



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO



FACULTAD DE ESTUDIOS SUPERIORES
ARAGÓN

Ingeniería en Computación

ORGANIZAC.Y ADMON.CENTROS COMPUTO

Profesor: AARON VELASCO AGUSTIN

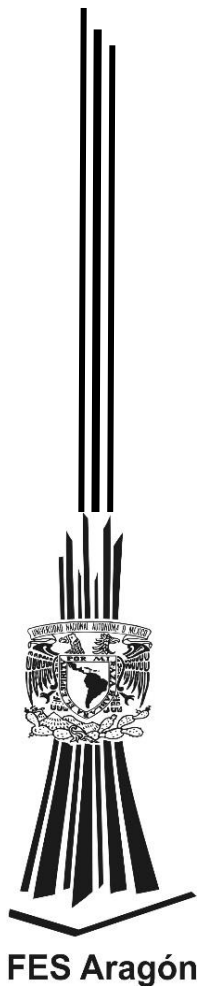
Proyecto Final

Alumnos:

Hernandez Mireles Daniel Jerico

Martínez Hernandez Fernando

Jiménez Prado Hassel



23 de mayo 2021

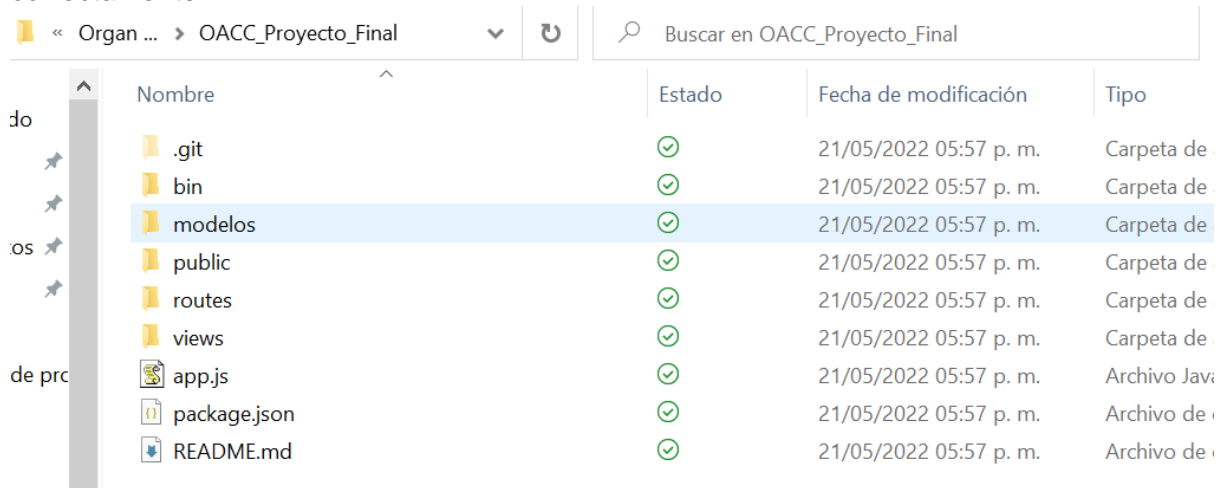
Para la creación de nuestro proyecto, fue necesario implementar 3 tipos de tecnologías, entre las cuales están Express, Node js y MongoDB.

Decidimos emplear la herramienta de Express ya que nos permite crear un entorno de desarrollo más rápido, implementando carpetas programadas para iniciar la programación de un crud.

Como primera parte nos dimos a la tarea de crear el entorno de desarrollo por medio del comando:

express

Por medio de este se nos generan todas las carpetas necesarias para que este funcione correctamente.



« Organ ... » OACC_Proyecto_Final		Buscar en OACC_Proyecto_Final		
	Nombre	Estado	Fecha de modificación	Tipo
	.git	✓	21/05/2022 05:57 p. m.	Carpeta de
	bin	✓	21/05/2022 05:57 p. m.	Carpeta de
	modelos	✓	21/05/2022 05:57 p. m.	Carpeta de
	public	✓	21/05/2022 05:57 p. m.	Carpeta de
	routes	✓	21/05/2022 05:57 p. m.	Carpeta de
	views	✓	21/05/2022 05:57 p. m.	Carpeta de
	app.js	✓	21/05/2022 05:57 p. m.	Archivo Javi
	package.json	✓	21/05/2022 05:57 p. m.	Archivo de
	README.md	✓	21/05/2022 05:57 p. m.	Archivo de

Después corremos el comando:

npm install

El cuál nos descargará todas las dependencias que necesitamos para trabajar.

