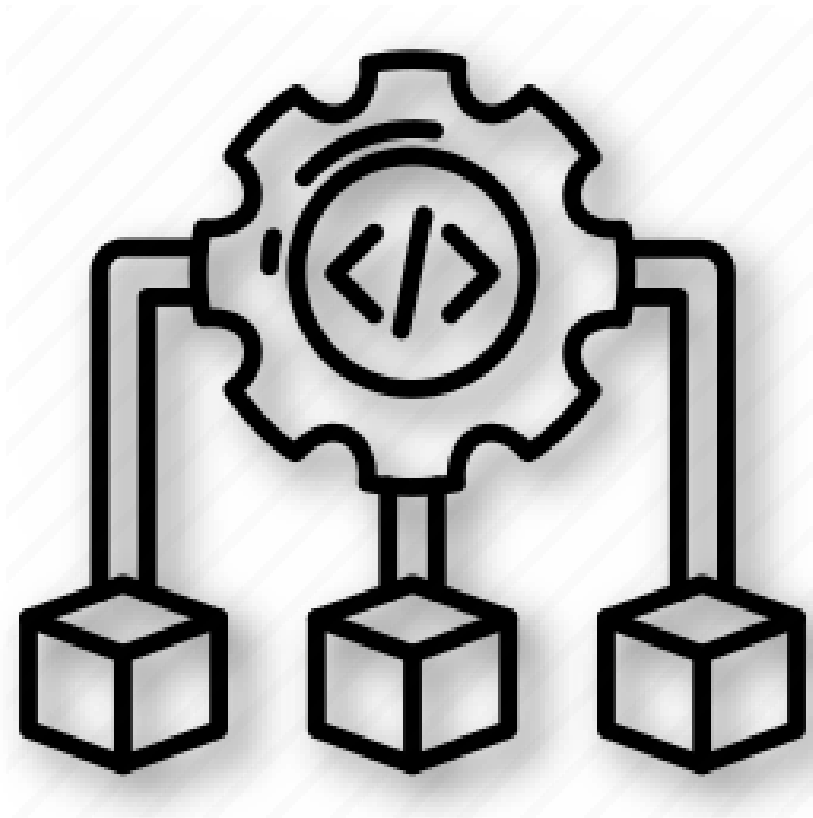




**Pontificia Universidad
Católica del Ecuador**
Seréis mis testigos

ESMERALDAS



API REST

Primeros pasos

Farit Alexander Reasco Torres.
Programación I



Creación general de la API Rest CRUD.

Para empezar, debemos tener instalado Node.JS, MongoDB y Postman, si así es, entonces proceda a crear un espacio de trabajo e inicializar la aplicación Node.js con un archivo *package.json*.

```
C:\Users\areas\Desktop>mkdir API-REST-HM
```

```
C:\Users\areas\Desktop>cd API-REST-HM
```

```
$ mkdir nombre_del_directorio  
$ cd nombre_del_directorio  
$ npm init
```

```
C:\Users\areas\Desktop\API-REST-HM>npm init  
This utility will walk you through creating a package.json file.  
It only covers the most common items, and tries to guess sensible defaults.  
  
See 'npm help init' for definitive documentation on these fields  
and exactly what they do.  
  
Use 'npm install <pkg>' afterwards to install a package and  
save it as a dependency in the package.json file.  
  
Press ^C at any time to quit.  
package name: (api-rest-hm)  
version: (1.0.0)  
description: API de servicios universitarios - test tarea  
entry point: (index.js) server.js  
test command:  
git repository:  
keywords: api rest, servicios, bases de datos, crud, mongodb, express  
author: Farit Reasco  
license: (ISC)  
About to write to C:\Users\areas\Desktop\API-REST-HM\package.json:  
{  
  "name": "api-rest-hm",  
  "version": "1.0.0",  
  "description": "API de servicios universitarios - test tarea",  
  "main": "server.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "keywords": [  
    "api",  
    "rest",  
    "servicios",  
    "bases",  
    "de",  
    "datos",  
    "crud",  
    "mongodb",  
    "express"  
  ],  
  "author": "Farit Reasco",  
  "license": "ISC"  
}
```

npm init:

1. Colocar un nombre al paquete, asignar el valor por defecto a la versión, darle una descripción, un punto de entrada (archivo base), comando de prueba (por defecto), repositorio de Git (en caso de tenerlo), palabras claves, autor, licencia (por defecto).
2. Revisar el resumen los datos ingresados y continuar.

