Construccion de una Api Rest en Node JS con express



1. Creamos una carpeta e iniciamos en VSC.
2. Creamos un proyecto con el comando **npm init -y** Esto nos generará una archivo **package.json**
3. Instalamos Express mediante **npm install express** Esto generará una carpeta node\_modules y una archivo package-lock.json
4. Creamos un archivo con nombre **index.js**
5. En el archivo package.json debajo de “main” agregamos "type":"module", para que nos deje usar la estructura import al requerir paquetes.
6. En el archivo index.js escribimos el siguiente código para levantar el servidor y comprobar su funcionamiento.

import express from 'express'

const app = express();

app.get("/",(req, res) =>{

res.send("Bienvenidos a mi Api Rest en Nodejs")

})

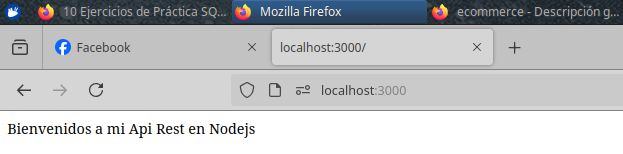
app.listen(3000,() =>{

console.log("Servidor corriendo en el puerto 3000")

})

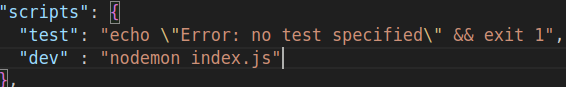
Si todo va bien, en un navegador escribimos [**http://localhost:3000/**](http://localhost:3000/)Esto debe arrojar en el DOM

***Bienvenidos a mi Api Rest en Nodejs***



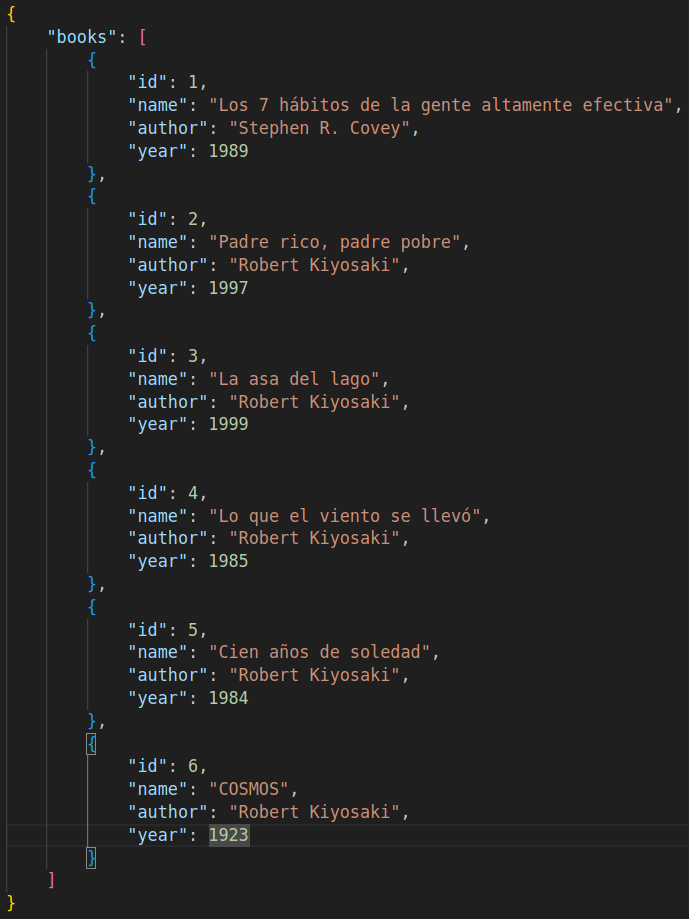
7. Para no detener e iniciar el servidor cada vez que se hace un cambio en el código, instalamos nodemon mediante **npm install nodemon -D** La D significa que instale como una dependencia de desarrollo.

Para que nodemon funcione correctamente debemos ir al archivo package.json y agregar en la parte de scripts la siguiente línea “dev” : “nodemon index.js”



Ahora iniciamos el servidor mediante **npm run dev**

8. Creamos una archivo **db.json** que contendrá los datos necesarios para las peticiones.



9. En nuetro archivo index.js escribimos **import fs from "fs";** para manipular archivos. También escribimos **import bodyParser from "body-parser"; app.use(bodyParser.json());** para trabajar en formato json los datos.

