

**ANGULAR
IONIC
FIREBASE**

INTRODUCTION

- Created and maintained by Google.
- Open Source.
- Big difference between Angular 1 and Angular 2.
- AngularJS is a framework that pretends to redefine the way we work with JS.
- In JS, you can define views, code when you want. AngularJS defines a separation between views, controllers and data.
- You are bound to follow this structure.

HYBRID APPS

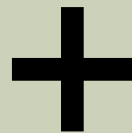
- Hybrid programming
 - Native vs Hybrid vs Web APPS
- One code – Several platforms
- Multiplatform languages (web technologies)
- Different options in the market
 - React native
 - Xamarin
 - jQuery Mobile
 - Ionic
 - ...

WHAT IS IONIC?

- Framework to build multiplatform web applications.
- Based in web technologies (js, html5, css)
- Mixes web code with native code
 - Views and general behaviour → web code
 - Access to hardware → native code
- App is a vitaminized webpage container
- It also generates the package to be installed on the mobile phone.

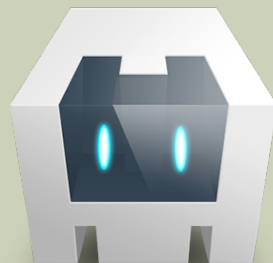
WHAT IS IONIC?

- Cordova + Angular



APACHE CORDOVA

- Framework that allows us to develop apps using web technologies.
- Framework for using mobile HW features using JS apis/plugins.
- This frameworks is extensible with more plugins.



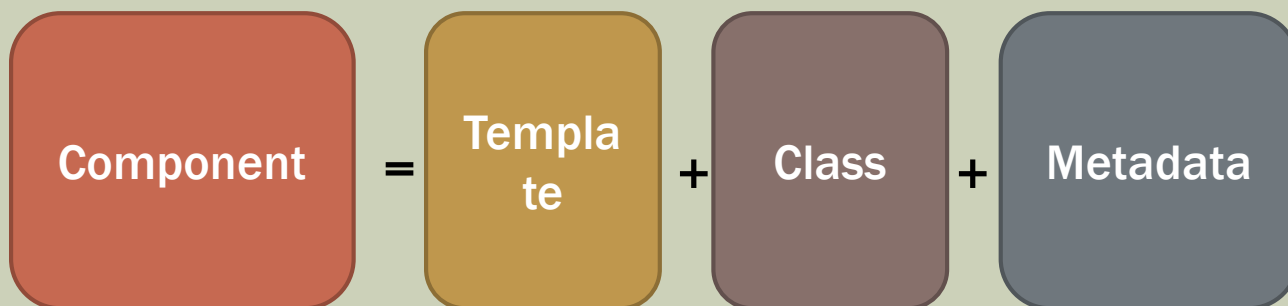
ANGULAR JS

- JS framework developed by Google used for building interactive and dynamic webpages, using only one page.
- This page is updated asynchronously with the user interactions.
- Open source.
- ~~Angular 1~~
- One of the most used nowadays in client side programming.
- It's easy to detach the view from the data and the logic. MVC.

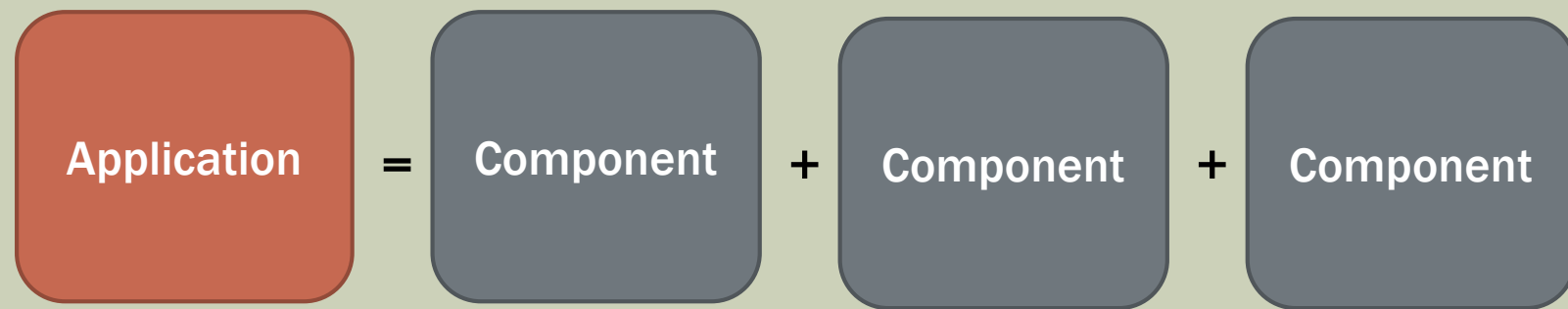


COMPONENTS IN ANGULAR JS

- In angular, all is about components.
- A component can be a part of the UI or the totality. Very similar to a view.
 - Template: HTML + CSS
 - TS/JS class: logic
 - Metadata: app data and config



