

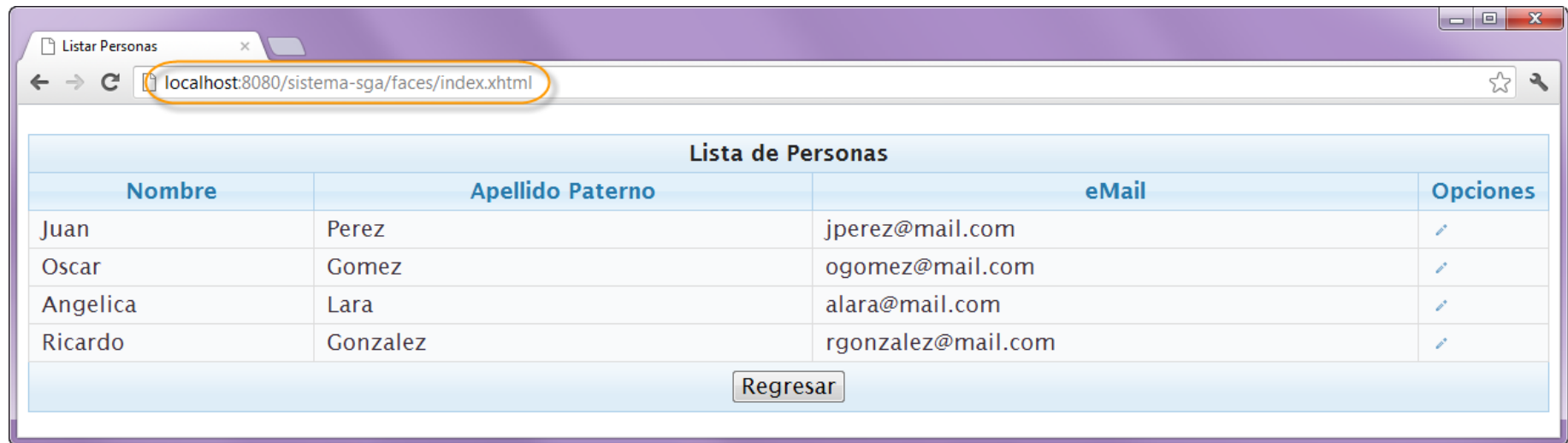






Ejercicio 14

Creación Proyecto SGA con Seguridad
en Java EE

Objetivo del Ejercicio

- El objetivo del ejercicio exponer agregar seguridad los métodos de negocio del EJB de PersonaService, y agregar formas de hacer login por cada cliente creado: Cliente Web, Cliente SOAP Web Service y Cliente REST Web Service y Cliente EJB. El resultado final es que cada cliente debe ejecutarse correctamente al enviar las credenciales respectivas (usuario y password) al sistema-sga. Esta figura es el resultado únicamente del cliente Web (JSF):

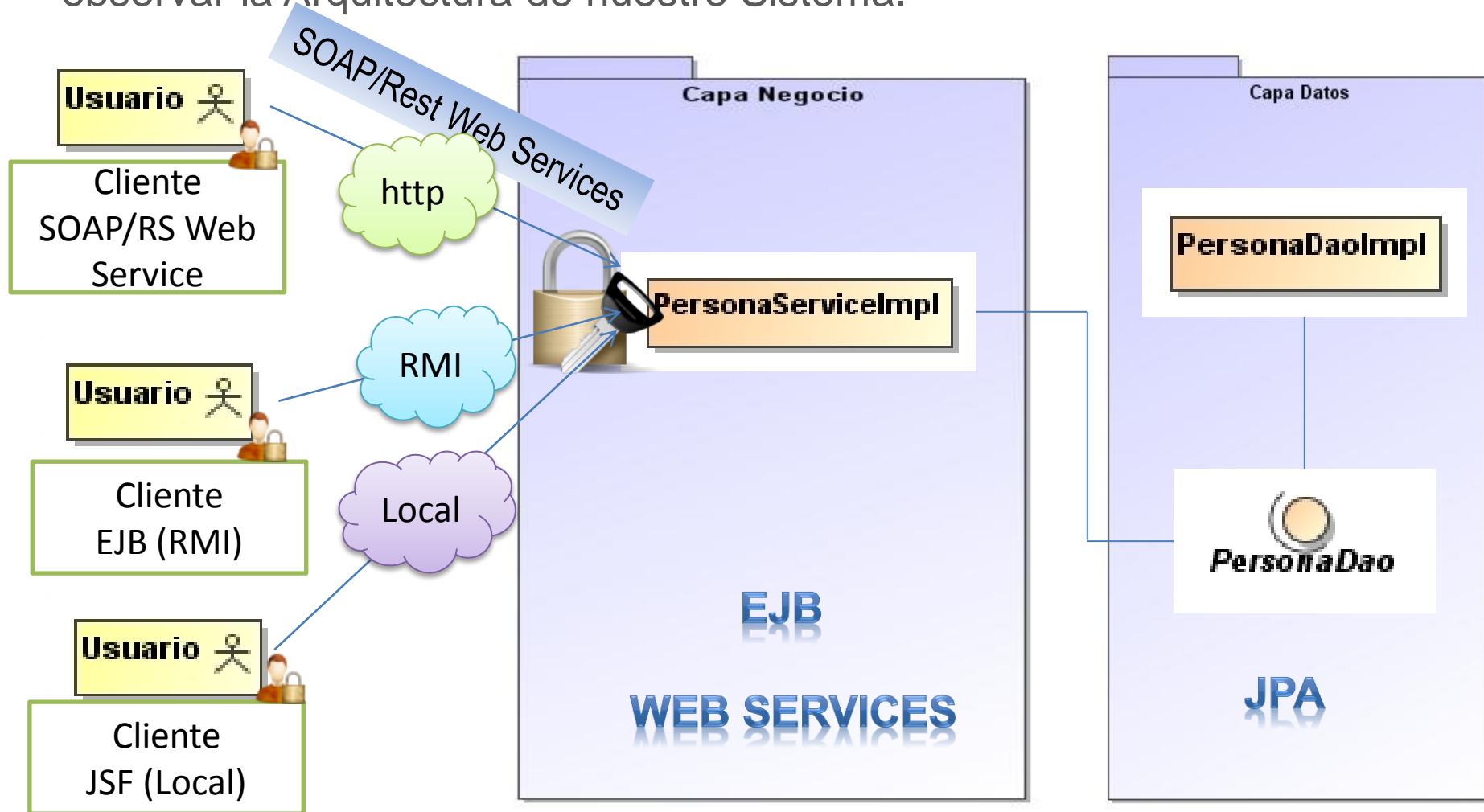


Nombre	Apellido Paterno	eMail	Opciones
Juan	Perez	jperez@mail.com	
Oscar	Gomez	ogomez@mail.com	
Angelica	Lara	alara@mail.com	
Ricardo	Gonzalez	rgonzalez@mail.com	

Regresar

Diagrama de Clases

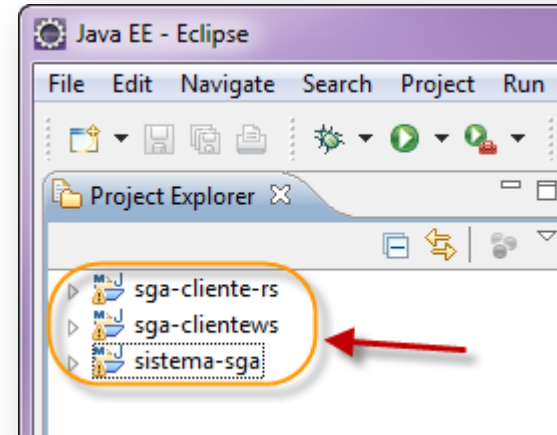
- Este es el Diagrama de Clases del Ejercicio, donde se pueden observar la Arquitectura de nuestro Sistema.



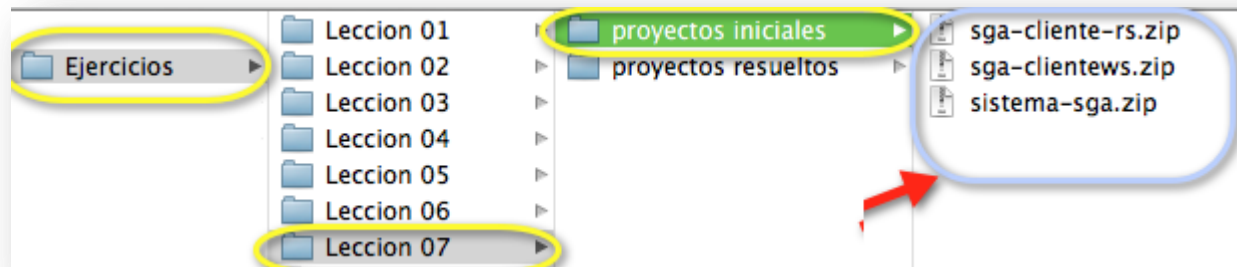
Paso 1. Cargar los proyectos a trabajar

Vamos a utilizar los proyectos siguientes:

- sistema-sga
- sga-clientews
- sga-cliente-rs



Estos proyectos se pueden cargar de la carpeta de Ejercicios del Curso -> Lección 7 -> proyectos iniciales:



Paso 2. Agregar seguridad al EJB PersonaService

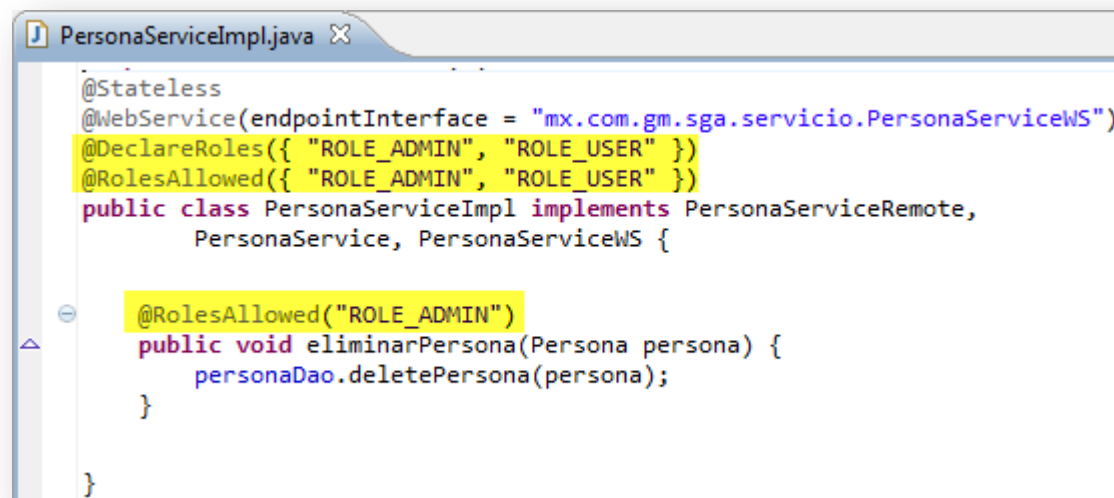
Agregamos las anotaciones siguientes al EJB de `PersonaServiceImpl.java` en la declaración de la clase:

```
@DeclareRoles({ "ROLE_ADMIN", "ROLE_USER" })  
@RolesAllowed({ "ROLE_ADMIN", "ROLE_USER" })
```

Además, al método `eliminarPersona`, le agregamos la anotación:

```
@RolesAllowed("ROLE_ADMIN")
```

El resultado final debe visualizarse como sigue (se omite el código completo):



```
PersonaServiceImpl.java  
  
@Stateless  
@WebService(endpointInterface = "mx.com.gm.sga.servicio.PersonaServiceWS")  
@DeclareRoles({ "ROLE_ADMIN", "ROLE_USER" })  
@RolesAllowed({ "ROLE_ADMIN", "ROLE_USER" })  
public class PersonaServiceImpl implements PersonaServiceRemote,  
    PersonaService, PersonaServiceWS {  
  
    @RolesAllowed("ROLE_ADMIN")  
    public void eliminarPersona(Persona persona) {  
        personaDao.deletePersona(persona);  
    }  
  
}
```

Paso 3. Verificar que los clientes fallan

Una vez agregada la anotación y para completar esta tarea solo es necesario ejecutar cada cliente (Cliente Web, EJB, REST y SOAP Web Service) y comprobar que ya no tenemos acceso a los métodos del EJB. Ej. Este es el resultado del cliente Web:

The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/sistema-sga/faces/index.xhtml`. The page title is "HTTP Status 500 -". The content area displays an "Exception report" with the following details:

- type** Exception report
- message**
- description** The server encountered an internal error () that prevented it from fulfilling this request.
- exception**
`javax.servlet.ServletException: Se ha producido un error al realizar la inyección de recurso en el bean administrado persona`
- root cause**
`javax.ejb.AccessLocalException: Client not authorized for this invocation`

A note at the bottom states: "The full stack traces of the exception and its root causes are available in the GlassFish Server Open Source Edition 3.1.2 logs."



