19-7-2021

Sistema de agencia de viajes

Sistemas Distribuidos

José Miguel Carrión Pinedo

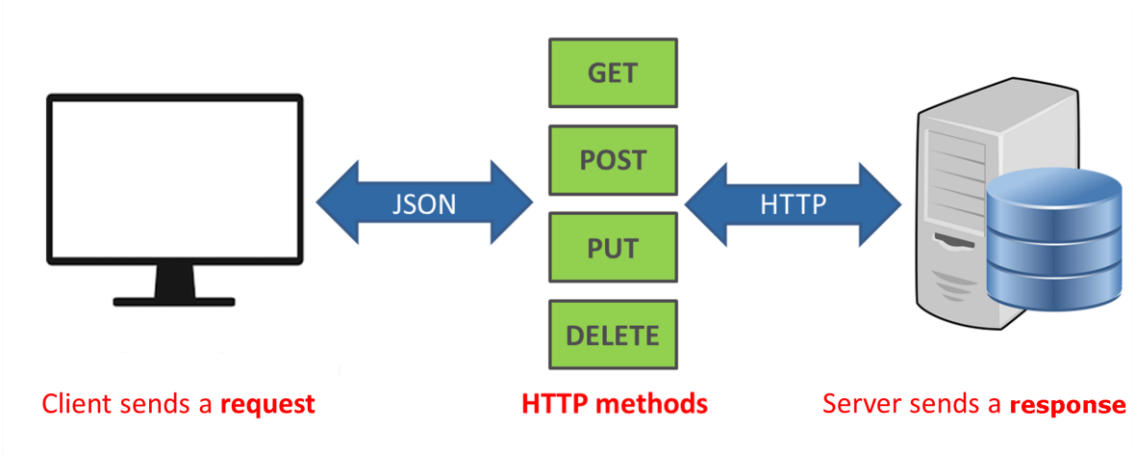
48625490V

TABLA DE CONTENIDO

1. INTRODUCCIÓN

El objetivo de esta práctica es poner en práctica los numerosos conceptos vistos en teoría con la finalidad de asimilarlos, para ello he explorado toda la serie de tecnologías que facilitan en el diseño de aplicaciones distribuidas basadas en microservicios distribuidos por internet.

Para entender de manera visual y simplificada lo que hago en cada uno de los servicios de la aplicación (un API RESTful sobre HTTP con un CRUD implementado en todos los servicios exceptuando el de usuarios por razones que explicaré más adelante).



Tras muchos intentos e investigación finalmente, me he decidido a desarrollarlo en una de las tecnologías más ampliamente usadas para hacer plataformas autosuficientes, MEAN Stack, lo cual significa que he usado los siguientes frameworks (todos están basados en JavaScript).

MongoDB para gestionar la BBDD, que he conectado a Mongo Atlas (BBDD desacoplada en línea y la aplicación Compass para facilitar su uso), ExpressJS y NodeJS para implementar los servidores, y, por último, AngularJS para el frontend desacoplado.

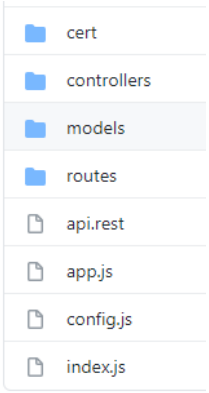
Además, cabe destacar que para escribir el código he usado VSCode con plugins para facilitar la lectura y corrección de errores de los framework anteriormente mencionados, y Postman para probar el CRUD del backend fácilmente.

El objetivo de haberlo implementado de este modo es facilitar su **escalabilidad**, lo cual es algo crucial como hemos visto en teoría. Por lo tanto, cada servicio esta en su propia carpeta, distribuido en directorios para facilitar su lectura y su modalización. Todo esto, le **proporciona una mayor resiliencia**, porque si cayera uno de los servicios, los otros no se verían afectados.

agencia	Sistema de anulacion implementado	yesterday
banco	Pack implementado	19 hours ago
frontend	Nuevo frontend para clase	23 hours ago
nuevoFront/frontend	Pack implementado	19 hours ago
proveedorAviones	Sistema de anulacion implementado	yesterday
proveedorCoches	Añadida entidad bancaria	3 days ago
proveedorHoteles	Implementado el sistema de reservar y pagos	2 days ago
proveedorUsuarios	Añadida entidad bancaria	3 days ago
reservas	Sistema de anulacion implementado	yesterday

Cada carpeta es un servicio desacoplado del resto

Está dividido en carpetas para facilitar la modulización.



Sin embargo, cabe destacar que de caer la agencia (que hace de Gateway, ya que hace de intermediario entre el frontend y el resto de los proveedores) hasta que fuera solventado, el cliente no podría conectarse al resto de servicios, además, es en la agencia dónde más esfuerzo se ha puesto a la securización de la misma.