Así debería verse el archivo *package.json* desde VS code:

Una vez aplicada la configuración inicial, podemos ejecutar el comando **dir** para verificar si las carpetas de la aplicación se crearon correctamente, si ese es el caso, ejecute **code** . para abrir vscode y empezar al desarrollar la API. Es necesario contar con una base de datos MongoDB y conexión con Postman.

```
$ dir  
$ code .
```

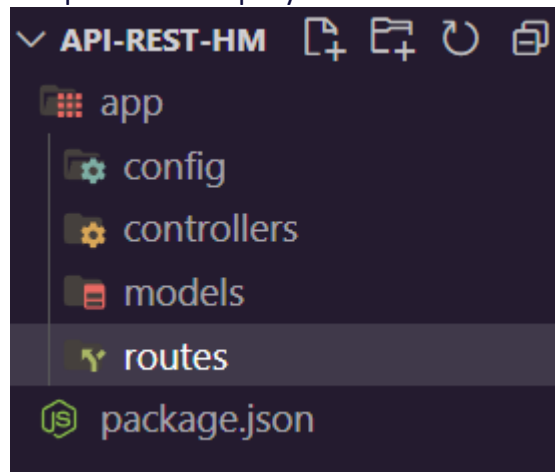
Se sugiere instalar Nodemon (1) y añadirlo en el package.json en "scripts" -> "test" colocando (2):

1. \$ npm i nodemon -g
2. "start": "nodemon server.js"

```
EXPLORER  
API-REST-HM  
package.json  
package.json X  
package.json > {} scripts > start  
1 {  
2   "name": "api-rest-hm",  
3   "version": "1.0.0",  
4   "description": "API de servicios universitarios - test tarea",  
5   "main": "server.js",  
6   "scripts": {  
7     "test": "echo \\\"Error: no test specified\\\" && exit 1",  
8     "start": "nodemon server.js"  
9   },  
10  "keywords": [  
11    "api",  
12    "rest",  
13    "servicios",  
14    "bases",  
15    "de",  
16    "datos",  
17    "crud",  
18    "mongodb",  
19    "express"  
20  ],  
21  "author": "Farit Reasco",  
22  "license": "ISC"  
23 }  
24
```

Preparación del entorno y primeros pasos:

Crearemos la siguiente estructura para nuestro proyecto:



También, necesitamos instalar los módulos necesarios: **express, mongoose y cors**.

```
npm install express mongoose cors --save
```

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
PS C:\Users\areas\Desktop\API-REST-HM> npm install express mongoose cors --save
added 86 packages, and audited 87 packages in 9s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\areas\Desktop\API-REST-HM>

```

Luego, crearemos el archivo **server.js** (archivo punto de entrada) en la carpeta raíz del proyecto y dentro se debe configurar el servidor web Express, usando las siguientes instrucciones:

```

package.json  server.js  X
server.js > app.get("/") callback
1  const express = require("express");
2  const cors = require("cors");
3
4  const app = express();
5
6  var corsOptions = {
7    origin: "http://localhost:8081"
8  };
9
10 app.use(cors(corsOptions));
11
12 // parse requests of content-type - application/json
13 app.use(express.json());
14
15 // parse requests of content-type - application/x-www-form-urlencoded
16 app.use(express.urlencoded({ extended: true }));
17
18 // simple route
19 app.get("/", (req, res) => {
20   res.json({ message: "Bienvenido a la API REST de Pucese_data" });
21 });
22
23 // set port, listen for requests
24 const PORT = process.env.PORT || 8080;
25 app.listen(PORT, () => {
26   console.log(`Server is running on port ${PORT}.`);
27 });

```

Para comprobar si el servidor está corriendo, debemos ejecutar:



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

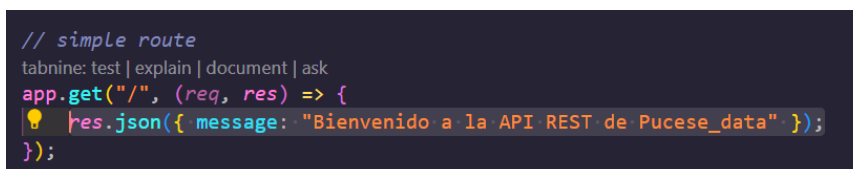
found 0 vulnerabilities
PS C:\Users\areas\Desktop\API-REST-HM> npm start

> api-rest-hm@1.0.0 start
> nodemon server.js

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server is running on port 8080.

