Tansatck query/React query and Devtools

Step 1:For mocking apis fake will create our own

# Install npm i json server to mock the api

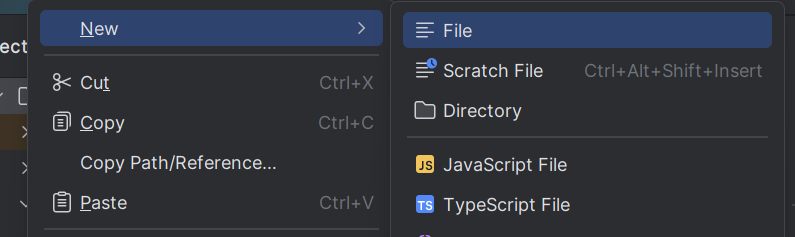
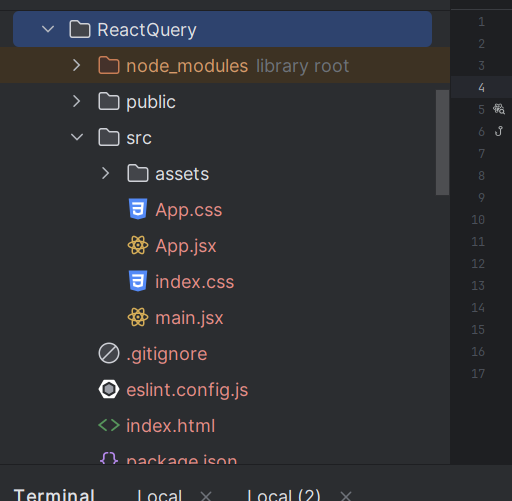
We need data for that we need lightweight Jason server

Install it…only for dev purpose and practice

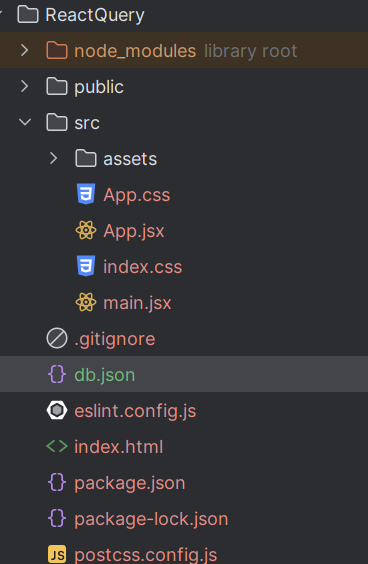
**>npm i json-server**

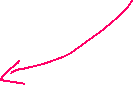
# Create Database file in the parent directory

