

# Desarrollando un Api Rest

Luis Antonio Manjarrez Torres

## Contenido

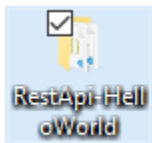
Requerimientos .....	2
Configuración Inicial .....	2
Archivos Json .....	5
Rutas .....	6
Pruebas .....	9
Haciendo un contenedor para la aplicación.....	13
Problemas Presentados .....	14

## Requerimientos

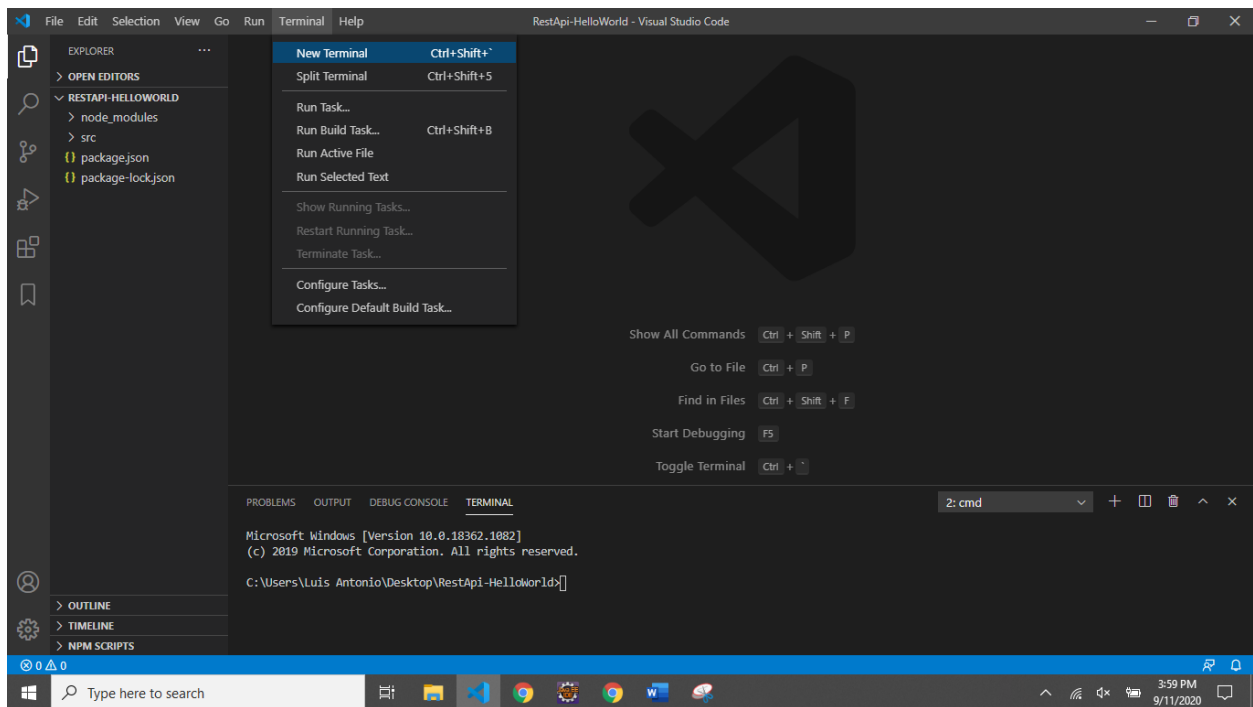
- Contar con, conexión a Internet
- Un editor de texto como Visual Studio Code
- Contar con node js instalado
- Contar con Docker instalado

## 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” donde 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.

JS index.js X

src > JS index.js > ...

```
1  const express = require('express');
2  const app = express();
3  const morgan = require('morgan');
4  //Settings
5  app.set('port', process.env.PORT || 3000);
6  app.set('json spaces', 1);
7  //Middlewares
8  app.use(morgan('dev'));
9  // Estos comando nos permite recibir e interpretar
10 // la informacion en formato json
11 app.use(express.urlencoded({ extended: false }));
12 app.use(express.json());
13
14 //Routes
15 // Aqui definimos los archivos que contendran las rutas a utilizar para la obtencion
16 // y recepcion de informacion
17 app.use(require('./routes/index-routes'));
18 // Se define la ruta /api/sps/helloworld/v1 para todas las rutas que se pudieran encontrar
19 // dentro del archivo
20 app.use('/api/sps/helloworld/v1', require('./routes/comic'));
21
22 //Starting server
23 app.listen(3000, () => {
24   console.log('Server on port ', app.get("port"));
25 });
```

