

Navigation

The React Router

Introduction

- A separate library.
- Allows multiple views and flows in an app.
- Keeps the URL in sync with what's being displayed.
- Supports traditional web principles:
 1. Addressability
 2. Information sharing.
 3. Deep linking.
 - 1st generation AJAX apps violated these principles

Basic routing configuration

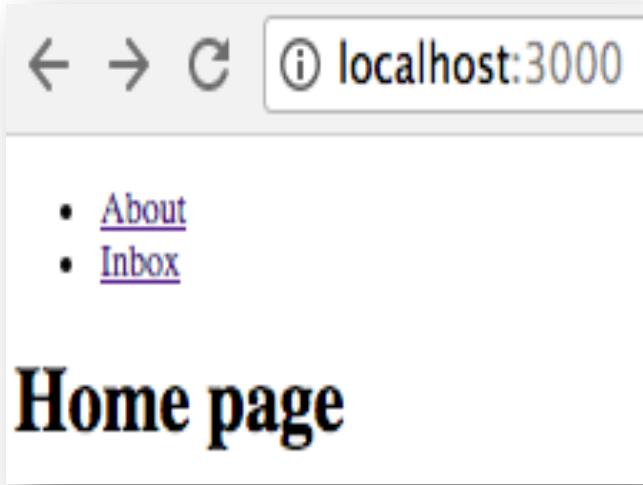
	URL	Components
1	/	App (Home)
2	/about	About
3	/inbox	Inbox

- **Declarative routing.**
- <Switch> - Matches browser's URL address to one of the nested Route **entries** – based on path prop.
 - Matching supports regular expression pattern matching.
 - Use exact argument for precision.
 - Use <Redirect> - to avoid 404-type error.
- ReactDOM.render() passed an app's Router component.
- Ref. src/sample1/

```
class Router extends Component {
  render() {
    return (
      <BrowserRouter>
        <Switch>
          <Route path='/about' component={ About } />
          <Route path='/inbox' component={ Inbox } />
          <Route exact path='/' component={ App } />
          <Redirect from='*' to='/' />
        </Switch>
      </BrowserRouter>
    )
  }
}
```

Hyperlinks

- Use the `<Link>` component for internal links.
 - Use anchor tag for external links - `<a href.....>`
- EX. Ref. src/sample2/



```
<Fragment>
  <ul>
    <li>
      <Link to="/about">About</Link>
    </li>
    <li>
      <Link to="/inbox">Inbox</Link>
    </li>
  </ul>
  <h1>Home page</h1>
</Fragment>
```

The code snippet shows a React component structure. It starts with a `<Fragment>` tag, followed by a `` tag containing two `` tags. Each `` tag contains a `<Link to="/about">About</Link>` or `<Link to="/inbox">Inbox</Link>` component. After the ``, there is an `<h1>Home page</h1>`. The final closing tag is `</Fragment>`. A callout bubble points from the 'About' link in the browser screenshot to the `<Link to="/about">About</Link>` line in the code.

- `<Link>` gives access to other useful router properties.
- Use `<LinkContainer>` when link wraps other 3rd party component, e.g. Bootstrap-React `<Button />`

Dynamic segments.

- Parameterized URLs, e.g. /users/22, /users/12/purchases
 - How do we declare a parameterized path in the routing configuration?
 - How does a custom component access the parameter value?
- Ex: Ref src/sample3/.
 - Suppose the Inbox component shows messages for a specific user, based on the browser URL e.g /inbox/123

.....

```
<Route path='/inbox/:userId' component={ Inbox } />
```

.....

**The colon (:) prefixes a parameter in the path.
Parameter name (e.g. userId) is arbitrary.**

Dynamic segments.

```
1 import React, { Component, Fragment } from "react";
2 import { withRouter } from "react-router-dom";
3
4 class BaseInbox extends Component {
5   render() {
6     return (
7       <Fragment>
8         <h2>Inbox page</h2>
9         <h3>Messages for user: {this.props.match.params.userId} </h3>
10      </Fragment>
11    );
12  }
13}
14 export default withRouter(BaseInbox);
15
```



- **withRouter() function:**
 - **Injects routing props into a component:**
 - `props.match.params.(parameter-name)`
 - `props.history`
 - **Returns a new, enriched component.**

Nested Routes

- EX.: See src/sample4/.

