

Requests

JSON-Formatted Response Body

The `.json()` method will resolve a returned promise to a JSON object, parsing the body text as JSON.

In the example code, the `.json()` method is used on the `response` object which returns a promise to a JSON-formatted response body as `jsonResponse`.

```
fetch('url')
  .then(
    response => response.json()
  ).then(jsonResponse => {
    console.log(jsonResponse);
  });
```

HTTP GET Request

HTTP GET requests are made with the intention of retrieving information or data from a source (server) over the web.

GET requests have no *body*, so the information that the source requires, in order to return the proper response, must be included in the request URL path or query string.

The fetch() Function

The JavaScript Fetch API is used to write HTTP requests using Promises. The main `fetch()` function accepts a URL parameter and returns a promise that resolves to a response object or rejects with an error message if a network error occurs.

The example code begins by calling the `fetch()` function. Then a `then()` method is chained to the end of the `fetch()`. It ends with the response callback to handle success and the rejection callback to handle failure.

```
fetch('url')
  .then(
    response => {
      console.log(response);
    },
    rejection => {
      console.error(rejection.message);
    }
  );
```

Customizing Fetch Requests

The `fetch()` function accepts an optional second argument, an options object, used to customize the request. This can be used to change the request type, headers, specify a request body, and much more.

In the example code below, the `fetch()` function as a second argument—an object containing options for the fetch request specifying the `method` and the `body`.

```
fetch('https://api-to-call.com/endpoint',
{
  method: 'POST',
  body: JSON.stringify({id: "200"})
}).then(response => {
```

```

    if(response.ok) {
        return response.json();
    }

    throw new Error('Request
failed!');
}, networkError => {
    console.log(networkError.message);
}).then(jsonResponse => {
    console.log(jsonResponse);
})

```

HTTP POST Request

HTTP POST requests are made with the intention of sending new information to the source (server) that will receive it.

For a POST request, the new information is stored in the *body* of the request.

Using async...await with Fetch

The `async ... await` syntax is used with the Fetch API to handle promises.

In the example code, the `async` keyword is used to make the `getSuggestions()` function an async function. This means that the function will return a promise. The `await` keyword used before the `fetch()` call makes the code wait until the promise is resolved.

```

const getSuggestions = async () => {
    const wordQuery = inputField.value;
    const endpoint =
`${url}${queryParams}${wordQuery}`;
    try{
const response = await fetch(endpoint,
{cache: 'no-cache'});
        if(response.ok){
            const jsonResponse = await
response.json()
        }
    }
    catch(error){
        console.log(error)
    }
}

```