Proyecto Integrador

Descripción del Proyecto: Se trata de realizar una aplicación móvil sobre una red social de turismo en la que los usuarios postean fotos de los lugares que han visitado y otros datos para compartir con otros usuarios de la red.

La configuración del Proyecto:

A- Propósito del archivo Manifest: Especifica los componentes y permisos.

1. **Declara componentes de la aplicación**:
    - **Activities**: Las pantallas de la app.
    - **Services**: Tareas que se ejecutan en segundo plano.
    - **Broadcast Receivers**: Responden a eventos globales del sistema o de la app.
    - **Content Providers**: Comparten datos entre aplicaciones.
2. **Específica permisos**: Define qué recursos del dispositivo requiere la app, como acceso a la cámara, internet o ubicación.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

3. **Configura el punto de entrada**: Define cuál es la **Activity principal** que se lanza cuando se abre la app.

```
<application
    android:name=".view.MyApplication"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_ppal"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_ppal"
    android:supportsRtl="true"
    android:theme="@style/Theme.AppChat"
    tools:targetApi="31">
```

4. **Define características y compatibilidad**

```
<uses-sdk android:minSdkVersion="24" />
```

5. **Configuraciones globales**: Establece configuraciones de la app, como configuraciones de servicio, tema, íconos, y nombres.

```
<meta-data
          android:name="com.parse.SERVER_URL"
          android:value="@string/back4app_server_url" />
       <meta-data
          android:name="com.parse.APPLICATION_ID"
          android:value="@string/back4app_app_id" />
       <meta-data
          android:name="com.parse.CLIENT_KEY"
          android:value="@string/back4app_client_key" />
```

Ejemplo de mis configuraciones en AndroidManifest.xml

```
<?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" />
    <uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
    <application
        android:name=".view.MyApplication"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_ppal"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_ppal"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppChat"
        tools:targetApi="31">
        <activity
            android:name=".view.PostActivity"
            android:exported="false" />
        <activity
            android:name=".view.HomeActivity"
            android:exported="false" />
        <activity
            android:name=".view.RegisterActivity"
            android:exported="false" />
        <activity
            android:name=".view.MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <meta-data
            android:name="com.parse.SERVER_URL"
            android:value="@string/back4app_server_url" />
        <meta-data
            android:name="com.parse.APPLICATION_ID"
            android:value="@string/back4app_app_id" />
        <meta-data
            android:name="com.parse.CLIENT_KEY"
            android:value="@string/back4app_client_key" />

    </application>
    <uses-sdk android:minSdkVersion="24" />
```

```
</manifest>
```

B- **Gradle**
Hay dos. El build.gradle a nivel de módulo (app):
Ubicado en `app/build.gradle`. Define configuraciones específicas de la aplicación, como:

- ID del paquete.
- Versión del SDK.
- Dependencias.
- Configuraciones de build.

Ejemplo de mi archivo build.gradle.kts (app)

```
plugins {
    alias(libs.plugins.android.application)
}
android {
    namespace = "com.example.appchat"
    compileSdk = 35

    defaultConfig {
        applicationId = "com.example.appchat"
        minSdk = 24
        targetSdk = 35
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner =
"androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(

getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }

    buildFeatures {
```

```
        buildConfig = true
        viewBinding = true
    }

}

dependencies {

    implementation("androidx.appcompat:appcompat:1.7.0")
    implementation("com.google.android.material:material:1.12.0")
    implementation("androidx.activity:activity-ktx:1.7.2")
    implementation("androidx.activity:activity:1.7.2")
    implementation("androidx.fragment:fragment-ktx:1.6.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    implementation("androidx.cardview:cardview:1.0.0")
    implementation
("androidx.coordinatorlayout:coordinatorlayout:1.2.0")
    implementation("androidx.recyclerview:recyclerview:1.3.0")

    implementation ("androidx.lifecycle:lifecycle-viewmodel:2.8.7")
// Verifica la última versión
    implementation ("androidx.lifecycle:lifecycle-livedata:2.8.7")
    implementation(libs.core)

    // Dependencias de testing
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")

androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1"
)

    // picasso para manejar imágenes
    implementation ("com.squareup.picasso:picasso:2.71828")

    //circle image view
    implementation ("de.hdodenhof:circleimageview:3.1.0")


    // ShapeOfView
    implementation("io.github.florent37:shapeofview:1.4.7")

    // Parse SDK

implementation("com.github.parse-community.Parse-SDK-Android:bolts-ta
sks:4.3.0")

implementation("com.github.parse-community.Parse-SDK-Android:parse:4.
3.0")

    // For building media UIs
    implementation ("androidx.media3:media3-ui:1.4.1")

    // For building media playback UIs for Android TV using the
Jetpack Leanback library
    implementation ("androidx.media3:media3-ui-leanback:1.4.1")
```

