

Two thick, horizontal, slightly wavy bars. The top one is teal and the bottom one is yellow, both spanning the width of the page.

LostPaws

Autor: Raúl Martín Torrabadella Mendoza

Tutor: Javier Martín Rivero

Fecha de entrega: 28/04/2025

Convocatoria: 2024 2025

Motivación

Encontré a mi gato en la calle y esa experiencia me hizo ver cuántos animales perdidos o abandonados necesitan una segunda oportunidad. Esta app busca conectar a quienes los encuentran con quienes los buscan o desean adoptar, facilitando su rescate y promoviendo la adopción responsable. Mi objetivo es que más animales tengan la oportunidad de encontrar un hogar y una nueva vida.

Objetivo General

Desarrollar una aplicación móvil que facilite la localización y adopción de animales perdidos o abandonados, conectando a los usuarios de manera sencilla y eficiente.

Objetivos Específicos

1. **Facilitar el reencuentro de animales perdidos** con sus dueños a través de publicaciones y alertas en la app.
2. **Ofrecer un espacio para la adopción de animales** encontrados o no deseados, promoviendo la tenencia responsable.
3. **Registrar información de mascotas**, incluyendo vacunas, citas médicas y tratamientos.
4. **Permitir la comunicación directa entre usuarios** para agilizar adopciones y rescates.
5. **Integrar información sobre refugios y centros de adopción** para que los usuarios puedan encontrar opciones cercanas.
6. **Proporcionar información sobre veterinarios** para facilitar el acceso a ayuda sanitaria.
7. **Enviar notificaciones sobre situaciones críticas**, como refugios saturados o eventos de rescate.
8. **Incluir una sección de donaciones** a refugios y organizaciones, sin integración de pagos, solo informativa.

Metodología

Metodología Utilizada: KANBAN

Descripción:

Dado que el proyecto se realizará de manera individual, se ha optado por utilizar la metodología **Kanban**, la cual se adapta perfectamente a las necesidades de un trabajo autónomo. Kanban es un enfoque ágil que permite gestionar el flujo continuo de tareas mediante un tablero visual donde se organizan las actividades en diferentes etapas: **Pendiente, En Progreso y Hecho**. Esta metodología proporciona flexibilidad y permite adaptar las tareas según las necesidades del momento, sin la necesidad de seguir ciclos fijos de desarrollo como en otras metodologías ágiles.

Ventajas:

- **Flujo continuo:** Permite avanzar en las tareas de manera constante y organizada, sin la rigidez de ciclos de tiempo.
- **Facilidad de uso:** Es simple de implementar y entender, lo que resulta ideal para un único desarrollador. Además, se puede gestionar fácilmente mediante herramientas como Trello o Google Sheets.
- **Flexibilidad:** La capacidad de ajustar las prioridades o el enfoque del proyecto a medida que surjan nuevos desafíos o cambios.
- **Visualización del trabajo:** Facilita la **gestión visual** del progreso, asegurando que ninguna tarea se quede olvidada o se retrase innecesariamente.

Desventajas:

- Si no se gestiona correctamente, puede ser difícil **mantener el enfoque** en las tareas clave, ya que la estructura flexible no tiene un marco rígido de entregas o plazos fijos.
- Requiere disciplina para no dejar que el proyecto se des controle si hay demasiadas tareas en progreso al mismo tiempo.

Justificación:

Kanban es la opción más adecuada para este TFG, ya que permite un control claro del avance del proyecto y facilita la adaptación a cambios imprevistos. Como el trabajo se realizará de manera individual, se beneficiará de su simplicidad y flexibilidad, permitiendo avanzar de manera eficiente sin la necesidad de coordinación constante con un equipo.

Tecnologías y Herramientas Utilizadas

Para el desarrollo de esta aplicación, se han seleccionado herramientas y tecnologías que permiten crear un producto eficiente, moderno y escalable. A continuación, se detallan las principales elecciones y sus razones:

1. Lenguaje de Programación

- **Kotlin:** Se ha elegido por ser el lenguaje oficial para el desarrollo de aplicaciones Android. Ofrece una sintaxis concisa, seguridad frente a NullPointerException y compatibilidad total con Java, facilitando el desarrollo y mantenimiento del proyecto.

2. Entorno de Desarrollo

- **Android Studio:** Es el entorno de desarrollo integrado (IDE) oficial para Android. Proporciona herramientas avanzadas como emuladores, depuración en tiempo real y compatibilidad con Jetpack, lo que facilita la creación de interfaces modernas y funcionales.

3. Base de Datos y Almacenamiento

- **Firebase Realtime Database:** Se utilizará para almacenar la información de los animales, usuarios y publicaciones en tiempo real.
- **Google Drive API:** Se integrará para el almacenamiento de imágenes, permitiendo a los usuarios subir y acceder a fotos de animales de manera segura y eficiente. Google Drive ofrece escalabilidad y control sobre los archivos sin necesidad de mantener una infraestructura de almacenamiento propia.

4. Diseño y Edición Gráfica

- **Adobe Photoshop:** Se utilizará para la creación de logotipos, iconos e imágenes dentro de la aplicación, garantizando una interfaz visual atractiva y profesional.

5. Notificaciones y Servicios en la Nube

- **Firebase Cloud Messaging (FCM):** Permitirá el envío de notificaciones push a los usuarios, alertando sobre animales perdidos o situaciones críticas en su área.

6. Comunicación entre Usuarios

- **Firestore Authentication:** Se utilizará para gestionar el registro y autenticación de usuarios de forma segura mediante correo electrónico o cuentas de Google.

Justificación de la Elección

La combinación de estas herramientas permite desarrollar una aplicación robusta, escalable y fácil de mantener. Firestore elimina la necesidad de un backend complejo, mientras que Kotlin y Android Studio garantizan una experiencia de desarrollo fluida y eficiente. La integración con Google Drive API para el almacenamiento de imágenes ofrece una solución segura y confiable sin necesidad de gestionar un servidor propio.

Planning - Diagrama de Gantt

Está en un documento aparte → [TFG - Planning](#)

Análisis

Requisitos Funcionales

1. **Registro de Usuarios:** Los usuarios pueden registrarse con su correo o Google y acceder de manera segura.
2. **Subida y Visualización de Imágenes:** Los usuarios pueden subir y ver fotos de animales perdidos a través de Google Drive API.
3. **Publicación de Animales:** Los usuarios pueden crear publicaciones con detalles de animales perdidos o encontrados, incluyendo fotos y ubicación.
4. **Búsqueda y Filtrado:** Los usuarios pueden buscar y filtrar publicaciones por tipo de animal, ubicación, fecha, etc.
5. **Reencuentro de Animales Perdidos:** Los usuarios pueden contactar entre sí para devolver mascotas perdidas.
6. **Adopción:** Los usuarios pueden ofrecer animales para adopción y contactar a interesados.
7. **Gestión de Información Médica:** Los usuarios pueden registrar y consultar información médica de sus mascotas.

