React Libraries:

--

Whenever we program in reeact via routing, we have tobring in a third party API. Styled-components and Axios are very important ones. Today we are focusing on React-Router.

With react, we handle routing a little differently. We want to hve a very smooth user experience, and to acheive this we use the library.

Route

Router (browser router)

Switch

Link

Redirect

These are going to be already built components, and weneed to leverage these in certain ways.

When we thnk about routing, I click on alink and the srver sends something back

Client side routing is not changing anything on the server side. We don’t have to do any sort of screen refresh. Client side routing gives you a lot more control over what is happening. refresh and redirect are not apart of the equation here.

Window.history.back

Window,history.forward

All a react router does is provide a very good if/else statement.

RR is not technically associated with react itself, and one of he contributors lives in Atlanta.

Today we will be focusing on routing, so the first thing to do is install react-router. So install react-router-dom

“npm i react-router-dom”

the app we are building is a very simple app. Check credits, check debits, a profile page. Along with a homepage.

These will have pieces of the state or data. They will all be sharing data so state has to be at the highest level

The react router only needs to be imported on one level (app.js)

Then, at the top of App, the first thing you import are APIs, and Routers. So that means :

import { BrowserRouter as Router } from 'react-router-dom'

Needs to go above any components Ive built. After installing the react-router-dom, the syntax to import it is above.

NEXT WE HAVE TO SET UP OUR REACT APP

In App.js :

class App extends Component {

render() {

return (

<div>

<Router>

<Switch />

</Router>

</div>

);

}

}

Switch renders only one view at a time. Renders only one component at a time

When routing, <Route exact path="/" component={Home} />

<Route path="/about" component={About} />

You have to put “exact” in route exact path = “/” THEN reguar syntax after; BECAUSE we are using the “Switch” component.

To be very very explicit, exact path helps, and with switch you can just use switch for most if not all routes, Switch helps keep the routing on track.

Now we create a NAvbar Component

<h3>Bank of React</h3>

<Link to="/">Home</Link>

<br/><br/>

<Link to="/"Home>Home</Link>

We import Link

import { Link } from 'react-router-dom'

Link creates a seamless UX instead of using an A which will just refresh

This is a much less clunky way of rendering a different “Page”

Declare a state in App, in order for it to be passed down to the other declared states:

class App extends Component {

state = {

accountBalance: 400

}

Whenever we declare a state on the top most level it can be filtered down to the other components, and called with, for instace:

accountBalance ={this.state.

Its only state on the component that its created on .

The state CREATES this.props.accountbalance.

You can say return <Home hamburger={this.state.accountBalance}

& then on the corresponding page, Home, you could say;

AccountBalance dogfood={this.props.hamburger} and it would cascade down the information through chain.

We keep the titles all matching so they are much easier to keep up with.

<Switch>

<Route exact path="/" render={Homewrapper} />

<Route path="/about" component={About} />

<Route path="/user/:userName" component={Userprofile} />

</Switch>

This.props.accountBalance

class App extends Component {

state = {

accountBalance: 400,

currentUser: {

userName: '',

memberSince: ''

}

}

updateCurrentUser = () => {}

render() {

const Homewrapper = () => {

return <Home accountBalance={this.state.accountBalance} />

}

const LogInWrapper = () => {

return <Login />

}

const UserProfileWrapper = () => {

return <UserProfile currentUser={this.updateCurrentUser} />

}

Whenever adding extra properties to for instance CurrentUser, setting a wrapper is wise, if its child within a parent it wouldn’t be necessary . If I go to my user page if I don’t have anything in the current props username itredirects

.Map() Notes & State:

class App extends Component {

state = {

stockData : [

{ "name": "Apple Inc.", "symbol": "AAPL", "lastPrice": 140.64, "change": -0.280000000000001, "high": 141.74, "low": 140.35, "open": 141.5 },

{ "name": "Microsoft Corporation", "symbol": "MSFT", "lastPrice": 64.98, "change": 0.109999999999999, "high": 65.45, "low": 64.76, "open": 65.12 },

