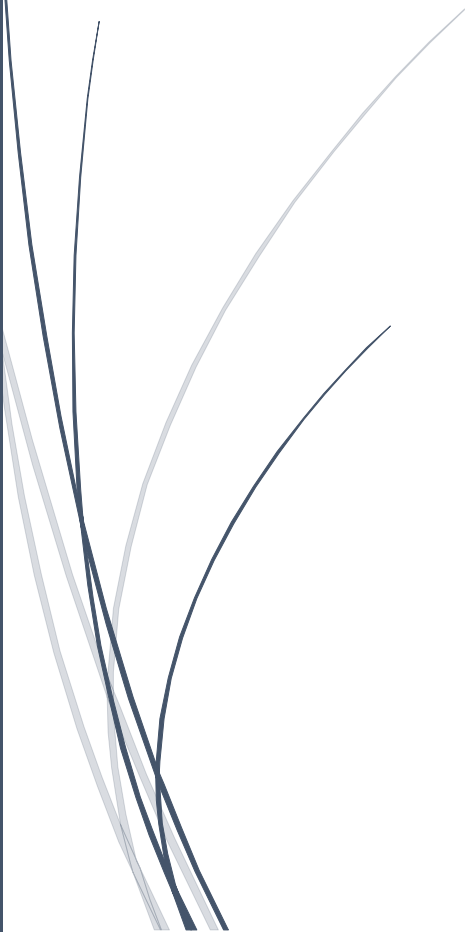


A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

13-7-2021

# Sistemas Distribuidos

Proyecto de prácticas

Several thin, curved lines in shades of blue and grey originate from the bottom left and sweep upwards and to the right.

Javier Martínez Molina  
20096647Y

# INDICE

<b>1. Introducción .....</b>	<b>2</b>
<b>2. Proveedores .....</b>	<b>4</b>
<b>3. Agencia .....</b>	<b>15</b>
<b>4. Banco .....</b>	<b>38</b>
<b>5. Front-End Desacoplado .....</b>	<b>41</b>
<b>6. Cumplimiento de requisitos .....</b>	<b>56</b>
<b>7. Despliegue .....</b>	<b>57</b>
<b>8. Bibliografía .....</b>	<b>58</b>

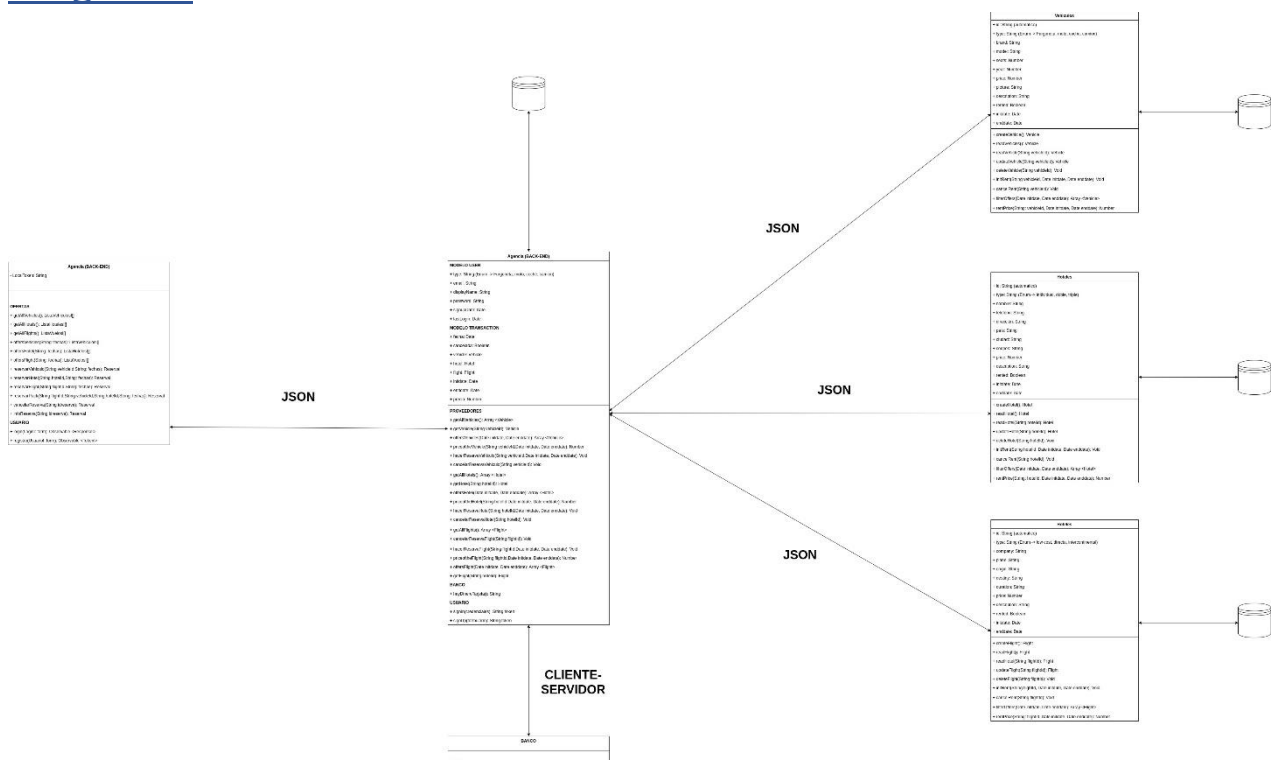
# Introducción

Para explicar de forma más organizada y concreta como he implementado la práctica y la ampliación de julio me gustaría comentar primero el diagrama de clases.

El proyecto lo he desarrollado en su mayoría con JavaScript con Nodejs a excepción de la parte del front-end desacoplado, en este caso he utilizado angular, en lenguaje TypeScript.

Como tecnologías de interconexión he utilizado servicios REST dentro de las api principales, y como segunda tecnología he utilizado un modelo Cliente-Servidor a través de un socket TCP en la conexión del Banco a la Agencia, he decidido hacerlo así por la simplicidad del banco en el sistema, pero esto lo explicaré más adelante.

## Diagrama



Como podemos ver se pueden distinguir 6 componentes en el sistema, en la parte derecha podemos ver a los proveedores que se encargarán de la comunicación directa con la base de datos, por otra parte, tenemos la agencia que actúa como componente principal de interconexión consumiendo las funcionalidades de los proveedores y sirviendo de servidor para el front-end donde los clientes podrán acceder a las funcionalidades principales de la aplicación, pero nunca podrán acceder a información o funcionalidades que puedan romper el sistema, como los métodos Create / Delete / Update de la agencia o proveedores.

Por último, tenemos el componente del Banco, es el más sencillo del sistema, se conecta con la agencia y su función es dar el ok a las transacciones que se puedan generar.

## Versiones y características

Versión de nodejs instalada: v14.17.2

Versión de npm instalada: 6.14.13

Version de angular CLI instalada: 12.1.0

OS: linux x64

Estas versiones son compatibles entre sí.

Comandos para la creación de los proyectos.

npm init --> inicia un proyecto de nodejs

Npm i -D nodemon --> instala nodemon (librería que permite lanzar el servidor cada vez que se produce un cambio)

Service mongod start --> lanza la instancia de mongodb

Npm i -S mongoose --> instala mongoose (librería que corre por encima de mongodb y permite gestionar y conectar de forma más sencilla la bbdd al proyecto de nodejs)

Npm run seed --> lanza los seeder y rellena la bbdd

Npm i -S bcrypt-nodejs --> instala bcrypt (librería para el encriptado de datos)

Npm i -save jwt-simple --> instala jwt (librería para utilizar json web token)

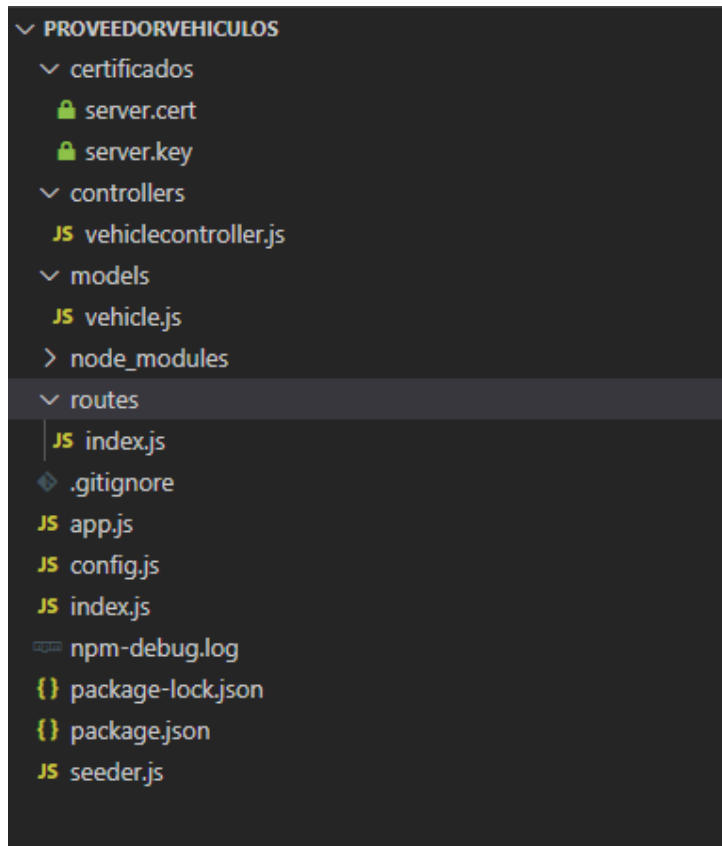
Npm i -save moment --> instala moment (librería para gestionar fechas de forma más sencilla)

Npm i request --> instala request (librería que permite realizar peticiones desde una api)

Npm i -g @angular/cli --> instala el cliente de angular (las ultimas versiones entran en conflicto con nodejs, al instalarlo seguramente habría que cambiar de versión)

## Proveedores

Hay 3 proveedores dentro de la aplicación, uno para Vehículos, otro para Hoteles y por último uno para Vuelos. Como los 3 proyectos tienen unas características similares y son casi idénticos analizaré uno de ellos.



Aquí podemos ver la estructura del proyecto, está organizado de una forma sencilla, hay 3 carpetas, una para los certificados que sirven para cifrar el canal, aunque esto lo explicaré con más detalle después, otra carpeta donde se encuentra el controlador que conectará con la base de datos y por último dos carpetas más, una que contiene el modelo de datos que tendrán los vehículos de la aplicación y otra que contiene las rutas.

En los ficheros app.js , config.js y index.js se encuentra la información necesaria para el lanzamiento de la aplicación. En el fichero config almaceno el puerto como variable de entorno y la conexión a la BBDD también como variable de entorno, para levantar que cuando arranque mongodb en la máquina se cree el esquema dentro automáticamente.

```
1 module.exports =  
2 {  
3   port: process.env.PORT || 3005,  
4   db: process.env.MONGODB || 'mongodb://localhost:27017/vehiculos'  
5 }
```

La aplicación se levanta en el puerto 3005, y la instancia de mongo se conecta a través del puerto por defecto de lanzamiento de mongodb en el puerto 27017. Los proveedores de hoteles y vuelos se levantan en los puertos 3006 y 3007 respectivamente.

Dentro del fichero app.js se encuentran las configuraciones del proyecto de express.

```
JS app.js > ...
1  'use strict'
2
3  const express = require('express')
4  const bodyParser = require('body-parser')
5
6  const app = express()
7  const api = require('./routes')
8
9  app.use(bodyParser.urlencoded({extended: false}))
10 app.use(bodyParser.json())
11 app.use('/api', api)
12
13 module.exports = app
14 process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0"
```

En el fichero index.js se realiza la conexión con mongoose a la BBDD y la inicialización del servidor.

```
JS index.js > ...
1  'use strict'
2
3  const mongoose = require('mongoose')
4  const app = require('./app')
5  const config = require('./config')
6  const https = require('https')
7  const fs = require('fs')
8
9  mongoose.connect(config.db, {useNewUrlParser: true, useUnifiedTopology: true}, (err, res) =>
10 {
11   if (err)
12   {
13     return console.log(`Error al conectar a la bbdd: ${err}`)
14   }
15   console.log('Conexión a la bbdd establecida...')
16
17   https.createServer({
18     key: fs.readFileSync('./certificados/server.key'),
19     cert: fs.readFileSync('./certificados/server.cert')
20   }, app).listen(config.port, () =>
21   {
22     console.log(`API PROVEEDOR VEHICULOS CORRIENDO EN http://localhost:${config.port}`)
23   })
24 })
25
```

Para crear los métodos del controlador primero se ha de desarrollar el modelo de datos, he creado el modelo vehicle.js.

```
1  'use strict'
2
3  const mongoose = require('mongoose')
4  const Schema = mongoose.Schema
5
6  const VehicleSchema = Schema
7  ({
8    type: { type: String, enum: ['furgoneta', 'moto', 'coche', 'camion']},
9    brand: String,
10   model: String,
11   seats: Number,
12   year: Number,
13   price: {type: Number, default: 0},
14   description: String,
15   rented: Boolean,
16   initdate: Date,
17   enddate: Date
18 })
19
20 module.exports = mongoose.model('Vehicle', VehicleSchema)
```

Definir el esquema es fácil gracias a mongoose, simplemente se han de definir las variables que deseemos y el tipo de dato de estas, también podemos darle valores por defecto y características como que solo admitan unos datos concretos en ellas (enumerados), el id se genera automáticamente en la BBDD en la variable `_id`.

Aquí se declara la estructura de un vehículo, y una de las cosas que quiero destacar de los proveedores es que he creado un campo `rented` en ellos, que marca de forma booleana si se encuentran alquilados o no. Esto es una mala práctica imposible de llevar al mundo real ya que no se pueden alquilar en distintas fechas los elemento que ya se encuentren reservados.

Esto se podría solucionar, por ejemplo, utilizando listas de pares de datos (donde los elementos serían la identificación del vehículo y las fechas de reserva), esto no lo he llevado a cabo ya que sería demasiado complicado y considero que lo importante aquí es el despliegue e interconexión de las tecnologías por lo que he decidido implementar de forma más sencilla este aspecto.

