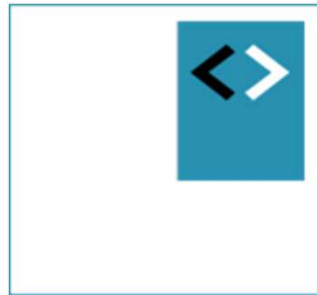


React Fundamentals

Module – React Router



Peter Kassenaar –
info@kassenaar.com



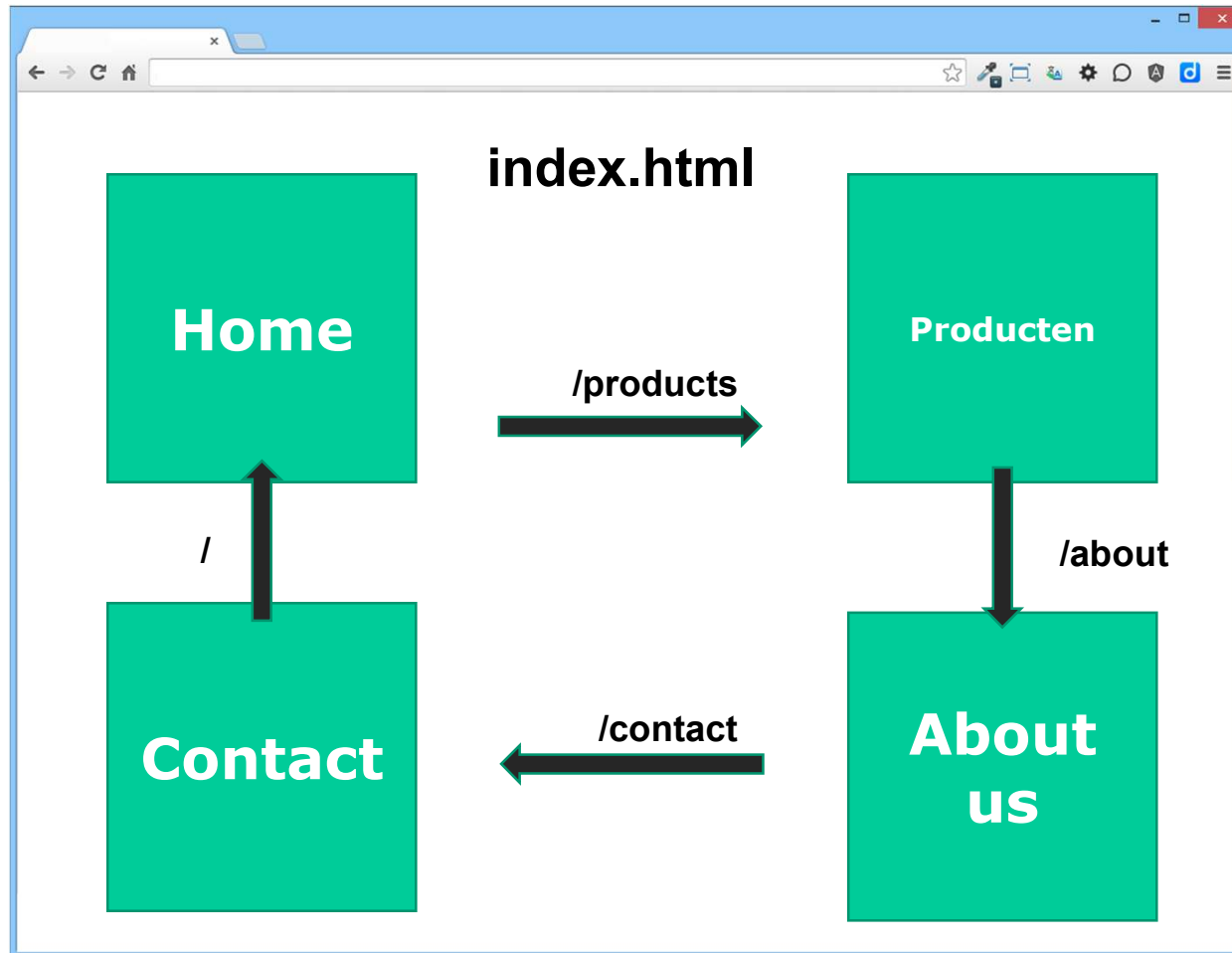
Routing in your application

Reflecting the state of your application in the URL

*"React Router is the **standard routing library** for React. React Router keeps your UI in sync with the URL. It has a simple API with powerful features like lazy loading, dynamic route matching and location transition."*

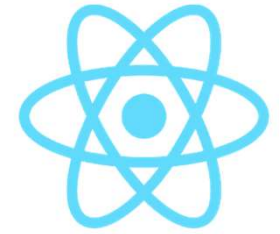
<https://www.freecodecamp.org/news/beginners-guide-to-react-router-4-8959ceb3ad58/>

Routing architecture and goal



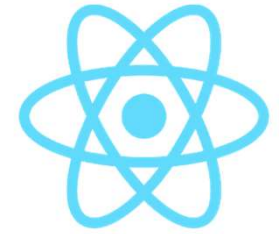
- Make use of SPA principle
- Making deep links possible

Router capabilities / characteristics

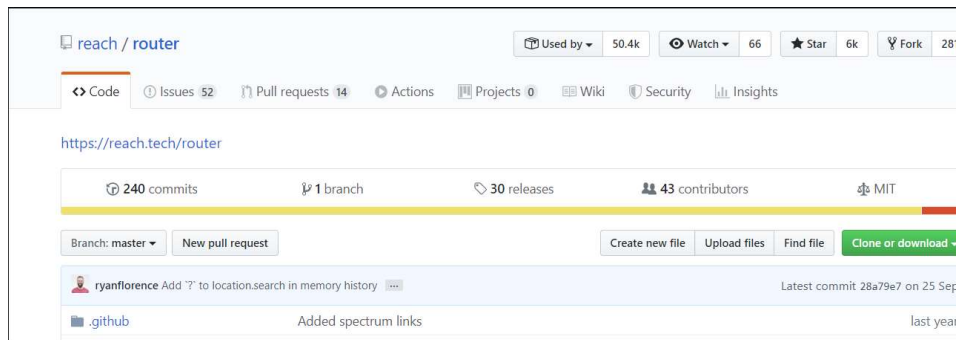


- **Update** URL when changing to routes/components
- **Nested** routes/view mapping
- **Modular**, component based router configuration
- Route **params**, query, wildcards
- View transition **effects**
- Link with automatic active CSS classes to denote **active route**
- HTML5 **history mode**
- Extensive **API**

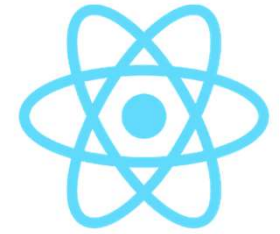
React Router



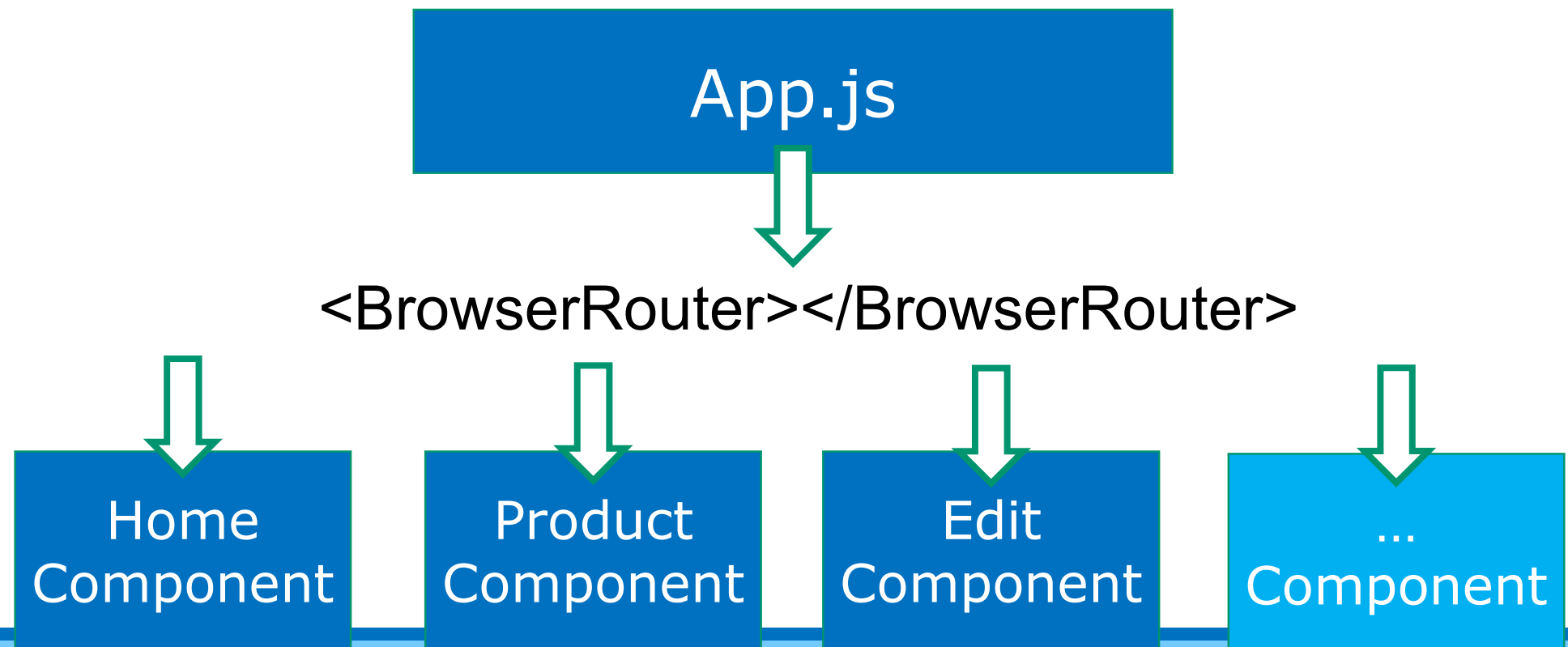
- Most popular choice: `react-router`
 - Created and maintained by team at reacttraining.com
- Also popular: Reach Router
 - Will be integrated with `react-router` in the near future
 - <https://github.com/reach/router>
 - Status – **don't use** for newer projects.



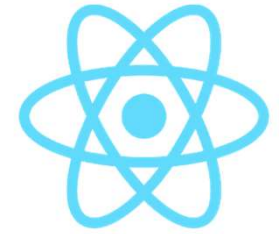
Routing – every route is a Component



- `App.js` gets a main menu (typically in its own component :-)
- Components are injected in `<BrowserRouter></BrowserRouter>`

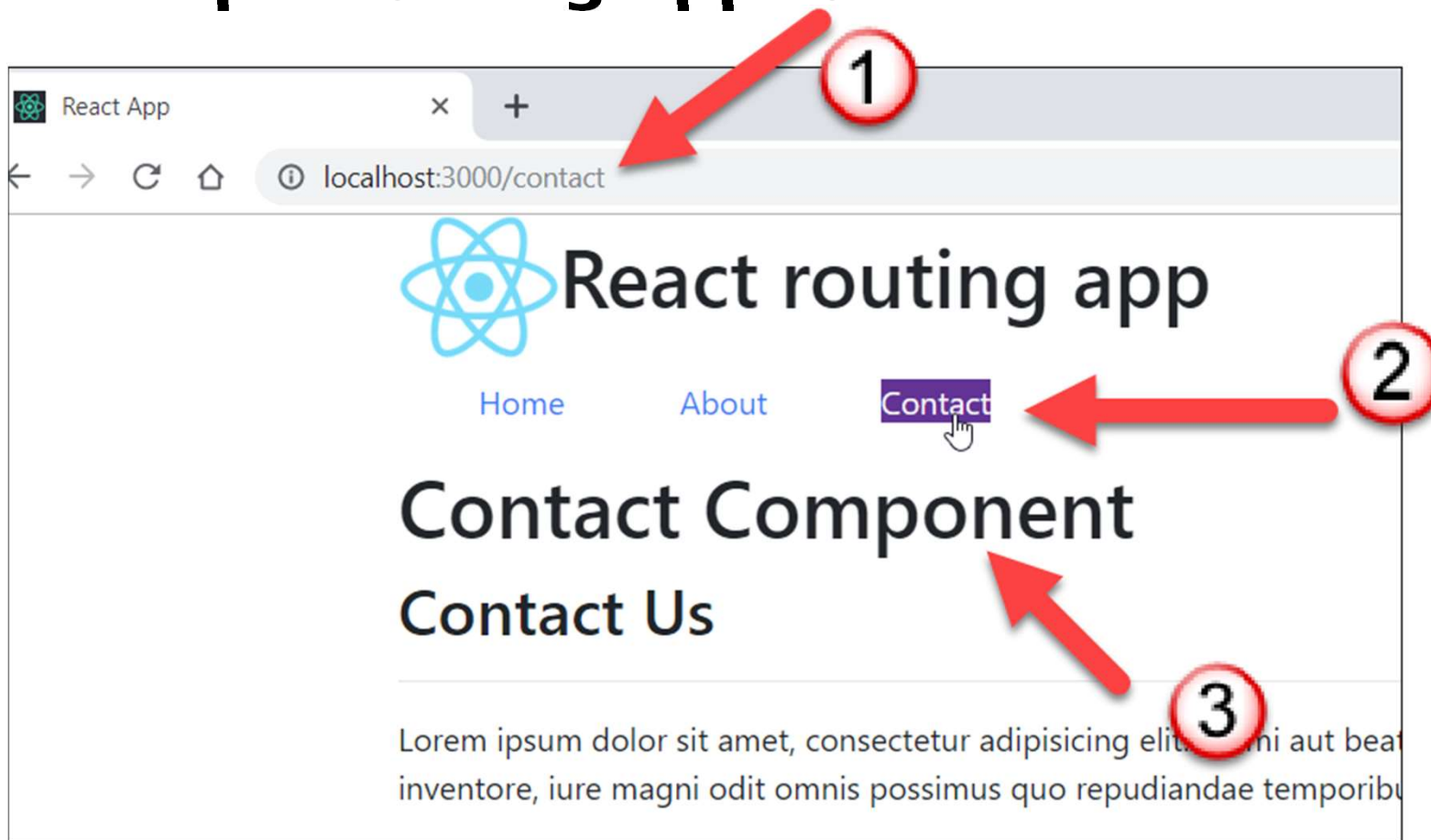
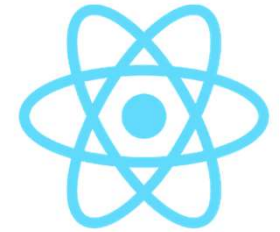