{ "name": "Alphabet Inc.", "symbol": "GOOGL", "lastPrice": 835.14, "change": -4.50999999999999, "high": 844, "low": 829.1, "open": 842 },

{ "name": "Facebook, Inc.", "symbol": "FB", "lastPrice": 140.34, "change": 0.810000000000002, "high": 141.0244, "low": 139.76, "open": 140.08 },

{ "name": "Oracle Corporation", "symbol": "ORCL", "lastPrice": 44.65, "change": -0.300000000000004, "high": 45.09, "low": 44.575, "open": 44.91 },

{ "name": "Intel Corporation", "symbol": "INTL", "lastPrice": 36.16, "change": -0.370000000000005, "high": 36.78, "low": 36.125, "open": 36.58 }

]

}

Here we put the array data into the state. This way we can loop through this data, and call upon it on a *different page* . State also create properties that can be called upon. React is trickle down with the Data. So we have to create states at the highest point in the application, *App.JS.*

ender() {

const stocks = () =>{

return(<Stock stockData={this.state.stockData}/>)

}

return (

Here, we are creating a variable underneath the state array. This variable is called *stocks* to remember it easily but it can be named anything. Now we create a function, create a state that can be called on in a different page. This function will be called camelcase *stockData*

import React, { Component } from 'react';

import { Link } from 'react-router-dom'

class Stock extends Component {

render() {

return (

<div>

<h3>Stock</h3>

{this.props.stockData.map((stock, index)=>{

The component “Stock” is where we wish to display data from the array stockData. We need To loop through that data after we declare it with *Props*. Props is declared in order for the original stockData array to be referenced. After *this.props.stockData* we will want to declare ***.map.*** this is how we will not only loop through the stockData, but also *pull this data from the array to be placed where we want it to go…*

***.map*** has two arguments, the first being stock, to reference the class. The second being index (which will come in handy when we reference this function again in a few lines, in order to chain this data together to work together).

return (

<div key={index}>

<Link to='/stocks/:symbol'>{stock.name} </Link>

<br/>

{stock.symbol}

</div>)

})}

so, what do we want from the map declaration? We want to loop through the stock data and ***return*** something. Declare a ***key*** within a div holder and let it equal the index.

**React Router**

**Lesson Objectives:**

* Describe the difference between server-side and client-side routing
* Add client-side routing to our React apps using React Router
* Set up React Router to enable common browser behavior
* Understand built in React Router components (Route, Router, Switch, Link, Redirect)

**Client-side Routing**

Client-side routing is a bit of a misnomer.

On the server, routing generally refers to the way we define the URLs and RESTful resources that make up our application. Whether we are asking for data from the database or persisting data, our server needs to know where the data lives. Server routes help us keep track of this information.

In the browser, things are a little different. When we build Single-Page Applications, we render our data inside of the browser. The data lives on a server, so our data's "addresses" have been defined elsewhere. We only need to know what these pre-defined addresses are to consume them. We'll still have a lot of different views for our data, and we won't want to show all of them on the page at once. Client-side routing is how we'll describe which views we are showing on the page at any given time.

**Routing in React**

There is no way to handle client-side routing in the React library. Instead, there are multiple libraries available to handle this specific task. The most popular library is called React Router. This library is hands down the most popular solution for client-side routing. Recently, React Router made a major overhaul to their library with the release of v4. In the newest version, the team behind React Router refactored the library to make everything just a regular React components. Some online tutorials have not yet updated their libraries, so we recommend using the [official docs](https://reacttraining.com/react-router/)

**Setting up React Router**

React Router is going to allow us to swap out sets of components using familiar "routing" patterns, rather than writing lots of complicated if-statements in our JavaScript.

To demonstrate the power of this tool, we're going to build a personal banking application, where we can independently display the Debits and Credits made to each account.

**Set up:**

Before we jump into React Router, let's create a fresh React application in our class exercises folder:

$ create-react-app bank-of-react

Let's start up our application and make sure everything looks good before we move on.

