Trainer guide
Examples and exercises (React)
React-11 – Using external data

Contents

Trainer demo | Fetch external data on load

Link to environment

In this demo, you will show a simple data-fetching pattern in React that uses fetch within useEffect to request data from an external API, and when a response is received, the component's state is updated.

Working demo

- *To illustrate the goal, open this page (link) containing a working demo.*
- This is a simple website that displays a picture of a random dog.
- *Refresh the page.*
- Every time you refresh the page, a new photo is displayed.
- The photo is received from an external API.
- We're going to look at how to fetch external data in a React application so that we can recreate this site.

src/App.jsx

- Here we have the shell of our "random dog" React application.
    - We have the page title in an <h1> tag, and an <img> tag which will hold the photo.
- The variable "apiUrl" at the top of the file holds the URL of the API we need to make a request to.
- *Paste the value of apiUrl into your browser to show the JSON response that is returned.*
    - If we make a request to this API, you can see that a response is returned in JSON format.
        - This JSON object has two properties, "message" and "status".
        - "message" is the URL of the image that we want to display. We want this to be the value of the `src` attribute of our <img> tag.

- The first thing we need to achieve is to make a request to this API from our code.
- This can be achieved with the `fetch` method in JavaScript, which we can use to fetch a resource from a server. Let's write the code to achieve this.
- *Write the following code in the global scope of App.jsx, just below the declaration of apiUrl:*

```
fetch(apiUrl)
  .then((response) => response.json())
  .then((data) => console.log(data))
```

- Let's break down what this is doing.
    - First, fetch will send a request to the API. The fetch function will return a Promise. All Promises have a "then" property, which allows us to define each step in this sequence of asynchronous operations.
    - The next step is to take the response, and extract the JSON data using the `json()` method. The `json()` function also returns a Promise. So we return the result of calling `json()` and add another `then` to the chain.
    - The final step is to take the processed JSON data and log it to the console.
- *View console.*
    - In the console, you can see the JSON data which we've received from the API, including the "message" property containing the URL of the image we want to display in our application.

- The next question is – now that we have the data we need, how do we get it into our component?
- There is one main problem that we need to work around in order to solve this problem: because fetch is asynchronous, the App component will render before the data is received from the API - so, by the time we get the image data, our app has already rendered, but it won't know what image to display.
- Some amount of loading time is inevitable as we wait for a response from the API, so we can expect some delay before our image displays. But how will we let our component know when we've received the data?
- We can solve this problem by relying on the fact that React components re-render when they undergo a state change. We could store the image URL in state, initially with an empty value, and then when we get the data back from the API, we could update the state, which would cause our component to update and display the image. Let's try that:
    - *Import useState at top of file:*
      `import React, { useState } from 'react'`
    - *Update App to include a useState call. Don't forget to reference imageUrl in the src attribute of the <img >tag.*

```
export default function App() {
  const [imageUrl, setImageUrl] = useState('')

  return (
    <main>
      <h1>Go fetch!</h1>
      <img width={300} src={imageUrl} />
      </main>
  )
}
```

- So, when we receive the data, we could call `setImageUrl`, which would update our component.
- But currently our fetch happens outside of our component, and we can only access the `setImageUrl` function from within our App component because of how scope works in JavaScript.
- So, we will have to move the fetch inside our component somehow.
- But we also want to make sure that the fetch only happens **the first time the component is rendered** – otherwise we might make unnecessary calls to the API.
- We can achieve this with the useEffect hook.
  - *Import useEffect at top of file:*
    `import React, { useState, useEffect } from 'react'`
  - *Add useEffect call within App(), before the return statement:*

```
useEffect(() => {
  // run this code
}, [])
```

- By passing an empty dependency array as the second argument of useEffect, we are telling React to only run this code when our App mounts, and not on any subsequent renders.
- Now, let's move our fetching code into the useEffect:

```
useEffect(() => {
  fetch(apiUrl)
    .then((response) => response.json())
    .then((data) => console.log(data))
}, [])
```

- Now all we need to do is update the state when we receive the data by calling `setImageUrl`. Remember that the image URL was contained in the "message" property of the JSON object.

```
useEffect(() => {
  fetch(apiUrl)
    .then((response) => response.json())
    .then((data) => setImageUrl(data.message))
}, [])
```

- Now when we receive the image data and process it, the image URL is updated in state, which then prompts a re-render of our App component with the data it needs. Our React application now has simple data fetching!
- You have a link to this example if you ever need to reference it. You'll get a chance to put data fetching into practice in the practical project.

Solution reference