Desarrollando un Api Rest

Luis Antonio Manjarrez Torres

Contenido

Requerimientos

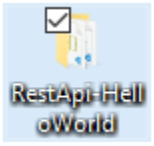
- Contar con, conexión a Internet
- Un editor de texto como Visual Studio Code
- Contar con node js instalado
- Contar con Docker instalado

Introducción

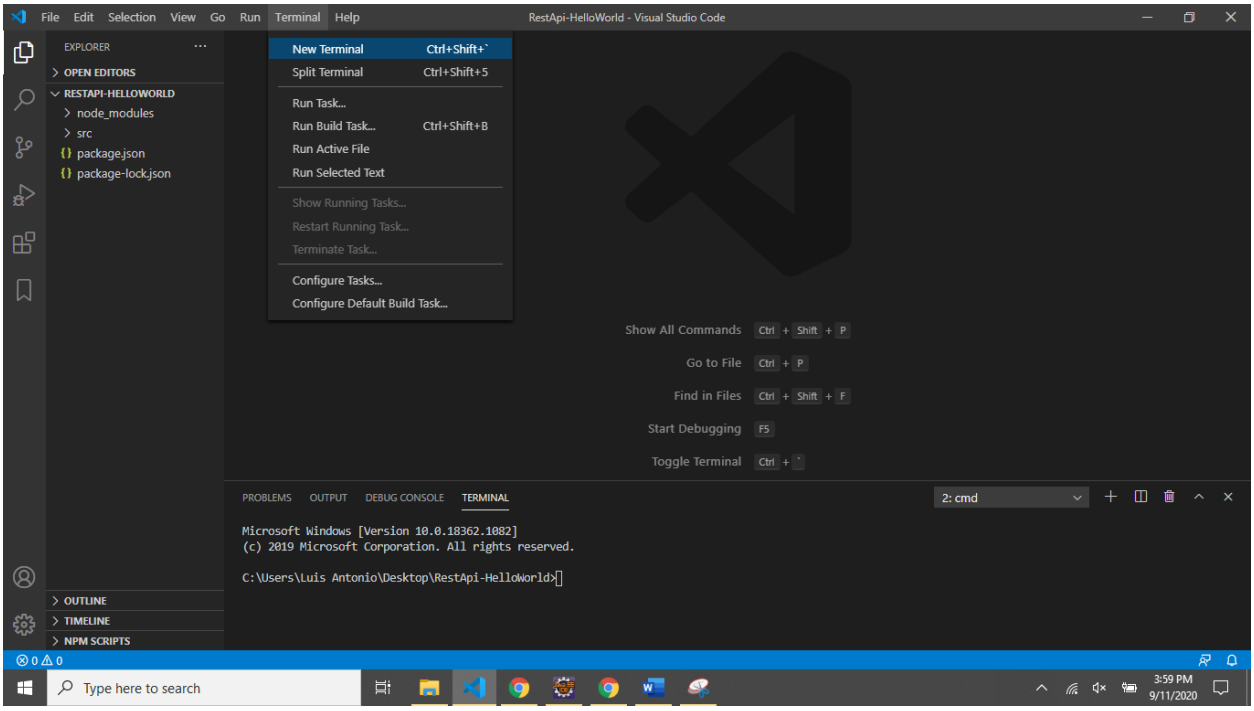
El lenguaje utilizado fue JavaScript, ya que es bastante flexible y sencillo de usar, para la comunicación con el api el formato de la información enviada y recibida será a través de json ya que es bastante fácil de utilizar.

Configuración Inicial

Para comenzar crearemos un directorio que va a contener nuestros servicios a utilizar en este caso desarrollaremos un api restful de comics.



Utilizaremos Visual Studio Code para la creación y manipulación de archivos que usaremos. Comenzaremos Abriendo una terminal que nos permitirá descargar los archivos necesarios para montar un servidor que nos permitirá utilizar nuestra aplicación.



