**Axios**

Big part of our upcoming project is going to be fetching data from and sending data to the server and even though in theory, we could accomplish all of our upcoming tasks with just good old fetch API, which is built into the browser in order to avoid a bunch of headaches and make our workflow smoother, we'll use HTTP library by the name of Axios instead, which makes data fetching a breeze.

Axios is a [promise-based](https://javascript.info/promise-basics) HTTP Client for [node.js](https://nodejs.org/) and the browser. It is [isomorphic](https://www.lullabot.com/articles/what-is-an-isomorphic-application) (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js http module, while on the client (browser) it uses XMLHttpRequests.

**Features**

* Make XMLHttpRequests from the browser
* Make http requests from node.js
* Supports the Promise API
* Intercept request and response
* Transform request and response data
* Cancel requests
* Timeouts
* Query parameters serialization with support for nested entries
* Automatic request body serialization to:
* JSON (application/json)
* Multipart / FormData (multipart/form-data)
* URL encoded form (application/x-www-form-urlencoded)
* Posting HTML forms as JSON
* Automatic JSON data handling in response
* Progress capturing for browsers and node.js with extra info (speed rate, remaining time)
* Setting bandwidth limits for node.js
* Compatible with spec-compliant FormData and Blob (including node.js)
* Client side support for protecting against XSRF

**Installing**

To install Axios, simply go to command line, to the folder you are working on and run npm install axios.

**Methods**

Axios comes with many methods but the most used ones are:

* axios.get(url)
* axios.post(url)
* axios.patch/put(url)
* axios.delete(url)

If we don’t specify the method, by default it runs axios.get().

* returns a promise
* response data located in data property
* error in error.response

**Setup**

import axios from 'axios';

const fetchData = async () => {

  try {

    // axios.get(), axios.post(),axios.put(), axios.delete()

    const response = await axios(url);

    console.log(response);

  } catch (error) {

    console.log(error.response);

  }

};

**axios.get() examples**

import { useEffect } from "react";

import axios from "axios";

// limit, if 429 wait for 15 min and try again

const url = "https://course-api.com/react-store-products";

const FirstRequest = () => {

  const fetchData = async () => {

    try {

      const response = await axios(url);

      const data = response.data;

    } catch (error) {

      console.log(error.response);

    }

  };

  useEffect(() => {

    fetchData();

  }, []);

  return <h2 className="text-center">first request</h2>;

};

export default FirstRequest;

**axios.get() examples**

What if we need to provide more options? For example, a request headers come to mind and for that we'll need to pass in additional argument and we'll set it up as an object.

If you are using straight up get requests, then is is going to be a second argument

axios.get(url,{})

However, if you're using one of the methods where actually we're passing in data to the server, for example, post, then this is going to be a third argument.

axios.post(url,{data},{})

In the example below, we picked an API that is looking for the header that is going to be Dad Joke API. Essectially we’ll set up a straight up get request.

There we want to set up the headers property and after that the API is looking for accept header and we want to pass in this application json. Otherwise, the API is going to send back the text. And then we'll add a little bit of additional logic.

So in order to get the Dad Joke, we also need to get to provide the accept header.

First let’s show what happens if we don’t do that.

import { useState } from "react";

import axios from "axios";

const url = "https://icanhazdadjoke.com/";

// Accept : 'application/json'

const Headers = () => {

  const [joke, setJoke] = useState("random dad joke");

  const fetchDadJoke = async () => {

    try {

      const resp = await axios(url);

      console.log(resp);

    } catch (error) {

      console.log(error.resp);

    }

  };

  return (

    <section className="section text-center">

      <button className="btn" onClick={fetchDadJoke}>

        random joke

      </button>

      <p className="dad-joke">{joke}</p>

    </section>

  );

};

export default Headers;



You can see that we are not getting our data in the json format. So let’s add that additional argument and set it up as an object:

  const fetchDadJoke = async () => {

    try {

      const resp = await axios(url, {

        headers: {

          Accept: "application/json",

        },

      });

