Sistemas Orientados a Servicios
Graduado en Ingeniería Informática

Práctica: Definición e implementación de un servicio web JAVA

La práctica consiste en implementar un servicio web, *UPMSocialReading*, que simula gestión sencilla de una aplicación de una red social destinada a lecturas de libros. Un administrador de la red social de lectura (*admin*) podrá añadir y borrar usuarios. Los usuarios de la red pueden realizar operaciones como añadir/eliminar amigos y añadir lecturas, así como consultar las lecturas realizadas. Para acceder a estas operaciones los usuarios deberán ser dados de alta en la red (*addUser*) e iniciar sesión (*login*). El usuario deberá cerrar sesión (*logout*) cuando no acceda y se podrá dar de baja (*quit*). Estas operaciones de gestión de usuarios (*addUser, login, quit*) deben ser redirigidas a un servicio web que está disponible, *UPMAuthenticationAuthorization* que actuará como servicio de autenticación central del sistema.

Las operaciones en Java del servicio web *UPMSocialReading*, generadas desde el WSDL son:

1. **AddUserResponse addUser (Username username)**

Esta operación añade un usuario a la red de lectura. Solo el usuario *admin* puede ejecutar esta operación. Esta operación debe usar la operación *addUser* del servicio *UPMAuthenticationAuthorization*. El parámetro *username* tiene el nombre (*name*). La respuesta (*Response*) es *true* si la operación tiene éxito. La operación devuelve por un lado la contraseña que se ha generado en el servicio *UPMAuthenticationAuthorization* y por otro lado un booleano con un resultado que será true si todo ha ido correctamente y *false* si el *admin* intenta añadir un usuario con un *name* ya registrado o si un usuario distinto del *admin* llama a esta operación.

2. **Response login (User user)**

Cada llamada a esta operación comienza una nueva sesión para un usuario (*user*). El parámetro *user* tiene dos elementos: nombre de usuario (*name*) y contraseña (*pwd*). La respuesta (*Response***)** es un *booleano.* El valor *true* se devuelve si la operación de *login* tiene éxito, es decir la llamada a la operación *login* del servicio *UPMAuthenticationAuthorization* tiene éxito. En caso contrario se devuelve *false*.

Si se llama repetidas veces a *login* en una sesión activa con el mismo usuario ya autenticado, el método devuelve *true* independientemente de la contraseña utilizada. La sesión del usuario en curso sigue activa.

Si se llama a *login* en una sesión activa del usuario *U1* con un usuario *U2*, distinto del usuario autenticado (*U2* distinto de *U1*) el método devuelve *false,* independientemente de la contraseña utilizada. El usuario *U2* no va a tener sesión valida, por lo tanto, no podrá acceder a ningún otro método. La sesión del usuario en curso (*U1*) sigue activa.

Un mismo usuario puede tener varias sesiones activas concurrentemente. Si ese usuario decide cerrar una de sus sesiones solo se cerrará la sesión elegida dejando activas todas las demás.

El login del usuario admin no se debe gestionar a través del servicio *UPMAuthenticationAuthorization.*

3. **void logout()**

Esta operación cierra la sesión del usuario que la invoca. Si esta operación es llamada sin que el usuario haya iniciado sesión (*login* correcto) la llamada no hace nada.

Si un usuario tiene varias sesiones abiertas (ha hecho varias llamadas a la operación *login* con éxito), una llamada a la operación *logout* solo cerrará una sesión. Para cerrar todas las sesiones deberá llamar a la operación *logout* tantas veces como sesiones hay abiertas desde el correspondiente programa/navegador.

4. **Response removeUser (Username username)**

Esta operación elimina al usuario invocando la operación *removeUser* del servicio *UPMAuthenticationAuthorization*. Solo el admin y el propio usuario que se quiere eliminar puede llamar esta operación habiéndose autenticado (*login*) previamente. La respuesta (*Response*) contiene un *booleano*. El valor *true* se devuelve si la operación elimina al usuario. En caso contrario se devuelve *false* (Si si el usuario no tiene permiso o si el admin se intenta eliminar).