Una vez creado el esquema de datos se puede exportar de manera sencilla con mongoose para así utilizar lo en el controlador como vamos a ver a continuación.

```
1  'use strict'
2
3  const Vehicle = require('../models/vehicle')
4  const moment = require('moment')
5  const { query } = require('express')
6
7  //Crea un vehiculo en la BBDD
8  function createVehicle(req, res)
9  {
10     console.log('POST /vehicle')
11     console.log(req.body)
12
13     let vehicle = new Vehicle()
14     vehicle.type = req.body.type
15     vehicle.brand = req.body.brand
16     vehicle.model = req.body.model
17     vehicle.seats = req.body.seats
18     vehicle.year = req.body.year
19     vehicle.price = req.body.price
20     vehicle.description = req.body.description
21     vehicle.rented = req.body.rented
22     vehicle.initdate = req.body.initdate
23     vehicle.enddate = req.body.enddate
24
25     vehicle.save((err, vehicleStored) =>
26     {
27         if (err) res.status(500).send({message: `Error al crear el vehiculo en la bbdd : ${err}`})
28
29         res.status(200).send(vehicleStored)
30     })
31 }
32
33 //Devuelve un JSON con todos los vehiculos
34 function readVehicles(req, res)
35 {
36     Vehicle.find({},(err, vehicles) =>
37     {
38         if (err) return res.status(500).send({message: `Error al realizar la lectura en la bbdd, petición incorrecta: ${err}`})
39         if (!vehicles) return res.status(404).send({message: `No hay vehiculos en la bbdd`})
40         res.status(200).send(vehicles)
41     })
42 }
43
```

Aquí se puede ver un pequeño fragmento del controlador de vehículos, donde se importa el esquema Vehicle que se utilizarán en los métodos que conectan con la BBDD. Los proveedores disponen de los métodos CRUD básicos y métodos extra que proporcionarán las funcionalidades de la aplicación. Estos métodos son:

**readVehicles** → Es un Read pero devuelve todos los vehículos en un array de JSON

**initRent** → Cambia el estado del objeto, poniendo el campo rented = true y cambia las fechas que se le pasan como parámetro

**cancelRent** → Cambia el estado del objeto, poniendo el campo rented = false y pone las fechas a null

**filterOffers** → Devuelve un array de JSON con todos los vehículos que no se encuentran reservados



**rentPrice** → Devuelve el precio de alquilar el objeto en las fechas pasadas como parámetro, en caso de los vuelos el precio es único y no varía en función de los días.

```
128 //Muestra las ofertas disponibles de vehiculos
129 async function filterOffers(req, res) // /:initdate/:enddate
130 {
131     const data = await Vehicle.find({rented: false}).exec();
132     res.status(200).send(data)
133 }
134
135
136 //Devuelve el precio total de alquilar ese vehiculo en las fechas dadas
137 async function rentPrice(req, res) // /:vehicleId/:initdate/:enddate
138 {
139
140     let vehicleId = req.params.vehicleId
141     var initdate = moment(req.params.initdate)
142     var enddate = moment(req.params.enddate)
143
144     var tiempo = enddate.diff(initdate, 'days')
145
146     if(tiempo == 0)
147         tiempo = 1
148
149     console.log('El tiempo a alquilar el vehiculo es ', tiempo, 'dias')
150
151     var elvehiculo = await Vehicle.find({_id:vehicleId}).select({"price": 1, "_id": 0})
152     var preciodia = elvehiculo[0].price
153
154     var preciototal = preciodia * tiempo
155
156     res.status(200).send({preciototal})
157 }
158
159
160 module.exports =
161 {
162     createVehicle,
163     readVehicles,
164     readVehicle,
165     updateVehicle,
166     deleteVehicle,
167     initRent,
168     cancelRent,
169     filterOffers,
170     rentPrice
171 }
```

Aquí se puede ver un ejemplo de los métodos filterOffers y rentPrice, quiero destacar que ha sido la primera vez que he trabajado con esta tecnología y con javascript y he tenido que aprender sobre el uso de funciones asíncronas y de su importancia al trabajar con consultas a las BBDD y lo importante que es para implementar un sistema transaccional consistente.

Para resumir brevemente, cuando declaramos una función async podemos utilizar la directiva await al realizar request, búsquedas, etc... estamos indicando que la función espere a recibir

una respuesta de los datos que requiere para así poder seguir con la función, si no realizamos este paso el código se seguiría ejecutando sin haber recibido los datos, dando lugar a errores.

En el controlador podemos elegir los módulos que serán exportados para así gestionar las rutas de la aplicación, esto lo realizo en el fichero ./routes/index.js

```
routes > JS index.js > ...
1  'use strict'
2
3  const express = require('express')
4  const vehicleCtrl = require('../controllers/vehiclecontroller') //controlador
5  const api = express.Router()
6
7  //CREATE
8  api.post('/vehicle', vehicleCtrl.createVehicle)
9
10 //READ COMPLETO
11 api.get('/vehicle', vehicleCtrl.readVehicles)
12
13 //READ UNICO
14 api.get('/vehicle/:vehicleId', vehicleCtrl.readVehicle)
15
16 //UPDATE
17 api.put('/vehicle/:vehicleId', vehicleCtrl.updateVehicle)
18
19 //DELETE
20 api.delete('/vehicle/:vehicleId', vehicleCtrl.deleteVehicle)
21
22 //HACER RESERVA
23 api.put('/vehicle/initrent/:vehicleId/:initdate/:enddate', vehicleCtrl.initRent)
24
25 //CANCELAR RESERVA
26 api.put('/vehicle/cancelrent/:vehicleId', vehicleCtrl.cancelRent)
27
28 //DEVUELVE EN UN ARRAY DE JSON LAS OFERTAS FILTRADAS
29 api.get('/vehicle/filtrarofertas/:initdate/:enddate', vehicleCtrl.filterOffers)
30
31 //PRECIO DEL VEHICULO EN LAS FECHAS PASADAS COMO PARAMETRO
32 api.get('/vehicle/precio/:vehicleId/:initdate/:enddate', vehicleCtrl.rentPrice)
33
34
35 module.exports = api
36 |
```

Podemos ver el fichero de rutas, primero destacar que para lanzar las rutas existe el elemento Router proporcionado por express, con él podemos declarar de forma sencilla rutas indicando el nombre de la variable al de la ruta declarada →

https://ip:puerto/nombrevARIABLERouter/rutametodo

ej. → <https://localhost:3005/api/vehicle>

Cuando se declara una ruta se ha de indicar el tipo de request que se hará, post = create, get = read, put = update, delete = delete. A continuación, solo hay que indicar la ruta deseada y el método del controlador donde se encuentra la lógica.

Respecto a la parte de seguridad, la gran mayoría de requerimientos los he implementado en la agencia (cifrado Salt) y autenticación con token (hacia el front), aunque si que he decidido realizar un cifrado del canal con certificados openssl entre la agencia y los proveedores, el certificado y la key están ubicados en la carpeta certificados, aunque más adelante en la agencia explicaré de forma más detallada como he implementado este cifrado en las comunicaciones.

```
certificados > server.key
1  -----BEGIN PRIVATE KEY-----
2  MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKwggSIAgEAAoIBAQC2TRpM9BUF7x13
3  kCAUoL2iTT8F3eJQ0z9jdjci/gz2QCAN+zaN7E+Iip9U7c5CHagWj8o1VciB00+5
4  ER6dTcKxh4+fGwmAxPn6JASWmZfkr3Q8QwePQGVdBYGy90bJXisaauko1SM77AW5
5  Zt0aB5L/17EQ9ai4MXmF+tu5j+nGxeVLoK1Cn+O+k8YoLavVNg2i1TQ0e8SA2PFP
6  m9etJpJraGEEYI5Q1zkMr8hiBYAXnuixiagQx0HN1TSJ0SMwTv0Wti/1nWk+hu1i
7  jXPYxojNP47Mm3WgQ0EDjNkXhIAw6f/EzPsK30KqJXwYKQvi5Y883Bu1hA15yUsp
8  VGmZIOjPAGMBAECggEBAJJq+O3HuL7sX1Fxa6LFu6mIu06d+LAR9gaPoxcyvzv
9  Yq14FEgG/ZNjzG7tFRVbNm7WjCspBSRyohsrJcRjodp5ruWfppzGoeCuEBE4MyG
10 8yeXepHe2crNqm8wViVtwFEU7NpQd10hYzOW1+tFnBZqiaeidyGBcyZd4TX7VXs0
11 iQD8MSb1k/pNurKL0CAA+bSmMfGA4WLVRD6coYB/8T9xyN2DU6yRgVbc8fJJUT5a
12 nBwjyqOYJnyR7go8GQ94EaHFYCI9S0fiaxbWyuCV2/D1MbGHru/iaox+GFPFI
13 sKqi4iSL7U27YbvRD1n79IF1QinJJM+61471q4LgyECgYEA5rf097Rxdn2x1S5zq
14 iF3suc7j6sbgxEB0n1ILjuVw7WcyjoiJo0+d/yTvmQa4JQcFw8g4tzQjeHPs3u
15 tqC66rqYBS19eSdSq7kqp5NpP8+ZsApdmG/ndJ/90ZPoxxySY5GVRg2RRLmJmgt
16 RMQVC4bfHPH4HL09CKghZbWmVf8CgYEAykcnHwjmXuxeyBM2Q61/Fw7xVFEpChm
17 aEMwPy30KVzigQ16vHn2xN0sH03mP1Do2VMntM6bw88fum4ZaofNBEd9h1Ex1C
18 99bPESfVVTZ8KekZp1LEHwIdUQSRFI4dU3fBEPzUV7QdMbe61yQX70WxfWuxGGno
19 4E66gMLWdTECgYEA1dAcByT7tMa4splWmF3q5dnWs3o0bM/+Kw7onKftqByjvCnZ
20 wJwVi0TQSmYlF+FX3x8rmYDfhMsvFD4G6bPzF60qXwE4wVVMp1+ESXLozEXzbQ
21 dfwqdx1AKfZScUfwbeJjPzSiXqsgDRHYeft8j5ojJJ0D9rayWznJh2P9C54MCgYEA
22 vMoyOFQwqzrJjthrwRJutxw15wtFQoAQoLyUufNi8gQGR2q8RhtMTNVQ0LRd4sGq
23 1mLrX4ov/J3c2Zrs8rMS3S5zTVzSkD98VHLBmoJyAvEyW696yQdfko1XrVkl1Go7
24 hr67jE5dJWou4UtyQsrLx/9ktFyuM1oPBuOUAD9ZB9ECgYAtMAJ8IpHqJ5Yyi1Xf
25 JWHmUPSPzo06i0EXuh/6INHISqK9txN7eFN1YVODRXnaRnPDW2mH6K7Mhw12DSPT
26 lZgZEKbyjSbvVEypgSstxpezjGb3swNqCpte357wh/4PJ4Xh3Jlbc8ZsKmnC9KZO
27 imyavApyXIFX8Wb0LIREQ9d1Nw==
28 -----END PRIVATE KEY-----

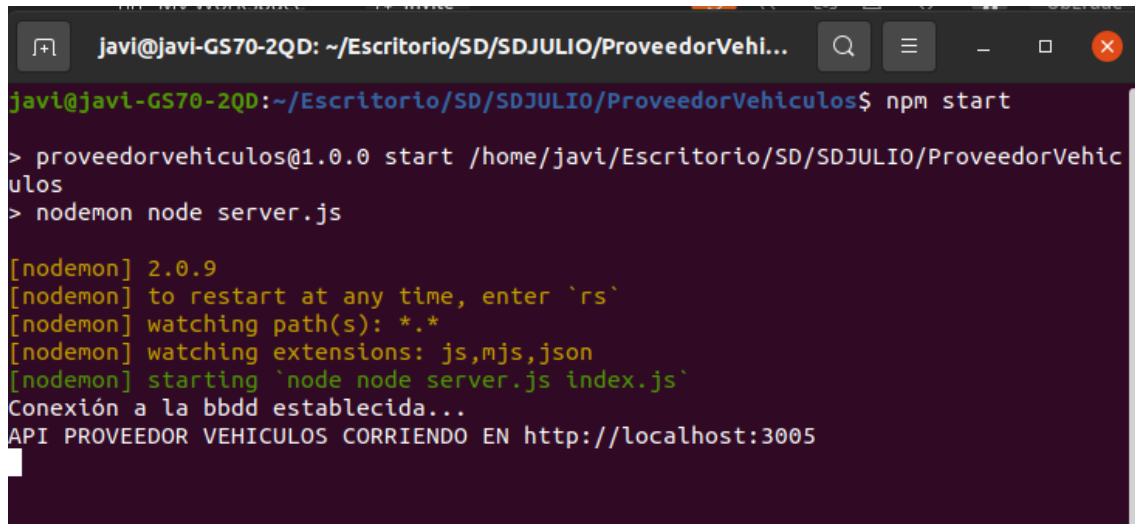
certificados > server.cert
1  -----BEGIN CERTIFICATE-----
2  MIID9zCCAt+gAwIBAgIUk0WRGsNfnFNoEP7b6S87m1+xrVQEWQYJkOZIHvcNAQEL
3  BQAwYoxCzAJBgNVBAYTAkVMTMREwDwYDVQQIDAhBbG1jYW50ZTEPMA0GA1UEBmVw
4  UGV0cmVyaWwMRQwEgYDVQQKDATKYXZpV2ViQXBwcEMMAoGA1UECwwDSlBMRITwEAYD
5  VQDDA1sb2NhbgHvc3QxHxAdBgkqhkiG9w0BCQEWEGptbTIyOUBhbHhUdWwEZXZMw
6  HhcNMjEwNjMwMjEzMTI2WmcNMjEwNzMwMjEzMjE2WjCBiEJLMakGA1UEBhMCRVMx
7  ETAPBgNVBAGMCFsaWVhbnRlMQ8wDQYDVQQHDAZQZXRYZXIxFDASBgNVBAoMC0ph
8  dm1XZwJBCHBzMQwwCgYDVQQLDANKV0ExEjAQBGNVBAMMChxvY2FsaG9zdDEfMB0G
9  CSqGS1b3DQEJARYQam1tMjI5QGfSdS51YS51c2CCASIdQYJkOZIHvcNAQEBBQAD
10 ggEPADCCAQoCggEBALZNGkz0FQXvHXeQIBSgvaJNPwXd41A7P2N2NyL+DPZATA37
11 No3sT4Uin1TtzkIdqBaPyjVvyJvQ77kRHp1NwrGHj58bCYDE+fokBJYzN+SvdDxD
12 B49AZV0Fgbl3RslKxq6qSiVizvsBb1m05oHkv/XsRD1qLgxeYX61TmP6cbF5Uug
13 qUKf476Txigtq9U2aLVNA57xIDY8U+b160mm0toYR5gjlCX0QyvyGIFgBee6LGJ
14 qBDE4c2VNIInRIzBO/Ra2L+WdYr6FTWKNc/LGiM0/jsybdaCrQQOM2ReEgDDp/8TM
15 +wrfQqo1f8gqq+L1jzzcG6WECXnJRKLuaZkg6M8CAwEAAANTMFEwHQYDVVR0OBBYE
16 FKtS7nIJSaY62CcG109KiZZIzGU7MB8GA1UdIwQYMBaAFKtS7nIJSaY62CcG109K
17 iZZIzGU7MA8GA1UdEwEB/wQFMAMBAf8wDQYJkOZIHvcNAQELBQADggEBAI1qtWnQ
18 b7sdtZv1qF0QRuBeway8lZ9ixmqe4aPcUrfnJwLfc1SwPubfjtJsj6U5279HVcm
19 mYL4kGf0zqgA6h21UAN7gWxmR/9TnBjsalCOFdugcRenVXHAW0ZBT6uEKBL+OS5
20 rPm+azmc6HTXcPz20B5SzFHDLMsfYR69YquEnvrLWlc2/ISG0URqn240YhdYKvB
21 ISA6GZXvUue34K0nnYkKwPe9LMh2Xj81YZ/bm+/m5sowTVh0xm/v1Kav+utXqLW
22 phThAVOCvuaIHWAavmLRpBC7p8RkWPdBJzu2OnaWBHG7NyWT4xx1RCqC9miZV1K
23 1uVXtXMNTHyo1gM=
24  -----END CERTIFICATE-----
25
```

Para la creación de datos y ahorro tiempo decidí crear seeder que permitan rellenar de forma más rápida la tabla de la base de datos, el seeder se encuentra en la raíz del proyecto y lo he implementado con la funcionalidad que proporciona mongoose-seed.

```
7 seeder.connect(config.db, function()
8 {
9   seeder.loadModels(["./models/vehicle.js"]);
10
11   seeder.clearModels(['Vehicle'], function()
12   {
13     seeder.populateModels(data, function (err, done)
14     {
15       if(err)
16       {
17         return console.log("Error al seedear bd", err)
18       }
19
20       if(done)
21       {
22         return console.log("Seeder correctos...", done)
23       }
24
25       seeder.disconnect()
26     })
27   });
28 });
29
30
31 var data = [
32
33   {
34     'model': 'Vehicle',
35     'documents':[
36       {
37         "price": "50",
38         "brand": "Ford",
39         "model": "Focus",
40         "seats": "5",
41         "type": "coche",
42         "year": "2008",
43         "description": "Gasolina 120cv",
44         "rented": "false",
45         "initdate": "2018-01-01",
46         "enddate": "2018-04-04"
47       },
48     ],
49   },
50 ]
```

## Pruebas con PostMan

Lo primero que implementé dentro del sistema fueron los proveedores y obviamente como no tenía la agencia ni el front creados para probar el correcto funcionamiento de estos realicé pruebas con la aplicación postman, las cuales vamos a ver de manera breve.

