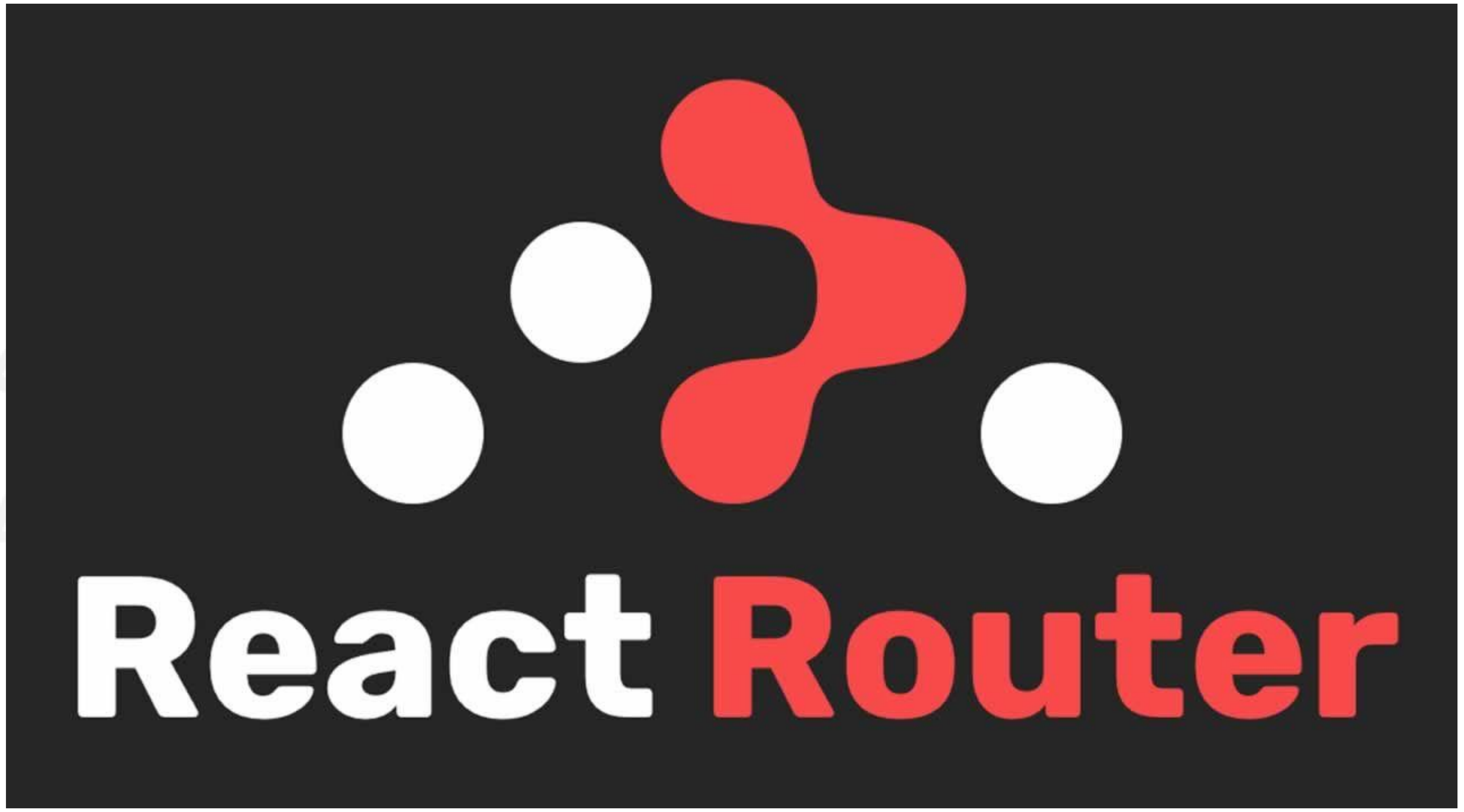
useParams & useHistory

useParams

What, When?

- useParams is one of the Hooks methods that React Router ships with.
- use useParams when you are creating navigational components in React
- According to the definition in React Router doc, useParams returns: an object of key/value pairs of URL parameters. Use it to access match.params of the current <Route>.
  - It means whatever you set up in useParams(ex: title), your params have to match with the <Route path='/path/:title'> .
  - useParams would direct me to a new Route from the current URL.



How to use?

