

API-REST



Despliegue del API-REST con docker

1. Creación de la API REST

Para la creación de este proyecto utilice lo siguiente:

- Node JS con la versión v12.17.0, para crear el servidor HTTP, que se encargará de gestionar las rutas de la API REST.
- TypeScript con la versión 3.9.7 como lenguaje de programación.
- MySQL en la versión 5.7.19 como gestor de la base de datos.
- Visual Studio Code cómo editor de código

Se puede clonar el proyecto con el link:

https://github.com/OWLSALEX/API_REST.git.

El servidor está configurado para arrancar en el puerto 5000.

Se creó un endpoint en `/api/sps/helloworld/v1`, para hacer el consumo desde un cliente REST.

El proyecto necesita una base de datos, por lo tanto se debe ejecutar el siguiente script en mysql.

```
CREATE DATABASE MI_API;
USE MI_API;
CREATE TABLE GAMES (
  ID_GAME          INT NOT NULL AUTO_INCREMENT,
  NAME_GAME        VARCHAR(30) NOT NULL,
  DESCRIPTION_GAME VARCHAR(30) NOT NULL,
  PRIMARY KEY (ID_GAME) );
```

El archivo para la conexión a la base de datos se encuentra en src/keys.ts

```
export default {  
  database: {  
    host: 'localhost', //Host de la base de datos  
    user: 'root',      //Usuario root  
    password: '',      //Contraseña del usuario root  
    database: 'MI_API' //Nombre de la base de datos  
  }  
}
```

Este proyecto está diseñado para realizar un CRUD sobre la tabla creada en la base de datos.

Estos son los métodos de petición para realizar el CRUD. (Esta configuración se encuentra en el archivo /src/Routes/apiRoutes.ts).

```
/**  
 * Método de configuración de las rutas,  
 * llamando los métodos creados para obtener  
 * los datos.  
 */  
config() {  
  this.router.get('/', api.list); //Ruta principal que muestra una lista  
  this.router.get('/:id', api.getOne); //Ruta que muestra un registro por ID  
  this.router.post('/', api.create); //Ruta que crea un nuevo registro  
  this.router.put('/:id', api.update); //Ruta que modifica un registro por ID  
  this.router.delete('/:id', api.delete); //Ruta que elimina un registro por ID  
}
```

Para desarrollo es necesario ejecutar los siguientes comandos en dos terminales diferentes:

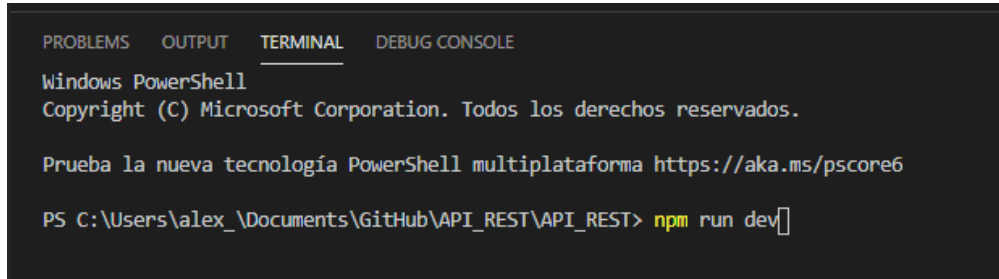
- npm run watch: Para convertir el código de TypeScript a JavaScript, mientras vigila los cambios.
- npm run dev: Para iniciar el servidor

Nota: Este proyecto ya lo había creado anteriormente como un ejercicio de práctica durante la cuarentena, solo lo reutilice y lo adapte a lo que se me pide. Me sabe en un video de YouTube del canal llamado Fazt



2. Iniciar Proyecto

Para iniciar el proyecto localmente, se debe de dirigir a la ruta principal del proyecto y ejecutar el comando `npm run dev`.

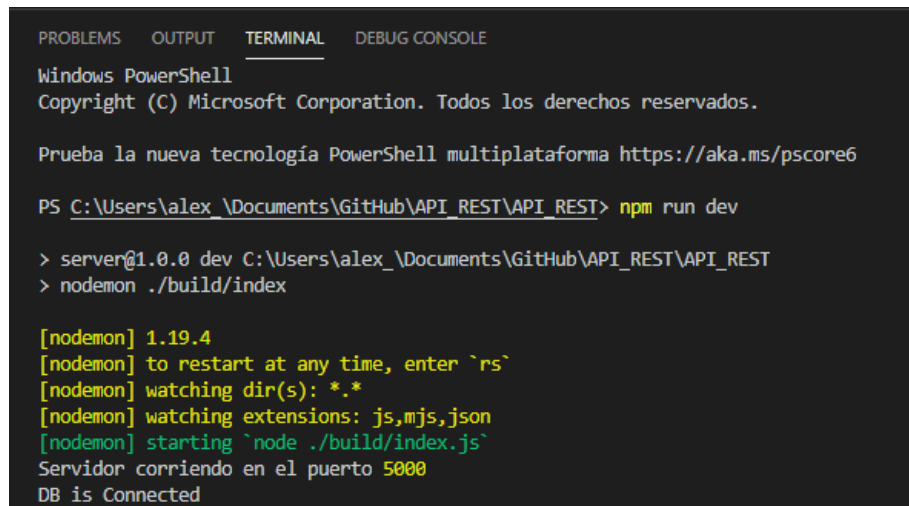


```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\alex\Documents\GitHub\API_REST\API_REST> npm run dev
```

Se mostrará un mensaje en consola donde indica, que el servidor está corriendo en el puerto 5000 y está conectado con la base de datos.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

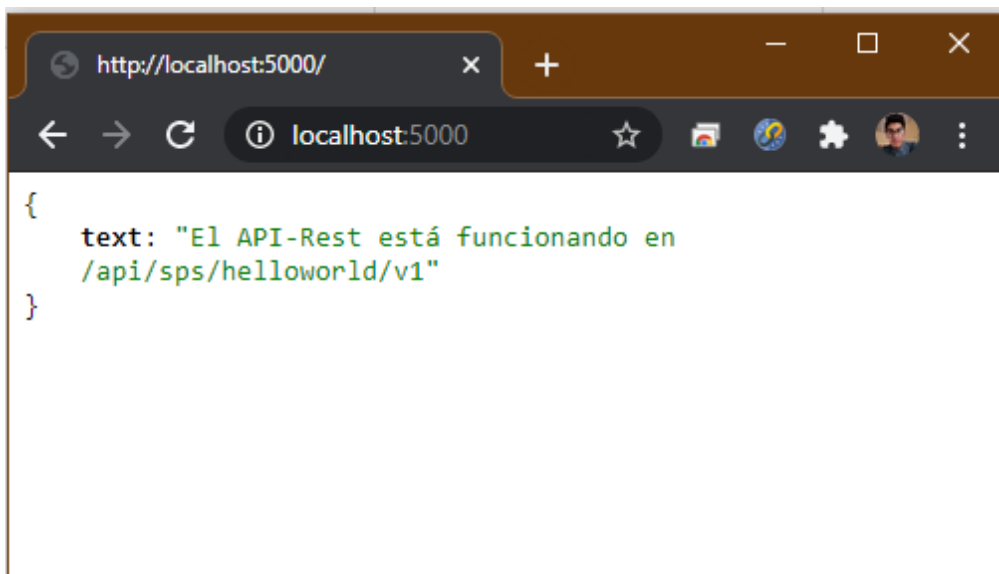
Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\alex\Documents\GitHub\API_REST\API_REST> npm run dev

> server@1.0.0 dev C:\Users\alex\Documents\GitHub\API_REST\API_REST
> nodemon ./build/index

