2020

Sistemas Distribuidos

Api-rest-agencia

Francisco Joaquín Murcia Gómez
48734281H
Universidad de Alicante

Índice

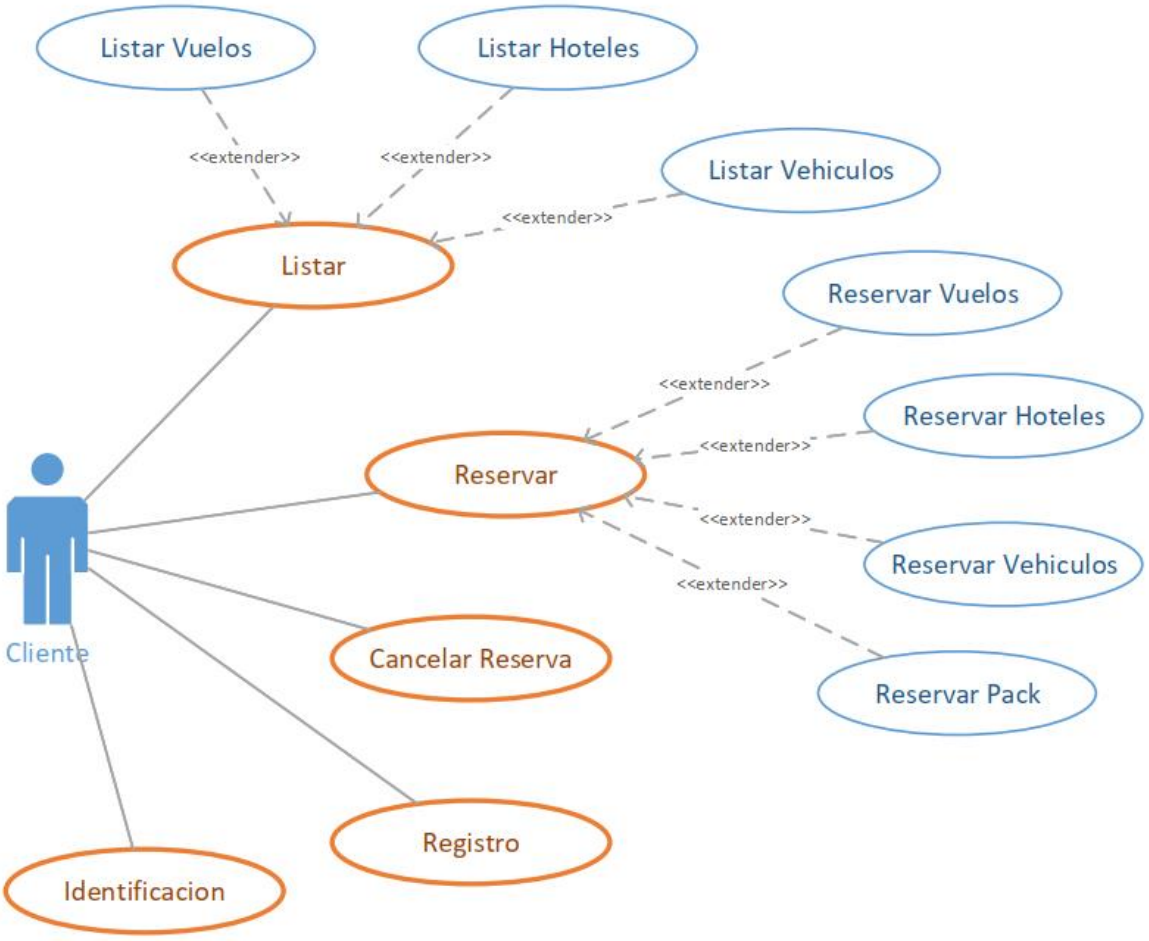
Introducción

En este proyecto se plantea el diseño de un sistema distribuido que permita la gestión de una agencia de viajes, donde los usuarios podrán reservar distintos servicios (hoteles, vuelos o coches). Dichos paquetes de viaje serán generados por la agencia de viajes a partir de las elecciones del cliente, este podrá elegir uno o varios servicios (paquetes).

