Diseño y Desarrollo de una Aplicación Android.

Utilizando API REST, Firebase, RetroFit, Picasso & Espresso.

Luis G. Reyes, Rubén Molina

1 RESUMEN

Este Trabajo de Fin de Grado es una aplicación móvil para dispositivos Android cuya función es la de servir como catálogo de juegos gratuitos para PC y móviles. En ella podemos encontrar todos los juegos gratuitos del momento, los cuales se actualizan de forma constante y cuyos cambios recibimos al momento. Toda la información pertinente a los videojuegos la recibimos desde la web **FreeToGame** y su correspondiente API REST.

Una vez dentro de la aplicación te ofrecemos las opciones de guardar los juegos que te interesen en favoritos, compartir el juego con quien quieras o acceder a su web oficial desde donde los puedes descargar.

2 ABSTRACT

This Final Degree Project is a mobile application for Android devices whose function is to serve as a catalog of free games for PC and mobile. In it we can find all the free games of the moment, which are constantly updated and what changes we receive at the moment. All the information pertaining to video games is received from the web **FreeToGame** and its corresponding REST API.

Once inside the application we offer you the options to save the games that interest you in favorites, share the game with whoever you want or access its official website where you can download them.

Contents

1 RESUMEN **3**

2 ABSTRACT **3**

3 INTRODUCCIÓN **5**
3.1 Contexto 5
3.2 Objetivos del Proyecto 5
3.3 Características del Proyecto 5

4 CONTEXTO FUNCIONAL & TECNOLÓGICO **6**
4.1 Contexto Funcional 6
4.2 Contexto Tecnológico 6

5 Planificación del Proyecto **7**

6 PRESUPUESTO **8**

7 ANÁLISIS & DISEÑO **9**
7.1 Especificaciones de Requisitos 9
7.2 Selección de la Plataforma Tecnológica 11
7.3 Descripción del Diseño de la Solución 12

8 CONSTRUCCIÓN **13**
8.1 Documentación Descriptiva 13
8.1.1 Fichero JSON 13
8.1.2 Modelos JSON / Base de Datos 14
8.1.3 Registro de Usuario en Firebase 18
8.1.4 Login de Usuario en Firebase 19
8.1.5 Dar de alta juegos dentro de un Usuario 20
8.1.6 Realizar petición HTTP a la API con Retrofit 21
8.1.7 Rellenar nuestro RecyclerView con la respuesta de la API 21
8.2 Problemas Encontrados & Soluciones 22
8.2.1 Cambios en la Base de Datos. 22
8.2.2 Cambiar de una base de datos SQL a una NoSQL. 23

9 VALIDACIÓN DE LA SOLUCIÓN **26**
9.1 Documentación Descriptiva 26
9.1.1 Configuración del entorno de pruebas 27
9.1.2 Login con campos vacíos 28
9.1.3 Registro con campos vacíos 28
9.1.4 Registro de una cuenta. 29

10 CONCLUSIONES **30**

11 FUTURAS MEJORAS **31**

12 BIBLIOGRAFÍA & CITACIONES **32**

3 INTRODUCCIÓN

3.1 Contexto

Nuestra idea nace a raíz de nuestra afición por los videojuegos. Nos dimos cuenta de que no existía ninguna aplicación para móvil en la que poder ver un catálogo con todos los videojuegos gratuitos que aunara no solo móvil y PC sino todas las grandes tiendas de videojuegos online como pueden ser *Epic Games* o *Steam*

3.2 Objetivos del Proyecto

El objetivo del proyecto es poder acercar el mayor número de videojuegos gratuitos al usuario y que este pueda conocer todos los detalles de los mismos desde su dispositivo móvil.

Hemos querido que la comodidad y el diseño primen en el trabajo, haciendo una aplicación limpia donde el usuario entienda en todo momento que está pasando.

3.3 Características del Proyecto

Nuestro proyecto cuenta con las siguientes características:

- **Retrofit**: cliente HTTP que nos permite realizar peticiones a la API[1] de FreeToGame, devolviendo un fichero JSON con la información solicitada.
- **GSON**: librería que utilizamos para convertir los elementos del fichero JSON a objetos Java (POJO).
- **Picasso**: librería que utilizamos para cargar imágenes, estas imágenes las obtenemos del fichero JSON que nos devuelve Retrofit.
- **Firebase**: base de datos no relacional que usaremos para almacenar los usuarios y la información.
- **Espresso**: librería que utilizaremos para hacer los tests sobre la aplicación.

La aplicación requiere que tengas una cuenta antes de poder utilizarla, una vez la tengas, podrá realizar las siguientes acciones:

- Ver una ficha detallada sobre el videojuego.
- Guardar el videojuego en favoritos.
- Compartir un enlace con la web oficial donde poder descargar el videojuego.
- Eliminar dicho videojuego de favoritos.

[1]Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones.Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).

4 CONTEXTO FUNCIONAL & TECNOLÓGICO

4.1 Contexto Funcional

Nuestra principal inspiración es *IMDb* donde puedes encontrar toda la información que necesitas sobre películas y series. Más adelante descubrimos que también existía una web llamada *IGDb* la cual ofrecía información detallada sobre videojuegos, así como opiniones.