Dentro de la terminal escribiremos los siguientes comandos:

- Npm init --yes: Crea un archivo json donde se encuentra información acerca del proyecto
- Npm i express morgan:
  - Express es un framework que nos proporciona un conjunto de funciones para aplicaciones web y móviles.
  - Morgan es un Middleware para registrar solicitudes HTTP en node.js.
- Npm i nodemon -D: Es una dependencia de desarrollo para que el servidor actualice la información con cada cambio que se hace

Crearemos el directorio “src” done tendremos nuestros servicios, dentro del directorio crearemos un index.js, este archivo tiene el propósito de crear las aplicaciones express, se definirá el puerto y crearan los servicios.

index.js

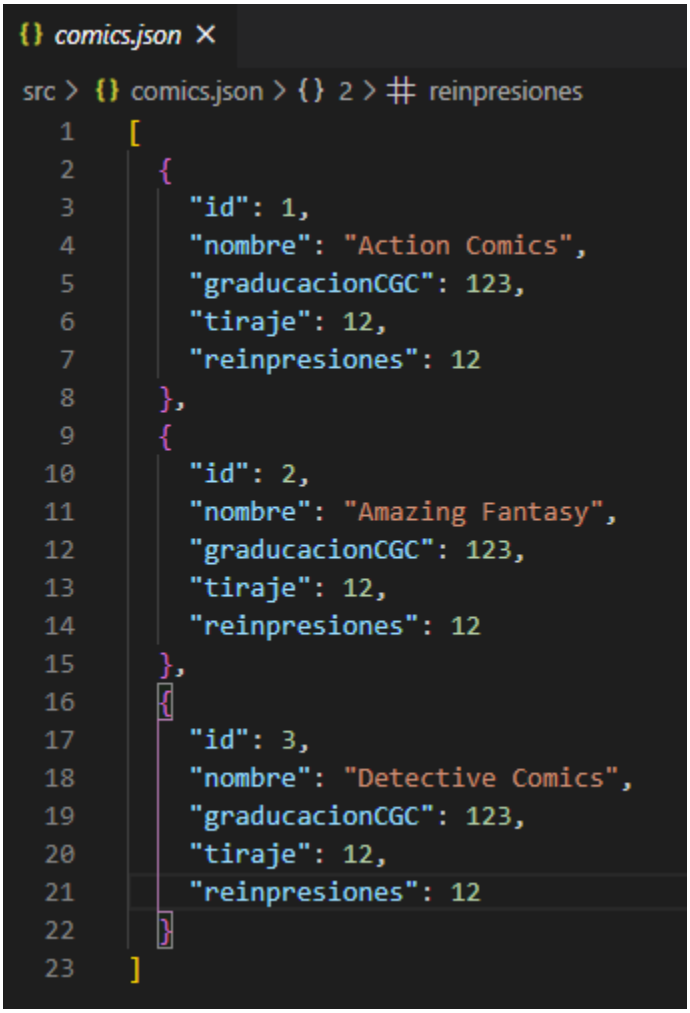
src > index.js > ...

```
const express = require('express');
const app = express();
const morgan = require('morgan');
//Settings
app.set('port', process.env.PORT || 3000);
app.set('json spaces', 1);
//Middlewares
app.use(morgan('dev'));
//  Estos comando nos permite recibir e interpretar
//  la informacion en formato json
app.use(express.urlencoded({ extended: false }));
app.use(express.json());

//Routes
//  Aqui definimos los archivos que contendran las rutas a utilizar para la obtencion
//  y recepcion de informacion
app.use(require('./routes/index-routes'));
//  Se define la ruta /api/sps/helloworld/v1 para todas las rutas que se pudieran encontrar
//  dentro del archivo
app.use('/api/sps/helloworld/v1', require('./routes/comic'));

//Starting server
app.listen(3000, () => {
  console.log('Server on port ', app.get("port"));
});
```

Archivos Json

Crearemos un directorio dentro de “src” llamado ”routes”, aquí agregaremos un archivo llamado comic donde se encontrarán los servicios disponibles, también crearemos un archivo json en el directorio “src” donde almacenamos información preestablecida, para un mejor entendimiento de los servicios.

El archivo json contendrá información como la siguiente:



Rutas

Mientras que el archivo de rutas tendrá los métodos http para el procesamiento de peticiones, comenzamos con una configuración inicial del archivo donde obtenemos los objetos necesarios como Router que nos permite establecer las rutas y los métodos http que se usaran.

```
JS comic.js
src > routes > JS comic.js > ...
1   //Cargamos y definimos el objeto router que nos permite
2   //utilizar las peticiones http
3   const { Router } = require("express");
4   const router = Router();
5   //Nos permite recorrer listas de informacion
6   const _ = require("underscore");
7   //Nos permite realizar peticiones a otros servicios
8   const fetch = require("node-fetch");
9   //Archivo json que contiene informacion preestablecida
10  const comics = require('../comics.json');
```