2. DESPLIEGUE

Para desplegarlo, el primer paso será descargar el proyecto entero de github, lo he hecho público para que sea más rápido el descargo, por lo tanto, haremos un "git clone", cuyo enlace al proyecto es https://github.com/josemicarrion99/sd-prac.

Así que en la terminal haremos: **"git clone https://github.com/josemicarrion99/sd-prac".**

Acto seguido, en los servicios que vayamos a iniciar en ese ordenador, entraremos en su directorio y haremos **"npm i -D && npm start".**

"npm i -D" creará la carpeta "node_modules" donde se descargarán todos los paquetes necesarios para el correcto funcionamiento de la aplicación.

Un paso crucial es cambiar las URL de la agencia y de los "services" en el frontend, ya que como hasta ahora lo he usado todo en mi localhost hay que poner en lugar de "localhost", la IP del ordenador en el que se esté corriendo dicho servicio, para ello pondremos el comando **"ifconfig"** en el terminal.

Para la agencia los URL se encuentran en el archivo "config.js":

```
urlUsuarios: 'https://localhost:3001/api/usuarios',

urlCoches: 'https://localhost:3002/api/coches',

urlHoteles: 'https://localhost:3003/api/hoteles',

urlAviones: 'https://localhost:3004/api/aviones',

urlBanco: 'https://localhost:3005/api/banco',

urlReservas: 'https://localhost:3006/api/reservas',
```

Y en el frontend, dentro de "src/app/services":

```
  URL_API = 'https://localhost:3000/api/aviones' //300
··//URL_API·=·'https://172.20.42.16:3000/api/aviones'·
```

Acto seguido, usaremos postman para probar algunos de los CRUD para asegurarnos de que funcionan correctamente, o directamente acceder al frontend para verlos en funcionamiento.

Una vez importadas las rutas, ya podemos proceder a usarlo.

3. END POINTS

En este apartado vamos a analizar los CRUD de cada servicio desde **el punto de vista de la agencia**, es decir, **no incluiremos los delete**, por ejemplo, ya que **están implementados** es el servicio en cuestión, pero no tiene acceso al mismo, ya que un usuario no puede tener la capacidad de eliminar un coche, por ejemplo.

GET

Ruta	Descripción	Autenticación necesaria
/api/coches	Devuelve todos los coches	NO
/api/coches/{productId}	Devuelve el coche con la id indicada	NO
/api/aviones	Devuelve todos los aviones	NO
/api/ aviones/{productId}	Devuelve el avión con la id indicada	NO
/api/hoteles	Devuelve todos los hoteles	NO
/api/ hoteles /{productId}	Devuelve el hotel con la id indicada	NO
/api/coches	Devuelve todos los coches	NO
/api/coches/{productId}	Devuelve el coche con la id indicada	NO
/api/usuarios	Devuelve todos los usuarios	SI
/api/banco/{productId}	Devuelve la cuenta bancaria con la id indicada	NO
/api/banco	Devuelve todas las cuentas bancarias	NO

POST

Ruta	Descripción	Autenticación necesaria
/api/coches	Crea un coche	SI
/api/aviones	Crea un avión	SI
/api/hoteles	Crea un hotel	SI
/api/usuarios	Crea un usuario	NO

/api/usuarios/tokens	Obtiene el token del usuario introducido, si la contraseña es correcta	NO
/api/banco	Crea una cuenta bancaria	SI

PUT

Ruta	Descripción	Autenticación necesaria
/api/coches	Modificamos un coche	SI
/api/aviones	Modificamos un avión	SI
/api/hoteles	Modificamos un hotel	SI
/api/banco	Modificamos una cuenta bancaria	SI

4. EXPLICACIÓN DE CADA SERVICIO

PROVEEDOR DE PRODUCTOS

Esto envuelve a tanto coches, como aviones, como hoteles, ya que lo único que cambia entre ellos solo atributos en su BBDD, todos ellos tienen en común el atributo "reservadoDesde", "reservadoHasta", "disponible" (indica si han sido reservados o no) y "precio", los otros atributos que los identifican varían, coches, por ejemplo, tiene "modelo" y "matrícula".

```
const productSchema = new Schema(
  {
    modelo: { type: String, required: true },
    matricula: { type: String, required: true },
    precio: { type: Number, required: true },
    disponible: { type: Boolean, default: true },
    reservadoDesde: {type: Date},
    reservadoHasta: {type: Date}
  },
```

Estructura en mongoose de la BBDD de coches

PROVEEDOR DE USUARIOS

Solo será usado para la creación de usuarios y la generación de tokens. Para la implementación de la generación de tokens he usado **JSON web tokens** que es ampliamente usada en este campo.

