

# Creación de una API

## 1. Levantar o configurar un proyecto en [start.spring.io](https://start.spring.io).

The screenshot shows the Spring Initializr web application. It has a sidebar on the left with a menu icon and a search bar. The main area is titled "spring initializr". It contains several configuration sections:

- Project**: Options for Gradle or Maven.
- Language**: Java (selected), Kotlin, Groovy.
- Dependencies**: A button to "ADD DEPENDENCIES... CTRL + B". Below it, a message says "No dependency selected".
- Spring Boot**: Version 3.2.5 selected from 3.0.0, 3.3.0, 3.3.0 RC1, 3.2.6, and 3.1.12.
- Project Metadata**: Fields for Group (com.example), Artifact (demo), Name (demo), and Description (Demo project for Spring Boot).
- Buttons at the bottom**: GENERATE (CTRL + ⌘), EXPLORE (CTRL + SPACE), and SHARE...

## 2. Realizamos las siguientes configuraciones:

This screenshot shows a more detailed configuration of the project:

- Project**: Maven selected.
- Language**: Java selected.
- Spring Boot**: Version 3.2.5 selected from 3.0.0, 3.3.0, 3.3.0 RC1, 3.2.6, and 3.1.12.
- Project Metadata**: Updated fields include Group (com.tesji.proyectoapi), Artifact (Proyecto API TESJI), Name (Proyecto API TESJI), Description (Proyecto para desarrollar una API consumida en una app con Android St), and Package name (com.tesji.proyectoapi.Proyecto API TESJI).
- Packaging**: War selected.
- Java**: Version 21 selected from 22, 21, and 17.

## 3. Agregamos las dependencias

The screenshot shows the "Dependencies" section with two items added:

- Spring Web WEB**: Described as "Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container." A red minus sign indicates it can be removed.
- Spring Data JPA SQL**: Described as "Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate." A red minus sign indicates it can be removed.

Traductor:

MySQL Driver SQL

MySQL JDBC driver.



4. Una vez agregadas las dependencias le damos en “Generate”.

The screenshot shows the configuration page for a Spring Boot project. Under 'Project', 'Maven' is selected. Under 'Spring Boot', '3.2.5' is selected. In the 'Project Metadata' section, the 'Group' is set to 'com.tesji.proyectoapi', 'Artifact' to 'Proyecto API TESJI', 'Name' to 'Proyecto API TESJI', 'Description' to 'Proyecto para desarrollar una API consumida en una app con Android St', 'Package name' to 'com.tesji.proyectoapi.Proyecto API TESJI', and 'Packaging' to 'War'. A warning message at the top states: '• 11 is not a valid Java version, 17 has been selected.  
• The following dependencies are not supported: flapdoodle-mongo.' On the right side, there are cards for 'Spring Web' (WEB), 'Spring Data JPA' (SQL), and 'MySQL Driver' (SQL). At the bottom, there are buttons for 'GENERATE' (CTRL + ⌘), 'EXPLORE' (CTRL + SPACE), and 'SHARE...'.

5. Después de esto se generará un archivo .zip

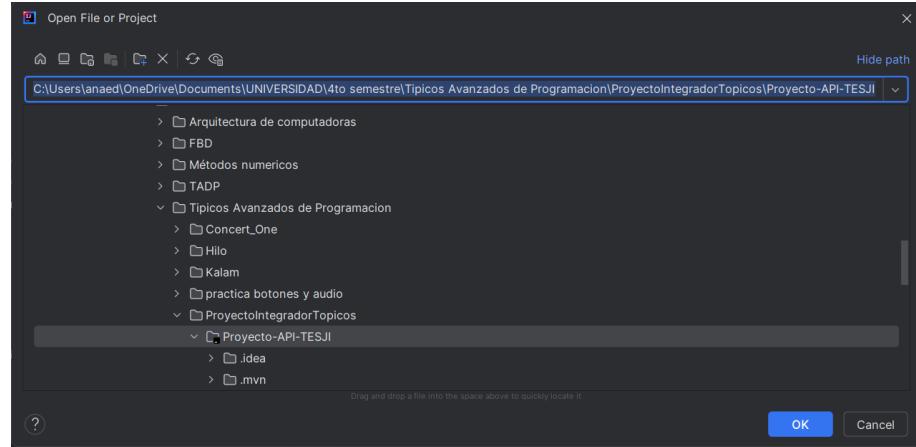
The screenshot shows the download history on start.spring.io. It lists a recent download of 'Proyecto API TESJI.zip' (17.7 KB, 6 minutes ago). The file is represented by a folder icon. There are icons for sharing, deleting, and opening the file.

23/05/2024

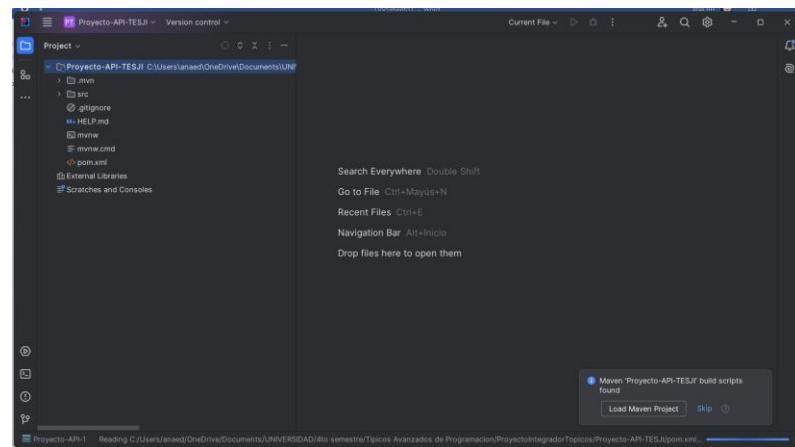
6. Abrir:

7. Creamos un nuevo proyecto con el API que acabamos de crear.
8. Descomprimimos el archivo .zip y lo pegamos en una carpeta que se llamara ProyectoIntegradorTopicos
9. Abrimos una carpeta:

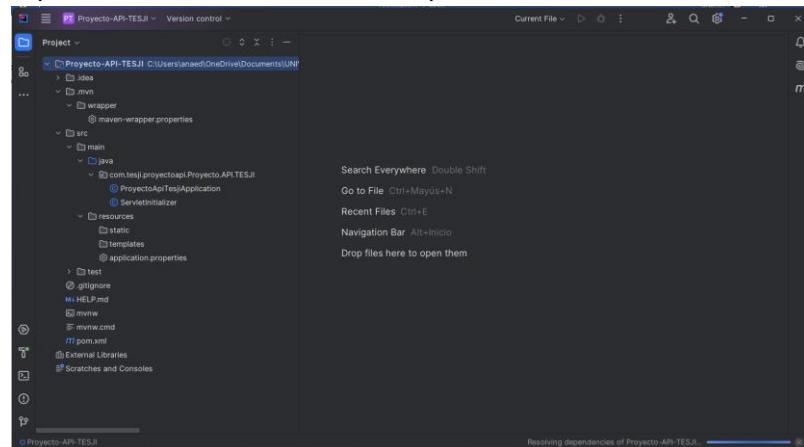
10. Copiamos la ubicación de donde tenemos el API y lo pegamos para abrir esa carpeta en la barra de búsqueda



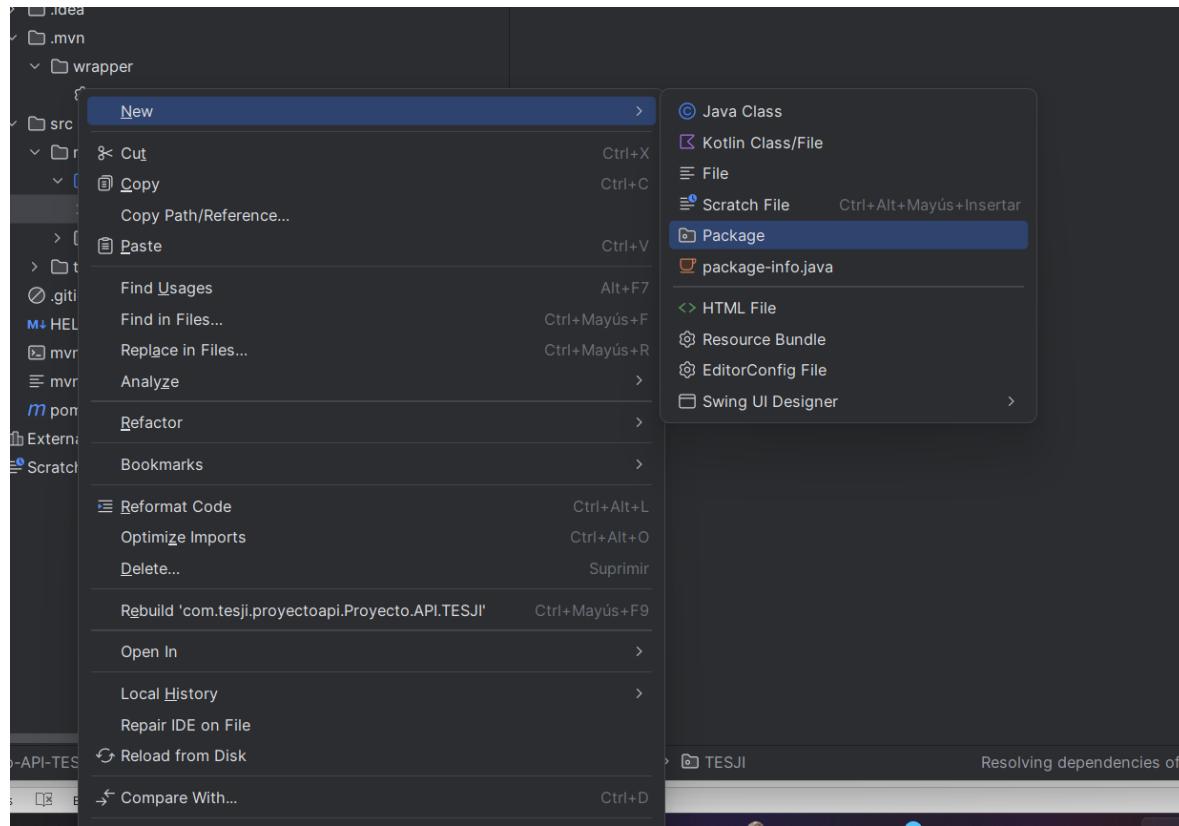
11. Comenzara a configurar el nuevo proyecto.



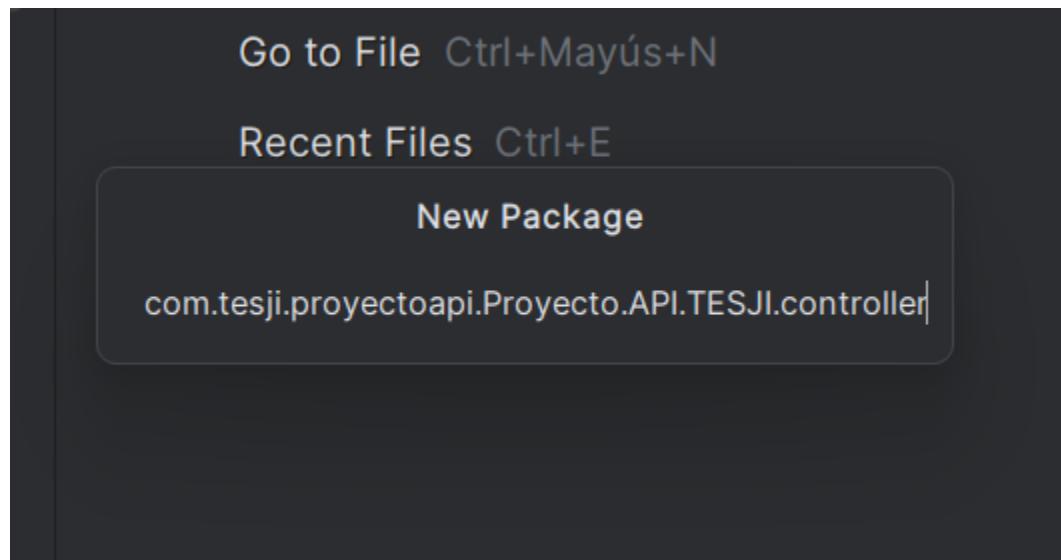
12. Y listo ya abrimos la carpeta en donde tenemos el API que se creó.



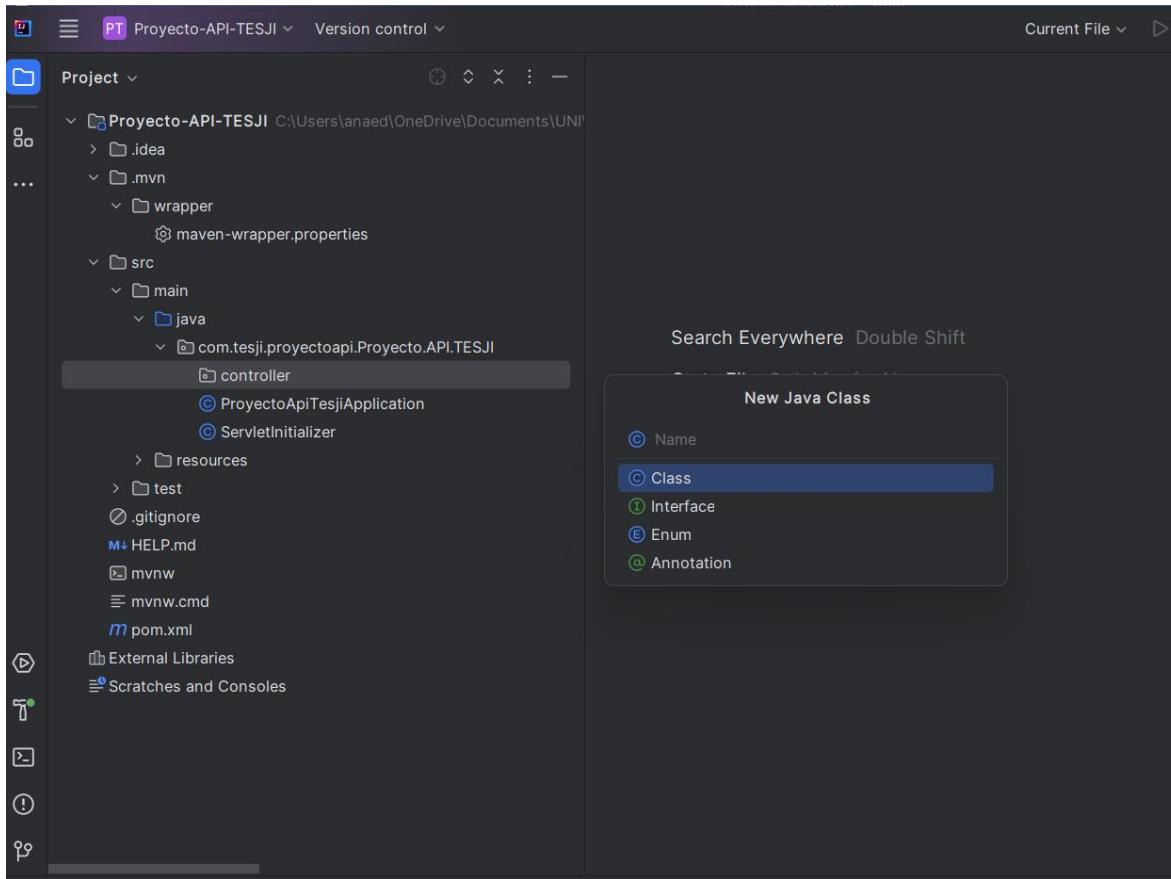
13. Crearemos un paquete en el paquete principal:



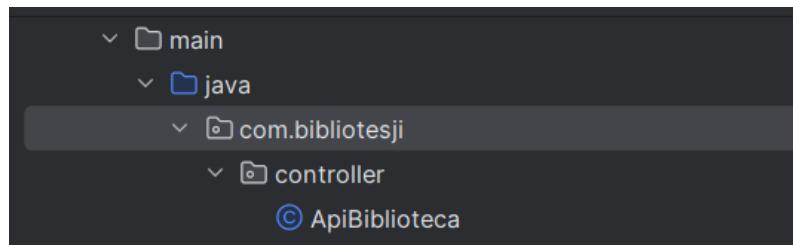
14. Ponemos el nombre



15. Creamos una clase en controller.



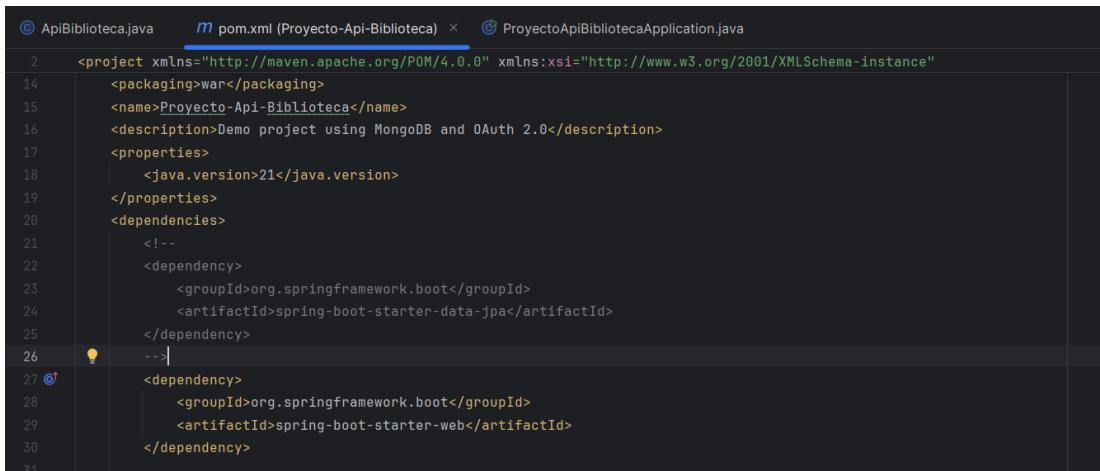
16. Dentro de ese paquete debemos crear una clase.



