Manual Del Programador

Tabla de contenido

Tabla de Figuras

1. Introducción

El propósito de la elaboración de este manual de programación es detallar como fue la construcción del aplicativo móvil con el fin de que el lector de los códigos los pueda conocer y así mismo detallar cada uno de estos.

2. Programación

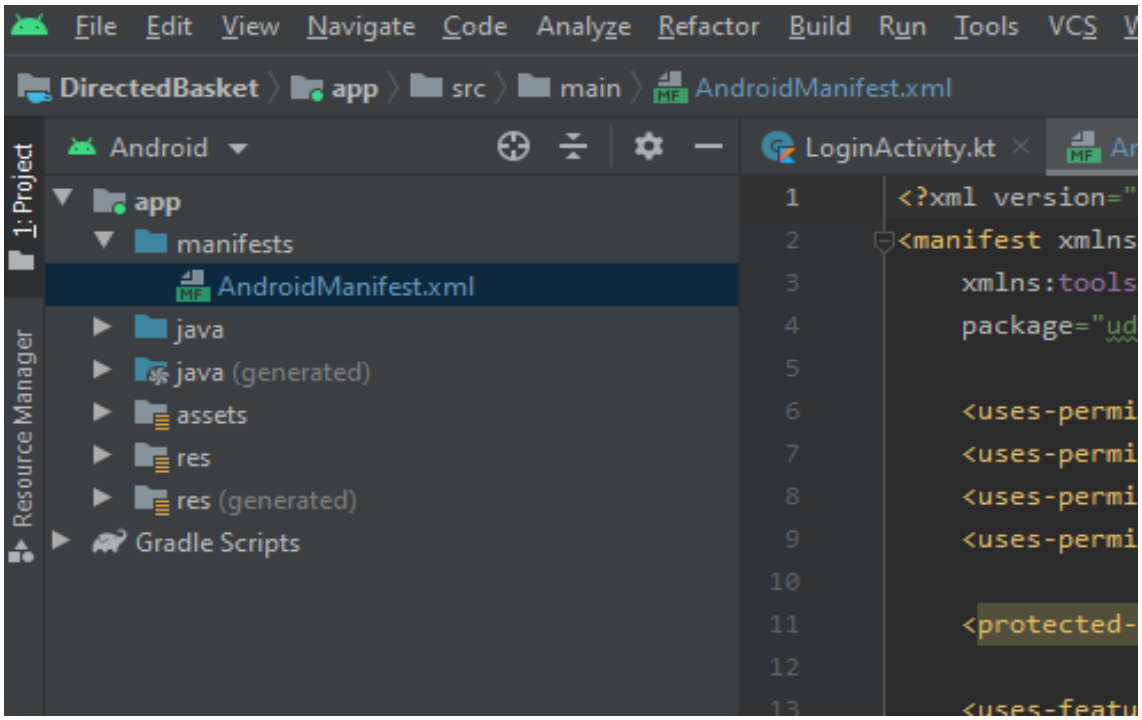
Para la construcción de esta Aplicación móvil se utilizó el entorno de desarrollo de Androidstudio, como lenguaje de programación se elaboró en Kotlin.

- ***App:*** es donde van a estar alojados todos los archivos necesarios y que se van a requerir dentro de la aplicación.

- ***Manifest:*** en esta carpeta este alojado un archivo llamado AndroidManifest.xml, aquí se hace la configuración de los permisos que se debe tener, la vistas, archivos, tipo de orientación de igual forma aquí se encuentra todos los nodos descriptivos sobre la aplicación de Android.

2.1.1. Estructura Manifest

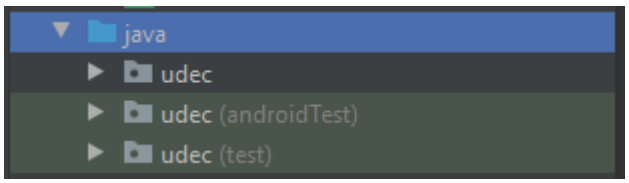
Figura 1 Manifest.



Fuente: Autores del proyecto.

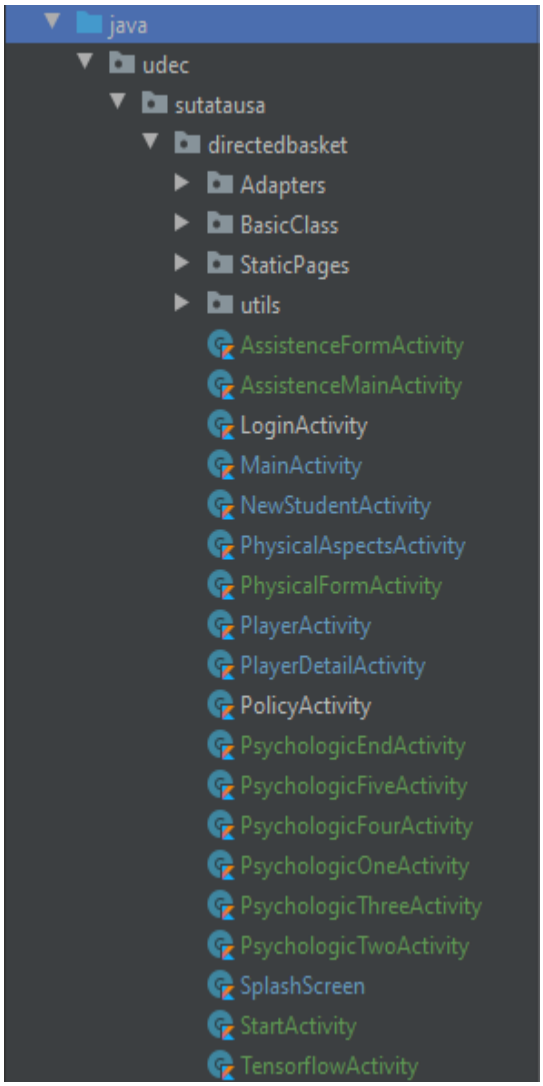
- ***Java:*** aquí están alojados todos los archivos con extensión .java de la misma manera aquí esta toda la parte lógica de la aplicación con cada uno de sus archivos por lo tanto viene siendo el controlador de la vista.

Figura 2 Archivos java



Fuente: Autores del proyecto.

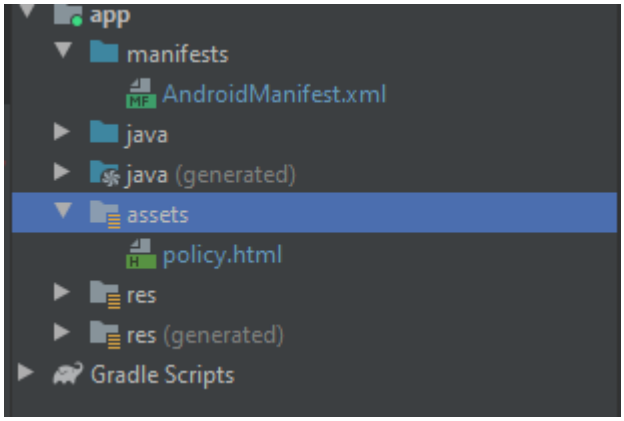
Figura 3 Estructura general.



Fuente: Autores del proyecto.

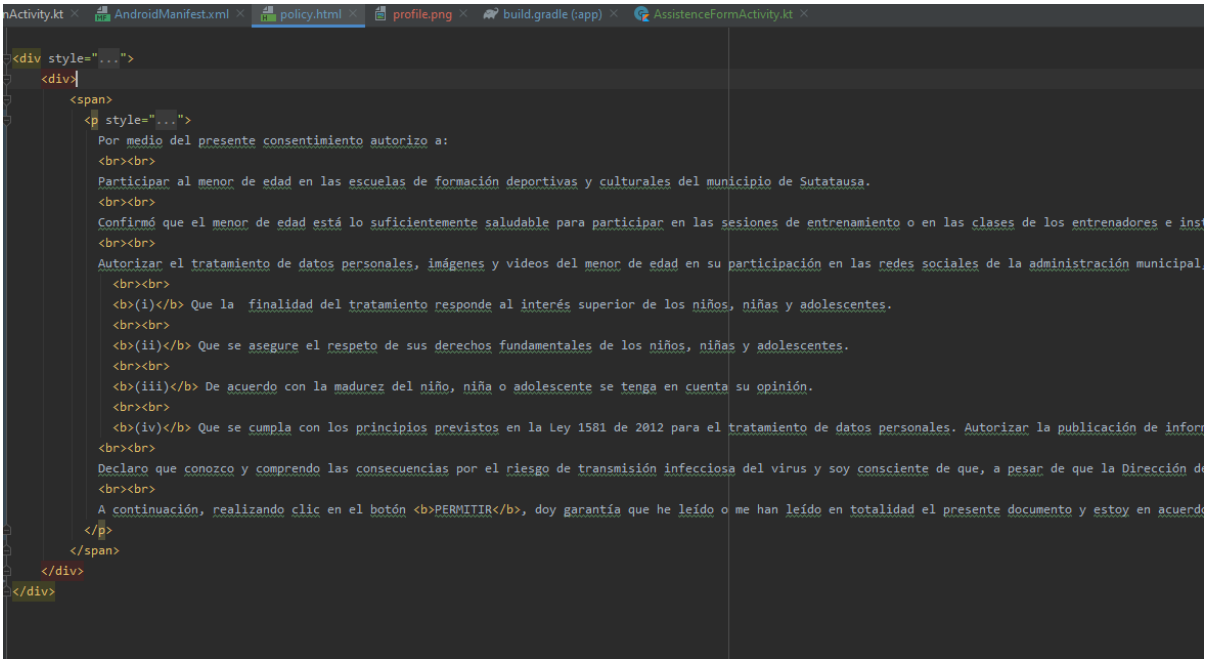
- ***Assets:*** aquí están los recursos, en esta carpeta se pueden cargar los archivos de imagen en este caso aquí se encuentra el consentimiento informado.

Figura 4 Assets



Fuente: Autores del proyecto.

Figura 5 Policy.html



Fuente: Autores del proyecto.

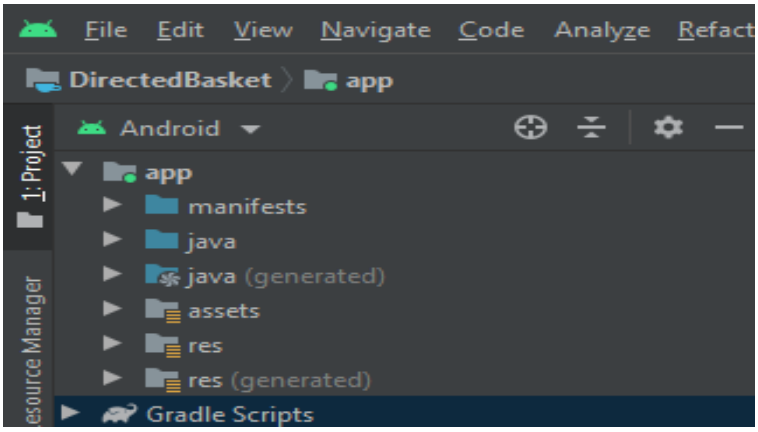
- ***Res:*** aquí están alojadas una serie de carpetas y todos los recursos como: imágenes, vistas de la aplicación, archivos de lista de valores y los recursos de la aplicación.

Figura 6 Estructura de archivos res.

Fuente: Autores del proyecto.

- ***Gradle Scripts:*** en esta carpeta se encuentran todos los manejadores de paquete y posee a la vez herramientas de complicación avanzadas.

Figura 7 Gradle scripts.



Fuente: Autores del proyecto.

3. Librerias

En total fueron 6 librerías que se utilizaron para el desarrollo del aplicativo móvil.

- **com.google.firebase.auth.FirebaseAuth:** este es el punto de entrada al SDK de autenticación que tiene Firebase.
- **com.google.firebase:firebase-auth-ktx:** librería para la autenticación con kotlin.
- **com.google.firebase:firebase-database-ktx:** librería para la autenticación con kotlin.
- **com.google.firebase:firebase-storage-ktx:** librería para la autenticación con kotlin.
- **com.squareup.picasso:picasso:2.71828:** librería para el manejo de imágenes.
- **Implementation.im.dacer:AndroidCharts:1.0.4:** librería para la gestión de gráficas.

4. Vistas del Aplicativo

AndroidManifest.xml: aquí se encuentra el panorama de todo el aplicativo de igual forma se describe toda la información para el desarrollo de la aplicación en cuanto a las herramientas de creación para el sistema operativo Android y Google play.

Figura 8 Android manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="udec.sutatausa.directedbasket">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <protected-broadcast android:name="android.intent.action.MEDIA_MOUNTED" />

    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera2" />

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="Directed Basket"
        android:requestLegacyExternalStorage="true"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="true">
        <activity
            android:name=".SplashScreen"
            android:theme="@style/SplashTheme"
            tools:ignore="Instantiatable">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
```

Fuente: Autores del proyecto.

User List Adapter*:* Manejo de vistas.

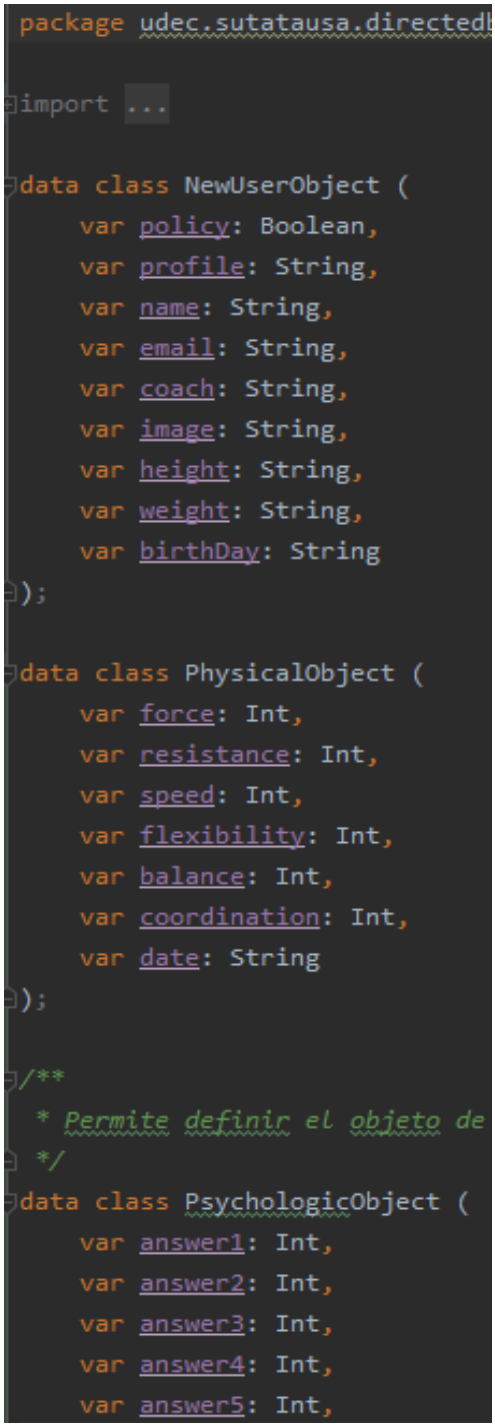
Figura 9 User list adapter.

```
class UserListAdapter: RecyclerView.Adapter<UserListAdapter.UserListHolder>() {

    var userList: MutableList<UserDataHome> = ArrayList();
    lateinit var context: Context;

    fun UserListAdapter(userList : MutableList<UserDataHome>, context: Context){
        this.userList = userList;
        this.context = context;
    }

    /**
     * Permite crear una vista por cada item de la lista
     */
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UserListHolder {

        // Definimos la vista que se desea tomar como referencia
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.recycler_user_home, parent, attachToRoot: false)

        // retornamos
        return UserListHolder(view);

    }

    /**
     * Define cuantos elementos debe manejar
     */
    override fun getItemCount(): Int {
        return userList.size;
    }
```

Fuente: Autores del proyecto.

CustomObjects.kt: aquí se define el objeto de un usuario y los constructos con cada uno de sus respectivos datos.

Figura 10 Custom objects



```
package udec.sutatausa.directedl

import ...

data class NewUserObject (
    var policy: Boolean,
    var profile: String,
    var name: String,
    var email: String,
    var coach: String,
    var image: String,
    var height: String,
    var weight: String,
    var birthDay: String
);

data class PhysicalObject (
    var force: Int,
    var resistance: Int,
    var speed: Int,
    var flexibility: Int,
    var balance: Int,
    var coordination: Int,
    var date: String
);

/**
 * Permite definir el objeto de
 */
data class PsychologicObject (
    var answer1: Int,
    var answer2: Int,
    var answer3: Int,
    var answer4: Int,
    var answer5: Int,
```

Fuente: Autores del proyecto.

Figura 11 Custom object

```
 * Permite definir el objeto de un usuario
 */
class UserObject {
    var policy: Boolean? = false;
    var profile: String? = "";
    var name: String? = "";
    var email: String? = "";
    var coach: String? = "";
    var image: String? = "";
    var height: String? = "";
    var weight: String? = "";
    var birthDay: String? = "";

    /**
     * Definimos el constructor
     */
    constructor(policy: Boolean?, profile: String
        this.policy = policy;
        this.profile = profile;
        this.name = name;
        this.email = email;
        this.coach = coach;
        this.image = image;
        this.height = height;
        this.weight = weight;
        this.birthDay = birthDay;
    }

    constructor()   // **Add this**
}
```

Fuente: Autores del proyecto.

4.1. StaticPages

***Page1Activity*:** Aquí se hacen las inicializaciones de las variables para las opciones en el aplicativo de continuar o regresar de igual forma aquí se obtienen las siguientes funcionalidades: información del id del usuario, inicialización de la clase, definición de actividad a mostrar, terminación de la actividad actual, actividad a mostrar, id del usuario que se desea procesar.

Figura 12 Page 1 activity

```
package udec.sutatausa.directedbasket.StaticPages

import ...

class Page1Activity : AppCompatActivity() {

    // se inicializa las variables de continuar o regresar
    var btnNextId: Button? = null;
    var btnPrevId: Button? = null;

    var G_CURR_DATA: String = "";
    var G_NEXT_PAGE: Int = 2;
    var G_PREV_PAGE: Int = 1;

    // instanciamos la clase de funciones generales
    lateinit var GlobalMethods: GlobalMethods;

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_page1);

        // Obtenemos la información del id del usuario
        val bundle = intent.extras;
        G_CURR_DATA = bundle?.getString( key: "currData").toString();
        G_NEXT_PAGE = bundle?.getString( key: "nextPage")?.toInt() ?: 2;
        G_PREV_PAGE = G_NEXT_PAGE - 1;

        // iniciamos la clase
        GlobalMethods = GlobalMethods( activity: this);

        // referenciamos los campos
        btnPrevId = findViewById(R.id.btnPrevId1);
        btnNextId = findViewById(R.id.btnNextId1);
```

Fuente: Autores del proyecto.

- Esta función permite ejecutar un evento al oprimir la fecha de atrás.

Figura 13 Static Pages

```
override fun onBackPressed() {

    // definimos cual es la actividad a mostrar
    var intent = GlobalMethods.getIntentByPos(G_PREV_PAGE);

    // agregamos el id del usuario que deseamo procesar
    intent.putExtra( name: "currData", G_CURR_DATA);

    // Mostramos la actividad
    startActivity(intent);

    // terminamos la actividad actual
    finish();
}
```

Fuente: Autores del proyecto.

4.1.1. Page2Activity

Inicialización de las variables de continuar o regresar.

Figura 14 Page 2 Activity.

