ISEL / ADEETC

Master in Communication and Multimedia Network Engineering

Interactive Multimedia Applications

Lab Work 5

Interactive Multimedia Applications

Ionic App

(Ionic Framework)

Rui Jesus

Introduction

This work aims to introduce the Ionic framework. The tutorial begins with the download and installation of the necessary tools. Next, a mobile application on a theme is developed with emphasis on the most used Ionic techniques for developing mobile user interfaces.

The default theme of the app is Benfica, but each student can choose a different topic to develop the application while maintaining the same structure. That is, **students only can change the multimedia contents**.

The development of the this app covers the fundamentals of Ionic framework in terms of mobile user interface. In this tutorial students will build a mobile app based on 3 tabs for navigation (see Figure 1). First tab, is mainly composed by information about Benfica. It also includes an Ionic icon representing a **Floating Action Button** (FAB) composed by a menu with several hyperlinks to the Ionic documentation. This FAB is the only part that is not related to the topic. Second tab includes a list of **slides** with information about the top players. Users can navigate between slides using **right and left swipe** (see Figure 2). It also includes an **Action Sheet** dialog to allow users have more information about the players. For instance, users can play a video about the player using a **Modal dialog**. Third tab shows a list of videos with the best goals of Benfica. Includes a **Nav** component to play one video (see Figure 3).

The following parts of the mobile user interface of the Ionic framework are used in the application:

(1) **Multi Tab** navigation scheme;
(2) **ion-badges** elements, typically they contain a number or other characters.
(3) **ion-cards,** standard pieces of UI that serves as an entry point to more detailed information;
(4) **Floating Action Buttons** (FABs) represent the primary action in an application;
(5) **ion-icons**, beautifully open source icons, built by the Ionic Framework team;
(6) **ion-action-sheet** dialog that displays a set of options. It appears on top of the app's content;
(7) **ion-slides**, multi-section container. Each section can be swiped or dragged between;

(8) **ion-modal,** dialog that appears on top of the app's content, and must be dismissed by the app before interaction can resume.

(9) **ion-nav,** standalone component for loading arbitrary components and pushing to new components on to the stack.

The final application will look like the following:

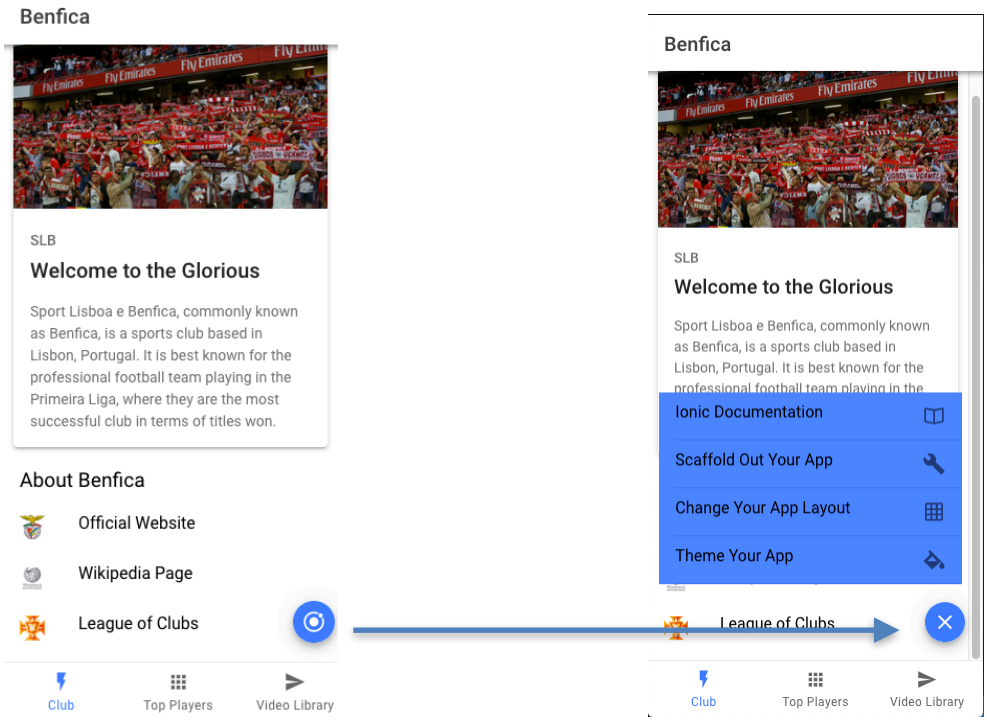


Figure 1. Main Tab (1st Tab) composed by a ion-card and a FAB with a menu of option.

