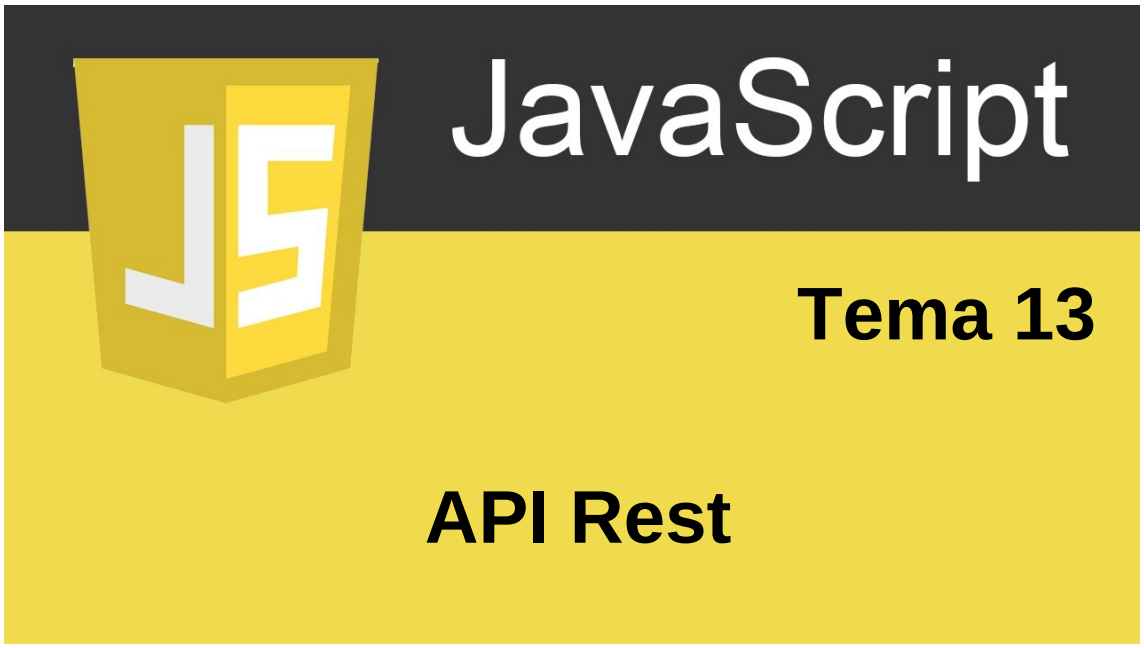
DWC

(Desarrollo Web en entorno cliente)



Índice

1.- ¿Que es una API?

Se conoce como API (del inglés: «Application Programming Interface»), o Interfaz de programación de Aplicaciones al conjunto de rutinas, funciones y procedimientos (métodos) que permite utilizar recursos de un software por otro, sirviendo como una capa de abstracción o intermediario.

Podemos describir un API, según David Berlind, en su artículo: What is an API, Exactly? como un método generalizado de consumir un servicio. Berlind nos presenta la analogía del conector eléctrico en las paredes de nuestro hogar u oficina. Este conector eléctrico que conocemos permite conectar equipos eléctricos que realizan distintas funciones. Este conector eléctrico es un API. El mismo permite consumir un servicio: electricidad sin importar que equipo lo consuma.

En programación y desarrollo de software o aplicaciones, esta analogía describe perfectamente una API REST. Creamos un API con el objetivo de que cualquier software pueda «consumir» nuestros recursos, por lo general datos, sin importar en que lenguaje o plataforma en que sea creado.

API REST es el sucesor de métodos anteriores como **SOAP** y **WSDL** cuya implementación y uso son un poco más complejos y requieren mayores recursos y especificaciones al ser usados.

2.- ¿Cómo funciona una API?

Una REST API, como describimos hace un momento es un servicio. Y si seguimos la analogía del conector eléctrico, entonces fácilmente identificamos que funciona como un estándar para compartir información, en un sistema de doble vía: Consulta y Respuesta (**Request -> Response**).

Al consultar una API se deben especificar parámetros de consulta para que el servicio sepa lo que queremos consultar, basado en una estructura previamente definida provista por el API por medio de una documentación.

3.- La arquitectura REST

La arquitectura REST (del inglés: Representational State Transfer) trabaja sobre **el protocolo HTTP**. Por consiguiente, los procedimientos o métodos de comunicación son los mismos que HTTP, siendo los principales: **GET, POST, PUT, PATCH y DELETE**.

Otros métodos que se utilizan en REST API son **OPTIONS** y **HEAD**. Este último se emplea para pasar parámetros de validación, autorización y tipo de procesamiento, entre otras funciones.

Otro componente de un REST API es el **«HTTP Status Code»**, que le informa al cliente o consumidor del API que debe hacer con la respuesta recibida. Estos son una referencia universal de resultado, es decir, al momento de diseñar un RESTful API toma en cuenta utilizar el «Status Code» de forma correcta.

Por ejemplo, el código 200 significa OK, que la consulta ha recibido respuesta desde el servidor o proveedor del API.

Los códigos de estado más utilizados son:

- 200 OK
- 201 Created (Creado)
- 304 Not Modified (No modificado)
- 400 Bad Request (Error de consulta)
- 401 Unauthorized (No autorizado)
- 403 Forbidden (Prohibido)
- 404 Not Found (No encontrado)
- 422 (Unprocessable Entity (Entidad no procesable)
- 500 Internal Server Error (Error Interno de Servidor)

Una respuesta de un RESTFul API sería lo siguiente:

```
Status Code: 200 OK
Access-Control-Allow-Methods: PUT, GET, POST, DELETE, OPTIONS
Connection: Keep-Alive
Content-Length: 186
Content-Type: application/json
```

```
Date: Mon, 24 May 2016 15:15:24 GMT
Keep-Alive: timeout=5, max=100
Server: Apache/2.4.9 (Win64) PHP/5.5.12
X-Powered-By: PHP/5.5.12
access-control-allow-credentials: true
```

Por lo general y mejor práctica, el cuerpo (Body) de la respuesta de un API es **una estructura en formato jSON**. Aunque también puede ser una estructura XML o cualquier otra estructura de datos de intercambio, incluso una personalizada. Sin embargo, como el objetivo es permitir que cualquier cliente pueda consumir el servicio de un API, lo ideal es mantener una estructura estándar**, por lo que jSON es la mejor opción**.

```
{
  "error": false,
  "message": "Autos cargados: 2",
  "autos": [
              {
                "make": "Toyota",
                "model": "Corolla",
               },
              {
                "make": "Nissan",
                "model": "Sentra",
              }
            ]
}
```

4.- Llamadas a la API REST

Las llamadas a una API REST generalmente se realizan para realizar un **CRUD.** En informática, **CRUD** es el acrónimo de "Crear, Leer, Actualizar y Borrar" (del original en inglés: Create, Read, Update and Delete), que se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.

Cada llamada o petición REST se implementan como peticiones HTTP, en las que:

- La URL representa el recurso http://www.server.com/api/cursos
- El método (HTTP Verbs) representa la operación
- El código de estado HTTP representa el resultado de la petición.

