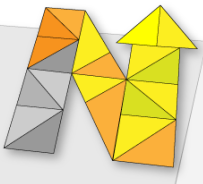

NextStep IT Training

powered by Smallrock Internet

React and Redux Overview

NS60609

React and Redux Overview

COPYRIGHT

Copyright 2019 by NextStep IT Training (Publisher), a division of Smallrock Internet Services, Inc. All rights reserved. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.¡/p¿

All trademarks are trademarks of their respective owners. Rather than put a trademark symbol after every occurrence of a trademarked name, we use names in an editorial fashion only, and to the benefit of the trademark owner, with no intention of infringement of the trademark. Where such designations appear in this book, they have been printed with initial caps.

Information has been obtained by Publisher from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, Publisher, or others, Publisher does not guarantee to the accuracy, adequacy, or completeness of any information included in this work and is not responsible for any errors or omissions or the results obtained from the use of such information.

Node.js is a registered trademark of Joyent, Inc. and/or its affiliates. JavaScript is a registered trademark of Oracle America, Inc and/or its affiliates. TypeScript is a registered trademark of Microsoft Corporation and/or its affiliates. Webpack is a registered trademark of JSFoundation, Inc. and/or its affiliates. All other trademarks are the property of their respective owners, and NextStep IT Training makes no claim of ownership by the mention of products that contain these marks.

TERMS OF USE

This is a copyrighted work and NextStep IT Training and its licensors reserve all rights in and to the work. Use of this work is subject to these terms. Except as permitted under the Copyright Act of 1976 and the right to store and retrieve one copy of the work, you may not decompile, disassemble, reverse engineer, reproduce, modify, create derivative works based upon, transmit, distribute, disseminate, sell, publish or sublicense the work or any part of it without NextStep IT Training's prior consent. You may use the work for your own noncommercial and personal use; any other use of the work is strictly prohibited. Your right to use the work may be terminated if you fail to comply with these terms.

THE WORK IS PROVIDED "AS IS." NEXTSTEP IT TRAINING AND ITS LICENSORS MAKE NO GUARANTEES OR WARRANTIES AS TO THE ACCURACY, ADEQUACY OR COMPLETENESS OF OR RESULTS TO BE OBTAINED FROM USING THE WORK, INCLUDING ANY INFORMATION THAT CAN BE ACCESSED THROUGH THE WORK VIA HYPERLINK OR OTHERWISE, AND EXPRESSLY DISCLAIM ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. NextStep IT Training and its licensors do not warrant or guarantee that the functions contained in the work will meet your requirements or that its operation will be uninterrupted or error free. Neither NextStep IT Training nor its licensors shall be liable to you or anyone else for any inaccuracy, error or omission, regardless of cause, in the work or for any damages resulting therefrom. NextStep IT Training has no responsibility for the content of any information accessed through the work. Under no circumstances shall NextStep IT Training and/ or its licensors be liable for any indirect, incidental, special, punitive, consequential or similar damages that result from the use of or inability to use the work, even if any of them has been advised of the possibility of such damages. This limitation of liability shall apply to any claim or cause whatsoever whether such claim or cause arises in contract, tort or otherwise.

999 Ponce de Leon Boulevard, Suite 830
Coral Gables, FL 33134
786-347-3155 / 800-418-0811

Contents

Chapter 1
React and Redux Overview

Elements and Components
Props, State, and Routes
Redux

Objectives

- Introduce the React environment, elements, and components
- Use props to move data into components, and make them stateful
- Explore Redux for global state

Overview

In 2018 React surpassed Angular as THE most popular tool for building user interfaces for the browser. In opposition to the template-based solution provided by Angular and other frameworks, React offers a purely programmatic solution to creating a browser-based user interface. And, because of its virtual DOM, React can be a much faster and robust solution. React programs can be written with either JavaScript, or even better, Microsoft TypeScript, and if you prefer to work with C# or Java you will see the similarities in this strongly-typed language. If you are a coder and like the idea of an application-driven interface, then this code-based overview is the jump-start you need!