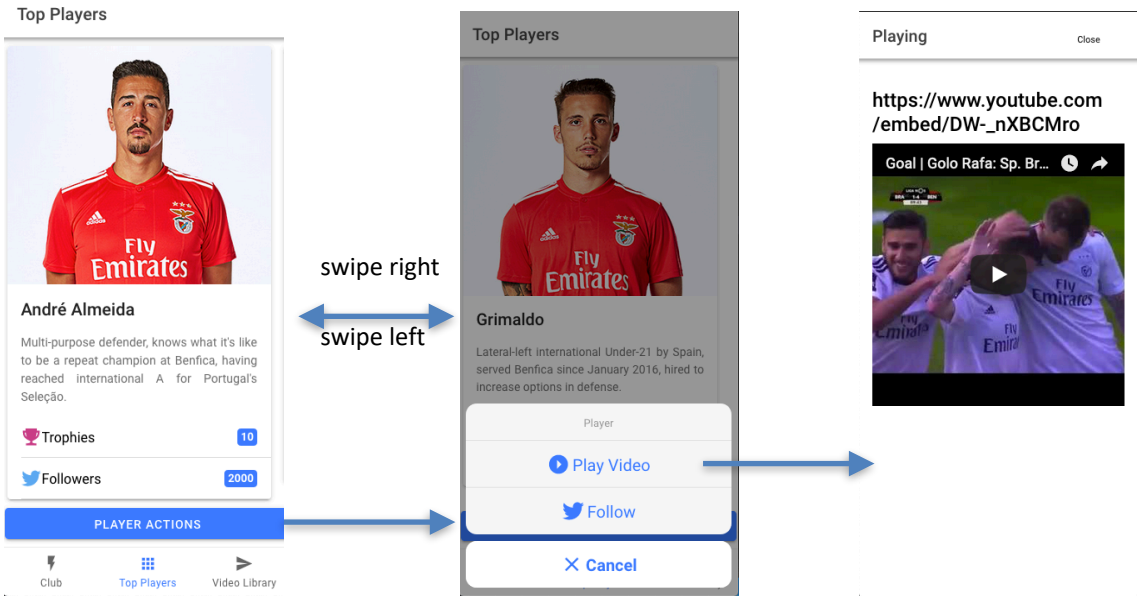


Figure 2. Second Tab with an ion-action-sheet and ion-slides of ion-cards.

