
QAREACTJS
Single Page Applications using React-Router
Developing Applications using ReactJS
QA

Objectives

- To understand how React Router is used to create single page applications
- To be able to define and use Routes to create single page applications
- To be able to use params to create multi-layered single page applications

Single Page Applications

- **So far only looked at discrete components and pages**
- **Many modern applications are based on a single page**
  - Content within the page changes depending on user input
  - Increases speed of web applications
    - Only content to be changed is affected instead of whole page reloads
- **React-Router is standard routing library to allow this behaviour**
  - Entry JavaScript file is configured to describe how navigation to different components should be handled
  - From the 'docs':

React Router is a powerful routing library built on top of React that helps you add new screens and flows to your application incredibly quickly, all while keeping the URL in sync with what's being displayed on the page.

Dynamic Routing v Static Routing

- **Most frameworks and libraries use static routing**
  - Routes declared as part of initialisation of app before rendering
  - Client side routers need routes to be declared upfront (such as in Angular)

- **From React Router v4, DYNAMIC ROUTING is used**
  - Routing takes place as app is rendering
    - Whole application is wrapped in a Router
    - Links defined within components
    - Route used to define which component should be rendered for the path
      - Route is just a component!
  - Routing thought of as UI not static configuration

Routers

- **react-router-dom package has `<Router>` with 5 variations on the common low-level interface**
- **Typically high-level router used in app**
  - `<BrowserRouter>`
    - Uses HTML5 history API to keep UI in sync with URL
  - `<HashRouter>`
    - Uses hash portion of URL to keep UI in sync
  - `<MemoryRouter>`
    - Keeps history of URL in memory (useful in testing and React Native)
  - `<StaticRouter>`
    - Never changes location – useful in server-side rendering
- **`<Router>` used to synchronise custom history with state management libraries (eg. Redux)**

For API information see: https://reacttraining.com/react-router/web/api/Router

Route

- **Most important to understand and learn to use**

```
<Route path render_method options />
```

- **Basic responsibility is to render a UI when a location matches route's `path`**
  - `path` – specifies the URL
- **3 ways to render from a `<Route>`:**
  - `component` – specified component will be created and rendered
  - `render` – allows for convenient inline rendering and wrapping
  - `children` – used to render whether the path matches the location or not – useful for animations
- **Other parameters:**
  - `exact` – match only if path matches location.pathname exactly
  - `strict` – Boolean to represent if match should be made if trailing slash is present
  - `object` – match path to passed location object's history (current by default)

130

For API information see: https://reacttraining.com/react-router/web/api/Route

Linking Views

- **`<Link to=path options>` component replaces `<a href=path>`**
  - Provides declarative, accessible navigation around application
  - **`path`** argument can be a string or an object
    - String is pathname or location to link to
    - Object is location to link to
  - Options: **`replace`** – boolean to indicate whether to replace history item instead of adding new
- **`<NavLink>` allows for active classes to be applied to the link**
  - Has additional options:
    - **`activeClassName`** – CSS class to be applied when active
    - **`activeStyle`** – can supply CSS-style object as alternate to class
    - **`exact`** – Boolean to represent that active should only be applied if path match is exact
    - **`strict`** – Boolean to represent trailing slash matching
    - **`isActive`** – function to add extra logic to determine if link is active
    - **`location`** – object to be used for comparing locations
- **`<Redirect>` allows navigation to a new location overriding current location in history stack**

131

For API information see: https://reacttraining.com/react-router/web/api/Link

Matching Paths

- **Clicking on a Route generates a match object containing information about how the path matched the URL**
- **Object contains:**
  - `params` – Object with key/value pairs parsed from the URL corresponding to dynamic segments of the path
  - `isExact` – Boolean representing if entire URL was matched
  - `path` – String path pattern used to match (useful for nested routes)
  - `url` – String representing matched portion of URL (useful for nested links)
- **Object can be accessed in many places in application**
- **`<Switch>` renders first child `<Route>` or `<Redirect>` that matches location**
  - Unique as it renders route exclusively
  - Every `<Route>` matching location renders inclusively (if not wrapped in a `<Switch>`)

132

For API information see: https://reacttraining.com/react-router/web/api/Swtich

Adding the Router to the application

- To use React-Router it has to be downloaded and added to the project dependencies
- This can be done using npm:

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

  - `--save` adds `react-router-dom` to `package.json`
- In most applications the following components of React Router could be used:
  - `Router`
  - `Route`
  - `Link` or `NavLink`
  - `Switch`

```
import { Router, Route, Link, NavLink, Switch } from 'react-
router-dom';
```

  - Note: Only import into JSX files where they are used

133

Defining and Linking to Routes

- **Wrap the entire application in a <Router> component**
  - <BrowserRouter>, <HashRouter>, etc
  - Either at start of root component or in ReactDOM.render
- **Provide <Link> or <NavLink> components to create hyperlinks in the appropriate components**

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

- **Provide <Route> information (wrapped in a <Switch> if required) in the appropriate components**

```
<Switch>
   <Route path="/" component={App} />
   <Route path="/subContent1" component={SubComponent1} />
   <Route path="/subContent2" component={SubComponent2} />
</Switch>
```

Params

- Links to parameterised paths can be defined by hard coding or by supplying an expression, perhaps based on an id property of an object

```
<Link to="/content/subContent1" component={SubContent} />
<Link to="/content/${subContent.id}" component={SubConent} />
```

- To define a route to a parameterised property, colon notation is used followed by the name of the parameter to use

```
<Route path="/content/:subContentId" component={SubContent} />
```

- To use the parameter in the matched URL, props need to be passed to the rendering component and the match object accessed

```
const SubContent = (props) => (
   <h1>props.match.params.subContentId</h1>
);
```

Objectives

- To understand how React Router is used to create single page applications
- To be able to define and use Routes to create single page applications
- To be able to use params to create multi-layered single page applications

Exercise Time

- Use React Router to create a simple single page application