Deep Dive on the React Lifecycle

whoami

Jonathan Creamer

whoami

- Currently Senior Front End Engineer at Lonely Planet
- Past JavaScript Engineer appendTo
- Nashville, TN

- Love JavaScript, tweet at @jcreamer898, blog at jonathancreamer.com
- Microsoft MVP

Agenda

1. What is a React component?
2. Lifecycle methods
3. Use cases
4. Testing

Pure Functions

A function is pure if...

Given the same input, will always return the same output.
Produces no side effects.
Relies on no external mutable state.

- One way to build a React component is as a pure function

Can you even Component bro?

```
function HelloWorld({
  text
}) {
  return (
    <h1>{text}</h1>
  );
}

ReactDOM.render(<HelloWorld text="hello world" />, document.body);
```

- Might be all you need
- Simple

JSX to JS

```
function HelloWorld(_ref) {
  var text = _ref.text;

  return React.createElement(
    "h1",
    null,
    text
  );
}
```

- JSX is an abstraction over creating element trees
- Different renderers like ReactDOM

React Components as Classes

```
class HelloWorld extends React.Component {
  render() {
    const { text } = this.props;
    return (
      <h1>{text}</h1>
    );
  }
}
```

- If all you have is `render`, stay functional
- Don't screw with props, they're Read Only
- Sometimes we need some state

constructor

constructor

```
constructor(prop) {
  super(props);
  this.state = {}:
}
```

- Perform any initial setup
- Called once per mounted component
- Initialize state

State

- What's going on around here right now?
- Has a user clicked or typed anything?
- Any data needing to be fetched?
- Any stored information?

setState

```
class WYSIWYG extends React.Component {
  update() { /*...*/ }
  render() {
    const { text } = this.state;
    return (
      <textarea
        onChange={this.update}
      />
    );
  }
}
```

- Pull from `state`
- A lot of things happen when you do this...

setState

```
constructor(props) {
  super(props);

  this.state = {
    text: "",
  };

  this.update = this.update.bind(this);
}
update(e) {
  const { text } = e.target.value;
  this.setState({ text });
}
```

- Must `bind` for `this` to work
- After `setState`, `render` will fire
- Do NOT update state w/ `this.state.text = "foo";`

componentWillMount

componentWillMount

- `props` and `state` both ready
- Can use for calling `setState`
- Most times just use `constructor`

componentDidMount

componentDidMount

```
import ace from "aceeditor";

export default class Editor extends React.Component {
  constructor() {
    super();
    this.state = { /* init state */ };
  }
  componentDidMount() {
    this.editor = ace.edit(this.$text);
  }
  render() {
    return (
      <div
        ref={(node) => this.$text = node}
      />
    )
  }
}
```

componentDidMount

- jQuery plugin time :trollface:
- DOM is ready here
- Stand up plugins
- `ref` is now a function

componentDidMount

```
import ace from "aceeditor";

export default class Editor extends React.Component {
  constructor() {
    super();
    this.state = {};
  }
  componentDidMount() {
    this.editor = ace.edit(this.$text);
  }
  render() {
    return (
      <div
        ref={(node) => this.$text = node}
      />
    )
  }
}
```

componentDidMount

```
import ace from "aceeditor";

export default class Editor extends React.Component {
  constructor() {
    super();
    this.state = {};
  }
  componentDidMount() {
    this.editor = ace.edit(this.$text);
  }
  render() {
    return (
      <div
        ref={(node) => this.$text = node}
      />
    )
  }
}
```

componentDidMount

```
import ace from "aceeditor";

export default class Editor extends React.Component {
  constructor() {
    super();
    this.state = {};
  }
  componentDidMount() {
    this.editor = ace.edit(this.$text);
  }
  render() {
    return (
      <div
        ref={(node) => this.$text = node}
      />
    )
  }
}
```

componentWillUnmount

Be a good citizen

```
componentWillUnmount() {
  this.editor.destroy();
}
```

- Remove any event handlers or plugins
- No leaks

componentWillUnmount

```
class Chat extends Component {
  constructor(props) {
    super(props);
    this.state = { messages: [] };
  }
  componentDidMount() {
    this.subscription = postal.subscribe({
      topic: "message.added",
      callback: (message) => {
        this.setState({
          messages: [...this.state.messages, message]
        })
      }
    });
  }
  componentWillUmount() {
    this.subscription.unsubscribe();
  }
  render() {
    return (
      <Messages messages={this.state.message} />
    )
  }
}
```