10. Creamos las funciones de lectura y escritura de los datos respectivamente.

const readData = () => {

try {

const data = fs.readFileSync("./db.json");

return JSON.parse(data);

} catch (error) {

console.log(error);

}

};

const writeData = (data) => {

try {

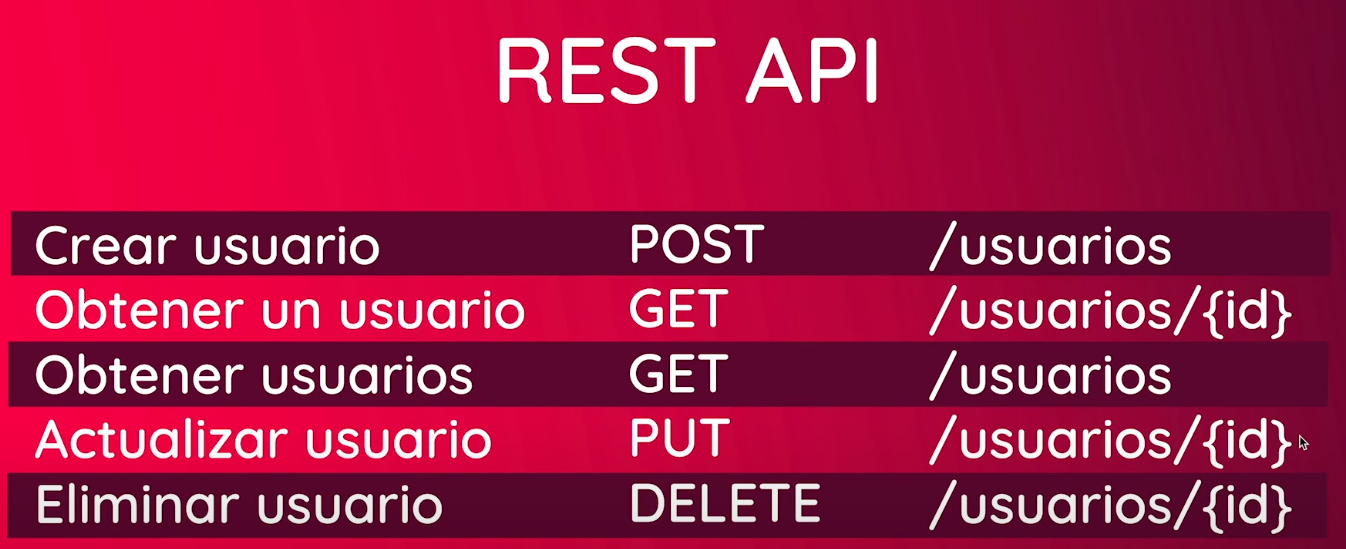
fs.writeFileSync("./db.json", JSON.stringify(data));

} catch (error) {

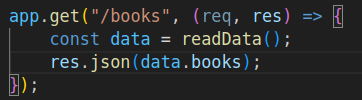
console.log(error);

}

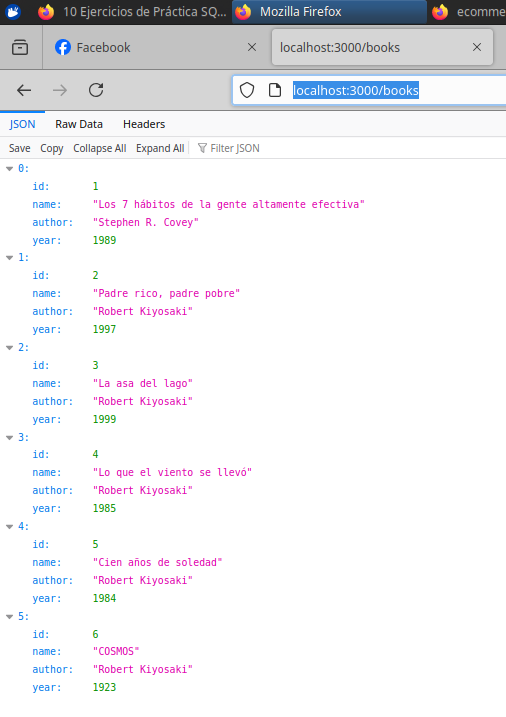
};



11. Creamos nuestra primera petición con el método GET. Este método nos permite leer los datos y mostrarlos en consola o el DOM del navegador.