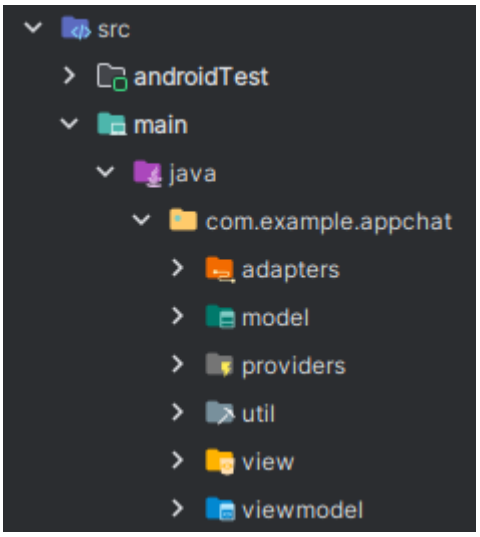
```
}
```

build.gradle a nivel de proyecto: Ubicado en el directorio raíz del proyecto. Este archivo configura aspectos globales, como los repositorios y versiones de herramientas utilizadas por todas los submódulos.

Este es el mio por ahora:

```
plugins {
    alias(libs.plugins.android.application) apply false
}
```

Paquetes que contienen clases java del proyecto



Responsabilidades de cada paquete

- adapters: Contiene clases responsables de conectar y transformar datos para ser visualizados en componentes de interfaz gráfica, en nuestro proyecto los RecyclerView que contiene el fragmento

- HomeFragment
- PostActivity
- FiltrosFragment.

Podría incluir por ejemplo un adaptador personalizado para un Spinner.

- model:Define las estructuras de datos y entidades del dominio de la aplicación. Contiene clases que representan la lógica de negocio, los objetos del modelo y los objetos persistentes. Interactúa con las fuentes de datos, como bases de datos. En este paquete encontramos las clases:

- User
- Post.

- providers: Encapsula la lógica para acceder a fuentes de datos externas, como servicios web o bases de datos. Actúa como puente entre los repositorios y las fuentes de datos.
Incluimos en este paquete

- AuthProvider
- PostProvider.
-

- util:Contiene clases auxiliar

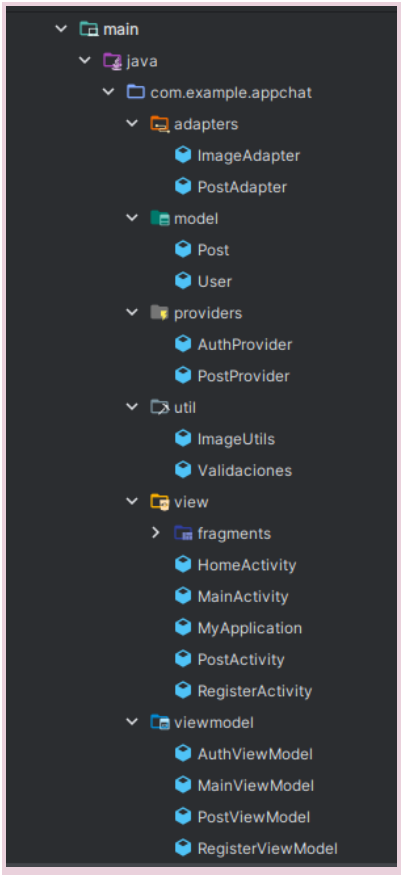
- validaciones
- ImageUtils

- view: Contiene todas las clases relacionadas con la interfaz de usuario (UI), como Activities, Fragments. Maneja la interacción del usuario.

- MyAplication
- MainActivity
- RegisterActivity
- HomeActivity
- PostActivity
- Paquete **Fragments**:
  - HomeFragment
  - PerfilFragment
  - ChatsFragment
  - PerfilFragment

- viewmodel: Maneja la lógica de presentación y expone datos a las vistas en forma de LiveData u observables.
Actúa como intermediario entre el modelo (datos) y la vista (UI).
No conoce los detalles de la vista ni depende de ella.

- MainViewModel
- AuthViewModel
- PostViewModel
- RegisterViewModel



Descripción de la responsabilidad de cada clase, atributos, métodos y relaciones con otras clases UML

1. En el paquete view debes crear la clase que extiende de `Application, ésta` es una clase global que se utiliza para mantener el estado global de la aplicación y no debería estar relacionada directamente con la UI o el ciclo de vida de una actividad.

