APP ADOPCION DE MASCOTAS

Documentacion Tecnica del Proyecto

Universidad Nacional de Cajamarca - 2026

Contenido de este documento:

- Arquitectura completa de la aplicacion Android + Supabase
- Explicacion de cada tabla, vista y trigger de la base de datos
- Referencia de todos los modelos Request/Response con sus campos
- Cada metodo de AppRepository: que hace, que recibe y que devuelve
- Flujos completos: registro, login, publicar mascota, favoritos, solicitudes
- Como obtener IDs (id_adoptante, id_refugio, etc.) desde el UUID
- VacunasMascotaRequest: por que es necesario y como usarlo
- Guia de uso del bucket publico de Storage

1. Vision General del Proyecto

La aplicacion conecta refugios de animales con personas interesadas en adoptar. Los refugios publican mascotas, los adoptantes las descubren, guardan favoritas y envian solicitudes de adopcion. Todo el backend corre en Supabase.

1.1 Stack Tecnologico

Capa	Tecnologia
Frontend	Android (Java) + ViewBinding + Material Design 3
Base de datos	Supabase / PostgreSQL 15 con Row Level Security (RLS)
Autenticacion	Supabase GoTrue - devuelve UUID + JWT
Almacenamiento	Supabase Storage (buckets publicos para imagenes)
HTTP / REST	Retrofit 2 + OkHttp 3 + Gson
Carga de imagenes	Glide 4 + clase ImageLoader propia
Mapas	Google Maps SDK + Geocoder de Android (gratuito)

1.2 Roles de Usuario

Rol	Que puede hacer
Adoptante	Buscar mascotas, agregar favoritos, enviar solicitudes de adopcion
Refugio	Publicar mascotas, gestionar solicitudes recibidas, ver agenda de visitas
Admin	Reservado para administracion futura del sistema

1.3 Flujo General de la App

```
WelcomeActivity
   |-- sesion guardada? --> MainActivity (directo, sin pasar por login)
   |-- btn Iniciar Sesion --> LoginActivity --> MainActivity
   |-- btn Registrarse --> BottomSheet (elegir rol)
                           |-- Adoptante --> RegistroAdoptanteActivity
                           |-- Refugio   --> RegistroRefugioActivity

MainActivity (BottomNavigationView)
   |-- tab Descubrir  --> DescubrirFragment  (lista + filtros de mascotas)
   |-- tab Refugios   --> RefugiosFragment   (mapa + lista de refugios)
   |-- tab Yo         --> PerfilFragment
                           |-- rol Adoptante --> AdoptantePerfilFragment
                           |-- rol Refugio   --> RefugioPerfilFragment
```

2. Base de Datos (Supabase / PostgreSQL)

2.1 Tablas y su Proposito

Tabla	Proposito
usuario	Perfil base de TODOS los usuarios. Se vincula con auth.users mediante UUID. Contiene correo, telefono, rol, url de foto.
adoptante	Datos propios del adoptante: nombre, apellido, genero, fecha nacimiento. FK a usuario.
refugio	Datos del refugio: nombre, direccion, latitud, longitud, descripcion, url portada. FK a usuario.
mascota	Animal publicado por un refugio. Tiene FK a refugio y a raza. Incluye ContadorFavoritos (actualizado por trigger).
especie	Catalogo maestro: Perro, Gato, Conejo, Ave, Otro.
raza	Catalogo de razas por especie. Ej: Labrador (Perro), Siames (Gato).
fotomascota	Fotos adicionales de una mascota (galeria). Una mascota puede tener varias fotos.
favorito	Relacion Adoptante <-> Mascota. Unico por par. Trigger actualiza ContadorFavoritos.
solicitudadopcion	Solicitud enviada por adoptante a refugio para adoptar una mascota. Tiene estado: Pendiente / Aprobada / Rechazada / Visita Agendada.
vacunabasica	Catalogo de vacunas por especie. Ej: Rabia (Perro+Gato), Parvovirus (Perro).
detallemascotavacunas	Vacunas que tiene una mascota especifica. Se inserta una fila por cada vacuna que el refugio marque como aplicada.
intervencionmedica	Historial de operaciones, tratamientos, castraciones, etc. de una mascota.

2.2 Vistas (Views) - Por Que Son Importantes

Por que usar vistas en lugar de JOINs en el codigo Android

PostgreSQL ejecuta el JOIN una sola vez en el servidor. El codigo Android recibe un objeto plano con todos los campos ya combinados. Esto simplifica el codigo, reduce errores y mejora el rendimiento.