Routing in Components

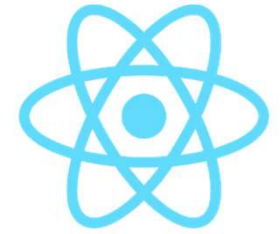


- Components can have **their own** router configs
 - Child routes
 - Nested routes
- They are loaded at runtime, when the component is loaded.
- **No central routing configuration** table like in Angular and Vue

What are we going to build – a simple routing app for starters



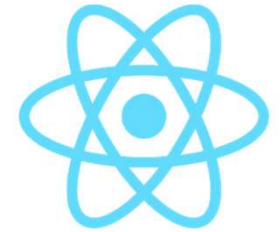
Steps in creating routing for your app



1. Install `react-router`
2. Create the main navigation
3. Create the basic routes
4. Create components, to be shown inside the routes
5. Update `App.js` with `<BrowserRouter>` to use the routes

*It is React, so **everything** is a component. Including*

*`<MainNavigation />` **and** `<Routes />`*



1. Install the router

- We're using a new project, created with CRA
- In a web environment, install three (3) libraries

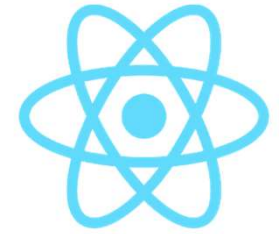
```
npm install react-router react-router-dom history
```

```
+ react-router-dom@5.1.2  
+ react-router@5.1.2  
+ history@4.10.1
```

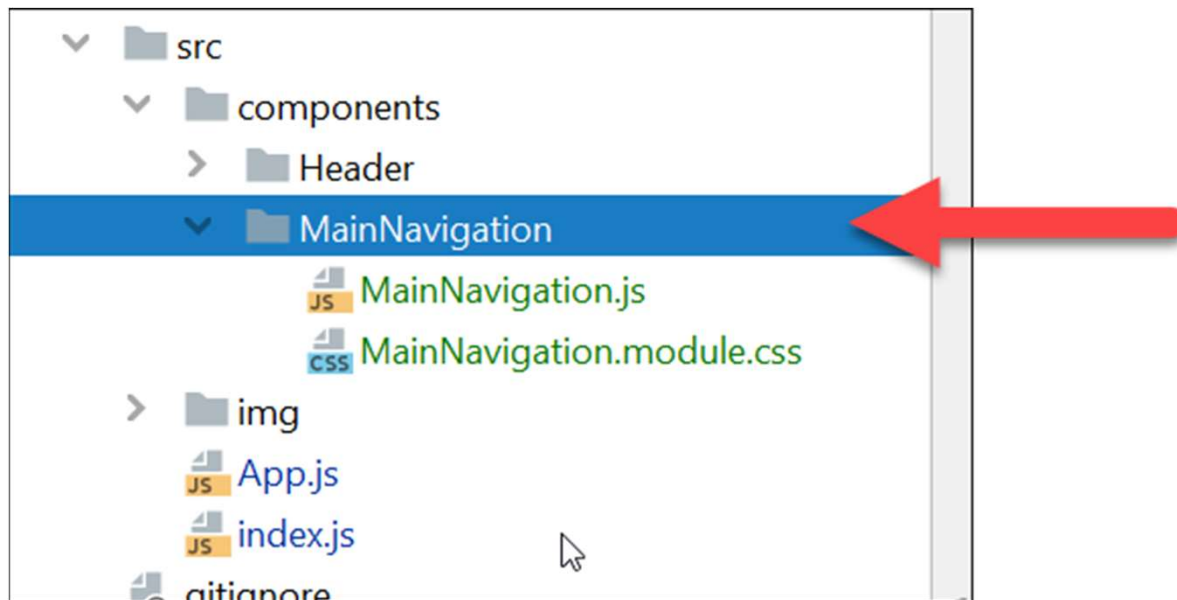
```
added 19 packages from 11 contributors and audited 92 packages in 1.365s  
found 0 vulnerabilities
```

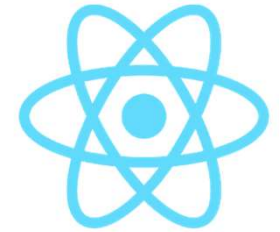
```
PS C:\Users\Gebruiker\Desktop\react-fundamentals>
```

2. Create the main navigation



- Choice: single component holding the main navigation
 - Also valid: having the navigation inside a component for nested/child routes
- Optional: having a CSS module for navigation styles





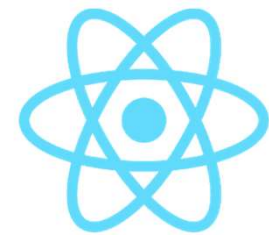
Using the `<Link />` element

```
import React, {Component} from 'react';
import {Link} from "react-router-dom";

class MainNavigation extends Component {
  render() {
    return (
      <div>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </div>
    );
  }
}

export default MainNavigation;
```

`<Link />` to navigate to routes



3. Create basic routes

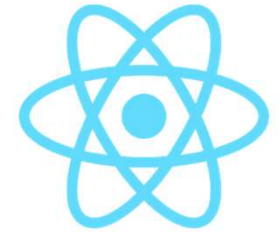
- New file: `App.routes.js`
 - Option 1 – inside a separate `./routes` folder
 - Option 2 – Routes inside component(s)
- Create a `<Routes />` Component

```
class Routes extends Component {  
  render() {  
    return (  
      <>  
        <Route path="/" component={Home}/>  
        <Route path="/contact" component={Contact}/>  
        <Route path="/about" component={About}/>  
        <Redirect to="/" />  
      </>  
    );  
  }  
}
```

Valid routes and
components to
load

Redirect, if none of
the routes is
matched. Always
last

4. Create Components

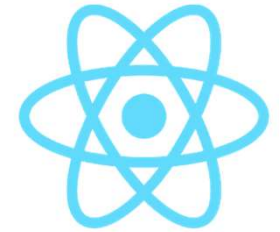


- We are using simple components w/ some text

```
class Home extends Component {  
  render() {  
    return (  
      <div>  
        <h1>Home Component</h1>  
        ...  
      </div>  
    )  
  }  
}
```

```
class About extends Component {  
  render() {  
    return (  
      <div>  
        <h1>About Component</h1>  
        <div>Our Mission</div>  
        ...  
      </div>  
    )  
  }  
}
```

```
class Contact extends Component {  
  render() {  
    return (  
      <div>  
        <h1>Contact Component</h1>  
        <div>Contact us:</div>  
        ...  
      </div>  
    )  
  }  
}
```



5. Update App.js

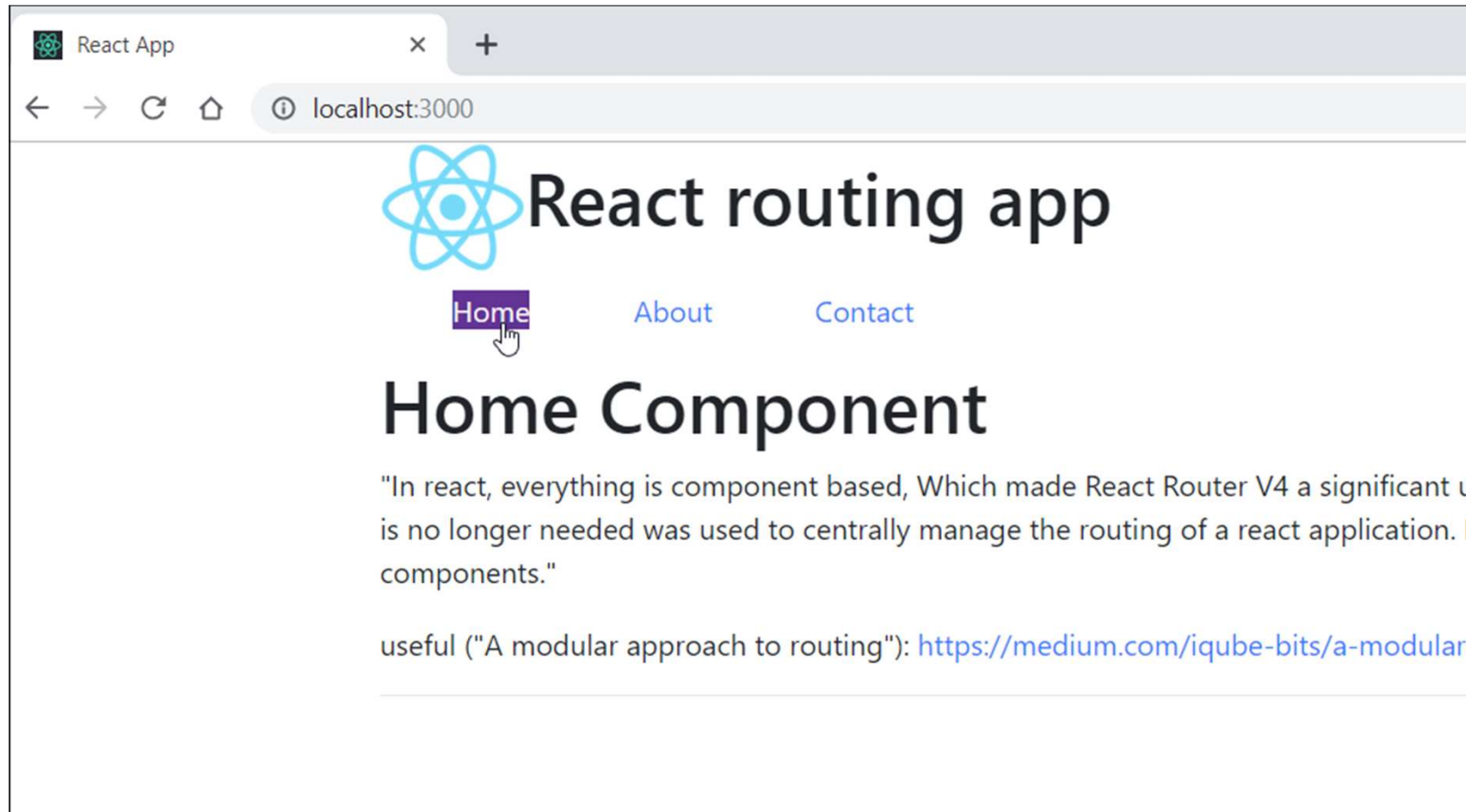
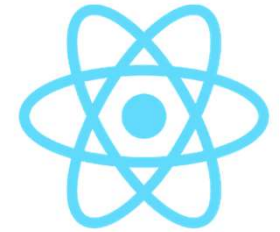
- Update `App.js` to use the routes.
- Note the usage of `<BrowserRouter>`
 - This is the *top level component* for every routing app
 - Remember to import from the correct libraries

```
function App() {  
  return (  
    <BrowserRouter>  
      <div className="container">  
        <Header/>  
        <MainNavigation/>  
        <Routes/>  
      </div>  
    </BrowserRouter>  
  );  
}
```



