


# Desplegament del servei Web i definició de protocols



# Índex...

<b>Descripció general Sprint</b>	<b>2</b>
<b>Desplegament del servei web</b>	<b>4</b>
Descripció general	4
Explicació software/hardware utilitzat	4
Manual d'usuari	4
Conclusions	4
<b>Definició de protocols</b>	<b>5</b>
Descripció general	5
Explicació/Justificació	5
Conclusions	5

En Blau el que estava  
en l'anterior Sprint 

## Descripció general Sprint 1

//Descripció global de les tasques realitzades i l'organització del grup. Part del projecte realitzada, estructuració del treball...

En aquest sprint ens hem centrat principalment en acabar el que havíem començat en l'anterior Sprint, és a dir el disseny programat de les pàgines. Apart del disseny hem començat a enllaçar-les, això ens permet tractar-la com una web local i identificar com seria la navegació entre elles per detectar possibles errades de usabilitat.

Gràcies a les reunions de SMs i de POs ens n'hem assabentat que el servidor del que dependrem serà Linux. Així que hem començat adaptar la API a Linux.

## Descripció general Sprint 2

Per aquest Sprint hem decidit acabar d'implementar el disseny programant-lo localment. Es a dir tenim gairebé tot el disseny de les dues pàgines tant la d'administrador com la d'usuari implementades.



The screenshot shows a web application interface titled "Gestión de alertas del Aeropuerto". It contains two tables. The first table lists flight alerts with columns: Vuelo, Aerolínea, Tipo, Estado, and Comentarios. The second table lists car rental alerts with columns: Número coche autónomo, Estado, and Comentarios. Both tables have "Modificar" and "Añadir" buttons below them.

Vuelo	Aerolínea	Tipo	Estado	Comentarios
FR6375	Ryanair	Salida	OK	-
QR335	QATAR Airways	Urgente	NOK	Vuelo con retraso
FR6378	Ryanair	Salida	OK	-
FR6379	Ryanair	Salida	OK	-

[Modificar](#) [Añadir](#)

Número coche autónomo	Estado	Comentarios
CA-VIA0009	Libre	-
CA-VIA0023	Ocupado	Desplazando pasajeros
Número de pasajeros	Puerta destino	Duración estimada del viaje
3 personas	12 salida	4 minutos
CA-VIA0007	Avariado	En reparación
CA-VIA0017	Libre	-

[Modificar](#) [Añadir](#)

Sobre l'API l'idea es començar-la a migrar a Linux per facilitar-ne la inclusió al servidor que ens proporioni el grup de la Terminal, ja que ens han confirmat que el final el servidor correrà sobre Linux.

# Desplegament del servei web

## Descripció general

//Descripció més .0

detallada i concreta d'aquesta part del treball.

El servei web està dividit en dos parts en la part d'usuari enfocada al consumidor on es poden realitzar consultes amb fins informatius com botigues, restaurants, mapes... I una altre part d'administrador per els operaris on es podran fer gestions de cotxes, passatgers i avions.

### Web Administrador:

En iniciar la web ens encarem amb un login d'usuari i contrasenya per a poder accedir a les següents funcions:

- **Alertes:** en aquest apartat informarem sobre les alertes de vols i dels cotxes autònoms dels quals disposa en l'aeroport, així com de l'administració de tots dos. D'una banda estan els vols amb les seves respectives columnes, de les quals la més important seria l'estat de l'avió, ja que en cas que estigui en mal estat hauríem de modificar-ho amb un petit comentari que estaria amagat per un desplegament. Més a baix trobarem l'administració dels cotxes autònoms, on disposarem dels noms dels diferents cotxes, el seu estat i quina és l'acció que s'està realitzant en ell.

En aquest Sprint esta

- **Mapa:** en entrar en Mapa trobarem el mapa de l'aeroport per a poder realitzar un filtrat dels passatgers. Cal dir que només coneixerem l'ubicació dels passatgers que tinguin descarregada l'aplicació i que estiguin registrats en la base de dades. D'aquesta manera podem fer un filtrat de l'ubicació de diferents passatgers separats per aerolínies, per la mena de vol o si volem una mica més genèric, per plantes.

- **Administració de cotxes:** en aquesta funció el que farem serà administrar tots els cotxes dels quals disposem. És a dir, haurem de canviar l'estat de cada cotxe, informant així de si està ocupat, lliure o avariats. En cas d'estar ocupat hem d'informar que passatgers estan sent transportats.

