Introduction to Bulma with React

joshtronic

> I'll come out and say it, I'm still a fan of Bootstrap. With that, I'm not oblivious to the many alternatives that are floating around out there. One of the more inspired CSS frameworks I have come across recently is Bulma. It does everything I need a CSS framework to do and does a great job of simplifying certain things while at the same time making them more powerful. Oh, and there's no shortage of React wrappers for it!

For this article we're going to be playing around with `react-bulma-components`, one of the more popular implementations of the Bulma CSS framework in React.

Not only is this library very well documented, it appears to support just about everything available in Bulma and is currently tracking against the latest release of Bulma.

So let's go!

Getting Started

To get started with `react-bulma-components` we'll need to add it to our project:

Via npm

```
$ npm install --save react-bulma-components
```

Or via Yarn

```
$ yarn add react-bulma-components
```

With our dependency added, next we'll need to `import`.

What I found great about this library is that you can import "full" versions of the components that come with the associated styles and eliminate the need to link to or import any additional styles.

If you wanted the `Button` component in all of it's stylistic glory, simply:

```
import { Button } from "react-bulma-components/full";
```

The Components

As mentioned, the `react-bulma-components` library supports just about everything Bulma has to offer as nice React-ready components. Here's a run down of what you will have available out of the box:

- `Box` - A white box with a shadow to wrap up content.
- `Breadcrumb` - Navigational breadcrumb menu.
- `Button` - That clicky thing usually found on forms ;)
- `Card` - A more formal "card"-like component (more than just a box).
- `Column` - Wrapper and columns for the grid system.

- `Container` - A no-frills content wrapper that responds to screen size.
- `Content` - A generic wrapper for content that contains HTML.
- `Dropdown` - An interactive drop down menu.
- `Footer` - A wrapper for page footer content, can contain anything though!
- `Form` - Form controls such a `Form.Input` and the like.
- `Heading` - Not to be confused with a header, this is for `h#` type content.
- `Hero` - This post needs a hero... a/k/a a jumbotron.
- `Icon` - A wrapper for any icon font you're comfortable with (Font Awesome, et cetera).
- `Image` - A wrapper for responsive image content.
- `Level` - Similar to columns, allows you to horizontally align content.
- `Loader` - Personal fave, a simple loader (which can be used with `Button`s!)
- `Media` - A CSS framework wouldn't be complete with a social media UI object.
- `Message` - Similar to a growl notification.
- `Menu` - A vertical menu for all your side menu needs!
- `Modal` - Classic modal UI component that can hold any content you'd like.
- `Navbar` - Not to be confused with `Heading`, this component makes a good header navigation bar.
- `Notification` - Similar to `Message` but without a title bar.

- `Pagination` - For all of your pages and pages of content.
- `Panel` - A control for compact controls. A hybrid menu and card, super powerful.
- `Progress` - Native HTML progress bars that can be colorized and sized easily.
- `Section` - Container for grouping content on your page.
- `Tabs` - Another horizontal nav with a “tabular” look and feel.
- `Table` - As per the documentation: “The inevitable HTML table”.
- `Tag` - You may refer to this as a badge or a label.
- `Tile` - One of the more inspired components, a simple way to implement a Pinterest/Metro-like tiled UI.

More than enough to really do some damage on your next project!

The Basics

Like most CSS frameworks out there, Bulma comes with styles for your common components and a series of customizations that can be accomplished by adding different classes to your elements.

`react-bulma-components` simplifies things further by providing components for each of the major elements, and eliminates the need for juggling class names in favor of passing in properties to your components.

Want to make a large button that uses the `danger` color, has rounded corners and is an outline? No problem:

```
<Button color="danger" size="large" rounded outlined>Wowza!</Button>
```

This is all well and good but what if we wanted that button to be a link (i.e. an anchor element)?

Fortunately, all of the `react-bulma-components` can accept a `renderAs` property which allows you to define what sort of element should be used to render the component.

Without the `renderAs` property the aforementioned `Button` will be rendered as... you guessed it! A `button` element.

Along with the `renderAs` property we should also include an `href` to tell it where to send folks when they click on the link:

```
<Button
  renderAs="a"
  href="https://alligator.io"
  color="danger"
  size="large"
  rounded
  outlined
>
  Wowza, it's a link!
</Button>
```

In fact, all of our Bulma components can accept whatever properties you may throw that them. That means you could always do some advanced styling with a `style` properties or add some additional CSS classes with `className`.

When using `className`, any classes you supply will be prepended to the library-generated list of Bulma class names.