Cabe destacar para aclararlo, que la BBDD no almacena los tokens, empleamos la librería bcrypt para generarlos y, además, hacer que tengan un tiempo de caducidad (20 min), de querer modificarlo, simplemente lo haríamos en el fichero "config.js" en la agencia, modificando el valor "tokenTime".

Para generar dichos tokens simplemente nos loggueamos con la con el correo de usuario y con la contraseña, y si son correctas, nos devolverá el token, el post que está en la ruta "/api/usuarios/tokens". Esto nos permitirá acceder a rutas que antes estaban restringidas, como los posts y updates.

AGENCIA

La agencia o Gateway es el punto central que por así decirlo gestiona las peticiones, redirigiéndolas al servicio que toque, en el apartado "3. End points" se concreta más sobre qué más se puede hacer en la agencia, es decir, en qué pueden hacer los usuarios.

Y, como hemos mencionado anteriormente, si cayera la agencia, se acabaría la capacidad del sistema de gestionar las peticiones del usuario, por lo que es crucial de que, si cayese, que fuese restaurada cuánto antes sea posible.

BANCO

La entidad bancaria tiene en su BBDD guardadas cuentas bancarias, dónde guarda el código de cuenta, el saldo, y el correo electrónico asociado a un cliente, por lo tanto, cuando un cliente intenta reservar un servicio, se buscará su cuenta bancaria en el banco, y si coincide el correo del cliente con el código de cuenta, entonces se llevará a cabo la transacción si tiene suficiente dinero.

FRONTEND

El frontend, al estar programado al completo en angular, y haciendo uso de **Bootswatch** significa que está desacoplado, tiene sus propios modelos de cada objeto, y usa los servicios para comunicarse con la agencia y que le devuelva o haga lo que haga falta.

Angular es un framework con una curva de aprendizaje empinada, por ello no quiero profundizar en su funcionamiento, pero se basa en uso de componentes y servicios y su relación entre ellos.