### - Web Client:

- **Pàgina principal:** simplement és la pàgina principal de la web, la qual ens informará de tot el que

sigui necessari, així com també ens proporcionarà els diferents serveis que demandem.

A la cantonada dreta de la part de dalt tindrem: Serveis, Mapa, Accessos i contacte.

**a) Serveis:** aquest apartat conté els diferents serveis comercials dels quals disposa l'aeroport.

Estan classificats per la mena de comerç que són i una vegada entrant en un d'ells ens proporciona el tipus de franquícies que hi ha així com la ubicació de cada botiga.

**b) Mapa:** no té una funció molt avançada, simplement ens mostrarà el mapa per plantes de l'aeroport.

**c) Accessos:** si cliquem en aquest enllaç ens portarà a una pestanya que contindrà un requadre. Aquest requadre que ens ofereix els recursos de Google Maps, ens demanarà una ubicació inicial i una ubicació final, la qual haurem de triar en el desplegament. Una vegada tot col·locat correctament ens mostrarà el mapa amb la ruta en color i la ruta escrita perquè ho entenguem millor.

Tornem a la pàgina principal. En aquest cas podem consultar els vols que vulguem. Podem consultar un vol en concret o tots els vols que arribin o surtin del nostre aeroport. Si continuem baixant, trobarem apartats informatius sobretot, com el mapa, els cotxes autònoms, una altra pestanya de com arribar...

## Explicació software/hardware utilitzat

Per programar el servei web de l'aeroport hem fet servir diferents entorns de programació, llenguatges de programació, frameworks i serveis de pàgines web.

Per picar codi hem utilitzat principalment dos editors de text bastant coneguts que són Sublime Text y Visual Studio Code.

En quant a llenguatges de programació i frameworks per fer l'estructura bàsica de la pàgina hem fet servir HTML, CSS, Javascript i Bootstrap 4 què és un framework CSS desenvolupat per Twitter que permet donar forma a un lloc web mitjançant llibreries CSS que inclouen tipografies, botons, quadres, menús i altres elements que poden ser utilitzats en qualsevol lloc web.

A més a més també hem fet ús de diversos widgets, icones i fotografies gratuïtes. Entre les quals es troben :

**-W3.CSS Icons:** Llibreria d'icones gratuïts de W3Schools.

**-Ionic icons:** Pàgina web d'icones.

**-Flaticons:** Pàgina que ofereix fotos en format png.

**-Pexels:** Servei que ofereix fotos de stock.

**-Weatherwidget:** Widget responsive sobre el clima en temps real de qualsevol ubicació.

## Manual d'usuari

//Interaccions que pot realitzar l'usuari amb el projecte actual.

Si es descàrrega l'arxiu el pot tractar com una pàgina web normal, tot i que funcionarà en local.

Web Administrador:

El nom d'usuari i la contrasenya no son importants pot posar qualsevols.

Un cop entri clicant sobre l'entrar veurà un menú amb diferents opcions.

Aquí pot clicar sobre Alertas, Mapa o Gestió de cotxes.

Si clica sobre Alertes apareixeran dues taules amb informació, algun dels apartats tindrà un botó desplegable, en aquest trobarem informació més extensa sobre el que volem saber.

En la pestanya de Mapes trobem el disseny d'una pestanya amb 3 menús desplegables quan fas clic sobre ells, en el que en un futur podràs filtrar el mapa per passatgers segons es vulgui filtrar per aereolinea, tipus de vol i/o la planta de l'aeroport on es troben .

I per acabar amb la web d'administrador trobem la pestanya de gestión de coches on trobem una taula amb informació sobre cotxes on en un futur es podràn ordenar per número de cotxe, de passatger o estat en el que es trobin.

Web Usuari:

Troblem 4 apartats principals:

Servicios: On trobarem un llistat dels diferents serveis que ofereix l'aeroport i podrem obrir cada un d'ells amb més detall

Mapa: Trobem un mapa de l'aeroport

Accessos ens porta a una Api de google maps on podrem veure el camí a seguir fins arribar a l'aeroport

Contacto en un futur trobarem informació sobre l'aeroport i com posar-se en contacte amb el diferents departaments.

## Conclusions

Creiem que com a primera aproximació ens ha servit per a familiaritzar-nos amb l'ús de les APIs al món real, a la espera de poder connectar-la amb la base de dades i APP reals del grup B2.

