

# Sarrera

Mobilentzako/tabletentzako aplikazioak egiterakoan, **bi aukera** ditugu:

- **Web app** (nabegadore batekin zabaltzen dena)
- **Mobile app** (mobilentzako aplikazioa app denda batetik deskargatzen dena)  
Honen barruan, bi mota topatu daiteke:
  - **Native app** (plataforma edo dispositibo zehatz batentzat kodifikatuta dagoena)
  - **Hybrid app** (bertako app modura instalatzen da, baina web app-a da barrutik, webview nabegadorean bistaratzen dena. Kodigo berdinarekin eta aldaketa gutxirekin mundu biak batzen ditu)

[Ionic](#) aplikazio hibridoak garatzeko framework-a da, hasiera batean telefono mugikorretarako eta tabletetarako diseinaturik zena, nahiz eta, orain, web aplikazioak inplementatzeko gai den ere..

Azpian Angular edo React erabiltzen du eta osagai ugari ditu erabiltzailearen esperientzia hobetzeko. Gainera, osagaiak aplikazioen garapena asko errazten dute.

HTML5 framework bat da eta horrela funtzionatzen du aplikazioaren muina, baina ezaugarri batzuk (kamara, ....) erabiltzeko eta bertako aplikazio modura exekutatzeko (native app), Cordova edo Capacitor bezalako bertako bilgarria (native wrapper) erabiltzen du.

## Ionic Framework

Ionic Framework is an open source UI toolkit for building performant, high-quality mobile and desktop apps using web technologies — HTML, CSS, and JavaScript — with integrations for popular frameworks like [Angular](#), [React](#), and [Vue](#).

# 1- Hasiera

## Behar dugun softwarea

### Ezkatatu

- **Node.js** eta **npm** erabiliko ditugu (Instalatzeko: <https://nodejs.org/es>)

> **node -v**

> **npm -v**

- Angular instalatuta behar dugu (aurretik eginda daukagu)

> **npm install -g @angular/cli@latest**

> **ng version**

- **Ionic** instalatu behar da kotsola batetik administradore bezala konektatuta

> **npm install -g @ionic/cli**

> **ionic -v**

- Ionic buruz dokumentazioa

<https://ionicframework.com/docs>

# Proiektu berria

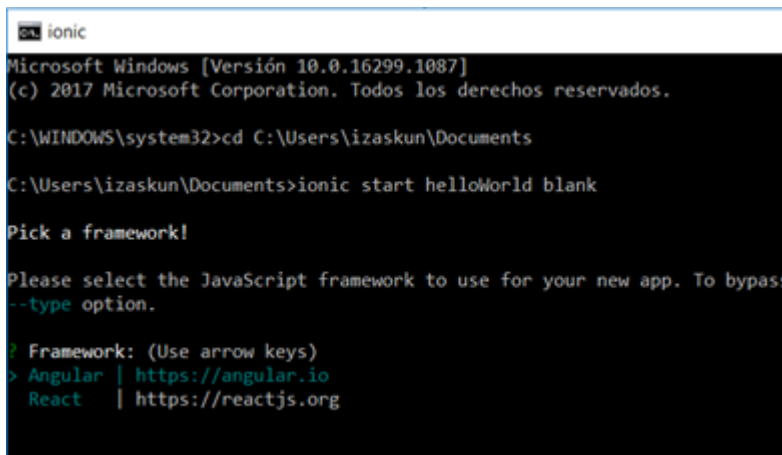
## Ezkatatu

Proiektu berria sortzeko existitzen den txantiloia bat erabili dezakegu: adibidez, **blank** txantiloia

```
> cd C:\Users\izaskun\Documents
```

```
> ionic start helloWorld blank
```

Framework bat aukeratzeko eskatuko digu: Angular



```
ionic
Microsoft Windows [Versión 10.0.16299.1087]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>cd C:\Users\izaskun\Documents

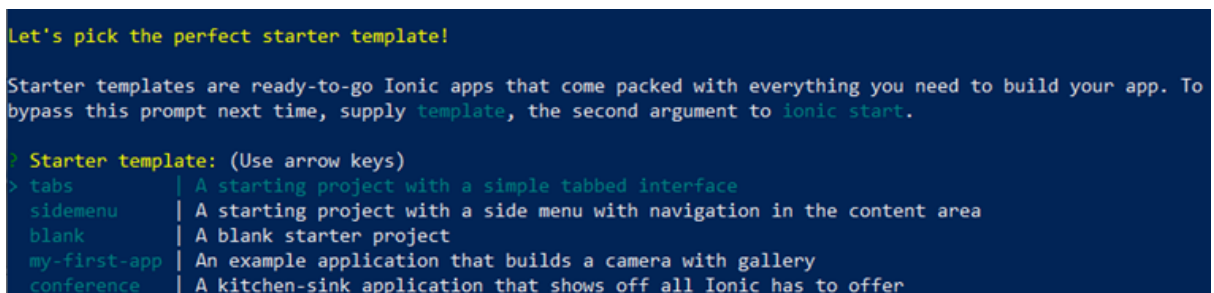
C:\Users\izaskun\Documents>ionic start helloWorld blank

Pick a framework!

Please select the JavaScript framework to use for your new app. To bypass
--type option.

> Framework: (Use arrow keys)
> Angular | https://angular.io
  React   | https://reactjs.org
```

Txantiloia ez badugu jartzen hasieratik, bat aukeratzeko eskatuko digu:



```
Let's pick the perfect starter template!

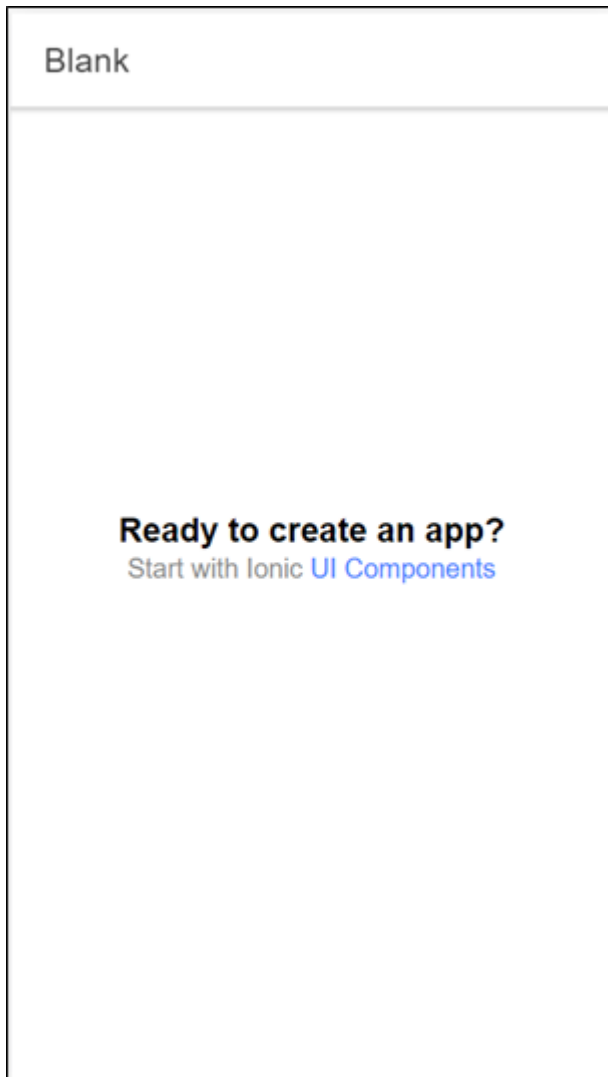
Starter templates are ready-to-go Ionic apps that come packed with everything you need to build your app. To
bypass this prompt next time, supply template, the second argument to ionic start.

> Starter template: (Use arrow keys)
> tabs | A starting project with a simple tabbed interface
  sidemenu | A starting project with a side menu with navigation in the content area
  blank | A blank starter project
  my-first-app | An example application that builds a camera with gallery
  conference | A kitchen-sink application that shows off all Ionic has to offer
```

Proiektua martxan jartzeko:

```
> cd helloWorld
```

```
> ionic serve
```



Aldaketak batzuk egingo ditugu "home" orrian: **src/app/home/home.page.html**

```
<ion-title> Hello World !! </ion-title>
```

```
.....
```

```
<div id="container">  
  <div class="ion-padding">  
    <p>Hello {{name}}</p>  
  </div>  
  <strong>Ready to create an app?</strong>
```

```
.....
```

```
</div>
```

Eta **src/app/home/home.page.ts**

```
export class HomePage {
```

```
  public name;
```

```
  constructor(){
```

```
    this.name = "Izaskun";
```

```
  }
```

```
}
```

# Oinarrizko gauzak

## [Ezkatatu](#)

**Ionic-eko proiektu bat beste leku batera eramateko**, dagoen bezala kopiatzea ez da komeni, **node-modules** karpeta handia delako. Egin behar dena da:

- Kopiatu karpeta osoa **node-modules** barik,
- Beste nonbaiten itsatsi
- Terminaletik **cd** agindua erabiliz, karpeta berri horretan sartu
- Agindu hau exekutatu: **npm install**

**Proiektu bateko dependentziak eskuz aldatzeko edo berriro birsortzeko, package.json** fitxategia eguneratu behar da. Hau egiten bada eskuz, dena berriro birsortzea komeni zaigu eta horretarako:

- Terminaletik agindu hauek exekutatu:
  - **Remove-Item -Recurse -Force .\node\_modules\**
  - **Remove-Item -Force .\package-lock.json**
  - **npm install**

## 2- Txantiloak eta Osagaiak

Txantiloak batzuk daude. Bat aukeratu ahal dugu eta gero horren gainean gure aldaketa propioak egin:

<https://ionicframework.com/docs/v3/cli/starters.html>

- **Terminalean aukerak ikusteko:**

> **ionic start --list**

```
E:\Aprobak_DAW\Ionic-ariketak\IonicStrava>ionic start --list

Starters for @ionic/angular (--type=angular)

name          | description
-----
tabs          | A starting project with a simple tabbed interface
sidemenu      | A starting project with a side menu with navigation in the content area
blank         | A blank starter project
my-first-app  | An example application that builds a camera with gallery
conference    | A kitchen-sink application that shows off all Ionic has to offer

Starters for @ionic/react (--type=react)

name          | description
-----
blank         | A blank starter project
sidemenu      | A starting project with a side menu with navigation in the content area
tabs          | A starting project with a simple tabbed interface
```

- **Proiektu berria sortu txantiloia bat erabiliz: tabs**

> **cd C:\Users\lizaskun\Documents**

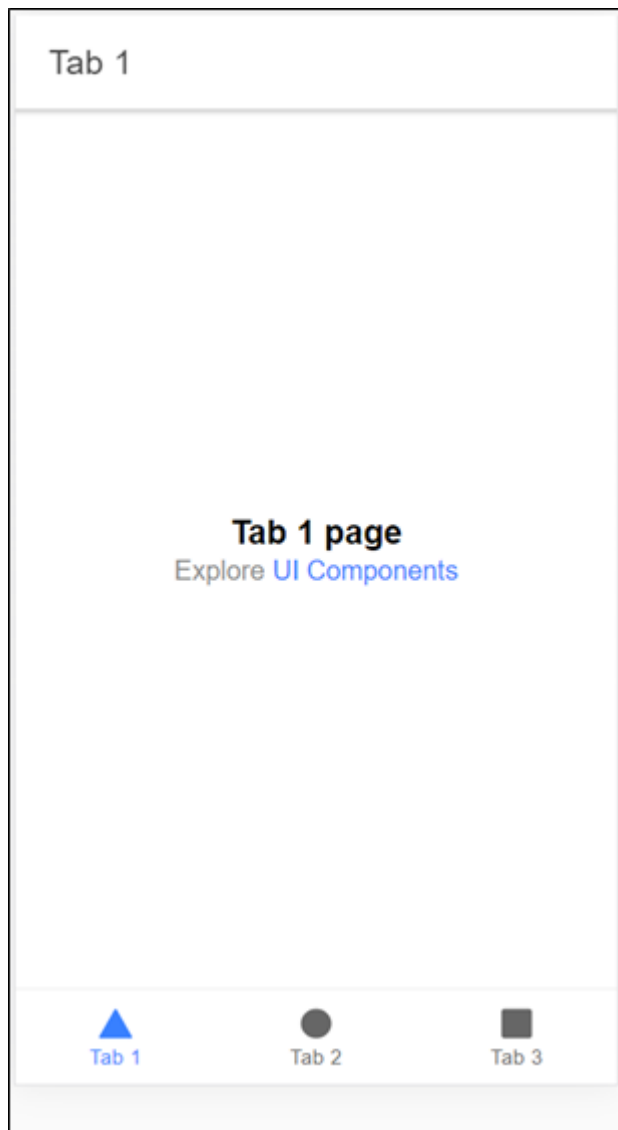
> **ionic start HelloTabs tabs**

- **Proiektua martxan jarri eta ikusi**

> **cd HelloTabs**

> **ionic serve -l**

(l -> --lab = plataforma denetan probatzeko)



- Aldaketak batzuk egingo ditugu "tab2" orrian: `src/app/tab2/tab2.page.html`

`<ion-header>`

`<ion-toolbar>`

`<ion-title>`

Argazki Galeria



</ion-title>

</ion-toolbar>

</ion-header>

- Aldaketak "tabs" orrian: src/app/tabs/tabs.page.html

<ion-tab-button tab="tab2">

<ion-icon name="irudiak"></ion-icon>

<ion-label>Galeria</ion-label>

</ion-tab-button>

- Aldaketak batzuk "tab3" orrian: src/app/tab3/tab3.page.html

<ion-badge> osagaia erabiliko dugu:

<https://ionicframework.com/docs/api/badge>

- Berdina egingo dugu beste pestaina berri bat eginda eta beste osagai bat erabiliz (adibidez <ion-card>)

<https://ionicframework.com/docs/api/card>

> ionic g page tab4

- Aldaketak tabs-routing-module.ts tab4 orrira nabigatzeko
- Aldaketak tabs.page.html prestatzeko
- Koloreak pertsonalizatu ahal dira, horretarako theme/variables.scss aldatzen da:

<https://ionicframework.com/docs/theming/colors>

Edozein etiketan color atributua jartzen bada, kolorea aldatzen da:

```
<ion-title color="danger">tab4</ion-title>
```

9 color ezberdinak: **primary, secondary, tertiary, success, warning, danger, light, medium, dark**

- Osagai oso garrantzitsua: **<ion-content>**

<https://ionicframework.com/docs/api/content#usage>

- **Native plugins**

<https://ionicframework.com/docs/native>

## 3- IonicStrava - REST API

### [Ezkatatu](#)

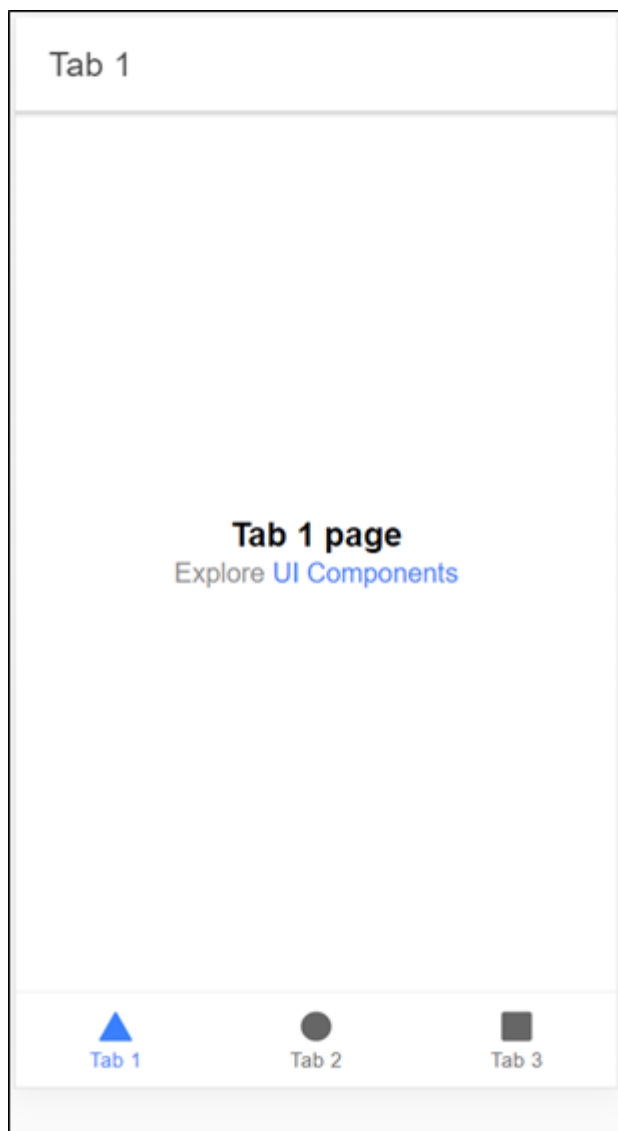
Strava atletismoarekin lotutako aplikazio bat da. Norberak egiten dituen jarduera fisikoak erregistratzeko erabiltzen dena eta gainera zure jarduera fisikoei edo parte hartzen duzun klubetako kideei buruzko datuak kontsultatzeko web orri bat du.

<https://www.strava.com>

Strava antzeko web orria egingo dugu eta klubaren informazioa erakusten saiatuko gara.

Lehenengo pausua txantiloia aukeratzea izango da: tabs aukeratuko dugu

> ionic start IonicStrava tabs




Lehenengo fitxan klub-en informazioa jarriko dugu, bigarren-ean norberaren informazioa eta hirugarren-ean Ebentoak.

# Klubaren informazioa bistaratu

[Ezkutatu](#)

Hasteko klub bakarra erakutsiko dugu:


uni eibar


company 

Lanbide Heziketako Ikastetxea

Kirol mota: **running**

Partehartzaileak: **3**

 Detaileak ikusi



RUNNING

Klub baten datuak (momentuz ez ditugu datu guztiak erabiliko):

```
kluba = {  
  id: 783189,  
  name: 'uni eibar',  
  cover_photo_small: 'assets/img/kluba.jpg',  
  sport_type: 'running',  
  privatea: true,  
  member_count: 3,  
  description: 'Lanbide Heziketako Ikastetxea',  
  club_type: 'company',  
};
```

## Egin beharrekoak:

- Lehenengo fitxan ipiniko dugu: Klubak.
- Izena erakutsi ("name")
- Klub mota (club\_type, berdez ikusten dena). Pribatua den ala ez ("privatea" true edo false; fa-lock, fa-lock-open)
- Deskripzioa erakutsi ("description")
- Kirol mota ("sport\_type")
- Partehartzaileak jartzen duena badge bat izango da ("member\_count"), baina koloreak aldatuta: ez badago partehartzailearik, gorriz agertzen da, 0 eta 10 partehartzaile badaude, horiz eta gehiago badago, urdinez.

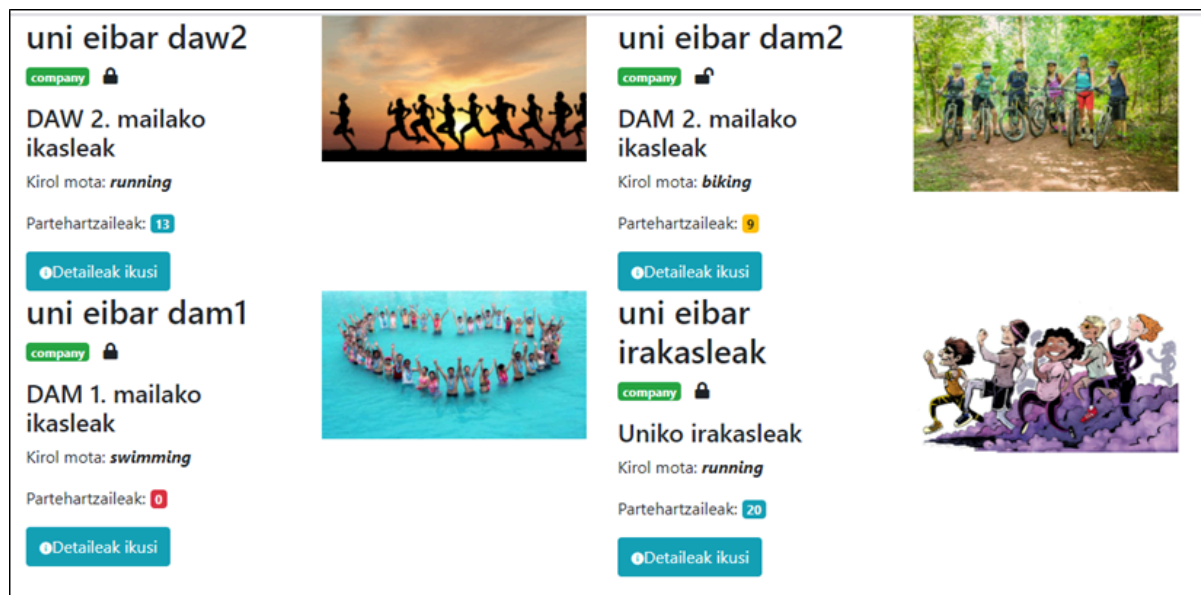
```
'warning'))">{{kluba.member count}}</ion-badge>
```

```
</p>
<ion-button color="primary" [routerLink]="['kluba', kluba.id]'>
  <ion-icon name="information-circle-outline"></ion-icon>Jarduerak ikusi
</ion-button>
</ion-col>
<ion-col size="6">
  <ion-img [attr.src]="kluba.cover_photo_small == null ?
'../../assets/img/generikoa.jpg' : kluba.cover_photo_small "></ion-img>
  </ion-col>
</ion-row>
</ion-grid>
....
</ion-content>
```

# REST API-a prestatzen

## [Ezkutatu](#)

Orain klub asko bistartzeko prestatuko dugu eta informazioa **REST API** batetik irakurriko dugu.



## Pausuak

- Rest Api-a Laravel (MySQL datu basearekin) batetik prestatuta edukiko dugu
  - Hau egiteko Apache+MySQL+phpMyAdmin erabiliko ditugu (Instalatuta badaukagu nahikoa da eta bestela XAMPP paketea erabili daiteke).
  - Php-aren bertsioa 8.0 edo handiagoa izan behar da. Hau jakiteko, xampp martxan jarri eta: <http://localhost/dashboard/phpinfo.php>
  - Bertsio barriago deskargatzeko: <https://windows.php.net/download#php-8.1> (VS16 x64 Thread Safe (2022-Nov-22 16:34:52))

Behin deskargatuta, deskonprimitu karpeta batean (adibidez, c:\xampp barruan) eta «path» sistemako ingurune aldagaia eguneratu, hortik irakurtzeko exekutagarria. Beste alde batetik, php.ini fitxategia karpeta barruan sortu (php.ini-development bezala egongo da seguruenik, beraz, izena aldatu php.ini bakarrik izateko)

- Laravel jartzeko: <https://laravel.com/docs/10.x/installation>
- **Laravel edukitzeko Composer behar dugu eta batzuetan XDebug luzapenarekin arazoak eduki ditzakegu (kasu hauetan xdebug lerroak komentatu daiteke instalatzeko. Ikusi irudia)**



```
; zend_extension = "path/to/xdebug.so"
; xdebug.mode = debug
; xdebug.start_with_request = yes
; xdebug.client_host = localhost
; xdebug.client_port = 9003
```

- phpMyAdmin erabiliz, MySQL-n **erabiltzailea sortu: strava (pasahitza:password)** eta izen berdinarekin **datu basea sortu**.
  - Batzuetan phpmyAdmin-ek erroreak ematen ditu "Cuentas de usuario" estekan klik egiterakoan: "#1034 - Clave de archivo erronea para la tabla: 'db'; intente repararlo". Taula hori konpontzeko, SQL editorea zabaldu eta agindu hau exekutatu: **REPAIR TABLE mysql.db;**
- Moodlen dagoen **.sql fitxategia** exekutatu
- Moodle-tik kodigoa deskargatu eta **c:\xampp\htdocs** barruan jarri apache zerbitzariak (xampp) exekutatzeko.
- **.env fitxategia** zabaldu eta behar diren konfigurazio datuak daudela baieztatu. Adibidez:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=strava
DB_USERNAME=strava
DB_PASSWORD=password
```

- cmd kotsola zabaldu, laravel-api karpetara sartu eta agindu hau exekutatu

**laravel-api> php artisan serve**

- Nabigatzailea zabaldu eta **url-a** hauek probatu:

**http://127.0.0.1:8000/api/klubak**

**http://127.0.0.1:8000/api/klubak/1**

# Klub guztien informazioa bistaratu

## [Ezkatatu](#)

Hiru klase sortuko ditugu "classes" karpeta berri batean:

```
> cd .\src\app\classes\
```

```
>ionic g class classes/kluba
```

```
>ionic g class classes/jarduera
```

```
>ionic g class classes/atleta
```

### **kluba.ts**

```
import { Jarduera } from './jarduera';
export class Kluba {
  id!: number;
  name!: string;
  cover_photo_small!: string;
  sport_type!: string;
  privatea!: boolean;
  member_count!: number;
  description!: string;
  club_type!: string;
  jarduerak!: Jarduera[];
}
```

(! erabiltzen dugu konpiladoreak ulertzeko aldagai hori, nahiz eta ez den konpilazio-orduan definitu, ejekutatzerakoan definituko dela, erabiltzen hasi aurretik. Ez da beharrezkoa konstruktore bat definitzen bada.)

### **jarduera.ts**

```
export class Jarduera {
  id!: number;
  name!: string;
  distance!: number;
  moving_time!: number;
  elapsed_time!: number;
  type!: string;
  workout_type!: number;
  atleta_id!: number;
}
```

### **atleta.ts**

```
export class Atleta {
  id!: string;
```

```

    firstname!: string;
    lastname!: string;
    kluba_id!: number;
}

```

Orain zerbitzu bat sortuko dugu "services" karpeta berrian:

**> ionic g service kluba**

**kluba.service.ts**

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

```

```

import { Kluba } from '../classes/kluba';

```

.....

```

export class KlubaService {

```

```

    private url = 'http://localhost:8000/api/klubak';
    constructor(private http: HttpClient) { }

```

```

    getKlubak(): Observable<Kluba[]>{
        return this.http.get<Kluba[]>(this.url);
    }
}

```

**app.module.ts**

```

import { provideHttpClient } from '@angular/common/http';

```

.....

```

providers: [provideHttpClient(), { provide: RouteReuseStrategy, useClass:
IonicRouteStrategy }],

```

.....