```

Y con ayuda de POSTMAN, verificaremos si esto es correcto, antes compruebe si está usando el método GET, ya que este es el que usaremos en POSTMAN; además, configure un mensaje de prueba para la API.

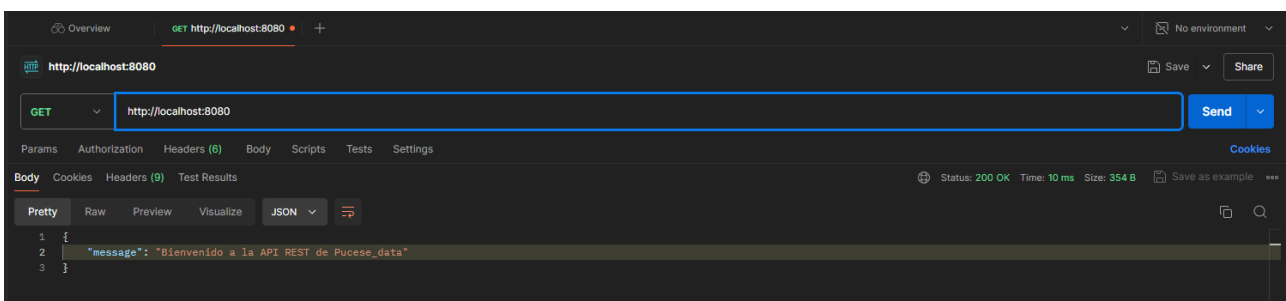


```

// simple route
tabnine: test | explain | document | ask
app.get("/", (req, res) => {
  res.json({ message: "Bienvenido a la API REST de Pucese_data" });
});

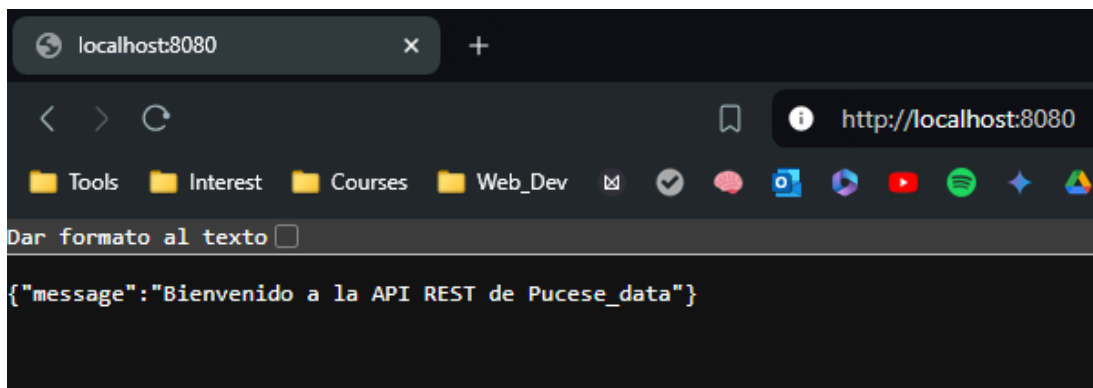
```

Solicitar una petición aplicando el método GET y colocando la URL correspondiente al servidor y el puerto que, en este caso es <http://localhost:8080>, luego darle en **enviar** y verificar los cambios en consola.



The screenshot shows the Postman interface with a GET request to `http://localhost:8080` sent. The response status is 200 OK, and the body is a JSON object: `{ "message": "Bienvenido a la API REST de Pucese_data" }`.

