

React Router

<https://reacttraining.com/react-router/web/guides/philosophy>

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

```
ReactDOM.render(  
  <BrowserRouter>  
    <HelloWorld />  
  </BrowserRouter>,  
  document.getElementById( 'react-root' )  
)
```

notice that we just wrap our app in a
<BrowserRouter> component to enable routing.

```
<Route path="/about" component={About} />
```

when the URL matches /about, render the
About component

```
<Link to="/about">Click me</Link>
```

when the user clicks this link, take them to the about page.

let's get our first routes going :)

```
npm run exercise router 1
```

**matching the index
page**

← → ↻ 🏠 ⓘ localhost:1234/about

this isn't quite what we want

My great app!

[Click here to view the about page](#)

this is the home page

this is the about page

```
const Home = () => {
  return <p>this is the home page</p>
}

const HelloWorld = () => {
  return (
    <div>
      <h1>My great app!</h1>
      <Link to="/about">Click here to view the about page</Link>
      <Route path="/" component={Home} />
      <Route path="/about" component={About} />
    </div>
  )
}
```


exact paths to the rescue

```
<Route path="/" component={Home} />
```

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

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

```
npm run exercise router 2
```

NavLink

NavLink

a link that has a css class applied to it when it's active.

Use NavLink

```
<NavLink activeClassName="active-link" to="/about">  
  Click here to view the about page  
</NavLink>
```

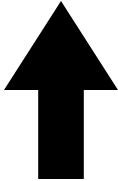
```
npm run exercise router 3
```

**dynamic URL
segments**

/user/jack

/user/alice

```
const HelloWorld = () => {  
  return (  
    <div>  
      <h1>My great app!</h1>  
      <Route path="/user/:name" component={UserPage} />  
    </div>  
  )  
}
```



note the :name - this marks it as a dynamic
segment of the URL

make the user page output the name of the user

```
const UserPage = props => {  
  return <p>this is the user page for {props.match.params.name}</p>  
}
```

```
npm run exercise routing 4
```


making API requests

/posts/1

fetch a post based on the URL parameter

hint: `this.match.params`

```
npm run exercise router 5
```

building a blog

```
class Posts extends Component {
  state = { posts: [] }

  componentDidMount() {
    fetch('https://jsonplaceholder.typicode.com/posts?_limit=5').then(
      response => {
        this.setState({ posts: response.data })
      }
    )
  }

  render() {
    return (
      <div>
        <ul>
          {this.state.posts.map(post => {
            return (
              <li key={post.id}>
                <NavLink to={`/${post.id}`}>{post.title}</NavLink>
              </li>
            )
          })}
        </ul>

        <Route path="/posts/:id" component={PostPage} />
      </div>
    )
  }
}
```

find the bug (hint:
think about
yesterday's lifecycle
exercises)

```
npm run exercise routing 6
```

**404s and matching
only one route**

```
<Route path="/foo" ... />
```

```
<Route path="/foo" ... />
```

```
<Route path="/foo" ... />
```

```
<Route path="/foo" ... />
```

by default all of these will match if I'm on
the /foo URL

but what if you want a route that is for
your 404 page?

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

a route without a path always matches
every URL

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

but we only want to render that if no other
route has matched

```
<Switch>  
  <Route path="/about" component  
  <Route component={NotFound} />  
</Switch>
```

but we only want to render that if no other
route has matched

use a `<Switch />` to
fix the 404 page
rendering

```
npm run exercise router 7
```

GraphQL

GraphQL

One endpoint that we
make a request to,
describing the data we
want.

// BEFORE:

```
fetch( '/user' ).then(user => {  
  fetch( `/posts?user_id=${user.id}` )  
})
```

REST APIs aren't always great when you
need to get different resources for a single
user


```
// AFTER:
```

```
makeGraphQLRequest(`  
  query {  
    user(id: 1) {  
      name,  
      posts {  
        id,  
        title  
      }  
    }  
  }  
`)
```

in GraphQL we give a query to the API that describes exactly what we want back in one request.

<https://fake-graphql.tech/graphql>

GraphiQL

Secure <https://fake-graphql.tech/graphql>

☆

GraphiQL

Prettify

History

```
1 # Welcome to GraphiQL
2 #
3 # GraphiQL is an in-browser tool for writing, validating, and
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and you will see intelligent
7 # typeheads aware of the current GraphQL type schema and live syntax and
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" character. Lines that starts
11 # with a # are ignored.
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query:  Shift-Ctrl-P (or press the prettify button above)
24 #
25 # Run Query:      Ctrl-Enter (or press the play button above)
26 #
27 # Auto Complete:  Ctrl-Space (or just start typing)
28 #
29
30
```

QUERY VARIABLES

< Schema

Query

×

Q Search Query...

...

FIELDS

person(id: Int): Person

people: [Person]

albums: [Album]

addresses: [Address]

photos: [Photo]

```
query {  
  person(id: 1) {  
    name  
  }  
}
```



```
{  
  "data": {  
    "person": {  
      "name": "Christine Crooks"  
    }  
  }  
}
```

[← Query](#)

Person



A fake person

FIELDS

id: ID

name: String

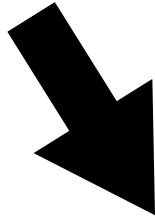
email: String

avatar: String

albums: [Album]

address: Address

```
query {  
  person(id:4) {  
    albums {  
      title  
    }  
  }  
}
```



```
{  
  "data": {  
    "person": {  
      "albums": [  
        {  
          "title": "Quia dolores dignissimos non esse nam corporis."  
        },  
        {  
          "title": "Voluptatem amet quisquam quae deserunt veniam nulla sapiente o  
        },  
        {  
          "title": "Nihil odio neque excepturi."  
        }  
      ]  
    }  
  }  
}
```

<https://github.com/prisma/graphql-request>

fetching a person with graphql

```
npm run exercise router 8
```


That's routing!

Any questions?