**API en Express con GET y POST**

Después de haber implementado una API de lectura sencilla, con sólo métodos GET, es hora de implementar una manera de agregar más registros a nuestro servidor.

Como es usual, deberemos instalar *Express* si aún no lo tenemos instalado con el comando “***npm install express@4.16.2 --save”*** y esta vez, también agregaremos un *package* llamado *Body Parser* con ***npm install body-parser@1.18.2 --save*** el cual explicaremos más adelante.

Una que tengamos instalado *Express* y *body-parser*, crearemos un archivo llamado “*app.js”* como es habitual. El código completo de la aplicación es el siguiente:

"use strict";

**const** express = require('express');

**const** app = express();

**const** bodyParser = require('body-parser');

app.use(bodyParser.json());

**const** port = 3000;

**const** host = '127.0.0.1';

**const** pendientesPorHacer = [

{id: 1, descripcion: "Comprar comida", completada: true},

{id: 2, descripcion: "Preparar comida", completada: false},

{id: 3, descripcion: "Pasear al perro", completada: true},

{id: 4, descripcion: "Estudiar React y VueJS", completada: false},

{id: 5, descripcion: "Terminar series pendientes", completada: false},

{id: 6, descripcion: "Reparar el celular", completada: true}

]

*//Función para encontrar el siguiente Id.*

**function** obtenerSiguienteId() {

**const** tamañoArray = pendientesPorHacer.length;

**let** idMayor = pendientesPorHacer[0].id || 0;

for (**let** i = 1; i < tamañoArray; i++) {

if(pendientesPorHacer[i].id > idMayor)

idMayor = pendientesPorHacer[i].id;

}

return idMayor + 1;

}

*//Consulta general*

app.get('/', (req, res) **=>** {

return res.status(200).json(pendientesPorHacer);

})

*//Consultar cuando registros hay en el array de pendientesPorHacer.*

.get('/count', (req, res) **=>** {

**const** miRespuesta = {

"numeroDeRegistros": pendientesPorHacer.length

}

return res.status(200).json(miRespuesta);

})

*//Consultar un elemento en el array de pendientesPorHacer por Id.*

.get('/:idPendiente', (req, res) **=>** {

**const** idPendiente = req.params.idPendiente;

**const** pendienteBuscado = pendientesPorHacer.find( pendiente **=>** pendiente.id == idPendiente );

if( pendienteBuscado === undefined )

return res.status(404).json({ mensaje : "No se encontró ese pendiente en el servidor :(" });

else

return res.status(200).json(pendienteBuscado);

})

*//Agregar un nuevo elemento al array.*

.post('/', (req, res) **=>** {

*//Body Parser nos permite obtener los parámetros de las peticiones POST*

*//en el objeto req.body*

**const** nuevoPendiente = {

id: obtenerSiguienteId(),

descripcion: req.body.descripcion,

completada: req.body.completada

}

pendientesPorHacer.push(nuevoPendiente);

**const** miRespuesta = {

registroAgregado: nuevoPendiente,

mensaje: "Registro agregado exitosamente :)"

}

return res.status(200).json(miRespuesta);

})

app.listen(port, host, () **=>** {

console.log(`¡Escuchando en ${host}:${port}/`);

});

Después de hayas analizado el código, seguramente notarás que agregamos lo siguiente:

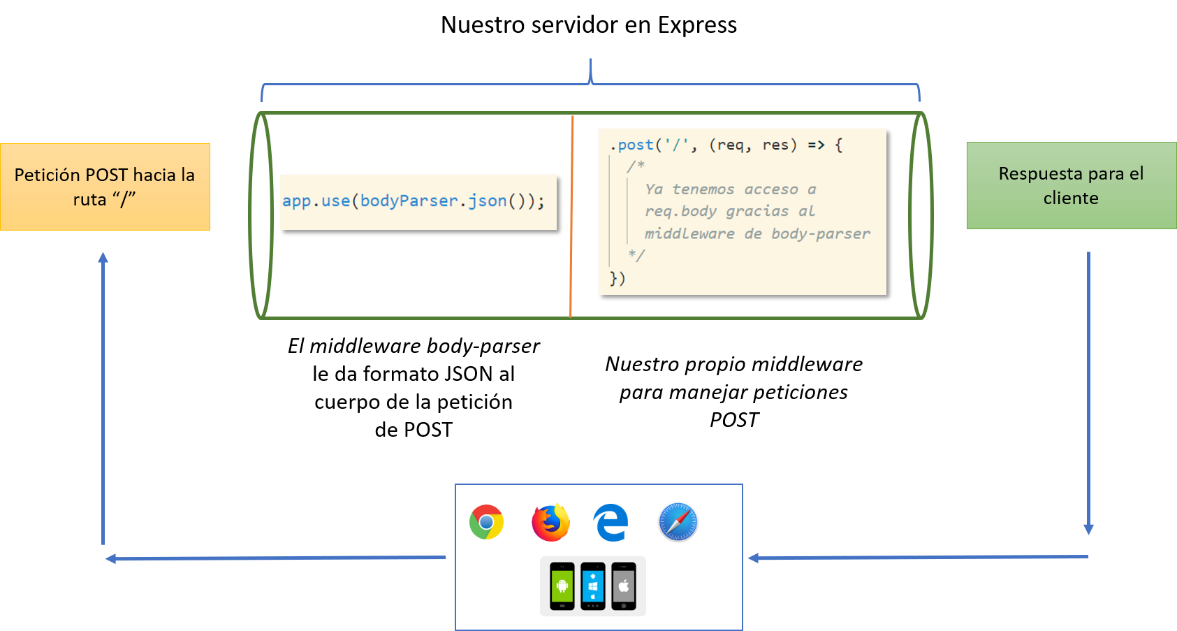
**const** bodyParser = require('body-parser');

app.use(bodyParser.json());

En resumen, ***body-parser*** convierte peticiones (*requests*) que estén entrando al servidor, en un ***middleware****,* antes de que la petición llegue a nuestros métodos, *app.get o app.post* o cualquier otro que establezcamos.

¿Pero qué es un ***middleware***?

Un *middleware* es una función o una capa de funciones que son llamadas por el enrutamiento de *Express* antes de que una petición (*request*) llegué a la función que nosotros hayamos establecido para manejar alguna determinada acción. Para una mejor comprensión de lo anterior, veamos la siguiente imagen.



En la anterior imagen, podemos ver cómo un cliente (por ejemplo, un navegador web) hace una petición (*request*) a nuestro servidor, después *Express* ejecuta el *middleware* de ***body-parser*** con el código “*app.use(bodyParser.json())*”. Después, ***body-parser*** internamente le da formato de tipo JSON a nuestra petición y esto permite que nosotros podamos acceder al cuerpo de la petición mediante ***req.body***en nuestro método “.post('/', (req, res) **=>** { */\* req.body ya está disponible \*/* })”

Después del proceso anterior, podremos manejar la petición de tipo POST como deseemos y responderle al cliente.

Continuando con el código, podemos ver que también se agregó una función llamada *obtenerSiguienteId().*

*//Función para encontrar el siguiente Id.*

**function** obtenerSiguienteId() {

**const** tamañoArray = pendientesPorHacer.length;

**let** idMayor = pendientesPorHacer[0].id || 0;

for (**let** i = 1; i < tamañoArray; i++) {

if(pendientesPorHacer[i].id > idMayor)

idMayor = pendientesPorHacer[i].id;

}

return idMayor + 1;

}

Esta simple función busca en el array de pendientesPorHacer cuál es el *id* con mayor valor numérico, lo guarda en una variable idMayor y posteriormente retorna el valor de idMayor + 1. Esto se hace con el fin de no tener IDs repetidos.