componentWillUnmount

```
class Chat extends Component {
  constructor(props) {
    super(props);
    this.state = { messages: [] };
  }
  componentDidMount() {
    this.subscription = postal.subscribe({
      topic: "message.added",
      callback: (message) => {
        this.setState({
          messages: [...this.state.messages, message]
        })
      }
    });
  }
  componentWillUmount() {
    this.subscription.unsubscribe();
  }
  render() {
    return (
      <Messages messages={this.state.message} />
    )
  }
}
```

componentWillUnmount

```
class Chat extends Component {
  constructor(props) {
    super(props);
    this.state = { messages: [] };
  }
  componentDidMount() {
    this.subscription = postal.subscribe({
      topic: "message.added",
      callback: (message) => {
        this.setState({
          messages: [...this.state.messages, message]
        })
      }
    });
  }
  componentWillUmount() {
    this.subscription.unsubscribe();
  }
  render() {
    return (
      <Messages messages={this.state.message} />
    )
  }
}
```

componentWillUnmount

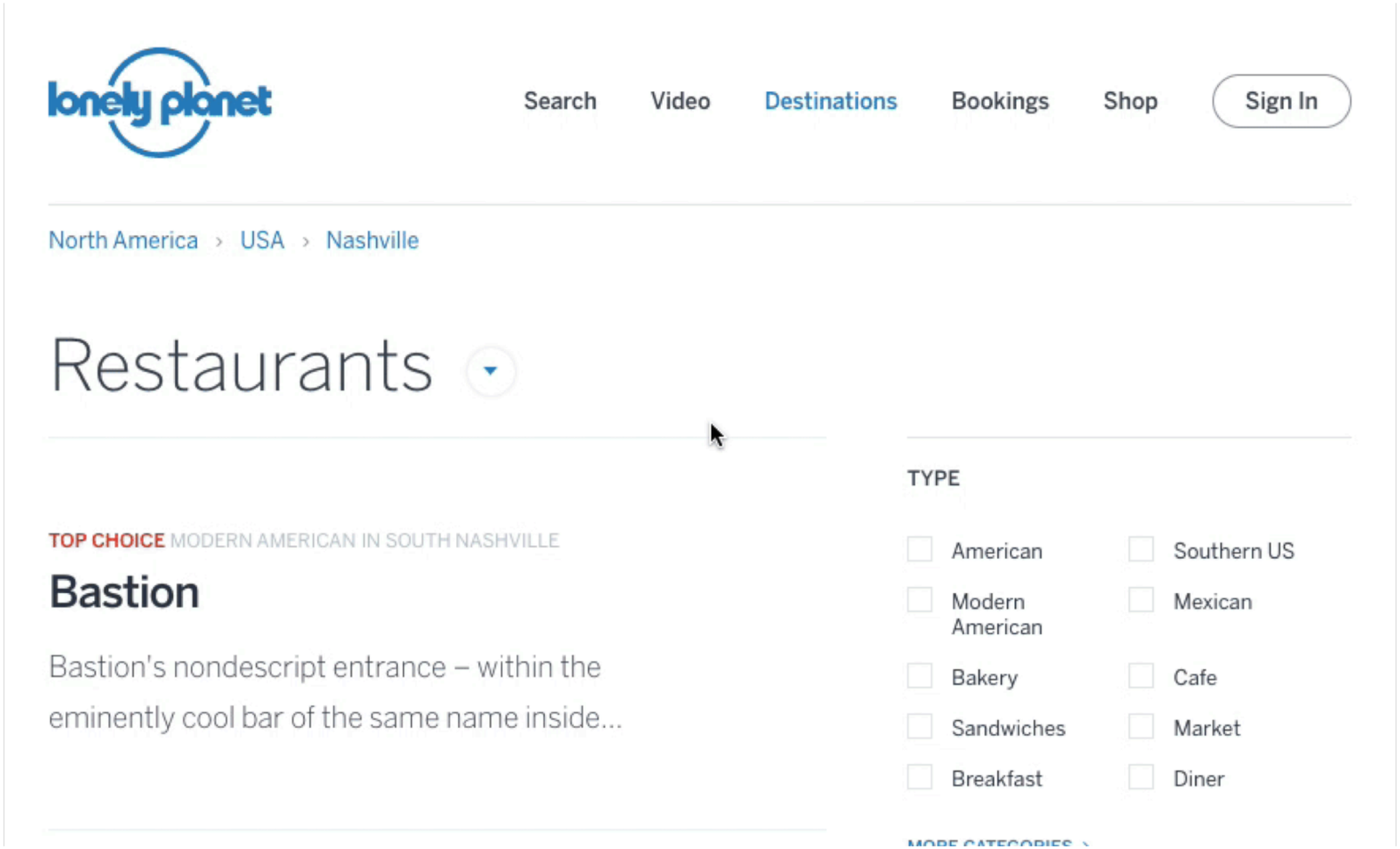
```
class Chat extends Component {
  constructor(props) {
    super(props);
    this.state = { messages: [] };
  }
  componentDidMount() {
    this.subscription = postal.subscribe({
      topic: "message.added",
      callback: (message) => {
        this.setState({
          messages: [...this.state.messages, message]
        })
      }
    });
  }
  componentWillUmount() {
    this.subscription.unsubscribe();
  }
  render() {
    return (
      <Messages messages={this.state.message} />
    )
  }
}
```