La principal diferencia con nuestra aplicación es que estas son aplicaciones webs, no son aplicaciones nativas para Android; esto puede hacer que su uso sea más tedioso y al mismo tiempo menos óptimo.

En nuestra aplicación hemos decidido ofrecer una alternativa y ofrecer las principales características de las webs anteriormente mencionadas en una aplicación nativa para Android.

4.2 Contexto Tecnológico

A la hora de realizar una aplicación para móvil nos encontramos ante un gran abanico de opciones, a continuación procederemos a destacar algunos de los puntos de mayor importancia del proyecto y por cual nos hemos decantado.

- **Lenguaje de Programación**: Cuando se trata de hacer una **aplicación móvil Android** de forma **nativa** nos encontramos ante dos opciones, *Java* o *Kotlin*.
- **Base de Datos**: A la hora de elegir una base de datos nos encontramos con dos paradigmas, o bien una **base de datos relacional**(*MariaDB, MySQL, SQL Server, SQLite*) o una **base de datos no relacional** (*MongoDB*); en nuestro caso hemos querido utilizar una base de datos que sea independiente del dispositivo móvil en el que se use, es decir, que desde cualquier dispositivo móvil puedas introducir tu correo electrónico y contraseña y poder acceder a tu cuenta.

  Para ello hemos utilizado **Firebase**[2], producto de Google que nos permite tener una **base de datos no relacional en la nube**, haciendo que pueda ser accedida desde cualquier dispositivo.
- **Cliente HTTP**: Los dos clientes HTTP más conocidos para Java, al menos dentro del desarrollo Android, son *Retrofit* y *Volley*; ambos de características similares, la elección se realizó en base a la documentación de una frente a la otra.
- **Testing**: La opción más documentada y utilizada es **Espresso**, tiene un uso muy similar al de *Selenium* y al tener experiencia con este último, nos decantamos por el primero.
- **Librerías**: Para realizar el proyecto necesitábamos una librería para la conversión de JSON a POJO y para la carga de imágenes, en ambos casos hay dos librerías que destacan frente a las demás y por ello las hemos utilizado.

[2]Firebase de Google es una plataforma en la nube para el desarrollo de aplicaciones web y móvil.

5 Planificación del Proyecto

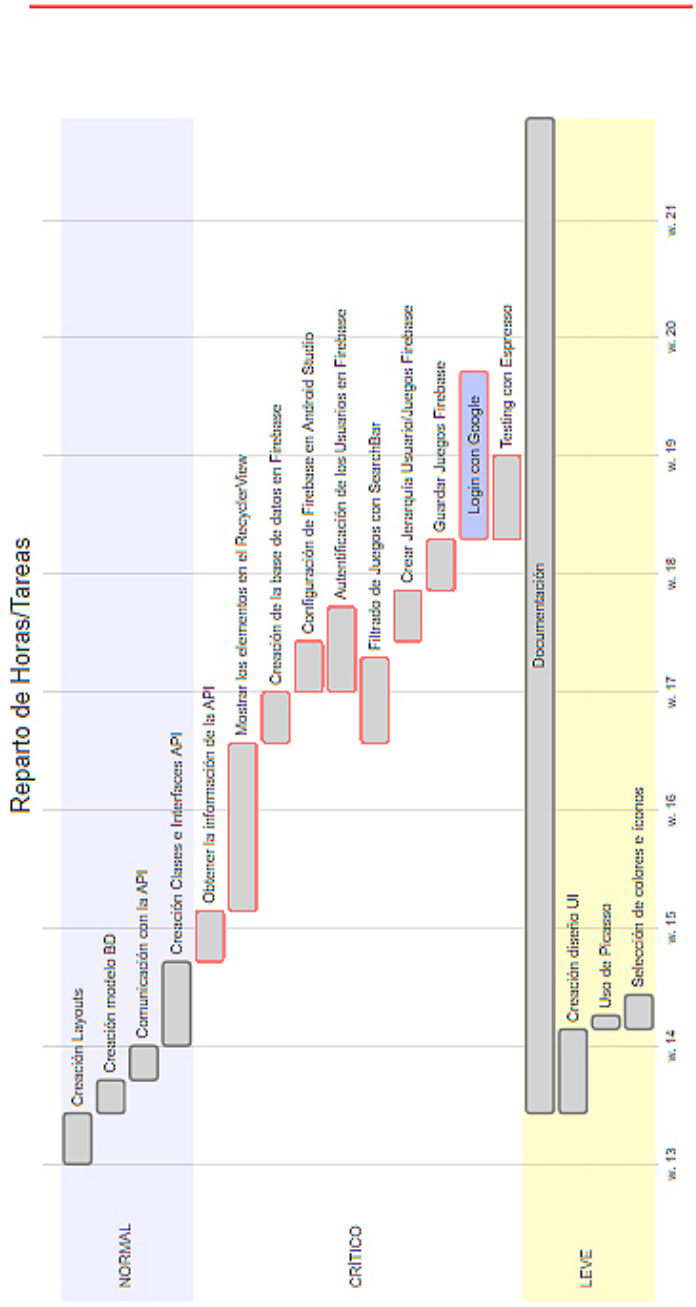

Figure 1: Diagrama de Gannt con la Planificación del Proyecto.

