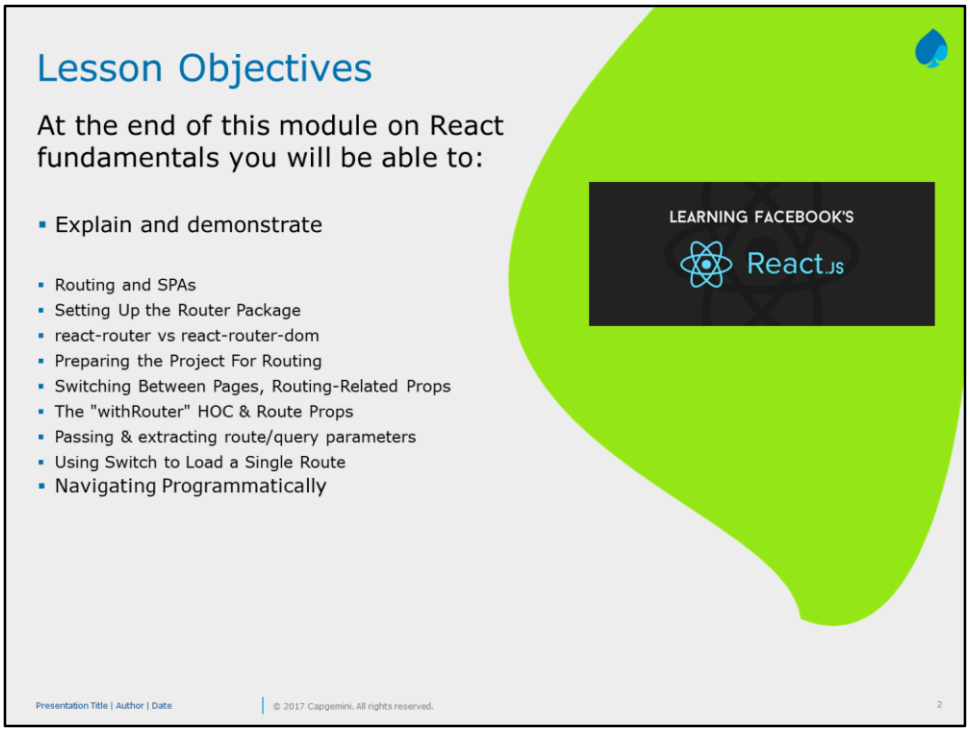
Instructor Notes:

Add instructor notes here.

Instructor Notes:

Add instructor notes here.



Instructor Notes:

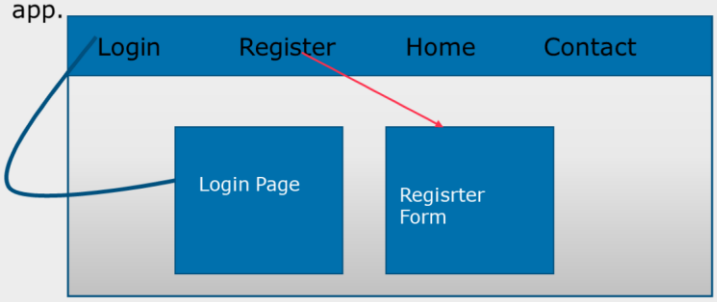
Routing and SPAs

Single-page apps are different from normal multi-page apps. The main difference between them is

- Navigating a single-page app doesn't involve going to an entirely new page. Instead, loads the content within the same page itself.

For Implementing SPA we make use of Routing(React-Router)

- Routing is where you try to map URLs to destinations that aren't physical pages such as the individual views in your single-page app.



Instructor Notes:

Add instructor notes here.

What's in a URL :

A URL is a reference to a web resource

Are building blocks for creating SPA's.

Single-page applications (SPAs) are web apps that load once and then dynamically update elements on the page using JavaScript. Every React app we've built so far has been a type of SPA.

There are many routing libraries for React, and best of it is **React Router**. React Router gives us a good foundation for building rich application which has views and urls

routing involves two functionality:

(1)Modifying the location of the app (the URL)

(2) determining what React components to render at a given location.

