AKE TENIS APP

Resource planning application

Contenido

1- Introduction

AKE Tenis App is an application created for a tennis club located in Arrasate/Mondragon. This app is set to run in mobile devices with Android or iOS operating systems, but it can also serve as a web application to run in computers.

For programming this application, I used different technologies for front end and back end. I wanted to use the newest hybrid-programming platform to optimize programming process on the front end, and for that we did some research and decided that Flutter was going to be the platform.

Flutter is a new platform created by Google that uses Dart language. It is a relatively new platform so it does not have all the functionalities that another platform could have, but they way to program in this language is very fast, so for a project like this, I think its appropriate. With this technology, we will be able to create projects for different operative systems using only one source code.

For the backend of the application, we decided to use Laravel/PHP platform to serve as an API, which will receive HTTP requests and will respond with the desired information. This platform also provides a quick and relatively easy way to make functional API solutions in very short time.

Storing all the information in a database is necessary, so for that we will use MySQL installed with the XAMPP package. To manage the database, we will use SQLYog Community Edition, which provides a nice visual interface and tools to generate database scripts.

2- Project context

This project is the final task of the web developing technical course and it has to be completed in three months. We were given total freedom in terms of which kind of application we wanted to make so we decided that it will be a nice idea to do a mobile application for the tennis club I play in.

It will serve as a way for the owners and the coaches to keep the needed information or used resources up to date, kind of like an ERP application, but adapted to a tennis club, which will not use many of those ERP functionalities.

This document will be a technical explanation about everything that needed to be done to create the solution, and it will include most of the needed explanations to understand what has been done.

3- Main goals/functionalities

As I said before, the main goal of this application is to centralize every component or resource needed in order to run a tennis club. This are the main functionalities that the application will have (not definitive, could be changed):

- Read and see information about the tennis sessions
    - See how many people are attending each session
    - Spot vacant slots in the sessions
    - Amount of sessions per three months
    - Adding players to a session if possible
- Add, Delete, Edit new players and assign them to different sessions
    - Personal information such as name, surname, telephone number and email address to enable direct calling from the app or email sending
- List of payments made by the players
- List of material purchases with descriptions, quantities and price

4- Extra functionalities

Aside from the main functionalities, we thought it would be a good idea to implement extra features for the app,

- Racket stringing historical
    - Amount of times a player has strung a racket, tension and type of string used in the string bed.

5- Tasks

In order to maintain a certain order when doing the application, we will try to list most of the tasks we are going to be doing and estimate the amount of time we will need to get it done.

Keeping the same rhythm when programming is vital, because otherwise things always are left behind and they never get finished. As a helping mechanism, we will use Canvan methodology with Trello, where each day new things will add and move as the project moves forward.

Here is a short list of the tasks we will perform during this project (subjected to changes, not definitive):

- Installation and setting up platforms
    - XAMPP and SQLYog installation
    - Visual Studio Code setup and Flutter package installation
    - Android Studio installation
    - Composer installation (PHP package manager for laravel)
- Research and tutorials
    - Flutter tutorial
    - Laravel/PHP tutorial (documentation)
    - Flutter state management tutorial (screen management)
    - Beginning Flutter (book)
- Database creation
    - Design first relational diagram
    - Create tables and script
    - Fill tables with significant data
- Layout design
    - Color palette, primary and secondary colors to use
    - Design the layouts that the application will have
        - Login layout
        - Listview and item design
        - Information layouts for classes, players
    - Brainstorm ideas
- Code implementation FRONTEND
    - Login screen
    - Listview screen and functionalities
    - Information screen implementation and functionalities
- Code implementation BACKEND

- Login credentials response, user info
- Object requests and related object information

6- Resources for the task

A group of a couple of members should have done an application like this with the work divided in different segments for each task. In this case, only one person did all tasks. To put an example in the case of human resources, this is an ideal composition:

- Front end code programmer
- Back end and database programmer
- Layout designer
- Lead programmer

With a group of four people, everything is covered in the application, but obviously, this is not always the case.

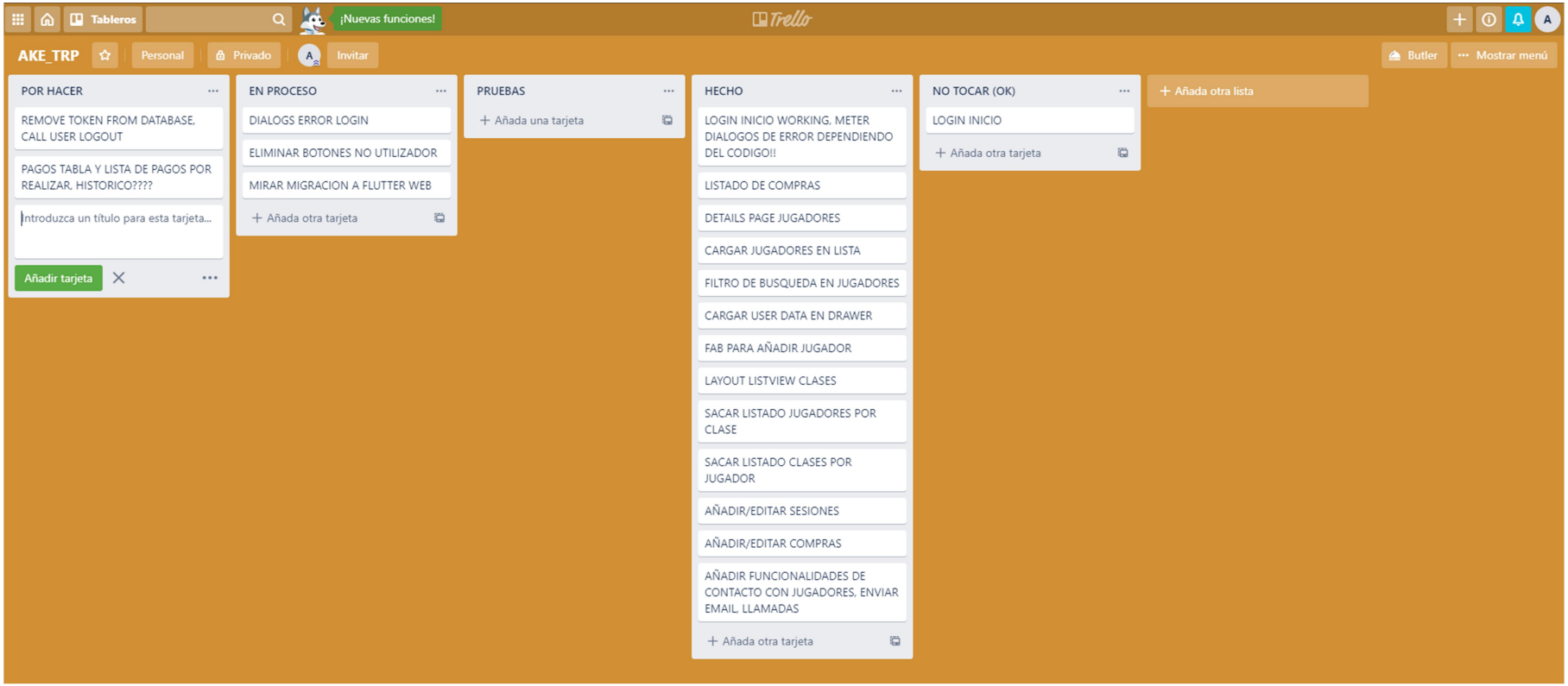
Given the fact that we are programming in a hybrid platform, more resources are required technologically to have everything covered. If the main goal is having the application in different platforms, we will need a device to be able to check that is working properly in each device. Luckily, we have emulators now, and by using a single computer, we could emulate other devices like Android phones. Ideally, it is much better to have a physical device instead of a virtual one, but if we are on a budget, it is an option still.

For the information to be available to all devices, mobile phones specially, the database must be located in a hosting cloud server. One thing to take into account in order to select the desired server is to know how much interaction will be between the devices and server. In this case, as maximum we could have between 3 and 10 devices connected at the same time to the server so it is not a great deal, we could get a cheap VPS server and store the database there. We also need the server to be able to respond to any API request, so for that it would

As the application right now is not in production state, the database is stored in the computer and we are using XAMPP for the database and web service. To test the application we are using Android Studio's emulator with an emulated Pixel 2 Android device.

7- Tracking evaluation protocol (TRELLO)

Tracking the tasks is one of the most essential things to do in order to complete each part of the work that comes with creating an application. There are many ways to keep track of those TODO's, starting from having a piece of paper writing them manually, until powerful desktop or web apps. Although we do not have a team involved and tasks don't have to be seen and completed by different people, it is necessary to have everything written in some way or another, because otherwise it would be terrorizing to manage everything and know what is done or it remains to be done. For that, Trello provided an easy to use and very powerful app for that.


