LEARNING ionic2

#ionic2

Table of Contents

About

Chapter 1: Getting started with ionic2

Remarks

Ionic 2 is a cross-platform mobile development technology. This framework is built for building hybrid mobile applications and it can be also used for desktop application as well. It is a write once, run everywhere technology. It uses web technologies such as JavaScript/Typescript, Angular 2, HTML and CSS(SCSS/LESS). Ionic2 apps works good on `>=android 4.4`, but you want run on `android 4.1` to `android 4.3` you have to use cross walk.

Examples

Installation or Setup

Since Ionic 2 is getting better and better every day, please always check the **official documentation** to keep track of the latest changes and improvements.

Prerequisites: You will need NodeJS in order to build Ionic 2 projects. You can download and install node here and learn more about npm and the packages Ionic 2 uses here.

---

1. Installing Ionic 2

Like Ionic 1, you can use the Ionic CLI or GUI to quickly build and test apps right in the browser. It even has all the functionality to work with your Ionic 1 apps, so you won't need to change a thing!

To use Ionic 2 simply install ionic from npm:

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

If you get an EACCES error, follow the instructions here to give node the permissions it needs.

2. Creating Your First App

Once the CLI is installed, run the following command to start your first app:

```
$ ionic start MyIonic2Project
```

The tabs template is used by default, but you can choose another template by passing in a flag. For example:

```
$ ionic start MyIonic2Project tutorial
$ cd MyIonic2Project
$ npm install
```

This will use the tutorial template.

To run your app, change into your projects directory and run `ionic serve -lc`:

```
$ ionic serve -lc
```

The -l activates the live reload of the page, the -c displays the console logs. If you're having issues building your app, make sure your package.json matches the one in the ionic2-app-base

You can play with your new app right there in the browser!

3. Building to a Device

You can also build your new app on a physical device or a device emulator. You will need Cordova to proceed.

To install Cordova, run:

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

Check out the iOS simulator docs for building iOS applications (NOTE: you cannot build to iOS devices or emulators on any operating system other than OSX), or the Genymotion docs to build an Android application.

Running on iOS device:

To build an iOS app, it is necessary for you to work on an OSX computer, because you will need the cocoa framework to be able to build for ios, if it's the case you will first need to add the platform to cordova by running the following command:

```
$ ionic cordova platform add ios
```

You will need Xcode to compile to an iOS device.

Finally, run your app with the following command:

```
$ ionic cordova run ios
```

Running on an Android device:

The steps for Android are almost identical. First, add the platform:

```
$ ionic cordova platform add android
```

Then install the Android SDK which allows you to compile to an Android device. Although the Android SDK comes with an emulator, it's really slow. Genymotion is much faster. Once installed, simply run the following command:

```
$ ionic cordova run android
```

And that's it! Congratulations on building your first Ionic 2 app!

Ionic has live reloading too. So if you want to develop your app and see changes taking place live on the emulator / device, you can do that by running the following commands:

For iOS:

```
$ ionic cordova emulate ios -lcs
```

Be careful, on iOS 9.2.2 the livereload doesn't work. If you want to work with livereload, edit the config.xml file by adding the following :

```
<allow-navigation href="*"/>
```

Then in the `<platform name="ios">` :

```
<config-file parent="NSAppTransportSecurity" platform="ios" target="*-Info.plist">
  <dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
  </dict>
</config-file>
```

For Android:

```
$ ionic cordova run android -lcs
```

The `l` stands for live-reload, `c` for console logs, and `s` for server logs. This will allow you to see if there are any errors / warnings during execution.

Building for Windows

If you want to build your project for windows, you need to work on a windows computer. To start, install the windows platform to your ionic2 project by running the following command :

```
$ionic cordova platform add windows
```

Then just run the following command :

```
$ionic cordova run windows
```

To run in browser

```
$ionic serve
```

for chrome browser inspect device.(type in address bar of chrome browser)

```
chrome://inspect/#devices
```

Read Getting started with ionic2 online: https://riptutorial.com/ionic2/topic/3632/getting-started-with-ionic2

Chapter 2: Add ionic app to ionic view

Introduction

ionic view is a mobile app which you have to install in your mobile, so that you can view your app without creating .apk files. By sharing your app id, others can also view your app in their mobile using ionic view.

Site: https://view.ionic.io/

Examples

Steps to add your app to ionic view

The following steps need to be done in the app.ionic.io

1. Create an account or login into your ionic account
2. Click "New App" in the Dashboard and give name for your app

   ```
   I named my app as 'MyIonicApp'
   ```

3. In the overview section of this newly created app, there will be a ID below the app name.

   ```
   MyIonicApp ID is 4c5051c1
   ```

Below steps are done in **Node.js** command prompt

1. Login into your ionic account by running

   ```
   $ ionic login
   ```

2. Root your app folder.
3. To upload your app to ionic view, first you have to link your app with the ID you created in ionic site. Run the following command to link,

   ```
   $ ionic link [your-app-id]
   ```

   For MyIoincApp, the command will be,

   ```
   $ ionic link 4c5051c1
   ```

   The above command will update the app id in the MyIonicApp's config file.

4. Once the linking is done, upload the app by executing

```
$ ionic upload
```

Note

Once the upload is successful, open ionic view in your mobile to view the app.

Others can view your app, by submitting the app id under 'Preview an app' section in ionic view.

Read Add ionic app to ionic view online: https://riptutorial.com/ionic2/topic/10542/add-ionic-app-to-ionic-view

Chapter 3: Angularfire2 with Ionic2

Introduction

Here ill show you how to integrate AngularFire2 and use this real time database in our Ionic App.

Examples

AngularFire initialization

First of all you need to initialize the angularfire modules in your app module like this:

```
  const firebaseConfig = {
  apiKey: 'XXXXXXXXXX',
  authDomain: 'XXXXXXXXXX',
  databaseURL: 'XXXXXXXXXX',
  storageBucket: 'XXXXXXXXXX',
  messagingSenderId: 'XXXXXXXXXX'
};
```

You can get this keys by signing on firebase and creating a new project.

```
imports: [
    AngularFireModule.initializeApp(firebaseConfig),
    AngularFireDatabaseModule,
    AngularFireAuthModule
  ],
```

Using AngularFire2

Once you have it on your app, just import it:

```
import { AngularFireDatabase } from 'angularfire2/database';
constructor (private _af: AngularFireDatabase) {}
```

With this Observable List you can access to a list of items under a path, for example if you have root/items/food you can get food items like this:

```
this._af.list('root/items/food');
```

And you can simple put a new item here and will appear on your firebase database, or you can update one item and you will see it update on your database. You can push and update like this:

```
this._af.list('root/items/food').push(myItemData);
this._af.list('root/items/food').update(myItem.$key, myNewItemData);
```