Objective: Given the route:

<Route path='/inbox/:userId' component={ Inbox } />,

when the browser URL is:

1. /inbox/XXX/statistics **then render Inbox + Stats components.**
2. /inbox/XXX/draft **then render Inbox + Drafts components.**

```
4  class BaseInbox extends Component {  
5    render() {  
6      return (  
7        <Fragment>  
8          <h1>Inbox page</h1>  
9          <Messages id={this.props.match.params.userId} />  
10         <Route path={`/inbox/:userId/statistics`} component={Stats} />  
11         <Route path={`/inbox/:userId/draft`} component={Draft} />  
12       </Fragment>  
13    );  
14  }  
15}  
16
```



Nested routes

Aside - The Spread operator (...)

- Allows an iterable (array/object) to expand in places where 0+ arguments are expected.

```
let partOfMe = {first: 'Diarmuid', address: '1 Main street'}
let allMe1 = {surname: 'O Connor', partOfMe, employer: 'WIT'}
// { surname: 'O Connor',
  partOfMe : { first: 'Diarmuid', address: '1 Main street'} ,
  employer: 'WIT' }
```

```
let allMe2 = { surname: 'O Connor', ...partOfMe, employer: 'WIT'}
// { surname: 'O Connor', first: 'Diarmuid',
  address: '1 Main street' , employer: 'WIT' }
```

```
let me = { surname: 'O Connor', first: 'Diarmuid',
          address: '1 Main street', employer: 'WIT' }
let updatedMe = { ...me, address: '2 High Street' }
// updatedMe = { surname: 'O Connor', first: 'Diarmuid',
               address: '2 High street', employer: 'WIT' }
```

Alternative <Route> API.

- **To-date:** `<Route path={...URL path...} component={ ComponentX } />`
- **Disadv.: We cannot pass custom props to the component.**
- **Alternative:**
`<Route path={...URL path...} render={...function....}>`
– **where function must return a component.**
- **EX.: See /src/sample5/.**

Objective: Pass usage data to the <Stats> component.

```
class Stats extends Component {  
  render() {  
    return (  
      <Fragment>  
        <h3>Statistical data for user: {this.props.match.params.id}</h3>  
        <h4>Emails sent (per day) = {this.props.usage[0]} </h4>  
        <h4>Emails received (per day) = {this.props.usage[1]} </h4>  
      </Fragment>  
    );  
  }  
}
```



Alternative <Route> API feature.

```
<Route  
  path={`/inbox/:id/statistics`}  
  render={ (props) => {  
    return <Stats {...props} usage={[5.4, 9.2]} />;  
  }  
/>
```

```
render= (props) => {  
  ... Some logic . . . . .  
  return <ComponentX prop1=... prop2=.... />  
}
```

The <Route> component's own props object is the default parameter for the function.

Aside - Destructuring

- **Assigning the elements of an array or object to variables using a declarative style (rather than an imperative/procedural style).**

```
let obj =  
  { alpha:100,  
    beta: 'Web Dev',  
    gamma: false}
```

Instead of:

```
let alpha = obj.alpha  
let beta = obj.beta  
let gamma = obj.gamma
```

Use :

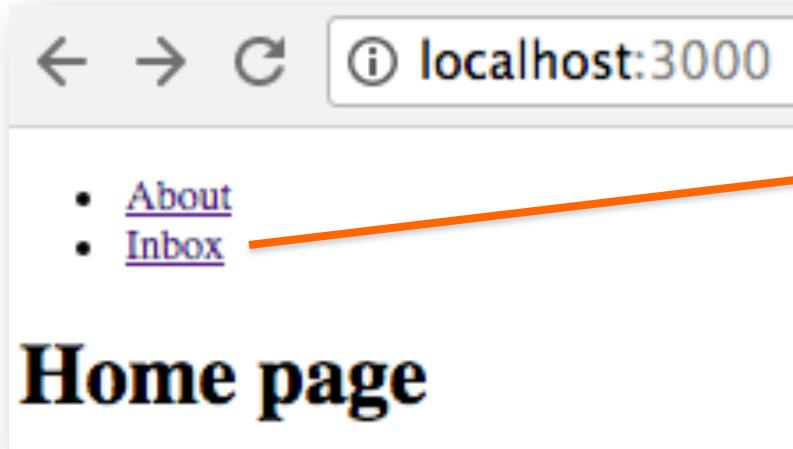
```
let {alpha, beta, gamma} = obj;
```

Can also do:

```
let { beta, gamma} = obj  
let {alpha : foo,  
     gamma : bar} = obj  
// foo = 100, bar = false
```

Extended <Link> API

- **Objective:** Passing additional props via a <Link>.
- **EX.: See** /src/sample6/.