`<BrowserRouter>`

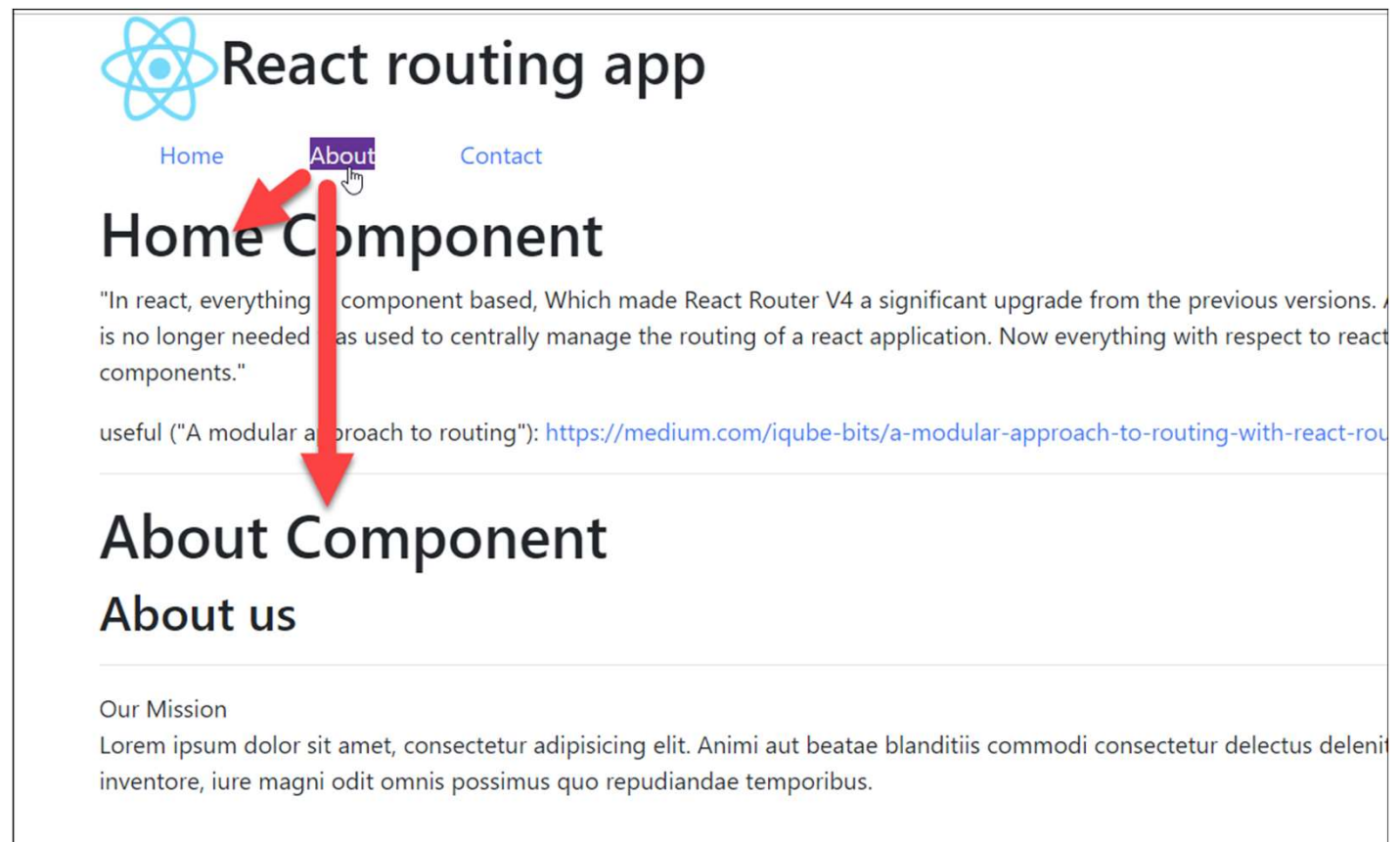
Result



But wait...



- HomeComponent is also shown when another route is active!



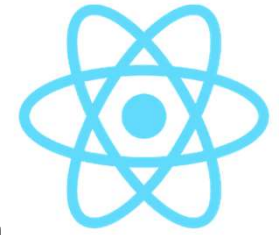
“Inclusive match”



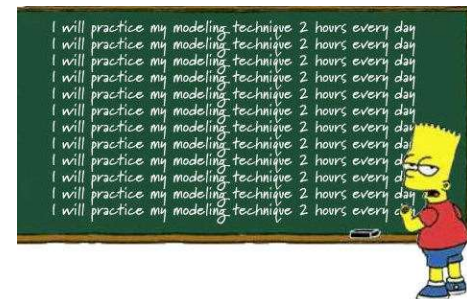
- This is b/c the router by default uses an *inclusive match*
 - The route `"/about"` also has the `"/` in it (which is the match for the `Home` route), so it shows both!
- Solution – use the `exact` keyword
 - Only exact matches of the route are shown

```
<Route path="/" exact component={Home}/>  
<Route path="/about" component={About}/>  
<Route path="/contact" component={Contact}/>
```

Workshop



- 1. Use your own app OR create a new app from scratch
- Add routing to it.
- Divide the components over different routes
- Maybe you are passing `props`. We're covering that later!
- 2. OR: Work from the example project
- Add a new component (`<Products />`) to it
- Add a `<Route>` and a `<Link>` to that new component, make sure it is accessible from the `<MainNavigation>`
- Example `../600-routing-basics`

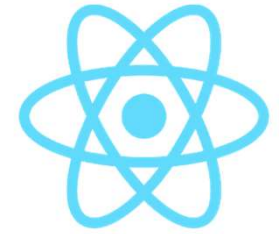




Styling the active link

Emphasizing the current route in the UI, based on a URL match

Special CSS class names



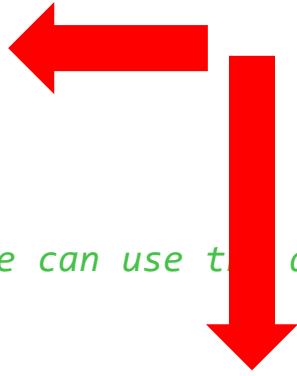
- When using `<NavLink>` instead of `<Link>`, the router assigns a special class `activeRoute` to the element
- You can write CSS for that
 - Global CSS or a CSS module
 - For instance a class `.activeRoute` (but you can name it any way you like)

```
.activeRoute {  
  border-bottom: 3px solid #09d3ac;  
}
```

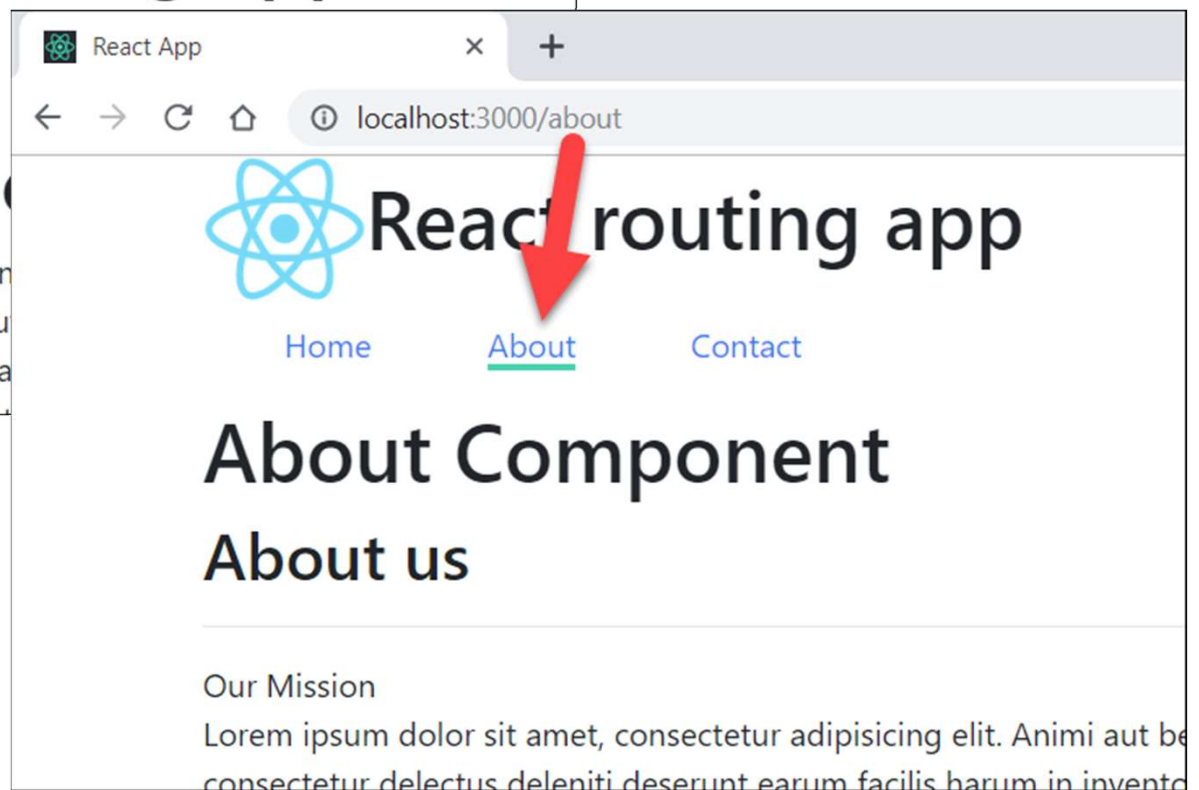
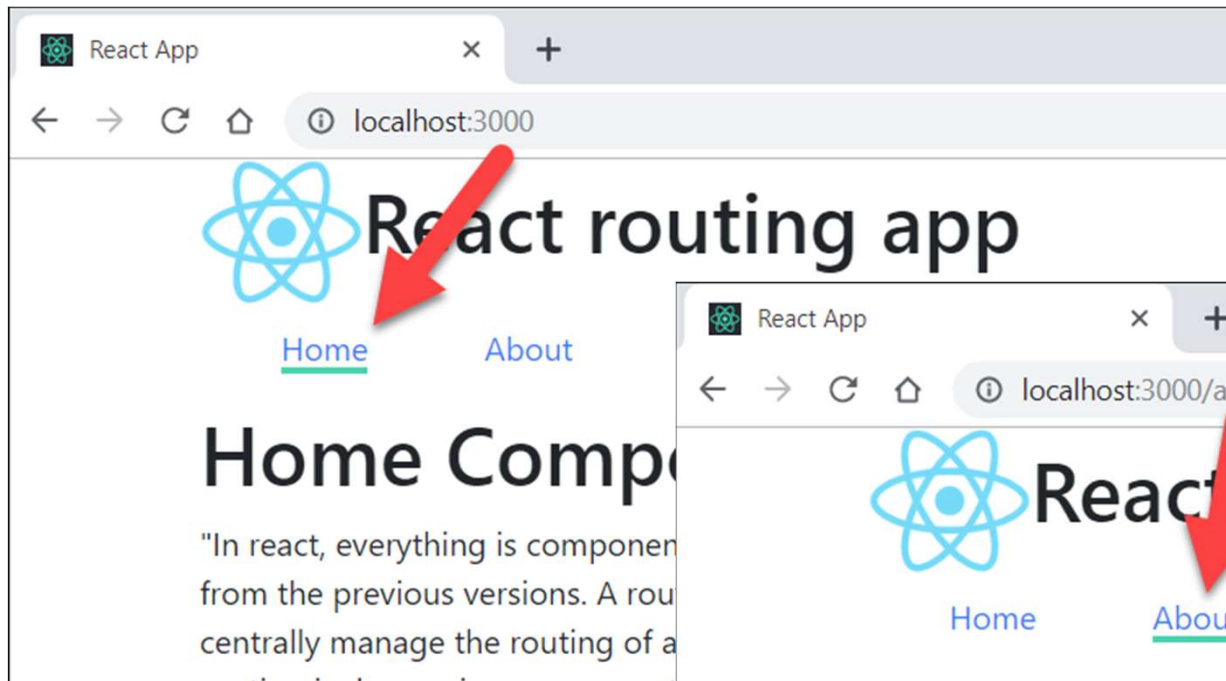
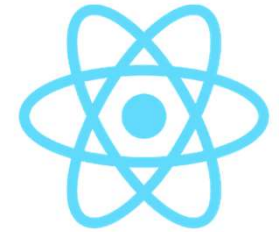
```
import React, {Component} from 'react';
import {NavLink} from "react-router-dom";
import styles from './MainNavigation.module.css'

class MainNavigation extends Component {
  render() {
    return (
      <nav>
        /*Another approach: using NavLink, we can use the active-link-class style*/
        <ul>
          <li>
            <NavLink exact activeClassName={styles.activeRoute} to="/">
              Home
            </NavLink>
          </li>
          <li>
            <NavLink exact activeClassName={styles.activeRoute} to="/about">
              About
            </NavLink>
          </li>
          ...
        </nav>
      )
    );
  }
}

export default MainNavigation;
```



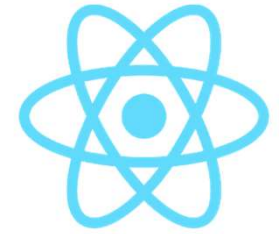
Result





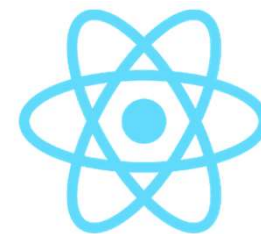
Navigating from code

Activating another Route from JavaScript



Navigating from JavaScript

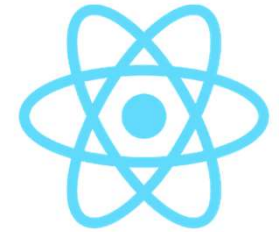
- Two options:
- Using `state` and `<Redirect />`
 - more code, but totally in line with React philosophy
- Using `History` API from the browser
 - Easy, but (maybe?) less elegant
- More info: <https://tylermcginnis.com/react-router-programmatically-navigate/>



*"When a component is rendered by
React Router, that component is
passed three different props:
location, match, **and** history"*

<https://tylermcginnis.com/react-router-programmatically-navigate/>

Using the History API



```
<button  
  className="btn btn-info"  
  onClick={() => this.navigate()}>  
  Navigate  
</button>
```

```
// programmatically navigate to another route  
navigate() {  
  this.props.history.push('/contact');  
}
```

About us

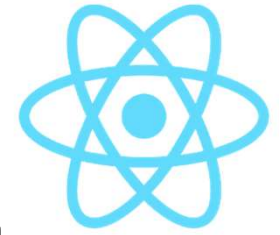
Our Mission

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Animi aut beatae blandit consectetur delectus deleniti deserunt earum facilis harum in inventore, iure magnam possimus quo repudiandae temporibus.