4.1.- Crear (POST)

La URL estará “abierta” (el recurso todavía no existe y por tanto no tiene id) para poder indicarlo en la URL

El método debe ser **POST**

Se deberá enviar en la petición el recurso que queremos añadir

4.2.- Leer (GET)

Podemos realizar lecturas de todos los registros de una entidad o únicamente un registro de la entidad. Si no indicamos el id en la URL nos devolverá todos los registros de dicha entidad, si especificamos el id nos devolverá únicamente el registro con ese id.

El método debe ser **GET**

4.3.- Actualizar (PUT y PATCH)

Para realizar una actualización hay que especificar en la URL la entidad y el id del elemento a modificar. Además, deberemos incluir en la petición el objeto con los valores que queremos actualizar.

Hay que tener cuidado con la estructura de objeto que mandamos (los campos del objeto), pues tenemos dos métodos para la actualización y los resultados son diferentes por la forma en que actúan (uso de la idempotencia)

Los Métodos pueden ser:

- **PUT** según la ortodoxia REST, actualizar significaría cambiar TODOS los datos
- **PATCH** es un nuevo método estándar HTTP pensado para cambiar solo ciertos datos. Algunos frameworks de programación REST no lo soportan

4.4.-Eliminar (DELETE)

Para realizar una actualización hay que especificar en la URL la entidad y el id del elemento a eliminar.

El Método debe ser **DELETE**

Tras ejecutar el DELETE con éxito, las siguientes peticiones GET a la URL del recurso deberían devolver 404

5.- Ventajas que ofrece REST para el desarrollo

REST se ha convertido probablemente en la tecnología más utilizada hoy en día para el desarrollo de aplicaciones Web Cliente/Servidor en las que haya que solicitar o mandar al servidor información para ser almacenada en una BD.

- **Separación entre el cliente y el servidor**: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

- **Visibilidad, fiabilidad y escalabilidad**: la separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

- **La API REST siempre es independiente del tipo de plataformas o lenguajes**: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

6.- JSON Server

Para poder comprobar cómo funciona una API REST lo primero que debemos tener es la API REST.

La API REST se implementa en el servidor (**backend**), nosotros estamos actuando de cliente (**front-end**). No vamos a diseñar una API REST en el servidor porque excede nuestros contenidos, no sería muy complicado ya que hoy en día casi todos los lenguajes de servidor llevan incorporadas funciones para poder realizar esta tarea.

Para poder trabajar en clase vamos a utilizar una herramienta llamada **JSON Sever** que nos permitirá implementar una API REST en nuestro equipo y que utilizará como BD un fichero en formato JSON en el que se almacenará la información de las entidades y registros.

Esta herramienta permite construir APIS REST de una manera muy sencilla y rápida, de hecho, es utilizada durante el proceso de desarrollo para poder testear en modo local las peticiones REST y de esta manera evitar las esperas innecesarias debidas a peticiones de servidores externos. Incluso se utiliza para poder empezar a desarrollar desde el front-end peticiones a una API REST que aún no ha sido implementada, ya que sí que sabemos las peticiones que vamos a realizar al servidor y que métodos y objetos debemos incluir en ellas.

Una vez finalizado el proceso de desarrollo y pasado a producción lo único que se debería de hacer es cambiar las URLs locales a las URLs del servidor externo real donde se haya diseñado la API REST.

JSON Server es un paquete npm que permite crear un servicio web REST JSON, respaldado por una base de datos simple. Está creado para desarrolladores **front-end** y ayuda a realizar todas las operaciones CRUD sin un prototipo o estructura de **backend** determinado.

Vamos a ver lo básico de la implementación de JSON Server así como de su funcionamiento para poder comprobar todo lo visto acerca de una API REST.

6.1.- Instalar JSON Server

Para instalar el JSON Server deberemos instalar el paquete correspondiente con npm. Para ello deberemos tener instalado **Node.js** en nuestro equipo

```
npm install -g json-server
```

JSON Server trabaja con una **base de datos no relacional** muy sencilla, para ello lo que se utiliza es un fichero en formato JSON. Por defecto se crea un archivo llamado **db.json** con algunos datos

```
{
  " publicaciones " : [
   { " id " : 1 , " title " : " json-server " , " author " : " typicode " }
 ],
 " comentarios " : [
   { " id " : 1 , " body " : " algún comentario " , " postId " : 1 }
 ],
 " profile " : { " name " : " typicode " }
}
```

Como podemos ver lo que se crea es un objeto JSON en el que cada propiedad es el nombre de una entidad (tabla del modelo relacional) y su value es un array con los datos de esa entidad (registros del modelo relacional), cada elemento tendrá la estructura que permita almacenar los datos deseados en el par name:value de JSON (campos del modelo relacional).

6.2.- Iniciar JSON Server

Para iniciar el servidor deberemos ejecutar en la consola el comando json-server - -watch con el nombre de la BD que queramos utilizar

```
json-server --watch db.json
```

En ese momento el servidor estará disponible en la url **http: // localhost: 3000**. Para poder probarla lo único que deberemos hacer es realizar las peticiones REST adecuadas.

Si realizamos la siguiente petición http: // localhost: 3000 / posts / 1 obtendremos el siguiente resultado

```
{ " id " : 1 , " title " : " json-server " , " author " : " typicode " }
```

JSON Server implementa las siguientes características

- Todas las entidades deben tener definido un id en el fichero json
- Si realizamos POST, PUT, PATCH o DELETE, los cambios serán automáticos y se guardaran en db.json.
- El body de las peticiones debe ser un objeto JSON, al igual que la devolución de una petición GET.
- Los valores de identificación no son mutables. Cualquier valor id en el cuerpo de la solicitud PUT o PATCH será ignorado.Solo se respetará un valor establecido en una solicitud POST.
- Una solicitud POST, PUT o PATCH debe incluir un encabezado **Content-Type: application/json** para usar el JSON en el cuerpo de la solicitud. De lo contrario, devolverá un código de estado 2XX, pero sin que se realicen cambios en los datos.

6.3.- Rutas

Según el archivo db.json que se crea por defecto, aquí están todas las rutas predeterminadas.

Rutas plurales

```
GET    /posts
GET    /posts/1
POST   /posts
PUT    /posts/1
PATCH  /posts/1
DELETE /posts/1
```

Rutas singulares

```
GET    /profile
POST   /profile
PUT    /profile
```

```
PATCH  /profile
```

6.4.- Los filtros

Los filtros están pensados para hacer peticiones GET a la API REST en las que la consulta no está basada en la id de la entidad, sino que está basada en uno varios campos de la entidad

Para acceder a propiedades profundas

```
GET /posts?title=json-server&author=typicode
GET /posts?id=1&id=2
GET /comments?author.name=typicode
```

7.- Clientes API Rest

Un Cliente REST (Representational State Transfer) o testadores de API son herramientas que nos permite construir y enviar peticiones REST para testear una comunicación entre cliente y servidor.