17. En esta nueva clase comenzaremos con la exportación de paquetes de GetMapping, RequestMapping y RestController.

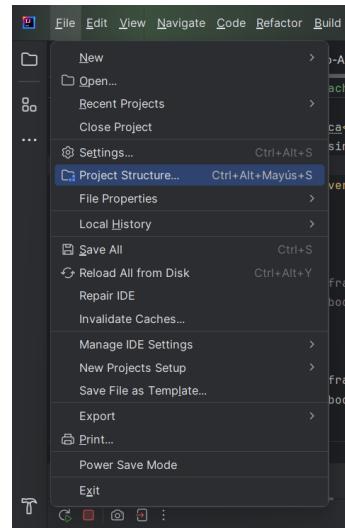
A screenshot of the IntelliJ IDEA code editor showing the 'ApiBiblioteca.java' file. The code is as follows:1 package com.bibliotesji.controller;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController no usages
8 @RequestMapping("biblioteji") //Define la URL de la API
9
10 public class ApiBiblioteca {
11 @GetMapping("/get-prueba") // Define el método como una peticion GET y su acceso no usages
12 public String pruebaApi(){
13 return "Funciona!! Hola Mundo, mi primera API";
14 }
15 }The code editor shows syntax highlighting for Java and Spring annotations. Other tabs visible include 'pom.xml (Proyecto-API-Biblioteca)' and 'ProyectoApiBibliotecaApplication.java'.

## 18. Comentamos una parte del código del código en la parte de pom.xml

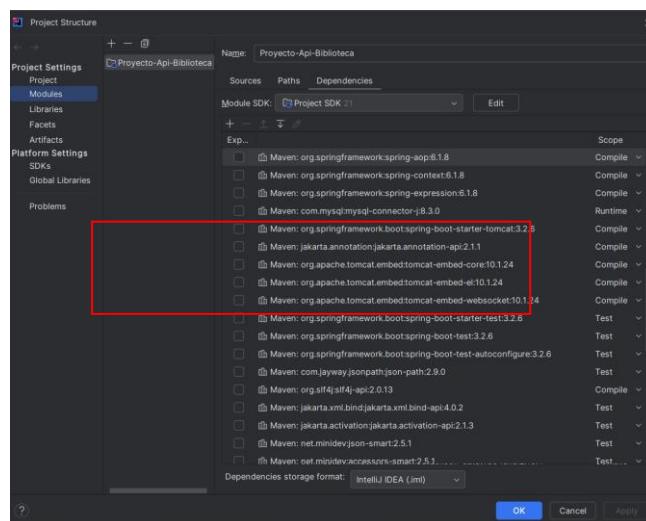


```
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
14 <packaging>war</packaging>
15 <name>Proyecto-Api-Biblioteca</name>
16 <description>Demo project using MongoDB and OAuth 2.0</description>
17 <properties>
18     <java.version>21</java.version>
19 </properties>
20 <dependencies>
21     <!--
22         <dependency>
23             <groupId>org.springframework.boot</groupId>
24             <artifactId>spring-boot-starter-data-jpa</artifactId>
25         </dependency>
26     -->
27     <dependency>
28         <groupId>org.springframework.boot</groupId>
29         <artifactId>spring-boot-starter-web</artifactId>
30     </dependency>
```

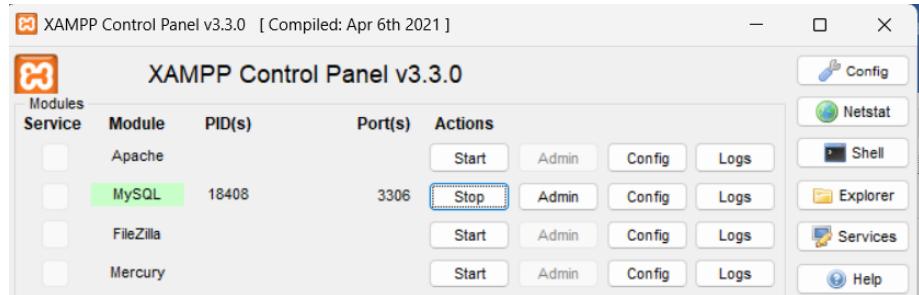
## 19. Vamos a File/ Project Structure



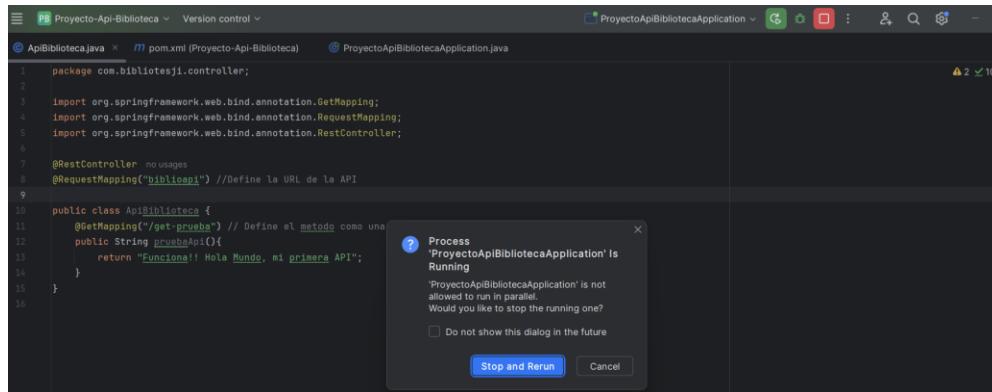
20. Clic en **Modules** y después en **Dependencies** y activamos las siguientes, tiene que aparecer en **Compile**, aplicamos y damos OK y reiniciamos en ID.



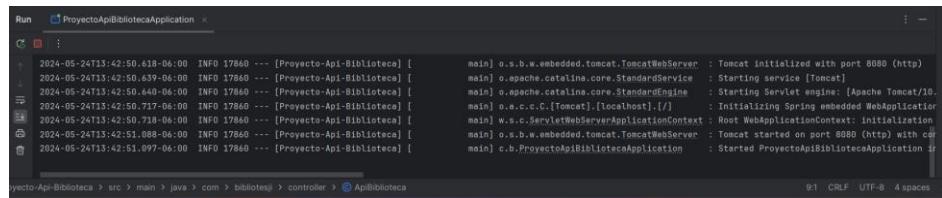
21. Despues ya podemos correr nuestra prueba, pero antes abrimos XAMPP y activamos MySQL.



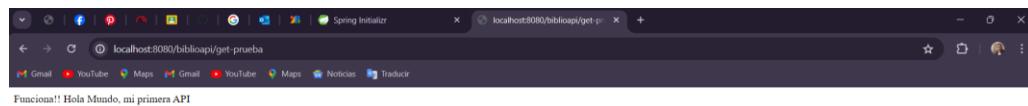
22. En nuestro proyecto damos clic en correr, clic en Stop and Rerun.



23. Comenzará a levantar la API.



24. Abrimos un Navegador y colocamos el siguiente link <http://localhost:8080/biblioapi/get-prueba>



25. La API ha funcionado correctamente.

26. Abrimos Postman, abrimos una nueva pestaña y colocamos el mismo link de arriba y también correrá la API.

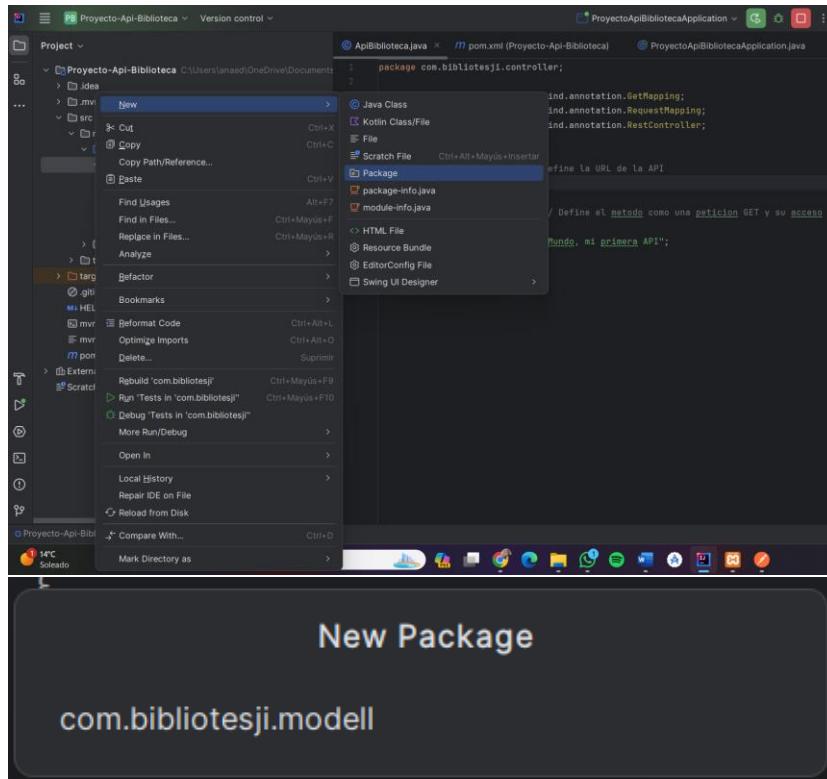
The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Overview', 'Getting started', 'GET Untitled Request', and a URL field containing 'GET http://localhost:8080/bibliapi/get-prueba'. Below the URL is a 'Send' button. The main area has tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Scripts', and 'Settings'. Under 'Headers', there's a 'Query Params' table:

Key	Value	Description	...	Bulk Edit
Key	Value	Description	...	

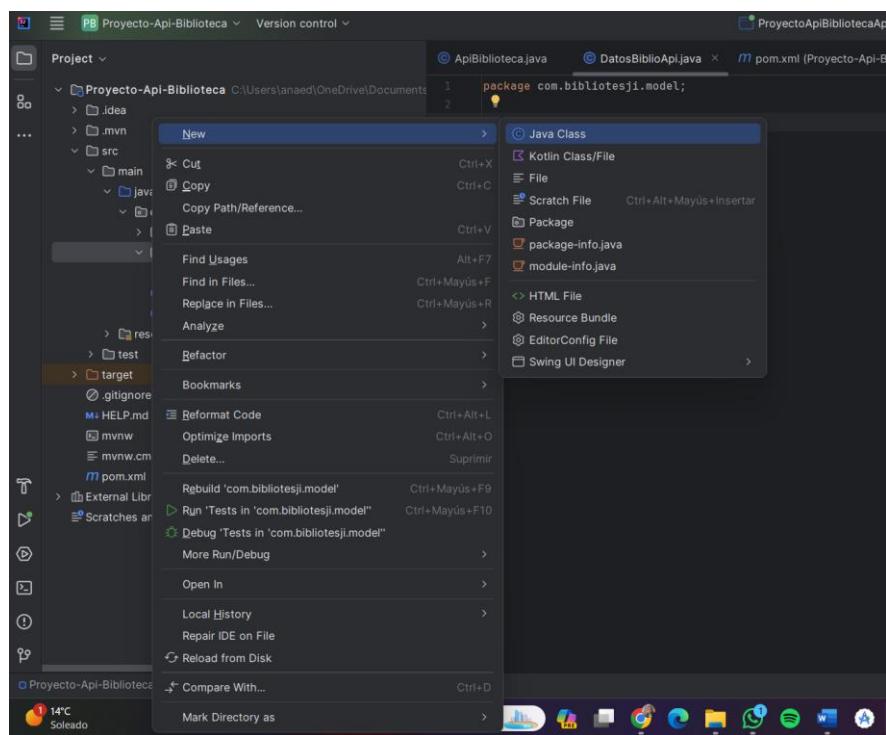
Below this, there's a 'Body' tab with sub-options: 'Cookies', 'Headers (5)', and 'Test Results'. The 'Test Results' section shows a status of 'Status: 200 OK', 'Time: 163 ms', and 'Size: 201 B'. It contains a text area with the response: '1 Funciona!! Hola Mundo, mi primera API'. At the bottom of the window, there's a toolbar with various icons and a system tray showing the date and time as '01:49 p. m. 24/05/2024'.

24/05/2024

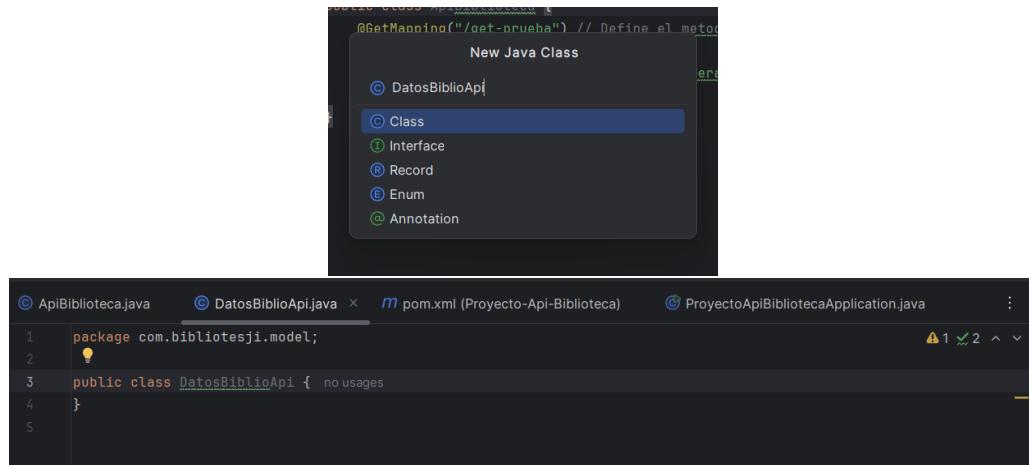
27. Creamos un nuevo paquete con el nombre de model. En este paquete definiremos los atributos.



28. Cremamos una nueva clase.



29. Con el nombre:



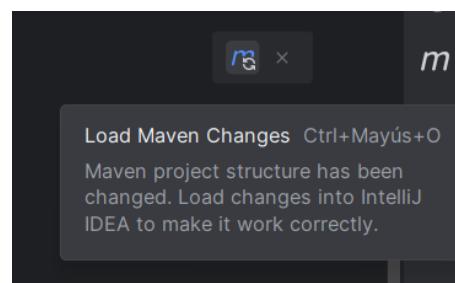
30. Comenzaremos a agregar los atributos estos serán encapsulados.



31. Vamos al archivo pom.xml y quitamos la documentación de:



32. Después actualizamos.



33. Agregamos la notación de @Entity, esta convierte la clase en una tabla con los atributos definidos.

```
34. @Entity
```

35. El atributo id lo convierte a una llave primaria.

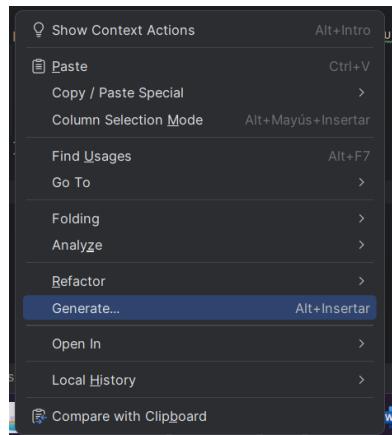
```
@Id
```

Un campo java nunca puede estar vacío.

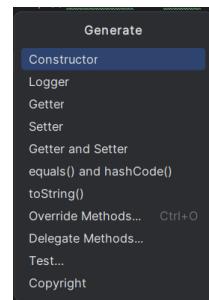
36. Identificador autoincrementable.

```
37. @GeneratedValue(strategy = GenerationType.IDENTITY)
```

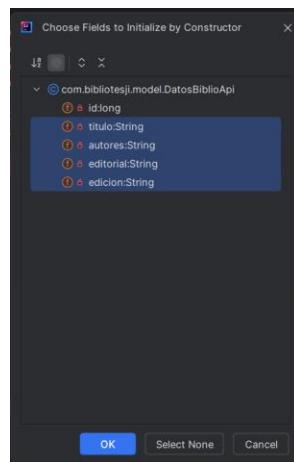
38. Generaremos un Constructor, clic derecho.



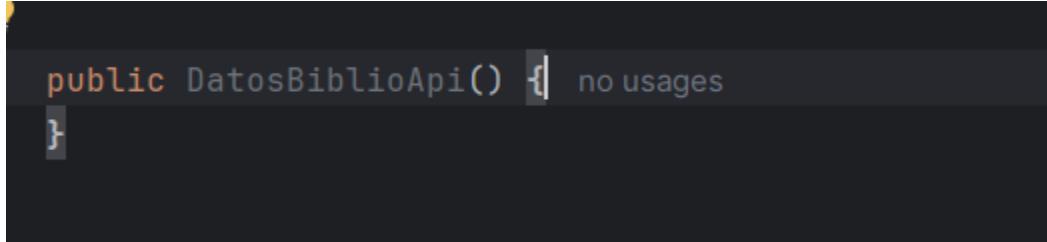
39. Clic en Generate



40. Seleccionamos los atributos:



41. Definiremos nuestro propio constructor.

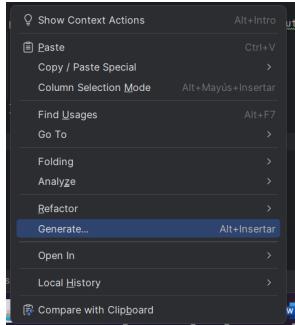


```
public DatosBiblioApi() { no usages }
```

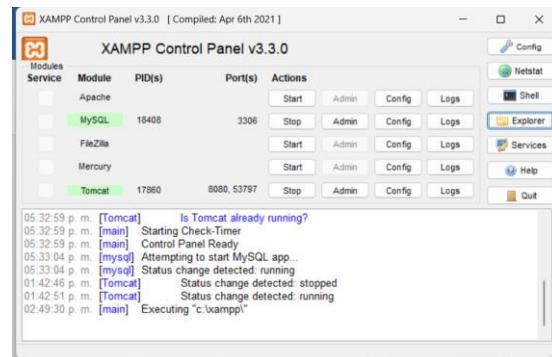
Por buena práctica el constructor que creamos nosotros debe de ir primero.

42. Ultimo paso para que quede el modelo de clase, crearemos los get y set de cada atributo.

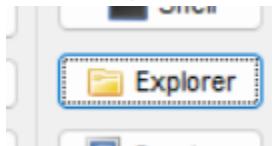
- Clic derecho.



