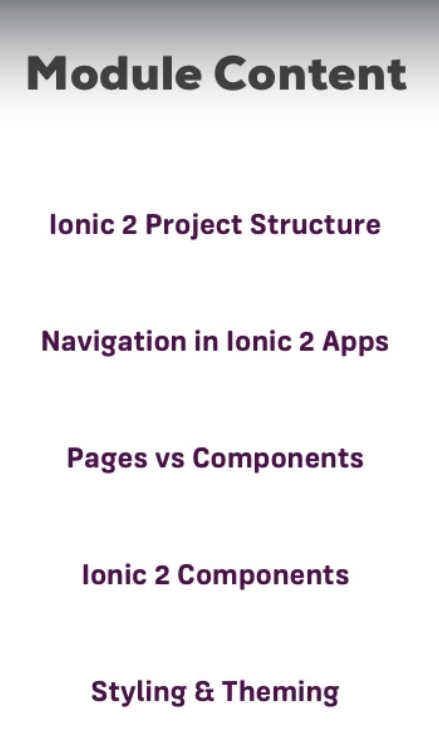
13)Module Introduction

Here we will learn about ionic project structure, what was created for use and how does ionic app get started. Then we will dive how navigation works in ionic app because that differ significantly from angular 2 web application. They we see difference between pages and components and see how they are connected and then dive deeper into pages, how to create them, how to use them and which life cycle they follow. We will alsohave look at ionic components, all those built in components which allows us to create beautiful native application, where we can learn more about them, how to use them and then we will also have a look at how we can adjust the styling and theme of our application to our needs.



Then we will all these things to build our apps.

14)A different way of creating a new Project

Herew e wiwllsee another way of creating project. Run-

**ionic start ionic2-basics blank**

with blank we are defining here that we want to use blank template. We do specify the template and just run  **ionic start ionic2-basics** then we will asked in next steps which template we want to use. In last lecture we used tabs. here we will use blank. blank will not give us an empty application, it will have default structure., the app component and so on but it will not have multiple pages. It won’t have tabs and that is exactly where I want to stat with. So run **ionic serve.** Your application will not have any tabs.

In next lecture I will walk you through all folder and files in project(most important ones atleast) and I will explain how such ionic app starts and how it is connected to normal angular 2 application.

15)Other available Project Templates

So far, we saw the **default** template (=> tabs ) and the **blank** template (=> blank  ).

Other options can be found here: <https://ionicframework.com/docs/cli/starters.html>

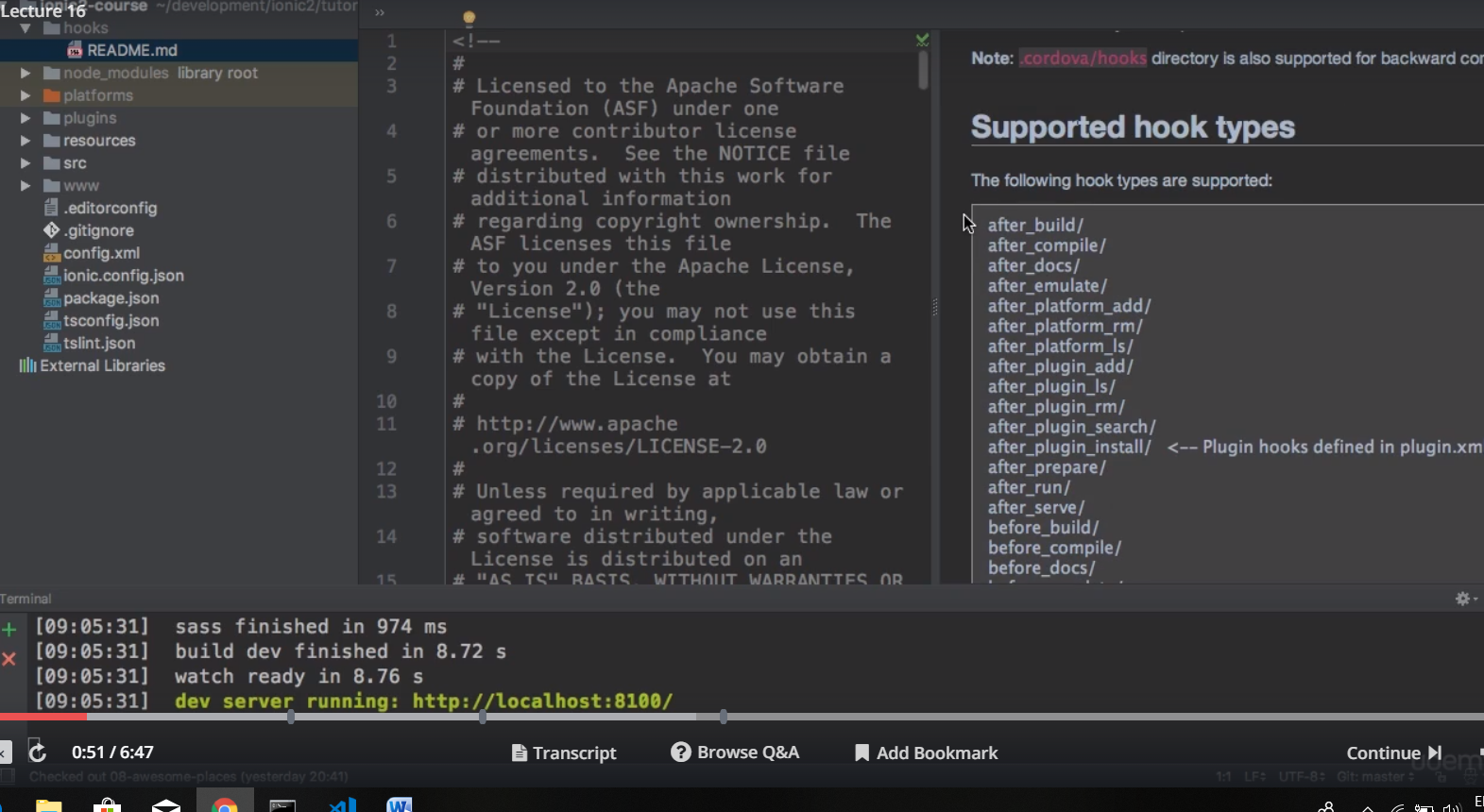
16)Understanding the structure of an ionic 2 Project

Our ionic app has lot of folders. Src is most important folder for us one because this is where we are going to develop our angular app, therefore mobile app in the end, this is where we will spend majority of our time.

However in code I had different structure of app then what max showed. He has answered it this question –

That is fine though -missing folders will be created automatically once you add certain features. So you can continue for now :)

We have hooks folder, you might need it of you are an advanced developer because here you can define certain hooks ( you can have a look at you can look at README.md file to find out more,) which are automatically run , so little scripts scripts I should say which are automatically triggered by cordova during the build process. Here you can see some of hooks that are supported.

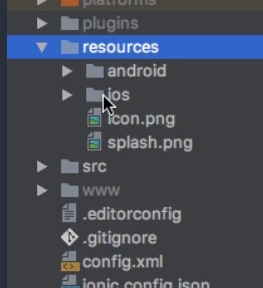


So you can basically instruct cordova to run a certain script after a build has been finished, after build has been run, after it has been server and so on. So some advanced hooks which could then automatically push your code to git, publish it or something like this. I wnt go into that in this course but its good to know that it is there.

Platforms folder is important folder. it basically stores all the platforms you have added to this project. I will come back to it on run it on emulator or real device section. but platforms are basically target you are developing your app for. If you want your app to run on ios then you need to add ios platform here. In case of max, ios folder was automatically added as he created his app on mac. That might not be case with you. Later I will you show how to add platforms. You don’t need to add platforms right now. We can develop our app and test it in the browser without any platforms added. You will learn we add platforms through command , so you do not need to change anything here.

Plugins folder stores all cordova plugins that you might use and that is required to access native device features(like to access camera). We use command line to install plugins, so you dnt need to touch this folder. for each command we run to add plugin, a new folder will be added here. So this folder is managed for you automatically.

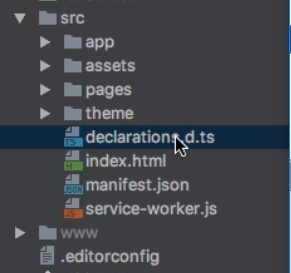
Resources folder is something that we will have a look in deployment section. here you can basically define your own app icons and splash screens. you can define this on on per platform basis, if you do not want to define a shared resource.



And then we will handy command which is shown in deployment section which will kind of automatically scaffold out couple of images ,screens and icons for different devices. This is something we will work with later for.

