



# React.4

---

## ▼ What is the <Switch /> component?

The <Switch /> component will wrap several <Route /> components and will render only the **first** <Route /> that matches the current browser URL.

The <Route /> has to be a direct descendant of the <Switch />. That means that you should not have a <div> or other elements in between the <Switch /> and the <Route />.

```
import React from "react";
import {BrowserRouter, Switch, Route} from "react-router-dom";

function App() {
  return <BrowserRouter>
    <div>The rest of your app goes here</div>

    <Switch>
      <Route exact path="/about">
        <About />
      </Route>
      <Route exact path="/">
        <Home />
      </Route>
    </Switch>
  </BrowserRouter>;
}
```

## ▼ What is the <Link /> component?

The <Link /> component takes a **to** prop. When the user clicks on the link, React Router will navigate to the path provided by the **to** prop.

This <Link /> element will render an <a> element, but with some event handlers that will **instruct React Router to do an instant redirect instead of a page navigation**.

So React Router is performing an `event.preventDefault()` to prevent the page from navigating to a new page. You don't have to do that, it's all declarative.

```
import React from "react";
import {BrowserRouter, Switch, Route, Link} from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      { /* Switch and Route goes here */ }
    </BrowserRouter>
  );
}
```

- ▼ What does the HTML `<nav>` element represent?  
section of a page whose purpose is to provide navigation links
- ▼ What does the HTML `<main>` element represent?  
the dominant content of the `<body>` of a document
- ▼ Why place `<Switch />` and `<Route />` components inside the `<main>` element?  
you're instructing the browser, search engines, etc. that this is where the main content is on your page
- ▼ Why do we use React Router's URL parameters?  
to create pages that will have the same page structure but the data will be different  
e.g. different product pages but they are all Product component

```
<Route exact path="/products/:id">
  <Product />
</Route>
```

▼ What does the useParams() hook allow you to do?

The useParams() hook allows us to extract the parameters from the URL.

▼ What does the useParams() hook return?

an object containing the URL parameters as key/value pairs

```
// Product.js
import React from "react";
import {useParams} from "react-router-dom";

function Product() {
  const params = useParams();
  const id = params.id; // "42"
  const type = params.type; // "food"

  return <h2>Product</h2>;
}
```

The values of the URL parameters are always in strings because they are extracted from the URL.

▼ What is the Effect Hook, useEffect()?

The Effect Hook, useEffect, adds the ability to perform side effects from a function component. It serves the same purpose as componentDidMount, componentDidUpdate, and componentWillUnmount in React classes, but unified into a single API.

When you call useEffect, you're telling React to run your "effect" function after flushing changes to the DOM.

▼ Product Details page (example)

```
// index.js
import React from "react";
import {render} from "react-dom";
import {BrowserRouter, Switch, Route} from "react-router-dom";
import StoreFront from "./StoreFront.js";
```

```

import ProductDetails from "../ProductDetails.js";

function App() {
  return (
    <BrowserRouter>
      <Switch>
        <Route exact path="/">
          <StoreFront />
        </Route>
        <Route exact path="/products/:id">
          <ProductDetails />
        </Route>
      </Switch>
    </BrowserRouter>
  );
}

render(<App />, document.querySelector("#react-root"));

```

```

// StoreFront.js

import React, {useEffect, useState} from "react";
import Product from "../Product.js";
import Loader from "../Loader.js";
import useFetch from "../useFetch.js";

export default function StoreFront() {
  const [products, setProducts] = useState([]);
  const {get, loading} = useFetch("https://react-tutorial-demo.firebaseio.com/");

  useEffect(() => {
    get("products.json")
      .then(data => {
        setProducts(data);
      })
      .catch(error => console.log(error))
  }, []);

  return <div className="store-front">
    {loading && <Loader />}
    {products.map(product => <Product key={product.id} details={product} />)}
  </div>;
}