Así es la vista desde el navegador:



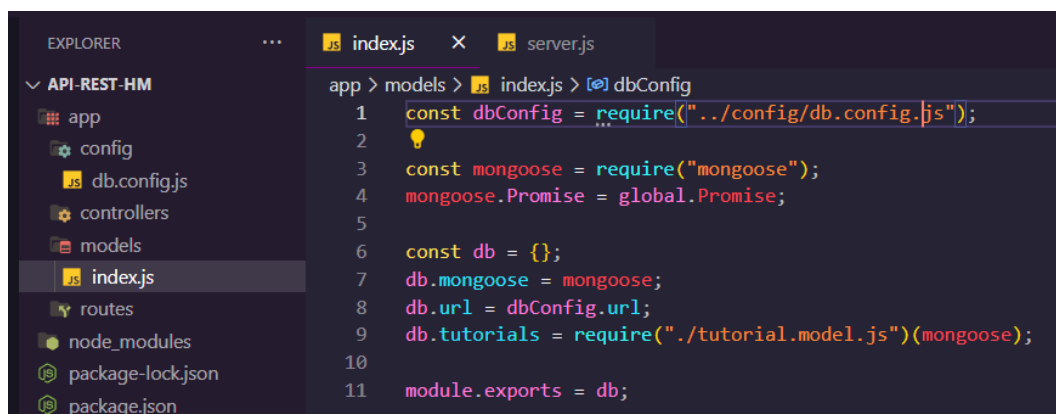
Ahora, vamos a configurar la base de datos MongoDB.

- En la carpeta **config** creamos el archivo **db.config.js**.
- Modifique el código colocando la URL de su base de datos.



Definir Mongoose

En el siguiente paso, también definiremos Mongoose (tutorial.model.js) en la carpeta app/models. Ahora crea app/models/index.js con el siguiente código:



En el archivo server.js se debe llamar al método connect() bajo el siguiente código:

```

server.js > ...
10 app.use(cors({credentials: true}));
11
12 // parse requests of content-type - application/json
13 app.use(express.json());
14
15 // parse requests of content-type - application/x-www-form-urlencoded
16 app.use(express.urlencoded({ extended: true }));
17
18 const db = require("../app/models");
19 db.mongoose
20   .connect(db.url, {
21     useNewUrlParser: true,
22     useUnifiedTopology: true
23   })
24   .then(() => {
25     console.log("Connected to the database!");
26   })
27   .catch(err => {
28     console.log("Cannot connect to the database!", err);
29     process.exit();
30   });
31
32 // simple route
33 app.get("/", (req, res) => {
34   res.json({ message: "Bienvenido a la API REST de Pucese_data" });
35 });
36

```

Definir el modelo Mongoose

En la carpeta de **model**, crea el archivo **collectionName.model.js** de esta manera:

```

EXPLORER
API-REST-HM
├── app
│   ├── config
│   │   └── db.config.js
│   ├── controllers
│   ├── models
│   │   ├── index.js
│   │   └── info_estudiantes.model.js
│   ├── routes
│   └── node_modules
├── package-lock.json
├── package.json
└── server.js

app > models > info_estudiantes.model.js > <unknown> > exports
1 module.exports = mongoose => {
2   const Estudiante = mongoose.model(
3     "info_estudiantes", //Referencias a la colección real
4     mongoose.Schema(
5       {
6         cedula: String,
7         apellido: String,
8         nombre: String,
9         edad: Number,
10        en_pareja: Boolean
11      },
12      { timestamps: true }
13    ),
14    "info_estudiantes" //Referencias a la colección real
15  );
16
17  return Estudiante;
18 };

```

Ahora, realizaremos algunos cambios en el archivo **index.js**, donde cambiaremos el nombre de `db.nombreVariable = require("../collectionName.model.js")(mongoose)`.

```

1  const dbConfig = require("../config/db.config.js");
2
3  const mongoose = require("mongoose");
4  mongoose.Promise = global.Promise;
5
6  const db = {};
7  db.mongoose = mongoose;
8  db.url = dbConfig.url;
9  db.estudiantes = require("../info_estudiantes.model.js")(mongoose);
10
11 module.exports = db;

```

(node:28376) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology will be removed in the next major version
 Server is running on port 8080.
 Connected to the database!

