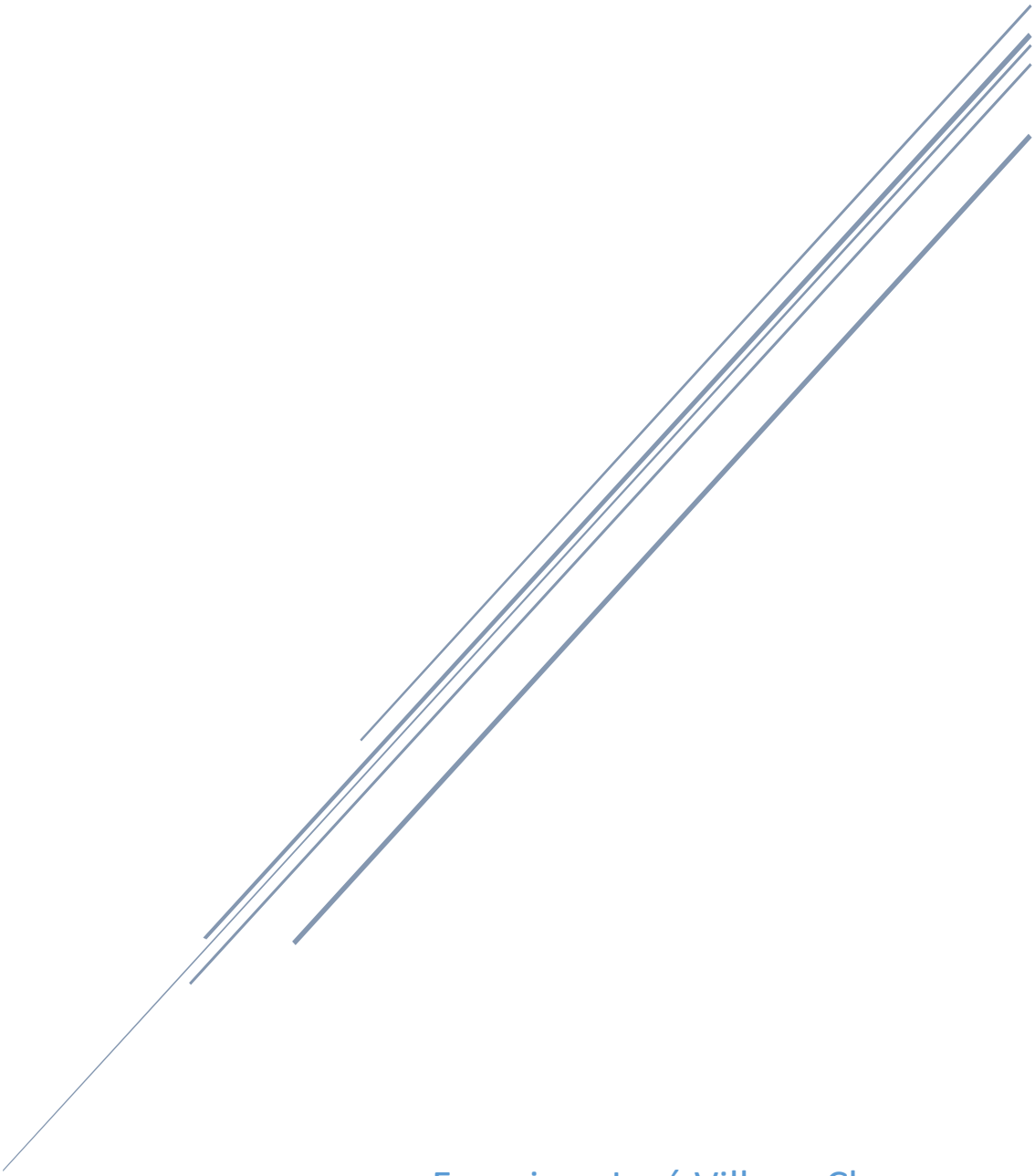
SISTEMA DE AGENCIA DE VIAJES

Agencia con API y RESTFull



Francisco José Villena Charcos

Sistemas Distribuidos

Índice

1. Introducción.

En este documento vamos a ver los diferentes aspectos que hemos visto y tenido que superar durante la realización de este proyecto. Este es una Agencia de viajes hecho con API REST, donde un cliente podrá ver y reservar los distintos servicios que ofrece la agencia.

