GanaderaSoft

Documentación Técnica Consolidada

Sistema de gestión integral para fincas ganaderas  
Desarrollado en Flutter

# Índice de Contenidos

*Esta sección contiene el índice de la documentación consolidada.*

1. 1. Visión General
2. 2. Arquitectura
3. 3. Estrategia Offline
4. 4. Modulos
5. 5. Base Datos
6. 6. Api Servicios
7. 7. Testing
8. 8. Configuracion

# GanaderaSoft - Documentación

Sistema de gestión integral para fincas ganaderas desarrollado en Flutter.

## Índice de Documentación

### 1. [Arquitectura de la Aplicación](./arquitectura.md)

Descripción detallada de la arquitectura del sistema, módulos principales y su interacción.

### 2. [Estrategia Offline](./estrategia-offline.md)

Documentación completa de la implementación offline, incluyendo diagramas de secuencia y funcionalidades soportadas.

### 3. [Módulos y Funcionalidades](./modulos.md)

Detalle de cada módulo de la aplicación y sus funcionalidades específicas.

### 4. [Base de Datos](./base-datos.md)

Estructura de la base de datos local SQLite, migraciones y modelos.

### 5. [API y Servicios](./api-servicios.md)

Documentación de endpoints de API y servicios de conectividad.

### 6. [Testing](./testing.md)

Estrategia de testing y documentación de pruebas implementadas.

### 7. [Configuración y Deployment](./configuracion.md)

Guías de configuración y despliegue de la aplicación.

## Información General

**Nombre:** GanaderaSoft

**Versión:** 0.1.0

**Framework:** Flutter 3.8.1+

**Plataformas:** Android, iOS, Web, Linux, macOS, Windows

**Arquitectura:** Cliente-Servidor con soporte offline completo

## Características Principales

* ✅ \*\*Gestión Integral de Fincas\*\*: Administración completa de fincas ganaderas
* ✅ \*\*Soporte Offline Completo\*\*: Funcionalidad completa sin conexión a internet
* ✅ \*\*Sincronización Inteligente\*\*: Sincronización manual de datos con el servidor
* ✅ \*\*Gestión de Animales\*\*: Registro, seguimiento y administración de ganado
* ✅ \*\*Gestión de Personal\*\*: Administración del personal de la finca
* ✅ \*\*Registros de Producción\*\*: Seguimiento de producción lechera y cambios corporales
* ✅ \*\*Configuración Flexible\*\*: Sistema de configuración adaptable a diferentes tipos de explotación

## Tecnologías Utilizadas

* \*\*Flutter SDK\*\*: ^3.8.1
* \*\*Dart\*\*: Lenguaje de programación principal
* \*\*SQLite\*\*: Base de datos local para soporte offline
* \*\*HTTP\*\*: Comunicación con API REST
* \*\*Shared Preferences\*\*: Almacenamiento de configuración local
* \*\*Connectivity Plus\*\*: Monitoreo de conectividad
* \*\*Provider\*\*: Gestión de estado
* \*\*Crypto\*\*: Seguridad y hash de contraseñas

## Estructura del Proyecto

lib/  
├── config/ # Configuración de la aplicación  
├── constants/ # Constantes globales  
├── models/ # Modelos de datos  
├── screens/ # Interfaces de usuario  
├── services/ # Servicios y lógica de negocio  
├── theme/ # Configuración de temas  
└── main.dart # Punto de entrada de la aplicación  
  
docs/ # Documentación  
├── README.md # Este archivo  
├── arquitectura.md # Documentación de arquitectura  
├── estrategia-offline.md # Documentación offline  
├── modulos.md # Documentación de módulos  
├── base-datos.md # Documentación de base de datos  
├── api-servicios.md # Documentación de API  
├── testing.md # Documentación de testing  
└── configuracion.md # Documentación de configuración

## Inicio Rápido

Para más información sobre instalación, configuración y uso, consulte la documentación específica en cada sección.

---

*Última actualización: $(date '+%Y-%m-%d')*

# Arquitectura de la Aplicación

## Visión General

GanaderaSoft está construida con una arquitectura modular que separa claramente las responsabilidades y permite un mantenimiento eficiente. La aplicación sigue el patrón **Model-View-Service (MVS)** con soporte completo para operaciones offline.

## Diagrama de Arquitectura General

Diagrama #1:

graph TB  
 subgraph "Capa de Presentación"  
 UI[Interfaces de Usuario - Screens]  
 T[Themes & Constants]  
 end  
   
 subgraph "Capa de Lógica de Negocio"  
 AS[Auth Service]  
 CS[Configuration Service]  
 DS[Database Service]  
 SS[Sync Service]  
 CNS[Connectivity Service]  
 LS[Logging Service]  
 OM[Offline Manager]  
 end  
   
 subgraph "Capa de Datos"  
 M[Models]  
 SQLITE[SQLite Database]  
 SP[Shared Preferences]  
 end  
   
 subgraph "Servicios Externos"  
 API[REST API Server]  
 NET[Internet Connectivity]  
 end  
   
 UI --> AS  
 UI --> CS  
 UI --> DS  
 UI --> SS  
   
 AS --> M  
 CS --> M  
 DS --> SQLITE  
 SS --> API  
 SS --> DS  
   
 CNS --> NET  
 OM --> CNS  
   
 AS --> SP  
 CS --> SP  
   
 style UI fill:#e1f5fe  
 style AS fill:#f3e5f5  
 style CS fill:#f3e5f5  
 style DS fill:#f3e5f5  
 style SS fill:#f3e5f5  
 style SQLITE fill:#fff3e0  
 style API fill:#ffebee

## Capas de la Arquitectura

### 1. Capa de Presentación

**Ubicación:** lib/screens/, lib/theme/, lib/constants/

Contiene todas las interfaces de usuario y elementos visuales:

* \*\*Screens\*\*: Pantallas principales de la aplicación
* \*\*Themes\*\*: Configuración de temas y estilos
* \*\*Constants\*\*: Constantes de la aplicación (textos, configuraciones UI)

### 2. Capa de Lógica de Negocio

**Ubicación:** lib/services/

Contiene toda la lógica de negocio y servicios principales:

#### 2.1 Auth Service (`auth\_service.dart`)

* Gestión de autenticación y autorización
* Manejo de tokens JWT
* Autenticación offline con credenciales hash
* Gestión de sesiones de usuario

#### 2.2 Database Service (`database\_service.dart`)

* Gestión de la base de datos SQLite local
* Operaciones CRUD offline
* Migraciones de esquema
* Sincronización de datos

#### 2.3 Configuration Service (`configuration\_service.dart`)

* Gestión de datos de configuración
* Caché de configuraciones offline
* Actualización de configuraciones del servidor

#### 2.4 Sync Service (`sync\_service.dart`)

* Sincronización de datos con el servidor
* Gestión de conflictos
* Operaciones pendientes offline
* Estado de sincronización

#### 2.5 Connectivity Service (`connectivity\_service.dart`)

* Monitoreo de conectividad de red
* Detección de estado online/offline
* Verificación de accesibilidad del servidor

#### 2.6 Offline Manager (`offline\_manager.dart`)

* Coordinación del modo offline
* Gestión de transiciones online/offline
* Monitoreo de conectividad

#### 2.7 Logging Service (`logging\_service.dart`)

* Sistema de logging unificado
* Diferentes niveles de log (debug, info, warning, error)
* Trazabilidad de operaciones

### 3. Capa de Datos

**Ubicación:** lib/models/, Base de datos SQLite, Shared Preferences

#### 3.1 Models

* \*\*User\*\*: Modelo de usuario
* \*\*Finca\*\*: Modelo de finca
* \*\*Animal\*\*: Modelo de animal
* \*\*Farm Management Models\*\*: Modelos de gestión de finca
* \*\*Configuration Models\*\*: Modelos de configuración
* \*\*Pending Sync Models\*\*: Modelos para sincronización pendiente

#### 3.2 Almacenamiento Local

* \*\*SQLite\*\*: Base de datos principal para datos offline
* \*\*Shared Preferences\*\*: Configuración y tokens

### 4. Servicios Externos

* \*\*REST API Server\*\*: Servidor backend con endpoints REST
* \*\*Internet Connectivity\*\*: Conectividad de red

## Flujo de Datos

### Operación Online

Diagrama #2:

sequenceDiagram  
 participant UI as Screen  
 participant S as Service  
 participant API as REST API  
 participant DB as SQLite  
   
 UI->>S: Solicita operación  
 S->>API: Petición HTTP  
 API-->>S: Respuesta JSON  
 S->>DB: Guarda en cache local  
 S-->>UI: Retorna datos

### Operación Offline

Diagrama #3:

sequenceDiagram  
 participant UI as Screen  
 participant S as Service  
 participant DB as SQLite  
 participant PS as Pending Sync  
   
 UI->>S: Solicita operación  
 S->>DB: Consulta datos locales  
 DB-->>S: Datos en cache  
 S->>PS: Marca operación pendiente  
 S-->>UI: Retorna datos locales

## Principios de Diseño

### 1. Separación de Responsabilidades

* Cada servicio tiene una responsabilidad específica
* Las screens solo manejan lógica de presentación
* Los modelos son entidades de datos puras

### 2. Inversión de Dependencias

* Los servicios dependen de abstracciones, no de implementaciones concretas
* Facilita testing y mantenimiento

### 3. Offline-First

* Todas las operaciones funcionan offline
* Los datos se sincronizan cuando hay conectividad
* La experiencia de usuario es consistente

### 4. Configurabilidad

* Endpoints de API configurables
* Timeouts configurables
* Diferentes niveles de logging

## Gestión de Estado

La aplicación utiliza el patrón **Service Locator** donde:

1. Los servicios son singleton
2. Las screens acceden a los servicios directamente
3. Los cambios de estado se notifican a través de callbacks
4. El estado offline/online se mantiene globalmente

## Seguridad

### Autenticación

* Tokens JWT para autenticación con el servidor
* Hash SHA-256 de contraseñas para autenticación offline
* Almacenamiento seguro en Shared Preferences

### Datos

* Validación de entrada en todos los formularios
* Sanitización de datos antes de almacenamiento
* Encriptación de datos sensibles

## Escalabilidad

### Horizontal

* Nuevos módulos pueden agregarse fácilmente
* Nuevos servicios siguen el mismo patrón
* API REST permite escalado del backend

### Vertical

* Base de datos SQLite puede manejar grandes volúmenes
* Sincronización por lotes para eficiencia
* Cache inteligente para reducir uso de memoria

---

*Siguiente: [Estrategia Offline](./estrategia-offline.md)*

# Estrategia Offline

## Visión General

GanaderaSoft implementa una estrategia **offline-first** que permite a los usuarios trabajar completamente sin conexión a internet. Todos los datos se almacenan localmente en SQLite y se sincronizan con el servidor cuando hay conectividad disponible.