## 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:

```
comics.json X
src > {} comics.json > {} 2 > # reinpresiones
1  [
2    {
3      "id": 1,
4      "nombre": "Action Comics",
5      "graduacionCGC": 123,
6      "tiraje": 12,
7      "reinpresiones": 12
8    },
9    {
10     "id": 2,
11     "nombre": "Amazing Fantasy",
12     "graduacionCGC": 123,
13     "tiraje": 12,
14     "reinpresiones": 12
15   },
16   {
17     "id": 3,
18     "nombre": "Detective Comics",
19     "graduacionCGC": 123,
20     "tiraje": 12,
21     "reinpresiones": 12
22   }
23 ]
```

## 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 X
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 http>("<ruta>",<async>(req,res)=>{ <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:

```

20 router.post('/', (req, res) => {
21   try {
22     //Recibimos informacion de la peticion
23     const { nombre, graducionCGC, tiraje, reinpresiones } = req.body;
24     //Verificamos que la informacion exista, si existe se agregara la informacion
25     //de lo contrario se devolvera un status de error
26     if (nombre && graducionCGC && tiraje && reinpresiones) {
27       const id = comics.length + 1;
28       const newComic = { id, ...req.body }
29       comics.push(newComic);
30       console.log(comics);
31       res.json(comics);
32     } else {
33       res.status(400).json({ error: "Petición incorrecta." });
34     }
35   } catch (err) {
36     res.status(500).json({ error: "Ocurrió un error inesperado." + err });
37   }
38 });

```

El método PUT:

```

40 router.put("/:id", (req, res) => {
41   try {
42     //Obtenemos el id a modificar
43     const { id } = req.params;
44     //Obtenemos la nueva informacion
45     const { nombre, graducionCGC, tiraje, reinpresiones } = req.body;
46     const bool = false;
47     //Validamos y reemplazamos la informacion
48     if (nombre && graducionCGC && tiraje && reinpresiones) {
49       _.each(comics, (comic, index) => {
50         if (comic.id == id) {
51           comic.nombre = nombre;
52           comic.graducionCGC = graducionCGC;
53           comic.tiraje = tiraje;
54           comic.reinpresiones = reinpresiones;
55           bool = true;
56         }
57       });
58       if (bool) {
59         res.json(comics);
60       } else {
61         res.status(404).json({ error: "Elemento no encontrado." });
62       }
63     } else {
64       res.status(400).json({ error: "Petición incorrecta." });
65     }
66   } catch (err) {
67     res.status(500).json({ error: "Ocurrió un error inesperado." + err });
68   }
69 });

```

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:

The screenshot shows the REST Client extension interface for a GET request. At the top, there's a 'DRAFT' label and a 'Save as' button. The 'METHOD' is set to 'GET'. The 'SCHEME' is 'http', 'HOST' is 'localhost:3000', 'PATH' is '/api/sps/helloworld/v1', and 'QUERY' is empty. The 'length' is 43 byte(s). Below the URL bar, there's a 'Send' button. Underneath, there's a 'QUERY PARAMETERS' section. The 'HEADERS' section is on the left, with a 'Form' dropdown and an 'Add header' button. The 'BODY' section is on the right, with a message: 'XHR does not allow payloads for GET request.'

The screenshot shows the 'Response' section of the REST Client extension. The status is '200 OK'. The 'Local Cache - Elapsed Time: 43ms' is shown. The 'HEADERS' section is on the left, with a 'pretty' dropdown. The 'BODY' section is on the right, with a 'pretty' dropdown. The response body is a JSON array with two objects:

```
[
  {
    id: 1,
    nombre: "Action Comics",
    graduacionCGC: 123,
    tiraje: 12,
    reinpresiones: 12
  },
  {
    id: 2,
    nombre: "Amazing Fantasy",
    graduacionCGC: 123,
    tiraje: 12,
    reinpresiones: 12
  }
]
```

POST:

The screenshot shows the REST Client extension interface for a POST request. The 'METHOD' is set to 'POST'. The 'SCHEME' is 'http', 'HOST' is 'localhost:3000', 'PATH' is '/api/sps/helloworld/v1', and 'QUERY' is empty. The 'length' is 43 byte(s). Below the URL bar, there's a 'Send' button. Underneath, there's a 'QUERY PARAMETERS' section. The 'HEADERS' section is on the left, with a 'Form' dropdown and an 'Add header' button. The 'BODY' section is on the right, with a 'Text' dropdown. The response body is a JSON object:

```
{
  "nombre": "Action Comics 5",
  "graduacionCGC": 778,
  "tiraje": 97,
  "reinpresiones": 20
}
```