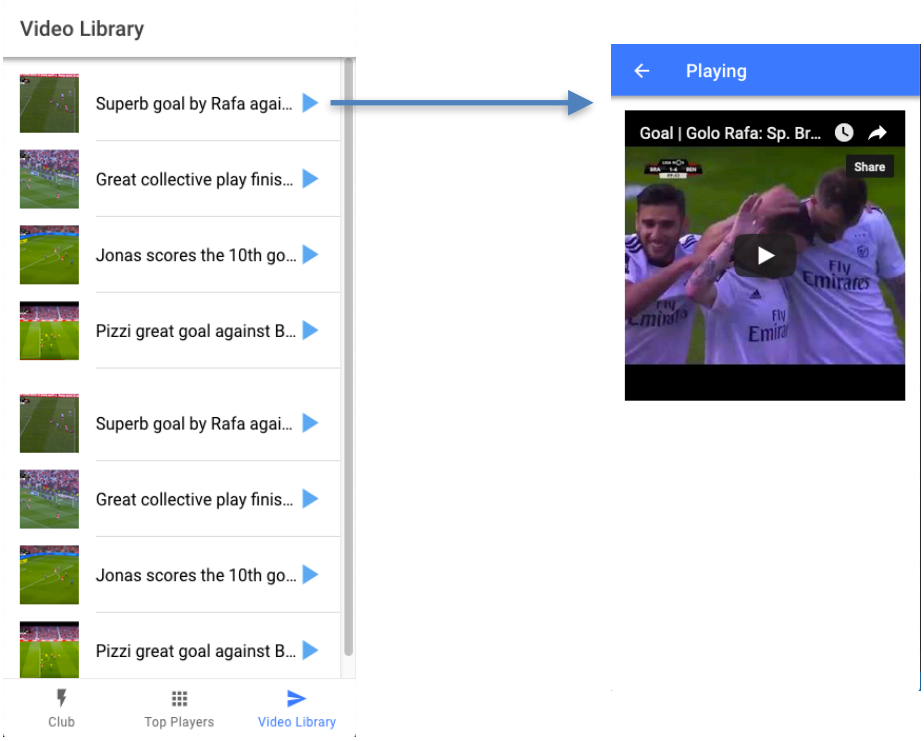


Figure 3. Third Tab with an ion-list an ion-nav component.

Note: this lab work should be done in class and the resulting code must be delivered through the Moodle platform until **May 30th**.

Laboratory Work

Download and installation of the tools

1. Ionic apps are created and developed primarily through the Ionic command-line (CLI) utility. The **Ionic CLI** is the preferred method of installation, as it offers a wide range of **dev** tools. It is also the main tool through which to **run** the app. The following list of tools must be installed:
    a. Node.Js (nodejs.org)
    b. Git (git-scm.com)
    c. Ionic (ionicframework.com)
    d. Apache Cordova (cordova.apache.org)

2. **“Node.js”**, **“npm”** (Node.js package manager) package manager and **“Git”** are already installed. Make sure your development environment includes these tools. To check the versions in a terminal/console window run:

   node -v

   npm -v

   git - -version

3. Although we can install both **“Cordova”** and **“Ionic”** at the same time, it is recommended installing each one individually in case there is an issue during the installation process. Open a terminal/window and enter the following command:

   npm install -g cordova

4. Like the installation of **“Cordova”**, the Ionic CLI is **i**nstalled via **“npm”**:

   npm install -g ionic

5. To check versions of the **“Ionic”** or **“Cordova”** do the same as in point 2 for **“node**” or **“npm”**.

First Ionic app

6. Create an Ionic app using one of the pre-made app templates, or a blank one to start fresh. The three most common starters are the `blank` starter, `tabs` starter, and `sidemenu` starter. Get started with the `ionic start` command::

   ionic start myApp tabs

   Note: If a starter template is not defined, the **Ionic CLI** will present a list of starter templates that can be selected. Choose the `tabs` options, and press enter. It also may ask if you wish to create an **Ionic.io** account. For now, ignore this part.

   Open the **Ionic** Project folder in **Visual Studio Code.** The **Ionic CLI** creates a structure for the starter project, very similar with the one create for an Angular project. The following table summarise the main elements.

src/	Contains the actual application code that we are developing. It includes the app/ and assets/ folders, and the files “index.html”, “main.ts” and “global.scss” for global styling.
node_modules/	Ionic supporting libraries can be found here.
resources/	The default icons and splash screens for both iOS and Android are included in the folder.
platforms/	This folder contains the specific build elements for each installed build platform. This directory will be added once a platform is installed.
plugins/	This directory contains Cordova plug-ins.
.gitignore	Default file generated.
config.xml	File used for Cordova to define various app-specific elements
ionic.config.json	File used by the Ionic CLI to define the various settings when executing commands.
package.json	List of all the npm packages that have been installed to the project
tsconfig.json	File specifies the root files and the compiler options required to compile the project
tslint.json	TypeScript linter rules.

This is the standard structure of any **Ionic** structure of a project. As we add platforms and plugins additional subdirectories and files will be created. The files included in the **app/** and **assets/**, subfolders of the **src/** folder, are the ones we are going to use most of the times.

7. One of the advantages of building hybrid applications is that much of the development and testing can be done locally in the browser. To run and try the starter app in the browser use the line command:

   ionic -o serve

   This command launches a simple web server, watches your files, and rebuilds the app as you make changes to those files, after saving the changes. The --open (or just -o) option automatically opens your browser to http://localhost:8100/. Now, by default, it is used a different port from the port used by Angular Js.

Running the App in a Android Device

8. There are many different options to test native functionality depending on your target platforms and needs:
   a. Implement Platform Detection for the native functionality and test with `ionic serve`.

b. Deploy to **Android**

First option, should be used by professionals that always work for Android devices. Second option generates the **.apk** file and lunches it in the Android Device.

