Playbook to Build Apps with Phonegap and Famo.us

The speed of prototyping, the quality of native.

Nicholas Reed

Welcome!

Available at http://www.practicalguidetomobileapps.com/

My name is Nicholas Reed. I'm a web developer in the Bay Area, and this is the "Playbook" that I use while developing apps for Android and iOS. I'm a proponent of this technique:

> "when you want the **speed** of prototyping, with the **quality** of native."

Instead of writing with Java, Obj-C, or C#, you'll write in Javascript, using the Famo.us framework for your display code using very little HTML. We'll use the Cordova toolchain, and PhoneGap Build, to wrap the code and produce native binaries for iOS and Android. Hopefully Windows soon too.

Through my experience building PhoneGap apps, I've encountered many unknowns/bugs/features, and if you've tried building a PhoneGap app, you likely have as well. From handling the Back button on Android and incorporating it into your routing, to having consistent and beautiful transitions across every page, the scope and challenges of building an end-to-end PhoneGap app can expand quickly. This guide is a series of patterns that can be used to augment existing knowledge, or as a reference resource.

Patterns and examples include:

- Comprehensive routing, including Back button
- Popovers/Modals
- Famo.us gotchas/clarifications
- Mouse/Finger movement and physics-based drag
- Welcome screens
- Wizards (step-by-step pages)
- Data fetching via models and collections
- Authentication
- Push Notifications

Included, and referenced throughout, is a sample app called **Waiting** (it may also be referenced as **Internal App**). Coupled with a node.js server (soon to be released), it works as a complete showcase of many of the concepts referenced, and can be used as a jumping off point for many apps you'd like to build.

Waiting includes some useful features like:

- login and welcome pages
- friends and invite-by-sms from local contacts
- username searching
- dragging and momentum
- messaging
- Push Notifications for Android and iOS

FAQs

How does this hybrid technique with Famo.us compare to other app-building methods?

The short answer: Using Famo.us for your View rendering is, to me, a revolution in accessiblity for myself as a frontend web developer. I can finally, legitimately, fool somebody into thinking I've built a "native" app , based on visuals and feel.

The long answer: It depends on your needs. If you're a team of Obj-C wizards who don't know a lick of Javascript, you'll probably be better served sticking to your guns. But, if you've been building front ends for websites in Javascript for any length of time, this technique can help you quickly create an app that would convince somebody into thinking you've built the app using the native toolchain on the platform (Obj-C, Java, C#), in respect to how it looks and feels through transitions and feedback.

What other libraries are similar to famo.us?

Framer.js by Koen Bok - http://framerjs.com/

Velocity.js by Julian Shapiro - http://julian.com/research/velocity/

How can I help improve this book?

I'd love if you would email me at **nicholas.a.reed@gmail.com**. I'm not sure of the best way to coordinate improvements yet.

Installation Requirements

Please install the following libraries/packages.

Node Version Manager (NVM)

https://github.com/creationix/nvm

Node.js through NVM

Simple as:

```
$ nvm install 0.10
$ nvm use 0.10
```

This should also install the **Node Package Manager (NPM)**.

Cordova CLI

```
$ npm install -g cordova
```

Android or iOS Toolkit (as needed)

You are not *required* to locally download and install the Android/iOS debug toolkits, but sometimes they are helpful! Specifically, it is useful to get Eclipse installed to debug some specific Android errors, or Xcode for iOS.

Server Requirements

If you're building the Waiting/Internal App, it uses:

- Node.JS 0.10
- MongoDB (MongoLab)

(more details forthcoming)

PhoneGap and Cordova

Blog post on phonegap.com

This is the best article for summarizing the history of and differences between PhoneGap and Cordova.

http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/

Building Locally

> Assumes you have the necessary components installed from Chapter 3 - Setup / Requirements.

The Cordova command line toolchain can be used to install, build, and run your app locally.

Build Hello World App

Initial Cordova CLI commands

Use the pattern `cordova create directoryName id.of.app NameOfApp` when creating your new app:

```
$ cordova create hello com.hello.world HelloWorld
$ cd hello
```

Android

```
$ cordova platform add android
$ cordova plugin add org.apache.cordova.device
$ cordova run android -d
```

When you're able to run this command successfully, you'll have a working version of the Hello World app.

If you encounter issues connecting to your attached Android device, I recommend running

```
$ adb devices
```

to make sure your device is visible.

iOS

iOS instructions assume the use of OSX.

```
$ cordova platform add ios
$ cordova plugin add org.apache.cordova.device
$ cordova build ios -d
```

After building successfully, we need to use XCode to finally launch our app.

Open the file `hello/platforms/ios/HelloWorld.xcodeproj` in XCode, choose your target device (either the emulator or a connected device/phone), and click the big **PLAY** icon in the top-left.

Failed?

If you're unable to build or run locally, please skip to the **PhoneGap Build** step (Chapter 4.3) where we'll use the online build service to get valid binaries.

Build Waiting/Internal App

Links

Internal/Waiting App (GitHub Repo): https://github.com/nicholasareed/internal_app

Local Steps

When using the Cordova CLI to build your app locally, you'll notice that only the `your_app_name/www` directory is committed to `git`. So to get a copy of the **Waiting App** that works locally, we need to replace our local `www` directory with the remote Waiting repository.

First step, create a new local project:

```
$ cordova create waiting_app com.waiting.app.dev Waiting
$ cd waiting_app
```

- directory: `waiting`
- app id: `com.waiting.app.dev`
- app name: `Waiting`

Add a platform:

```
$ cordova platform add android
```

Now we'll remove the `www` directory and replace it with a remote repo.

```
// still in waiting_app/
$ rm -rf www
$ git clone git@github.com:nicholasareed/internal_app.git www
```

> Notice the trailing `www` on line directly above me

Let's remove the remote .git connection as well

```
$ cd www
$ git remote rm origin
```

Finally we need to install the native plugins that the **Waiting App** expects. You can find these listed in the `config.xml` file with the prefix `gap:plugin`:

```
$ cordova plugin add org.apache.cordova.device
$ cordova plugin add org.apache.cordova.console
$ cordova plugin add org.apache.cordova.file
$ cordova plugin add org.apache.cordova.file-transfer
$ cordova plugin add org.apache.cordova.camera
$ cordova plugin add org.apache.cordova.inappbrowser
$ cordova plugin add org.apache.cordova.geolocation
$ cordova plugin add org.apache.cordova.globalization
$ cordova plugin add com.phonegap.plugins.pushplugin
$ cordova plugin add com.phonegap.plugins.barcodescanner
$ cordova plugin add com.phonegap.plugin.statusbar
$ cordova plugin add com.verso.cordova.clipboard
```

Neglecting to install **at least** the `device` plugin will result in the app not appearing to work on devices, because it is expecting a `device_ready` event to fire from that plugin.

And now we'll build and run it!

```
$ cordova run android -d
```

It may take a few minutes to build. If it **fails**, see Chapter 4.1 about building the default Hello World app for Android or iOS.

PhoneGap Build

> https://build.phonegap.com/

PhoneGap Build provides an easy way to compile your apps in the cloud and receive a downloadable APK (Android) or IPA (iOS).

Build the Waiting/Internal App for Android

1. Sign up for a Free account on PhoneGap Build
2. Visit your Apps page (https://build.phonegap.com/apps)
3. Click `[New App]` in the top-right
4. Click the `[open-source]` tab
5. Copy/Paste the GitHub repository: https://github.com/nicholasareed/internal_app
6. Click `[Pull from GitHub Repository]`
7. Click `[Build App]`
8. Wait until the page refreshes, and the "Download" button is available (1-5 minutes)
9. Success!

You should now have a working, unsigned .APK download available for Android.

iOS

iOS builds require a provisioning profile and certificate key (see Chapter 10: Production - iOS).

Intro to Famous

What is Famo.us?

> The following is copied from http://famo.us

Famo.us is a free, open source JavaScript framework that helps you create smooth, complex UIs for any screen.

Famo.us is the only JavaScript framework that includes an open source 3D layout engine fully integrated with a 3D physics animation engine that can render to DOM, Canvas, or WebGL.

> Canvas and WebGL rendering are coming soon

Famo.us University

http://famo.us/university

Before continuing, make sure you've gone through all the tutorials Famo.us University. This will give you a beginners understanding of what Famo.us is, and how it works.

GitHub Repository

https://github.com/Famous/core

You can also see all of the reference documentation too at http://famo.us/docs/, but it is much more beneficial to pour over the source.

Guides (read ALL of these!)

http://famo.us/guides/

Read and re-read these guides! The RenderTree guide in particular is key to understanding Famo.us.

> "It's modifiers all the way down" - Anon

When the above quote makes sense, you understand Famo.us.

Contrived Examples

https://github.com/Famous/examples

These examples are useful if you've forgotten how basic Famo.us components works. No widgets though!

Gotchas

What is a layman's description of famo.us? What is famo.us actually doing?

Famo.us tries to take away the things that a browser is bad at (rendering html, simplistically), while taking advantage of browser benefits (webkit CSS 3d transforms are hardware accelerated!). If you use Chrome Dev Tools to inspect the sample app, you'll notice the `<div>`s only go 2-3 levels deep. Famo.us creates a "virtual DOM" and re-uses the divs on the page. This means you can NEVER use jQuery to reliably query the DOM!

Famo.us uses a scene graph (http://en.wikipedia.org/wiki/Scene_graph), basically a game engine that renders to the screen at (a target) 60 frames-per-second. At 60 FPS, it means that every 16 milliseconds, Famo.us will update its virtual DOM by calling many of the `render` methods on famo.us components, and then render the result to the browser.

Keep in mind that famo.us can be used in many different ways. If you're familiar with Angular.js, check out those integrations!

Famo.us maintains a list of common pitfalls/gotchas as well: http://famo.us/guides/pitfalls

Following are the biggest issues I've encountered when writing app-level code.

FastClick issues (double-click of elements)

This seems to manifest itself when writing blocking code, such as for an `alert()`.

```
mySurface.on('click', function(){ alert(1); });
```

will result in *2* alert messages instead of 1. Until a fix is written, you can easily add a `Timer` that waits a moment and allows the native `click` event to be suppressed by Famo.us:

```
mySurface.on('click', function(){
    Timer.setTimeout(function(){
        alert(1);
    }, 200);
});
```

Sizing Issues (true, 0)

When creating some Views, such as a `SequentialLayout`, Famo.us has a rough time ordering `Surfaces` if they have a height/width size of `true` or `0` (in the same direction as the layout: a `size:[100,true]` on `direction:1` causes problems).

A common pattern around this looks like:

```
var mySeq = new SequentialLayout({
    direction: 1 // vertical, "Y"
});
mySeq.Views = [];

var myView = new View();
myView.Surface = new Surface({
    content: 'Content',
    size: [undefined, true]
});
myView.getSize = function(){
    return [undefined, myView.Surface._size ? myView.Surface._size[1] : undefined]
};
myView.add(myView.Surface);

mySeq.Views.push(myView);

mySeq.sequenceFrom(mySeq.Views);
```

This makes it easy to also add StateModifiers to the Surface/View:

Instead of

```
myView.add(myView.Surface);
```

we create a new `StateModifier` and add it to the RenderTree above our Surface.

```
myView.StateMod = new StateModifier();
myView.add(myView.StateMod).add(myView.Surface);
```

Now we can easily modify the position, scale, opacity, etc. of the Surface.

FrontMod (Z-space transforms)

A common problem you'll encounter is one Surface being "on top of" or "in front" of another Surface, causing clicks to fail, or the "behind" Surface to not be visible.

Z-space positioning is affected by 3 things:

- Render order of Surfaces
- Famo.us Transforms
- CSS z-index

If you have overlapping Surfaces, always use Transforms to get it resolved. For example:

```
var SlightlyInFrontMod = new Modifier({
    // move forward in z-space by 0.0001
    transform: Transform.translate(0,0,0.0001)
});
this.someView.add(SlightlyInFrontMod).add(this.someView.Surface);
this.someView.add(this.anotherSurfaceThatWillBeBehind);
```

Avoid using the CSS `z-index` property, it'll just cause problems with your Transforms.

Building the App

Now that you understand how Famo.us works and have seen your first PhoneGap app running, let's look at a larger application structure.

Make sure you have access to the **Waiting/Internal App** source code, so you'll be able to understand how each piece fits into the overall structure.

Structure of the following pages

- Explain the structure of the Internal/Waiting App
- Go over misc components such as `config.xml`, `CSS Styles`, `credentials.json`, etc.
- Dive into the launching point, `main.js`, and explain the order of everything being initialized
- Explain line-by-line many of the core components of initialization, including `routing`, `utilities`, and `localization`.
- Explore Internal Apps method of gathering data from a server, and explain the setup of our Internal App Server
- Go line-by-line over many of Internal App's PageViews, and try out various view patterns such as Wizards, Popovers, and Searching/Autocomplete.
- Discuss debugging using Chrome on your local computer, and Eclipse/XCode when necessary
- Encourage the use of Selenium
- Deploy an app on the Play Store (Android Market), and on TestFlight

Core Libraries and Plugins

Famo.us: Used for our Views, currently using 0.2.

Require.js (http://requirejs.org/)

Backbone.js: A few modifications to the backbone core, mostly centered around Model and Collection handling, including adding the `populated()` promise and `hasFetched` boolean flag. These will be moved to plugins eventually.

Handlebars.js: For HTML templating. Used sparingly, mostly prototyping, and should be removed in favor of native Famo.us layout components. Remember that a Surface is the lowest thing you want to animate; you would never animate a `div` that exists inside a Surface!

jQuery: `jquery-adapter.js` combines all the jquery plugins.

Directory Structure

Ordinary, default structure of a PhoneGap app:

```
/app_name
./hooks
./merges
./platforms
./plugins
./www
```

And in our `www` directory we have our app-specific structure:

```
css
    app.css
    ionicons.css
fonts
    ionicons....
img
locales
res
    icon/
    screen/
splash
src
    famous/
    lib/
    lib2/
    models/
    views/
    config.xml
    credentials.json
    device_ready.js
    history.js
    main.js
    router.js
    text.js
.gitignore
config.xml
icon.png
index.html
splash.png
```

config.xml

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="com.nemesis.app.pub" version="0.0.1.52" xmlns="http://www.w3.org/ns/widgets" xml
    <name>Hello World</name>
    <description>
        My First App
    </description>
    <author email="your_email_here@example.com" href="http://cordova.io">
        Nicholas Reed
    </author>
    <content src="index.html" />

    <access origin="*" />

    <preference name="phonegap-version" value="3.4.0" />

    <!-- Android SDK Version -->
    <preference name="android-minSdkVersion" value="14" />
    <!-- iOS Version -->
    <preference name="deployment-target" value="7.0" />

    <preference name="orientation" value="portrait" />
    <preference name="fullscreen" value="false" />

    <preference name="target-device" value="handset" />
    <preference name="disallowOverscroll" value="true" />
    <preference name="webviewbounce" value="false" />
    <preference name="exit-on-suspend" value="false" />
    <preference name="detect-data-types" value="false" />
    <preference name="StatusBarOverlaysWebview" value="false" />

    <gap:platform name="ios" />
    <gap:platform name="android" />

    <!-- Hide Status Bar iOS -->
    <gap:config-file platform="ios" parent="UIViewControllerBasedStatusBarAppearance" overwr
        <false/>
    </gap:config-file>
    <gap:config-file platform="ios" parent="UIStatusBarHidden" overwrite="true">
        <true/>
    </gap:config-file>

    <gap:plugin name="org.apache.cordova.device" />
    <gap:plugin name="org.apache.cordova.console" />
    <gap:plugin name="org.apache.cordova.file" />
    <gap:plugin name="org.apache.cordova.file-transfer" />
    <gap:plugin name="org.apache.cordova.camera" />
    <gap:plugin name="org.apache.cordova.inappbrowser" />
    <gap:plugin name="org.apache.cordova.geolocation" />
    <gap:plugin name="org.apache.cordova.globalization" />
    <gap:plugin name="com.phonegap.plugins.pushplugin" />
    <gap:plugin name="com.phonegap.plugins.barcodescanner" />
    <gap:plugin name="com.phonegap.plugin.statusbar" />
    <gap:plugin name="com.verso.cordova.clipboard" />

    <gap:plugin name="nl.x-services.plugins.socialsharing" />
    <gap:plugin name="nl.x-services.plugins.toast" version="1.0" />
    <gap:plugin name="de.appplant.cordova.plugin.local-notification" />
    <gap:plugin name="com.adobe.plugins.gaplugin" />


    <feature name="BarcodeScanner">
        <param name="ios-package" value="CDVBarcodeScanner" />
        <param name="android-package" value="com.phonegap.plugins.barcodescanner.BarcodeScan
    </feature>

    <feature name="SocialSharing">
      <param name="ios-package" value="SocialSharing" />
      <param name="android-package" value="nl.xservices.plugins.SocialSharing" />
    </feature>

    <feature name="Toast">
      <param name="ios-package" value="Toast" />
```

```
      <param name="android-package" value="nl.xservices.plugins.Toast" />
    </feature>

    <feature name="Device">
        <param name="android-package" value="org.apache.cordova.device.Device" />
    </feature>

    <feature name="File">
        <param name="android-package" value="org.apache.cordova.file.FileUtils" />
        <param name="ios-package" value="CDVFile" />
    </feature>

    <feature name="FileTransfer">
        <param name="android-package" value="org.apache.cordova.filetransfer.FileTransfer" />
        <param name="ios-package" value="CDVFileTransfer" />
    </feature>

    <feature name="InAppBrowser">
        <param name="android-package" value="org.apache.cordova.inappbrowser.InAppBrowser" />
        <param name="ios-package" value="CDVInAppBrowser" />
    </feature>

    <feature name="PushPlugin" >
        <param name="android-package" value="com.plugin.gcm.PushPlugin"/>
        <param name="ios-package" value="PushPlugin"/>
    </feature>

    <feature name="LocalNotification">
        <param name="android-package" value="de.appplant.cordova.plugin.localnotification.Lo
        <param name="ios-package" value="APPLocalNotification"/>
    </feature>

    <icon src="icon.png" />
    <icon src="icon_ios.png" gap:platform="ios" />

    <gap:splash src="splash.png" />
    <gap:splash src="splash/ios/Default_iphone5.png" gap:platform="ios" width="640" height="

</widget>
```

When building locally, all of the `<gap...` tags are ignored, and the `<feature...` tags are used. On PhoneGap Build, the opposite holds true. The rest of the tags work for both build locations. Local builds use the `root_dir/config.xml` while PhoneGap Build uses one a directory deeper: `root_dir/www/config.xml`. Because it is annoying to update multiple config files and keep local/remote config files separate, we simply use symbolic links (symlinks) to a single `config.xml` file.

```
$: cd ~/Sites/hello
$: [ -f config.xml ] && echo "Found" || echo "Not"
"Found" // just confirming that our config.xml exists
$: cd www
$: ln -s ../config.xml
```

CSS Styles

Because we're still simply using JS/HTML/CSS, we can do all of our styling in CSS!

At the top of our index.html, we include:

```
<link rel="stylesheet" type="text/css" href="src/famous/core/famous.css" />
<link rel="stylesheet" type="text/css" href="css/ionicons.css" />
<link rel="stylesheet" type="text/css" href="css/app.css" />
```

- famous CSS styles
- Ion Icons fonts (great icons)
- our app's CSS

Inline Styles

Famo.us does offer the option of doing "inline" styles like:

```
var s = new Surface({
    content: "hello",
    size: [undefined, undefined],
    properties: {
        fontSize: "14px",
        lineHeight: "18px",
        color: "red"
    }
});
```

Do not get in the habit of doing this! Eventually you will need to copy-paste some code, and it is MUCH easier dealing with CSS styles. You should almost always leave the `properties` out and instead use the `classes` object:

```
var s = new Surface({
    content: "hello",
    size: [undefined, undefined],
    classes: ["my-class-here"]
});

// and then in app.css
.my-class-here {
    font-size: 14px;
    line-height: 18px;
    color: red;
}
```

credentials.json

```
{
    "server_root" : "http://30021fb0.ngrok.com/",
    "server_root" : "https://internalappserver1.herokuapp.com/",

    "GoogleAnalytics" : "UA-0000000-1"
}
```

Many methods exist for switching between development/staging/production servers. I use this setup to quickly comment out (or add an underscore) to switch (`_server_root`). When parsed, last-seen is the "winner" between keys. In the above example, the `...herokuapp.com/` server would be the one used.

index.html

There is a conflict between Cordova, require.js, and Famo.us, so we load everything in this order to ensure it works correctly. When `deviceready` is fired by cordova, we'll load require.js, which in turn loads our `main.js`.

```
<html>
    <head>
        <title>HelloWorld</title>
        <meta name="viewport" content="width=device-width, maximum-scale=1, user-scalable=no"
        <meta name="mobile-web-app-capable" content="yes" />
        <meta name="apple-mobile-web-app-capable" content="yes" />
        <meta name="apple-mobile-web-app-status-bar-style" content="black" />
        <link rel="stylesheet" type="text/css" href="src/famous/core/famous.css" />
        <link rel="stylesheet" type="text/css" href="css/ionicons.css" />
        <link rel="stylesheet" type="text/css" href="css/app.css" />

        <script type="text/javascript" src="cordova.js"></script>

        <script type="text/javascript" src="src/lib/functionPrototypeBind.js"></script>
        <script type="text/javascript" src="src/lib/classList.js"></script>
        <script type="text/javascript" src="src/lib/requestAnimationFrame.js"></script>


        <script type="text/javascript">
            var GLOBAL_onReady = false;

            // track.js customer id
            var _trackJs = {
                customer: '...'
            };

            // Required scripts to add after deviceready
            var require_scripts = [
                {
                    src: "src/lib/require.js",
                    attr: [{
                        "data-main" : "src/main"
                    }]
                }
            ];

            var addScripts = function(scripts){

                // Add scripts
                scripts.forEach(function(script_info){

                    var script = document.createElement( 'script' );
                    script.type = 'text/javascript';

                    if(typeof(script_info) === 'string'){
                        script.src = script_info;
                    } else {
                        script.src = script_info.src;
                        if(script_info.attr){
                            script_info.attr.forEach(function(scriptAttr){
                                Object.keys(scriptAttr).forEach(function(key){
                                    script.setAttribute(key, scriptAttr[key]);
                                });
                            });
                        }
                    }

                    document.body.appendChild( script );

                });

            };

            function onLoad() {

                // browser?
                if (!navigator.userAgent.match(/(iPhone|iPod|iPad|Android|BlackBerry)/)) {
                    // browser
                    addScripts(require_scripts);
```

```
                }

                // wait for deviceready from cordova
                document.addEventListener("deviceready", function(){
                    // alert('deviceready listener fired!');
                    GLOBAL_onReady = true;
                    addScripts(require_scripts);
                    // loadGoogleMapsScript();
                }, false);

            }
        </script>

    </head>
    <body onload="onLoad()" style="background:#666;"></body>
</html>
```

main.js

This is the entry point for your app. It takes care of removing the splash screen, creating the global App variable, checking authentication, figuring out what page to start the user, etc.

I've broken main.js into three sections:

- **require.config**: Setting up paths and naming used elsewhere
- **general**: Startup functions, loading the most basic of content/libraries
- **MainContext**: Adding the different display layers (main, main footer, popovers, etc.)

require.config

require.config

- paths

require.js patterns

- backbone-adapter
- jquery-adapter

Notice the `path` that is set for `utils` pointing to `../src/lib2/utils`. This makes it so that we can simply do

```
var Utils = require('utils');
```

In any of our Views in order to get access to the Utility functions without having to type `../src/lib2/utils` every time. Every keystroke counts!

general

Global App variable

```
// Data store
App = {
    t: null, // for translation
    Flags: {},
    MainContext: null,
    MainController: null,
    Events: new EventHandler(),
    Config: null, // parsed in a few lines, symlinked to src/config.xml
    ConfigImportant: {},
    BackboneModels: _.extend({}, Backbone.Events),
    Router: null,
    Views: {
        MainFooter: null
    },
    Analytics: null,
    DefaultCache: tmpDefaultCache,
    Cache: _.defaults({},tmpDefaultCache),
    Data: {
        User: null,
        Players: null // preloaded
    },
    Defaults: {
        ScrollView: {
            friction: 0.001,
            drag: 0.0001,
            edgeGrip: 0.5,
            edgePeriod: 500, //300,
            edgeDamp: 1,
            speedLimit: 2

            // friction: 0.0001, // default 0.001
            // edgeGrip: 0.05, // default 0.5
            // speedLimit: 2.5 // default 10
        },
        Header: {
            size: 60
        },
        Footer: {
            size: 0
        }
    },
};
```

Parse config.xml

The version number parsed out of `config.xml` is used for analytics and displayed on the Settings page.

```
var ConfigXml = require('text!config.xml');
App.Config = $($.parseXML(ConfigXml));
if(App.Config.find("widget").get(0).attributes.id.value.indexOf('.pub') !== -1){
    App.Prod = true;
    App.ConfigImportant.Version = App.Config.find("widget").get(0).attributes.version.value;
}
```

Analytics

Load Google Analytics (cordova plugin).

```
// Google Analytics Plugin
Utils.Analytics.init();
```

Device Ready init

```
// Run DeviceReady actions
App.DeviceReady = DeviceReady;
App.DeviceReady.init();
```

Check out the device_ready.js article/chapter (below, not this same page).

Localization / Globalization

We initialize our localization efforts by determining the last-used language, or by trying to guess what language the user would like to use based on their phone's preferences (using the Globalization plugin).

```
App.Cache.Language = localStorage.getItem('language_v1');
var LanguageDef = $.Deferred();
if(!App.Cache.Language || App.Cache.Language.length <= 0){
    App.Cache.Language = 'en';
    try {
        navigator.globalization.getLocaleName(
            function (locale) {
                // Set the locale
                // - should validate we support this locale!
                var localeNormalized = Utils.Locale.normalize(locale.value);
                if(localeNormalized !== false){
                    App.Cache.Language = localeNormalized;
                }
                LanguageDef.resolve();
            },
            function () {
                alert('Error getting locale');
            }
        );
    }catch(err){
        // use default
        LanguageDef.resolve();
    }
} else {
    // Have the language already
    LanguageDef.resolve();
}

// Localization
LanguageDef.promise().then(function(){
    $.i18n.init({
        lng: App.Cache.Language, //'ru',
        ns: {namespaces: ['ns.common'], defaultNs: 'ns.common'},
        useLocalStorage: false,
        debug: true
    },function(t){
        // localization initialized
        App.t = t;

        ... code continues here...

    });
```

Initialize Router

```
App.Router = require('router')(App);
```

Hammer.js for swipes

Famo.us should be able to handle this, but for now I'm continuing to use Hammer.js to handle basic "swipe" gestures.

MainContext

stub

MainContext MainController (Lightbox, with defaults) BgSurface for MainController (with zIndex) MainFooter

- frontMod Popover ToastNode FPS calculator

Start backbone.history (should be App.history) Ajax Parameters (todo, fix this, leaks token!) Get localUser from localStorage

- set headers with token (should do it in the model, per-fetch?)
- go to Profiles tab if logged in
- preload Models
- update the user (necessary here, can't we just call App.Data.User.fetch()?)
    - more like testing that they are still valid, and then logging out if not if not logged in:
- unregister from Push
- go to logout/login page

Main

Context

You'll do this ONE time. The "perspective" can be anything from 1 to 1000, but the actual value has no discernable effect on our app. You MUST set it though, otherwise Android will have a rough time displaying some text.

```
        App.MainContext = Engine.createContext();
        App.MainContext.setPerspective(1000);
```

Lightbox (App.MainController)

Let's create the `App.MainController` as a Lightbox, which allows us to easily (and beautifully) switch between arbitrary PageViews.

```
        App.MainController = new Lightbox();
        App.MainController.resetOptions = function(){
            this.setOptions(Lightbox.DEFAULT_OPTIONS);
        };

        // Add Lightbox/RenderController to mainContext
        App.MainContext.add(App.MainController);
```

Main Background

We want a basic `white` background at first. This is also one of the few places we'll use `zIndex` (or `z-index` in CSS) for our front/back positioning.

```
        App.MainContext.add(new Surface({
            size: [undefined, undefined],
            properties: {
                // background: "url(img/mochaGrunge.png) repeat",
                backgroundColor: "white",
                zIndex : -10
            }
        }));
```

Footer

> Eventually this will be moved to Components.CreateStandardFooter (or similar function)

```
// Main Footer
var createMainFooter = function(){
```

Familiar pattern, always creating a View instead of a RenderNode:

```
    App.Views.MainFooter = new View();
```

Create the TabBar, we'll add content in a minute:

```
    App.Views.MainFooter.Tabs = new StandardTabBar();
    var tmpTabs = App.Views.MainFooter.Tabs;
```

Define a Tab (key, content, css classes):

```
    tmpTabs.defineSection('story', {
        content: '<i class="icon ion-ios7-photos"></i><div><span class="ellipsis-all">'+App.t
        onClasses: ['footer-tabbar-default', 'on'],
        offClasses: ['footer-tabbar-default', 'off']
    });
```

`content`: we're including an icon and text. The text is added after translation (`App.t(...)`) and surrounded by a `span` with class `ellipsis-all` (see CSS Styles). `onClasses`/`offClasses`: Array of ALL the classes you want included on that state. Generally I include a default class, plus an `on` and `off` class and my css looks like:

```
.footer-tabbar-default.on {
    border-bottom: 4px solid blue;
}
```

Let's define the rest of the Tabs (automatic sizing/width):

```
    tmpTabs.defineSection('explore', {
        content: '<i class="icon ion-play"></i><div><span class="ellipsis-all">'+App.t('foote
        onClasses: ['footer-tabbar-default', 'on'],
        offClasses: ['footer-tabbar-default', 'off']
    });

    tmpTabs.defineSection('new', {
        content: '<i class="icon ion-ios7-plus-outline big-icon"></i>',
        onClasses: ['footer-tabbar-default', 'footer-tabbar-withbackground', 'on'],
        offClasses: ['footer-tabbar-default', 'footer-tabbar-withbackground', 'off']
    });
    tmpTabs.defineSection('news', {
        content: '<i class="icon ion-android-note"></i><div><span class="ellipsis-all">'+App
        onClasses: ['footer-tabbar-default', 'on'],
        offClasses: ['footer-tabbar-default', 'off']
    });
    tmpTabs.defineSection('profiles', {
        content: '<i class="icon ion-person"></i><div><span class="ellipsis-all">'+App.t('fo
        onClasses: ['footer-tabbar-default', 'on'],
        offClasses: ['footer-tabbar-default', 'off']
    });
```

Now we need to handle tabs being selected:

```
    tmpTabs.on('select', function(result){
        switch(result.id){
            case 'profiles':
                App.history.navigate('dash');
                break;

            case 'ranking':
```

```
                App.history.navigate('ranking/overall/friends/' + App.Data.Players.findMe().g
                break;

            case 'inbox':
                App.history.navigate('inbox');
                break;

            case 'story':
                App.history.navigate('story/all');
                break;

            case 'new':
                App.history.navigate('add/things', {history: false});
                break;

            case 'news':
                App.history.navigate('actions/all');
                break;

            case 'explore':
                App.history.navigate('explore/home');
                break;

            default:
                console.error("None chosen");
                return;
                break;
        }
    });
```

Attach footer to the layout, and add a `frontMod` so that it is now the "most in front" Transform on this RenderNode.

```
    App.Views.MainFooter.frontMod = new StateModifier({
        transform: Transform.inFront
    });
    App.Views.MainFooter.originMod = new StateModifier({
        origin: [0, 1]
    });
    App.Views.MainFooter.positionMod = new StateModifier({
        transform: Transform.translate(0,60,0)
    });
    App.Views.MainFooter.sizeMod = new StateModifier({
        size: [undefined, 60]
    });
```

When we add our tabs to the main Context, it looks confusing but is basically: `View.FrontMod.OriginMod.PositionMod.Tabs`. The order is important; if you switch the OriginMod and PositionMod, for example, you'll get a Footer in a competely different place on-screen!

```
    App.Views.MainFooter.add(App.Views.MainFooter.frontMod).add(App.Views.MainFooter.originM
```

We need to handle events for showing/hiding our footer. On `show` we'll bring the footer up from the bottom by adjusting its `PositionMod` using an easing function. `hide` just reverses our values, but quicker!

```
    App.Views.MainFooter.show = function(transition){
        transition = transition || {
            duration: 750,
            curve: Easing.outExpo
        };
        App.Views.MainFooter.positionMod.setTransform(Transform.translate(0,0,0), transition
    };

    App.Views.MainFooter.hide = function(transition){
        transition = transition || {
            duration: 250,
            curve: Easing.itExpo
        };
        App.Views.MainFooter.positionMod.setTransform(Transform.translate(0,60,0), transitio
    };
```

Finally, we add the footer to our App.MainContext

```
    // Add to maincontext
    App.MainContext.add(App.Views.MainFooter);

};
createMainFooter();
```

Popover

> Eventually this will be moved to Components.CreateStandardPopover (or similar function)

```
var createPopover = function(){
```

Our Popover is simply a Lightbox, with a `FrontMod` to make sure it is always ahead of our MainContext and Footer (but behind our Toast!).

We also get rid of any default Transition/Transforms as our Popover will be defining them.

```
    App.Views.Popover = new Lightbox({
        inTransition: false,
        outTransition: false,
    });

    App.Views.Popover.frontMod = new StateModifier({
        transform: Transform.inFront
    });
    App.MainContext.add(App.Views.Popover.frontMod).add(App.Views.Popover);

};
createPopover();
```

Toast

We're currently using the native system-wide toast, exposed in a manner like:

```
var Utils = require('utils');
Utils.Notifications.Toast('Some message');
```

This causes a 2-second rectangular message box to appear at the bottom of the screen.

Problems

Messages stack up in the history, and every message is shown for 2-seconds. If I did:

```
for(var i=0;i<60;i++){
    Utils.Notification.Toast(i);
}
```

I'd be seeing messages for ~ the next 2 minutes, even though it runs instantly!

Solution

Creating a more customizable Toast that supports options such as:

- vertical stacking
- grouped Toasts (global upload progress meter)
- custom position
- delegate position to current PageView

device_ready.js

Waiting for .onReady to be...ready

> the `onReady` event is emitted by Cordova when the webview has loaded all the native plugins

At the start we're creating some promises that will get resolved in `init`.

```
test: "hello",
readyDeferred: readyDeferred,
ready: readyDeferred.promise(),
init: function(){
    var that = this;
```

Are we running on a phone/emulator with access to native functions? (we assign this `App.Data.usePg`)

```
    App.Data.usePg = false;
    if (navigator.userAgent.match(/(iPhone|iPod|iPad|Android|BlackBerry)/)) {
        // Yes, we're running on a phone/emulator
        App.Data.usePg = true;
    }

    // Trigger .onReady
```

If we are in the browser (in Development, versus Production/Device), we'll manually call `onReady` because the native plugin never runs (because we are not on a phone)

```
    if(!App.Data.usePg){
        // Yes, we're running in a browser
        that.onReady();
    }
```

The following lines are to make sure the `onReady` promise gets resolved. Just in case the "deviceready" listener on the document wasn't already on there, we add it. If it has already resolved (from the previous addition, in index.html) then `GLOBAL_onReady` would have been set previously.

```
    if(GLOBAL_onReady === true){
        that.onReady();
    } else {
        document.addEventListener("deviceready", function(){
            that.onReady();
        }, false);
    }
```

.onReady function

In the .onReady function, we enable many different functions, as long as `App.Data.usePg == true`.

error tracking via track.js

Notice that we're only enabling track.js on devices, in production. This reduces clutter in your error logs, and keeps your debugging local.

```
        if(App.Data.usePg && App.Prod){

            // lazy-load track.js
            var script = document.createElement( 'script' );
            script.type = 'text/javascript';
            script.src = 'src/lib2/track.js';
            // $(script).attr('data-customer','2138571d0e004d109396748e01e291a0');
            $("body").append( script );

            // Need to init it first
            console.error();

            //override the error function (the immediate call function pattern is used for da
            console.error = (function () {
              //save a reference to the original error function.
              var originalConsole = console.error;
              //this is the function that will be used instead of the error function
              function myError (stackTrace) {
                // alert( 'Error is called. ' );
                try {
                    trackJs.track(stackTrace);
                }catch(err){
                    console.log('trackJs.track non-existant for now');
                }

                //the arguments array contains the arguments that was used when console.erro
                originalConsole.apply( this, arguments );
              }
              //return the function which will be assigned to console.error
              return myError;
            })();

            // // Testing in production
            // window.setTimeout(function(){
            //     // Throw a failure
            //     throw WeAreInProduction
            // },3000);

        }
```

Pausing

```
        // Pausing (exiting)
        document.addEventListener("pause", function(){
            // Mark as Paused
            // - this prevents Push Notifications from activating all at once when Resuming

            App.Data.paused = true;
            App.Data.was_paused = true;

        }, false);
```

Resuming

```
        // Resume
        // - coming back to application
        document.addEventListener("resume", function(){
            // Gather existing Push Notifications and see if we
            // should summarize them, or show individually (confirm, etc.)

            App.Events.trigger('resume');

            App.Data.paused = false;
            App.Data.was_paused = true;
```

```
            // Run 1 second after returning
            // - collecting all the Push Notifications into a queue
            // - enough time for all Push events to be realized
            setTimeout(function(){

                App.Data.paused = false;
                App.Data.was_paused = false;

            },1000);

        }, false);
```

Menu button

```
        // Init MENU button on Android (not always there?)
        document.addEventListener("menubutton", function(){
            App.history.navigate('settings');
        }, false);

        // Init BACK button on Android
        // - disable default first
        document.addEventListener("backbutton", function(killa){
            App.Events.emit('backbutton');
            killa.stopPropagation();
            killa.preventDefault();
            return false;
        }, true);
```

Online / Offline state

```
        // // Online/Offline state

        // //Create the View
        // // - render too
        // App.Data.GlobalViews.OnlineStatus = new App.Views.OnlineStatus();
        // App.Data.GlobalViews.OnlineStatus.render();

        // // Online
        // // - remove "not online"
        // document.addEventListener("online", function(){
        //  // Am now online
        //  // - emit an event?
        //  App.Data.GlobalViews.OnlineStatus.trigger('online');

        // }, false);
        // document.addEventListener("offline", function(){
        //  // Am now online
        //  // - emit an event?
        //  App.Data.GlobalViews.OnlineStatus.trigger('offline');

        // }, false);
```

Back button

In-Browser (development) capturing of the DELETE/Backspace button

```
        // Back button capturing in Browser
        if(!App.Data.usePg){
            $(document).keydown(function (e) {
                var preventKeyPress;
                if (e.keyCode == 8) {
                    var d = e.srcElement || e.target;
                    switch (d.tagName.toUpperCase()) {
                        case 'TEXTAREA':
                            preventKeyPress = d.readOnly || d.disabled;
                            break;
                        case 'INPUT':
                            preventKeyPress = d.readOnly || d.disabled ||
                                (d.attributes["type"] && $.inArray(d.attributes["type"].valu
                            break;
                        case 'DIV':
                            preventKeyPress = d.readOnly || d.disabled || !(d.attributes["co
                            break;
                        default:
                            preventKeyPress = true;
                            break;
                    }
                }
                else
                    preventKeyPress = false;

                if (preventKeyPress){
                    e.preventDefault();
                    App.Events.emit('backbutton');
                }
            });
        }
```

On Device Back button (Android)

```
        // Init BACK button on Android
        // - disable default first
        document.addEventListener("backbutton", function(killa){
            App.Events.emit('backbutton');
            killa.stopPropagation();
            killa.preventDefault();
            return false;
        }, true);
```

GPS Updates

```
    // GPS updates
    runGpsUpdate: function(){
        // We only want one running at a time
        if(App.Cache.running_gps_update === true){
            return;
        }

        App.Cache.running_gps_update = true;

        var runGpsUpdateFunc = function(){
            navigator.geolocation.getCurrentPosition(function(position){
                console.log('coords');
                console.log(position.coords);
                App.Events.emit('updated_user_current_location');
                App.Cache.geolocation_coords = position.coords;
                // alert(position.coords);
                // alert(position.coords.latitude);
                // $.ajax({
                //     url: Credentials.server_root + 'user_coords',
                //     cache: false,
                //     method: 'POST',
                //     success: function(){
                //         console.log('Made call OK');
```

```
                //     }
                // });

            }, function(err){
                console.log('GPS failure');
                console.log(err);
            });

            window.setTimeout(runGpsUpdateFunc, 1000 * 60 * 5); // 5 minutes
        };

        // runGpsUpdateFunc(); // uncomment for coordinate uploads

    },
```

Push Notifications

Initializing Push Notifications

```
    initPush: function(){
        console.info('Registering for PushNotification');

        // Disable Push in debug mode
        if(App.Prod != true){
            console.error('Development mode');
            return;
        }
        try {
            App.Data.pushNotification = window.plugins.pushNotification;
            if (device.platform.toLowerCase() == 'android') {

                App.Data.pushNotification.register(function(result){
                    console.log('Push Setup OK');

                }, function(err){
                    alert('failed Push Notifications');
                    console.error(err);
                },
                {
                    "senderID": "your_sender_id_here",
                    "ecb": "onNotificationGCM"
                });
            } else if (device.platform.toLowerCase() == 'ios') {

                App.Data.pushNotification.register(function(token){
                    console.log('ios token');
                    console.log(token);

                    // Write the key
                    // - see if the user is logged in
                    var i = 0;
                    var pushRegInterval = function(){
                        window.setTimeout(function(){
                            // See if logged in
                            if(App.Data.User.get('_id')){
                                App.Data.User.set({ios: [{reg_id: token, last: new Date()}]}
                                App.Data.User.save(); // update the user
                            } else {
                                // Try running again
                                // App.Utils.Notification.debug.temp('NLI - try again' + i);
                                console.log('Not logged in - try android registration update
                                console.log(App.Data.User.get('_id'));
                                i++;
                                pushRegInterval();
                            }
                        },3000);
                    };
                    pushRegInterval();


                },function(err){
                    console.log('Failed ios Push notifications');
                    console.log(err);
                },   {"badge":"true","sound":"true","alert":"true","ecb":"onNotificationAPN"}
            }
        }
        catch(err) {
            // txt="There was an error on this page.\n\n";
            // txt+="Error description: " + err.message + "\n\n";
            // alert(txt);
            // alert('Push Error2');
            if(App.Data.usePg){
                console.error('Push Error 2');
            }

            // console.log(err);
            // App.Utils.Notification.debug.temporary('Push Error');
        }
    },
```

Handling incoming Push Notifications

```
function onNotificationAPN(event) {
    var pushNotification = window.plugins.pushNotification;
    console.log("Received a notification! " + event.alert);
    console.log("event sound " + event.sound);
    console.log("event badge " + event.badge);
    console.log("event " + event);
    if (event.alert) {
        navigator.notification.alert(event.alert);
    }
    if (event.badge) {
        console.log("Set badge on  " + pushNotification);
        pushNotification.setApplicationIconBadgeNumber(function(){
            console.log('succeeded at something in pushNotification for iOS');
        }, event.badge);
    }
    if (event.sound) {
        var snd = new Media(event.sound);
        snd.play();
    }
};

// GCM = Google Cloud Messag[something]
function onNotificationGCM(e){
    // Received a notification from GCM
    // - multiple types of notifications

    // App.Utils.Notification.debug.temp('New Notification: ' + e.event);
    // alert('onNotificationGCM');
    console.log('onNotificationGCM');

    switch( e.event ){
        case 'registered':
            // alert('registered');
            // Registered with GCM
            if ( e.regid.length > 0 ) {
                // Your GCM push server needs to know the regID before it can push to this d
                // here is where you might want to send it the regID for later use.
                // alert('registration id: ' + e.regid);
                // App.Utils.Notification.debug.temp('Reg ID:' + e.regid.substr(0,25) + '...
                console.log('Android registration ID for device');
                console.log(e.regid);

                // // Got the registration ID
                // // - we're assuming this happens before we've done alot of other stuff
                // App.Credentials.android_reg_id = e.regid;

                // Write the key
                // - see if the user is logged in
                var i = 0;
                var pushRegInterval = function(){
                    window.setTimeout(function(){
                        // See if logged in
                        if(App.Data.User.get('_id')){
                            // Sweet, logged in, update user's android_reg_id
                            // alert('saving user!'); // ...
                            // alert(App.Data.User.get('_id'));
                            // alert(e.regid);
                            App.Data.User.set({android: [{reg_id: e.regid, last: new Date()}]
                            App.Data.User.save(); // update the user

                            // App.Plugins.Minimail.updateAndroidPushRegId(App.Credentials.a
                        } else {
                            // Try running again
                            // App.Utils.Notification.debug.temp('NLI - try again' + i);
                            console.log('Not logged in - try android registration update aga
                            console.log(App.Data.User.get('_id'));
                            i++;
                            pushRegInterval();
                        }
                    },3000);
                };
                pushRegInterval();
```

```
            }
            break;

        case 'message':
            // if this flag is set, this notification happened while we were in the foregrou
            // you might want to play a sound to get the user's attention, throw up a dialog

            Utils.process_push_notification_message(e.payload.payload);

            break;

        case 'error':
            alert('GCM error');
            alert(e.msg);
            break;

        default:
            alert('An unknown GCM event has occurred');
            break;
    }
};
```

Display PageViews (router.js)

Our `router.js` is where you'll list all of the Pages your app will use, as well as the way those Pages will be loaded, and how to transition between them.

history.js

First we load our `history.js` file.

```
    App.history = AppHistory(App);
```

> This is where you segue into reading about history.js!

Routes, Views, and Stored Transitions

Read the subtopics.

routes, ViewToView, StoredTransitions

Set up our Routes

We extend Backbone.router to create `DefaultRouter`, and specify our routes. A few routes are explained more in-depth.

"" or "nowhere"

Sometimes your app will end up at the "" route, though you should try and prevent it. Sometimes you'll have a wayward `App.history.eraseUntilTag` that clears history for users. Catch that bug! But, fallback to just semi-restarting the app, in case you do end up here.

```
    '' : function(){
        if(App.history.data.length == 0){
            window.location = window.location.href.split('#')[0];
        }
    },
```

logout

We punt to th Utils.logout function, after verifying the phone is ready (to prevent some nasty redirect loop bugs).

```
    'logout' : function(){

        // Unregister from Push Notifications
        // - do this before exiting
        App.DeviceReady.ready.then(function(){
            Utils.logout();
        });

    },
```

welcome screens

Notice that the `welcome/username` route comes before `welcome`. Routes match only the first matched key, so the order matters!

```
    'welcome/username' : function(){
        defaultRoute('WelcomeUsername', 'User/WelcomeUsername', arguments, {cache: false});
    },

    'welcome' : function(){
        defaultRoute('Welcome', 'User/Welcome', arguments, {cache: false});
    },
```

We'll use the `defaultRoute` many, many times in our routes.

DefaultRoute

Go read about `defaultRoute` (one or two chapters below).

More, common routes:

```
    'modal/helppopover' : function(){
        App.Flags.InPopover = true;
        App.history.navigate('random', {history: false});
        defaultRoute('HelpPopover', 'Misc/HelpPopover', arguments, {cache: false, popover: t
    },

    'modal/list' : function(){
        App.Flags.InPopover = true;
        App.history.navigate('random', {history: false});
        defaultRoute('Popover', 'Misc/Popover', arguments, {cache: false, popover: true});
    },

    'login' : function(){
        // eh, I should be able to cache this route before login, then destroy after login
        defaultRoute('Login', 'Misc/Login', arguments, {cache: false});
    },
```

```
    'signup' : function(){
        defaultRoute('Signup', 'Misc/Signup', arguments, {cache: false});
    },

    'forgot' : function(){
        defaultRoute('Forgot', 'Misc/Forgot', arguments, {cache: false});
    },

    'settings': function(){
        defaultRoute('Settings', 'Misc/Settings', arguments);
    },
    'perks': function(){
        defaultRoute('Perks', 'Misc/Perks', arguments);
    },
    'feedback(/:indicator)': function(){
        defaultRoute('Feedback', 'Misc/Feedback', arguments);
    },

    'add/things' : function(){
        App.Views.MainFooter.route_show = true;
        App.Views.MainFooter.Tabs.select('new', false);
        defaultRoute('AddThings', 'Misc/AddThings', arguments, {cache: false});
    },

    'inbox' : function(){
        defaultRoute('Inbox', 'Message/Inbox', arguments);
    },

    'dash(/:id)' : function(){
        App.history.modifyLast({
            tag: 'Dash'
        });
        App.Views.MainFooter.route_show = true;
        App.Views.MainFooter.Tabs.select('profiles', false);
        defaultRoute('Dash', 'Player/Player', arguments); // used to be Player/Dash
    },


    'actions/all' : function(){
        App.Views.MainFooter.route_show = true;
        App.Views.MainFooter.Tabs.select('news', false);
        defaultRoute('Action', 'Action/Actions', arguments);
    },

    'explore/home' : function(){
        App.Views.MainFooter.route_show = true;
        App.Views.MainFooter.Tabs.select('explore', false);
        defaultRoute('Explore', 'Explore/Explore', arguments);
    },

    'players/search' : function(){
        defaultRoute('PlayerSearch', 'Player/PlayerSearch', arguments, { cache: true });
    },

    'player/add/nolink' : function(){
        defaultRoute('PlayerAddNoLink', 'Player/PlayerAdd', arguments, {cache: false});
    },
    'player/add' : function(){
        defaultRoute('PlayerAdd', 'Player/PlayerAddLink', arguments, {cache: false});
    },
    'player/edit/:id' : function(){
        defaultRoute('PlayerEdit', 'Player/PlayerEdit', arguments, {cache: false});
    },
    'player/relationship_code/:id' : function(){
        defaultRoute('Player', 'Player/PlayerRelationshipCode', arguments);
    },
    'player/:id' : function(){
        App.Views.MainFooter.route_show = true;
        App.Views.MainFooter.Tabs.select('profiles', false);
        defaultRoute('Player', 'Player/Player', arguments);
    },


    'profile/edit' : function(){
        defaultRoute('ProfileEdit', 'User/ProfileEdit', arguments, {cache: false});
    },
```

```
    'playerselect': function(){
        defaultRoute('PlayerSelect', 'Misc/PlayerSelect', arguments, {cache: false});
    }
```

View To Views

In `defaultRoute` you'll notice the ViewToView variable being accessed. This is for setting custom default transitions between PageViews (such as `slideLeft`). It can be especially useful when prototyping where you want to move a whole screen left/right or spring in/out.

There are three ways of specifying Views:

- * (matches any/all)
- empty space
- ViewName

When matching your specified ViewToViews, multiple may end up matching, and the transition to use is specified by the order:

- **most specific / highest priority**: ViewName-ViewName
- **mid**: X -> *
- **lowest priority**: * -> X

Here are some examples:

```
    // Handle special transitions if the View->View conditions match
    var ViewToView = {

        // Potentially request data from the PageView here, or tell it how to render in/out
        // - we might simply emit events like "Hey, you're about to get shown" and "here is

        '* -> Login' : function(){
            return StoredTransitions.OpacityIn;
        },

        'Login -> *' : function(){
            return StoredTransitions.SlideDown;
        },

        'Login -> Signup' : function(){
            return StoredTransitions.SlideLeft;
        },

        'Signup -> Login' : function(){
            return StoredTransitions.SlideRight;
        },


        'Login -> Forgot' : function(){
            return StoredTransitions.SlideLeft;
        },

        'Forgot -> Login' : function(){
            return StoredTransitions.SlideRight;
        }

    };
```

routes

- defaultRoute
- modifyLast (Dash, etc) for tags
- MainFooter tabs route showing (without triggering)
- logout is the only different one ViewToView defaults

StoredTransitions

Here we defined the options available to our ViewToViews, and how quickly they'll move.

```
    var StoredTransitions = {

        Identity: {
            inOpacity: 1,
```

```
            outOpacity: 1,
            inTransform: Transform.identity,
            outTransform: Transform.identity,
            inTransition: { duration: 2400, curve: Easing.easeIn },
            outTransition: { duration: 2400, curve: Easing.easeIn },
        },

        OpacityIn: {
            inOpacity: 0,
            outOpacity: 0,
            inTransform: Transform.identity,
            outTransform: Transform.identity,
            inTransition: { duration: 750, curve: Easing.easeIn },
            outTransition: { duration: 750, curve: Easing.easeOut }
        },

        HideOutgoingSpringIn: {
            inOpacity: 0,
            outOpacity: 0,
            inTransform: Transform.scale(0,-0.1, 0), //Transform.translate(window.innerWidth
            outTransform: Transform.translate(0, 0, 1),
            inTransition: { method: 'spring', period: 500, dampingRatio: 0.6 },
            outTransition: { duration: 300, curve: Easing.easeOut }
        },

        SlideDown: {
            inOpacity: 1,
            outOpacity: 1,
            inTransform: Transform.identity, //Transform.translate(0,window.innerHeight * -1
            outTransform: Transform.translate(0,window.innerHeight,0),
            inTransition: { duration: 750},
            outTransition: { duration: 750},
            overlap: true
        },

        SlideUp: {
            inOpacity: 1,
            outOpacity: 1,
            inTransform: Transform.translate(0,window.innerHeight,0),
            outTransform: Transform.identity, //Transform.translate(0,window.innerHeight * -1
            inTransition: { duration: 750},
            outTransition: { duration: 750},
            overlap: true
        },

        SlideLeft: {
            inOpacity: 1,
            outOpacity: 1,
            inTransform: Transform.translate(window.innerWidth,0,0),
            outTransform: Transform.translate(window.innerWidth * -1,0,0),
            inTransition: { duration: 500, curve: Easing.easeIn },
            outTransition: { duration: 500, curve: Easing.easeIn },
            overlap: true
        },

        SlideRight: {
            inOpacity: 1,
            outOpacity: 1,
            inTransform: Transform.translate(window.innerWidth * -1,0,0),
            outTransform: Transform.translate(window.innerWidth,0,0),
            inTransition: { duration: 500, curve: Easing.easeIn },
            outTransition: { duration: 500, curve: Easing.easeIn },
            overlap: true
        }
    };
```

history.js

Note that we're continuing to use Backbone's route handling via hash fragments, but simply not managing history using the browser's native implementation. We do this because we wanted to add a few useful history features that the browser makes difficult.

`navigate`

This is the meat and potatoes of the navigation. Other functions like `.back` end up calling this function.

In the app, you'll use `App.history.navigate(...);`.

```
    historyObj.navigate = function(path, opts, back, reloadCurrent){

        console.log('historyObj.navigate: path:', path, arguments);

        opts = opts || {};

        opts = _.defaults(opts, {
            history: true, // add to history
            group: null,
            tag: null
        });
```

First we'll see if this was a "going back" type of navigation. If it is, we pop out the last entry in our history, and `navigate` to it using the same arguments as it was original called. We also set a `yes, I'm going back` variable that we'll pick up and reset in a moment.

```
        if(back === true){
            // Get the last entry, see if we can go back to it

            // Remove the "current entry"
            // - reload the last one (after pop'ing it out too)
            var currentArgs = historyObj.data.pop();
            console.log(currentArgs);

            var lastArgs = historyObj.data.pop();
            console.log('lastArgs', lastArgs);
```

Sometimes, things break or you do an `eraseUntilTag` for a tag that doesn't exist. In that instance, we want to make sure the user isn't left in limbo, so we just head to our "favorite" route.

```
            if(lastArgs === undefined || !lastArgs || lastArgs.length < 1 || lastArgs[0] ==
                // No last arguments exist
                Utils.Notification.Toast('Exiting App');
                console.error('exiting app');
                historyObj.navigate('dash');
                return;
            }


            historyObj.isGoingBack = true;
            console.log('isGoingBack==true');
            historyObj.navigate.apply(HistoryContext, lastArgs);

            return;
        }
```

Sometimes you'll need (especially in the case of popovers, where we set `{history: false}`) to simply reload the current page. This works just like `back` but doesn't erase the current entry, it just `navigate`s to the existing page.

```
        if(reloadCurrent === true){
            // Reload the currently-in-place history event
            // - should be used when you want to go "back" from a history:false page

            // pop it out from the array, it'll get popped back in
            var lastArgs = historyObj.data.pop();
            console.log('lastArgs', lastArgs);

            if(lastArgs === undefined || !lastArgs || lastArgs.length < 1 || lastArgs[0] ==
                // No last arguments exist
                Utils.Notification.Toast('Exiting App');
                console.error('Exiting app, reloadCurrent failed');
```

```
                historyObj.navigate('dash');
                return;
            }

            historyObj.isGoingBack = true; // yes, keep as "is going back" from whatever is
            historyObj.navigate.apply(HistoryContext, lastArgs);

            return;
        }
```

Some routes (like popovers or wizard-starters) don't need or want to have a place in the history stack, so by passing `{history:false}` they can avoid being added.

```
        if(opts.history == true){
            historyObj.data.push([
                path, opts, back
            ]);
        }
```

We're done modifying/redirecting our route request, so now we use Backbone to trigger a hash change event.

```
        Backbone.history.navigate(path, {trigger: true, replace: true});


    };
```

`back`

Just a wrapper for `navigate`.

```
historyObj.back = function(opts){
    historyObj.navigate(null, opts, true);
};
```

`reloadCurrent`

Just a wrapper for `navigate`.

```
historyObj.reloadCurrent = function(opts){
    historyObj.navigate(null, opts, false, true);
};
```

`modifyLast`

Used for adding tags to the last history entry. Use like `App.history.modifyLast({tag: 'PostStart'});`.

In `modifyLast` you'll notice that `tag`s are stored in an array, so you can have multiple tags on one history item.

```
    historyObj.modifyLast = function(opts){
        var args = historyObj.data[historyObj.data.length - 1];
        if(!args){ args = []; }
        if(args.length < 1){
            // no args
        } else if(args.length < 2){
            // no options exist anyways
            args[1] = opts;
            historyObj.data[historyObj.data.length - 1] = args;
        } else {
            console.info('modifyLast2', opts.tag);
            if(opts.tag){
                // add tag to array of tags
                var currentTag = historyObj.data[historyObj.data.length - 1][1].tag;
                if(currentTag == undefined || currentTag == null || currentTag == false){
                    currentTag = [];
                }
                if(typeof currentTag == "string"){
                    currentTag = [currentTag];
                }
                // add to array
                currentTag.push(opts.tag);
```

```
                historyObj.data[historyObj.data.length - 1][1].tag = currentTag;

                delete opts.tag
            }

            // altering anything but the 'tag'
            historyObj.data[historyObj.data.length - 1][1] = _.defaults(historyObj.data[hist

        }

    };
```

backTo

We'll erase all the history up until a `tag` matches, and then visit that route (after removing it from the history, but then it immediately gets re-added when navigated to).

```
    historyObj.backTo = function(tag, opts){
        historyObj.eraseUntilTag(tag, false);

        var lastArgs = historyObj.data.pop(); // get last
        historyObj.navigate.apply(RouterContext, lastArgs);
    };
```

eraseUntilTag

Erase all the history up to (and if `eraseLast===true`) and including a specified tag. This is especially useful when existing a Wizard and you want to remove the entire process from the history.

Real world example: When adding a Post with an image, or other piece of content in your app, the user flow could be:

`dash` > `post/add` > `post/add/image` > `post/add/confirmAndSave` > [done] > `post/view/:id`

If the user presses "Back" from the `post/view/:id` route, they'd expect to back to `dash`, not `post/add/confirmAndSave`, correct? You can easily do this by calling `post/add` like

```
App.history.modifyLast({
    tag: 'PostStart'
});
App.history.navigate('post/add');
```

and when you complete the Post, simply erase using

```
App.history.eraseUntilTag('PostStart');
```

> Notice that we didn't pass `true` also, otherwise it would erase the history entry for whatever PageView launched the Post starting

```
    historyObj.eraseUntilTag = function(tag, eraseLast){
        // Erase entries up until the last tag
        // - and including the last tag! (if included, by default, ERASE)
        eraseLast = eraseLast === undefined ? true : (eraseLast ? true : false);

        historyObj.data.reverse();

        var continueErasing = true;
        historyObj.data = _.filter(historyObj.data, function(args){
            // check options for tag that matches
            if(continueErasing !== true){
                return true;
            }
            if(args.length < 2){
                return false;
            }
            if(!args[1].tag){
                return false;
            }
            if(typeof args[1].tag == "string" && args[1].tag != tag){
                return false
```

```
            }
            if(typeof args[1].tag === typeof [] && args[1].tag.indexOf(tag) === -1){
                return false;
            }
            continueErasing = false;
            if(eraseLast === true){
                return false;
            } else {
                return true;
            }

        });

        historyObj.data.reverse();
    };
```

defaultRoute

This is the logistics of loading PageViews, determining transitions, and displaying PageViews.

It ends up switching PageViews using the previously-created `App.MainController`, `App.MainFooter`, and `App.Views.Popover`.

```
var defaultRoute = function(viewName, viewPath, args, options){

    options = options ? options : {};
    if(args === undefined){
        args = viewPath;
        viewPath = viewName;
    }

    console.log('viewPath:', viewPath);
```

Using our `Utils.Analytics.trackRoute` function we'll immediately report the loaded page's route to our Google Analytics plugin.

```
    Utils.Analytics.trackRoute(window.location.hash);
```

We require each PageView at runtime because it is easier than listing every PageView we could potentially need.

```
    require(['views/' + viewPath], function(LoadedView){
```

You'll see `delayShowing` a few times. The logic is that we should give uncached PageViews a moment to "get ready" before displaying.

```
        var delayShowing = 0;
```

Get our PageView from the cache, if it is cached. If it isn't cached, then we go ahead and load it up, then cache if told to.

```
        var PageView = App.Router.Cache.get();

        var cachePath = window.location.hash;

        if(PageView === false || options.cache === false){
            // create it!
            // - first time creating it
            PageView = new LoadedView({
                args: args,
                App: App
            });

            // Only a pass-through?
            if(PageView.doNotShow){
                return;
            }

            // Cache it
            App.Router.Cache.set(PageView, cachePath);

            delayShowing = 100;
        } else {
            // Already cached, use the cached version
        }

        // Cache pageview
        App.Views.currentPageView = PageView;
```

If the route specified it was a popover, then we use the `App.Views.Popover` Lightbox to display the PageView.

```
        if(options.popover === true){
            PageView.inOutTransitionPopover('showing');
            App.Views.Popover.show(PageView);
            return;
        }
```

If we're not a popover, then we know were going to be using our `App.MainController`.

Reset the Lightbox's transition back to its defaults, and then determine what transition we'll end up using. We reset to defaults, otherwise an updated `curve` property could be propogated across **every** `App.MainController` transition.

```
        App.MainController.resetOptions();
        var transitionOptions = {};

        // Setting using default options
        console.log('Animating using Default SlideLeft');
        transitionOptions = StoredTransitions.SlideLeft;

        // See if we're going back a page
        // - use a Default "back" animation SlideRight
        var goingBack = false;
        if(App.history.isGoingBack){
            goingBack = true;
            App.history.isGoingBack = false; // reset

            console.log('Animating using BACK');
            transitionOptions = StoredTransitions.SlideRight;
        }
```

ViewToView checking (see ViewToView in the above section) is used to find the most-specific transition to use.

```
        App.Cache.LastViewName = App.Cache.LastViewName === undefined ? "" : App.Cache.LastV
        var exactMatchView = App.Cache.LastViewName + ' -> ' + viewName, // highest priority
            toAnyView = App.Cache.LastViewName + ' -> *',
            fromAnyView = '* -> ' + viewName; // lowest priority

        // Extract out multiple key values
        // - todo, would allow: "* -> Login, Home -> Login" for the occurrences
        // - could also add an "!important" flag?

        // Check broastest->specific ViewToView transitions
        if(ViewToView[exactMatchView]){
            // Most specific match
            console.log('ViewToView Match: Specific:', exactMatchView);
            transitionOptions = ViewToView[exactMatchView]();
        } else if(ViewToView[toAnyView]){
            // mid
            console.log('ViewToView Match: ToAny:', toAnyView);
            transitionOptions = ViewToView[toAnyView]();
        } else if(ViewToView[fromAnyView]){
            // low
            console.log('ViewToView Match: FromAny:', fromAnyView);
            transitionOptions = ViewToView[fromAnyView]();
        }


        // Lastly, send the selected "options" to the incoming and outgoing views
        // - each View should receive

        // takes in the direction it is going (in/out) and returns an options to set (which v
        // - views are assuming that we're doing a "show" immediately following this, no pro
        // - also serves as a trigger for the view that they are doing something
```

Tell the "hiding" PageView first

```
        if(App.MainController.lastView && App.MainController.lastView.inOutTransition){
            transitionOptions = App.MainController.lastView.inOutTransition('hiding', viewNa
        }
```

Tell "showing" PageView last

```
        if(PageView.inOutTransition){
            transitionOptions = PageView.inOutTransition('showing', App.Cache.LastViewName,
        }
```

Set `transitionOptions` for the `App.MainController` Lightbox.

```
        App.MainController.setOptions(transitionOptions);
```

Update our lastView with the now-current view.

```
        App.MainController.lastView = PageView;
```

Switch displaying the PageViews

Notice the `delayShowing`, as well as

```
        Timer.setTimeout(function(){
            App.MainController.show(PageView);
        }, delayShowing);


        // Update LastViewName
        App.Cache.LastViewName = '' + viewName;
```

Finally, show/hide the footer depending on the latest value for `App.Views.MainFooter.route_show`. We `.hide()` it by default.

```
        if(App.Views.MainFooter){
            if(App.Views.MainFooter.route_show === true){
                App.Views.MainFooter.show();
            } else {
                App.Views.MainFooter.hide();
            }
            App.Views.MainFooter.route_show = false;
        }

    });

};
```

cache

We use the `window.location.hash` value to cache each view. The PageView is only cached in memory, under `App.Cache.RoutesByHash`.

```
Cache : {
    get: function(options){
        var hash = window.location.hash;

        if(!App.Cache.RoutesByHash){
            App.Cache.RoutesByHash = {};
        }
        options = options || {};

        // Cached View?
        if(App.Cache.RoutesByHash[hash] != undefined){
            return App.Cache.RoutesByHash[hash];
        }

        return false;
    },
    set: function(view, hash){// Returns a cached view for this route
        hash = hash || window.location.hash;
        App.Cache.RoutesByHash[hash] = view;
        view.isCachedView = true;
        return view;
    }
}
```

utils.js

I wanted a more central place to handle Utility functions that I was repeatedly calling. `utils.js` serves that purpose.

```
var Utils = {
```

CheckFlag / PostFlag

These functions are great for triggering one-time actions for a user, such as a Help popover when then visit a page for the first time, or when you've added a new feature to an app.

Requires the necessary DB routes.

```
    CheckFlag: function(flag){
        var def = $.Deferred()
        App.Data.User.populated().then(function(){
            if(App.Data.User.get('flags.' + flag)){
                def.reject();
            } else {
                def.resolve(true);
            }
        });
        return def.promise();
    },
    PostFlag: function(flag, value){
        var tmpData = {
            flags: {}
        };
        tmpData.flags[flag] = value;
        App.Data.User.set('flags.' + flag, value);
        return $.ajax({
            url: Credentials.server_root + 'user/flag',
            method: 'PATCH',
            data: tmpData,
            error: function(){
                Utils.Notification.Toast('Failed Flag');
                debugger;
            },
            success: function(){
                // awesome
            }
        });
    },
```

Sample usage:

```
    var localFlag = 'highlights/home';
    Utils.CheckFlag(localFlag).then(function(){
        // popover
        alert('this will only happen once!');
        // update flag
        Utils.PostFlag(localFlag, true);
    });
```

QuickModels

Here we provide a simple way to return a single model instance.

```
    QuickModel: function(ModelName, model_file){
        if(model_file == undefined){
            model_file = ModelName.toLowerCase();
        }

        var defer = $.Deferred();

        require(['models/' + model_file], function(Model){

            if(!id || id.length < 1){
                defer.reject();
                return;
            }
```

```
            var newModel = new Model[ModelName]({
                _id: id
            });
            if(newModel.hasFetched){
                // Already fetched
                defer.resolve(newModel);
            } else {
                newModel.fetch({prefill: true});
                newModel.populated().then(function(){
                    defer.resolve(newModel);
                });
            }
        });

        return defer.promise();

    },
```

Replacing surfaces with resolved models

aka **dataModelReplaceOnSurface** is a handy way of adding Surfaces with content tags that get filled in as the model populates.

```
    dataModelReplaceOnSurface : function(Surface){
        // Load driver name and image
        // console.log(context);
        if(typeof App.Cache != typeof {}){
            App.Cache = {};
        }

        var context = $('<div/>').html(Surface.getContent());

        App.Cache.ModelReplacers = App.Cache.ModelReplacers || {};

        context.find('[data-replace-model]').each(function(index){
            var elem = this;
            var model = $(elem).attr('data-replace-model'),
                id = $(elem).attr('data-replace-id'),
                field = $(elem).attr('data-replace-field'),
                target = $(elem).attr('data-target-type'),
                pre = $(elem).attr('data-replace-pre') || '',
                cachestring = 'cached_display_v1_' + model + id + field;

        });



    },
```

This lets us do something like:

```
var s = new Surface({
    content: '<span data-replace-id="'+player_id+'" data-replace-model="Player" data-replace
});

Utils.dataModelReplaceOnSurface(s);
```

and after my "Player" is resolved (probably cached) it will replace the corresponding fields.

Locale

Localization / Globalization

```
    Locale: {

        normalize: function(value){
            var tmpValue = value.toString().toLowerCase();
            var allowed_locales = {
                'en' : ['undefined','en','en_us','en-us']
            };

            var normalized = false;
            _.each(allowed_locales, function(locales, locale_normalized){
```

```
            if(locales.indexOf(tmpValue) !== -1){
                normalized = locale_normalized;
            }
        });

        return normalized;
    }

},
```

Analytics

Called from `device_ready.js.

```
Analytics: {
    init: function(){
        try {
            App.Analytics = window.plugins.gaPlugin;
            App.Analytics.init(function(){
                // success
                console.log('Success init gaPlugin');
            }, function(){
                // error
                if(App.Data.usePg){
                    console.error('Failed init gaPlugin');
                    Utils.Notification.Toast('Failed init gaPlugin');
                }
            }, Credentials.GoogleAnalytics, 30);
        }catch(err){
            if(App.Data.usePg){
                console.error(err);
            }
            return false;
        }

        return true;

    },

    trackRoute: function(pageRoute){
        // needs to wait for Utils.Analytics.init()? (should be init'd)
        try{
            App.Analytics.trackPage( function(){
                // success
                console.log('success');
            }, function(){
                // error
                console.error('error');
            }, 'nemesis.app/' + pageRoute);
        }catch(err){
            if(App.Data.usePg){
                console.error('Utils.Analytics.trackPage');
                console.error(err);
            }
        }
    }
},
```

Take Picture

We use a consistent image size and quality across the app (though you can provide options to combine with).

Requires `options.sourceType` to be either `Camera.PictureSourceType.PHOTOLIBRARY` or `Camera.PictureSourceType.CAMERA`.

```
takePicture: function(camera_type_short, opts, successCallback, errorCallback){
    // Take a picture using the camera or select one from the library
    var that = this;

    var options = {
        quality: 80,
        targetWidth: 1000,
        targetHeight: 1000,
        destinationType: Camera.DestinationType.FILE_URI,
```

```
            encodingType: Camera.EncodingType.JPEG,
            sourceType: null // Camera.PictureSourceType.CAMERA
        };

        switch(camera_type_short){
            case 'gallery':
                options.sourceType = Camera.PictureSourceType.PHOTOLIBRARY;
                break;

            case 'camera':
            default:
                // camera
                options.sourceType = Camera.PictureSourceType.CAMERA;
                break;
        }

        navigator.camera.getPicture(successCallback, errorCallback, options);

        return false;

    },
```

HTML Encoding

By default, you can use

```
S( "<h1>Bad HTML HERE!</h1>" );
```

for encoding/sanitizing user input before displaying. `S()` maps to the `Utils.hbSanitize` function.

```
    hbSanitize: function(string){
        // copied from Handlebars.js sanitizing of strings
        var escape = {
            "&": "&amp;",
            "<": "&lt;",
            ">": "&gt;",
            '"': "&quot;",
            "'": "&#x27;",
            "`": "&#x60;"
        };

        var badChars = /[&<>"'`]/g;
        var possible = /[&<>"'`]/;

        function escapeChar(chr) {
            return escape[chr] || "&amp;";
        }

        if (!string && string !== 0) {
          return "";
        }
        string = "" + string;

        if(!possible.test(string)) { return string; }
        return string.replace(badChars, escapeChar);
    },
```

We also have the option of using jQuery to do our sanitization.

```
    htmlEncode: function(value){
      //create a in-memory div, set it's inner text(which jQuery automatically encodes)
      //then grab the encoded contents back out.  The div never exists on the page.
      return $('<div/>').text(value).html();
    },
```

Notifications

```
    Notification: { ...
```

Toast

Defaults to using a native plugin for displaying a system-wide Toast dialog box, with positioning (top, middle, bottom).

You can also use the `App.MainToast.show('text')` to display a toast for a moment.

```
        Toast: function(msg, position){
            // attempting Toast message
            // - position is ignored
            var defer = $.Deferred();
            try {
                window.plugins.toast.showShortBottom(msg,
                    function(a){
                        defer.resolve(a);
                    },
                    function(b){
                        defer.reject(b);
                    }
                );
            }catch(err){
                console.log('TOAST failed');
            }
            return defer.promise();
        }
    },
```

Base64 libraries

```
    /**
    *
    *  Base64 encode / decode
    *  http://www.webtoolkit.info/
    *
    **/
    Base64: {

        // private property
        _keyStr : "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=",

        // public method for encoding
        encode : function (input) {
            var output = "";
            var chr1, chr2, chr3, enc1, enc2, enc3, enc4;
            var i = 0;

            input = this._utf8_encode(input);

            while (i < input.length) {

                chr1 = input.charCodeAt(i++);
                chr2 = input.charCodeAt(i++);
                chr3 = input.charCodeAt(i++);

                enc1 = chr1 >> 2;
                enc2 = ((chr1 & 3) << 4) | (chr2 >> 4);
                enc3 = ((chr2 & 15) << 2) | (chr3 >> 6);
                enc4 = chr3 & 63;

                if (isNaN(chr2)) {
                    enc3 = enc4 = 64;
                } else if (isNaN(chr3)) {
                    enc4 = 64;
                }

                output = output +
                this._keyStr.charAt(enc1) + this._keyStr.charAt(enc2) +
                this._keyStr.charAt(enc3) + this._keyStr.charAt(enc4);

            }

            return output;
        },

        // public method for decoding
        decode : function (input) {
            var output = "";
```

```
        var chr1, chr2, chr3;
        var enc1, enc2, enc3, enc4;
        var i = 0;

        input = input.replace(/[^A-Za-z0-9\+\/\=]/g, "");

        while (i < input.length) {

            enc1 = this._keyStr.indexOf(input.charAt(i++));
            enc2 = this._keyStr.indexOf(input.charAt(i++));
            enc3 = this._keyStr.indexOf(input.charAt(i++));
            enc4 = this._keyStr.indexOf(input.charAt(i++));

            chr1 = (enc1 << 2) | (enc2 >> 4);
            chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);
            chr3 = ((enc3 & 3) << 6) | enc4;

            output = output + String.fromCharCode(chr1);

            if (enc3 != 64) {
                output = output + String.fromCharCode(chr2);
            }
            if (enc4 != 64) {
                output = output + String.fromCharCode(chr3);
            }

        }

        output = this._utf8_decode(output);

        return output;

    },
```

UTF-8 encoding/decoding

```
    // private method for UTF-8 encoding
    _utf8_encode : function (string) {
        string = string.replace(/\r\n/g,"\n");
        var utftext = "";

        for (var n = 0; n < string.length; n++) {

            var c = string.charCodeAt(n);

            if (c < 128) {
                utftext += String.fromCharCode(c);
            }
            else if((c > 127) && (c < 2048)) {
                utftext += String.fromCharCode((c >> 6) | 192);
                utftext += String.fromCharCode((c & 63) | 128);
            }
            else {
                utftext += String.fromCharCode((c >> 12) | 224);
                utftext += String.fromCharCode(((c >> 6) & 63) | 128);
                utftext += String.fromCharCode((c & 63) | 128);
            }

        }

        return utftext;
    },

    // private method for UTF-8 decoding
    _utf8_decode : function (utftext) {
        var string = "";
        var i = 0;
        var c2 = 0,
            c1 = c2,
            c = c1;

        while ( i < utftext.length ) {

            c = utftext.charCodeAt(i);
```

```
                if (c < 128) {
                    string += String.fromCharCode(c);
                    i++;
                }
                else if((c > 191) && (c < 224)) {
                    c2 = utftext.charCodeAt(i+1);
                    string += String.fromCharCode(((c & 31) << 6) | (c2 & 63));
                    i += 2;
                }
                else {
                    c2 = utftext.charCodeAt(i+1);
                    c3 = utftext.charCodeAt(i+2);
                    string += String.fromCharCode(((c & 15) << 12) | ((c2 & 63) << 6) | (c3 &
                    i += 3;
                }

            }

            return string;
        }

    },
```

Logout

Logging out can get tricky, so we try and handle all the use cases here. Clear caches, unregister from Push Notifications, reset the $.ajax headers.

```
    logout: function(){

        // Reset caches
        App.Cache = {}; //_.defaults({},App.DefaultCache);
        // console.log(App.Cache);
        App.Data = {
            User: null,
            Players: null // preloaded
        };
        localStorage.clear();

        // Unregister from Push
        console.info('Unregisering from PushNotification');
        try {
            window.plugins.pushNotification.unregister();
        }catch(err){
            console.error('Failed unregistering from PushNotification');
        }

        // Reset credentials
        $.ajaxSetup({
            headers: {
                'x-token' : ''
            }
        });

        // // Try and exit on logout, because we cannot effectively clear views
        // try {
        //     navigator.app.exitApp();
        // } catch(err){
        // }

        // try {
        //     navigator.device.exitApp();
        // } catch(err){
        // }

        // Last effort, reload the page
        // - probably lose all native hooks
        // console.log(window.location.href);
        window.location = window.location.href.split('#')[0] + '#login';

        return true;
    },
```

GPS functions, updating and haversine

```
    updateGpsPosition: function(){

        try {
            navigator.geolocation.getCurrentPosition(function(position){
                console.log('coords');
                console.log(position.coords);
                App.Events.emit('updated_user_current_location');
                App.Cache.geolocation_coords = position.coords;

            }, function(err){
                console.log('GPS failure');
                console.log(err);
            });
        } catch(err){
            return false;
        }

        return true;

    },

    haversine : function(lat1,lat2,lon1,lon2){

        // Run haversine formula
        var toRad = function(val) {
           return val * Math.PI / 180;
        };

        // var lat2 = homelat;
        // var lon2 = homelon;
        // var lat1 = lat;
        // var lon1 = lon;

        var R = 3959; // km=6371. mi=3959

        //has a problem with the .toRad() method below.
        var x1 = lat2-lat1;
        var dLat = toRad(x1);
        var x2 = lon2-lon1;
        var dLon = toRad(x2);
        var a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                        Math.cos(toRad(lat1)) * Math.cos(toRad(lat2)) *
                        Math.sin(dLon/2) * Math.sin(dLon/2);
        var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
        var d = R * c;

        return d;

    },
```

Push Notifications

```
    process_push_notification_message : function(e){
        // Processing a single Push Notification
        // - not meant for handling a bunch in a row
```

We are told if the notification was received while the app was being looked at (imagine when somebody instantly accepts a friend request).

```
        if (e.foreground) {
            switch(e.payload.type){
                // no case: statements yet
                case 'default':
                    // nothing
                    console.log('default');
                default:
                    break;
            }
```

If the notification contains a soundname, play it.

```
            // var my_media = new Media("/android_asset/www/"+e.soundname);
            // my_media.play();
```

If we launched and the app wasn't in the foreground (running), we could do something else.

```
        } else {
            // Launched because the user touched a notification in the notification tray.
        }
```

We can now run our default actions based on the payload of the Push Notification. You don't have to worry about actually registering, receiving, etc. from here, that is handled in `device_ready.js`

```
        // Default actions to follow

        // Is there a URL we should be visiting?
        switch(e.payload.type){
            case 'url':
                // Visit an internal url
                App.history.navigate(e.payload.url);
                break;

            default:
                // Unknown type
                // - don't do anything
                alert('Unable to process Push Notification');
                break;
        }

    },
```

Random Functions

slugToCamel

```
    slugToCamel: function (slug) {
        var words = slug.split('_');

        for(var i = 0; i < words.length; i++) {
          var word = words[i];
          words[i] = word.charAt(0).toUpperCase() + word.slice(1);
        }

        return words.join(' ');
    },
```

Random integer

```
    randomInt: function(min, max){
        return Math.floor(Math.random() * (max - min + 1)) + min;
    },
```

Fixed int or none

```
    toFixedOrNone: function(val, len){
        var tmp = parseFloat(val).toFixed(len).toString();
        if (parseInt(tmp, 10).toString() == tmp){
            return isNaN(tmp) ? '--' : parseInt(tmp, 10).toString();
        }
        return isNaN(tmp) ? '--' : tmp;
    },
```

Push Notifications

Push Notifications have a server and client-side component.

Certificates and Setup

iOS

Apple uses the Apple Push Notifiation service (APN). This is an excellent tutorial on obtaining keys:
http://www.raywenderlich.com/32960/apple-push-notification-services-in-ios-6-tutorial-part-1

Android

Use Google Cloud Messaging to obtain the necessary keys. http://developer.android.com/google/gcm/gs.html

Server Side

We use a PushSetting model per-user to keep track of what they'd like to be notified on.

We use a common format for sending a Push Notification to a user. This checks a user's PushSetting and PushSettingMod to determine if a Push Notifiation should be sent, and then sends the corresponding iOS or Android notification.

```
m.PushNotification.pushToUser(user_id, {
    ios_title: 'iOS Push Message',
    title: 'Title of Push Notification',
    message: 'body of message, Android only',
    payload: {type: 'new_message', id: newMessage._id}
}, 'pushsetting_mod_key', pushsetting_mod_conditions);
```

iOS has only one field (`ios_title`) while **Android** accepts both `title` and `message`.

Client Side

Couple of steps:

- register device for Push Notifications
- upload token to server
- handle incoming messages and payloads

Inside `device_ready.js` we handle registration for iOS and Android Push Notifications.
`Utils.process_push_notification_message` is the function that handles the incoming payload.

```
process_push_notification_message : function(payload){

    if(typeof payload === typeof ""){
        payload = JSON.parse(payload);
    }
```

Your payload could be anything... in our example we show a Popover asking the user if they want to view the message.

```
    switch(payload.type){
        case 'new_message':
            Utils.Popover.Buttons({
                title: 'New Message',
                buttons: [
                    {
                        text: 'Ignore'
                    },
                    {
                        text: 'View Message',
                        success: function(){
                            App.history.navigate('message/' + payload.id);
                        }
                    }
                ]
            });

            break;
```

When will a Push Notification pop up?

On the server, you need to include the `title` and `message` parameters (Android) or `body` (iOS). Skipping these will result in the Push

Notification being delivered in the background (nothing audible or visual to the user).

localization

Localization can be done pretty easily. Everything is initialized in `main.js` and anywhere that you want to change strings, simply use `App.t(stringName)`.

Basics

i18next plugin: powers everything. Configuration in `main.js`.

Directory Structure:

```
locales/
    dev/
        locale.strings
        ns.common.json
    en/
        locale.strings
        ns.common.json
```

Add additional languages (like `ru`) by including both `locale.strings` and `ns.common.json`.

> Notice that `locale.strings` is simply a dummy file, used by PhoneGap Build for telling the iOS/etc. stores what languages you intend to support

Models / Collections

stub

This section is under development.

Model

An except from `models/action.js`:

```
var Action = Backbone.DeepModel.extend({

    idAttribute: '_id',

    urlRoot: Credentials.server_root + "action/",

    initialize: function () {
        // ok
        this.url = this.urlRoot + '';
        if(this.id){
            this.url = this.urlRoot + '/' + this.id;
        }
    }

});

Action = Backbone.UniqueModel(Action);
```

Backbone.DeepModel

Allows us to query nested attributes in a model. We can do:

```
ourModel.get('data.nested.attr');
```

instead of repetitive code like:

```
if(ourModel.get('data') && ourModel.get('data').nested && ourModel.get('data').nested.attr){
```

Backbone.UniqueModel

Gives us an App-wide version of *the same* model everytime we instantiate.

```
var ActionModel = require('models/action');
var a = ActionModel.Action({
    _id: "xyz" // ObjectId
});
a.on('question', function({
    console.log('answer');
});
var b = ActionModel.Action({
    _id: "xyz", // ObjectId
});

a === b
> true

b.trigger('question');
> "answer"
```

Collections

Collections can come in a few flavors:

- pagination
- summaries
- normal

Pagination

This needs to be updated to the current version of the Pagination component

```
var ActionCollection = Backbone.Paginator.requestPager.extend({

    model: Action,

    urlRoot: Credentials.server_root + "actions",

    // Paginator Core
    paginator_core: {
      type: 'GET',
      dataType: 'json',
      url: function(){return this.urlRoot}
    },

    // Paginator User Interface (UI)
    paginator_ui: {
      firstPage: 0,
      currentPage: 0, // start/current page
      perPage: 5,
      totalPages: 10 // gets overwritten
    },

    // Paginator Server API
    server_api: {
      '$filter': '',
      '$top': function() { return this.perPage },
      '$skip': function() { return this.currentPage * this.perPage },
      '$format': 'json',
      '$inlinecount': 'allpages',
      // '$callback': 'callback'
    },

    // Paginator Parsing
    parse: function (response) {
        this.totalPages = Math.ceil(response.total / this.perPage);
        this.totalResults = response.total;
        return response.results;
    },


    initialize: function(models, options){
        options = options || {};
        if(options.player_id){
            // Viewing actions for an individual player (what everybody would see about that
            // - used for what I see about myself, and news/actions about other individual p
            this.url = this.urlRoot + '/player/' + options.player_id;
        } else {
            // Summary of actions that I care about (me, as a logged in user_id)
            if(options.type){
              this.url = this.urlRoot + '/' + options.type;
            }
        }
    }

});
```

_id and __v instead of full model representations

A big difference between mobile and desktop apps is the **mobile connection**, both **speed** and **reliability**. In order to make your app more responsive when loading data, you can avoid returning entire model representations and instead simply return `_id` and `__v` from your server.

If your collection only contains `_id` and `__v` it will NOT trigger a `reset` or other events (or return `success` or resolve the `.then` promise) until all of the models have been updated for the collection (using Backbone.UniqueModel) to the correct version.

Live Updating / Streams / Websockets / Push Models

stub

How does this work?

- rooms created per-user (like Emailbox)
- is Firebase overkill? (yeah)
- Just need to be notified of new models, changes, and to know the last __version, corret?

This will be tested/used on the Inbox PageView at first. It is a perfect candidate to be "pushed" updates.

When a collection is fetched, it would want to know when a certain model type is updated, correct?

Collection "live" status can be

- triggers a "fetch" or "pager" or something on the collection?
  - nah, just listen for an update, then do the necessary fetch/pager
  - `App.Events.on('model-added-of-type-Profile', ...-`

Models listen on App.Events.

if App.Events is a Firebase queue, can I choose which queue it is on? And then set permissions to access that queue?

- yes -

Looks like I need to simply add an Auth rules for every queue (

https://www.firebase.com/docs/security/custom-login.html

one of your models has been updated, or a friend's model updated

- on server: determine everybody the action/db change affects (user_ids)
- determine the Models affected (User, Message)
- increment _version of that Model for each affected user
- basically creates a few rooms/channels/urls(firebase) for each user
- on client: connect to stream/websocket from server
- when an update occurs, we'll get the Model it affects, and update all the created local models (by checking version?)
- ...does it make sense to update all the models? -

Views

stub

Views Ok, you've got a decent understanding of how we're choosing which page to display, so now lets begin at the start…what happens when somebody wants to login?

Login (have them create their own Login page) structure of PageView and prototypes create layout (using HeaderFooterLayout) no header, no footer for this view, but you'll use this pattern frequently create content, add surfaces

- different patterns for creating spacers (either add a new Surface, or a StateModifier) inOutTransition submitting via button press, ajax request for login/signup

signup requesting multiple fields, but not username!

- server handles most of the profile-building and whatnot, simply returns success/fail of creation on success, does a login and continues as normal

Welcome screens (popovers) occurs after login or signup, after the user has been populated once (to check if username exists)

Directory Structure

Our views are contained in

```
root_app/www/src/views/
```

Notable directories include `common` and `Misc`.

`common` contains re-usable components such as `StandardHeader` and `StandardTabBar` that are slightly-modified version of the normal Famo.us components.

`Misc` contains a mishmash of Views such as

- Login
- HelpPopover
- OptionModal
- Settings
- Feedback

Naming Conventions

- models: singluar. lowercase (slugged)
- views: generally singular, CamelCased

Patterns

stub

FullPage Option List (not modal) Popover List HelpPopover (CheckFlag/PostFlag) Wizards Handlebars.js for templating (prototyping) QR code (or other tool that modifies the DOM, waits for “deploy” event)

Subviews

- loading, displaying, attaching correctly to the RenderTree (with correct getSizing’ing)

Lists

- infinite lists
- removing items from a list (how to transition it out)
- not fetched
- loading
- 0 total results

Common Views

I've slightly modified a few famo.us components, and included them in the `views/common/` directory.

`StandardHeader` and `StandardNavigationBar`

Basically every view uses this combination. Together, they handle "Back" transitions, title surfaces, addditional surfaces (the "more" surfaces in the top-right).

StandardHeader

Includes a back icon, title text, and optional `moreSurfaces` for including icons/links/buttons in the header.

```
PageView.prototype.createHeader = function(){
    var that = this;

    this.headerContent = new View();
    this.headerContent.Settings = new Surface({
        content: '<i class="icon ion-gear-a">',
        size: [60, undefined],
        classes: ['header-tab-icon-text-big']
    });
    this.headerContent.Settings.on('click', function(){
        App.history.navigate('settings');
    });

    // create the header
    this.header = new StandardHeader({
        content: 'Invite Friends',
        classes: ["normal-header"],
        backClasses: ["normal-header"],
        moreSurfaces: [
            this.headerContent.Settings
        ]
    });
    this.header._eventOutput.on('back',function(){
        App.history.back();
    });
    this.header.navBar.title.on('click',function(){
        App.history.back();

    });
    this.header.pipe(this._eventInput);
```

The StandardHeader is equipped to handle `inOutTransition` events.

```
    this._eventOutput.on('inOutTransition', function(args){
        this.header.inOutTransition.apply(this.header, args);
    })

    // Attach header to the layout
    this.layout.header.add(this.header);

};
```

StandardNavigationBar

This is used as part of the `StandardHeader`, and includes the logic for `moreSurfaces`.

StandardTabBar

Uses `StandardToggleButton` and changed to not emit an event if `triggerEvent === false` on `.select`.

StandardToggleButton

Modified `.select` to allow a change of state, without triggering an event.

```
App.MainFooter.Tabs.select('dash', false);
```

Login

The Login view has no Header or Footer.

It is simply the necessary Surfaces, centered vertically and horizontally.

Route

Helps to know what we're being initialized with!

> Notice that we are not caching the Login View

`www/router.js`

```
'login' : function(){
    defaultRoute('Login', 'Misc/Login', arguments, {cache: false});
},
```

Initialization

Back to `www/views/Misc/Login.js`, we initialize the PageView by applying all of a Famo.us `View`'s properties.

```
function PageView(options) {
    var that = this;
    View.apply(this, arguments);
    this.options = options;
```

We've already required (`var UserModel = require('models/user');`) our model, now we create an empty User to hold our credentials to login with later.

```
    // Model
    this.model = new UserModel.User();
```

Add your background surface. Remember, order counts (but so does Transform position), so we use both this "put it on the RenderTree first" and add a Modifier to move it backwards in Z space.

```
    // Add background
    var bgSurface = new Surface({
        size: [undefined, undefined],
        classes: ['bg-surface']
    });
    var backMod = new StateModifier({
        transform: Transform.behind
    });
    this.add(backMod).add(bgSurface);
```

We'll use a simple SequentialLayout (link to Famo.us University, or links to our included resources?) here, as we only have a few fields and don't anticipate a need to scroll.

```
    // Create the layout
    this.layout = new SequentialLayout();
```

You'll notice this pattern used frequently:

```
    this.layout.Views = [];
```

> Notice that the SequentialLayout is going to sequenceFrom "itself" i.e. `this.layout.views` later

Now we add our surfaces, in order.

Each Surface is `.push`ed onto the `this.layout.Views` array.

```
    this.topWelcomeSurface = new Surface({
        content: "Nemesis",
        size: [undefined, 80],
        classes: ['login-page-welcome-top-default']
    });
    this.layout.Views.push(this.topWelcomeSurface);

    this.inputEmailSurface = new InputSurface({
        name: 'email',
        placeholder: 'Email Address',
        type: 'text',
        size: [undefined, 50],
        value: '' //nicholas.a.reed@gmail.com
    });
    this.layout.Views.push(this.inputEmailSurface);
```

This is one way of adding spacers: simply add a blank Surface with whatever size you need. If this were a ScrollView though, you'd also need to `.pipe` the Surface in order to avoid any weird un-scrollable patches on the screen.

```
    this.spacer1 = new Surface({
        content: "",
        size: [undefined,4]
```

```
    });
    this.layout.Views.push(this.spacer1);
```

Here you'll see another method of adding a spacer after the `inputPasswordSurface`. In this instance, we add a `StateModifier` with a slightly larger `size` that the contained Surface.

```
    this.inputPasswordView = new View();
    this.inputPasswordView.Surface = new InputSurface({
        name: 'password',
        placeholder: 'Password',
        type: 'password',
        size: [undefined, 50],
        value: '' //testtest
    });
    this.inputPasswordView.PaddingMod = new StateModifier({
        size: [undefined, 54]
    });
    this.inputPasswordView.add(this.inputPasswordView.PaddingMod).add(this.inputPasswordView
    this.layout.Views.push(this.inputPasswordView);
```

Our Login button will trigger the `login` function.

```
    this.submitSurface = new Surface({
        size: [undefined,60],
        classes: ['form-button-submit-default'],
        content: 'Login'
    });
    this.submitSurface.on('click', this.login.bind(this));
    this.layout.Views.push(this.submitSurface);


    this.spacerSurface = new Surface({
        content: "",
        size: [undefined, 20]
    });
    this.layout.Views.push(this.spacerSurface);
```

Our Signup and Forgot Password links are styled differently.

```
    this.signupLinkSurface = new Surface({
        content: 'Signup &gt;',
        size: [undefined,40],
        classes: [],
        properties: {
            color: "black",
            lineHeight: "40px",
            textAlign: "right"
        }
    });
    this.signupLinkSurface.on('click', function(){
        App.history.navigate('signup', {trigger: true});
    });
    this.layout.Views.push(this.signupLinkSurface);


    this.forgotLinkSurface = new Surface({
        content: 'Forgot Password &gt;',
        size: [undefined,40],
        classes: [],
        properties: {
            color: "black",
            lineHeight: "40px",
            textAlign: "right"
        }
    });
    this.forgotLinkSurface.on('click', function(){
        App.history.navigate('forgot', {trigger: true});
    });
    this.layout.Views.push(this.forgotLinkSurface);
```

Now that all our Surfaces (and one View!) are created and added to the array, we'll `sequenceFrom`.

```
    this.layout.sequenceFrom(this.layout.Views);
```

Finally, we created an `originMod` and `sizeMod` to get everything centered and sized correctly. The last step is adding to `this` PageView.

```
    var originMod = new StateModifier({
        origin: [0.5, 0.5]
    });
    var sizeMod = new StateModifier({
        size: [window.innerWidth - 16, undefined]
    });

    this.add(sizeMod).add(originMod).add(this.layout);

}
```

Login function

When the Login button is pressed, we need to:

- gather the inputs
- validate them slightly (no sense making a user wait for a "hey, you didn't enter a password" response from the server)
- check against the server
- store the token if the login succeeded
- redirect to their homepage

```
PageView.prototype.login = function(){
    var that = this;

    if(this.checking === true){
        return;
    }
    this.checking = true;

    var email = this.inputEmailSurface.getValue(),
        password = this.inputPasswordSurface.getValue();
```

After clicking, we'll disable the input for a moment, and inform the user.

```
    this.submitSurface.setContent('Please wait...');
```

Set the data for the login request.

```
    var body = {
        email: email,
        password: password
    }
```

Make the request using the `login` method on our `User` model.

```
    this.model.login(body)
```

If the login fails, inform the user and reset the submit button text.

```
    .fail(function(){
        alert('Failed logging in');
        that.submitSurface.setContent('Login');
        that.checking = false;

    })
```

> We check for a failure condition with the response from the server too

```
    .then(function(response){
        if(response.code != 200){
            alert('Failed signing in (3424)');
            that.submitSurface.setContent('Login');
            that.checking = false;
            return;
        }
```

When the login succeeds, we save the `token` and preload all the models for the new User.

```
        localStorage.setItem('usertoken_v1_',response.token);
        App.Data.UserToken = response.token;

        that.model.fetch({
            error: function(){
                alert("Failed gathering user model");
            },
            success: function(userModel){
                console.log('UserModel');
```

```
                console.log(userModel);
```

You can access the User model anytime at `App.Data.User`.

```
                that.options.App.Data.User = userModel;

                localStorage.setItem('user_v3_',JSON.stringify(userModel.toJSON()));


Preload the user's models (friends, Posts, etc.)

                require(['models/_preload'], function(PreloadModels){
                    PreloadModels(that.options.App);
                });
```

We need to register for Push Notifications too.

```
                // Register for Push Notifications
                App.DeviceReady.initPush();
```

Finally, use `eraseUntilTag` to erase the whole history (because no tags called `allofem` are created, of course) and then navigate to our homepage.

```
                // Reload home
                App.history.eraseUntilTag('allofem');
                App.history.navigate('dash');

            }
        });

    });

};
```

Transitions

Left out of our Login PageView is an inOutTransition function. Instead, we use a simple StoredTransition and ViewToView for displaying the Login page.

Signup

Signup uses many of the concepts introduced in the `Login` PageView, but introduces the `HeaderFooterLayout` that is used in almost every other PageView.

Route

www/router.js

```
'signup' : function(){
    defaultRoute('Signup', 'Misc/Signup', arguments, {cache: false});
},
```

Initialization

```
function PageView(options) {
    var that = this;
    View.apply(this, arguments);
    this.options = options;
```

Initialize User model, same reasoning as `Login`.

```
    // User
    this.model = new UserModel.User();
```

`HeaderFooterLayout`

This is a standard Famo.us component. It is basically a vertical `FlexibleLayout` with up 3 surfaces allowed, but defined using `header`, `content`, and `footer`. `content` will always fill the remaining vertical space between

If you think of it as a `FlexibleLayout` it would have `ratios: [true, 1, true]`.

layout.header /////// layout.content ////// layout.footer

We also define our sizing here, though in the future you may want to provide a function for `headerSize` (or a workaround to offer a Modifier on the header's `getSize` function).

Notice that we've made footerSize

```
    this.layout = new HeaderFooterLayout({
        headerSize: 50,
        footerSize: 0
    });
```

Create Header

```
    this.createHeader();
```

Create Content

```
    this.createContent();
```

Add layout to RenderTree

```
    this.add(this.layout);
}
```

function `createHeader()`

> You should read about `StandardHeader` and how to use it.

We'll create a header with only a "Back" button, and the standard "her I want to go back" event options.

```
PageView.prototype.createHeader = function(){
    var that = this;

    this.header = new StandardHeader({
        content: "Signup",
        classes: ["normal-header"],
        backClasses: ["normal-header"],
        moreContent: false
    });
    this.header._eventOutput.on('back',function(){
        App.history.back();//.history.go(-1);
    });
    this.header.navBar.title.on('click',function(){
        App.history.back();//.history.go(-1);
    });
    this._eventOutput.on('inOutTransition', function(args){
        this.header.inOutTransition.apply(that.header, args);
    })
```

Add the header to the created `HeaderFooterLayout`.

```
    this.layout.header.add(this.header);
};
```

function `createContent()`

Our `Signup` PageView is similar to `Login` in that it uses a vertically-centered `SequentialLayout` (instead of a `ScrollView`).

But, there is now a `this.layout.content.StateModifier` that we'll use for transitioning in everything under our `this.layout.content`.

```
PageView.prototype.createContent = function(){
    var that = this;

    // create the scrollView of content
    this.contentScrollView = new SequentialLayout(); //(App.Defaults.ScrollView);
    this.contentScrollView.Views = [];
    this.contentScrollView.sequenceFrom(this.contentScrollView.Views);
```

Adding the input fields and submit button Surfaces.

```
    this.addSurfaces();
```

Adding our `StateModifier`, and the `RenderNode/View` we'll be using to store everything before having it `add`ed it to `this.layout.content`.

```
    this.layout.content.StateModifier = new StateModifier();
    this.contentView = new View();
    this.contentView.SizeMod = new Modifier({
        size: //[window.innerWidth - 50, true]
            function(){
                var tmpSize = that.contentScrollView.getSize(true);
                if(!tmpSize){
                    return [window.innerWidth, undefined];
                }
                return [window.innerWidth - 16, tmpSize[1]];
            }
    });
    this.contentView.OriginMod = new StateModifier({
        origin: [0.5, 0.5]
    });
    this.contentView.add(this.contentView.OriginMod).add(this.contentView.SizeMod).add(this.
```

```
    this.layout.content.add(this.layout.content.StateModifier).add(this.contentView);

};
```

function `addSurfaces()`

Adding only three Surface:, **Email** and **Password** and **Signup**.

```
PageView.prototype.addSurfaces = function() {
    var that = this;

    // Email
    this.inputEmailSurface = new InputSurface({
        name: 'email',
        placeholder: 'Email Address',
        type: 'text',
        size: [undefined, 50],
        value: ''
    });
    this.contentScrollView.Views.push(this.inputEmailSurface);

    this.contentScrollView.Views.push(new Surface({
        size: [undefined, 4]
    }));

    // Password
    this.inputPasswordSurface = new InputSurface({
        name: 'password',
        placeholder: 'Password',
        type: 'password',
        size: [undefined, 50],
        value: ''
    });
    this.contentScrollView.Views.push(this.inputPasswordSurface);

    this.contentScrollView.Views.push(new Surface({
        size: [undefined, 4]
    }));

    // Submit button
    this.submitButtonSurface = new Surface({
        size: [undefined,60],
        classes: ['form-button-submit-default'],
        content: 'Signup'
    });
    this.contentScrollView.Views.push(this.submitButtonSurface);
```

Handle the Signup button being clicked.

```
    this.submitButtonSurface.on('click', this.create_account.bind(this));

};
```

Creating the account

We do a slight amount of validation on the email address and password, before submitting using collected form data (`inputEmailSurface.getValue()`), verifying a successful signup, logging in, and redirecting to the `welcome` PageView.

```
PageView.prototype.create_account = function(ev){
    var that = this;

    if(this.checking === true){
        return;
    }
    this.checking = true;

    // Get email and password
    var email = $.trim(this.inputEmailSurface.getValue().toString());
    if(email.length === 0){
        this.checking = false;
        return;
    }

    var password = this.inputPasswordSurface.getValue().toString();

    // Disable submit button
    this.submitButtonSurface.setContent('Please wait...');
```

Gather our data, and submit to the `signup` endpoint on our server.

```
    var dataBody = {
        email: email,
        password: password
    };

    // Fetch user
    $.ajax({
        url: Credentials.server_root + 'signup',
        method: 'POST',
        data: dataBody,
```

If signup fails in transit (500 error, timeout, etc.) handle it here.

```
        error: function(err){

            Utils.Notification.Toast('Failed creating Nemesis account');
            that.submitButtonSurface.setContent('Signup');
            that.checking = false;

        },
```

On `success` we try to login using the same email/password.

```
        success: function(response){

            that.checking = false;

            if(response.complete == true){
                // Awesome, created the new user on the server
```

If the login fails (should happen rarely/never) we redirect to the login page.

If successful, we do everything from the success part of the `Login` PageView, but the redirect to `welcome` instead.

```
                that.model.login(dataBody)
                .fail(function(){
                    Utils.Notification.Toast('Failed login after signup');
                    App.history.navigate('login');

                })
                .then(function(response){
                    // Success logging in

                    // Fetch model
```

```
                    if(response.code != 200){
                        console.error('Failed signing in (3424)');
                        console.log(response);
                        console.log(response.code);
                        Utils.Notification.Toast('Failed signing in (3424)');
                        that.submitButtonSurface.setContent('Signup');
                        return;
                    }

                    // Store access_token in localStorage
                    localStorage.setItem('usertoken_v1_',response.token);
                    App.Data.UserToken = response.token;

                    // Get's the User's Model
                    that.model.fetch({
                        error: function(){
                            alert("Failed gathering user model");
                        },
                        success: function(userModel){

                            // Set global logged in user
                            that.options.App.Data.User = userModel;

                            localStorage.setItem('user_v3_',JSON.stringify(userModel.toJSON(

                            // Preload Models
                            require(['models/_preload'], function(PreloadModels){
                                PreloadModels(that.options.App);
                            });

                            // Register for Push Notifications
                            App.DeviceReady.initPush();
```

Here is where we go to the `welcome` route (after clearing history). App.history.eraseUntilTag('allofem'); App.history.navigate('welcome');

```
                        }
                    });

                });

                return;

            }
```

Here is where our error messages for the responses from server. You could also return a `msg` paramter with your responses and use that instead.

Remember that we only get this far if the signup was unsuccessful.

```
            // See what the error was
            switch(response.msg){
                case 'bademail':
                    alert('Sorry, that does not look like an email address to us.');
                    break;
                case 'duplicate':
                    alert('Sorry, that email is already in use.');
                    break;
                case 'unknown':
                default:
                    alert('Sorry, we could not sign you up at the moment, please try again!'
                    break;

            }

            // Re-enable submit button
            that.submitButtonSurface.setContent('Signup');

            return false;

        }

    });

};
```

Transitions

We want to transition our PageView differently depending on whether it is being shown or hidden, and how we're arriving/leaving the page. This `inOutTransition` function is used in nearly every PageView and makes it easy to define custom PageView-level transition/transforms.

For `Signup`, we want to modify the **opacity** of our

```
PageView.prototype.inOutTransition = function(direction, otherViewName, transitionOptions, d
    var that = this;
```

We told the `StandardHeader` to listen for this event, because it does its own transition/transform. Leaving out either this `emit` or the header's `_eventOutput.on` will cause the header to not animate in/out (but you could create your own transitions here for the header).

```
    this._eventOutput.emit('inOutTransition', arguments);

    switch(direction){
        case 'hiding':
            switch(otherViewName){

                default:
```

We don't want to have any default transforms applied, so we tell the `App.MainController` to use `Transform.identity` for this PageView's transform.

```
                    // Overwriting and using default identity
                    transitionOptions.outTransform = Transform.identity;

                    // Content
                    Timer.setTimeout(function(){

                        // Bring content back
                        // that.layout.content.StateModifier.setTransform(Transform.translat

                    }, delayShowing);

                    break;
            }

            break;
        case 'showing':
            if(this._refreshData){
                // window.setTimeout(this.refreshData.bind(this), 1000);
            }
            this._refreshData = true;
            switch(otherViewName){

                default:

                    // No animation by default
                    transitionOptions.inTransform = Transform.identity;
```

Our content starts out inivisble, where `opacity == 0`.

```
                    that.layout.content.StateModifier.setOpacity(0);
```

We then wait for the other view to finish its transition (`delayShowing + transitionOptions.outTransition.duration`) before bringing up the Opacity.

```
                    Timer.setTimeout(function(){

                        // Bring content back
                        that.layout.content.StateModifier.setOpacity(1, transitionOptions.in

                    }, delayShowing +transitionOptions.outTransition.duration);


                    break;
            }
            break;
```

```
    }
```

`inOutTransition` returns our modified transitionOptions.

```
    return transitionOptions;
};
```

Welcome Screens

After Signup, you'll want a user to complete a series of basic tasks, to Welcome them to the service

- Say hello
- choose username
- ...

Addition Welcome Screens

Following the `welcome/xyz` pattern, you could add additional Welcome screens easily. Common examples include:

- Upload Profile Picture
- Add biography or other text
- Connect with Friends
- Basic feature walkthrough -

Profile View

This is a basic Profile View

- change profile picture
- change

Doubles as a Dashboard too, we can edit ourselves from here

- because we're using a promise, and getting our User._id, if nobobody else is specified

MainFooter will be shown, with 2 options:

- Profiles
- Inbox

Inbox

Inbox should show Messages that are intended for you. What determines who can send you messages?

- have no limits on who can send messages, for now
- have an algorithm that limits sending?
- only actived when somebody legit sends you a message?
    - requires that you get "invited" manually by somebody?
    - (put tape over the person's mouth until they've been sent a message by somebody who we validated?)

target_user_id = my_user_id

Wizard - Send Message

This shows the common patterns used in a Wizard. It isn't the most expedient way to send a message, but gets our example point across.

- MessageAdd (handles routing)
- MessageAddText
- MessageAddImage (optional)
- MessageConfirm
- View Message (and eraseUntilTag)

When going through a Wizard, there are a few extra things to keep in mind.

- before starting, `modifyLast` of the page you want to return to, or erase up until that `tag`.
- use `{history:false}` when calling the original `ThingAdd` route, because the Wizard will manage it's own history (and the first page viewed is

Structure

Our example wizard is going to be sending a message to a username, and will include (optional) text and media.

In most cases, you'd want to have more than one action per-screen, or combine all of this sending into a single page.

```
www/src/views/Message/
    Subviews/
    Inbox.js
    Add.js
    AddUsername.js
    AddText.js
    AddMedia.js
```

Launching Wizard

Before launching the wizard you need to create a `tag` in the history for you to return to after the Wizard completes, or to erase to (in case you are navigating somewhere else after going through the wizard).

We launch the wizard from our `inbox` route by tapping the button in the top-right.

```
    this.header._eventOutput.on('more',function(){
        App.history.modifyLast({
            tag: 'StartMessageAdd'
        });
        App.history.navigate('message/add',{history: false});
    });
```

Wizard Entry Point

Our `Add.js` acts like a router/controller for the wizard. On entry, we set

```
this.doNotShow = true;
```

so that router.js defaultRoute does not try and display this PageView.

We also create the `model` we'll save later.

```
this.model = new MessageModel.Message();
```

Create a hash to use for this wizard process

```
this.wizard_hash = CryptoJS.SHA3(new Date().toString());
```

This allows us to use `cache: true` in wizard routes, so that going back doesn't wipe whatever page you were just on.

Create paths / routes / defaults

```
    this.wizard_startTag = 'StartMessageAdd';
```

List all of the possible paths/pages involved in the Wizard.

We're using `username`, `text`, and `media` pages.

```
    this.wizardPaths = {

        'username': {
                cacheOptions: 'UsernameOptions',
                title: 'Username',
                route: 'message/add/username',
                summaryPath: 'username'
            },
        'text' : {
                cacheOptions: 'TextOptions',
                title: 'Text',
                route: 'message/add/text',
                summaryPath: 'text',
```

The `post` function is used by the `on_choose` function (below). It can be useful for dynamic routes. For example, if you leave the text blank, we'll show media, otherwise we don't.

```
                post: function(){
                    if(this.summary.text == ''){
                        // show Media
                        if(this.wizardRoute.indexOf('media') == -1){
```

The following use of the `splice` method is helpful for adding the route to visit:

```
                            this.wizardRoute.splice(this.wizard_current_route_index+1,0,'med
                        }
                    } else {
                        this.wizardRoute = _.without(this.wizardRoute,'media');
                    }

                }
            },
        'media' : {
                cacheOptions: 'MediaOptions',
                title: 'Media',
                route: 'message/add/media',
                summaryPath: 'media'
            }
    };
```

Now we define the path we'll take through our Route. Keep in mind that this is a dynamic list; we only splice in `media` if no `text` is set (see `post` above).

```
    this.wizardRoute = [
        'username',
        'text'
    ];
```

Set the starting position for the wizard (always `0`):

```
    this.wizard_current_route_index = 0;
    this.run_wizard();

}
```

run_wizard

```
PageView.prototype.run_wizard = function(){
```

If we've ended our wizard (the last step has completed), we call our `complete` function and gtfo.

```
    if(this.wizard_current_route_index >= this.wizardRoute.length){
        this.complete();
        return;
    }
```

Determine which route/page we're showing, and set up the `App.Cache.xyz` options.

You will notice that Step 1 uses this `App.Cache.xyz`, so make sure (at each step) that the `App.Cache.xyz` is correct!

```
    this.currentRouteName = this.wizardRoute[this.wizard_current_route_index];
    this.currentRoute = this.wizardPaths[this.currentRouteName];

    App.Cache[this.currentRoute.cacheOptions] = {
        on_choose: this.on_choose.bind(this),
        on_cancel: this.on_cancel.bind(this),
        title: this.currentRoute.title
    };
```

And then we navigate to the route

> Notice that we've appended `this.wizard_hash` to the route

```
    App.history.navigate(this.currentRoute.route + '/' + this.wizard_hash);
```

on_cancel

Canceling, or when the Back button is pressed while in a step, can result in either returning to the previous step, or will utilize `this.wizard_startTag` to completely exit the wizard.

```
PageView.prototype.on_cancel = function(){
    // Go backwards

    // first one, need to reset
    if(this.wizard_current_route_index == 0){
        App.history.backTo(this.wizard_startTag);
        return;
    }

    // Go back a page/step
    this.wizard_current_route_index -= 1;
```

Call the `run_wizard` function again

```
    this.run_wizard();
```

on_choose (after the page returns)

```
PageView.prototype.on_choose = function(data){
```

Update our `this.summary.xyz` for the page.

```
    this.summary[this.currentRoute.summaryPath] = data;
```

Call the `post` function (if it exists) for the page.

```
    if(this.currentRoute.post){
        this.currentRoute.post.apply(this);
    }
```

Let's go to the next page/step!

```
    this.wizard_current_route_index += 1;
    this.run_wizard();

};
```

Wizard Step 1 (username)

Choosing a username doesn't currently have any validation, until actually saving.

You can quickly improve the usefulness of this example by looking at the `User/Search` PageView and allowing the user to search for an existing username, instead of typing the whole thing out.

```
function PageView(options) {
    var that = this;
    View.apply(this, arguments);
    this.options = options;
```

See above (`App.Cache.xyz`) for how the object is created. If it doesn't exist, we just restart the app.

```
    if(!this.options.App.Cache.UsernameOptions){
        window.location = '';
        return;
    }

    this.options.passed = _.extend({}, App.Cache.UsernameOptions || {});
```

Create our layout, with header and content.

```
    this.layout = new HeaderFooterLayout({
        headerSize: 50,
        footerSize: 0
    });

    this.createContent();
    this.createHeader();

    this.add(this.layout);
} // end of "function PageView"
```

Header - going "Back"

If the "Back" button is pressed, always delegate to the passed `on_cancel` function (the wizard's "controller/router" function will handle Back events).

```
that.options.passed.on_cancel();
```

Content

The only difference between the content of this wizard step and most every other PageView with a form (for example, the `Signup` page) is that submitting the form simply returns the data to the wizard's controller/router.

```
that.options.passed.on_choose(that.inputUsernameSurface.getValue());
```

Wizard Step 2 (text)

Same as Step 1, simply adding `text` instead of a `username`.

Wizard Step 3 (media)

This is an ignored step, but you could modify it to include picture/video upload.

Saving (end of Wizard)

Saving occurs in the wizard's controller/router PageView, in the `complete` function.

```
PageView.prototype.complete = function(ev){
    var that = this;
```

Inform the user that we're trying to save the new message.

```
    Utils.Notification.Toast('Saving...');
```

Gather all your data from `this.summary` and update the model we previously created

```
    this.model.set({
        to_username: this.summary.username,
        text: this.summary.text
    });
```

and save

```
    this.model.save()
        .then(function(newModel){
```

Note that we're not really handling errors here, just assuming the save was successful.

```
            Utils.Notification.Toast('Message Created!');
            App.history.backTo('StartMessageAdd');

        });
};
```

Search Profiles

We'll be leaving out the actual user-to-user (connecting friends) mechanics, but lets add a way of finding and viewing other user's profiles.

Route

```
'users/search' : function(){
    defaultRoute('UsersSearch', 'User/Search', arguments, { cache: true });
},
```

Init

Totally standard.

```
function PageView(options) {
    var that = this;
    View.apply(this, arguments);
    this.options = options;

    // Models
    this.loadModels();

    // create the layout
    this.layout = new HeaderFooterLayout({
        headerSize: 50,
        footerSize: 0
    });

    this.createContent();
    this.createHeader();

    // Attach to render tree
    this.add(this.layout);

}
```

Content

The layout for our content is a simple `HeaderFooterLayout`. This helps us achieve a fixed SearchHeader and a scrollable ResultsBody, while still maintaining a consistent overall Header (with Back button) aesthetic.

Our SearchHeader is simply an `InputSurface`, where we also listen for `keyup` events to trigger searches.

```
this.SearchHeader = new InputSurface({
    type: 'text',
    size: [undefined, undefined],
    placeholder: 'username',
    name: 'search_username',
    value: ''
});
this.SearchHeader.on('keyup', function(){
    that.search_username();
});
```

search_username function

When triggerd by a `keyup` event on `this.SearchHeader`, we re-initialize the collection with the new (partial) username, and make a request to update with usernames that match.

We also prevent late-arriving (expired, superceded) username searches from affecting the collection.

```
var val = this.SearchHeader.getValue();
this.collection.initialize([],{
    type: 'username',
    username:  val
});
if(val == ''){
    this.collection.set([]);
    this.collection.trigger('reset');
} else {
    this.collection.fetch()
        .then(function(){
            if(val == that.SearchHeader.getValue()){
                that.collection.trigger('reset');
            }
        });
}
```

rebuild_username_list

Triggered by a `reset` on our collection.

Responsible for resorting all the views in our `ScrollView`. Instead of simply re-writing the whole display, we're deteriming which views to show/hide, and prevent unnecessary Surface/View creations.

addOne

Called by `rebuild_username_list` when a username's model does not yet have a View/Surface created in `this.contentScrollView .Views`.

Invite Friends

Spreading your app, whether among friends, coworkers, or random consumers, can be encouraged through the use of a multitude of methods.

- Find friends on the network (through address book)
- Send SMS with Link
- Post to Social Network

In this example you'll see how to invite somebody to use your app, by either posting on a social network, or sending an SMS.

Route

```
    'invite' : function(){
        defaultRoute('Invite', 'Misc/Invite', arguments, {cache: false});
    },
```

Initialization

Back to `www/views/Misc/Invite.js`, we initialize the PageView by applying all of a Famo.us `View`'s properties.

```
function PageView(options) {
    var that = this;
    View.apply(this, arguments);
    this.options = options;
```

We've already `var UserModel = require('models/user');`'d our model, now we create an empty User to hold our credentials to login with later.

```
    // Model
    this.model = new UserModel.User();
```

Add your background surface. Remember, order counts (but so does Transform position), so we use both this "put it on the RenderTree first" and add a Modifier to move it backwards in Z space.

```
    // Add background
    var bgSurface = new Surface({
        size: [undefined, undefined],
        classes: ['bg-surface']
    });
    var backMod = new StateModifier({
        transform: Transform.behind
    });
    this.add(backMod).add(bgSurface);
```

We'll use a simple SequentialLayout (link to Famo.us University, or links to our included resources?) here, as we only have a few fields and don't anticipate a need to scroll.

```
    // Create the layout
    this.layout = new SequentialLayout();
```

You'll notice this pattern used frequently:

```
    this.layout.Views = [];
```

> Notice that the SequentialLayout is going to sequenceFrom "itself" i.e. `this.layout.views` later

Now we add our surfaces, in order.

Each Surface is `.push`ed onto the `this.layout.Views` array.

```
    this.topWelcomeSurface = new Surface({
        content: "Nemesis",
        size: [undefined, 80],
        classes: ['login-page-welcome-top-default']
    });
    this.layout.Views.push(this.topWelcomeSurface);

    this.inputEmailSurface = new InputSurface({
        name: 'email',
        placeholder: 'Email Address',
        type: 'text',
        size: [undefined, 50],
        value: '' //nicholas.a.reed@gmail.com
    });
    this.layout.Views.push(this.inputEmailSurface);
```

This is one way of adding spacers: simply add a blank Surface with whatever size you need. If this were a ScrollView though, you'd also need to `.pipe` the Surface in order to avoid any weird un-scrollable patches on the screen.

```
    this.spacer1 = new Surface({
        content: "",
        size: [undefined,4]
```

```
    });
    this.layout.Views.push(this.spacer1);
```

Here you'll see another method of adding a spacer after the `inputPasswordSurface`. In this instance, we add a `StateModifier` with a slightly larger `size` that the contained Surface.

```
    this.inputPasswordView = new View();
    this.inputPasswordView.Surface = new InputSurface({
        name: 'password',
        placeholder: 'Password',
        type: 'password',
        size: [undefined, 50],
        value: '' //testtest
    });
    this.inputPasswordView.PaddingMod = new StateModifier({
        size: [undefined, 54]
    });
    this.inputPasswordView.add(this.inputPasswordView.PaddingMod).add(this.inputPasswordView
    this.layout.Views.push(this.inputPasswordView);
```

Our Login button will trigger the `login` function.

```
    this.submitSurface = new Surface({
        size: [undefined,60],
        classes: ['form-button-submit-default'],
        content: 'Login'
    });
    this.submitSurface.on('click', this.login.bind(this));
    this.layout.Views.push(this.submitSurface);


    this.spacerSurface = new Surface({
        content: "",
        size: [undefined, 20]
    });
    this.layout.Views.push(this.spacerSurface);
```

Our Signup and Forgot Password links are styled differently.

```
    this.signupLinkSurface = new Surface({
        content: 'Signup &gt;',
        size: [undefined,40],
        classes: [],
        properties: {
            color: "black",
            lineHeight: "40px",
            textAlign: "right"
        }
    });
    this.signupLinkSurface.on('click', function(){
        App.history.navigate('signup', {trigger: true});
    });
    this.layout.Views.push(this.signupLinkSurface);


    this.forgotLinkSurface = new Surface({
        content: 'Forgot Password &gt;',
        size: [undefined,40],
        classes: [],
        properties: {
            color: "black",
            lineHeight: "40px",
            textAlign: "right"
        }
    });
    this.forgotLinkSurface.on('click', function(){
        App.history.navigate('forgot', {trigger: true});
    });
    this.layout.Views.push(this.forgotLinkSurface);
```

Now that all our Surfaces (and one View!) are created and added to the array, we'll `sequenceFrom`.

```
    this.layout.sequenceFrom(this.layout.Views);
```

Finally, we created an `originMod` and `sizeMod` to get everything centered and sized correctly. The last step is adding to `this` PageView.

```
    var originMod = new StateModifier({
        origin: [0.5, 0.5]
    });
    var sizeMod = new StateModifier({
        size: [window.innerWidth - 16, undefined]
    });

    this.add(sizeMod).add(originMod).add(this.layout);

}
```

Login function

When the Login button is pressed, we need to:

- gather the inputs
- validate them slightly (no sense making a user wait for a "hey, you didn't enter a password" response from the server)
- check against the server
- store the token if the login succeeded
- redirect to their homepage

```
PageView.prototype.login = function(){
    var that = this;

    if(this.checking === true){
        return;
    }
    this.checking = true;

    var email = this.inputEmailSurface.getValue(),
        password = this.inputPasswordSurface.getValue();
```

After clicking, we'll disable the input for a moment, and inform the user.

```
    this.submitSurface.setContent('Please wait...');
```

Set the data for the login request.

```
    var body = {
        email: email,
        password: password
    }
```

Make the request using the `login` method on our `User` model.

```
    this.model.login(body)
```

If the login fails, inform the user and reset the submit button text.

```
    .fail(function(){
        alert('Failed logging in');
        that.submitSurface.setContent('Login');
        that.checking = false;

    })
```

> We check for a failure condition with the response from the server too

```
    .then(function(response){
        if(response.code != 200){
            alert('Failed signing in (3424)');
            that.submitSurface.setContent('Login');
            that.checking = false;
            return;
        }
```

When the login succeeds, we save the `token` and preload all the models for the new User.

```
        localStorage.setItem('usertoken_v1_',response.token);
        App.Data.UserToken = response.token;

        that.model.fetch({
            error: function(){
                alert("Failed gathering user model");
            },
            success: function(userModel){
                console.log('UserModel');
```

```
                console.log(userModel);
```

You can access the User model anytime at `App.Data.User`.

```
                that.options.App.Data.User = userModel;

                localStorage.setItem('user_v3_',JSON.stringify(userModel.toJSON()));


Preload the user's models (friends, Posts, etc.)

                require(['models/_preload'], function(PreloadModels){
                    PreloadModels(that.options.App);
                });
```

We need to register for Push Notifications too.

```
                // Register for Push Notifications
                App.DeviceReady.initPush();
```

Finally, use `eraseUntilTag` to erase the whole history (because no tags called `allofem` are created, of course) and then navigate to our homepage.

```
                // Reload home
                App.history.eraseUntilTag('allofem');
                App.history.navigate('dash');

            }
        });

    });

};
```

Transitions

Left out of our Login PageView is an inOutTransition function. Instead, we use a simple StoredTransition and ViewToView for displaying the Login page.

More Views

We've provided a few working sample views, now you'll want to use your knowledge of Famo.us, combined with the sample code provided, to build your more custom views (dashboards, friend connecting, full onboarding flows, etc.).

Debugging

basic debugging Patterns

- serve using heroku's `foreman` and the handy `ngrok`.
- using Chrome console and localhost -

Android: Eclipse debugging for native problems

Install Eclipse and check the log (limit to `tag:chromium`) to detect native-level errors (contacts SUCK).

Setting up foreman and ngrok

```
heroku create myapp
foreman start
./ngrok 5000 my-subdomain-choice
```

Phonegap build links (direct download for Android)

> This only works for Android, you have to use TestFlight for iOS

To easily distribute your app to folks on Android, simply copy the links on PhoneGap Build. They should include a `qr_code` string.

Server (node.js)

You can build a server in any language. Included with **Waiting** is a sample server in Node.js.

GitHub repository

https://github.com/nicholasareed/waitingapp_nodeserver

Cloning and Building Heroku app

```
git clone git@github.com:nicholasareed/waitingapp_nodeserver.git
cd waitingapp_nodeserver
mv config_example.json config.json
heroku create
```

You will need to create a MongoDB database and add the connection string to `config.json` (or as an environment URL). I've been using https://mongolab.com/.

Update any other variables in `config.json`.

Then run foreman:

```
foreman start
```

Visit at `http://localhost:5000/` for the sample homepage.

Push Notifications

Android and iOS use different systems for Push Notifications.

- more details forthcoming.

Production

---

stub

Putting your app in the store:

- android
- ios

App Icons: http://makeappicon.com/

- photoshop download > batch script

Android

stub

Go over:

- production certs for Push Notifications
- resources to create (icons, splash screens, etc.)

iOS

stub

Go over:

- setup TestFlight
- adding people, getting device ids
- pay Apple
- mobile provisioning profiles and devices
- adding p12 and whatnot to Build
- adding more devices, repeat steps d-z
- resources to create (icons, splash screens, etc.)

Table of Contents