Vista	Tablas que combina	Campos destacados que agrega
vista_mascotas_completa	mascota + raza + especie + refugio + usuario	nombreRaza, nombreEspecie, nombreRefugio, telefonoRefugio, latitudRefugio, longitudRefugio
vista_refugios_completa	refugio + usuario + count(mascota)	correo, telefono, fotoPerfil, mascotasDisponibles
vista_adoptante_completo	adoptante + usuario + count(favorito) + count(solicitud)	totalFavoritos, totalSolicitudes

2.3 Triggers Automaticos

Los triggers corren en el servidor automaticamente. El codigo Android NO necesita hacer nada extra.

Trigger	Cuando se activa	Que hace automaticamente
trg_favoritos_contador	INSERT o DELETE en favorito	Suma o resta 1 en mascota.ContadorFavoritos
trg_fecha_adopcion	UPDATE mascota: Estado pasa a 'Adoptado'	Guarda la fecha actual en mascota.FechaAdopcion
trg_solicitud_aprobada	UPDATE solicitudadopcion: Estado pasa a 'Aprobada'	Cambia mascota.Estado a 'En Proceso'

3. SessionManager - Gestion de Sesion

SessionManager guarda los datos de sesion en SharedPreferences del dispositivo. Es el punto de partida en CUALQUIER pantalla autenticada. Nunca hardcodees el UUID o el token: siempre usalo desde aqui.

3.1 Que Guarda

Clave guardada	Tipo	De donde viene
uuid	String	auth.users.id que devuelve GoTrue al registrarse o hacer login
token	String	access_token JWT que devuelve GoTrue. Se manda en el header Authorization
rol	String	Campo Rol de la tabla usuario. Se consulta justo despues del login
isLoggedIn	boolean	true despues de guardarSesion(), false despues de cerrarSesion()

3.2 Metodos

Metodo	Retorna	Cuando usarlo
guardarSesion(uuid, token, rol)	void	Justo despues de login o registro exitoso
isLoggedIn()	boolean	En WelcomeActivity para saltar el login si ya hay sesion
getUuid()	String	Para consultar tabla usuario, adoptante o refugio por id_usuario
getToken()	String	Como primer parametro de new AppRepository(token)
getRol()	String	Para decidir que fragment/activity mostrar
esAdoptante()	boolean	Atajo: getRol().equals('Adoptante')
esRefugio()	boolean	Atajo: getRol().equals('Refugio')
cerrarSesion()	void	En el boton 'Cerrar Sesion' de los fragments de perfil

3.3 Uso Tipico en un Fragment

```
// Al inicio de onViewCreated():
SessionManager session = new SessionManager(requireContext());
AppRepository   repo    = new AppRepository(session.getToken());

// Acceder a datos de sesion:
String uuid  = session.getUuid();    // 'a1b2c3d4-e5f6-...'
String token = session.getToken();   // 'eyJhbGciOiJIUzI1NiIsIn...'
String rol   = session.getRol();     // 'Adoptante' o 'Refugio'

// Verificar rol para mostrar/ocultar elementos:
if (session.esRefugio()) {
```

```
    binding.btnPublicarMascota.setVisibility(View.VISIBLE);
    binding.btnAgregarFavorito.setVisibility(View.GONE);
}
```

4. Modelos de Datos (Request y Response)

La convencion es clara: los modelos Request se usan para ENVIAR datos al servidor (POST/PATCH). Los modelos Response se usan para RECIBIR datos del servidor (GET).

4.1 Modelos de Autenticacion

AuthRequest

Campo	Tipo	Descripcion
email	String	Correo del usuario
password	String	Contrasena del usuario

AuthResponse

Campo	Tipo	Como se usa
access_token	String	JWT Bearer. Se guarda en SessionManager como 'token'
user.id	String	UUID del usuario. Se guarda en SessionManager como 'uuid'

4.2 Modelos de Usuario

UsuarioRequest (POST y PATCH /usuario)

Campo JSON	Tipo Java	Descripcion
id_usuario	String	UUID de auth.users. Se obtiene de AuthResponse.getUser().getId()
correo	String	Email del usuario
telefono	String	Numero de telefono
rol	String	'Adoptante' o 'Refugio'. Se decide en el registro
urlimagenusuario	String	URL publica de la foto de perfil en el bucket 'avatars'

4.3 Modelos de Adoptante

AdoptanteRequest (POST /adoptante y PATCH /adoptante)

Campo JSON	Tipo Java	Descripcion
id_usuario	String	UUID que vincula con la tabla usuario
nombre	String	Nombre del adoptante

Campo JSON	Tipo Java	Descripcion
apellido	String	Apellido del adoptante
genero	String	'Masculino', 'Femenino', 'Otro', 'Prefiero no decir'
fechanacimiento	String	Formato 'YYYY-MM-DD'. Ej: '2000-05-15'

AdoptanteResponse (GET /vista_adoptante_completo)

Incluye TODOS los campos de AdoptanteRequest mas los campos de la vista:

Campo extra	Tipo Java	De donde viene
idAdoptante	int	id_adoptante (clave primaria). NECESARIO para favoritos y solicitudes
correo	String	De la tabla usuario
telefono	String	De la tabla usuario
fotoPerfil	String	URL publica de la foto. De usuario.urlimagenusuario
totalFavoritos	int	Calculado por la vista (COUNT)
totalSolicitudes	int	Calculado por la vista (COUNT)

Como obtener id_adoptante

El id_adoptante (int) es diferente al UUID del usuario (String). Para obtenerlo: repo.obtenerAdoptanteCompleto(session.getUuid(), callback) -> en el callback, list.get(0).idAdoptante. Este int es el que se usa en favoritos y solicitudes.