We already talked about src folder. I will come back to those files in a second but lets first be finished with www folder. this is also a folder that you wnt touch. Because this folder is managed by ionic it holds your web app.so its in between steps before your application gets compiled to a native app through cordova. This is the app you are actually seeing in browser when running ionic serve. So it is created for you automatically with this ionic serve process. It’s in the end your compiled angular2 code. But again not your native app, a web app. All other files are configuration files which allow you to configure your project except for config.xml. we will have look at it in deployment section. here you can configure your app like name of app and son on. Other files are really just for the project.

Back to src folder,



app folder holds app component, app module, so everything you need to start your app, you could say. We will see how app starts in next lectures. In assets folder you could store static images, so if you have an image , that you are going to use in your application which is not fetched from server or something like this. You could store it here and refrence it in your template and then it will be copied , when you compile the application. So nice folder to store static file. pages folder is obviously where we will store all our pages in ionic app and we will soon learn what really pages are and how they compare to angular components. And theme folder is a folder which allows you to adjust the theme or look of your application. We will have detailed look at this file in this section too.

Now other 4 files, declarations.d.ts is a file where you can basically declare your own variables , you want type script to know. You really need to work with file,It’s kind of advance useage if you know that something exists and type script does know . you can add it here. Then we have index.html , this is the file which really kicks off our application which starts our app and ofcourse in the end is taken by cordova and will be all packed into a native code, code that runs on mobile device. But especially for inbetween step we can find the same file here in ww folder. this is where as normal angular app kick starts our application.

manifest.json and service-worker.js are files that I am not going to have lokk in this course. Because these files are required if you rae building progressive web app, which is a web app which is not a native app, its not compiled to native code for devices and therefore it is not in scope of this course. But instead that would be app which you can kind of install on your device directly from a webpage and it kind of runs in the browser and also as anative app depending upon where you started it.

17)How an ionic 2 app works

Here we will see how ionic app is started. How ionic 2 takes our application and kind of wraps itself around it to make it compilable to native applications through cordova. I already showed you this index.html file in last section(in src folder) and as an normal angular web app , we do find this ion-app selector in it.

<ion-app></ion-app>

This not html tag this seems to be selector for one of our components which is then probably picked up by some code in main.js file which is our main bundle, which bootstraps our angular application as you know it from web angular application.

<body>

<!-- Ionic's root component and where the app will load -->

<ion-app></ion-app>

<!-- The polyfills js is generated during the build process -->

<script src="build/polyfills.js"></script>

<!-- The vendor js is generated during the build process

It contains all of the dependencies in node\_modules -->

<script src="build/vendor.js"></script>

<!-- The main bundle js is generated during the build process -->

<script src="build/main.js"></script>

</body>

And remember in this in between steps where we see it in browser, we just have such a web application, it no native app. So this index.html is pretty much same as you might know it in normal web aplicaton. It also includes some imports like cordova.js which is required for some native device functionality and we will come to it later in this course again.

<script src="cordova.js"></script>

But in the end our app gets started here as a normal web app though we willsee a difrence if we look at app folder. here we get the files we would expect , the main.ts , which holds the codes which gets executed when this bundle(main.js, which is imported in index.html) is executed. In main.ts we have code that we have in main.ts for normal angular app , here we bootstrap a module and module we bootstrap is app.module. now this app.moudle looks slightly different from app.module of normal angular app.

App.module-

import { BrowserModule } from '@angular/platform-browser';

import { ErrorHandler, NgModule } from '@angular/core';

import { IonicApp, IonicErrorHandler, IonicModule } from 'ionic-angular';

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

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

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

import { HomePage } from '../pages/home/home';

@NgModule({

declarations: [

MyApp,

HomePage

],

imports: [

BrowserModule,

IonicModule.forRoot(MyApp)

],

bootstrap: [IonicApp],

entryComponents: [

MyApp,

HomePage

],

providers: [

StatusBar,

SplashScreen,

{provide: ErrorHandler, useClass: IonicErrorHandler}

]

})

export class AppModule {}

here we have declarations , we know that. But then it starts with imports ,

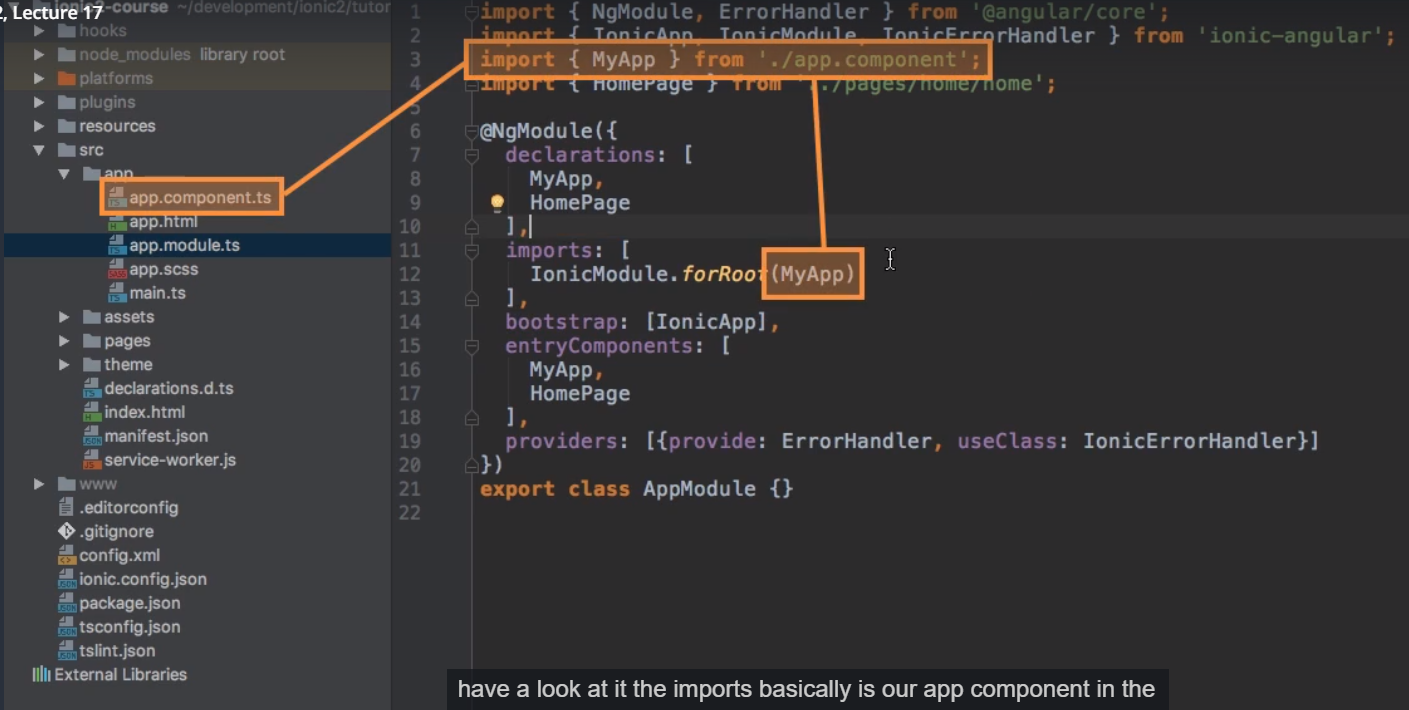
imports: [

BrowserModule,

IonicModule.forRoot(MyApp)

