PROJECT # Dynamically Change State-Bound Content on Successful API Call in React

Introduction

Elements of the rendered content of a React component can be bound against properties of the component's `state`. Whenever we update the values of `state`, we automatically update the content that references its attributes. In this guide, we'll take a look at how to update the rendered content of a component dynamically from data fetched from a third-party API. The process is as follows:

1. Establish an object `data` within the component's `state`
2. Establish the content to be rendered with sections referencing attributes in `data`
3. Create an event handler that fetches data from an API
4. On success of the API call, update the component's `state`

We'll be using `jQuery` as our library to make an HTTP request against an API endpoint.

Setup

For this example, we will fetchthe latest Bitcoin price from CoinDesk's public API. Our content will include the price of USD, GBP, and EUR against BTC (Bitcoin). The JSON structure of the data returned by the API endpoint CODE https://api.coindesk.com/v1/bpi/currentprice.json will be as follows:

CODE JSON

```
{
  time: {
    updated: "Sep 16, 2020 18:43:00 UTC",
    updatedISO: "2020-09-16T18:43:00+00:00",
    updateduk: "Sep 16, 2020 at 19:43 BST"
  },
  disclaimer: "This data was produced from the CoinDesk Bitcoin Price Index
(USD). Non-USD currency data converted using hourly conversion rate from
openexchangerates.org",
  chartName: "Bitcoin",
  bpi: {
    USD: {
      code: "USD",
      symbol: "&#36;",
      rate: "11,091.2212",
      description: "United States Dollar",
      rate_float: 11091.2212
    },
    GBP: {
      code: "GBP",
      symbol: "&pound;",
      rate: "8,436.3156",
      description: "British Pound Sterling",
      rate_float: 8436.3156
    },
    EUR: {
      code: "EUR",
      symbol: "&euro;",
      rate: "9,395.0075",
      description: "Euro",
      rate_float: 9395.0075
    }
  }
}
```

Setting the State

Call your component `BTCPrice`. Within its constructor, set the value of an attribute `data` to have the same structure as the expected return value of the API endpoint.

CODE JAVASCRIPT

```
import React from 'react';
import $ from 'jquery';

export default class BTCPrice extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      data: {
        time: {
          updated: "",
          updatedISO: "",
          updateduk: ""
        },
        disclaimer: "This data was produced from the CoinDesk Bitcoin Price Index (USD). Non-USD currency data converted using hourly conversion rate from openexchangerates.org",
        chartName: "Bitcoin",
        bpi: {
          USD: {
            code: "USD",
            symbol: "&#36;",
            rate: "",
            description: "United States Dollar",
            rate_float: 0
```

```
        },
        GBP: {
          code: "GBP",
          symbol: "&pound;",
          rate: "",
          description: "British Pound Sterling",
          rate_float: 0
        },
        EUR: {
          code: "EUR",
          symbol: "&euro;",
          rate: "",
          description: "Euro",
          rate_float: 0
        }
      }
    }
  }
}
```

Setting the Rendered Content

The component will render content that references attributes in `data`, which can be referenced in turn from the component's `state`. Copy the `render()` function below and paste it within your component.

CODE JAVASCRIPT

```
render() {
  return (
    <div>
      <h1>
        How Much is 1 BTC?
```

```
        <small>
          {this.sate.data.time.updated}
        </small>
      </h1>

      <h2>USD: {this.state.data.bpi.USD.rate}</h2>
      <h2>GBP: {this.state.data.bpi.GBP.rate}</h2>
      <h2>EUR: {this.state.data.bpi.EUR.rate}</h2>

      <hr/>
      <button>
        Fetch Latest
      </button>
    </div>
  );
}
```

Notice how we position the currency's rates within the different HTML dom elements. Think of it as a template wherein the references of `data` attributes are placeholders for values that may change whenever the `state` of the component is updated. We say that these pieces are bound to the `state`. In this case, the attribute values of `data` that we will be referencing represent the three different currencies and their rates against Bitcoin, namely:

1. `this.state.data.bpi.USD.rate` for USD
2. `this.state.data.bpi.GBP.rate` for GBP
3. `this.state.data.bpi.EUR.rate` for EUR

We also reference the `data.time.updated` value at the top to indicate the timestamp for when these prices were taken.

