

DOCUMENTACIÓN DEL API (CRUD).

Juan Diego Álzate Suarez

Jaider Londoño Gutiérrez

CIAF

Ingeniería de software (Risaralda profesional)

Programación de Redes

25/11/2023

API (CRUD)

-En el siguiente documento se podrá visualizar un orden de pasos y de instrucciones que nos llevaron a la creación de una API, la cual implementa operaciones CRUD básicas (Crear, Leer, Actualizar, Eliminar) para una entidad llamada 'productos'. Utiliza un array para almacenar los productos. La API la creamos, la escribimos y la utilizamos en el code visual, utilizando el punto js, una vez creada le hicimos las pruebas de funcionamiento en la terminal del code visual y en la aplicación de Postman en la cual nos permite poder hacer varias funciones con la API como por ejemplo: Probar colecciones o catálogos APIs, ya sea para Frontend o Backend, clasificar y organizar en carpetas funciones y módulos de los servicios web, ofrece la posibilidad de gestionar el ciclo de vida de la API generar documentos y monitorizar las APIs y finalmente poder trabajar con entornos que permitan compartir, en un entorno cloud, información con los miembros del equipo que forman parte del desarrollo.

❖ Herramientas:

- **Node js:** Nos permite trabajar en tiempo de ejecución, de código abierto y multi-plataforma.



- **Visual Studio Code:** Nos permite crear la API.

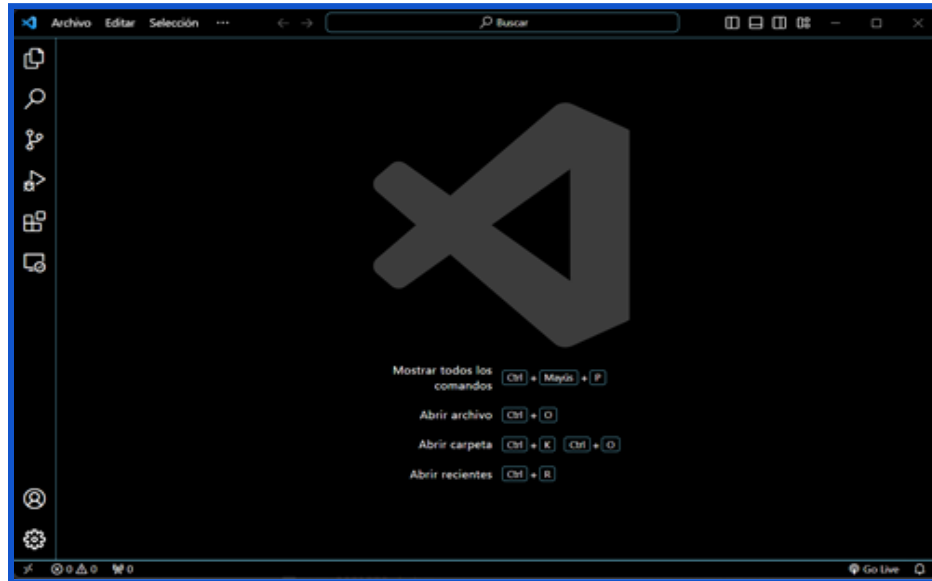


- **Postman:** Nos permite mejor a detalle el funcionamiento de la API creada.

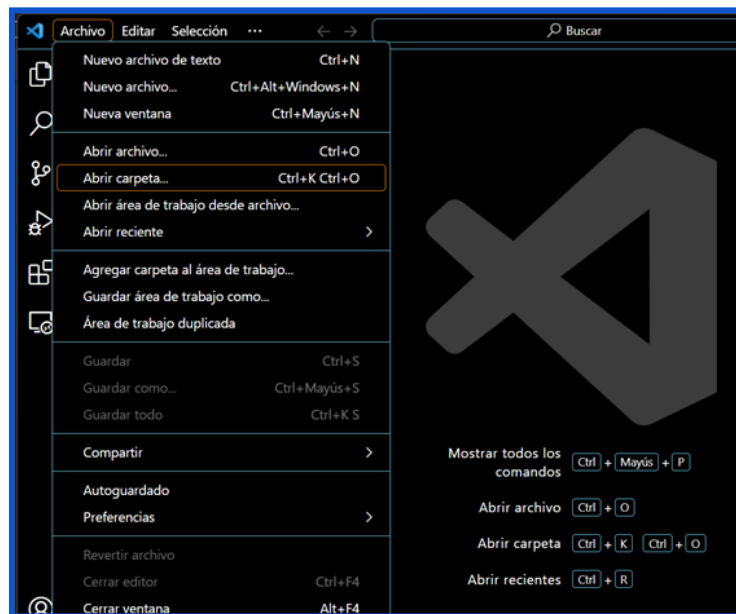


-Los pasos en que se crearon la API (CRUD) son los siguientes:

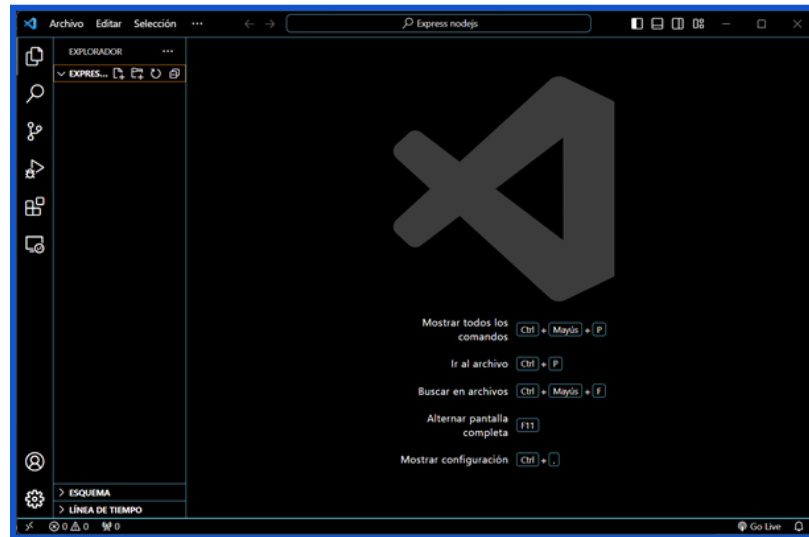
❖ **Paso 1:** Abrimos Visual Studio Code.



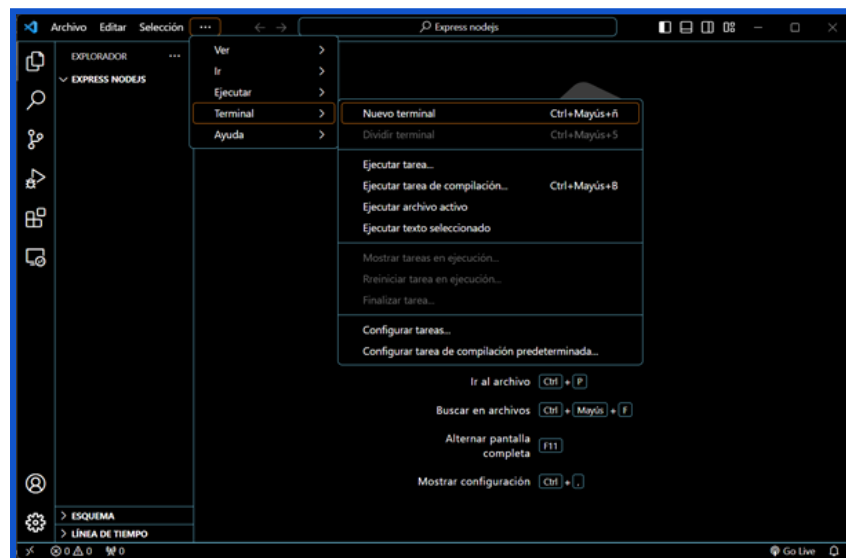
❖ **Paso 2:** Creamos una carpeta en el visual y la abrimos.