Or you can even remote items from your food list:

```
this._af.list('root/items/food').remove(myItem.$key);
```

Read Angularfire2 with Ionic2 online: https://riptutorial.com/ionic2/topic/10918/angularfire2-with-ionic2

Chapter 4: Constructor and OnInit

Introduction

In respect of ionic2 the `constructor`: in simple terms we use it to create instance of our plugins, services etc. for example: You have a page(view) where you want to show the list of all students, and you have a json file that contains all the students (this file is your data file) what you have to do is to create a service in this service you will create a method and hit a http.get request to get the json data, so here you need what? http simply do this way:

Examples

Student Service Method example for using Http in constructor

```
import {Http} from '@angular/http';
@Injectable()
export class StudentService{
    constructor(public http: Http){}
    getAllStudents(): Observable<Students[]>{
        return this.http.get('assets/students.json')
        .map(res => res.json().data)
        }
    }
```

notice the constructor now again if we want to use this service method we will go to our view/page and :

```
import {StudentService} from './student.service';
import { SocialSharing } from '@ionic-native/social-sharing';
export class HomePage implements OnInit {

  constructor(public _studentService: StudentService, public socialSharing: SocialSharing) {
   }
```

again notice the constructor here, we are creating an instance of StudentService in constructor and one more thing, we are using socialSharing plugin so to use that we are creating instance of that in constructor as well.

ngOnInit method to get the list of students on view load

`OnInit`: this is really amazing thing in ionic2 or we can say in AngularJs2. With the same above example we can see what is ngOnInit is. So you are ready with the service method, now in your view/page you want that student list data available as soon as your view is going to appear, this should be the first operation happend automatically on load, because as the view load the student list should be visible. So the class implements OnInit and you define ngOnInit. Example:

ngOnInit example to get the list of students on page/view

```
export class HomePage implements OnInit {
...
....
constructor(....){}

ngOnInit(){
    this._studentService.getAllStudents().subscribe(
     (students: Students[]) => this.students = students,
    )
```

Read Constructor and OnInit online: https://riptutorial.com/ionic2/topic/9907/constructor-and-oninit

Chapter 5: From code to the App store - Android

Introduction

You will find step by step instructions on how to prepare and upload production ionic app onto Google Play.

Examples

Production ready

Creating app project

When creating an Android app ready for the app store it's important when using `ionic start` that we add `--appname|-a` and `--id|-i` flags which is used for google play to identify your app from other apps.

If you're starting a new mobile app project you can use the cli example below.

```
$ ionic start --v2 -a "App Example" -i "com.example.app" -t "tabs"
```

1. App configuration file

if you want to set this info inside an existing app you can modify `config.xml`. I recommend those who used the command above to modify `config.xml` as well.

Confirm/edit `widget id`, `name`, `description`, and `author` attributes.

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<widget id="com.example.app" version="1.0.0" xmlns="http://www.w3.org/ns/widgets"
xmlns:cdv="http://cordova.apache.org/ns/1.0">
  <name>Example App</name>
  <description>Example app for stackoverflow users</description>
  <author email="admin@example.com" href="http://example.com/">Your name or team</author>
  ...
</widget>
```

2. icon and splash screen

Both icon and splash image supported file types are png, psd or ai and must have a file name that corresponds to what it is `icon` or `splash` and placed under the resources dir at the root of your project. The icon image’s minimum dimensions should be 192x192 px, and should have no rounded corners. and the splash screen is much more complicated so click here to read more.

Nonetheless, minimum dimensions should be 2208x2208 px.

if you have icon file to generate use this command `ionic resources --icon` if you have splash file to generate use this command `ionic resources --splash`

3. Building production app

Before building your production app remove any sensitive log data.

To build a release version with all default optimizations in place use the --release & --prod tag

`ionic build android --release --prod`

For a full list of available optimizations you may visit the @ionic/app-scripts repository

4. Create private key

Now, we need to sign the unsigned APK (`android-release-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.

Next, locate your unsigned APK file `android-release-unsigned.apk` inside project dir `/platforms/android/build/outputs/apk/` and use `keytools` command that will be used to sign our apk file. You can use the example below:

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

you can find `my-release-key.keystore` in your current directory.

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:

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!

5. Sign APK

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.

```
$ 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!

Publish your app on 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. It will cost $25 one time fee.

Once you have a developer account, you can go ahead and click "Publish an Android App on Google Play" and follow the on-screen instruction.

Read From code to the App store - Android online: https://riptutorial.com/ionic2/topic/9659/from-code-to-the-app-store---android

Chapter 6: Geolocation

Examples

Simple usage

In your `package.json` make sure to include the dependencies:

```
{
    ...
    "dependencies": {
        ...
        "ionic-native": "^1.3.10",
        ...
    },
    ...
}
```

To use geolocation:

```
// custom-component.ts

import {Geolocation} from 'ionic-native';
import template from './custom-component.html';

@Component({
    selector: 'custom-component',
    template: template
})
export class CustomComponent {

    constructor() {

        // get the geolocation through a promise
        Geolocation.getCurrentPosition().then((position:Geoposition)=> {
            console.log(
                position.coords.latitude,
                position.coords.longitude);
        });
    }
}
```

Watching the position

For a more real time solution you can use watchPosition function in Geolocation that notifies whenever an error or a position change occurs. Unlike the getCurrentPosition the watchPosition returns an Observable

```
import {Geolocation} from 'ionic-native';
import template from './custom-component.html';

@Component({
```

```
selector: 'custom-component',
template: template
})
export class CustomComponent {
constructor() {

    // get the geolocation through an observable
        Geolocation.watchPosition(<GeolocationOptions>{
            maximumAge: 5000, // a maximum age of cache is 5 seconds
            timeout: 10000, // time out after 10 seconds
            enableHighAccuracy: true // high accuracy
        }).subscribe((position) => {
            console.log('Time:' + position.timestamp);
            console.log(
                'Position:' + position.coords.latitude + ',' +
                position.coords.longitude);
            console.log('Direction:' position.coords.heading);
            console.log('Speed:' position.coords.speed);

        });
}
```

Read Geolocation online: https://riptutorial.com/ionic2/topic/5840/geolocation

Chapter 7: InAppBrowser

Introduction

Sometimes client just require to open a web app in mobile app, for this we can use InAppBrowser in such a way that it looks like an app instead we are opening a website/webApp in mobile, as soon as the user will tap the app icon than instead of opening the first app view we can open InAppBrowser directly.

Examples

A live example of this usage is this app:

In this app i am directly opening InAppBrowser when the user tap the app icon instead of loading first page of app. So that would look like to user that they are viewing the app of the same website/webapp.