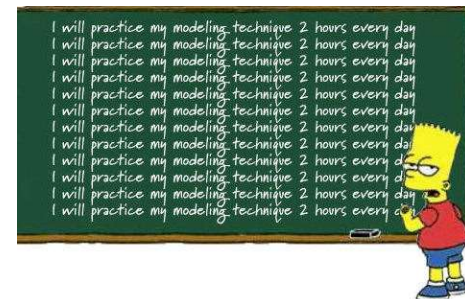
Navigate to Contact, using JavaScript



Workshop



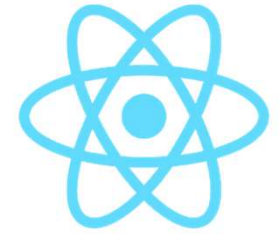
- **1. Use your own app** OR create a new app from scratch
- Create an 'active link' class and use it with `<NavLink />`
- Create a button and navigate from code using JavaScript
- **2. OR: Work from the example project**
- Give the Active route class a different styling and make sure it works. Also inspect the rendered HTML
- Create a button on the `Contact` component to navigate to the Home page.
- Example `../610-active-link-and-code`





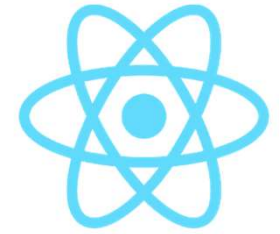
Using Route Parameters

Passing detail parameters to the next view



Route Parameters

- Creating Master/Detail Views
- Select a `country|product|employee|etc` on the first page and pass the selection to the next page
- Steps:
 1. Create/Update a route to send detail parameters
 2. Create/Update a DetailComponent to receive parameters
 3. Update `<Link>` or `<NavLink>` to send parameters
 4. Fetch Details and show them in the UI



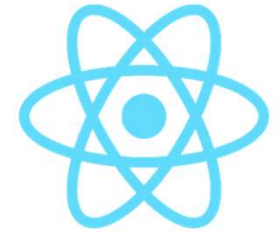
0. Prepare Home component

- Showing a list of countries
- Import `countryData.js` and prepare state as usual

```
import countryData from "../data/CountryData";

class Home extends Component {
  state = {
    countries: countryData.countries
  };
  ...
};
```

```
<ul className="list-group">
  {this.state.countries.map(country =>
    <li className="list-group-item"
      key={country.id}>
      {country.name}
    </li>
  )}
</ul>
```

1. Create a route to detail page

- Update `Routes.js` to show a new Detail Component
- Note the colon `/:id` and `/:name`. These are the dynamic parameters

```
<Route path="/" exact component={Home}/>  
...  
<Route path="/detail/:id/:name" component={CountryDetail}/>  
<Redirect to="/" />
```

2. Create detail component



```
class CountryDetail extends Component {  
  // We are passing in a country ID and name, so we need (at least)  
  // two state parameters.  
  state = {  
    id: null,  
    name: ''  
  };  
  ...  
};  
  
render() {  
  return (  
    <div>  
      <h3>CountryDetail works!</h3>  
      <p>Requested country id: {this.state.id}</p>  
      <p>Requested country name: {this.state.name}</p>  
    </div>  
  );  
}
```

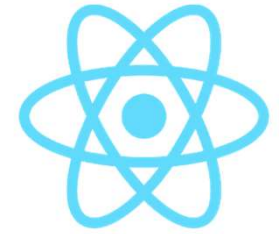
Retrieve parameters



- Retrieve parameters from the `componentDidMount()` function
- Use `this.props.match.params`

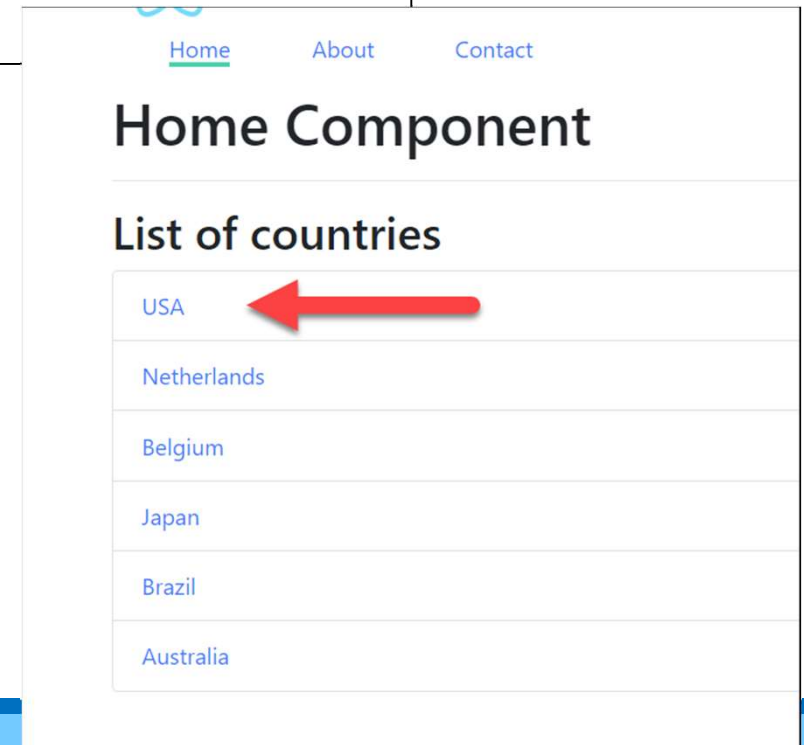
```
// retrieve id and name from the querystring  
// Note the usage of ES6 destructuring here and  
// composing the setState() object  
componentDidMount() {  
  const {id, name} = this.props.match.params;  
  this.setState({  
    id,  
    name  
  })  
}
```

3. Create <Link> route to detail page

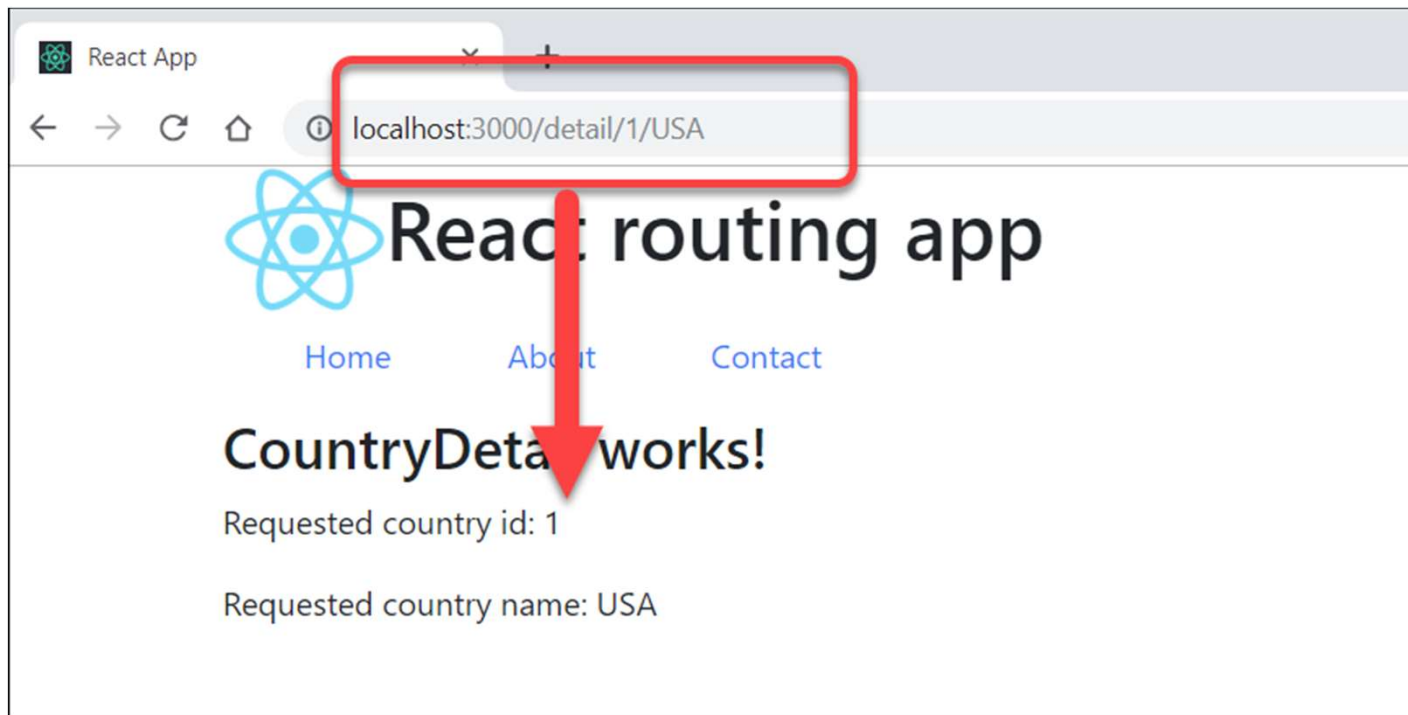
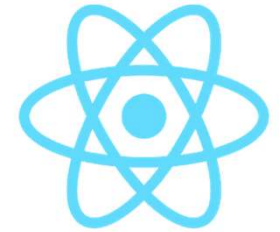


Create a link to send the now mandatory `id` and `name`

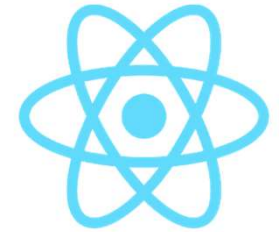
```
<li className="list-group-item"
  key={country.id}>
  <Link to={`/detail/${country.id}/${country.name}`} >
    {country.name}
  </Link>
</li>
```



4. Show Results in the UI

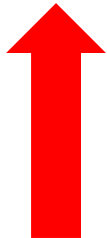


Show Country data

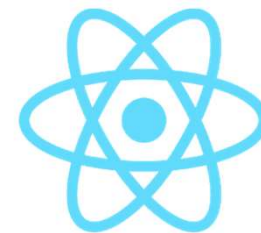


- Grab the data (for now: local, to get the correct country)
 - You know how to fetch data from an API from the previous module
- Look for the data based on the `id`

```
componentDidMount() {  
  const {id, name} = this.props.match.params;  
  const country = this.state.countries.find(c => c.id === +id);  
  this.setState({  
    id,  
    name,  
    country  
  })  
}
```

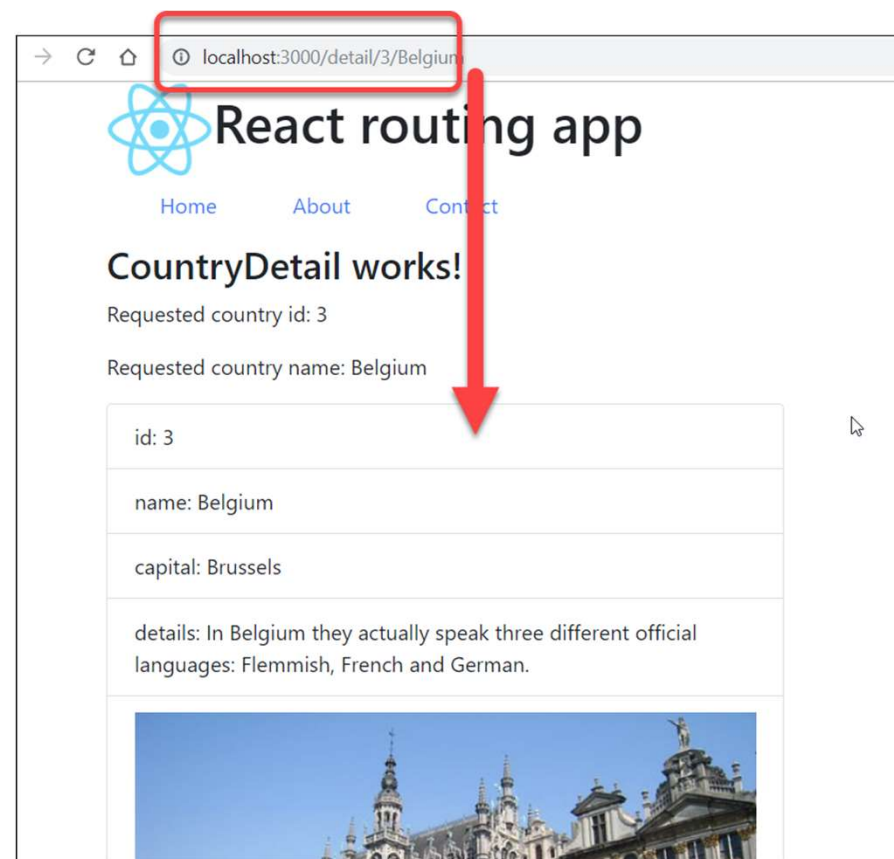
A large red arrow pointing upwards, highlighting the `country` property in the `setState` call.

