**Topic: JSON Server**

**How to create a local json server**

1. Download and Install Node js
2. Check whether it is installed or not by using below commands

**node –version**

**npm –version**

**Open powershell run as administrator and run the commands**

**Get-ExecutionPolicy**

**Set-ExecutionPolicy RemoteSigned**

**Y (for yes)**

**Get-ExecutionPolicy**

1. Select the folder in your local , open in vs code terminal and use below command

**npm init –y**

You will find json packages then you can install any libraries

1. Install json server for api creation in local server

**npm install -g json-server@0**

1. After installation watch the server using below command if it throws error follow the next step for script enabalation.

**json-server --watch db.json –port num**

1. Adjust the version by reinstalling using below command

**npm install -g json-server@0**

**json-server --watch db.json –port 6000**

**Stepwise All Methods**

**Step 1 – create db.json file**

**Step 2 – add the data**

{

    "data":[{

    "name":"teja",

    "id":"4"

},

{

    "name":"sai",

    "id":"2"

},

{

    "name":"hemanth",

    "id":"3"

},

{

    "name":"chaitanya",

    "id":"1"

}

]}

**Step 3 – json-server --watch db.json –port 6000**

**Step 4 - open js file paste the below program in it**

    //  get method - used to get data from the server

            fetch("http://localhost:3000/data")

            .then(val=>val.json())

            .then(data=>console.log(data))

            .catch(err=>console.log("data not found"))

**How to use post or patch in fetch**

Index.html

<button onclick="getdata()">get data</button>

    <button onclick="setdata()">set data</button>

Script.js:

    <script>

        function getdata(){

            //  get method - used to get data from the server

            fetch("http://localhost:3000/data")

            .then(val=>val.json())

            .then(data=>console.log(data))

            .catch(err=>console.log("data not found"))

        }

        let obj={

            name:"Raju",

            id:5

        };

        function setdata(){

            //  set method - used to set   data to the server

            fetch("http://localhost:3000/data", {

                method: "POST",

                headers: {

                    "Content-Type":"application/json"

                },

                body: JSON.stringify(obj)

            })

            .then(val=>val.json())

            .then(data=>console.log(data))

            .catch(err=>console.log("data not found"))

        }

    </script>

Db.json:

{

  "data": [

    {

      "name": "Abhi",

      "id": "1"

    },

    {

      "name": "sai",

      "id": "2"

    },

    {

      "name": "hemanth",

      "id": "3"

    },

    {

      "name": "manoj",

      "id": "4"

    }

  ]

}

**Json server**

HTTP methods (GET, POST, PUT, PATCH, DELETE) using fetch() and interacting with a JSON Server. JSON Server is a simple API tool that allows you to simulate a REST API with basic HTTP methods

Assume your JSON server is running at http://localhost:3000 with the following db.json:

{

    "data":[{

    "name":"teja",

    "id":"4"

}

,

{

    "name":"sai",

    "id":"2"

},

{

    "name":"hemanth",

    "id":"3"

},

{

    "name":"chaitanya",

    "id":"1"

}

]}

**1. GET Request: Fetching Data from the Server**

fetch('http://localhost:3000/posts')

  .then(response => response.json())  // Parse the JSON response

  .then(data => console.log(data))    // Handle the received data

  .catch(error => console.error('Error:', error));  // Handle errors

If you want to add queries params

By name

 fetch("http://localhost:3000/data?name=teja")

By id

 fetch("http://localhost:3000/data?id =2")

By both

 fetch("http://localhost:3000/data?id =2&name=teja")

By limit- used for pagination

 fetch("http://localhost:3000/data?\_limit=2 ")

By \_sort

 fetch("http://localhost:3000/data?\_sort=-id")

**2. POST Request: Sending Data to the Server**

fetch('http://localhost:3000/posts', {

  method: 'POST',

  headers: {

    'Content-Type': 'application/json'  // Indicate we are sending JSON

  },

  body: JSON.stringify({                // Data to be added

    title: 'New post',

    author: 'Sam'

  })

})

  .then(response => response.json())    // Parse the JSON response

  .then(data => console.log('Post added:', data))  // Handle the response

  .catch(error => console.error('Error:', error));  // Handle errors

. Explanation: This adds a new post with the title "New post" and author "Sam" to the posts endpoint.

**3. PUT Request: Replacing Existing Data on the Server**

  fetch('http://localhost:3000/posts/1', {

  method: 'PUT',

  headers: {

    'Content-Type': 'application/json'

  },

  body: JSON.stringify({

    id: 1,            // `PUT` requires the `id`

    title: 'Updated first post',

    author: 'John Doe'

  })

})

  .then(response => response.json())

  .then(data => console.log('Post updated:', data))

  .catch(error => console.error('Error:', error));

Explanation: This updates the post with id: 1, changing its title and author.

**4. PATCH Request: Partially Updating Data on the Server**

  fetch('http://localhost:3000/posts/2', {

  method: 'PATCH',

  headers: {

    'Content-Type': 'application/json'

  },

  body: JSON.stringify({

    author: 'Jane Smith'  // Only update the author field

  })

})

  .then(response => response.json())

  .then(data => console.log('Post partially updated:', data))

  .catch(error => console.error('Error:', error));

Explanation: This updates only the author field of the post with id: 2.

**5. DELETE Request: Removing Data from the Server**

  fetch('http://localhost:3000/posts/1', {

  method: 'DELETE'

  })

  .then(() => console.log('Post deleted'))  // Handle deletion success

  .catch(error => console.error('Error:', error));