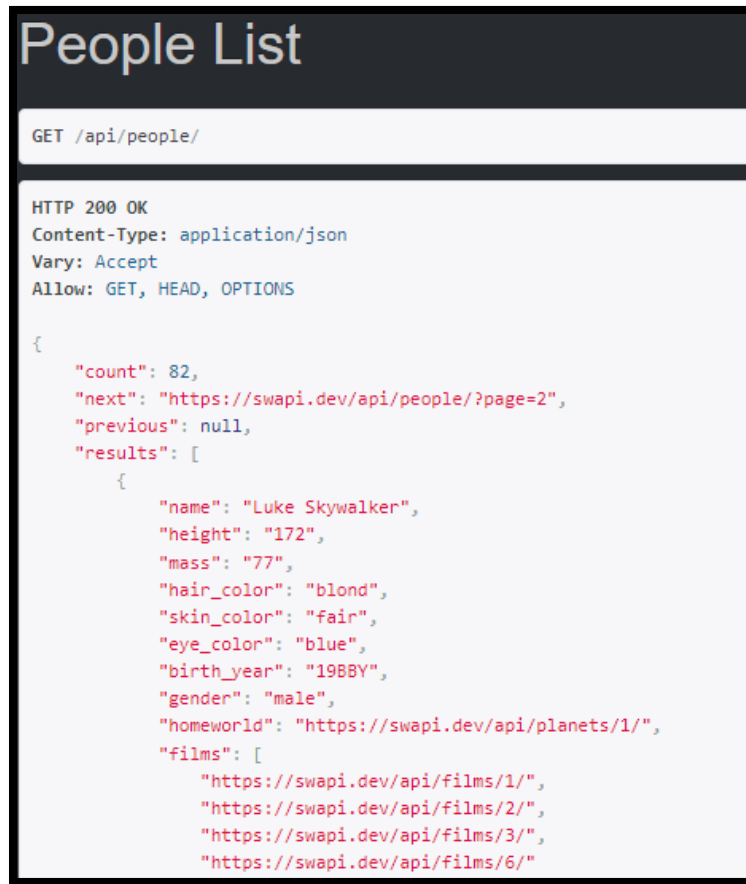


Jhonathan Grisales Giraldo

1. **Servicio javascript**, se creó un script para hacer el consumo de un api publica que se muestra a continuación:



```
GET /api/people/

HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS

{
  "count": 82,
  "next": "https://swapi.dev/api/people/?page=2",
  "previous": null,
  "results": [
    {
      "name": "Luke Skywalker",
      "height": "172",
      "mass": "77",
      "hair_color": "blond",
      "skin_color": "fair",
      "eye_color": "blue",
      "birth_year": "1988Y",
      "gender": "male",
      "homeworld": "https://swapi.dev/api/planets/1/",
      "films": [
        "https://swapi.dev/api/films/1/",
        "https://swapi.dev/api/films/2/",
        "https://swapi.dev/api/films/3/",
        "https://swapi.dev/api/films/6/"
      ]
    }
  ]
}
```

Con la ayuda de **Fetch** que nos brinda la capacidad de acceder y manipular partes del canal HTTP, se accedió a la url y se extrajeron los datos, posteriormente se hizo un filtro con el fin de determinar que personajes tenían un valor mayor a 100 en su atributo “height”, esta información se visualiza en el servidor de la siguiente manera.

API CONSUMPTION		
#	Name	height
1	Luke Skywalker	172
2	C-3PO	167
3	Darth Vader	202
4	Leia Organa	150
5	Owen Lars	178
6	Beru Whitesun lars	165
7	Biggs Darklighter	183
8	Obi-Wan Kenobi	182

Jhonathan Grisales Giraldo

Nota: Se hizo uso de Async/Await para la elaboración de dicho ejercicio con el fin de mejorar el performance de las consultas.

2. **Servicios Nodejs**, se creó un API que brinda la capacidad de hacer un CRUD gestionado en in array dentro del código, esta se elaboró con nodejs y express, para la solución de este punto se implementaron las rutas necesarias para el CRUD, las cuales se comunicaban con el controlador para hacer la gestión de estos procesos, para testear el API se utilizo la herramienta de POSTMAN el cual hace las peticiones directamente a las url's, como observamos en la siguiente imagen:

POST http://localhost:8000/users

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

x-www-form-urlencoded

KEY	VALUE
id	2
name	Michael
lastName	Giraldoi
userType	admin
Key	Value

Body Cookies Headers (7) Test Results 200 OK 19 ms 378 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "1",
3   "name": "Jhonathan ",
4   "lastName": "Grisales",
5   "userType": "admin"
6 },
7 {
8   "id": "2",
9   "name": "Michael",
10  "lastName": "Giraldoi",
11  "userType": "admin"
12 }
13 }
```

GET http://localhost:8000/users/1

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

x-www-form-urlencoded

KEY	VALUE
id	2
name	Michael
lastName	Giraldoi
userType	admin
Key	Value

Body Cookies Headers (7) Test Results 200 OK 10 ms 306 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "1",
3   "name": "Jhonathan ",
4   "lastName": "Grisales",
5   "userType": "admin"
6 }
```

PUT http://localhost:8000/users/2

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

x-www-form-urlencoded

KEY	VALUE
id	2
name	Esteban
lastName	Medina
userType	diseñador
Key	Value

Body Cookies Headers (7) Test Results 200 OK 10 ms 268 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "mensaje": "Usuario actualizado"
3 }
```

DELETE http://localhost:8000/users/2

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

x-www-form-urlencoded

KEY	VALUE
Key	Value

Body Cookies Headers (7) Test Results 200 OK 10 ms 375 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "1",
3   "name": "Esteban",
4   "lastName": "Medina",
5   "userType": "diseñador"
6 },
7 {
8   "id": "3",
9   "name": "Pedro",
10  "lastName": "Alvarez",
11  "userType": "admin"
12 }
13 }
14 }
```

Jhonathan Grisales Giraldo

En las anteriores imágenes se muestran las pruebas que se hicieron y los resultados que arrojan las peticiones, también creo una tabla en el Front para ir visualizando los cambios de esta misma, según las peticiones, esta tabla se observa en la siguiente imagen:

Name	Last Name	Type User
Esteban	Medina	diseñador
Pedro	Alvarez	admin

PLUS: Este mismo servicio se implemento con la base de datos Mongodb, la cual reemplaza el array del anterior ejercicio, en la siguiente imagen se observa la parte del front y la base de datos con el resultado final, después de hacer una serie de consultas.

Algunas de las librerías utilizadas fueron:

mongoose → Para la BD

bodyParser → Librería para acceder al body de las peticiones

path → Libreria para leer los archivos que existen en las carpetas

The screenshot displays a web application interface on the left and a MongoDB database view on the right. The web application shows a table with user data:

Name	Last Name	Type User
Jhonathan	Grisales Giraldo	admin
Michael	Giraldo	diseñador
Natalia	Angel	admin

The MongoDB database view on the right shows the 'api.users' collection with three documents:

```
{ "_id": "620674c42e445062674d50b6", "name": "Jhonathan", "lastName": "Grisales Giraldo", "userType": "admin", "__v": 0 }, { "_id": "620674d22e445062674d50b7", "name": "Michael", "lastName": "Giraldo", "userType": "diseñador", "__v": 0 }, { "_id": "6206e1f642c2ed0790b418d7", "name": "Natalia", "lastName": "Angel", "userType": "admin", "__v": 0 }
```

Jhonathan Grisales Giraldo

Middleware: Para la verificación del rol de los usuarios se realizó un Middleware que verifica si el usuario es admin y de esta manera nos indica si dicho usuario puede modificar y eliminar usuarios, en la siguiente imagen se muestra la prueba que se hizo con Postman para esta funcionalidad.

GET http://localhost:8000/users/620674d22e445b62674d5bbf

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

x-www-form-urlencoded

KEY	VALUE
id	3
name	Pedro
lastName	Alvarez
userType	admin

Body

200 OK 80 ms 340 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "620674d22e445b62674d5bbf",
3   "name": "Michael",
4   "lastName": "Giraldo",
5   "userType": "diseñador",
6   "__v": 0
7 }
```