6 PRESUPUESTO

CONCEPTO	CANTIDAD	PRECIO UNITARIO	TOTAL
MANO DE OBRA			
Horas	180	70,00€	12.600,00€
		TOTAL MANO DE OBRA:	*12.600,00€*
HARDWARE			
Ordenador	2 Unidades	950,00€	1.900€
Licencia Microsoft	2 Unidades	130,00€	260€
Auriculares	2 Unidades	20,00€	40,00€
Móviles	2 Unidades	300,00€	600,00€
		TOTAL HARDWARE:	*2.800,00€*
SOFTWARE			
Licencia Android Studio	0 Unidades	0,00€	0,00€
Licencia Discord	0 Unidades	0,00€	0,00€
Licencia JDK	0 Unidades	0,00€	0,00€
Licencia FireBase	0 Unidades	0,00€	0,00€
Licencia GitHub	0 Unidades	0,00€	0,00€
		TOTAL SOFTWARE:	0,00€
		PRESUPUESTO TOTAL:	**15.400€**

7 ANÁLISIS & DISEÑO

7.1 Especificaciones de Requisitos

Los equipos utilizados son:

- Asus X541U, IntelCore i7-7500U, 8GB RAM, 1TB HDD, Windows 10 Home 64 bits Version 20H2.
- Samsung A42 5G, Snapdragon 750G, 4GB RAM, 128GB, Android 10 y One UI 2.5.
- Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90 GHz, 32GB RAM, 1TB SDD, Windows 10 Pro 64 bits, Version 21H2.
- Samsung S20+5G, Qualcomm Snapdragon 865, 12GB RAM, 128GB, Android 11, One UI 3.1.
- **Dispositivo Virtual**: Nexus, 4.95 pulgadas, 560dpi, Android 7 (Google API's), CPU/ABI x86_64.

Versiones de Software utilizadas:

- Android Studio 4.1.3
- JRE 1.8.0_281
- SDK 24

Dependencias del Proyecto:

```
dependencies {

    implementation platform('com.google.firebase:firebase-bom:29.3.1')
    implementation 'com.google.firebase:firebase-analytics'

    implementation 'com.squareup.retrofit2:retrofit:2.1.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.1.0'
    implementation 'com.squareup.retrofit2:converter-scalars:2.3.0'
    implementation 'com.squareup.picasso:picasso:2.5.2'

    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.cardview:cardview:1.0.0'
    implementation 'androidx.recyclerview:recyclerview:1.1.0'
    implementation 'com.github.bumptech.glide:glide:3.7.0'

    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'androidx.annotation:annotation:1.2.0'
    implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.3.1'
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1'
    implementation 'com.google.firebase:firebase-auth:21.0.3'
    implementation 'androidx.navigation:navigation-fragment:2.3.5'
    implementation 'androidx.navigation:navigation-ui:2.3.5'
    implementation 'com.google.firebase:firebase-firestore:24.1.1'
    implementation 'androidx.test.espresso:espresso-intents:3.4.0'
    implementation 'com.google.android.gms:play-services-auth:19.0.0'

    debugImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    debugImplementation 'androidx.test.ext:junit:1.1.3'
    debugImplementation 'junit:junit:4.13.2'
}
```

7.2 Selección de la Plataforma Tecnológica

- **Plataforma de Desarrollo**: Android Studio, actualmente creeemos que es la mejor platanforma para el desarrollo de aplicaciones nativas en Android, además de la familiaridad que tenemos con ella después de utilizarla durante el curso.
- **Lenguaje de Programación**: Java, actualmente el principal lenguaje de programación para aplicaciones Android nativas es Kotlin; pero decidimos hacer la aplicación en Java, ya que es un lenguaje de programación en el que tenemos una mayor experiencia.
- **Base de Datos**: FireBase, es una base de datos cloud que nos permite tener un control de los usuarios y que la información sea independiente del dispositivo, lo cual permite que la información pueda ser accedida desde cualqueir lugar. Además cuenta con un servicio experimentado y robusto.
- **Control de Versiones**: Git, actualmente es el controlador de versiones más utilizado y lo utilizamos en conjunto con GitHub.