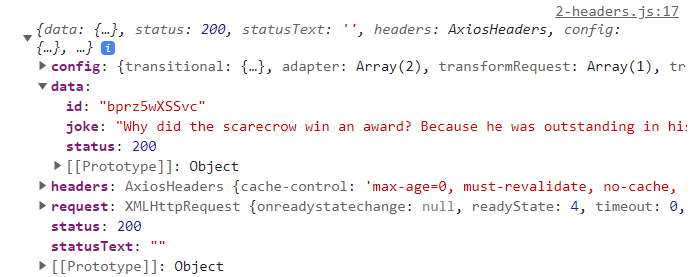
      console.log(resp);

    } catch (error) {

      console.log(error.resp);

    }

  };



You can also destructure it right away:

  const fetchDadJoke = async () => {

    try {

      const { data } = await axios(url, {

        headers: {

          Accept: "application/json",

        },

      });

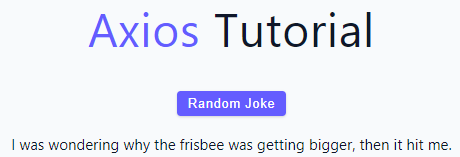
      setJoke(data.joke);

    } catch (error) {

      console.log(error.resp);

    }

  };



So that's how we can set up additional info to our request, more specifically, headers. We add an object as additional argument and in there we look for headers property and whichever error you want to add.

**Post Requests**

Let’s have a look on how we have send data to the server. In here you want to pass the url as well as the data you want to send to the server.

axios.post(url, { data })

If you need to add more info about the request (e.g. authorization header), like a get request you pass as a third argument.

axios.post(url, { data },{})

The common setup for post request is the following:

try {

  const resp = await axios.post(url, { data });

} catch (error) {

  console.log(error.response.data);

}

**Global Defaults**

now let's take a look at how we can set up a global Axios instance. What it means is that we set some kind of default functionality, and every time we make a request with Axios instance, that default functionality is going to be added.

For example, If you know you are going to send various requests to the same server you can add baseURL:

axios.defaults.baseURL = "https://api.example.com"

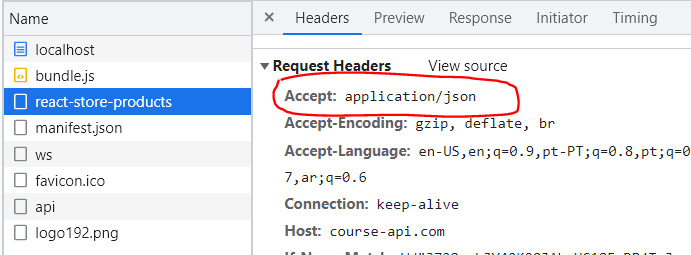
we can add the accept header and set it equal to application/json. So every request is going to have that Accept header.

axios.defaults.headers['Accept'] = 'application/json';

For example consider the following code:

|  |
| --- |
| App.js |
| import Title from "./components/Title";  import Setup from "./examples/4-global-instance";  import "./axios/global.js";  function App() {    return (      <main>        <Title />        <Setup />      </main>    );  }  export default App; |
| 4-global-instance.js |
| import { useEffect } from "react";  import axios from "axios";  const productsUrl = "https://course-api.com/react-store-products";  const randomUserUrl = "https://randomuser.me/api";  const GlobalInstance = () => {    const fetchData = async () => {      try {        const resp1 = await axios(productsUrl);        const resp2 = await axios(randomUserUrl);      } catch (error) {}    };    useEffect(() => {      fetchData();    }, []);    return <h2 className="text-center">global instance</h2>;  };  export default GlobalInstance; |
| global.js |
| import axios from “axios”;  axios.defaults.headers[“Accept”] = “application/json”; |

If we navigate to the Network tab, then we can see the Accept in the request headers in both requests because we set up that global instance.



This case you can make an argument is not a big deal. But we need to keep in mind that if we have something sensitive like, for example, JSON web token, we don't want to start this sending left and right. And this is something that we'll fix with a custom instance.

