Consuming REST APIs In React With Fetch And Axios

Consuming REST APIs in a React Application can be done in various ways, but in this tutorial, we will be discussing how we can consume REST APIs using two of the most popular methods known as Axios (a promise-based HTTP client) and Fetch API (a browser in-built web API). I will discuss and implement each of these methods in detail and shed light on some of the cool features each of them have to offer.

n React, there are various ways we can consume REST APIs in our applications, these ways include using the JavaScript inbuilt fetch() method and Axios which is a promise-based HTTP client for the browser and Node.js.

**What Is A REST API #**

A REST API is an API that follows what is structured in accordance with the REST Structure for APIs. REST stands for “Representational State Transfer”. It consists of various rules that developers follow when creating APIs.

**Consuming APIs Using The Fetch API #**

The fetch() API is an inbuilt JavaScript method for getting resources from a server or an API endpoint. It’s similar to XMLHttpRequest, but the fetch API provides a more powerful and flexible feature set.

It defines concepts such as CORS and the HTTP Origin header semantics, supplanting their separate definitions elsewhere.

The fetch() API method always takes in a compulsory argument, which is the path or URL to the resource you want to fetch. It returns a promise that points to the response from the request, whether the request is successful or not. You can also optionally pass in an init options object as the second argument.

**THE DIFFERENCE BETWEEN THE FETCH API AND JQUERY AJAX #**

The Fetch API is different from jQuery Ajax in three main ways, which are:

The promise returned from a fetch() request will not reject when there’s an HTTP error, no matter the nature of the response status. Instead, it will resolve the request normally, if the response status code is a 400 or 500 type code, it’ll set the ok status. A request will only be rejected either because of network failure or if something is preventing the request from completing

fetch() will not allow the use of cross-site cookies i.e you cannot carry out a cross-site session using fetch()

fetch() will also not send cookies by default unless you set the credentials in the init option.

**PARAMETERS FOR THE FETCH API #**

**resource**

This is the path to the resource you want to fetch, this can either be a direct link to the resource path or a request object

**init**

This is an object containing any custom setting or credentials you’ll like to provide for your fetch() request. The following are a few of the possible options that can be contained in the init object:

**method**

This is for specifying the HTTP request method e.g GET, POST, etc.

**headers**

This is for specifying any headers you would like to add to your request, usually contained in an object or an object literal.

**body**

This is for specifying a body that you want to add to your request: this can be a Blob, BufferSource, FormData, URLSearchParams, USVString, or ReadableStream object

**mode**

This is for specifying the mode you want to use for the request, e.g., cors, no-cors, or same-origin.

**credentials**

This for specifying the request credentials you want to use for the request, this option must be provided if you consider sending cookies automatically for the current domain.

**BASIC SYNTAX FOR USING THE FETCH() API #**

A basic fetch request is really simple to write, take a look at the following code:

fetch('https://api.github.com/users/hacktivist123/repos')

.then(response => response.json())

.then(data => console.log(data));

Copy

In the code above, we are fetching data from a URL that returns data as JSON and then printing it to the console. The simplest form of using fetch() often takes just one argument which is the path to the resource you want to fetch and then return a promise containing the response from the fetch request. This response is an object.

**USING FETCH API IN REACT APPS #**

import React from 'react';

class myComponent extends React.Component {

componentDidMount() {

const apiUrl = 'https://api.github.com/users/hacktivist123/repos';

fetch(apiUrl)

.then((response) => response.json())

.then((data) => console.log('This is your data', data));

}

render() {

return <h1>my Component has Mounted, Check the browser 'console' </h1>;

}

}

export default myComponent;

**Let’s Consume A REST API With Fetch Method #**

In this section, we will be building a simple react application that consumes an external API, we will be using the Fetch method to consume the API.

The simple application will display all the repositories and their description that belongs to a particular user. For this tutorial, I’ll be using my GitHub username, you can also use yours if you wish.

The first thing we need to do is to generate our React app by using create-react-app:

**npx create-react-app myRepos**

The command above will bootstrap a new React app for us. As soon as our new app has been created, all that’s left to do is to run the following command and begin coding:

**npm start**

If our React is created properly we should see this in our browser window when we navigate to localhost:3000 after running the above command.

**Initial App Screen**

(Large preview)

In your src folder, create a new folder called component. This folder will hold all of our React components. In the new folder, create two files titled List.js and withListLoading.js. These two files will hold the components that will be needed in our app.

The List.js file will handle the display of our Repositories in the form of a list, and the withListLoading.js file will hold a higher-order component that will be displayed when the Fetch request we will be making is still ongoing.

In the List.js file we created inside the components folder, let’s paste in the following code:

import React from 'react';

const List = (props) => {

const { repos } = props;

if (!repos || repos.length === 0) return <p>No repos, sorry</p>;

return (

<ul>

<h2 className='list-head'>Available Public Repositories</h2>

{repos.map((repo) => {

return (

<li key={repo.id} className='list'>

<span className='repo-text'>{repo.name} </span>

<span className='repo-description'>{repo.description}</span>

</li>

);

})}

</ul>

);

};

The code above is a basic React list component that would display the data, in this case, the repositories name and their descriptions in a list.

Now, Let me explain the code bit by bit.

**const { repos } = props;**

We are initializing a prop for the component called repos.

Here, all we are doing is making a conditional statement that will render a message when the length of the repos we get from the request we make is equal to zero.

return (

<ul>

<h2 className='list-head'>Available Public Repositories</h2>

{repos.map((repo) => {

return (

<li key={repo.id} className='list'>

<span className='repo-text'>{repo.name} </span>

<span className='repo-description'>{repo.description}</span>

</li>

);

})}

</ul>

);

Here, we are mapping through each of the repositories that will be provided by the API request we make and extracting each of the repositories names and their descriptions then we are displaying each of them in a list.

Here we are exporting our List component so that we can use it somewhere else.

In the withListLoading.js file we created inside the components folder, let’s paste in the following code:

import React from 'react';

function WithListLoading(Component) {

return function WihLoadingComponent({ isLoading, ...props }) {

if (!isLoading) return <Component {...props} />;

return (

<p style={{ textAlign: 'center', fontSize: '30px' }}>

Hold on, fetching data may take some time :)

</p>

);

};

}

export default WithListLoading;

The code above is a higher-order React component that takes in another component and then returns some logic. In our case, our higher component will wait to check if the current isLoading state of the component it takes is true or false. If the current isLoading state is true, it will display a message Hold on, fetching data may take some time :). Immediately the isLoading state changes to false it’ll render the component it took in. In our case, it’ll render the List component.

**In your \*App.js file inside the src folder, let’s paste in the following code:**

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

import './App.css';

import List from './components/List';

import withListLoading from './components/withListLoading';

function App() {

const ListLoading = withListLoading(List);

const [appState, setAppState] = useState({

loading: false,

repos: null,

});

useEffect(() => {

setAppState({ loading: true });

const apiUrl = `https://api.github.com/users/hacktivist123/repos`;

fetch(apiUrl)

.then((res) => res.json())

.then((repos) => {

setAppState({ loading: false, repos: repos });

});

}, [setAppState]);

return (

<div className='App'>

<div className='container'>

<h1>My Repositories</h1>

</div>

<div className='repo-container'>

<ListLoading isLoading={appState.loading} repos={appState.repos} />

</div>

<footer>

<div className='footer'>

Built{' '}

<span role='img' aria-label='love'>

💚

</span>{' '}

with by Shedrack Akintayo

</div>

</footer>

</div>

);

}

export default App;

Our App.js is a functional component that makes use of React Hooks for handling state and also side effects. If you’re not familiar with React Hooks, read my Getting Started with React Hooks Guide.

Let me explain the code above bit by bit.

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

import './App.css';

import List from './components/List';

import withListLoading from './components/withListLoading';

Here, we are importing all the external files we need and also the components we created in our components folder. We are also importing the React Hooks we need from React.

const ListLoading = withListLoading(List);

const [appState, setAppState] = useState({

loading: false,

repos: null,

});

Here, we are creating a new component called ListLoading and assigning our withListLoading higher-order component wrapped around our list component. We are then creating our state values loading and repos using the useState() React Hook.

useEffect(() => {

setAppState({ loading: true });

const user = `https://api.github.com/users/hacktivist123/repos`;

fetch(user)

.then((res) => res.json())

.then((repos) => {

setAppState({ loading: false, repos: repos });

});

}, [setAppState]);

Here, we are initializing a useEffect() React Hook. In the useEffect() hook, we are setting our initial loading state to true, while this is true, our higher-order component will display a message. We are then creating a constant variable called user and assigning the API URL we’ll be getting the repositories data from.

We are then making a basic fetch() request like we discussed above and then after the request is done we are setting the app loading state to false and populating the repos state with the data we got from the request.

return (

<div className='App'>

<div className='container'>

<h1>My Repositories</h1>

</div>

<div className='repo-container'>

<ListLoading isLoading={AppState.loading} repos={AppState.repos} />

</div>

</div>

);

}

export default App;

Here we are basically just rendering the Component we assigned our higher-order component to and also filling the isLoading prop and repos prop with their state value.

Now, we should see this in our browser, when the fetch request is still being made, courtesy of our withListLoading higher-order component

**Consuming APIs With Axios #**

Axios is an easy to use promise-based HTTP client for the browser and node.js. Since Axios is promise-based, we can take advantage of async and await for more readable and asynchronous code. With Axios, we get the ability to intercept and cancel request, it also has a built-in feature that provides client-side protection against cross-site request forgery.

**FEATURES OF AXIOS #**

Request and response interception

Streamlined error handling

Protection against XSRF

Support for upload progress

Response timeout

The ability to cancel requests

Support for older browsers

Automatic JSON data transformation

**MAKING REQUESTS WITH AXIOS #**

Making HTTP Requests with Axios is quite easy. The code below is basically how to make an HTTP request.

The code above shows the basic ways we can make a GET and POST HTTP request with Axios.

Axios also provides a set of shorthand method for performing different HTTP requests. The Methods are as follows:

axios.request(config)

axios.get(url[, config])

axios.delete(url[, config])

axios.head(url[, config])

axios.options(url[, config])

axios.post(url[, data[, config]])

axios.put(url[, data[, config]])

axios.patch(url[, data[, config]])

For example, if we want to make a similar request like the example code above but with the shorthand methods we can do it like so:

// Make a GET request with a shorthand method

axios.get('https://api.github.com/users/hacktivist123');

// Make a Post Request with a shorthand method

axios.post('/signup', {

firstName: 'shedrack',

lastName: 'akintayo'

});

In the code above, we are making the same request as what we did above but this time with the shorthand method. Axios provides flexibility and makes your HTTP requests even more readable.

**MAKING MULTIPLE REQUESTS WITH AXIOS #**

Axios provides developers the ability to make and handle simultaneous HTTP requests using the axios.all() method. This method takes in an array of arguments and it returns a single promise object that resolves only when all arguments passed in the array have resolved.

For example, we can make multiple requests to the GitHub api using the axios.all() method like so

axios.all([

axios.get('https://api.github.com/users/hacktivist123'),

axios.get('https://api.github.com/users/adenekan41')

])

.then(response => {

console.log('Date created: ', response[0].data.created\_at);

console.log('Date created: ', response[1].data.created\_at);

});

**Let’s Consume A REST API With Axios Client #**

In this section, all we’ll be doing is replacing fetch() method with Axios in our existing React Application. All we need to do is to install Axios and then use it in our App.js file for making the HTTP request to the GitHub API.

Now let’s install Axios in our React app by running either of the following:

With NPM:

**npm install axios**

With Yarn:

**yarn add axios**

After installation is complete, we have to import axios into our App.js. In our App.js we’ll add the following line to the top of our App.js file:

import axios from 'axios'

After adding the line of code our App.js all we have to do inside our useEffect() is to write the following code:

useEffect(() => {

setAppState({ loading: true });

const apiUrl = 'https://api.github.com/users/hacktivist123/repos';

axios.get(apiUrl).then((repos) => {

const allRepos = repos.data;

setAppState({ loading: false, repos: allRepos });

});

}, [setAppState]);

You may have noticed that we have now replaced the fetch API with the Axios shorthand method axios.get to make a get request to the API.

axios.get(apiUrl).then((repos) => {

const allRepos = repos.data;

setAppState({ loading: false, repos: allRepos });

});

In this block of code, we are making a GET request then we are returning a promise that contains the repos data and assigning the data to a constant variable called allRepos. We are then setting the current loading state to false and also passing the data from the request to the repos state variable.

If we did everything correctly, we should see our app still render the same way without any change.