COMPONENTS IN ANGULAR JS



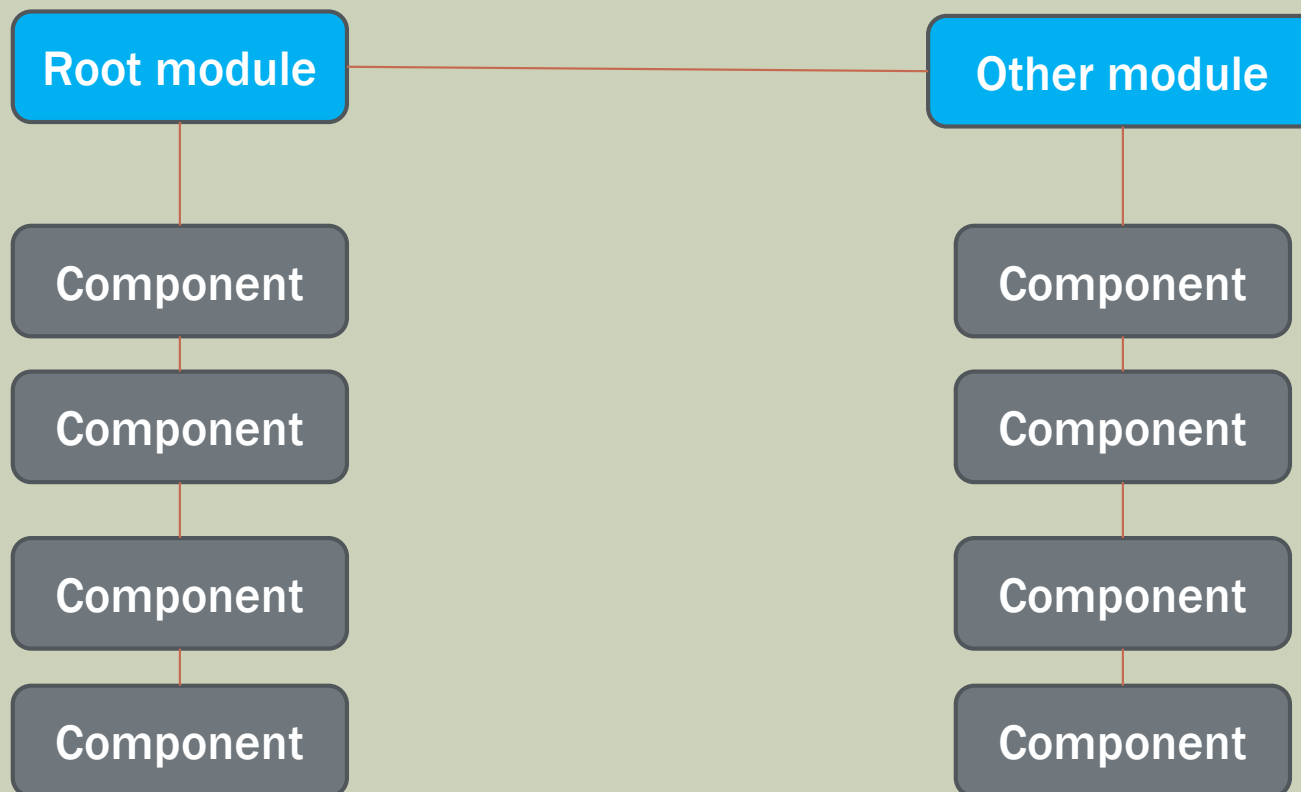
IONIC CLI COMMANDS

- `ionic start projectName blank/tabs/tutorial/super/sidemenu`
- `ionic serve`
- `ionic cordova build android/ios`
- `ionic cordova run android/ios`
- `ionic cordova emulate`
- `ionic generate page/provider/...`
- <https://ionicframework.com/docs/cli/commands.html>
- Testing
 - Ionic DevApp
 - Chrome Debugging

PROJECT STRUCTURE

- **node_modules:** Repository of all the packages used by angular in our app. We don't touch these files directly. We'll do it through node package manager (npm).
- **resources:** Contains the icon and splash for the different platforms. We can generate the different images sizes dynamically.
- **src:** Contains the pages with their code. We will work in this folder.
- **www:** Result of the folder src. We'll never ever modify these files.

ANGULAR APP STRUCTURE



IONIC APP STRUCTURE



TYPESCRIPT CLASS

```
@Component({  
  selector: 'page-home',  
  templateUrl: 'home.html'  
})  
export class HomePage {  
  baseUrlURL : string = '../..../assets/imgs/';  
  animalData: Array<any> = [];  
  animationInProgress: boolean = false;  
  
  constructor(){  
    ...  
  }  
  
  method(param){  
    ...  
  }  
}
```

Class attributes.

TYPESCRIPT CLASS

```
@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {
  baseUrlURL : string = '../..../assets/imgs/';
  animalData: Array<any> = [];
  animationInProgress: boolean = false;

  constructor(){
    ...
  }

  method(param){
    ...
  }
}
```

Constructor for building the component.

TYPESCRIPT CLASS

```
@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {
  baseUrlURL : string = '../../assets/imgs/';
  animalData: Array<any> = [];
  animationInProgress: boolean = false;

  constructor(){
    ...
  }

  method(param){
    ...
  }
}
```

More methods to do some logic or update the class attributes.