Browser makes a request to the server for this page.
The server doesn't care about the pathname. Instead, it just returns a standard index.html that includes the React app and any static assets.
The React app loads.
The React app extracts the identifiers from the URL and uses these identifiers to make an API call to fetch the data for the artist and the album. It might make this call to the same server.
The React app renders the page using data it received from the API call.

```
var React = require('react');
var ReactDOM = require('react-dom');
var ReactRouter = require('react-router');  // require react-router module
var Router = ReactRouter.Router; //Get Router Reference
var Route = ReactRouter.Route; //Get Route Reference
    ReactDOM.render((
      <Router>
                    <Route path="/" component={Home}/>
                    <Route path="/about" component={AboutUs}/>
                    <Route path="/contact" component={ContactUs}/>
    </Router>
    ), document.getElementById('app'));
```

Each <Route>will render its respective component when its path matches the URL. Only one of these three components will be rendered into the at any given time. With this strategy, we mount the router to the DOM once, then the router swap components in and out with route changes.

Route Matching:
<Route path="users/:userId" component={UserProfile}/>

The above route will match when the user visits any path that starts with users/ and has any value afterwards. It will match /users/1, /users/143, or even /users/abd(which you'll need to be validated on our own).

Instructor Notes:

Setting Up the Router Package

To use routing, we have to pull down React, React Router and React DOM:

We have to make use below command for installation

npm install react react-dom react-router --save

Instructor Notes:

Add instructor notes here.

React-route vs react-route-dom in React Router v4

The latest version of React Router, v4, is a major shift from its predecessors.

1. With **React router v4**, routing is not centralized anymore instead it becomes a part of the rest of the app layout and UI.
2. Browser specific routing components live in **react-router-dom** instead of **react-router** so imports need to be changed to be from **react-router-dom** package
3. Introducing new components such as **BrowserRouter** and **HashRouter** for specific use cases
4. No more use of **{props.children}** for nesting components in v4 React Router.

When we go for program ensure we install **npm i react-router-dom –save** through command prompt

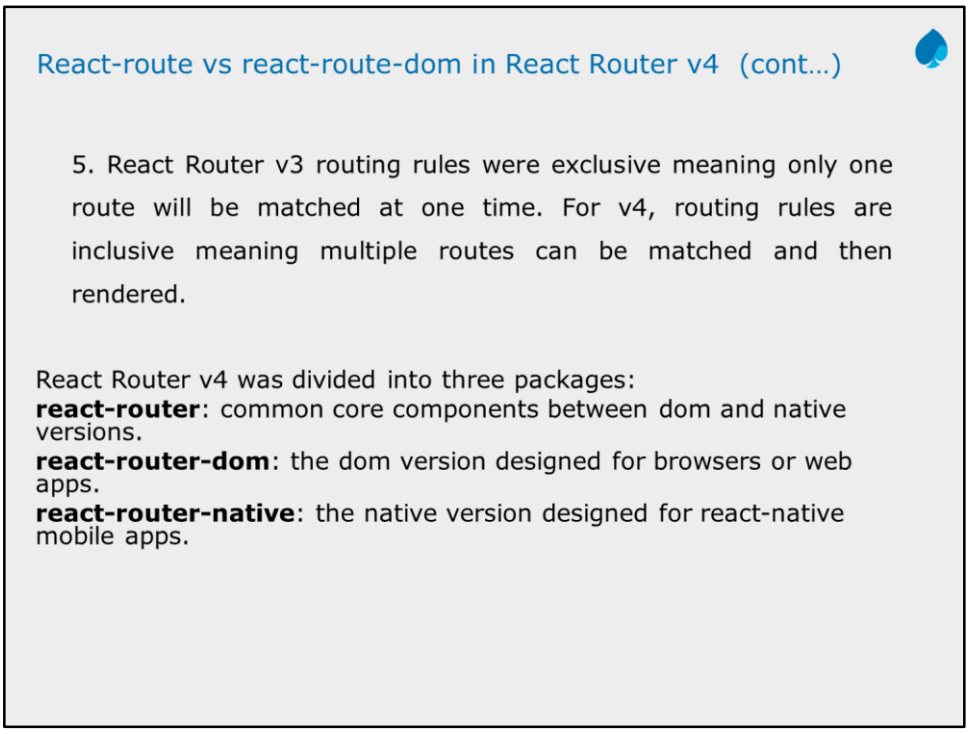
```
var React = require('react');
var ReactDOM = require('react-dom');
var ReactRouter = require('react-router');  // require react-router module
var Router = ReactRouter.Router; // Get Router Reference
var Route = ReactRouter.Route;  //Get Route Reference

ReactDOM.render((
  <Router>
                <Route path="/" component={Home}/>
                <Route path="/about" component={AboutUs}/>
                <Route path="/contact" component={ContactUs}/>
</Router>
), document.getElementById('app'));
```

Each <Route>will render its respective component when its path matches the URL. Only one of these three components will be rendered into the at any given time. With this strategy, we mount the router to the DOM once, then the router swap components in and out with route changes.