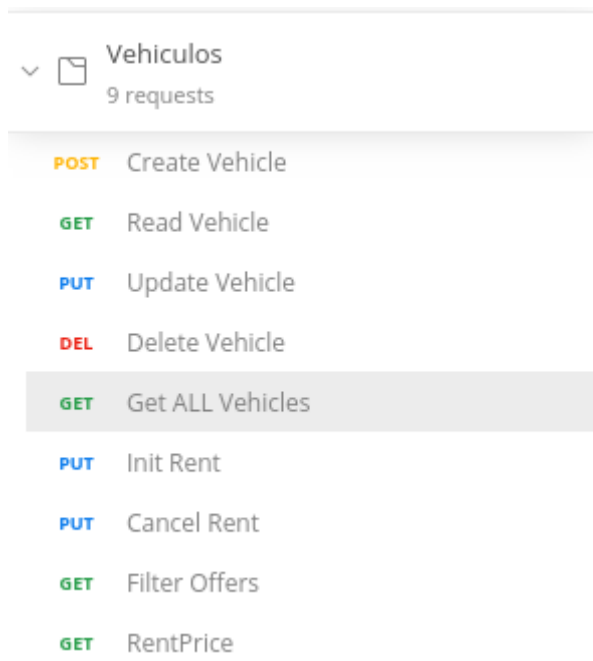


```
javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO/ProveedorVehi...
javi@javi-GS70-2QD:~/Escritorio/SD/SDJULIO/ProveedorVehiculos$ npm start

> proveedorvehiculos@1.0.0 start /home/javi/Escritorio/SD/SDJULIO/ProveedorVehiculos
> nodemon node server.js

[nodemon] 2.0.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node node server.js index.js`
Conexión a la bdd establecida...
API PROVEEDOR VEHICULOS CORRIENDO EN http://localhost:3005
```

Se lanza el proveedor



He realizado varias pruebas para los proveedores

GET Get ALL Vehicles
+
...

▶ Get ALL Vehicles

GET
https://localhost:3005/api/vehicle

Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings

Query Params

KEY	VALUE
-----	-------

Body
Cookies
Headers (7)
Test Results

Pretty
Raw
Preview
Visualize
JSON

```

1  [
2    {
3      "price": 399,
4      "_id": "60d8ef32a3df8692e0401bfa",
5      "brand": "Mercedes",
6      "model": "Vito",
7      "seats": 3,
8      "type": "coche",
9      "year": 2000,
10     "picture": "rutaford",
11     "description": "furgoneta de mercedes",
12     "rented": true,
13     "initdate": "2021-07-12T00:00:00.000Z",
14     "enddate": "2021-07-22T00:00:00.000Z",
15     "__v": 0
16   },
17   {
18     "price": 200,
19     "_id": "60d8ef32a3df8692e0401bfb",
20     "brand": "Ferrari",
21     "model": "LaFerrari",
22     "seats": 2,
23     "type": "coche",
24     "year": 2016,
25     "picture": "rutaFerrari",
26     "description": "Gasolina 800cv",
27     "rented": true,
28     "initdate": "2021-07-11T00:00:00.000Z",
29     "enddate": "2021-07-21T00:00:00.000Z",
30     "__v": 0
31   },
32 ]

```

Ejemplo de la request que devuelve el array de vehículos en formato JSON

Función que devuelve el precio a partir del id de un vehículo y las fechas

GET Get ALL Vehicles

GET RentPrice

X

+

...

▶ RentPrice

GET

▼

https://localhost:3005/api/vehicle/precio/60d8ef32a3df8692e0401bfa/2020-01-01/2020-01-05

ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettings

KEY	VALUE
Key	Value

BodyCookiesHeaders (7)Test Results

PrettyRawPreviewVisualize

JSON

▼

≡

1

{

2

"preciototal": 1596

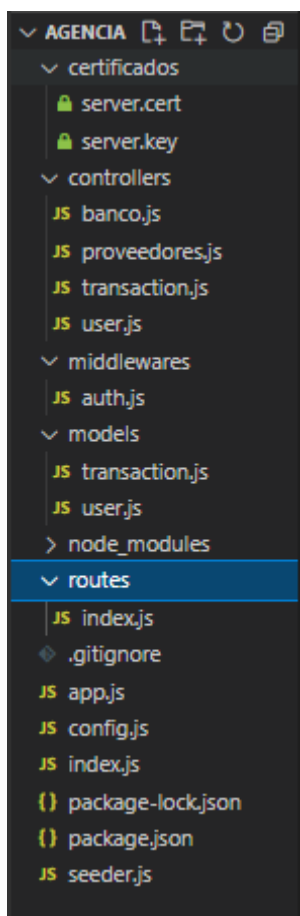
3

}

# Agencia

La agencia es el componente principal de la aplicación, ya que se interconecta con los demás elementos y sin ella no sería posible realizar las funcionalidades de la aplicación cara a los clientes. Para analizar el proyecto de la agencia lo haré por partes, primero veremos la estructura, y después profundizaré en los conceptos de seguridad y transacciones implementados.

## Estructura



La estructura es similar a primera vista comparado con la estructura de un proveedor, pero aquí se añaden bastantes funcionalidades. La configuración de express es igual que en los proveedores en este caso la agencia se despliega en el puerto 3008. Primero analizaré los modelos de datos.



## Modelos

Existen dos modelos en la agencia, uno que se encargará de gestionar el formato de los usuarios de la aplicación y otro que gestionará el formato de las transacciones.

El formato de los **usuarios** es el siguiente:

```
1  'use strict'
2
3  const mongoose = require('mongoose')
4  const Schema = mongoose.Schema
5  const bcrypt = require('bcrypt-nodejs')
6
7
8  const UserSchema = new Schema(
9  {
10   email: {type: String, unique: true, lowercase: true },
11   displayName: String,
12   password: {type: String, select: false},
13   signupDate: {type: Date, default: Date.now()},
14   lastLogin: Date
15 })
16
17
18 //antes de crear el modelo en la bbdd hashea la contraseña del usuario aplicando el concepto salt
19 UserSchema.pre('save', function (next)
20 {
21   if (!this.isModified('password')) return next()
22
23   bcrypt.genSalt(10, (err, salt) =>
24   {
25     if (err) return next(err)
26
27     bcrypt.hash(this.password, salt, null, (err, hash) =>
28     {
29       if (err) return next(err)
30       this.password = hash
31     })
32     next()
33   })
34 })
35
36
37
38 UserSchema.methods.comparePassword = function (candidatePassword, cb)
39 {
40   bcrypt.compare(candidatePassword, this.password, (err, isMatch) =>
41   {
42     cb(err, isMatch)
43   });
44 }
45
46
47
48 module.exports = mongoose.model('User', UserSchema)
49
50
```

En este caso además del schema que contendrá el email, nombre de usuario, contraseña y fecha de creación de la cuenta y lastlogin que ha realizado el usuario tenemos 2 funciones más.

La primera función `UserSchema.pre` es una función que se invoca cada vez que se genera un nuevo usuario, en ella implemento el concepto SALT de encriptado de contraseñas, esto lo hago con la ayuda de la librería `bcrypt` que proporciona `node`, en ella podemos generar un salt con la función `.genSalt(numsalt)`. Esta medida de seguridad lo que trata es de hashear la contraseña en cuanto el usuario se registra para que así en la BBDD no quede reflejada la contraseña real del usuario.

El hash es reforzado con un salt, que aumenta la dificultad de desencriptado de la contraseña siendo prácticamente imposible desencriptar la contraseña a la fuerza. Salt al fin y al cabo es información extra que se añade al hashear la contraseña para así aumentar la dificultad de desencriptado y así almacenar de forma segura y robusta las contraseñas. Las contraseñas antes de ser almacenadas en la BBDD pasan por este método que las hashea

Por último tenemos la función `comparePassword`, necesaria para comparar la contraseña que introducirá el usuario en el front-end con la que se encuentra hasheada en la bbdd, esta función se apoya en el método `.compare` que proporciona la librería `bcrypt`, es necesaria para gestionar el login de un usuario en mi caso, pero también si se desea por ejemplo realizar comprobaciones extra (ej. actualizar la contraseña en un formulario comprobando la contraseña actual).

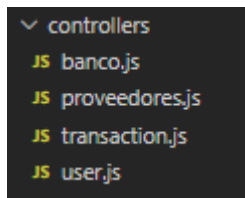
El otro modelo que he creado en la agencia es el de **transacciones**:

```
models > JS transactions > ...
1  'use strict'
2
3  const mongoose = require('mongoose')
4  const Schema = mongoose.Schema
5
6  const TransactionSchema = Schema(
7  {
8    fecha: {type: Date, default: Date.now()},
9    cancelada: {type:Boolean, default: false},
10   vehicle: {type: String, default: null},
11   hotel:{type: String, default: null},
12   flight:{type:String, default: null},
13   initdate:{type: Date},
14   enddate:{type:Date},
15   precio:{type:Number, default: 0}
16 })
17
18
19 module.exports = mongoose.model('Transaction', TransactionSchema)
20
21
```

Con este esquema puedo almacenar la información de las reservas que se produzcan, registro una marca temporal de cuando se ha producido, si ha sido cancelada o no, las variables vehicle hotel y flight permiten almacenar el id del objeto que se ha asociado a la reserva (en caso de ser una reserva individual los objetos no reservados serán null) y por último la fecha de inicio y fin de la reserva y el precio total de la misma, sumando el coste de alquilar los objetos en las fechas dadas.

## Controladores

Una vez vistos los modelos pasará a analizar los controladores que permiten gestionar la agencia.



### Banco

El **banco** es un controlador que analizaré en el siguiente punto, ya que se trata de un método muy sencillo que entrará en juego con las transacciones.

### Proveedores

El controlador de **proveedores** permite consumir los proyectos de los proveedores, se intercambian mensajes de tipo JSON a través de las API Rest, como ya he comentado en el UML, la agencia no tendrá acceso a todas las funcionalidades de los proveedores, simplemente a las que necesita para dar el servicio de reservas, cancelaciones a los usuarios...

- Mostrar todos los objetos del proveedor
- Mostrar las ofertas disponibles “objetos sin reservar”
- Ver un objeto concreto
- Obtener el precio de un objeto para unas fechas determinadas (ej. un coche que cuesta 50 euros al día, durante 4 días → 200 euros)
- Reservar un objeto
- Cancelar la reserva de un objeto

Aquí también he tenido que hacer uso de funciones asíncronas para la obtención de los datos por parte de las apis de los proveedores.

```

5  const util = require('util')
6  var dateFormat = require('dateformat')
7
8
9  //PROVEEDOR DE VEHICULOS
10 ///////////////////////////////////////////////////////////////////
11
12 function getAllVehicles(req, res)
13 {
14   request.get("https://localhost:3005/api/vehicle", (err, response, body) =>
15   {
16     if (err) return res.status(500).send({message: `Error al contactar con el proveedor de vehiculos: ${err}`})
17     var datos = JSON.parse(body)
18     res.status(200).send(datos)
19   })
20 }
21
22
23 function getVehicle(req, res)
24 {
25   let vehicleId = req.params.vehicleId
26
27   request.get
28   ({
29     "headers": { "content-type": "application/json" },
30     "url": `https://localhost:3005/api/vehicle/${vehicleId}`,
31   },
32
33   (err, response, body) =>
34   {
35     if (err) return res.status(500).send({message: `Error al contactar con el proveedor de vehiculos: ${err}`})
36
37     var datos = JSON.parse(body)
38     res.status(200).send(datos)
39
40     console.dir(JSON.parse(body));
41   });
42 }
43
44
45
46 async function offersVehicles(req, res)
47 {
48   let initdate = req.params.initdateId
49   let enddate = req.params.enddateId
50
51
52   const url = `https://localhost:3005/api/vehicle/filtrarofertas/${initdate}/${enddate}`
53
54
55   const requestPromise = util.promisify(request);
56   const response = await requestPromise(url);
57   const respu = JSON.parse(response.body)
58   const estatus = response.statusCode
59   res.status(estatus).send(respu)
60 }

```

Las llamadas se producen al proveedor correspondiente indicando el puerto y la info necesaria para la función (ej. id objeto, fecha inicio, fecha fin), la Agencia gestiona la información devuelta por los proveedores, en caso de producirse un error se catchea y se muestra el mensaje correspondiente.

Si todo ha ido bien se devuelve un status(200) junto con el json con la información, en caso de haberse producido algún error como no poder contactar con el proveedor (apagado o caído) se devuelve un status (500). Más adelante explicaré como el front-end gestiona estos mensajes en caso de error y las dificultades que he encontrado al comunicar el front, con la agencia y esta con los proveedores.

Finalmente se exportan los módulos para así poder acceder desde el fichero de rutas.

```
356 module.exports =  
357 {  
358     getAllVehicles,  
359     getAllHotels,  
360     getAllFlights,  
361     getVehicle,  
362     getHotel,  
363     getFlight,  
364     offersVehicles,  
365     offersHotels,  
366     offersFlights,  
367     priceoftheVehicle,  
368     priceoftheHotel,  
369     priceoftheFlight,  
370     hacerReservaVehiculo,  
371     hacerReservaHotel,  
372     hacerReservaVuelo,  
373     cancelarReservaVehiculo,  
374     cancelarReservaHotel,  
375     cancelarReservaVuelo  
376 }
```

### Transacciones

Se trata del controlador “más importante” de la aplicación ya que en el gestiono que no se produzcan errores al generar reservas o cancelarlas.

