Product Implementation

Rabbit Music was an app created by Ian Wilkerson to bring users together in finding new music and being able to share their thoughts on that music. This is the first real functioning version of this app. In this file I will be going through my source code and explaining how each piece of major code functions in my project.

Initial html set-up:

Home Page:

<ion-header>

<ion-navbar color = "secondary">

<ion-title>Home</ion-title>

</ion-navbar>

</ion-header>

<ion-content padding class = "body">

<h6 color = "icons">Rabbit Music!</h6>

<p color = "isecond">

Rabbit Music was made for music lovers!

</p>

<ion-card class = "centerCards" color = "secondary">

<ion-card-header class = "topOfCard" color = "icons">

Featured Genres

</ion-card-header>

<ion-slides pager>

<ion-slide>

<ion-card (click)="goToRap(Rap)"> //Function called from the ts file to navigate to the rap page

<img src="assets/img/rap.jpg"/> //Puts a picture as the background of the slide

<div class="card-title">Rap</div> //Title of slide

</ion-card>

</ion-slide>

<ion-slide>

<ion-card (click)="goToPop(Pop)"> //Function called from the ts file to navigate to the pop page

<img src="assets/img/chainsmokers.jpg"/> //Puts a picture as the background of the slide

<div class="card-title">Pop</div> //Title of slide

</ion-card>

</ion-slide>

</ion-slides>

</ion-card>

</ion-content>

CSS files:

I have a general layout for all of my CSS files. The code below is from my home.css file but I use most of the same code for my other page files so that I can make the app have a uniform look to it. It is pretty easy to follow so if you reference the home.css file you should be able to understand the other css files

h6{ //Applies to all h6 headers

text-align: center;

font-size: 45px;

}

p{

text-align: center;

font-size: 20px;

}

.card-title { //Must be called to apply to an item in html

position: absolute;

top: 36%;

font-size: 5.0em;

width: 100%;

font-weight: bold;

color: #fff;

}

.artist-card-title {

position: absolute; //Positions according to the screen

top: 36%;

font-size: 3.0em;

width: 100%;

font-weight: bold;

color: #fff;

}

.topOfCard{

text-align: center;

font-weight: bold;

font-size: 2.0em;

width: 100%;

text-color: #74B3E6;

}

.centerCards {

text-align: center;

shadow: 25px 4px 6px rgba(0, 0, 0, 0.8);

//Adds a shadow around the outside of the card giving it more contrast on the screen

border: 1px #74B3E6;

//Borders around the card. Helps to add contrast from the card and background like the shadow

border-radius: 30px / 30px;

//Rounds out the corners of the cards. I did this on all cards so that the app will have a uniform look

}

Typescript:

I worked in typescript more than any other language in this project. Most all functionality comes from typescript

Imports:

Must import all pages in the app.module.ts file. This is the page that you declare all of the pages that the user will be able to navogate to. Also import all dependanies for FireBase which is the backend database that I used for this project.

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

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

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

import { AboutPage } from '../pages/about/about';

import { ContactPage } from '../pages/contact/contact';

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

import { TabsPage } from '../pages/tabs/tabs';

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

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

import { RapPage } from '../pages/rap/rap';

import { RockPage } from '../pages/rock/rock'

import { PopPage } from '../pages/pop/pop'

//import the Angular fire module

import { AngularFireModule } from 'angularfire2';

import { FIREBASE\_PROVIDERS, defaultFirebase } from 'angularfire2';

Firebase:

This is one of the most important parts of code in my project. The values are from Firebase and link my app to my project in Firebase. Must get the keys from Firebase when you initialize your database. Firebase is a google hosted database service. It stores the data very similar to a JSON tree. There is a parent node and then subtrees branch off from there. In my app the parent node is Rabbit Music or the app itself and then each genre is the first subtree with only one value which is the name. Next level down is where the music objects are stored. All of their values are in the same node. I do plan on putting the comments on level down from here that that each tree will have a leaf for each comment. I have them stored with the rest of the object values but I believe this is why the comments override.