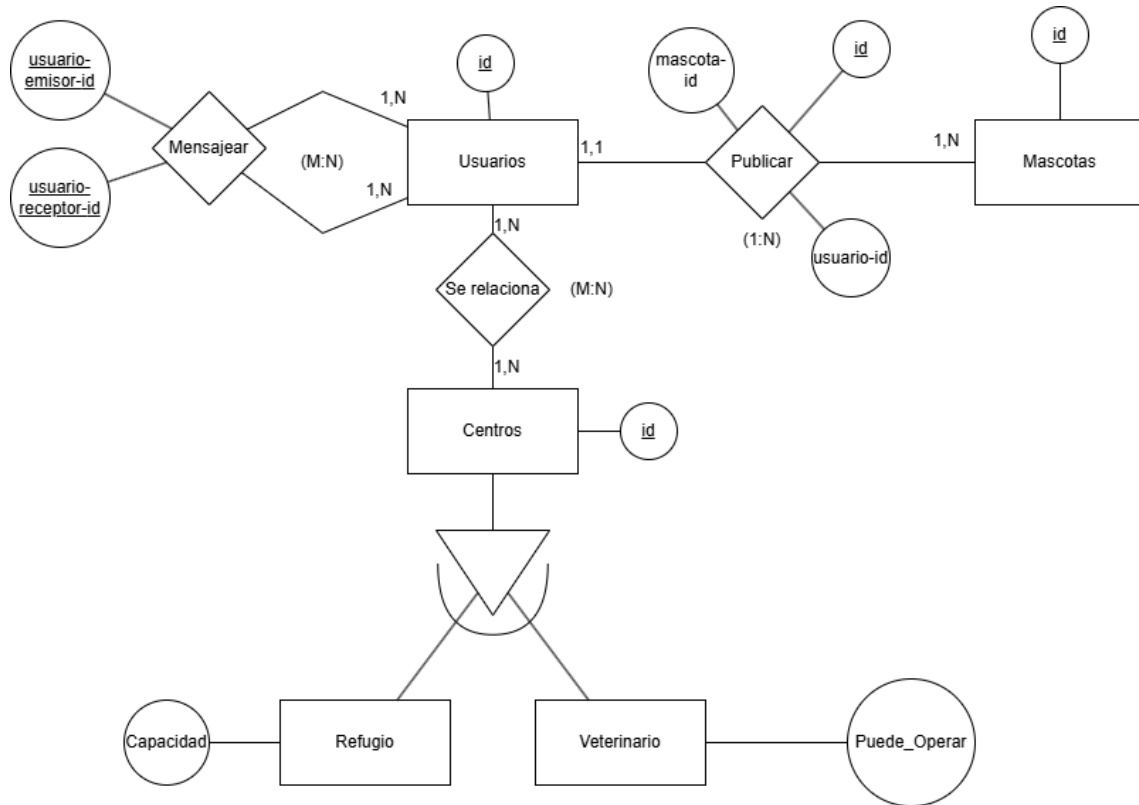
8. **Notificaciones Push:** Los usuarios recibirán notificaciones sobre animales cercanos o situaciones críticas.
9. **Comunicación entre Usuarios:** Los usuarios pueden comunicarse para facilitar adopciones y rescates.
10. **Donaciones:** Los usuarios pueden ver opciones de refugios para donar, sin gestión de pagos.

Refugios y Veterinarios: Se incluirá información de refugios y veterinarios cercanos.

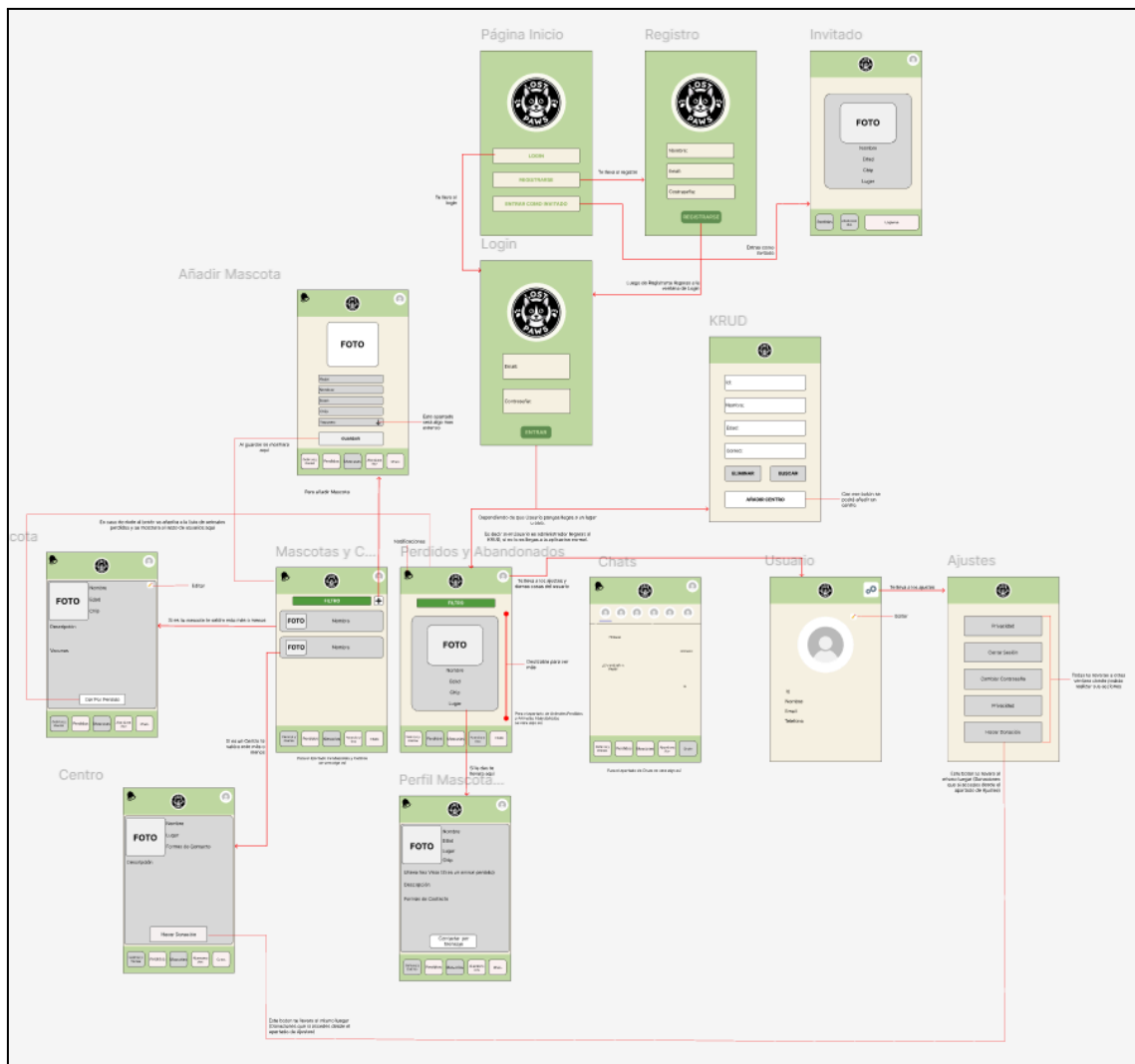
Requisitos No Funcionales

1. **Seguridad:** Uso de autenticación segura con Firebase y almacenamiento seguro de imágenes en Google Drive.
2. **Rendimiento:** Respuesta rápida y capacidad de manejar alto volumen de usuarios y datos.
3. **Escalabilidad:** Arquitectura escalable para futuras funcionalidades o más usuarios.
4. **Compatibilidad:** Compatible con dispositivos Android desde la versión 5.0.
5. **Usabilidad:** Interfaz intuitiva y fácil de usar, con diseño limpio.
6. **Disponibilidad:** La app debe estar disponible 24/7.
7. **Accesibilidad:** Accesible para personas con discapacidades.
8. **Backup y Recuperación:** Copias de seguridad automáticas para restaurar datos si ocurre un fallo.

