Routers for Architects

Building maintainable, flexible and extensible routing architecture in web applications

Presented by
Peter Pavlovich
Chief Technology Officer
Censinet, Inc.
pavlovich@gmail.com

Introductions ...

- Peter Pavlovich
  - Front-End Evangelist and Technology Addict
  - Current: Chief Technology Officer, Censinet
    - Vue/React/Angular/Meteor/Polymer/Aurelia, Node, Scala/Java, OrientDB/Titan/MongoDB
- Entrepreneur, Open Source Contributor and Community Member
- Instructor, Mentor, Speaker and Leader.
- http://linkedin.com/in/peterpavlovich
- pavlovich@gmail.com

Current Work

- Greenfield UI, middle tier and backend
  - Vue/React/Rails/Postgres
  - Angular/Meteor/Node/MongoDB/Postgres
  - Java/Embedded C
- Open Source contributor
  - Meteor Metadata-driven Modeling Pkg.
  - Angular/Meteor Forms Package
  - Multiple 'side-projects'

Current Work

- Mentor for 55 startup companies co-located within "Greentown Labs", an incubator lab facility in Boston, MA
  - Focused on Green Energy & Clean Tech.
  - Office and Lab space
  - Access to expert advisors/mentors
  - Training, funding help, general support.

Basic Concepts

What is a router?

- Provides different response actions depending on parameters passed to it:
  - Protocol
  - Server Address
  - Port
  - Path
  - Route Parameters
  - Query Parameters
  - Bookmark / Hash / Index

Typical Rouer Input

- Typical Universal input format:
  - <protocol>://<server>:<port>/...
  - ... <path>/<p1>?<q1=v1>&<q2=v2>#<hash>

Definitions

Definitions: 'Routing System'

- One or more 'Routers' acting together to monitor and service 'requests'.
- Servicing a request involves analyzing, processing and reacting to the response(s).
- 'Router's can live on different physical or virtual systems.

Definitions: 'Router'

- Routers are software modules which:
  - Establish a communications 'channel'
  - Intercept 'requests' from a 'caller'
  - Interpret the 'request'
  - Process the 'request'
  - Receive one or more 'responses'
  - Update 'caller' based on the 'response(s)'

Types of Routers

By Design Pattern

- Stateless
  - Each request results in the same response
- Stateful
  - Request history alters responses

By 'Physical' Location

- Server-side
  - Usually stateless:
    - Each request yields same response
  - Client will typically reload entire app each request unless it is an AJAX request system.

By 'Physical Location'

- Client-side
  - Typically in a SPA
  - App code all loaded at once.
  - Reactions to responses
    - Populate existing view layout region(s).
    - Can replace layout or portions too.
    - Typically no complete screen repaints.
    - Needs to be able to detect if request is intended for currently loaded app or if a new app (via a server-side request) is needed

Router System options

- Pure Server-side driven routing:
  - All URL processing on the client results in a full request to the server with a complete server-side rerendering and client-side reloading of the resulting HTML into the browser.
- No 'data-only' client/server requests.

Router System options

- Server-side dominant routing:
  - All context switching (one view to another) requests are processed via a URL request to the server resulting in a complete server-side rerendering and client-side reloading of the resulting HTML into the browser.
- Some 'data-only' client/server requests via AJAX to update/refresh data for the current context/view.

Router System options

- Pure Client-side driven routing:
  - Hit initial server-side URL to load SPA codebase which includes client-side router.
  - Server-side router/server provides initial routing to retrieve code. Then only provides REST request servicing.
- Client-side router handles all URL interpretation/processing
  - Client side intercepts all URL requests, determines if it is meant for current app or a different website. If this app, it prevents the browser from issuing a GET/POST to the server. Instead, the router interprets the URL and causes the MVC framework or component-based framework on the client to activate and render different views/components and place them into specified regions on the current or a newly selected layout/component.