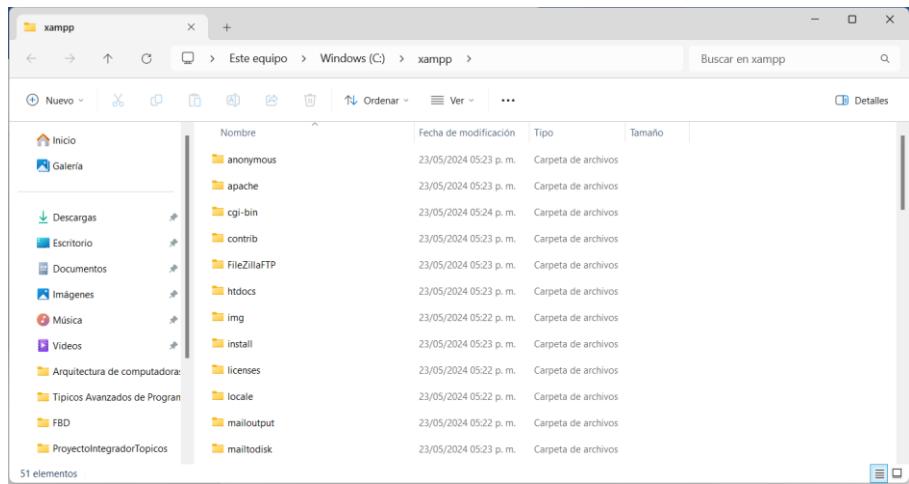
43. Abrimos XAMPP



44. Clic en Explorer



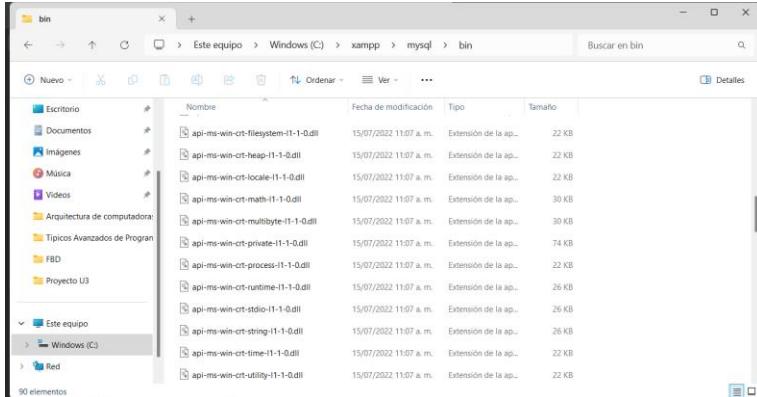
45. Abrirá la carpeta en donde esta guardado XAMPP.



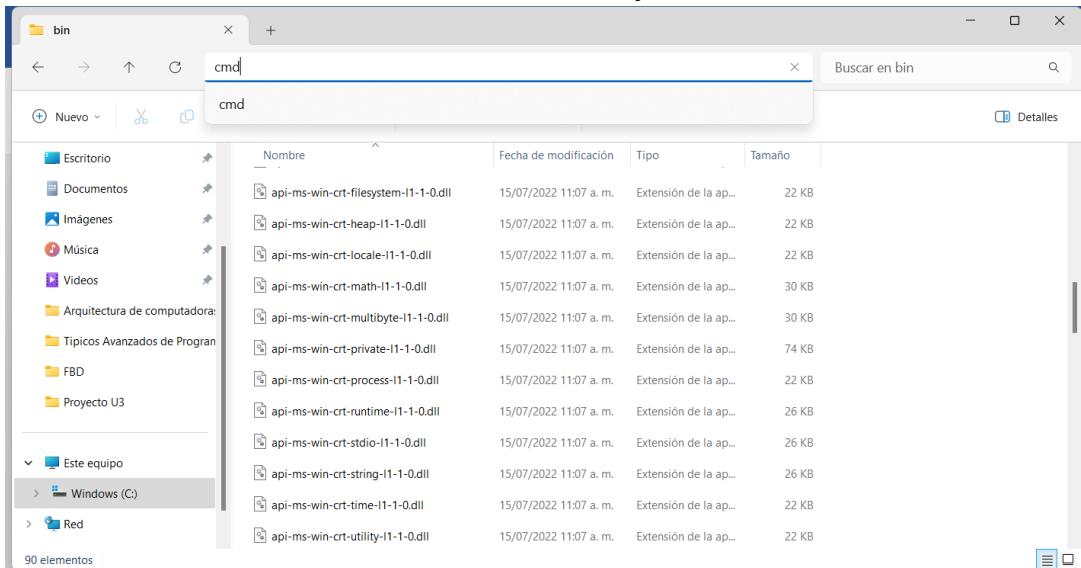
46.

31/05/2024

47. Entramos a XAMPP desde explorador de archivos.



48. Colocamos cmd en donde va la ruta de ubicación y te abrirá una terminal con esa ruta.



49. Colocamos lo siguiente:

```
Microsoft Windows [Versión 10.0.22631.3593]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\xampp\mysql\bin>mysql -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

50. Ahora abriremos las bases de datos que tengamos

```
MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database          |
+-----+
| biblioapi_db     |
| information_schema|
| mysql             |
| performance_schema|
| phpmyadmin        |
| test              |
+-----+
6 rows in set (0.020 sec)
```

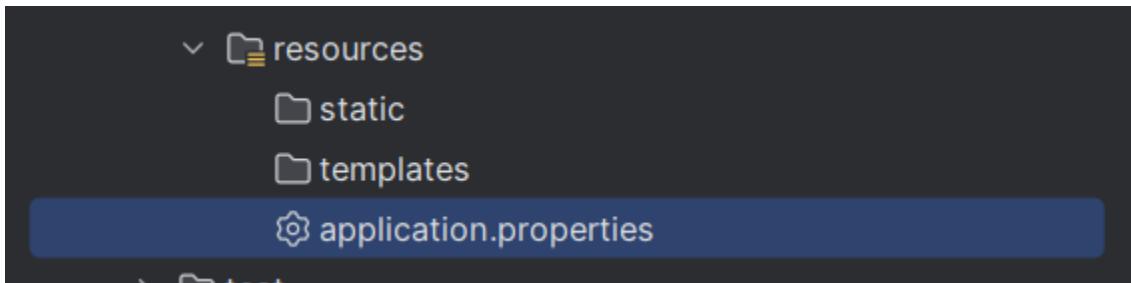
51. Inicializamos la base de datos.

```
MariaDB [(none)]> USE biblioapi_db;  
Database changed
```

52. Colocamos lo siguiente para ver cuantas tablas tiene nuestra base de datos

```
MariaDB [biblioapi_db]> SHOW TABLES;  
Empty set (0.011 sec)
```

53. En nuestro proyecto iremos a **resources/application.properties** que contiene los parámetros de conexión de la base de datos



54. Colocaremos las siguientes líneas de código

```
spring.application.name=Proyecto-Api-Biblioteca  
spring.datasource.url=jdbc:mysql://localhost:3306/biblioapi_db  
spring.datasource.username=root  
spring.datasource.password=  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

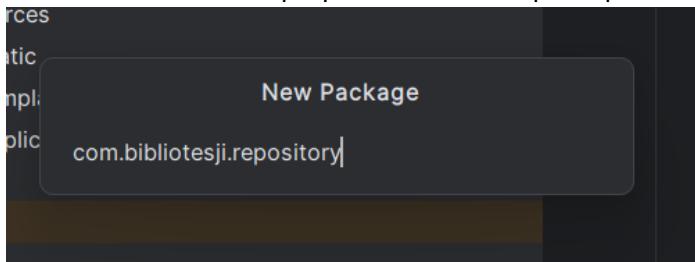
55. Despues abriremos la terminal en donde inicializamos la base de datos y colocamos y mostrara la tabla que creo en automático.

```
MariaDB [biblioapi_db]> SHOW TABLES;  
+-----+  
| Tables_in_biblioapi_db |  
+-----+  
| datos_biblio_api |  
+-----+  
1 row in set (0.001 sec)
```

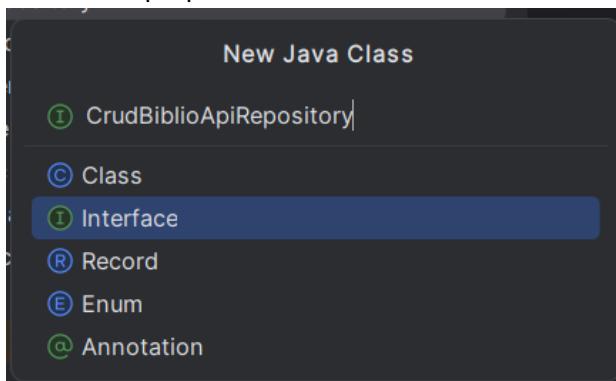
56. Posterior a eso colocaremos **DESCRIBE datos\_biblio\_api;** para ver los atributos que inicializamos desde nuestro proyecto

```
MariaDB [biblioapi_db]> DESCRIBE datos_biblio_api;  
+-----+-----+-----+-----+-----+-----+  
| Field | Type | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| id | bigint(20) | NO | PRI | NULL | auto_increment |  
| autores | varchar(255) | YES | | NULL | |  
| edicion | varchar(255) | YES | | NULL | |  
| editorial | varchar(255) | YES | | NULL | |  
| titulo | varchar(255) | YES | | NULL | |  
+-----+-----+-----+-----+-----+-----+  
5 rows in set (0.023 sec)
```

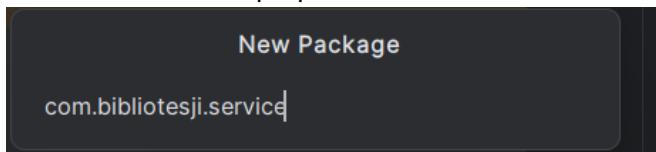
57. Crearemos un nuevo paquete en la clase principal



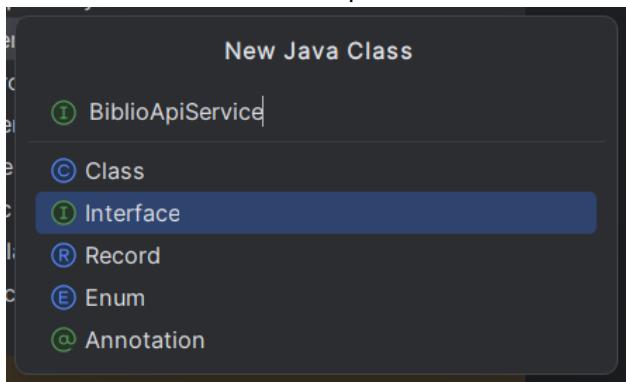
58. Dentro del paquete crearemos una nueva clase de tipo **interface**.



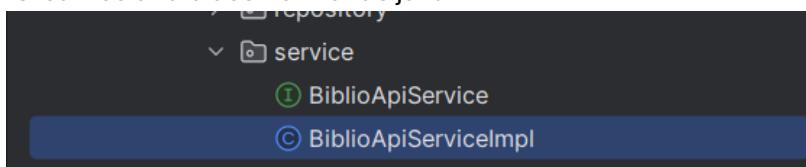
59. Creamos un nuevo paquete



60. Crear una nueva clase de tipo **interface** dentro de ese paquete



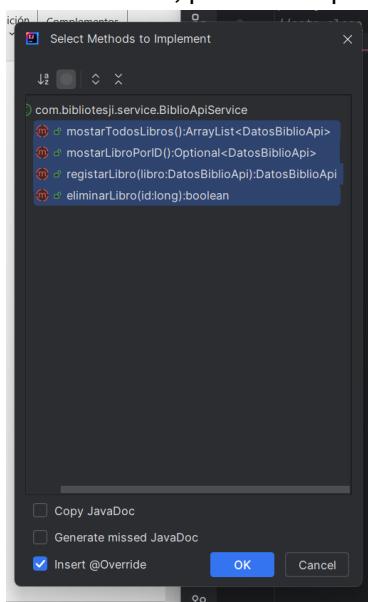
61. Creamos una clase normal de java



62. Creada la clase la implementaremos con Biblio ApiService

```
public class BiblioApiServiceImpl implements BiblioApiService{
```

63. Marcará error, pero tiene que llamar sus métodos de esa clase, usamos la ayuda y clic en OK.



- #### 64. Importamos @Service

`@Service` no usages

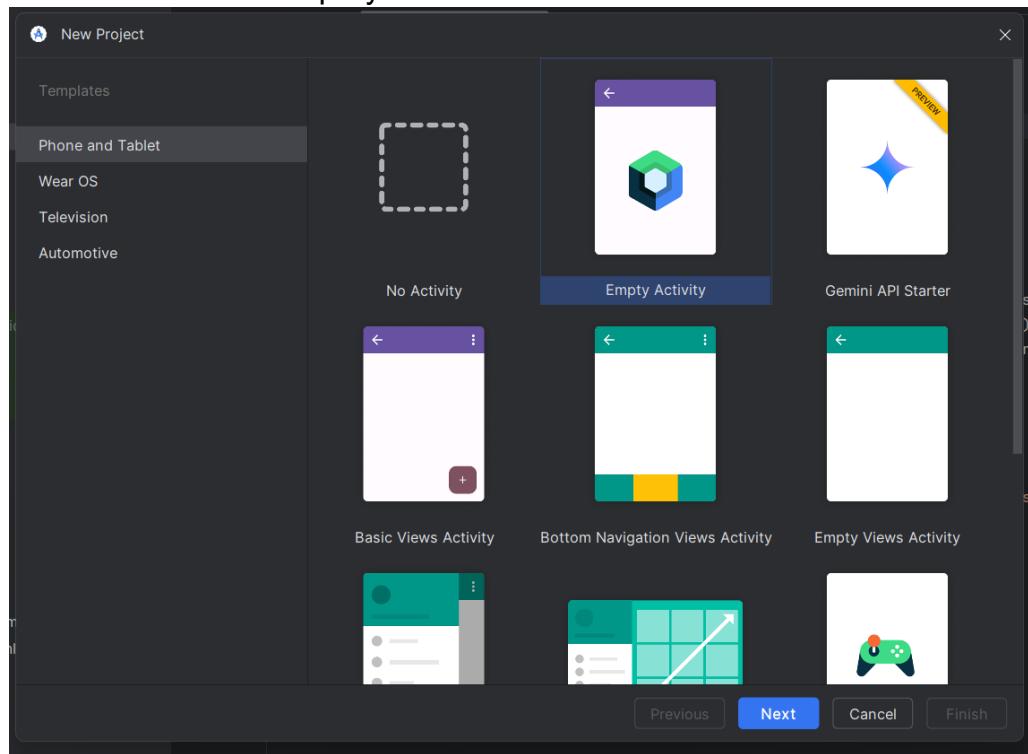
```
import org.springframework.stereotype.Service;
```

- #### 65. En la clase ApiBiblioteca.java

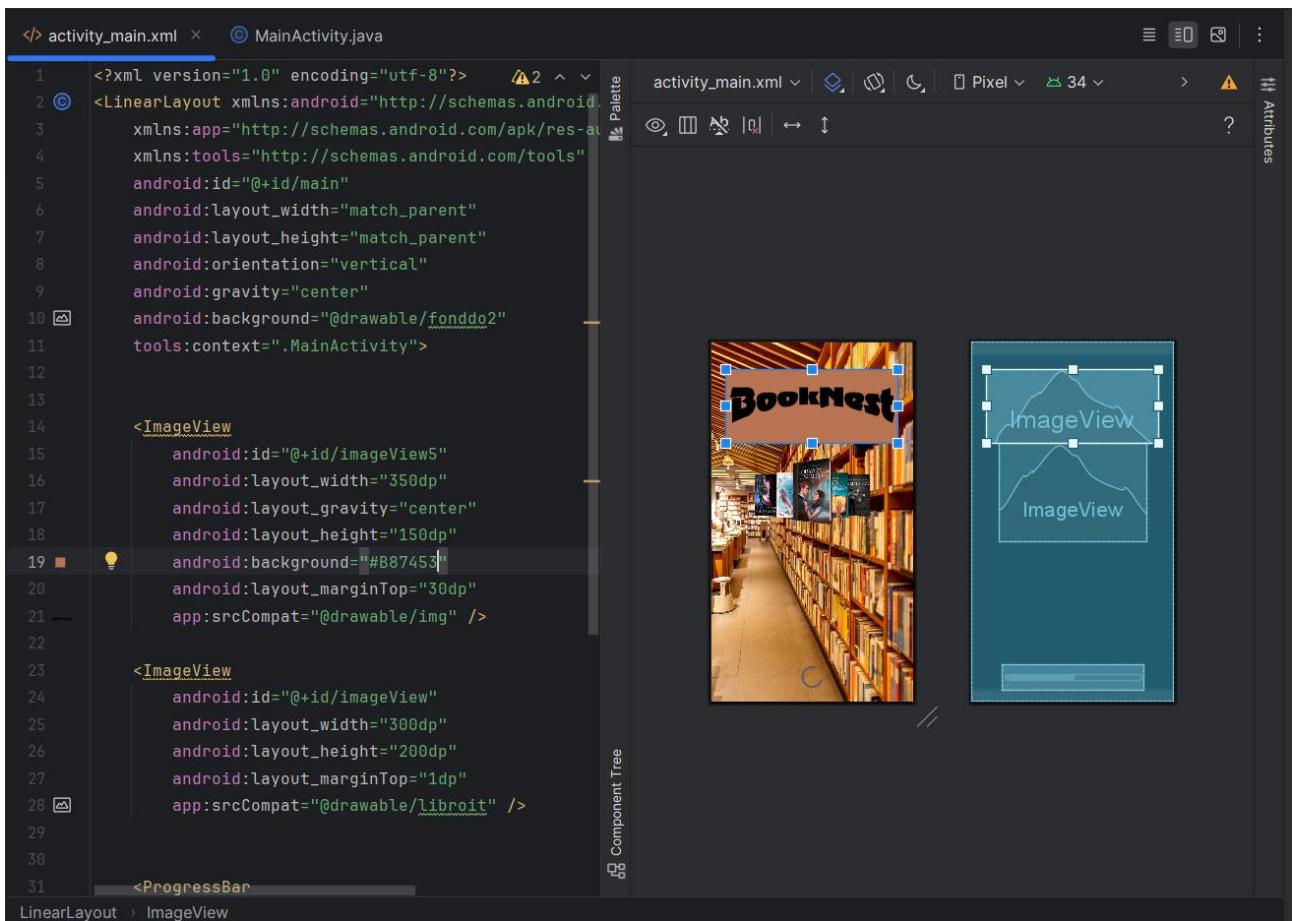
```
Structure Service no usages
public class BiblioApiServiceImpl implements Biblio ApiService{
    //Inyectar una clase para ocuparla como objeto (ya no se requiere instanciar)
    //El objeto hereda los métodos CRUD
    @Autowired 4 usages
    CrudBiblioApiRepository crudBiblioApiRepository;
    @Override 1 usage
    public ArrayList<DatosBiblioApi> mostrarTodosLibros() {
        return (ArrayList<DatosBiblioApi>) crudBiblioApiRepository.findAll();
    }
    @Override 2 usages
    public Optional<DatosBiblioApi> mostrarLibroPorID(long id) {
        return crudBiblioApiRepository.findById(id);
    }
    @Override 1 usage
    public DatosBiblioApi registrarLibro(DatosBiblioApi libro) {
        return crudBiblioApiRepository.save(libro);
    }
    @Override 1 usage
    public boolean eliminarLibro(long id) {
        try {
            //buscar el libro
            Optional<DatosBiblioApi> libro = mostrarLibroPorID(id);
            //Eliminar
            crudBiblioApiRepository.delete(libro.get());
            return true;
        } catch (Exception err){
            return false;
        }
    }
}
```