Sirven para ayudar a los desarrolladores para desarrollar y probar las API de servicios web REST con todos los métodos admitidos, como GET, POST, PUT, PATCH, DELETE.

Estos clientes son compatibles con la autenticación básica HTTP, soportan múltiples cabeceras y el formato de respuesta.

Además, aceptan application/json por defecto con lo cual las peticiones mandan y reciben objetos json.

Los más representativos son:

- Postman
- RestClient
- Advanced Rest Client

Realmente estos clientes de API REST son complementos o extensiones que se añaden al navegador.

Desde el navegador sólo podemos comprobar las peticiones REST que no necesitan que se le envíen datos, con estos complementos ya podemos enviar datos al servidor y comprobar todos los tipos de peticiones.

7.1.- Advanced Rest Client (ARC)

Como hemos visto ARC es un complemento que se añade a nuestro navegador y está disponible como una aplicación más.

Admite los métodos vistos anteriormente para poder realizar un CRUD utilizando la tecnología REST.

Es muy sencillo e intuitivo de usar:

1. seleccionamos el método a utilizar del desplegable
2. introducimos la url
3. si realizamos una petición que debe mandar datos al servidor debemos especificar el tipo de datos en el body content type y en el body introducir el objeto JSON a mandar
4. enviamos la petición

8.- Funcionamiento API REST JSON Server y ARC

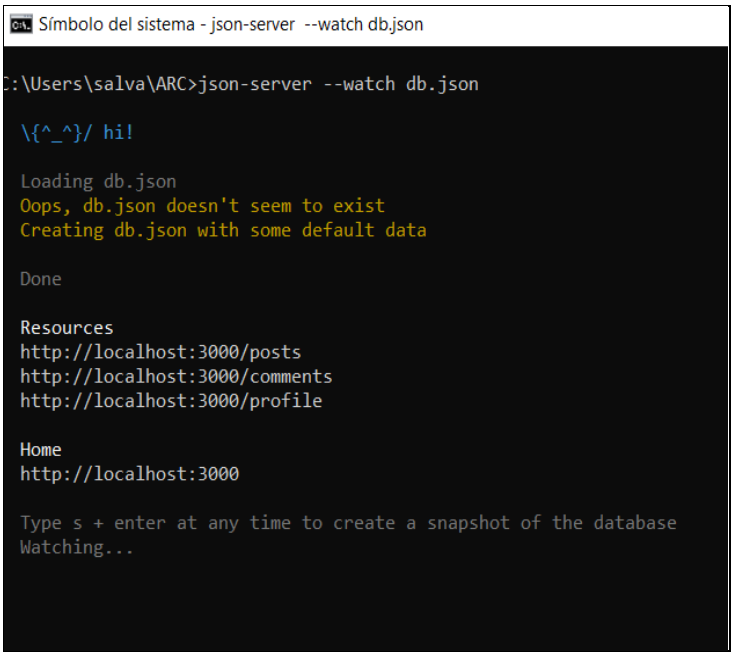
Vamos a probar todos los métodos disponibles de una API REST en JSON Server utilizando el db.json que lleva por defecto realizando las operaciones sobre la entidad comments

8.1.- Inicio de la API REST

Iniciamos JSON Server como vimos anteriormente

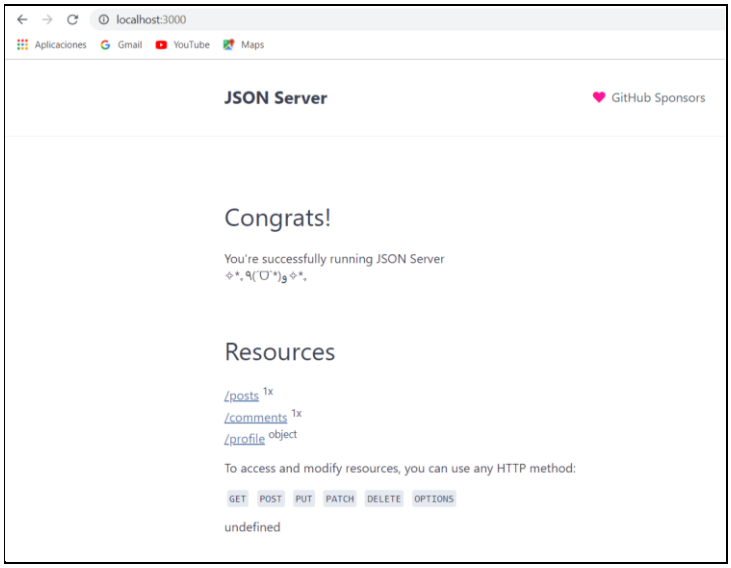
```
json-server –watch db.json
```

Nos aparecerá un mensaje indicando que ya está disponible la base de datos db.json. Nos mostrará los recursos disponibles en esa BD y nos dira que ya podemos realizar las operaciones en http://www.localhost:3000



La consola se queda en modo watch esperando recibir peticiones.

Si abrimos el navegador y ponemos la url http://www.localhost:3000 nos parecerá un mensaje de inicio indicando que todo ha ido bien y enlaces a las entidades del fichero db.json



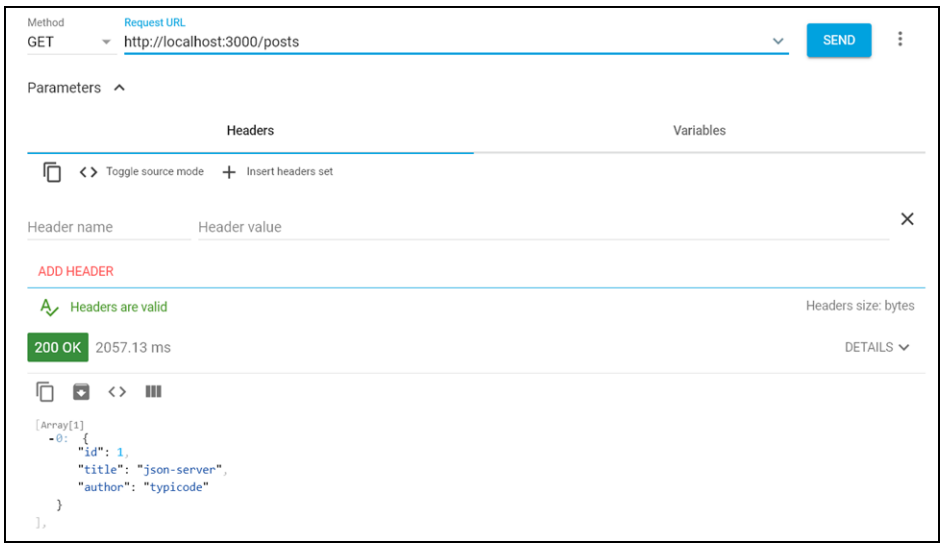
Si en el navegador ponemos la url http://www.localhost:3000/posts nos mostrará el contenido de la entidad posts (inicialmente un solo registro)

Si en el navegador ponemos la url http://www.localhost:3000/posts nos mostrará el registro de posts cuyo id es 1