# Desplegament de la API

## Descripció general

//Descripció més detallada i concreta d'aquesta part del treball.

## Explicació software/hardware utilitzat

Per realitzar la API hem utilitzat diferents eines que són les següents:

- **Node.js**: Entorn que treballa en temps real d'execució, de codi obert, multiplataforma que permet al programador crear tota classe d'eines del costat servidor (BackEnd) i aplicacions en Javascript.  
Això ens permetrà crear una API REST que representi les operacions que podem tenir en una aplicació Web i realitzar operacions sobre qualsevol repositori de dades (Base de dades, fitxers o qualsevol altre).
- **Fitxer package.json**: Es crea en iniciar el projecte (`npm init --yes`) i conté totes les dependències que necessitem, les quals instal·larem conforme anem necessitant i es guardaran aquí.
- **Servidor Express JS**: Framework de Node, crea l'estructura del servidor i ens permet escriure codi de manera senzilla.
- **Body-Parser**: Mòdul que ens permet convertir les dades que ens arriben en les peticions al servidor, en objectes amb format JSON.
- **Mongoose**: Mòdul que ens proveeix mètodes i funcionalitats per a treballar millor amb MongoDB.
- **Morgan**: Mòdul que ens permet veure per consola les peticions que arriben.
- **Nodemon**: Mòdul que farà que amb qualsevol canvi en el servidor, aquest es reiniciï (fa com una compilació), a més de que mostra per consola totes les peticions HTTP (GET, POST, PUT, DELETE...) que rep el servidor (200, 404, 500, etc.)
- **POSTMAN** és un programari que simula peticions a la API de rutes, des d'una aplicació web/mòbil. Ens cap afegir capçalera en Extensió per a Firefox: Content-Type: application/json

## Manual d'usuari

//Interaccions que pot realitzar l'usuari amb el projecte actual.

En aquest primer prototip, l'usuari pot llistar, afegir, actualitzar i eliminar usuaris, així com llistar, afegir i eliminar vols, tot això simplement tenint corrent el servidor i una base de dades MongoDB. En aquest cas, gràcies al mòdul Mongoose, em connectat el servidor amb una base de dades en local per a fer proves de peticions. L'usuari simplement haurà d'obrir un terminal i inserir la comanda *mongod*:

```
C:\Users\hek_b\Documents\UNI\Tercero\Q6\PTIN\Proyecto VIA\Sprint1\APIRESTNode>mongod
2020-03-29T10:45:38.082+0200 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify
2020-03-29T10:45:38.089+0200 I CONTROL [initandlisten] MongoDB starting : pid=16708 port=27017 dbpath=C:\data\
2020-03-29T10:45:38.090+0200 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2020-03-29T10:45:38.090+0200 I CONTROL [initandlisten] db version v4.2.3
2020-03-29T10:45:38.090+0200 I CONTROL [initandlisten] git version: 6874650b362138df74be53d366bbefc321ea32d4
2020-03-29T10:45:38.090+0200 I CONTROL [initandlisten] allocator: tcmalloc
2020-03-29T10:45:38.090+0200 I CONTROL [initandlisten] modules: none
2020-03-29T10:45:38.091+0200 I CONTROL [initandlisten] build environment:
2020-03-29T10:45:38.091+0200 I CONTROL [initandlisten] distmod: 2012plus
2020-03-29T10:45:38.091+0200 I CONTROL [initandlisten] distarch: x86_64
2020-03-29T10:45:38.091+0200 I CONTROL [initandlisten] target_arch: x86_64
```

I des d'un altre terminal, arrancar el servidor Express:

```
C:\Users\hek_b\Documents\UNI\Tercero\Q6\PTIN\Proyecto VIA\Sprint1\APIRESTNode>node src/index.js
Servidor Express escuchando en el puerto 3000
Base de datos conectada
```

Si l'usuari havia arrencat primer la base de dades, apareixerà el missatge de Base de dades connectada, si l'arrenca després d'arrencar el servidor Express, llavors el missatge apareixerà en el moment en què aquesta es connecti.

Un cop està tot arrencat, l'usuari ja pot fer proves amb les dades disponibles. En aquesta primera aproximació del prototip, hem creat dos models de dades, un per dades d'usuaris (gestors) i un altre per dades dels vols. Els paràmetres que es llisten per cadascun dels grups es reduït per el moment, però es podrà ampliar de cara a següents Sprints.