```
var btnNextId: Button? = null;
var btnPrevId: Button? = null;

var G_CURR_DATA: String = "";
var G_NEXT_PAGE: Int = 2;
var G_PREV_PAGE: Int = 1;
```

Fuente: Autores del proyecto.

Se obtiene la información del id del usuario.

Figura 15 Page 2 activity

```
lateinit var GlobalMethods: GlobalMethods;

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_page2);
```

Fuente: Autores del proyecto.

Figura 16 Page 2 Activity

```
// definimos cual es la actividad a mostrar
var intent = GlobalMethods.getIntentByPos(G_PREV_PAGE);

// agregamos el id del usuario que deseamo procesar
intent.putExtra( name: "currData", G_CURR_DATA);

// Mostramos la actividad
startActivity(intent);

// terminamos la actividad actual
finish();
```

Fuente: Autores del proyecto.

Permite ejecutar un evento al oprimir la fecha de atrás.

Figura 17 Page 2 Activity

```
override fun onBackPressed() {

    // definimos cual es la actividad a mostrar
    var intent = GlobalMethods.getIntentByPos(G_PREV_PAGE);

    // agregamos el id del usuario que deseamo procesar
    intent.putExtra( name: "currData", G_CURR_DATA);

    // Mostramos la actividad
    startActivity(intent);

    // terminamos la actividad actual
    finish();
```

Fuente: Autores del proyecto.

4.1.2. Page3Activity

Se inicializan las variables de continuar o regresar

Figura 18 Page 3 Activity

```
var btnNextId: Button? = null;
var btnPrevId: Button? = null;


var G_CURR_DATA: String = "";
var G_NEXT_PAGE: Int = 2;
var G_PREV_PAGE: Int = 1;
```

Fuente: Autores del proyecto.

Figura 19 Page 3 Activity

```
// Obtenemos la información del id del usuario
val bundle = intent.extras;
G_CURR_DATA = bundle?.getString( key: "currData").toString();
G_NEXT_PAGE = bundle?.getString( key: "nextPage")?.toInt() ?: 2;
G_PREV_PAGE = G_NEXT_PAGE - 1;

// iniciamos la clase
GlobalMethods = GlobalMethods( activity: this);

// referenciamos los campos
btnPrevId = findViewById(R.id.btnPrevId3);
btnNextId = findViewById(R.id.btnNextId3);
```

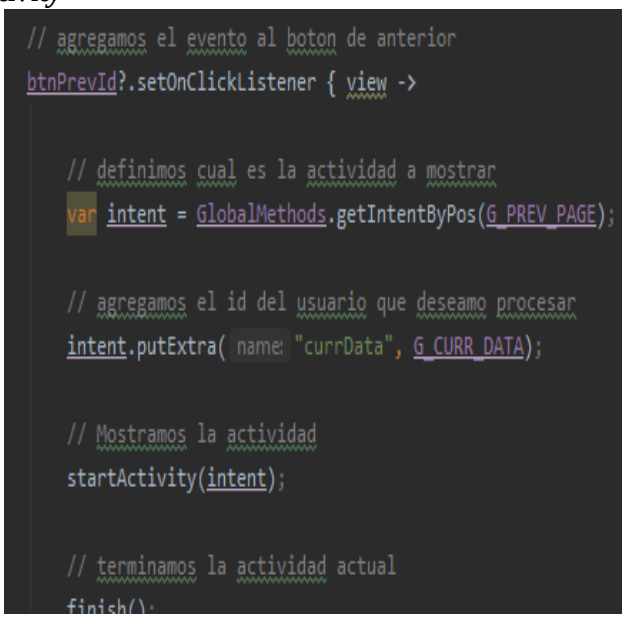
Fuente: Autores del proyecto.

Figura 20 Page 3 Activity

```
// agregamos el evento al boton de siguiente
btnNextId?.setOnClickListener { view ->

    // definimos cual es la actividad a mostrar
    var intent = GlobalMethods.getIntentByPos(G_NEXT_PAGE);

    // agregamos la información actual
    intent.putExtra( name: "currData", G_CURR_DATA);

    // Mostramos la actividad
    startActivity(intent);

    // terminamos la actividad actual
    finish();
}
```

Fuente: Autores del proyecto.

Figura 21 Page 3 Activity



```
// agregamos el evento al boton de anterior
btnPrevId?.setOnClickListener { view ->

    // definimos cual es la actividad a mostrar
    var intent = GlobalMethods.getIntentByPos(G_PREV_PAGE);

    // agregamos el id del usuario que deseamo procesar
    intent.putExtra( name: "currData", G_CURR_DATA);

    // Mostramos la actividad
    startActivity(intent);

    // terminamos la actividad actual
    finish();
```

Fuente: Autores del proyecto.

4.1.3. Page4 Activity

Figura 22 Page 4 Activity

```
package udec.sutatausa.directedbasket.StaticPages

import ...

class Page4Activity : AppCompatActivity() {

    // se inicializa las variables de continuar o regresar
    var btnNextId: Button? = null;
    var btnPrevId: Button? = null;

    var G_CURR_DATA: String = "";
    var G_NEXT_PAGE: Int = 2;
    var G_PREV_PAGE: Int = 1;

    // instanciamos la clase de funciones generales
    lateinit var GlobalMethods: GlobalMethods;

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_page4);
```

Fuente: Autores del proyecto.

Figura 23 Page 4 Activity

```
lateinit var GlobalMethods: GlobalMethods;

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_page4);

    // Obtenemos la información del id del usuario
    val bundle = intent.extras;
    G_CURR_DATA = bundle?.getString( key: "currData").toString();
    G_NEXT_PAGE = bundle?.getString( key: "nextPage")?.toInt() ?: 2;
    G_PREV_PAGE = G_NEXT_PAGE - 1;

    // iniciamos la clase
    GlobalMethods = GlobalMethods( activity: this);

    // referenciamos los campos
    btnPrevId = findViewById(R.id.btnPrevId4);
    btnNextId = findViewById(R.id.btnNextId4);

    // agregamos el evento al boton de siguiente
    btnNextId?.setOnClickListener { view ->
```

Fuente: Autores del proyecto.

Figura 24 Page 4 Activity

```
// agregamos el evento al boton de anterior
btnPrevId?.setOnClickListener { view ->

    // definimos cual es la actividad a mostrar
    var intent = GlobalMethods.getIntentByPos(G_PREV_PAGE);

    // agregamos el id del usuario que deseamo procesar
    intent.putExtra( name: "currData", G_CURR_DATA);

    // Mostramos la actividad
    startActivity(intent);

    // terminamos la actividad actual
    finish();
```

Fuente: Autores del proyecto.

Figura 25 Page 4 Activity

```
override fun onBackPressed() {

    // definimos cual es la actividad a mostrar
    var intent = GlobalMethods.getIntentByPos(G_PREV_PAGE);

    // agregamos el id del usuario que deseamo procesar
    intent.putExtra( name: "currData", G_CURR_DATA);

    // Mostramos la actividad
    startActivity(intent);

    // terminamos la actividad actual
    finish();
```

Fuente: Autores del proyecto.

4.1.4. Page5 Activity

Figura 26 Page 5 Activity

```
// se inicializa las variables de continuar o regresar
var btnNextId: Button? = null;
var btnPrevId: Button? = null;

var G_CURR_DATA: String = "";
var G_NEXT_PAGE: Int = 2;
var G_PREV_PAGE: Int = 1;
```

Fuente: Autores del proyecto.

Figura 27 Page 5 Activity

```
// instanciamos la clase de funciones generales
lateinit var GlobalMethods: GlobalMethods;
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_page5);

    // Obtenemos la información del id del usuario
    val bundle = intent.extras;
    G_CURR_DATA = bundle?.getString( key: "currData").toString();
    G_NEXT_PAGE = bundle?.getString( key: "nextPage")?.toInt() ?: 2;
    G_PREV_PAGE = G_NEXT_PAGE - 1;
```

Fuente: Autores del proyecto.

Figura 28 Page 5 Activity

```
// iniciamos la clase
GlobalMethods = GlobalMethods( activity: this);

// referenciamos los campos
btnPrevId = findViewById(R.id.btnPrevId5);
btnNextId = findViewById(R.id.btnNextId5);

// agregamos el evento al boton de siguiente
btnNextId?.setOnClickListener { view ->

    // definimos cual es la actividad a mostrar
    var intent = GlobalMethods.getIntentByPos(G_NEXT_PAGE);

    // agregamos la información actual
    intent.putExtra( name: "currData", G_CURR_DATA);

    // Mostramos la actividad
    startActivity(intent);

    // terminamos la actividad actual
    finish();
```

Fuente: Autores del proyecto.

Utils: Aquí se hace el manejo de todas las actividades que se encuentran dentro del aplicativo móvil.

APIAcessTask: manejo de las peticiones http

Figura 29 API Acess Task

```
class HttpTask(callback: (String?) -> Unit) : AsyncTask<String, Unit, String>()  {
    // Variable globales
    var callback = callback;
    val TIMEOUT = 10*1000;
```

Fuente: Autores del proyecto.

Figura 30 Api Acess Task

```
try {
    if (params[0] == "POST") {
        // Estalecemos la propiedades unicas de un metodo POST
        //httpClient.instanceFollowRedirects = false;
        httpClient.doInput = true;
        httpClient.useCaches = false;
```

Fuente: Autores del proyecto.

Figura 31 Api Acess Task

```
// Se realiza la conexión
httpClient.connect();
val os = httpClient.getOutputStream();
val writer = BufferedWriter(OutputStreamWriter(os, charsetName: "UTF-8"));
writer.write(params[2]);
writer.flush();
writer.close();
os.close();
```

Fuente: Autores del proyecto.

Figura 32 Api Acess Task

```
/**
 * Permite obtener los datos que retorna el webservices
 */
fun readStream(inputStream: BufferedInputStream): String {
    val bufferedReader = BufferedReader(InputStreamReader(inputStream))
    val stringBuilder = StringBuilder()
    bufferedReader.forEachLine { stringBuilder.append(it) }
    return stringBuilder.toString()
}
```

Fuente: Autores del proyecto.

Figura 33 API Acess Task

```
/**
 * Al finalizar el consumo del webservice se ejecuta la función del callback
 */
override fun onPostExecute(result: String?) {
    super.onPostExecute(result);
    callback(result);
}
```

Fuente: Autores del proyecto.

Global Methods: aquí se tiene en cuenta todas las actividades existentes de igual forma permite mostrar los mensajes que aparecen en la interfaz.

Figura 34 Global Methods

```
class GlobalMethods( activity: Activity){

    // definimos la variable que referencia la actividad donde se inicio la clase
    lateinit var myActivity: Activity;

    // initializer block
    init {
        println("Clase Iniciada")
        myActivity = activity;
    }
```

Fuente: Autores del proyecto.

Figura 35 Global method

```
/**
 * Permite mostrar mensaje en la interfaz
 */
@SuppressLint( ...value: "ResourceAsColor")
fun showAlert(title: String?, message: String?, listener: DialogInterface.OnClickListener? ) {

    // Creamos una interfaz de dialogo para mostrar el mensa
    val builder = MaterialAlertDialogBuilder(myActivity, R.style.AlertDialogTheme);

    // definimos la variable para guardar el titulo
    var titleAlert = title;

    // valido si el titulo es null
    if(titleAlert == null) titleAlert = "Mensaje".toString();
```

Fuente: Autores del proyecto.

Figura 36 Global methods

```
@SuppressLint( ...value: "ResourceAsColor")
fun showConfirm(message: String?, listener: DialogInterface.OnClickListener? ) {

    // Creamos una interfaz de dialogo para mostrar el mensaje
    val builder = MaterialAlertDialogBuilder(myActivity, R.style.AlertDialogTheme);

    // agregamos las propiedades
    builder.setTitle("Mensaje")
        .setMessage(message)
        .setCancelable(false)
        .setPositiveButton("Aceptar", listener)
        .setNegativeButton("Cancelar", listener: null);

    // Se crea el alerta y se visualiza en la interfaz
    builder.show();
}
```

Fuente: Autores del proyecto.

Figura 37 Global Methods

```
/**
 * Se valida si es un correo
 */
fun isEmailValid(email: CharSequence?): Boolean {
    return Patterns.EMAIL_ADDRESS.matcher(email).matches();
}

/**
 * Permite validar un tipo de dato
 */
fun verifyField(stringValue: String, fieldName: String, type: String?): String {

    // definimos el mensaje a retornar
    var returnMessage = "";

    // removemos los espacios
    val newStringValue = stringValue.trim();

    // validamos si el campo esta vacio
    if(newStringValue.isEmpty()){
        // retornamos por defecto el mensaje que es obligatorio el campo
        return "Este campo '$fieldName' es obligatorio.";
    }
```

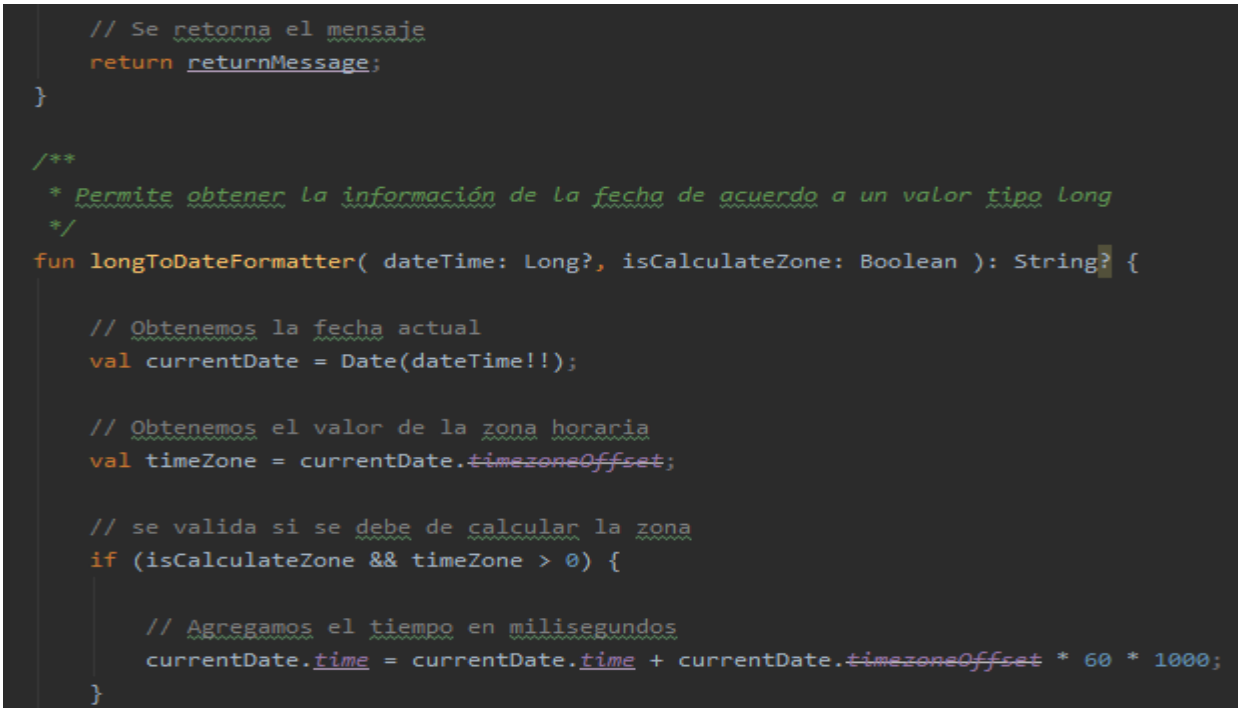
Fuente: Autores del proyecto.

Figura 38 Global Methods

```
// validamos los tipos de datos
when (type){
    "email" ->
        if(!Patterns.EMAIL_ADDRESS.matcher(newStringValue).matches()) {
            returnMessage = "Ingresar un correo valido.";
        }
    "number" ->
        if(!"\\d+".toRegex().matches(newStringValue)) {
            returnMessage = "Ingresar un número valido.";
        }
    "password" ->
        if(newStringValue.length < 6) {
            returnMessage = "La contraseña debe contener 6 dígitos como mínimo.";
        }
}
```

Fuente: Autores del proyecto.

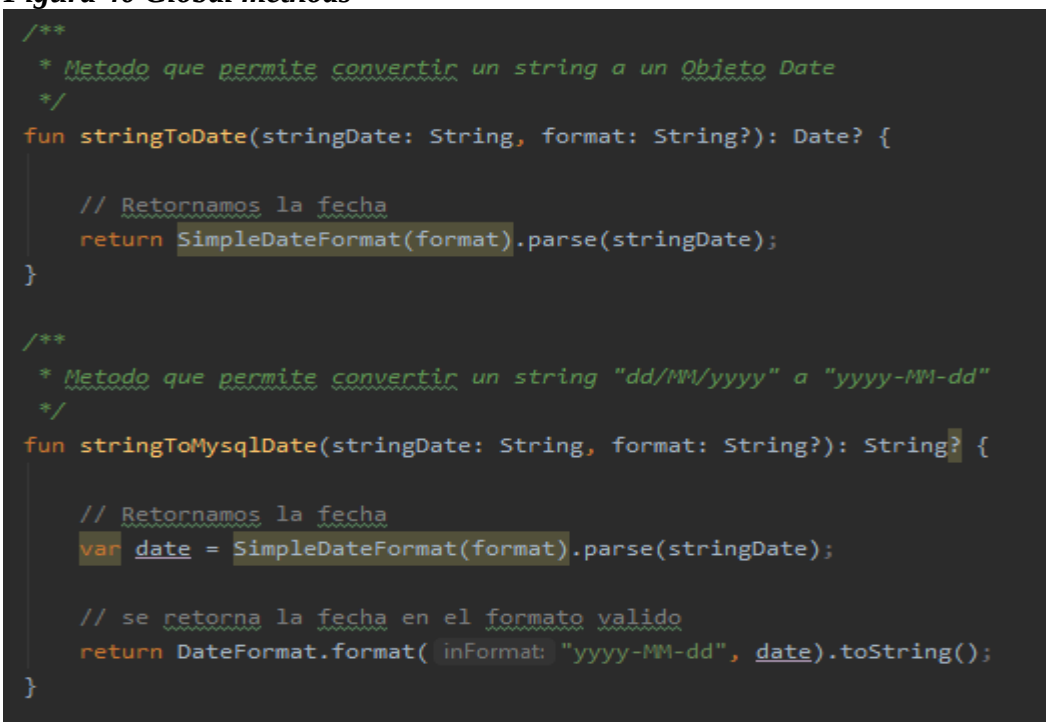
Figura 39 Global Methods



```
    // Se retorna el mensaje
    return returnMessage;
}

/**
 * Permite obtener la información de la fecha de acuerdo a un valor tipo long
 */
fun longToDateFormatter( dateTime: Long?, isCalculateZone: Boolean ): String? {

    // Obtenemos la fecha actual
    val currentDate = Date(dateTime!!);

    // Obtenemos el valor de la zona horaria
    val timeZone = currentDate.timezoneOffset;

    // se valida si se debe de calcular la zona
    if (isCalculateZone && timeZone > 0) {

        // Agregamos el tiempo en milisegundos
        currentDate.time = currentDate.time + currentDate.timezoneOffset * 60 * 1000;
    }
```

Fuente: Autores del proyecto.

Figura 40 Global methods



```
/**
 * Metodo que permite convertir un string a un Objeto Date
 */
fun stringToDate(stringDate: String, format: String?): Date? {

    // Retornamos la fecha
    return SimpleDateFormat(format).parse(stringDate);
}

/**
 * Metodo que permite convertir un string "dd/MM/yyyy" a "yyyy-MM-dd"
 */
fun stringToMysqlDate(stringDate: String, format: String?): String? {

    // Retornamos la fecha
    var date = SimpleDateFormat(format).parse(stringDate);

    // se retorna la fecha en el formato valido
    return DateFormat.format( inFormat: "yyyy-MM-dd", date).toString();
}
```

Fuente: Autores del proyecto.

Figura 41 Global Methods

```
/**
 * Metodo que permite convertir un string "yyyy-MM-dd" to "May dd"
 */
fun getLabelDate(stringDate: String?): String? {

    // Retornamos la fecha
    var date = SimpleDateFormat( pattern: "yyyy-MM-dd").parse(stringDate);

    // lista de meses
    val months = arrayOf("Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep", "Oct", "Nov", "Dic");

    // se retorna la fecha en el formato valido
    return months[date.month] + " " + date.date;
}
```

Fuente: Autores del proyecto.

Figura 42 Global Methods

```
/**
 * Metodo que permite agregar un valor que puede ser accedido en cualquier parte de la aplicación
 */
fun addPropertyValue(activity: Activity, property: String, value: String?) {

    // Referenciamos las propiedades por defecto de la aplicación
    val myPreferences = activity.getSharedPreferences( name: "udec.sutatausa.directedbasket_preferences", Context.MODE_PRIVATE)

    // Habilitamos las preferencias del editor para agregar datos
    val myEditor = myPreferences.edit();

    // agregamos los datos
    myEditor.putString(property, value);

    // Aplicamos los cambios
    myEditor.apply();
```

Fuente: Autores del proyecto.

Figura 43 Global Methods.