MyApplication (inherits Application)
+ onCreate(): void

```
public class MyApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        // Enable Local Datastore.
        Parse.enableLocalDatastore(this);

        Parse.initialize(new Parse.Configuration.Builder(this)
                .applicationId(getString(R.string.back4app_app_id))
                .clientKey(getString(R.string.back4app_client_key))
                .server(getString(R.string.back4app_server_url))
```

```
                .build()
        );
        ParseACL defaultACL = new ParseACL();
        defaultACL.setPublicReadAccess(true);
        defaultACL.setPublicWriteAccess(true);
        ParseACL.setDefaultACL(defaultACL, true);
    }
}
```

2. MainActivity.java

MainActivity
- binding: ActivityMainBinding - viewModel: MainViewModel
+ onCreate(savedInstanceState: Bundle): void + onStart(): void + onResume(): void + manejarEventos(): void + showToast(message: String): void + limpiarCampos(): void - obtenerTextoSeguro(editText: EditText): String

Responsabilidad: manejar la actividad principal de la aplicación

3. MainViewModel.java

MainViewModel

- authProvider: AuthProvider
+ MainViewModel() + login(email: String, password: String): LiveData<String>

Responsabilidad: es responsable de la gestión de datos y la comunicación con el modelo relacionado con el manejo del estado de usuario o autenticación.

4.

AuthProvider
+ AuthProvider() + signIn(email: String, password: String): LiveData<String> + signUp(user: User):LiveData<String> + getCurrentUserID(): LiveData<String> + logout(): LiveData<Boolean>

Responsabilidad: maneja la autenticación y el registro de usuarios utilizando Parse como backend. Actúa como un intermediario entre las operaciones de autenticación (login, logout, registro, etc.) y las capas superiores del sistema, encapsulando la lógica relacionada con Parse.

```
public class AuthProvider {
    public AuthProvider() {}
    public LiveData<String> signIn(String email, String password) {
        MutableLiveData<String> authResult = new MutableLiveData<>();
        ParseUser.logInInBackground(email, password, (user, e) -> {
            if (e == null) {                // Login exitoso
                authResult.setValue(user.getObjectId());
                Log.d("AuthProvider", "Usuario autenticado exitosamente: " +
user.getObjectId());
            } else {
                Log.e("AuthProvider", "Error en inicio de sesión: ", e);
                authResult.setValue(null);}
        });
        return authResult;
    }
    // Registro con Parse
    public LiveData<String> signUp(User user) {
        MutableLiveData<String> authResult = new MutableLiveData<>();
        ParseUser parseUser = new ParseUser();
        parseUser.setUsername(user.getUsername());
        parseUser.setPassword(user.getPassword());
        parseUser.setEmail(user.getEmail());
```

```
        parseUser.signUpInBackground(e -> {
            if (e == null) {
                authResult.setValue(parseUser.getObjectId());
                Log.d("AuthProvider", "Usuario registrado exitosamente: " +
parseUser.getObjectId());
            } else {
                Log.e("AuthProvider", "Error en registro: ", e);
                authResult.setValue(null); }
          });
        return authResult;
    }
    public LiveData<String> getCurrentUserID() {
        MutableLiveData<String> currentUserId = new MutableLiveData<>();
        ParseUser currentUser = ParseUser.getCurrentUser();
        if (currentUser != null) {
            currentUserId.setValue(currentUser.getObjectId());
        }
        return currentUserId;
    }
    public LiveData<Boolean> logout() {
        MutableLiveData<Boolean> logoutResult = new MutableLiveData<>();
        ParseUser.logOutInBackground(e -> {
            if (e == null) {
                logoutResult.setValue(true);
                Log.d("AuthProvider", "Caché eliminada y usuario desconectado.");

            } else {
                logoutResult.setValue(false);
                Log.e("AuthProvider", "Error al desconectar al usuario: ", e);
            }
        });
        return logoutResult; }
}
```

AuthViewModel
- authProvider: AuthProvider
+ AuthViewModel() + logout(): LiveData<Boolean>

5. RegisterActivity.java

RegisterActivity
- binding: ActivityRegisterBinding - viewModel: RegisterViewModel

+ onCreate(savedInstanceState: Bundle): void + manejarEventos(): void + realizarRegistro(): void + showToast(message: String): void

Responsabilidad: Administra la interacción del usuario mediante la captura de eventos, valida las entradas, y delega la lógica de negocio al RegisterViewModel. Además, observa los resultados del registro para mostrar mensajes de retroalimentación que informan al usuario sobre el éxito o los errores en el proceso.