Entre estos servicios se encuentran los coches, el cual un cliente podrá reservar con un destino y unas fechas ya predefinidas por los proveedores del servicio, los aviones, donde podrá reservar un asiento de un vuelo con un destino y fechas predefinidas por los proveedores y los hoteles, donde se reservará una habitación de un hotel en concreto con unas fechas ya predefinidas también.

Todas las reservas tendrán que comunicarse con el banco para poder pagar si se tuviese el dinero necesario.

A continuación seguiremos viendo los distintos objetivos que hemos ido cumpliendo y como los distintos servicios se interconectan y comunican con la agencia.

2. Objetivos.

Estudiar las diferentes tecnologías y herramientas que facilitan el diseño de la aplicación en un entorno distribuido, la cual estará basada en servicios o micro servicios.

El objetivo principal es crear una API REST la cual se comportará como un CRUD, o sea, que haces operaciones de Create, Read, Update y Delete. Con fin de conseguir esto, usaremos tecnologías MEAN:

- Node + Exppress como servidor HTTPS
- Un gestor de base de Datos Mongo en la nube.
- Un front end de fácil uso para el cliente en Angular.

3. Herramientas y Tecnologías Utilizadas.

Para la realización de este proyecto se ha usado una máquina virtual con Linux puesto que en mi ordenador personal tengo Windows y en la universidad se usará Linux también. Aparte en la máquina virtual son más fáciles las copias de seguridad o clonado de discos. La versión de Linux es la 20.04 LTS.

Entre las diferentes herramientas y tecnologías utilizadas encontramos: Visual Studio, Postman, Firefox, NodeJS, Angular, Git y Mongo Atlas.

3.1. Virtual Studio 1.58.0

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que podemos cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft.

3.2. Postman 7.36.6

Postman es una herramienta que se utiliza, sobre todo, para el testing de API REST, aunque también admite otras funcionalidades que se salen de lo que engloba el testing de este tipo de sistemas.

Gracias a esta herramienta, además de testear, consumir y depurar API REST, podremos monitorizarlas, escribir pruebas automatizadas para ellas, documentarlas, mockearlas, simularlas, etc.

Al comienzo del proyecto para probar las funciones que iba haciendo en el backend usaba la herramienta de Postman para hacer los GETS de los servicios, los POST,… En caso de no tener frontend, postman en un sustituto, aunque los resultados no se vean tan bien, bastante viable y funcional.

3.3. Firefox 89.0.2

Es un navegador web libre y de código abierto desarrollado para distintas plataformas, está coordinado por la Corporación Mozilla y la Fundación Mozilla. Usa el motor Gecko para renderizar páginas web, el cual implementa actuales y futuros estándares web.

Es el navegador que se ha usado para ver y usar la aplicación de la Agencia. También se podría usar Chrome.

3.4. NodeJS 14.17.1

Node.js es un entorno de tiempo de ejecución de JavaScript. Node.js utiliza un modelo de entrada y salida sin bloqueo controlado por eventos que lo hace ligero y eficiente. Puede referirse a cualquier operación, desde leer o escribir archivos de cualquier tipo hasta hacer una solicitud HTTP.

El funcionamiento interno es bastante interesante. Node.js opera en un solo subproceso, utilizando el modelo entrada y entrada sin bloqueo de la salida, lo que le permite soportar decenas de miles de conexiones al mismo tiempo mantenidas en el bucle de eventos.

El nodo está completamente controlado por eventos. Resumiendo podemos decir que el servidor consta de un subproceso que procesa un evento tras otro.

Sobre este entorno introduciremos otras librerías y tecnologías, así como middlewares que nos ayudarán a optimizar y dar eficiencia a la implementación. Estas se pueden ver en nuestro package.json, y destacan: npm, express, nodemon y mongodb.

3.5. Angular 11.2.14

Angular es un framework opensource desarrollado por Google para facilitar la creación y programación de aplicaciones web de una sola página, las webs SPA.

Angular separa completamente el frontend y el backend en la aplicación, evita escribir código repetitivo y mantiene todo más ordenado gracias a su patrón MVC (Modelo-Vista-Controlador) asegurando los desarrollos con rapidez, a la vez que posibilita modificaciones y actualizaciones.