Crear el controlador

Dentro de la carpeta app/controllers, creamos tutorial.controller.js con estas funciones CRUD

```

1  const db = require("../models");
2  const Estudiantes = db.estudiantes;
3
4  // Retrieve all Students from the database.
5  tabnine: test | explain | document | ask
6  exports.findAll = (req, res) => {
7
8

```

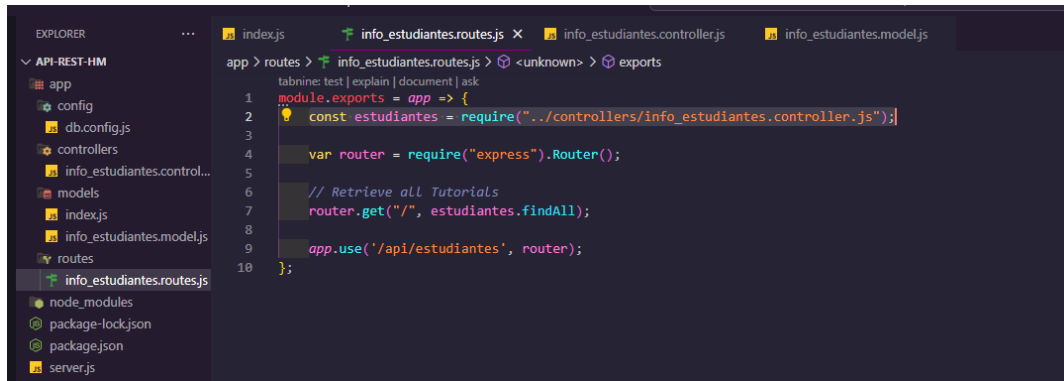
Ahora, vamos a recuperar información de la colección.

```

1  const db = require("../models");
2  const Estudiantes = db.estudiantes;
3
4  // Retrieve all Students from the database.
5  tabnine: test | explain | document | ask
6  exports.findAll = (req, res) => {
7
8      const apellido = req.query.apellido;
9      var condition = apellido ? { apellido: { $regex: new RegExp(apellido), $options: "i" } } : {};
10
11      Estudiantes.find(condition)
12        .then(data => {
13            res.send(data);
14        })
15        .catch(err => {
16            res.status(500).send({
17                message:
18                    err.message || "Hubo un error al obtener los datos de Estudiantes."
19            });
20        });
21    };

```

Ahora debemos referenciar dicho método definiendo las rutas, para ello, creamos una carpeta llamada `collectionName.routes.js` y agregamos lo siguiente:



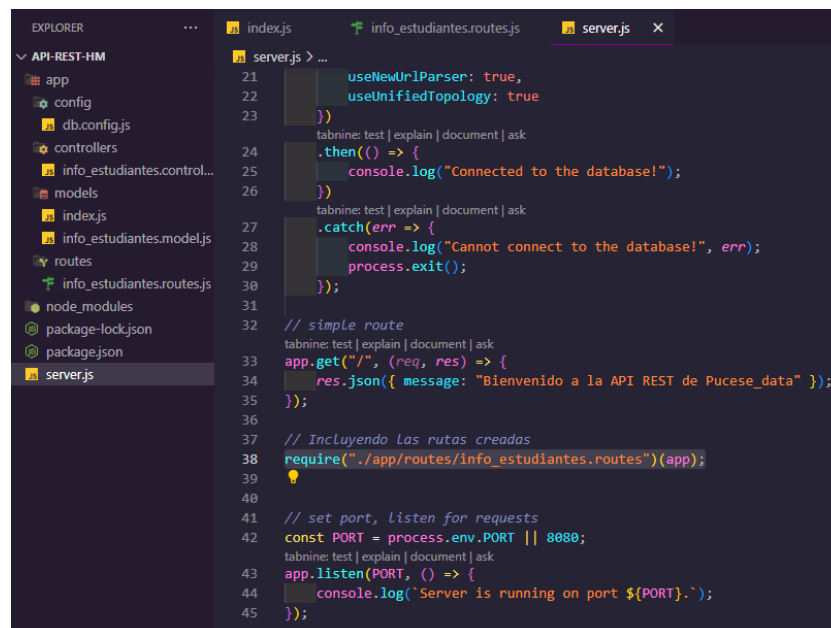
```

1  module.exports = app => {
2    const estudiantes = require("../controllers/info_estudiantes.controller.js");
3
4    var router = require("express").Router();
5
6    // Retrieve all Tutorials
7    router.get("/", estudiantes.findAll);
8
9    app.use('/api/estudiantes', router);
10 };

```

(Debe cambiar el nombre de la constante dependiendo de la información a tratar, referenciar el llamado del método `findAll` y definir la ruta).