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./build/index.js`
Servidor corriendo en el puerto 5000
DB is Connected
```

En el navegador escribimos `localhost:5000` y mostrará un mensaje.



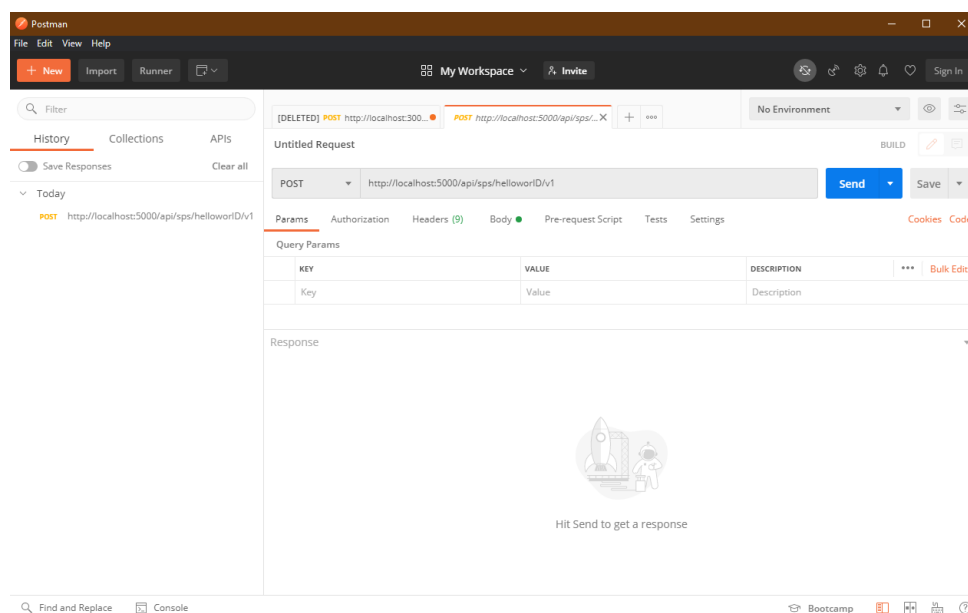
```
{
  text: "El API-Rest está funcionando en /api/sps/helloworld/v1"
}
```

Al escribir la nueva ruta, se mostraran los registros obtenidos desde la tabla de la base de datos, si no es así, se debe de ingresar información con un cliente REST.



3. Cliente REST

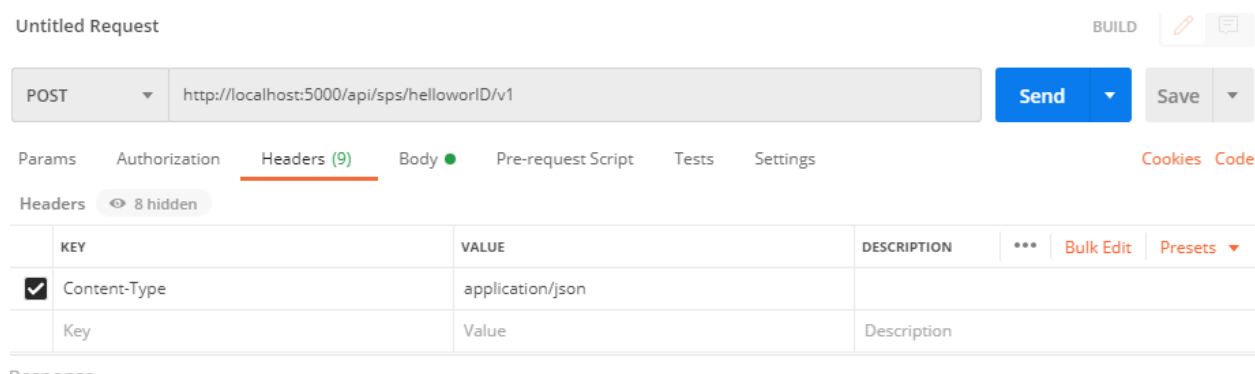
El cliente rest con el que trabajaré es POST MAN.



4. Registrar datos

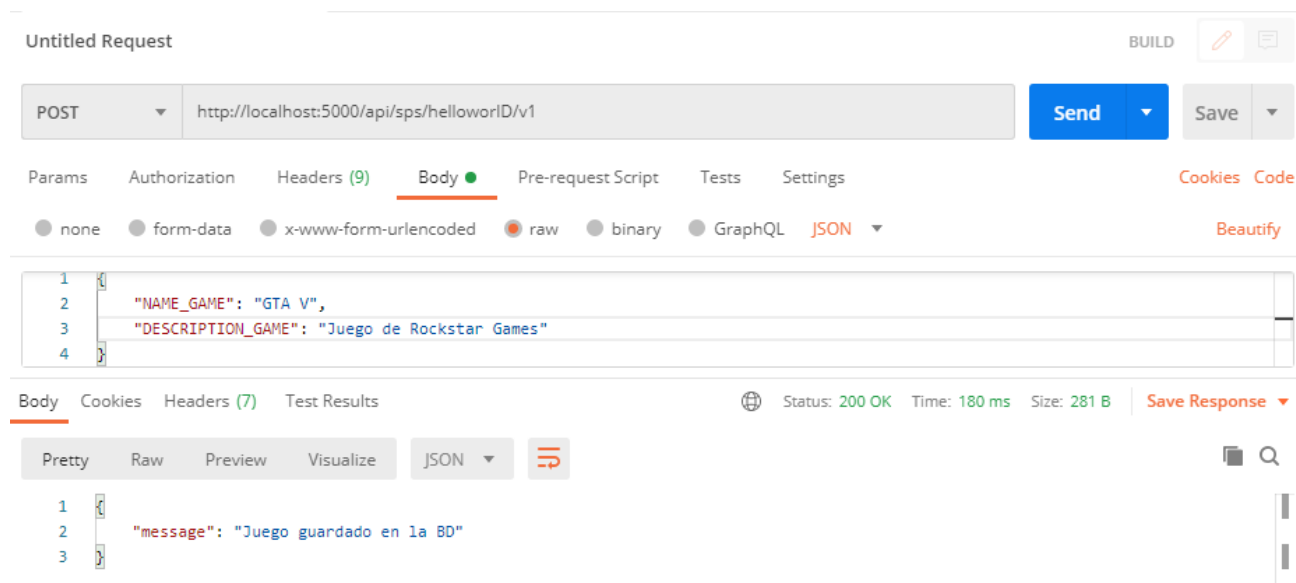
Para registrar un nuevo dato, es necesario indicar la ruta del API REST, en este caso es <http://localhost:5000/api/sps/helloworld/v1>.

En la parte de “Headers”, se le indica el tipo de valor que se va a enviar, es decir, se va a enviar una petición en formato Json.



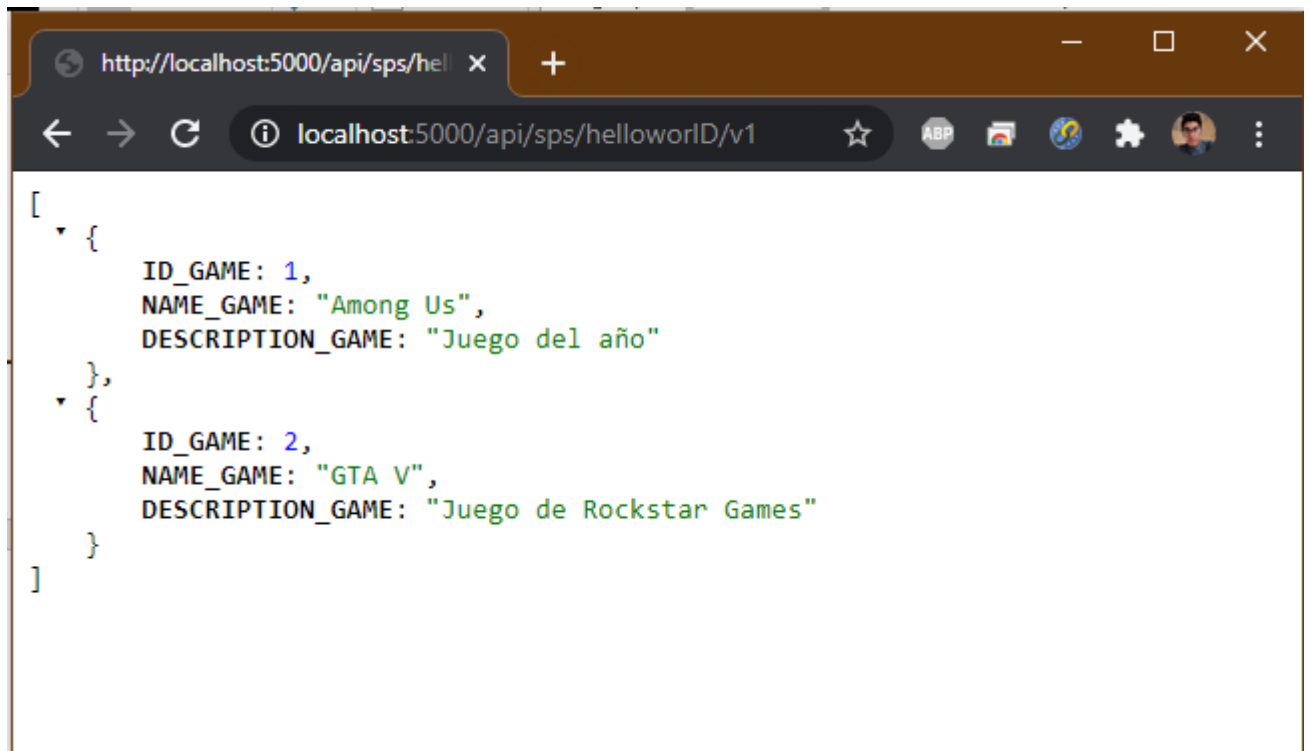
En la sección de “Body” se activa la opción “raw”, para enviarle los valores al servidor para que los almacene en la base de datos, esto en formato json con los datos correspondientes a la tabla.

Al enviar la solicitud al servidor de tipo POST, nos mostrara un mensaje de respuesta.

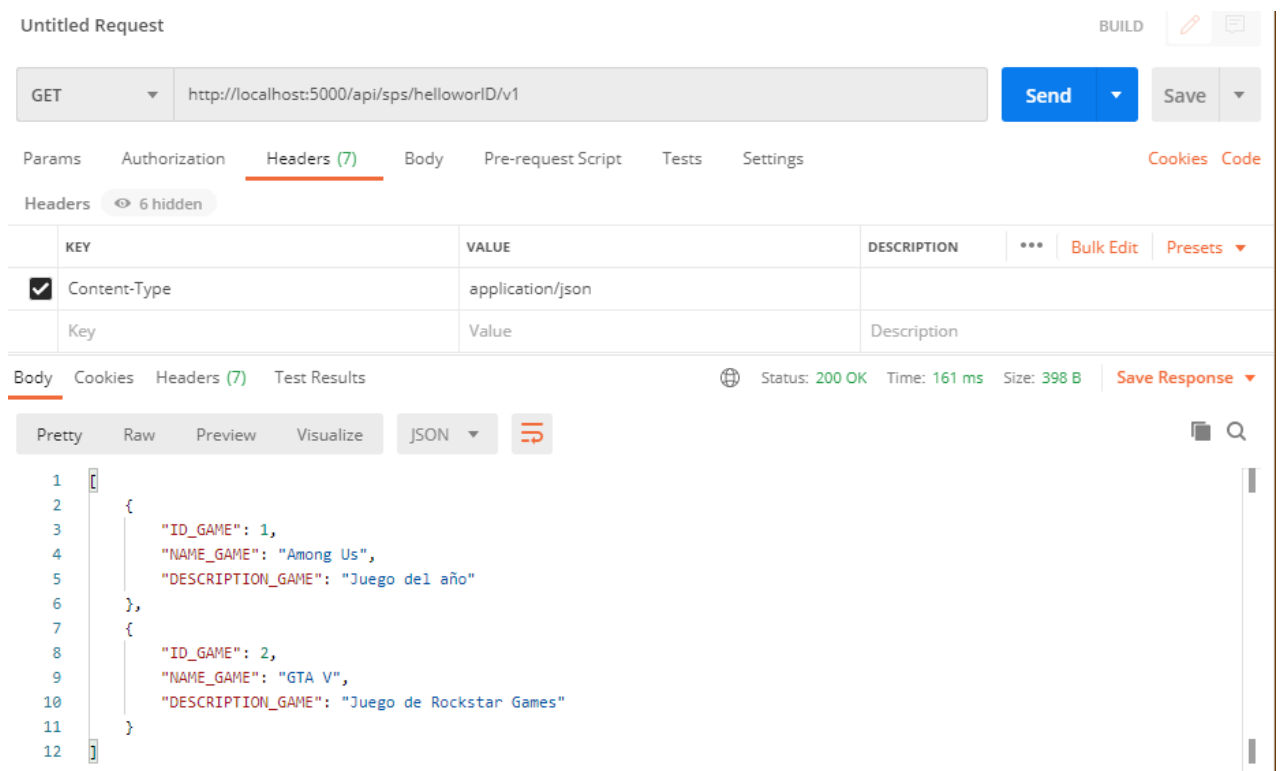


5. Consultar

Automáticamente en la ruta de la API REST, se muestran los registros almacenados en la base de datos.



Para consultar la información con el cliente rest, cambiamos el tipo de petición a GET.



Si deseamos consultar un registro mediante su ID, se agrega el ID delante de la ruta del API REST, por ejemplo <http://localhost:5000/api/sps/helloworld/v1/2>

The screenshot shows a REST client interface with a GET request to `http://localhost:5000/api/sps/helloworld/v1/2`. The 'Headers' tab is active, showing a `Content-Type: application/json` header. The 'Body' tab shows the response in JSON format:

```
{
  "ID_GAME": 2,
  "NAME_GAME": "GTA V",
  "DESCRIPTION_GAME": "Juego de Rockstar Games"
}
```

At the bottom, the status is `200 OK`, time is `193 ms`, and size is `322 B`.

6. Actualizar

Para actualizar algún registro, se debe de indicar lo siguiente:

- Seleccionar el método PUT
- Indicar el ID del registro que se desea actualizar al final de la ruta
- Enviar petición

Vamos a actualizar el segundo registro, cambiándole la descripción del juego.

The screenshot shows a REST client interface with a PUT request to `http://localhost:5000/api/sps/helloworld/v1/2`. The 'Body' tab is active, showing the request in JSON format:

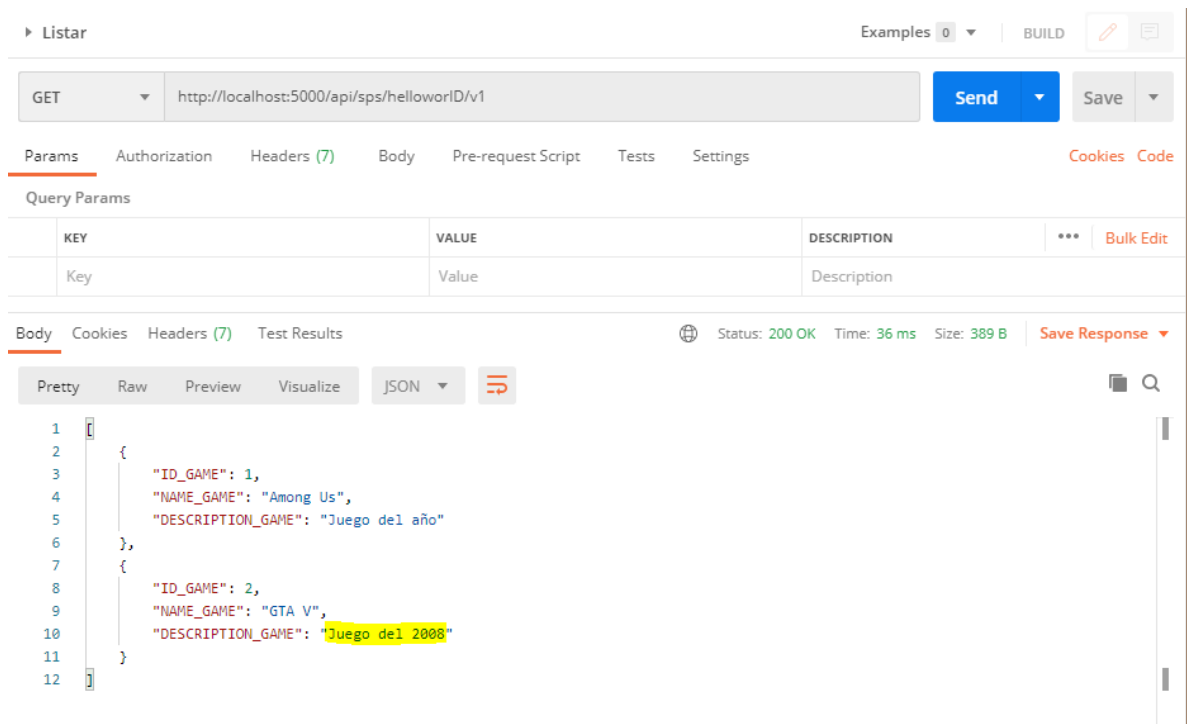
```
{
  "NAME_GAME": "GTA V",
  "DESCRIPTION_GAME": "Juego del 2008"
}
```

The 'Body' tab also shows the response in JSON format:

```
{
  "message": "El juego ha sido actualizado en la BD"
}
```

At the bottom, the status is `200 OK`, time is `36 ms`, and size is `295 B`.

Para verificar, listamos todos los registros.



▶ Listar

Examples 0 BUILD

GET http://localhost:5000/api/sps/helloworld/v1 Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 36 ms Size: 389 B Save Response

Pretty Raw Preview Visualize JSON

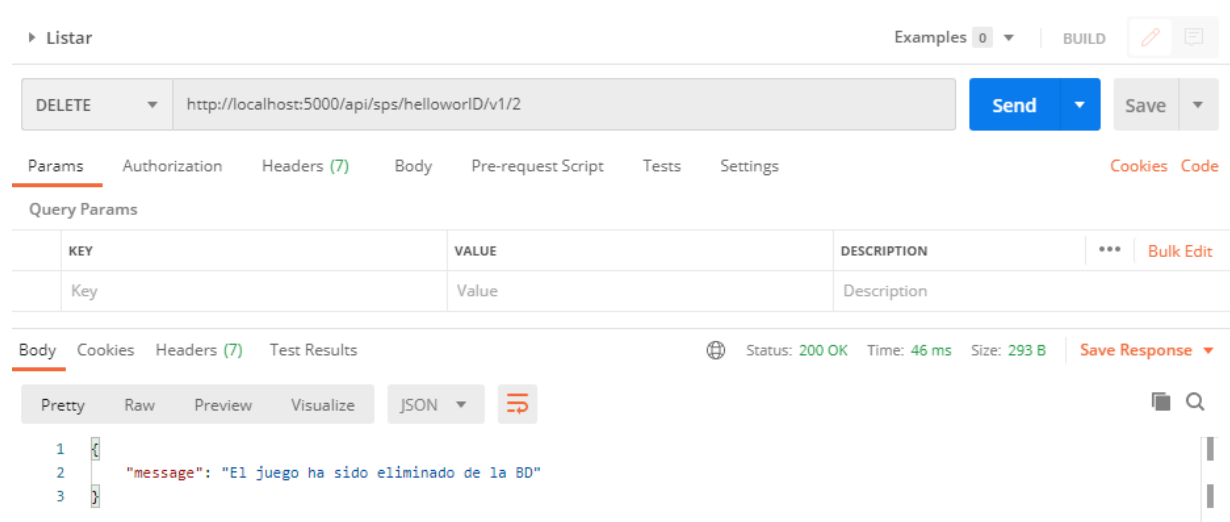
```
1 {
2   {
3     "ID_GAME": 1,
4     "NAME_GAME": "Among Us",
5     "DESCRIPTION_GAME": "Juego del año"
6   },
7   {
8     "ID_GAME": 2,
9     "NAME_GAME": "GTA V",
10    "DESCRIPTION_GAME": "Juego del 2008"
11  }
12 }
```

7. Eliminar

Para eliminar algún registro, se debe sé indicar lo siguiente:

- Seleccionar el método DELETE
- Indicar el ID del registro a eliminar al final de la ruta
- Enviar petición

Vamos a eliminar el segundo registro.



▶ Listar

Examples 0 BUILD

DELETE http://localhost:5000/api/sps/helloworld/v1/2 Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

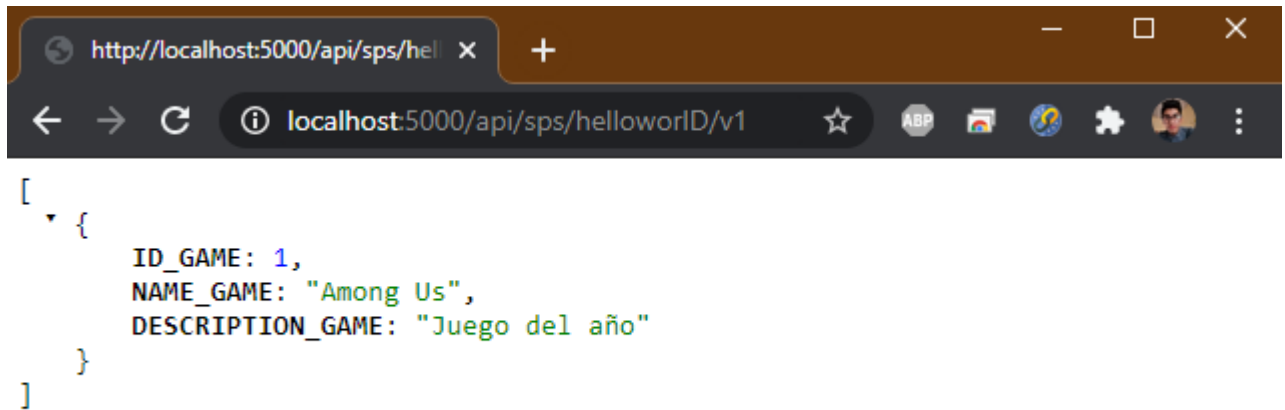
Body Cookies Headers (7) Test Results

Status: 200 OK Time: 46 ms Size: 293 B Save Response

Pretty Raw Preview Visualize JSON

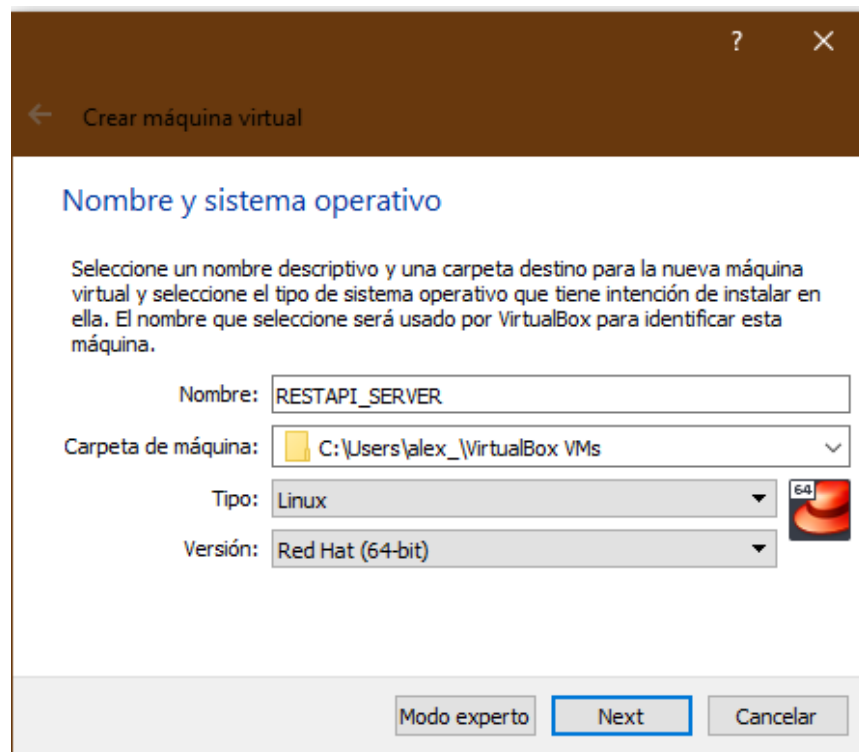
```
1 {
