Lesson 07 Demo 02

API Calls Using Error Handling

Objective: To develop a React application that demonstrates error handling in API calls with React

Tools Required: Node Terminal, React App, and Visual Studio Code

Prerequisites: Knowledge of creating a React app and understanding of the folder structure

Steps to be followed:

1. Create a new React app
2. Modify the **App.js** file in the **src** directory
3. Run the app

Step 1: Create a new React app

1.1 Open the terminal and run the command **npx create-react-app error-demo**

This command will create a new **React** app with the name **error-demo.**

1.2 Move to the newly created directory by running the command **cd error-demo** in the terminal

Step 2: Modify the App.js file in the src directory

2.1 Open your React project with **Visual Studio Code** and navigate to the **src/App.js** file to define a state variable **error** and function **fetchData**

2.2 Within the **App.js** file, import **React**, **useState** and **useEffect** hooks

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

2.3 Define the **App** functional component, and inside the component, define a state variable called **error** using the **useState** hook, and initialize it as **null**

```
function App() {
const [error, setError] = useState(null);
```

> **Note:** The state variable and the function make an **API** call using **fetch** and update the **state** with the response data or the error message. We will use a **try...catch block** to catch any errors that occur during the **API** call.

2.4 Define an asynchronous function called **fetchData** that makes an **API** call using the **fetch** function and updates the state with the response data or the error message

```
async function fetchData() {
```

2.5 Use a **try...catch block** to catch any errors that occur during the **API** call. If an error occurs, set the **error** state to the error message

```
async function fetchData() {
try {
const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');
const jsonData = await response.json();
console.log(jsonData);
} catch (err) {
setError(err.message);
}
}
```

2.6 Use the **useEffect** hook to call the **fetchData** function when the component mounts by passing an empty **dependency array []** as the second argument to **useEffect**

```
useEffect(() => {
fetchData();
}, []);
```

2.7 In the **return** statement of the **App** component, render a **div** with the **className=" App"**. Inside the **div**, use a conditional rendering to display the error message if the error state is not null. If the **error** state is null, render a **p** element with the text **Loading...** and export the **App** component as the default export

```
return (
<div className="App">
<h1>Error Demo</h1>
{error ? (
<p>Error: {error}</p>
) : (
<p>Loading...</p>
)}
</div>
);
}
Run the app

export default App;
```

Note: Refer to the following code to configure the **App.js** file:

```
import React, { useState, useEffect } from 'react';
import './App.css';

function App() {
const [error, setError] = useState(null);

async function fetchData() {
try {
const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');
const jsonData = await response.json();
console.log(jsonData);
```

```
} catch (err) {
setError(err.message);
}
}

useEffect(() => {
fetchData();
}, []);

return (
<div className="App">
<h1>Error Demo</h1>
{error ? (
<p>Error: {error}</p>
) : (
<p>Loading...</p>
)}
</div>
);
}

export default App;
```

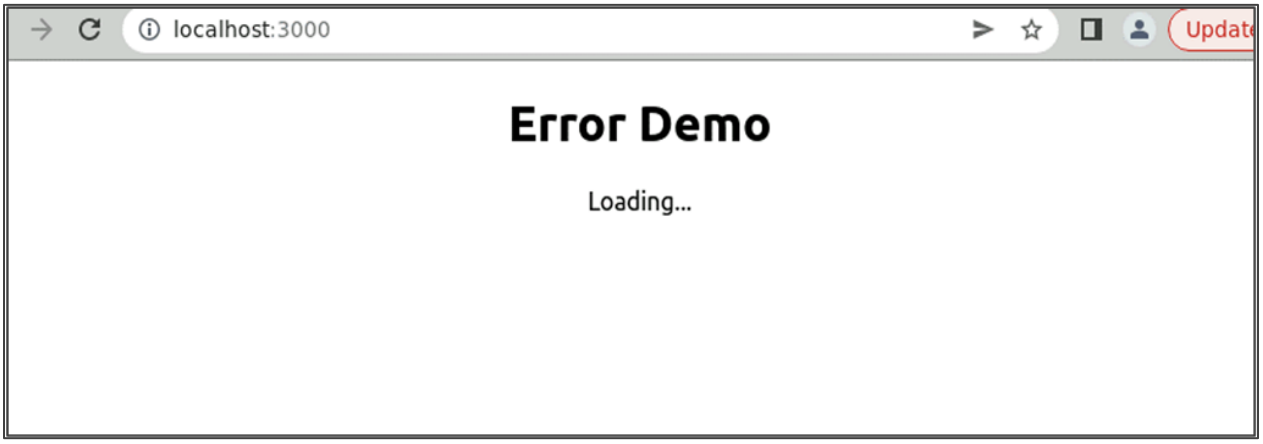
src > JS App.js > ...

```
1  import React, { useState, useEffect } from 'react';
2  import './App.css';
3
4  function App() {
5  const [error, setError] = useState(null);
6
7  async function fetchData() {
8  try {
9  const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');
10 const jsonData = await response.json();
11 console.log(jsonData);
12 } catch (err) {
13 setError(err.message);
14 }
15 }
16
17 useEffect(() => {
18 fetchData();
19 }, []);
20
21 return (
22 <div className="App">
23 <h1>Error Demo</h1>
24 {error ? (
25 <p>Error: {error}</p>
26 ) : (
27 <p>Loading...</p>
28 )}
29 </div>
30 );
31 }
32
33 export default App;
34
```

Step 3: Run the app

3.1 In the terminal, navigate to the project directory and run the command **npm start** to start the development server

3.2 Open your browser and navigate to **http://localhost:3000**



You should see a simple app that displays a loading message while the **API** call is being made and then displays any error message that occurs during the **API** call.

With this, you have successfully created an application to demonstrate error handling in API calls while working on React.