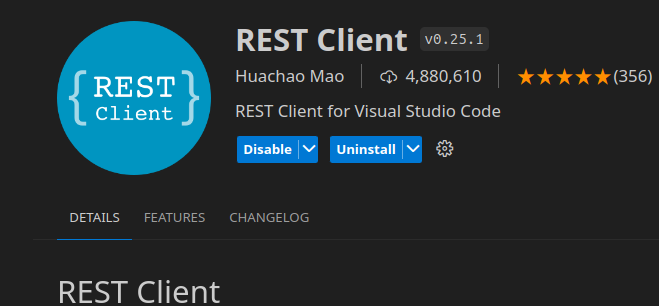


Si en la url del navegador escribimos localhost:3000/books debe aparecer los datos en el DOM del archivo db.json



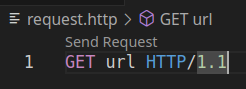
Otra forma de comprobar esta y las siguientes peticiones es mediante agunas herramientas tales como POSTMAN, ThunderClient o RestClient. En este caso utilizaremos la última señalada.

Para ello, en VSC buscamos la extensión RestClient y la instalamos.



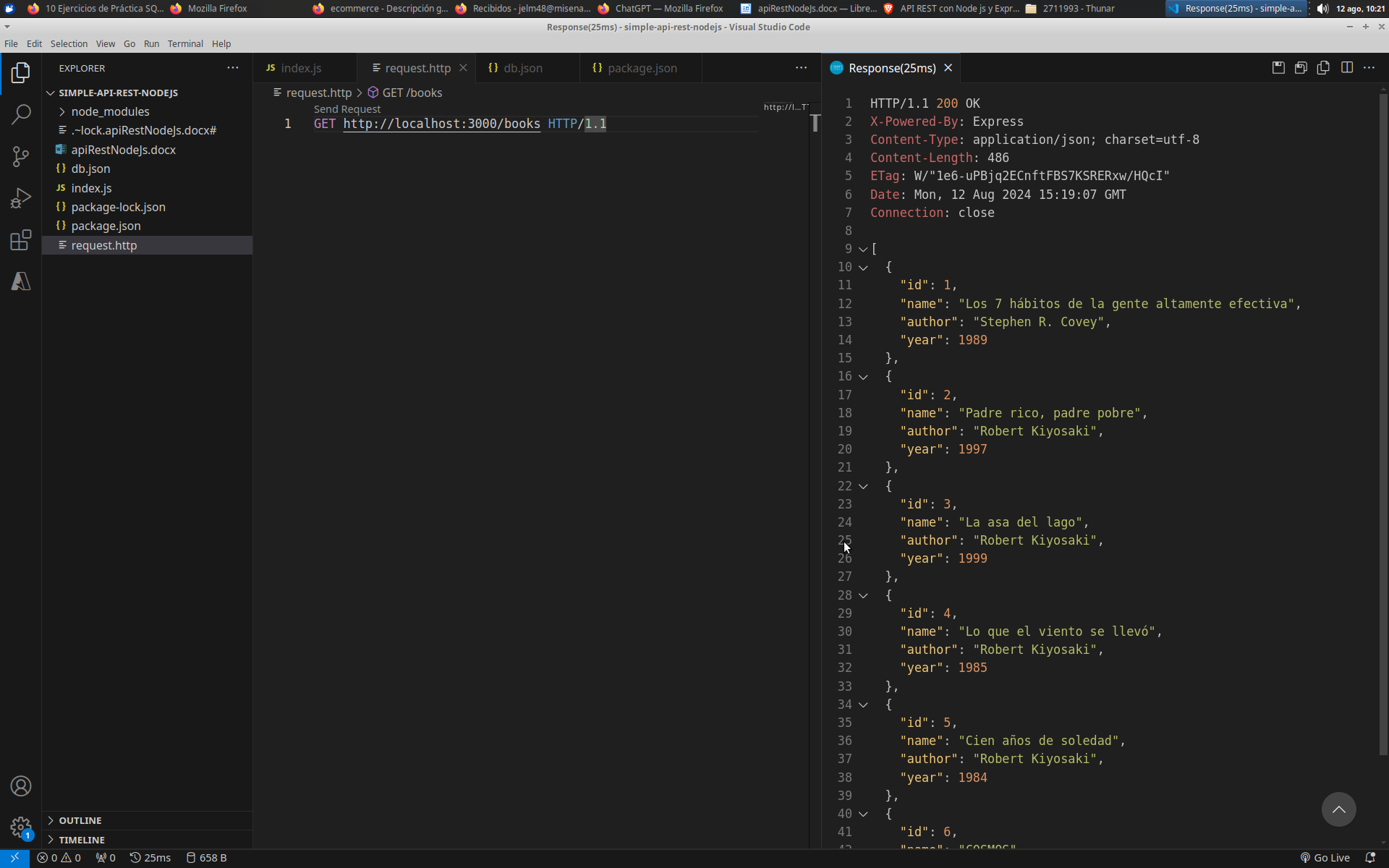
Una vez instalada la extensión creamos una archivo llamado **request.http**