Entidad Relación



Diseño



Para comprender mejor el **mockup**, es recomendable abrir el enlace y verlo en grande. Esto permite leer con claridad cada comentario y seguir las flechas, que indican a qué parte lleva cada elemento, facilitando así la comprensión del flujo y la interacción dentro del diseño.

Mock Up → [LostPaws](#)

Abstract

The motivation for this project stems from a personal experience that deeply affected me. I found my current pet, a black and white cat named Fígaro, on the street. That experience made me understand how many animals are abandoned and deserve a second chance. This was an experience that stuck with me and eventually formed the idea of a project that I proposed to my teacher Soraya last year. That is when I realized that I wanted to do something meaningful, not just academically but socially significant too.

This project was born out of the necessity to give people a way to easily interact with animal shelters and veterinary clinics. I wanted to simplify adoption processes and offer a simple tool for people who, like me, adore animals and would like to help. The application's logo is derived from Fígaro himself, representing where this project began and why the entire project existed.

Choosing this as my final project was both sensible and instinctual. It is both in sync with my technical interest as well as with my background. I have the motivation of wanting to put technology to use for something meaningful, and this project has given me that.

```
class MainActivity {  
    + onCreate(savedInstanceState: Bundle)  
}  
  
class LoginActivity {  
    + onCreate(savedInstanceState: Bundle)  
    + verificarUsuario(email: String, password: String)  
    + guardarSesion(email: String, context: Context)  
    - validacionLogin(context: Context, email: String, password: String): Boolean  
}  
  
class RegistroActivity {  
    + onCreate(savedInstanceState: Bundle)
```

```
+ guardarInfoUsuarios(name: String, email: String, password: String)

- validacionDatos(context: Context, name: String, email: String, password:
String, callback: (Boolean) -> Unit): Boolean

}
```

```
class MasterActivity {

+ onCreate(savedInstanceState: Bundle)

+ onFragmentChange(fragment: Fragment)

}
```

```
class AdminActivity {

+ onCreate(savedInstanceState: Bundle)

+ realizarBusqueda(searchType: String)

+ eliminarUsuario(usuario: UsuarioVistaAdmin)

}
```

```
class AdminCentrosActivity {

+ db: DatabaseReference

+ onCreate(savedInstanceState: Bundle)

}
```

```
class UsuarioOpcionesActivity {

+ onCreate(savedInstanceState: Bundle)

+ onFragmentChange(fragment: Fragment)

+ onFragmentSuperiorChange(fragment: Fragment)

}
```

' Fragments superiores

```
class FragmentSuperior {
```

```
+ onCreate(savedInstanceState: Bundle)

    + onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View?

+ onCreateView(view: View, savedInstanceState: Bundle?)
}

class FragmentSuperiorUsuario {

    + onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View?

    + onAttach(context: Context)

    + onCreateView(view: View, savedInstanceState: Bundle?)
}

class FragmentSuperiorInvitado {

    + onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View?
}
```

' Funcionalidades

```
class Ajustes {

    + onCreateView(inflater: LayoutInflater, container: ViewGroup,
savedInstanceState: Bundle)

    + onCreateView(view: View, savedInstanceState: Bundle?)

    + onAttach(context: Context)
}

class CerrarSesion {

    + onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?)

    + onCreateView(view: View, savedInstanceState: Bundle?)
```

```
+ cerrarsesion()

}

class EditarUsuario {

    + onCreateView(inflater: LayoutInflater, container:
ViewGroup?,savedInstanceState: Bundle?)

    + onCreateView(view: View, savedInstanceState: Bundle?)

    + onAttach(context: Context)

    + obtenerSesion(context: Context): String?

    + guardarSesion(context: Context, email: String)

    + obtenerDatosUsuario(email: String, callback: (String?, String?, String?) ->
Unit)

    + actualizarDatosUsuario(nuevoNombre: String, nuevoEmail: String)

    + comprobarEmailExistente(email: String, callback: (Boolean) -> Unit)

}

class BorrarCuenta {

    + onCreateView(inflater: LayoutInflater, container:
ViewGroup?,savedInstanceState: Bundle?)

    + onCreateView(view: View, savedInstanceState: Bundle?)

    + obtenerSesion(context: Context): String?

    + borrarCuenta()

}

class CambiarContrasenya {

    + onCreateView(inflater: LayoutInflater, container:
ViewGroup?,savedInstanceState: Bundle?)

    + onCreateView(view: View, savedInstanceState: Bundle?)

    + obtenerSesion(context: Context): String?
```

```
+ obtenerDatosUsuario(email: String, callback: (String?) -> Unit)
+ cambiarcontrasenia(actual: String, nueva: String)
+ onAttach(context: Context)
}

class CambioDePestancias {
    + onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?)
    + onViewCreated(view: View, savedInstanceState: Bundle?)
    + onCreate(savedInstanceState: Bundle?)
    + resetButtonColors()
    + onAttach(context: Context)
}

class Donaciones {
    + onCreateView(inflater: LayoutInflater, container: ViewGroup,
savedInstanceState: Bundle)
}

class Privacidad {
    + onCreateView(inflater: LayoutInflater, container: ViewGroup,
savedInstanceState: Bundle)
}

' Módulos principales

class Usuarios {
    + Por desarrollar
}

class Mascotas {
```

```
+ Por desarrollar
}

class Abandonados {
    + Por desarrollar
}

class Perdidos {
    + Por desarrollar
}

class Chats{
    + Por desarrollar
}

class Centros {
    + onCreate(savedInstanceState: Bundle)
        + onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?)
}
```

' Adaptadores

```
class UserAdapter {
    + onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder
    + onBindViewHolder(holder: ViewHolder, position: Int)
    + getItemCount(): Int
}

class RefugioAdapter {
    + onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder
```

```
+ onBindViewHolder(holder: ViewHolder, position: Int)
+ getItemCount(): Int
}
class VeterinarioAdapter {
    + onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder
    + onBindViewHolder(holder: ViewHolder, position: Int)
    + getItemCount(): Int
}
```

' Interfaces

```
interface OnFragmentChangeListener {
    + onFragmentChange(fragment: Fragment)
}
interface OnFragmentSuperiorChangeListener {
    + onFragmentSuperiorChange(fragment: Fragment)
}
```

' Relaciones entre clases (actualizadas)