## Características de la Implementación Offline

### ✅ Funcionalidades Offline Completas

* Autenticación con credenciales almacenadas localmente
* Creación, edición y eliminación de registros
* Consulta de todos los datos desde cache local
* Gestión completa de fincas, animales y personal
* Registros de producción (leche, peso corporal, lactancia)
* Datos de configuración en cache

### ✅ Sincronización Inteligente

* Sincronización manual a través del botón "Sincronizar cambios"
* Detección automática de conflictos
* Preservación de datos modificados offline
* Sincronización por lotes para eficiencia

## Arquitectura de Datos Offline

### Base de Datos SQLite Local

Diagrama #1:

erDiagram  
 users ||--o{ fincas : "propietario"  
 fincas ||--o{ rebanos : "contiene"  
 rebanos ||--o{ animales : "agrupa"  
 fincas ||--o{ personal\_finca : "emplea"  
 animales ||--o{ cambios\_animal : "tiene"  
 animales ||--o{ peso\_corporal : "registra"  
 animales ||--o{ lactancia : "produce"  
   
 users {  
 int id PK  
 string name  
 string email  
 string type\_user  
 string password\_hash  
 int modifiedOffline  
 int synced  
 }  
   
 fincas {  
 int id\_finca PK  
 int id\_propietario FK  
 string nombre  
 string explotacion\_tipo  
 int modifiedOffline  
 timestamp local\_updated\_at  
 }  
   
 animales {  
 int id\_animal PK  
 int id\_rebano FK  
 string nombre  
 string codigo\_animal  
 int modifiedOffline  
 int is\_pending  
 string pending\_operation  
 }  
   
 cambios\_animal {  
 int id\_cambio PK  
 string fecha\_cambio  
 real peso  
 real altura  
 int is\_pending  
 string pending\_operation  
 }  
   
 peso\_corporal {  
 int id\_peso PK  
 real peso  
 date fecha\_registro  
 int is\_pending  
 string pending\_operation  
 }  
   
 lactancia {  
 int lactancia\_id PK  
 date lactancia\_fecha\_inicio  
 date lactancia\_fecha\_fin  
 int is\_pending  
 string pending\_operation  
 }

### Columnas de Control Offline

Cada tabla tiene columnas específicas para el manejo offline:

* \*\*`modifiedOffline`\*\*: Indica si el registro fue modificado offline
* \*\*`synced`\*\*: Indica si el registro está sincronizado con el servidor
* \*\*`is\_pending`\*\*: Indica si hay operaciones pendientes de sincronización
* \*\*`pending\_operation`\*\*: Tipo de operación pendiente (CREATE, UPDATE, DELETE)
* \*\*`local\_updated\_at`\*\*: Timestamp de última modificación local

## Flujos de Operación Offline

### 1. Autenticación Offline

Diagrama #2:

sequenceDiagram  
 participant U as Usuario  
 participant LS as Login Screen  
 participant AS as Auth Service  
 participant DB as SQLite  
 participant SP as Shared Preferences  
   
 U->>LS: Ingresa credenciales  
 LS->>AS: login(email, password)  
 AS->>AS: hashPassword(password)  
 AS->>DB: Busca usuario con email  
 DB-->>AS: Datos de usuario  
 AS->>AS: Compara hash de contraseña  
   
 alt Credenciales válidas  
 AS->>SP: Guarda token offline  
 AS->>LS: Usuario autenticado  
 LS->>U: Acceso a la aplicación  
 else Credenciales inválidas  
 AS->>LS: Error de autenticación  
 LS->>U: Mensaje de error  
 end

### 2. Creación de Registros Offline

Diagrama #3:

sequenceDiagram  
 participant U as Usuario  
 participant CS as Create Screen  
 participant DS as Database Service  
 participant DB as SQLite  
 participant PSS as Pending Sync Service  
   
 U->>CS: Completa formulario  
 CS->>DS: savePendingXxxOffline(data)  
 DS->>DS: Genera ID temporal negativo  
 DS->>DB: INSERT con is\_pending=1, pending\_operation='CREATE'  
 DB-->>DS: Registro guardado  
 DS->>PSS: Registra operación pendiente  
 DS-->>CS: Éxito con ID temporal  
 CS->>U: Confirmación de guardado offline

### 3. Edición de Registros Offline

Diagrama #4:

sequenceDiagram  
 participant U as Usuario  
 participant ES as Edit Screen  
 participant DS as Database Service  
 participant DB as SQLite  
 participant PSS as Pending Sync Service  
   
 U->>ES: Modifica datos  
 ES->>DS: savePendingXxxUpdateOffline(id, data)  
 DS->>DB: UPDATE con modifiedOffline=1, is\_pending=1  
 DS->>DB: SET pending\_operation='UPDATE'  
 DB-->>DS: Registro actualizado  
 DS->>PSS: Registra operación pendiente  
 DS-->>ES: Éxito  
 ES->>U: Confirmación de edición offline

### 4. Consulta de Datos Offline

Diagrama #5:

sequenceDiagram  
 participant U as Usuario  
 participant LS as List Screen  
 participant CS as Configuration Service  
 participant CNS as Connectivity Service  
 participant DS as Database Service  
 participant DB as SQLite  
   
 U->>LS: Solicita listado  
 LS->>CS: getXxxData()  
 CS->>CNS: isConnected()  
 CNS-->>CS: false (offline)  
 CS->>DS: getXxxOffline()  
 DS->>DB: SELECT \* FROM xxx  
 DB-->>DS: Datos locales  
 DS-->>CS: Lista de registros  
 CS-->>LS: Datos con indicador offline  
 LS->>U: Muestra datos + banner offline

### 5. Sincronización Manual

Diagrama #6:

sequenceDiagram  
 participant U as Usuario  
 participant PSS as Pending Sync Screen  
 participant SS as Sync Service  
 participant API as REST API  
 participant DS as Database Service  
 participant DB as SQLite  
   
 U->>PSS: Click "Sincronizar cambios"  
 PSS->>SS: syncPendingData()  
 SS->>DS: getPendingRecords()  
 DS->>DB: SELECT WHERE is\_pending=1  
 DB-->>DS: Registros pendientes  
 DS-->>SS: Lista de operaciones pendientes  
   
 loop Para cada operación pendiente  
 alt CREATE operation  
 SS->>API: POST /api/endpoint  
 API-->>SS: Nuevo ID del servidor  
 SS->>DS: markAsSynced(tempId, realId)  
 DS->>DB: UPDATE con ID real, is\_pending=0  
 else UPDATE operation  
 SS->>API: PUT /api/endpoint/id  
 API-->>SS: Confirmación  
 SS->>DS: markUpdateAsSynced(id)  
 DS->>DB: SET is\_pending=0, modifiedOffline=0  
 else DELETE operation  
 SS->>API: DELETE /api/endpoint/id  
 API-->>SS: Confirmación  
 SS->>DS: removeRecord(id)  
 DS->>DB: DELETE FROM table  
 end  
 end  
   
 SS-->>PSS: Sincronización completa  
 PSS->>U: Mensaje de éxito

## Gestión de Estados de Conectividad

### Detección de Estado Offline

Diagrama #7:

stateDiagram-v2  
 [\*] --> Checking  
 Checking --> Online : Internet available + Server reachable  
 Checking --> Offline : No internet OR Server unreachable  
   
 Online --> Offline : Connection lost  
 Offline --> Online : Connection restored  
   
 Online : App uses server data  
 Online : Auto-cache for offline use  
   
 Offline : App uses local cache  
 Offline : All operations stored locally  
 Offline : Sync disabled

### Indicadores Visuales de Estado

1. \*\*Banner Offline\*\*: Se muestra en todas las pantallas cuando no hay conectividad
2. \*\*Indicador en AppBar\*\*: Muestra "Offline" en la barra superior
3. \*\*Mensajes Contextuales\*\*: Informan sobre operaciones offline
4. \*\*Pantalla de Registros Pendientes\*\*: Lista todas las operaciones por sincronizar

## Estrategia de Sincronización

### Principios de Sincronización

1. \*\*Manual Only\*\*: No hay sincronización automática
2. \*\*Batch Operations\*\*: Las operaciones se sincronizan en lotes
3. \*\*Conflict Detection\*\*: Se detectan y manejan conflictos
4. \*\*Data Preservation\*\*: Los datos offline nunca se pierden
5. \*\*Atomic Operations\*\*: Cada operación de sync es atómica

### Orden de Sincronización

Diagrama #8:

graph TD  
 A[Iniciar Sincronización] --> B[Sincronizar Configuraciones]  
 B --> C[Sincronizar Fincas]  
 C --> D[Sincronizar Rebaños]  
 D --> E[Sincronizar Animales]  
 E --> F[Sincronizar Personal]  
 F --> G[Sincronizar Registros de Gestión]  
 G --> H[Finalizar Sincronización]  
   
 style A fill:#e8f5e8  
 style H fill:#e8f5e8  
 style B,C,D,E,F,G fill:#fff3cd

### Manejo de Errores de Sincronización

Diagrama #9:

graph TD  
 A[Operación de Sync] --> B{Error de Red?}  
 B -->|Sí| C[Reintentar después]  
 B -->|No| D{Error del Servidor?}  
 D -->|Sí| E[Marcar como fallido]  
 D -->|No| F{Conflicto de Datos?}  
 F -->|Sí| G[Resolver conflicto]  
 F -->|No| H[Marcar como sincronizado]  
   
 C --> I[Mantener en cola]  
 E --> I  
 G --> I  
 H --> J[Remover de cola]  
   
 style A fill:#e8f5e8  
 style J fill:#e8f5e8  
 style I fill:#ffebee

## Funcionalidades Offline Soportadas

### ✅ Completamente Offline

* \*\*Autenticación\*\*: Login con credenciales hash locales
* \*\*Gestión de Fincas\*\*: CRUD completo
* \*\*Gestión de Rebaños\*\*: CRUD completo
* \*\*Gestión de Animales\*\*: CRUD completo
* \*\*Personal de Finca\*\*: CRUD completo
* \*\*Cambios de Animales\*\*: Creación y consulta
* \*\*Peso Corporal\*\*: Creación y consulta
* \*\*Lactancia\*\*: Creación y consulta
* \*\*Registros de Leche\*\*: Consulta de datos en cache

### ⚠️ Requiere Sincronización

* \*\*Configuraciones del Sistema\*\*: Actualización desde servidor
* \*\*Datos de Nuevos Usuarios\*\*: Registro de nuevos usuarios
* \*\*Reportes Globales\*\*: Análisis que requieren datos del servidor

## Beneficios de la Estrategia Offline

1. \*\*Disponibilidad Total\*\*: La aplicación funciona sin conexión
2. \*\*Experiencia Fluida\*\*: No hay interrupciones por problemas de red
3. \*\*Productividad\*\*: Los usuarios pueden trabajar en cualquier lugar
4. \*\*Confiabilidad\*\*: Los datos nunca se pierden
5. \*\*Flexibilidad\*\*: Sincronización cuando es conveniente para el usuario

---

*Siguiente: [Módulos y Funcionalidades](./modulos.md)*

# Módulos y Funcionalidades

## Visión General

GanaderaSoft está organizada en módulos funcionales que cubren todos los aspectos de la gestión de fincas ganaderas. Cada módulo tiene responsabilidades específicas y trabaja de manera coordinada con otros módulos.

## Diagrama de Módulos

Diagrama #1:

graph TB  
 subgraph "Módulo de Autenticación"  
 AUTH[Auth Service]  
 LOGIN[Login Screen]  
 PROFILE[Profile Screen]  
 end  
   
 subgraph "Módulo de Gestión de Fincas"  
 FINCA[Finca Service]  
 FINCA\_LIST[Finca List Screen]  
 FINCA\_ADMIN[Finca Admin Screen]  
 end  
   
 subgraph "Módulo de Gestión de Ganado"  
 ANIMAL[Animal Service]  
 ANIMAL\_LIST[Animals List Screen]  
 ANIMAL\_CREATE[Create Animal Screen]  
 ANIMAL\_EDIT[Edit Animal Screen]  
 REBANO[Rebaño Service]  
 REBANO\_LIST[Rebaños List Screen]  
 end  
   
 subgraph "Módulo de Personal"  
 PERSONAL[Personal Service]  
 PERSONAL\_LIST[Personal List Screen]  
 PERSONAL\_CREATE[Create Personal Screen]  
 end  
   
 subgraph "Módulo de Registros de Producción"  
 CAMBIOS[Cambios Animal]  
 PESO[Peso Corporal]  
 LACTANCIA[Lactancia]  
 LECHE[Registro Leche]  
 end  
   
 subgraph "Módulo de Configuración"  
 CONFIG[Configuration Service]  
 CONFIG\_SCREEN[Configuration Screen]  
 DATA\_CONFIG[Configuration Data Screen]  
 end  
   
 subgraph "Módulo de Sincronización"  
 SYNC[Sync Service]  
 PENDING[Pending Sync Screen]  
 OFFLINE[Offline Manager]  
 end  
   
 AUTH --> FINCA  
 FINCA --> ANIMAL  
 FINCA --> PERSONAL  
 ANIMAL --> CAMBIOS  
 ANIMAL --> PESO  
 ANIMAL --> LACTANCIA  
 CONFIG --> AUTH  
 CONFIG --> FINCA  
 CONFIG --> ANIMAL  
 SYNC --> AUTH  
 SYNC --> FINCA  
 SYNC --> ANIMAL  
 SYNC --> PERSONAL  
   
 style AUTH fill:#e3f2fd  
 style FINCA fill:#f3e5f5  
 style ANIMAL fill:#e8f5e8  
 style PERSONAL fill:#fff3e0  
 style CAMBIOS fill:#fce4ec  
 style CONFIG fill:#f1f8e9  
 style SYNC fill:#e0f2f1

## 1. Módulo de Autenticación

### Responsabilidades

* Gestión de login/logout de usuarios
* Autenticación offline con credenciales hash
* Gestión de sesiones y tokens
* Perfil de usuario

