ANGULAR.JS IN THE ENTERPRISE

Christian Ulbrich, Zalari UG

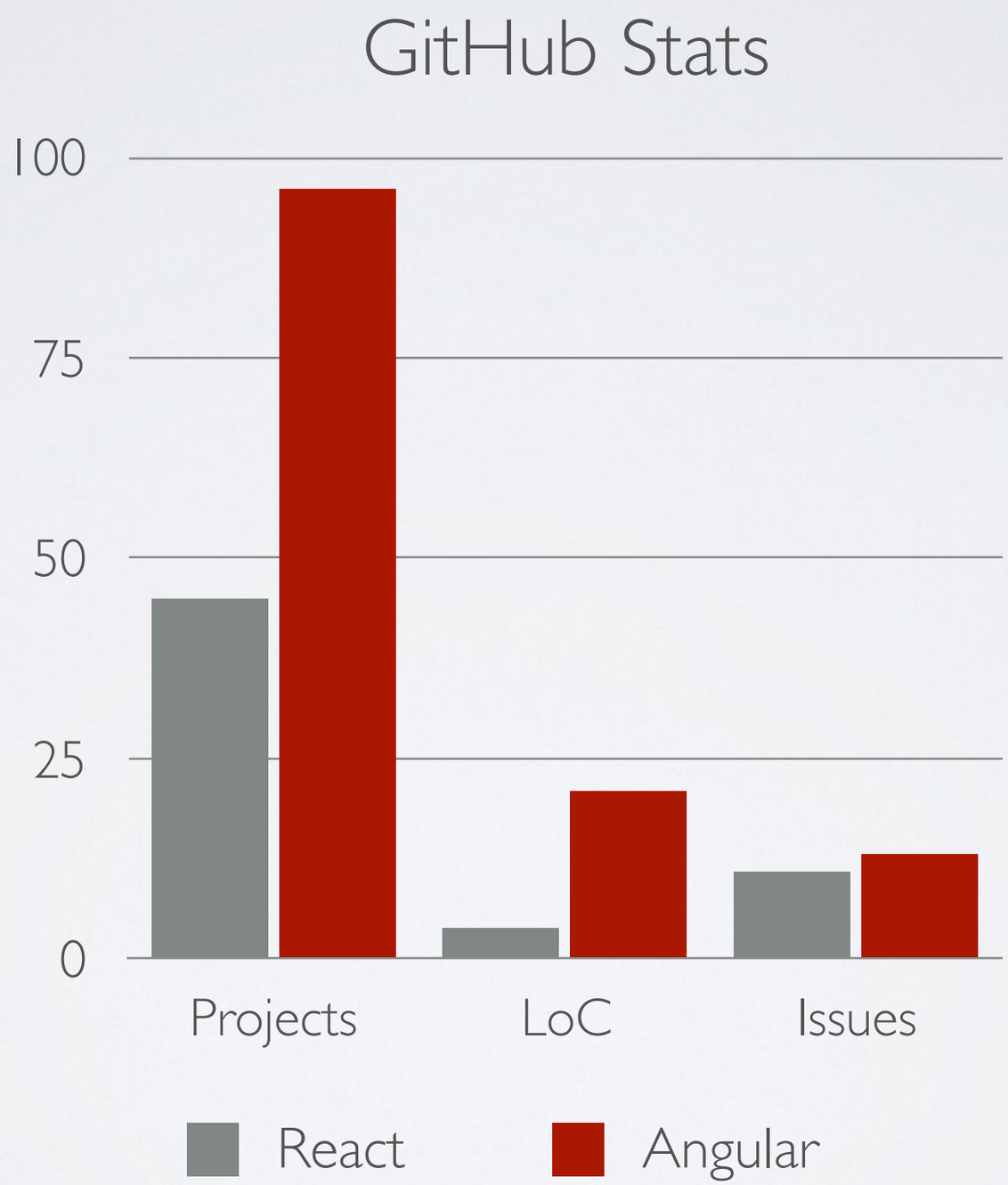
AGENDA PROPAGANDA

- Motivation
- File layout
- Reliable build
- Advanced routing
- Model layer
- Framework hooks
- Links

MOTIVATION

- What has Angular - that the other uber frameworks do not have?
  - Opinion
  - Completeness
  - Scope
  - Maturity

MOTIVATION


GitHub Stats
100
75
50
25
0
Projects
LoC
Issues
React
Angular

CREDIBILITY

- 15k LoC JS, 5k CSS
- 100+ services
- 65 directives
- 50+ controllers
- German-wide deployment on-site
- roughly 30 man months

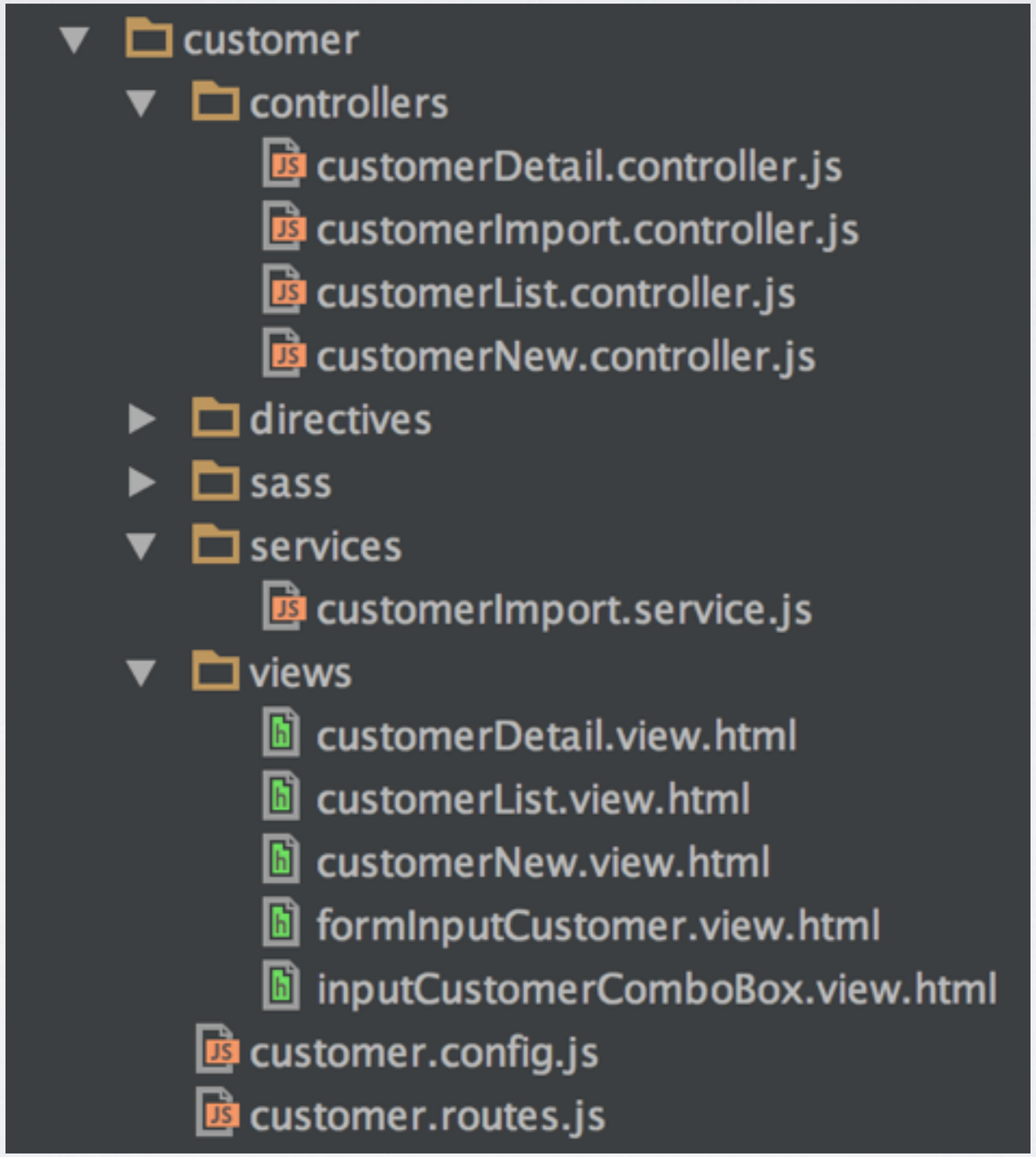
FILE LAYOUT - GOALS

- Find files fast, for everyone!
- Navigating the source must be easy!
- assets in Angular form a relation
  - directives have associated views
  - controllers have associated tests
  - routes have controllers and views
- re-factoring is very expensive

FILE LAYOUT - ADVISES

- angular-seed / yo angular is a NO-GO, IT WON'T SCALE
- add type of asset to file name
  - home.controller.js, home.view.html
- group files by feature / page / state
- pre-create empty directories and version them (using .gitkeep)

FILE LAYOUT - ADVISES



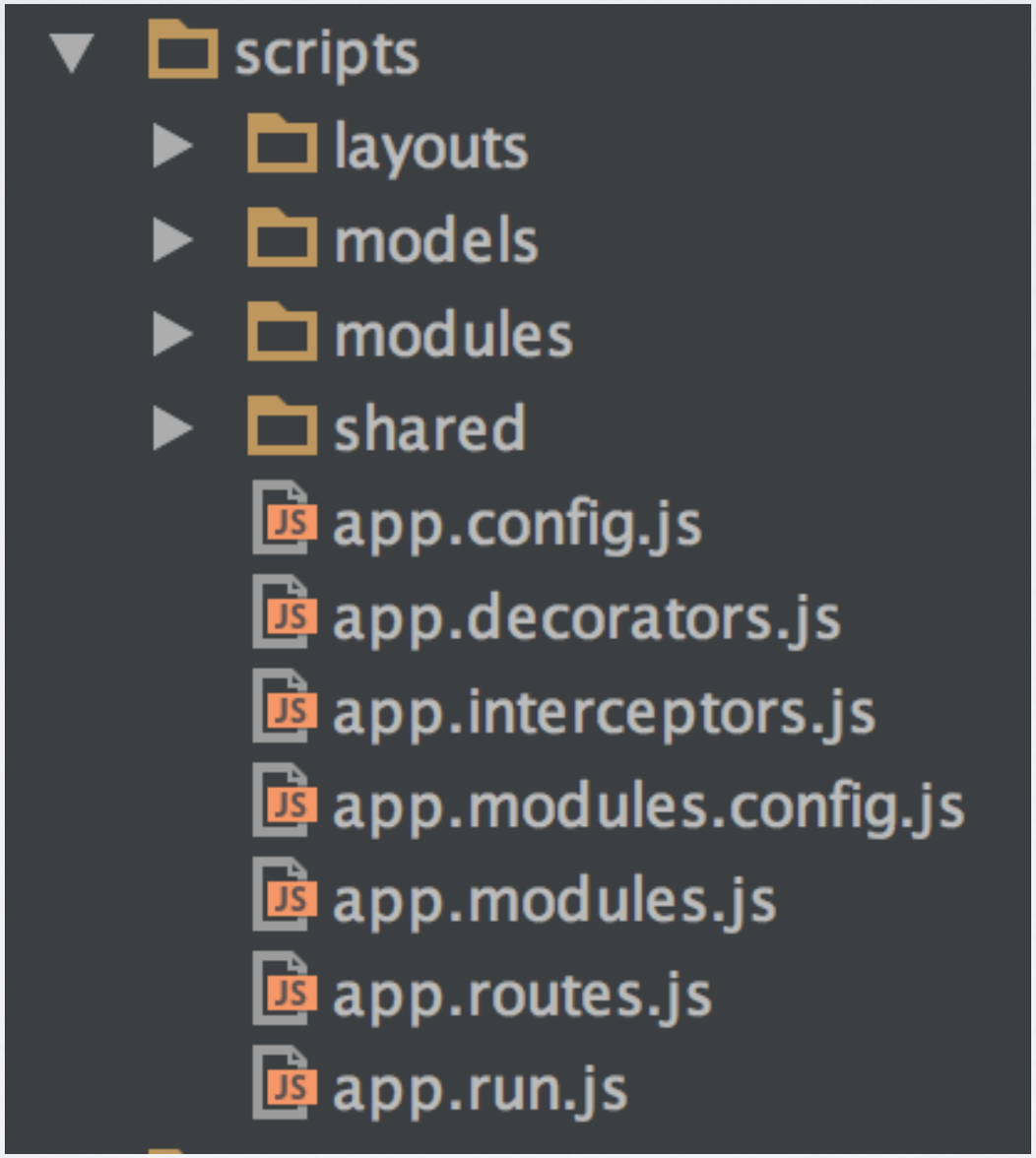
FILE LAYOUT - ADVISES

- pages are reachable by certain routes
- application routes can be split across files
- use individual routes for exactly THAT page / feature
- add individual config as well

FILE LAYOUT - ADVISES

- global configuration of angular app should be in one place -> preferably in root
- split global configuration of *things* into different files
  - app.*config*.js , app.*run*.js , app.*decorators*.js , …

FILE LAYOUT - ADVISES



FILE LAYOUT - ADVISES

app.interceptors.js

```
1  angular.module('[redacted]')
2    .config(['$httpProvider', 'appConfig', function ($httpProvider) {
3
4      $httpProvider.interceptors.push('httpErrorInterceptor');
5      $httpProvider.interceptors.push('loadingNotificationInterceptor');
6
7    }]);
8
```

FILE LAYOUT - ADVISES

app.modules.js

```
1   'use strict';
2
3   angular
4     .module('[redacted]', [
5       'ngSanitize',
6       'ngMessages',
7       'ngTouch',
8       'ui.router',
9       'ui.bootstrap',
10      'ui.select',
11      'zalari.pickadate.datepicker',
12      'zalari.pickadate.timepicker',
13      'angular-data.DSCacheFactory',
14      'cfp.hotkeys'
15    ]);
16
```

FILE LAYOUT - ADVISES

- one piece at a time -> each service should reside in exactly on file
  - easier debugging
  - composable pieces of software
  - small files have better testability and are easier to review
- consistent file naming (camelCase)

FILE LAYOUT - GOTCHAS

- manually and individually requiring files is error-prone

  -> use WebPack, for programmatically requiring

  -> use automation for creating boiler plate (e.g. yo)

- sometimes obscure behavior for missed files

RELIABLE BUILD - GOALS

- The build must be automated as far as possible
- Developers should interact with tasks, targets, nothing more

RELIABLE BUILD - GOALS

- The build should never break!
- The build should never break by itself!
- The build should never break by some external dependency!
- When the build breaks, somebody is responsible and it is feasible why the build broke!

RELIABLE BUILD - ADVISES

- The build should only depend on the software itself
  - version 3rd-party libraries
  - pin the version of 3rd-party libraries
- reduce external dependencies
  - think twice about using certain libraries and tools

RELIABLE BUILD - ADVISES

```
"angular-bootstrap": "0.13.0",
"angular-cache": "3.2.5",
"angular-i18n": "1.3.15",
"angular-messages": "1.3.15",
"angular-touch": "1.3.15",
"angular-pickadate-zalari": "https://github.com/zalari/angular-pickadate.js.git#develop",
"angular-sanitize": "1.3.15",
"angular-ui-router": "0.2.15",
"angular-ui-select": "0.9.9",
```

RELIABLE BUILD - ADVISES

- use as few build tools as possible
- BUT use build tools, that you are comfortable with
  - no-one likes Maven
- use as few package manager as possible
  - maven is a crappy package manager as well
- consume your own software via package managers and package it
- think about ditching bower and just use npm

RELIABLE BUILD - KISS

- there are some bridges between build tools
  - Maven/Gradle can process package.json
- Grunt vs. Gulp vs. *Burp* does not matter
- optimize the right things; production build can be long, a dev build should be as fast as possible
- evolve your own build scripts to UNDERSTAND them

RELIABLE BUILD - GOTCHAS

- npm packages are not really cross-platform
- npm 2 essentially does not work under Windows
- not all environments have direct Internet access
- npm public registry is archaic, unreliable and insecure

ADVANCED ROUTING - GOALS

- Single Page Apps are complex client-side applications with lots of different states
- Complex applications must be modeled
- states must be abstracted

ADVANCED ROUTING - GOALS

- multiple views must be displayed at once
- multiple views must be changed at once
- ngInclude is $rootScope!

ADVANCED ROUTING - ADVISES

- do not bother with ngRoute, it is a toy router
- use angular-ui-router (and pin it!)
- angular-new-router is not production-ready

ADVANCED ROUTING - ANGULAR-UI-ROUTER

- models your app as state machine
- a state machine consists of states, transitions and guards
- allows nesting of states
  - nested states have parent states and thus parent controllers
  - -> very easy implementation of tabs

ADVANCED ROUTING - ANGULAR-UI-ROUTER

- state machine is a powerful software concept
- it is a better model, than just complex URLs
- $states can have the usual properties (controller, templateUrl, template, …)
- but also arbitrary data, that is prototypical inherited
- as $state is in the end just a service and having the reference to the current $state, they allow for nice features:
  - permission-based $state access; preventing $state to enter
  - small gui-specific configuration in child states
- child $states can be re-used across your app (e.g. dialogs)

ADVANCED ROUTING - ANGULAR-UI-ROUTER

```
.constant('customerStates', {
  'detail': {
    url: '/detail/:customerId/:importId',
    templateUrl: 'scripts/modules/customer/views/customerDetail.view.html',
    controller: 'CustomerDetailCtrl',
    resolve: {
      Customer: function (CustomerDetailedService, $stateParams) {
        return CustomerDetailedService.getDetailedById($stateParams.customerId);
      }
    },
    data: {
      permissions: {
        only: ['READ_CUSTOMER_DETAILS'],
        redirectTo: 'main.access.forbidden'
      }
    }

  },
```

ADVANCED ROUTING - ANGULAR-UI-ROUTER

```
.state('main.customers.detail', _.merge({}, customerStates.detail, {
  data: {
    parentState: 'main.customers',
    parentReload: true
  }
}))
```

ADVANCED ROUTING - ANGULAR-UI-ROUTER

- powerful, yet simple API:
  - events for states: $stateChangeStart, $stateChangeSuccess, $stateChangeError, $stateNotFound
  - events for views: $viewContentLoading, $viewContentLoaded
  - state callbacks: onEnter, onExit
- named views
  - have multiple views displayed at once and swap them at once

ADVANCED ROUTING - GOTCHAS

- you NEED a stateChangeError Handler

```
$rootScope.$on('$stateChangeError', function (event, toState, toParams, fromState, fromParams, error) {
  $log.error('from state %s to state %s failed with error %s:', fromState.name, toState.name, error);
  throw new appExceptions.StateChangeErrorException(error);

});
```

- not entirely minification-safe out of the box
- hard to debug
- do not use ambiguous URLs like
  - /customer/:id/:action should match /customer/56/delete
  - /customer/import/:id also matches above… -> use something like /customerImport/:id

MODEL LAYER

- SPAs are about putting as much logic on the client as possible, to reduce server round-trips
- business logic is inevitably pushed on the client
- business logic and business models should be first class citizens on the client as well
- you will need some sort of model layer

MODEL LAYER

- SPAs usually do more than just displaying the raw data
- JSON from REST-Service must be augmented
- -> create business models on the client as well, that have rich business methods

MODEL LAYER - USE CASES

- client side authorization
- converting raw data (e.g. timestamps)
- client side caching
- abstract data structure by using getters / setters

MODEL LAYER

- either use existing libs
- or roll your own
- map business entities to at least one dedicated service
- e.g. CustomerEntity is produced, worked and consumed by a CustomerService
- this is software engineering, give it some thoughts!

MODEL LAYER

- inheritance is not the solution of all problems
- use composition over inheritance
- use a library for inheritance or know what you are doing
- consider using functional design patterns instead of common inheritance design patterns
- remember: every Service is a singleton in AngularJS

FRAMEWORK HOOKS

- Know thy framework!
- AngularJS is *complete*
  - interceptors for all $http requests
  - transformation chain for all $http requests
  - decorators for decorating *any* service
  - filters
  - filter predicate functions
  - ngModelController and its friend formController
    - with their children validators and parsers

FRAMEWORK HOOKS DECORATING $LOG

- never use console.log in your application
- use $log, to enable decoration
- have specific decoration depending on environment

FRAMEWORK HOOKS DECORATING $LOG

app.decorators.js

```
angular.module('[redacted]')
  .config(function($provide) {

    //decorate $log.debug for production
    $provide.decorator('$log', function($delegate, appConfig) {

      //for runtime configuration of debugging, we need to make $log.debug
      //aware of the app config

      var originalDebugFn = $delegate.debug;

      $delegate.debug = function(args) {
        //if debugging is enabled call original implementation with provided args
        if (angular.isDefined(appConfig.debug) && appConfig.debug.enabled) {
          originalDebugFn.apply(null, arguments);
        }
        //otherwise do nothing...
      };


      return $delegate;

    });

  });

```

FRAMEWORK HOOKS TRANSFORMATIONS

- every $http call goes through two transformation chains:
  - transformRequest and transformResponse
- useful for global $http transformation logic:
  - deserialization / serialization of non-JSON formats (e.g. XML)
  - smart processing (e.g. fix broken REST services)

FRAMEWORK HOOKS INTERCEPTORS

- every $http call goes through the interceptor chain
- useful for global $http logic:
  - authentication (customer headers, …)
  - error handling
  - loading notification
  - performance tracking

FRAMEWORK HOOKS LOADING INTERCEPTOR

- global loading notification for every $http call
- -> give feedback to the user
- single point of implementation
- actually it should be configurable, because some $http calls might not be communicated to the user
- beware: order of interceptors does matter!

TESTING

- sorry no time for that any more

LINKS

- File layout:
  - Scalable code organization in AngularJS: http://bit.ly/1nJ8Ktk
- Reliable build:
  - The reliable build: http://bit.ly/1PlnCW1
  - Why We Should Stop Using Bower: http://bit.ly/1JUwaa8
- Advanced Routing
  - Angular-UI Router: http://bit.ly/1idkZH7
  - Angular-UI Router FAQ: http://bit.ly/1cZMjve
  - Advanced routing and resolves: http://bit.ly/1OrQr5H

LINKS

- Model layer:
  - AngularJS Data Modeling (paywalled): http://bit.ly/209vbam
  - Angular model objects with JavaScript classes: http://bit.ly/1U6JHfi
- Framework hooks:
  - Winning with HTTP Interceptors: http://bit.ly/1PlpqhU
  - Centralized Application loading status: http://bit.ly/1ZGNnLD