HTML TEMPLATE

```
<ion-content>
  <ion-input class="ii_fields" placeholder="Please, input your
name" disabled="{{!enableInputs}}" text-center
[(ngModel)]="userName" type="text">
  </ion-input>
  
  <ion-input class="ii_fields" placeholder="Plase, input your
job position" disabled="{{!enableInputs}}" text-center
[(ngModel)]="jobPosition" type="text">
  </ion-input>
</ion-content>

<ion-footer>
  <button ion-button
(click)="changeButtonString()">{{saveEditButtonString}}</button>
</ion-footer>
```

Marked language. Tags are used to refer the document elements. Ionic has its own tags to do our work easier.

HTML TEMPLATE

```
<ion-content>
  <ion-input class="ii_fields" placeholder="Please, input your
name" disabled="{{!enableInputs}}" text-center
[(ngModel)]="userName">
  </ion-input>
  
  <ion-input class="ii_fields" placeholder="Plase, input your
job position" disabled="{{!enableInputs}}"
[(ngModel)]="jobPosition">
  </ion-input>
</ion-content>

<ion-footer>
  <button ion-button (click)="changeButton">
  </ion-footer>
```

Tag modifier. Ionic-CSS utilities. You can find all in:
<https://ionicframework.com/docs/theming/css-utilities>

HTML TEMPLATE

```
<ion-content>
  <ion-input class="ii_fields" placeholder="Please, input your
name" disabled="{{!enableInputs}}" text-center
[(ngModel)]="userName">
  </ion-input>
  
  <ion-input class="ii_f placeholder="Plase, input your
job position" disabled="{{!enableInputs}}"
[(ngModel)]="jobPosition">
  </ion-input>
</ion-content>

<ion-footer>
  <button ion
(click)="changeBut
</ion-footer>
```

Value interpolation. The value between {{}} is replaced by its value in JS. Link the value in the html with the value in the component class.
<https://docs.angularjs.org/guide/interpolation>

HTML TEMPLATE

```
<ion-content>
```

```
    <ion-input class="ii_fields" placeholder="Please, input your  
name" disabled="{{!enableInputs}}" text-center  
[(ngModel)]="userName">
```

Value interpolation. The value between `{{}}` is replaced by its value in JS. Link the value in the html with the value in the component class.
<https://docs.angularjs.org/guide/interpolation>

```
    </ion-input>  
      
    <ion-input class="ii_fields" placeholder="Please, input your  
job position" disabled="{{!enableInputs}}" text-center  
[(ngModel)]="jobPosition">  
    </ion-input>  
</ion-content>
```

```
<ion-footer>
```

```
    <button ion-button  
(click)="changeButtonString()">{{saveEditButtonString}}</button>  
</ion-footer>
```

HTML TEMPLATE

<ion-content>

```
<ion-input class="ii_fields" placeholder="Please, input your  
name" disabled="{{!enableInputs}}" text-center  
[(ngModel)]="userName">
```

Event linking. Using () and the event name links the event handler over the html element.

<https://developer.mozilla.org/en-US/docs/Web/Events>
<https://developer.mozilla.org/en-US/docs/Web/API/Event>

Event linking. Using () and the event name links the event handler over the html element.

<https://developer.mozilla.org/en-US/docs/Web/Events>
<https://developer.mozilla.org/en-US/docs/Web/API/Event>

job posit
[ngMode

</ion-content>

<ion-footer>

```
<button ion-button  
(click)="changeButtonString()">{{saveEditButtonString}}</button>
```

~~</ion-footer>~~

HTML TEMPLATE

```
<ion-content>
  <ion-input class="ii_fields" placeholder="Please, input your
name" disabled="{{!enableInputs}}" text-center
[(ngModel)]="userName">
  </ion-input>
  <img class="portrait" src='.././../assets/imgs/victor.jpg'/>
  <ion-input class="ii_fields" placeholder="Plase, input your
job position" disabled="{{!enableInputs}}"
[(ngModel)]="jobPosition" text-center>
  </ion-input>
</ion-content>

<ion-footer>
  <button ion-k
(click)="changeButto
</ion-footer>
```

Double linking

When the value changes , the value in the object attribute changes too. Usually used in forms.

DATA BINDING

■ From DOM to Component

- (event)="method()"
- (event)="method(\$event)"
- [(ngModel)]="atrComponent"

■ From Component to DOM

- attribute="{{interpolation}}"
- <tag>{{interpolation}}</tag>
- [attribute]="expresion"
- [(ngModel)]="atrComponent"

TYPES IN ANGULAR

- Angular is based on typescript.
- Typescript has data types, but are not mandatory to use them.
- Some of the most importants are:
 - `string` → `name:string="GTA5";`
 - `boolean` → `best_game:boolean=true;`
 - `number` → `year:number=2018;`
 - `Array` --> `videogames:Array<string> = ["GTA5", "COD","Tekken"];`
 - `any` → `director:any="John Doe"; director=206;`

CLASS VS INTERFACES

```
class Cliente {  
  nombre: String;  
  cif: String;  
  direccion: String;  
  creado: Date;  
}  
  
let cliente: Cliente;
```

```
interface Cliente {  
  nombre: String;  
  cif: String;  
  direccion: String;  
  creado: Date;  
}  
  
let cliente: Cliente;
```