Tableros
¡Nuevas funciones!
Trello
AKE_TRP
Personal
Privado
Invitar
Butler
Mostrar menú
POR HACER
REMOVE TOKEN FROM DATABASE, CALL USER LOGOUT
PAGOS TABLA Y LISTA DE PAGOS POR REALIZAR, HISTORICO????
Introduzca un título para esta tarjeta...
Añadir tarjeta
EN PROCESO
DIALOGS ERROR LOGIN
ELIMINAR BOTONES NO UTILIZADOR
MIRAR MIGRACION A FLUTTER WEB
Añada otra tarjeta
PRUEBAS
Añada una tarjeta
HECHO
LOGIN INICIO WORKING, METER DIALOGOS DE ERROR DEPENDIENDO DEL CODIGO!!
LISTADO DE COMPRAS
DETAILS PAGE JUGADORES
CARGAR JUGADORES EN LISTA
FILTRO DE BUSQUEDA EN JUGADORES
CARGAR USER DATA EN DRAWER
FAB PARA AÑADIR JUGADOR
LAYOUT LISTVIEW CLASES
SACAR LISTADO JUGADORES POR CLASE
SACAR LISTADO CLASES POR JUGADOR
AÑADIR/EDITAR SESIONES
AÑADIR/EDITAR COMPRAS
AÑADIR FUNCIONALIDADES DE CONTACTO CON JUGADORES, ENVIAR EMAIL, LLAMADAS
Añada otra tarjeta
NO TOCAR (OK)
LOGIN INICIO
Añada otra tarjeta
Añada otra lista

8- Software application analysis

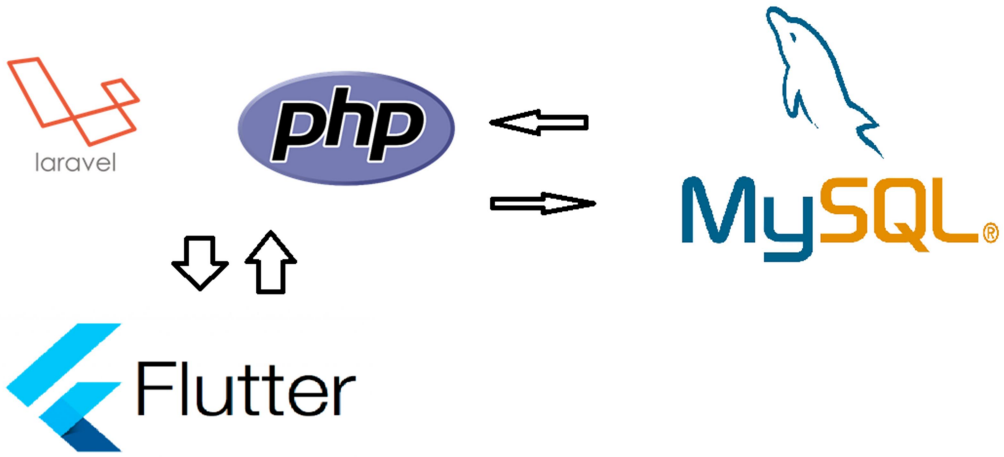
As I said in the introduction of this document, this program is a combination of different technologies, usually as it happens most of the time.

Flutter, even though is based on the Dart programming language and it is capable of recompiling the code to run in the platform we want, is still using some native components to enhance the performance on those devices.

For example, for Android OS components, the installation of Android Studio is required when programming and emulating the device, and there are some functionalities that need to be programmed directly on Android Studio, such as Splash Activities to show at the start of the app. To create the needed apk, Flutter uses the components installed in the Android Studio to be able to do it as well; it does not come as a functionality of the platform on its own.

With the back end, our goal is to create an API REST that will send the desired information back in JSON form for the Flutter app to receive. For that, we used PHP based Laravel platform, which once installed, it gives us the foundation for us to modify and create methods and API URL-s that will automatically executes certain functions.

There will be back and forth communications between front and back ends, but to improve the flow of the application I am trying to minimize the amount of requests made by the client. As a consequence of those low request rates and the use of JSON objects, data loads are fast and there are not a lot of load times when using the app normally.



8.1 Required installations and setting up the app to work

Setting up the environment could be the trickiest part of the whole programming process. I will list up the required apps to be installed.

The most important things to install locally are the web server and MySql database, which as a package can be downloaded with XAMPP. That will be the first thing we will have to install.

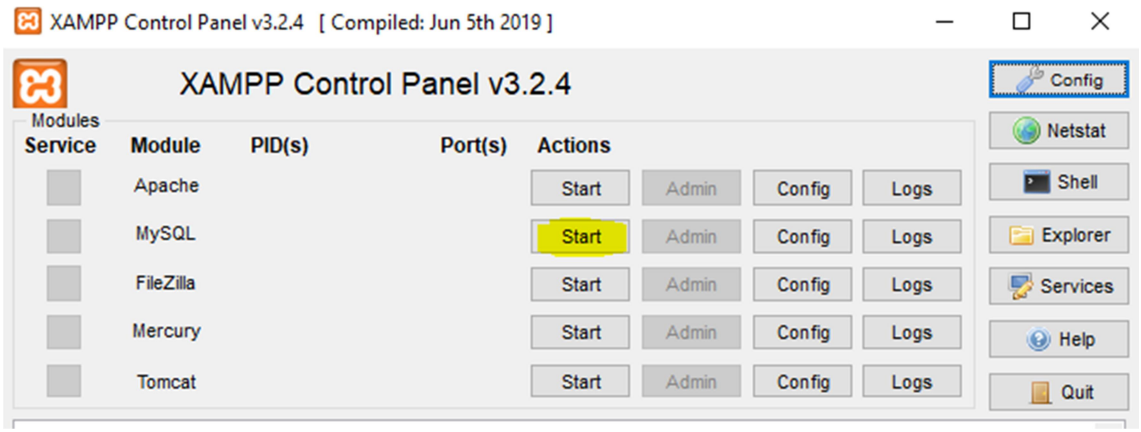
Furthermore, we will have to install different components for both front and back ends, so first we will start with the front part.

- Android Studio (https://developer.android.com/studio)
    - Android Studio needs to be installed in order to render and use functionalities for Android devices.
    - Once installed, we will have to install the Flutter plugin in the IDE.
    - Finally, create and install a new Emulated Device.
- Visual Studio Code (https://flutter.dev/docs/get-started/install/windows, https://flutter.dev/docs/get-started/editor?tab=vscode)
    - Install Flutter and Dart extensions on Visual Studio Code.
    - This will be the main programming IDE, once installed, we will have to get the Flutter SDK unzip it and copy the folder to the route you desire, update the PATH to use it from the CMD and run ***flutter doctor*** command. This command will tell us what we need for the app to run.
    - We can use the tutorial that is on the links above.

Now for the back end, in my case, I also used Visual Studio Code as the IDE so I did not install any other editor for it.

- PHPs latest release (https://www.php.net/downloads.php).
- Composer, PHP-s package manager to install Laravel.
- To install Laravel (https://laravel.com/docs/7.x/installation) once we have Composer installed, execute this command: ***composer global require laravel/installer***
    - To create a project with laravel:
        - ***composer create-project --prefer-dist laravel/laravel {NAME}***

To run the application, we need Laravel to be ready for responding the requests and we need Flutter to run the application on the device we need. For that, first of all, start MySQL server from XAMPP.

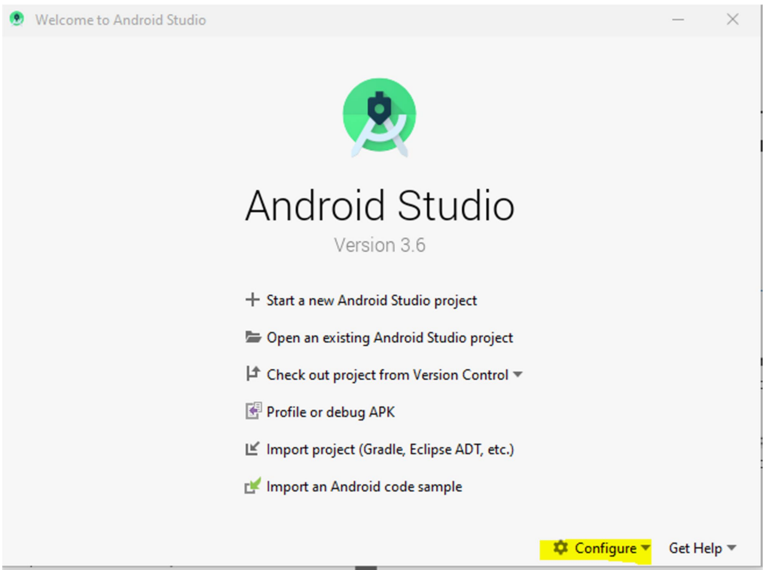


Once MySQL is running, open CMD and go to the route you have created the laravel project in, and execute this command: ***php artisan serve***

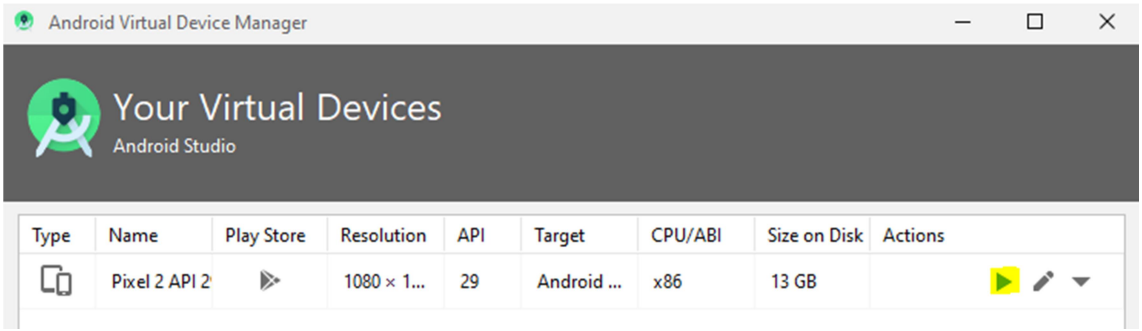
```
C:\Users\kinga>cd Documents\GitHub\TRPAitor\backend\aketrp_backend

C:\Users\kinga\Documents\GitHub\TRPAitor\backend\aketrp_backend>php artisan serve
Laravel development server started: http://127.0.0.1:8000
[Thu May 14 10:12:54 2020] PHP 7.4.2 Development Server (http://127.0.0.1:8000) started
```

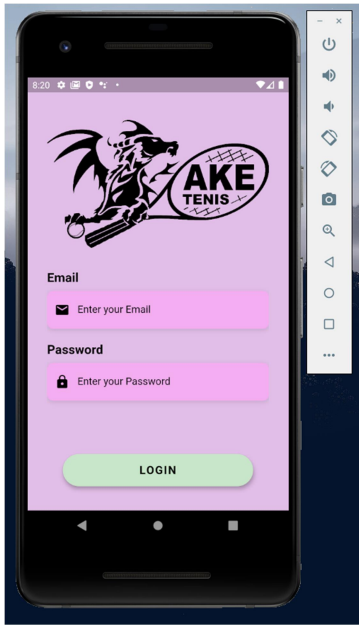
With that, the Laravel will be up and running, ready to answer any request given.

In addition, open Android Studio and run the emulated device in the background, first by clicking on Configure-> AVD Manager.