**Installing React Router:**

React Router is packaged up nicely in npm. We can use npm to quickly pull the Router code into our application. The package we'll need is react-router-dom:

$ npm i react-router-dom

**Setting Up the Router:**

* Now that we've pulled in the React Router code, we need to create a new Router within our application. Because React is not a browser-only framework, the React Router team has provided for us a few different routers, specific to each environment. For our browser-rendered React applications, we'll want to use the BrowserRouter.

Let's go into our App.js and pull in the BrowserRouter as a component called Router. Let's also take the opportunity to clear out our React boilerplate and replace it with "Hello, World!":

import React, {Component} from 'react'; import {BrowserRouter as Router} from 'react-router-dom'; class App extends Component { render() { return ( <div className="App"> Hello, World! </div> ); } } export default App;

* Once we've imported our Router, the only set-up left is to wrap our top-level component inside of a <Router />component, like so:

import React, {Component} from 'react'; import {BrowserRouter as Router} from 'react-router-dom'; class App extends Component { render() { return ( <Router> <div className="App"> Hello, World! </div> </Router> ); } } export default App;

* With a few very simple lines of code, all of the goodness of React Router becomes available to us. Let's explore!

**Creating Routes**

* Let's add a Home component to our application. The first thing we'll need to do is create our component:

// src/components/Home.js import React, { Component } from 'react'; class Home extends Component { render() { return ( <div> <img src="https://letstalkpayments.com/wp-content/uploads/2016/04/Bank.png" alt="bank"/> <h1>Bank of React</h1> </div> ); } } export default Home;

* Now that we have a nicely packaged Home component, we'll want to display this component on the page instead of the obligatory "Hello, World!". We could simply mount this component on the App.js component, like we've done in the past. This would work absolutely fine, but what will happen once we've navigated away from this page?

**How would we get back to the very simple, static Home page that we've set up for ourselves?**

* We could create a "flag" on the state of the main component called showHomePage and whenever showHomePage === true we'd replace everything else with the Home component. This would work, but in the long run we'd have to pass showHomePage as a prop to any component that needed to link back to the Home component.

This may seem doable for a small application. However, if we were to "scale" this to any reasonably-sized app, we'd quickly see that every component would need if-statements for each and every page it could possibly link to. Our component structure would quickly become un-manageable. We haven't even displayed our Home page yet, and the need for React Router has already set in...

* Instead of simply mounting our Home component, let's instead assign a Route to it. We can do this very simply by importing the Route component from react-router-dom and mounting it within our App.js. Let's try it out:

import React, {Component} from 'react'; import {BrowserRouter as Router, Switch, Route} from 'react-router-dom'; import Home from './components/Home'; class App extends Component { render() { return ( <Router> <Switch> <Route exact path="/" component={Home}/> </Switch> </Router> ); } } export default App;

* If we check the browser, we should now see our Home page rendered instead of the previous "Hello, World!". Let's break down the new syntax:

... <Route exact path="/" component={Home}/> ...

* The first thing we should notice is that adding Routes to our application is as simple as mounting any other component. All we have to do is import it and mount it with some props passed in. The props that we are using tell the Route to mount a Home component any time the browser sees a request to *exactly* the '/' route. The exact prop is very important, so the browser doesn't show our Home page when we add any other paths after the initial /.

**By Default, more than one route can be rendered at a time. We don't have a use case to show multiple routes, so we will wrap our code in a <Switch> component to make sure we only show the results of one route at a time. For more info about switch, check out**[**this link**](https://reacttraining.com/react-router/web/api/Switch)

**Passing Props to Routes**

* Now let's add some props to our Home component! On our banking home page, we'll want to view our account balance. Let's create an AccountBalance component and pass the balance as a prop. We'll want to use this information elsewhere, so we'll keep the value on the state of our App.js.

// src/components/AccountBalance.js import React, {Component} from 'react'; class AccountBalance extends Component { render() { return ( <div> Balance: {this.props.accountBalance} </div> ); } } export default AccountBalance;

// src/components/Home.js import React, {Component} from 'react'; import AccountBalance from './AccountBalance'; class Home extends Component { render() { return ( <div> <img src="https://letstalkpayments.com/wp-content/uploads/2016/04/Bank.png" alt="bank"/> <h1>Bank of React</h1> <AccountBalance accountBalance={this.props.accountBalance}/> </div> ); } } export default Home;

// src/components/App.js ... constructor() { super(); this.state = { accountBalance: 14568.27 } } ...