Paso 4. Modificar el cliente Web

Para dar acceso al cliente Web necesitamos realizar varios pasos, a continuación vamos a ver cada uno:

Modificamos el archivo web.xml

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Login in</realm-name>
</login-config>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Aplicacion WEB JSF</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <role-name>ROLE_ADMIN</role-name>
    <role-name>ROLE_USER</role-name>
    <role-name>ROLE_GUEST</role-name>
  </auth-constraint>
</security-constraint>
```


Paso 4. Modificar el cliente Web (cont)

Configuración File Realms en GlassFish. Entramos a la consola de GlassFish -> Configurations -> server-config -> Security. Habilitamos **Default Principal To Role Mapping**. Esta opción indica que el nombre del grupo es el mismo que el del rol, y por lo tanto ya no es necesario agregar el archivo glassfish-web.xml al proyecto.

The screenshot displays the GlassFish Administration Console interface. On the left, the 'Tree' pane shows the navigation structure: 'Common Tasks', 'Domain' (containing 'server (Admin Server)' and 'Clusters'), and 'Configurations' (containing 'default-config' and 'server-config'). The 'Security' option under 'server-config' is selected and highlighted with a red circle. A red arrow points from this 'Security' entry to the main configuration area. The main area is titled 'Security' and contains the following settings:

- Configuration Name:** server-config
- Default Principal To Role Mapping:** Enabled (checkbox checked). Description: Apply default principal-to-role mapping at deployment time if mapping is not defined; does not affect current.
- Mapped Principal Class:** (empty text field). Description: Customize the java.security.Principal implementation principal-to-role mapping.
- JACC:** default (dropdown menu). Description: Required if Default Principal contains a value.
- Audit Modules:** default (text field). Description: List of audit provider modules used by the audit to multiple-select.

At the bottom, there is an 'Additional Properties (1)' section with 'Add Property' and 'Delete Properties' buttons. A red arrow points from the 'Save' button in the top right corner of the main configuration area to the 'Save' button in the 'Additional Properties' section.

Paso 4. Modificar el cliente Web (cont)

Ahora configuramos la opción de Security-> Realms -> file

