
► Construir la interfaz de usuario

- Crear página de registro (interfaz y beans correspondientes)
- Crear página de login (interfaz y beans correspondientes)
- Crear función de logout
- Crear páginas privadas para cada rol
- Actualizar template

Crear página de login

- ▶ Para crear la página que gestione el login de usuarios necesitaremos crear varios elementos:
 - Una página (facelets template client) con el formulario de login de usuario (pediremos el email y el password)
 - Un backing bean que guarde la información del usuario logeado y llame al método que hace el login

Proyecto cines (login)



Crear página de login

Página de login

Acceso de usuario

E-Mail:

Contraseña:

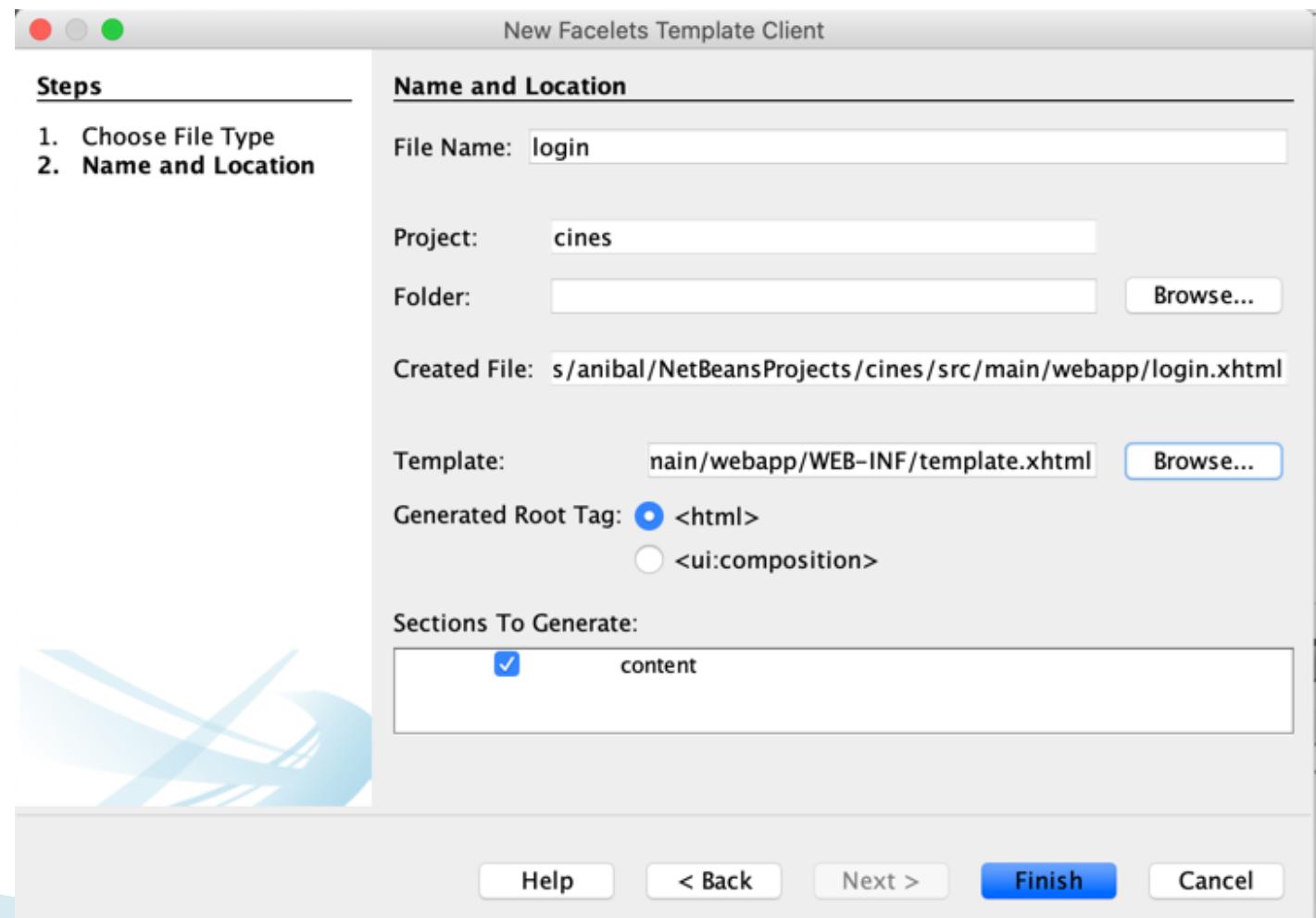
 **Login**



Crear página de login

Página de login

- ▶ Creamos una nueva página (Facelets Template Client) que cuelgue de “Web Pages” llamada “login” y añadimos lo siguiente...



Crear página de login

Página de login

```
<h:form>
    <p:messages id="messages" closable="true"/>

    <p:panel header="Acceso de usuario">
        <h:panelGrid id="grid" columns="2">
            <h:outputLabel for="email" value="E-Mail:" style="font-weight:bold"/>
            <p:inputText id="email" value="#{loginView.email}" required="true"
requiredMessage="Por favor, introduce tu email."/>

            <h:outputLabel for="password" value="Contraseña:" style="font-weight:bold"/>
            <p:password id="password" value="#{loginView.password}" required="true"
requiredMessage="Por favor, introduce tu contraseña."/>
        </h:panelGrid>

        <p:commandButton action="#{loginView.login}" value="Login" icon="ui-icon-check"
update="messages"/>

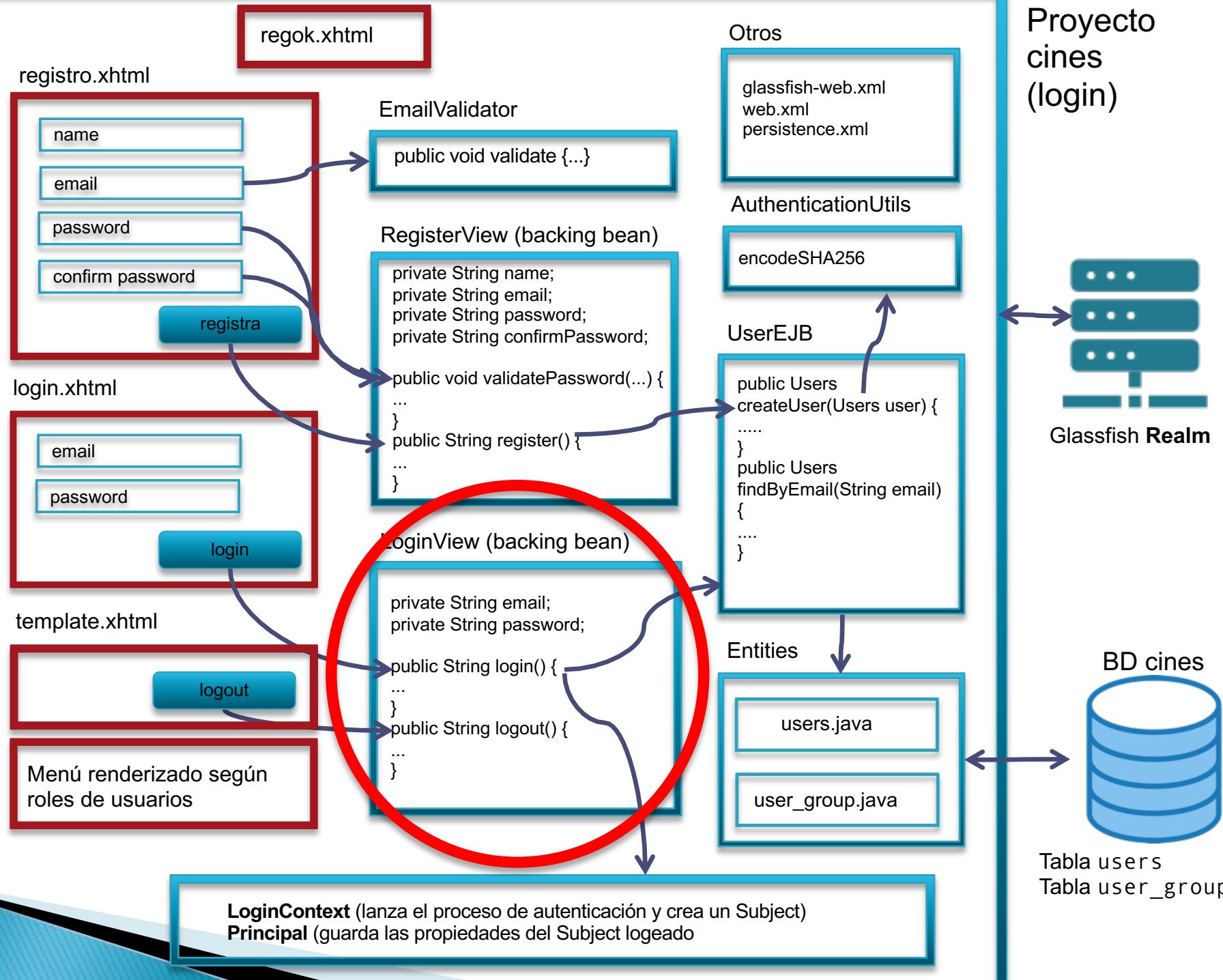
    </p:panel>
</h:form>
```