- Before useParams , make sure you have installed React Router in your React app:

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

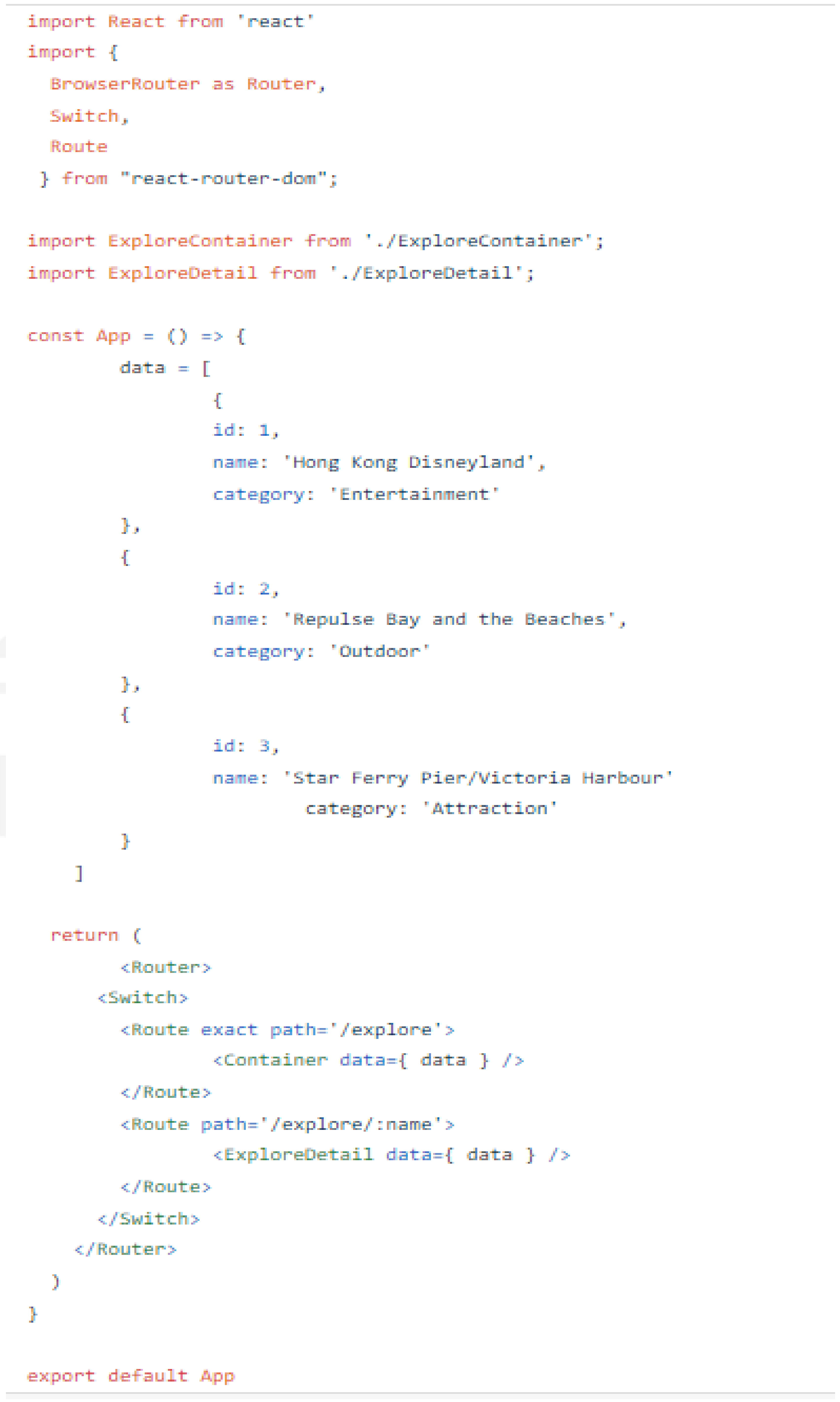
Walkthrough

1. create three components:
    - ExploreContainer — the container of the explore page;
    - ExploreCard — display everything from data (our array of objects)
    - ExploreDetail — display the details of the data.