* Now we have all of the pieces set up. We just need to tell our "/" Route to pass the accountBalance as a prop each time it loads the Home component. We might try something like this:

<Route exact path="/" component={<Home accountBalance={this.state.accountBalance}/>}/>

* ...but if we check the browser we'll see that this throws an error. Route components expect us to pass a "reference" to a component as our component= prop. This means we can't pass an already-built component, like the one above. Fortunately, there is a very simple alternative:

... render() { const HomeComponent = () => (<Home accountBalance={this.state.accountBalance}/>); return ( <Router> <div> <Route exact path="/" render={HomeComponent}/> </div> </Router> ); } ...

* The component prop won't accept a pre-built component, but React Router has provided an alternative rendermethod that will. All we have to do is save our component as a variable and pass it straight in as a prop.

Now we can refresh the page and see our Home page with a proper AccountBalance.

* + Notice the difference between the render and component prop. If you need to pass props at the Router level, you must use render... otherwise use the component prop.

**Linking to Routes**

* Now that we have set up a Route for our Home component, we can navigate back to the component at any time. Let's build out a UserProfile page and create a Route for it in our App.js:

// src/components/UserProfile.js import React, {Component} from 'react'; class UserProfile extends Component { render() { return ( <div> <h1>User Profile</h1> <div>Username: {this.props.userName}</div> <div>Member Since: {this.props.memberSince}</div> </div> ); } } export default UserProfile;

// src/App.js import React, {Component} from 'react'; import {BrowserRouter as Router, Route} from 'react-router-dom'; import Home from './components/Home'; import UserProfile from './components/UserProfile'; class App extends Component { constructor() { super(); this.state = { accountBalance: 14568.27, currentUser: { userName: 'bob\_loblaw', memberSince: '08/23/99', } } } render() { const HomeComponent = () => (<Home accountBalance={this.state.accountBalance}/>); const UserProfileComponent = () => ( <UserProfile userName={this.state.currentUser.userName} memberSince={this.state.currentUser.memberSince} /> ); return ( <Router> <div> <Route exact path="/" render={HomeComponent}/> <Route exact path="/userProfile" render={UserProfileComponent}/> </div> </Router> ); } } export default App;

* If we type http://localhost:3000/userProfile into our browser, we see our new UserProfile on the screen. This isn't a great user experience, obviously. Let's instead add a link to our UserProfile from the Home component. React Router provides another very simple solution for this: the Link component.

import React, {Component} from 'react'; import AccountBalance from './AccountBalance'; import {Link} from 'react-router-dom'; class Home extends Component { render() { return ( <div> <img src="https://letstalkpayments.com/wp-content/uploads/2016/04/Bank.png" alt="bank"/> <h1>Bank of React</h1> <Link to="/userProfile">User Profile</Link> <AccountBalance accountBalance={this.props.accountBalance}/> </div> ); } } export default Home;

* React Router's Link component takes a very simple prop to= that tells us where we want to redirect. **We can link to any of our components, even within child components, as long as a Route has been initialized next to or above the Link in our tree of components.**

**You Do: Linking Back to the Home Component (5 minutes)**

* We are now able to view our UserProfile component through the Home page link, but we are left stranded once we get there. Let's improve our user experience by adding a link back to the Home component inside of the UserProfilecomponent.

**Programattic Redirects**

While the <Link> component is really powerful, it requires a user interaction to navigate to another route. There are plenty of use cases that require a redirect outside of user clicks. One common example is wanting to redirect after a form is submitted. When this happens, we need to use the <Redirect> component that is build into React Router. Let's take a look at this mock Sign In component that we can add to our bank.

// App.js ... mockLogIn = (logInInfo) => { const newUser = {...this.state.currentUser} newUser.userName = logInInfo.userName this.setState({currentUser: newUser}) } ... const LogInComponent = () => (<LogIn user={this.state.currentUser} mockLogIn={this.mockLogIn} {...this.props}/>) ... <Route exact path="/login" render={LogInComponent}/> ...