Crear página de login

Página de login

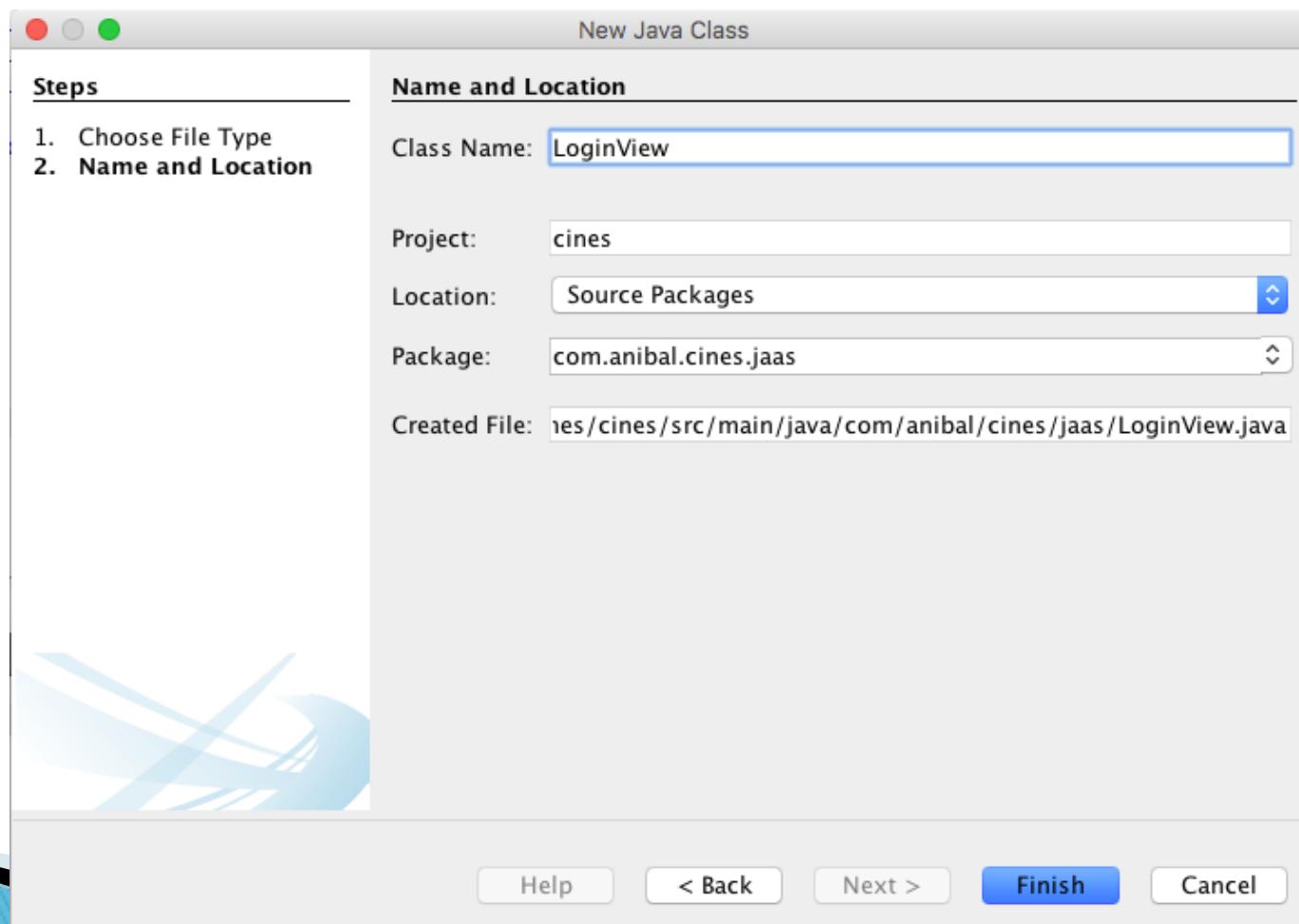
- ▶ <p:messages id="messages" closable="true" ...
 - Con p:messages conseguimos que los mensajes aparezcan de distinta manera a como hasta ahora
 - p:messages muestra todos los mensajes de error juntos y arriba de la página
- ▶ <p:commandButton action="#{loginView.login}" value="Login" icon="ui-icon-check"/>
 - Llamamos al método login de LoginView para realizar la comprobación de login del usuario

Proyecto cines (login)



Crear página de login BackingBean LoginView

- ▶ Crear una nueva clase java llamada “LoginView” dentro del paquete jaas



Crear página de login

BackingBean LoginView

- ▶ Añadimos las variables que van a guardar la información de login introducida por el usuario:
 - email, password
- ▶ Generamos getter y setter
- ▶ Añadimos las anotaciones @Named y @SessionScoped
 - **(cuidado con los import que usamos aquí!!! – de javax.enterprise)**
- ▶ Implementamos la interfaz Serializable
- ▶ Inyectamos UserEJB ya que es la capa que contiene la lógica de negocio para consultar el email y crear usuarios en la base de datos:

```
@Inject  
private UserEJB userEJB;
```

Crear página de login

BackingBean LoginView

- ▶ También vamos a necesitar una variable que guarde la información del usuario que ha hecho login
- ▶ Esta variable se utilizará para obtener información del usuario que se ha logeado
- ▶ Para ello declaramos la variable:

```
private Users user;
```

- ▶ Y el método para devolver su valor:

```
public Users getAuthenticatedUser() {  
    return user;  
}
```

- ▶ Por último creamos el método de login...