Show Country Details in UI



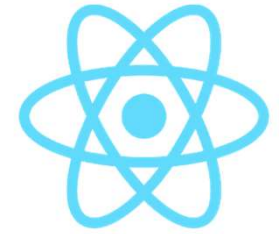
- Use conditional rendering, b/c setting state is asynchronous!

```
{
  this.state.country &&
  <ul className="list-group">
    <li className="list-group-item">
      id: {this.state.country.id}
    </li>
  </ul>
  ...
}
```

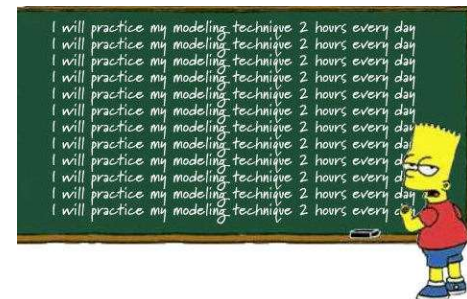


[../examples/620-routing-parameters](https://github.com/robertwaite/react-router-620-routing-parameters)

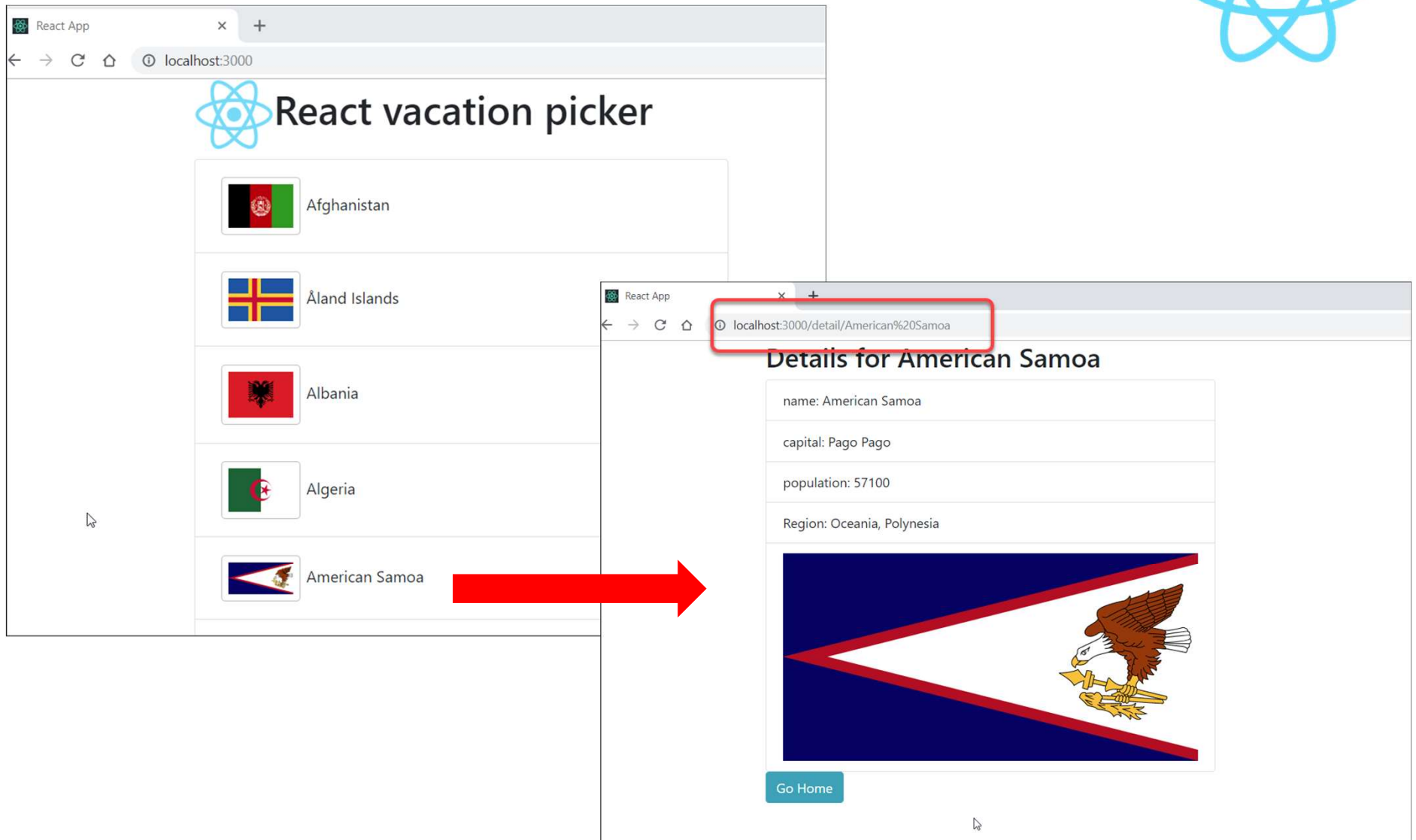
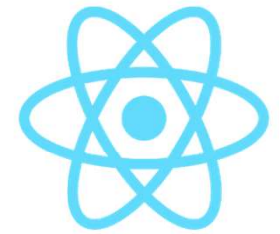
Workshop



- Study the example project
- See how routing parameters work.
- Try some different parameters, or adding/removing a parameter
- Example `../620-routing-parameters`
- OR: Work from example `../500-api-call`
 - Add routing to the project
 - Clicking on a specific country from the API navigates to a detail component, which shows and fetches country details
 - (solution: `../workshops/50-routing-api`)



Sample result from the workshop



The image displays two browser windows from a React application. The left window, titled 'React App' and running on 'localhost:3000', shows a page titled 'React vacation picker'. It features a list of countries with their flags: Afghanistan, Åland Islands, Albania, Algeria, and American Samoa. A red arrow points from the 'American Samoa' entry to the right window. The right window, also titled 'React App', shows a detailed view for American Samoa at the URL 'localhost:3000/detail/American%20Samoa'. The URL is highlighted with a red box. The page title is 'Details for American Samoa'. It contains a table with the following information:

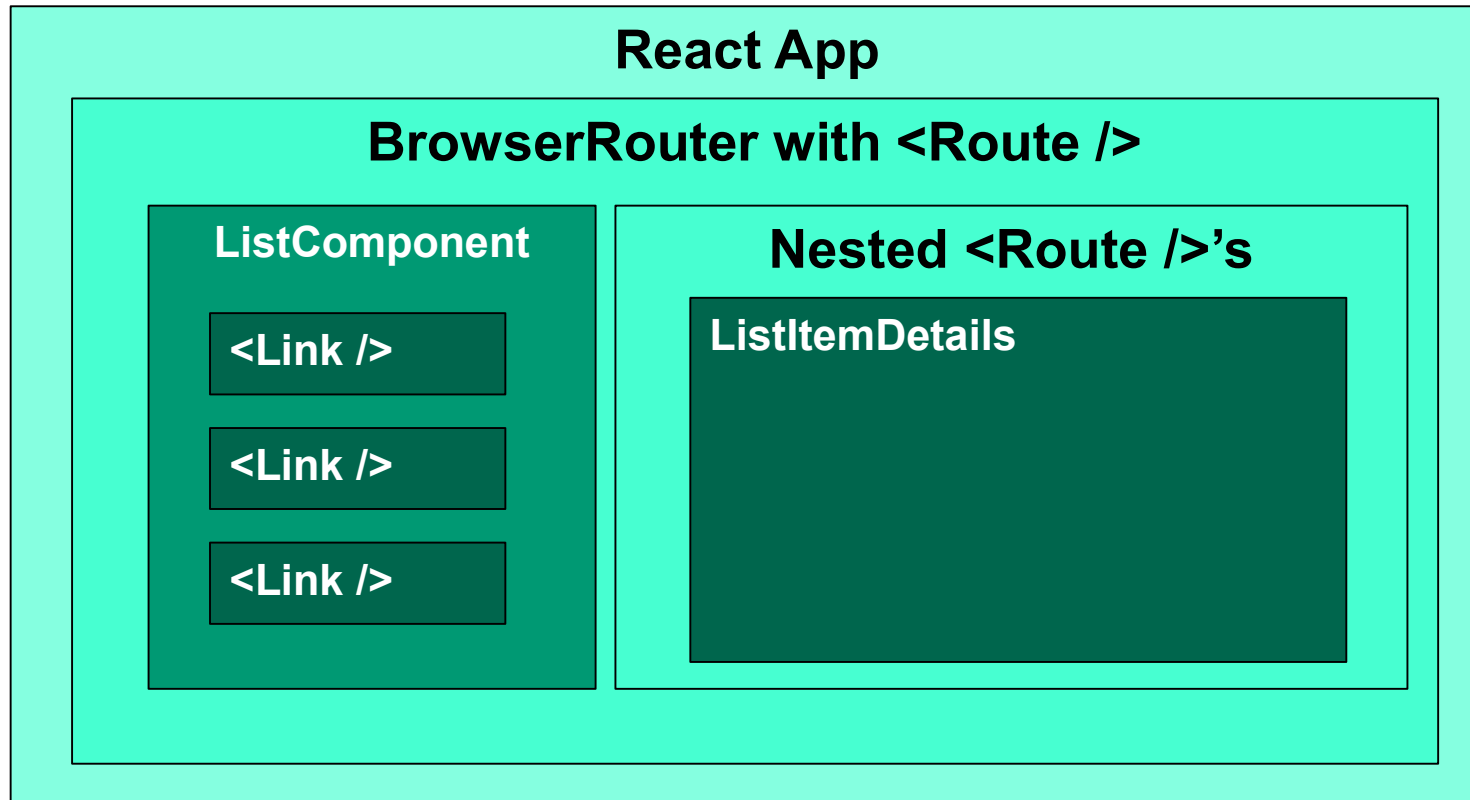
name: American Samoa
capital: Pago Pago
population: 57100
Region: Oceania, Polynesia

Below the table is a large image of the flag of American Samoa, which features a blue field with a white triangle and a red border, and a brown eagle with a shield on its chest. At the bottom left of the page is a 'Go Home' button.

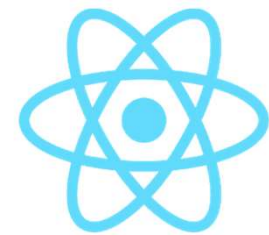


Nested Routes

What are nested router views?



Show router links inside another routed component, but still update the URL

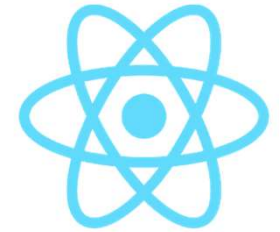


Adding a nested router

- The `<Route />` is just a component, so you can place it inside other components
 - And, if necessary – conditionally.
 - In example app: `<Home />`
- React does *not have* an array of 'child' routes in the main routing table, like other frameworks

- ```
children: [
 { <child-router-path-1> }, { <child-router-path-2> }
]
```

**Invalid in  
React!**



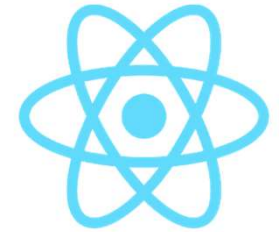
# 1. Create nested routes

```
class Home extends Component {
 render() {
 return (
 <div>
 ...
 <div className="col-md-6">
 {/*The nested routes*/}
 <h3>Detail component with nested route</h3>
 <Route path="/detail/:id/:name" component={CountryDetail}/>
 </div>
 </div>
)
 }
}
```

**<Route /> in Home component.  
If the URL matches, the  
component renders. Otherwise  
the route returns null**

(Don't forget to import `Route` and  
`CountryDetail` in this case)

## 2. Remove `<Route />` from `Routes.js`



```
class Routes extends Component {
 render() {
 return (
 <>
 <Route path="/" component={Home}/>
 <Route path="/about" component={About}/>
 <Route path="/contact" component={Contact}/>
 {/*Route below not used anymore, b/c the nested route is now a child*/}
 {/*inside the Home Component*/}
 {/*<Route path="/detail/:id/:name" component={CountryDetail}/>*/}
 </>
);
 }
}
```

Remove `<Route />`  
from the main  
routing  
table/navigation

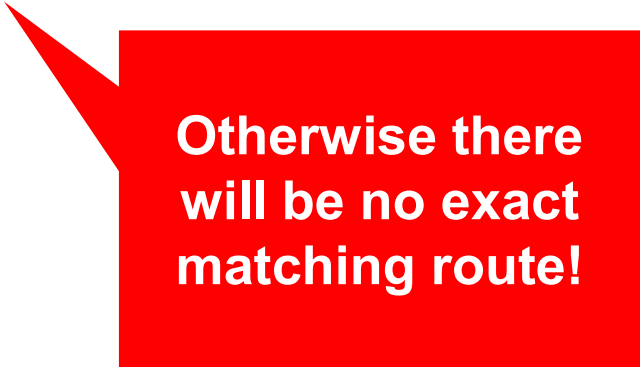
### 3. Remove exact attribute from Main route