```

```

// Product.js

import React, {useState} from "react";
import {Link} from "react-router-dom";

export default function Product(props) {

```

```

const [count, setCount] = useState(0);

const {details} = props;

function handleIncrementClick() {
  setCount(count + 1);
}
function handleDecrementClick() {
  if (count > 0){
    setCount(count - 1);
  }
}

return (
<Link to={`products/${details.id}`}>
  <div className="product">
    <img src={details.image} width="50" alt={details.name} />
    <div className="product-info">
      <h2>{details.name}</h2>
      <p>{details.description}</p>
    </div>
  </div>
</Link>
);
}

```

```

// ProductDetails.js
import React, {useEffect, useState} from "react";
import {useParams, Link} from "react-router-dom";
import useFetch from "../useFetch.js";

export default function ProductDetails () {
  const params = useParams();
  const [product, setProduct] = useState({});
  const {get} = useFetch("https://react-tutorial-demo.firebaseio.com/");

  const id = params.id;

  useEffect(() => {
    get(`productdetails/id${id}.json`)
      .then(data => {
        setProduct(data);
      })
      .catch(error => console.log(error))
  }, []);

  return <div>
    <Link to="/">Back home</Link>
    {product && <div>
      <h2>{product.name}</h2>
      <p>{product.description}</p>
    }
  </div>

```

```

        <h3>${product.price}</h3>
        <img src={product.image} width="100" />
      </div>
    </div>
  }
}

```

### ▼ What are nested routes?

it's the rendering of two routes at the same time

aka two different contents from two routes at the same time

e.g. /about/us page is rendering the content from the /about route as well as the content from the /about/us route

### ▼ Nested routes (example)

Making nested routes work in React Router is a 2 step process:

1. You can make **any** component return a (nested) `<Switch />` which contains routes. In our example, the `<About />` component will return a `<Switch />`.
2. The route where you'd like to have nested routes support should **NOT** have the `exact` prop anymore. In our example, the `/about` supports nested routes so we should remove the `exact` prop from that `<Route />`.

Whenever you need to use nested routing, you **have** to get rid of the `exact` prop only on that specific route.

That's because if you keep the `exact` prop on the `<Route path="/about">`, it won't match for `/about/us` or `/about/team`. So you won't get any nested routes.

```

import React from "react";
import {BrowserRouter, Link, Switch, Route} from "react-router-dom";
import {render} from "react-dom";

function About() {
  return <>
    <h1>About</h1>
    <ul>
      <li><Link to="/about/us">About us</Link></li>
      <li><Link to="/about/team">About the team</Link></li>
    </ul>

    <Switch>

```

```

    <Route exact path="/about/us">
      <h2>About us</h2>
    </Route>
    <Route exact path="/about/team">
      <h2>About the Team</h2>
    </Route>
  </Switch>
</>;
}

function App() {
  return (
    <BrowserRouter>
      <Switch>
        <Route exact path="/">
          <Link to="/about">Go to /about page</Link>
        </Route>
        <Route path="/about">
          <About />
        </Route>
      </Switch>
    </BrowserRouter>
  );
}

render(<App />, document.querySelector("#react-root"));

```

#### ▼ How will React Router match without an exact prop?

When the exact prop is NOT used in the Route, then React router will match the beginning of the URL. If the beginning of the URL matches, then the `<Route />` will be considered matching.

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

this will match everything after /about

#### ▼ Why do we need Switch component?

The `<Switch />` component will always make sure to render the **first** (and only the **first**) `<Route />` that matches the path. Once it finds this route, it will skip the remaining routes.

Whereas if you omit the `<Switch />` and directly define your routes, then several routes **could** be rendered at the same times if they match the current path.

