Class 6: Integrating APIs with Asynchronous JavaScript

Session Overview

Class Topics:

- Asynchronous JavaScript
  - Callbacks
  - Promises
  - Async/Await
- Fetch API

Learning Objectives: By the end of this session, students will be able to:

- Understand and utilize callbacks, promises, and async/await in JavaScript.
- Use the Fetch API to make network requests and handle responses.
- Integrate data from an E-commerce API to display products dynamically on a web page.

Asynchronous JavaScript enables operations to be performed without blocking the main thread. This is crucial for keeping the UI responsive while performing tasks such as data fetching or timers. The primary techniques for handling asynchronous operations are:

1. **Callbacks**
2. **Promises**
3. **Async/Await**
4. **Fetch API**

1. Callbacks

Definition: A callback function is passed as an argument to another function and is executed once the other function completes its task. Callbacks are often used for handling asynchronous operations.

```
function fetchDataCallback(url, callback) {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url, true);
    xhr.onload = function() {
        if (xhr.status >= 200 && xhr.status < 300) {
            callback(null, JSON.parse(xhr.responseText));
        } else {
            callback(xhr.statusText, null);
        }
    };
```

```
    xhr.onerror = function() {
        callback(xhr.statusText, null);
    };
    xhr.send();
}
```

Explanation:

- **Creating XMLHttpRequest**: Initializes an XMLHttpRequest object to make an HTTP request.
- **Handling Responses**: onload and onerror handle successful responses and errors.
- **Callback Execution**: The callback function is executed with the data or error.

Drawbacks:

- **Callback Hell**: Nested callbacks can make the code difficult to read and maintain.
- **Error Handling**: Error management becomes cumbersome with deeply nested callbacks.

---

2. Promises

Definition: A Promise is an object representing the eventual completion or failure of an asynchronous operation. It allows chaining of operations and is cleaner than callbacks.

```
function fetchDataPromise(url) {
    return new Promise((resolve, reject) => {
        const xhr = new XMLHttpRequest();
        xhr.open('GET', url, true);
        xhr.onload = function() {
            if (xhr.status >= 200 && xhr.status < 300) {
                resolve(JSON.parse(xhr.responseText));
            } else {
                reject(xhr.statusText);
            }
        };
        xhr.onerror = function() {
            reject(xhr.statusText);
        };
        xhr.send();
    });
}
```

Explanation:

- **Creating a Promise**: A Promise object is created with resolve and reject functions.

- **Handling Responses**: resolve is called on success, and reject is called on failure.
- **Chaining**: .then handles the successful outcome, while .catch deals with errors.

Advantages:

- **Chaining**: Promises can be chained to perform multiple asynchronous operations.
- **Error Handling**: More straightforward error handling compared to callbacks.

3. Async/Await

Definition: async and await provides a way to write asynchronous code that looks synchronous. They simplify working with promises and improve code readability.

```
async function fetchDataAsync(url) {
    try {
        const response = await fetch(url);
        if (!response.ok) {
            throw new Error('Network response was not ok ' +
response.statusText);
        }
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error fetching products with async/await:', error);
        throw error;
    }
}
```

Explanation:

- **Defining Async Function**: async functions return a promise and allow the use of await inside.
- **Using await**: await pauses execution until the promise is resolved or rejected.
- **Error Handling**: Errors are managed using try and catch, making the code easier to read.

Advantages:

- **Readability**: Code is more readable and easier to understand.
- **Error Handling**: Simplified with try and catch.

---

4. Fetch API

Definition: The Fetch API is a modern alternative to XMLHttpRequest for making HTTP requests. It returns promises and is more versatile and easier to use.

```
async function fetchDataUsingFetch(url) {
    try {
        const response = await fetch(url);
        if (!response.ok) {
            throw new Error('Network response was not ok ' +
response.statusText);
        }
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error fetching products with Fetch API:', error);
        throw error;
    }
}
```