4.4 Modelos de Refugio

RefugioRequest (POST /refugio y PATCH /refugio)

Campo JSON	Tipo Java	Descripcion
id_usuario	String	UUID que vincula con la tabla usuario
nombre	String	Nombre del refugio
direccion	String	Texto de direccion. Viene del MapPickerActivity
latitud	Double	Coordenada. Viene del MapPickerActivity (nullable)
longitud	Double	Coordenada. Viene del MapPickerActivity (nullable)
descripcion	String	Descripcion del refugio (nullable)
urlportada	String	URL publica de la foto de portada en bucket 'refugio-covers'

RefugioResponse (GET /vista_refugios_completa)

Campo extra	Tipo Java	De donde viene
idRefugio	int	id_refugio (clave primaria). NECESARIO para publicar mascotas y solicitudes
correo	String	De la tabla usuario
telefono	String	De la tabla usuario. Se usa para el boton de WhatsApp
fotoPerfil	String	URL de la foto de perfil del refugio
mascotasDisponibles	int	Calculado por la vista (COUNT WHERE Estado='Disponible')

Como obtener id_refugio

Similar al adoptante: repo.obtenerRefugioPorUuid(session.getUuid(), callback) -> list.get(0).idRefugio. Este int es la FK en mascota, solicitudadopcion y favorito.

4.5 Modelos de Mascota

MascotaRequest (POST /mascota y PATCH /mascota)

Campo JSON	Tipo Java	Descripcion
id_refugio	Integer	FK al refugio dueno. Se obtiene de RefugioResponse.idRefugio
id_raza	Integer	FK a la tabla raza. El usuario lo elige de un Spinner/Dropdown
nombremascota	String	Nombre de la mascota
edadanios	Integer	Edad en anos (puede ser 0)
edadmeses	Integer	Edad en meses (complementa edadanios)
urlportadamascota	String	URL publica de la foto principal en bucket 'mascotas'
genero	String	'Macho' o 'Hembra'
temperamento	String	Descripcion del temperamento (nullable)
historia	String	Historia de la mascota (nullable)
estado	String	Default 'Disponible'. El constructor lo pone automaticamente

MascotaResponse (GET /vista_mascotas_completa)

Esta es la respuesta mas completa del sistema porque usa la vista. Incluye datos de mascota + raza + especie + refugio en un solo objeto:

Campo	Tipo Java	Descripcion
idMascota	int	PK de la mascota. Necesario para favoritos, solicitudes, fotos
nombre	String	Nombre de la mascota
edadAnios / edadMeses	int	Edad
urlPortada	String	URL publica de la foto principal
genero	String	'Macho' o 'Hembra'
temperamento	String	Temperamento descrito por el refugio
historia	String	Historia de la mascota
estado	String	'Disponible', 'En Proceso' o 'Adoptado'
contadorFavoritos	int	Veces que fue marcada como favorita (trigger lo mantiene)
fechaRegistro	String	Fecha de publicacion en formato ISO
idRaza / nombreRaza	int/String	De la tabla raza

Campo	Tipo Java	Descripcion
idEspecie / nombreEspecie	int/String	De la tabla especie
idRefugio / nombreRefugio	int/String	Del refugio que la publico
direccionRefugio	String	Para mostrar en detalle de mascota
latitudRefugio / longitudRefugio	Double	Para abrir el mapa desde detalle de mascota
telefonoRefugio	String	Para el boton de WhatsApp en detalle de mascota

4.6 Modelos de Favorito

FavoritoRequest (POST /favorito)

Campo JSON	Tipo Java	Descripcion
id_mascota	int	PK de la mascota. Viene de MascotaResponse.idMascota
id_adoptante	int	PK del adoptante. Viene de AdoptanteResponse.idAdoptante

FavoritoResponse (GET /favorito)

Campo JSON	Tipo Java	Para que se usa
idFavorito	int	PK del favorito. Puede usarse para eliminar
idMascota	int	Para saber que mascota es favorita y cargar su info
idAdoptante	int	Para filtrar solo los del usuario actual
fechaAgregado	String	Fecha en que se agrego a favoritos

4.7 Modelos de Solicitud

SolicitudRequest (POST /solicitudadopcion y PATCH para actualizar estado)