Después de este paso procedemos a instalar nuestra herramienta necesaria para ocupar mongoDB, en este caso usaremos el comando:

npm install --save mongoose

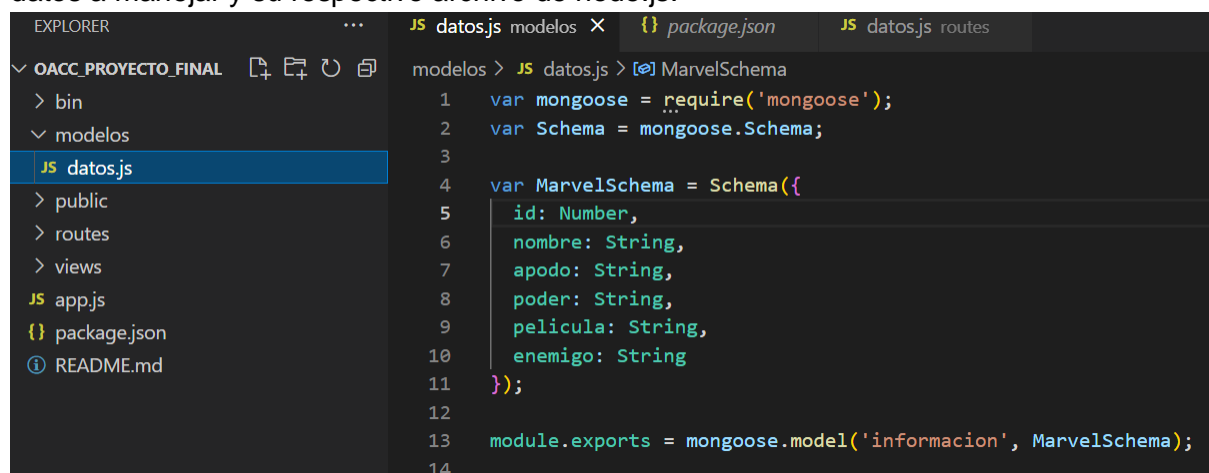
Este comando nos sirve para instalar las dependencias de mongo dentro de nuestro archivo package.json creado, para este caso nos descarga la versión 6.3.3.

```

1  {
2    "name": "proyecto-aaron",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./bin/www"
7    },
8    "dependencies": {
9      "cookie-parser": "~1.4.4",
10     "debug": "~2.6.9",
11     "express": "~4.16.1",
12     "http-errors": "~1.6.3",
13     "jade": "~1.11.0",
14     "mongoose": "^6.3.3",
15     "morgan": "~1.9.1"
16   }
17 }
18

```

Después de este paso, ahora sí procedemos a crear una carpeta que contendrá nuestros datos a manejar y su respectivo archivo de node.js.



```

1  var mongoose = require('mongoose');
2  var Schema = mongoose.Schema;
3
4  var MarvelSchema = Schema({
5    id: Number,
6    nombre: String,
7    apodo: String,
8    poder: String,
9    pelicula: String,
10   enemigo: String
11 });
12
13 module.exports = mongoose.model('informacion', MarvelSchema);
14