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-fm2bog6oRiN8Kikj+b8v9MXv3i4"  
Date: Fri, 11 Sep 2020 22:02:46 GMT  
Connection: keep-alive

COMPLETE REQUEST HEADERS

BODY

pretty

```
[
  {id: 1, nombre: "Action Comics", graduacionCGC: 123, tira:
  {id: 2, nombre: "Amazing Fantasy", graduacionCGC: 123, tir
  {id: 3, nombre: "Detective Comics", graduacionCGC: 123, ti
  {
    id: 4,
    nombre: "Action Comics 5",
    graduacionCGC: 778,
    tiraje: 97,
    reinpresiones: 20
  }
}
```

length: 449 bytes

PUT:

DRAFT

Save as

METHOD

SCHEME // HOST [ ":" PORT ] [ PATH [ "?" QUERY ] ]

PUT

http://localhost:3000/api/sps/helloworld/v1/4

length: 45 byte(s)

Send

QUERY PARAMETERS

HEADERS

Form

☒ Content-Type: application/json

Add header

Add authorization

BODY

Text

```
1 {
2   "nombre": "Action Comics 6",
3   "graduacionCGC": 777,
4   "tiraje": 97,
5   "reinpresiones": 20
6 }
```

Response

Cache Detected - Elapsed Time: 10ms

200 OK

HEADERS

pretty

X-Powered-By: Express  
Content-Type: application/json; charset=utf-8  
Content-Length: 449 bytes  
ETag: W/"1c1-fER0MwZlqlsLBL6L5j/iPB-r7Vuk"  
Date: Fri, 11 Sep 2020 22:07:25 GMT  
Connection: keep-alive

COMPLETE REQUEST HEADERS

BODY

pretty

```
[
  {id: 1, nombre: "Action Comics", graduacionCGC: 123, tira:
  {id: 2, nombre: "Amazing Fantasy", graduacionCGC: 123, tir
  {id: 3, nombre: "Detective Comics", graduacionCGC: 123, ti
  {
    id: 4,
    nombre: "Action Comics 6",
    graduacionCGC: 777,
    tiraje: 97,
    reinpresiones: 20
  }
}
```

## DELETE:

DRAFT

Save as

METHOD

DELETE

SCHEME :// HOST [ ":" PORT ] [ PATH [ "?" QUERY ] ]

http://localhost:3000/api/sps/helloworld/v1/4

length: 45 byte(s)

Send

QUERY PARAMETERS

HEADERS

Form

+

Add header

Add authorization

BODY

XHR does not allow payloads for DELETE request.

Response

Cache Detected - Elapsed Time: 12ms

200 OK

HEADERS

pretty

X-Powered-By: Express  
Content-Type: application/json; charset=utf-8  
Content-Length: 337 bytes  
ETag: W/"151-Y1rHj+1UYFQ9SGIA9Ljj+9w8J0"  
Date: Fri, 11 Sep 2020 22:08:16 GMT  
Connection: keep-alive

COMPLETE REQUEST HEADERS

BODY

pretty

```
[
  {id: 1, nombre: "Action Comics", graduacionCGC: 123, tira: 12, reinpresiones: 12},
  {id: 2, nombre: "Amazing Fantasy", graduacionCGC: 123, tira: 12, reinpresiones: 12},
  {id: 3, nombre: "Detective Comics", graduacionCGC: 123, tira: 12, reinpresiones: 12}
]
```

petición a otro Servicio:

DRAFT

Save as

METHOD

GET

SCHEME :// HOST [ ":" PORT ] [ PATH [ "?" QUERY ] ]

http://localhost:3000/api/sps/helloworld/v1/users

length: 49 byte(s)

Send

QUERY PARAMETERS

HEADERS

Form

+

Add header

🔗

Add authorization

BODY

XHR does not allow payloads for GET request.

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-AvcWicoMRgCidio8jaPk/ZG5sPU"