2   "message": "El juego ha sido eliminado de la BD"
3 }
```

Para verificar la eliminación del registro, lo podemos revisar en el navegador refrescando el sitio ó en el cliente REST.

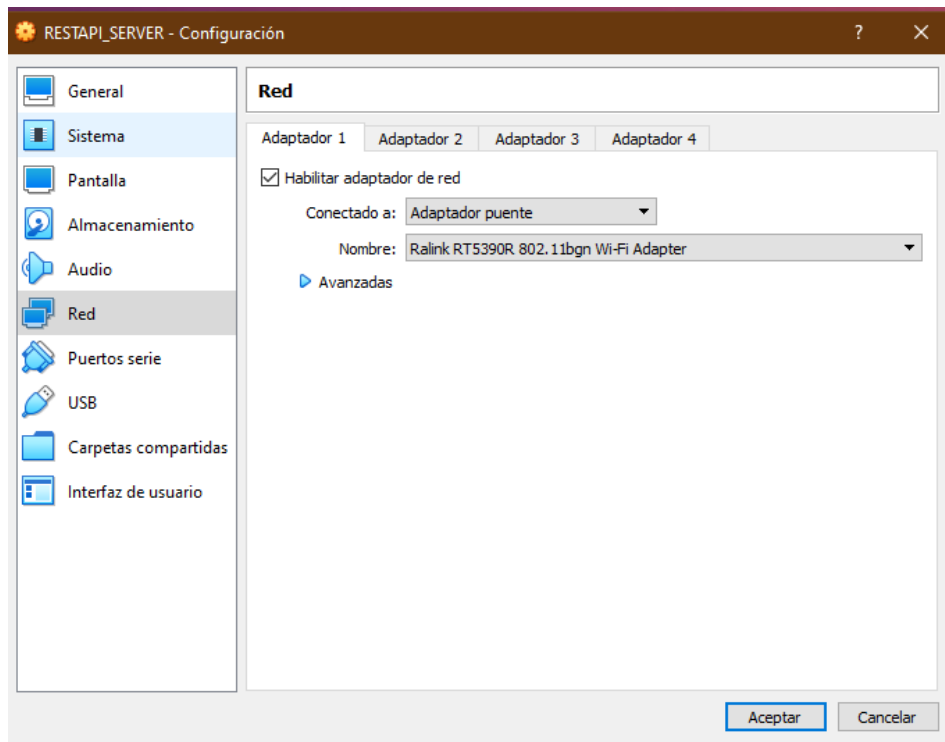


8. MV Centos 7

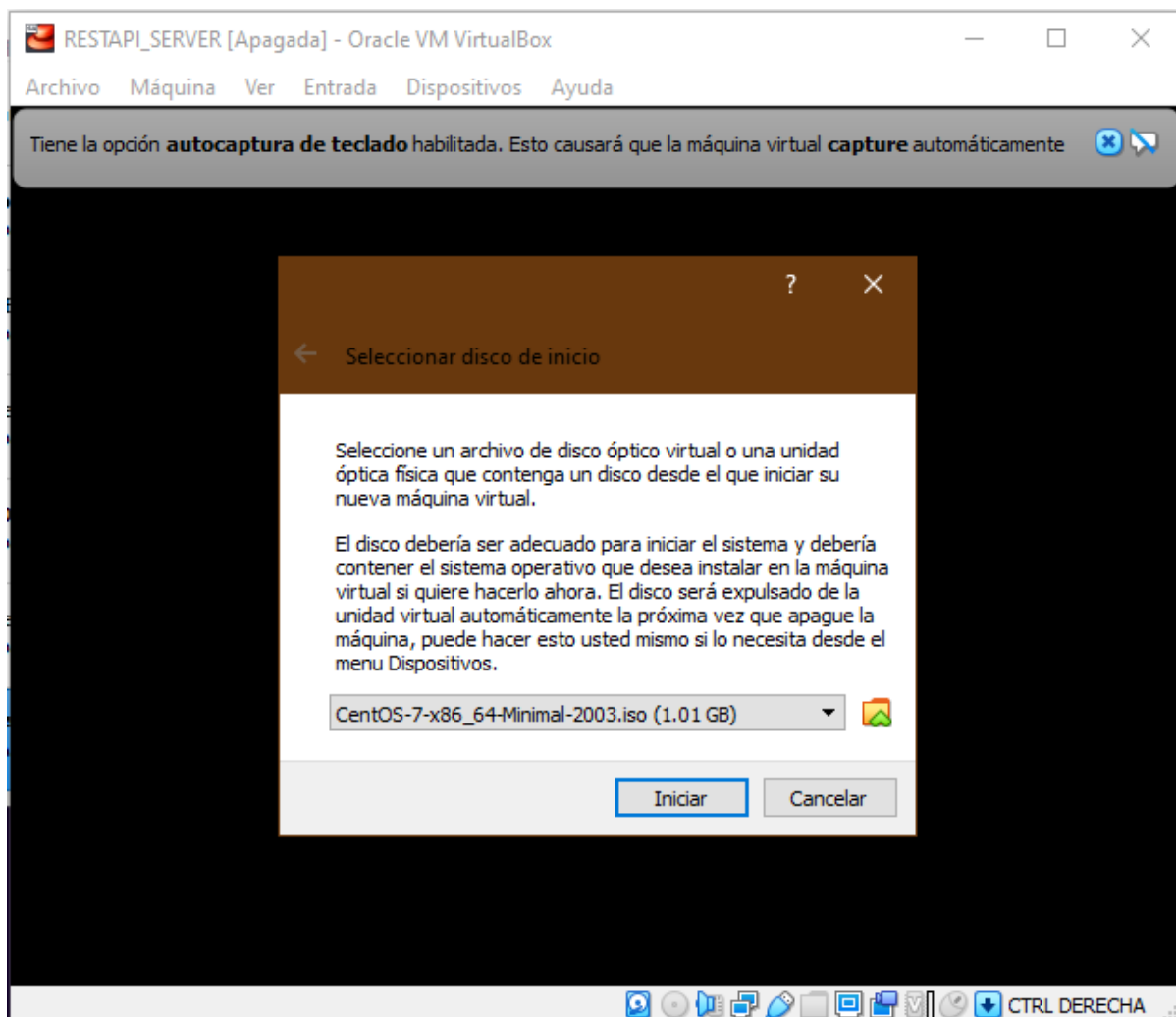
Para realizar el deploying del API REST, voy a crear una máquina virtual con Centos 7, para realizar la instalación de docker de una manera limpia y crear microservicios, uno para el API REST y otro para Mysql.

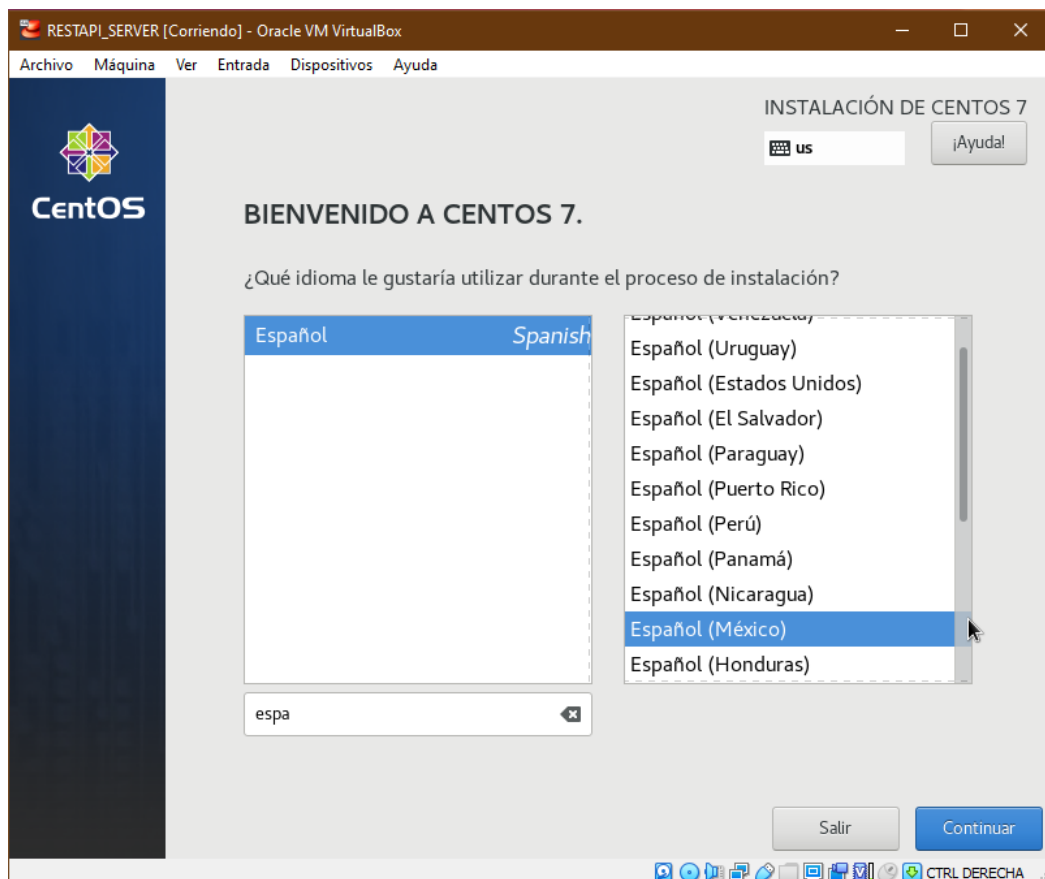
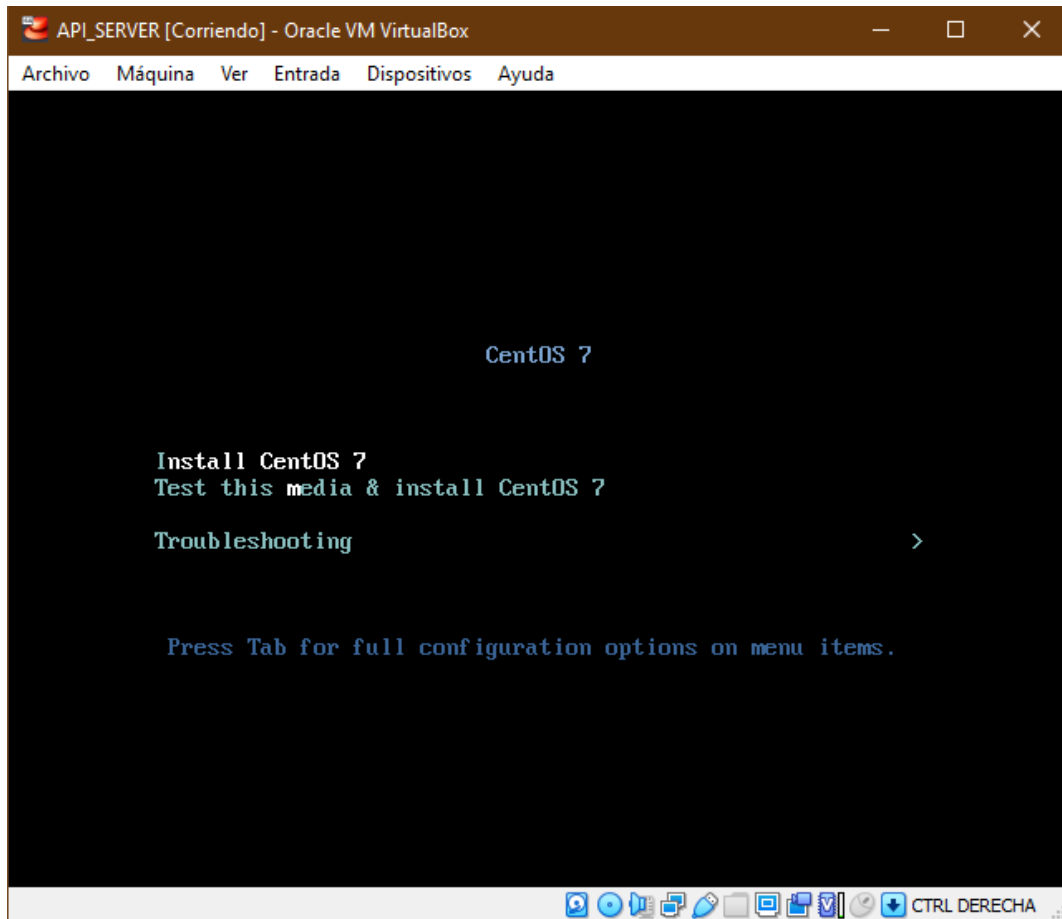


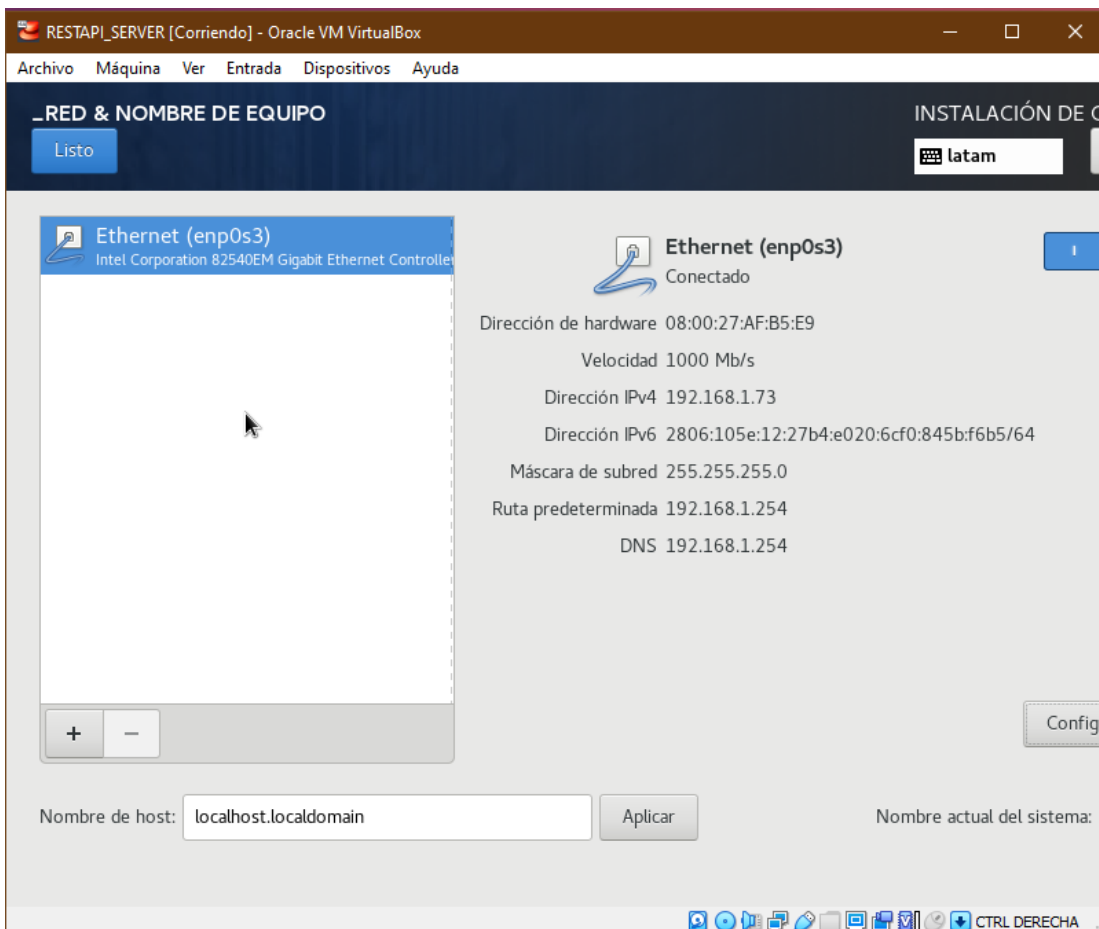
Configuro el adaptador de red a un adaptador puente, para tener acceso a internet y vincularlo con mi red local.

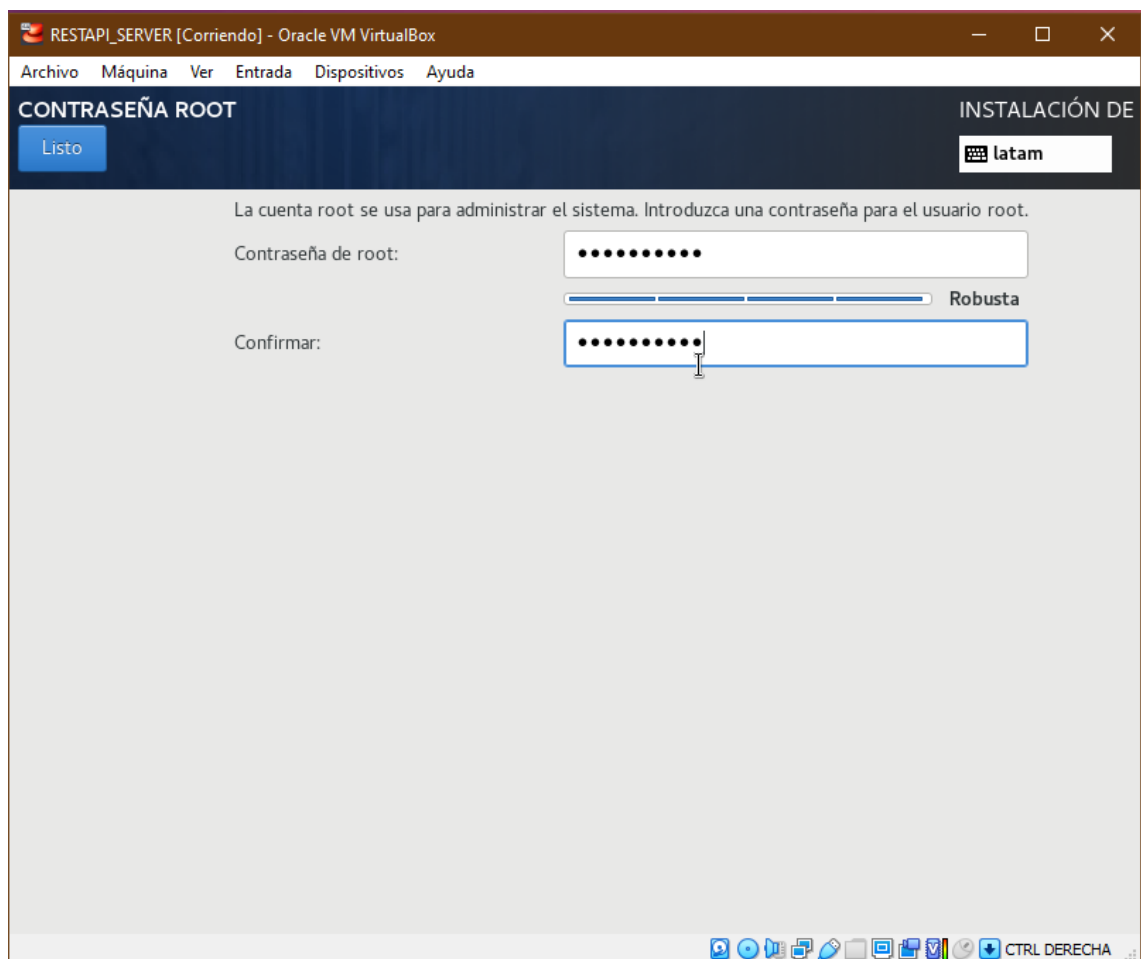
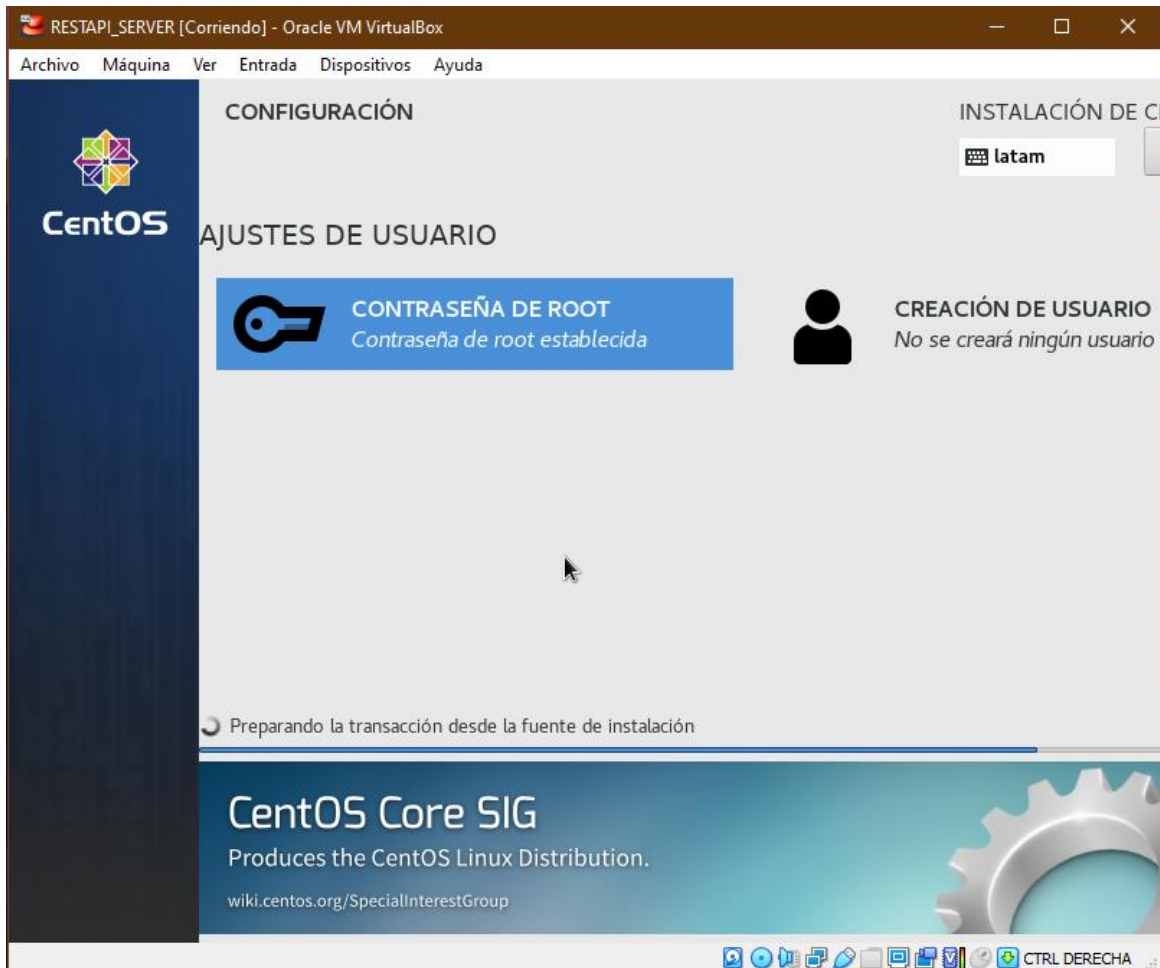


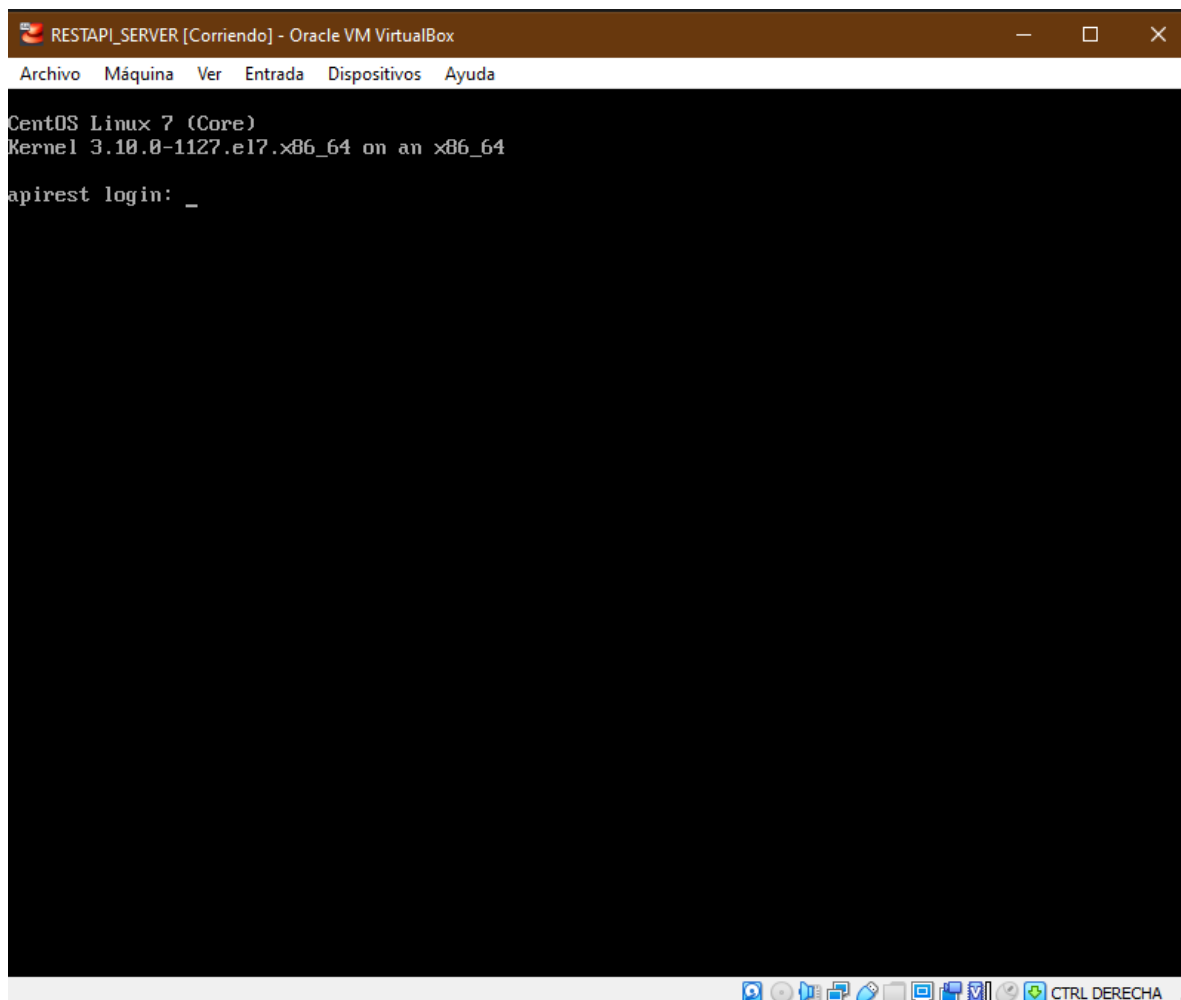
Al arrancar la MV, selecciono la imagen iso del SO.



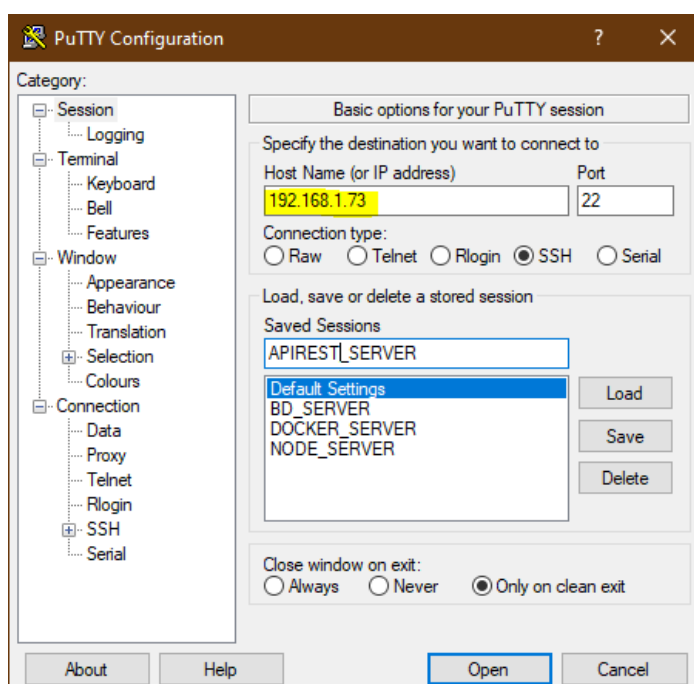








Para conectarme remotamente desde mi máquina física y realizar tareas de manera cómoda, utilizare el programa Putty para conectarme a la MV mediante SSH. Debo de indicarle la dirección IP de la MV.



Ya dentro de mi MV se debe seguir los siguientes pasos en la terminal.

1. Instalación de Docker

Instalar las dependencias

