Prácticas de Sistemas Distribuidos

Creando una Agencia de Viajes

WS, RESTful, CRUD, MEAN, JSON

FRANCISCO JAVIER PÉREZ MARTÍNEZ

74384305M – Turno: lunes 9:00-11:00

Contenido

Introducción

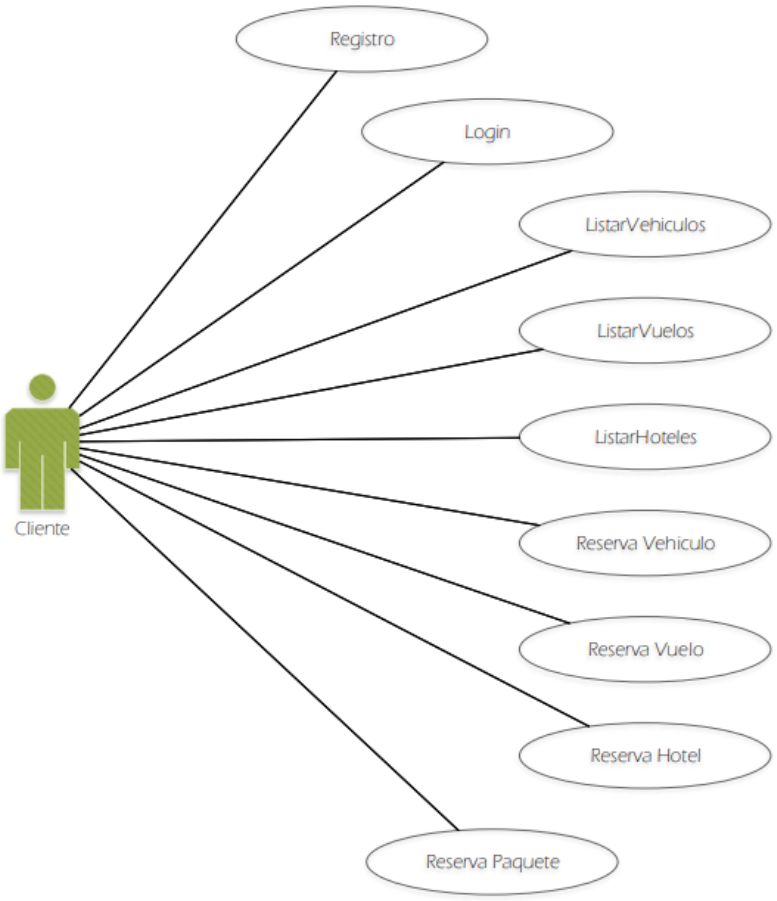
En esta práctica, trabajaremos con un tipo de arquitectura de desarrollo web la cual se apoya en el estándar de HTTP denominada API REST empleando servicios web restful. En esta arquitectura, las llamadas al API se implementan como peticiones HTTP, en las que básicamente, la URL representa el recurso y los HTTP Verbs la operación.

Para ello, desarrollaremos y diseñaremos un sistema distribuido que permita la gestión de una agencia de viajes en el que los clientes puedan reservar de forma conjunta o individual los diferentes servicios (hoteles, vehiculos y/o vuelos). Estas reservas serán proporcionadas por unos proveedores que conforman los diferentes servicios y gestionadas a través de la agencia.

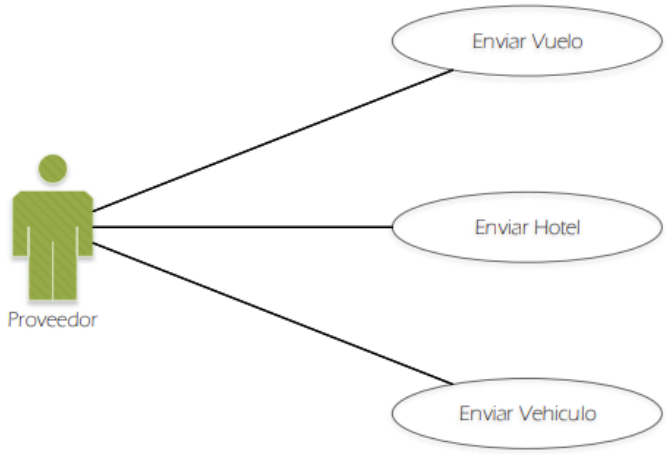
Casos de uso

En el apartado de casos de uso nos encontramos con las distintas funcionalidades de un cliente y un proveedor las cuales posteriormente explicaré haciendo uso de los end-points para mayor comodidad.

