

Introduction To ReactJS

February 17, 2018

Steve Pietrek



**App Dev Practice Manager
Front End Developer
Triangle ReactJS Meetup Organizer**

**@spietrek
spietrek@gmail.com
Cardinal Solutions
Raleigh/Durham**

Presentation References

<https://github.com/spietrek/triangle-reactjs-intro-to-reactjs>

Topics

React Overview

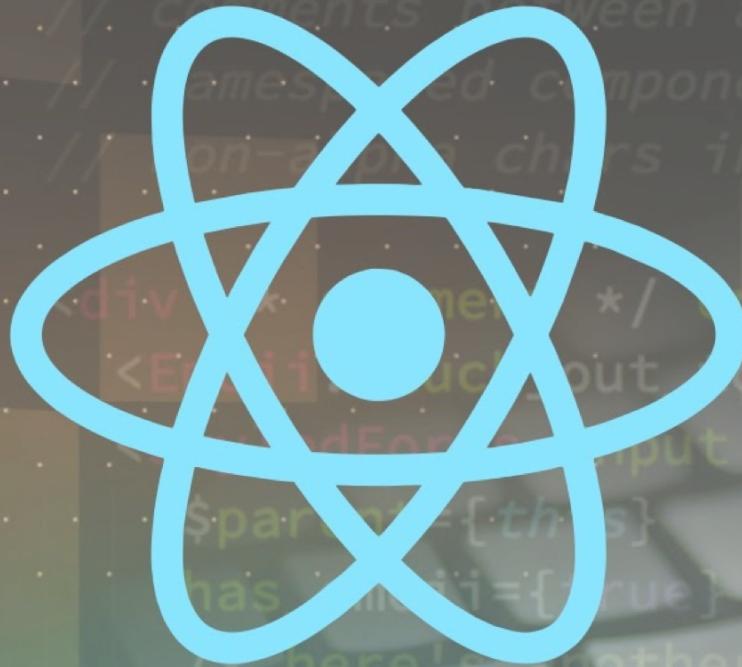
JSX

Create React App

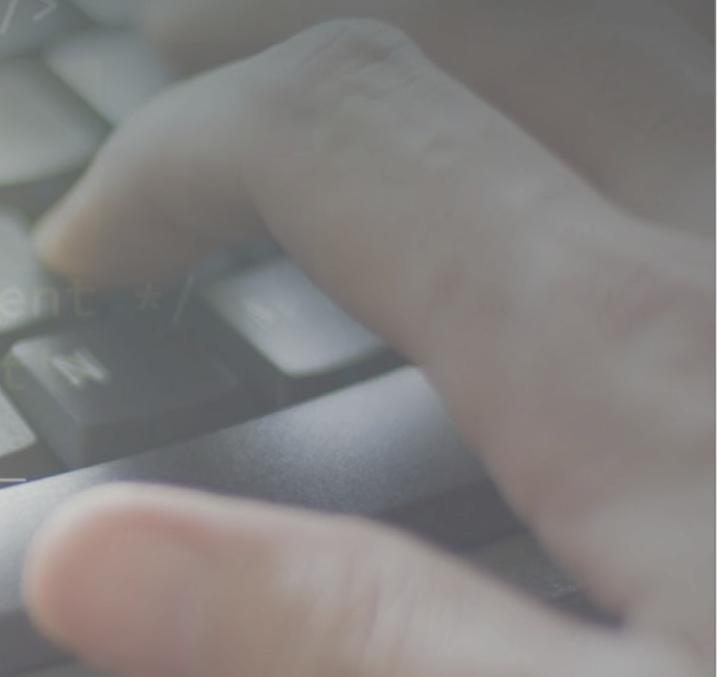
Lifecycle Methods

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

FOCUSES ON
THE “VIEW” IN
MODEL VIEW
CONTROLLER



```
// comments between attributes,  
// namespaced components, and  
// non-ascii chars in tag/attribute name  
  
onClick>  
<div><!-- * --> onClick={this.onClick}  
<EditorInput tongue />  
  
$parent={this}  
hasError={true}  
/> here's another comment */  
className='styled-input'  
</StyledForms.Input>  
</div>
```



React Key Concepts

Virtual DOM

Components

Vanilla
JavaScript

Virtual DOM

Model

Real DOM

Diff

In-memory DOM

Patch

EVERYTHING IN REACT IS A COMPONENT

A screenshot of a React application interface. At the top left is a search bar with the placeholder "Search...". Below it is a checkbox labeled "Only show products in stock". The main area contains a table with two columns: "Name" and "Price". The table is organized into categories: "Sporting Goods" (Football \$49.99, Baseball \$9.99), "Basketball" (\$29.99), and "Electronics" (iPod Touch \$99.99, iPhone 5 \$399.99, Nexus 7 \$199.99). The entire component is styled with a blue border.