Campo JSON	Tipo Java	Descripcion
id_refugio	Integer	FK al refugio. Viene de MascotaResponse.idRefugio
id_mascota	Integer	FK a la mascota que se quiere adoptar
id_adoptante	Integer	FK al adoptante que envia la solicitud

Campo JSON	Tipo Java	Descripcion
mensaje	String	Mensaje del adoptante al refugio (nullable)
estadodesolicitud	String	'Pendiente' (default), 'Aprobada', 'Rechazada', 'Visita Agendada'
fechavisita	String	Solo cuando el refugio agenda la visita. Formato ISO
notasrefugio	String	Notas internas del refugio al gestionar la solicitud

SolicitudResponse (GET /solicitudadopcion)

Campo	Tipo	Para que se usa
idSolicitud	int	PK. Necesario para PATCH (aprobar/rechazar/agendar)
idRefugio	int	Para filtrar solicitudes del refugio actual
idMascota	int	Para mostrar datos de la mascota en la lista de solicitudes
idAdoptante	int	Para filtrar solicitudes del adoptante actual
mensaje	String	Mensaje enviado por el adoptante
fechaSolicitud	String	Fecha de envio de la solicitud
estado	String	Estado actual: Pendiente / Aprobada / Rechazada / Visita Agendada
fechaVisita	String	Fecha de visita agendada (null si no se ha agendado)

5. VacunasMascotaRequest - Por Que Es Necesario

Respuesta directa a tu pregunta

Si, necesitas VacunasMascotaRequest. La tabla detallemascotavacunas es una tabla de relacion N:M entre mascota y vacunabasica. Cada fila representa 'esta mascota tiene esta vacuna aplicada'. Sin un Request para insertar en esa tabla, no hay forma de registrar las vacunas desde la app.

5.1 Como Funciona el Flujo de Vacunas

Cuando un refugio registra una mascota, debe poder marcar cuales vacunas basicas tiene aplicadas. El flujo es:

```
PASO 1: Cargar catalogo de vacunas segun la especie de la mascota
        repo.obtenerVacunasPorEspecie(idEspecie, callback)
        --> devuelve List<VacunaResponse> con id + nombre de cada vacuna

PASO 2: Mostrar checkboxes al usuario
        Por cada VacunaResponse mostrar un CheckBox con el nombre
        El usuario marca las vacunas que tiene la mascota

PASO 3: Al guardar, insertar una fila por cada vacuna marcada
        for (VacunaResponse v : vacunasSeleccionadas) {
            repo.registrarVacunaMascota(idMascota, v.id, fechaHoy, callback);
        }
```

5.2 VacunasMascotaRequest

Clase data/model/VacunasMascotaRequest.java

```
public class VacunasMascotaRequest {
    @SerializedName("id_mascota")
    public int idMascota;

    @SerializedName("id_vacunabasica")
    public int idVacunaBasica;

    @SerializedName("fechaaplicacion")
    public String fechaAplicacion;  // formato 'YYYY-MM-DD', puede ser null

    public VacunasMascotaRequest(int idMascota, int idVacuna, String fecha) {
        this.idMascota      = idMascota;
        this.idVacunaBasica = idVacuna;
        this.fechaAplicacion = fecha;
    }
}
```

5.3 Agregar el Endpoint en SupabaseApi.java

```
// En SupabaseApi.java, agregar:
@POST("rest/v1/detallemascotavacunas")
Call<Void> registrarVacunaMascota(@Body VacunasMascotaRequest request);
```

5.4 Agregar el Metodo en AppRepository.java

```
// En AppRepository.java, agregar:
public void registrarVacunaMascota(int idMascota, int idVacuna,
                                   String fecha, Callback<Void> cb) {
    api.registrarVacunaMascota(
        new VacunasMascotaRequest(idMascota, idVacuna, fecha)
    ).enqueue(cb);
}
```

5.5 Ejemplo de Uso en la Activity de Publicar Mascota

```
// Despues de crear la mascota y obtener su id:
private void guardarVacunas(int idMascota, List<VacunaResponse>
vacunasSeleccionadas) {
    String hoy = new SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())
                     .format(new Date());
    for (VacunaResponse vacuna : vacunasSeleccionadas) {
        repo.registrarVacunaMascota(idMascota, vacuna.id, hoy,
            new Callback<Void>() {
                @Override
                public void onResponse(Call<Void> call, Response<Void> resp) {
                    // vacuna guardada correctamente
                }
                @Override
                public void onFailure(Call<Void> call, Throwable t) {
                    Log.e("VACUNAS", "Error: " + t.getMessage());
                }
            }
        );
    }
}
```

6. AppRepository - Referencia Completa de Metodos

AppRepository es el UNICO punto de acceso a datos para todo el equipo. Nunca hay que tocar SupabaseApi.java directamente. Cada metodo acepta un Callback<T> de Retrofit.