MainActivity --> LoginActivity

MainActivity --> RegistroActivity

LoginActivity --> MasterActivity

MasterActivity --> OnFragmentChangeListener

MasterActivity --> FragmentSuperior

MasterActivity --> CambioDePestanas

MasterActivity --> Mascotas

MasterActivity --> Abandonados

MasterActivity --> Perdidos

MasterActivity --> Chats

MasterActivity --> Centros

FragmentSuperior <|-- FragmentSuperiorUsuario

FragmentSuperior --> FragmentSuperiorInvitado

FragmentSuperiorUsuario --> Ajustes

Ajustes --> CerrarSesion

Ajustes --> BorrarCuenta

Ajustes --> Privacidad

Ajustes --> Donaciones

Ajustes --> CambiarContrasenya

Usuarios --> UserAdapter

Usuarios --> EditarUsuario

AdminActivity --> AdminCentrosActivity

AdminActivity --> Usuarios

AdminCentrosActivity --> Centros

Centros --> MasterActivity

Centros --> RefugioAdapter

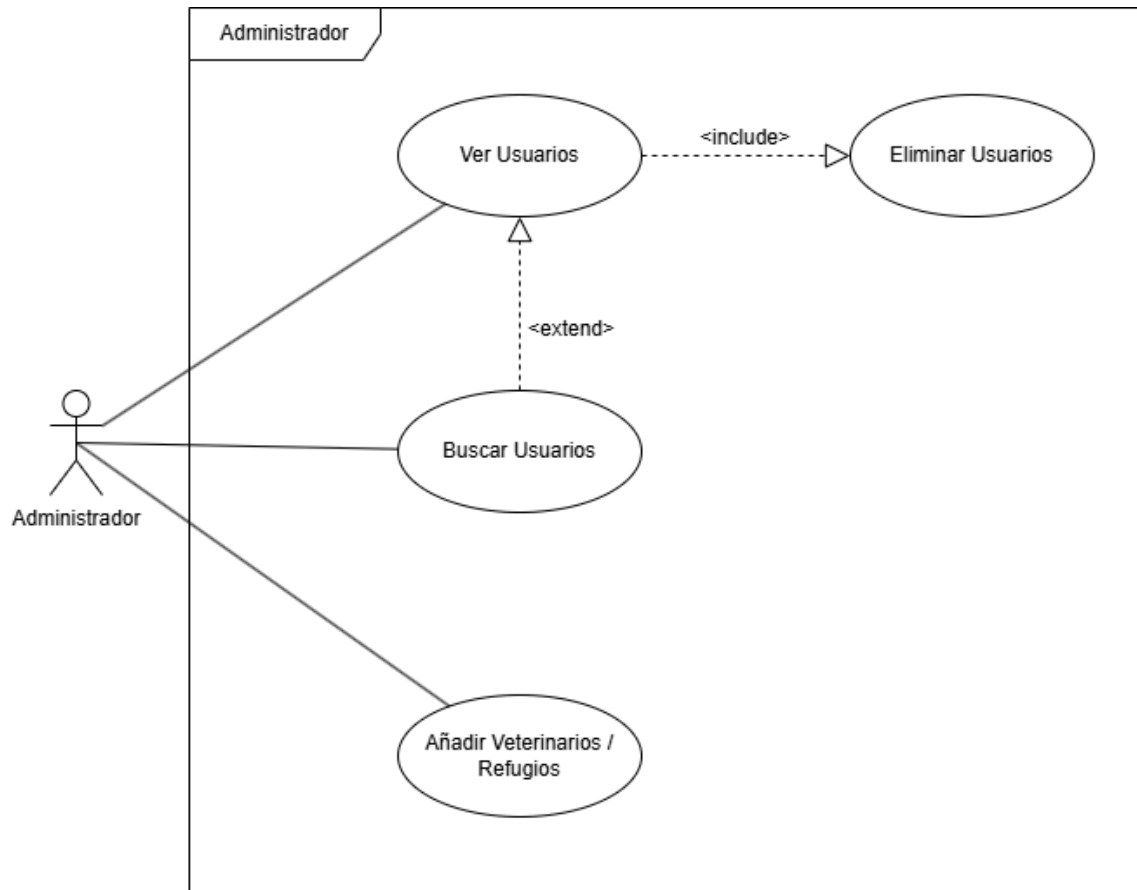
Centros --> VeterinarioAdapter

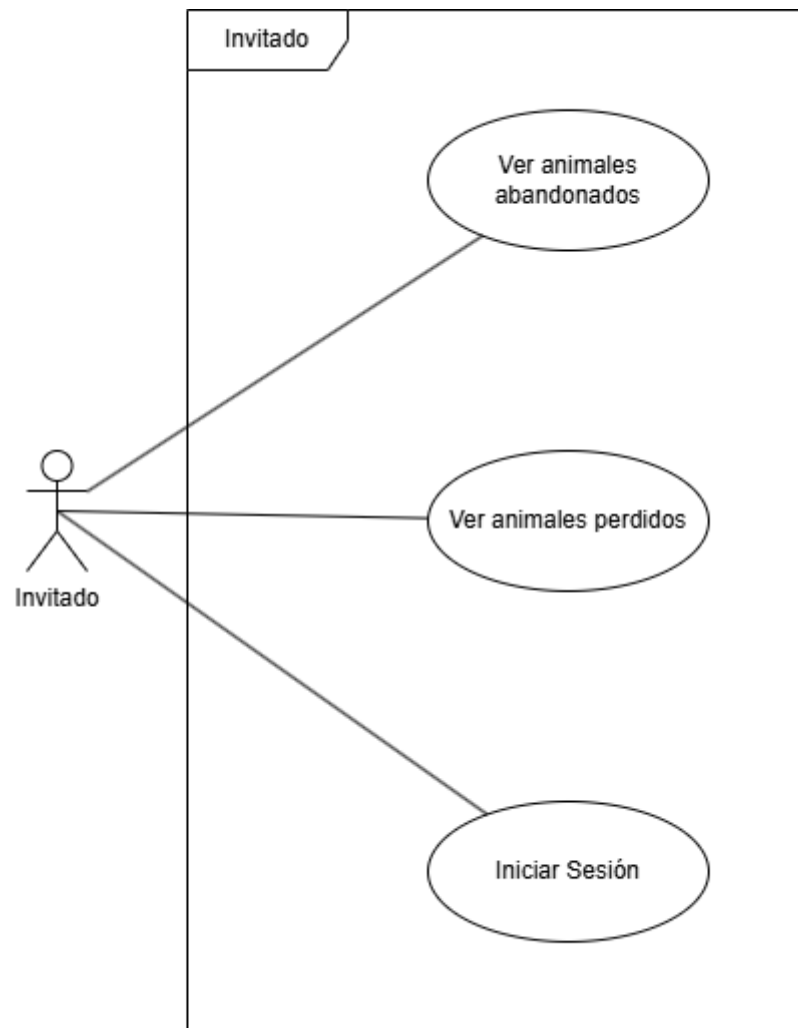
MainActivity --> OnFragmentChangeListener

