



# **Práctica REST API y Contenedores**

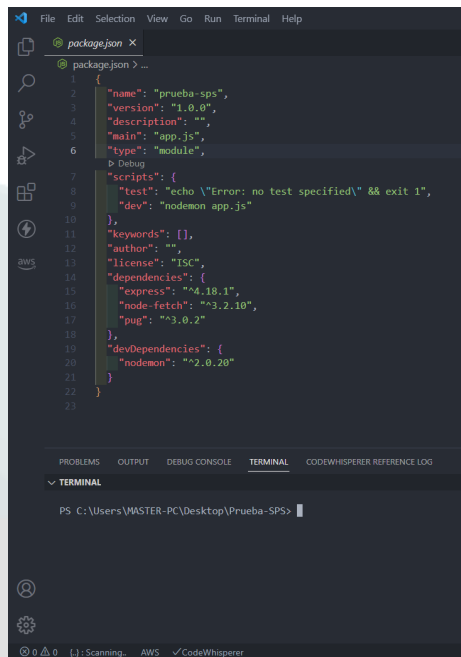
Dev

# Como primer paso inicie el proyecto con nodeJS



Img. 1 Comando para iniciar el proyecto con nodeJS

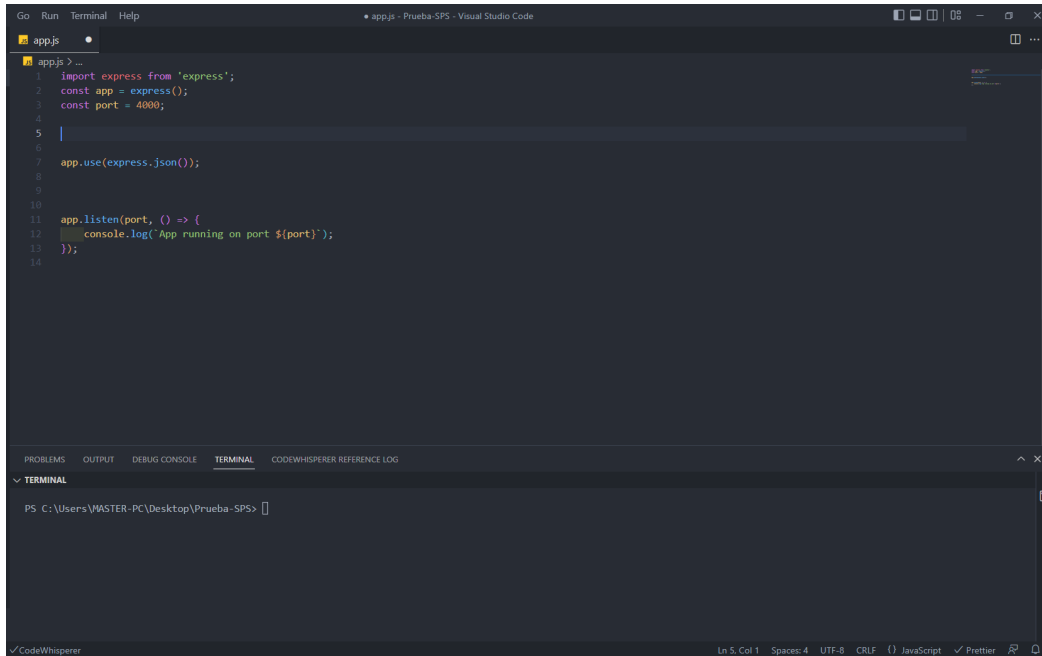
## Se crea el archivo “package.json”



Img. 2 Estructura del archivo “package.json”

Este archivo contiene las dependencias de nuestro proyecto así como también los scripts de ejecución que este puede correr.

# Creamos el servidor en un archivo llamado “app.js”



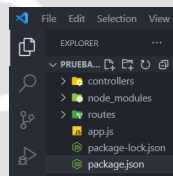
The screenshot shows the Visual Studio Code editor with a file named `app.js` open. The code in the editor is as follows:

```
1 import express from 'express';
2 const app = express();
3 const port = 4000;
4
5
6
7 app.use(express.json());
8
9
10
11 app.listen(port, () => {
12   console.log(`App running on port ${port}`);
13 });
14
```

Below the editor, the TERMINAL panel is visible, showing the command prompt at `PS C:\Users\WASTER-PC\Desktop\Prueba-SPS>`. The status bar at the bottom indicates the file is at `Ln 5, Col 1`, uses `Spaces: 4`, `UTF-8` encoding, `CRLF` line endings, and is a `JavaScript` file with `Prettier` formatting.

