MEDAC
Instituto Oficial de Formación Profesional

Documentación
PokeAPI

Juan Carlos Castilla Guisado

Índice

1. Introducción

Mi grupo consta de Pablo Ortiz Vergara, Juan Aparicio Falcón, Fernando Gomez Fernandez y yo, donde hemos decidido trabajar sobre la API de Pokemon ya que, esta es pública y no necesitamos de ninguna clave para acceder a los datos con lo que queremos trabajar.

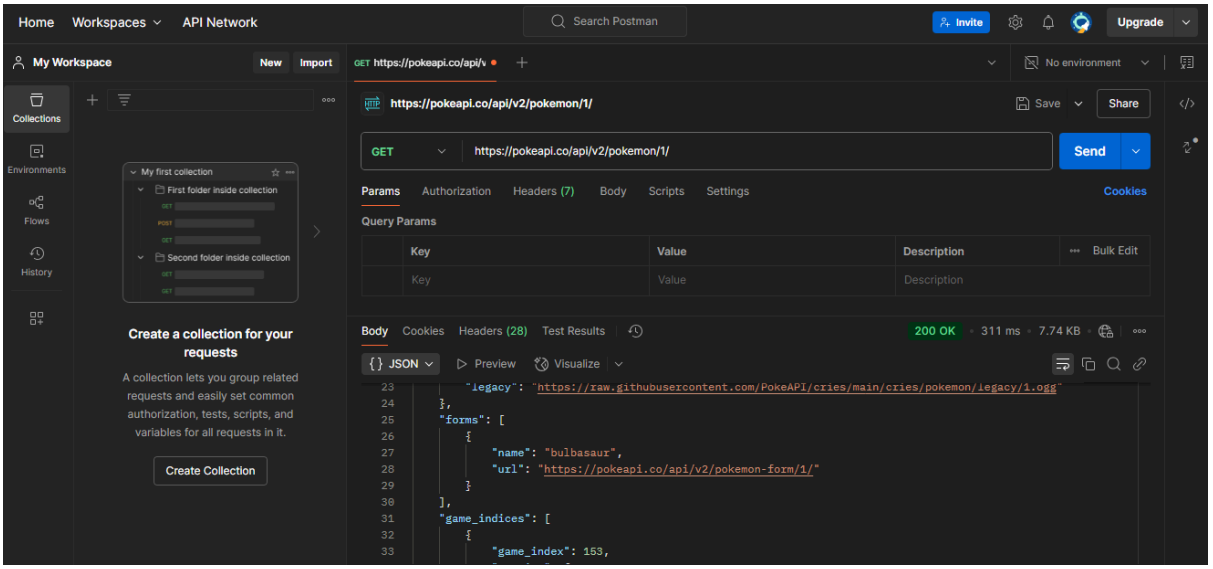
Nos vamos a basar en trabajar sobre los pokemons de la primera generación, como son por ejemplo: bulbasaur, charizard, machop...

La url principal de la PokeAPI es: https://pokeapi.co/api/v2/ donde nosotros vamos a trabajar desde la url: https://pokeapi.co/api/v2/pokemon/ y despues añadiendo el id del pokemon que va en base al número de este en la pokédex nos sale la información de cada uno de los pokémon que queremos utilizar como en este caso por ejemplo https://pokeapi.co/api/v2/pokemon/1/ esta que es la id del primer pokemon que es bulbasaur.

Al mostrar los pokemons nosotros vamos a trabajar sobre el nombre de cada pokémon , donde los vamos a separar con los tipos que hay como son lucha, agua, fuego... y nos da los datos generales de cada pokémon, donde queremos que nuestro trabajo sea una de las mejores opciones que tengas para ver los pokemons de la primera generación, como recomendación para pasar con mucha más facilidad el juego pokemon.

2. Comprobación funcionamiento en Postman

Aquí podemos ver que para hacer la comprobaciones de que funcione la url de la API seleccionada, debemos poner bien la ruta donde estamos probando la url: https://pokeapi.co/api/v2/pokemon/1/ . Al clicar en send se puede ver el resultado de esta url donde nos da un JSON de más de 12000 líneas de código con información de este pokémon en este caso, donde para probar los demas pokemons es con la misma url solo que cambiando la id del pokemon y nos daría el mismo resultado solo que cambiando los datos por los datos de el otro pokemon que queramos usar.