9. To run the app in the mobile, first run the following command line:

   ionic cordova prepare android

   This command copies assets to **Cordova** platforms, preparing them for native builds:

   - Perform an **Ionic build**, which compiles web assets to **www/**.
   - Copy the **www/** directory into your **Cordova** platforms.
   - Transform **config.xml** into platform-specific manifest files.
   - Copy icons and splash screens from **resources/** to into your Cordova platforms.
   - Copy plugin files into specified platforms.

Publishing the App (Cordova)

10. To publish the app, it is needed to generate a release build for Android/iOS platform:

    ionic cordova build --release android

    Before we deploy, we should take care to adjust plugins needed during development that should not be in production mode. For example, we probably don't want the debug console plugin enabled, so we should remove it before generating the release builds:

    ionic cordova plugin rm cordova-plugin-console

    If we are in development stage, and we only want to generate the **.apk** to try the app in the mobile device, **this is not a relevant step, and can be skipped**.

    NOTE: to generate this build for the Android platform, the **Android SDK** need to be installed. Installation of the **Android SDK** can be done by via **Android Studio** or by standalone SDK tools. Go to the Android Developer website and in install it. Installing the Android Studio or SDK will require the installation of the Java Development Kit as well.

11. Next, the unsigned **.apk** file can be found in **platforms/android/app/build/outputs/apk** folder. Now, we need to sign the **unsigned .apk** and run an **alignment utility** on it to optimize it and prepare it for the **app store**. If you already have a **signing key**, skip the next step and use that one instead. If we only want to try the app, the alignment utility can be skipped and in some devices, the **unsigned .apk** can be used.

12. To generate your **private key** using the **keytool** command that comes with the **JDK** use (If this tool is not found, refer to the installation guide):

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

    First, you will be prompted to create a password for the keystore. Then, answer the rest of the tools's questions and when it is 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!

13. 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

14. Finally, we need to run the **zipalign** tool to optimize the **.apk**. The **zipalign** tool can be found in /path/to/Android/sdk/build-tools/VERSION/zipalign. For example, on OS X with Android Studio installed, **zipalign** is in ~/Library/Android/sdk/build-tools/VERSION/zipalign:

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

    Now the final release binary called HelloWorld.apk is created and it can be released on the Google Play Store. To try it in your device, copy the file to the mobile device, install the app and run it.

Publishing the App (Capacitor - recommended)

15. Change the **app id** in the **capacitor.config.json** file. Add the platforms by running the following:

```
ionic build
```

```
ionic cap add android
```

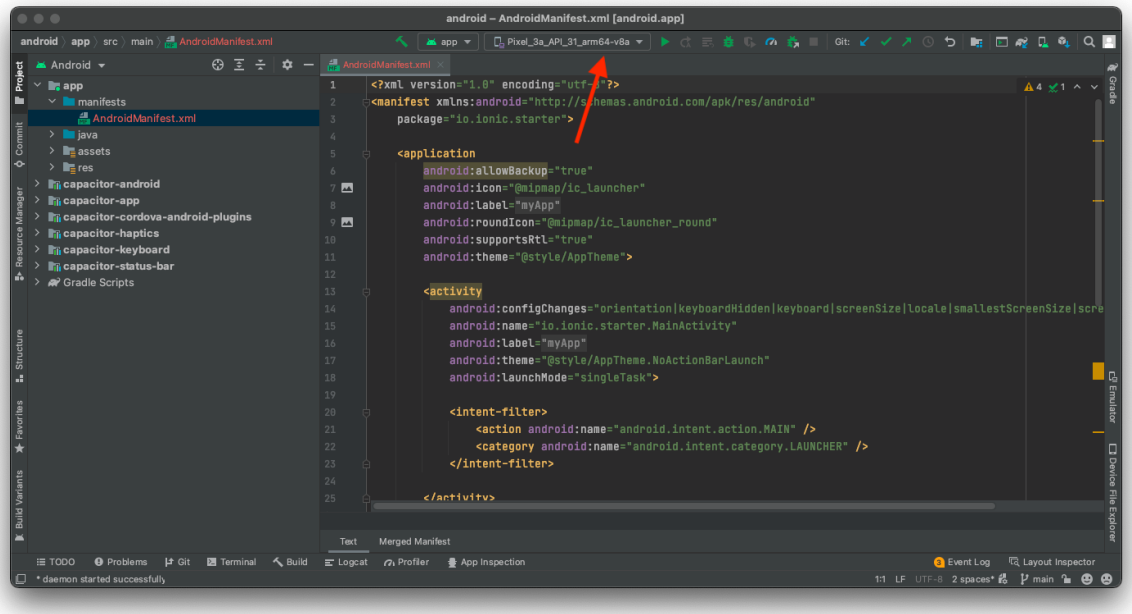
16. Open Android Studio with **npx**:

```
npx cap open android
```