CLASS VS INTERFACES

- Interfaces doesn't need to be initialized
- Doesn't need functionality.
- If you use classes, the "new" keyword has to be used.

```
let cliente1 = new Cliente();
cliente.nombre = 'EscuelaIT S.L.';
cliente.cif = 'B123';
cliente.direccion = 'C/ Del Desarrollo Web .com';
cliente.creado = new Date(Date.now());
```

Class

```
let cliente: Cliente;
cliente = {
  nombre: 'EscuelaIT',
  cif: 'ESB123',
  direccion: 'C/ de arriba, 1',
  creado: new Date(Date.now())
};
```

Interface

INTERFACES

- It's a good practice (MVC) to define the data structure in different files so if we want to define the interface in a file (i.e. clientes.modelo.ts), we would need to use the special keyword `export` to be available from the other files.

```
export interface Cliente {  
  nombre: String;  
  cif: String;  
  direccion: String;  
  creado: Date;  
}
```

- Then from the files when you need to use this interface it has to be imported.

```
import { Cliente } from './cliente.modelo';
```

DIRECTIVES

- Directives are special keyword to be used in our templates (html)
- We can define our own directives and define its behaviour but its not in scope this course.
- We are going to study only the structural directives
 - ngIf
 - ngSwitch
 - ngFor
- Add or remove HTML elements.

NGIF

- Allows us to add or remove elements based on a condition.

```
<h2 *ngIf="true">  
  Cliente rules  
</h2>
```

Variable defined in ts file

- We can use ng-template:

```
<h2 *ngIf="displayName; else elseBlock">  
  Codevolution  
</h2>  
  
<ng-template #elseBlock>  
  <h2>  
    Name is hidden  
  </h2>  
</ng-template>
```

Name we give to the
else block

Our else block in an ng-
template structure

If var displayName is true,
first h2 is shown, else, second
h2 is shown (inside ng-
template h2)

NGIF

- We can separate in different ng-template.

Then

else

```
<div *ngIf="displayName; then thenBlock; else elseBlock"></div>

<ng-template #thenBlock>
  <h2>Codevolution</h2>
</ng-template>

<ng-template #elseBlock>
  <h2>Hidden</h2>
</ng-template>
```

NGSWITCH

```
export class TestComponent implements OnInit {  
  
    public color = "red";  
    constructor() { }  
  
    ngOnInit() {  
    }  
}
```

```
<div [ngSwitch]="color">  
  <div *ngSwitchCase="'red'">You picked red color</div>  
  <div *ngSwitchCase="'blue'">You picked blue color</div>  
  <div *ngSwitchCase="'green'">You picked green color</div>  
  <div *ngSwitchDefault>Pick again</div>  
</div>
```

NGFOR

- Helps us to render a lists of elements.
- Is like a for loop in any language.
- The scope of variable used in the for is only inside the element where its been created.

```
export class TestComponent implements OnInit {  
  public colors = ["red", "blue", "green", "yellow"];
```

```
<div *ngFor="let color of colors">  
  <h2>{{color}}</h2>  
</div>
```


GENERATING NEW PAGES

- Angular provide a page and provider generator.
- This tool generate all the files needed to create a page or provide with the basic structure.
- The command we need to run in the project folder is:
 - *ionic generate page <pagename>*
- We must check if the page has been added to the app.module.ts as a declaration and as an entryComponent.

BASIC NAVIGATION

- To understand this we need to open our app/app-routing.module.ts in which we will find:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', loadChildren: './home/home.module#HomePageModule' },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

BASIC NAVIGATION

- Right now, we have two routes defined inside the array
- The first, is actually a simple redirect that will change the empty path '' to the 'home' path.
- So, we now have a router and are loading a page through a path.
- How is this connected with actual HTML or the *index page of the app*????

BASIC NAVIGATION

- Let's go to index.html

```
<body>  
  <app-root></app-root>  
</body>
```

- The only thing we display is an app-root. This app root is replaced by the first real HTML of our app, which is **always inside the app/app.component.html**

```
<ion-app>  
  <ion-router-outlet></ion-router-outlet>  
</ion-app>
```

BASIC NAVIGATION

```
<ion-app>  
  <ion-router-outlet></ion-router-outlet>  
</ion-app>
```

- The Angular Router will **replace router outlets with the resolved information for a path.**
- This means inside the body, at the top level, we have this special Ionic router outlet wrapped inside a tag for the Ionic app itself. Once we navigate to a certain path, the router will look for a match inside the routes we defined, and display the page inside the outlet.