Img. 3 Creación del servidor corriendo en el puerto 4000

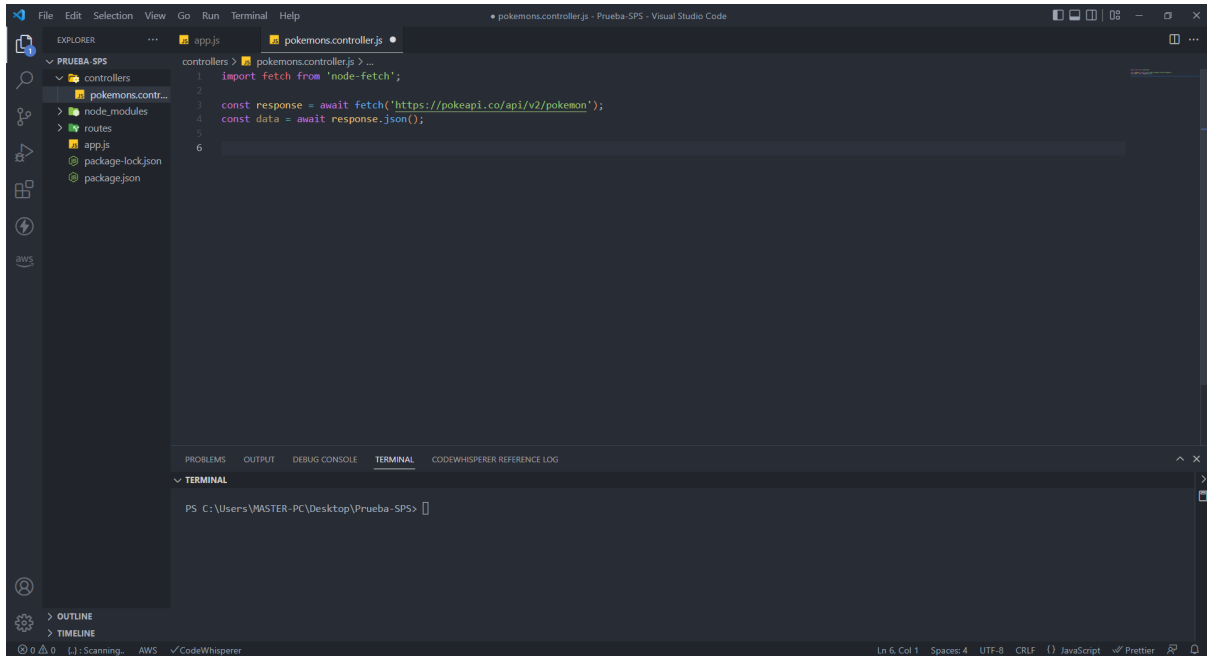
## Estructura de carpetas



Img. 4 Estructura de carpetas

Inicialmente se pensó en un patrón de diseño MVC. Al carecer de modelos y de vistas solo se mantuvo la carpeta de controladores y la carpeta de rutas.

# Construyendo el controlador de los pokemons



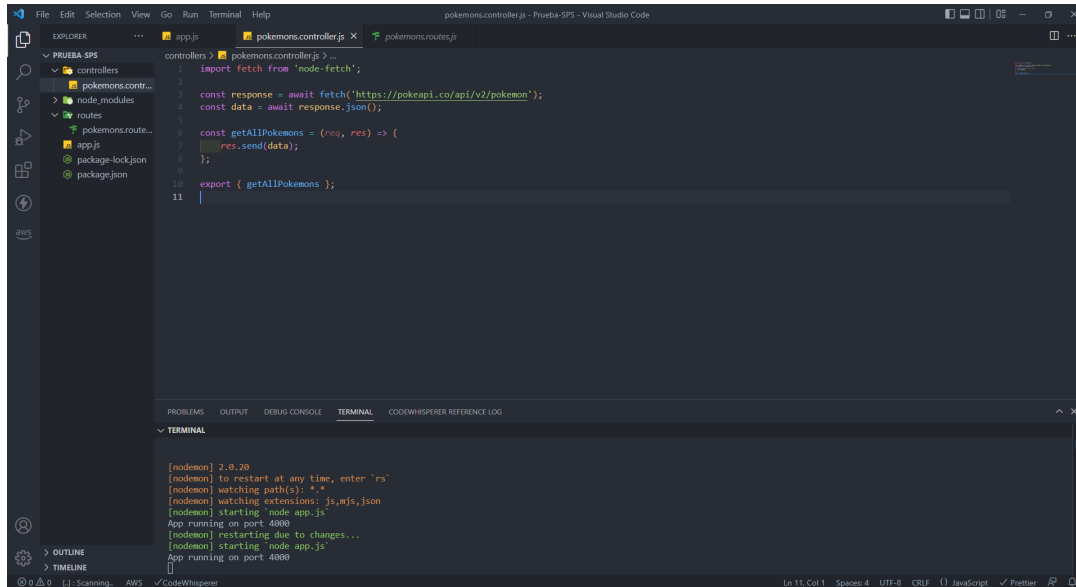
Img. 5 Consumo de la API de Pokemon y transformándola a formato JSON

Se importó la librería de “node-fetch” para poder realizar la llamada a la API con fetch.

