**CityScape Ionic App**

**Members**: Ben Holmes, Timothy Rogers

**Description**: This application was built using AngularJS and Ionic. It follows the John Papa style guidelines to the best of our abilities, and is in MVC format. You will find each page has it’s own directories for Controllers, Directives, Services, and Views.

**Note**: We would like you to keep in mind that our application was built using text files to store the users and images. We could have just as easily used a SQL database, and our php files actually include the commented out code for interacting with the database, but we decided to use the txt files to make it easier for anyone to use and review, without having to go through the trouble of creating a database.

Table of Contents

CODING 3

SNIPPET 1 3

SNIPPET 2 3

SNIPPET 3 4

SNIPPET 4 4

SNIPPET 5 5

SNIPPET 6 5

Overview Page 6

New and Complex Issues 6

Work Towards Bloom’s Taxonomy 7

New And Complex Section 10

New Things 10

Complex Things 12

Bloom’s Taxonomy Section 15

# 

# CODING

## SNIPPET 1

(**function**(){  
 **'use strict'**;  
  
 myApp.controller(**'homeCtrl'**, *homeCtrl*);  
  
 *homeCtrl*.**$inject** = [**'$state'**, **'logOutService'**];  
  
 **function** *homeCtrl*($state, $logOutService)  
 {  
 //ALL CONTROLLER FUNCTIONALITY GOES HERE  
 }  
  
})();

## SNIPPET 2

<**body**>  
  
 <**ion-nav-view**></**ion-nav-view**>  
  
</**body**>

## SNIPPET 3

<**ion-side-menu side="left" class="side-menu" slide-along offset="-15"**>  
 <**ion-header-bar class="bar-royal"**>  
 <**h1 class="title"**>Navigation</**h1**>  
 </**ion-header-bar**>  
 <**ion-content**></**ion-content**>  
</**ion-side-menu**>

## SNIPPET 4

<**ion-side-menus enable-menu-with-back-views="false"**>  
  
 <**ion-side-menu-content**>  
 <**ion-nav-bar class="bar-positive"**>  
 <**ion-nav-back-button**>  
 </**ion-nav-back-button**>  
  
 <**ion-nav-buttons side="left"**>  
 <**button class="button button-icon button-clear ion-navicon" menu-toggle="left"**></**button**>  
 </**ion-nav-buttons**>  
  
 <**ion-nav-buttons side="right"**>  
 <**a href = "#/menu/home" class="button button-icon button-clear ion-home"**></**a**>  
 </**ion-nav-buttons**>  
  
 </**ion-nav-bar**>  
  
 <**ion-nav-view name="menuContent"**></**ion-nav-view**>  
 </**ion-side-menu-content**>  
  
 <**ion-side-menu side="left" class="side-menu" slide-along offset="-15"**>  
 <**ion-header-bar class="bar-royal"**>  
 <**h1 class="title"**>Navigation</**h1**>  
 </**ion-header-bar**>  
 <**ion-content**>  
  
 </**ion-content**>  
 </**ion-side-menu**>  
</**ion-side-menus**>

## SNIPPET 5

**var** reader = **new FileReader**();  
reader.**onload** = **function**(loadEvent) {  
 scope.$apply(**function**() {  
 scope.vm.**uploadMe** = reader.**result**;  
 });  
}  
reader.readAsDataURL(changeEvent.**target**.**files**[0]);

## SNIPPET 6

(**function**(){  
 **'use strict'**;  
  
 myApp.controller(**'homeCtrl'**, *homeCtrl*);  
  
 *homeCtrl*.**$inject** = [**'$state'**, **'logOutService'**];  
  
 **function** *homeCtrl*($state, $logOutService)  
 {  
 var vm = this;

vm.doSomething = doSomething;

function doSomething(){

//Content in here, but you see It reads way better than

//$scope.doSomething = function(){};

}

}  
  
})();

# Overview Page

## New and Complex Issues

There were two huge reoccurring difficulties in building this app. First was learning how to use Ionic, and its markup. The second was following John Papa’s style guide. There were many times when we knew what we wanted to do, and how to do it, but because we were following John Papa, the implementation took much longer than it should have.

Some new implementations we have are the use of the UI Router in angular, which is considered to be better and more robust than the NgRouter. This seems to be because the UI Router allows nested views, meaning a view within a view, which we implemented for literally every view. You will find the main view is menu, which displays our cool menu bar, and then every other view is nested below that menu bar.

Another new thing was creating a directive. You will find on the Post page, we created a directive, which takes the content of an uploaded image, and shows it inside the directive. This was added so that users could view the image they are uploading.

We also added Services, and constants, to follow the John Papa guidelines. A service is basically the part of an application, which communicates with a database, or in our case a text file. The service is injected into the controller, and then the services methods can be accessed from within the controller. The service basically just returns an $http.post() reference. A constant is an angular object, which just contains references that never change. In our case it was relative file paths to the PHP pages our services were calling.

Some random cool things we implemented which were also new and probably more complex to us than they should have been are a favicon, Ionic loading bar which displays while services are making calls to the database, element.bind, fileReader for reading base64 encoded photos, and a scope.$apply() inside our post directive.

## Work Towards Bloom’s Taxonomy

-Creation

Towards the creation, it was very important to us to follow John Papa’s angular guide. This meant a lot of reading, every single time we started something new. We wanted our code to be easy to follow, so we made sure all the views, had corresponding controllers, but beyond that we also used the vm (view module) syntax, which makes it super easy to distinguish between functions inside a service and those inside a controller. A view module is the controller, so everytime you are looking through an html page, and you find vm.function(), you know that you can find that function inside the controller of that html page. This way there is no confusion.

We also wanted to avoid rewriting code between pages. This meant many small functions, and also using an index.html page. The index page contains all the links and scripts that we use throughout the app. Beyond the index page, the use of the UI Router’s nested views really helped us a lot in this aspect. Because we needed to reuse the html content of the menu on every view, we created a state for the menu inside the app.js file, and then we allowed all subsequent pages to just be nested views from that menu. This meant the menu was displaying all the time, and inside of it was an <ion-nav-view> tag which would be populated with the templateURL specified in the app.js for each specific view.

-Evaluation

Many of the ideas we had in regards to this application came through reading John Papa’s guide. This helped a lot when it came to knowing we are making the correct decisions. There wasn’t really any debate necessary.

The parts where John Papa didn’t help included the html markup, and site layout. One thing we didn’t want was for the menu to display on the login and signup pages. We accomplished this by making the login and signup pages the only ones to not be nested inside the menu page.

We also had to decide what our overall goal was, and we both agreed that it would be really cool if we could have an app that looked and functioned well enough to be put into production by the time we turn this in. With that in mind, the layout and look were important to us. We spent just as much time on design as we did writing inner functionality. For that reason, our app may not be extremely complex, but it does its job, and it looks good.

We decided to use Ionic for building this app because using the Ionic mark-up allows the code to be compiled into both Android and IOS, meaning this app could potentially be run on Apple and Android phones, which meant it would be able to reach more people.

-Analysis

This app was created as a place for people to share their photos. It has four basic parts to it. The login/signup page, cities page, home page, and posts page. The home was designed to give a little insight into the app and navigation, while the Post and Cities pages were created to provide an interface for posting new images, and viewing already existing images. The login/signup pages are pretty self-explanatory.

# New And Complex Section

## New Things

The UI Router was something we did not learn to implement in class. It was a little difficult to get the syntax down, but once it was understood, it really opened a lot of doors for us. The main one being nested views, and the second one being a way to change the view inside javascript using the $state.go function. This was implemented in the login and signup pages. After the javscript was validated, if all requirements were meant the user would be sent to the next page, either the home page of the website, or the login page if they were on the signup page.

Directives were another new thing for us. We had never used one before, but it really proved to be a cool way of coding. Directives are great because they contain html markup, and they can also have basically their own controller which is provided through the link: attribute of the directive. In the directives html, we could have different functions, and event if a function had the same name as a function in the pages main controller, it would be out of the scope of the directive, therefore it would only use the function inside the directive’s link function. We could also inject the entire scope into the directives controller, so that we could use scope.mainFunction to use functionality in the main controller. This made it kind of one way binding.

Lastly in regards to the directives, we discovered a cool workaround called scope.$apply. This basically was a way to make the angular bindings update upon a change. In the case of the posts page, we had to update it once a new file was uploaded. We basically just called scope.$apply, and changed the variable we needed, which instantly sent that new information to the main scope.

Services provided another interesting battle for us. A service was a way for the front end client code to work with the back end. Services are like controllers, except they are declared using angular.service() instead of angular.controller(). Once the service is declared, it has to be injected as a dependency into all controllers that use it. Then all the functions inside the service can be used with serviceName.function() syntax. We used these everytime we needed to make a back end call.

A constants angular object was another cool way for us to keep track of our file path names. Constants are also declared similar to controllers, except they use angular.constants() to declare them. Also similar to a service, anytime the members of a constant were used, the constant had to be injected into that controller, and then accessed using constanName.constant, very similar to services. We only every injected our constants into the services, and then used them in our $http request to declare which file path to go to.

Ionic was a super cool discovery to us. We had never used it before, but after learning it could be used to cross compile a web app into Android and IOS, we were sold. The ionic syntax wasn’t too difficult to learn, but there truly is a lot to it. Ionic has its own classes and everything, so it can get really complex. It has buttons, headers, content, and slide markup, which is basically what we stayed with. It also has a cool ion-nav-view which is similar to ng-view, and basically it just populates with the html of whatever page it is on.

The last new thing we used, which is probably also interesting enough to include in this write is storing images as base64. It was really cool for us to learn, that and img src can actually just be a string of base64 information. We used this knowledge to store our images. Our images.txt file contains all the base64 information, and then each time it is loaded inside the cities view, we use an ng-repeat to loop through all the images, and then display them using the src attribute.

## Complex Things

We both agree that the most complex things were again, using the John Papa syntax, using Ionic, and learning how to display and store images. Once these things were learned, they really stopped being so complex, but going through the process of their discovery could be very time consuming at times.

John Papa really covers a lot in his blogs. Using $http request inside services meant everytime we had to call the backend, we needed to create a service directory, and service.js file. This adds a lot of overhead, but in the end it pays off, because all the code is very well organized and easy to find should something need to be changed.

In regards to John Papa, we used the vm.function() syntax inside our controllers, which was another way of distinguishing where a function was coming from. In order to use the vm syntax we had to provide a controllerAs : vm attribute inside our app.js file, otherwise angular doesn’t really seem to know what is going on. Using vm.function() is basically the same as using $scope.function(), except it’s easier to read. You can see this at snippet 6.

Also, to follow John Papa, we surrounded all our controller code in a (function() declaration. This you can find at snippet **1.**

With using Ionic, it allowed us to keep our angularJS code, but it required changes to the html. For instance, we were still able to use our UIRouter, but instead of providing a div with the ui-view syntax, we simply used the ion-nav-view markup. Somehow Ion still knows to look into the UI View states to decide what to populate the ion-nav-view with. You Can see this at snippet 2.

Ion also uses a cool ion-header-bar markup which will display as a fully functional header layout, without requiring the coder to write any CSS. The CSS is automatically included. You can see this at snippet 3.

The last cool Ion thing to point out is the amazing nav menu Ion provided. All the functionality for that literally comes with ion. It’s included inside the menu.html file. Basically an ion-nav-button tag is provided with a menu-toggle attribute. Whatever the attribute is labeled as, links to the ion-side-menu with the same value as the side attribute. This can be seen in snippet 4. The use of this Ion gem saved us what could have been hours in coding.

Finally, in regards to new and complex things is the fileReading directive we used on the posts page. We had to read a file’s content in base64, and then provide that text as the src attribute to an image file. We had never used the javascript file reader before, so this was a really interesting thing for both of us. This can be seen at snippet 5.

# Bloom’s Taxonomy Section

**-**Creation

Using the MVC format, and following John Papa’s style guide really taught us a lot about using angularJS. One of the best things we learned was just how to write nice, clean code, that is easy to follow and understand. Using separate directories for each page is a really good technique that we plan on doing in the future. Its easy to see, especially as you start designing bigger applications, how this could be useful. Every page has its own directories for the views, controllers, services, and directives associated with it, then when you’re looking through the website and you see an error, you know exactly where to go to find it in your code.

We also think the use of vm in our controllers was a really great syntax. It just helped us to be sure where a function call was going. If It had vm.function() we knew it was going to the controller of whatever page we were on. Angular is also great because instead of having just one huge javascript file that contains all the code for the website, we have many small files, that only relate to a single page.

The angular.constants() angular.factory(), and angular.service() objects were another cool format that we found useful. The constants provide a great way of storing what are basically global variables, that can be accessed anywhere in the code. Then the factories and services are really similar to each other, but we learned that the services should basically be used to make calls to the back end, while the factory provides a means of sharing code between controllers. We created a factory for everytime we need to show or hide the loading bar. We would inject the loadbar factory into every controller that needed it, and then call it’s functions, that way we didn’t need to rewrite the showLoadBar() and hideLoadBar() methods in every controller.

**-**Analysis

This app is a way for people to share photos of different places they have visited. Because of this, we decided to design the app around that basic functionality, and avoiding from over complicating anything. We did a lot of design, and learned a lot about the different Ionic classes. Ionic is just like bootstrap, except it basically has different names. For instance, bootstrap buttons would use classes such as “btn btn-primary, btn-danger”, but Ionic uses the classes “button button-calm button-positive”, which actually have very similar looks, but are just different names.

As far as the functionality, one of the biggest hurdles we had was storing images. Now that we know how to do it, we realize it’s really an easy concept, but figuring it out was difficult. We looked online for advice, and learned that the best way to store images is to upload them to the server, and then store their url in a database, so this is what we did. We had some trouble with our XAMPP server, but after changing the file permissions in the xamppfiles (mac) folder, we were able to read and write to text files in our project directory.

Honestly, as far as this whole project went, some of the most trouble we had was just in making simple little things work. We couldn’t access $\_POST variables in our php files, so we used the file\_get\_contents(php://input) workaround for all of our scripts. This took way too long to figure out. Then we also had an error that we couldn’t initialize our session variables because we didn’t have the right permissions, so we had to find our temp folder in the server, and modify the permissions so we could use it. We also learned a ton about setting up servers with IntelliJ, and I’m pretty sure both of us are experts now. We learned that we can’t put our src folder inside the .idea folder for IntelliJ, or else it will cause IntellIiJ to use the default server, regardless of having a different server specified. There were lots of little things in this project that held us back, but after overcoming all these obstacles, I think we are substantially better off.

**-**Evaluation

We decided to construct the application the way we did simply because that’s what we read, and learned would be the best. As stated above, we read it was best to upload images to the server, and store their url’s in a database, we also learned it was a good practice to download all images used in an app, store them in a directory, and call them whenever they need to be displayed, this way you don’t have to rely on a picture from a url somewhere else on the web.

Also in regards to the angularJS, John Papa taught us the importance of constants, factories, and services. We learned if we wanted to share code between controllers, we should create a factory, and inject that factory into each controller that needs the code. We learned that services are used to make calls to the back end, and we learned that controllers should be created using the vm syntax, to create that coherence between views and their controllers.

We didn’t do anything in this app without first researching the best practices, and coming to an agreement about what we should implement. As we have stated, this caused us to spend a lot more time than was probably necessary, but we know now how to make a legitimate application in the future.