CIT 208: Web Technology - II Assignment #9

Anup Adhikari
anup.adhikari@gandakiuniversity.edu.np

Gandaki University March 15, 2023

1 Introduction

1.1 aync ... await

The await operator is used to wait for a Promise and get its fulfillment value. It can only be used inside an async function or at the top level of a module.

2 Lab Objectives

1. Understanding the FETCH Api in Depth
2. To familiarize with async/await function in JavaScript
3. Understanding the Laravel Model Concept

3 Examples

3.1 Using Fetch to POST JSON-encoded Data

```
const data = { username: "example" };

fetch("https://example.com/profile", {
  method: "POST", // or ’PUT’
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify(data),
})
  .then((response) => response.json())
  .then((data) => {
    console.log("Success:", data);
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

3.1.1 Using fetch for DatoCMS

- Note the Bearer Token Bearer Token is a must for DATOCMS:

```
495fd247dfd42fd787e69ca8eb4d57
```

- Post URL: https://graphql.datocms.com/

- The other is POST body:

```
query MyQuery {
    allGalleries {
        title
        id
        image {
        width
        filename
        url
        }
    }
}
```

- Now let's set the JS Script:

datocms.js

```
const data = { "query": `
    query MyQuery {
        allGalleries {
        title
        category {
            title
            }
        image{
            url
            }
        }
    }
` };

fetch("https://graphql.datocms.com/", {
  method: "POST", // or 'PUT'
  headers: {
    "Content-Type": "application/json",
    "Authorization": "Bearer 495fd247dfd42fd787e69ca8eb4d57",
  },
  body: JSON.stringify(data),
})
  .then((response) => response.json())
  .then((data) => {
    // do other activites with data
    // data has the required results
    console.log("Success:", data);
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

async.js

```javascript
async function getData() {
try {
    const response = await fetch('https://bit2021.anup.pro.np/exam/web2/org.json');
    const data = await response.json();
    console.log(data);
} catch (error) {
    console.error(error);
}
}

getData();
```

In this example, the getData() function uses async to make it an asynchronous function. Within the function, we use await to wait for the response from a fetch request to a hypothetical API endpoint. Once we have the response, we use await again to extract the JSON data from the response.

If any errors occur during this process, we catch them using a try/catch block and log them to the console.

Note that the use of async/await requires the function to return a promise. In this case, the getData() function implicitly returns a promise that resolves to the JSON data from the API endpoint.

3.2 Awaiting a promise to be fulfilled

AwaitWithFunction.js

```javascript
function resolveAfter2Seconds(x) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(x);
    }, 2000);
  });
}

async function f1() {
  const x = await resolveAfter2Seconds(10);
  console.log(x); // 10
}

f1();
```

3.3 Control Flow Effects of AWAIT

When an await is encountered in code (either in an async function or in a module), the awaited expression is executed, while all code that depends on the expression's value is paused and pushed into the microtask queue. The main thread is then freed for the next task in the event loop. This happens even if the awaited value is an already-resolved promise or not a promise. For example, consider the following code:

WithoutAwait.js

```
async function foo(name) {
  console.log(name, "start");
  console.log(name, "middle");
  console.log(name, "end");
}

foo("First");
foo("Second");

// First start
// First middle
// First end
// Second start
// Second middle
// Second end
```

With the introduction of one await,

Await.js

```
async function foo(name) {
  console.log(name, "start");
  await console.log(name, "middle");
  console.log(name, "end");
}

foo("First");
foo("Second");

// First start
// First middle
// Second start
// Second middle
// First end
// Second end
```

4 Lab Questions