Casos de uso

Aquí podemos ver los diferentes usuarios que tendremos los clientes y los proveedores, a continuación, sus casos de uso:

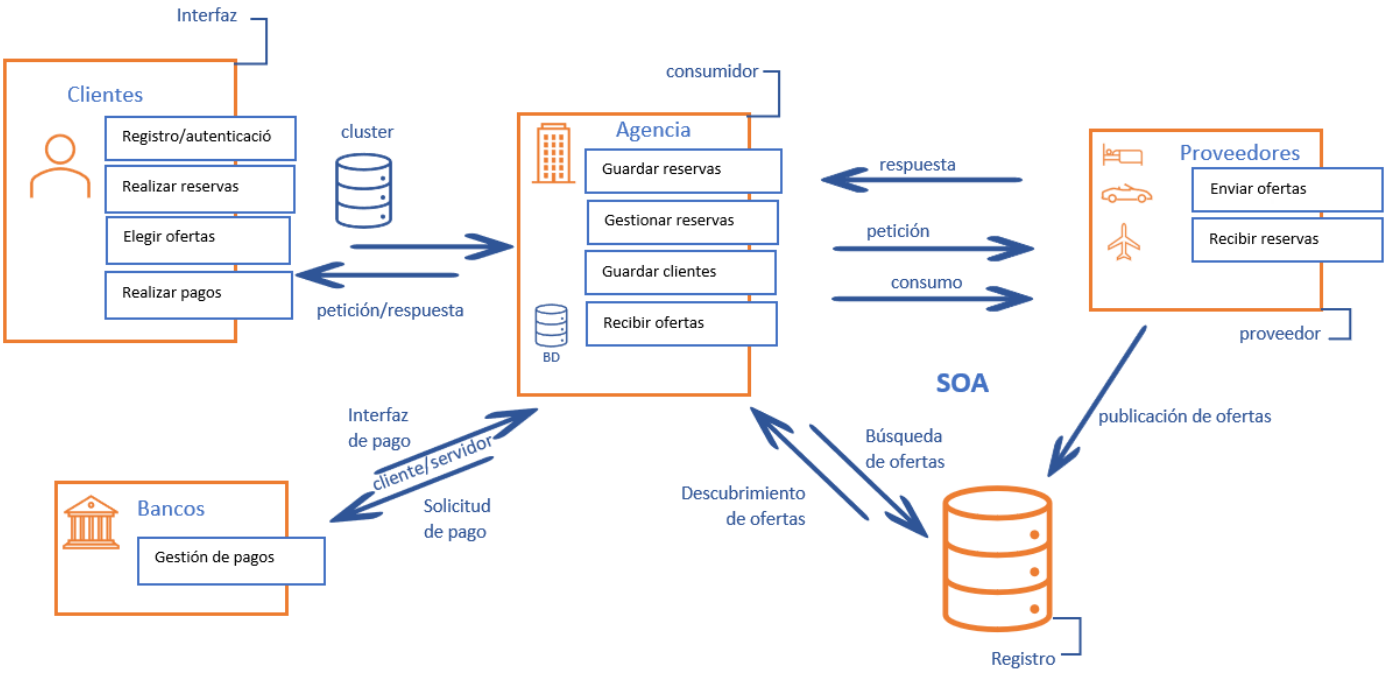
Aquí podemos observar las funciones que podrán realizar los clientes



A continuación, las funciones que podría realizar un proveedor

Arquitectura conceptual

He diseñado un esquema donde se puede observar de manera sencilla los actores, su función y como se comunican entre sí, no obstante, se indagará más en los siguientes apartados.



Actores

Encontraremos cuatro actores:

- **Clientes**
  Los clientes podrán registrarse, realizar ofertas y lo que conlleva consigo ver los productos cancelarlas…, elegir ofertas y realizar pagos
- **Agencia**
  Es la encargada de gestionar y guardar las reservas que hagan los clientes en su base de datos, en esta base de datos guardarán los clientes que se vayan registrando, también recibirán ofertes de los proveedores
- **Proveedores**
  Los proveedores se encargan de recibir las reservas que realizan los clientes que son gestionadas por la reserva y de enviar a esta las ofertas
- **Bancos**
  Los bancos son los encargados de gestionar los pagos

