Fetch API

Contents

Overview

`Fetch()` allows us to make network requests similar to `XMLHttpRequest(XHR)`. The biggest difference with using the Fetch API is that it utilises promises, which allows for simpler and cleaner code. As a result, this avoids callback hell and remembering the complex API of `XHR`

Tutorial

A traditional `GET` XMLHttpRequest would look like this:

```
const req = new XMLHttpRequest();
req.onreadystatechange = () => {
  // Example handle logic
  if (req.status === 200 && req.readyState == 4) {
    if (req.getResponseHeader("Content-Type") === "application/json") {
      console.log("oh look its some JSON: " + req.responseText);
    } else {
      console.log(
        "Looks like its not JSON but lets see what it is... " + req.responseText
      );
    }
  } else {
    console.log("Oh no... handle error");
  }
};
req.open("GET", "https://jsonplaceholder.typicode.com/comments");
req.setRequestHeader("example-header", "some-value");
req.send();
```

GET Request

As you can tell there is a lot more to keep track of and can be a little more complicated. The fetch `GET` request below on the other hand is more cleaner and simpler compared to the `XMLHttpRequest` above:

```
fetch(`https://jsonplaceholder.typicode.com/comments`) // 1
    .then((response) => {
        if (response.status !== 200) {  //  2
            console.error(`status: ${reponse.status}`);
            return;
        }
        response.json() // 3
        .then(data => console.info(data)) // 4
    }).catch((err)=> console.error(`${err}`)); // 5
}
```

Let's break this `fetch()` request down:

1. At line one we start using the `fetch()` method and we provide it with the URL we want to target.

2. At line two we check if the request is **anything** but a status code of **200**. If this condition is true, we log the status code. If we get a status code of 200 then the code continues to execute.
3. At line three we take the response from the fetch and convert it into a JSON format using `.json()`
4. We log the data to the console so we can see what our fetch call is returning (*This is optional however*).
5. Finally the fifth line will only execute if there is an error with the fetch request itself.

POST/PUT Request

When posting data using the `fetch()` API, we have a bit more configuring that is required.

1. We must specify the URL we want to post to.
2. Next we specify the `method` e.g. POST, PUT
3. We specify the content type that we are post e.g. `"application/json"` or `application/x-www-form-urlencoded; charset=UTF-8` ...etc
4. `JSON.Stringify()` - Is a method that converts a JavaScript object to a JSON string.
5. We need to specify the JSON object that we want to `POST`/`PUT`
    1. e.g. `console.log(JSON.stringify({ x: 5, y: 6 }));` will produce the following output `{"x":5,"y":6}`

```
fetch("http://www.someurl.com/users", { //1
    method: 'post', //2
    headers: {
      "Content-type": "application/json" //3
    },
    body: JSON.stringify( //4
      {
        "name": "John doe",//5
        "age": 5,
        "gender": "Male"
      }
    )
  })
  .then(res => res.json())
  .then((data) => console.log(`Request succeeded with JSON response ${data}`))
  .catch((error) => console.log(`Request failed ${error}`))
  });
```

Delete Request

When we want to delete a record we need to know the URL we are calling and the `id` of the record.

1. We set a variable called `id` that can be used to set the id of the record we want to delete.
2. we pass in the URL and we also pass in the id that we want to remove.
3. we specify the method in the case `delete`

```
    let id = 1; //1
    fetch("someURL/"+id, {  //2
        method: 'delete'  //3
      })
      .then((data) => {
        console.log(`Request succeeded with JSON response ${data}`);
        // some function to execute if successful
      })
      .catch((error) => {
        //some function to execute if error
      });
```

Exercises

Using this link as your API, create the following:

1. `GET` request for 'List User'

   ▶ Solution
2. `GET` request for 'Single User' with the id of `2`

   ▶ Solution
3. `POST` request for 'Create'

   - `name` with a value of "Morpheus"
   - `job` with a value of "Leader"

   ▶ Solution
4. `POST` request for 'Register - Successful'

   - `email` with a value of "eve.holt@reqres.inheus"
   - `password` with a value of "pistol"

     ▶ Solution
5. `POST` request for 'Login - Successful'

   - `email` with a value of "eve.holt@reqres.inheus"
   - `password` with a value of "cityslicka"

     ▶ Solution