#### ▼ How to handle a route like this?: /products/:id/details

useRouteMatch hook

ask React Router about the match information that it has for this route

### ▼ What is useRouteMatch() hook?

a hook from React Router that returns an object describing the path, url and, params of the current route that matched the browser URL

### ▼ useRouteMatch hook (example)

```
import React from "react";
import {useRouteMatch} from "react-router-dom";

function Product() {
  const match = useRouteMatch();
  console.log(match);

  return null;
}
```

```
// if visited /products/1, this is match:
```

```
{
  path: "/products/:id",
  url: "/products/1",
  isExact: true,
  params: {
    id: "1"
  }
}
```

```
// so if we want to visit the product details: /products/:id/details
// we can use the match.url
```

```
<Link to={`${match.url}/details`} >See details</Link>
```

```
// if we want to construct the NESTED route, we use match.path
```

```
<Switch>
  <Route path={`${match.path}/details`} >Details</Route>
  <Route path={`${match.path}/buy`} >Buy</Route>
</Switch>
```



## ▼ Nested routes (example)

```
import React from "react";
import {BrowserRouter, Link, Switch, Route} from "react-router-dom";
import {render} from "react-dom";
import Product from "./Product.js";

function App() {
  return (
    <BrowserRouter>
      <Switch>
        <Route exact path="/">
          <ul>
            <li><Link to="/products/1">Go to /products/1 page</Link></li>
            <li><Link to="/products/2">Go to /products/2 page</Link></li>
            <li><Link to="/products/3">Go to /products/3 page</Link></li>
          </ul>
        </Route>
        <Route path="/products/:id">
          <Product />
        </Route>
      </Switch>
    </BrowserRouter>
  );
}

render(<App />, document.querySelector("#react-root"));
```

```
// Product.js

import React from "react";
import {useParams, useRouteMatch, Switch, Route, Link} from "react-router-dom";
import ProductDetails from './ProductDetails.js';

export default function Product() {
  const params = useParams();
  const match = useRouteMatch();

  return <>
    <Link to="/">Back home</Link>
    <h1>Product {params.id}</h1>
    <Link to={`/${match.url}/details`}>See details</Link>

    <Switch>
      <Route path={`/${match.path}/details`}>
        <ProductDetails />
      </Route>
    </Switch>
  </>;
}
```

```
// ProductDetails.js

import React from "react";

export default function ProductDetails() {
  return <>
    <h2>Product details</h2>
    <p>The details of the product show up here.</p>
  </>;
}
```

### ▼ Why is order important in <Switch />?

Because it stops looking for routes after the first one matches.

### ▼ How to link to a 404 not found page?

the <Route> for the 404 page has no path prop. That means it will always match and must be the last Route in the Switch component

```
import React from "react";
import {BrowserRouter, Switch, Route} from "react-router-dom";

function App() {
  return <BrowserRouter>
    <Switch>
      <Route exact path="/">Landing page here</Route>
      <Route exact path="/products">Products page here</Route>
      <Route>404 page here</Route>
    </Switch>
  </BrowserRouter>
}
```

### ▼ How to highlight active page in menu using <NavLink> and activeClassName prop? (example)

```
import React from "react";
import {NavLink} from "react-router-dom";

function App() {
  return <ul>
    <li>
```

```

        <NavLink to="/" activeClassName="active">Home</NavLink>
      </li>
      <li>
        <NavLink to="/about" activeClassName="active">About</NavLink>
      </li>
    </ul>
  }

```

```

.active {
  font-weight: bold;
}

```

```

// if you only want to highlight the link given it's path (and not every NavLink
// at the same time) then use --> exact activeClassName

import React from "react";
import {NavLink} from "react-router-dom";

function App() {
  return <ul>
    <li>
      <NavLink to="/products" exact activeClassName="active">Products</NavLink>
    </li>
    <li>
      <NavLink to="/products/1" exact activeClassName="active">Product 1</NavLink>
    </li>
  </ul>
}

```

#### ▼ How can you redirect to a new page using useHistory() hook?

Use the useHistory() hook which returns a method called push(newRoute).

Make sure to call the useHistory() hook in a component that is nested inside <BrowserRouter /> or else it won't work.

