Class-10: Working with APIs

Introduction

In modern web development, APIs (Application Programming Interfaces) are essential for fetching and manipulating data. This class covers **fetching data using fetch and axios, handling API errors, and displaying loading states effectively**.

Fetching Data with fetch

What is fetch?

The fetch API is a built-in JavaScript method for making network requests. It returns a Promise that resolves with the response data.

Syntax

```
fetch(url)
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error("Error fetching data:", error));
```

Example: Fetching Data in React

```
import React, { useState, useEffect } from 'react';

const FetchDataComponent = () => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then(response => {
        if (!response.ok) {
          throw new Error("Network response was not ok");
        }
        return response.json();
      })
      .then(data => {
        setData(data);
        setLoading(false);
```

```
    })
    .catch(error => {
      setError(error.message);
      setLoading(false);
    });
  }, []);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error}</p>;

  return (
    <ul>
      {data.map(item => (
        <li key={item.id}>{item.title}</li>
      ))}
    </ul>
  );
};

export default FetchDataComponent;
```

Fetching Data with axios

What is axios?

axios is a popular HTTP client for making requests. It simplifies API calls and provides built-in error handling.

Installing axios

npm install axios

Syntax

```
axios.get(url)
  .then(response => console.log(response.data))
  .catch(error => console.error("Error fetching data:", error));
```

Example: Fetching Data in React with axios

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const AxiosFetchComponent = () => {
  const [data, setData] = useState([]);
```

```
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get("https://jsonplaceholder.typicode.com/posts")
      .then(response => {
        setData(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error.message);
        setLoading(false);
      });
  }, []);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error}</p>;

  return (
    <ul>
      {data.map(item => (
        <li key={item.id}>{item.title}</li>
      ))}
    </ul>
  );
};

export default AxiosFetchComponent;
```

Why Use axios Over fetch?

Feature	fetch	axios
Response Handling	Requires .json() conversion	Auto-converts response data
Error Handling	Must manually check response.ok	Automatically rejects on errors
Request Configuration	More complex	Simplified syntax

Handling API Errors

Handling errors properly ensures a smooth user experience.

Common Errors and Solutions

Error Type	Cause	Solution
Network Error	No internet connection	Show a user-friendly message
Server Error (500)	Issue on the API server	Display an error message and retry
Not Found (404)	Requested resource does not exist	Show a 'Not Found' message
CORS Error	Cross-Origin restriction	Ensure API supports CORS or use a proxy

Example: Centralized Error Handling with axios

```
const api = axios.create({
  baseURL: "https://jsonplaceholder.typicode.com",
});

api.interceptors.response.use(
  response => response,
  error => {
    console.error("API Error:", error);
    return Promise.reject(error);
  }
);
```

Displaying Loading States

Showing a loading indicator improves user experience while waiting for API responses.

Example: Adding a Loading Spinner

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const LoadingComponent = () => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get("https://jsonplaceholder.typicode.com/posts")
      .then(response => {
        setData(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error.message);
```

```
      setLoading(false);
    });
  }, []);

  return (
    <div>
      {loading && <p>Loading...</p>}
      {error && <p>Error: {error}</p>}
      <ul>
        {data.map(item => (
          <li key={item.id}>{item.title}</li>
        ))}
      </ul>
    </div>
  );
};

export default LoadingComponent;
```

Best Practices for Loading States

- **Show a spinner or skeleton UI** while fetching data.
- **Disable buttons** or prevent user interactions during loading.
- **Provide feedback** if loading takes longer than expected.

Summary

- **fetch vs axios**: axios provides better error handling and simpler syntax.
- **Error Handling**: Catch errors and show meaningful messages.
- **Loading States**: Improve user experience by indicating when data is being fetched.

By following these best practices, developers can create robust React applications that handle API data efficiently.