# GETTING STARTED

A hands-on guide to creating your first Ionic app.

Richard Ramdat
rramdat@infusion.com

# Contents

# Part I: Getting Started

## 1. Installation

a. Install Node.js
   https://nodejs.org/download/
b. Install Ionic
   Open a command line as administrator and run the command below:
   ```
   npm install ionic -g
   ```
c. Install Cordova
   Open a command line as administrator and run the command below:
   ```
   npm install cordova -g
   ```

**Note:** The -g option will install an NPM module globally so you can access it from the command line regardless of your current location.

## 2. Creating your first project

Let's get started by creating a quick app based on one of the provided Ionic starter templates.

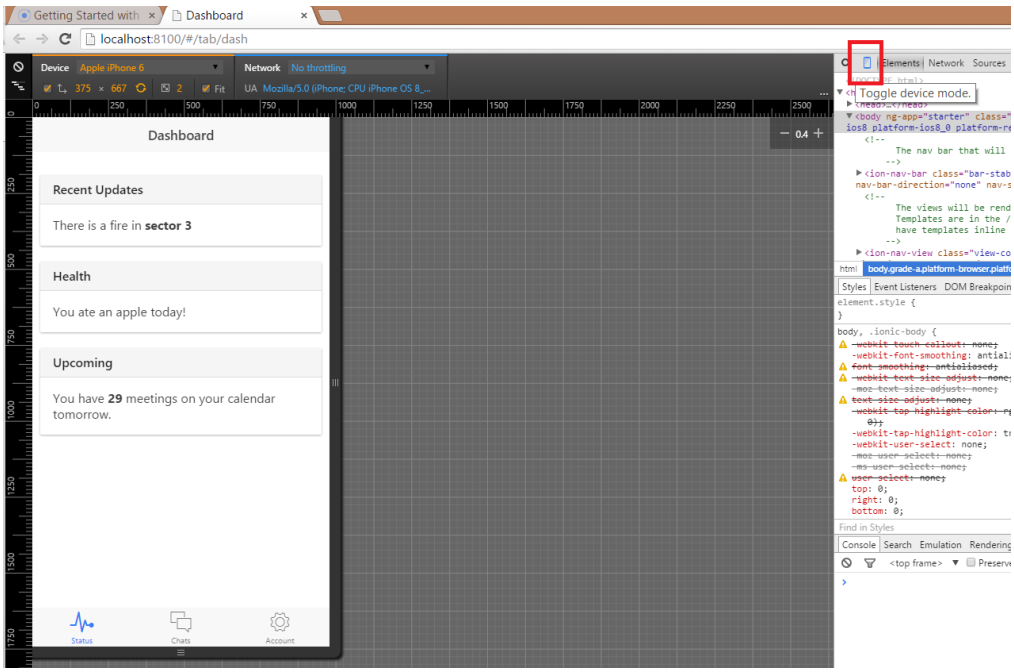
a. Open a command-line window and run the following:
   ```
   ionic start myApp tabs
   ```
b. This will create a new folder **myApp** within the current directory. Change into this directory:
   ```
   cd myApp
   ```
c. Setup your application to use Sass (CSS pre-processor):
   ```
   ionic setup sass
   ```
d. Add a platform (either iOS, Android, or both) to your application:
   ```
   ionic platform add ios
   ```

**Note:** OS X is required to run either the ionic **build, emulate, or run** commands for iOS.

e. Test your application locally:
   ```
   ionic serve
   ```

   This will launch your browser to view the default Ionic app that you just created. Switching to the command-line, you will notice that it is running a local server and watching for any changes within the application directory. It will re-sync the browser to update any changes that you make automatically.

**Note:** If you are using Google Chrome, access the developer tools by pressing F12, or using the Ctrl + Shift + I shortcut. Once open, use the 'Toggle device mode' option (shown below) to change the user-agent string and re-configure the viewport to an appropriate mobile-device resolution.




f. Make a change to the default application by editing the following file in your current application directory. Using a new command line window:

```
notepad www/templates/tab-dash.html
```

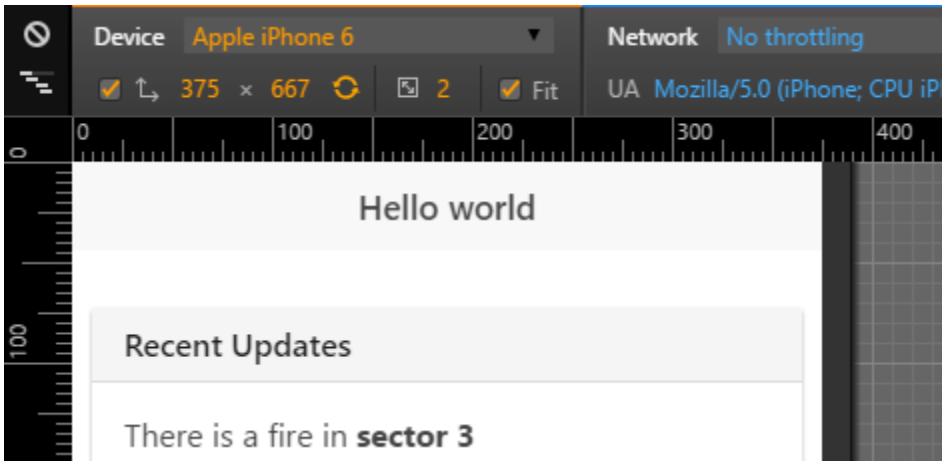
Replace:

```
<ion-view view-title="Dashboard">
```

With:

```
<ion-view view-title="Hello world">
```

Save the current file (Ctrl + S). You should notice your browser window refresh to display the following:



And your original command line window should display the following:

```
ionic $ HTML changed: myApp1\www\templates\tab-dash.html
```

**Note:** You can use any text editor you prefer. Atom (https://atom.io/) is a great open-source editor with available plug-ins that are well-suited for front-end development.

## 3. Testing with Ionic View

If you have an iOS or Android device, you can test your changes using the Ionic View companion app. To get started, download the app using the links below:

- Android: https://play.google.com/store/apps/details?id=com.ionic.viewapp
- Apple: https://itunes.apple.com/us/app/ionic-view/id849930087?ls=1&mt=8

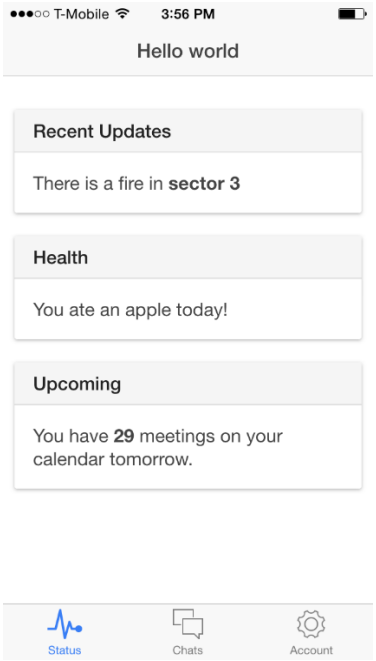
Next, register for an Ionic developer account:

- Create a new account: https://apps.ionic.io/signup

Once completed, test your changes by publishing them to Ionic:

```
ionic upload
```

and then logging into Ionic View on your device and selecting to run the available app.




Congratulations, you're up and running with Ionic and just created your first application. In the next section, we'll build upon the starter application to add custom functionality.

# Part II: Adding UI Elements, Views and Components

## 1. Add a new tab and view

Our starter template comes with three tabs, let's add a fourth.

a. Fire up your preferred text editor. Open the file **www/templates/tabs.html** and make the following changes from lines 23 to 26.

```
16    </ion-tab>
17
18    <!-- Account Tab -->
19    <ion-tab title="Account" icon-off="ion-ios-gear-outline" icon-on="ion-ios-gear" href="#/tab/account">
20      <ion-nav-view name="tab-account"></ion-nav-view>
21    </ion-tab>
22
23    <!-- Playlist Tab -->
24    <ion-tab title="Playlist" icon-off="ion-ios-shuffle" icon-on="ion-ios-shuffle-strong" href="#/tab/playlist">
25      <ion-nav-view name="tab-playlist"></ion-nav-view>
26    </ion-tab>
27
28
29  </ion-tabs>
30
```

This will place an additional tab on the strip located near the footer of the app. We'll choose the shuffle icon for our Playlist tab.

**Note:** You can find additional tabs from the site below:

- Ionic Icons: http://ionicons.com/

b. Create a new file: **www/templates/tab-playlist.html** This will be the view that is presented once the user clicks the Playlist tab. For now, we'll keep it simple and enter the following:

```
1   <ion-view view-title="Playlist">
2     <ion-content class="padding">
3       <div class="list card">
4         <div class="item item-divider">Item 1</div>
5         <div class="item item-body">
6           <div>
7             Artist 1
8           </div>
9         </div>
10      </div>
11      <div class="list card">
12        <div class="item item-divider">Item 2</div>
13        <div class="item item-body">
14          <div>
15            Artist 2
16          </div>
17        </div>
18      </div>
19      <div class="list card">
20        <div class="item item-divider">Item 3</div>
21        <div class="item item-body">
22          <div>
23            Artist 3
24          </div>
25        </div>
26      </div>
27    </ion-content>
28  </ion-view>
```

c. We now need to wire up our new tab button and its corresponding view together. To do so, edit the following file: **www/js/app.js**

Line 26 onwards details routes for our application. Ionic leverages Angular's UI-router, which allows for the configuration of different UI states to URLs, views and corresponding controllers for functionality. Let's add a state for our new tab:

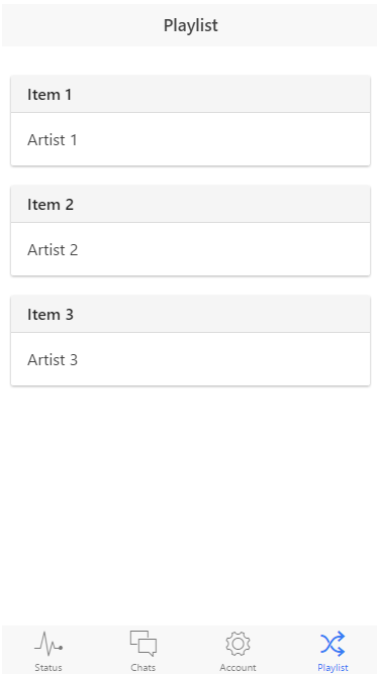
```
39  // Each tab has its own nav history stack:
40
41  .state('tab.playlist', {
42    url: '/playlist',
43    views: {
44      'tab-playlist': {
45        templateUrl: 'templates/tab-playlist.html',
46        controller: 'PlaylistCtrl'
47      }
48    }
49  })
```

d. Our route will call into a controller **PlaylistCtrl**, let's go ahead and create this. Edit the following file: **www/js/controllers.js**

```
5  .controller('PlaylistCtrl', function($scope) {})
```

Our controller currently doesn't do much, but we'll revisit it in future steps to add additional functionality.

e. At this point we should be all set. Ensure all files are saved. Ionic should reload your changes automatically if you still have it running – otherwise, run `ionic serve` and switch to your browser:



## 2. Binding static data

Now that we've added a new view and hooked it up to our tab menu, let's bind some data to the view. For now, we'll keep it simple and just use static data.

a. Edit the **tab-playlist.html** view as follows:

```
1  <ion-view view-title="Playlist">
2    <ion-content class="padding">
3      <div class="list">
4        <a ng-repeat="item in items"
5           href="#/{{item.id}}"
6           class="item item-thumbnail-left">
7          <img ng-src="{{ item.image }}">
8          <h2>{{ item.album }}</h2>
9          <h4>{{ item.artist }}</h4>
10         <span class="badge badge-assertive">0</span>
11       </a>
12     </div>
13   </ion-content>
14 </ion-view>
```

**Note:** Pay close attention to the use of the ng-repeat directive on line 4. This will allow us to bind a collection of data to this anchor element to create the list view we need for our playlist. For now we'll use the Item Thumbnail list view that Ionic provides, for more information about Ionic's List component and other views, read more.

On line 10 we make use of a badge element provided by Ionic to call out a counter that we'll use later.

b. Edit the **PlaylistCtrl** in **controllers.js**:

```
5  .controller('PlaylistCtrl', function($scope) {
6    $scope.items = [
7      { id: 1, album: 'Gotta Be Somebody', artist: 'Nickelback', image: 'data:image/
8      { id: 2, album: 'Dark Horse', artist: 'Nickelback', image: 'data:image/jpeg;ba
9      { id: 3, album: 'Someday', artist: 'Nickelback', image: 'data:image/jpeg;base6
10     { id: 4, album: 'All The Right Reasons', artist: 'Nickelback', image: 'data:im
11   ];
12 })
```

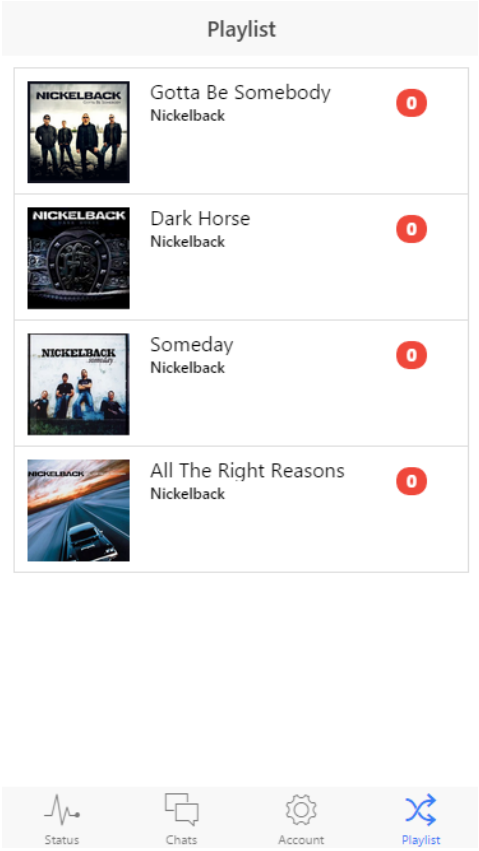
You can copy and paste the above from the following snippet:

- https://gist.github.com/rramdat-i/0ace7912610732a287f0

**Note:** The snippet above makes use of the data-uri format which allows image data for small resources to be directly embedded in HTML, JS, and CSS resources, eliminating the need for a separate HTTP request.

The **$scope** Angular variable above allows shared data from the controller to the view. We use it as a mount-point for the static data we want to bind in the Playlist tab using our ng-repeat template.

c. Save the files above and switch back to your browser, you should see the following displayed:




## 3. Binding live data

Now that we have a collection of data added to our Playlist view, let's take it a step further by adding live-data from an API.

a. We'll use Angular's Service Factory provider to define a service for our Playlist that will centralize the coordination of data that we need for our controller and view. To begin, edit the file **www/js/services.js**:

```
.factory('PlaylistService', function($http) {
  return {
    getAll: function () {
      return $http.get("http://trancefusionapp.azurewebsites.net/api/lobby/rooms");
    }
  };
})
```

API endpoint: http://trancefusionapp.azurewebsites.net/api/lobby/rooms

This creates a new factory named **PlaylistService** that will service requests to fetch data from the above REST endpoint.

**Note:** The above code utilizes Angular's **$http** module which is useful for making HTTP requests. Note that this service returns a promise, with two methods: success and error, depending on the outcome of the request to the specified resource. Read more about **$http** by referring to the documentation.

b. Now let's leverage our newly created service from our **PlaylistCtrl**. Edit the file: **www/js/controllers.js**

```
5  .controller('PlaylistCtrl', function($scope, PlaylistService) {
6      var promise = PlaylistService.getAll();
7
8      promise.success(function (response) {
9        $scope.items = response;
10     });
11 })
```

**Note:** Pay attention to the **PlaylistService** parameter added to our controller method, this indicates to Angular that it will need to wire-up a reference to our service for use within our controller.

Within the controller, we take-over the promise that has been initiated by the service and indicate that when successful, the items from the API should be assigned to the scope which then will be available to our view.

c. Let's update our Playlist view to work with the new data that we have. Update the file **tab-playlist.html**:

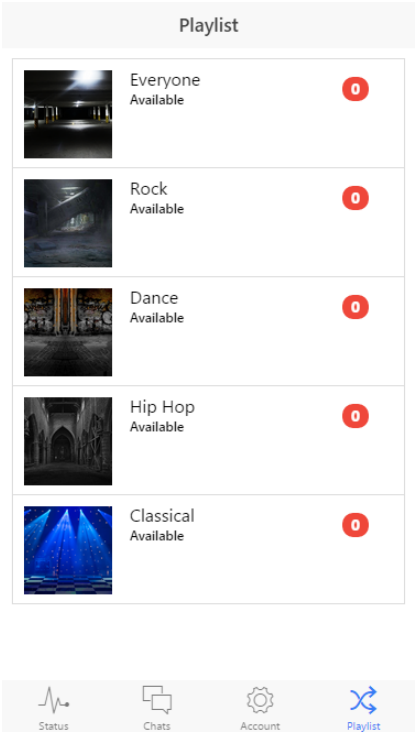
```
1  <ion-view view-title="Playlist">
2    <ion-content class="padding">
3      <div class="list">
4        <a ng-repeat="item in items"
5           data-uri="#"
6           href="javascript:void(0);"
7           class="item item-thumbnail-left">
8
9          <img ng-src="http://trancefusionapp.azurewebsites.net/{{ item.thumbnail }}">
10         <h2>{{ item.name }}</h2>
11         <h4>{{ item.full ? 'Room full' : 'Available' }}</h4>
12         <span class="badge badge-assertive">{{ item.count }}</span>
13       </a>
14     </div>
15   </ion-content>
16 </ion-view>
```

**Note:** We don't have an item identifier, so we set our href property on line 6 to void. This is different than setting it to a hash #, as that would reset our application to the initial view by modifying the current url.

On line 9, we prepend a static URL to our data-binding to form the absolute URL needed to correctly load our images.

One line 11, we use a ternary operator to map our true/false value to more meaningful strings.

d. Now, let's save our changes, and switch back to the browser:




## 4. Implementing pull-to-refresh

Currently, the only way users can see updated information returned from our API is to switch tabs. Let's give them a slightly better way to refresh our Playlist view by implementing the common pull-to-refresh technique.

a. Update the file **tab-playlist.html**:

```
1  <ion-view view-title="Playlist">
2    <ion-content class="padding">
3      <ion-refresher pulling-text="Pull to refresh" on-refresh="doRefresh()"></ion-refresher>
4      <div class="list">
5        <a ng-repeat="item in items"
```

Add the ion-refresher component shown on line 3.

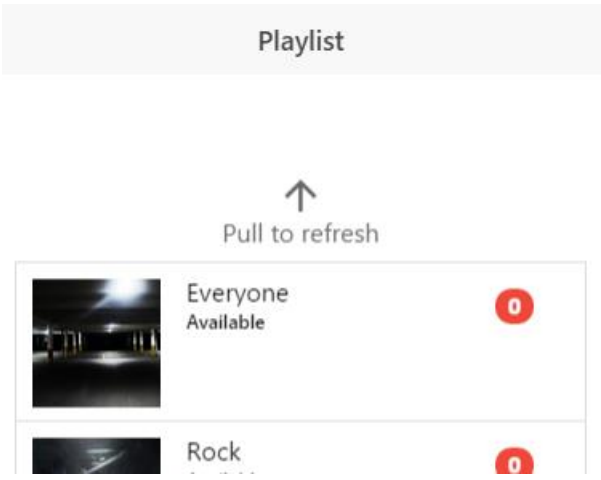
b. Note that the ion-refresher invokes an **doRefresh()** function as specified by on-refresh attribute. Let's add this function in our **PlaylistCtrl**. Edit the file **controllers.js:**

```
5  .controller('PlaylistCtrl', function($scope, PlaylistService) {
6    var promise = PlaylistService.getAll();
7
8    promise.success(function (response) {
9      $scope.items = response;
10   });
11
12   $scope.doRefresh = function() {
13     var refreshPromise = PlaylistService.getAll();
14
15     refreshPromise
16     .success(function(response) {
17       $scope.items = response;
18       $scope.$apply();
19       $scope.$broadcast('scroll.refreshComplete');
20     })
21     .error(function (error) {
22       $scope.$broadcast('scroll.refreshComplete');
23     });
24   };
25 })
```

**Note:** On line 12 we create our function **doRefresh** that is referenced from the Playlist view. Notice that we create a new promise specific to our refresh. If it is successful, we do a couple things:

1. Update the items declared on our **$scope** context.
2. Force Angular to be aware that we have made changes (line 18)
3. Broadcast that the refresh is now complete. Note this is needed so the UI can know to change back to the non-refreshing state. If there's an error with the promise, we also notify the UI so the user can continue using the app.

c. Now let's save our changes, and try it out:



## 5. Adding a header button

Our tab option is great for a simple shortcut, but header buttons are also another option we can use. Let's take a look at how we can implement one in Ionic.

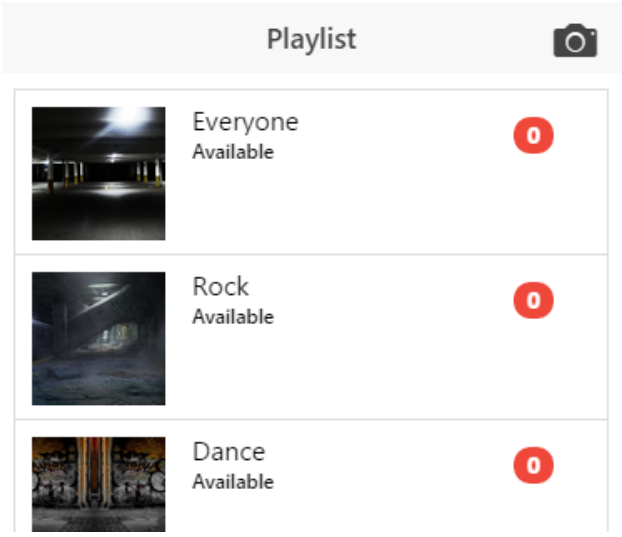
a. Edit the file **www/index.html**

```
26  <body ng-app="starter">
27    <!--
28      The nav bar that will be updated as we navigate between views.
29    -->
30    <ion-nav-bar class="bar-stable">
31      <ion-nav-back-button>
32      </ion-nav-back-button>
33      <ion-nav-buttons side="secondary">
34        <a class="button button-icon icon ion-camera" ng-click="takeSelfie()"></a>
35      </ion-nav-buttons>
36    </ion-nav-bar>
37    <!--
38      The views will be rendered in the <ion-nav-view> directive below
```

**Note:** Lines 33-35 defines a button group with a single icon button. We specify the button group as 'secondary' which depending on iOS or Android will affect the placement of the button within the header region.

For more information about Ionic's navigation component, refer to:
http://ionicframework.com/docs/api/directive/ionNavButtons/

b. Save the **index.html** file and switch to your browser to view the updated changes:




Congratulations, you've completed the second part of this Ionic walkthrough, adding and customizing the user interface of your app.

## Part III: Using the Device APIs

An important aspect of building a mobile application is interfacing with the actual device hardware such as the camera, GPS, and other sensors. Let's take a look at how we can wire up our header button from the previous step to the device camera.

### 1. Using the Cordova Camera API

a. Install the Cordova plugin. From a command-line in your current application directory, run the following command:

```
cordova plugin add org.apache.cordova.camera
```

b. Just as we did for our API endpoint, let's create a service that can broker requests from our controllers to the appropriate device API.

Edit the file **www/js/services.js**:

```
11  .factory('Camera', ['$q', function($q) {
12    return {
13      getPicture: function(options) {
14        var q = $q.defer();
15
16        navigator.camera.getPicture(function(result) {
17          q.resolve(result);
18        }, function(err) {
19          q.reject(err);
20        }, options);
21
22        return q.promise;
23      }
24    }
25  }])
```

**Note:** We use Angular's promise service **$q** to asynchronously resolve the result or any error from the Camera API. This promise is what we will use from our controller.

c. Let's edit **www/js/ controllers.js** to create a new controller to wire up the event that our navigation button is expecting to call:

```
3   .controller('AppCtrl', function($scope, Camera) {
4     $scope.takeSelfie = function() {
5         Camera.getPicture().then(function(imageURI) {
6           console.log(imageURI);
7         }, function(err) {
8           console.err(err);
9         });
10      };
11  })
```

**Note:** We call our new controller **AppCtrl** to indicate that it serves as a default controller for our Ionic app.

Pay attention to the **Camera** parameter that is passed to our controller method, this allows us to access the service we created in the previous step.

For this example, we simply log the file URI of the image taken by the device camera. You can easily use some imagination to do other things with the image, such as setting the background image of a view.

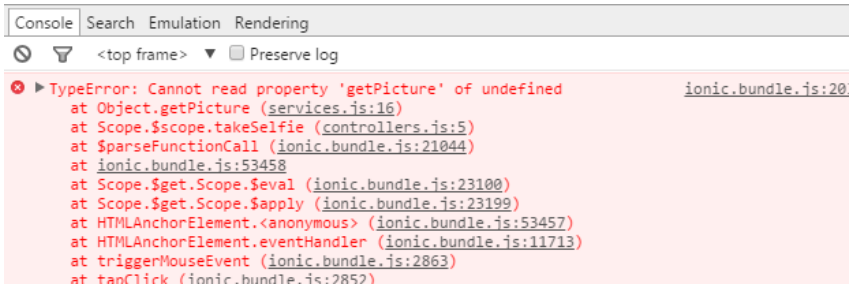
**Note:** You may be required to whitelist the file:// URI protocol to allow Angular to serve a local file. For example in **www/js/app.js**:

```
module.config(function($compileProvider){
  $compileProvider.imgSrcSanitizationWhitelist(/^\s*(https?|ftp|mailto|file|tel):/);
})
```

d. We now need to link our new controller to the default view loaded thru **index.html**. Previously we linked controllers to our view thru routing, in this example, we'll use the **ng-controller** attribute to do the same. Edit the file **www/index.html**:

```
26    <body ng-app="starter" ng-controller="AppCtrl">
27      <!--
28        The nav bar that will be updated as we naviga
```

e. Save your files, and attempt to view your changes in the browser. Click on the Camera button and notice that your dreams have been shattered:



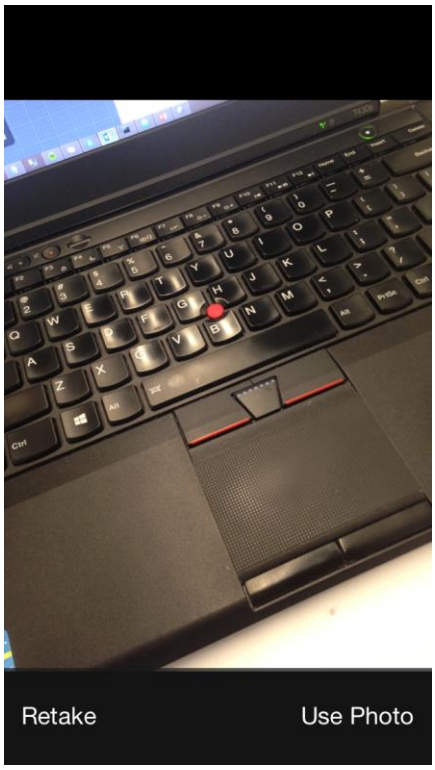

This happens as result of the Cordova Camera plug-in being unable to find a suitable device API for the camera. Makes sense as we're running this from a desktop browser. Let's use the Ionic View app to test our changes instead. Run the following command from the root of your application directory:

```
ionic upload
```

Open the Ionic View app from your mobile device, and load your mobile application.

**Note:** You may need to choose to Clear App Data if you've previously loaded your app before. Once your app loads, click on the Camera icon.

If all works right, you should be prompted to allow access to the camera and be able to take a photo (or a selfie, if that's your thing):




Congratulation, you've completed the third part of this walkthrough and successfully integrated device functionality within your app.

## Part IV: Changing the Splash Screen and App Icon

The app icon and launch image (splash screen) are important aspects of your app and one of the first things that your users will see. Managing these assets can be surprisingly tedious, requiring different file sizes and appropriate naming conventions to be maintained. Luckily Ionic has tools to help with this.

### 1. Using Ionic Resources

a. Create a directory named **resources** from within the root of your main application directory.
   **Note:** This directory may already exist.
b. Within this directory add two files:
   i. icon.png – minimum dimension 192x192px
   ii. splash.png – minimum dimension 2208x2208px

   **Note:** Within **resources** you can create two additional sub-folders (**android** and **ios**) to place platform-specific versions of the above files in. If you don't have different versions of these files, just toss them within resources.

   **Tip:** For the purposes of this walk-through, you can use an online photo editor (http://apps.pixlr.com/editor/) to quickly create sample PNGs of the required dimensions.
c. From a command-line in your application root folder, run the following:
   `ionic resources`

## 2. Running your app on the iOS simulator

To view our app icon and splash screen, we must run our app from a simulator or physical device. We'll choose to use the iOS simulator on a Mac running Xcode 6. You can also use the Android SDK to build and run your app on the Android device emulator or Genymotion.

**Note:** Before getting started, if you are switching to a new machine, you must re-run the steps above so Ionic can properly re-reference your app resources.

a. From your Mac, launch Terminal and navigate to your application directory. Once there, run the following command:

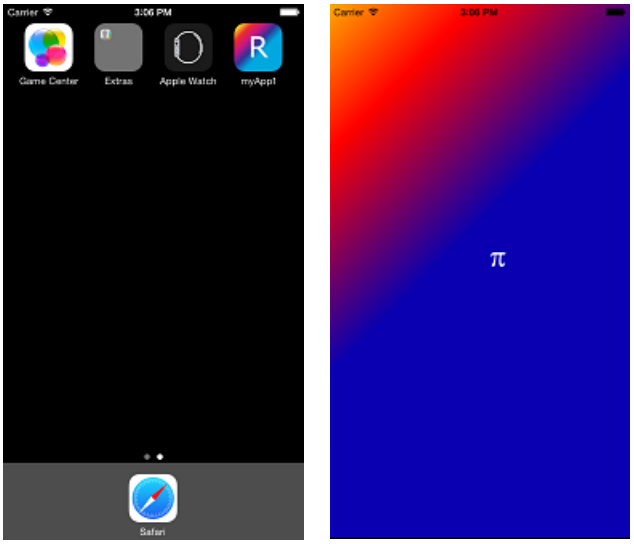
```
ionic build ios
```

b. Next, run

```
ionic emulate ios
```

**Note:** You may be prompted to install additional NPM modules when running the above commands. Follow the instructions as prompted.

If all is successful, the iOS simulator will run your app, allowing you to check out your custom app icon and splash screen:




# Part V: Wrapping Up

You should be feeling pretty accomplished at this point. Not only have you created an Ionic app from the ground up, you had the chance to see first-hand some core Angular principles in action. Your learning continues with the links below:

- Ionic Docs: http://ionicframework.com/docs/
- Videos, Formulas, and Talks: http://learn.ionicframework.com/
- Ionic on CodePen: http://codepen.io/ionic/public-list/
- Andrew McGivery's Blog: http://mcgivery.com/
- Source code for this walk-through: https://github.com/rramdat-i/myApp