**tab1.page.ts**

```

import { KlubaService } from '../services/kluba.service';
import { Kluba } from '../classes/kluba';

```

....

```

export class Tab1Page implements OnInit{

```

```

    klubak: Kluba[] = [];

```

```
constructor(private klubaService: KlubaService) {}
```

```
getKlubak(): void{  
  this.klubaService.getKlubak()  
    .subscribe({  
      next: data => {this.klubak = data; },  
      error: error => console.log('Error:.' + error)  
    });  
}
```

```
ngOnInit(){  
  this.getKlubak();  
}
```

**tab1.page.html**

```
<ion-grid>  
  <ion-row>  
    <ion-col *ngFor="let kluba of klubak">  
      <ion-item-divider color="light">  
        <ion-row>  
          <ion-col size="6">  
  
            .....  
            .....  
          </ion-col>  
        </ion-row>  
      </ion-item-divider>  
    </ion-col>  
  </ion-row>  
</ion-grid>
```

Gure estilo bereziak sartzeko, **global.scss** fitxategian sartu ditzakegu

```
.....  
  
@import "~@ionic/angular/css/text-transformation.css";  
@import "~@ionic/angular/css/flex-utils.css";  
  
<span style="color: red;">@import "../assets/nirestiloa.css";
```

Eta **nirestiloa.css** fitxategian nahi ditugun estiloak ipiniko ditugu:

```
h3 {  
  color: red;
```

}

**Irudiak: assets/img karpetan sartu behar dira**

# Klub bakoitzaren jarduerak bistaratu

## [Ezkutatu](#)

Orain aplikazioa prestatuko dugu klub bakoitzaren jarduerak bistaratzeko. Hori egiteko orri berri bat sortuko dugu:

**ionicStrava> ionic g page tab1-jarduerak**

Enrutadorean aldaketak egin behar ditugu, orri berrira sartzeko

### **tab1-routing.module.ts:**

```
const routes: Routes = [  
  {  
    path: '',  
    component: Tab1Page,  
  },  
  {  
    path: 'kluba/:id',  
    loadChildren: () => import('../tab1-jarduerak/tab1-jarduerak.module').then(m =>  
m.Tab1JarduerakPageModule)  
  }  
];
```

Eta **\*\*\*EZABATU\*\*** hemendik:

### **app-routing.module.ts:**

```
{  
  path: 'tab1-jarduerak',  
  loadChildren: () => import('../tab1-jarduerak/tab1-jarduerak.module').then( m =>  
m.Tab1JarduerakPageModule)  
}
```

Hasierako orritik botoi bat prestatuta daukagu jarduerak ikusteko, **[routerLink]** atributua erabilita:

### **tab1.page.html**

```
<ion-button color="primary" [routerLink]="['kluba', kluba.id]'>  
  <ion-icon name="information-circle-outline"></ion-icon>Jarduerak ikusi  
</ion-button>
```

Orain, kodigoa, typescript eta html, orri berriarena. Kontutan izan behar dugu KlubaService erabiliko duela ere:

### **tab1-jarduerak.page.ts**

```

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Location } from '@angular/common';
import { KlubaService } from '../services/kluba.service';
import { Kluba } from '../interfaces/kluba';

@Component({
  selector: 'app-tab1-jarduerak',
  templateUrl: './tab1-jarduerak.page.html',
  styleUrls: ['./tab1-jarduerak.page.scss'],
})
export class Tab1JarduerakPage implements OnInit {

  kluba = {} as Kluba;

  constructor(private klubaService: KlubaService, private route: ActivatedRoute, private
location: Location) { }

  getKluba(): void {
    const id = Number(this.route.snapshot.paramMap.get('id'));
    this.klubaService.getKluba(id).subscribe({
      next: kluba => {
        this.kluba = kluba;
        this.kluba.jarduerak.sort((a, b): number => {
          return b.moving_time - a.moving_time;
        });
      },
      error: error => console.log('Error :: ' + error),
    });
  }
  goBack(): void {
    this.location.back();
  }
  ngOnInit() {
    this.getKluba();
  }
}

```

## tab1-jarduerak.page.html

```

<ion-header>
  <ion-toolbar>
    <ion-title>tab1-jarduerak</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-content class="ion-padding">

```

```

<ion-grid [fixed]="true">
  <ion-row>
    <ion-col>
      ><p><b>Atleta</b></p></ion-col>
    >
    <ion-col>
      ><p><b>lzena</b></p></ion-col>
    >
    <ion-col>
      ><p><b>Mota</b></p></ion-col>
    >
    <ion-col>
      ><p><b>mt.</b></p></ion-col>
    >
    <ion-col>
      ><p><b>sg.</b></p></ion-col>
    >
  </ion-row>
  <ion-row *ngFor="let aktibitatea of kluba.jarduerak">
    <ion-col> {{aktibitatea.atleta_id}} </ion-col>
    <ion-col> {{aktibitatea.name}} </ion-col>
    <ion-col> {{aktibitatea.type}} </ion-col>
    <ion-col> {{aktibitatea.distance}} </ion-col>
    <ion-col> {{aktibitatea.moving_time}} </ion-col>
  </ion-row>
</ion-grid>
<ion-button color="primary" size="small" (click)="goBack()"
  >Bueltatu</ion-button>
>
</ion-content>
</ion-content>

```

## kluba.service.ts

```
export class KlubaService {
```

```
.....
```

```

  getKluba(id: number): Observable<Kluba> {
    return this.http.get<Kluba>(this.url + '/' + id);
  }

```



## Klubaren jarduerak

Izena	Abizena	Mota	mt.	sg.
Izaskun	K.	Walk	20211.3	15220
Izaskun	K.	Ride	19000	10800
Diana	M.	Ride	17400	3600
Oskar	G.	Ride	6000	3600
Diana	M.	Run	0	1380
Oskar	G.	Run	834.5	434
Izaskun	K.	Walk	23.1	15

BUELTATU

## 4- IonicStrava - SQLite

# Instalazioak eta Konfigurazioa

### Ezkutatu

Orain informazioa **SQLite** motako datu basean gordeko dugu. Hau egiteko esteka honetan dauden pausuak jarraituko ditugu gure beharizanetara egokituta:

<https://devdactic.com/ionic-4-sqlite-queries>

**@ionic-native** ez du jarraituko eguneraketekin eta **@awesome-cordova-plugins** deituko da: <https://ionic.io/blog/a-new-chapter-for-ionic-native>

<https://github.com/danielsogl/awesome-cordova-plugins>

**Apache Cordova** erabiliko dugu: aplikazio mugikorren garapen framework bat da. Software programatzaileei CSS3, HTML5 eta JavaScript erabiltzen uzten die mugikorretarako web aplikazio hibridoak eraikitzeke, Android-en, iOS-en edo Windows Phone-en bezalako plataforma espezifikoetako APIetan oinarritu beharrean.

<https://cordova.apache.org/>

Lehenengo gauza Cordova instalatzea izango da (-g jarrita instalazio globala izateko) gure proiektuaren karpeta barruan:

```
ionicStrava> npm install -g cordova
```

Eta orain SQLite erabiltzeko hurrengo paketeak eta plugin-ak:

```
ionicStrava> npm i @awesome-cordova-plugins/core
```

```
ionicStrava> npm install @awesome-cordova-plugins/sqlite
```

```
ionicStrava> npm i cordova-sqlite-storage
```

```
ionicStrava> npm install @awesome-cordova-plugins/sqlite-porter
```

```
ionicStrava> npm i uk.co.workingedge.cordova.plugin.sqliteporter
```

Hurrengo komando hauek exekutatu behar ditugu, cordova plugina eta Android-eko gradlearekin konpilatu eta sinkronizatzeko.

```
ionicStrava> ionic cap add android
```

```
ionicStrava> ionic cap sync
```

Aurreko exekuzioa amaitu ondoren, proiektua zuzenean edo komando-lerro bidez ireki dezakegu, honako hauek exekutatuz:

## **ionicStrava> ionic cap open android**

Kontuan izan beharrezkoa dela Android Studio instalatuta izatea, baita SDKak ere aldeztu aurretik.

*\*\* SQLite Porter plugin-a SQL edo JSON formatuetatik SQLite datu-base batera inportatzeko eta esportatzeko erabiltzen da \*\**

Ondoren, inportatu eta erregistratu behar dira plugin horiek aplikazioaren moduluaren fitxategi nagusian, **app.module.ts**. Horrela, SQLite datu-basearen metodo eta zerbitzu guztietara sartu ahal izango gara.

**Http** eskaerak egingo ditugu datuak SQL datu-fitxategitik bertatik bidaltzeko, beraz, **httpClientModule** erabiliko dugu ere.

### **app.module.ts**

.....

```
import { AppRoutingModuleModule } from './app-routing.module';
```

```
import { AppComponent } from './app.component';
```

```
import { provideHttpClient } from '@angular/common/http';
```

*// plugins*

```
import { SQLite } from '@awesome-cordova-plugins/sqlite/ngx';
```

```
import { SQLitePorter } from '@awesome-cordova-plugins/sqlite-porter/ngx';
```

```
@NgModule({
```

```
  declarations: [AppComponent],
```

```
  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModuleModule],
```

```
  providers: [
```

```
    provideHttpClient(),
```

```
    SQLite,
```

```
    SQLitePorter,
```

```
    { provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],
```

```
    bootstrap: [AppComponent],  
  })  
  export class AppModule {}
```

# Kluba irakurri SQLite-arekin

## [Ezkutatu](#)

Lehenengo pausua zerbitzu berri bat sortzea izango da **"services"** karpetan:

```
> ionic g service services/api
```

```
api.service.ts
```

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

import { Platform } from '@ionic/angular';

import { Kluba } from '../classes/kluba';

import { Jarduera } from '../classes/jarduera';

import { HttpClient } from '@angular/common/http';

import { BehaviorSubject, Observable, of } from 'rxjs';

import { SQLitePorter } from '@awesome-cordova-plugins/sqlite-porter/ngx';

import { SQLite, SQLiteObject } from '@awesome-cordova-plugins/sqlite/ngx';
```

```
@Injectable({
  providedIn: 'root'
})

export class ApiService {

  private storage!: SQLiteObject;

  klubakList = new BehaviorSubject<Kluba[]>([]);

  JarduerakList = new BehaviorSubject<Jarduera[]>([]);

  private isDbReady: BehaviorSubject<boolean> = new BehaviorSubject(false);

  constructor(
    private platform: Platform,
```

```

private sqlite: SQLite,
private httpClient: HttpClient,
private sqlPorter: SQLitePorter,
) {
    this.platform.ready().then(() => {
        this.sqlite.create({
            name: 'Strava_db.db',
            location: 'default'
        })
        .then((db: SQLiteObject) => {
            this.storage = db;
            this.getData();
        });
    });
}

```

//Datu basea listo dagoen jakiteko, tab1 orrian erabiltzen da

```

dbState() {
    return this.isDbReady.asObservable();
}

```

// Render data

```

getData() {

```

//Lehen aldia bada, taula sortuko du datu batzuekin (sqlPorter erabiltzen du sql-tik datubasera pasatzeko). Gero konexioa badago sinkronizatu eta amaieran getKlubak() exekutatu da.

```

this.httpClient.get(
    'assets/dump.sql',

```

```

    {responseType: 'text'}
  ).subscribe(data => {

    this.sqlPorter.importSqlToDb(this.storage, data)

    .then(_ => {

      this.getKlubak();

      this.isDbReady.next(true);

    })

    .catch(error => console.error(error));

  });
}

// Kluben zerrenda prestatu, konstruktoreetik deitzen zaio
async getKlubak(){
  try {

    const res = await this.storage.executeSql('SELECT * FROM klubas', []);

    let items: Kluba[] = [];

    console.log(res);

    if (res.rows.length > 0) {

      for (var i = 0; i < res.rows.length; i++) {

        items.push({

          id: res.rows.item(i).id,

          name: res.rows.item(i).name,

          cover_photo_small: res.rows.item(i).cover_photo_small,

          sport_type: res.rows.item(i).sport_type,

          privatea: res.rows.item(i).privatea,

          member_count: res.rows.item(i).member_count,

```

```

        description: res.rows.item(i).description,
        club_type: res.rows.item(i).club_type,
        jarduerak: []
    });
}
}

this.klubakList.next(items);
} catch (error) {
    console.error ("errorea getKlubak", error);
}
}

```

```

//getKlubak() sortutako zerrenda bueltatzen du, tab1 orrian erabiltzen da
fetchKlubak(): Observable<Kluba[]> {
    return this.klubakList.asObservable();
}

```

**Garrantzitsua da kodigo honetan datu basea nola sortzen/erabiltzen den. Nahiz eta agindua "this.sqlite.create()" izan, ez du behin eta berriz datubasea sortzen. Bakarrik lehen aldian eta gero existitzen denez, zabaldu egiten du.**

Probarako datuak edukitzeko eta "getData()" funtzioa erabiltzeko, "assets" karpeta barruan fitxategi bat sortuko dugu: **dump.sql**

**assets/dump.sql**

```

CREATE TABLE IF NOT EXISTS klubas(

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT,

    cover_photo_small TEXT,

    sport_type TEXT,

```



```
privatea BOOLEAN,  
  
member_count NUMBER,  
  
description TEXT,  
  
club_type TEXT  
  
);
```

```
INSERT or IGNORE INTO klubas(id, name, cover_photo_small, sport_type, privatea,  
member_count, description, club_type) VALUES (1, 'uni eibar dam1',  
'../assets/img/klubadam1.jpg', 'swimming', true, 0, 'DAM 1. mailako ikasleak',  
'company');
```