componentWillUnmount

```
class Chat extends Component {
  constructor(props) {
    super(props);
    this.state = { messages: [] };
  }
  componentDidMount() {
    this.subscription = postal.subscribe({
      topic: "message.added",
      callback: (message) => {
        this.setState({
          messages: [...this.state.messages, message]
        })
      }
    });
  }
  componentWillUmount() {
    this.subscription.unsubscribe();
  }
  render() {
    return (
      <Messages messages={this.state.message} />
    )
  }
}
```

IRL

IRL


lonely planet
Search
Video
Destinations
Bookings
Shop
Sign In
North America › USA › Nashville
Restaurants
TOP CHOICE MODERN AMERICAN IN SOUTH NASHVILLE
Bastion
Bastion's nondescript entrance – within the eminently cool bar of the same name inside...
TYPE
American
Southern US
Modern American
Mexican
Bakery
Cafe
Sandwiches
Market
Breakfast
Diner

componentDidMount IRL

```
export default class PoiDetail {
  componentDidMount() {
    if (!this.state.poi) {
      this.fetchPoi(this.props.params.id);
    }

    if (!this.state.related) {
      this.fetchRelated(this.props.params.id);
    }
  }
}
```

- `List` is mounted on load, `Poi` gets mounted on route change
- Use `react-router` params to get ID