**Nota: Primer problema enfrentado:** Se trabajó con CommonJS por el hecho de que node-fetch ya no funcionaba con los módulos convencionales de node (require).

Al ser esto necesario todo el proyecto se trabajó de esa manera, ya que no se puede trabajar de las dos de manera simultánea.

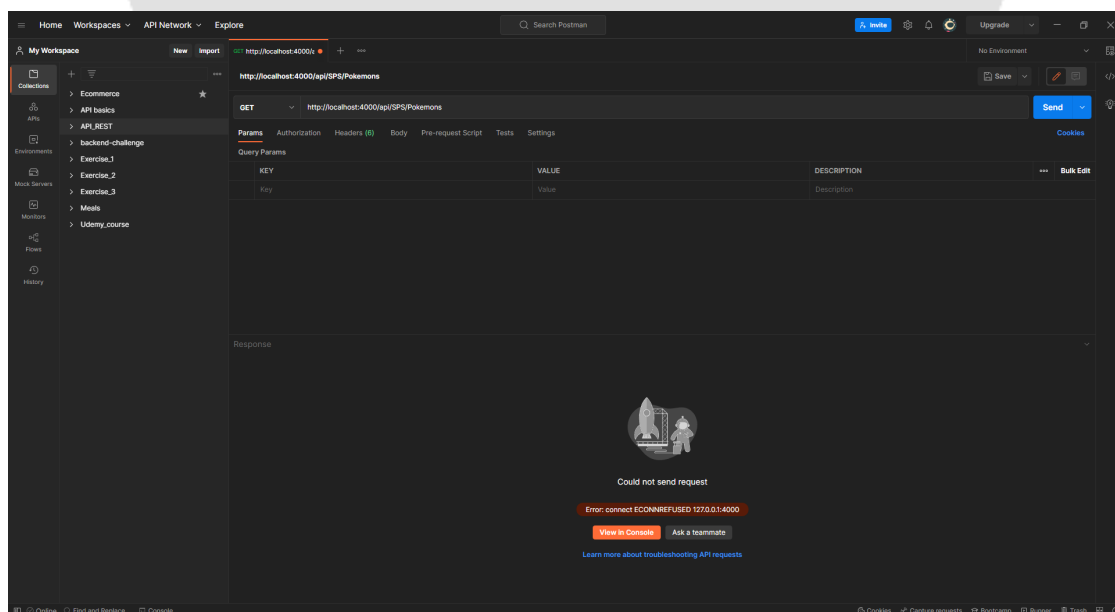
# Construcción del controlador que envía en la respuesta a los pokemons



Img. 5 Creación del controlador y envío de la respuesta

Como la respuesta de la llamada a la API ya se tenía en la variable “data” solo hizo falta regresarla al cliente REST por medio de una función y la exportamos.