17. Whenever changes are performed in the code, please run the following to build and sync changes:

```
ionic build
npx cap sync
```

18. After open Android Studio (npx cap open android), make sure the device is plugged in, then select it from the list at the top (red arrow in the figure below) and press the run button.



19. To run the code directly in the mobile device, use the following:

```
ionic capacitor run android -l  --external
```

Benfica App (1st Tab)

20. To build the app about Benfica (or other topic), the app generate automatically by the Ionic CLI is going to be changed. Open the file "tab1.page.html" and change the title of the layout (tab) as in Figure 1. Then, open the "tabs.page.html" and change the name of the tabs at the bottom of the app (like in Figure 1).

21. Inside the element <ion-content> of the file "tab1.page.html" change the image and the text of the title, subtitle and content of the <ion-card> element. Follow figure 1.

22. Change the icons, the text and the hyperlinks (keep the previous hyperlink because they are going to be used in the FAB) of the <ion-list> element of the file "tab1.page.html", according to Figure 1.

23. At bottom right of the first tab (file "tab1.page.html") insert a Floating Action Button (FAB) that will open a menu with the ionic links (see second image of Figure 1):

```
<ion-fab vertical="bottom" horizontal="end" slot="fixed">
  <ion-fab-button size="small">
    <ion-icon name="logo-ionics"></ion-icon>
  </ion-fab-button>

  <ion-fab-list side="top">
    <ion-item href="https://ionicframework.com/docs/theming/basics">
      <ion-label>Theme Your App</ion-label>
      <ion-icon name="color-fill"></ion-icon>

    </ion-item>
    <ion-item href="https://ionicframework.com/docs/layout/structure">
      <ion-label>Change Your App Layout</ion-label>
      <ion-icon name="grid"></ion-icon>
    </ion-item>
    <ion-item href="https://ionicframework.com/docs/building/scaffolding">
      <ion-label>Scaffold Out Your App</ion-label>
      <ion-icon name="build"></ion-icon>
    </ion-item>
```

```
      <ion-item href="https://ionicframework.com/docs/">
        <ion-label>Ionic Documentation</ion-label>
        <ion-icon name="book"></ion-icon>
      </ion-items>
    </ion-fab-list>
</ion-fab>
```

24. Change the style in file "tab1.page.scss" as you wish or use the follow rules:

```
.welcome-card ion-img {
      max-height: 35vh;
      overflow: hidden;
}

ion-card {
      margin-bottom: 0;
      padding-bottom: 0;
      border-bottom: 0;
}

ion-label {
      margin-top: 0;
      border: 0;
      padding: 0;
      font-size: 1.2rem;
}

ion-fab {
      ion-fab-list {
      transform: translateX(-96%);
      }
      ion-item {
            width: 131%;
            height: 100%;
            --background: rgb(72, 138, 255);
      }
}
```

Benfica App (2nd Tab)

25. To do the layout of the second tab (as in figure 2), first, create a Player class in its own file in the scr/app folder:

```
export class Player {
      name: string;
      description: string;
      image: string;
      followers:number;
      trophies: number;
      url: string;
}
```

26. To create a "in memory" database of players, create in its own file the PLAYERS array of Player:

```
export const PLAYERS: Player[] = [
{
  name: "Odysseas",
  description: "Arrived in Luz after having represented
Panathinaikos. He was European Under-21 Champion for Germany in 2017.",
  image: "assets/odyseas2.png",
  followers:2600,
  trophies:2,
  url: "https://www.youtube.com/embed/DW-_nXBCMro",
},
{
  name: "André Almeida",
  description: "Multi-purpose defender, knows what it's like to be a
repeat champion at Benfica, having reached international A for Portugal's
Seleção.",
  image: "assets/aalmeida2.png",
  followers:2000,
  trophies:10,
  url: "https://www.youtube.com/embed/DW-_nXBCMro",
},
{
  name: "Grimaldo",
  description: "Lateral-left international Under-21 by Spain, served
Benfica since January 2016, hired to increase options in defense. ",
```

```
      image: "assets/grimaldo2.png",
      followers:3000,
      trophies:8,
      url: "https://www.youtube.com/embed/DW-_nXBCMro",
    }
];
```

Note: the "url" attribute should be assigned with URL of a YouTube video about the player.