We can set up a custom instance where we create a new file, instead of setting it up globally. We import Axios, but instead of defaults, we go with create again. See the example below. See that in custom we create a custom request which sets the base URL, as well as the header.

So, resp1 will only have to provide the second part of the complete url, because it uses the custom header which provides the first part. And also resp1 will have the fetch header. Resp2 just uses the normal axios instance, so it’s just going to be a standard request.

|  |
| --- |
| 5-custom-instance |
| import { useEffect } from "react";  import authFetch from "../axios/custom";  import axios from "axios";  const randomUserUrl = "https://randomuser.me/api";  const CustomInstance = () => {    const fetchData = async () => {      try {        const resp1 = await authFetch("/react-store-products");        const resp2 = await axios(randomUserUrl);      } catch (error) {}    };    useEffect(() => {      fetchData();    }, []);    return <h2 className="text-center">custom instance</h2>;  };  export default CustomInstance;  };  export default GlobalInstance; |
| custom.js |
| import axios from "axios";  const autoFetch = axios.create({    baseURL: "https://course-api.com",    headers: {      Accept: "application/json",    },  });  export default authFetch; |

And as you can see, this is totally up to you. More specifically, depends on your application. If you have a setup where you can just add everything as a global instance.

So, if you don't have anything that possibly might be a security issue, then you can just add those things to a global instance and you're good to go.

However, if you have a situation where you need more detailed setup, where effectively you want to add this only to specific requests, then of course, custom instance makes a little bit more sense.

**Interceptors**

Interceptors are functions that Axios calls for every request and we can use them to transform the request before it leaves the application, as well as add some custom logic when we handle the response.

Interceptors make more sense in a complex application (e.g. authentication). In our examples we’ll mostly just lock stuff.

In our examples we will add them to the custom instance, but we could add to the global instances instead.

The syntax for request is the following. In this case we are just adding the request header, but of course you can add other things like authorization header, base url, etc.

authFetch.interceptors.request.use(

  (request) => {

    // request.headers.common['Accept'] = `application/json`;

    request.headers['Accept'] = `application/json`;

    console.log('request sent');

    // must return request

    return request;

  },

  (error) => {

    return Promise.reject(error);

  }

);

And for the response:

authFetch.interceptors.response.use(

  (response) => {

    console.log('got response');

    return response;

  },

  (error) => {

    console.log(error.response);

    if (error.response.status === 404) {

      // do something

      console.log('NOT FOUND');

    }

    return Promise.reject(error);

  }

);

For authentication purposes, because this gives us an option to set up code for the entire application. So for example, if we’re getting some type of response, we can log out the user in all the requests.

See the following program.

|  |
| --- |
| 6-interceptors.js |
| import { useEffect } from "react";  import authFetch from "../axios/interceptors";  const url = "https://course-api.com/react-store-products";  const Interceptors = () => {    const fetchData = async () => {      try {        const resp = await authFetch("/react-store-products");      } catch (error) {}    };    useEffect(() => {      fetchData();    }, []);    return <h2 className="text-center">interceptors</h2>;  };  export default Interceptors; |
| interceptors.js |
| import axios from "axios";  // Custom instance  const authFetch = axios.create({    baseURL: "https://course-api.com",  });  // Interceptor  authFetch.interceptors.request.use(    (request) => {      request.headers["Accept"] = "application/json";      console.log("request sent");      return request;    },    (error) => {      return Promise.reject(error);    }  );  authFetch.interceptors.response.use(    (response) => {      console.log("got response");      return response;    },    (error) => {      console.log(error.response);      if (error.response.status === 404) {        console.log("NOT FOUND");      }      return Promise.reject(error);    }  );  export default authFetch; |

So we send a request using the custom instance. If that request is successful, we log “request send” and request the accept header. If not, an error message is returned.

As for the response, if we get it (the URL exists), we simply log “got response. But if we change the URL so it’s wrong

      const resp = await authFetch("/react-store-productss");

A 404 error will happen and we will log NOT FOUND.