componentDidMount IRL

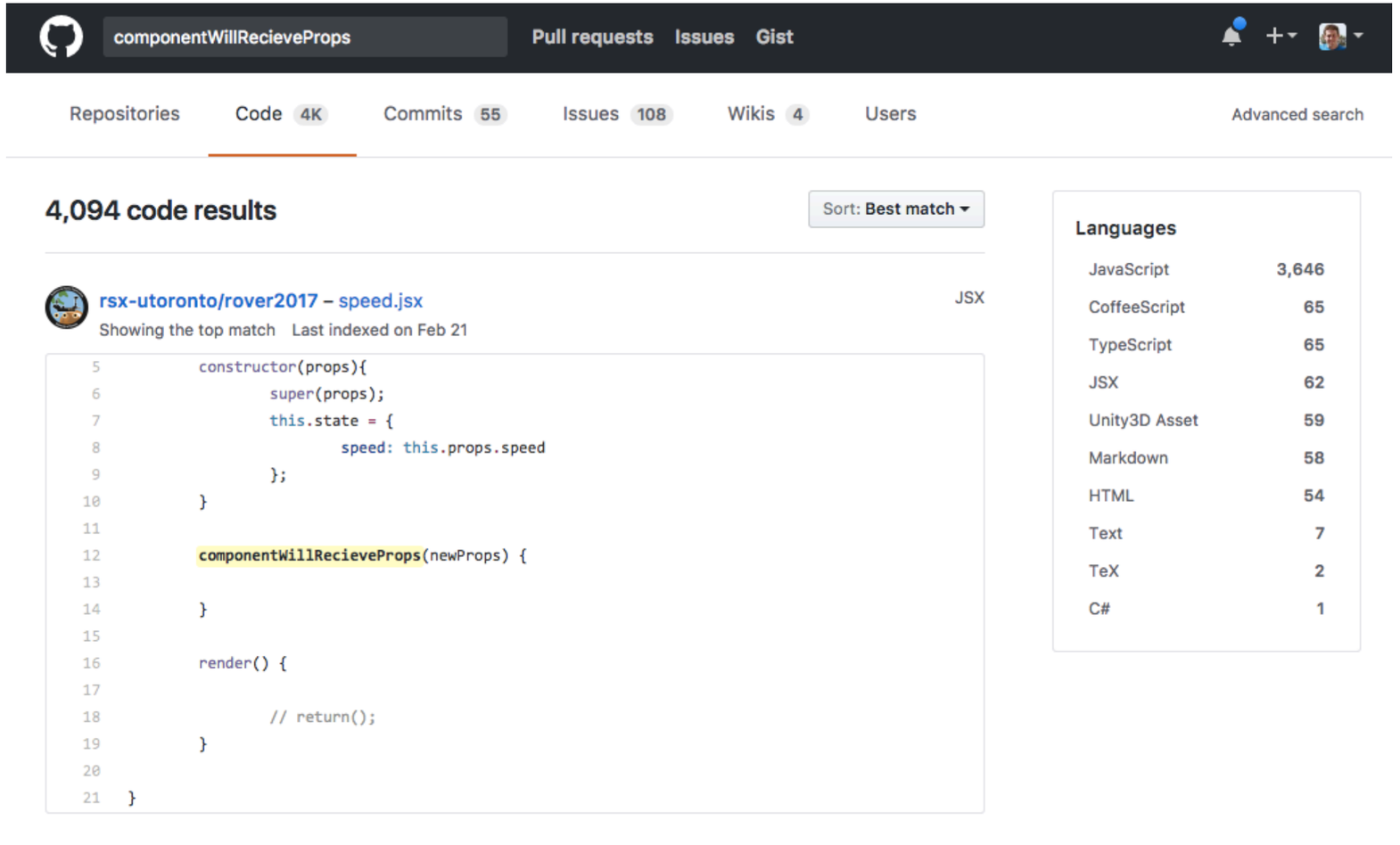
```
import { connect } from "react-redux";
import { bindActionCreators } from "redux";

const mapStateToProps = state => ({
  poi: state.poi,
  related: state.related
});
const mapDispatchToProps = (dispatch) => ({
  ...bindActionCreators(actions, dispatch)
});
const connected = connect(mapStateToProps, mapDispatchToProps)(Poi);
export { connected };
```

- Setup the state, and props

componentWillReceiveProps

I before E...


componentWillRecieveProps
Pull requests
Issues
Gist
Repositories
Code 4K
Commits 55
Issues 108
Wikis 4
Users
Advanced search
4,094 code results
Sort: Best match
rsx-utoronto/rover2017 – speed.jsx
JSX
Showing the top match Last indexed on Feb 21
5 constructor(props){
6 super(props);
7 this.state = {
8 speed: this.props.speed
9 };
10 }
11
12 componentWillRecieveProps(newProps) {
13
14 }
15
16 render() {
17
18 // return();
19 }
20
21 }
Languages
JavaScript 3,646
CoffeeScript 65
TypeScript 65
JSX 62
Unity3D Asset 59
Markdown 58
HTML 54
Text 7
TeX 2
C# 1

componentWillReceiveProps

- Called when the props passed in change
- NOT called on initial render
- `props` have changed
- Be careful, can cause loops i.e. don't compare objects
- Update state if props don't match
- Very useful in React Router SPA

componentWillReceiveProps

```
class AvatarUploader extends Component {
  constructor(props) {
    super(props);
    this.state = { src: props.src };
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.src !== this.props.src) {
      this.setState({
        src: nextProps.src,
      });
    }
  }
  uploadFiles() {
    this.props.upload()
  }
  render() { /* ... */ }
}
```

- Useful when `state` is set with a `prop`.
- Anti-pattern alert!

componentWillReceiveProps

```
class Editor extends PureComponent {
  upload(files) {
    this.props.dispatch(uploadAction(files));
  }
  render() {
    const { image } = this.props;
    return (
      <AvatarUploader
        src={image}
        upload={this.upload}
      />
    )
  }
}
```

```
const mapStateToProps = (state) => ({ image: state.image })
export default connect(mapStateToProps)(Editor);
```