Router System options

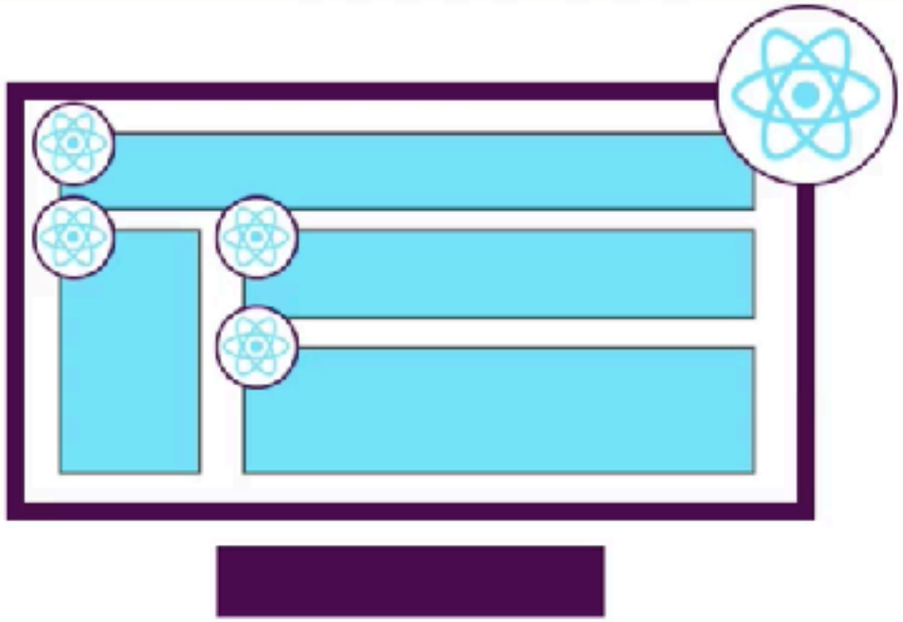
- Hybrid Client/Server-side driven routing:
  - System is composed of multiple SPAs, typical one for each major functional area (e.g. 'Accounting', 'HR')
  - URLs are interpreted by the client first.
    - URLs targeting currently loaded SPA are serviced by client-side router. No page reloads/full rerendering needed. AJAX requests for new data are sent to server router for servicing.
    - URLs targeting a different SPA are sent to be serviced by the server router. Typically will result in a full page reload to load and activate a different SPA.

Client-centric decision

Two Kinds of Applications

Single Page Applications

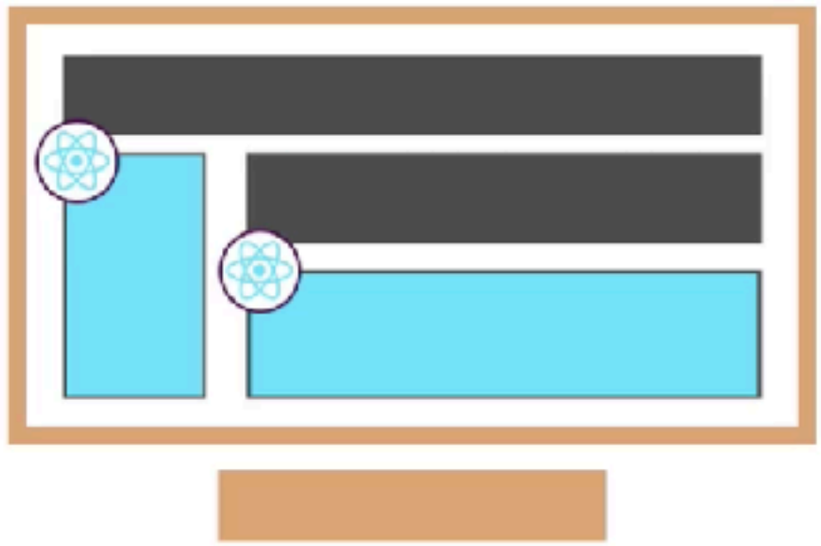
Only ONE HTML Page, Content is (re)rendered on Client



Typically only ONE ReactDOM.render() call

Multi Page Applications

Multiple HTML Pages, Content is rendered on Server



One ReactDOM.render() call per "widget"

Our Focus today?

Client-side SPA routing

Motivation

Abstracting SPA routing systems

Router architecture?

- It is not the complicated, right?
- We have chosen our framework
- It has a ‘default’ router
- Just use that and follow the examples!

Why study router architecture?

- Learning a single app framework is like learning single sentences in a foreign language.
- You might be able to communicate, but you miss the bigger picture and how the sentences fit together optimally.

Patterns are important

- Capabilities and how they are used are patterns.
  - Even if your router doesn't directly support one, you may still benefit.
    - Might be a plug-in or you could write one!
    - Knowing what is available helps make better, more informed decisions
    - Might even affect your choice of framework

Patterns are important

- If your router is a close match but is missing some capability you deem desirable (now you are aware of it!):
  - Look for or build a plug-in!
  - Contribute back to your community!

Design Decisions

Do I Need a Router?

What? Go Routerless?

- Start with:
  - Nested tree of components
    - Each maintains one or more vars
    - Displays/inserts selected component(s) based on var values
- UI components/links get/set var values

What? Go Routerless?

- Continue with:
  - Links/buttons call functions:
    - set values conditionally (auth, etc)
    - Fetch data async
    - Set up animation attributes, etc

So, yes ... it is possible

- But you will eventually build a router!
  - Might be a 'simple' one
  - Still performs the same function
    - Takes input parameters
    - Performs different actions
    - Displays different content

Router Benefits

- A good router / route schema helps us control:
  - how the application behaves
  - optimize memory usage and load times
  - Improve security
  - Provide bookmark services
  - Accurately represent the users intention.