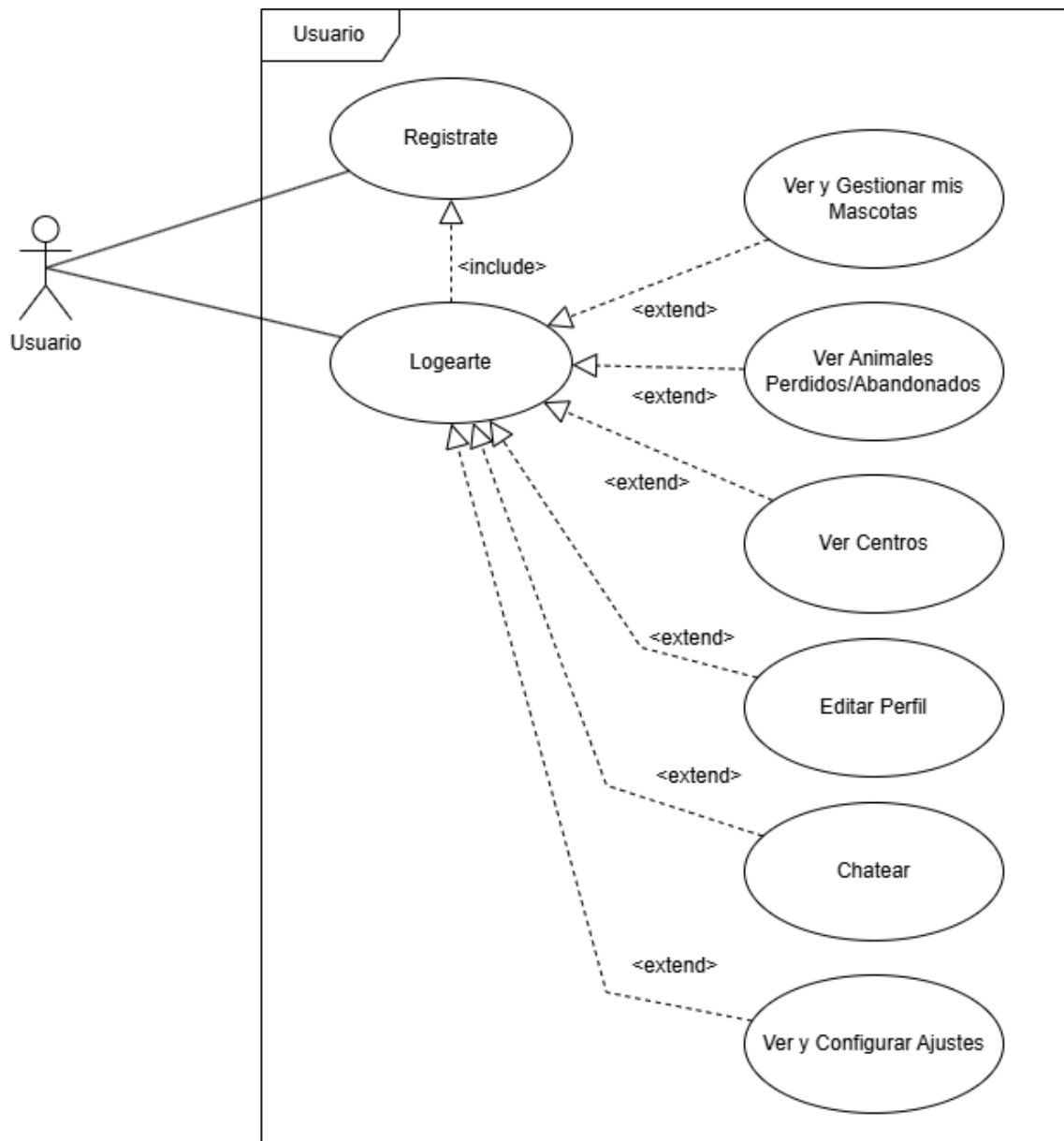
FragmentSuperior --> OnFragmentSuperiorChangeListener

@enduml

Diagrama de Casos de Uso







Documentación de los Códigos

Hasta ahora he realizado los siguientes apartados de mi app:

- Login y Registro en Firebase.
- Cambio de pestañas.
- Apartado del Usuario (junto con su editor de perfil).
- Ajustes (Cerrar Sesión, Cambiar Contraseña, Borrar Cuenta, Donaciones, Privacidad)
- KRUD.
- Centros.

Debido a la cantidad, no voy hablar de todos:

- **Registro**

Primero recojo los datos del usuario y realizo una validación para asegurarme de que todo está correcto antes de enviarlos a Firebase.

```
fun validacionDatos(context: Context, name: String, email: String, password: String, callback: (Boolean) -> Unit): Boolean {  
  
    var correcto = true  
  
    // Validar que la contraseña tenga al menos 6 caracteres  
    if (password.length < 6) {  
        Toast.makeText(context, text: "La contraseña debe tener al menos 6 caracteres.", Toast.LENGTH_SHORT).show()  
        correcto = false  
    }  
  
    if (!android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {  
        Toast.makeText(context, text: "El formato del correo electrónico no es válido.", Toast.LENGTH_SHORT).show()  
        correcto = false  
    }  
  
    // Verificar si el correo ya existe en la base de datos  
    if (correcto) {  
        val database = FirebaseDatabase.getInstance().getReference(path: "users")  
        database.orderByChild(path: "email").equalTo(email).addListenerForSingleValueEvent(object : ValueEventListener {  
            override fun onDataChange(snapshot: DataSnapshot) {  
                if (snapshot.exists()) {  
                    // El correo ya existe en la base de datos  
                    Toast.makeText(context, text: "El correo ya está registrado", Toast.LENGTH_SHORT).show()  
                    callback(false) // Indica que el correo ya está registrado  
                } else {  
                    // Si el correo no existe, indica que la validación es exitosa  
                    callback(true)  
                }  
            }  
            override fun onCancelled(error: DatabaseError) {  
                Toast.makeText(context, text: "Error en la base de datos", Toast.LENGTH_SHORT).show()  
                callback(false) // En caso de error en la base de datos  
            }  
        })  
    } else {  
        callback(false) // En caso de que alguna validación falle  
    }  
  
    return true // La función siempre retorna true ya que la validación asíncronica se maneja en el callback  
}
```

Una vez validados, los datos se guardan en Firebase y se redirige al usuario a la pantalla de **Login**.

```
fun guardarInfoUsuarios(name: String, email: String, password: String){  
    // Crear un objeto User  
    val user = Usuario(name, email, password)  
  
    // Obtener la referencia de Firebase Realtime Database  
    val database = FirebaseDatabase.getInstance()  
    val myRef = database.getReference(path: "users") // Aquí "users" es el nodo donde se guardan los datos  
  
    // Generar un ID único para el usuario (por ejemplo, utilizando push() para crear una nueva entrada)  
    val userId = myRef.push().key // Esto genera una clave única para cada usuario  
  
    // Guardar los datos  
    if (userId != null) {  
        myRef.child(userId).setValue(user)  
        .addOnCompleteListener { task ->  
            if (task.isSuccessful) {  
                // Los datos se guardaron correctamente  
                println("Datos guardados correctamente")  
            } else {  
                // Hubo un error  
                println("Error al guardar los datos: ${task.exception?.message}")  
            }  
        }  
    }  
}
```

```
validacionDatos(context: this, name, email, password) { isValid ->  
    if (isValid) {  
        // Si la validación fue exitosa, guarda la información  
        guardarInfoUsuarios(name, email, password)  
  
        // Navega a la pantalla de login  
        val intent = Intent(packageContext: this, LoginActivity::class.java)  
        startActivity(intent)  
    }  
}
```

- **Login**

El proceso de inicio de sesión comienza con la validación de los datos ingresados.


```
fun validacionLogin(context: Context, email: String, password: String): Boolean {  
    if (password.isEmpty()) {  
        Toast.makeText(context, text: "La contraseña no puede estar vacía.", Toast.LENGTH_SHORT).show()  
        return false  
    }  
  
    if (!android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {  
        Toast.makeText(context, text: "El correo electrónico no es válido.", Toast.LENGTH_SHORT).show()  
        return false  
    }  
  
    return true  
}
```

Luego se realiza una comprobación en Firebase para verificar que el usuario existe.