Walkthrough

2. In our App.js (the parent), make sure you have to import the things we need from react-router-dom.

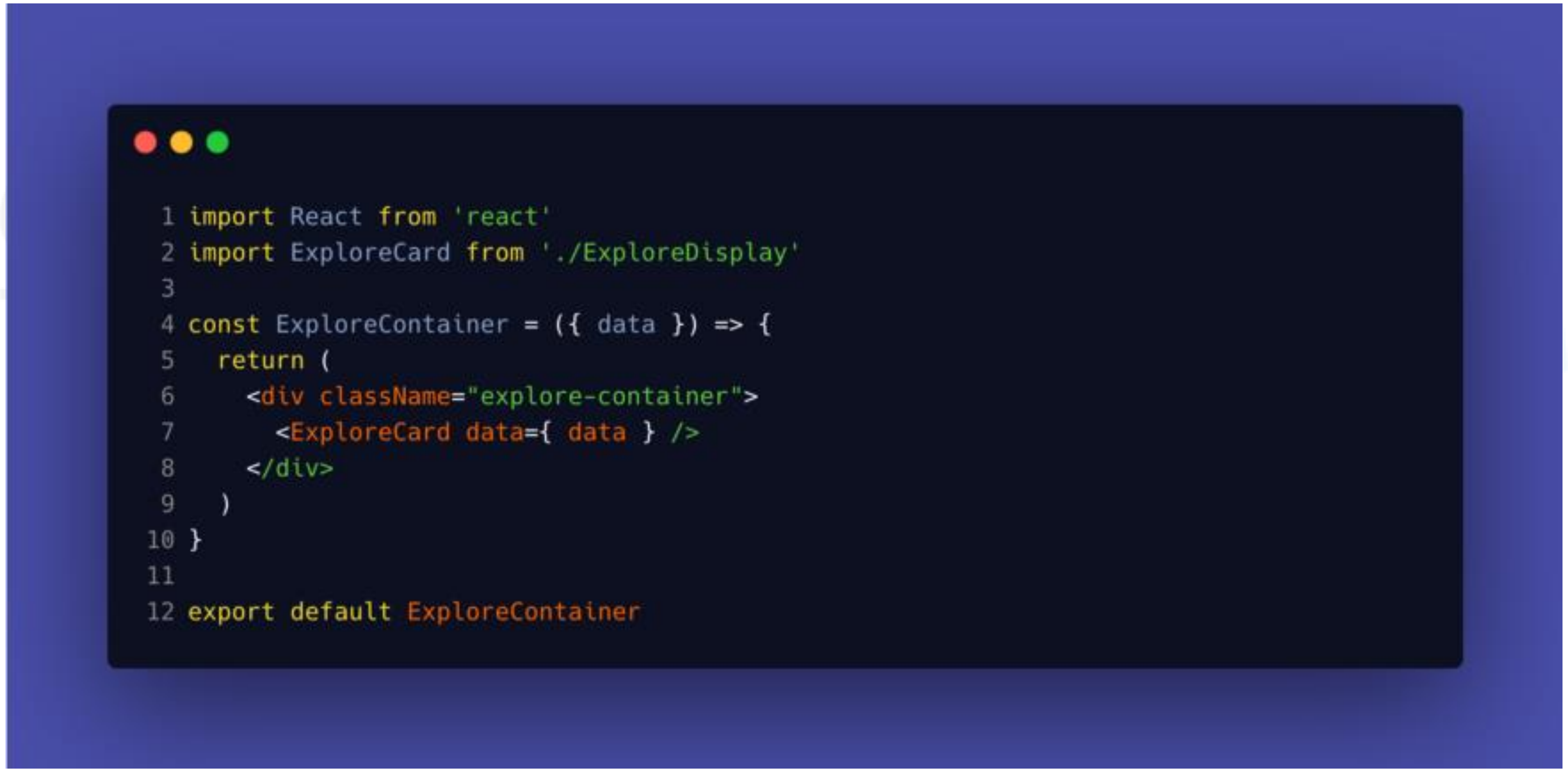
- BrowseRouter is to store the URL and to communicate with the webserver;
- the purpose of Switch and Route is when Switch is rendered, it searches through its children Route elements to match with the current URL. When it finds the match, the Switch will direct to the path in Route .
- In the last Route , it looks quite different from the others, because of /explore/:name . This /:name can be anything, as long as it matches with the parameters/keys of your data. This is also the key when we need to use useParams later on.