Finalmente, la última parte que le agregamos al código del anterior ejercicio fue la siguiente:

*//Agregar un nuevo elemento al array.*

.post('/', (req, res) **=>** {

*//Body Parser nos permite obtener los parámetros de las peticiones POST*

*//en el objeto req.body*

**const** nuevoPendiente = {

id: obtenerSiguienteId(),

descripcion: req.body.descripcion,

completada: req.body.completada

}

pendientesPorHacer.push(nuevoPendiente);

**const** miRespuesta = {

registroAgregado: nuevoPendiente,

mensaje: "Registro agregado exitosamente :)"

}

return res.status(200).json(miRespuesta);

})

Primero le indicamos a *Express* que todas las peticiones *POST* que ingresen con la ruta '/', sean enviadas a este *endpoint.*

.post('/', (req, res) **=>** {

Después construimos un objeto llamado nuevoPendiente, basándonos en el objeto ***req.body*** que ***body-parser*** no proporcionó con su *middleware.*

**const** nuevoPendiente = {

id: obtenerSiguienteId(),

descripcion: req.body.descripcion,

completada: req.body.completada

}

Obtenemos el siguiente *id* con la función de obtenerSiguienteId(). Y las propiedades de “*descripcion*” *y* “*completada*” la obtendremos del ***req.body,*** es decir, de los datos que nos proporcione la petición (*request*) del cliente.

Después añadimos al array de *pendientesPorHacer*, el nuevo objeto *nuevoPendiente* con la línea de código “*pendientesPorHacer.push(nuevoPendiente);*”.

***Nota:*** A modo de recordatorio, todos los arrays en JavaScript tienen una función o método llamado ***push*** que nos permite añadir nuevos elementos al final del array en cuestión.

Por último, en lo que respecta al código nuevo, creamos un objeto de respuesta:

**const** miRespuesta = {

registroAgregado: nuevoPendiente,

mensaje: "Registro agregado exitosamente :)"

}

return res.status(200).json(miRespuesta);

})

Dicho objeto se llamará miRespuesta, y contendrá el registro que se acabe de agregar y un mensaje que indica que todo se agregó exitosamente.

Finalmente, le responderemos al cliente con un *status* con valor de “*200*” y con el objeto miRespuesta convertido a JSON.

Ahora que ya sabemos cómo funciona todo el nuevo código que agregamos, sólo tendremos que ejecutar el comando “*nodemon app.js*” para que nuestro servidor se ponga a la espera.

Ahora… ¿cómo probamos nuestro nuevo código? Quizá podríamos pensar que con ingresar en nuestro navegador “*127.0.0.1:3000/*” obtendríamos llegar a la ruta “*.post('/', (req, res)* ***=>***{...})”, pero de esa manera sólo se pueden hacer peticiones GET.

Hay varias soluciones:

**HTML:** Con HTML y sus etiquetas *form* podríamos realizar una petición POST.

<form action="/" method="post">

Descripción: <input type="text" name="descripcion"><br>

Completada: <input type="checkbox" name="completada"><br>

<input type="submit" value="Submit">

</form>

**Angular:** Con Angular (específicamente Angular 5), podemos hacer una petición HTTP de tipo POST de la siguiente manera:

public create( pendiente ) {

return http.post('127.0.0.1/', pendiente, {

headers: new HttpHeaders().set('"Content-Type', 'application/json'),

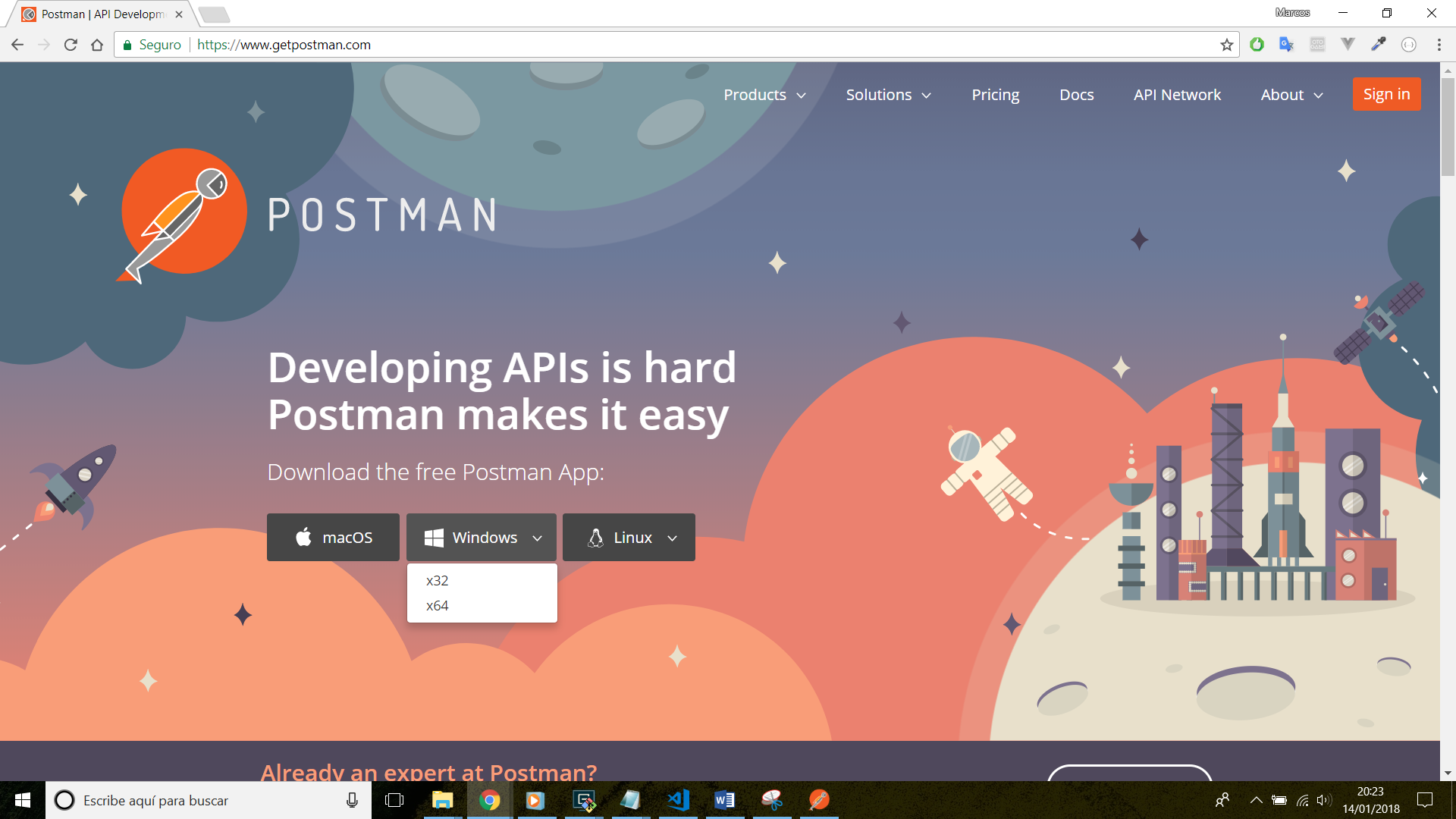
})