Router Benefits

- Bookmarking allows users to store references to particular views under their own names and then navigate directly to those views when they desire
- It also provides convenient ways for search engines to index your page or application providing direct access to information which is relevant for a given search.
- SEO is a huge field and is reliant upon externally linkable apps.

Router Benefits

- Can provide a full state machine to provide convenient consistent and centralized management of your application logic and flow
- Provide a way to control the history stack built by and used by your browser.
- You can control which transitions or user actions result in a modification to the history stack allowing you to insert ignore or delete entries in that history stack.

Router Benefits

- Having a good and logical structure to your URL schema also provides a visual and convenient way to convey the logical structure of your application to your users.
- It also provides a way to pass pertinent information to components via a globally available string which can be parsed to provide that information to components and other code blocks in a convenient and consistent fashion.
- It can also provide a convenient and centralized location for code which must run and return data to the application prior to loading particular components or making them visible.

Best option?

- Most routers do the same sorts of things:
  - Have similar mechanisms
  - Have similar components
  - Use similar architectural patterns

Best option?

- Understand the patterns & what is available.
- Pick the best option for your app
- Write 'interface' code generically as possible.
- Insulate yourself via encapsulation

Design Patterns

What is a Pattern

- For our purposes:
  - A commonly implemented capability
    - Usually identifies/encapsulates a:
      - "good idea"
      - "best practice"

What is a Pattern

- For our purposes:
- Typically has:
  - Many instantiations
  - Common weaknesses/strengths

Router Design Patterns

Pattern Categories

- Configuration
- Template Extensions
- Route Controller
- Route Definition DSL / JSON / Objects
- Lifecycle Hooks
- State Management Extensions

Configuration Patterns

Browsers send URL changes to the server

- Navigating on a SPA in a browser:
  - Normally URL change = server round trip
  - We want to suppress this behavior
  - We want the client-side to be in control

Solution?

- The two URL processing techniques that allow software to intercept URL requests typically only handled by the client are known as:
  - HTML5 style (bypass browser)
  - Hash-bang style (trick browser)

Preventing reloads

- Method 1:
  - Use # tags (bookmarks)
    - Only changing the URL portion following the # does NOT cause a browser to reload the page
    - Framework watches the URL and if it changes, re-reads the # portion and reacts accordingly.

Hash-bang style

- http://site.com#!/products/list
  - Original/subsequent requests to server all results in loading from: http://site.com/index.html
    - Client-side router can read URL and react to changes.
- Pros
  - Older style: supported fairly universally
  - Uses 'bookmark' feature of most browsers
  - The 'bookmark' is everything after the #

Hash-bang style

- Pros (continued)
  - All URLs in the app use same prefix to #
  - The browser thinks: 'URL is unchanged'
    - Bookmarkable / linkable
    - No server-side configuration needed for it to work.
- Cons
  - Looks 'ugly'
  - Limits design (can't use bookmarks 'regularly')

Hash-bang Style

- Server-side configuration: None needed.
- Client-side configuration:

```
var app = angular.module("MyApp", [])
  .config(function(…,$locationProvider){
      $locationProvider.hashPrefix(‘!’);
      $locationProvider.html5Mode(false);
  }
```

Preventing reloads

- Method 2:
  - Use the HTML5 history API
  - Manipulate the URL history stack
  - Does not cause a page reload
  - Framework (router) watches stack and reacts accordingly

HTML5 Style

Needs server-side configuration

With configuration:

```
                    ----- server ------
          http://site.com --> I              I
                    I   /index.html   I
            index.html <-- I              I
                    ------------------
```

Without configuration:

```
                    ----- server ------
http://site.com/products/list --> I              I
                    I   /index.html   I
        404 page not found <-- I              I
                    ------------------
```

ASCII art credit: http://krasimirtsonev.com/blog/article/deep-dive-into-client-side-routing-navigo-pushstate-hash

HTML5 Style

- Server-side configuration: Apache example

```
<ifModule mod_rewrite.c>
  Options +FollowSymLinks
  IndexIgnore */*
  RewriteEngine On
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule (.*) index.html
</ifModule>
```

- Client-side configuration:

```
var app = angular.module("MyApp", [])
  .config(function(…,$locationProvider){
        $locationProvider.html5Mode(true);
  }
```

HTML5 Style

- Webpack development Server Config

```
devServer: {
  historyApiFallback: true
}
```

Router choice considerations

- HTML5 is the better choice
- The router package should support pushing or popping items on / off of the HTML5 history stack programmatically without having to actually click a link (a la the react router, angular, vue, UI Router)

Base URL specification

- Router needs to know what part of the URL to extract and attempt to match.
- Website could be hosted at, for example:
  - http://myserver.com/myapp/...
- The myapp part must not be interpreted.
- Need to specify what the base url is

Base URL specification

- Each router framework will specify how to configure this option.
- Examples:
  - Angular reads it from the page meta info
  - React uses a prop on the BrowserRouter component

Template Extensions

Viewport

- Provides a location in the DOM to house content associated with a given route definition.
- Only ONE viewport per template with this pattern.

