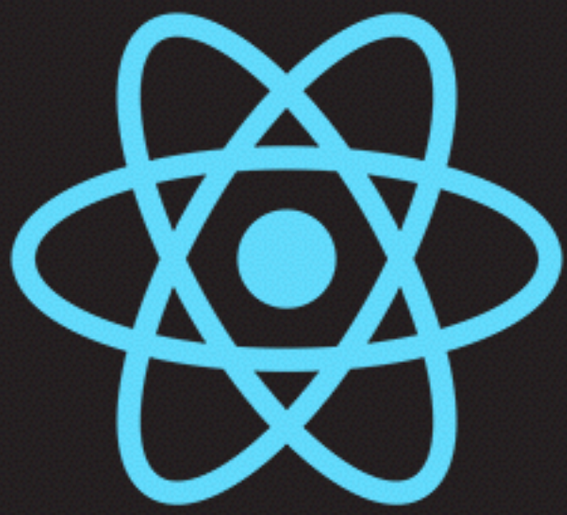
COLLEGIO TECHNOLOGIES INC.

REACT-ROUTER



BEFORE WE BEGIN CODING...LET'S CHAT ABOUT CLIENT VS. SERVER SIDE ROUTING

- You may be accustomed to server-side routing if you have used languages such as PHP, Ruby or Node.js.

- Client-side routing uses JavaScript to dynamically change the user view.
  - This approach is used by many current JavaScript frameworks, such as Angular, Backbone, Ember, and Vue.js.

- With server-side routing, a full page-reload is required to display a new page to the user, which is more expensive than just updating some data inside of an already loaded view.

- Client-side routing watches for changes inside of the browser URL, and overloads the standard page refresh with it's own view.
  - With React, we will watch for these changes, and when detected, will render some new components to the screen.

CLIENT VS. SERVER-SIDE ROUTING

- Client-side routing, because it does not require a full page reload, tends to run faster than server-side routing.

- For our project, we will be using **react-router,** a third party package that will allow us to perform client-side routing inside of our apps.

REACT ROUTER

- React-router is a fantastic 3rd party package that can be used for performing routing inside of a React or React Native project. You can find the official repo for the page at https://github.com/ReactTraining/react-router
  - However, you will find more interesting information on how to actually use it at https://reacttraining.com/react-router/. In our case, we want to see how it will work with a web application, so click on 'web'.
- The official documentation is great if you need a refresher down the road, as it outlines all of the features of React router and provides examples.
- You can install react-router via **yarn:**
  - **yarn add react-router-dom** (We are building a web application)

REACT ROUTER

- Once React Router is installed, we'll need to add it to our application. You will import most of the functionality inside of your **app.js** file:

- Import your react-router classes. To begin, we'll use **BrowserRouter** and **Route** components, which you can find inside of the documentation.
  - **BrowserRouter** is the main component that we'll use to implement client-side routing via the **HTML5 history API**.
  - **Route** is the component used to specify each individual route.

- **import { BrowserRouter, Route } from 'react-router-dom';**

REACT ROUTER

- Once the required router components are imported, we can begin creating the overall routing structure for our application within a **BrowserRouter** component, which will have multiple **Route** components as children.
- Each route component should contain at least 3 props:
  - **path**: the actual path that the route is being assigned to.
  - **component**: the main component to render for that route. Think something similar to our main ***FreeAgentTracker*** component.
  - **exact**: specify whether to match the ***exact*** URL of the path.
- Let's look at a basic example:

EXAMPLE: REACT-ROUTER WITH 2 ROUTES

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter, Route } from 'react-router-dom';
import 'normalize.css/normalize.css';
import './styles/style.scss';

const DashboardPage = () => (
    <div>
        <h1>Hey There! This is the dashboard!</h1>
    </div>
);

const ProfilePage = () => (
    <div>
        <h1>Hey There! This is your profile!</h1>
    </div>
);

const ourRoutes = (
    <BrowserRouter>
        <div>
            <Route path="/" component={DashboardPage} exact={true} />
            <Route path="/profile" component={ProfilePage} exact={true} />
        </div>
    </BrowserRouter>
);