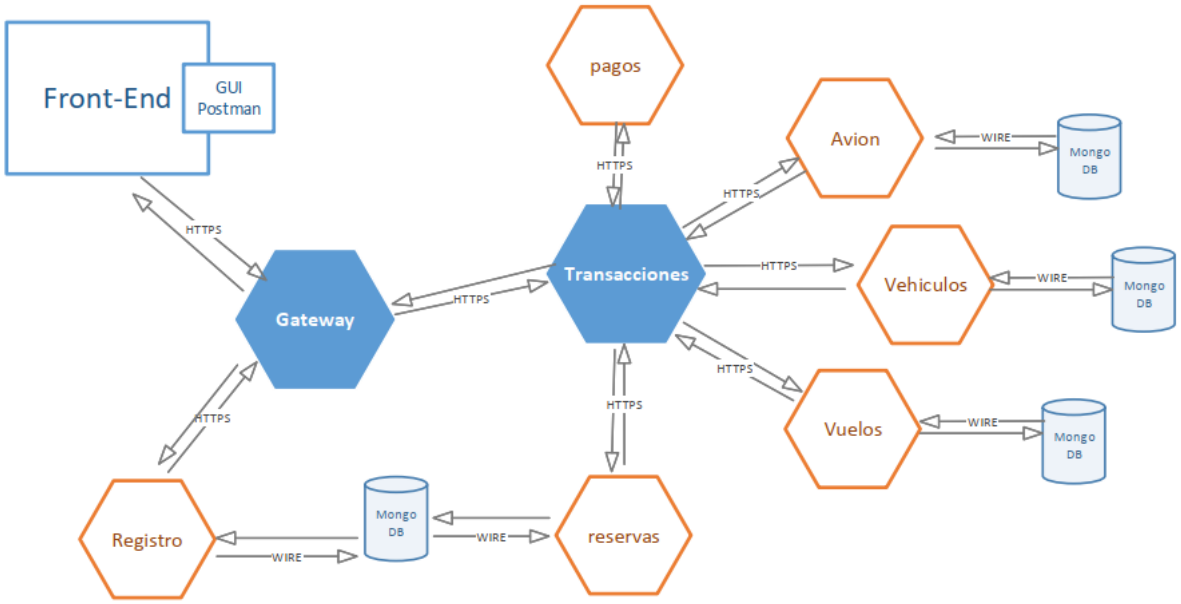
Arquitecturas empleadas

Para explicar el funcionamiento de la agencia de viajes hemos de dividir el esquema en tres partes:

- **Clientes-Agencia**
  Para la conexión de los clientes con el sistema de agencia estos se conectarán por media de cliente/servidor ya que clientes sería el Front-End y la agencia sería el back-end, a aparte hay un fuerte acoplamiento debido a que la tecnología es la misma y es la misma empresa, a pate es la arquitectura más utilizada en entornos web.
- **Agencia-Proveedores**
  Para este caso he usado una arquitectura tipo SOA debido a que cumple todos los requisitos de este, necesitamos interoperabilidad, tenemos un contrato que facilita el desacoplamiento y la independencia con ellos, se va a hacer una composición de varios servicios (packs), necesitamos localizar como acceder a esos proveedores y es síncrono. También con SOA proporcionamos libertad al proveedor que dependiendo de su contrato ya sea REST o SOAP se hará un servicio punto a punto con su contrato.
- **Agencia-Bancos**
  Para la agencia con los bancos se utilizará cliente/servidor debido a que normalmente los bancos son los que nos indican como conectaros a ellos, y normalmente con un protocolo propiedad del banco que suele estar basado en sockets, por eso he decidido emplear C/S con sockets en el que el banco es el servidor y la agencia el cliente.

Arquitectura conceptual-técnica

Teniendo en cuenta los anteriores apartados obtenemos el siguiente esquema de la aplicación que se puede ver las diferentes Web Services (hexágonos) y como interactúan entre ellos.



Los Web Services implementadas con NodeJS, las conexiones entre ellas son por medio de internet (HTTPS) empleando el rango de puertos 3000-3999, como podemos observar encontramos diferentes bases de datos, esta esta hechas con NoSQL, más concretamente con MongoDB, para facilitar despliegues en mi caso he usado MongoDB Atlas que es un cluster en línea donde he creado allí mis diferentes bases de datos, la base de datos está directamente conectada a los WS directamente y almacena los campos que crea cada WS (usuarios, vuelos, hoteles, vehículos o reservas).