Este controlador se compone por 3 funciones dedicadas a reservas individuales de objetos en los proveedores, una función dedicada a la reserva de un pack (3 objetos simultáneamente), otra función para cancelar cualquier tipo de transacción y por último dos funciones para devolver la información de todas las transacciones (únicamente utilizada en pruebas con postman) y otra función que devuelve la información de una transacción concreta.

```
609 module.exports =  
610 {  
611     transindvehiculos,  
612     transindhoteles,  
613     transindvuelos,  
614     transaccionreservamult,  
615     cancelartransaccion,  
616     getAllTrans,  
617     getTrans  
618 }
```

Para analizar como he implementado las transacciones primero explicaré en que consiste una **transacción individual**.

```
37 async function transindvehiculos(req, res)
38 {
39   let initdate = req.params.initdate
40   let enddate = req.params.enddate
41   let vehicleId = req.params.vehicleId
42
43
44   const url = `https://localhost:3008/api/banco`
45
46   const requestPromise = util.promisify(request);
47   var response = await requestPromise(url);
48
49   var saldo = response.body
50
51   const estatus = response.statusCode
52
53
54   if(estatus == 500)
55   {
56     res.status(500).send({message:"Error al conectar con el banco"})
57   }
58   else
59   {
60     if(saldo == "0")
61     {
62       res.status(500).send({message:"Se cancela por falta de saldo"})
63       console.log({message:"Se cancela la transaccion"});
64     }
65     else
66     {
67       console.log("Continuo con la transaccion");
68       const url2 = `https://localhost:3008/api/reservavehiculo/${vehicleId}/${initdate}/${enddate}`
69       response = await requestPromise(url2);
70       var estadorespuesta = response.statusCode;
71       console.log(estadorespuesta)
72     }
73   }
74 }
```

En este caso realizaré para realizar una reserva de un vehículo se ha de pasar como parámetro el id del vehículo y las fechas deseadas para la reserva. Primero llamaré al banco que aunque explicaré en el siguiente punto básicamente devuelve 0 o 1 en función de si existe o no saldo para realizar la reserva de forma aleatoria. Por lo tanto, primero llamaré al banco, si se produce cualquier tipo de error no continuaré y cancelaré la transacción. En caso de que el banco devuelva un 0 (no hay saldo suficiente) mostraré el mensaje pertinente y devolveré un código de error.

```

if(estadorestado == 201) //se ha podido reservar correctamente
{

    //CALCULO EL PRECIO
    const url3 = `https://localhost:3008/api/preciovehiculo/${vehicleId}/${initdate}/${enddate}`
    var responseprecio = await requestPromise(url3);
    var preciovehi = JSON.parse(responseprecio.body)
    var p = preciovehi.preciototal

    //CREO LA TRANSACCION
    let transaccion = new Transaction()
    transaccion.vehicle = vehicleId
    transaccion.initdate = initdate
    transaccion.enddate = enddate
    transaccion.precio = p

    transaccion.save((err, transactionStored) =>
    {
        if(err) res.status(500).send({message: `Error al crear la transaccion en la bbdd: ${err}`})
        res.status(200).send([transactionStored])
    })
}
else
{
    res.status(500).send({message:"Se cancela la transaccion. Error al reservar, el vehículo ya se encuentra reservado en la bbdd"})
}
}

```

En caso de que el banco haya autorizado la transacción primero trataré de reservar el vehículo a través del método de la agencia que conecta con el proveedor y gestiona la información, si el código que devuelve ese método es 201 (se ha creado la petición con éxito y se ha actualizado la info en la bbdd) entonces creo la transacción y la almaceno en la bbdd de la agencia y la devuelvo, en caso de que el código sea distinto de 201 significa que el proveedor no ha sido capaz de reservar el vehículo en la bbdd, por lo que cancelo la transacción y devuelvo un código de estado incorrecto.

Este caso es el más sencillo dentro del sistema de transacciones, donde realmente se ha de gestionar la realización de un rollback en la BBDD en cuando se realiza la **reserva de un pack**, ya que se produce una transacción múltiple.

En mi caso he utilizado el patrón SAGA orquestado, ya que el método que he creado se comporta como un gestor, donde envía las señales necesarias a las api de los proveedores para reservar los elementos o hacer rollback en la bbdd, comprobando que todos lo han conseguido sin problema, vamos a verlo con más detalle a continuación.

```

242 async function transaccionreservamult(req,res)
243 {
244     let initdate = req.params.initdate
245     let enddate = req.params.enddate
246     let vehicleId = req.params.vehicleId
247     let hotelId = req.params.hotelId
248     let flightId = req.params.flightId
249
250
251     //COMPRUEBO BANCO
252     var url = `https://localhost:3008/api/banco`
253     var requestPromise = util.promisify(request);
254     var response = await requestPromise(url);
255     var saldo = response.body
256     var estatusbanco = response.statusCode
257
258
259     if(estatusbanco == 500)
260     {
261         res.status(500).send({message:"Error al conectar con el banco"})
262     }
263     else
264     {
265         if(saldo == "0")
266         {
267             res.status(500).send({message:"Se cancela por falta de saldo"})
268             console.log("Se cancela la transaccion");
269         }
270         else
271         {
272
273             console.log("AQUI PRINCIPIO PETO")
274
275             //COMPRUEBO VEHICULOS
276             var urlvehiculo = `https://localhost:3008/api/reservavehiculo/${vehicleId}/${initdate}/${enddate}`
277             var responsev = await requestPromise(urlvehiculo);
278             var estadovehiculo = responsev.statusCode;
279             console.log(estadovehiculo + " estado del vehiculo")
280
281
282             //COMPRUEBO HOTELES
283             var urlhotel = `https://localhost:3008/api/reservahotel/${hotelId}/${initdate}/${enddate}`
284             var responseh = await requestPromise(urlhotel);
285             var estadohotel = responseh.statusCode;
286             console.log(estadohotel + " estado del hotel")
287
288
289             //COMPRUEBO VUELOS
290             var urlvuelo = `https://localhost:3008/api/reservavuelo/${flightId}/${initdate}/${enddate}`
291             var responsef = await requestPromise(urlvuelo);
292             var estadovuelo = responsef.statusCode;
293             console.log(estadovuelo + " estado del vuelo")
294
295
296

```

El método comienza igual que una transacción individual comprobando el banco y a continuación llamando a cada uno de los proveedores para tratar de reservar los objetos.



```

297 //si todos han podido se almacena la transaccion
298 if(estadovehiculo == 201 && estadohotel == 201 && estadovuelo == 201)
299 {
300
301
302     //CALCULO LOS PRECIOS DE CADA COSA Y LUEGO LOS SUMO
303
304     //PRECIO VEHICULO
305     const url3 = `https://localhost:3008/api/preciovehiculo/${vehicleId}/${initdate}/${enddate}`
306     var responseprecio = await requestPromise(url3);
307     var preciovehi = JSON.parse(responseprecio.body)
308     var pve = preciovehi.preciototal
309
310     //PRECIO HOTEL
311     const url4 = `https://localhost:3008/api/preciohotel/${hotelId}/${initdate}/${enddate}`
312     responseprecio = await requestPromise(url4);
313     var preciohot = JSON.parse(responseprecio.body)
314     var pho = preciohot.preciototal
315
316     //PRECIO VUELO
317     const url5 = `https://localhost:3008/api/preciovuelo/${flightId}`
318     responseprecio = await requestPromise(url5);
319     var preciofly = JSON.parse(responseprecio.body)
320     var pvu = preciofly.precio
321
322
323     //PRECIO FINAL DEL PACK
324     var preciototal = pve + pho + pvu
325
326     console.log("Continuo con la transaccion");
327     //GENERO LA TRANSACCION
328     let transaccion = new Transaction();
329     transaccion.vehicle = vehicleId;
330     transaccion.hotel = hotelId;
331     transaccion.flight = flightId;
332     transaccion.initdate = initdate
333     transaccion.enddate = enddate
334     transaccion.precio = preciototal
335
336     transaccion.save((err, transactionStored) =>
337     {
338         if(err) res.status(500).send({message: `Error al crear la transaccion en la bbdd: ${err}`})
339         res.status(200).send([transactionStored])
340     })
341 }
342

```

En caso de que todos ellos devuelvan un estado 201 (que se han podido reservar todos correctamente), calculo el precio total de la reserva, sumando el coste de cada uno de los objetos y finalmente almaceno la transacción en la BBDD.

La parte más delicada es cuando uno de estos estados no es 201, esto significa que alguno/s de los objetos no se han podido reservar y se ha de anular la transacción.

```

343     else
344     {
345         //BUSCO CUAL/CUALES HAN CONSEGUIDO RESERVAR Y ANULO
346
347         console.log("AL ENTRAR A ABORTAR")
348
349         var abortvehicle = false
350         var aborthotel = false
351         var abortflight = false
352
353         if(estadovehiculo == 201)
354         {
355             console.log("PREABORT VEHICULO")
356
357             //ABORTO VEHICULO
358             var urlabortvehiculo = `https://localhost:3008/api/cancelarreservavehiculo/${vehicleId}`
359             var responsevehiculo = await requestPromise(urlabortvehiculo);
360             var estadoabortvehiculo = responsevehiculo.statusCode;
361             console.log(estadoabortvehiculo)
362
363             console.log("POSTABORTVEHICULO")
364
365             if(estadoabortvehiculo == 201)
366             {
367                 abortvehicle = true
368             }
369             else
370             {
371                 res.status(500).send({message: `Error al abortar la transaccion en el proveedor de vehiculos: ${err}`})
372             }
373         }
374     }
375     else
376     {
377         abortvehicle = true
378     }
379
280

```

```

381     if(estadohotel == 201)
382     {
383         //ABORTO HOTEL
384
385         console.log("PREABORT HOTEL")
386         var urlaborthotel = `https://localhost:3008/api/cancelarreservahotel/${hotelId}`
387         var responsehotel = await requestPromise(urlaborthotel);
388         var estadoaborthotel = responsehotel.statusCode;
389         console.log(estadoaborthotel)
390         console.log("POSTABORT HOTEL")
391
392         if(estadoaborthotel == 201)
393         {
394             aborthotel = true
395         }
396         else
397         {
398             res.status(500).send({message: `Error al abortar la transaccion en el proveedor de hoteles: ${err}`})
399         }
400     }
401     else
402     {
403         aborthotel = true
404     }
405
406     if(estadovuelo == 201)
407     {
408         //ABORTO VUELO
409         console.log("PREABORT VUELO")
410         var urlabortflight = `https://localhost:3008/api/cancelarreservavuelo/${flightId}`
411         var responsevuelo = await requestPromise(urlabortflight);
412         var estadoabortvuelo = responsevuelo.statusCode;
413         console.log(estadoabortvuelo)
414
415         console.log("POSTABORTVUELO")
416
417         if(estadoabortvuelo == 201)
418         {
419             abortflight = true
420         }
421         else
422         {
423             res.status(500).send({message: `Error al abortar la transaccion en el proveedor de vuelos: ${err}`})
424         }
425     }
426     else
427     {
428         abortflight = true
429     }
430
431
432     if(abortvehicle == true && aborthotel == true && abortflight == true)
433     {
434         //se ha abortado la transaccion correctamente
435         res.status(200).send({message: `Se ha abortado la transaccion correctamente`})
436     }
437

```

En caso de que alguno de ellos haya fallado al reservar compruebo cuales SI han conseguido hacerlo y les mando una señal de cancelar la reserva dada, finalmente cuando se han cancelado correctamente las transacciones mando un código de estado 200 a pesar de no haber realizado la reserva ya que no se ha producido un error como tal en la agencia, en caso de haber fallado algo devuelvo un código de estado 500 (internal server error).

En el caso de **cancelar una reserva** realizo el mismo proceso visto, pero en caso de no haberse podido cancelar algún elemento de la reserva se vuelve a mandar la señal de reserva a los proveedores.

```
448 async function cancelartransaccion(req, res)
449 {
450
451     let transId = req.params.transId
452
453
454
455     console.log("ARRANCAMOS")
456
457     Transaction.findById(transId, async (err, trans) =>
458     {
459         if (err) return res.status(500).send({message: `Error al realizar la petición de lectura en la bbdd: ${err}`})
460         if (!trans) return res.status(404).send({message: `No existe el id de transaccion en la bbdd`})
461
462         console.log("PRIMERA VUELTA")
463
464         var correctovehicle = false
465         var correctovuelo = false
466         var correctohotel = false
467
468         if(trans.vehicle != null) // si existe un vehiculo reservado mando cancelarlo
469         {
470
471             console.log("ENTRO A CANCELAR VEHICULO")
472             console.log(trans.vehicle)
473
474             //ABORTO VEHICULO
475             var urlabortvehiculo = `https://localhost:3008/api/cancelarreservavehiculo/${trans.vehicle}`
476             var requestPromise = util.promisify(request);
477             var responsevehiculo = await requestPromise(urlabortvehiculo);
478             var estadoabortvehiculo = responsevehiculo.statusCode;
479             console.log(estadoabortvehiculo)
480
481
482             console.log("ACABO DE CANCELAR VEHICULO")
483             console.log(estadoabortvehiculo)
484             if(estadoabortvehiculo == 201)
485             {
486                 console.log("CAMBIO EL ESTADO DE ABORTVEHICULO")
487                 correctovehicle = true
488             }
489         }
490         else
491         {
492             correctovehicle = true
493         }
494     }
495 }
```

En este caso no es necesario la verificación del banco (el cliente simplemente perdería el dinero de la reserva), y comienzo leyendo la reserva de la bbdd para acceder a los campos que me harán falta.

Una vez tengo la reserva almacenada en la variable trans compruebo que tipo de objeto está almacenado en la reserva (vehículo hotel vuelo) o varios simultáneamente.

Para comprobar que la transacción ha funcionado correctamente compruebo en las variables booleanas correctovehicle, correctohotel y correctovuelo. La variable cambiará su estado a true en los dos casos posibles, que se haya podido cancelar la reserva en caso de que el objeto exista o simplemente que la reserva no contenga ese objeto y se de el estado correcto directamente.

Una vez que he mandado la señal de cancelación a los proveedores necesarios compruebo que se haya realizado correctamente la cancelación, cambiando el estado de la reserva a cancelada y poniendo los id de los objetos a null y devolviendo la transacción al front.

```
541 if(correctovehicle == true && correctohotel == true && correctovuelo == true) // se comprueba que habia reserva del element
542 {
543     //la transaccion se cancela cambiando el estado del campo cancelada
544
545     const transFilter = {_id: transId}
546     const data = await Transaction.findOneAndUpdate(transFilter, {cancelada: true, vehicle: null, hotel: null, flight: nul
547
548
549     Transaction.findById(transId, async (err, trans) =>
550     {
551         res.status(200).send([trans])
552     })
553
554     //cancelado = true
555
556 }
557
```

En caso de que alguno haya fallado es cuando se ha de producir el rollback necesario para cancelar la cancelación valga la redundancia, enviando de nuevo las señales de reserva a los elementos necesarios y devolviendo el estado pertinente.

```
558 else // si no se ha podido cancelar la reserva correctamente se vuelven a reservar los elementos que sean necesarios
559 {
560
561     if(correctovehicle == false)
562     {
563         //vuelvo a reservar el vehículo
564         //COMPRUEBO VEHICULOS
565         var urlvehiculo = `https://localhost:3008/api/reservavehiculo/${trans.vehicle}/${trans.initdate}/${trans.enddate}`
566         var responsev = await requestPromise(urlvehiculo);
567         var estadovehiculo = responsev.statusCode;
568         console.log(estadovehiculo + " estado del vehiculo")
569     }
570
571     if(correctohotel == false)
572     {
573         //vuelvo a reservar el hotel
574         //COMPRUEBO HOTELES
575         var urlhotel = `https://localhost:3008/api/reservahotel/${trans.hotel}/${trans.initdate}/${trans.enddate}`
576         var responseh = await requestPromise(urlhotel);
577         var estadohotel = responseh.statusCode;
578         console.log(estadohotel + " estado del hotel")
579     }
580
581     if(correctovuelo == false)
582     {
583         //vuelvo a reservar el vuelo
584         //COMPRUEBO VUELOS
585         var urlvuelo = `https://localhost:3008/api/reservavuelo/${trans.flight}/${trans.initdate}/${trans.enddate}`
586         var responsef = await requestPromise(urlvuelo);
587         var estadovuelo = responsef.statusCode;
588         console.log(estadovuelo + " estado del vuelo")
589     }
590
591
592     if(estadovehiculo == 201 && estadohotel == 201 && estadovuelo == 201)
593     {
594         res.status(200).send({message: "Ha habido un error inesperado al cancelar la reserva pero se ha abortado correctamente el intento"})
595     }
596     else
597     {
598         res.status(500).send({message: "Error al cancelar la transaccion..."})
599     }
600 }
601 })
602 }
```

## User

Por último, voy a explicar el controlador de usuarios y como he implementado la autorización a través de token y como se transmite hacia el frontend.

```
1  'use strict'
2
3  const mongoose = require('mongoose')
4  const User = require('../models/user')
5  const service = require('../middlewares/auth')
6  const config = require('../config')
7  const bcrypt = require('bcrypt-nodejs')
8  const express = require('express')
9  const router = express.Router();
10
11
12  const signUp = (req, res) =>
13  {
14    const user = new User
15    ({
16      email: req.body.email,
17      displayName: req.body.displayName,
18      password: req.body.password
19    })
20
21    user.save(err =>
22    {
23      if (err) return res.status(500).send({ msg: `Error al crear usuario: ${err}` })
24      return res.status(200).send({ token: service.createToken(user) })
25    })
26  }
27
28
29
30
31
32  const signIn = (req, res) =>
33  {
34    User.findOne({ email: req.params.email }, (err, user) =>
35    {
36      if (err) return res.status(500).send({ msg: `Error al ingresar: ${err}` })
37      if (!user) return res.status(404).send({ msg: `No existe el usuario: ${req.body.email}` })
38
39      return user.comparePassword(req.params.password, (err, isMatch) =>
40      {
41        if (err) return res.status(500).send({ msg: `Error al ingresar: ${err}` })
42        if (!isMatch) return res.status(404).send({ msg: `Error de contraseña: ${req.body.email}` })
43
44        req.user = user
45        return res.status(200).send({ msg: 'Te has logueado correctamente', token: service.createToken(user) })
46      });
47    })
48    .select('_id email +password');
49  }
50
51
52
53  module.exports =
54  {
55    signUp,
56    signIn
57  }
```

En el controlador solo existen 2 métodos como se puede ver, signUp y signIn, uno destinado al registro y otro al inicio de sesión de los usuarios, el primer método es el de registro, simplemente recibirá los datos del formulario del frontend correspondientes a la información del nuevo usuario y cuando se realiza.save se almacena en la BBDD pero como ya he explicado antes pasará por el método del modelo .pre que almacenará la contraseña hasheada.