Crear página de login

BackingBean LoginView

```
public String login() {  
    FacesContext context = FacesContext.getCurrentInstance();  
    HttpServletRequest request = (HttpServletRequest) context.getExternalContext().getRequest();  
    try {  
        request.login(email, password);  
    } catch (ServletException e) {  
        context.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_WARN, "Login  
incorrecto!", null));  
        return "login";  
    }  
  
    this.user = userEJB.findByEmail(request.getUserPrincipal().getName());  
    if (request.isUserInRole("users")) {  
        return "/users/privatepage?faces-redirect=true";  
    } else if (request.isUserInRole("admin")) {  
        return "/admin/privatepage?faces-redirect=true";  
    } else {  
        return "login";  
    }  
}
```

Crear página de login

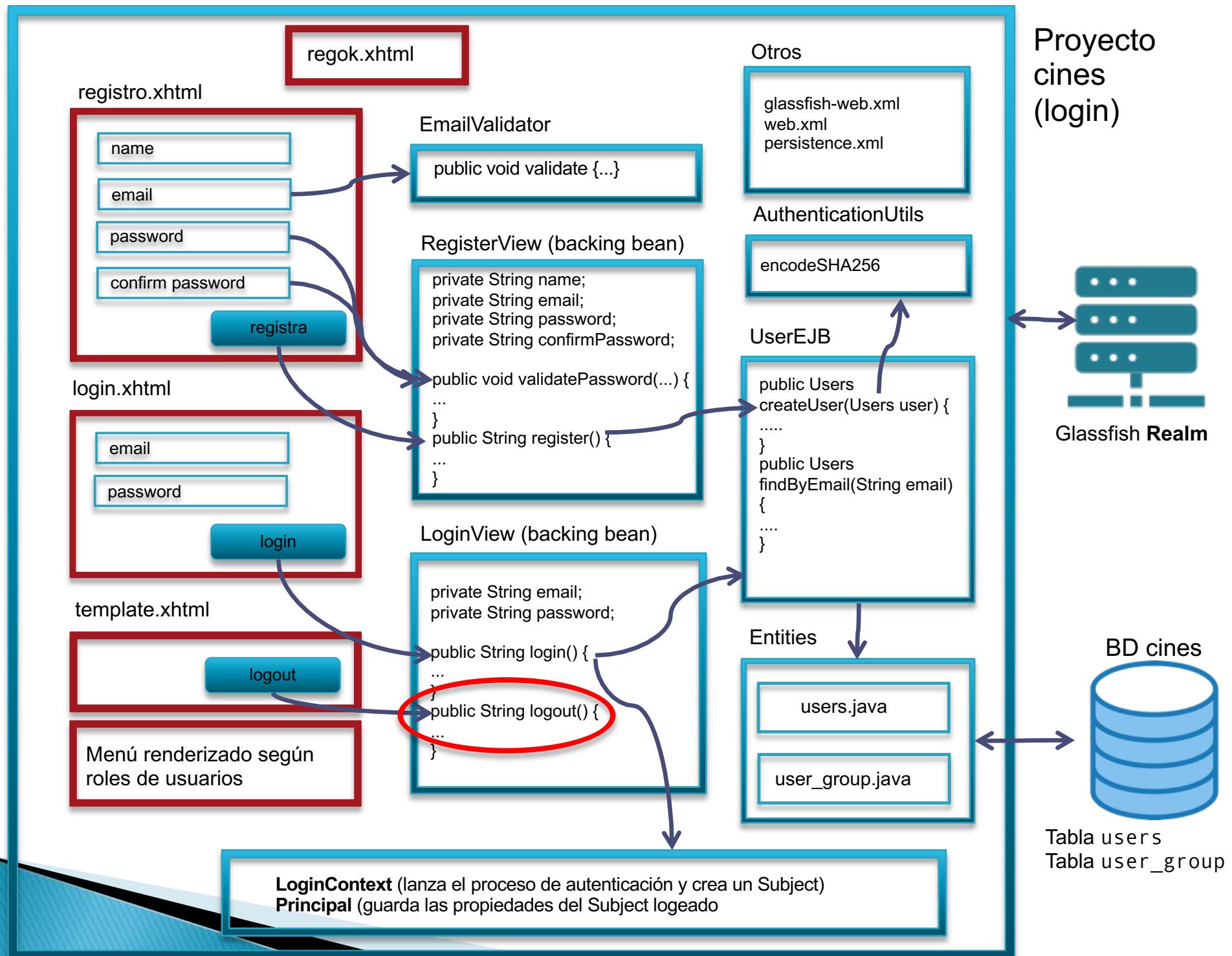
BackingBean LoginView

- ▶ `request.login(email, password);`
 - Hacemos el login llamando al contexto que JAAS nos proporciona para ello
- ▶ `this.user =`
`userEJB.findByEmail(request.getUserPrincipal().getName());`
 - Si la información de login es correcta, automáticamente se crea el objeto Principal
 - El objeto Principal es básico en JAAS y guarda el identificador (o identificadores) únicos del usuario que lo distinguen de otros usuarios
 - En este ejemplo, sólo tenemos el email, por lo que al hacer `getUserPrincipal().getName()` nos devuelve el email del usuario que está logeado
 - Si tuviéramos, por ejemplo, email y NIF, JAAS podría crear dos objetos Principal para cada id
 - Lo utilizamos para buscar el usuario logeado en la base de datos y guardar toda su información en la entidad user
- ▶ `if (request.isUserInRole("users")) {`
 - Dependiendo del rol del usuario, redireccionamos la página principal a una o a otra

Construir la interfaz de usuario

► Construir la interfaz de usuario

- Crear página de registro (interfaz y beans correspondientes)
- Crear página de login (interfaz y beans correspondientes)
- **Crear función de logout**
- Crear páginas privadas para cada rol
- Actualizar template



Añadir función de logout BackingBean LoginView

- ▶ Añadimos la función de logout (en LoginView)

```
public String logout() {  
    FacesContext context = FacesContext.getCurrentInstance();  
    HttpServletRequest request = (HttpServletRequest) context.getExternalContext().getRequest();  
    try {  
        this.user = null;  
        request.logout();  
        ((HttpSession) context.getExternalContext().getSession(false)).invalidate();  
    } catch (ServletException e) {  
        System.out.println("Fallo durante el proceso de logout!");  
    }  
    return "/index?faces-redirect=true";  
}
```