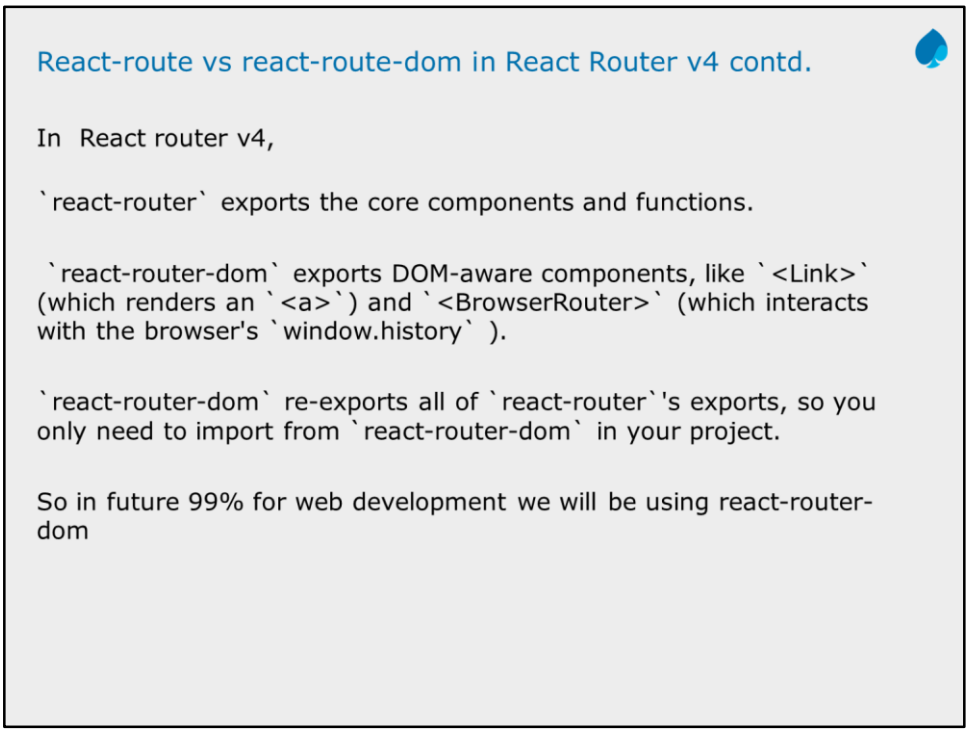
Route Matching:
<Route path="users/:userId" component={UserProfile}/>

The above route will match when the user visits any path that starts with users/ and has any value afterwards. It will match /users/1, /users/143, or even /users/abd(which you'll need to be validated on our own).

Instructor Notes:



Instructor Notes:



Instructor Notes:

Add instructor notes here.

React Router's core components

There are three types of components in React Router

1. Router components
2. Route matching components
3. Navigation components.

All of the components that you use in a web application should be imported from react-router-dom.

```
import { BrowserRouter, Route, Link } from 'react-router-dom'
```

React-Routing

A routing library is a key component of any complex, single-page application.
React Router isn't the only viable solution in the React/Redux ecosystem, its growing and there are many in the system.(React Redux is not covered in this syllabus)

Browser makes a request to the server for this page.
The server doesn't care about the pathname. Instead, it just returns a standard index.html that includes the React app and any static assets.
The React app loads.
The React app extracts the identifiers from the URL, and uses these identifiers to make an API call to fetch the data for the artist and the album. It might make this call to the same server.
The React app renders the page using data it received from the API call.

```
var React = require('react');
var ReactDOM = require('react-dom');
var ReactRouter = require('react-router'); // require react-router module
var Router = ReactRouter.Router; //Get Router Reference
var Route = ReactRouter.Route; //Get Route Reference
ReactDOM.render((
  <Router>
        <Route path="/" component={Home}/>
        <Route path="/about" component={AboutUs}/>
        <Route path="/contact" component={ContactUs}/>
</Router>
), document.getElementById('app'));
```

Each <Route>will render its respective component when its path matches the URL. Only one of these three components will be rendered into the at any given time. With this strategy, we mount the router to the DOM once, then the router swap components in and out with route changes.

Route Matching:
<Route path="users/:userId" component={UserProfile}/>

The above route will match when the user visits any path that starts with users/ and has any value afterwards. It will match /users/1, /users/143, or even /users/abd(which you'll need to be validated on our own).