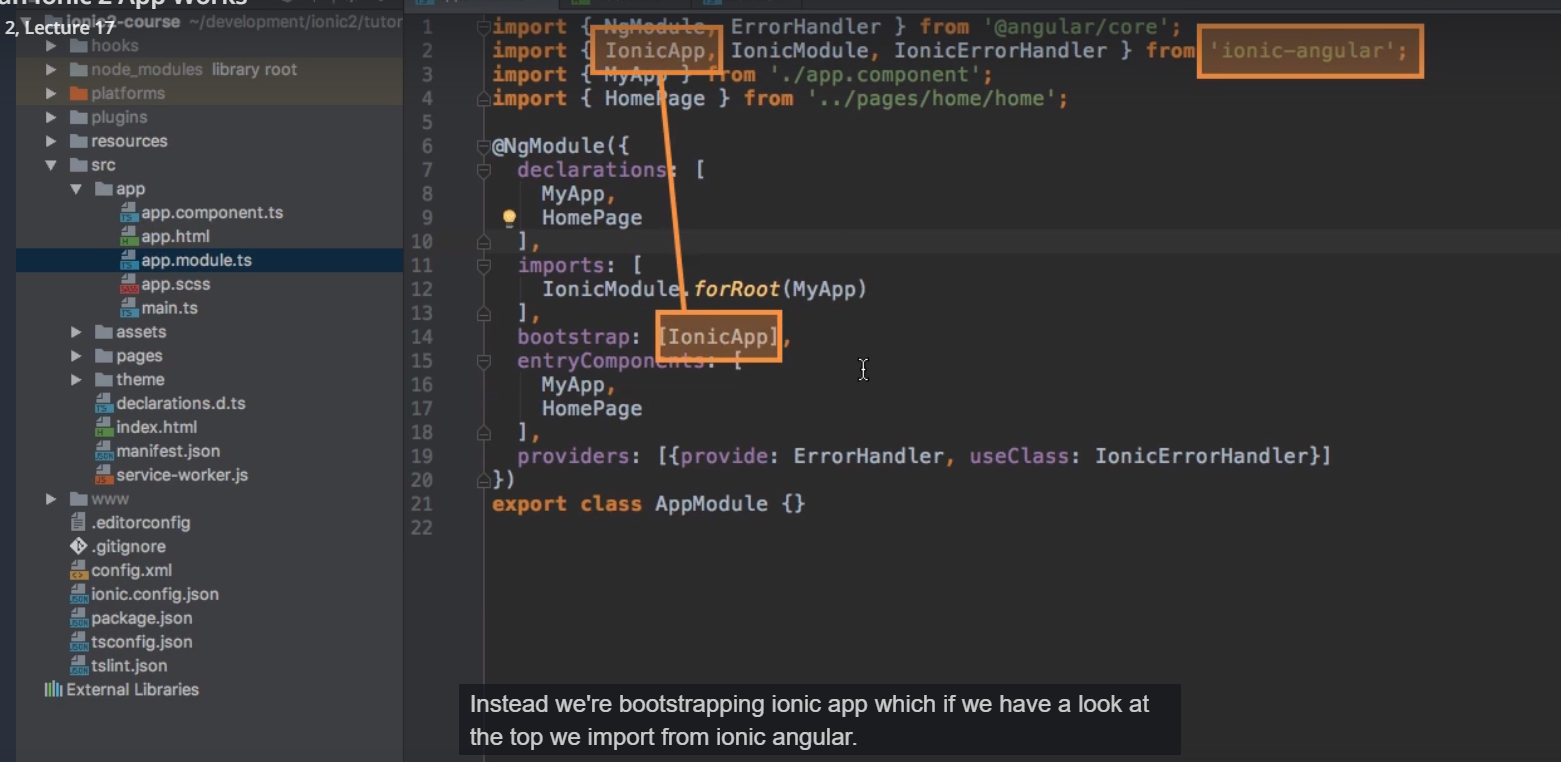
],

here we import some IonicModule, then we execute some forRoot method on it and we pass MyApp as a argument, which is if we look at imports is basically our app component.



In bootstrap array we are bootstrapping our app component, as we would do in a normal angular app. Instead we are bootstrapping ionic app which if we have a look at top we are importing from **ionic-angular.**

bootstrap: [IonicApp],



And then we have this strange entryComponent array-

entryComponents: [

MyApp,

HomePage

],

in providers we are providing something which is just error handler that ionic gives us.

App.module.ts-

import { BrowserModule } from '@angular/platform-browser';

import { ErrorHandler, NgModule } from '@angular/core';

import { IonicApp, IonicErrorHandler, IonicModule } from 'ionic-angular';

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

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

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

import { HomePage } from '../pages/home/home';

@NgModule({

declarations: [

MyApp,

HomePage

],

imports: [

BrowserModule,

IonicModule.forRoot(MyApp)

],

bootstrap: [IonicApp],

entryComponents: [

MyApp,

HomePage

],

providers: [

StatusBar,

SplashScreen,

{provide: ErrorHandler, useClass: IonicErrorHandler}

]

})

export class AppModule {}

so bootstrap, imports and entry component are not how we used to see in normal angular app.lets start with IonicModule that we import. It is module provided by ionic 2 which includes lot of functionalities we will need, for example it also already incudes forms module and http modules. So if yu are working with forms and http in angular 2 app you dnt need to import FormsModule(or ReactibeFormsModule) and HttpModule, its already included in IonicModule. It also does something else, it has forRoot method and we pass our app component to it, if we have a look at our app.component

import { Component } from '@angular/core';

import { Platform } from 'ionic-angular';

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

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

import { HomePage } from '../pages/home/home';

@Component({

templateUrl: 'app.html'

})

export class MyApp {

rootPage:any = HomePage;

constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {

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();

splashScreen.hide();

});

}

}

Besides to code in constructor which I will come back in a second, it looks like a normal app component. so we pass app component to forRoot method, what this will do is basically ,it will configure ionic 2 to wrap itself around our app component, our app. And this is needed because we need some extra features that ionic 2 provides. For example those modules that it automatically gives us but also some other features it kind of needs to set up for it to work correctly on a native device for example to initialize cordova, make sure that we have access to those device features and so on. It also gives us access to all angular 2 components ionic 2 ships with. So that is all happening behind the scenes and this is why we can’t bootstrap our own app instead why we bootstrap ionic app which in the end will the take into account the configuration done by ionic module, which again keep this in mind wraps itself or kind of uses app component. so we are kind of bootstraping our app component plus plus you could say. IonicApp simple wraps around our own app but simply gives us some extra functionlaity. This all what is happening here. But its important to understand.

entryComponents is something you might have never seen in angular web app. entyComponent is rarely used in angular web app, because entry component allows you to force angular to create some factories for your components. That means to be prepared to give you instances of those components. In declaration you register your compoennts to make angular app aware of them, then in normal angular web app, how are you using your components? Either by palcing selector in html or by routing. It turns out in ionic 2 you don’t use either of the 2 for your pages. Pages is something I will come back to. What pages are basically the things you see on your different well, pages in your app. You can still use angular components and embed through their selectors. If we look at page here in home folder, home.html-

<ion-header>

<ion-navbar>

<ion-title>

Ionic Blank

</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding>

The world is your oyster.

<p>

If you get lost, the <a href="http://ionicframework.com/docs/v2">docs</a> will be your guide.

</p>

</ion-content>

Here also we do use components, the ones’ provided by ionic. but in the end these are all components you could have written.so you can still use those as selectors. And in this case you would’nt have to define those components which you use through selectors in app module(not in entryComponents to be precise). But you will learn in next lectures that ionic has different ides of navigating through app. It does not use angular router. Instead it will have a different concept of loading different screens in your app and each screen( and I will come back to it) has to be registered here (in entryComponents)to force angular to be able to give instances of this because normally angular is able to see in your code if you are about to use a component, you either have a selector or you are referring to it by router. Well if you don’t’t do either of 2 it has no chance of knowing that you are about to use it and therefore you have to inform it here in entryComponents.

So this is all about app.module, key take away is you have to register some components here in entryCompoennts(again I will come back to this) and you bootstrap ionic app instead of your own app components but this in the end wraps itself around your app component plus some extra features.

Now lets go to app component, again this is’nt in the end loaded through ionic 2.

import { Component } from '@angular/core';

import { Platform } from 'ionic-angular';

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

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

import { HomePage } from '../pages/home/home';

@Component({

templateUrl: 'app.html'

})

export class MyApp {

rootPage:any = HomePage;

constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {

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();

splashScreen.hide();

});

}

}

What we see here is that it looks pretty much like normal app component ,through one important thing to recognize is that we have extra code here in the constructor and this some code which kind of should be easy to understand. It is basically a callback added by ionc which allows us to run some code after we know that the platform (platform is ios or android) has been initialized successfully. Ofcourse when your app starts up it still is’t loaded completely on the device. this is where you typically see this splash screen, the loading screen and the platform.ready which is executed right at beginning, we kind of can listen to , when has this app been fully loaded on that device and then execute some code for example, hide the splash screen, which is done here automatically. We rarely need to work with this, but if have some code that you want to make sure you only execute after the is really running on device and has been fully loaded, this would be great place to place it in. but again default code here most f the time does the trick.

Lastly lets look at app.component.html, template of our app component.

App.html-

<ion-nav [root]="rootPage"></ion-nav>

This might look strange. it is related to how navigation works in ionic though and that’s something we will have look at in next lectures so that it becomes clear in the end , the startup process is like you know it from a normal angular app. The most important take away is ionic wraps itself around your app component. lets now see difference between components and pages and see then how navigation works and that should also then clear up what entry component does.

18)Pages vs Components

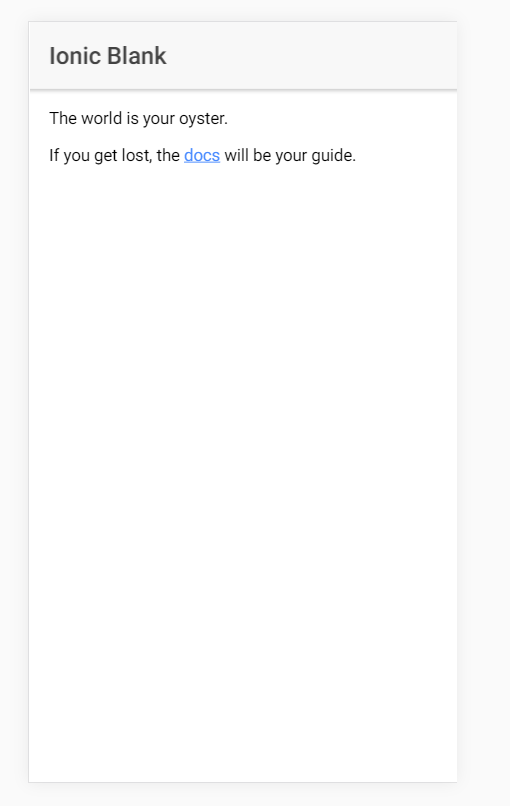
In normal angular 2 application we use components to build that app. We have root component, the app component with its template. In this template we have selector of other components or we use angular router to load different components depending upon which route we are on.