7.3 Descripción del Diseño de la Solución

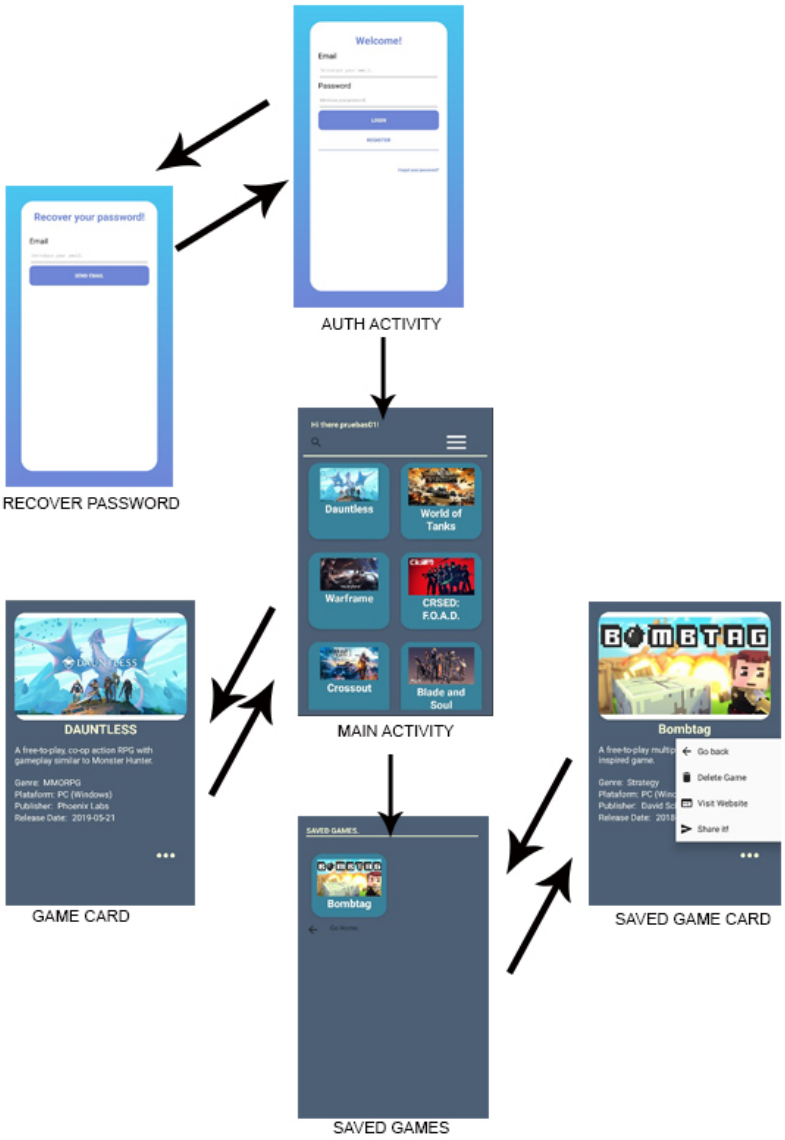

Figure 2: Diseño Activities

8 CONSTRUCCIÓN

8.1 Documentación Descriptiva

8.1.1 Fichero JSON

El objetivo es transformar los elementos de este fichero JSON en representaciones de juegos, con los que poder realizar acciones.

Breve ejemplo del fichero JSON que nos devuelve la API de FreeToGame:


```
[
  {
    "id": 1,
    "title": "Dauntless",
    "thumbnail": "https://www.freetogame.com/g/1/thumbnail.jpg",
    "short_description": "A free-to-play, co-op action RPG with gameplay ...",
    "game_url": "https://www.freetogame.com/open/dauntless",
    "genre": "MMORPG",
    "platform": "PC (Windows)",
    "publisher": "Phoenix Labs",
    "developer": "Phoenix Labs, Iron Galaxy",
    "release_date": "2019-05-21",
    "freetogame_profile_url": "https://www.freetogame.com/dauntless"
  },
  {
    "id": 2,
    "title": "World of Tanks",
    "thumbnail": "https://www.freetogame.com/g/2/thumbnail.jpg",
    "short_description": "If you like blowing up tanks, with a quick and ...",
    "game_url": "https://www.freetogame.com/open/world-of-tanks",
    "genre": "Shooter",
    "platform": "PC (Windows)",
    "publisher": "Wargaming",
    "developer": "Wargaming",
    "release_date": "2011-04-12",
    "freetogame_profile_url": "https://www.freetogame.com/world-of-tanks"
  },
]
```

8.1.2 Modelos JSON / Base de Datos

```
public class Game {

    @SerializedName("id")
    @Expose
    private Integer id;

    @SerializedName("title")
    @Expose
    private String title;

    @SerializedName("thumbnail")
    @Expose
    private String thumbnail;

    @SerializedName("short_description")
    @Expose
    private String shortDescription;

    @SerializedName("game_url")
    @Expose
    private String gameUrl;

    @SerializedName("genre")
    @Expose
    private String genre;

    @SerializedName("platform")
    @Expose
    private String platform;

    @SerializedName("publisher")
    @Expose
    private String publisher;

    @SerializedName("developer")
    @Expose
    private String developer;

    @SerializedName("release_date")
    @Expose
    private String releaseDate;

    @SerializedName("freetogame_profile_url")
    @Expose
    private String freetogameProfileUrl;
}
```