Cliente

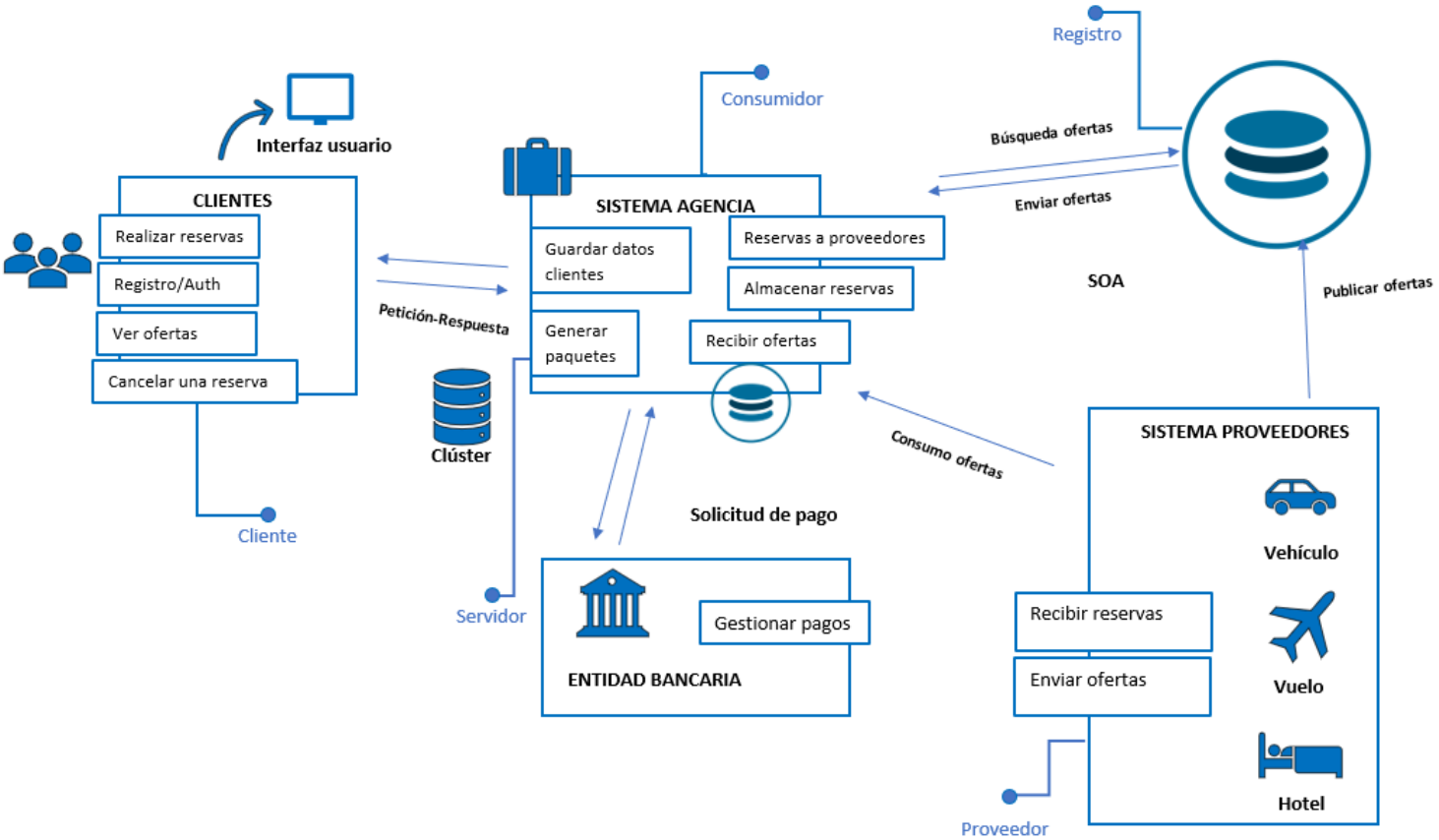


Proveedor



Arquitectura conceptual

En el siguiente esquema se muestra la arquitectura conceptual de nuestro proyecto:



Como podemos observar, se muestran los principales actores que intervienen como son los clientes, el sistema de agencia, el sistema de proveedores y el sistema de bancos.

En cuanto a las funcionalidades de cada actor:

- Los clientes, mediante una interfaz podrán identificarse (Iniciar Sesión o Registro), visualizar los tipos de ofertas y posteriormente realizar una reserva o cancelarla.

- La agencia, recibe las ofertas por parte de los proveedores que posteriormente se mostrarán en la agencia y los clientes podrán realizar una reserva. Además, la agencia deberá almacenar los datos de los clientes, así como sus reservas.

- Los proveedores, se limitan a enviar ofertas y recibir las reservas realizadas por los clientes, dichas reservas son enviadas por la agencia.

- Entidad bancaria, la cual proporciona la interfaz B2B para la gestión de pagos.

En cuanto a los modelos arquitectónicos seguidos:

- Entre los clientes y la agencia, se conectarán mediante C/S ya que el acoplamiento es directo, fuerte y con gran dependencia. Además, tendremos una simple llamada al servidor para que proporcione recursos, típico de las arquitecturas webs.