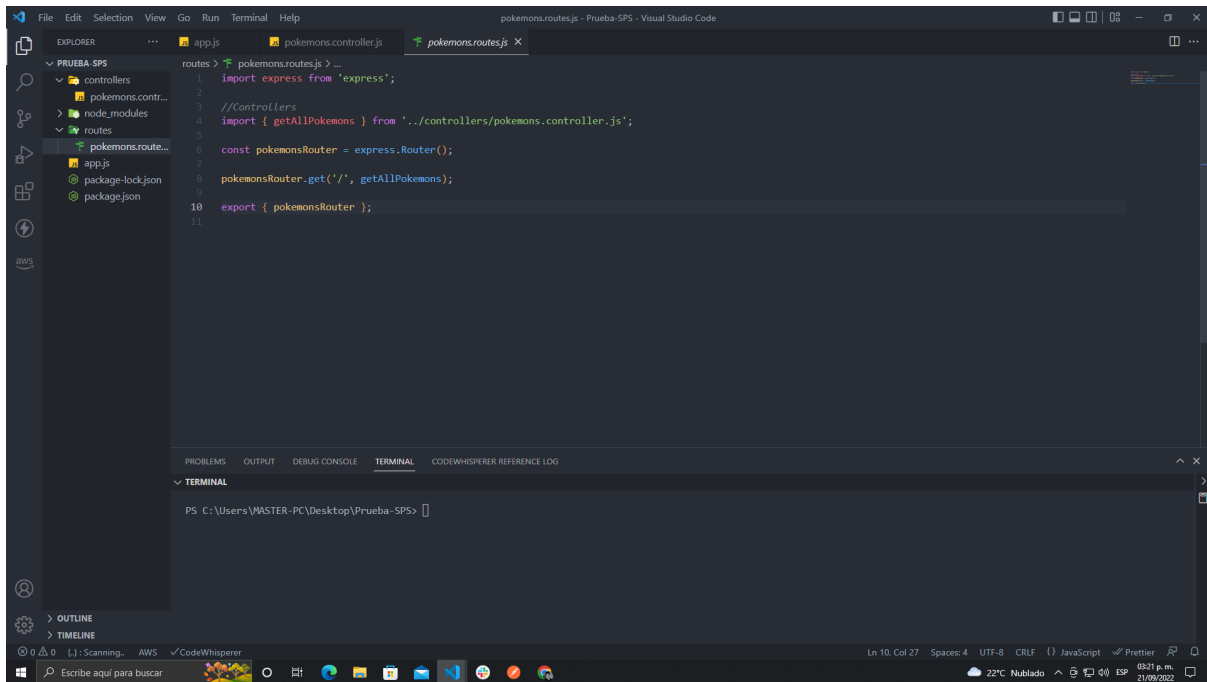
## Postman como cliente



Img. 6 Interfaz de peticiones con Postman

Como cliente se utilizó Postman para la prueba de hacia nuestra API.

# Creación de la ruta a la cual acceder para poder mostrar los datos en formato JSON

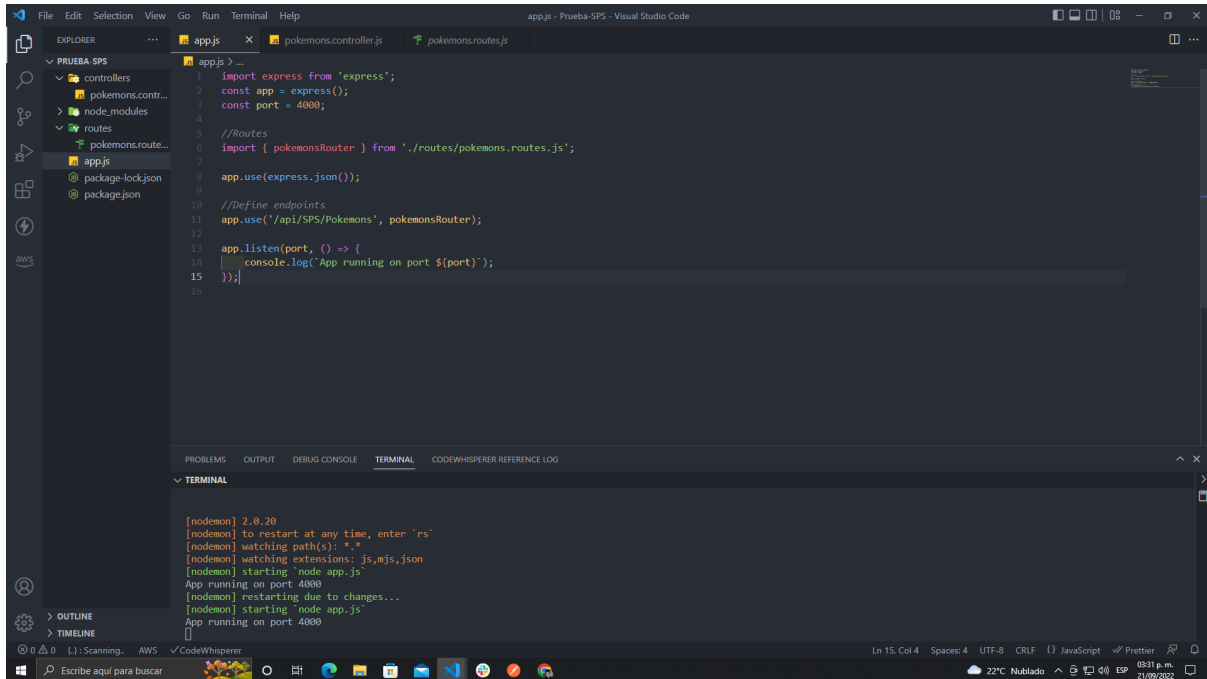


Img. 7 Creación de la ruta (endpoint) de nuestra REST API

Para esta ruta se importó a “express” que es el framework por excelencia que trabaja con nodeJS.

Se importó también el controlador que nos envía la respuesta de la llamada a la API de Pokemon, se crea el Router y posteriormente se asigna el endpoint por default, se ejecuta el controlador y la exportamos.

