

Desarrollando un CRUD en ANGULAR

Parte 4
Agregando un backend

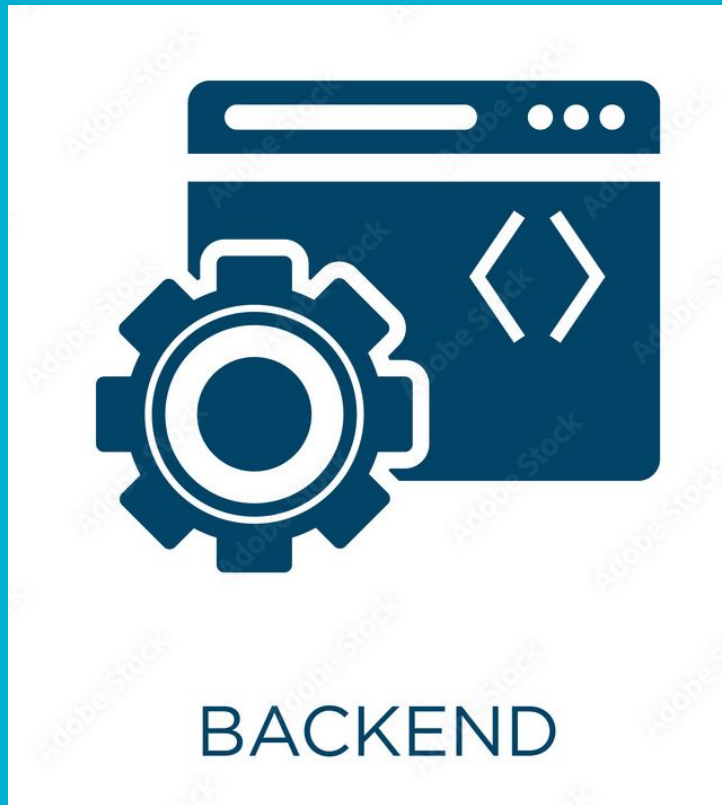
Mct. Esteban Calabria



Enunciado

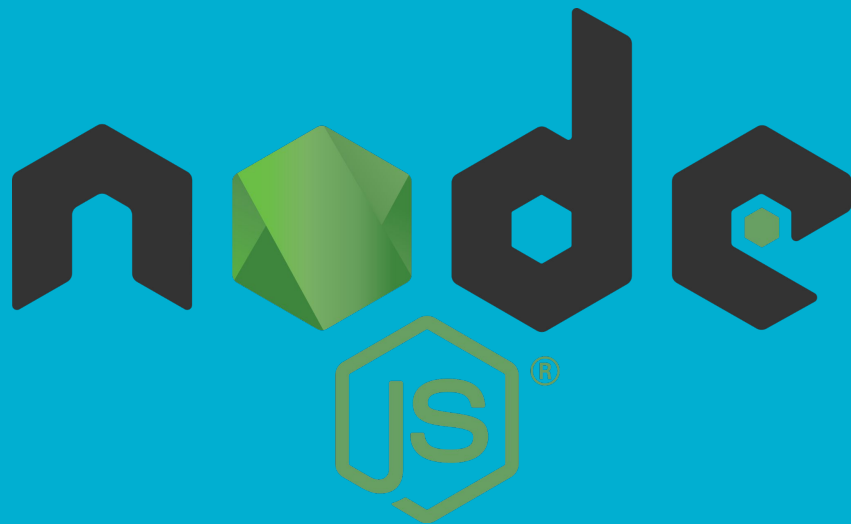
En el laboratorio pasado hicimos un CRUD en angular que funciona todo en memoria.

Vamos a Agregar un backend.



Opciones

1. Crear un backend con JSON server
2. Crear un backend con NodeJS
 - a. RECOMENDADA




Crear un backend con JSON-Server


Paso 1

- Instalar JSON Server.
 - Crear un archivo JSON con los datos.
 - Ejecutar JSON Server.
 - Uso en desarrollo.
-

← → ↻ github.com/typicode/json-server

☰  typicode / json-server 🔍 Type to search

<> Code ⓘ Issues 595 🔗 Pull requests 90 ⏮ Actions 📁 Projects 🛡 Security 📈 Insights

 json-server Public ❤ Sponsor 👁 Watch 996 ▼

🔗 main ▼ 🔗 9 Branches 🏷 153 Tags 🔍 Go to file t + <> Code ▼

JSON Server es una herramienta que permite crear una API RESTful rápida y fácilmente basada en un archivo JSON para el desarrollo y pruebas de aplicaciones.

<https://github.com/typicode/json-server>

Pasos a seguir

- Instalar JSON Server.
- Crear un archivo JSON con los datos.
- Ejecutar JSON Server.
- Acceder a los datos.
- Configuración adicional (opcional).
- Uso en desarrollo.