```
INSERT or IGNORE INTO klubas(id, name, cover_photo_small, sport_type, privatea,  
member_count, description, club_type) VALUES (2, 'uni eibar irakasleak',  
'../assets/img/klubairakasleak.jpg', 'running', true, 20, 'Uniko irakasleak',  
'company');
```

```
INSERT or IGNORE INTO klubas(id, name, cover_photo_small, sport_type, privatea,  
member_count, description, club_type) VALUES (3, 'Uni eibar daw2',  
'../assets/img/klubadaw2.jpg', 'running', true, 13, 'DAW 2. mailako ikasleak',  
'company');
```

```
INSERT or IGNORE INTO klubas(id, name, cover_photo_small, sport_type, privatea,  
member_count, description, club_type) VALUES (4, 'uni eibar dam2',  
'../assets/img/klubadam2.jpg', 'biking', false, 9, 'DAM 2. mailako ikasleak',  
'company');
```

```
CREATE TABLE IF NOT EXISTS atletas (
```

```
    id TEXT,  
  
    firstname TEXT,  
  
    lastname TEXT,  
  
    kluba_id NUMBER  
  
);
```

```
INSERT or IGNORE INTO atletas (id, firstname, lastname, kluba_id) VALUES  
(111111111A, 'izaskun', 'Kortabitarte', 1);
```

```
INSERT or IGNORE INTO atletas (id, firstname, lastname, kluba_id) VALUES
```

```
('22222222B', 'pedro', 'perez', 1);
```

```
CREATE TABLE IF NOT EXISTS jardueras (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    name TEXT,  
    distance NUMBER,  
    moving_time NUMBER,  
    elapsed_time NUMBER,  
    type TEXT,  
    workout_type NUMBER,  
    atleta_id TEXT  
);
```

```
INSERT or IGNORE INTO jardueras (id, name, distance, moving_time,  
elapsed_time, type, workout_type, atleta_id) VALUES
```

```
(1, 'prueba bat', 10.00, 2.00, 2.00, 'korrika', 1, '11111111A');
```

```
INSERT or IGNORE INTO jardueras (id, name, distance, moving_time,  
elapsed_time, type, workout_type, atleta_id) VALUES
```

```
(2, 'beste jarduera bat', 30.00, 1.50, 2.00, 'bizikletan', 2, '22222222B');
```

**tab1.page.html**

\*\*\* Ez dago aldaketa garrantzitsurik egin behar \*\*\*

Ionic duen osagai bat erabiliko dugu: datuak kargatzen dauden bitartean ikusteko barra antzeko bat da

```
<ion-header [translucent]="true">
```

```
<ion-toolbar>
```

```
<ion-title> Kluba </ion-title>
```

```
<ion-progress-bar type="indeterminate" *ngIf="showLoader"></ion-progress-bar>
```

```
</ion-toolbar>
```

```
</ion-header>
```

### tab1.page.ts

**Aurretik sortu genuen KlubaService erabili barik utziko dugu, geroago erabiltzeko, eta zerbitzu berria erabiliko dugu**

```
import { Component, OnInit } from '@angular/core';
import { KlubaService } from '../services/kluba.service';
import { ApiService } from '../services/api.service';
import { Kluba } from '../classes/kluba';
```

```
@Component({
  selector: 'app-tab1',
  templateUrl: 'tab1.page.html',
  styleUrls: ['tab1.page.scss']
})
export class Tab1Page implements OnInit {
  klubak: Kluba[] = [];
  showLoader=true;

  constructor(private apiService: ApiService) {}
  getKlubak(): void {
    this.apiService.dbState().subscribe((res) => {
      if(res){
        this.apiService.fetchKlubak().subscribe(
          data => {this.klubak = data;
            this.showLoader=false;}
        )
      }
    });
  }

  ngOnInit(){
    this.getKlubak();
  }
}
```

**App-a martxan jartzeko android emuladore bat behar dugu SQLite datu basea erabiltzeko.**

**\*\*\*\*\* Begiratu atal honen azkenengo zatian \*\*\*\*\***

# Klubaren jarduerak bistaratu

## [Ezkutatu](#)

Klub baten jarduerak bistaratuko ditugu orain. Horretarako, zerbitzua aldatu behar da eta metodo berri bat gehitu klub baten jarduerak bilatzeko eta array-ean gordetzeko.

### **api.service.ts**

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

import { Platform } from '@ionic/angular';

import { Kluba } from '../classes/kluba';

import { HttpClient } from '@angular/common/http';

import { BehaviorSubject, Observable, of } from 'rxjs';

import { SQLitePorter } from '@awesome-cordova-plugins/sqlite-porter/ngx';

import { SQLite, SQLiteObject } from '@awesome-cordova-plugins/sqlite/ngx';

import { Jarduera } from '../classes/jarduera';
```

```
@Injectable({
  providedIn: 'root'
})

export class ApiService {

  private storage!: SQLiteObject;

  klubakList = new BehaviorSubject<Kluba[]>([]);

  private isDbReady: BehaviorSubject<boolean> = new BehaviorSubject(false);

  constructor(

    private platform: Platform,

    private sqlite: SQLite,
```

```

private httpClient: HttpClient,
private sqlPorter: SQLitePorter,
) {
    this.platform.ready().then(() => {
        this.sqlite.create({
            name: 'Strava_db.db',
            location: 'default'
        })
        .then((db: SQLiteObject) => {
            this.storage = db;
            this.getData();
        });
    });
}

```

//Datu basea listo dagoen jakiteko, tab1 orrian erabiltzen da

```

dbState() {
    return this.isDbReady.asObservable();
}

```

// Render data

```

getData() {

```

//Lehen aldia bada, taula sortuko du datu batzuekin (sqlPorter erabiltzen du sql-tik datubasera pasatzeko). Gero konexioa badago sinkronizatu eta amaieran getKlubak() exekutatuko da.

```

this.httpClient.get(
    'assets/dump.sql',
    {responseType: 'text'}

```

```

).subscribe(data => {

  this.sqlPorter.importSqlToDb(this.storage, data)

  .then(_ => {

    this.getKlubak();

    this.isDbReady.next(true);

  })

  .catch(error => console.error(error));

});

}

// Kluben zerrenda prestatu, konstruktoreetik deitzen zaio
async getKlubak(){

  try {

    const res = await this.storage.executeSql('SELECT * FROM klubas', []);

    let items: Kluba[] = [];

    console.log(res);

    if (res.rows.length > 0) {

      for (var i = 0; i < res.rows.length; i++) {

        //jarduerak lortzen dira -> getJarduerak(id)

        const jarduerak = await this.getJarduerak(res.rows.item(i).id) || [];

        items.push({

          id: res.rows.item(i).id,

          name: res.rows.item(i).name,

          cover_photo_small: res.rows.item(i).cover_photo_small,

          sport_type: res.rows.item(i).sport_type,

          privatea: res.rows.item(i).privatea,

```



```

        moving_time: res.rows.item(i).moving_time,
        elapsed_time: res.rows.item(i).elapsed_time,
        type: res.rows.item(i).type,
        workout_type: res.rows.item(i).workout_type,
        atleta_id: res.rows.item(i).atleta_id
    });
    }
}

return items;
} catch (error) {
    console.error("errorea getJarduerak", error);
    return [];
}
}

//getKlubak() sortutako zerrenda bueltatzen du, tab1 orrian erabiltzen da
fetchKlubak(): Observable<Kluba[]> {
    return this.klubakList.asObservable();
}

//getKluba() lortutako datuak bueltatzen ditu, tab1-jarduerak orrian erabiltzen da
fetchKluba(id: any): Observable<Kluba> {
    const kluba = this.klubakList.value.find(kluba => kluba.id === id);
    return of(kluba || {} as Kluba);
}

```



Eta jardueren orrian aldaketak egin behar ditugu zerbitzuko metodo berria erabiltzeko.

**tab1-jarduerak.page.ts**

```
import { ApiService } from '../services/api.service';
```

```
constructor(
```

```
  private klubaService: KlubaService,
```

```
  private apiService: ApiService,
```

```
  private route: ActivatedRoute,
```

```
  private location: Location
```

```
) {}
```

```
getKluba(): void {
```

```
  this.apiService.dbState().subscribe((res) => {
```

```
    if(res){
```

```
      const id = Number(this.route.snapshot.paramMap.get('id'));

```

```
      this.apiService.fetchKluba(id).subscribe(kluba => {
```

```
        this.kluba = kluba;
```

```
        this.kluba.jarduerak.sort( (a, b): number => {
```

```
          return (b.moving_time - a.moving_time);
```

```
        });
```

```
      }
```

```
    })
```

```
  });
```

```
}
```

# Klub berria sortu

## [Ezkatatu](#)

Klub berria alta emateko, botoi bat jarriko dugu eta orri berria sortuko dugu: **klubaGehitu**

**ionicStrava> ionic g page klubaGehitu**

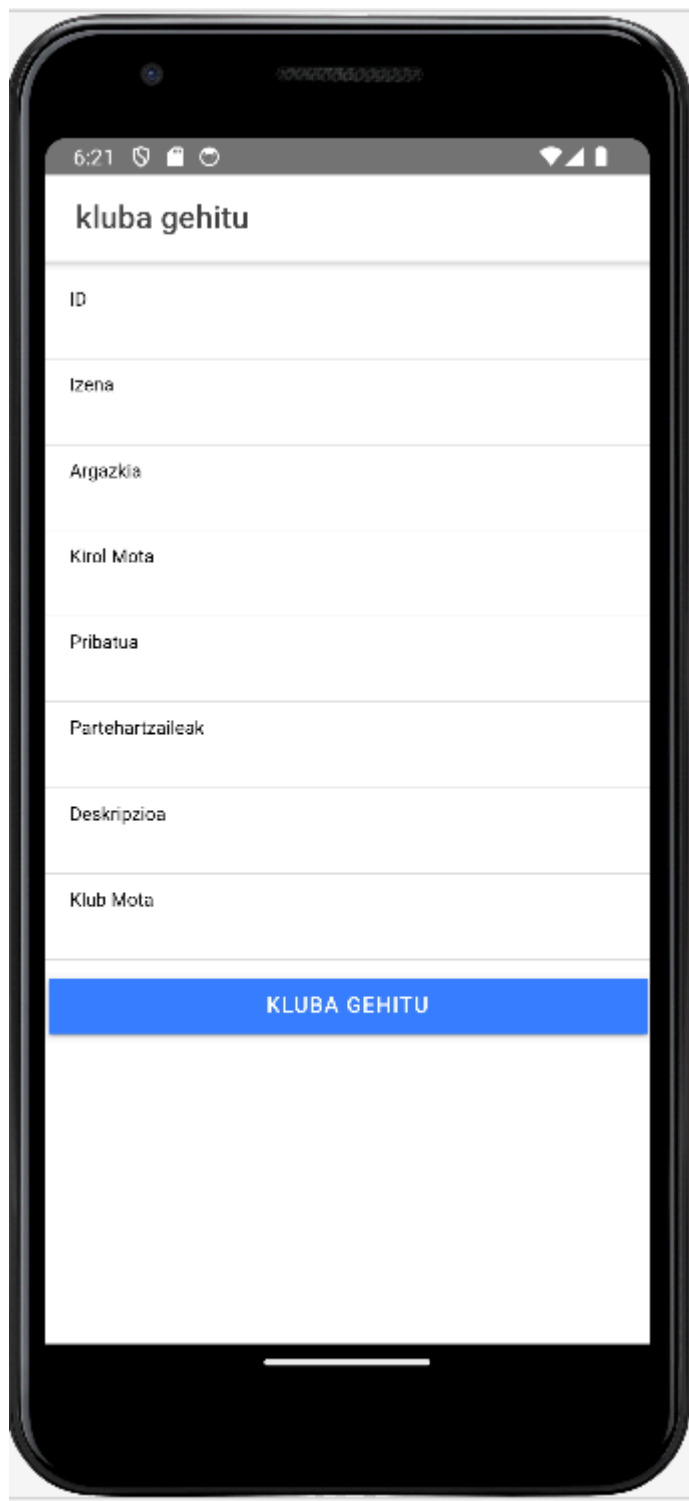
Botoi bat jarriko dugu **tab1 orritik**, formulario bat ikusteko eta datuak sartzeko. Beheko aldian gehitu kodigoa:

**tab1.page.html**

```
<ion-fab vertical="bottom" horizontal="center" slot="fixed">
  <ion-fab-button color="success" [routerLink]="['/kluba-gehitu']">
    <ion-icon name="add"></ion-icon>
  </ion-fab-button>
</ion-fab>
```

Eta formulario bat erakutsiko da:

**kluba-gehitu.page.html**



```
<ion-content [fullscreen]="true">
```

```
<ion-header collapse="condense">
```

```
<ion-toolbar>
```

```
<ion-title size="large">kluba gehitzeko formularioa</ion-title>
```