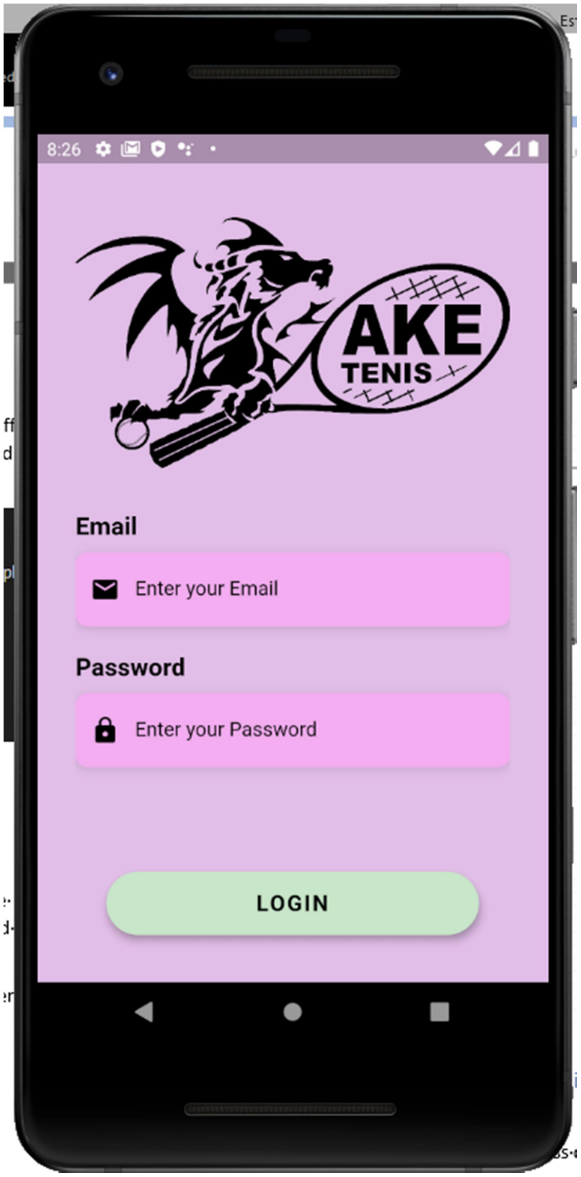
Once the list of devices pops up, click run on the desired device and it will start running.





Finally, open the Flutter project in Visual Studio Code, and open the terminal the IDE offers. Then, as the device is connected and everything is set up, execute ***flutter run*** command. This will serve the application on the device that we already have running.

```
PS C:\Users\kinga\Documents\GitHub\TRPAitor> cd .\ake_trp\
PS C:\Users\kinga\Documents\GitHub\TRPAitor\ake_trp> flutter run
Using hardware rendering with device Android SDK built for x86. If you get graphics ar
Launching lib\main.dart on Android SDK built for x86 in debug mode...
Running Gradle task 'assembleDebug'...
Running Gradle task 'assembleDebug'... Done                        23,4s
√ Built build\app\outputs\apk\debug\app-debug.apk.
Installing build\app\outputs\apk\app.apk...                         2,0s
D/FlutterActivity(28828): Using the launch theme as normal theme.
D/FlutterActivityAndFragmentDelegate(28828): Setting up FlutterEngine.
```



9- Application design

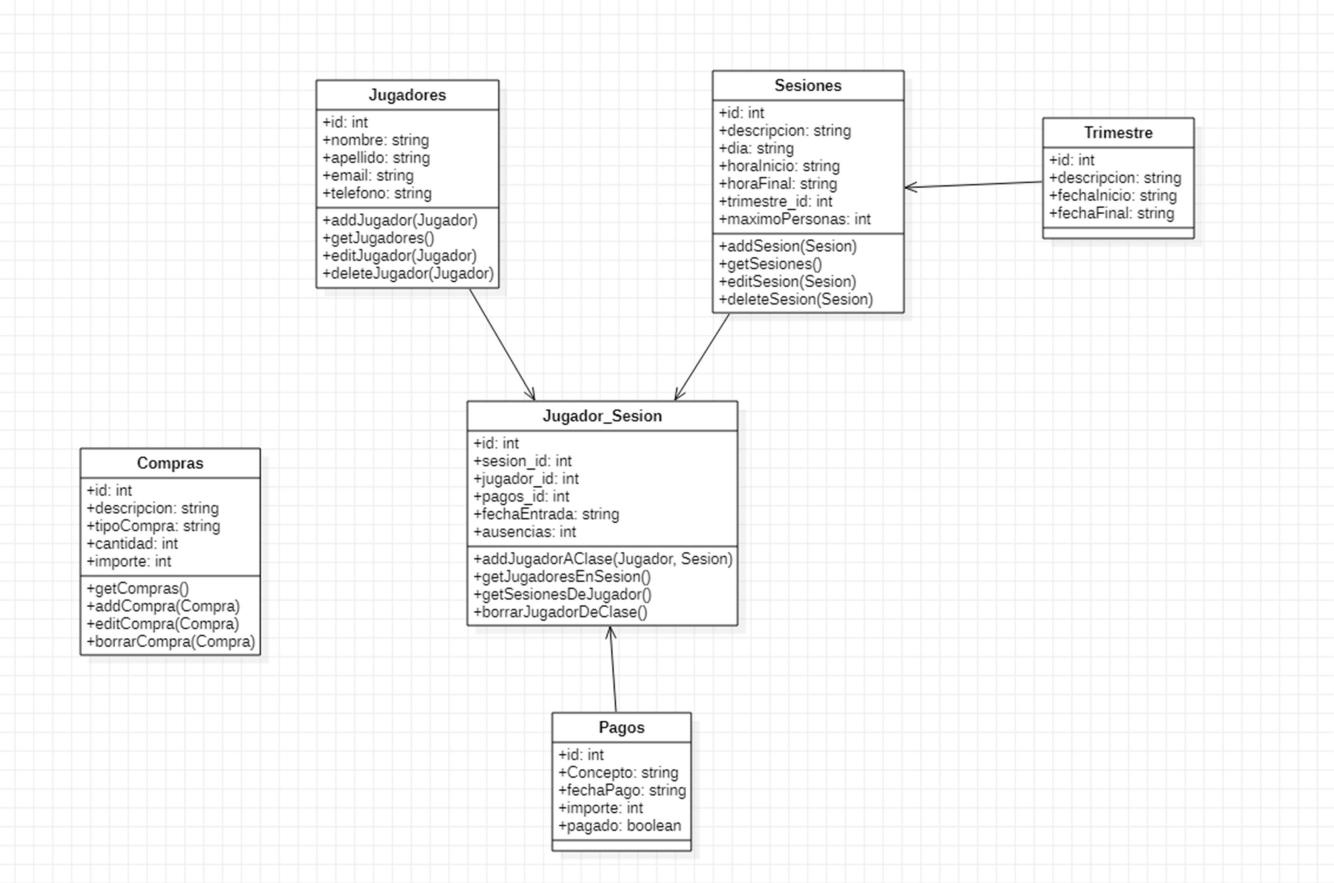
Once the main idea of the application was clear, the first ideas came up about how the application will look like, how it will be put together, what information should be saved and many more things.

In this part, I will explain how the applications database, class diagrams and layouts were designed and thought at first, and how they were changed after it was being programmed.

9.1 Class diagram

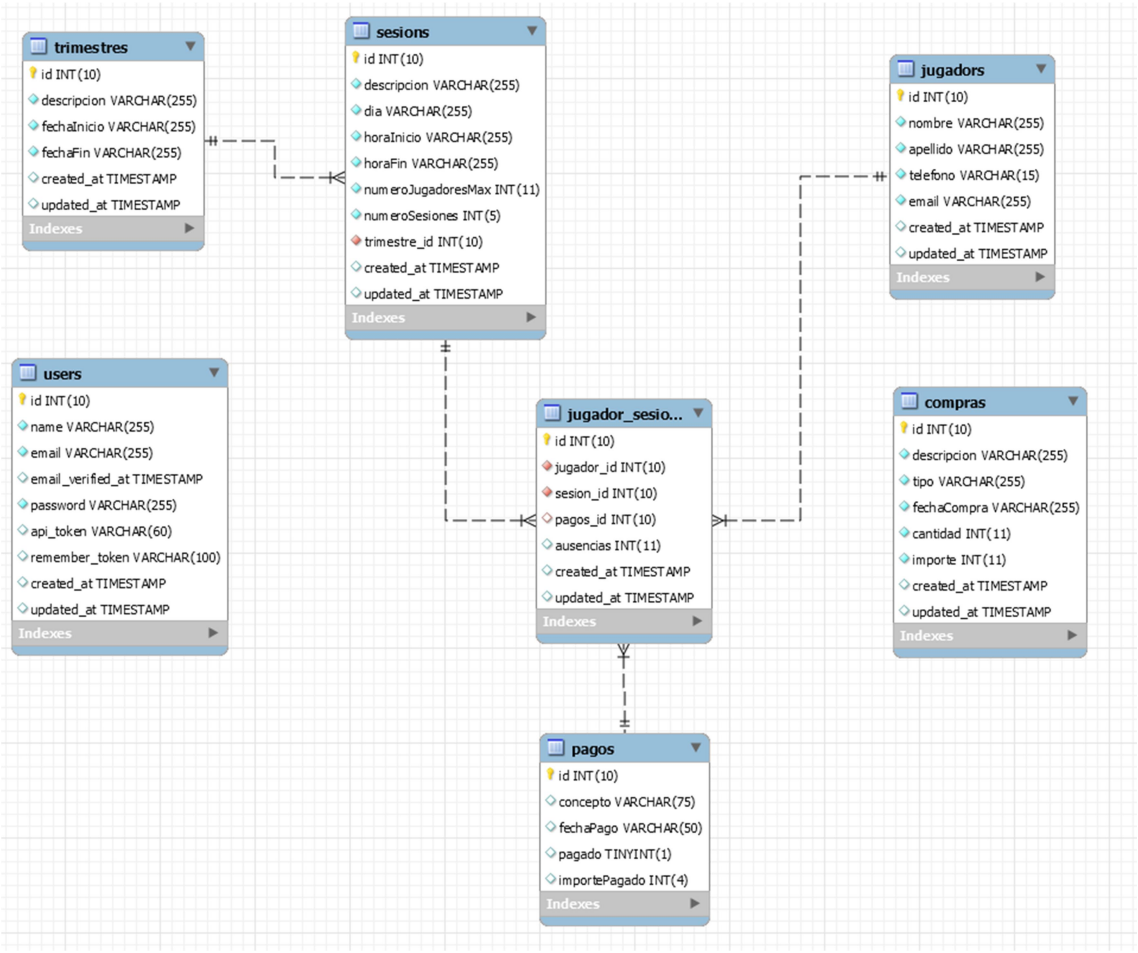
With the first class diagram, I wanted to make a list of the attributes I wanted to store for every object or class we are going to have in the application. Apart from that, I also made an initial listing of the method each classes will have.