}

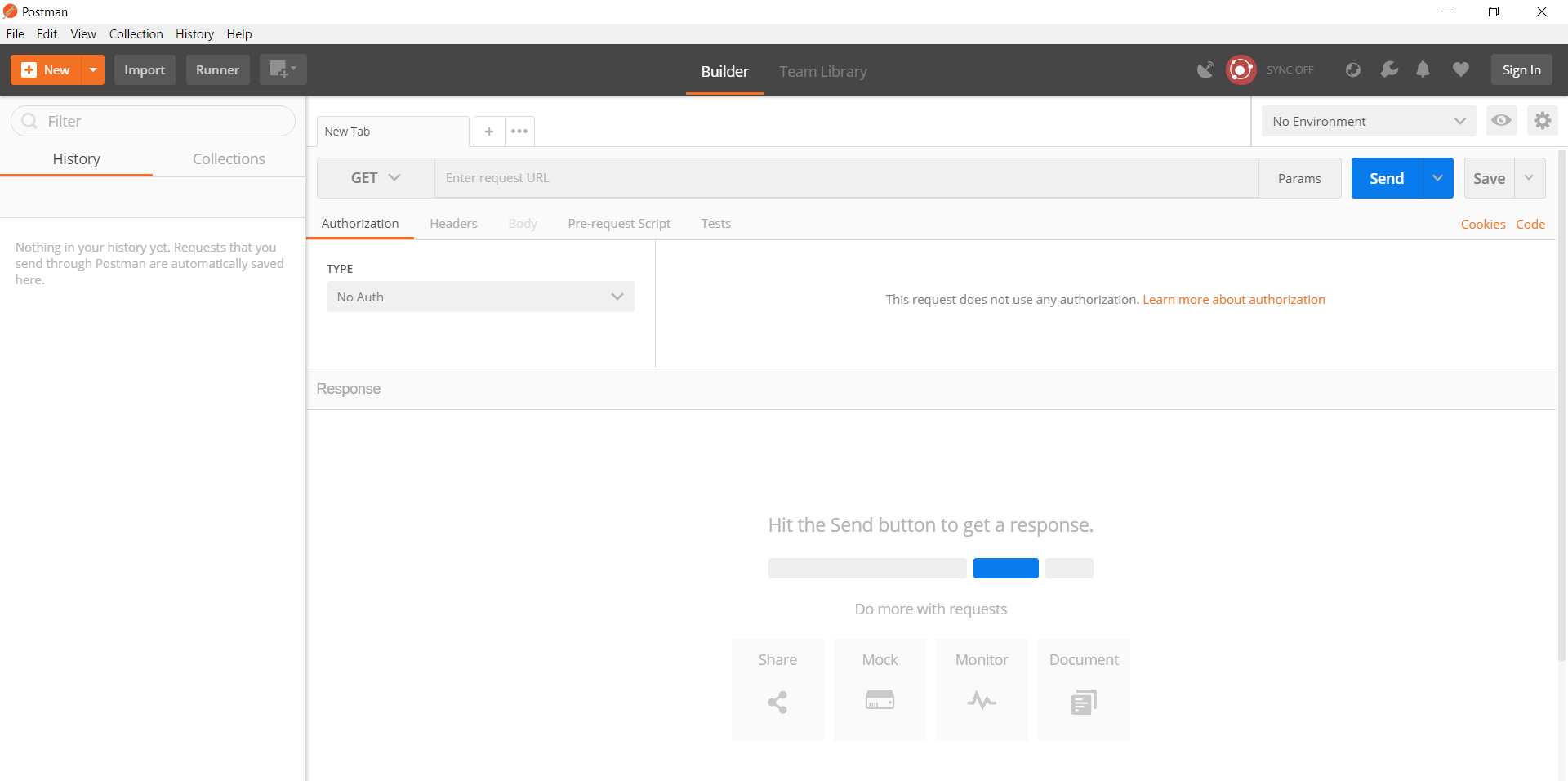
**Nota:** En la sección de Angular de esta misma guía se especificó cómo hacer peticiones HTTP a un servidor. Para más información, consulte esa sección.

**Postman:** Y otra de las opciones para hacer peticiones POST (y también GET, PUT, DELETE, etc.), es Postman. Esta herramienta es una poderosa plataforma con interfaz gráfica para hacer el desarrollo de APIs más rápido y sencillo. Permite hacer peticiones HTTP de todo tipo para poder testear los distintos *endpoint.*