</ion-toolbar>

</ion-header>

<ion-content>

<ion-list>

<ion-item>

<ion-label position="stacked">Izena</ion-label>

<ion-input [(ngModel)]="kluba.name" type="text"></ion-input>

</ion-item>

<ion-item>

<ion-label position="stacked">Argazkia</ion-label>

<ion-input [(ngModel)]="kluba.cover\_photo\_small" type="text"></ion-input>

</ion-item>

<ion-item>

<ion-label position="stacked">Kirol Mota</ion-label>

<ion-input [(ngModel)]="kluba.sport\_type" type="text"></ion-input>

</ion-item>

<ion-item>

<ion-label position="stacked">Pribatua</ion-label>

<ion-input [(ngModel)]="kluba.privatea" type="text"></ion-input>

</ion-item>

```
<ion-item>

  <ion-label position="stacked">Partehartzaileak</ion-label>

  <ion-input [(ngModel)]="kluba.member_count" type="text"></ion-input>

</ion-item>
```

```
<ion-item>

  <ion-label position="stacked">Deskripzioa</ion-label>

  <ion-input [(ngModel)]="kluba.description" type="text"></ion-input>

</ion-item>
```

```
<ion-item>

  <ion-label position="stacked">Klub Mota</ion-label>

  <ion-input [(ngModel)]="kluba.club_type" type="text"></ion-input>

</ion-item>
```

```
</ion-list>
```

```
<ion-button expand="full" (click)="gehituForm()">Kluba Gehitu</ion-button>
```

```
</ion-content>
```

```
</ion-content>
```

**Eta kodigoan: kluba-gehitu.page.ts (gorriz dagoena gehitu)**

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

```
import { Kluba } from '../classes/kluba';
```

```
import { ApiService } from '../services/api.service';
```

```
import { NavController } from '@ionic/angular';
```

```
@Component({
```

```
  selector: 'app-kluba-gehitu',
```

```
  templateUrl: './kluba-gehitu.page.html',
```

```
  styleUrls: ['./kluba-gehitu.page.scss'],
```

```
})
```

```
export class KlubaGehituPage {
```

```
  constructor(private apiService: ApiService, private navCtrl: NavController) {}
```

```
  kluba = {} as Kluba;
```

```
  errorMessage= "";
```

```
  gehituForm(): void {
```

```
    if (!this.kluba) { return; }
```

```
    try {
```

```
      this.apiService.addKluba(this.kluba);
```

```
      this.reset();
```

```
      //Hasierako orrira bueltatu
```

```
      this.navCtrl.navigateForward('tabs/tab1');
```

```
    } catch (error) {
```

```
      this.errorMessage = error as any;
```

```
    }
```

```

    }

    reset(): void {

        this.kluba = {

            id: 0,

            name: "",

            cover_photo_small: "",

            sport_type: "",

            privatea: false,

            member_count: 0,

            description: "",

            club_type: "",

            jarduerak: []

        };

    }

}

```

Zerbitzua ere aldatu behar dugu Kluba gehitzeko (lehengoari gehitu hurrengo kodigoa):

#### **api.service.ts**

// Add - Lerro berria gehitu eta klub guztiak irakurri

```

async addKluba(kluba: Kluba) {

    let data = [kluba.name, kluba.cover_photo_small, kluba.sport_type, kluba.privatea,
kluba.member_count, kluba.description, kluba.club_type];

    alert(data);

    const res = await this.storage.executeSql('INSERT INTO klubas (name,
cover_photo_small, sport_type, private, member_count, description, club_type)
VALUES (?, ?, ?, ?, ?, ?, ?)', data);

```

```
this.getKlubak();
```

```
}
```



# App-a martxan jarri

## [Ezkatatu](#)

Emuladore bat eduki behar dugu aplikazioa ikusteko eta baita JAVA eta SDK bat, beraz, guk [Android Studio](#) instalatzea aukeratuko dugu. Aplikazio horrek guztia emango digu, eta, gainera, aurrerago ondo etorriko zaigu.

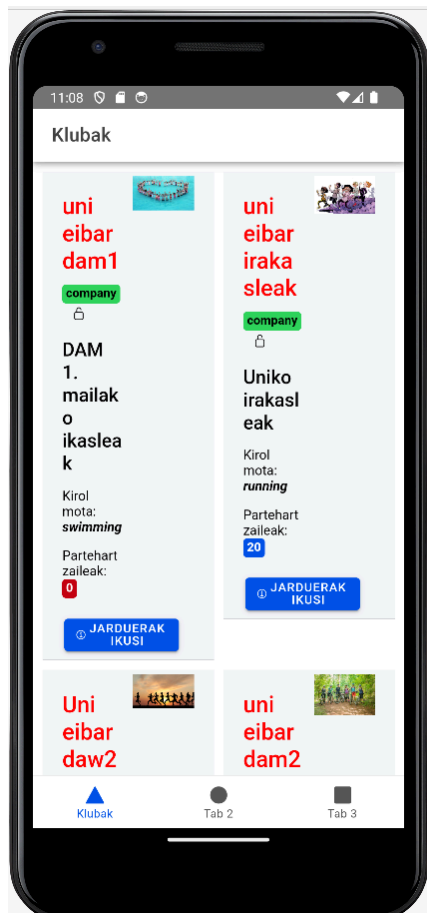
Mobil birtual bat sortu behar dugu, "Device Manager" tresna topatu eta "Create Device" egin. Adibidez: **Pixel 3a**.