export const firebaseConfig = {

apiKey: "AIzaSyAMJnW0cosE9S\_GdkmptyQdw5CF3l9nn04",

authDomain: "rabbit-music.firebaseapp.com",

databaseURL: "https://rabbit-music.firebaseio.com",

projectId: "rabbit-music",

storageBucket: "rabbit-music.appspot.com",

messagingSenderId: "204994865710"

};

Data Functions:

These are the functions that I used in typescript to work with my data. I will be putting the source code for the pop genre but the source code for other genres is very similar and works the same way

pop: FirebaseListObservable<any>; //Creates a firebase oberservable to be referenced

constructor(public navCtrl: NavController, public navParams: NavParams, public alerCtrl: AlertController, af: AngularFire, public actionSheetCtrl: ActionSheetController) {

this.pop = af.database.list('/pop'); //references to the ‘pop’ subtree of my database

}

The function below adds music to the pop subtree of my database

addMusic(){ //This is the functiont that is called whenever the plus icon is clicked to add another music item

let prompt = this.alerCtrl.create({

//This pulls up an alert controller which is a pop up window that the user will use to input the information about the music item

title: 'Add a piece of music',

message: "Fill out the information below to add a new piece of music",

//Creates fields for the user to input the values that they wish to store in the database

inputs: [

{

name: 'album', //Is the value that will be pushed to Firebase

placeholder: 'Album Name'

},

{

name: 'artist',

placeholder: 'Artist Name'

},

{

name: 'type',

placeholder: 'Album or Single'

},

{

name: 'date',

placeholder: 'Release Date'

},

{

name: 'devRating',

placeholder: 'Dev Rating'

}

],

buttons: [

{

text: 'Cancel', //This will cancel the pop up and bring the user back to the main screen for the genre

handler: data => {

console.log('Cancel clicked');

}

},

{

text: 'Save',

handler: data => {

this.pop.push({ //Pushes an object to Firebase. Each music item is saved as an object with values inside

album: data.album, //Music objects are stored and the furthest branches on the JSON tree

artist: data.artist,

type: data.type,

date: data.date,

devRating: data.devRating

});

}

}

]

});

prompt.present();

}

The next function is the showOptions function. This allows users to see the options they can perform on each music item. This will only be allowed for admins to see. From this pop up menu that will be able to delete or update any value in the music object.

showOptions(album, title, artist, type, date, rating) { //Passes in all the values in the music object

let actionSheet = this.actionSheetCtrl.create({ //Creates an action sheet which is a pop up menu from the bottom

title: 'What do you want to do?',

buttons: [

{

text: 'Delete Music',

role: 'destructive',

handler: () => {

this.removeSong(album); //Calls the remove function and will delete that individual object from the database

}

},{

text: 'Update Music',

handler: () => {

this.updateSong(album, title, artist, type, date, rating); //Will call the update function to update all values

}

},{

text: 'Cancel',

role: 'cancel',

handler: () => {

console.log('Cancel clicked');

}

}

]

});

actionSheet.present();

}

Below is the remove function

removeSong(albumId: string){

this.pop.remove(albumId);

}

The update function is called inside of the show options function and passes all the values including the key for that object in the database. This will allow users to edit any individual piece of the object that they wish

updateSong(albumId, albumTitle, albumArtist, albumType, albumDate, albumRating){

let prompt = this.alerCtrl.create({ //Brings up an alert control just like when users create an object

title: 'Song Name',

message: "Update the name for this song",

inputs: [ //Creates input fields for the user to edit any value that they wish

{

name: 'album',

placeholder: 'Title',

value: albumTitle

},

{

name: 'artist',

placeholder: 'Artist',

value: albumArtist

},

{

name: 'type',

placeholder: 'Type',

value: albumType

},

{

name: 'date',

placeholder: 'Release Date',

value: albumDate

},

{

name: 'rating',

placeholder: 'Dev Rating',

value: albumRating

}

],

buttons: [

{

text: 'Cancel',

handler: data => {

console.log('Cancel clicked');

}

},

{

text: 'Save',

handler: data => {

this.pop.update(albumId, { //will update any information that you changed about the object

title: data.album,

artist: data.artist,

type: data.type,

date: data.date,

rating: data.rating

});

}

}

]

});

prompt.present();

}