Finalmente exportamos el módulo desarrollado:

```
89    module.exports = router;
```

La estructura de las funciones http será la siguiente

Router.<método hhtp>(“<ruta>”,<async>(req,resp)=>{ <codigo>})

El método GET: este método es el mas sencillo de entender ya que únicamente devolvemos información a través del response de la aplicación en formato json, mediante la excepción controlamos cualquier error que pudiera ocurrir

```
12  router.get('/', (req, res) => {
13    try {
14      res.json(comics);
15    } catch (err) {
16      res.status(500).json({ error: "Ocurrio un error inesperado." + err });
17    }
18  });
```

El método POST:

```
router.post('/', (req, res) => {
  try {
    //Recibimos informacion de la peticion
    const { nombre, graducacionCGC, tiraje, reinpresiones } = req.body;
    //Verificacmos que la informacion exista, si exite se agregara la informacion
    //de lo contrario se devolvera un status de error
    if (nombre && graducacionCGC && tiraje && reinpresiones) {
      const id = comics.length + 1;
      const newComic = { id, ...req.body }
      comics.push(newComic);
      console.log(comics);
      res.json(comics);
    } else {
      res.status(400).json({ error: "Peticion incorrecta." });
    }
  } catch (err) {
    res.status(500).json({ error: "Ocurrio un error inesperado." + err });
  }
});
```

El método PUT:

```
router.put("/:id", (req, res) => {
  try {
    //Obtenemos el id a modificar
    const { id } = req.params;
    //Obtenemos la nueva informacion
    const { nombre, graducacionCGC, tiraje, reinpresiones } = req.body;
    const bool = false;
    //Validamos y reemplazamos la informacion
    if (nombre && graducacionCGC && tiraje && reinpresiones) {
      _.each(comics, (comic, index) => {
        if (comic.id == id) {
          comic.nombre = nombre;
          comic.graducacionCGC = graducacionCGC;
          comic.tiraje = tiraje;
          comic.reinpresiones = reinpresiones;
          bool = true;
        }
      });
      if (bool) {
        res.json(comics);
      } else {
        res.status(404).json({ error: "Elemento no encontrado." });
      }
    } else {
      res.status(400).json({ error: "Peticion incorrecta." });
    }
  } catch (err) {
    res.status(500).json({ error: "Ocurrio un error inesperado." + err });
  }
});
```

El método DELETE:

```
71  router.delete("/:id", (req, res) => {
72    try {
73      //Obtenemos el id a borrar
74      const { id } = req.params;
75      //Buscamos la informacion y la borramos
76      _.each(comics, (comic, index) => {
77        if (comic.id == id) {
78          comics.splice(index, 1);
79          bool = true;
80        }
81      });
82      if (bool) {
83        res.json(comics);
84      } else {
85        res.status(404).json({ error: "Elemento no encontrado." });
86      }
87    } catch (err) {
88      res.status(500).json({ error: "Ocurrio un error inesperado." + err });
89    }
90  });
```

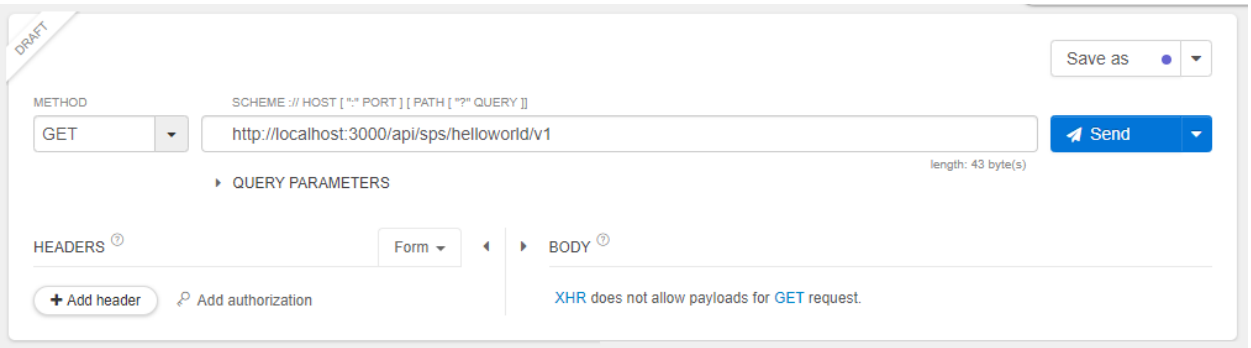
Obteniendo datos de otro Servicio:

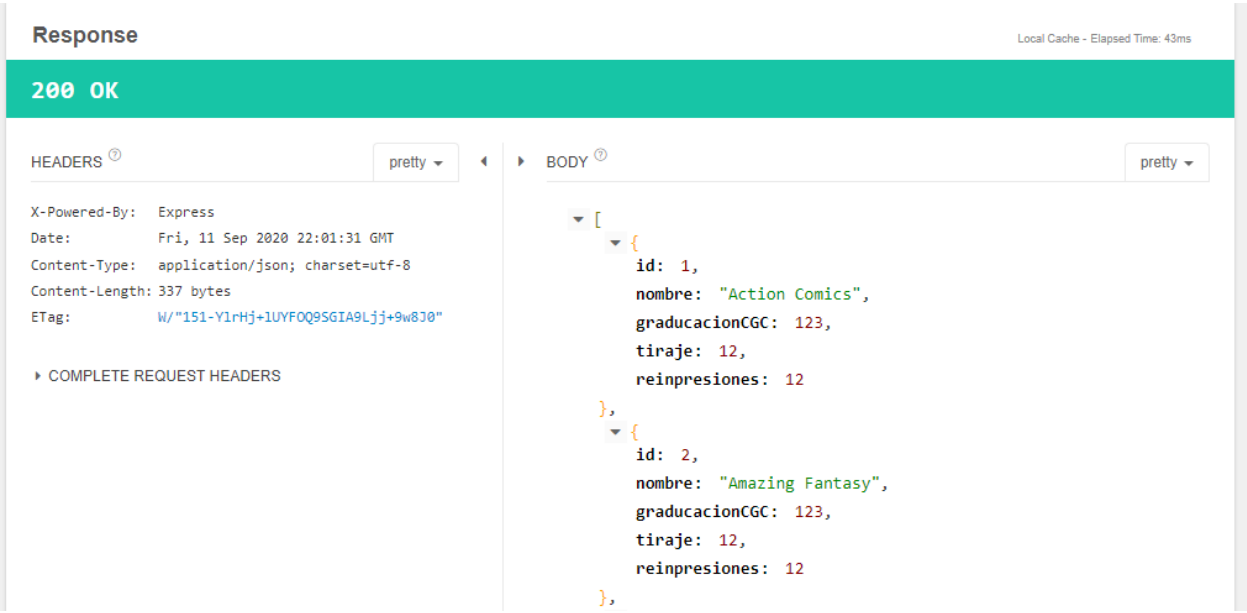
```
92  router.get("/users", async (req, res) => {
93    try {
94      const response = await fetch("https://jsonplaceholder.typicode.com/users");
95      const users = await response.json();
96      res.json(users);
97    } catch (err) {
98      res.status(500).json({ error: "Ocurrio un error inesperado." + err });
99    }
100 });
```

Pruebas

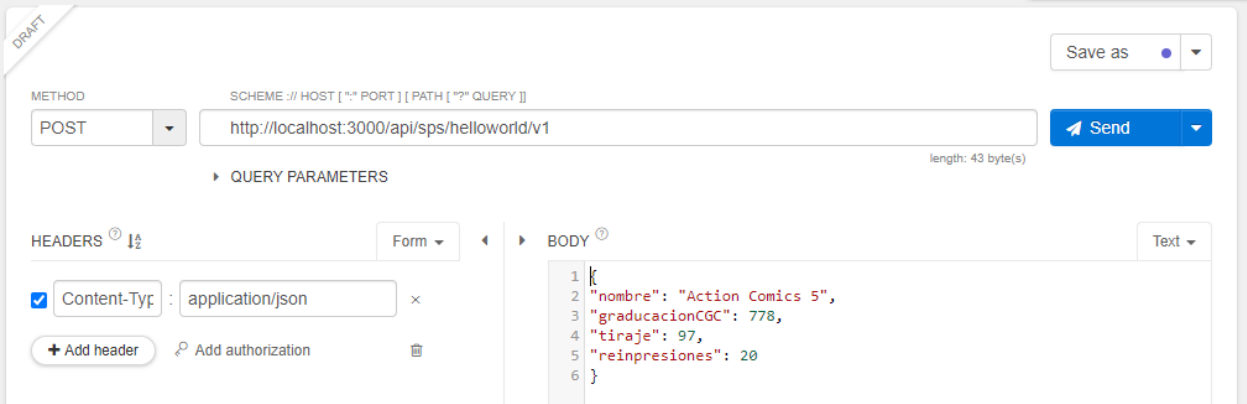
Para probar los servicios utilizaremos una extensión de Chrome, que nos permite probar API REST.