D'un usuari es tindran les següents dades:

- Nom
- Cognom
- Rol [Usuari / Administrador]

D'un vol, per altre costat, es tindran les següents dades:

- Número de vol
- Origen
- Destí



- Aerolinea (Només disponibles les aerolínies que actualment fan ús de l'aeroport) [Ryanair / QATAR Airways / Vilanova Airlines / Vueling / Iberia]
- Estat de vol [En hora / Retrassat / Cancel·lat]

Com a increment en aquest Sprint, s'ha adaptat un tercer model per a Passagers:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const pasajeroSchema = new Schema({
  id: String,
  password: String // Se podrían añadir más datos si fuese necesario
});

module.exports = mongoose.model('Pasajero', pasajeroSchema);
```

Aquest model únicament té, pel moment, com a paràmetres id i password.

Aquest model s'ha creat degut a la petició del grup B2, encarregat de la Base de dades, per a què, quan connectem base de dades i API, en el següent Sprint (com s'ha acordat per necessitats de l'altre equip) es pugui comprovar si un passager existeix o no.

Això ja s'ha provat amb POSTMAN en local i està funcionant actualment.

Podem comprovar amb el servidor encés, mitjançant la comanda GET i la ruta indicada per a passagers, els passagers que hi han a la base de dades (entrats anteriorment manualment per a fer proves):

GET http://localhost:3000/api/pasajeros/ Send

Params Auth Headers (10) Body ● Pre-req. Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (6) Test Results 200 OK 457 ms 435 B Save

Pretty Raw Preview Visualize JSON ⌵

```

1  [
2    {
3      "_id": "5e919fe428e7753b883121d4",
4      "id": "555D",
5      "password": "password123",
6      "__v": 0
7    },
8    {
9      "_id": "5e91a279dc65dd2f4cc328d1",
10     "id": "555T",
11     "password": "password123",
12     "__v": 0
13   }
14 ]
  
```

Per altra banda s'ha adaptat una comprovació, a petició del grup B2 per a comprovar si un passager està o no a la base de dades:

Observem que si fem una petició GET a la ID generada d'un passager existent, ens retorna el seu ID (que seria el DNI) i la seva contrasenya:

GET http://localhost:3000/api/pasajeros/5e91a279dc65dd2f4cc328d1 Send

Params Auth Headers (10) Body ● Pre-req. Tests Settings

raw JSON ⌵

```

1 {
2   "id": "555T",
3   "password": "password123"
4 }
  
```

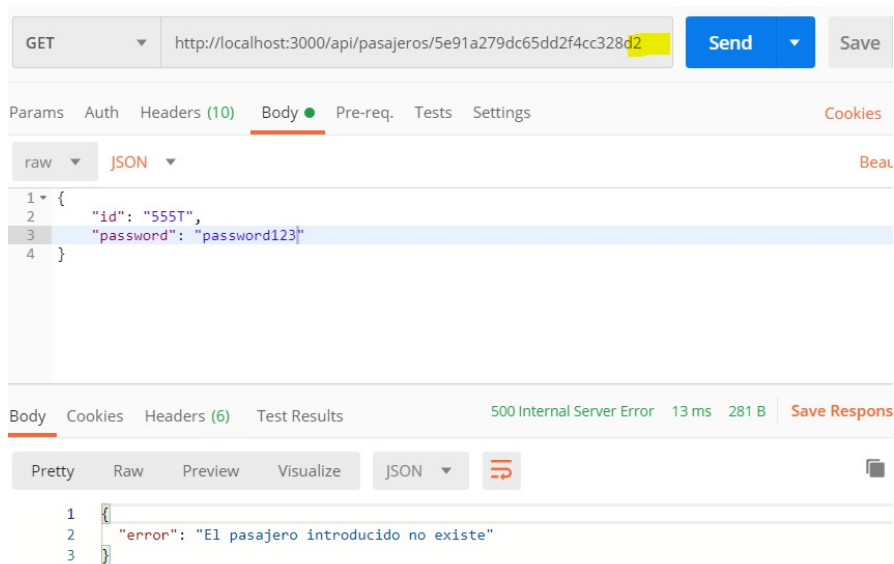
Body Cookies Headers (6) Test Results 200 OK 55 ms 308 B

Pretty Raw Preview Visualize JSON ⌵