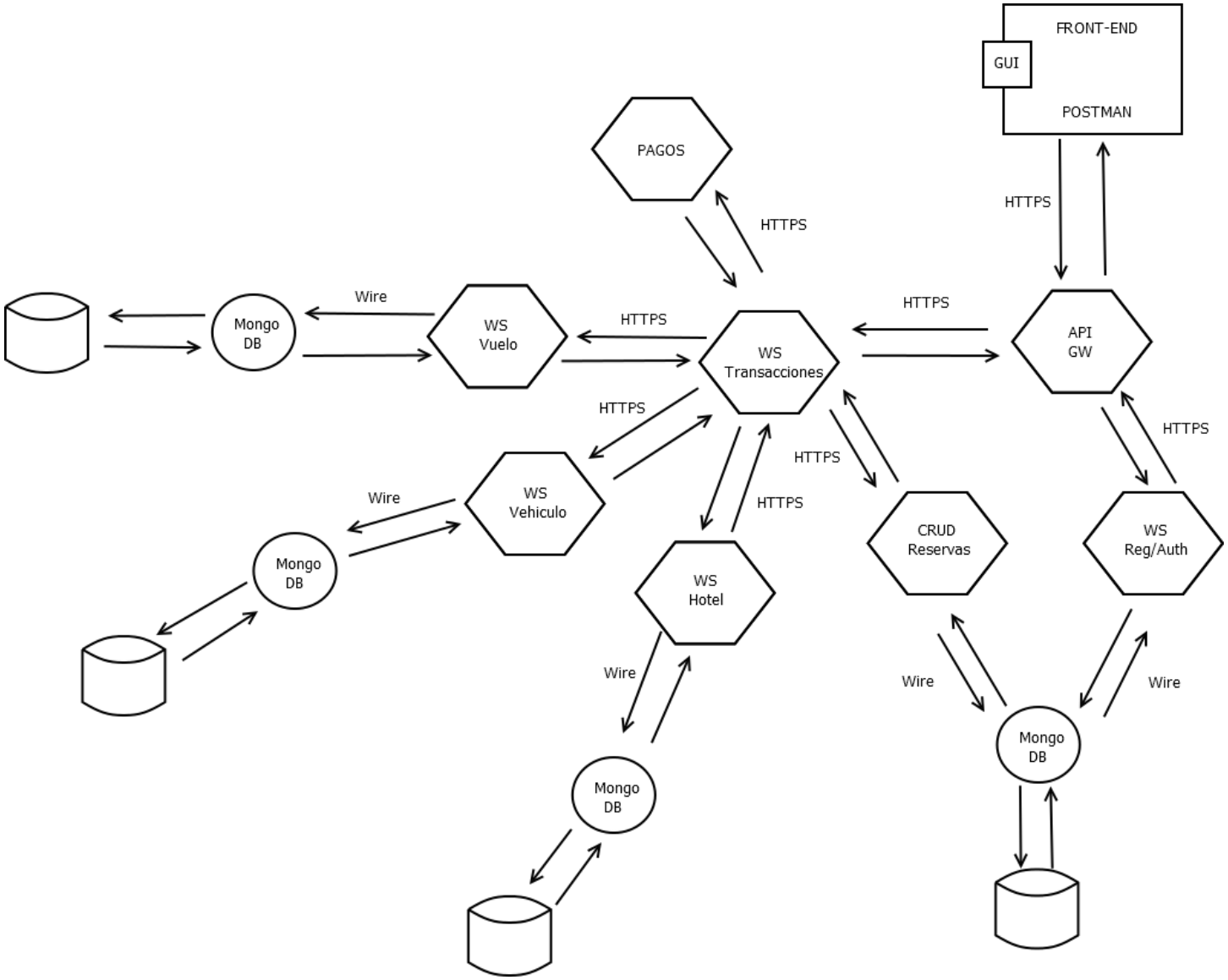
- Entre la agencia y los proveedores, he decidido implementar una arquitectura tipo SOA debido a que principalmente esta arquitectura está orientada a servicios y los proveedores envían ofertas a un registro intermedio y este mismo debe ser el que usará la agencia para poder obtener estas ofertas y mostrarlas para que los clientes procedan a realizar una reserva. Además, se ha empleado este tipo de arquitectura debido a que necesitamos interoperabilidad.

- Entre agencia y bancos, se conectarán también mediante C/S, de forma que cuando se precise realizar un pago se mostrará la interfaz proporcionada por la entidad bancaria correspondiente. Esta comunicación se basa en solicitud y respuesta ya que la agencia solicitará a la entidad bancaria el proceso de gestión de un pago.

Arquitectura técnica conceptual

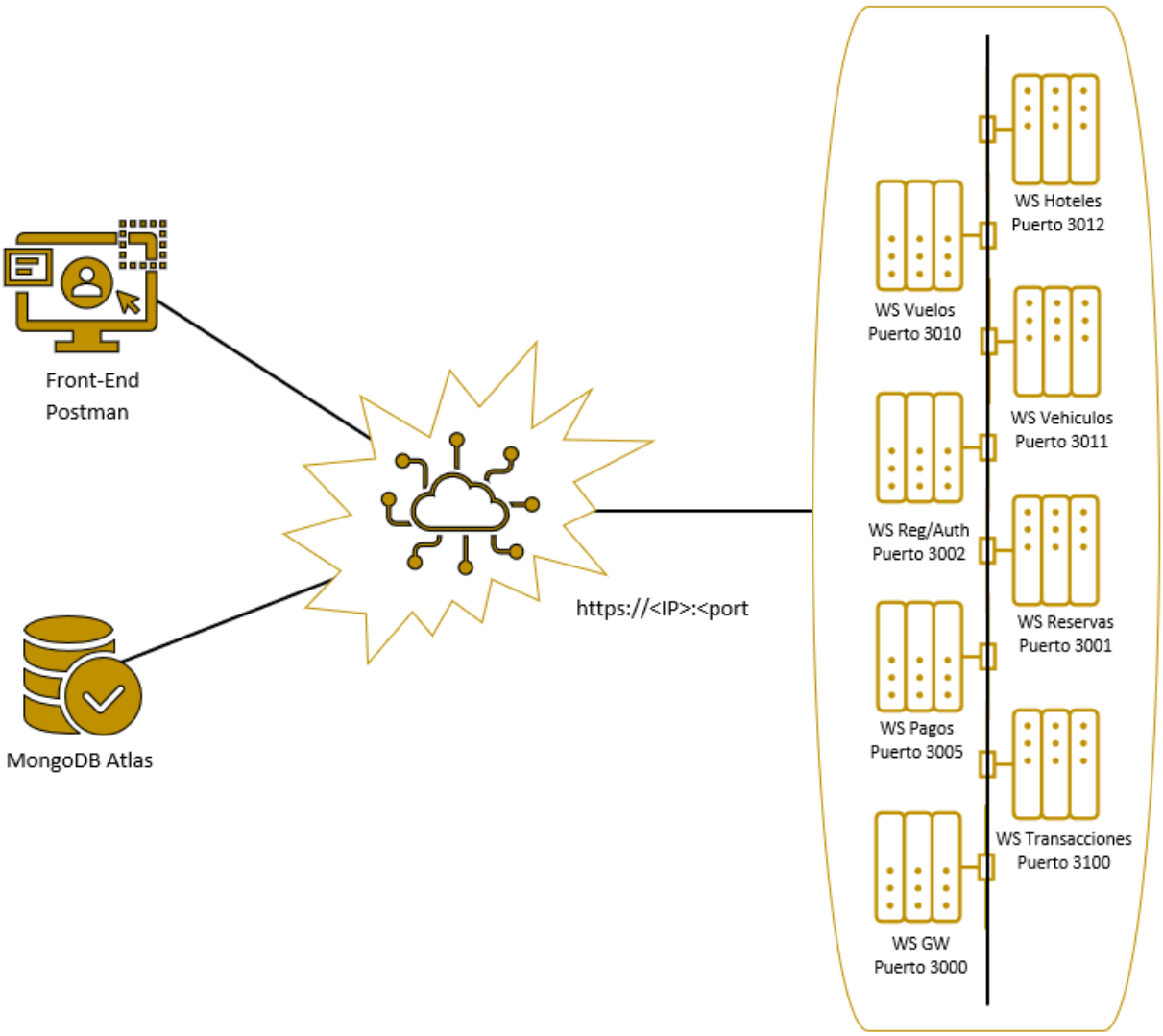
En cuanto al esquema de arquitectura técnica conceptual, nos profundizaremos mas en las tecnologías empleadas en nuestro sistema iniciando con la comunicación entre los diferentes módulos que lo componen en la que empleamos el protocolo seguro de transferencia HTTPS.

