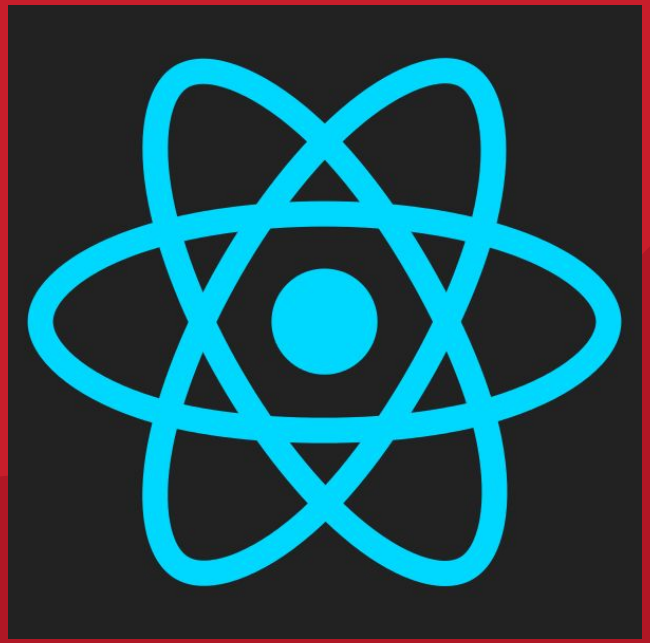
React.js

a crash course

Jake Zimmerman
January 29th, 2016



CRASH COURSE

Key Features of React.js

Easily express user interfaces

Richly express the visual elements of a design, as well as the interactions users can take on them

Makes for better maintainability

Abstracts the browser's DOM

Write your UI modularly up front, then render it down to the platform you're using

Render to the browser DOM, an HTML string (server-side!), or to native libraries (React Native)

Data flows are predictable

Control the flow to control the logic

Flow is always one-directional to avoid difficult to debug "magic"

JSX Syntax (two links)

JSX Syntax

It's (basically) just HTML

```
var MyComponent = React.createClass({
  render: function() {
    return (
      <div>
        <h3>Hello, world!</h3>
        <p>React is quite cool.</p>
      </div>
    );
  }
});
```

Define and nest custom components

```
var App = React.createClass({
  render: function() {
    return (
      <div>
        <Heading />
        <MyComponent />
        <Footer />
      </div>
    );
  }
});
```

Embed JavaScript expressions

```
var MyComponent = React.createClass({
  render: function() {
    return (
      <div>
        <p>React is {this.props.adv}
cool.</p>
      </div>
    );
  }
});

...

<MyComponent adj="remarkably" />

...
```

JSX Syntax -- Gotcha!

Always one node returned from render

```
var MyComponent = React.createClass({
  render: function() {
    return (
      <h3>Hello, world!</h3>
      <p>React is quite cool.</p>
    );
  }
});
```

INCORRECT:
two nodes returned from render

Need to use self-closing tags

```
var MyComponent = React.createClass({
  render: function() {
    return (
      <div>
        <h3>Hello, world!</h3>
        <hr>
        <p>React is quite cool.</p>
      </div>
    );
  }
});
```

INCORRECT:
hr takes no children (make self-closing)

```
<hr />
```

classNameName == class, htmlFor == for

```
var MyComponent = React.createClass({
  render: function() {
    return (
      <div>
        <h3 class="title">
          Hello, world!
        </h3>
      </div>
    );
  }
});
```

INCORRECT:
class should be className

```
<h3 className="title">
```

Component Specs and Lifecycle (link)

this.props & this.state

this.props

```
var Contact = React.createClass({
  render: function() {
    return (
      <div>
        <Header
          active="contact" />
        <MyComponent
          adj="amazingly" />
        <Footer
          date={today}
          author="@jez" />
      </div>
    );
  }
});
```

this.state

```
var MyComponent = React.createClass({
  getInitialState: function() {
    return {enabled: false};
  },

  componentDidMount: function() {
    var data = asyncLoadData();
    this.setState(enabled: data.enabled);
  },

  render: function() {
    if (this.state.enabled) {
      return <h3>Hello, world!</h3>;
    }
    else {
      return <h3>Goodbye, world D:</h3>;
    }
  },
});
```

this.props & this.state

this.props

Used for initializing a component

`this.props` is never mutated; don't try to overwrite anything in this.props

this.state

Used for determining when to re-render a component.

`this.state` is also never mutated, but only for technical reasons. Use `this.setState()` to mutate the state.

Some important lifecycle methods

render

Should be a “pure function of props and state”

Basically, as long as you aren’t trying to interact with global variables or the DOM you’re fine.

Lets React only re-render when needed.

getInitialState, propTypes

Useful for initialization and handling edge cases.

One note: don’t copy props to state if you don’t need to.

componentDidMount

Useful to populate your data *asynchronously* after we’ve shown something to the user.

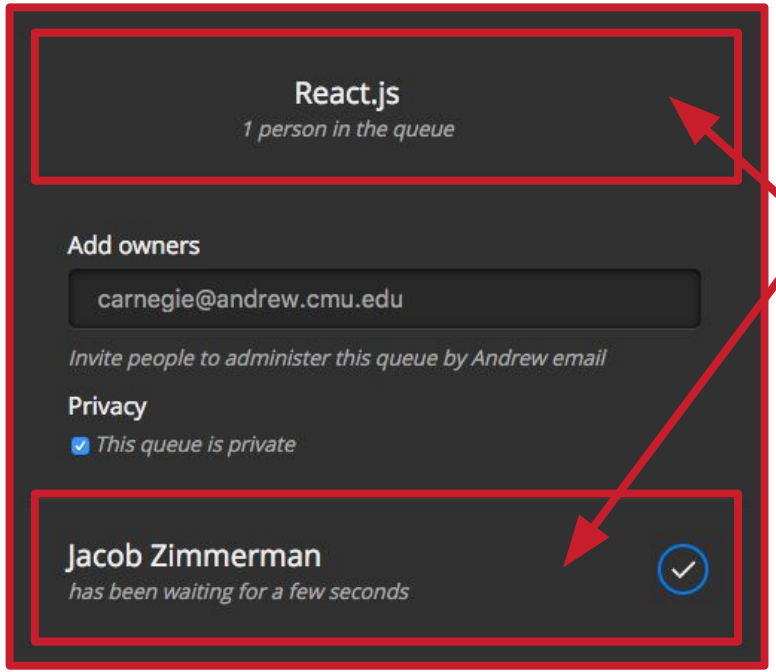
Reminiscent of how most apps load some UI elements, and the data shows up later.

Example: ⌘ + Queue (link)

From Design Mock to Implementation

motivating idea: small, reusable components (actual)

```
<Queue>
...
</Queue>
```




```
<QueueItem />
...
<QueueItem
  action="check-off" />
```

Fetch the data

```
// Uses jQuery.get to handle AJAX calls
componentDidMount: function() {
  return $.get("/api/queues/" + this.props.params.key,
    (queue) => this.setState(queue));
}

// this.setState() will notify React to re-render

// Also, note how this encourages a clear separation
// of concerns: your backend is just returning JSON,
// your frontend takes the JSON and manipulates it.
```

Dive right in!

The React documentation (both by Facebook and by the community) is amazing:

- Facebook's tutorial
- Another great React tutorial

In practice, sometimes the hardest thing to get working with React is the compilation.

References:

- Getting Started
- gulpfile.js

Check these things out later:

- Flux
- ReactDOMServer
- React Native