The methods for each class cover the basic operations with objects, such as create, edit, delete and search. Apart from that, as making operations with the relational table (Jugadores_sesion) was going to be different, I listed also the possible methods that I will need for adding a player to a new session or removing him, or add a new absence to a player if he misses a session.



9.2 – Database diagram

Once the first class diagram was clear, I started to create the first database on the platform, MySQL. All the first tables had been created with their attributes with the needed relations and constraints. This was the first diagram of that database:



Due to the use of laravel as a back end provider, it created some new tables to the database, since it needs those tables to manage the logging part of the application and many other functionalities also.

As an example, the back end provider creates the User table with some specific attributes to manage API_TOKEN storage and remembering. That does not mean that we cannot make any changes to it, so I added the name attribute as an identifier that will appear in the layout of the final app.

In conclusion, there were not many changes made to the database from the first version of it, some attributes were added and removed from it, but not much more. This is not a large application with a lot of tables and relations, so it was easy to make modifications to it.

9.3 – Layout design and Colour Palettes

When watching videos about the platform for this application (Flutter), the way of programming the user interface was very beautiful and easy to the eyes. It uses native design elements such as material design icons and layouts for Android and Cuppertino icons for iOS, so that makes it appear as if it's a completely native programmed app.

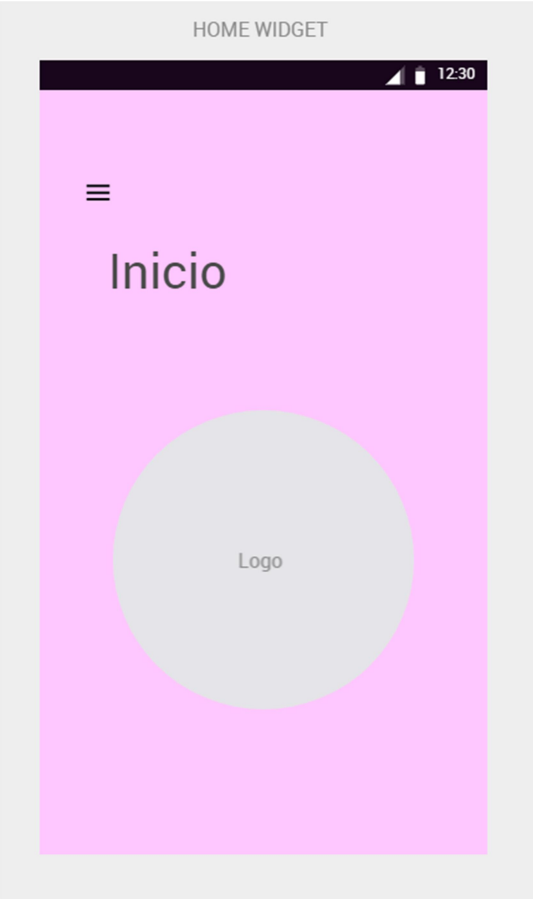
Alongside that, Flutter provides specific code for elements in every platform, which allows the programmer to change icons depending on the platform and many other things.

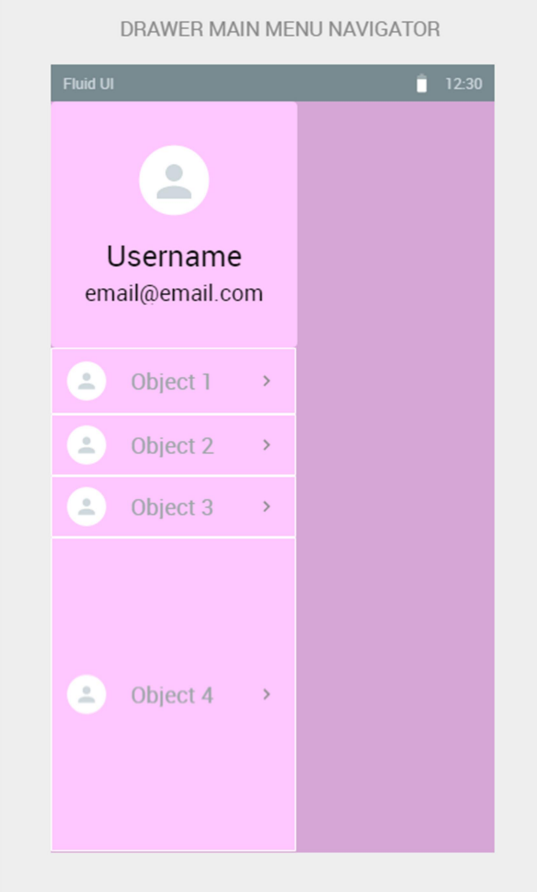
Although the application can be served in many platforms, I focused more in the Android part this time around, so the first layout designs are oriented to mobile platforms.

For the first designs, I used a web app called Fluid UI (https://www.fluidui.com/), which allows to create different layouts for different platforms. In our case, we selected the mobile application platform and it provided us with different elements of the OS.

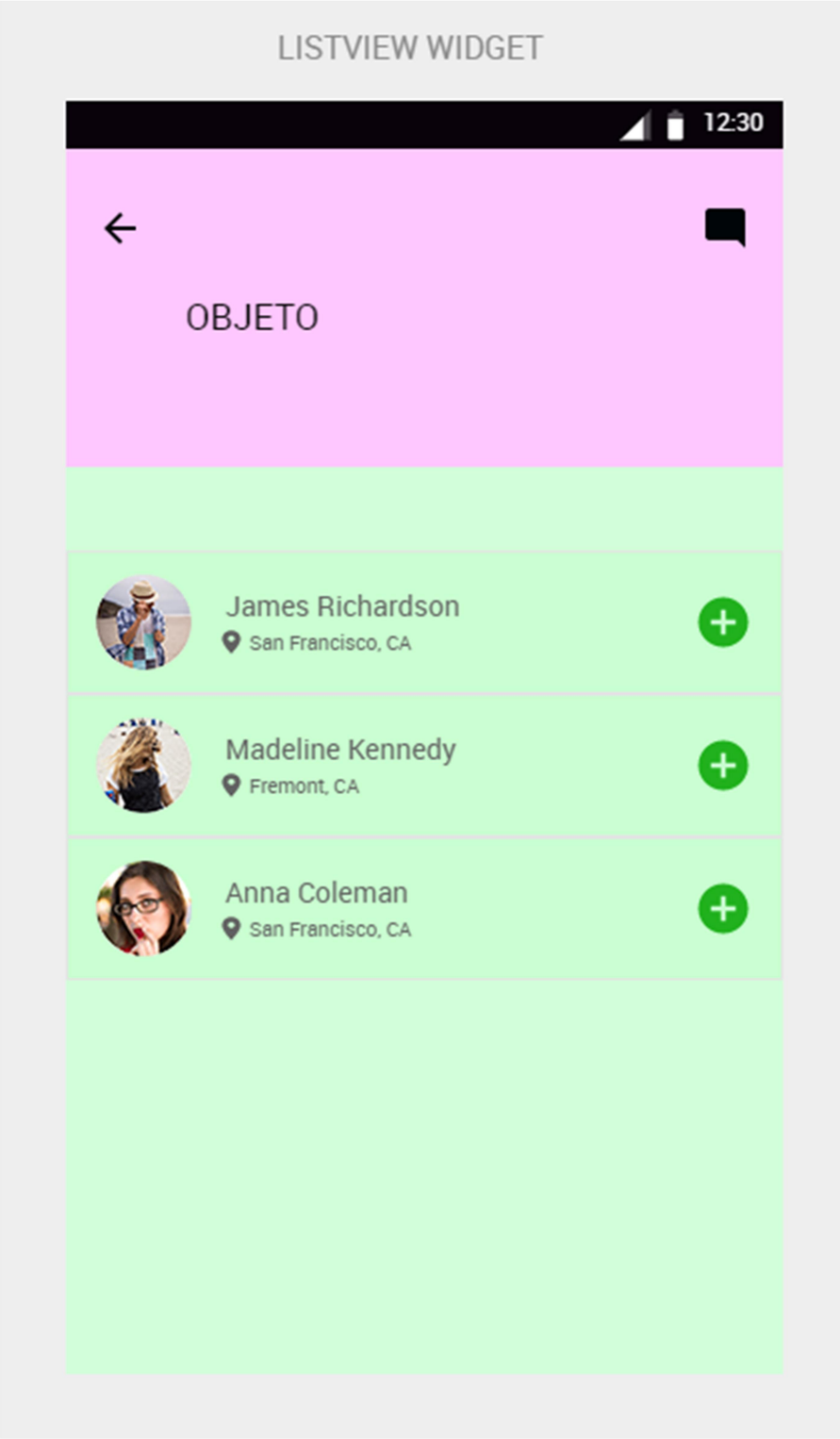
Here are the first layouts designs of the project:

- Home and navigator drawers

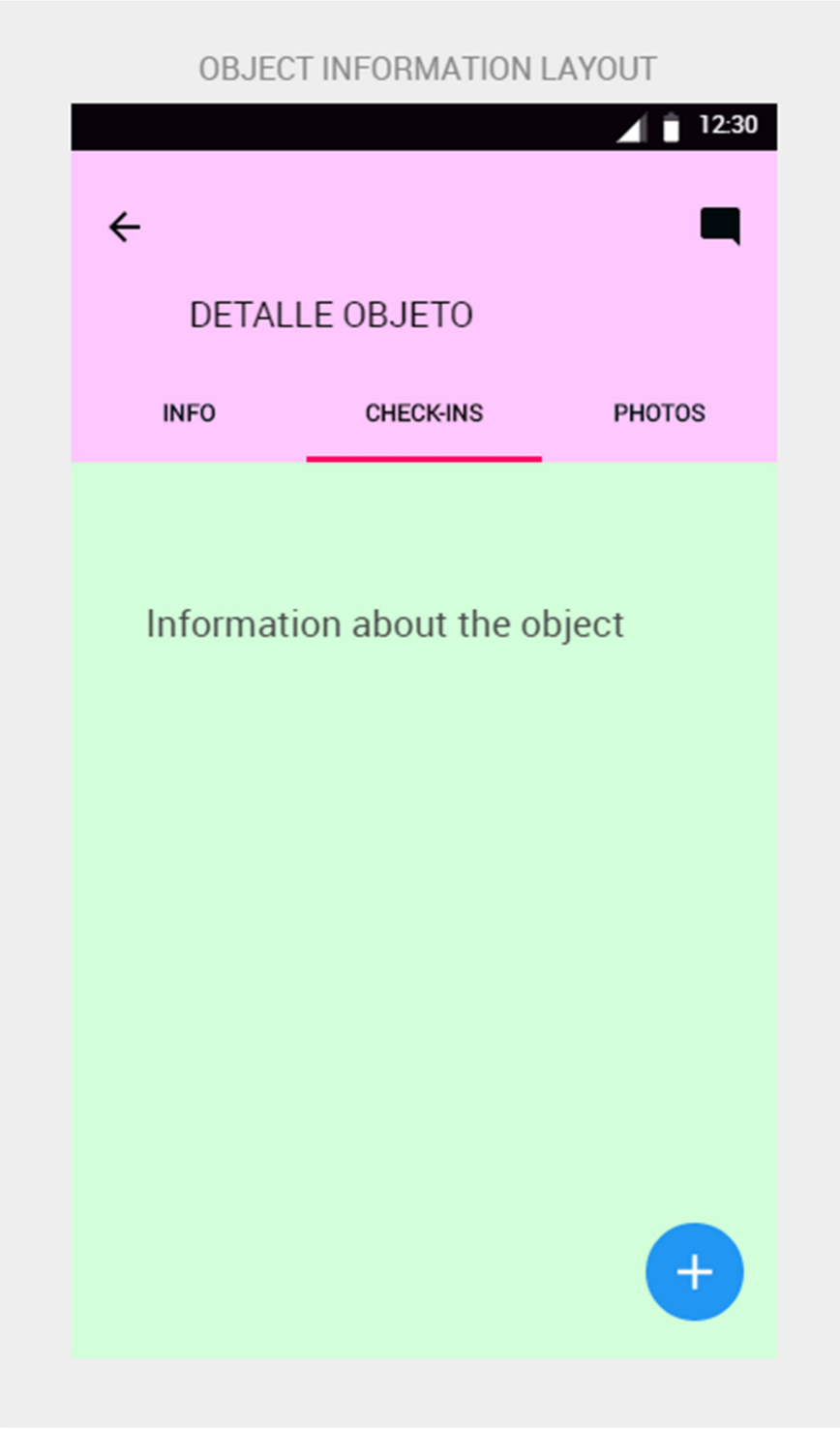




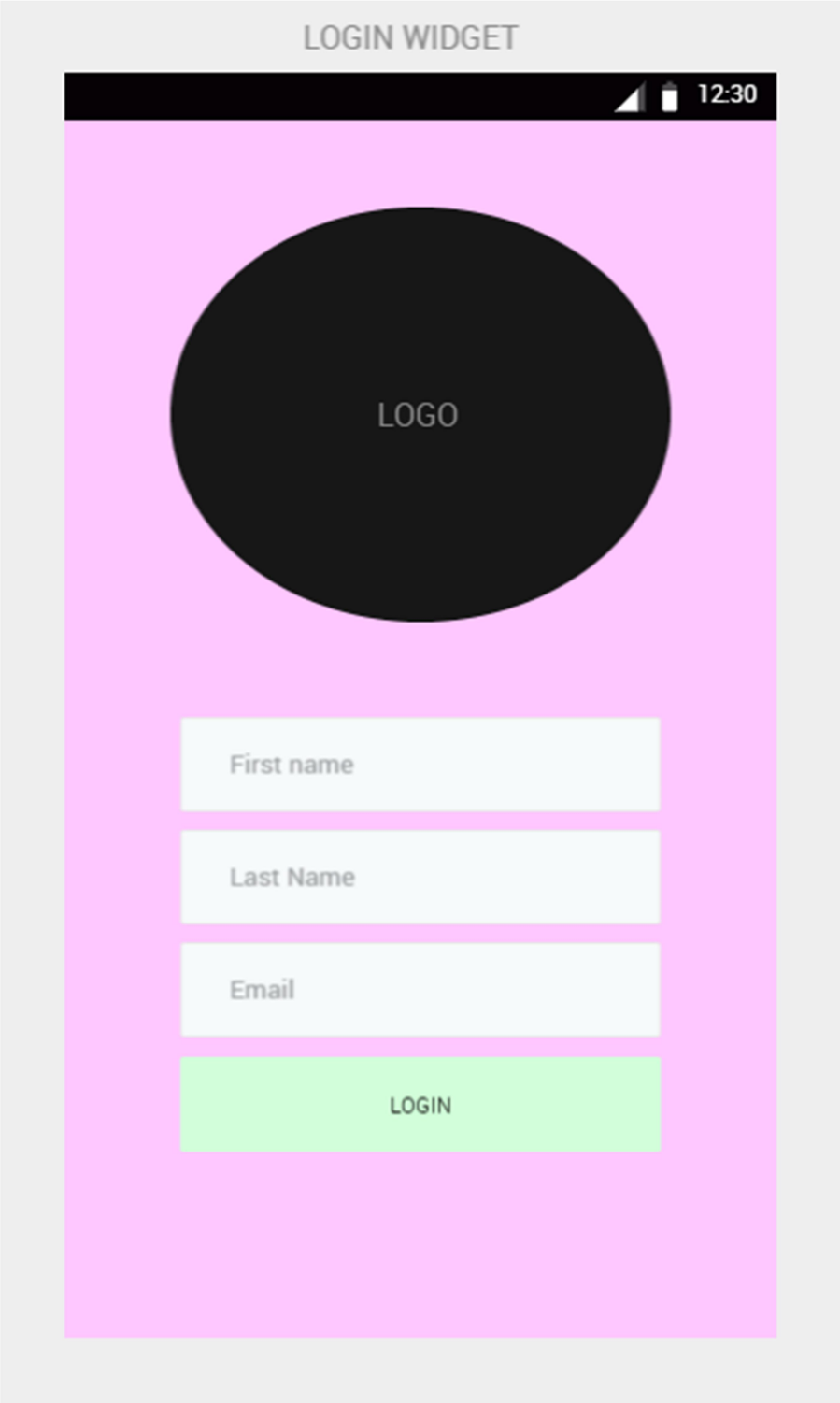
- Layouts for showing lists:



- Object detail layout



- Login layout:



The colour pallet used for layouts and elements in the UI is linked to the tennis courts in the club. As the primary colour, I am using a very light pink colour (CODE), which is the same colour as the outer part of the court. For the secondary colour, I used the inner colour of the court, a light green (CODE), with the intention of creating a light contrast.

10- Code and final app explanations

Here I will explain some of the applications most important pieces of code, dividing in by back-end and front-end code. I will show the most used functions and how I link both ends.

I am not showing the full code, I will show how the application executes certain code in every layout, and how we call and receive the data we need to show.

10.1 Laravel serving routes and redirections

First of all, a quick explanation of what Laravel is doing in this situation. When creating the project, along with a lot of other files, Laravel creates a file called ***api.php*** where we have to add or modify the URL routes we want the server to be responding to.

To classify every request made, we divided the URLs in groups depending the type of object we want to receive. Then, we created a Controller for each object that will manage everything that we need for that object, inside those controllers, there will be methods executing certain pieces of code. Here is an example of an ***api.php*** group and the URLs, where we link the URL and the Controllers to execute the code we want.

The Android Studio emulated device uses a different localhost IP direction to the one that computers use, so we needed to change the direction to where the app will make the request.

```
http://10.0.2.2:8000/api/
```

api.php

```
Route::group([
    'prefix' => 'Jugadores',
    'middleware' => 'auth:api'
], function() {
    Route::get('lista', 'JugadorController@index');
    Route::get('/{id}', 'JugadorController@show');
    Route::post('new', 'JugadorController@store');
    Route::put('edit/{id}', 'JugadorController@update');
    Route::delete('delete/{id}', 'JugadorController@update');


});
```

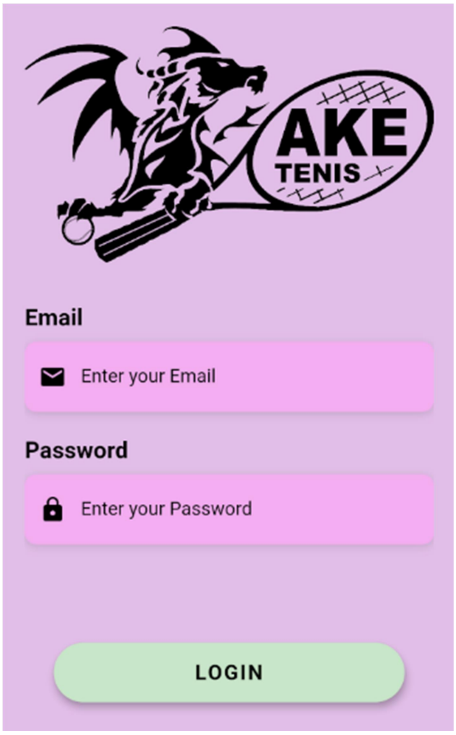
Alongside the front end code, we will show the code that executes in the back end as well, to encapsulate almost everything when explaining what the app does in each widget.

To finish, for managing the Login and Authentication of the users, we are using Laravels **Passport** plugin, which takes care of the API_TOKEN creations and storing them in the database, as well as other many things.

With that set, let's get started with the front end.

10.2 Landing/Login Widget

This will be the first window we will see in the application, which will check is we are already logged into the application or we need to log in first. If we need to login, this form will appear, for us to enter the access credentials.



