Topic: Fetch, json, status codes and methods

Fetch

The **fetch()** API is a modern way to make network requests and handle responses. It is widely used to interact with web APIs, allowing to request and send data to servers. It returns a Promise that resolves to the Response object representing the response to the request.
fetch(url, options)
* **url**: The endpoint from which the resource is to be fetched.
* **options** (optional): An object containing additional settings such as HTTP method, headers, body, etc.

Example using fetch() API with Promises:

When Data fetched successfuly

```
fetch("https://fakestoreapi.com/products/1")
.then(res=>res.json())
.then(data=>console.log(data))
.catch(err=>console.log("there was a problem"))
```

When there is a error

```
fetch("https://fakestoreai.com/products/1")
.then(res=>res.json())
.then(data=>console.log(data))
.catch(err=>console.log("there was a problem"))
```

Explanation:
• fetch() returns a promise that resolves with the response object once the data is available.
• The .then() method is used to handle the fulfilled promise.
• If something goes wrong, we can use .catch() to handle the rejection.

. Async/Await
The async and await keywords simplify working with promises, allowing us to write asynchronous code that looks synchronous.
- async: Marks a function as asynchronous. This means it always returns a promise.
- await: Pauses the execution of the function until the promise is resolved or rejected.

Example using fetch() API with async/await:

```
async function executor(){
   var res= await fetch("https://fakestoreapi.com/products/1");
   var data = await res.json();
   console.log(data);
}
executor()
```

async with try and catch

```
async function executor() {
  try {
    var res = await fetch("https://fakestoreapi.com/products/1");
    var data = await res.json();
    console.log(data);
  } catch (err) {
    console.log("data not loading");
  }
}
executor();
```

Explanation:
• The executor function is marked as async, which means it can use await.
• Inside the function, await fetch() pauses the execution until the fetch() promise is resolved.

- Once resolved, the response is checked and the JSON data is awaited.
- If there's an error (e.g., network failure), it is caught by the try...catch block.

Differences Between Promises and Async/Await:
- Readability: Async/await makes asynchronous code look synchronous, which improves readability, especially in complex scenarios with many chained .then() blocks.
- Error Handling: Async/await uses try...catch, which is often more intuitive for handling errors compared to .catch() in promises.

JSON- javascript object notation

Ajax

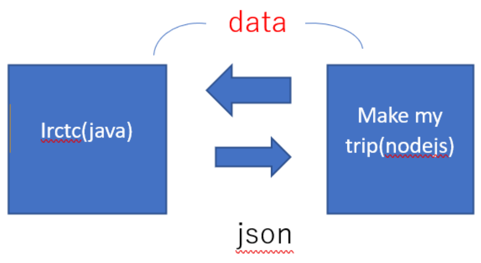
Ajax, which stands for "Asynchronous JavaScript and XML," is a web development technique used to create interactive web applications. It allows for the asynchronous exchange of data between the browser and the server

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. Overall, JSON's simplicity, readability, and wide support make it a popular choice for **data interchange in various software applications**.

Why json

B2b



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()**

Some apis

Food api: **https://api.edamam.com/search?q=biriyani&app_id=a52b4d43&app_key=e0e5c667605f5e91d8275c973531b80a**

Weather api - https://api.openweathermap.org/data/2.5/weather?q=**hyderabad**&units=metric&appid=466ddaa21a8de191e9f608bd11a56acb

Quotes api: https://api.quotable.io/random

Random joke: **https://v2.jokeapi.dev/joke/Programming?blacklistFlags=nsfw,religious,political,racist,explicit&type=single**

Movies info:- https://www.omdbapi.com/?t=**titanic**&apikey=76d079f0

Random joke

```
async function getRandomJoke() {
  const rawRes = await fetch('https://icanhazdadjoke.com/', {
    headers: {
      Accept: 'application/json'
    }
  });
  const res = await rawRes.json();
  console.log(res.joke);

}
getRandomJoke()
```

https://github.com/saiteja-yernagula/javascript-mini-projects