Documentación del Proyecto: *Ticketing*

Ángel Millán

Noviembre 2025

Índice

1. Introducción

El proyecto **Ticketing** es una aplicación Android diseñada para la gestión y control de incidencias mediante la creación, actualización y visualización de *tickets*. Su objetivo es permitir un flujo sencillo y ordenado para registrar problemas y realizar su seguimiento.

La aplicación está desarrollada utilizando una sola *Activity* y múltiples *Fragments*, siguiendo un enfoque tipo MVVM para una mejor organización del código. Se ha trabajado con herramientas como Android Studio, AndroidX y JUnit para pruebas unitarias.

2. Objetivos del Proyecto

- Crear un sistema funcional de gestión de tickets.
- Aplicar el uso de *Fragments* dentro de una única *Activity*.
- Implementar pruebas unitarias con JUnit.
- Practicar el patrón MVVM y separación por capas.

3. Descripción General de la Aplicación

La aplicación permite realizar las siguientes acciones:

- Mostrar la lista de tickets.
- Crear nuevos tickets.
- Editar tickets existentes.
- Filtrar por estado.

Cada ticket contiene:

- ID único.
- Título.
- Descripción.
- Responsable.
- Estado (Nuevo, Abierto, Pendiente, Resuelto o Cerrado).

4. Requisitos del Sistema

- IDE: Android Studio.
- SDK de Android configurado.
- Dispositivo físico o emulador para pruebas.
- Lenguaje principal: Java.
- Librerías AndroidX.

5. Arquitectura del Proyecto

El proyecto sigue un enfoque simplificado del patrón MVVM:

Modelo

Incluye las clases relacionadas con los datos:

- **Ticket**: Contiene título, descripción, responsable, estado e ID.
- **TicketStatus**: Enumeración con los diferentes estados del ticket.

Vista

Formada por los *Fragments*:

- **HomeFragment**: Muestra la lista de tickets.
- **CreateFragment**: Formulario para crear tickets.
- **UpdateFragment**: Formulario para editar un ticket.

ViewModel / Lógica

- **TicketManager**: Gestiona la lista de tickets, creación, edición y almacenamiento.

6. Diagrama de Flujo

El diagrama de flujo muestra cómo se gestiona la creación, actualización y visualización de tickets dentro de la aplicación.

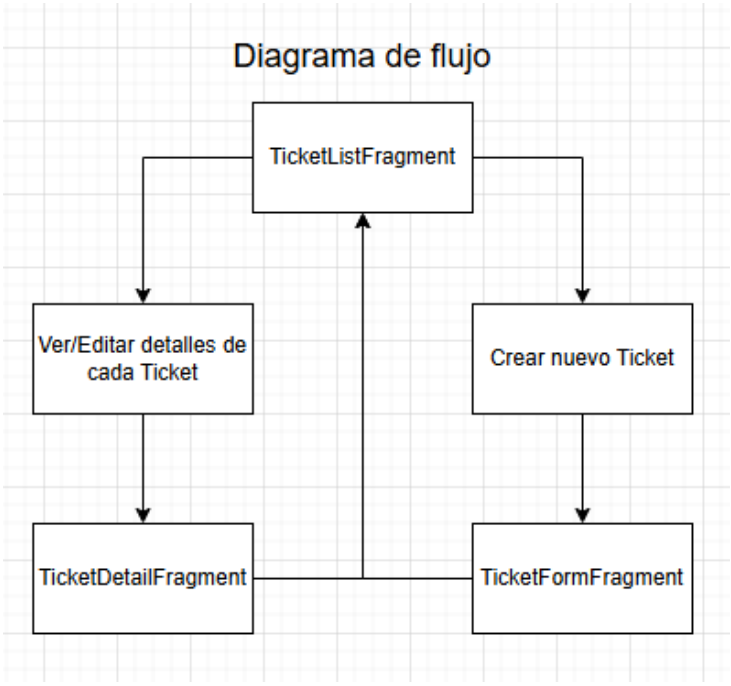

Figura 1: Diagrama de flujo de gestión de tickets

7. Diagrama de Arquitectura y Clases

A continuación se muestra un diagrama unificado que representa la arquitectura general de la aplicación *Ticketing* junto con las relaciones entre las clases principales.