Más en profundidad, se trata de un conjunto de tecnologías que nos ayudan en el diseño e implementación de aplicaciones distribuidas basadas en microservicios distribuidos por Internet, también conocidas como pila MEAN. Para ello, se emplearán tecnologías MEAN: Node + Express como servidor HTTP, y el gesto de DB Mongo haciendo uso de varias bibliotecas para facilitar la redacción de nuestro código. Por otro lado, nos encontramos con Angular para el front-end, en mi caso me he centrado más en back-end y utilizando Postman como herramienta de testeo.



Arquitectura de despliegue

En cuanto a la arquitectura de despliegue de nuestro sistema, existen diferentes combinaciones que se podrían plantear, pero en mi caso utilizaremos la muestro a continuación:



Resaltar que para la base de datos se ha utilizado el servicio de base de datos en la nube de MongoDB Atlas para externalizar así este servicio y evitar problemas de despliegue en los laboratorios.

Servicios (end-points)

Primer de todo, presentaré una tabla resumen con los servicios propuestos:

Servicio	Tipo	Breve descripción
Proveedor Vehículos	WS tipo REST	Permite gestionar las reservas de vehículos.
Proveedor Vuelos	WS tipo REST	Permite gestionar las reservas de vuelos.
Proveedor Hoteles	WS tipo REST	Permite gestionar las reservas de hoteles.
Registro	WS tipo REST	Permite gestionar los usuarios del servicio
Autenticación	WS tipo REST	Proporciona un sistema de autenticación para los usuarios
CRUD Reservas	WS tipo REST	Conector con la DB, almacenamos las reservas.
Transacciones	WS tipo REST	Gestiona las transacciones
Pago	WS tipo REST	Gestiona los pagos con las entidades bancarias y de crédito
Gateway	WS tipo REST	Gestiona las peticiones de los clientes.

En cuanto a los end-points, destacaremos en primer lugar los situados en el API Gateway, módulo que actúa de pasarela entre los clientes y la agencia, éstos coinciden con las funcionalidades del cliente de forma que obtendremos los mostrados a continuación:

Verbo HTTP	Ruta	Descripción
GET	/api/listarVuelos	Obtenemos una lista de todas las ofertas de vuelos.
GET	/api/listarVehículos	Obtenemos una lista de todas las ofertas de vehículos.
GET	/api/listarHoteles	Obtenemos una lista de todas las ofertas de hoteles.
POST	/api/reservar	Realiza una reserva ya sea conjunta o individual. Necesaria autorización con token.
DELETE	/api/reservar	Cancela una reserva ya sea conjunta o individual. Necesaria autorización con token.
POST	/api/auth/login	Login de usuario generando su token de sesión asociado.
POST	/api/auth/signup	Registro de usuario.

GET	/api/listarVuelos	Obtenemos una lista de todas las ofertas de vuelos.

Solicitud (HTTP Request):

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Parámetros (Parameters):

Cuerpo (Body):

```
{

}
```

Respuestas (HTTP Response):

Código de estado:

200 OK

Cabeceras (Headers):

```
Content-Type: "application/json"
...
```

Cuerpo (Body):

```
{
  Result: 'OK',
  Coleccion: 'vuelos'
  Elementos: [

  ]
}
```

GET	/api/listarVehiculos	Obtenemos una lista de todas las ofertas de vehiculos.