Se puede eliminar a usuarios con sesiones activas, al eliminarse el usuario con una sesión activa, cualquier petición posterior deberá retornar como si el usuario no existiera. El *admin* no puede auto-eliminarse.

5. **Response changePassword (PasswordPair password)**

Esta operación accede al método *changePassword* del servicio *UPMAuthenticationAuthorization* para cambiar la contraseña del usuario. El parámetro *password* incluye la contraseña actual (*oldpwd*) y la nueva (*newpwd*). Si hay varias sesiones abiertas de un usuario que llaman a esta operación, solo quedará registrado el último cambio. La respuesta (*ChangePasswordResponse*) tiene un campo *result* de tipo *boolean* que es *true* si la operación remota tiene éxito. La operación

devuelve *false* si se intenta acceder sin haber hecho previamente *login* con éxito o si la contraseña *oldpwd* es incorrecta.

El usuario admin puede cambiar la contraseña, si bien esta gestión no se debe realizar a través del servicio *UPMAuthenticationAuthorization.*

6. **Response addFriend (Username username)**

Esta operación añade un amigo al usuario que la invoca. El parámetro *Username* tiene un campo, *name* (*String*) con el usuario del amigo a añadir. No se almacenarán amigos repetidos. La operación devuelve un objeto de tipo *Response* que incluye un *boolean* (*result*) que es *true* si se añade a un amigo que esté registrado en la red de lectura. Devuelve *false* si el usuario no ha hecho *login* con éxito o si el amigo a añadir no existe en la red social. Esta relación de amistad no tiene que porqué ser recíproca, es decir, un usuario U1 puede ser amigo de U2 pero eso no implica que U2 sea amigo de U1.

7. **Response removeFriend (Username username)**

Esta operación elimina un amigo de la lista de amigos del usuario que la invoca. De manera análoga al método anterior, el parámetro *Username* tiene un campo, *name* (*String*) con el usuario del amigo a eliminar. La operación devuelve un objeto de tipo *Response* que incluye un *boolean* (*result*) que es *true* si se elimina correctamente a un amigo. Devuelve *false* si el usuario no ha hecho *login* con éxito o si el amigo a eliminar no existe en la red social o no aparece en la lista de amigos.

8. **FriendList getMyFriendList()**

Esta operación devuelve la lista de amigos del usuario. La operación devuelve un objeto *FriendList* con un *boolean* (result) que es *true* si el usuario que llama a la operación ha hecho *login* previo con éxito y un *array* con los nombres de usuario de los amigos (friends).

9. **Response addReading (Book book)**

Esta operación permite añadir la lectura de un libro (Book) al usuario que llama a la operación. El parámetro book tiene un campo title y un campo author de tipo String que contiene el título y autor del libro, además de un campo calification de tipo entero, que contiene la calificación que asigna el usuario a ese libro. La operación devuelve Response con un campo boolean que será true si el usuario llama este método después de haber hecho login con éxito, en caso contrario devuelve false. Llamada repetidas al mismo título de libro sirven para actualizar la calificación o el autor y devolverán true en caso de éxito.

10. **TitleList getMyReadings()**

Esta operación devuelve la lista de libros léidos del usuario que invoca la operación. La operación devuelve un objeto *TitleList* con un *boolean* (result) que es *true* si el usuario que llama a la operación ha hecho *login* previo con éxito y un *array* con los títulos de los libros leídos (titles) en orden de lectura, primero los últimos libros leídos. En caso de no haber leído ningún libro deberá devolver true y la lista vacía.

11. TitleList getMyFriendReadings(Username username)

Esta operación devuelve la lista de libros léidos por el amigo del usuario que invoca la operación. La operación devuelve un objeto TitleList con un boolean (result) que es true si el usuario que llama a la operación ha hecho login previo con éxito y es amigo del usuario pasado por parámetro (username). Además, devuelve un array con los títulos de los libros leídos (titles) en orden de lectura, primero los últimos libros leídos. La operación devuelve false si el usuario no ha hecho login previo o no es amigo del usuario. En caso de no haber leído ningún libro deberá devolver true y la lista vacía.

