Framework Training
React
London July 2018
Exercise L-router

- This exercise uses **React Router** to manage navigation in a Single Page React Application.

Installation

- Install and run the starter version of the project.

```
npm install
npm start
```

- The React Router was installed after creating the project with create-react-app: **npm install react-router-dom**

BrowserRouter

- The **BrowserRouter** component manages routing and maps changes of url to changes of component on the page.
- The **Route** component will render a component on the page when the current URL matches a defined path.
- Here, the Team component will appear on the page when the url is "/team"

```
<Route path="/team" component={Team}
```

- The **Switch** component is wrapped around multiple Routes, matching only the first suitable route.
- In the Hike component render method, create four routes:

```
<BrowserRouter>
    <section>
        <NavBar/>
        <Switch>
            <Route path='/team'     component={Team}/>
            <Route path='/contact'  component={Contact}/>
            <Route path='/packs' component={Packs}/>
            <Route path='/admin' component={Admin}/>
        </Switch>
    </section>
</BrowserRouter>
```

- Add a **home path** at the start of the Switch construct.
- Note this requires an **exact attribute** to prevent it matching routes like “/team” which contain “/”.

```
<Route exact={true} path='/' component={Home}/>
```

- Add an **error route** (which has no defined path) to catch any other route.

```
<Route component={Error}/>
```

Link components

- The navigation bar needs to be updated to use React Router Link components rather than HTML link elements.
- This will prevent that page reloading when the user navigates around the application.

```
    <li><Link to='/'>Home</Link></li>
    <li><Link to='/team'>Team</Link></li>
    <li><Link to='/contact'>Contact</Link></li>
    <li><Link to='/packs'>Packs</Link></li>
    <li><Link to='/admin'>Admin</Link></li>
```

Route component render attributes

- The Route component can define a render attribute. This contains an **inline function** that executes instead of instantiating a name Component.
- This code defines a “/version” route which renders version information in a header.

```
<Route path="/version" render= { () => <h2>v14.78</h2> }/>
```

Private Routes

- We want to **limit access** to the “/admin” route which opens an Admin-Tools page.
- We can define a render attribute which checks some boolean value.
- If true, the Admin component is rendered.
- If false, the Router redirects the user back to the home page.
- In the browser web tools define a **localStorage** property called admin.

```
localStorage.admin = 1
```

- Note, localStorage stores strings. To test for this value use an expression like:

```
Number( localStorage.admin ) === 1
```

- Define a new method in the Hike component.

```
  getAuth() {
  return (Number( localStorage.admin ) === 1);
  }
```

- In the Hike component render method, define an object which will be used by the Router Redirect component.

```
let goHome = { pathname: '/',state: { from: this.props.location
}} ;
```

- Define a Route which uses getAuth() to decide whether to render the Admin component or redirect the user back to the home page.

```
<Route
    path='/admin'
    render= { () => this.getAuth() ? <Admin/> :
    <Redirect to={goHome}/> }
/>
```

- Test that this functionality works.

Style the ADMIN link

- The Admin link can be **conditionally styled** to show if access is available.
- The NavBar component needs to use the getAuth() method defined in the Hike component.
- Pass this method down as a prop.

```
<NavBar getAuth={this.getAuth}/>
```

- Conditionally add the “no-admin” CSS class to the Admin link if getAuth returns false.

```
<Link to='/admin'
className={props.getAuth() ? null : "no-admin"} >
```

Packs component

- Packs.js defines an array of objects containing the backpacks that are available in the shop.

- The Packs component should read this array and render a submenu for each pack found:

```
// { code:"2806", desc:"Yellow" },
// <Link to="/packs/2806">Yellow</Link>
```

- We need to pass the pack data into the Packs component as a prop. *Note, the Router also passes additional props into the Pack component.*
- The Route component does not allow this, but we can achieve this using the render attribute.

```
<Route
exact path='/packs'
render={ props =>
<Packs packs={MountainShed} {...props}/>} />
```

- Inside the Packs component, the data is now in scope.

```
let {packs} = props;
```

- We can map over the packs array to create a NavBar local to the Packs page.

```
{ packs.map( (p,n) =>
<li key={n}>
<Link to={"/packs/"+p.code}>{p.desc}</Link>
</li> )}
```

Pack routes

- Clicking on the Yellow link in the Packs page generates a 404 error for route **localhost:4016/packs/2806**.
- We need to define a variable route in the Hike component.

```
    <Route exact path='/packs/:code'
    render={ props =>
    <Packs packs={MountainShed} {...props}/>}
    />
```

- This variable route will be passed down to the Packs component in **props.match.params.code**

```
    console.log( props.match.params.code );
```

- We can search the array of packs for the matching pack code.

```
packs.filter( p => p.code === props.match.params.code )
```

- This will return an array of one object if the pack is found.
- Invalid codes will return an empty array.
- Store this first pack to a variable.
- The variable will be undefined for invalid codes.

```
let pack = packs.filter( p => p.code === props.match.params.code
)[0];

console.log(pack); // e.g. {code: "4765", desc: "Orange"}
```

- Using the pack variable, we can conditionally render a matching image with the correct ALT attribute.

```
<section>
{ pack ? <img src={"../images/" + pack.code + ".png"} alt=
{pack.desc} /> : null }
```

- Test that the packs page works.