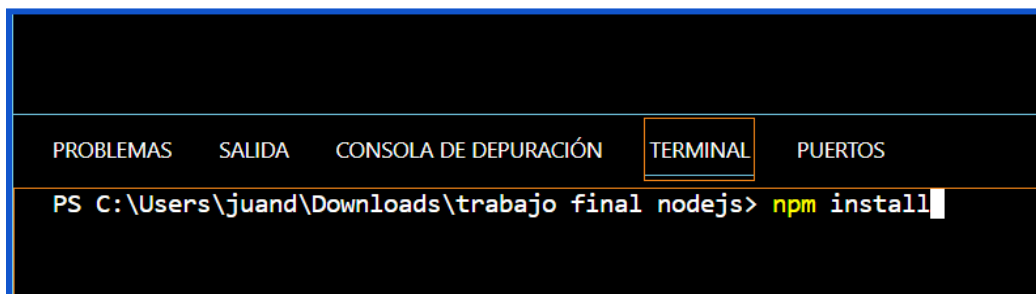
❖ **Paso 3:** Descargamos las librerías.



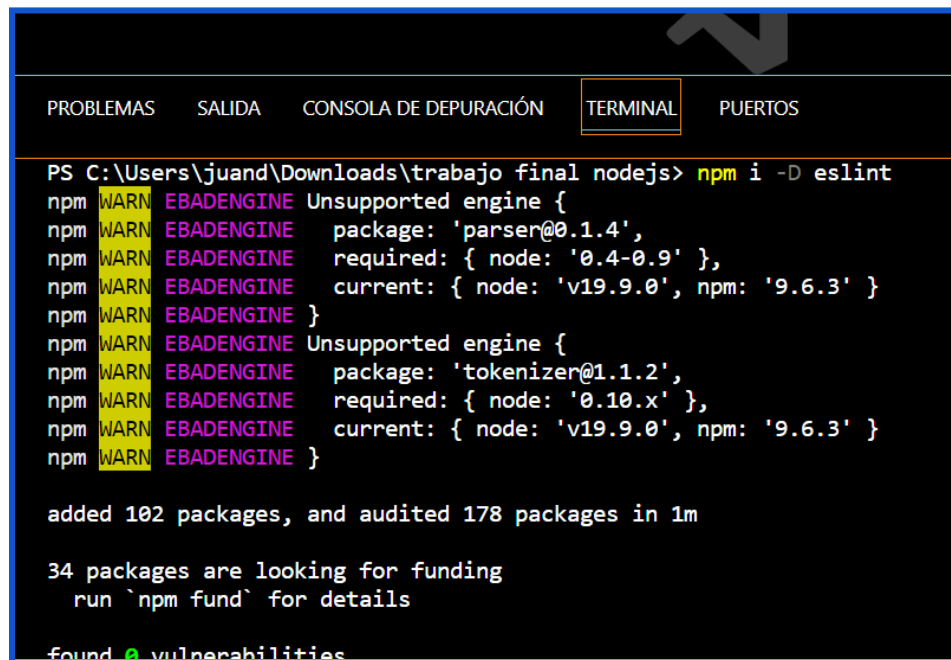
❖ **Paso 4:** Abrimos la terminal ctrl + mayús+ñ.



❖ **Paso 5:** Ingresamos el comando npm install para instalar las librerías.



- ❖ **Paso 6:** Ingresamos el comando `npm i -D eslint` que nos descargara todas las librerías entre ellas la de `express`.



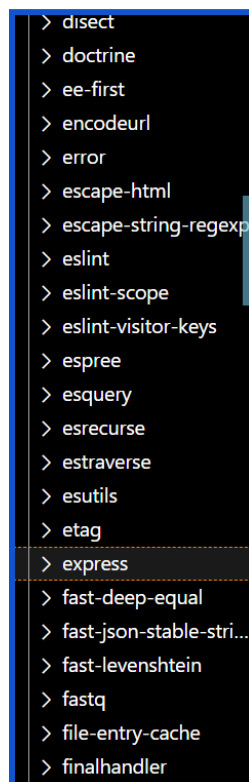
```
PS C:\Users\juand\Downloads\trabajo final nodejs> npm i -D eslint
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'parser@0.1.4',
npm WARN EBADENGINE   required: { node: '0.4-0.9' },
npm WARN EBADENGINE   current: { node: 'v19.9.0', npm: '9.6.3' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'tokenizer@1.1.2',
npm WARN EBADENGINE   required: { node: '0.10.x' },
npm WARN EBADENGINE   current: { node: 'v19.9.0', npm: '9.6.3' }
npm WARN EBADENGINE }

added 102 packages, and audited 178 packages in 1m

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

found 0 vulnerabilities
```

- Después de terminar la descarga se crea una nueva carpeta en esa carpeta está `express`.



```
> dissect
> doctrine
> ee-first
> encodeurl
> error
> escape-html
> escape-string-regexp
> eslint
> eslint-scope
> eslint-visitor-keys
> espree
> esquery
> esrecurse
> estraverse
> esutils
> etag
> express
> fast-deep-equal
> fast-json-stable-str...
> fast-levenshtein
> fastq
> file-entry-cache
> finalhandler
```

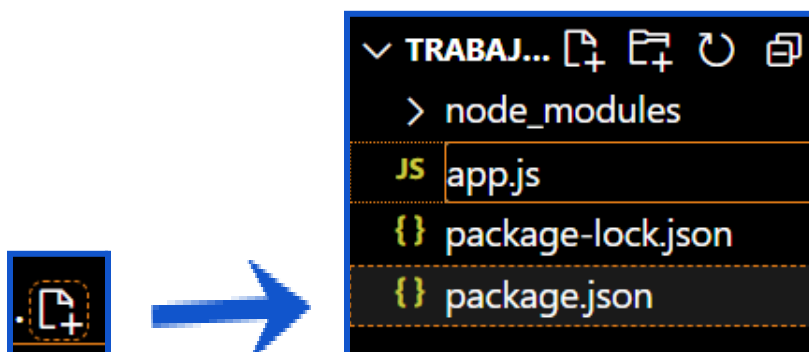
- ❖ **Paso 7:** Ingresamos el comando `npm install body-parser` para analizar las solicitudes HTTP en formato JSON.

```
PS C:\Users\juand\Downloads\trabajo final nodejs> npm install body-parser
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'parser@0.1.4',
npm WARN EBADENGINE   required: { node: '0.4-0.9' },
npm WARN EBADENGINE   current: { node: 'v19.9.0', npm: '9.6.3' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'tokenizer@1.1.2',
npm WARN EBADENGINE   required: { node: '0.10.x' },
npm WARN EBADENGINE   current: { node: 'v19.9.0', npm: '9.6.3' }
npm WARN EBADENGINE }
```