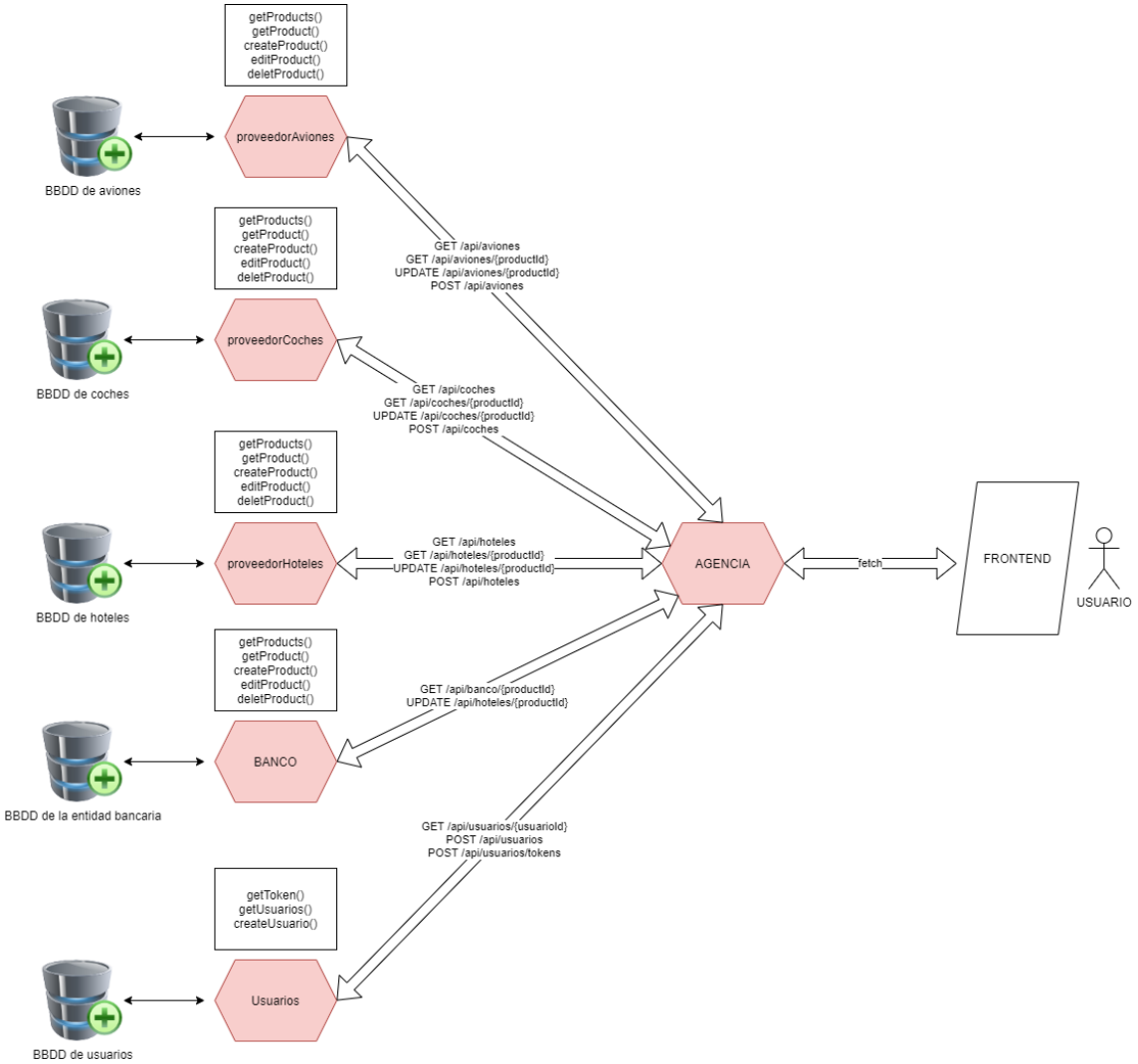
5. PATRÓN SAGA Y GESTIÓN TRANSACCIONAL

El patrón SAGA que he escogido implementar, es el **patrón SAGA mediante coreografía**, que es necesario para la gestión de transacciones, porque para hacer una, como por ejemplo una reserva, hay que realizar una serie de operaciones de manera secuencial, que depende cada una de la siguiente y de la anterior, y de fallar una de ellas, habría que revertir el proceso, por ello mismo he empleado "async await" para que no se superpongan solicitudes, y se creen colas en la BBDD. Y también, el "toPromise()" en TS.

Todo esto nos permite tener un sistema distribuido completamente consistente, gracias a usar un patrón SAGA coreografiado, no requerimos de un punto centralizado, es mejor que depender de la agencia para cada una de las operaciones.

6. ARQUITECTURA CONCEPTUAL

En el dibujo a continuación, he situado todos los proveedores de distintos servicios a la izquierda, para enfatizar que todos pasan por la agencia, incluido el banco y la BBDD de usuarios para asegurarnos de que, en caso de caer la agencia, se pueda navegar por la web, pero no se pueda comprar ni reservar nada, ya que no se podría ofertar nada.



7. SEGURIDAD

La seguridad de los microservicios puede resumirse en los siguientes apartados:

HTTPS

Gracias al uso del paquete del mismo nombre, todos los microservicios (no solo la agencia) hacen uso de este mediante un sistema de **certificados auto-firmados**, que han sido generados con **OpenSSL**, que están subidos al github por comodidad, pero que, en una situación real, no sería así.

USO DE TOKENS.

Los tokens, más concretamente, de tipo 'Bear token' es el método usado para el acceso a rutas protegidas, y al caducación de la sesión de usuarios, es gestionado por el proveedor de usuarios.

Son generados cada vez que un usuario inicia sesión, las rutas que han sido protegidas puede consultarse en el apartado "3. End points".

Para explicarlo superficialmente:

Usuario inicia sesión => proveedor de usuarios genera la cadena encriptada (token) => si el usuario intenta acceder a una ruta protegida => el microservicio los descodifica => comprueba si es válido => permite el acceso o lo rechaza.

BCRYPT

Para asegurarnos de que al hacer un get de un usuario, no sea revelada su contraseña, o que alguien acceda a la BBDD y la obtenga, nada más el usuario sea creado, encriptaremos la contraseña usando la librería pública BCRPT, usando hash, cuando el usuario más adelante inicie sesión hará el proceso inverso y comparará las contraseñas.

8. CONCLUSIÓN

Este proyecto ha sido el más desafiante que he experimentado en lo que llevo de carrera, abrumador al principio, pero cuando llegas al final y ves todo lo que has aprendido es increíblemente gratificante, todas las tecnologías usadas las he tenido que aprender por mi cuenta desde 0 y no pensaba que fuera capaz, honestamente, aunque dado que este es el primer año que se ha llevado a cabo este método de evaluación de las prácticas de Sistemas Distribuidos, tengo que confesar que opino que la carga y dificultad de la práctica es quizás demasiada, y sugerir de cara a años venideros considerar alguna modificación de algún tipo.

Pero sopesando todo, ha sido una ardúa aunque grata experiencia.

BIBLIOGRAFÍA

Serie de vídeos del canal de youtube Fazt Code sobre MEAN Stack o Angular, en vídeos como:

https://www.youtube.com/watch?v=qf8-JzU-4lE

https://www.youtube.com/watch?v=AR1tLGQ7COs

Información sobre MEAN Stack: https://es.wikipedia.org/wiki/MEAN

Información sobre el patrón SAGA: https://enmilocalfunciona.io/el-lado-oscuro-de-los-microservicios-transacciones-cross-service/

Información sobre API RESTful: https://www.redhat.com/es/topics/api/what-is-a-rest-api

Y una larga de lista de foros de Stack Overflow y vídeos de youtube entre otros.