Name	Price
Sporting Goods	
Football	\$49.99
Baseball	\$9.99
Basketball \$29.99	
Electronics	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99

1. **FilterableProductTable** (orange): contains the entirety of the example
2. **SearchBar** (blue): receives all user input
3. **ProductTable** (green): displays and filters the data collection based on user input
4. **ProductCategoryRow** (turquoise): displays a heading for each category
5. **ProductRow** (red): displays a row for each product

React Components

```
import React, { Component, Fragment } from 'react';
import formatDateTime from './utils';

class Demo1a extends Component {
  constructor(props) {
    super(props);

    this.state = {
      date: new Date()
    };
  }

  render() {
    const { date } = this.state;
    const dateTime = formatDateTime(date);

    return (
      <Fragment>
        <h1>DEMO 1a</h1>
        <h2>{dateTime}</h2>
      </Fragment>
    );
  }
}

export default Demo1a;
```

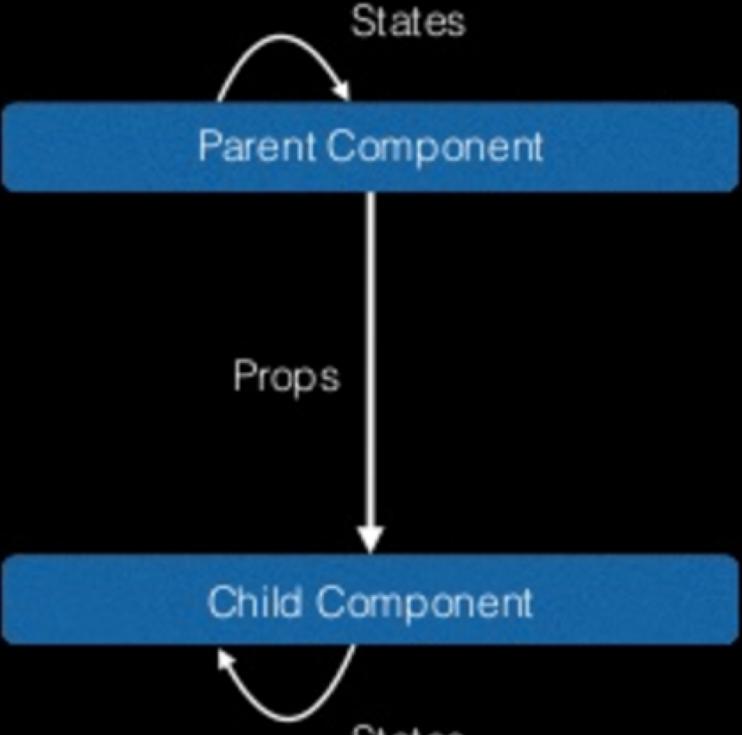
```
import React from 'react';
import PropTypes from 'prop-types';
import formatDateTime from './utils';

const VisualDateTime = props => {
  const { date } = props;
  const dateTime = formatDateTime(date);
  return <h2>{dateTime}</h2>;
};

VisualDateTime.propTypes = {
  date: PropTypes.instanceOf(Date).isRequired
};

export default VisualDateTime;
```

State: Holds information about the component
Props: Information passed from parent component to child component

<p>Data flows down from parent component to child component</p>	<p>Props are read-only (immutable) Props cannot be modified Anything can be passed to children as props (strings, numbers, objects, functions, events)</p>	 <pre> graph TD PC[Parent Component] -- "Props" --> CC[Child Component] CC -- "States" --> PC </pre> <p>The diagram illustrates the data flow between a Parent Component and a Child Component. The Parent Component is represented by a blue rounded rectangle at the top, and the Child Component is below it, also in a blue rounded rectangle. A vertical white arrow points downwards from the Parent Component to the Child Component, labeled 'Props' to its left. Another curved white arrow points upwards from the Child Component back to the Parent Component, labeled 'States' to its right.</p>
<p>Events flow up from child component to parent component</p>	<p>States changes can be asynchronous State can be modified using this.setState Through parent passed event handler props</p>	