This does not work like this in ionic 2. We can still place selectors of compoents there. In the end , ion-nav is just a component.

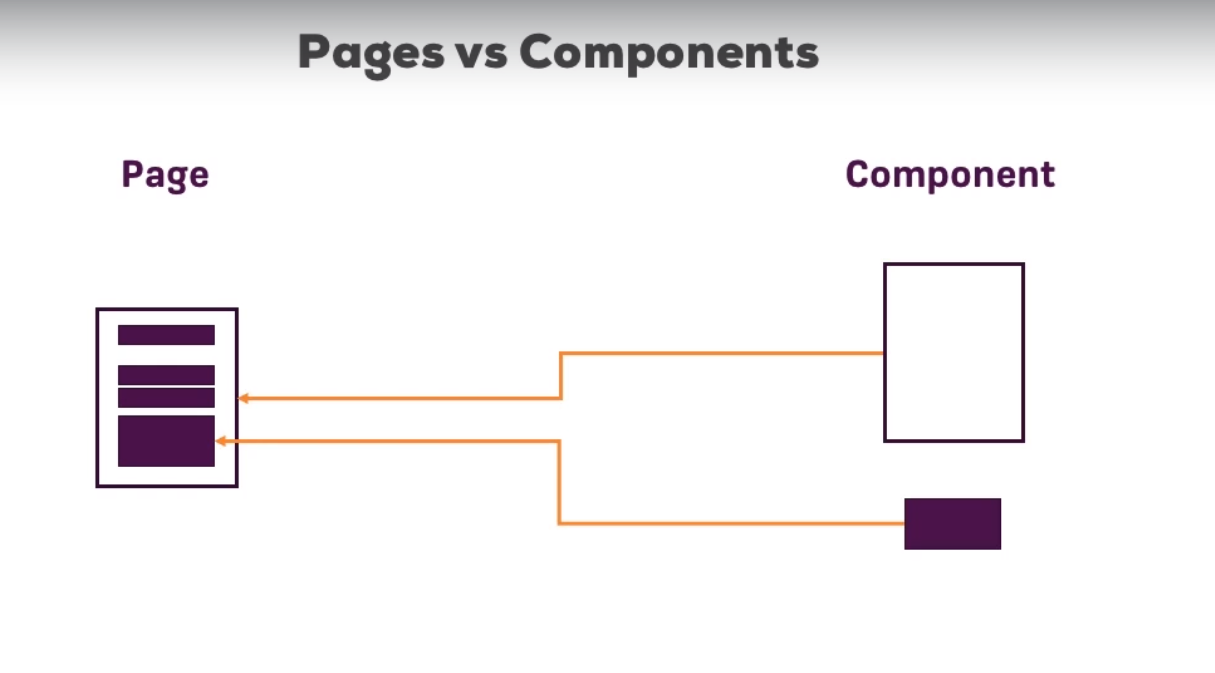
App.html-

<ion-nav [root]="rootPage"></ion-nav>

But pages that we see in app, like this starting blank page-



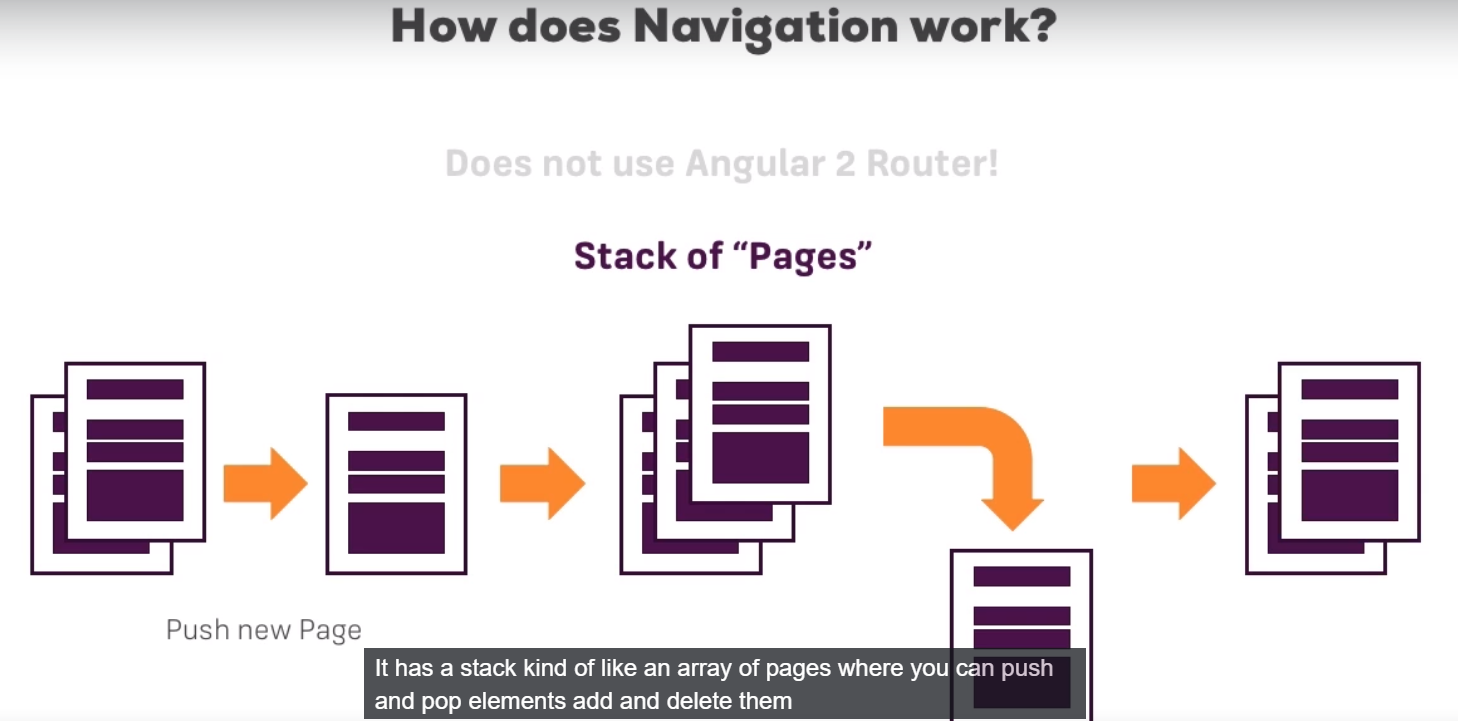
Are not rendered by placing selector of it anywhere. Beuacse this page, page with page-home selector(go to pages folder then to home) is not placed in app.html(template of our root component) instead ionic has different way of navigating between pages. This brings us to important question, what is page? You can think of a page what you see on the whole screen , so page takes up your full screen. It is normal angular 2 component and this is super important to understand . it is normal component, we call it page because it takes up the full screen and it is loaded through ionic 2’s navigation concept, you could say. to which I will come back in next lecture. A component on the other hand is for one whole page as I just said page is a component but could also be part of page. We are still able to include normal angular 2 components on a page. So not every component is a page but every page is the component. to clear the confusion we will see navigation in ionic app.



19)How navigation works in Ionic 2 app

Until now we learnt how ionic app starts and works. But I am sure , It’s a bit confusing to understand what pages and how exactly they get rendered. To understand how they are rendered , we need to understand how navigation works. Important thing is ionic does not use angular router instead it has its own navigation concept which is stack of pages and remember page are normal components. Its just that well, they render our full screen so they contain everything we want to see on the screen and then ionic 2 manages stack of such pages, of such components in the background. This is why it needs to be started with its own root app which then wraps our root app because this navigation stack here, this stack of pages needs to be initilaized and this is what ionic 2 behind the scenes.

We are then able to push new pages on stack and important thing about stack is we always view the top most page. This is the page which is currently displayed on the screen. So there might be multiple pages on the stack and we only view the top most. In this case page which was recently pushed on the stack. Well ofcourse we can go back by popping off the page which means now the stack only has 2 pages and viewing this second page the top most one.



This is how ionic 2 works. It has a stack, kind of like an array of pages where you can push and pop elements, add and delete them and view the top most one. Lets see this in action to fully understand how this works and what page are.

