Sistemas Orientados a Servicios
Graduado en Ingeniería Informática

Práctica: Definición e implementación de un servicio web en Java

La práctica consiste en implementar un servicio web, *Travel Agency*, que simula una agencia de viajes empleando las herramientas de Axis2 para Java y desplegando el servicio en Tomcat. El WSDL que define la interfaz de este servicio se encuentra disponible en la siguiente dirección:

http://adapt08.ls.fi.upm.es:8080/static/TravelAgency.wsdl

El servicio web (SW) *Travel Agency* accede a tres servicios auxiliares: *Login Service , Flight Booking y Hotel Booking*. *Flight Booking y Hotel Booking* proporcionan servicios para buscar, reservar y cancelar asientos en aviones y habitaciones en hoteles, respectivamente. El servicio *Login Service* proporciona un servicio de autenticación de los usuarios. En concreto estos servicios tienen las siguientes operaciones:

- *Flight Booking:*
  - **getOriginList**: devuelve la lista de ciudades desde las que salen vuelos.
  - **getDestinationList**: dada una ciudad origen devuelve la lista de ciudades de destino.
  - **checkFlight:** dada una cuidad de origen y una de destino esta operación devuelve el número de asientos disponibles en este avión y el precio de cada asiento.
  - **bookFlight:** dada una cuidad de origen, una de destino y el número de asientos deseado esta operación devuelve un *boolean* con el resultado de la reserva y el precio total.
  - **cancelFlight:** dada una cuidad de origen, una de destino, y el número de asientos a cancelar este método devuelve un *boolean* con el resultado de la cancelación y el importe a devolver.
- *Hotel Booking:*
  - **getCityList**: devuelve la lista de ciudades donde hay hoteles.
  - **getHotelList**: dada una ciudad devuelve la lista de hoteles en esa ciudad.

- **checkHotel:** dada una ciudad y un hotel, esta operación devuelve el número de habitaciones disponibles es ese hotel y el precio de cada habitación.
- **bookHotel:** dada una cuidad, un hotel y el número de habitaciones deseado esta operación devuelve un *boolean* con el resultado de la reserva y el precio a pagar.
- **cancelHotel:** dada una cuidad, un hotel y el número de habitaciones a cancelar este método devuelve un *boolean* con el resultado de la cancelación y el importe a devolver.

- **Login Service:**
  - **authenticateUser:** recibe un nombre de usuario y contraseña. Devuelve un cierto si la autenticación ha tenido éxito.

Los métodos de los tres SWs devuelven varios tipos de excepciones como se puede apreciar en sus WSDLs.

Flight Booking

WSDL: http://adapt08.ls.fi.upm.es:8080/static/FlightBooking.wsdl

Despliegue: http://adapt08.ls.fi.upm.es:8080/axis2/services/FlightBookingWS

Hotel Booking

WSDL: http://adapt08.ls.fi.upm.es:8080/static/HotelBooking.wsdl

Despliegue: http://adapt08.ls.fi.upm.es:8080/axis2/services/HotelBookingWS

Login Service

WSDL: http://adapt08.ls.fi.upm.es:8080/static/LoginService.wsdl

Despliegue: http://adapt08.ls.fi.upm.es:8080/axis2/services/LoginServiceWS

Requisitos del servicio web *Travel Agency*

1. Los servicios *Flight Booking* y *Hotel Booking* manejan sesiones de tipo ***soapsession***.
2. El servicio *Travel Agency* debe ser desplegado con manejo de sesión de tipo **soapsession**
3. Todos los usuarios deben tener un presupuesto inicial de **10.000.**
4. Toda la información relacionada con los usuarios (autenticado/noautenticado, presupuesto) deberá almacenarse en estructuras de datos independientes de la sesión actual de un usuario y por lo tanto deben ser compartidas entre ellos. Por ejemplo, si un usuario hace *login*, compra un viaje (por tanto, se modifica el presupuesto) y hace

logout, la siguiente vez que haga un *login* tendrá que empezar con el presupuesto posterior a su última compra.

5. Las instancias de los clientes utilizadas para conectarse a los servicios *Flight Booking* y *Hotel Booking* deben ser compartidas entre todos los usuarios del servicio web *Travel Agency*. Esto significa que si un primer usuario compra 10 de los 100 asientos disponibles en un avión, el siguiente usuario que se conecte verá como disponible en este mismo avión solo 90 asientos.

