# Pass props to React Router's Link component

Often when building an app with React Router you’ll need to pass props through a Link component to the new route.

There are two different ways to pass data from a Link component through to the new route that’s being rendered. The first is through URL Parameters and the second is through state.

First, let’s take a look at URL parameters. If you’ve read our [URL Parameters](https://tylermcginnis.com/react-router-url-parameters/) post, you’ll be familiar with this example. Say we were in charge of building out the Route that renders Twitter’s [profile](https://twitter.com/tylermcginnis) page. If created with React Router, that route may look like this.

<Route path='/:handle' component={Profile} />

Notice that handle is going to be dynamic. It could be anything from tylermcginnis or dan\_abramov to realDonaldTrump.

So in our app, we may have a Link component that looks like this.

<Link to='/tylermcginnis'>Tyler McGinnis</Link>

If clicked, the user would be taken to /tylermcginnis and the Profile component would be able to access the dynamic handle (tylermcginnis) from props.match.params.handle.

class Profile extends React.Component {

state = {

user: null

}

componentDidMount () {

const { handle } = this.props.match.params

fetch(`https://api.twitter.com/user/${handle}`)

.then((user) => {

this.setState(() => ({ user }))

})

}

render() {

...

}

}

URL parameters are great, but they’re not really meant to serve as a way to get data from one route to another as they’re limited to just strings. What if instead of just a string, we wanted to pass something a little more complex, like an object or an array? There would be no way to do that with URL parameters. This brings us to the second way to pass data from one route to another and that’s with state.

Going back to our example from earlier, what if we wanted to pass along if the user is coming from a certain route through to the Profile component when the user clicks on the Link? React Router gives us a way to do that and the API looks like this.

<Link to={{

pathname: '/tylermcginnis',

state: {

fromNotifications: true

}

}}>Tyler McGinnis</Link>

Now, the component that’s being rendered for that route (in this case, Profile) would be able to access fromNotifications by accessing props.location.state.

class Profile extends React.Component {

state = {

user: null

}

componentDidMount () {

const { handle } = this.props.match.params

const { fromNotifications } = this.props.location.state

fetch(`https://api.twitter.com/user/${handle}`)

.then((user) => {

this.setState(() => ({ user }))

})

}

render() {

...

}

}

To recap, there are two ways to pass data from a Link through to the new route: URL parameters and state. URL parameters work great for strings, but break down after that. By making the Links to prop an object, you can pass along any sort of data you need under the state property and that data can be accessed in the new route under props.location.state.



Pass props to a component rendered by React Router

React Router uses a declarative, component-based approach to routing. What that means is when you want to create a new route, you render a Route component. If you’re familiar with React Router you know that to do that, you pass Route a path and a component to render when the app’s location matches that path.

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

Now, what if we also wanted to pass Dashboard a prop? There are a few different ways to solve this problem but only one right way. The first idea you might have is to just pass it as a prop on Route.

<Route

path='/dashboard'

component={Dashboard}

isAuthed={true}

/>

Unfortunately, this won’t work. React Router won’t forward that prop onto the component. Instead, it will just ignore it.

The next idea you might have, and a pattern I’ve seen a few places is to pass component an inline function that creates the element.

<Route

path='/dashboard'

component={() => <Dashboard isAuthed={true} />}

/>

Though technically this will work, it’s not the best solution. The reason for this is because of performance. According to the official docs…

*“When you use the component props, 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 would 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.”*

So if you’re not supposed to pass a function to component, what’s the solution? Turns out the React Router team predicted this problem and gave us a handy solution. Instead of using component, use the renderprop. 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 take those and pass those along to the rendered component.

<Route

path='/dashboard'

render={(props) => <Dashboard {...props} isAuthed={true} />}

/>

So to recap, if you need to pass a prop to a component being rendered by React Router, instead of using Routes component prop, use its render prop passing it an inline function then pass along the arguments to the element you’re creating.