Adding navigation to a React application using

react-router

Agenda

- Understand different navigation architectures
  - single-page vs multi-page
- Learn about client-side vs server-side routing
- intro to the react-router library
- installing, configuring, and using react-router in a react app
  - react-router concepts
    - Configuration
    - Links
    - Nested Views
    - History Types
- Summary

Agenda

Understand different navigation architectures

single-page vs multi-page

- Learn about client-side vs server-side routing
- intro to the react-router library
- installing, configuring, and using react-router in a react app
  - react-router concepts
    - Configuration
    - Links
    - Nested Views
    - History Types
- Summary

Navigation comes in two flavors:

- Multi-page
- Single-page

Terminology

What is a "page"?

A **page** is an HTML file served to the browser, which can itself also serve CSS, JS, and media files through various html tags (such as...?).

What is a "route"?

- a URL pattern that is mapped to a handler
- a **route** tells an app that a certain part of the content/functionality will exist at a certain URL

Multi-page Applications

Multi-page App ("traditional")

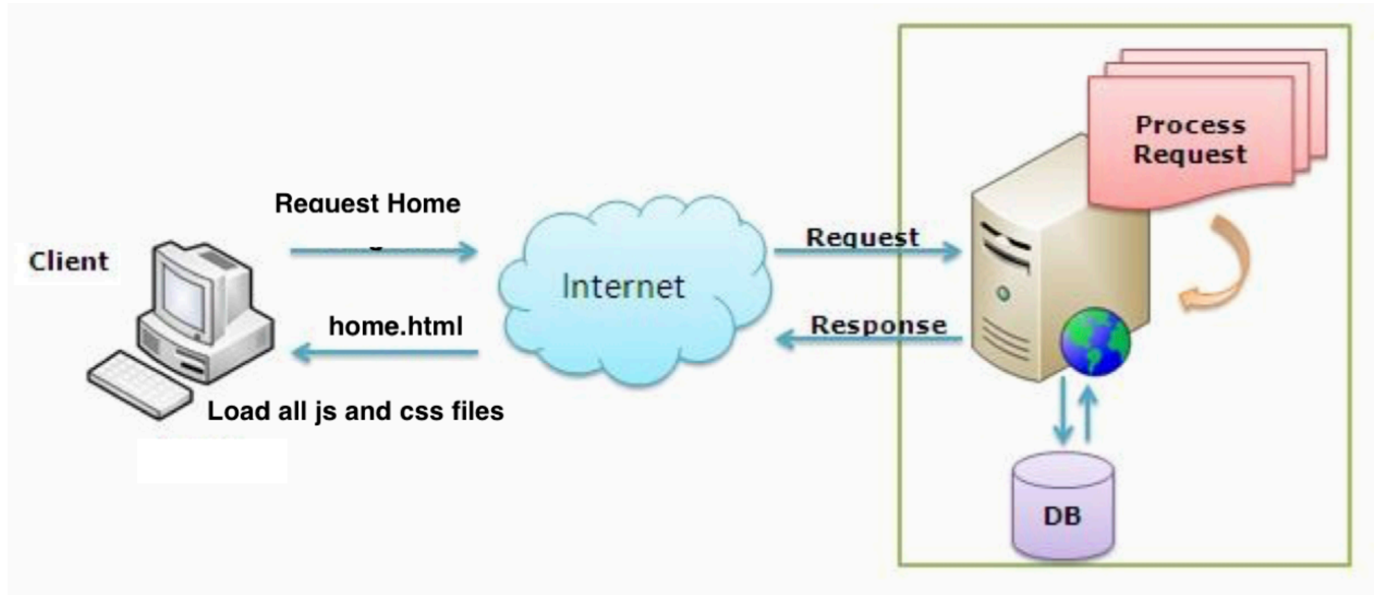
- Browser makes initial request to load the home page
- Anytime a user clicks on a link, another page has to be requested, served and rendered in the browser.
- All JavaScript and CSS files have to be loaded for every page.