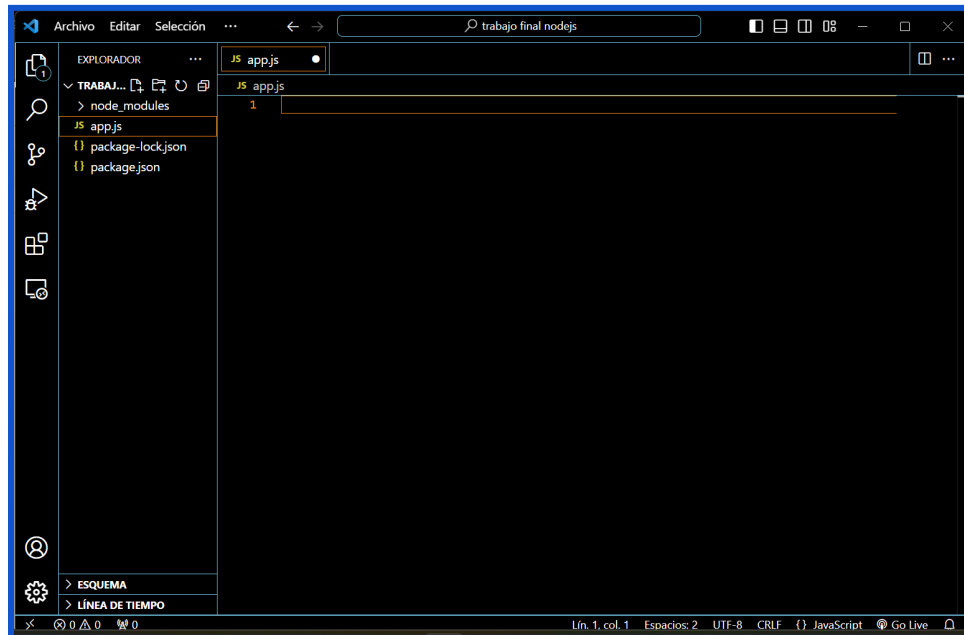
- Si queremos ahorrar tiempo, ingresamos el comando `npm install express bodyparser` que instala los dos al mismo tiempo `express` y `body-parser`.

```
PS C:\Users\juand\Downloads\trabajo final nodejs> npm install express body-parser
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'parser@0.1.4',
npm WARN EBADENGINE   required: { node: '0.4-0.9' },
npm WARN EBADENGINE   current: { node: 'v19.9.0', npm: '9.6.3' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'tokenizer@1.1.2',
npm WARN EBADENGINE   required: { node: '0.10.x' },
npm WARN EBADENGINE   current: { node: 'v19.9.0', npm: '9.6.3' }
npm WARN EBADENGINE }
```

- ❖ **Paso 8:** Presionamos el siguiente icono para crear un nuevo archivo que llamaremos `app.js`.



- Nos queda similar a:



❖ **Paso 9:** Creamos nuestras variables globales.

```
const express = require('express');//variable asignado a express
const bodyParser = require('body-parser');//variable asignada a body-parser
const app = express();//igualacion de la variable app el metodo express
const port = 3000;//creamos una variable con el puerto en este caso 300 de
//localhost
```

❖ **Paso 10:** Creamos un array y configuramos la variable app con body-parser.

```
const productos = [];//creamos un array para almacenar los productos
app.use(bodyParser.json());// Configura el body-parser
//para analizar solicitudes en formato JSON
```

❖ **Paso 11:** Primero crearemos la ruta /productos para obtener todos los datos que contiene el array.

```
app.get('/productos', (req, res) => {
  res.json(productos);// Devuelve la lista de productos en formato JSON
});
```

- ❖ **Paso 12:** Creamos la ruta `producto/:id` para consultar un producto ingresando el id después de `/` dentro de ella creamos dos variables una para el id y otra para producto.

```
app.get('/productos/:id', (req, res) => {  
  const id = req.params.id; // Obtiene el ID de los parámetros de la URL  
  
  // Busca el producto en el array por ID  
  const producto = productos.find(p => p.id === parseInt(id));  
  
});
```

- ❖ **Paso 13:** Creamos dentro un condicional que se encargará de buscar el producto en el array.

```
app.get('/productos/:id', (req, res) => {  
  const id = req.params.id; // Obtiene el ID de los parámetros de la URL  
  
  // Busca el producto en el array por ID  
  const producto = productos.find(p => p.id === parseInt(id));  
  
  if (producto) { // si el producto se encuentra  
    res.json(producto); // devuelve el producto en formato JSON  
  } else {  
    res.status(404).json({ message: 'Producto no encontrado' }); // devuelve error  
  }  
});
```

- ❖ **Paso 14:** Creamos una nueva ruta donde podremos crear un nuevo producto y a parece lo siguiente `/producto/agregar` dentro creamos una variable **NuevoProducto** igualada para contener el cuerpo de la solicitud, luego asignamos un id que se incrementa en 1 agregamos el nuevo producto al array y devolvemos el producto creado.

```
app.post('/productos/agregar', (req, res) => {  
  const nuevoProducto = req.body; // Obtiene el nuevo producto del cuerpo de la solicitud  
  nuevoProducto.id = productos.length + 1; // Asigna un ID al nuevo producto  
  productos.push(nuevoProducto); // Agrega el nuevo producto al array de productos  
  res.status(201).json(nuevoProducto); // Devuelve el nuevo producto y un código de estado 201  
});
```


- ❖ **Paso 15:** Creamos una ruta para actualizar un producto de acuerdo a su id productos/buscar/:id.

```
app.put('/productos/buscar/:id', (req, res) => {  
  const id = req.params.id; // Obtiene el ID de los parámetros de la URL  
  
  // Encuentra el índice del producto en el array por ID  
  const productoIndex = productos.findIndex(p => p.id === parseInt(id));  
  
  if (productoIndex !== -1) {  
    // Actualiza el producto con los datos de la solicitud  
    productos[productoIndex] = { ...productos[productoIndex], ...req.body };  
    res.json(productos[productoIndex]); // Devuelve el producto actualizado en formato JSON  
  } else {  
    // Devuelve un mensaje de error si el producto no se encuentra  
    res.status(404).json({ message: 'Producto no encontrado' });  
  }  
});
```

- ❖ **Paso 16:** Creamos una ruta para eliminar un producto con la ruta productos/eliminar/.

```
app.delete('/productos/eliminar/:id', (req, res) => {  
  
});
```

- Creamos dos variables una que obtendrá el id que ingresemos y otra que encuentra el producto en el array.

```
app.delete('/productos/eliminar/:id', (req, res) => {  
  const id = req.params.id; // Obtiene el ID de los parámetros de la URL  
  // Encuentra el índice del producto en el array por ID  
  const productoIndex = productos.findIndex(p => p.id === parseInt(id));  
  
});
```

- Creamos un condicional (**productoIndex !== -1**) que verifica si se encontró el elemento en el array con el método **findIndex()** que devuelve **-1** si no se encuentra ningún elemento que cumple la condición si se cumple elimina el producto y devuelve qué producto se eliminó de lo contrario devuelve mensaje en código de estado **402**.

```

if (productoIndex !== -1) {
  // Elimina el producto del array y obtiene el producto eliminado
  const productoEliminado = productos.splice(productoIndex, 1);
  res.json(productoEliminado[0]); // Devuelve el producto eliminado en formato JSON
} else {
  // Devuelve un mensaje de error si el producto no se encuentra
  res.status(404).json({ message: 'Producto no encontrado' });
}
});

```

- ❖ **Paso 17:** Por último iniciamos el servidor en el puerto que ingresamos en la variable **port** e imprimimos un mensaje en la terminal.

```

// Inicia el servidor y escucha en el puerto especificado
app.listen(port, () => {
  console.log(`El programa del servidor se inicio correctamente`);
});