# CONECTAR LA API BIBLIOTESTJI CON UN PROYECTO DE ANDROID STUDIO

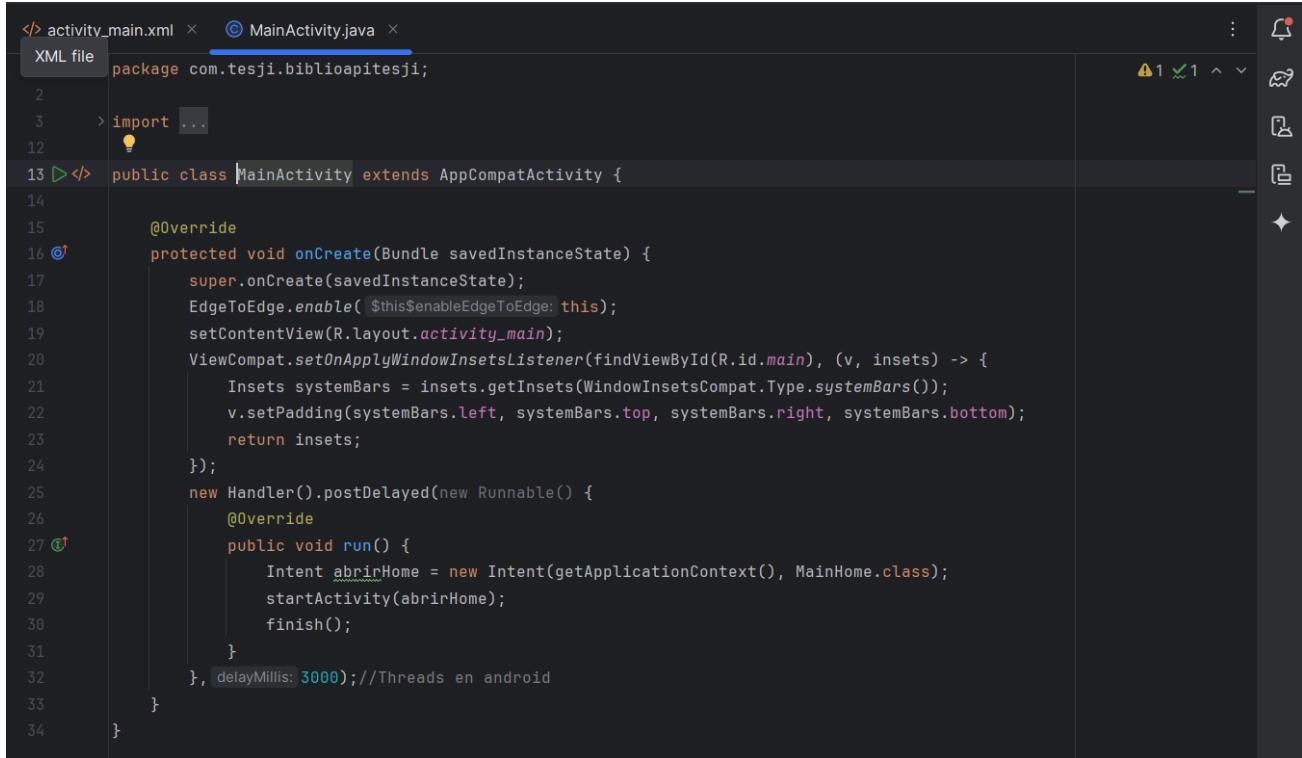
- Primero crearemos un nuevo proyecto.



- Posterior comenzaremos con la pantalla de Hacemos su diseño



Al igual que su MainActivity.java



```
<activity_main.xml x:>MainActivity.java x:
```

```
XML file package com.tesji.biblioapitesji;
2
3     > import ...
12
13 <> public class MainActivity extends AppCompatActivity {
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         EdgeToEdge.enable( $this$enableEdgeToEdge: this);
19         setContentView(R.layout.activity_main);
20         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
21             Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
22             v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
23             return insets;
24         });
25         new Handler().postDelayed(new Runnable() {
26             @Override
27             public void run() {
28                 Intent abrirHome = new Intent(getApplicationContext(), MainHome.class);
29                 startActivity(abrirHome);
30                 finish();
31             }
32         }, delayMillis: 3000); //Threads en android
33     }
34 }
```

Explicación del método onCreate:

1. super.onCreate(savedInstanceState);: Llama al método onCreate de la clase base para realizar la inicialización estándar de la actividad.
2. EdgeToEdge.enable(this);: Este método habilita el modo Edge-to-Edge en la actividad, permitiendo que la UI de la actividad se extienda hasta las áreas ocupadas por las barras del sistema (como la barra de estado y la barra de navegación).
3. setContentView(R.layout.activity\_main);: Establece el archivo de diseño XML activity\_main.xml como la vista principal de esta actividad.
4. ViewCompat.setOnApplyWindowInsetsListener(...);: Esta línea configura un listener para ajustar el padding del contenido principal de la vista (R.id.main) para que tenga en cuenta las barras del sistema (como la barra de estado y la barra de navegación), asegurando que el contenido no se solape con estas barras.

findViewById(R.id.main): Encuentra la vista con el ID main.

Lambda ( (v, insets) -> {...} ): Especifica qué hacer cuando se aplican los insets (bordes del sistema). Aquí, se ajusta el padding de la vista v (la vista principal) según los insets obtenidos de WindowInsetsCompat.Type.systemBars().

5. new Handler().postDelayed(new Runnable() { ... }, 3000);: Utiliza un Handler para ejecutar un Runnable después de un retraso de 3000 milisegundos (3 segundos).

new Runnable() { @Override public void run() { ... } }: Define la tarea que se ejecutará después del retraso.

Intent abrirHome = new Intent(getApplicationContext(), MainHome.class);: Crea un nuevo Intent para iniciar la actividad MainHome.

startActivity(abrirHome);: Inicia la actividad MainHome.

finish();: Finaliza la actividad actual (MainActivity), eliminándola de la pila de actividades y previniendo que el usuario vuelva a ella presionando el botón "Atrás".

- Después crearemos nuestro MainHome.java, este yo puse una pequeña descripción e los que hace la app.

```
>MainHome.java
1 package com.tesji.biblioapitesji;
2
3
4 > import ...
5
6
7
8 > public class MainHome extends AppCompatActivity {
9
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.main_home);
15
16         Button startButton = findViewById(R.id.startButton);
17         startButton.setOnClickListener(new View.OnClickListener() {
18             @Override
19             public void onClick(View v) {
20                 Intent intent = new Intent(getApplicationContext(), MainActivity3.class);
21                 startActivity(intent);
22             }
23         });
24     }
25 }
26 }
```

### Explicación

- `@Override`: Indica que este método sobrescribe el método `onCreate` de la clase base.
- `protected void onCreate(Bundle savedInstanceState)`: El método `onCreate` se llama cuando se crea la actividad. Aquí es donde se inicializa la actividad.
- `super.onCreate(savedInstanceState)`: Llama al método `onCreate` de la clase base para realizar la inicialización estándar de la actividad.
- `setContentView(R.layout.main_home)`: Establece el archivo de diseño XML `main_home.xml` como la vista principal de esta actividad.

### Botón

- `Button startButton = findViewById(R.id.startButton);`: Encuentra la vista con el ID `startButton` en el archivo de diseño `main_home.xml` y la asigna a la variable `startButton`.
- `startButton.setOnClickListener(new View.OnClickListener() { ... })`: Establece un listener para el botón que se ejecutará cuando el botón sea clicado.

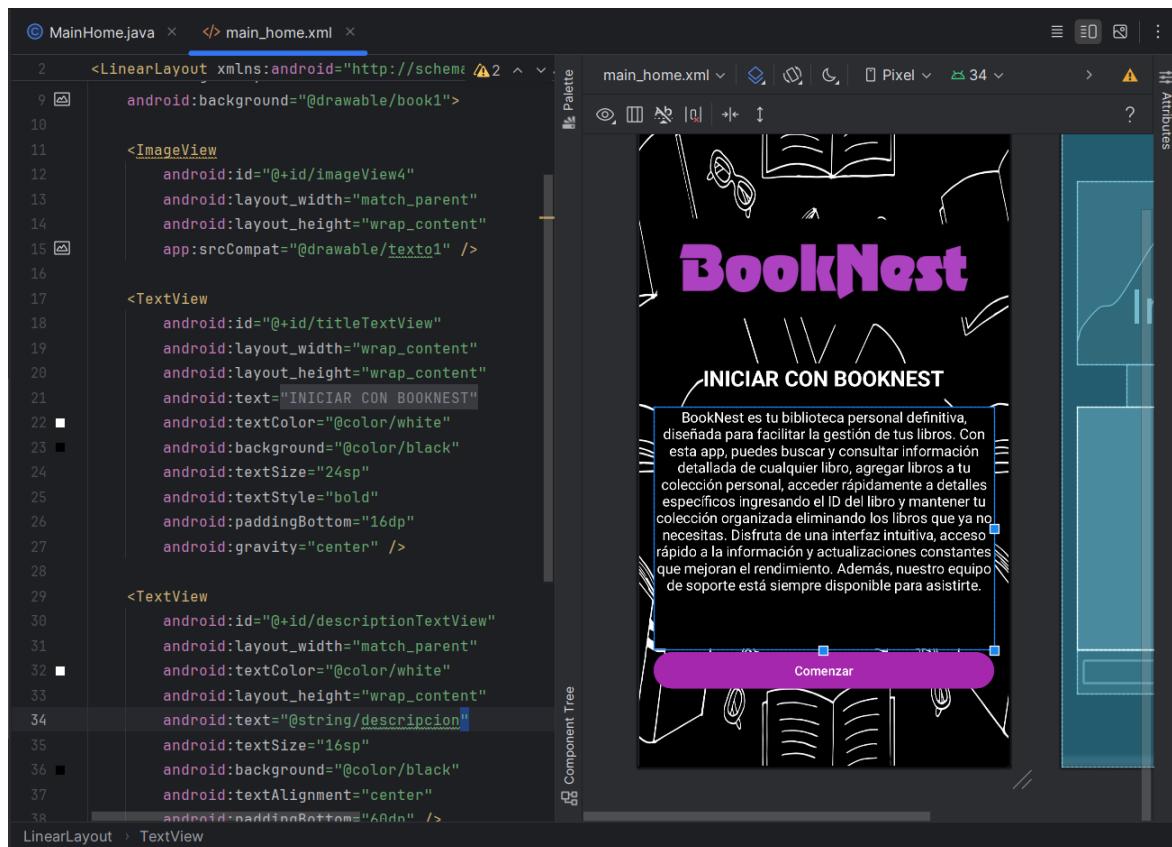
`new View.OnClickListener() { ... }:` Crea una nueva instancia de `View.OnClickListener`.

`public void onClick(View v) { ... }:` Define el método `onClick` que se ejecutará cuando se haga clic en el botón.

Intent intent = new Intent(MainHome.this, MainActivity3.class);: Crea un nuevo Intent para iniciar la actividad MainActivity3.

startActivity(intent);: Inicia la actividad MainActivity3.

- Agregamos su diseño



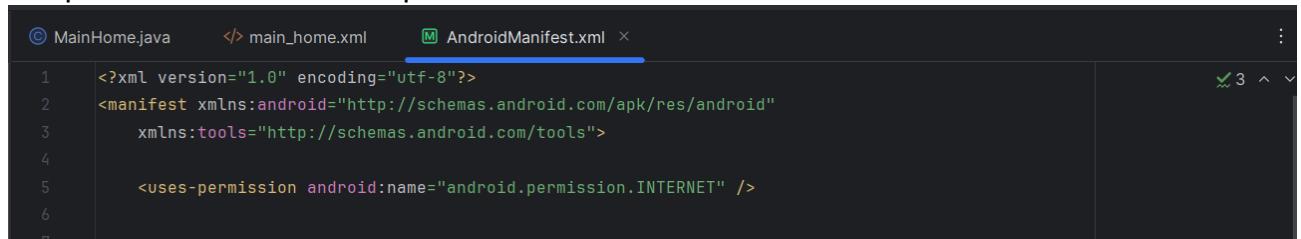
- Despu s de tener eso, necesitamos sincronizar con lo que haremos la conexi n de la API (dependencias), ser a de la siguiente manera:

```
34     dependencies {
35         implementation(libs.appcompat)
36         implementation(libs.material)
37         implementation(libs.activity)
38         implementation(libs.constraintlayout)
39
40         implementation("com.squareup.okhttp3:okhttp:4.9.3")
41         implementation("com.squareup.retrofit2:retrofit:2.9.0")
42         implementation("com.squareup.retrofit2:converter-gson:2.9.0")
43
44         implementation("com.github.bumptech.glide:glide:4.12.0")
45         implementation(libs.firebaseio.inappmessaging)
46
47         testImplementation(libs.junit)
48         androidTestImplementation(libs.ext.junit)
49         androidTestImplementation(libs.espresso.core)
50     }
```

El bloque dependencies en un archivo build.gradle especifica las bibliotecas y versiones que el proyecto Android necesita para compilarse y ejecutarse. Aqu  se incluyen implementaciones para las bibliotecas de interfaz de usuario de Android como.appcompat, material, activity, y constraintlayout;

bibliotecas de red y JSON como okhttp y retrofit junto con el convertidor gson; la biblioteca de carga de imágenes Glide; y la integración con Firebase para mensajería en la aplicación. También se incluyen dependencias para pruebas unitarias (junit) y pruebas instrumentadas en Android (ext.junit y espresso.core). Estas dependencias permiten a los desarrolladores utilizar funcionalidades y herramientas específicas proporcionadas por estas bibliotecas en su aplicación Android.

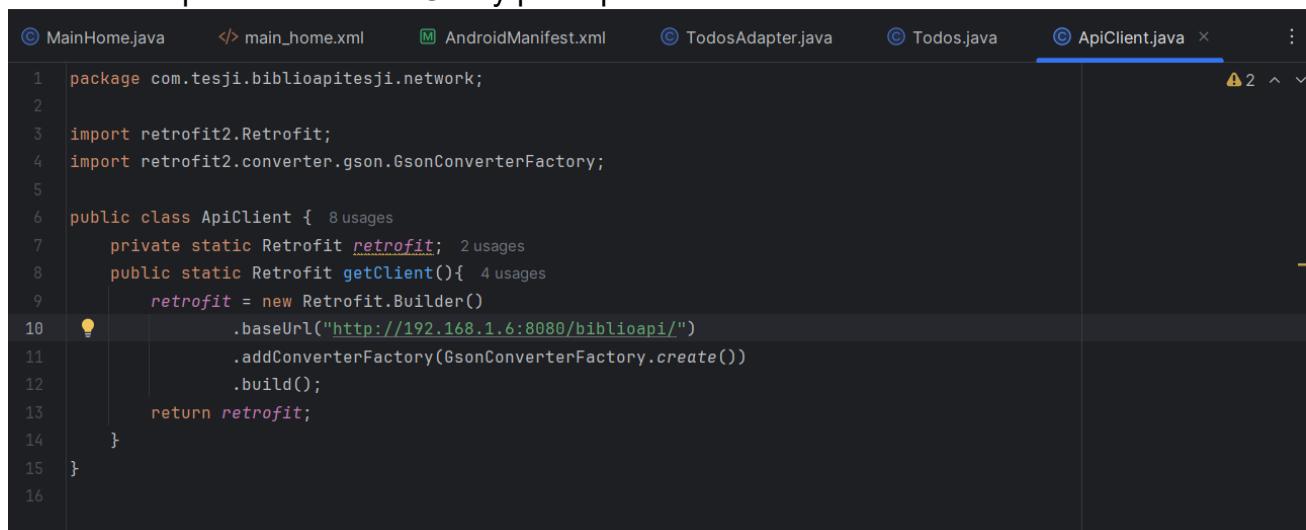
- Despues de eso damos los permisos de internet



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

    <uses-permission android:name="android.permission.INTERNET" />
```

- Creamos un paquet que tenga el nombre de **netwoks**, en este haremos la conexión de nuestra API por medio de su URL y para que funcione correctamente nuestra IP local.



```
package com.tesji.biblioapitesji.network;

import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class ApiClient { 8 usages
    private static Retrofit retrofit; 2 usages
    public static Retrofit getClient(){ 4 usages
        retrofit = new Retrofit.Builder()
            .baseUrl("http://192.168.1.6:8080/biblioapi/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();
        return retrofit;
    }
}
```

- Creamos un paquete llamado **model** en donde haremos la inicializaremos a nuestros datos de la Api.