When entering the correct credentials and hitting the login button, the app will send a request with the two strings (email and password) and if the response is correct, Laravel will generate an API_TOKEN that will be stored both in the database and in the device via Shared Preference. Then, it will move on to the Home window.

landingWidget.dart

```
child: RaisedButton(
  elevation: 5.0,
  onPressed: () {
        final String emailOut = emailController.text;
        final String passOut = passwordController.text;
        UserMethods.login(emailOut, passOut, context);

          //Navigator.push(context, MaterialPageRoute(builder: (context) => HomeWidget()));
          //print('Login Button Pressed');

  } ,
```

userMethods.dart

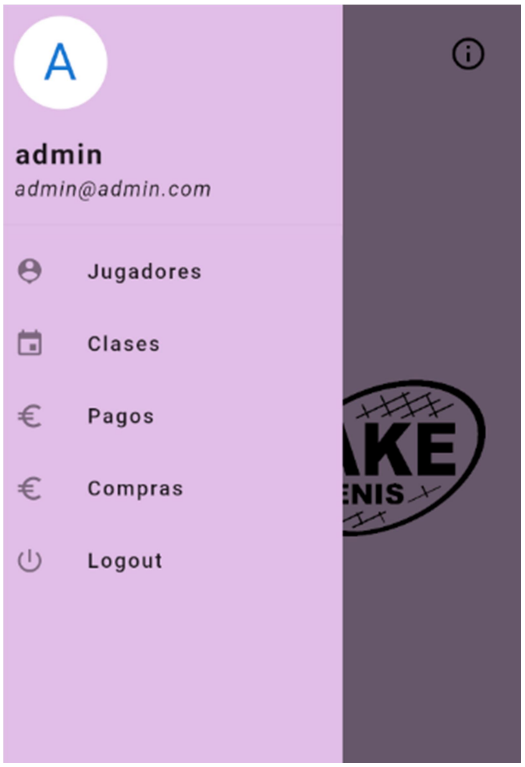
```dart
static login(String email, String pass, BuildContext context) async {
    Map data = {
      'email': email,
      'password': pass
    };
    var jsonData = null;
    sharedPreferences = await SharedPreferences.getInstance();
    var response = await http.post(apiRoute + 'auth/login', body: data);
    if (response.statusCode == 200){
      //EL SERVIDOR RESPONDE OK
      jsonData = json.decode(response.body);
      sharedPreferences.setString('access_token', jsonData['access_token']);

      Navigator.of(context).pushAndRemoveUntil(
                  MaterialPageRoute(
                  builder:(BuildContext context) => HomeWidget()), (Route<dynamic> route) => false);
      //print(jsonData['access_token']);

    }else{
      print(response.statusCode);
    }
}
```

AuthController.dart

```php
public function login(Request $request)
{
    $request->validate([
        'email'       => 'required|string|email',
        'password'    => 'required|string',
        'remember_me' => 'boolean',
    ]);
    $credentials = request(['email', 'password']);
    if (!Auth::attempt($credentials)) {
        return response()->json([
            'message' => 'Unauthorized'], 401);
    }
    $user = $request->user();
    $tokenResult = $user->createToken('Personal Access Token');
    $token = $tokenResult->token;
    if ($request->remember_me) {
        $token->expires_at = Carbon::now()->addWeeks(1);
    }
    $token->save();
    return response()->json([
        'access_token' => $tokenResult->accessToken,
        'token_type'   => 'Bearer',
        'expires_at'   => Carbon::parse(
            $tokenResult->token->expires_at)
                ->toDateTimeString(),
    ]);
}
```

10.3 HomeWidget

This will be the main navigation window, where we will be able to navigate to any page we need. The navigation in this application is lineal, and this activity is the main anchor that enables this. In this widget, we use a Navigation Drawer that acts as a menu, with the options to manage every object in the app.



When this window loads, it executes a method that checks if a user is already logged into the app, by checking if there is an access token stored on the device. If there is no token, it automatically redirects us to the login page for us to enter credentials again. By default, when we close the application, the token stays stored until a period of time passes, so if we close the app and enter again, we will not be redirected to the initial login screen.

```
@override
void initState(){
  super.initState();
  UserMethods.checkIfLogin(context);
  userData = UserMethods.getUser();
  //print(userData.toString());

}
```

Each one of the options in the navigation drawer is responsible of loading the desired screen, which will be a screen with a list of the objects that we selected. This is the executed code to load up the Jugadores list screen:

```
Column(
  children: <Widget>[
    Container(

      //padding: EdgeInsets.zero,
      child: ListTile(
        leading: Icon(Icons.person_pin),
        title: Text('Jugadores', style: TextStyle(fontSize: 16.0, letterSpacing: 1.5),),
        onTap: () {
          // Update the state of the app
        Navigator.push(context, MaterialPageRoute(builder: (context) => JugadoresWidget()));
          // Then close the drawer
          //Navigator.pop(context);
        },
      ), // ListTile
    ), // Container
  ], // <Widget>[]
), // Column
```

In the case of the Logout option, when clicked, the stored token is removed from the device and it redirects us to the login page.

```
Container(
  //color: primaryMorado,
  child: Column(
    mainAxisAlignment: MainAxisAlignment.spaceAround,
    children: <Widget>[
      ListTile(
        leading: Icon(Icons.power_settings_new),
        title: Text('Logout', style: TextStyle(fontSize: 16.0, letterSpacing: 1.5),),
        onTap: () async {
          UserMethods.logout(context);
          // ...
          // Then close the drawer

        },

      ), // ListTile
    ], // <Widget>[]
  ), // Column
), // Container
```

```
static logout(BuildContext context) async{
    SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
    sharedPreferences.clear();

    Navigator.of(context).pushAndRemoveUntil(
                MaterialPageRoute(
                  builder: (BuildContext context) => LandingWidget()), (Route<dynamic> route) => false);
}
```

10.4 Listview Widget

Here I will explain how the screens that have the lists of objects work. This widgets will be the ones that will show the players, sessions, purchases and payments (not available) stored in the

database in a list. Therefore, as most of them are doing the same operations, I will only explain how the players listview works and also the one that shows the sessions.

When selecting the option Jugadores form the shown navigation drawer, the app will start loading the new window. As we do with the Home Screen, when loading the screen we execute some functions in order to retrieve the information from the database.

These methods are the one responsible to make the requests and store the objects in the needed variables, in order for us to create the objects that are going to appear in those list.

JugadoresWidget.dart

```
@override
 void initState(){
   super.initState();
   llamadaJugadores();

   //jugadores = PlayerMethods.getJugadoresList();
 }
```

```
llamadaJugadores(){
  jugadoresList = PlayerMethods.getJugadores().then((players){
      setState(() {
        jugadoresParsed = players;
        jugadoresFiltered = players;
      });

    });
}
```

playerMethods.dart

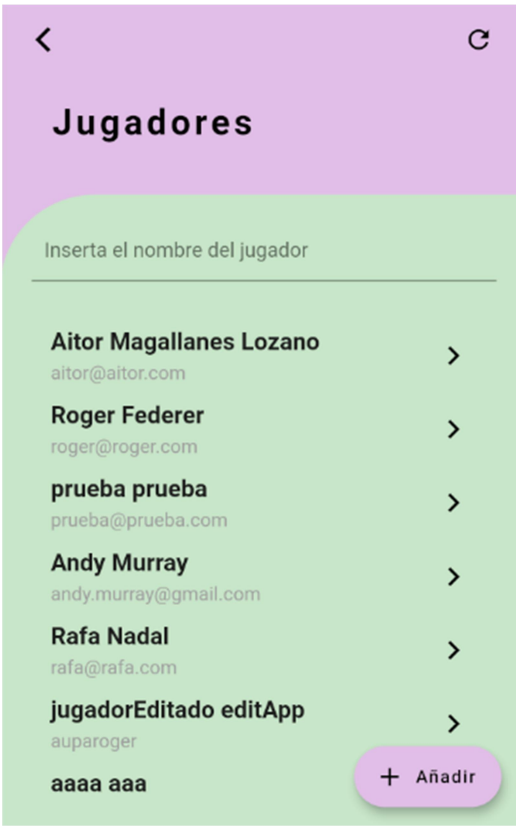
```
static Future<List<Jugador>> getJugadores() async{

sharedPreferences = await SharedPreferences.getInstance();
  final response =
      await http.get(apiRoute + 'Jugadores/lista',
      headers: {HttpHeaders.contentTypeHeader: "application/json",
      HttpHeaders.authorizationHeader: "Bearer "+ sharedPreferences.getString('access_token')},);

  if (response.statusCode == 200) {

    // If the server did return a 200 OK response, then parse the JSON.
      List<Jugador> players = [];
      var jsonData = jsonDecode(response.body);
      for (var u in jsonData){

      players.add(Jugador.fromJson(u));
      }
      return players;

    } else {
      // If the server did not return a 200 OK response, then throw an exception.
      throw Exception('Failed to load Jugador');

    }

    }
```

JugadorController.php – Returns list of players in JSON

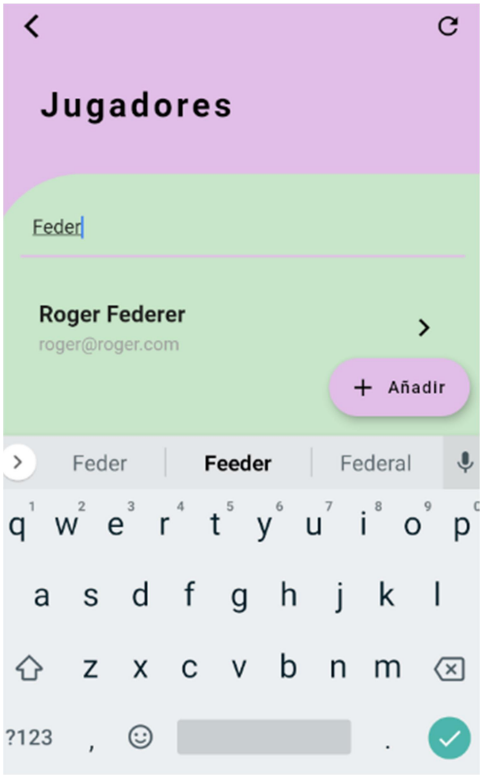
```
public function index()
{
    $jugadores = Jugador::get();
    return $jugadores;
    //
}
```

Once the variables are filled with objects, we pass that information to the list builder as a parameter and it shows the object on the screen.

```
Padding(
    padding: EdgeInsets.only(top: 25.0),
    child: Container(
        height: MediaQuery.of(context).size.height - 300.0,
        child: ListView.builder(
                itemCount: jugadoresParsed.length,
                itemBuilder: (context, int index){
                  //Jugador newPlayer = jugadoresParsed[index];
                  //jugadoresParsed.add(newPlayer);
                  //jugadoresFiltered.add(newPlayer);
                  return _buildJugadorItem(context, jugadoresParsed[index]);
                })  // ListView.builder

            ,) // Container
        ), // Padding
```