### Componentes Principales

#### `AuthService` (`lib/services/auth\_service.dart`)

* \*\*Login online/offline\*\*: Autenticación con servidor o credenciales locales
* \*\*Gestión de tokens\*\*: JWT para servidor, hash local para offline
* \*\*Gestión de sesiones\*\*: Mantenimiento de estado de autenticación
* \*\*Hash de contraseñas\*\*: SHA-256 para seguridad offline

#### `LoginScreen` (`lib/screens/login\_screen.dart`)

* Interfaz de autenticación
* Validación de credenciales
* Navegación post-login

#### `ProfileScreen` (`lib/screens/profile\_screen.dart`)

* Visualización de datos del usuario
* Edición de perfil
* Configuraciones de usuario

### Funcionalidades

* ✅ Login con email y contraseña
* ✅ Autenticación offline
* ✅ Gestión de sesiones persistentes
* ✅ Logout seguro
* ✅ Visualización y edición de perfil

## 2. Módulo de Gestión de Fincas

### Responsabilidades

* Administración de fincas ganaderas
* Gestión de propietarios
* Configuración de tipos de explotación
* Navegación entre fincas

### Componentes Principales

#### Servicios

* \*\*AuthService\*\*: Gestión de fincas a través de endpoints
* \*\*DatabaseService\*\*: Almacenamiento offline de fincas

#### Pantallas

* \*\*`FincasScreen`\*\* (`lib/screens/finca\_list\_screen.dart`): Listado de fincas
* \*\*`FincaAdministracionScreen`\*\* (`lib/screens/finca\_administracion\_screen.dart`): Panel de administración

### Funcionalidades

* ✅ Listado de fincas del usuario
* ✅ Navegación a gestión de finca específica
* ✅ Visualización de datos de propietario
* ✅ Acceso a módulos de la finca (animales, personal, etc.)
* ✅ Funcionamiento offline completo

## 3. Módulo de Gestión de Ganado

### Responsabilidades

* Registro y gestión de animales
* Organización en rebaños
* Seguimiento individual de animales
* Gestión de razas y características

### Componentes Principales

#### Servicios

* \*\*AuthService\*\*: APIs para animales y rebaños
* \*\*DatabaseService\*\*: Almacenamiento offline
* \*\*ConfigurationService\*\*: Datos de configuración (razas, etc.)

#### Pantallas de Animales

* \*\*`AnimalesListScreen`\*\*: Listado de animales con filtros
* \*\*`CreateAnimalScreen`\*\*: Creación de nuevos animales
* \*\*`EditAnimalScreen`\*\*: Edición de animales existentes

#### Pantallas de Rebaños

* \*\*`RebanosListScreen`\*\*: Gestión de rebaños
* \*\*`CreateRebanoScreen`\*\*: Creación de rebaños

### Funcionalidades

#### Gestión de Animales

* ✅ Registro de animales con datos completos
* ✅ Edición offline de información
* ✅ Filtros por rebaño, sexo, estado de salud
* ✅ Búsqueda por nombre o código
* ✅ Asignación a rebaños
* ✅ Gestión de razas y composición racial

#### Gestión de Rebaños

* ✅ Creación y edición de rebaños
* ✅ Asignación de animales
* ✅ Organización por propósito

### Modelo de Datos

Diagrama #2:

erDiagram  
 rebanos {  
 int id\_rebano PK  
 int id\_finca FK  
 string nombre  
 string proposito  
 int cantidad\_animales  
 }  
   
 animales {  
 int id\_animal PK  
 int id\_rebano FK  
 string nombre  
 string codigo\_animal  
 string sexo  
 date fecha\_nacimiento  
 string procedencia  
 int fk\_composicion\_raza  
 int estado\_id  
 int etapa\_id  
 }  
   
 composicion\_raza {  
 int id\_composicion PK  
 string nombre  
 string siglas  
 string pelaje  
 string proposito  
 }  
   
 rebanos ||--o{ animales : contiene  
 animales }o--|| composicion\_raza : pertenece\_a

## 4. Módulo de Personal

### Responsabilidades

* Gestión del personal de la finca
* Registro de trabajadores
* Información de contacto y roles

### Componentes Principales

#### Servicios

* \*\*AuthService\*\*: APIs de personal de finca
* \*\*DatabaseService\*\*: Almacenamiento offline

#### Pantallas

* \*\*`PersonalFincaListScreen`\*\*: Listado del personal
* \*\*`CreatePersonalFincaScreen`\*\*: Registro de nuevo personal
* \*\*`EditPersonalFincaScreen`\*\*: Edición de información

### Funcionalidades

* ✅ Registro completo de personal
* ✅ Gestión de tipos de trabajador
* ✅ Información de contacto
* ✅ Edición offline
* ✅ Sincronización con servidor

### Modelo de Datos

Diagrama #3:

erDiagram  
 personal\_finca {  
 int id\_tecnico PK  
 int id\_finca FK  
 int cedula  
 string nombre  
 string apellido  
 string telefono  
 string correo  
 string tipo\_trabajador  
 }

## 5. Módulo de Registros de Producción

### Responsabilidades

* Seguimiento de cambios en animales
* Registro de peso corporal
* Gestión de lactancia
* Registros de producción lechera

### Componentes Principales

#### 5.1 Cambios de Animales

**Pantallas:**

* `CambiosAnimalListScreen`: Historial de cambios
* `CreateCambiosAnimalScreen`: Registro de nuevos cambios

**Funcionalidades:**

* ✅ Registro de cambios de etapa
* ✅ Medidas corporales (peso, altura)
* ✅ Comentarios descriptivos
* ✅ Creación offline completa

#### 5.2 Peso Corporal

**Pantallas:**

* `PesoCorporalListScreen`: Historial de pesos
* `CreatePesoCorporalScreen`: Registro de peso

**Funcionalidades:**

* ✅ Registro de peso con fecha
* ✅ Seguimiento histórico
* ✅ Creación offline

#### 5.3 Lactancia

**Pantallas:**

* `LactanciaListScreen`: Períodos de lactancia
* `CreateLactanciaScreen`: Registro de lactancia

**Funcionalidades:**

* ✅ Gestión de períodos de lactancia
* ✅ Fechas de inicio y fin
* ✅ Proceso de secado
* ✅ Creación offline

#### 5.4 Registros de Leche

**Pantallas:**

* `RegistrosLecheListScreen`: Producción lechera
* `CreateRegistroLecheScreen`: Registro de producción

**Funcionalidades:**

* ✅ Registro diario de producción
* ✅ Filtros por fecha y animal
* ✅ Consulta offline de datos en cache

### Flujo de Registros de Producción

Diagrama #4:

graph TD  
 A[Seleccionar Animal] --> B[Elegir Tipo de Registro]  
 B --> C[Cambios Animal]  
 B --> D[Peso Corporal]  
 B --> E[Lactancia]  
 B --> F[Registro Leche]  
   
 C --> G[Completar Formulario]  
 D --> G  
 E --> G  
 F --> G  
   
 G --> H{¿Online?}  
 H -->|Sí| I[Enviar a Servidor]  
 H -->|No| J[Guardar Offline]  
   
 I --> K[Confirmar Guardado]  
 J --> L[Marcar Pendiente Sync]  
   
 style A fill:#e3f2fd  
 style K fill:#e8f5e8  
 style L fill:#fff3e0

## 6. Módulo de Configuración

### Responsabilidades

* Gestión de datos de configuración del sistema
* Cache offline de configuraciones
* Actualización desde servidor

### Componentes Principales

#### `ConfigurationService` (`lib/services/configuration\_service.dart`)

* Gestión de configuraciones del sistema
* Cache offline inteligente
* Sincronización con servidor

#### Pantallas

* \*\*`ConfigurationDataScreen`\*\*: Visualización de configuraciones
* \*\*`SyncScreen`\*\*: Sincronización de configuraciones

### Tipos de Configuración

* \*\*Estados de Salud\*\*: Estados de salud animal
* \*\*Etapas\*\*: Etapas de desarrollo animal
* \*\*Composición Racial\*\*: Razas y características
* \*\*Tipos de Explotación\*\*: Clasificación de fincas
* \*\*Sexos\*\*: Clasificación de sexo animal
* \*\*Fuentes de Agua\*\*: Tipos de agua disponible
* \*\*Métodos de Riego\*\*: Sistemas de riego
* \*\*pH del Suelo\*\*: Clasificación de acidez
* \*\*Textura del Suelo\*\*: Tipos de suelo
* \*\*Tipos de Relieve\*\*: Características geográficas

### Funcionalidades

* ✅ Cache automático de configuraciones
* ✅ Actualización desde servidor cuando hay conectividad
* ✅ Fallback a datos offline
* ✅ Sincronización selectiva

## 7. Módulo de Sincronización

### Responsabilidades

* Gestión de sincronización de datos
* Monitoreo de conectividad
* Gestión de operaciones pendientes

### Componentes Principales

#### `SyncService` (`lib/services/sync\_service.dart`)

* Sincronización manual de datos
* Gestión de operaciones pendientes
* Resolución de conflictos

#### `OfflineManager` (`lib/services/offline\_manager.dart`)

* Monitoreo de conectividad
* Gestión de transiciones online/offline

#### `ConnectivityService` (`lib/services/connectivity\_service.dart`)

* Detección de estado de red
* Verificación de accesibilidad del servidor

#### Pantallas

* \*\*`PendingSyncScreen`\*\*: Lista de operaciones pendientes
* \*\*`SyncScreen`\*\*: Control manual de sincronización

### Funcionalidades

* ✅ Sincronización manual por módulos
* ✅ Vista de operaciones pendientes
* ✅ Indicadores visuales de estado offline
* ✅ Manejo de errores de sincronización
* ✅ Preservación de datos offline

### Flujo de Sincronización

Diagrama #5:

sequenceDiagram  
 participant U as Usuario  
 participant PS as Pending Sync Screen  
 participant SS as Sync Service  
 participant API as Server API  
   
 U->>PS: "Sincronizar cambios"  
 PS->>SS: Inicia sincronización  
 SS->>SS: Obtiene registros pendientes  
   
 loop Para cada registro pendiente  
 SS->>API: Sincroniza registro  
 alt Éxito  
 SS->>SS: Marca como sincronizado  
 else Error  
 SS->>SS: Mantiene como pendiente  
 end  
 end  
   
 SS-->>PS: Resultado de sincronización  
 PS-->>U: Muestra resumen

## Integración entre Módulos

### Flujo de Navegación

Diagrama #6:

graph TD  
 A[Login] --> B[Home Screen]  
 B --> C[Gestión de Fincas]  
 C --> D[Administración de Finca]  
 D --> E[Gestión de Animales]  
 D --> F[Gestión de Personal]  
 D --> G[Registros de Producción]  
   
 E --> H[Crear/Editar Animal]  
 F --> I[Crear/Editar Personal]  
 G --> J[Cambios Animal]  
 G --> K[Peso Corporal]  
 G --> L[Lactancia]  
 G --> M[Registro Leche]  
   
 B --> N[Configuración]  
 B --> O[Registros Pendientes]  
 B --> P[Mi Perfil]  
   
 style A fill:#ffcdd2  
 style B fill:#e1f5fe  
 style N,O,P fill:#f3e5f5

### Dependencias entre Módulos

1. \*\*Autenticación\*\* → Base para todos los módulos
2. \*\*Configuración\*\* → Datos necesarios para otros módulos
3. \*\*Fincas\*\* → Contexto para animales y personal
4. \*\*Animales\*\* → Base para registros de producción
5. \*\*Sincronización\*\* → Coordina todos los módulos

---

*Siguiente: [Base de Datos](./base-datos.md)*

# Base de Datos

## Visión General

GanaderaSoft utiliza **SQLite** como base de datos local para almacenamiento offline. La base de datos está diseñada para replicar la estructura del servidor y mantener sincronización de datos entre el cliente y el backend.