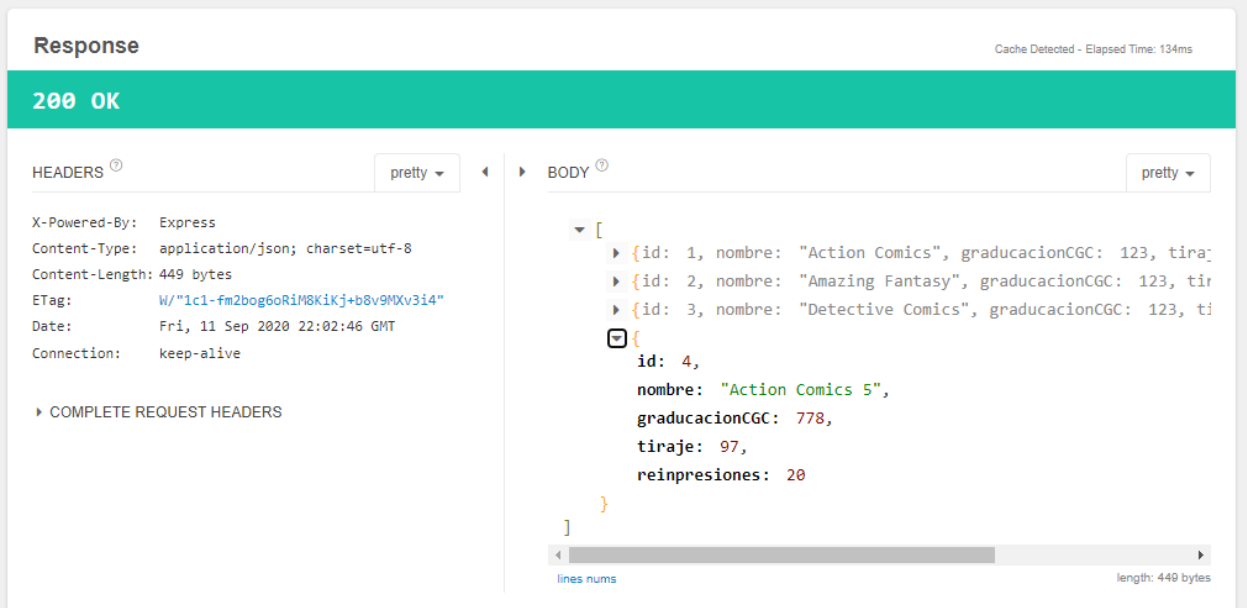
GET:



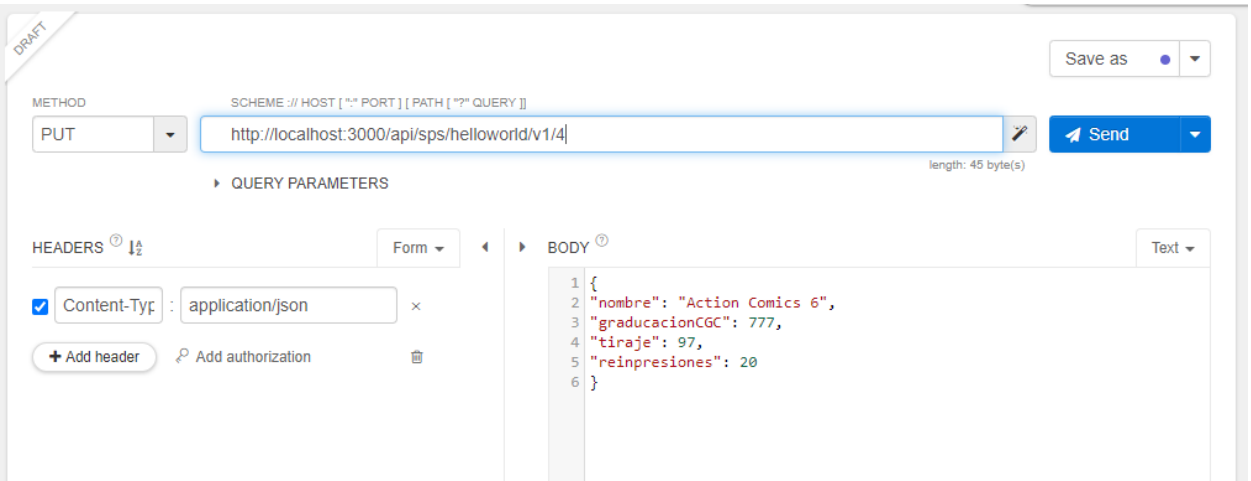


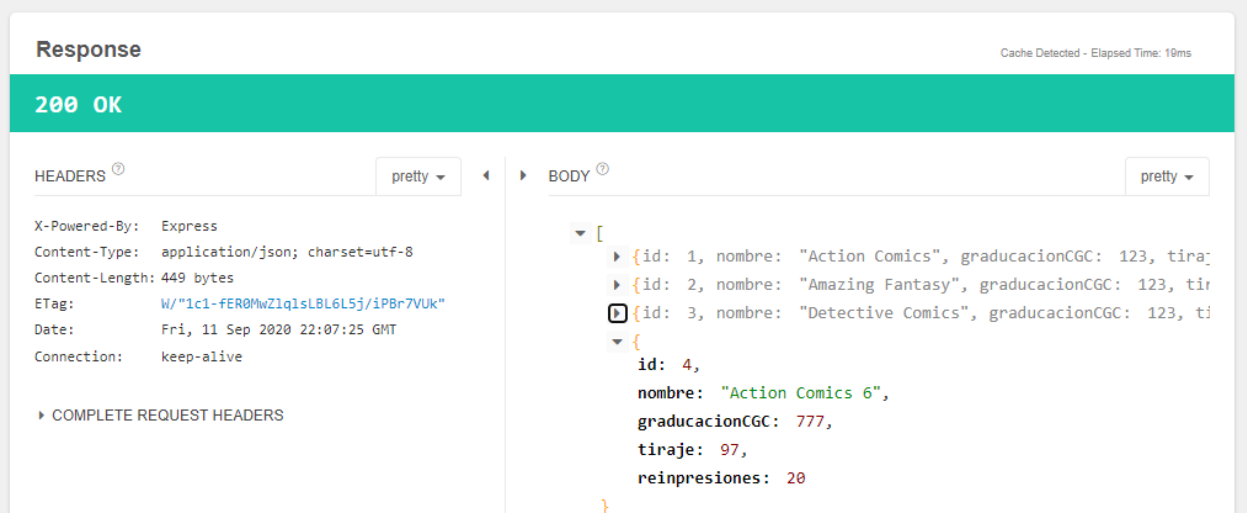
POST:



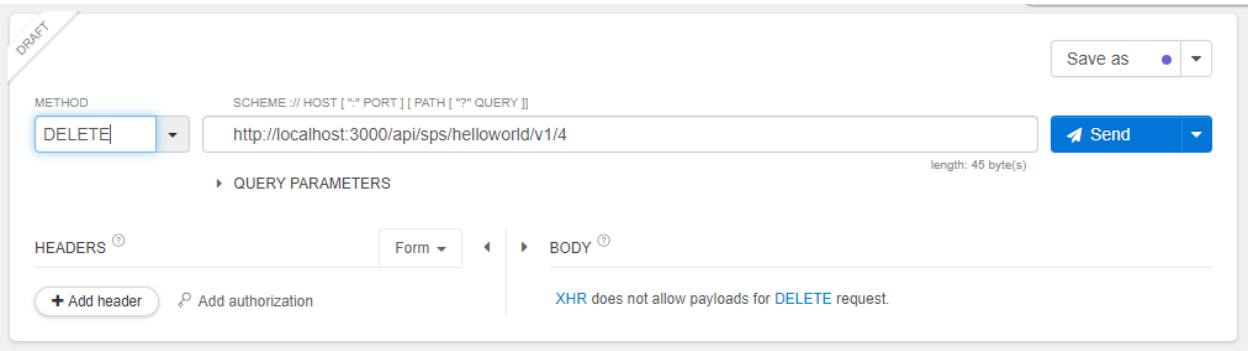

Response
Cache Detected - Elapsed Time: 134ms
200 OK
HEADERS
pretty
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 449 bytes
ETag: W/"1c1-fm2bog6oRiM8KiKj+b8v9MXv3i4"
Date: Fri, 11 Sep 2020 22:02:46 GMT
Connection: keep-alive
COMPLETE REQUEST HEADERS
BODY
pretty
[
{id: 1, nombre: "Action Comics", graducacionCGC: 123, tira
{id: 2, nombre: "Amazing Fantasy", graducacionCGC: 123, tir
{id: 3, nombre: "Detective Comics", graducacionCGC: 123, ti
{
id: 4,
nombre: "Action Comics 5",
graducacionCGC: 778,
tiraje: 97,
reinpresiones: 20
}
]
lines nums
length: 449 bytes