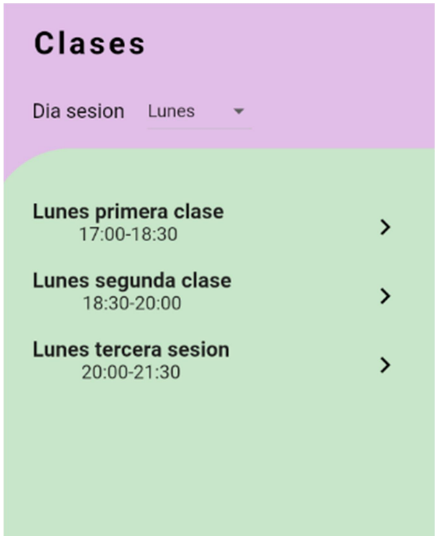
To classify the objects by one of its attributes, we apply filtering methods that are executed in real time. In the case of players, we use a text field that functions as a search bar, where we get the text that is written on it and compare it to the items in the list, showing the ones that match.



```
TextField(
  decoration: InputDecoration(
    contentPadding: EdgeInsets.all(10.0),
    hintText: 'Inserta el nombre del jugador',
  ), // InputDecoration
  onChanged: (stringName){
    setState(() {
      jugadoresParsed = jugadoresFiltered
                        .where((u) => (u.nombre.toLowerCase()
                                       .contains(stringName.toLowerCase())
                                       || u.apellido.toLowerCase()
                                       .contains(stringName.toLowerCase()))).toList();
  });

  },
```

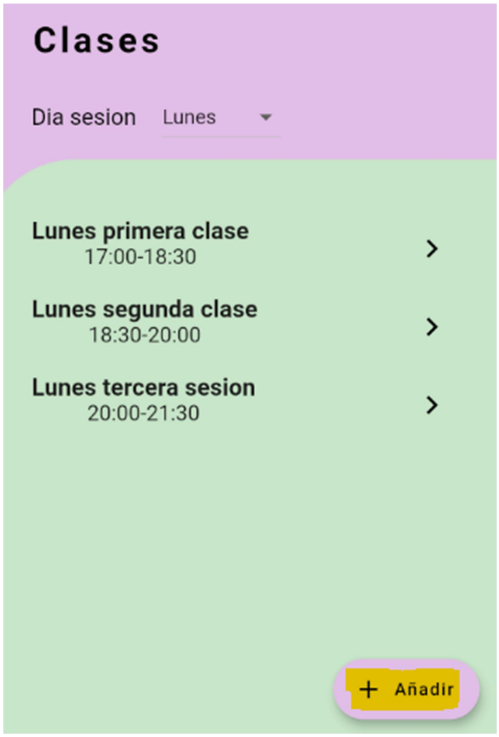
If we add a new object to the database, the records **do not** automatically refresh on screen and show the latest one that was added. As a workaround, I have included a refresh button on the top right of the screen, which updates the list. It could have been better to have included some type of pull to refresh option, but I had rendering problems when including it, so I left it out of this version.

Another filtering method I have used is included on the screen that shows the list of sessions that are offered at the moment. In this method, I use a Dropdown Button with the days of the week, filtering the results based on the selected day of the week. Here, I capture the event that triggers when a value is changed on the dropdown button and then execute the request to the database.



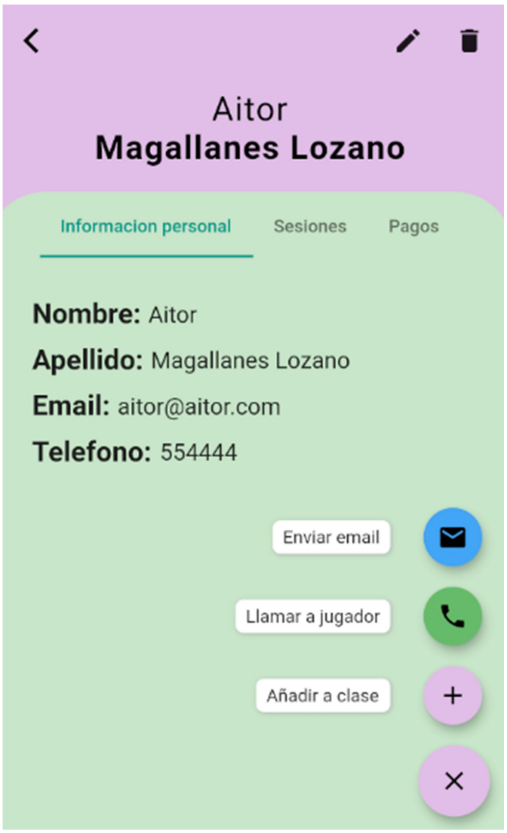
By tapping into one of those items, we will access to the information of that object, opening a new Detail Screen that will appear later in the document.

To finish with these screens, I also added a Floating Action Button to the bottom right of the screen that opens a new screen for adding a new object to the database. That button is the same in every other widget of this type. We will explain the screen for adding objects later on in the document.

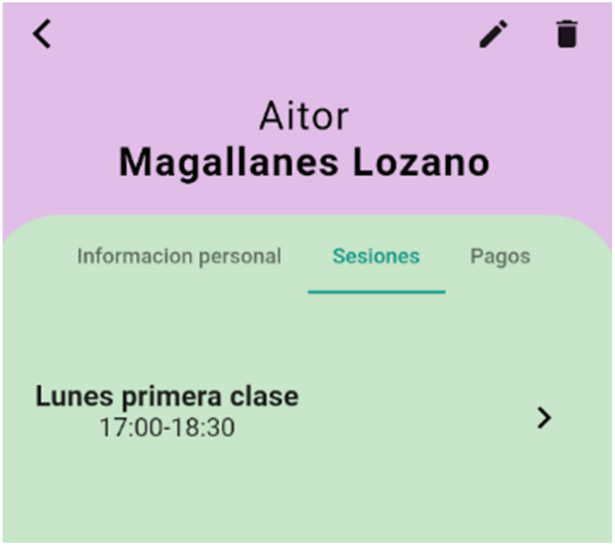


10.5 Detail Screens

As I wrote earlier, when selecting one of the objects of the list appeared on screen, we will enter a detail screen of the selected object. In this screen, we could see the main attributes of the objects as well as the information of the item that is related to. Here is an example of a player items detail screen:



To be able to see different information about the object, a tab bar was used. If the object that we want to see has any other related table in the database, we retrieve that information with a new request and then show it in another tab. As a result of that, when navigating through information about the players for example, we could see if that player is assigned to any class, and we could also see the payments done by that player (not included in this version). In addition, all the information about the related object is available as well if we tap it again, so we could go from watching a players information to see the payment details.



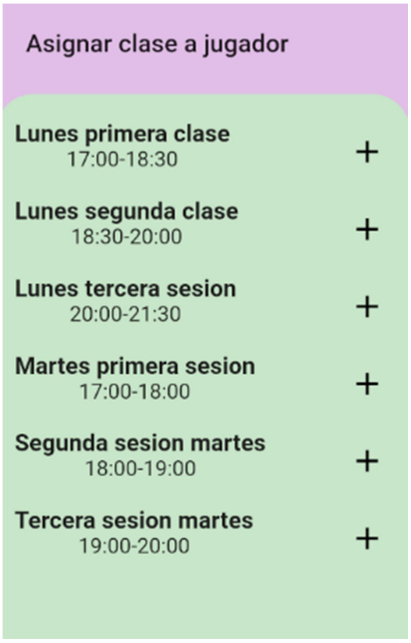
JugadoresDetail.dart

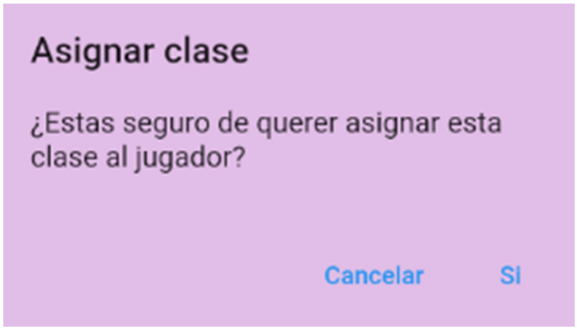
```
void initState() {
  // TODO: implement initState
  _tabController = new TabController(length: 3, vsync: this);
  //print(widget.jugadorIn.jugador_id);
  ClasesMethods.getSesionesJugador(widget.jugadorIn.jugador_id).then((valores){
    setState(() {
      sesionesJugador = valores;
    });

    print(sesionesJugador.length);
  });

  ClasesMethods.getSesiones().then((sesiones){
    setState(() {
      sesionesAElegir=sesiones;
    });

  });

  super.initState();

}
```

JugadorSesionController.php ->Returns list of sessions in which the player takes part

```
public function getSesionesJugador(Request $request )
{
    $listaSesiones = array();
    $idSesionesJugador = jugador_sesion::where('jugador_id', $request->jugadorId)->get();

    foreach ($idSesionesJugador as $registro) {
        $sesionIn = Sesion::where('id', $registro->sesion_id)->get();
        array_push($listaSesiones, $sesionIn[0]);
    }

    return $listaSesiones;
    //
}
```

In addition to being able to see information about the father or child of the object, we can algo assign one object top another. For example, we can add a player to a new class, if we click in the button that is in the bottom right corner of the screen, various options will show up one of those being ***Añadir a clase***. By clicking in that button, a new Dialog will show up with the list of available classes, to assign it, simply tap again and confirm. The same applies to the sessions class, in which we could add and remove players from a session.