Por ejemplo para usar la línea evolutiva de planta de la primera generación debemos usar estos enlaces:

- Para bulbasaur: https://pokeapi.co/api/v2/pokemon/1/
- Para ivysaur: https://pokeapi.co/api/v2/pokemon/2/
- Para venusaur: https://pokeapi.co/api/v2/pokemon/3/

Con el mismo procedimiento que he mostrado para todas las demás, ya que no requiere mucha más complicación a la hora de usar las url.

3. Implementación del Servicio

Para la implementación del servicio he creado una función en node.js donde con la api de pokemon utilizamos la ruta y pidiendo al final el nombre del pokemon para buscar información sobre este, ya que en el final del código le digo el pokemon que quiero que busque donde puedo poner tantos como yo quiera y lo hace todo correctamente. Para que funcione debemos usar axios que lo debemos instalar en la misma carpeta que estamos trabajando.

Código:

```
const axios = require('axios');

const fs = require('fs'); // Requerimos el módulo fs para manejar archivos

async function getPokemonData(pokemonName) {
  try {
  const response = await axios.get(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`);
    console.log('Datos de Pokémon:', response.data);
    // Guardar los datos en un archivo JSON
   const fileName = `${pokemonName}.json`; // El nombre del archivo será el nombre del Pokémon
    fs.writeFileSync(fileName, JSON.stringify(response.data, null, 2)); // Guardamos el JSON con formato legible (espacios de 2)
   console.log(`Datos guardados en el archivo ${fileName}`);
  } catch (error) {
    console.error('Error al obtener datos de Pokémon:', error);
  }
```

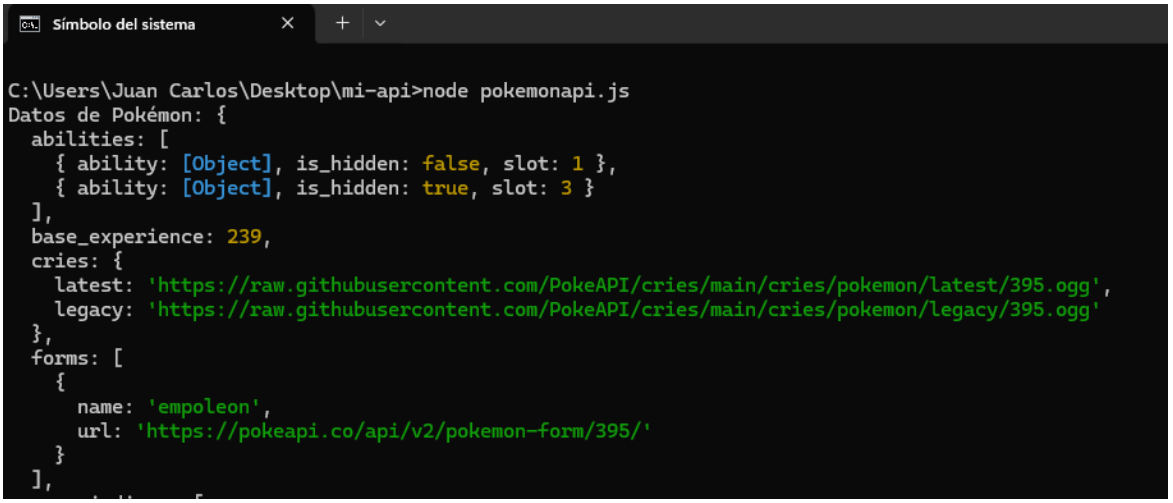
```
}

// Probar la función con un Pokémon (por ejemplo, Pikachu)

getPokemonData('charmander');

getPokemonData('charizard');

getPokemonData('empoleon');
```

Explicación:

Este código utiliza axios para obtener datos de la API de Pokémon. La función getPokemonData recibe el nombre de un Pokémon, hace una solicitud HTTP a la API y luego guarda la respuesta en un archivo JSON con el nombre del Pokémon. Si la solicitud es exitosa, los datos se imprimen en la consola y se almacenan en un archivo. Si ocurre un error, se muestra un mensaje de error. En el ejemplo se obtiene la información de "charmander", "charizard" y "empoleon". Donde este resultado lo guardamos en un json para tener los datos recogidos a mano en todo momento.