Solicitud (HTTP Request):

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Parámetros (Parameters):

Cuerpo (Body):

```
{

}
```

Respuestas (HTTP Response):

Código de estado:

200 OK

Cabeceras (Headers):

```
Content-Type: "application/json"
...
```

Cuerpo (Body):

```
{
  Result: 'OK',
  Coleccion: 'vehiculos'
  Elementos: [

  ]
}
```

GET	/api/listarHoteles	Obtenemos una lista de todas las ofertas de hoteles.

Solicitud (HTTP Request):

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Parámetros (Parameters):

Cuerpo (Body):

```
{

}
```

Respuestas (HTTP Response):

Código de estado:

```
200 OK
```

Cabeceras (Headers):

```
Content-Type: "application/json"
...
```

Cuerpo (Body):

```
{
  Result: 'OK',
  Coleccion: 'hoteles'
  Elementos: [

  ]
}
```

POST	/api/reservar	Realiza una reserva ya sea conjunta o individual. Necesaria autorización con token.

Solicitud (HTTP Request):

Cabeceras (Headers):

```
Content-Type: "application/json"
Authoritation: bearer <<ACCESS_TOKEN>>
```

Parámetros (Parameters):

Cuerpo (Body):

```
{
   "vuelo": "5fe8a6e5acfc7124561ca2d5",
    "hotel": "5fe8a376acfc712456165c47",
    "vehiculo": "5fe8a3d5acfc71245616e2ae"
}
```

Solicitud (HTTP Request):

Cabeceras (Headers):

```
Content-Type: "application/json"
Authoritation: bearer <<ACCESS_TOKEN>>
```

Parámetros (Parameters):

Cuerpo (Body):

```
{
   "vuelo": null,
    "hotel": null,
    "vehiculo": "5fe8a3d5acfc71245616e2ae"
}
```

Respuestas (HTTP Response):

Código de estado:

```
200 Created
```

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Cuerpo (Body):

```
{
 "estadoTransaccion": "OK",
 "reservaVuelo": ...,
 "reservaVehiculo": ...,
 "reservaHotel": ...,
 "pago": true
  ...
}
```

Respuestas (HTTP Response):

Código de estado:

```
400 Bad request
```

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Cuerpo (Body):

```
{
      "estadoTransaccion": "abortada"
      ...
}
```

Ejemplo práctico:

Si intentamos realizar una reserva y el WS de pagos se encuentra caído, nos daría un fallo al intentar conectarse con la pasarela de pago y procedería a anular las reservas dadas, anulando así la transacción.


```
{
    "result": {
        "estadoTransaccion": "Abortada",
        "reservaVuelo": {},
        "reservaVehiculo": {},
        "reservaHotel": {},
        "Entidad_Bancaria": {
            "message": "request to https://localhost:3005/api/payment failed, reason: connect ECONNREFUSED 127.0.0.1:3005",
            "type": "system",
            "errno": "ECONNREFUSED",
            "code": "ECONNREFUSED"
        },
        "Error_de_conexion?": true,
        "Error_al_reservar?": false,
        "Mensaje": "Fallo al intentar conectarse con la pasarela de pago, transacción abortada",
        "Compensacion_de_vuelo?": "La reserva del vuelo ha sido anulada",
        "Compensacion_de_vehiculo?": "La reserva del vehiculo ha sido anulada",
        "Compensacion_de_hotel?": "La reserva del hotel ha sido anulada"
    }
```


Al igual que con el WS de pagos se ha comprobado con los diferentes proveedores.

Ejemplo de salida correcta de una reserva conjunta:


```
{
    "result": {
        "estadoTransaccion": "OK",
        "reservaVuelo": {
            "compañia": "Vueling Airlines",
            "fecha_ida": "27/12/2020",
            "fecha_vuelta": "03/01/2020",
            "origen": "Bourg-en-Bresse",
            "destino": "Gniezno",
            "precio": "€1093,91",
            "reservado": true,
            "usuario": "email1234@gmail.com"
        },
        "reservaVehiculo": {
            "marca": "Lexus",
            "modelo": "LX",
            "ciudad": "Shelopugino",
            "fecha_ida": "07/01/2020",
            "fecha_vuelta": "07/01/2020",
            "precio": "€59,78",
            "reservado": true,
            "usuario": "email1234@gmail.com"
        },
        "reservaHotel": {
            "hotel": "Grand Hotel Villa Santorini",
            "ciudad": "Sonta",
            "fecha_entrada": "22/01/2020",
            "fecha_salida": "30/01/2020",
            "precio": "€119,59",
            "reservado": true,
            "usuario": "email1234@gmail.com"
        },
        "Entidad_Bancaria": {
            "pago": true,
            "message": "Pago realizado"
        },
        "Error_de_conexion?": false,
        "Error_al_reservar?": false,
        "Compensacion_de_vuelo?": "Nada que compensar",
        "Compensacion_de_vehiculo?": "Nada que compensar",
        "Compensacion_de_hotel?": "Nada que compensar"
    }
}
```

DELETE	/api/reservar	Cancela una reserva ya sea conjunta o individual. Necesaria autorización con token.