Para realizar la práctica se emplearán las herramientas de Axis2 para Java y se deberá desplegar el servicio en Tomcat. El WSDL que define este servicio se encuentra disponible en la siguiente dirección:

http://porter.dia.fi.upm.es:8080/practica2021/UPMSocialReading.wsdl

El WSDL del servicio ***UPMAuthenticationAuthorization*** se encuentra disponible y funcionando en: http://porter.dia.fi.upm.es:8080/axis2/services/UPMAuthenticationAuthorizationWSSkeleton?wsdl2

y tiene las siguientes operaciones:

1. **LoginResponseBackEnd login(LoginBackEnd login)**

El parámetro *login* tiene dos elementos: nombre de usuario (*name*) y contraseña (*password*). La respuesta (*LoginResponseBackEnd*) es un booleano. Esta operación comprueba que el usuario existe y tiene la contraseña suministrada. El valor *true* se devuelve si la operación de *login* tiene éxito. En caso contrario se devuelve *false*.

2. **AddUserResponseBackEnd addUser(UserBackEnd user)**

Esta operación añade un usuario al sistema. El parámetro *user* tiene el nombre de usuario (*name*) del usuario a añadir. La respuesta (*AddUserResponseBackEnd*) tiene un campo *result* de tipo *boolean* que es *true* si la operación tiene éxito y un campo *password* de tipo *String* con la

contraseña autogenerada para el nuevo usuario. La operación devuelve *false* si se intenta añadir un usuario con un nombre de usuario ya registrado, en este caso el campo *password* está vacío.

3. **RemoveUserResponse removeUser(RemoveUser removeUser)**

Esta operación borra un usuario del sistema. El parámetro *removeUser* tiene un campo de tipo *String* con el nombre de usuario (*name*). La respuesta (*RemoveUserResponse*) tiene un campo *result* de tipo *boolean* que es *true* si la operación tiene éxito (el nombre de usario y la contraseña son correctos). La operación devuelve *false* si se intenta borrar un usuario con un *name* que no existe.

4. **ChangePasswordResponseBackEnd changePassword(ChangePasswordBackend changePassword)**

Esta operación permite que un usuario ya registrado pueda cambiar su contraseña. El parámetro *changePassword* incluye el nombre del usuario (*name*), la contraseña actual (*oldpwd*) y la nueva (*newpwd*). La respuesta (*ChangePasswordResponse*) tiene un campo *result* de tipo *boolean* que es *true* si la operación tiene éxito, es decir, la contraseña actual coincide con la contraseña actual del usuario y se ha realizado el cambio de contraseña. La operación devuelve *false* en caso contrario.

5. **ExistUserResponse existUser(Username username)**

Esta operación permite averiguar si un usuario está registrado en el sistema. El parámetro *username* incluye un campo *String* (*name*) con el nombre del usuario a buscar. La respuesta (*ExistUserResponse*) tiene un campo *result* de tipo *boolean* que es *true* si el usuario existe en el sistema.

Requisitos del servicio web *UPMSocialReading*

1. En el momento del despliegue del servicio UPMSocialReading, éste debe tener al usuario *admin* con *username* ***admin*** y contraseña ***admin***. Solo puede haber un *admin* en el sistema y éste debe ser gestionado en el propio *UPMSocialReading* y no en el servicio de *UPMAuthenticationAuthorization*.
2. La información de los usuarios (*username*, *password*) se gestiona en el servicio *UPMAuthenticationAuthorization* pero podéis almacenarla en memoria.
3. La información de las lecturas y amigos de un usuario debe ser almacenada en el servicio *UPMSocialReading* (en memoria)
4. Se deberán hacer pruebas con distintos clientes conectados al mismo tiempo.
5. El estado del servicio tiene que persistir en memoria a las sesiones de los clientes. Si se añade un amigo y se cierra la sesión, cuando el usuario vuelva a acceder, al consultar la lista de amigos, se le devolverá una lista que contenga el amigo añadido.