A continuación se describe la funcionalidad de cada una de las operaciones del servicio web *Travel Agency:*

1. **LoginResponse login(Login login)**

Esta función debe autenticar los usuarios utilizando el servicio remoto *Login Service*. El objeto *login* contiene dos parámetros: *username* y *password*. El método devuelve el valor *true* si el *login* fue hecho con éxito y *false* en caso contrario. Si el usuario se autentica con éxito, podrá llamar al resto de métodos del servicio usando esa misma sesión. Cada llamada a esta operación comienza una nueva sesión. Para hacer *login* se empleará como usuario el **"**DNI**"** y como contraseña el **"**DNI al revés**"**. Si el servicio de *Login Service* devuelve la excepción *loginerror,* la operación login devolverá la excepción *RemoteServerError*.

Si se llama a cualquier otra operación (salvo *logout*, descrito a continuación) sin haber comenzado una sesión con éxito, la operación elevará la excepción *NotValidSessionError*. Si se llama a este mismo método tras haber hecho *login* con usuario y contraseña correctos, y no se ha llamado a *logout*, entonces la llamada devolverá *true*.

2. **void logout()**

Si esta operación se llama sin que el usuario haya iniciado sesión, la llamada es ignorada. Si el usuario sí inició una sesión, entonces ésta es descartada, a partir de ese momento cualquier llamada a esa sesión devolverá la excepción *NotValidSessionError* hasta que vuelva a hacerse *login* con usuario y contraseña correctos.

3. **Budget getBudget()**

Esta operación devuelve el valor del presupuesto del usuario en el momento de la llamada.

4. **CheckingTripResponse checkTrip(CheckingTrip checkingTrip)**

Esta operación recibe un objeto *checkingTrip* con la cuidad de origen, una cuidad de destino y un hotel. El método devuelve un objeto de la clase *CheckingTripResponse* con el resultado de la búsqueda: número de asientos disponibles en el vuelo que conecta la cuidad origen con el

destino y el precio de cada asiento, número de habitaciones disponibles en el hotel seleccionado en la cuidad de destino y el precio de cada habitación. Si la cuidad origen no existe, si no hay aviones que salen desde la cuidad origen hacia el destino, si en la cuidad destino no hay hoteles, si el hotel seleccionado no existe, si los servicios *Flight Booking* y/o *Hotel Booking* no están disponibles o si el usuario no ha hecho el login, el método devolverá una de las siguientes excepciones: *NotValidOriginFlightError, NotValidDestinationFlightError, NotValidCityHotelError, NotValidHotelHotelError, RemoteServiceError, NotValidSessionError*.

5. **BookingTripResponse bookTrip(BookingTrip bookingTrip)**

Esta operación recibe un objeto *bookingTrip* con la cuidad de origen, la cuidad de destino, el hotel seleccionado, el número de asientos en el vuelo y el número de habitaciones en el hotel. La operación devuelve un objeto de la clase *BookingTripResponse* con un booleano indicando si la reserva tuvo éxito. En tal caso, se habrá actualizado el presupuesto del usuario. A las excepciones definidas para la operación *checkTrip* se añaden las siguientes: se intenta reservar un número de asientos no disponibles (*NotEnoughSeatsFlightError*) o el número de asientos es negativo (*NotValidSeatFlightError*), se intenta reservar un número de habitaciones no disponibles (*NotEnoughRoomsHotelError*) o el número de habitaciones es negativo (*NotValidRoomHotelError*). Finalmente, si el usuario no tiene el suficiente presupuesto para finalizar la compra del viaje, se devolverá la excepción *NotEnoughBudgetError.*

Las siguientes operaciones hacen de interfaz hacia los métodos del servicio web remoto *Flight Booking* comprobando siempre que el usuario haya hecho el *login* con éxito y que tenga el presupuesto necesario para finalizar la compra de los asientos deseados en un avión. Estas operaciones pueden lanzar las siguientes excepciones: *NotValidOriginFlightError, NotValidDestinationFlightError, NotEnoughSeatsFlightError, NotValidSeatFlightError, RemoteServiceError, NotValidSessionError, NotEnoughBudgetError.*

OriginFlightList getOriginFlightList()

DestinationFlightList getDestinationFlightList(OriginFlight originFlight)