Example: see *./resources/multi-page-app/index.html*

Mutli-page Navigation

Requesting the homepage (index.html)

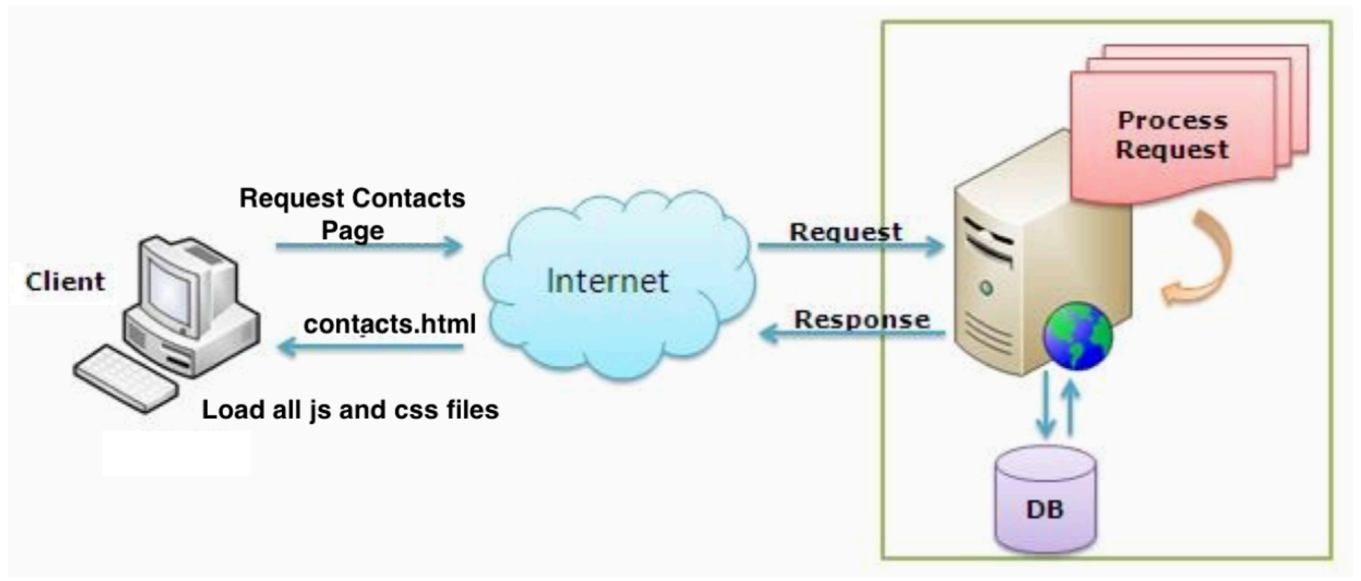
- makes a round trip to the server



Mutli-page Navigation

Requesting /contacts route from index

- ALSO makes a round trip to the server
- looks very much like requesting the initial page
- no optimization made for shared assets or HTML components between the *from* page (index.html) and the *to* page (contacts.html)



Multi-page Navigation Pros & Cons

What are the pros & cons of this software architecture?

...In terms of software architecture design principles:

- Performance
- Maintainability
- Reliability
- Reusability
- Usability (by the user)
- Learnability
- Modularity

Sidenote: interested in software design principles? See: "w3c design guide" https://www.w3.org/People/Bos/DesignGuide/toc.html

Single Page Applications

Single Page App ("modern")