# Definimos el endpoint en el archivo “app.js” con la respectiva ruta



The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows a project named 'PRUEBA SPS' with a folder 'routes' containing 'pokemons.routes.js'. The main editor window shows the 'app.js' file with the following code:

```
1 import express from 'express';
2 const app = express();
3 const port = 4000;
4
5 //Routes
6 import { pokemonsRouter } from './routes/pokemons.routes.js';
7
8 app.use(express.json());
9
10 //Define endpoints
11 app.use('/api/SPS/Pokemons', pokemonsRouter);
12
13 app.listen(port, () => {
14   console.log(`App running on port ${port}`);
15 });
```

The terminal at the bottom shows the output of the application running:

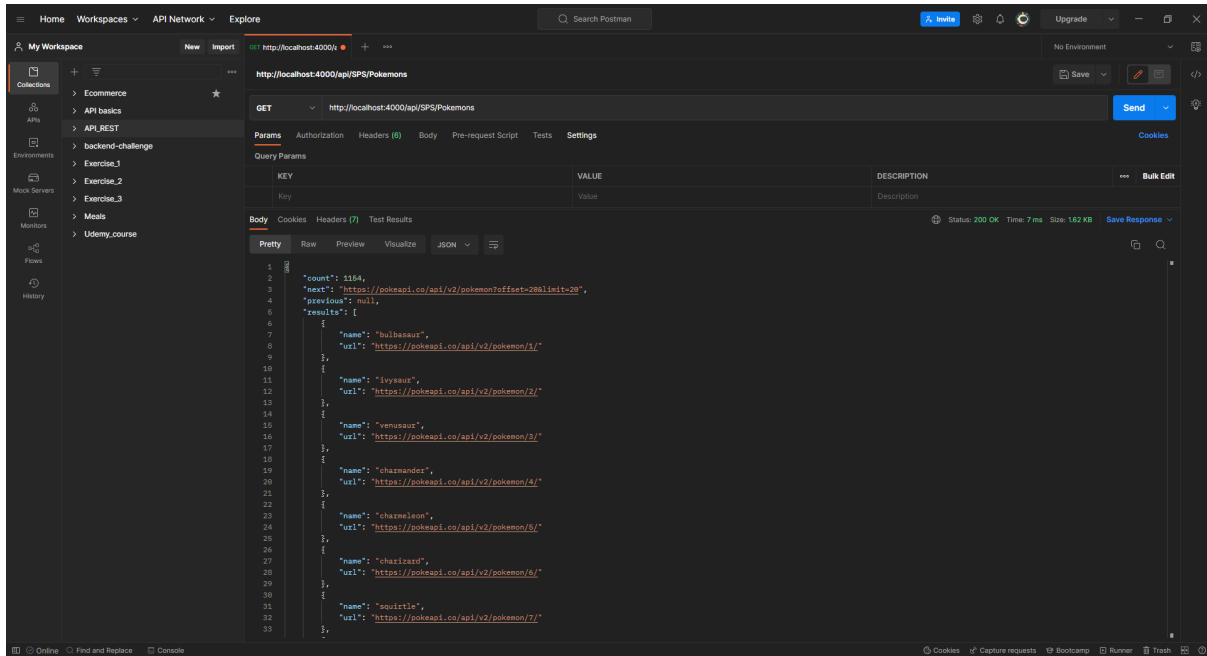
```
[nodemon] 2.0.20
[nodemon] to restart at any time, enter "rs"
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node app.js'
App running on port 4000
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
App running on port 4000
```

Img. 8 Definiendo el endpoint y asignando el router correspondiente

Ya solo hacía falta definir el endpoint al que se debe acceder para poder traer la respuesta desde nuestro servidor

Dev

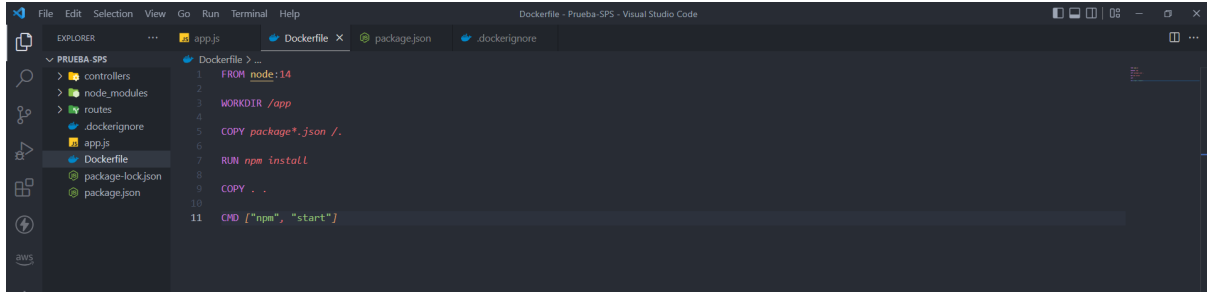
# Respuesta en formato JSON con los detalles de los pokemons



Img. 9 Respuesta de nuestro servidor en formato JSON



# Comenzando con el contenedor para nuestra REST API



Img. 10 Configuración del contenedor

Ejecutamos los comandos necesarios con los pasos a seguir para crear nuestra imagen

“**FROM node:14**” es un comando para especificar la versión que nuestro proyecto necesita. Al ser un proyecto sencillo con la versión 14 está perfecto.

