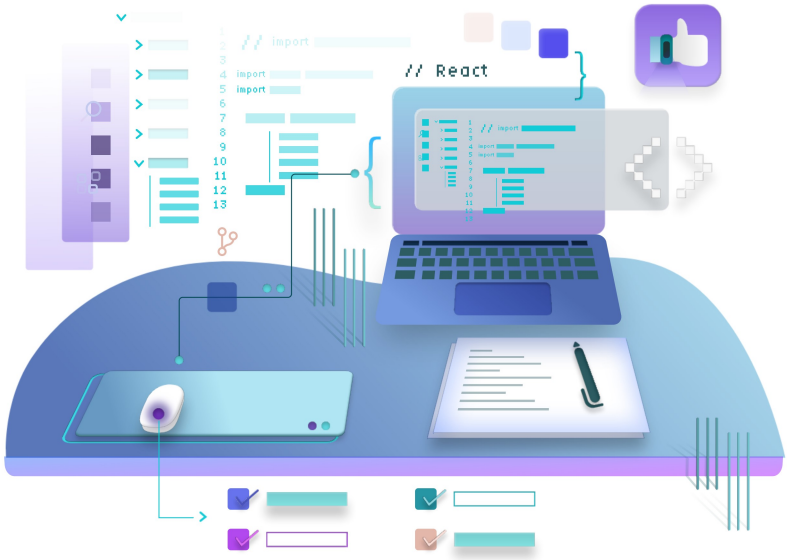


Fetch Data from an API

ADD TO FAVORITES

Learn the basics of asynchronous functions and promises by fetching data from an API using fetch, useEffect and useState

CodeSandbox link

You can find the full code for this tutorial at https://codesandbox.io/s/fetch-data-from-an-api-hko1m.

Import useEffect

Import useEffect at the top of your file.

```
import React, { useEffect } from "react"
```

Then, define your useEffect hook inside of your component.

```
import React, { useEffect } from "react";

const App = () => {
  useEffect(() => {

  }, []);

  return <div></div>;
};

export default App;
```

Learn more about the useEffect hook and the component lifecycle in the **useEffect section** of this handbook.

Define your URL

When the user lands on our page, we want to call the API. In other words, we want to call the API during the mounting part of the component's lifecycle.

The API url to get a random advice we want to call is **https://api.adviceslip.com/advice**. Let's save it in a variable inside of the useEffect hook.

```
t(() => {
st url = "https://api.adviceslip.com/advice"
}, []);
```

Create the asynchronous function

Then, let's create an **asynchronous function** to fetch our data. An asynchronous function is a function that needs to wait after the **promise** is resolved before continuing. In our case, the function will need to wait after the data is fetched (our promise) before continuing.

```
const fetchData = async () => {
  try {
    const response = await fetch(url);
    const json = await response.json();
    console.log(json);
    } catch (error) {
    console.log("error", error);
    }
};
```

