Proyecto AST 2016 - 17

Servicio de vuelos nacionales

Grupo C3-2

Rubén Pérez Vaz
Sara Sánchez Piñeiro
Sara Berezo Loza
Marta Blanco Caamaño

ÍNDICE

Introducción ……... 3

Diagrama de orquestación ……..…….…… 4 - 5

Servicios propios

Aeropuertos ……..…… 6
Vuelos ……..…… 6
Orquestador ……... 7
Banco ……..…... 7

Servicios externos

Servicios SOAP: WEBSERVICESX ……..... 8
Servicios REST: SKYSCANNER ……..… 8

Llamada asíncrona ……... 9

Handler ……... 9

UDDI ……..…… 9

Funcionamiento del sistema global

Inicio y registro de los servicios ……..……. 10
Funcionamiento tras el inicio y el registro de los servicios …….. 10

Modificaciones respecto al diseño inicial ……..... 11

Pruebas ……..…. 12

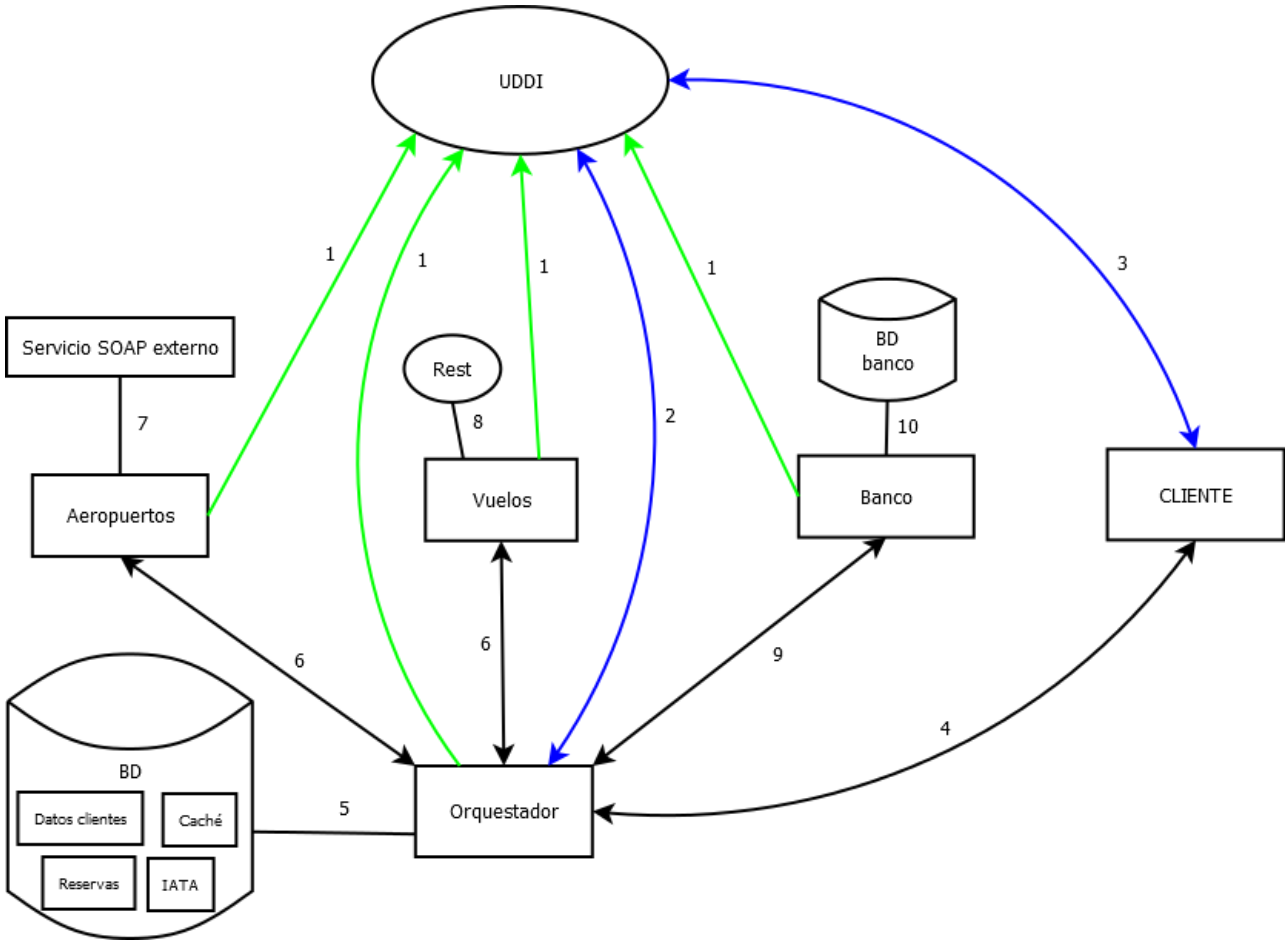
Introducción

Nuestro servicio web ofrecerá al cliente una plataforma para poder consultar vuelos nacionales (España). Cuando un cliente se registre podrá iniciar sesión y hacer una búsqueda de los vuelos nacionales introduciendo la ciudad de origen y de destino de su vuelo así como las fechas en las que desea realizar el viaje. Nuestro sistema, primero analizará si existe o no algún aeropuerto en las localidades seleccionadas y, en caso afirmativo, buscará los vuelos disponibles para las fechas seleccionadas. Este devolverá al cliente una lista que contenga dicha información.

A continuación el cliente tendrá la opción de comprar un billete para alguno de esos vuelos, o simplemente salir y utilizar el servicio a modo de consulta. Por último, si escoge la opción de compra, nuestro servicio le enviará una notificación vía email al cliente cuando se haya hecho efectivo el pago.

La segunda opción principal que ofrece el servicio es la de consultar todos las reservas de vuelos que haya hecho un cliente.

Diagrama de orquestación



1 – Los servicios internos (Aeropuertos, Vuelos y Banco) y el Orquestador se registran en UDDI aportando el nombre de su servicio y su Endpoint.

2 – El orquestador busca los servicio cuando necesite comunicarse con ellos, es decir, cuando necesite hacer uso de ese servicio para una tarea específica, lo buscará. El orquestador buscará según el nombre del servicio que precise, y JUDDI le devolverá el Endpoint del servicio buscado para que el Orquestador pueda comunicarse con él.