A continuación, se devuelve un token que se genera con la función que he implementado en un middleware que sirve como capa intermedia de comprobación para así proteger la parte privada de la aplicación.

El middleware está ubicado en la ruta → ./middlewares/auth.js

```
41 function createToken(user)
42 {
43   const payload =
44     {
45       sub: user._id, //id usuario, es el de mongodb aunque no es lo mas recomendable se puede poner en peli
46       iat: moment().unix(),
47       exp: moment().add(14, 'days').unix() // añade a la etiqueta 14 dias al tiempo de unix que es el de a
48     }
49
50   //ahora codificamos el payload
51   return jwt.encode(payload, config.SECRET_TOKEN) // el secret esta en el fichero config.js
52 }
53
```

El token se genera con el formato estándar, donde primero tiene el sub, que en este caso por simplicidad es el id del usuario pero esto no es recomendable ya que puede comprometer la seguridad de la aplicación, pero por simplicidad lo haré así, este campo actúa como identificador, a continuación tiene la marca temporal en la que se ha generado el token y por último el tiempo de expiración de este, yo he puesto 14 días para realizar pruebas de manera tranquila, en una página real sería de unos minutos.

El token se genera cuando se encripta el payload con la ayuda de la función .encode de la librería jwt (JSON Web Token) de nodejs, para poder desencriptar posteriormente el token se ha de encriptar con la ayuda de una clave secreta, que en mi caso almaceno como una variable de entorno en el fichero .config (Esto no es seguro en el mundo real).

```
1 module.exports =
2 {
3   port: process.env.PORT || 3008,
4   db: process.env.MONGODB || 'mongodb://localhost:27017/agencia',
5   SECRET_TOKEN: 'miltonb2117af6667bf3w'.toString('base64')
6 }
```

Bueno ahora ya hemos visto como se genera almacena un usuario con la contraseña hasheada en la BBDD y como se genera el token que se enviará al cliente en el frontend, ahora vamos a ver como se puede utilizar ese token y como se protegerá la aplicación.

El otro método del controlador de usuarios obviamente es el que permite iniciar sesión:

```
32 ∨ const signIn = (req, res) =>
33 {
34 ∨ User.findOne({ email: req.params.email }, (err, user) =>
35 {
36   if (err) return res.status(500).send({ msg: `Error al ingresar: ${err}` })
37   if (!user) return res.status(404).send({ msg: `No existe el usuario: ${req.body.email}` })
38
39   return user.comparePassword(req.params.password, (err, isMatch) =>
40   {
41     if (err) return res.status(500).send({ msg: `Error al ingresar: ${err}` })
42     if (!isMatch) return res.status(404).send({ msg: `Error de contraseña: ${req.body.email}` })
43
44     req.user = user
45     return res.status(200).send({ msg: 'Te has logueado correctamente', token: service.createToken(user) })
46   });
47
48   }).select('_id email +password');
49 }
50
```

Se busca el usuario en la BBDD y ahora es necesario hacer uso del método `.comparePassword` para comparar la contraseña que viene por parte del front con la que esta hasheada en la BBDD como ya expliqué antes.

```
38 ∨ UserSchema.methods.comparePassword = function (candidatePassword, cb)
39 {
40 ∨ bcrypt.compare(candidatePassword, this.password, (err, isMatch) =>
41 {
42   cb(err, isMatch)
43 });
44 }
```

Si se comprueba que el usuario y la contraseña son correctos se devuelve de nuevo un token que se almacenará en el navegador del usuario, en caso negativo se indica el error correspondiente.

## Rutas

Una vez hemos visto los controladores de la Agencia podemos ver de manera más clara las rutas.

```
1  'use strict'
2
3  const express = require('express')
4  const api = express.Router()
5  const auth = require('../middlewares/auth')
6  const userCtrl = require('../controllers/user')
7  const proveedorCtrl = require('../controllers/proveedores')
8  const bancoCtrl = require('../controllers/banco')
9  const transacCtrl = require('../controllers/transaction')
10
11  //RUTAS USUARIOS
12  ////////////////////////////////////////////////// primero llama al middleware
13  api.get('/private', auth.isAuthenticated, function(req,res)
14  {
15    res.status(200).send({message:'Tienes Acceso'})
16  })
17
18  api.post('/signup', userCtrl.signUp)
19
20  api.post('/signin/:email/:password', userCtrl.signIn)
21
22  //////////////////////////////////////////////////
23
24  //Rutas proveedor vehiculos
25  api.get('/allvehicles', proveedorCtrl.getAllVehicles)
26  api.get('/:getvehicle/:vehicleId', proveedorCtrl.getVehicle)
27  api.get('/ofertasvehiculos/:initdate/:enddate', auth.isAuthenticated, proveedorCtrl.offersVehicles)
28  api.get('/preciovehiculo/:vehicleId/:initdate/:enddate', proveedorCtrl.priceoftheVehicle)
29  api.get('/reservavehiculo/:vehicleId/:initdate/:enddate', proveedorCtrl.hacerReservaVehiculo)
30  api.get('/cancelarreservavehiculo/:vehicleId', proveedorCtrl.cancelarReservaVehiculo)
31
32
33  //Rutas proveedor hoteles
34  api.get('/allhotels', proveedorCtrl.getAllHotels)
35  api.get('/:gethotel/:hotelId', proveedorCtrl.getHotel)
36  api.get('/ofertashoteles/:initdate/:enddate', auth.isAuthenticated, proveedorCtrl.offersHotels)
37  api.get('/preciohotel/:hotelId/:initdate/:enddate', proveedorCtrl.priceoftheHotel)
38  api.get('/reservahotel/:hotelId/:initdate/:enddate', proveedorCtrl.hacerReservaHotel)
39  api.get('/cancelarreservahotel/:hotelId', proveedorCtrl.cancelarReservaHotel)
40
41
42  //Rutas proveedor aviones
43  api.get('/allflights', proveedorCtrl.getAllFlights)
44  api.get('/:getflight/:flightId', proveedorCtrl.getFlight)
45  api.get('/ofertasvuelos/:initdate/:enddate', auth.isAuthenticated, proveedorCtrl.offersFlights)
46  api.get('/preciovuelo/:flightId', proveedorCtrl.priceoftheFlight)
47  api.get('/reservavuelo/:flightId/:initdate/:enddate', proveedorCtrl.hacerReservaVuelo)
48  api.get('/cancelarreservavuelo/:flightId', proveedorCtrl.cancelarReservaVuelo)
49
50
51  //Ruta banco
52  api.get('/banco', bancoCtrl.hayDineroTarjeta)
53
54
55  //Ruta transacciones
56
57
58    //todas las transacciones
59  api.get('/alltrans', transacCtrl.getAllTrans)
60  api.get('/:transinfo/:transId', auth.isAuthenticated, transacCtrl.getTrans)
61
62    //reservas individuales
63  api.get('/:transindvehiculo/:vehicleId/:initdate/:enddate', transacCtrl.transindvehiculos)
64  api.get('/:transindhoteles/:hotelId/:initdate/:enddate', transacCtrl.transindhoteles)
65  api.get('/:transindvuelo/:flightId/:initdate/:enddate', transacCtrl.transindvuelos)
66
67    //reservar pack
68  api.get('/:transmult/:vehicleId/:hotelId/:flightId/:initdate/:enddate', transacCtrl.transaccionreservamult)
69
70    //cancelar reserva
71  api.get('/:canceltrans/:transId', auth.isAuthenticated, transacCtrl.cancelartransaccion)
72
73  module.exports = api
```



Las rutas funcionan de manera idéntica a las de los proveedores, con una pequeña diferencia, hay funciones que han de pasar por el middleware de autorización, para acceder a estas funcionalidades del back-end se ha de tener un token válido y por lo tanto eso significa estar logueado o recién registrado. Para comprobar que el usuario envía el token válido dispongo de una función que voy a analizar, desde el frontend llegará el token en la cabecera de la request.

```
10 function isAuth (req, res, next)
11 {
12   if (!req.headers.authorization)
13   {
14     return res.status(403).res({ message: `Acceso Denegado` })
15   }
16
17   const token = req.headers.authorization.split(' ')[1]
18
19
20   try
21   {
22     const payload = jwt.decode(token, config.SECRET_TOKEN)
23
24     if (payload.exp < moment().unix())
25     {
26       return res.status(401).send({ message: `El token ha expirado, no tienes acceso` })
27     }
28
29     req.user = payload.sub
30     next()
31   }
32   catch(err)
33   {
34     return res.status(500).send({message: `Token invalido`})
35   }
36 }
37
```

Los tokens cuando se envían a través de las cabeceras se identifican a través de un campo llamado Authorization, y tienen el siguiente formato:

**Authorization: Bearer [token]**

Dentro del campo authorization se escribe primero la palabra Bearer y a continuación se encuentra el token. Por lo tanto lo que hago es comprobar que exista el campo authorization primero, en caso de que exista parto la cadena y me quedo únicamente con el token.

Hago uso de la función .decode que proporciona la librería jwt-simple para decodificar el token, comprobar que no haya expirado y ver que es válido, en caso de que no haya habido ningún problema y el token sea válido se llamará sin problema al método dado. En mi caso he decidido que el usuario tenga que estar registrado/logueado para poder reservar aunque esto lo veremos con más detalle en el front, por otra parte podrá ver todos los objetos que dispone la agencia (vehículos, hoteles, vuelos).

Ya hemos visto el funcionamiento de la agencia y la seguridad implementada en ella, a falta del cifrado de los canales que hay en ella. El cifrado lo realizaré con openssl, estos certificados autofirmados se pueden generar de manera sencilla y incluirlos en la aplicación es muy sencillo.

Para generar los certificados:

**openssl req -nodes -new -x509 -keyout server.key -out server.cert**

Aquí podemos ver como al arrancar el servidor se arrancan con los certificados:

```
19   https.createServer({
20     key: fs.readFileSync('./certificados/server.key'),
21     cert: fs.readFileSync('./certificados/server.cert')
22   }, app).listen(config.port, () =>
23     {
24       console.log(`AGENCIA DE VIAJES CORRIENDO EN http://localhost:\${config.port}`)
25     })
```

Ahora cuando se arranca el servidor pasamos de utilizar http a https, a pesar de ellos los navegadores siguen mostrando la página como insegura ya que el certificado es autofirmado y no es contrastado por una entidad certificadora real.

Los navegadores modernos como Chrome, Firefox, Safari, Opera... dan una gran cantidad de problemas para enviar y recibir peticiones en entornos de desarrollo con certificados autofirmados, por lo que para probar sin tener problemas todas las funcionalidades de la agencia me vi obligado a insertar la siguiente línea.

Con esto se evita que se comprueben los certificados, de esta manera no se tiene que estar añadiendo a la lista de certificados de confianza en los navegadores de los sistemas, pero sigue existiendo openssl (https).

```
29   app.use(express.json())
30   process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0"
31
```

Esto lo que hace es que no comprobar la validación de los certificados ya que si no los navegadores no me dejan acceder ni al entorno del frontend, esto es problema de los navegadores ya que no dejan acceder a la página.

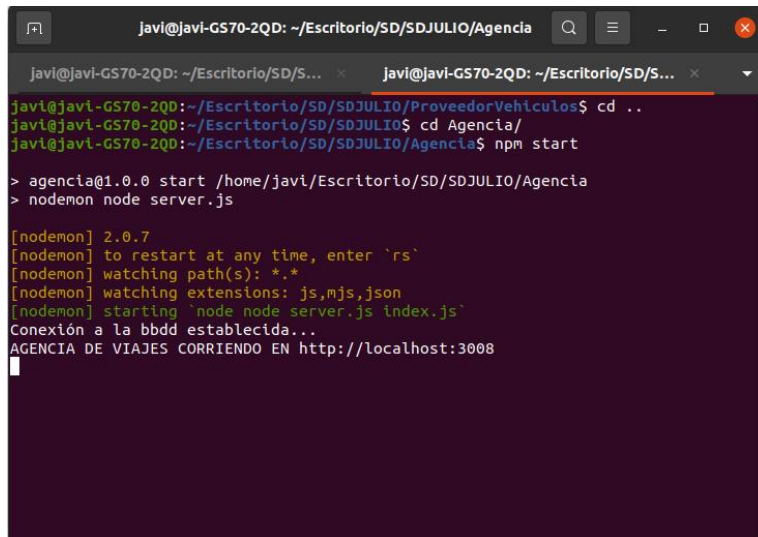
Al mismo tiempo me encontré con un problema de autorización de acceso desde el frontend a la agencia por los navegadores, con la política cors, es un mecanismo que utiliza cabeceras http adicionales para permitir acceder a los recursos de un servidor (en este caso la agencia), para solucionarlo tuve que dar acceso manualmente desde la agencia al frontend.

```

13  var cors = require('cors');
14
15  // use it before all route definitions
16  app.use(cors({origin: 'https://localhost:4200'}));
17

```

## Pruebas con PostMan



```

javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO/Agencia
javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO/ProveedorVehiculos$ cd ..
javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO$ cd Agencia/
javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO/Agencia$ npm start

> agencia@1.0.0 start /home/javi/Escritorio/SD/SDJULIO/Agencia
> nodemon node server.js

[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node node server.js index.js`
Conexión a la bdd establecida...
AGENCIA DE VIAJES CORRIENDO EN http://localhost:3008

```

### Agencia-Usuario

Agencia

3 requests

GET

private

POST

crear usuario signup

POST

signin

Prueba signin que devuelve el token

POSTsignin

▶signin

POST

https://localhost:3008/api/signin/javi/pass

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

password

tardes

Key

Value

Body

Cookies

Headers (9)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"msg": "Te has logueado correctamente",

3

"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI2MGRjYzEwNDdlYTMsZDI2OWI3M2FjNWUiLCJpYXQiOiJE2Mj

