

Microservicios y contenedores 2020

27 SEPTIEMBRE

Estudiante de Matemáticas Aplicadas y
Computación

Creado por: Brayan Quirino Muñoz



Microservicios

1. Lenguaje de programación y tecnologías necesarias

- **JavaScript, NodeJS, NPM** para desarrollar el API
- **Postman** para probar el API
- **Docker Desktop** para contenerizar el API

2. Mensajes de entrada y salida

Los mensajes básicos de un API son:

- **GET** (Conseguir)
- **POST** (Poner-insertar)
- **PUT** (Actualizar)
- **DELETE** (Borrar)

Cada uno realizado la función que su nombre dice, por ejemplo, si mandamos a llamar **GET** a nuestra API estaremos **consiguiendo** algún tipo de información.

En nuestra API usaremos mensajes del tipo **JSON** por dos razones:

1. JSON es un formato sencillo de manejar, escribir y de poca complejidad.
2. JSON es derivado de JavaScript y su uso con NodeJS se vuelve relativamente fácil.

Mediante el uso de **Express** y su método **Router** crearemos los mensajes de la siguiente forma:

```
router.get(endpoint, (req, res) => { ... })
```

Obtiene la base de datos.

```
router.get(endpoint+' /jsonplaceholder', (req, res) => { ... })
```

Obtiene datos de otra API.

```
router.put(endpoint+' / :id', (req, res) => { ... })
```

Actualiza un elemento de la base de datos.

```
router.post(endpoint, (req, res) => { ... })
```

Inserta un nuevo element a la base de datos.

```
router.delete(endpoint+' / :id', (req, res) => { ... })
```

Borra un elemento de la base de datos.

Donde `endpoint` es igual a `"/api/sps/helloworld/v1"`.

3. Endpoint

Para definir un endpoint en JavaScript basta con ponerlo en el método que responderá:

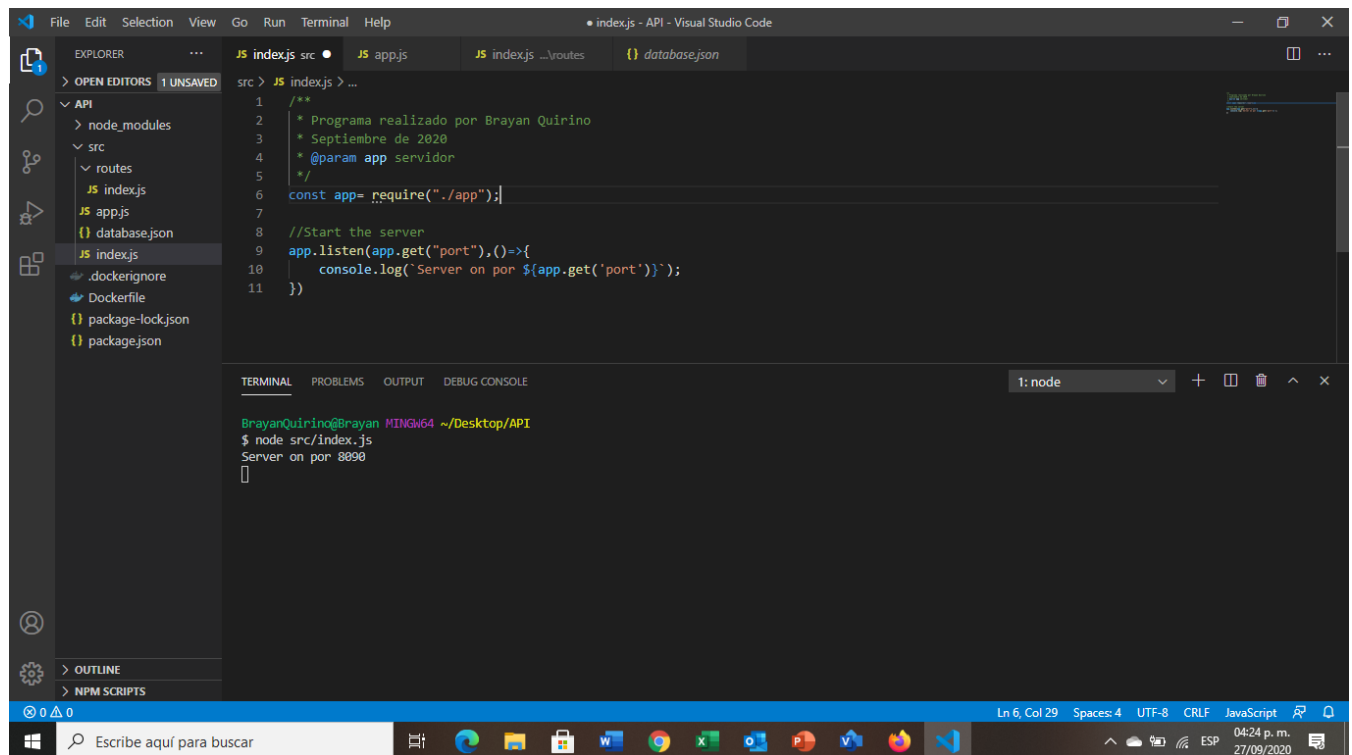
```
router.get(endpoint, (req, res) => { ... })
```

O podemos escribirlo de la siguiente manera:

```
router.get('/api/sps/helloworld/v1', (req, res) => { ... })
```

4. Desplegar el API

Una vez escrito el código necesitaremos levantar el servicio con el comando **node index.js** que ejecutaremos en una terminal dentro de la raíz del proyecto:



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays the project structure with folders like 'API', 'node_modules', 'src', and 'routes'. The 'index.js' file is selected in the 'src' folder. The main editor area shows the content of 'index.js', which includes a comment block and a simple Express.js server setup. The terminal at the bottom shows the command `node src/index.js` being executed, resulting in the output 'Server on port 8090'.

```
1  /**
2   * Programa realizado por Brayan Quirino
3   * Septiembre de 2020
4   * @param app servidor
5   */
6  const app= require("./app");
7
8  //Start the server
9  app.listen(app.get("port"),()=>{
10    console.log(`Server on port ${app.get('port')}`);
11  })
```

```
BrayanQuirino@Brayan MINGW64 ~/Desktop/API
$ node src/index.js
Server on port 8090
```

