SPA & Routing

CS568 – Web Application Development I

Assistant Professor Umur Inan

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- SPA
- React Router
    - BrowserRouter (Router)
    - Link
    - Redirect
    - Route
    - withRouter HOC
    - Passing Route Parameter
    - Switch

SPA (Single-page application)

A single-page application (SPA) is a web application or website that interacts with the user by **dynamically rewriting** the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages. The goal is **faster transitions**.

Some of the most popular SPA frameworks include:

- React
- Angular
- Vue.JS

SPA (Single-page application)

In a SPA, a page refresh never occurs; instead, all necessary HTML, JavaScript, and CSS code is either

- retrieved by the browser with a single page load
- or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions. (Lazy loading)

The HTML5 History API can be used to provide the perception and navigability of separate logical pages in the application

History API

The DOM Window object provides access to the browser's session through the history object. It exposes useful methods and properties that let you navigate back and forth through the user's history, and manipulate the contents of the history stack.

Some useful methods:

- window.history.back() or window.history.go(-1)
- window.history.forward() or window.history.go(1)
- window.history.go(0) or window.history.go() = refreshes the page

Routing & SPA

- Being able to show different pages to the user.
- SPA has a single HTML file. We do not have multiple HTML files.
- We need routing for loading different components for different path. For example, you want to bookmark a url.

React Router

React Router is a collection of navigational components.

The official documentation is pretty good. There are examples and you can play with it. https://reactrouter.com/web/guides/quick-start

React Package

- Parses URL / Path.
- Reads config.
- Renders / Loads appropriate JSX / Component.
- npm install --save react-router react-router-dom

<BrowserRouter>

A <Router> that uses the HTML5 history API (pushState, replaceState and the popstate event) to keep your UI in sync with the URL.

```
class App extends Component {
 render() {
   return (
     <BrowserRouter>
       <div className="App">
         <Blog />
       </div>
     </BrowserRouter>
   );
 }
}
```

Setting up Links

```
class Blog extends Component {
  render() {
    return (
    <div>
      <header>
        <nav>
          <ul>
            <li><a href='/'>Home</a></li>
            <li><a href='/new-post'>New Post</a></li>
          ...
```

<Link>

When we use <a> tag, it reloads the app. That is not what we want. We will use ‘Link’ to prevent that behaviour.

<Link> Provides declarative, accessible navigation around your single page application.

Links example

```
class Blog extends Component {
  render() {
    return (
    <div>
      <header>
        <nav>
          <ul>
            <li><Link to='/'>Home</Link></li>
            <li><Link to='/new-post'>New Post</Link></li>
          ...
```

<Redirect>

<Redirect> will navigate to a new location. The new location will override the current location in the history stack, like server-side redirects (HTTP 3xx) do.

```
<Route exact path="/">
  {loggedIn ? <Redirect to="/dashboard" /> : <Home />}
</Route>
```

<Route>

The Route component is perhaps the most important component in React Router to understand and learn to use well. Its most basic responsibility is to render some UI when its path matches the current URL.

Route props:

1. **Match**
2. Location
3. History

The match prop of the Route

A match object contains information about how a <Route path> matched the URL. match objects contain the following properties:

- params - (object) Key/value pairs parsed from the URL corresponding to the dynamic segments of the path
- isExact - (boolean) true if the entire URL was matched (no trailing characters)
- path - (string) The path pattern used to match. Useful for building nested <Route>s
- url - (string) The matched portion of the URL. Useful for building nested <Link>s

```
{
  "path": "/user/:username",
  "url": "/user/unubold",
  "isExact": true,
  "params": {
    "username": "Unubold"
  }
}
```

Route example

```
class Blog extends Component {
  render() {
    return (
    <div>
      <header>
        <nav>
          <ul>
            <li><Link to='/'>Home</Link></li>
            <li><Link to='/new-post'>New Post</Link></li>
          ...
      <Route path='/' exact component={Posts} />
      <Route path='/new-post' exact component={NewPost} />
...
```

withRouter HOC

Router props will not be passed down to component tree. We can use withRouter HOC for that.

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.

withRouter HOC example

```
// A simple component that shows the pathname of the current location
class ShowTheLocation extends React.Component {
  render() {
    const { match, location, history } = this.props;
    return <div>You are now at {location.pathname}</div>;
  }
}

ReactDOM.render(
  <Router>
    <Route path="/" component={ withRouter(ShowTheLocation) }/>
  </Router>,
  document.getElementById('root')
);
```

withRouter HOC Blog example

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

const post = (props) => {
   return (
       <article className="Post" onClick={props.postClicked}>
           <h1>{props.title}</h1>
           <div className="Info">
               <div className="Author">{props.author}</div>
...

export default withRouter(post);
```

Passing Route Parameters

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

<Route path='/new-post' exact component={NewPost} />

<Route path='/:id' exact component={FullPost} />
```

Passing Route Parameters

```
render() {
  const posts = this.state.posts.map(item => {
    return (<Link to={'/' + item.id} key={item.id}>
              <Post title={item.title} author={item.author}
                postClicked={() => { this.postClickedHandler(item.id) }} />
            </Link>)
  });
  return (
      <section className="Posts">
          {posts}
      </section>
  )
}
```

Extracting Route Parameters

```
componentDidMount() { //FullPost.js
  const reqId = this.props.match.params.id;
  const currentPost = this.state.post;
  if (reqId != 0) {
    if(!currentPost || (currentPost && currentPost.id !== reqId)) {
      axios.get('https://jsonplaceholder.typicode.com/posts/' + reqId)
        .then((response) => {
          this.setState({ post: response.data });
        });
  ...
```

Using Switch to Load Single Route

```
//blog.js
<Switch>
  <Route path='/' exact component={Posts} />
  <Route path='/new-post' exact component={NewPost} />
  <Route path='/:id' exact component={FullPost} />
</Switch>
```

<Switch>

<Switch> is unique in that it renders a route exclusively. In contrast, every <Route> that matches the location renders inclusively.

All <Route>s get rendered. But if you want to pick only one <Route> to render, then use Switch.

React Router Summarize

```
<Router>
  <div>
    <Link to="/">Home</Link>
    <Link to="/about">About</Link>
    <Link to="/users">Users</Link>

    <Switch>
      <Route path="/about" component={About}/>
      <Route path="/users" component={Users}/>
      <Route path="/" exact component={Home}/>
    </Switch>
  </div>
</Router>
```