```

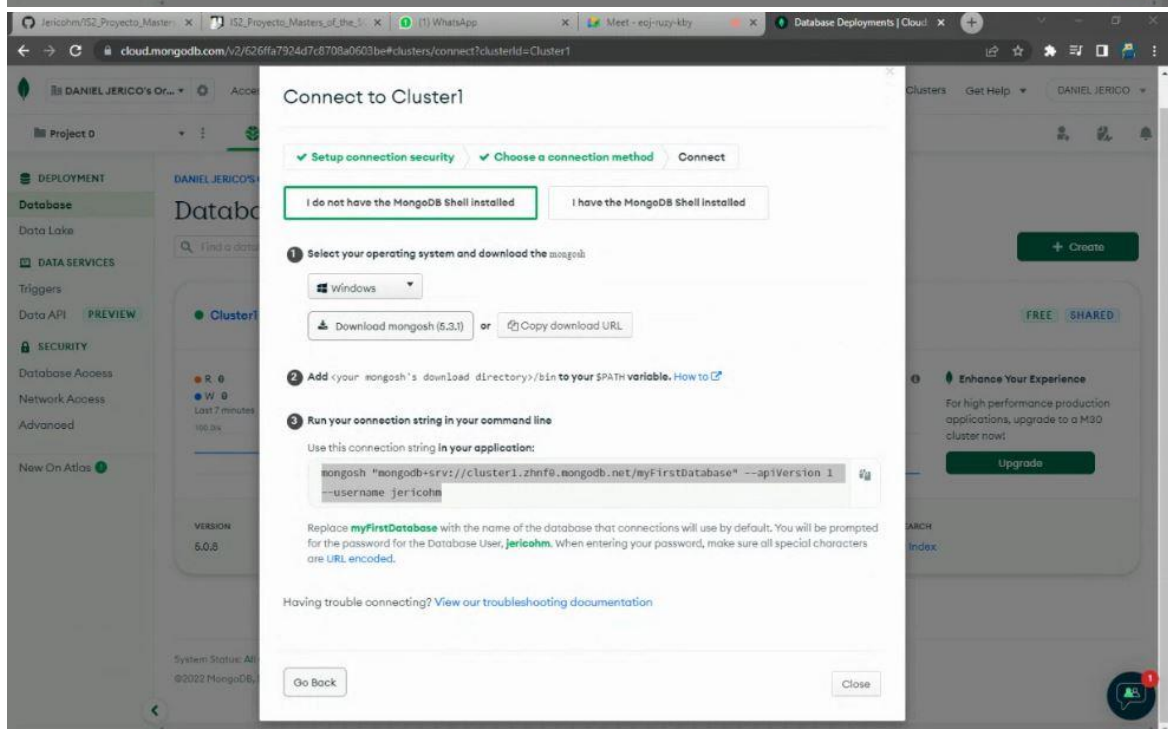
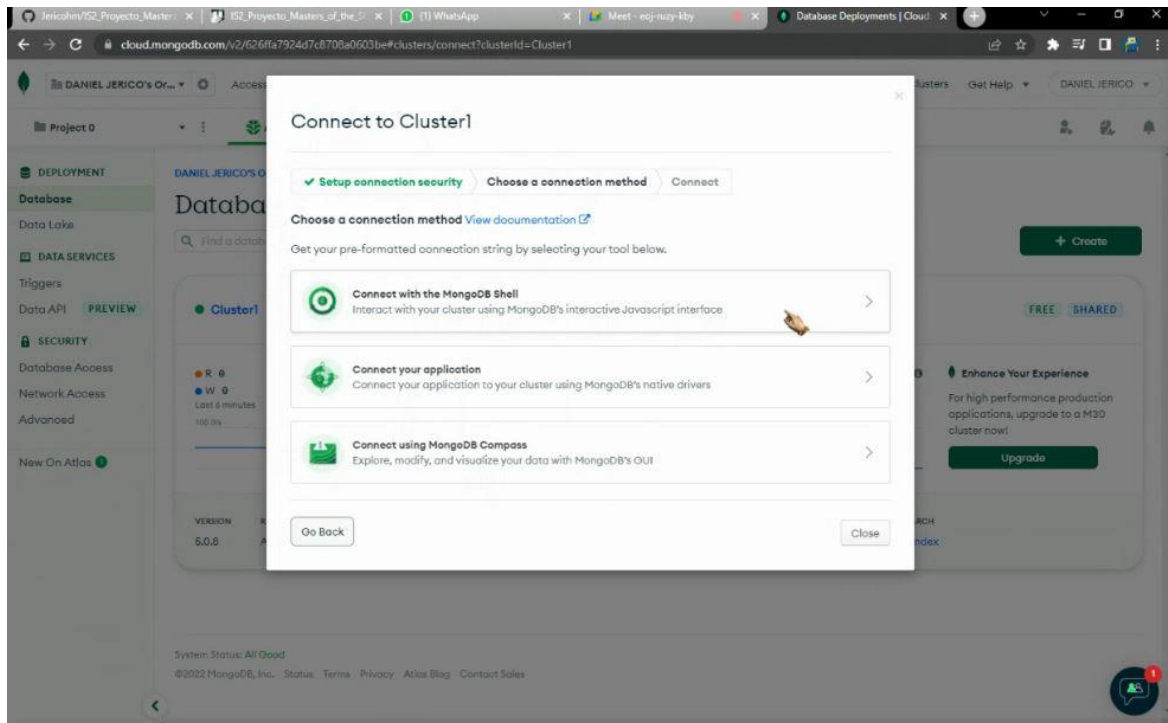
Dentro de este archivo creamos una variable que maneja mongoose y un objeto de tipo Schema proveniente de mongoose que nos permite crear el constructor de nuestros datos. La última parte del código es la exportación de un modelo de mongoose que llamamos información y el constructor que nombramos como MarvelSchema. Cabe aclarar que ocupamos una estructura para el constructor.