```
/**
 * Metodo que permite obtener el valor de una propiedad compartida
 */
fun getPropertyValue(activity: Activity, property: String?): String? {

    // Referenciamos las propiedades por defecto de la aplicación
    val myPreferences = activity.getSharedPreferences( name: "udec.sutatausa.directedbasket_preferences", Context.MODE_PRIVATE)

    // retornamo s el valor
    return myPreferences.getString(property, "");
}
```

Fuente: Autores del proyecto.

Figura 44 Global Methods.

```
/**
 * Metodo que permite eliminar una propiedad de las configuraciones del aplicativo
 */
fun removeProperty(activity: Activity, property: String) {

    // Referenciamos las propiedades por defecto de la aplicación
    val myPreferences = activity.getSharedPreferences( name: "udec.sutatausa.directedbasket_preferences", Context.MODE_PRIVATE)

    // Habilitamos las preferencias del editor para agregar datos
    val myEditor = myPreferences.edit();

    // removemos los datos
    myEditor.remove(property);

    // Aplicamos los cambios
    myEditor.apply();
}
```

Fuente: Autores del proyecto.

Figura 45 Global Methods.

```
/**
 * Permite obtener la edad
 */
fun getAge(dobString: String?): Int {
    var date: Date? = null;
    val sdf = SimpleDateFormat( pattern: "dd/MM/yyyy");
    try {
        date = sdf.parse(dobString);
    } catch (e: ParseException) {}
    println(date)
    // validamos si no existe fecha
    if (date == null) return 0;

    // obtenemos 2 instanciaas de fechas actuales
    val dob: Calendar = Calendar.getInstance();
    val today: Calendar = Calendar.getInstance();
    println(dob)
    // agregamos al primer objeto la fecha de nacimiento
    dob.setTime(date);

    // Obtenemos los datos de la fecha
    val year: Int = dob.get(Calendar.YEAR);
    val month: Int = dob.get(Calendar.MONTH);
    val day: Int = dob.get(Calendar.DAY_OF_MONTH);
```

Fuente: Autores del proyecto.

Figura 46 Global Methods.

```
/**
 * Permite obtener el Intent de acuerdo a la posición
 **/
fun getIntentByPos(pos: Int): Intent {

    // se valida si la posici+on es 1
    if(pos == 1){
        return Intent(myActivity, PsychologicOneActivity::class.java);
    } else if(pos == 2){
        return Intent(myActivity, PsychologicTwoActivity::class.java);
    } else if(pos == 3){
        return Intent(myActivity, PsychologicThreeActivity::class.java);
    } else if(pos == 4){
        return Intent(myActivity, PsychologicFourActivity::class.java);
    } else if(pos == 5){
        return Intent(myActivity, PsychologicFiveActivity::class.java);
    } else {
        return Intent(myActivity, PsychologicEndActivity::class.java);
    }

}
```

Fuente: Autores del proyecto.

Figura 47 Global Methods.

```
 * Permite obtener el Intent de una pagina estatica
 **/
fun getStaticPage(): Intent {

    // Calculamos un númeo ramdomico
    val random = Random();
    val pos = random.nextInt( bound: 5 ) + 1;

    // se valida si la posici+on es 1
    if(pos == 1){
        return Intent(myActivity, Page1Activity::class.java);
    } else if(pos == 2){
        return Intent(myActivity, Page2Activity::class.java);
    } else if(pos == 3){
        return Intent(myActivity, Page3Activity::class.java);
    } else if(pos == 4){
        return Intent(myActivity, Page4Activity::class.java);
    } else {
        return Intent(myActivity, Page5Activity::class.java);
    }
```

Fuente: Autores del proyecto.

Loading Dialog: aquí se tiene en cuenta la variable de referencia correspondiente a cada actividad en donde se inició la clase.

Figura 48 Loading Dialog

```
lateinit var myActivity: Activity;

private val dialog: AlertDialog?;
private var loadingMessage = "Loading...";
var messageView: TextView? = null;

/**
 * metodo que permite agregar el texto
 */
fun setText(text: String?) {
    messageView!!.text = text;
}
```

Fuente: Autores del proyecto.

Figura 49 Loading Dialog

```
/**
 * Creamos el metodo que muestra la actividad con el loading personalizado
 */
fun startLoadingDialog() {

    // Se muestra el loading
    dialog!!.show();
```

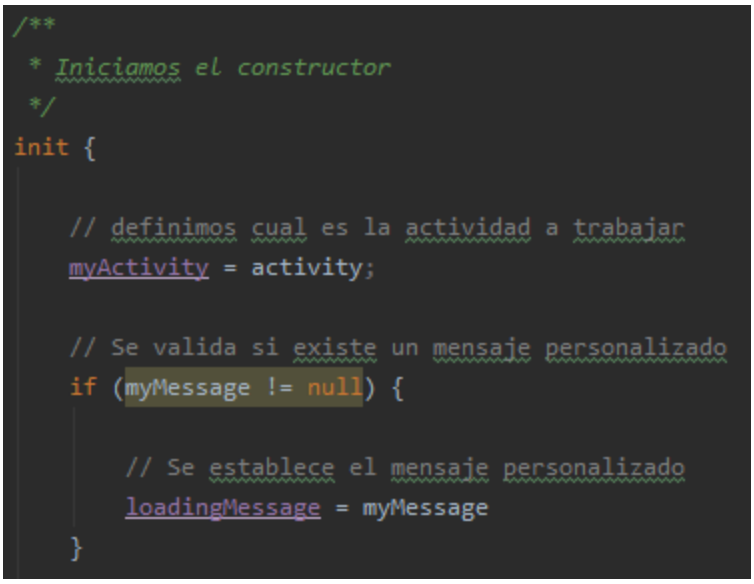
Fuente: Autores del proyecto.

Figura 50 Loading Dialog

```
/**
 * Creamos el metodo que permite ocultar el loading
 */
fun dissmissDialog() {

    // Se valida si au existe  la actividad para ocultar la ventana
    if (myActivity != null && !myActivity.isFinishing && dialog != null && dialog.isShowing) {

        // Se oculta la ventana de dialog
        dialog.dismiss();
    }
}
```

Fuente: Autores del proyecto.

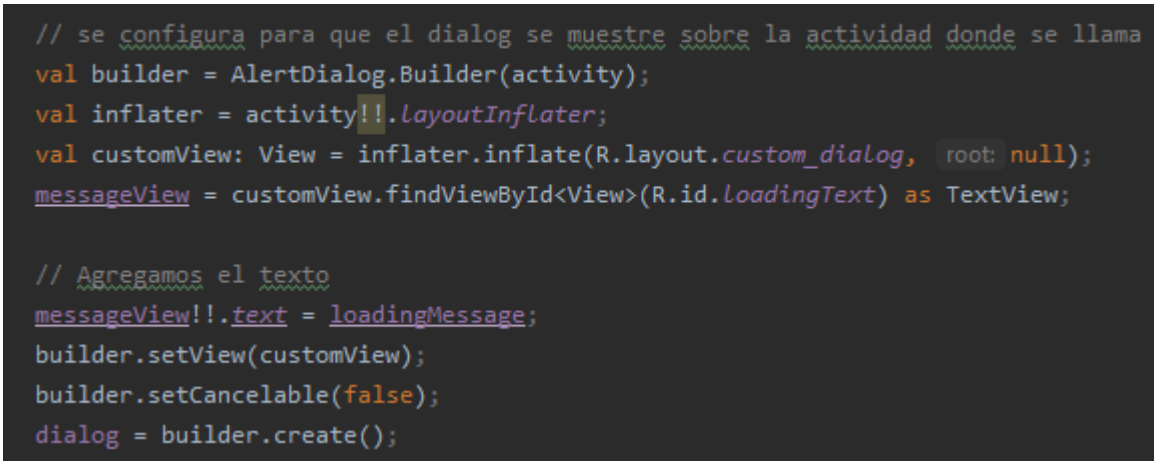
Figura 51 Loading Dialog



```
/**
 * Iniciamos el constructor
 */
init {

    // definimos cual es la actividad a trabajar
    myActivity = activity;

    // Se valida si existe un mensaje personalizado
    if (myMessage != null) {

        // Se establece el mensaje personalizado
        loadingMessage = myMessage
    }
```

Fuente: Autores del proyecto.

Figura 52 Loading Dialog



```
// se configura para que el dialog se muestre sobre la actividad donde se llama
val builder = AlertDialog.Builder(activity);
val inflater = activity!!.layoutInflater;
val customView: View = inflater.inflate(R.layout.custom_dialog,  root: null);
messageView = customView.findViewById<View>(R.id.loadingText) as TextView;

// Agregamos el texto
messageView!!.text = loadingMessage;
builder.setView(customView);
builder.setCancelable(false);
dialog = builder.create();
```

Fuente: Autores del proyecto.

Assistence Form Activity: clase para hacer las respectivas verificaciones en cuanto a las actividades del usuario.

Figura 53 Assistence Form Activity

```
// instancia para verificar si el usuario ya esta autenticado
private lateinit var auth: FirebaseAuth;
private lateinit var mDatabase: DatabaseReference;

// instanciamos la clase de funciones generales
lateinit var GlobalMethods: GlobalMethods;
lateinit var loadingDialog: LoadingDialog;

// referenciamos los elementos
var myAssistenceFormToolbar: Toolbar? = null;
```

Fuente: Autores del proyecto.

Figura 54 Assistence Form Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_assistence_form);

    // Incializamos la instancia con Firebase autentication y database realtime
    auth = Firebase.auth;
    mDatabase = Firebase.database.getReference();

    // inicializaos el toolbar
    myAssistenceFormToolbar = findViewById(R.id.toolbar_assistence_form_bar);

    // se agrega el toolbar
    setSupportActionBar(findViewById(R.id.toolbar_assistence_form_bar));

    //Set Home screen icon
    getSupportActionBar()?.setHomeAsUpIndicator(R.drawable.ic_baseline_arrow_back_32_orange);

    // Mostramos la opción de regresar
    getSupportActionBar()?.setDisplayHomeAsUpEnabled(true);
```

Fuente: Autores del proyecto.

Figura 55 Assistence Form Activity

```
/**
 * Permite establecer las opciones personalizadas
 **/
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    return true;
}

/**
 * Permite definir el evento a las opciones del menu superios
 **/
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    // Se valida si se realizo clic sobre la ventana de la flecha de atras
    if (item.getItemId() === android.R.id.home) {

        // Regresamos al panel anterior
        backMainAssistence();
    }

    return super.onOptionsItemSelected(item);
}
```

Fuente: Autores del proyecto.

Figura 56 Assistence Form Activity

```
/**
 * Permite regresar al panel anterior
 */
fun backMainAssistence(){

    // mostramos la ventana de confirmación
    GlobalMethods.showConfirm("¿Seguro(a) desea cancelar el registro de asistencia?", DialogInterface.OnClickListener

        // Mostramos la actividad
        startActivity(Intent( packageContext: this, AssistenceMainActivity::class.java));

        // Cerramos esta actividad
        finish();
    });
```

Fuente: Autores del proyecto.

Figura 57 Assistence Form Activity

```
/**
 * Permite ejecutar un evento al oprimir la fecha de atras
 */
override fun onBackPressed() {

    // Regresamos al panel anterior
    backMainAssistence();
}
```

Fuente: Autores del proyecto.

Assitence Main Activity: A diferencia de la clase anterior aquí se hace la verificación en cuanto a las autenticaciones por parte del usuario en conjunto de Firebase authentication y real time database.

Figura 58 Assitence Main Activity

```
// instancia para verificar si el usuario ya esta autenticado
private lateinit var auth: FirebaseAuth;
private lateinit var mDatabase: DatabaseReference;

// instanciamos la clase de funciones generales
lateinit var GlobalMethods: GlobalMethods;
lateinit var loadingDialog: LoadingDialog;

// referenciamos los elementos
var myAssistenceMainToolbar: Toolbar? = null;
var btnShowAssistencePanel: View? = null;
```

Fuente: Autores del proyecto.

Figura 59 Assitence Main Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_assistence_main);

    // Incializamos la instancia con Firebase autentication y database realtime
    auth = Firebase.auth;
    mDatabase = Firebase.database.getReference();

    // inicializaos el toolbar
    myAssistenceMainToolbar = findViewById(R.id.toolbar_assistence_main_bar);
    btnShowAssistencePanel = findViewById(R.id.btnShowPanelAssistence);

    // se agrega el toolbar
    setSupportActionBar(findViewById(R.id.toolbar_assistence_main_bar));

    //Set Home screen icon
    getSupportActionBar()?.setHomeAsUpIndicator(R.drawable.ic_baseline_arrow_back_32_orange);

    // Mostramos la opción de regresar
    getSupportActionBar()?.setDisplayHomeAsUpEnabled(true);

    // establecemos el tiulo del app
    this.setTitle("");
```

Fuente: Autores del proyecto.

Figura 60 Assitence Main Activity

```
// iniciamos la clase
GlobalMethods = GlobalMethods( activity: this);

// Se inicializa el dialog
loadingDialog = LoadingDialog( activity: this, "Obteniendo lista de asistencia...");

// agregamos el ebvento al panel de crear usuario
btnShowAssistencePanel?.setOnClickListener { view ->

    // Mostramos la actividad del formulario de asistenncia
    startActivity(Intent( packageContext: this, AssistenceFormActivity::class.java));

    // Cerramos esta actividad
    finish();
}
```

Fuente: Autores del proyecto.

Figura 61 Assitence Main Activity

```
/**
 * Permite establecer las opciones personalizadas
 **/
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    return true;
}

/**
 * Permite definir el evento a las opciones del menu superios
 **/
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    // Se valida si se realizo clic sobre la ventana de la flecha de attras
    if (item.getItemId() === android.R.id.home) {

        // Mostramos la actividad
        startActivity(Intent( packageContext: this, MainActivity::class.java));

        // Cerramos esta actividad
        finish();
    }
```

Fuente: Autores del proyecto.

Figura 62 Assitence Main Activity

```
    return super.onOptionsItemSelected(item);
}

/**
 * Permite ejecutar un evento al oprimir la fecha de atras
 */
override fun onBackPressed() {

    // Mostramos la actividad
    startActivity(Intent( packageContext: this, MainActivity::class.java));

    // Cerramos esta actividad
    finish();
}
```

Fuente: Autores del proyecto.

Login Activity: en esta clase se encuentran todas las actividades en cuanto al login y de la misma manera el manejo de sesiones.

Figura 63 Assistence Login Activity

```
// instancia para verificar si el usuario ya esta autenticado
private lateinit var auth: FirebaseAuth;
private lateinit var mDatabase: DatabaseReference;

// referenciamos los campos del Layaut para el login
var emailLoginElement: EditText? = null;
var passwordLoginElement: EditText? = null;
lateinit var buttonLoginElement: Button;

// instanciamos la clase de funciones generales
lateinit var GlobalMethods: GlobalMethods;
lateinit var loadingDialog: LoadingDialog;
```

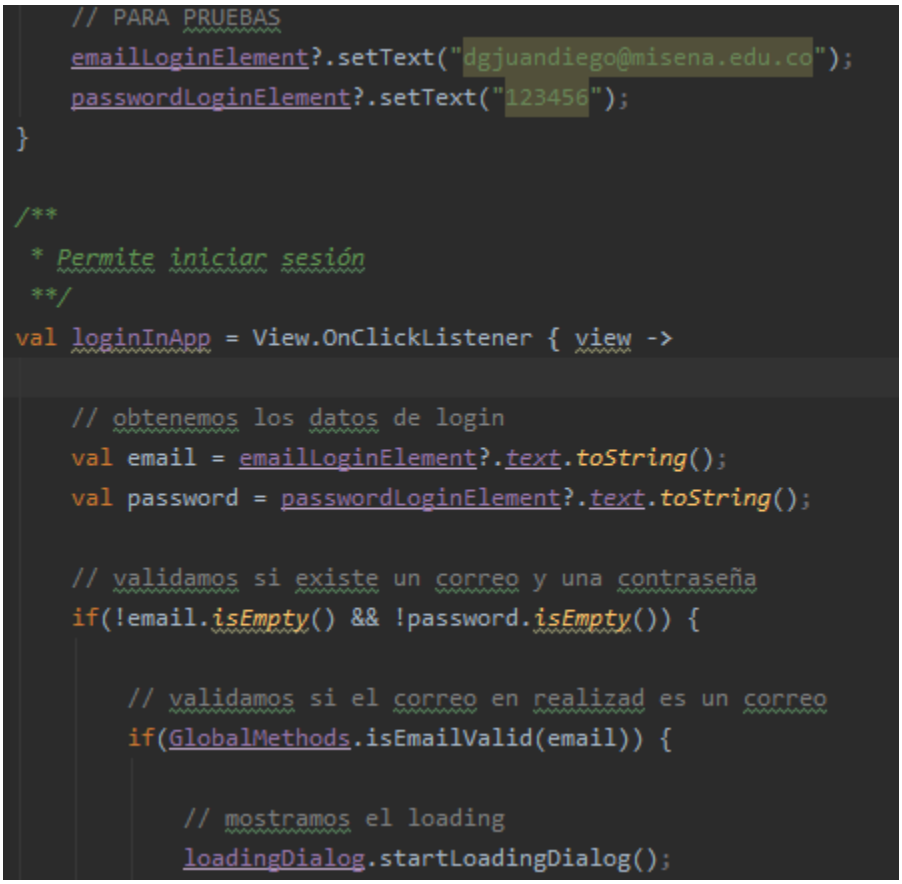
Fuente: Autores del proyecto.

Figura 64 Assistence Login Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_login);

    // Incializamos la instancia con Firebase autentication y database realtime
    auth = Firebase.auth;
    mDatabase = Firebase.database.getReference();

    // iniciamos la referencia de los campos
    emailLoginElement = findViewById(R.id.txtLoginEmail);
    passwordLoginElement = findViewById(R.id.txtLoginPassword);
    buttonLoginElement = findViewById(R.id.btnLoginApp);

    // asignamos el evento click al boton de login
    buttonLoginElement.setOnClickListener(loginInApp);

    // iniciamos la clase
    GlobalMethods = GlobalMethods( activity: this);

    // Se inicializa el dialog
    loadingDialog = LoadingDialog( activity: this, "Iniciando sesión...");
```

Fuente: Autores del proyecto.