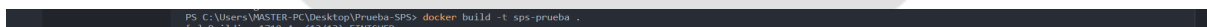
“**WORKDIR /app**” nos crea un directorio de trabajo y lleva el nombre de “app”

“**COPY**” se usa para copiar todos los archivos que comiencen por la palabra “package” y estos se usan para verificar que librerías son necesarias para que el proyecto funcione dentro de un contenedor.

“**RUN npm install**” nos ayuda a instalar esas librerías.

“**CMD [“npm”, “start”]**” ejecuta el script para iniciar el proyecto.

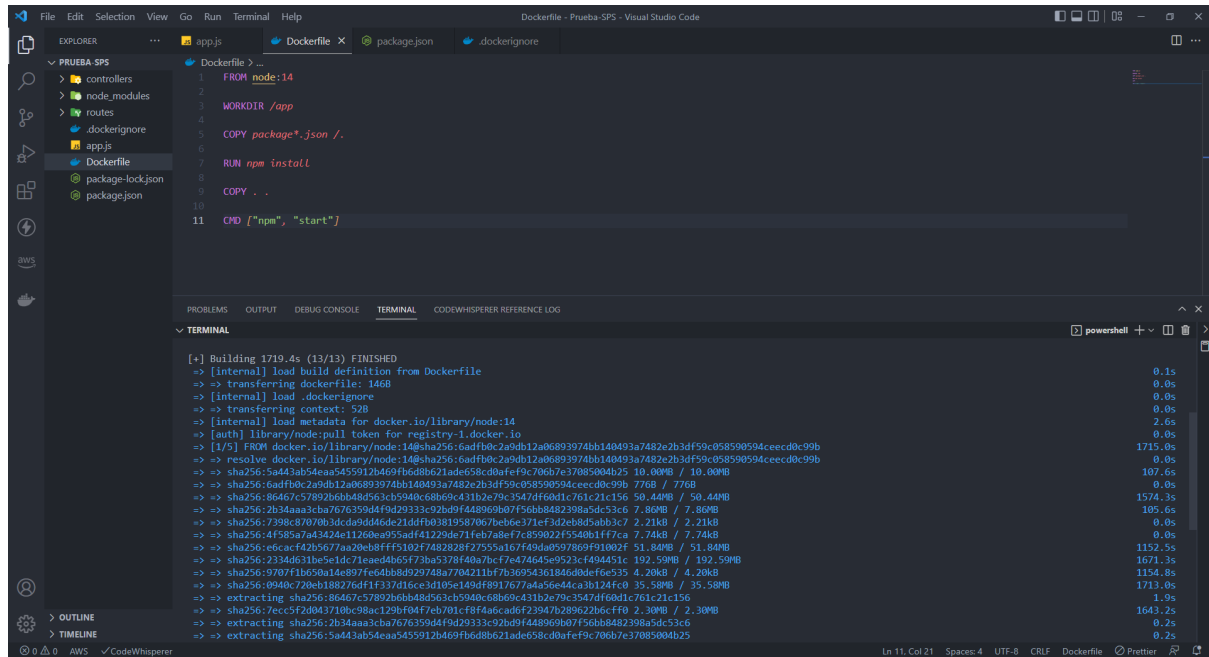
## Creando la imagen y descargando todos los



Img. 11 Comando para la creación de la imagen

Al correr este comando lo que se hace es crear una nueva imagen con el nombre de “sps-prueba” y con el punto al final agregamos todos los archivos que contiene este proyecto con todas las configuraciones antes realizadas.

Esto comenzará a descargar todos los requerimientos necesarios para que el proyecto funcione correctamente.



Img. 12 Descargando todas las configuraciones de la imagen

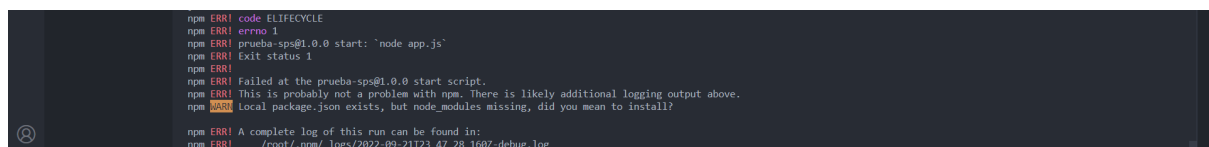
## Verificando que la imagen se haya creado

PS C:\Users\WASTER-PC\Desktop\Prueba-SPS> docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sp-s-prueba	latest	b54116cacab4	31 seconds ago	930MB
<none>	<none>	a01d3fd90472	33 minutes ago	915MB

Img. 13 Imagen creada correctamente

En esta parte ejecutamos el comando “docker images” para corroborar que nuestra imagen se ha creado correctamente.