6.1 Inicializacion

```
// En cualquier Fragment o Activity autenticado:
SessionManager session = new SessionManager(requireContext());
AppRepository   repo    = new AppRepository(session.getToken());

// Solo para registro y login (sin token):
AppRepository repo = new AppRepository();
```

6.2 Metodos de Auth

Metodo	Parametros	Que devuelve (T en Callback<T>)
login(email, password, cb)	email: String, password: String	AuthResponse (tiene access_token y user.id)
registrar(email, password, cb)	email: String, password: String	AuthResponse

6.3 Metodos de Usuario, Adoptante y Refugio

Metodo	Parametros clave	Devuelve
crearUsuario(req, cb)	UsuarioRequest	Void
obtenerUsuario(uuid, cb)	uuid: String (de SessionManager)	List<UsuarioRequest>
actualizarUsuario(uuid, req, cb)	uuid + UsuarioRequest parcial	Void
crearAdoptante(req, cb)	AdoptanteRequest	Void
obtenerAdoptantePorUuid(uuid, cb)	uuid de la sesion	List<AdoptanteRequest>
obtenerAdoptanteCompleto(uuid, cb)	uuid de la sesion	List<AdoptanteResponse> (con contadores)
actualizarAdoptante(uuid, req, cb)	uuid + AdoptanteRequest	Void
crearRefugio(req, cb)	RefugioRequest	Void
obtenerRefugioPorUuid(uuid, cb)	uuid de la sesion	List<RefugioResponse> (vista completa)
obtenerRefugioPorId(idRefugio, cb)	idRefugio: int	List<RefugioResponse>
obtenerTodosRefugios(cb)	ninguno	List<RefugioResponse>

Metodo	Parametros clave	Devuelve
actualizarRefugio(uuid, req, cb)	uuid + RefugioRequest	Void

6.4 Metodos de Mascotas

Metodo	Parametros	Devuelve
obtenerMascotasDisponibles(orden, limit, offset, cb)	orden: 'fecharegistro.desc' o 'contadorfavoritos.desc' limit/offset: int	List<MascotaResponse>
obtenerMascota(idMascota, cb)	idMascota: int	List<MascotaResponse> (1 elemento)
buscarMascotasPorNombre(nombre, orden, cb)	nombre: String (busqueda parcial)	List<MascotaResponse>
filtrarPorEspecie(idEspecie, orden, limit, offset, cb)	idEspecie: int	List<MascotaResponse>
filtrarPorRaza(idRaza, orden, cb)	idRaza: int	List<MascotaResponse>
filtrarPorGenero(genero, orden, cb)	genero: 'Macho' o 'Hembra'	List<MascotaResponse>
obtenerMascotasDeRefugio(idRefugio, orden, cb)	idRefugio: int	List<MascotaResponse>
crearMascota(req, cb)	MascotaRequest	Void
actualizarMascota(idMascota, req, cb)	idMascota: int + MascotaRequest	Void

6.5 Metodos de Favoritos

Metodo	Parametros	Devuelve / Uso
obtenerFavoritos(idAdoptante, cb)	idAdoptante: int	List<FavoritoResponse> con los idMascota guardados
verificarEsFavorito(idMascota, idAdoptante, cb)	ambos int	List<FavoritoResponse> - si size()>0 entonces es favorito
agregarFavorito(idMascota, idAdoptante, cb)	ambos int	Void - el trigger actualiza ContadorFavoritos automaticamente
eliminarFavorito(idMascota, idAdoptante, cb)	ambos int	Void - el trigger reduce ContadorFavoritos automaticamente
mascotaMasPopularDeRefugio(idRefugio, cb)	idRefugio: int	List<MascotaResponse> con 1 elemento (la mas popular)

6.6 Metodos de Solicitudes

Metodo	Parametros	Devuelve / Uso
crearSolicitud(idRefugio, idMascota, idAdoptante, mensaje, cb)	3 ints + String mensaje	Void
obtenerSolicitudesAdoptante(idAdoptante, cb)	idAdoptante: int	List<SolicitudResponse> ordenadas por fecha desc
obtenerSolicitudesRefugio(idRefugio, cb)	idRefugio: int	List<SolicitudResponse> ordenadas por fecha desc
aprobarSolicitud(idSolicitud, fechaVisita, cb)	idSolicitud: int, fechaVisita: String ISO	Void - el trigger cambia mascota a 'En Proceso'
rechazarSolicitud(idSolicitud, notas, cb)	idSolicitud: int, notas: String	Void
agendarVisita(idSolicitud, fechaVisita, cb)	idSolicitud: int, fecha: String ISO	Void