Para la implementación de estas he usado pila MEAN (MongoDB, Exprex, Angular.js y Node.js) así como las siguientes librerías de npm: mongojs, morgan, moment, jwt-simple, express, https, node-fetch, fs, bcrypt helmet entre otras.

He usado Pasman para emular una interfaz Front-End ya que no he podido implementar este.

Servicios

En esta parte explicare los diferentes microservicios y sus End-Points en los que se vara el verbo HTTP, la ruta, lo que ha de contener el body, si necesita un token de usuario y una breve descripción

Gateway

Funciones

En la Gateway es un WS de tipo Rest, en este nos encargamos de gestionar lo que directamente nos envía el Front-End que básicamente son los casos de uso del cliente, (listar, reservar, registrarse o loguearse), implementado en la carpeta api-gw, este nos redirige a el WS de registro y al de transacciones.

End-Points

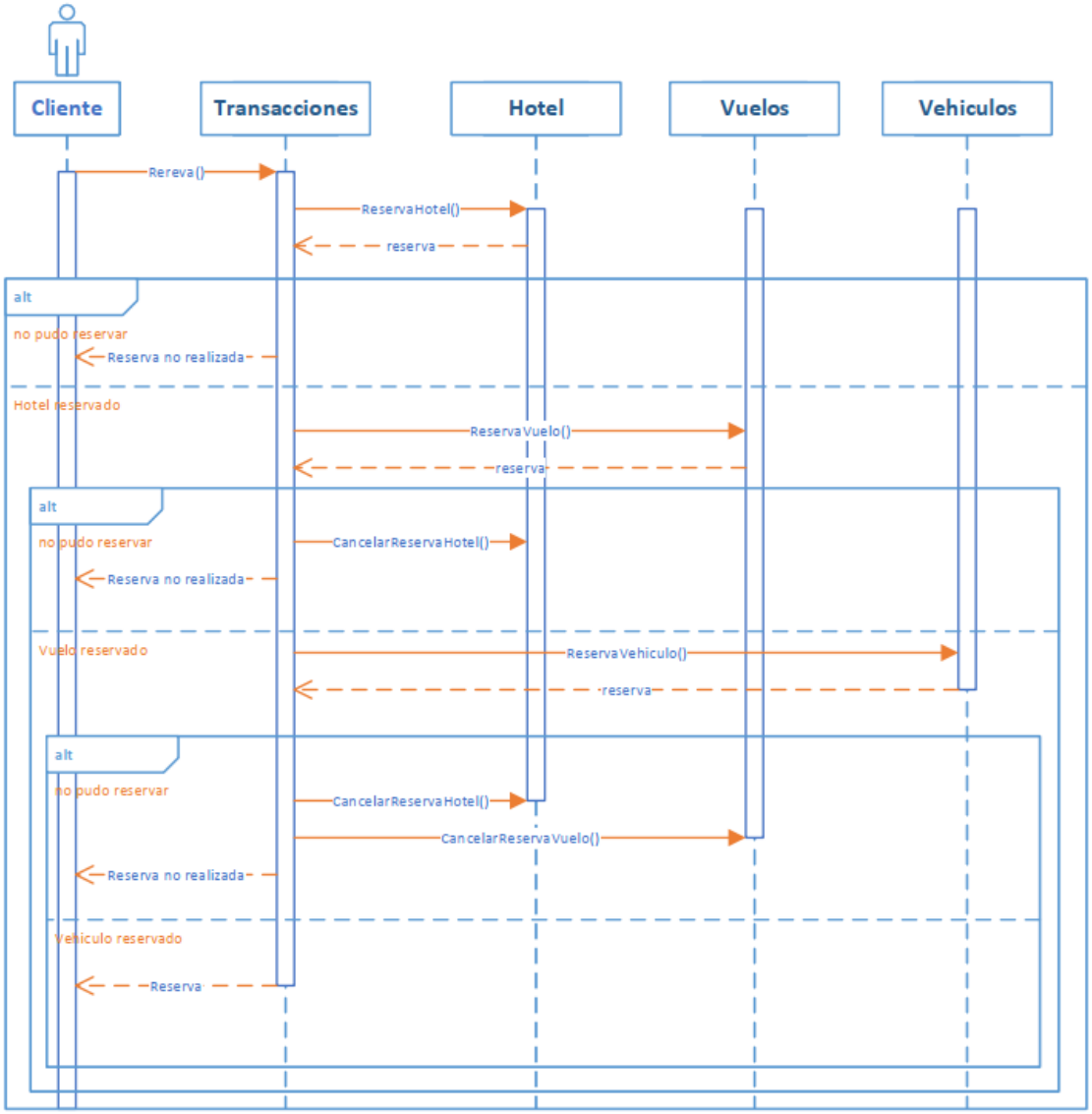
Verbo HTTP	Ruta	Body	Token	Descripción
GET	/vuelos		Si	Obtiene todos los vuelos
GET	/hoteles		Si	Obtiene todos los hoteles
GET	/vehiculos		Si	Obtiene todos los vehículos
POST	/registro	nombre, email, contraseña	No	Registra un usuario nuevo
POST	/login	email, contraseña	No	Realiza el logueo de un usuario y genera el token
POST	/reservar	email, idVuelo, idHotel, idVehiculos	Si	Realiza la reserva de uno o varios servicios
DELETE	/cancelar	email, idVuelo, idHotel, idVehiculos	Si	Cancela la reserva de uno o varios servicios