x-powered-by: Express

COMPLETE REQUEST HEADERS

BODY

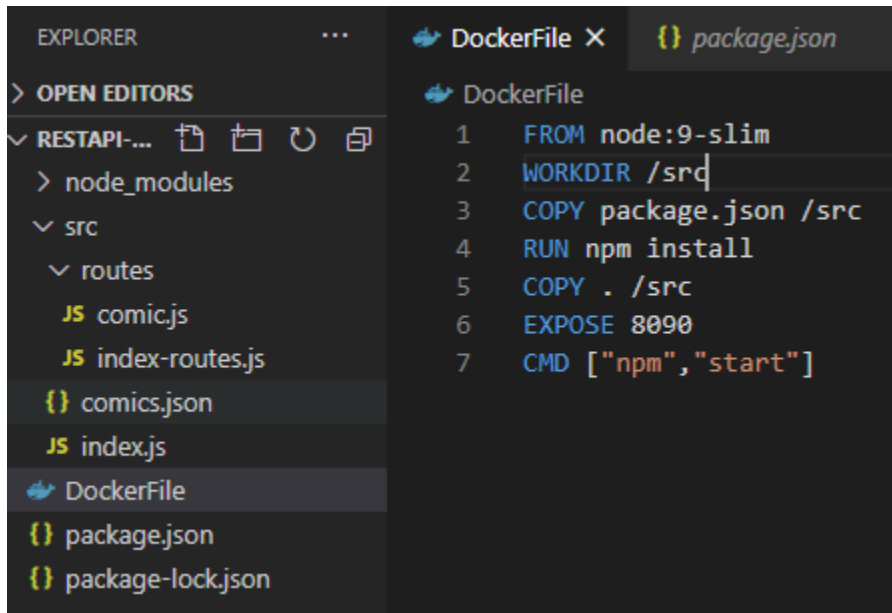
pretty

[  
 {id: 1, name: "Leanne Graham", username: "Bret", email: "S  
 {id: 2, name: "Ervin Howell", username: "Antonette", email:  
 {id: 3, name: "Clementine Bauch", username: "Samantha", em:  
 {id: 4, name: "Patricia Lebsack", username: "Karianne", em:  
 {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  
]

## Haciendo un contenedor para la aplicación

Una vez que nuestra aplicación fue desarrollada el siguiente paso, es crear un contenedor.

En el directorio raíz del proyecto crearemos un archivo “DockerFile” este nos servirá para lanzar la aplicación en el contenedor en Docker



Ya que tenemos el contenedor ejecutamos la siguiente instrucción que construirá el proyecto, para poder desplegarlo

```
C:\Users\Luis Antonio\Desktop\RestApi-HelloWorld>docker build -t node-docker .
```

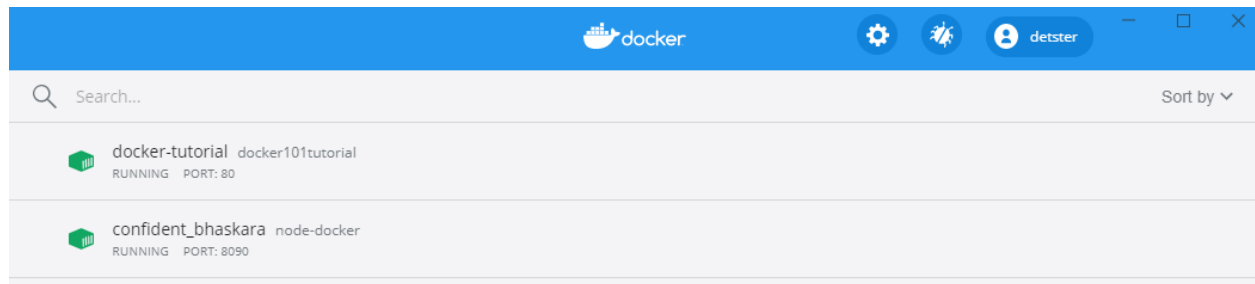
Ahora lo desplegaremos en el puerto 8090

```
C:\Users\Luis Antonio\Desktop\RestApi-HelloWorld>docker run -it -p 8090:8090 node-docker

> restapi-helloworld@1.0.0 start /src
> nodemon src/index.js

[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
Server on port 8090
```

Docker cuenta un Dashboard donde podemos visualizar la información de las aplicaciones desplegadas, podemos observar que el dashboard muestra dos aplicaciones desplegadas una en el puerto 80 y otra en 8090, en este caso la nuestra es la que desplegamos en el puerto 8090



## Problemas Presentados

Al conocer el lenguaje de programación sobre el cual trabajaría, los problemas presentados fueron muy pocos, realmente el problema más grande que tuve fue la desactualización de mi ordenador para instalar Docker, bastó con actualizar el ordenador para poder concluir con la práctica, investigue bastante y me documente acerca de las API Rest, microservicios y contenedores, aprendí a hacerlo en Java y me resulto bastante fácil de desarrollar, no contaba con los programas necesarios por lo cual no fue una opción, opte por Javascript porque no lo conozco del todo bien y es una nueva oportunidad de aprender nuevas cosas.

Aunque desarrolle lo solicitado aun me falta aprender mucho sobre los contenedores y microservicios ya que esta fue una aplicación bastante sencilla.