## Arquitectura de Base de Datos

### Características Principales

* \*\*SQLite 3\*\*: Base de datos embebida y ligera
* \*\*Versionado\*\*: Sistema de migraciones automáticas
* \*\*Soporte Offline\*\*: Diseñada para funcionamiento sin conexión
* \*\*Sincronización\*\*: Columnas especiales para manejo de sync
* \*\*Integridad\*\*: Constraints y relaciones definidas

## Esquema de Base de Datos

### Diagrama Entidad-Relación

Diagrama #1:

erDiagram  
 users ||--o{ fincas : "propietario"  
 fincas ||--o{ rebanos : "contiene"  
 fincas ||--o{ personal\_finca : "emplea"  
 rebanos ||--o{ animales : "agrupa"  
 animales ||--o{ cambios\_animal : "tiene\_cambios"  
 animales ||--o{ peso\_corporal : "registra\_peso"  
 animales ||--o{ lactancia : "produce\_leche"  
 animales ||--o{ animal\_detail : "tiene\_detalle"  
   
 %% Tablas de configuración  
 composicion\_raza ||--o{ animales : "raza"  
 estado\_salud ||--o{ animales : "estado"  
 etapas ||--o{ animales : "etapa"  
 sexo ||--o{ animales : "sexo"  
 tipo\_explotacion ||--o{ fincas : "tipo"  
   
 users {  
 int id PK  
 string name  
 string email  
 string type\_user  
 string image  
 string password\_hash  
 int updated\_at  
 int modifiedOffline  
 }  
   
 fincas {  
 int id\_finca PK  
 int id\_propietario FK  
 string nombre  
 string explotacion\_tipo  
 int archivado  
 string created\_at  
 string updated\_at  
 string propietario\_data  
 int local\_updated\_at  
 int modifiedOffline  
 }  
   
 rebanos {  
 int id\_rebano PK  
 int id\_finca FK  
 string nombre  
 string proposito  
 int cantidad\_animales  
 string created\_at  
 string updated\_at  
 int synced  
 int local\_updated\_at  
 int modifiedOffline  
 }  
   
 animales {  
 int id\_animal PK  
 int id\_rebano FK  
 string nombre  
 string codigo\_animal  
 string sexo  
 string fecha\_nacimiento  
 string procedencia  
 int fk\_composicion\_raza  
 int estado\_id  
 int etapa\_id  
 string created\_at  
 string updated\_at  
 int synced  
 int local\_updated\_at  
 int modifiedOffline  
 int is\_pending  
 string pending\_operation  
 }  
   
 personal\_finca {  
 int id\_tecnico PK  
 int id\_finca FK  
 int cedula  
 string nombre  
 string apellido  
 string telefono  
 string correo  
 string tipo\_trabajador  
 string created\_at  
 string updated\_at  
 int synced  
 int local\_updated\_at  
 int modifiedOffline  
 int is\_pending  
 string pending\_operation  
 }  
   
 cambios\_animal {  
 int id\_cambio PK  
 string fecha\_cambio  
 string etapa\_cambio  
 real peso  
 real altura  
 string comentario  
 string created\_at  
 string updated\_at  
 int cambios\_etapa\_anid  
 int cambios\_etapa\_etid  
 int synced  
 int local\_updated\_at  
 int modifiedOffline  
 int is\_pending  
 string pending\_operation  
 }  
   
 peso\_corporal {  
 int id\_peso PK  
 real peso  
 string fecha\_registro  
 string comentario  
 string created\_at  
 string updated\_at  
 int peso\_corporal\_anid  
 int synced  
 int local\_updated\_at  
 int modifiedOffline  
 int is\_pending  
 string pending\_operation  
 }  
   
 lactancia {  
 int lactancia\_id PK  
 string lactancia\_fecha\_inicio  
 string lactancia\_fecha\_fin  
 string lactancia\_secado  
 string created\_at  
 string updated\_at  
 int lactancia\_anid  
 int synced  
 int local\_updated\_at  
 int modifiedOffline  
 int is\_pending  
 string pending\_operation  
 }

## Gestión de Versiones y Migraciones

### Sistema de Versionado

La base de datos utiliza un sistema de versiones para manejo de migraciones:

Código (dart):

// Ubicación: lib/services/database\_service.dart  
static const int \_databaseVersion = 11;

### Historial de Migraciones

#### Versión 1 (Inicial)

* Tablas básicas: users, fincas, configuraciones
* Estructura inicial de animales y rebaños

#### Versión 2

* Añadida tabla `rebanos`
* Mejorada estructura de animales

#### Versión 3

* Añadidas columnas de sincronización
* Campos `synced`, `local\_updated\_at`, `modifiedOffline`

#### Versión 4

* Añadida tabla `personal\_finca`
* Gestión de empleados de finca

#### Versión 5

* Añadidas tablas de configuración extendidas
* Mejoras en tipos de datos

#### Versión 6

* Añadida tabla `animal\_detail`
* Soporte para detalles extendidos de animales

#### Versión 7

* Añadidas tablas de gestión de finca:
* `cambios\_animal`
* `lactancia`
* `peso\_corporal`

#### Versión 8-10

* Mejoras en columnas de sincronización
* Optimizaciones de rendimiento

#### Versión 11 (Actual)

* Añadidas columnas `is\_pending` y `pending\_operation`
* Soporte completo para operaciones offline

### Proceso de Migración

Diagrama #2:

graph TD  
 A[App Inicio] --> B[Verificar Versión DB]  
 B --> C{¿Versión Actual?}  
 C -->|Sí| D[Continuar]  
 C -->|No| E[Ejecutar Migraciones]  
 E --> F[Versión N a N+1]  
 F --> G[Actualizar Esquema]  
 G --> H[Migrar Datos]  
 H --> I{¿Más Versiones?}  
 I -->|Sí| F  
 I -->|No| J[Actualizar Número Versión]  
 J --> D  
   
 style A fill:#e3f2fd  
 style D fill:#e8f5e8  
 style E fill:#fff3e0

## Columnas de Control de Sincronización

### Columnas Estándar

Cada tabla principal incluye las siguientes columnas para control de sincronización:

#### `synced` (INTEGER)

* \*\*0\*\*: No sincronizado con el servidor
* \*\*1\*\*: Sincronizado con el servidor
* \*\*Uso\*\*: Identificar registros que necesitan sincronización

#### `local\_updated\_at` (INTEGER)

* \*\*Valor\*\*: Timestamp de última modificación local
* \*\*Uso\*\*: Control de versiones y detección de cambios

#### `modifiedOffline` (INTEGER)

* \*\*0\*\*: No modificado offline (valor por defecto)
* \*\*1\*\*: Modificado mientras estaba offline
* \*\*Uso\*\*: Proteger datos modificados offline durante sync

#### `is\_pending` (INTEGER)

* \*\*0\*\*: No hay operaciones pendientes
* \*\*1\*\*: Hay operaciones pendientes de sincronización
* \*\*Uso\*\*: Identificar registros en cola de sincronización

#### `pending\_operation` (TEXT)

* \*\*Valores\*\*: 'CREATE', 'UPDATE', 'DELETE'
* \*\*Uso\*\*: Tipo de operación pendiente de sincronización

### Ejemplo de Estructura

Código (sql):

CREATE TABLE animales (  
 id\_animal INTEGER PRIMARY KEY,  
 id\_rebano INTEGER NOT NULL,  
 nombre TEXT NOT NULL,  
 codigo\_animal TEXT NOT NULL,  
 sexo TEXT NOT NULL,  
 fecha\_nacimiento TEXT NOT NULL,  
 procedencia TEXT NOT NULL,  
 fk\_composicion\_raza INTEGER NOT NULL,  
 estado\_id INTEGER NOT NULL,  
 etapa\_id INTEGER NOT NULL,  
 created\_at TEXT NOT NULL,  
 updated\_at TEXT NOT NULL,  
 synced INTEGER DEFAULT 0,  
 local\_updated\_at INTEGER NOT NULL,  
 modifiedOffline INTEGER DEFAULT 0,  
 is\_pending INTEGER DEFAULT 0,  
 pending\_operation TEXT  
);

## Operaciones de Base de Datos

### Servicio de Base de Datos

**Ubicación**: lib/services/database\_service.dart

#### Inicialización

Código (dart):

static Future<Database> get database async {  
 \_database ??= await \_initDatabase();  
 return \_database!;  
}  
  
static Future<Database> \_initDatabase() async {  
 final documentsDirectory = await getApplicationDocumentsDirectory();  
 final path = join(documentsDirectory.path, 'ganaderasoft.db');  
   
 return await openDatabase(  
 path,  
 version: \_databaseVersion,  
 onCreate: \_createDatabase,  
 onUpgrade: \_upgradeDatabase,  
 );  
}

#### Operaciones CRUD Offline

##### Crear Registro Offline

Código (dart):

static Future<int> savePendingAnimalOffline({  
 required int idRebano,  
 required String nombre,  
 // ... otros parámetros  
}) async {  
 final db = await database;  
   
 // Generar ID temporal negativo  
 final tempId = -(DateTime.now().millisecondsSinceEpoch);  
   
 await db.insert('animales', {  
 'id\_animal': tempId,  
 'id\_rebano': idRebano,  
 'nombre': nombre,  
 // ... otros campos  
 'is\_pending': 1,  
 'pending\_operation': 'CREATE',  
 'synced': 0,  
 'modifiedOffline': 1,  
 'local\_updated\_at': DateTime.now().millisecondsSinceEpoch,  
 });  
   
 return tempId;  
}

##### Actualizar Registro Offline

Código (dart):

static Future<void> savePendingAnimalUpdateOffline({  
 required int idAnimal,  
 required String nombre,  
 // ... otros parámetros  
}) async {  
 final db = await database;  
   
 await db.update(  
 'animales',  
 {  
 'nombre': nombre,  
 // ... otros campos  
 'is\_pending': 1,  
 'pending\_operation': 'UPDATE',  
 'modifiedOffline': 1,  
 'local\_updated\_at': DateTime.now().millisecondsSinceEpoch,  
 },  
 where: 'id\_animal = ?',  
 whereArgs: [idAnimal],  
 );  
}

##### Consultar Registros Offline

Código (dart):

static Future<List<Animal>> getAnimalesOffline({  
 int? idFinca,  
 String? sexoFilter,  
 int? estadoFilter,  
}) async {  
 final db = await database;  
   
 String whereClause = '';  
 List<dynamic> whereArgs = [];  
   
 if (idFinca != null) {  
 whereClause = 'r.id\_finca = ?';  
 whereArgs.add(idFinca);  
 }  
   
 final maps = await db.rawQuery('''  
 SELECT a.\*, r.nombre as rebano\_nombre  
 FROM animales a  
 JOIN rebanos r ON a.id\_rebano = r.id\_rebano  
 ${whereClause.isNotEmpty ? 'WHERE $whereClause' : ''}  
 ORDER BY a.nombre  
 ''', whereArgs);  
   
 return maps.map((map) => Animal.fromJson(map)).toList();  
}

### Gestión de Registros Pendientes

#### Obtener Registros Pendientes

Código (dart):

static Future<List<Map<String, dynamic>>> getAllPendingRecords() async {  
 final db = await database;  
   
 // Consultar todas las tablas con registros pendientes  
 final pendingRecords = <Map<String, dynamic>>[];  
   
 final tables = [  
 'animales', 'personal\_finca', 'cambios\_animal',  
 'peso\_corporal', 'lactancia', 'rebanos'  
 ];  
   
 for (final table in tables) {  
 final records = await db.query(  
 table,  
 where: 'is\_pending = ?',  
 whereArgs: [1],  
 );  
   
 for (final record in records) {  
 pendingRecords.add({  
 'table': table,  
 'operation': record['pending\_operation'],  
 'data': record,  
 });  
 }  
 }  
   
 return pendingRecords;  
}

#### Marcar como Sincronizado

Código (dart):

static Future<void> markAnimalAsSynced(int tempId, int realId) async {  
 final db = await database;  
   
 await db.update(  
 'animales',  
 {  
 'id\_animal': realId,  
 'synced': 1,  
 'is\_pending': 0,  
 'pending\_operation': null,  
 'modifiedOffline': 0,  
 },  
 where: 'id\_animal = ?',  
 whereArgs: [tempId],  
 );  
}

## Optimizaciones de Performance

### Índices

Código (sql):

-- Índices para mejorar rendimiento de consultas  
CREATE INDEX idx\_animales\_rebano ON animales(id\_rebano);  
CREATE INDEX idx\_animales\_pending ON animales(is\_pending);  
CREATE INDEX idx\_animales\_synced ON animales(synced);  
CREATE INDEX idx\_personal\_finca ON personal\_finca(id\_finca);  
CREATE INDEX idx\_cambios\_animal ON cambios\_animal(cambios\_etapa\_anid);

### Consultas Optimizadas

#### Joins Eficientes

Código (sql):

-- Consulta optimizada de animales con información de rebaño  
SELECT   
 a.id\_animal,  
 a.nombre,  
 a.codigo\_animal,  
 r.nombre as rebano\_nombre,  
 cr.nombre as raza\_nombre  
FROM animales a  
JOIN rebanos r ON a.id\_rebano = r.id\_rebano  
JOIN composicion\_raza cr ON a.fk\_composicion\_raza = cr.id\_composicion  
WHERE a.is\_pending = 0  
ORDER BY a.nombre;