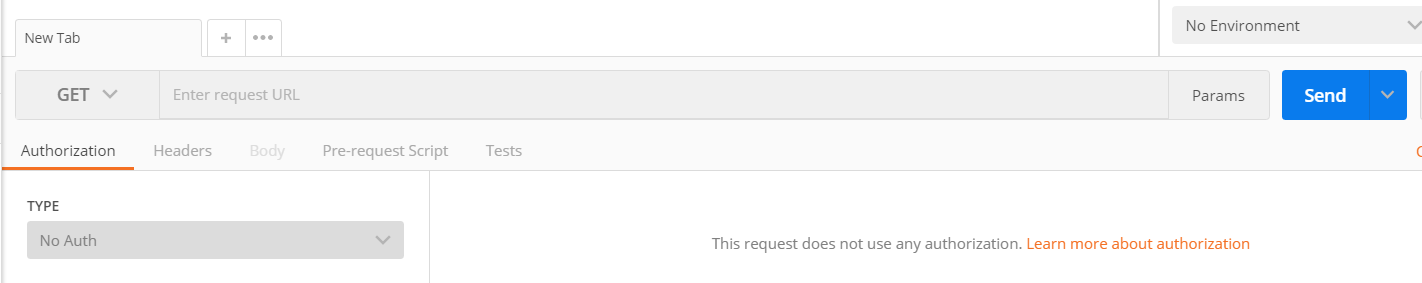
Lo podremos descargas de este link: <https://www.getpostman.com/>



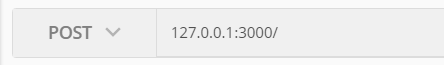
Elegiremos nuestro sistema operativo y la versión adecuada para nuestro procesador. Después comenzará la descarga del instalador. Ejecutaremos le instalador y seguiremos los pasos que nos indique. Una vez que lo tengamos instalado, lo abriremos y nos aparecerá una interfaz similar a la siguiente:



Lo que nos interesa a nosotros es la siguiente sección:



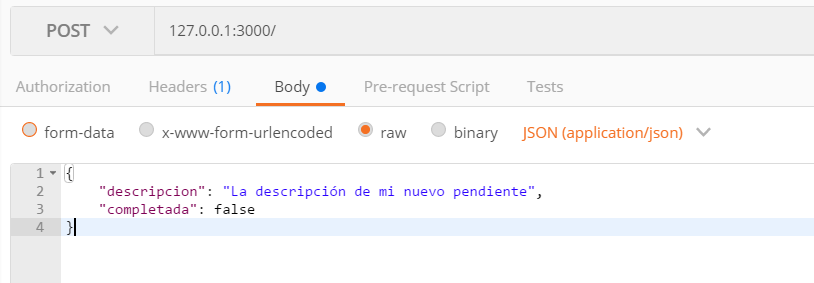
Seleccionaremos el método *POST* e indicaremos que nuestra petición POST sea dirigida a nuestro *enpoint* '/', al poner “*127.0.0.1/*”.



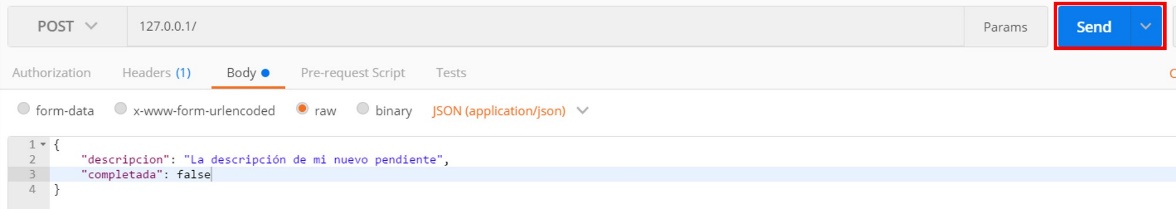
Después agregaremos un Header llamado “*Content-Type”* a la petición POST. El Header de “*Content-Type”* le indicará al servidor, que tipo de contenido tendrá nuestra petición. Y como valor le pondremos “*application/json*”, como lo muestra la siguiente imagen.