Instructor Notes:

Add instructor notes here.

React-Routing

React isn't a framework, it's a library. Therefore, it doesn't solve all an application's needs.

To create complex SPA (single page application) task like routing requires supporting cast.

React router is one among the front-end router which uses JSX Syntax.

- To install this module : ***npm install react-router***

React router uses <Router> and <Route> components to perform routing.

Like other components <Router> and <Route> does not create DOM, it just defines the rules about how application needs to work based on routes.

```
ReactDOM.render(( <Router>
    //<Route>  Renders Home component  while vising  the path  /
          <Route path="/" component={Home} />
    </Router> ), document.getElementById('app'));
```

Routing is the process of keeping the browser URL in sync with what's being rendered on the page.

React is a popular library for creating single-page applications (SPAs) that are rendered on the client side. An SPA might have multiple **views** (pages).
and unlike the conventional multi-page apps, navigating through these views shouldn't result in the entire page being reloaded. Instead, we want the views to be rendered inline within the current page.

```
var React = require('react');
var ReactDOM = require('react-dom');
var ReactRouter = require('react-router');  // require react-router module
var Router = ReactRouter.Router; //Get Router Reference
var Route = ReactRouter.Route; //Get Route Reference
ReactDOM.render((
  <Router>
                <Route path="/" component={Home}/>
                <Route path="/about" component={AboutUs}/>
                <Route path="/contact" component={ContactUs}/>
</Router>
), document.getElementById('app'));
```

Each <Route>will render its respective component when its path matches the URL. Only one of these three components will be rendered into the at any given time. With this strategy, we mount the router to the DOM once, then the router swap components in and out with route changes.

Route Matching:
<Route path="users/:userId" component={UserProfile}/>

The above route will match when the user visits any path that starts with users/ and has any value afterwards. It will match /users/1, /users/143, or even /users/abd(which you'll need to be validated on our own).

Instructor Notes:

Preparing and setting up react project for routing

Installation

React Router DOM is published to npm so you can install it with either npm

Building a react router firstly we have to import Route and Router from 'react-router' library

```
import {Router, Route} from 'react-router';
```

Then add the below code snippet to render method of react Component

```
<Router>
  <Route exact path="/" component={Home}/>
  <Route path="/contactus" component={Contactus}/>
  <Route path="/login" component={Login}/>
  <Route path="/register" component={Register}/>
</Router>
```

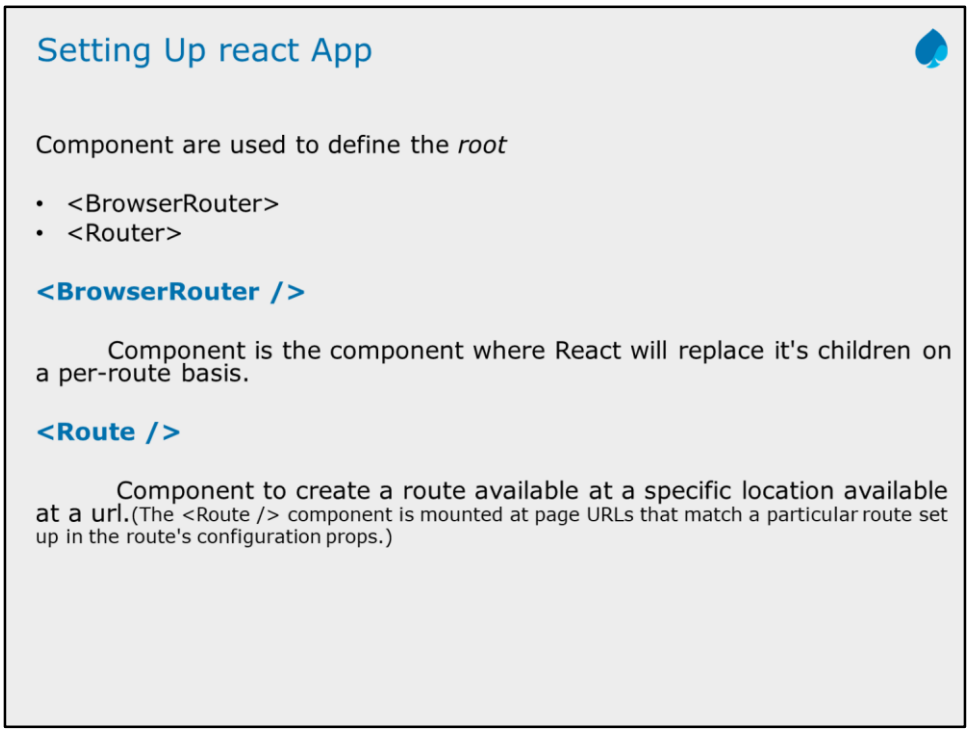
The path attribute defines the route URL and component attribute defines the component for this route.