20)Initializing navigation in the AppComponent

Now with the additional knowledge about how navigation works and it has this stack of pages at its core. It makes much more sense to see the content of our app.html file.

App.html-

<ion-nav [root]="rootPage"></ion-nav>

Did you wondered why we only have this strange ion-nav component in here? Well ion-nav is built in component which ship with ionic and that is true for any components starting with **ion-**. So what ion-nav does is, it initializes this stack of pages, it iniliazes this ionic-nav controller, a class ionic 2 ships with which is responsible for managing our stack of pages. This is why in app module we are bootsraping ionic app and not our own root component and why we instead pass our own root component to ionic module, because ionic app, ionic 2 in background need to set up this stack of pages , needs to initialize this app and it does this in its own code, in its own module, inits own component and our component is only passed in and tells it that please initialize navigation and very importantly initialize it with this root page-

<ion-nav [root]="rootPage"></ion-nav>

We are binding root property on ion-nav component and we are passing in a property which holds the page ,which should be root page of our application because if you have stack of pages which decides which page you are currently viewing, well you obviously need to have a bottom page atleast which is the page you are viewing at start. And this is the so-called root page.

App.compoent.ts-

import { Component } from '@angular/core';

import { Platform } from 'ionic-angular';

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

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

import { HomePage } from '../pages/home/home';

@Component({

templateUrl: 'app.html'

})

export class MyApp {

rootPage:any = HomePage;

constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {

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();

splashScreen.hide();

});

}

}

This rootPage property is setup in app component and it points to home page, the only page we have. This is default page . and this how navigation stack is setup at the beginning.

In next lectures lets add more pages to our app and see how to navigate around.

21)Ionic generate page XY problems

If your project uses Ionic 3 (check the package.json to see the version number), ionic generate page XY  will give you a slightly different output than shown in the videos:

a) You get an additional .module.ts  file: This file might contain a little bug, make sure it uses IonicPageModule.forChild(...)  and also import { IonicPageModule } from 'ionic-angular'   and NOT IonicModule.forChild(...) !

b) Your page will be named YourName  and not YourNamePage . Refer to it as YourName  in your code then (and not YourNamePage ).

Besides that, it works the same. Make sure to add it to your declarations[] and entryComponents[] array in AppModule and you should be good to go.

To see which version of ionic you are using , go to project folder and run this-

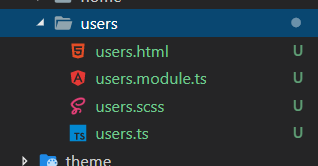
**ionic info**

22)Creating a page and how to navigate to it

In order to navigate around we need to crate more page, right now we have only one page. Now there are 2 ways of creating pages, first one is easy one is probably the quicker one, the automatic way. We can use ionic cli and run this command-

**ionic generate page users**

here users is the name of page. It will create users page in pages folder. in this folder we have ts, html, module.ts and scss.



I will come back to what scss file. in ionic 2 this module file is not generated(max is using ionic 2), but as i am using ionic 3 this file is generated.

Users.ts-

import { Component } from '@angular/core';

import { IonicPage, NavController, NavParams } from 'ionic-angular';

/\*\*

\* Generated class for the UsersPage page.

\*

\* See https://ionicframework.com/docs/components/#navigation for more info on

\* Ionic pages and navigation.

\*/

@IonicPage()

@Component({

selector: 'page-users',

templateUrl: 'users.html',

})

export class UsersPage {

constructor(public navCtrl: NavController, public navParams: NavParams) {

}

ionViewDidLoad() {

console.log('ionViewDidLoad UsersPage');

}

}

It is normal component and downside of using ionic command is that it gives us a lot of boillerplate like for example. Like we dnt need this comment. We do not need this import-

import { IonicPage, NavController, NavParams } from 'ionic-angular';

because we do not need to inject these things. I will come back to them later. Also we do not need this strange method. What we need is-

import { Component } from '@angular/core';

import { IonicPage } from 'ionic-angular';

@IonicPage()

@Component({

selector: 'page-users',

templateUrl: 'users.html',

})

export class UsersPage {

}

In video even @IonicPage is not there. In Q and A of last lecture, I found why this additional @IonicPage and that extra module file is linked. Here is link-

<https://www.udemy.com/ionic-2-the-practical-guide-to-building-ios-android-apps/learn/v4/questions/2207990>

so here i will go with @IonicPage approach. It is for lazy loading. So this user.ts file is normal compoenent and I am stressing this because it is important. Now our component has a external template(we can also use internal).

Users.html-

<ion-header>

<ion-navbar>

<ion-title>users</ion-title>

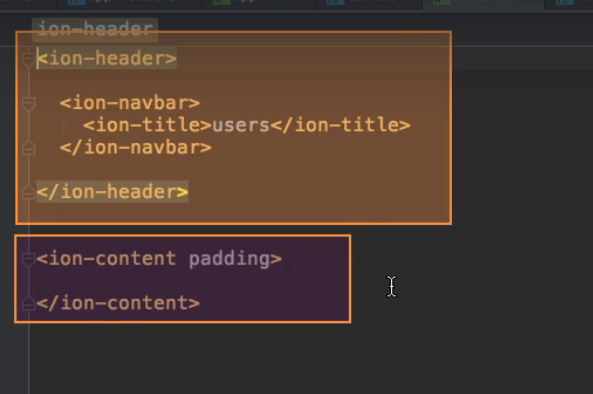
</ion-navbar>

</ion-header>

<ion-content padding>

</ion-content>

Here we have this header and content section marked by ion-header and ion-content compoennts.



Compoents that ionic page shipd with. A typical page always has a header and a content, you can also add footer with ion-footer selector. so these are 3 things you have in typical page-

<ion-header>

<ion-navbar>

<ion-title>users</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding>

</ion-content>

<ion-footer>

</ion-footer>

In header we have another built in component the ion-navbar which basically sets up this tool bar at top.it is because of this ion-navbar we see back button at top when we navigate to a page. With in that we have another component ion-title. And these are all compoentns shipping with ionic 2 which provide nice styling, place those things correctly , make them look nice on mobile devices. Lets change the title to **The Users** and in content I want to place button s for different users you could say. Sow e place a button, to make them look nicer, ionic ships with a directive ion-button. I will come back to tis directive and ionic components shortly. In conetne t we add 2 buttons. Now w emove to footer, here we add directive **padding** which is already placed on ion-content. This is another built in helper directive that ionic 2 ships with which simply adds some padding to make it look nice, to have content not sit directly on the edges.

Users.html-

<ion-header>

<ion-navbar>

<ion-title>The Users</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding>

<button ion-button>User 'Sumeet'</button>

<hr>

<button ion-button>User 'Sood'</button>

</ion-content>

<ion-footer padding>

<p></p>

</ion-footer>

Now we do not have a way to reach to this users page. Default page is home page. In home page we add a button, on clicking this button we wwant to navigate. We excute a function on click, this is angular code. Then in home.ts we implement that function.

Home.html-

<ion-header>

<ion-navbar>

<ion-title>

Ionic 3 Basics

</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding>

<button ion-button (click)="onGoToUsers()">Users</button>

</ion-content>

Home.ts-

import { Component } from '@angular/core';

import { NavController } from 'ionic-angular';

@Component({

selector: "page-home",

templateUrl: "home.html"

})

export class HomePage {

constructor(public navCtrl: NavController) {}

onGoToUsers() {

this.navCtrl.push("UsersPage");

}

}

Now in ts file we have already injected NavController. We will need this to navigate around. Now NavController is basically a tool , a class provided by ionic, which hep us to manage this stack of pages. Rememeber this was important concept ionic 2 was using when we talked about navigating around. So NavController is responsible for managing this. Therefore if you want to switch the page we should us the NavController to push a new page on that stack. Because this is again how the navigation works. We simply push and pop items and we always view the top most one. Now our navigation is working.