Named Viewports

- Provides a NAMED location in the DOM to house content associated with a given route definition.
- Can have multiple of these per template
  - Can handle multiple pieces of content per route definition: one per named viewport

Link Wrapper

- A component or directive (typically)
- Wraps an anchor (<a>) tag.
- Acts to change the route without causing a page load (usual behavior for an anchor)
  - Auto-prevents default action

Navigation Link

- Acts as a link wrapper but adds:
  - Monitor current route
  - Apply CSS class if current route matches one specified for this link.

Navigation Link Examples

- Manually (possible through a dynamic style selector calculated in a function)
- Automagic through component/directive
- Angular: Via directive
- React: NavLink component (vs just Link)
- Vue: Via component

Navigation Link: Issues

- Highlighting doesn't always work as you would expect.
  - Parent: {`/`, `Entries} => list of blog entries
  - Child: {`/:id'} => selected blog
  - Another: {`/about', `About'} => about page

Navigation Link: Issues

- Highlighting doesn't always work as you would expect.
  - Go to Entries or a blog post, Entries lit up
  - Go to 'About'
    - Both About and Entries are lit up
  - In fact, Entries is ALWAYS lit up!

Navigation Link: EXACT

- Many routers provide an 'exact match' pattern to fix this.
- Of course then '/:id' still would not match 'Entries' so that won't solve this particular problem.

Navigation Link: Sub-level == app area

- Easiest way to avoid route hierarchy issues
- One sub-route for each app area
  - `/` => redirects to `/home`
  - `/entries` => blog entries
  - `/entries/:id` => individual blog

Animation Transition

- Typically provided as a component/directive
- Configurable to provide flexible animation for components entering/exiting the DOM based on route changes
- Sometimes separate component or just an option on the viewport.

Route-based content wrapper

- Define routes as components in DOM that map state/url to a component.
- If route matches, component is rendered
- Can have multiple wrappers per route definition.
- React Router is like this.

Route-based content wrapper

- Pros:
  - Very flexible
  - Reactive and easy to implement
- Cons
  - Difficult to track/manage routes
  - Maintaining route definitions/changing schema is hard.

Route-based wrapper

- In component-based routing, the system will often not reload the component when it had already had it loaded.
- Instead it will reuse it if it can (more efficient).
- It is your responsibility to be able to detect when the URL changes (from, say /posts/1 to /posts/2)

Route-based wrapper

- This can be done either through:
  - Subscribing to an observable (angular)
    - Beware of memory leaks!
  - Tapping in to component lifecycle hooks (react, angularJS, vue)

Route Grouping

- For systems that allow more than one route definition to be active at one time (React)
- Configurable to make part of the route definition hierarchy ‘single select’ or ‘radio button’ like.
- Only ONE route in the group will be active.
- Selected based on order in the group

Marketing NavLink

- Situation: Link to Paid Feature pages for user without paid account
  - Question: Render it or not.

Marketing NavLink

- Considerations:
  - Rendering it as disabled exposes link address even if disabled status prevents nav.
  - ALSO ensure route guard in place in case user manually enters URL!
  - Provides 'surface area' for hacker attack

Marketing NavLink

- Considerations:
  - Could be a marketing opportunity!
    - Render link but send user to marketing page instead.
    - If you do render it 'disabled', possibly with hover help 'ad', ensure link is '#' or similar to not leak extra info (URL) to hackers.