Code example to use InAppBrowser

```
   platform.ready().then(() => {
       // Okay, so the platform is ready and our plugins are available.
       // Here you can do any higher level native things you might need.
       var url= "https://blog.knoldus.com/";
       var browserRef = window.cordova.InAppBrowser.open(url, "_self", "location=no",
 "toolbar=no");
       browserRef.addEventListener("exit", (event) => {
           return navigator["app"].exitApp();
         }
       );
```

Read InAppBrowser online: https://riptutorial.com/ionic2/topic/9801/inappbrowser

Chapter 8: Ionic2 CSS components

Examples

Grid

Ionic’s grid system is based on flexbox, a CSS feature supported by all devices that Ionic supports. The grid is composed of three units-grid, rows and columns. Columns will expand to fill their row, and will resize to fit additional columns.

Class	Width
width-10	10%
width-20	20%
width-25	25%
width-33	33.3333%
width-50	50%
width-67	66.6666%
width-75	75%
width-80	80%
width-90	90%

Example.

```
<ion-grid>
  <ion-row>
    <ion-col width-10>This column will take 10% of space</ion-col>
  </ion-row>
</ion-grid>
```

Cards

Cards are a great way to display important pieces of content, and are quickly emerging as a core design pattern for apps. They're are a great way to contain and organize information, while also setting up predictable expectations for the user. With so much content to display at once, and often so little screen real estate, cards have fast become the design pattern of choice for many companies.

Example.

```
<ion-card>
  <ion-card-header>
        Header
  </ion-card-header>
  <ion-card-content>
        The British use the term "header", but the American term "head-shot" the English
simply refuse to adopt.
  </ion-card-content>
</ion-card>
```

Read Ionic2 CSS components online: https://riptutorial.com/ionic2/topic/8011/ionic2-css-components

Chapter 9: Modals

Examples

Using Modals

Modals slide in off screen to display a temporary UI, often used for login or signup pages, message composition, and option selection.

```
import { ModalController } from 'ionic-angular';
import { ModalPage } from './modal-page';

export class MyPage {
  constructor(public modalCtrl: ModalController) {
  }

  presentModal() {
    let modal = this.modalCtrl.create(ModalPage);
    modal.present();
  }
}
```

NOTE: A Modal is a content pane that goes over the user's current page.

Passing data through a Modal

Data can be passed to a new modal through `Modal.create()` as the second argument. The data can then be accessed from the opened page by injecting `NavParams`. Note that the page, which opened as a modal, has no special "modal" logic within it, but uses `NavParams` no differently than a standard page.

First Page:

```
import { ModalController, NavParams } from 'ionic-angular';

export class HomePage {

 constructor(public modalCtrl: ModalController) {

 }

 presentProfileModal() {
   let profileModal = this.modalCtrl.create(Profile, { userId: 8675309 });
   profileModal.present();
 }

}
```

Second Page:

```
import { NavParams } from 'ionic-angular';
```

```
export class Profile {

 constructor(params: NavParams) {
   console.log('UserId', params.get('userId'));
 }

}
```

Read Modals online: https://riptutorial.com/ionic2/topic/6415/modals

Chapter 10: Modals

Examples

Simple Modal

Modal is a temporary UI that is displayed on top of your current page. This is often used for login, signup, editing existing options and selecting options.

Let us look in to a simple example with modals used. To begin with we are creating an ionic blank project. Let us create a simple modal displaying a message and exit on button click. To do that first we are creating view for our modal.

Message.html

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Modal
    </ion-title>
    <ion-buttons start>
      <button (click)="dismiss()">
        <span primary showWhen="ios">Cancel</span>
        <ion-icon name="md-close" showWhen="android,windows"></ion-icon>
      </button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
<ion-content padding>
  <h1>Modal Without Params is created successfully.</h1>
  <button full (click)="dismiss()"> Exit </button>
</ion-content>
```

Message.ts

```
import { Component } from '@angular/core';
import { ViewController } from 'ionic-angular';
@Component({
  templateUrl: 'build/pages/message/message.html',
})
export class MessagePage {
  viewCtrl;
  constructor(viewCtrl: ViewController) {
    this.viewCtrl = viewCtrl;
  }
  dismiss(){
    this.viewCtrl.dismiss();
  }
}
```

This modal displays a message. The modal can be closed or “dismissed” by using the View controllers **dismiss** method.

Home.html

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Modal Example
    </ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
  <button full (click)="openModal()">ModalWithoutParams-Message</button>
</ion-content>
```

Home.ts

```
import { Component } from '@angular/core';
import { ModalController } from 'ionic-angular';
import {MessagePage} from '../message/message';
@Component({
  templateUrl: 'build/pages/home/home.html'
})
export class HomePage {
  modalCtrl;
  data;
  constructor(modalCtrl: ModalController) {
    this.modalCtrl = modalCtrl;
    this.data = [{name: "aaa", email: "aaa.a@som.com", mobile: "1234567890", nickname: "zzz"},
      {name: "bbb", email: "bbb.a@som.com", mobile: "1234567890", nickname: "yyy"},
      {name: "ccc", email: "ccc.a@som.com", mobile: "1234567890", nickname: "xxx"}]
  }
  openModal() {
    let myModal = this.modalCtrl.create(MessagePage);
    myModal.present();
  }
}
```

Now we are creating our home page importing the **ModalController** and our data model MessagePage. ModalController's **create** method creates modal for our data model MessagePage that is saved to control variable myModal. **Present** method opens the modal on top of our current page.

Modal with Parameters on dismiss:

We now know how to create a modal. But what if we want to pass some data from modal to our home page. To do so, let us look into an example with modal as Register page passing parameters to parent page.

Register.html

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Login
    </ion-title>
    <ion-buttons start>
```

```
      <button (click)="dismiss()">
        <span primary showWhen="ios">Cancel</span>
        <ion-icon name="md-close" showWhen="android,windows"></ion-icon>
      </button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
<ion-content padding>
  <ion-list>
    <ion-item>
      <ion-label>Name</ion-label>
      <ion-input type="text" [(ngModel)]="name"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label>Email</ion-label>
      <ion-input type="text" [(ngModel)]="email"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label>Mobile</ion-label>
      <ion-input type="number" [(ngModel)]="mobile"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label>Nickname</ion-label>
      <ion-input type="text" [(ngModel)]="nickname"></ion-input>
    </ion-item>
  </ion-list>
  <button full (click)="add()">Add</button>
</ion-content>
```

Register.ts