```
# yum install -y yum-utils device-mapper-persistent-data lvm2
```

Agregar el repositorio de Docker a CentOS

```
# yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

Instalar Docker

```
# yum install docker
```

Verificar la version de Docker

```
# docker version
```

Iniciar Docker

```
# systemctl start docker
```

Habilitar Docker

```
# systemctl enable docker
```

Verificar el estado del servicio

```
# systemctl status docker 0
```

2. Descarga del proyecto

El proyecto está en un repositorio de github, por lo tanto hay que descargarlo utilizando git.

Instalar git

```
# yum install git
```

Ahora de sebe de crear unca capeta en donde se van a guardar los archivos del proyecto. En este caso la carpeta se llamará github (Puede ser otro nombre), dentro del directorio home (Puede ser en otro directorio).

```
# cd /home  
# mkdir github
```

Dentro de la capeta creada, se descarga el proyecto con el comando

```
# git clone https://github.com/OWLSALEX/API_REST.git
```

Nota:

Se necesita de una cuenta de github para poder descargar el proyecto, ya que se pide las credenciales de autenticación.

3. Crear el contenedor de la base de datos

```
# docker run -p 3306:3306 --name BD_SERVER -e MYSQL_ROOT_PASSWORD=#Admin2020 -v BD_APIREST:/var/lib/mysql -d mysql:5.7
```

Con este comando estamos diciendo lo siguiente.

Inicia y crea el contenedor	docker run
En el puerto 3306 para conexiones remotas, utilizando el puerto por default de mysql	-p 3306:3306
Asignándole el nombre BD_SERVER a este contenedor	--name BD_SERVER
Estableciéndole una contraseña para el usuario root de MYSQL	-e MYSQL_ROOT_PASSWORD=#Admin2020
Creando un volumen en el equipo host, para persistir los datos, por si se llega a eliminar este contenedor	-v BD_AMOLLI:/var/lib/mysql
Finalmente, inicia el contenedor como un servicio con la versión 5.7 de mysql	-d mysql:5.7

Nota:

El primer puerto, se puede cambiar, ya que con ese puerto se accede remotamente al servidor de la base de datos, pero no se recomienda. La contraseña del usuario root, y/o el nombre del contenedor, también se puede cambiar, pero teniendo en cuenta que en el archivo keys.js se debe actualizar, ya que este contiene los datos para realizar la conexión del proyecto a la base de datos.