3 – Análogamente al paso anterior, cada vez que el cliente desee hacer una operación buscará el servicio Orquestador (según el nombre del servicio), y jUDDI le devolverá su Endpoint correspondiente en caso de que esté registrado.

4 – En este paso, el cliente se comunicará directamente con el Orquestador, el cual le ofrecerá una serie de opciones entre las cuales el cliente podrá elegir (Registrarse o Iniciar sesión). Una vez se inicie la sesión, el usuario podrá buscar, reservar o pagar vuelos.

5 – El orquestador hace uso de la base de datos para cuatro fines diferentes. La primera de ellas es para introducir el registro de un nuevo cliente y sus datos personales para poder

consultarlos posteriormente (o bien para la validación de un inicio de sesión de un cliente previamente registrado, o bien para obtener el email para poder enviar un correo electrónico de confirmación cuando se compre un vuelo). Por otra parte, almacenará los códigos IATA de los aeropuertos de las ciudades que el cliente consulte, para agilizar futuras búsquedas de los mismos destinos. Esta llamada será asíncrona (no bloqueante). La tercera finalidad es almacenar todas las ofertas que un cliente consulte, con el objetivo de consultar el precio cuando el cliente quiera comprar el vuelo. Por último, el Orquestador almacenará para cada cliente todas las reservas que haya efectuado para que el pueda consultarlas a posteriori.

6 – Cuando un cliente quiera consultar un vuelo, el Orquestador buscará a los servicios Aeropuertos y Vuelos. El cliente introducirá una ciudad de origen y una ciudad de destino, además de las fechas de salida y regreso. En primer lugar, el Orquestador consultará en su base de datos local los códigos de los aeropuertos de las ciudades introducidas por el usuario (como ya comentamos en el punto 5). En caso de no hallar ninguna, consultará al servicio interno Aeropuertos por dichos códigos. Una vez obtenidos de cualquiera de las dos formas posibles, el Orquestador enviará los códigos IATA junto con las fechas de salida y regreso del vuelo al servicio Vuelos. Este servicio le devolverá una serie de ofertas y el Orquestador, tras actualizar la caché borrando ofertas anteriores e introduciendo las nuevas en su base de datos local, las devolverá al cliente que las mostrará por pantalla.