Ahora iniciamos con el proceso de creación de la base de datos ocupando la herramienta de MongoDB que nos permitirá almacenar nuestros datos del constructor.



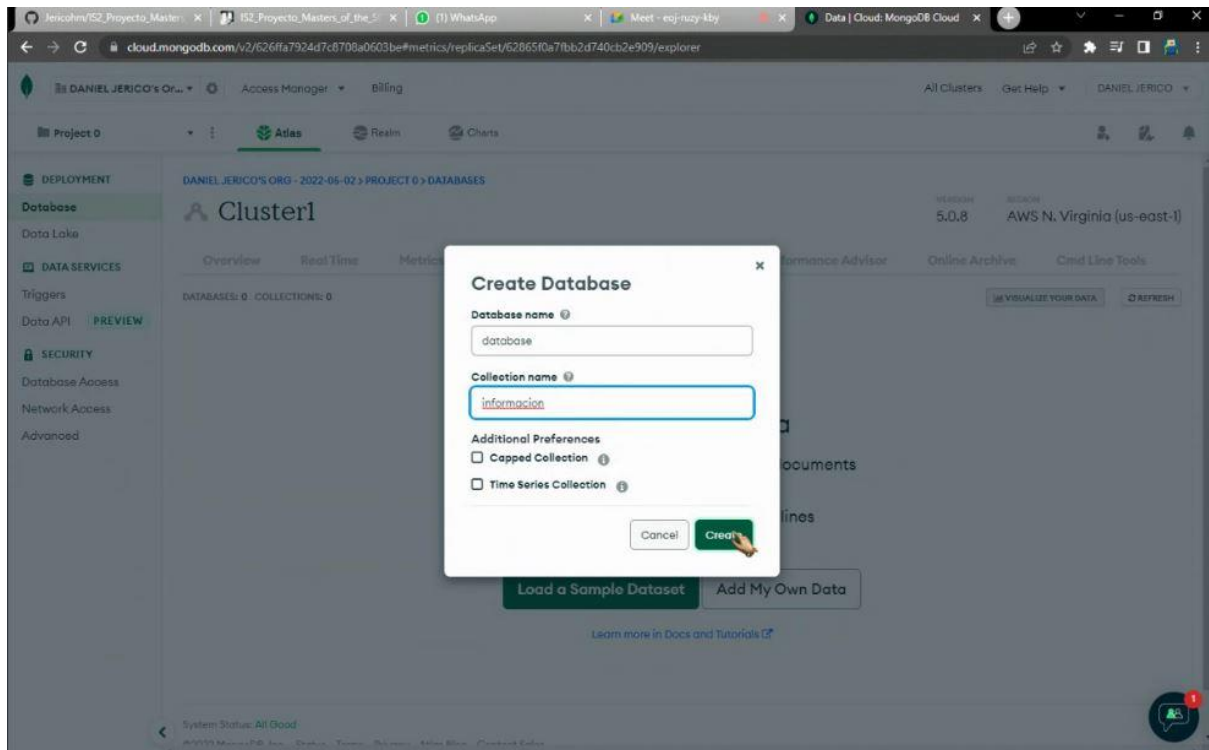
[Log in to your account](#)

Creamos un nuevo usuario dentro de MongoDB y continuamos con la creación de nuestra base de datos.

