**Routing and CRUD Operation Basics**

In this blog post, let’s take a pause (from all the learning) and try to understand i) what is routing ii) what are CRUD operations and iii) the relationship that exists between routing and CRUD operations in web applications.

**Routing**

Setting up a url such that it performs some operation is what is known as routing. In static websites, urls redirect users to different pages of the website. In web applications, these same urls are designated with the task of performing some functionality (dynamically) that the user is requesting for.

For example, users might want to create a new task if they are in a task manager application. A specific url can be set aside for this purpose using routing. And the job of that URL would be to allow users to create a new task.

So what we did in our previous blog posts using app.get() (Serving a Page to the Users) and app.post() methods (Displaying User Submitted Data on the Browser) is essentially routing.

In the first one, when the users reached the endpoint localhost:3000, we routed them to a simple page that was serving up a message. In the second one, when the same endpoint was reached, this time we routed them to a simple html form, so that they could submit/post their data to the server.

**Note:** this is routing in its basic form. Later, we’ll see routing in detail.

**CRUD Operations**

Now what are CRUD operations? Well CRUD stands for create, read, update and delete. They are the most common operations any web application performs - without which web apps cannot exist. Because, after all, as long as there is data, the need to create, read, update and delete them also exists.

**What is their relationship with Routing?**

Well, you might have guessed it by now. Every route we create will be responsible for one of the CRUD operations.

Let’s assume you are creating a task manager application. Then you’ll need specific routes that will allow the users to do the following : i) Create the task ii) Read the task iii) Update the task and iv) Delete the task.

In other words, you will need specific routes to fulfill the CRUD operations.

**Creating Your Own CRUD Using Routing To Understand Web Apps Better**

Simply put, we are going to configure a bunch of URLs that will be responsible for: Creating, Reading, Updating and Deleting (CRUD) data. [mention backend only somewhere]

For now we are not even going to setup our own server using Express. Instead we’ll be using an npm package called JSON server that will do the job for us, so that we can exclusively focus on setting up urls for performing the CRUD operations.

**But What Is JSON Server NPM Package Exactly?** [not needed or shorten it]

It’s an npm package front-end developers use to quickly set up mock APIs (urls) that mirror the backend operation so that they can focus exclusively on designing the front-end part without really having a dedicated backend in place yet.

(This is the reason why we don’t have to create our own server while using JSON server npm package).

It is also a great tool for beginners when it comes to understanding the CRUD. That’s why we are using it here.

**Note:** But What Are APIs? It’s just a fancy name given to the bunch of URLs we’ll be creating to perform the CRUD.  We’ll come back to APIs at the end of this post. For now, let’s just focus on our CRUD.

**How Does JSON Server Operate?**

To perform CRUD operations, we need data. That means we need a database. And the JSON server npm package makes use of a .json file (which we are required to create) as our database to store the data.

As far as the server is concerned, the JSON server creates it for us (and runs it at localhost:3000) behind the scenes, so that we can just focus on creating the urls for performing the CRUD.

**Installing JSON Server NPM Package**

You just have to run the following command:

    npm install -g json-server

-g stands for global install. This simply means you can use json server package anywhere within your system without having to install it again.

**Configuring the db.json file**

It’s time to set up our database. So create a file called db.json somewhere in your system and add the following code to it and save the file.

{

 "users": [

   {

     "name": "Robin",

     "email": "robin@gmail.com",

     "id": "101"

   },

   {

     "name": "Rehman",

     "email": "rehman@gmail.com",

     "id": "102"

   },

   {

     "name": "Ramesh",

     "email": "ramesh@gmail.com",

     "id": "103"

   }

 ]

}

If you observe carefully, you’ll realize that users is an array of objects. This means the data we will be dealing with (for performing the CRUD) are basically objects within the array. (To know more about array of objects visit this article)

**Note:** This data structure is very similar to what you’ll find in mongo database. (To know more about mongodb, read this).