```

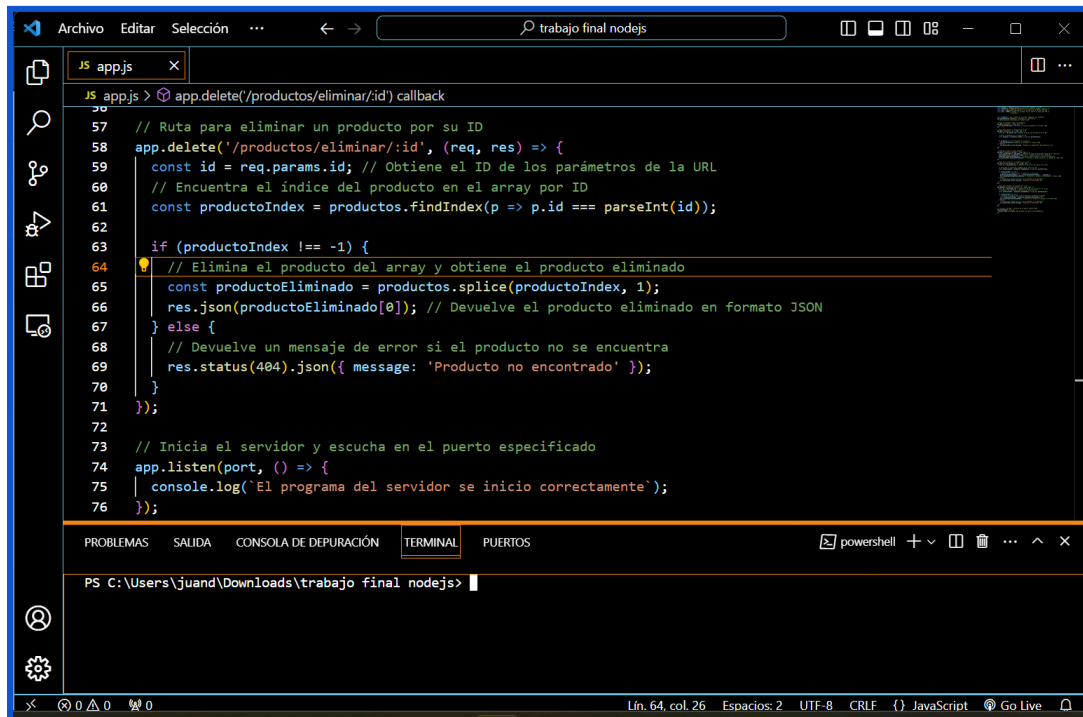
- ❖ **Paso 18:** Para probar el código utilizamos la combinación de teclas **”Ctrl + ñ”** o presionamos en los tres puntos que se encuentran en la parte superior derecha junto a el botón seleccionar:

Selección ...

- Este abrirá un menu desplegable donde seleccionaremos la terminal y posteriormente nueva terminal:



- Se abre la terminal:



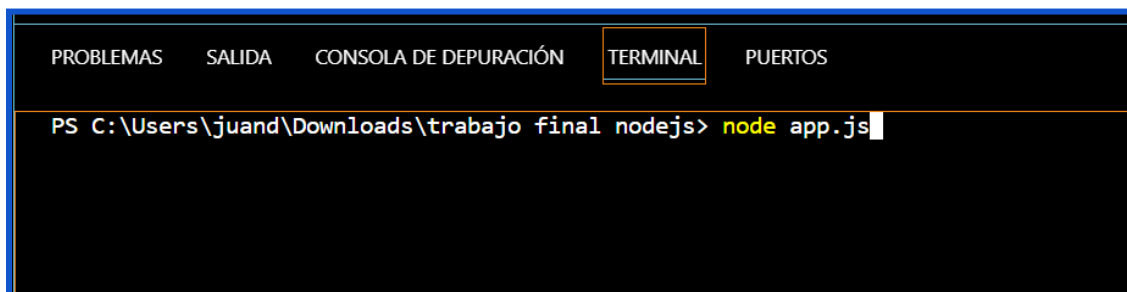
The screenshot shows the Visual Studio Code interface. The editor window displays a file named `app.js` with the following code:

```
56 app.delete('/productos/eliminar/:id') callback
57 // Ruta para eliminar un producto por su ID
58 app.delete('/productos/eliminar/:id', (req, res) => {
59   const id = req.params.id; // Obtiene el ID de los parámetros de la URL
60   // Encuentra el índice del producto en el array por ID
61   const productoIndex = productos.findIndex(p => p.id === parseInt(id));
62
63   if (productoIndex !== -1) {
64     // Elimina el producto del array y obtiene el producto eliminado
65     const productoEliminado = productos.splice(productoIndex, 1);
66     res.json(productoEliminado[0]); // Devuelve el producto eliminado en formato JSON
67   } else {
68     // Devuelve un mensaje de error si el producto no se encuentra
69     res.status(404).json({ message: 'Producto no encontrado' });
70   }
71 });
72
73 // Inicia el servidor y escucha en el puerto especificado
74 app.listen(port, () => {
75   console.log('El programa del servidor se inicio correctamente');
76 });
```

Below the editor, the TERMINAL tab is active, showing the command prompt:

```
PS C:\Users\juand\Downloads\trabajo final nodejs>
```

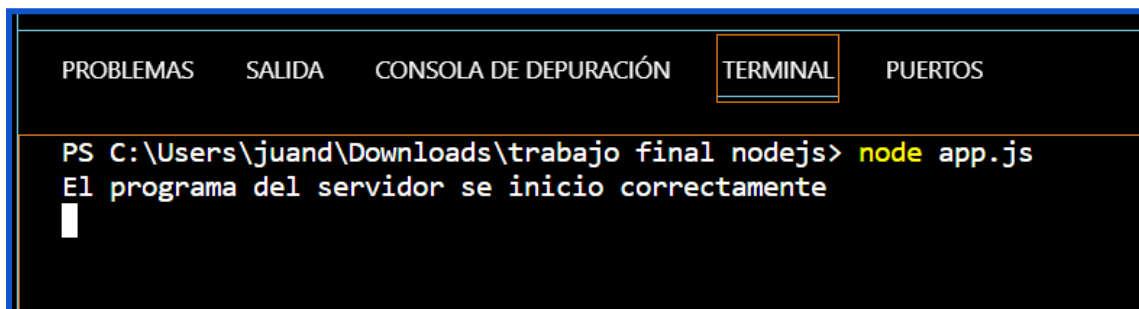
- Para ejecutar el código ingresamos node + nombre de archivo **“node app.js”**.



The screenshot shows the terminal window with the command `node app.js` entered at the prompt:

```
PS C:\Users\juand\Downloads\trabajo final nodejs> node app.js
```

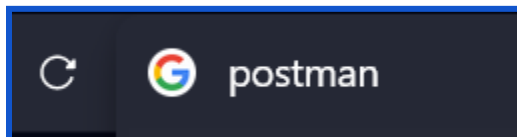
- Al presionar enter se inicia el puerto que imprime el mensaje.



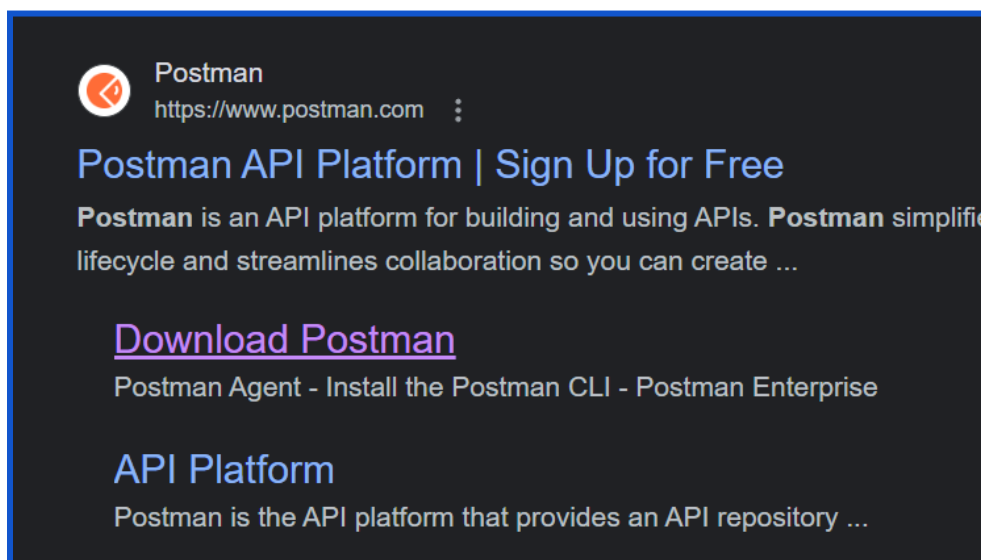
The screenshot shows the terminal window after pressing Enter. The output message is displayed:

```
PS C:\Users\juand\Downloads\trabajo final nodejs> node app.js
El programa del servidor se inicio correctamente
```