RegisterViewModel
- registerResult: MutableLiveData<String> - authProvider: AuthProvider
+ RegisterViewModel() + getRegisterResult(): LiveData<String> + register(user: User): void

Responsabilidad: sirve como el intermediario entre la capa de vista (RegisterActivity) y la capa de datos (AuthProvider) en el proceso de registro de usuarios. Gestiona la lógica del negocio relacionada con el registro, interactúa con el proveedor de autenticación para realizar la operación, y expone los resultados del registro como un LiveData observable para actualizar la interfaz de usuario de manera reactiva.

7.

HomeActivity
- binding: ActivityHomeBinding
+ onCreate(savedInstanceState: Bundle): void + openFragment(fragment: Fragment): void

Responsabilidad: actúa como el controlador principal de la navegación en la aplicación, gestionando la interacción del usuario con la barra de navegación inferior. Su función principal es cargar y cambiar entre los fragmentos correspondientes (HomeFragment, ChatsFragment, PerfilFragment, y FiltrosFragment) según las selecciones del usuario. También inicializa la vista principal mediante ActivityHomeBinding y establece la interfaz gráfica base para alojar los fragmentos en el contenedor correspondiente.

8- HomeFragment.java

HomeFragment
- binding: FragmentHomeBinding - authProvider: AuthProvider - postViewModel: PostViewModel - authViewModel: AuthViewModel
+ newInstance(): HomeFragment + onCreate(savedInstanceState: Bundle): void + onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle): View + onViewCreated(view: View, savedInstanceState: Bundle): void + setupMenu(): void + onDestroyView(): void + onLogout() : void

Responsabilidad: gestiona la interfaz que proporcionando una lista de publicaciones y opciones de navegación. Su responsabilidad incluye inicializar el diseño de la interfaz, configurar el RecyclerView para mostrar publicaciones observadas desde el PostViewModel, y gestionar acciones del usuario como abrir la actividad para crear nuevas publicaciones mediante un FloatingActionButton. Además, administra un menú para realizar acciones como cerrar sesión, delegando esta operación al AuthViewModel. También garantiza la limpieza adecuada de recursos para evitar fugas de memoria.

PerfilFragment
- binding: FragmentPerfilBinding - galleryLauncher: ActivityResultLauncher<Intent>
+ onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle): View + setupMenu(): void + setupToolbar(): void + displayUserInfo(): void + setupGalleryLauncher(): void + setupProfileImageClick(): void + handleImageSelection(imageUri: Uri): void

Responsabilidad: La clase PerfilFragment maneja la visualización y personalización del perfil del usuario.

ChatsFragment
- binding: FragmentPerfilBinding
+ onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle): View + setupMenu(): void

Responsabilidad: permite la comunicación entre usuarios.

FiltrosFragment
- binding: FragmentPerfilBinding - galleryLauncher: ActivityResultLauncher<Intent>
+ onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle): View + setupMenu(): void

Responsabilidad: maneja búsquedas de post según su categoría.

Clases de Utilidad

Validaciones
+ validarTexto(texto: String): boolean + validarNumero(numero: String): int + validarMail(email: String): boolean + validarPass(pass: String, pass1: String): String + controlarPasword(pass: String): boolean

ImageUtils
+ pedirPermisos(activity: Activity, permisos: String[], requestCode: int): void + openGallery(context: Context, launcher: ActivityResultLauncher<Intent>): void + subirImagenAParse(context: Context, imageUri: Uri, callback: ImageUploadCallback): void - getBytesFromInputStream(inputStream: InputStream): byte[]
<<interface>> ImageUploadCallback + onSuccess(imageUrl: String): void + onFailure(e: Exception): void

Responsabilidad: tiene la responsabilidad de gestionar operaciones relacionadas con el manejo de imágenes en una aplicación Android. Esta clase actúa como una utilidad y proporciona métodos estáticos que encapsulan tareas comunes para trabajar con imágenes. Sus principales responsabilidades son:

1. Solicitar permisos y seleccionar imágenes desde la galería.
2. Procesar la imagen seleccionada para convertirla en un formato adecuado para subirla a Parse.
3. Manejar los resultados de la subida, notificando al usuario en caso de éxito o error.

```java
public class ImageUtils {
   // permisos para acceder a imágenes
   public static void pedirPermisos(Activity activity, String[] permisos, int
requestCode) {
       if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
           permisos = new String[]{Manifest.permission.READ_MEDIA_IMAGES};
       } else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
           permisos = new String[]{Manifest.permission.READ_EXTERNAL_STORAGE};
       }
       ActivityCompat.requestPermissions(activity, permisos, requestCode);
   }

   public static void openGallery(Context context, ActivityResultLauncher<Intent>
launcher) {
       Intent intent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
       launcher.launch(intent);
   }

   public static void subirImagenAParse(Context context, Uri imageUri,
ImageUploadCallback callback) {
       InputStream inputStream = null;
       try {
           inputStream = context.getContentResolver().openInputStream(imageUri);
           byte[] bytes = getBytesFromInputStream(inputStream);

           if (bytes == null) {
               callback.onFailure(new Exception("El arreglo de bytes es null"));
               return;
           }
           ParseFile parseFile = new ParseFile("image.jpg", bytes);
           parseFile.saveInBackground(new SaveCallback() {
               @Override
               public void done(ParseException e) {
                   if (e == null) {
                       String imageUrl = parseFile.getUrl();
                       callback.onSuccess(imageUrl);
                   } else {
                       callback.onFailure(e);
                   }
               }
           });
       } catch (IOException e) {
           callback.onFailure(e);
       } finally {
           if (inputStream != null) {
               try {
                   inputStream.close();
               } catch (IOException e) {
                   e.printStackTrace();
               }
           }
       }
   }
```

```
    // Leer bytes de un InputStream  biblioteca de manejo de flujos de datos
    private static byte[] getBytesFromInputStream(InputStream inputStream) throws
IOException {
        ByteArrayOutputStream byteBuffer = new ByteArrayOutputStream();
        int bufferSize = 1024;
        byte[] buffer = new byte[bufferSize];

        int len;
        while ((len = inputStream.read(buffer)) != -1) {
            byteBuffer.write(buffer, 0, len);
        }

        return byteBuffer.toByteArray();
    }

    // Interfaz para manejar el resultado de la subida de imágenes
    public interface ImageUploadCallback {
        void onSuccess(String imageUrl);
        void onFailure(Exception e);
    }
}
```

Clases de Modelo

User
- private String id - private String username - private String email - private String password - private String fotoperfil - private String[] intereses
+ User(String username, String email, String password) //getters y setters

Post
- titulo: String - descripcion: String - duracion: int - categoria: String - presupuesto: double - imagenes: List<String>
+ Post(titulo, descripcion, duracion, categoria, presupuesto) + Post(titulo, descripcion, duracion, categoria, presupuesto, imagenes) //getters y setters

PostActivity
- MAX_IMAGES: int - REQUEST_IMAGE: int - binding: ActivityPostBinding - postViewModel: PostViewModel - imagenesUrls: List<String> - adapter: ImageAdapter - categoria: String - galleryLauncher: ActivityResultLauncher<Intent>
+ onCreate(savedInstanceState: Bundle): void - setupRecyclerView(): void - setupViewModels(): void - setupCategorySpinner(): void - setupGalleryLauncher(): void - publicarPost(): void - updateRecyclerViewVisibility(): void + onRequestPermissionsResult(requestCode: int, permissions: String[], grantResults: int[]): void

Responsabilidad: gestiona la interfaz y el flujo principal de publicación de un post. Su objetivo es permitir que los usuarios creen y publiquen posts que incluyan título, descripción, duración, presupuesto, categoría y un máximo de tres imágenes por post.

PostViewModel
- postSuccess: MutableLiveData<String> - postProvider: PostProvider - posts: LiveData<List<Post>>
+ PostViewModel() + getPostSuccess(): LiveData<String> + publicar(post: Post): void + getPosts(): LiveData<List<Post>>

Responsabilidad: Actúa como intermediario entre la vista (PostActivity) y los datos proporcionados por el modelo (PostProvider).

PostProvider
+ addPost(post: Post): LiveData<String> + getPostsByCurrentUser(): LiveData<List<Post>>

Responsabilidad muestra los dos métodos principales que interactúan con Parse para gestionar las publicaciones de los usuarios y las imágenes asociadas.

PostAdapter
- posts: List<Post>
+ PostAdapter(posts: List<Post>) + onCreateViewHolder(parent: ViewGroup, viewType: int): PostViewHolder + onBindViewHolder(holder: PostViewHolder, position: int): void + getItemCount(): int
<<nested class>> PostViewHolder
+ tvTitulo: TextView *+ tvDescripcion: TextView* *+ ivImage1: ImageView* *+ ivImage2: ImageView* *+ ivImage3: ImageView*
+ PostViewHolder(itemView: View)

Responsabilidad es un adaptador para el RecyclerView, que tiene la responsabilidad de vincular los datos de una lista de objetos Post.