Primero se comprueba si es un **admin** para redirigirlo al módulo **KRUD**.

```
private fun verificarUsuario(email: String, password: String) {  
    val databaseAdmin = FirebaseDatabase.getInstance().getReference(path: "admin")  
    databaseAdmin.orderByChild(path: "email").equalTo(email).addListenerForSingleValueEvent(object :  
        ValueEventListener {  
            override fun onDataChange(snapshot: DataSnapshot) {  
                if (snapshot.exists()) {  
                    for (userSnapshot in snapshot.children) {  
                        val storedPassword = userSnapshot.child(path: "password").getValue(String::class.java)  
                        if (storedPassword == password) {  
                            // Usuario autenticado correctamente  
                            Toast.makeText(applicationContext, text: "Bienvenido Administrador", Toast.LENGTH_SHORT).show()  
  
                            // Ir a la pantalla principal  
                            val intent = Intent(applicationContext, AdminActivity::class.java)  
                            startActivity(intent)  
                            finish()  
                        }  
                    }  
                }  
            }  
        })  
    } else {
```

Si no es admin, se verifica el correo y la contraseña:

```
} else {  
  
    val database = FirebaseDatabase.getInstance().getReference(path: "users")  
    database.orderByChild(path: "email").equalTo(email).addListenerForSingleValueEvent(object :  
        ValueEventListener {  
            override fun onDataChange(snapshot: DataSnapshot) {  
                if (snapshot.exists()) {  
                    for (userSnapshot in snapshot.children) {  
                        val storedPassword = userSnapshot.child(path: "password").getValue(String::class.java)  
                        if (storedPassword == password) {  
                            // Usuario autenticado correctamente  
                            Toast.makeText(applicationContext, text: "Inicio de sesión exitoso", Toast.LENGTH_SHORT).show()  
  
                            // Me guardo el correo para saber quien se ha registrado y pillar su info  
                            guardarSesion(email, applicationContext)  
  
                            // Ir a la pantalla principal  
                            val intent = Intent(applicationContext, MasterActivity::class.java)  
                            startActivity(intent)  
                            finish()  
                        } else {  
                            Toast.makeText(applicationContext, text: "Contraseña incorrecta", Toast.LENGTH_SHORT).show()  
                        }  
                    }  
                } else {  
                    Toast.makeText(applicationContext, text: "No existe una cuenta con este correo", Toast.LENGTH_SHORT).show()  
                }  
            }  
        })  
  
    override fun onCancelled(error: DatabaseError) {  
        Toast.makeText(applicationContext, text: "Error en la base de datos", Toast.LENGTH_SHORT).show()  
    }  
}  
})
```

Si las credenciales son correctas:

- **Admin** → se redirige al KRUD.
- **Usuario normal** → se redirige a la MasterActivity, donde se realiza la actividad principal del usuario.
- **Cambio de Pestañas:**

No hay mucho que destacar aquí, ya que es el mismo sistema utilizado en **Dating Love** (el cual también desarrollé yo).

La única diferencia es que tuve que crear un controlador adicional para el **Fragment Superior**, ya que existen tres versiones distintas visibles para el usuario.

```
LoginActivity.kt  OnFragmentManager.kt x
1 package com.example.lostpaws
2
3 import androidx.fragment.app.Fragment
4
5 interface OnFragmentManager {
6     fun onFragmentManager(fragment: Fragment)
7 }
8
```

```
LoginActivity.kt  OnFragmentManagerSuperior.kt x
1 package com.example.lostpaws
2
3 import androidx.fragment.app.Fragment
4
5 interface OnFragmentManagerSuperior {
6     fun onFragmentManagerSuperior(fragment: Fragment)
7 }
8
```

```
override fun onFragmentManager(fragment: Fragment) {
    val fragmentManager = supportFragmentManager
    val fragmentTransaction = fragmentManager.beginTransaction()

    // Reemplazar el fragmento actual con el nuevo fragmento
    fragmentTransaction.replace(R.id.fragment_container2, fragment)
    fragmentTransaction.addToBackStack(name = null) // Agregar a la pila de retroceso si es necesario

    fragmentTransaction.commit()
}

override fun onFragmentManagerSuperior(fragment: Fragment) {
    val fragmentManager = supportFragmentManager
    val fragmentTransaction = fragmentManager.beginTransaction()

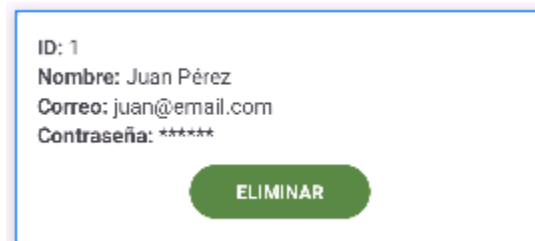
    // Reemplazar el fragmento actual con el nuevo fragmento
    fragmentTransaction.replace(R.id.fragment_container1, fragment)
    fragmentTransaction.addToBackStack(name = null) // Agregar a la pila de retroceso si es necesario

    fragmentTransaction.commit()
}
```

Solo es necesario crear una variable para cada listener y luego llamarla junto con su función, pasándole el nuevo fragment correspondiente.

- **KRUD:**

Para este módulo utilizo un **RecyclerView** donde muestro tarjetas diseñadas en XML. Estas tarjetas contienen la información del usuario y un botón para eliminarlo.



La búsqueda de usuarios se gestiona mediante un **Spinner** que define el tipo de búsqueda. Al pulsar el botón de buscar, se cargan los datos según el criterio seleccionado y se muestran en las tarjetas.

```
private fun realizarBusqueda(searchType: String) {
    val query = findViewById<EditText>(R.id.textoBusqueda).text.toString()

    userList.clear()

    when (searchType) {
        "Todo" -> {
            db.child(pathString: "users")
                .get()
                .addOnSuccessListener { snapshot ->
                    for (child in snapshot.children) {
                        val id = child.key ?: ""
                        val name = child.child(path: "name").getValue(String::class.java) ?: ""
                        val email = child.child(path: "email").getValue(String::class.java) ?: ""
                        val password = child.child(path: "password").getValue(String::class.java) ?: ""

                        val usuario = UsuarioVistaAdmin(id, name, email, password)
                        userList.add(usuario)
                    }
                    adapter.notifyDataSetChanged()
                }
        }
        "ID" -> {
            db.child(pathString: "users").child(query) // query contiene el ID completo
                .get()
                .addOnSuccessListener { snapshot ->
                    userList.clear()
                    if (snapshot.exists()) {
                        val id = snapshot.key ?: ""
                        val name = snapshot.child(path: "name").getValue(String::class.java) ?: ""
                        val email = snapshot.child(path: "email").getValue(String::class.java) ?: ""
                        val password = snapshot.child(path: "password").getValue(String::class.java) ?: ""

                        val usuario = UsuarioVistaAdmin(id, name, email, password)
                        userList.add(usuario)
                    }
                    adapter.notifyDataSetChanged()
                }
            addOnFailureListener {
            }
        }
    }
}
```

Este es un ejemplo de la recogida de los datos según el tipo de búsqueda.

```
recyclerView = findViewById(R.id.userList)
recyclerView.layoutManager = LinearLayoutManager(context: this)
adapter = UserAdapter(context: this, userList) { usuario ->
    eliminarUsuario(usuario)
}
recyclerView.adapter = adapter

// Fuera del spinner.setOnItemSelectedListener
botonBusqueda.setOnClickListener {
    val searchType = spinner.selectedItem.toString() // Opción actual del spinner
    realizarBusqueda(searchType)
}
```

Aparte cuenta con un botón de eliminar.

```
private fun eliminarUsuario(usuario: UsuarioVistaAdmin) {
    usuario.id?.let {
        db.child(pathString: "users").child(it).removeValue()
            .addOnSuccessListener {
                userList.remove(usuario)
                adapter.notifyDataSetChanged()
            }
            .addOnFailureListener { e ->
                // Manejar error al eliminar usuario
            }
    }
}
```

Además de esto, implementé un adaptador que conecta los datos con las tarjetas, permitiendo que toda la información se visualice correctamente.

En general, hay mucho código ya funcional en la app. Si tuviera que explicar cada módulo con detalle, fácilmente ocuparía entre 10 y 15 páginas.