1.1 Elements and Components

Elements and Components
Props, State, and Routes
Redux

1.1.1 Bootstrap

```
    <div id="root"></div>
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './app/App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App />, document.getElementById('root'));
```

- Entry point is public/index.html
- Loads src/index.js, which renders onto the div with id "root"

1.1.2 createElement

```
// ReactDOM.render(createElement(App), document.getElementById('root'));

ReactDOM.render(<App />, document.getElementById('root'));
```

- The key is *createElement* which renders elements and components
- JSX is prefered to simplify the syntax and readability
- JSX is XML, it isn't an element, it translates into code that creates an element

1.1.3 Elements and Components

```
import React, { Component } from 'react';

class App extends Component {
    render() {

        const now = new Date()
        let message = null

        if (now.getHours() < 12) {

            message = 'Good morning! Welcome to the Caribbean Coffee Company!'

        } else if (now.getHours() < 17) {
```

```
            message = 'Good afternoon! Welcome to the Caribbean Coffee Company!'

        } else {

            message = 'Welcome to the Caribbean Coffee Company! Sorry, we are
                ↪ closed.'
        }

        return (
            <div>
                <span>{message}</span>
            </div>
        );
    }
}

export default App;
```

- Elements are leaf nodes, nothing smaller, and map to HTML
- Components are complex structures that *render* elements and/or other components
- Components may be built from functions or classes

1.1.4 Structure

- The default structure is flat
- Use a structure to organize the application by feature
- Move App into *app*, move App code into *landing/Landing*

1.2 Props, State, and Routes

Elements and Components
Props, State, and Routes
Redux

```
render() {

    return (
        <header className="app-header">
            <img id="app-logo" src={logo} alt="logo" />
        </header>
    )
}
```

```
render() {

    return (
        <footer className="app-footer">
            Copyright &copy; NextStep IT Training. All rights reserved.
        </footer>
    )
}
```

- Add the *common/header* and *common/footer* views and link them into the App

1.2.1 Props

```
    constructor(props) {

        super(props)

        this.beverages = [
            new Product( { id: 1, name: 'Juan Valdez Reserve Cafe - Small',
                ↪ price: 1.85  } ),
            new Product( { id: 2, name: 'Cappuccino - Small', price: 2.65 } ),
            new Product( { id: 3, name: 'Cafe Latte', price: 2.95 } )
        ]
    }

    render() {

        return (
            <div className="app-content">
                <ProductList title="Beverages" products={ this.beverages } />
                <ProductList title="Pastries" products={ this.pastries } />
            </div>
        )
    }
```

```
class ProductList extends Component {

    static get propTypes() {

        return {

            title: PropTypes.string.isRequired,
            products: PropTypes.arrayOf(PropTypes.instanceOf(Product)).
                ↪ isRequired
        }
    }

    render() {

        let productItems = this.props.products.map( (product) => <ProductItem
            ↪ key={ product.id } product={ product } /> )

        return (
            <div className="list">
                <div className="list-title">{ this.props.title }</div>
                <table className="list">
                    <thead>
                        <tr>
                            <th className="list-name"></th>
                            <th className="list-price">price</th>
                        </tr>
                    </thead>
                    <tbody>
                        { productItems }
                    </tbody>
                </table>
            </div>
        )
    }
}
```

```

```

- Components receive data as "props"
- Props may be string values, or any type of JavaScript object
- Add the *Menu/ProductList* and *Menu/ProductItem* views that display the table of products

1.2.2 State

```

class Checkout extends Component {

    constructor(props) {

        super(props)

        this.state = {

            name: '',
```

```
            cardNumber: ''
        }
    }

    render() {

        return (
            <div>
                <h1>Checkout</h1>
                <form>
                    <div className="cc-form">
                        <div className="cc-form-row">
                            <div className="cc-form-label"><label className="
                                ↪ cc-form-label">Name:</label></div>
                            <div className="cc-form-field"><input type="text"
                                ↪ className={ 'cc-form-field ' + ( /^\w+ \w+$/.
                                ↪ test(this.state.name) ? '
                                ↪ cc-form-field-requirement-met' : '
                                ↪ cc-form-field-required' ) } value={ this.
                                ↪ state.name } onChange={ event => this.
                                ↪ setState({ name: event.target.value }) } /></
                                ↪ div>
                            <div className="clear-all"></div>
                        </div>
                        <div className="cc-form-row">
                            <div className="cc-form-label"><label className="
                                ↪ cc-form-label">Card Number:</label></div>
                            <div className="cc-form-field"><input type="text"
                                ↪ className={ 'cc-form-field ' + ( /\d{13,}$/.
                                ↪ test(this.state.cardNumber) ? '
                                ↪ cc-form-field-requirement-met' : '
                                ↪ cc-form-field-required' ) } value={ this.
                                ↪ state.cardNumber } onChange={ event => this.
                                ↪ setState({ cardNumber: event.target.value })
                                ↪ } /></div>
                            <div className="clear-all"></div>
                        </div>
                        <div className="cc-form-row">
                            <div className="cc-form-label"></div>
                            <div className="cc-form-field">
                                <button>Cancel</button> 
                                <button>Submit</button>
                            </div>
                            <div className="clear-all"></div>
                        </div>
                    </div>
                </form>
            </div>
        )
    }
}
```