Podemos ver que el servidor se levanto con éxito, otra forma de verificarlo es abriendo un navegador y escribiendo la dirección **http://localhost:8090/api/sps/helloworld/v1** que nos arrojará el siguiente JSON:



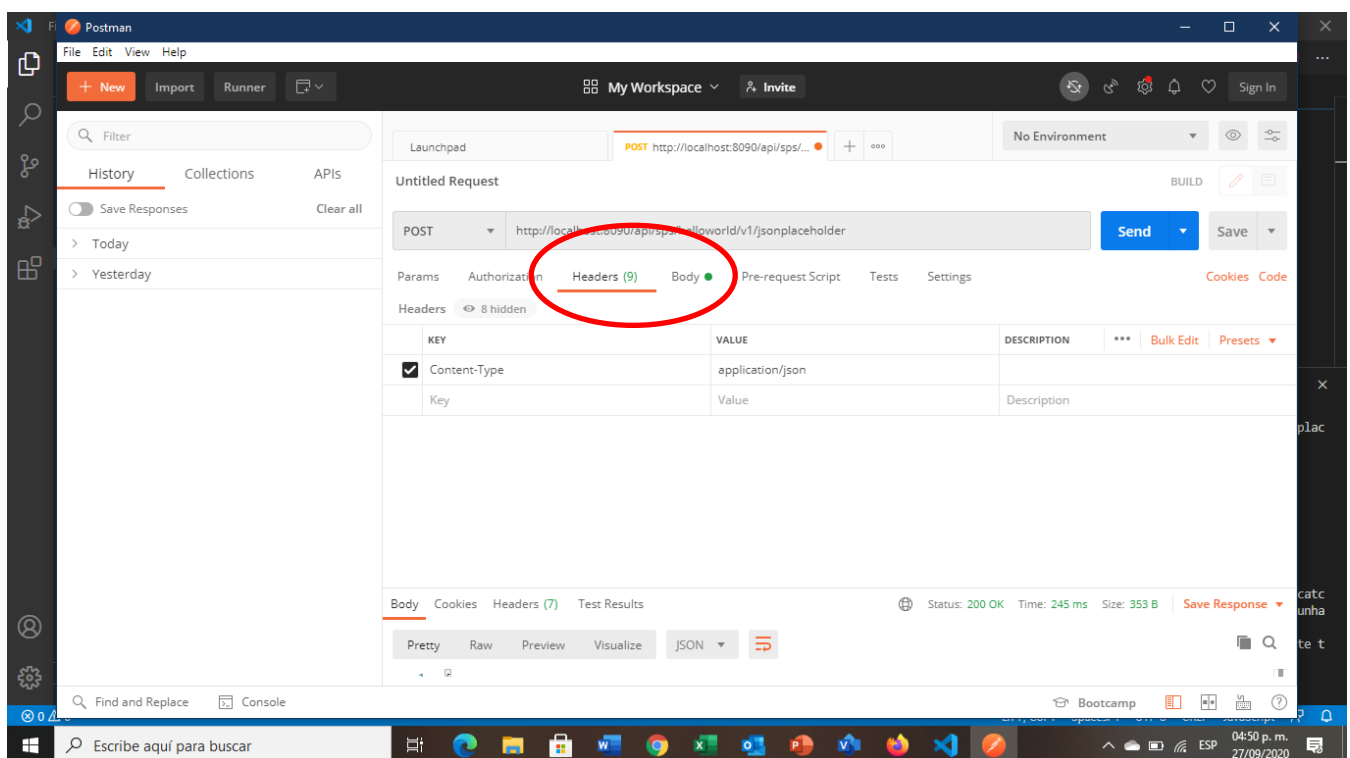
The screenshot shows a web browser window with the address bar set to `localhost:8090/api/sps/helloworld/v1`. The page content displays a JSON array of employee objects.

```
{ "employees": [ { "name": "Axel", "age": 16, "id": 0 }, { "name": "Jonathan", "age": 19, "id": 1 }, { "name": "Brayan", "age": 22, "id": 2 } ] }
```

5. Probar el API con un cliente REST (Postman)

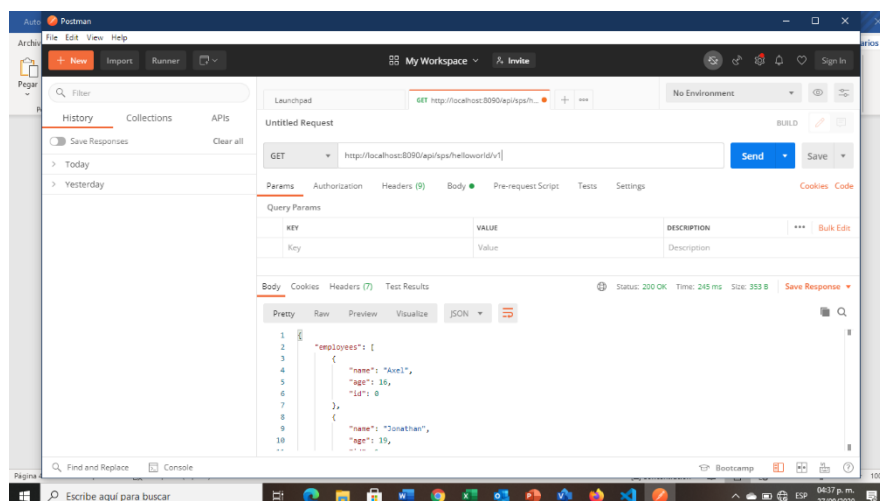
Para usar Postman basta con inicializarlo. Postman es un software que nos permite simular las peticiones de otras aplicaciones hacia nuestra API, con esta lógica enviaremos información desde este software y su vez recibiremos información desde nuestra API.

En cada solicitud seleccionaremos el método y colocaremos la dirección correspondiente. Cuando usemos métodos que envíen información del tipo JSON tendremos que modificar algunos campos, entraremos a **Headers**, en **Key** seleccionaremos la opción *Content-Type* y en **Value** *application/json*. Posteriormente en la sección **Body** pondremos nuestro JSON.