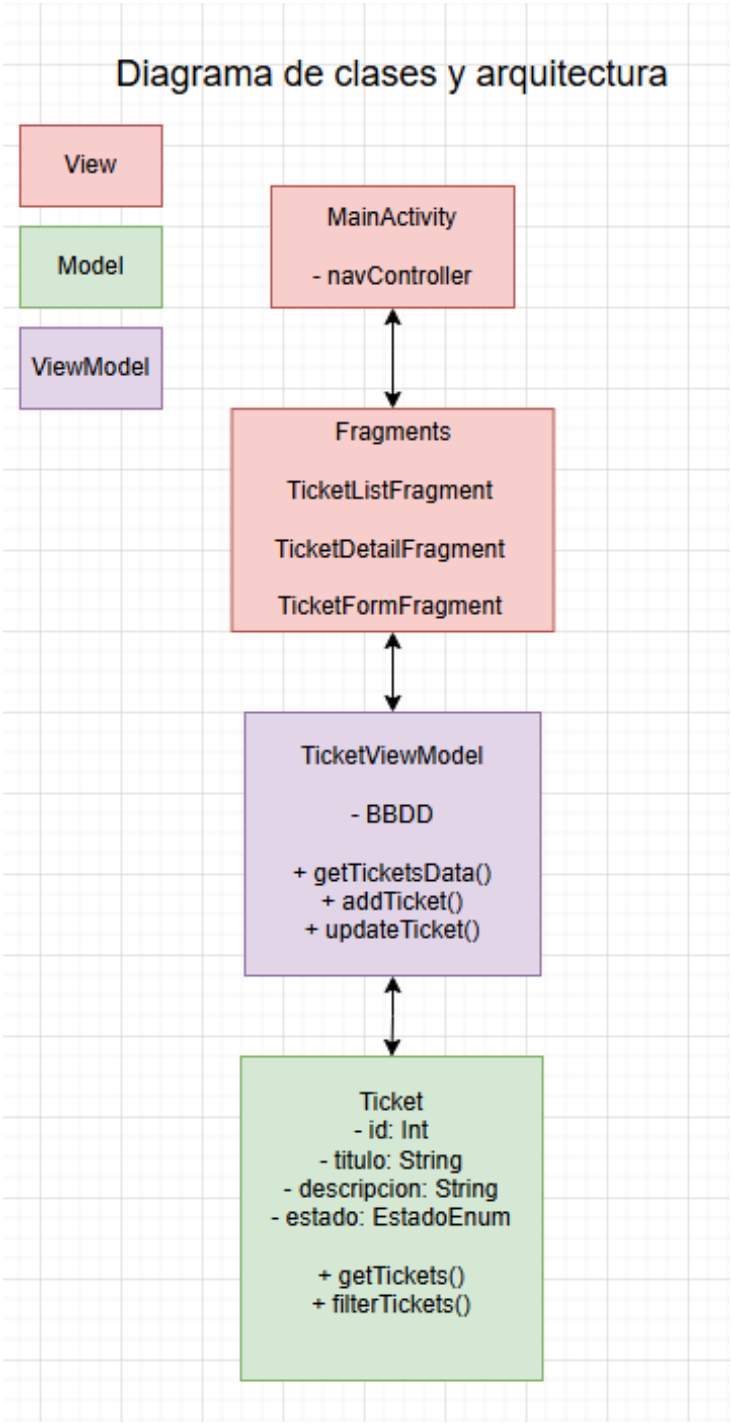

Figura 2: Diagrama unificado de arquitectura y clases de la aplicación

8. Mockups de la Aplicación

A continuación se presenta un mockup que integra todas las pantallas principales de la aplicación: la lista de tickets, el formulario de creación y el formulario de edición.