4

}

Prueba signup error el usuario ya esta creado

POSTsignin

POSTcrear usuario signup

▶crear usuario signup

POST

https://localhost:3008/api/signup

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

KEY

email

displayName

password

Key

VALUE

javifwqefasg

javimt98

pass

Value

DESCRIP

Descrip

Body

Cookies

Headers (9)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"msg": "Error al crear usuario: MongoError: E11000 duplicate key error collection: agencia.users index: email\_1 dup key: { email: \"javifw

3

}

## Agencia-Proveedores

Agencia-Proveedores
18 requests
GET Get All Vehicles
GET Get All Hotels
GET Get All Flights
GET Get Vehicle
GET Get Hotel
GET Get Flight
GET Filtrar ofertas Vehiculos
GET Filtrar ofertas Hoteles
GET Filtrar ofertas Vuelos
GET Precio Vehiculo
GET Precio Hotel
GET Precio Vuelo
GET Reserva Vehiculo
GET Reserva Hotel
GET Reserva Vuelos
GET Cancelar Reserva Vehiculo
GET Cancelar Reserva Hotel
GET Cancelar Resrva Vuelo

## Método que devuelve un vehículo desde la agencia

GEThttps://localhost:3008/api/getvehicle/60d8ef32a3df8692e0401c00

ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

BodyCookiesHeaders (9)Test Results

PrettyRawPreviewVisualizeJSON

```
1 {
2   "price": 100,
3   "_id": "60d8ef32a3df8692e0401c00",
4   "brand": "Harley Davidson",
5   "model": "Street Bob",
6   "seats": 1,
7   "type": "moto",
8   "year": 2013,
9   "picture": "rutadavidson",
10  "description": "Gasolina 140cv",
11  "rented": true,
12  "initdate": "2021-01-01T00:00:00.000Z",
13  "enddate": "2021-04-04T00:00:00.000Z",
14  "__v": 0
15 }
```

javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO/Agencia

javi@javi-GS70-2QD: ~/Escritorio/SD/S... xjavi@javi-GS70-2QD: ~/Escritorio/SD/S...

```
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node node server.js index.js'
Conexión a la bbdd establecida...
AGENCIA DE VIAJES CORRIENDO EN http://localhost:3008
(node:15900) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes TLS connections and HTTPS requests insecure by disabling certificate verification.
(Use 'node --trace-warnings ...' to show where the warning was created)
{
  price: 100,
  _id: '60d8ef32a3df8692e0401c00',
  brand: 'Harley Davidson',
  model: 'Street Bob',
  seats: 1,
  type: 'moto',
  year: 2013,
  picture: 'rutadavidson',
  description: 'Gasolina 140cv',
  rented: true,
  initdate: '2021-01-01T00:00:00.000Z',
  enddate: '2021-04-04T00:00:00.000Z',
  __v: 0
}
```

## Agencia-Transacción Individual Vehículo

GET Trans Ind Vehiculos

+

...

▶ Trans Ind Vehiculos

GET

https://localhost:3008/api/transindvehiculo/60d8ef32a3df8692e0401bfa/2008-02-02/2010-04-06

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

KEY	VALUE
Key	Value

Body

Cookies

Headers (9)

Test Results

Pretty

Raw

Preview

Visualize

JSON

≡

```
1  {
2
3    "fecha": "2021-07-13T17:38:02.719Z",
4    "cancelada": false,
5    "vehicle": "60d8ef32a3df8692e0401bfa",
6    "hotel": null,
7    "flight": null,
8    "precio": 316806,
9    "_id": "60edd0eba073613e1ce83be6",
10   "initdate": "2008-02-02T00:00:00.000Z",
11   "enddate": "2010-04-06T00:00:00.000Z",
12   "__v": 0
13 }
14
```

## Agencia-Transacción Múltiple

GET Trans Multiple

+

...

▶ Trans Multiple

GET

https://localhost:3008/api/transmult/60d8ef32a3df8692e0401bfa/60d8ffaf7916c1a29390f06e/60d989cee8734e1da148d9f7/2008-02-02/2010-04-06

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

KEY	VALUE
Key	Value

Body

Cookies

Headers (9)

Test Results

Pretty

Raw

Preview

Visualize

JSON

≡

```
1  {
2
3    "fecha": "2021-07-13T17:38:02.719Z",
4    "cancelada": false,
5    "vehicle": "60d8ef32a3df8692e0401bfa",
6    "hotel": "60d8ffaf7916c1a29390f06e",
7    "flight": "60d989cee8734e1da148d9f7",
8    "precio": 364481,
9    "_id": "60edd13ca073613e1ce83be8",
10   "initdate": "2008-02-02T00:00:00.000Z",
11   "enddate": "2010-04-06T00:00:00.000Z",
12   "__v": 0
13 }
14
```

## Banco

El banco es el elemento más sencillo del sistema. Se trata de una estructura Cliente-Servidor, donde el cliente se encuentra en la Agencia y el servidor se encuentra en un proyecto aparte.

La tecnología de interconexión es un socket TCP creado con la ayuda de la librería net, el banco se despliega en el puerto 3001 de la máquina en la que se encuentre.

La función que cumplirá este elemento del sistema es muy sencilla, simplemente cuando el cliente se conecte al servidor se generará un número aleatorio de 0 a 500, si el número es menor o igual a 250 el socket devolverá un 0, o un 1 en caso contrario, Indicando si la tarjeta no tiene saldo o sí para realizar la reserva dada.

He decidido implementar de la forma más sencilla posible este componente ya que realizar esto de manera más realista aumenta bastante la dificultad y el tiempo de implementación del sistema. Aquí podemos ver la parte del servidor:

```
1  'use strict'
2
3  const mongoose = require('mongoose')
4  const app = require('./app')
5  var net = require('net');
6
7
8  // Crea un net.Socket, objeto que representa una conexión TCP
9  const server = net.createServer()
10
11
12  function between(min, max)
13  {
14    return Math.floor
15    (
16      Math.random() * (max - min) + min
17    )
18  }
19
20
21  server.on('connection', (socket) =>
22  {
23
24    socket.on('data', (data) =>
25    {
26      console.log('Se acaba de conectar un cliente');
27
28      var respuesta = between(0, 500);
29      console.log(respuesta)
30
31      //genero un random de 0 a 500 para ver si hay saldo o no de forma aleatoria
32      if(respuesta <= 250)
33      {
34        socket.write("0")
35        console.log('La tarjeta no tiene saldo.');
```

Cuando se produce una conexión desde la parte del cliente se pueden observar dos funciones principales, la primera es cuando el server recibe información `socket.on("data", (data) =>)` y la otra es cuando se cierra la comunicación `socket.on("close", () =>)`.

En el caso de que el socket reciba comunicación por parte de un cliente se genera el número aleatorio y se escribe en el canal, mostrando por consola si la tarjeta dispone o no de saldo.

Por otra parte, tenemos la parte del cliente en la Agencia. Se trata de un controlador con un único método que se conecta al servidor del banco.

```
let / ... bancojs / ... hayDineroTarjeta / ... clienton(data, callback)

'use strict'

const net = require('net')
const readline = require('readline-sync')

// COMUNICACION CON EL BANCO A TRAVES DE SOCKET TCP CON LA AYUDA DE LA LIBRERIA NET
function hayDineroTarjeta(req, res)
{
  const options =
  {
    port: 3001,
    host: 'localhost'
  }

  const client = net.createConnection(options)
  client.setEncoding('utf-8');

  client.on('data', (data) =>
  {
    console.log('////////////////////////////////////')

    if(data == "0")
      console.log('La tarjeta no tiene saldo suficiente para realizar la reserva. Cancelar Operación.')
    else
      console.log('La tarjeta tiene saldo suficiente para realizar la reserva. Realizar Operación.')

    console.log('////////////////////////////////////')

    client.destroy()

    res.status(200).send(data.toString('ascii'))
  })

  client.on('connect', () =>
  {
    console.log('Conexion establecida')
    client.write('Aquí la agencia te envío una tarjeta para comprobar el saldo!')
  })

  client.on('error', (err) =>
  {
    console.log('Error no se ha podido entablar la conexion con el banco...')
    console.log(err.message)
    res.status(500).send(err.message)
  })
}

module.exports =
{
  hayDineroTarjeta
}
```



He declarado de forma estática el puerto al que se conecta y la dirección del host, a continuación, declaro la conexión con las opciones que tendrá el socket, algo muy importante es declarar esta conexión con la codificación utf-8 ya que, si no el canal por defecto envía un Buffer de información y no mensajes de texto plano, para así poder leerlos y compararlos de forma sencilla. Esto problema me costó bastante tiempo solucionarlo.

En este caso tenemos 3 funciones dentro del cliente, una para cuando se inicia la conexión, otra para cuando se finaliza y otra para cuando se reciben datos en el canal, en este caso compruebo que se ha escrito desde la parte del servidor ("1" o "0") para así devolver esta respuesta hacia la parte transaccional de la agencia. Al enviar la información lo envío en formato string para que sea más sencillo trabajar con ella más adelante.

Al tratarse de un componente tan sencillo del sistema simplemente hay una método y por lo tanto una ruta declarada en la agencia.

Una traza de lo que ocurre cuando se llama en las transacciones al banco:

- Cliente se conecta y escribe en el canal un mensaje
- Servidor recibe data y se activa la función 'data'
- Servidor destruye la comunicación con el cliente y escribe en el canal la respuesta de si hay saldo suficiente
- Cliente recibe información en al activar la función 'data' y devuelve el mensaje respuesta a la parte transaccional en formato String de forma legible.

## Prueba con PostMan

The image shows a screenshot of a Postman REST client interface and a terminal window. The Postman interface is set to a GET request to `https://localhost:3008/api/banco`. The 'Body' tab is selected, showing a 'Pretty' view with the response `1 0`. The terminal window shows the command prompt of a user named 'javi' in a directory `~/Escritorio/SD/SDJULIO/Banco`. The user runs `npm start`, which starts a Node.js server. The server logs show that it is running on port 3001 and has received a client connection. The response is `La tarjeta no tiene saldo. Cerrando comunicacion`.

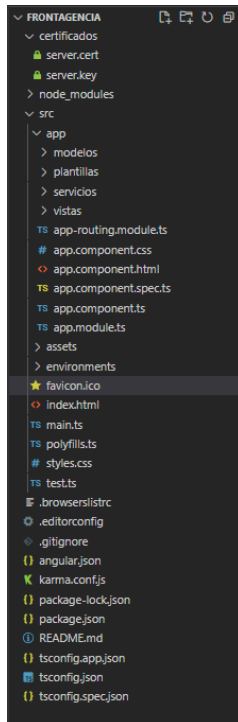
```
javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO/Banco
javi@javi-GS70-2QD: ... x javi@javi-GS70-2QD: ... x javi@javi-GS70-2QD: ... x
javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO/Banco$ cd ..
javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO$ cd Ban
bash: cd: Ban: No existe el archivo o el directorio
javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO$ cd Banco/
javi@javi-GS70-2QD: ~/Escritorio/SD/SDJULIO/Banco$ npm start

> banco@1.0.0 start /home/javi/Escritorio/SD/SDJULIO/Banco
> nodemon node server.js

[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting node server.js index.js
Servidor corriendo en el puerto 3001 localhost
Se acaba de conectar un cliente
211
La tarjeta no tiene saldo.
Cerrando comunicacion
//////////
```

# Front-End

El front-end esta totalmente desacoplado del back-end ubicado en la agencia, lo he implementado en angular, es un framework de código abierto desarrollado por Google que permite crear aplicaciones web de una sola página SPA(Single Page Application) en typescript, es la primera vez que he gastado esta tecnología y estoy bastante satisfecho con ella ya que esta muy bien documentada y es amigable con el usuario, vamos a ver la estructura del proyecto.



La estructura es similar a la de los proyectos de nodejs que he implementado, en la carpeta certificados almaceno estos para la comunicación con openssl. Por otra parte tenemos la configuración general de las rutas del proyecto en el fichero app-routing-module.ts, la declaración de componentes en el fichero app.module.ts

```
20 const routes: Routes =
21 [
22   {path: '', redirectTo: 'dashboard', pathMatch: 'full'},
23   {path: 'login', component: LoginComponent},
24   {path: 'dashboard', component: DashboardComponent},
25   {path: 'header', component: HeaderComponent},
26   {path: 'footer', component: FooterComponent},
27   {path: 'allvehicles', component: AllvehiculosComponent},
28   {path: 'allhotels', component: AllhotelsComponent},
29   {path: 'allflights', component: AllvuelosComponent},
30   {path: 'ofertasvehiculos', component: OfertasVehiculosComponent},
31   {path: 'ofertashoteles', component: OfertashotelesComponent},
32   {path: 'ofertasvuelos', component: OfertasvuelosComponent},
33   {path: 'registro', component: RegistroComponent},
34   {path: 'cancelar', component: CancelarreservaComponent},
35   {path: 'pack', component: ReservarpackComponent},
36   {path: 'consultarreserva', component: ConsultarreservaComponent}
37
38 ];
39
```

En la carpeta assets almaceno las fotos que utiliza las vistas de la aplicación. En la carpeta modelos declaro las interfaces que utilizaré para gestionar el paso de los mensajes, aunque eso lo veremos ahora con mayor profundidad.

Angular se basa en el Modelo-Vista-Controlador, al tratarse de una web SPA, la carga la primera vez que se abre la página puede ser un poco lenta, pero después todo es instantáneo ya que carga toda la página de golpe, en mi caso al ser una página tan pequeña el tiempo de carga es ínfimo.

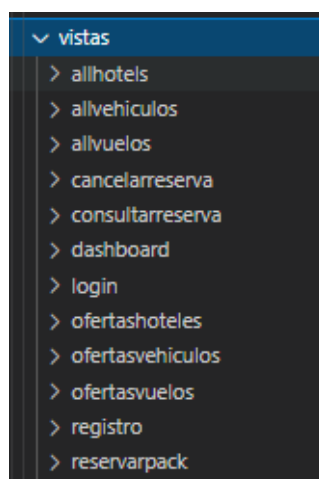
Angular se basa en componentes, un componente es un bloque de código que consta de 3 archivos con el mismo nombre pero distinta terminación, un archivo css (estilo de la página), un html (plantilla web) y un archivo .ts(que contiene la lógica de negocio). He creado un componente para cada una de las páginas de la aplicación que dentro contienen la lógica que se comunicará con el backend situado en la agencia. Dentro de la carpeta app encontramos la implementación real del front.

./app

- ./modelos : Interfaces donde se define el tipo de objeto y formato que se va a recibir
- ./plantillas: 2 componentes, el footer y el header (pie y encabezado de la página)
- ./servicios: Carpeta donde se almacena el archivo de lógica que consultan a la Agencia
- ./vistas: Contiene los componentes que forman la página.

Cuando definimos un componente en angular se puede exportar y utilizar como una ruta, es la imagen que mostré más arriba.