CheckingOnlyFlightResponse checkOnlyFlight(CheckingOnlyFlight checkingOnlyFlight)

BookingOnlyFlightResponse bookOnlyFlight(BookingOnlyFlight bookingOnlyFlight)

CancellingOnlyFlightResponse cancelOnlyFlight(CancellingOnlyFlight cancellingOnlyFlight)

Las siguientes operaciones hacen de interfaz de los métodos del servicio web *Hotel Booking* comprobando siempre que el usuario haya hecho *login* con éxito y que tenga el presupuesto necesario para finalizar la compra de las habitaciones deseadas en un hotel, actualizando el presupuesto del usuario. Estas operaciones pueden lanzar las siguientes excepciones:

NotValidCityHotelError, NotValidHotelHotelError, NotEnoughRoomsHotelError, NotValidRoomHotelError, RemoteServiceError, NotValidSessionError, NotEnoughBudgetError.

CityHotelList getCityHotelList()

HotelHotelList getHotelHotelList(CityHotel cityHotel)

CheckingOnlyHotelResponse checkOnlyHotel(CheckingOnlyHotel checkingOnlyHotel)

BookingOnlyHotelResponse bookOnlyHotel(BookingOnlyHotel bookingOnlyHotel)

CancellingOnlyHotelResponse cancelOnlyHotel(CancellingOnlyHotel cancellingOnlyHotel)

Se debe programar un cliente que acceda al servicio *Travel Agency*; este cliente deberá, al menos:

1) Llamar a la operación **login** con un usuario y contraseña incorrectos.

2) Hacer una llamada a la operación **getBudget** y comprobar que se obtiene **NotValidSessionError**.

3) Llamar a la operación **login** con un usuario y contraseña correctos.

4) Llamar a la operación **getOriginFlightList** para obtener la lista de ciudades de origen desde la que salen aviones.

5) Hacer una llamada a la operación **getBudget** para saber cuál es el presupuesto del usuario.

6) Obtener la lista de los destinos de las primeras 3 ciudades de la lista anterior.

7) Comprar 3 habitaciones en un hotel con la operación **bookOnlyHotel**

8) Comprobar que el nuevo presupuesto es correcto utilizando las operaciones **getBudget** y **checkOnlyHotel.**

9) Efectuar la compra de un viaje mediante llamada a la operación **bookTrip**.

10) Comprobar de nuevo que el budget del usuario se ha actualizado correctamente. Para ello se llamará de nuevo a **getBudget** y se comprobará que lo que devuelve es el resultado de sustraer el coste del viaje del presupuesto que tenía el usuario antes de la compra (obtenido en el punto 5).

11) Llamar a la operación **logout**.

12) Llamar a la operación **getBudget** y comprobar que se obtiene **NotValidSessionError.**

13) Efectuar un nuevo **login** con el mismo usuario y comprobar que el presupuesto es el obtenido en el punto 10.

14) Hacer login con los dos usuarios de la pareja.

Hay que tener en cuenta que Axis2 define un *timeout* para las sesiones de tipo *soapsession*. Cuando este *timeout* vence, Axis2 invalida la sesión.
El valor por omisión de este *timeout* es de 30 segundos. Durante el desarrollo de la práctica, antes de lanzar cada prueba con los clientes del servicio *Travel Agency*, se aconseja hacer un nuevo despliegue del servicio *TravelAgency* en vuestra instancia de Tomcat para evitar la invalidación de la sesión con los servicios *FlightBooking* y *HotelBooking*.

Instrucciones para la entrega de la práctica

La práctica debe realizarse por parejas.

Todas las parejas deberán subir a *Moodle* el fichero comprimido (.tar.gz o .rar) **con el siguiente contenido:**

– Una carpeta con nombre src con el código fuente del servicio (travelAgency).
– El fichero de despliege .aar.
– Una carpeta resources con el fichero services.xml.

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: :

http://adapt04.ls.fi.upm.es/VirtualBoxCursoSOS.7z

Para aprobar la práctica, ésta deberá funcionar bien con el software incluido en SOSSwStack.tgz. (http://adapt04.ls.fi.upm.es/SOSSwStack.tgz.), con JDK versión 1.6. Además del cliente descrito, se realizarán otras pruebas del servicio con un cliente automatizado que comprobará la corrección de la misma.