Not here we have used modules for individual pages, it is concpt introduced in ionic 3. In ionic 2 we have to pass component to push method instead of string(string is name of component class) like this-

import { Component } from '@angular/core';

import { NavController } from 'ionic-angular';

import { UsersPage } from '../users/users';

@Component({

selector: "page-home",

templateUrl: "home.html"

})

export class HomePage {

constructor(public navCtrl: NavController) {}

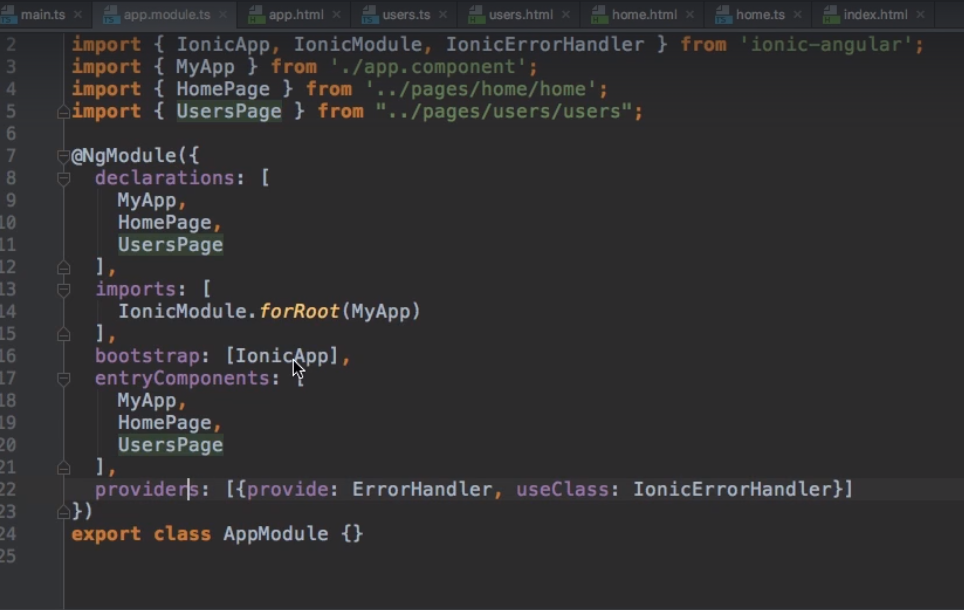
onGoToUsers() {

this.navCtrl.push(UsersPage);

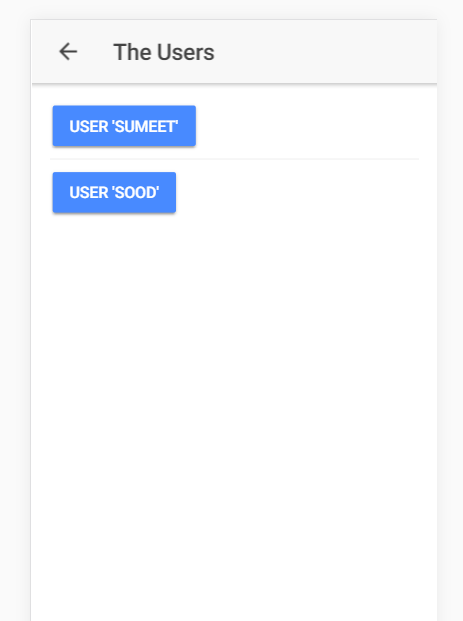
}

}

You also need to add UsersPage to app.module, if you are using second approach. You need to add it in declaration and entryComponents array-



Now we are using multiple pages. Whenw e go to users page we automatiaclly get option of going back to home component.



We can also see that there are nice animations when we switch pages. This is hwat ionic is doing for you, giving you this animation and back button when you use its’s navigation cocept, this stack of pages and this is super – super important to understand, this is core of ionic 2. This idea of navigating around this stack of pages which you manage

23)First Summary

We learned how ionic app gets started. We learned about pages. pages are normal components, they are simply called pages they hold the content for a full page in our application and they are loaded through ionic2’s navigation stack, we push and pop those compoents called pages onto stack. We can still use normal compoennts and you will see this later in this course. You can still create component holding 2 buttons and a div for ex, which you then embedd in your page. This is perfectly possible and we are inded already doing this. In home component we used ion-header, ion-content compoennts. These are components that are shipped with ionic 2. So you can still use compoents which are not pages inside of your pages.

24)An alternative way of creating Pages

Lets dive deeper into ionic navigation and what we can do with it. I told you that there is an alternative way to creating pages automatically by cli. This if we manually create all those files which are generated by cli. Lets say we want to create a user page inside users folder, as f now cli cannot do it. In user.html-

We have 2 important pieces, <ion-header> and <ion-content>. Now in header we typically have <ion-navbar>, this is where tat back button will be added. Then in nav-bar we typically have a title. User.ts-

import { Component } from '@angular/core';

import { IonicPage, NavController, NavParams } from 'ionic-angular';

/\*\*

\* Generated class for the UsersUserPage page.

\*

\* See https://ionicframework.com/docs/components/#navigation for more info on

\* Ionic pages and navigation.

\*/

@IonicPage()

@Component({

selector: 'page-user',

templateUrl: 'user.html',

})

export class UserPage {

name: string;

}

User.html-

<!--

Generated template for the UsersUserPage page.

See http://ionicframework.com/docs/components/#navigation for more info on

Ionic pages and navigation.

-->

<ion-header>

<ion-navbar>

<ion-title>{{name}}</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding>

<p>HI, I am {{name}}</p>

</ion-content>

What we want to do is, we want to load the user page when we click on some user button on users page. We want to display the name of user, whose button we have clicked. So we want to pass data from one page to another.

25)Passing Data between Pages

Now what we want to do is when we click on sumit button, we want to pass sumit string to user page and display it there. We want to do same thing with sood. This is how w edo it-

Users.html-

<ion-header>

<ion-navbar>

<ion-title>The Users</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding>

<button ion-button (click)="onLoadUser('Sumeet')">User 'Sumeet'</button>

<hr>

<button ion-button (click)="onLoadUser('Sood')">User 'Sood'</button>

</ion-content>

<ion-footer padding>

<p>THis is Footer</p>

</ion-footer>

Users.ts-

import { Component } from '@angular/core';

import { IonicPage, NavController, NavParams } from 'ionic-angular';

@IonicPage()

@Component({

selector: 'page-users',

templateUrl: 'users.html',

})

export class UsersPage {

constructor(private navCtrl: NavController) {}

onLoadUser(name: string) {

// this.navCtrl.push('UserPage', name);

this.navCtrl.push('UserPage', {userName: name});

}

}

Note that data is passed as second argument to push method of NavController. We either pass name variable directly or we can pass object or array. then in user page we receive this value-

User.ts-

import { Component, OnInit } from "@angular/core";

import { IonicPage, NavController, NavParams } from "ionic-angular";

@IonicPage()

@Component({

selector: "page-user",

templateUrl: "user.html"

})

export class UserPage implements OnInit {

name: string;

constructor(private navParams: NavParams) {}

ngOnInit() {

this.name = this.navParams.data.userName;

// this.name = this.navParams.get('userName');

}

}

Again there are 2 ways to receive the value , one is commented out as shown.

Then in user.html, we display this valye-

User.html-

<ion-header>

<ion-navbar>

<ion-title>{{name}}</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding>

<p>HI, I am {{name}}</p>

</ion-content>

26)Popping Pages – going back

Lets say that we are at user page and we want to go back without clicking on back button. May be we have confirm button in our app and when we click on it we want to go back programmatically.

NavController is the class which has to do everything with pushing or popping or changes any pages on this navigation stack.

Now we navigated to user page by clicking on users page. Now we want to go back to users page. We do this by-

User.html-

<button ion-button (click)="onGoBack()">Confirm</button>

User.ts-

import { Component, OnInit } from "@angular/core";

import { IonicPage, NavController, NavParams } from "ionic-angular";

@IonicPage()

@Component({

selector: "page-user",

templateUrl: "user.html"

})

export class UserPage implements OnInit {

name: string;

constructor(private navParams: NavParams,

private navCtrl: NavController) {}

ngOnInit() {

this.name = this.navParams.data.userName;

// this.name = this.navParams.get('userName');

}

onGoBack() {

this.navCtrl.pop();

}

}

So we pop the last page and go back to ne step. Lets say we want to pop all pages and go to root page, this can be done by using this-

onGoBack() {

// this.navCtrl.pop();

this.navCtrl.popToRoot();

}

Now you will be moved to root page, popping out all pages. As a side note you cannot pop off root page, so cnt empty your stack because that would basically crash your application.

29)saving time with helpful navigation directives

We learned about the basics of navigating around and we will learn even more in various apps we are going to build. I dnt want to cover tat all in this basics module, which is there to introduce the basics of ionic 2. We will dive much deeper into it in individual apps.

Here we will see some directives. Here to navigate we have to right lot of coe, we have to inject NavController, then we have to call fucntio on click. There is shortcut for that. We will use it in home page. Old way is commented out for your refrence.

Home.html-

<ion-header>

<ion-navbar>

<ion-title>

Ionic 3 Basics

</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding>

<!-- <button ion-button (click)="onGoToUsers()">Users</button> -->

<button ion-button [navPush]="usrPage" [navParams]="'sumit'">Users</button>

</ion-content>

Here you can see that you can also pass data using navParams directive.

