Will Soares
Posted on Feb 13, 2018

How to build an image carousel with React

#react #carousel #javascript

```
const imgUrls = [ ];

class ImageSlide extends React.Component { }

class Carousel extends React.Component {
  constructor (props) {
    super(props);

    this.state = {
      currentImageIndex: 0
    };

    this.nextSlide = this.nextSlide.bind(this);
    this.previousSlide = this.previousSlide.bind(this);
  }

  previousSlide () {
    const shouldResetIndex = this.state.currentImageIndex === 0;
    const index = shouldResetIndex ? imgUrls.length - 1 : this.state.currentImageIndex;
```

Hey everyone!

For this post, I decided to work with something I'm currently passionate about, React.

Recently, I'm having the opportunity to work a lot with React and as its advantages become clearer to me, I'm more and more willing to dive into this path of learning to build user interfaces powered with the library.

As you may know, [React]() is a JavaScript library with which you can build complex interactive UIs following a component-based approach. It uses techniques and concepts that focus on making the interaction with the interface a more efficient task. By using a concept called virtual Dom, which is a lightweight representation of the real DOM, React makes interaction with the interface a really fast task, since for every change it compares the virtual DOM to the real DOM and updates only the portion that has changed.

Alright, so that is only one of the reasons why React is much more powerful than other libraries when you use it to build UIs. As in this post I'll focus on a real example of

using React, you can look at the documentation if you find yourself lost while reading the code or if you don't know any concepts mentioned here.

Component-based thinking

One of the first things to do when you start creating a UI, is to think about it as a set of components that will wrap parts of your interface and then work together to create a good user experience, that is one of the things React does to you, it changes the way you organize your apps.

If you look at the header image for this post you'll get the idea of what we are going to build here. It looks really simple, right? And it, in fact, can be simple since we are using React :)

You should be aware that there are tons of ways of doing an image carousel in React, regarding the way you organize components and even how many components you create. For this example I decided to create three basic components, the first, the `Carousel` component, will be the wrapper for the entire layout and logic, the second, `ImageSlide` component, will simply render the image slides for the carousel, and the third will be the `Arrow` component, which will act as both the right and left arrows for the carousel transition. So, with that, you will have an instance of the `ImageSlide` for each image you give as the input for the carousel and two instances of the `Arrow` component, one for each arrow.

First things first, let's build the `Carousel` component.

The `Carousel` component

Before anything, you have to tell React in which part of your HTML file you want it to render your component.

In your HTML file, add this:

```
<div id="container">
  <!-- Your component will be rendered here. -->
</div>
```

In your JavaScript file, enter this:

```
ReactDOM.render(
  <Carousel />,
  document.getElementById('container')
);
```

As you see, we are binding the div container to the Carousel component, so React will use that placeholder to render your entire component.

Notice that you have to have the `React` and `ReactDOM` libraries available for you in your script. If you are trying this out in some sort of Code Playground such as CodePen or JsFiddle you can simply add the relevant scripts in the settings section. However, if you have a setup for React development on your local machine you probably know what to do :)

Now it's time to set up our `Carousel` component.

```
class Carousel extends React.Component {
  render () {
    return (
      <div className="carousel"></div>
    );
  }
}
```

Here we are creating a simple wrapper class that will be responsible to handle all the logic you'll have in your interface. The render method of this class will be responsible to return the markup for your entire layout, so we'll add all the other components to the `div` block being returned.

You should notice that here we are extending the `React.Component` class in order to declare that this specific class is going to be a React component. It is important to point that this is the ES6 equivalent of the `React.createClass` method. The former is just a "syntactic sugar" provided in the ES6 set of features. With that, you'll have several differences while implementing your component methods. Check this well written post by Todd Motto to see the details of each option.

As we already have the setup for the `Carousel` class, we should start looking at what we are going to render inside of it.

The `ImageSlide` component

This is going to be a really simple component, with no logic attached to it. This is a common pattern while building components in React and these types are known as `stateless` or `functional` components. The reason for that is due to the inexistence of a logic to control the state of those components, since they do not have a declared state. They end up being simply JavaScript functions that receive parameters (or not) and return a markup built based upon those input values (or not).

Let's see what the `ImageSlide` component looks like.

```
const ImageSlide = ({ url }) => {
  const styles = {
    backgroundImage: `url(${url})`,
    backgroundSize: 'cover',
    backgroundPosition: 'center'
  };

  return (
    <div className="image-slide" style={styles}></div>
  );
}
```

You can see that there is not much to do with this component. It should basically receive the image URL and create the required markup so it will act as one of the slides in our carousel.

`ImageSlide` is a function that receives a String, which is the image URL, creates an object that will describe how the component should be styled and returns the markup filled with the info we provided. As mentioned, you will have one instance of this component for each URL you provide in the array of images.

Regarding the style of this component, you can see that the image is being set as its background. However, it is up to you to style your component as you want, it won't affect the purpose of this post.

So after creating the markup for the image slides, we should add it to the render method in the `Carousel` component.

```
class Carousel extends React.Component {
  render () {
    return (
      <div className="carousel">
        <ImageSlide url={ imgUrl } />
      </div>
    );
  }
}
```

The URL is passed to the `ImageSlide` component as a property in the `props` object. Later in this post, we'll see how to properly get the image URL form the array of images used as a reference.

