Mobile and Grails

The Search is indeed over

Zachary Lendon
Object Partners
@zachlendon

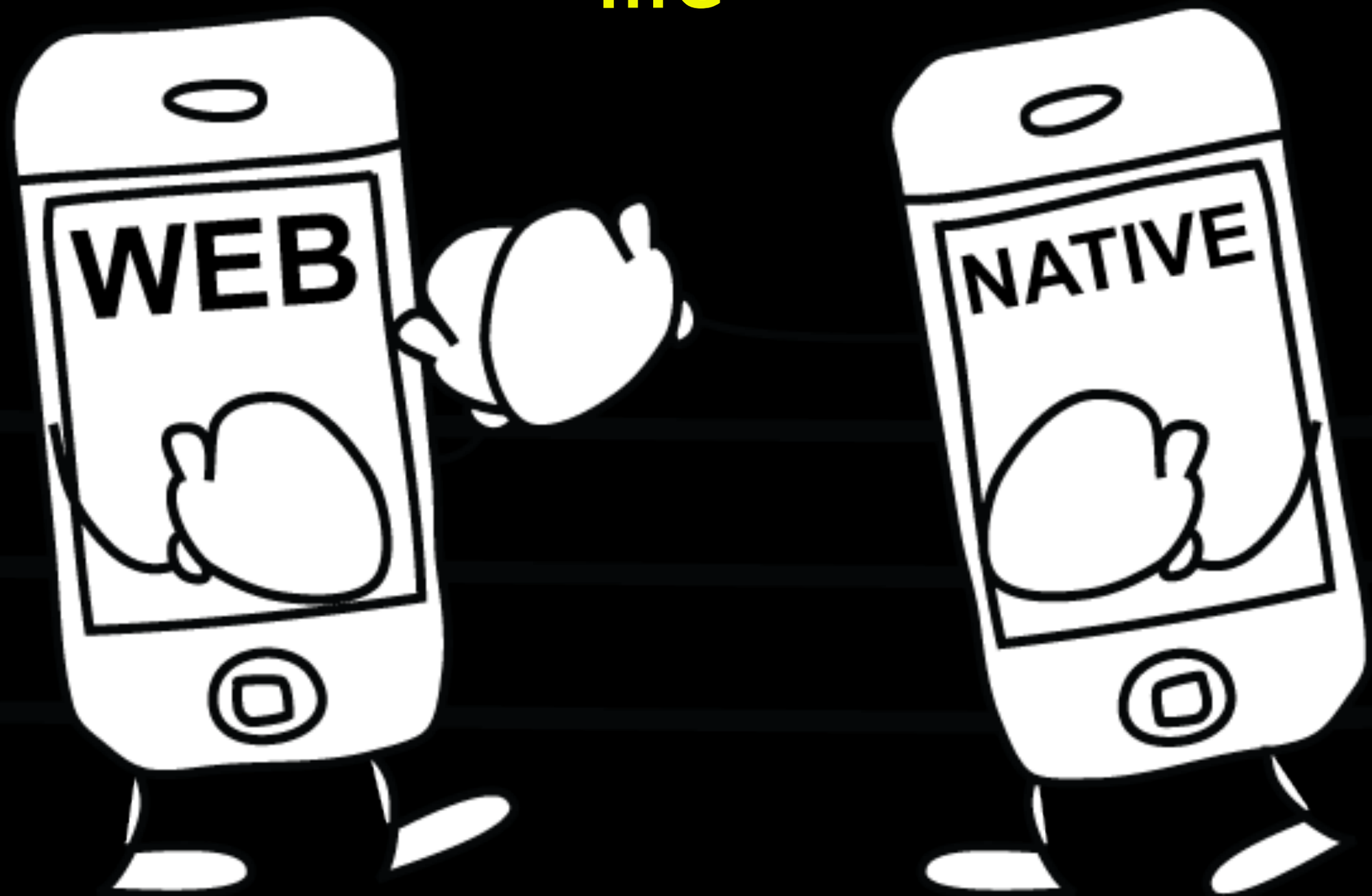
Follow https://github.com/zachlendon for update of demo project from this talk

3 lead drivers of discussion

Fat is in!
(clients that is)

- Server is for limited uses
  - Security
  - Files
  - Data (longer-lived)
    - And many of these are pushed to 3rd party services
- Client is for everything else

Web wanting to live the native life



Spaghetti code

To which you try to

And before you know it you're just trying to just hold things together

Are these our solutions to these issues?

Spine

Kendo UI
THE ART OF WEB DEVELOPMENT

dynamo

The Truth About "Solutions"

"A solution's merit is tied directly to the skill/knowledge/career/biases of the person swinging the solutions"

-Cody Lindley

Talk Agenda

To help make you better at swinging solutions with Grails and other best of breed mobile solutions that best fit your application's needs

Beyond looking and functioning great (a given), awesome mobile web apps are developed with

- A loosely coupled architecture
- Functionality broken down into smaller independent modules
- Framework* or library agnostic
  - *Except for Grails
- Where Testing, TDD, Testability, etc., etc. - concerns as a **main** priority

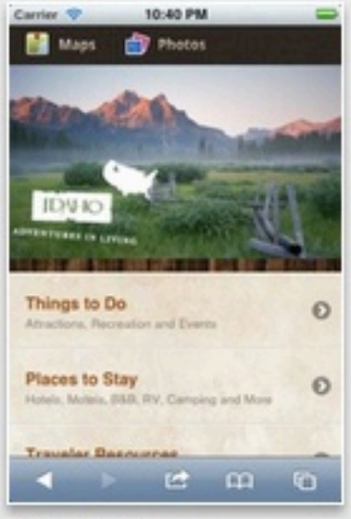
We should be creating "Maintainable Javascript"* that lasts 5 years

* term and timeframe by Nicholas Zakas

“Standard” Building Blocks: The Mobile End

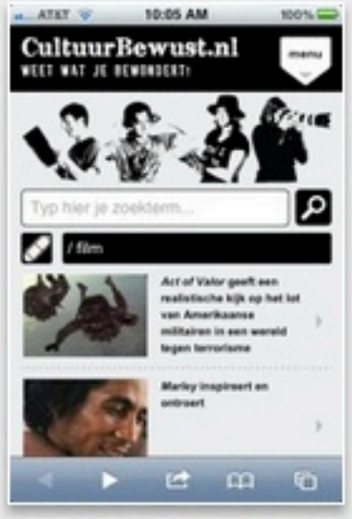
jQuery Mobile Gallery

Verizon Wireless

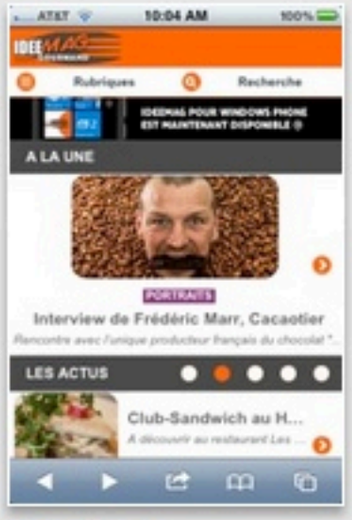


Visit Idaho

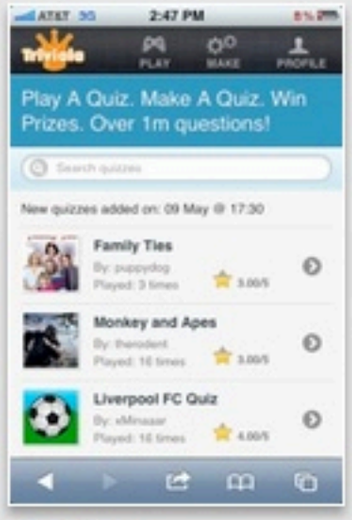
Arby's



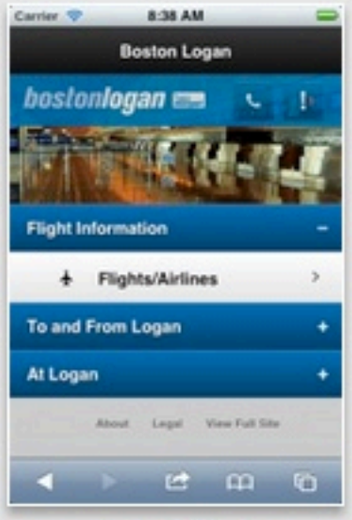
CultuurBewust



Ideemag

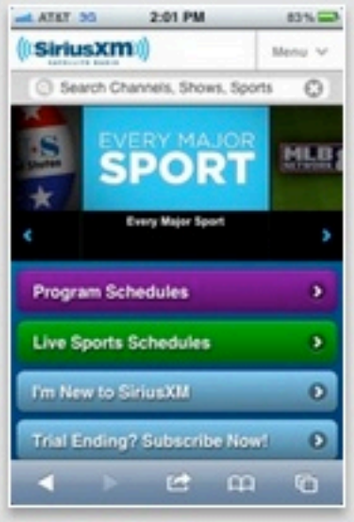


Triviala

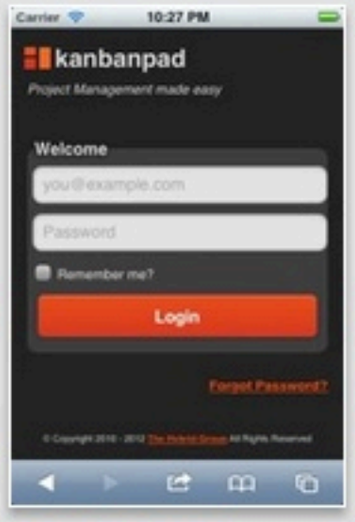


Boston Logan A...

Humana



SiriusXM

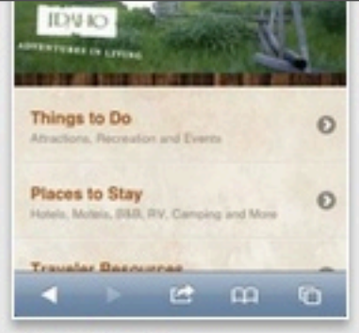


kanbanpad

United Healthca...

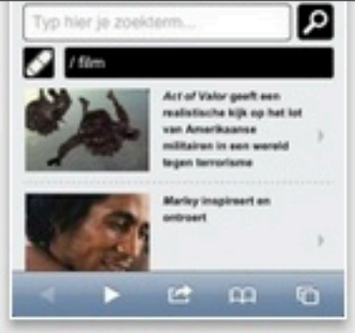
Fiat Spain

Verizon Wireless

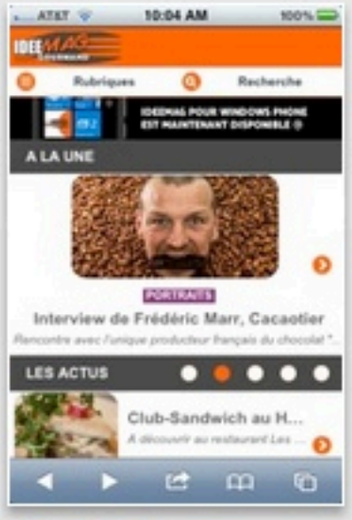


Visit Idaho

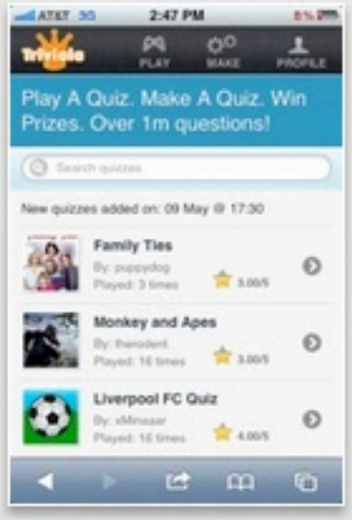
Arby's



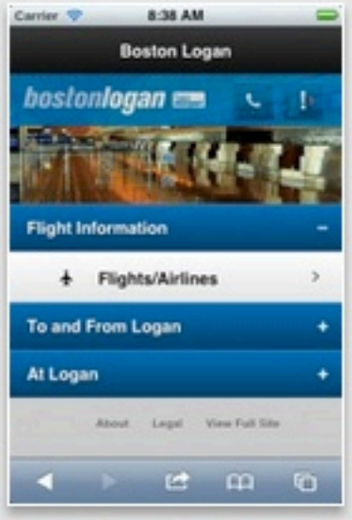
CultuurBewust



Ideemag

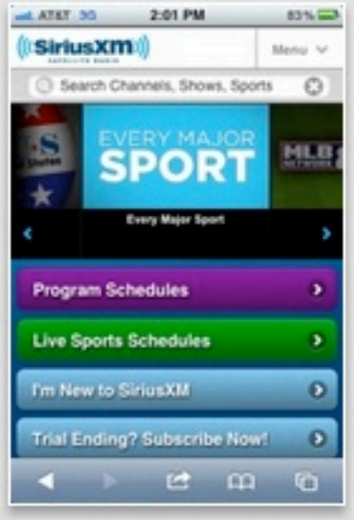


Triviala

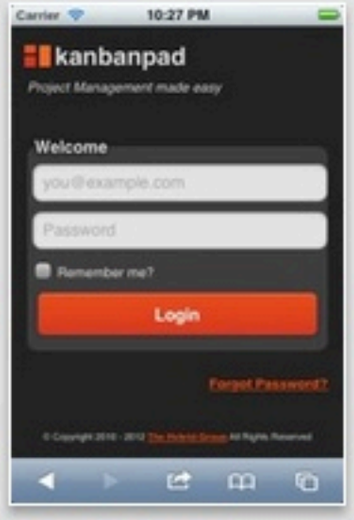


Boston Logan A...

Humana



SiriusXM



kanbanpad

United Healthca...

Fiat Spain

JQuery Mobile

- A one size fits most framework
- Unified UI across all major platforms
  - new and old, mobile and not
- Uniform design
  - that is theme-able
  - many native-looking design elements
- Leverages semantic markup and progressive

JQuery gives us

- AJAX
- DOM manipulation/traversal/etc
- Rich API for event handling
- Huge plugin community access
- Huge knowledge base - people, docs, etc.

But they don't give us

- Unit Testing
- Browser State Management
- Architecture Patterns (MV*)
- Dependency Injection
- Cross-browser persistent storage
- Communication patterns (pub/sub)
- Dynamic Transitions
- Client-side templating
- HTML5/CSS Feature detection
- and more...

Many of which are needed for large-scale javascript applications

And when do we need all of these things? When building large-scale javascript applications

"Large-scale JavaScript apps are non-trivial applications requiring significant developer effort to maintain, where most heavy lifting of data manipulation and display falls to the browser" -Addy Osmani

Model-View-Controller (MVC)

- Model - Data
- View - Presentation
- Controller - User input

Example*: Ember.js
* - For the most part

Model-View-Presenter (MVP)

- P (Presenter) acts as the controller
  - Retrieves data (model)
  - Formats it for display (in view)
- Controller responsibility in view
  - View handles user-input

Example*: Backbone.js

Model-View-ViewModel (MVVM)

- MVC with a ViewModel for data bindings between model and view
  - Takes model data and passes to view

Examples*: Knockout.js

Model-View-*

- Everything else
- Also often referred to as MVW - (Whatever you want) or MVV (Model-View-Variation)
- A conglomeration of the former patterns
- What predominance of client-side market choices actually are

Structuring the client side (MV* generalities)

Models

What unique data does my app represent

Spaghetti code

What does the user need to see and do?

What repeatable output can I shift to JS templates?

Views

If a router, what happens when a route is navigated to (varying approaches) /#route

Controllers

Module pattern - Holy Grail of Javascript Patterns

```javascript
var module = (function () {
    // private variables and functions
    var foo = 'bar';

    // constructor
    var module = function () {
    };

    // prototype
    module.prototype = {
        constructor: module,
        something: function () {
        }
    };

    // return module
    return module;
})();

var my_module = new module();
```

Modules

- Emulate classes through public and private function/variables so that parts of an object can be protected from global scope
- Based on object-literal notation
  - Set of name/value pairs enclosed with curly braces
- Work independently
  - helps with GC and scope chain walking

Modules

- Localized

```
var module = (function (window, document, $) {
    // module stuff
})(window, document, jQuery);
```

- Extensible

```
var module = (function (module) {
    module.prototype.something_else = function () {
    };
    return module;
})(module);
```

Further module reading topics

- AMD (Asynchronous Module definition)
  - asynchronously load modules and dependencies
- CommonJS
  - Uses exports object containing objects to export and require function modules can use to import exports of other modules
- AngularJS

Facade pattern

- Simplified: An interface to our modules

Simplifying usage of a module through a limited API

- $(document).ready()
- $(el).animate()
- $(el).css()

Mediator pattern

- Promoted loose coupling by handling interaction between components instead of components talking to each other themselves
- Essentially a shared subject in the Observer Pattern
- Example in MV* frameworks is State Manager in Ember

“Standard” Building Blocks: Grails

Leveraging RESTful Grails support, which includes

Leveraging simple RESTful controller

```
class UrlMappings {

    static mappings = {
        "/$controller/$id?"{
            action = [GET:"show", POST:"save", PUT:"update", DELETE:"remove"]
        }
```

To Grails controller renders

HomeController.groovy

Home | HomeController | Home Views | Tests

```
    }

    def headline() {
        renderData()
    }

    def messages() {
        renderData()
    }

    def schedule() {
        def scheduleList = Schedule.list()
        render(contentType: "text/json") {
            for (Schedule schedule in scheduleList)
            schedule {
                id = id
                events {
                    for (Event e in scheduleList.events) {
                        event(name: e.name, date: e.date,
                                imageLocation: e.defaultImageLocation)
                    }
                }
```

To full-fledged groovy.json.JsonBuilder use

MessageConverterService.groovy

```
class MessageConverterService {

    JsonBuilder toJSONBuilder(Message message, def activeId = "") {

        groovy.json.JsonBuilder builder = new JsonBuilder()

        def root = builder {
            id message.id
            if (activeId == message.id) {
                currentActive true
            }
            if (message.headline) {
                headline message.headline
            }
            text message.text
            if (message.imageName) {
                imageName message.imageName
            }
            def contactMap = [:]
            contactMap.firstName = message.contact.firstName
            contactMap.lastName = message.contact.lastName
            contactMap.businessName = message.contact.businessName
            contactMap.defaultImageLocation = message.contact.defaultImageLocation
            contactMap.twitterHandle = message.contact.twitterHandle
```

```
JsonBuilder toJSONBuilder(Message message, def activeId = "") {

    groovy.json.JsonBuilder builder = new JsonBuilder()

    def root = builder {
        id message.id
        if (activeId == message.id) {
            currentActive true
        }
        if (message.headline) {
            headline message.headline
        }
        text message.text
        if (message.imageName) {
            imageName message.imageName
        }
        def contactMap = [:]
        contactMap.firstName = message.contact.firstName
        contactMap.lastName = message.contact.lastName
        contactMap.businessName = message.contact.businessName
        contactMap.defaultImageLocation = message.contact.defaultImageLocation
        contactMap.twitterHandle = message.contact.twitterHandle
        contact contactMap
    }
    root
}
}
```

To Good old fashion Spring Framework JSON View Resolvers

JsonViewResolver.groovy

```
package socialtracker

import org.springframework.web.servlet.View
import org.springframework.web.servlet.ViewResolver
import socialtracker.converters.MessageConverterService

class JsonViewResolver implements ViewResolver {

    MessageConverterService messageConverterService

    View resolveViewName(String viewName, Locale locale) {
        return viewName?.contains("json") ?
            new JsonView(messageConverterService:
                    messageConverterService) : null
    }
}
```

```
class JsonView implements View {

    def messageConverterService

    String getContentType() {
        return "text/json"
    }

    void render(Map model, HttpServletRequest request,
                HttpServletResponse response) {
        getViewForModel(model).render(model, request, response)
    }

    def getViewForModel(Map model) {
        return getViewForViewable(model.result)
    }

    def getViewForViewable(viewable) {
        switch (viewable) {
            case Message:
                return new MessageView()
                break
            case Schedule:
                return new ScheduleView()
                break
        }
    }

}
```

```
void render(Map model, HttpServletRequest request,
            HttpServletResponse response) {
    getViewForModel(model).render(model, request, response)
}

def getViewForModel(Map model) {
    return getViewForViewable(model.result)
}

def getViewForViewable(viewable) {
    switch (viewable) {
        case Message:
            return new MessageView()
            break
        case Schedule:
            return new ScheduleView()
            break
    }
}

}
```

MessageView.groovy

```
package socialtracker

import ...

class MessageView implements View {

    def messageConverterService

    String getContentType() {
        return "text/json"
    }

    def buildJSON(Map model) {
        messageConverterService.
                toJSONBuilder(model.message).toString()
    }

    void render(Map model, HttpServletRequest request,
                HttpServletResponse response) {
        def result = buildJSON()
        def writer = response.writer
        writer.println result
    }
}
```

```
import ...

class MessageView implements View {

    def messageConverterService

    String getContentType() {
        return "text/json"
    }

    def buildJSON(Map model) {
        messageConverterService.
                toJSONBuilder(model.message).toString()
    }

    void render(Map model, HttpServletRequest request,
                HttpServletResponse response) {
        def result = buildJSON()
        def writer = response.writer
        writer.println result
    }
}
```

So as you can see, we have lots of ways to get data to the client. Now what?

Grails GSPs and Resources Pipeline both are great for tieing together an application's client-side pieces

index.gsp

gsp:root r:script

Home HomeController:index Home Views Tests

```
<head>
    <meta name="layout" content="main"/>
    <r:require module="status"/>
</head>
<body>
    <div data-role="content">
    <r:script>
        SocialTracker.init(${json.messages},
        ${json.headlines}, ${json.events});
    </r:script>
    </div>
</div>
</body>
```

main.gsp

gsp:root

```
    <!-- Main Stylesheet -->

    <link rel="stylesheet" href="${resource(dir: 'css/vendor/jquery/', fi
    <link rel="stylesheet" href="${resource(dir: 'css', file: 'style.css'
    <link rel="stylesheet" href="${resource(dir: 'css', file: 'list.css')
    <g:layoutHead/>
    <r:layoutResources />
</head>
<body>

<div data-role="page" data-theme="f">
    <g:render template="/common/header"/>
    <div class="content" data-role="content">
        <g:layoutBody/>
        <ul data-role="listview" data-theme="f">
            <li><a href="upcomingEvents" class="text">POST A MESSAGE</a></
            <li><a href="leaderboard" class="text">SIGN UP FOR UPDATES</a
        </ul>
    <g:render template="/common/footer"/>
</div>

<r:layoutResources/>
</body>
</html>
```

ApplicationResources.groovy

```
modules = {
    core {
        resource url:'js/vendor/jquery/jquery-1.7.2.js'
        resource url: 'js/global-init.js'
        resource url:'js/vendor/jquery/jquery.mobile-1.1.0.js'
        resource url:'js/vendor/backbone/underscore-1.3.3.js'
        resource url:'js/vendor/backbone/backbone-0.9.2.js'
    }
    dust {
        dependsOn 'core'
        resource url: 'js/vendor/dust/dust-core-1.0.js'
    }
    status {
        dependsOn 'dust', 'handlebars_templates'
        resource url:'js/pages/SocialTracker.js'
        resource url:'js/backbone/models/headline.js'
        resource url:'js/backbone/models/message.js'
        resource url:'js/backbone/collections/headlines.js'
        resource url:'js/backbone/collections/messages.js'
        resource url:'js/backbone/views/baseView.js'
        resource url:'js/backbone/views/headlineView.js'
        resource url:'js/backbone/views/messageView.js'
        resource url:'js/backbone/routers/SocialRouter.js'
        resource url: 'dust/headline.dust', attrs: [type: 'js'],
                bundle: 'bundle_status'
    }
    handlebars_templates {
        dependsOn 'core', 'handlebars_runtime'
```

```
        dependsOn 'core'
        resource url: 'js/vendor/dust/dust-core-1.0.js'
    }
    status {
        dependsOn 'dust', 'handlebars_templates'
        resource url:'js/pages/SocialTracker.js'
        resource url:'js/backbone/models/headline.js'
        resource url:'js/backbone/models/message.js'
        resource url:'js/backbone/collections/headlines.js'
        resource url:'js/backbone/collections/messages.js'
        resource url:'js/backbone/views/baseView.js'
        resource url:'js/backbone/views/headlineView.js'
        resource url:'js/backbone/views/messageView.js'
        resource url:'js/backbone/routers/SocialRouter.js'
        resource url: 'dust/headline.dust', attrs: [type: 'js'],
                bundle: 'bundle_status'
    }
    handlebars_templates {
        dependsOn 'core', 'handlebars_runtime'
        resource url: 'templates/message.handlebars', attrs: [type: 'js']
    }
}
```

Dynamic Transitions

- Transitions are an example of integrating native application behavior into web applications
- Typically using JQuery Mobile $.mobile.changePage and its accompanying events
- Sometimes will need to animate to a new view that you can't create until right before you want to go to it

Browser State Management

- Bookmarkable URLs
  - No hard-refresh when changing but can load up application state with them
  - Allows sharing of URLs (important)
- Done by changing location.hash (Backbone.History.navigate, RouteMap, History.JS)
  - PushState vs #!

Cross Browser Persistent Storage

- Solutions
  - Store.js
    - key/value with store.set, store.get, store.remove
    - store.clear()
  - PersistJS
  - JQuery Offline Plugin

Key MV* Differences

- MV* approach
- Bindings
- Approach to templating
- Approach to routing
- Approach to ‘views’

Binding of data between model and view

- Change listeners for dirty-checking
- Backbone - essentially a ‘roll your own’
  - Or use plugins like Backbone.ModelBinder or Rivets.js
- Ember - bi-directional - rich API between objects and tight integration with Handlebars
- Angular - bi-directional - uses ng-model attribute in DOM

Client Side Templating

- Cached and Rendered without an HTTP request to the server
- Logic-less - separation of concerns
  - Options exist for adding in conditions, helpers, variables and more

Client-side templating: framework perspective

- Most are flexible, while some provided enhancements for certain uses
- Backbone ships with Underscore templates
- Ember provides special binding capabilities with Handlebars

Handlebars vs DustJS frameworks

- Both provide all the same basic and most commonly requested advanced features including mustache syntax support (and configuration for other syntaxes), helpers, partials, conditionals, looping, etc
  - Helpers provide means to bring your favorite missing gsp tag functionality into the fold

Handlebars vs DustJS frameworks

- Dust provides inline parameters for conflict resolution between parent and child contexts
- Handlebars lets you backtrack ‘paths’, while dust.js believes you should pass parent context and only support nested paths

Routing

Angular

The App Module

app/js/app.js:

```
angular.module('phonecat', []).
  config(['$routeProvider', function($routeProvider) {
  $routeProvider.
      when('/phones', {templateUrl: 'partials/phone-list.html',   controller: PhoneListCtrl}).
      when('/phones/:phoneId', {templateUrl: 'partials/phone-detail.html', controller: PhoneDetailCtrl}).
      otherwise({redirectTo: '/phones'});
}]);
```

Routing cont

Ember

```
var router = Ember.Router.create({
  root: Ember.State.extend({
    index: Ember.State.extend({
      route: '/'
    }),
    posts: Ember.State.extend({
      route: '/posts',

      show: Ember.State.extend({
        route: '/:post_id',
        modelType: 'App.Post'

        setupControllers: function(router, post) {
          var postController = router.get('postController');
          postController.set('content', post);
        }
      })
    })
  })
```

```
      show: Ember.State.extend({
        route: '/:post_id',
        modelType: 'App.Post'

        setupControllers: function(router, post) {
          var postController = router.get('postController');
          postController.set('content', post);
        }
      })
    })
  })
});

// If the user navigates to /, he will start in the
// root.index state.

router.transitionTo('posts.show', post)

// URL => www.myapp.com/posts/1
```

View layer

- Angular - no view "object" perse - it's the model as shown through the HTML template, which gets updated through binding definitions with model
- Backbone - Backbone.View object
  - el and events attributes
- Ember - Ember.View object
  - has template and layout property
  - events can have property matching event name or event manager

Mobile Testing

- Standard Grails unit/integration testing for server components
- Using frameworks such as QUnit or Jasmine for Javascript testing
- Using Geb for functional testing
- Headless or not (or both)

A few Geb notes of interest

Support user gestures through Selenium actions

```
class HomePageSpec extends MobileSpeck {
    def "swipe from home page works as expected"() {
        given:
            to HomePage
            at HomePage
        when:
            interact {
                dragAndDropBy(headline, -400,0)
            }
        then:
            assert text == """Johnny and Jamie had their first
                day of school together today!"""
    }
}
```

Support for running on mobile devices (and in emulators)

```
environments {
    // run as "grails -Dgeb.env=localiphone test-app"
    localiphone {
        baseUrl = "http://10.0.1.5:3001/wd/hub/" //address of your local app instance, on your network
        driver = {
            def driver = new RemoteWebDriver(new URL(baseUrl), DesiredCapabilities.iphone())
            driver
        }
    }

    // run as "grails -Dgeb.env=localipad test-app"
    localipad {
        baseUrl = "http://10.11.12.109:3001/wd/hub/" //address of your local app instance, on your net
        driver = {
            def driver = new RemoteWebDriver(new URL(baseUrl), DesiredCapabilities.ipad())
            driver
        }
    }

    // run as "grails -Dgeb.env=localxoom test-app"
    localxoom {
        baseUrl = "http://10.11.12.105:8080/wd/hub/" //address of your local app instance, on your net
        driver = {
            def driver = new RemoteWebDriver(new URL(baseUrl), DesiredCapabilities.android())
            driver
        }
    }
}
```

```
        baseUrl = "http://10.0.1.5:3001/wd/hub/" //address of your local app instance, on your network
        driver = {
            def driver = new RemoteWebDriver(new URL(baseUrl), DesiredCapabilities.iphone())
            driver
        }
    }

    // run as "grails -Dgeb.env=localipad test-app"
    localipad {
        baseUrl = "http://10.11.12.109:3001/wd/hub/" //address of your local app instance, on your net
        driver = {
            def driver = new RemoteWebDriver(new URL(baseUrl), DesiredCapabilities.ipad())
            driver
        }
    }

    // run as "grails -Dgeb.env=localxoom test-app"
    localxoom {
        baseUrl = "http://10.11.12.105:8080/wd/hub/" //address of your local app instance, on your net
        driver = {
            def driver = new RemoteWebDriver(new URL(baseUrl), DesiredCapabilities.android())
            driver
        }
    }
}
```

Final thoughts

- Worry less about which MVC and templating framework and more about your mastery of concepts that unite them
  - Such as Modules, Facades, and Mediators
- Test your javascript using BDD/TDD and frameworks such as Jasmine or QUnit
- Grails testing, REST support, and resources management are excellent elements to employ in your mobile web application

Additional Resources

- http://addyosmani.com/resources/essentialjsdesignpatterns/book/
- https://github.com/addyosmani/todomvc/
- Handlebars Grails Plugin
- DustJS Grails Plugin
- https://github.com/derickbailey for some great Backbone plugins (ModelBinder, Marionette, etc)

Special Content-related thanks

- Matt Sheehan and Zach Legein for their Handlebars and Dust.js Grails Plugin work
- Zan Thrash for refactoring support of JSON Views
- Addy Osmani for the TodoMVC application and his brilliant takes on Javascript Architecture Patterns

THANK YOU!

and

QUESTIONS?