27. Insert a <ion-slides> element in the html file inside de <ion-content> element. Follow figure 2:

```
<ion-slides #slides>
      <ion-slide *ngFor="let player of players">
          <ion-card>
              <img [src]="player.image" class="player-image">
              <ion-card-header>
                  <ion-card-title>{{player.name}}</ion-card-title>
              </ion-card-header>
              <ion-cardcontent>{{player.description}}</ion-card-content>
              <ion-item>
                  <ion-icon name='trophy' item-start style="color:
#d03e84"></ion-icon>
                  <ion-label>Trophies</ion-label>
                  <ion-badge item-end>{{player.trophies}}</ion-badge>
              </ion-item>

              <ion-item>
                  <ion-icon name='logo-twitter' item-start style="color:
#55acee"></ion-icon>
                  <ion-label>Followers</ion-label>
                  <ion-badge item-end>{{player.follower}}</ion-badge>
              </ion-item>
          </ion-card>
      </ion-slide>
  </ion-slides>
```

Do not forget to import the PLAYERS array of players in the class "Tab2Page".

28. At the bottom of the page, there is an **Action Sheet dialog** as is shown in the second image of figure 2, first insert the button:

```
<ion-button expand="block" (click)="openMenu1()">
  Player Actions
</ion-button>
```

29. Insert the following code, to transform the button in a Action Sheet dialog and to answer the user click, in the class related to the second tab:

```
  async openMenu() {
 const actionSheet = await this.actionSheetController.create({
   header: 'Player',
   mode: 'ios',
   buttons: [{
     text: 'Play Video',
     icon: 'arrow-dropright-circle',
     handler: () => {
       console.log('Play clicked');
       this.presentModal();
     }
   }, {
     text: 'Follow',
     icon: 'logo-twitter',
     handler: () => {
       console.log('Favorite clicked');
     }
   }, {
     text: 'Cancel',
     icon: 'close',
     role: 'cancel',
     handler: () => {
       console.log('Cancel clicked');
     }
   }]
 });
 await actionSheet.present();
}
```

Run the code.

30. Add the code to play the right video when the user select the option "Play Video" in the Action Sheet Dialog. The right video is the video of the active player in the slides. To know which is the active player add the following code:

```
export class Tab2Page {
      @ViewChild('slides') slides:IonSlides;
      selected_index = 0;
      …

      getInfo() {
            this.slides.getActiveIndex().then(data => {
                  console.log("active index", data);
                  this.selected_index = data;
            });

      }
```

Do not forget to include the id="slides" in the HTML slides. Run the code.

31. To play the video of the player, a **Modal dialog** is used as shown in the third image of figure 2. A new tab is open and the URL of the video is sent to the new page:

```
async presentModal() {
      this.getInfo();
      let url = this.players[this.selected_index].url;
      const modal = await this.modalCtrl.create({
            component: ModalpagePage,
            componentProps: { value: url }
      });
      return await modal.present();
}
```

32. To generate the new page (component) do the following in the command line:

```
ionic generate page modalpage
```

33. Go to the html file of the **modalpage** and include a "close button" and play the video (figure 2):

```
            <ion-header>
            <ion-toolbar>
        <ion-buttons>
            <button ion-button (click)="dismiss()">
                Close
            </button>
        </ion-buttons>
        <ion-title>Playing</ion-title>

    </ion-toolbar>
</ion-header>

<ion-content padding>
    <h2>{{value}}</h2>
      <iframe [src]='sanitizer.bypassSecurityTrustResourceUrl(getValue())'
width="560" height="315" frameborder="0" webkitallowfullscreen
mozallowfullscreen allowfullscreen></iframe>
</ion-content>
```

34. Change the **Modalpagepage** class to receive the URL of the video and close the the page:

```
            export class ModalpagePage implements OnInit {

  @Input() value: string;

  constructor(public modalContr: ModalController, public sanitizer:
DomSanitizer) { }
  getValue():string {
    return this.value;
  }

  dismiss() {
    this.modalContr.dismiss();
  }

  ngOnInit() {
  }

}
```

Do not forget to style (scss file) the second tab and the Modalpagepage.