The .push() method should only be used whenever a <Link /> is not possible.

That means whenever the redirect is programmatic (that means, it's based on some logic, such as the result of fetch()).

```

import React, {useEffect} from "react";
import {useHistory} from "react-router-dom";

function HelpPage() {

```

```

const history = useHistory();

useEffect(() => {
  const isLoggedIn = window.localStorage.getItem("isLoggedIn");
  if (!isLoggedIn) {
    history.push("/login");
  }
}, []);

return <h2>Help page</h2>;
}

```

▼ What if you need to run a piece of code whenever React Router navigates to a new URL?

use the `useLocation` hook

The `useLocation` hook **returns information about the currently active route**.

The `location` variable will be an object containing the **pathname**. e.g.  
`location.pathname`

You can then use the `location.pathname` to know the current route the user is browsing. For example `/` or `/about`, etc. depending on your routes.

Just like the `useHistory` hook, you need to make sure that the code is running **inside** a component wrapped by `<BrowserRouter />` or else it won't work.

```

import React from "react";
import {BrowserRouter, Switch, Route, useLocation} from "react-router-dom";
import {render} from "react-dom";

function App() {
  const location = useLocation();
  console.log(location.pathname);

  return <Switch>
    { /* routes here... */ }
  </Switch>
}

render(<BrowserRouter><App /></BrowserRouter>, document.querySelector("#react-root"));

```

▼ `useEffect()`, `useLocation()` and analytics or logging when user visits a page (example)

The code inside the `useEffect` hook will run whenever the location object changes, which happens whenever the user navigates to a new route.

```
import React, {useEffect} from "react";
import {BrowserRouter, Switch, Route, useLocation} from "react-router-dom";
import {render} from "react-dom";

function App() {
  const location = useLocation();

  // run a piece of code on location change
  useEffect(() => {
    console.log(location.pathname);
    // send it to analytic, or do some conditional logic here
  }, [location]);

  return <Switch>
    { /* routes here... */ }
  </Switch>
}

render(<BrowserRouter><App /></BrowserRouter>, document.querySelector("#react-root"));
```

#### ▼ Composition over inheritance for what?

code reuse between components

#### ▼ How to approach components that don't know their child components ahead of time? (aka the problem of containment)

Such components should use the special `children` prop to pass children elements directly into their output:

```
function FancyBorder(props) {
  return (
    <div className={'FancyBorder FancyBorder-' + props.color}>
      {props.children}
    </div>
  );
}
```

This lets other components pass arbitrary children to them by nesting the JSX:

```
function WelcomeDialog() {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        Welcome
      </h1>
      <p className="Dialog-message">
        Thank you for visiting our spacecraft!
      </p>
    </FancyBorder>
  );
}
```

### ▼ How to achieve a specialized component through composition?

we might say that a WelcomeDialog is a special case of Dialog

a more “specific” component renders a more “generic” one and configures it with props

```
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        {props.title}
      </h1>
      <p className="Dialog-message">
        {props.message}
      </p>
    </FancyBorder>
  );
}

function WelcomeDialog() {
  return (
    <Dialog
      title="Welcome"
      message="Thank you for visiting our spacecraft!" />
  );
}
```

### ▼ What is mounting?

when a component is rendered to the DOM for the first time

### ▼ What is unmounting?

when the DOM produced by a component is removed

▼ Given an ES6 class, what can you do in React?

convert a function component to a class by extending `React.Component`

```
class Message extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>Hello, world!</h1>  
      </div>  
    );  
  }  
}
```

▼ What are lifecycle phases?

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

▼ What are the three main lifecycle phases?

1. Mounting (adding nodes to the DOM)
2. Updating (altering existing nodes in the DOM)
3. Unmounting (removing nodes from the DOM)

▼ What are lifecycle methods?

Each React lifecycle phase has a number of lifecycle methods that you can override to run code at specified times during the process.