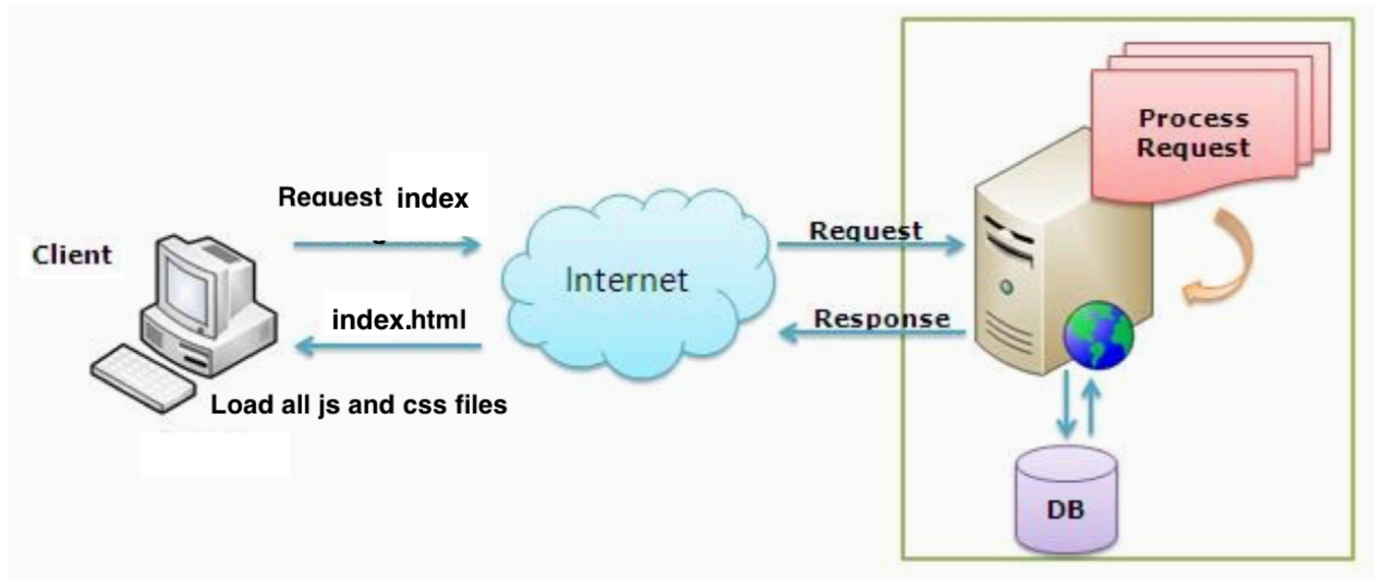
- The **initial** request loads **all** the javascript and css required for going to any route in the application, all at once on intiial load.
- Any additional requests are used to ask for **json data** from the server, if any is needed (via AJAX), and adds/removes elements to the DOM dynamically using Javascript to manipulate the DOM.
- Browser never refreshes
- Initial load time takes a little longer, but subsequent data requests are much faster

Examples:
www.google.com/flights, http://www.zillow.com/ (made with react)
...most modern websites

Single-page Navigation

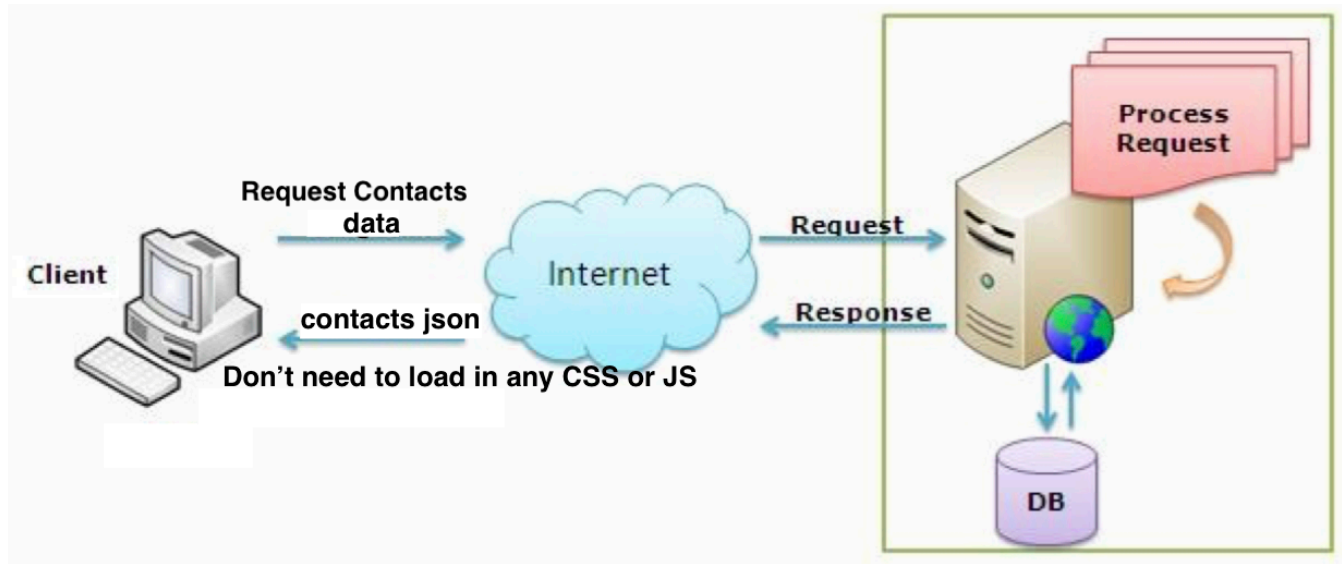
Requesting initial home page (index.html)