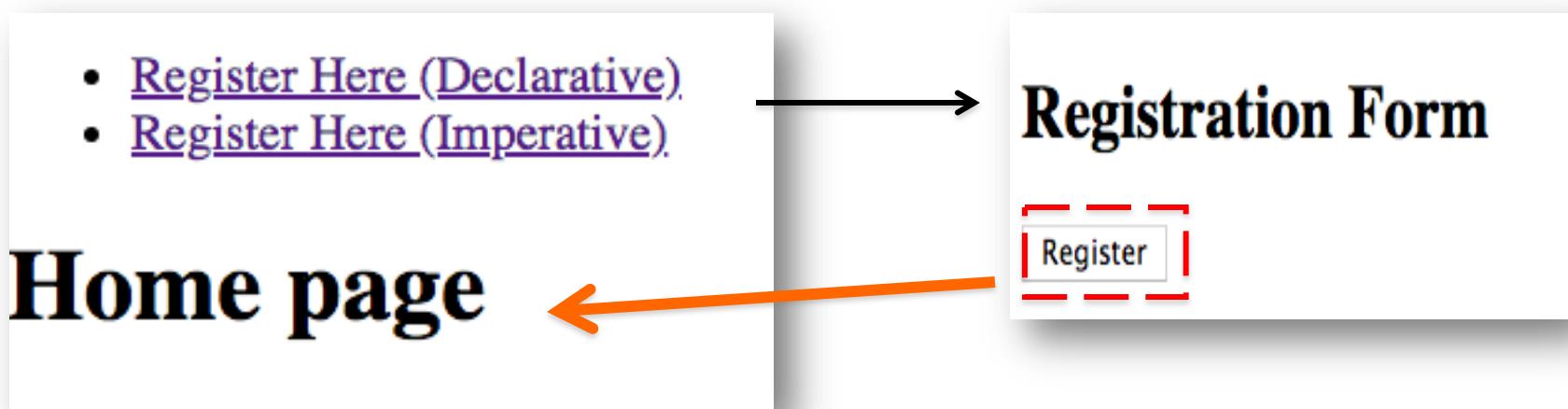


```
<Link
  to={{
    pathname: "/inbox",
    state: {
      alpha: "A",
      beta: "something else"
    }
  }}>
  Inbox
</Link>
```

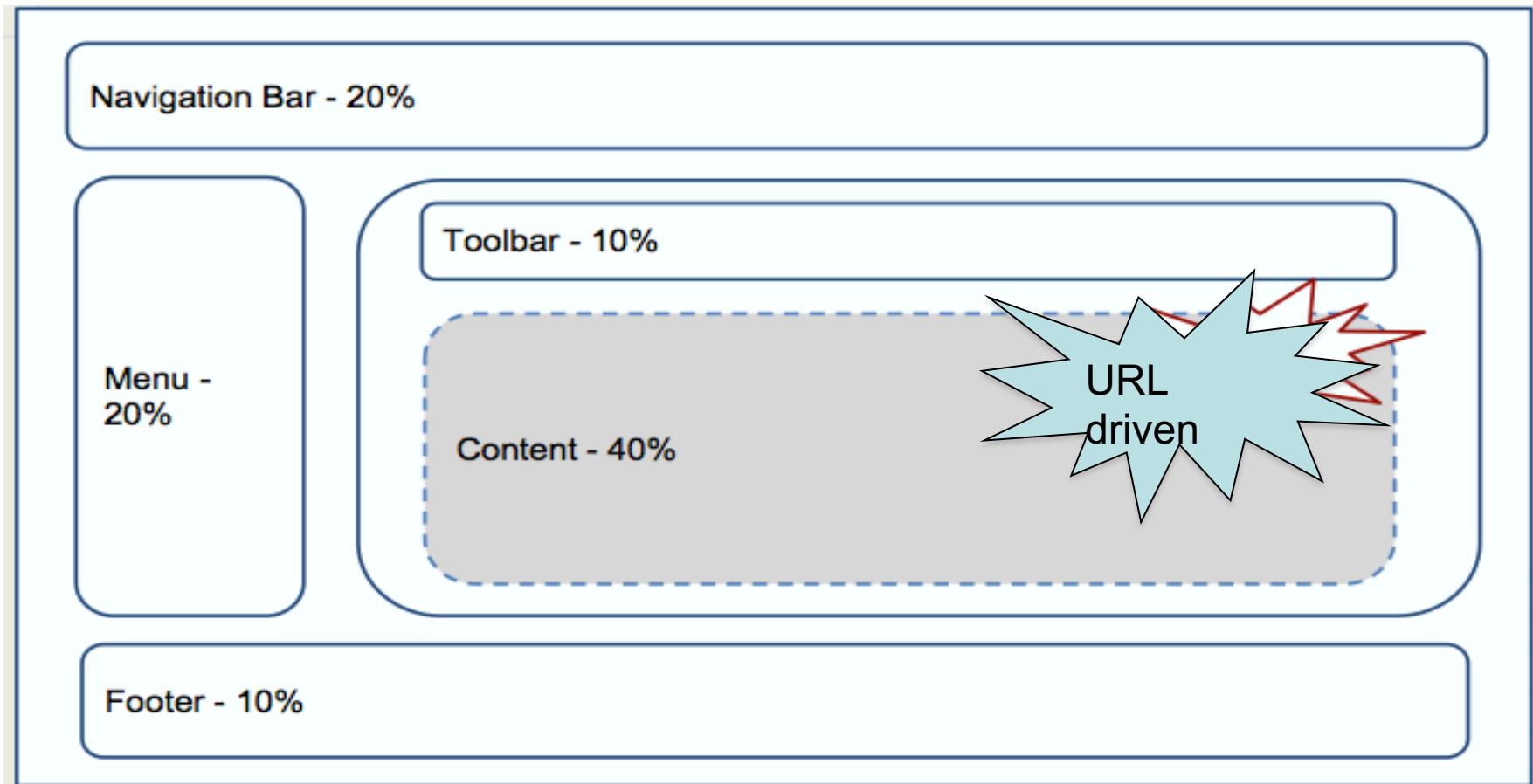
```
class Inbox extends Component {
  render() {
    const {alpha, beta} = this.props.location.state
    return (
      <div>
        <h2>Inbox page</h2>
        <p>Props: ${alpha}, ${beta}</p>
      </div>
    )
  }
}
```