Route Controller Patterns

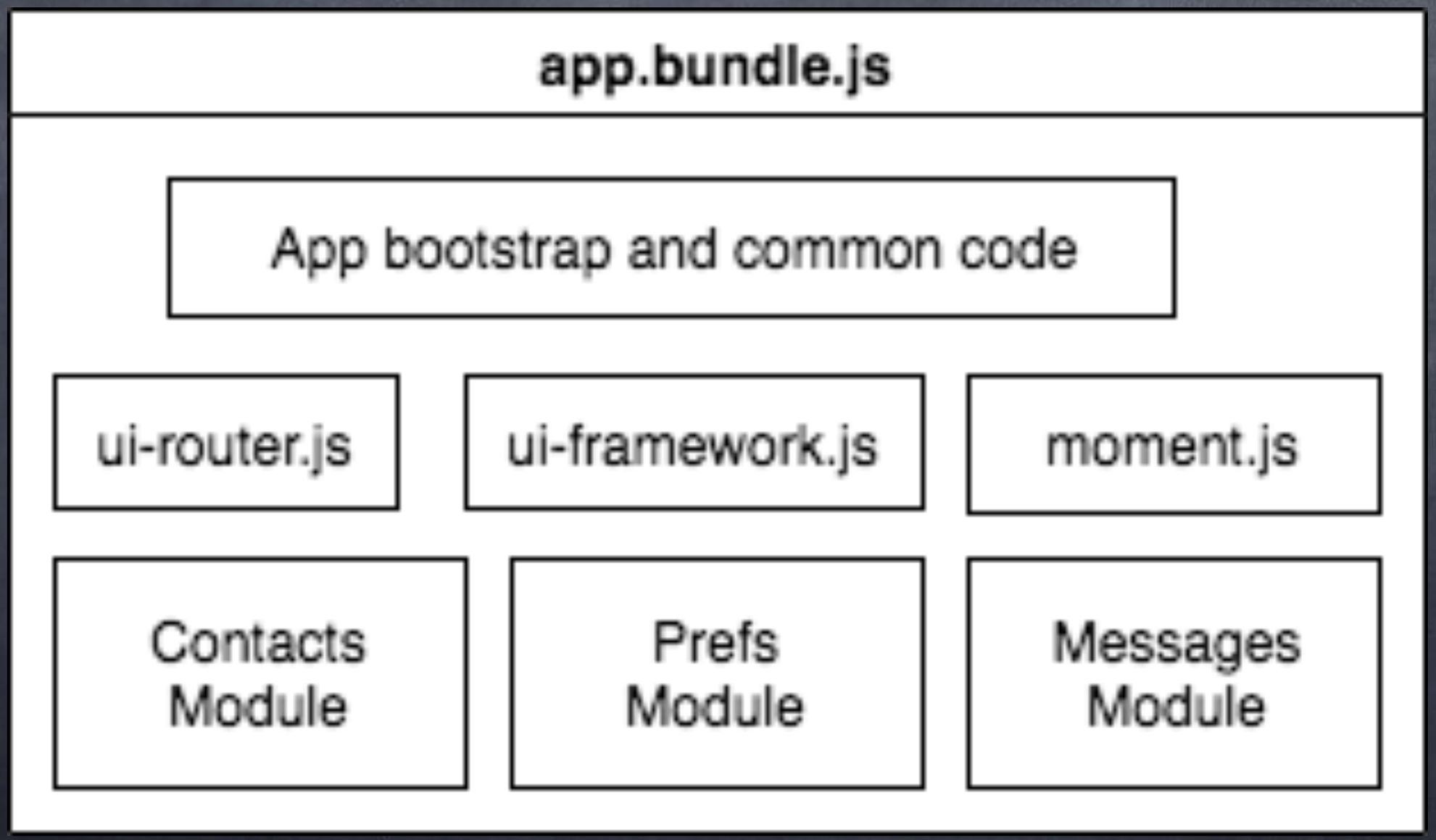
Component-based loading

- Specifies one or more components to load
- Router takes care of loading logic
- Interacts with Template extensions to provide homes for specified components