- Similar to initial page load on multi-page apps



Single-page Navigation

Requesting /contacts from index



Single Page Navigation

What are the pros & cons of this software architecture?

...In terms of software architecture design principles:

- Performance
- Maintainability
- Reliability
- Reusability
- Usability (by the user)
- Learnability
- Modularity

...We'll revisit this later as we learn more

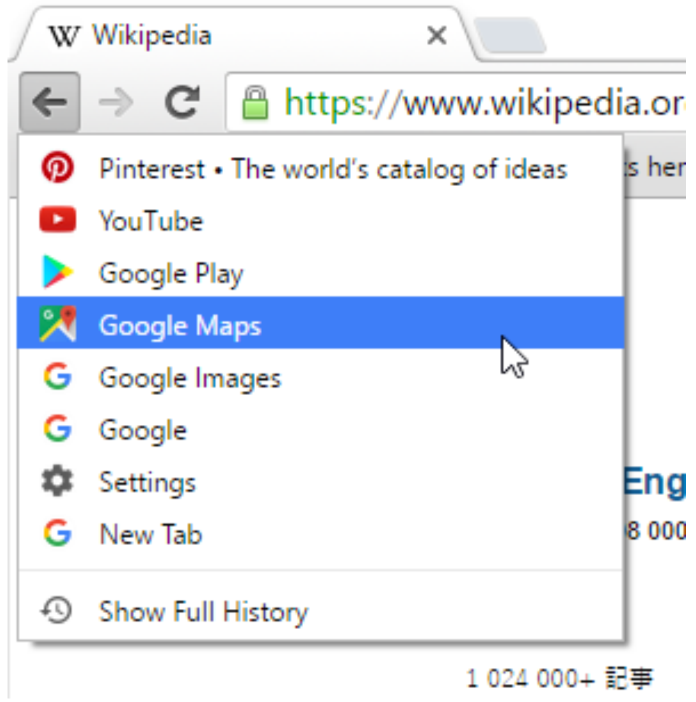
Single Page Navigation

- **Performance**: loads less every navigation --> faster to deliver content
- **Maintainability**: able to keep shared components from one route to another, while removing only what you don't need
  - No need to keep track of where you include partials or shared code per page
- **Reusability**: able to reuse shared components from one route to another
- **Learnability**: more complex to learn and keep track of what components belong to which routes
- **Modularity**: a route corresponds to a set of data and components to manipulate --> a route corresponds to a more highly separated set of concerns
  --> DRY