ADDING NEW PAGES

- In order to add new pages to our app, we need to run the following command in the CLI

```
ionic g page pages/login
```

- After creating pages your **app-routing.module.ts** will automatically be changed.
- Right now, it also contains routing information for new pages we added with the according path of their module

BASIC NAVIGATION

- In order to navigate from one page to another we need to import the Router object (if not already imported) and add it to the constructor (if it's not in the constructor, we can not use it).

```
import { Component } from '@angular/core';  
import { Router, NavigationExtras } from '@angular/router';
```

```
constructor(private router: Router, pri
```

BASIC NAVIGATION

- Navigation without parameters

```
this.router.navigateByUrl('/sign-up');
```

- Back button

```
<ion-header>  
  <ion-toolbar>  
    <ion-buttons slot="start">  
      <ion-back-button [text]=" 'Atrás' " defaultHref="/home"></ion-back-button>  
    </ion-buttons>  
  </ion-toolbar>  
</ion-header>
```


BASIC NAVIGATION

- Navigation With parameters
 - Sending parameters

```
this.router.navigate(['/new-comment', { wc: JSON.stringify(this.wc) }]);
```

- Receiving parameters

```
ngOnInit() {  
  this.route.params.subscribe(params => {  
    this.wc = JSON.parse(params['wc']);  
  });  
}
```

CHANGING OUR ENTRY POINT

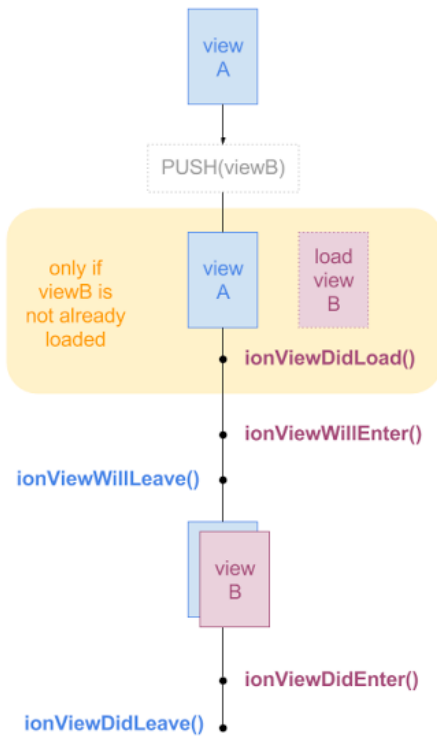
- If we need to change the initial page to be a different component we can simply remove the routing information for the home page (and delete the component if we are not going to use it) and change the redirectTo to point to the page we generated earlier.

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

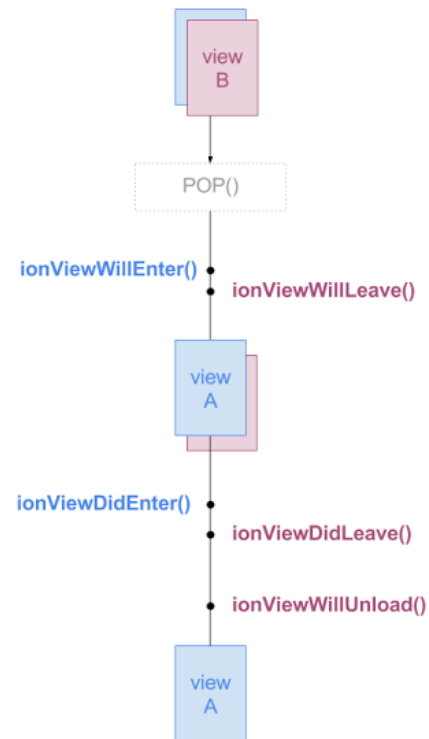
const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  { path: 'login', loadChildren: './pages/login/login.module#LoginPageModule' },
  { path: 'dashboard', loadChildren: './pages/dashboard/dashboard.module#DashboardPageM
```

LIFECYCLE

View lifecycle events on pushing
a new view



View lifecycle events on pop
current view



FIREBASE

- It's time to add more features to our application.



firebase

WHAT'S FIREBASE???

- Firebase is a Backend as a Service (BaaS) that started YC11 startup and grew up into a next-gen-app-development platform on Google Cloud Platform.
- Firebase frees developers, basically client developers as you will be (or not) in a nearly future, to focus only in user experience. You don't need to manage servers. You don't need to write APIs. You don't need to configure databases.
- Firebase is your server, your API, your datastore, your file storage and much more.

WHAT DOES FIREBASE OFFER?

- **Cloud Firestore**

- No relational database (JSON format)

- **Authentication**

- Get all the authentication method in one.

- **Storage**

- Some space to store files, pictures, etc.

- **More...**

- Analytics, Distributed hosting, etc.

AND BESIDES...



WELL...

NOT COMPLETELY FREE




FIREBASE PRICES

Planes de precios de Firebase




Spark

Gratis USD 0 por mes

 Cuotas de uso de Database, Firestore, Storage, Functions, Phone Auth, Hosting y Test Lab

 Capacidad para extender tu proyecto con Google Cloud Platform


 **Se incluyen en todos los planes**
Analytics, notificaciones, informes de fallos, asistencia y más


[Ver los detalles del plan completo](#) 


Plan actual

Flame

Fijo USD 25 por mes

 Aumento del espacio de Database, Firestore, Storage, Phone Auth, Hosting y Test Lab. Conexiones de salida para Functions.

 Capacidad para extender tu proyecto con Google Cloud Platform


 **Se incluyen en todos los planes**
Analytics, notificaciones, informes de fallos, asistencia y más


[Ver los detalles del plan completo](#) 


Seleccionar plan

Blaze

Pago por uso

 Incluye el uso gratuito que se calcula por día. Después, pagas solo por lo que use tu proyecto.

 Capacidad para extender tu proyecto con Google Cloud Platform

 **Se incluyen en todos los planes**
Analytics, notificaciones, informes de fallos, asistencia y más

[Ver los detalles del plan completo](#) 

Seleccionar plan

CONFIGURING FIREBASE

- Create a new firebase project with the free plan.
- Install AngularFire and firebase
 - <https://github.com/angular/angularfire>

```
npm install @angular/fire firebase --save
```

CONFIGURING FIREBASE

- Add Firebase config to environment variable
 - Open environments/environment.ts file and add your Firebase project configuration. You can find this information in the firebase project configuration in the firebase console.