Programmatic Navigation.

- Performing navigation in JavaScript.
- Two options:
 1. Declarative – requires state; use <Redirect />.
 2. Imperative – requires withRouter() ; use this.props.history
- EX.: See /src/sample7/.

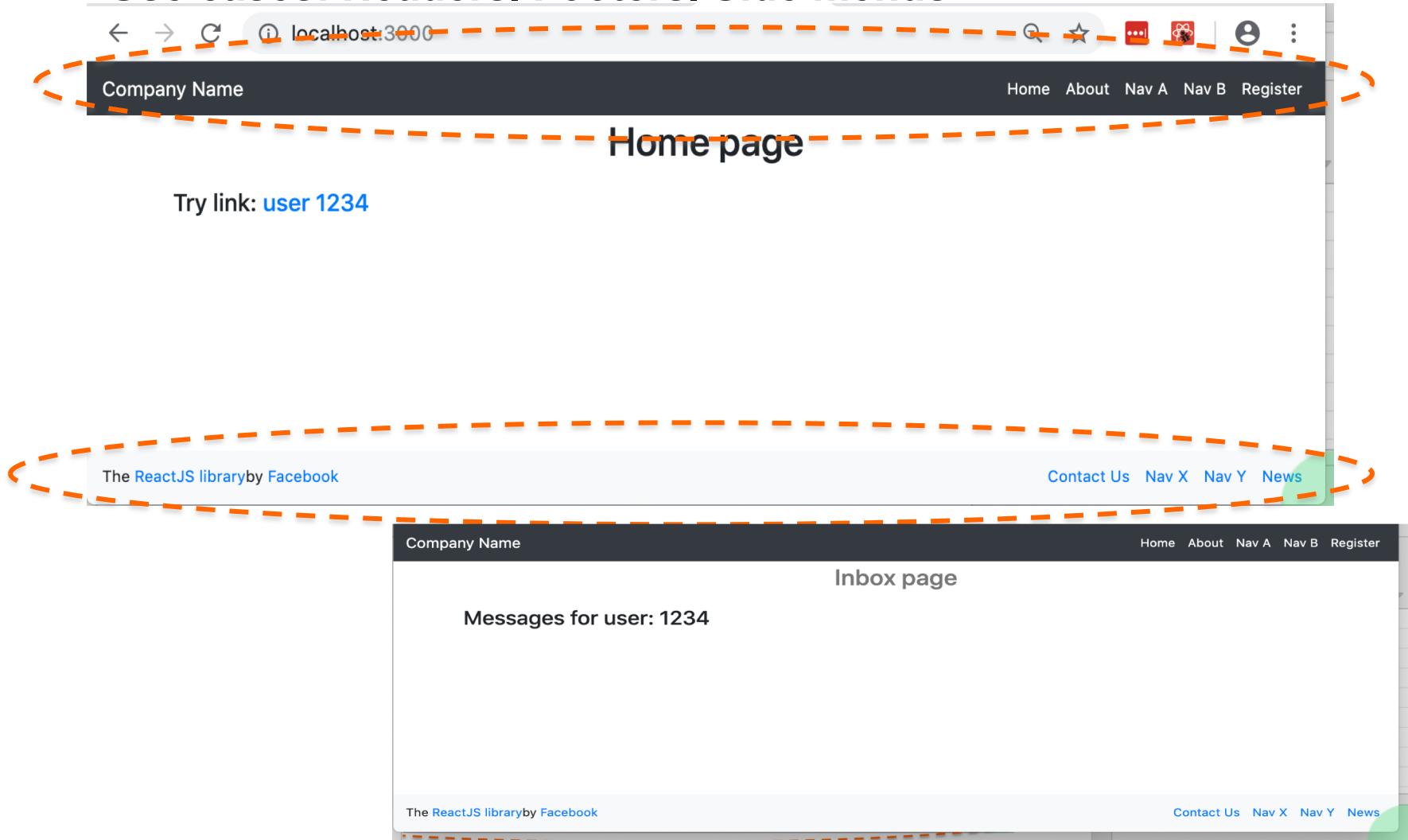


Typical Web app layout



Persistent elements/components

- Use cases: Headers. Footers. Side menus



Persistent elements/components

- Ref. src/sample8

```
class Router extends Component {
  render() {
    return (
      <BrowserRouter>
        <div>
          <Header/>
          <div className="container">
            <Switch>
              <Route path='/about' component={ About } />
              <Route path='/register' component={ Register } />
              <Route path='/contact' component={ Contact } />
              <Route path='/inbox/:userId' component={ Inbox } />
              <Route exact path='/' component={ Home } />
              <Redirect from='*' to='/' />
            </Switch>
          </div>
          <Footer />
        </div>
      </BrowserRouter>
    )
  }
}
```

..... Back to React core

Stateless Functional components

- Many components only require the render() method.
 - The lifecycle methods are still inherited, which impacts performance.
- Use stateless functional components (sfc) where possible.

```
const ComponentName = (props) => {  
  .... body of render method .....  
}
```

> Samples

01 - static component

02 - JSX embedded
variables

03 - component with props

**04 - Component collection
(Iteration)**05 - component
composition

00 - component with props

> 06 stateful component

```
.add("04 - Component collection (Iterati
const frameworks = [