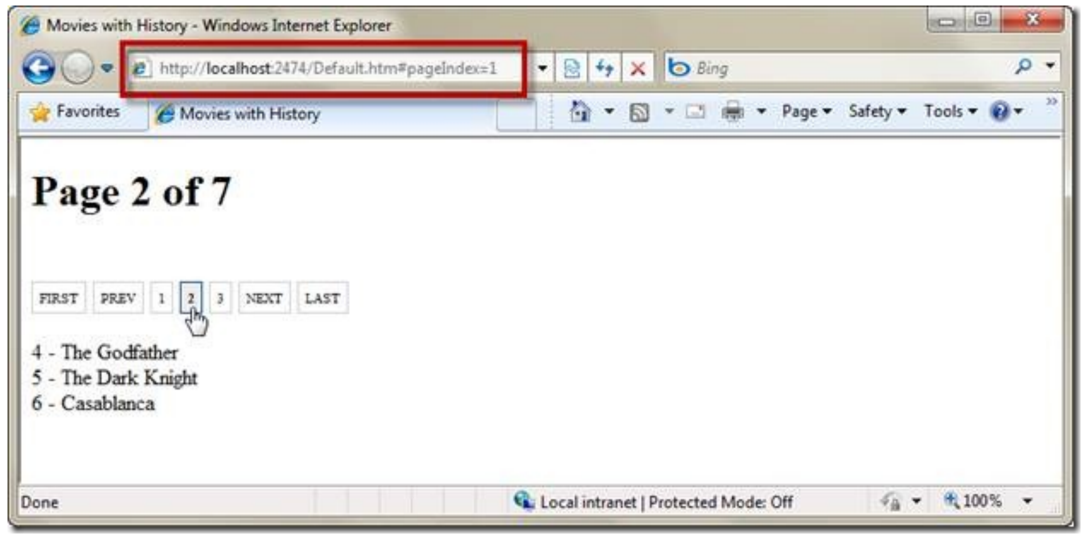
Single page navigation sounds nice, but we want to keep some of the features of multi-page navigation in a single page app

features like…

browser history: ability to use back and forth buttons



Two-way URL tracking: we want the url to update based on changes to the DOM, and for the DOM to update based on changes to the URL


Movies with History - Windows Internet Explorer
Favorites
Movies with History
Bing
Page
Safety
Tools
FIRST
PREV
1
2
3
NEXT
LAST
Done
Local intranet | Protected Mode: Off
100%

http://localhost:2474/Default.htm#pageIndex=1

Page 2 of 7

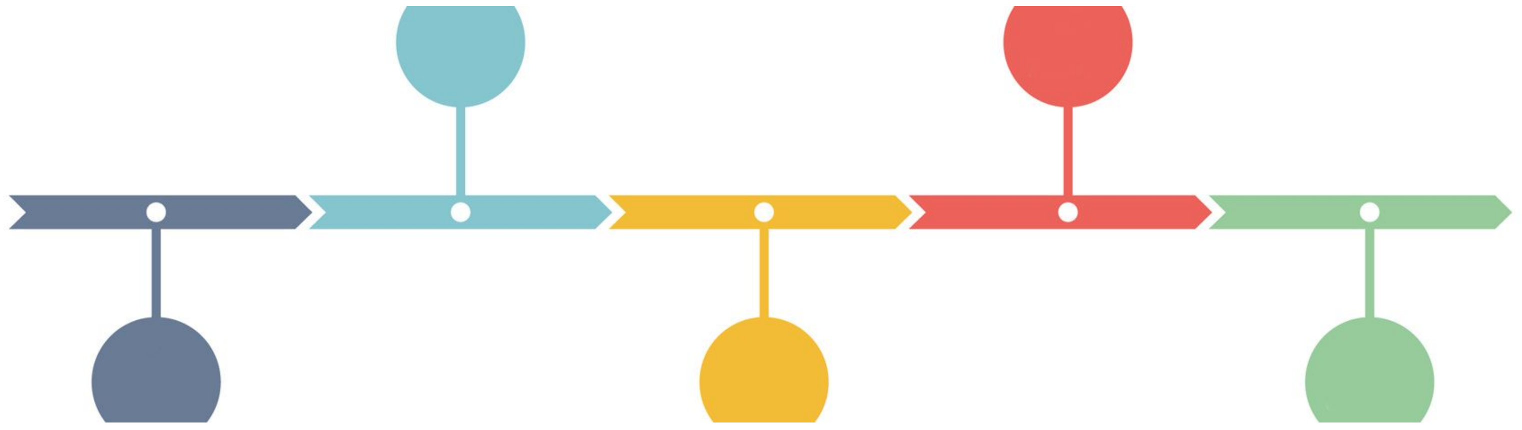
4 - The Godfather
5 - The Dark Knight
6 - Casablanca

Solution 1: ability to manipulate the browser history ("HTML5 History API")