```
export const environment = {  
  production: false,  
  firebase: {  
    apiKey: '<your-key>',  
    authDomain: '<your-project-authdomain>',  
    databaseURL: '<your-database-URL>',  
    projectId: '<your-project-id>',  
    storageBucket: '<your-storage-bucket>',  
    messagingSenderId: '<your-messaging-sender-id>'  
  }  
};
```

CONFIGURING FIREBASE

- Inject the Firebase providers and specify your firebase configuration.
 - Open /src/apps/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { AngularFireModule } from '@angular/fire';
import { environment } from '../environments/environment';

@NgModule({
  imports: [
    BrowserModule,
    AngularFireModule.initializeApp(environment.firebase)
  ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

CONFIGURING FIREBASE

- Adding Firebase features we need to use.
 - For example, if we need Firebase authentication, Firebase database, and Firebase Storage we would add:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { AngularFireModule } from '@angular/fire';
import { AngularFireStoreModule } from '@angular/fire/firestore';
import { AngularFireStorageModule } from '@angular/fire/storage';
import { AngularFireAuthModule } from '@angular/fire/auth';
import { environment } from '../environments/environment';

@NgModule({
  imports: [
    BrowserModule,
    AngularFireModule.initializeApp(environment.firebase, 'my-app-name'), // imports firebase/app
    AngularFireStoreModule, // imports firebase/firestore, only needed for database features
    AngularFireAuthModule, // imports firebase/auth, only needed for auth features,
    AngularFireStorageModule // imports firebase/storage only needed for storage features
  ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

CONFIGURING FIREBASE

- Now, you can use Firebase in the components you need importing the class and working on them following the instructions.

```
import { Component } from '@angular/core';
import { AngularFireStore } from '@angular/fire/firestore';

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
  styleUrls: ['app.component.css']
})
export class AppComponent {
  constructor(db: AngularFireStore) {

