PHP Backend & the Javascript Frontend

by Ralph Schindler for #SunshinePHP

Offers.com™
Your Place to Save Every Day

About Me

about.me/ralphschindler

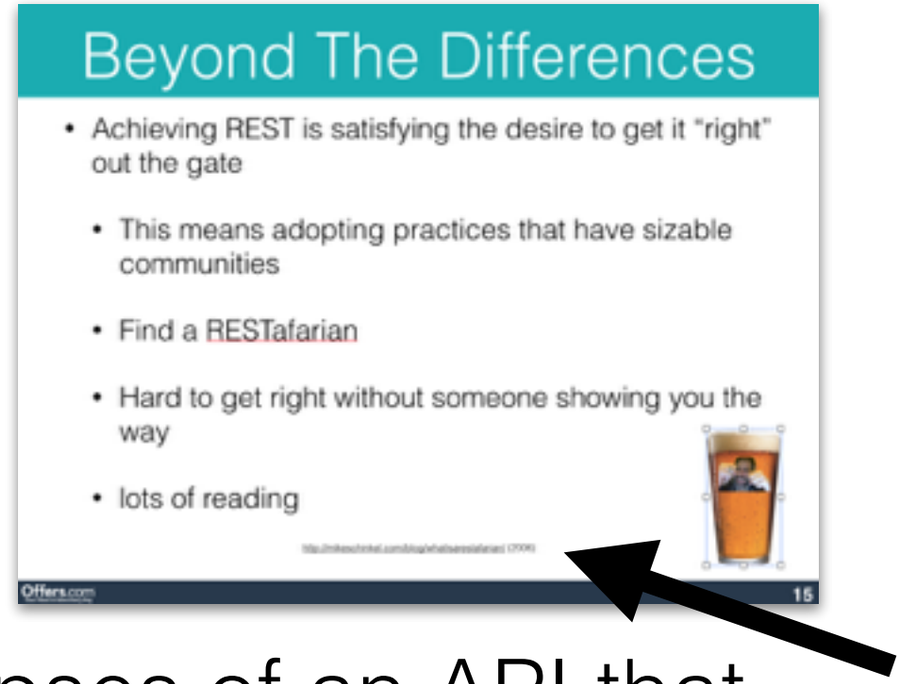
- Ralph Schindler
- PHP Programmer for 17-18 years
- Employment
  - Offers.com since May 2014
  - Zend Framework @ Zend (08-14)
  - 3Com's Tippingpoint (05-08)

Is This Talk For Me?

- Are you coming from full stack framework?
  - Used to lots of abstraction and are further from specification (example: form objects)
  - Frameworks support your “get things done” attitude
- Want to explore micro-service architecture, HTTP, Single Page Apps, REST
- We’re going to build an API and a SPA

Talk Format

- Slides Located at:
  - https://github.com/ralphschindler/php-backend-js-frontend
- Beer me icon
  - opinions that can be hashed out over a beer
- Some meme references
- Content, lots of information, glimpses of an API that been built for this talk, glimpses of a SPA client



New Terminology

- **“Developer Plasticity”**
  - Ability to work in the same space, with the same technologies in new ways
  - Ability to continue molding and shaping the way *you* write applications to fit an increasingly iterative landscape of changing methodologies
  - Some people call this “continuing education”

“You must build your **API First**.”

–Zeus (from the Cloud)

What Does That Mean?

- API First in practice means starting with exposing as much of your core business as a HTTP API/REST-ish application… **first**

It's What The Cloud Wants

- "Modern day" / trending architectures
  - highly decoupled *micro-services*
  - hardware abstraction / 

  - ability to iterate services quickly
  - 12 Factor Apps (talk to @dzulke)

http://martinfowler.com/articles/microservices.html

- Once you have your API built out **first**, you'll build your first API consuming client: the web app.

- JS “App” is #1 App, but anticipate others:
  - different devices
  - mobile sites & site widgets
  - other APIs
  - developers simply interacting with it
    - httpie, Paw, php one-off-scripts

Building a PHP API Backend

“Legacy” PHP Application (Full Stack)

- Controller: Front, Routing, Dispatching
- Model: Database, Document Engine, Web Services
- View: Forms, HTML, Helpers
- App Stack Services:
  - Contexts, events, service management, sessions
- Other & cross-cutting concerns
  - Sending emails, logging, statistics, authentication

PHP Backend (REST-ish Stack)

- Controller: Front, Routing, Dispatching, **HTTP**
- Model: Database, Document Engine, Web Services
- View: ~~Forms, HTML, Helpers~~
- App Stack Services:
  - Contexts, events, service management, ~~sessions~~
- Other & cross-cutting concerns
  - Sending emails, logging, statistics, authentication

PHP Backend (REST-ish Stack)

- Goals:
  - extremely fast response
  - “session-less” (stateless, the “ST” in REST)
  - much lighter weight handling of “views”: HTTP response and content representations

http://stackoverflow.com/questions/3105296/if-rest-applications-are-supposed-to-be-stateless-how-do-you-manage-sessions

Beyond The Differences

- Achieving REST is satisfying the desire to get it “right” out the gate
  - This means adopting practices that have sizable communities
  - Find a RESTafarian
  - Hard to get right without someone showing you the way
  - lots of reading

http://mikeschinkel.com/blog/whatisarestafarian/ (2006)

The REST We Want

- We want Roy Fielding's concept of REST
  - client/server
  - stateless
  - cacheable
  - uniform / layered
  - Code On Demand (optional)

http://en.wikipedia.org/wiki/Roy_Fielding
http://en.wikipedia.org/wiki/Representational_state_transfer
http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

The REST We Want

- Achieve full “Glory of Rest”
  - Done through Level 3 of Richardson Maturity Model (RMM)

http://martinfowler.com/articles/richardsonMaturityModel.html

The REST We Want

- HATEOAS
  - Hypertext as the engine of application state

http://en.wikipedia.org/wiki/HATEOAS

Findings

- While we get “closer to the metal” (HTTP Specification)
- Fewer application concerns means simpler architectures (opinion)
  - can move away from MVC as an architectural *prescription (diction)*

You Gotta Know HTTP

- Re-familiarize yourself with RFC 2616
- Request / Response
- Formats
- Base / common headers

http://www.w3.org/Protocols/rfc2616/rfc2616.html

HTTP Request (httpie)

```
$ http --verbose -j localhost:9000/
GET / HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Content-Type: application/json; charset=utf-8
Host: localhost:9000
User-Agent: HTTPie/0.8.0
```

https://github.com/jakubroztocil/httpie

HTTP Request (firefox)

Request URL: `http://localhost:9000/`
Request method: `GET`
Status code: 406 Not Acceptable

Edit and Resend | Raw headers

Filter headers

Response headers (0.120 KB)

Request headers (0.621 KB)

Host: "localhost:9000"
User-Agent: "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:35.0) Gecko/20100101 Firefox/35.0"
Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
Accept-Language: "en-US,en;q=0.5"
Accept-Encoding: "gzip, deflate"
Cookie: "wp-settings-time-1=1409114636; __utma=111872281.507114394…73874295995058c84a6448329a527d1b76czo1OiJhZG1pbiI7%22%3B"
Connection: "keep-alive"

HTTP Request

- Two different clients, talking to the same web service, different looking requests…

HTTP Response (httpie)

```
HTTP/1.1 200 OK
Access-Control-Allow-Headers: Content-Type, X-Requested-With
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Origin: *
Cache-Control: no-cache
Connection: close
Content-Type: application/hal+json
Date: Sat, 07 Feb 2015 15:24:24 GMT
Host: localhost:9000
X-Powered-By: PHP/5.5.17

{
    "_links": {
        "ra:locations": {
            "href": "/location",
            "title": "Locations"
        },
        "ra:reminders": {
            "href": "/reminder",
            "title": "Reminders"
        },
        "self": {
            "href": "/"
        }
    }
}
```

HTTP Response (firefox)

Request URL: http://localhost:9000/
Request method: GET
Status code: 406 Not Acceptable

Edit and Resend | Raw headers

Filter headers

Response headers (0.120 KB)

Connection: "close"
Content-Type: "text/html"
Host: "localhost:9000"
X-Powered-By: "PHP/5.5.17"

Informing the Architecture

- HTTP and common REST approaches must drive the design of architecture

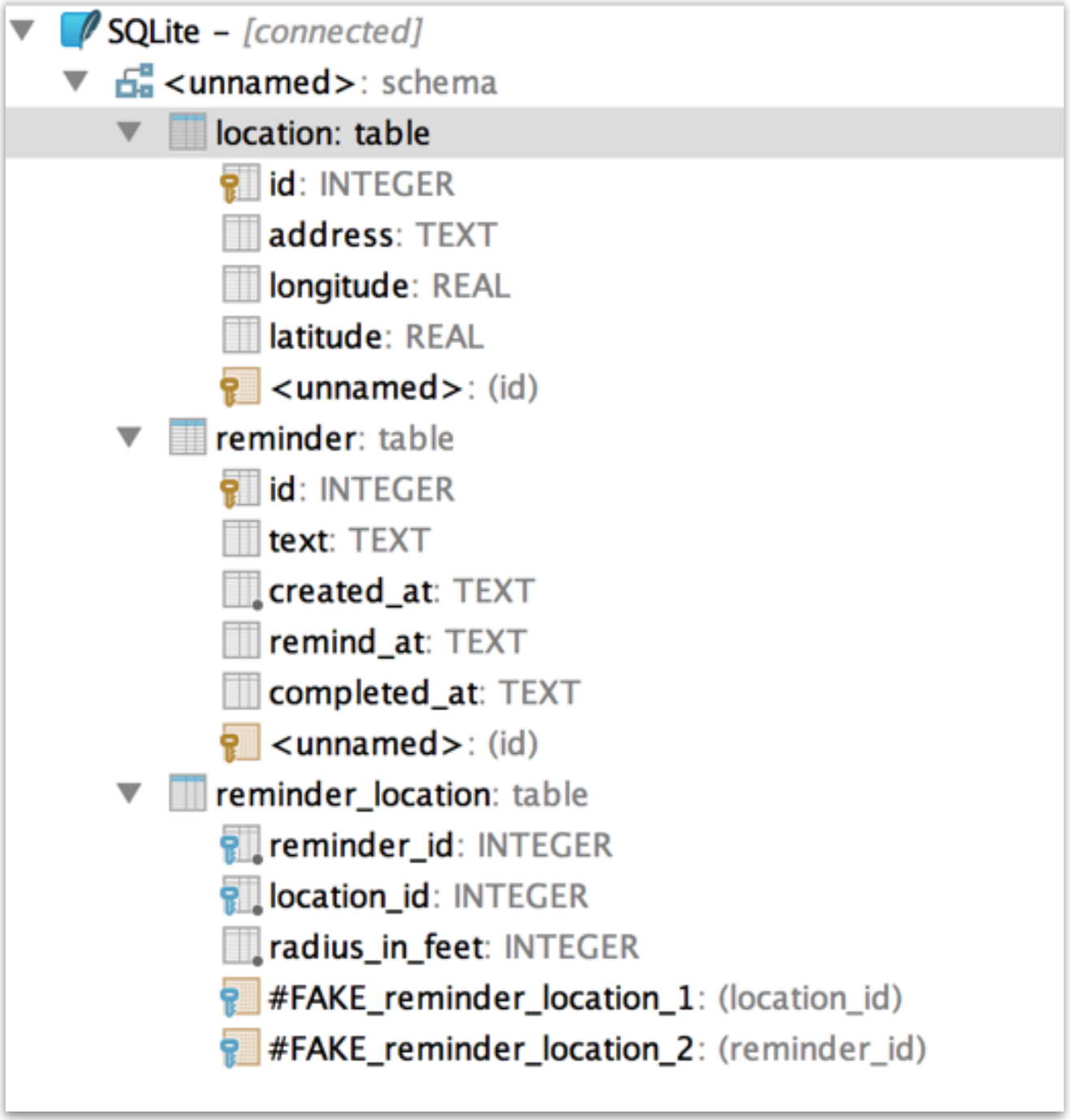
Let’s Build The PHP Backend

Example Application

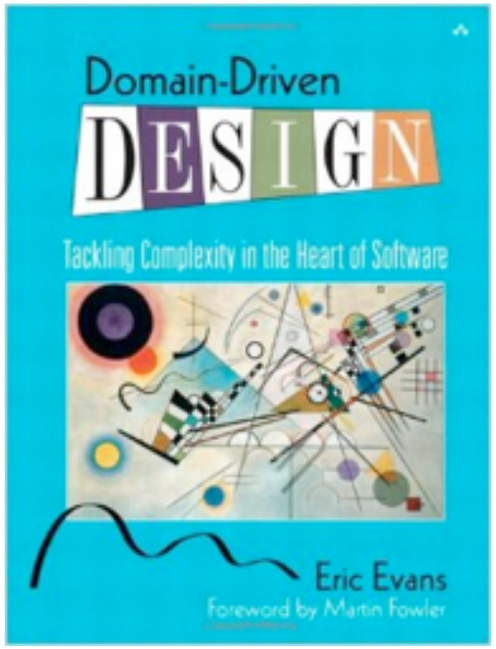
- Reminder App
  - Remind me to do something
    - At a particular time
    - Or, when I'm at a particular location

Example Application

- Database Schema:



Example Application



- What a model looks like:
  - Repository
  - Entities

http://www.infoq.com/minibooks/domain-driven-design-quickly (PDF can be found on the web)

RMM Level 0

- Choose a web (HTTP) framework
  - “Micro” framework (API centric frameworks)
  - Silex, Slim, …
  - Larger full-stack frameworks are moderately suited for API development
    - (Apigility is both full stack and RESTful)

http://www.slimframework.com/
http://silex.sensiolabs.org/
https://apigility.org/

Example Application

- Picking a non-standard application architecture (personally)
- Adding to the straight-forwardness of a “micro” architecture, but using REST terminology
  - Endpoints (synonymous to controllers)
  - Model (the domain and the datasource)
  - Resources (synonymous to views, but inclusive of HTTP)

RMM Level 1

- Start organizing and planning your **resources**
- Do this using URLs
  - `/ - A root`
  - `/reminder/{id} - A reminder collection and entities`
    - `also searchable collection (?current=true&near=x,y)`
  - `/location/{id} - A location collection and entities`

RMM Level 2

- Start using verbs:

  - GET, POST, PUT, DELETE, *PATCH*, OPTIONS, HEAD
  - Safe: GET, HEAD, OPTIONS
  - Idempotent: OPTIONS, HEAD, GET, PUT, DELETE

https://tools.ietf.org/html/rfc5789
https://tools.ietf.org/html/draft-ietf-appsawg-json-patch-10
http://williamdurand.fr/2014/02/14/please-do-not-patch-like-an-idiot/ (I disagree with this)

RMM Level 3

- Start using hypermedia & links
  - HAL and others
    - Collection+json
    - Siren

On HAL

- Don’t be afraid of HAL
- It’s plain json, but with _links & _embedded, nothing more
- It only applies to the server representation
  - You’ll never be posting/putting/patching into _links or _embedded

- How can we make our “Resources” / “Views” speak HAL?
- PHP HAL library
  - https://github.com/blongden/hal

https://github.com/blongden/hal

Let’s explore the code

How Do We Know …

- … if we got the HAL portion correct?
  - HAL Browser (Mike Kelly)
  - Let’s use relations to document!

https://github.com/mikekelly/hal-browser

Building the JS Frontend

Javascript Frontend

- Goals
  - Decoupled from datasource / REST backend
    - (Separate product life-cycles)
  - Maintainable & quickly iterable

SPA (Single Page Application)

- Single “page load” (HTML)
  - Not including assets & support files
  - There is no “page reload”
- Goal is to be more *desktop*-like in experience

Sidenote: Learn JS Ecosystem

- There's lots of small packages
  - Unix-philosophy: "do one thing well"
  - Why NPM is so prolific
- Lots of choices in packages
  - This means even with a moderately opinionated framework, you have to weigh pros/cons of all decisions
  - Github stars are important, lingering PR's are important
- **Adopt the best practices, be agile in your learning!**

Sidenote: Learn JS Ecosystem

- Use NPM
- Use Bower
- Use Gulp
- Why? …

```html
<!DOCTYPE html>
<html data-ng-app="app" ng-strict-di xmlns="http://www.w3.org/1999/html">
<head lang="en">
  <meta charset="UTF-8">
  <title>Reminders</title>
</head>
<body>
  <a href="/#/reminders">Reminders</a> | <a href="/#/locations">Locations</a><br>
  <div ng-view></div>
  <script type="text/javascript" src="bower_components/lodash/dist/lodash.js"></script>
  <script type="text/javascript" src="bower_components/angular/angular.js"></script>
  <script type="text/javascript" src="bower_components/angular-route/angular-route.js"></script>
  <script type="text/javascript" src="app/location/location.module.js"></script>
  <script type="text/javascript" src="app/location/location.js"></script>
  <script type="text/javascript" src="app/reminder/reminder.module.js"></script>
  <script type="text/javascript" src="app/reminder/reminder.js"></script>
  <script type="text/javascript" src="app/app.module.js"></script>
  <script type="text/javascript" src="app/api.service.js"></script>
  <script type="text/javascript" src="app/route-config.js"></script>
</body>
</html>
```

Example Application

- Very simple feature-set:
  - Front-end for the RemindersApp backend
  - Ability to view reminders, edit the name of a reminder

JS Frontend

- Choose a framework for our SPA
  - Our choice: Angular
- Know all your choices
  - Ember, React, Backbone, etc

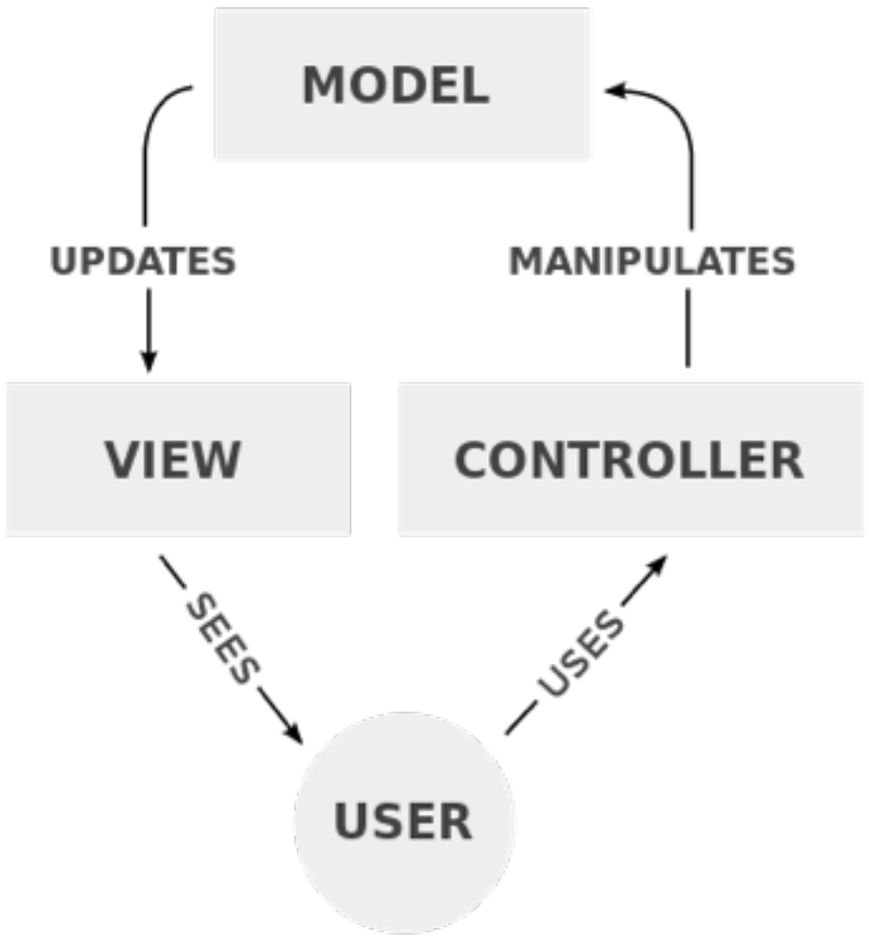
Angular

- Complete shift in how applications are built for the HTML/DOM/Browser
- Read “How do I “think in AngularJS” if I have a jQuery background?”

http://stackoverflow.com/questions/14994391/how-do-i-think-in-angularjs-if-i-have-a-jquery-background

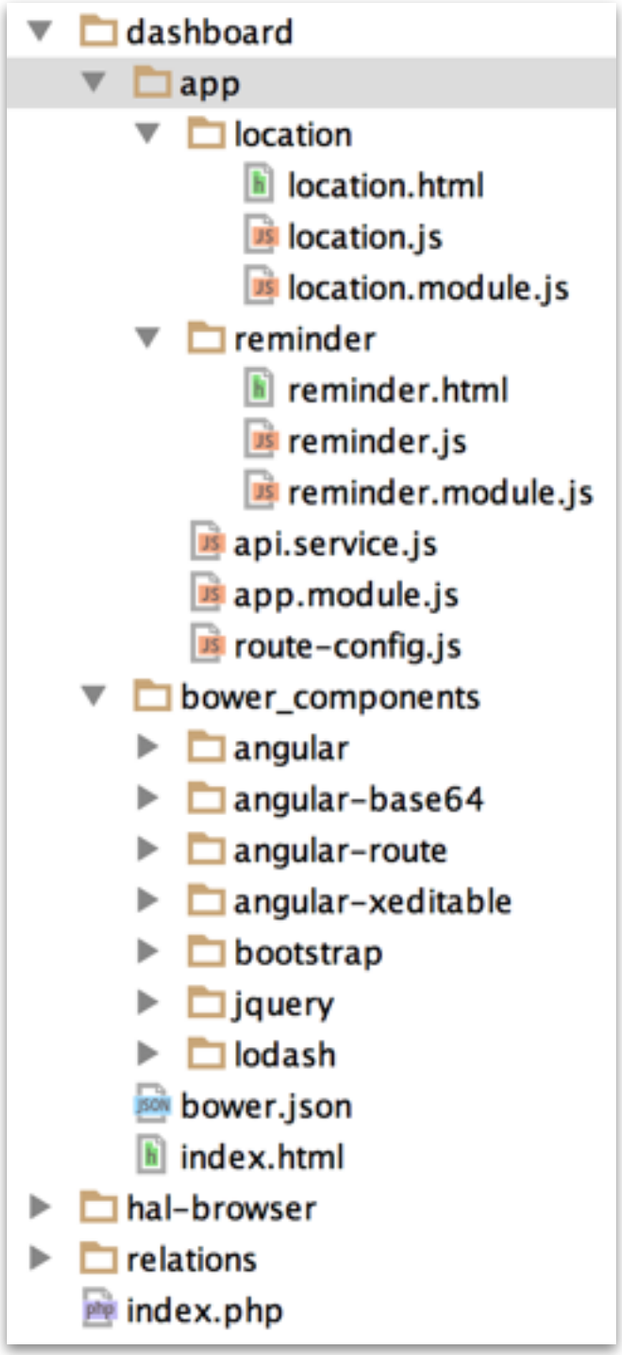
Angular

- Full-stack client side (browser) MVC framework
- Moderately coupled and opinionated:
  - **2-way** Data-binding
  - Built-in “templating”
  - Dependency Injection
  - Modular



Angular Best Practices

- John Papa's Style Guide
  - 1 file per concern
  - highly component driven



https://github.com/johnpapa/angularjs-styleguide

Modeling

- Build model/service
  - respect REST, utilize links in HAL
  - (Would like to use HAL parser, if one worked well)

Before we explore the code…

- How is it deployed?
  - separate web service (www.domain.com & separate api.domain.com)
  - from the same root url?
    - (we're going to deploy it from the doc root, under /dashboard)

Let’s explore the code

JS Frontend

- Parts we didn’t discuss (that could be their own talk)?
  - Authentication (subject for a whole other talk)
  - ACLs / Conditional Views

Things We Can't Fit In

- Authentication
  - OAuth2
- Versioning (it can be done! see Apigility)

- Thank you
- Questions?
- @ralphschindler / http://ralphschindler.com