Solicitud (HTTP Request):

Cabeceras (Headers):

```
Content-Type: "application/json"
Authoritation: bearer <<ACCESS_TOKEN>>
```

Parámetros (Parameters):

Cuerpo (Body):

```
{
   "vuelo": "5fe8a6e5acfc7124561ca2d5",
    "hotel": "5fe8a376acfc712456165c47",
    "vehiculo": "5fe8a3d5acfc71245616e2ae"
}
```

Solicitud (HTTP Request):

Cabeceras (Headers):

```
Content-Type: "application/json"
Authoritation: bearer <<ACCESS_TOKEN>>
```

Parámetros (Parameters):

Cuerpo (Body):

```
{
   "vuelo": null,
    "hotel": null,
    "vehiculo": "5fe8a3d5acfc71245616e2ae"
}
```

Respuestas (HTTP Response):

Código de estado:

```
200 Created
```

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Cuerpo (Body):

```
{
 "estadoTransaccion": "OK",
 "reservaVuelo": ...,
 "reservaVehiculo": ...,
 "reservaHotel": ...,
 "pago": true
  ...
}
```

Respuestas (HTTP Response):

Código de estado:

```
400 Bad request
```

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Cuerpo (Body):

```
{
      "estadoTransaccion": "abortada"
      ...
}
```

POST	/api/auth/login	Login de usuario generando su token de sesión asociado.

Solicitud (HTTP Request):

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Parámetros (Parameters):

Cuerpo (Body):

```
{
    "email": "email1234@gmail.com"
    "password": "1111"
}
```

Respuestas (HTTP Response):

Código de estado:

```
201 Created
```

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Cuerpo (Body):

```
{
  "result": "OK",
  "message": "Login realizado con éxito",
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.xxxxx
}
```

Respuestas (HTTP Response):

Código de estado:

```
400 Bad request
```

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Cuerpo (Body):

```
{
  "result": "Error",
  "message": "El usuario no existe"
}
```

Respuestas (HTTP Response):

Código de estado:

```
400 Bad request
```

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Cuerpo (Body):

```
{
  "result": "Error",
  "message": "Contraseña incorrecta"
}
```

POST	/api/auth/signup	Registro de usuario.

Solicitud (HTTP Request):

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Parámetros (Parameters):

Cuerpo (Body):

```
{
   "usuario": "Javi"
   "email": "email1234@gmail.com"
   "password": "1111"
}
```

Respuestas (HTTP Response):

Código de estado:

```
201 Created
```

Cabeceras (Headers):

```
Content-Type: "application/json"
```

Cuerpo (Body):

```
{
  "result": "OK",
  "coleccion": "users",
  "elemento": {
      "_id": "5fe9c0622a58221cf66ad842",
      "usuario": "Javi",
      "email": email1234@gmail.com
      "password": "$2b$10$pXsicrH.xLRqBxAzEwSBy.xxxxx ",
      "signUpDate": 1609154657,
      "lastLoginDate": "Thursday, December 30, 2020 12:12 AM"
  }
}
```

Respuestas (HTTP Response):

Código de estado:

```
400 Bad request
```

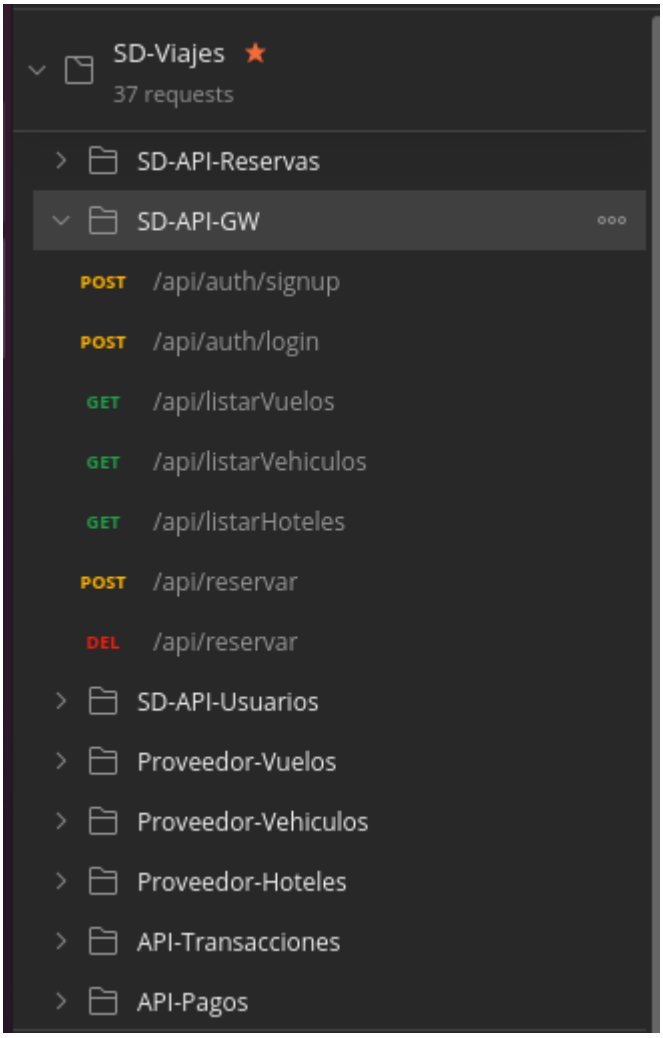
Cabeceras (Headers):

```
Content-Type: "application/json"
```

Cuerpo (Body):

```
{
  "result": "Error",
  "message": "El email ya existe"
}
```

Para la redacción de estos end-points ha resultado más sencillo gracias a Postman, ya que esta herramienta nos permite testear todas nuestras API Rest y almacenar nuestras operaciones tal que así:



Web Services

Nuestro sistema está compuesto por diferentes módulos o Web Services tipo REST los cuales describiré a continuación.

WS Gateway

Este es el módulo que nos ayuda a conectar entre un cliente y nuestra colección de microservicios. Este Gateway actúa como un proxy inverso para aceptar todas las llamadas a la interfaz de programación de aplicaciones (API), agregar los diversos servicios necesarios para cumplirlas y devolver el resultado apropiado.