Cómo ejecutar código:

Para ejecutar este archivo debemos abrir la terminal y situarnos en la carpeta donde tenemos nuestra api y para ejecutar el archivo pokemonapi.js debemos usar el comando node que es para iniciar estos tipos de archivos ya que estamos trabajando todo con node.js



4. Integración con el Backend

Con la ayuda del backend hemos creado un endpoint donde con el código en nuestro server.js donde debemos usar express, axios y fs que hemos implementado anteriormente en nuestra carpeta para añadirlos en nuestro proyecto y donde la función de este código es obtener los datos con una funcion y despues obtenerlos también con un endpoint donde iniciamos iniciamos el servidor con el puerto 3000 que es el que hemos puesto anteriormente en nuestro código.

Código:

```
const express = require('express');

const axios = require('axios');

const fs = require('fs');

const app = express();

const port = 3000;

// Función que obtiene datos de Pokémon

async function getPokemonData(pokemonName) {

  try {

  const response = await axios.get(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`);

    return response.data; // Retorna los datos obtenidos

  } catch (error) {

    console.error('Error al obtener datos de Pokémon:', error);

    return { error: 'No se pudo obtener datos del Pokémon.' };

  }
```

```
}

// Endpoint para obtener los datos de un Pokémon

app.get('/pokemon/:name', async (req, res) => {

    const pokemonName = req.params.name.toLowerCase(); // Captura el nombre del Pokémon desde la URL

  const data = await getPokemonData(pokemonName); // Llama a la función con el nombre del Pokémon

  res.json(data); // Devuelve los datos como respuesta JSON

});

// Iniciar el servidor

app.listen(port, () => {

  console.log(`Servidor corriendo en http://localhost:${port}`);

});
```

Explicación:

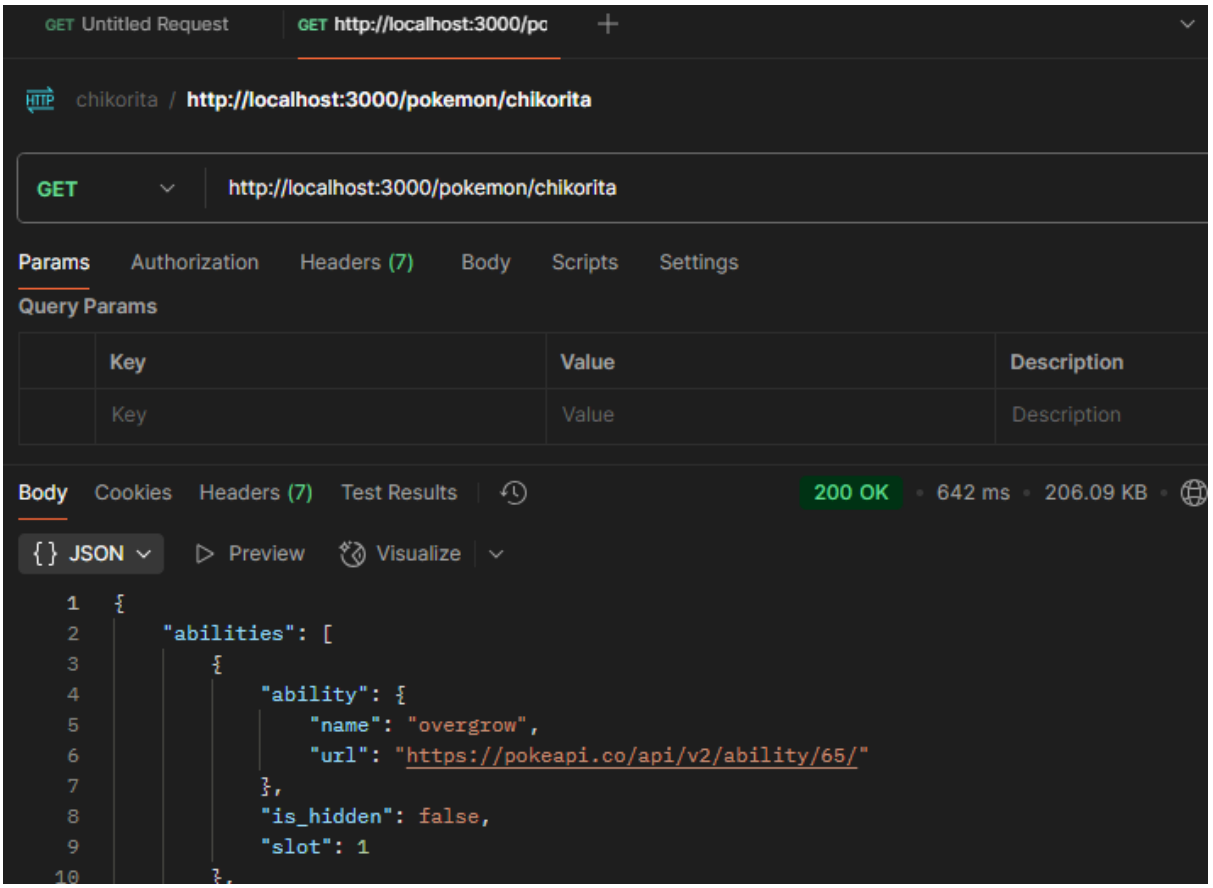
Con este código hacemos que al utilizar postman para comprobar si funciona el endpoint debemos usar la ruta poniendo el localhost:3000 que es donde hemos colocado nuestro servidor, donde por ejemplo con esta ruta http://localhost:3000/pokemon/chikorita encontramos los datos que nos da la api de este pokémon en concreto donde nos puede dar el resultado tanto en postman que se ve todo mejor y más ordenado o en el navegador donde sale la información desordenada.

