# Handling HTTP Requests in React

In this chapter, we will learn how to handle **HTTP requests** in React applications, which is essential for fetching data from APIs or sending data to a server.

## Making HTTP Requests with axios

Many developers prefer **axios** for its cleaner syntax and automatic handling of JSON responses. Axios is a promise-based HTTP client for the browser and Node.js.

#### Install Axios

npm install axios

#### Example: Making a GET Request with axios

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

import axios from 'axios';

function App() {

const [data, setData] = useState(null);

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 (

<div>

<h1>Posts</h1>

<ul>

{data.map((post) => (

<li key={post.id}>{post.title}</li>

))}

</ul>

</div>

);

}

export default App;

### Key Differences Between fetch and axios:

* **Error Handling**: In fetch, you need to manually check if the response is successful (i.e., response.ok). In contrast, axios automatically handles errors for HTTP status codes outside the range 200–299.
* **Parsing JSON**: fetch requires manually parsing the response with .json(), whereas axios automatically parses the response as JSON.

## Handling Asynchronous Operations and Rendering Data

React apps often deal with asynchronous operations, such as fetching data from a server. This is done using **Promises** (like the one returned by fetch and axios), and React’s state is updated once the data is retrieved.

### Example: Fetching Data Asynchronously in React

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

function App() {

const [data, setData] = useState([]);

const [loading, setLoading] = useState(true);

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

// useEffect will run once after the initial render

useEffect(() => {

async function fetchData() {

try {

const response = await fetch('https://jsonplaceholder.typicode.com/posts');

if (!response.ok) {

throw new Error('Failed to fetch data');

}

const data = await response.json();

setData(data); // Update state with the fetched data

setLoading(false); // Set loading to false once data is fetched

} catch (error) {

setError(error.message);

setLoading(false);

}

}

fetchData();

}, []); // Empty array ensures this effect runs only once after the initial render

if (loading) {

return <p>Loading...</p>;

}

if (error) {

return <p>Error: {error}</p>;

}

return (

<div>

<h1>Posts</h1>

<ul>

{data.map((post) => (

<li key={post.id}>{post.title}</li>

))}

</ul>

</div>

);

}

export default App;

### Explanation:

1. **async/await**: We use async/await syntax for a cleaner, more readable way to handle asynchronous operations.
2. **State management**: The state (data, loading, error) is updated based on the result of the fetch operation.
3. **Rendering data**: Once the data is successfully fetched, it is displayed by mapping through the array of posts.

## 3. Error Handling and Loading States

Handling loading states and errors properly is crucial for a smooth user experience, especially when fetching data from APIs.

### Error Handling

React applications can run into various errors when dealing with HTTP requests, such as:

* Server errors (e.g., 500 Internal Server Error).
* Network issues (e.g., no internet connection).
* Unexpected response formats (e.g., invalid JSON).

You can handle errors using try/catch blocks (with async/await) or .catch() method with Promises.

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

import axios from 'axios';

function App() {

const [data, setData] = useState(null);

const [loading, setLoading] = useState(true);

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

const [newPost, setNewPost] = useState({

title: '',

body: ''

});

useEffect(() => {

// Fetching data (GET request)

axios

.get('https://jsonplaceholder.typicode.com/posts')

.then((response) => {

setData(response.data);

setLoading(false);

})

.catch((error) => {

handleError(error);

});

}, []);

// Handle errors

const handleError = (error) => {

if (error.response) {

setError(`Server responded with status: ${error.response.status}`);

} else if (error.request) {

setError('Network error: No response from the server');

} else {

setError(`Error: ${error.message}`);

}

setLoading(false);

};

// Handle POST request

const handlePost = () => {

setLoading(true);

axios

.post('https://jsonplaceholder.typicode.com/posts', {

title: newPost.title,

body: newPost.body,

})

.then((response) => {

setData((prevData) => [...prevData, response.data]);

setLoading(false);

setNewPost({ title: '', body: '' }); // Reset form after post

})

.catch((error) => handleError(error));

};

// Handle PUT request

const handlePut = (postId) => {

setLoading(true);

axios

.put(`https://jsonplaceholder.typicode.com/posts/${postId}`, {

title: 'Updated Title', // Example title

body: 'Updated Body', // Example body

})

.then((response) => {

const updatedData = data.map((post) =>

post.id === postId ? { ...post, ...response.data } : post

);

setData(updatedData);

setLoading(false);

})

.catch((error) => handleError(error));

};

// Handle DELETE request

const handleDelete = (postId) => {

setLoading(true);

axios

.delete(`https://jsonplaceholder.typicode.com/posts/${postId}`)

.then(() => {

const filteredData = data.filter((post) => post.id !== postId);

setData(filteredData);

setLoading(false);

})

.catch((error) => handleError(error));

};

if (loading) {

return <p>Loading...</p>;

}

if (error) {

return <p>Error: {error}</p>;

}

return (

<div>

<h1>Posts</h1>

<ul>

{data.map((post) => (

<li key={post.id}>

{post.title}

<button onClick={() => handlePut(post.id)}>Update</button>

<button onClick={() => handleDelete(post.id)}>Delete</button>

</li>

))}

</ul>

<h2>Create a new post</h2>

<form

onSubmit={(e) => {

e.preventDefault();

handlePost();

}}

>

<input

type="text"

placeholder="Title"

value={newPost.title}

onChange={(e) => setNewPost({ ...newPost, title: e.target.value })}

/>

<textarea

placeholder="Body"

value={newPost.body}

onChange={(e) => setNewPost({ ...newPost, body: e.target.value })}

/>

<button type="submit">Submit</button>

</form>

</div>

);

}

export default App;