

1. All about Ionic
2. Installation
3. Starting your app
4. Testing your app
5. Building out your app
6. Publishing your app

## Chapter 1: All about Ionic

Welcome to the official guide to building HTML5 mobile apps with the Ionic Framework, written by the [creators](#) of Ionic. It contains all you need to know to get started building apps with Ionic, and lays a foundation for more advanced development.

If you've used other mobile development frameworks in the past, you should find Ionic fairly similar to use. But getting started with any framework is always daunting, so we will start simple and expand on some basic concepts. But first, we need to talk a bit about the Ionic project itself, where it fits into the dev stack, and why we built it.

### What is Ionic, and where does it fit?

Ionic is an HTML5 mobile app development framework targeted at building hybrid mobile apps. Hybrid apps are essentially small websites running in a browser shell in an app that have access to the native platform layer. Hybrid apps have many benefits over pure native apps, specifically in terms of platform support, speed of development, and access to 3rd party code.

Think of Ionic as the front-end UI framework that handles all of the look and feel and UI interactions your app needs in order to be compelling. Kind of like "Bootstrap for Native," but with support for a broad range of common native mobile components, slick animations, and beautiful design.

Unlike a responsive framework, Ionic comes with very native-styled mobile UI elements and layouts that you'd get with a native SDK on iOS or Android but didn't really exist before on the web. Ionic also gives you some opinionated but powerful ways to build mobile applications that eclipse existing HTML5 development frameworks.

Since Ionic is an HTML5 framework, it needs a native wrapper like Cordova or PhoneGap in order to run as a native app. We strongly recommend using Cordova proper for your apps, and the Ionic tools will use Cordova underneath.

## Why did we build Ionic?

We built Ionic because we strongly believed that HTML5 would rule on mobile over time, exactly as it has on the desktop. Once desktop computers became powerful enough and browser technology had advanced enough, almost everyone was spending their computing time in the browser. And developers were overwhelmingly building web applications. With recent advancements in mobile technology, smartphones and tablets are now capable of running many of those same web applications.

With Ionic, we wanted to build an HTML5 mobile development framework that was focused on *native* or hybrid apps instead of mobile websites, since we felt there were [great tools](#) already for mobile website development. So Ionic apps aren't meant to be run in a mobile browser app like Chrome or Safari, but rather the low-level browser shell like iOS's UIWebView or Android's WebView, which are wrapped by tools like Cordova/PhoneGap.

And above all, we wanted to make sure Ionic was as open source as possible, both by having a permissive open source license that could be used in both commercial and open source apps, but by cultivating a strong community around the project. We felt there were too many frameworks that were *technically* open source, but were not open source in spirit or were not possible to use in both closed source and open source projects without purchasing a commercial license.

## Building Hybrid Apps With Ionic

Those familiar with web development will find the structure of an Ionic app straightforward. At its core, it's just a web page running in a native app shell! That means we can use any kind of HTML, CSS, and Javascript we want. The only difference is, instead of creating a website that others will link to, we are building a self-contained application experience.

The bulk of an Ionic app will be written in HTML, Javascript, and CSS. Eager developers might also dig down into the native layer with custom Cordova plugins or native code, but it's not necessary to get a great app.

Ionic also uses AngularJS for a lot of the core functionality of the framework. While you can still use Ionic with just the CSS portion, we recommend investing in Angular as it's one of the best ways to build browser-based applications today.

## Get building!

Now that you have and understanding of what Ionic is and why it exists, you are ready to start building your first app with it. Continue on to get everything installed and start building with Ionic!

# Chapter 2: Installation

In this chapter, we are going to walk through the process of downloading Ionic and installing all necessary dependencies for development.

## Platform notes

First, we need to start with a note about minimum requirements for building your app with the current release of Ionic. Ionic targets iPhone and Android devices (currently). We support iOS 6+, and Android 4.0+ (though 2.3 should work). However, since there are a lot of different Android devices, it's possible certain ones might not work. As always, we are looking for help testing and improving our device compatibility and would love help from the community on our [GitHub](#) project.

You can develop Ionic apps on any operating system you prefer. In fact, Ionic has been developed at various times on Mac OS X, Linux, and Windows. However, right now you'll need to use the command line in order to follow this guide and you must have OS X in order to develop and deploy iPhone apps, so OS X is recommended if possible.

If you are on Windows, make sure to download and install [Git for Windows](#) and optionally [Console2](#). You will be executing any commands in this guide in the Git Bash or Console2 windows.

First, we will go and install the most recent version of [Apache Cordova](#), which will take our app and bundle it into a native wrapper to turn it into a traditional native app.

To install Cordova, make sure you have [Node.js](#) installed, then run

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

Drop `sudo` from the above command if running on Windows. Depending on the platforms you wish to develop for, you'll need to install platform-specific tools. Follow the Cordova platform guides for [Android](#) and [iOS](#) to make sure you have everything needed for development on those platforms. Luckily, you'll only need to do this once.

Linux Android note

Windows note on Java, Ant and Android

Windows users developing for Android: You'll want to make sure you have the following installed and set up.

*NOTE: Whenever you make changes to the PATH, or any other environment variable, you'll need to restart or open a new tab in your shell program for the PATH change to take effect.*

## Java JDK

Install the most recent [Java JDK](#) (NOT just the JRE).

Next, create an environment variable for `JAVA_HOME` pointing to the root folder where the Java JDK was installed. So, if you installed the JDK into `C:\Program Files\Java\jdk7`, set `JAVA_HOME` to be this path. After that, add the JDK's `bin` directory to the `PATH` variable as well. Following the previous assumption, this should be either `%JAVA_HOME%\bin` or the full path `C:\Program Files\Java\jdk7\bin`

## Apache Ant

To install Ant, download a zip from [here](#), extract it, move the first folder in the zip to a safe place, and update your `PATH` to include the `bin` folder in that folder. For example, if you moved the Ant folder to `c:/`, you'd want to add this to your `PATH`: `C:\apache-ant-1.9.2\bin`.

## Android SDK

Installing the [Android SDK](#) is also necessary. The Android SDK provides you the API libraries and developer tools necessary to build, test, and debug apps for Android.

Cordova requires the `ANDROID_HOME` environment variable to be set. This should point to the `[ANDROID_SDK_DIR]\android-sdk` directory (for example `c:\android\android-sdk`).

Next, update your `PATH` to include the `tools/` and `platform-tools/` folder in that folder. So, using `ANDROID_HOME`, you would add both `%ANDROID_HOME%\tools` and `%ANDROID_HOME%\platform-tools`.

# Install Ionic

Ionic comes with a convenient command line utility to start, build, and package Ionic apps.

To install it, simply run:

```
$ sudo npm install -g ionic
```

## Create the project

Now, we need to create a new Cordova project somewhere on the computer for the code for our app:

```
$ ionic start todo blank
```

That will create a folder called `todo` in the directory the command was run. Next, we will go into that directory and list the contents. Here is what the outer structure of your Ionic project will look like:

```
$ cd todo && ls
```

```
├─ bower.json    // bower dependencies
├─ config.xml    // cordova configuration
├─ gulpfile.js   // gulp tasks
├─ hooks         // custom cordova hooks to execute on specific commands
├─ ionic.project // ionic configuration
├─ package.json  // node dependencies
├─ platforms     // iOS/Android specific builds will reside here
├─ plugins       // where your cordova/ionic plugins will be installed
├─ scss          // scss code, which will output to www/css/
└─ www          // application - JS code and libs, CSS, images, etc.
```

If you are planning on using any version control system, you can go ahead and set it up in this new folder.

## Configure Platforms

Now, we need to tell ionic that we want to enable the iOS and Android platforms. Note: unless you are on MacOS, leave out the iOS platform:

```
$ ionic platform add ios
```

```
$ ionic platform add android
```

If you see errors here, make sure to follow the platform guides above to install necessary platform tools.

Android on OS X note

## Test it out

Just to make sure the default project worked, try building and running the project (substitute ios for android to build for Android instead):

```
$ ionic build ios  
$ ionic emulate ios
```

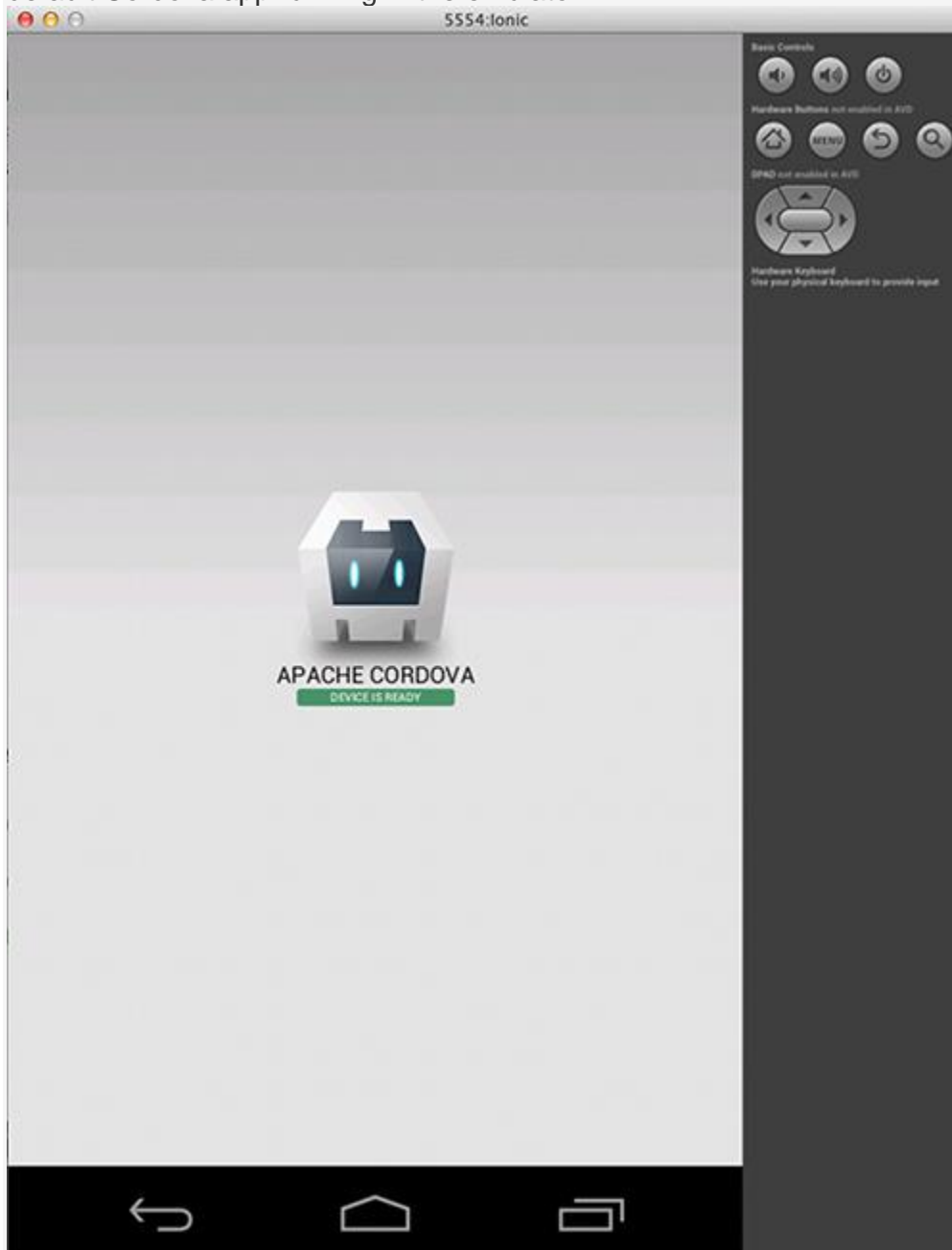
Android emulator note

We don't recommend using "emulate" for Android development. Unfortunately, the default Android emulator is horribly slow and will not accurately represent a real device. Using the emulator isn't even recommended for native Android development.

Fortunately, there are some great alternatives out there. Our favorite is a tool called [Genymotion](#) which can run an Android device as a virtual machine on your computer. It's much faster! If you use Genymotion, you'll use `ionic run` instead of `ionic emulate` as a Genymotion image appears as a physical device to the operating system.

If you chose to emulate on Android, be patient as this takes several minutes as the Android emulator is booted up. If you don't see anything happen for a few minutes, make sure you've created an Android Virtual Device (AVD), and that it is using the Android SDK version 17 or above. You might have to [reduce the amount of memory](#) the AVD is using if you don't see the emulator boot up in a minute. The platform guide above has more information. You may also want to double check that you have the sdk and platform tools in your PATH as noted in the platform guide.

The emulator takes a LONG time to boot. After about 5 or 10 minutes, you should see the default Cordova app running in the emulator:



Of course, you can always test directly on the device, and that is the recommended way to develop on Android due to the slow emulator. To do that, substitute `emulate` with `run` and ensure you have an Android device connected to the computer.

Let's go!

Now we are ready to start building our app, so continue on to the next chapter:

## Chapter 3: Starting your app

Now that we've got everything installed and a new Cordova project created, let's jump in and start building a real app!

The Todo list app is pretty much a rite of passage for frameworks, so we will walk through the process of building one with Ionic.

Since every Ionic app is basically a web page, we need to have an `index.html` file in our app which defines the first page that loads in the app. Let's create `www/index.html` and initialize it with this:

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

    <link href="lib/ionic/css/ionic.css" rel="stylesheet">

    <script src="lib/ionic/js/ionic.bundle.js"></script>

    <!-- Needed for Cordova/PhoneGap (will be a 404 during development) -->
    <script src="cordova.js"></script>
  </head>
  <body>
  </body>
</html>
```

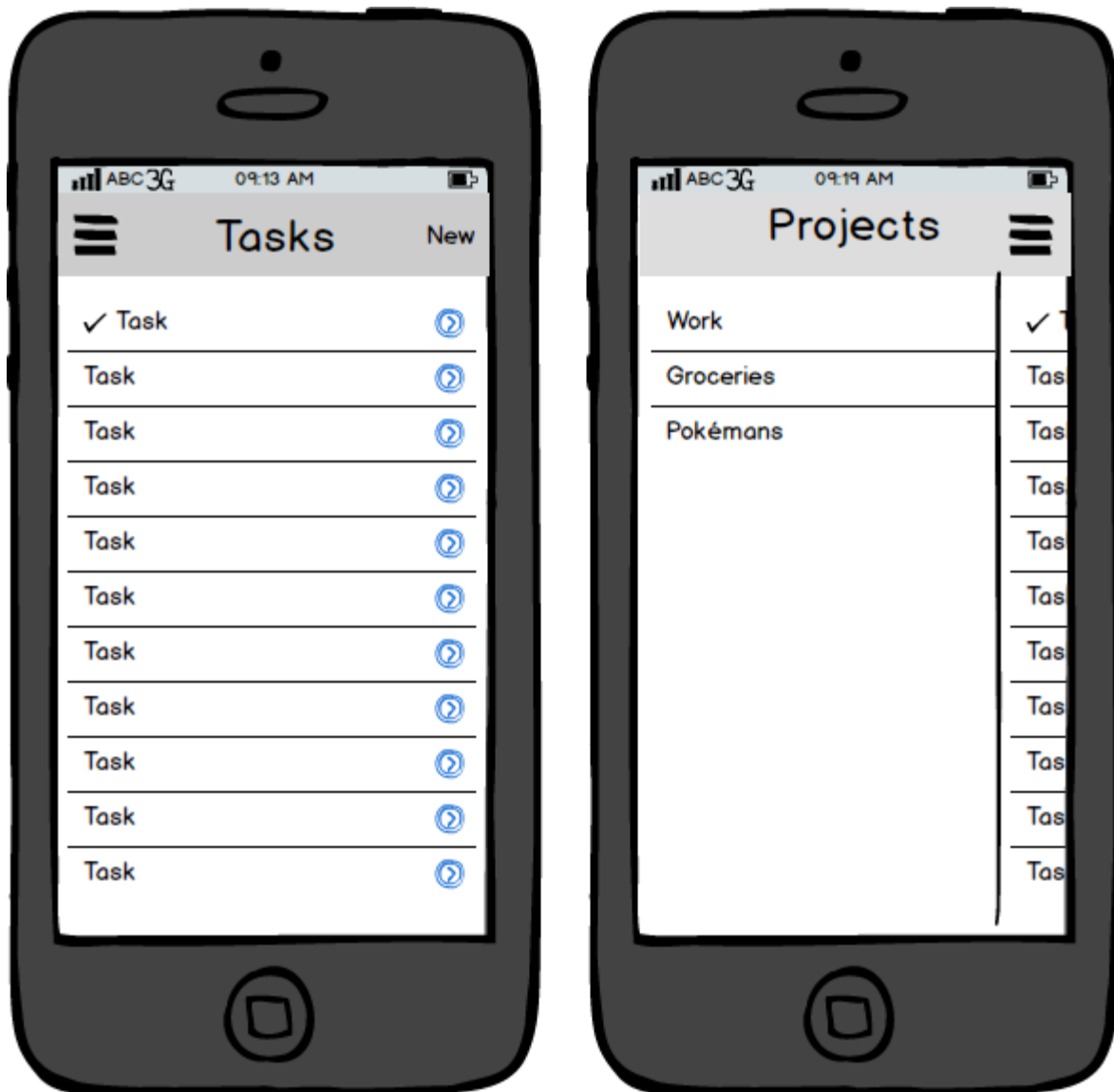
In the shell above, we are including the Ionic CSS and both the core Ionic JS and the Ionic AngularJS extensions in the `ionic.bundle.js` file. Ionic comes with `ngAnimate` and `ngSanitize` bundled in, but to use other Angular modules you'll need to include them from the `lib/js/angular` directory.



Also note that the cordova.js or phonegap.js file must be the last script, and that this file will not be found in your development files, but rather automatically included when running or emulating your app.

Now that we have our starting place, we need to figure out what the UI of the app will look like. We have a ton of choices when it comes to UI design on mobile. There are a few standard layouts, like tabs and side menus, but there are practically infinite custom layouts that we could implement if we really wanted to (which Ionic encourages!). For the sake of this example, we are going to pick a simple Side Menu layout which lets us drag and expose a side menu with center content.

In the center content area, we will list the various tasks that we have to get completed for the current project. We can add new tasks, or edit existing ones. If we drag the center content over to the right, we expose the left side menu which lets us choose the current project we want to edit, or create new projects. Take a look at the mockup below for an example of what we are trying to build:



To create side menus in Ionic, we can use [ion-side-menus](#). Feel free to read up on it, but the markup needed is simple. Edit the [index.html](#) file and change the `<body>` content to look like this:

```
<body>
  <ion-side-menus>
    <ion-side-menu-content>
    </ion-side-menu-content>
    <ion-side-menu side="left">
    </ion-side-menu>
  </ion-side-menus>
</body>
```

In the code above, we've added our `<ion-side-menus>` controller which will handle the dragging and exposing of the side menu. Inside of the controller we have a `<ion-side-menu-content>` which is the center content area of the app, and a `<ion-side-menu side="left">` which is a left, initially hidden, side menu.

## Initializing the app

Now, if you run this code (more on testing in a bit), you wouldn't see anything! There are two reasons for that: We haven't created an AngularJS app to turn the custom tags (like `<ion-side-menus>`) into anything functional, and we don't have any content yet!

Let's fix that. First, we need to create a new AngularJS module and tell Angular to initialize it. Let's create a new file located at [www/js/app.js](http://www/js/app.js). Put this code into the file:

```
angular.module('todo', ['ionic'])
```

This is the Angular way of creating an application, and we are telling angular to include the `ionic` module which includes all of the Ionic code which will process the tags above and make our app come to life.

Now, go back to `index.html` and right before the `<script src="cordova.js"></script>` line, add:

```
<script src="js/app.js"></script>
```

This includes the script we just created.

And to make our new app run, we need to add the `ng-app` attribute to the body tag:

```
<body ng-app="todo">
```

Now, we still won't see anything if we run the code, because we need to add some content to the application. Let's go ahead and add a header for both the center content area and the left menu.

Update the body content to be:

```
<body ng-app="todo">
  <ion-side-menus>

    <!-- Center content -->
    <ion-side-menu-content>
      <ion-header-bar class="bar-dark">
        <h1 class="title">Todo</h1>
```

```
</ion-header-bar>

<ion-content>

</ion-content>

</ion-side-menu-content>

<!-- Left menu -->

<ion-side-menu side="left">
  <ion-header-bar class="bar-dark">
    <h1 class="title">Projects</h1>
  </ion-header-bar>
</ion-side-menu>

</ion-side-menus>

</body>
```

## Testing it

With our shell and app ready to run, let's test it out and see how it looks!

# Chapter 4: Testing your app

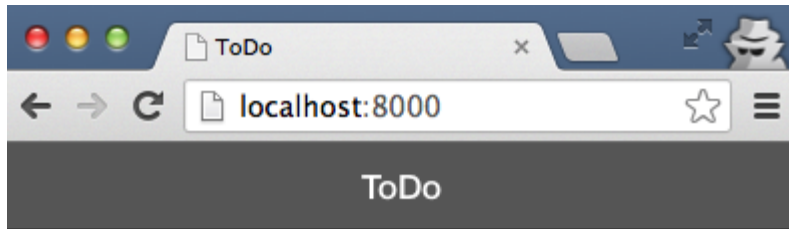
Now, since we actually have something to look at, we need to talk about the testing and development process for our app. There are four ways to test your app as you develop: in a desktop WebKit browser, in the iOS or Android simulator, in a mobile browser on your phone, or as a native app on the phone.

## Desktop browser testing

Testing your app in a browser is as simple as running the serve command in your project's root folder.

```
$ ionic serve
```

This will start a live-reload server for your project. When changes are made to any HTML, CSS, or JavaScript files, the browser will automatically reload when the files are saved.



Try dragging the center content to the right (works with the mouse as well) to expose the left menu. Smooth like butter!

## Simulator testing

You can also test right in the simulator using the cordova commands from the previous chapter. For example, to test in the iOS simulator, run:

```
$ ionic build ios  
$ ionic emulate ios
```

Substitute ios with android for Android testing. If you want to get advanced, you can also open up the project file for a specific platform by opening the required XCode or Android Eclipse project in `platforms/PLATFORM` inside the root of your project. Then, you can build and test from inside the platform-specific IDE. Note: if you go this route, I recommend still working inside of the root `www` folder, and when you've made changes to this folder, run the command:

```
$ cordova prepare ios
```

Which will update the ios specific project with the code from the `www` folder. Note: this will overwrite any changes you've made to the `platforms/ios/www` and other platform-specific folders.

## Mobile browser testing

You can also test the app directly in a mobile browser. For OS X users, Safari on OS X can directly debug websites and simulator applications. First you have to enable the remote web inspector on both the device and Safari on desktop. To do this with iOS 7 and OS X Mavericks, enable the `Web Inspector` option in the iOS Settings -> Safari -> Advanced section, and also enable the Developer Menu in the Advanced section of the Safari OS X settings.

Android apps supporting Android 4.4 or above can also use Chrome for remote debugging. Check out the Android docs for [more info](#).

If you are using the local server method from the Desktop testing section and you are on the same network as the desktop computer, you can connect to the ip address of the desktop computer to test. So, if our desktop is running a test server at `192.168.1.123:8000`, we can just load that address into our mobile Chrome or Safari to test it.

One problem with testing in a mobile browser is that it's probably the furthest of the three options from the actual app experience. This is largely because the browser app is meant for browsing websites, so it often adds functionality that conflicts with your app. For example, Chrome and Safari both listen for drag events on the sides of the app which let you switch between open tabs. They also have issues with the URL bars getting in the way, and some scrolling behavior is not the same as it is in the web view running in Cordova. It is fine for small tests, but not recommended for more complex apps.

## Testing as a native app

Since we are building a native (or "hybrid") app, we can (and should!) test it as one. There are several ways to do this. If you are building for iOS, you'll need to sign up for an [Apple Developer](#) account to test as a native app on an iPhone or iPad. Unfortunately, this costs \$99 per year (don't blame us!). Once you have an account and you have set up XCode with

your certificates to enable device testing, you'll want to open the XCode project from `platforms/ios/` and do your testing from XCode.

Testing on Android is much easier and faster. To test on the device, simply plug it in, and run

```
$ ionic run android
```

If this doesn't work, make sure you have USB debugging enabled on your device, as [described](#) on the Android developer site.

## Building it out

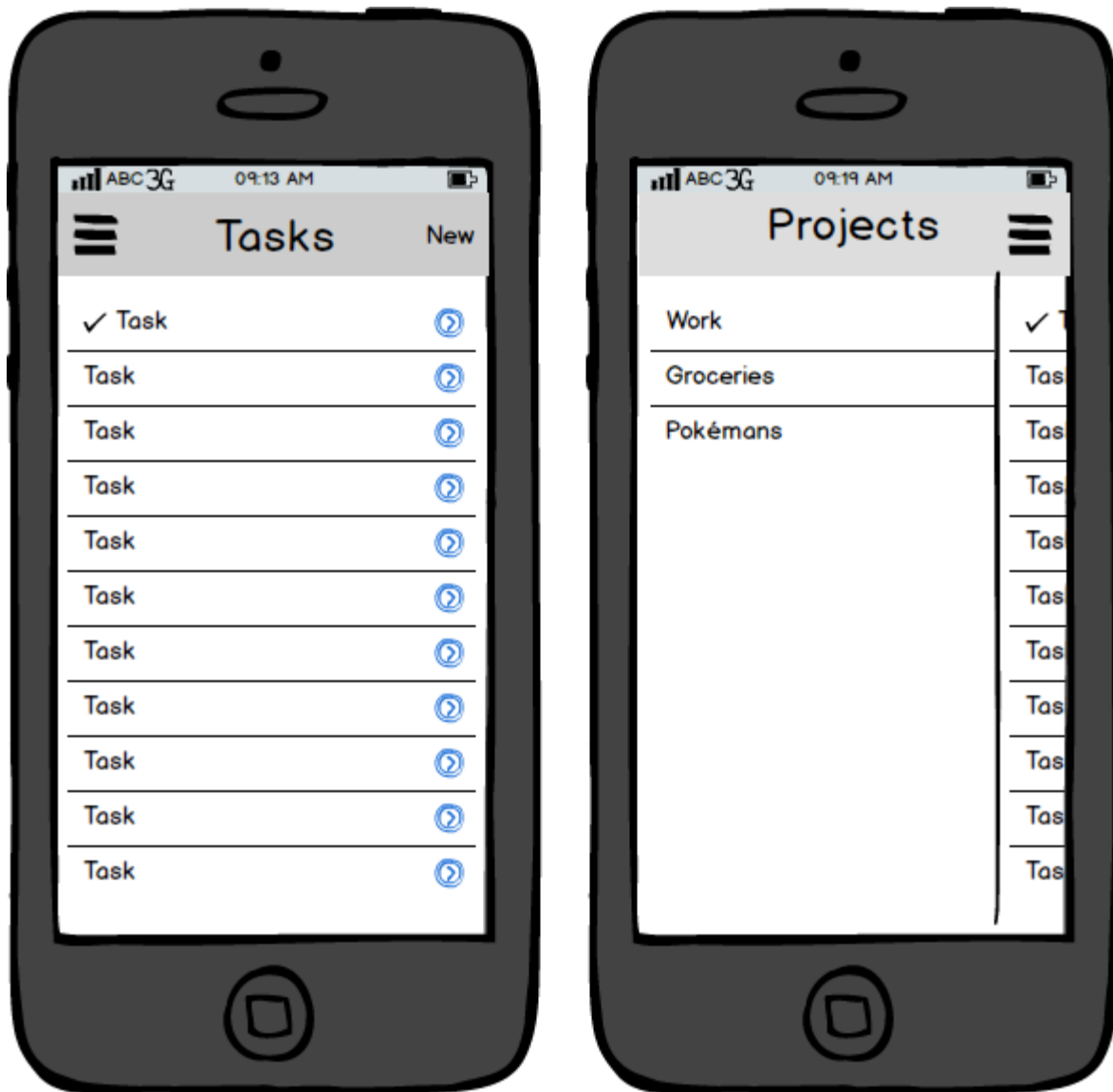
Now that we have a shell to test and we know how to test our app, let's start building out the guts of the app!



# Chapter 5: Building out your app

Now that we know everything there is to know about testing our Ionic apps, and we have a working app shell, let's move on to actually making some bacon!

So, let's take another look at our mockup:



We can see that both the center content and side menus have lists. Lists in Ionic are very powerful, and come with a lot of different features commonly see in native apps. Luckily, adding them is really simple.

Since we are using AngularJS, we are going to use the [ng-repeat](#) directive to create a new list item for every single task we have in a given project:

## AngularJS n00b?

Never fear! You can pick up the basics with the ever-growing selection of great tutorials on the web. If you like videos, John Lindquist of [egghead.io](http://egghead.io) has a great selection of short, straight-to-the point AngularJS tutorial videos. You can start with Video #1: [AngularJS Binding](#). Matt Frisbie of [Thinkster.io](http://Thinkster.io) also has a [great selection](#) of tutorials.

One of the toughest parts about learning Angular is not knowing "the way" to do certain things. We hope that by providing a great selection of examples and guides for Ionic, you'll pick up on how to write Angular in practice. There is no better way to learn Angular than by building something real!

Keep in mind, when you see tags that look new, like `<ion-content>`, those are custom AngularJS [directives](#) that we have added to Ionic to make it much easier to use Ionic components. They will get expanded by Angular into more lower-level markup, and also controlled by our lower level Javascript controllers that give them increased functionality.

With the list code and the Angular `ng-repeat`, the center content becomes:

```
<!-- Center content -->
<ion-side-menu-content>
  <ion-header-bar class="bar-dark">
    <h1 class="title">Todo</h1>
  </ion-header-bar>
  <ion-content>
    <!-- our list and list items -->
    <ion-list>
      <ion-item ng-repeat="task in tasks">
        {{task.title}}
      </ion-item>
    </ion-list>
  </ion-content>
</ion-side-menu-content>
```

But this doesn't do anything yet, because we don't have any tasks or any code to drive our application. To do this, we need to create an Angular controller and add it to the page. We are going to just use one controller for this app, called `TodoCtrl`. We are going to add it directly to the body tag:

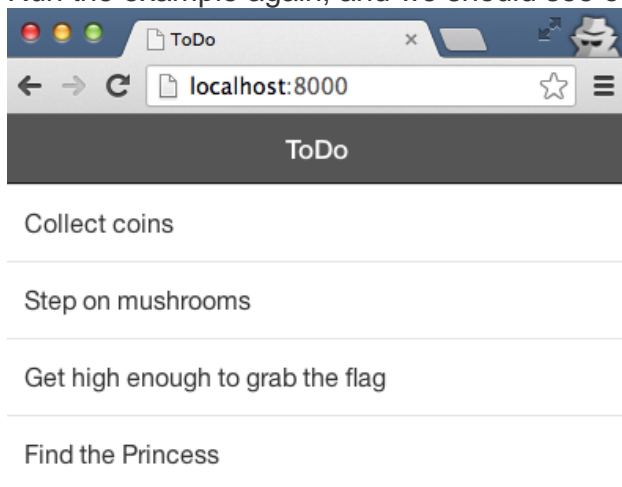
```
<body ng-app="todo" ng-controller="TodoCtrl">
```

Then, we need to define this controller in our `app.js` file, and we can add some testing tasks in:

```
angular.module('todo', ['ionic'])

.controller('TodoCtrl', function($scope) {
  $scope.tasks = [
    { title: 'Collect coins' },
    { title: 'Eat mushrooms' },
    { title: 'Get high enough to grab the flag' },
    { title: 'Find the Princess' }
  ];
});
```

Run the example again, and we should see our list of very important tasks!



# Creating tasks

Okay, so we have some testing data for tasks, but what about creating them? We need some ways to do that. Working with our test data, let's add a simple Modal window that slides up, letting us put in a new task. Place the following script tag after the closing `</ion-side-menu>` tag in the `<body>` of the HTML file:

```
<script id="new-task.html" type="text/ng-template">

  <div class="modal">

    <!-- Modal header bar -->
    <ion-header-bar class="bar-secondary">
      <h1 class="title">New Task</h1>
      <button class="button button-clear button-positive" ng-click="closeNewTask()">
Cancel</button>
    </ion-header-bar>

    <!-- Modal content area -->
    <ion-content>

      <form ng-submit="createTask(task)">
        <div class="list">
          <label class="item item-input">
            <input type="text" placeholder="What do you need to do?" ng-model="task.
title">
          </label>
        </div>
        <div class="padding">
          <button type="submit" class="button button-block button-positive">Create T
ask</button>
        </div>
      </form>

    </ion-content>
```

```
</div>
```

```
</script>
```

There is a lot of information in the above code. First of all, we are defining this template as an Angular template using `<script id="new-task.html" type="text/ng-template">`. The good thing about Angular templates is they can be loaded from anywhere: locally or remote. The URL of the template is the unique identifier, and if the template is defined locally, it will be fetched locally. Templates are a great way to separate layouts and UIs, so we use them extensively.

We then set a header with a button to close the modal, and then set up our content area. For the form, we are calling `createTask(task)` when the form is submitted. The `task` that is passed to `createTask` is the object corresponding to the entered form data. Since our text input has `ng-model="task.title"`, that text input will set the `title` property of the `task` object.

In order to trigger the Modal to open, we need a button in the main header bar and some code to open the modal, the center content then becomes:

```
<!-- Center content -->
<ion-side-menu-content>
  <ion-header-bar class="bar-dark">
    <h1 class="title">Todo</h1>
    <!-- New Task button-->
    <button class="button button-icon" ng-click="newTask()">
      <i class="icon ion-compose"></i>
    </button>
  </ion-header-bar>
  <ion-content>
    <!-- our list and list items -->
    <ion-list>
      <ion-item ng-repeat="task in tasks">
        {{task.title}}
      </ion-item>
    </ion-list>
  </ion-content>
</ion-side-menu-content>
```

And in our controller code:

```
angular.module('todo', ['ionic'])

.controller('TodoCtrl', function($scope, $ionicModal) {
  // No need for testing data anymore
  $scope.tasks = [];

  // Create and Load the Modal
  $ionicModal.fromTemplateUrl('new-task.html', function(modal) {
    $scope.taskModal = modal;
  }, {
    scope: $scope,
    animation: 'slide-in-up'
  });

  // Called when the form is submitted
  $scope.createTask = function(task) {
    $scope.tasks.push({
      title: task.title
    });
    $scope.taskModal.hide();
    task.title = "";
  };

  // Open our new task modal
  $scope.newTask = function() {
    $scope.taskModal.show();
  };

  // Close the new task modal
  $scope.closeNewTask = function() {
    $scope.taskModal.hide();
  };
});
```

Now run the example and try adding a task. It should slide up the modal and then show the new task after submitting it.

## Adding projects

Now we can add support for adding and selecting projects. To do this, we are going to do a lot of the same work we did for the tasks list. We will add a list to display the projects, and a button to add a new project. We are also going to take the chance to abstract away the `Project` model into an angular service that will also handle saving and loading our projects and tasks from localStorage.

We are also going to slip in a few more little things to make the app feel right. Specifically, we've added support for selecting a project (and showing that it's selected), and automatically closing the side menu when creating a new project or selecting an existing one.

Here is the new content area markup:

```
<!-- Center content -->
<ion-side-menu-content>
  <ion-header-bar class="bar-dark">
    <button class="button button-icon" ng-click="toggleProjects()">
      <i class="icon ion-navicon"></i>
    </button>
    <h1 class="title">{{activeProject.title}}</h1>
    <!-- New Task button-->
    <button class="button button-icon" ng-click="newTask()">
      <i class="icon ion-compose"></i>
    </button>
  </ion-header-bar>
  <ion-content scroll="false">
    <ion-list>
      <ion-item ng-repeat="task in activeProject.tasks">
        {{task.title}}
      </ion-item>
    </ion-list>
  </ion-content>
```

```
</ion-side-menu-content>
```

And the new side menu markup:

```
<!-- Left menu -->
<ion-side-menu side="left">
  <ion-header-bar class="bar-dark">
    <h1 class="title">Projects</h1>
    <button class="button button-icon ion-plus" ng-click="newProject()">
  </button>
</ion-header-bar>
<ion-content scroll="false">
  <ion-list>
    <ion-item ng-repeat="project in projects" ng-click="selectProject(project, $
index)" ng-class="{active: activeProject == project}">
      {{project.title}}
    </ion-item>
  </ion-list>
</ion-content>
</ion-side-menu>
```

This adds a side menu of projects, letting us click on each project and also add a new one with a small plus icon button in the header bar. The `ng-class` directive in the `<ion-item>` makes sure to add the `active` class to the currently active project.

To enable adding, saving, and loading projects, we've had to add a bit of code to the controller. Here is the new version of the `app.js` file:

```
angular.module('todo', ['ionic'])
/**
 * The Projects factory handles saving and loading projects
 * from local storage, and also lets us save and load the
 * last active project index.
 */
.factory('Projects', function() {
  return {
    all: function() {
      var projectString = window.localStorage['projects'];
```



```

        if(projectString) {
            return angular.fromJson(projectString);
        }
        return [];
    },
    save: function(projects) {
        window.localStorage['projects'] = angular.toJson(projects);
    },
    newProject: function(projectTitle) {
        // Add a new project
        return {
            title: projectTitle,
            tasks: []
        };
    },
    getLastActiveIndex: function() {
        return parseInt(window.localStorage['lastActiveProject']) || 0;
    },
    setLastActiveIndex: function(index) {
        window.localStorage['lastActiveProject'] = index;
    }
}
}))

.controller('TodoCtrl', function($scope, $timeout, $ionicModal, Projects, $ionicSide
MenuDelegate) {

    // A utility function for creating a new project
    // with the given projectTitle
    var createProject = function(projectTitle) {
        var newProject = Projects.newProject(projectTitle);
        $scope.projects.push(newProject);
        Projects.save($scope.projects);
    };

```

```

    $scope.selectProject(newProject, $scope.projects.length-1);
}

// Load or initialize projects
$scope.projects = Projects.all();

// Grab the last active, or the first project
$scope.activeProject = $scope.projects[Projects.getLastActiveIndex()];

// Called to create a new project
$scope.newProject = function() {
    var projectTitle = prompt('Project name');
    if(projectTitle) {
        createProject(projectTitle);
    }
};

// Called to select the given project
$scope.selectProject = function(project, index) {
    $scope.activeProject = project;
    Projects.setLastActiveIndex(index);
    $ionicSideMenuDelegate.toggleLeft(false);
};

// Create our modal
$ionicModal.fromTemplateUrl('new-task.html', function(modal) {
    $scope.taskModal = modal;
}, {
    scope: $scope
});

$scope.createTask = function(task) {

```

```
if(!$scope.activeProject || !task) {
    return;
}
$scope.activeProject.tasks.push({
    title: task.title
});
$scope.taskModal.hide();

// Inefficient, but save all the projects
Projects.save($scope.projects);

task.title = "";
};

$scope.newTask = function() {
    $scope.taskModal.show();
};

$scope.closeNewTask = function() {
    $scope.taskModal.hide();
}

$scope.toggleProjects = function() {
    $ionicSideMenuDelegate.toggleLeft();
};

// Try to create the first project, make sure to defer
// this by using $timeout so everything is initialized
// properly
$timeout(function() {
    if($scope.projects.length == 0) {
        while(true) {
```

```
var projectTitle = prompt('Your first project title:');  
if(projectTitle) {  
  createProject(projectTitle);  
  break;  
}  
}  
}  
});  
  
});
```

I know, that was a lot of code to jump right to, but it's largely the same concepts from before, just with more details. If you run this version of the app, you should now have a pretty polished and usable multi-project Todo app!

## Chapter 6: Publishing your app

Now that we have a working app, we are ready to push it live to the world! Since [Drifty](#), the creators of Ionic, already submitted the Todo app from this guide to the app store, chances are you'll want to follow this chapter with a new app that you make on your own.

So first, we need to generate a release build of our app, targeted at each platform we wish to deploy on. Before we deploy, we should take care to adjust plugins needed during development that should not be in production mode. For example, we probably don't want the debug console plugin enabled, so we should remove it before generating the release builds:

```
$ cordova plugin rm org.apache.cordova.console
```

## Android Publishing

To generate a release build for Android, we can use the following cordova cli command:

```
$ cordova build --release android
```

This will generate a release build based on the settings in your `config.xml`. Your Ionic app will have preset default values in this file, but if you need to customize how your app is built, you can edit this file to fit your preferences. Check out [the config.xml file](#) documentation for more information.

Next, we can find our *unsigned* APK file in `platforms/android/bin`. In our example, the file was `platforms/android/bin/HelloWorld-release-unsigned.apk`. Now, we need to sign the unsigned APK and run an alignment utility on it to optimize it and prepare it for the app store. If you already have a signing key, skip these steps and use that one instead.

Let's generate our private key using the `keytool` command that comes with the JDK. If this tool isn't found, refer to the [installation guide](#):

```
$ keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

You'll first be prompted to create a password for the keystore. Then, answer the rest of the nice tools's questions and when it's all done, you should have a file called `my-release-key.keystore` created in the current directory.

**Note:** Make sure to save this file somewhere safe, if you lose it you won't be able to submit updates to your app!

To sign the unsigned APK, run the `jarsigner` tool which is also included in the JDK:

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore HelloWorld-release-unsigned.apk alias_name
```

This signs the apk in place. Finally, we need to run the zip align tool to optimize the APK:

```
$ zipalign -v 4 HelloWorld-release-unsigned.apk HelloWorld.apk
```

Now we have our final release binary called `HelloWorld.apk` and we can release this on the Google Play Store for all the world to enjoy!

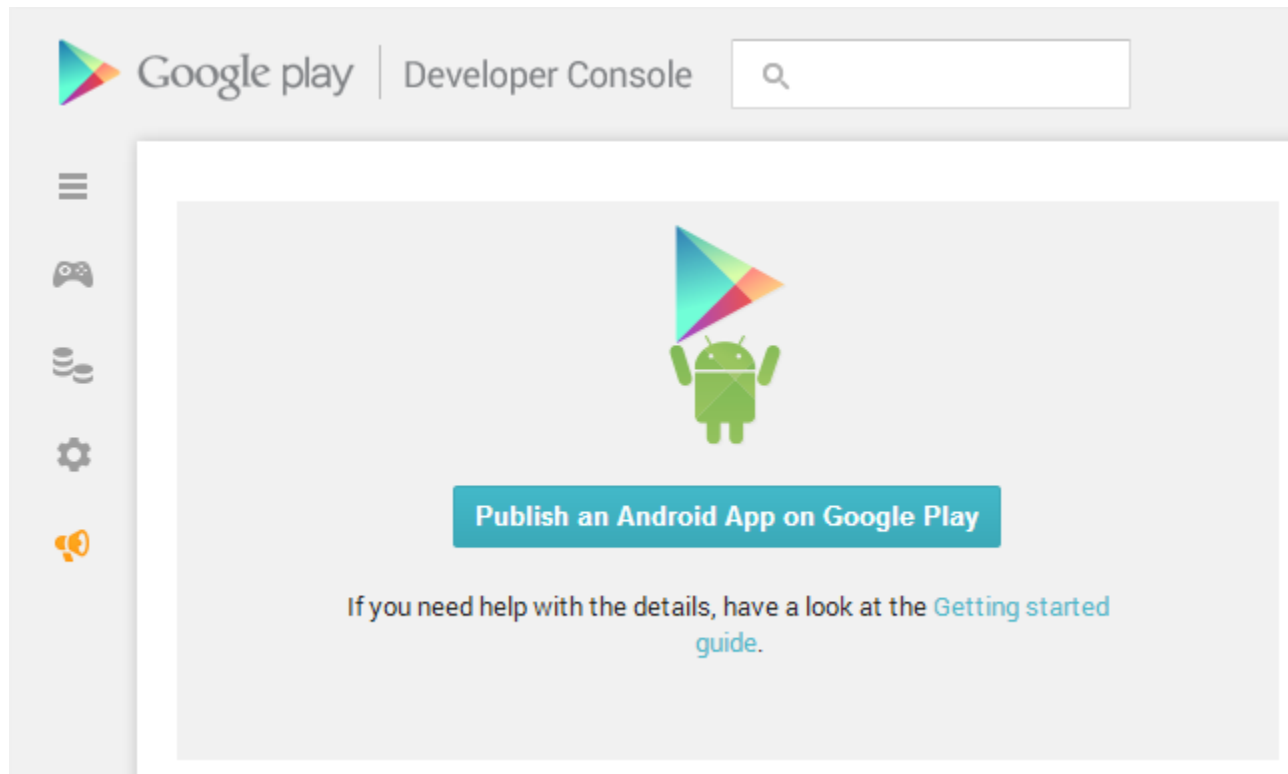
*(There are a few other ways to sign APKs. Refer to the official [Android App Signing](#) documentation for more information.)*

## Google Play Store


Now that we have our release APK ready for the Google Play Store, we can create a Play Store listing and upload our APK.

To start, you'll need to visit the [Google Play Store Developer Console](#) and create a new developer account. Unfortunately, this is not free. However, the cost is only \$25 compared to Apple's \$99.




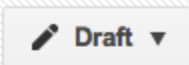
Once you have a developer account, you can go ahead and click "Publish an Android App on Google Play" as in the screenshot below:




Then, you can go ahead and click the button to edit the store listing (We will upload an APK later). You'll want to fill out the description for the app. Here is a little preview from when we filled out the application with the Ionic Todo app:


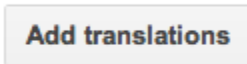


Drifty ▾ Help & Feedback ▾

 IONIC TODO 

STORE LISTING 

PRODUCT DETAILS Fields marked with \* need to be filled before publishing.

 English (United States) – en-US 

**Title \***  
English (United States) – en-US   
10 of 30 characters

**Description \***  
English (United States) – en-US 

Ionic Todo is a simple multi-project Todo app built as a demo of the open source Ionic Framework for building cross-platform HTML5 mobile apps.

  
143 of 4000 characters

Please check out these [tips on how to create policy compliant app descriptions](#) to avoid some common reasons for app suspension.

When you are ready, upload the APK for the release build and publish the listing. Be patient and your hard work should be live in the wild!

## Updating your App

As you develop your app, you'll want to update it periodically.



In order for the Google Play Store to accept updated APKs, you'll need to edit the `platforms/android/AndroidManifest.xml` file to increment the `android:versionCode` value.