```
import { Component } from '@angular/core';
import { ViewController } from 'ionic-angular';
@Component({
  templateUrl: 'build/pages/register/register.html',
})
export class ResisterPage {
  viewCtrl;
  name;
  email;
  mobile;
  nickname;
  constructor(viewCtrl: ViewController) {
    this.viewCtrl = viewCtrl;
    this.name = "";
    this.email = "";
    this.mobile = "";
    this.nickname = "";
  }
  dismiss(){
    this.viewCtrl.dismiss();
  }
  add(){
    let data = {"name": this.name, "email": this.email, "mobile": this.mobile, "nickname":
this.nickname};
    this.viewCtrl.dismiss(data);
  }
}
```

Register modal gets data object with values entered by user and the parameters are passed to our current page on dismiss with viewControllers dismiss method. Now the parameters are sent.

So how we are going to retrieve the parameters in home page? To do so, we are creating a button on home page and call Register modal on click. To display the user, we are displaying a list.

Home.html

```
<ion-list>
  <ion-item *ngFor="let datum of data">
    <h1>{{datum.name}}</h1>
  </ion-item>
</ion-list>
<button full secondary (click)="openModalParams()">ModalWithParams-Register</button>
```

Home.ts

```
import {ResisterPage} from '../register/register';

  openModalParams(){
    let modalWithParams = this.modalCtrl.create(ResisterPage);
    modalWithParams.present();

    modalWithParams.onDidDismiss((result) =>{
      if(result){
        this.data.unshift(result);
      }
    });
  }
```

ViewController **onDidDismiss** method gets executed whenever a modal is closed. If data is passed as parameter from modal, then we can retrieve it using onDidDismiss method. Here the data entered by user is appended to the existing data. If no data is passed as parameter, then the returned value will be null.

Modal with parameters on create:

Passing parameters to a modal is similar to how we pass values to a NavController. To do so, we are altering our list in home.html to open a modal when clicking a list item and passing the required parameters as a second argument to the **create** method.

Home.html

```
 <ion-list>
    <ion-item *ngFor="let datum of data" (click)="openModalwithNavParams(datum)">
      <h1>{{datum.name}}</h1>
    </ion-item>
  </ion-list>
```

Home.ts

```
import {EditProfilePage} from '../edit-profile/edit-profile';
```

```
  openModalwithNavParams(data){
    let modalWithNavParams = this.modalCtrl.create(EditProfilePage,{Data: data});
    modalWithNavParams.present();
  }
```

Similar to other views, we use NavParams to retrieve the data sent from the previous view.

Edit-Profile.html

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Login
    </ion-title>
    <ion-buttons start>
      <button (click)="dismiss()">
        <span primary showWhen="ios">Cancel</span>
        <ion-icon name="md-close" showWhen="android,windows"></ion-icon>
      </button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
<ion-content padding>
  <h2>Welcome {{name}}</h2>
  <ion-list>
    <ion-item>
      <ion-label>Email</ion-label>
      <ion-input type="text" value={{email}}></ion-input>
    </ion-item>
    <ion-item>
      <ion-label>Mobile</ion-label>
      <ion-input type="number" value={{mobile}}></ion-input>
    </ion-item>
    <ion-item>
      <ion-label>Nickname</ion-label>
      <ion-input type="text" value={{nickname}}></ion-input>
    </ion-item>
  </ion-list>
  <button full (click)="dismiss()">Close</button>
</ion-content>
```

Edit-Profile.ts

```
import { Component } from '@angular/core';
import { ViewController, NavParams } from 'ionic-angular';
@Component({
  templateUrl: 'build/pages/edit-profile/edit-profile.html',
})
export class EditProfilePage {
  viewCtrl;
  navParams;
  data;
  name;
  email;
  mobile;
  nickname;
  constructor(viewCtrl: ViewController, navParams: NavParams) {
    this.viewCtrl = viewCtrl;
    this.navParams = navParams;
    this.data = this.navParams.get('Data');
    this.name = this.data.name;
    this.email = this.data.email;
    this.mobile = this.data.mobile;
    this.nickname = this.data.nickname;

  }
  dismiss(){
    this.viewCtrl.dismiss();
  }
}
```

Read Modals online: https://riptutorial.com/ionic2/topic/6612/modals

Chapter 11: Push notification sent & receive

Remarks

The SenderID that is present in the initialization example is a gcm sender id that is given to you by google. It should also be present when you install the plugin

```
ionic plugin add phonegap-plugin-push --variable SENDER_ID="XXXXXXX"
```

If you wish to add additional data to your push notifications look in to this link explaining how to add more typings https://github.com/phonegap/phonegap-plugin-push/blob/master/docs/TYPESCRIPT.md

Examples

Initialization

The push notification plugin requires an init an initialization which tells the plugin to start running using the sender id provided.

```
  let push = Push.init({
      android: {
        senderID: "------------",
      },
      ios: {
        alert: "true",
        badge: true,
        sound: "false",
      },
      windows: {},
    });
```

Registration

The registration step registers the app with the device's system and returns a registration id

```
 import { Push, RegistrationEventResponse} from "ionic-native";

        //the push element is created in the initialization example
        push.on("registration", async (response: RegistrationEventResponse) => {
                    //The registration returns an id of the registration on your device
                    RegisterWithWebApi(response.registrationId);

        });
```

Receiving a push notification

To receive push notifications we are supposed to tell the plugin to listen to incoming push notifications. This step is done after initialization & registration

```
import { Push, NotificationEventResponse} from "ionic-native";

        //the push element is created in the initialization example
        push.on("notification", (response: NotificationEventResponse) => {
            let chatMessage: ChatMessage = <ChatMessage>{
              title: response.title,
              message: response.message,
              receiver: response.additionalData.replyTo,
              image: response.image
            };
            DoStuff(chatMessage));
       });
```

Read Push notification sent & receive online: https://riptutorial.com/ionic2/topic/5874/push-notification-sent---receive

Chapter 12: Setup and Debugging Ionic 2 in Visual Studio Code

Introduction

Visual Studio is a open Source IDE which provides intellisense and editing facility for code .This IDE supports many languages like(Ionic ,C, C# ,AngularJs, TypeScript ,Android and so on) . These languages are able to execute there code by adding its Extensions in VSCode. By using VSCode we able to run and debug the code of different different languages.

Examples

Installation of VSCode

Firstly you need to download and install the VSCode. This VSCode latest version is available for download in its Official website . After download the VSCode you should install and open it.

```
 Introduction of Extensions in VSCode
```

VSCode is a open editor so it provide editor for all languages but to execute a code you need to add the Extension for that particular language. For running and editing your ionic code you should add **ionic2-vscode** Extension in yourVSCode. In the left side of VSCode Editor there are 5 icons in which the lowest one icon is use for Extension. The Extensions you may get by using **shortcut key (ctrl+shift+X)**.

```
Add Extension for Ionic2 in VsCode
```

By pressing **ctrl+shift+X** you shown the part of extension where on top **three dots** are shown **...** these dots are known as more icon.On click of it a dialog is open and shows the numbers of options to choose .you may choose the option as per your need but for getting all extension you should select **Shown Recommended Extension** .iN the list of all eXtension you may install your Extension `(ionic2-vscode),npm`

Create and Add your Ionic Project in VSCode

VsCode is unable to create the ionic project because it is a code editor.So you can create your ionic project by **CLI** or **cmd**. create your project by below command

```
$ ionic start appName blank
```

Above command use to create blank template ionic application. Ionic2 provide three type of templates **blank, tabs and sidemenu**. So. you may replace blank template by any other two templates as per your need.