Añadir función de logout BackingBean LoginView

- ▶ `request.logout();`
 - Hacemos el logout llamando al contexto que JAAS nos proporciona para ello
- ▶ `((HttpSession) context.getExternalContext().getSession(false)).invalidate();`
 - Limpiamos la sesión
- ▶ `?faces-redirect=true`
 - Esto hace que se cambie la página en el navegador cuando nos vamos a ella (es la diferencia entre el redirect y el forward)

Construir la interfaz de usuario

► Construir la interfaz de usuario

- Crear página de registro (interfaz y beans correspondientes)
- Crear página de login (interfaz y beans correspondientes)
- Crear función de logout
- **Crear páginas privadas para cada rol**
- Actualizar template

Crear páginas privadas para cada rol

- ▶ Crear dos carpetas en “Web Pages” una llamada “admin” y la otra llamada “users”
- ▶ Al contenido de la carpeta “admin” sólo podrán entrar los usuarios que tengan ese rol (es decir, sólo admin)
- ▶ Al contenido de la carpeta “users” sólo podrán entrar los usuarios que tengan ese rol (es decir, admin y users)
- ▶ Creamos dos páginas sencillas, para cada una de las carpetas (ambas llamadas “privatepage.xhtml”) y ambas con el mismo contenido...

Crear páginas privadas para cada rol

privatepage.xhtml

```
<h3>Hola #{loginView.authenticatedUser.name}</h3>
<p>Tu id de usuario es #{loginView.authenticatedUser.email}</p>
<p>Tienes el rol users?: #{request.isUserInRole('users')}</p>
<p>Tienes el rol admin?: #{request.isUserInRole('admin')}</p>
<p>Esta es una página privada, requiere acceso mediante password.</p>
```

Construir la interfaz de usuario

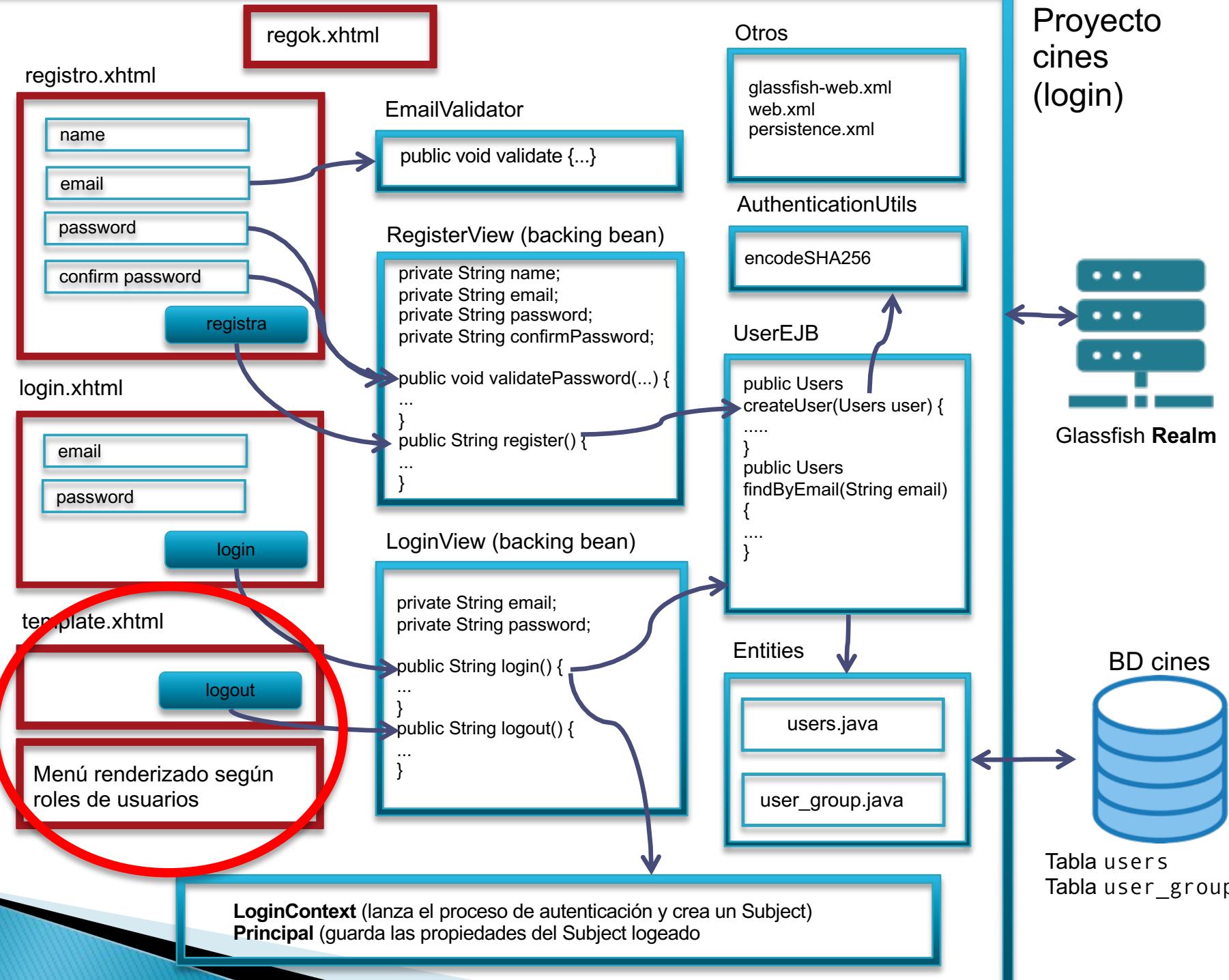
► Construir la interfaz de usuario

- Crear página de registro (interfaz y beans correspondientes)
- Crear página de login (interfaz y beans correspondientes)
- Crear función de logout
- Crear páginas privadas para cada rol
- **Actualizar template**

Actualizar template

- ▶ El último paso será actualizar el template
- ▶ Vamos a añadir dos cambios:
 - Cambiar la cabecera para ponerle una toolbar: esto nos servirá por una parte para ver cómo se pone una barra de herramientas en Primefaces, y por otra parte para poner el botón de logout
 - Renderizar los elementos del menú dependiendo del rol que tengan permitido

Proyecto cines (login)



Actualizar template

Cabecera

```
<p:layoutUnit position="north" size="100" resizable="true" closable="true" collapsible="true" >
    <h2><h:outputText value="Aplicación Empresarial Cines Luz de Castilla - Segovia"/></h2>
    <h:form>
        <p:toolbar rendered="#{request.isUserInRole('users')}">
            <f:facet name="left">
                <p:commandButton type="button" value="New" icon="ui-icon-document"/>
                <p:commandButton type="button" value="Open" icon="ui-icon-folder-open" />
                <span class="ui-separator">
                    <span class="ui-icon ui-icon-grip-dotted-vertical" />
                </span>
                <p:commandButton type="button" title="Save" icon="ui-icon-disk" />
            </f:facet>

            <f:facet name="right">
                <p:menuButton value="#{loginView.authenticatedUser.name}">
                    <p:menuItem value="Cerrar sesión" action="#{loginView.logout}" icon="ui-icon-power" />
                </p:menuButton>
            </f:facet>
        </p:toolbar>
    </h:form>
</p:layoutUnit>
```