Note: Instead of <a href="/"> we use <Link to="/">.

show above snippet will look like :

 <Link to="/">Home</Link>   <Link to="/about">About</Link>   <Link to="/topics">Topics</Link>

Instructor Notes:



<Route /> component to create a route available at a specific location available at a url. The <Route /> component is mounted at page URLs that match a particular route set up in the route's configuration props. Previously for client-side navigation we used # (hash) mark to denote the application endpoint.

In <Route /> pass exact= {true} props to the router with path='/':
If you want a route to be rendered only if the paths are exactly the same, you should use the exact props.

```
const Home = () => (<div><h1>Welcome home</h1><Link to='/about'>Go to about</Link></div>)
const About = () => (<div><h1>About</h1><Link to='/'>Go home</Link></div>)
```

In our view we'll need to add a link (or an anchor tag -- <a />) to enable our users to travel freely between the two different routes. However, using the <a /> tag will tell the browser to treat the route like it's a server-side route. Instead, we'll need to use a different component called: <Link />.
The <Link /> component requires a prop called to to point to the client-side route where we want to render. Let's update our Home and About components to use the Link:

If any error thrown in code when using Link , import Link from react-router-dom.

React router 4.x introduced some breaking changes, you'll need to import Link from react-router-dom:

After adding up above code snippet … both routes to show up after clicking on link “go to about”, it displays Info in about component on same page.
This happens because the react router will render *all* content that matches the path (unless otherwise specified). For this case, react router supplies us with the Switch component.

The <Switch /> component will *only render the first matching route* it finds. Let's update our component to use the Switch component. As react router will try to render *both* components, we'll need to specify that we only want an exact match on the root component.

If any error thrown in code when using Switch , import Switch from react-router-dom.

React router 4.x introduced some breaking changes, you'll need to import Switch from react-router-dom:

Advanced :
Routing with render prop:

The most powerful method of using a prop is called render. The render prop is expected to be a function that will be called with the match object along with the location and route configuration.

The render prop allows us to render *whatever* we want in a subroute, which includes rendering other routes

Instructor Notes:

Setting Up react App (contd…)

To define a route, we'll use the <Route /> component export from react-router and pass it a few props:

- path - The path for the route to be active
- component - The component that defines the view of the route

The <Link /> component requires a prop called to to point to the client-side route where we want to render.

The <Switch /> component will *only render the first matching route* it finds.

Instructor Notes:

Add instructor notes here.

Demo

React Router

react-axios-demo-crud(also has roter demo)

Add the notes here.

Instructor Notes:

Routing related Props

We have three prop choices for how you render a component for a given <Route>: component, render, and children.

component should be used when you have an existing component (either a React.Component or a stateless functional component) that we want to render.

Render, which takes an inline function, should only be used when you have to pass in-scope variables to the component we want to render.

We should not use the component prop with an inline function to pass in-scope variables because you will get undesired component unmounts/remounts.

Instructor Notes:

Routing related Props

You want to render a Dashboard component, whenever a user navigates to a /dashboard path, you can render by using code below:

```
<Route path='/dashboard' component={Dashboard} />
```

Consider you wanted to pass the Dashboard component prop also along with it.

Then you can achieve it by using the inline function which creates an React element.

```
<Route
  path='/dashboard'
  component={() => <Dashboard isAuthed={true} />}
/>
```

Instructor Notes:

Routing related Props

Using inline function you will be able to pass the props, but it is not the recommended way because of Performance Issue.

When you use component, the router uses React.createElement to create a new React element from the given component.

That means if you provide an inline function to the component attribute, you will create a new component every render.

This results in the existing component unmounting and the new component mounting instead of just updating the existing component.

There is one more way available which you can use to pass the props with the component while routing.

Instructor Notes:

Routing related Props

You can use the render prop, instead of using the component prop.

render accepts a functional component, and that function won't get unnecessarily remounted like with component.