En dicho archivo escribimos GET y damos enter. Se mostrará algo similar a la imagen

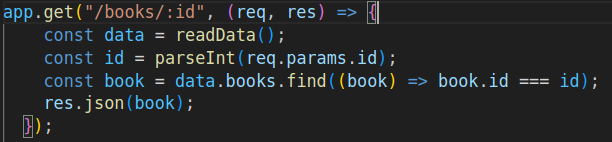


En url escribimos [**http://localhost:3000/books**](http://localhost:3000/books)y damos click en Send Request

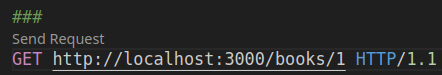
Debe a parecer en la parte derecha el resultado de la petición que en este caso son los datos.



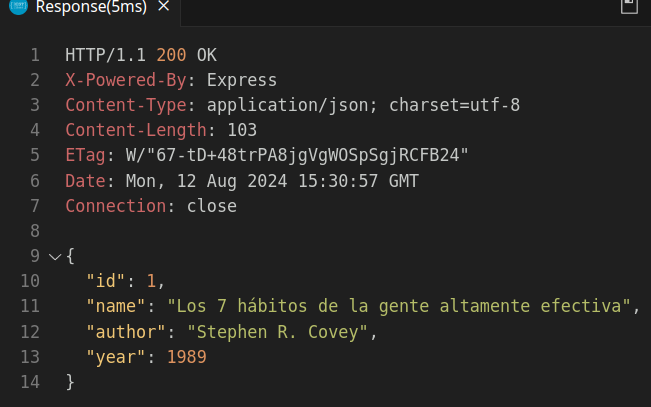
12. Creamos otra petición tipo GET pero para traer un recurso específico y no todos.



Para comprobar esta petición, en el archivo request.http creamos otra petición y pasamos como argumento el id del libro o recuso que queremos traer.



En la imagen observamos que se pasa como parámetro el id 1 dando como resultado



13. Creando una petición Post para crear un recurso

app.post("/books", (req, res) => {

const data = readData();

const body = req.body;

const newBook = {

id: data.books.length + 1,

...body, //con el spreadOperator le decimos que lo que venga por el body lo agregue a la variable newBook

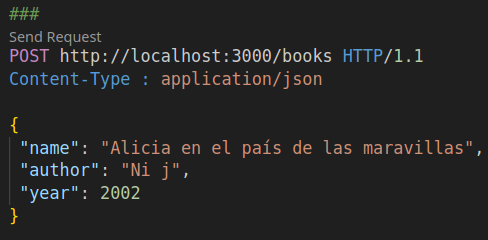
};

data.books.push(newBook);

writeData(data);

res.json(newBook);

});



14. Creando una petición Put para actualizar un recurso

app.put("/books/:id", (req, res) => {

const data = readData();

const body = req.body;

const id = parseInt(req.params.id);

const bookIndex = data.books.findIndex((book) => book.id === id);

data.books[bookIndex] = {

...data.books[bookIndex],

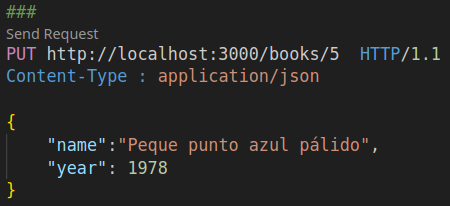
...body,

};

writeData(data);

res.json({ message: "Book updated successfully" });

});



15. Creando una petición DELETE para borrar un recurso.

app.delete("/books/:id", (req, res) => {

const data = readData();

const id = parseInt(req.params.id);

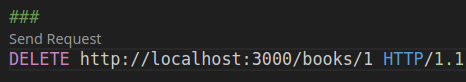
const bookIndex = data.books.findIndex((book) => book.id === id);

data.books.splice(bookIndex, 1);

writeData(data);

res.json({ message: "Book deleted successfully" });

});



https://www.youtube.com/watch?v=BImKbdy-ubM