- State is internal data for a Component
- *setState* updates component state and calls render
- Add the *checkout/Checkout* view and change *App* to render it

1.2.3 Routes

```
import React, { Component } from 'react'
import { Route, Switch } from 'react-router'
import { BrowserRouter as Router } from 'react-router-dom'

import Checkout from '../checkout/Checkout'
import Footer from '../common/Footer'
import Header from '../common/Header'
import Landing from '../landing/Landing'
import Menu from '../menu/Menu'
import Navigation from './Navigation'

class App extends Component {

    render() {

        return (
            <Router>
                <div>
                    <Header />
                    <Navigation />
                    <Switch>
                        <Route path="/menu" component={ Menu } /> } />
                        <Route path="/checkout" component={ Checkout } /> } />
                        <Route path="/" exact={ true } component={ Landing } />
                    </Switch>
                    <Footer />
                </div>
            </Router>
        )
    }
}
```

```
import React, { Component } from 'react'
import { withRouter } from 'react-router'

class Navigation extends Component {

    render() {

        return (
            <div className="navigation">
                <button className={ `${ this.props.location.pathname === '/' ? '
                    ↪ navbutton-selected' : 'navbutton' }` }
                    onClick={ event => this.props.history.push('/') }>Home</
                        ↪ button>
                <button className={ `${ this.props.location.pathname === '/menu'
                    ↪ ? 'navbutton-selected' : 'navbutton' }` }
                    onClick={ event => this.props.history.push('/menu') }>Menu</
                        ↪ button>
                <button className={ `${ this.props.location.pathname === '/
                    ↪ checkout' ? 'navbutton-selected' : 'navbutton' }` }
                    onClick={ event => this.props.history.push('/checkout') }>
                        ↪ Checkout</button>
            </div>
        )
    }
}
```

```
export default withRouter(Navigation)
```

- The Router establishes *router*, *history*, and *location* objects
- The *Switch* and *Route* objects define where to go
- The *withRouter* injects the history, location into a component

1.3 Redux

Elements and Components
Props, State, and Routes
Redux

1.3.1 Service/Controller

```
import { createModelAction } from '../model/ModelAction'
import dataContext from './dataContext'
import ProductActionType from './ProductActionType'

class ProductActionController {

    constructor(dispatch) {

        this.dispatch = dispatch
    }

    async getBeverages() {

        try {

            const beverages = await dataContext.beverageContext.getBeverages()

            this.dispatch(createModelAction(ProductActionType.
                ↪ SET_BEVERAGES_ACTION, beverages))
        }

        catch (error) {

            console.log(error)
            this.dispatch(createModelAction(ProductActionType.
                ↪ SET_BEVERAGES_ACTION, []))
        }
    }

    async getPastries() {

        try {

            const pastries = await dataContext.pastryContext.getPastries()

            this.dispatch(createModelAction(ProductActionType.
                ↪ SET_PASTRIES_ACTION, pastries))
        }

        catch (error) {

            console.log(error)
            this.dispatch(createModelAction(ProductActionType.
                ↪ SET_PASTRIES_ACTION, []))
        }
    }
}
```

```
export default ProductActionController
```

- The *Service* or *Controller* separates the work of getting data from the component
- It will dispatch the changes to the Redux store when the work is complete

1.3.2 Action Object and Types

```
const ProductActionType = {
    SET_BEVERAGES_ACTION: 'set_beverages_action',
    SET_PASTRIES_ACTION: 'set_pastries_action'
}

export default ProductActionType
```

```
function createModelAction(type, payload) {

    return { type: type, payload: payload }
}

export { createModelAction }
```