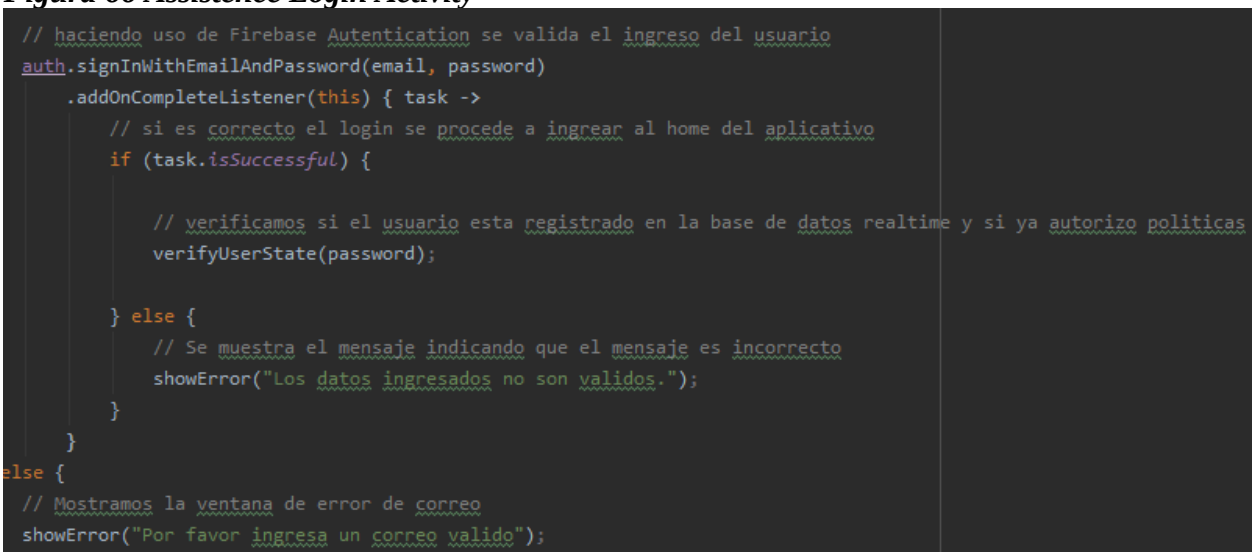
Figura 65 Assistence Login Activity



```
    // PARA PRUEBAS
    emailLoginElement?.setText("dgjuandiego@misena.edu.co");
    passwordLoginElement?.setText("123456");
}

/**
 * Permite iniciar sesión
 **/
val loginInApp = View.OnClickListener { view ->

    // obtenemos los datos de login
    val email = emailLoginElement?.text.toString();
    val password = passwordLoginElement?.text.toString();

    // validamos si existe un correo y una contraseña
    if(!email.isEmpty() && !password.isEmpty()) {

        // validamos si el correo en realizad es un correo
        if(GlobalMethods.isEmailValid(email)) {

            // mostramos el loading
            loadingDialog.startLoadingDialog();
```

Fuente: Autores del proyecto.

Figura 66 Assistence Login Activity



```
// haciendo uso de Firebase Autentication se valida el ingreso del usuario
auth.signInWithEmailAndPassword(email, password)
    .addOnCompleteListener(this) { task ->
        // si es correcto el login se procede a ingrear al home del aplicativo
        if (task.isSuccessful) {

            // verificamos si el usuario esta registrado en la base de datos realtime y si ya autorizo politicas
            verifyUserState(password);

        } else {
            // Se muestra el mensaje indicando que el mensaje es incorrecto
            showError("Los datos ingresados no son validos.");
        }
    }
else {
 // Mostramos la ventana de error de correo
 showError("Por favor ingresa un correo valido");
```

Fuente: Autores del proyecto.

Figura 67 Assistence Login Activity

```
  } else {
      // Mostramos la ventana de error de correo
      showError("Por favor ingresa un correo valido");
  }
else {
  println("mal")
  // Mostramos la ventana de error
  showError("Por favor ingresa el correo y contraseña para continuar");
```

Fuente: Autores del proyecto.

Figura 68 Assistence Login Activity

```
 * Permite verificar el tipo de perfil, y si ya autorizo politicas
 */
private fun verifyUserState(password: String) {

    // obtenemos el id del usuario
    val userId: String = auth.currentUser.uid;

    // consultamos si existe información del usuario
    mDatabase.child( pathString: "users").child(userId).get().addOnSuccessListener {  it: DataSnapshot!
        try{

            // Obtenemos los datos del usuario
            val userDataLogin: UserObject? = it.getValue(UserObject::class.java);

            // Ocultamos el loading
            loadingDialog.dissmissDialog();

            // realizamos la apertura de la actividad principal
            var intent: Intent? = null;

            // Agregamos en una propieda la constraseña del usuario activo
            GlobalMethods.addPropertyValue( activity: this,  property: "password_curr", password);

            // validamos si el usuario ya se logueo
            if(userDataLogin?.policy!!){
```

Fuente: Autores del proyecto.

Figura 69 Assistence Login Activity

```
    // se valida si el usuario tiene el rol de "coach"
    if(userDataLogin.profile == "coach") {
        // realizamos la apertura de la actividad principal
        intent = Intent( packageContext: this, MainActivity::class.java);
    } else {
        // realizamos la apertura de la actividad principal del jugador
        intent = Intent( packageContext: this, PlayerActivity::class.java);
    }
} else {

    // realizamos la apertura de la actividad principal
    intent = Intent( packageContext: this, PolicyActivity::class.java);
}

// Mostramos la actividad
startActivity(intent);

// Cerramos esta actividad
finish();
```

Fuente: Autores del proyecto.

Figura 70 Assistence Login Activity

```
}.addOnFailureListener{  it: Exception

    // ELiminamos la propiedad en la que guarda la contraseña
    GlobalMethods.removeProperty( activity: this,  property: "password_curr");

    // cerramos sesión por si la inicio
    Firebase.auth.signOut();

    // Se muestra el mensaje indicando que el mensaje es incorrecto
    showError("Los datos ingresados no son validos." + it);
}
```

Fuente: Autores del proyecto.

Figura 71 Assistence Login Activity

```
private fun showError(message: String) {

    // definimos el mensaje a mostrar
    var messageText = message;

    // Ocultamos el loading
    loadingDialog.dissmissDialog();

    // validamos si el mensaje viene vacio
    if (messageText.isEmpty()) {

        // Tomamos un mensaje por defecto
        messageText = "Lo sentimos no fue posible procesar su solicitud";
    }

    // Mostramos la ventana de error
    GlobalMethods.showAlert( title: "Mensaje de error", message,  listener: null);
}
```

Fuente: Autores del proyecto.

New Student Activity: por medio de esta clase se hace la verificación de que si el usuario ya se encuentra autenticado para subir los archivos al storage.

Figura 72 New Student Activity

```
// referenciamos los elementos
var myHomeToolbar: Toolbar? = null;

// referenciamos los campos del Layaut para la creación de un estudiante nuevo
var nameStudentElement: EditText? = null;
var emailStudentElement: EditText? = null;
var passwordStudentElement: EditText? = null;
var birthDayStudentElement: EditText? = null;
var heightStudentElement: EditText? = null;
var weightStudentElement: EditText? = null;
lateinit var buttonStudentElement: Button;
lateinit var buttonImageStudentElement: ImageButton;
lateinit var imageViewStudentElement: ImageView;

// instanciamos la clase de funciones generales
lateinit var GlobalMethods: GlobalMethods;
lateinit var loadingDialog: LoadingDialog;
```

Fuente: Autores del proyecto.

Figura 73 New Student Activity

```
// Variable para controlar permisos
private val MY_PERMISSIONS: Int = 100;
private val PHOTO_CODE = 200;
private val SELECT_PICTURE = 300;
private val MEDIA_DIRECTORY = "/DirectedBaskedApp";
private var mPath: String? = null;
```

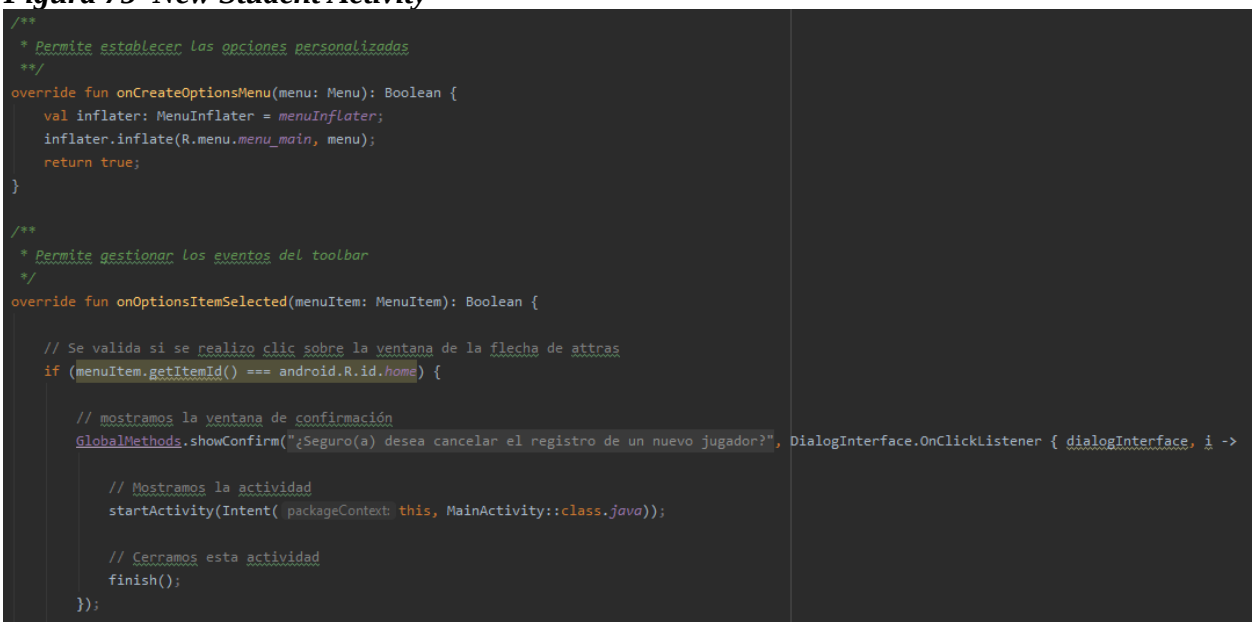
Fuente: Autores del proyecto.

Figura 74 New Student Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_new_student);

    // Incializamos la instancia con Firebase autentication y database realtime
    auth = Firebase.auth;
    mDatabase = Firebase.database.getReference();
    mStorage = Firebase.storage;

    // inicializaos el toolbar
    myHomeToolbar = findViewById(R.id.toolbar_student_bar);

    // se agrega el toolbar
    setSupportActionBar(findViewById(R.id.toolbar_student_bar));

    //Set Home screen icon
    getSupportActionBar()?.setHomeAsUpIndicator(R.drawable.ic_baseline_arrow_back_32_orange);

    // Mostramos la opción de regresar
    getSupportActionBar()?.setDisplayHomeAsUpEnabled(true);

    // establecemos el tiulo del app
    this.setTitle("");

    // iniciamos la clase
    GlobalMethods = GlobalMethods( activity: this);

    // Se inicializa el dialog
    loadingDialog = LoadingDialog(
        activity: this,
        "Registrando usuario..."
```

Fuente: Autores del proyecto.

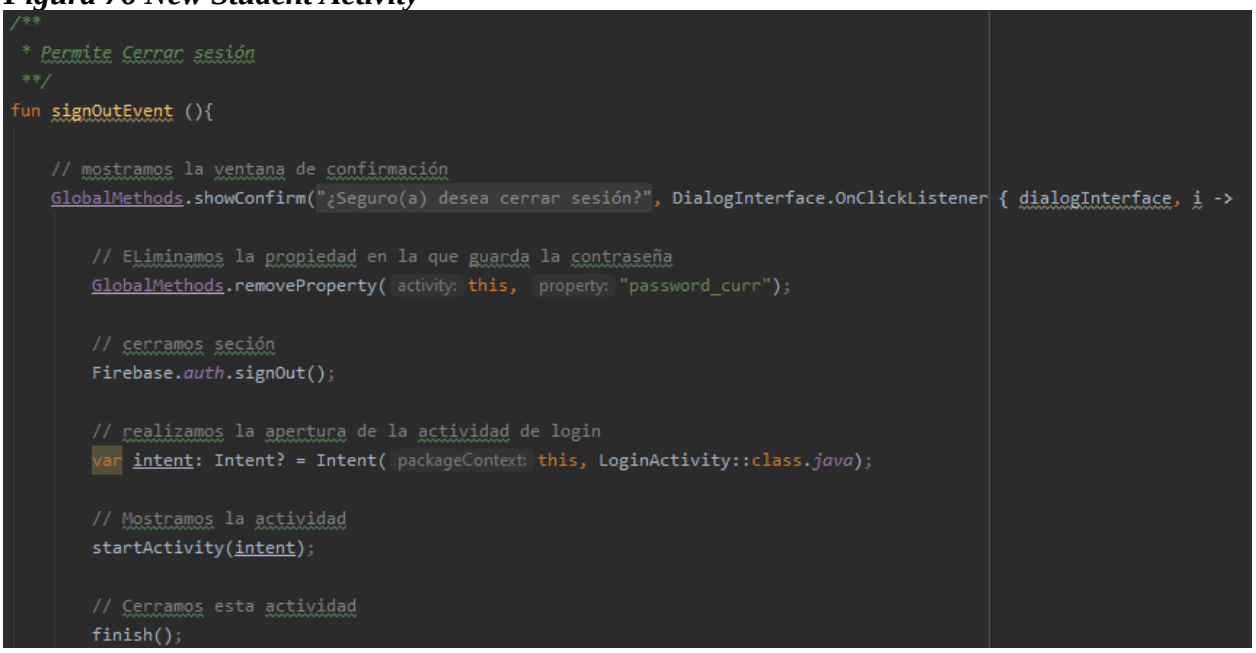
Figura 75 New Student Activity



```
/**
 * Permite establecer las opciones personalizadas
 **/
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    val inflater: MenuInflater = menuInflater;
    inflater.inflate(R.menu.menu_main, menu);
    return true;
}

/**
 * Permite gestionar los eventos del toolbar
 */
override fun onOptionsItemSelected(menuItem: MenuItem): Boolean {

    // Se valida si se realizo clic sobre la ventana de la flecha de attras
    if (menuItem.getItemId() === android.R.id.home) {

        // mostramos la ventana de confirmación
        GlobalMethods.showConfirm("¿Seguro(a) desea cancelar el registro de un nuevo jugador?", DialogInterface.OnClickListener { dialogInterface, i ->

            // Mostramos la actividad
            startActivity(Intent( packageContext: this, MainActivity::class.java));

            // Cerramos esta actividad
            finish();
        });
```

Fuente: Autores del proyecto.

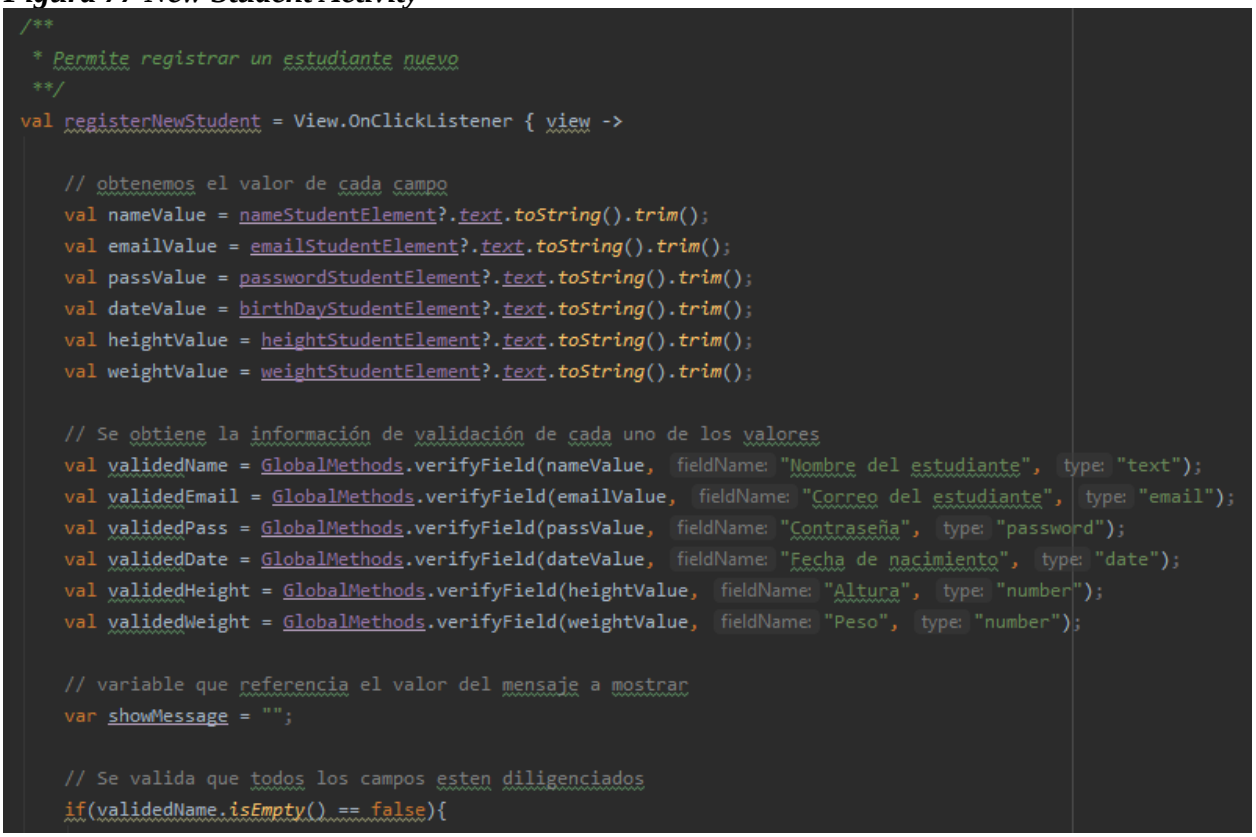
Figura 76 New Student Activity



```
/**
 * Permite Cerrar sesión
 **/
fun signOutEvent (){

    // mostramos la ventana de confirmación
    GlobalMethods.showConfirm("¿Seguro(a) desea cerrar sesión?", DialogInterface.OnClickListener { dialogInterface, i ->

        // ELiminamos la propiedad en la que guarda la contraseña
        GlobalMethods.removeProperty( activity: this,  property: "password_curr");

        // cerramos seción
        Firebase.auth.signOut();

        // realizamos la apertura de la actividad de login
        var intent: Intent? = Intent( packageContext: this, LoginActivity::class.java);

        // Mostramos la actividad
        startActivity(intent);

        // Cerramos esta actividad
        finish();
```

Fuente: Autores del proyecto.

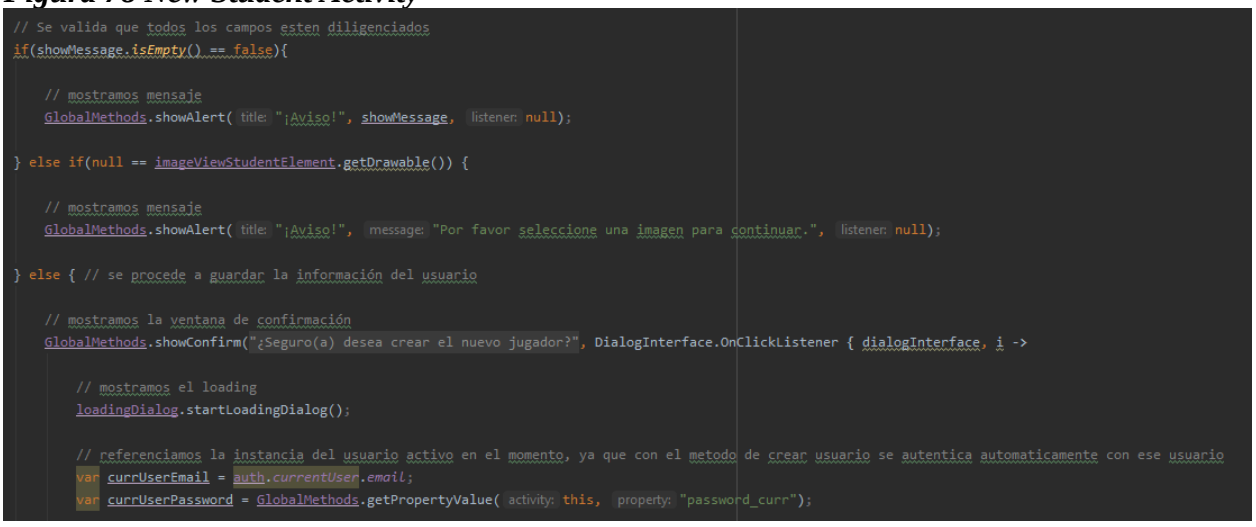
Figura 77 New Student Activity



```
/**
 * Permite registrar un estudiante nuevo
 **/
val registerNewStudent = View.OnClickListener { view ->

    // obtenemos el valor de cada campo
    val nameValue = nameStudentElement?.text.toString().trim();
    val emailValue = emailStudentElement?.text.toString().trim();
    val passValue = passwordStudentElement?.text.toString().trim();
    val dateValue = birthDayStudentElement?.text.toString().trim();
    val heightValue = heightStudentElement?.text.toString().trim();
    val weightValue = weightStudentElement?.text.toString().trim();