**Ionic-eko aplikazioa martxan ipini aurretik, mobil birtuala martxan egon behar da**

**IonicStrava> ionic capacitor run** (plataforma bezala, Android aukeratu, eta gero, sortu dugun dispositiboa)

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\Aprobak_DAW_2023_2024\IonicStrava> ionic capacitor run
? What platform would you like to run? android
? Which device would you like to target? Genymobile Pixel 2 (192.168.114.101:5555)
> ng.cmd run app:build
```

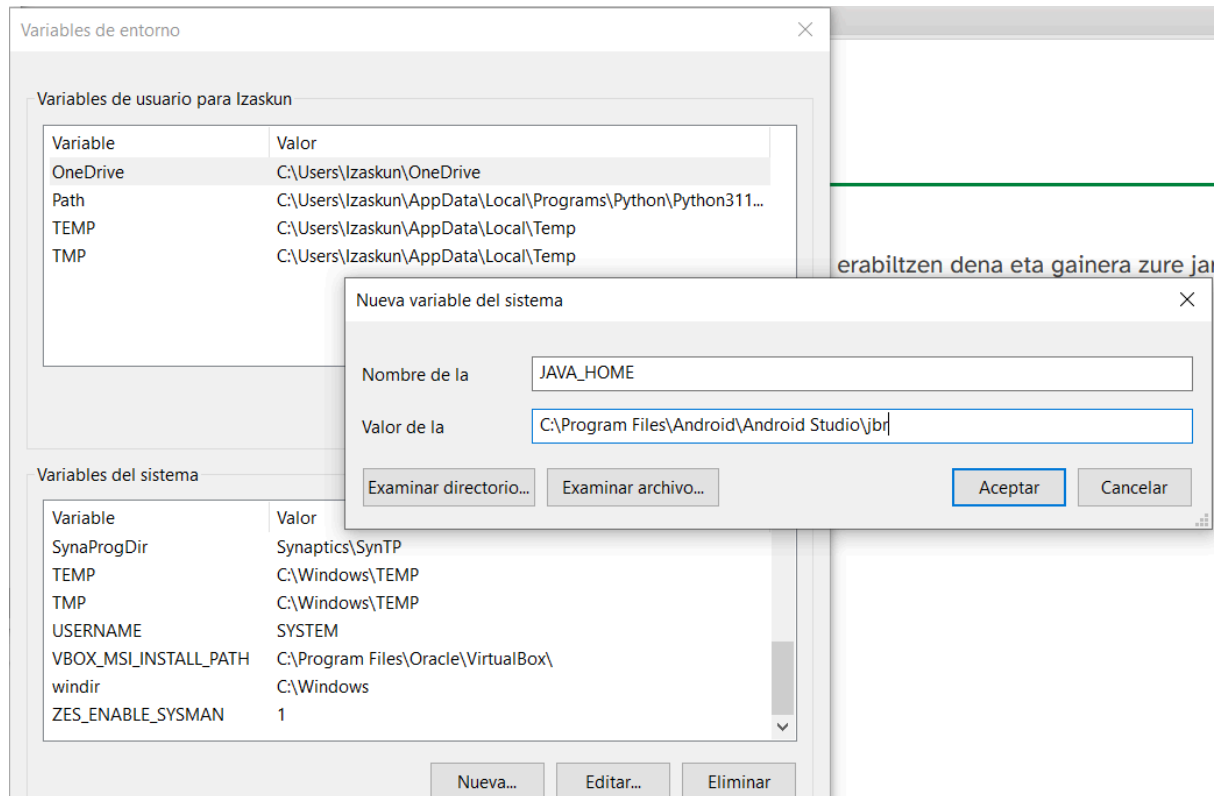


**Gertatu ahal diren errore batzuk:**

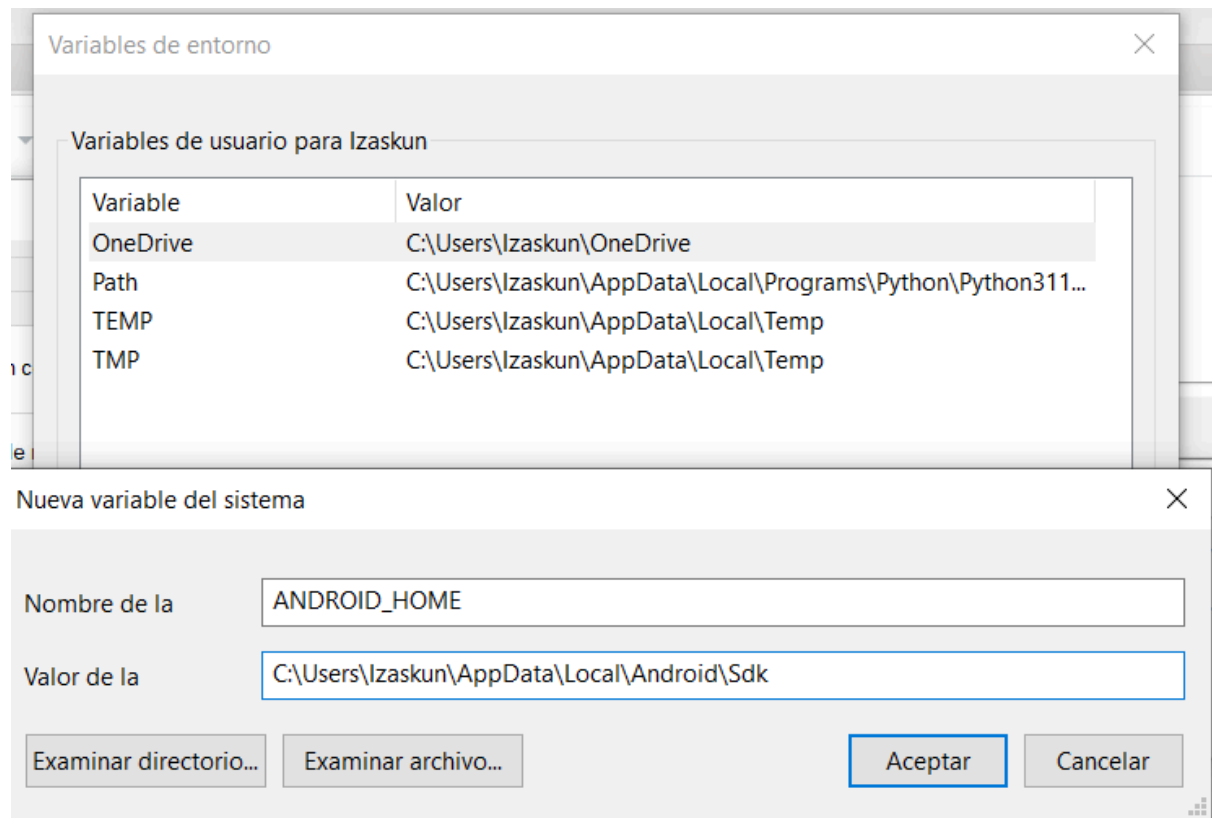
[https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-fix-common-JAVA\\_HOME-errors-quickly](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-fix-common-JAVA_HOME-errors-quickly)

**JAVA\_HOME** non dagoen ez du bilatzen, beraz, ingurune aldagai bezala gehituko dugu:

**C:\Program Files\Android\Android Studio\jbr**



Berdina gertatzen da **ANDROID\_HOME** ingurune aldagaiarekin: **Android Studio - Tools - SDK Manager -> C:\Users\Izaskun\AppData\Local\Android\Sdk**




Aldagai hauek gehituta gero **berrabiarazi behar dugu.**

# Depurazioa

## [Ezkutatu](#)

Erroreak baditugu aplikazioa depuratu behar dugu. Horretarako **Google Chrome** erabil daiteke: **chrome://inspect/devices**

Hor agertzen diren dispositiboetatik aukeratu behar dena eta "inspect" eman.

 Chrome | chrome://inspect/devices#devices

---

## Devices

☒ Discover USB devices Port forwarding...

☒ Discover network targets Configure...

[Open dedicated DevTools for Node](#)

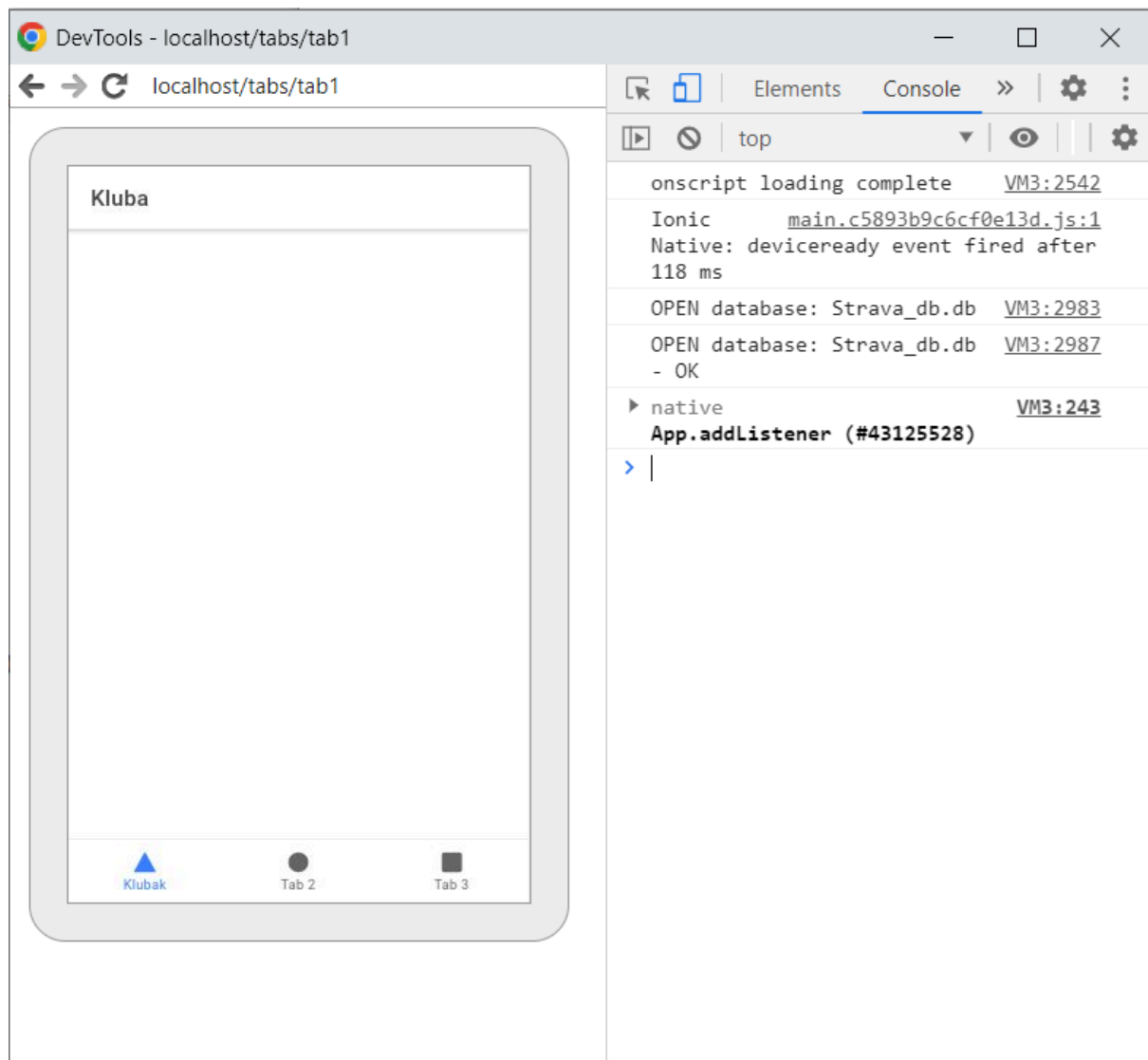
---

## Remote Target #LOCALHOST

**Pixel 2** #192.168.114.102:5555

**WebView in io.ionic.starter (83.0.4103.120)** [trace](#)

Ionic App <http://localhost/tabs/tab1>  
at (0, 63) size 1080 × 1731  
[inspect](#) [pause](#)



Eta adibidez datu basea nahi badugu ikusi, Android Studio datorren "Device Manager - Device File Explorer" erabiliz gure datu basea topatu ahal dugu:

**/data/data/io.ionic.starter/databases/Strava\_db.db** eta deskargatu eta beste aplikazio batekin ikuskatu: [DB Browser for SQLite](#)

Device Manager

VirtualPhysical

Create device

Device

API

Size on Disk

Actions

No virtual devices added. Create a virtual device to test applications without owning a physical device.  
Create virtual device

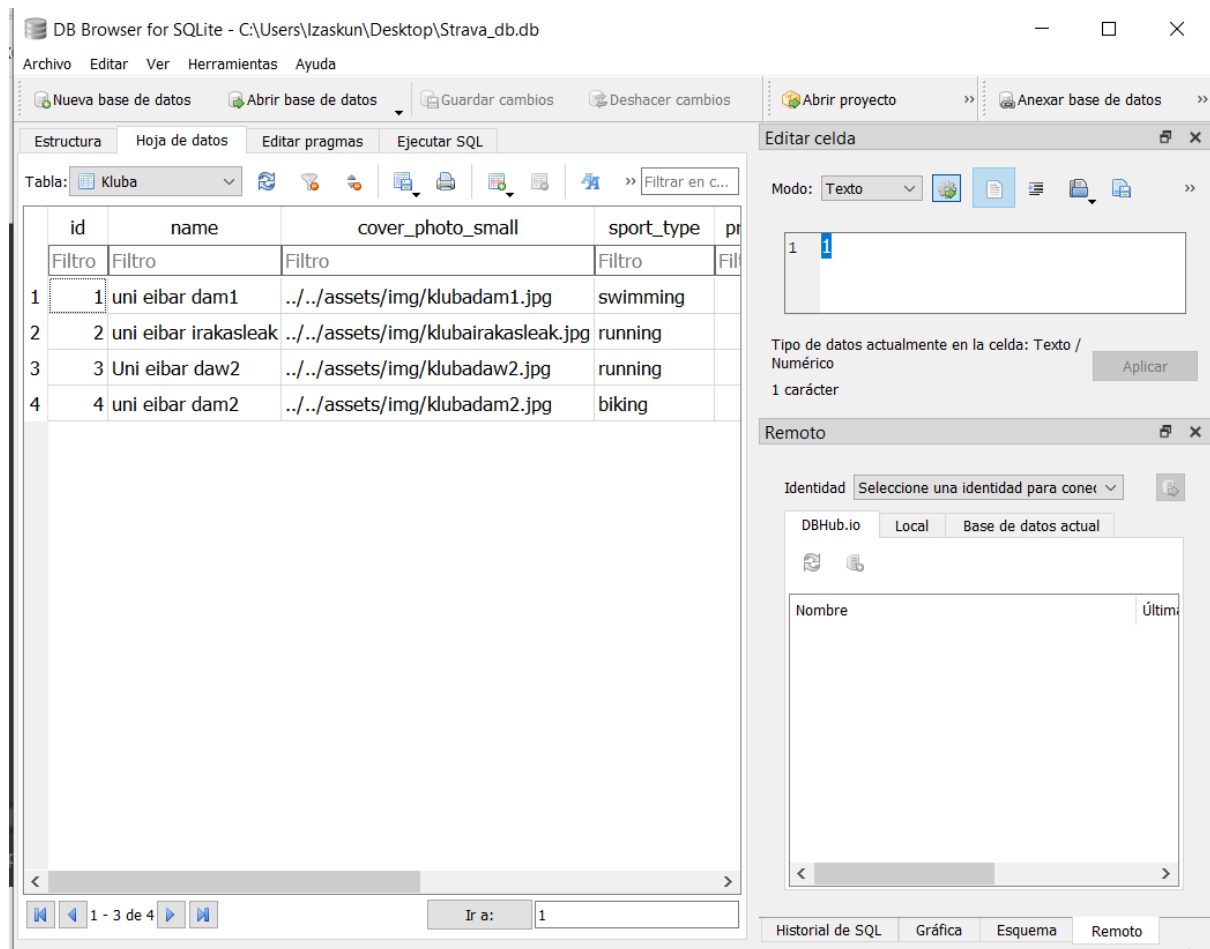
Device File Explorer

Genymobile Pixel 2 Android 11, API 30

Name	Permissions	Date	Size
> com.android.wallpaperpicker	drwxr-x--x	2023-05-04 10:33	4 KB
> com.android.webview	drwx-----	2023-05-04 10:46	4 KB
> com.android.wifi.resources	drwx-----	2023-05-04 10:33	4 KB
> com.example.android.livecubes	drwx-----	2023-05-04 10:33	4 KB
> com.genymotion.genyd	drwx-----	2023-05-04 10:33	4 KB
> com.genymotion.settings	drwx-----	2023-05-04 10:33	4 KB
> com.genymotion.superuser	drwx-----	2023-05-04 10:33	4 KB
> com.genymotion.systempatcher	drwx-----	2023-05-04 10:33	4 KB
> com.genymotion.tasklocker	drwx-----	2023-05-04 10:33	4 KB
> com.google.android.launcher.layouts.genyr	drwx-----	2023-05-04 10:33	4 KB
▼ io.ionic.starter	drwx-----	2023-05-04 10:46	4 KB
> app_textures	drwxrwx--x	2023-05-04 10:46	4 KB
> app_webview	drwx-----	2023-05-04 10:46	4 KB
> cache	drwxrws--x	2023-05-04 10:46	4 KB
> code_cache	drwxrws--x	2023-05-04 10:46	4 KB
▼ databases	drwxrwx--x	2023-05-04 10:46	4 KB
Strava_db.db	-rw-----	2023-05-04 10:46	12 KB
> shared_prefs	drwxrwx--x	2023-05-04 10:46	4 KB
>			

App InspectionLayout Inspector

Device ManagerGradleNotificationsDevice File ExplorerEmulator



Beste aukera bat da android studio aplikazioan dagoen plugin bat erabil daiteke: **Database Navigator**

Eta gero, berrabiarazi android studio eta **DB Browser** panela zabaldu eta konexio berria sortu.





## 5- IonicStrava - Osoa

### Ezkutatu

Gehienetan, Internet konexioa dugu gure mobilean, baina gertatu daiteke konexioa ez edukitzea momentu batzuetan. Gure aplikazioa martxan jarraitu behar du konexioa eduki ala ez. Hau lortzeko estrategia bat planteatu behar dugu:

**Gure aplikazioak SQLite datu basean dagoen informazioa erabiliko du beti.** Baina, zer gertatzen da datuak aldatzen baditugu, bai SQLite-an eta/edo RESTAPI-an? Datuen aldaketarik egiten badugu, SQLite datu basean eta REST API-an ez dira datu berdinak egongo. Sinkronizatzeko zerbait behar dugu, eta hori ondo egiteko, transakzioen taula bat erabiliko dugu aldaketak gordetzen joateko.

**Aurreko guztia kontutan izanda, pausu hauek egin behar ditugu:**

- Aplikaziora lehen aldiz sartzerakoan, konexioa dugun ala ez konprobatuko da. Holan bada, sinkronizazioa egingo dugu (behean azalduta) eta bigarren pausura pasatuko gara. Ez badago konexiorik, bigarren pausura pasatuko gara.
- SQLite datu baseko informazioa bistaratuko dugu.
- Edozein aldaketa eginez gero (INSERT, UPDATE, DELETE), SQLite-an egingo da eta Transakzioen taulan gordeko da ere. Jarraian, konexiorik badago, sinkronizazioa egingo dugu eta ez badago, ez da ezer egiten.
- Sinkronizazioa:
  - Transakzioen taulako eguneraketak REST API-ra pasatu, egin barik geratu diren aldaketa guztiak eguneratzeko urruneko datu basean.
  - REST API-a SQLiten kargatu (azkenengo bertsioan dagoelako).
  - Eguneratutakoa transakzioa taulatik ezabatu.
- Edozein momentutan freskatzeko eskatzen bada, eta konexioa badugu, sinkronizazioa egingo dugu.

Esteka honetan agertzen diren pausuak jarraituko ditugu gure beharizanetara egokituta:

<https://enappd.com/blog/build-offline-ionic-angular-apps/192/>

# Network zerbitzua

## [Ezkatatu](#)

Konexioa daukagun ala ez jakiteko zerbitzu bat sortuko dugu eta capacitor-rek daukan liburutegi bat erabiliko dugu.

**IonicStrava> npm install @capacitor/network**

```
npm install @capacitor/network
npm ERR! code ERESOLVE
npm ERR! ERESOLVE unable to resolve dependency tree
npm ERR!
npm ERR! While resolving: IonicStrava@0.0.1
npm ERR! Found: @capacitor/core@4.7.3
npm ERR! node_modules/@capacitor/core
npm ERR! @capacitor/core@"4.7.3" from the root project
npm ERR!
npm ERR! Could not resolve dependency:
npm ERR! peer @capacitor/core@"^5.0.0" from @capacitor/network@5.0.2
npm ERR! node_modules/@capacitor/network
npm ERR! @capacitor/network@"*" from the root project
```

Batzuetan bertsioen arazoak gertatzen dira. Holan bada, behar den bertsio zenbakia instalatzen saiatu behar gara

<https://www.npmjs.com/package/@capacitor/network?activeTab=versions> :

**IonicStrava> npm i @capacitor/network@4.1.1-nightly-20230316T150919.0**

**IonicStrava> npx cap sync**

**npx cap sync**, web karpetako assetak jatorrizko karpetara kopiatze ditu bakarrik.  
**ionic cap sync** egiten badugu, konpilatu eta kopiatzen du app-ak behar duen guztia.

**IonicStrava> ionic g service services/network**

**network.service.ts**

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

import { Network } from '@capacitor/network';
```

```
@Injectable({
  providedIn: 'root'
})
```

```
export class NetworkService {  
  
  connected = true;  
  
  constructor() {  
  
    // Hasierako egoera jakin  
  
    this.checkInitialStatus();  
  
    Network.addListener('networkStatusChange', async status => {  
  
      console.log('Sarearen konexioa aldatu da:', status);  
  
      this.connected = status.connected;  
  
    });  
  }  
  
  async checkInitialStatus() {  
  
    const status = await Network.getStatus();  
  
    console.log('Sarearen hasierako egoera:', status);  
  
    this.connected = status.connected;  
  
  }  
  
  getStatus(): boolean {  
  
    return this.connected;  
  
  }  
}
```