Para que todo esto funcione debemos iniciar el servidor ejecutando el archivo server.js en nuestra terminal situada en nuestra carpeta del proyecto donde ponemos el comando **node server.js**.

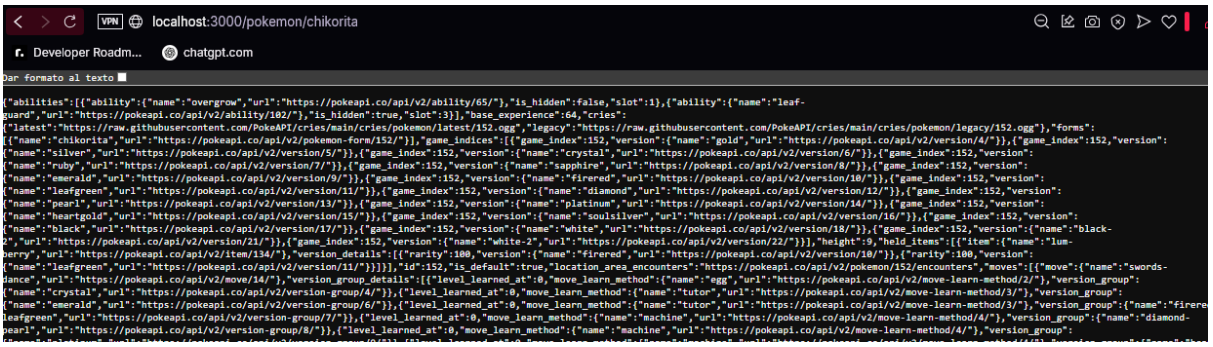
```
C:\Users\Juan Carlos\Desktop\mi-api>node server.js
Servidor corriendo en http://localhost:3000
```