One Way Data Flow

Typechecking with PropTypes

PropTypes ensure your props data is valid

A warning will display in the console if an invalid value was provided for a prop

defaultProps is used to set a default prop value when it is not required

TypeScript or Flow can be used to typecheck your **whole** app

PropTypes / defaultProps Example

```
import React from 'react';
import PropTypes from 'prop-types';

const VisualAnalogClock = props => {
  ...
};

VisualAnalogClock.propTypes = {
  date: PropTypes.instanceOf(Date)
};

VisualAnalogClock.defaultProps = {
  date: new Date()
}

export default VisualAnalogClock;
```

JSX

Syntax extension to JavaScript which expresses the Virtual DOM

Declarative description of the UI inline in JavaScript code

Use inside if statements and for loops, assign to variables, accept as arguments, return from functions

Wrap JavaScript expressions in {} to be used within an element

Preprocessor translates syntax into plain JavaScript objects (Babel)

JSX vs. Create Element Example

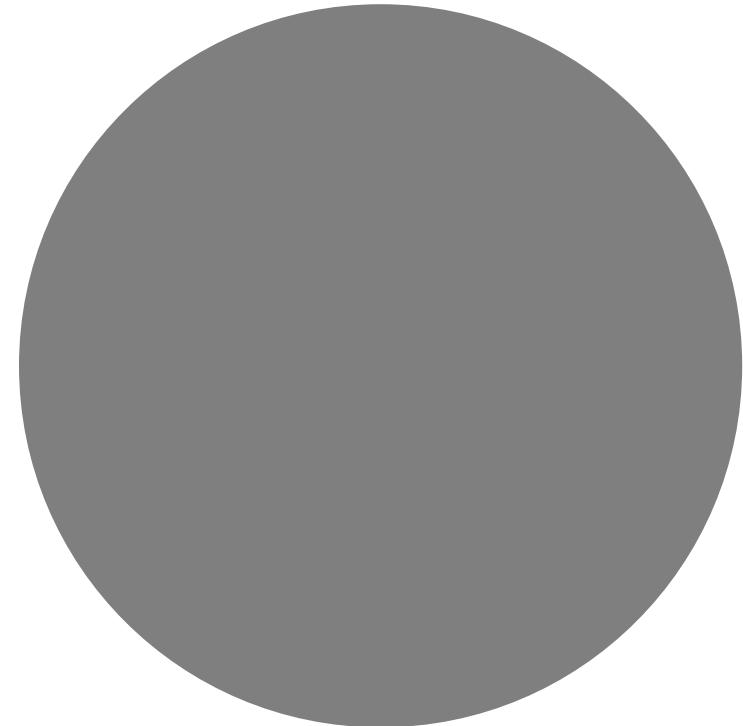
```
class Hello extends React.Component {  
  
  render () {  
  
    return (  
      <div className="container">  
        <h1>Getting Started</h1>  
      </div>  
    );  
  }  
  
  ReactDOM.render(<Hello />, mountNode);
```

```
class Hello extends React.Component {  
  
  render () {  
  
    return (  
      React.createElement("div",  
        { className: "container"},  
        React.createElement("h1", null, "Getting Started")  
      );  
  }  
  
  ReactDOM.render(React.createElement(Hello, null), mountNode);
```

Create React App

- Create React apps with no build configuration
- Every project built with it has a familiar base, reducing the ramp-up time as you and your teammates switch projects
- Quick start (in Terminal/Console):
 - `npm install -g create-react-app`
 - `create-react-app my-app`
 - `cd my-app`
 - `npm start` or `yarn start`
- Uses ES6, Webpack, Babel, and ESLint
- Can be upgraded in-place to get the latest config and tooling
- You can “eject” to a custom setup at any time. No going back.

React Demo #1



Container vs. Presentational Components

- Note: React components do not need to emit DOM
- Container component does data fetching/business logic and renders its corresponding (shares the same name) sub-component
- Benefits:
 - Allow for separation of concerns
 - Allow for reusability of presentational components
 - Allow for better collaboration with designers and other developers

Container vs. Presentational Components

Container Component	Presentational Component
Concerned with how things work	Concerned with how things look
Provide data and behavior to presentational or other container components	Have no dependencies on the rest of the app (e.g. Redux, stores)
Interact with Redux or other state management libraries	Receive data and callbacks only through props
Generally stateful	Only state is UI state (not App state)
Generated using higher order components such as connect() (Redux)	Are generally written as functional components