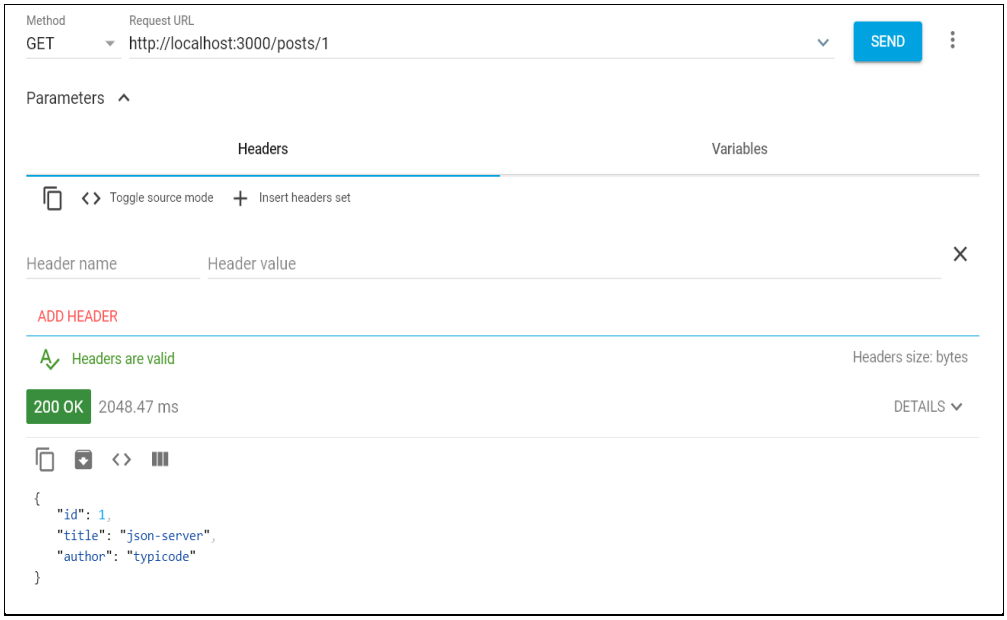
Con el navegador solo hemos podido probar el método GET, para poder probar el resto de métodos ya debemos utilizar un cliente REST

8.2. Petición GET

Vamos a realizar un GET a la url http://www.localhost:3000/posts para ello seleccionamos el método GET y ponemos la url.



Nos devuelve un listado con todos los registros de post (inicialmente sólo hay uno)



Además, nos devuelve el código http 200 indicando que todo ha ido bien.

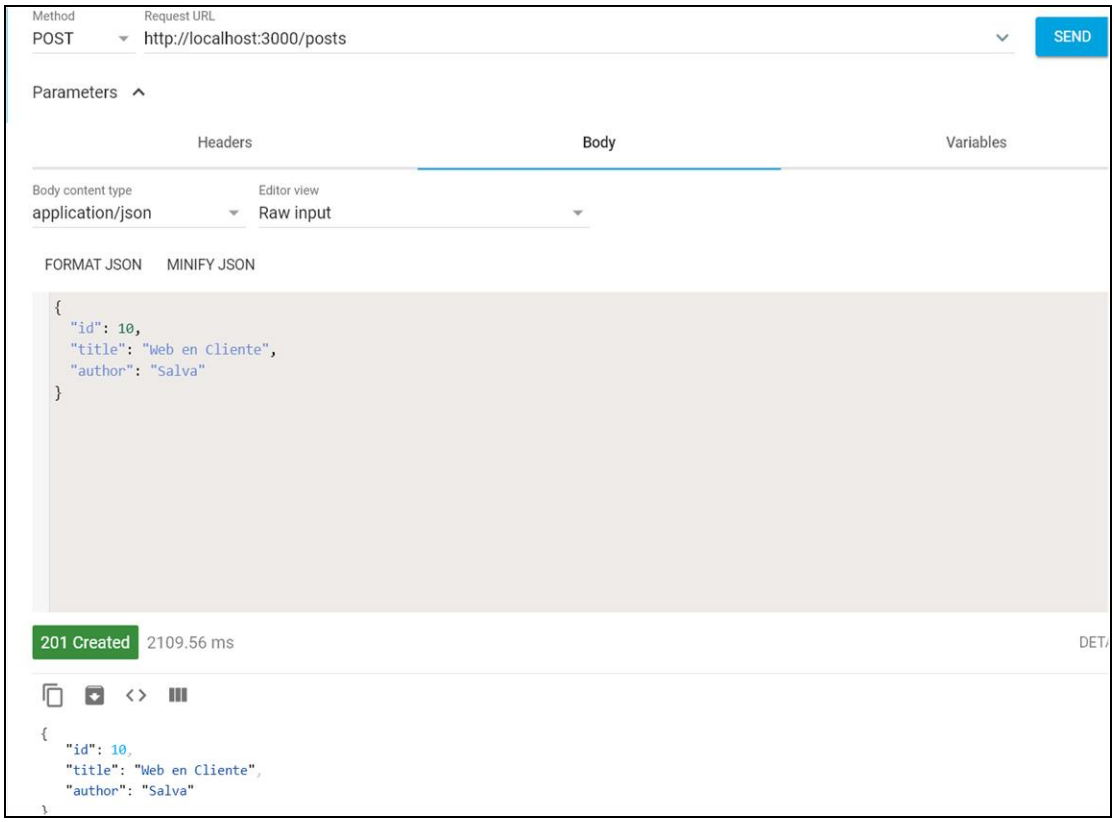
8.3. Petición POST

Vamos a añadir un registro nuevo en la entidad posts, para ello deberemos utilizar el método POST.

Además de seleccionar el método y de poner la url, en este caso deberemos indicar que vamos a mandar un objeto json en el body content type y en el body deberemos indicar el registro que queremos mandar al servidor para que sea añadido.