Modelo Java de un Juego

Representación de un juego dentro nuestra aplicación, gracias a este modelo podremos crear juegos, manipularlos y finalmente mostrarlos dentro del **RecyclerView**.

```
public class GameOO {

    private final String title;
    private final String thumbnail;
    private final String shortDescription;
    private final String gameUrl;
    private final String genre;
    private final String platform;
    private final String publisher;
    private final String releaseDate;

    private GameOO(final GameOOBuilder builder) {
        title = builder.title;
        thumbnail = builder.thumbnail;
        shortDescription = builder.shortDescription;
        gameUrl = builder.gameUrl;
        genre = builder.genre;
        platform = builder.platform;
        publisher = builder.publisher;
        releaseDate = builder.releaseDate;
    }

    public String getTitle() {
        return title;
    }

    public String getThumbnail() {
        return thumbnail;
    }

    public String getShortDescription() {
        return shortDescription;
    }

    public String getGameUrl() {
        return gameUrl;
    }

    public String getGenre() {
        return genre;
    }

    public String getPlatform() {
        return platform;
```

```
}

public String getPublisher() {
    return publisher;
}

public String getReleaseDate() {
    return releaseDate;
}

@Override
public String toString() {
    return "GameOO{" +
            ", title='" + title + '\'' +
            ", thumbnail='" + thumbnail + '\'' +
            ", shortDescription='" + shortDescription + '\'' +
            ", gameUrl='" + gameUrl + '\'' +
            ", genre='" + genre + '\'' +
            ", platform='" + platform + '\'' +
            ", publisher='" + publisher + '\'' +
            ", releaseDate='" + releaseDate + '\'' +
            '}';
}

/**
 * Builder Class:
 * The one used to create the GameOO instances.
 */

public static class GameOOBuilder {

    private String title;
    private String thumbnail;
    private String shortDescription;
    private String gameUrl;
    private String genre;
    private String platform;
    private String publisher;
    private String releaseDate;

    public GameOOBuilder setTitle(String title){
        this.title = title;
        return this;
    }

    public GameOOBuilder setThumbnail(String thumbnail){
        this.thumbnail = thumbnail;
        return this;
```

```
        }

        public GameOOBuilder setShortDescription(String shortDescription){
            this.shortDescription = shortDescription;
            return this;
        }

        public GameOOBuilder setGameURL(String gameURL){
            this.gameUrl = gameURL;
            return this;
        }

        public GameOOBuilder setGenre(String genre){
            this.genre = genre;
            return this;
        }

        public GameOOBuilder setPlatform(String platform){
            this.platform = platform;
            return this;
        }

        public GameOOBuilder setPublisher(String publisher){
            this.publisher = publisher;
            return this;
        }

        public GameOOBuilder setReleaseDate(String releaseDate){
            this.releaseDate = releaseDate;
            return this;
        }

        public GameOO build(){
            return new GameOO(this);
        }
    }
}
```

8.1.3 Registro de Usuario en Firebase

```
// Definimos los elementos UI
private EditText etEmail, etPassword;

// Creamos los objetos necesarios para poder trabajar con Firebase
// En este caso un firebaseAuth para el control de la sesión
private FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();

// Método en el que realizamos el registro del Usuario
public void register(View view){
  String strEmail = etEmail.getText().toString();
  String strPassword = etPassword.getText().toString();

  // Comprobamos que los campos están rellenos
  if(!strEmail.isEmpty() && !strPassword.isEmpty()){
      // Llamamos al método createUserWithEmailAndPassword
      fireBaseAuth.createUserWithEmailAndPassword(strEmail, strPassword)
      .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
          @Override
          // En caso de que se realice el método, lanzamos un mensaje.
          public void onComplete(@NonNull Task<AuthResult> task) {
              if(task.isSuccessful()){
                  startActivity(new Intent(AuthActivity.this, MainActivity.class));
              } else {
                  Toast.makeText(AuthActivity.this, task.getException().getMessage(),
                  Toast.LENGTH_SHORT).show();
              }
          }
      });
  } else {
      Toast.makeText(this, "Please, fill the data.", Toast.LENGTH_SHORT).show();
  }
}
```

8.1.4 Login de Usuario en Firebase

Exactamente igual que el método anterior a excepción de que en esta ocasión llamamos al método **signWithEmailAndPassword()**

```
public void login(View view){
  String strEmail = etEmail.getText().toString();
  String strPassword = etPassword.getText().toString();

  if(!strEmail.isEmpty() && !strPassword.isEmpty()){
      firebaseAuth.signInWithEmailAndPassword(strEmail, strPassword)
      .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
          @Override
          public void onComplete(@NonNull Task<AuthResult> task) {
              if(task.isSuccessful()){
                  startActivity(new Intent(AuthActivity.this, MainActivity.class));
              } else {
                  Toast.makeText(AuthActivity.this, task.getException().getMessage(),
                  Toast.LENGTH_SHORT).show();
              }
          }
      });
  } else {
      Toast.makeText(this, "Please, fill the data.", Toast.LENGTH_SHORT).show();
  }
}
```

8.1.5 Dar de alta juegos dentro de un Usuario

```
public void insertUserGamesRelation(GameOO gameOO, Context context) {

  // Crear un mapa donde almacenemos los valores
  // que queremos guardar en Firebase
  Map<String, Object> game = new HashMap<>();

  game.put(TITLE, gameOO.getTitle());
  game.put(THUMBNAIL, gameOO.getThumbnail());
  game.put(SHORT_DESC, gameOO.getShortDescription());
  game.put(RELEASE_DATE, gameOO.getReleaseDate());
  game.put(PUBLISHER, gameOO.getPublisher());
  game.put(PLATFORM, gameOO.getPlatform());
  game.put(GENRE, gameOO.getGenre());
  game.put(GAME_URL, gameOO.getGameUrl());

  // Añadimos la información a la colección
  // en el caso de esta aplicación la jerarquía
  // sería la siguiente:
  // user_games -> UID -> games_list -> games
  // Cada usuario tiene una lista de juegos.
  db.collection("user_games")
    .document(firebaseUser.getUid())
    .collection("games_list")
    .add(game)
    .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
      @Override
      public void onSuccess(DocumentReference documentReference) {
        // Mensaje diciendo que el juego ha sido guardado.
        Toast.makeText(context, "The game was saved.",
        Toast.LENGTH_SHORT).show();
        }
      })
      .addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
          // Mensaje de error al guardar el juego.
          Toast.makeText(context, "An error occurred while saving the game.",
          Toast.LENGTH_SHORT).show();
        }
  });
```