En nuestro sistema, está conectado con el WS de transacciones y con el de registro y autenticación.

WS Transacciones

Esta API está conectada directamente con nuestro API Gateway, pagos, los 3 proveedores (vuelos, vehículos y hoteles) y el CRUD de reservas. Aplicando una lógica de negocio realizamos las reservas pertinentes ya sea de forma individual o en paquete haciendo uso de todas estas APIs mencionadas.

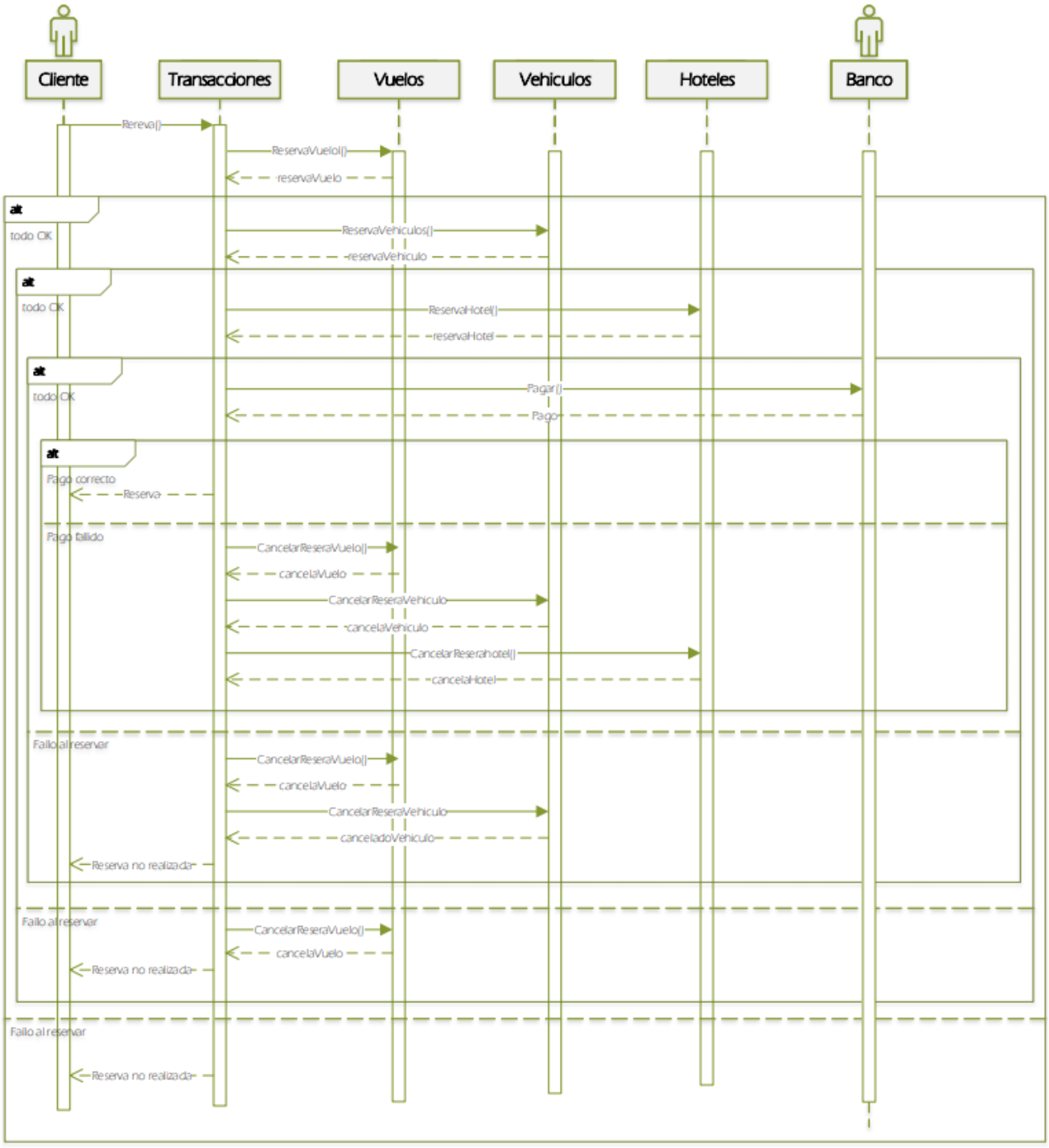
Destacaremos 2 tipos de transacciones, las concurrentes y las distribuidas.

Las concurrentes se refieren a cuando existe un problema de concurrencia en el que dos clientes intentan reservar el mismo producto a la vez. Esto se puede resolver aplicando un cerrojo, este cerrojo bloqueará cualquier acción realizada sobre él para poder evitar así cualquier inconsistencia de datos. Además, para evitar problemas de inanición, añadimos marcas de tiempo para poder realizar los registros de todo lo que va ocurriendo e ir añadiendo dichas marcas mediante FIFO para la sincronización y servir las reservas por orden de llegada. En nuestra práctica, todo esto está solucionado ya que, al emplear el servicio de MongoDB, éste SGBD usa un bloqueo de granularidad múltiple que permite que las operaciones se bloqueen a nivel global, de base de datos o colección permitiendo así que los motores de almacenamiento individuales implementen su propio control de concurrencia por debajo del nivel de colección (como por ejemplo un objeto en concreto de reserva). Para ello, utilizaremos WiredTiger en el que mediante un control de simultaneidad optimista nos permite establecer a qué nivel queremos realizar el bloqueo. Este motor es el predeterminado de MongoDB en el que podemos ponerlo en marcha mediante la siguiente línea de comandos:

```
mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
```

En cuanto a las transacciones distribuidas, un ejemplo sería en el que cuando un cliente esté intentando realizar una reserva y por ejemplo el servidor del banco correspondiente haya caído tendríamos que anular toda la reserva que hemos hecho.