Time to Run Our JSON Server

Once you save the db.json file, navigate into the folder where the db.json file is, and run the json server by typing the following cmd command:

   json-server --watch db.json

The json server will serve a couple of endpoints (urls) back to us as response.

**‘R’ Of The CRUD Is Probably The Easiest**

Copy the url below the Resources section and access it via the browser.

You’ll notice that the response contains the data we just typed in db.json file.

This means we are reading the data (from our db.json database file) by accessing the localhost:3000/users url.

You should’ve realized by now that the json server is using the app.get() method behind the scenes to render the data on the browser (just as we rendered the simple message previously in our blog post using express).

This is what is known as a GET request.

GET requests are used by servers to read the data from the database. They are of 2 types: the one we just saw now that gets all the data available in that route; and another one for getting hold of a specific data in that route (by using its id).

**Getting Specific Data By Using Its ID**

To get a specific data, say Rehman, from our /users route, we have to pass the data’s id as a part of the url as shown below:

  http://localhost:3000/users/102

You’ll notice that the Rehman gets fetched and displayed in the browser.

**Note:** databases contain id(s) for uniquely identifying each and every record. Our json-server is replicates the same behaviour.

**Moving Onto The ‘C’ ‘U’ ‘D’ Of CRUD**

Create, Update and Delete Operation require data submission of some sort. Now we are not going to setup a user interface here (since our goal is to get the endpoints working and not data submission).

Instead we’ll be using an endpoint (or api or application url) testing tool called Postman that will help us to submit data to our server.

**Introducing Postman - The Endpoint Testing Tool**

Postman is very easy to setup and can even be installed in your system locally.

As shown in the image above, Postman contains an address bar (where you can type out your server endpoints) and a send button. To the left of the address bar there will be a dropdown that lists several methods.

For now, we’ll just concern ourselves with those methods that help with CRUD operation. They are: GET, POST, PATCH and DELETE.

**Making a GET REQUEST**

Though we have already done this within our browser itself. You can try it out again using postman by doing the following.

* Select the method as ‘GET’ from the dropdown
* Type in the endpoint (localhost:3000/users) and
* Hit Send button

The fetched records get displayed as the response.

**Making a GET REQUEST USING ID**

To get a information relating to a specific record, we have to pass its id as a part of the url, like we did in the browser.

* Select the method as ‘GET’ from the dropdown
* Type in the endpoint with the id of the data you want to fetch (localhost:3000/users/103) and
* Hit Send button

The record’s details get displayed as the response.

**Making a POST REQUEST**

We know that to post something to the database (db.json file), we need data. This data can be passed from the postman tool to our server as follows:

* Select the method as ‘POST’ from the dropdown
* Type in the endpoint (localhost:3000/users)
* In the Body section, select raw as the option, and the set JSON(application/json) as the option for sending the data.
* Now type in the record’s content. This should follow the same pattern that we saw in the db.json file.
* Hit Send button

The data you just posted gets displayed as the response.

**Making a PATCH REQUEST**

Patch is another term for update. To update a specific record, we need its id, just like we did when we got (or fetched) the details of a specific record.

* Select the method as ‘PATCH’ from the dropdown
* Type in the endpoint with the id of the data (localhost:3000/users/103) you want to update.
* Set the Content-Header Type as ‘application/json’
* In the Body section, select raw as the option, and the set JSON(application/json) as the option for sending the data.
* Now modify the record’s content.
* Hit Send button

The data you just updated gets displayed as the response.

**Making a DELETE REQUEST**

Again you need to pass the id of the record you wish to delete.

* Select the method as ‘DELETE’ from the dropdown
* Type in the endpoint with the id of the data you want to delete (localhost:3000/users/103) and
* Hit Send button

The data you just deleted gets displayed as the response.

1. Install JSON Server

2. Configure db.json file

3. Running JSON Server

4 Testing Endpoints Using Postman