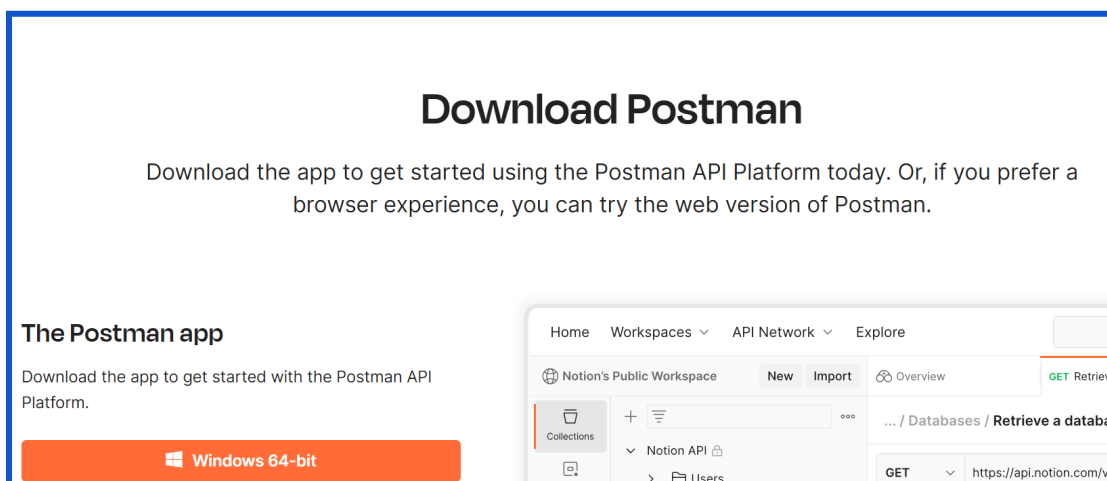
- ❖ **Paso 19:** Para comprobar nuestro código API de la aplicación CRUD esté funcionando correctamente, utilizaremos el software de Postman que nos permitirá realizar solicitudes HTTP y HTTPS y recibir respuestas, para ello ingresamos en el navegador con el siguiente <https://www.postman.com/downloads/> o simplemente buscamos postman.



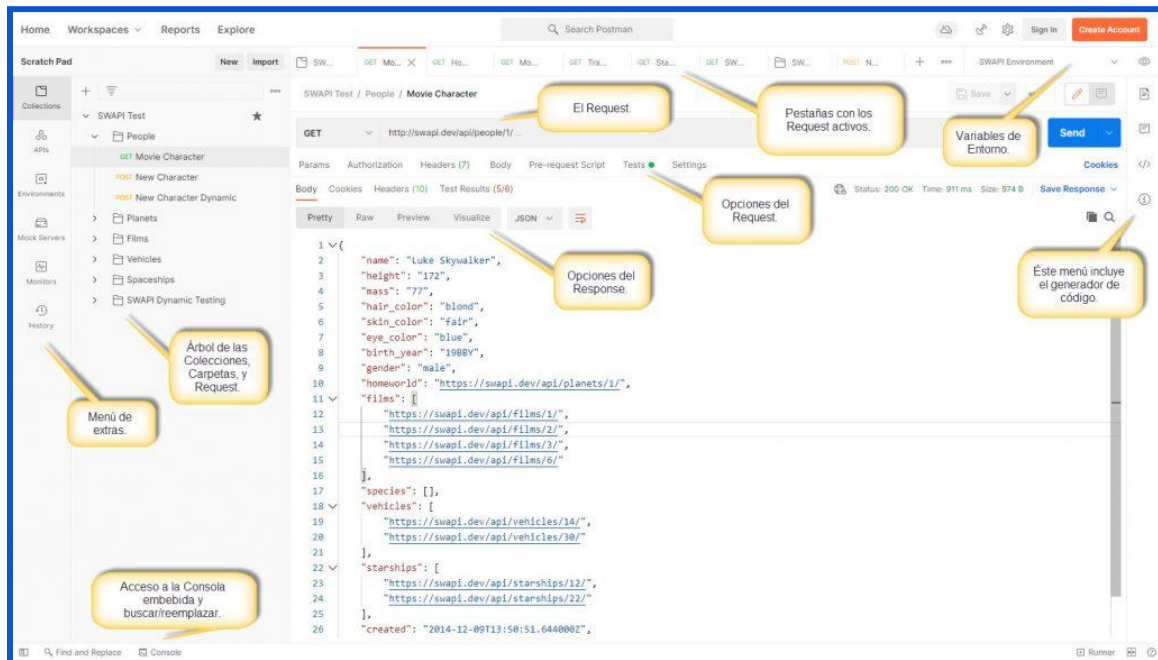
- En la primer url presionamos en download:



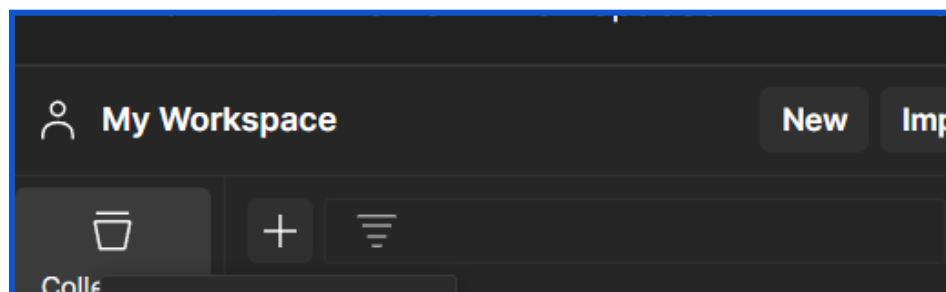
- ❖ **Paso 20:** Presionamos el botón de descargar naranja que se encuentra en la parte inferior izquierda.



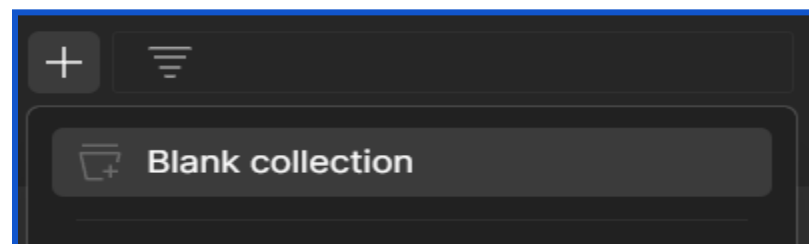
- ❖ **Paso 21:** Después de que la descarga se haya completado damos click directamente o lo buscamos en el explorador de archivos, esperamos a que se instale y cree un acceso directo al escritorio.



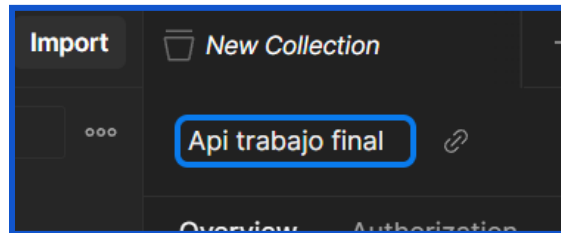
- ❖ **Paso 22:** Abrimos postman y creamos una nueva colección presionando en el **botón +**.