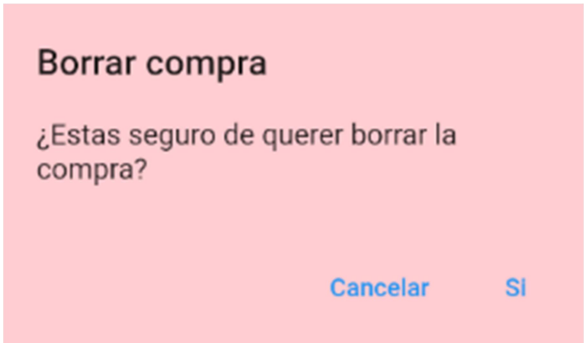


JugadorSesionController.php -> Assign player to a class

```php
public function addSesionToPlayer(Request $request){

    $existe = jugador_sesion::where('jugador_id', $request->jugadorId)->where('sesion_id', $request->sesionId)->first();

    if(is_null($existe)){
        $newJugador = new jugador_sesion();
        $newJugador->jugador_id = $request->jugadorId;
        $newJugador->sesion_id = $request->sesionId;
        $newJugador->save();
        return response()->json($newJugador);
    }else{
        return response('El jugador ya esta en esa clase', 202)->header('Content-Type', 'text/plain');
    }



}
```

Finally, as the last operations that can be done with an object, there are two buttons on the top right corner of the widget that serve as an edit and delete triggers.

If we tap on the pencil, the screen for adding an object will show up on screen with the data from the object, only their attributes, we cannot edit any information from the related objects. We reuse the same layout that is used on the adding object layout that I will show after this. If he hit the trash can icon however, a small dialog shows up for us to confirm the action, which will trigger a delete code that will remove the object from the database. Here is shows how is done in the purchases:



ComprasDetail.dart

```
builder: (BuildContext context){
  return AlertDialog(
    backgroundColor: Colors.red[100],
    title: Text("Borrar compra"),
    content: SingleChildScrollView(
      child: ListBody(
        children: <Widget>[
          Text("¿Estas seguro de querer borrar la compra?")
        ], // <Widget>[]
      ) // ListBody
    ), // SingleChildScrollView
    actions: <Widget>[
      FlatButton(
        onPressed: () => Navigator.pop(context),
        child: Text("Cancelar"),
      ), // FlatButton
      FlatButton(
        onPressed: (){
          ComprasMethods.borrarCompra(widget.compraIn, context);
          //Navigator.pop(context);
          },
        child: Text("Si"),
      ) // FlatButton
```

comprasMethods.dart

```
static borrarCompra(Compras compraIn, BuildContext context) async {
  print(compraIn.id);
sharedPreferences = await SharedPreferences.getInstance();
var response = await http.delete(apiRoute + 'Compras/delete/'+compraIn.id.toString(),
              headers:
              {
              HttpHeaders.contentTypeHeader: "application/json",
              HttpHeaders.authorizationHeader: "Bearer "+ sharedPreferences.getString('access_token')
              },
               );
if (response.statusCode == 200){

  dialogs.information(context, "Compra borrada", "Compra borrada con exito!");
  //Navigator.pop(context);
  print("Compra borrada con exito");

}else{
  dialogs.error(context, "Error al borrar", "Error al borrar compra");
  print(response.statusCode);

}

}
```

ComprasController.php

```
public function destroy($id)
{
    $compraIn = Compras::findorfail($id);
    $compraIn->delete();
}
```

10.6 Adding object widget

To conclude with the screens, we will take a look at the widget for adding a new object to the database. These screens share the same layout with input fields and buttons, the only thing that is different is the text field quantity, depending on the object there will be more or less fields to fill up. In addition to these input fields, we have also used date and time pickers for sessions and purchases, to select the purchase day and initial and final hours for the sessions.

There are two options in this layout, confirm the form that we want to send, or reset all the inputs, green button to confirm, red button to reset.

This layout is not only used for adding the objects, we reuse the same layout for editing the object as well, by changing the title and the code executed when pressing the confirm button.

To differentiate between the two screens, we modified the constructor of the layout to receive two parameters, one for the object that is going to appear in case of editing, and a Boolean that confirms if is called for editing or not. With that second parameter we can call the code for editing or creating the object.

AddJugadorWidget.dart

```dart
@override
void initState() {
  if (widget.editar){
    nombreController.text = widget.jugadorAEditar.nombre;
    apellidoController.text = widget.jugadorAEditar.apellido;
    telefonoController.text = widget.jugadorAEditar.telefono;
    emailController.text = widget.jugadorAEditar.email;
  }else{
    widget.editar = false;
    widget.jugadorAEditar = null;
  }
  super.initState();
}
```

```dart
MaterialButton(
  onPressed: () {
    var form = formKey.currentState;
    int responseStatus;
    if (form.validate()){
      if(widget.editar){
        widget.jugadorAEditar.nombre =  nombreController.text;
        widget.jugadorAEditar.apellido =  apellidoController.text;
        widget.jugadorAEditar.telefono = telefonoController.text;
        widget.jugadorAEditar.email = emailController.text;
        PlayerMethods.editarJugador(widget.jugadorAEditar, context);
      }else{
        Jugador jugadorAGuardar = new Jugador();
        jugadorAGuardar.nombre = nombreController.text;
        jugadorAGuardar.apellido = apellidoController.text;
        jugadorAGuardar.telefono = telefonoController.text;
        jugadorAGuardar.email = emailController.text;
        PlayerMethods.guardarJugador(jugadorAGuardar, context);
      }

      nombreController.clear();
      apellidoController.clear();
      telefonoController.clear();
      emailController.clear();
      Navigator.pop(context);
    }
```

playerMethods.dart ->code that makes the API call to create or edit

```
static guardarJugador(Jugador jugadorIn, BuildContext context) async {
    Map data = {
          'nombre': jugadorIn.nombre,
          'apellido': jugadorIn.apellido,
          'telefono': jugadorIn.telefono,
          'email': jugadorIn.email
        };

    //ENCAPSULAMOS EL MAP EN JSON PARA ENVIAR
    var jsonOut = json.encode(data);
  sharedPreferences = await SharedPreferences.getInstance();
  var response = await http.post(apiRoute + 'Jugadores/new', body: jsonOut,
                headers:
                {
                HttpHeaders.contentTypeHeader: "application/json",
                HttpHeaders.authorizationHeader: "Bearer "+ sharedPreferences.getString('access_token')
                },
                );
  if (response.statusCode == 200){

    dialogs.information(context, "Jugador añadido", "Jugador añadido con exito!");
    //print("JUGADOR AÑADIDO CON EXITO");

  }else{
    print(response.statusCode);

  }

  }
```

```
  static editarJugador(Jugador jugadorIn, BuildContext context) async {
    Map data = {
          'nombre': jugadorIn.nombre,
          'apellido': jugadorIn.apellido,
          'telefono': jugadorIn.telefono,
          'email': jugadorIn.email
        };

    //ENCAPSULAMOS EL MAP EN JSON PARA ENVIAR
    var jsonOut = json.encode(data);
  sharedPreferences = await SharedPreferences.getInstance();
  var response = await http.put(apiRoute + 'Jugadores/edit/'+jugadorIn.jugador_id.toString(), body: jsonOut,
                headers:
                {
                HttpHeaders.contentTypeHeader: "application/json",
                HttpHeaders.authorizationHeader: "Bearer "+ sharedPreferences.getString('access_token')
                },
                );
  if (response.statusCode == 200){

    dialogs.information(context, "Jugador editado", "Jugador editado con exito!");
    print("JUGADOR Editado CON EXITO");

  }else{
    dialogs.error(context, "Error al editar", "Error al editar jugador");
    print(response.statusCode);

  }

  }
```

JugadorController.php ->back-end code to add or edit

```php
public function store(Request $request)
{
    $jugadorIn = new Jugador();
    $jugadorIn->nombre = $request->nombre;
    $jugadorIn->apellido = $request->apellido;
    $jugadorIn->telefono = $request->telefono;
    $jugadorIn->email = $request->email;
    $jugadorIn-> save();

    return response()->json($jugadorIn);
}
```

```php
public function update(Request $request, $id)
{
    $jugador = Jugador::findorfail($id);
    $jugador->update($request->all());
    return $jugador;
}
```

11- Thoughts and conclusions

This was more like a test for me, I could have taken the easy way and make something in a platform I already have knew, but really, I wanted to make sure that I could do something more complicated, prepare for something that could happen in the future.

To take a new platform and to be able to program something from the scratch to almost making a fully functional app in three months, I knew it was going to require some work, but I tried to maintain a regular schedule and not leave everything for the last moment as I usually do.

Working in a company also made me look at it from this perspective easily, I often see the other workers have a new problem with their applications and they have very little time to make it function, so watching them how they handled that pressure and how they organized everything helped a lot. I feel like my mentality has changed from the start of the course to now.

Therefore, I think that this approach has made me learn in a much more efficient way and have more time to do most of the things that were planned from the start. Flutter is a great and somewhat difficult but rewarding platform to program with, which made me take it more seriously and try to do my best.

12- Bibliography

LARAVEL AUTH PASSPORT

https://medium.com/@cvallejo/sistema-de-autenticación-api-rest-con-laravel-5-6-240be1f3fc7d

---

API LARAVEL

https://www.toptal.com/laravel/restful-laravel-api-tutorial

https://www.youtube.com/watch?v=c9DCvWim8gA

LARAVEL RELATIONS

https://www.youtube.com/watch?v=3Oxfi3DLdkI

FLUTTER HTTP REQUESTS

https://flutter.dev/docs/cookbook/networking/fetch-data

https://flutter.dev/docs/development/data-and-backend/networking (ANDROID PERMISSIONS)

https://www.youtube.com/watch?v=-PRrdG163to JSON TO LISTVIEW

FLUTTER LOGIN MYSQL + CRUD

https://www.youtube.com/watch?v=kwZeCZr8Axsç

https://www.youtube.com/watch?v=2DtFGF2v_vk

FLUTTER SWIPABLE

https://stackoverflow.com/questions/46651974/swipe-list-item-for-more-options-flutter

https://github.com/letsar/flutter_slidable

FLUTTER SEARCH DELEGATE

https://www.youtube.com/watch?time_continue=361&v=Wm3OiFBZ2xI&feature=emb_logo (6:00)

FLUTTER FUTURE BUILDER -> LIST WITH JSON -> FILTERING

https://www.youtube.com/watch?v=EwHMSxSWIvQ&t=54s

https://www.youtube.com/watch?v=-EUsRa2G1zk

FLUTTER CRASH COURSE

https://fluttercrashcourse.com/

FLUTTER NESTED TAB BAR

https://medium.com/flutterarsenal/build-a-nested-tabbar-in-flutter-7e0cae5cfc7