Después, referenciamos la ruta en el archivo `server.js`.

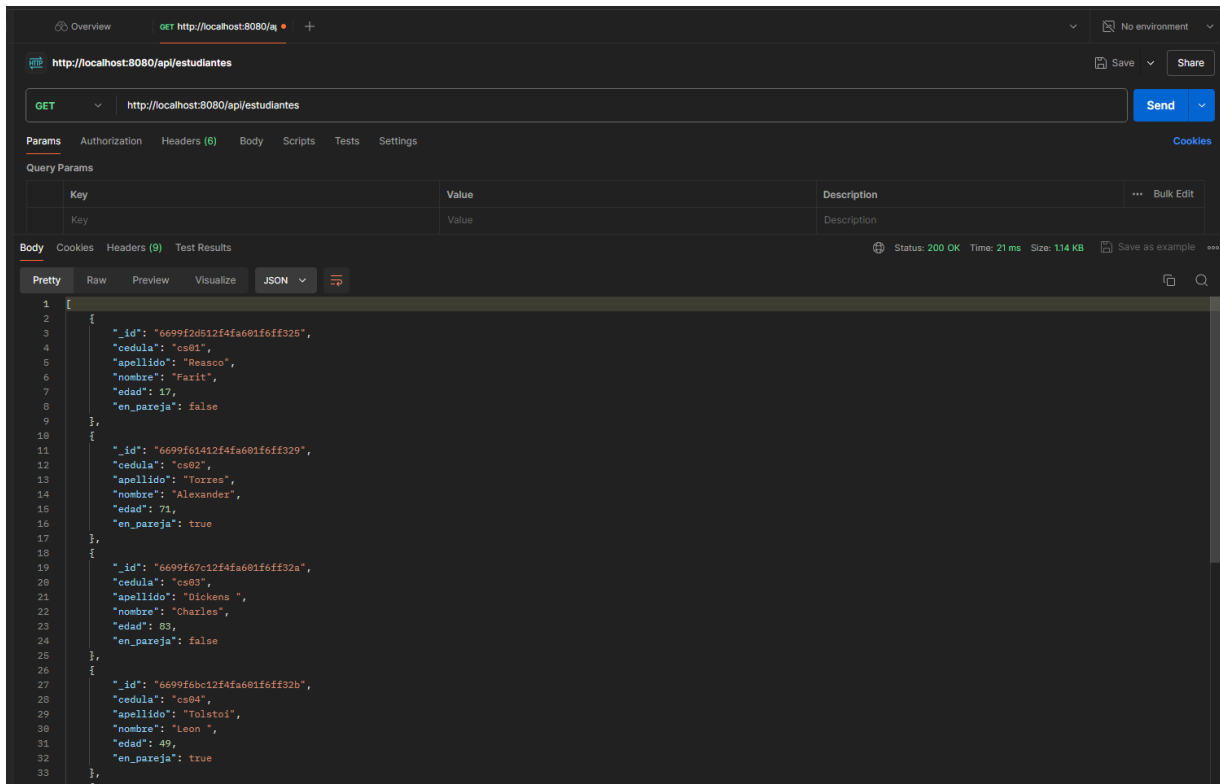


```

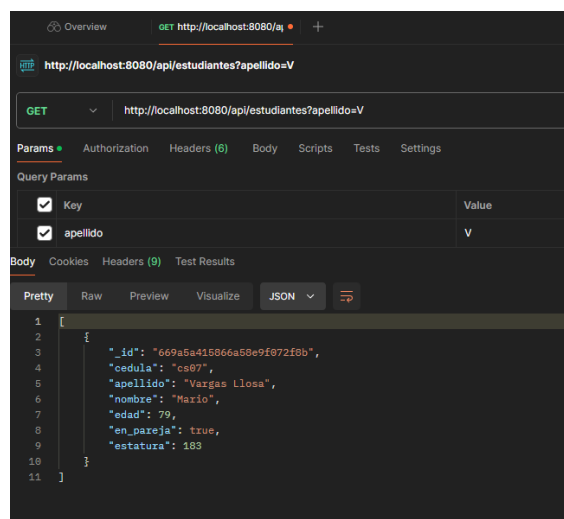
21 useNewUrlParser: true,
22 useUnifiedTopology: true
23 });
24
25 tabnine: test | explain | document | ask
26 .then(() => {
27   console.log("Connected to the database!");
28 })
29
30 tabnine: test | explain | document | ask
31 .catch(err => {
32   console.log("Cannot connect to the database!", err);
33   process.exit();
34 });
35
36 // simple route
37 tabnine: test | explain | document | ask
38 app.get("/", (req, res) => {
39   res.json({ message: "Bienvenido a la API REST de Pucese_data" });
40 });
41
42 // Incluyendo las rutas creadas
43 require("../app/routes/info_estudiantes.routes")(app);
44
45 // set port, listen for requests
46 const PORT = process.env.PORT || 8080;
47 tabnine: test | explain | document | ask
48 app.listen(PORT, () => {
49   console.log(`Server is running on port ${PORT}.`);
50 });