```
>MainHome.java  </> main_home.xml  AndroidManifest.xml  TodosAdapter.java  Todos.java  ApiClient.java
```

```
1 package com.tesji.biblioapitesji.model;
2
3 public class Todos {
4     private String id; 2 usages
5     private String titulo; 2 usages
6     private String autores; 2 usages
7     private String editorial; 2 usages
8     private String edicion; 2 usages
9
10    public String getId() { return id; }
11
12    public void setId(String id) { this.id = id; }
13
14    public String getTitulo() { 2 usages
15        return titulo;
16    }
17
18    public void setTitulo(String titulo) { this.titulo = titulo; }
19
20    public String getAutores() { return autores; }
21
22    public void setAutores(String autores) { this.autores = autores; }
23
24    public String getEditorial() { return editorial; }
25
26    public void setEditorial(String editorial) { this.editorial = editorial; }
27
28    public String getEdicion() { return edicion; }
29
30    public void setEdicion(String edicion) { this.edicion = edicion; }
31
32}
33
34}
35
36}
37
38}
39
40}
41
42}
43
44}
45
46}
47
48}
49
50}
```

- Creamos un paquete que lleve el nombre de **adapter**

## Importaciones

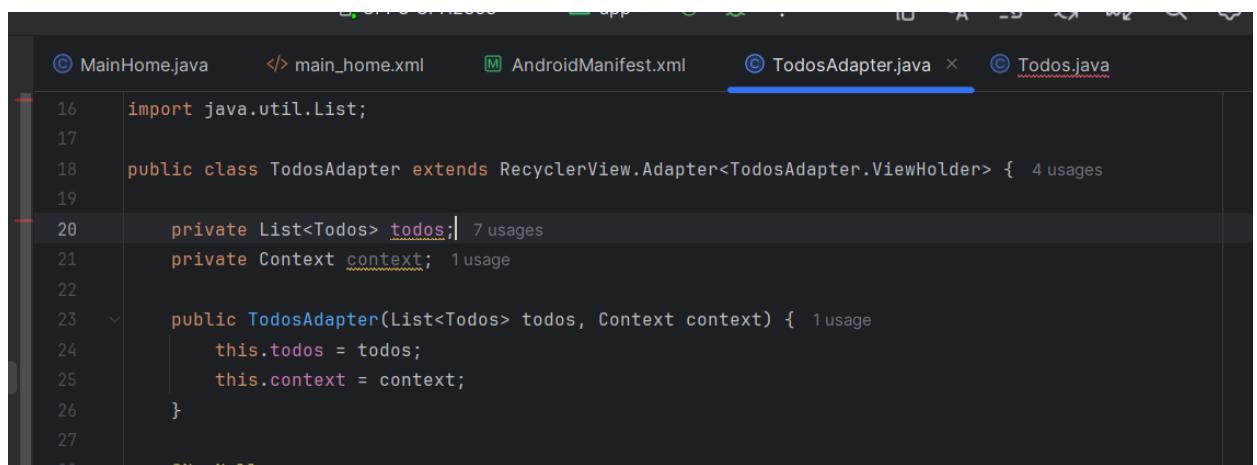
```
>MainHome.java  </> main_home.xml  AndroidManifest.xml  TodosAdapter.java  Todos.java
```

```
1 package com.tesji.biblioapitesji.adapter;
2
3 import android.content.Context;
4 import android.view.LayoutInflater;
5 import android.view.View;
6 import android.view.ViewGroup;
7 import android.widget.TextView;
8
9 import androidx.annotation.NonNull;
10 import androidx.recyclerview.widget.RecyclerView;
11
12 import com.bumptech.glide.Glide;
13 import com.tesji.biblioapitesji.R;
14 import com.tesji.biblioapitesji.model.Todos;
15
16 import java.util.List;
17
18 public class TodosAdapter extends RecyclerView.Adapter<TodosAdapter.ViewHolder> { 4 usages
19
20     private List<Todos> todos; 7 usages
21     private Context context; 1 usage
22
23     public TodosAdapter(List<Todos> todos, Context context) { 1 usage
```

Estas líneas importan las clases necesarias para que el adaptador funcione:

- `Context`, `LayoutInflater`, `View`, `ViewGroup`, `TextView`: Utilizadas para trabajar con la interfaz de usuario.
- `RecyclerView`: Biblioteca de soporte para listas.
- `R`: Acceso a los recursos, como las vistas de diseño.
- `Todos`: El modelo de datos.

## Definición de la Clase `TodosAdapter`



```
16 import java.util.List;
17
18 public class TodosAdapter extends RecyclerView.Adapter<TodosAdapter.ViewHolder> { 4 usages
19
20     private List<Todos> todos; 7 usages
21     private Context context; 1 usage
22
23     public TodosAdapter(List<Todos> todos, Context context) { 1 usage
24         this.todos = todos;
25         this.context = context;
26     }
27 }
```

- `List<Todos> todos`: Lista de objetos `Todos` que se van a mostrar en el `RecyclerView`.
- `Context context`: Contexto de la aplicación o actividad.
- `TodosAdapter(List<Todos> todos, Context context)`: Constructor del adaptador que inicializa los miembros `todos` y `context` .

## Método `onCreateViewHolder`

```
27
28     @NonNull
29     @Override
30     public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
31         View view= LayoutInflater.from(parent.getContext())
32             .inflate(R.layout.item_libros,parent, attachToRoot: false);
33         return new ViewHolder(view);
34     }
35 }
```

- `onCreateViewHolder`: Este método infla el diseño de cada elemento de la lista (`item\_libros`) y crea una instancia de `ViewHolder` .
- `LayoutInflater.from(parent.getContext()).inflate(R.layout.item\_libros, parent, false);` : Infla el diseño `item\_libros.xml` .
- `return new ViewHolder(view);` : Devuelve una nueva instancia de `ViewHolder` .

## Método `onBindViewHolder`

```
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    holder.tv_id.setText(todos.get(position).getId());
```

```

        holder.tv_titulo.setText(todos.get(position).getTitulo());
        holder.tv_autores.setText(todos.get(position).getAutores());
        holder.tv_editorial.setText(todos.get(position).getEditorial());
        holder.tv_edicion.setText(todos.get(position).getEdicion());
    }
}

```

- `onBindViewHolder`: Este método enlaza los datos del objeto `Todos` en la posición especificada con las vistas en el `ViewHolder`.
  - `holder.tv\_id.setText(todos.get(position).getId());`: Establece el texto del `TextView` con el ID del libro.
  - Similarmente, las otras líneas establecen los valores para el título, autores, editorial y edición del libro.

#### Método `getItemCount`

```

@Override
public int getItemCount() {
    return todos.size();
}

```

- `getItemCount`: Devuelve el número total de elementos en la lista `todos`.

#### Clase `ViewHolder`

```

public class ViewHolder extends RecyclerView.ViewHolder {
    private TextView tv_id;
    private TextView tv_titulo;
    private TextView tv_autores;
    private TextView tv_editorial;
    private TextView tv_edicion;

    public ViewHolder(@NonNull View itemView) {
        super(itemView);
        tv_id = itemView.findViewById(R.id.idlibro);
        tv_titulo = itemView.findViewById(R.id.titulolibro);
        tv_autores = itemView.findViewById(R.id.autoreslibro);
        tv_editorial = itemView.findViewById(R.id.editoriallibro);
        tv_edicion = itemView.findViewById(R.id.edicionlibro);
    }
}
```

```

- `ViewHolder`: Clase interna que representa la vista de cada elemento en el `RecyclerView`.
- `ViewHolder(View itemView)`: Constructor que inicializa las vistas de cada elemento. Usa `findViewById` para obtener referencias a los `TextView` de `item\_libros.xml`.

Screenshot of the Android Studio code editor showing the `TodosAdapter.java` file. The code defines a RecyclerView adapter for a list of `Todos` objects. It includes imports for Context, LayoutInflater, View, ViewGroup, TextView, androidx.annotation.NonNull, Glide, and R. The adapter has a private list of `Todos` and a context. It overrides  `onCreateViewHolder` to inflate a layout item from the parent's context, and  `onBindViewHolder` to set text for five TextViews based on the `Todo` object's properties. The  `getItemCount` method returns the size of the `todos` list.

```
1 package com.tesji.biblioapitesji.adapter;
2
3 import android.content.Context;
4 import android.view.LayoutInflater;
5 import android.view.View;
6 import android.view.ViewGroup;
7 import android.widget.TextView;
8
9 import androidx.annotation.NonNull;
10 import androidx.recyclerview.widget.RecyclerView;
11
12 import com.bumptech.glide.Glide;
13 import com.tesji.biblioapitesji.R;
14 import com.tesji.biblioapitesji.model.Todos;
15
16 import java.util.List;
17
18 public class TodosAdapter extends RecyclerView.Adapter<TodosAdapter.ViewHolder> { 4 usages
19
20     private List<Todos> todos; 7 usages
21     private Context context; 1 usage
22
23     public TodosAdapter(List<Todos> todos, Context context) { 1 usage
24         this.todos = todos;
25         this.context = context;
26     }
27
28     @NonNull
29     @Override
30     public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
31         View view= LayoutInflater.from(parent.getContext())
32             .inflate(R.layout.item_libros,parent, attachToRoot: false);
33
34         return new ViewHolder(view);
35     }
36
37     @Override
38     public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
39         holder.tv_id.setText(todos.get(position).getId());
40         holder.tv_titulo.setText(todos.get(position).getTitulo());
41         holder.tv_autores.setText(todos.get(position).getAutores());
42         holder.tv_editorial.setText(todos.get(position).getEditorial());
43         holder.tv_edicion.setText(todos.get(position).getEdicion());
44     }
45
46     @Override
47     public int getItemCount() { return todos.size(); }
48
49
50     public class ViewHolder extends RecyclerView.ViewHolder { 4 usages
51         private TextView tv_id; 2 usages
52         private TextView tv_titulo; 2 usages
53         private TextView tv_autores; 2 usages
54         private TextView tv_editorial; 2 usages
55         private TextView tv_edicion; 2 usages
56
57         public ViewHolder(@NonNull View itemView) { 1 usage
58             super(itemView);
59             tv_id=itemView.findViewById(R.id.idlibro);
60             tv_titulo=itemView.findViewById(R.id.titulolibro);
61             tv_autores=itemView.findViewById(R.id.autoreslibro);
62             tv_editorial=itemView.findViewById(R.id.editoriallibro);
63             tv_edicion=itemView.findViewById(R.id.edicionlibro);
64         }
65     }
66 }
```

Screenshot of the Android Studio code editor showing the `Todos.java` file. This is a simple data model class with a constructor that takes a list of `String`s and initializes five private `String` fields: `id`, `titulo`, `autores`, `editorial`, and `edicion`. It also has getters for these fields.

```
1 package com.tesji.biblioapitesji.model;
2
3 import java.util.List;
4
5 public class Todos {
6     private String id;
7     private String titulo;
8     private String autores;
9     private String editorial;
10    private String edicion;
11
12    public Todos(List<String> todos) {
13        this.id=todos.get(0);
14        this.titulo=todos.get(1);
15        this.autores=todos.get(2);
16        this.editorial=todos.get(3);
17        this.edicion=todos.get(4);
18    }
19
20    public String getId() { return id; }
21
22    public String getTitulo() { return titulo; }
23
24    public String getAutores() { return autores; }
25
26    public String getEditorial() { return editorial; }
27
28    public String getEdicion() { return edicion; }
29 }
```

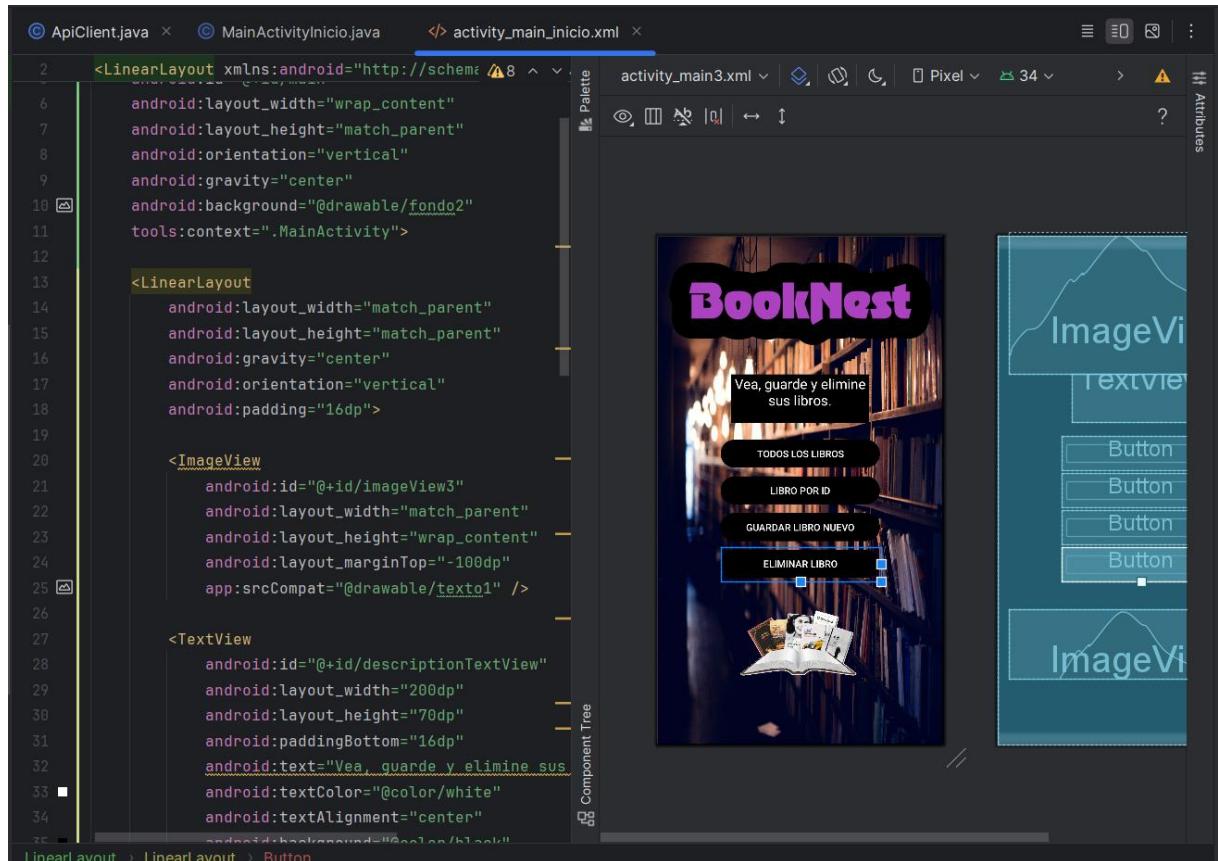
```

18     public class TodosAdapter extends RecyclerView.Adapter<TodosAdapter.ViewHolder> {
19
20         @Override
21         public void onBindViewHolder(@NotNull ViewHolder holder, int position) {
22             holder.tv_id.setText(todos.get(position).getId());
23             holder.tv_titulo.setText(todos.get(position).getTitulo());
24             holder.tv_autores.setText(todos.get(position).getAutores());
25             holder.tv_editorial.setText(todos.get(position).getEditorial());
26             holder.tv_edicion.setText(todos.get(position).getEdicion());
27
28         }
29
30         @Override
31         public int getItemCount() { return todos.size(); }
32
33     }
34
35     public class ViewHolder extends RecyclerView.ViewHolder {
36         private TextView tv_id; 2 usages
37         private TextView tv_titulo; 2 usages
38         private TextView tv_autores; 2 usages
39         private TextView tv_editorial; 2 usages
40         private TextView tv_edicion; 2 usages
41
42         public ViewHolder(@NotNull View itemView) { 1 usage
43             super(itemView);
44             tv_id=itemView.findViewById(R.id.idlibro);
45             tv_titulo=itemView.findViewById(R.id.titulolibro);
46             tv_autores=itemView.findViewById(R.id.autoreslibro);
47             tv_editorial=itemView.findViewById(R.id.editoriallibro);
48             tv_edicion=itemView.findViewById(R.id.edicionlibro);
49
50         }
51     }
52 }

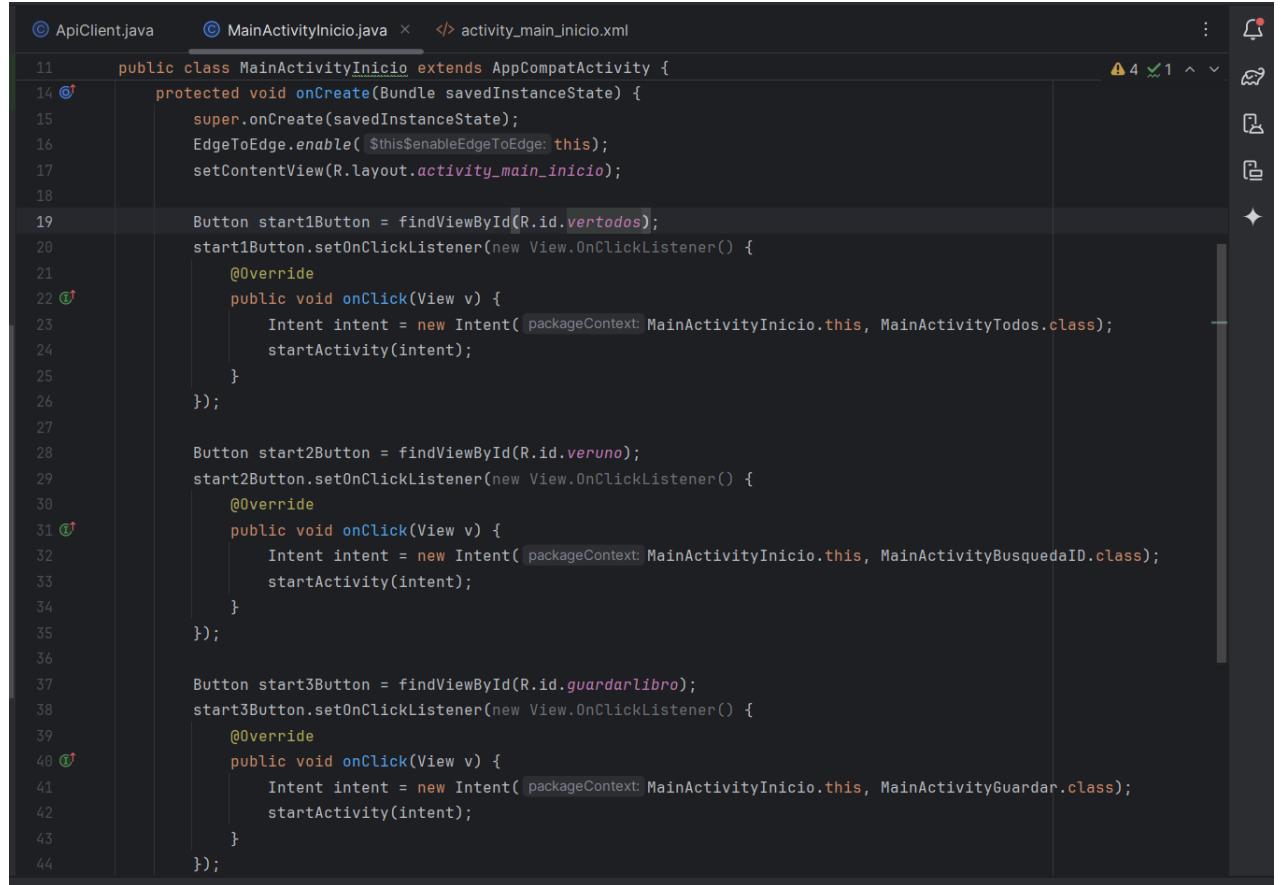
```

- Ahora comenzaremos con la creación de la pantalla en donde podremos hacer nuestras peticiones a la API.