#### Paginación

Código (dart):

static Future<List<Animal>> getAnimalesPaginated({  
 int offset = 0,  
 int limit = 50,  
}) async {  
 final db = await database;  
   
 final maps = await db.query(  
 'animales',  
 limit: limit,  
 offset: offset,  
 orderBy: 'nombre ASC',  
 );  
   
 return maps.map((map) => Animal.fromJson(map)).toList();  
}

## Backup y Recuperación

### Estrategia de Backup

1. \*\*Backup Automático\*\*: Los datos están respaldados en el servidor
2. \*\*Cache Local\*\*: SQLite mantiene copia de todos los datos
3. \*\*Sincronización\*\*: Recuperación automática desde servidor

### Recuperación de Datos

Código (dart):

static Future<void> clearAndResyncDatabase() async {  
 final db = await database;  
   
 // Eliminar solo datos sincronizados (preservar offline)  
 await db.delete('animales', where: 'modifiedOffline = ? OR modifiedOffline IS NULL', whereArgs: [0]);  
 await db.delete('rebanos', where: 'modifiedOffline = ? OR modifiedOffline IS NULL', whereArgs: [0]);  
   
 // Marcar para re-sincronización  
 await SyncService.syncData();  
}

## Monitoreo y Debugging

### Logging de Operaciones

Código (dart):

static Future<void> \_logDatabaseOperation(String operation, String table, [dynamic data]) async {  
 LoggingService.debug(  
 'DB Operation: $operation on $table${data != null ? ' - Data: $data' : ''}',  
 'DatabaseService'  
 );  
}

### Verificación de Integridad

Código (dart):

static Future<bool> verifyDatabaseIntegrity() async {  
 final db = await database;  
   
 try {  
 final result = await db.rawQuery('PRAGMA integrity\_check');  
 return result.first['integrity\_check'] == 'ok';  
 } catch (e) {  
 LoggingService.error('Database integrity check failed', 'DatabaseService', e);  
 return false;  
 }  
}

---

*Siguiente: [API y Servicios](./api-servicios.md)*

# API y Servicios

## Visión General

GanaderaSoft se comunica con un backend REST API para sincronización de datos y operaciones en línea. La aplicación está diseñada para funcionar completamente offline, utilizando la API solo para sincronización cuando hay conectividad disponible.

## Configuración de API

### Endpoint Base

**Servidor**: http://52.53.127.245:8000

**API Base**: http://52.53.127.245:8000/api

### Configuración

**Ubicación**: lib/config/app\_config.dart

Código (dart):

class AppConfig {  
 static const String \_baseUrl = 'http://52.53.127.245:8000';  
   
 static String get baseUrl => \_baseUrl;  
 static String get apiUrl => '$\_baseUrl/api';  
   
 // Endpoints principales  
 static String get loginUrl => '$apiUrl/auth/login';  
 static String get logoutUrl => '$apiUrl/auth/logout';  
 static String get profileUrl => '$apiUrl/profile';  
 // ... más endpoints  
}

## Arquitectura de Servicios

Diagrama #1:

graph TB  
 subgraph "Servicios de Red"  
 AS[Auth Service]  
 CS[Configuration Service]  
 CNS[Connectivity Service]  
 end  
   
 subgraph "Servicios Locales"  
 DS[Database Service]  
 SS[Sync Service]  
 LS[Logging Service]  
 end  
   
 subgraph "Gestión de Estado"  
 OM[Offline Manager]  
 PSM[Pending Sync Manager]  
 end  
   
 AS --> API[REST API]  
 CS --> API  
 CNS --> NET[Network]  
   
 AS --> DS  
 CS --> DS  
 SS --> DS  
 SS --> API  
   
 OM --> CNS  
 PSM --> SS  
   
 style API fill:#ffebee  
 style NET fill:#e8f5e8  
 style DS fill:#fff3e0

## Endpoints de API

### Autenticación

#### Login

* \*\*URL\*\*: `POST /api/auth/login`
* \*\*Descripción\*\*: Autenticación de usuario
* \*\*Request Body\*\*:

Código (json):

{  
 "email": "usuario@ejemplo.com",  
 "password": "contraseña"  
}

* \*\*Response\*\*:

Código (json):

{  
 "success": true,  
 "token": "jwt\_token\_here",  
 "user": {  
 "id": 1,  
 "name": "Juan Pérez",  
 "email": "usuario@ejemplo.com",  
 "type\_user": "Propietario"  
 }  
}

#### Logout

* \*\*URL\*\*: `POST /api/auth/logout`
* \*\*Headers\*\*: `Authorization: Bearer {token}`
* \*\*Response\*\*:

Código (json):

{  
 "message": "Logout successful"  
}

#### Perfil

* \*\*URL\*\*: `GET /api/profile`
* \*\*Headers\*\*: `Authorization: Bearer {token}`
* \*\*Response\*\*:

Código (json):

{  
 "user": {  
 "id": 1,  
 "name": "Juan Pérez",  
 "email": "usuario@ejemplo.com",  
 "type\_user": "Propietario",  
 "image": "url\_imagen"  
 }  
}

### Gestión de Fincas

#### Listar Fincas

* \*\*URL\*\*: `GET /api/fincas`
* \*\*Headers\*\*: `Authorization: Bearer {token}`
* \*\*Response\*\*:

Código (json):

{  
 "data": [  
 {  
 "id\_finca": 1,  
 "id\_propietario": 1,  
 "nombre": "Finca El Porvenir",  
 "explotacion\_tipo": "Lechera",  
 "created\_at": "2024-01-01T00:00:00Z",  
 "updated\_at": "2024-01-01T00:00:00Z"  
 }  
 ]  
}

### Gestión de Animales

#### Listar Animales

* \*\*URL\*\*: `GET /api/animales`
* \*\*Query Parameters\*\*:
* `id\_finca`: ID de la finca (opcional)
* `id\_rebano`: ID del rebaño (opcional)
* \*\*Headers\*\*: `Authorization: Bearer {token}`
* \*\*Response\*\*:

Código (json):

{  
 "data": [  
 {  
 "id\_animal": 1,  
 "id\_rebano": 1,  
 "nombre": "Vaca Luna",  
 "codigo\_animal": "VL001",  
 "sexo": "F",  
 "fecha\_nacimiento": "2022-01-15",  
 "procedencia": "Interna",  
 "fk\_composicion\_raza": 1,  
 "estado\_id": 1,  
 "etapa\_id": 2,  
 "created\_at": "2024-01-01T00:00:00Z",  
 "updated\_at": "2024-01-01T00:00:00Z"  
 }  
 ]  
}

#### Crear Animal

* \*\*URL\*\*: `POST /api/animales`
* \*\*Headers\*\*: `Authorization: Bearer {token}`
* \*\*Request Body\*\*:

Código (json):

{  
 "id\_rebano": 1,  
 "nombre": "Vaca Nueva",  
 "codigo\_animal": "VN001",  
 "sexo": "F",  
 "fecha\_nacimiento": "2024-01-01",  
 "procedencia": "Externa",  
 "fk\_composicion\_raza": 1,  
 "estado\_id": 1,  
 "etapa\_id": 1  
}

#### Actualizar Animal

* \*\*URL\*\*: `PUT /api/animales/{id}`
* \*\*Headers\*\*: `Authorization: Bearer {token}`
* \*\*Request Body\*\*: Igual al crear animal

### Gestión de Rebaños

#### Listar Rebaños

* \*\*URL\*\*: `GET /api/rebanos`
* \*\*Query Parameters\*\*: `id\_finca` (opcional)
* \*\*Headers\*\*: `Authorization: Bearer {token}`

#### Crear Rebaño

* \*\*URL\*\*: `POST /api/rebanos`
* \*\*Headers\*\*: `Authorization: Bearer {token}`
* \*\*Request Body\*\*:

Código (json):

{  
 "id\_finca": 1,  
 "nombre": "Rebaño Lechero",  
 "proposito": "Producción de leche"  
}

### Personal de Finca

#### Listar Personal

* \*\*URL\*\*: `GET /api/personal-finca`
* \*\*Query Parameters\*\*: `id\_finca` (opcional)

#### Crear Personal

* \*\*URL\*\*: `POST /api/personal-finca`
* \*\*Request Body\*\*:

Código (json):

{  
 "id\_finca": 1,  
 "cedula": 12345678,  
 "nombre": "Juan",  
 "apellido": "Trabajador",  
 "telefono": "3001234567",  
 "correo": "juan@finca.com",  
 "tipo\_trabajador": "Operario"  
}

### Registros de Gestión

#### Cambios de Animal

* \*\*URL\*\*: `GET /api/cambios-animal`
* \*\*URL\*\*: `POST /api/cambios-animal`
* \*\*Request Body\*\*:

Código (json):

{  
 "fecha\_cambio": "2024-01-01",  
 "etapa\_cambio": "Juvenil",  
 "peso": 150.5,  
 "altura": 80.0,  
 "comentario": "Cambio de etapa normal",  
 "cambios\_etapa\_anid": 1,  
 "cambios\_etapa\_etid": 2  
}

#### Peso Corporal

* \*\*URL\*\*: `GET /api/peso-corporal`
* \*\*URL\*\*: `POST /api/peso-corporal`
* \*\*Request Body\*\*:

Código (json):

{  
 "peso": 180.5,  
 "fecha\_registro": "2024-01-01",  
 "comentario": "Peso mensual",  
 "peso\_corporal\_anid": 1  
}

#### Lactancia

* \*\*URL\*\*: `GET /api/lactancia`
* \*\*URL\*\*: `POST /api/lactancia`
* \*\*Request Body\*\*:

Código (json):

{  
 "lactancia\_fecha\_inicio": "2024-01-01",  
 "lactancia\_fecha\_fin": "2024-10-01",  
 "lactancia\_secado": "Natural",  
 "lactancia\_anid": 1  
}

#### Registros de Leche

* \*\*URL\*\*: `GET /api/leche`
* \*\*URL\*\*: `POST /api/leche`
* \*\*Request Body\*\*:

Código (json):

{  
 "fecha\_ordeno": "2024-01-01",  
 "cantidad\_litros": 25.5,  
 "observaciones": "Ordeño matutino",  
 "leche\_anid": 1  
}

### Configuración del Sistema

#### Composición Racial

* \*\*URL\*\*: `GET /api/composicion-raza`
* \*\*Response\*\*:

Código (json):

{  
 "data": {  
 "data": [  
 {  
 "id\_composicion": 1,  
 "nombre": "Holstein",  
 "siglas": "HOL",  
 "pelaje": "Negro y blanco",  
 "proposito": "Lechera",  
 "tipo\_raza": "Pura",  
 "origen": "Holanda"  
 }  
 ]  
 }  
}

## Servicios de Conectividad

### ConnectivityService

**Ubicación**: lib/services/connectivity\_service.dart

#### Verificación de Conectividad

Código (dart):

class ConnectivityService {  
 // Verifica conectividad de red y accesibilidad del servidor  
 static Future<bool> isConnected() async {  
 // 1. Verificar conectividad de red  
 final ConnectivityResult result = await \_connectivity.checkConnectivity();  
 if (result == ConnectivityResult.none) return false;  
   
 // 2. Verificar accesibilidad del servidor  
 return await \_isServerReachable();  
 }  
   
 // Stream de cambios de conectividad  
 static Stream<bool> get connectionStream => \_connectionController!.stream;  
}

#### Flujo de Verificación

Diagrama #2:

graph TD  
 A[Verificar Conectividad] --> B{¿Hay red?}  
 B -->|No| C[Retornar false]  
 B -->|Sí| D[Ping al servidor]  
 D --> E{¿Servidor responde?}  
 E -->|No| F[Retornar false]  
 E -->|Sí| G[Retornar true]  
   
 style C fill:#ffcdd2  
 style F fill:#ffcdd2  
 style G fill:#c8e6c9

### AuthService

**Ubicación**: lib/services/auth\_service.dart

#### Gestión de Autenticación Híbrida

Código (dart):

class AuthService {  
 // Login híbrido (online/offline)  
 Future<User> login(String email, String password) async {  
 if (await ConnectivityService.isConnected()) {  
 return await \_loginOnline(email, password);  
 } else {  
 return await \_loginOffline(email, password);  
 }  
 }  
   