```

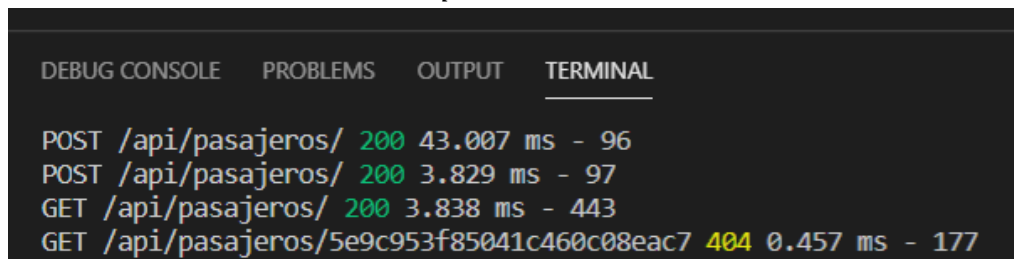
1 {
2   "_id": "5e91a279dc65dd2f4cc328d1",
3   "id": "555T",
4   "password": "password123",
5   "__v": 0
6 }
  
```

per altra banda, si s'entra un ID no existent, ens retornaria el següent missatge:



Com a detall, observem que s'ha programat per a que sempre ens mostri el nombre de resposta HTTP, encara que POSTMAN ho mostra sempre. En aquest últim cas resposta 500, i al primer cas 200 OK.

*Al servidor anem veient totes les peticions:*



De cara al proper Sprint, a més de fer la connexió amb la base de dades i les proves que això comportarà, s'està adaptant el mòdul de vols per a poder comprovar si un passager està o no en aquell vol, es a dir, un mòdul anidat. Això s'ha de terminar d'adaptar, així com adaptar bé les dades que venen de la base de dades i amb les que treballarem, per exemple, al mòdul de vols saber què necessitem tenir (origen i destinació, numero de vol, aerolinia, nombre de passagers... etc). Pel moment s'estàn fent proves per adaptar la API a Linux (Ubuntu) per a simular la connexió que es farà a Debian segons ens han dit els altres grups i així ens anticipem a possibles errors abans de temps.

En els dos casos, pels paràmetres que tenim opcions definides, només veurem alguna d'aquestes opcions, ja que a l'hora afegir noves dades, si en aquests camps s'intenta afegir una opció diferent, donarà error.

A més, cada cop que s'afegeix una nova dada (usuari o vol) es genera un ID únic, i amb aquest ID es pot modificar o esborrar aquest camp.

El servidor Express utilitzarà el port 3000, a no ser que hi hagi un port definit, i pel moment, l'usuari podrà accedir-hi a les dades, simplement entrant a un navegador i escrivint l'enllaç <http://localhost:3000/api/users> ó <http://localhost:3000/api/vuelos> respectivament, pel moment no tenim definida la ruta arrel (<http://localhost:3000>) que permet-hi accedir des d'aquí als dos conjunts de dades (i als que puguin haver-hi pròximament), però es farà en una propera implementació. Als dos enllaços també es pot accedir-hi sense posar la paraula api a l'enllaç i les dades es mostraran igualment, però per conveni s'acostuma a utilitzar i per això hem afegit aquesta opció.

Des del navegador l'usuari podrà llistar (corresponent a una petició HTTP GET), però no podrà fer res més, qualsevol inserció, modificació o eliminació (corresponents a peticions HTTP POST, PUT i DELETE), l'haurà de fer des del software POSTMAN, un cop afegida la capçalera indicada i el tipus de dades.

Per exemple, l'usuari administrador vol afegir un nou vol:

POST http://localhost:3000/api/vuelos + ... No Environment

Untitled Request Co

POST http://localhost:3000/api/vuelos Send

Params Auth Headers (10) **Body** Pre-req. Tests Settings

raw JSON

```

1 {
2   "numvuelo": "VY1234",
3   "origen": "BGY",
4   "destino": "VIA",
5   "aerolinea": "Vueling",
6   "estado": "En hora"
7 }

```

un cop envia la dada:

Body Cookies Headers (6) Test Results 200 OK 69ms 376 B Save Response

Pretty Raw Preview Visualize JSON ≡