Para conocer más a fondo el funcionamiento, se recomienda **probar la app** o directamente **consultar el código fuente**.

Pruebas de Caja Negra

Todos los tests están incluidos y subidos dentro de la app.

RegisterActivity

Comenzamos con el módulo de registro.

Para realizar la **validación de datos**, fue necesario crear una función aparte, ya que comprobar directamente los **Toast** o las subidas a **Firestore** resulta muy complejo en pruebas automatizadas.

Además, la función que normalmente utilizo para validar correos (!android.util.Patterns.EMAIL_ADDRESS.matcher(email)) **no está disponible en los entornos de test**, así que opté por una **aproximación funcional** para realizar las validaciones de email.

```
fun validacionDatosParaTest(email: String, password: String): Boolean {  
    if (password.length < 6) return false  
  
    val emailPattern = "[a-zA-Z0-9._-]+@[a-z]+\\.[a-z]+"  
    if (!email.matches(emailPattern.toRegex())) return false  
    return true  
}
```

Estas son las únicas comprobaciones que pueden hacerse sin conexión directa a la base de datos, por lo tanto, todas las validaciones de campos se probaron de forma aislada mediante funciones auxiliares.

Debido a las limitaciones para capturar **Toast**, comprobar accesos a Firebase y validar la navegación automática entre actividades, se creó una **función aparte** para poder comprobar el comportamiento esperado ante diferentes inputs.

```
// Para el test
fun validacionLoginParaTest(email: String, password: String): Boolean {
    if (password.isEmpty()) return false

    val emailPattern = "[a-zA-Z0-9._-]+@[a-z]+\\.+[a-z]+"
    if (!email.matches(emailPattern.toRegex())) return false
    return true
}
```

```
LogginActivity.kt  LogginActivityKtTest.kt x
package com.example.lostpaws

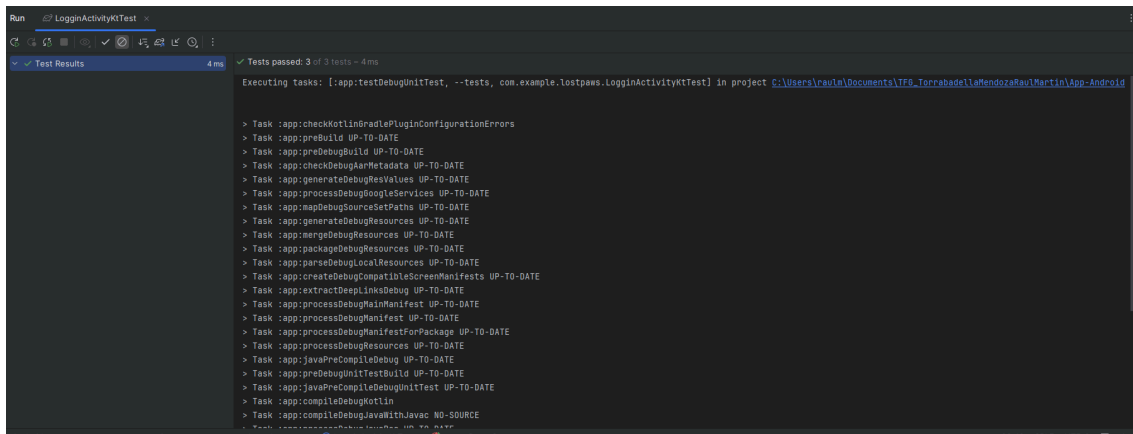
import ...

class LogginActivityKtTest {

    @Test
    fun testSinPassword() {
        val resultado = validacionLoginParaTest(email: "test@example.com", password: "")
        assertFalse(resultado)
    }

    @Test
    fun testEmailInvalido() {
        val resultado = validacionLoginParaTest(email: "correo-mal", password: "123456")
        assertFalse(resultado)
    }

    @Test
    fun testDatosCorrectos() {
        val resultado = validacionLoginParaTest(email: "test@example.com", password: "123456")
        assertTrue(resultado)
    }
}
```

Donaciones

Para esta sección se siguió la misma metodología.

Debido a que se usan elementos que dependen de servicios externos y no son fácilmente simulables en tests, también se creó una función auxiliar para realizar las comprobaciones básicas de lógica y validación.

```
// Para los Test (Debido a que usa toast)
// Versión para tests de la función validaciondatos (elimina dependencias de Android)

fun validacionDatosDonacionParaTest(emisor: String, caducidad: String, codigoCVV: String, destinatario: String): Boolean {
    var correcto = true

    if (emisor.length != 16) {
        correcto = false
    }

    if (destinatario.length != 16) {
        correcto = false
    }

    if (codigoCVV.length != 3) {
        correcto = false
    }

    correcto = validarFechaCaducidadParaTest(caducidad, correcto)

    return correcto
}
```

```
// Versión para tests de la función validarfechacaducidad
fun validarFechaCaducidadParaTest(fCaducidad: String, correcto: Boolean): Boolean {
    var correctoLocal = correcto

    // Validación del formato de la fecha
    if (!fCaducidad.matches(Regex(pattern: "\\d{2}/\\d{2}"))) {
        correctoLocal = false
        return correctoLocal
    }

    // Extraemos el mes y año
    val partes = fCaducidad.split(delimiters: "/")
    if (partes.size != 2) {
        return false
    }

    // Convertimos a enteros para hacer comparaciones
    try {
        val mes = partes[0].toInt()
        val año = partes[1].toInt()

        // Validamos el mes
        if (mes !in 1..12) {
            correctoLocal = false
        }

        // Validamos el año
        val añoActual = java.util.Calendar.getInstance().get(java.util.Calendar.YEAR) % 100
        if (año < añoActual) {
            correctoLocal = false
        }
    } catch (e: NumberFormatException) {
        return false
    }

    return correctoLocal
}
```