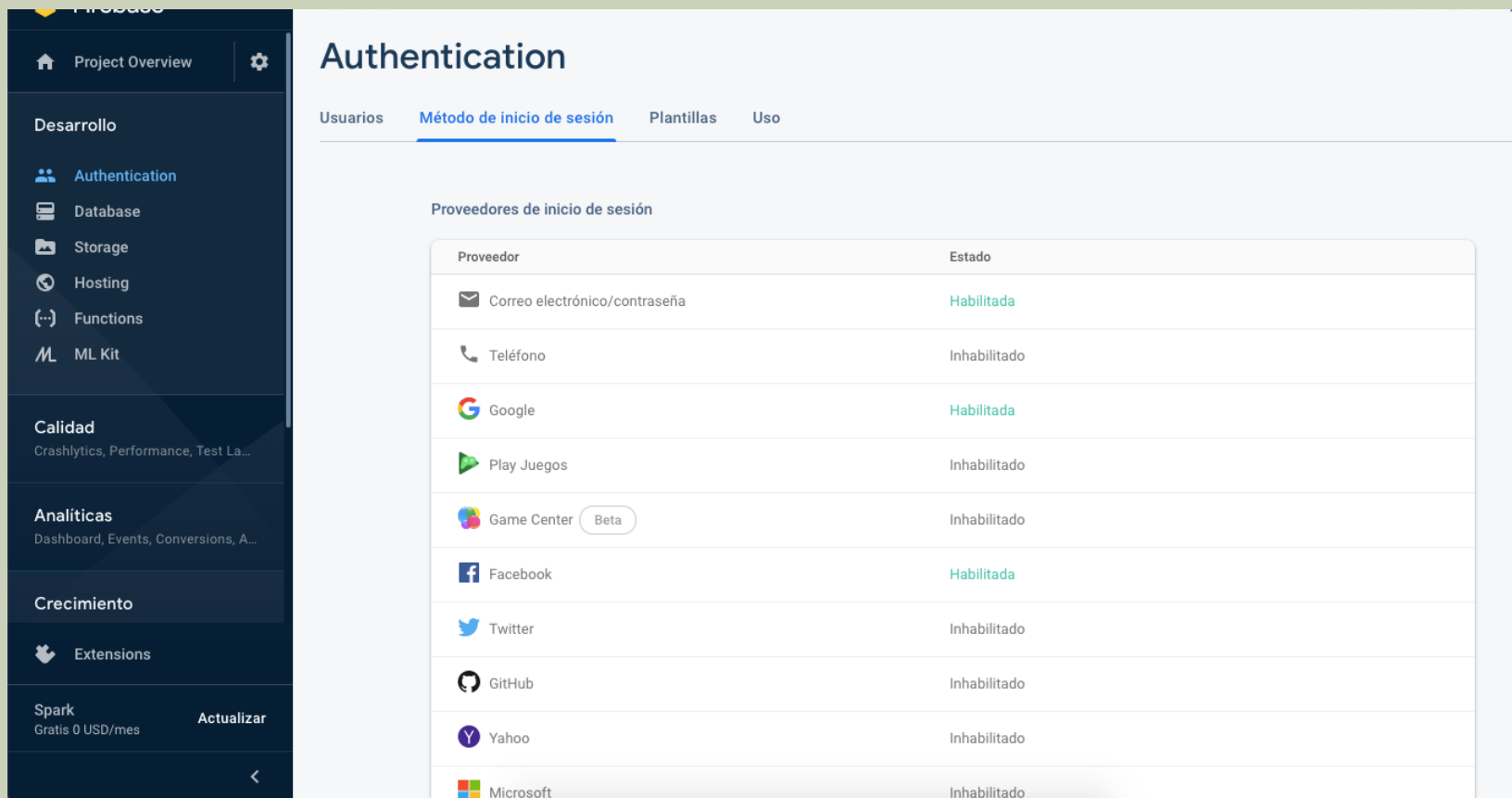
  }
}
```

FIREBASE AUTHENTICATION

- One of the features Firebase offers is a service to authenticate users through different platforms (user-password, facebook, google, etc).
- The developer is totally abstracted of this server-side development and only has to call the proper function provided by the library.
- Note that all these methods are asynchronous (like Ajax)

FIREBASE AUTHENTICATION

- First of all, we need to enable this feature in the Firebase Console. Let's activate authentication via user-pass. As you can see, there are many other authentication.



The screenshot displays the Firebase Authentication console interface. The left sidebar contains a navigation menu with the following sections:

- Desarrollo**
 - Authentication
 - Database
 - Storage
 - Hosting
 - Functions
 - ML Kit
- Calidad**
 - Crashlytics, Performance, Test La...
- Analíticas**
 - Dashboard, Events, Conversions, A...
- Crecimiento**
 - Extensions
- Spark**
 - Gratis 0 USD/mes
 - Actualizar

The main content area is titled 'Authentication' and has four tabs: 'Usuarios', 'Método de inicio de sesión' (selected), 'Plantillas', and 'Uso'. Below the tabs, the section 'Proveedores de inicio de sesión' contains a table of authentication providers.

Proveedor	Estado
Correo electrónico/contraseña	Habilitada
Teléfono	Inhabilitado
Google	Habilitada
Play Juegos	Inhabilitado
Game Center Beta	Inhabilitado
Facebook	Habilitada
Twitter	Inhabilitado
GitHub	Inhabilitado
Yahoo	Inhabilitado
Microsoft	Inhabilitado

FIREBASE AUTHENTICATION

- In the page where we want to authenticate user (or register) we need to inject **AngularFireAuth** into the component's constructor and in the app.module.
- Next use the injected instance (let's call it *afAuth*) to call
 - *this.afAuth.auth.signInWithEmailAndPassword(user.email, user.password)* for authenticating a user with email and password
 - *this.afAuth.auth.createUserWithEmailAndPassword(user.email, user.password)* for registering users.
 - *this.afAuth.auth.sendPasswordResetEmail(email)* for recover password.

FIREBASE AUTHENTICATION

- All these functions return a promise.
- All these functions must be isolated in a service (let's call it `authService`).
- Component has to response to the user event and call the service with the data.
- Service has to call firebase and return the response to the component.

FIREBASE AUTHENTICATION

- ts of the component (create user)

```
async onClickSaveButton(){
  this.newUser.mail = this.mailText;
  this.newUser.password = this.passwordText;

  this.authService.signupUser(this.newUser)
    .then(() => {
      this.toastService.createToast('Usuario creado correctamente. Bienvenido.', false);
      this.router.navigateByUrl('home');
    }, error => {
      console.log(JSON.stringify(error));
      this.toastService.createAlertOK("Error al crear el usuario. Por favor inténtelo de nuevo más tarde");
    });
}
```

FIREBASE AUTHENTICATION

- Service (create user function)

```
async signupUser(user:User): Promise<any> {  
  return this.afAuth  
    .auth.createUserWithEmailAndPassword(user.mail, user.password)  
    .then((newUserCredential: firebase.auth.UserCredential) => {  
      user.id = newUserCredential.user.uid;  
      user.password = "";  
      this.db.doc(`/users/${user.id}`).set({ user });  
    })  
    .catch(error => {  
      console.error(error);  
      throw new Error(error);  
    });  
}
```

FIREBASE AUTHENTICATION

■ ts of the component (login user)

```
async onClickLoginButton(){
  if(this.isFormValidated()){
    const loading = await this.loadingCtrl.create({
      message: 'Iniciando sesión...'
    });
    await loading.present();

    this.authService.loginUser(this.mailText, this.passwordText)
      .then(() => {
        this.toastService.createToast('Login correcto. Bienvenido', false);
        this.router.navigateByUrl('home');
        this.showErrorMessage=false;
      },error => {
        if(error.message.includes("user")){
          this.errorMessage="El usuario no existe";
          this.showErrorMessage=true;
        }

        if(error.message.includes("password")){
          this.errorMessage="Contraseña incorrecta";
          this.showErrorMessage=true;
        }
      });
  }
}
```

FIREBASE AUTHENTICATION

- Service (login user function)