Now, your Ionic project has been created. So, you are able to add your project in VSCode to edit. To add your project follow below points.

1. Go to **File** menu in VScode.
2. Click the **Open Folder** inside File menu.
3. Find and open your project folder.

   You may directly open the folder by using shortcut key **ctrl+O** or **ctrl+k**

`

Run and Debug your Ionic Project

> Run and Debug in Chrome

To run the ionic project use below command in **terminal or cmd or CLI**

```
 $ ionic serve
```

For debugging the ionic project ,Firstly you should add extension **(Debugger for chrome)** and then configure launch.json file like this.

```
 {
        "version": "0.2.0",
        "configurations": [

            {
          "name": "Launch in Chrome",
          "type": "chrome",
          "request": "launch",
          "url": "http://localhost:8100",
          "sourceMaps": true,
          "webRoot": "${workspaceRoot}/src"
        }
]
}
```

>***Run and Debug in Android***

For Run ionic project in Android you should add android platform by below command in terminal or cmd or CLI:

```
$ ionic cordova platform add android
```

Build Android by this command

```
$ ionic cordova build android
```

Run command for android platform

```
$ ionic cordova run android
```

Now, Your application run on real Android device.

For debug into Android device you need to add **Cordova or Android Extension** in VSCode. and configure launch.json file like this.

```
{
    "version": "0.2.0",
    "configurations": [
    {
                "name": "Run Android on device",
                "type": "cordova",
                "request": "launch",
                "platform": "android",
                "target": "device",
                "port": 9222,
                "sourceMaps": true,
                "cwd": "${workspaceRoot}",
                "ionicLiveReload": false
            },
            {
                "name": "Run iOS on device",
                "type": "cordova",
                "request": "launch",
                "platform": "ios",
                "target": "device",
                "port": 9220,
                "sourceMaps": true,
                "cwd": "${workspaceRoot}",
                "ionicLiveReload": false
            },
            {
                "name": "Attach to running android on device",
                "type": "cordova",
                "request": "attach",
                "platform": "android",
                "target": "device",
                "port": 9222,
                "sourceMaps": true,
                "cwd": "${workspaceRoot}"
            },
            {
                "name": "Attach to running iOS on device",
                "type": "cordova",
                "request": "attach",
                "platform": "ios",
                "target": "device",
                "port": 9220,
                "sourceMaps": true,
                "cwd": "${workspaceRoot}"
            },
            {
                "name": "Run Android on emulator",
                "type": "cordova",
                "request": "launch",
                "platform": "android",
                "target": "emulator",
                "port": 9222,
                "sourceMaps": true,
                "cwd": "${workspaceRoot}",
                "ionicLiveReload": false
```

```
        },
        {
            "name": "Run iOS on simulator",
            "type": "cordova",
            "request": "launch",
            "platform": "ios",
            "target": "emulator",
            "port": 9220,
            "sourceMaps": true,
            "cwd": "${workspaceRoot}",
            "ionicLiveReload": false
        },
        {
            "name": "Attach to running android on emulator",
            "type": "cordova",
            "request": "attach",
            "platform": "android",
            "target": "emulator",
            "port": 9222,
            "sourceMaps": true,
            "cwd": "${workspaceRoot}"
        },
        {
            "name": "Attach to running iOS on simulator",
            "type": "cordova",
            "request": "attach",
            "platform": "ios",
            "target": "emulator",
            "port": 9220,
            "sourceMaps": true,
            "cwd": "${workspaceRoot}"
        },
        {
            "name": "Serve to the browser (ionic serve)",
            "type": "cordova",
            "request": "launch",
            "platform": "serve",
            "cwd": "${workspaceRoot}",
            "devServerAddress": "localhost",
            "sourceMaps": true,
            "ionicLiveReload": true
        },
        {
            "name": "Simulate Android in browser",
            "type": "cordova",
            "request": "launch",
            "platform": "android",
            "target": "chrome",
            "simulatePort": 8000,
            "livereload": true,
            "sourceMaps": true,
            "cwd": "${workspaceRoot}"
        },
        {
            "name": "Simulate iOS in browser",
            "type": "cordova",
            "request": "launch",
            "platform": "ios",
            "target": "chrome",
            "simulatePort": 8000,
            "livereload": true,
                "sourceMaps": true,
                "cwd": "${workspaceRoot}"
            }
]
}
```

After configuration you follow the following steps or short keys for debugging:

1. Go to **debug** menu.

2. Click **start debugging**.

   or

```
Short keys
```

- Debugging - F5
- StepOver - F10
- Step Into and Step Out - F11
- Stop Debugging - Shift+F5
- Restart Debugging -ctrl+shift_F5

Read Setup and Debugging Ionic 2 in Visual Studio Code online:
https://riptutorial.com/ionic2/topic/10559/setup-and-debugging--ionic-2-in-visual-studio-code

Chapter 13: Social Login with Angularfire2/Firebase

Examples

Native Facebook Login with Angularfire2/Firebase

app.ts

```
import {Component} from '@angular/core';
import {Platform, ionicBootstrap} from 'ionic-angular';
import {StatusBar} from 'ionic-native';
import {LoginPage} from './pages/login/login';
import {FIREBASE_PROVIDERS, defaultFirebase, AuthMethods, AuthProviders, firebaseAuthConfig}
from 'angularfire2';

@Component({
  template: '<ion-nav [root]="rootPage"></ion-nav>'
})

export class MyApp {

  private rootPage: any;

  constructor(private platform: Platform) {
    this.rootPage = LoginPage;

    platform.ready().then(() => {
      // Okay, so the platform is ready and our plugins are available.
      // Here you can do any higher level native things you might need.
      StatusBar.styleDefault();
    });
  }
}

ionicBootstrap(MyApp, [
  FIREBASE_PROVIDERS,
  defaultFirebase({
    apiKey: myAppKey,
    authDomain: 'myapp.firebaseapp.com',
    databaseURL: 'https://myapp.firebaseio.com',
    storageBucket: 'myapp.appspot.com',
  }),
  firebaseAuthConfig({})
]);
```

login.html

```
<ion-header>
  <ion-navbar>
    <ion-title>Home</ion-title>
  </ion-navbar>
</ion-header>
```

```
<ion-content padding class="login">
  <button (click)="facebookLogin()">Login With Facebook</button>
</ion-content>
```