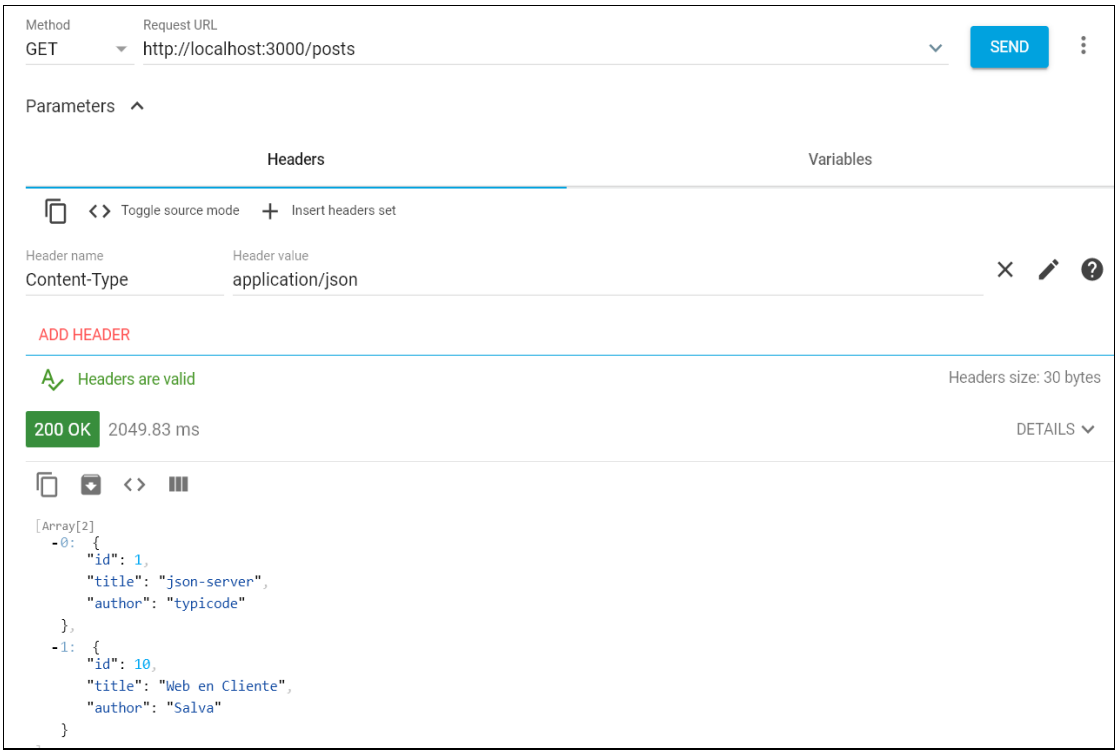
Vamos a mandar el siguiente json

```
{
  "id": 10,
  "title": "Web en Cliente",
  "author": "Salva"
}
```



La respuesta del servidor es 201 indicando que la inserción se ha realizado correctamente y además nos devuelve el registro insertado

Si ahora hacemos un GET nos debería de salir el nuevo registro dentro de posts



8.3. Petición PUT

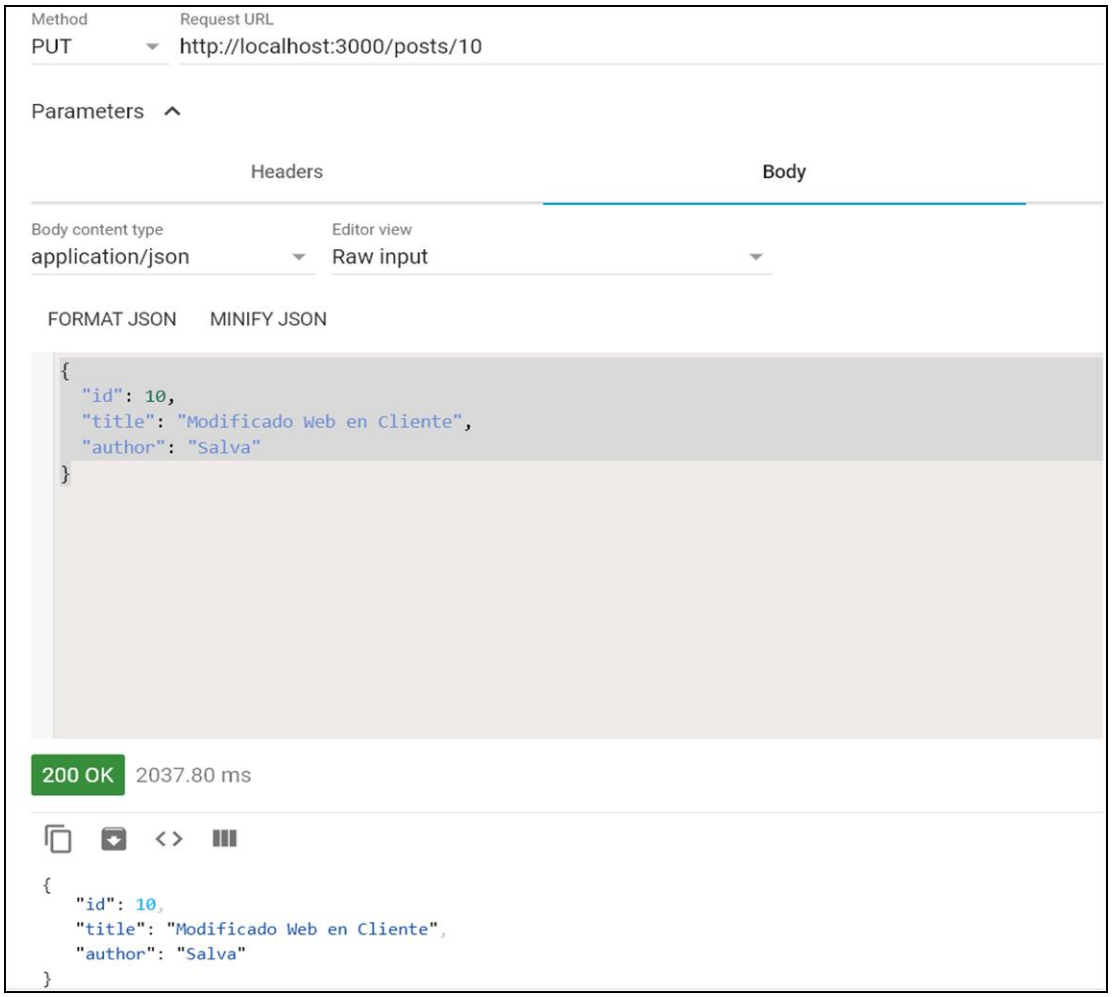
Vamos a modificar un registro existente en la entidad posts, para ello deberemos utilizar el método PUT.

Vamos a modificar los datos del registro que hemos insertado anteriormente con el id 10

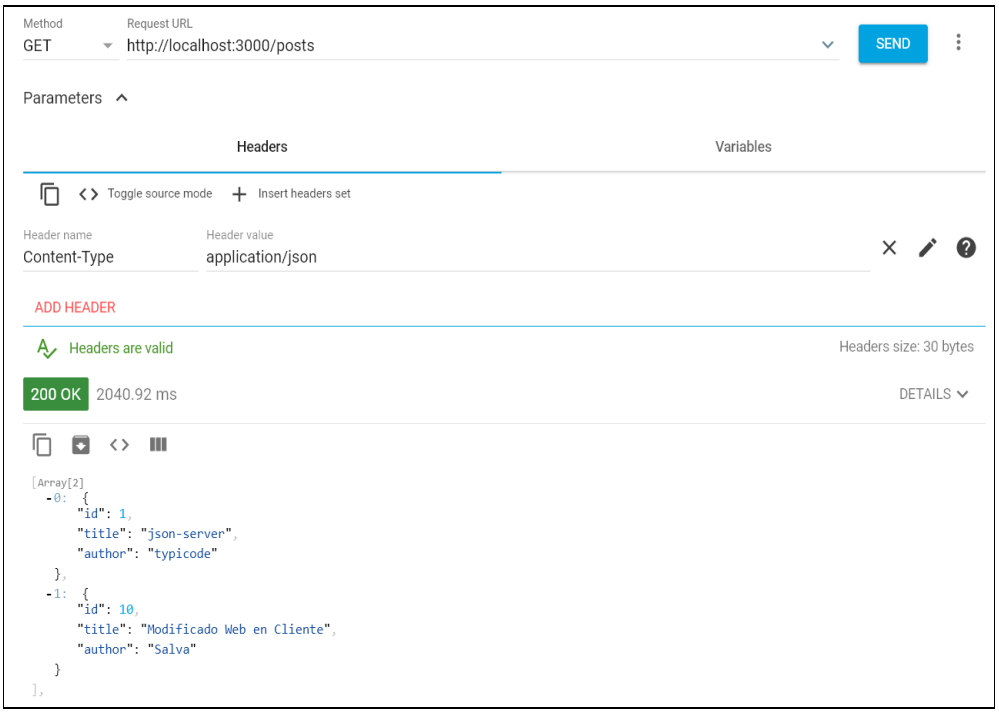
Además de seleccionar el método y de poner la url, en este caso deberemos indicar que vamos a mandar un objeto json en el body content type y en el body deberemos indicar las modificaciones que queremos realizar