```
[nodemon] starting `node ./index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node ./index.js`
El usuario no puede modificar ni agregar usuarios
```

El usuario Michael es de rol diseñador y no puede hacer estas gestiones, como lo indica en el mensaje de consola

GET http://localhost:8000/users/6206a1f642c2edb79bb418d7

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

x-www-form-urlencoded

KEY	VALUE
id	3
name	Pedro
lastName	Alvarez
userType	admin

Body

200 OK 15 ms 332 B Save Response

Pretty Raw Preview Visualize JSON

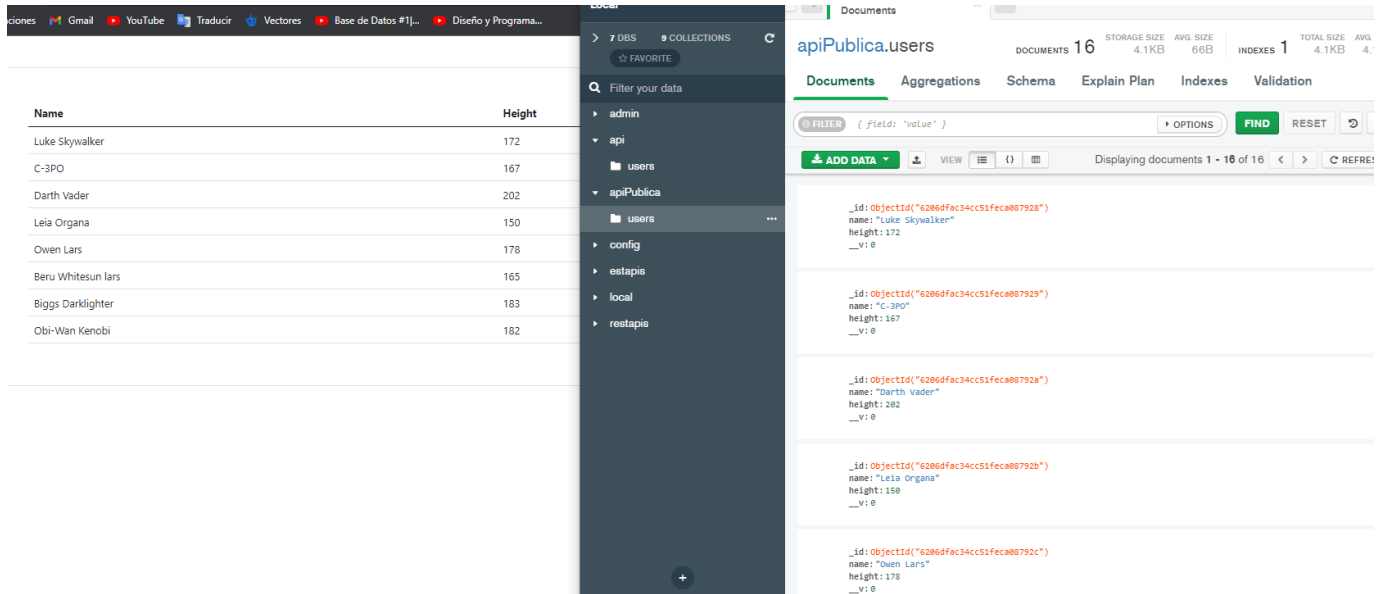
```
1 {
2   "_id": "6206a1f642c2edb79bb418d7",
3   "name": "Natalia",
4   "lastName": "Angel",
5   "userType": "admin",
6   "__v": 0
7 }
```

```
[nodemon] starting `node ./index.js`
El usuario no puede modificar ni agregar usuarios
[nodemon] starting `node ./index.js`
El usuario no puede modificar ni agregar usuarios
El usuario puede modificar y agregar usuarios
```

El usuario Natalia es de rol admin y si puede hacer estas gestiones, como lo indica el segundo mensaje de consola.

Jhonathan Grisales Giraldo

PLUS: Se implemento el ejercicio elaborado en Javascript (consumo de api publica), en Nodejs con ayuda de express y mongoDB, Con la librería **Node-Fetch** que nos brinda la capacidad de acceder y manipular partes del canal HTTP, se accedió a la url y se extrajeron los datos, posteriormente se hizo un filtro con el fin de determinar que personajes tenían un valor mayor a 100 en su atributo “height”, esta información se visualiza en el servidor y la base de datos de la siguiente manera.



The image shows a web application interface on the left and a MongoDB Compass interface on the right. The web application displays a table of Star Wars characters with their names and heights. The MongoDB interface shows the same data stored in a collection named 'users' in a database named 'apiPublica'.

Name	Height
Luke Skywalker	172
C-3PO	167
Darth Vader	202
Leia Organa	150
Owen Lars	178
Beru Whitesun lars	165
Biggs Darklighter	183
Obi-Wan Kenobi	182

The MongoDB interface shows the following documents in the 'apiPublica.users' collection:

```
{ "_id": ObjectId("6206dfac34cc51feca087928"), "name": "Luke Skywalker", "height": 172, "__v": 0 }
{ "_id": ObjectId("6206dfac34cc51feca087929"), "name": "C-3PO", "height": 167, "__v": 0 }
{ "_id": ObjectId("6206dfac34cc51feca08792a"), "name": "Darth Vader", "height": 202, "__v": 0 }
{ "_id": ObjectId("6206dfac34cc51feca08792b"), "name": "Leia Organa", "height": 150, "__v": 0 }
{ "_id": ObjectId("6206dfac34cc51feca08792c"), "name": "Owen Lars", "height": 178, "__v": 0 }
```

Nota: Se hizo uso de Async/Await para la elaboración de dicho ejercicio con el fin de mejorar el performance de las consultas.