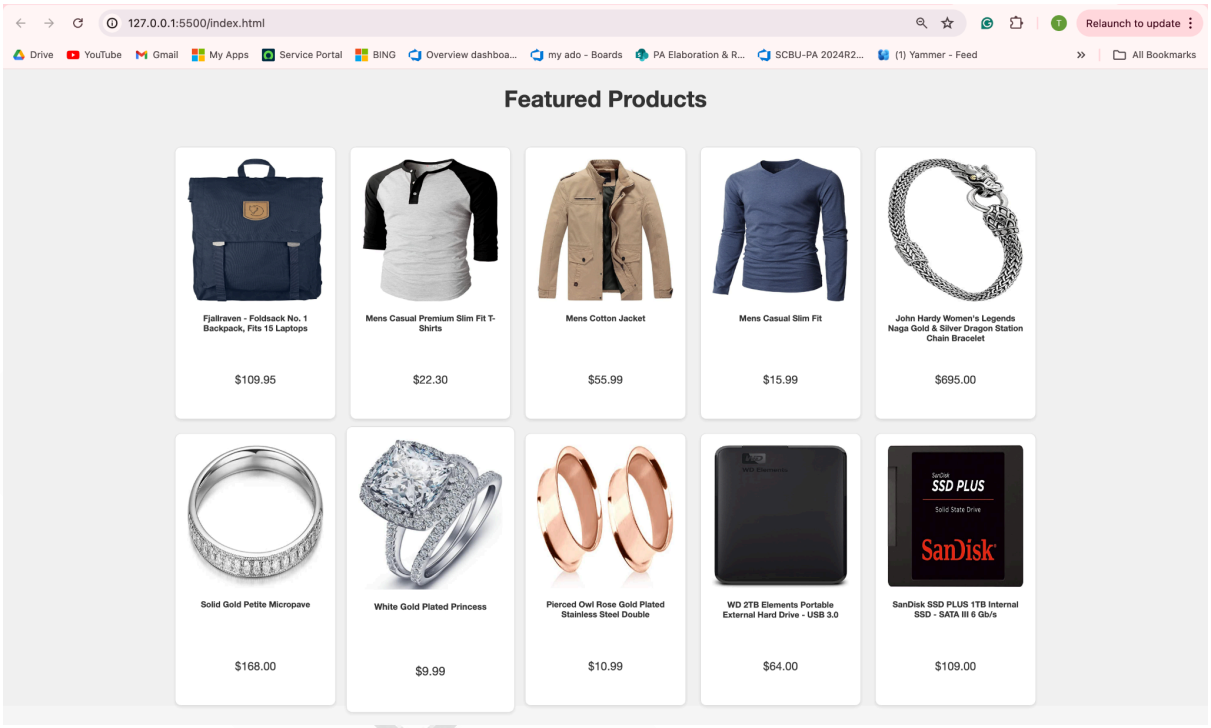
Explanation:

- **Using fetch**: fetch is used to make the HTTP request and return a promise.
- **Handling Responses**: response.ok, check if the response status is successful.
- **Parsing JSON**: response.json() converts the response to JSON format.

Advantages:

- **Modern API**: More modern and versatile compared to XMLHttpRequest.
- **Promises**: Uses promises, making it easier to work with asynchronous code.

Output Screenshot:



Interview and FAQ References

Asynchronous JavaScript Concepts

1. **Callbacks**
    - **Definition**: A function passed into another function as an argument, which is executed after the completion of the outer function's task.
    - **Common Issues**: This can lead to "callback hell," making code difficult to read and maintain.
2. **Promises**
    - **Definition**: An object representing the eventual completion (or failure) of an asynchronous operation and its resulting value.
    - **States**: Pending, Fulfilled, and Rejected.
    - **Chaining**: Allows chaining of .then() and .catch() methods for handling results and errors.
3. **Async/Await**
    - **Definition**: Syntactic sugar built on top of promises, allowing asynchronous code to be written in a synchronous style.
    - **Usage**: async functions return a promise, and await pauses execution until the promise is resolved or rejected.
4. **Fetch API**
    - **Definition**: A modern interface for making network requests, using promises to handle responses and errors.

- **Features**: More powerful and flexible compared to XMLHttpRequest, with support for promises and easier handling of JSON responses.

Interview Questions and Answers

Callbacks

1. **What is a callback function in JavaScript?**
   - **Answer**: A callback function is a function passed into another function as an argument, which is then executed after the outer function completes its execution. It's used to handle asynchronous operations.
2. **What are some problems associated with callbacks?**
   - **Answer**: Callbacks can lead to "callback hell," where nested callbacks become complex and hard to manage, resulting in code that is difficult to read and maintain.
3. **How can you avoid callback hell?**
   - **Answer**: By using promises or async/await, which provide a more structured way to handle asynchronous operations and avoid deeply nested callbacks.

Promises

1. **What is a promise in JavaScript?**
   - **Answer**: A promise is an object representing the eventual result of an asynchronous operation. It can be in one of three states: pending, fulfilled, or rejected.
2. **How do you handle success and failure with promises?**
   - **Answer**: Use .then() to handle successful fulfilment and .catch() to handle errors. Promises can also be chained to handle multiple asynchronous operations sequentially.
3. **What is the difference between .then() and .catch() in promises?**
   - **Answer**: .then() is used to handle the successful completion of a promise, while .catch() is used to handle any errors that occur. They can be chained to manage asynchronous workflows and error handling.

Async/Await

1. **What are async and await in JavaScript?**
   - **Answer**: async functions always return a promise. Inside an async function, await can be used to pause execution until a promise is resolved, making asynchronous code look more like synchronous code.
2. **How does async/await simplify working with promises?**
   - **Answer**: async/await provides a cleaner syntax for handling asynchronous operations, avoiding the need for chaining .then() and .catch(), and making the code easier to read and maintain.
3. **How do you handle errors in async/await?**
   - **Answer**: Errors can be handled using try...catch blocks within async functions, allowing for more straightforward error management compared to promise chaining.

Fetch API

1. **What is the Fetch API?**
    - **Answer**: The Fetch API is a modern interface for making network requests. It uses promises to handle responses and errors, providing a more flexible and powerful alternative to XMLHttpRequest.
2. **How does the Fetch API differ from XMLHttpRequest?**
    - **Answer**: The Fetch API provides a more straightforward and modern approach to handling network requests with promises and cleaner syntax. Unlike XMLHttpRequest, it does not require callback functions and supports chaining and async/await.
3. **What are some common methods used with the Fetch API?**
    - **Answer**: Common methods include fetch(), which is used to initiate a request, and methods like .json(), .text(), and .blob() for handling response data.