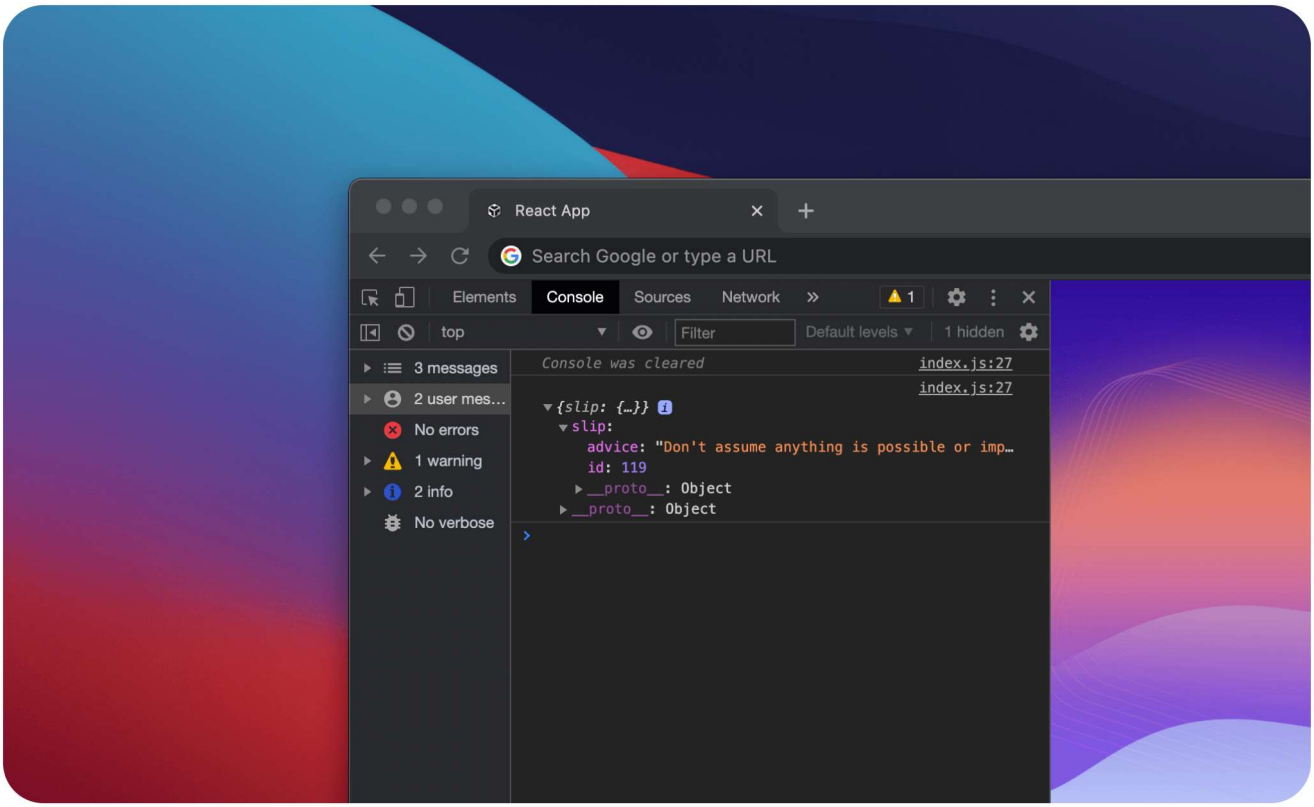
Put the fetchData function above in the useEffect hook and call it, like so:

```
useEffect(() => {
  const url = "https://api.adviceslip.com/advice";

  const fetchData = async () => {
    try {
      const response = await fetch(url);
      const json = await response.json();
      console.log(json);
    } catch (error) {
      console.log("error", error);
    }
  };

  Data();
```

The function we just created is wrapped in a **try...catch statement** so that the function catches the errors and prints them in the console. This helps debug and prevents the app to crash unexpectedly.

Inside of the try, we are using the built-in **fetch API from JavaScript** to get the advice from the Advice Slip JSON API. We put the **await** keyword just in front of it to tell the function to wait for the fetch task to be done before running the next line of code.

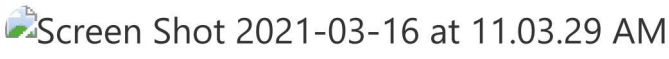
Once we get a response, we are parsing it using the **.json() function**, meaning that we are transforming the response into JSON data that we can easily read. Then, we are printing the JSON data in the console.



Get the advice string

As we can see in the console, we got an advice slip from the API. We only need th ice, and not the id, so let's just print the advice itself:

```
console.log(json.slip.advice);
```



Save the advice in a local state

Now that we got the actual advice string, we can save it into a local state using the useState hook. Learn more about the useState hook in **the useState hook section** of this handbook.

Let's import useState and define it.

```
import React, { useEffect, useState } from "react"

const [advice, setAdvice] = useState("")
```

Then, instead of printing the advice in the console, we'll save it in the advice state.

```
setAdvice(json.slip.advice)
```

Display the advice

To display the advice, simply call the advice state in your return body:

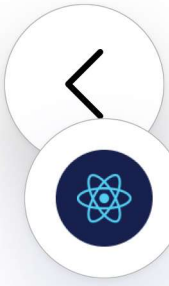
```
return (
  <div>
    <p>{advice}</p>
  </div>
);
```