En el siguiente ejemplo de diagrama, se encuentra el paso de mensajes al realizar una reserva en el que intervienen los 3 proveedores y el de pagos con el API de transacciones:



Como podemos observar, cuando se produce algún fallo inesperado o una caída de un servicio, existe una acción compensatoria para deshacer el cambio que ha hecho la transacción. Este proceso corresponde al 2PC con el mecanismo de patrón de saga de tipo orquestación, el cual se identificaría con el WS de transacciones.

WS Reg/Auth

Este WS se identifica con los dos end-points de registro y login el cual va conectado al Gateway para que los clientes puedan acceder y así poder realizar las diferentes funcionalidades que poseen haciendo uso de una autenticación.

En esta API destacaremos los diferentes frentes de seguridad a la hora de poner en producción nuestro sistema de una forma mínimamente segura:

- Uso de canales de comunicación seguros con HTTPS sobre SSL/TLS y el uso de certificados digitales.
- Mantenimiento de una traza de todo lo que ocurre mediante un Logger con morgan.
- Uso del paquete helmet para revisar y evitar diferentes vulnerabilidades conocidas.
- Uso de estrategia de registro, identificación y autorización basada en Access Token sin estado empleando un hash, salt, JWT, OAuth.

WS Proveedores

Ya que prácticamente los 3 WS que componen el sistema de proveedores son idénticos, los explicaré en este apartado en conjunto. Estos proveedores son los encargados de enviar las ofertas a la agencia mediante dos llamadas, una para reservar y otra para cancelar la reserva. Estas dos llamadas ya contemplan los casos de reserva individual o conjunta. Una vez realizado un envío de reserva cuando se ha realizado la transacción correspondiente se le envía a dicho proveedor los datos necesarios de los clientes de dicha reserva.

En nuestro proyecto, serían las carpetas de api-proveedor-vuelos, api-proveedor-vehículos y api-proveedor-hoteles.

WS Pagos

Para simular el sistema de pagos se ha realizado una operación GET generando un valor random en el que el 80% de las veces devolverá que el pago ha sido realizado correctamente y el 20% que no.

Por mi parte, no he decidido almacenar los pagos en una BD ya que he supuesto que esto es realizado por la correspondiente entidad bancaria.

WS CRUD Reservas

Este WS sería el encargado de recibir las reservas del módulo de transacciones que nos ayudaría a modo de histórico, tanto al cliente como a la agencia, saber que reservas ha realizado un cliente almacenando éstas en una base de datos con los datos de toda la reserva en conjunto. En caso de producirse una cancelación de reserva, ésta sería eliminada de la base de datos. Este WS no lo tengo implementado por lo que intentaré tenerlo para el día del despliegue.

Conclusiones

A modo de resumen, en esta práctica se ha empleado 8 WS que formarían el conjunto de back-end, todos estos se encuentran conectados mediante una comunicación segura como es HTTPS. Además, destacamos el API Gateway ya que es el encargado de gestionar todas las peticiones del front-end y el método de transacciones que será el encargado de orquestar los diferentes servicios como bien se ha explicado anteriormente.

Como conclusión, una vez finalizado el proyecto puedo decir que ya tengo las ideas mucho más claras en comparación con las primeras entregas de las arquitecturas. Ahora ya entiendo como se conectan todos los microservicios entre ellos, cosa que al principio no. Es cierto que, el proyecto no está acabado en su totalidad ya que me faltan algunas cosas como el front-end y el WS mencionado, pero ya ha sido por falta de tiempo más que nada. No obstante, no descarto seguir investigando todo lo relacionado con sistemas distribuidos ya que en mi opinión, la asignatura me ha resultado bastante interesante debido a que para el desarrollo de estos sistemas, las tecnologías que hemos empleado se encuentran entre las más punteras e innovadoras.

Entrega

Repositorio:

El repositorio se encuentra privado y le he enviado una invitación como colaborador al proyecto con el siguiente enlace:

https://github.com/Javiiiis/proyecto-rest-sd

Aclaraciones para la puesta en marcha del proyecto:

El proyecto consta de 8 carpetas, todas con el prefijo api-, correspondiendo cada una a los WS mencionados. Para su funcionamiento, extraemos el código fuente desde nuestro repositorio mediante el commando: git clone https://github.com/Javiiiis/proyecto-rest-sd

Una vez clonado, ejecutamos npm i, para obtener la carpeta node_modules, y npm start para arrancar el servicio en cada api.

Las bases de datos se encuentran configuradas en la nube por lo que si se quiere ejecutar el código, éste funcionaria perfectamente ya que se encuentra con el acceso de red para todas las direcciones 0.0.0.0/0.

Para tener acceso a los datos desde la web de Mongo Atlas le he enviado una invitación de proyecto al correo del anuncio. Mi correo es fjpm26@gcloud.ua.es.

En el servicio de registro y autenticación, tenemos el usuario “Javi” en el que bastaría con el correo email1234@gmail.com y la contraseña “1111” para poder obtener el token de sesión y así poder usar las distintas funcionalidades del usuario en nuestra gateway.