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, *UPMBank*, que simula gestión sencilla de un banco. Un administrador del banco (*superuser*) podrá añadir y borrar clientes, los clientes pueden realizar ingresos y retiradas de dinero, así como consultar los movimientos. Para acceder a estas operaciones los usuarios deberán darse de alta banco (*addUser*) e iniciar sesión (*login*). El usuario deberá cerrar sesión (*logout*) cuando no acceda y se podrá dar de baja del banco (*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 *UPMBank*, generadas desde el fichero WSDL son:

1. **AddUserResponse addUser (User user)**

Esta operación añade un usuario al sistema. Solo el usuario *superuser* puede ejecutar esta operación. Esta operación debe usar la operación *addUser* del servicio *UPMAuthenticationAuthorization*. El parámetro *user* 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 *superuser* intenta añadir un usuario con un *username* ya registrado o si un usuario distinto del *superuser* llama a esta operación.

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

Cada llamada a esta operación comienza una nueva sesión para un usuario (*user*). El parámetro *login* tiene dos elementos: nombre de usuario (*name*) y contraseña (*password*). 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.

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 puede borrarse un usuario si no tiene cuentas en el banco. Solo el administrador 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 se intenta eliminar un usuario con cuentas abiertas o si el usuario no tiene permiso).

El usuario *superuser* puede eliminar a usuarios con sesiones activas. El *superuser* 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.

6. **BankAccountResponse addBankAcc (Deposit quantity)**

Esta operación crea una cuenta bancaria para el usuario que la invoca. El parámetro *quantity* tiene un campo *double* con la cantidad inicial con la que se crea la cuenta. La operación devuelve *un BankAccountResponse* con un campo *boolean* que será *true* si la operación se ha realizado correctamente, y un campo IBAN (*String*) con el número de cuenta creado. Devuelve *false* si el usuario no ha hecho *login* con éxito.

7. **Response closeBankAcc (BankAccount account)**

Esta operación permite al usuario que la invoca cerrar la cuenta bancaria si el saldo es 0. El parámetro tiene un campo IBAN, con el número de cuenta que desea cerrar. La operación devuelve *true* si ha tenido éxito, y *false* en caso contrario. Es decir, si no ha hecho *login* previo, no existía esa cuenta o tenía saldo.

8. **AddMovementResponse addIncome (Movement movement)**

Esta operación permite al usuario ingresar dinero en una de sus cuentas bancaria. El parámetro tiene un campo IBAN con el número de cuenta en el que se desea ingresar dinero y un campo quantity con la cantidad a ingresar. La operación devuelve un objeto *AddMovementResponse* que tiene un campo booleano, *result*, que estará a *true* si ha tenido éxito, *false* en caso contrario. Es decir, si no ha hecho *login* previo o no existe esa cuenta. También tendrá un campo balance (*double*) con el saldo total de la cuenta tras realizar el ingreso.

9. **AddMovementResponse addWithdrawal (Movement movement)**

Esta operación permite al usuario retirar dinero de una de sus cuentas bancaria. El parámetro tiene un campo IBAN con el número de cuenta del que se desea retirar dinero y la cantidad. La operación devuelve un objeto *AddMovementResponse* que tiene un campo booleano *result* que estará a *true* si ha tenido éxito, *false* en caso contrario. Es decir, si no ha hecho *login* previo, no había saldo suficiente o no existe esa cuenta. También tendrá un campo balance (*double*) con el saldo total de la cuenta tras retirar el dinero.

10. **MovementList getMyMovements ()**

Esta operación devuelve la lista con los últimos movimientos que ha realizado el usuario que invoca la operación en todas sus cuentas. La operación devuelve un objeto *MovementList* 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 las cantidades de los últimos 10 movimientos de todas sus cuentas (retiradas e ingresos) en orden de realización, primero los últimos realizados.

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/practica1920/UPMBank.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 dos campos de tipo *String* con el nombre de usuario (*name*) y contraseña del usuario a eliminar. 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 o si no coincide la contraseña.

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 *UPMBank*

1. En el momento del despliegue del servicio, éste debe tener al usuario *superuser* con *username* ***admin*** y contraseña ***admin***. Solo puede haber un *superuser* en el sistema y éste puede cambiar su contraseña utilizando la operación *changePassword*.
2. La información de los usuarios (*username*, *password*) es almacenada en el servicio *UPMAuthenticationAuthorization*.
3. La información de las cuentas y transferencias por usuario debe ser almacenada en el servicio *UPMBank* (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 ingreso y se cierra la sesión, cuando el usuario vuelva a acceder, al consultar la lista de movimientos, se le devolverá una lista que contenga el ingreso añadido.

Se pide:

- Implementar el servicio web *UPMBank* 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: 07-06-2020 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.

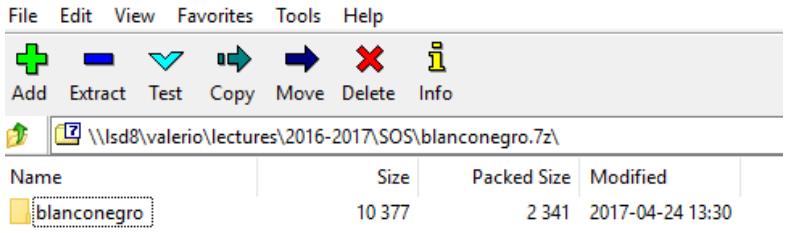
– Una copia de la clase "skeleton" con la implementación del servicio.

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

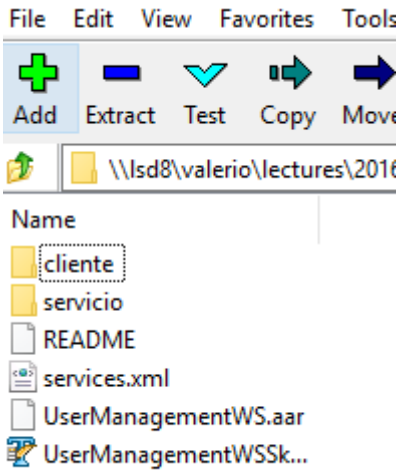
– Un README con los datos de la pareja: Nombre Apellidos y número de matricula de cada uno.

Ejemplo de archivo de entrega

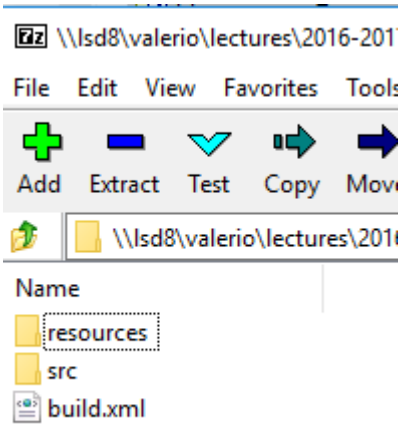
- Pareja: Juan Blanco y Paco Negro Raíz del fichero rar:



- Explorando la carpeta "blanconegro" hay:



- Explorando la carpeta “servicio” hay:



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

es.upm.fi.sos.upmbank.UPMBankWSSkeleton.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).