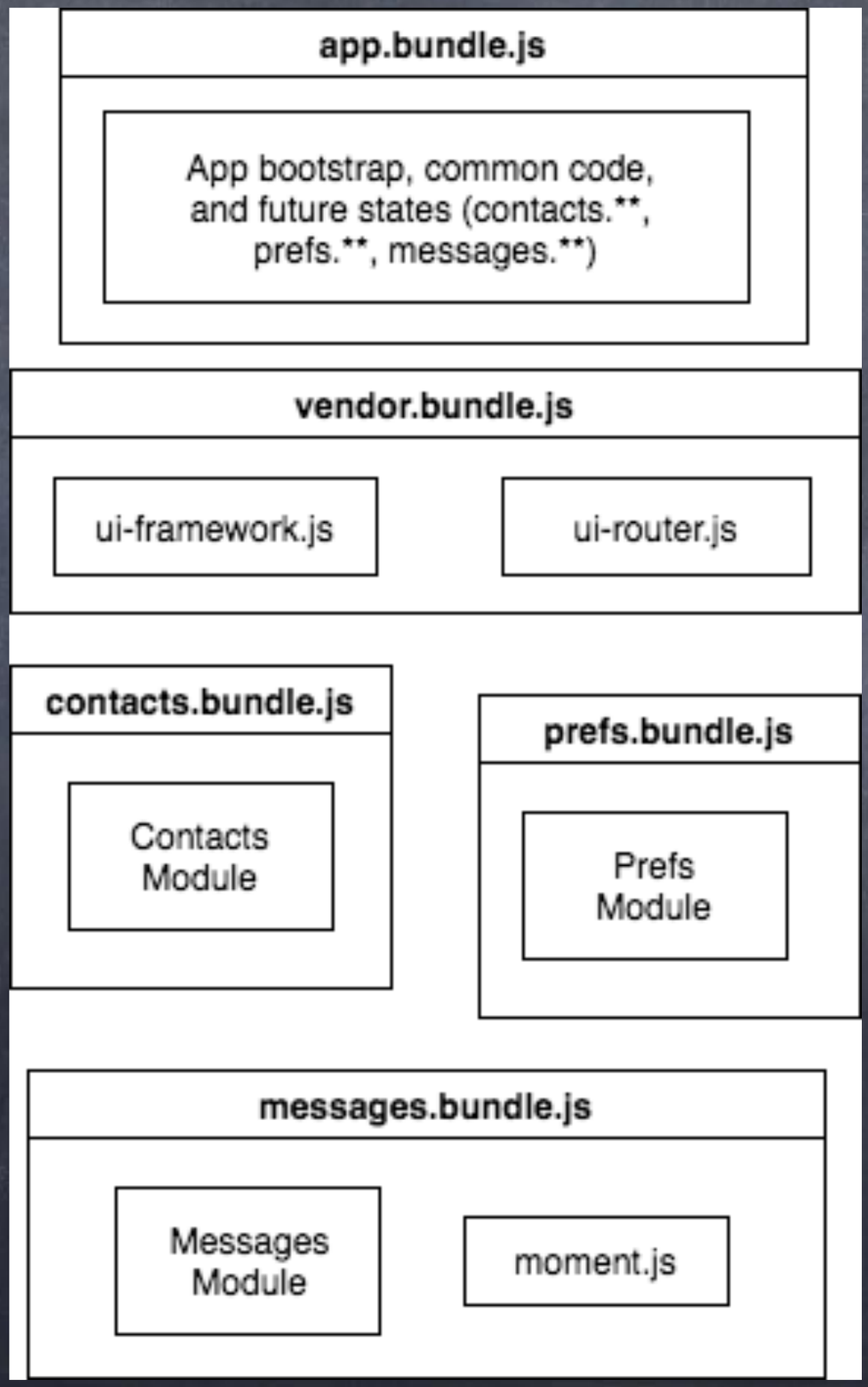
Lazy Loading

- Works with bundler/build pipeline
- Creates bundles based on routing patterns
- Bundles are typically per app functional area
- Bundles load lazily as user navigates

Lazy Loading (pre)



Lazy Loading (post)



Multiple Versions Adapter

- Ability to host content from multiple frameworks or versions in the same application
- Wraps each content in adapters
- Router engine provides plumbing to provide communications/compatibility

Multiple Versions Adapter

- Example
  - UI Router (native + plug ins)
  - Angular / React via component wrappers

Redirect or Push?

- Problem: Async save of detail record then nav to list view.
- Decision to make: on success, redirect or push?

Redirect or Push?

- Redirect removes previous/current route from history.
- User goes to the list page but back button does not take them to back to the edit view.

Redirect or Push?

- Could be desired behavior but typically users find it confusing / annoying.

Redirect or Push?

- Better solution: Push list view onto history stack programmatically.
- Maintains history and gets the user where they need to be.

Redirect or Push?

- Another problem: Route guard detects auth issue with proposed route change.
- Redirect or push?
- Answer: Redirect!
  - We want to remove the current/ proposed route from the stack as the user should not be going there in the first place!

Rendering multiple components for one URL

- React:
  - Route components are all evaluated individually and will (all) render if matched unless you use <Switch> component around them.
- Angular
  - Only one router outlet per URL.
  - Have to load cor render components conditionally through code

Route Definition Patterns

Route-based content wrapper

- Define routes as components in DOM that map state/url to a component.
- If route matches, component is rendered
- Can have multiple wrappers per route definition.
- React Router is like this.

Hirarchial Routes

- Most route definitions are hierarchal.
- Can be arranged in a tree.
- Can be URL or State based

Route 'State' object

- Always one and only one route 'state' active
- May be derived from parent and 'active' children (url based routing)
- May be absolute (state based routing)

Extensible 'State' metadata

- Ability to define non-framework-defined meta data on a per route definition basis
- Set values during definition of routes
- Change values dynamically
- Route objects typically available via injection so can be used for conditional logic and processing.