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

import ExploreContainer from './ExploreContainer';
import ExploreDetail from './ExploreDetail';

const App = () => {
        data = [
                {
                id: 1,
                name: 'Hong Kong Disneyland',
                category: 'Entertainment'
        },
        {
                id: 2,
                name: 'Repulse Bay and the Beaches',
                category: 'Outdoor'
        },
        {
                id: 3,
                name: 'Star Ferry Pier/Victoria Harbour'
                        category: 'Attraction'
        }
    ]

  return (
        <Router>
      <Switch>
        <Route exact path='/explore'>
                <Container data={ data } />
        </Route>
        <Route path='/explore/:name'>
                <ExploreDetail data={ data } />
        </Route>
      </Switch>
    </Router>
  )
}

export default App
```

Walkthrough

3. Pass down the children In ExploreContainer.js:



```
import React from 'react'
import ExploreCard from './ExploreDisplay'

const ExploreContainer = ({ data }) => {
  return (
    <div className="explore-container">
      <ExploreCard data={ data } />
    </div>
  )
}

export default ExploreContainer
```

Walkthrough

3.`<Link>` to the following card

4. In ExploreCard.js , use map to map out all the data from the data list:

- using `<Link>` to connect what we have from `<Route>` in App.js and letting our apps know we are taking this to another location from the current URL. The `{list.name}` is essential since we would use that for a condition check later when we use `useParams` and make sure it matches the parameters from our Data list.

```
import React from 'react'

import { Link } from 'react-router-dom'

const ExploreCard = ({ data }) => {

return ( <> { data.map(list => ( <div key={ list.id }>

<h4>Name: <Link to={`/explore/${list.name}`}>{list.name}

</Link>

</h4>

</div> ))}

</> )

}

export default GuideCard
```

Walkthrough

5. Step 4: Filter /:name and match with the correct data we want to see with useParams

- Now we are in our final step when we are going to use `useParams` and don't forget to import `useParams`
- First off, we need to make sure the `{ name }` matches with our `/explore/:name parameter`. Basically, you can name the parameter anything you want, but you would want to match with one of the key in your list so there's no confusion.
- The URL bar would add the name of the person. That's the magic of `<Router> -> <Route> -> <Link>` .

```
import { useParams } from 'react-router-dom’

import React from 'react'

import { useParams } from 'react-router-dom'

const ExploreDetails = ({ data }) => {

  const { name } = useParams();

  return (

      <div className="full-detail">

        <div className="explore-container">

{data.filter((list) => list.name === name).map((list) => (

<div className="full-card" key={ list.id }>

<h2>Name: {list.name}</h2>

<h4>Category: {list.category}</h4></div>))}

</div></div>)

}export default ExploreDetails
```

useHistory

- The majority of browsers currently expose a history object on the DOM's Window object, which is used to access the browser's session history and navigate foward and backwards using the history.back() and history.forward() methods (which also function like the back and forward buttons in the browser), and many other methods such as go() and pushState().

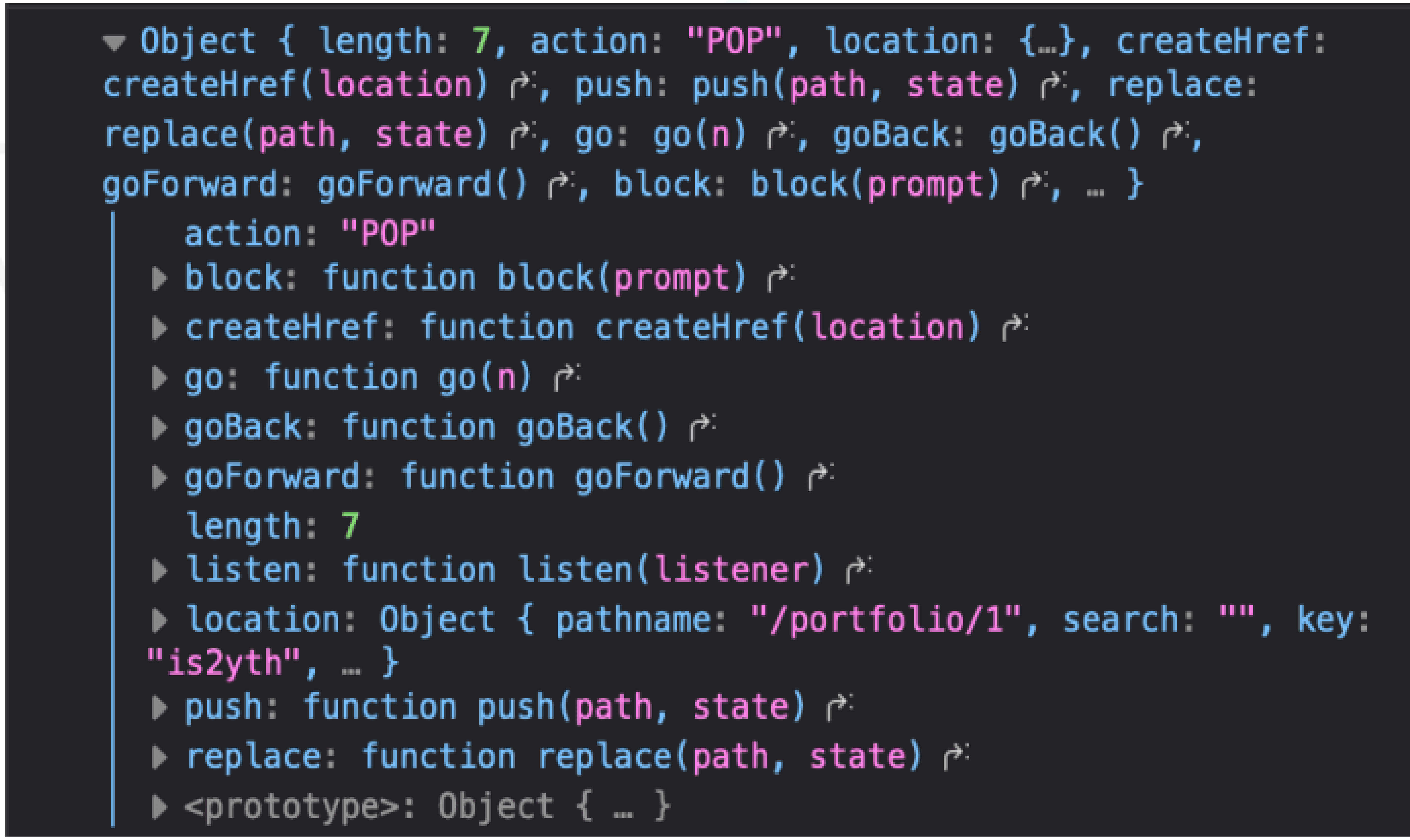
- This hook gives you access to history objects and you have access to several functions to navigate your page. It's all about navigation.

This is how you can use useHistory.

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

const Portfolio = (props) => {
    const history = useHistory();
    console.log(history);

    return (
        <div>
            Portfolio
        </div>
    );
};

export default Portfolio;
```