Vamos a ver algunos de los componentes que he implementado:



Las funcionalidades que he implementado en el front que pueden ser consumidas del back son las siguientes:

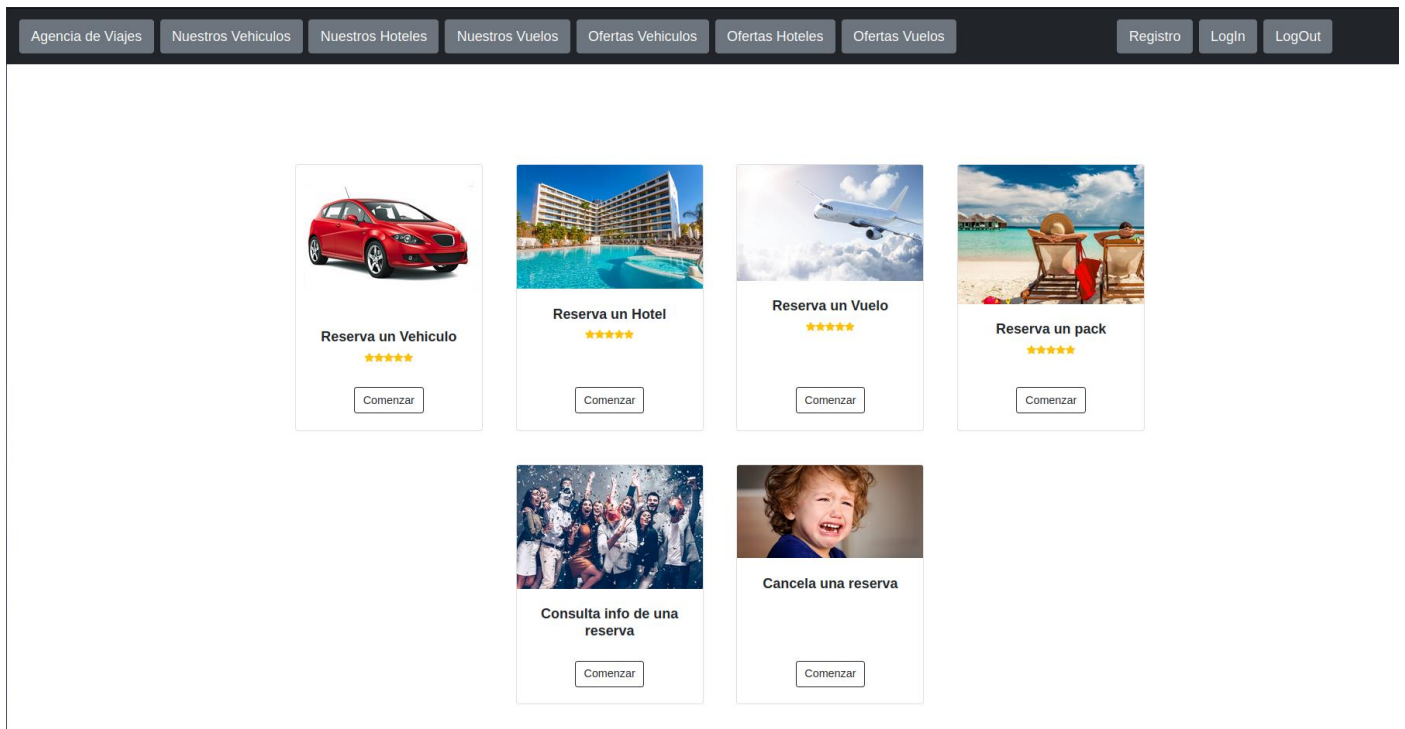
- Allvehiculos, allhotels, allvuelos : Muestran consumen la info del backend y muestra todos los objetos que disponen los proveedores.
- Cancelarreserva: Cancela una reserva dada.
- Consultarreserva: Consultar la información de una reserva concreta.
- Dashboard: Página de inicio
- Login: Loguear al usuario
- Registro: Registrar al usuario
- Ofertasvehiculos, ofertashoteles, ofertasvuelos: Ver las ofertas que corresponden con objetos no reservados y reservar alguno de ellos
- Reservarpack: Reservar un pack de 3 objetos (vehículo, hotel, vuelo).

## Componentes

Voy a analizar los componentes y como se comunican con el backend, para no ser redundante si los componentes son muy similares los agruparé.

### Dashboard

Es la página de principal de la aplicación y no implemento ninguna funcionalidad ahí, simplemente es una vista.



## Allvehicles/Allhotels/Allvuelos

```
5  import{ListaVehiculosI} from '../modelos/listavehiculos.interface'
6
7
8  @Component({
9    selector: 'app-allvehiculos',
10   templateUrl: './allvehiculos.component.html',
11   styleUrls: ['./allvehiculos.component.css']
12 })
13 export class AllvehiculosComponent implements OnInit {
14
15   constructor(private api: AgenciaService, private router:Router) { }
16
17
18   vehiculos:ListaVehiculosI[] = [];
19
20   ngOnInit(): void {
21     this.api.getAllVehicules().subscribe(data =>
22     {
23       console.log(data);
24       this.vehiculos = data;
25     })
26   }
27 }
28
```

Aquí podemos ver el .ts del componente allvehicles simplemente se encarga de llamar al método situado en el fichero agencia.service.ts que devolverá una lista de vehículos en formato json, por lo que primero importo la interfaz que contiene el formato con el que llegará el mensaje.

```
1  export interface ListaVehiculosI
2  {
3    price: String;
4    _id : String;
5    brand : String;
6    model : String;
7    seats : String;
8    type: String;
9    year: String;
10   description: String;
11   rented: String;
12 }
```

Al llamar al método que contiene el servicio:

```

50   getAllVehicles():Observable<ListaVehiculosI[]>
51   {
52
53       let url = "https://localhost:3008/api/allvehicles";
54
55
56       return this.http.get<ListaVehiculosI[]>(url);
57   }
58

```

Se consume el back que puede estar ubicado en otra máquina perfectamente que contiene la lógica para recibir la info de los proveedores. Si nos fijamos arriba podemos declarar parámetros que recibe la función y la salida de la misma, en mi caso defino que el método de salida será de tipo ListaVehiculosI[], aunque si nos fijamos esta encapsulado por Observable<>.

Esto es aquello que nos permite escuchar eventos que van a suceder, es una forma de suscripción en la cual el observador se suscribe al observable. En este caso se realiza una petición de tipo get a la url declarada.

De vuelta al component.ts se asigna la información recibida de hacer la petición a una variable de tipo ListaVehiculosI que será el que mostremos en el .html en una tabla con formato. En caso de el objeto recibido no sea del tipo ListaVehiculosI (por ejemplo un status error 500) se puede catchear el error y mostrar un mensaje de error por pantalla, aunque esto lo mostraré en la demostración.

```

1   <app-header></app-header>
2
3   <table class="table table-dark">
4       <thead>
5           <tr>
6               <th scope="col">ID</th>
7               <th scope="col">Marca</th>
8               <th scope="col">Modelo</th>
9               <th scope="col">Precio</th>
10              <th scope="col">Asientos</th>
11              <th scope="col">Año</th>
12              <th scope="col">Descripcion</th>
13              <th scope="col">Alquilado</th>
14          </tr>
15      </thead>
16      <tbody>
17          <tr *ngFor = "let vehiculo of vehiculos">
18              <th scope="row">{{vehiculo._id}}</th>
19              <td>{{vehiculo.brand}}</td>
20              <td>{{vehiculo.model}}</td>
21              <td>{{vehiculo.price}}</td>
22              <td>{{vehiculo.seats}}</td>
23              <td>{{vehiculo.year}}</td>
24              <td>{{vehiculo.description}}</td>
25              <td>{{vehiculo.rented}}</td>
26          </tr>
27      </tbody>
28  </table>
29  <app-footer> </app-footer>

```

Por último, se puede mostrar en una tabla con formato la información en la variable llamada vehículos, como es un array se puede mostrar gracias a ngFor, donde declaro una variable vehículo que será cada iteración correspondiente al array de vehículos (variable declarada con la data devuelta por la request)

All vehicles.

<div> <div>Agencia de Viajes</div> <div>Nuestros Vehiculos</div> <div>Nuestros Hoteles</div> <div>Nuestros Vuelos</div> <div>Ofertras Vehiculos</div> <div>Ofertras Hoteles</div> <div>Ofertras Vuelos</div> <div>Registro</div> <div>Login</div> <div>LogOut</div> </div>							
ID	Marca	Modelo	Precio	Asientos	Año	Descripcion	Alquilado
60d8ef32a3df8692e0401bfa	Mercedes	Vito	399	3	2000	furgoneta de mercedes	true
60d8ef32a3df8692e0401bfb	Ferrari	LaFerrari	200	2	2016	Gasolina 800cv	true
60d8ef32a3df8692e0401bfc	Fiat	Punto	50	4	2003	Diesel 75cv	true
60d8ef32a3df8692e0401bfd	Mercedes	Vito	80	7	2015	Diesel 110cv	true
60d8ef32a3df8692e0401bfe	Volvo	Gama FH	150	3	2020	Diesel 200cv	true
60d8ef32a3df8692e0401bff	Suzuki	GSX	90	2	2015	Gasolina 100cv	true
60d8ef32a3df8692e0401c00	Harley Davidson	Street Bob	100	1	2013	Gasolina 140cv	true
60d8ef32a3df8692e0401c01	Audi	R8	200	2	2020	Gasolina 360cv	true
60d8ef32a3df8692e0401c02	Beta	ARK	35	2	2001	Scooter Gasolina 4cv	true
60d8ef32a3df8692e0401c03	Peugeot	3008	95	5	2021	Diesel 140cv	true
60d90227c23098a39d1fe5eb	Ferrari	Italia	500	2	2020	Ferrari de prueba	false
60dc0e94e0675d7437c353042	Ferrari	Italia	500	2	2020	Ferrari de prueba	false
60e5950b9fc3fb39801c877d	Ferrari	Italia	500	2	2020	Ferrari de prueba	false
60e5a2934d13b64b47b9326	Ferrari	Italia	500	2	2020	Ferrari de prueba	false

[Login/Registro](#)

Vamos a ver ahora como estos componentes dan las funcionalidades y gestionan el token que se ha de almacenar en el navegador.

Primero el usuario rellena un formulario html con el formato UsuarioI

```

1  export interface UsuarioI
2  {
3
4      email: string;
5      displayName: string,
6      password: string
7  }
8  
```

La información de este formulario se pasa al método de agenciaservice.ts que hace la request al backend.

```

36  registro(form:UsuarioI):Observable<TokenI>
37  {
38
39      console.log("holaaa")
40      //en el form van los parametros (email, displayname, password)
41      let url = "https://localhost:3008/api/signup";
42
43      let direccion = url;
44      return this.http.post<TokenI>(direccion, form);
45  }
46  
```

El método recibe un form de tipo UsuarioI y devuelve un form de tipo TokenI

```
1  export interface TokenI
2  {
3      token: string;
4  }
```

```
14  export class RegistroComponent implements OnInit {
15
16      constructor(private api: AgenciaService, private router:Router) { }
17
18
19      registroForm = new FormGroup({
20          email: new FormControl('', Validators.required),
21          password: new FormControl('', Validators.required),
22          displayName: new FormControl('', Validators.required)
23      })
24
25
26      ngOnInit(): void {
27      }
28
29      errorStatus:boolean = false;
30      errorMsj: any = "";
31
32      registrarse(form: UsuarioI)
33      {
34
35          this.api.registro(form).subscribe(data =>
36          {
37              let dataResponse = data;
38
39              if(dataResponse.token != null)
40              {
41                  //si el api envia el token y todo ha ido bien almaceno el token
42                  localStorage.setItem("token", dataResponse.token);
43
44                  //y redirecciono al dashboard
45                  this.router.navigate(['dashboard']);
46              }
47          },
48          error =>
49          {
50              console.log('Ha llegado un error desde la api!!!! No se ha podido registrar');
51              this.errorStatus = true;
52              this.errorMsj = "Error inesperado";
53          });
54      }
55
56  }
```

En caso del registro se rellena el formulario y se envía como he explicado antes, en caso de que el usuario se reserve correctamente se almacena en la variable 'token' en el localStorage del navegador.



Uno de los errores que no he podido solucionar por falta de previsión en el proyecto, es que cuando catcheo un error en el front, en este caso que no se ha podido registrar el usuario, el código de error que devuelve el backend en la agencia es el status.(500), por lo que NO puedo mostrar el mensaje que lanzo en el body de la respuesta desde la agencia.

La causa es que cuando se devuelve un Internal Server Error, no se envía ningún body de respuesta, esto se puede arreglar enviando otro estado de error y mostrando el mensaje correspondiente.

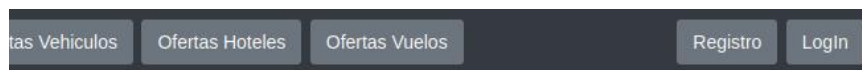
En este caso me he visto obligado a mostrar un mensaje genérico en el front de error, a pesar de que en la agencia si que discierno que tipo de error se ha producido y trato de mandar el mensaje más concreto de vuelta.

Una vez he almacenado en el localStorage el token redirijo al usuario a la página de inicio.

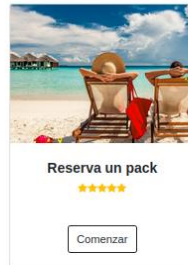
En el caso del login, compruebo si el token está ya almacenado en el navegador y en ese caso lo redirijo directamente al dashboard sin tener que iniciar sesión, cuando el usuario hace logout borro el token del navegador para así que la próxima vez tenga que iniciar sesión para realizar reservas.

```
39  revisarTokenLocalStorage()
40  {
41    if(localStorage.getItem('token'))
42    {
43      this.router.navigate(['dashboard']);
44    }
45  }
```

## Login



Token generado al hacer login



## Registro

Regístrate en la agencia para disfrutar de las mejores ofertas.