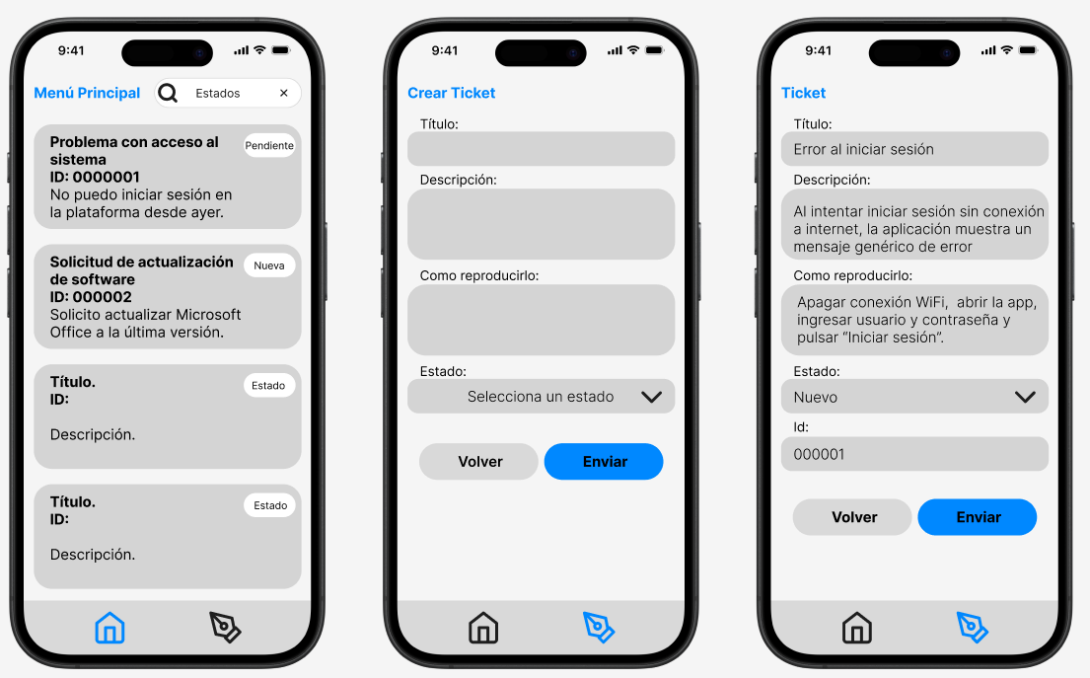

Figura 3: Mockup general de la aplicación *Ticketing*

9. Estructura

- **Model**: Ticket, TicketStatus.
- **View**: Fragments, Adapter, MainActivity.
- **ViewModel**: TicketManager.

10. Código Esencial del Proyecto

A continuación se muestra el código esencial para el funcionamiento de la aplicación *Ticketing*, incluyendo el modelo, la gestión de datos y los fragments principales.

10.1. Ticket.java

```
package com.example.tickets.Model;

public class Ticket {
    private int id;
    private String title;
    private String description;
    private String responsible;
```

```
    private TicketStatus status;

    public Ticket(int id, String title, String description,
                  String responsible, TicketStatus status) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.responsible = responsible;
        this.status = status;
    }

    public int getId() { return id; }
    public String getTitle() { return title; }
    public String getDescription() { return description; }
    public String getResponsible() { return responsible; }
    public TicketStatus getStatus() { return status; }

    public void setTitle(String title) { this.title = title; }
    public void setDescription(String description) { this.description
       ↪  = description; }
    public void setResponsible(String responsible) { this.responsible
       ↪  = responsible; }
    public void setStatus(TicketStatus status) { this.status = status
       ↪ ; }
}
```

10.2. TicketStatus.java

```
package com.example.tickets.Model;

public enum TicketStatus {
    New,
    Open,
    Pending,
    Resolved,
    Closed
}
```

10.3. TicketManager.java

```
package com.example.tickets.ViewModel;

import com.example.tickets.Model.Ticket;
import com.example.tickets.Model.TicketStatus;
import java.util.ArrayList;
import java.util.List;

public class TicketManager {
```

```
    private List<Ticket> tickets = new ArrayList<>();
    private int nextId = 1;

    public List<Ticket> getTickets() {
        return tickets;
    }

    public boolean createTicket(String title, String description,
       ↪ String responsible) {
        Ticket t = new Ticket(nextId++, title, description,
           ↪ responsible, TicketStatus.New);
        tickets.add(t);
        return true;
    }

    public boolean updateTicket(int id, String title, String
       ↪ description,
                                String responsible, TicketStatus
                                   ↪ status) {
        for (Ticket t : tickets) {
            if (t.getId() == id) {
                t.setTitle(title);
                t.setDescription(description);
                t.setResponsible(responsible);
                t.setStatus(status);
                return true;
            }
        }
        return false;
    }
}
```

10.4. TicketAdapter.java

```
package com.example.tickets.View;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.recyclerview.widget.RecyclerView;
import com.example.tickets.Model.Ticket;
import java.util.List;

public class TicketAdapter extends RecyclerView.Adapter<
   ↪ TicketViewHolder> {

    private List<Ticket> tickets;
    private OnTicketClick listener;

    public interface OnTicketClick {
```

```
        void onClick(Ticket ticket);
    }

    public TicketAdapter(List<Ticket> tickets, OnTicketClick listener
        ↪ ) {
        this.tickets = tickets;
        this.listener = listener;
    }

    @Override
    public TicketViewHolder onCreateViewHolder(ViewGroup parent, int
        ↪ viewType) {
        View v = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_ticket, parent, false);
        return new TicketViewHolder(v);
    }

    @Override
    public void onBindViewHolder(TicketViewHolder holder, int
        ↪ position) {
        Ticket t = tickets.get(position);
        holder.bind(t, listener);
    }

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

10.5. HomeFragment.java