Fetching Data from an API

We'll now create our event handler function, which contains logic to make an AJAX request against CoinDesk's API. To do this, copy the following function and paste it within your component.

CODE JAVASCRIPT

```
fetch() {
  var context = this;

  $.ajax({
    url: 'https://api.coindesk.com/v1/bpi/currentprice.json',
    method: 'GET',
    dataType: 'json',
    success: function(response) {
      context.setState({
        data: response
      });
    }
  });
}
```

Updating State on Success

There are two key ideas in this function's implementation. First, you create a temporary variable called `context` that points to `this` referring to the instance of this component. This is needed so that upon a successful AJAX call given by the `function(response)` passed to `jQuery`'s AJAX call's `success` property, you can invoke the `setState({ data: response })` call from `context`. Without `context`, `this` within `function(response)` will refer to the calling function and not the instance of the component. Second, you have to set the `dataType` property to `json` since we're

expecting `response` to represent a `json` object returned by the API endpoint.
Once `setState()` is called, the component will re-render the content with all of the `state`'s updated values.

Binding to a Button

The `fetch()` method of the component will be triggered every time the user clicks the button. Update the button in HTML within the `render()` method of the component and bind the `fetch()` method on the button's `onClick` attribute.

CODE JSX

```
<button onClick={this.fetch.bind(this)}>
  Fetch Latest
</button>
```

Include the `.bind(this)` function to retain the value of `this` when it is used within `fetch()`, that is, a reference to the instance of the component.

Overall Code

CODE JAVASCRIPT

```
import React from 'react';
import $ from 'jquery';

export default class BTCPrice extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      data: {
        time: {
```

```
      updated: "",
      updatedISO: "",
      updateduk: ""
    },
    disclaimer: "",
    chartName: "",
    bpi: {
      USD: {
        code: "USD",
        symbol: "&#36;",
        rate: "",
        description: "United States Dollar",
        rate_float: 0
      },
      GBP: {
        code: "GBP",
        symbol: "&pound;",
        rate: "",
        description: "British Pound Sterling",
        rate_float: 0
      },
      EUR: {
        code: "EUR",
        symbol: "&euro;",
        rate: "",
        description: "Euro",
        rate_float: 0
      }
    }
  }
}

fetch() {
  var context = this;

  $.ajax({
    url: 'https://api.coindesk.com/v1/bpi/currentprice.json',
    method: 'GET',
    dataType: 'json',
    success: function(response) {
      context.setState({
        data: response
      });
```

```
    }
  });
 }

 render() {
  return (
   <div>
    <h1>
     How Much is 1 BTC?
     <small>
      {this.sate.data.time.updated}
     </small>
    </h1>

    <h2>USD: {this.state.data.bpi.USD.rate}</h2>
    <h2>GBP: {this.state.data.bpi.GBP.rate}</h2>
    <h2>EUR: {this.state.data.bpi.EUR.rate}</h2>

    <hr/>
    <button
     onClick={this.fetch.bind(this)}
    >
     Fetch Latest
    </button>
   </div>
  );
 }
}
```

After a user clicks the button, an AJAX call will fetch the data from CoinDesk's API. On success of the call, the component will invoke its `setState()` method, causing it to re-render the content with updated values.

Conclusion

You now have the necessary facilities to perform dynamic updating of a component's rendered content. The practice is to maintain a state object within the component. Values from the state object can be referenced in the rendered content. Once an event occurs in any part of the component, the succeeding logical process will be to update any values within the state if

needed. An update in the state will cause a re-rendering of the content with all the updated values without needing a refresh to the page or manually target DOM elements and update its inner content.

Source:
https://app.pluralsight.com/guides/dynamically-change-state-bound-content-with-react.js-on-successful-jquery-request

Hashtags: #WebDevelopment #frontEndWebDevelopment #clientSideFramework #React #Bitcoin #DeFi #decentralizedFinance