[Registrarse](#)

### [CancelarReserva/Consultar reserva](#)

Los dos componentes funcionan de forma muy similar por lo que los voy a agrupar en esta explicación, no hay gran diferencia respecto al componente anterior simplemente se ha de enviar el id de la reserva que se desee cancelar/consultar y estar autenticado con el token que nos da la agencia cuando nos registramos/iniciamos sesión y se devuelve la información de la reserva cancelada o activa.

```
29 search(form:IDReservaI)
30 {
31   let idreserva = form.idreserva
32
33   const token = localStorage.getItem('token')
34
35   if(token != null)
36   {
37     this.api.infoReserva(idreserva).subscribe(data =>
38     {
39       this.reservas = data;
40     },
41     error =>
42     {
43       console.log('Error desde la api no ha sido posible obtener la reserva ')
44       this.errorStatus = true;
45       //CUANDO LA API DEVUELVE STATUS 500(ERROR) NO ENVIA EL BODY, EN POSTMAN :
46       this.errorMsj = "Error, no se ha podido obtener la reserva"
47     });
48   }
49   else
50   {
51     console.log("NO HAY TOKEN ALMACENADO no tienes acceso")
52     console.log(token)
53     this.router.navigate(['/login'])
54   }
55 }
```

Buscar Reserva

Agencia de Viajes

Nuestros Vehiculos

Nuestros Hoteles

Nuestros Vuelos

Ofertas Vehiculos

Ofertas Hoteles

Ofertas Vuelos

Registro

Login

LogOut

60edd2a9a073613e1ce83bef

BUSCAR

ID	Fecha	Vehiculo	Hotel	Vuelo	Precio Total	Fecha Inicio	Fecha Fin	Ca
60edd2a9a073613e1ce83bef	2021-07-13T17:38:02.719Z		60d8f8af7916c1a29390074		5000	2021-07-12T22:00:00.000Z	2021-07-22T22:00:00.000Z	fa

En estos componentes he tomado de internet un componente que he agregado que permite seleccionar de manera limpia las fechas para mostrar/reservar las ofertas.

```

73 vehiculos:ListaVehiculosI[] = [];
74
75 async montarinfo()
76 {
77
78     var añoini = this.fromDate.year.toString();
79     var añofin = this.toDate?.year.toString();
80
81     var mesini = this.fromDate.month.toString();
82     var mesfin = this.toDate?.month.toString();
83
84     var diaini = this.fromDate.day.toString();
85     var diafin = this.toDate?.day.toString();
86
87
88     var fechaini = añoini + "-" + mesini + "-" + diaini;
89     var fechafin = añofin + "-" + mesfin + "-" + diafin;
90
91     var fech = fechaini + "/" + fechafin;
92
93
94     const token = localStorage.getItem('token')
95
96
97     if(token != null)
98     {
99         console.log(token)
100
101         this.api.getOffersVehicles(fech).subscribe(data =>
102         {
103             console.log(data);
104             this.vehiculos = data;
105         },
106         error =>
107         {
108             console.log('TOKEN INVALIDO NO TIENES ACCESO')
109             this.errorStatus = true;
110             this.errorMsj = "TOKEN INVALIDO NO TIENES ACCESO"
111         });
112     }
113
114     else
115     {
116         console.log("NO HAY TOKEN ALMACENADO no tienes acceso")
117         console.log(token)
118         this.router.navigate(['/login'])
119     }
120
121 }
122

```

La vista tiene un botón para mostrar la información de las reservas y despliega una tabla en la cual podemos pinchar en el botón reservar que nos mostrará mas abajo la información de la reserva si se ha podido realizar

```
124     reservas:ReservaI[] = [];
125     errorStatus:boolean = false;
126     errorMsj: any = "";
127
128     redirigeareserva(vehiculoid:String)
129     {
130
131         var añoini = this.fromDate.year.toString();
132         var añofin = this.toDate?.year.toString();
133
134         var mesini = this.fromDate.month.toString();
135         var mesfin = this.toDate?.month.toString();
136
137         var diaini = this.fromDate.day.toString();
138         var diafin = this.toDate?.day.toString();
139
140
141         var fechaini = añoini + "-" + mesini + "-" + diaini;
142         var fechafin = añofin + "-" + mesfin + "-" + diafin;
143
144         var fech = fechaini + "/" + fechafin;
145
146
147         this.api.reservarVehiculo(vehiculoid, fech).subscribe(data =>
148         {
149             this.reservas = data;
150             console.log(data);
151         },
152         error =>
153         {
154             console.log('Error desde la api no ha sido posible efectuar la reserva');
155             this.errorStatus = true;
156             //CUANDO LA API DEVUELVE STATUS 500(ERROR) NO ENVIA EL BODY, EN POSTMAN
157             //(NO SALDO) O (YA ESTA RESERVADO EL VEHICULO) POR LO QUE AQUI SIMPLEMENTE
158             this.errorMsj = "Error, no se ha podido efectuar la reserva"
159         });
160     }
161
162
```

Reserva Individual

<

Jul

2021

>

July 2021

August 2021

Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
			1	2	3	4							1
5	6	7	8	9	10	11	2	3	4	5	6	7	8
12	13	14	15	16	17	18	9	10	11	12	13	14	15
19	20	21	22	23	24	25	16	17	18	19	20	21	22
26	27	28	29	30	31		23	24	25	26	27	28	29
							30	31					

MostrarOfertas

ID	Marca	Modelo	Precio/Dia	Asientos	Año	Descripcion	Alquilado	
60d90227c23098a39d1fe5eb	Ferrari	Italia	500	2	2020	Ferrari de prueba	false	Reservar
60dce84e0675d7437c353042	Ferrari	Italia	500	2	2020	Ferrari de prueba	false	Reservar
60e595b9f9c3fb39801c677d	Ferrari	Italia	500	2	2020	Ferrari de prueba	false	Reservar
60e5a2934d13b64bb47b9326	Ferrari	Italia	500	2	2020	Ferrari de prueba	false	Reservar

ID	Fecha	Vehiculo	Hotel	Vuelo	Precio Total	Fecha Inicio	Fecha Fin	Cancelado
60edd222a073613e1ce83bea	2021-07-13T17:38:02.719Z	60d90227c23098a39d1fe5eb			5000	2021-07-12T22:00:00.000Z	2021-07-22T22:00:00.000Z	false

## Reservarpack

Por último, tenemos la reserva de un pack, la lógica del componente se comporta igual que los componentes de reserva individual.

Método que muestra las ofertas:

```
115     if(token != null)
116     {
117         this.api.getOffersVehicles(fech).subscribe(data =>
118         {
119             console.log(data);
120             this.vehiculos = data;
121         },
122         error =>
123         {
124             console.log('TOKEN INVALIDO NO TIENES ACCESO')
125             this.errorStatus = true;
126             this.errorMsj = "TOKEN INVALIDO NO TIENES ACCESO"
127         });
128
129         this.api.getOffersHotels(fech).subscribe(data =>
130         {
131             console.log(data);
132             this.hoteles = data;
133         },
134         error =>
135         {
136             console.log('TOKEN INVALIDO NO TIENES ACCESO')
137             this.errorStatus = true;
138             this.errorMsj = "TOKEN INVALIDO NO TIENES ACCESO"
139         });
140
141
142         this.api.getOffersFlights(fech).subscribe(data =>
143         {
144             console.log(data);
145             this.vuelos = data;
146         },
147         error =>
148         {
149             console.log('TOKEN INVALIDO NO TIENES ACCESO')
150             this.errorStatus = true;
151             this.errorMsj = "TOKEN INVALIDO NO TIENES ACCESO"
152         });
153     }
154     else
155     {
156         console.log("NO HAY TOKEN ALMACENADO no tienes acceso")
157         console.log(token)
158         this.router.navigate(['/login'])
159     }
160 }
```

Método que llama al back para realizar las reservas

```
171   redirigeareserva(form: PackI)
172   {
173
174     console.log(form);
175
176     var añoini = this.fromDate.year.toString();
177     var añofin = this.toDate?.year.toString();
178
179     var mesini = this.fromDate.month.toString();
180     var mesfin = this.toDate?.month.toString();
181
182     var diaini = this.fromDate.day.toString();
183     var diafin = this.toDate?.day.toString();
184
185
186     var fechaini = añoini + "-" + mesini + "-" + diaini;
187     var fechafin = añofin + "-" + mesfin + "-" + diafin;
188
189     var fech = fechaini + "/" + fechafin;
190
191     let vehiculoid = form.idvehiculo
192     let hotelid = form.idhotel
193     let flightid = form.idvuelo
194
195     console.log(vehiculoid)
196     console.log(hotelid)
197     console.log(flightid)
198
199
200     this.api.reservarPack(vehiculoid, hotelid, flightid, fech).subscribe(data =>
201     {
202       this.reservas = data;
203       console.log(data);
204     },
205     error =>
206     {
207       console.log('Error desde la api no ha sido posible efectuar la reserva ')
208       this.errorStatus = true;
209       //CUANDO LA API DEVUELVE STATUS 500(ERROR) NO ENVIA EL BODY, EN POSTMAN SI
210       //(NO SALDO) O (YA ESTA RESERVADO EL VEHICULO) POR LO QUE AQUI SIMPLEMENTE
211       this.errorMsj = "Error, no se ha podido efectuar la reserva"
212     });
213   }
```

La única diferencia respecto al html es que en el componente anterior la lista tenía botones para seleccionar el objeto a reservar y ahora tiene una rejilla de selección (donde se selecciona un objeto de cada tipo y se mandan pinchando en el botón reservar).

## Reservar Pack

Agencia de Viajes Nuestros Vehiculos Nuestros Hoteles Nuestros Vuelos Ofertas Vehiculos Ofertas Hoteles Ofertas Vuelos

Registro Login LogOut

< Jul 2021 2021 >

July 2021 August 2021

Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
					1	2	3	4					1
5	6	7	8	9	10	11	2	3	4	5	6	7	8
12	13	14	15	16	17	18	9	10	11	12	13	14	15
19	20	21	22	23	24	25	16	17	18	19	20	21	22
26	27	28	29	30	31		23	24	25	26	27	28	29
							30	31					

MostrarOfertas

ID	Marca	Modelo	Precio/Dia	Asientos	Año	Descripcion	Alquilado		
60dce84e0675d7437c353042	Ferrari	Italia	500	2	2020	Ferrari de prueba	false	<input checked="" type="radio"/>	Seleccionar
60e595b99c3fb39801c677d	Ferrari	Italia	500	2	2020	Ferrari de prueba	false	<input type="radio"/>	Seleccionar
60e5a2934d13b64bb47b9326	Ferrari	Italia	500	2	2020	Ferrari de prueba	false	<input type="radio"/>	Seleccionar

ID	Nombre	Tipo	Precio/Dia	Telefono	Dirección	País	Ciudad	CodPostal	Alquilado		
60d8ffaf7916c1a29390f070	NH Collection León Plaza Mayor	individual	83	849856789	Plaza Mayor 15	España	León	24000	false	<input checked="" type="radio"/>	Seleccionar
60d8ffaf7916c1a29390f074	Kakslautanen Arctic Resort	triple	500	1498541365149	Kaipsonk 513	Finlandia	Kakslautanen	894913191	false	<input type="radio"/>	Seleccionar
60d900d579fbf1a2bb7edf7b	Hotel NH Eida	individual	50	68491494	Calle Murillo 23	España	Eida	03610	false	<input type="radio"/>	Seleccionar
60e5a19c143d0049fd499e8b	Hotel NH Eida	individual	50	68491494	Calle Murillo 23	España	Eida	03610	false	<input type="radio"/>	Seleccionar

ID	Tipo	Compañia	Precio	Avión	Duración	Origen	Destino	Descripcion	Lleno		
60dce94b9a85d343e126fbb3	directo	Ryanair	35	Boeing2124	35	Alicante	Ibiza	Vuelo directo Alicante-Ibiza	false	<input checked="" type="radio"/>	Seleccionar
60e5a00b5eb266464de9337a	directo	Ryanair	35	Boeing2124	35	Alicante	Ibiza	Vuelo directo Alicante-Ibiza	false	<input type="radio"/>	Seleccionar
60e5a0c55eb266464de93384	directo	Ryanair	35	Boeing2124	35	Alicante	Ibiza	Vuelo directo Alicante-Ibiza	false	<input type="radio"/>	Seleccionar

Reservar

ID	Fecha	Vehiculo	Hotel	Vuelo	Precio Total	Fecha Inicio	Fecha Fin	Cancelada
60edd240a073613e1ce83bec	2021-07-13T17:38:02.719Z	60dce84e0675d7437c353042	60d8ffaf7916c1a29390f070	60dce94b9a85d343e126fbb3	5865	2021-07-12T22:00:00.000Z	2021-07-22T22:00:00.000Z	false

Por último que para que los certificados openssl no entren en conflicto con los navegadores y la aplicación de angular conviene arrancarla con la directiva :

Ng serve --ssl --ssl-key <path> --ssl-cert <path>

En mi caso para simplificar el lanzamiento de la aplicación simplemente lanzo:

Ng -s -o ---ssl true

Se autogeneran los certificados en caso de que no existan así se puede lanzar más fácilmente.



## Cumplimiento de requisitos

Los requisitos exigidos en la práctica en función de los bloques que considero que he cumplido:

- Arquitectura:
  - o Módulo por cada entidad
  - o Despliegue en 3 equipos físicos distintos (rezando para el viernes estoy)
  - o Solución tolerante a fallos (al apagar un proveedor no se cae la app y el redesplicue es independiente, los mensajes de error son trazables y adecuados)
  - o Integración con 2 tecnologías (Api Rest con mensajes JSON (Agencia-Proveedores) (Agencia-Front), Cliente-Servidor Socket TCP con mensajes entre Agencia y Banco)
- Seguridad:
  - o Autenticación y autorización (SALT en la agencia y verificación por token en la agencia y en el front)
  - o Cifrado openssl en la Agencia hacia Proveedores y en el Front
- Transacciones:
  - o Estrategia definida (patrón saga orquestado, coherencia y trazabilidad)
  - o Creación de un módulo específico para transacciones.
- Front-End:
  - o Completamente desacoplado gracias a Angular
  - o Acceso a todas las funcionalidades destinadas al usuario

## Posibles mejoras

- Gestión de las reservas con un Array de Pares para así poder reservarlos en distintas fechas simultáneamente
- Paso y almacenamiento de token a través de cookies
- Implementación de Sistema Bancario más realista (con modelos de tarjetas de crédito etc...)
- Certificados firmados por una entidad real
- Reinicio automático de proveedores en caso de caída
- Encriptado de contraseñas en el formulario del front

## Despliegue

En cuanto al despliegue de la aplicación adjunto un fichero **README.txt** con todas las instrucciones tanto de bajada de proyecto como de despliegue del mismo en un entorno local, en una máquina con Ubuntu con las versiones especificadas en la introducción.

Mi estrategia para desplegar en el aula el proyecto será tratar de hacerlo igual que en local, simplemente cambiando las ip de las rutas en los proyectos, desplegando en un ordenador los proveedores, en otro la agencia y el banco, y por último el front en otro ordenador. Probablemente esto no suceda ya que como los ordenadores están capados puede haber algún tipo de error ya sea con las instancias de la bbdd etc.

Como plan B voy a montar 3 máquinas virtuales idénticas al entorno en el que he desarrollado el proyecto para así lanzarlas en los ordenadores del aula, esto es mucho mas rápido que la primera opción pero para ello ha de haber VirtualBox en los ordenadores del aula y también he de redirigir los puertos de las tarjetas de red de las máquinas hacia las máquinas virtuales que yo lleve.

Antes de la presentación realizaré algunas comprobaciones en los ordenadores de la escuela para comprobar que es posible realizar el despliegue.

# Bibliografía

<https://www.mongodb.com/blog/post/quick-start-nodejs--mongodb--how-to-implement-transactions>

<https://ng-bootstrap.github.io/#/components/daterangepicker/examples>

<https://www.youtube.com/watch?v=1XThP-q9jI4>

<https://hackernoon.com/set-up-ssl-in-nodejs-and-express-using-openssl-f2529eab5bb>

<https://flaviocopes.com/express-https-self-signed-certificate/>

<https://www.linuxito.com/seguridad/598-como-crear-un-certificado-ssl-autofirmado-en-dos-simples-pasos>

<https://ciphertrick.com/salt-hash-passwords-using-nodejs-crypto/>

<https://es.quora.com/Qu%C3%A9-significa-agregar-un-salt-a-una-contrase%C3%B1a-hash>

[https://www.youtube.com/watch?v=6qR\\_EpxadMo](https://www.youtube.com/watch?v=6qR_EpxadMo)

<https://github.com/ericvicenti/csr-gen/blob/master/README.md>

<https://www.sitepoint.com/indexeddb-store-unlimited-data/>

[https://www.youtube.com/watch?v=l\\_r9nRJ9YTk](https://www.youtube.com/watch?v=l_r9nRJ9YTk)

<https://unpocodejava.com/2020/01/02/que-es-el-patron-saga/>

<https://www.youtube.com/watch?v=7nafaH9SddU>

<https://masteringsjs.io/tutorials/mongoose/find-by-id>

[https://dev.to/dev\\_tycodez/seed-mongodb-mongoose-seed-51d5](https://dev.to/dev_tycodez/seed-mongodb-mongoose-seed-51d5)

<https://www.npmjs.com/package/mongoose-seed>

<https://www.dev2qa.com/node-js-tcp-socket-client-server-example/>

<https://www.youtube.com/watch?v=UjH7hw9fWWQ>

<https://carlosazaustre.es/websockets-como-utilizar-socket-io-en-tu-aplicacion-web>

<https://medium.com/@carlosazaustre/usando-websockets-con-nodejs-y-socketio-b02f66bcb58d>

<https://www.youtube.com/watch?v=fBQe3ZRPVdI>