JSON

JSON, or JavaScript Object Notation, is a **lightweight data interchange format** widely used in web development and other software applications. Its **syntax is derived from JavaScript object notation, but it is language independent** making it easy to read and write for humans. JSON is commonly used for transmitting data between a server and a web application due to its simplicity, universality, and support for complex data structures like nested objects and arrays. It is supported by virtually all modern programming languages and is **commonly used in web APIs** for its lightweight nature and ease of parsing. JSON's security is generally robust, but precautions should be taken to prevent vulnerabilities like JSON injection attacks. Additionally, JSON Schema provides a way to define and validate the structure and constraints of JSON documents, enhancing data integrity and interoperability. Overall, JSON's simplicity, readability, and wide support make it a popular choice for **data interchange in various software applications**.

JSON.stringify():

JSON.stringify() is a built-in JavaScript method used to convert a JavaScript object into a JSON string.

JSON.parse():

JSON.parse() is a built-in JavaScript method used to parse a JSON-formatted string and convert it into a JavaScript object.

Http status codes

HTTP status codes are standard response codes returned by web servers to indicate the outcome of a client's request.

Important HTTP status codes along with their meanings:

1. **200 OK**: This status code indicates that the request was successful, and the server has returned the requested resource.
2. **201 Created**: Indicates that the request was successful, and a new resource has been created as a result.
3. **204 No Content**: The server successfully processed the request, but there is no content to return.
4. **400 Bad Request**: This status code is returned when the server cannot process the request due to a client error, such as malformed syntax or invalid parameters.
5. **401 Unauthorized**: Indicates that the client needs to authenticate itself to access the requested resource.
6. **403 Forbidden**: The server understood the request, but the client is not allowed to access the requested resource.

Http methods

HTTP methods, also known as HTTP request methods, are actions that indicate the desired operation to be performed on a resource identified by a URI (Uniform Resource Identifier).

1. **GET**: The GET method requests a representation of the specified resource. It is primarily used for retrieving data from the server. GET requests should only retrieve data and should not have any other effect on the server.
2. **POST**: The POST method submits data to be processed to a specified resource. It is commonly used for creating new resources on the server or submitting form data.
3. **PUT**: The PUT method replaces all current representations of the target resource with the request payload. It is typically used to update or create a resource with a specific identifier.

4. **PATCH**: The PATCH method is used to apply partial modifications to a resource. It is similar to the PUT method but only updates the parts of the resource specified in the request.
5. **DELETE**: The DELETE method requests the removal of the specified resource. It is used to delete resources identified by the URI from the server.

How to call api using fetch by then method

```
fetch("https://fakestoreapi.com/products")
  .then((val) => {
    return val.json();
  })
  .then((val) => {
    console.log(val);
  });
```

1. **fetch("https://fakestoreapi.com/products")**: This line initiates a request to the specified URL, which returns a Promise representing the response to that request.
2. **.then((response) => { return response.json(); })**: Once the request is complete, this line chains a **.then()** method to the Promise returned by **fetch()**. Inside this **.then()** method, it takes the response object, and the **json()** method is called on it. This method returns a Promise that resolves to the JSON representation of the response body.
3. **.then((data) => { console.log(data); })**: After parsing the JSON response, this line chains another **.then()** method to the Promise returned by **response.json()**. Inside this **.then()** method, it receives the parsed JSON data as a JavaScript object. In this example, it logs the retrieved data to the console using **console.log()**

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
   Get-ExecutionPolicy

3) 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
4) Install json server for api creation in local server
   npm install -g json-server@0
5) 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

6) Adjust the version by reinstalling using below command
   npm install -g json-server@0

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

How to use post or patch in fetch

```
fetch("url", {
```

```
      method: "POST",
      body: JSON.stringify({
        name: "John",
        age: 20,
      }), //need to write headers
      headers: {
        "Content-type": "application/json; charset=UTF-8",
      },
    });

    //put changes the entire value

    //patch changes the one value which is updated
```

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

```
script>

      //  get method - used to get data from the server
      fetch("http://localhost:3000/data")
      .then(response=>{

         if(response.ok){

            return response.json()
         }else{
            return "error code" + response.statusText
         }
      }).then(data=>{
         console.log(data);
      })
```

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**")