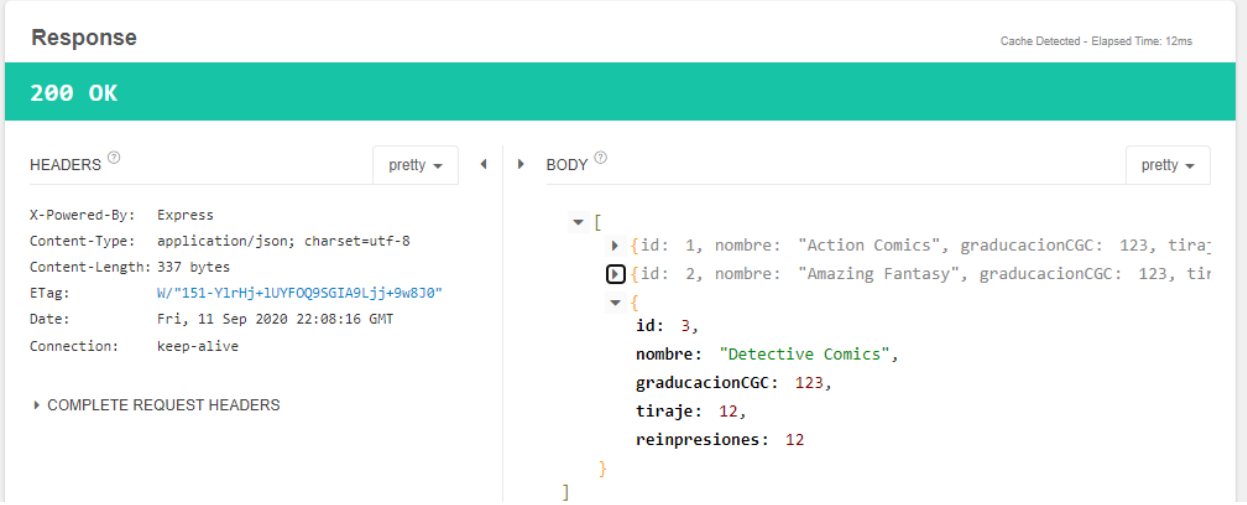
PUT:


DRAFT
Save as
METHOD
SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]
PUT
http://localhost:3000/api/sps/helloworld/v1/4
Send
length: 45 byte(s)
QUERY PARAMETERS
HEADERS
Form
Content-Typ
application/json
Add header
Add authorization
BODY
Text
{
"nombre": "Action Comics 6",
"graducacionCGC": 777,
"tiraje": 97,
"reinpresiones": 20
}

