In this article, you will see examples of how to use Axios to access the popular JSON Placeholder API within a React application.

Prerequisites

To follow along with this article, you'll need the following:

- Node.js version 10.16.0 installed on your computer. To install this on macOS or Ubuntu 18.04, follow the steps in How to Install Node.js and Create a Local Development Environment on macOS or the **Installing Using a PPA** section of How To Install Node.js on Ubuntu 18.04.
- A new React project set up with Create React App by following the How to Set up a React Project with Create React App tutorial.
- It will also help to have a basic understanding of JavaScript, which you can find in the How To Code in JavaScript series, along with a basic knowledge of HTML and CSS.

Step 1 — Adding Axios to the Project

In this section, you will add Axios to the `digital-ocean-tutorial` React project you created following the How to Set up a React Project with Create React App tutorial.

To add Axios to the project, open your terminal and change directories into your project:

```
$ cd digital-ocean-tutorial
```

Then run this command to install Axios:

```
$ npm install axios
```

Next, you will need to import Axios into the file you want to use it in.

Step 2 — Making a `GET` Request

In this example, you create a new component and import Axios into it to send a `GET` request.

Inside the `src` folder of your React project, create a new component named `PersonList.js`:

```
$ nano src/PersonList.js
```

Add the following code to the component:

digital-ocean-tutorial/src/PersonList.js

```
import React from 'react';

import axios from 'axios';

export default class PersonList extends React.Component {
  state = {
    persons: []
  }

  componentDidMount() {
    axios.get(`https://jsonplaceholder.typicode.com/users`)
      .then(res => {
        const persons = res.data;
        this.setState({ persons });
      })
  }

  render() {
    return (
      <ul>
        { this.state.persons.map(person => <li>{person.name}</li>)}
      </ul>
    )
  }
}
```

First, you import React and Axios so that both can be used in the component. Then you hook into the `componentDidMount` lifecycle hook and perform a `GET` request.

You use `axios.get(url)` with a URL from an API endpoint to get a promise which returns a response object. Inside the response object, there is data that is then assigned the value of `person`.

You can also get other information about the request, such as the status code under `res.status` or more information inside of `res.request`.

Step 3 — Making a `POST` Request

In this step, you will use Axios with another HTTP request method called `POST`.

Remove the previous code in `PersonList` and add the following to create a form that allows for user input and subsequently `POST`s the content to an API:

digital-ocean-tutorial/src/PersonList.js

```
import React from 'react';
import axios from 'axios';

export default class PersonList extends React.Component {
  state = {
    name: '',
  }

  handleChange = event => {
    this.setState({ name: event.target.value });
  }

  handleSubmit = event => {
    event.preventDefault();

    const user = {
      name: this.state.name
    };

    axios.post(`https://jsonplaceholder.typicode.com/users`, { user })
      .then(res => {
        console.log(res);
        console.log(res.data);
      })
  }

  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <label>
            Person Name:
            <input type="text" name="name" onChange={this.handleChange} />
          </label>
          <button type="submit">Add</button>
        </form>
      </div>
    )
  }
}
```

Inside the `handleSubmit` function, you prevent the default action of the form. Then update the `state` to the `user` input.

Using `POST` gives you the same response object with information that you can use inside of a `then` call.

To complete the `POST` request, you first capture the `user` input. Then you add the input along with the `POST` request, which will give you a response. You can then `console.log` the response, which should show the `user` input in the form.

Step 4 — Making a `DELETE` Request

In this example, you will see how to delete items from an API using `axios.delete` and passing a URL as a parameter.

Change the code for the form from the `POST` example to delete a user instead of adding a new one:

digital-ocean-tutorial/src/PersonList.js

```
import React from 'react';
import axios from 'axios';

export default class PersonList extends React.Component {
  state = {
    id: '',
  }

  handleChange = event => {
    this.setState({ id: event.target.value });
  }

  handleSubmit = event => {
    event.preventDefault();

    axios.delete(`https://jsonplaceholder.typicode.com/users/${this.state.id}`)
      .then(res => {
        console.log(res);
        console.log(res.data);
      })
  }

  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <label>
            Person ID:
            <input type="text" name="id" onChange={this.handleChange} />
          </label>
          <button type="submit">Delete</button>
        </form>
      </div>
    )
  }
}
```

Again, the `res` object provides you with information about the request. You can then `console.log` that information again after the form is submitted.

Step 5 — Using a Base Instance in Axios

In this example, you will see how you can set up a *base instance* in which you can define a URL and any other configuration elements.

Create a separate file named `api.js`:

```
$ nano src/api.js
```

Export a new `axios` instance with these defaults:

digital-ocean-tutorial/src/api.js

```
import axios from 'axios';

export default axios.create({
  baseURL: `http://jsonplaceholder.typicode.com/`
});
```

Once the default instance is set up, it can then be used inside of the `PersonList` component. You import the new instance like this:

digital-ocean-tutorial/src/PersonList.js

```
import React from 'react';
import axios from 'axios';

import API from '../api';

export default class PersonList extends React.Component {
  handleSubmit = event => {
    event.preventDefault();

    API.delete(`users/${this.state.id}`)
      .then(res => {
        console.log(res);
        console.log(res.data);
      })
  }
}
```

Because `http://jsonplaceholder.typicode.com/` is now the base URL, you no longer need to type out the whole URL each time you want to hit a different endpoint on the API.

Step 6 — Using `async` and `await`

In this example, you will see how you can use `async` and `await` to work with promises.

The `await` keyword resolves the `promise` and returns the `value`. The `value` can then be assigned to a variable.

```
handleSubmit = async event => {
  event.preventDefault();

  //
  const response = await API.delete(`users/${this.state.id}`);

  console.log(response);
  console.log(response.data);
};
```

In this code sample, the `.then()` is replaced. The `promise` is resolved, and the value is stored inside the `response` variable.

Conclusion

In this tutorial, you explored several examples on how to use Axios inside a React application to create HTTP requests and handle responses.

If you'd like to learn more about React, check out the How To Code in React.js series, or check out the React topic page for more exercises and programming projects.

About the authors

PaulHalliday

I create educational content over at YouTube and https://developer.school.

christinagorton

Editor