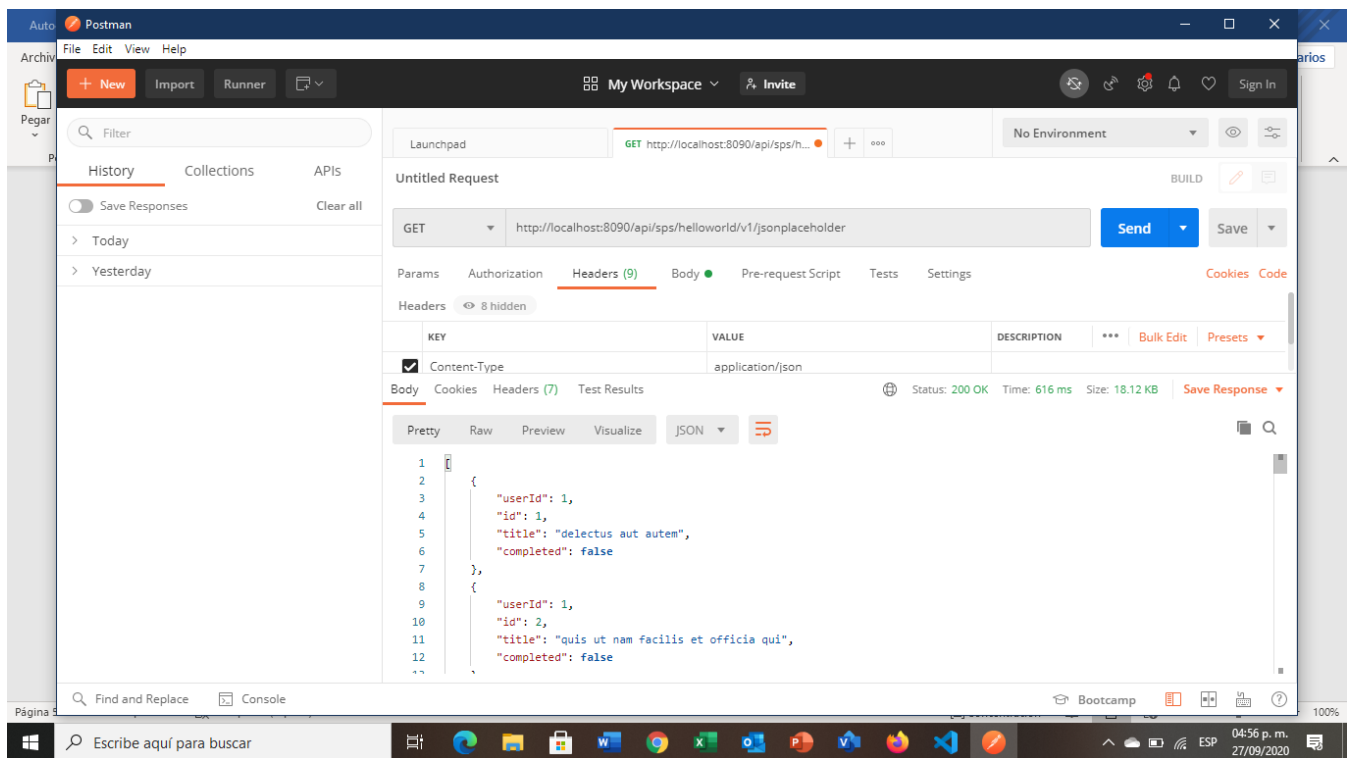


Método **GET**, dirección **`http://localhost:8090/api/sps/helloworld/v1`**:

La respuesta aparecerá en la parte inferior de la pantalla.