Se ha utilizado Angular para la parte del frontend sobretodo porque es modular y escalable, aparte de que es de fácil aprendizaje y en internet hay mucha información.

Para la versión de angular se ha tenido que bajar a la 11.2.14 pues es la mejor versión compatible con la versión de Node.js de la universidad.

3.6. Git 2.25.1

Git es un sistema de control de versiones. Es de código abierto y con un mantenimiento activo. Presenta también una arquitectura distribuida y al contrario de otros sistemas de control de versiones más antiguos, Git también alberga el historial completo de todos los cambios.

Guardaremos nuestro proyecto en un repositorio de Git donde a diario o casi a diario he subido versiones de mi código. También tengo otro repositorio como una especie de copia de seguridad por cualquier problema que pueda pasar.

3.7. Mongo Atlas 4.4.6

MongoDB Atlas es una base de datos en nube global pensada para aplicaciones modernas que es distribuida y segura por defecto, además de estar disponible como servicio totalmente gestionado en AWS, Azure y Google cloud.

Esta es la base de datos que utilizaremos para guardar los distintos servicios, los usuarios registrados y las reservas que hayan hecho los mismos.

4. API y Arquitectura técnico-Conceptual de la API.

API es una abreviatura de Application Programming Interfaces, o sea, interfaz de programación de aplicaciones. Es un conjunto de definiciones y protocolos utilizados para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones. Básicamente podemos decir que establece como un módulo de un software se comunica con otro.

Una de las principales funciones de las API's es facilitar el trabajo a los desarrolladores ahorrándoles tiempo y dinero. API te permite utilizar una API ya existente para que no tengas que crearla desde 0, perdiendo mucho tiempo por el camino.

Imagine la API como si fuera el mediador entre los usuarios o clientes y los recursos o servicios web que quieren obtener. Con ellas, las empresas pueden compartir recursos e información y mantener la seguridad, el control y la autenticación, lo cual les permite determinar el contenido al que puede acceder cada usuario.

Para este proyecto se ha llegado más lejos pues se implementado API de Restful. Aquí cuando se envía una solicitud del cliente esta transfiere una representación del estado del recurso requerido a quien lo haya solicitado o al extremo. La información, o representación, se entrega por medio de HTTP en uno de estos formatos: JSON. Un ejemplo del formato JSON:

{"email": "PracticaSD@gmail.com",

"pass": "NecesitoUn5ParaAprobar"}

JSON es el lenguaje de programación más popular, ya que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje, a pesar de que su nombre indique lo contrario.

Los encabezados y los parámetros también son importantes en los métodos HTTP de una solicitud HTTP de la API de RESTful, ya que contienen información de identificación importante con respecto a los metadatos, la autorización,…

Hay unos criterios que debe seguir la API para que sea RESTfull.

4.1. Criterios API RESTfull

- Arquitectura cliente-servidor compuesta de clientes, servidores y recursos, con la gestión de solicitudes a través de HTTP.

- Comunicación entre el cliente y el servidor sin estado, lo cual implica que la información del cliente no se almacena entre las solicitudes de GET y que cada una de ellas es independiente y está desconectada del resto.

- Datos que pueden almacenarse en caché y optimizan las interacciones entre el cliente y el servidor.

- Una interfaz uniforme entre los elementos, para que la información se transfiera de forma estandarizada.

- Un sistema en capas que organiza en jerarquías invisibles para el cliente cada uno de los servidores que participan en la recuperación de la información solicitada.

- Código disponible según se solicite (opcional), es decir, la capacidad de enviar códigos ejecutables del servidor al cliente cuando se requiera, lo cual amplía las funciones del cliente.

4.2. Funciones API RESTful

Las API RESTful se basan en funciones principales:

- GET: Es el tipo más simple de método de solicitud HTTP. Los datos nunca deben ser modificados en el lado del servidor como resultado de una solicitud. En este sentido, una petición GET es de sólo lectura, pero por supuesto, una vez que el cliente recibe los datos, es libre de hacer cualquier operación con ella por su cuenta, por ejemplo, formatearla para su visualización. Por ejemplo:
  - GET -> https://localhost:3100/api/coche. Esto es un ejemplo de una ruta get que he usado en este proyecto.