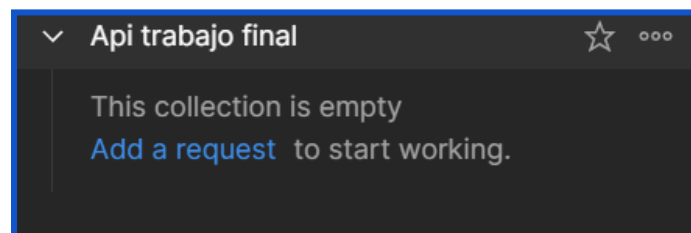
- Se nos despliega un menú y seleccionamos blank collection.



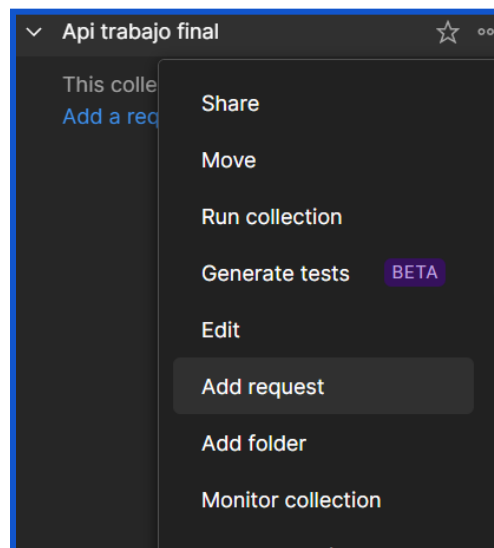
- Cambiamos el nombre que por defecto es “new collection” por API trabajo final.



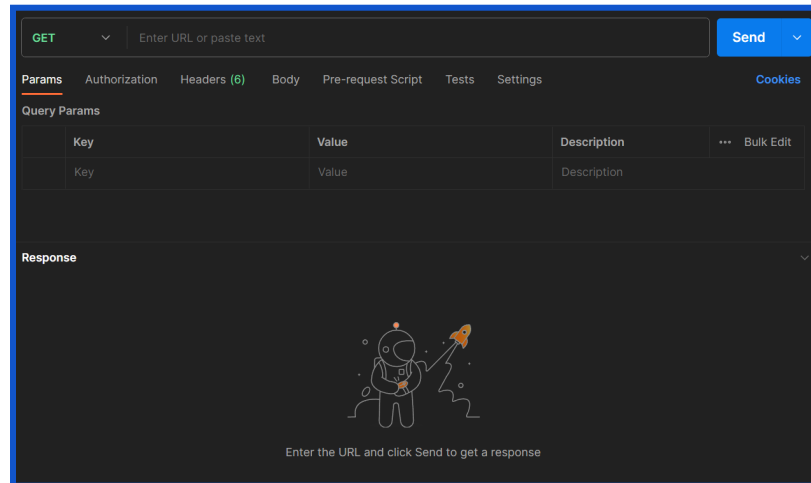
- ❖ **Paso 23:** Presionamos en los tres puntos ubicados junto a la estrella.



- ❖ **Paso 24:** Se despliega un menú y la opción que utilizaremos es Add request esta opción nos ayudará a agregar una nueva solicitud.



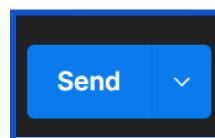
❖ **Paso 25:** Nos aparecerá algo similar a:



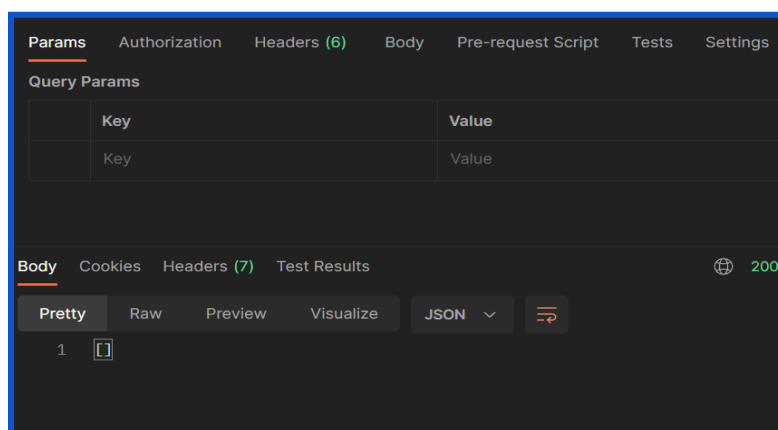
❖ **Paso 26:** Para ver los productos que tenemos en el array ingresamos en el campo URL la url de localhost + la ruta de productos: <http://localhost:3000/productos>

```
// Ruta para obtener todos los productos
app.get('/productos', (req, res) => {
  | res.json(productos); // Devuelve la lista de productos en formato JSON
});
```

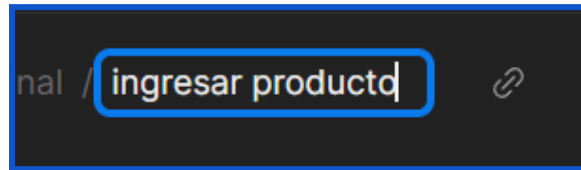
❖ **Paso 27:** Presionamos en el botón send.



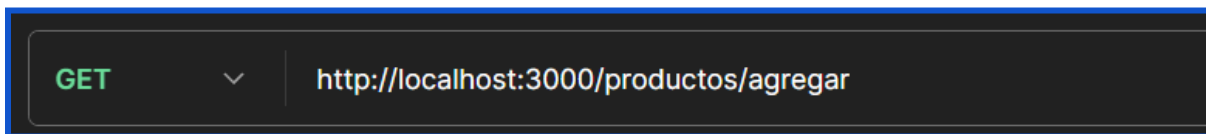
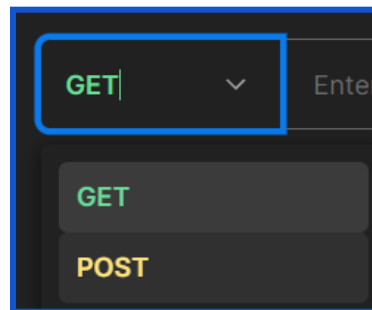
❖ **Paso 28:** Como podemos observar en la imagen nos devuelve [] vacío que pertenece al array ya que no se ha agregado ningún producto.



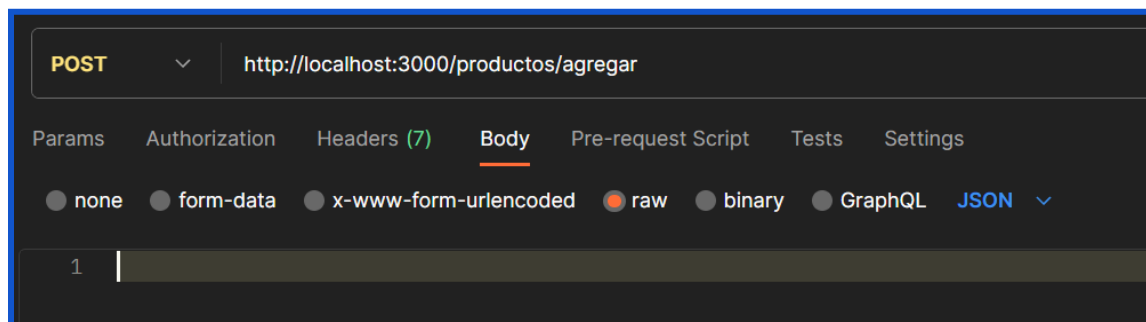
- ❖ **Paso 29:** Crearemos un nuevo producto para ello realizaremos el mismo paso pero presionamos los tres puntos junto la estrella y seleccionamos send creamos un nombre lo llamaremos ingresar producto.