Transacciones

Funciones

Transacciones es el servicio que genera las reservas, la Gateway le envía una petición de reservar y transacciones reserva lo pedido por el cliente (vuelo, coche, hotel o una combinación de estos), a continuación, mostrare la reserva de un pack de vuelo y hotel y su cancelación, este es un el WS de tipo Rest implementado en la carpeta api-trans.

El WS de transacciones es transaccional, es decir, que gestiona múltiples servicios a la vez; para la parte concurrente MongoDB de por sí mismo se auto bloquea cuando se está realizando una operación de escritura para que solo se pueda escribir de uno en uno, evitando así inconsistencias de datos; pata las transacciones distribuidas en mi caso he implementado el patrón saga, debido a que no he encontrado algún framework que implemente 2PC con saga, lo he implementado yo mismo, su diagrama de secuencia seria asi:



Como se puede observar, por cada reserva que hago, después hay una acción compensatoria por si sucede algún problema, caída de servidor del proveedor o argumentos incorrectos, esto también valdría si solo se reservan dos elementos, o incluso uno, también este algoritmo lo uso para cancelar las reservas. Las reservas y cancelaciones generan un resultado tal que asi:

```
{
Usuario:fran@gmail.com
Vuelo:{
        origen:"Oman"
        destino:"China"
        compañia:"ryanair"
        precio:"$238.32"
        fecha_salida:"9/11/2020"
        fecha_llegada:"13/10/2020"
        disponible:"si"
        }
Hotel:{
        nombre:"Harber, Cummerata and Frami"
        pais:"Philippines"
        ciudad:"Bolo"
        precio:"$100.00"
        fecha_salida:"9/5/2020"
        fecha_llegada:"8/4/2020"
        disponible:"si"
        }
Vehiculo:""
}
```

```
{
Usuario:fran@gmail.com
Vuelo:"Reserva cancelada correctamente"
Hotel:"Reserva cancelada correctamente"
Vehiculo:""
}
```

Si ocurriera un fallo por ejemplo al reservar el vuelo hotel se cancelaria y se vería así:

```
{
Usuario:fran@gmail.com
Vuelo:"El vuelo no esta disponible"
Hotel:"Reserva cancelada correctamente"
Vehiculo:""
}
```

End-Points

Verbo HTTP	Ruta	Body	Token	Descripción
POST	/reservar	email, idVuelo, idHotel, idVehiculos	Si	Reserva un servicio o una combinación de estos
DELETE	/cancelar	email, idVuelo, idHotel, idVehiculos	Si	Cancela la reserva de un servicio o una combinación de estos

Registro

Funciones

Registro es un WS de tipo Rest, este se encarga de las funciones CRUD de los usuarios, más concretamente su registro y logueo, este WS emplea mecanismos de encriptación para no guardar nunca la contraseña del usuario, ya que guarda un hash, también hacemos uso de tokens para encriptar la identificación del usuario, implementado en la carpeta api-registro

End-Points

Verbo HTTP	Ruta	Body	Token	Descripción
POST	/login	nombre, email, contraseña	No	Se loguea un usuario
POST	/registro	email, contraseña	No	Se registra un usuario