Auxilliary/Sticky Routes

- A route within a route
- Defines two separate route definitions in one
- Used to maintain 2 route stacks at once

Auxilliary/Sticky Routes

- Each stack is independent
- Can change one or other or both
- Used to drive ‘sticky windows/dialogs/tabs’
- Think of gmail and the ‘compose mail’
- Think of Facebook with messenger windows

Auxilliary/Sticky Routes

- URL Based:
  - Typically encodes secondary/subordinate routes by tacking them on the end of the URL with separators
  - Framework will extract them and do the right thing
  - Usually requires named viewports for each of the secondary route objects

Auxilliary/Sticky Routes

- State Based:
  - Typically encodes secondary/subordinate routes as objects in a map held in a 'children' or 'sticky' or similar property in the main state object.
  - Framework will extract them and do the right thing
  - Usually requires named viewports for each of the secondary route objects

Exact Route Matching

- An option offered on most route definition schemas
- Most routes will match even if they represent only the *start* of a given route
- 'Exact' forces an exact match only

Fallback/Default route

- Ensure it is the last one defined in the route definition object/mechanism
- Router typically match only the first matching route definition.
- Use the default as a security route/logging route.

Per module/component route definition files

- Do not store all your route definitions in one file.
- Componentize or, at least, modularize the definitions
- Keeps things better organized.
- Encourage Reuse
- Improve testing

Relative paths for children

- Often we can specify route definitions either as relative or absolute paths
  - Relative allows insertion of child anywhere in component tree.
- Where not possible (react needs absolute paths), dynamically determine parent path and prepend relative child path with it.
- Ensures maximal flexibility/reuse!

Optional Route Parameters

- Ability to optionally include route parameters but still have route definition match if they are omitted.
- Example: React: /xxx/yyy/:aaa?/:bbb?

Need to process search params only

- If you only need to process search params, you can use the built in JavaScript function:
  - URLSearchParams is reg JS function.
  - Use entries() to get iterator

Navigate to external URL (outside app)

- window.location
- <a href> (not the router link component)

Lifecycle Hook Patterns

Definition: Hook Functions

- Hooks are callbacks that fire on lifecycle events.
- Typically accept info about the 'from' and 'to' router state / url

Hook return values

- Hook return values can alter transition:
  - false: abort transition
  - truthy: continue transition
  - object: redirect/other
  - Promise: pause until resolved, then as above depending on resolved value

Hook registration

- Hooks can be registered:
  - Globally (apply to all state transitions)
  - Per state/route definition

Hook state filter

- Some frameworks allow filtering of events before execution based on to/from criteria
  - Makes your hook code cleaner/tighter
  - Allows meta programming

Hook result filter

- Some frameworks allow filtering of events before execution based on transition result
  - Success/Fail
  - Again, makes your hook code cleaner/tighter

Hook wildcard filter

- Some frameworks allow filtering of events before execution based on wildcard matches against to/from state names
  - Much more flexible than exact matching

Hook functional filter

- Some frameworks allow filtering of events before execution based on result of executing a function passed router-based data.
  - Return value specifies how to proceed
  - Ultimate flexibility
  - Meta programming capabilities

Custom Cross-cutting Concern Hook

- Listens to all route changes
- Applies logic or processing for each change:
  - Logging
  - Security checking
  - Updating Data store

Animation Hook

- Typically provided as a component/directive
- Configurable to provide configurable animation for components entering/exiting the DOM based on route changes

Router Change Observable

- Provides an observable of route change events
- Subscribe in a component's controller to react to route changes

Router Change Observable

- Problem: Potential memory leaks!
- Do not rely on components to clean up observables.
- Explicitly destroy them when the component is destroyed!

Unsaved Changes Hook

- Typically a custom extension
  - Relies on an 'about to change route' hook
  - Relies on ability to detect changes
    - Important to have solid change detection strategy/architecture in place.
    - Usually a 'hasChanges' flag on the CM

Unsaved Changes Hook

- Issue with component based 'unsavedChanges' flag.
  - If your hook asks user if they wish to proceed and they say 'yes'
- Ensure you set 'unsaved changes' flag to FALSE.
- Component may be reused and not reinitialized completely.
- Makes for interesting debugging challenges!

Unsaved Changes Hook

- Also, this hook needs to be global and installed in the ROOT component!
- Ensures that it will be called on EVERY route change.

Hook Examples: React

Component Lifecycle - Creation

constructor()	componentWillMount()
componentWillReceiveProps()	shouldComponentUpdate()
componentWillUpdate()	componentDidUpdate()
componentDidCatch()	componentDidMount()
componentWillUnmount()	render()

State Management Hooks

Central Store Updater

- If you use a central state management store:
  - Redux/Flux/Vuex/NgRx
- This pattern keeps store informed of Route (state or URL) changes.
- Record of router state changes become part of central state history.

Central Store Updater

- Central State tracks Router state
  - Router state changes become part of central state history.
- Time travel works better
- Components can directly query/observe as with other props
- No additional listeners/watchers needed

Central Store Action

- Some routers will dispatch an action automatically (NgRx + Angular, for example)
- Reducers can react to the change in URL / State and update app state accordingly

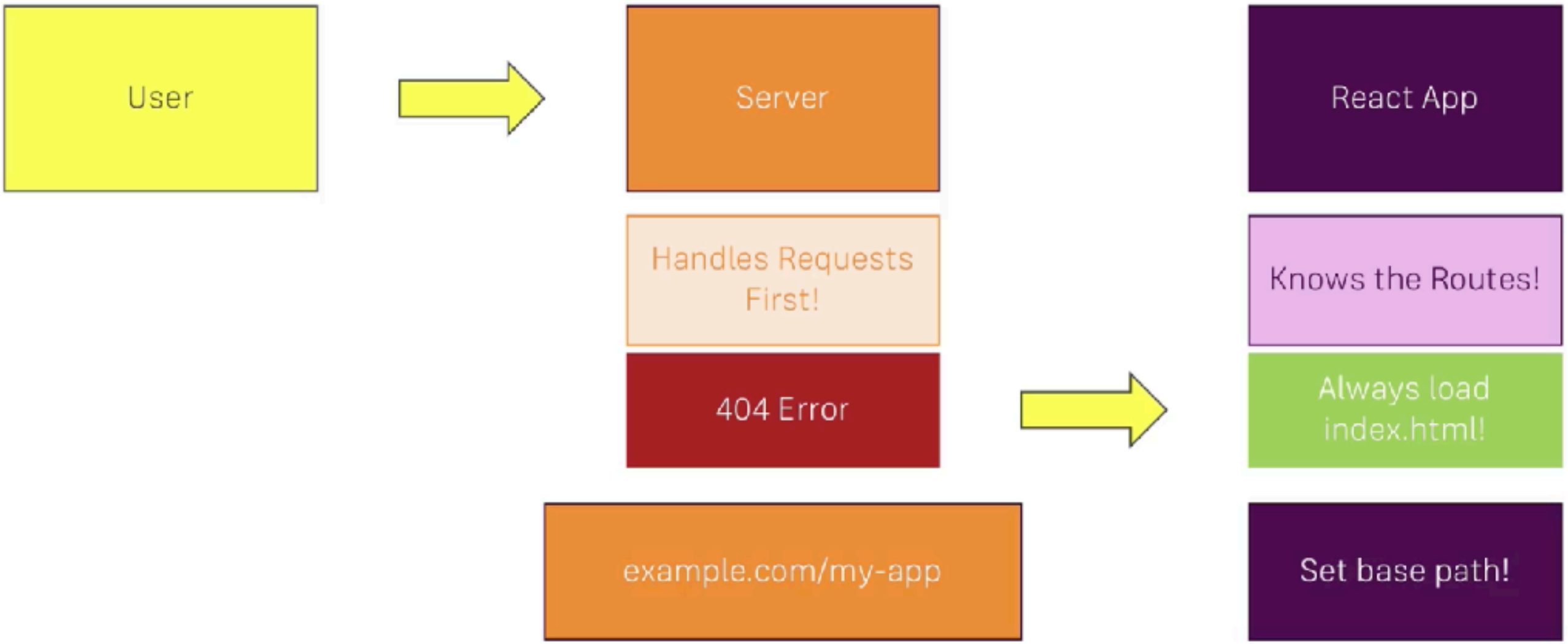
Query params vs DB

- Sometimes all I need for a child view is one or two props of a given object.
- Could pass id as a query param and look up the entity in the DB in the child component
- Better might be to pass in 1 or 2 props via query parameters on the route.
- Even better ... use a centralized data store :)

Typical Router Input

- HTML5 style:
  - <protocol>://<server>:<port>/…
    - … <path>/<p1>?<q1=v1>&<q2=v2>#<hash>
- Hash style:
  - <protocol>://<server>:<port>/#…
    - … <path>/<p1>?<q1=v1>&<q2=v2>

Routing & The Server (Deployment)



Hybrid App Strategies via a router

- https://www.npmjs.com/package/@uirouter/react-hybrid
- https://github.com/ng2-ui/react
- https://stackoverflow.com/questions/45840096/using-react-component-in-angular-2
- http://www.syntaxsuccess.com/viewarticle/integrating-react-with-angular-2.0
- https://github.com/LookLikeAPro/Angular2-React
- https://www.packtpub.com/books/content/integrating-angular-2-react
- http://angularjs.blogspot.com/2016/04/angular-2-react-native.html

Thank You!

vuejs.org

Questions?

References

- Udemy courses:
  - https://www.udemy.com/react-the-complete-guide-incl-redux,
  - https://www.udemy.com/angular-router
  - https://www.udemy.com/vuejs-2-the-complete-guide
  - https://www.udemy.com/angular-2-routing-up-and-running
- https://medium.freecodecamp.org/you-might-not-need-react-router-38673620f3d
- http://marcobotto.com/frontend-javascript-single-page-application-architecture/
- https://auth0.com/blog/react-router-alternatives/