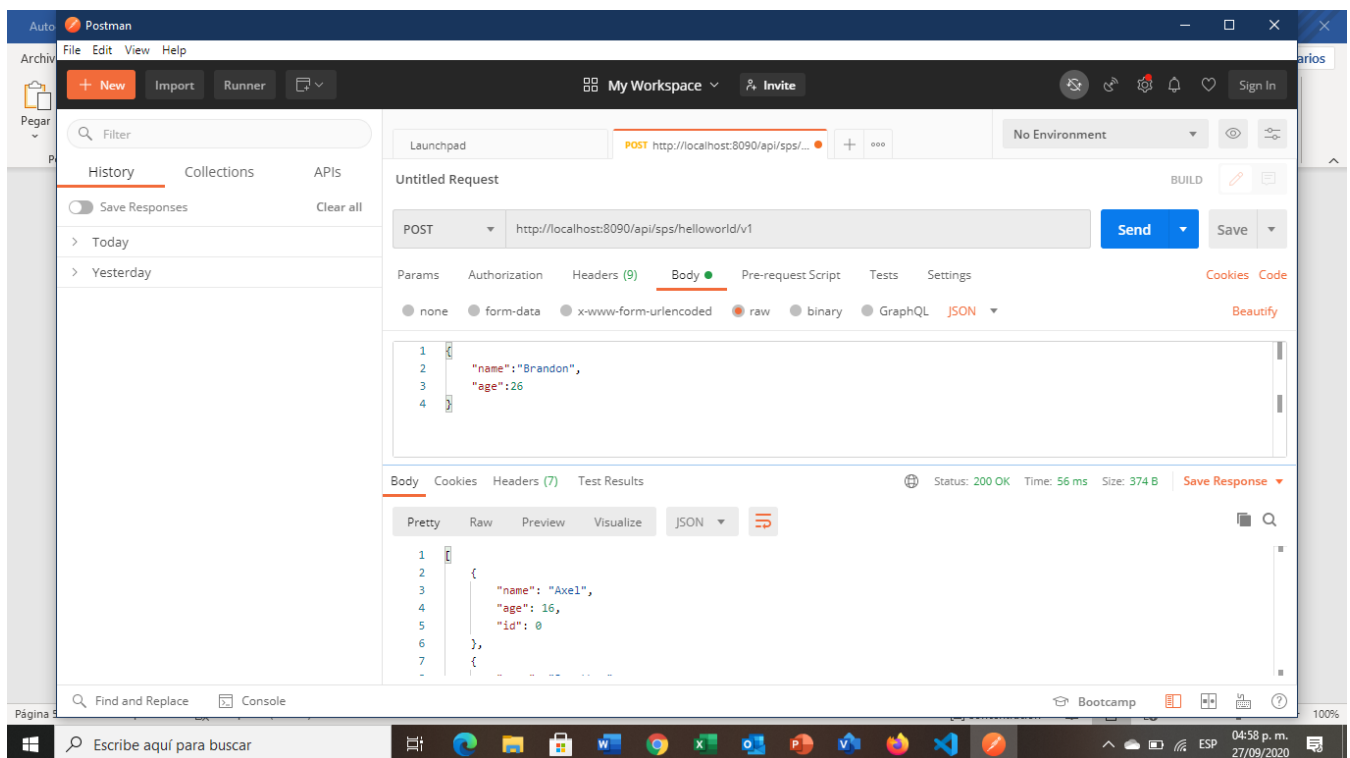


Método **GET** hacia otra API, dirección **http://localhost:8090/api/sps/helloworld/v1/jsonplaceholder**:



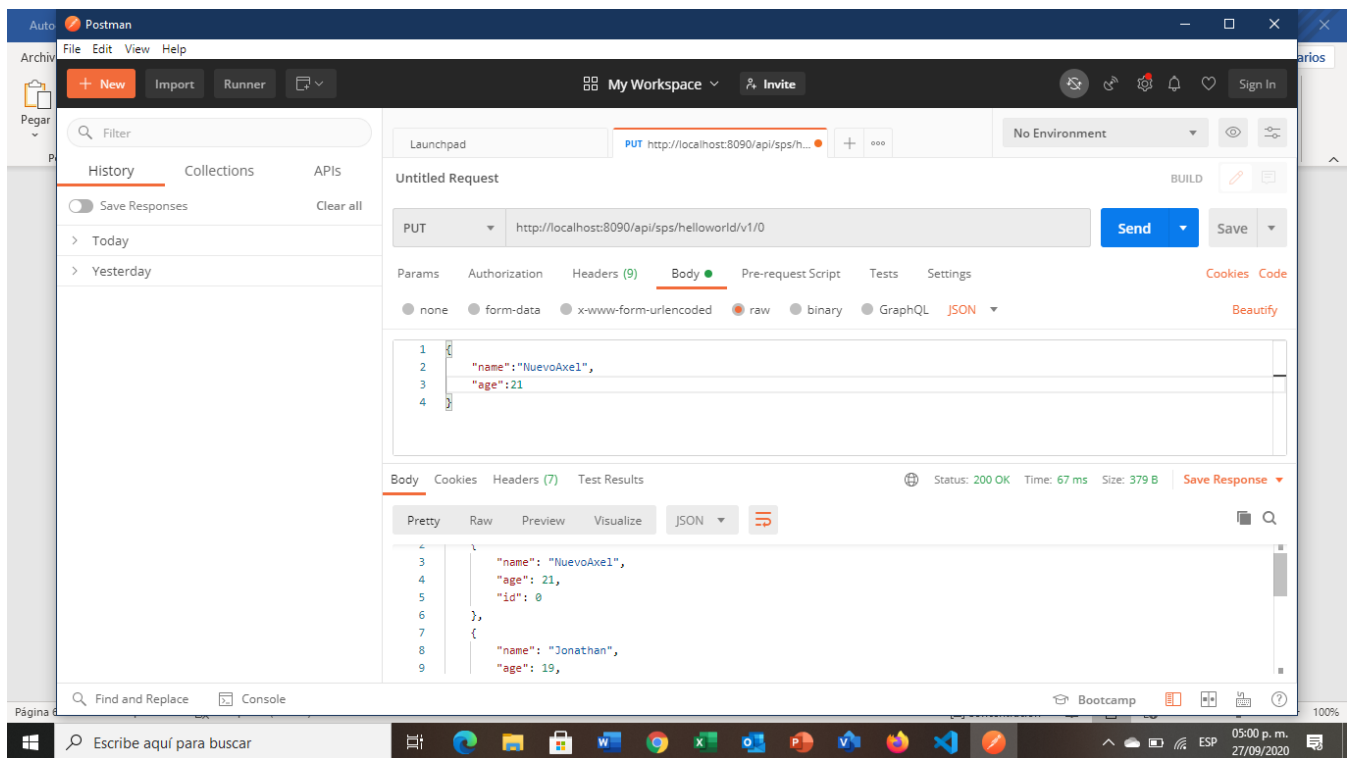
Método **POST**, dirección **http://localhost:8090/api/sps/helloworld/v1**:

JSON enviado { "name": "Brandon", "age": 26 }

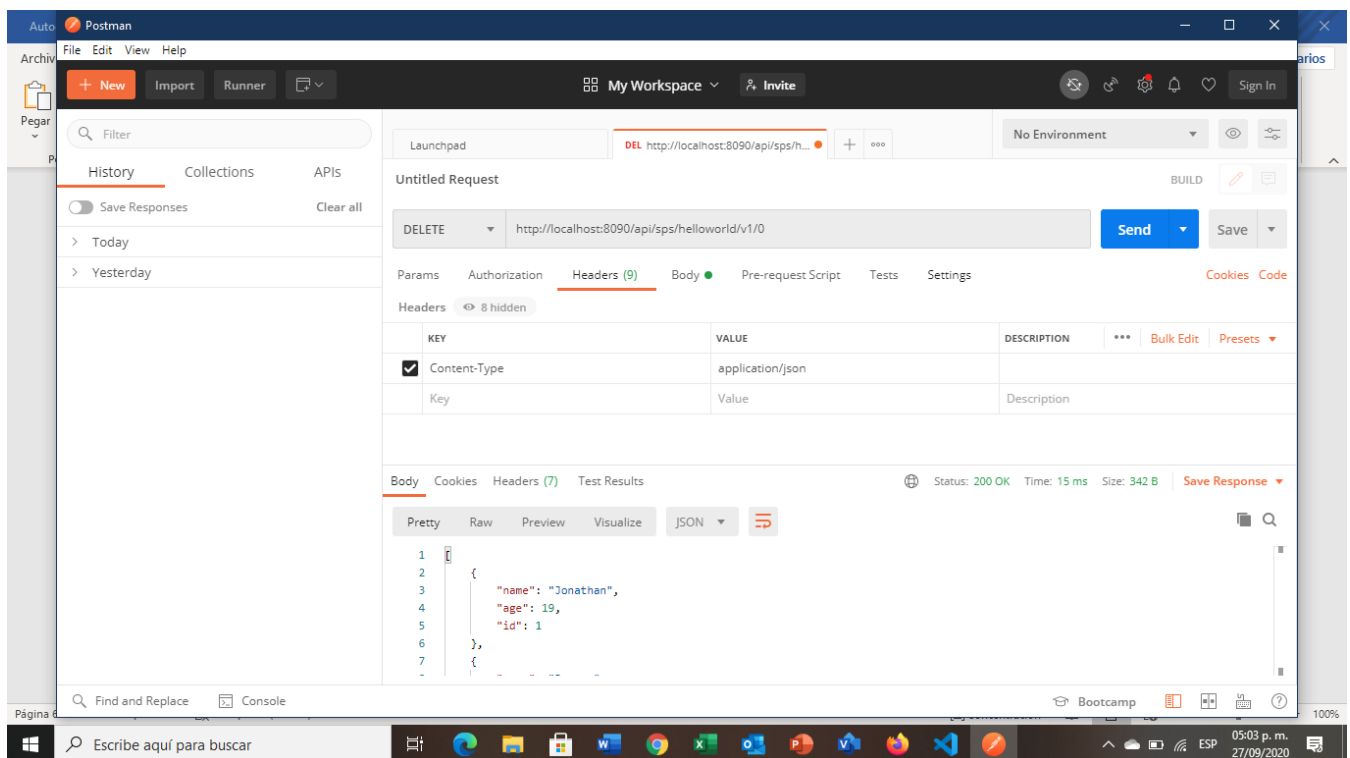


Método **PUT**, dirección **http://localhost:8090/api/sps/helloworld/v1/jsonplaceholder/0**:

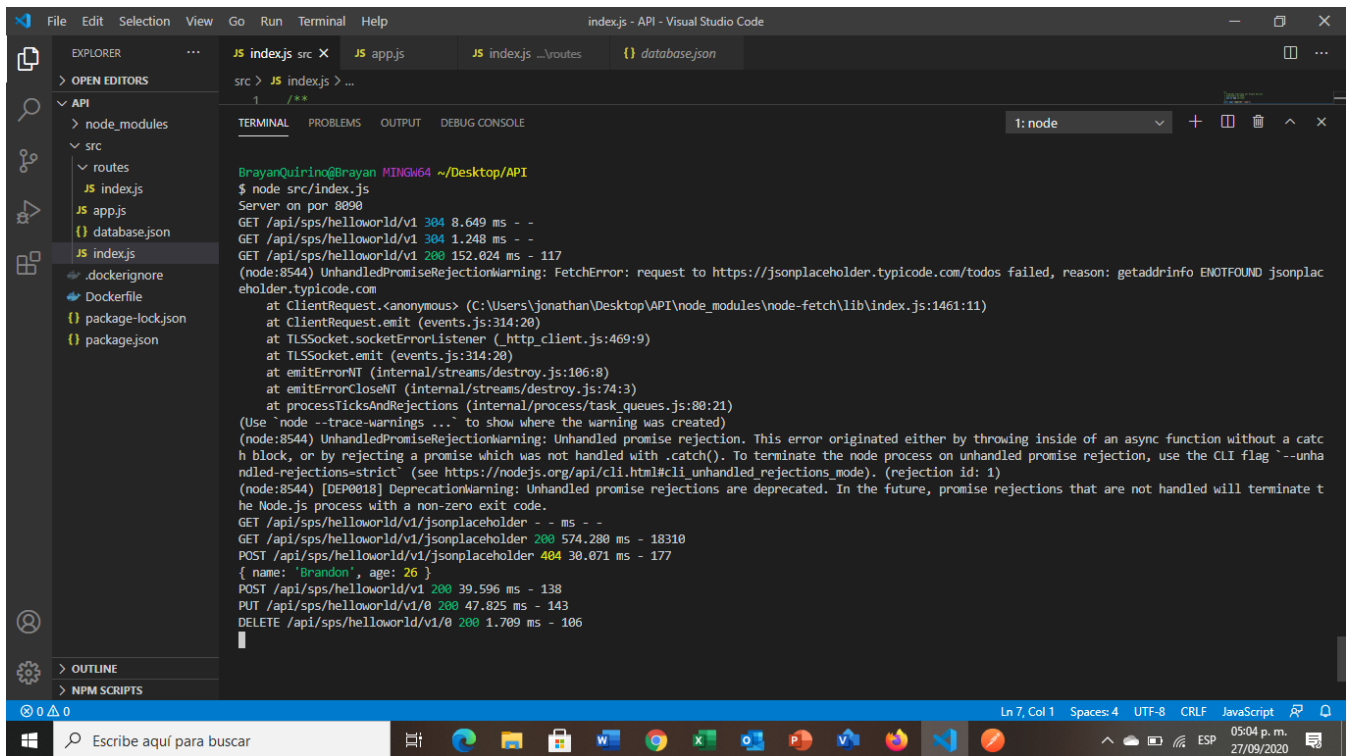
JSON enviado { "name": "NuevoAxel", "age": 21 }



Método **DELETE**, dirección **http://localhost:8090/api/sps/helloworld/v1/jsonplaceholder/0**:



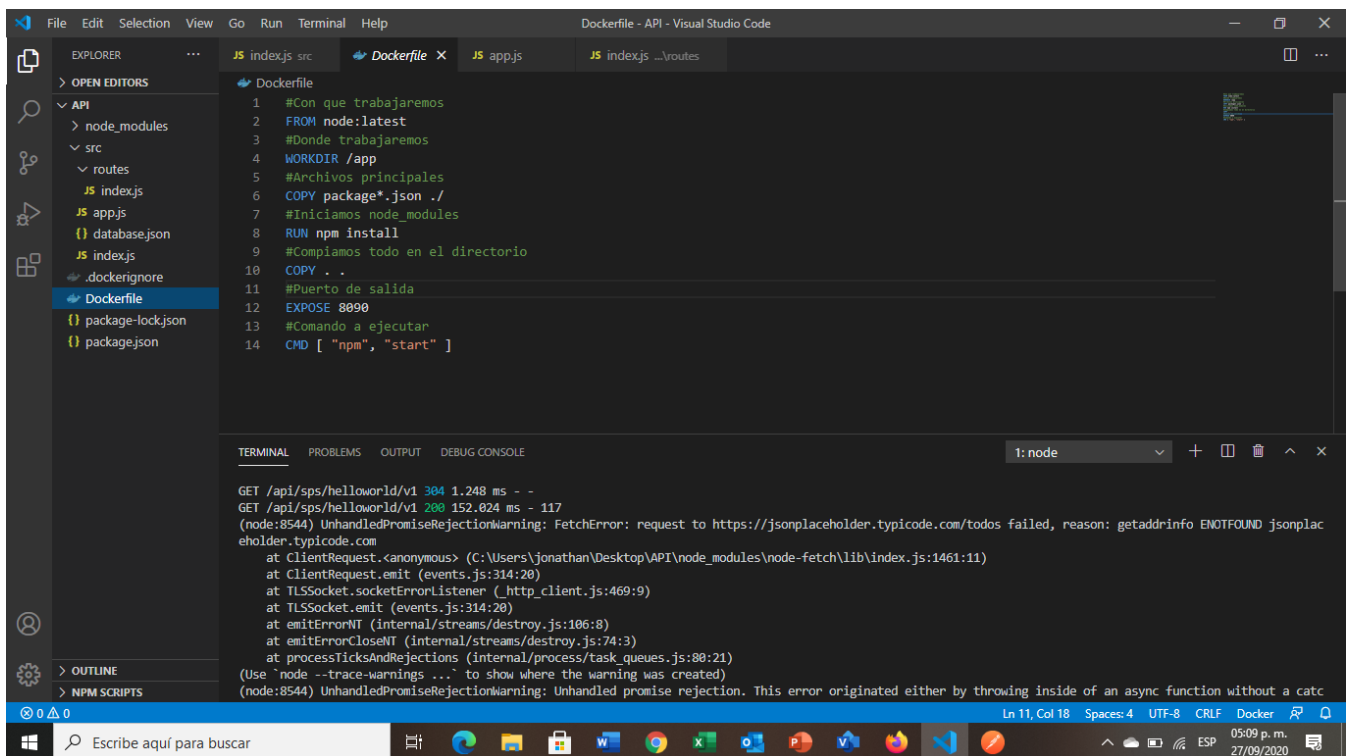
Para poder hacer estas pruebas es necesario que el servidor o servicio este levantado.



```
src > JS index.js > ...
1 /**
2  *
3  * @param {string} url
4  * @param {Object} options
5  * @param {Function} callback
6  * @returns {Promise}
7  */
8
9  // GET /api/sps/helloworld/v1
10 GET /api/sps/helloworld/v1 304 8.649 ms - -
11 GET /api/sps/helloworld/v1 304 1.248 ms - -
12 GET /api/sps/helloworld/v1 200 152.024 ms - 117
13 (node:8544) UnhandledPromiseRejectionWarning: FetchError: request to https://jsonplaceholder.typicode.com/todos failed, reason: getaddrinfo ENOTFOUND jsonplaceholder.typicode.com
14   at ClientRequest.<anonymous> (C:\Users\jonathan\Desktop\API\node_modules\node-fetch\lib\index.js:1461:11)
15   at ClientRequest.emit (events.js:314:20)
16   at TLSSocket.socketErrorListener (_http_client.js:469:9)
17   at TLSSocket.emit (events.js:314:20)
18   at emitErrorNT (internal/streams/destroy.js:106:8)
19   at emitErrorCloseNT (internal/streams/destroy.js:74:3)
20   at processTicksAndRejections (internal/process/task_queues.js:80:21)
21 (Use 'node --trace-warnings ...' to show where the warning was created)
22 (node:8544) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch block, or by rejecting a promise which was not handled with .catch(). To terminate the node process on unhandled promise rejection, use the CLI flag '--unhandled-rejections=strict' (see https://nodejs.org/api/cli.html#cli_unhandled_rejections_mode). (rejection id: 1)
23 (node:8544) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not handled will terminate the Node.js process with a non-zero exit code.
24 GET /api/sps/helloworld/v1/jsonplaceholder - - ms - -
25 GET /api/sps/helloworld/v1/jsonplaceholder 200 574.280 ms - 18310
26 POST /api/sps/helloworld/v1/jsonplaceholder 404 30.071 ms - 177
27   { name: 'Brandon', age: 26 }
28 POST /api/sps/helloworld/v1 200 39.596 ms - 138
29 PUT /api/sps/helloworld/v1/0 200 47.825 ms - 143
30 DELETE /api/sps/helloworld/v1/0 200 1.709 ms - 106
```

6. Contenerizar la aplicación; Expón tu servicio por el puerto 8090

Para contenerizar el API tendremos que haber arrancado Docker en nuestra computadora, también necesitaremos escribir un archivo del tipo Docker en la raíz del proyecto; **Dockerfile**. Con las siguientes especificaciones:

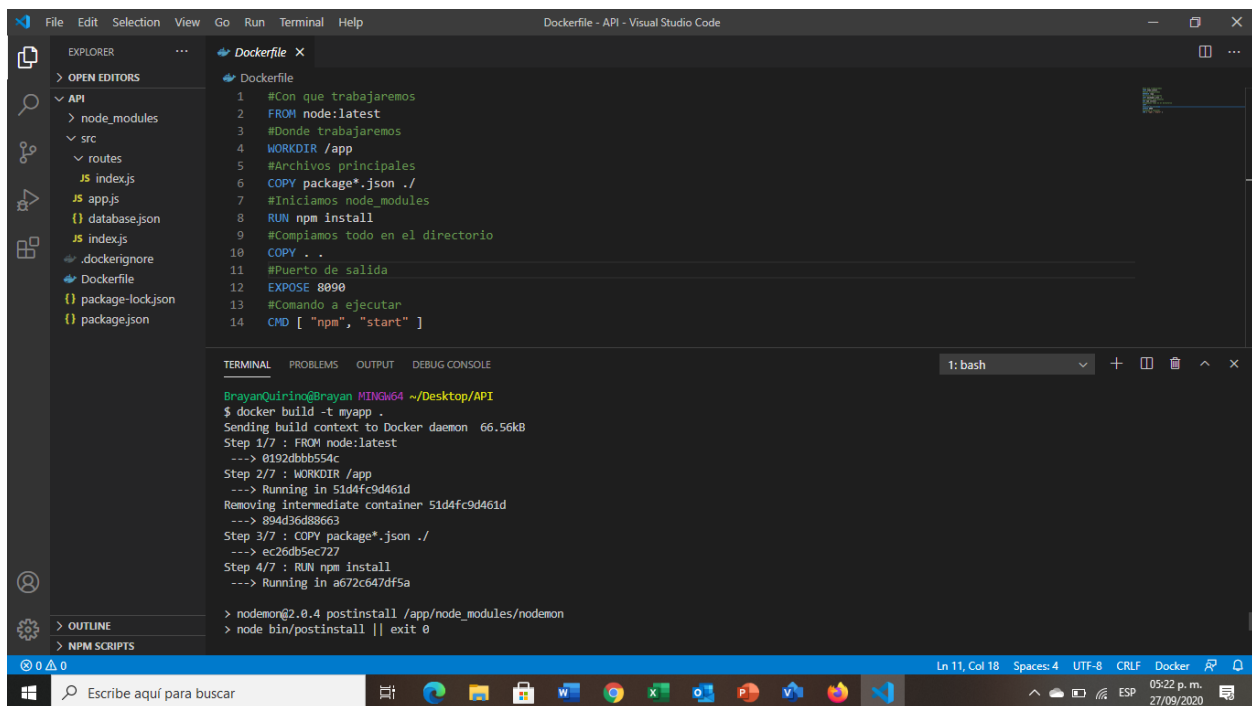


```
1 #Con que trabajaremos
2 FROM node:latest
3 #Donde trabajaremos
4 WORKDIR /app
5 #Archivos principales
6 COPY package*.json ./
7 #Iniciamos node_modules
8 RUN npm install
9 #Compiamos todo en el directorio
10 COPY . .
11 #Puerto de salida
12 EXPOSE 8090
13 #Comando a ejecutar
14 CMD [ "npm", "start" ]
```

```
GET /api/sps/helloworld/v1 304 1.248 ms - -
GET /api/sps/helloworld/v1 200 152.024 ms - 117
(node:8544) UnhandledPromiseRejectionWarning: FetchError: request to https://jsonplaceholder.typicode.com/todos failed, reason: getaddrinfo ENOTFOUND jsonplaceholder.typicode.com
  at ClientRequest.<anonymous> (C:\Users\jonathan\Desktop\API\node_modules\node-fetch\lib\index.js:1461:11)
  at ClientRequest.emit (events.js:314:20)
  at TLSSocket.socketErrorListener (_http_client.js:469:9)
  at TLSSocket.emit (events.js:314:20)
  at emitErrorNT (internal/streams/destroy.js:106:8)
  at emitErrorCloseNT (internal/streams/destroy.js:74:3)
  at processTicksAndRejections (internal/process/task_queues.js:80:21)
(Use 'node --trace-warnings ...' to show where the warning was created)
(node:8544) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch
```


Una vez listo ejecutaremos el comando: **docker build -t mypp .**

Donde **myapp** será el nombre de nuestra imagen (contenedor).



The screenshot shows the Visual Studio Code interface with a Dockerfile open in the editor. The Dockerfile contains the following instructions:

```
1 #Con que trabajaremos
2 FROM node:latest
3 #Donde trabajaremos
4 WORKDIR /app
5 #Archivos principales
6 COPY package*.json ./
7 #Iniciamos node_modules
8 RUN npm install
9 #Compiamos todo en el directorio
10 COPY . .
11 #Puerto de salida
12 EXPOSE 8090
13 #Comando a ejecutar
14 CMD [ "npm", "start" ]
```