    // Se obtiene la información de validación de cada uno de los valores
    val validedName = GlobalMethods.verifyField(nameValue, fieldName: "Nombre del estudiante", type: "text");
    val validedEmail = GlobalMethods.verifyField(emailValue, fieldName: "Correo del estudiante", type: "email");
    val validedPass = GlobalMethods.verifyField(passValue, fieldName: "Contraseña", type: "password");
    val validedDate = GlobalMethods.verifyField(dateValue, fieldName: "Fecha de nacimiento", type: "date");
    val validedHeight = GlobalMethods.verifyField(heightValue, fieldName: "Altura", type: "number");
    val validedWeight = GlobalMethods.verifyField(weightValue, fieldName: "Peso", type: "number");

    // variable que referencia el valor del mensaje a mostrar
    var showMessage = "";

    // Se valida que todos los campos esten diligenciados
    if(validedName.isEmpty() == false){
```

Fuente: Autores del proyecto.

Figura 78 New Student Activity



```
// Se valida que todos los campos esten diligenciados
if(showMessage.isEmpty() == false){

    // mostramos mensaje
    GlobalMethods.showAlert( title: "¡Aviso!", showMessage, listener: null);

} else if(null == imageViewStudentElement.getDrawable()) {

    // mostramos mensaje
    GlobalMethods.showAlert( title: "¡Aviso!", message: "Por favor seleccione una imagen para continuar.", listener: null);

} else { // se procede a guardar la información del usuario

    // mostramos la ventana de confirmación
    GlobalMethods.showConfirm("¿Seguro(a) desea crear el nuevo jugador?", DialogInterface.OnClickListener { dialogInterface, i ->

        // mostramos el loading
        loadingDialog.startLoadingDialog();

        // referenciamos la instancia del usuario activo en el momento, ya que con el metodo de crear usuario se autentica automaticamente con ese usuario
        var currUserEmail = auth.currentUser.email;
        var currUserPassword = GlobalMethods.getPropertyValue( activity: this, property: "password_curr");
```

Fuente: Autores del proyecto.

Figura 79 New Student Activity

```
// se procede a crear un nuevo usuario
auth.createUserWithEmailAndPassword(emailValue, passValue)
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            // obtenemos la información del nuevo usuario
            val newUserID = auth.currentUser.uid;

            // Cerramos el inicio de sesión automatico del nuevo usuario
            Firebase.auth.signOut();

            // volvemos a iniciar seción con el usuario actual
            auth.signInWithEmailAndPassword(currUserEmail, currUserPassword)
                .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {

                    // Se procede a subir la imagen del usuario
                    uploadImageInStorage(newUserID, passValue, nameValue, emailValue, dateValue, heightValue, weightValue);

                } else {
                    // Se muestra el mensaje indicando que el mensaje es incorrecto
                    showError("Lo sentimos no es posible continuar con el proceso, por favor cierra el aplicativo y vuelve a iniciar sesión.");
                }
            };
```

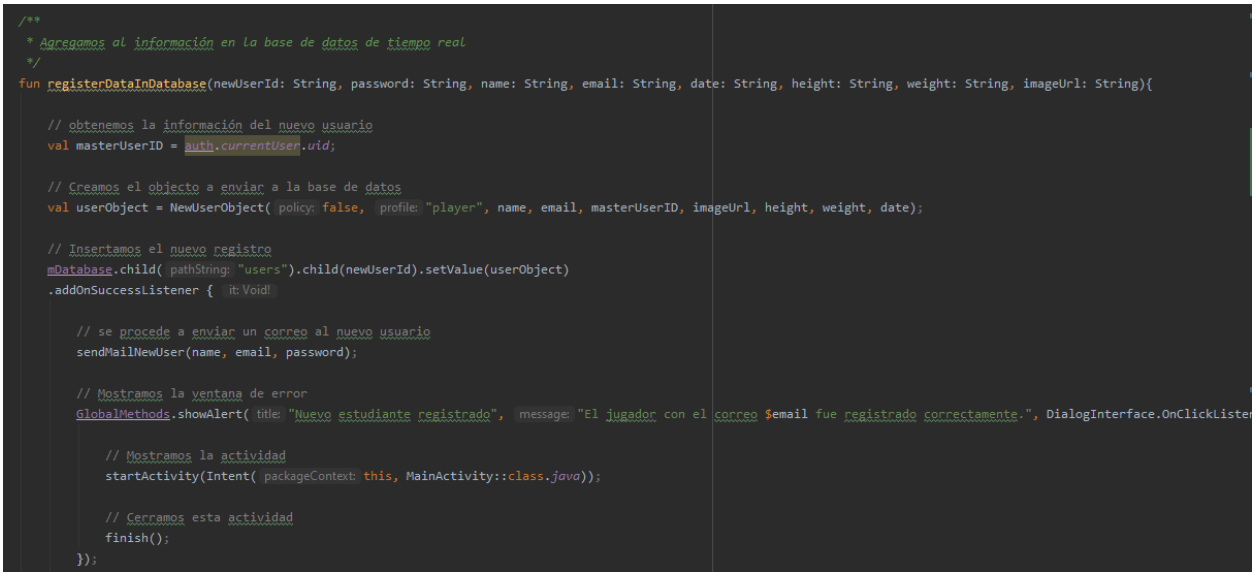
Fuente: Autores del proyecto.

Figura 80 New Student Activity

```
/**
 * Permite subir la imagen a firebase Storage
 */
fun uploadImageInStorage(newUserId: String, password: String, name: String, email: String, date: String, height: String, weight: String){

    // obtenemos el dato de la imagen
    imageViewStudentElement.invalidate();
    val drawable = imageViewStudentElement.getDrawable() as BitmapDrawable;
    val bitmap = drawable.bitmap;
    val baos = ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.JPEG,  quality: 100, baos);
    val data = baos.toByteArray();

    // creamos la referencia del buchet a donde deseamos guardar la imagen
    val storageRef = mStorage.reference;

    // Definimos el nombre
    val fileName = newUserId +"_profile.jpg"

    // refenciamos el hijo que sera donde queda guardado
    val child = storageRef.child(fileName);

    // creamos la tarea que permite proceder a subir la imagen
    child?.putBytes(data).addOnCompleteListener( activity: this) { task ->
        if (task.isSuccessful) {
```

Fuente: Autores del proyecto.

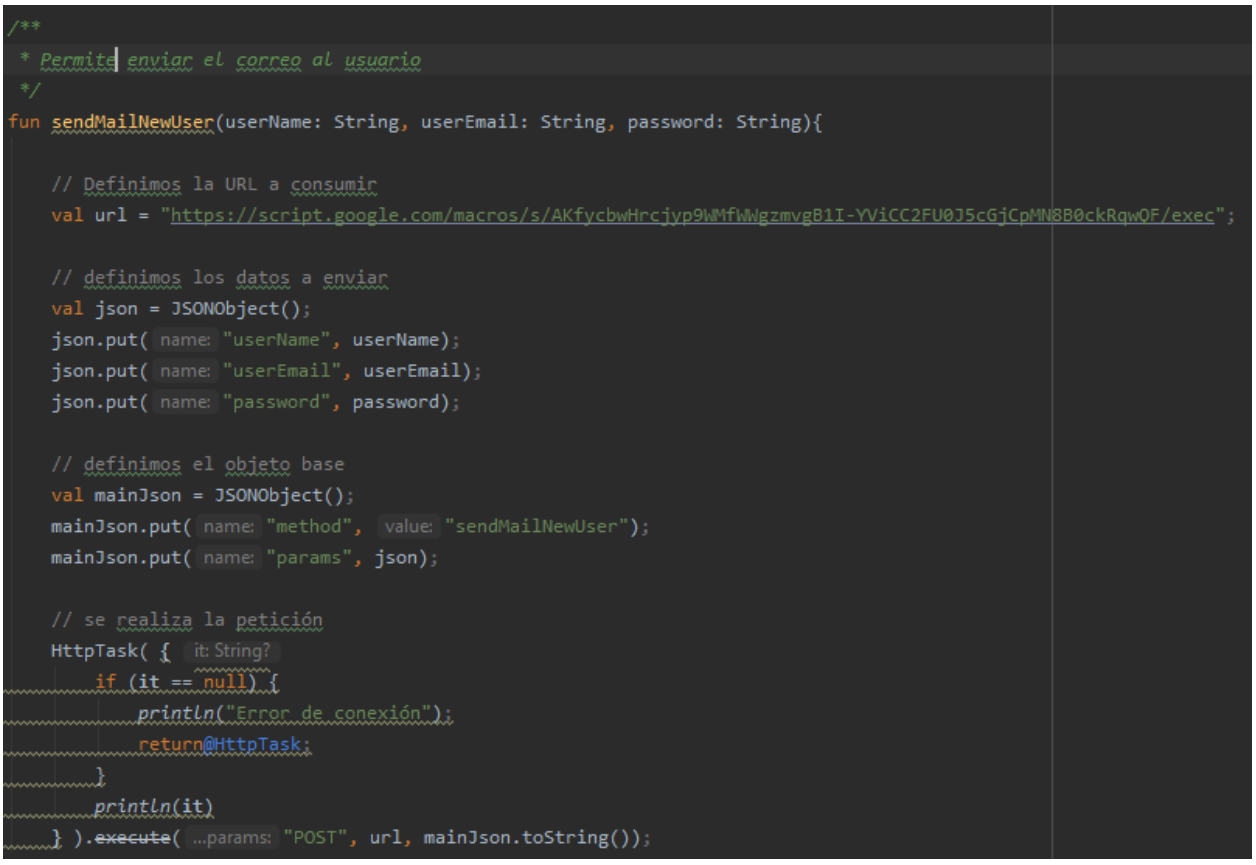
Figura 81 New Student Activity



```
/**
 * Agregamos al información en la base de datos de tiempo real
 */
fun registerDataInDatabase(newUserId: String, password: String, name: String, email: String, date: String, height: String, weight: String, imageUrl: String){

    // obtenemos la información del nuevo usuario
    val masterUserID = auth.currentUser.uid;

    // Creamos el objecto a enviar a la base de datos
    val userObject = NewUserObject( policy: false,  profile: "player", name, email, masterUserID, imageUrl, height, weight, date);

    // Insertamos el nuevo registro
    mDatabase.child( pathString: "users").child(newUserId).setValue(userObject)
    .addOnSuccessListener {  it: Void!

        // se procede a enviar un correo al nuevo usuario
        sendMailNewUser(name, email, password);

        // Mostramos la ventana de error
        GlobalMethods.showAlert( title: "Nuevo estudiante registrado",  message: "El jugador con el correo $email fue registrado correctamente.", DialogInterface.OnClickListe

            // Mostramos la actividad
            startActivity(Intent( packageContext: this, MainActivity::class.java));

            // Cerramos esta actividad
            finish();
        });
```

Fuente: Autores del proyecto.

Figura 82 New Student Activity



```
/**
 * Permite enviar el correo al usuario
 */
fun sendMailNewUser(userName: String, userEmail: String, password: String){

    // Definimos la URL a consumir
    val url = "https://script.google.com/macros/s/AKfycbwHrcjyp9WMfWWgzmvgB1I-YViCC2FU0J5cGjCpMN8B0ckRqwOF/exec";

    // definimos los datos a enviar
    val json = JSONObject();
    json.put( name: "userName", userName);
    json.put( name: "userEmail", userEmail);
    json.put( name: "password", password);

    // definimos el objeto base
    val mainJson = JSONObject();
    mainJson.put( name: "method",  value: "sendMailNewUser");
    mainJson.put( name: "params", json);

    // se realiza la petición
    HttpTask( {  it: String?
        if (it == null) {
            println("Error de conexión");
            return@HttpTask;
        }
        println(it)
    } ).execute( ...params: "POST", url, mainJson.toString());
```

Fuente: Autores del proyecto.

Figura 83 New Student Activity

```
/**
 * Metodo que permite mostrar un mensaje
 */
private fun showError(message: String) {

    // definimos el mensaje a mostrar
    var messageText = message;

    // Ocultamos el loading
    loadingDialog.dissmissDialog();

    // validamos si el mensaje viene vacio
    if (messageText.isEmpty()) {

        // Tomamos un mensaje por defecto
        messageText = "Lo sentimos no fue posible procesar su solicitud";
    }

    // Mostramos la ventana de error
    GlobalMethods.showAlert( title: "Mensaje de error", message,  listener: null);
}
```

Fuente: Autores del proyecto.

Figura 84 New Student Activity

```
/**
 * Permite abrir la actividad de imagen
 */
private fun openCamera() {

    // Se cera la vista para mostar la captura de una foto
    val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE);

    // Se muestra la vista
    startActivityForResult(intent, PHOTO_CODE);
}
```

Fuente: Autores del proyecto.

Figura 85 New Student Activity

```
/**
 * Permite abrir el selector de imagenes
 **/
val showSelectorImage = View.OnClickListener { view ->

    // Definimos las opciones
    val option = arrayOf<CharSequence>("Tomar foto", "Elegir de galeria", "Cancelar")
    val builder: AlertDialog.Builder = AlertDialog.Builder( context: this);
    builder.setTitle("Eleige una opción")
    builder.setItems(option, DialogInterface.OnClickListener { dialog, which ->
        if (option[which] === "Tomar foto") {
            openCamera();
        } else if (option[which] === "Elegir de galeria") {
            val intent = Intent(
                Intent.ACTION_PICK,
                MediaStore.Images.Media.EXTERNAL_CONTENT_URI
            )
            intent.type = "image/*";
            startActivityForResult(Intent.createChooser(intent,  title: "Selecciona una imagen"), SELECT_PICTURE);
        } else {
            dialog.dismiss();
            imageViewStudentElement.setVisibility(View.GONE);
            imageViewStudentElement.setImageDrawable(null);
        }
    });

    builder.show();
}
```

Fuente: Autores del proyecto.

Figura 86 New Student Activity

```
/**
 * Gestionar la respuesta post verificación de permisos
 **/
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String?>, grantResults: IntArray) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == MY_PERMISSIONS) {
        if (grantResults.size == 2 && grantResults[0] == PackageManager.PERMISSION_GRANTED && grantResults[1] == PackageManager.PERMISSION_GRANTED
        ) {
            Toast.makeText( context: this,  text: "Permisos aceptados", Toast.LENGTH_SHORT).show();

            // Habilita el boton
            buttonImageStudentElement.setEnabled(true);
            buttonStudentElement.setEnabled(true);
        }
    } else {
        showExplanation();
    }
}
```

Fuente: Autores del proyecto.

Figura 87 New Student Activity

```
/**
 * Permite mostrar una interfaz explicando el motivo de los permisos
 */
private fun showExplanation() {
    val builder = AlertDialog.Builder( context: this);
    builder.setTitle("Permisos denegados");
    builder.setMessage("Para usar las funciones de la app necesitas aceptar los permisos");
    builder.setPositiveButton(
        text: "Aceptar"
    ) { dialog, which ->
        val intent = Intent()
        intent.action = Settings.ACTION_APPLICATION_DETAILS_SETTINGS
        val uri =
            Uri.fromParts( scheme: "package", packageName,  fragment: null)
        intent.data = uri
        startActivity(intent)
    }
    builder.setNegativeButton(
        text: "Cancelar"
    ) { dialog, which ->
        dialog.dismiss()
        finish()
    }
    builder.show();
```

Fuente: Autores del proyecto.

Figura 88 New Student Activity

```
/**
 * Permite revisar si el usuario ya concedio permisos para seleccionar archivos o tomar una foto
 */
private fun verifyStoragePermission(): Boolean {

    // Se valida si la versión de andriod es menor a la M,e s decir que no es necesario solicitar permisos, ya que al momento de instalar la aplicación concedio estos
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.M) return true;

    // se chequea los permisos necesarios
    if (checkSelfPermission(WRITE_EXTERNAL_STORAGE) == PackageManager.PERMISSION_GRANTED && checkSelfPermission(CAMERA) == PackageManager.PERMISSION_GRANTED) {
        return true;
    }

    // Se valida si un permiso esta pendiente por mostrar para que lo apruebe o lo deniege
    if (shouldShowRequestPermissionRationale(WRITE_EXTERNAL_STORAGE) || shouldShowRequestPermissionRationale(CAMERA)) {

        // referenciamos el contexto actual
        val contextView = findViewById<View>(android.R.id.content);

        // Mostramos un mensaje indicando la lista de permisos necesarios
        Snackbar.make(contextView, text: "Los permisos son necesarios para poder elegir la foto del jugador", Snackbar.LENGTH_INDEFINITE).setAction( text: "AUTORIZAR") {
            requestPermissions(
                arrayOf(
                    WRITE_EXTERNAL_STORAGE,
                    CAMERA
                ), MY_PERMISSIONS
            )
        }.show();
```

Fuente: Autores del proyecto.

Figura 89 New Student Activity

```
/**
 * Configuramos el campos de fecha
 */
private fun settingDate() {

    // Obtenemos la fecha actual en formato claro para el usuario
    val currentDateStr: String? = GlobalMethods.longToDateFormatter( Calendar.getInstance().timeInMillis,  isCalculateZone: false);

    // Agregamos la fecha
    birthDayStudentElement?.setText("$currentDateStr")
    //fieldDate.setText("29/12/2020 - 29/01/2021");

    // Se crea la instancia del selector de fecha
    val builder = MaterialDatePicker.Builder.datePicker();

    // Se establece el titulo y el tema
    builder.setTitleText("Seleccionar fecha")
    builder.setTheme(R.style.ThemeOverlay_MaterialComponents_MaterialCalendar)

    // Se compila el selector
    val materialDatePicker = builder.build();

    // Agregamos la acción para cuando el usuarios
    materialDatePicker.addOnPositiveButtonClickListener {  it: Long!
        // Convertimos la fecha en un string para mostrarlo en el campo
        val dateSelected = GlobalMethods.longToDateFormatter(it,  isCalculateZone: true);

        // Agregamos la fecha
        birthDayStudentElement?.setText("$dateSelected");
    }
```

Fuente: Autores del proyecto.

Figura 90 New Student Activity

```
// Agregamos el evento para cuando seleccione el valor
birthDayStudentElement?.setOnClickListener { // Obtenemos el valor actual en el campo
    val currentDateText = birthDayStudentElement?.text.toString();

    // Establecemos como selección la fecha actual
    val currentTimeInMillis = Calendar.getInstance().timeInMillis;

    // se define que fecha se va agregar
    var setStartTime = currentTimeInMillis;

    // se valida si existe un valor en el campos
    if (!currentDateText.isEmpty()) {
        setStartTime = GlobalMethods.stringToDate(currentDateText,  format: "dd/MM/yyyy")?.time!!;
    }

    // Agregamos la fecha actual o la que existe en el campo
    builder.setSelection(setStartTime);

    // Volvemos a compilar para que tome la fecha seleccionada
    builder.build();

    // Mostramso el selector
    materialDatePicker.show(supportFragmentManager, materialDatePicker.toString());
}
```

Fuente: Autores del proyecto.

Figura 91 New Student Activity

```
/**
 * Permite ejecutar un evento al oprimir la fecha de atras
 */
override fun onBackPressed() {

    // mostramos la ventana de confirmación
    GlobalMethods.showConfirm("¿Seguro(a) desea cancelar el registro de un nuevo jugador?", DialogInterface.OnClickListener { dialogInterface, i ->

        // Mostramos la actividad
        startActivity(Intent( packageContext: this, MainActivity::class.java));

        // Cerramos esta actividad
        finish();
    });
}
```

Fuente: Autores del proyecto.

Policy Activity: en esta clase se hace la referencia del layout de igual forma se hace la verificación del usuario autenticado y se instancia la clase de funciones generales.

Figura 92 Policy Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_policy);

    // iniciamos la referencia de los campos
    policyTextView = findViewById(R.id.textViewPolicy);
    buttonCancelElement = findViewById(R.id.btnPolicyCancel);
    buttonOkElement = findViewById(R.id.btnPolicyOk);

    // establecemos el contenido
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
        policyTextView?.setText(Html.fromHtml(getPolicyHtml(), Html.FROM_HTML_MODE_COMPACT));
    } else {
        policyTextView?.setText(Html.fromHtml(getPolicyHtml()));
    }
```

Fuente: Autores del proyecto.