- PUT: Utilizada cuando se desea actualizar un recurso. Las peticiones contienen los datos que se utilizarán para actualizar. Si por ejemplo queremos actualizar un coche concreto:
  - PUT -> https://localhost:3100/api/coche/6073123123112 . Esto sería la ruta de un put si queremos actualizar el coche con id: 6073123123112. En el body del cuerpo tendremos que tener todos los datos del coche y cambiaremos el dato o datos que queramos cambiar.

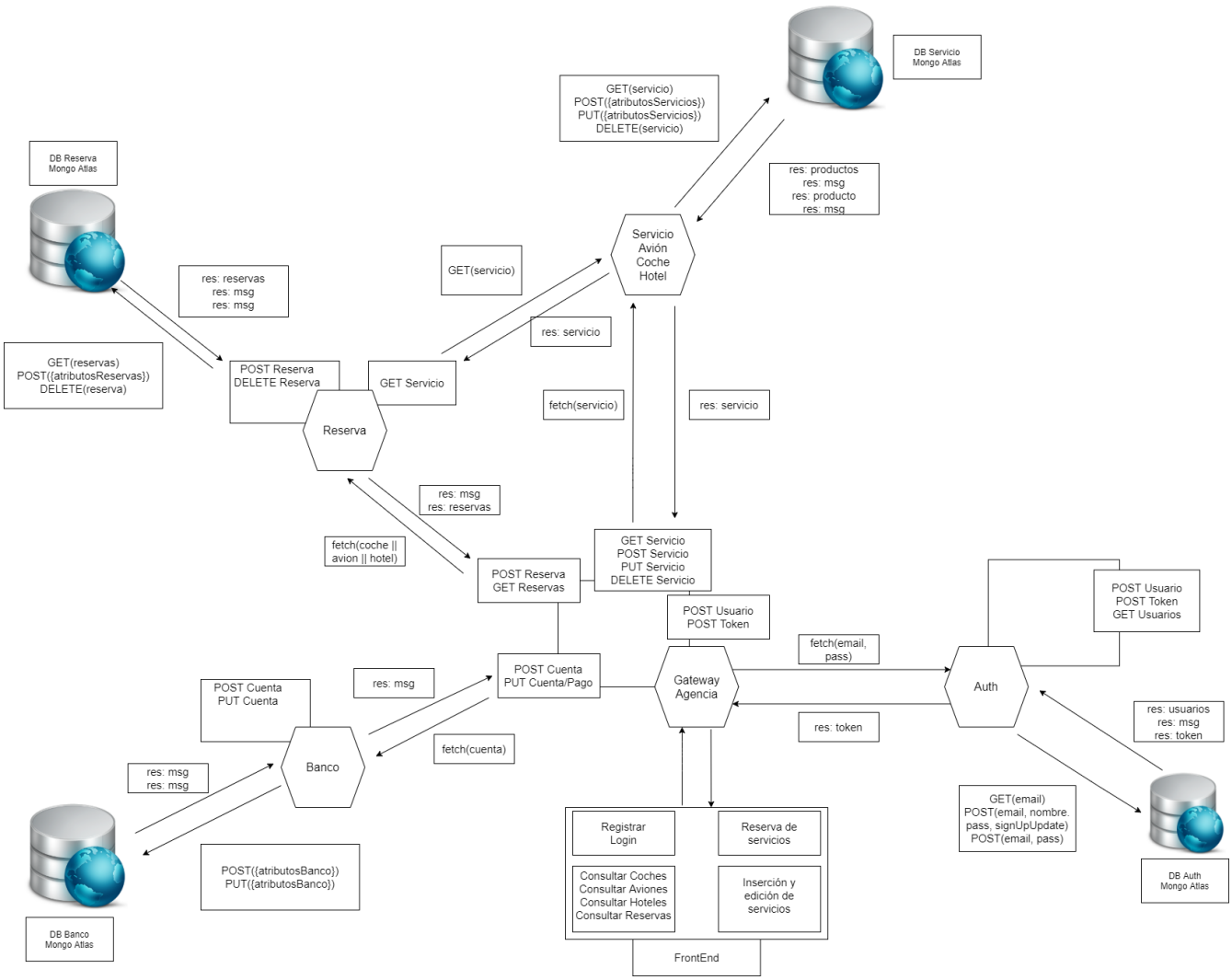
- DELETE: Debe utilizarse cuando desee eliminar el recurso identificado por la solicitud. Por ejemplo para borrar un coche concreto:
  - DELETE -> https://localhost:3100/api/coche/6073123123112.