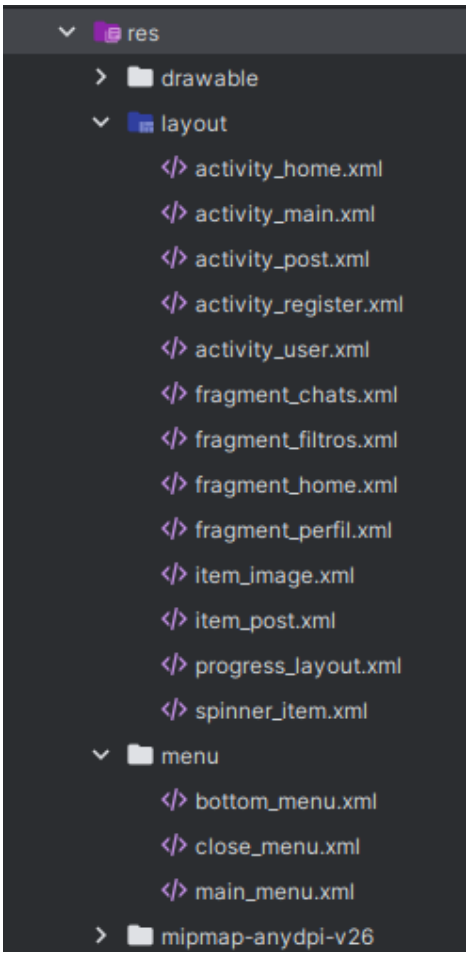
```java
public class PostAdapter extends RecyclerView.Adapter<PostAdapter.PostViewHolder>
{
   private List<Post> posts;
   public PostAdapter(List<Post> posts) {this.posts = posts;}
   @NonNull
   @Override
   public PostViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
       View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.item_post, parent,
false);
       return new  PostViewHolder(view);
   }

   @Override
   public void onBindViewHolder(@NonNull PostViewHolder holder, int position) {
       Post post = posts.get(position);
       holder.tvTitulo.setText(post.getTitulo());
       holder.tvDescripcion.setText(post.getDescripcion());
       Log.d("PostAdapter", "Título: " + post.getImagenes());
       if (post.getImagenes().size() > 0) {
           Picasso.get()
                   .load(post.getImagenes().get(0))
                   .into(holder.ivImage1);
           holder.ivImage1.setVisibility(View.VISIBLE);
       }
       if (post.getImagenes().size() > 1) {
           Picasso.get()
                   .load(post.getImagenes().get(1)) // Cargar la segunda imagen
                   .into(holder.ivImage2);
           holder.ivImage2.setVisibility(View.VISIBLE);
       }
       if (post.getImagenes().size() > 2) {
           Picasso.get()
                   .load(post.getImagenes().get(2)) // Cargar la tercera imagen
                   .into(holder.ivImage3);
           holder.ivImage3.setVisibility(View.VISIBLE);
       }
   }
   @Override
   public int getItemCount() {
       return posts.size();
   }
   public static class PostViewHolder extends RecyclerView.ViewHolder {
       TextView tvTitulo, tvDescripcion;
       ImageView ivImage1, ivImage2, ivImage3;

       public PostViewHolder(View itemView) {
           super(itemView);
           tvTitulo = itemView.findViewById(R.id.tvTitulo);
           tvDescripcion = itemView.findViewById(R.id.tvDescripcion);
           ivImage1 = itemView.findViewById(R.id.ivImage1);
           ivImage2 = itemView.findViewById(R.id.ivImage2);
           ivImage3 = itemView.findViewById(R.id.ivImage3);
       }
```

```
    }
}
```

