# AngularJS and Ionic

## Building Mobile Applications

by Aliaksandr Tarasevich

Smart Web Squad

# Aliaksandr Tarasevich

- Co-founder of Smart Web Squad
- Work with Progress ~10 years (starting from 9.1D)
  - CHUI / GUI / GUI for .NET
- Building hybrid Mobile solutions ~4 years
  - jQuery Mobile
  - mGWT
  - Ionic Framework
- Was a big fan of GWT (and still like it)
- 1/2 projects built using MongoDB with NodeJs

# GWT

**Why we chose it:**

1. It's Java
    1. Team still can write back-end and front-end in Java
    2. You have all advantages of Java IDE
    3. You can use all cool Java tools your like (for example Maven)
    4. Need only a couple developers to wrap HTML/JS/CSS into Java classes
2. Compiler
    1. Generates optimized JS/HTML/CSS code for each browser
    2. Your generated code evolves as compiler evolving
    3. You can extend compiler (write generator) to generate any code
3. Debugging
    1. Debug your application right in IDE
4. uiBinder
    1. Keep UI layer separate (designer / developers separation)
    2. Allow to write UI in natural way (using HTML/CSS)
5. GWT Designer
    1. Drag-and-drop designed (very similar to PDSOE)

**Why we moved from it:**

1. It's Java, but
    1. Your web team MUST have JS/CSS/HTML experience to build good UX
    2. You need to compile code to see result:
        1. GWT DevMode is dead

            Browsers stopped supporting APIs
        2. GWT SuperDevMode was very slow

            Supposedly fixed using incremental compile (available in 2.7)
    3. Many popular WEB libraries are not available in GWT

        You can integrate any library you like using JSNI, but this can be time consuming
2. Compiler
    1. Today's browsers compatibility is not that big a problem
3. Debugging
    1. You write code in Java, but debug in JavaScript (in browser)

        Since DevMode is dead you have to use SuperDevMode which doesn't keep connection to IDE anymore

        Source maps help with this, but since original code in Java, fixing the found issues is not always trivial
4. GWT Designer
    1. Is officially dead


ANGULARJS
by Google

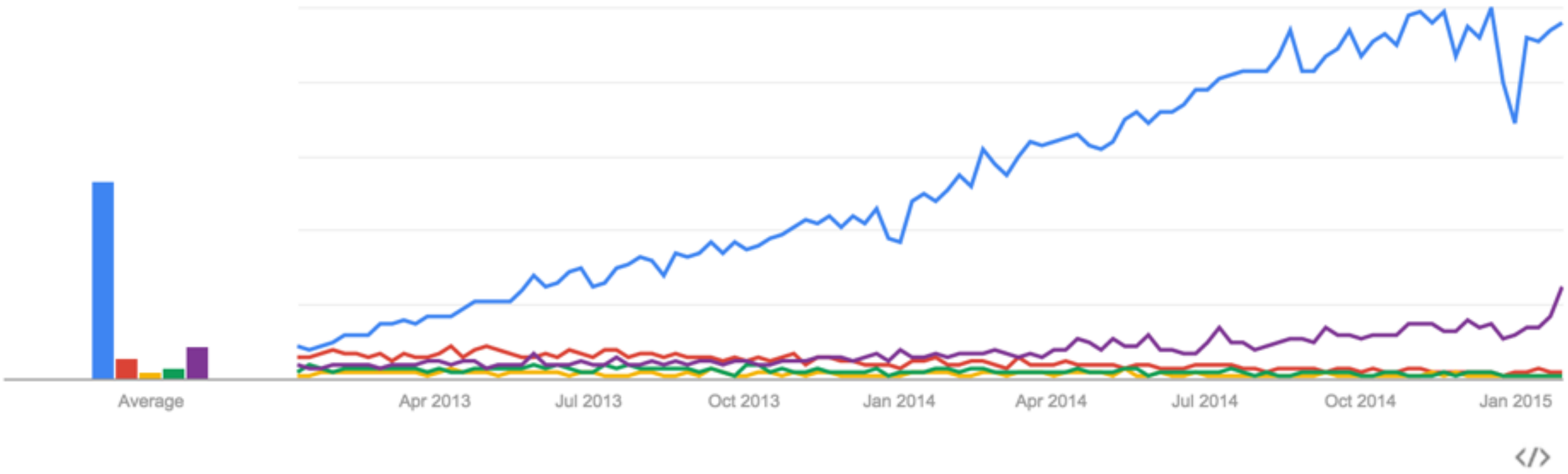
- Fast JavaScript MVW Framework
- A complete client-side solution
- Currently uses jQuery Lite (in Angular 2.0 will be replaced)
- Has small footprint ~135Kb (zipped ~50Kb)
- Built with unit testing in mind
- Used to build SPA (Single Page Applications)
- Supported by Google and a very big community

Compare Search terms

| angularjs | backbon... | ember.js | knockoutjs | react |
|---|---|---|---|---|
| Search term | Search term | Search term | Search term | Search term |

Interest over time

Compare to category  News headlines  Forecast




ANGULARJS by Google

Custom Directives

Modular

Built in services

All JavaScript

Dependency Injection

Data Binding

Filters

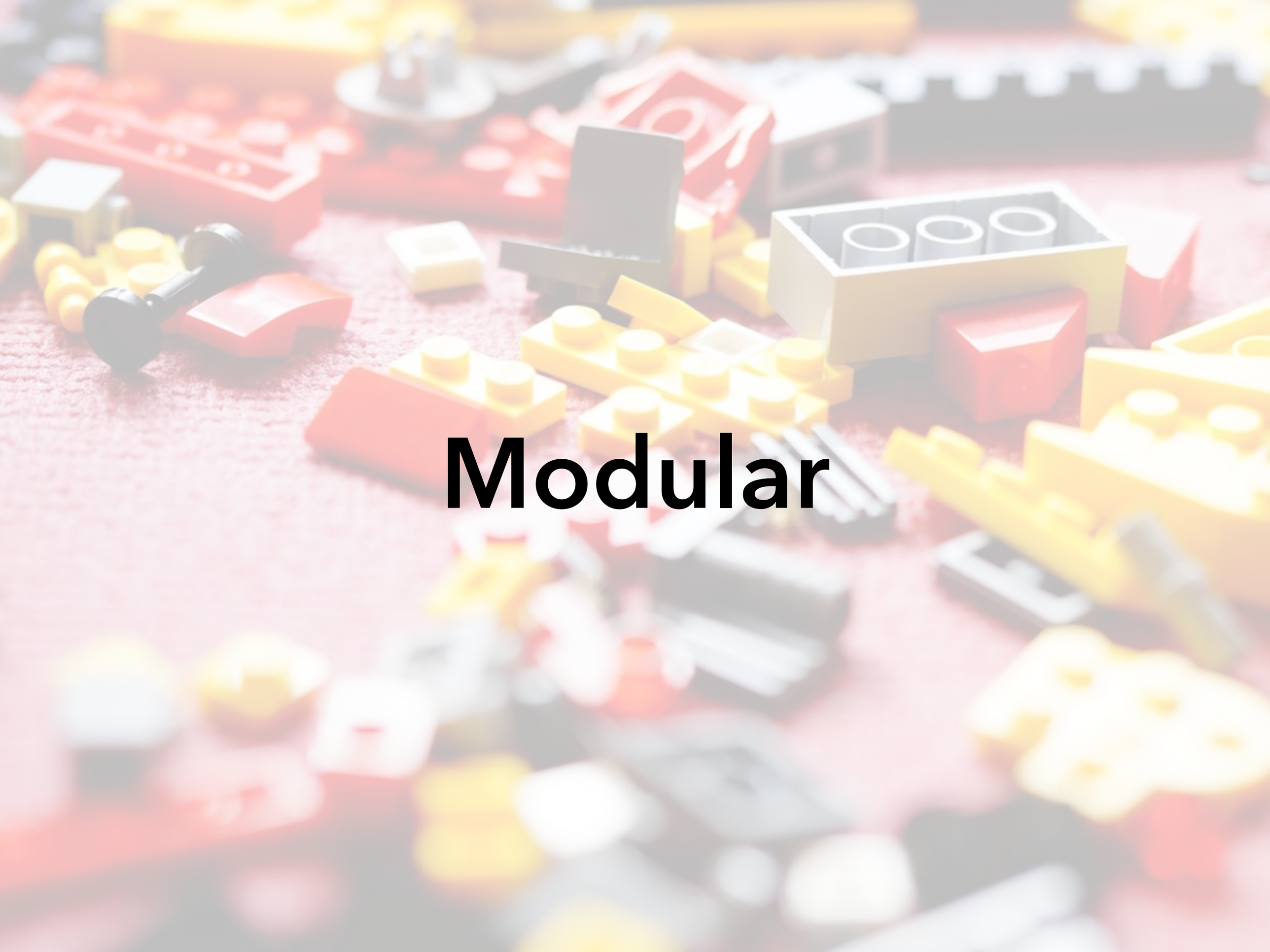
Fast

Unit Testing

Templates

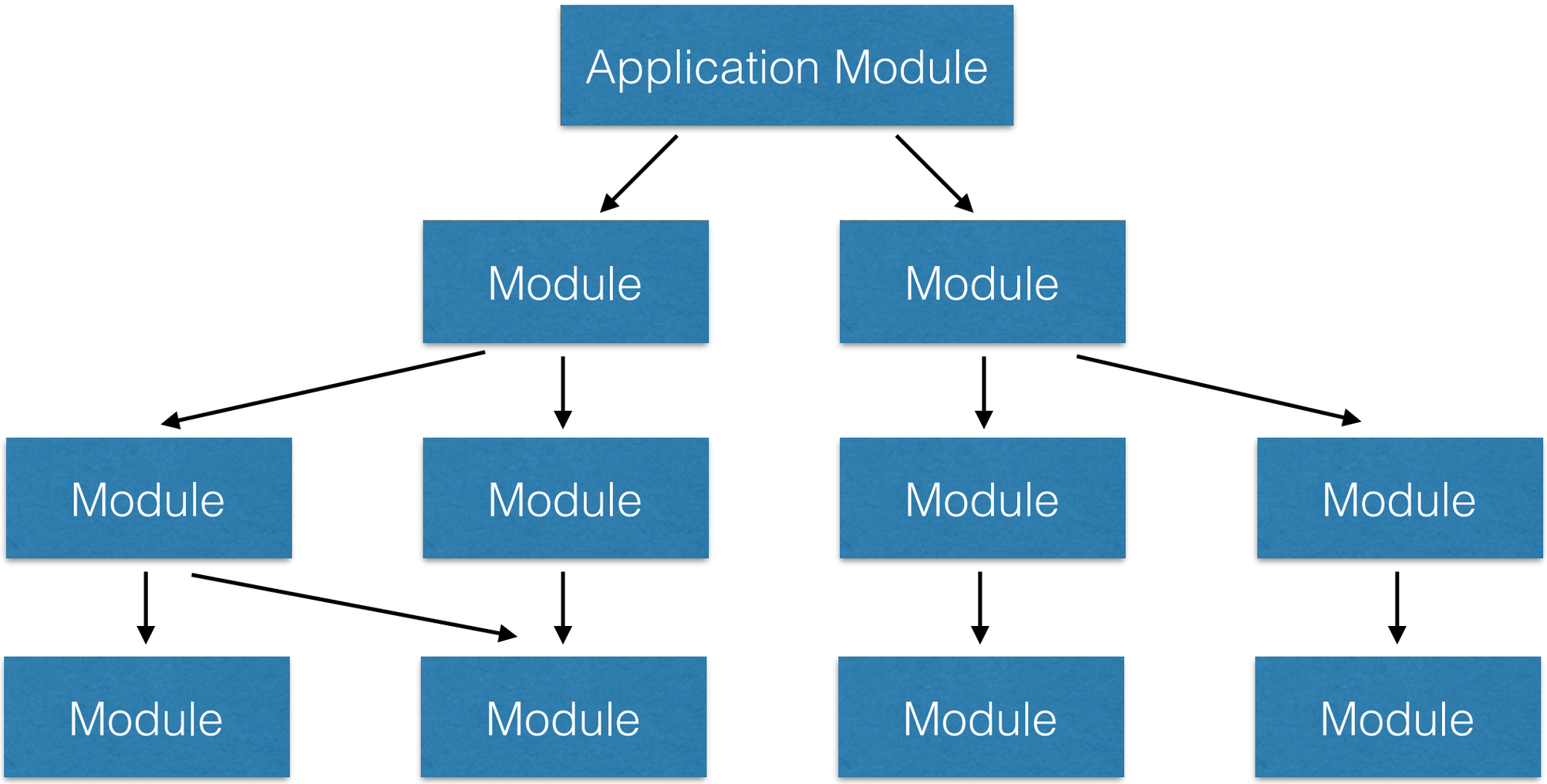
End-to-end testing

Fast Prototyping



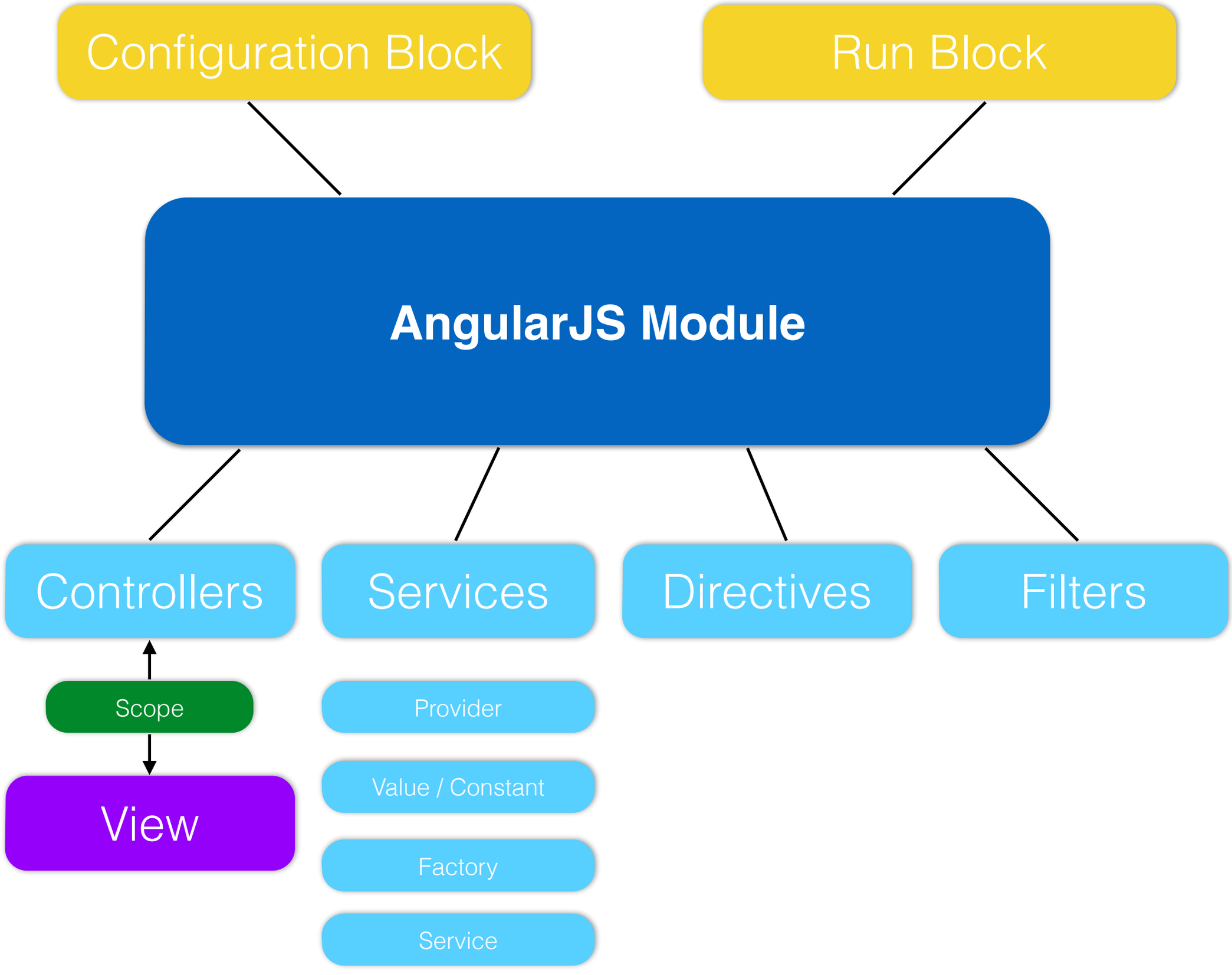
# Modular

# AngularJS Application




## Dependency Injection

The dependency injection in AngularJS allows you to declaratively describe how your application is wired. The Angular injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested.


Configuration Block
Run Block
AngularJS Module
Controllers
Services
Directives
Filters
Scope
View
Provider
Value / Constant
Factory
Service

```
angular.module(‘appModule’,['dependencyModule']).

config(function ($myDataAccessProvider, $myLocalStorageProvider) {
    // configure data-access endpoint
    $myDataAccessProvider.endpoint('customer')
                         .type('secure')
                         .route('/customer')
                         .trackChanges(true)
                         .stripPDS('dsCustomer', 'ttCustomer')
                         .cache(true);
    // configure local-storage namespace
    $myLocalStorageProvider.namespace('debuglogger')
                           .autoIncrementKey(true);
}).

factory('customerSvc', function($myDataAccess) {
    return {
        updateAddress: function(custNum, newAddress) {
            return $myDataAccess.customer.get({custNum: custNum})
                                         .then(function(customers) {
                angular.extend(customers[0], newAddress);
                return $myDataAccess.customer.saveChanges(customers);
            });
        }
    }
}).

run(function($myLocalStorage) {
    $myLocalStorage.debuglogger.clear();
    $myLocalStorage.debuglogger.add(‘Application Started...’);
})
```

# Scope

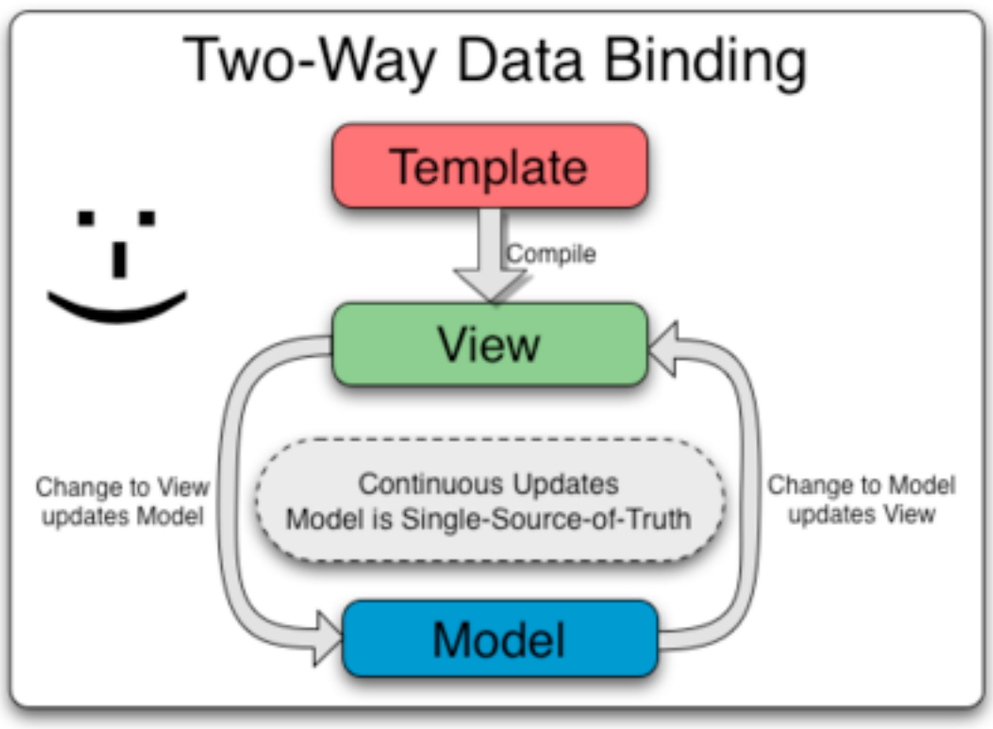
Scope is the glue between application controller and the view

# Controllers

Controllers are the behavior behind the DOM elements. AngularJS lets you express the behavior in a clean readable form without the usual boilerplate of updating the DOM, registering callbacks or watching model changes.

# Data-Binding

Is an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes. This is awesome because it eliminates DOM manipulation from the list of things you have to worry about.




**Terms:**

- Watchers
- Dirty-checking
- Digest cycle

**Performance Tips:**

- Don’t use watchers everywhere just because you can
- Keep logic within watch simple
- Use bind-once when you can
- Use $scope.$digest() vs $apply
- Keep number of watchers low:
  - Desktop: below 2000
  - Mobile: below 1000

# Directives

Let you invent new HTML syntax, specific to your application. Directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler ($compile) to attach a specified behavior to that DOM element or even transform the DOM element and its children

Directives allow us to create reusable components. A component allows you to hide complex DOM structure, CSS, and behavior. This lets you focus either on what the application does or how the application looks separately.

ng-app

ng-class

ng-show

ng-style

ng-if

ng-focus

ng-controller

ng-click

# Built in directives

ng-options

ng-required

ng-href

ng-src

ng-repeat

ng-init

ng-blur

ng-switch

ng-disabled

# Define Directive:

```
directive('clientInfo', function() {
    return {
        restrict: 'E',
        bindToController: {
            address: '='
        },
        controller: ‘CliInfCtrl as cliInfCtrl',
        templateUrl: 'client.info.tpl.html'
    }
});
```

# client.info.tpl.html (directive HTML template):

```
<address>
    <strong>{{cliInfCtrl.address.title}}</strong><br>
    {{cliInfCtrl.address.address1}}<br>
    {{cliInfCtrl.address.address2}}<br>
    {{cliInfCtrl.address.city}} {{cliInfCtrl.address.state}} {{cliInfCtrl.address.zip}}<br>
    {{cliInfCtrl.address.phone | phone}}
</address>
```

# On HTML Page:

```
<client-info address="addressObject"></client-info>
```

# Result

**Smart Web Squad, LLC**
10103 Angular Ave
Richmond, VA 23233
(804) 396-08-12

```
<app-form security-group="Manager" title="GENERAL.CUSTOMERS" theme="positive">

    <header-panel height="30%">
        <!-- Data Grid -->
        <ui-grid data="controller.customers" config="controller.customerGridConfig"
                 reorder-columns="true" add-rows="true" inline-edit="true">
        </ui-grid>
    </header-panel>

    <body-panel height="50%">
        <tabset view-mode="vertical" type="pills">
            <tab select="controller.onTabSelect('General')" heading="GENERAL.GENERAL">
                <input type="text" ng-model="controller.activeCustomer.Name">
                <input type="text" ng-model="controller.activeCustomer.Address">
                <input type="text" ng-model="controller.activeCustomer.Address2">
                <input type="text" ng-model="controller.activeCustomer.City">
            </tab>
            <tab select="controller.onTabSelect('Orders')" heading="GENERAL.ORDERS">
                <ui-grid data="controller.orders" config="controller.ordersGridConfig"
                         reorder-columns="false" add-rows="false" inline-edit="false">
                </ui-grid>
            </tab>
            <tab select="controller.onTabSelect('Feedback')" heading="GENERAL.FEEDBACK">
                <ui-grid data="controller.feedback" config="controller.feedbackGridConfig"
                         reorder-columns="false" add-rows="false" inline-edit="false">
                </ui-grid>
            </tab>
        </tabset>
    </body-panel>

    <footer-panel theme="balanced" height="20%">
        <signature title="GENERAL.SIGNHERE"></signature>
        <button ng-click="controller.saveChanges()" translate="GENERAL.SAVE"></button>
        <button ng-click="controller.closeForm()" translate="GENERAL.CLOSE"></button>
    </footer-panel>

</app-form>
```

# Services

- Use to organize and share code across application
- Lazily instantiated – only instantiates when a component depends on it
- Singletons – Each component dependent on a service gets a reference to the single instance generated by the service factory

Value / Constant

Factory

Service

Provider

# Factory

most often used recipe

```
angular.module('myApp', []).
    factory('customerModel', function(myDataAccess) {
        return {
            // variables
            activeCustomer: void 0,
            listOfCustomers: void 0,
            // functions
            loadCustomers: function loadCustomers() {
              return myDataAccess.customer.get();
            }
        }
    })
    .controller('MyController', function(customerModel) {
        if (!customerModel.listOfCustomers) {
            this.customers = customerModel.loadCustomers();
        }
    });
```

# Routing (uiRouter)

http://www.smartwebsquad.com

hashbang (default)

http://www.smartwebsquad.com/#/**pricing**

html5 pushState

http://www.smartwebsquad.com/**pricing**

# Nested States & Views

Services: https://smartwebsquad.com/**pricing**

Services: https://smartwebsquad.com/**services**

```
$stateProvider
    .state('app', {
        url: "",
        abstract: true,
        templateUrl: 'main-page.tpl.html'
   });
```

```
$stateProvider
    .state('app.pricing', {
        url: "/pricing",
        controller: 'PricingCtrl as pricingCtrl',
        templateUrl: 'pricing-page.tpl.html'
    });
```

```
$stateProvider
    .state('app.services', {
        url: “/services/:serviceId”,
        controller: 'ServicesCtrl as servicesCtrl',
        templateUrl: 'services-page.tpl.html'
    });
```

main-page.tpl.html

```
<header>...</header>
<div ui-view></div>
<footer>...</footer>
```

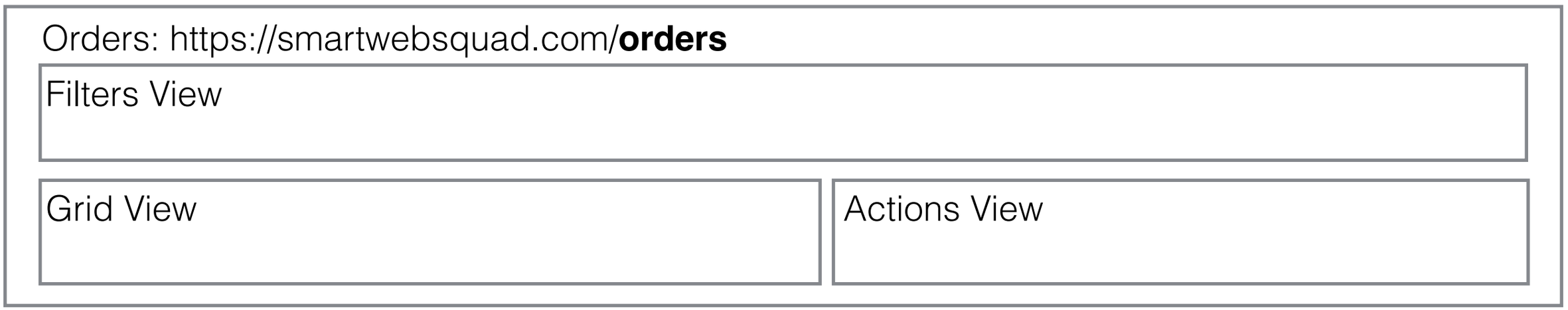
pricing-page.tpl.html

```
<div>
    Pricing Page Content
</div>
```

services-page.tpl.html

```
<div>
    Services Page Content
</div>
```

# Multiple & Named Views




```
<div>
    <div ui-view="filters"></div>
    <div ui-view="grid"></div>
    <div ui-view="actionbar"></div>
</div>
```

```
$stateProvider
    .state('orders',{
        views: {
            'filters': {
                templateUrl: 'orders-filters.html',
                controller: function(){ ... controller stuff just for filters view ... }
            },
            'grid': {
                templateUrl: 'orders-table.html',
                controller: function(){ ... controller stuff just for grid view ... }
            },
            'actionbar': {
                templateUrl: ‘orders-action-bar.html',
                controller: function(){ ... controller stuff just for actions view ... }
            }
        }
    })
```

PROGRESS

OPENEDGE

# Angular Services

## $http (use to make API calls or to build custom DA framework)

$http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP

## $resource (use for applications with RESTful web API)

factory which creates a resource object that lets you interact with RESTful server-side data sources

## JSDO (use if you don’t want to spend any time on DA framework)

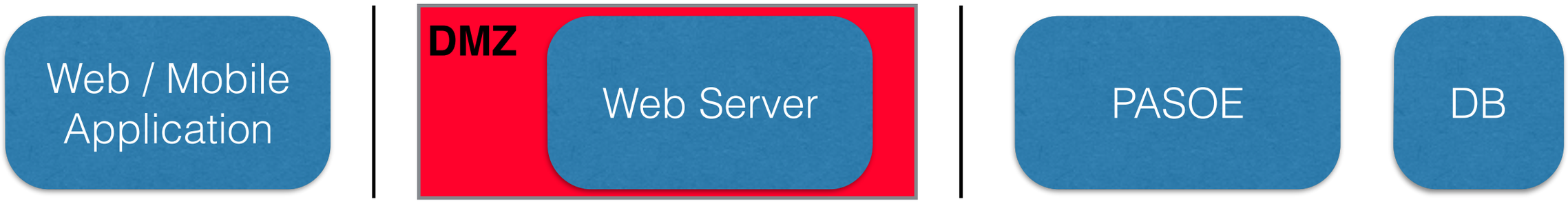
provides support for a complex data model and API to manipulate that data while maintaining data integrity. The JSDO catalog defines the logical schema and mapping to a remote data source

# OpenEdge Services

**WebSpeed (use if you still on OE < 11.2)**


**REST Adapter**


**Pacific AppServer (with REST Adapter)**




ionic

- Open-source
- Built with Sass and optimized for AngularJS
- Beautifully designed
- Extends the HTML vocabulary
- UI Components using Directives and Services
- Proven for large-scale app development
- Ionicons (over 700 MIT licensed font-icons)
- Supported by Drifty and has a large community:
    - Very active internal forum

# Hybrid Apps

- HTML 5 that acts like native
- Web wrapper in native layer
- Direct access to native APIs
- A single code base
- Familiar web development environment

# Web technologies you already know

# A lot of components

- Side menus
- Actionsheets
- Tabs
- Pull to Refresh
- Slidebox
- Infinite Scroll
- Swipeable List Options
- Popup
- Popover
- Loading Overlay
- Inputs
- Buttons

Go to http://ionicframework.com/docs/components to see more

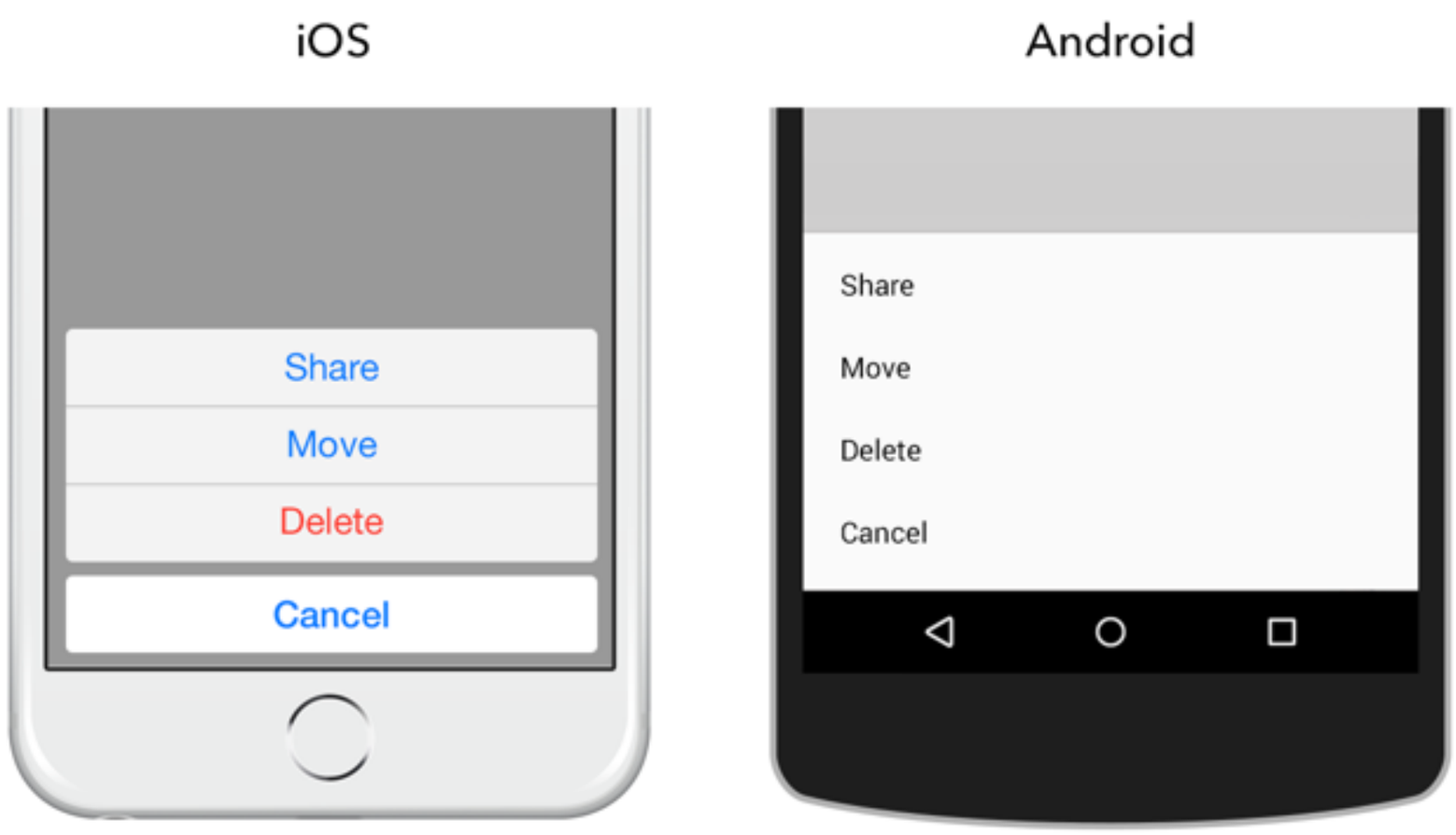
# Cached Views

- View elements left in DOM
- $scope disconnected from cache
- State maintained
- Scroll position maintained
- Life Cycle events
- Highly configurable

# Collection-Repeat

- Replacement for ng-repeat
- Scroll through thousands of items
- Only renders the viewable items
- Smooth scrolling

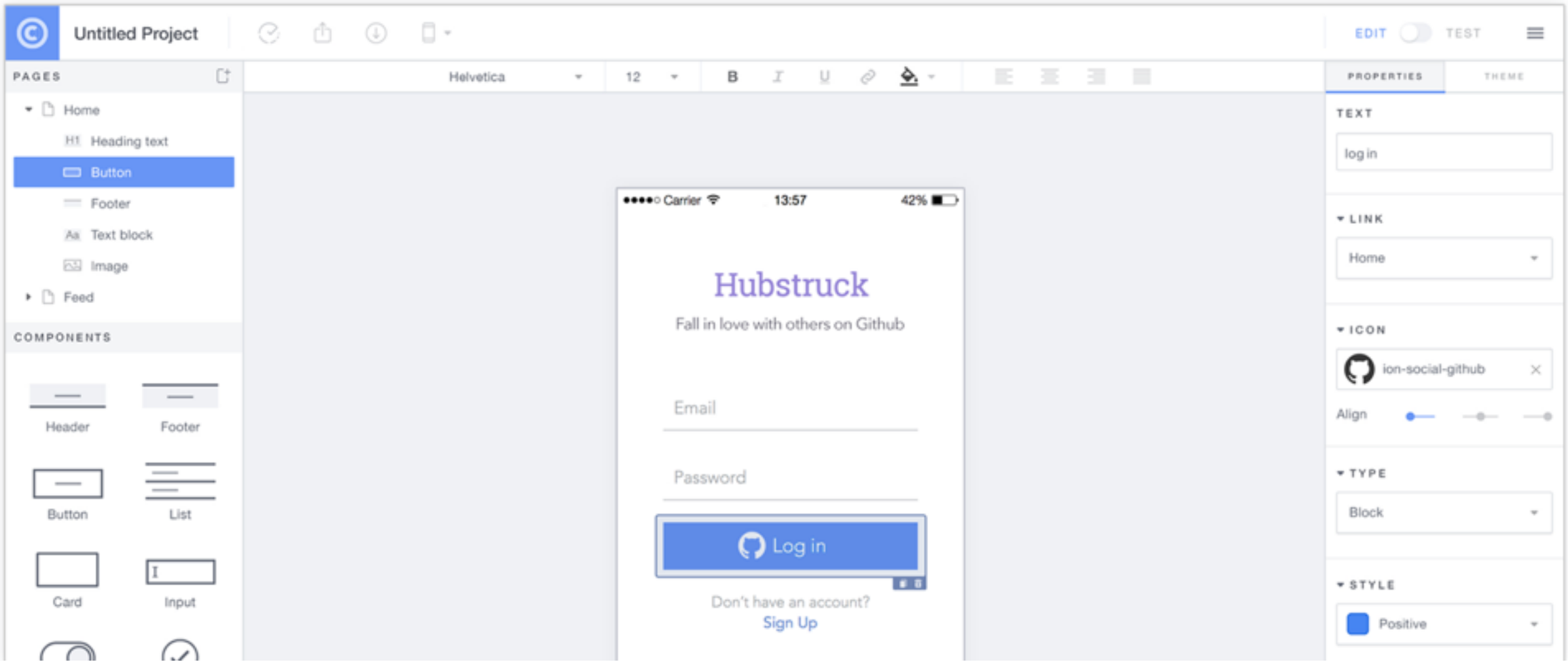
# platformcontinuity



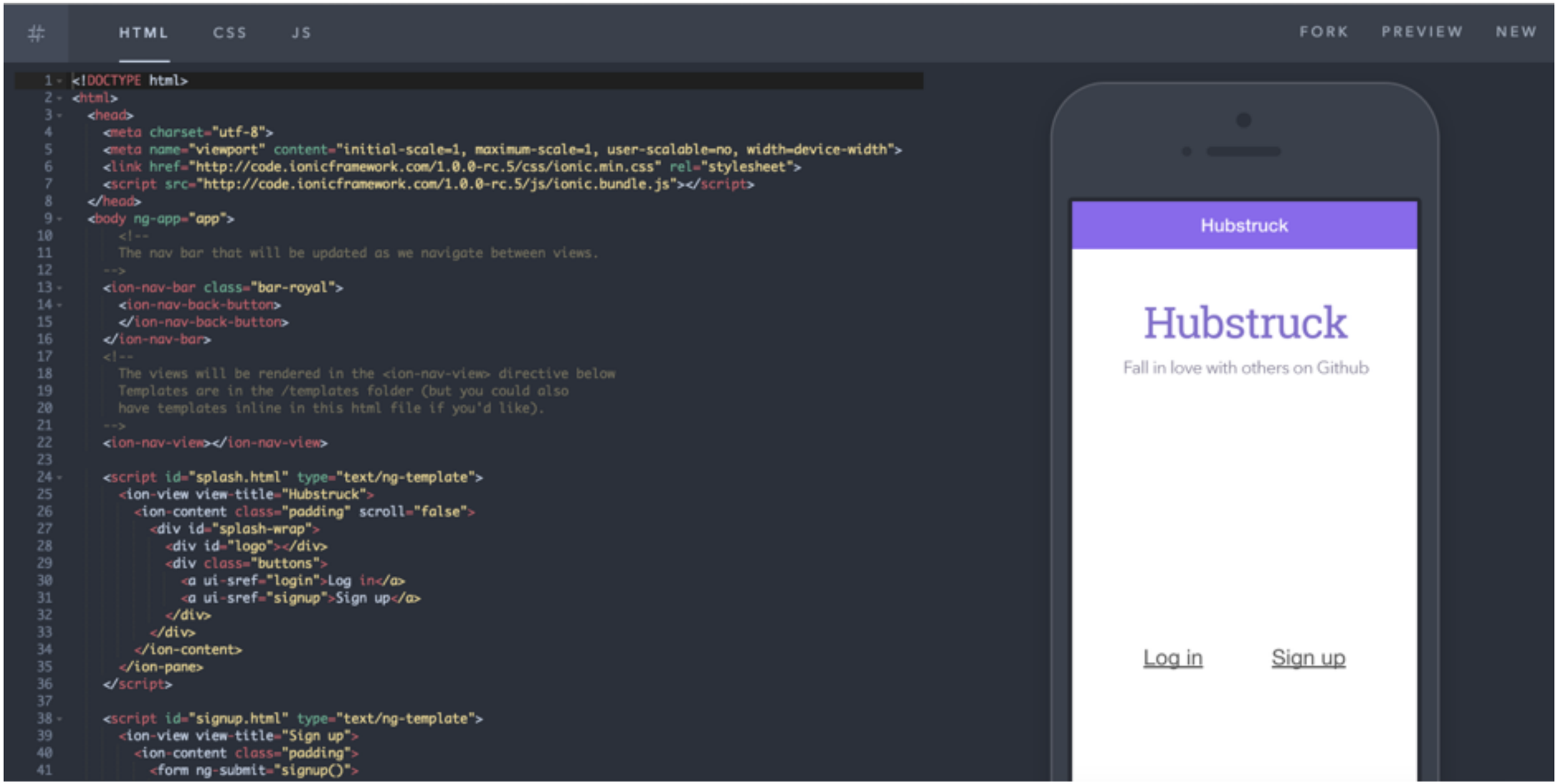
# Ionic-Cli

- Testing in a browser
- Live Reload App During Development
- Emulating your app
- Running your app on device
- Building your app (with or without SDK)
- Icon and Splash Screen Image Generation
- Crosswalk for Android

# ioniccreator



# Ionic **Playground**



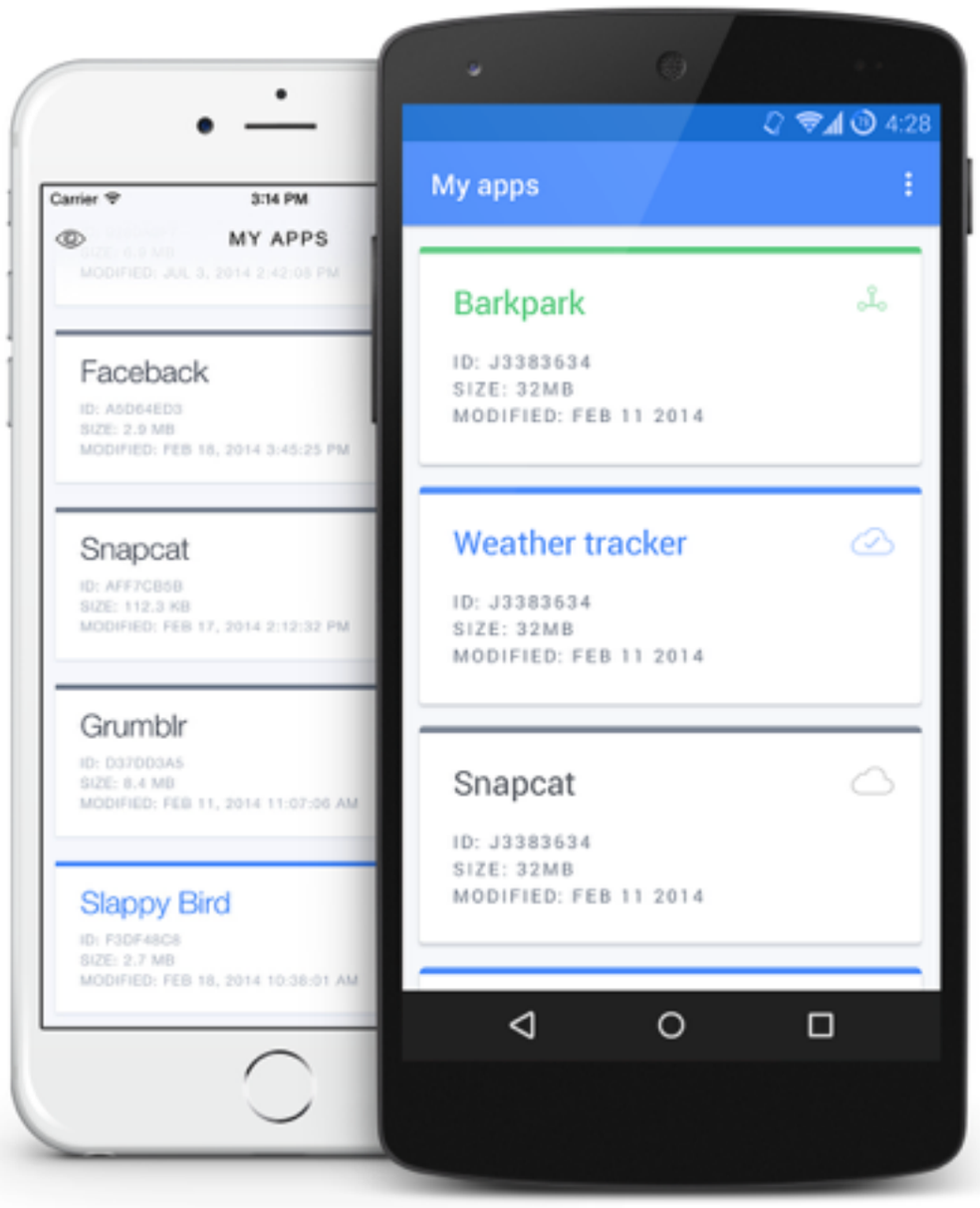
# ngCordova

CORDOVA WITH THE POWER OF ANGULARJS

**ngCordova** is a collection of **63+** AngularJS extensions on top of the Cordova API that make it easy to build, test, and deploy Cordova mobile apps with AngularJS.

# ionicview

Ionic View makes it easy to share your Ionic and Cordova apps with clients and testers around the world, all without ever going through the App Store.



# Questions?

aliaks@smartwebsquad.com