7 – El servicio Aeropuertos hace uso de un servicio externo consultandole todos los aeropuertos de un país (Spain en nuestro buscador de vuelos nacionales). Este devuelve los aeropuertos con sus correspondientes códigos IATA y el nombre del aeropuerto (además de otra información que no utilizaremos).

8 – Cuando el servicio Vuelos recibe del Orquestador los datos para consultar un vuelo (ciudad de origen, ciudad de destino, fecha de salida y fecha de regreso), este servicio se comunica con un servicio REST (skyscanner) para obtener ofertas de vuelos en las fechas y lugares de origen y destino especificados por el cliente. La caché del Orquestador se actualizará con estas nuevas ofertas obtenidas del servicio REST.

9 – En caso de que el cliente decida comprar un vuelo, el Orquestador buscará el servicio Banco para simular la compra. Primero consultará a su base de datos el precio de la oferta que desea el cliente así como su email (para enviar al cliente un correo de confirmación tras la compra). En esta comunicación entre Orquestador y Banco se encuentra el handler. El Orquestador inserta en la cabecera una contraseña de seguridad asociada al IBAN bancario (previamente introducido por el usuario). Cuando el banco lo recibe examina esta cabecera y comprueba en su base de datos si la contraseña está asociada a ese IBAN. En caso de no coincidir no lee el resto del mensaje. En caso contrario, si los datos son correctos, la compra se efectuará y la reserva del vuelo se guardará en la base de datos del Orquestador.

10 – Cuando se dé la situación de que la compra se puede realizar, el servicio Banco se comunicará con su base de datos y, tras comprobar que el cliente posee saldo suficiente en su cuenta, actualizará el saldo de las cuentas del cliente y de la compañía después de la compra del vuelo. Así mismo, también hace uso de ella cuando, como hemos explicado en

el paso anterior, necesita comprobar si el IBAN se corresponde con la clave de seguridad del usuario.

Servicios propios

Hemos implementado cuatro servicios: Aeropuertos, Vuelos, Banco y Orquestador. Este último será el encargado comunicarse con los clientes y con el resto de servicios internos para hacer uso de ellos. Los describimos a continuación:

Aeropuertos

- Funcionalidad: El Orquestador le enviará la ciudad de origen y la ciudad de destino. Tras ello, Aeropuertos contactará con el servicio externo SOAP WebServicesX y obtendrá todos los aeropuertos de España. Tras esto, convertiremos la respuesta del servicio web externo SOAP a JSON y parsearemos el mensaje obtenido para obtener los códigos IATA (códigos de los aeropuertos) de la ciudad de salida y la ciudad de regreso. Una vez obtenidos los códigos, se le enviará la respuesta al servicio Orquestador.
- Librerias a mayores
    - json-20160810
    - jackson (convierte XML a JSON)
- Action
    - getInfoAeropuerto

Vuelos:

- Funcionalidad: El Orquestador le enviará el código IATA de la ciudad de origen, el código IATA de la ciudad de destino, la fecha de salida y la fecha de regreso. Tras ello, Vuelos contactará con el servicio REST de Skyscanner (Ejemplo de envío de petición GET: http://partners.api.skyscanner.net/apiservices/browsequotes/v1.0/ES/eur/es-ES/VGO/MAD/2017-05-15/2017-05-16?apikey=prtl6749387986743898559646983194). Una vez recibido la respuesta de Skyscanner, parsearemos la respuesta (JSON) y enviaremos la respuesta de las ofertas encontradas al Orquestador en formato JSON.
- Librerias a mayores
    - Json-20160810
    - jackson (convierte XML a JSON)
- Action
    - getInfoVuelos

Orquestador:

- Funcionalidad: El Orquestador será el encargado de contactar con los servicios web Aeropuertos, Vuelos y Banco y responderá a todas las peticiones del cliente. Además almacenará en una base de datos local todos los clientes registrados, las reservas de vuelos de cada uno de ellos, los códigos IATA de los aeropuertos buscados por cada cliente, así como las ofertas buscadas más recientemente (a modo de caché).
- Librerias a mayores
    - json-20160810
- Action
    - comprobarClienteRegistrado
    - registrarCliente
    - verReservasCliente
    - obtenerOfertas
    - comprarBillete

Banco:

- Funcionalidad: Transfiere un importe de una cuenta origen a una cuenta destino si la cuenta de origen dispone del saldo suficiente. Esto queda registrado en una base de datos local.
- Librerías a mayores:
    - mysql-connector-java-5.1.41-bin.jar
- Action:
    - pagar

Servicios externos

Hacemos uso de dos servicios externos: un servicio SOAP y un servicio REST. El primero establecerá la comunicación con el servicio interno Aeropuertos y el segundo con el servicio interno Vuelos. A continuación pasamos a describirlos:

Servicios SOAP: WEBSERVICESX.

- Endpoint : http://www.webservicex.net/airport.asmx?WSDL
- SOAP Action: GetAirportInformationByCountry
- Funcionalidad: se le inserta el nombre de un país, en este caso Spain y nos devuelve todos los aeropuertos del país introducido.

Servicios REST: SKYSCANNER.

- Endpoint: https://skyscanner.github.io/slate/#browse-quotes
- Con él obtenemos información de vuelos con el origen, destino, fecha de salida y fecha de regreso especificadas por el cliente.
- Ejemplo petición GET:
    - http://partners.api.skyscanner.net/apiservices/browsequotes/v1.0/ES/eur/es-ES/VGO/MAD/2017-05-15/2017-05-16?apikey=prtl6749387986743898559646983194

Llamada asíncrona

Entre Orquestador y Aeropuertos. Cuando un cliente le pide información sobre un vuelo entre dos ciudades, el Orquestador asíncronamente se comunica con Aeropuertos para pedirle el código de los aeropuertos en las ciudades indicadas. A continuación comprueba si tiene en su caché (base de datos local) si tiene los códigos para continuar con la ejecución y sinó usará el resultado de la respuesta asíncrona, esperando en caso de que aún no haya llegado la respuesta.

Handler

En el flujo de entrada del Banco.
Busca en la cabecera del mensaje SOAP un token más el iban para autenticarse, si estos son correctos permite la ejecución del método pagar, en caso contrario aborta la ejecución.
Los credenciales están guardados en una base de datos local.

UDDI

Todos los servicios internos y el cliente contará con un fichero uddi.xml en el cual se indicará la IP de la máquina en la que se encuentra desplegado jUDDI. Sin este fichero no podrán registrarse ni buscar a otros servicios.
Por otra parte, todos los servicios internos contarán con una clase Publish que permitirá publicar un servicio proporcionando su nombre y su endpoint. De esta forma, cada vez que se ejecute Tomcat, se publicarán en jUDDI todos los servicios desplegados en Axis2 en esa máquina. Análogamente, al detener Tomcat, dichos servicios serán eliminados del registro de jUUDI. En el caso de que el servicio sufra una caída y no se haya “des-registrado”, cuando vuelva a estar funcional, se comprobará si ya estaba publicado, en cuyo caso no se volverá a registrar y así evitaremos servicios duplicados. De esta forma, cuando se desconecte el servicio correctamente (deteniendo Tomcat), se eliminará de jUDDI el primer servicio publicado.
Finalmente, el Orquestador y el Cliente contarán con una clase Browse que se encargará de buscar los servicios publicados que necesitan para el funcionamiento del buscador de vuelos. El Orquestador buscará los servicios internos (Aeropuertos, Vuelos y Banco) y el Cliente buscará al Orquestador. La búsqueda de un servicio se hará únicamente cuando se necesite una comunicación con él, para así poder detectar la caída de cualquiera de ellos. En caso de no encontrar un servicio, se devolverá un mensaje de error.