Pagos

Funciones

El Web Service de pagos simula la respuesta de una entidad bancaria, es decir hay un método GET que con una probabilidad del 80%devuelveayqe el pago ha sido exitoso, en caso contrario se refiera a que un usuario fallo al introducir su pago (tarjeta invalida, saldo insuficiente o directamente el banco ha rechazado el pago), he realizado esto porque como ya he mencionado en la arquitectura conceptual, lo normal que nosotros nos conectemos al banco con su protocolo y este nos devuelva las peticiones como el banco desea, lo he implementado en la carpeta api-pagos

End-Points

Verbo HTTP	Ruta	Body	Token	Descripción
GET	/pago		Si	Simula la interacción de un banco

Aviones, Vehículos, Vuelos

Funciones

Estos tres WS son casi idénticos, por eso los explicare en un mismo, estos gestionan todo o relacionado con las ofertas, es decir su CRUD, también usamos dos llamadas para realizar reservas, el cual se encargará de marcar la oferta como no disponible y asignara el usuario que ha reservado esa oferta, junto con otro que cancela la reserva, es decir deshace los cambios mencionados anteriormente, estas implementaciones están hechas en las capetas api-hoteles, api-vuelos y api-vehiculos

End-Points Aviones

Verbo HTTP	Ruta	Body	Token	Descripción
GET	/api/ofertas		Si	Obtenemos las ofertas
GET	/api/ofertas/{id}		Si	Obtenemos la oferta
POST	/api/ofertas/{id}	origen, destino, compañía, precio, fecha_salida(d/mm/yyyy), fecha_llegada(d/mm/yyyy)	No	Añadimos una oferta
PUT	/api/ofertas/	origen, destino, compañía, precio, fecha_salida(d/mm/yyyy), fecha_llegada(d/mm/yyyy)	No	Modificamos una oferta
PUT	/api/reserva/{id}	email	Si	Creamos una reserva
DELETE	/api/reserva/{id}	email	Si	Cancelamos la reserva
DELETE	/api/ofertas/{id}		Si	Borramos una oferta

End-Points Vehículos

Verbo HTTP	Ruta	Body	Token	Descripción
GET	/api/ofertas		Si	Obtenemos las ofertas
GET	/api/ofertas/{id}		Si	Obtenemos la oferta
POST	/api/ofertas/{id}	Marca, modelo, año, kilómetros, cochera, precio, fecha_salida(d/mm/yyyy), fecha_llegada(d/mm/yyyy),	No	Añadimos una oferta
PUT	/api/ofertas/	Marca, modelo, año, kilómetros, cochera, precio, fecha_salida(d/mm/yyyy), fecha_llegada(d/mm/yyyy),	No	Modificamos una oferta
PUT	/api/reserva/{id}	email	Si	Creamos una reserva
DELETE	/api/reserva/{id}	email	Si	Cancelamos la reserva
DELETE	/api/ofertas/{id}		Si	Borramos una oferta

End-Points Hoteles

Verbo HTTP	Ruta	Body	Token	Descripción
GET	/api/ofertas		Si	Obtenemos las ofertas
GET	/api/ofertas/{id}		Si	Obtenemos la oferta
POST	/api/ofertas/{id}	Nombre, pais, ciudad, precio, fecha_salida(d/mm/yyyy), fecha_llegada(d/mm/yyyy)	No	Añadimos una oferta
PUT	/api/ofertas/		No	Modificamos una oferta
PUT	/api/reserva/{id}	email	Si	Creamos una reserva
DELETE	/api/reserva/{id}	email	Si	Cancelamos la reserva
DELETE	/api/ofertas/{id}		Si	Borramos una oferta