HTML5 History API reference:
https://developer.mozilla.org/en-US/docs/Web/API/History_API

- stores a timeline of historically visited routes -- aka **"states"** -- you've gone to.
- gives developers the ability to modify a website's URL without a full page refresh

```
window.history; // object for manipulating history
```



Navigate history

```
window.history.back(); // move forward through history
window.history.forward(); // move backward through history
window.history.go(-2); // move to a specific point (back 2 pages)
```

Modify history

```
window.history.pushState; // method for adding new entries to history (like a better `window.location = "#new"`)
window.history.replaceState; // method for replacing entries in history
```

Good reference:

https://css-tricks.com/using-the-html5-history-api/.

Not important to understand the ins and outs of HTML5 History methods

Fact: History is offered as an object constructor

```
>  window.history
History {length: 4, state: null, scrollRestoration: "auto"}
>  History
History() { [native code] }
```

- window.history is just an instance of a History **interface**
- We have this ability to create our own history (more on that later)

https://developer.mozilla.org/en-US/docs/Web/API/History

Aside: what did I mean by "interface"?

- An interface generally defines the set of methods and properties that an object that has that interface will have.
- Computer Science term for a more general concept of a "class" in ES6, or function prototype in ES5+. Exists in other languages (Java, etc)

Solution 2: manipulate the hashtag (#) in a browser, typically used to navigate down an HTML file, to implement a history.

- format it like a route:
  - `mywebsite.com/#/some/path/here`
- changes in the hash (aka, the string after the hash tag) do not trigger a browser redirect
- browsers still track changes in the hash, so you can go back and forth
- `window.location.hash = someHashString`

Takeaway

Somehow, we can use HTML5 History (aka Browser History) or Hash History to allow us to have URL tracking and navigate back and forth on an application using the browser.

What does this have to do with React?

React as a Single Page App

- React is in the category of JavaScript libraries that are used to create what's called a **Single Page Application** (SPA)
- Entire app CSS and Javascript is loaded on initial load

```
ReactDOM.render(<EntireApp />, document.getElementById('app'));
```

Somehow, React can use HTML5 History (aka Browser History) or Hash History to allow us to have URL tracking and navigate back and forth on an application using the browser.

react-router

- react-router is what's known as a client-side router that can use Browser History or Hash History to allow for URL tracking and browser navigation.
- Enables you to keep your React application's UI in sync with your URL.
- Takes advantage of the benefits of single page applications

react-router

Powerful features like

- Nested routes
  - `/foo/bar` & `/foo/baz` use components in `/foo`
- Ability to pass URL parameters to components
  - at `/foo?bar=baz`, can use `{ bar: 'baz' }` in your React component
- Supports Hash History and Native Browser History

React router seems really cool! Now how should we learn react-router?...

Read the docs!!!

https://github.com/ReactTraining/react-router

The Docs are constantly improved and are usually the best source for learning a library.

Note: react-router docs can be esoteric. Examples and tutorials are helpful.

Docs helper slides:

react-router provides 3 interfaces:

- `Router` - Parent react component that sets up Routes and manipulates routing state
  - much like `express().router`
- `Route` - React component that specifies which (react) component to render at a specific path

```
<Router>
  <Route path="/foo" component={<SomeReactComponent />} />
</Router>
```

- `Link` - React component within components to change route state

```
<Link to="/foo/bar">Go to This Page!</Link>
```

much like <a>, but is now aware of the router it was in (and stores its history accordingly)

Using Link to Navigate

- This is nice but we don't want our users to type in a url every time they want to go to a different page.
- For this, react-router provides us with another Component called:
  - `<Link to={URL_PATH}>__LINK_TEXT_HERE__</Link>`
- Let's add this to our sample application

Example of using `Link`

```
// client.js
ReactDOM.render(
  <Router>
    <Route path="/" component={<Home />} />
    <Route path="/about" component={<About />} />
    <Route path="/contact" component={<Contact />} />
  </Router>
  , document.getElementById('app')
);


//Create Root.js

import { Link } from 'react-router';

class Root extends Component {
  render() {
    return (
      <div>
        <h1> Welcome to the Home Page</h1>
        <ul role="nav">
          <li><Link to="/">Home</Link></li>
          <li><Link to="/about">About</Link></li>
          <li><Link to="/contacts">Contacts</Link></li>
        </ul>
      </div>
    );
  }
}
```