- POST: Utilizada cuando se desea crear un recurso. Muy parecido al PUT en cuanto que hay que rellenar un body con los parámetros a introducir. Por ejemplo:
    - POST -> https://localhost:3100/api/coche. Suponiendo que queramos introducir un coche nuevo en nuestra base de datos, pondríamos en el body los atributos de la clase coche que queramos que se guarden en la base de datos.

4.3. Arquitectura Técnico-Conceptual

Esta es la arquitectura del proyecto que se ha implementado. Como se puede ver es necesario siempre estar autorizado para poder hacer cualquier cosa, desde ver un simple servicio hasta hacer una reserva de uno y también que exceptuando el servicio de agencia, cada uno tiene su base de datos.

Cabe destacar un punto importante y es que se han implementado 2 tecnologías, la primera una arquitectura SOA, que utiliza el protocolo HTTP y otra una cliente-servidor para la comunicación de los micro servicios y sus bases de datos.



5. Servicios de la Agencia.

A continuación se va a explicar los distintos servicios y módulos que encontramos en el proyecto así como sus distintas funciones que se pueden hacer con ellas. Entre los distintos módulos y servicios encontramos: Coche, Avion, Hotel, Auth, Reserva, Banco y la Agencia. Tanto el servicio Coche, Hotel y Avión son muy parecidos y sus funciones son similares, por lo que los vamos a explicar juntos.

5.1. Coche/Hotel/Avión.

En cuanto a los tres servicios se componen cada uno por los siguientes atributos:

Coche = {

modelo: String, matricula: String, destino: String, fechaIni: String, fechaFin: String, disponible : {type: Boolean, default: true}, precio : {type : Number, default: -1}

}

Avión = {

asiento: String, destino: String, fechaIni: String, fechaFin: String, disponible : {type: Boolean, default: true}, precio : {type : Number, default: -1}

}

Hotel = {

nombre: String, direccion: String, numHabitacion: String, categoria: String, destino: String, fechaIni: String, fechaFin: String, disponible : {type: Boolean, default: true}, precio : {type : Number, default: -1}

}

Estos servicios se encargan de darnos los productos que necesitamos, los cuales podrán ser reservados por los clientes en el caso de estar disponibles. Estos servicios serán vistos a través de la agencia y todos ellos tendrán una seguridad de auntenticación, por lo que es necesario loggearse para poder acceder a alguno de ellos.

Las tres tienen su propia base de datos donde se guardarán los productos que ofrece. Esta base de datos se irá modificando conforme funciones vayamos haciendo.

5.2. Auth

Este servicio es el de Autorización. No dará las herramientas para encriptar las contraseñas protegiendo la privacidad de los clientes.

La estructura de los usuarios será la siguiente:

Auth = {

email:{type:String, unique:true, index:true, lowercase:true}, nombreComp: String, pass: String, signUpdate: {type: Date, default: Date.now()}

}

En este módulo tendremos las funciones para crear los usuarios y para iniciar sesión, esto no es más que la creación y verificación de los tokens, el cual, es necesario para poder acceder a las funciones de la Agencia que accederá a su vez a los servicios de Coches/Hoteles/Aviones. Los tokens los generamos con la librería de 'jwt-simple', esto nos permitirá cerciorarnos de que solo nuestra agencia pueda generarlos.

Se ha implementado también una diferenciación entre roles, esto quiere decir que tendremos un usuario Administrador, el cual será el que añada, edite o borre los servicios de la base de datos. Hablaremos más de él en el frontend.

5.3. Reserva

Es el encargado de las Reservas de nuestro MEAN y estarán las funciones de creación de Reservas, borrado,… En las reservas vamos a guardar los siguientes datos:

Reserva = {

emailUser: String, modeloCoche: String, matricula:String, habitacionHotel: String, nombreHotel:String, asientoAvion: String, desAvi: String

}

El cliente mediante el fronend, por supuesto, podrá reservar cualquier servicio que se encuentre disponible siempre que se cumplan las condiciones necesarias:

- Que ya tenga una cuenta bancaria (En caso negativo, se le redirigirá para que se cree una)
- Que el saldo bancario le llegue para pagar los servicios que contrate.
- Que los servicios que vaya a contratar estén disponibles.