Home.ts-

import { Component } from '@angular/core';

import { NavController } from 'ionic-angular';

@Component({

selector: "page-home",

templateUrl: "home.html"

})

export class HomePage {

usrPage = 'UsersPage';

// constructor(public navCtrl: NavController) { }

// onGoToUsers() {

// this.navCtrl.push("UsersPage");

// }

}

We also have directive to go back i.e to pop the stack. In users page we add a new button to go back-

<button ion-button navPop>Go Back</button>

30)Configuring Page transitions

Besides the page you want to go to and data you want to pass along, you can pass a third argument to push() (and a first argument to pop()): [Navigation Options](https://ionicframework.com/docs/v2/api/navigation/NavController/" \l "navoptions)

These options allow you to configure the page transition. You do set your own configuration by passing a JS object where you may set the following properties:

* **animate** (boolean): Whether or not the transition should animate.
* **animation** (string): What kind of animation should be used.
* **direction** (string): The conceptual direction the user is navigating. For example, is the user navigating forward, or back?
* **duration** (number): The length in milliseconds the animation should take.
* **easing** (string): The easing for the animation.

Example:

this.navCtrl.push(NewPage, {}, {

direction: 'back', // default for push is 'forward'

duration: 2000, // 2 seconds

easing: 'ease-out'

});

31)Understanding the Lifecycle of Page

As I said we are going to dive a lot deeper into navigation on how to use this and different use cases in this course when we build real apps. Here we will se lifecycle of page.

By page I mean component which we treat as a page because you load it from stack by pushing and popping, has certain methods which gets executed by ionic 2. And its important to undersand those method only get executed when this page, so when this component is loaded by pushing and popping. I am talking about these methods-

1)ionViewCanEnter

If you add tis method to your page, ionic 2 will execute it. And this method serves as a navigation guard. It has to return true or false or a promise resolving to true or false which simply allows or denies access to the page . so you can use this to protect a page and to see if user is allowed to visit this page or not. You will see this later in course when we use authentication.lets say we are allowed to access the page, the next hook is-

2)ionViewDidLoad

This method is executed right after the page is loaded. And its important that this method is not executed if the page was cached. Ionic 2 will cache your page, for example if you have a page and you click on a button which pushes a new page. The page you were on , is still in stack of pages, therefore it is cached. So if you click on back button on next page and are therefore taken to old page , it will not be recreated, it will be taken from cache. So this method will not be executed. So ionViewDidLoad will not be executed each time you see a page it will only be executed if page is actually created.

Remember this usecase if we have page3 in stack but in page 4 we again push page3. Then in that case IonViewDidLoad will be called again. Because we are pushing new instance of that page on stack.

Thereafter we reach-

3)ionViewWillEnter

Now this is executed whenever a page is about to become active no matter if it’s taken from cache or newly created after that we reach-

4)ionViewDidEnter

This is executed right after page did become active. So once you are able to fully see this page. This is also fired even if page is cahed.

5)ionViewCanLeave

If you want to leave a page then we have this method. This is opposite of ionViewCanenter method. It is also navigation gurad which decides whetherw e are allowed to leave a guard or not. So it also have to return true or false or promise resolving to true or false. Granted we are allowed to leave, we then reach-

6)ionViewWillLeave

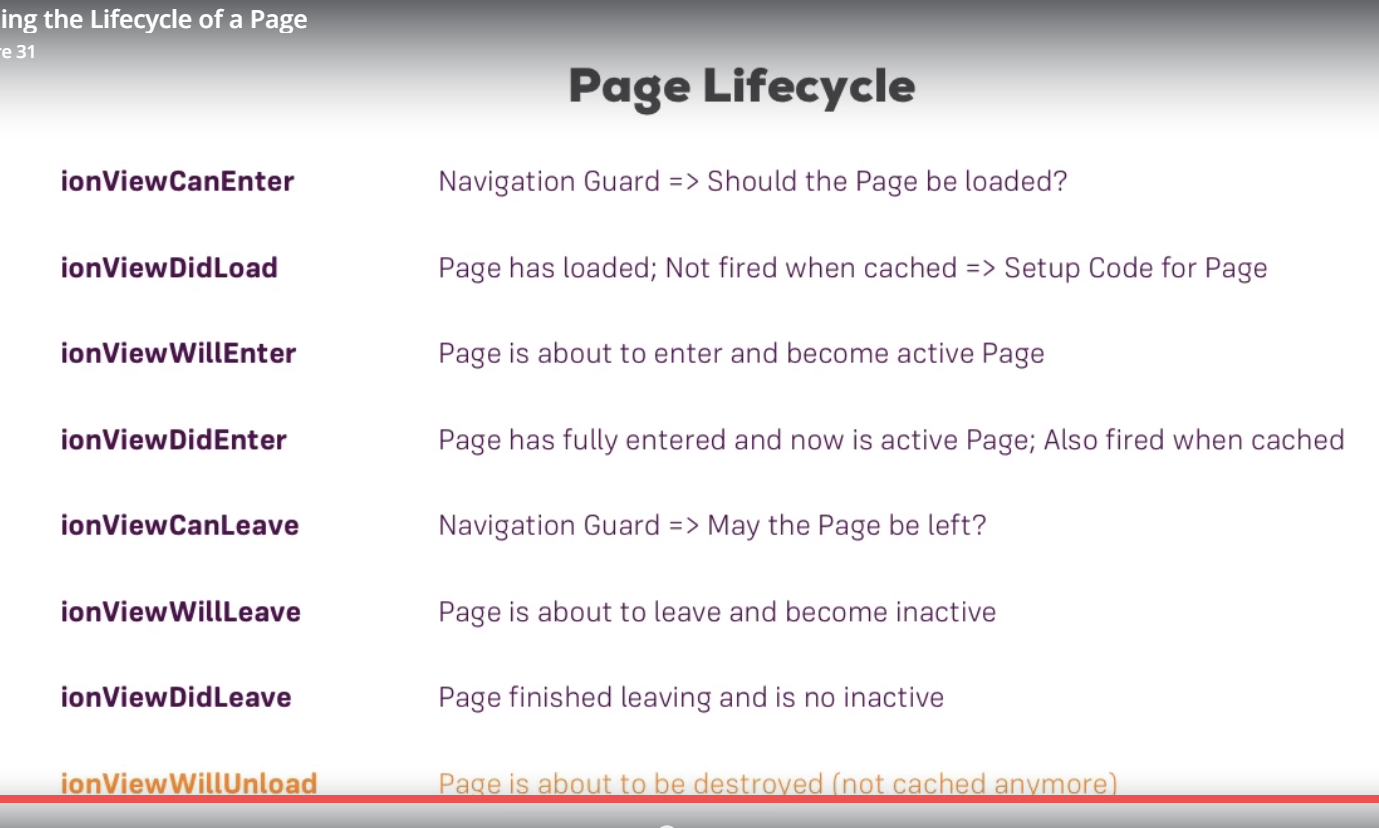
This is executed right before page is about to become inactive. After this we reach-

7)ionViewDidLeave

This is executed after page did become inactive well if page is destroyed and not cahed then this is executed-

8)ionViewWillUnload

This is great place if you need to run some code if you want tat code to be executed only when page is destroyed, to do some cleanup things and like that.



32)The PageLifeCycle hooks in action

Herew e will execute life cycle hooks in users page. Now a page is compoennet, so angular component life cycle hooks still exist on it. In addition to them ionic adds its own life cycle hooks., it adds its own methids. These methods will only be called for compoents which are used as pages which means all are loaded through navigation stack, through NavController. If you create another components and simply embed it in template through selector, you will get angular 2 life cycle but not page life cycle.

You can simply add ionic hooks methods in class. Unlike angular you do not need to implement some interface.

Users.ts-

import { Component } from '@angular/core';

import { IonicPage, NavController, NavParams } from 'ionic-angular';

@IonicPage()

@Component({

selector: 'page-users',

templateUrl: 'users.html',

})

export class UsersPage {

constructor(private navCtrl: NavController) {}

onLoadUser(name: string) {

// this.navCtrl.push('UserPage', name);

this.navCtrl.push('UserPage', {userName: name});

}

ionViewCanEnter() {

console.log('Ion view Can enter');

const rnd = Math.random();

return rnd > 0.5;

}

}

So here we allow access to users page depending upon some random number.now in older ionic version whenever we are not allowed to navigate, i.e navigation is denied like above we receive a error. So we need to nadle that failed case but in latest version of ionic that case is handled and you do not need to do anything. Max also said that it is diffilcult to handle that case when we are using directives for navigation. So we switched to calling method.

This is how we handle it-

Home.html-

<ion-content padding>

<button ion-button (click)="onGoToUsers()">Users</button>

<!-- <button ion-button [navPush]="usrPage" [navParams]="'sumit'">Users</button> -->

</ion-content>

So we switch to calling methos.

Home.ts-

onGoToUsers() {

console.log('method called');

this.navCtrl.push("UsersPage")

.catch(error => console.log('Access denied, argument was '+ error));

}

To handle the case we are not allowed to access a page, I can catch any error. In the end it returns the promise. In the end push method returns us a promise which is resolved when we see the page or when push request is rejected. In this case we can simply catch the error , this will simply be false. However as I told in new ionic version we do not have to do this thing. Even if you do that thing, callback passed to catch is not executed.