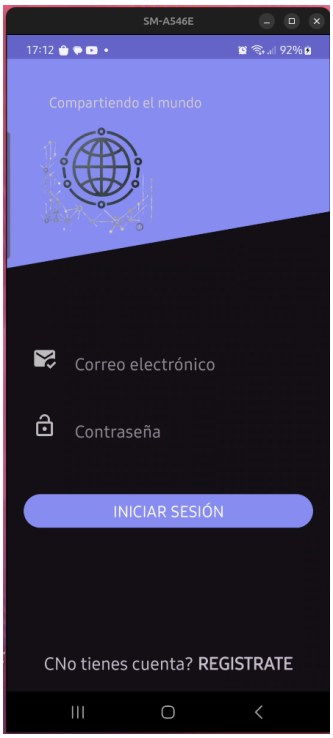
Algunas imagenes aqui

Paquetes que contienen archivos xml del proyecto

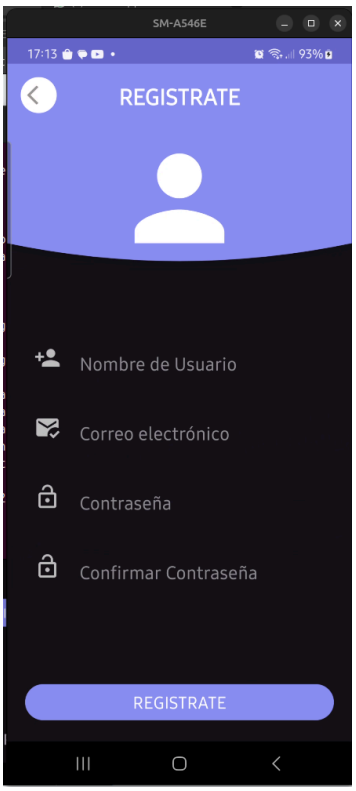
Los archivos XML y las clases Java trabajan en conjunto para definir y manejar la interfaz gráfica y la lógica de una aplicación.
- Inflar (inflate) en Android se refiere al proceso de convertir un archivo XML de diseño (layout) en una jerarquía de objetos `View` en memoria que Android puede mostrar en pantalla. Esto sucede porque el XML es solo una representación declarativa que necesita transformarse en objetos reales para ser utilizados.



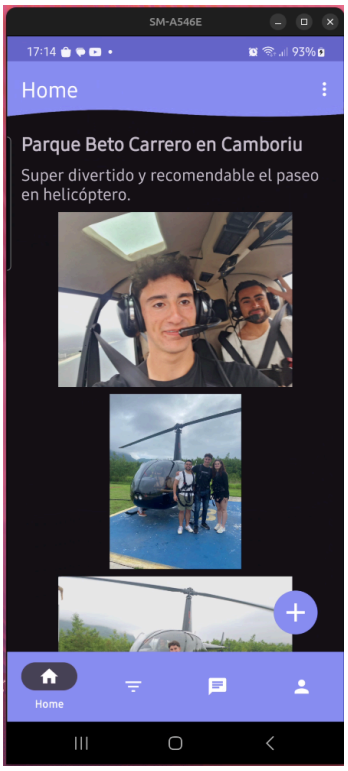
activity_main.xml ver código



activity_register.xml ver código



fragment_home.xml ver código

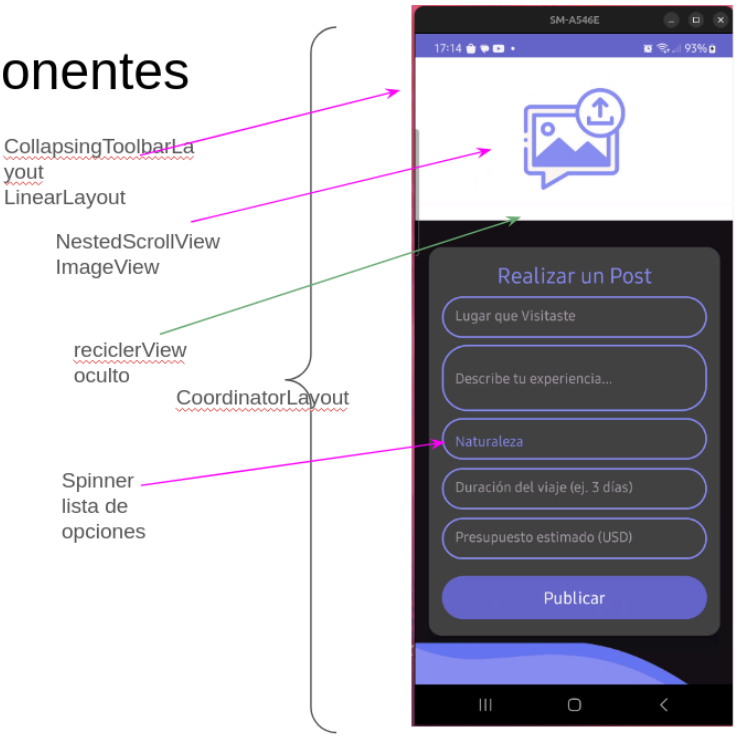


activity_post.xml ver código e item_post.xml ver

actyity_post.xml - Componentes

-**AppBarLayout** es la toolbar
- **CollapsingToolbarLayout** es un componente de la interfaz de usuario en Android que se utiliza comúnmente junto con un AppBarLayout para crear una barra de herramientas que se desplaza y colapsa a medida que el usuario hace scroll.