EL usuario también podrá cancelar una reserva si fuese necesario. Para las reservas se usado un patrón saga coreografiado, para tener un mayor orden en la reserva. En este tipo de arquitectura, lo que se busca es que todos los servicios implementados, puedan trabajar de una manera independiente, que es uno de los problemas que existe en la arquitectura de orquestación.

Por lo general, cada servicio no requiere ninguna instrucción para que funcione. Se busca un sistema descentralizado, que en muchas ocasiones, pueda funcionar por eventos, y los servicios estarán subscritos a esos eventos. Este tipo de aproximación es conocido también como arquitectura reactiva, la cual nos permite una arquitectura reactiva.

5.4. Banco

Este servicio será el encargado de las funciones que cobre o registren el saldo de los usuarios. El banco estará compuesto por:

Banco = {

nombreBanco: String, emailUser: String, numTarjeta: String, saldo: String

}

El usuario se comunicará con el banco para dos cosas:

- En caso de no tener una cuenta bancaria registrada, a la hora de reservar, el cliente tendrá que introducir sus datos bancarios. Esto solo lo hará una vez, la primera.

- En caso de tener ya la cuenta registrada, el usuario se comunicará con el banco de manera automática a la hora de reservar un servicio o varios.

El banco solo tiene las funciones de crear la cuenta, editar el saldo de la propia cuenta y mostrarla.

5.5. Agencia

Este para mi parecer, es el servicio más importante pues todo pasa por este servicio. Es el servicio de Agencia o Gateway el encargado de comunicar el frontend con todos los demás microservicios. Es el encargado de poder reservar, introducir un nuevo coche (Solo el administrador),…

Es importante saber que aquí es donde aplicaremos la seguridad del sistema. En las funciones importantes como POST o PUT introduciremos seguridad de autenticación para que necesites estar loggeado para modificar o introducir un nuevo producto a un servicio.

También cabe destacar que si este servicio cae, el sistema será completamente inútil, pues no podrá hacerse ni gets, ni post, ni reservas ni nada.

5.6. Tabla Funciones API y end-points

Función HTTP	Ruta	Descripción	Autenticación	ADMIN
GET	/api/coche	Devuelve una lista con todos los coches guardados en la Base de Datos	NO	NO
GET	/api/coche/:id	Devuelve un solo coche en concreto	NO	NO
GET	/api/avion	Devuelve una lista con todos los aviones guardados en la Base de Datos	NO	NO
GET	/api/avion/:id	Devuelve un solo avión en concreto	NO	NO
GET	/api/hotel	Devuelve una lista con todos los hoteles guardados en la Base de Datos	NO	NO
GET	/api/hotel/:id	Devuelve un solo hotel en concreto	NO	NO
GET	/api/auth/usuarios	Devuelve una lista con todos los usuarios guardados en la Base de Datos	SI	SI
GET	/api/auth/usuarios/:id	Devuelve un solo usuario en concreto	SI	SI
GET	/api/cuenta	Devuelve una lista con todas las cuentas guardados en la Base de Datos	SI	SI
GET	/api/cuenta/:id	Devuelve una sola cuenta en concreto	SI	NO
GET	/api/reservas/	Devuelve una lista con todas los reservas guardados en la Base de Datos	SI	SI
GET	/api/reservas/:id	Devuelve un sola reserva en concreto	SI	NO