```

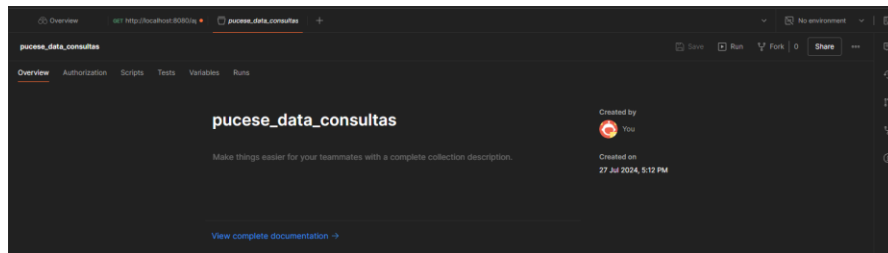
En la ruta debemos colocar el nombre del archivo: `collectionName.routes`. Por consiguiente, verifiquemos los datos en **Postman** modificando la ruta (**`http://localhost:8080/api/estudiantes`**) y dando clic al botón **Enviar**.



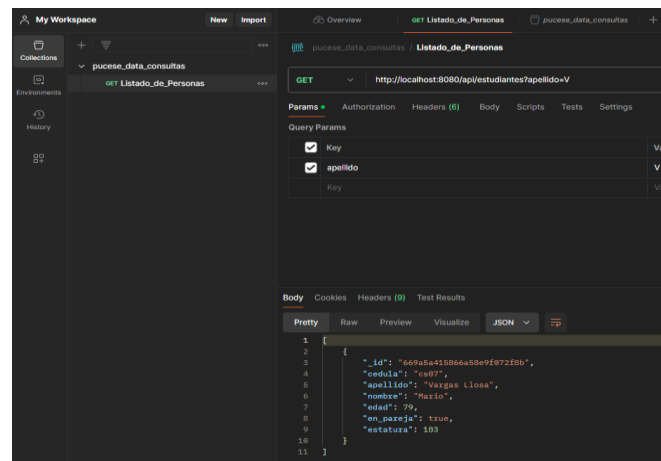
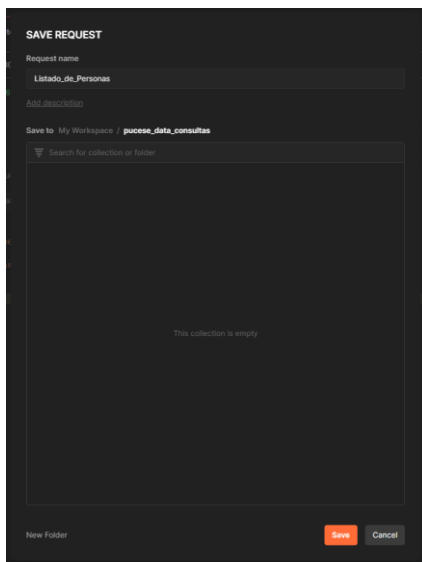
Siguiendo estos pasos, no deberías tener algún problema; incluso, podemos filtrar algunos valores según su llave y valor:



Para organizar nuestra información, podemos agregarla a una conexión en Postman, para ello, damos al botón **Create Collection** o al botón **+** y asignar un nombre adecuado a la colección:



Damos al botón **Save** o **control + s**, colocar un nombre a la solicitud y guardarlo en la colección:



Con esto está finalizado la práctica en clase.