Reserva

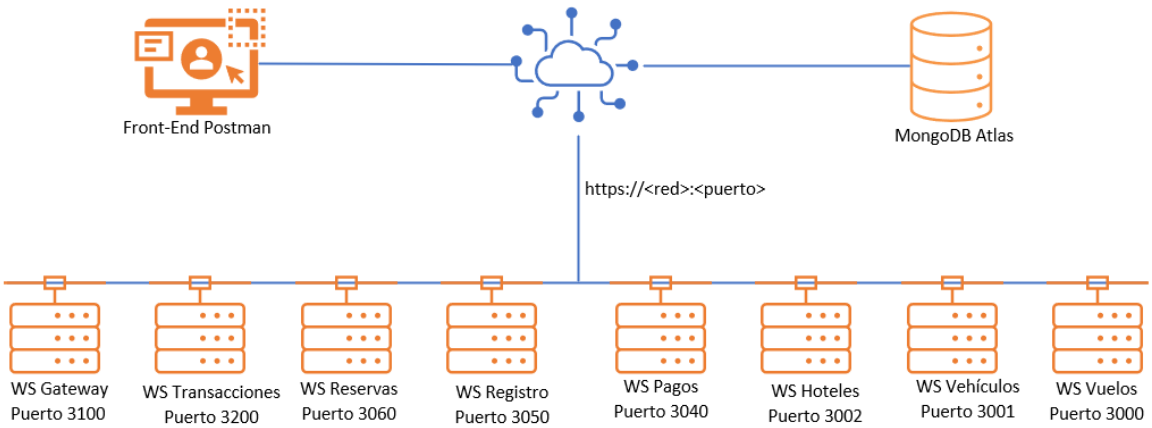
Funciones

Este WS no lo tengo implementado, tendría que implementarlo en enero antes del dia del despliegue, su funcionamiento seria simple, cada vez que se crea una reserva el WS de transacción enviaría esta reserva al api reserva que actuaria como un CRUD, guardaría estas reservas en la misma base de datos que la de usuarios ya que entiendo una reserva tendría un usuario asignado, cuando se cancelara una reserva transacciones le mandarina la orden y este la eliminaría de la base de datos, implementare esto en api-reserva

End-Points

Verbo HTTP	Ruta	Body	Token	Descripción
GET	/api/reservas		Si	Obtenemos las ofertas
GET	/api/ reservas /{id}		Si	Obtenemos la oferta
POST	/api/ofertas	email, vuelo, hotel, vehiculo	Si	Añadimos una reserva
DELETE	/api/reserva	email, vuelo, hotel, vehiculo	Si	Cancelamos la reserva

Arquitectura de Despliegue



Como se puede observar mi arquitectura de despliegue seria simple, los microservicios estarían ejecutándose en puertos diferentes de la misma res, el Front-End y el cluster de bases de datos se conectarían a estos por medio de internet

Seguridad

Para securizar el sistema he empleado lo siguiente:

- Canales de comunicación en red con certificados TLS/SSL (HTTPS)
- Uso de la librería helmet
- Encriptación de las contraseñas de los usuarios basándonos en un hash
- Empleo de tokens más concretamente JSON Web Token (jwt) para la gestión de procesos por parte del usuario
- Uso de Open Authorization (OAuth) para que los usuarios no den toda su información

Resultados y conclusiones

Conclusiones

Como resumen de lo mencionado he usado 8 microservicios para formar el back-end de la agencia de viajes, estos son de tipos Rest conectados entre sí con HTTPS, entre los 8 encontramos una gateway para gestionar las peticiones del front-end y un método de transacciones para orquestar diferentes servicios a la vez, estos servicios están conectados a una base de datos de NoSQL en línea.

Esta práctica la podría definir como un puzle que no sabas de que imagen se trata a hasta que lo finalizas de montar, al principio de la asignatura no entendía como se podría conectar todo, de hecho, en el día que simulamos el despliegue tenía solo 3 microservicios que no se conectaban bien entre ellos y que no sabia si iba en el buen camino o no, y cuando monte casi todo, empecé a entender como funcionaban las cosas, no obstante, la práctica no está acabada al 100%. Me hubiera gustado dedicarle más tiempo a esta asignatura ya que en mi opinión es interesante, el problema es que para gestionar todas las asignaturas no me ha dado el tiempo.

Proyecto

Repositorio del proyecto:

Le he invitado al repositorio de github, mi usuario es fmurciag, solo tiene que clonarlo en una carpeta e iniciar cada api.

https://github.com/fmurciag/back-end-agencia.git

Base de datos:

Para la base de datos le he invitado al proyecto de MongoDB atlas por si desea ver la base de datos, le he invitado en los correos pmacia@gcloud.ua.es y pmacia@dtic.ua.es ya que no se cual es el que usa, no obstante, le llegara una invitación mía por parte del correo fmurciag@gmail.com