```

1 {
2   "_id": "5e806663c84c7a3ea40eb4ec",
3   "numvuelo": "VY1234",
4   "origen": "BGY",
5   "destino": "VIA",
6   "aerolinea": "Vueling",
7   "estado": "En hora",
8   "__v": 0
9 }

```

Al peu de POSTMAN veiem que la dada s'ha inserit bé (codi 200 HTTP).

També ens ho mostrarà el nostre Servidor (gràcies a tenir el mòdul Nodemon):

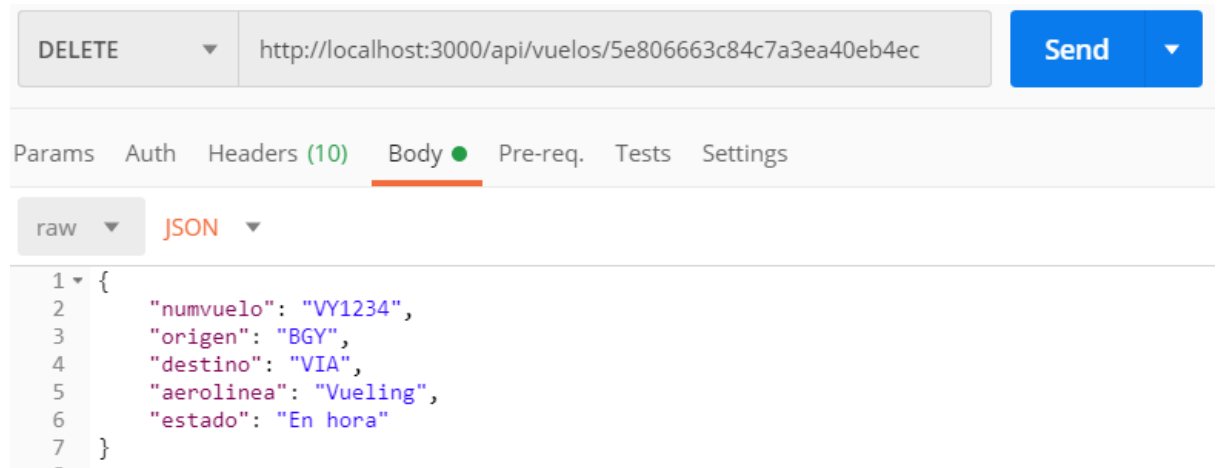
```
POST /api/vuelos 200 48.348 ms - 163
```

I ara podem llistar la nova dada des del navegador, o fent una petició GET des del mateix POSTMAN:

```
localhost:3000/api/vuelos
{
  "_id": "5e806663c84c7a3ea40eb4ec",
  "numvuelo": "VY1234",
  "origen": "BGY",
  "destino": "VIA",
  "aerolinea": "Vueling",
  "estado": "En hora",
  "__v": 0
}
```

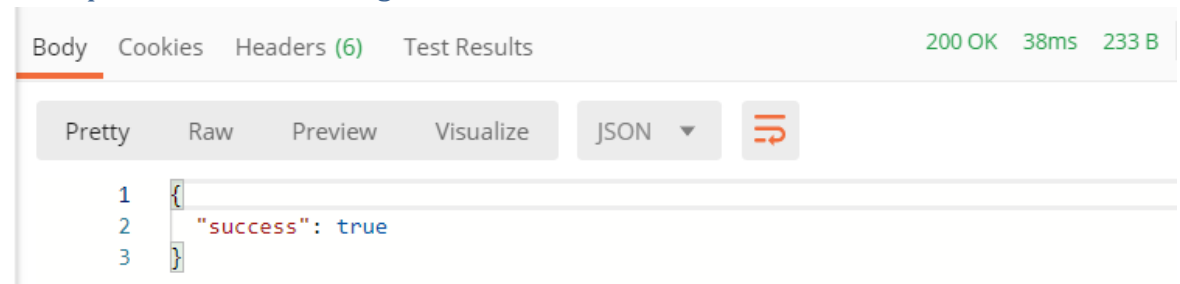
En aquest cas, podem veure des del navegador la dada inserida amb els seus paràmetres i la ID generada.

Ara que tenim la ID, des de POSTMAN es pot fer una modificació o eliminació d'aquesta dada (només capturarem una de les dues ja que el procés es idèntic):



En aquest cas es fa una eliminació (petició DELETE), hem de fixar-nos que a l'enllaç, s'afegit la ID, per a que el servidor pugui saber quina dada s'eliminarà.

Un cop enviada veiem el següent:



La dada s'ha eliminat correctament, i ara al fer un visionat dels vols, veiem que ja no apareix aquest:

localhost:3000/api/vuelos

A part d'això, com a administradors podem fer un seguiment de les dades desde la propia base de dades. Hem afegit uns quants usuaris i uns quants vols com s'ha explicat a dalt i hem obert una altra terminal (tot això amb el servidor Express i la base de dades arrencats, es clar).

En aquest cas ens connectem a la base de dades mitjançant la comanda **mongo** i amb algunes comandes típiques del SGBD MongoDB, podem llistar les bases de dades que tenim actualment, els tipus de dades i les pròpies dades:

```
> show dbs;
admin                0.000GB
config               0.000GB
local                0.000GB
rest-api-prueba     0.000GB
> use rest-api-prueba;
switched to db rest-api-prueba
> show collections;
usuarios
vuelos
```

```
> db.usuarios.find();
{ "_id" : ObjectId("5e7c851ae194b253b078e206"), "nombre" : "Héctor", "apellido" : "Montesinos", "rol" : "Administrador", "__v" : 0 }
{ "_id" : ObjectId("5e805ec9191ee0481ce2ac06"), "nombre" : "Nil", "apellido" : "Blanca", "rol" : "Administrador", "__v" : 0 }
{ "_id" : ObjectId("5e805ed7191ee0481ce2ac07"), "nombre" : "Josep", "apellido" : "Marches", "rol" : "Usuario", "__v" : 0 }
{ "_id" : ObjectId("5e805ee0191ee0481ce2ac08"), "nombre" : "Roger", "apellido" : "Tarrès", "rol" : "Usuario", "__v" : 0 }
{ "_id" : ObjectId("5e805ef2191ee0481ce2ac09"), "nombre" : "Javi", "apellido" : "Delgado", "rol" : "Administrador", "__v" : 0 }
```

```
> db.vuelos.find();
{ "_id" : ObjectId("5e805e65191ee0481ce2ac05"), "numvuelo" : "VB1234", "origen" : "
VIA", "destino" : "BCN", "aerolinea" : "Vilanova Airlines", "estado" : "En hora", "__
v" : 0 }
{ "_id" : ObjectId("5e805f77191ee0481ce2ac0a"), "numvuelo" : "QR1234", "origen" : "
VIA", "destino" : "DOH", "aerolinea" : "QATAR Airways", "estado" : "Cancelado", "__
v" : 0 }
{ "_id" : ObjectId("5e805fa1191ee0481ce2ac0b"), "numvuelo" : "FR0012", "origen" : "
DUB", "destino" : "VIA", "aerolinea" : "Ryanair", "estado" : "Retrasado", "__v" : 0
}
{ "_id" : ObjectId("5e805fb6191ee0481ce2ac0c"), "numvuelo" : "FR2356", "origen" : "
MAD", "destino" : "VIA", "aerolinea" : "Ryanair", "estado" : "En hora", "__v" : 0 }
```

## Conclusions

Creiem que com a primera aproximació ens ha servit per a familiaritzar-nos amb l'ús de les APIs al món real, a la espera de poder connectar-la amb la base de dades i APP reals del grup B2.

# Definició de protocols

## Descripció general

//Descripció més detallada i concreta d'aquesta part del treball.

Tant el login, com les peticions a les bases de dades per a la ubicació dels clients i dels cotxes, es fan mitjançant protocols HTTPS al servidor, ja que es tracten de peticions com a POST, GET, PUT... De tot això s'encarregarà la API que ens connecta la web amb el que he dit abans, amb la base de dades, així com també ho farà amb l'aplicació de mòbil de cada client.

Ressaltar també que en la web de client, l'apartat de com arribar serà també una petició al servidor de Google, per tant tornarem a tenir el protocol HTTPS.

En resum, el protocol que s'usa sobretot és l'HTTPS, el qual usa un servei orientat a connexió (TCP).

Per detectar si un dels nostres usuaris es troba a l'Aeroport o no el que farem serà implementar un protocol UDP el qual vagi fent broadcast tota l'estona amb la nostre xarxa si aquest es detecta es trobarà dintre l'aeroport en cas negatiu , haurà marxat o perdut la connexió.

## Explicació/Justificació

//Explicació de la raó per la qual s'utilitzaran els protocols definits



Utilitzem TCP en HTTP degut a que es un protocol segur.

Utilitzem UDP per la detecció dels usuaris degut a que es més ràpid i la pèrdua d'alguns paquets es menyspreable.

## Conclusions

//\*\*Cal afegir un manual o be un link amb documentació dels protocols utilitzats.