Comenzamos con el diseño:



Posteriormente con su funcionamiento, lo que hará cada botón al hacer clic es redireccionar a otra pagina para poder lo que el usuario desee.

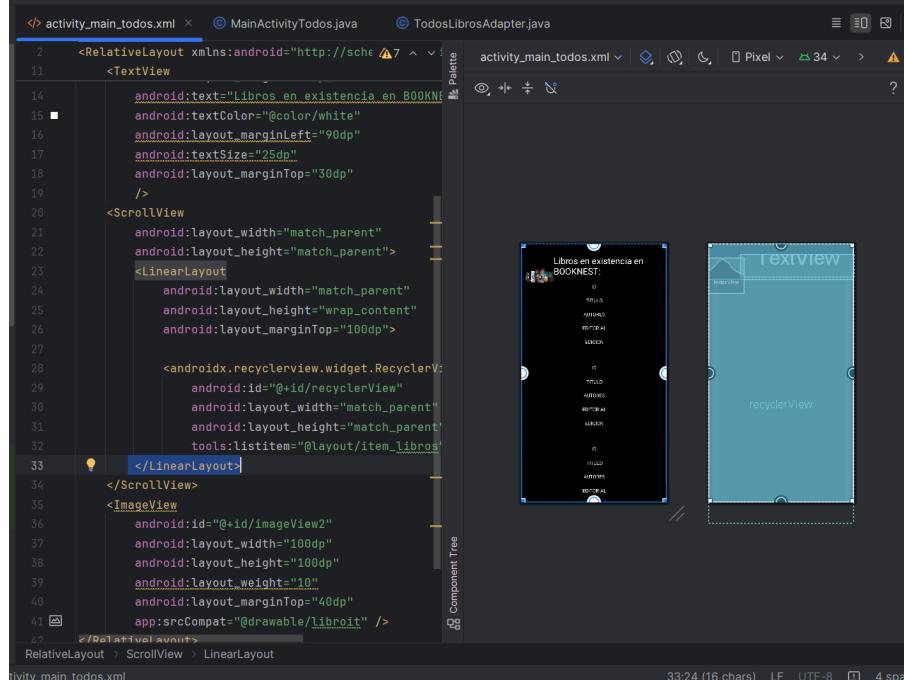


```
ApiClient.java>MainActivityInicio.java>activity_main_inicio.xml
11  public class MainActivityInicio extends AppCompatActivity {
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         EdgeToEdge.enable( $this$enableEdgeToEdge: this);
18         setContentView(R.layout.activity_main_inicio);
19
20         Button start1Button = findViewById(R.id.vertodos);
21         start1Button.setOnClickListener(new View.OnClickListener() {
22             @Override
23             public void onClick(View v) {
24                 Intent intent = new Intent( packageContext: MainActivityInicio.this, MainActivityTodos.class);
25                 startActivity(intent);
26             }
27         });
28
29         Button start2Button = findViewById(R.id.veruno);
30         start2Button.setOnClickListener(new View.OnClickListener() {
31             @Override
32             public void onClick(View v) {
33                 Intent intent = new Intent( packageContext: MainActivityInicio.this, MainActivityBusquedaID.class);
34                 startActivity(intent);
35             }
36         });
37
38         Button start3Button = findViewById(R.id.guardarlibro);
39         start3Button.setOnClickListener(new View.OnClickListener() {
40             @Override
41             public void onClick(View v) {
42                 Intent intent = new Intent( packageContext: MainActivityInicio.this, MainActivityGuardar.class);
43             }
44         });

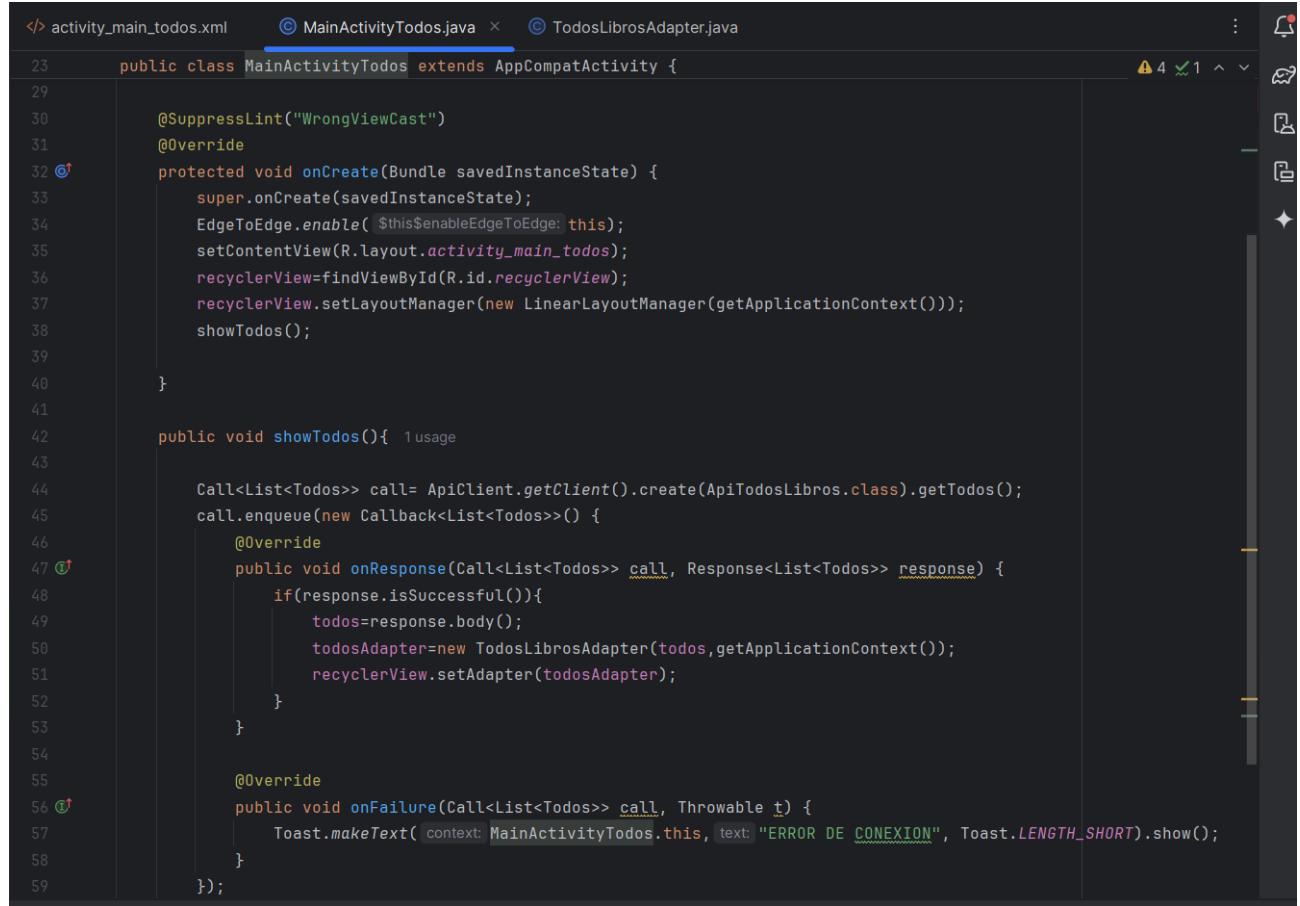
```

- Comenzaremos con el botón de ver todos los libros, en este lo que hace es que me muestra todos los libros que hay en mi base de datos con la vinculación que se hizo por medio de la URL que se creó en la API.

Diseño del activity\_main\_inicio;



## Funcionalidad:

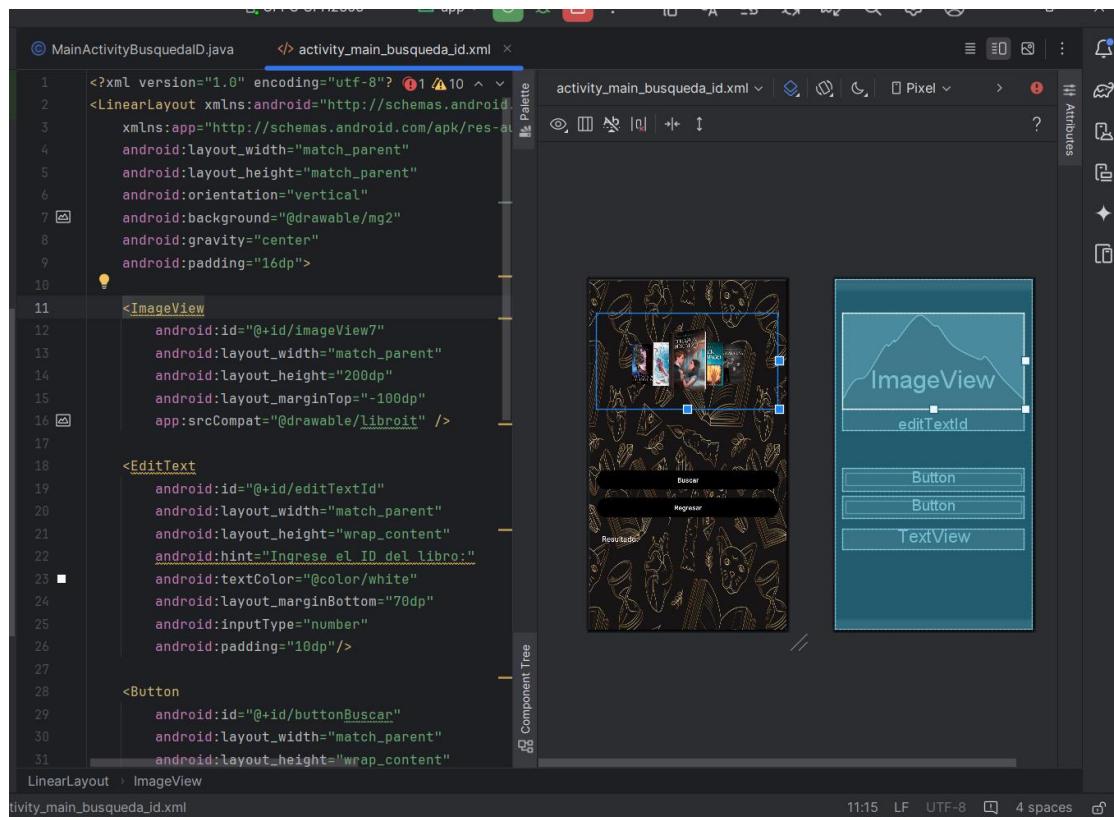


```
activity_main_todos.xml MainActivityTodos.java TodosLibrosAdapter.java
23     public class MainActivityTodos extends AppCompatActivity {
29
30         @SuppressLint("WrongViewCast")
31         @Override
32         protected void onCreate(Bundle savedInstanceState) {
33             super.onCreate(savedInstanceState);
34             EdgeToEdge.enable(this);
35             setContentView(R.layout.activity_main_todos);
36             recyclerView=findViewById(R.id.recyclerView);
37             recyclerView.setLayoutManager(new LinearLayoutManager(getApplicationContext()));
38             showTodos();
39         }
40
41         public void showTodos(){ 1 usage
43
44             Call<List<Todos>> call= ApiClient.getClient().create(ApiTodosLibros.class).getTodos();
45             call.enqueue(new Callback<List<Todos>>() {
46                 @Override
47                 public void onResponse(Call<List<Todos>> call, Response<List<Todos>> response) {
48                     if(response.isSuccessful()){
49                         todos=response.body();
50                         todosAdapter=new TodosLibrosAdapter(todos,getApplication());
51                         recyclerView.setAdapter(todosAdapter);
52                     }
53                 }
54
55                 @Override
56                 public void onFailure(Call<List<Todos>> call, Throwable t) {
57                     Toast.makeText(context, MainActivityTodos.this, "ERROR DE CONEXION", Toast.LENGTH_SHORT).show();
58                 }
59             });
}

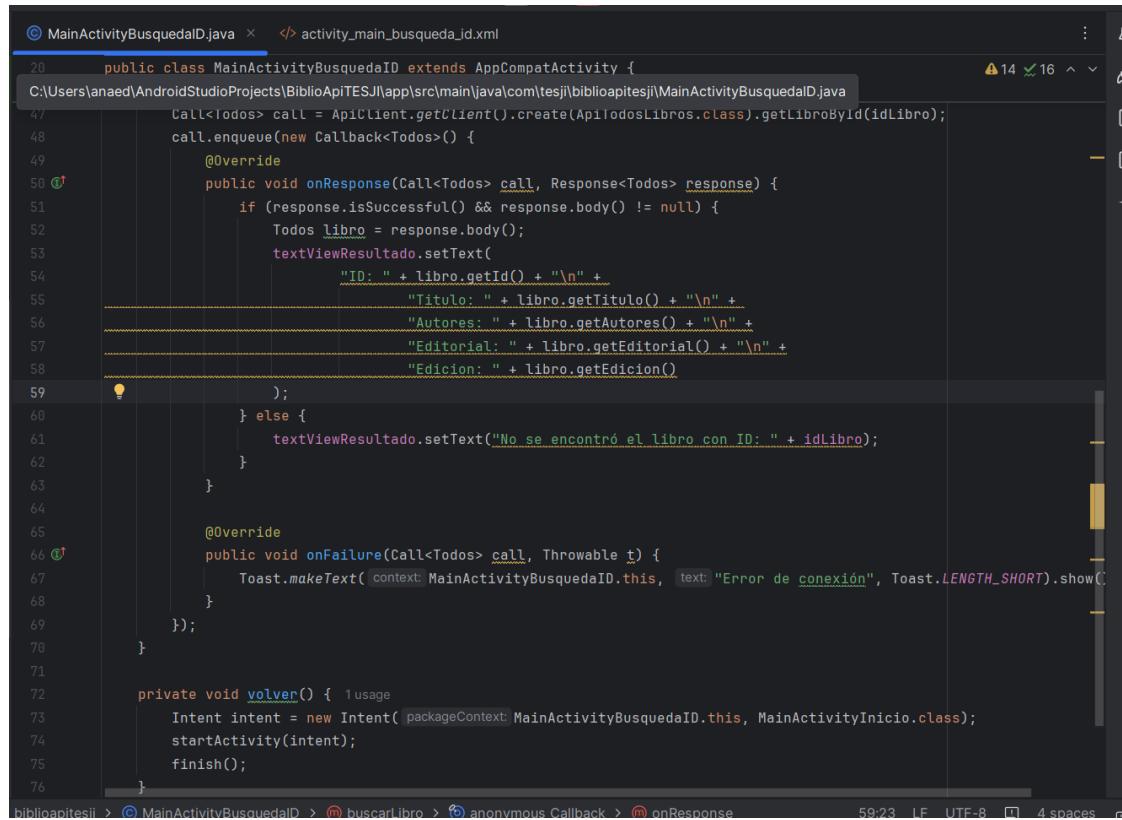
```

- Ahora crearemos la búsqueda por id

Diseño:



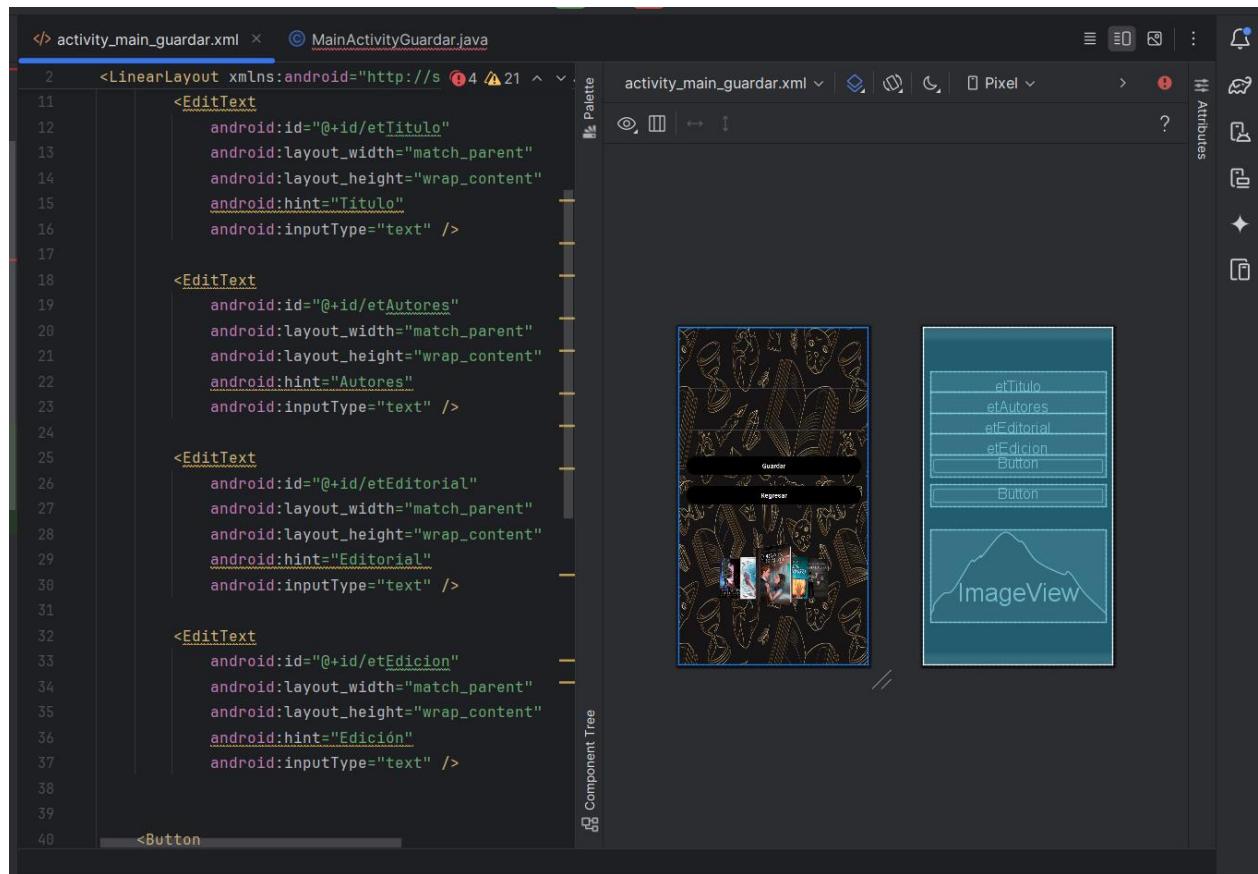
## Funcionalidad:



```
20     public class MainActivityBusquedaID extends AppCompatActivity {
21
22         Call<Todos> call = ApiClient.getClient().create(ApiLibros.class).getLibroById(idLibro);
23
24         call.enqueue(new Callback<Todos>() {
25             @Override
26             public void onResponse(Call<Todos> call, Response<Todos> response) {
27                 if (response.isSuccessful() && response.body() != null) {
28                     Todos libro = response.body();
29                     textViewResultado.setText(
30                         "ID: " + libro.getId() + "\n" +
31                         "Título: " + libro.getTitulo() + "\n" +
32                         "Autores: " + libro.getAutores() + "\n" +
33                         "Editorial: " + libro.getEditorial() + "\n" +
34                         "Edición: " + libro.getEdicion()
35
36                 );
37             } else {
38                 textViewResultado.setText("No se encontró el libro con ID: " + idLibro);
39             }
40         }
41
42         @Override
43         public void onFailure(Call<Todos> call, Throwable t) {
44             Toast.makeText(context, MainActivityBusquedaID.this, text: "Error de conexión", Toast.LENGTH_SHORT).show();
45         }
46     });
47
48     private void volver() { 1 usage
49         Intent intent = new Intent(packageContext: MainActivityBusquedaID.this, MainActivityInicio.class);
50         startActivity(intent);
51         finish();
52     }
53 }
```

- Crearemos el de guardar libro

### Diseño:

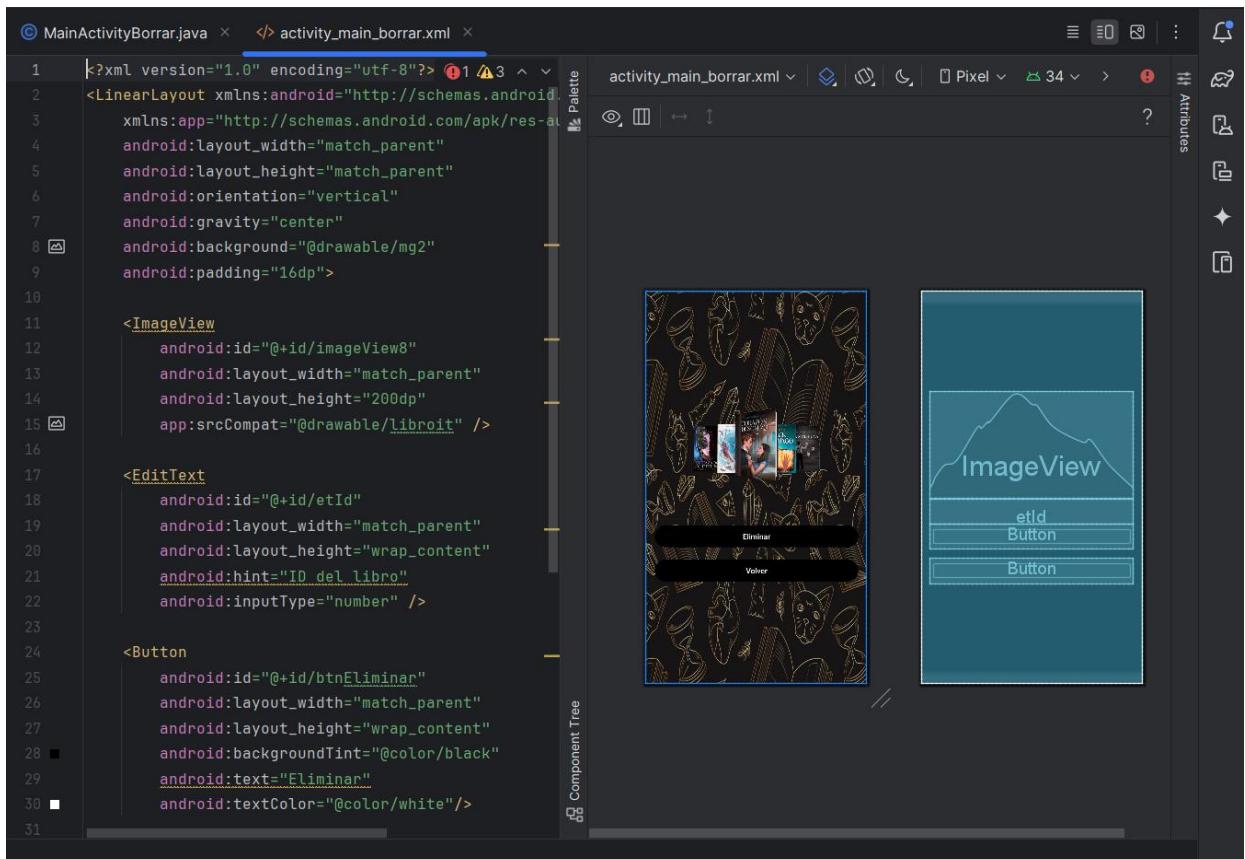


## Funcionalidad:

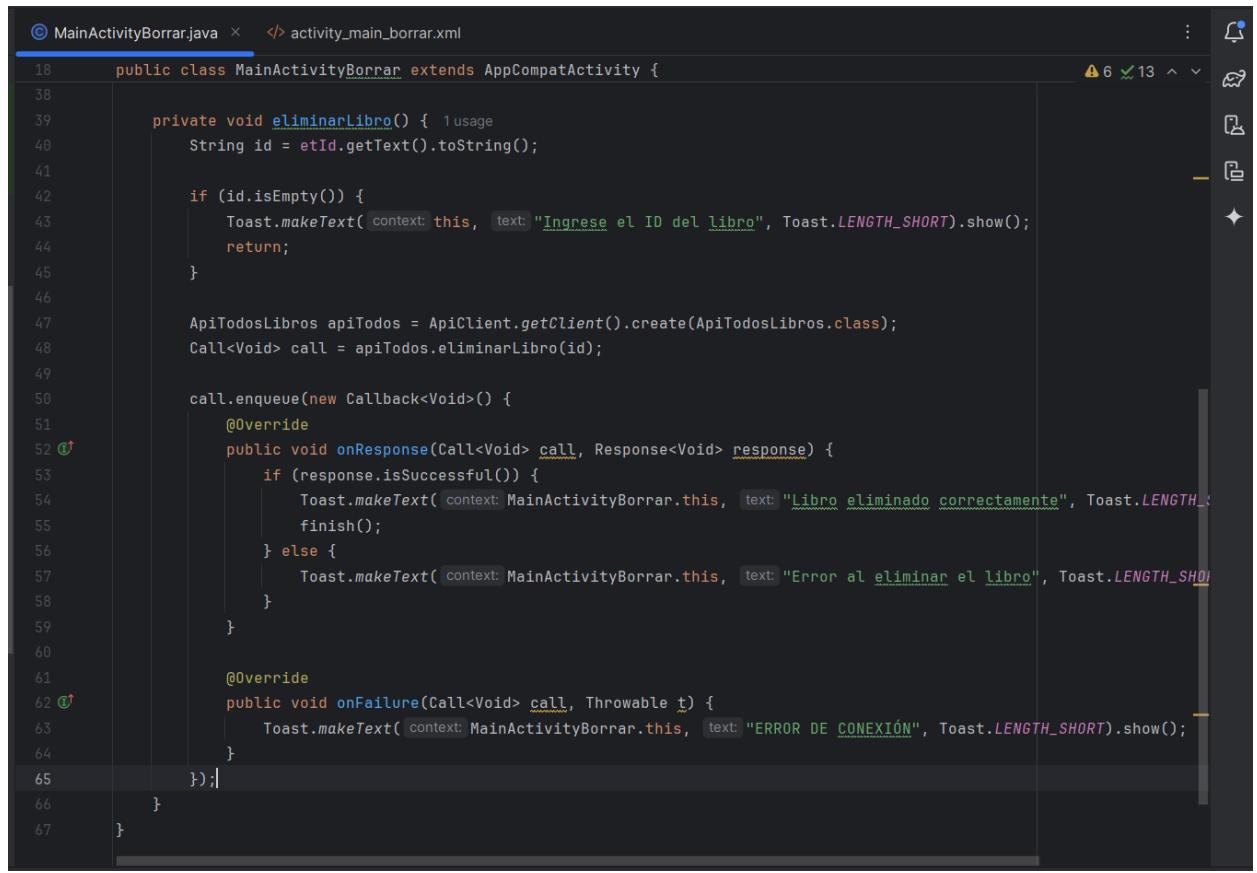
```
</> activity_main_guardar.xml  MainActivityGuardar.java x
19     public class MainActivityGuardar extends AppCompatActivity {
43         private void guardarLibro() {  1 usage
51             libro.setAutores(autores);
52             libro.setEditorial(editorial);
53             libro.setEdicion(edicion);
54
55             ApiTodosLibros apiTodos = ApiClient.getClient().create(ApiTodosLibros.class);
56             Call<Void> call = apiTodos.guardarLibro(libro);
57
58             call.enqueue(new Callback<Void>() {
59                 @Override
60                 public void onResponse(Call<Void> call, Response<Void> response) {
61                     if (response.isSuccessful()) {
62                         Toast.makeText(context, "Libro guardado correctamente", Toast.LENGTH_SHORT).show();
63                     } else {
64                         Toast.makeText(context, "Error al guardar el libro", Toast.LENGTH_SHORT).show();
65                     }
66                 }
67
68                 @Override
69                 public void onFailure(Call<Void> call, Throwable t) {
70                     Toast.makeText(context, "Error de conexión", Toast.LENGTH_SHORT).show();
71                 }
72             });
73         }
74     }
75
```

- Crearemos el de guardar libro

Diseño:



Funcionalidad:



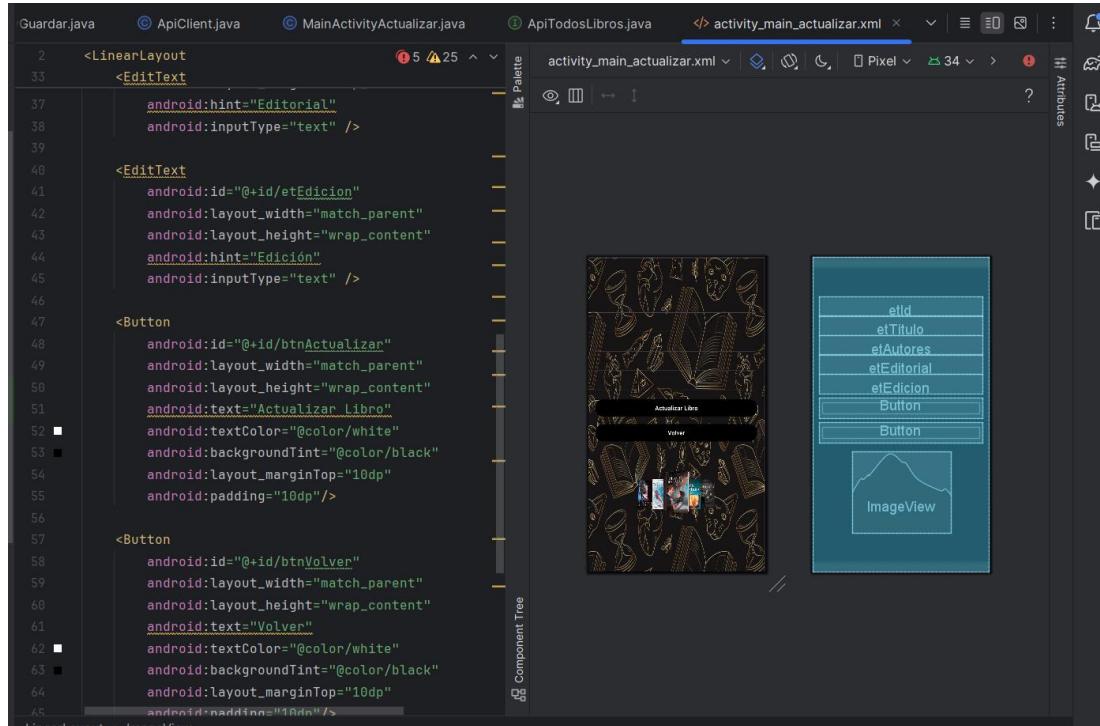
```

18     public class MainActivityBorrar extends AppCompatActivity {
38
39         private void eliminarLibro() { 1 usage
40             String id = etId.getText().toString();
41
42             if (id.isEmpty()) {
43                 Toast.makeText(context: this, text: "Ingrese el ID del libro", Toast.LENGTH_SHORT).show();
44                 return;
45             }
46
47             ApiTodosLibros apiTodos = ApiClient.getClient().create(ApiTodosLibros.class);
48             Call<Void> call = apiTodos.eliminarLibro(id);
49
50             call.enqueue(new Callback<Void>() {
51                 @Override
52                 public void onResponse(Call<Void> call, Response<Void> response) {
53                     if (response.isSuccessful()) {
54                         Toast.makeText(context: MainActivityBorrar.this, text: "Libro eliminado correctamente", Toast.LENGTH_SHORT).show();
55                         finish();
56                     } else {
57                         Toast.makeText(context: MainActivityBorrar.this, text: "Error al eliminar el libro", Toast.LENGTH_SHORT).show();
58                     }
59                 }
60
61                 @Override
62                 public void onFailure(Call<Void> call, Throwable t) {
63                     Toast.makeText(context: MainActivityBorrar.this, text: "ERROR DE CONEXIÓN", Toast.LENGTH_SHORT).show();
64                 }
65             });
66         }
67     }

```

- Crearemos el de editar libro

Diseño:



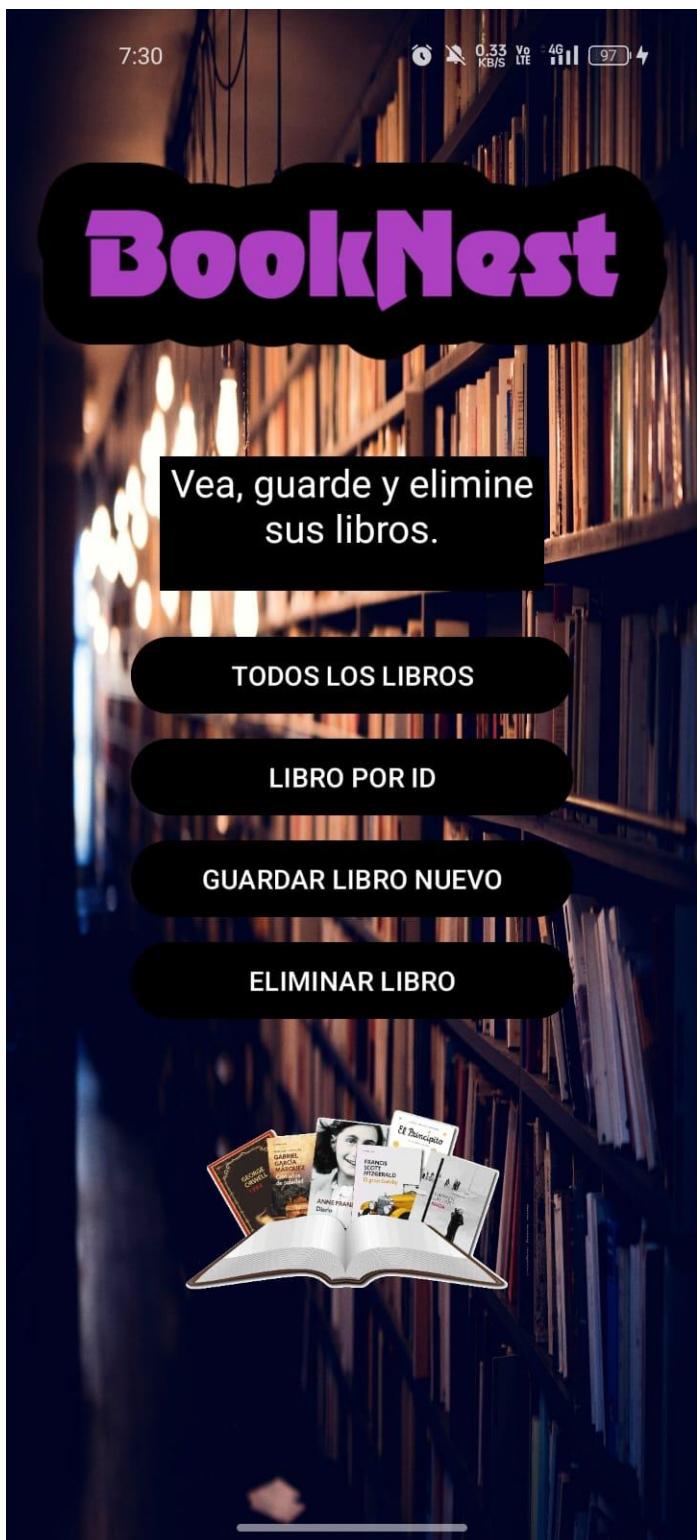
Funcionalidad:

The screenshot shows the Android Studio interface with the code editor open to the file `MainActivityActualizar.java`. The code implements a logic to update a book in a database using an API. It includes methods for setting up a `Todos` object with book details, creating a `Call<Void>` to the `guardarLibro` endpoint, and handling the response or failure of the call.