- Histories
- react-router also includes a version of the npm library `history`, including:
  - browserHistory
  - hashHistory
- react-router defaults to using hash history to store the history of your user's session

react-router - Installation

From https://github.com/ReactTraining/react-router:

- In your express react application

```
// using an ES6 transpiler, like babel
import { Router, Route, Link } from 'react-router'
```

react-router - Installation

From https://github.com/ReactTraining/react-router:

- In your express react application

```
// using an ES6 transpiler, like babel
import { Router, Route, Link } from 'react-router'

// not using an ES6 transpiler
var Router = require('react-router').Router
var Route = require('react-router').Route
var Link = require('react-router').Link
```

Route Configuration

```
<Router>
    <Route path="URL_PATH" component={COMPONENT_TO_RENDER} />
</Router>
```

- Disregard any hashHistory error for the time being, we'll cover this in a little bit.

Parameterized routing in react-router: `this.props.params`

```
<Router>
  <Route path="/foo/:bar" component={App} />
</Router>
```

- App can access the value of `bar` through `this.props.params.bar`

Route wildcard: `*`

- `*` in a route means "any value"
- `<Route path="/foo/bar*" component={Baz} />` will render Baz for
    - /foo/bar
    - /foo/barzoo
    - /foo/bar!

Primer: children

A given react component has access to children through props:

```
render() {
  return (<div className="App">{this.props.children}</div>);
}
```

Every react component has access to `this.props.children` - refers to the contents inside of its tag:

```
<App>Children of App go here</App>
```

Children example

```
// Sister.js
export default () => {
  return <h1 className="Sister">I am a sister!</h1>;
}

// Brother.js
export default () => {
  return <h1 className="Brother">I am a brother!</h1>;
}

// Mother.js
export default () => {
  return <div className="Mother">{this.props.children}</div>;
}
```

what do you think `<Mother><Sister/><Brother/></Mother>` would return?

```
ReactDOM.render(
  <Mother>
    <Sister />
    <Brother />
  </Mother>
);

// returns

<div class="App">
  <h1 class="Sister">I am a Sister!</h1>
  <h1 class="Brother">I am a Brother!</h1>
</div>
```

See doc example for example usage

https://github.com/ReactTraining/react-router#whats-it-look-like

```
render((
  <Router history={browserHistory}>
    <Route path="/" component={App}>
      <Route path="about" component={About}/>
      <Route path="users" component={Users}>
        <Route path="/user/:userId" component={User}/>
      </Route>
      <Route path="*" component={NoMatch}/>
    </Route>
  </Router>
), document.getElementById('root'))
```

Idea of:
* Nested routes (/users, /user/:userId)
* Using a component as a 404 page (NoMatch)

Exercise

- Create three simple components that will show the /posts, /posts/:postId page.
- Configure the Router so that the appropriate Component is rendered for the corresponding url paths.
- Test it is working by going to localhost:3000/#/posts or localhost:3000/#/posts/2
- Consult the docs to figure out how to have the homepage (/) redirect to /posts

Why does my url look funny?

- You should see a hash at the end of your url.
- This is what the router uses to manage history.
- Later we'll discuss different history implementations and how to remove the hash.

Link - Problem Statement

- Well this would be nice if only we had a single link to another page.
- But these are navigational links we want to have on every page.
- We don't want to copy and paste this code on every other page, because we could have a lot of pages.
- We could build it as a component and include it in each of the views, but this doesn't scale very well as the application gets larger.
- Let's see what ReactJS gives us to handle this.

Solution: Nesting-Routes

- Used when you want to have a parent-child relationship between routes.
- React-router uses the router to nest child views inside a parent view by nesting routes in the configuration.