ReactDOM.render(ourRoutes, document.getElementById('app'));
```

REVIEW: REACT-ROUTER EXAMPLE

- Let's review what we have in the previous slide:
  - We import our BrowserRouter and Route components via ***import***.
  - We create two stateless functional components: ***DashboardPage*** and ***ProfilePage***.
  - We declare our Routes inside of a JSX variable called ***ourRoutes***.
  - We use ***ourRoutes*** as the first argument of ReactDOM.render() to render the ***BrowserRouter*** component as the top level component, controlling the routing.

- Now, let's fire up a server and go to the **/profile** page. Hmmm…

REACT-ROUTER: CONFIGURING THE DEV-SERVER

- The Webpack dev server by default resorts to performing server-side routing. In order to set up your dev-server to perform client-side routing, you must first let it know inside of **webpack.config.js**.
  - We do this by setting a **historyApiFallback** property of the **devServer** object to ***true***.
  - Please note, you will need to restart the dev server before this change is performed.

```
devServer: {
    contentBase: path.join(__dirname, 'public'),
    historyApiFallback: true        // needed to handle client-side routing
}
```

REACT-ROUTER: WHY SET EXACT TO TRUE?

- You may have noticed that inside of the **Route** components that we declare, there is an **exact** prop set to ***true***. This is very important to ensuring that your app runs smoothly.

- If the exact prop wasn't set or was set to false, routes are called based on a route hierarchy. In the case of a route called **/profile/team/player**, the router would match routes (and render) all of the following:
  - **/**
  - **/profile**
  - **/profile/team**
  - **/profile/team/player**

- You almost never want this exact scenario, which is why this is important.

REACT-ROUTER: THE 404 PAGE

- An important part of most apps is setting up your erroneous conditions. You want to provide your user with a pleasant experience, and one of the ways to do this is by not showing them a generic 404 page if they are in the wrong location, or showing them a pile of error code (really wrong!!)

- Adding a 404 page is easy using React-router:
  - You begin by specifying a route without a **path** prop as the last **Route** component.
  - In addition, you will need to import a new component from react-router called ***Switch***. This component will run through each route from top to bottom, and when a match is found, will stop running. If no match is found, it will run the last match.
  - Finally, replace the ** ** tag inside of our previous example with the **<Switch>** tag.

EXAMPLE: REACT-ROUTER WITH 404 PAGE

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter, Route, Switch } from 'react-router-dom';
import 'normalize.css/normalize.css';
import './styles/style.scss';

const DashboardPage = () => (…
);

const ProfilePage = () => (…
);

const NotFoundPage = () => (
    <div>
        <h1>Uh oh, looks like you got lost somewhere!</h1>
    </div>
);

const ourRoutes = (
    <BrowserRouter>
        <Switch>
            <Route path="/" component={DashboardPage} exact={true} />
            <Route path="/profile" component={ProfilePage} exact={true} />
            <Route component={NotFoundPage} />
        </Switch>
    </BrowserRouter>
);

ReactDOM.render(ourRoutes, document.getElementById('app'));
```

REACT-ROUTER: LINKING BETWEEN ROUTES USING THE **LINK** COMPONENT

- When linking between different routes within our application, we want to avoid re-loading the entire page, which is done whenever we use the standard anchor **<a>** tag.
- In order to make use of client-side routing within our application links, we'll need to use another React-router component: **Link**.
  - This component is used to create a generic link within your application and link via the router.
  - Instead of a **href** attribute, you specify a **to** attribute in the Link component.
- You can also use another component called **NavLink** that is great for declaring navigational links, allowing you to take advantage of additional features such as specifying the active link with **activeClassName**.

```
const NotFoundPage = () => (
    <div>
        <h1>Uh oh, looks like you got lost somewhere!</h1>
        <p><Link to="/">Back to the Dashboard!</Link></p>
    </div>
);
```

REACT-ROUTER: USING COMMON COMPONENTS WITHIN YOUR OVERALL STRUCTURE

- When we declared our routes inside of our **BrowserRouter** component, know that if there are components that are used on every page, they can also be declared here.
  - Think of things such as an app header and footer.
- Remember, you still need to encapsulate all sub-components of the **BrowserRouter** into a single element.
- Using this method can save us some time from having to declare a common element inside of every route main component.