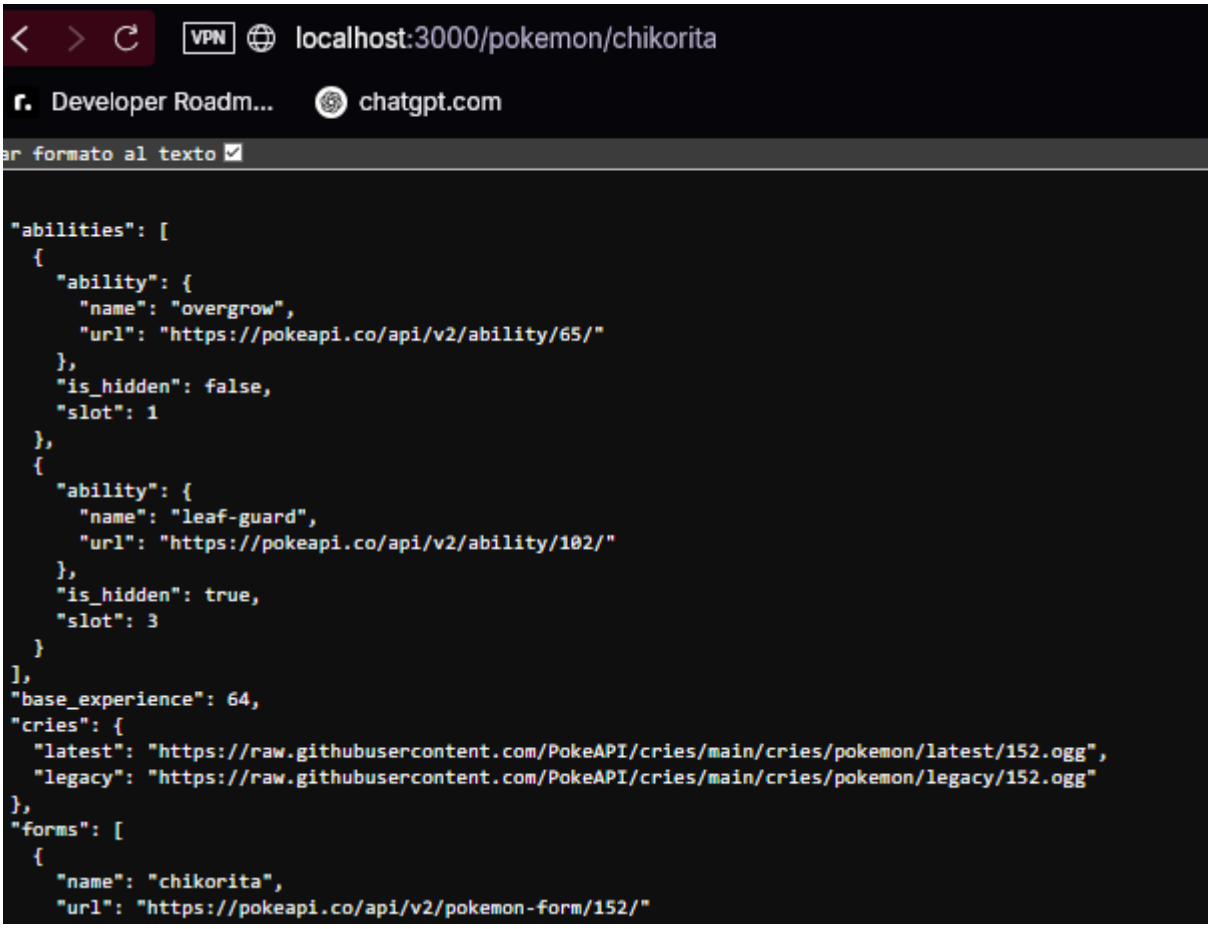
Y como resultado final estos son los resultados que nos da al meterlo en el postman y en el navegador.

Postman



Navegador


localhost:3000/pokemon/chikorita
Developer Roadm...
chatgpt.com


VPN
localhost:3000/pokemon/chikorita
Developer Roadm...
chatgpt.com
formato al texto
"abilities": [
{
"ability": {
"name": "overgrow",
"url": "https://pokeapi.co/api/v2/ability/65/"
},
"is_hidden": false,
"slot": 1
},
{
"ability": {
"name": "leaf-guard",
"url": "https://pokeapi.co/api/v2/ability/102/"
},
"is_hidden": true,
"slot": 3
}
],
"base_experience": 64,
"cries": {
"latest": "https://raw.githubusercontent.com/PokeAPI/cries/main/cries/pokemon/latest/152.ogg",
"legacy": "https://raw.githubusercontent.com/PokeAPI/cries/main/cries/pokemon/legacy/152.ogg"
},
"forms": [
{
"name": "chikorita",
"url": "https://pokeapi.co/api/v2/pokemon-form/152/"