# Transakzio zerbitzua

## [Ezkatatu](#)

**assets/dump.sql** fitxategia aldatuko dugu taula bat gehiago erabiltzeko, egin barik geratu diren transakzioak gordetzeko:

```
CREATE TABLE IF NOT EXISTS pending_transactions (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    endpoint TEXT NOT NULL,  
    method TEXT NOT NULL,  
    payload TEXT NOT NULL,  
    timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP  
);
```

Lau zutabe ditu:

**id**: automatikoko gehitzen doan gako nagusia.

**endpoint**: REST API urruneko zerbitzuaren transakzioaren url amaiera.

**method**: http metodoa (GET, POST, PUT, DELETE, etab.).

**payload**: transakzioaren payload-a, JSON edo XML izan daitekeena.

**timestamp**: zein momentutan izan den egin barik geratu den transakzioa.

Gainera klase berri bat sortuko dugu taula horretako objektuak kudeatzeko: **Transaction**

**ionicStrava> ionic g class classes/transaction**

**transaction.ts**

```
export class Transaction {  
    id: number;  
    endpoint: string;  
    method: string;  
    payload: string;  
    timestamp: string;
```

```

    constructor(id: number, endpoint: string, method: string, payload: string, timestamp:
string) {

        this.id = id;

        this.endpoint = endpoint;

        this.method = method;

        this.payload = payload;

        this.timestamp = timestamp;

    }
}

```

Transakzio zerbitzu bat sortuko dugu: **TransactionService**

**IonicStrava> ionic g service services/transaction**

**transaction.service.ts**

```

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

import { SQLitePorter } from '@awesome-cordova-plugins/sqlite-porter/ngx';

import { SQLite, SQLiteObject } from '@awesome-cordova-plugins/sqlite/ngx';

import { Platform } from '@ionic/angular';

import { Transaction } from '../classes/transaction';

@Injectable({
    providedIn: 'root'
})

export class TransactionService {

    private storage!: SQLiteObject;

    constructor(

        private platform: Platform,

```

```

private sqlite: SQLite,
private sqlPorter: SQLitePorter,
) {
  this.platform.ready().then(() => {
    this.sqlite.create({
      name: 'Strava_db.db',
      location: 'default'
    })
    .then((db: SQLiteObject) => {
      this.storage = db;
    });
  });
}

//Transakzio berria gehitzeko
async addTransaction(transaction: Transaction) {
  //insert-a;

  const query = 'INSERT INTO pending_transactions (endpoint, method, payload)
VALUES (?, ?, ?)';

  this.storage.executeSql(query, [transaction.endpoint, transaction.method,
transaction.payload])
    .catch(error => console.error('Errorea transakzioa gehitzen', error));
}

//Dauden transakzio guztiak lortzeko
async getPendingTransactions(): Promise<Transaction[]> {
  const query = 'SELECT * FROM pending_transactions';
  return this.storage.executeSql(query, [])

```

```

.then(result => {
    const transactions: Transaction[] = [];
    for (let i = 0; i < result.rows.length; i++) {
        const row = result.rows.item(i);
        transactions.push({
            id: row.id,
            endpoint: row.endpoint,
            method: row.method,
            payload: row.payload,
            timestamp: row.timestamp
        });
    }
    return transactions;
})

.catch(error => {
    console.error('Errorea egiteke dauden transakzioak lortzen', error);
    return [];
});
}

//Transakzio bat (id) ezabatzeko
async removeTransaction(transaction: Transaction) {
    const query = 'DELETE FROM pending_transactions WHERE id = ?';
    this.storage.executeSql(query, [transaction.id])
        .catch(error => console.error('Errorea transakzio bat ezabatzen', error));
}

```

```
//Transakzio guztiak ezabatzeko  
async removeAllTransactions() {  
  const query = 'DELETE FROM pending_transactions';  
  this.storage.executeSql(query)  
    .catch(error => console.error('Errorea transakzio guztiak ezabatzen', error));  
}  
}
```



# Sinkronizazio zerbitzua

## [Ezkutatu](#)

Sinkronizazio lanak egiteko, zerbitzu berri bat sortuko dugu: **SyncService**

**ionicStrava> ionic g service services/sync**

**sync.service.ts**

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

import { Transaction } from '../classes/transaction';

import { Kluba } from '../classes/kluba';

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { NetworkService } from '../network.service';

import { TransactionService } from '../transaction.service';

import { SQLiteObject } from '@awesome-cordova-plugins/sqlite/ngx';
```

```
@Injectable({
  providedIn: 'root'
})
```

```
export class SyncService {
  private storage!: SQLiteObject;
```

```
  //url nagusia, orain ip-a jarriko da bestela mobilaren localhost-arekin nahasketa
  sortzen da
```

```
  private url = 'http://192.168.56.1:8000/api/klubak';
```

```
  constructor(
    private networkService: NetworkService,
```

```
private httpClient: HttpClient,  
private transactionService: TransactionService  
) {}
```

```
synchronize() {
```

//pending\_transactions taulatik dauden lerro bakoitzeko irakurri eta  
sendTransaction metodora bidali. Gero, synchronize\_2 metodora.

```
if (this.networkService.getStatus()) {  
    this.transactionService.getPendingTransactions().then(transactions => {  
        transactions.forEach(transaction => {  
            this.sendTransaction(transaction);  
        });  
    });  
    this.synchronize_2()  
}  
}
```

//transakzio bakoitza REST API-ra pasatu modu asinkronoan eta gero transakzioa  
ezabatu.

```
async sendTransaction(transaction: Transaction): Promise<void> {  
    const headers = new HttpHeaders().set('Content-Type', 'application/json');  
    this.httpClient.request(transaction.method, transaction.endpoint, { body:  
transaction.payload, headers: headers } )  
    .subscribe({  
        next: (response: any) => {  
            const kluba: Kluba = response;  
            console.log('Received Kluba object:', kluba);
```

```

        this.transactionService.removeTransaction(transaction);
    },
    error: (error: any) => {
        console.error('Error synchronizing transaction', error);
    }
});
}

```

// SQLite-ko taula osoa ezabatu eta REST API-koa deskargatu (azkenengo bertsioan dago) modu asinkronoan eta SQLitekoa kargatu

```

async synchronize_2(): Promise<void>{
    //lehenengo REST API-aren datu guztiak deskargatu
    this.httpClient.get<Kluba[]>(this.url).subscribe(apiData => {
        //Lokalean dagoen guztia ezabatu
        this.storage.executeSql('DELETE FROM klubas');
        //REST API-an dagoena lokalera pasatu
        apiData.forEach(Record => {
            const { id, name, cover_photo_small, sport_type, privatea, member_count,
description, club_type } = Record;
            this.storage.executeSql('INSERT INTO klubas (id, name, cover_photo_small,
sport_type, private, member_count, description, club_type) VALUES (?, ?, ?, ?, ?, ?,
?, ?)', [id, name, cover_photo_small, sport_type, privatea, member_count,
description, club_type]);
        });
    });
}

```

}

# API zerbitzua eguneratu

## [Ezkatatu](#)

Aurretik sortu genuen zerbitzua, **"ApiService"** eguneratuko dugu. Zerbitzu honek beste zerbitzu guztiak erabiliko ditu, eta kudeatuko du momentu bakoitzean zer egin behar den. Konexioa duen ala ez galdetu, eta horren arabera sinkronizazioa egin lehenengo eta gero SQLite datu baseko informazioa bistaratu. Aldaketak egiterakoan ere hasieran aipatu diren pausuak emango ditu transakzio taula erabiliz.

### **api.service.ts**

Lehenengo lerro hau // **@ts-nocheck**, objektu batzuen eremuak hutsik hasten uzteko erabiltzen dugu (adibidez transakzioak)

// @ts-nocheck

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

import { Platform } from '@ionic/angular';

import { Kluba } from '../classes/kluba';

import { Jarduera } from '../classes/jarduera';

import { HttpClient } from '@angular/common/http';

import { BehaviorSubject, Observable, of } from 'rxjs';

import { SQLitePorter } from '@awesome-cordova-plugins/sqlite-porter/ngx';

import { SQLite, SQLiteObject } from '@awesome-cordova-plugins/sqlite/ngx';

import { Transaction } from '../classes/transaction';

import { NetworkService } from '../network.service';

import { SyncService } from '../sync.service';

import { TransactionService } from '../transaction.service';
```

```
@Injectable({
  providedIn: 'root'
```

```
}}
```

```
export class ApiService {
```

```
    private storage!: SQLiteObject;
```

```
    klubakList = new BehaviorSubject<Kluba[]>([]);
```

```
    JarduerakList = new BehaviorSubject<Jarduera[]>([]);
```

```
    private isDbReady: BehaviorSubject<boolean> = new BehaviorSubject(false);
```

```
    //url nagusia, orain ip-a jarriko da bestela mobilaren localhost-arekin nahasketa  
    sortzen da
```

```
    private url = 'http://192.168.56.1:8000/api/klubak'; //edo 10.0.2.2
```

```
    constructor(
```

```
        private platform: Platform,
```

```
        private sqlite: SQLite,
```

```
        private httpClient: HttpClient,
```

```
        private sqlPorter: SQLitePorter,
```

```
        private networkService: NetworkService,
```

```
        private syncService: SyncService,
```

```
        private transactionService: TransactionService
```

```
    ) {
```

```
        this.platform.ready().then(() => {
```

```
            this.sqlite.create({
```

```
                name: 'Strava_db.db',
```

```
                location: 'default'
```

```
            })
```

```

        .then((db: SQLiteObject) => {

            this.storage = db;

            this.getData();

        });

    });

}

```

//Datu basea listo dagoen jakiteko, tab1 orrian erabiltzen da

```

dbState() {

    return this.isDbReady.asObservable();

}

```

// Render data

```

getData() {

```

//Lehen aldia bada, taula sortuko du datu batzuekin (sqlPorter erabiltzen du sql-tik datubasera pasatzeko). Gero konexioa badago sinkronizatu eta amaieran getKlubak() exekutatu da.

```

    this.httpClient.get(

        'assets/dump.sql',

        {responseType: 'text'}

    ).subscribe(data => {

        this.sqlPorter.importSqlToDb(this.storage, data)

        .then(_ => {

            if (this.networkService.getStatus()){

                //online gaude. Sinkronizatu

                this.syncService.synchronize();

            }

            this.getKlubak();

```

```

        this.isDbReady.next(true);

    })

    .catch(error => console.error(error));

});

}

// Kluben zerrenda prestatu, konstruktoreetik deitzen zaio
async getKlubak(){

    try {

        const res = await this.storage.executeSql('SELECT * FROM klubas', []);

        let items: Kluba[] = [];

        if (res.rows.length > 0) {

            for (var i = 0; i < res.rows.length; i++) {

                //jarduerak lortzen dira -> getJarduerak(id)

                const jarduerak = await this.getJarduerak(res.rows.item(i).id);

                items.push({

                    id: res.rows.item(i).id,

                    name: res.rows.item(i).name,

                    cover_photo_small: res.rows.item(i).cover_photo_small,

                    sport_type: res.rows.item(i).sport_type,

                    privatea: res.rows.item(i).privatea,

                    member_count: res.rows.item(i).member_count,

                    description: res.rows.item(i).description,

                    club_type: res.rows.item(i).club_type,

                    jarduerak: jarduerak

                });

            }

        }

    }

}

```



```

    }
}

this.klubakList.next(items);

} catch (error) {

    console.error ("errorea getKlubak", error);

}

}

// Klub bateko jarduerak lortzeko

async getJarduerak(id: any){

    try {

        const res = await this.storage.executeSql('SELECT j.id as id, j.name as name,
j.distance as distance, j.moving_time as moving_time, j.elapsed_time as
elapsed_time, j.type as type, j.workout_type as workout_type, j.atleta_id as atleta_id
FROM jardueras as j, atletas as a WHERE j.atleta_id = a.id and a.kluba_id = ?', [id]);

        let items: Jarduera[] = [];

        if (res.rows.length > 0) {

            for (var i = 0; i < res.rows.length; i++) {

                items.push({

                    id: res.rows.item(i).id,

                    name: res.rows.item(i).name,

                    distance: res.rows.item(i).distance,

                    moving_time: res.rows.item(i).moving_time,

                    elapsed_time: res.rows.item(i).elapsed_time,

                    type: res.rows.item(i).type,

                    workout_type: res.rows.item(i).workout_type,

                    atleta_id: res.rows.item(i).atleta_id
                });
            }
        }
    } catch (error) {
        console.error("errorea getJarduerak", error);
    }
}

```

```

    });
  }
  return items;
}
} catch (error) {
  console.error("errorea getJarduerak", error);
  return [];
}
}

```

//getKlubak() sortutako zerrenda bueltatzen du, tab1 orrian erabiltzen da

```

fetchKlubak(): Observable<Kluba[]> {
  return this.klubakList.asObservable();
}