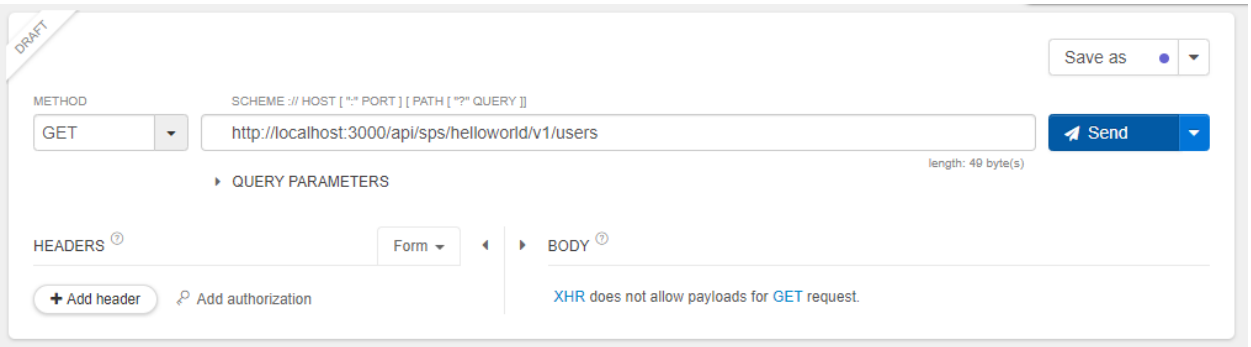

Response
Cache Detected - Elapsed Time: 19ms
200 OK
HEADERS
pretty
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 449 bytes
ETag: W/"1c1-fER0MwZlqlsLBL6L5j/iPBr7VUk"
Date: Fri, 11 Sep 2020 22:07:25 GMT
Connection: keep-alive
COMPLETE REQUEST HEADERS
BODY
pretty
[
{id: 1, nombre: "Action Comics", graducacionCGC: 123, tira
{id: 2, nombre: "Amazing Fantasy", graducacionCGC: 123, tir
{id: 3, nombre: "Detective Comics", graducacionCGC: 123, ti
{
id: 4,
nombre: "Action Comics 6",
graducacionCGC: 777,
tiraje: 97,
reinpresiones: 20
}

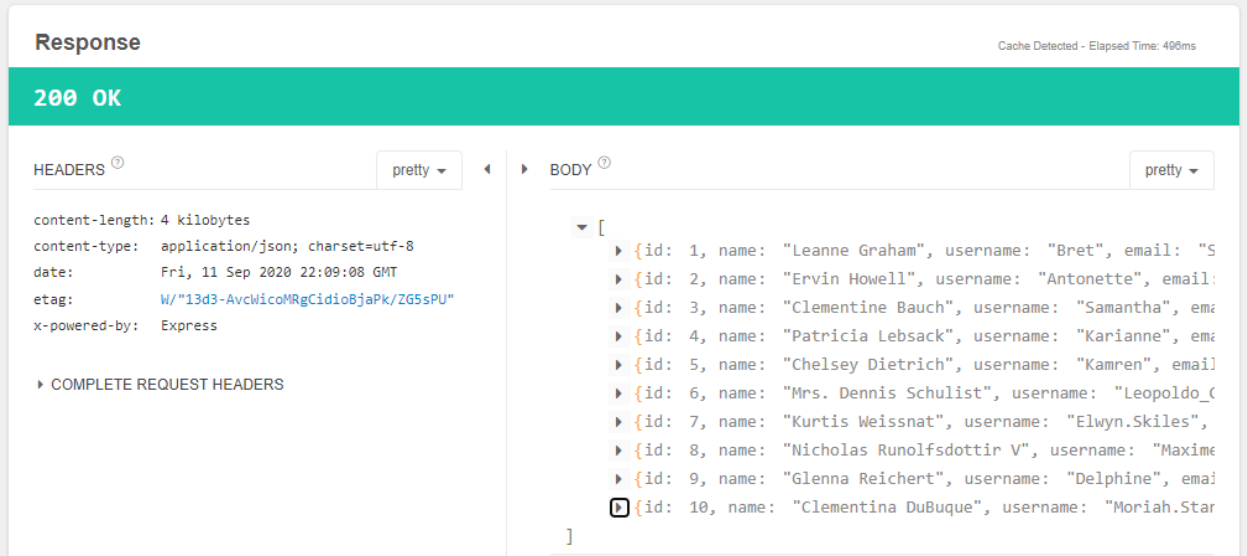
DELETE:





petición a otro Servicio:




Response
Cache Detected - Elapsed Time: 496ms
200 OK
HEADERS
pretty
content-length: 4 kilobytes
content-type: application/json; charset=utf-8
date: Fri, 11 Sep 2020 22:09:08 GMT
etag: W/"13d3-AvcWicoMRgCidioBjaPk/ZG5sPU"
x-powered-by: Express
COMPLETE REQUEST HEADERS
BODY
pretty
[
{id: 1, name: "Leanne Graham", username: "Bret", email: "S
{id: 2, name: "Ervin Howell", username: "Antonette", email
{id: 3, name: "Clementine Bauch", username: "Samantha", ema
{id: 4, name: "Patricia Lebsack", username: "Karianne", ema
{id: 5, name: "Chelsey Dietrich", username: "Kamren", email
{id: 6, name: "Mrs. Dennis Schulist", username: "Leopoldo_(
{id: 7, name: "Kurtis Weissnat", username: "Elwyn.Skiles",
{id: 8, name: "Nicholas Runolfsdottir V", username: "Maxime
{id: 9, name: "Glenna Reichert", username: "Delphine", emai
{id: 10, name: "Clementina DuBuque", username: "Moriah.Star
]