Actualizar template

Cabecera

- ▶ <p:toolbar rendered="#{request.isUserInRole('users')}">
 - Ponemos una barra de herramientas que sólo se mostrará cuando usuarios con privilegios para el rol “users” hagan login
- ▶ <f:facet name="left">
 - A la izquierda de la barra de herramientas ponemos varios botones (como ejemplo, no tienen ninguna funcionalidad de momento)
- ▶ <p:menuButton value="#{loginView.authenticatedUser.name}">
 - A la derecha ponemos un desplegable que tiene como etiqueta el nombre del usuario que ha hecho login
- ▶ <p:menuitem value="Cerrar sesión" action="#{loginView.logout}" icon="ui-icon-power" />
 - Dentro del desplegable con el nombre de usuario ponemos un item para cerrar la sesión

Actualizar template

Menú lateral

```
<p:layoutUnit position="west" size="200" header="Menú" collapsible="true">
    <h:form>
        <p:menu>
            <p:menuItem value="Inicio" outcome="/index.xhtml" icon="ui-icon-home" />
            <p:menuItem value="Iniciar Sesión" outcome="/login.xhtml"
                rendered="#{request.isUserInRole('users')==false}" icon="ui-icon-key"/>
            <p:menuItem value="Registro" outcome="/registro.xhtml"
                rendered="#{request.isUserInRole('users')==false}" icon="ui-icon-pencil"/>
            <p:submenu label ="Usuario" rendered="#{request.isUserInRole('users')}>
                <p:menuItem value="Booking" action="booking" icon="ui-icon-cart"/>
            </p:submenu>
            <p:submenu label ="Administrador" rendered="#{request.isUserInRole('admin')}>
                <p:menuItem value="Películas" outcome="/client/movies.xhtml" icon="ui-icon-calculator"/>
            </p:submenu>
        </p:menu>
    </h:form>
</p:layoutUnit>
```

Actualizar template

Menú lateral

- ▶ <p:menuitem value="Iniciar Sesión" outcome="/faces/login.xhtml" rendered="#{request.isUserInRole('users')}==false" icon="ui-icon-key"/>
 - Añadimos la opción para iniciar sesión, que sólo se mostrará a aquellos usuarios que tengan un rol distintos de users (es decir, a todos los que accedan a la página, y una vez un usuario o el administrador accedan, que no aparezca)
- ▶ <p:menuitem value="Registro" outcome ="/faces/registro.xhtml" rendered="#{request.isUserInRole('users')}==false" icon="ui-icon-pencil"/>
 - Igual que para iniciar sesión pero para el registro
- ▶ <p:submenu label ="Usuario" rendered="#{request.isUserInRole('users')}">
 - Submenú con las opciones permitidas a todos los usuarios con el rol “users”
- ▶ <p:submenu label ="Administrador" rendered="#{request.isUserInRole('admin')}"/>
 - Submenú con las opciones permitidas a todos los usuarios con el rol “admin”

JAAS

Facelets Template localhost:8080/cines/

Aplicación Empresarial Cines Luz de Castilla - Segovia

Mostrando 20 películas en 7 cines!

| Id | Name | Actors |
|----|-------------------------------------|---|
| 1 | The Matrix | Keanu Reeves, Laurence Fishburne, Carrie-Ann Moss |
| 2 | The Lord of The Rings | Elijah Wood, Ian McKellen, Viggo Mortensen |
| 3 | Inception | Leonardo DiCaprio |
| 4 | The Shining | Jack Nicholson, Shelley Duvall |
| 5 | Mission Impossible | Tom Cruise, Jeremy Renner |
| 6 | Terminator | Arnold Schwarzenegger, Linda Hamilton |
| 7 | Titanic | Leonardo DiCaprio, Kate Winslet |
| 8 | Iron Man | Robert Downey Jr, Gwyneth Paltrow, Terence Howard |
| 9 | Inglourious Basterds | Brad Pitt, Diane Kruger |
| 10 | Million Dollar Baby | Hilary Swank, Clint Eastwood |
| 11 | Kill Bill | Uma Thurman |
| 13 | The Hangover | Bradley Cooper, Zach Galifianakis |
| 14 | Toy Story | Tom Hanks, Michael Keaton |
| 15 | Harry Potter | Daniel Radcliffe, Emma Watson |
| 16 | Avatar | Sam Worthington, Sigourney Weaver |
| 17 | Slumdog Millionaire | Anil Kapoor, Dev Patel, Freida Pinto |
| 18 | The Curious Case of Benjamin Button | Brad Pitt, Cate Blanchett |
| 19 | The Bourne Ultimatum | Matt Damon, Julia Stiles |
| 20 | The Pink Panther | Steve Martin, Kevin Kline |