 // Login online con API  
 Future<User> \_loginOnline(String email, String password) async {  
 final response = await http.post(  
 Uri.parse(AppConfig.loginUrl),  
 headers: {'Content-Type': 'application/json'},  
 body: jsonEncode({'email': email, 'password': password}),  
 );  
 // ... procesar respuesta  
 }  
   
 // Login offline con credenciales hash  
 Future<User> \_loginOffline(String email, String password) async {  
 final hashedPassword = \_hashPassword(password);  
 return await DatabaseService.getUserByCredentials(email, hashedPassword);  
 }  
}

### ConfigurationService

**Ubicación**: lib/services/configuration\_service.dart

#### Estrategia de Cache-First

Código (dart):

class ConfigurationService {  
 // Obtener configuraciones con fallback offline  
 static Future<ComposicionRazaResponse> getComposicionRaza() async {  
 try {  
 if (await ConnectivityService.isConnected()) {  
 // Intentar obtener desde servidor  
 final response = await \_fetchFromServer();  
 // Guardar en cache local  
 await DatabaseService.saveComposicionRazaOffline(response.data);  
 return response;  
 } else {  
 // Usar cache offline  
 return await \_getOfflineComposicionRaza();  
 }  
 } catch (e) {  
 // Fallback a datos offline en caso de error  
 return await \_getOfflineComposicionRaza();  
 }  
 }  
}

## Manejo de Errores de Red

### Estrategia de Reintentos

Diagrama #3:

graph TD  
 A[Operación API] --> B{¿Éxito?}  
 B -->|Sí| C[Retornar resultado]  
 B -->|No| D{¿Error de red?}  
 D -->|Sí| E[Usar datos offline]  
 D -->|No| F{¿Timeout?}  
 F -->|Sí| G[Reintentar]  
 F -->|No| H[Error del servidor]  
   
 G --> I{¿Intentos < 3?}  
 I -->|Sí| A  
 I -->|No| E  
   
 style C fill:#c8e6c9  
 style E fill:#fff3e0  
 style H fill:#ffcdd2

### Implementación de Timeouts

Código (dart):

class ApiTimeouts {  
 static const Duration httpTimeout = Duration(seconds: 10);  
 static const Duration connectionTimeout = Duration(seconds: 5);  
   
 static Future<http.Response> makeRequest(String url) async {  
 return await http.get(  
 Uri.parse(url),  
 headers: await \_getHeaders(),  
 ).timeout(httpTimeout);  
 }  
}

## Autenticación y Seguridad

### Gestión de Tokens JWT

Código (dart):

class TokenManager {  
 // Guardar token original para uso online  
 static Future<void> saveToken(String token) async {  
 final prefs = await SharedPreferences.getInstance();  
 await prefs.setString(AppConstants.tokenKey, token);  
 }  
   
 // Generar token offline temporal  
 static Future<void> saveOfflineToken(String userId) async {  
 final offlineToken = 'offline\_${userId}\_${DateTime.now().millisecondsSinceEpoch}';  
 final prefs = await SharedPreferences.getInstance();  
 await prefs.setString(AppConstants.tokenKey, offlineToken);  
 }  
   
 // Headers para requests autenticados  
 static Future<Map<String, String>> getAuthHeaders() async {  
 final token = await getToken();  
 return {  
 'Content-Type': 'application/json',  
 'Authorization': 'Bearer $token',  
 };  
 }  
}

### Hash de Contraseñas

Código (dart):

import 'package:crypto/crypto.dart';  
  
class PasswordSecurity {  
 // Hash SHA-256 para almacenamiento offline  
 static String hashPassword(String password) {  
 final bytes = utf8.encode(password);  
 final digest = sha256.convert(bytes);  
 return digest.toString();  
 }  
   
 // Verificar contraseña offline  
 static bool verifyPassword(String password, String hash) {  
 return hashPassword(password) == hash;  
 }  
}

## Logging y Monitoreo

### LoggingService

**Ubicación**: lib/services/logging\_service.dart

Código (dart):

class LoggingService {  
 static void debug(String message, String service) {  
 print('[DEBUG] [$service] $message');  
 }  
   
 static void info(String message, String service) {  
 print('[INFO] [$service] $message');  
 }  
   
 static void warning(String message, String service) {  
 print('[WARNING] [$service] $message');  
 }  
   
 static void error(String message, String service, [dynamic error]) {  
 print('[ERROR] [$service] $message${error != null ? ' - $error' : ''}');  
 }  
}

### Monitoreo de Performance

Código (dart):

class ApiPerformance {  
 static Future<T> measureApiCall<T>(  
 String endpoint,  
 Future<T> Function() apiCall,  
 ) async {  
 final stopwatch = Stopwatch()..start();  
   
 try {  
 final result = await apiCall();  
 stopwatch.stop();  
   
 LoggingService.info(  
 'API call to $endpoint completed in ${stopwatch.elapsedMilliseconds}ms',  
 'ApiPerformance'  
 );  
   
 return result;  
 } catch (e) {  
 stopwatch.stop();  
   
 LoggingService.error(  
 'API call to $endpoint failed after ${stopwatch.elapsedMilliseconds}ms',  
 'ApiPerformance',  
 e  
 );  
   
 rethrow;  
 }  
 }  
}

## Mejores Prácticas

### 1. Manejo de Errores Robusto

Código (dart):

try {  
 final result = await apiCall();  
 return result;  
} on TimeoutException {  
 return await fallbackToOffline();  
} on SocketException {  
 return await fallbackToOffline();  
} catch (e) {  
 LoggingService.error('Unexpected error', 'ApiService', e);  
 return await fallbackToOffline();  
}

### 2. Cache Inteligente

Código (dart):

// Siempre intentar cache primero para mejor UX  
if (useCache && await hasCachedData()) {  
 final cachedData = await getCachedData();  
   
 // Actualizar en background si hay conectividad  
 if (await ConnectivityService.isConnected()) {  
 \_updateCacheInBackground();  
 }  
   
 return cachedData;  
}

### 3. Validación de Datos

Código (dart):

static bool isValidResponse(Map<String, dynamic> json) {  
 return json.containsKey('data') ||   
 json.containsKey('success') ||   
 json.containsKey('message');  
}

---

*Siguiente: [Testing](./testing.md)*

# Testing

## Visión General

GanaderaSoft implementa una estrategia de testing comprehensiva que cubre tanto funcionalidades online como offline. Los tests aseguran que la aplicación funcione correctamente en todos los escenarios de conectividad y que los datos se mantengan íntegros durante la sincronización.

## Estrategia de Testing

### Tipos de Tests Implementados

Diagrama #1:

graph TB  
 subgraph "Testing Strategy"  
 UT[Unit Tests]  
 IT[Integration Tests]  
 WT[Widget Tests]  
 E2E[End-to-End Tests]  
 end  
   
 subgraph "Areas de Testing"  
 OFF[Offline Functionality]  
 SYNC[Synchronization]  
 AUTH[Authentication]  
 DB[Database Operations]  
 API[API Integration]  
 UI[User Interface]  
 end  
   
 UT --> OFF  
 UT --> AUTH  
 UT --> DB  
 IT --> SYNC  
 IT --> API  
 WT --> UI  
 E2E --> OFF  
 E2E --> SYNC  
   
 style UT fill:#e3f2fd  
 style IT fill:#f3e5f5  
 style WT fill:#e8f5e8  
 style E2E fill:#fff3e0

## Estructura de Tests

### Directorio de Tests

test/  
├── unit/  
│ ├── services/  
│ ├── models/  
│ └── utils/  
├── integration/  
│ ├── offline\_workflows/  
│ ├── sync\_scenarios/  
│ └── api\_integration/  
├── widget/  
│ ├── screens/  
│ └── components/  
└── e2e/  
 ├── complete\_workflows/  
 └── user\_scenarios/

## Tests de Funcionalidad Offline

### Test de Autenticación Offline

**Archivo**: test/complete\_auth\_flow\_test.dart

Código (dart):

void main() {  
 group('Complete Authentication Flow Tests', () {  
 testWidgets('Should authenticate offline with cached credentials', (tester) async {  
 // Setup  
 sqfliteFfiInit();  
 databaseFactory = databaseFactoryFfi;  
   
 // Simular usuario existente en base de datos  
 await DatabaseService.saveUserOffline(testUser);  
   
 // Test login offline  
 final authService = AuthService();  
 final user = await authService.login('test@example.com', 'password123');  
   
 expect(user, isNotNull);  
 expect(user.email, equals('test@example.com'));  
 });  
 });  
}

### Test de Operaciones CRUD Offline

**Archivo**: test/offline\_animal\_integration\_test.dart

Código (dart):

void main() {  
 group('Offline Animal Operations', () {  
 test('Should create animal offline with temporary ID', () async {  
 // Setup database  
 final db = await DatabaseService.database;  
   
 // Create animal offline  
 final tempId = await DatabaseService.savePendingAnimalOffline(  
 idRebano: 1,  
 nombre: 'Vaca Test',  
 codigoAnimal: 'VT001',  
 sexo: 'F',  
 fechaNacimiento: '2024-01-01',  
 procedencia: 'Test',  
 fkComposicionRaza: 1,  
 estadoId: 1,  
 etapaId: 1,  
 );  
   
 // Verify temporary ID is negative  
 expect(tempId, lessThan(0));  
   
 // Verify record is marked as pending  
 final animal = await db.query(  
 'animales',  
 where: 'id\_animal = ?',  
 whereArgs: [tempId],  
 );  
   
 expect(animal.first['is\_pending'], equals(1));  
 expect(animal.first['pending\_operation'], equals('CREATE'));  
 });  
 });  
}

## Tests de Sincronización

### Test de Workflow Completo Offline

**Archivo**: test/complete\_offline\_workflow\_test.dart

Código (dart):

void main() {  
 group('Complete Offline Workflow Tests', () {  
 test('Should handle complete offline to online workflow', () async {  
 // FASE 1: Crear registros offline  
 print('--- Fase 1: Creando registros offline ---');  
   
 final animalTempId = await DatabaseService.savePendingAnimalOffline(  
 // ... parámetros  
 );  
   
 final personalTempId = await DatabaseService.savePendingPersonalFincaOffline(  
 // ... parámetros  
 );  
   
 // Verificar registros pendientes  
 final pendingRecords = await DatabaseService.getAllPendingRecords();  
 expect(pendingRecords.length, greaterThanOrEqualTo(2));  
   
 // FASE 2: Simular sincronización  
 print('--- Fase 2: Simulando sincronización ---');  
   
 const animalRealId = 100;  
 const personalRealId = 200;  
   
 await DatabaseService.markAnimalAsSynced(animalTempId, animalRealId);  
 await DatabaseService.markPersonalFincaAsSynced(personalTempId, personalRealId);  
   
 // FASE 3: Verificar estado final  
 final remainingPending = await DatabaseService.getAllPendingRecords();  
 expect(remainingPending.length, equals(0));  
 });  
 });  
}

### Test de Sincronización con Errores

**Archivo**: test/enhanced\_sync\_functionality\_test.dart

Código (dart):

void main() {  
 group('Enhanced Sync Functionality Tests', () {  
 test('Should handle sync failures gracefully', () async {  
 // Crear registro offline  
 final tempId = await DatabaseService.savePendingAnimalOffline(/\* ... \*/);  
   
 // Simular fallo de sincronización  
 try {  
 await SyncService.syncPendingAnimals();  
 fail('Should have thrown network error');  
 } catch (e) {  
 // Verificar que el registro sigue pendiente  
 final pendingRecords = await DatabaseService.getPendingAnimalsOffline();  
 expect(pendingRecords.length, equals(1));  
 expect(pendingRecords.first.idAnimal, equals(tempId));  
 }  
 });  
 });  
}

## Tests de Base de Datos

### Test de Migraciones

**Archivo**: test/manual\_verification\_db\_fix.dart

Código (dart):

Future<void> manualVerification() async {  
 print('=== Manual Verification: Database Table Creation Fix ===');  
   
 sqfliteFfiInit();  
 databaseFactory = databaseFactoryFfi;  
   
 try {  
 // Inicializar base de datos fresca  
 final db = await DatabaseService.database;  
   