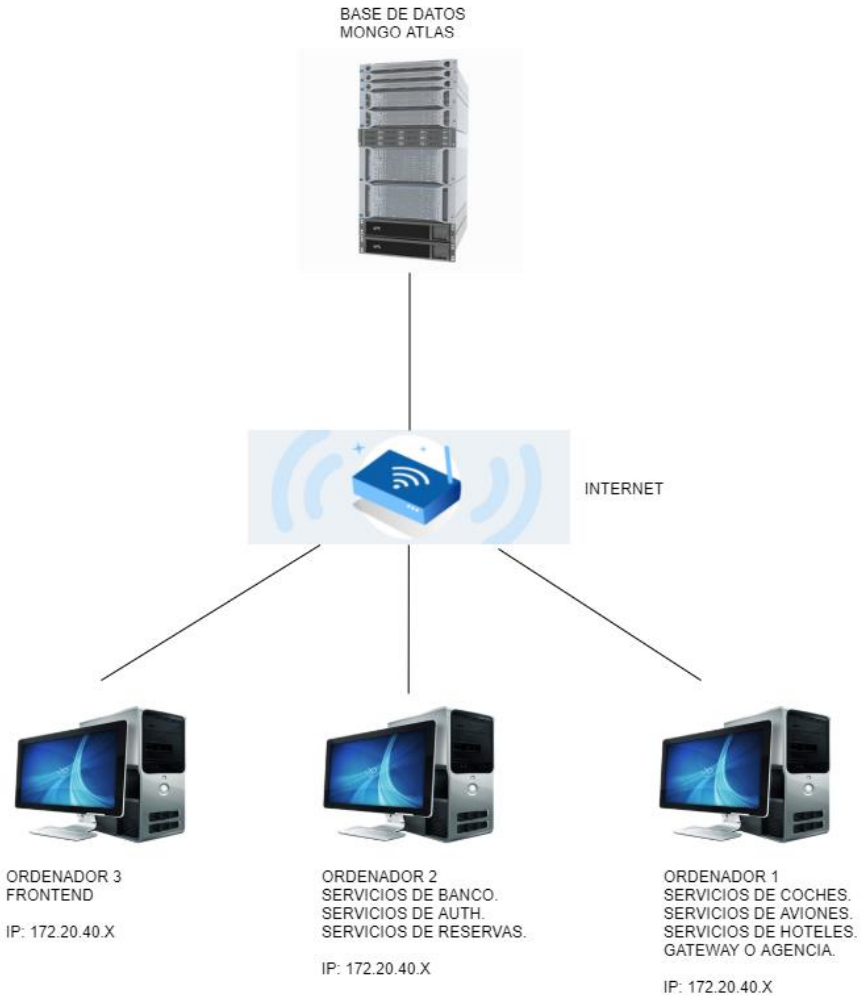
Función HTTP	Ruta	Descripción	Autenticación	ADMIN
POST	/api/coche	Permite registrar un coche en la Base de Datos	SI	SI
POST	/api/avion	Permite registrar un avión en la Base de Datos	SI	SI
POST	/api/hotel	Permite registrar un hotel en la Base de Datos	SI	SI
POST	/api/auth/usuarios	Permite registrar un usuario en la Base de Datos	NO	NO
POST	/api/cuenta	Permite registrar un cuenta en la Base de Datos	SI	NO
POST	/api/reservas	Permite registrar una reserva en la Base de Datos	SI	NO
POST	/api/auth/tokens	Permite recuperar un token introduciendo un usuario y su contraseña ya registradas en la BBDD.	NO	NO

Función HTTP	Ruta	Descripción	Autenticación	ADMIN
PUT	/api/coche/:id	Permite modificar un coche en la Base de Datos	SI	SI
PUT	/api/avion/:id	Permite modificar un avión en la Base de Datos	SI	SI
PUT	/api/hotel/:id	Permite modificar un hotel en la Base de Datos	SI	SI
PUT	/api/usuarios/:id	Permite modificar un usuario en la Base de Datos	SI	SI
PUT	/api/cuenta/:id	Permite modificar un cuenta en la Base de Datos	SI	SI

Función HTTP	Ruta	Descripción	Autenticación	ADMIN
DELETE	/api/coche/:id	Permite borrar un coche de la Base de Datos	SI	SI
DELETE	/api/avion/:id	Permite borrar un avión de la Base de Datos	SI	SI
DELETE	/api/hotel/:id	Permite borrar un hotel de la Base de Datos	SI	SI
DELETE	/api/auth/usuarios/:id	Permite borrar un usuario de la Base de Datos	SI	SI
DELETE	/api/cuenta/:id	Permite borrar un cuenta de la Base de Datos	SI	SI
DELETE	/api/reservas/:id	Permite borrar una reserva de la Base de Datos	SI	SI

6. Guía para Despliegue.

Para la guía de despliegue se distribuirán los distintos servicios que tenemos en tres ordenadores.