6.7 Metodos de Catalogos y Vacunas

Metodo	Parametros	Devuelve
obtenerEspecies(cb)	ninguno	List<EspecieResponse> - para poblar un Spinner de especie
obtenerRazas(cb)	ninguno	List<RazaResponse> - todas las razas
obtenerRazasPorEspecie(idEspecie, cb)	idEspecie: int	List<RazaResponse> - filtradas por especie
obtenerRazasConMascotas(cb)	ninguno	List<MascotaResponse> - razas que tienen mascotas publicadas actualmente
obtenerVacunasPorEspecie(idEspecie, cb)	idEspecie: int	List<VacunaResponse> - para mostrar checkboxes al registrar mascota
obtenerVacunasMascota(idMascota, cb)	idMascota: int	List<VacunaMascotaResponse> - vacunas de una mascota especifica
obtenerIntervencionesMascota(idMascota, cb)	idMascota: int	List<IntervencionResponse> - historial medico
registrarVacunaMascota(idMascota, idVacuna, fecha, cb)	2 ints + String fecha	Void - agregar vacuna a una mascota (nuevo metodo)

7. Flujos Completos Paso a Paso

7.1 Flujo de Registro (Adoptante)

```
PASO 1 - Auth: RegistroAdoptanteActivity llama a repo.registrar(email,
password)
         --> Supabase GoTrue crea el usuario en auth.users
         --> devuelve AuthResponse con uuid y token

PASO 2 - Subir imagen de perfil:
         ImageHelper.comprimirImagen(uri) --> byte[]
         StorageClient.uploadImage('avatars', uuid + '/profile.jpg', bytes)
         --> devuelve URL publica de la imagen

PASO 3 - Crear registro en tabla usuario:
         new AppRepository(token).crearUsuario(
             new UsuarioRequest(uuid, email, telefono, 'Adoptante', urlImagen)
         )

PASO 4 - Crear registro en tabla adoptante:
         repo.crearAdoptante(
             new AdoptanteRequest(uuid, nombre, apellido, genero, fechaNac)
         )

PASO 5 - Guardar sesion y navegar:
         new SessionManager(this).guardarSesion(uuid, token, 'Adoptante')
         startActivity(new Intent(this, MainActivity.class))
```

7.2 Flujo de Login

```
PASO 1 - Autenticar: repo.login(email, password)
         --> devuelve AuthResponse con uuid + token

PASO 2 - Obtener rol: nuevo AppRepository(token).obtenerUsuario(uuid)
         --> devuelve List<UsuarioRequest>, tomar .get(0).getRol()

PASO 3 - Guardar sesion:
         sessionManager.guardarSesion(uuid, token, rol)

PASO 4 - Navegar a MainActivity (con FLAG_CLEAR_TASK)

NOTA: Si WelcomeActivity detecta session.isLoggedIn() == true,
      salta directamente a MainActivity sin mostrar la pantalla de bienvenida.
```

7.3 Flujo de Publicar Mascota (Refugio)

```
PASO 1 - Obtener id_refugio del refugio actual:
         repo.obtenerRefugioPorUuid(session.getUuid())
         --> int idRefugio = list.get(0).idRefugio

PASO 2 - Obtener id_raza segun especie elegida:
         repo.obtenerRazasPorEspecie(idEspecie)  --> List<RazaResponse>
         usuario elige de un Spinner --> int idRaza = razaSeleccionada.idRaza

PASO 3 - Subir foto de la mascota:
         StorageClient.uploadImage('mascotas', uuid+'/'+timestamp+'.jpg',
bytes)
         --> String urlFoto (URL publica directa)
```

```
PASO 4 - Crear mascota:
         repo.crearMascota(new MascotaRequest(
             idRefugio, idRaza, nombre, edadAnios, edadMeses,
             urlFoto, genero, temperamento, historia
         ))
         NOTA: el Response de Supabase incluye el id de la fila creada
               Si usas @Headers({"Prefer: return=representation"}) en
SupabaseApi
               podras obtener el idMascota del response body

PASO 5 - Registrar vacunas marcadas:
         para cada CheckBox marcado:
             repo.registrarVacunaMascota(idMascota, idVacuna, fechaHoy, cb)
```

7.4 Flujo de Agregar/Quitar Favorito (Adoptante)

```
// Al abrir el detalle de una mascota, verificar si ya es favorita:
repo.verificarEsFavorito(idMascota, idAdoptante, new Callback<>() {
    onResponse: {
        if (resp.body().size() > 0) {
            // ya es favorita --> mostrar corazon relleno
            binding.btnFavorito.setIconResource(R.drawable.ic_favorite);
            esFavorita = true;
        } else {
            // no es favorita --> mostrar corazon vacio
            binding.btnFavorito.setIconResource(R.drawable.ic_favorite_border);
            esFavorita = false;
        }
    }
});

// Al tocar el boton de favorito:
binding.btnFavorito.setOnClickListener(v -> {
    if (esFavorita) {
        repo.eliminarFavorito(idMascota, idAdoptante, cb);
        // El trigger en Supabase reduce ContadorFavoritos automaticamente
    } else {
        repo.agregarFavorito(idMascota, idAdoptante, cb);
        // El trigger en Supabase suma a ContadorFavoritos automaticamente
    }
    esFavorita = !esFavorita;
    actualizarIconoFavorito();
});
```

7.5 Flujo de Solicitud de Adopcion

```
// Desde el detalle de mascota, el adoptante envia solicitud:

// Necesitamos: idMascota (del objeto MascotaResponse actual)
//              idRefugio (de MascotaResponse.idRefugio)
//              idAdoptante (consultar al cargar la pantalla)

// 1. Obtener idAdoptante al iniciar el fragment:
repo.obtenerAdoptantePorUuid(session.getUuid(), cb)
    --> idAdoptante = list.get(0).idAdoptante

// 2. Al tocar 'Enviar solicitud':
repo.crearSolicitud(
    mascota.idRefugio,   // de donde vino la mascota
    mascota.idMascota,   // cual mascota
```

```
    idAdoptante,         // quien solicita
    mensajeEditText.getText().toString()
, cb);

// El trigger trg_solicitud_aprobada cambiara la mascota a 'En Proceso'
// cuando el refugio apruebe la solicitud.
```

8. Supabase Storage - Manejo de Imagenes

8.1 Buckets del Proyecto

Bucket	Visibilidad	Que guarda	Estructura de paths
avatars	Publico	Fotos de perfil de todos los usuarios	avatars/{uuid}/profile.jpg
refugio-covers	Publico	Fotos de portada de refugios	refugio-covers/{uuid}/cover.jpg
mascotas	Publico	Fotos de mascotas	mascotas/{uuid}/{timestamp}.jpg

Todos los buckets son publicos
Al ser publicos, la URL de cualquier imagen se puede construir directamente sin necesidad de Signed URLs. Formato: SUPABASE_URL + '/storage/v1/object/public/' + rutaRelativa

8.2 Construccion de URLs Publicas

```
// URL base del proyecto (definida en BuildConfig):
// SUPABASE_URL = 'https://xxxx.supabase.co'

// Foto de perfil de usuario:
String urlPerfil = BuildConfig.SUPABASE_URL
    + "/storage/v1/object/public/avatars/"
    + uuid + "/profile.jpg";

// Foto de portada de refugio:
String urlPortada = BuildConfig.SUPABASE_URL
    + "/storage/v1/object/public/refugio-covers/"
    + uuid + "/cover.jpg";

// Foto de mascota:
String urlMascota = BuildConfig.SUPABASE_URL
    + "/storage/v1/object/public/mascotas/"
    + uuid + "/" + timestamp + ".jpg";

// Cargar con ImageLoader (bucket publico):
ImageLoader.cargarPublica(context, url, imageView, R.drawable.placeholder);
```

8.3 Como Subir una Imagen (StorageClient)

```
// 1. Comprimir la imagen seleccionada (Uri del gallery):
byte[] bytes = ImageHelper.comprimirImagen(context, imageUri, 80);
//   80 = calidad JPEG (0-100). 80 es un buen balance calidad/tamano

// 2. Subir al bucket:
StorageClient.uploadImage(token, 'avatars', uuid + '/profile.jpg', bytes,
    new Callback<Void>() {
        onResponse: {
            // exito - construir URL y guardarla en base de datos
            String url = BuildConfig.SUPABASE_URL
                + '/storage/v1/object/public/avatars/'
                + uuid + '/profile.jpg';
            // actualizar usuario.urlimagenusuario con esta URL
        }
```

```
        onFailure: { /* manejar error */ }
    }
);
```

9. SupabaseApi.java - Como Funciona PostgREST

Supabase expone la base de datos como una API REST automaticamente mediante PostgREST. Cada tabla y vista tiene un endpoint. Los parametros se pasan como query params con una sintaxis especial.

9.1 Sintaxis de Filtros PostgREST

Operador	Ejemplo de valor	SQL equivalente
eq. (igual)	"eq.Disponible"	WHERE estado = 'Disponible'
ilike. (busqueda parcial, case-insensitive)	"ilike.*luna*"	WHERE nombre ILIKE '%luna%'
gt. (mayor que)	"gt.0"	WHERE campo > 0
order (ordenar)	"fecharegistro.desc"	ORDER BY fecharegistro DESC
limit (limitar)	"20"	LIMIT 20
offset (paginar)	"40"	OFFSET 40 (pagina 3 de 20 en 20)
select (columnas)	"id,nombre,raza"	SELECT id, nombre, raza

