Routing

Overview

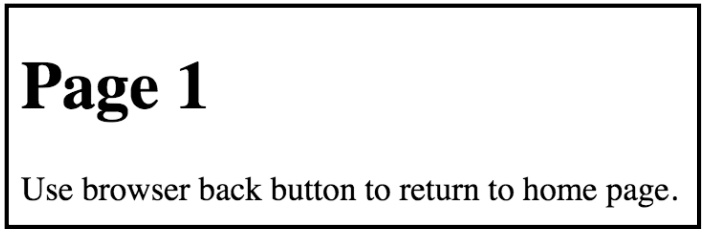
- Option #1: Hash-based routing
  - uses portion of URL after **#** ("fragment")
  - very simple to implement; only requires listening for "hashchange" events
  - don't need to install a library
- Option #2: Push-based routing
  - uses HTML5 History API
  - allows search engines to index multiple pages within site
  - allows all pages in app to be rendered on server side
  - most popular library is **react-router**
- Deciding
  - "Most apps are hidden behind a login screen, and thus have at most one page which needs to be indexed by Google" - James R. Nelson
    - http://jamesknelson.com/push-state-vs-hash-based-routing-with-react-js/
  - hash-based may be preferred when advanced routing features aren't needed

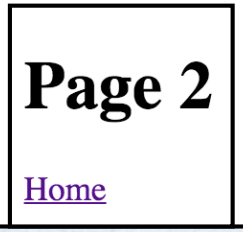
For apps that can use the same URL for all pages, consider using a state property to select a component to render. Much simpler!

Hash-based Example ...

hash-based-routing

```
import React from 'react';
import ReactDOM from 'react-dom';

const Home = () => <div>
  <h1>Home</h1>

  {/* Demonstrate hash change via a link. */}
  <a href="#page1">Page 1</a>
  {' '}

  {/* Demonstrate hash change via code. */}
  <button onClick={() => location.hash = '#page2'}>Page 2</button>
</div>;

const Page1 = () => <div>
  <h1>Page 1</h1>
  <p>Use browser back button to return to home page.</p>
</div>;

const Page2 = () => <div>
  <h1>Page 2</h1>
  <a href="#">Home</a>
</div>;
```





… Hash-based Example

```
function render() {
  const hash = location.hash;
  // Can perform validation here to prevent certain navigations
  // or redirect to a different hash.

  // Select component to render based on hash portion of URL.
  const jsx =
    hash === '#page1' ? <Page1/> :
    hash === '#page2' ? <Page2/> :
    <Home/>; // default page; could use a custom 404 page
  ReactDOM.render(jsx, document.getElementById('content'));
  // Browser back and forward buttons will work.
  // User can change hash in browser address bar.
}

window.addEventListener('hashchange', render);
render();
```

Another option is to dispatch a Flux/Redux action for changing the hash so it can be part of the session history. The new hash would be added to the state and a re-render of the top component would be triggered.

Hash-based Parameters

- URL syntax review
  - `scheme://[user:password@]host[:port][/]path[?query][#fragment]`
- Can access query parameters with `location.search`
  - example: `http://localhost:8080/?alpha=one&beta=2#page2`
  - note that query parameters precede the hash value (a.k.a. fragment)
  - `location.search = "?alpha=one&beta=2"`
- Accessing path parameters is more difficult
  - may need to configure server to ignore path parts after root part
  - path parts precede query parameters
  - consider using a regular expression to extract some path parts

react-router

- Provides push-based routing
- “A complete routing solution for React.js”
- “Keeps your UI in sync with the URL”
- Created by Ryan Florence and Michael Jackson
- https://github.com/rackt/react-router

Hash-based Routing

- In constructor of top component, listen for **`hashchange`** events generated any time the URL hash changes

```
window.addListener('hashchange', () => this.forceUpdate());
```

  - `forceUpdate` is used to cause `render` to be called no props or state have changed

- Add router method to top component

```
router = () => {
  const {hash} =
    getLocationParts(window.location);
  switch (hash) {
    case 'page1':
      return <Page1 />;
    case 'page2':
      return <Page2 />;
    default:
      return null;
  }
};
```

```
function getLocationParts(loc) {
  return {
    hash: loc.hash.substring(1),
    path: loc.pathname,
    query: new URLSeachParams(loc.search)
  };
}
```

- Call router method in render method

```
render = () => <div className="app">{this.router()}</div>;
```

Changing Routes

- Using hyperlinks

```
<a href="#page2">Page 2</a>
```

- Using code

```
document.location.href = '#page2';
```

- What features are sacrificed by not using react-router?

react-router Features ...

