JavaScript Fetch API (Network Requests) 26th Day

The Fetch API is a modern interface that allows you to make HTTP requests in the web browsers.

If you have worked with XMLHttpRequest (XHR) object, the Fetch API can perform all the tasks as the XHR object. In addition, it is a simple and clean API that uses the Promise to deliver more flexible features to request resources from the server.

The fetch() method is available in the global scope that instructs the browser to send a request to a provided URL.

Sending a Request.

The fetch() has only one parameter which most of the time is the URL of the resource that you want to fetch.

```
let response = fetch(url);
```

The fetch() method returns a Promise so you can use the then() and catch() methods to handle it

Example:

```
fetch(url)
.then(response => {
 // handle the response
}) .catch(error => {
 // handle the error
  });
```

When the request completes, the resource is available. At this time, the promise will resolve into a Response object. The Response object is the API wrapper for the fetched resource. The Response object has a number of useful properties and methods to inspect the response.

Reading a Response.

If the contents of the response are in the raw text format, you can use the text() method. The text() method returns a Promise that resolves with the complete contents of the fetched resource.

Example:

```
fetch('/readme.txt')
  .then(response => response.text())
  .then(data => console.log(data));
```

Or you can use the async/await.

Example:

```
async function fetchText() {
    let response = await fetch('/readme.txt');
    let data = await response.text();
  console.log(data);
}
```

Besides the text() method, the Response object has other methods such as json(), blob(), formData() and arrayBuffer() to handle respective data.

Status Codes of a Response.

The Response object provides the status code and status text via the status and statusText properties. When a request is successful, the status code is 200 and status text is OK.

Example:

```
async function fetchText() {
  let response = await fetch('/readme.txt');
   console.log(response.status); // 200
   console.log(response.statusText); // OK
if (response.status === 200) {
  let data = await response.text(); // handle data
} }
 fetchText();
```

Output:

```
200
OK
```

If the requested resource doesn't exist, the response code is 404:

Example:

```
let response = await fetch('/non-existence.txt');
 console.log(response.status); // 400
 console.log(response.statusText); // OK
```

Output:

```
400
Not Found
```

If the requested URL throws a server error, the response code will be 500. If the requested URL is redirected to the new one with the response 300–309, the status of the Response object is set to 200. In addition the redirected property is set to true.

The fetch() returns a promise that rejects when a real failure occurs such as a web browser timeout, a loss of network connection, and a CORS violation.

Fetch API example Project.

Suppose that you have user json file located on a web server with the following contents. users.json:

```
[{
        "username": "Cliff",
        "firstName": "Jones",
        "lastName": "Beth",
        "gender": "Female",
        "profileURL": "img/female.png",
        "email": "cliff@example.com"
    },
    {
        "username": "Gray",
        "firstName": "Grayhat",
        "lastName": "Harun",
        "gender": "male",
        "profileURL": "img/male.png",
        "email": "Gray@example.com"
    }
]
```

The following shows the HTML page:

```
<!DOCTYPE html>
 <html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Fetch API Demo</title>
<link rel="stylesheet" href="css/style.css">
</head>
<body>
<div class="container"></div>
<script src="js/app.js"></script>
 </body>
 </html>
```

In the app.js, we'll use the fetch() method to get the user data and render the data inside the element with the class container.

First, declare the getUsers() function that fetches users.json from the server.

```
async function getUsers() {
  let url = 'users.json';
try { let res = await fetch(url);
  return await res.json();
}
 catch (error) { console.log(error);
}
}
```

Then, develop the renderUsers() function that renders user data:

```
async function renderUsers() {
    let users = await getUsers();
    let html = '';
    users.map(user => {
        let htmlSegment = `
            <div class="user">
            <img src="${user.profileURL}" >
            <h2>${user.firstName} ${user.lastName}</h2>
            <div class="email">
            <a href="email:${user.email}">${user.email}</a>
            </div>
            </div>`;
        html += htmlSegment;
    });
    let container = document.querySelector('.container');
    container.innerHTML = html;
}
renderUsers();
```

The Fetch API allows you to asynchronously request for a resource. Use the fetch() method to return a promise that resolves into a Response object. To get the actual data, you call one of the methods of the Response object e.g text() or json(). These methods also resolve into the actual data. Use the status and statusText properties of the Response object to get the status and status text of the response.

Best Wishes. Lux Tech Academy.