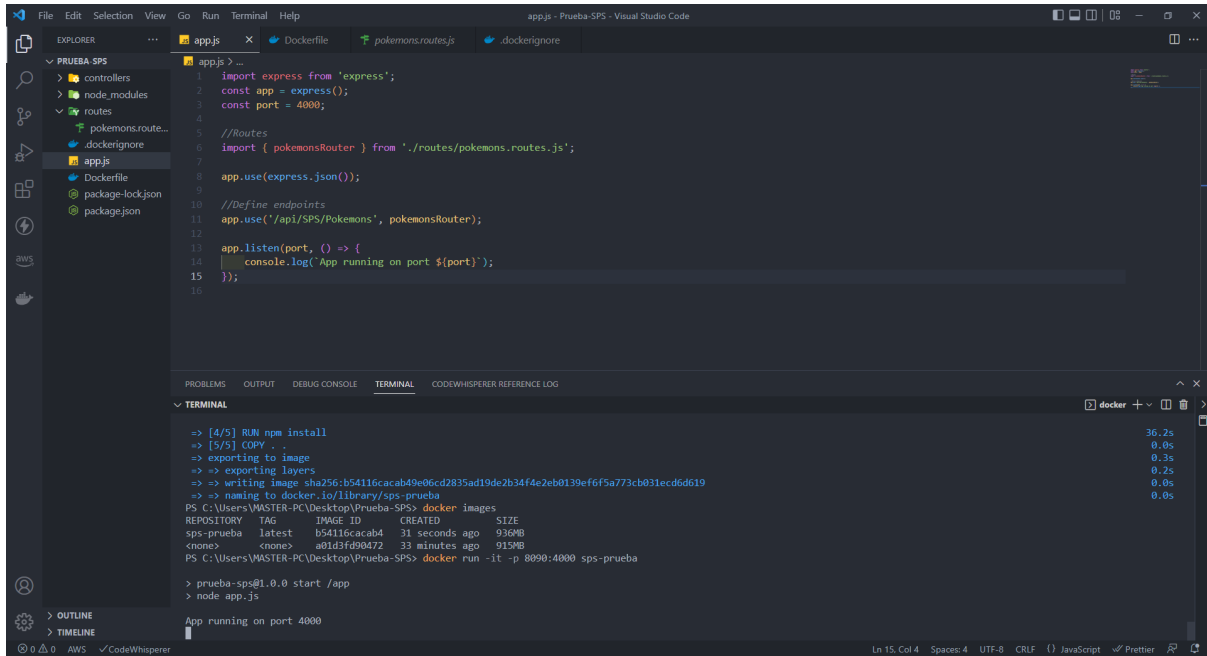
**Nota:** Un problema que se presentó fue no estaba escrito el directorio de /app correctamente el cual nos mostró el siguiente error



Img. 14 Error al ejecutar el comando para poder ejecutar nuestro proyecto en el puerto 8090 dentro de ese contenedor

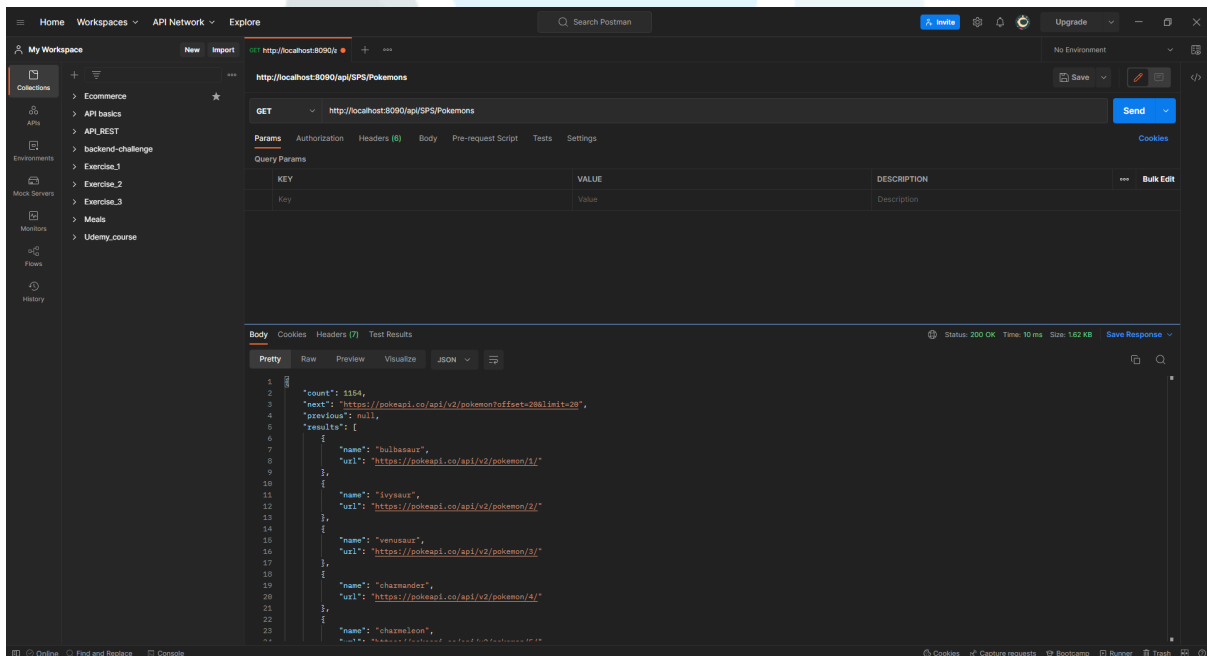
Investigando un poco se llegó a la conclusión de que fue un error ortográfico a la hora de crear el directorio /app (cosa que casi no pasa en el mundo de la programación)