The screenshot shows the JBoss Administration Console. On the left, a tree view displays the configuration structure. The 'server-config' folder is expanded, showing sub-items like 'JVM Settings', 'Logger Settings', 'Web Container', 'EJB Container', 'Java Message Service', 'Security', 'Realms', 'admin-realm', 'certificate', and 'file'. The 'Security' and 'Realms' folders are highlighted with orange circles. The 'file' realm is selected and highlighted with a blue background. A blue arrow points from the 'file' realm to the 'Manage Users' button in the 'Edit Realm' dialog.

The 'Edit Realm' dialog is open, showing the 'Manage Users' button highlighted with an orange circle. A red arrow points to this button. The dialog title is 'Edit Realm' and it contains the text 'Edit an existing security (authentication) realm.' Below this, the 'Configuration Name' is set to 'server-config'. The 'Additional Properties (0)' section is empty, showing a table with columns 'Name' and 'Value'. The 'Save' and 'Cancel' buttons are visible at the bottom right of the dialog.

Tree

- Common Tasks
- Domain
 - server (Admin Server)
 - Clusters

Configuration Name: server-config

* Indicates required field

default-config

- server-config
 - JVM Settings
 - Logger Settings
 - Web Container
 - EJB Container
 - Java Message Service
 - Security
 - Realms
 - admin-realm
 - certificate
 - file
 - Audit Modules
 - JACC Providers

Additional Properties (0)

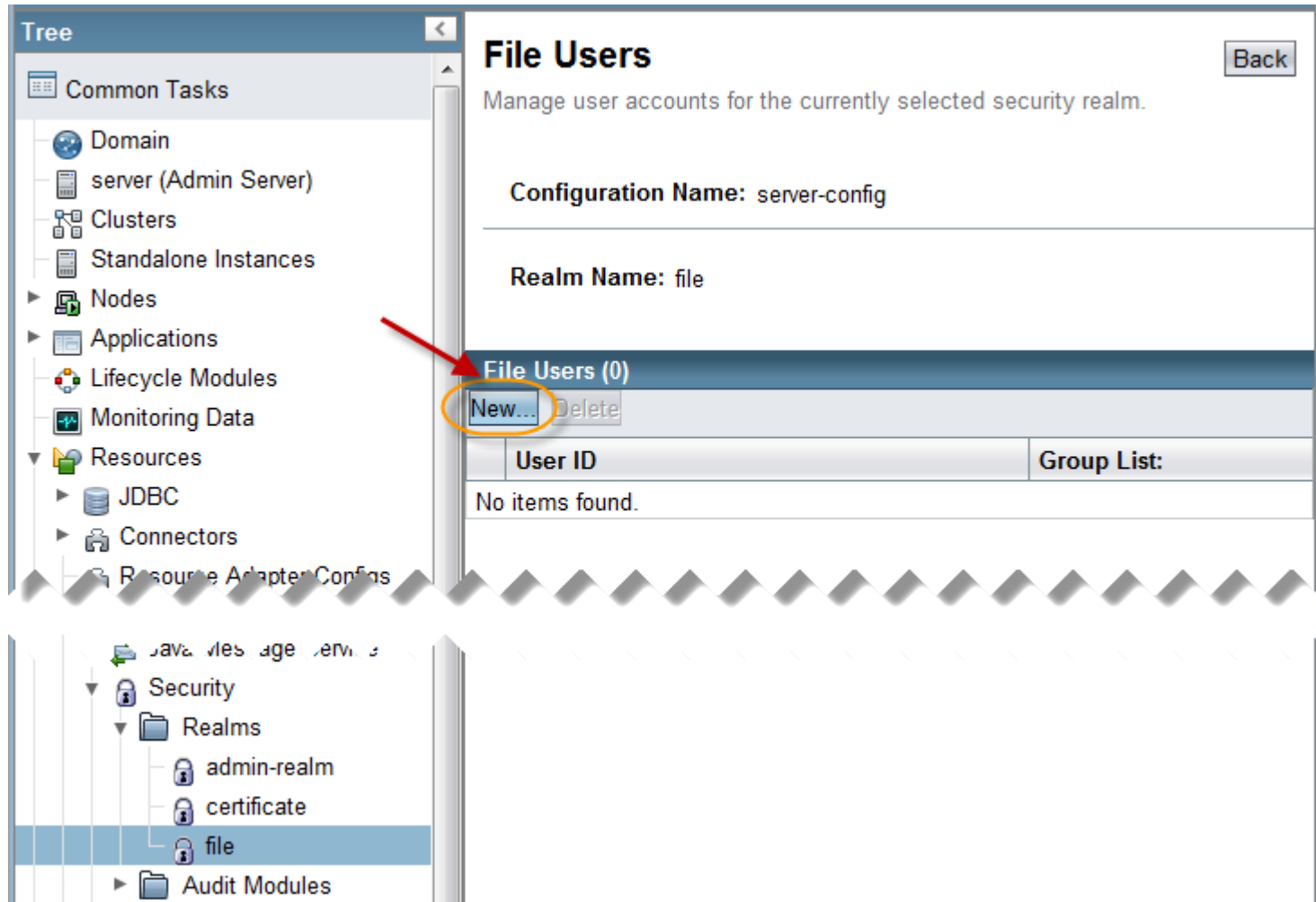
Add Property Delete Properties

Name	Value
No items found.	

Save Cancel

Paso 4. Modificar el cliente Web (cont)

Esta es la pantalla de administración de usuarios. Agregamos uno nuevo:



Paso 4. Modificar el cliente Web (cont)

Agregamos un usuario con los siguientes datos:

New File Realm User
Create new user accounts for the currently selected security realm.

OK Cancel

* Indicates required field

Configuration Name: server-config

Realm Name: file

User ID: * admin
Name can be up to 255 characters, must contain only alphanumeric, underscore, dash, or dot characters

Group List: ROLE_ADMIN, ROLE_USER, ROLE_GUEST
Separate multiple groups with colon

New Password: admin

Confirm New Password:

Paso 4. Modificar el cliente Web (cont)

Repetimos el proceso anterior, hasta agregar los siguientes usuarios, con su respectivo password y grupo:

File Users
Manage user accounts for the currently selected security realm.