login.ts

```
import {Component} from '@angular/core';
import {Platform} from 'ionic-angular';
import {AngularFire, AuthMethods, AuthProviders} from 'angularfire2';
import {Facebook} from 'ionic-native';

declare let firebase: any; // There is currently an error with the Firebase files, this will
fix it.

@Component({
  templateUrl: 'build/pages/login/login.html'
})
export class LoginPage {

  constructor(private platform: Platform, public af: AngularFire) {

  }

  facebookLogin() {
    Facebook.login(['public_profile', 'email', 'user_friends'])
      .then(success => {
        console.log('Facebook success: ' + JSON.stringify(success));
        let creds =
firebase.auth.FacebookAuthProvider.credential(success.authResponse.accessToken);
        this.af.auth.login(creds, {
          provider: AuthProviders.Facebook,
          method: AuthMethods.OAuthToken,
          remember: 'default',
          scope: ['email']
        }).then(success => {
          console.log('Firebase success: ' + JSON.stringify(success));
        }).catch(error => {
          console.log('Firebase failure: ' + JSON.stringify(error));
        });
      }).catch(error => {
        console.log('Facebook failure: ' + JSON.stringify(error));
      });
  }
}
```

Read Social Login with Angularfire2/Firebase online:
https://riptutorial.com/ionic2/topic/5518/social-login-with-angularfire2-firebase

Chapter 14: Unit Testing

Introduction

Unit Testing in general gives additional safety to a product to prevent issues when modifying/adding features. A safety net that says "EVERYTHING STILL WORKS". Unit Tests do not replace in any way the actual user tests that a proper QA can do.

In this document we will base the examples on this repository: https://github.com/driftyco/ionic-unit-testing-example

Examples

Unit Tests with Karma/Jasmine

Unit testing in ionic is the same as in any angular app.

We'll be using a few frameworks to do this.

Karma - a framework for running tests

Jasmine - a framework for writing tests

PhantomJS - an application that runs javascript without a browser

First of all lets install everything, so make sure your package.json includes these lines in the dev dependencies. I feel its important to note that that dev dependencies don't affect your app at all and are just there to help the developer.

```
"@ionic/app-scripts": "1.1.4",
"@ionic/cli-build-ionic-angular": "0.0.3",
"@ionic/cli-plugin-cordova": "0.0.9",
"@types/jasmine": "^2.5.41",
"@types/node": "^7.0.8",
"angular2-template-loader": "^0.6.2",
"html-loader": "^0.4.5",
"jasmine": "^2.5.3",
"karma": "^1.5.0",
"karma-chrome-launcher": "^2.0.0",
"karma-jasmine": "^1.1.0",
"karma-jasmine-html-reporter": "^0.2.2",
"karma-sourcemap-loader": "^0.3.7",
"karma-webpack": "^2.0.3",
"null-loader": "^0.1.1",
"ts-loader": "^2.0.3",
"typescript": "2.0.9"
```

To go over packages a bit

```
"angular2-template-loader": "^0.6.2", - will load and compile the angular2 html files.

"ts-loader": "^2.0.3", - will compile the actual typescript files

"null-loader": "^0.1.1", - will not load the assets that will be missing, such as fonts and
images. We are testing, not image lurking.
```

We should also add this script to our package.json scripts:

```
"test": "karma start ./test-config/karma.conf.js"
```

Also take note in tsconfig that you are excluding the spec.ts files from compilation:

```
 "exclude": [
    "node_modules",
    "src/**/*.spec.ts"
  ],
```

Ok, now lets take the actual testing configuration. Create a `test-config` folder in your project folder. (Just as it was mentioned in the package.json script) Inside the folder create 3 files:

`webpack.test.js` - which will tell the webpack what files to load for the testing process

```
var webpack = require('webpack');
var path = require('path');

module.exports = {
  devtool: 'inline-source-map',

  resolve: {
    extensions: ['.ts', '.js']
  },

  module: {
    rules: [
      {
        test: /\.ts$/,
        loaders: [
          {
            loader: 'ts-loader'
          } , 'angular2-template-loader'
        ]
      },
      {
        test: /\.html$/,
        loader: 'html-loader'
      },
      {
        test: /\.(png|jpe?g|gif|svg|woff|woff2|ttf|eot|ico)$/,
        loader: 'null-loader'
      }
    ]
  },

  plugins: [
    new webpack.ContextReplacementPlugin(
      // The (\\|\/) piece accounts for path separators in *nix and Windows
      /angular(\\|\/)core(\\|\/)(esm(\\|\/)src|src)(\\|\/)linker/,
      root('./src'), // location of your src
      {} // a map of your routes
    )
  ]
};

function root(localPath) {
  return path.resolve(__dirname, localPath);
}
```

`karma-test-shim.js` - which will load the angular related libraries, such as zone and test libraries as well as configure the module for testing.

```
Error.stackTraceLimit = Infinity;

require('core-js/es6');
require('core-js/es7/reflect');

require('zone.js/dist/zone');
require('zone.js/dist/long-stack-trace-zone');
require('zone.js/dist/proxy');
require('zone.js/dist/sync-test');
require('zone.js/dist/jasmine-patch');
require('zone.js/dist/async-test');
require('zone.js/dist/fake-async-test');

var appContext = require.context('../src', true, /\.spec\.ts/);

appContext.keys().forEach(appContext);

var testing = require('@angular/core/testing');
var browser = require('@angular/platform-browser-dynamic/testing');

testing.TestBed.initTestEnvironment(browser.BrowserDynamicTestingModule,
browser.platformBrowserDynamicTesting());
```

`karma.conf.js` - defines the configuration of how to test with karma. Here you can switch from Chrome to PhantomJS to make this process invisible and faster among other things.

```
var webpackConfig = require('./webpack.test.js');

module.exports = function (config) {
  var _config = {
    basePath: '',

    frameworks: ['jasmine'],

    files: [
      {pattern: './karma-test-shim.js', watched: true}
    ],

    preprocessors: {
      './karma-test-shim.js': ['webpack', 'sourcemap']
    },

    webpack: webpackConfig,

    webpackMiddleware: {
      stats: 'errors-only'
    },

    webpackServer: {
      noInfo: true
    },

    browserConsoleLogOptions: {
      level: 'log',
      format: '%b %T: %m',
      terminal: true
    },

    reporters: ['kjhtml', 'dots'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome'],
    singleRun: false
  };

  config.set(_config);
};
```

Now that we configured everything lets write some actual test. For this example we will write an app.component spec file. If you would like to see tests for a page and not the main component you can look here: https://github.com/driftyco/ionic-unit-testing-example/blob/master/src/pages/page1/page1.spec.ts

What we need to do first is to test out our constructor. This will create and run the constructor of our app.component

```
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [MyApp],
      imports: [
        IonicModule.forRoot(MyApp)
      ],
      providers: [
        StatusBar,
        SplashScreen
      ]
    })
  }));
```

The declaration will include our main ionic app. The Imports will the imports needed for this test. Not everything.