What's inside history?

- length(number): the length of the page that you visited.
- action(string): (POP, PUSH, REPLACE)
    1. POP: Visiting the route via url, Using history go function(history.goBack(), history.goForward(), history.go())
    2. PUSH: Using history.push()
    3. REPLACE: using history.replace()



```
▼ Object { length: 7, action: "POP", location: {…}, createHref:
createHref(location) ↱, push: push(path, state) ↱, replace:
replace(path, state) ↱, go: go(n) ↱, goBack: goBack() ↱,
goForward: goForward() ↱, block: block(prompt) ↱, … }
    action: "POP"
  ▶ block: function block(prompt) ↱
  ▶ createHref: function createHref(location) ↱
  ▶ go: function go(n) ↱
  ▶ goBack: function goBack() ↱
  ▶ goForward: function goForward() ↱
    length: 7
  ▶ listen: function listen(listener) ↱
  ▶ location: Object { pathname: "/portfolio/1", search: "", key:
  "is2yth", … }
  ▶ push: function push(path, state) ↱
  ▶ replace: function replace(path, state) ↱
  ▶ <prototype>: Object { … }
```

What's inside history?

- .push(pathname: string, state: any)/(location: object): push a path or location to the history stack. There are several ways to use push, and I show the examples below.
- .replace(pathname: string, state: any)/(location: object): this is basically similar to push, but it will remove the existing history and update to the new one. Whenever the user clicks back in the browser after .replace, it will not go back to the previous one.
- .goBack(): move back to the previous history.
- .goForward(): move forward to the previous history.
- .go(delta: number): move to a different index and can specify how many indexes from this position (can be minus or positive)

Using useHistory Hook

- The useHistory hook gives you access to the history instance that we can use to navigate between pages, whether the component has been rendered by React Router or not, and this eliminates the need for using withRouter.

```
import { useHistory } from "react-router-dom";
const HomeButton = () =>{
 let history = useHistory();
 const handleClick = () => {
  history.push("/home");
 }
 return (
  <button type="button" onClick={handleClick}>
   Go home
  </button>
 );
}
export default HomeButton;
```

Resources

- https://medium.com/geekculture/how-to-use-react-router-useparams-436851fd5ef6
- https://www.youtube.com/watch?v=t7VmF4WsLCo
- https://dev.to/raaynaldo/react-router-usehistory-uselocation-and-useparams-10cd
- https://www.pluralsight.com/guides/using-react-with-the-history-api
- https://www.telerik.com/blogs/programmatically-navigate-with-react-router