EXAMPLE: USING A COMMON COMPONENT INSIDE OF YOUR ROUTING STRUCTURE

```
const Header = () => (
    <header>
        <h1>This is my app!</h1>
    </header>
);

const ourRoutes = (
    <BrowserRouter>
        <div>
            <Header />
            <Switch>
                <Route path="/" component={DashboardPage} exact={true} />
                <Route path="/profile" component={ProfilePage} exact={true} />
                <Route component={NotFoundPage} />
            </Switch>
        </div>
    </BrowserRouter>
);

ReactDOM.render(ourRoutes, document.getElementById('app'));
```

REACT-ROUTER: DYNAMIC URLS AND PASSING QUERY STRINGS

- One of the main things that makes routing so powerful is the ability to display dynamic pages using passed-in parameters.
  - We see this all the time when making GET requests on the web.

- With React, it is very easy to pass in dynamic URLs to your app:
  - Dynamic components of the URL within the path inside of the **Route** component are preceded by a colon (:).

```
const MainRouter = () => (
    <BrowserRouter>
        <div>
            <Header />
            <Switch>
                <Route path="/" component={DashboardPage} exact={true} />
                <Route path="/profile/:userid" component={ProfilePage} />
                <Route component={NotFoundPage} />
            </Switch>
        </div>
    </BrowserRouter>
);
```

REACT-ROUTER: DYNAMIC URLS AND PASSING QUERY STRINGS

- Once a dynamic variable is passed into a URL, it can then be accessed inside of the component via the **props.match.params** object.
  - In our case, we'd be looking for **props.match.params.userid**.

```
const ProfilePage = (props) => (
    <div>
        <h2>Hey There! This is your profile! Your user ID is {props.match.params.userid}</h2>
    </div>
);
```

- That's it! adding dynamic elements to your routes is easy with React Router!

CONCLUSION: REACT-ROUTER

- We've only scratched the surface of what the React Router can do, and we'll be using it more once we start working with React Native, but for now, know that we can use it to:
  - Perform client-side routing within our web app.
  - Set up pretty URLS on our app routes.
  - Add dynamic variables to our URLs
  - Add a custom 404 page.

- One last thing to note about setting up routing inside of your app: like with components, it is a good idea to separate out routes inside of their own folder. I like to use a folder called **routes** to specify the specific components designed to contain routes.

EXERCISE: SETTING UP AN APP CLIENT-SIDE ROUTING STRUCTURE!

- Time to try it out! The best way to get used to React router is by setting up some routing yourself.

- Let’s add an edit page to the free agent tracker app. Don’t worry about whether you created it or not, we’ll get to that part later!

RESOURCES

- **React Router:**
  - **Official Repo:** https://github.com/ReactTraining/react-router
  - **Documentation:** https://reacttraining.com/react-router/
- **Client-side Routing vs. Server-side Routing (for more information):**
  - https://medium.com/@wilbo/server-side-vs-client-side-routing-71d710e9227f
  - https://stackoverflow.com/questions/23975199/when-to-use-client-side-routing-or-server-side-routing
  - https://medium.freecodecamp.org/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d

DISCLAIMER

- Copyright 2018 Collegio Technologies Inc.
- All content within this presentation has been compiled by Collegio Technologies Inc. All material has been sourced through open source resources and source code and as such conforms to Canadian copyright laws and regulations. All rights reserved.