Vamos a mandar el siguiente json

```
{
  "id": 10,
  "title": "Modificado Web en Cliente",
  "author": "Salva"
}
```



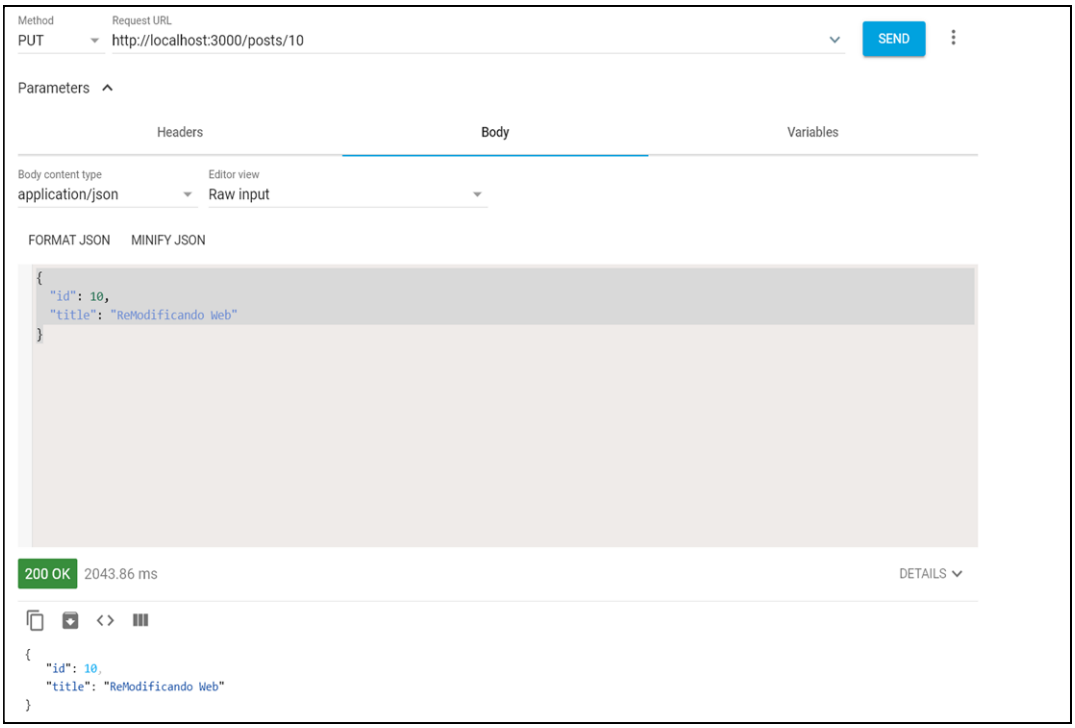
En este caso nos contesta con un código HTTP 200 que nos indica que todo ha ido bien. Además, nos devuelve el registro modificado.
Vamos a realizar una petición GET para comprobar que el nuevo registro existe.



Hay que tener mucho cuidado con lo que se manda en una petición PUT, es decir la **estructura del objeto** que mandamos. En el ejemplo anterior hemos mandado la estructura completa y ha hecho los cambios en todos los campos.
Si hubiéramos mandado únicamente el campo que queríamos modificar hubiera reemplazo el registro original por el nuevo y el resultado en la BD hubiera sido un registro que sólo tendría los datos de los campos mandados en el objeto del body de la petición PUT.

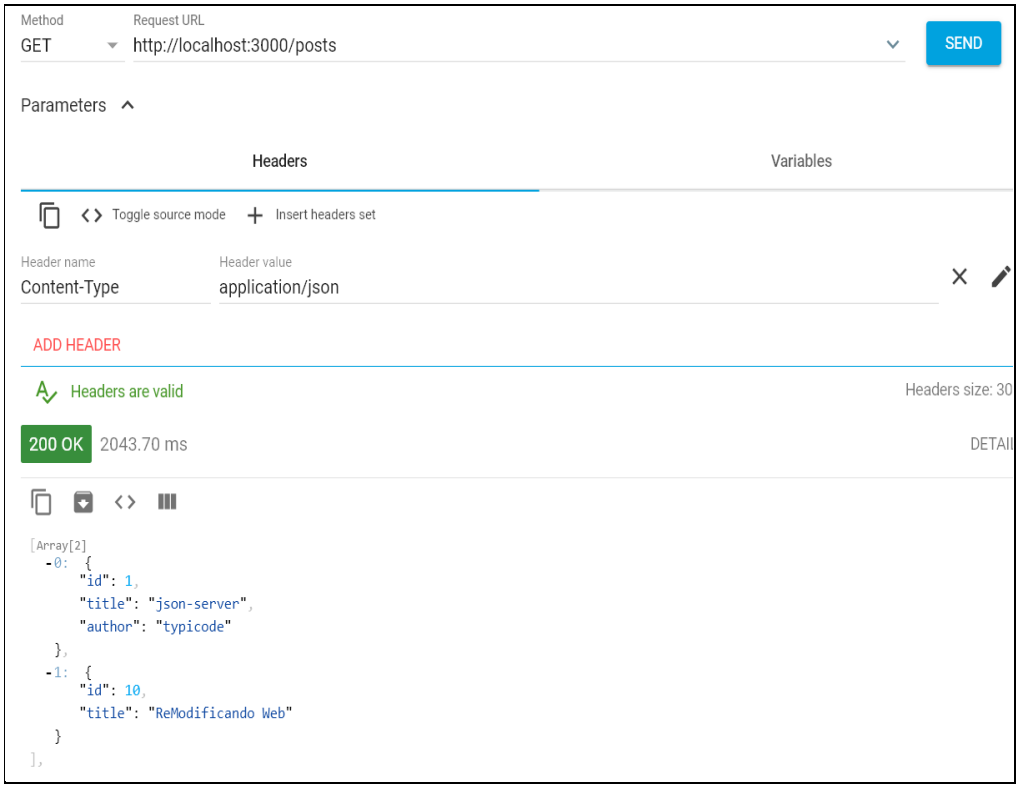
Supongamos que queremos modificar únicamente el title del registro con id 10, vamos a mandar únicamente el campo title, el autor no, queremos que sea el mismo.

```
{
  "id": 10,
  "title": "ReModificando Web"
}
```



Recibimos un código 200 indicando que todo ha ido bien. Además, recibimos el objeto modificado.