Before

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

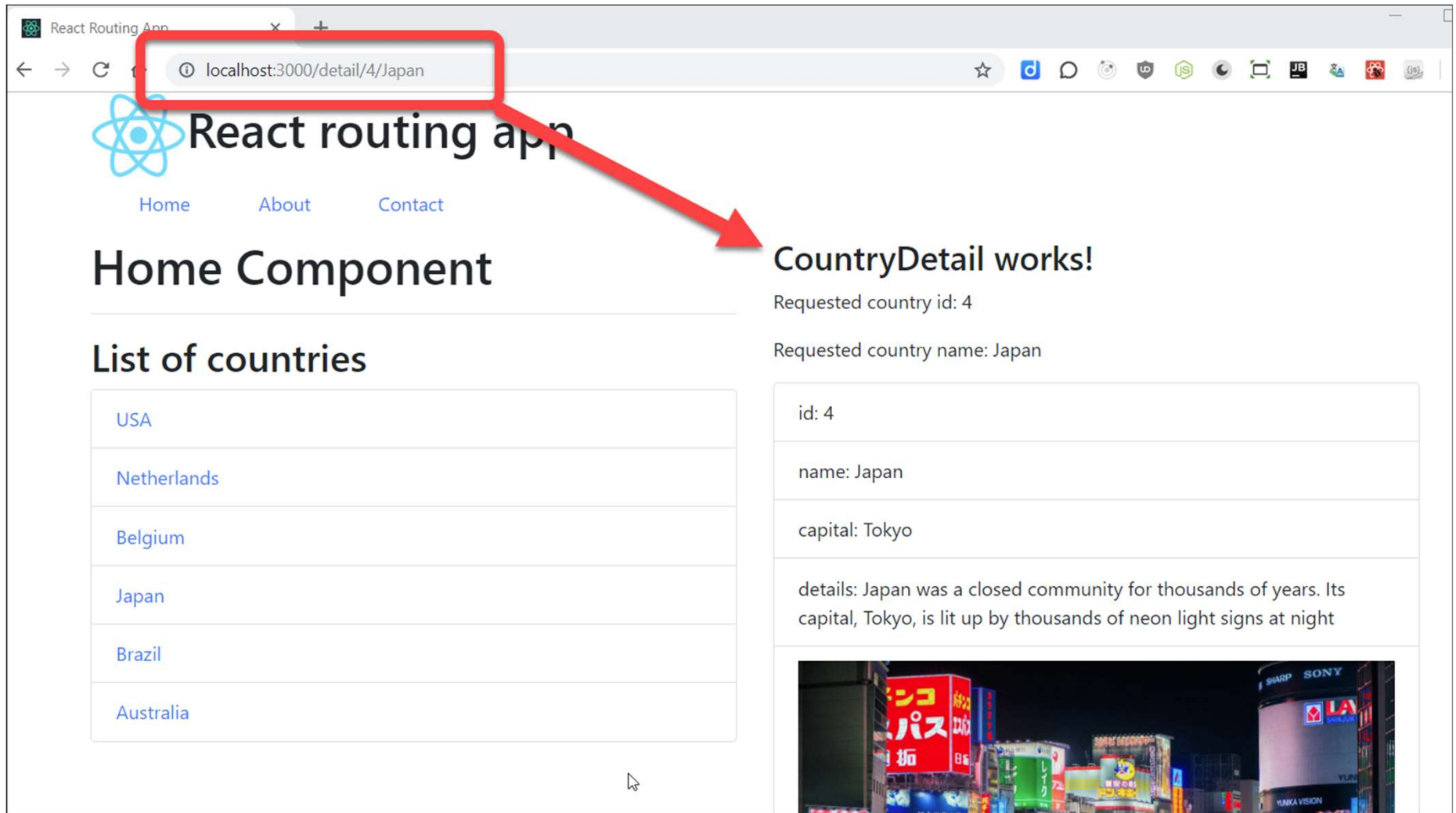
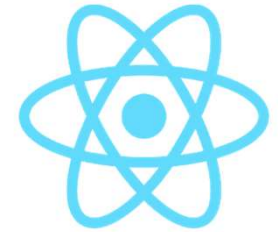
After

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



Otherwise there  
will be no exact  
matching route!

# Result – it works (the first time!)



React Routing App

localhost:3000/detail/4/Japan

React routing app

Home About Contact

## Home Component

### List of countries

- USA
- Netherlands
- Belgium
- Japan
- Brazil
- Australia

## CountryDetail works!

Requested country id: 4


Requested country name: Japan

id: 4

name: Japan

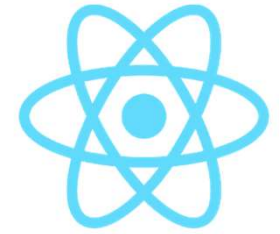
capital: Tokyo

details: Japan was a closed community for thousands of years. Its capital, Tokyo, is lit up by thousands of neon light signs at night





# Updating the detail component



- The nested `<Route />` component is *not* rerendered when the route changes
- So `componentDidMount()` only fires once!
- we explicitly need to set that using the `componentDidUpdate()` lifecycle hook

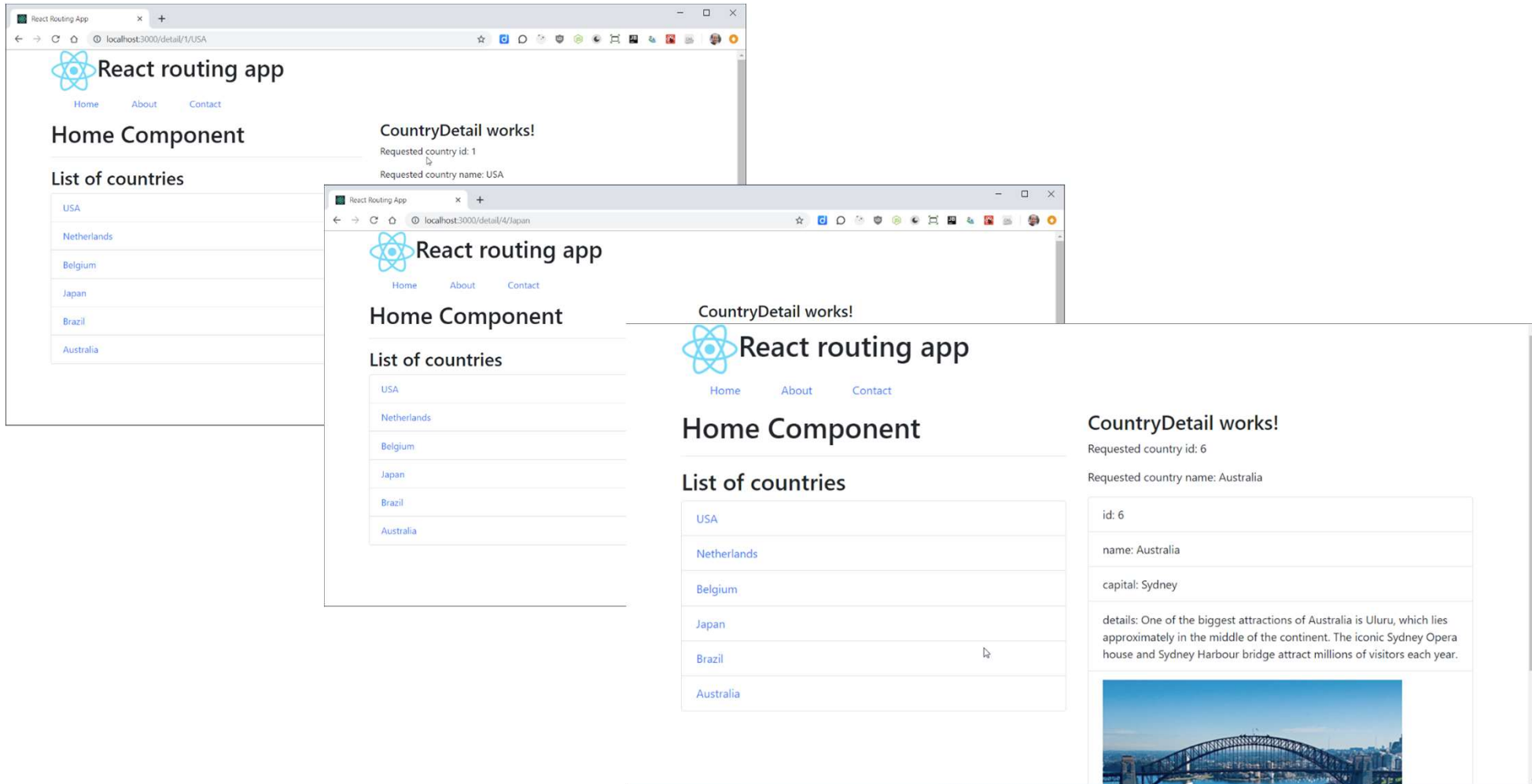
# Updating Country detail component



```
// in CountryDetail.js:
// Re-render contents of the component when the route updates
componentDidUpdate(prevProps, prevState, snapshot) {
 // get id and name for current country
 const {id, name} = this.props.match.params;

 // Get previous country. Fetch a new country and update state only if
 // they are different, for preventing an infinite loop
 if (prevProps.match.params.id !== id) {
 const country = this.state.countries.find(c => c.id === +id);
 this.setState({
 id,
 name,
 country
 })
 }
}
```

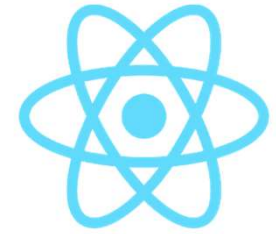
# Result



The image displays three browser windows showing the 'React routing app' at different stages of its development. Each window has a title bar 'React Routing App' and a browser address bar.

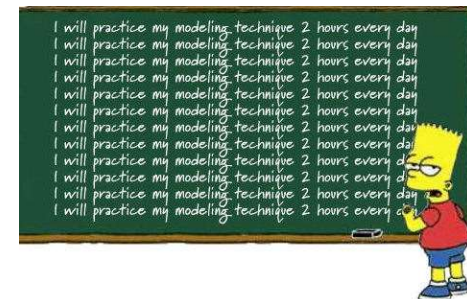
- Top-left window:** The address bar shows `localhost:3000/detail/1/USA`. The page content includes a 'Home Component' with a 'List of countries' (USA, Netherlands, Belgium, Japan, Brazil, Australia) and a 'CountryDetail works!' section showing 'Requested country id: 1' and 'Requested country name: USA'.
- Middle window:** The address bar shows `localhost:3000/detail/4/Japan`. The page content is identical to the top-left window, but the 'CountryDetail works!' section is not visible, suggesting it is rendered but not captured in this specific screenshot.
- Bottom-right window:** The address bar shows `localhost:3000/detail/6/Australia`. The page content includes the 'Home Component' and 'List of countries'. The 'CountryDetail works!' section shows 'Requested country id: 6' and 'Requested country name: Australia'. Below this, there is a 'details' section with text about Uluru and the Sydney Opera house, and a photograph of the Sydney Harbour Bridge.