Entrar a la consola de mysql (Pedirá la contraseña del usuario root)


```
# docker exec -it BD_SERVER mysql -p
```

Crear la base de datos

```
mysql> CREATE DATABASE MI_API;
      USE MI_API;
      CREATE TABLE GAMES (
        ID_GAME          INT NOT NULL AUTO_INCREMENT,
        NAME_GAME         VARCHAR(30) NOT NULL,
        DESCRIPTION_GAME  VARCHAR(30) NOT NULL,
        PRIMARY KEY (ID_GAME)
      );
```

Salir de la consola

```
mysql> exit
```

```
root@apiest:/
f3cebc0b4691: Pull complete
1862755a0b37: Pull complete
489b44f3dbb4: Pull complete
690874f836db: Pull complete
baa8be383ffb: Pull complete
55356608b4ac: Pull complete
277d8f888368: Pull complete
21f2da6feb67: Pull complete
2c98f818bcb9: Pull complete
031b0a770162: Pull complete
Digest: sha256:14fd47ec8724954b63d1a236d2299b8da25c9bbb8eacc739bb88038d82da4919
Status: Downloaded newer image for docker.io/mysql:5.7
17ef3089ee52292a07ff576057c9ae6c1e806e4b132da266361911132d1c5421
[root@apiest /]# docker exec -it BD_SERVER mysql -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.31 MySQL Community Server (GPL)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE MI_API;
Query OK, 1 row affected (0.00 sec)

mysql>     USE MI_API;
Database changed
mysql>     CREATE TABLE GAMES (
->       ID_GAME          INT NOT NULL AUTO_INCREMENT,
->       NAME_GAME         VARCHAR(30) NOT NULL,
->       DESCRIPTION_GAME  VARCHAR(30) NOT NULL,
->       PRIMARY KEY (ID_GAME)
->     );
Query OK, 0 rows affected (0.02 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| MI_API |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)

mysql> exit
Bye
[root@apiest /]#
```

4. Modificar archivo keys.js

Acceder a los archivos del proyecto

```
# cd /home/github/API_REST/build
```

Modificar el archivo keys.js

```
# vi keys.js
```

Este archivo contiene los datos de conexión a la base de datos. Debe de estar de la siguiente manera:

```
database: {  
  host      : 'BD_SERVER', //Nombre del contenedor de la base de datos  
  user      : 'root',      //Usuario root  
  password  : '#Admin2020', //Contraseña del usuario root  
  database  : 'MI_API'     //Nombre de la base de datos  
}
```

5. Crear contenedor de Node JS

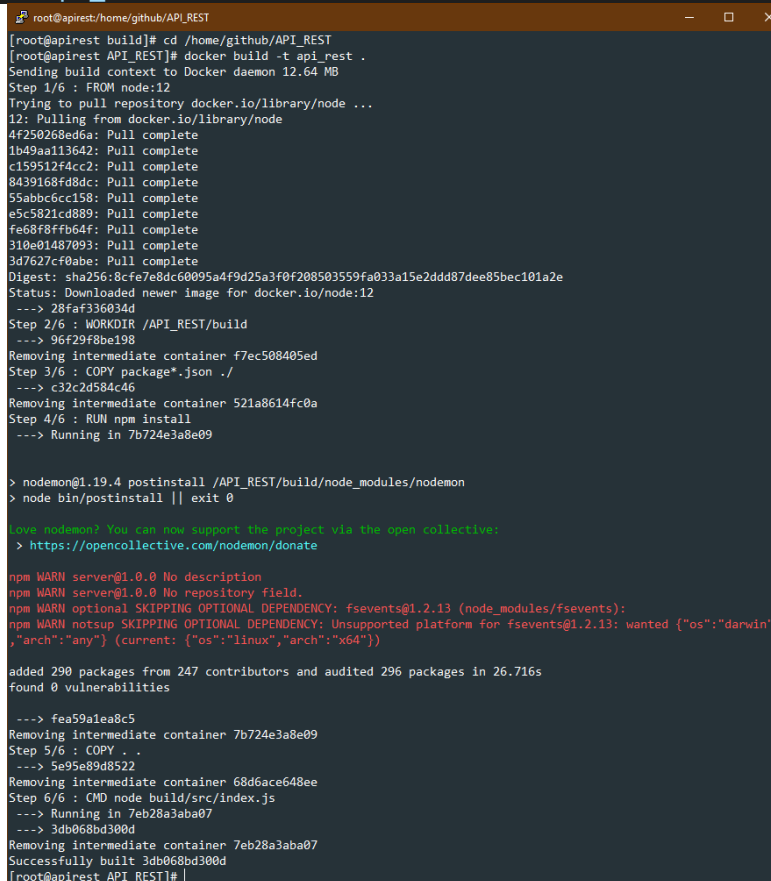
El proyecto contiene un archivo dockerfile, el cual se encuentra la configuración de instalación de las dependencias del proyecto, el cual se debe ejecutar.

Navegar hacia la carpeta del proyecto

```
# cd /home/github/API_REST
```

Ejecutar archivo docker file

```
# docker build -t api_rest .
```



```
root@apirest:/home/github/API_REST  
[root@apirest build]# cd /home/github/API_REST  
[root@apirest API_REST]# docker build -t api_rest .  
Sending build context to Docker daemon 12.64 MB  
Step 1/6 : FROM node:12  
Trying to pull repository docker.io/library/node ...  
12: Pulling from docker.io/library/node  
4f250268ed6a: Pull complete  
1b49aa113642: Pull complete  
c159512f4cc2: Pull complete  
8439168fd8dc: Pull complete  
55abbc6cc158: Pull complete  
e5c5821cd889: Pull complete  
fe68f8ffb64f: Pull complete  
310e01487093: Pull complete  
3d7627cf0abe: Pull complete  
Digest: sha256:8cfe7e8dc60095a4f9d25a3f0f208503559fa033a15e2ddd87dee85bec101a2e  
Status: Downloaded newer image for docker.io/node:12  
----> 28faf336034d  
Step 2/6 : WORKDIR /API_REST/build  
----> 96f29f8be198  
Removing intermediate container f7ec588405ed  
Step 3/6 : COPY package*.json ./  
----> c32c2d584c46  
Removing intermediate container 521a8614fc0a  
Step 4/6 : RUN npm install  
----> Running in 7b724e3a8e09  
  
> nodemon@1.19.4 postinstall /API_REST/build/node_modules/nodemon  
> node bin/postinstall || exit 0  
  
Love nodemon? You can now support the project via the open collective:  
> https://opencollective.com/nodemon/donate  
  
npm WARN server@1.0.0 No description  
npm WARN server@1.0.0 No repository field.  
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules/fsevents):  
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin",  
"arch":"any"} (current: {"os":"linux","arch":"x64"})  
  
added 290 packages from 247 contributors and audited 296 packages in 26.716s  
found 0 vulnerabilities  
  
----> fea59a1ea8c5  
Removing intermediate container 7b724e3a8e09  
Step 5/6 : COPY . .  
----> 5a95a89d8522  
Removing intermediate container 68d6ace648ee  
Step 6/6 : CMD node build/src/index.js  
----> Running in 7eb28a3aba07  
----> 3db068bd300d  
Successfully built 3db068bd300d  
[root@apirest API_REST]#
```

Ejecuta el contenedor del proyecto como un proceso

```
# docker run -d -p 8090:5000 --link BD_SERVER --name API_REST api_rest
```

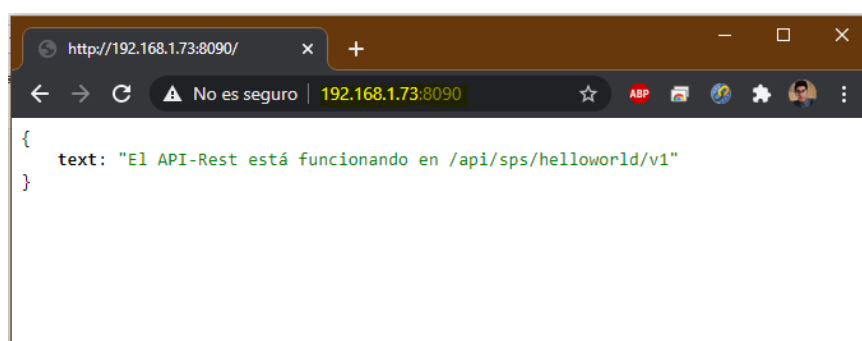
Inicia y crea el contenedor	docker run
Como un proceso	-d
En el puerto 8090 para conexiones remotas, utilizando el puerto por default del proyecto	8090:5000
Vinculándolo con el contenedor BD_SERVER	--Link BD_SERVER
Y asígnale el nombre API_REST a este contenedor	--name API_REST
Con ayuda de la imagen api_rest	api_rest

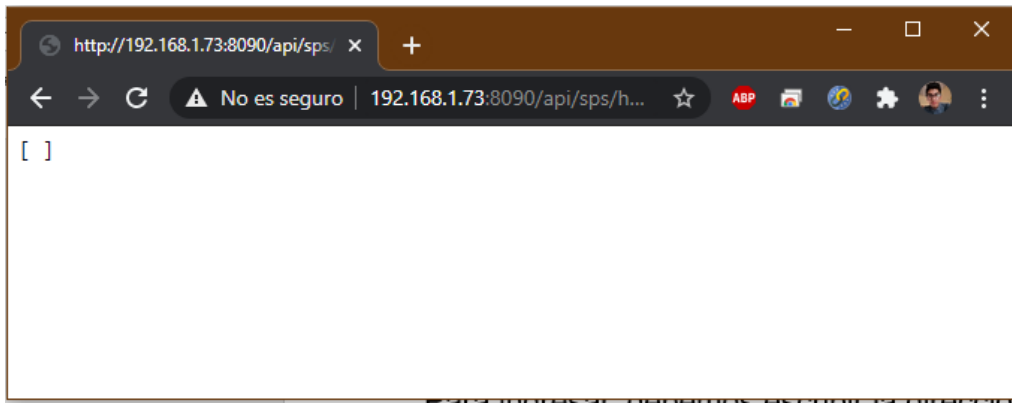
Nota:

El puerto 8090 se puede cambiar, según las necesidades. Este puerto sirve para acceder al proyecto desde cualquier equipo en la red, escribiendo la dirección IP del equipo host junto con el puerto. El puerto 5000, es el puerto que utiliza el proyecto para levantar el servidor de Node, por lo tanto no es recomendable cambiar.

```
root@api-rest:/home/github/API_REST
[root@api-rest API_REST]# docker run -d -p 8090:5000 --link BD_SERVER --name API_REST api_rest
bc16c053abc0943bbf41d4d44ab6cc4776061127cb8c854d1e51f605e3ab4175
[root@api-rest API_REST]#
```

Para ingresar, debemos escribir la dirección IP de la MV junto con el puerto, en este caso es <http://192.168.1.73:8090/>.