- ❖ **Paso 30:** Cambiamos a get por e ingresamos la url.



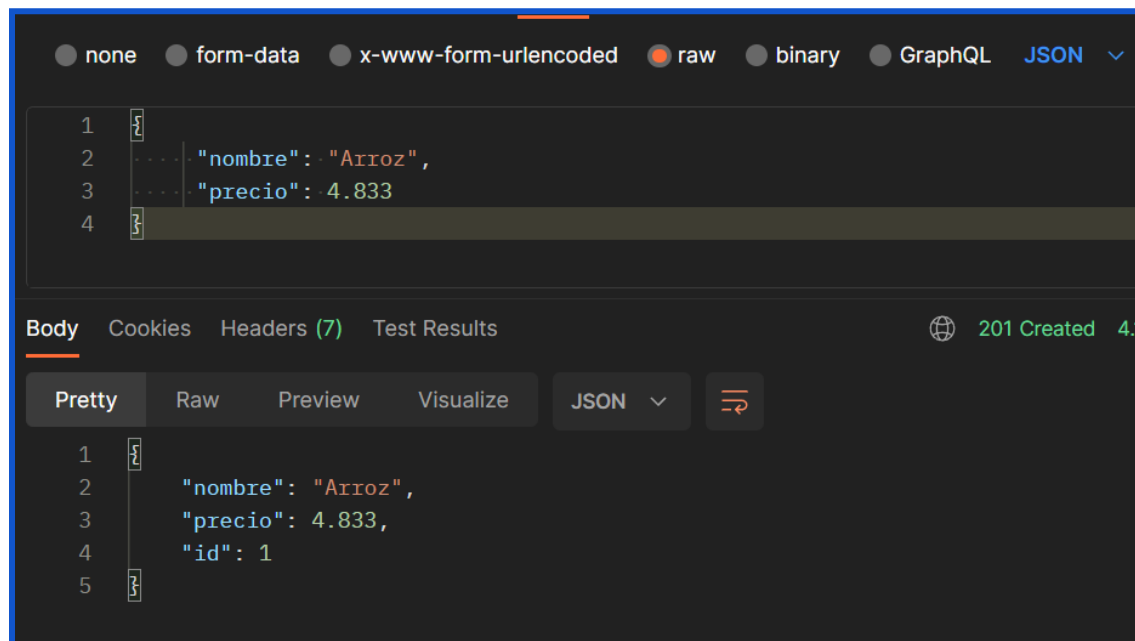
- ❖ **Paso 31:** Marcamos la opción body, la opción raw y cambiamos el tipo de entrada "text" a JSON.



- ❖ **Paso 32:** Ingresamos los datos en formato json y presionamos el botón send.

```
1 {  
2   "nombre": "Arroz",  
3   "precio": 4.833  
4 }
```

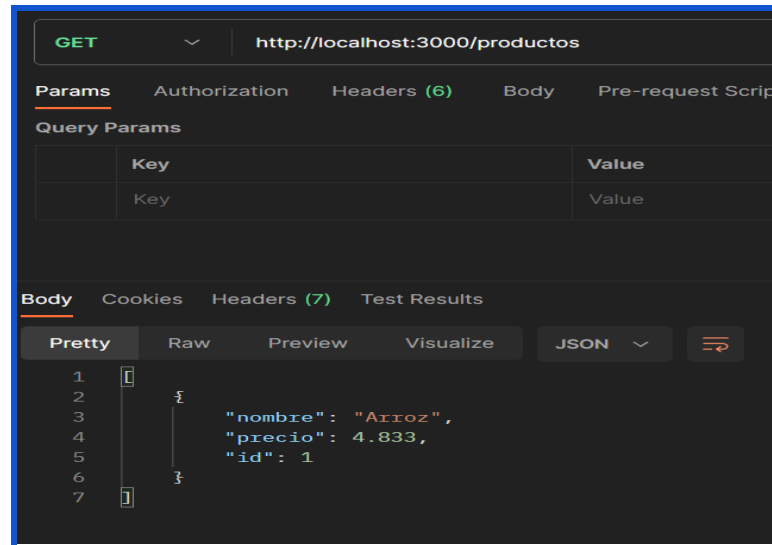
- ❖ **Paso 33:** Podemos ver que nos devuelve el producto que ingresamos más el id asignado.



The screenshot shows a REST client interface with the 'JSON' tab selected. The response body is displayed in a 'Pretty' format, showing a JSON object with three properties: 'nombre' (Arroz), 'precio' (4.833), and 'id' (1). The status bar at the bottom indicates '201 Created'.

```
1 {  
2   "nombre": "Arroz",  
3   "precio": 4.833,  
4   "id": 1  
5 }
```

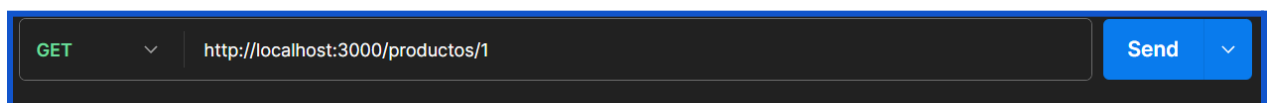
- ❖ **Paso 34:** Si regresamos al archivo que creamos anteriormente “ver productos” y recargamos presionando el botón send y vemos que en el array ya se encuentra un NuevoProducto.



- ❖ **Paso 35:** Para buscar un producto por su id, crearemos otro archivo de la misma forma como creamos los anteriores solo que se cambia la opción GET que es por defecto y se pone PUT que se encuentra en:

```
app.put('/productos/buscar/:id', (req, res) => {  
  const id = req.params.id; // Obtiene el ID de 1
```

- Ingresamos esta misma url productos/buscar/:id quedando de esta manera:

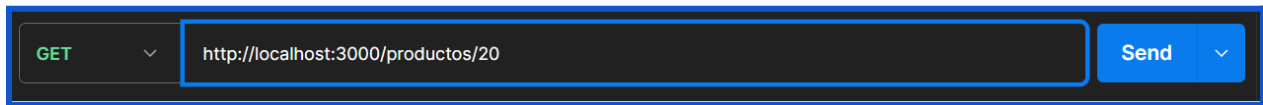


- En el final de la dirección ingresamos el id del producto que queremos que busque utilizamos “1” como id ya que nos buscará el producto “Arroz” presionamos el botón send.

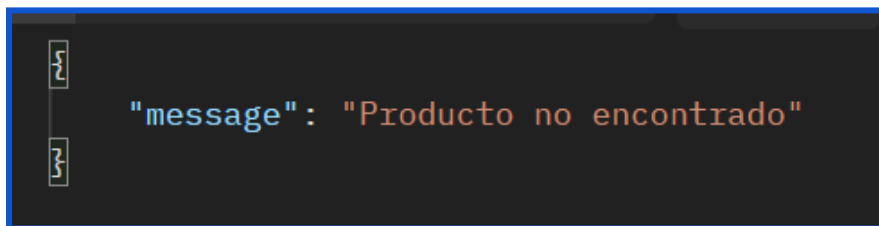
-Lo que nos devuelve :

```
{  
  \"nombre\": \"Arroz\",  
  \"precio\": 4.833,  
  \"id\": 1
```

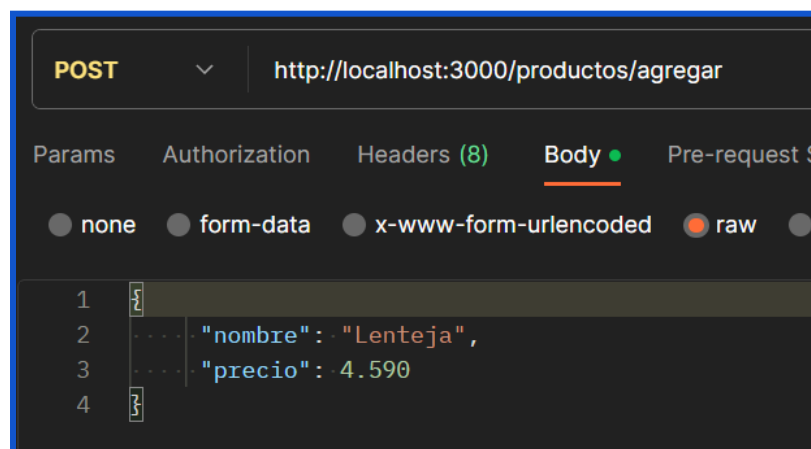
- ❖ **Paso 36:** Ingresado un id de un producto que no se ha creado presionamos el botón send.



