

Create Your First Mobile App with AngularJS and Ionic

By **Chris Nwamba**, scotch.io

December 17th, 2015

The Scotchmas Day 4 giveaway can be found at the end of this article.

In the past years, making a mobile app involved being good with the native language you are creating an app for. That means Objective C for iOS, Java for Android, and C# for Windows Phone. That is so demoralizing to have to learn all those technologies.

My dreams of being a mobile app developer became a reality after I realized I could actually create mobile apps using the skills I had that were web technologies.

Cordova is a game changer. With it you can target multiple platforms with one code base. All you need to learn is the usual HTML, CSS and JavaScript used for web development and by reading some Cordova docs, you are already to become a mobile developer.

What Really is Ionic?

Ionic is a front-end SDK for building cross-platform mobile apps. Built on top of Angular, Ionic also provides a platform for integrating services like push notifications and analytics.



Ionic is not an alternative option to Cordova, but instead a UI library to make a better Cordova project. Ionic can be compared to a framework like Bootstrap or Foundation but this time for Mobile and not Web.

In this tutorial which has a nice demo, we will make a mobile app using Ionic framework and Cordova. I will also explain how to build mobile apps with Phonegap app so we won't have to bother with installing Platform SDKs.

What You Should Know

Fortunately you just need to understand the basics of JavaScript and its popular library AngularJS. Ionic is completely made up of Angular directives and components and uses its popular ui router for SPA.

Currently, Ionic works with Angular 1.0, but the Ionic 2 Alpha that works with Angular 2 was announced.

Setup and Installation

These days it seems like everything now depends on Node to exist. Ionic and Cordova are not exceptions. Installing Cordova or Ionic requires Node and npm installed on your machine. Run the following command after installing Node if you haven't:

```
npm install -g cordova ionic
```

The **-g** flag installs both cordova and ionic globally so we can access it from any where in our machines. It also adds them to our **PATH** so it won't be a surprise to our CLI. To confirm installation:

```
ioniv -v && cordova -v
```

Creating a Project

A new project in ionic can be created using the **ionic** CLI command. It will scaffold the app's file and download it's dependencies. You can supply arguments to the CLI command to alter the type of template generated.

```
ionic start {appname} {template}
```

That is the syntax for a new project in Ionic. The **appname** is any name of your choice and **template** is one of the Ionic supported template. It can also be a GitHub URL to a custom template.

There is nothing wrong with tradition - let us create a Todo app to better see what Ionic is made of.

Making a Todo App

We can go ahead and create an app as seen in the previous section. So as usual, pick your favorite spot on you machine and run:

```
ionic start scotch-todo blank
```

We are using the **blank** template for simplicity in as much as we can use other types of templates like **tabs**, **sidemenus**, etc. The command will generate a new project for us and also install some dependencies.

```
|— hooks          // custom cordova hooks to execute on
specific commands
|— platforms      // iOS/Android specific builds will reside
here
|— plugins        // where your cordova/ionic plugins will
be installed
|— resources      // icon and splash screen
|— scss           // scss code, which will output to
www/css/
|— www            // application - JS code and libs, CSS,
images, etc.
    |-----css          //customs styles
    |-----img          //app images
    |-----js           //angular app and custom
JS
    |-----lib          //third party libraries
    |-----index.html   //app master page
|— bower.json      // bower dependencies
|— config.xml      // cordova configuration
|— gulpfile.js     // gulp tasks
|— ionic.project   // ionic configuration
|— package.json    // node dependencies
```

Configuring Platforms

As we discussed, Cordova gives you the power to write once and run on many platforms (iOS, Android, Blackberry, Windows, etc) but you have to explicitly tell Cordova (with Ionic) which platform should be included in a given project. For our Todo app, we will be trying with two platforms (iOS and Android).

```
cd scotch-todo
ionic platform add ios
ionic platform add android
```

Previewing the App

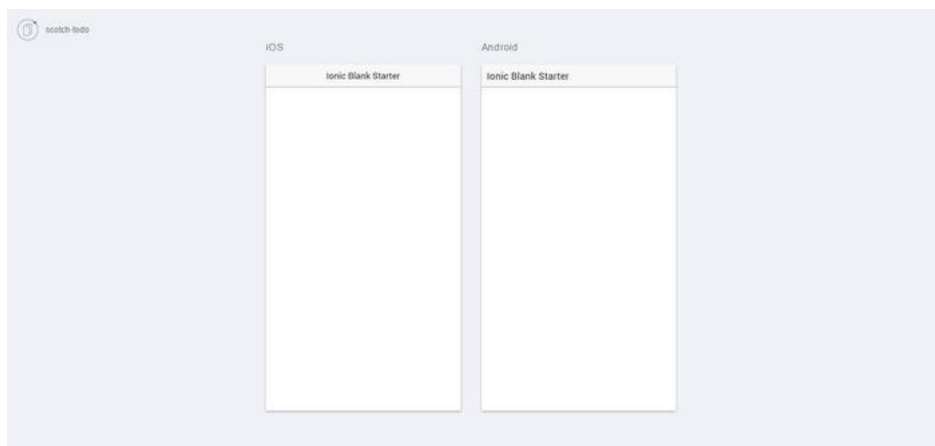
We can preview our app in the browser using Ionic. This is not recommended though (about that later) but we can use it to preview (not really test) our basic Todo application.

To see what our blank app looks like, run the below command in the CLI on the **scotch-todo** directory:

```
ionic serve
```

The command will open up a browser at **http://localhost:8100**. The browser screen is too large to preview a mobile app and for that reason, Ionic added **--lab** as an option to running the app in the browser so as to preview the app better as it would be seen in Android and iOS.

```
ionic serve --lab
```



Building Out the Real Stuff

Fine we have a good ionic app but then it is useless. We agreed on making a Todo app not a blank app so let us complete the deal.

Ionic is a basically a SPA web app, we need a master index.html though we don't need routes as the Todo app is just a one view app.

We should first update our app's `www/index.html` and `www/js/app.js` to make sure the structure looks like the one below.

`www/index.html`

```
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-
scale=1, user-scalable=no, width=device-width">
    <title></title>

    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">
    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>

    <!-- cordova script (this will be a 404 during
development) -->
    <script src="cordova.js"></script>

    <!-- your app's js -->
    <script src="js/app.js"></script>
  </head>
  <!-- your app's js -->
  <body ng-app="scotch-todo">

    <ion-pane>
      <ion-header-bar class="bar-dark">
        <h1 class="title">Scotch-Todo</h1>
      </ion-header-bar>
      <ion-content>
      </ion-content>
    </ion-pane>
  </body>
</html>
```

`www/js/app.js`

```
//setup angular
var app = angular.module('scotch-todo', ['ionic']);
```

Ionic creates a bundle of Angular and required dependencies (angular-ui-router, angular-animate, angular-sanitize, etc) bundled in one file as `ionic.bundle.js`.

Those are the files we will work with throughout this tutorial. Before continuing, let us define some tasks we need to complete in our app so we don't lose focus:

- Setup Local Storage service to store todo data
- Create a **main** controller to interact with our view
- Update our **index.html** to work with the controller

A Local Storage Service

HTML 5 which is Ionic's best friend comes with a lot of storage options. Local Storage is one of them and it suits our app more because of all the options provided, it has the ability to persist data even when the browser or app is closed.

Angular on the other hand has a service written to wrap the Local Storage API and is available on GitHub as `angular-local-storage`. Fortunately, Ionic uses bower to manage dependencies so installing **angular-local-storage** takes this CLI command:

```
bower install angular-local-storage --save
```

The command will download and add `angular-local-storage` to our app libraries. Just below where we added the **ionic.bundle.js** include `angular-local-storage` as seen below:

```
<script src="lib/angular-local-storage/dist/angular-local-storage.js"></script>
```

Tell angular it is a dependency by injecting it in the angular app in the **www/js/app/js** file like this:

```
var app = angular.module('scotch-todo', ['ionic',  
  'LocalStorageModule']);
```

It is a good practice to add a prefix to our stored entities so as avoid being overwritten in future. Add a config angular block to set the Local Storage prefix

```
app.config(function (localStorageServiceProvider) {  
  localStorageServiceProvider  
    .setPrefix('scotch-todo');  
});
```

Let us leave it at that and continuing to the second item on our task - the controller.

The main Controller

Our controller will be serving as the middle man between our data (Local Storage) and the view ([www/index.html](#)). Let us first create a structure of the controller just below our config block in the [www/js/app.js](#):

```
app.controller('main', function ($scope, $ionicModal,
localStorageService) { //store the entities name in a
variable var taskData = 'task';

//initialize the tasks scope with empty array
$scope.tasks = [];

//initialize the task scope with empty object
$scope.task = {};

//configure the ionic modal before use
$ionicModal.fromTemplateUrl('new-task-modal.html', {
  scope: $scope,
  animation: 'slide-in-up'
}).then(function (modal) {
  $scope.newTaskModal = modal;
});

$scope.getTasks = function () {
  //fetches task from local storage
  ...
}
$scope.createTask = function () {
  //creates a new task
  ...
}
$scope.removeTask = function () {
  //removes a task
  ...
}
$scope.completeTask = function () {
  //updates a task as completed
  ...
}
})
```

Notice how we have injected **localStorageService** in the controller so as to help us interact with Local Storage from the controller. We also injected **\$ionicModal** service to help us create tasks from a modal.

We also need to tell the **index.html** which controller to use and run **getTasks()** when the app starts by updating the **body** tag as shown:

```
<body ng-app="scotch-todo" ng-controller="main" ng-  
init="getTasks()">  
...  
</body>
```

We are making reasonable progress. Let us now implement those empty methods in the controller. We will start with the **getTasks()** which basically checks for **tasks** entity in the Local Storage and if it exists, fetch it.

```
$scope.getTasks = function () {  
    //fetches task from local storage  
    if (localStorageService.get(taskData)) {  
        $scope.tasks =  
localStorageService.get(taskData);  
    } else {  
        $scope.tasks = [];  
    }  
}
```

Furthermore we can create a new task by pushing a **task** object to the **tasks** array and updating the Local Storage with the **tasks**.

```
$scope.createTask = function () {  
    //creates a new task  
    $scope.tasks.push($scope.task);  
    localStorageService.set(taskData, $scope.tasks);  
    $scope.task = {};  
    //close new task modal  
    $scope.newTaskModal.hide();  
}
```

To remove task we can now remove a task object from the task array and update the Local Storage.

```
$scope.removeTask = function (index) {  
    //removes a task  
    $scope.tasks.splice(index, 1);  
    localStorageService.set(taskData, $scope.tasks);  
}
```

Then to complete a task we find task from the array using its index, update its completed value and update Local Storage.


```
$scope.completeTask = function (index) {  
  //updates a task as completed  
  if (index !== -1) {  
    $scope.tasks[index].completed = true;  
  }  
  
  localStorageService.set(taskData, $scope.tasks);  
}
```

That is it! We can boast of a controller now. Let us not get too excited though because our view is waiting patiently to get a treat as we did to our controller.

The View

For a cleaner display of tasks, let us use ionic's card component for display of tasks. So right inside the **<ion-content>** tag (which is actually a directive) add the following block:

```

<div class="list card" ng-repeat="task in tasks track by
$index">
  <div class="item item-divider">
    <span ng-bind="task.title"></span>
  </div>
  <div class="item item-body">
    <strong>Title: <span ng-bind="task.title">
</span></strong>
    <p>
      <span ng-bind="task.content"></span>
    </p>
  </div>

  <div class="item tabs tabs-secondary tabs-icon-
left">
    <span class="item item-checkbox">
      <label class="checkbox">
        <input type="checkbox" ng-
model="task.completed" ng-click="completeTask($index)">
      </label>
    </span>
    <a class="tab-item assertive" ng-
click="removeTask($index)">
      <i class="icon ion-android-close"></i>
    </a>
  </div>

</div>

```

With that we have a list of tasks presented with cards in our view. Now to create a new task, we will use the modal which we already configured in our controller. We are tracking the array with a unique \$index as it has none specified explicitly in the data source. Just before the **body** closing tag:

```

<script id="new-task-modal.html" type="text/ng-template">
  <ion-modal-view>
    <ion-header-bar class="bar-dark">
      <h1 class="title">Create a new Task</h1>
      <button class="button button-icon" ng-
click="closeTaskModal()">
        <i class="icon ion-android-close"></i>
      </button>
    </ion-header-bar>
    <ion-content>
      <form ng-submit="createTask()">
        <div class="list list-inset">
          <label class="item item-input">
            <input ng-model="task.title"
type="text" placeholder="Task title">
          </label>
          <label class="item item-input">
            <textarea ng-model="task.content"
rows="5" placeholder="Task content"></textarea>
          </label>
          <ul class="list">
            <li class="item item-toggle">
              Completed?
              <label class="toggle toggle-
balanced">
                <input type="checkbox"
ng-model="task.completed">
                <div class="track">
                  <div class="handle">
</div>
                </div>
              </label>
            </li>
          </ul>
          <button type="submit" class="button
button-block button-positive">Create Task</button>
        </div>
      </form>
    </ion-content>
  </ion-modal-view>
</script>

```

Finally on the view, add a button to the **ion-header-bar** directive which will be used to open the modal.

```
<ion-header-bar class="bar-dark">
  <h1 class="title">Scotch-Todo</h1>
  <!-- New Task button-->
  <button class="button button-icon" ng-
click="openTaskModal()">
    <i class="icon ion-compose"></i>
  </button>
</ion-header-bar>
```

And that is it. That is our sweet simple Todo mobile app. On the other hand, I don't think we are comfortable testing an app that is supposed to run on a mobile app in a browser. Let's do something about that.

Testing with Phonegap App

PhoneGap is a free and open source framework that allows you to create mobile apps using standardized web APIs for the platforms you care about.

PhoneGap just meets Cordova halfway to complement it where it is weak.

One of those weaknesses is realtime testing while developing. With just Cordova we can only install platform SDKs, build our app once and test with the SDKs which might be slow and affects productivity.

PhoneGap App lets us test our app in realtime while it is being built. No need for any sort of compiling or building before testing. All we need is a to place our machine and our device on the same network. First things first:

```
npm install -g phonegap
```

This will install the command line tool and give us access to use PhoneGap from the CLI. Next we need to install the app for our platform. I have an Android mobile device therefore am downloading from Play Store. You can see other platforms from the PhoneGap App website.

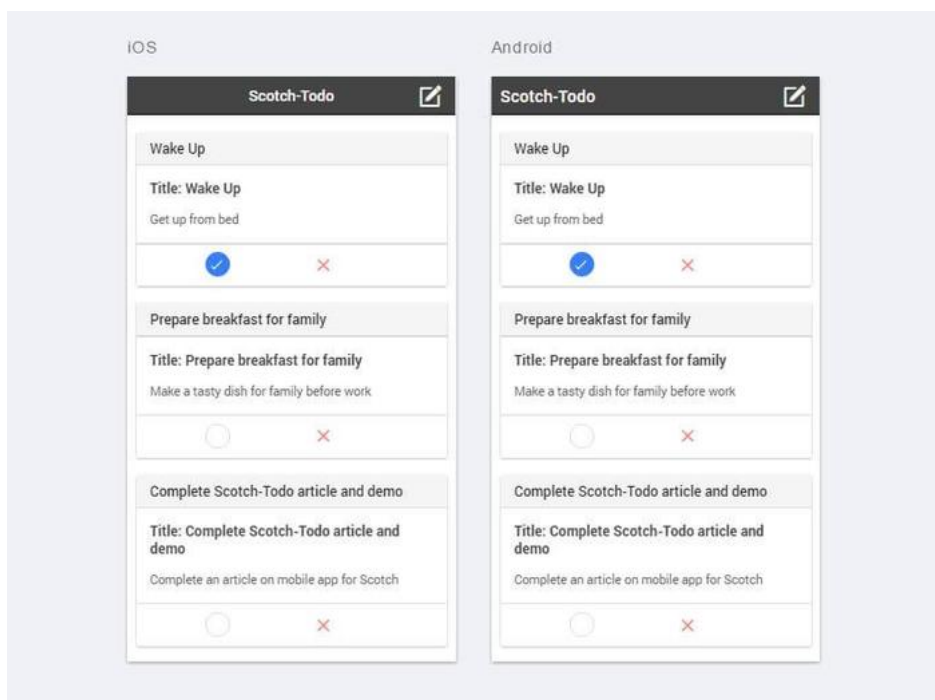
Finally on PhoneGap app, instead of using **ionic serve** on the Todo App, we will run with

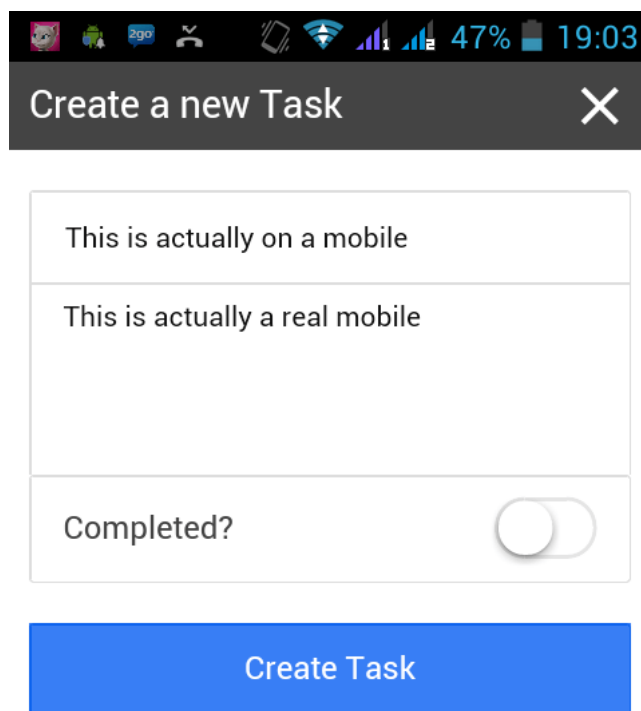
```
phonegap serve
```

```
MINGW64/c:/Users/CHRISTABEL/Documents/Articles/Scotch-Ionic/1-Scotch-Ionic-Intro/scotch-todo
CHRISTABEL@CHRISTABEL-PC MINGW64 ~/Documents/Articles/Scotch-Ionic/1-Scotch-Ionic-Intro/scotch-todo
$ phonegap serve
add to body class: platform-android
phonegap starting app server...
phonegap listening on 10.160.13.93:3000
phonegap ctrl-c to stop the server
phonegap
```

Notice the I.P address because that is what we will use to talk to our mobile device.

Open the installed app on your device and replace the I.P address and port with the one supplied by PhoneGap CLI and tap connect. Below is a preview of the app:





The screenshot shows a mobile app interface for creating a task. At the top is a status bar with various icons and a battery level of 47% at 19:03. Below the status bar is a dark header with the text "Create a new Task" and a close button (X). The main content area is a light gray box with three sections: a text input field containing "This is actually on a mobile", another text input field containing "This is actually a real mobile", and a toggle switch labeled "Completed?". At the bottom is a blue button with the text "Create Task".

Feel free to feel like the man (or woman). We did it! Our app is running live on our mobile device. You owe me a cup of coffee though.

Conclusion

I am so excited and proud to have completed this tutorial with you. As you can see, you didn't have to be a Java geek or Objective C nerd or C# professional to develop a mobile app, we just need to apply our Web development skills. I think that is amazing. This is just the beginning, expect more on Ionic from Scotch.

1 Item deleted