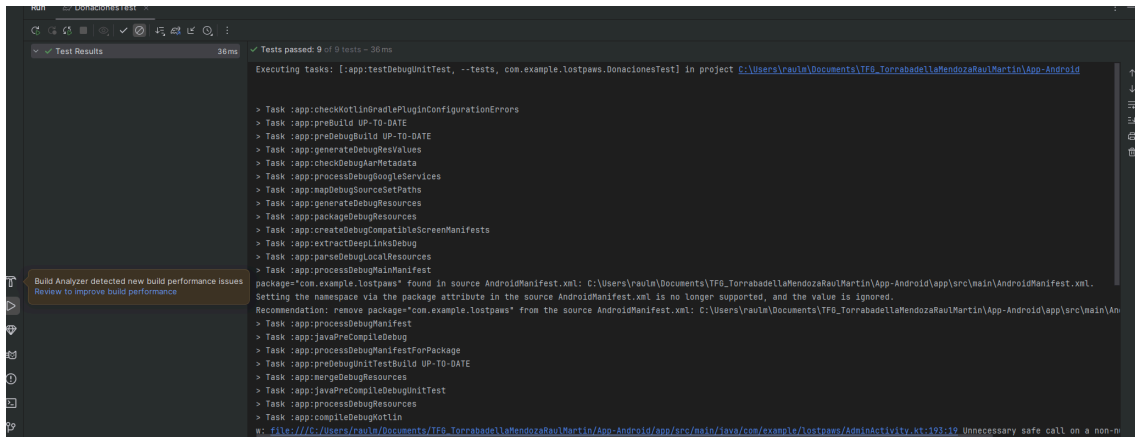
Tests

```
1 package com.example.lostpaws
2
3 import org.junit.Assert.*
4
5 import org.junit.Test
6 import java.util.Calendar
7 import com.example.lostpaws.Donaciones.*
8
9 class DonacionesTest {
10
11     private val donaciones = Donaciones() // Instancia de la clase
12
13     @Test
14     fun testNumeroEmisorCorrecto() {
15         val resultado = donaciones.validacionDatosDonacionParaTest(
16             emisor: "1234567890123456",
17             caducidad: "12/25",
18             codigoCVV: "123",
19             destinatario: "1234567890123456"
20         )
21         assertTrue(resultado)
22     }
23
24     @Test
25     fun testNumeroEmisorIncorrecto() {
26         val resultado = donaciones.validacionDatosDonacionParaTest(
27             emisor: "12345",
28             caducidad: "12/25",
29             codigoCVV: "123",
30             destinatario: "1234567890123456"
31         )
32         assertFalse(resultado)
33     }
34
35     @Test
36     fun testNumeroDestinatarioIncorrecto() {
37         val resultado = donaciones.validacionDatosDonacionParaTest(
38             emisor: "1234567890123456",
39             caducidad: "12/25",
40             codigoCVV: "123",
41             destinatario: "12345"
42         )
43     }
```

```
CambiarContraseña.kt  Donaciones.kt  DonacionesTest.kt x
9      class DonacionesTest {
47     fun testCVVIncorrecto() {
48         val resultado = donaciones.validacionDatosDonacionParaTest(
49             emisor: "1234567890123456",
50             caducidad: "12/25",
51             codigoCVV: "12",
52             destinatario: "1234567890123456"
53         )
54         assertFalse(resultado)
55     }
56
57     @Test
58     fun testFormatoFechaCaducidadIncorrecto() {
59         val resultado = donaciones.validacionDatosDonacionParaTest(
60             emisor: "1234567890123456",
61             caducidad: "1225",
62             codigoCVV: "123",
63             destinatario: "1234567890123456"
64         )
65         assertFalse(resultado)
66     }
67
68     @Test
69     fun testMesFechaCaducidadIncorrecto() {
70         val resultado = donaciones.validacionDatosDonacionParaTest(
71             emisor: "1234567890123456",
72             caducidad: "13/25",
73             codigoCVV: "123",
74             destinatario: "1234567890123456"
75         )
76         assertFalse(resultado)
77     }
78
79     @Test
80     fun testAñoFechaCaducidadCaducado() {
81         val añoActual = Calendar.getInstance().get(Calendar.YEAR) % 100
82         val resultado = donaciones.validacionDatosDonacionParaTest(
83             emisor: "1234567890123456",
84             caducidad: "12/${añoActual - 1}",
85             codigoCVV: "123",
86             destinatario: "1234567890123456"
87         )
88     }
```

```
9 class DonacionesTest {
79     @Test
80     fun testAñoFechaCaducidadCaducado() {
81         val añoActual = Calendar.getInstance().get(Calendar.YEAR) % 100
82         val resultado = donaciones.validacionDatosDonacionParaTest(
83             emisor: "1234567890123456",
84             caducidad: "12/${añoActual - 1}",
85             codigoCVV: "123",
86             destinatario: "1234567890123456"
87         )
88         assertFalse(resultado)
89     }
90
91     @Test
92     fun testDatosCompletos() {
93         val añoActual = Calendar.getInstance().get(Calendar.YEAR) % 100
94         val resultado = donaciones.validacionDatosDonacionParaTest(
95             emisor: "1234567890123456",
96             caducidad: "12/${añoActual + 1}",
97             codigoCVV: "123",
98             destinatario: "1234567890123456"
99         )
100         assertTrue(resultado)
101     }
102
103     @Test
104     fun testFechaConLetras() {
105         val resultado = donaciones.validacionDatosDonacionParaTest(
106             emisor: "1234567890123456",
107             caducidad: "ab/cd",
108             codigoCVV: "123",
109             destinatario: "1234567890123456"
110         )
111         assertFalse(resultado)
112     }
113 }
114 }
```

Resultado



The screenshot shows the Android Studio interface with the 'Test Results' tab selected. The top bar indicates 'Tests passed: 9 of 9 tests - 36ms'. The main panel displays a list of build tasks executed during the test run, including tasks like 'Task :app:checkKotlinGradlePluginConfigurationErrors', 'Task :app:preBuild UP-TO-DATE', 'Task :app:preDebugBuild UP-TO-DATE', 'Task :app:generateDebugResValues', 'Task :app:checkDebugJarMetadata', 'Task :app:processDebugGoogleServices', 'Task :app:mapDebugSourceSetPaths', 'Task :app:generateDebugResources', 'Task :app:packageDebugResources', 'Task :app:createDebugCompatibleScreenManifests', 'Task :app:extractDebugSymbols', 'Task :app:mergeDebugLocalResources', 'Task :app:processDebugMainManifest', 'Task :app:processDebugManifest', 'Task :app:javaPreCompileDebug', 'Task :app:processDebugManifestForPackage', 'Task :app:preDebugUnitTestBuild UP-TO-DATE', 'Task :app:mergeDebugResources', 'Task :app:javaPreCompileDebugUnitTest', 'Task :app:processDebugResources', and 'Task :app:compileDebugKotlin'. A notification on the left states 'Build Analyzer detected new build performance issues. Review to improve build performance.' The bottom of the console shows a warning: 'w: file:///C:/Users/raulm/Documents/TF6_TorrabadellaMendozaRaulMartin/App-Android/app/src/main/java/com/example/lostpaws/AdminActivity.kt:193:19 Unnecessary safe call on a non-null receiver of type com.example.lostpaws.AdminActivity'.

```
Build Analyzer detected new build performance issues
Review to improve build performance

> Task :app:checkKotlinGradlePluginConfigurationErrors
> Task :app:preBuild UP-TO-DATE
> Task :app:preDebugBuild UP-TO-DATE
> Task :app:generateDebugResValues
> Task :app:checkDebugJarMetadata
> Task :app:processDebugGoogleServices
> Task :app:mapDebugSourceSetPaths
> Task :app:generateDebugResources
> Task :app:packageDebugResources
> Task :app:createDebugCompatibleScreenManifests
> Task :app:extractDebugSymbols
> Task :app:mergeDebugLocalResources
> Task :app:processDebugMainManifest
package="com.example.lostpaws" found in source AndroidManifest.xml: C:\Users\raulm\Documents\TF6_TorrabadellaMendozaRaulMartin\AppData\local\Temp\AndroidManifest.xml
Setting the namespace via the package attribute in the source AndroidManifest.xml is no longer supported, and the value is ignored.
Recommendation: remove package="com.example.lostpaws" from the source AndroidManifest.xml: C:\Users\raulm\Documents\TF6_TorrabadellaMendozaRaulMartin\AppData\local\Temp\AndroidManifest.xml
> Task :app:processDebugManifest
> Task :app:javaPreCompileDebug
> Task :app:processDebugManifestForPackage
> Task :app:preDebugUnitTestBuild UP-TO-DATE
> Task :app:mergeDebugResources
> Task :app:javaPreCompileDebugUnitTest
> Task :app:processDebugResources
> Task :app:compileDebugKotlin
w: file:///C:/Users/raulm/Documents/TF6_TorrabadellaMendozaRaulMartin/App-Android/app/src/main/java/com/example/lostpaws/AdminActivity.kt:193:19 Unnecessary safe call on a non-null receiver of type com.example.lostpaws.AdminActivity
```