Figura 93 Policy Activity

```
// asignamos el evento a los botones
buttonCancelElement.setOnClickListener(closeApp);
buttonOkElement.setOnClickListener(aprovePolicy);

// Incializamos la instancia con Firebase autentication y realtime
auth = Firebase.auth;
mDatabase = Firebase.database.getReference();

// iniciamos la clase
GlobalMethods = GlobalMethods( activity: this)

// Se inicializa el dialog
loadingDialog = LoadingDialog( activity: this, "Aceptando terminos y condiciones...");
```

Fuente: Autores del proyecto.

Figura 94 Policy Activity

```
/**
 * Permite aprobar las politicas
 **/
val aprovePolicy = View.OnClickListener { view ->

    // obtenemos el id del usuario
    val userId: String = auth.currentUser.uid;

    // mostramos el loading
    loadingDialog.startLoadingDialog();

    // consultamos si existe información del usuario
    mDatabase.child( pathString: "users").child(userId).get().addOnSuccessListener {  it: DataSnapshot!

        // Obtenemos los datos del usuario
        val userDataPolicy: UserObject? = it.getValue(UserObject::class.java);

        // actualizamos el estado del usuario
        mDatabase.child( pathString: "users").child(userId).child( pathString: "policy").setValue(true).addOnSuccessListener {  it: Void!

            // realizamos la apertura de la actividad principal
            var intent: Intent? = null;

            // se valida si el usuario tiene el rol de "coach"
            if(userDataPolicy?.profile == "coach") {
                // realizamos la apertura de la actividad principal
                intent = Intent( packageContext: this, MainActivity::class.java);
            } else {
                // realizamos la apertura de la actividad principal del jugador
                intent = Intent( packageContext: this, PlayerActivity::class.java);
            }
```

Fuente: Autores del proyecto.

Figura 95 Policy Activity

```
        // Ocultamos el loading
        loadingDialog.dissmissDialog();

        // Mostramos la actividad
        startActivity(intent);

        // Cerramos esta actividad
        finish();

    }.addOnFailureListener{ it: Exception

        // Se muestra el mensaje indicando que el mensaje es incorrecto
        showError("Lo sentimos no fue posible almacenar la aprobación de las políticas por el siguiente motivo: " + it);
    };

}.addOnFailureListener{ it: Exception

    // Se muestra el mensaje indicando que el mensaje es incorrecto
    showError("Lo sentimos no fue posible almacenar la aprobación de las políticas por el siguiente motivo: " + it);
}
```

Fuente: Autores del proyecto.

Figura 96 Policy Activity

```
/**
 * Metodo que permite mostrar un mensaje
 */
private fun showError(message: String) {

    // definimos el mensaje a mostrar
    var messageText = message;

    // Ocultamos el loading
    loadingDialog.dissmissDialog();

    // validamos si el mensaje viene vacio
    if (messageText.isEmpty()) {

        // Tomamos un mensaje por defecto
        messageText = "Lo sentimos no fue posible procesar su solicitud"
    }

    // Mostramos la ventana de error
    GlobalMethods.showAlert( title: "Mensaje de error", message,  listener: null);
}
```

Fuente: Autores del proyecto.

Figura 97 Policy Activity

```
/**
 * Permite salir del aplicativo
 **/
val closeApp = View.OnClickListener { view ->

    // cerramos sesión
    Firebase.auth.signOut();

    // ELiminamos la propiedad en la que guarda la contraseña
    GlobalMethods.removeProperty( activity: this, property: "password_curr");

    // Validamos la versión con el objetivo de cerar por completo la aplicación
    if (Build.VERSION.SDK_INT >= 16 && Build.VERSION.SDK_INT < 21) {
        finishAffinity();
    } else if (Build.VERSION.SDK_INT >= 21) {
        finishAndRemoveTask();
    } else {
        finish();
    }

    // Salimos del aplicativo
    System.exit( status: 0);
}
```

Fuente: Autores del proyecto.

Figura 98 Policy Activity

```
    /**
     * Permite obtener el Html del consentimiento
     */
    fun getPolicyHtml(): String? {
        var inputStream: InputStream? = null
        try {
            inputStream = getAssets().open( fileName: "policy.html"
            val r =
                BufferedReader(InputStreamReader(inputStream))
            val total = StringBuilder()
            var line: String?
            while (r.readLine().also { line = it } != null) {
                total.append(line).append("\n")
            }
            return total.toString()
        } catch (e: IOException) {}
        return "";
    }
}
```

Fuente: Autores del proyecto.

Psychologic End Activity: por medio de esta clase se verifica si el usuario ya está autenticado para subir archivos al storage de igual forma aquí se inicializa la instancia con firebase athentication y database realtime.

Figura 99 Psychologic End Activity

```
// instanciamos la clase de funciones generales
var G_CURR_DATA: JSONObject = JSONObject();
lateinit var GlobalMethods: GlobalMethods;
lateinit var loadingDialog: LoadingDialog;

private var radGroupOne: RadioGroup? = null;
private var radGroupTwo: RadioGroup? = null;
private var radGroupThree: RadioGroup? = null;
private var radGroupFour: RadioGroup? = null;
private var radGroupFive: RadioGroup? = null;
private lateinit var selectedRadioButton: RadioButton;
```

Fuente: Autores del proyecto.

Figura 100 Psychologic End Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_psychologic_end);

    // Incializamos la instancia con Firebase autentication y database realtime
    auth = Firebase.auth;
    mDatabase = Firebase.database.getReference();

    // Obtenemos la información del id del usuario
    val bundle = intent.extras;
    G_CURR_DATA = JSONObject(bundle?.getString( key: "currData").toString());

    // referenciamos los campos
    btnNext = findViewById(R.id.btnNextForm6);
    btnPrev = findViewById(R.id.btnPrevForm6);
    radGroupOne = findViewById(R.id.radGroupOne);
    radGroupTwo = findViewById(R.id.radGroupTwo);
    radGroupThree = findViewById(R.id.radGroupThree);
    radGroupFour = findViewById(R.id.radGroupFour);
    radGroupFive = findViewById(R.id.radGroupFive);
```

Fuente: Autores del proyecto.

Figura 101 Psychologic End Activity

```
// agregamos los valores actuales
setValueRadio(radGroupOne!!,  property: "answer51");
setValueRadio(radGroupTwo!!,  property: "answer52");
setValueRadio(radGroupThree!!,  property: "answer53");
setValueRadio(radGroupFour!!,  property: "answer54");
setValueRadio(radGroupFive!!,  property: "answer55");

// asignamos el evento click al boton de continuar o regresar
btnNext.setOnClickListener(nextPanel);
btnPrev.setOnClickListener(prevPanel);

// iniciamos la clase
GlobalMethods = GlobalMethods( activity: this);

// Se inicializa el dialog
loadingDialog = LoadingDialog(
     activity: this,
    "Almacenando datos psicológicos..."
);
```

Fuente: Autores del proyecto.

Figura 102 Psychologic End Activity

```
/**
 * Permite avanzar al panel anterior
 **/
val prevPanel = View.OnClickListener { view ->

    // Se refresa a la vista anterior
    backView();
}

/**
 * Agregamos los valores en vada una de las opciones
 **/
fun setValueRadio(groupRadio: RadioGroup, property: String){
    try {
        // obtenemos el valor a insertar
        var value = G_CURR_DATA.getString(property);

        // se valida si ya existe
        if(value != "") {

            // obtenemos el id del radio seleccionado
            var opcionI2: RadioButton = groupRadio.getChildAt(value.toInt()) as RadioButton;

            // marcamos el radio button
            opcionI2.isChecked = true;
        }
    } catch (e: Exception) {}
}
```

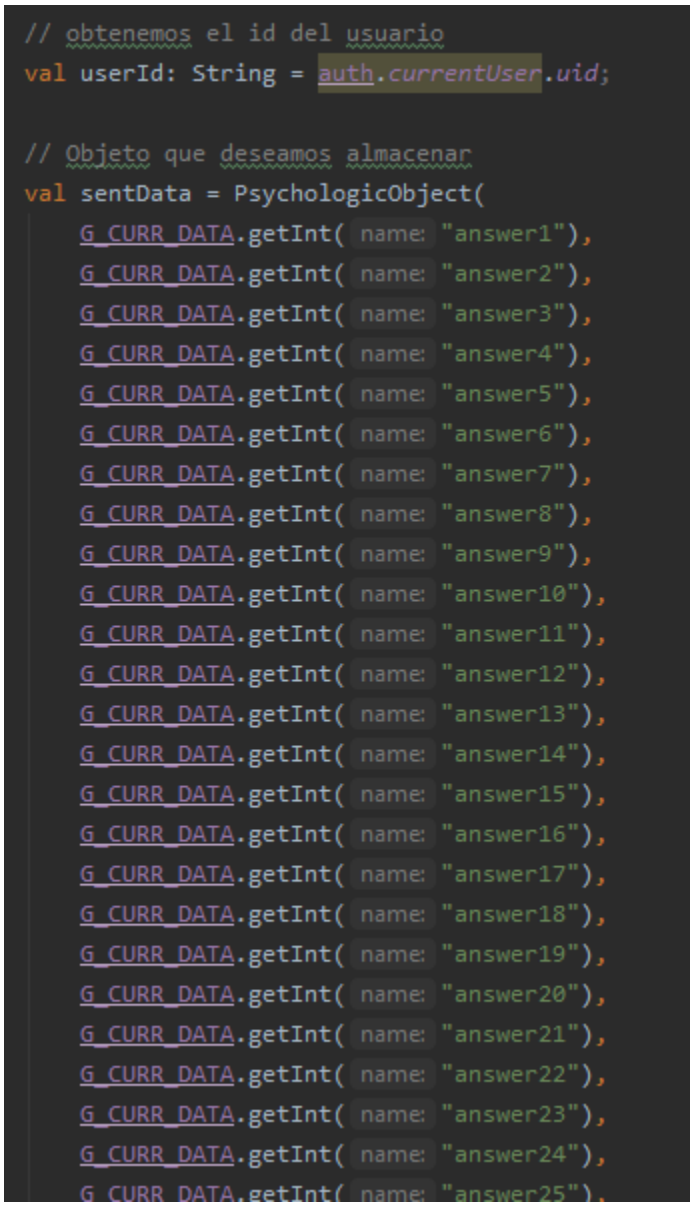
Fuente: Autores del proyecto.

Figura 103 Psychologic End Activity

```
/**
 * Permite avanzar al siguiente panel
 **/
val nextPanel = View.OnClickListener { view ->

    var answer1: Int? = radGroupOne?.let { getRadioValue(it) };
    var answer2: Int? = radGroupTwo?.let { getRadioValue(it) };
    var answer3: Int? = radGroupThree?.let { getRadioValue(it) };
    var answer4: Int? = radGroupFour?.let { getRadioValue(it) };
    var answer5: Int? = radGroupFive?.let { getRadioValue(it) };

    // validamos si todos los radios estan seleccionados
    if(answer1 != -1 && answer2 != -1 && answer3 != -1 && answer4 != -1 && answer5 != -1) {

        // agregamos cada una de las propiedades
        G_CURR_DATA.put( name: "answer51", answer1);
        G_CURR_DATA.put( name: "answer52", answer2);
        G_CURR_DATA.put( name: "answer53", answer3);
        G_CURR_DATA.put( name: "answer54", answer4);
        G_CURR_DATA.put( name: "answer55", answer5);

        // Obtenemos la fecha actual
        val currDate = DateFormat.format( inFormat: "yyyy-MM-dd HH:mm:ss", Date()).toString();

        // agregamos la fecha al objeto
        G_CURR_DATA.put( name: "date", currDate);
```

Fuente: Autores del proyecto.

Figura 104 Psychologic End Activity



```
// obtenemos el id del usuario
val userId: String = auth.currentUser.uid;

// Objeto que deseamos almacenar
val sentData = PsychologicObject(
    G_CURR_DATA.getInt( name: "answer1"),
    G_CURR_DATA.getInt( name: "answer2"),
    G_CURR_DATA.getInt( name: "answer3"),
    G_CURR_DATA.getInt( name: "answer4"),
    G_CURR_DATA.getInt( name: "answer5"),
    G_CURR_DATA.getInt( name: "answer6"),
    G_CURR_DATA.getInt( name: "answer7"),
    G_CURR_DATA.getInt( name: "answer8"),
    G_CURR_DATA.getInt( name: "answer9"),
    G_CURR_DATA.getInt( name: "answer10"),
    G_CURR_DATA.getInt( name: "answer11"),
    G_CURR_DATA.getInt( name: "answer12"),
    G_CURR_DATA.getInt( name: "answer13"),
    G_CURR_DATA.getInt( name: "answer14"),
    G_CURR_DATA.getInt( name: "answer15"),
    G_CURR_DATA.getInt( name: "answer16"),
    G_CURR_DATA.getInt( name: "answer17"),
    G_CURR_DATA.getInt( name: "answer18"),
    G_CURR_DATA.getInt( name: "answer19"),
    G_CURR_DATA.getInt( name: "answer20"),
    G_CURR_DATA.getInt( name: "answer21"),
    G_CURR_DATA.getInt( name: "answer22"),
    G_CURR_DATA.getInt( name: "answer23"),
    G_CURR_DATA.getInt( name: "answer24"),
    G_CURR_DATA.getInt( name: "answer25"),
```

Fuente: Autores del proyecto.

Figura 105 Psychologic End Activity

```
// mostramos la ventana de confirmación
GlobalMethods.showConfirm("¿Seguro(a) desea almacenar el estado psicológico del ju...", DialogInterface.OnClickListener { dialogInterface, i ->

    // mostramos el loading
    loadingDialog.startLoadingDialog();

    // Insertamos el nuevo registro
    mDatabase.child( pathString: "users").child(userId).child( pathString: "psychologicData").push().setValue(sentData)
        .addOnSuccessListener {  it: Void!

            // Mostramos la ventana de error
            GlobalMethods.showAlert( title: "Aspectos físicos",  message: "Se ha registrado el estado psicológico del juegador correctamente.", Dial

                // Mostramos la actividad
                startActivity(Intent( packageContext: this, PlayerActivity::class.java));

                // Cerramos esta actividad
                finish();
            });

            // Ocultamos el loading
            loadingDialog.dissmissDialog();

        }.addOnFailureListener {  it: Exception
            // mostramos el error
            showError("No es posible registar los datos en este momento por el siguiente motivo: '$it'.");
        }
```

Fuente: Autores del proyecto.

Figura 106 Psychologic End Activity

```
/**
 * Metodo que permite mostrar un mensaje
 */
private fun showError(message: String) {

    // definimos el mensaje a mostrar
    var messageText = message;

    // Ocultamos el loading
    loadingDialog.dissmissDialog();

    // validamos si el mensaje viene vacio
    if (messageText.isEmpty()) {

        // Tomamos un mensaje por defecto
        messageText = "Lo sentimos no fue posible procesar su solicitud";
    }

    // Mostramos la ventana de error
    GlobalMethods.showAlert( title: "Mensaje de error", message,  listener: null);
}
```

Fuente: Autores del proyecto.

Figura 107 Psychologic End Activity

```
/**
 * Permite Obtener el valor de un radioButton
 * */
fun getRadioValue(groupRadio: RadioGroup): Int{

    // obtenemos el id del radio seleccionado
    val selectedRadioButtonId = groupRadio?.checkedRadioButtonId;

    // se valida realmente existe
    if (selectedRadioButtonId != -1) {
        selectedRadioButton = findViewById(selectedRadioButtonId);

        // retornamos el número
        return selectedRadioButton.text.toString().toInt();
    } else {
        return -1;
    }
}
```

Fuente: Autores del proyecto.

Figura 108 Psychologic End Activity

```
/**
 * Permite regresar a la vista anterior
 */
fun backView(){

    // realizamos la apertura de la vista 5
    var intent = Intent( packageContext: this, PsychologicFiveActivity::class.java);

    // agregamos el id del usuario que deseamo procesar
    intent.putExtra( name: "currData", G_CURR_DATA.toString());

    // Mostramos la actividad
    startActivity(intent);

    // Cerramos esta actividad
    finish();
}

/**
 * Permite ejecutar un evento al oprimir la fecha de atras
 */
override fun onBackPressed() {

    // Se refresa a la vista anterior
    backView();
}
```

Fuente: Autores del proyecto.

Psychologic Five Activity: por medio de esta clase se hace la instancia de funciones generales en lo que corresponde a la actividad cinco de esta misma.

Figura 109 Psychologic Five Activity

```
// se inicializa las variables de continuar o regresar
lateinit var btnNext: Button;
lateinit var btnPrev: Button;

// instanciamos la clase de funciones generales
var G_CURR_DATA: JSONObject = JSONObject();
lateinit var GlobalMethods: GlobalMethods;

private var radGroupOne: RadioGroup? = null;
private var radGroupTwo: RadioGroup? = null;
private var radGroupThree: RadioGroup? = null;
private var radGroupFour: RadioGroup? = null;
private var radGroupFive: RadioGroup? = null;
private var radGroupSix: RadioGroup? = null;
private var radGroupSeven: RadioGroup? = null;
private var radGroupEight: RadioGroup? = null;
private var radGroupNine: RadioGroup? = null;
private var radGroupTen: RadioGroup? = null;
private lateinit var selectedRadioButton: RadioButton;
```

Fuente: Autores del proyecto.

Figura 110 Psychologic Five Activity

```
// Obtenemos la información del id del usuario
val bundle = intent.extras;
G_CURR_DATA = JSONObject(bundle?.getString( key: "currData").toString());

// referenciamos los campos
btnNext = findViewById(R.id.btnNextForm5);
btnPrev = findViewById(R.id.btnPrevForm5);
radGroupOne = findViewById(R.id.radGroupOne);
radGroupTwo = findViewById(R.id.radGroupTwo);
radGroupThree = findViewById(R.id.radGroupThree);
radGroupFour = findViewById(R.id.radGroupFour);
radGroupFive = findViewById(R.id.radGroupFive);
radGroupSix = findViewById(R.id.radGroupSix);
radGroupSeven = findViewById(R.id.radGroupSeven);
radGroupEight = findViewById(R.id.radGroupEight);
radGroupNine = findViewById(R.id.radGroupNine);
radGroupTen = findViewById(R.id.radGroupTen);
```

Figura 111 Psychologic Five Activity

```
// agregamos los valores actuales
setValueRadio(radGroupOne!!,  property: "answer41");
setValueRadio(radGroupTwo!!,  property: "answer42");
setValueRadio(radGroupThree!!,  property: "answer43");
setValueRadio(radGroupFour!!,  property: "answer44");
setValueRadio(radGroupFive!!,  property: "answer45");
setValueRadio(radGroupSix!!,  property: "answer46");
setValueRadio(radGroupSeven!!,  property: "answer47");
setValueRadio(radGroupEight!!,  property: "answer48");
setValueRadio(radGroupNine!!,  property: "answer49");
setValueRadio(radGroupTen!!,  property: "answer50");

// asignamos el evento click al boton de continuar o regresar
btnNext.setOnClickListener(nextPanel);
btnPrev.setOnClickListener(prevPanel);

// iniciamos la clase
GlobalMethods = GlobalMethods( activity: this);
```

Fuente: Autores del proyecto.

Figura 112 Psychologic Five Activity

```
 * Permite avanzar al panel anterior
 **/
val prevPanel = View.OnClickListener { view ->

    // Se refresa a la vista anterior
    backView();
}

/**
 * Agregamos los valores en vada una de las opciones
 **/
fun setValueRadio(groupRadio: RadioGroup, property: String){
    try {
        // obtenemos el valor a insertar
        var value = G_CURR_DATA.getString(property);

        // se valida si ya existe
        if(value != "") {

            // obtenemos el id del radio seleccionado
            var opcionI2: RadioButton = groupRadio.getChildAt(value.toInt()) as RadioButton;

            // marcamos el radio button
            opcionI2.isChecked = true;
        }
    } catch (e: Exception) {}
}
```

Figura 113 Psychologic Five Activity