9.2 Headers Necesarios en Cada Llamada

SupabaseClient.java configura estos headers automaticamente usando OkHttp Interceptor:

```
// SupabaseClient.java agrega estos headers a TODAS las requests:
request.addHeader("apikey", BuildConfig.SUPABASE_KEY);
request.addHeader("Authorization", "Bearer " + token);
request.addHeader("Content-Type", "application/json");
request.addHeader("Accept", "application/json");

// Para operaciones que deben retornar el objeto creado (obtener el ID):
// Agregar en la anotacion del metodo en SupabaseApi.java:
@Headers({"Prefer: return=representation"})
@POST("rest/v1/mascota")
Call<List<MascotaResponse>> crearMascotaConRetorno(@Body MascotaRequest req);
// Esto devuelve List<MascotaResponse> con el id_mascota recien creado
```

9.3 Como Obtener el ID de un Registro Recien Creado

Problema comun: necesito el id de la mascota que acabo de crear

Por defecto, un POST a Supabase devuelve 201 Created con body vacio. Para obtener el ID del registro creado, agrega el header 'Prefer: return=representation' al endpoint. Esto hace que Supabase devuelva el objeto completo incluyendo el id generado.

```
// En SupabaseApi.java - version que retorna el objeto creado:
@Headers({"Prefer: return=representation"})
@POST("rest/v1/mascota")
Call<List<MascotaResponse>> crearMascotaYObtenerID(@Body MascotaRequest req);

// En AppRepository.java:
public void crearMascotaConId(MascotaRequest req,
Callback<List<MascotaResponse>> cb) {
    api.crearMascotaYObtenerID(req).enqueue(cb);
}

// Uso en la Activity:
repo.crearMascotaConId(mascotaReq, new Callback<List<MascotaResponse>>() {
    @Override
    public void onResponse(...) {
        if (resp.isSuccessful() && resp.body() != null) {
            int idNuevo = resp.body().get(0).idMascota;
            guardarVacunas(idNuevo, vacunasSeleccionadas); // PASO 5
        }
    }
});
```

10. Regla del Null Check - Evitar Crashes

Error mas comun en el proyecto: NullPointerException en callbacks de Retrofit
Cuando el usuario navega rapido entre tabs, el Fragment se destruye pero el callback de red puede llegar despues. Si intentas acceder a binding.algoText desde el callback y binding ya es null, la app crashea. La solucion es siempre verificar al inicio de cada callback.

10.1 Patron Correcto Para Todos los Callbacks

```
// PATRON OBLIGATORIO en onResponse y onFailure de todos los fragments:

new Callback<List<MascotaResponse>>() {
    @Override
    public void onResponse(Call<List<MascotaResponse>> call,
                           Response<List<MascotaResponse>> resp) {

        // === LINEA CRITICA: verificar antes de tocar cualquier View ===
        if (binding == null || !isAdded()) return;
        // ================================================================

        binding.progressBar.setVisibility(View.GONE);
        if (resp.isSuccessful() && resp.body() != null) {
            // usar resp.body() con seguridad
        }
    }

    @Override
    public void onFailure(Call<List<MascotaResponse>> call, Throwable t) {
        if (binding == null || !isAdded()) return;  // mismo chequeo
        binding.progressBar.setVisibility(View.GONE);
    }
};

// Y SIEMPRE en onDestroyView():
@Override
public void onDestroyView() {
    super.onDestroyView();
    binding = null;  // <- esto es lo que previene el crash
}
```

11. Resumen: Como Obtener Cada ID

Esta es la pregunta mas frecuente del equipo. Aqui esta la respuesta completa:

Necesito...	Lo obtengo de...	Cuando obtenerlo
UUID del usuario (String)	session.getUuid() - SessionManager	Siempre disponible tras login
Token JWT (String)	session.getToken() - SessionManager	Siempre disponible tras login. Pasarlo a AppRepository
Rol del usuario	session.getRol() - SessionManager	Siempre disponible. Usar para mostrar/ocultar UI
id_adoptante (int)	repo.obtenerAdoptanteCompleto(uuid) -> list.get(0).idAdoptante	Al inicio de pantallas del adoptante. Guardar en variable
id_refugio (int)	repo.obtenerRefugioPorUuid(uuid) -> list.get(0).idRefugio	Al inicio de pantallas del refugio. Guardar en variable
id_mascota (int)	Viene en cada MascotaResponse.idMascota	Al recibir lista de mascotas o al crear una con return=representation
id_raza (int)	repo.obtenerRazasPorEspecie(idEspecie) -> spinner -> razaSeleccionada.idRaza	Al llenar formulario de mascota
id_especie (int)	repo.obtenerEspecies() -> spinner -> especieSeleccionada.idEspecie	Al llenar formulario de mascota
id_solicitud (int)	Viene en SolicitudResponse.idSolicitud	Al listar solicitudes. Necesario para aprobar/rechazar
id_favorito (int)	Viene en FavoritoResponse.idFavorito	Al listar favoritos. Se puede usar para eliminar