React

- food for thought:
    - use `localStorage` for Podible listening position, etc? Or use Postgres? Firebase, reThink?
- **probably one of the more important concepts is using ContainerComponents, or the same as saying Sub Components**
    - you have a parent/child relationship
    - the parent manages the `state` and passes the `state` of an object to child components through `props`

Animate Transitions With:

```
import ReactCSSTransitionGroup from 'react-addons-css-transition-group'
```

Should be declaring `PropTypes`
This will help developers know if they are using your components correctly and will catch/throw errors in console in dev mode.
PropTypes Documentation Here

```
App.PropTypes = {
    yourProp: React.PropType.object.isRequired,
    yourPropIndex: React.PropType.int.isRequired
}
```

- read through the lifecycle hooks, eg: `shouldComponentUpdate`, `componentDidUpdate`, etc.
- run `node server.js` to generate webpack bundles automatically into memory and to hot-reload React components. `fab webpack` or `fab webpack_local` to generate bundles(will need to update `webpack.local.config.js` is doing the latter).

Use Stateless Functions:
this is ONLY for stateless functions.
ES6 syntax:

```
const HelloComponent = (props) => <div> Hello, {props.name}</div>;
ReactDOM.render(<HelloComponent name="Sebastian" />, document.querySelector(".root"));
```

React Router:

- React Router is the standard routing library for React Router. From the docs: React Router keeps your UI in sync with the URL. It has a simple API with powerful features like lazy code loading, dynamic route matching, and location transition handling built right
- basically shows/hides components depending if you are on a page.
- Index.JS

Events

- are created inline
- are passed instances of `SyntheticEvent`

Using `this`
- works inside of a class' `render` method, becuase `render` is *bound* to the class.
- when you try to use `this` in a custom method in a class it will not work because that method is **not bound** to the class.

Using function `refs`

- string refs are older
- adds a reference to an object in the class,
- example:

```
render(){
    return(
        <form onSubmit={this.sendEmail}>
            <input className="email" type="text" ref={(input) => {this.emailInput}}
        </form>
    )
}
```

- doing this, will let you reference the object in your class now, but using `this.emailInput` – **THIS WILL NOT WORK IN A CUSTOM METHOD ==** `this`

will be equal to `null`. **YOU NEED TO BIND IT TO THE CLASS** – there are two ways to do this:

- 1) in the `constructor` function:

```
constructor(){
    super();
    this.sendEmail = this.sendEmail.bind(this)
}
```

another way to do it, is bind `this` on a handler event(*note: this is more resource intensive than binding it in the constructor*), example:

```
<form onSubmit={this.sendEmail.bind(this)}> </form>
```

this is basically saying, "allow me to reference `this` in the method `sendEmail`"
another way to do the same thing w/ arrow function, by passing the method the `event`:

```
<form onSubmit={(event) => {this.sendEmail(event)}}>

</form>
```

Working With State

- set the state in the `constructor`

```
this.state = {
    yourObject: {},
    anotherObject: {},
    lastObject: {}
}
```

If you are working with the state in another method, you should **ALWAYS** create a copy of the state to work with then update it via `this.setState`

```
sendEmail(event){
    // create a copy/update our state.
    const formDetails = {...this.state.formDetails}
    formDetail['name'] = "John"
    formDetails['email'] = "john@gmail.com"
    this.setState({formDetails})
}
```

Working With Complex Data

- you should use javascript's `map` method to loop through data.

```
render(){
    const randomData = this.state.data;
    randomData = randomData.map(([index, value]) => ({
    <li key={index}>this.state.data[index].name</li>
    }).bind(this));
    return(
        <ul>{randomDataList}</ul>
    )
}
```

NOTICE HOW THE `map` METHOD MUST BE BINDED TO `this`, if it isn't then `this` will refer to the `window` object

Passing props from Parent Components to Child Components…

- the parent can access properties via `this.propertyName`
- If it is passed to a child component, be sure to access it via `props`, i.e: `this.props.propertyName`

Looping over keys:

```
{
    Object
        .keys(this.state.formDetails)
        .map(key => <FormDetail key={key} details{this.state.form
Details[key]} />)
}
```

…so this will call the `FormDetail` class component which would basically just render the display out to the page:

```
    class FormDetail extends React.Component {
        render(){
            return(
             <li className="form-detail">
                 {this.props.detail.name} - {this.props.detail.em
ail}
             </li>
            )
        }
    }
```

instead of using `this.props.details` you can cut down to just using `detail` by storing it in a `const`.
ES6 destructing:

```
1. const { details } = this.props;
```

normal way

```
1. const details = this.props.details;
```

can't *just* pass variables into methods on event handles
example of INCORRECT way

```
1. <button onClick={this.props.addToCart(someProperty)}>Add To Car
   t</button>
```

example of CORRECT way

```
1. <button onClick={() => this.props.addToCart(someProperty)}>Add To
   Cart</button>
```

in the example above the method `addToCart` is NOT declared in the class with the `<button>`.. it is declared in the PARENT class.

- React lets you create methods in a class and then pass those methods to other components through the use of properties.
- you can pass a method as a prop like:

```
1. <SomeComponent key={key} addToCart={this.addToCart} />
```

and then in the Component you access it like so:

```
1. <button onClick={() => this.props.addToCart(someProperty)}>Add To
   Cart</button>
```