Example - Routes inside of Routes

```
// Inside of Root.js

import { Link } from 'react-router';

class Root extends Component {
  render() {
    return (
      <div>
        <ul role="nav">
          <li><Link to="/">Home</Link></li>
          <li><Link to="/about">About</Link></li>
          <li><Link to="/contacts">Contacts</Link></li>
        </ul>
        {this.props.children /* more on this to come */}
      </div>
    );
  }
}

export default Root;
```

Example - Routes inside of Routes

```
//Add to App.js

<Router history={hashHistory}>
  <Route path="/" component={Root}>
    {/* doing this makes the below Components (Home, AboutUs, Contacts) children of `Root` */}
    <Route path="/" component={Home}/>
    <Route path="/about" component={AboutUs}/>
    <Route path="/contacts" component={Contacts}/>
  </Route>
</Router>
```

- My home page stopped working!!!

Nesting Routes - IndexRoute

- When nesting Routes, we can't have a child have the same path as the parent.
- But we want to have the "/" display both "Root" and "Home"
- Solution: The `<IndexRoute component={MY_COMP}/>` component (don't forget to import IndexRoute).

```
//Add to App.js

<Router>
  <Route path="/" component={Root}>
    <IndexRoute component={Home}/>
    <Route path="/about" component={AboutUs}/>
    <Route path="/contacts" component={Contacts}/>
  </Route>
</Router>
```

Exercise - children inside of children.

- Create three contact links that are to be displayed under the Contacts Page
    - President
    - CEO
    - Founder
- These links should be displayed only when the user is on the Contacts page.
- Each time the user clicks on a contact link, the page should display some information about that contact.
    - eg: clicking 'President' should give me some information about the president
- If the user clicks on 'Home' or 'About Us' all contacts information should disappear.
- (Hint: create three more components and some children routes under the 'contacts' route)

Route Parameters

- Sometimes we want to have certain parts of our URL be dynamic in nature.
  - eg: /post/view-detail/12345
  - The '12345' could be anything in this case
- To allow for this react-router uses a placeholder url component.
  - eg: /post/view-detail/:postId
- react-router also passes this information into your components via the this.props.params object
  - eg: this.props.params.postId

This is often used to make DB calls when a page loads, allowing for loading of application data.

Example - Route Parameters

- We're going to modify our Router to have a route that contains some route params.
- Let's create a new Post component that will display the id for a particular post.

```
// Create Post.js
import React, { Component } from 'react';

class Post extends Component {
  render() {
    return (
        <h1>Getting details for Post {this.props.params.postId}</h1>
    );
  }
}

export default Post;
```

Example - Route Parameters

```
// Add to App.js

import Post from './Post';
...

          <Route path="/about" component={AboutUs}/>
          <Route path="/post/view-detail/:postId" component={Post}/>
```

- Go to localhost:3000/#/post/view-detail/abc
- change the 'abc' to '123' and verify your page updates

Browser History

- When setting up our route configuration earlier, we are getting an error related to 'hashHistory':

```
browser.js:49 Warning: [react-router] `Router` no longer defaults the history
prop to hash history. Please use the `hashHistory` singleton instead.
```

- We need to define the 'history' attribute on our 'Router' Component.

```
<Router history={HISTORY_TYPE}>
```

History Types

react-router provides 3 different implementations for how to handle browser history:

- `browserHistory` (`/posts`) - Uses HTML5 History browser API to navigate like a multi-page site. Unsupported on old browsers and needs some slight server configuration, but otherwise the **recommended history choice**.
- `hashHistory` (`/#/posts?_k=p7y44z`) - uses hashes and a hash query parameter to emulate moving around on the page without a page reload. Easy to get started, but not ideal for a production application.
- `createMemoryHistory` - Often used in server-side rendering and testing, this allows you to create a custom browser history without using the address bar.

Example - Browser History

Up to this point we'd been using `hashHistory`. Change `index.js` to use `browserHistory`

```
// index.js
...
import { Router, Route, browserHistory } from 'react-router'
...

  <Router history={browserHistory}>
...
```

Load up http://localhost:3000/, your application should now navigate just like a native multi-page web application.

Resources:

https://github.com/reactjs/react-router-tutorial/tree/master/lessons