8.1.6 Realizar petición HTTP a la API con Retrofit

```
public class APIClient {
  private static final String BASE_URL = "https://www.freetogame.com/api/";
  private static Retrofit retrofit;

  public static Retrofit getClient(){
      if(retrofit == null){
          retrofit = new Retrofit.Builder()
                      .baseUrl(BASE_URL)
                      .addConverterFactory(GsonConverterFactory.create())
                      .build();
      }
      return retrofit;
  }
}
```

8.1.7 Rellenar nuestro RecyclerView con la respuesta de la API

```
 public void feedRecyclerView() {
  // Muestra los juegos en columnas de dos.
  RecyclerView.LayoutManager layoutManager = new GridLayoutManager(this, 2);
  recyclerView.setLayoutManager(layoutManager);

  GameApiInterface apiService = retrofit.create(GameApiInterface.class);
  Call<List<Game>> gamesList = apiService.loadGames();

  gamesList.enqueue(new Callback<List<Game>>() {
      @Override
      public void onResponse(Call<List<Game>> call, Response<List<Game>> response) {
          recyclerAdapter = new RecyclerAdapter(response.body(), MainActivity.this);
          recyclerView.setHasFixedSize(true);

          recyclerView.setAdapter(recyclerAdapter);
      }

      @Override
      public void onFailure(Call<List<Game>> call, Throwable t) {
          Log.e(TAG, "onFailure: " + t.getMessage());
      }
  });
}
```

8.2 Problemas Encontrados & Soluciones

8.2.1 Cambios en la Base de Datos.

El primer problema que nos encontramos fue con la base de datos. Al principio del proyecto pensamos en hacer una base de datos SQLite local que nos permitiera guardar los juegos dentro del dispositivo, pero vimos que era un parche más que una solución, ya que si el usuario cambiaba de dispositivo móvil perdería todos sus datos.

La única solución que vimos fue usar una base de datos Cloud que nos permitiera tener una base de datos independiente del dispositivo, después de ver cuales eran las mejores opciones, tomamos la decisión de utilizar Firebase.

Firebase nos permite almacenar nuestra información en una base de datos en la nube, esto junto con su servicio de autentificación nos hizo cambiar nuestra base de datos, actualmente permite registros y cada uno de los usuarios cuenta con su propia lista de juegos favoritos.

8.2.2 Cambiar de una base de datos SQL a una NoSQL.

Nuestra primera idea fue crear 3 tablas (users, games, users_games) como hubieramos hecho en una base de datos SQL, a continuación mostramos los componenetes de dichas tablas.

USERS		
ID	INTEGER	AUTOINCREMENT
EMAIL	STRING	
PASSWORD	STRING	

GAMES		
ID	INTEGER	AUTOINCREMENT
TITLE	STRING	
SHORT_DESC	STRING	
THUMBNAIL	STRING	
GENRE	STRING	
PLATFORM	STRING	
PUBLISHER	STRING	
GAME_URL	STRING	

USERS_GAMES		
ID	INTEGER	AUTOINCREMENT
ID_USER	INTEGER	
ID_GAME	INTEGER	

Jerarquía Base de Datos Firebase

Pronto nos dimos cuenta de que con este sistema, habría duplicidad de datos, ya que Firebase, al ser un sistema de gestión de bases de datos no relacional, gestiona las relaciones de una forma diferente a como lo planteamos con SQLite.

Firebase trabaja de una forma jerárquica, y eso fue lo que hicimos, creando una sola colección con la siguiente estructura:

```
|-- user_games
|   |-- USER_ID
|      |-- games_list
|          |-- GAME_ID
|              |-- title
|              |-- thumbnail
|              |-- short_desc
|              |-- release_date
|              |-- publisher
|              |-- platform
|              |-- genre
|              |-- game_url
```

De esta forma lo que conseguimos es que todos los juegos de un usuario estén dentro del ID del mismo. Siguiendo una estructura de árbol.

A continuación mostramos unas imágenes reales de la base de datos.