Funcionamiento del sistema global

A. <u>Inicio y registro de los servicios:</u>

a. Inicialmente, se arrancará jUDDI en un único equipo.
b. Una vez iniciado jUDDI, arrancamos TOMCAT en todas las máquinas que contengan al menos un servicio.
c. Al ir desplegándose los servicios, estos se irán registrando en el UDDI.
d. Una vez registrados todos los servicios en UDDI, el sistema ya podrá recibir peticiones de clientes.

B. <u>Funcionamiento tras el inicio y registro de los servicios:</u>

a. El cliente en primera instancia podrá, o bien iniciar sesión en nuestro sistema (si ya se había registrado con anterioridad) o bien registrarse (si no se ha registrado anteriormente).

b. Una vez inicie sesión, se accederá al menú principal en el que podrá elegir una de estas tres opciones:

   i. Hacer una búsqueda de vuelos: El cliente indicará una ciudad de origen, una ciudad de destino, una fecha de partida y una fecha de regreso. A continuación, le enviará un mensaje con todo ello al Orquestador, quien le responderá con las ofertas disponibles para los datos introducidos. A continuación el cliente podrá optar entre:

      1. Efectuar una compra: Introduciendo el vuelo que desea comprar y su IBAN y un código de seguridad (proporcionado anteriormente por su entidad bancaria).
      2. Volver al menú principal

   ii. Ver las reservas que tenga ya efectuadas: El cliente verá por pantalla todas las compras (reservas) de vuelos que ha realizado hasta el momento.

   iii. Cerrar sesión: Se saldrá del buscador de vuelos.

Modificaciones respecto al diseño inicial

Basándonos en el diseño inicial para elaborar el servicio, hemos modificado tres aspectos respecto a este primer planteamiento base.

La primera modificación es en el handler. En nuestro sistema final el handler se encuentra en el servicio Banco. Su funcionamiento se basa en la lectura por parte del servicio interno de un IBAN y un código de seguridad (asociado a dicho IBAN) alojados en la cabecera del mensaje SOAP. Banco comprobará en su base de datos si son correctos y, en caso de no serlo, no leerá el cuerpo del mensaje. En caso contrario, procesará el mensaje y continuará su ejecución.

La segunda modificación se haya en la llamada asíncrona. Actualmente esta se encuentra en el Orquestador y se basa en una actualización de los códigos IATA de los aeropuertos de las ciudades que consulta el Cliente, dentro de una base de datos. De esta forma, las futuras búsquedas de los aeropuertos de dichas localidades será más rápida y en una base de datos local (en vez de consultando de nuevo al servicio externo SOAP). Cabe destacar que en un principio establecimos como llamada asíncrona el envío de un email, y aunque hemos corregido dicho aspecto, hemos dejado el envío del correo electrónico de confirmación del pago como una funcionalidad a mayores de nuestro servicio.

La última modificación afecta al registro y búsqueda de los servicios. Aunque en un principio no contemplamos que el Orquestador buscase a los servicios internos, finalmente decidimos implementar esta funcionalidad. De esta forma, todos los servicios internos se registrarán y, cuando el Orquestador necesite comunicarse con uno de ellos lo buscará en jUDDI y hará uso de su servicio.

Pruebas

1) El cliente busca y encuentra a nuestro servicio Orquestador inicia sesión correctamente, el Orquestador cae y el cliente quiere consultar Vuelos.
    - Resultado: El cliente detectará que Orquestador ya no está registrado en UDDI, se indicará y terminará la ejecución.

2) El orquestador no se registra a UDDI y el cliente lo busca.
    - Resultado: El cliente detectará que Orquestador no está registrado en UDDI, se indicará y terminará la ejecución.

3) El orquestador quiere acceder a cualquier servicio interno que está caído.
    - Resultado: Detectará que no está/n registrado/s en UDDI e informará del error al cliente.

4) Autenticación incorrecta al banco.
    - Resultado: El handler abortará la ejecución no dejando acceder al método de pagar.

5) Un servicio se cae sin eliminarse del registro UDDI (se apaga la máquina sin finalizar Tomcat). Tras unos minutos vuelve a conectarse (ejecuta Tomcat) para registrarse en UDDI.
    - Resultado: El servicio comprueba que ya está registrado por lo que no se vuelve a publicar. Busca la clave de su último registro y la usa para eliminarse cuando se apague Tomcat.