```
package com.example.tickets.View;

import android.os.Bundle;
import android.view.View;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import com.example.tickets.ViewModel.TicketManager;

public class HomeFragment extends Fragment {

    private TicketManager manager;

    public HomeFragment(TicketManager manager) {
        super(R.layout.fragment_home);
        this.manager = manager;
    }

    @Override
```

```
    public void onViewCreated(View view, Bundle savedInstanceState) {
        RecyclerView recycler = view.findViewById(R.id.
            ↪ recyclerTickets);
        recycler.setLayoutManager(new LinearLayoutManager(getContext
            ↪ ()));

        TicketAdapter adapter =
            new TicketAdapter(manager.getTickets(), ticket -> {
                // navegaci n al fragmento de actualizaci n
            });

        recycler.setAdapter(adapter);
    }
}
```

10.6. CreateFragment.java

```
package com.example.tickets.View;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import androidx.fragment.app.Fragment;
import com.example.tickets.ViewModel.TicketManager;

public class CreateFragment extends Fragment {

    private TicketManager manager;

    public CreateFragment(TicketManager manager) {
        super(R.layout.fragment_create);
        this.manager = manager;
    }

    @Override
    public void onViewCreated(View v, Bundle savedInstanceState) {
        EditText txtTitle = v.findViewById(R.id.txtTitle);
        EditText txtDesc = v.findViewById(R.id.txtDescription);
        EditText txtResp = v.findViewById(R.id.txtResponsible);
        Button btnCreate = v.findViewById(R.id.btnCreate);

        btnCreate.setOnClickListener(view -> {
            manager.createTicket(
                txtTitle.getText().toString(),
                txtDesc.getText().toString(),
                txtResp.getText().toString()
            );
        });
    }
```

```
}
```

10.7. UpdateFragment.java

```
package com.example.tickets.View;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import androidx.fragment.app.Fragment;
import com.example.tickets.Model.Ticket;
import com.example.tickets.Model.TicketStatus;
import com.example.tickets.ViewModel.TicketManager;

public class UpdateFragment extends Fragment {

    private TicketManager manager;
    private Ticket ticket;

    public UpdateFragment(TicketManager manager, Ticket ticket) {
        super(R.layout.fragment_update);
        this.manager = manager;
        this.ticket = ticket;
    }

    @Override
    public void onViewCreated(View v, Bundle savedInstanceState) {
        EditText txtTitle = v.findViewById(R.id.txtTitle);
        EditText txtDesc = v.findViewById(R.id.txtDescription);
        EditText txtResp = v.findViewById(R.id.txtResponsible);

        txtTitle.setText(ticket.getTitle());
        txtDesc.setText(ticket.getDescription());
        txtResp.setText(ticket.getResponsible());

        Button btnUpdate = v.findViewById(R.id.btnUpdate);

        btnUpdate.setOnClickListener(view -> {
            manager.updateTicket(
                ticket.getId(),
                txtTitle.getText().toString(),
                txtDesc.getText().toString(),
                txtResp.getText().toString(),
                TicketStatus.Open
            );
        });
    }
}
```

11. Pruebas Unitarias

A continuación se muestra una prueba simple para comprobar el correcto funcionamiento de la visualización de tickets:

Listing 1: Prueba unitaria para mostrar tickets

```
@Test
public void showTickets_SimpleCheck() throws IOException {
    manager = new TicketManager();

    // Crear algunos tickets
    manager.createTicket("Ticket 1", "Desc 1", "Resp 1");
    manager.createTicket("Ticket 2", "Desc 2", "Resp 2");

    // Obtener lista
    var lista = manager.getTickets();

    // Comprobaciones b sicas
    assertNotNull(lista);
    assertEquals(2, lista.size());
    assertEquals("Ticket 1", lista.get(0).getTitle());
    assertEquals("Ticket 2", lista.get(1).getTitle());
}
```

12. Conclusiones

El proyecto *Ticketing* demuestra el uso práctico de un patrón MVVM simplificado aplicado a una aplicación Android real. La separación por capas, el uso de Fragments y la implementación de pruebas unitarias ayudan a mejorar la mantenibilidad y escalabilidad del sistema.