App now works as expected



# Workshop

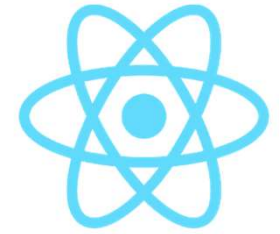
- Study the example project.
- Example: show the `<About />` component in nested route
- Do this for your own app with a list of data
  - Clicking on a listitem, shows the details in a nested route
- Example `../630-nested-routes`
- OR: Work from your previous workshop
  - Clicking on a specific country from the API navigates to a detail component, which shows and fetches country details *inside a nested route*
  - (solution: `../workshops/60-nested-routing`)





# Miscellaneous routing

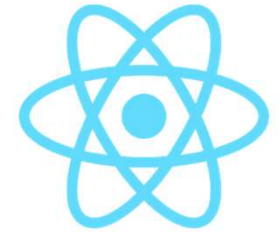
Using `<Switch />`, passing props and more



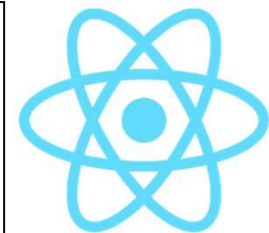
# Various routing tips & tricks

- Using `<Switch>`
- Using `render`
- Creating a 404 page

## Using `<Switch />`



- **Problem:** you have a top component, holding all state.
- You want to pass props down to a routed component
- **Solution:** use the `<Switch />` element
  - For instance inside your `<App />`
- NOT in a separate `<Routes />` element anymore
- You can pass parameters as usual
- Switch renders *the first* matching route



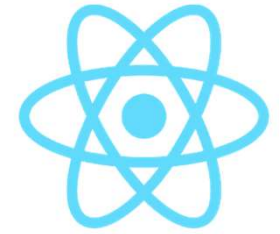
```
// App.js
// Render UI
render() {
 return (
 <BrowserRouter>
 ...
 <MainNavigation/>
 { /*Using a switch to render the one route at a time.*/ }
 { /*Render the one route at a time.*/ }
 <Switch>
 { /*Home route*/ }
 <Route exact path="/">
 <VacationPicker
 select={(country) => this.selectCountry(country)}
 countries={this.state.countries}/>
 ...
 <CountryDetail country={this.state.currentCountry}/>
 </Route>
 { /*Cats route*/ }
 <Route path="/cats">
 <CatPics/>
 </Route>
 ...
 </Switch>
 </div>
 </BrowserRouter>
)
};
```

Switching  
between routes

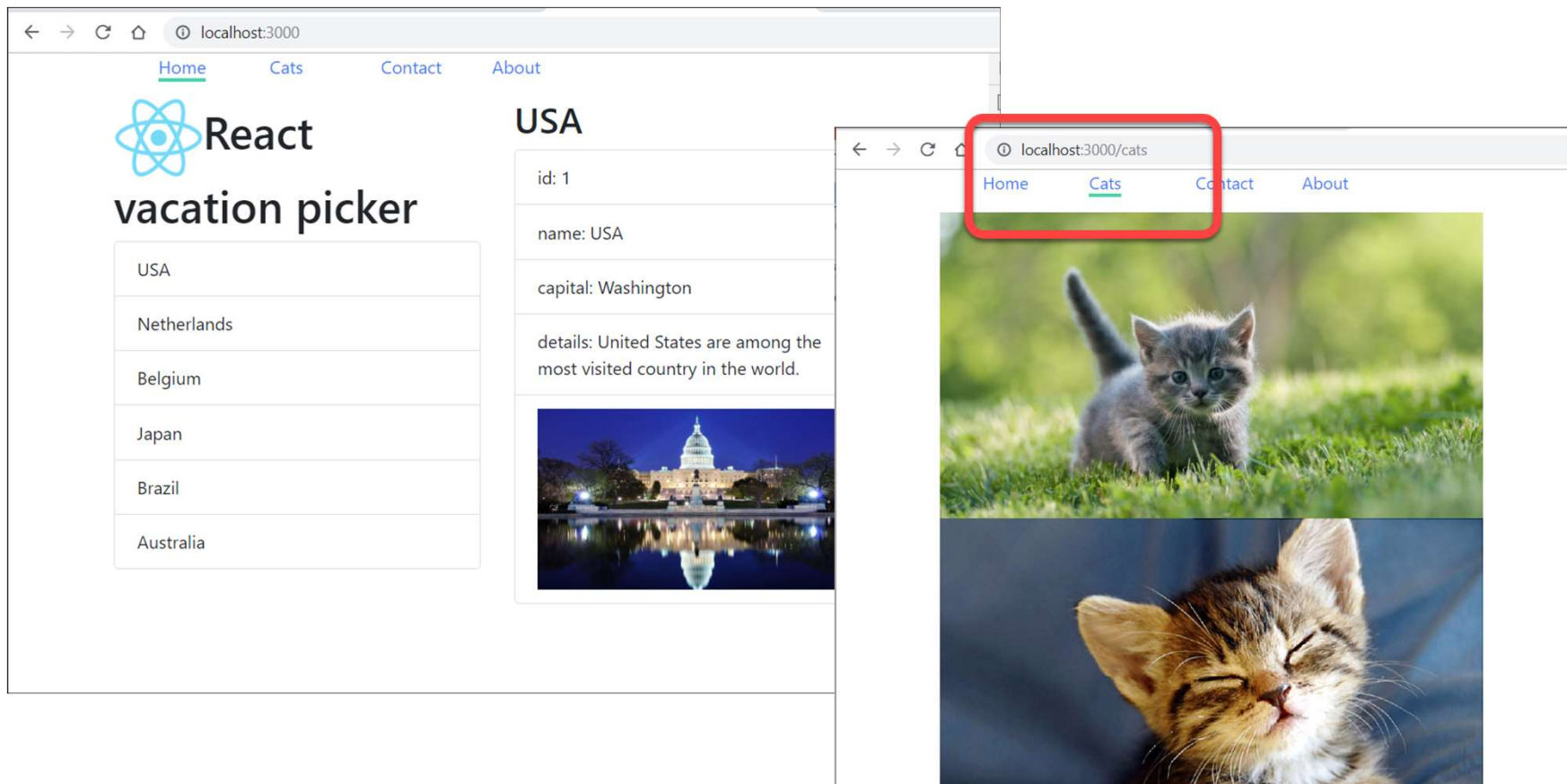
Passing  
parameters as  
usual



# Result



- Visually the same:

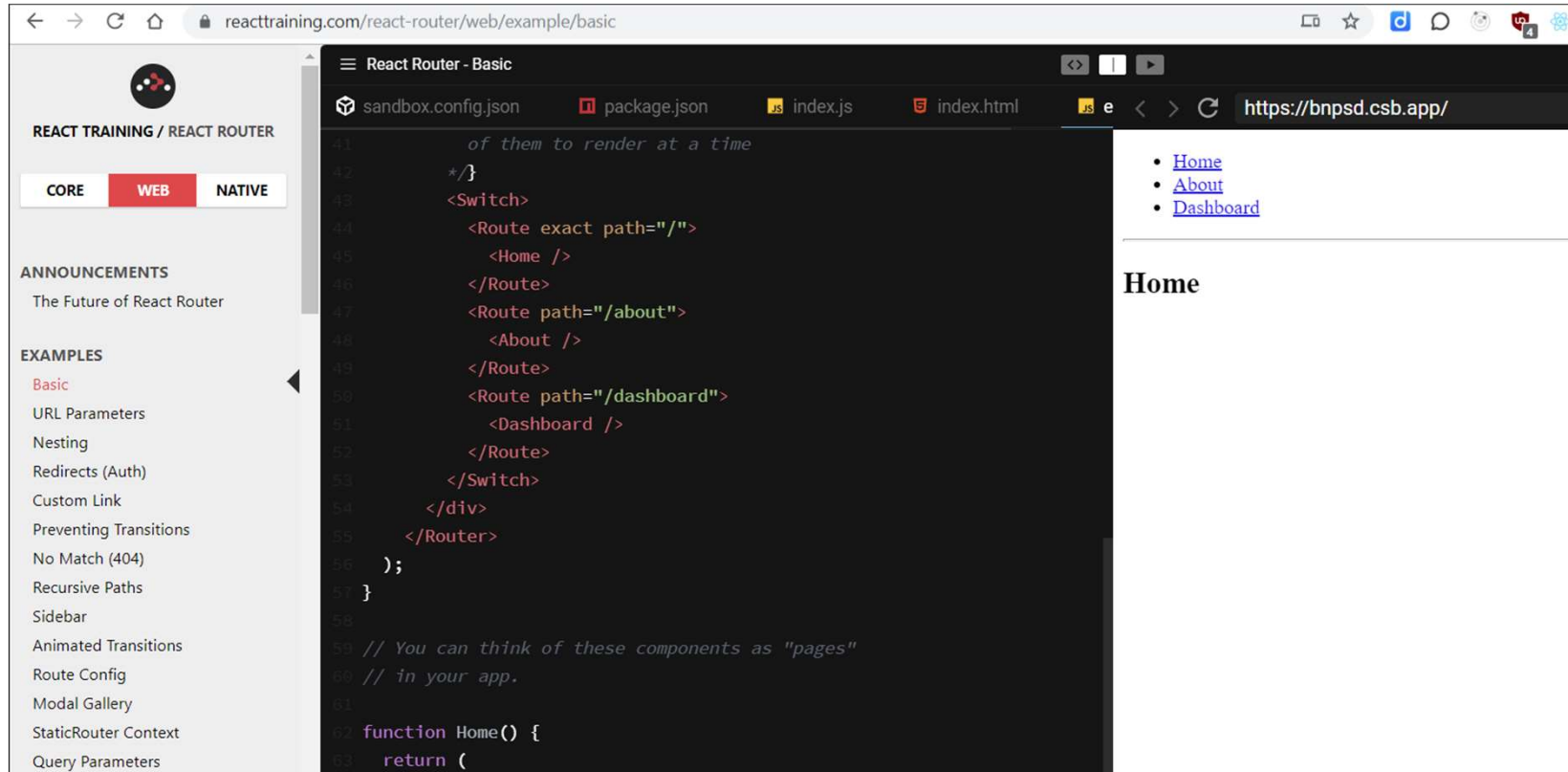


../examples/640-switch

# Documentation on Switch



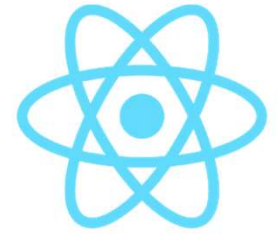
- Switch is the default method that `reacttraining.com` uses
- <https://reacttraining.com/react-router/web/example/basic>



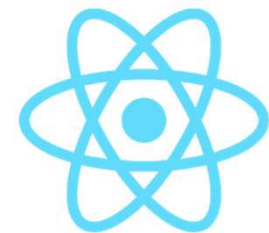
The screenshot shows a web browser displaying the React Router Basic example page. The browser's address bar shows the URL `https://bnpsd.csb.app/`. The page is titled "React Router - Basic" and features a sidebar with navigation links for "CORE", "WEB", and "NATIVE". The "WEB" section is selected, and the "Basic" example is highlighted under the "EXAMPLES" section. The main content area displays the code for the `Switch` component, which renders the `Home` component for the root path and the `About` component for the `/about` path. The `Home` component is rendered in the preview area on the right, showing the text "Home".

```
41 of them to render at a time
42 */}
43 <Switch>
44 <Route exact path="/">
45 <Home />
46 </Route>
47 <Route path="/about">
48 <About />
49 </Route>
50 <Route path="/dashboard">
51 <Dashboard />
52 </Route>
53 </Switch>
54 </div>
55 </Router>
56);
57 }
58
59 // You can think of these components as "pages"
60 // in your app.
61
62 function Home() {
63 return (
```

# Using the `render` attribute



- **Problem:** you still want to pass props down to a child component, but don't want the `<Switch>` overhead
- **Solution:** use the `render` function as an attribute
- A little bit more inline code, but easy to pass props and attributes



## Using render inside <Route />

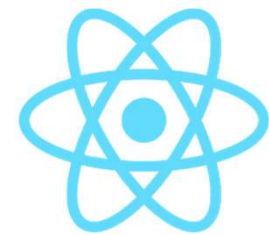
“If the path is matched, render this component”

```
{/*Home route*/}
<Route exact path="/" render={() => {
 <VacationPicker
 select={(country) => this.selectCountry(country)}
 countries={this.state.countries}/>
}}/>

{/*Cats route*/}
<Route path="/cats" render={() => <CatPics/>}/>

...
```

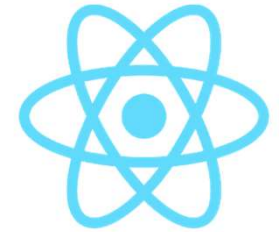
Result is (visually) the same



## Creating a 404 page

- Create a new 404 component, for instance  
`<FileNotFound />`
- Create a route *without* a path
  - The component is rendered if none of the above routes are matched
- Wrap the routes (again) in a `<Switch>` block
  - `<Switch>` can be used with component *and* render