JAAS

Facelets Template localhost:8080/cines/faces/registro.xhtml

Aplicación Empresarial Cines Luz de Castilla - Segovia

Menú

- Inicio
- Iniciar Sesión
- Registro

Crear una nueva cuenta

Nombre: Anibal Bregon

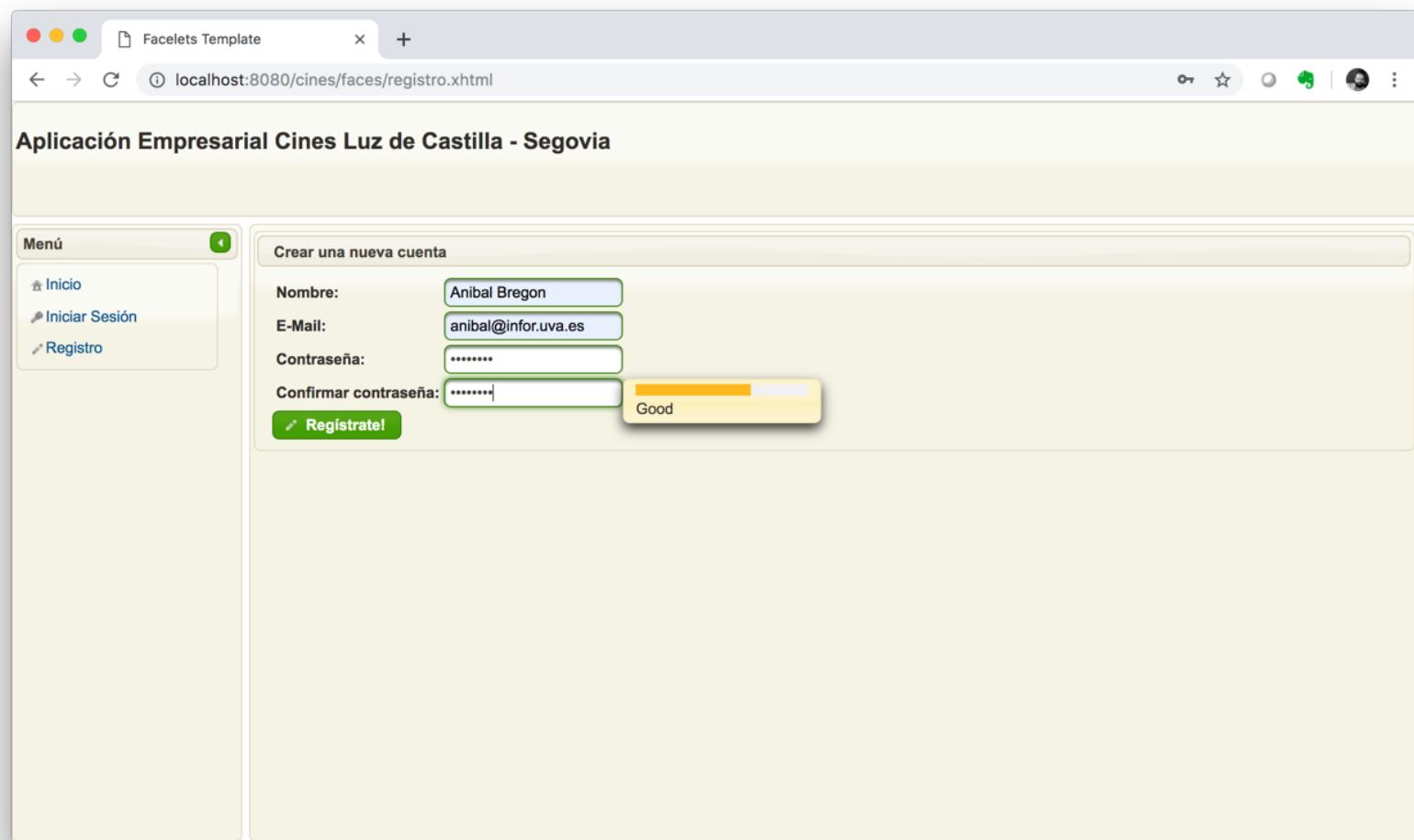
E-Mail: anibal@infor.uva.es

Contraseña:

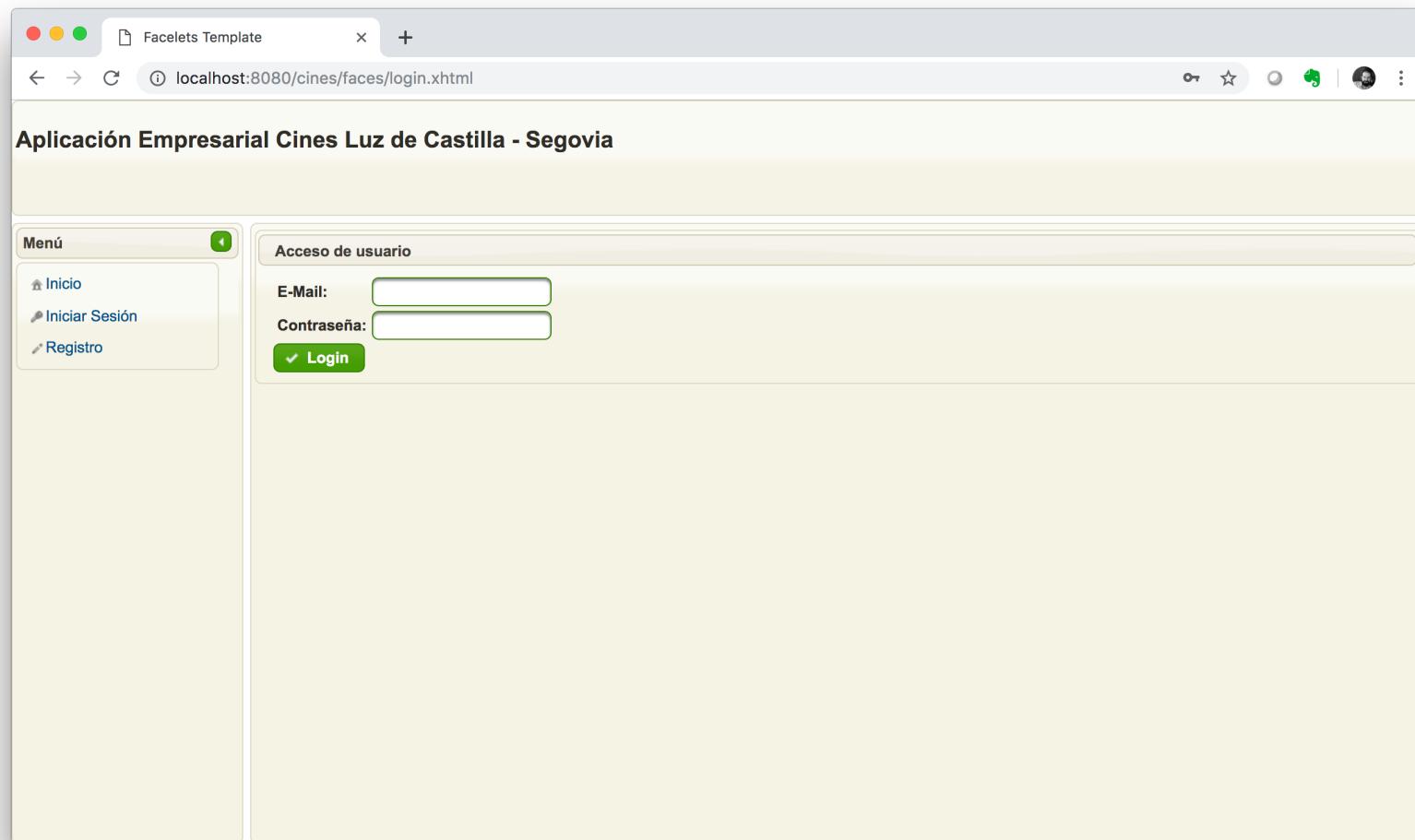
Confirmar contraseña:

Good

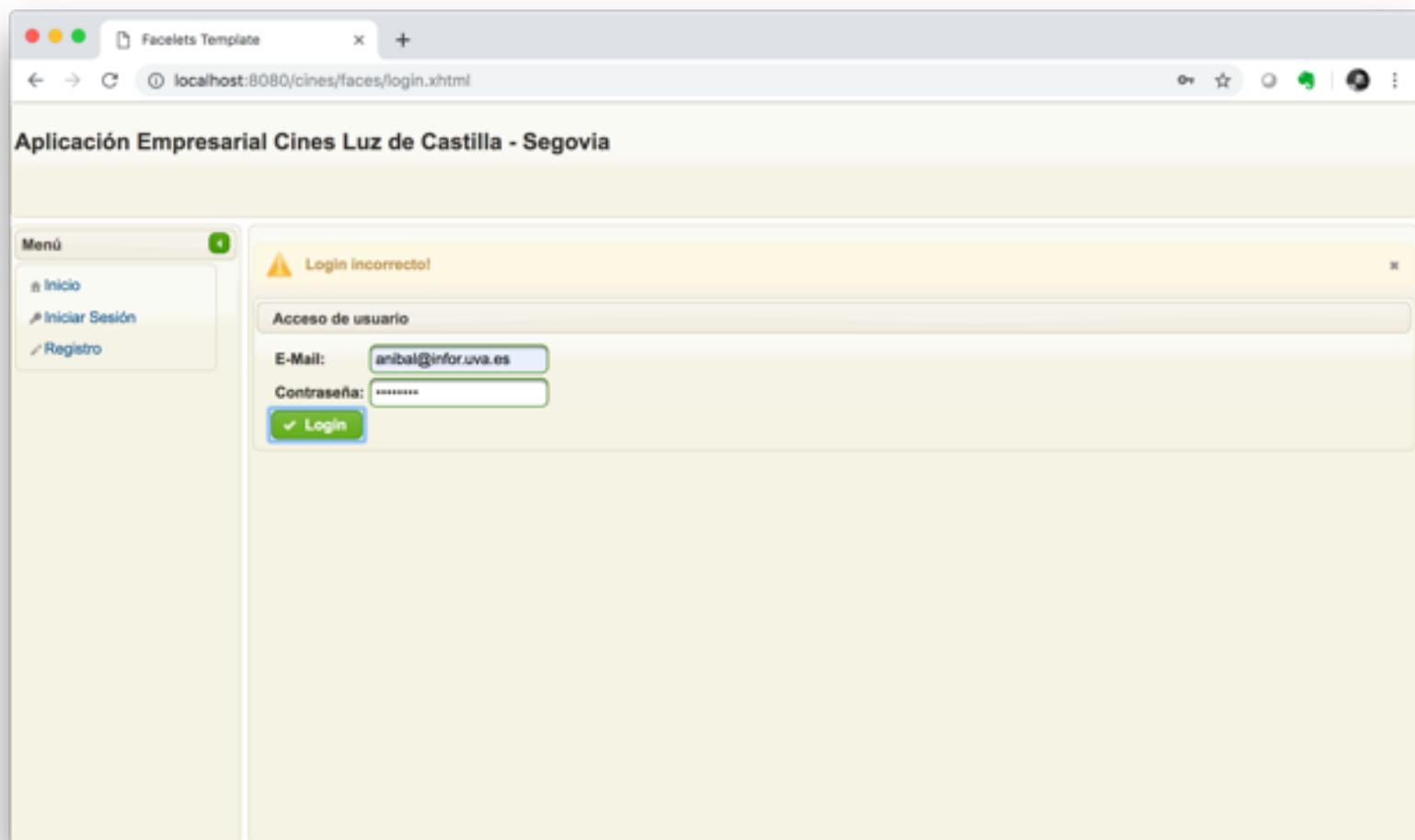
✓ Regístrate!