Db.json

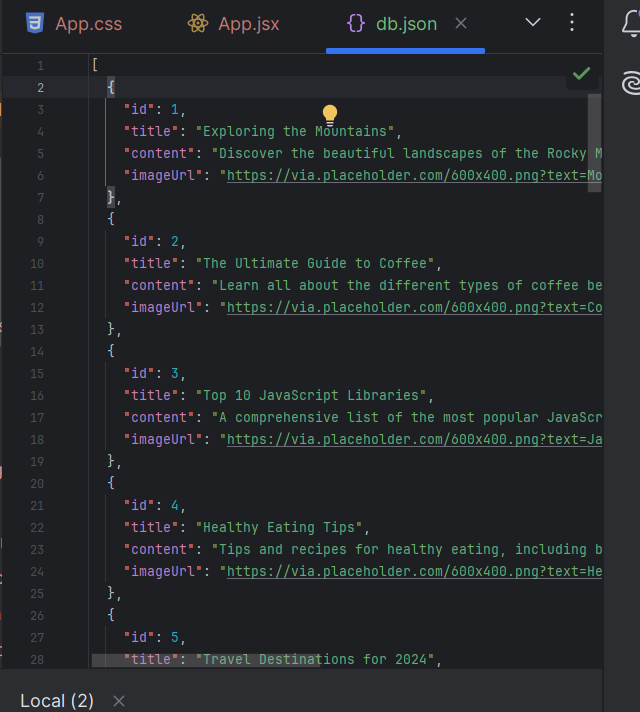








# Add data in db.json

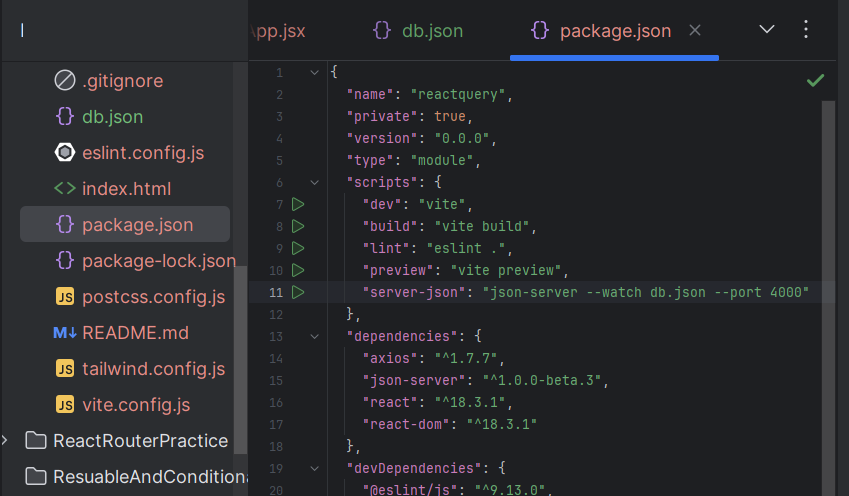


# Setup server to run and it port on package.json

Uder scripts

And yes you can give any port

"server-json": "json-server --watch db.json --port 4000"





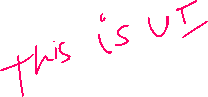
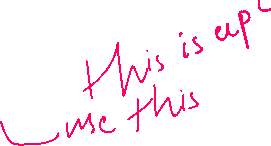
Command to start the server

>npm run server-json



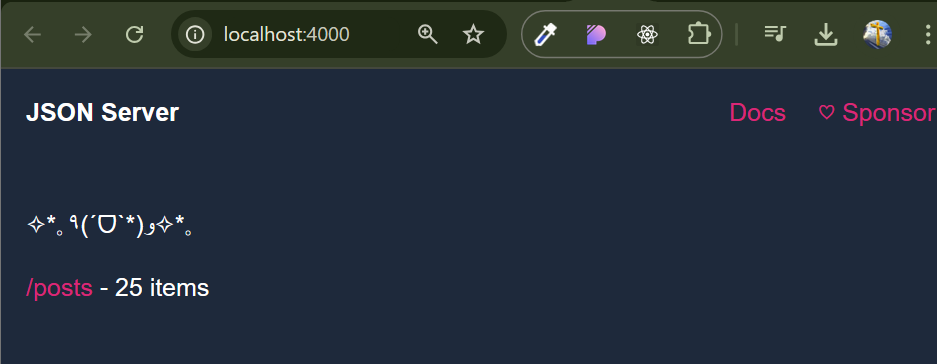


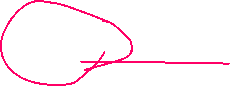




If you click on index link

This also provides docks to do all opertations





Docks on handling api

* **json-server**

Important

Viewing beta v1 documentation – usable but expect breaking changes. For stable version, see [here](https://github.com/typicode/json-server/tree/v0)

👋 *Hey! Using React, Vue or Astro? Check my new project*[*MistCSS*](https://github.com/typicode/mistcss)*to write 50% less code.*

* **Install**

npm install json-server

* **Usage**

Create a db.json or db.json5 file

{

"posts": [

{ "id": "1", "title": "a title", "views": 100 },

{ "id": "2", "title": "another title", "views": 200 }

],

"comments": [

{ "id": "1", "text": "a comment about post 1", "postId": "1" },

{ "id": "2", "text": "another comment about post 1", "postId": "1" }

],

"profile": {

"name": "typicode"

}

}

View db.json5 example

Pass it to JSON Server CLI

$ npx json-server db.json

Get a REST API

$ curl http://localhost:3000/posts/1

{

"id": "1",

"title": "a title",

"views": 100

}

Run json-server --help for a list of options

* **Sponsors ✨**

| **Sponsors** |
| --- |
|  |
|  |

| **Sponsors** |
| --- |
|  |

| **Sponsors** |  |
| --- | --- |
|  |  |
| [Delivery Routing Software and Route Optimization Software](https://route4me.com/) |  |

[Become a sponsor and have your company logo here](https://github.com/users/typicode/sponsorship)

* **Sponsorware**

Note

This project uses the [Fair Source License](https://fair.io/). Only organizations with 3+ users are kindly asked to contribute a small amount through sponsorship [sponsor](https://github.com/sponsors/typicode) for usage. **This license helps keep the project sustainable and healthy, benefiting everyone.**

For more information, FAQs, and the rationale behind this, visit <https://fair.io/>.

* **Routes**

Based on the example db.json, you'll get the following routes:

GET /posts

GET /posts/:id

POST /posts

PUT /posts/:id

PATCH /posts/:id

DELETE /posts/:id

# Same for comments

GET /profile

PUT /profile

PATCH /profile

* **Params**
* **Conditions**
* → ==
* lt → <
* lte → <=
* gt → >
* gte → >=
* ne → !=

GET /posts?views\_gt=9000

* **Range**
* start
* end
* limit

GET /posts?\_start=10&\_end=20

GET /posts?\_start=10&\_limit=10

* **Paginate**
* page
* per\_page (default = 10)

GET /posts?\_page=1&\_per\_page=25

* **Sort**
* \_sort=f1,f2

GET /posts?\_sort=id,-views

* **Nested and array fields**
* x.y.z...
* x.y.z[i]...