Vamos a realizar una petición GET a la entidad post a ver que registros contiene.



Aquí podemos ver que el registro con id 10 solo tiene 2 campos debido a lo que hemos comentado, con lo cual tendremos problemas con ese registro en futuras consultas

8.4. Petición PATCH

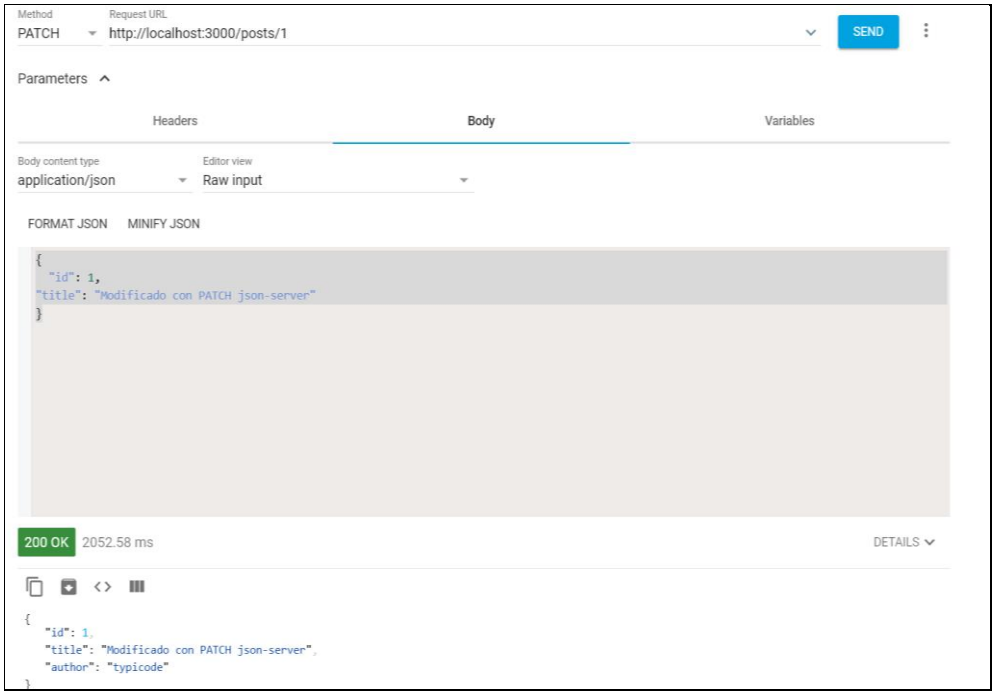
El problema anterior lo resuelve el método PATCH.

Como en el PUT hemos de seleccionar el método y de poner la url, en este caso deberemos indicar que vamos a mandar un objeto json en el body content type y en el body deberemos indicar las modificaciones que queremos realizar.

Con el PATCH si queremos podemos mandar los campos que únicamente queremos modificar sin que tengamos el problema que pasaba con el PUT

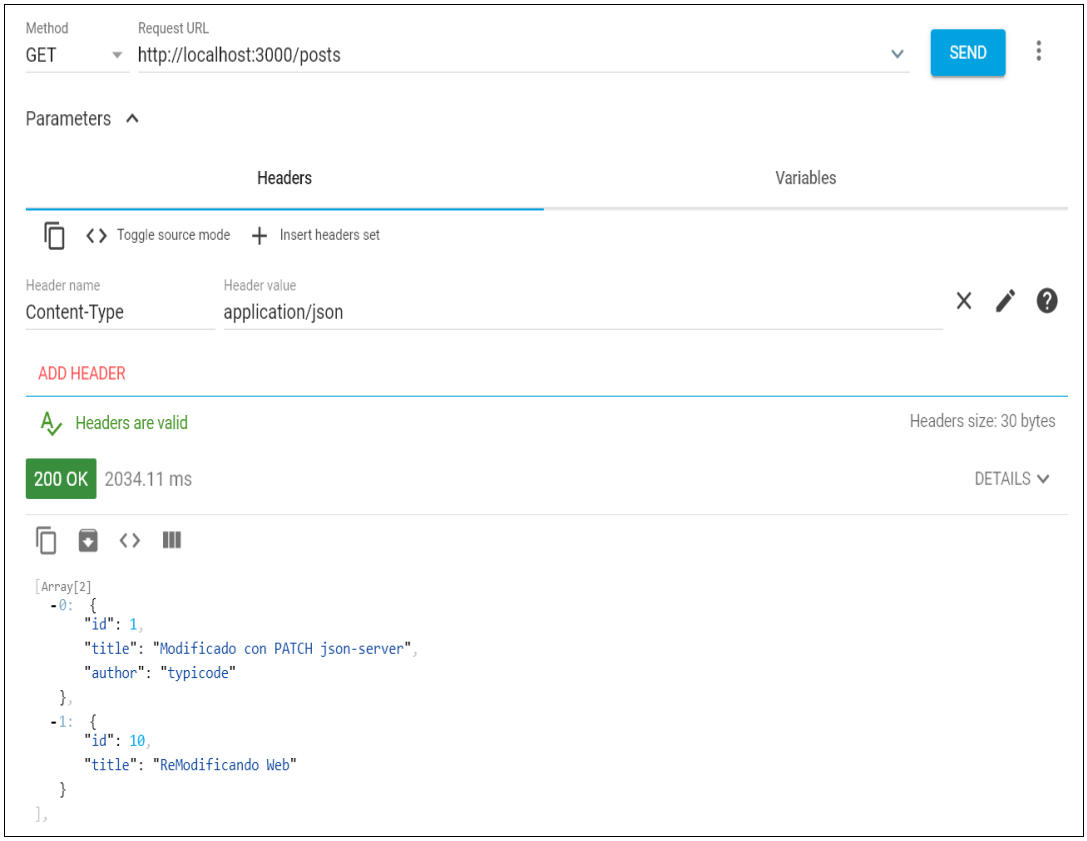
Vamos a modificar el registro con id 1 mandando solo el campo title

```
{
  "id": 1,
"title": "Modificado con PATCH json-server"
}
```



En este caso nos contesta con un código HTTP 200 que nos indica que todo ha ido bien. Además, nos devuelve el registro modificado. Aquí ya se puede ver que la estructura del registro sigue intacta, tiene los tres campos.

Vamos a realizar una petición GET para ver cómo está la entidad posts



Aquí podemos ver que aunque en la petición del PATCH para el id 1 que acabamos de realizar hayamos mandado solo el campo title el campo autor sigue manteniendo su valor. En cambio en el registro con id 10 en la que hicimos anteriormente un PUT sin el campo autor como se ve ha desaparecido de ese registro el campo autor

Esto se debe a la propiedad de idempotencia, es decir si se **debe modificar el registro original (datos y estructura)** por el registro que se manda en la petición o **dejar la estructura del registro original y modificar únicamente el contenido de los campos** incluidos en el body de la petición.