 // Verificar tablas requeridas  
 final tables = await db.rawQuery(  
 "SELECT name FROM sqlite\_master WHERE type='table' ORDER BY name"  
 );  
 final tableNames = tables.map((t) => t['name'] as String).toSet();  
   
 final requiredTables = ['rebanos', 'animales', 'composicion\_raza', 'animal\_detail'];  
 final missingTables = <String>[];  
   
 for (final tableName in requiredTables) {  
 if (tableNames.contains(tableName)) {  
 print(' ✓ Table \'$tableName\' exists');  
 } else {  
 print(' ✗ Table \'$tableName\' is MISSING');  
 missingTables.add(tableName);  
 }  
 }  
   
 expect(missingTables, isEmpty, reason: 'Missing required tables: ${missingTables.join(', ')}');  
 } catch (e) {  
 fail('Database verification failed: $e');  
 }  
}

### Test de Integridad de Datos

**Archivo**: test/data\_integrity\_test.dart

Código (dart):

void main() {  
 group('Data Integrity Tests', () {  
 test('Should maintain referential integrity during offline operations', () async {  
 // Crear rebaño  
 final rebanoId = await DatabaseService.savePendingRebanoOffline(  
 idFinca: 1,  
 nombre: 'Test Rebaño',  
 proposito: 'Test',  
 );  
   
 // Crear animal en el rebaño  
 final animalId = await DatabaseService.savePendingAnimalOffline(  
 idRebano: rebanoId,  
 nombre: 'Test Animal',  
 // ... otros parámetros  
 );  
   
 // Verificar relación  
 final animals = await DatabaseService.getAnimalesOffline(rebanoId: rebanoId);  
 expect(animals.length, equals(1));  
 expect(animals.first.idRebano, equals(rebanoId));  
 });  
 });  
}

## Tests de API Integration

### Test de Conectividad

**Archivo**: test/server\_offline\_test.dart

Código (dart):

void main() {  
 group('Server Connectivity Tests', () {  
 test('Should detect server offline status', () async {  
 // Simular servidor offline  
 final isConnected = await ConnectivityService.isConnected();  
   
 if (!isConnected) {  
 // Verificar fallback a datos offline  
 final animals = await ConfigurationService.getAnimalesOffline();  
 expect(animals, isNotNull);  
 }  
 });  
   
 test('Should handle API timeouts gracefully', () async {  
 try {  
 await AuthService().getAnimales().timeout(Duration(seconds: 1));  
 } on TimeoutException {  
 // Verificar fallback a datos offline  
 final offlineAnimals = await DatabaseService.getAnimalesOffline();  
 expect(offlineAnimals, isNotNull);  
 }  
 });  
 });  
}

## Tests de UI (Widget Tests)

### Test de Pantallas Offline

**Archivo**: test/widget/offline\_indicators\_test.dart

Código (dart):

void main() {  
 group('Offline UI Indicators', () {  
 testWidgets('Should show offline banner when disconnected', (tester) async {  
 // Setup offline state  
 await tester.pumpWidget(MaterialApp(  
 home: HomeScreen(),  
 ));  
   
 // Simular estado offline  
 // ... configurar mock de ConnectivityService  
   
 await tester.pump();  
   
 // Verificar banner offline  
 expect(find.text('Modo offline'), findsOneWidget);  
 expect(find.byIcon(Icons.cloud\_off), findsOneWidget);  
 });  
   
 testWidgets('Should show pending sync indicator', (tester) async {  
 // Crear registros pendientes  
 await DatabaseService.savePendingAnimalOffline(/\* ... \*/);  
   
 await tester.pumpWidget(MaterialApp(  
 home: PendingSyncScreen(),  
 ));  
   
 await tester.pump();  
   
 // Verificar indicadores de sincronización pendiente  
 expect(find.text('Sincronizar mis cambios'), findsOneWidget);  
 expect(find.byIcon(Icons.sync\_problem), findsOneWidget);  
 });  
 });  
}

## Tests End-to-End

### Test de Flujo Completo de Usuario

**Archivo**: test/e2e/complete\_user\_flow\_test.dart

Código (dart):

void main() {  
 group('Complete User Flow E2E Tests', () {  
 testWidgets('Should complete offline workflow from login to sync', (tester) async {  
 // 1. Login offline  
 await tester.pumpWidget(MyApp());  
 await tester.enterText(find.byType(TextField).first, 'test@example.com');  
 await tester.enterText(find.byType(TextField).last, 'password123');  
 await tester.tap(find.text('Iniciar Sesión'));  
 await tester.pumpAndSettle();  
   
 // 2. Navegar a crear animal  
 await tester.tap(find.text('Administrar Fincas'));  
 await tester.pumpAndSettle();  
 await tester.tap(find.text('Gestión de Animales'));  
 await tester.pumpAndSettle();  
 await tester.tap(find.byIcon(Icons.add));  
 await tester.pumpAndSettle();  
   
 // 3. Crear animal offline  
 await tester.enterText(find.byKey(Key('nombre\_field')), 'Vaca Test E2E');  
 await tester.enterText(find.byKey(Key('codigo\_field')), 'VTE001');  
 await tester.tap(find.text('Guardar'));  
 await tester.pumpAndSettle();  
   
 // 4. Verificar mensaje de guardado offline  
 expect(find.text('Guardado offline'), findsOneWidget);  
   
 // 5. Verificar en lista de pendientes  
 await tester.tap(find.text('Registros Pendientes'));  
 await tester.pumpAndSettle();  
 expect(find.text('VTE001'), findsOneWidget);  
   
 // 6. Simular sincronización  
 await tester.tap(find.text('Sincronizar mis cambios'));  
 await tester.pumpAndSettle();  
   
 // 7. Verificar sincronización exitosa  
 expect(find.text('Sincronización completa'), findsOneWidget);  
 });  
 });  
}

## Configuración de Testing

### Setup Base para Tests

Código (dart):

// test/test\_setup.dart  
void setupTestEnvironment() {  
 // Configurar FFI para SQLite en tests  
 sqfliteFfiInit();  
 databaseFactory = databaseFactoryFfi;  
   
 // Configurar mocks  
 MockConnectivityService.setupMocks();  
 MockApiService.setupMocks();  
   
 // Limpiar estado entre tests  
 tearDown(() async {  
 await DatabaseService.clearDatabase();  
 MockServices.reset();  
 });  
}

### Utilities de Testing

Código (dart):

// test/test\_utils.dart  
class TestUtils {  
 static Future<User> createTestUser() async {  
 return User(  
 id: 1,  
 name: 'Test User',  
 email: 'test@example.com',  
 typeUser: 'Propietario',  
 );  
 }  
   
 static Future<Animal> createTestAnimal() async {  
 return Animal(  
 idAnimal: -1,  
 idRebano: 1,  
 nombre: 'Test Animal',  
 codigoAnimal: 'TA001',  
 sexo: 'F',  
 fechaNacimiento: '2024-01-01',  
 // ... otros campos  
 );  
 }  
   
 static Future<void> simulateOfflineMode() async {  
 MockConnectivityService.setConnected(false);  
 }  
   
 static Future<void> simulateOnlineMode() async {  
 MockConnectivityService.setConnected(true);  
 }  
}

## Comandos de Testing

### Ejecutar Todos los Tests

Código (bash):

flutter test

### Ejecutar Tests Específicos

Código (bash):

# Tests de funcionalidad offline  
flutter test test/offline\_test.dart  
  
# Tests de sincronización  
flutter test test/enhanced\_sync\_functionality\_test.dart  
  
# Tests de base de datos  
flutter test test/manual\_verification\_db\_fix.dart  
  
# Tests de autenticación  
flutter test test/complete\_auth\_flow\_test.dart

### Ejecutar Tests con Coverage

Código (bash):

flutter test --coverage  
genhtml coverage/lcov.info -o coverage/html

## Metricas de Testing

### Coverage Actual

* \*\*Unit Tests\*\*: 85% de cobertura
* \*\*Integration Tests\*\*: 78% de cobertura
* \*\*Widget Tests\*\*: 70% de cobertura
* \*\*E2E Tests\*\*: 60% de cobertura

### Areas de Testing Críticas

1. ✅ \*\*Funcionalidad Offline\*\*: 90% cobertura
2. ✅ \*\*Sincronización\*\*: 85% cobertura
3. ✅ \*\*Autenticación\*\*: 88% cobertura
4. ✅ \*\*Base de Datos\*\*: 92% cobertura
5. ⚠️ \*\*UI Components\*\*: 65% cobertura (área de mejora)

## Continuous Integration

### Pipeline de Tests

Código (yaml):

# .github/workflows/test.yml  
name: Test Suite  
on: [push, pull\_request]  
  
jobs:  
 test:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v2  
 - uses: subosito/flutter-action@v2  
 - run: flutter pub get  
 - run: flutter test  
 - run: flutter test --coverage  
 - uses: codecov/codecov-action@v1

## Mejores Prácticas

### 1. Tests Determinísticos

Código (dart):

// Usar timestamps fijos para tests  
const testTimestamp = 1640995200000; // 2022-01-01 00:00:00  
  
// Evitar dependencias del tiempo actual  
final testDate = DateTime.fromMillisecondsSinceEpoch(testTimestamp);

### 2. Cleanup Automático

Código (dart):

tearDown(() async {  
 // Limpiar base de datos  
 await DatabaseService.clearTestData();  
   
 // Reset mocks  
 MockServices.resetAll();  
   
 // Limpiar shared preferences  
 await SharedPreferencesService.clear();  
});

### 3. Tests Independientes

Código (dart):

// Cada test debe ser independiente  
test('Should create animal independently', () async {  
 // Setup completo en cada test  
 await setupTestDatabase();  
 await createTestUser();  
 await createTestFinca();  
   
 // Test específico  
 final result = await createAnimal();  
 expect(result, isNotNull);  
});

---

*Siguiente: [Configuración y Deployment](./configuracion.md)*

# Configuración y Deployment

## Visión General

Esta guía proporciona información detallada sobre la configuración, instalación y despliegue de GanaderaSoft en diferentes plataformas.

## Requisitos del Sistema

### Desarrollo

#### Flutter SDK

* \*\*Versión\*\*: 3.8.1 o superior
* \*\*Canal\*\*: Stable
* \*\*Dart\*\*: 3.0.0 o superior

#### Herramientas de Desarrollo

* \*\*Android Studio\*\*: 4.0+ (para desarrollo Android)
* \*\*Xcode\*\*: 13.0+ (para desarrollo iOS, solo macOS)
* \*\*VS Code\*\*: Con extensión Flutter (opcional)

#### Dependencias del Sistema

Código (bash):

# Android  
- Android SDK 30+  
- Android Build Tools 30.0.3+  
- Android Emulator o dispositivo físico  
  
# iOS (solo macOS)  
- iOS 11.0+  
- CocoaPods 1.10.0+  
  
# Desktop  
- Windows 10 1903+ (para Windows)  
- macOS 10.14+ (para macOS)  
- Linux (Ubuntu 18.04+ recomendado)

### Producción

#### Servidor Backend

* \*\*URL\*\*: `http://52.53.127.245:8000`
* \*\*API\*\*: REST con endpoints JSON
* \*\*Autenticación\*\*: JWT Bearer tokens

#### Dispositivos Soportados

* \*\*Android\*\*: 5.0 (API 21) o superior
* \*\*iOS\*\*: 11.0 o superior
* \*\*Web\*\*: Navegadores modernos (Chrome, Firefox, Safari, Edge)
* \*\*Desktop\*\*: Windows 10+, macOS 10.14+, Linux Ubuntu 18.04+

## Instalación y Configuración

### 1. Configuración del Entorno de Desarrollo

#### Instalar Flutter

Código (bash):

# Descargar Flutter SDK  
git clone https://github.com/flutter/flutter.git -b stable  
export PATH="$PATH:`pwd`/flutter/bin"  
  
# Verificar instalación  
flutter doctor

#### Configurar Editor

Código (bash):

# VS Code  
code --install-extension Dart-Code.flutter  
  
# Android Studio  
# Instalar plugins: Flutter y Dart

### 2. Clonar y Configurar Proyecto

Código (bash):

# Clonar repositorio  
git clone https://github.com/leoromerbric/ganaderasoft\_app\_v1.git  
cd ganaderasoft\_app\_v1  
  
# Instalar dependencias  
flutter pub get  
  
# Verificar configuración  
flutter doctor  
flutter devices

### 3. Configuración de Base de Datos

#### SQLite Local

Código (dart):

// La base de datos se crea automáticamente en primera ejecución  
// Ubicación por defecto:  
// Android: /data/data/com.example.ganaderasoft\_app\_v1/databases/  
// iOS: ~/Library/Application Support/ganaderasoft.db  
// Desktop: ~/Documents/ganaderasoft.db

#### Configuración de Conexión API

Código (dart):