That function will also receive all the same props that component would receive so you can pass those through to the rendered component.

```
<Route
  path='/dashboard'
  render={(props) => (
    <Dashboard {...props} isAuthed={true} />
  )}
/>
```

With render, you're in charge of creating the element and can pass the component any props you'd like

Instructor Notes:

The "withRouter" HOC & Route Props

You can get access to the history object's properties and the closest <Route>'s match via the withRouter higher-order component. withRouter will pass updated match, location, and history props to the wrapped component whenever it renders.

```
const ShowTheLocationWithRouter = withRouter(ShowTheLocation);
```

withRouter does not subscribe to location changes but re-renders after location changes propagate out from the <Router> component. This means that withRouter does *not* re-render on route transitions unless its parent component re-renders.

If we are using withRouter to prevent updates from being blocked by shouldComponentUpdate, it is important that withRouter wraps the component that implements shouldComponentUpdate.

To use in program , we have to import as below

```
import { withRouter } from 'react-router-dom';
```

When you include a main page component in your app, it is often wrapped in a <Route> component like this:
<Route path="/movies" component={Movies} />

Because of the above code snippet

Movies component has access to this.props.history so it can redirect the user with this.props.history.push.

Some components (commonly a header component) appear on every page, so are not wrapped in a <Route>:
render() { return (<Header />); }This means the header cannot redirect the user.

To solve this the header component can be wrapped in a withRouter function, either when it is exported:

export default withRouter(Header)

Thus, Header component access to this.props.history, which means the header can now redirect the user.

Instructor Notes:

Passing & extracting route/query parameters

When you're building for the web, if you want to pass information via the URL, we use Query Strings.

Given below is an example of it, where movieName and rating are query part of query string.

http://www.url.com/user/Prabu/post?theatreName=Devi&rating=4.5

Route Parameter Query Strings

React Router does not have any opinions about how we parse URL query strings.

Some applications use simple key=value query strings, but others embed arrays and objects in the query string.

So it's up to us to parse the search string yourself.

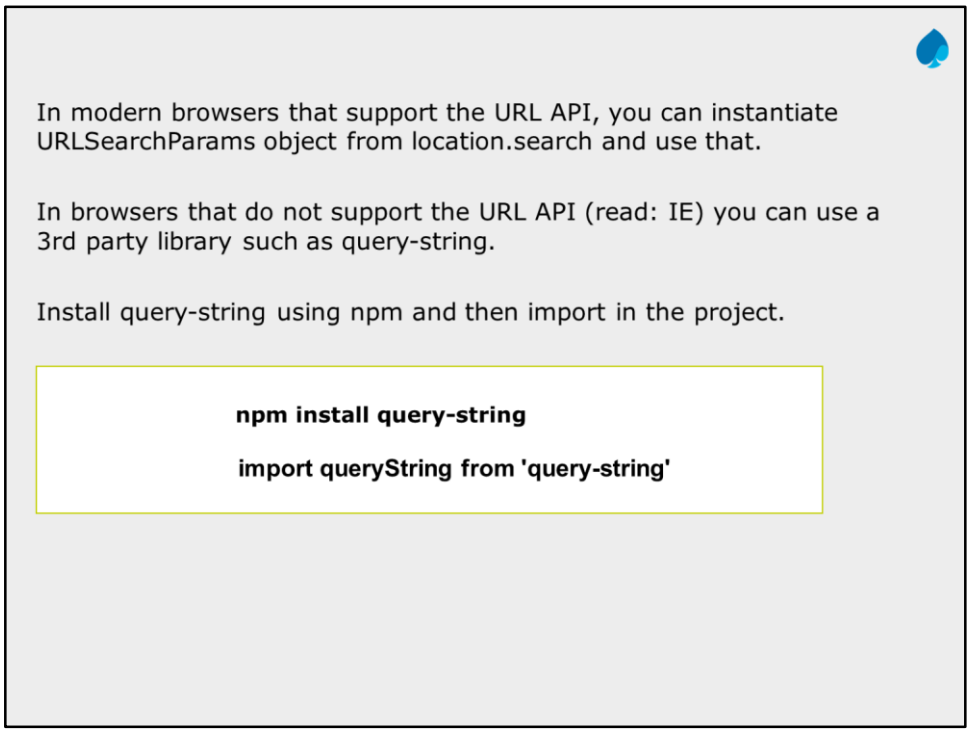
We can get the value using componentDidMount method as shown below

componentDidMount() { const values = queryString.parse(this.props.location.search) console.log(values.filter) // "top" console.log(values.origin) // "im" }

Without using query string also we can do by using below code snippet

// let params = new URLSearchParams(location.search);
//return <Child name={params.get("name")} />;

Instructor Notes:



Instructor Notes:

Add instructor notes here.