Instalar JSON Server

Instala JSON Server globalmente para poder usarlo desde cualquier lugar

Para instalarlo globalmente, ejecuta:

```
npm install -g json-server
```

Luego crea un archivo JSON con los datos llamado contactos.json



```
{
  "contactos": [
    {
      "id": "1",
      "documento": 12345678,
      "nombre": "Juan Pérez",
      "fechaNacimiento": "1985-10-15T00:00:00.000Z",
      "genero": "Masculino"
    },
    {
      "id": "2",
      "documento": 23456789,
      "nombre": "María García",
      "fechaNacimiento": "1990-05-25T00:00:00.000Z",
      "genero": "Femenino"
    }
  ]
}
```

Ejecutar JSON Server.

Para ejecutar el JSON-SERVER en el mismo directorio donde tenemos el archivo contactos.json ejecutar:

```
json-server -watch contactos.json  
-port 3000
```

O Bien...

```
npx Json-server -watch  
contactos.json -port 3000
```

```
Microsoft Windows [Version 10.0.22631.3737]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Cursos\Angular\Angular-PWA\backend>json-server --watch contactos-json-server.json  
--watch/-w can be omitted, JSON Server 1+ watches for file changes by default  
JSON Server started on PORT :3000  
Press CTRL-C to stop  
Watching contactos-json-server.json...  
  
( ^_ ^ )  
  
Index:  
http://localhost:3000/  
  
Static files:  
Serving ./public directory if it exists  
  
Endpoints:  
http://localhost:3000/contactos
```




Podremos acceder al
JSON-Server en la url

<http://localhost:3000/>



Endpoints de JSON Server para /contactos

GET /contactos

- Devuelve una lista de todos los contactos.

GET /contactos/{id}

- Devuelve un contacto específico basado en su id.

POST /contactos

- Crea un nuevo contacto.

PUT /contactos/{id}

- Actualiza un contacto existente basado en su id.

DELETE /contactos/{id}

- Elimina un contacto específico basado en su id.



Crear una api con Node y Express

Paso 1

- Crear proyecto de Nodejs
- Instalar Librerias
-

Proyecto de Backend

Primero hay que verificar que tenemos nodejs instalado con el comando

```
npm -version
```

Sino lo instalamos de

<https://nodejs.org/en>

Creamos un proyecto con el comando

```
npm init
```

Inicializa un nuevo proyecto Node.js creando un archivo package.json interactivo con la configuración básica del proyecto.

```
C:\backend>npm init
```

This utility will walk you through creating a package.json file. It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and save it as a dependency in the package.json file.

Press ^C at any time to quit.

package name: (backend)

version: (1.0.0)

description:

entry point: (index.js)

test command:

git repository:

keywords:

author:

license: (ISC)

About to write to C:\backend\package.json:

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Is this OK? (yes)

Instalar Dependencias Necesarias

Para desarrollar nuestro backend vamos a instalar las siguientes dependencias

```
npm install cors express jsonwebtoken
```

- **cors**
 - Middleware para habilitar CORS (Cross-Origin Resource Sharing) en aplicaciones Express.
- **express**
 - Framework de servidor web minimalista y flexible para Node.js, utilizado para construir aplicaciones y APIs robustas.
- **jsonwebtoken**
 - Librería para crear y verificar JSON Web Tokens (JWT) para autenticación y autorización segura en aplicaciones.

```
"dependencies": {  
  "cors": "^2.8.5",  
  "express": "^4.19.2",  
  "jsonwebtoken": "^9.0.2"  
}
```

Así queda el package.json

Creamos Servidor

Creamos un archivo llamado

`server.json`

Este script configura y arranca un servidor web utilizando Express.

Primero, se importan las librerías necesarias: Express para crear el servidor, CORS para manejar las políticas de recursos compartidos entre distintos orígenes,\

Se definen algunas constantes importantes como puerto en el que correrá el servidor, y la URL base para los endpoints de la API de contactos.

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

const PASSWORD_JWT = 'passwordjwt';
const PORT = process.env.PORT || 3000;
const BASE_URL = "/contactos";

const app = express();

app.use(cors());
app.use(express.json());

app.listen(PORT, () => {
  console.log(`Server is on port ${PORT}`);
});
```

Probar que el server funcione

Vamos a probar que el server funcione con el siguiente comando

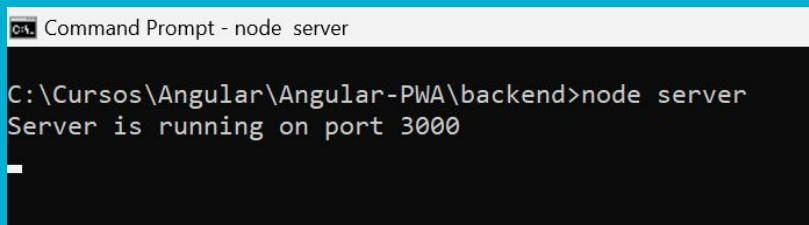
```
node server.js
```

Debe ser ejecutado en el mismo lugar donde está el package.json y el server.js

Si deseas no tener que reiniciar el servidor por cada cambio puedes usar la utilidad nodemon instalandola así:

```
Npm install -g nodemon
```

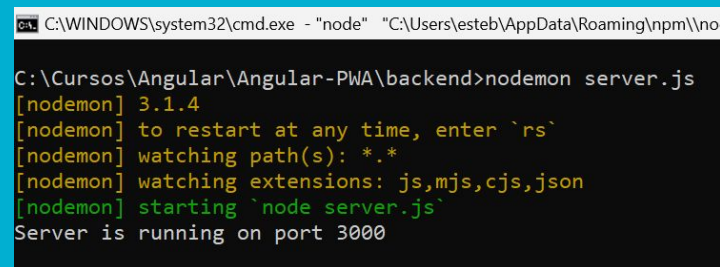
Cada paso hay que probar el serve



```
Command Prompt - node server

C:\Cursos\Angular\Angular-PWA\backend>node server
Server is running on port 3000
```

Puedes ejecutarlo con nodemon sino...



```
C:\WINDOWS\system32\cmd.exe - "node" "C:\Users\esteb\AppData\Roaming\npm\nodemon"
C:\Cursos\Angular\Angular-PWA\backend>nodemon server.js
[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server is running on port 3000
```

Manejar contactos

Vamos a agregar a nuestro código la lógica para agregar contactos.

- Probar nuevamente el server
- Verificar que se genera el archivo contactos.json

El código define una lista de contactos y proporciona dos funciones: una para guardar esta lista en un archivo JSON y otra para leer los contactos desde ese archivo. Al final, se llama a la función que lee los contactos desde el archivo.

```
app.use(express.json());
```

```
let contactos = [  
  {  
    "id": "1",  
    "documento": 12345678,  
    "nombre": "Juan Pérez",  
    "fechaNacimiento": "1985-10-15T00:00:00.000Z",  
    "genero": "Masculino"  
  },  
  {  
    "id": "2",  
    "documento": 23456789,  
    "nombre": "María García",  
    "fechaNacimiento": "1990-05-25T00:00:00.000Z",  
    "genero": "Femenino"  
  }  
];
```

```
function salvarContactosEnArchivo(){  
  const fs = require('fs');  
  fs.writeFileSync('contactos.json', JSON.stringify(contactos));  
}
```

```
function leerContactosDeArchivo(){  
  const fs = require('fs');  
  contactos = JSON.parse(fs.readFileSync('contactos.json'));  
}
```

```
leerContactosDeArchivo();
```


Crear Endpoints

```
app.put(`${BASE_URL}/${id}`, (req, res) => {
  console.log('PUT /contactos/' + req.params.id);
  const id = req.params.id;
  const contacto = req.body;
  const index = contactos.findIndex(contacto => parseInt(contacto.id) === parseInt(id));

  if (index === -1){
    return res.status(404).json({error: 'Contacto no encontrado'});
  }

  contactos[index] = contacto;
  res.json(contactos[index]);

  salvarContactosEnArchivo();
});

app.delete(`${BASE_URL}/${id}`, (req, res) => {
  console.log('DELETE /contactos/' + req.params.id);
  const id = req.params.id;
  contactos = contactos.filter(contacto => parseInt(contacto.id) !== parseInt(id));
  res.json({message: 'Contacto eliminado'});

  salvarContactosEnArchivo();
});
```

```
app.get(BASE_URL, (req, res) => {
  console.log('GET /contactos');
  leerContactosDeArchivo();
  console.log(contactos);
  res.json(contactos);
});
```

```
app.post(BASE_URL, (req, res) => {
  console.log('POST /contactos');
  //Para que req.body tenga el json convertido en un objeto tengo que agregar el
  middleware express.json()
  const contacto = req.body;
```

```
  if (!contacto){
    return res.status(400).json({error: 'El contacto es requerido'});
  }
```

```
  if ((contacto.nombre) && (contacto.nombre.length <= 4)){
    return res.status(400).json({error: 'El nombre es requerido o es muy corto'});
  }
```

```
  contacto.id = Math.max(...contactos.map(contacto => parseInt(contacto.id)), 0) + 1;
  contactos.push(contacto);
  res.json(contacto);
```

```
  salvarContactosEnArchivo();
});

});
```

Endpoints de JSON Server para /contactos

GET /contactos

- Devuelve una lista de todos los contactos.

POST /contactos

- Crea un nuevo contacto.

PUT /contactos/{id}

- Actualiza un contacto existente basado en su id.

DELETE /contactos/{id}

- Elimina un contacto específico basado en su id.



Probar endpoints

- Probar endpoints con una herramienta tipo postman o la extension de VsCode RapidApi
- Integrar backend con nuestro proyecto de frontend



RapidAPI

Sigamos Trabajando...