# Corriendo el proyecto en el puerto 8090



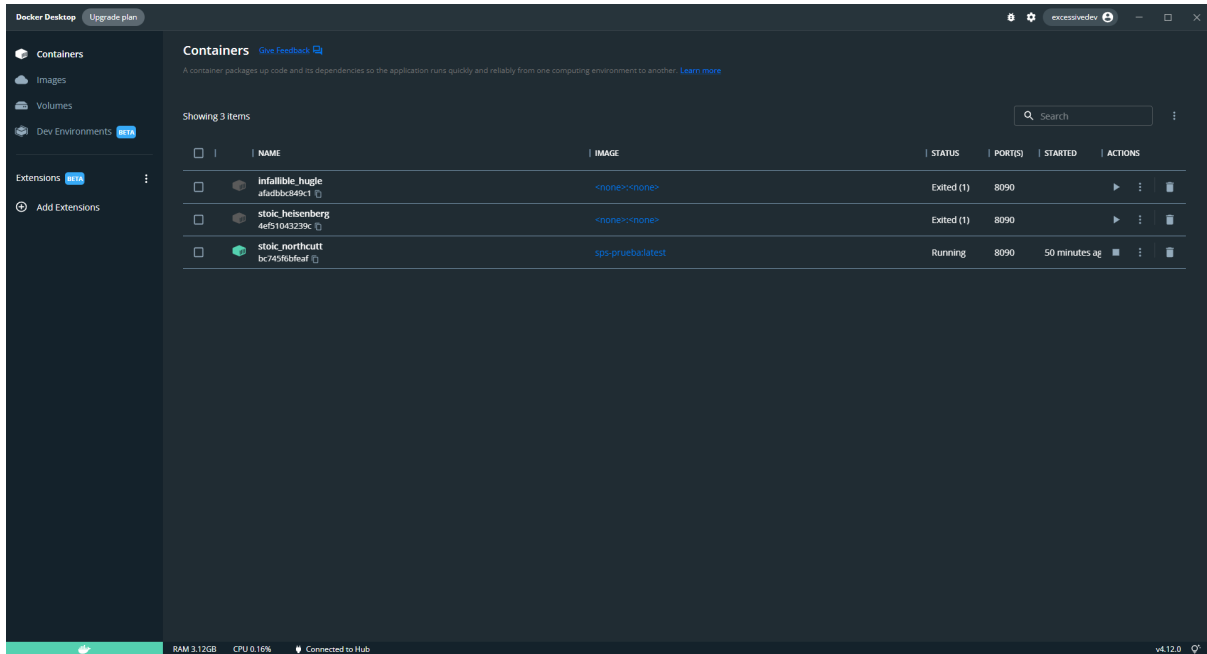
Img. 15 Proyecto funcionando correctamente

Una vez el contenedor está listo podemos acceder a él ahora a través del puerto 8090



Img. 16 Proyecto corriendo ahora a través del puerto asignado (8090)

# Y nuestro proyecto está listo y funcionando



Img. 17 proyecto corriendo y funcionando

!!!!Brooooo!!! !!!!Funciona!!!! !!!!Funciona!!!!

Esta es mi prueba para el proceso de selección en SPSolutions.

Agradezco infinitamente a Karen Vanegas por tomarme en cuenta para participar por la vacante, definitivamente no creí que funcionara, pero uno nunca sabe de lo que es capaz hasta que no lo pone a prueba.

Jamás había consumido una API con nodeJS, se crear una con los diferentes verbos HTTP, pero definitivamente fue un nuevo reto para mi.

¡Muchísimas gracias por esta oportunidad!