The providers will include the things that are injected in to the constructor but are not part of the import. For instance the app.component injects the Platform service but since its a part of the IonicModule there is no need to mention it in the providers.

For the next tests we will need to get an instance of our component:

```
  beforeEach(() => {
    fixture = TestBed.createComponent(MyApp);
    component = fixture.componentInstance;
  });
```

Next a few tests to see that everything is in order:

```
  it ('should be created', () => {
    expect(component instanceof MyApp).toBe(true);
  });

  it ('should have two pages', () => {
    expect(component.pages.length).toBe(2);
  });
```

So in the end we will have something like this:

```
import { async, TestBed } from '@angular/core/testing';
import { IonicModule } from 'ionic-angular';

import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';

import { MyApp } from './app.component';

describe('MyApp Component', () => {
  let fixture;
  let component;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [MyApp],
      imports: [
        IonicModule.forRoot(MyApp)
      ],
      providers: [
        StatusBar,
        SplashScreen
      ]
    })
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(MyApp);
    component = fixture.componentInstance;
  });

  it ('should be created', () => {
    expect(component instanceof MyApp).toBe(true);
  });

  it ('should have two pages', () => {
    expect(component.pages.length).toBe(2);
  });

});
```

Run the tests by

```
npm run test
```

And that's about it for the basic testing. There are a few ways to shortcut the test writing like writing your own TestBed and having inheritance in tests which might help you out in the long run.

Read Unit Testing online: https://riptutorial.com/ionic2/topic/9561/unit-testing

Chapter 15: Using Services

Remarks

One very important thing about using shared services, is that they must be included in the `providers` array of the top-most component where they must be shared.

Why is that? Well, Let's suppose that we include the `MyService` reference in the `providers` array from each `Component`. Something like:

```
@Component({
  templateUrl:"page1.html",
  providers: [MyService]
})
```

And

```
@Component({
  templateUrl:"page2.html",
  providers: [MyService]
})
```

That way **a new instance of the service will be created for each component** so the instance where one page will save the data, will be different from the instance used to get the data. So that won't work.

In order to make the entire app use the same instance (making the service work as a *singleton* service) we can add its reference in the `App Component` like this:

```
@Component({
  template: '<ion-nav [root]="rootPage"></ion-nav>',
  providers: [MyService]
})
```

You could also add the `MyService` reference in the `ionicBootstrap(MyApp, [MyService]);` but according Angular2 style guides

> Do provide services to the Angular 2 injector at the top-most component where they will be shared.
>
> Why? The Angular 2 injector is hierarchical.
>
> Why? When providing the service to a top level component, that instance is shared and available to all child components of that top level component.
>
> Why? This is ideal when a service is sharing methods or state.
>
> Why? This is not ideal when two different components need different instances of a

> service. In this scenario it would be better to provide the service at the component level that needs the new and separate instance.

And

> It will work. It's just not a best practice. **The bootstrap provider option is intended for configuring and overriding Angular's own preregistered services**, such as its routing support.

... the `App Component` would be the best choice.

Examples

Share information between different pages

One of the easiest examples of using *shared services* is when we want to store some data from a given page of our application, and then get that data again but from another page.

One option could be to send that data as a parameter (for instance, if one page calls the other one) but if we want to use that data from a completely different part of the application, that seems to be not the best way to do it. That's when *shared services* comes to play.

In this example, we're going to use a simple service called `MyService` which only has two simple methods: `saveMessage()` to store a string and `getMessage()` to get it again. This code is part of this working plunker where you can see it in action.

```
import {Injectable} from '@angular/core';

@Injectable()
export class MyService {

  private message: string;

  constructor(){ }

  public saveMessage(theMessage: string): void {
    this.message = theMessage;
  }

  public getMessage(): string {
    return this.message;
  }
}
```

Then, when we want to store a new message, we can just use the `saveMessage(theMessageWeWantToSave);` method from the `MyService` instance (called just `service`).

```
import { Component } from "@angular/core";
import { MyService } from 'service.ts';

@Component({
  templateUrl:"page1.html"
})
export class Page1 {

  message: string;

  // ...

  public saveSecretMessage(): void {
    this.service.saveMessage(this.message);
  }
}
```

In the same way, when we want to get that data we can use the `getMessage()` method from the service instance like this:

```
import { Component } from "@angular/core";
import { MyService } from 'service.ts';

@Component({
  templateUrl:"page2.html"
})
export class Page2 {

  enteredMessage: string;

  constructor(private service: MyService) {
    this.enteredMessage = this.service.getMessage();
  }

  // ...
}
```

Please remember to check the *Remarks* section to see where should the reference for the `MyService` service be included and why.

Read Using Services online: https://riptutorial.com/ionic2/topic/4407/using-services

Chapter 16: Using Tabs

Remarks

Always remember to check out Ionic 2 Tab docs to be aware of the latest changes and updates.

Examples

Change selected tab programatically from child Page

You can take a look at the full code in this working Plunker.

In this example I use a shared service to handle the communication between the pages inside the tab (child pages) and the tab container (the component that holds the tabs). Even though you probably could do it with Events I like the shared service approach because is easier to understand and also to mantain when the application starts growing.

TabService

```
import {Injectable} from '@angular/core';
import {Platform} from 'ionic-angular/index';
import {Observable} from 'rxjs/Observable';

@Injectable()
export class TabService {

  private tabChangeObserver: any;
  public tabChange: any;

  constructor(private platform: Platform){
    this.tabChangeObserver = null;
    this.tabChange = Observable.create(observer => {
        this.tabChangeObserver = observer;
    });
  }

  public changeTabInContainerPage(index: number) {
    this.tabChangeObserver.next(index);
  }
}
```

So basically the `TabService` only creates an `Observable` to allow the tabs container to subscribe to it, and also declares the `changeTabInContainerPage()` method that will be called from the child pages.