```
/**
 * Permite avanzar al siguiente panel
 **/
val nextPanel = View.OnClickListener { view ->

    var answer1: Int? = radGroupOne?.let { getRadioValue(it) };
    var answer2: Int? = radGroupTwo?.let { getRadioValue(it) };
    var answer3: Int? = radGroupThree?.let { getRadioValue(it) };
    var answer4: Int? = radGroupFour?.let { getRadioValue(it) };
    var answer5: Int? = radGroupFive?.let { getRadioValue(it) };
    var answer6: Int? = radGroupSix?.let { getRadioValue(it) };
    var answer7: Int? = radGroupSeven?.let { getRadioValue(it) };
    var answer8: Int? = radGroupEight?.let { getRadioValue(it) };
    var answer9: Int? = radGroupNine?.let { getRadioValue(it) };
    var answer10: Int? = radGroupTen?.let { getRadioValue(it) };

    // validamos si todos los radios estan seleccionados
    if(answer1 != -1 && answer2 != -1 && answer3 != -1 && answer4 != -1

        // agregamos cada una de las propiedades
```

Fuente: Autores del proyecto.

Figura 114 Psychologic Five Activity

```
// agregamos cada una de las propiedades
G_CURR_DATA.put( name: "answer41", answer1);
G_CURR_DATA.put( name: "answer42", answer2);
G_CURR_DATA.put( name: "answer43", answer3);
G_CURR_DATA.put( name: "answer44", answer4);
G_CURR_DATA.put( name: "answer45", answer5);
G_CURR_DATA.put( name: "answer46", answer6);
G_CURR_DATA.put( name: "answer47", answer7);
G_CURR_DATA.put( name: "answer48", answer8);
G_CURR_DATA.put( name: "answer49", answer9);
G_CURR_DATA.put( name: "answer50", answer10);

// definimos cual es la actividad a mostrar
var intent = GlobalMethods.getStaticPage();
```

Fuente: Autores del proyecto.

Figura 115 Psychologic Five Activity

```
    // definimos cual es la actividad a mostrar
    var intent = GlobalMethods.getStaticPage();

    // Pasamos los datos actuales la siguiente página a mostrar
    intent.putExtra( name: "currData", G_CURR_DATA.toString());
    intent.putExtra( name: "nextPage",  value: "6");

    // Mostramos la actividad
    startActivity(intent);

    // terminamos la actividad actual
    finish();
} else {

    // Mostramos la ventana de error
    GlobalMethods.showAlert( title: "Mensaje de error",  message: "Por favor
}
```

Fuente: Autores del proyecto.

Figura 116 Psychologic Five Activity

```
/**
 * Permite Obtener el valor de un radioButton
 * */
fun getRadioValue(groupRadio: RadioGroup): Int{

    // obtenemos el id del radio seleccionado
    val selectedRadioButtonId = groupRadio?.checkedRadioButtonId;

    // se valida realmente existe
    if (selectedRadioButtonId != -1) {
        selectedRadioButton = findViewById(selectedRadioButtonId);

        // retornamos el número
        return selectedRadioButton.text.toString().toInt();
    } else {
        return -1;
    }
}
```

Fuente: Autores del proyecto.

Figura 117 Psychologic Five Activity

```
/**
 * Permite regresar a la vista anterior
 */
fun backView(){

    // realizamos la apertura de la vista 4
    var intent = Intent( packageContext: this, PsychologicFourActivity::class.java);

    // agregamos el id del usuario que deseamo procesar
    intent.putExtra( name: "currData", G_CURR_DATA.toString());

    // Mostramos la actividad
    startActivity(intent);

    // Cerramos esta actividad
    finish();
}

/**
 * Permite ejecutar un evento al oprimir la fecha de atras
 */
override fun onBackPressed() {

    // Se refresa a la vista anterior
    backView();
}
```

Fuente: Autores del proyecto.

Psychologic Four Activity: por medio de esta clase se inicializan las variables de continuar o regresar De igual forma se instancia las clases de funciones generales.

Figura 118 Psychologic Four Activity

```
class PsychologicFourActivity : AppCompatActivity() {

    // se inicializa las variables de continuar o regresar
    lateinit var btnNext: Button;
    lateinit var btnPrev: Button;

    // instanciamos la clase de funciones generales
    var G_CURR_DATA: JSONObject = JSONObject();
    lateinit var GlobalMethods: GlobalMethods;

    private var radGroupOne: RadioGroup? = null;
    private var radGroupTwo: RadioGroup? = null;
    private var radGroupThree: RadioGroup? = null;
    private var radGroupFour: RadioGroup? = null;
    private var radGroupFive: RadioGroup? = null;
    private var radGroupSix: RadioGroup? = null;
    private var radGroupSeven: RadioGroup? = null;
    private var radGroupEight: RadioGroup? = null;
    private var radGroupNine: RadioGroup? = null;
    private var radGroupTen: RadioGroup? = null;
    private lateinit var selectedRadioButton: RadioButton;
```

Fuente: Autores del proyecto.

Figura 119 Psychologic Four Activity

```
// Obtenemos la información del id del usuario
val bundle = intent.extras;
G_CURR_DATA = JSONObject(bundle?.getString( key: "currData").toString());

// referenciamos los campos
btnNext = findViewById(R.id.btnNextForm4);
btnPrev = findViewById(R.id.btnPrevForm4);
radGroupOne = findViewById(R.id.radGroupOne);
radGroupTwo = findViewById(R.id.radGroupTwo);
radGroupThree = findViewById(R.id.radGroupThree);
radGroupFour = findViewById(R.id.radGroupFour);
radGroupFive = findViewById(R.id.radGroupFive);
radGroupSix = findViewById(R.id.radGroupSix);
radGroupSeven = findViewById(R.id.radGroupSeven);
radGroupEight = findViewById(R.id.radGroupEight);
radGroupNine = findViewById(R.id.radGroupNine);
radGroupTen = findViewById(R.id.radGroupTen);
```

Fuente: Autores del proyecto.

Figura 120 Psychologic Four Activity

```
// agregamos los valores actuales
setValueRadio(radGroupOne!!, property: "answer31");
setValueRadio(radGroupTwo!!, property: "answer32");
setValueRadio(radGroupThree!!, property: "answer33");
setValueRadio(radGroupFour!!, property: "answer34");
setValueRadio(radGroupFive!!, property: "answer35");
setValueRadio(radGroupSix!!, property: "answer36");
setValueRadio(radGroupSeven!!, property: "answer37");
setValueRadio(radGroupEight!!, property: "answer38");
setValueRadio(radGroupNine!!, property: "answer39");
setValueRadio(radGroupTen!!, property: "answer40");

// asignamos el evento click al boton de continuar o regresar
btnNext.setOnClickListener(nextPanel);
btnPrev.setOnClickListener(prevPanel);

// iniciamos la clase
GlobalMethods = GlobalMethods( activity: this);
```

Fuente: Autores del proyecto.

Figura 121 Psychologic Four Activity

```
    // iniciamos la clase
    GlobalMethods = GlobalMethods( activity: this);
}



/**
 * Permite avanzar al panel anterior
 **/
val prevPanel = View.OnClickListener { view ->

    // Se refresa a la vista anterior
    backView();
}
```

Fuente: Autores del proyecto.

Figura 122 Psychologic Four Activity

```
/**
 * Agregamos los valores en vada una de las opciones
 **/
fun setValueRadio(groupRadio: RadioGroup, property: String){
    try {
        // obtenemos el valor a insertar
        var value = G_CURR_DATA.getString(property);

        // se valida si ya existe
        if(value != "") {

            // obtenemos el id del radio seleccionado
            var opcionI2: RadioButton = groupRadio.getChildAt(value.toInt()) as RadioButton;

            // marcamos el radio button
            opcionI2.isChecked = true;
        }
    } catch (e: Exception) {}
}
```

Fuente: Autores del proyecto.

Figura 123 Psychologic Four Activity

```
/**
 * Permite avanzar al siguiente panel
 **/
val nextPanel = View.OnClickListener { view ->

    var answer1: Int? = radGroupOne?.let { getRadioValue(it) };
    var answer2: Int? = radGroupTwo?.let { getRadioValue(it) };
    var answer3: Int? = radGroupThree?.let { getRadioValue(it) };
    var answer4: Int? = radGroupFour?.let { getRadioValue(it) };
    var answer5: Int? = radGroupFive?.let { getRadioValue(it) };
    var answer6: Int? = radGroupSix?.let { getRadioValue(it) };
    var answer7: Int? = radGroupSeven?.let { getRadioValue(it) };
    var answer8: Int? = radGroupEight?.let { getRadioValue(it) };
    var answer9: Int? = radGroupNine?.let { getRadioValue(it) };
    var answer10: Int? = radGroupTen?.let { getRadioValue(it) };

    // validamos si todos los radios estan seleccionados
    if(answer1 != -1 && answer2 != -1 && answer3 != -1 && answer4 != -1 && answer5 != -1
```

Fuente: Autores del proyecto.

Figura 124 Psychologic Four Activity

```
// agregamos cada una de las propiedades
G_CURR_DATA.put( name: "answer31", answer1);
G_CURR_DATA.put( name: "answer32", answer2);
G_CURR_DATA.put( name: "answer33", answer3);
G_CURR_DATA.put( name: "answer34", answer4);
G_CURR_DATA.put( name: "answer35", answer5);
G_CURR_DATA.put( name: "answer36", answer6);
G_CURR_DATA.put( name: "answer37", answer7);
G_CURR_DATA.put( name: "answer38", answer8);
G_CURR_DATA.put( name: "answer39", answer9);
G_CURR_DATA.put( name: "answer40", answer10);

// definimos cual es la actividad a mostrar
var intent = GlobalMethods.getStaticPage();

// Pasamos los datos actuales la siguiente página a mostrar
intent.putExtra( name: "currData", G_CURR_DATA.toString());
intent.putExtra( name: "nextPage",  value: "5");

// Mostramos la actividad
startActivity(intent);
```

Fuente: Autores del proyecto.

Figura 125 Psychologic Four Activity

```
        // terminamos la actividad actual
        finish();
    } else {

        // Mostramos la ventana de error
        GlobalMethods.showAlert( title: "Mensaje de error",  message: "Por favor seleccionar un ítem
    }
}


/**
 * Permite Obtener el valor de un radioButton
 * */
fun getRadioValue(groupRadio: RadioGroup): Int{

    // obtenemos el id del radio seleccionado
    val selectedRadioButtonId = groupRadio?.checkedRadioButtonId;

    // se valida realmente existe
    if (selectedRadioButtonId != -1) {
        selectedRadioButton = findViewById(selectedRadioButtonId);

        // retornamos el número
        return selectedRadioButton.text.toString().toInt();
    } else {
        return -1;
    }
}
```

Figura 126 Psychologic Four Activity

```
/**
 * Permite regresar a la vista anterior
 */
fun backView(){

    // realizamos la apertura de la tercera sección
    var intent = Intent( packageContext: this, PsychologicThreeActivity::class.java);

    // agregamos el id del usuario que deseamo procesar
    intent.putExtra( name: "currData", G_CURR_DATA.toString());

    // Mostramos la actividad
    startActivity(intent);

    // Cerramos esta actividad
    finish();
}

/**
 * Permite ejecutar un evento al oprimir la fecha de atras
 */
override fun onBackPressed() {

    // Se refresa a la vista anterior
    backView();
}
```

Fuente: Autores del proyecto.

Splash Screen: clase que carga la pantalla de bienvenida en el aplicativo móvil.

Figura 127 Splash Screen

```
package udec.sutatausa.directedbasket;

import ...

internal class SplashScreen : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState);
    }

    /**
     * Metodo que se ejecuta al momento de cargar la intancia de firebase
     */
    public override fun onStart() {
        super.onStart();

        // Mostramos la actividad
        startActivity(Intent( packageContext: this, StartActivity::class.java));

        // Cerramos esta actividad
        finish();
    }
}
```

Fuente: Autores del proyecto.

Tensor Flow Activity: en esta clase se utiliza la librería de tensorflow para el modelo machine learning de igual forma se hace la validación de la recomendación de jugadores.

Figura 128 Tensor Flow Activity

```
/**
 * Permite realizar la validación de la recomendación de jugadores
 **/
fun initTensorflow (){

    // mostramos el loading
    loadingDialog.startLoadingDialog();

    // ocultamos los jugadores
    cardBase?.visibility = View.INVISIBLE;
    cardEscolta?.visibility = View.INVISIBLE;
    cardAlero?.visibility = View.INVISIBLE;
    cardPivot?.visibility = View.INVISIBLE;
    cardPoste?.visibility = View.INVISIBLE;
```

Fuente: Autores del proyecto.

Figura 129 Tensor Flow Activity

```
/**
 * Metodo que permite mostrar un mensaje
 */
private fun showError(message: String) {

    // definimos el mensaje a mostrar
    var messageText = message;

    // Ocultamos el loading
    loadingDialog.dissmissDialog();

    // validamos si el mensaje viene vacio
    if (messageText.isEmpty()) {

        // Tomamos un mensaje por defecto
        messageText = "Lo sentimos no fue posible procesar su solicitud";
    }

    // Mostramos la ventana de error
    GlobalMethods.showAlert( title: "Mensaje de error", message,  listener: null);
}
```

Fuente: Autores del proyecto.

Figura 130 Tensor Flow Activity

```
/**
 * Permite establecer las opciones personalizadas
 **/
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    return true;
}

/**
 * Permite definir el evento a las opciones del menu superios
 **/
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    // Se valida si se realizo clic sobre la ventana de la flecha de attras
    if (item.getItemId() === android.R.id.home) {

        // Mostramos la actividad
        startActivity(Intent( packageContext: this, MainActivity::class.java));

        // Cerramos esta actividad
        finish();
    }

    return super.onOptionsItemSelected(item);
}
```

Fuente: Autores del proyecto.

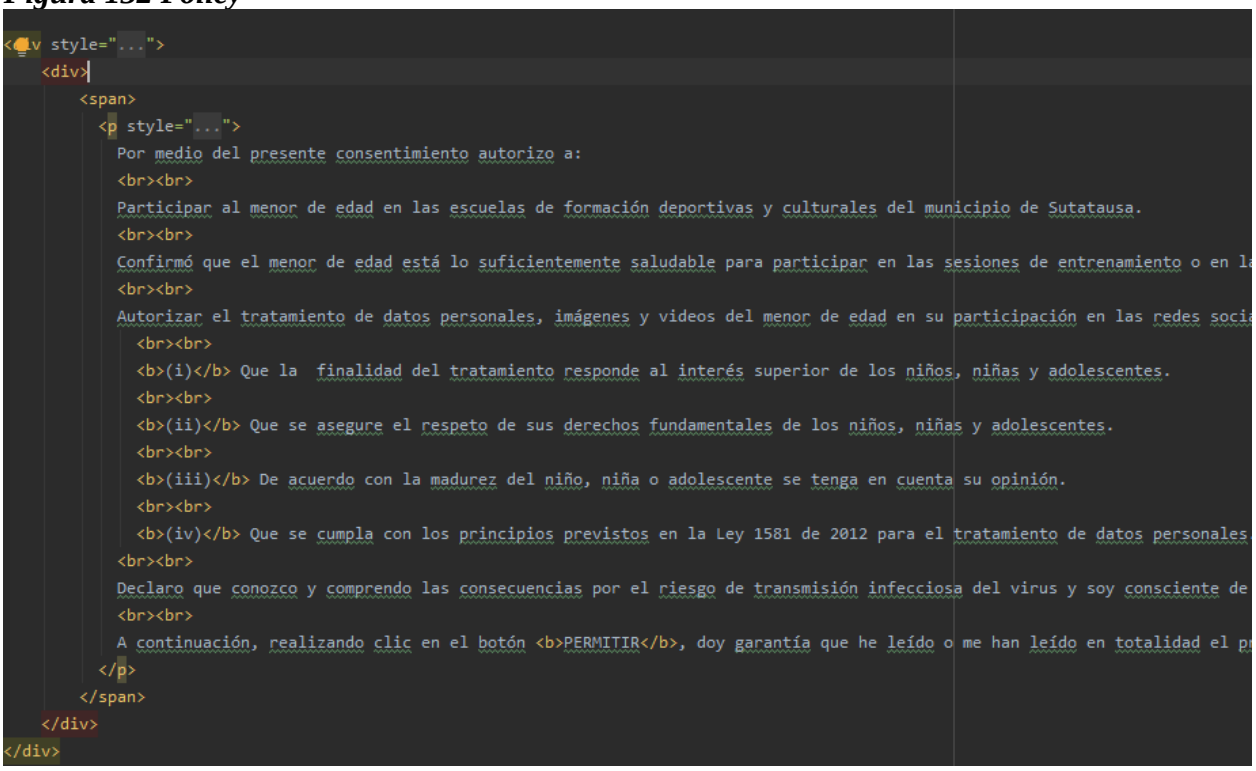
Figura 131 Tensor Flow Activity

```
/**
 * Permite ejecutar un evento al oprimir la fecha de atras
 */
override fun onBackPressed() {

    // Mostramos la actividad
    startActivity(Intent( packageContext: this, MainActivity::class.java));

    // Cerramos esta actividad
    finish();
}
```

Fuente: Autores del proyecto.

Policy.html: en este archivo se encuentra la política de privacidad que aparecerá antes de que el usuario empieza a interactuar con el aplicativo.

Figura 132 Policy



Fuente: Autores del proyecto.

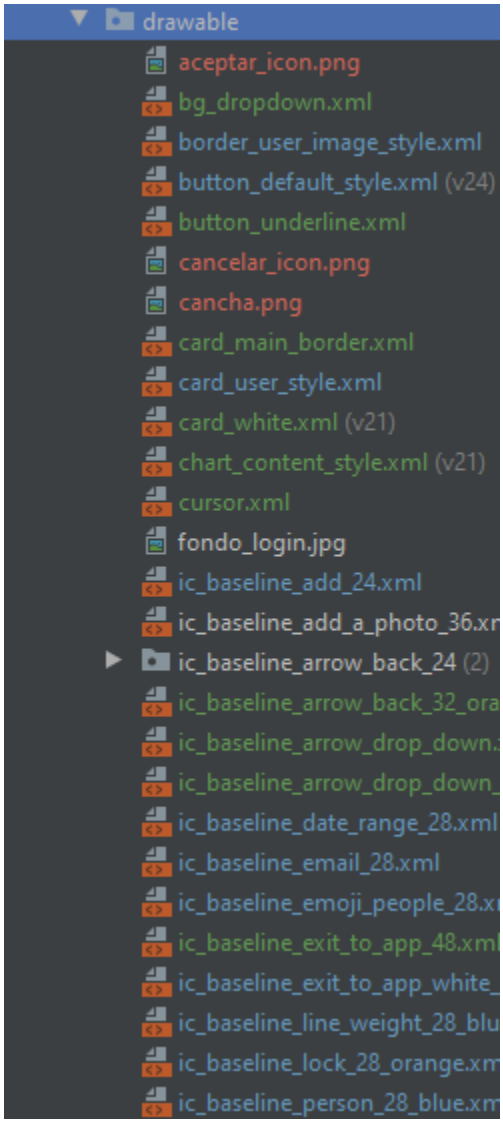
Zoom_in: aquí se encuentran los recursos de animación que están definidas por el archivo XML que es quien modifica las propiedades del objeto.

Figura 133 Zoom_in

Fuente: Autores del proyecto.

Drawable: aquí están los recursos de los elementos en cuanto al diseño por medio de unos gráficos que se pueden dibujar en la pantalla generando un API.

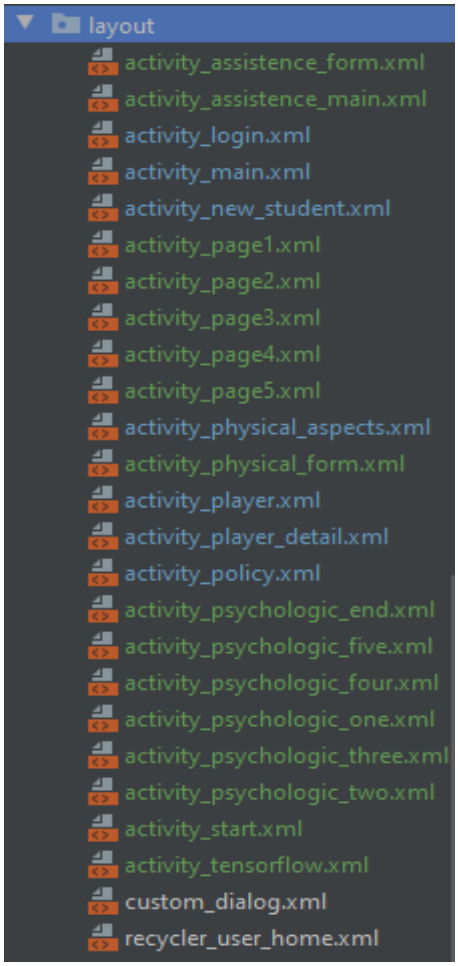
Figura 134 Drawable



Fuente: Autores del proyecto.

Layout: aquí está el contenedor de todas las vistas que tiene el aplicativo móvil y a la vez controla el comportamiento y posición de las funcionalidades de cada módulo.

Figura 135 Layout



Fuente: Autores del proyecto.

Menu: en esta clase se hace uso de las API y a su vez presenta al usuario las acciones y demás opciones en cuanto a las actividades y funcionalidades.

Figura 136 Menu

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/iconLogout"
        android:icon="@drawable/ic_baseline_exit_to_app_48"
        android:title="Cerrar sesión"
        app:showAsAction="always"></item>
</menu>
```

Fuente: Autores del proyecto.

