[Chapter 1: About Ionic](#_r6ww5xf03od)

[iOS & Android Mobile Application Development in a nutshell](#_og49w7zcfvt)

[Native Development](#_o5c65bexmiim)

[Hybrid Development](#_290e5vracqu2)

[Ionic Framework](#_6u6t4v93d4ho)

[Progressive Web Apps and the future of Mobile Application Development](#_jai6og9s5wae)

[Why Ionic?](#_g3cg8r5ie3tm)

[Chapter 2: Ionic Framework Installation & Starting A Project](#_8r1fwdx2pzc0)

[What do you need?](#_rvda6xm0e1z2)

[How to install Ionic?](#_2loxifubmuzv)

[Starting a new Ionic Project](#_783e3zpjpszb)

[Trying out your app - ionic serve](#_itz2qvkt1lga)

[Chapter 3: Creating Pages and Navigation](#_af1ar1r1ev1d)

[How to create a page?](#_kyljd0pcgpqp)

[How to navigate to a page?](#_kj75xvxoohto)

[Navigate via HTML](#_nteef4crzpfi)

[Navigate via TypeScript](#_yc00yslbjm2r)

[Navigate via Modal](#_fij6fmafl4ho)

[Chapter 4: Service, Components and Pipes](#_d7fkktxdhlfd)

[What is Services?](#_hfyddgtz0j9t)

[Using Services](#_zhyd1bmytyxb)

[What is Component?](#_4lpfleevjgud)

[Using Component](#_baz3ugusqup9)

[What is Pipes?](#_cz70tqv3pchl)

[How to use Pipes?](#_gpd7bt5ppkwq)

[Chapter 5: Basic Angular Commands](#_6mzfb92kpo3e)

[(click)](#_1tod4wkh70eu)

[[(ngModel)]](#_s8ws90i6x61m)

[For Loop & \*ngFor](#_poq82po3yri2)

[If...else if...else Statement & \*ngIf](#_h8ktjy46iomr)

[</div>](#_2al6w7rzfrg9)

[Chapter 6: Theming, Assets & LocalStorage](#_h2j8e8rr1b29)

[Theming your app](#_7krcxxxa44ml)

[How to use the colours in your app?](#_tyru3eqaszc2)

[Assets](#_prmsq0mp1dio)

[How to use locally based assets in your application](#_jiafcowt47z2)

[ADDITIONAL MODULE: Developer Setup Requirement, Android and iOS](#_28kbq2y37lwx)

[ANDROID](#_35fbtvqnnv2z)

[iOS](#_1cojlpehxekj)

[Connecting Xcode with your developer account](#_gqr15dtf1rbd)

[Signing](#_emnpigcyxj1e)

[Setting up the app identifier](#_vvf35h5huio8)

[Creating the app listing](#_1esgxzlr4tp1)

[Building the app for production](#_tdq3vjgs7ohd)

[Opening the project in Xcode](#_hxpj9kfmrksq)

[Creating an archive of the application](#_thd3ldv28ci2)

[Finishing the app list process](#_nwu0a76oscik)

[Updating the app](#_s5loatpxjbiu)

[ADDITIONAL MODULE: Page Navigation Method](#_17l6e5z1ngm2)

[What Is Angular Routing?](#_3klb9pttbn72)

[ADDITIONAL MODULE: Firebase](#_ibeuajd6vjc9)

[Creating the Firebase App](#_9sw7rubdmww)

[Starting our Ionic App & Firebase Integration](#_h4fsqtk1kpo)

[Creating a Firebase Data Service](#_ieb1sqok5k7i)

[Displaying a Firestore Collection List](#_bt26iq5peq4j)

[Working with a Firestore Document](#_3tnyituycgzt)

[Conclusion](#_dmejivd23xql)

[ADDITIONAL MODULE: Local Storage CRUD Services](#_8tk9nhgno87d)

[Starting our CRUD App](#_yy3eqe8jh43i)

[CRUD Functions for Ionic Storage](#_rbthwglyj777)

[Working with our Storage Service](#_flg0blu3f77k)

[Conclusion](#_o0da47k5t2h0)

[ADDITIONAL MODULE: Build Using Capacitor](#_8m28nl65nizb)

[Existing Ionic Project](#_c3eest3npx5u)

[Initialize Capacitor with your app information](#_my7ifevzkjbp)

[Build your Ionic App](#_wspi6ln027u0)

[Add Platforms](#_s7i9ggq4tvtk)

[Open IDE to build, run, and deploy](#_2zpke8mcqjd8)

[Syncing your app with Capacitor](#_wx488r82xxit)

[ADDITIONAL MODULE: MINIMUM SPEC](#_c1uxbtf986h5)

[1. Make use of Ionic/Angular](#_7bk13wiv6byk)

[2. Ahead-of-Time compiling --prod for ios --release for android](#_velabigpwu22)

[3. Minimise system intensive processes](#_m0b044v3inl5)

[4. Use DOM rendering sparingly](#_aoi36qlepo9x)

[5. Collect your garbage](#_rt78grptji44)

[6. Data persistence](#_31rvrs2qn8lb)

[7. Manage network requests](#_a5g9ehfl26mg)

[8. Efficiently handle media content](#_nkocy03rljog)

[ADDITIONAL MODULE: GOOGLE MAPS INSTALLATION](#_mrzxb7vd5wsj)

[ADDITIONAL MODULE: IONIC PWA](#_x7bdhbubgjsa)

[ADDITIONAL MODULE: USING NETLIFY AS DEV OPS](#_vvb0wo2zxr1p)

[AT NETLIFY](#_nvne4mmsfy65)

[AT IONIC CLI](#_9c705p1rc7g2)

[AT CODE ROUTING](#_pxnlz4pgi8s7)

[ADDITIONAL MODULE: ChartJS](#_s0iwjabqxnro)

[ADDITIONAL MODULE: OneSignal Push Notification](#_hoagg8f18e4j)

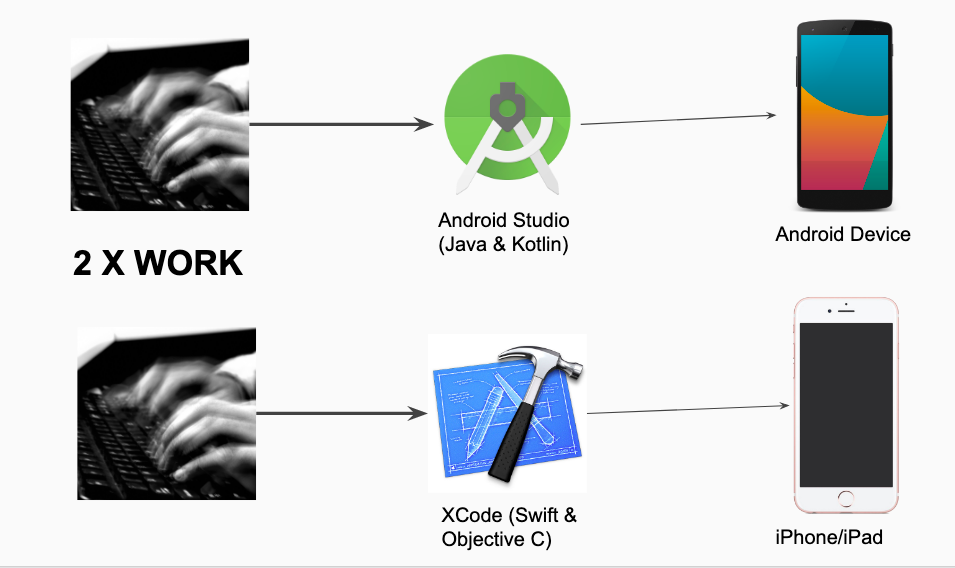
# 

# Chapter 1: About Ionic

## iOS & Android Mobile Application Development in a nutshell

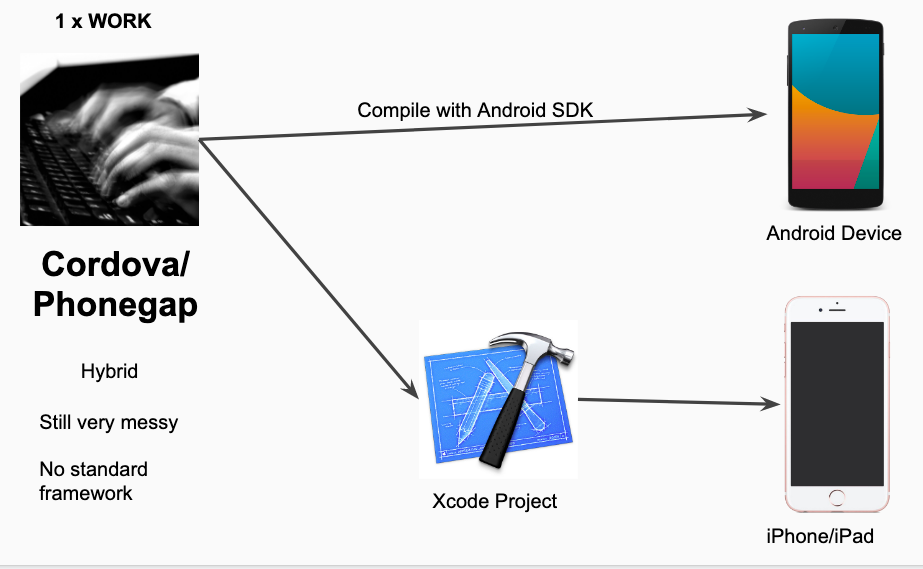
### Native Development

In order to build a native mobile application, it would require 2 different distinct codebase to develop for either Android or XCode:



### Hybrid Development

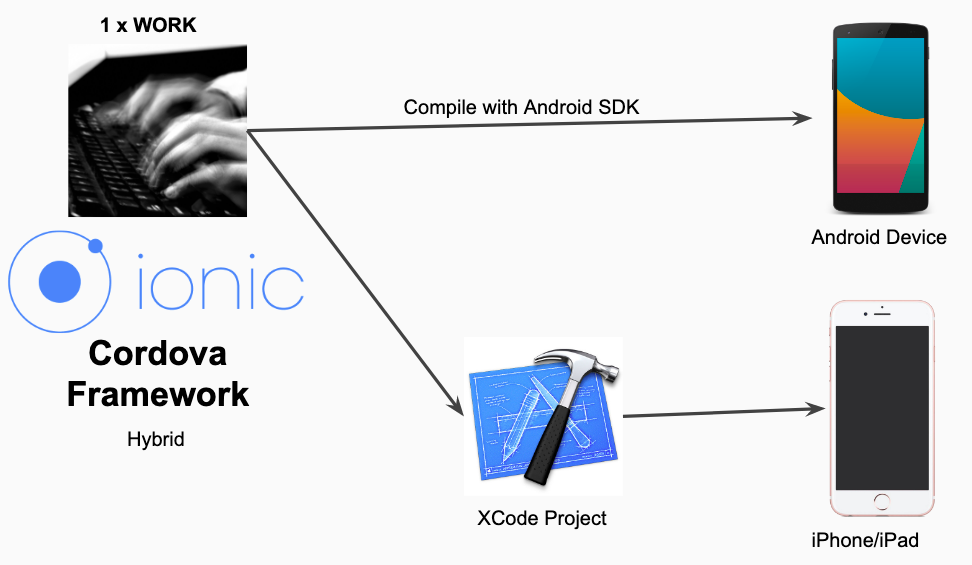
Hybrid development came into the picture to assist mobile app developers to build mobile applications with one single codebase. By creating a web layer on top of a native layer, developers are able to use knowledge in web development to develop mobile applications. However, this was not enough, the codes are very messy because there is no standard framework to control the device capabilities and theme the applications effectively. Early hybrid mobile applications were also ‘slow’ (Please note: differences in micro seconds)



### Ionic Framework

Ionic Framework is a hybrid framework that provides a standard framework to assist developers to effectively build mobile applications with a standard set of user interface controls and components and native access to devices’ API, such as cameras, vibration and device controls in a single codebase.

Using Angular framework (which has since been expanded to support many JavaScript/TypeScript based UI frameworks) as the basis for the controllers and HTML conventions, developers are able to build fully fledged mobile application with performance on par with natively develop mobile applications



### Progressive Web Apps and the future of Mobile Application Development

Being a web based hybrid framework, an Ionic Framework project can easily be converted into a Progressive Web Application (PWA). This would make it possible for web apps to be developed with Ionic Framework and installed directly to user’s devices, bypassing the Apple App Store and Google Play Store.

With the addition of the Capacitor framework (a new framework to assist Ionic developers, as an alternative to Cordova, by the Ionic team), users are able to build mobile application not only for iOS and Android, but a proper computer software application as well.

### Why Ionic?

* Framework specifically to develop mobile apps across multiple platforms
* ’One codebase to rule them all’
* Apps development using HTML, CSS, TypeScript (a subset of JavaScript)
* Ability to use plugins such as cameras, bluetooth and device controls, all of this available to be referred to at ionicframework.com

# Chapter 2: Ionic Framework Installation & Starting A Project

## What do you need?

|  |  |
| --- | --- |
| **To start/run Ionic**   * NodeJS (see website, LTS version) * Node Package Manager (NPM) * Ionic (installed via NPM, see below)   **To turn Ionic project into Android Application**   * Android Studio * Java Runtime Environment (JRE) 1.8 * Java Development Kit (JDK) 1.8   **To turn Ionic project into iOS Application**   * Xcode | **To edit/develop**   * Any Integrated Development Environment (IDE), example: VSCode, Atom, Sublime etc. Please install TypeScript plugin if you are using Atom or Sublime * Google Chrome Web Browser * Ionic DevApp * GitLab (All codes for this course will be made available here) |

## How to install Ionic?

1. Download NodeJS from <http://www.nodejs.org> . Get the LTS version of Node. LTS stands for 'Long Term Support'.
2. Open command prompt
   1. Windows: Start button-> Search-> type 'cmd'
   2. Mac: Search for 'terminal'
3. Type the following:
   1. node -v
   2. npm -v
4. If both command responds with version numbers you are all set to install Ionic. To install Ionic, type the following:
   1. npm install -g ionic cordova
5. If this is successful, type the following, there should be a response:
   1. ionic -v
6. If all the above steps are completed, you are ready to start building a mobile app with the Ionic Framework

## Starting a new Ionic Project

1. In order to start an Ionic project, navigate to the folder with 'cd' of where you want your Ionic project to be
2. Type the following:
   1. ionic start project\_name
   2. \*Please note there is no spaces allowed in the project name. Changing the app name will be done later, but this is the project name.
3. You will be presented with several project types. For details of the project types, please refer to the next section.
4. After all the steps above is done, you should have your new project folder project\_name

## Trying out your app - ionic serve

1. You can try to run your app to see what it looks like
2. 'cd' into your project folder and run the following command:
   1. ionic serve
3. This will automatically launch a web browser with your project application
4. If the web browser is Google Chrome, right click and select 'Inspect'
5. Press this button to view the application in a mobile phone format of your choice

# Chapter 3: Creating Pages and Navigation

Referring to the codes from the previous application created in the previous chapter, Ionic looks and feel like a typical web project. It uses pages and each page has its own html and typescript functions.

## How to create a page?

1. To create a page, inside the project folder, you will need to type the following:
   1. ionic g page name
   2. \*Please note, similar to the app name, the page name must not have any spaces, must not include the word 'page'
2. This will generate a new page folder inside your project.
3. You can 'scaffold' the pages by using '/' operator, example:
   1. ionic g page start/abcd
4. This will generate ‘abcd’ page in the start folder, by ‘scaffolding’ your application, you can organise the pages efficiently

## How to navigate to a page?

There are 3 methods to navigate to a page:

* 1. Via HTML
  2. Via TypeScript commands
  3. With a Modal Controller

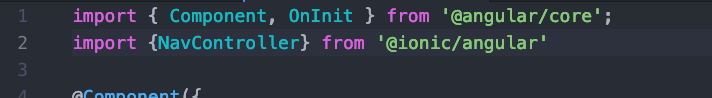
The following are instructions on how to navigate to a page AFTER the page has been created (refer to the previous section)

#### Navigate via HTML

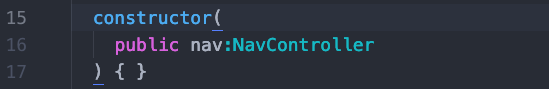
1. At the button or desired HTML component to be used to navigate to another page, insert the following snippet inside the html, in this example we are going to use a button to move between pages:
   1. <ion-button routerDirection=”forward” routerLink=”/first’>Go to First Page</ion-button>
2. This will bring you to the next page using html. However, since it is done in HTML, you can’t place conditions on the code, or do any processes to ensure the navigation is done correctly. For this, you will need to use the TypeScript method.

#### Navigate via TypeScript

1. The first thing you need to do is add the following code inside the desired page.ts:



1. Next you will need to add into the constructor brackets the following line of code:



1. After this is done, you can use the nav command with the following line of code in a function that you want to use to navigate back, in this example, to navigate to a page called home:



1. With the above code you are able to navigate forward to a page called home. You can also use other commands such as navigateBack and navigateRoot

#### Navigate via Modal

In some cases you will want to create a modal. A modal is a page that is opened above an existing page that is still ‘alive’. This can be done in TypeScript part of any page.



1. In order to navigate to a page via Modal, you will need to import the modal page module into the app module. For this example, I will be calling my page new page modal. Open app.module.ts and add the following code at the top of the page:



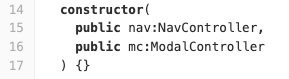
1. You will then need to add the page together with the rest of the modules inside the app.module.ts page:



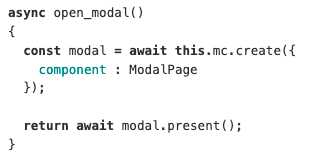
1. Next, you will need to import Modal Page inside the page where you will call this particular modal page. In this example, I will add it in a page called home.ts. I will also need to import ModalController, similar to the way how we imported the NavController



1. I will need to declare the ModalController in the constructor, similar on how we declared this with our NavController



1. And then, we can use it in any of our functions. In this case, I am going to use it in a function called open\_modal():

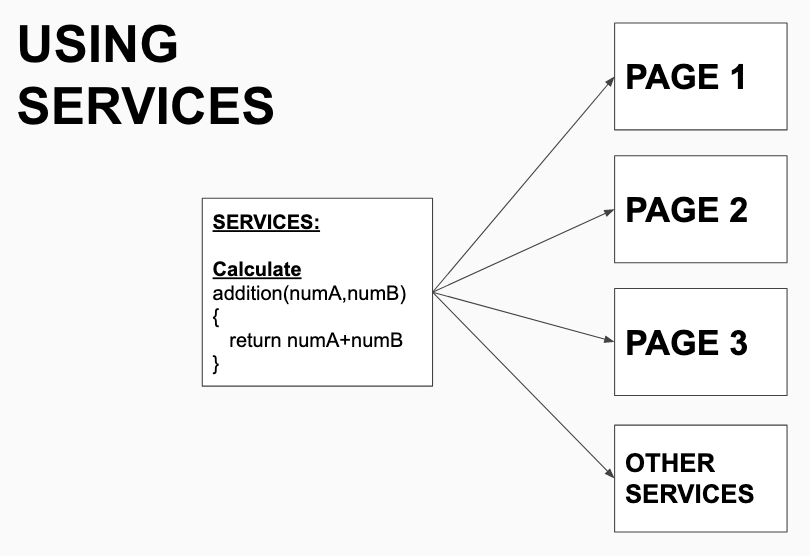


# Chapter 4: Service, Components and Pipes

In addition to navigation, it is useful to learn about Services, Components and Pipes and how to use them when building an Ionic project.

## What is Services?

Services is particularly useful when creating a mobile application because it allows the developer to place commonly used functions in one single place.



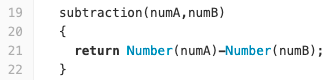
In this example, the function addition is in a service called calculate. It is being used by Page 1, Page 2, Page 3 as well as other services.

After identifying a particular, commonly used, function, the function can be placed in a service and be used anywhere in the application. The advantage here is that any changes can be done at the service itself, rather than on the individual page where the same function is written multiple times.

Services can be used globally, repeatedly and by different pages as well as other services

## Using Services

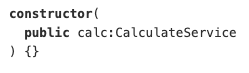
1. Generate a service inside your application by using the following command. In this example, I will be creating a service for calculate:
   1. ionic g service service/calculate
2. This will create a service called calculate in the folder service.
3. You can then add a function into the service. In this example, lets create a service called subtraction to take two values and return the result of subtraction of the two values.



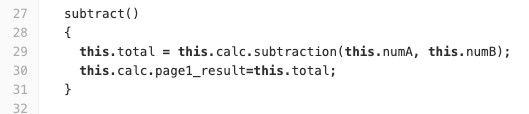
1. After this is done, we can use this in any page to call the subtraction function, for example to call it in a home page, we will need to first declare the calculate service:



1. Declare calculate service in the constructor



1. Finally after this is done, you will be able to use it in any function in your application



## What is Component?

A component is an independent construct which encapsulates the controller code, the view and styles,and has inputs and outputs.

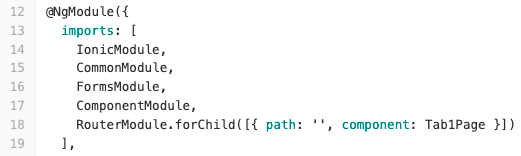
## Using Component

1. Generate a component with the following command. In this example, we will generate a component called Title component:
   1. ionic g component component/title
2. This will generate a component called title in the component folder. You can then edit the component HTML and ts files.
   1. Please take note of the .ts title in the app inside the selector .ts file
3. In order to use the title component, we will need to create a component module. Create a new file called component.module.ts



1. You will then need to add the file into a local module file. In this example, we will add the component inside the home.module.ts file. We will import it and add inside the list of imports inside the home.module.ts page





1. After adding the file into the home.module.ts page, it is then possible to use the component inside of your app with the following command inside the home.page.html



## What is Pipes?

Similar to components, pipes is an independent construct to filter variables when it is presented inside the app's html files using the curly braces.

## How to use Pipes?

1. Similar to components, we will need to generate a new pipe for the project
2. Import the pipes into the pages own module file
3. Add the pipe into declarations, under the home.module.ts file
4. Use the custom pipe like this: {{ abc | mypipe }}

# 

# Chapter 5: Basic Angular Commands

## (click)

As seen in the previous chapter, (click) will be used to execute any function that was written in the pages own TypeScript, by inserting (click) in the HTML part of the tag and the function name.

For example:

IN TYPESCRIPT:

fireAlert()

{

alert(“FIRE”);

}

IN HTML:

<ion-button (click)=”fireAlert()”>

FIRE

</ion-button>

When the button is pressed, it will execute the fireAlert() function inside the TypeScript file.

## [(ngModel)]

This is used as a link to capture what is typed into the input box in the TypeScript. A variable need to be first declared inside the TypeScript in order to use [(ngModel)] to ‘capture’ the value.

For example:

IN TYPESCRIPT:

User\_xyz:any;

IN HTML:

<ion-input [(ngModel)]= “user\_xyz”>

</ion-input>

## For Loop & \*ngFor

The for loop method is used inside TypeScript, for a conditional loop, ie. stop after doing it x amount of times. This is particularly useful when ‘looping’ through an array.

The method of using for loop inside TypeScript is exactly the same as using for loop in JavaScript. The only difference is there is a new method other than the traditional for loop method.

**Method 1: For Loop--All in TypeScript**

the\_array[0]= “ayam”

the\_array[1]= “lembu”

the\_array[2]= “kerbau”

for{var x=0;x<4;x++)

{

console.log(the\_array[x]);

}

Ouput:

ayam

lembu

Kerbau

**Method 2: For Loop--All in TypeScript**

the\_array[0]= “ayam”

the\_array[1]= “lembu”

the\_array[2]= “kerbau”

for{let item of the\_array)

{

console.log(item);

}

Ouput:

ayam

lembu

Kerbau

\*ngFor is used inside HTML to loop through a list of items quickly and easily. The list would reside in the form of an array inside the TypeScript. Using \*ngFor loop, it is possible to output the whole list.

TypeScript:

the\_array[0]= “ayam”

the\_array[1]= “lembu”

the\_array[2]= “kerbau”

HTML:

<ion-item \*ngFor=”let item of the\_array”>

{{item}}

</ion-item>

This will output all the animals in the\_array.

## If...else if...else Statement & \*ngIf

If else statement is used to filter, search and select a particular item. It is used in the TypeScript exactly the same way one would use it in JavaScript

Example:

if(word\_x== “duck”)

{

console.log(“quack”);

}

else if(word\_x== “cow”)

{

console.log(“moo”);

}

else

{

console.log(“NO ANIMALS”);

}

\*ngIf is used to inside HTML to filter variables similar to how if else statement is used inside the TypeScript

IN HTML

<div \*ngIf= “ word\_x==’cow’ ”>

MOO!

</div>

<div \*ngIf= “ word\_x==’duck’ ”>

QUACK!

</div>

<div \*ngIf= “ word\_x==null ”>

NO ANIMALS

# </div>

# Chapter 6: Theming, Assets & LocalStorage

## Theming your app

Ionic Framework is built to be a blank slate that can easily be customized and modified to fit a brand, while still following the standards of the different platforms.

Theming Ionic apps is now easier than ever. Because the framework is built with CSS, it comes with pre-baked default styles which are extremely easy to change and modify.

There are 9 colours already pre baked inside Ionic that can be used straight away. The 9 colours can also be changed by accessing the ionicframework.com website under the colour generator part of the website <https://ionicframework.com/docs/theming/basics>

## How to use the colours in your app?

The colours can be used in the app through the "color" operator.

1. For example, if you would like to change the colour of a button, this can be done with the following line of code:
   1. <ion-button color="success"> Button with different colour </ion-button>
2. This can also be applied to many other ionic properties throughout your app, such as the ion-toolbar etc

## Assets

In certain cases, where you will need to place local assets such as pictures and videos into your applications, you can place these local files inside the assets folder. The assets folder can be located under your src/app folder

## How to use locally based assets in your application

1. After placing the file inside the assets folder, the file can be used in your app with the following command, in this example, an image with the name abc.png
   1. <img src="assets/abc.png">
2. Files in assets can be placed into folders to organise and arrange them effectively.

# ADDITIONAL MODULE: Developer Setup Requirement, Android and iOS

## ANDROID

**\*\*After generating an APK from previous lesson, the following is the next step to prepare and publish the application for ANDROID**

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

*$ ionic 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/build/outputs/apk. In our example, the file was platforms/android/build/outputs/apk/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. The zipalign tool can be found in /path/to/Android/sdk/build-tools/VERSION/zipalign. For example, on OS X with Android Studio installed, zipalign is in ~/Library/Android/sdk/build-tools/VERSION/zipalign:

$ 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!

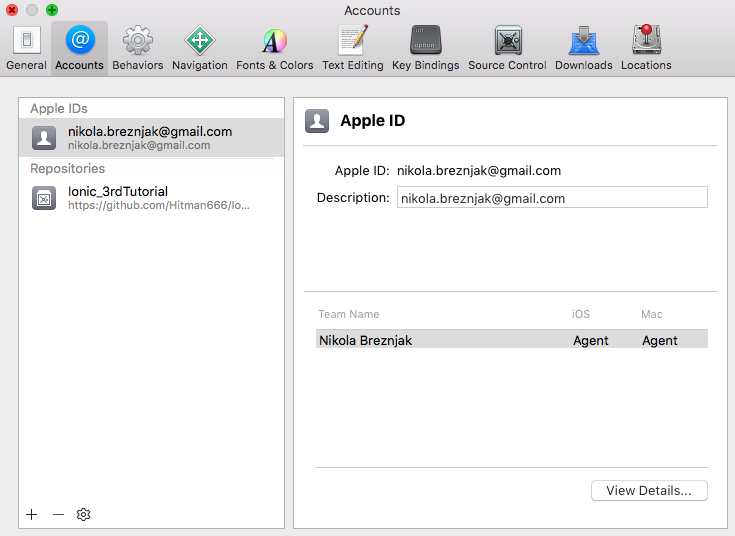
(There are a few other ways to sign APKs. Refer to the official Android App Signing documentation for more information.)

## iOS

First, you need to enroll in [Apple Developer Program](https://developer.apple.com/programs/). As with Google, if you have a personal account with Apple, you can create an additional one for your applications.

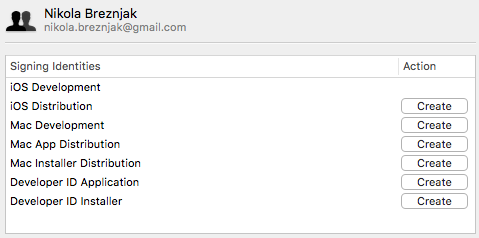
### Connecting Xcode with your developer account

After you receive your developer status, open Xcode on your Mac and go to Preferences -> Accounts and add your account to Xcode by clicking the + button on the lower left hand side, and follow the instructions:



## Signing

Now that you linked Xcode with your developer account, go to Preferences -> Accounts, select your Apple Id on the left hand side and then click the View Details button shown on the previous image. You should see the popup similar to the one on the image below:



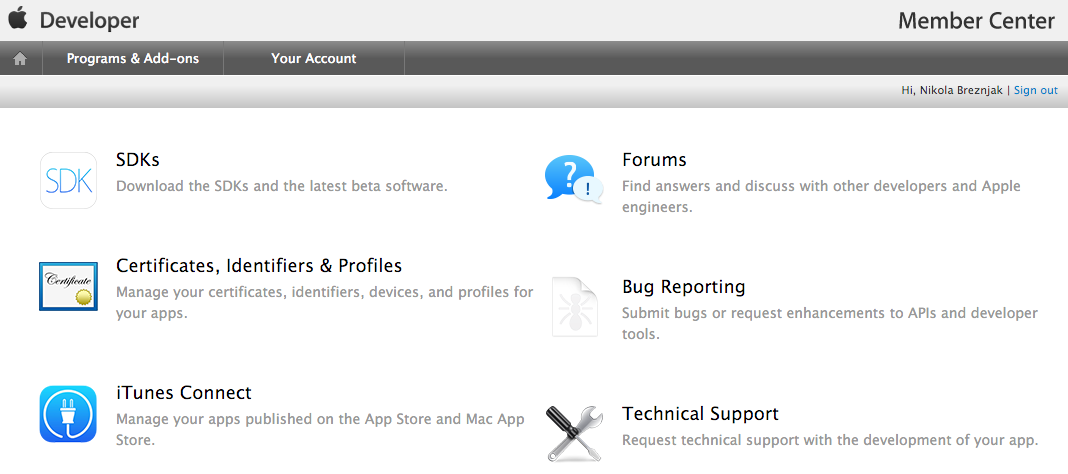
Click the Create button next to the iOS Distribution option.

You can learn more about maintaining your signing identities and certificates from the official documentation

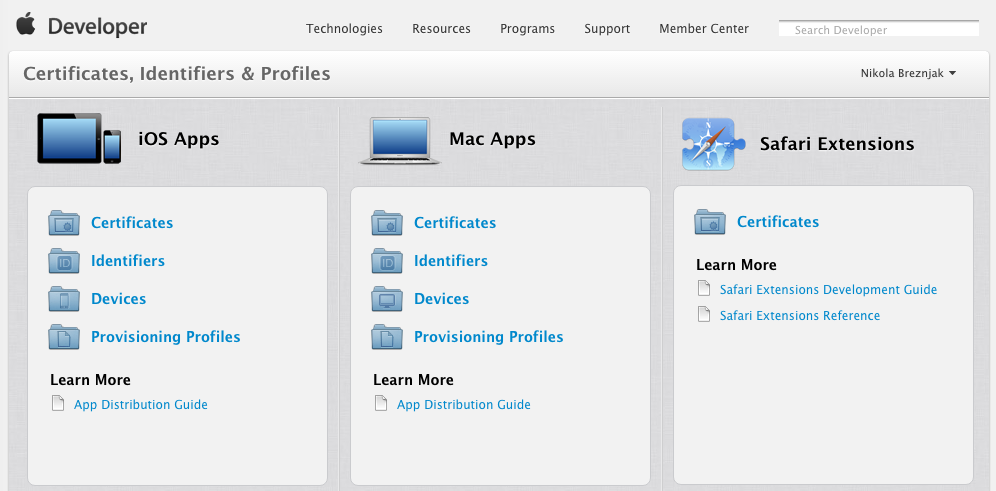
## Setting up the app identifier

Next, through the Apple Developer Member Center we’ll set up the app ID identifier details. Identifiers are used to allow an app to have access to certain app services like for example Apple Pay. You can login to Apple Developer Member Center with your Apple ID and password.

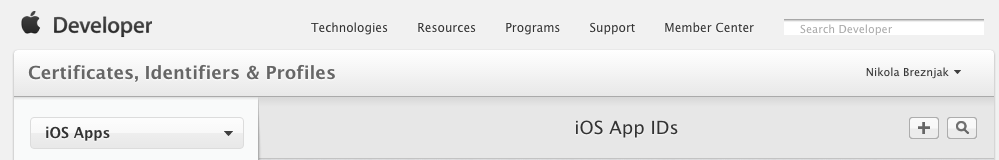
Once you’re logged in you should choose Certificates, Identifiers, and Profiles option as shown on the image below:



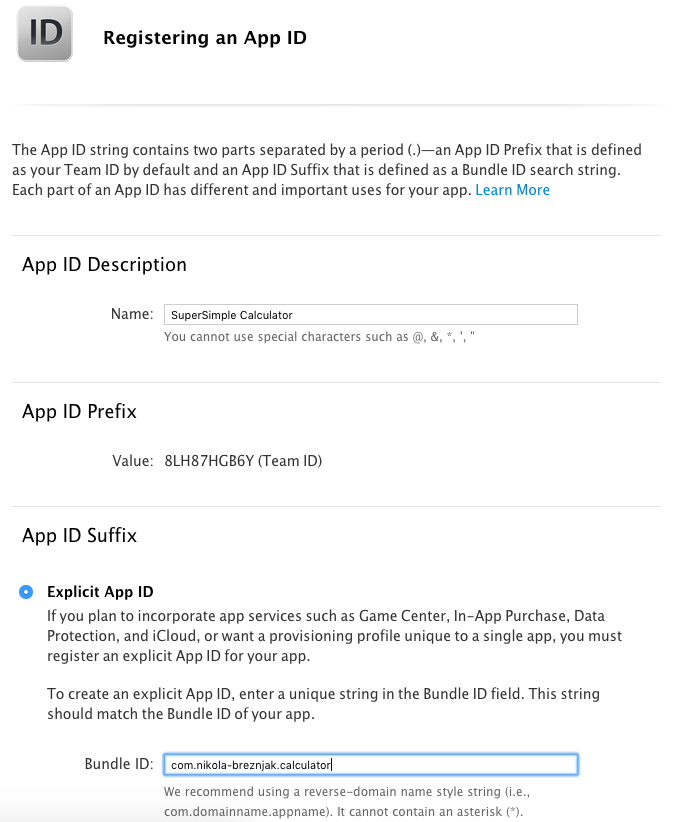
On the next screen, shown on the image below, select the Identifiers option under the iOS Apps.



On the next screen, shown on the image below, select the plus (+) button in order to add a new iOS App ID.



On the next screen, shown partialy on the image below, you’ll have to set the name of your app, and use the Explicit App ID option and set the Bundle ID to the value of the id in your Cordova config.xml tag.

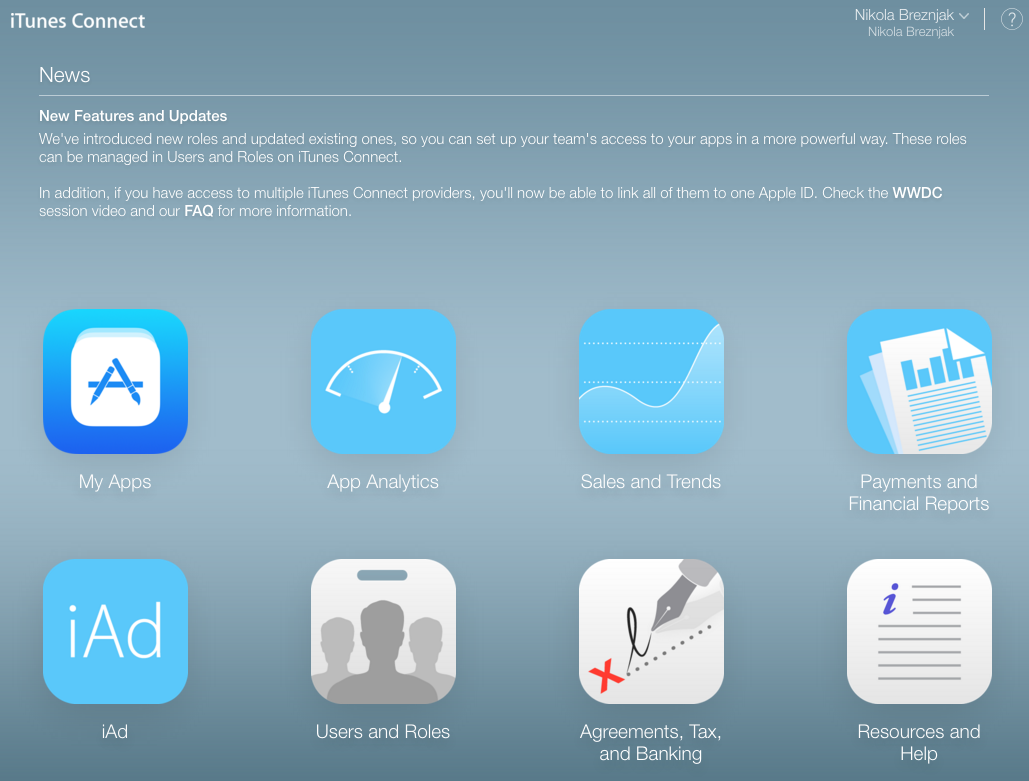


Additionally, you’ll have to choose any of the services that need to be enabled. For example, if you use Apple Pay or Wallet in your app, you need to choose those option.

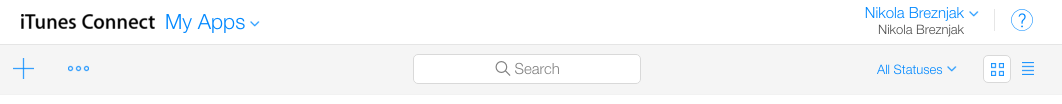
You can learn more about registering app identifiers from the [official documentation](https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html).

## Creating the app listing

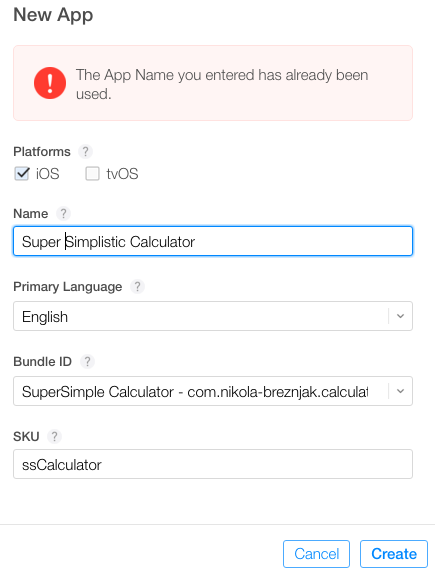
Apple uses [iTunes Connect](https://itunesconnect.apple.com/) to manage app submissions. After you login, you should see a screen similar to the one on the image below:



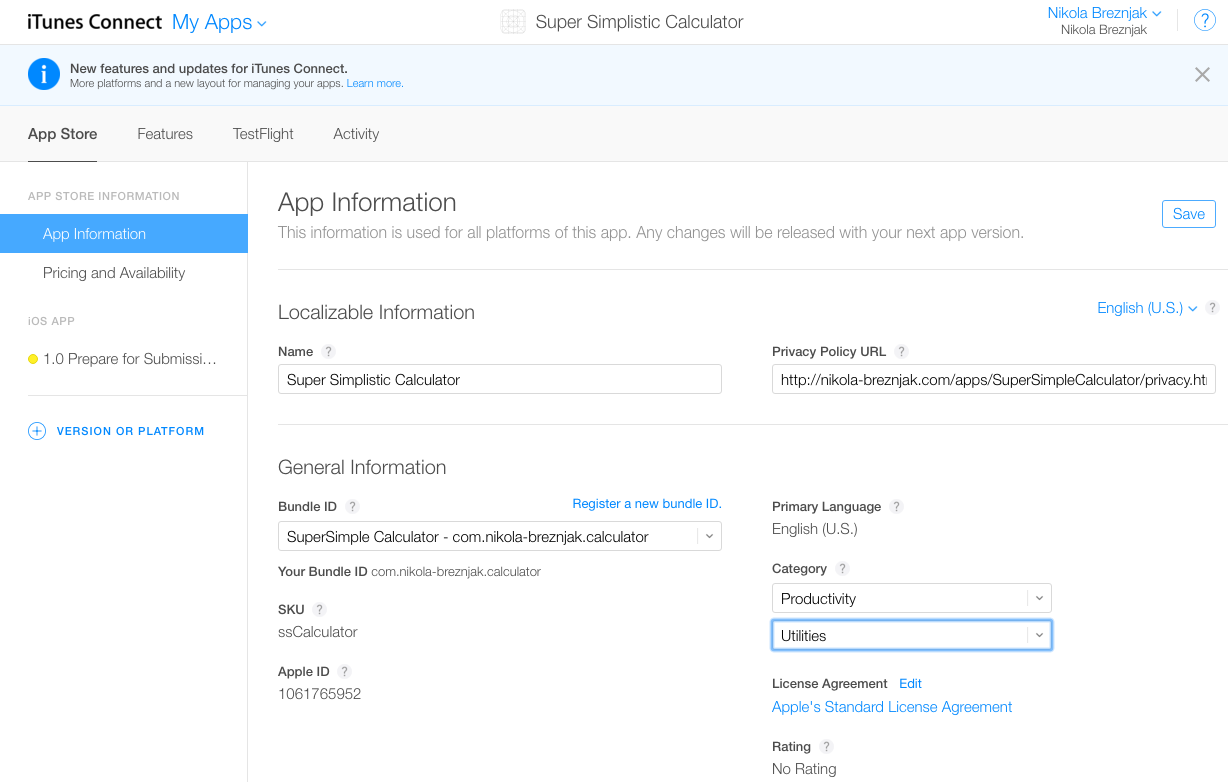
Here you have to select the My Apps button, and on the next screen select the + button, just below the iTunes Conenct My Apps header, as shown on the image below:



This will show three options in a dropdown, and you should select the New App. After this the popup appears, as shown on the image below, where you have to choose the name of the application, platform, primary language, bundle ID and SKU.



Once you’re done, click on the Create button and you’ll be presented with the following screen where you’ll have to set some basic options like Privacy Policy URL, category and sub category.



Now, before we fill out everything in the listing, we’ll build our app and get it uploaded with Xcode. Then you’ll come back to finish the listing.

You can learn more about managing your app in iTunes Connect from the [official documentation](https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/UsingiTunesConnect/UsingiTunesConnect.html).

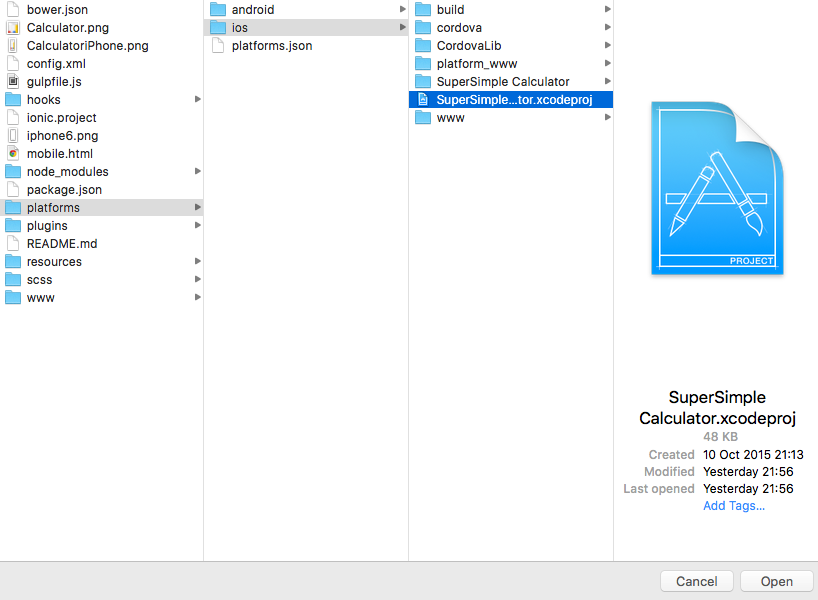
## Building the app for production

In the root directory of your application execute the following command: ionic cordova build ios --release

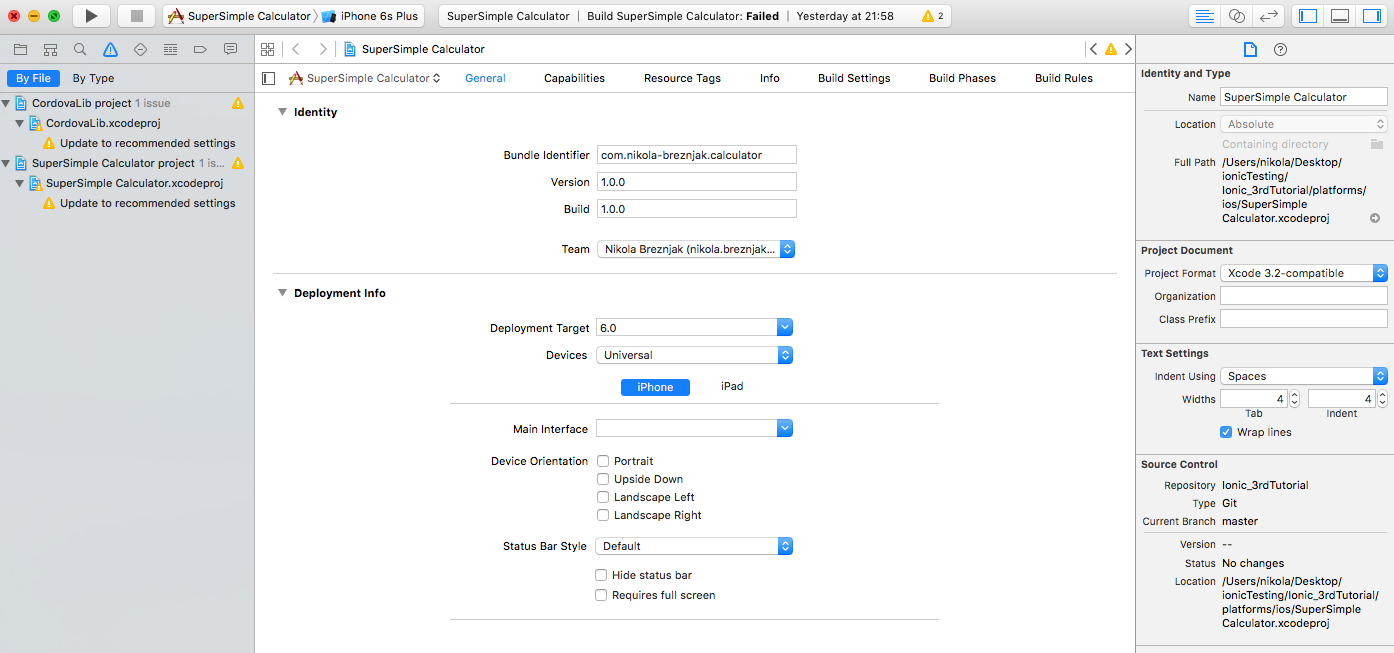
If everything went well you’ll see the **BUILD SUCCEEDED** output in the console.

## Opening the project in Xcode

Now, open the platforms/ios/SuperSimpleCalculator.xcodeproj file in Xcode (of course you would change SuperSimpleCalculatorwith your own name).



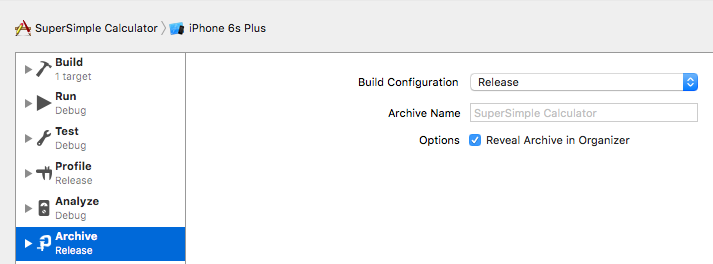
Once the Xcode opens up the project, you should see the details about your app in the general view, as shown on the image below:



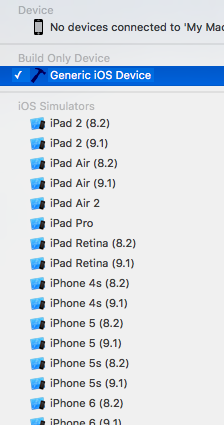
You should just check that the bundle identifier is set up correctly, so that it’s the same as the value you specified earlier in the app ID. Also, make sure that the version and build numbers are correct. Team option should be set to your Apple developer account. Under the deployment target you can choose which devices your application will support.

## Creating an archive of the application

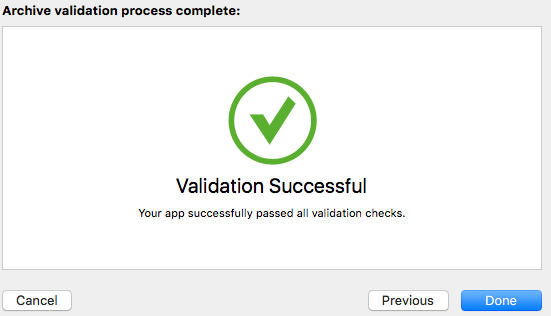
In Xcode, select Product -> Scheme -> Edit Scheme to open the scheme editor. Next, select the Archive from the list on the left hand side. Make sure that the Build configuration is set to Release as shown on the image below:



To create an archive choose a Generic iOS Device, or your device if it’s connected to your Mac (you can’t create an archive if simulator is selected), from the Scheme toolbar menu in the project editor, as shown on the image below:



Next, select Product -> Archive, and the Archive organizer appears and displays the new archive.

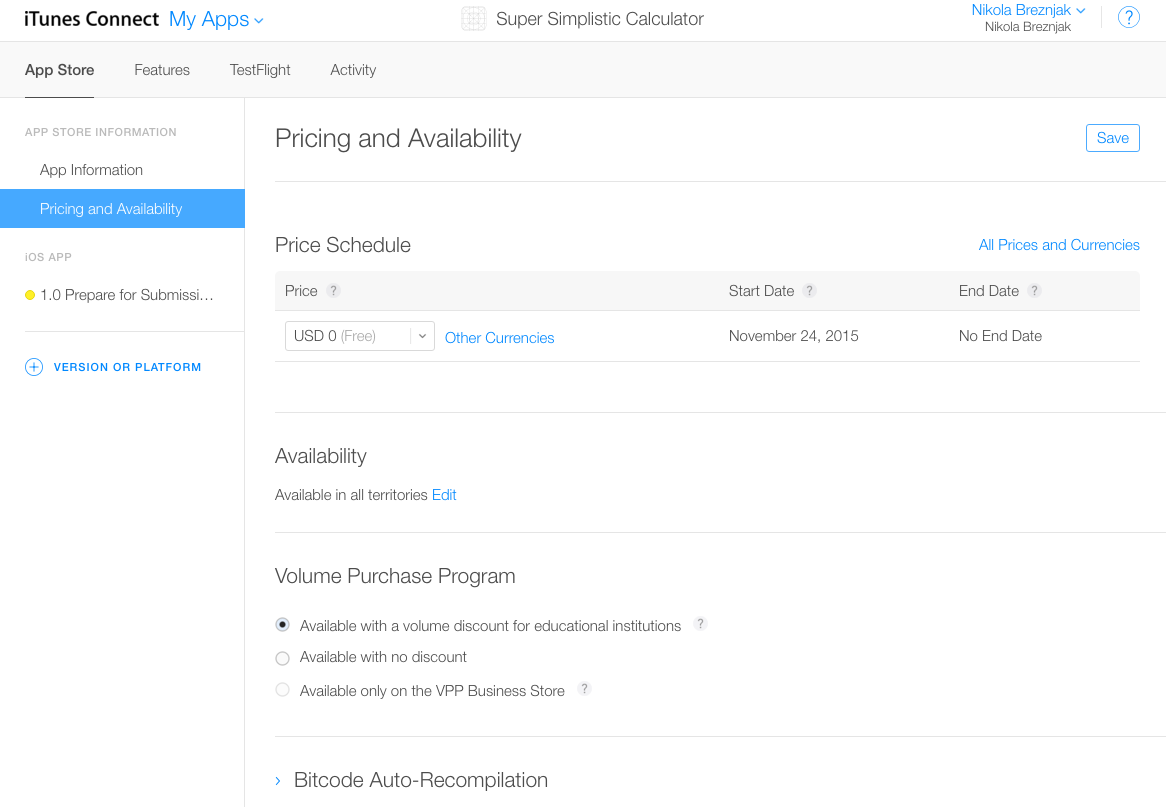


At this point you can click the Upload to App Store... button, and if everything goes fine you’ll have an uploaded app, and the only thing that’s left to do is to complete the iTunes Connect listing and submit it for review!

If you get an email from iTunes Connect shortly after you uploaded the archive with the content similar to this:

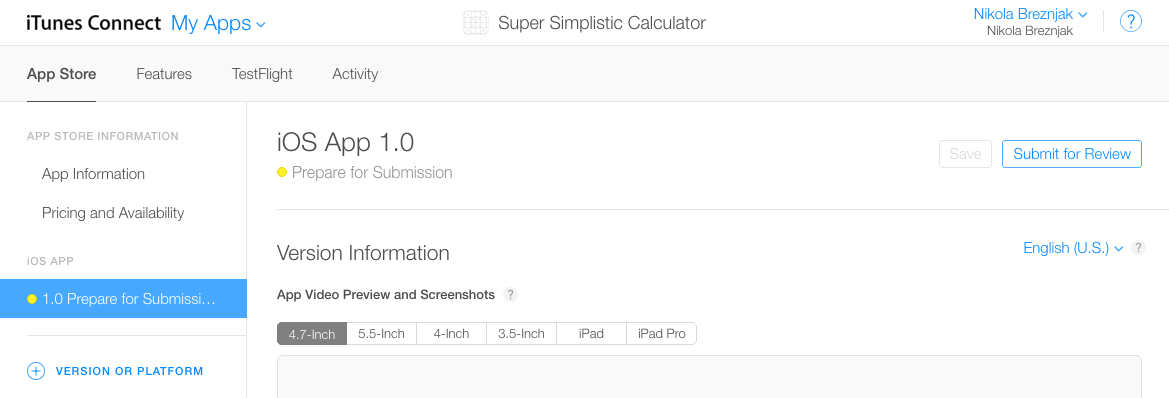
## Finishing the app list process

Now you should head back to the [iTunes Connect portal](https://itunesconnect.apple.com/) and login. Next, click on the Pricing and Availability on the left hand side under APP STORE INFORMATION.

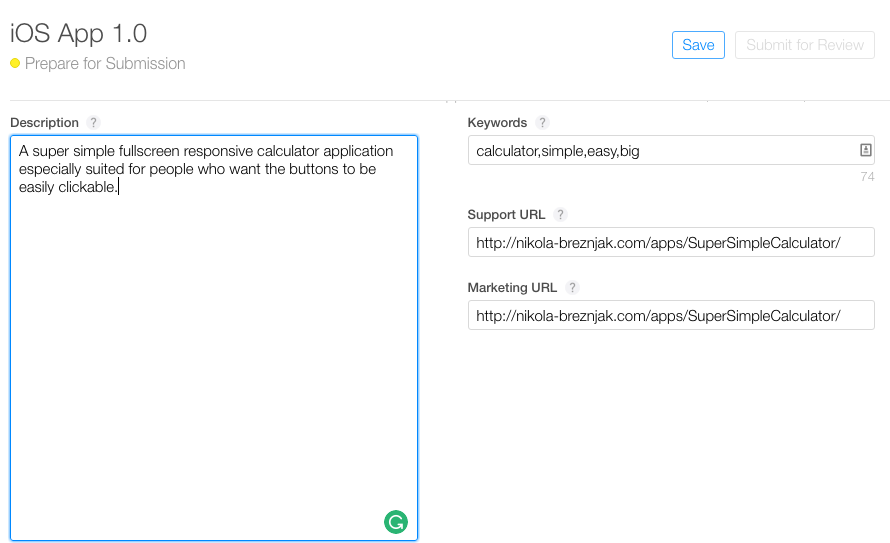


You don’t have to worry about forgetting to insert any crucial and required information about your application, since you’ll be notified about what’s missing and what needs to be added/changed if you try to submit the app for review before all details are filled in.

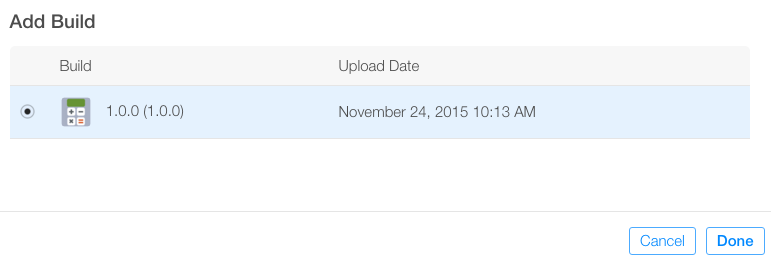
Next, click on the 1.0 Prepare for Submission button on the left hand side, as shown on the image below. When we uploaded our archive, iTunes Connect automatically determined which device sizes are supported. As you can also see on the image below, you’ll need to upload at least one screenshot image for each of the various app sizes that were detected by iTunes Connect.



Next you’ll have to insert Description, Keywords, Support URL and Marketing URL (optionally), as shown on the image below:

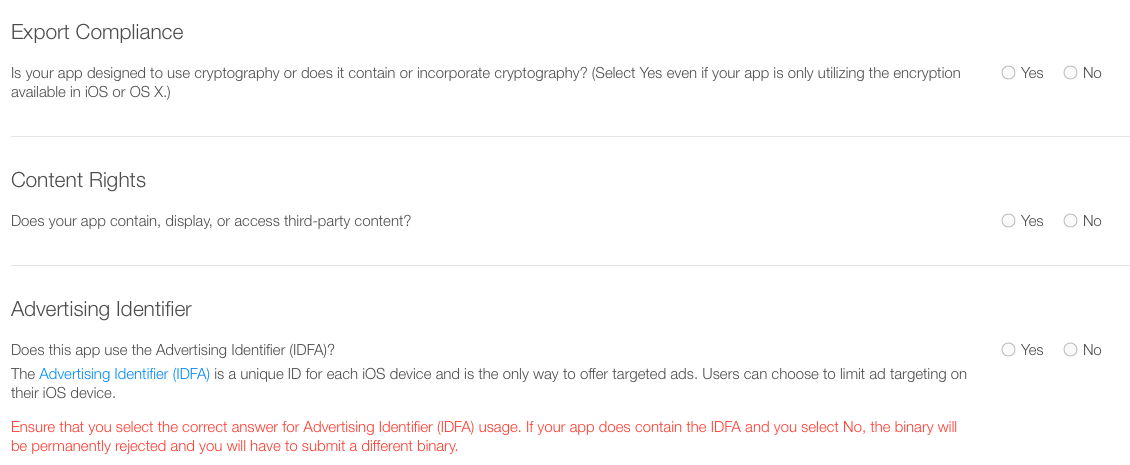


In the Build section you have to click on the + button and select the build that was uploaded through Xcode in the previous steps, as shown on the image below:

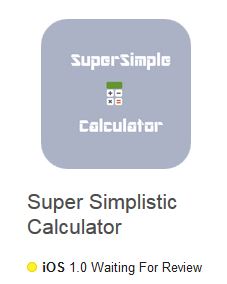


Next you’ll have to upload the icon, edit the rating, and set some additional info like copyright and your information. Note that the size of the icon that you’ll have to upload here will have to be 1024 by 1024 pixels. Thankfully, you can use the splash.png from the second tutorial. If you’re the sole developer then the data in the App Review Information should be your own. Finally, as the last option, you can leave the default checked option that once your app is approved that it is automatically released to the App Store.

Now that we’re finished with adding all of the details to the app listing, we can press Save and then Submit for Review. Finally, you’ll be presented with the last form that you’ll have to fill out:



After you submit your app for review you’ll see the status of it in the My Apps as Waiting for review, as shown on the image below. Also, shortly after you submit your app for review you’ll get a confirmation email from iTunes Connect that your app is in review.



Apple prides itself with a manual review process, which basically means it can take several days for your app to be reviewed. You’ll be notified of any issues or updates to your app status.

## Updating the app

Since you’ll probably want to update your app at some point you’ll first have to update the build and version numbers in the Cordova config.xml file and then rebuild the application and open it up from the Xcode and follow the same steps all over again.

Once you submit for the review, you’ll have to wait for the review process again. It’s pivotal to note that if your changes aren’t too big you could use [Ionic Deploy](http://docs.ionic.io/docs/deploy-from-scratch) to update your application without going through the review process.

# 

# ADDITIONAL MODULE: Page Navigation Method

The Ionic team is aiming to make Ionic more generic so that it isn’t tied to any specific framework, and implementing their own navigation/routing for each framework could get quite messy and would ultimately be somewhat unnecessary. Instead, you should just rely on the native navigation for whatever framework it is that you are using Ionic with. If you are not using a framework, you can still use Ionic’s own navigation components for push/pop style navigation, but you probably shouldn’t do that in an Angular application.

Angular is a bit of a special case – since Ionic/Angular have been tightly integrated in the past, Ionic already has a specific @ionic/angularpackage that helps with integration with Angular. The NavController is included in this package and can be used for interacting with the Angular router, but it is still just using Angular routing underneath. The NavController in Ionic 4 is different to the NavController in Ionic 3 even though it has retained the same name.

You can just use Angular router methods directly, but by using Ionic’s NavController instead it will make sure to apply the appropriate screen transition animations since screen transitions in a typical Ionic application have a “direction” (e.g. a forward navigation will animate the new screen in from the right).

Keep in mind that although the way Angular routing will work with Ionic is mostly the same as normal Angular routing, and so you can just rely on Angular documentation or resources for routing examples/advice, Ionic does have its own router outlet implementation called <ion-router-outlet> (basically, you just plop the router outlet wherever you want the component for the active route to be displayed). This is mostly the same as Angular’s <router-outlet> except that it will automatically apply the screen transition animations that I just mentioned.

## **What Is Angular Routing?**

If you’ve been building Ionic applications then you would be used to navigating through your applications by using the NavController to push new pages to navigate forward in the navigation stack:

this.navCtrl.push('SignupPage');

Popping pages to move backward in the navigation stack:

this.navCtrl.pop();

or starting a completely new navigation stack by changing the root page:

this.navCtrl.setRoot('HomePage')

Angular routing is different in that it is based on the browser model of navigation and uses the URL to determine which page/component to show. In your application, you would need to supply a set of routes that might look something like this:

const routes: Routes = [

{ path: 'login', component: LoginPage },

{ path: 'home', component: HomePage },

{ path: 'detail/:id', component: DetailPage },

{ path: '', redirectTo: '/login', pathMatch: 'full'}

];

The path property defines the URL, and the component property defines which component should be displayed by the <ion-router-outlet>when that URL is hit. If I were to go to the following URL:

http://localhost:8100/home

then the HomePage would be displayed. We also have a default route defined at the bottom so that if no route is supplied, it will use the loginroute.

NOTE: This is a simple approach to routing, make sure to also read the section on lazy loading later in this article.

Navigation in the application is then based on whatever the URL currently is, and what route it matches. Simply changing the URL would change the current page, but in order to navigate in your application, you have a few options.

You can use routerLink to link to another page:

<ion-item [routerLink]="'/detail/' + item.id" routerDirection="forward">

In this case, we would be launching the DetailPage and supplying it with an id route parameter (we could then use that data inside of the component). This is somewhat advantageous because it means you don’t always need to set up an event binding and involve the NavControllerevery time you want to transition to another page. Also note that we supply a routerDirection to indicate whether this is a forward or backward navigation so that the screen transition animation can be applied correctly. If this is not supplied, Ionic will guess at what animation it should use but it is better to be explicit.

You can also navigate programmatically by injecting the standard Angular Router into your pages and calling either of these methods:

this.router.navigateByUrl('/login');

or

this.router.navigate(['/detail', { id: itemId }]);

However, the best way to navigate programmatically in an Ionic/Angular application is to use the NavController from @ionic/angular. This is actually quite similar to how you would use the NavController previously to push/pop, except now you would just use one of these methods:

this.navCtrl.navigateForward('/route');

this.navCtrl.navigateBack('/route');

this.navCtrl.navigateRoot('/route');

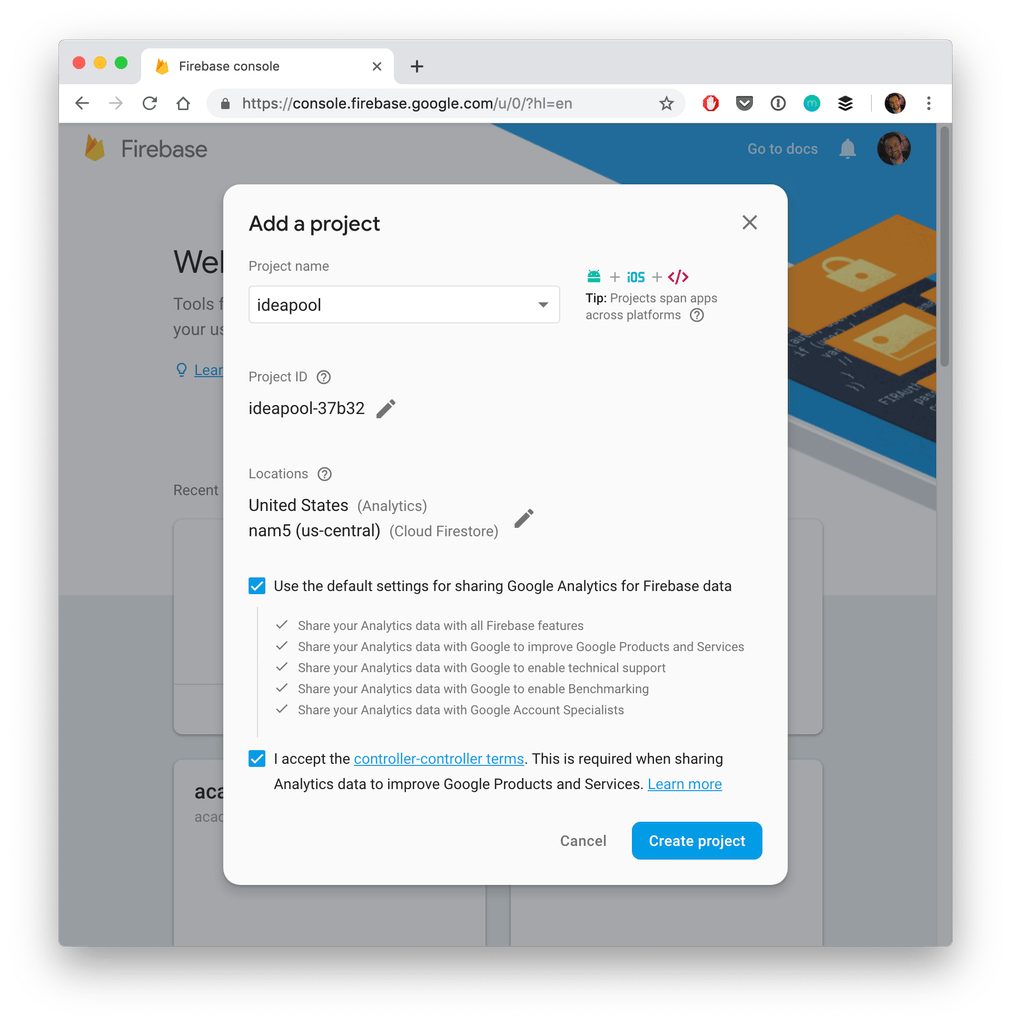
The benefit of using NavController to navigate is that it allows you to specify a “direction” for the navigation, which will help Ionic’s <ion-router-outlet> display the page transition properly.

# ADDITIONAL MODULE: Firebase

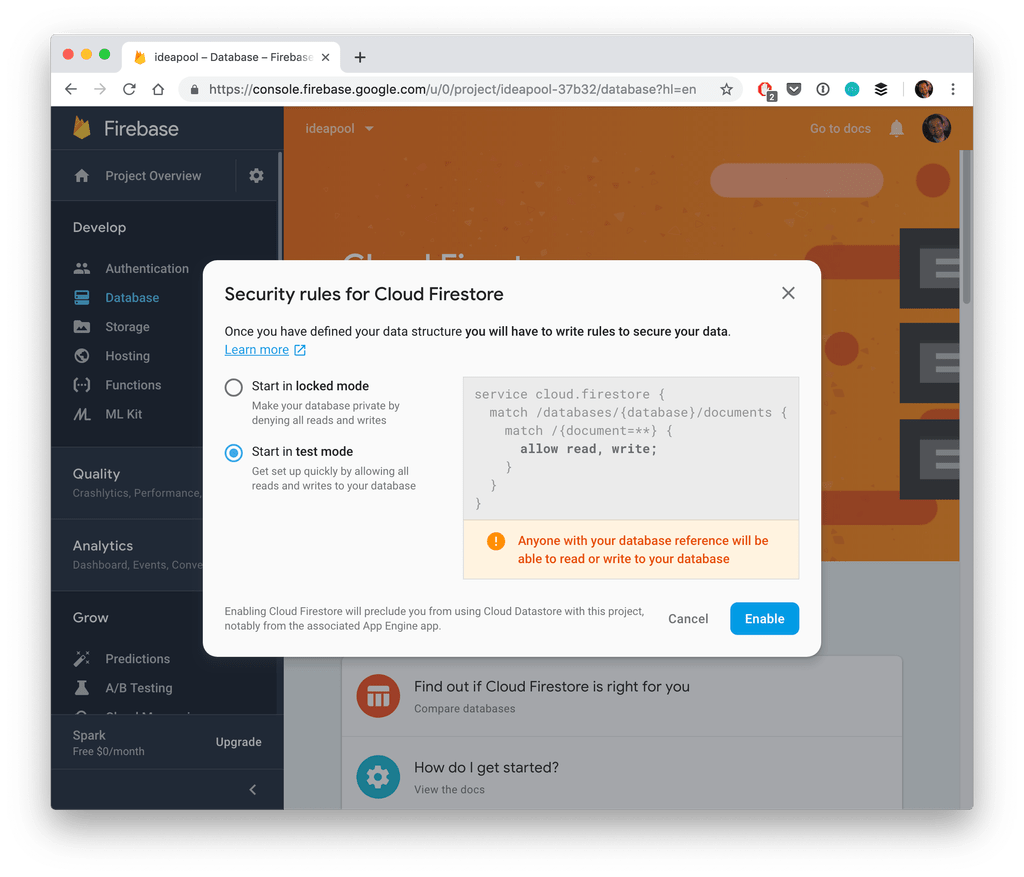
## Creating the Firebase App

Before we dive into Ionic we need to make sure we actually have a Firebase app configured. If you already got something in place you can of course skip this step.

Otherwise, make sure you are signed up (it’s free) and then hit **Add project** inside the [Firebase console](https://console.firebase.google.com/). Give your new app a name, optional select a region and then create your project once you checked the boxes like below.



After a short time you will be brought to the dashboard of your app and the only thing we need from here is the information for our Ionic app. We’ll copy this soon, but for now we also need to navigate to **Database** which will automatically open a security information (you might have to click get started first).



Here we can set the default **security rules** for our database and because this is a simple tutorial we’ll roll with the **test mode** which allows everyone access.

## 

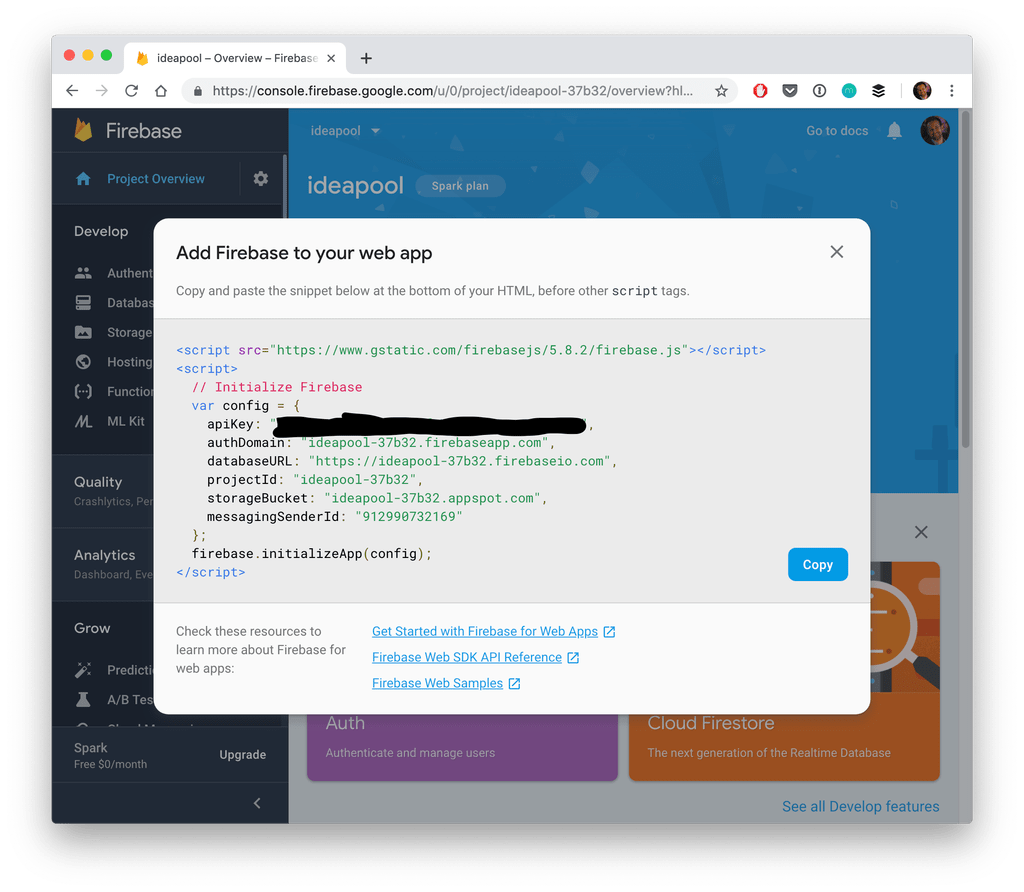
## Starting our Ionic App & Firebase Integration

Now we get into the Ionic side of things. We create a new blank Ionic 4 app and install the Firebase and AngularFire package from NPM. Additionally we need 2 more pages to navigate around and also a service so we can separate our Firebase calls from our pages, so go ahead and run:

Create a new app, install AngularFire and create some pages and a service

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | ionic start devdacticFire blank --type=angular  cd devdacticFire    npm install firebase @angular/fire    ionic g page pages/ideaList  ionic g page pages/ideaDetails  ionic g service services/idea |

Once the app is ready we need the information from Firebase to connect our Ionic app to it. Therefore, navigate to your dashboard and click on the code icon for web above **Add an app to get started** which will bring up a screen like below.



You can now simply the information from the **config** object and paste it below into your **src/environments/environment.ts** under a new firebase key like this:

Add the Firebase information to your environment

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | export const environment = {  production: false,  firebase: {  apiKey: "",  authDomain: "",  databaseURL: "",  projectId: "",  storageBucket: "",  messagingSenderId: ""  }  }; |

The cool thing about the environment is that we could have different information in this and the **.prod** file which would be used if build our app later with that command line flag!

In all of our files the import will stay the same – it’s just a different file that will be used in the end.

Just by pasting the information into the environment we are not yet done. Now it’s time to let AngularFire know about this information and we do so inside our **app/app.module.ts**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | import { NgModule } from '@angular/core';  import { BrowserModule } from '@angular/platform-browser';  import { RouteReuseStrategy } from '@angular/router';    import { IonicModule, IonicRouteStrategy } from '@ionic/angular';  import { SplashScreen } from '@ionic-native/splash-screen/ngx';  import { StatusBar } from '@ionic-native/status-bar/ngx';    import { AppComponent } from './app.component';  import { AppRoutingModule } from './app-routing.module';    import { AngularFireModule } from '@angular/fire';  import { environment } from '../environments/environment';  import { AngularFirestoreModule, FirestoreSettingsToken } from '@angular/fire/firestore';    @NgModule({  declarations: [AppComponent],  entryComponents: [],  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule,  AngularFireModule.initializeApp(environment.firebase),  AngularFirestoreModule],  providers: [  StatusBar,  SplashScreen,  { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },  { provide: FirestoreSettingsToken, useValue: {} }  ],  bootstrap: [AppComponent]  })  export class AppModule { } |

The last thing we need now is to setup our **routing information**.

Our app should have a list of ideas (*yeah, just wanted something else than a todo list..*) and also a details page for an idea, a very classic pattern you’ll most likely have in all your apps.

Therefore change the routing information inside your **app/app-routing.module.ts**where the pages and modules have been automatically added to:

The routing for our Firebase App

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import { NgModule } from '@angular/core';  import { Routes, RouterModule } from '@angular/router';    const routes: Routes = [  { path: '', loadChildren: './pages/idea-list/idea-list.module#IdeaListPageModule' },  { path: 'idea', loadChildren: './pages/idea-details/idea-details.module#IdeaDetailsPageModule' },  { path: 'idea/:id', loadChildren: './pages/idea-details/idea-details.module#IdeaDetailsPageModule' },  ];    @NgModule({  imports: [RouterModule.forRoot(routes)],  exports: [RouterModule]  })  export class AppRoutingModule { } |

Now your app will work already and you shouldn’t see any error logs – let’s continue with the Firebase interaction.

## Creating a Firebase Data Service

We can use the AngularFire service from all of our pages – but I think it makes sense to still keep the interaction with Firebase in a specific service which will simply return the data to our pages later.

Therefore we’ve created a service upfront. In this service we will store a reference to the **ideas collection** which is basically a link to one collection in our Firestore database.

Through this connection we receive all information about current documents but also add, remove and update documents.

We also got this strange **map()** block in the snapshotChanges function. This means, whenever the data changes this block will triggered and we transform the data a bit – because we need both the real data of the document but also the ID so we can apply changes to documents later, otherwise this key would not exist in the response object.

All further functionality is the simple usage of AngularFire on our collection reference. Only for getting one idea by id we add some more rxjs code. It’s mostly the same case like before – **the document itself doesn’t contain its ID**, therefore we map the data so it now also has it.

That’s just to make our life easier at a later point but nothing mandatory to have! Ok enough talking, here’s the code for your **services/idea.service.ts**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | import { Injectable } from '@angular/core';  import { AngularFirestore, AngularFirestoreCollection, AngularFirestoreDocument, DocumentReference } from '@angular/fire/firestore';  import { map, take } from 'rxjs/operators';  import { Observable } from 'rxjs';    export interface Idea {  id?: string,  name: string,  notes: string  }    @Injectable({  providedIn: 'root'  })  export class IdeaService {  private ideas: Observable<Idea[]>;  private ideaCollection: AngularFirestoreCollection<Idea>;    constructor(private afs: AngularFirestore) {  this.ideaCollection = this.afs.collection<Idea>('ideas');  this.ideas = this.ideaCollection.snapshotChanges().pipe(  map(actions => {  return actions.map(a => {  const data = a.payload.doc.data();  const id = a.payload.doc.id;  return { id, ...data };  });  })  );  }    getIdeas(): Observable<Idea[]> {  return this.ideas;  }    getIdea(id: string): Observable<Idea> {  return this.ideaCollection.doc<Idea>(id).valueChanges().pipe(  take(1),  map(idea => {  idea.id = id;  return idea  })  );  }    addIdea(idea: Idea): Promise<DocumentReference> {  return this.ideaCollection.add(idea);  }    updateIdea(idea: Idea): Promise<void> {  return this.ideaCollection.doc(idea.id).update({ name: idea.name, notes: idea.notes });  }    deleteIdea(id: string): Promise<void> {  return this.ideaCollection.doc(id).delete();  }  } |

We also used the according Typing information for all functions using our very own **idea interface** that we defined at the top – cool typing in all pages incoming!

## Displaying a Firestore Collection List

The first page of our app is the list that will display all documents of the collection. Because we created everything important upfront the actual page doesn’t have a lot of logic, but see self and add this to your **pages/idea-list/idea-list.page.ts**

Getting our Firestore Collection

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import { Component, OnInit } from '@angular/core';  import { IdeaService, Idea } from 'src/app/services/idea.service';  import { Observable } from 'rxjs';    @Component({  selector: 'app-idea-list',  templateUrl: './idea-list.page.html',  styleUrls: ['./idea-list.page.scss'],  })  export class IdeaListPage implements OnInit {    private ideas: Observable<Idea[]>;    constructor(private ideaService: IdeaService) { }    ngOnInit() {  this.ideas = this.ideaService.getIdeas();  }  } |

We now have an Observable to which we haven’t subscribed yet so right now you wouldn’t get any data from Firebase at all. We let our view take care of handling the subscription by using the **async pipe** which is most of the time the cleanest way of handling subscriptions in your view!

In order to create a new idea we also add a little FAB button and also construct the correct routerLink on our items by combining the path with the id of the **idea object of each iteration**.

Now change your **pages/idea-list/idea-list.page.html** to this:

Displaying our List with Routing action

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <ion-header>  <ion-toolbar color="primary">  <ion-title>My Ideas</ion-title>  </ion-toolbar>  </ion-header>    <ion-content>  <ion-fab vertical="bottom" horizontal="end" slot="fixed">  <ion-fab-button routerLink="/idea">  <ion-icon name="add"></ion-icon>  </ion-fab-button>  </ion-fab>    <ion-list>  <ion-item button [routerLink]="['/idea', idea.id]" \*ngFor="let idea of (ideas | async)">  {{ idea.name }}  </ion-item>  </ion-list>  </ion-content> |

As you can see the route to our details page is basically the same – it works because we specified that both of these routes resolve to the same page and module in the beginning!

## Working with a Firestore Document

This means our detail page either receives an id or not – that’s the way to tell if we are about to create a new document or are simply updating an existing one.

To access this information of the URL we can use the activatedRoute of the Angular router. So if we got an idea we need to load the details for the document which we through our service!

Also, all other functions are simply based on the service function we’ve created upfront. The only thing we need to take care of is how we react (*pun intended*) to those functions.

Sometimes we might want to display a little toast as information for the user, and sometimes we also want to directly go back once the operation has finished which we can do through the router as well.

Although the code is long there’s not really a lot of magic to it so go ahead and change your **pages/idea-details/idea-details.page.ts**:

Getting the information for one Idea

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | import { Component, OnInit } from '@angular/core';  import { ActivatedRoute, Router } from '@angular/router';  import { IdeaService, Idea } from 'src/app/services/idea.service';  import { ToastController } from '@ionic/angular';    @Component({  selector: 'app-idea-details',  templateUrl: './idea-details.page.html',  styleUrls: ['./idea-details.page.scss'],  })  export class IdeaDetailsPage implements OnInit {    idea: Idea = {  name: '',  notes: ''  };    constructor(private activatedRoute: ActivatedRoute, private ideaService: IdeaService,  private toastCtrl: ToastController, private router: Router) { }    ngOnInit() { }    ionViewWillEnter() {  let id = this.activatedRoute.snapshot.paramMap.get('id');  if (id) {  this.ideaService.getIdea(id).subscribe(idea => {  this.idea = idea;  });  }  }    addIdea() {  this.ideaService.addIdea(this.idea).then(() => {  this.router.navigateByUrl('/');  this.showToast('Idea added');  }, err => {  this.showToast('There was a problem adding your idea :(');  });  }    deleteIdea() {  this.ideaService.deleteIdea(this.idea.id).then(() => {  this.router.navigateByUrl('/');  this.showToast('Idea deleted');  }, err => {  this.showToast('There was a problem deleting your idea :(');  });  }    updateIdea() {  this.ideaService.updateIdea(this.idea).then(() => {  this.showToast('Idea updated');  }, err => {  this.showToast('There was a problem updating your idea :(');  });  }    showToast(msg) {  this.toastCtrl.create({  message: msg,  duration: 2000  }).then(toast => toast.present());  }  } |

**One Note:** In a first attempt I had the logic for loading the data for one idea inside the ngOnInit which caused the app to freeze after some fast navigation back and forth, that’s why I moved the block to one of the Ionic lifecycle events. This issue might need deeper investigation.

For now though we wrap up our app by completing the details view of our idea page that needs input fields for the **name** and **notes** of an idea.

The only cool thing here is the footer which we use to display the buttons we need – either with operations to update or delete the idea or to save it. We could add even more logic as it might flash up right now as we set the initial value of idea as if we would always edit the idea.

Maybe getting the id inside onInit and calling the subscribe in viewEnter might work to prevent this!

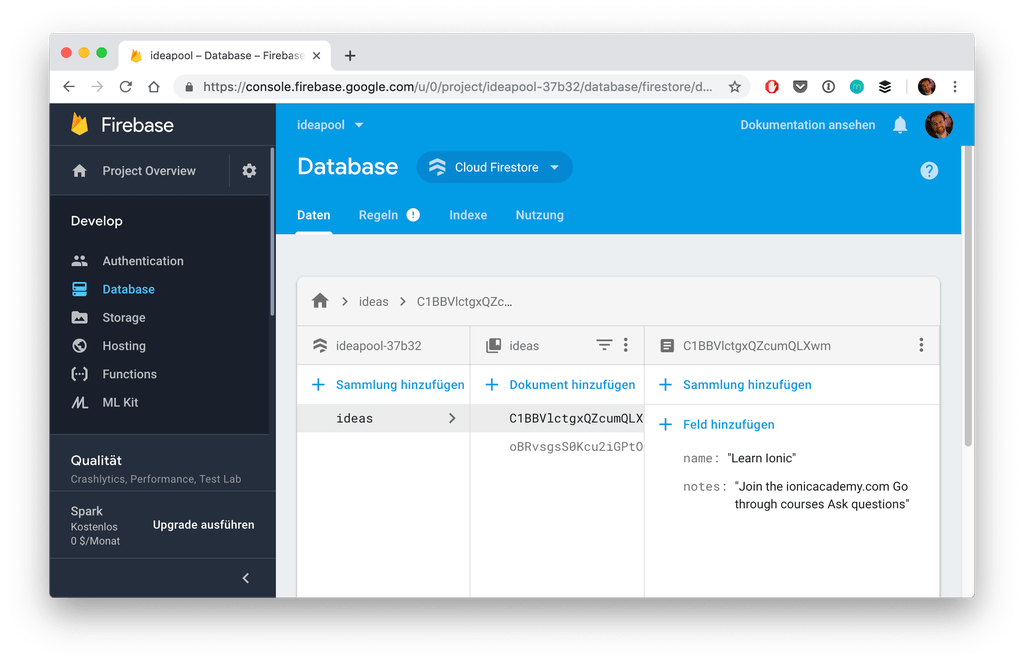
Anyhow, for now finish your view by changing the **pages/idea-details/idea-details.page.html** to:

The Idea details with footer buttons

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | <ion-header>  <ion-toolbar color="primary">  <ion-buttons slot="start">  <ion-back-button defaultHref="/"></ion-back-button>  </ion-buttons>  <ion-title>Idea Details</ion-title>  </ion-toolbar>  </ion-header>    <ion-content padding>    <ion-item>  <ion-label position="stacked">Name</ion-label>  <ion-input [(ngModel)]="idea.name"></ion-input>  </ion-item>    <ion-item>  <ion-label position="stacked">Notes</ion-label>  <ion-textarea [(ngModel)]="idea.notes" rows="8"></ion-textarea>  </ion-item>  </ion-content>    <ion-footer \*ngIf="!idea.id">  <ion-toolbar color="success">  <ion-button expand="full" fill="clear" color="light" (click)="addIdea()">  <ion-icon name="checkmark" slot="start"></ion-icon>  Add Idea  </ion-button>  </ion-toolbar>  </ion-footer>    <ion-footer \*ngIf="idea.id">  <ion-row no-padding text-center>  <ion-col size="6">  <ion-button expand="block" fill="outline" color="danger" (click)="deleteIdea()">  <ion-icon name="trash" slot="start"></ion-icon>  Delete  </ion-button>  </ion-col>  <ion-col size="6">  <ion-button expand="block" fill="solid" color="success" (click)="updateIdea()">  <ion-icon name="save" slot="start"></ion-icon>  Update  </ion-button>  </ion-col>  </ion-row>  </ion-footer> |

You can now run your app (browser or device; doesn’t matter) and play around with your connection to Firebase.

The coolest thing is still to observe how your database is updated in realtime so open it in another tab like the one below and see how your cool elements roll in or get removed!



## Conclusion

Firebase is one of the best choices as a backend for Ionic developers because what you’ve seen above – the integration works in minutes and you got a fully functional realtime database.

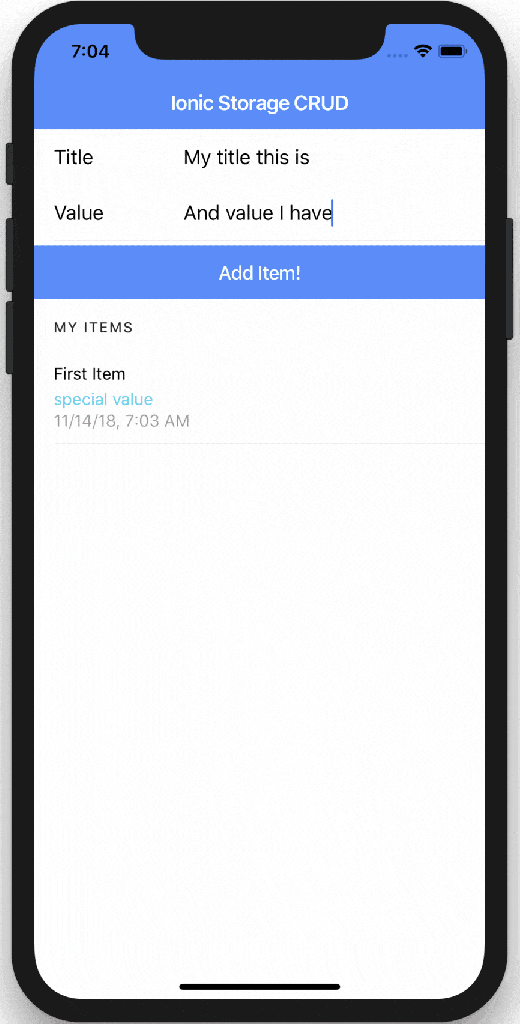
Take some more time and you have an authentication system, file storage and even more!

You can also find a video version of this tutorial below.

# ADDITIONAL MODULE: Local Storage CRUD Services

The Ionic Storage package is a great wrapper around the local storage or SQLite database inside your Ionic app, but when it comes to basic functionality there are not a lot of functions available.

Therefore we will create a simple app to demonstrate the basic CRUD (*Create, Read, Update, Delete*) functionalities for Ionic Storage on our key/value entries.



This is of course just one example how to access and work with your data as you might store it differently and of course different information!

## Starting our CRUD App

To get started we create a blank new Ionic app and generate a service where we will implement the CRUD functionalities. Also, since Ionic 4 you also need to add the storage plugin by yourself as it’s not included with the blank starter.

Finally, we add the SQLite Cordova plugin so Ionic Storage will use this database once deployed on your device. Otherwise it would use the localstorage of the browser but inside a native app this engine would be recommended.

For all of this simply run:

Start our Ionic Storage CRUD app

|  |  |
| --- | --- |
| 1  2  3  4  5 | ionic start storageCrud blank --type=angular  cd storageCrud  ionic g service services/storage  npm i @ionic/storage  ionic cordova plugin add cordova-sqlite-storage |

We also need to add the Ionic Storage package to the root of our app so open your **app/app.module.ts** and add:

Include the Storage Module

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | import { NgModule } from '@angular/core';  import { BrowserModule } from '@angular/platform-browser';  import { RouteReuseStrategy } from '@angular/router';    import { IonicModule, IonicRouteStrategy } from '@ionic/angular';  import { SplashScreen } from '@ionic-native/splash-screen/ngx';  import { StatusBar } from '@ionic-native/status-bar/ngx';    import { AppComponent } from './app.component';  import { AppRoutingModule } from './app-routing.module';    import { IonicStorageModule } from '@ionic/storage';    @NgModule({  declarations: [AppComponent],  entryComponents: [],  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule,  IonicStorageModule.forRoot()  ],  providers: [  StatusBar,  SplashScreen,  { provide: RouteReuseStrategy, useClass: IonicRouteStrategy }  ],  bootstrap: [AppComponent]  })  export class AppModule {} |

That’s everything you need to add the Ionic Storage wrapper to your app!

## CRUD Functions for Ionic Storage

Because we want to have all logic in one place we generated a new service previously from which we will perform all calls to the storage.

To give this example a bit more structure I also added a simple **interface** that describes an item with some basic fields. The important one is the ID here, which you normally also get from most API responses that will later help to identify certain objects that are stored!

Besides that our service only consists of the **CRUD functions**. In all of the functions we need to load the current values from the storage before working with them in order to have the most up to date values.

We could also keep the information inside a local variable and only store it back occasionally, but the approach here is a bit easier for the beginning and works totally fine.

In detail our functions are:

* **Create** (addItem): Check the if we already have an array saved and append the new item or create an array with the new item and save it to the storage
* **Read** (getItems): Simple return the information stored under our key
* **Update** (updateItem): Grab all stored items, then iterate all of them and replace the one we want to update with the new one
* **Delete** (deleteItem): Grab all stored items, then iterate all of them and only save back the ones where the ID is different from the item that we want to remove

All of our values are stored under the ITEMS\_KEY which we only define once and simply use the constant from then on!

Now go ahead and change your **app/services/storage.service.ts** to:

The service with all of our CRUD operations

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | import { Injectable } from '@angular/core';  import { Storage } from '@ionic/storage';    export interface Item {  id: number,  title: string,  value: string,  modified: number  }    const ITEMS\_KEY = 'my-items';    @Injectable({  providedIn: 'root'  })  export class StorageService {    constructor(private storage: Storage) { }    // CREATE  addItem(item: Item): Promise<any> {  return this.storage.get(ITEMS\_KEY).then((items: Item[]) => {  if (items) {  items.push(item);  return this.storage.set(ITEMS\_KEY, items);  } else {  return this.storage.set(ITEMS\_KEY, [item]);  }  });  }    // READ  getItems(): Promise<Item[]> {  return this.storage.get(ITEMS\_KEY);  }    // UPDATE  updateItem(item: Item): Promise<any> {  return this.storage.get(ITEMS\_KEY).then((items: Item[]) => {  if (!items || items.length === 0) {  return null;  }    let newItems: Item[] = [];    for (let i of items) {  if (i.id === item.id) {  newItems.push(item);  } else {  newItems.push(i);  }  }    return this.storage.set(ITEMS\_KEY, newItems);  });  }    // DELETE  deleteItem(id: number): Promise<Item> {  return this.storage.get(ITEMS\_KEY).then((items: Item[]) => {  if (!items || items.length === 0) {  return null;  }    let toKeep: Item[] = [];    for (let i of items) {  if (i.id !== id) {  toKeep.push(i);  }  }  return this.storage.set(ITEMS\_KEY, toKeep);  });  }  } |

There are many ways to perform different actions, we could use RxJS for some more filtering but writing the things out might help especially beginners to understand how to work with Ionic Storage.

## Working with our Storage Service

We have set up all functionalities and now it’s time to see them in action!

To use our service we just need a simply class to call of these functions. Here we will keep the list of items but reload it once we make any changes.

I’ve added some comments inline where we could also call a push() or splice() on our list so we don’t have to reload it from storage. Anyway, if you don’t have super mach data stored the time it takes should be fine.

All we have to do inside our functions here is to make the appropriate call to our service and then perhaps display a little toast afterwards to show the user what happened.

Because we are returning promises we can simply use the then() block where we are sure the **operation has finished and was written back to the storage**! We will also get a nice code completion as we have added the return values inside our service which comes in super handy especially when you work together with other developers on different parts of your app.

There isn’t much other logic included in here so simply go ahead and change your **app/home/home.page.ts** to:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  4 | import { Component, ViewChild } from '@angular/core';  import { StorageService, Item } from '../services/storage.service';  import { Platform, ToastController, IonList } from '@ionic/angular';    @Component({  selector: 'app-home',  templateUrl: 'home.page.html',  styleUrls: ['home.page.scss'],  })  export class HomePage {    items: Item[] = [];    newItem: Item = <Item>{};    @ViewChild('mylist')mylist: IonList;    constructor(private storageService: StorageService, private plt: Platform, private toastController: ToastController) {  this.plt.ready().then(() => {  this.loadItems();  });  }    // CREATE  addItem() {  this.newItem.modified = Date.now();  this.newItem.id = Date.now();    this.storageService.addItem(this.newItem).then(item => {  this.newItem = <Item>{};  this.showToast('Item added!')  this.loadItems(); // Or add it to the array directly  });  }    // READ  loadItems() {  this.storageService.getItems().then(items => {  this.items = items;  });  }    // UPDATE  updateItem(item: Item) {  item.title = `UPDATED: ${item.title}`;  item.modified = Date.now();    this.storageService.updateItem(item).then(item => {  this.showToast('Item updated!');  this.mylist.closeSlidingItems(); // Fix or sliding is stuck afterwards  this.loadItems(); // Or update it inside the array directly  });  }    // DELETE  deleteItem(item: Item) {  this.storageService.deleteItem(item.id).then(item => {  this.showToast('Item removed!');  this.mylist.closeSlidingItems(); // Fix or sliding is stuck afterwards  this.loadItems(); // Or splice it from the array directly  });  }    // Helper  async showToast(msg) {  const toast = await this.toastController.create({  message: msg,  duration: 2000  });  toast.present();  }    } |

There is one part where we need to manually close the sliding items in order to keep them working afterwards. This might also change in the future as this Quick Win was created using an Ionic 4 beta!

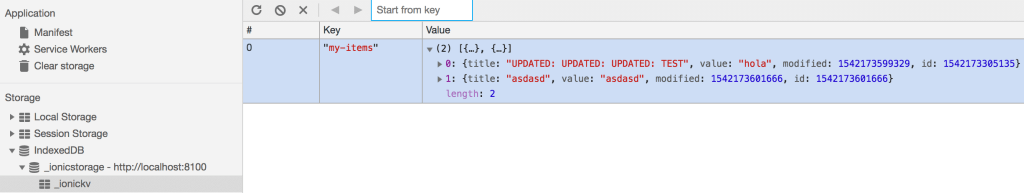
Now finally we have to build a tiny view that incorporates all the functions we have added. We don’t need much, just some inputs to create a new item and then a list with sliding buttons that will reveal the update and delete functionality.

No magic in here so let’s finish our **app/home/home.page.html** like this:

The view for our CRUD App

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | <ion-header>  <ion-toolbar color="primary">  <ion-title>  Ionic Storage CRUD  </ion-title>  </ion-toolbar>  </ion-header>    <ion-content>  <ion-item>  <ion-label position="fixed">Title</ion-label>  <ion-input [(ngModel)]="newItem.title"></ion-input>  </ion-item>    <ion-item>  <ion-label position="fixed">Value</ion-label>  <ion-input [(ngModel)]="newItem.value"></ion-input>  </ion-item>    <ion-button expand="full" (click)="addItem()">Add Item!</ion-button>    <ion-list #mylist>    <ion-list-header>  <ion-label>My Items</ion-label>  </ion-list-header>    <ion-item-sliding \*ngFor="let item of items">    <ion-item>  <ion-label text-wrap>  <h3>{{ item.title }}</h3>  <ion-text color="secondary">  <p>{{ item.value }}</p>  </ion-text>  <p>{{ item.modified | date:'short'}}</p>  </ion-label>  </ion-item>    <ion-item-options side="end">  <ion-item-option color="secondary" (click)="updateItem(item)">Update</ion-item-option>  <ion-item-option color="danger" (click)="deleteItem(item)">Delete</ion-item-option>  </ion-item-options>  </ion-item-sliding>    </ion-list>    </ion-content> |

You can now run your Ionic Storage CRUD app in the browser and if you inspect the app with the **developers tools** and open the **application** tab you can even drill down to find the information that is written to your storage!



From this place you can also change or delete the information, but you need to reload your app in order to reflect those changes (this is not Firebase realtime!).

## Conclusion

It’s certainly not hard to work with Ionic Storage if you respect the Promise flow and understand how your data is saved.

Just make sure the amount of data is not getting out of hand, and also images or files in generally shouldn’t be stored inside this storage. For this you can directly use the apps folder and simply keep a reference for later.

You can also find a vide version of this Quick Win below.

# ADDITIONAL MODULE: Build Using Capacitor

Capacitor is easily installed directly into any Ionic project (1.0-4.x+).

**[New Ionic Project](https://capacitor.ionicframework.com/docs/getting-started/with-ionic/#new-ionic-project)**

ionic start myApp tabs --capacitor

cd myApp

### **[Existing Ionic Project](https://capacitor.ionicframework.com/docs/getting-started/with-ionic/#existing-ionic-project)**

cd myApp

ionic integrations enable capacitor

### **[Initialize Capacitor with your app information](https://capacitor.ionicframework.com/docs/getting-started/with-ionic/#initialize-capacitor-with-your-app-information)**

*Note: npx is a new utility available in npm 5 or above that executes local binaries/scripts to avoid global installs.*

npx cap init [appName] [appId]

where appName is the name of your app, and appId is the domain identifier of your app (ex: com.example.app).

*Note: Use the native IDEs to change these properties after initial configuration.*

### **[Build your Ionic App](https://capacitor.ionicframework.com/docs/getting-started/with-ionic/#build-your-ionic-app)**

You must build your Ionic project at least once before adding any native platforms.

ionic build

This creates the www folder that Capacitor has been [automatically configured](https://capacitor.ionicframework.com/docs/basics/configuring-your-app) to use as the webDir in capacitor.config.json.

### **[Add Platforms](https://capacitor.ionicframework.com/docs/getting-started/with-ionic/#add-platforms)**

npx cap add ios

npx cap add android

Both android and ios folders at the root of the project are created. These are entirely separate native project artifacts that should be considered part of your Ionic app (i.e., check them into source control, edit them in their own IDEs, etc.).

### **[Open IDE to build, run, and deploy](https://capacitor.ionicframework.com/docs/getting-started/with-ionic/#open-ide-to-build-run-and-deploy)**

npx cap open ios

npx cap open android

The native iOS and Android projects are opened in their standard IDEs (Xcode and Android Studio, respectively). Use the IDEs to run and deploy your app.

## **[Syncing your app with Capacitor](https://capacitor.ionicframework.com/docs/getting-started/with-ionic/#syncing-your-app-with-capacitor)**

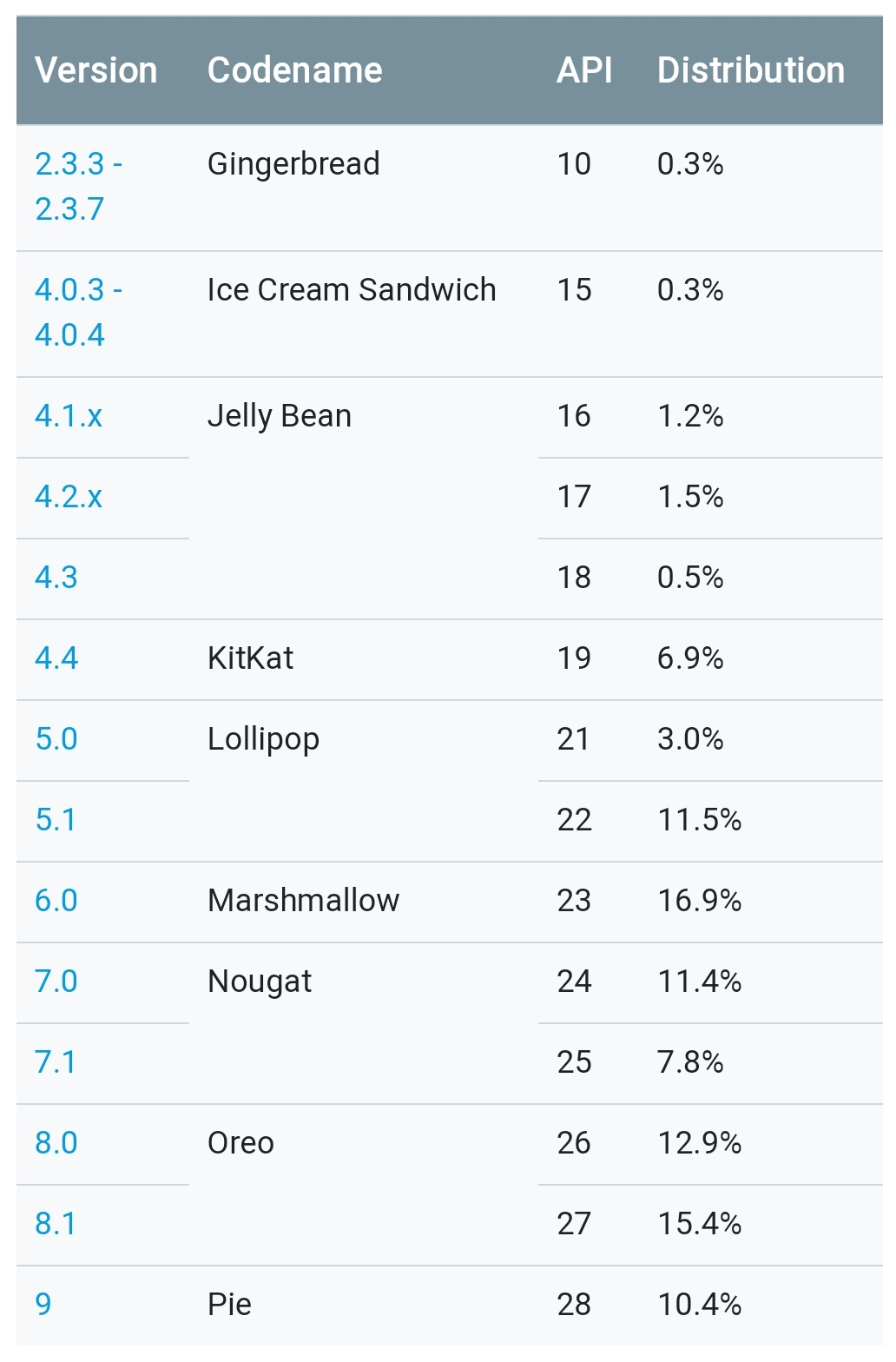
Every time you perform a build (e.g. ionic build) that changes your web directory (default: www), you'll need to copy those changes down to your native projects:

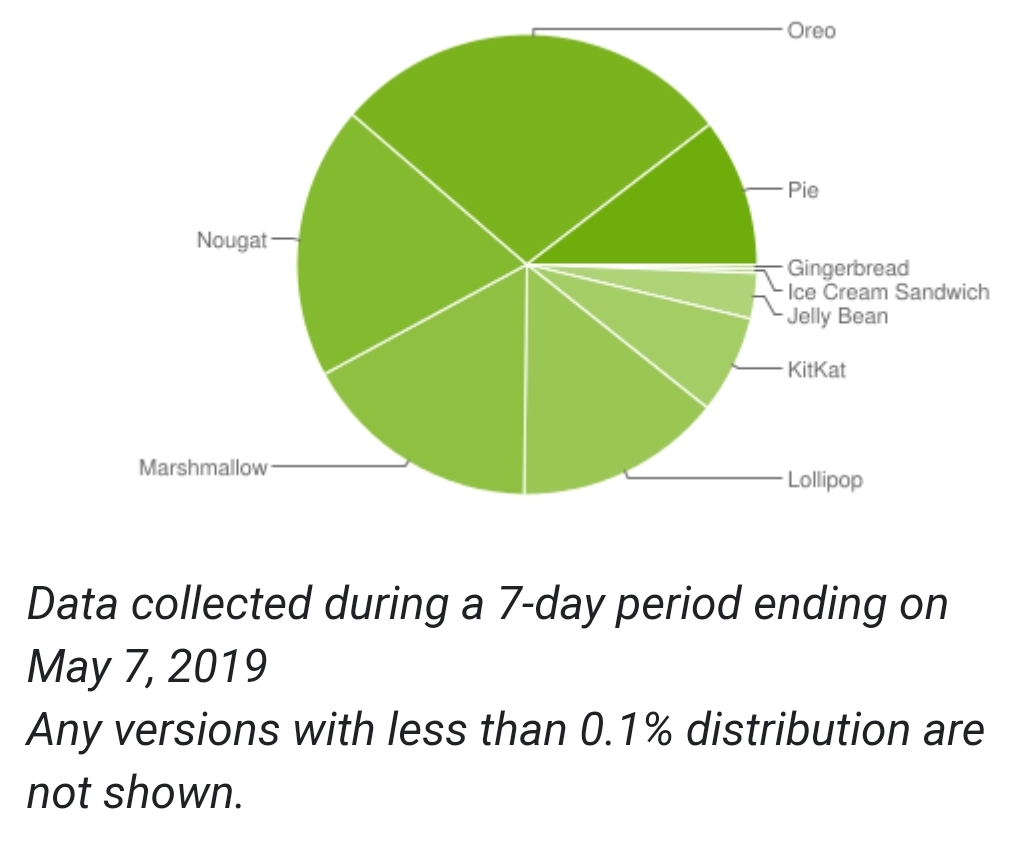
npx cap copy

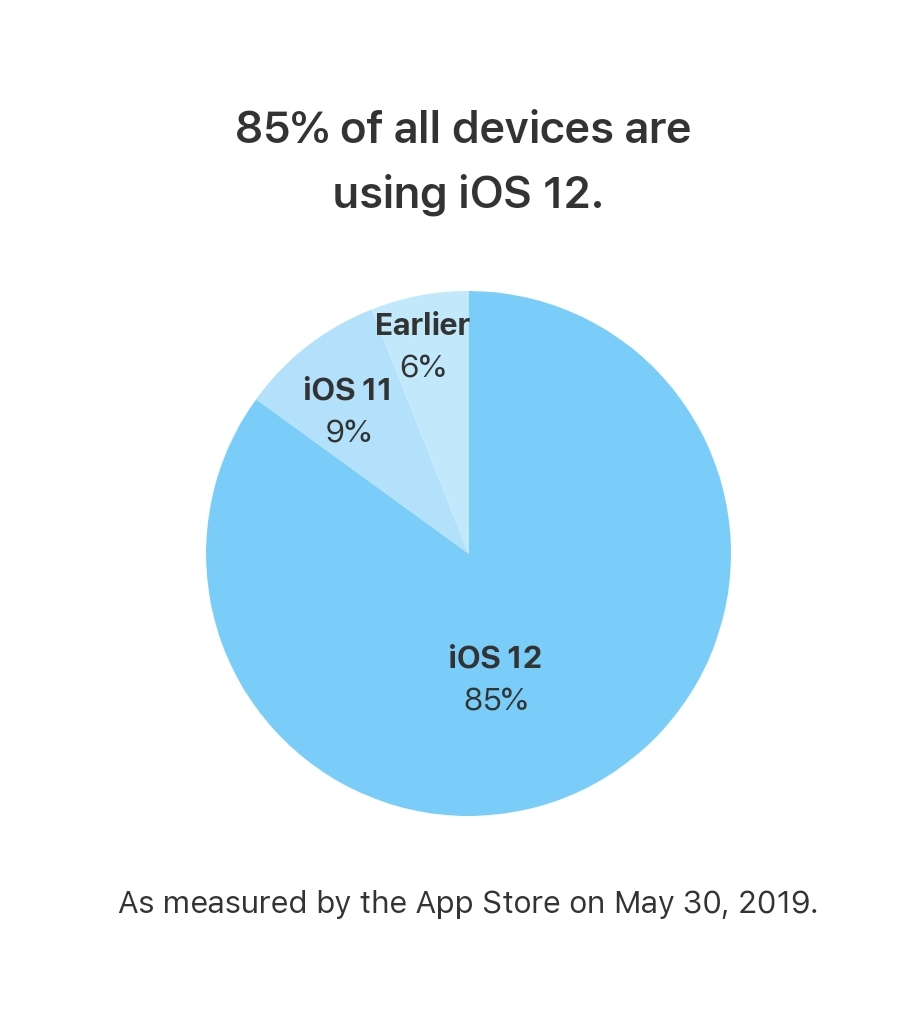
# ADDITIONAL MODULE: MINIMUM SPEC

ANDROID KITKAT 4.4

iOS 10







## 1. Make use of Ionic/Angular

## 2. Ahead-of-Time compiling --prod for ios --release for android

Ahead of time compiling vs runtime compiling

Faster view rendering and decrease launch times

## 3. Minimise system intensive processes

Avoid 'heavy' js calls, like timeouts

Avoid over animation css tricks

## 4. Use DOM rendering sparingly

Reduce the load on your device's system resources by re-using DOM items wherever possible.

## 5. Collect your garbage

Clear out unused variables, objects and arrays so that they are not kept in memory.

## 6. Data persistence

Avoid local storage in favour of a database solution wherever possible.

## 7. Manage network requests

Intelligently handle network requests and/or how remote content is deployed within your application.

## 8. Efficiently handle media content

Use svg preferable for graphic

Compress bitmap file where possible

If it is possible to download, make sure its efficient

# ADDITIONAL MODULE: GOOGLE MAPS INSTALLATION

1. npm install @ionic-native/core@beta @ionic-native/google-maps@beta
2. ionic cordova plugin add cordova-plugin-googlemaps

IONIC 4 DEMO APP

<https://github.com/mapsplugin/ionic-googlemaps-quickdemo-v4>

# ADDITIONAL MODULE: IONIC PWA

1. Start a new app
2. Run:
   1. npm install -g @angular/cli
3. Run the following command to start a Service Worker:
   1. ng add @angular/pwa --project app
4. Run ionic build --prod
5. Run npm i serve
6. CD to www and run the following command
   1. serve -p 8080
7. Run localhost:8080 in browser

# 

# ADDITIONAL MODULE: USING NETLIFY AS DEV OPS

www.netlify.com

##### AT NETLIFY

ionic build --prod

www

##### AT IONIC CLI

npm install --save-dev ionic

##### AT CODE ROUTING

app.routing.module.ts

**RouterModule.forRoot(routes,{useHash:true})**

# ADDITIONAL MODULE: ChartJS

1. Start a new project
2. Install ChartJS
   1. npm install chart.js --save
3. Build chart based with the following command:
   1. this.doughnutChart = new Chart(this.doughnutCanvas.nativeElement, {.....})
4. Please refer to <https://www.chartjs.org/docs/latest/charts/> for more chart commands

# ADDITIONAL MODULE: OneSignal Push Notification

1. Start a new Ionic project
2. Start a new Firebase Project
3. Start a new OneSignal Project
4. Run the following command
   1. ionic cordova plugin add onesignal-cordova-plugin --save
5. At app.component.ts after **this.splashscreen.hide(),** add the following code

this.oneSignal.startInit(

'108ef5be-6481-4fef-8f41-04d9a00c2b25', //FROM ONESIGNAL

'150684570494' //FROM FIREBASE

);

this.oneSignal.inFocusDisplaying(this.oneSignal.OSInFocusDisplayOption.InAppAlert);

//TO GET PUSH ID

this.oneSignal.getIds().then(resp=>{

//1. Get user information

//2. Register device

//Register user based on resp ID

})

this.oneSignal.handleNotificationReceived().subscribe(() => {

// do something when notification is received

// alert("NEW NOTIFICATION");

});

this.oneSignal.handleNotificationOpened().subscribe(() => {

// do something when a notification is opened

});

this.oneSignal.endInit();

6. For more info, please refer to sample app provided with this course