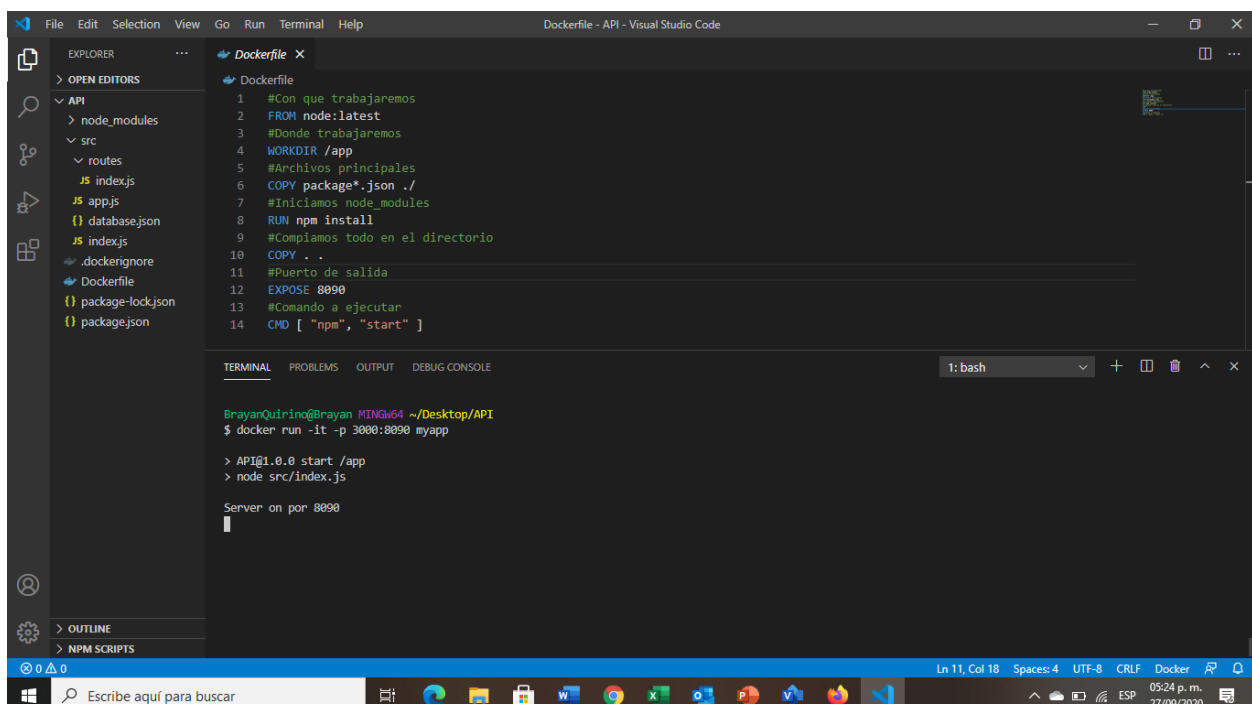
The terminal output shows the execution of the `docker build -t mypp .` command, which successfully builds the image. The output includes the following steps:

```
BrayanQuirino@Brayan MINGW64 ~/Desktop/API
$ docker build -t mypp .
Sending build context to Docker daemon 66.56kB
Step 1/7 : FROM node:latest
----> 0102dbb554c
Step 2/7 : WORKDIR /app
----> Running in 51d4fc9d461d
Removing intermediate container 51d4fc9d461d
----> B94d36d88663
Step 3/7 : COPY package*.json ./
----> ec26db5ec727
Step 4/7 : RUN npm install
----> Running in a672c647df5a
> nodemon@2.0.4 postinstall /app/node_modules/nodemon
> node bin/postinstall || exit 0
```

Una vez que termine de ejecutarse el comando iniciaremos el microservicio dentro del contenedor con el comando **docker -it -p 8090:8090 myapp.**

Donde **-it o -i** es una opción para ver en consola las peticiones hacia nuestro servicio.

8090 (Puerto de salida en nuestra computadora) **:8090** (Puerto expuesto en el contenedor).



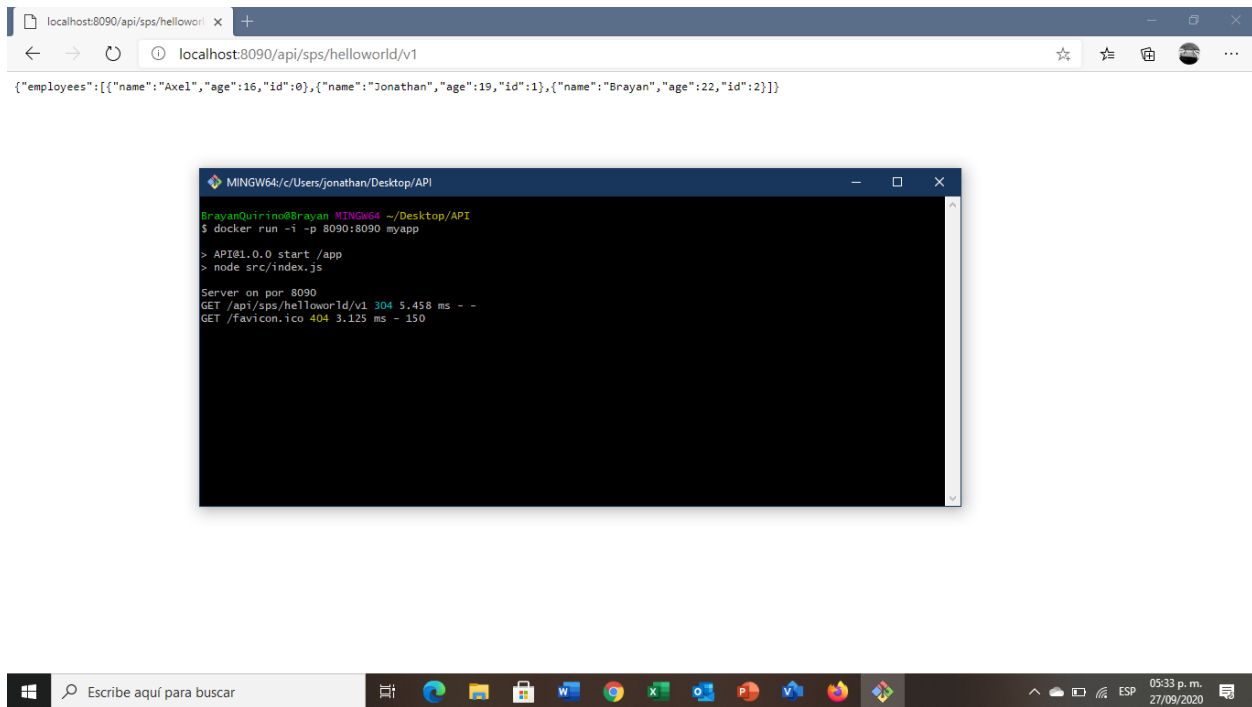
The screenshot shows the Visual Studio Code interface with the same Dockerfile as before. The terminal output shows the execution of the `docker run -it -p 8090:8090 mypp` command, which successfully starts the container. The output includes the following steps:

```
BrayanQuirino@Brayan MINGW64 ~/Desktop/API
$ docker run -it -p 8090:8090 mypp
> API@1.0.0 start /app
> node src/index.js
Server on por 8090
```


7. Prueba la API corriendo en el contenedor

Ya que la API este corriendo en el contenedor podemos verificar su funcionamiento mediante Postman. Por cuestiones de espacio (RAM) en mi computador simplemente lo verificaremos por el buscador.

Método **GET**, dirección **http://localhost:8090/api/sps/helloworld/v1**:



Método **GET**, dirección **http://localhost:8090/api/sps/helloworld/v1/jsonplaceholder**:

