Thursday and Friday of week 41

I didn’t get any useful information from my school to get the id for this task

<https://openweathermap.org/faq#error401>

<https://www.freecodecamp.org/news/how-to-use-axios-with-react/>

## How to obtain access to the Free Data for Students

If you are a student or a university scholar, you may obtain access to these packages by doing the following:

1. [**Sign up**](https://home.openweathermap.org/users/sign_up?student=true) using your school-issued email or [**login**](https://home.openweathermap.org/users/sign_in?student=true) to OpenWeather using an existing OpenWeather account.  
   (No credit card or payment information required)
2. After a successful registration, you will be transferred to the student form automatically. Please, prepare the following information to fill in the form:
   * Full name and the name of educational institution
   * Student ID and its expiration date
   * Your project description (please note, that a more thorough description of your project would facilitate the processing of your application)
3. Once you submit the student form, your application will be taken for validation. You will receive an email as a confirmation of the student access activation.

[Get access](https://home.openweathermap.org/users/sign_in?student=true)

If you are an educator or speaking on university/college behalf, please contact us directly via [info@openweathermap.org.](mailto:info@openweathermap.org)

## What we ask in return

1. **Share your results**: Once you finish using OpenWeather's data, you are welcome to share the results of your work or research project with us and our community. You could provide us with a research article, link to the publication or any other material to contribute to our initiative.
2. **Branding:** We welcome any attribution of OpenWeather in your research project. You can use OpenWeather’s logo or link to our [website](https://openweathermap.org/) upon making your work or research project public.
3. **Do not share your API key**: Please, do not share the API key with anyone who doesn’t take part in your project. If you are an educator or a university representative, we would invite you to [ask us](mailto:info@openweathermap.org) for a consultation on how to provide your students with access to our data.

## FAQ

How long does the activation process take, after submitting my request for the free data for students?

I have obtained access to the free data for students, but I don’t know how to use it.

What should I do if my 6-months access to the free data expired, but my project is not yet finished?

May I still access the free data for students once my research project is over?

Can I have access to free data for students if my studentship expires sooner, than in 6 months?

What should I do if I receive service errors?

Can I share the API key to my friends outside the university?

Can I use the free data for students in the development of my commercial projects?

I already have a paid account. Can I still get a free license?

Do you have any more questions? Please, contact us via [info@openweathermap.org](mailto:info@openweathermap.org).

npm install axios

**# Group Activity**

> Work in group to solve the task below.

> The complete source code is in` App.js` file. You can compare your work at the end of the activity.

**## Exercise**

We will use the useState hook to create a controlled form component. At the end of this exercise, you should have the outcome shown in the `weather.png` and be able to type a city name and display its current temperature from [Openweather API](https://openweathermap.org/).

![](./weather.png "")

1. To get started, first install create-react-app and then follow these steps:

2. Select all the files in the src folder and remove them all. We are going to create our own files.

3. Create a new file called index.js in the src folder and add the following code:

```js

import ReactDOM from 'react-dom/client'

import App from './App'

ReactDOM.createRoot(document.getElementById('root')).render(<App />)

```

4. let's create a CSS file and add styles. In the src folder, create a file called App.css and add the following code:

```css

body {

  margin: 0;

}

.page {

  align-items: center;

  display: flex;

  height: 100vh;

  justify-content: center;

  width: 100vw;

}

.box {

  display: flex;

  flex-direction: column;

}

.weather-button {

  display: block;

  margin: 20px auto 0;

}

.temp {

  padding: 20px;

  text-align: center;

}

```

5. In the src folder, create another file called App.js and import React and the CSS file we created earlier. Also add some boilerplate code:

```js

import './App.css';

const App = () => {

  return <div>App page</div>;

};

export default App;

```

6. Let's add JSX code to display the form and the initial temperature, which will be 0. First, add the page wrapper with the class name page. Inside the page wrapper, add another wrapper called box. The box wrapper will align the form and the output of the temperature in the middle horizontally and vertically. Inside the box wrapper, add a form (with the class name weather-form) with an input field and a button (with the class name weather-button). Below the form, add a div element with the class name temp:

```js

import './App.css';

const App = () => {

  return (

  <div className="page">

   <div className="box">

    <form className="weather-form">

      <label htmlFor="city">City name: </label>

          <input type="text" id="city" placeholder="Type a city" /></label>

      <button className="weather-button">Get temperature</button>

    </form>

    <div className="temp">

      0 &#8451;

    </div>

  </div>

</div>

  );

};

export default App;

```

7. Before we fetch any data, let's create a controlled form component, to get the city name from the input field. By using the controlled form component, our React components will control the input field and store the value from the input field to a state.

8. To do that, use the useState hook. Let's import the hook first and then declare a new state variable called city.

9. Update the city value when we make changes in the input field, and we are going to use the updated city value for the input field value. This step will allow you to update the city value while typing text in the input field so that we can use it when fetching data:

```js

import React, { useState } from 'react';

import './App.css';

const App = () => {

  const [city, setCity] = useState('');

  return (

  <div className="page">

  <div className="box">

    <form className="weather-form">

      <label htmlFor="city">City name: </label>

           <input type="text" id="city" placeholder="Type a city" value={city} onChange={e => setCity(e.target.value)} />

      <button className="weather-button">Get temperature</button>

    </form>

    <div className="temp">

      0 &#8451;

    </div>

  </div>

  </div>

  );

};

export default App;

```

10.

- To fetch data, install Axios by running  `npm install axios` command from your terminal,

- To use Axios, let's import the library: `import axios from 'axios';`

11. Add the onSubmit attribute to the form element and add a reference to a function called submitForm. We are going to fetch data when submitting the form:

```html

<form onSubmit={submitForm} className="weather-form">

```

12. Create new function called submitForm above the return method and receive the event. Firstly, add e.preventDefault();. By default, when you click on the submit button in a form, the browser will be reloaded. When the browser is reloaded, our React app will be restarted, which means all our previously stored values in a state will be lost. To prevent this from happening, we are going to add e.preventDefault();:

```js

const submitForm = e => {

  e.preventDefault();

};

```

13. Now let's fetch data. To get the current temperature of the city, we are going to use the API from https://openweathermap.org/current. So, the API should look like this:

https://api.openweathermap.org/data/2.5/weather?

  q=city-name

  &appid=your-api-key

  &units=metric

We are going to send three values parameter values:

- q: The city name that we will get from the city state updated by setCity

- appid: Your API key, which can be found on the API keys page.

- units=metric: Receiving values in the Celsius scale.

Now we can get the city name so the endpoint should look like this:

https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=your-api-key&units=metric

> If you don't have an api key, you can register or use my key for this lab:

https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=35e7798da3a6b15be336c4db70f936b4&units=metric

14. Let's fetch the data using Axios. Use the GET method, and let's see what data we are getting back:

```js

const submitForm = e => {

  e.preventDefault();

  const url = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=your-api-key&units=metric`;

  axios.get(url)

    .then(res => {

        console.log(res);

        setTemp(res.data.main.temp);

  })

};

```

15. We can retrieve the temperature from data.main.temp.

Declare a new state called temp with the default value 0, and update it in then() method. Finally, let's replace 0 to {temp} to display the temperature:

16. Let's search Helsinki to get the current temperature.

**## Ref**

- The React Workshop: Brandon Richey etc.

**# Group Activity**

> Work in group to solve the task below.

**## Exercise 1**

Visit the website below, and test how to:

- How to Make a GET Request

- How to Make a POST Request

- How to Make a PUT Request

- How to Make a DELETE Request

**## Exercise 2**

- Modify the code so that  you refer to the state and effect hooks without prepending React i.e.

- `React.useState(null)` to `useState(null)`

- `React.useEffect(()` to `useEffect()`

**## Links**

- [How To Use Axios With React: The Definitive Guide](<https://www.freecodecamp.org/news/how-to-use-axios-with-react/>)

Exercise 2

import React, { useState } from 'react';

import axios from 'axios';

import './App.css';

const App = () => {

  const [city, setCity] = useState('');

  const [temp, setTemp] = useState(0);

// ----------------------------------------------------------------------------------------------------------------------------

## How to Make a GET Request

## READ

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts/1";

export default function App() {

  const [post, setPost] = React.useState(null);

  React.useEffect(() => {

    axios.get(baseURL).then((response) => {

      setPost(response.data);

    });

  }, []);

  if (!post) return null;

  return (

    <div>

      <h1>{post.title}</h1>

      <p>{post.body}</p>

    </div>

  );

}

# sunt aut facere repellat provident occaecati excepturi optio reprehenderit

quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto

## How to Make a POST Request

## CREATE

// -----------------------------------------------------

// How to Make a POST Request

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

  const [post, setPost] = React.useState(null);

  React.useEffect(() => {

    axios.get(`${baseURL}/1`).then((response) => {

      setPost(response.data);

    });

  }, []);

  function createPost() {

    axios

      .post(baseURL, {

        title: "Hello World!",

        body: "This is a new post."

      })

      .then((response) => {

        setPost(response.data);

      });

  }

  if (!post) return "No post!"

  return (

    <div>

      <h1>{post.title}</h1>

      <p>{post.body}</p>

      <button onClick={createPost}>Create Post</button>

    </div>

  );

}

## PRESS button and

# Hello World!

This is a new post.

## How to Make a PUT Request

## UPDATE

// -----------------------------------------------

// How to Make a PUT Request

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

  const [post, setPost] = React.useState(null);

  React.useEffect(() => {

    axios.get(`${baseURL}/1`).then((response) => {

      setPost(response.data);

    });

  }, []);

  function updatePost() {

    axios

      .put(`${baseURL}/1`, {

        title: "Hello World!",

        body: "This is an updated post."

      })

      .then((response) => {

        setPost(response.data);

      });

  }

  if (!post) return "No post!"

  return (

    <div>

      <h1>{post.title}</h1>

      <p>{post.body}</p>

      <button onClick={updatePost}>Update Post</button>

    </div>

  );

}

## How to Make a DELETE Request

## DELETE

// How to Make a DELETE Request

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

  const [post, setPost] = React.useState(null);

  React.useEffect(() => {

    axios.get(`${baseURL}/1`).then((response) => {

      setPost(response.data);

    });

  }, []);

  function deletePost() {

    axios

      .delete(`${baseURL}/1`)

      .then(() => {

        alert("Post deleted!");

        setPost(null)

      });

  }

  if (!post) return "No post!"

  return (

    <div>

      <h1>{post.title}</h1>

      <p>{post.body}</p>

      <button onClick={deletePost}>Delete Post</button>

    </div>

  );

}

GET request

READ

Graphical user interface, text, application

Description automatically generated

POST request

CREATE

Graphical user interface, text, application

Description automatically generated

PUT Request

UPDATE

Graphical user interface

Description automatically generated

DELETE request

DELETE

Graphical user interface, text, application

Description automatically generated

// --------------------------------------------------------------------------------------------------------------------------

// ---------------------------------------------------------

// 404

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

  const [post, setPost] = React.useState(null);

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

  React.useEffect(() => {

    // invalid url will trigger an 404 error

    axios.get(`${baseURL}/asdf`).then((response) => {

      setPost(response.data);

    }).catch(error => {

      setError(error);

    });

  }, []);

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

  if (!post) return "No post!"

  return (

    <div>

      <h1>{post.title}</h1>

      <p>{post.body}</p>

    </div>

  );

}

Error: Request failed with status code 404

// ---------------------------------------------------------------------------------------

## How to Create an Axios Instance

// -------------------------------------------------------------

// How to create an axios instance

import axios from "axios";

import React from "react";

const client = axios.create({

  baseURL: "https://jsonplaceholder.typicode.com/posts"

});

export default function App() {

  const [post, setPost] = React.useState(null);

  React.useEffect(() => {

    client.get("/1").then((response) => {

      setPost(response.data);

    });

  }, []);

  function deletePost() {

    client

      .delete("/1")

      .then(() => {

        alert("Post deleted!");

        setPost(null)

      });

  }

  if (!post) return "No post!"

  return (

    <div>

      <h1>{post.title}</h1>

      <p>{post.body}</p>

      <button onClick={deletePost}>Delete Post</button>

    </div>

  );

}

Graphical user interface, text, email

Description automatically generated

Graphical user interface, text, email

Description automatically generated

// -------------------------------------------------------------

// How to Use the Async-Await Syntax with Axios

import axios from "axios";

import React from "react";

const client = axios.create({

  baseURL: "https://jsonplaceholder.typicode.com/posts"

});

export default function App() {

  const [post, setPost] = React.useState(null);

  React.useEffect(() => {

    async function getPost() {

      const response = await client.get("/1");

      setPost(response.data);

    }

    getPost();

  }, []);

  async function deletePost() {

    await client.delete("/1");

    alert("Post deleted!");

    setPost(null);

  }

  if (!post) return "No post!"

  return (

    <div>

      <h1>{post.title}</h1>

      <p>{post.body}</p>

      <button onClick={deletePost}>Delete Post</button>

    </div>

  );

}

**sunt aut facere repellat provident occaecati excepturi optio reprehenderit**

quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto

Delete Post

npm install use-axios-client

npm ERR! to accept an incorrect (and potentially broken) dependency resolution.

npm ERR!

npm ERR! See C:\Users\Seppo\AppData\Local\npm-cache\eresolve-report.txt for a full report.

npm ERR! A complete log of this run can be found in:

npm ERR! C:\Users\Seppo\AppData\Local\npm-cache\\_logs\2022-10-15T20\_19\_54\_623Z-debug-0.log

PS C:\Users\Seppo\Downloads\Metropolia\React\weather>

52 error code ERESOLVE

53 error ERESOLVE unable to resolve dependency tree

54 error

55 error While resolving: weather@0.1.0

55 error Found: axios@1.1.3

55 error node\_modules/axios

55 error axios@"^1.1.2" from the root project

55 error

55 error Could not resolve dependency:

55 error peer axios@"^0.19.0" from use-axios-client@2.0.0

55 error node\_modules/use-axios-client

55 error use-axios-client@"\*" from the root project

55 error

55 error Fix the upstream dependency conflict, or retry

55 error this command with --force, or --legacy-peer-deps

55 error to accept an incorrect (and potentially broken) dependency resolution.

55 error

55 error See C:\Users\Seppo\AppData\Local\npm-cache\eresolve-report.txt for a full report.

Module not found: Error: Can't resolve 'use-axios-client' in 'C:\Users\Seppo\Downloads\Metropolia\React\weather\src'

ERROR in ./src/App.js 307:0-44

Module not found: Error: Can't resolve 'use-axios-client' in 'C:\Users\Seppo\Downloads\Metropolia\React\weather\src'

webpack compiled with 1 error

// tryed this

rm -rf node\_modules/

npm cache clean

npm install

This will install all the required dependencies correctly.

AXIOS HAS MANY ProBLemS

I just installed it and it does not work

Finally

npm install use-axios-client

worked when I started from a clean project??? I think that npm install use-axios-client

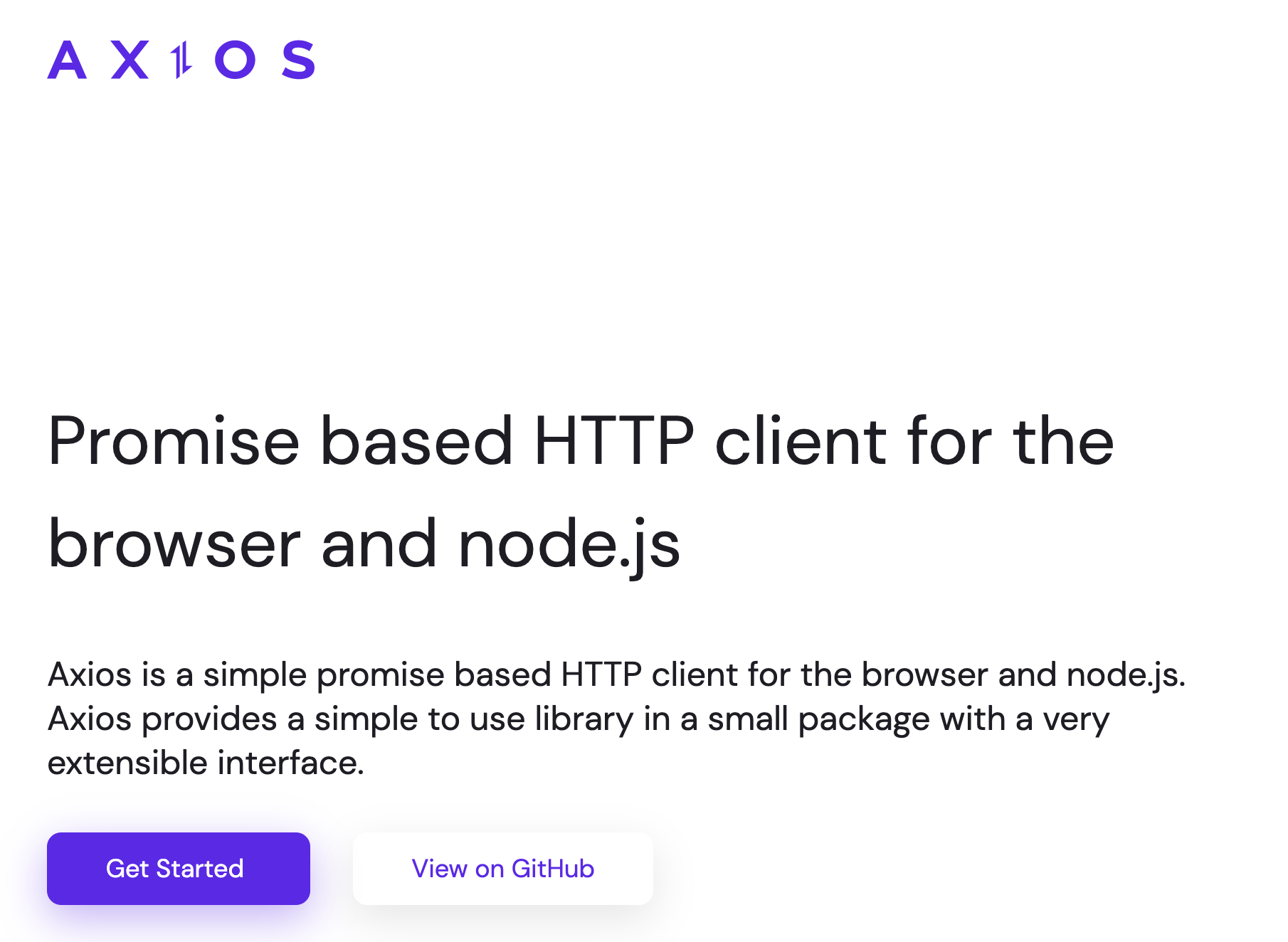
is not working with npm install axios

Graphical user interface, application

Description automatically generated

## What is Axios?

Axios is an HTTP client library that allows you to make requests to a given endpoint:



This could be an external API or your own backend Node.js server, for example.

By making a request, you expect your API to perform an operation according to the request you made.

For example, if you make a GET request, you expect to get back data to display in your application.

## Why Use Axios in React

There are a number of different libraries you can use to make these requests, so why choose Axios?

Here are **five reasons** why you should use Axios as your client to make HTTP requests:

1. It has good defaults to work with JSON data. Unlike alternatives such as the Fetch API, you often don't need to set your headers. Or perform tedious tasks like converting your request body to a JSON string.
2. Axios has function names that match any HTTP methods. To perform a GET request, you use the .get() method.
3. Axios does more with less code. Unlike the Fetch API, you only need one .then() callback to access your requested JSON data.
4. Axios has better error handling. Axios throws 400 and 500 range errors for you. Unlike the Fetch API, where you have to check the status code and throw the error yourself.
5. Axios can be used on the server as well as the client. If you are writing a Node.js application, be aware that Axios can also be used in an environment separate from the browser.

## How to Set Up Axios with React

Using Axios with React is a very simple process. You need three things:

1. An existing React project
2. To install Axios with npm/yarn
3. An API endpoint for making requests

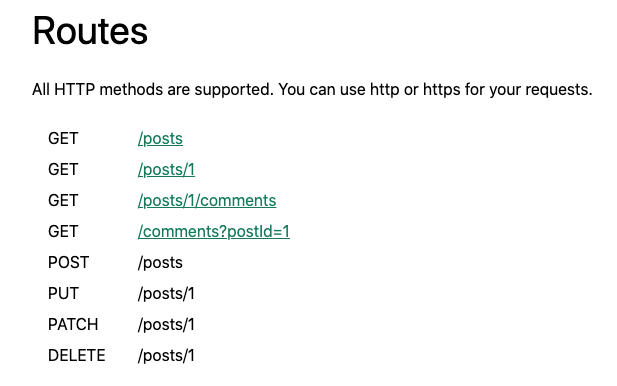
The quickest way to create a new React application is by going to [react.new](https://react.new/).

If you have an existing React project, you just need to install Axios with npm (or any other package manager):

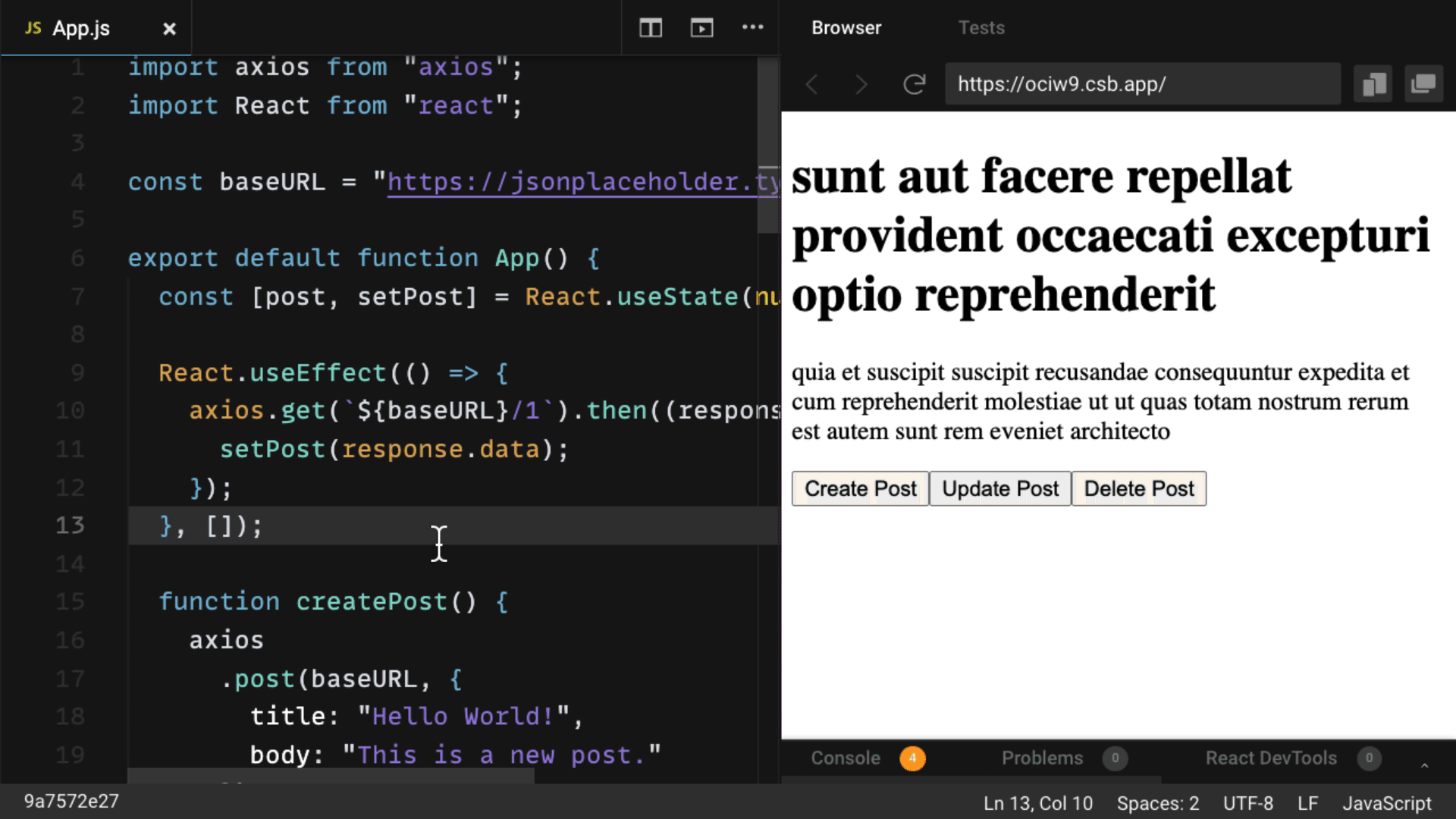
npm install axios

In this guide, you'll use the JSON Placeholder API to get and change post data.

Here is a list of all the different routes you can make requests to, along with the appropriate HTTP method for each:



Here is a quick example of all of the operations you'll be performing with Axios and your API endpoint — retrieving, creating, updating, and deleting posts:



## How to Make a GET Request

To fetch data or retrieve it, make a GET request.

First, you're going to make a request for individual posts. If you look at the endpoint, you are getting the first post from the /posts endpoint:

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts/1";

export default function App() {

const [post, setPost] = React.useState(null);

React.useEffect(() => {

axios.get(baseURL).then((response) => {

setPost(response.data);

});

}, []);

if (!post) return null;

return (

<div>

<h1>{post.title}</h1>

<p>{post.body}</p>

</div>

);

}

To perform this request when the component mounts, you use the useEffect hook. This involves importing Axios, using the .get() method to make a GET request to your endpoint, and using a .then() callback to get back all of the response data.

The response is returned as an object. The data (which is in this case a post with id, title, and body properties) is put in a piece of state called post which is displayed in the component.

Note that you can always find the requested data from the .data property in the response.

## How to Make a POST Request

To create new data, make a POST request.

According to the API, this needs to be performed on the /posts endpoint. If you look at the code below, you'll see that there's a button to create a post:

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

const [post, setPost] = React.useState(null);

React.useEffect(() => {

axios.get(`${baseURL}/1`).then((response) => {

setPost(response.data);

});

}, []);

function createPost() {

axios

.post(baseURL, {

title: "Hello World!",

body: "This is a new post."

})

.then((response) => {

setPost(response.data);

});

}

if (!post) return "No post!"

return (

<div>

<h1>{post.title}</h1>

<p>{post.body}</p>

<button onClick={createPost}>Create Post</button>

</div>

);

}

When you click on the button, it calls the createPost function.

To make that POST request with Axios, you use the .post() method. As the second argument, you include an object property that specifies what you want the new post to be.

Once again, use a .then() callback to get back the response data and replace the first post you got with the new post you requested.

This is very similar to the .get() method, but the new resource you want to create is provided as the second argument after the API endpoint.

## How to Make a PUT Request

To update a given resource, make a PUT request.

In this case, you'll update the first post.

To do so, you'll once again create a button. But this time, the button will call a function to update a post:

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

const [post, setPost] = React.useState(null);

React.useEffect(() => {

axios.get(`${baseURL}/1`).then((response) => {

setPost(response.data);

});

}, []);

function updatePost() {

axios

.put(`${baseURL}/1`, {

title: "Hello World!",

body: "This is an updated post."

})

.then((response) => {

setPost(response.data);

});

}

if (!post) return "No post!"

return (

<div>

<h1>{post.title}</h1>

<p>{post.body}</p>

<button onClick={updatePost}>Update Post</button>

</div>

);

}

In the code above, you use the PUT method from Axios. And like with the POST method, you include the properties that you want to be in the updated resource.

Again, using the .then() callback, you update the JSX with the data that is returned.

## How to Make a DELETE Request

Finally, to delete a resource, use the DELETE method.

As an example, we'll delete the first post.

Note that you do not need a second argument whatsoever to perform this request:

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

const [post, setPost] = React.useState(null);

React.useEffect(() => {

axios.get(`${baseURL}/1`).then((response) => {

setPost(response.data);

});

}, []);

function deletePost() {

axios

.delete(`${baseURL}/1`)

.then(() => {

alert("Post deleted!");

setPost(null)

});

}

if (!post) return "No post!"

return (

<div>

<h1>{post.title}</h1>

<p>{post.body}</p>

<button onClick={deletePost}>Delete Post</button>

</div>

);

}

In most cases, you do not need the data that's returned from the .delete() method.

But in the code above, the .then() callback is still used to ensure that your request is successfully resolved.

In the code above, after a post is deleted, the user is alerted that it was deleted successfully. Then, the post data is cleared out of the state by setting it to its initial value of null.

Also, once a post is deleted, the text "No post" is shown immediately after the alert message.

## How to Handle Errors with Axios

What about handling errors with Axios?

What if there's an error while making a request? For example, you might pass along the wrong data, make a request to the wrong endpoint, or have a network error.

To simulate an error, you'll send a request to an API endpoint that doesn't exist: /posts/asdf.

This request will return a 404 status code:

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

const [post, setPost] = React.useState(null);

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

React.useEffect(() => {

// invalid url will trigger an 404 error

axios.get(`${baseURL}/asdf`).then((response) => {

setPost(response.data);

}).catch(error => {

setError(error);

});

}, []);

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

if (!post) return "No post!"

return (

<div>

<h1>{post.title}</h1>

<p>{post.body}</p>

</div>

);

}

In this case, instead of executing the .then() callback, Axios will throw an error and run the .catch() callback function.

In this function, we are taking the error data and putting it in state to alert our user about the error. So if we have an error, we will display that error message.

In this function, the error data is put in state and used to alert users about the error. So if there's an error, an error message is displayed.

When you run this code code, you'll see the text, "Error: Request failed with status code 404".

## How to Create an Axios Instance

If you look at the previous examples, you'll see that there's a baseURL that you use as part of the endpoint for Axios to perform these requests.

However, it gets a bit tedious to keep writing that baseURL for every single request. Couldn't you just have Axios remember what baseURL you're using, since it always involves a similar endpoint?

In fact, you can. If you create an instance with the .create() method, Axios will remember that baseURL, plus other values you might want to specify for every request, including headers:

import axios from "axios";

import React from "react";

const client = axios.create({

baseURL: "https://jsonplaceholder.typicode.com/posts"

});

export default function App() {

const [post, setPost] = React.useState(null);

React.useEffect(() => {

client.get("/1").then((response) => {

setPost(response.data);

});

}, []);

function deletePost() {

client

.delete("/1")

.then(() => {

alert("Post deleted!");

setPost(null)

});

}

if (!post) return "No post!"

return (

<div>

<h1>{post.title}</h1>

<p>{post.body}</p>

<button onClick={deletePost}>Delete Post</button>

</div>

);

}

The one property in the config object above is baseURL, to which you pass the endpoint.

The .create() function returns a newly created instance, which in this case is called client.

Then in the future, you can use all the same methods as you did before, but you don't have to include the baseURL as the first argument anymore. You just have to reference the specific route you want, for example, /, /1, and so on.

## How to Use the Async-Await Syntax with Axios

A big benefit to using promises in JavaScript (including React applications) is the async-await syntax.

Async-await allows you to write much cleaner code without then and catch callback functions. Plus, code with async-await looks a lot like synchronous code, and is easier to understand.

But how do you use the async-await syntax with Axios?

In the example below, posts are fetched and there's still a button to delete that post:

import axios from "axios";

import React from "react";

const client = axios.create({

baseURL: "https://jsonplaceholder.typicode.com/posts"

});

export default function App() {

const [post, setPost] = React.useState(null);

React.useEffect(() => {

async function getPost() {

const response = await client.get("/1");

setPost(response.data);

}

getPost();

}, []);

async function deletePost() {

await client.delete("/1");

alert("Post deleted!");

setPost(null);

}

if (!post) return "No post!"

return (

<div>

<h1>{post.title}</h1>

<p>{post.body}</p>

<button onClick={deletePost}>Delete Post</button>

</div>

);

}

However in useEffect, there's an async function called getPost.

Making it async allows you to use the await keword to resolve the GET request and set that data in state on the next line without the .then() callback.

Note that the getPost function is called immediately after being created.

Additionally, the deletePost function is now async, which is a requirement to use the await keyword which resolves the promise it returns (every Axios method returns a promise to resolve).

After using the await keyword with the DELETE request, the user is alerted that the post was deleted, and the post is set to null.

As you can see, async-await cleans up the code a great deal, and you can use it with Axios very easily.

## How to Create a Custom useAxios Hook

Async-await is a great way to simplify your code, but you can take this a step further.

Instead of using useEffect to fetch data when the component mounts, you could create your own custom hook with Axios to perform the same operation as a reusable function.

While you can make this custom hook yourself, there's a very good library that gives you a custom useAxios hook called use-axios-client.

First, install the package:

npm install use-axios-client

To use the hook itself, import useAxios from use-axios-client at the top of the component.

Because you no longer need useEffect, you can remove the React import:

import { useAxios } from "use-axios-client";

export default function App() {

const { data, error, loading } = useAxios({

url: "https://jsonplaceholder.typicode.com/posts/1"

});

if (loading || !data) return "Loading...";

if (error) return "Error!";

return (

<div>

<h1>{data.title}</h1>

<p>{data.body}</p>

</div>

)

}

Now you can call useAxios at the top of the app component, pass in the URL you want to make a request to, and the hook returns an object with all the values you need to handle the different states: loading, error and the resolved data.

In the process of performing this request, the value loading will be true. If there's an error, you'll want to display that error state. Otherwise, if you have the returned data, you can display it in the UI.

The benefit of custom hooks like this is that it really cuts down on code and simplifies it overall.

If you're looking for even simpler data fetching with Axios, try out a custom useAxios hook like this one.