- Lo que nos devuelve:



- ❖ **Paso 37:** Para utilizar la ruta para eliminar un producto agregaremos otro “Lenteja” en el archivo ingresar producto que creamos anteriormente.



- Ahora si recargamos en el archivo ver productos vemos que se agregó al array con 2 como id.

```
1  [
2    {
3      "nombre": "Arroz",
4      "precio": 4.833,
5      "id": 1
6    },
7    {
8      "nombre": "Lenteja",
9      "precio": 4.59,
10     "id": 2
11   }
12 ]
```

- ❖ **Paso 38:** Para eliminar el producto id 1 creamos otro archivo lo nombrado “Eliminar producto” cambiamos la opción por defecto GET por DELETE, ingresamos la url de productos + id y presionamos send.

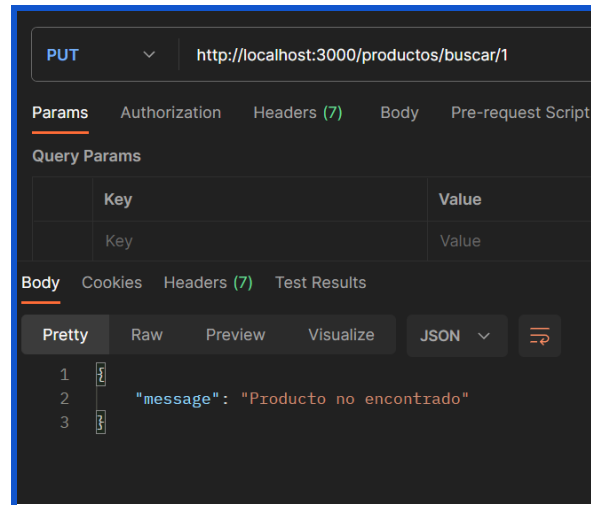
DELETE http://localhost:3000/productos/eliminar/1 Send

```
app.delete('/productos/eliminar/:id', (req, res) => {
  const id = req.params.id; // Obtiene el ID de los parám
  // Encuentra el índice del producto en el array por ID
```

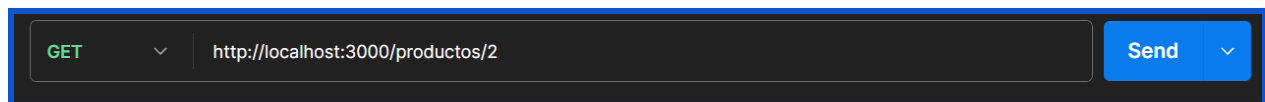
- Nos devuelve las características del producto con el id “1”:

```
1  {
2    "nombre": "Arroz",
3    "precio": 4.833,
4    "id": 1
5  }
```

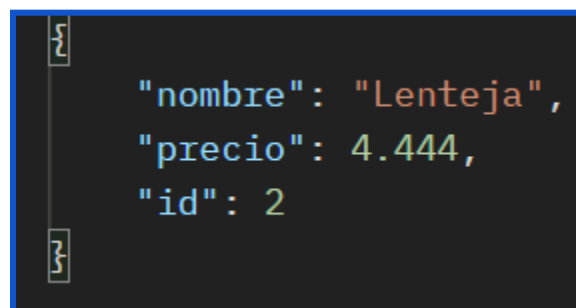
- Si buscamos el id “1” que fue el producto que eliminamos, vemos que no se encuentra.



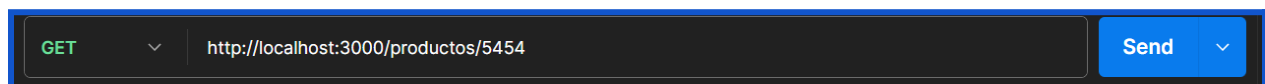
- ❖ **Paso 39:** Utilizamos la ruta actualizar producto por su id creamos un archivo en este caso dejamos la opción GET marcada e ingresamos la url, + el id en este caso “2” actualizaremos el precio 4.833 a 4.444.



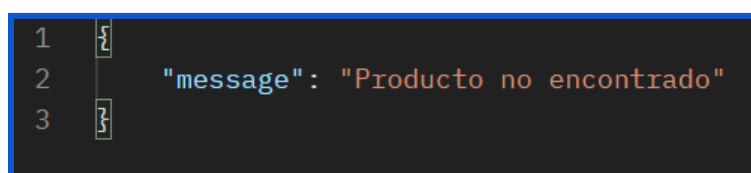
- Lo que nos devuelve:



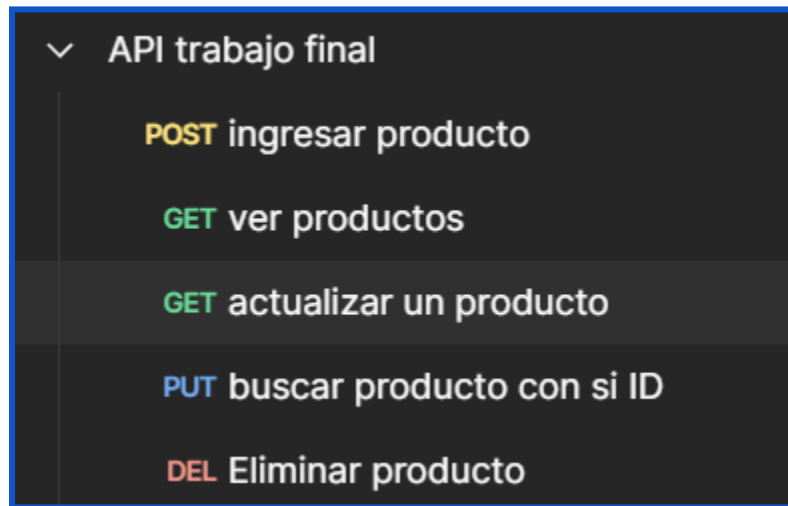
- ❖ **Paso 40:** Intentamos actualizar un producto que no se encuentra en el array.



- Lo que nos devuelve:



-En conclusión creamos en postman una carpeta llamada API trabajo final, que contiene 5 archivos que nos permiten (crear, leer, buscar, actualizar, eliminar datos).



-A lo largo del documento se pudo evidenciar el manejo y uso de distintas herramientas, que gracias a ellas logramos consolidar una muy buena implementación de la aplicación API (CRUD), fue un gran reto porque a pesar de que si habíamos programado ciertas cosas en el lenguaje JS, programar un API está ciertamente a otro nivel, se aprendieron cosas nuevas y muy buenas que serán útiles para futuro cercano, decidimos terminar programando en el code visual, por que nos generaba más facilidad a la hora de programar, quedamos atentos a cualquier inquietud que tenga.