Final code with styling

The final code, along with styling the text using **styled-components**, will look like so:

```
import React, { useEffect, useState } from "react";
import styled from "styled-components";

const App = () => {
  const [advice, setAdvice] = useState("");

  useEffect(() => {
    const url = "https://api.adviceslip.com/advice";

    const fetchData = async () => {
      try {
        const response = await fetch(url);
        const json = await response.json();
        console.log(json.slip.advice);
        setAdvice(json.slip.advice);
      } catch (error) {
        console.log("error", error);
      }
    };

    fetchData();
  }, []);

  return (
    <Wrapper>
      <Paragraph>{advice}</Paragraph>
    </Wrapper>
  );
};

export default App;

const Wrapper = styled.div`
```

```
  padding-top: 150px;
  margin: 0 auto;
`;

const Paragraph = styled.h2`
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Oxygen, U
  font-style: normal;
  font-weight: bold;
  font-size: 20px;
  line-height: 48px;
  text-align: center;
`;
```

BACK TO

Load Local Data

READ NEXT

Toggle a state

TEMPLATES AND SOURCE CODE

Download source files

Download the videos and assets to refer and learn offline without interuption.

- **Design** template
- **Source code** for all sections
- **Video files,** ePub and subtitles

Videos

ePub

Assets

1 **Intro to React Hooks** 3:39

An overview of React Hooks and the frameworks you can use to...

2 **Create your first React app** 4:23

Create your first React project from the Terminal and save it on...

React Component 2:54

Create your first JSX component using React

4 **Styling in React** 5:06

How to style your React components using inline styling,...

Styles and Props 2:22

Understanding 3:21

5 **Styles and Props** 2:22

Render different styles depending on different properties passed to...

6 **Understanding Hooks** 3:21

Learn about the basics of React Hooks, which introduced at Reac...

7 **useState Hook** 2:54

Use the useState hook to manage local state in your React...

8 **useEffect Hook** 3:41

Manage with your component's lifecycle with the useEffect hook

9 **useRef Hook** 3:00

Learn about the useRef hook, which replaces the JavaScript...

10 **Props** 3:11

Learn about props in React to pass data from parent to child...

11 **Conditional Rendering** 4:21

Render different UIs depending on different conditions and states

12 **Load Local Data** 4:04

Load local JSON data into your React application

13 **Fetch Data from an API** 5:40

Learn the basics of asynchronous functions and promises by...

14 **Toggle a state** 4:05

Learn how to toggle a state from true to false and back again

15 **useInput Hook** 6:04

Create a hook to get the value and the onChange event of inpu...

16 **Intro to Firebase** 6:59

Learn about Firebase and set it up in your React project

17 **Firebase Auth** 11:59

Set up authentication in your React application using Firebase...

18 **Firestore** 10:51

Enable Firestore as your database in your React application

19 **Firebase Storage** 6:40

Enable Firebase Storage in your application to store photos or...

20 **Gatsby and React** 6:44

Create a static content-oriented website using React on Gatsby

21 **NextJS and React** 5:24

Create your first NextJS React application

22 **React TypeScript Part 1** 8:19

Learn how to create a React TypeScript application using the...

23 **React TypeScript Part 2** 7:35

Learn the basics of TypeScript and how to use TypeScript in a React...

24 **useScrollPosition Hook** 4:26

Create a custom hook to listen to the current window position of...

useOnScreen hook 8:08

Create a custom hook to listen to

26 **useContext Hook** 8:32

Manage global states throughout the entire application

when an element is visible on...

27 **Fragments** 2:43
Group multiple children together with React Fragments

28 **Lazy Loading** 4:05
Lazy Load heavy components to improve performance

29 **React Suspense** 3:13
Wait for data with React Suspense and React.lazy

30 **Environment Variables** 4:43
Make environment variables secret with a .env file

31 **Reach Router** 5:31
Create a multiple-pages React application with Reach Router

32 **URL Params** 4:04
Create unique URL with URL Params

33 **SEO and Metadata** 6:47
Optimize a React application for search engines with React Helmet

34 **Favicon** 3:03
Add an icon to a React website

35 **Dynamic Favicon** 2:14
Change the favicon's fill color depending on the user's system...