Benfica App (3rd Tab)

35. Create a "in memory" database of videos with Benfica goals. Do the same as for the players case:

```
import { vid } from './Video';

export const VIDEOS: vid[] = [
     {
        id: 1,
        name: "31st Matchday, 2018/19 Season",
        description: "Superb goal by Rafa against Braga",
        image: "assets/rafa2019.png",
        url:"https://www.youtube.com/embed/DW-_nXBCMro",
      },
      {
        id: 2,
        name: "32nd Matchday, 2018/19 Season",
        description: "Great collective play finished by Rafa",
        image: "assets/rafa2019p.png",
        url:"https://www.youtube.com/embed/H93Dy2Xd7EM",
      },
      {
        id:3,
        name: "21st Matchday, 2018/19 Season",
        description: "Jonas scores the 10th goal of the team against
Nacional",
        image: "assets/jonas2019.png",
        url:"https://www.youtube.com/embed/21dxCrAKdio",
      },
      {
        id:4,
        name: "2015/16 Season",
        description: "Pizzi great goal against Braga with music",
        image: "assets/pizzi2016.png",
        url:"https://www.youtube.com/embed/CgST3EjpG84",
      },
```

```
];
```

Do not forget to import the VIDEOS array of videos in the class "Tab3Page".

36. Add the list of videos to the html file of the 3rd tab:

```
    <ion-content>
  <ion-list>
      <ion-item *ngFor="let item of videos">
          <ion-thumbnail slot="start">
              <ion-img [src]="item.image"></ion-img>
          </ion-thumbnail>
          <ion-label>{{item.description}}</ion-label>
          <ion-icon name='play' item-start style="color:
#55acee" (click)="OpenNavVideoPlay(item.id)"></ion-icon>
      </ion-item>
  </ion-list>
  <ion-list>
```

Run the code.

37. When the user selects one video to play, a Nav component is used. Create this new page:

```
ionic generate page videoplay
```

38. Add the following code to the html file of the videoplay component, to play the video:

```
    <ion-header>
  <ion-toolbar color="primary">
      <ion-buttons slot="start">
          <ion-back-button defaultHref="/tab3"></ion-back-button>
      </ion-buttons>
      <ion-title>Playing</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content padding>
    <iframe [src]='sanitizer.bypassSecurityTrustResourceUrl(getURL())'
width="560" height="315" frameborder="0" webkitallowfullscreen
mozallowfullscreen allowfullscreen></iframe>
</ion-content>
```

Run the code in the browser.

39. In the **3rd tab** we need to lunch the new **videoplay** page:

```
export class Tab3Page {
      videos = VIDEOS;

      constructor(public nav: NavController) {}

      OpenNavVideoPlay(id:number) {
      this.nav.navigateForward("/videoplay/" + id);
      console.log ("teste" + id);
   }
}
```

The "id" of the video to play is passed to the **videoplay** page through the URL.

40. The **VideoplayPage** class has to receive the "id" of the video and translate it in the URL:

```
      export class VideoplayPage implements OnInit {
  id = null;
  vids = VIDEOS;

  constructor(public sanitizer: DomSanitizer, private activatedRoute:
ActivatedRoute) { }

  ngOnInit() {
    this.id = this.activatedRoute.snapshot.paramMap.get('sid');
    console.log ("details "+ this.id);
  }

  getURL(): string {
    let vid = this.vids.filter( v => v.id == this.id);
    console.log (vid[0].url);
    return vid[0].url;
  }

}
```

Run the code.

41. Check the routes of the application in the "app-routing.module.ts" file:

```
    const routes: Routes = [
  { path: '', loadChildren: './tabs/tabs.module#TabsPageModule' },
  { path: 'tab3', loadChildren: './tabs/tabs.module#TabsPageModule' },
  { path: 'videoplay/:sid', loadChildren: './videoplay/
videoplay.module#VideoplayPageModule' },
  { path: 'modalpage', loadChildren: './modalpage/
modalpage.module#ModalpagePageModule' },
];
```

42. Run the code in the mobile device using the ionic **DevApp.**

43. Build the "**apk**" and run it the mobile device.