- Extracts path parameters
  - `:name` in path -> `this.props.params.name`
- Extracts query parameters
  - `?name=value` in URL -> `this.props.location.query.name`
- Supports use of browser back and forward buttons
- Adds **`routerWillLeave`** lifecycle hook
  - "to prevent a transition from happening or to prompt the user before leaving a route"
- Adds **`onEnter`** and **`onLeave`** hooks
  - invoked after transition has been confirmed
  - could use to require authentication on enter and persist data on leave
- Adds ability to change route programmatically
  - **`transitionTo`** method

... react-router Features

- Nested route support - for child routes
  - navigating to a child route causes
    the child component and all its ancestors to be rendered
- Lazy code loading
  - can delay loading code for child routes and their components
    until their first render
- Server-side rendering of route views
  - using `match` method

Defining Routes

- Routes map a URL to a component to be rendered
- Can use JSX or a JavaScript object
- Properties
  - **`path`** - URL that matches route
  - **`component`** - to be rendered
    - rendered inside parent route component with **`this.props.children`**
  - **`components`** - for multiple, named components
    - each can be rendered inside parent route component with **`this.props[`*`name`*`]`**
  - **`onEnter`** - function called when route is about to be entered
  - **`onLeave`** - function called when route is about to be exited

react-router Example ...

react-router

```
import React from 'react';
import ReactDOM from 'react-dom';
import {IndexRoute, Lifecycle, Link, Route, Router} from 'react-router';

class App extends React.Component {
  render() {
    return <div>
     {this.props.children}
    </div>;
  }
}
const {node} = React.PropTypes;
App.propTypes = {
  children: node.isRequired
};

const Home = () => <div>
  <h1>Home</h1>
  <Link to="/page1">Page 1</Link>
  {' '}
  <Link to="/page2">Page 2</Link>
</div>;
```

could have content here for common content on all pages

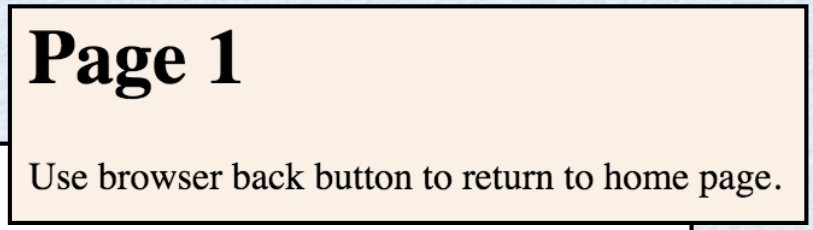
renders matching child route component

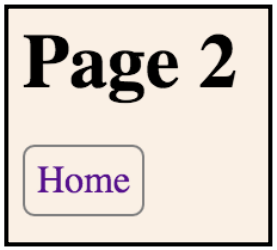
uses CSS to make these links look like buttons

... react-router Example ...