▼ What does the 'data flows down' mean in React?

Any state is always owned by some specific component, and any data or UI derived from that state can only affect components “below” them in the tree.

▼ What is React?

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.

▼ Can you have multiple useEffects in one file?

yes

▼ What is a memory leak?

a memory leak is a type of resource leak that occurs when a computer program incorrectly manages memory allocations in a way that **memory which is no longer needed is not released**

▼ What is stack overflow error?

When a program attempts to use more space than is available on the call stack (that is, when it attempts to access memory beyond the call stack's bounds, which is essentially a buffer overflow), the stack is said to overflow, typically resulting in a program crash.

▼ What is a call stack or execution stack?

a call stack is a stack data structure that stores information about the active subroutines of a computer program.

▼ What is a subroutine?

a sequence of program instructions that performs a specific task, packaged as a unit

▼ Why use onSubmit property in a form instead of onClick for submit?

because it will handle enter key and click for submit. if the click is a button element.

▼ React supported events (reference) e.g. onChange, onClick

<https://reactjs.org/docs/events.html#supported-events>

▼ Why does React use keys?

Keys help React identify which items have changed (added/removed/re-ordered). To give a unique identity to every element inside the array, a key is required

▼ What is React.StrictMode?



React has a new strict mode. If you wrap your app in `<React.StrictMode>` `</React.StrictMode>` it will give you additional warnings about things you shouldn't be doing.

It's good to keep your team in line and not using legacy features or things that will be soon be deprecated.

#### ▼ Why does Context fill the same need as Redux?

Context is like state, but instead of being confined to a component, it's global to your application. It's application-level state.

This is dangerous. Avoid using context until you have to use it. One of React's primary benefit is it makes the flow of data obvious by being explicit. Things like context obscure it.

Context (mostly) replaces Redux. Well, typically. It fills the same need as Redux. I really can't see why you would need to use both. Use one or the other.

#### ▼ What is a Portal in React (related: modals)?

You can think of the portal as a separate mount point (the actual DOM node which your app is put into) for your React app.

A common use case for this is going to be doing modals.

You'll have your normal app with its normal mount point and then you can also put different content into a separate mount point (like a modal or a contextual nav bar) directly from a component.

#### ▼ What is a mount point?

the actual DOM node which your app is put into for your React app

#### ▼ Portal and modal (example)

```
<!-- above #root -->  
<div id="modal"></div>
```

```
// Modal.js  
  
import React, { useEffect, useRef } from "react";
```

```
import { createPortal } from "react-dom";

const modalRoot = document.getElementById("modal");

const Modal = ({ children }) => {
  const elRef = useRef(null);
  if (!elRef.current) {
    elRef.current = document.createElement("div");
  }

  useEffect(() => {
    modalRoot.appendChild(elRef.current);
    return () => modalRoot.removeChild(elRef.current);
  }, []);

  return createPortal(<div>{children}</div>, elRef.current);
};

export default Modal;
```

- This will mount a div and mount inside of the portal whenever the Modal is rendered and then *remove* itself whenever it's unrendered.
- We're using the feature of `useEffect` that if you need to clean up after you're done (we need to remove the div once the Modal is no longer being rendered) you can return a function inside of `useEffect` that cleans up.
- We're also using a ref here via the hook `useRef`. Refs are like instance variables for function components. Whereas on a class you'd say `this.myVar` to refer to an instance variable, with function components you can use refs. They're containers of state that live outside a function's closure state which means anytime I refer to `elRef.current`, it's **always referring to the same element**. This is different from a `useState` call because the variable returned from that `useState` call will **always refer to the state of the variable when that function was called**. It seems like a weird hair to split but it's important when you have async calls and effects because that variable can change and nearly always you want the `useState` variable, but with something like a portal it's important we always refer to the same DOM div; we don't want a lot of portals.
- Down at the bottom we use React's `createPortal` to pass the children (whatever you put inside `<Modal></Modal>`) to the portal div.

```

// Details.js

// at the top
import Modal from "../Modal";

// add showModal
state = { loading: true, showModal: false };

// above render
toggleModal = () => this.setState({ showModal: !this.state.showModal });
adopt = () => (window.location = "http://bit.ly/pet-adopt");

// add showModal
const {
  animal,
  breed,
  city,
  state,
  description,
  name,
  images,
  showModal,
} = this.state;

// add onClick to <button>
<button onClick={this.toggleModal} style={{ backgroundColor: theme }}>
  Adopt {name}
</button>;

// below description
{
  showModal ? (
    <Modal>
      <div>
        <h1>Would you like to adopt {name}?</h1>
        <div className="buttons">
          <button onClick={this.adopt}>Yes</button>
          <button onClick={this.toggleModal}>No</button>
        </div>
      </div>
    </Modal>
  ) : null
}