// Login.js import React, { Component } from 'react' import { Redirect } from 'react-router-dom' class LogIn extends Component { constructor () { super() this.state = { user: { userName: '', password: '' }, redirect: false } } handleChange = (e) => { const updatedUser = {...this.state.user} const inputField = e.target.name const inputValue = e.target.value updatedUser[inputField] = inputValue this.setState({user: updatedUser}) } handleSubmit = (e) => { e.preventDefault() this.props.mockLogIn(this.state.user) this.setState({redirect: true}) } render () { if (this.state.redirect) { return (<Redirect to="/userProfile"/>) } return ( <div> <form onSubmit={this.handleSubmit}> <div> <label htmlFor="userName">User Name</label> <input type="text" name="userName" onChange={this.handleChange} value={this.state.user.userName} /> </div> <div> <label htmlFor="password">Password</label> <input type="password" name="password" /> </div> <button>Log In</button> </form> </div> ) } } export default LogIn

**LAB: Adding Debits and Credits**

Let's add some more features to our banking app, using the following User Stories!

**There is a starter API providing Debits and Credits data located in the bank-of-react-api folder next to this lesson.**

* Run npm install and then npm start to get the API running.
* The API will run on http://localhost:4000.
* The Debits index endpoint is located at http://localhost:4000/debits.
* The Credits index endpoint is located at http://localhost:4000/credits.

**Updating the Account Balance**

Making the Account Balance dynamic: GIVEN I am on any page displaying the Account Balance WHEN I view the Account Balance display area THEN I should see an Account Balance that accurately represents my Debits subtracted from my Credits AND I should be able to see a negative account balance if I have more Debits than Credits

**Adding Debits**

Viewing the Debits page: GIVEN I am on the Home Page WHEN I click on 'Debits' THEN I should be redirected to the Debits page AND I should see a title of 'Debits' on the page

Displaying debits: GIVEN I am on the Debits page WHEN I view the Debits display area THEN I should see all of my debits displayed AND each Debit should display a Debit description AND each Debit should display a Debit amount AND each Debit should display a Debit date

Viewing the Account Balance on the Debits page: GIVEN I am on the Debits page WHEN I view the Account Balance display area THEN I should see my Account Balance displayed

Adding debits: GIVEN I am on the Debits page WHEN I enter a new Debit description AND WHEN I enter a new Debit amount AND WHEN I click 'Add Debit' THEN I should see my new debit added to the Debits display area with the current date AND I should see my Account Balance updated to reflect the new Debit

Viewing the Account Balance on the Debits page: GIVEN I am on the Debits page WHEN I view the Account Balance display area THEN I should see my Account Balance displayed

**Adding Credits**

Viewing the Credits page: GIVEN I am on the Home Page WHEN I click on 'Credits' THEN I should be redirected to the Credits page AND I should see a title of 'Credits' on the page

Displaying debits: GIVEN I am on the Credits page WHEN I view the Credits display area THEN I should see all of my Credits displayed AND each Debit should display a Debit description AND each Debit should display a Debit amount AND each Debit should display a Debit date

Viewing the Account Balance on the Credits page: GIVEN I am on the Credits page WHEN I view the Account Balance display area THEN I should see my Account Balance displayed

Adding Credits: GIVEN I am on the Credits page WHEN I enter a new Debit description AND WHEN I enter a new Debit amount AND WHEN I click 'Add Debit' THEN I should see my new debit added to the Credits display area with the current date AND I should see my Account Balance updated to reflect the new Debit

Viewing the Account Balance on the Credits page: GIVEN I am on the Credits page WHEN I view the Account Balance display area THEN I should see my Account Balance displayed

* © 2018 GitHub, Inc.
* [Help](https://help.github.com/enterprise/2.12)
* [Support](mailto:github@generalassemb.ly)
* [API](https://developer.github.com/enterprise/2.12)
* [Training](https://training.github.com/)
* [Shop](https://shop.github.com/)
* [Blog](https://github.com/blog)
* [About](https://github.com/about)