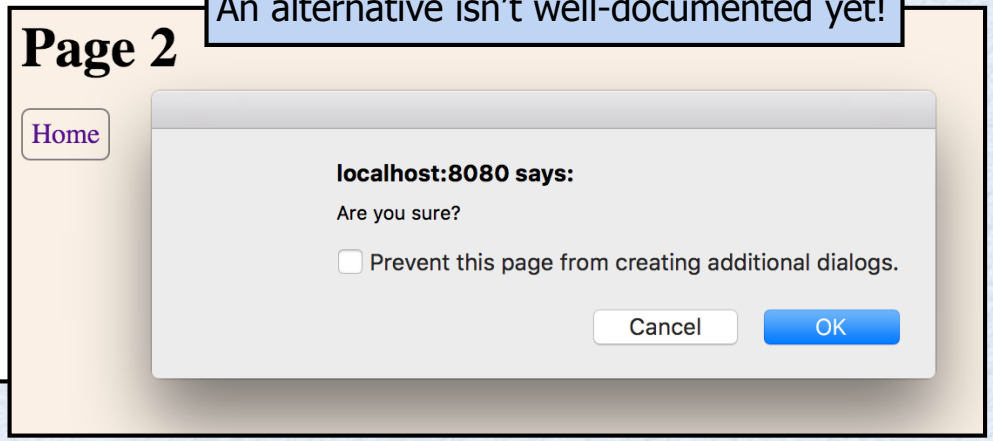
choose one version of **Page2**

```
const Page1 = () => <div>
  <h1>Page 1</h1>
  <p>Use browser back button to return to home page.</p>
</div>;

const Page2 = () => <div>
  <h1>Page 2</h1>
  <Link to="/">Home</Link>
</div>;

// This version demonstrates use of the Lifecycle mixin.
const Page2 = React.createClass({
  mixins: [Lifecycle],
  routerWillLeave(/*nextLocation*/) {
    return 'Are you sure?';
  },
  render() {
    return <div>
      <h1>Page 2</h1>
      <Link to="/">Home</Link>
    </div>;
  }
});
```





The **Lifecycle** mixin is **deprecated** and cannot be used with ES6 classes. An alternative isn't well-documented yet!



… react-router Example

```
// Defining routes using JSX
// Note that the top route can have path of "/"
// if that component renders content
// that should appear for all routes.
const routes = (
  <Route path="/" component={App}>
    <IndexRoute component={Home}/>
    <Route path="page1" component={Page1}/>
    <Route path="page2" component={Page2}/>
  </Route>
);
```

IndexRoute specifies component to be rendered at root path

routes are often defined in a separate **.js** file

```
// Defining routes using JavaScript
const routes = {
  path: '/', component: App,
  childRoutes: [
    {indexRoute: true, component: Home},
    {path: 'page1', component: Page1},
    {path: 'page2', component: Page2}
  ]
};
```

```
ReactDOM.render(
  <Router routes={routes}/>,
  document.getElementById('container'));
```

multiple routes could render the same component with different state and/or props

react-router API Overview ...

- Components
  - **`Router`** - top component to render; manages routes
  - **`Link`** - renders hyperlink that navigates to named route specified with **`to`** prop
    can style differently with CSS
  - **`IndexLink`** - affects when a link to "/" is active
    - see https://github.com/rackt/react-router/blob/master/docs/guides/basics/IndexRoutes.md
  - **`RoutingContext`** - "renders the component tree for a given router state and sets the history object and the current location in context"
- Configuration Components
  - **`Route`** - maps a URL to one or more components to be rendered
  - **`PlainRoute`** - JavaScript object route definition; alternative to JSX
  - **`Redirect`** - configures a redirect to another route so it can be accessed with an alternate URL
  - **`IndexRoute`** - specifies component to be rendered at root path
  - **`IndexRedirect`** - to redirect from URL of parent route to another route
    - "can be used to allow a child route to serve as the default route for its parent, while still keeping a distinct URL"

… react-router API Overview

- Route components
  - named components - related to **`Route component`** property
    - specifies multiple named components that become available by name on **`this.props`**
    - when used, **`this.props.children`** is undefined
- Mixin
  - **`Lifecycle`** - specifies a function to be invoked when about to leave a route DEPRECATED!
    - return **`false`** to prevent leaving
    - return a string to prompt user for confirmation (uses standard browser confirm dialog, so ugly)
- Utilities
  - **`useRoutes`** - to create history objects that know about routing
  - **`match`** - for server-side rendering of routes
  - **`createRoutes`** - creates an array of routes from JSX, a JavaScript object, or an array of either
  - **`PropTypes`** - not documented yet

Also See

- "You might not need React Router"
  - describes a simpler approach to React routing
  - https://medium.freecodecamp.com/you-might-not-need-react-router-38673620f3d