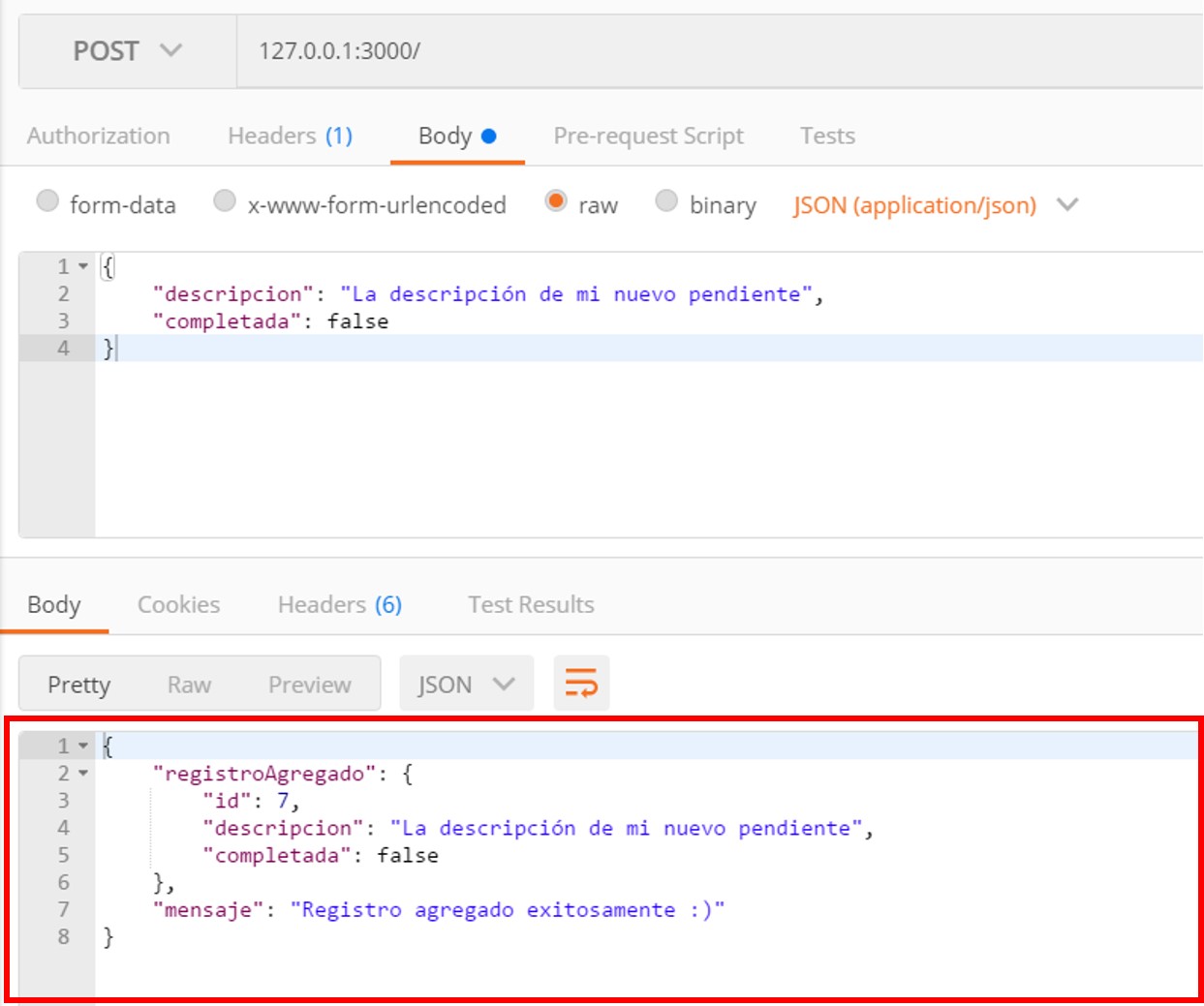
Posteriormente, daremos clic en la pestaña de “*Body*”, seleccionaremos la casilla de “*raw*” y agregaremos un objeto JSON con las propiedades de “*descripcion*” y “*completada*” para coincidir con el código de nuestra función “*.post('/', (req, res)* ***=>***{...})”.



Finalmente daremos clic en el botón de “*SEND*”



Y como nuestro servidor ya está siendo ejecutado, recibirá la petición y nos responderá como le especificamos en el *endpoint* “*.post('/', (req, res)* ***=>***{...})”.