```

//getKluba() lortutako datuak bueltatzen ditu, tab1-jarduerak orrian erabiltzen da

```

fetchKluba(id: any): Observable<Kluba> {
  const kluba = this.klubakList.value.find(kluba => kluba.id === id);
  return of(kluba);
}

```

// Add - Lerro berria gehitu, transakzio taulara pasatu eta sarea badugu sinkronizatzen saiatu.

```

async addKluba(kluba: Kluba) {

  let data = [kluba.name, kluba.cover_photo_small, kluba.sport_type, kluba.privatea,
kluba.member_count, kluba.description, kluba.club_type];

  const res = await this.storage.executeSql('INSERT INTO klubas (name,
cover_photo_small, sport_type, privatea, member_count, description, club_type)
VALUES (?, ?, ?, ?, ?, ?, ?)', data);

```

```
let payload = {  
    name: kluba.name,  
    cover_photo_small: kluba.cover_photo_small,  
    sport_type: kluba.sport_type,  
    private: kluba.privatea,  
    member_count: kluba.member_count,  
    description: kluba.description,  
    club_type: kluba.club_type  
};  
  
const jsonString: string = JSON.stringify(payload);  
  
let transaction: Transaction = {  
    endpoint : this.url,  
    method : "POST",  
    payload : jsonString,  
};  
  
this.addTransaction(transaction);  
  
if (this.networkService.getStatus()){  
    //online gaude. Sinkronizatu  
    this.syncService.synchronize();  
}  
  
this.getKlubak();  
}  
  
// Update - Eguneratu eta transakzio taulara pasatu  
async updateKluba(id: any, kluba: Kluba) {
```

```
let data = [kluba.name, kluba.cover_photo_small, kluba.sport_type, kluba.privatea, kluba.member_count, kluba.description, kluba.club_type];
```

```
const res = await this.storage.executeSql(`UPDATE Klubas SET name = ?, cover_photo_small = ?, sport_type = ?, privatea = ?, member_count = ?, description = ?, club_type = ? WHERE id = ${id}`, data);
```

```
let payload = {  
  name: kluba.name,  
  cover_photo_small: kluba.cover_photo_small,  
  sport_type: kluba.sport_type,  
  private: kluba.privatea,  
  member_count: kluba.member_count,  
  description: kluba.description,  
  club_type: kluba.club_type  
};  
  
const jsonString: string = JSON.stringify(payload);  
  
let transaction: Transaction = {  
  endpoint : this.url + '/' + id,  
  method : "PUT",  
  payload : jsonString,  
};  
  
this.addTransaction(transaction);  
  
if (this.networkService.getStatus()){  
  //online gaude. Sinkronizatu  
  this.syncService.synchronize();  
}
```

```

        this.getKlubak();
    }

    // Delete - Ezabatu eta transakzio taulara pasatu
    async deleteKluba(id: any) {
        const _ = await this.storage.executeSql('DELETE FROM Klubas WHERE id = ?',
[id]);

        let transaction: Transaction = {
            endpoint : this.url + '/' + id,
            method : "DELETE",
            payload : "",
        };
        this.addTransaction(transaction);
        if (this.networkService.getStatus()){
            //online gaude. Sinkronizatu
            this.syncService.synchronize();
        }
        this.getKlubak();
    }

    //transakzio taulan gordetzeko
    async addTransaction(transaction: Transaction) {
        this.transactionService.addTransaction(transaction);
    }
}

```

# Erroreak

## Laravel API at `http://192.168.56.1:8000/api/klubak` results in a connection being refused

### [Ezkutatu](#)

IonicStrava prestatu dugu SQLite eta REST API-a erabiltzeko, beraz, sinkronizazioa martxan eduki behar dugu. Gure aplikazioa Android Studio barruan dagoen mobil emuladorean daukagu martxan eta REST API-a atzitu nahi bada, ezin dugu **localhost** erabili, emuladoreak, mobila bera dela ulertzen baitu. Beraz, IP-a erabili behar dugu. Moldaketa honek beste arazo txiki bat sortzen du Laravel API-a martxan jartzerakoan, orain arte localhost-en entzuteko bakarrik eduki dugulako eta hemendik aurrera interfaze guztietan entzuteko prest egon behar da, ez localhost bakarrik.

Hori lortzeko hurrengo agindu honekin jarri behar dugu martxan, eta horrela edozein interfazetan entzuteko prest egoteko esaten diogu:

```
laravel-api> php artisan serve --host=0.0.0.0
```

# Failed to load resource: net::ERR\_CLEARTEXT\_NOT\_PERMITTED

[Ezkatatu](#)

\*\*\* Errore hau ez da seguruenik gertatuko, baina badaezpada, hemen azalduta nola konpondu daitekeen \*\*\*

To configure the `network_security_config.xml` file in your Ionic project for allowing clear text (non-HTTPS) traffic, specifically for localhost, follow these steps:

Create the `network_security_config.xml` File: This file should be placed in the `res/xml` directory of your Android project. If this directory doesn't exist, you need to create it. The path is typically **`platforms/android/app/src/main/res/xml`**.

Edit the **`network_security_config.xml`** File: Open or create the `network_security_config.xml` file and add the following configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <base-config cleartextTrafficPermitted="true">
    <trust-anchors>
      <certificates src="system" />
    </trust-anchors>
  </base-config>

  <domain-config cleartextTrafficPermitted="true">
    <domain includeSubdomains="true">localhost</domain>
    <domain includeSubdomains="true">10.0.2.2</domain> <!-- for Android emulator -->
  </domain-config>
</network-security-config>
```

This configuration allows clear text traffic for localhost and 10.0.2.2 (which is the alias for localhost on the Android emulator).

Reference the Config File in your `AndroidManifest.xml`: Open the **`AndroidManifest.xml`** file in your Android project (usually located at **`platforms/android/app/src/main/AndroidManifest.xml`**) and add the `networkSecurityConfig` attribute to the **`<application>`** tag:

```
<application
  android:networkSecurityConfig="@xml/network_security_config"
```

```
... >  
...  
</application>
```

Build Your Ionic Project: After making these changes, you need to build your Ionic project to apply them. Use the following command:

**IonicStrava> npx cap sync android**

Testing: Run your application on an Android device or emulator to test the changes. Ensure that your API requests to localhost are now working.

Remember, allowing clear text traffic is not recommended for production environments due to security risks. This configuration should only be used for development purposes. For production, always use HTTPS to ensure secure data transmission.



## 6- Gehigarriak

[Ezkutatu](#)

Ionic-eko temak erabili daitezke: `src/theme/variables.scss`

<https://ionicframework.com/docs/theming/basics>

Theming atal honetan, Platform Styles estekan, informazio hau ikusi ditzakegu:

Platform	Mode	Description
ios	ios	Viewing on an iPhone, iPad, or iPod will use the iOS styles.
android	md	Viewing on any Android device will use the <b>Material Design styles.</b>
core	md	Any platform that doesn't fit any of the above platforms will use the Material Design styles.

**Material Design:** <https://m2.material.io/design>

(Material Design is a key approach to the Android Platform for both UI and UX. Designed and developed by Google it is intended to facilitate a consistent user experience on the platform so that apps do not promote user confusion)

Eta gainera: For example, an app being viewed on an Android platform will use the md (Material Design) mode by default

Beraz, «Ionic components» erabiltzen ditugunean, md estiloa jarraituko dute.

<https://ionicframework.com/docs/components>

Angular Material erabiltzeko Ionic aplikazio batean:

<https://devdactic.com/angular-material-ionic-4>

# Osagaia - Adibidea: Datepicker

[Ezkutatu](#)

<https://material.angular.io/components/datepicker/examples>

**app.module.ts:**

```
import { MatNativeDateModule } from '@angular/material/core';

@NgModule({

  declarations: [AppComponent],

  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule,
HttpClientModule, MatNativeDateModule, BrowserAnimationsModule],

  providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],

  bootstrap: [AppComponent],

})
```

**Html-a:**

```
<mat-form-field class="example-full-width" appearance="fill">

  <mat-label>Choose a date</mat-label>

  <input matInput [matDatepicker]="picker">

  <mat-hint>MM/DD/YYYY</mat-hint>

  <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-toggle>

  <mat-datepicker [dateClass]="dateClass" #picker></mat-datepicker>

</mat-form-field>
```

**Css-a:**

```
button.example-custom-date-class {

  background: orange;
```

```
border-radius: 100%;
```

```
}
```

### **Tab3-page.ts:**

```
import { Component, ViewChild, OnInit, AfterViewInit, ViewEncapsulation } from  
'@angular/core';
```

```
import { MatCalendarCellClassFunction } from '@angular/material/datepicker';
```

```
@Component({
```

```
  selector: 'app-tab3',
```

```
  templateUrl: 'tab3.page.html',
```

```
  styleUrls: ['tab3.page.scss'],
```

```
  encapsulation: ViewEncapsulation.None,
```

```
})
```

```
export class Tab3Page implements AfterViewInit {
```

```
.....
```

```
  dateClass: MatCalendarCellClassFunction<Date> = (cellDate, view) => {
```

```
    // Only highlight dates inside the month view.
```

```
    if (view === 'month') {
```

```
      const date = cellDate.getDate();
```

```
      // Highlight the 1st and 20th day of each month.
```

```
      return date === 1 || date === 20 ? 'example-custom-date-class' : '';
```

```
    }
```

```
    return '';
```

```
  };
```

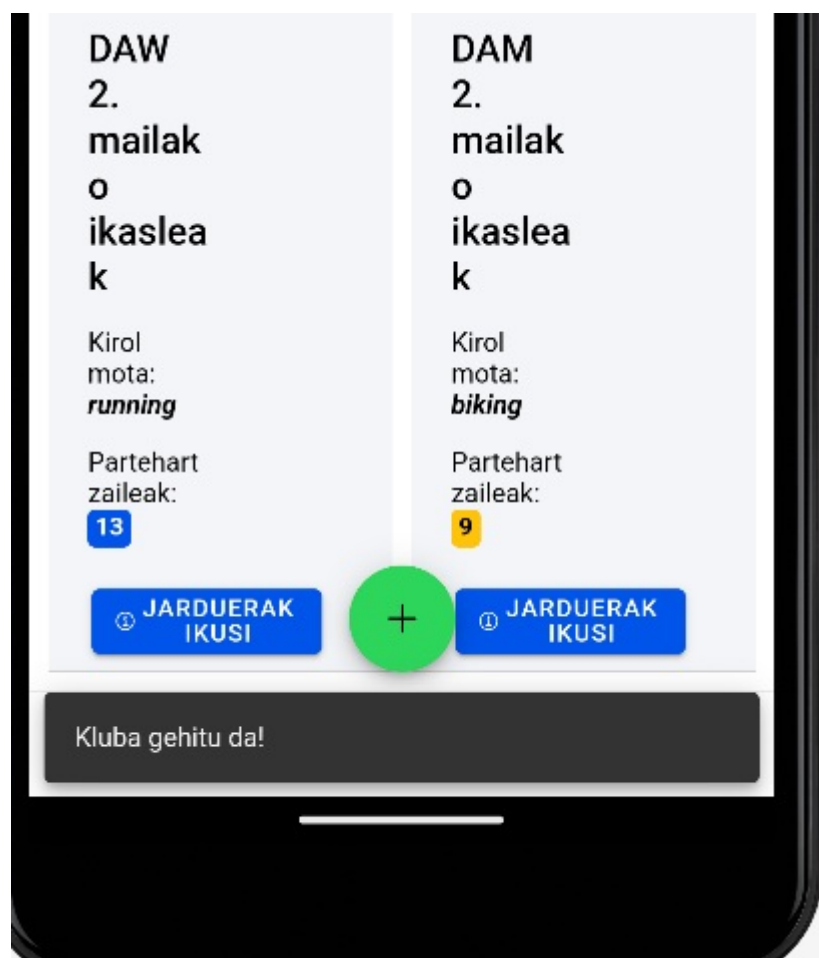
```
}
```

# 7- Ariketak

## 1 - Ariketa

[Ezkutatu](#)

**Kluba-gehitu** orrian hobekuntza bat txertatu: **Toast** motako osagai bat bistaratu klub berria sortzen dugunean erabiltzaileari abisatzeko.



## 2 - Ariketa

### [Ezkatatu](#)

**Tab2** pestaina prestatu behar da atleta guztiak bistaratzeko:

- **api.service.ts** eguneratu behar da atletak irakurtzeko datu basetik: metodo berri bat, adibidez, **fetchAtletak()**. Metodo hau ez da konstruktoretik deitu behar. Bigarren pestainatik deituko zaio behar denean.
- **tab2.page.ts** eguneratu behar da, **ngOnInit()** metodoa gehitzeko (hemendik api zerbitzuari deitzeko)
- **tab2.page.html** eguneratu atleten datuak bistaratzeko
- **Sinkronizazio** prozesu guztia gainbegiratu datu baseko **taula guztiak ondo sinkronizatzeko** eta ez Klubas taula bakarrik. Gainera **ApiService** erabili behar du tauletan aldaketak egiteko.

Hurrengo pausua, atleten jarduerak bistaratzea izango da.

## 3 - Ariketa

### [Ezkatatu](#)

ApiService gainbegiratu behar da hobekuntza bat txertatzeko:

- **addKluba, updateKluba eta deleteKluba aldatu**, eta metodo generikoak sortu, edozein taulatan aldaketak egiteko bezala.
- **addTaula, updateTaula eta deleteTaula** deituko dira metodo generiko berriak