Anti-Pattern

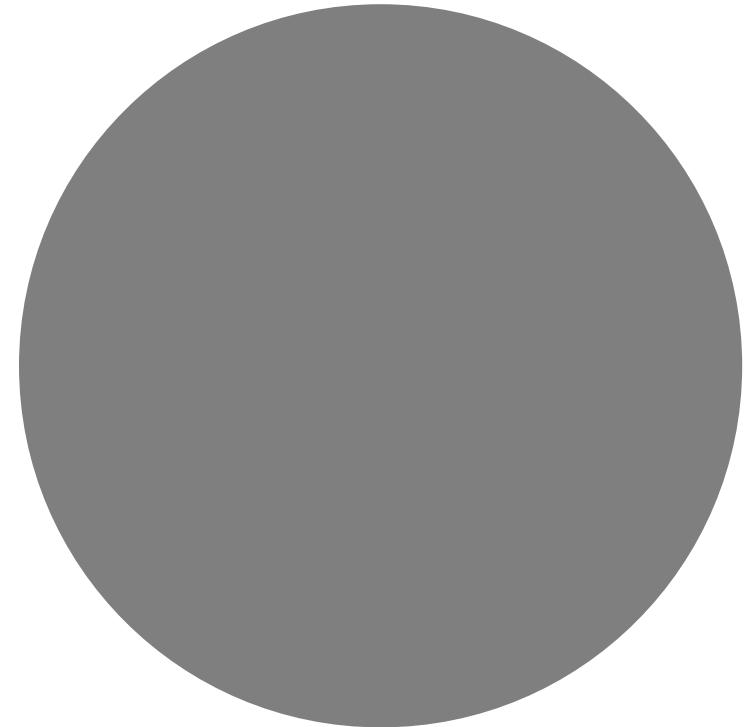
```
// CommentList.js
import React from "react";

class CommentList extends React.Component {
  constructor() {
    super();
    this.state = { comments: [] }
  }

  componentDidMount() {
    fetch("/my-comments.json")
      .then(res => res.json())
      .then(comments => this.setState({ comments }))
  }

  render() {
    return (
      <ul>
        {this.state.comments.map(({ body, author }) =>
          <li>{body}-{author}</li>
        )}
      </ul>
    );
  }
}
```

React Demo #2



Lifecycle Methods

Initialization

setup props and state

Mounting

componentWillMount

render

componentDidMount

Updation

props

componentWillReceiveProps

shouldComponentUpdate

componentWillUpdate

render

componentDidUpdate

states

shouldComponentUpdate

componentWillUpdate

render

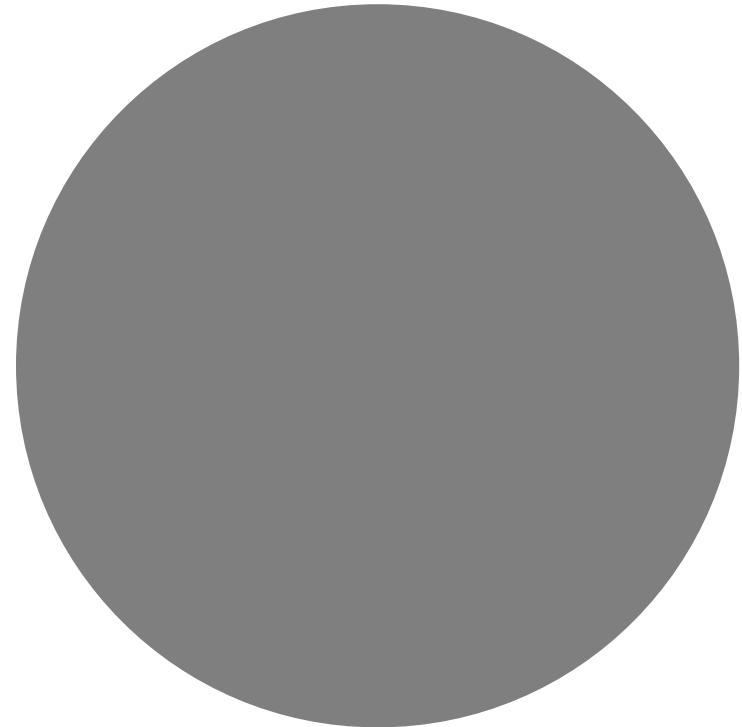
componentDidUpdate

Unmounting

componentWillUnmount

componentDidCatch(error, info)

React Demo #3



Additional things to know

ES6 / ES7 / ES8

NodeJS / NPM / Yarn

Babel / Webpack / Parcel

State Management / Redux / Mobx

ESLint

Prettier

CSS-in-JS / styled-components

TypeScript

Flow

Functional Programming

Reactive Programming