```

- We're using a simple `window.location` redirect since we're heading off site. This is bad accessibility so you should be extra cautious when doing this. The button should be an `<a>` tag but I wanted to show you how to do it. But now if you click Yes on the adopt modal it'll take you to the page when you actually can adopt a pet.

- Notice that despite we're rendering a whole different part of the DOM we're still referencing the state in Details.js. This is the magic of Portals. You can use state but render in different parts of the DOM. Imagine a sidebar with contextual navigation. Or a contextual footer.

#### ▼ What are Refs in React?

Refs are **like instance variables for function components**.

Whereas on a class you'd say `this.myVar` to refer to an instance variable, with function components you can use refs.

**They're containers of state** that live outside a function's closure state which means anytime I refer to `elRef.current`, it's **always referring to the same element**.

#### ▼ What are propTypes?

To run typechecking on the props for a component, you can assign the special propTypes property:

```
class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

Greeting.propTypes = {
  name: PropTypes.string
};
```

#### ▼ What is the prop-types library?

Runtime type checking for React props and similar objects.

<https://www.npmjs.com/package/prop-types>

#### ▼ What are Dynamic Pages?

HTML pages are created dynamically, on the server - with the help of a server-side programming language and typically also a “templating engine”

▼ What are Single Page Applications (SPA)?

The server returns one single, pre-generated HTML page which in turn contains JavaScript code that changes the page dynamically in the browser.

Your client-side JavaScript code will **not connect to a database**.

This provides a mobile-app-like user experience.

▼ Why is React a popular choice for SPAs?

Since updating the entire HTML page dynamically via client-side JavaScript code requires quite a lot of JavaScript instructions, you typically use a framework or library for the heavy lifting.

▼ What are Static Pages?

HTML pages were pre-generated and are stored as such on the server

▼ What are Dynamic websites?

It means that the website you're viewing is dynamically created on the server, by the server-side language used there.

server-side languages like Node.js could be used

every page a user visits needs to be generated on the server. That means, that the user always has to wait for a new page to be sent.

developers building dynamic pages need both frontend (HTML + CSS + JS) and backend development knowledge. At least to a certain degree.

▼ How does a static site generator like Gatsby.js work? (React for static sites)

The basic idea is that rather than rendering on the client side, it renders on the dev's machine during the build process, thus saving on load times. You then take this static site's HTML, and serve it like a regular webpage.

▼ Should you choose a static page/website?

A static page - no matter if created with or without a static site generator - might be a poor choice for a website with content that changes multiple times per hour or minute.

But if you have no dynamic (e.g. user-specific) content then it's a good choice.

▼ Should you choose a SPA app?

A SPA in general often is a decent choice - it's especially great for web apps that don't really need search engine crawling.

▼ Should you choose a dynamic website?

A dynamic page on the other hand can be perfect if you have fast-changing content and SEO matters a lot. An online shop for example.

▼ React and dynamic web apps

React is a popular framework used to make dynamic web apps. It splits HTML into component files using a format called JSX, and uses JavaScript to dynamically render pages.