React State, Props and Hooks

State

- State refers to the important data a component needs to keep track off, and how the component renders if that data were to change.

- Example - you log onto your bank’s website, and your balance is displayed. You send an online payment to someone causing the balance to decrease. The balance now needs to be updated to reflect this transaction.

useState

- We manage our state variables using the useState module from React.

- We declare these state variables at the same level as functions and hooks within a React component

useState

- Here we are declaring a state variable called stateCounter:

```
const [stateCounter, setStateCounter] = useState(0);
```

name of variable

a setter function to change its value if needed

The starting value is 0.

- Here we declare a state variable called celestialObjects, which is meant to be an array:

```
const [celestialObjects, setCelestialObjects] = useState([]);
```

Importance of State

- Managing state variables is of paramount importance in React development.
    - When the value of a state variable is changed, the component will refresh and the updated data will be visible!

Let’s practice setting up some State Variables

Axios

- Before we talk about useState… we will be using Axios, a module that allows us to make asynchronous calls to an API.
- When making an asynchronous call, we initially get back a Promise from the server, the server will only agree to eventually send you the data you are looking for.
- In the meantime, your application can continue loading its components while we wait for the server's response.
- We define an anonymous function inside of a then(...) section that specifies the actions we should take when the server does finally reply with the data.

Axios

It’s best practice to place our Axios code in its own separate Service file:

```
import axios from 'axios';

export default {

    getCelestialObjects() {
        return axios.get('https://teapi.netlify.app/api/celestial-objects');
    }
}
```

Make an HTTP GET request

Axios

From a REACT component, we can now call the Axios methods in the service file:

```
function getPageData() {
        setIsLoading(true);
        CelestialObjectsService.getCelestialObjects()
        .then(
                (response) => {
                        setCelestialObjects(response.data);
                        setIsLoading(false);
                }
         )
        .catch(
                (error) => {
                // error logic
                }
        );
}
```

- The code inside of then(...) jumps into action asynchronously, that is to say only after the server gets back to you with the data
- The actual data (in JSON format) can be accessed with **response.data**.
- A catch(...) section will run if the we got either a 4XX or 5XX status code.

useEffect

- Suppose we need some data from an external API, we know the endpoint URI already and understand the JSON response.
    - For the purpose of this class, we are using a dependency called **Axios** to help us make API calls asynchronously.
- We must get this from the data right at the start of the component lifecycle - right after the component loads, useEffect can help us with that.
- The useEffect hook takes two parameters, a function (defined anonymously) and an array.

```
useEffect (
      () => { …. // the action you want to take }    ,
      [ ]
 );
```

useEffect

Component Code

```
 function getPageData() {
      setIsLoading(true);
      CelestialObjectsService.getCelestialObjects()
      .then(
            (response) => {
                  setCelestialObjects(response.data);
                  setIsLoading(false);
            }
       );
}

useEffect (
      () => {
            getPageData();
      },
      []
);
```

Service JS File

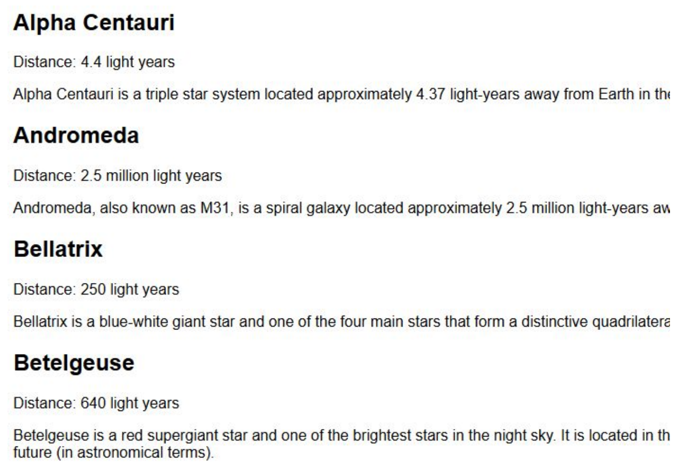
```
import axios from 'axios';

export default {
  getMyBookmarks() {
    return axios.get('/bookmarks');
  }
}
```

Using the Data

Once the data has been collected into the array, we can display it in our JSX code:

Each JSON object becomes an <article>

```
    celestialObjects.map(
            celestialObject => (
                    <article key={celestialObject.id}>
                            <h2>{celestialObject.name}</h2>
                            <p>Distance: {celestialObject.distance}</p>
                            <p>{celestialObject.description}</p>
                    </article>
            )
     )
```



Alpha Centauri

Distance: 4.4 light years

Alpha Centauri is a triple star system located approximately 4.37 light-years away from Earth in th

Andromeda

Distance: 2.5 million light years

Andromeda, also known as M31, is a spiral galaxy located approximately 2.5 million light-years aw

Bellatrix

Distance: 250 light years

Bellatrix is a blue-white giant star and one of the four main stars that form a distinctive quadrilatera

Betelgeuse

Distance: 640 light years

Betelgeuse is a red supergiant star and one of the brightest stars in the night sky. It is located in th
future (in astronomical terms).