- The controller dispatches *actions*, objects with a *type* and data
- All actions are the same, so the function to create them is in *model*
- Action types are defined in the feature containing the controller, types, state, and reducer

1.3.3 Reducer

```
import update from 'immutability-helper';

import ProductActionType from './ProductActionType'
import ProductModelState from './ProductModelState'

class ProductModelStateReducer {

    constructor() {

        this.reduce = this.reduce.bind(this)
    }

    reduce(state, action) {

        let resultState = state ? state : new ProductModelState()

        switch (action.type) {

            case ProductActionType.SET_BEVERAGES_ACTION:
                resultState = this.reduceBeverages(resultState, action.payload)
                break
```

```
            case ProductActionType.SET_PASTRIES_ACTION:
                resultState = this.reducePastries(resultState, action.payload)
                break

            default:
                break
        }

        return resultState
    }

    reduceBeverages(state, beverages) {

        return update(state, { beverages: { $set: beverages }})
    }

    reducePastries(state, pastries) {

        return update(state, { pastries: { $set: pastries }})
    }
}

export default ProductModelStateReducer
```

- The Redux store is immutable
- The reducer accepts a store and an action, and creates a new state to replace the original

1.3.4 State

```
class ProductModelState {

    constructor() {

        this.beverages = [];
        this.pastries = [];
    }
}

export default ProductModelState
```

- The Redux store contains an immutable state, composed of the state of different features
- The reducer is passed the current state for its feature

1.3.5 Store

```
import { combineReducers, createStore } from 'redux'

import ProductModelState from '../data-access/ProductModelState'
import ProductModelStateReducer from '../data-access/ProductModelStateReducer'

class ApplicationModelStoreController {
```

```
    constructor() {

        const productModelStateReducer = new ProductModelStateReducer()
        const mapReducersToModelState = { products: productModelStateReducer.
            ↪ reduce }

        this.reduxStore = createStore(combineReducers(mapReducersToModelState),
            ↪ this.initialState)
    }

    get store() {

        return this.reduxStore
    }

    get initialState() {

        // The initial state passed to createStore MUST be a plain object, not
            ↪ an instance of
        // ApplicationModelState. The constructor bound in the class gets kicked
            ↪  back from
        // createStore as another property it doesn't expect! That is why the
            ↪ ApplicationModelState
        // is an interface, not an initialized class.

        return {

            products: new ProductModelState()
        }
    }
}

export default ApplicationModelStoreController
```

- This class builds the Redux store from the states, and declare the reducers to use

1.3.6 Declaring the Redux Store

```
class App extends Component {

    constructor(props) {

        super(props)

        this.applicationModelStoreController = new
            ↪ ApplicationModelStoreController()
    }

    render() {

        return (
            <Provider store={ this.applicationModelStoreController.store }>
```

- The App will declare the store by building an instance of *ApplicationModelStore*
- The Store becomes a global value wrapping the rest of the application

1.3.7 Inject the Data

```
import React, { Component } from 'react'
import { connect } from 'react-redux'

import '../assets/styles/application.css'
import ProductList from './ProductList'
import ProductActionController from '../data-access/ProductActionController'

class Menu extends Component {

    componentDidMount() {

        this.props.productActionController && this.props.productActionController
            ↪ .getBeverages();
        this.props.productActionController && this.props.productActionController
            ↪ .getPastries();
    }

    static mapStateToProps(state, ownProps) {

        return {

            beverages: state.products.beverages,
            pastries: state.products.pastries,
        }
    }

    static mapDispatchToProps(dispatch, ownProps) {

        return {

            productActionController: new ProductActionController(dispatch)
        }
    }

    render() {

        return (
            <div className="app-content">
                <ProductList title="Beverages" products={ this.props.beverages }
                    ↪ />
                <ProductList title="Pastries" products={ this.props.beverages }
                    ↪ />
            </div>
        )
    }
}

export default connect(Menu.mapStateToProps, Menu.mapDispatchToProps)(Menu)
```

- The component needs to trigger loading the data, and get the data after the store is updated
- The data can be injected into the component, and the component is re-renderd every time it changes
- The Redux dispatch function can be injected, we need it to create the controller