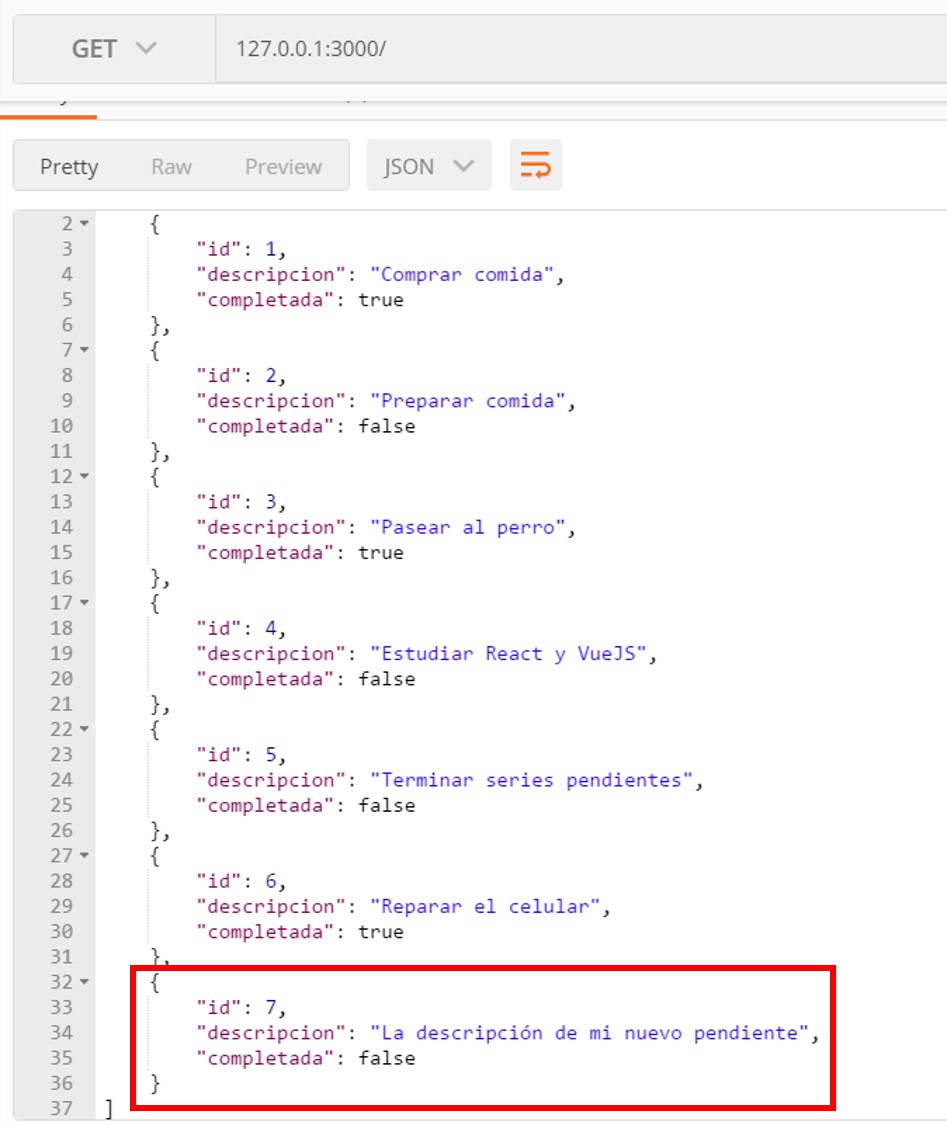


Como podemos ver, *Express* nos respondió con el objeto que le mandamos, justo como le especificamos en nuestro *enpoint* que recibe peticiones POST, es decir, “*.post('/', (req, res)* ***=>***{...})”.

Desde Postman, podremos hacer peticiones GET al igual que lo hacíamos con nuestro navegador web. Solo tenemos que elegir la opción de GET con el mismo con la misma URL (“*127.0.0.1/”*), lo cual será manejado por nuestro *enpoint* app.get('/', (req, res) **=>** {...})



Al dar clic en “*SEND*”*,* nuestro servidor nos responderá con lo siguiente:



Y finalmente, veremos que el nuevo registro ya estará presente en la consulta general.