# Rendering the 404 component



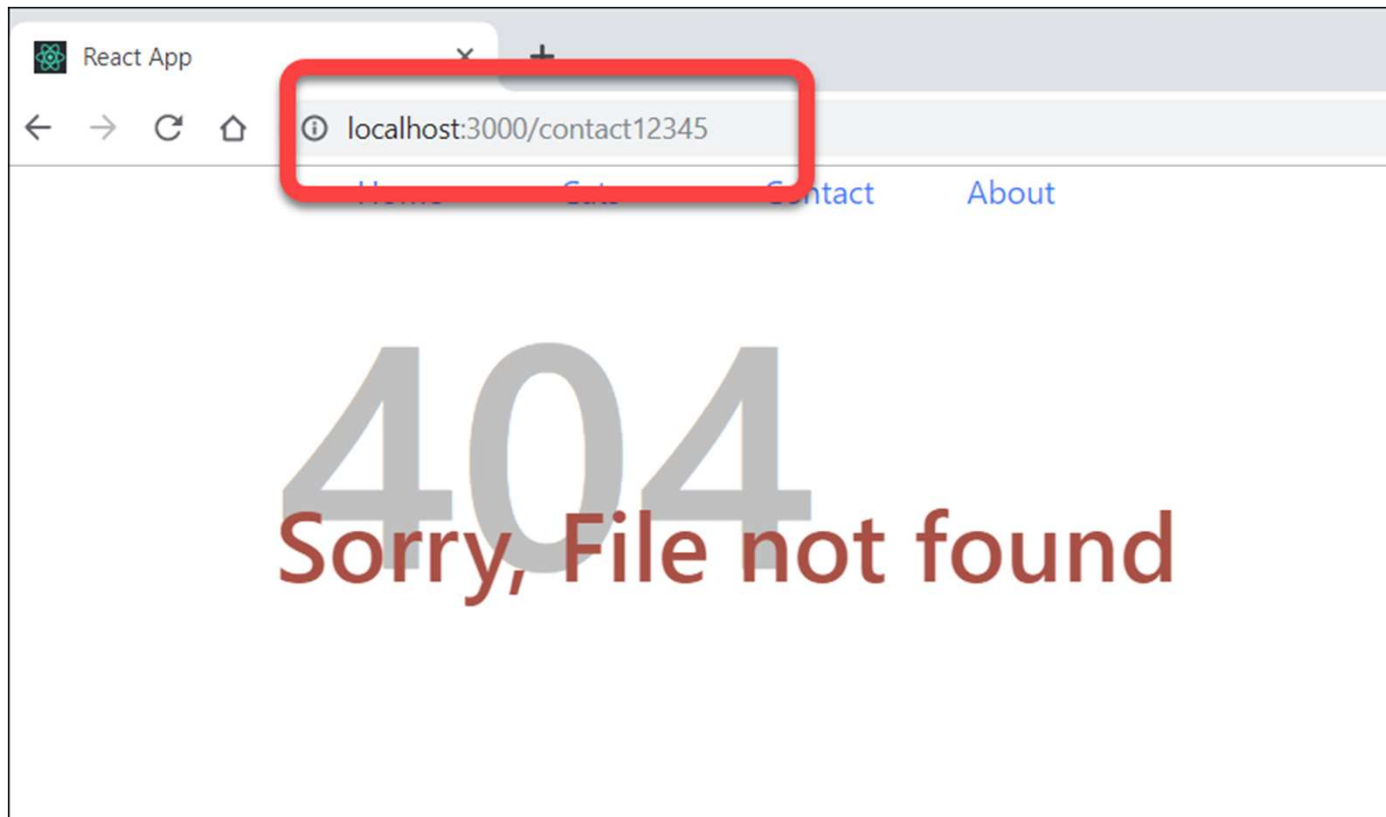
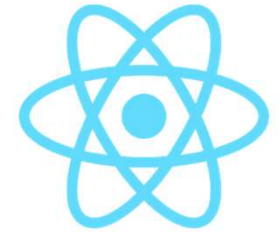
```
<Switch>
 <Route exact path="/" render={() => {
 ...
 }}/>

 {/* About route*/}
 <Route path="/about" render={() => <About/>}/>

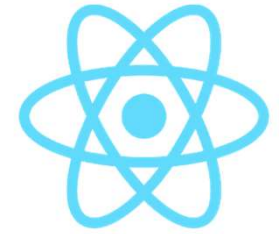
 {/* Catch-all / 404-route*/}
 {/*Just don't give the Route a path, it will render if none*/}
 {/*of the above options was matched*/}
 <Route component={NotFound}/>
</Switch>
```

**404-component inside  
a <Switch> block**

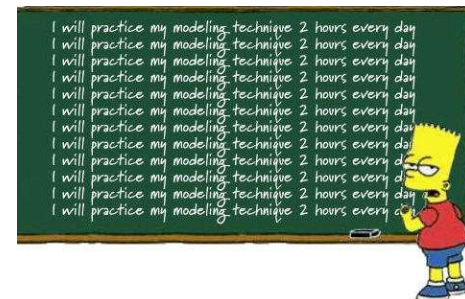
# Result



# Workshop

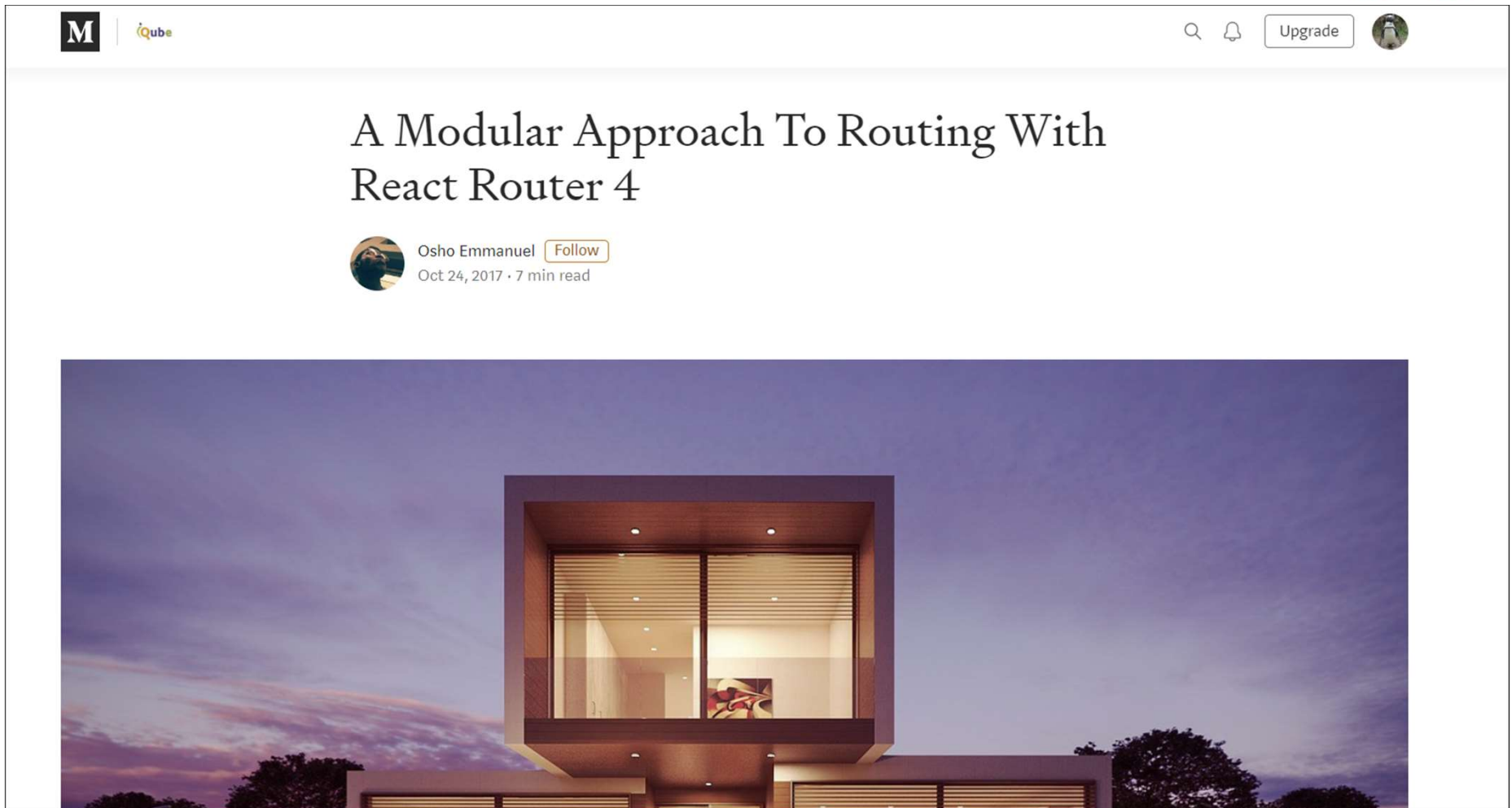
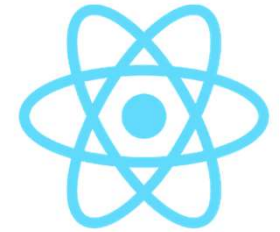


- Rewrite your own application to use a `<Switch>` block
- Also try a render function
- Create a 404 component for your app
- Example `../650-render`



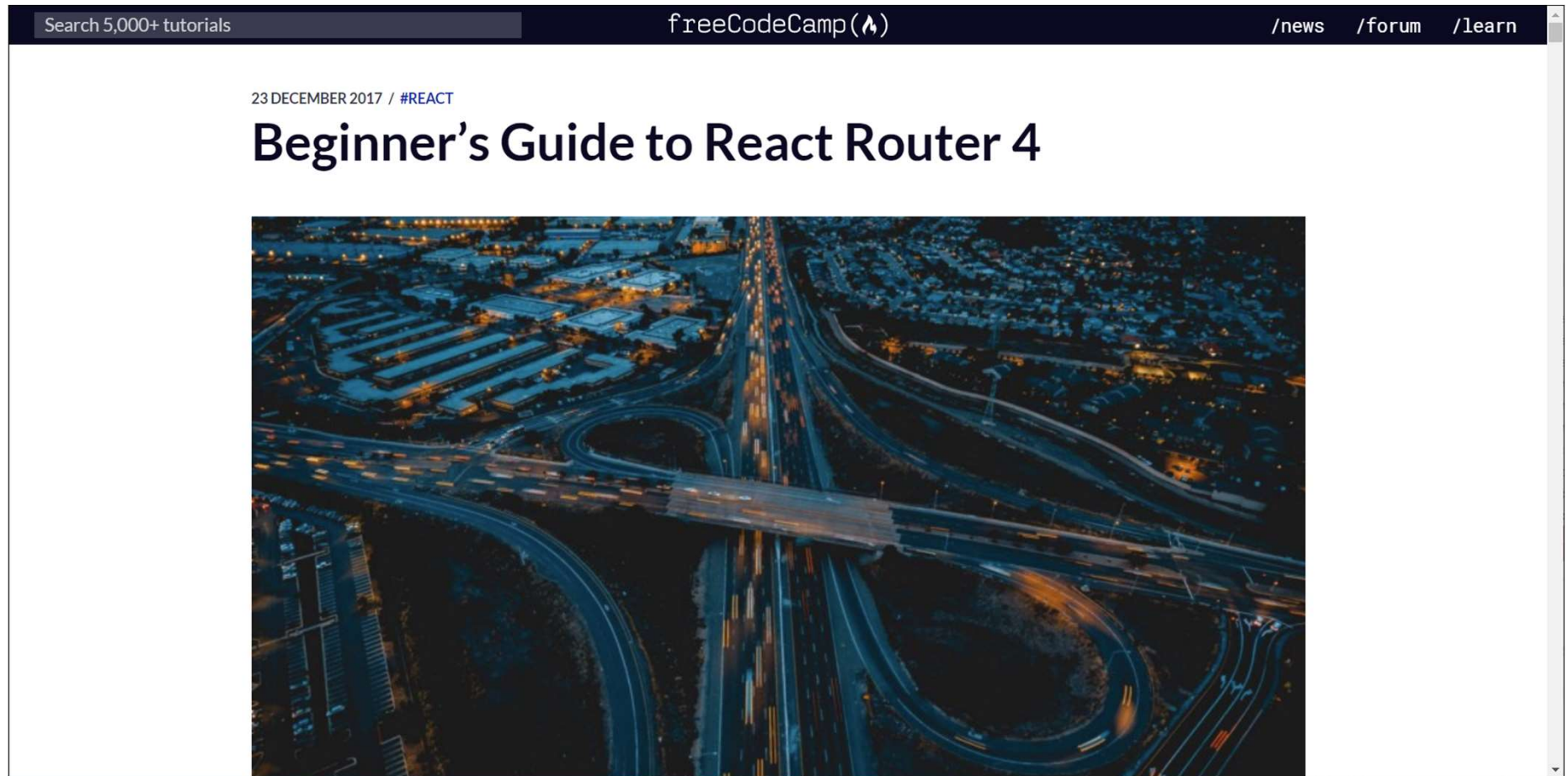
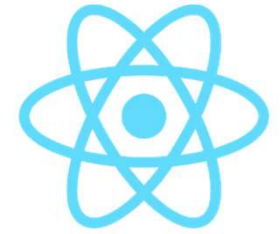


# More info



<https://medium.com/iqube-bits/a-modular-approach-to-routing-with-react-router-4-d4a3db9f56ae>

# Beginner's Guide to React Router



<https://www.freecodecamp.org/news/beginners-guide-to-react-router-4-8959ceb3ad58/>

# Course React Router



TYLERMCGINNIS.COM

For Business **Courses** Blog Newsletter Reviews Login

## React Router

4.8 ★★★★★ 419 Reviews

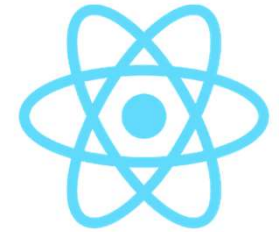
Updated 15 months ago

This course is **guaranteed** to be up to date.

For good reason, React Router is the most popular 3rd party library in the React ecosystem. If you're using React, odds are you're also using React Router. React Router v4 introduced a new dynamic, component based approach to routing. If you're used to static routing (like in Express, Angular, or Ember), this new paradigm might be difficult to grasp. The goal of this course is to tackle every scenario you might encounter when building an app with React Router so that when the time comes, you're ready. With over six hours of video, 18 different routing scenarios, and an entire real world project you'll build, we can confidently say this course is the most comprehensive and effective way to

<https://tylermcginis.com/courses/react-router/>

# Checkpoint



- You know how to install the router and how it works
- You are familiar with `<BrowserRouter>`, `<Route>`, `<Link>`, `<NavLink>` and `<Switch>`
- You can highlight the current route
- You can navigate from code
- You know how to handle route parameters
- You can create nested routes and components