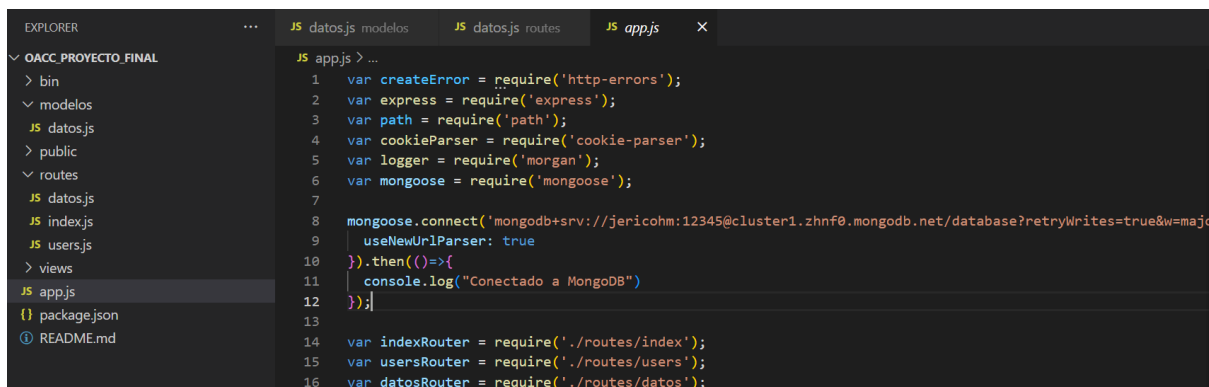


Conectamos con nuestra aplicación de Mongo mediante NodeJS y creamos una conexión a la base de datos la cuál nos genera una dirección.

```
mongoose.connect('mongodb+srv://jericohm:12345@cluster1.zhnf0.mongodb.net/database?retryWrites=true&w=majority',{
  useNewUrlParser: true
}).then(()=>{
  console.log("Conectado a MongoDB")
});
```



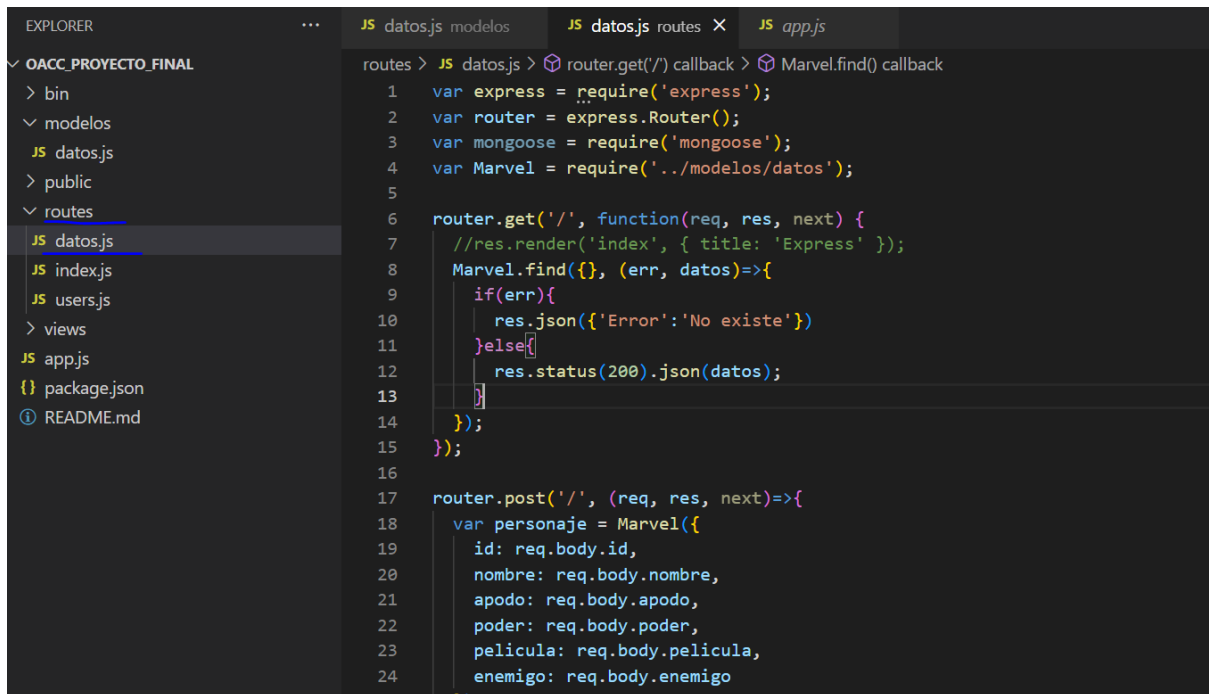
Creamos nuestra base de datos que colocamos en el modelo exportado al archivo de datos en modelos para que encuentre el espacio de almacenamiento.
información



Esta conexión la agregamos en nuestro archivo de app.js para redireccionar la ruta de subida de datos junto con el apoyo de la variable mongoose. Agregamos nuestra ruta de datos para que se pueda encontrar la información desde el navegador ocupando la siguiente url:

<http://127.0.0.1:3000/datos>

Ahora procedemos a crear la parte de datos en rutas y a colocar todos los métodos de manipulación de datos.



```
EXPLORER
OACC_PROYECTO_FINAL
  > bin
  > modelos
  JS datos.js
  > public
  > routes
  JS datos.js
  JS index.js
  JS users.js
  > views
  JS app.js
  {} package.json
  ① README.md

routes > JS datos.js > router.get('/') callback > Marvel.find() callback
1  var express = require('express');
2  var router = express.Router();
3  var mongoose = require('mongoose');
4  var Marvel = require('../modelos/datos');
5
6  router.get('/', function(req, res, next) {
7    //res.render('index', { title: 'Express' });
8    Marvel.find({}, (err, datos)=>{
9      if(err){
10        res.json({'Error': 'No existe'})
11      }else{
12        res.status(200).json(datos);
13      }
14    });
15  });
16
17  router.post('/', (req, res, next)=>{
18    var personaje = Marvel({
19      id: req.body.id,
20      nombre: req.body.nombre,
21      apodo: req.body.apodo,
22      poder: req.body.poder,
23      pelicula: req.body.pelicula,
24      enemigo: req.body.enemigo
25    });
26  });
```

En este archivo ocupamos la estructura de users.js para facilitarnos el código y únicamente implementar una variable de mongoose, una variable la cuál será Marvel, la que hará la solicitud de datos con el direccionamiento que pusimos en require.

Ahora creamos el método get para obtener todos los datos contenidos en la base de datos y todo lo que demos de alta en la página.

```
router.get('/', function(req, res, next) {
  //res.render('index', { title: 'Express' });
  Marvel.find({}, (err, datos)=>{
    if(err){
      res.json({'Error': 'No existe'})
    }else{
      res.status(200).json(datos);
    }
  });
});
```

Creamos el método post para dar de alta los datos y almacenarlos en el espacio de Mongo.

```
router.post('/', (req, res, next)=>{
  var personaje = Marvel({
    id: req.body.id,
    nombre: req.body.nombre,
    apodo: req.body.apodo,
    poder: req.body.poder,
    pelicula: req.body.pelicula,
    enemigo: req.body.enemigo
  });
  personaje.save((err, data)=>{
    if(err){
```

```
    res.json({'error':"Error al insertar"});  
  }else{  
    res.status(200).json(data);  
  }  
});  
});
```

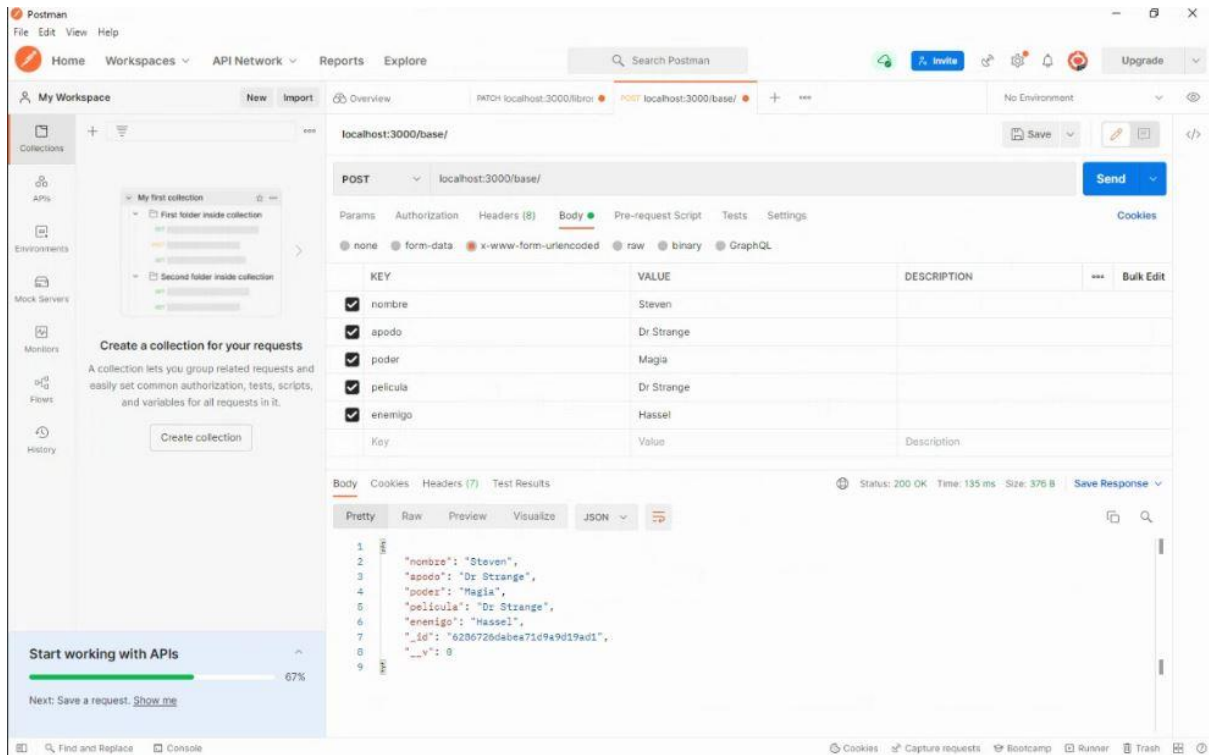
Incluyendo los datos del constructor que creamos.