The `Arrow` component

```
const Arrow = ({ direction, clickFunction, glyph }) => (
  <div
    className={ `slide-arrow ${direction}` }
    onClick={ clickFunction }>
    { glyph }
  </div>
);
```

This component is even simpler since we don't have to setup anything before returning its markup. I decided to use three properties here because this component is being used both for the left arrow and the right one. Due to that, its implementation should be a little more generic. The `direction` property will tell the component which class to use for each instance (`left` or `right`). The `clickFunction` describes what should happen when each arrow is clicked and finally the `glyph` component refers to what will be the content of this component, that means what will be rendered.

With that, let's add both arrows to the `Carousel` component.

```
class Carousel extends React.Component {
  render () {
    return (
      <div className="carousel">
        <Arrow
          direction="left"
          clickFunction={ this.previousSlide }
          glyph="&#9664;" />

        <ImageSlide url={ imgUrl } />

        <Arrow
          direction="right"
          clickFunction={ this.nextSlide }
          glyph="&#9654;" />
      </div>
    );
  }
}
```

From here we can have a better idea of the final markup we'll have for our carousel.

Next, we should go back to the `Carousel` component since there are still several things to finish.

You notice that we are passing two different functions to the `Arrow` s components. Those are responsible for handling the transition of slides. But how do they do that?

First, we should look at the states we need to create in order to tell the wrapper component which image it should render at each time. So, let's set up the `constructor` function so we can create the initial state for the `Carousel` component.

```
class Carousel extends React.Component {
  constructor (props) {
    super(props);

    this.state = {
      currentImageIndex: 0
    };
  }

  render () {...}
}
```

The first thing to do within the `constructor` function is to call `super()` passing it the props as a parameter, in case you want to access properties through the `this` keyword within this context. For now, it is optional to send the `props` object since we are not using props within the constructor.

Initially, we are setting a state called `currentImageIndex` set to `0`. This is going to hold the current index for the image that has to be rendered on the screen at each time. Here we are starting from the first image in the images array.

This state is going to be used to get the correct URL passed to the `ImageSlide` component. Let's check how to do that.

```
class Carousel extends React.Component {
  constructor (props) {...}

  render () {
    return (
      <div className="carousel">
        <Arrow .../>

        <ImageSlide url={ imgUrls[this.state.currentImageIndex] } />

        <Arrow .../>
      </div>
      );


    }
  }
```

After this, all we have to do is to tell the component how to update that state according to user interaction. That work is related to our `Arrow` components, and since we are already passing two functions (`previousSlide` and `nextSlide`) as properties, we now have to implement them.

You will notice that these two functions are analogous. All they do is to update the `currentImageIndex` state by one, either by adding or removing from it. There is only one detail to point here. There will be situations in which we will have to reset the value for the current index since it will at some point reach the maximum or the minimum index for the array of images. Therefore, it is important to check the length of the array in order to know whether we should reset the index or not.

```
class Carousel extends React.Component {
  constructor (props) {...}

  previousSlide () {
    const lastIndex = imgUrls.length - 1;
    const { currentImageIndex } = this.state;
    const shouldResetIndex = currentImageIndex === 0;
    const index =  shouldResetIndex ? lastIndex : currentImageIndex - 1;

    this.setState({
      currentImageIndex: index
    });
  }

  nextSlide () {
    const lastIndex = imgUrls.length - 1;
    const { currentImageIndex } = this.state;
    const shouldResetIndex = currentImageIndex === lastIndex;
    const index =  shouldResetIndex ? 0 : currentImageIndex + 1;

    this.setState({
      currentImageIndex: index
    });
  }

  render () {...}
}
```

For the `previousSlide` function you can notice that the reset condition is set to be the

For the previousSlide function you can notice that the reset condition is set to be the case in which the currentImageIndex state is 0, which means that if the index is pointing to the first image in the array and then the user clicks the left arrow, the index should then point to the last image in the array (imgUrls.length - 1).

The nextSlide does quite the same, the difference is that if the index is currently pointing at the last image, then it should be reset to point at the first (index = 0).

In all the remaining situations both methods simply change the mentioned state by one in order to get the previous or next image.

At last, there is one important thing to notice here. In order to update states in a React component, we have to use the setState method. This method is responsible for telling React that it should update that component and its children. This is the primary way of updating your user interface.

Therefore, whenever you click on the left or right arrow, you are basically updating the state of the currentImageIndex and consequently updating your interface with the new image slide.

However, in order to access the this keyword within those two functions, you have to properly bind the context to them. A common way to do that is in the component's constructor method.

```
class Carousel extends React.Component {
  constructor (props) {
    ...

    this.nextSlide = this.nextSlide.bind(this);
    this.previousSlide = this.previousSlide.bind(this);
  }

  previousSlide () {...}

  nextSlide () {...}

  render () {...}
}
```

Finally, we have implemented all the logic for our carousel component. You can take a look at the working demo and the full version of the code in this codepen.

PS1.: as it was not the purpose of the post, I omitted the styles used for the components, thus if you are interested in that you should take a look at the codepen