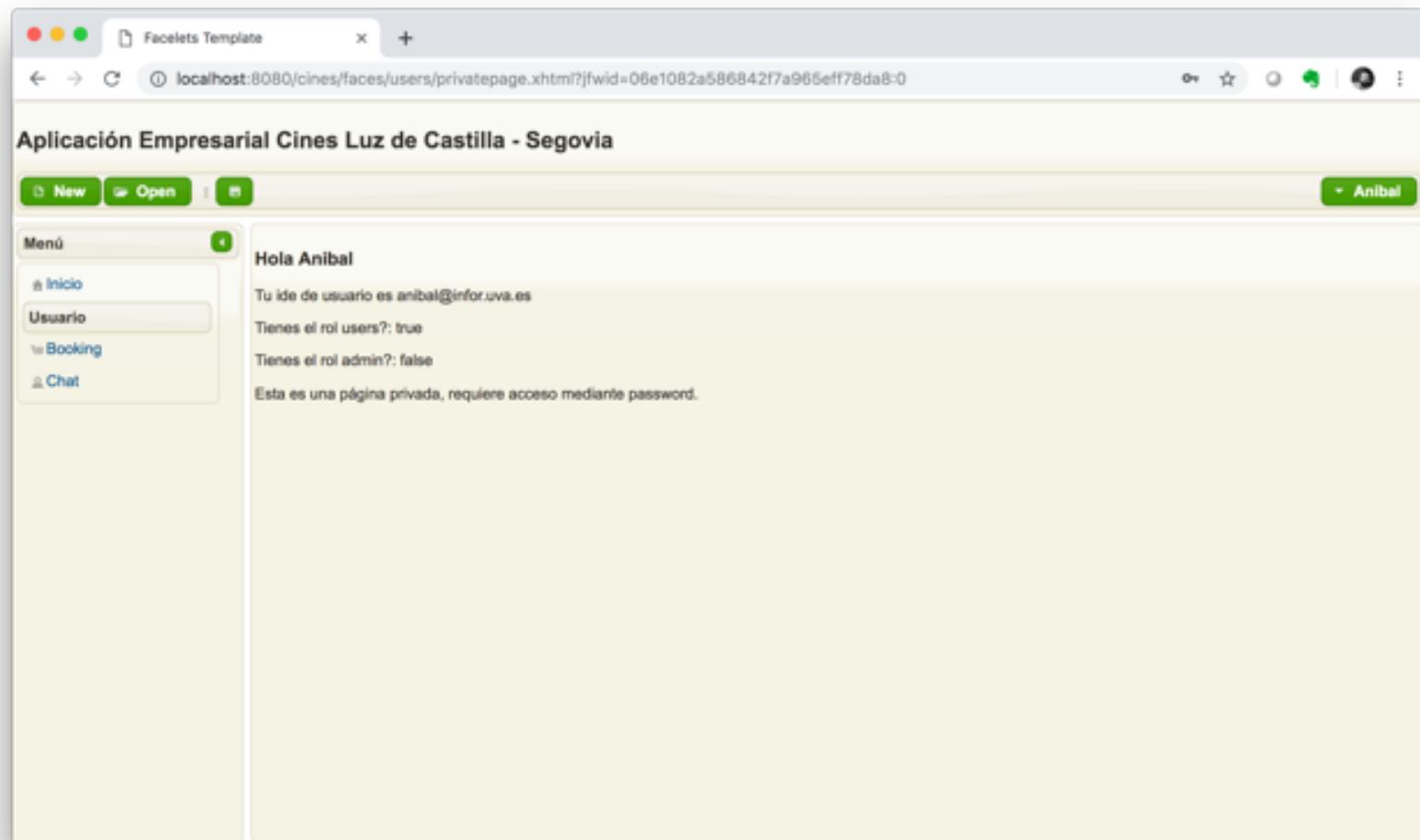
JAAS



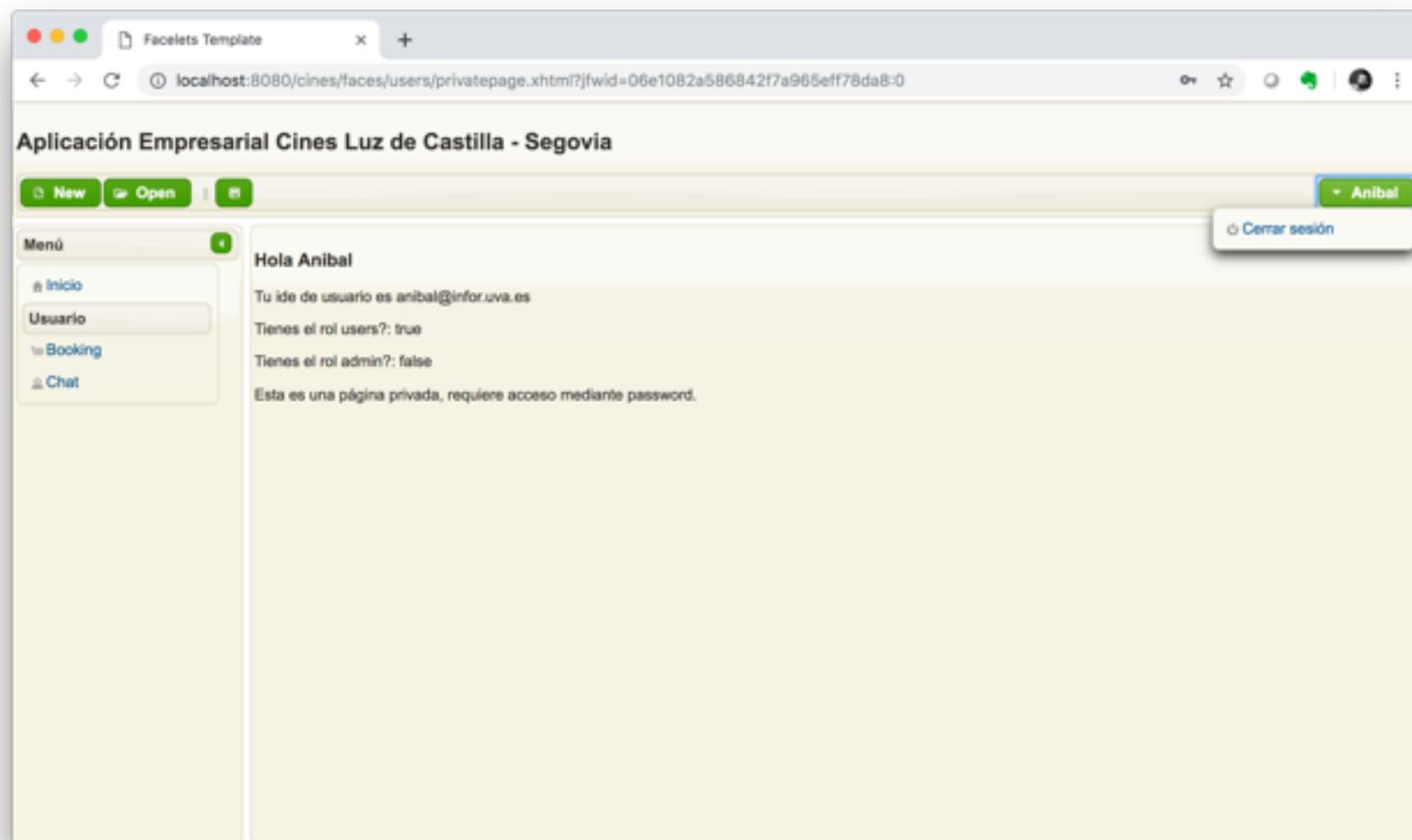
JAAS



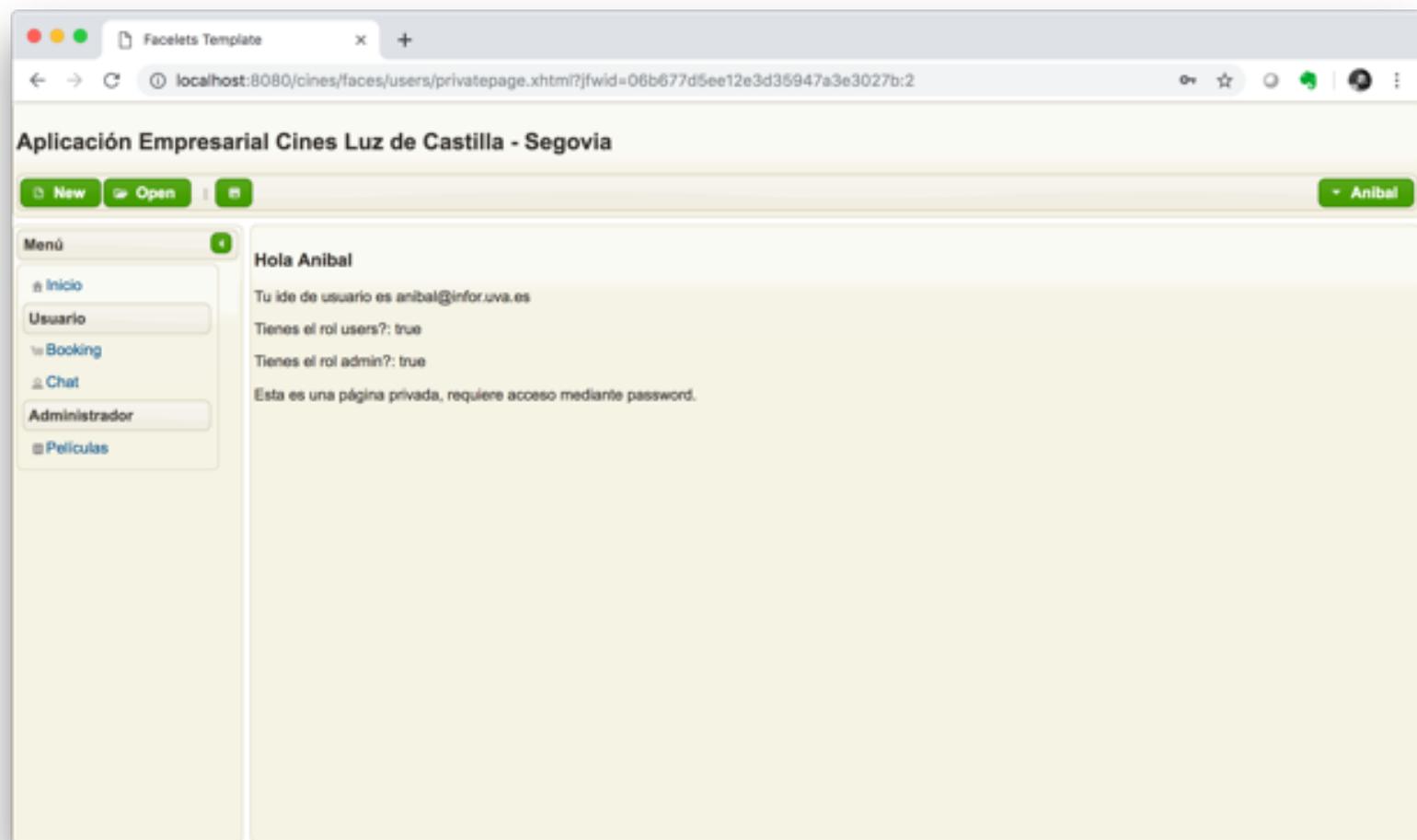
JAAS



JAAS



JAAS



¿PREGUNTAS?