Después continuamos con la creación del último método, el cuál nos permitirá eliminar datos que no sean necesarios con el comando delete.

```
router.delete('/:idMar', function(req, res, next) {  
  Marvel.deleteOne({'id':req.params.idMar}, (err)=>{  
    if(err){  
      res.json({'Error':'No existe'})  
    }else{  
      res.json({'Estatus':'Borrado'})  
    }  
  });  
});
```

Cada uno de los métodos tiene una respuesta según sea el caso, en los que saldrá un mensaje de error si la solicitud se hizo incorrectamente o si no existe el dato y otra respuesta indicándonos el estado de la solicitud como correcta.

Por último realizamos la prueba de las solicitudes con el software de Postman, en el que podemos generar fácilmente la creación de datos y subirlos al espacio o en su caso hacer consultas con el formato de http.



Realizamos las diferentes solicitudes creadas con los comandos:

POST e incluimos el cuerpo del constructor dándole los datos y el valor de cada uno.
GET para mandar a desplegar la información existente.
DELETE para borrar un dato dándole el id.

Empleamos la ruta:

localhost:3000/datos/

Y damos en el botón de send para mandar la solicitud.

OACC_Proyecto_Final	23/05/2022 04:44 p. m.	Carpeta de archivos	
Firma.txt	23/05/2022 04:50 p. m.	Documento de te...	1 KB
OACC_Proyecto_Final.rar	23/05/2022 04:45 p. m.	Archivo WinRAR	4,129 KB

```

MINGW64/c:/Users/volve/Desktop/FES ARAGON/OCTAVO SEMESTRE/Organ...
volve@LAPTOP-1U2LAKSF MINGW64 ~/Desktop/FES ARAGON/OCTAVO SEMESTRE/Organización
y Administración de Centros de Computo/Proyecto_Version_Hassel
$ mkdir firma.txt

volve@LAPTOP-1U2LAKSF MINGW64 ~/Desktop/FES ARAGON/OCTAVO SEMESTRE/Organización
y Administración de Centros de Computo/Proyecto_Version_Hassel
$ ls
Firma.txt  OACC_Proyecto_Final/  OACC_Proyecto_Final.rar

volve@LAPTOP-1U2LAKSF MINGW64 ~/Desktop/FES ARAGON/OCTAVO SEMESTRE/Organización
y Administración de Centros de Computo/Proyecto_Version_Hassel
$ nano Firma.txt

volve@LAPTOP-1U2LAKSF MINGW64 ~/Desktop/FES ARAGON/OCTAVO SEMESTRE/Organización
y Administración de Centros de Computo/Proyecto_Version_Hassel
$ cat Firma.txt
OACC_Proyecto_Final.rar 1
be91d25e31f15c2c8c6574348928dec08ab37430

volve@LAPTOP-1U2LAKSF MINGW64 ~/Desktop/FES ARAGON/OCTAVO SEMESTRE/Organización
y Administración de Centros de Computo/Proyecto_Version_Hassel
$

```


Esta última parte extra de nuestro proyecto fue la creación de una firma de autenticidad que nos genera una clave, cuando se hagan modificaciones al proyecto esta clave se verá modificada luego de correr el comando:

shasum -a 1 OACC_Proyecto_Final.rar

Guardamos la clave en un documento de texto y podemos confirmar que nuestra clave ya existe.