Then, in each child page (the ones inside the tabs) we only add a button and bind the `click` event to a method that calls the service:

Page1.html

```
<ion-content class="has-header">
```

```
    <h1>Page 1</h1>
    <button secondary (click)="changeTab()">Select next tab</button>
</ion-content>
```

Page1.ts

```
import { Component } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import { TabService } from 'tabService.ts';

@Component({
  templateUrl:"page1.html"
})
export class Page1 {

  constructor(private tabService: TabService) { }

  public changeTab() {
    this.tabService.changeTabInContainerPage(1);
  }
}
```

And finally, in the `TabsPage`, we only subscribe to the service, and then we change the selected tab with `this.tabRef.select(index);`

```
import { Component, ViewChild } from "@angular/core";
import { Page1 } from './page1.ts';
import { Page2 } from './page2.ts';
import { TabService } from 'tabService.ts';

@Component({
  templateUrl: 'tabs.html'
})
export class TabsPage {
  @ViewChild('myTabs') tabRef: Tabs;

  tab1Root: any = Page1;
  tab2Root: any = Page2;

  constructor(private tabService: TabService){
    this.tabService.tabChange.subscribe((index) => {
      this.tabRef.select(index);
    });
  }
}
```

Please notice that we're getting a reference to the Tabs instance by adding `#myTabs` in the `ion-tabs` element, and we get it from the component with `@ViewChild('myTabs') tabRef: Tabs;`

```
<ion-tabs #myTabs>
  <ion-tab [root]="tab1Root" tabTitle="Tab 1"></ion-tab>
  <ion-tab [root]="tab2Root" tabTitle="Tab 2"></ion-tab>
</ion-tabs>
```

Change tab with selectedIndex

Instead of getting a reference to the DOM you can simply change the index of the tab using the selectedIndex attribute on the ion-tabs

HTML:

```
<ion-tabs [selectedIndex]="tabIndex" class="tabs-icon-text" primary >
     <ion-tab tabIcon="list-box"   [root]="tabOne"></ion-tab>
     <ion-tab tabIcon="contacts"  [root]="tabTwo"></ion-tab>
     <ion-tab tabIcon="chatboxes"  [tabBadge]="messagesReceived" [root]="tabFive"></ion-tab>
</ion-tabs>
```

TS:

```
import { Events} from "ionic-angular";

export class tabs {
 public tabIndex: number;
 constructor(e: Events) {
   tabs.mySelectedIndex = navParams.data.tabIndex || 0;
   e.subscribe("tab:change", (newIndex) => this.tabIndex = newIndex);
 }
}
```

If you want to change it from some other controller service you can send an event:

```
e.publish("tab:change",2);
```

Read Using Tabs online: https://riptutorial.com/ionic2/topic/5569/using-tabs

Chapter 17: Workaround for 'show-delete' in deprecation

Examples

Solution

I am developing a mobile app using ionic 2 with Angular 2.

I have an ion-list filled ion-items. I want those ion-item to have the ability to be deleted if needed as presented here on the ionic website.

However, a lot have changed in **ionic 2** since the first version and the above style of one button opening all the **ion-item** at one is not possible anymore since the **show-delete** and **show-reorder** are no longer supported. The only option available is to have **ion-item-sliding** as ion-item, which gives us the ability to slide each item one at a time in order to reveal the delete button.

That is not what I wanted. I wanted one button that opens all ion-item at the same time.

After spending some time on that, I came up with a working solution and managed to achieve the desired outcome using ionic 2, and I am going to share it with you.

Here is my solution:

In the .html file:

```
<ion-header>
  <ion-navbar>
    <ion-buttons start (click)="manageSlide()">
      <button>
        <ion-icon name="ios-remove"></ion-icon>
      </button>
    </ion-buttons>
    <ion-title>PageName</ion-title>
  </ion-navbar>
</ion-header>
```

and for the list:

```
<ion-list #list1>
  <ion-item-sliding #slidingItem *ngFor="let contact of contacts | sortOrder">
    <button #item ion-item>
      <p>{{ item.details }}</p>
      <ion-icon id="listIcon" name="arrow-forward" item-right></ion-icon>
    </button>
    <ion-item-options side="left">
      <button danger (click)="doConfirm(contact, slidingItem)">
        <ion-icon name="ios-remove-circle-outline"></ion-icon>
      Remove
      </button>
    </ion-item-options>
  </ion-item-sliding>
</ion-list>
```

In the **.ts** file, first do your imports:

```
import { ViewChild } from '@angular/core';
import { Item } from 'ionic-angular';
import { ItemSliding, List } from 'ionic-angular';
```

then refer to the html element by declaring a ViewChild:

```
@ViewChild(List) list: List;
```

Finally, add your classes to handle the work:

```
public manageSlide() {

    //loop through the list by the number retreived of the number of ion-item-sliding in the
list
    for (let i = 0; i < this.list.getElementRef().nativeElement.children.length; i++) {

        // retreive the current ion-item-sliding
        let itemSlide = this.list.getElementRef().nativeElement.children[i].$ionComponent;

        // retreive the button to slide within the ion-item-sliding
        let item = itemSlide.item;

        // retreive the icon
        let ic = item._elementRef.nativeElement.children[0].children[1];

        if (this.deleteOpened) {
            this.closeSlide(itemSlide);
        } else {
            this.openSlide(itemSlide, item, ic);
        }
    }

    if (this.deleteOpened) {
        this.deleteOpened = false;
    } else {
        this.deleteOpened = true;
    }
}
```

Then the opening class:

```
private openSlide(itemSlide: ItemSliding, item: Item, inIcon) {
  itemSlide.setCssClass("active-sliding", true);
  itemSlide.setCssClass("active-slide", true);
  itemSlide.setCssClass("active-options-left", true);
  item.setCssStyle("transform", "translate3d(72px, 0px, 0px)")
}
```

And the closing class:

```
private closeSlide(itemSlide: ItemSliding) {
  itemSlide.close();
  itemSlide.setCssClass("active-sliding", false);
  itemSlide.setCssClass("active-slide", false);
  itemSlide.setCssClass("active-options-left", false);
```

}

I hope it will help some you out there.

Enjoy and good coding...

Read Workaround for 'show-delete' in deprecation online:
https://riptutorial.com/ionic2/topic/6620/workaround-for--show-delete--in--ion-list--deprecation

Credits

S. No	Chapters	Contributors
1	Getting started with ionic2	Akilan Arasu, Cameron637, carstenbaumhoegger, Community, FreeBird72, Guillaume Le Mière, Ian Pinto, Ketan Akbari, misha130, Raymond Ativie, sebaferreras, tymspy, Will.Harris
2	Add ionic app to ionic view	Saravanan Sachi
3	Angularfire2 with Ionic2	Fernando Del Olmo
4	Constructor and OnInit	niks
5	From code to the App store - Android	Luis Estevez, misha130
6	Geolocation	Matyas, misha130
7	InAppBrowser	niks
8	Ionic2 CSS components	Ketan Akbari
9	Modals	Raymond Ativie
10	Push notification sent & receive	misha130
11	Setup and Debugging Ionic 2 in Visual Studio Code	misha130, PRIYA PARASHAR
12	Social Login with Angularfire2/Firebase	Cameron637, Gianfranco P.
13	Unit Testing	misha130
14	Using Services	sebaferreras
15	Using Tabs	misha130, sebaferreras
16	Workaround for 'show-delete' in	Amr ElAdawy, Roman Lee

deprecation	