GET /foo?a.b=bar

GET /foo?x.y\_lt=100

GET /foo?arr[0]=bar

* **Embed**

GET /posts?\_embed=comments

GET /comments?\_embed=post

* **Delete**

DELETE /posts/1

DELETE /posts/1?\_dependent=comments

* **Serving static files**

If you create a ./public directory, JSON Server will serve its content in addition to the REST API.

You can also add custom directories using -s/--static option.

json-server -s ./static

json-server -s ./static -s ./node\_modules

* **Notable differences with v0.17**
* id is always a string and will be generated for you if missing
* use \_per\_page with \_page instead of \_limitfor pagination
* use Chrome's Network tab > throtling to delay requests instead of --delay CLI option

# SERVERLESS DEPLOYEMENT ON VERSEL

# # Deploying a React App with JSON Server on Vercel This guide will walk you through deploying a React app that uses JSON Server on \*\*Vercel\*\*. Vercel supports serverless functions, which can be used to simulate the behavior of a backend API using JSON Server. ## Prerequisites - Node.js installed on your machine. - Vercel account: [Sign up here](https://vercel.com/signup). - GitHub account (for Vercel to connect to your project). ## Steps ### 1. \*\*Prepare Your React App (Frontend)\*\* Ensure that your React app is fully functional locally, and your API calls (e.g., `fetchFruits`) are working as expected. ### 2. \*\*Install `json-server` as a Dependency\*\* Install `json-server` in your React app to use it in a serverless function. ```*bash* npm install json-server 3. Create API Routes Using Serverless Functions In Vercel, create an api folder in the root of your project to handle serverless API requests. Create a file for the fruits API, for example api/fruits.js. Add the following code to api/fruits.js to handle requests using json-server: javascript Copy code // File: api/fruits.js import jsonServer from 'json-server'; import fs from 'fs'; import path from 'path'; const dbPath = path.resolve('./db.json'); export default function handler(req, res) { const server = jsonServer.create(); const router = jsonServer.router(dbPath); const middlewares = jsonServer.defaults(); server.use(middlewares); server.use(router); if (req.method === 'GET') { // Use json-server's router for GET requests server.use(router); } server(req, res); // Route the request to json-server } 4. Add Your db.json File Create a db.json file in the root of your project to store your data. Here is an example of how the db.json might look: json Copy code { "fruits": [ { "id": 1, "name": "Apple" }, { "id": 2, "name": "Banana" }, { "id": 3, "name": "Orange" } ] } 5. Deploy Your Project to Vercel Once the fruits API route is set up and the data is in place, follow these steps to deploy your project to Vercel: Push your code to a GitHub repository. Connect your GitHub repository to Vercel via the Vercel dashboard. Vercel will automatically detect that you are deploying a React app and will handle the deployment. Make sure your api folder with the serverless functions and the db.json file is included in the repository. Deploy the project. 6. Update React Code to Use the Vercel Serverless API In your React app, update your API requests to point to the serverless functions on Vercel. For example: javascript Copy code const fetchFruits = async ({ pageParam = 1 }) => { const url = `https://your-vercel-project-name.vercel.app/api/fruits?\_page=${pageParam}&\_per\_page=5`; const response = await axios.get(url); return response.data; }; Replace your-vercel-project-name with the actual name of your Vercel project. 7. Test the Deployment Once your project is deployed, visit your Vercel app URL to verify that the frontend is fetching data correctly from the serverless API. 8. Possible Issues and Workarounds Cold Starts: Serverless functions take some time to start after being idle. This might result in slight delays during API requests. File System Access: Serverless functions don't persist the file system. The data in db.json won't be saved between invocations. You may want to use a persistent database (e.g., MongoDB, PostgreSQL) for production. Rate Limits: Ensure your API calls don't exceed the rate limits of Vercel's serverless functions. Conclusion This guide shows how to deploy a React app with JSON Server as a serverless backend on Vercel. For more robust, production-ready backends, consider integrating a persistent database instead of using json-server. Feel free to customize this guide as needed for your project. If you encounter any issues, check the Vercel documentation for troubleshooting: Vercel Docs. csharp Copy code ### Notes: - Replace `your-vercel-project-name` with the actual project name from your Vercel dashboard when updating the URLs. - This approach simulates a backend using `json-server` in serverless functions.