- **NestedScrollView** es un tipo de contenedor en Android que permite que los elementos dentro de él se desplacen (scroll) de manera anidada, es decir, proporciona un área de desplazamiento que puede contener otros elementos desplazables, como listas, formularios o imágenes, sin perder el control sobre el desplazamiento.



El Spinner - Menú desplegable

ArrayAdapter: El **Spinner** necesita un **ArrayAdapter** para conectar una fuente de datos (como un arreglo de cadenas) con el widget. El **ArrayAdapter** convierte cada ítem del arreglo en un elemento visual que se muestra en el Spinner.

Selección de ítem: Al hacer clic en el Spinner, se muestra un menú desplegable con las opciones disponibles. El usuario selecciona un ítem y el Spinner muestra la opción seleccionada.

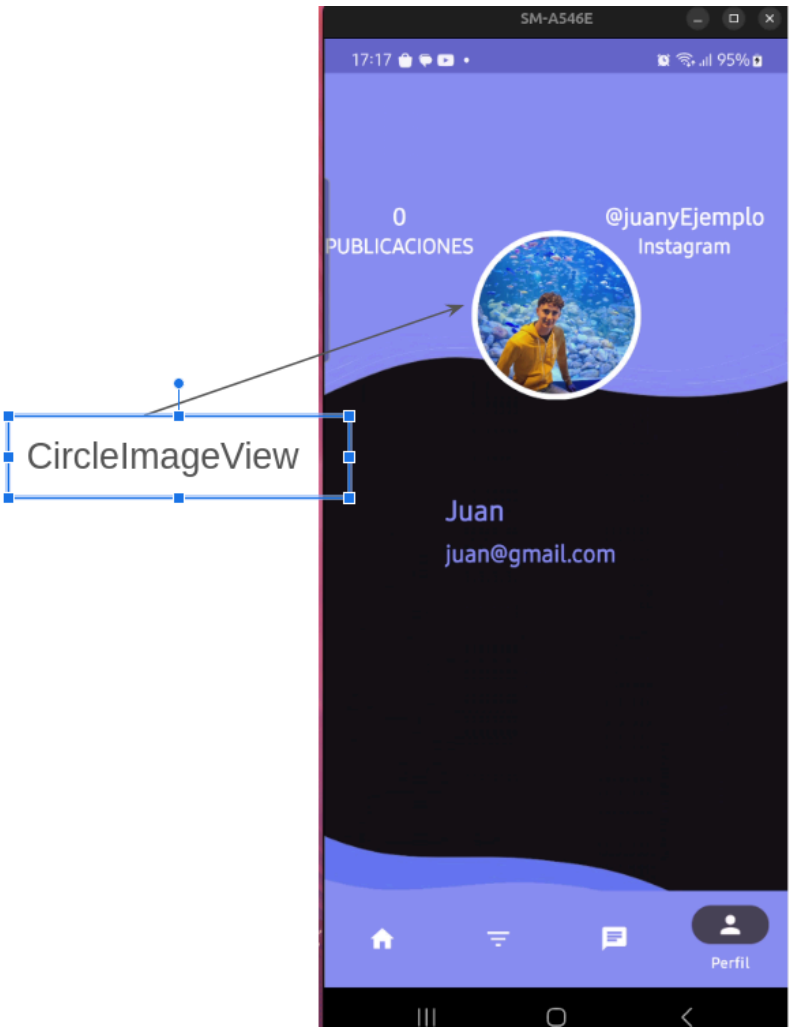
Listener: Un **OnItemSelectedListener** permite manejar la selección del usuario, obteniendo el ítem elegido y ejecutando acciones basadas en la selección.

spinner_item.xml

```
<TextView xmlns:android="http://schemas.android.com/ap
    android:id="@android:id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@color/lavender"
    android:textSize="13sp"
    android:padding="16dp"
/>
```

```
private void setupCategorySpinner() {  1 usage  mariel
    ArrayAdapter<String> adapter = new ArrayAdapter<>(
            context: this, R.layout.spinner_item, getResources().getStringArray(R.array.categorias_array)
    );
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    binding.spinnerCategoria.setAdapter(adapter);
    binding.spinnerCategoria.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {  mariel
        @Override  1 usage  mariel
        public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
            categoria = (String) parent.getItemAtPosition(position);
        }

        @Override  1 usage  mariel
        public void onNothingSelected(AdapterView<?> parent) { categoria = null; }
    });
}
```

fragment_perfil.xml ver código


SM-A546E
17:17
95%
0
PUBLICACIONES
@juanyEjemplo
Instagram
CircleImageView
Juan
juan@gmail.com
Perfil