Let’s implement an Axios service, call it when the component loads, then get its data

Props

- Props are a way for a parent component to pass data to its child components. Suppose we have the following child component:

```
export default function QuizAnswer( { answer, isCorrect } ) {
        // … component code
}
```

- We’ve defined two props: answer and isCorrect
- It will be QuizAnswer’s parent’s responsibility to populate the props with values.

Props

Now let’s see the parent component’s code, more specifically in its return JSX:

parent

```
  {
      quizData.answers.map(
            (answer, index) => (
                  <li key={index} className={styles.answerItem}>
                        <QuizAnswer
                              answer={answer}
                              isCorrect={answer === quizData.correctAnswer}
                        />
                  </li>
            )
      )
}
```

child

```
export default function QuizAnswer( { answer, isCorrect } ) {
      // … component code
}
```

Props

- Now within the **child component** you can use props the same way you would as any variable:

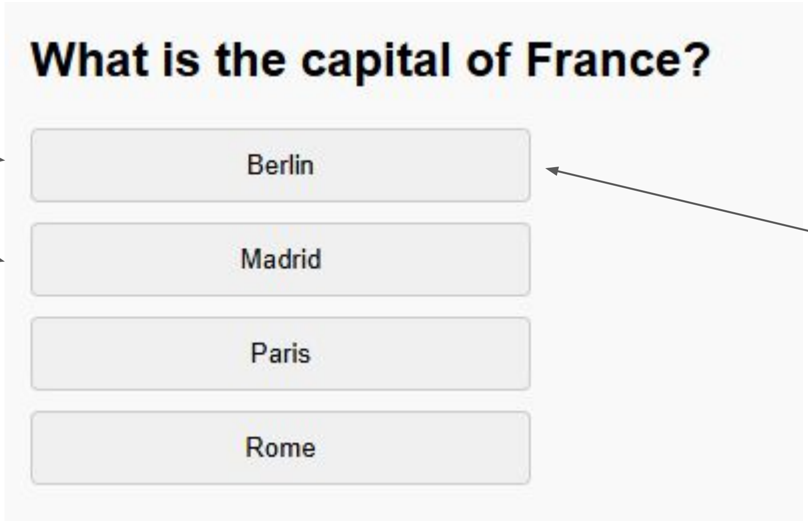
child

```
export default function QuizAnswer( { answer, isCorrect } ) {
  function handleClick(event) {

    const button = event.currentTarget;
    if (isCorrect) {
      button.classList.add(styles.correct);
    } else {
      button.classList.add(styles.incorrect);
    }
  }
  return (
    <button className={styles.answerButton} onClick={handleClick}>
      {answer}
    </button>
  );
}
```

Props

In summary, we will make four QuizAnswer child components.

answers: [
'Berlin',
'Madrid',
'Paris',
'Rome'
]


What is the capital of France?

Example - for the first QuizAnswer component, its prop values will be:

answer = Berlin

isCorrect = false

Let’s practice using Props

Updating a Parent's State

A child component can update a parent component's state by receiving a function as a prop, then calling that function

parent

```
function handleAnswer(answer) {
   setSelectedAnswer(answer);
 }

 return (
     …
     <ul className={styles.answerList}>
              { quizData.answers.map(
                                    (answer, index) => (
                                    <li key={index} className={styles.answerItem}>
                                               <QuizAnswer
                                               …
                                               onAnswer={handleAnswer}
                                               …
                                                />
                                    </li>
                                    )
                         )
              }

     </ul>
   </div>
 );
```

child

```
export default function QuizAnswer({ … onAnswer … }) {

           function handleClick(event) {
             …
             onAnswer(answer); // Notify parent component
           }

           return (
             <button className={styles.answerButton} onClick={handleClick}>
               {answer}
             </button>
           );

   …
}
```

The function handleAnswer is passed to the child component like a prop

We now call this function from the child component while passing in a value for its parameter

Let’s practice changing the parent’s state