React and APIs

As you use React to create your front-end, you will probably need to make POST or GET requests at some point (e.g. using an API). As React is just a Javascript library and it runs client-side, you can use any JS method for POST or GET requests (e.g. jQuery, XmlHttpRequest, Fetch, Axios).

Let's try out fetching the JSON data from the given API endpoint (this API was chosen because it doesn't have an API key, which keeps things simple): https://uselessfacts.jsph.pl/api/v2/facts/random

The following example is a functional component used to display a random useless fact and its source (***UselessFact.jsx***):

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

export default function UselessFact() {
  const [randomFact, setRandomFact] = useState(""); //initialize randomFact state
variable to empty string
  const [factSource, setFactSource] = useState("");

  useEffect(() => {

const getFact = async () => {
      let response = await fetch(
        "https://uselessfacts.jsph.pl/api/v2/facts/random",
        {
          method: "get"
        }
      );
      let data = await response.json();
      setRandomFact(data.text);
      setFactSource(data.source_url);
    }
    getFact();
  }, []);

  let source = "";
  if (factSource) {
    source = `(Source: ${factSource})`;
  }
  return (
    <div>
      <h2>Today's random fact</h2>
      <div>{randomFact}</div>
      <div>{source}</div>
    </div>
  )
```

```
}
```

- In the above code, state variables are being used to store the fact text and the source URL.

> *Sample data*
>
> Loading the API endpoint URL in the browser will give you something like:
>
> ```
> {
>   "id":"56ffd45c1bd9faf72c148736a883a360",
>   "text":"Months that begin on a Sunday will always have a `Friday the 13th`.",
>   "source":"djtech.net",
>   "source_url":"http://www.djtech.net/humor/useless_facts.htm",
>   "language":"en",
>   "permalink":"https://uselessfacts.jsph.pl/api/v2/facts/56ffd45c1bd9faf72c148736a883a360"
> }
> ```
>
> We will only need to store the `text` and `source_url` (in `randomFact` and `factSource`, respectively).

- We are using the `useEffect()` hook function to make an API request. There are a couple things happening here.
  - The `useEffect()` hook function allows you to write a custom function **which runs after the component initially renders**. An effect is always run after every render.
  - API requests can be slower due to making requests from a different server so we want to use an asynchronous request. Since `useEffect()` *itself* should not be asynchronous, we need to create our asynchronous function **inside** the `useEffect()` hook.
  - Once the data is received (await response.json()), save the appropriate values in the state variables using the setter functions.
  - Right after defining our asynchronous function (which makes the API request), we run the function immediately. This triggers the API request.
  - The empty square brackets for the effect hook function are used for the dependency array. Any dependency listed there will trigger the effect if the value changes. Since we only want to run the effect once to load the random fact data, we can use empty brackets. This effectively sets the effect to only run once the component renders.
  - Conditional rendering was used for the fact source. Notice the if statement outside of the return(). This will render different text depending on if `factSource` is empty or not.

For more detailed info about component lifecycles and React and AJAX, read:

- https://retool.com/blog/the-react-lifecycle-methods-and-hooks-explained/
  - Note that when talking about lifecycle methods, you'll always come across the older class component syntax. This is because the lifecycle methods (e.g. componentDidMount())

are only available in class components. For functional components (newer, recommended syntax), we use hook functions to do the same thing but it's useful to understand the general lifecycle of a component (see diagram on next page, taken from this resource).

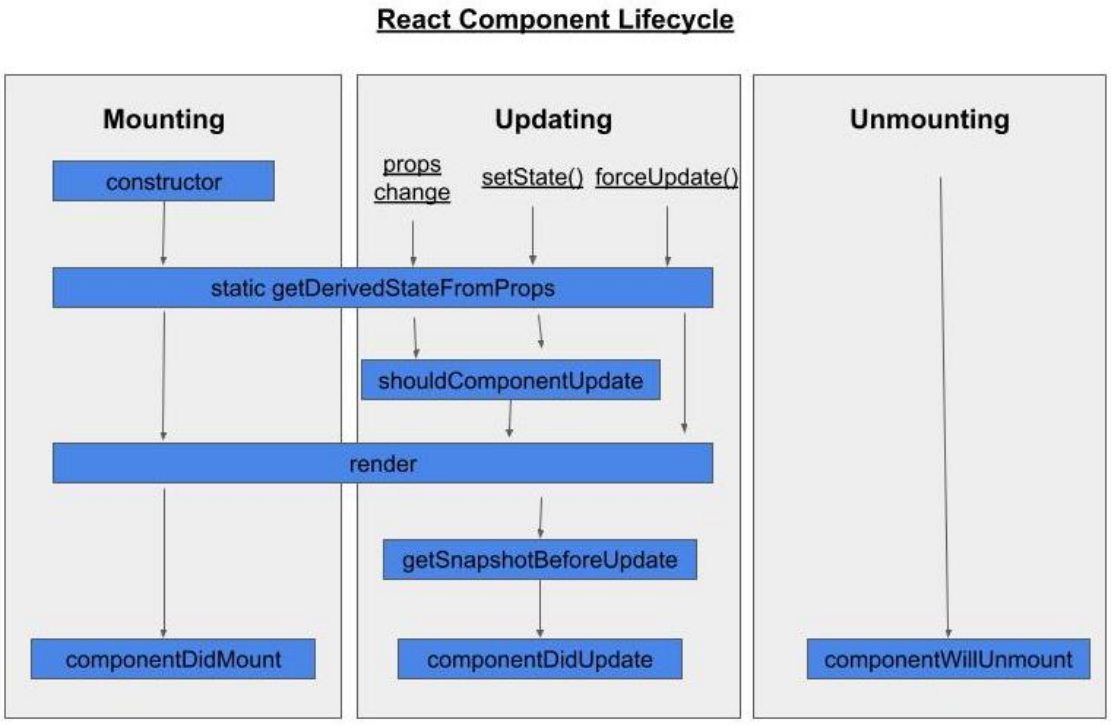

Figure 1. React component lifecycle (from: https://retool.com/blog/the-react-lifecycle-methods-and-hooks-explained/)