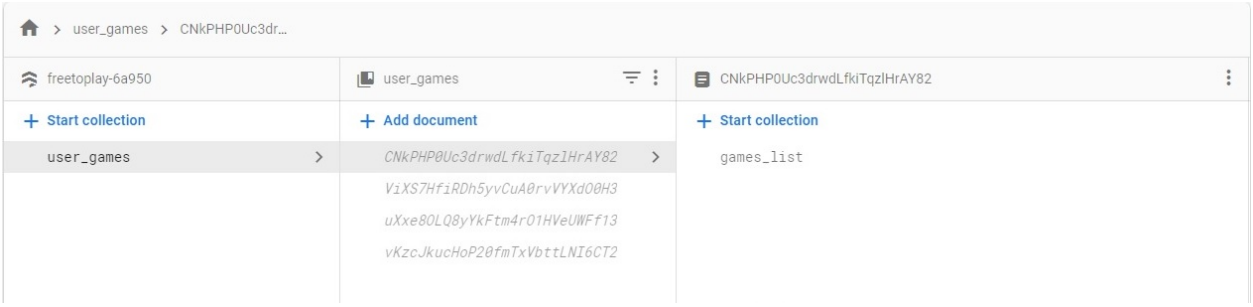

Figure 3: Firebase Jerarquía 01

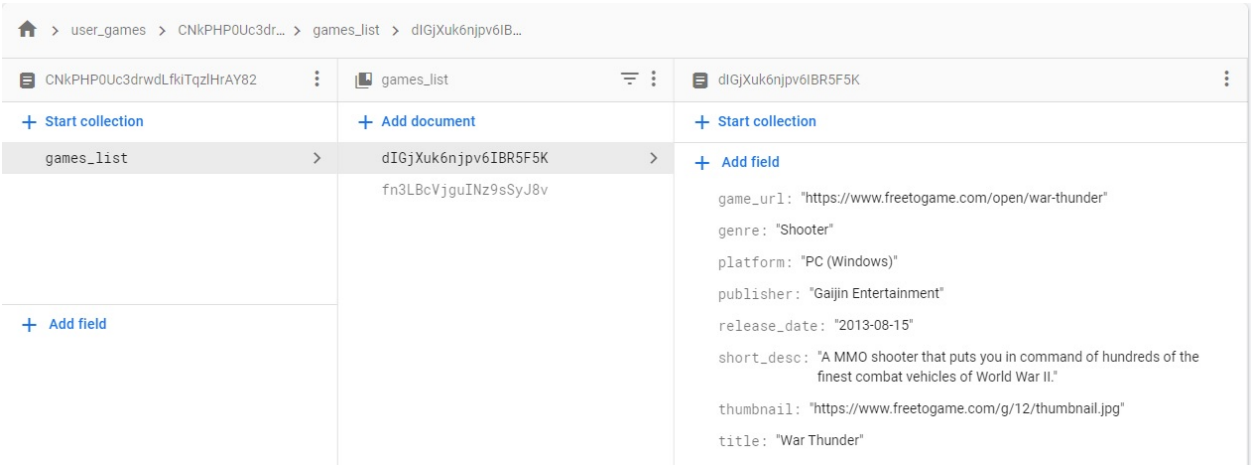

Figure 4: Firebase Jerarquía 02

9 VALIDACIÓN DE LA SOLUCIÓN

9.1 Documentación Descriptiva

Es importante hacer una mención a las dependencias que hemos utilizado para la realización de los tests, y la forma en la que estos se ejecutan.

```
debugImplementation 'androidx.test.espresso:espresso-core:3.4.0'
debugImplementation 'androidx.test.ext:junit:1.1.3'
debugImplementation 'junit:junit:4.13.2'
```

Como podemos ver, están en modo **debugImplementation** lo que quiere decir que a la hora de ejecutar nuestra Suite de Tests, tendremos que utilizar el modo *debug*.

Esta información fue de vital importancia, ya que por conflictos de dependencias / versiones, ejecutar los tests de cualquier otro modo, generaba errores; nos llevo bastantes días comprender el error hasta que al final dimos con la solución.

9.1.1 Configuración del entorno de pruebas

Lo primero que tenemos que definir es que versión de JUnit queremos que utilicen nuestros tests, en nuestro caso será la versión 4.

A continuación definimos nuestra **Rule** que determinará el Activity en el cual vamos a realizar los tests.

Por último con el decorador **Before** definimos los parámetros que queremos que se ejecuten antes de realizar los tests, en este caso definimos el objeto decorView como parte del Activity para poder testear los mensajes Toast[3].

```
@RunWith(AndroidJUnit4.class)
@LargeTest
public class EspressoTest {

  @Rule
  public ActivityScenarioRule<AuthActivity>
  authActivityActivityScenarioRule
          = new ActivityScenarioRule<>(AuthActivity.class);

  private View decorView;

  @Before
  public void setUp() {
      authActivityActivityScenarioRule.getScenario()
      .onActivity(new ActivityScenario.ActivityAction<AuthActivity>() {
          @Override
          public void perform(AuthActivity activity) {
              decorView = activity.getWindow().getDecorView();
          }
      });
  }
```

[3]El Toast En Android es un objeto de vista que se despliega como un elemento emergente en la interfaz del usuario, con el fin de mostrar un mensaje relacionado hacia alguna interacción realizada por el usuario.