If access is allowed then we reach ionViewDidLoad. And like that we reach all methods.

Users.ts-

import { Component } from '@angular/core';

import { IonicPage, NavController, NavParams } from 'ionic-angular';

@IonicPage()

@Component({

selector: "page-users",

templateUrl: "users.html"

})

export class UsersPage {

constructor(private navCtrl: NavController) {}

onLoadUser(name: string) {

// this.navCtrl.push('UserPage', name);

this.navCtrl.push("UserPage", { userName: name });

}

ionViewCanEnter(): boolean | Promise<boolean> {

console.log("Ion view Can enter");

const rnd = Math.random();

return rnd > 0.5;

}

ionViewDidLoad() {

console.log("IonViewDidLoad");

}

ionViewWillEnter() {

console.log("ionViewWillEnter");

}

onViewDidEnter() {

console.log("onViewDidEnter");

}

ionViewCanLeave(): boolean | Promise<void> {

const promise: Promise<void> = new Promise((resolve, reject) => {

setTimeout(() => {

resolve();

}, 1000);

});

return promise;

}

ionViewWillLeave() {

console.log("ionViewWillLeave");

}

ionViewDidLeave() {

console.log("ionViewDidLeave");

}

ionViewWillUnload() {

console.log("ionViewWillUnload");

}

}

In ionViewCanLeave we have returned a promise that returns void. Actually value returned by promise does not matter. If promise is resolved that means access is granted, it it is rejected that means access is denied. Now lets test it.

33)How to use Ionic 2 documentation

We learned about basics in this module. There is lot more to learn about navigation and pages, that is what we will do in applications, that we are going to develop.

Link to docs-

[https://ionicframework.com/docs/#](https://ionicframework.com/docs/)

on left hand side we have couple of points.ionic docs in general is good. First I want to show you api, click on api. Now on left hand side you can see all the classes that ionic 2 ships with. We will use lot of them and will learn about them without reading all the docs. But it is some great place to look at somethings if you want to dive deeper. Especially interesting right now is, nav related parts here(classes starting with nav). Click on NavController. Here you can learn more about pushing and popping and other methods navController offers you. But gain we will use lot of these methods in course, in apps that we are going to build. So its probably not best idea to dive super deep into them.

Another thing I want to show is compoents section. ionic 2 shipd with rich component library.go to home page and click on compoents. Cool thing is , you can click on component and get the live preview on right and these previews are interactive, you can click the buttons and there are some compoents which will actually do something in preview. You can even switch between different operating systems to see how it works . in next section we will lot of these components. Since it is impossible to cover all compoents and all possible configurations, directives you can add and so on. So its good idea to go to docs and see components with which you want to work and see how can you configure it. Again we will use lot of these component in course so need to read them just right away.

Last thing that I want to show is ionicicons. Ionic ships with lot of icons. And they are very easy to use. When you click on some icon you can see the code you need to use. Again we are going to use lot of them. But I want you to be aware of this repository of icons. You can also search icons, cool thing is if you search map, you not only see icon with name map, but also icons which have other name but can be used as map icons. This is cool thing and we will use it a lot.

34)Styling the app and setting a theme

Befire finishing I want to talk about styling. And I wll strat with scss files that are generated with pages. It is css plus you can say. You can write your css code but you get some extra features. Its like ts is for js. in the end scss is compiled to css. Ionic uses scss by default, but you ca srie normal css code in these files.

The great feature that scss offers and it’s all based on sas, so feature is you an nest your selectors.

Users.scss-

page-users {

p{

background-color: red;

padding: 10px;

}

}

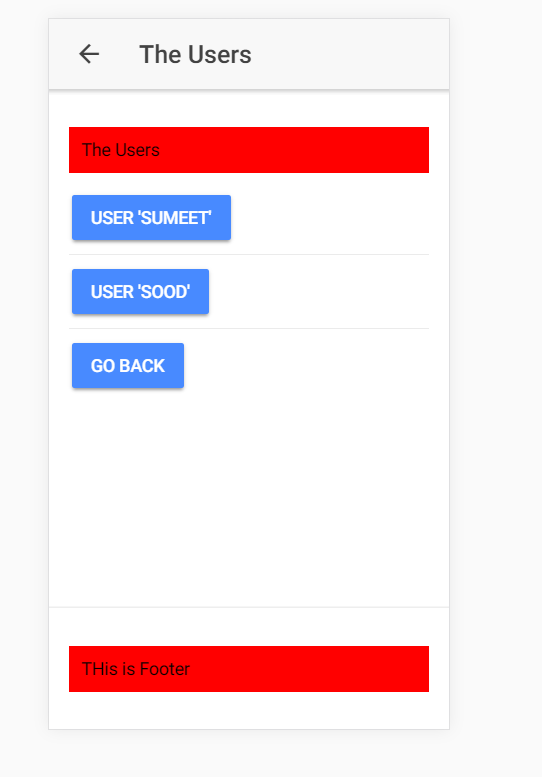
// p{

// background-color: red;

// padding: 10px;

// }

Notice that same results can be achieved if you do not include p tag inside, page-users tag.



Users.html-

<ion-header>

<ion-navbar>

<ion-title>The Users</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding>

<p>The Users</p>

<button ion-button (click)="onLoadUser('Sumeet')">User 'Sumeet'</button>

<hr>

<button ion-button (click)="onLoadUser('Sood')">User 'Sood'</button>

<hr>

<button ion-button navPop>Go Back</button>

</ion-content>

<ion-footer padding>

<p>THis is Footer</p>

</ion-footer>

Now I can add p selector insise page-users selectors and this is important, this nesting. So this is how styling works in angular.

Now you might notice that in @compoennt decorator we do not have styleurl array .

@Component({

selector: "page-users",

templateUrl: "users.html"

})

You can add it to use angular 2 ‘s way of styling but typically in ionic 2/3 we use scss approach. What it will do is it will take all those scss files togather and bundle them into one css file. and you have nested your p tag inside selector of users page like this-

page-users {

p{

background-color: red;

padding: 10px;

}

}

In other words if you have this in users.scss –

p{

background-color: red;

padding: 10px;

}

Then this typing will be applied to all p tags , no matter they are in which page.

However try not to change the app styles like in pages. Scss file. if you want to change the styles for whole application, use the app.scss file.

So this is important to understand there is no view encapsulation as in normal angular app by default though you could enforce it by adding StyleUrls property in object passed to component decorator. But typically use scss files, and there you encapsulate styles by nesting it inside your page’s selector. this is how can style your component, how you can style your page specifically.

Beside that way of styling you’re app and styling individual parts of your app I should say, you also have general theme you can change. You do it in themes folder. there you have variables.scss file. another feature of scss and saas is that you can use variable which are then when bundling your css code are injected into it. Ionic 2 is handling all that. So in this fle you can override some default varibales and you can learn more about this on official docs under theming. Here you can learn which variable you can overwrite and how all this works. The important theming option or most important one is theming here where you override the colors-

$colors: (

primary: #488aff,

secondary: #32db64,

danger: #f53d3d,

light: #f4f4f4,

dark: #222

);

Lets say we change the primary color to red, then all buttons will have red color. this is because by default buttons have primary color. so this is easy way of giving new look to your app.

Ofcorse in real app you will use mixture of all those colors, some buttons have secondary color other have danger color and so on. And tgis is easy way to give your app a completely different look by only changing the couple of here. Again you can learn more about it in docs and in app we are going to build we will use these options.

35)Using utility attribute

The last thing I want to talk about theming and styling is utilities attribute. You saw me using some of them for example we use padding in ion-content

<ion-content padding>

<p>The Users</p>

<button ion-button (click)="onLoadUser('Sumeet')">User 'Sumeet'</button>

<hr>

<button ion-button (click)="onLoadUser('Sood')">User 'Sood'</button>

<hr>

<button ion-button navPop>Go Back</button>

</ion-content>

Ionic 2 ships with lot of such utility directives or attributes. In docs go to-

Theming -> css utilities

We will use them in apps we build. But this is where youcan get complete list.

37)Section Source Code and links

You can find the source code of this section attached to this lecture!

If you got problems running it, you might be using a newer version of the Ionic CLI. Try npm run ionic:serve  after npm install in the project folder in such cases!

Definitely check out the official docs if you want to dive deeper into one of the topics taught in this section: <https://ionicframework.com/docs>

Especially interesting might be the articles about the [NavigationController](https://ionicframework.com/docs/api/navigation/NavController/), [Components](https://ionicframework.com/docs/components/), [Ionicons](https://ionicframework.com/docs/ionicons/), [Theming](https://ionicframework.com/docs/theming/) and the [Utility Attributes](https://ionicframework.com/docs/theming/css-utilities/).