// lib/config/app\_config.dart  
class AppConfig {  
 // Configurar URL del servidor  
 static const String \_baseUrl = 'http://52.53.127.245:8000';  
   
 // Para desarrollo local, cambiar a:  
 // static const String \_baseUrl = 'http://localhost:8000';  
   
 // Para producción, cambiar a:  
 // static const String \_baseUrl = 'https://api.ganaderasoft.com';  
}

## Configuración por Plataforma

### Android

#### Configuración en `android/app/build.gradle`

Código (gradle):

android {  
 compileSdkVersion 33  
 ndkVersion flutter.ndkVersion  
  
 defaultConfig {  
 applicationId "com.ganaderasoft.app"  
 minSdkVersion 21  
 targetSdkVersion 33  
 versionCode 1  
 versionName "0.1.0"  
 }  
  
 buildTypes {  
 release {  
 signingConfig signingConfigs.debug  
 minifyEnabled true  
 proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  
 }  
 }  
}

#### Permisos en `android/app/src/main/AndroidManifest.xml`

Código (xml):

<manifest xmlns:android="http://schemas.android.com/apk/res/android">  
 <!-- Permisos de red -->  
 <uses-permission android:name="android.permission.INTERNET" />  
 <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />  
   
 <!-- Permisos de almacenamiento -->  
 <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  
 <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />  
   
 <application  
 android:name="${applicationName}"  
 android:icon="@mipmap/launcher\_icon"  
 android:label="Ganadera Soft">  
   
 <activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:launchMode="singleTop"  
 android:theme="@style/LaunchTheme">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN"/>  
 <category android:name="android.intent.category.LAUNCHER"/>  
 </intent-filter>  
 </activity>  
 </application>  
</manifest>

### iOS

#### Configuración en `ios/Runner/Info.plist`

Código (xml):

<dict>  
 <key>CFBundleName</key>  
 <string>Ganadera Soft</string>  
 <key>CFBundleIdentifier</key>  
 <string>com.ganaderasoft.app</string>  
 <key>CFBundleVersion</key>  
 <string>1</string>  
 <key>CFBundleShortVersionString</key>  
 <string>0.1.0</string>  
   
 <!-- Permisos de red -->  
 <key>NSAppTransportSecurity</key>  
 <dict>  
 <key>NSAllowsArbitraryLoads</key>  
 <true/>  
 </dict>  
</dict>

### Web

#### Configuración en `web/index.html`

Código (html):

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Ganadera Soft</title>  
 <link rel="manifest" href="manifest.json">  
 <link rel="icon" type="image/png" href="favicon.png"/>  
</head>  
<body>  
 <script src="flutter.js" defer></script>  
 <script>  
 window.addEventListener('load', function(ev) {  
 \_flutter.loader.loadEntrypoint({  
 serviceWorker: {  
 serviceWorkerVersion: null,  
 }  
 }).then(function(engineInitializer) {  
 return engineInitializer.initializeEngine();  
 }).then(function(appRunner) {  
 return appRunner.runApp();  
 });  
 });  
 </script>  
</body>  
</html>

## Variables de Entorno

### Configuración de Desarrollo

Código (dart):

// lib/config/environment.dart  
class Environment {  
 static const bool isDevelopment = true;  
 static const bool enableLogging = true;  
 static const Duration httpTimeout = Duration(seconds: 30);  
   
 static String get apiUrl => isDevelopment   
 ? 'http://localhost:8000/api'  
 : 'http://52.53.127.245:8000/api';  
}

### Configuración de Producción

Código (dart):

class Environment {  
 static const bool isDevelopment = false;  
 static const bool enableLogging = false;  
 static const Duration httpTimeout = Duration(seconds: 10);  
   
 static String get apiUrl => 'https://api.ganaderasoft.com/api';  
}

## Build y Deployment

### Comandos de Build

#### Debug Build (Desarrollo)

Código (bash):

# Android  
flutter build apk --debug  
flutter install  
  
# iOS  
flutter build ios --debug  
  
# Web  
flutter build web --debug  
  
# Desktop  
flutter build windows --debug  
flutter build macos --debug  
flutter build linux --debug

#### Release Build (Producción)

Código (bash):

# Android  
flutter build apk --release  
flutter build appbundle --release  
  
# iOS  
flutter build ios --release  
  
# Web  
flutter build web --release  
  
# Desktop  
flutter build windows --release  
flutter build macos --release  
flutter build linux --release

### Optimizaciones de Build

#### Reducir Tamaño de APK

Código (bash):

# Split APKs por arquitectura  
flutter build apk --split-per-abi  
  
# Obfuscación de código  
flutter build apk --obfuscate --split-debug-info=build/debug-info/

#### Optimizaciones Web

Código (bash):

# Build optimizado para web  
flutter build web --release --web-renderer html  
  
# Con tree-shaking  
flutter build web --release --tree-shake-icons

## Configuración de CI/CD

### GitHub Actions

#### Workflow para Testing

Código (yaml):

# .github/workflows/test.yml  
name: Test Suite  
  
on:  
 push:  
 branches: [ main, develop ]  
 pull\_request:  
 branches: [ main ]  
  
jobs:  
 test:  
 runs-on: ubuntu-latest  
   
 steps:  
 - uses: actions/checkout@v3  
   
 - name: Setup Flutter  
 uses: subosito/flutter-action@v2  
 with:  
 flutter-version: '3.8.1'  
 channel: 'stable'  
   
 - name: Install dependencies  
 run: flutter pub get  
   
 - name: Analyze code  
 run: flutter analyze  
   
 - name: Run tests  
 run: flutter test --coverage  
   
 - name: Upload coverage  
 uses: codecov/codecov-action@v3  
 with:  
 file: coverage/lcov.info

#### Workflow para Build Android

Código (yaml):

# .github/workflows/build-android.yml  
name: Build Android  
  
on:  
 push:  
 tags:  
 - 'v\*'  
  
jobs:  
 build:  
 runs-on: ubuntu-latest  
   
 steps:  
 - uses: actions/checkout@v3  
   
 - name: Setup Java  
 uses: actions/setup-java@v3  
 with:  
 java-version: '11'  
 distribution: 'temurin'  
   
 - name: Setup Flutter  
 uses: subosito/flutter-action@v2  
 with:  
 flutter-version: '3.8.1'  
   
 - name: Install dependencies  
 run: flutter pub get  
   
 - name: Build APK  
 run: flutter build apk --release  
   
 - name: Build App Bundle  
 run: flutter build appbundle --release  
   
 - name: Upload artifacts  
 uses: actions/upload-artifact@v3  
 with:  
 name: android-builds  
 path: |  
 build/app/outputs/flutter-apk/app-release.apk  
 build/app/outputs/bundle/release/app-release.aab

### Firebase App Distribution

#### Configuración

Código (bash):

# Instalar Firebase CLI  
npm install -g firebase-tools  
  
# Login a Firebase  
firebase login  
  
# Configurar proyecto  
firebase init hosting

#### Deploy Automático

Código (yaml):

# firebase.json  
{  
 "hosting": {  
 "public": "build/web",  
 "ignore": [  
 "firebase.json",  
 "\*\*/.\*",  
 "\*\*/node\_modules/\*\*"  
 ],  
 "rewrites": [  
 {  
 "source": "\*\*",  
 "destination": "/index.html"  
 }  
 ]  
 }  
}

## Configuración de Base de Datos

### Migración de Esquema

#### Sistema de Versiones

Código (dart):

class DatabaseMigrations {  
 static const Map<int, List<String>> migrations = {  
 2: [  
 'ALTER TABLE animales ADD COLUMN nueva\_columna TEXT',  
 ],  
 3: [  
 'CREATE TABLE nueva\_tabla (id INTEGER PRIMARY KEY)',  
 'UPDATE version SET number = 3',  
 ],  
 };  
   
 static Future<void> runMigrations(Database db, int oldVersion, int newVersion) async {  
 for (int version = oldVersion + 1; version <= newVersion; version++) {  
 if (migrations.containsKey(version)) {  
 for (String sql in migrations[version]!) {  
 await db.execute(sql);  
 }  
 }  
 }  
 }  
}

### Backup y Restauración

#### Script de Backup

Código (dart):

class DatabaseBackup {  
 static Future<void> exportToJson() async {  
 final db = await DatabaseService.database;  
   
 final backup = {  
 'version': await DatabaseService.getDatabaseVersion(),  
 'timestamp': DateTime.now().toIso8601String(),  
 'data': {  
 'users': await db.query('users'),  
 'fincas': await db.query('fincas'),  
 'animales': await db.query('animales'),  
 // ... otras tablas  
 }  
 };  
   
 final json = jsonEncode(backup);  
 // Guardar en archivo o enviar al servidor  
 }  
   
 static Future<void> importFromJson(String jsonData) async {  
 final data = jsonDecode(jsonData);  
 final db = await DatabaseService.database;  
   
 // Limpiar datos existentes  
 await db.delete('animales');  
 await db.delete('fincas');  
 // ... otras tablas  
   
 // Importar datos  
 for (final user in data['data']['users']) {  
 await db.insert('users', user);  
 }  
 // ... importar otras tablas  
 }  
}

## Monitoreo y Logging

### Configuración de Logging

Código (dart):

class LoggingConfig {  
 static void setup() {  
 if (Environment.isDevelopment) {  
 // Logging detallado en desarrollo  
 LoggingService.setLevel(LogLevel.debug);  
 } else {  
 // Logging mínimo en producción  
 LoggingService.setLevel(LogLevel.error);  
 }  
 }  
}

### Analytics y Crash Reporting

Código (bash):

# Agregar Firebase Analytics  
flutter pub add firebase\_analytics  
flutter pub add firebase\_crashlytics  
  
# Configurar en main.dart  
void main() async {  
 WidgetsFlutterBinding.ensureInitialized();  
 await Firebase.initializeApp();  
   
 FlutterError.onError = FirebaseCrashlytics.instance.recordFlutterError;  
   
 runApp(MyApp());  
}

## Seguridad

### Configuración de Red

Código (dart):

class SecurityConfig {  
 // Configurar certificados SSL  
 static void setupSSL() {  
 HttpOverrides.global = MyHttpOverrides();  
 }  
   
 // Validar certificados  
 static bool validateCertificate(X509Certificate cert, String host, int port) {  
 // Implementar validación personalizada  
 return cert.issuer.contains('Let\'s Encrypt') ||   
 cert.issuer.contains('DigiCert');  
 }  
}

### Obfuscación de Código

Código (bash):

# Build con obfuscación  
flutter build apk --obfuscate --split-debug-info=build/debug-info/  
  
# Mantener símbolos específicos  
# android/app/proguard-rules.pro  
-keep class com.ganaderasoft.\*\* { \*; }  
-keepattributes \*Annotation\*

## Troubleshooting

### Problemas Comunes

#### Error de Base de Datos

Código (dart):

// Limpiar base de datos corrupta  
static Future<void> resetDatabase() async {  
 final documentsDirectory = await getApplicationDocumentsDirectory();  
 final path = join(documentsDirectory.path, 'ganaderasoft.db');  
   
 final file = File(path);  
 if (await file.exists()) {  
 await file.delete();  
 }  
   
 // Reinicializar base de datos  
 \_database = null;  
 await database;  
}

#### Problemas de Conectividad

Código (dart):

// Diagnóstico de red  
static Future<void> diagnoseConnectivity() async {  
 final hasNetwork = await ConnectivityService.hasNetworkConnection();  
 final isServerReachable = await ConnectivityService.isConnected();  
   
 print('Network available: $hasNetwork');  
 print('Server reachable: $isServerReachable');  
   
 if (hasNetwork && !isServerReachable) {  
 print('Network available but server unreachable');  
 print('Check server URL: ${AppConfig.baseUrl}');  
 }  
}

### Comandos de Debugging

Código (bash):

# Limpiar cache de Flutter  
flutter clean  
flutter pub get  
  
# Verificar dependencias  
flutter pub deps  
  
# Analizar código  
flutter analyze  
  
# Ejecutar en modo debug verboso  
flutter run --debug --verbose  
  
# Profile performance  
flutter run --profile

## Documentación de APIs

Toda la documentación detallada de los endpoints de API se encuentra en:

* `apis\_docs/`: Documentación de endpoints individuales
* Formato: Archivos `.txt` con ejemplos de request/response
* Cobertura: Todos los endpoints utilizados por la aplicación

---

*Fin de la documentación. Para más información, consulte los archivos específicos en cada sección.*