React and Redux: Lesson 3

Routing, Refs, Forms, Performance Optimization, Higher Order Components, React DevTools

William Abboud

* Covers React v15 - 16

Photo by Bruno Bergher on Unsplash

React Router v4

Repo:

https://github.com/ReactTraining/react-router/tree/master/packages/react-router-dom

Docs: https://reacttraining.com/react-router/web/guides/philosophy

React router offers declarative dynamic routing and stands out from conventional based static routing by letting you define your your app routes as it renders.

React Router v4 - Example

```
function App() {
  return (
    <Router>
      <div className="app">
        <SiteHeader />

        <Route path="/" component={Home} exact />
        <Route path="/rules" component={Rules} />
        <Route path="/ranking" component={Ranking} />
        <Route path="/login" component={Login} />
        <Route path="/register" component={Register} />
        <Route path="/game" component={Game} />
      </div>
    </Router>
  );
}

export default App;
```

Wrapp app in Router Component

Specify routes with Route component

Define path for the component to match

Render component on path match

React Router v4 - Example

```
function Header(props) {
  return (
    <header className="site-header">
      <nav className="main-nav">
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to="/books">Books</Link></li>
        </ul>
      </nav>

      <nav className="account-nav">
        <ul>
          <li><Link to="/register">Register</Link></li>
        </ul>
      </nav>
    </header>
  );
}

export default Header;
```

Link component acts a an anchor

Default path

Specific path

React Forms

Docs: https://reactjs.org/docs/forms.html

React handles forms a bit differently than other HTML elements.

There are 2 types of form components: Controlled and Uncontrolled.

Controlled Forms

Docs: https://reactjs.org/docs/forms.html

Controlled Form components are such components where the state of the form is controlled in the react component rather than the DOM itself.

```
    this.state = {
      firstName: "",
      lastName: "",
      age: ""
    };
  }

  handleChange({ target }) {
    this.setState({ [target.id]: target.value });
  }

  render() {
    const { firstName, lastName, age } = this.state;

    return (
      <section className="register-view">
        <form>
          <div>
            <label htmlFor="firstName">First Name:</label>
            <input type="text" id="firstName" value={firstName} onChange={this.handleChange} />
          </div>

          <div>
            <label htmlFor="lastName">Last Name:</label>
            <input type="text" id="firstName" value={lastName} onChange={this.handleChange} />
          </div>

          <div>
            <label htmlFor="age">Age:</label>
            <input type="number" id="age" value={age} onChange={this.handleChange} />
          </div>
        </form>
```

Uncontrolled Forms

Docs: https://reactjs.org/docs/uncontrolled-components.html

Uncontrolled Form components are such components where the state of the form is controlled in the DOM itself. Meaning React has nothing to do with how the values are changed and saved everything is handled natively by the DOM.

```
    return (
      <section className="register-view">
        <form>
          <div>
            <label htmlFor="firstName">First Name:</label>
            <input type="text" id="firstName" />
          </div>

          <div>
            <label htmlFor="lastName">Last Name:</label>
            <input type="text" id="firstName" />
          </div>

          <div>
            <label htmlFor="age">Age:</label>
            <input type="number" id="age" />
          </div>
        </form>
      </section>
    );
  }
}
```

Refs

Docs: https://reactjs.org/docs/refs-and-the-dom.html

Refs is a feature of React which lets you access the underlying DOM element your React element or component corresponds to.

To use refs just define the “ref” prop on a react element or component and for value pass in a function which will take the DOM element as its first parameter.

Refs are only used on class components and not functional components.

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

    this.saveNameRef = this.saveNameRef.bind(this);
  }

  saveNameRef(inputEl) {
    this.nameInput = inputEl;
  }

  render() {
    return (
      <form>
        <input ref={this.saveNameRef} id="name">
      </form>
    );
  }
}
```

Refs Continued…

Refs are set when a component is mounted and unmounted.

A common gotcha especially when you are testing React components which rely on refs is that when a component is unmounted then refs will be set to null so it's important to always do a null check.

Always access a ref dependant value in componentDidMount.

Performance

1. Do not optimize prematurely
2. Identify performance issues further up the hierarchy
3. Install React DevTools to monitor wasted renders
4. Use Chrome DevTools Performance Tab to view slow functions
5. Extend React.PureComponent
6. Implement your own version of shouldComponentUpdate

shouldComponentUpdate

The lifecycle method shouldComponentUpdate is your first stop when you try to fix your react component's wasted render performance.

shouldComponentUpdate returns a boolean value and is responsible for any consecutive rerenders.

It can be very difficult to nail it right but it does offer a lot of benefits.

shouldComponentUpdate Continued...

- Do not compare children elements
- Only compare props and state simple values or arrays of simple values
- Make sure you get it right

High Order Components (HOC)

Docs: https://reactjs.org/docs/higher-order-components.html

High Order Components is an emerging pattern in React which is used to reuse component logic.

In essence a high order component is just a function that takes a component and returns a component.

It is useful when you have to give more than one component additional information.

```
import React from 'react';

export function withMediaQuery(Component) {
  return class extends React.Component {
    constructor(props) {
      super(props);

      this.state = {
        viewport: ""
      };
    }

    render() {
      return <Component {...this.state} />
    }
  };
}
```

Return wrapped up class

Return the passed component in the render

Pass additional information