36 **PropTypes** 3:54
Implement props type-checking with PropTypes

37 **Custom PropTypes** 3:58
Create a custom PropType using a validator function

38 **useMemo Hook** 4:05
Prevent unnecessary re-renders when the component stays the...

39 **forwardRef Hook** 3:28
Forward a ref to a child component

40 **Serverless Email Sending** 10:02
Use EmailJS to send an email without backend

41 **Handling Events** 5:44
How to handle events in React

42 **Geocoding with Mapbox** 9:24
Create an address autocomplete with Mapbox's Geocoding API

43 **Spread attributes** 3:35
Learn how to make use of the spread operator

44 **useMousePositio n Hook** 4:55
Detect the user's mouse position on a bound element

useReducer with useContext Part 1 7:33
Create a reducer to be used in a context

46 **useReducer with useContext Par...** 6:48
Incorporate useReducer with useContext

47 **useReducer with useContext Par...** 5:43
Connect the context and reducer with the frontend

48 **Netlify** 5:08
Deploy to production using Netlify

49 **Gatsby Cloud** 6:19
Deploy to production using Gatsby Cloud

50 **Gatsby Plugin Image** 8:11
Use gatsby-plugin-image for automatic image resizing,...

51 **Contentful Part 1** 4:59
Create a Contentful account, add a model and content

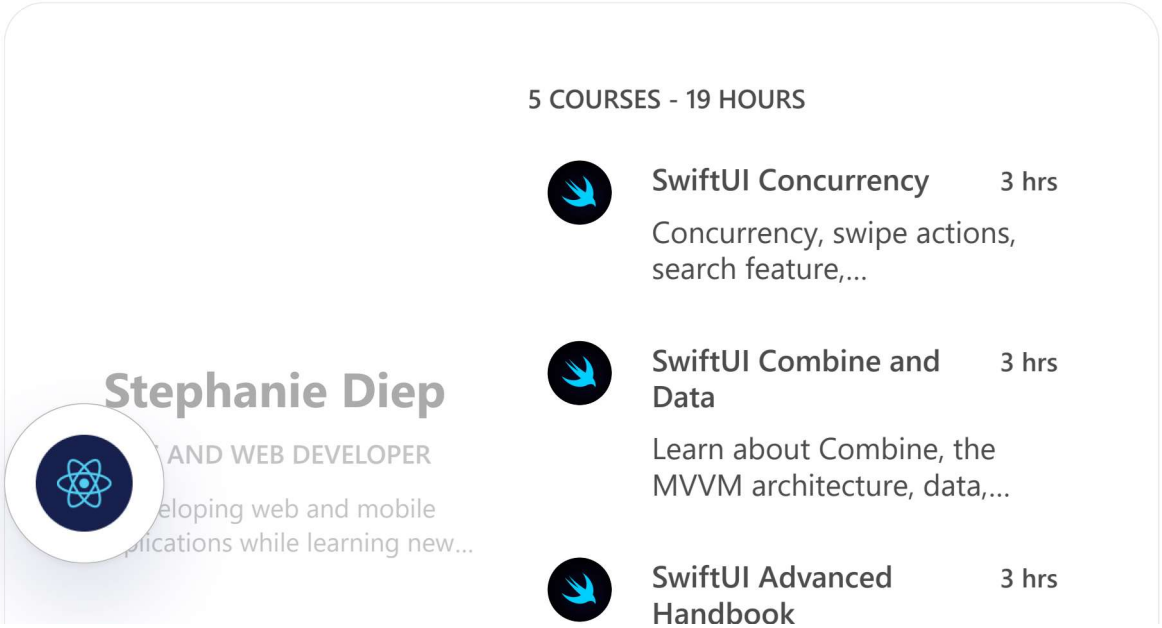
52 **Contentful Part 2** 8:52
How to use the Content Delivery API to fetch data from Contentful

53 **Gatsby and Shopify Part 1** 5:20
Create a Shopify store, generate a password and add products to t...

54 **Gatsby and Shopify Part 2** 13:21
Connect Shopify to Gatsby and display all products

Meet the instructor

We all try to be consistent with our way of teaching step-by-step, providing source files and prioritizing design in our courses.



An extensive series of tutorials covering advanced topics...