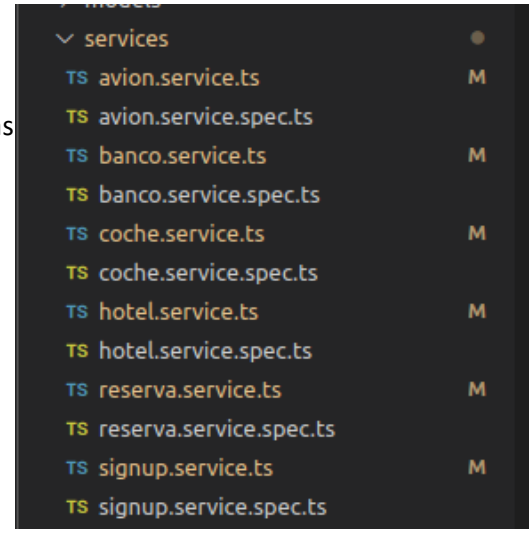
- En el primer ordenador se desplegarán los servicios de Coche – Avión – Hotel – Agencia. Como la Agencia está en el mismo equipo que los estos microservicios, sus rutas se quedarán como localhost para mayor efectividad y menor probabilidad de error con las rutas. Esto se cambiará en el fichero config de la Agencia:

```
config.js
servercode > Agencia > config.js > <unknown>
1   const PORT = process.env.PORT || 3100;
2
3   const URL_Avion = "https://localhost:3000/api/avion";
4   const URL_Coche = "https://localhost:3006/api/coche";
5   const URL_Hotel = "https://localhost:3002/api/hotel";
6   const URL_Auth =  "https://localhost:3004/api/auth/usuarios";
7   const URL_Token=  "https://localhost:3004/api/auth/tokens";
8   const URL_Reserva= "https://localhost:3007/api/reservas";
9   const URL_Banco=    "https://localhost:3008/api/cuenta";
10
11  SECRET_TOKEN = 'francisco8';
12
13
14  module.exports = { PORT, URL_Coche, URL_Avion, URL_Hotel, URL_Auth, SECRET_TOKEN, URL_Token,URL_Reserva,URL_Banco }
```

Para el resto de rutas se pondrá en lugar de localhost, la ip del ordenador donde se desplieguen. Para saber la ip, solo usar “ip a” en la terminal y ponerlo en el fichero de configuración.

- En un segundo ordenador se desplegarán los servicios de Banco – Auth y Reserva. Para acceder a estos servicios será necesario acceder a la Agencia primero, por lo que las rutas modificadas solo serán las que están en la agencia.

- Por último en un tercer ordenador se abrirá el frontend creado en angular. Aquí se tendrán que cambiar todas las ip’s de los ficheros servicios.ts. Todas las rutas van a la Agencia, por lo que la ruta que pondremos será la del ordenador de la Agencia.



7. Bibliografía.

Atlassian. (s. f.). *Qué es Git: conviértete en todo un experto en Git con esta guía*. Recuperado 16 de julio de 2021, de https://www.atlassian.com/es/git/tutorials/what-is-git

Calle, N. R. (2021, 16 enero). *Orquestación vs Coreografía en microservicios*. Refactorizando. https://refactorizando.com/orquestacion-vs-coreografia-microservicios/

colaboradores de Wikipedia. (2021a, julio 13). *Mozilla Firefox*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Mozilla_Firefox

colaboradores de Wikipedia. (2021b, julio 13). *Visual Studio Code*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Visual_Studio_Code

Devs, Q. (2019, 17 septiembre). *¿Qué es Angular y para qué sirve?* Quality Devs. https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/

Fernández, Y. (2019, 23 agosto). *API: qué es y para qué sirve*. Xataka. https://www.xataka.com/basics/api-que-sirve

Fischer, L. (2016, 4 diciembre). *Guía para principiantes de HTTP y REST*. Code Envato Tuts+. https://code.tutsplus.com/es/tutorials/a-beginners-guide-to-http-and-rest--net-16340

López, A. (2021, 26 febrero). *Qué es Postman y para qué sirve*. OpenWebinars.net. https://openwebinars.net/blog/que-es-postman/

Lucas, J. (2020, 1 junio). *Qué es NodeJS y para qué sirve*. OpenWebinars.net. https://openwebinars.net/blog/que-es-nodejs/

MongoDB. (s. f.). *La base de datos líder del mercado para aplicaciones modernas*. Recuperado 16 de julio de 2021, de https://www.mongodb.com/es

¿Qué es una API de REST? (s. f.). RED HAT. Recuperado 16 de julio de 2021, de https://www.redhat.com/es/topics/api/what-is-a-rest-api