- `image` gets passed in to `AvatarUploader`
- Causes `componentWillReceiveProps` to fire

React Router

```
class SightComponent extends Component {
  render() {
    return (
      <div>
        <Link to="/a/poi-sig/381139/362228">Country Music Hall of Fame</Link>
      </div>
    )
  }
}
```

- Country Music Hall of Fame
- If a component is mounted already

React Router

```
componentWillReceiveProps(nextProps) {
  const { id: currentId } = this.props.params;
  const { id: nextId } = nextProps.params;

  if (currentId !== nextId) {
    this.props.fetchPoi(nextId);
  }
}
```

- Compare last id to current id
- Name variables for readability

componentDidUpdate

componentDidUpdate

```
componentDidUpdate(prevProps) {
  if (prevProps.params.id !== this.params.id) {
    window.scrollTop = 0;
    this.props.pageView();
  }
}
```

- Gives you previous props `prevProps` to compare
- Called AFTER render
- Update the DOM

shouldComponentUpdate

shouldComponentUpdate

```
shouldComponentUpdate(nextProps, nextState) {
  return shallowCompare(nextProps, nextState);
}


export class Counter extends React.PureComponent {
  // ...
}
```

- Tell the component whether or not to render
- Can increase performance
- Used to call `shallowCompare`
- MOST of the time, use `React.PureComponent`

shouldComponentUpdate

```
shouldComponentUpdate(nextProps) {
  return nextProps.poi.id !== this.props.poi.id;
}
```

- If you need fine grain control
- Parent could change a prop
- Return false will cancel...
- `componentWillUpdate`, `render`, and `componentDidUpdate`

componentWillUpdate

componentWillUpdate

- Called every re-render like when `setState` called
- Do NOT call `setState` here
- Useful for triggering CSS animations or transitions

Full Example

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

React Router

```
render() {
  return (
    <Link to={poiLink(1234, 362228)} />
  )
}
```

- Click a link
- Will pass props down to `PoiDetail`

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```


Fetching...

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

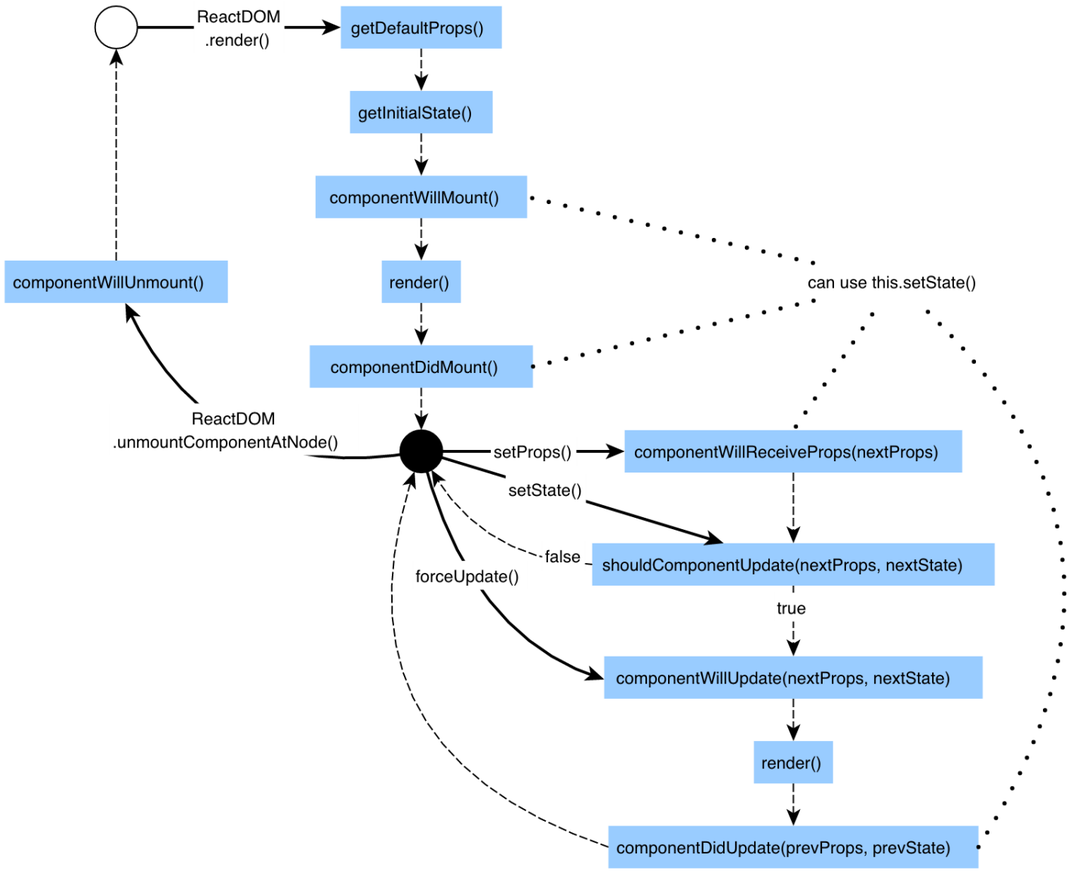
```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

tldr;

Name of thing	Sorta like...	Mounted?	Can you even setState?	What would you say... ya do here?
constructor	initialize()	nope	nope	init stuff NO side effects
componentWillMount	beforeDomReady()	nope	yeah but don't	Only needed in createClass now use constructor for most things
render	render	nope	please no	render stuff and don't set any state please
componentDidMount	domReady()	yup	yup	DOM is a go init jQuery plugins dispatch stuff
componentWillReceiveProps	onChange()	yup	yup	Props changed feel free to update state if needed
componentWillUpdate	beforeRender()	yup	nope	The props or state changed need to do anything else before rendering?
shouldComponentUpdate	shouldRender()	yup	nope	So yeah something changed but do we REALLY need to update?
componentDidUpdate	afterRender()	yup	yup	Great success we've rendered a thing... anything else?
componentWillUnmount	destroy()	too late	too late	Only you can prevent memory leaks aka unbind things