Colors

Similar to most modern CSS frameworks, Bulma comes with a color theme that uses some semantic naming conventions.

These theme colors include `primary`, `link`, `info`, `success`, `warning`, and `danger`.

In addition, there are also some more literal colors available: `black`, `dark`, `light`, and `white`.

Components that support colors accept a `color` property:

```
<Button color="success">Alligators are the best!</Button>
```

Which will assign the correct color class to the rendered element. Because this color property assigns classes back to the rendered element, you can't just assign an arbitrary color value.

Sizes

Working with sizes with Bulma is actually a bit less straightforward than working with colors. This is because some components (like `Button`) use textual names for the sizes while others (like `Heading`) use numerical values:

```
<Heading size={1}>Large Heading</Heading>
<Heading size={2}>Not So Large Heading</Heading>

<Button size="large">Large Button</Button>
<Button size="small">Small Button</Button>
```

For components that accept textual sizes, you have `small`, `normal`, `medium`, and `large` available. The `normal` size is the one that is used by default when no `size` is specified.

"Just Works" Grid

Something that the Bulma marketing touts is that their grid system is "the simplest grid system".

Having used a number of grid systems, and even writing my own back in the day, I will admit it's pretty freakin' simple to use.

Even though it's simple, it's still feature-rich with things like responsive breakpoints, column offsets and nesting.

What I found very impressive is that while the Bulma grid system does work quite well in a standard 12 column layout, it doesn't force you into said paradigm.

This flexibility is also where the `size` property takes a turn. Unlike a `Button` that uses the

`small`...`large` naming convention, `Column` components use a different set of `size` names that correspond to how much space the column will occupy.

Column sizes are also divided into two types, percentages and by number of columns in a 12 column layout.

For percentage based sizing you can set `size` to be `one-fifth`, `one-quarter`, `one-third`, `half`, `two-thirds`, or `three-quarters`:

```
<Columns>
  <Columns.Column size="one-fifth">20%</Columns.Column>
  <Columns.Column>80%</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size="one-quarter">25%</Columns.Column>
  <Columns.Column>75%</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size="one-third">33.333333333%</Columns.Column>
  <Columns.Column>66.666666667%</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size="half">50%</Columns.Column>
  <Columns.Column>Also 50%</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size="two-thirds">66.666666667%</Columns.Column>
  <Columns.Column>33.333333333%</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size="three-quarters">75%</Columns.Column>
  <Columns.Column>25%</Columns.Column>
</Columns>
```

And for sizes based on a 12 column layout you set the `size` to the numeric value between `1` and `12`:

```
<Columns>
  <Columns.Column size={1}>One</Columns.Column>
  <Columns.Column>Eleven</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={2}>Two</Columns.Column>
  <Columns.Column>Ten</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={3}>Three</Columns.Column>
  <Columns.Column>Nine</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={4}>Four</Columns.Column>
  <Columns.Column>Eight</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={5}>Five</Columns.Column>
  <Columns.Column>Seven</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={6}>Six</Columns.Column>
  <Columns.Column>Six</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={7}>Seven</Columns.Column>
  <Columns.Column>Five</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={8}>Eight</Columns.Column>
  <Columns.Column>Four</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={9}>Nine</Columns.Column>
  <Columns.Column>Three</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={10}>Ten</Columns.Column>
```

```
  <Columns.Column>Two</Columns.Column>
</Columns>
<Columns>
  <Columns.Column size={11}>Eleven</Columns.Column>
  <Columns.Column>One</Columns.Column>
</Columns>
```

Conclusion

With over 30,000 stars on GitHub, it's hard to consider Bulma just another CSS framework in an already crowded space.

I personally found it quite familiar coming from other CSS frameworks but it also brought it's own level of simplicity and flexibility that made it easy to pick up.

Couple this simplicity with the React bindings provided by `react-bulma-components`, I will definitely be considering it for my next project.

I hope you found this high-level overview of Bulma and `react-bulma-components` informative and helpful, cheers!

stay in the loop with our monthly web dev newsletter:

you@awesome.com

Subscribe!

follow us @alligatorio

more react

React Icons Gets You Access to Hundreds of Open Source Icons

Get a Performance Boost in React with PureComponent

Build a Beautiful Photo Gallery PWA with React + JHipster

Using Lightboxes to Create Stunning Image Galleries in React

all react posts

Published: October 12, 2018

Catch up with the latest in front-end web dev with our monthly newsletter:

you@unicorn.com

Subscribe

Write for Us

About

Contact

Credits

Privacy