Se pide:

- Implementar el servicio web UPMSocialReading en Java empleando Axis2.
- Programar un cliente que acceda al servicio web que pruebe el servicio desarrollado.
- Hacer pruebas con distintos clientes.

Instrucciones para la entrega de la práctica:

FECHA DE ENTREGA: 30-05-2021 hasta las 23:55.

La práctica debe realizarse por parejas. Solo se entregará una práctica por pareja a través de Moodle.

SE COMPRUEBAN COPIAS Y PLAGIOS

NO UTILIZAR REPOSITORIOS PÚBLICOS DE GITHUB

Todas las parejas deberán subir a Moodle el fichero comprimido (.tar.gz, .rar, .zip, .7z) con una carpeta llamada appellido1appellido2 con el siguiente contenido:

– **Una carpeta llamada "servicio**" con todo el código fuente del servicio, la carpeta resources y el build.xml. El formato de la carpeta tiene que estar listo para que ejecutando el comando "ant" dentro de la carpeta "servicio" se cree el fichero con el servicio (extensión .aar).

– **Una carpeta llamada "cliente"** con todo el código fuente del cliente, la carpeta resources y el build.xml.

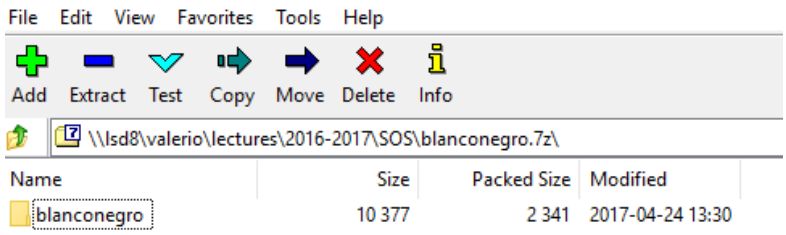
– Una copia de la clase "skeleton" con la implementación del servicio.

– El fichero de despliegue .aar para desplegar el servicio en Tomcat.

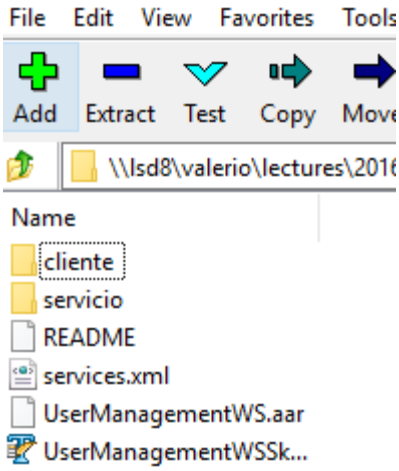
– Un README con los datos de la pareja: Nombre Apellidos y número de matricula de cada uno.

Ejemplo de archivo de entrega

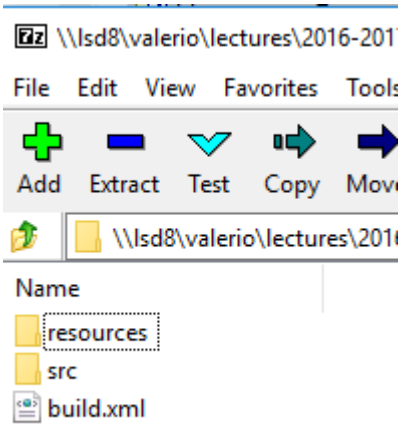
- Pareja: Juan Blanco y Paco Negro Raíz del fichero rar:



- Explorando la carpeta "blanconegro" hay:



- Explorando la carpeta “servicio” hay:



El nombre completo de la clase *skeleton* debe ser:

es.upm.fi.sos.UPMSocialReadingSkeleton.java

Los alumnos pueden descargarse una máquina virtual con el mismo entorno que se utilizará para la corrección de la práctica. Ésta se encuentra disponible en Moodle

Para aprobar la práctica, ésta deberá funcionar bien con el software incluido en la máquina virtual y descrito en la guía de instalación de herramientas (con JDK versión 1.7 y axis2 versión 1.6.2).