```
/**
 * ------------------ The Life-Cycle of a Composite Component ------------------
 *
 * - constructor: Initialization of state. The instance is now retained.
 *   - componentWillMount
 *   - render
 *   - [children's constructors]
 *     - [children's componentWillMount and render]
 *     - [children's componentDidMount]
 *     - componentDidMount
 *
 *       Update Phases:
 *       - componentWillReceiveProps (only called if parent updated)
 *       - shouldComponentUpdate
 *         - componentWillUpdate
 *           - render
 *           - [children's constructors or receive props phases]
 *         - componentDidUpdate
 *
 *     - componentWillUnmount
 *     - [children's componentWillUnmount]
 *   - [children destroyed]
 * - (destroyed): The instance is now blank, released by React and ready for GC.
 *
 * -----------------------------------------------------------------------------
 */
```



- https://tylermcginnis.com/an-introduction-to-life-cycle-events-in-react-js/

Testing

Testing

```
npm i install -S jest enzyme
```

- Jest for suite
- Enzyme for sweet
- Super important to test
- Always test logic
- Use mount to run JSDom

Testing willMount

```
describe("Detail Page", () => {
  it("should fetch a page if there isn't one loaded", () => {
    const fetch = jest.fn();
    const wrapper = mount(
      <Details
        poi={null}
        fetchPoi={fetch}
      />
    );

    expect(fetch).toHaveBeenCalled();
  });
});
```

Testing willReceiveProps

```
describe("Detail Page", () => {
  it("should fetch a new page", () => {
    const fetch = jest.fn();
    const wrapper = mount(
      <Details
        poi={{ id: 1}}
        params={{
          id: 1
        }}
        fetchPoi={fetch}
      />
    );
    wrapper.setProps({
      params: { id: 2 }
    });

    expect(fetch).toHaveBeenCalled();
  });
});
```

- Use `setProps`

Additional Resources

- https://gist.github.com/jcreamer898/aeaf4b7a08b9871c3a48ad4bb7ccb35c
- https://engineering.musefind.com/react-lifecycle-methods-how-and-when-to-use-them-2111a1b692b1
- https://developmentarc.gitbooks.io/react-indepth/content/life*cycle/lifecycle*methods_overview.html
- http://busypeoples.github.io/post/react-component-lifecycle/

Thanks!

@jcreamer898

jonathancreamer.com