Demo

React Router props

react-router-props
React-route-withrouter (need to do)
react-route-query-param

Add the notes here.

Instructor Notes:

Using Switch to Load a Single Route

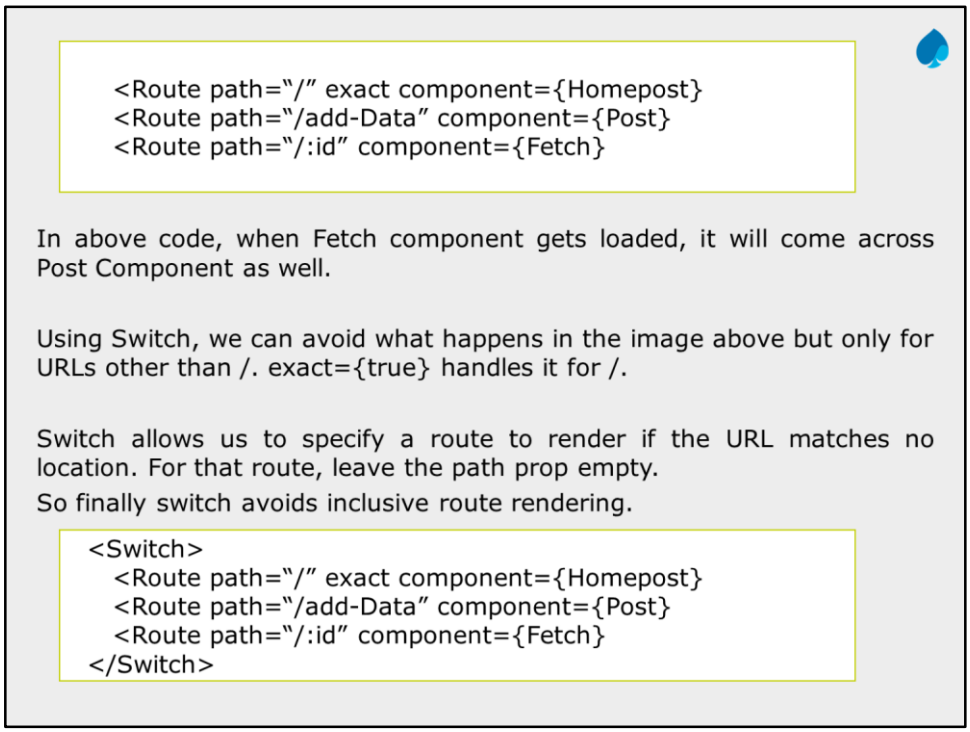
<Switch> returns only one first matching route and this will be wrapped around Route components.

Earlier we had an issue with / rendering the Home component and that of other paths.

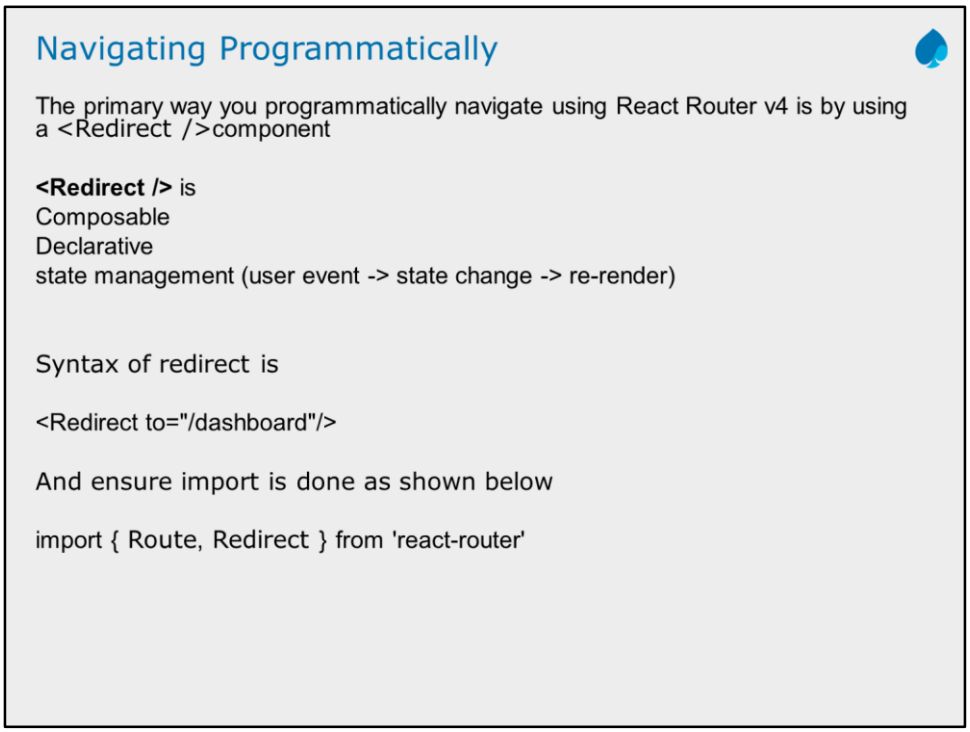
If we create a /welcome/movies path, what happens is, the components of the /welcome and /welcome/movies path will be rendered.