9.1.2 Login con campos vacíos

- Definimos el mensaje que vamos a obtener en el Toast.
- Realizamos click en el *botón de login*.
- Comparamos el texto que nos devuelve el Toast con el que hemos definido.
- Comprobamos que en caso de querer hacer Login con los campos vacíos nos da el error esperado.

```
@Test
public void tryToLoginWithEmptyEditTextTest(){
    final String expectedWarning = "Please, fill the data.";
    onView(withId(R.id.btn_login))
            .perform(click());

    onView(withText(expectedWarning))
            .inRoot(withDecorView(not(decorView)))
            .check(matches(isDisplayed()));
}
```

9.1.3 Registro con campos vacíos

- Definimos el mensaje que vamos a obtener en el Toast.
- Realizamos click en el *botón de registro*.
- Comparamos el texto que nos devuelve el Toast con el que hemos definido.
- Comprobamos que en caso de querer hacer Login con los campos vacíos nos da el error esperado.

```
@Test
public void tryToRegisterWithEmptyEditTextTest(){
    final String expectedWarning = "Please, fill the data.";
    onView(withId(R.id.btn_register))
            .perform(click());

    onView(withText(expectedWarning))
            .inRoot(withDecorView(not(decorView)))
            .check(matches(isDisplayed()));
}
```

9.1.4 Registro de una cuenta.

- Definimos el email y contraseña que tendrá nuestro usuario.
- Escribimos en ambos EditText el contenido de nuestras constantes.
- Cerramos el KeyBoard para que pueda ver las coordenadas del botón
- Avisamos a la aplicación de que después se iniciará un nuevo Activity
- Realizamos el click
- Salimos de la Activity

```
@Test
public void tryRegisterAccountTest(){

    final String email = "usuarioTestPruebas@pruebas.com";
    final String password = "123456";

    onView(withId(R.id.et_email))
            .perform(typeText(email), closeSoftKeyboard());

    onView(withId(R.id.et_password))
            .perform(typeText(password), closeSoftKeyboard());

    Intents.init();
    onView(withId(R.id.btn_register))
            .perform(click());
    Intents.release();
}
```

10 CONCLUSIONES

Se han cumplido los objetivos de crear una aplicación 100% funcional que el usuario final pueda utilizar. Así mismo, hemos aprendido como funcionan nuevos protocolos (*API REST*), tecnologías (*Firebase*) y librerías (*Retrofit, Picasso, Gson o Espresso*).

Ha sido un proyecto lleno de retos, ya que desde el principio, propusimos hacer algo que fuese útil y realizarlo con tecnologías de vanguardia, hemos aprendido a ser más proactivos, a enfrentarnos a los errores desconocidos y a buscar la mejor solución para los diferentes inconvenientes que han sido apareciendo.

En definitiva, creo que hemos aprendido mucho y que somo un poco mejores programadores que antes de afrontar este proyecto.

11 FUTURAS MEJORAS

A nivel de mejoras hay dos campos principales en los que nos gustaría centrarnos:

1. **Mejorar el menú**: actualmente usamos un menú pop up, pero nos gustaría en un futuro poder hacer un menú estilo drawer[4] que hiciera que la UI se viera mucho más moderna y estética. La razón por la cual no lo hemos llevado a cabo en la actual versión de la aplicación reside en el uso casi obligatorio de fragments para ello y en nuestro caso supondría rehacer el 100% de la apliación, por eso queremos ver y estudiar la mejor forma de migrar nuestros activities a fragments y posteriormente realizar la mejora del menú.
2. **Registros con cuenta Google**: uno de los problemas que nos hemos encontrado es que el registro con cuenta Google funciona en el emulador pero no en el dispositivo físico, lo cual tenemos que subsanar y conseguir en futuras versiones.

[4]Tipo de menú especial el cual funciona deslizando el dedo, algunos ejemplos donde podemos encontrar este tipo de menú son Twitter o JustEat.

12 BIBLIOGRAFÍA & CITACIONES

A continuación mencionamos todas las fuentes utilizadas a lo largo de este trabajo.

albodelu. 2015. *Espresso Click Menu Item*. https://stackoverflow.com/questions/33965723/espresso-click-menu-item.

Alvarez Tech. 2016. *Consumir Una API, Web Service En Android, Creando Una Pokedex*. https://www.youtube.com/watch?v=xQn8u4Htib4.

Code Palace. 2020. *Navigation Search Bar Tutorial*. https://www.youtube.com/watch?v=8q-4AJFlraI.

Codelia. 2020. *Android Studio 4.0 - RecyclerView Con CardView Tutorial Completo*. https://www.youtube.com/watch?v=HrZgfoBeams.

Firebase. 2022. *Lee y Escribe Datos En Android*. https://firebase.google.com/docs/database/android/read-and-write?hl=es-419.

FoxAndroid. 2021. *How to Add Search View in Toolbar in Android Studio*. https://www.youtube.com/watch?v=M3UDh9mwBd8.

Frank van Puffelen. 2017. *Forgot Password in Firebase for Android*. https://stackoverflow.com/questions/42800349/forgot-password-in-firebase-for-android.

FreeToGame. 2019. https://www.freetogame.com/.

longkai. 2015a. *Android Espresso Intents Test Randomly Fail with "Init() Must Be Called Prior to Using This Method"*. https://stackoverflow.com/questions/33624802/android-espresso-intents-test-randomly-fail-with-init-must-be-called-prior-t.

———. 2015b. *Check If a New Activity Is Started with Espresso*. https://stackoverflow.com/questions/41890943/check-if-a-new-activity-is-started-with-espresso.

R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org.