```
loginUser(email: string, password: string):Promise<firebase.auth.UserCredential> {  
    return this.afAuth.auth.signInWithEmailAndPassword(email, password);  
}
```

FIRESTORE

- No relational DATABASE
- JSON Format. Collections and documents.
- As a general rule the database only will be accessed through a service.
- As in the authentication, we need to add in the constructor of the class (service) the reference to the FirestoreModule and in the app.module

```
constructor(  
  private db:AngularFirestore,  
  private afAuth: AngularFireAuth  
) {
```

FIRESTORE

■ CREATE

- Basic and easy operation. We need to refer the object we created in the constructor, set the path when we want to create the document in our database and pass the object we want to store.
- If the path doesn't exist, Firestore will create it on your behalf.
- This code stores an object named “user” with all the attributes of the object user

```
this.db.doc(`/users/${user.id}`).set({ user });
```


FIRESTORE

■ GET

- When we get data from firestore we are retrieving an especial object named Observable which is continuously waiting for changes and if there is any, it will be shown in our app without doing anything special.
- Each element is a document of the collection.

```
let usersCollection:AngularFirestoreCollection = this.db.collection<User>('users');
usersCollection.valueChanges().subscribe(
  res=>{
    res.forEach(element => {
      if(element.user.mail == email){
        sessionStorage.setItem('userobject', JSON.stringify(element.user));
      }
    });
  }
)
```

FIRESTORE

■ UPDATE

- To update an object, we only need the key of the document being updated. Then, we call the function `update` passing the data we need to update in JSON format.

```
this.db.doc(`users/${users.id}`).update({email:new_email});
```

FIRESTORE

■ DELETE

- To delete a document we only need to refer the id of the document.

```
deleteSong(songId: string): Promise<void> {  
    return this.firestore.doc(`songList/${songId}`).delete();  
}
```

FIREBASE STORAGE

- Firebase offers a function to store images (or files) in its server.
- It's free but when you reach some size, you need to pay.
- Images must be saved in BASE 64
- Each image (or file) is identified by an ID. This id must be created by ourself. We can use the same method than in the db to get an ID.
- In order to use the storage, we need to import it the `AngularFirestore` in the constructor and in the `app.module`(as we did with other services).

FIREBASE STORAGE

- To upload an image from camera (or from gallery) we need the image data returned by them.

```
this.camera.getPicture(options).then((imageData) => {  
  let image64='data:image/jpeg;base64,'+imageData;
```

```
this.imagePicker.getPictures(optionsImagePicker).then((results) => {  
  for (var i = 0; i < results.length; i++) {  
    let image64='data:image/jpeg;base64,'+results[i];
```

FIREBASE STORAGE

- To upload a new image to storage we need.
 - A path to the image we need to store.

```
const loading = await this.loadingCtrl.create({  
  message: 'Guardando foto...'  
});  
await loading.present();  
  
let photoName = `wc/${pictureObject.id_wc}/${pictureObject.id}`;
```

FIREBASE STORAGE

- A reference to the path.

```
const loading = await this.loadingCtrl.create({  
  message: 'Guardando foto...'  
});  
await loading.present();  
  
let photoName = `wc/${pictureObject.id_wc}/${pictureObject.id}`;  
const fileRef = this.storage.ref(photoName);
```

FIREBASE STORAGE

- Upload the image.

```
const loading = await this.loadingCtrl.create({  
  message: 'Guardando foto...'  
});  
await loading.present();  
  
let photoName = `wc/${pictureObject.id_wc}/${pictureObject.id}`;  
const fileRef = this.storage.ref(photoName);  
const task = fileRef.putString(imageData, 'data_url');
```


FIREBASE STORAGE

- Wait the image to finalize the upload. Then we got our URL referring to the image and it can be used to link it in the src of an img in our app.

```
const loading = await this.loadingCtrl.create({
  message: 'Guardando foto...'
});
await loading.present();

let photoName = `wc/${pictureObject.id_wc}/${pictureObject.id}`;
const fileRef = this.storage.ref(photoName);
const task = fileRef.putString(imageData, 'data_url');

task.snapshotChanges().pipe(
  finalize(() => {
    fileRef.getDownloadURL().subscribe(url => {
      pictureObject.url=url;
      loading.dismiss();
    });
  })
).subscribe();
```

RUNNING ON A DEVICE

- In order to have our application running on a real devices, either iOS or Android we need to first add the platform to our project and then compile it.
 - ionic cordova platform add android (or ios)
 - ionic cordova build android (or ios)
- With these instructions, a platforms folder will be created with the Android (or iOS) project inside them.

RUNNING ON A DEVICE

- From Android Studio (or Xcode) we can open the project created in the platforms folder and run it into the emulator or a real device.
- Sometimes, it is necessary to remove and add again the platform to get the app working.
 - `ionic cordova remove android` (or `ios`)
- Note that some plugins work on a real device and you are forced to test them there.

NATIVE PLUGINS

- We have a full list of native plugins ready to be implemented in our application.
- Native plugins use Cordova for accessing to the native functionalities of the mobile phone.
- Each native plugin has to be installed in each project you want to use it.
- Each native plugin has a different way to be configured.
- Read documentation, read examples and good luck.