This is where Switch will help in this case again by choosing only one route to render, but the most important route must be arranged to come first.

Instructor Notes:



Instructor Notes:



React Router is mostly a wrapper around the history library. history handles interaction with the browser's window.history for you with its browser and hash histories. It also provides a memory history which is useful for environments that don't have a global history
A history instance has two methods for navigating: push and replace. If you think of the history as an array of visited locations, push will add a new location to the array and replace will replace the current location in the array with the new one. Typically you will want to use the pushmethod when you are navigating.
In earlier versions of React Router, you had to create your own history instance, but in v4 the <BrowserRouter>, <HashRouter>, and <MemoryRouter> components will create a browser, hash, and memory instances for you. React Router makes the properties and methods of the history instance associated with your router available through the context, under the router object.

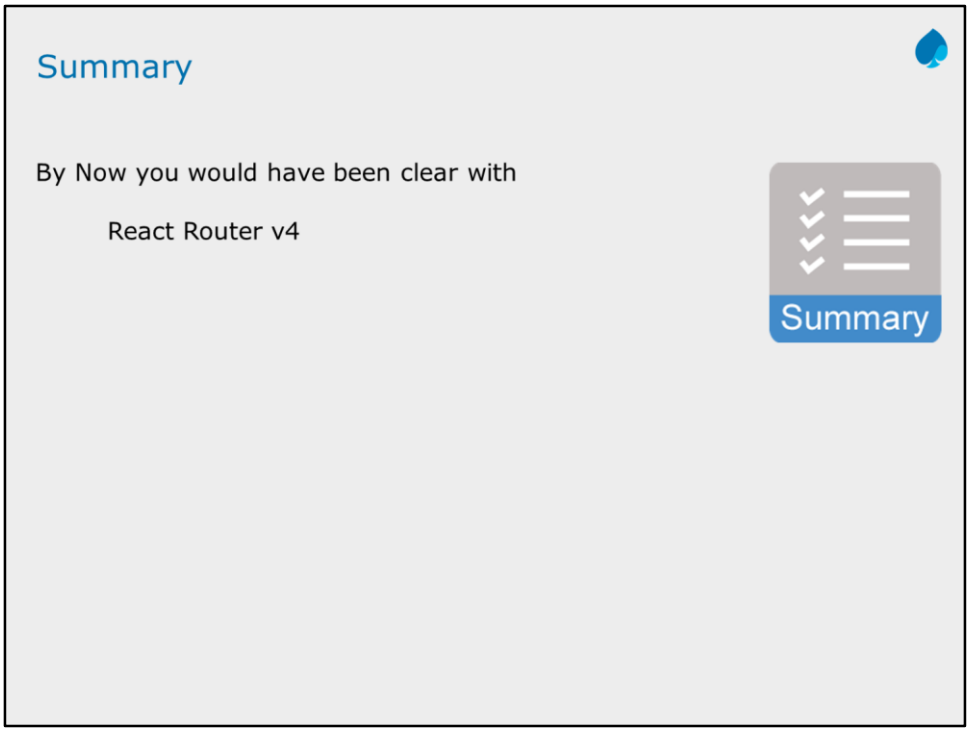
Instructor Notes:

Using Switch to Load a Single Route

react-route-switch-demo
react-route-program-programmatically

Instructor Notes:

Add instructor notes here.



Add the notes here.

Instructor Notes:

Questions

How will you import route and router from 'react-router' library

1. import {Router, Route} from 'react-router';
2. import Router, Route from 'react-router';
3. import Router, Route from react-router;
4. import all from 'react-router';