Configuration Name: server-config

Realm Name: file

File Users (3)

New... Delete

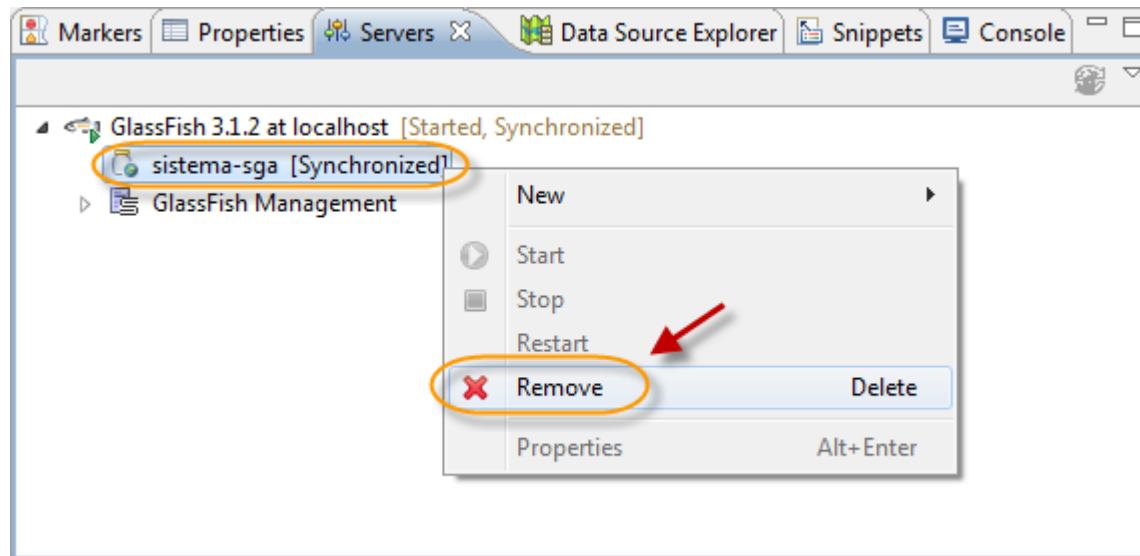
User ID	Group List:
<input type="checkbox"/> admin	ROLE_ADMIN ROLE_USER ROLE_GUEST
<input type="checkbox"/> guest	ROLE_GUEST
<input type="checkbox"/> user	ROLE_USER

Annotations in the image:

- password: admin (pointing to the admin user)
- password: guest (pointing to the guest user)
- password: user (pointing to the user user)

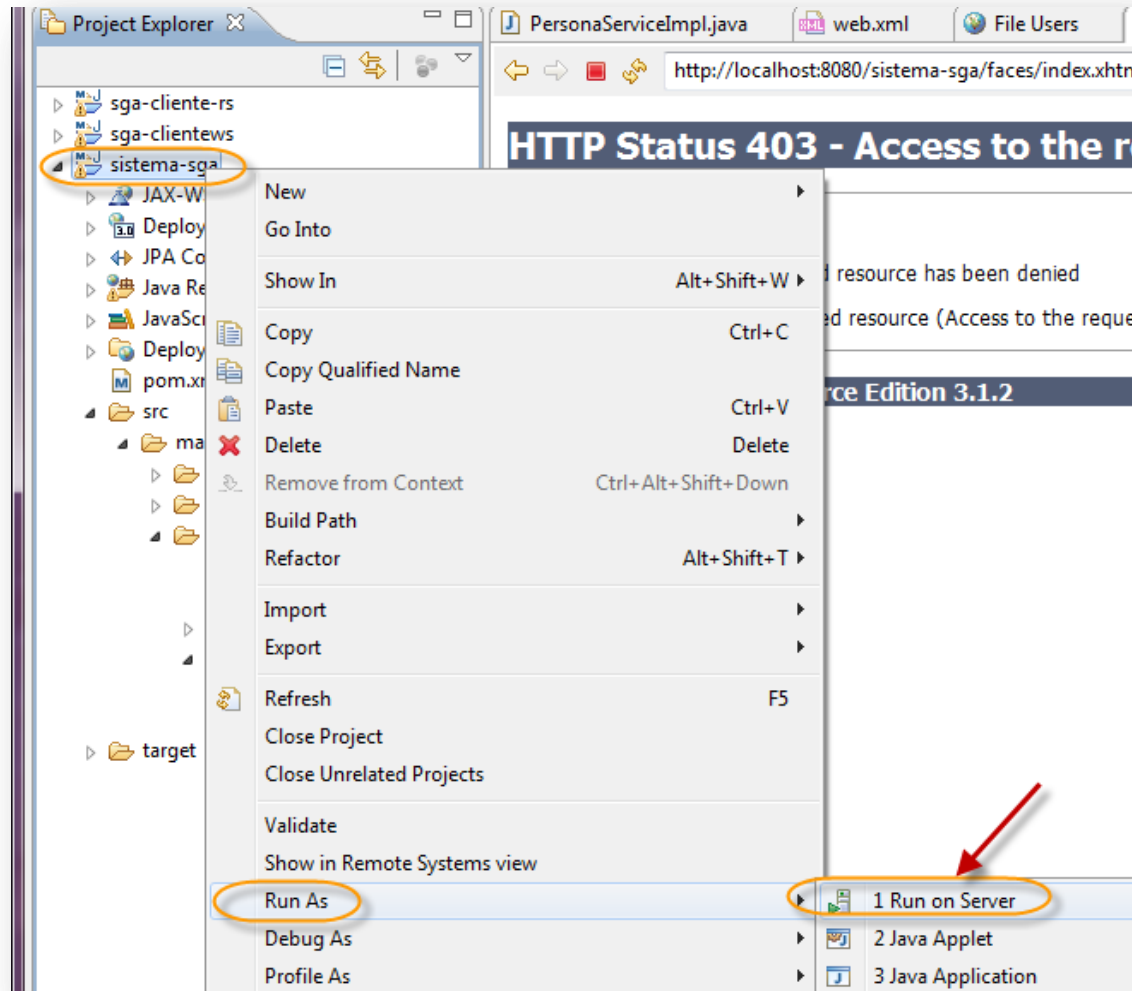
Paso 4. Modificar el cliente Web (cont)

Es necesario hacer undeploy de la aplicación sistema-sga para que se reflejen los cambios:



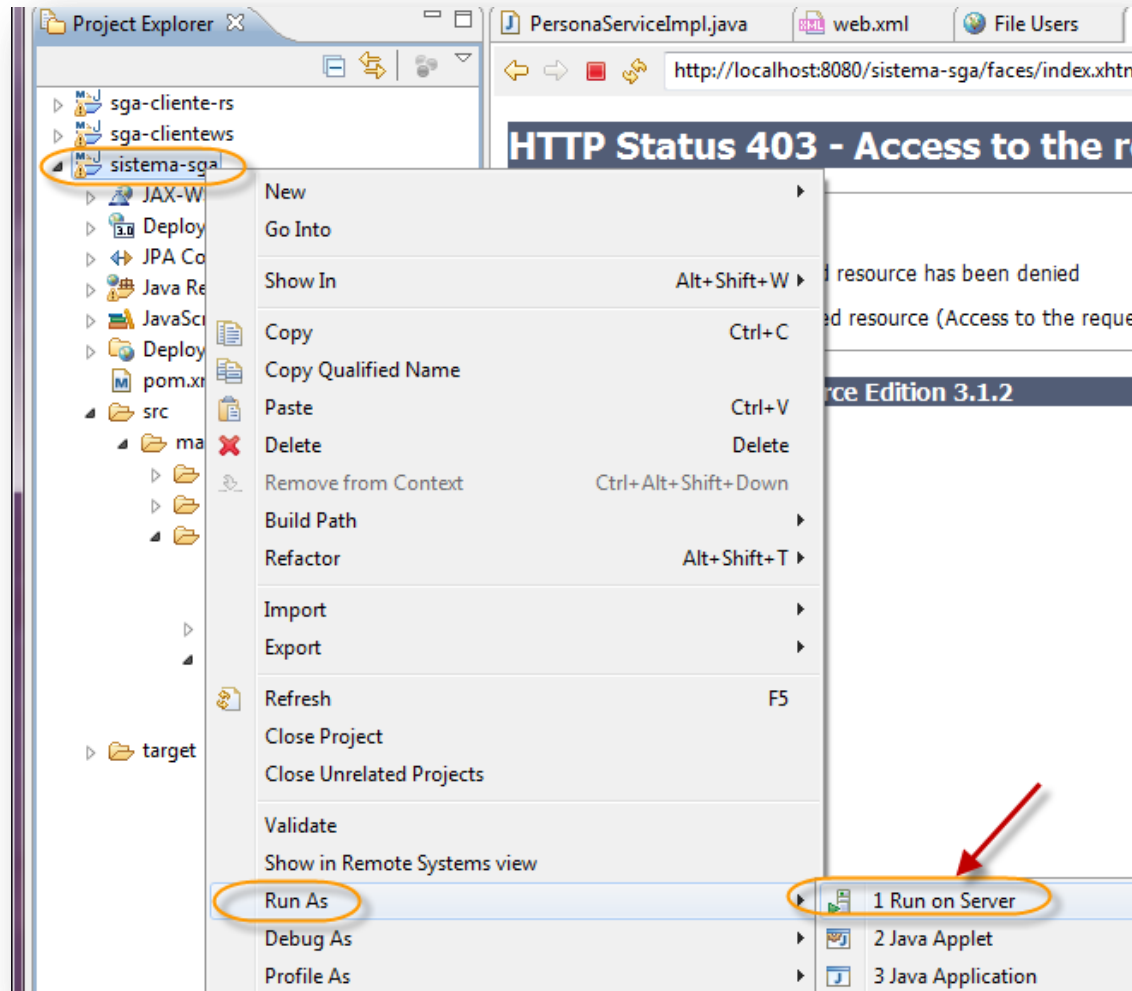
Paso 4. Modificar el cliente Web (cont)

Desplegamos nuevamente la aplicación sistema-sga:



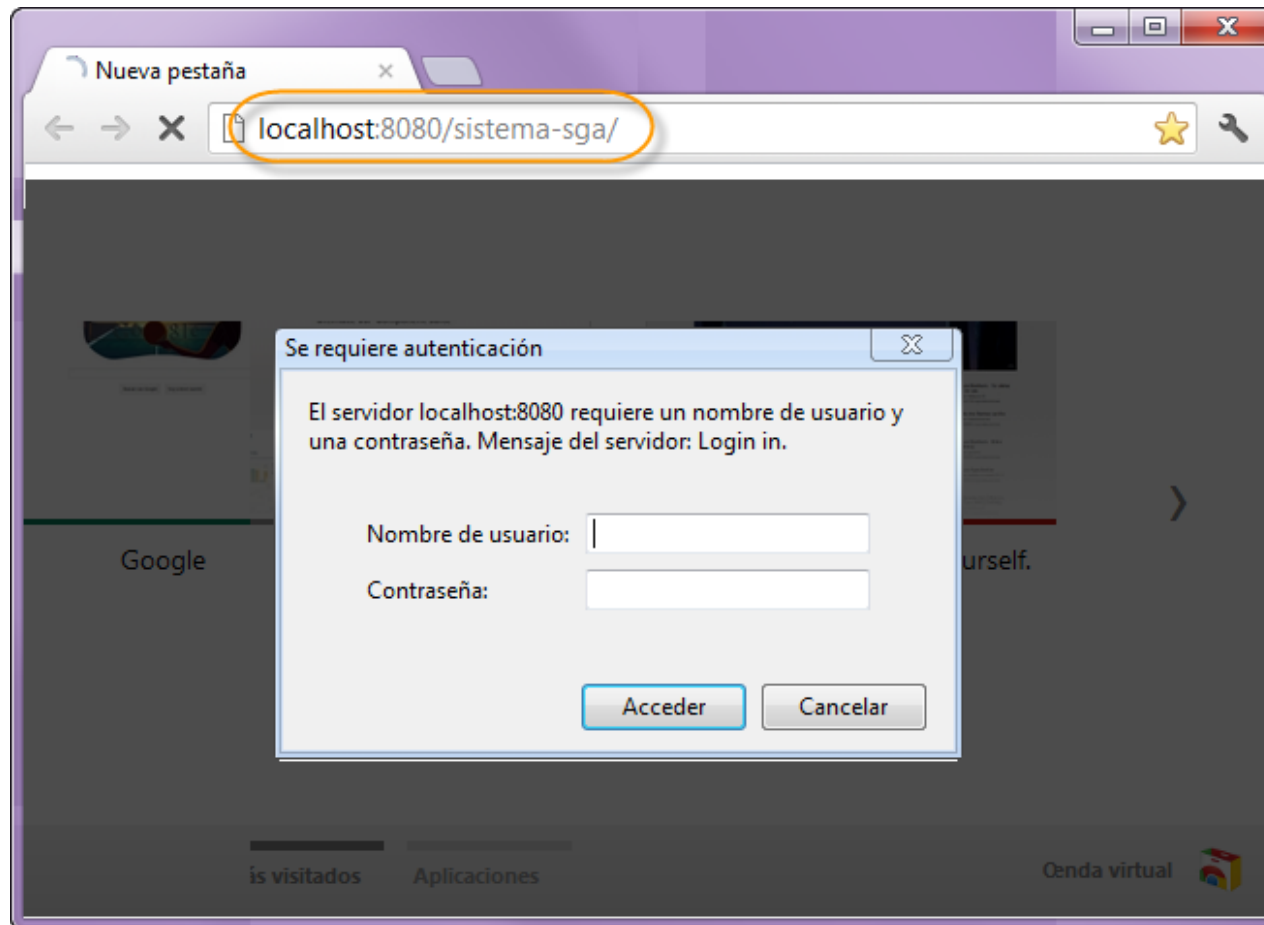
Paso 4. Modificar el cliente Web (cont)

Desplegamos nuevamente la aplicación sistema-sga:



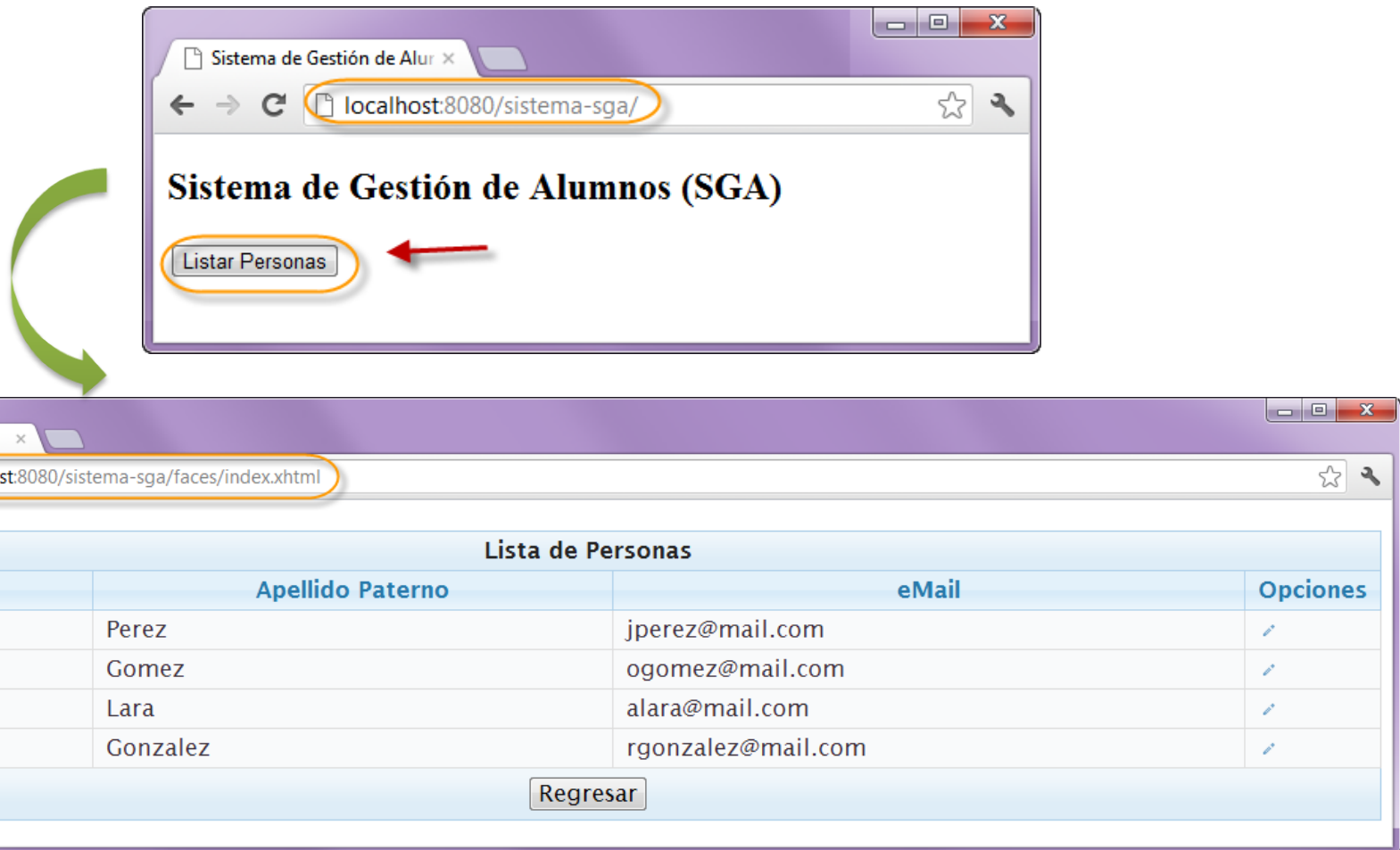
Paso 4. Modificar el cliente Web (cont)

Podemos probar con valores erróneos, y con valores correctos:
Usuario: admin , Password: admin



Paso 4. Modificar el cliente Web (cont)

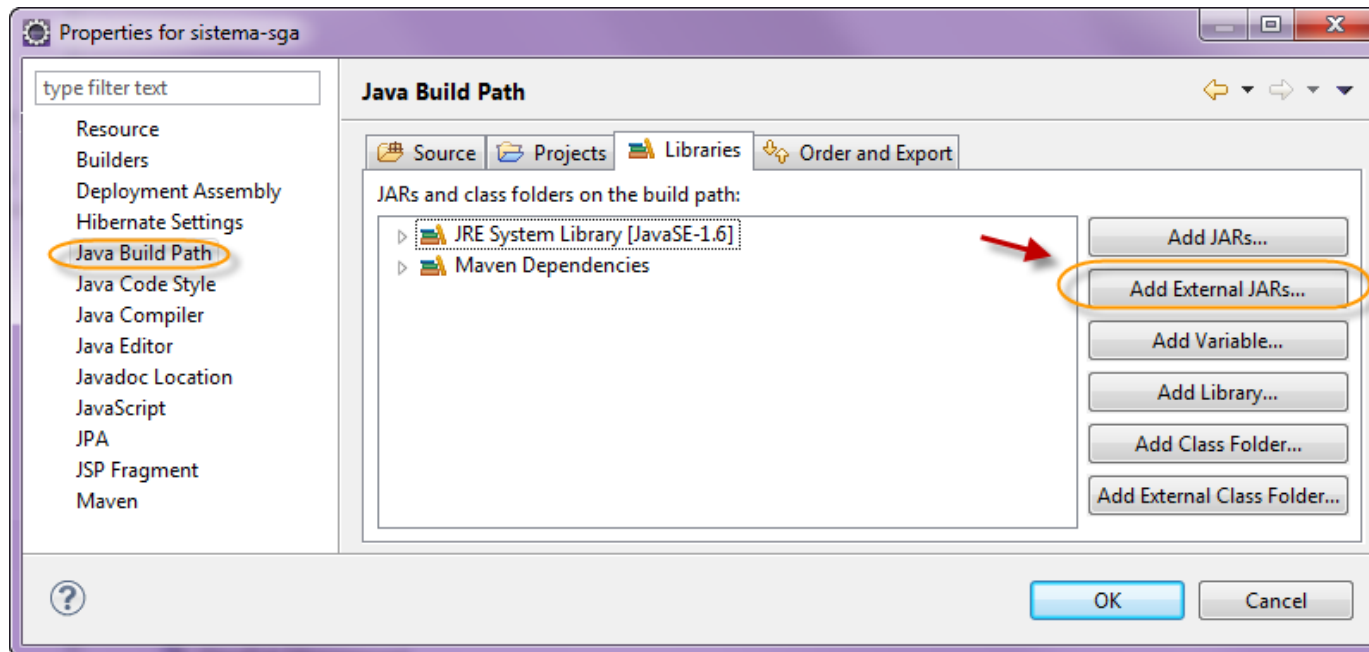
Una vez proporcionado los datos correctamente, nos mostrará las pantallas del sistema:



Paso 5. Modificación del Cliente EJB

Vamos a validar a continuación el cliente EJB. Primero agregamos las siguientes librerías de GlassFish al proyecto sistema-sga o a un nuevo proyecto cliente si se desea. **Nota:** Esta solución es específica para GlassFish, otros servidores de aplicaciones ofrecen otras formas de agregar seguridad de clientes EJB.

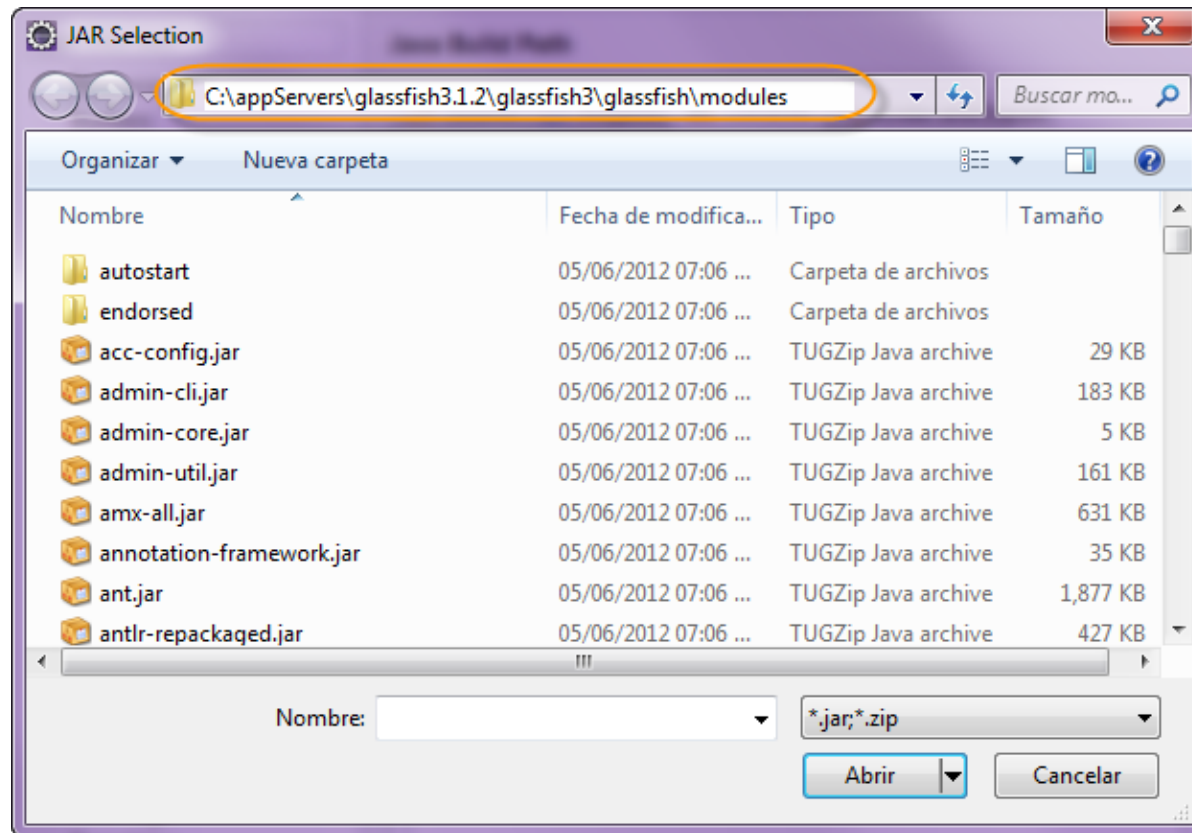
- gf-client-module.jar
- auto-depends.jar
- security.jar



Paso 5. Modificación del Cliente EJB (cont)

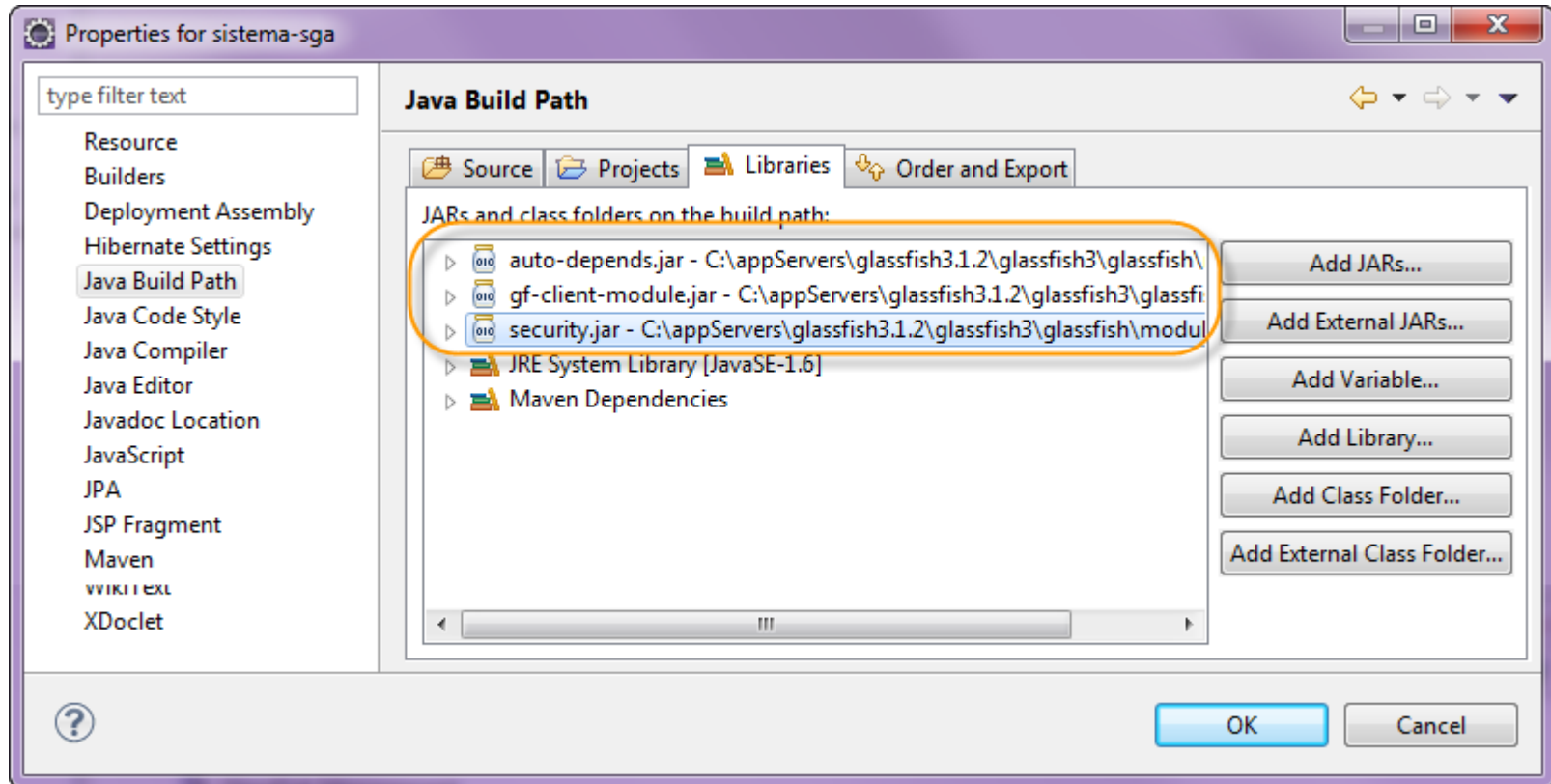
Ubicamos los .jar de la instalación de GlassFish, y los agregamos al classpath de nuestra aplicación:

Ej. C:\appServers\glassfish3.1.2\glassfish3\glassfish\modules



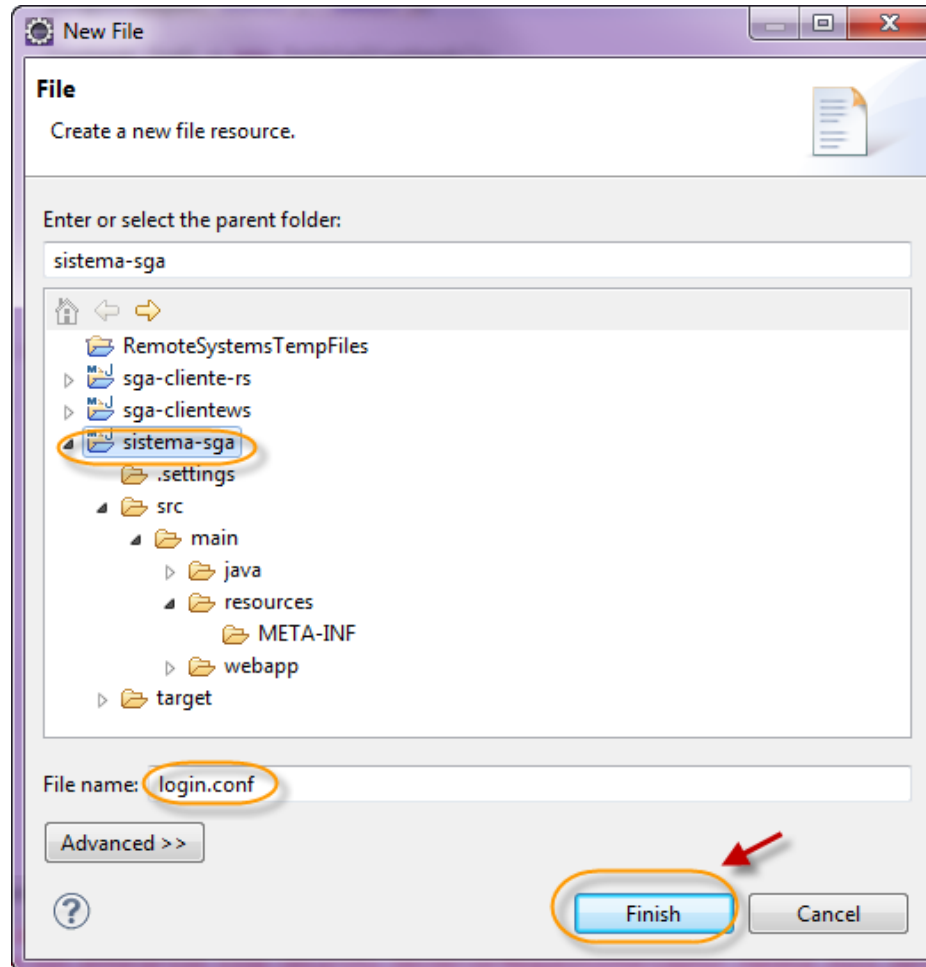
Paso 5. Modificación del Cliente EJB (cont)

Una vez ubicados y agregados los .jar listados, deberemos ver la siguiente figura:



Paso 5. Modificación del Cliente EJB (cont)

Agregamos un archivo de configuración llamado login.conf a nivel raíz de la aplicación. Esto es necesario para poder hacer login vía el cliente EJB:





Paso 5. Modificación del Cliente EJB (cont)

Agregamos el siguiente contenido al archivo login.conf:

```
default {  
com.sun.enterprise.security.auth.login.ClientPasswordLoginModule required;  
};
```


Paso 5. Modificación del Cliente EJB (cont)

Modificamos la clase ClientePersonaService.java. Agregamos el siguiente código antes de la llamada al contexto de JNDI:

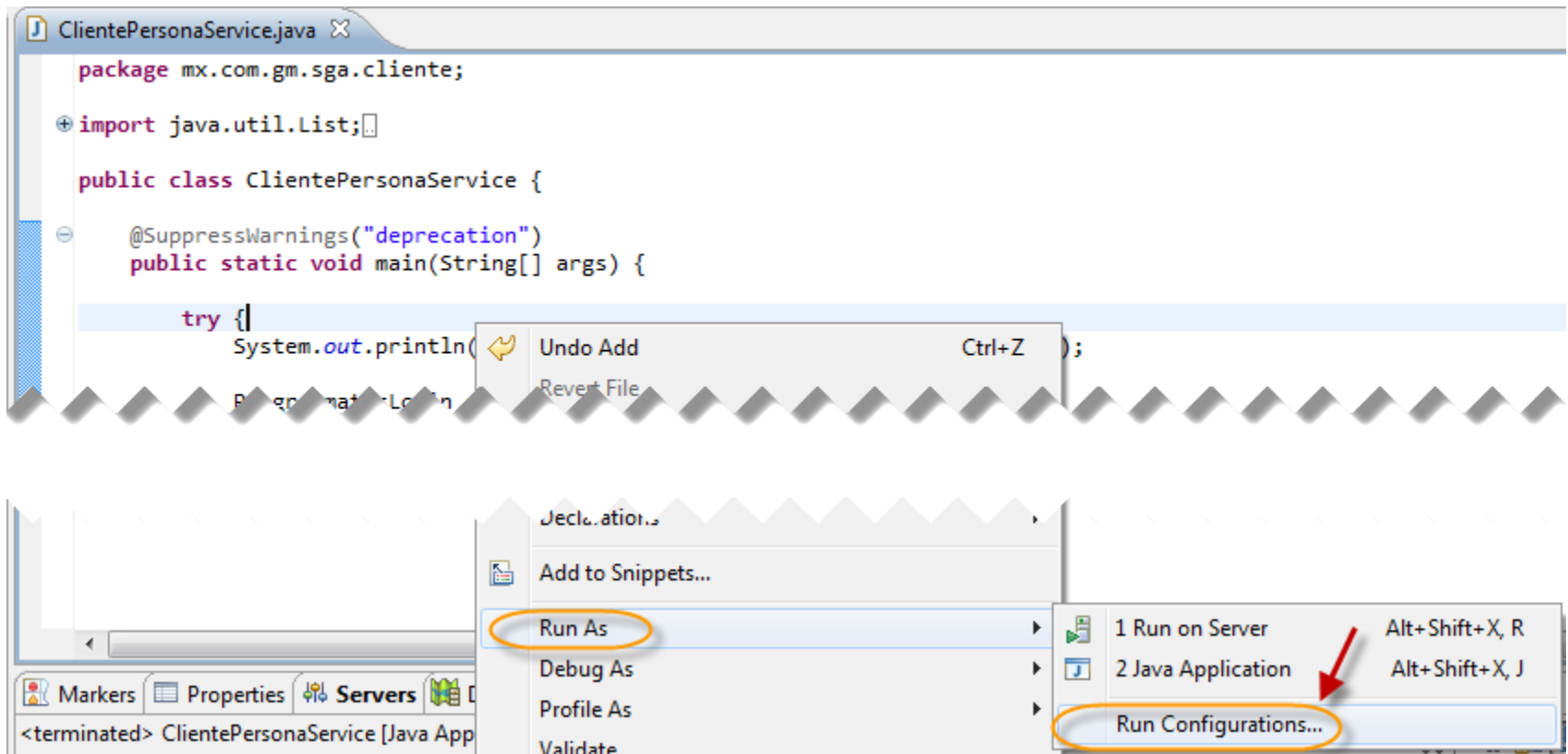
```
ProgrammaticLogin pLogin = new ProgrammaticLogin();  
pLogin.login("admin", "admin");
```

El resultado debe ser similar a la siguiente figura:

```
public class ClientePersonaService {  
  
    @SuppressWarnings("deprecation")  
    public static void main(String[] args) {  
  
        try {  
            System.out.println("Iniciando llamada al EJB desde el cliente\n");  
  
            ProgrammaticLogin pLogin = new ProgrammaticLogin();  
            pLogin.login("admin", "admin");  
  
            Context jndi = new InitialContext();  
  
            PersonaServiceRemote personaService =  
                (PersonaServiceRemote) jndi.lookup("java:global/sistema-sga.  
            PersonaServiceRemote personaService =  
                (PersonaServiceRemote) jndi.lookup("java:global/sistema-sga.  
            System.out.println("\nFin llamada al EJB desde el cliente");  
        } catch (NamingException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Paso 5. Modificación del Cliente EJB (cont)

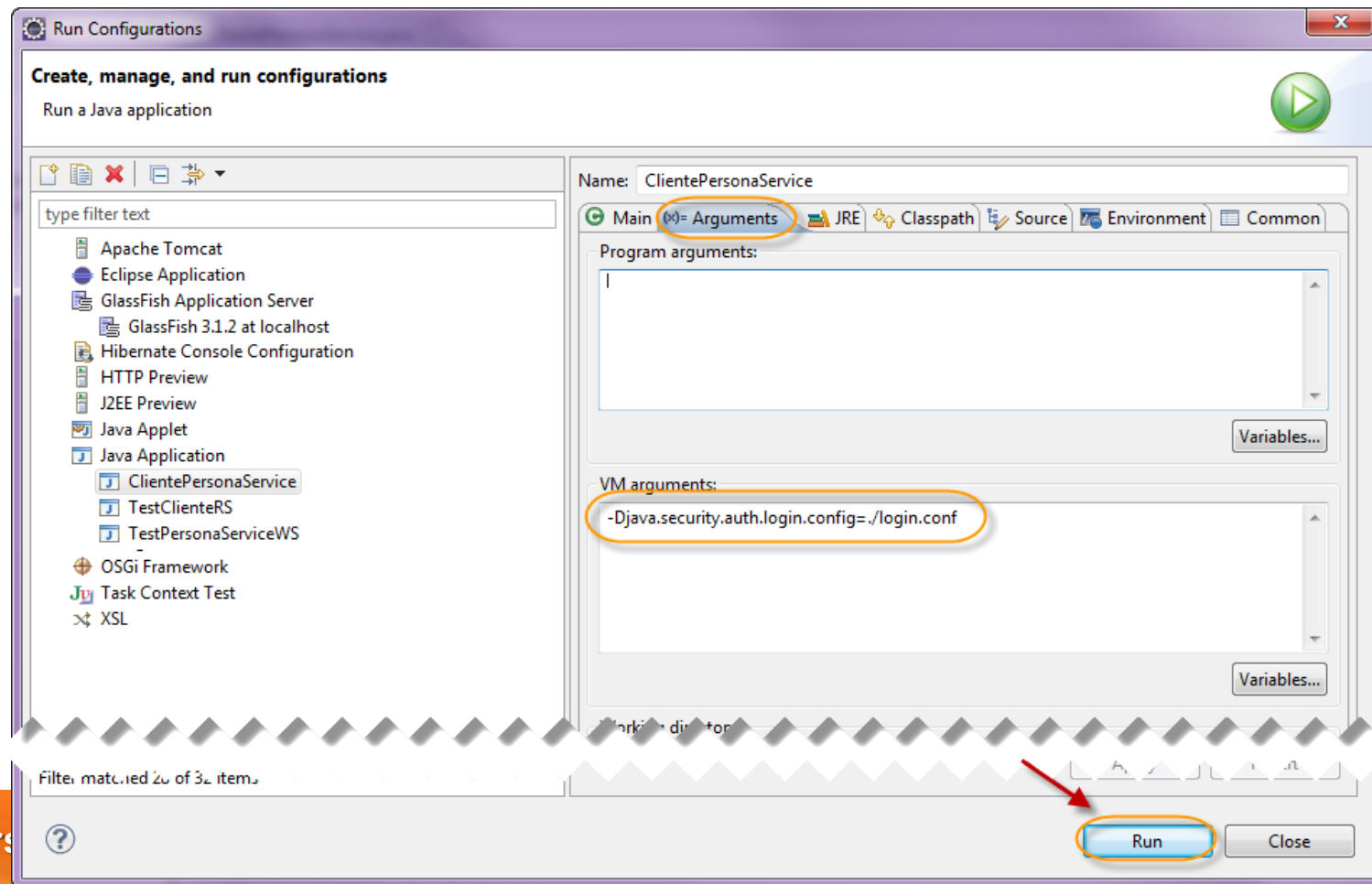
Ejecutamos la clase ClientePersonaService.java, agregando el archivo login.conf en la ejecución de la clase.



Paso 5. Modificación del Cliente EJB (cont)

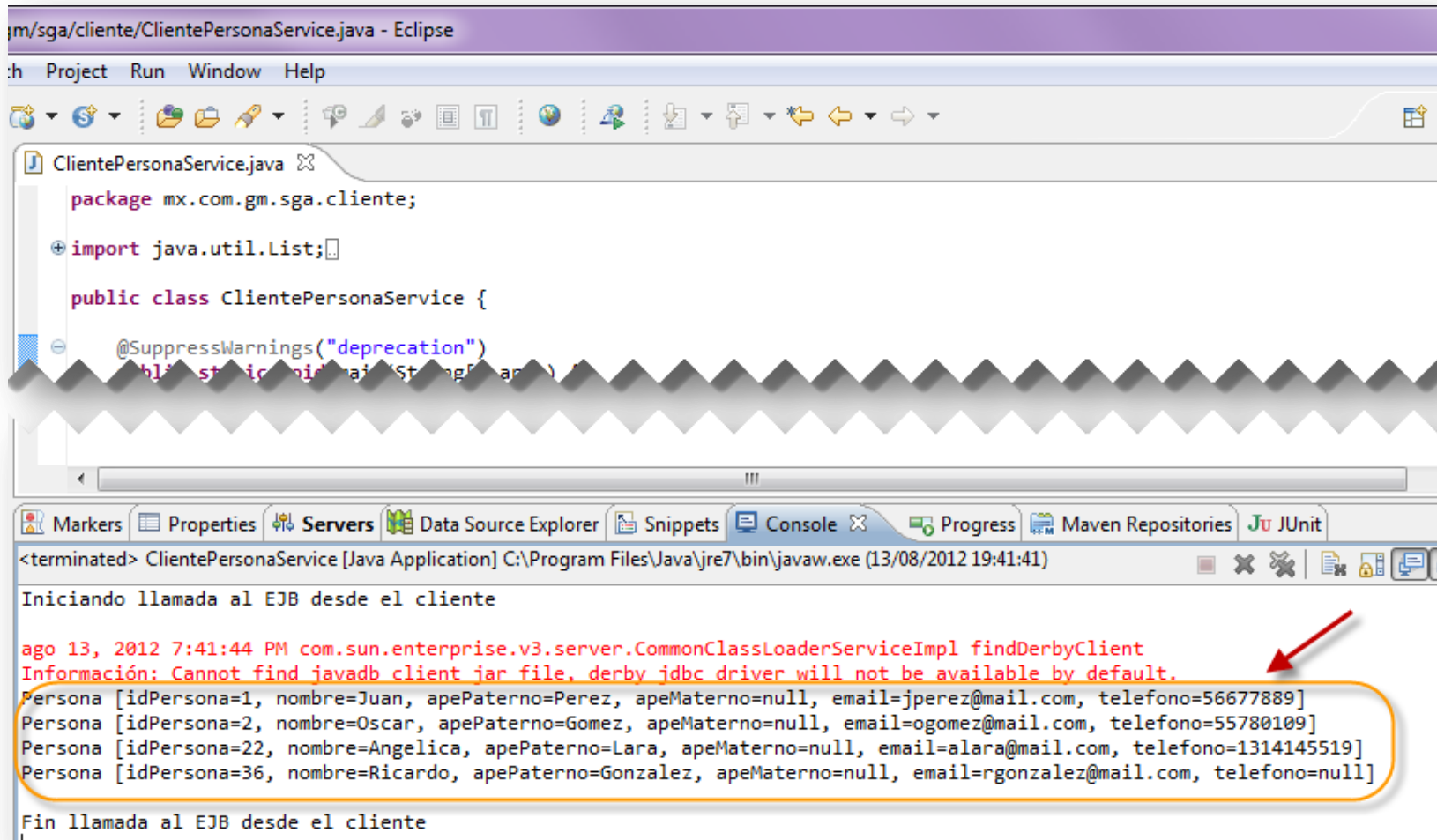
Agregamos el siguiente parámetro en la sección de Argumentos:

-Djava.security.auth.login.config=./login.conf



Paso 5. Modificación del Cliente EJB (cont)

Al ejecutar la clase, podemos observar el listado de personas.



The screenshot shows the Eclipse IDE with the file `ClientePersonaService.java` open. The code defines a package `mx.com.gm.sga.cliente`, imports `java.util.List`, and defines a public class `ClientePersonaService`. The class has a method `findDerbyClient` that returns a list of `Persona` objects. The console output shows the execution of the class, displaying the list of persons and a warning message about the Derby JDBC driver.

```
package mx.com.gm.sga.cliente;

import java.util.List;

public class ClientePersonaService {

    @SuppressWarnings("deprecation")
    public List<Persona> findDerbyClient() {
        // ...
    }
}
```