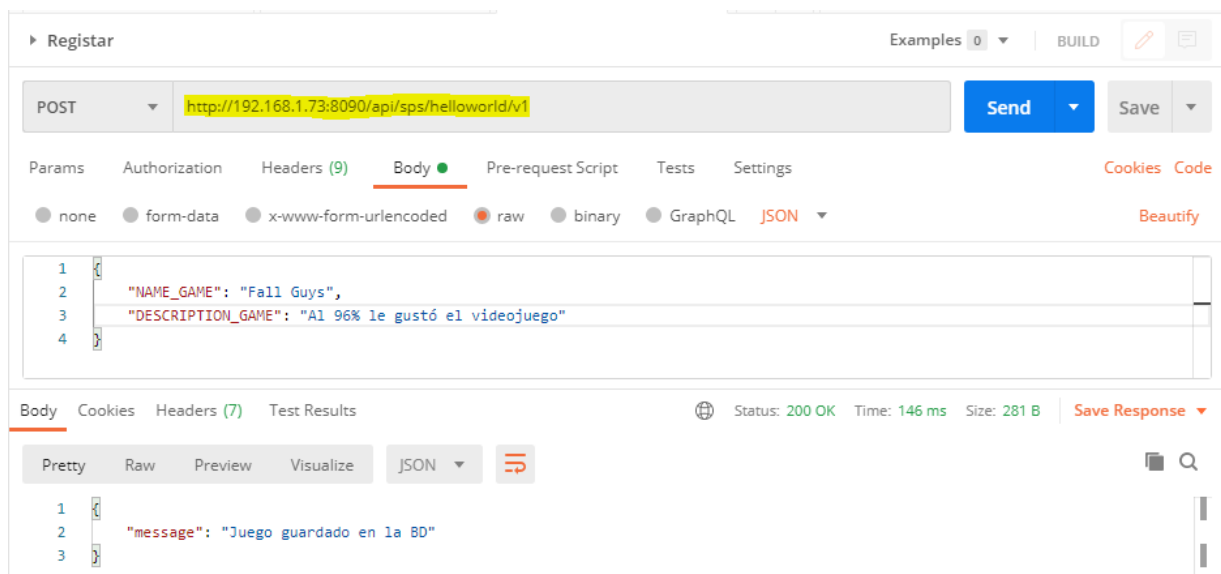




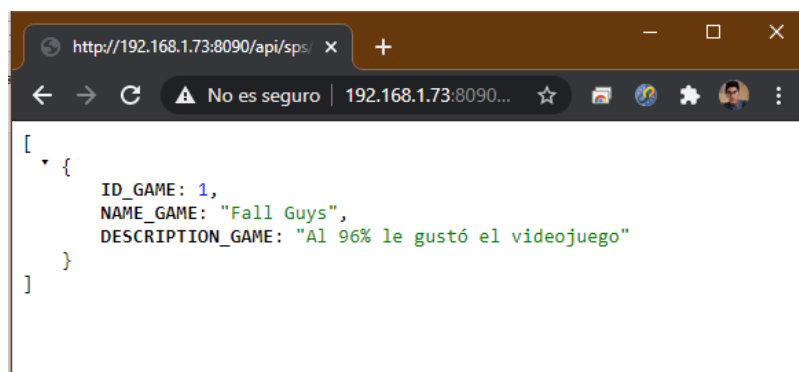
Para ingresar realizar el CRUD con un cliente REST, repetiríamos los pasos 4,5,6 y 7, cambiando la URL por: <http://192.168.1.73:8090/api/sps/helloworld/v1>

6. Registrar datos

Vamos a registrar un nuevo juego llamado Fall Guys. En Postman, realizamos lo siguiente.



Para validar este proceso, refrescamos el navegador u obtenemos los datos desde Postman.



► Listar

Examples 0 BUILD

GET `http://192.168.1.73:8090/api/sps/helloworld/v1` Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results Status: 200 OK Time: 49 ms Size: 335 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "ID_GAME": 1,
4     "NAME_GAME": "Fall Guys",
5     "DESCRIPTION_GAME": "Al 96% le gustó el videojuego"
6   }
7 }
```

Recordemos, que la información se está guardando en la base de datos de la MV.
Vamos a ingresar un juego nuevo llamado Call of Duty: Mobile.

► Registrar

Examples 0 BUILD

POST `http://192.168.1.73:8090/api/sps/helloworld/v1` Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "NAME_GAME": "Call of Duty: Mobile",
3   "DESCRIPTION_GAME": "Juego gratuito tipo shooter"
4 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 129 ms Size: 281 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Juego guardado en la BD"
3 }
```

► Listar

Examples 0 BUILD

GET `http://192.168.1.73:8090/api/sps/helloworld/v1` Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results Status: 200 OK Time: 27 ms Size: 434 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "ID_GAME": 1,
4     "NAME_GAME": "Fall Guys",
5     "DESCRIPTION_GAME": "Al 96% le gustó el videojuego"
6   },
7   {
8     "ID_GAME": 2,
9     "NAME_GAME": "Call of Duty: Mobile",
10    "DESCRIPTION_GAME": "Juego gratuito tipo shooter"
11   }
12 }
```

7. Mostrar

Vamos a mostrar la información del juego con el ID 1

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** `http://192.168.1.73:8090/api/sps/helloworld/v1/1`
- Buttons:** Send, Save
- Tabs:** Params, Authorization, Headers (9), Body, Pre-request Script, Tests, Settings. The 'Body' tab is selected.
- Query Params Table:**

KEY	VALUE	DESCRIPTION
Key	Value	Description
- Response Status:** 200 OK, Time: 51 ms, Size: 333 B
- Response Body (JSON):**

```
{  "ID_GAME": 1,  "NAME_GAME": "Fall Guys",  "DESCRIPTION_GAME": "Al 96% le gustó el videojuego"}
```

8. Actualizar

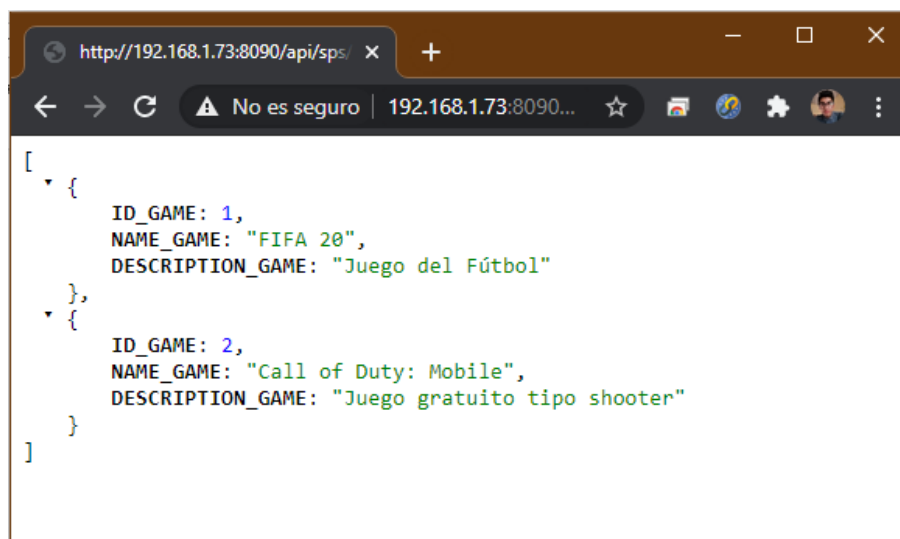
Vamos a modificar completamente la información del primer registro.

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** `http://192.168.1.73:8090/api/sps/helloworld/v1/1`
- Buttons:** Send, Save
- Tabs:** Params, Authorization, Headers (9), Body, Pre-request Script, Tests, Settings. The 'Body' tab is selected.
- Body Type:** raw
- Request Body:**

```
{  "NAME_GAME": "FIFA 20",  "DESCRIPTION_GAME": "Juego del Fútbol"}
```
- Response Status:** 200 OK, Time: 133 ms, Size: 295 B
- Response Body (JSON):**

```
{  "message": "El juego ha sido actualizado en la BD"}
```



9. Eliminar

Vamos a eliminar el segundo registro.

Eliminar Examples 0 BUILD

DELETE Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results Status: 200 OK Time: 130 ms Size: 293 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "El juego ha sido eliminado de la BD"
3 }
```

Listar Examples 0 BUILD

GET Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results Status: 200 OK Time: 21 ms Size: 320 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "ID_GAME": 1,
3   "NAME_GAME": "FIFA 20",
4   "DESCRIPTION_GAME": "Juego del Fútbol"
5 }
6
7 }
```

10. Extras

Visualizar contenedores

```
# docker ps
```

```
root@api-rest:~
[root@api-rest ~]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
5f39e60adb28   api_rest      "docker-entrypoint..." 59 minutes ago Up 59 minutes 0.0.0.0:8090->5000/tcp             API_REST
17ef3089ee52   mysql:5.7     "docker-entrypoint..." About an hour ago Up About an hour 0.0.0.0:3306->3306/tcp, 33060/tcp  BD_SERVER
```

Visualizar contenedores activos

```
# docker ps -a
```

```
root@api-rest:~
[root@api-rest ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
5f39e60adb28   api_rest      "docker-entrypoint..." About an hour ago Up About an hour 0.0.0.0:8090->5000/tcp             API_REST
17ef3089ee52   mysql:5.7     "docker-entrypoint..." About an hour ago Up About an hour 0.0.0.0:3306->3306/tcp, 33060/tcp  BD_SERVER
```

Detener contenedor

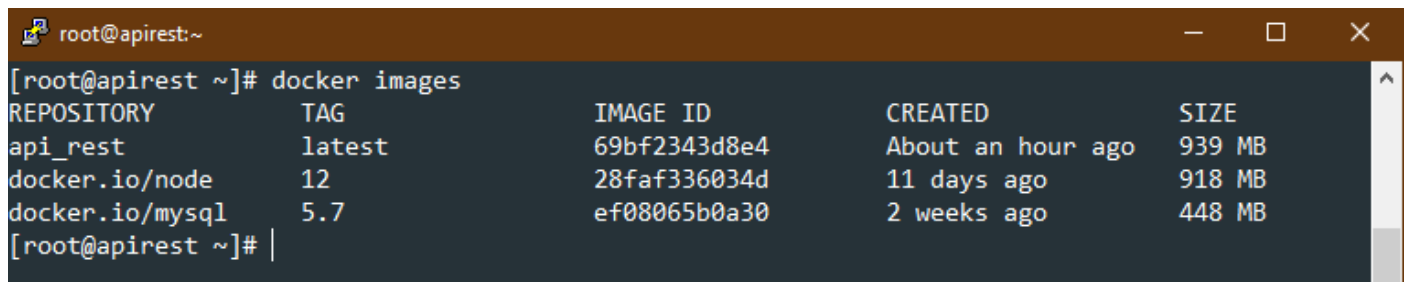
```
# docker stop API_REST
```

Eliminar contenedor

```
# docker rm API_REST
```

Mostrar imágenes

```
# docker images
```



```
root@api-rest:~  
[root@api-rest ~]# docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
api_rest             latest             69bf2343d8e4       About an hour ago  939 MB  
docker.io/node       12                 28faf336034d       11 days ago        918 MB  
docker.io/mysql      5.7                ef08065b0a30       2 weeks ago        448 MB  
[root@api-rest ~]#
```

Eliminar Imagen

```
# docker image rm ID
```

Me base el lo siguientes videos

<https://www.youtube.com/watch?v=NVvZNMfgg6M>

<https://www.youtube.com/watch?v=iLImm0L-VpQ>

<https://www.youtube.com/watch?v=AwZrEQaaYcA>

11.Docker Hub

docker pull mysql

Imagen de mysql

docker pull node

Imagen de node js

docker pull alexae01/api_rest

Imagen del proyecto API_REST