  { name: "React", url: "https://facebook.github.io/react/docs/introduction.html" },
  { name: "Vue", url: "https://vuejs.org/v2/guide/introduction.html" },
  { name: "Angular", url: "https://angular.io/guide/get-started" }
];
const type = "JS client-side Web";
return <JSXCollection frams={frameworks} type={type} />;
})
```

JS client-side Web

- React
- Vue
- Angular

Sample – Class component

```
export default class JSXCollection extends Component {
  render() {
    let list = this.props.frams.map((f, index) => (
      <li key={index}>
        <a href={f.url}> {f.name} </a>
      </li>
    )));
    return (
      <div>
        <h1>{this.props.type}</h1>
        <ul>{list}</ul>
      </div>
    );
  }
}
```

Sample - Stateless Functional components

```
const JSXCollection = (props) => {
  let list = props.frams.map((f, index) => (
    <li key={index}>
      <a href={f.url}> {f.name} </a>
    </li>
  ));
  return (
    <div>
      <h1>{props.type}</h1>
      <ul>{list}</ul>
    </div>
  );
}
export default JSXCollection;
```

```
const JSXCollection = ({frams,type}) => {
  let list = frams.map((f, index) => (
    <li key={index}>
      <a href={f.url}> {f.name} </a>
    </li>
  ));
  return (
    <div>
      <h1>{type}</h1>
      <ul>{list}</ul>
    </div>
  );
}
export default JSXCollection;
```

Summary

- **React Router (version 4) adheres to React principles:**
 - Declarative UI
 - Component composition
 - The event → state change → re-render
- **Main components - <BrowserRouter>, <Route>, <Link>**
- **The withRouter() higher order component.**
- **Additional props: this.props.match.params; this.props.history, this.props.location.**

- **Use stateless functional components where possible.**