Console Output:

```
<terminated> ClientePersonaService [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (13/08/2012 19:41:41)

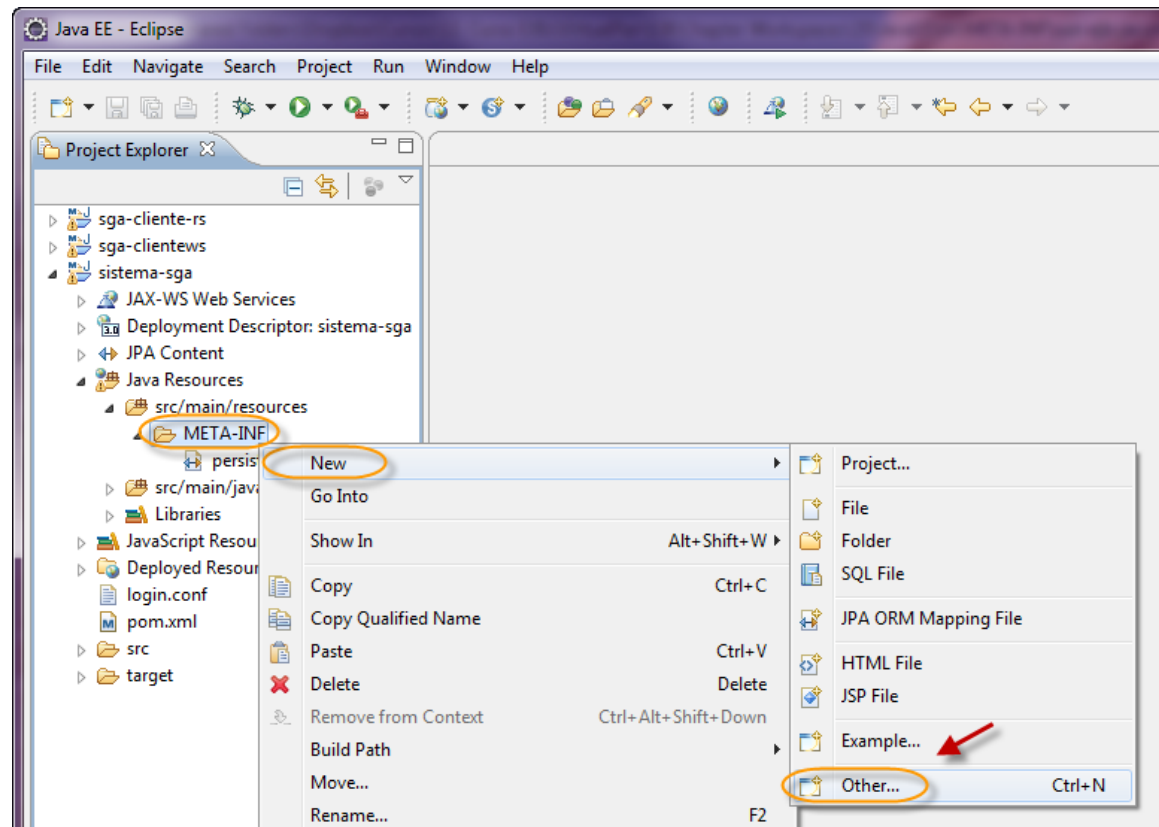
Iniciando llamada al EJB desde el cliente

ago 13, 2012 7:41:44 PM com.sun.enterprise.v3.server.CommonClassLoaderServiceImpl findDerbyClient
Información: Cannot find javadb client jar file, derby jdbc driver will not be available by default.
Persona [idPersona=1, nombre=Juan, apePaterno=Perez, apeMaterno=null, email=jperez@mail.com, telefono=56677889]
Persona [idPersona=2, nombre=Oscar, apePaterno=Gomez, apeMaterno=null, email=ogomez@mail.com, telefono=55780109]
Persona [idPersona=22, nombre=Angelica, apePaterno=Lara, apeMaterno=null, email=alara@mail.com, telefono=1314145519]
Persona [idPersona=36, nombre=Ricardo, apePaterno=Gonzalez, apeMaterno=null, email=rgonzalez@mail.com, telefono=null]

Fin llamada al EJB desde el cliente
```

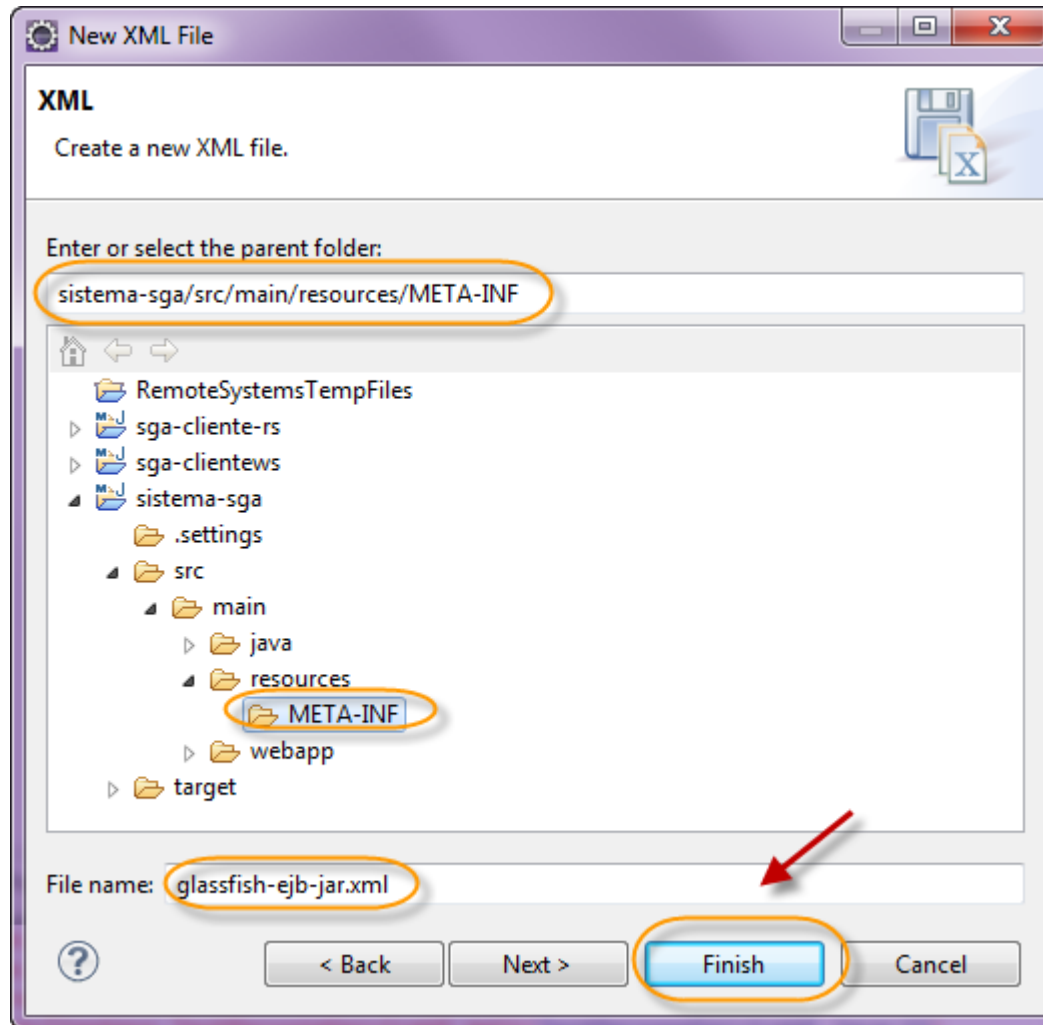
Paso 6. Modificación del Cliente SOAP WS

Para que el cliente SOAP pueda funcionar, es necesario indicar a GlassFish el tipo de autenticación que el SOAP Web Services desea ejecutar, para ello creamos el archivo de configuración `glassfish-ejb-jar.xml`, en la carpeta de META-INF del proyecto sistema-sga:



Paso 6. Modificación del Cliente SOAP WS (cont)

Creamos el archivo `glassfish-ejb-jar.xml`





Paso 6. Modificación del Cliente SOAP WS (cont)

Agregamos el siguiente contenido al archivo glassfish-ejb-jar.xml:

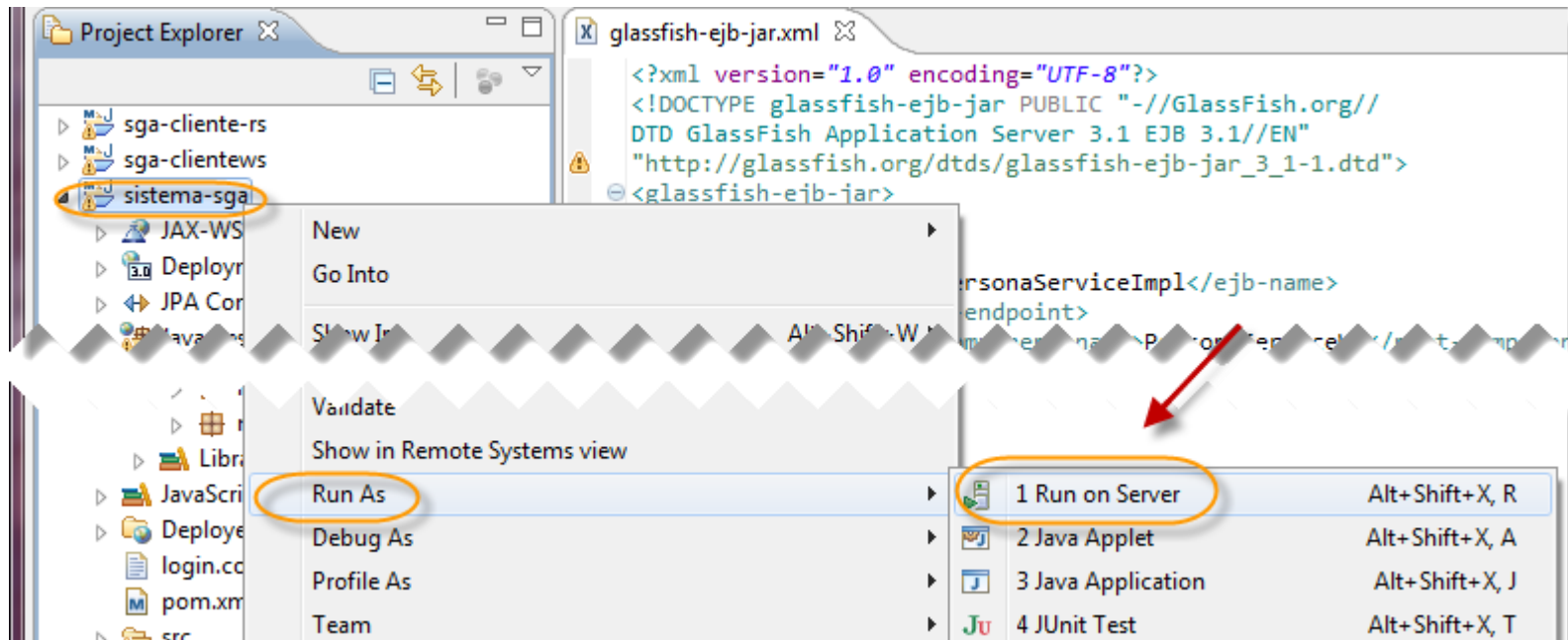
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-ejb-jar PUBLIC "-//GlassFish.org//
DTD GlassFish Application Server 3.1 EJB 3.1//EN"
"http://glassfish.org/dtds/glassfish-ejb-jar_3_1-1.dtd">
<glassfish-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>PersonaServiceImpl</ejb-name>
      <webservice-endpoint>
        <port-component-name>PersonaServiceImplService</port-component-name>
        <login-config>
          <auth-method>BASIC</auth-method>
          <realm>file</realm>
        </login-config>
      </webservice-endpoint>
    </ejb>
  </enterprise-beans>
</glassfish-ejb-jar>
```

Referencia de GlassFish para la configuración del archivo glassfish-ejb-jar.xml:

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

Paso 6. Modificación del Cliente SOAP WS (cont)

Desplegamos nuevamente la aplicación sistema-sga sobre GlassFish:



Paso 6. Modificación del Cliente SOAP WS (cont)

Con el servidor de GlassFish iniciado, abrimos el proyecto sga-clientews y modificamos la clase TestPersonaServiceWS.java, agregando el siguiente código:

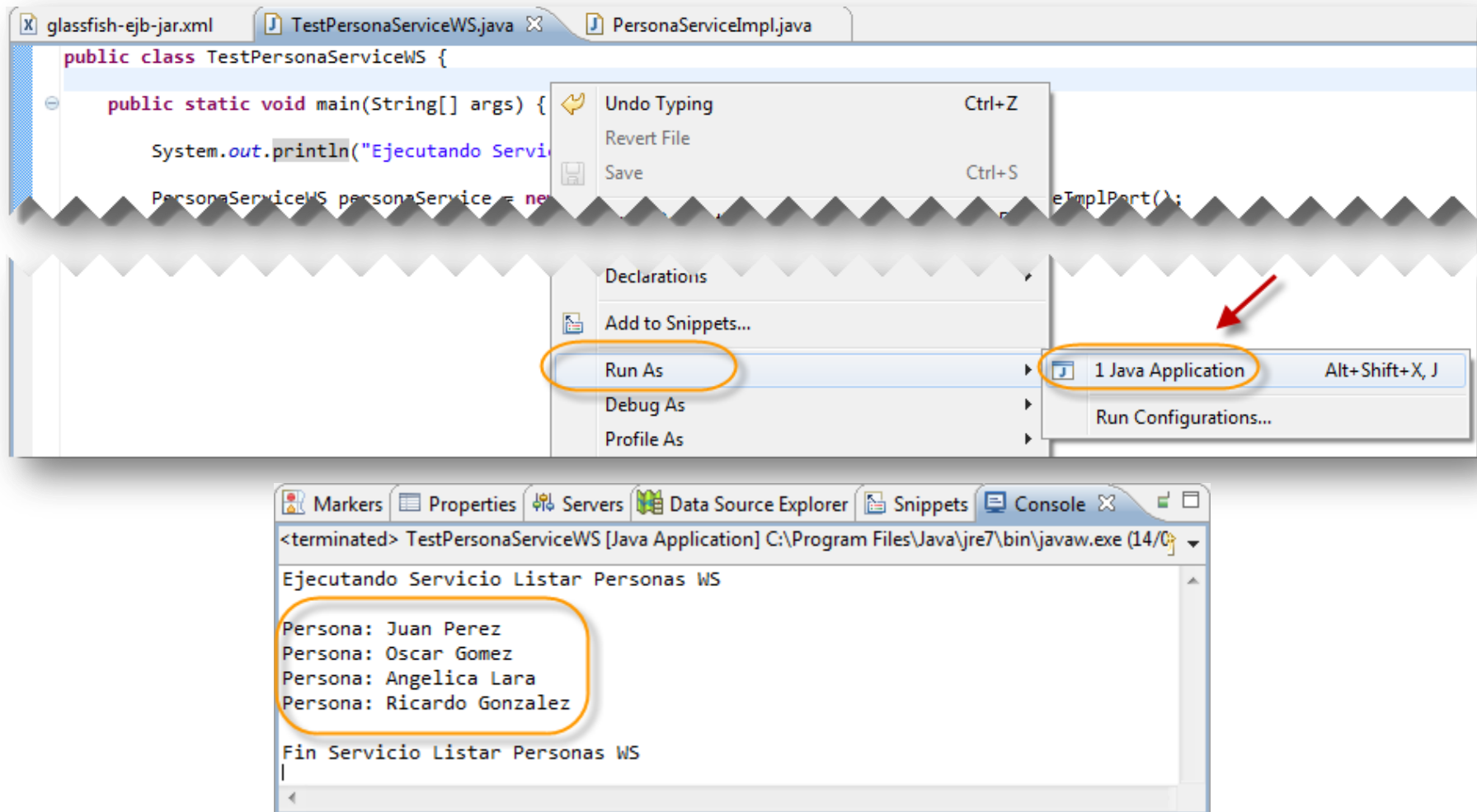
```
((BindingProvider)personaService).getRequestContext().put(BindingProvider.USERNAME_PROPERTY, "admin");  
((BindingProvider)personaService).getRequestContext().put(BindingProvider.PASSWORD_PROPERTY, "admin");
```

El resultado debe ser similar a la siguiente figura:

```
public class TestPersonaServiceWS {  
    public static void main(String[] args) {  
        System.out.println("Ejecutando Servicio Listar Personas WS");  
        PersonaServiceWS personaService = new PersonaServiceImplService().getPersonaServiceImplPort();  
        ((BindingProvider)personaService).getRequestContext().put(BindingProvider.USERNAME_PROPERTY, "admin");  
        ((BindingProvider)personaService).getRequestContext().put(BindingProvider.PASSWORD_PROPERTY, "admin");  
        List<Persona> personas = personaService.listarPersonas();  
        for (Persona persona : personas) {  
            System.out.println("Persona: " + persona.getNombre() + " " + persona.getApePaterno());  
        }  
        System.out.println("Fin Servicio Listar Personas WS");  
    }  
}
```

Paso 6. Modificación del Cliente SOAP WS (cont)

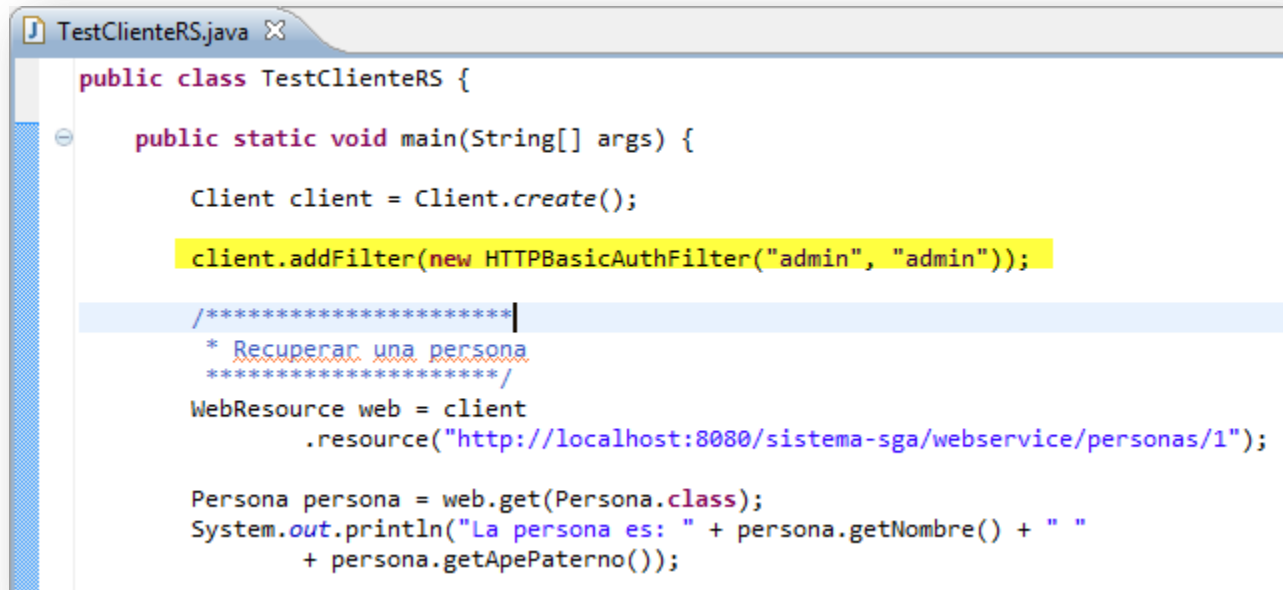
Ejecutamos la clase TestPersonaServiceWS.java y obtenemos la siguiente salida:



Paso 7. Modificación del Cliente REST WS

A continuación, vamos a agregar la seguridad al cliente REST WS. Para ello abrimos el proyecto sga-cliente-rs y modificamos la clase TestClienteRS.java vamos a agregar la siguiente línea de código después de la creación del cliente:

```
client.addFilter(new HTTPBasicAuthFilter("admin", "admin"));
```



```
public class TestClienteRS {  
    public static void main(String[] args) {  
        Client client = Client.create();  
        client.addFilter(new HTTPBasicAuthFilter("admin", "admin"));  
        /*****  
        * Recuperar una persona  
        *****/  
        WebResource web = client  
            .resource("http://localhost:8080/sistema-sga/webservice/personas/1");  
  
        Persona persona = web.get(Persona.class);  
        System.out.println("La persona es: " + persona.getNombre() + " "  
            + persona.getApePaterno());  
    }  
}
```

Paso 7. Modificación del Cliente REST WS (cont)

A continuación, vamos a agregar la seguridad al cliente REST WS. Para ello vamos a agregar la siguiente línea de código después de la creación del cliente:

The screenshot shows an IDE with the following components:

- TestClienteRS.java**:

```
package test;  
  
import java.util.List;  
  
public class TestClienteRS {  
  
    public static void main(String[] args) {  
  
        Client client = Client.create();  
  
        client.addFilter(new HTTPBasicAuthFilter("admin", "admin"));  
  
        // Recuperar una persona  
        // ...  
    }  
}
```
- Run As** context menu: The "Run As" option is circled in orange. The submenu is open, showing "1 Java Application" (also circled in orange) with the shortcut "Alt+Shift+X, J". A red arrow points to this option.
- Console**: The output of the application is shown. A red arrow points to the first line of output.

```
<terminated> TestClienteRS (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (14/08/2014)  
La persona es: Juan Perez  
El código de respuesta en la inserción fue: 500  
El código de respuesta de la modificación fue: 200  
Nueva persona: 1 Juan  
  
La persona a eliminar NO existe  
1 Juan  
2 Oscar  
22 Angelica  
30 Ricardo
```



Paso 8. Modificación del Cliente WEB

Abrimos el archivo index.xhtml y agregamos reemplazamos el contenido del formulario h:form por el siguiente:

```
Bienvenido: <h:outputText value="#{p:userPrincipal()}" />
```

```
<h:panelGrid columns="1" rendered="#{p:ifGranted('ROLE_ADMIN') || p:ifGranted('ROLE_USER')}">  
  <h:commandButton value="Listar Personas" action="listarPersonas" />  
</h:panelGrid>
```

Al ejecutar la aplicación podremos observar el nombre del usuario que se autenticó, y si el usuario posee el rol de ROLE_ADMIN o ROLE_USER podrá visualizar el botón de Listar Personas, en caso contrario este botón no se desplegará.

Si probamos con el usuario guest, no visualizaremos el botón de listarPersonas, y aunque lo pudiéramos ver o ejecutar directamente el URL

<http://localhost:8080/sistema-sga/faces/listarPersonas.xhtml> no podríamos ejecutar ningún método del EJB debido a que tampoco tiene permisos para ello.



Conclusión

- ✓ Con este ejercicio hemos visto cómo las anotaciones `@DeclareRoles` y `@RolesAllowed` al agregarlas a nuestra capa de Servicio, en automático agrega seguridad a todo nuestro sistema. De allí la importancia que la lógica de negocio se encuentre separada de las demás capas.
- ✓ Al agregar estas anotaciones al EJB y según se configure, en automático los clientes que utilizan los métodos de negocio dejan de funcionar, ya que necesitan enviar sus credenciales (usuario y password) para poder ejecutar la funcionalidad de la capa de negocio.
- ✓ En este ejercicio, modificamos cada uno de los clientes con el objetivo de obligarlos a autenticarse y verificar si están o no autorizados para ejecutar la funcionalidad solicitada del sistema. Tanto el cliente Web (JSF), EJB, SOAP y REST se adecuaron para ello.
- ✓ Utilizamos el tipo de seguridad más simple, y aunque existen otros tipos de seguridad más avanzados, este ejercicio establece el proceso completo necesario para agregar seguridad a nuestras aplicaciones Java EE.



www.globalmentoring.com.mx

Pasión por la tecnología Java

Experiencia y Conocimiento para tu vida