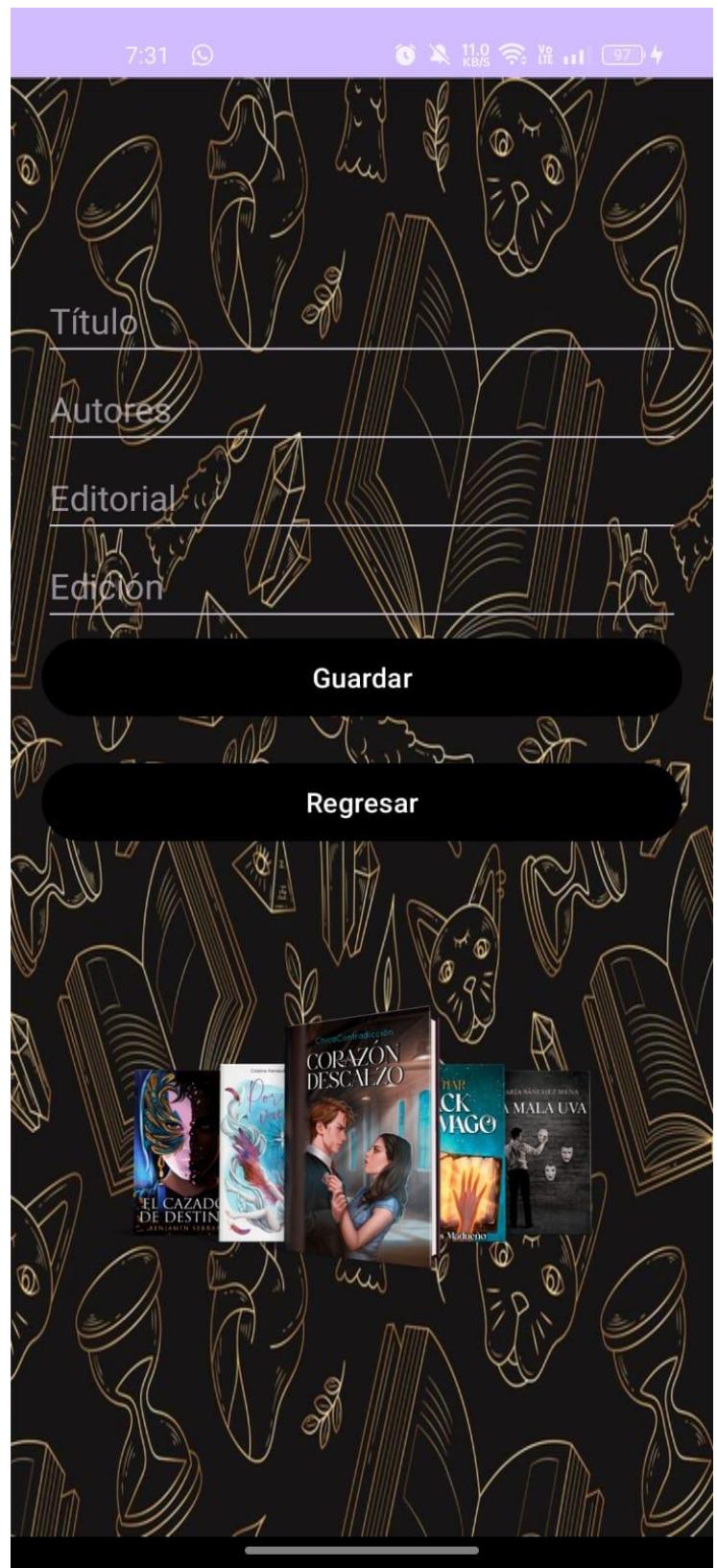
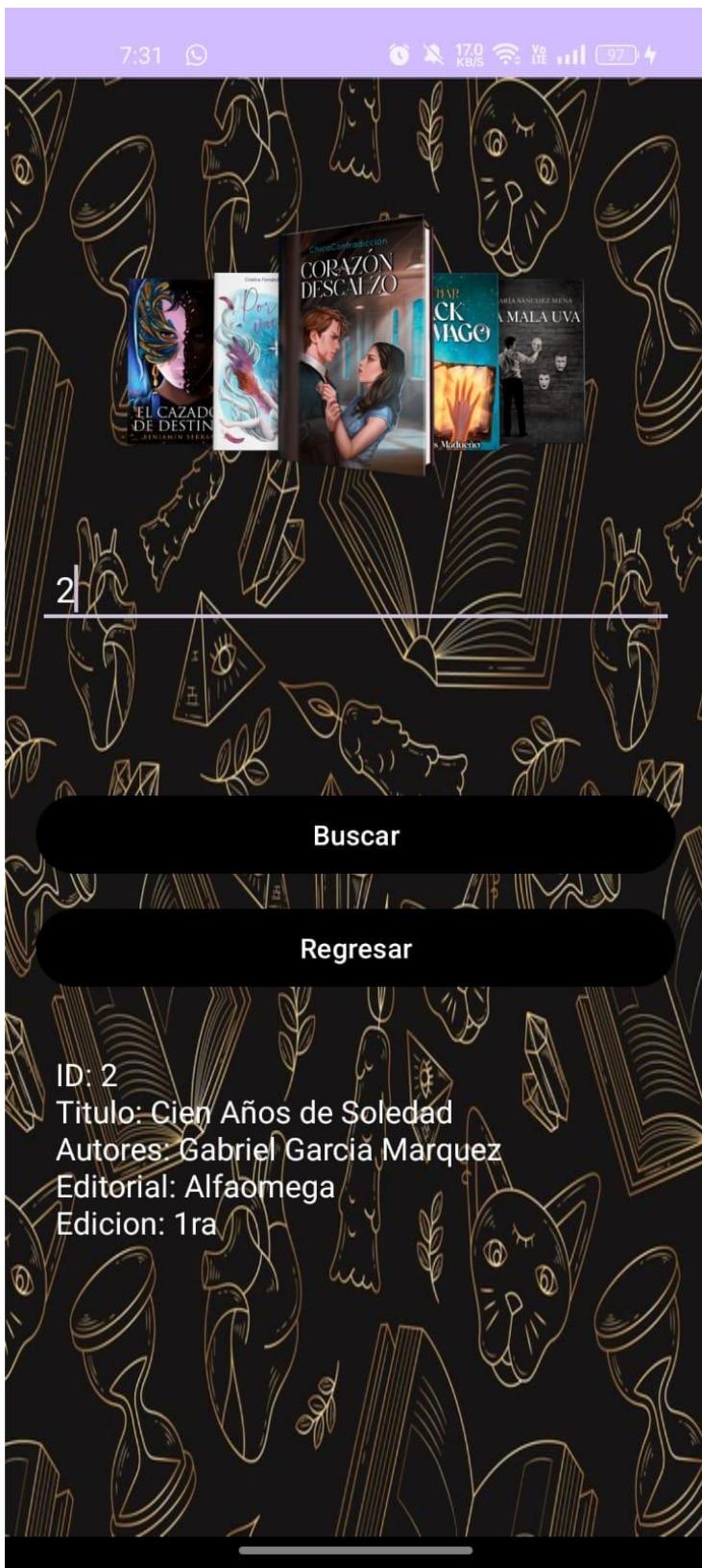
```
20     public class MainActivityActualizar extends AppCompatActivity {
53         private void actualizarLibro() { 1usage
65             Todos libro = new Todos();
66             libro.setId(String.valueOf(id)); // Establece el ID del libro
67             libro.setTitulo(titulo);
68             libro.setAutores(autor);
69             libro.setEditorial(editor);
70             libro.setEdicion(edicion);
71
72             ApiTodosLibros apiTodos = ApiClient.getClient().create(ApiTodosLibros.class);
73
74             Call<Void> call = apiTodos.guardarLibro(libro); // Llama al método guardarLibro
75             call.enqueue(new Callback<Void>() {
76                 @Override
77                 public void onResponse(Call<Void> call, Response<Void> response) {
78                     if (response.isSuccessful()) {
79                         Toast.makeText(context, MainActivityActualizar.this, "Libro editado exitosamente", Toast.LENGTH_SHORT).show();
80                     } else {
81                         Log.e("Edit", "Error al editar libro: " + response.message());
82                         Toast.makeText(context, MainActivityActualizar.this, "Error al editar libro: " + response.message(), Toast.LENGTH_SHORT).show();
83                     }
84                 }
85
86                 @Override
87                 public void onFailure(Call<Void> call, Throwable t) {
88                     Log.e("Edit", "Error en la llamada: " + t.getMessage());
89                     Toast.makeText(context, MainActivityActualizar.this, "Error en la llamada: " + t.getMessage(), Toast.LENGTH_SHORT).show();
90                 }
91             });
92         }
93     }
```

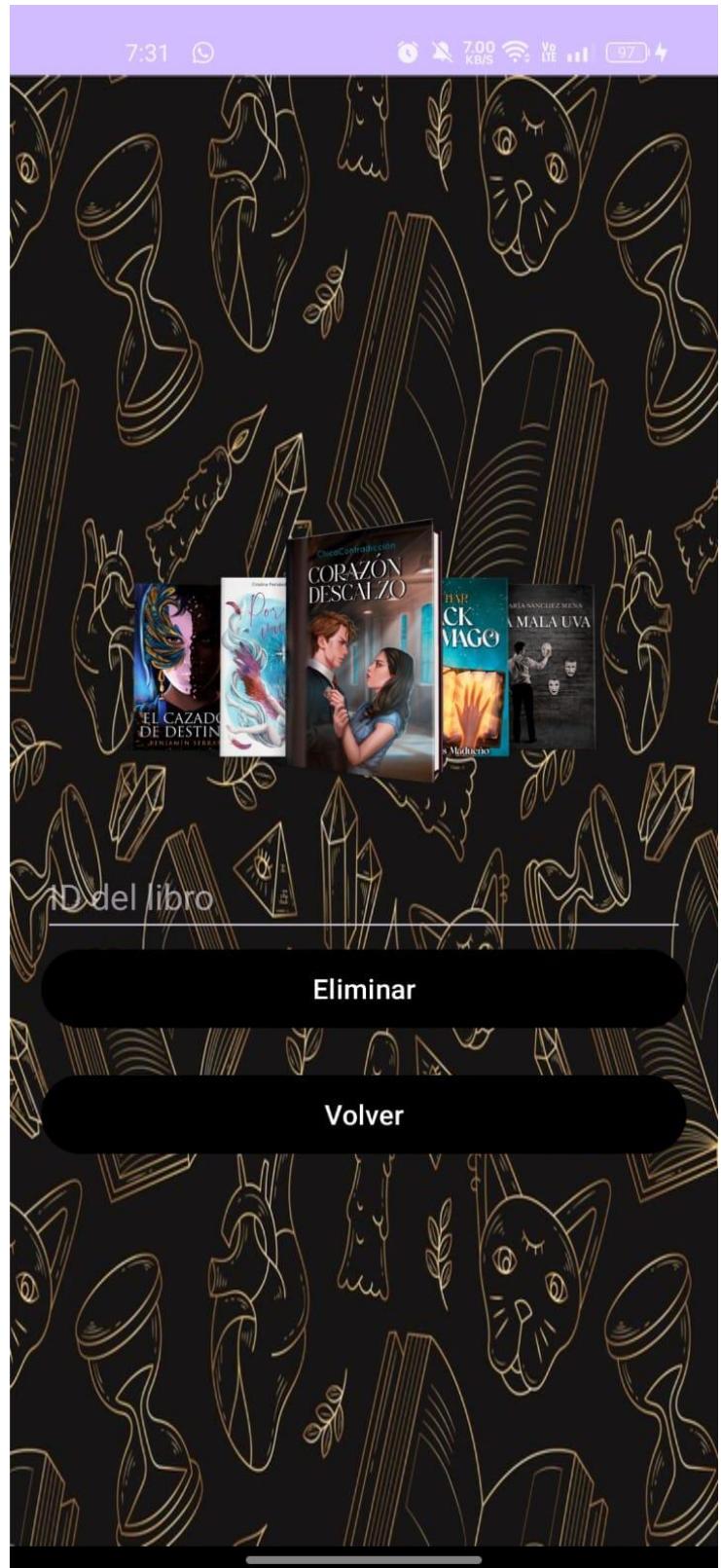
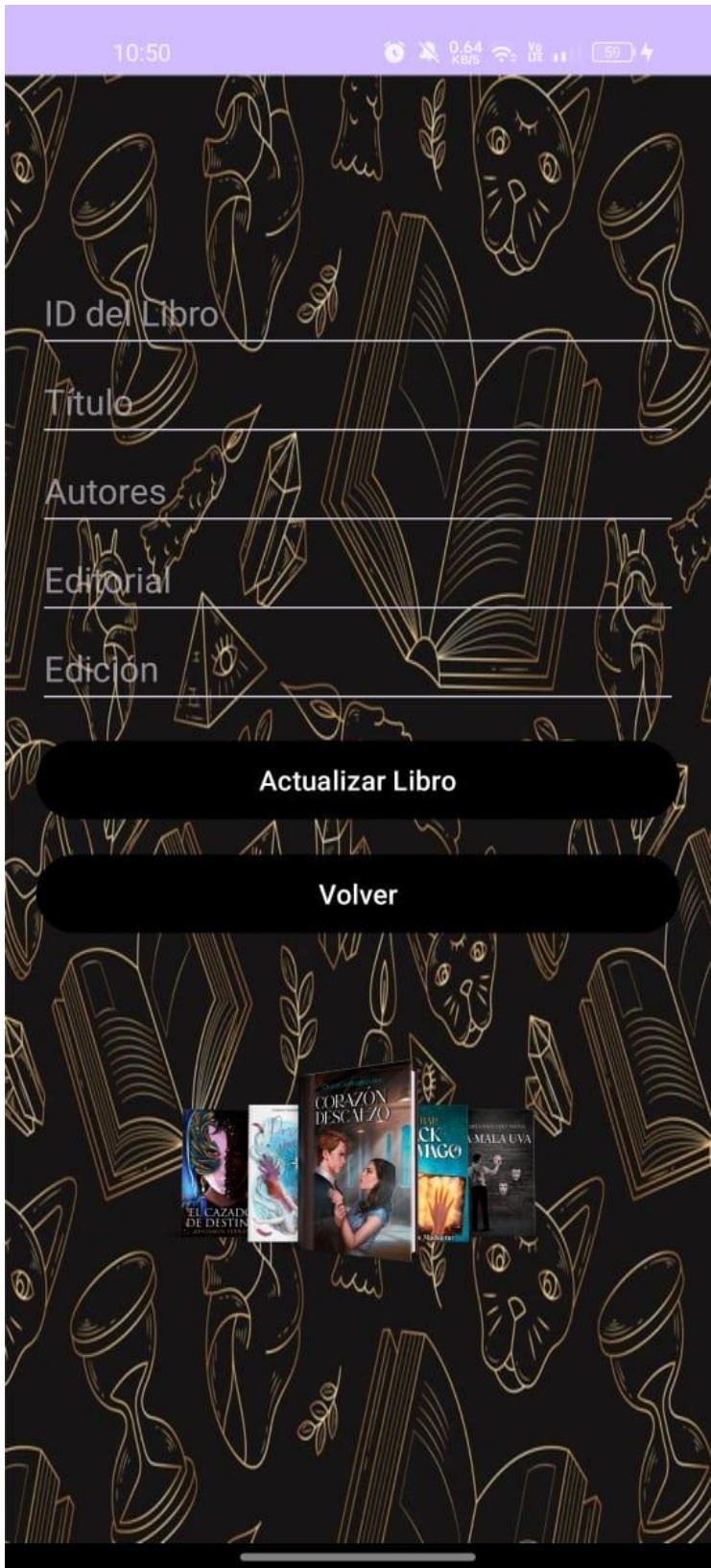
- Vista de previa de la aplicación.





- 7:30
- 0.33 KB/S 4G 97%
- Libros en existencia en BOOKNEST:
- 1
- EL PERFUME, HISTORIA DE UN ASESINO  
PATRICK SUSKIND
- 2
- CIEN AÑOS DE SOLEDAD  
GABRIEL GARCIA MARQUEZ  
ALFAOMEGA
- 3
- LOS 10 AÑOS QUE MAS TE AME  
KARLA JM  
WATTPAD
- 1RA





## **CONCLUSION**

El proyecto de aplicación de biblioteca desarrollado en Android Studio implementa de manera efectiva las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) para la gestión de libros, permitiendo a los usuarios visualizar todos los libros disponibles, buscar libros por ID, agregar nuevos libros mediante un formulario y eliminar libros no deseados o desactualizados por su ID. Utilizando tecnologías como Retrofit, Gson y Material Design, la aplicación ofrece una interfaz de usuario intuitiva y moderna que facilita la administración eficiente de una colección de libros. Este proyecto no solo digitaliza y simplifica la gestión bibliográfica, sino que también sirve como una valiosa herramienta educativa para entender la implementación de funcionalidades CRUD en aplicaciones móviles, proporcionando una base sólida para futuras ampliaciones y mejoras.