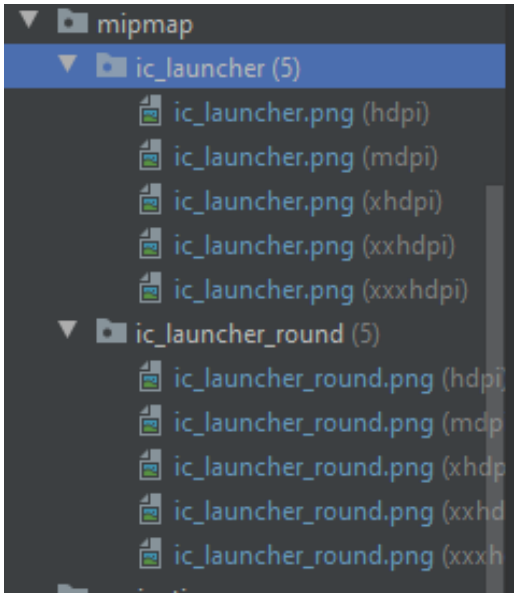
Figura 137 Menu_Main.Admin

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/iconLogout"
        android:icon="@drawable/ic_baseline_exit_to_app_48"
        android:title="Cerrar sesión"
        app:showAsAction="never|collapseActionView|withText"></item>
    <item
        android:id="@+id/iconAssistence"
        android:icon="@drawable/ic_baseline_exit_to_app_48"
        android:title="Tomar asistencia"></item>
    <item
        android:id="@+id/iconTensorflow"
        android:icon="@drawable/ic_baseline_exit_to_app_48"
        android:title="Obtener recomendación"></item>
</menu>
```

Fuente: Autores del proyecto.

Mipmap: aquí se encuentran todos los iconos del aplicativo.

Figura 138 MipMap



Fuente: Autores del proyecto.

Values: aquí se cuentran los archivos XML que a su vez contienen valores simples, como los valores enteros colores y strings.

Figura 139 Values

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#ff823e</color>
    <color name="colorPrimaryDark">#56524a</color>
    <color name="colorAccent">#000</color>
    <color name="colorAccent1">#ffff06</color>
    <color name="colorWhite">#ffffff</color>
    <color name="colorBorder">#777777</color>
    <color name="colorGrayDark">#484141</color>
    <color name="colorGrayLight">#56524a</color>
    <color name="colorHint">#afacac</color>
    <color name="colorSecondText">#e0e0e0</color>
</resources>
```

Fuente: Autores del proyecto.

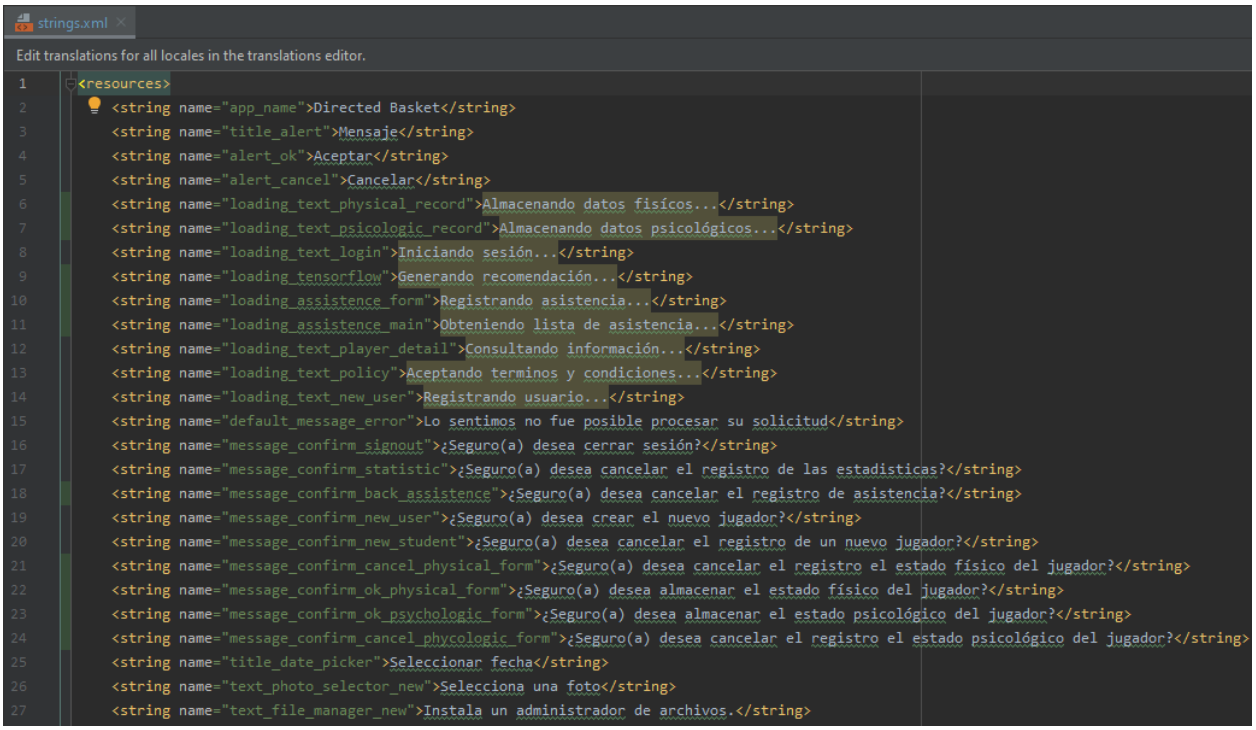
Figura 140 Dimens

dimens.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="margin_field_student">24dp</dimen>
    <dimen name="actionBarSize">60dp</dimen>

</resources>
```

Fuente: Autores del proyecto.

Figura 141 Strings



Fuente: Autores del proyecto.

Gradle Scripts: aquí se encuentra todo el sistema de compilación y el empaquetamiento de en APK o Android Bundles.

Figura 142 Build.gradle

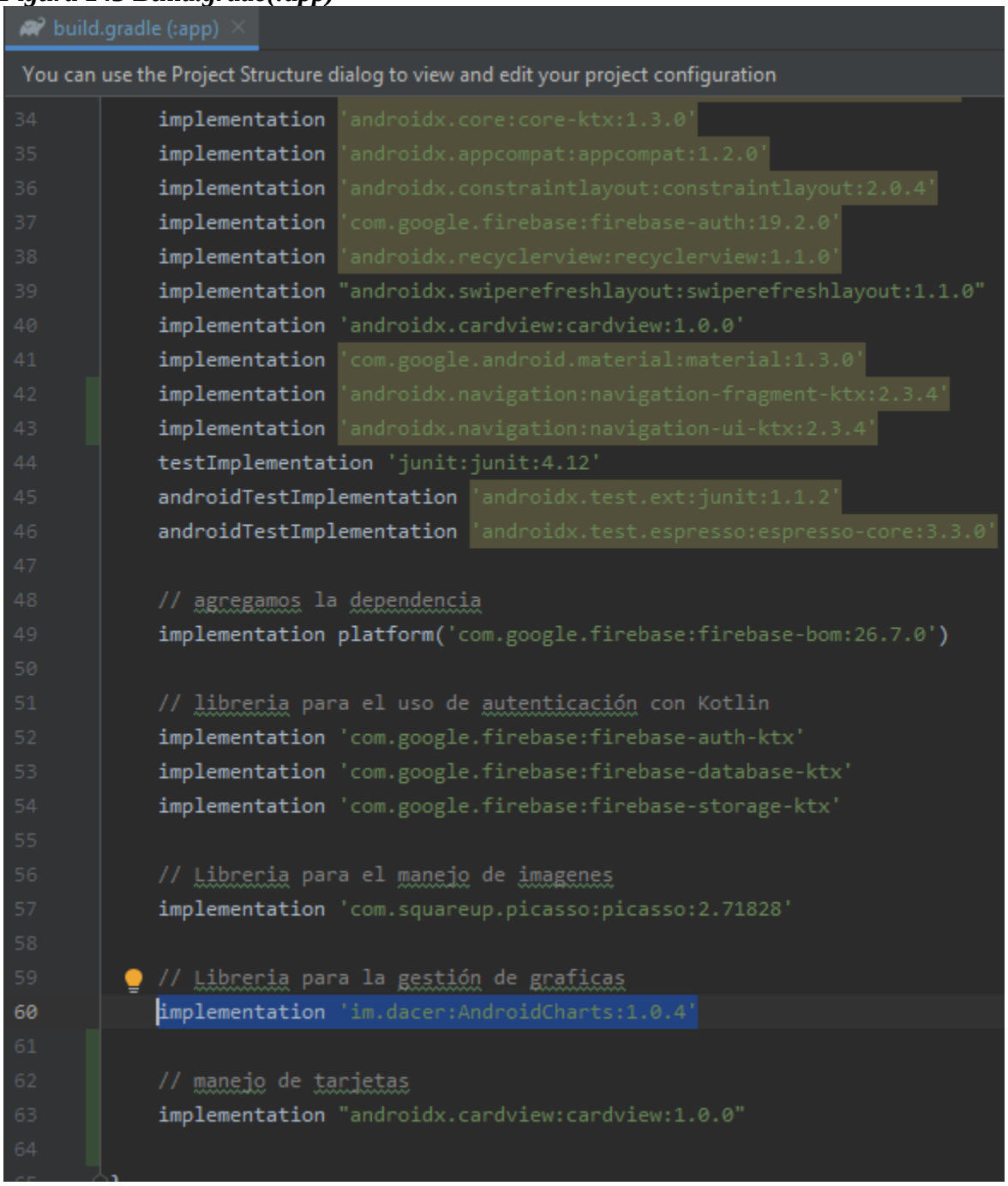
build.gradle (Directed Basket)

You can use the Project Structure dialog to view and edit your project configuration

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    ext.kotlin_version = "1.3.72"
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.0.0"
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"

        // Libreria requerida para el usuaro de Firebase
        classpath "com.google.gms:google-services:4.3.5"
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Fuente: Autores del proyecto.

Figura 143 Build.grade(:app)



Fuente: Autores del proyecto.

Figura 144 Gradle-wrapper.properties

gradle-wrapper.properties

```
#Thu Mar 18 13:04:41 COT 2021
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-6.1.1-all.zip
```

Fuente: Autores del proyecto.

Figura 145 Gradle.Properties(Project Properties

```
# Project-wide Gradle settings.
# IDE (e.g. Android Studio) users:
# Gradle settings configured through the IDE *will override*
# any settings specified in this file.
# For more details on how to configure your build environment visit
# http://www.gradle.org/docs/current/userguide/build_environment.html
# Specifies the JVM arguments used for the daemon process.
# The setting is particularly useful for tweaking memory settings.
org.gradle.jvmargs=-Xmx2048m
# When configured, Gradle will run in incubating parallel mode.
# This option should only be used with decoupled projects. More details, visit
# http://www.gradle.org/docs/current/userguide/multi_project_builds.html#sec:decoupled_projects
# org.gradle.parallel=true
# AndroidX package structure to make it clearer which packages are bundled with the
# Android operating system, and which are packaged with your app"s APK
# https://developer.android.com/topic/libraries/support-library/androidx-rn
android.useAndroidX=true
# Automatically convert third-party libraries to use AndroidX
android.enableJetifier=true
# Kotlin code style for this project: "official" or "obsolete":
kotlin.code.style=official
```

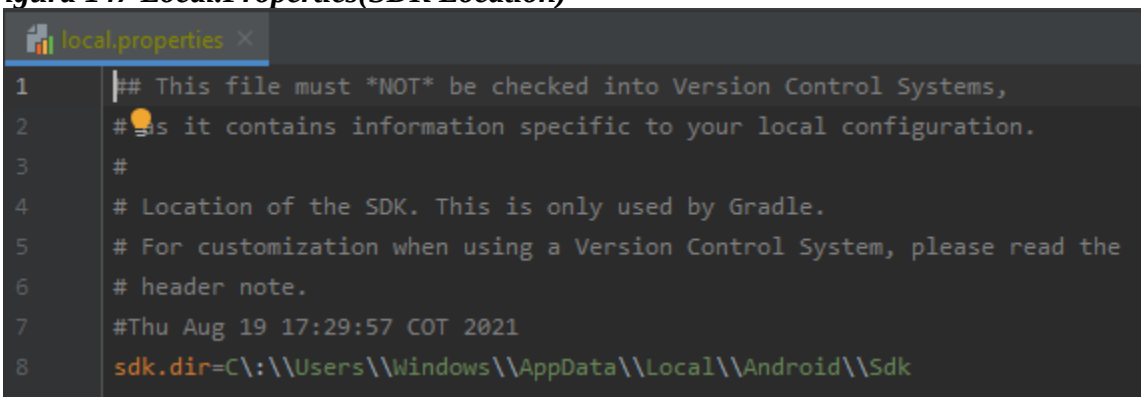
Fuente: Autores del proyecto.

Figura 146 Settings.Gradle(Projects Settings)

settings.gradle

```
include ':app'
rootProject.name = "Directed Basket"
```

Figura 147 Local.Properties(SDK Location)



Fuente: Autores del proyecto.

5. Dataset

Figura 148 Dataset

	playerId	force	resistance	speed	flexibility	balance	coordination	psychologicalAspects
2	3	5	6	10	10	10	4	250
5	6	10	10	10	5	6	5	250
4	5	10	5	6	10	3	4	250
0	1	7	8	5	6	5	7	55
1	2	6	5	4	10	10	10	250
3	4	10	5	5	4	5	10	250

Fuente: Autores del proyecto.

Funcionamiento del DataSet

Este data set está compuesto por 12 valores para cada deportista los cuales también están divididos en cuatro secciones.

Figura 149 Funcionamiento del dataset

Edad persona	Genero	Categoria	Estatura	Fuerza	Resistencia	Velocidad	Flexibilidad	Equilibrio	Coordinación	Aspectos Psicológicos	Posición
10 ->50	0-1	1->3	140 -> 190	1->10	1->10	1->10	1->10	1->10	1->10	55- 275	1-> 5

Fuente: Autores del proyecto.

La primera sección del dataset está compuesta por los valores de edad, género y categoría los cuales los cuales son usados para poder diferenciar los tipos de deportistas que hay.

la edad esta presentada entre un rango de 10 a 50 años ya que este es el rango de edad que se presenta en la escuela de formación de baloncesto, el género esta diferenciado por los valores 0 y 1 siendo 0 para femenino y 1 para masculino y la categoría esta diferenciada por los valores 1,2 y 3 siendo 1 la categoría de prejuvenil que esta es la categoría que se le da a los deportistas que están en el rango de edad de 10 a 13 años, el numero 2 es para la categoría juvenil que comprende el rango de edad de 14 a 17 y el numero 3 es para la categoría de adultos que es desde los 18 en adelante.

Figura 150 Campos del dataset

Edad persona	Genero	Categoria
10 ->50	0-1	1->3
11	0	1
11	0	1
11	0	1
11	0	1

Fuente: Autores del proyecto.

La segunda sección del dataset es la evaluación de aspectos físicos que está compuesta por estatura, fuerza, resistencia, velocidad, flexibilidad, equilibrio y coordinación.

El primer valor de esta sección es la estatura la cual está comprendida en un rango de 140 cm a 190 cm.

Los siguientes valores son fuerza, resistencia, velocidad flexibilidad equilibrio y coordinación estos valores son tomados por el entrenador según el rendimiento del deportista en la última sesión de entrenamiento y están comprendidos de 1 hasta 10 siendo 1 el valor más bajo en la calificación y 10 el más alto.

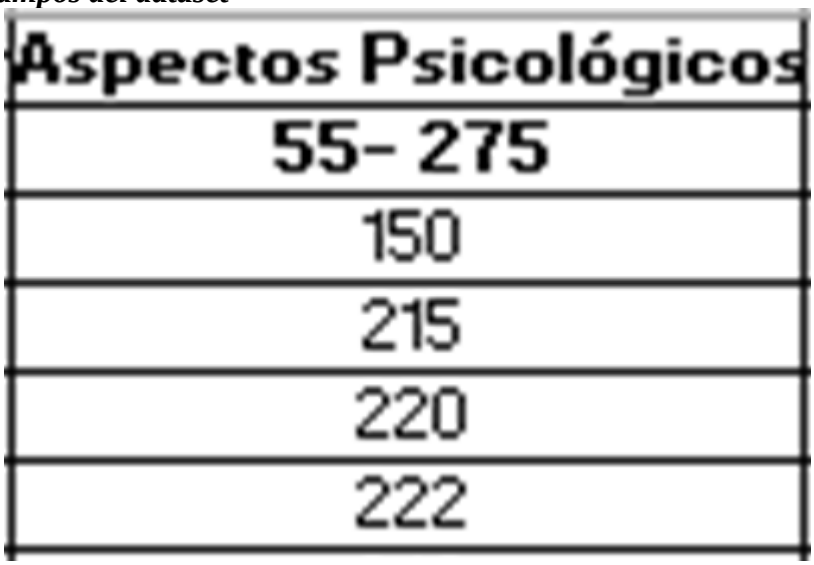
Figura 151 Campos del dataset

Estatura	Fuerza	Resistencia	Velocidad	Flexibilidad	Equilibrio	Coordinación
140 -> 190	1->10	1->10	1->10	1->10	1->10	1->10
144	7	8	5	6	5	7
143	6	5	4	7	8	5
145	5	6	7	8	8	4
149	8	5	5	4	5	9

Fuente: Autores del proyecto.

La siguiente sección del dataset corresponde a los aspectos psicológicos del deportista los cuales son evaluados por este mismo estos valores pueden variar desde 55 hasta 275 siendo 55 el resultado de la encuesta más bajo posible y 275 el resultado más alto.

Figura 152 Campos del dataset



Aspectos Psicológicos
55- 275
150
215
220
222

Fuente: Autores del proyecto.

La última sección del dataset es la posición en la que se recomienda que el deportista juegue, esta posición se calcula y recomienda gracias a los valores de las secciones de aspectos físicos y psicológicos, para cada una de las posiciones se toma en cuenta varios aspectos en específico.

Los aspectos valorados para la posición 1 o posición base son coordinación, equilibrio, flexibilidades superiores a 7 y estatura mayor del promedio según la categoría del deportista.

Para la posición 2 o posición alero es velocidad, equilibrio y flexibilidad superiores a 7 y estatura mayor del promedio según la categoría del deportista.

Para la posición 3 o posición ala pívot es fuerza y coordinación superiores a 7 y estatura mayor del promedio según la categoría del deportista.

Para la posición 4 o posición pívot es fuerza y flexibilidad superiores a 7 y estatura mayor del promedio según la categoría del deportista.

Para la posición 5 o posición escolta es fuerza, resistencia y velocidad superiores a 7 y en esta posición no se tiene en cuenta la estatura.

También existe otro valor en posiciones que es el 0 el cual se usa para mostrar que un deportista no es apto para el juego según sus valores psicológicos ya que estos no superan el valor de 200.