El primer caso lo hace el PUT y el segundo caso lo hace el PATCH

En desarrollo Web lo habitual es que la interface que se use para una modificación sea un formulario en el que previamente se hayan cargado los datos originales del registro a modificar y el usuario luego pueda modificar los campos que desee, de esta manera nos aseguramos que la estructura que se va a mandar sea la original (todos los campos) dando igual usar el PUT o el PATCH.

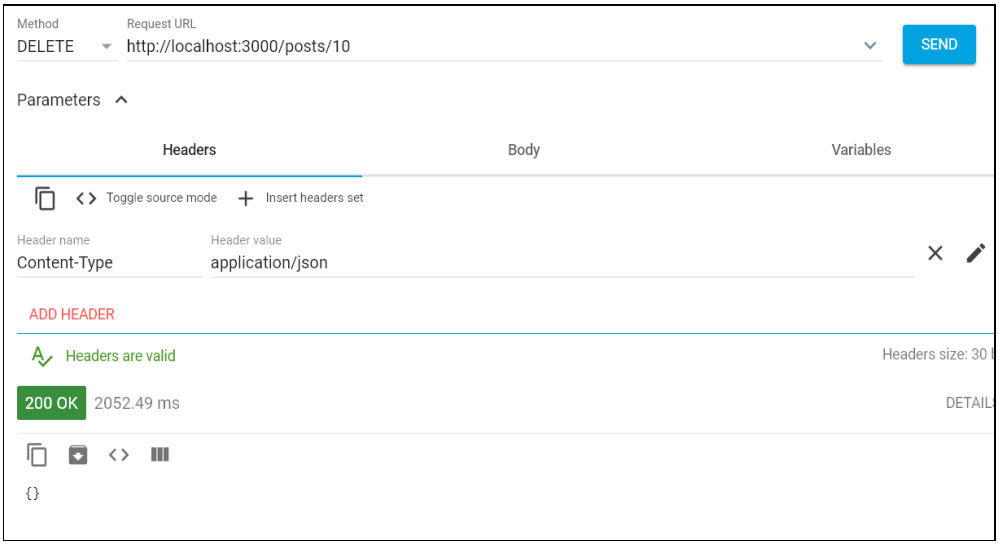
8.5. Petición DELETE

Vamos a eliminar un registro existente en la entidad posts, para ello deberemos utilizar el método DELETE.

Vamos a eliminar el registro con el id 10

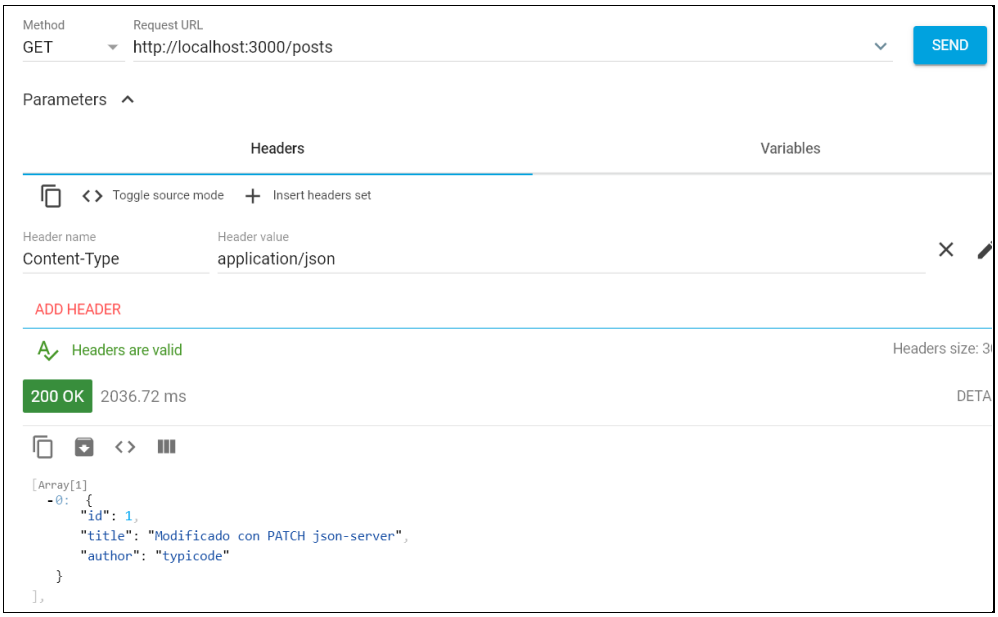
Para ello

Sólo debemos indicar el método y la url.

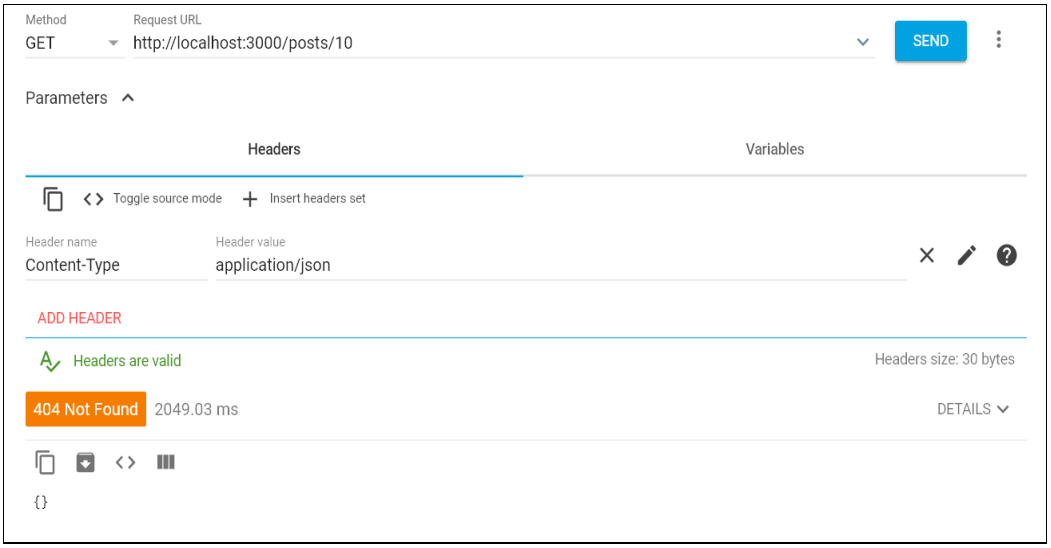


Aquí vemos que el servidor nos ha contestado con un 200 indicando que todo ha ido bien. Además, nos devuelve un objeto vacío.

Si realizamos una petición GET a la entidad post veremos que el registro con id 10 ya no existe.



Si ahora hiciéramos una petición GET para ese registro en concreto nos debería devolver un error, ya que no existe



Aquí podemos ver que la respuesta es un código 404.

Como JSON Server esta iniciado en modo watch todas las peticiones que ha recibido las ha ido registrando por la consola.