




Manual Técnico

Temas	Manuales del Proyecto Dacky
Tipo	Manual Técnico
Estado	En progreso
Creado por	 Victoria Vielma
Fecha	@03/06/2025
Versión	Versión 2.0





DACKY - Aplicativo de Rastreo GPS para Mascotas

Descripción General

Dacky es una aplicación móvil diseñada para el rastreo GPS de mascotas, permitiendo a los dueños gestionar la información básica de las mascotas, su tarjeta virtual de vacunas y facilitar su recuperación en caso de extravío mediante un código QR en el collar.

Objetivo del Proyecto

Desarrollar una aplicación móvil con **Flutter** y un backend en **Flask** que permita a los usuarios:

-  Registrar información básica de las mascotas.
-  Generar una tarjeta de vacunación digital
-  Realizar seguimiento en tiempo real mediante GPS
-  Escanear un código QR para mostrar información de contacto en caso de pérdida

Equipo de Trabajo

Victoria Vielma - Desarrollador

Licencia

Este proyecto es propiedad exclusiva de **Victoria Saleck Adelaide Vielma Romero**.

Está protegido por una licencia propietaria en español e inglés. Consulta el archivo [LICENSE](#) para más información.

Objetivo del Manual Técnico

Documentar de forma técnica el funcionamiento y mantenimiento de la aplicación Dacky. A continuación se presentara las tecnologías y herramientas utilizadas, guía de instalación y configuración para trabajar en el proyecto, estructura de

la base de datos, control de versiones y anexos técnicos que explican fragmentos de código del proyecto.

Público Objetivo del Manual Técnico

- Técnicos
- Desarrolladores
- Futuros Mantenedores

Convenciones Utilizadas

▼ Paleta de Colores



▼ Tipografía

Google Fonts (Montserrat)

▼ Logo de la App



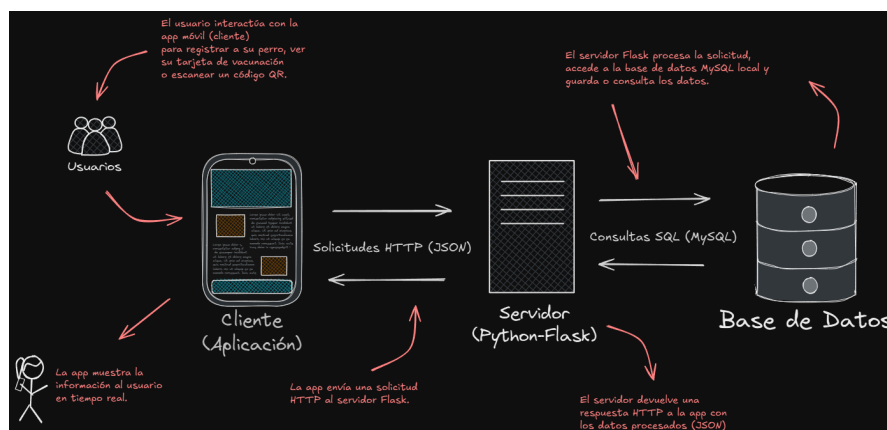
Arquitectura del Sistema

Arquitectura Cliente-Servidor

Dacky está basado en una arquitectura cliente-servidor, donde el cliente es la aplicación móvil que usan los dueños de mascotas, y el servidor es una aplicación web desarrollada en Flask (Python) que gestiona la lógica del sistema y la conexión con la base de datos MySQL.

Componente	Rol
Cliente	App Flutter en Android (o emulador), interfaz para el usuario
Servidor	Flask (Python), ejecutando lógica y gestionando la base de datos
Base de Datos	MySQL, almacena la información de mascotas, usuarios, vacunas, GPS
Red local o IP pública	Canal de comunicación entre Flutter y Flask

A continuación un gráfico que lo explica de manera visual:



Modelo-Vista-Controlador (MVC)

Dacky implementa el patrón de diseño Modelo-Vista-Controlador (MVC)

Modelo

El modelo, programa que almacena los datos, este representa las tablas de la base de datos mediante clases en Python usando SQLAlchemy.

```
Projecto Dacky > DackyApp > backend > models.py > ...
1 from flask_sqlalchemy import SQLAlchemy
2
3 db = SQLAlchemy()
4
5 class InicioSesion(db.Model):
6     __tablename__ = 'iniciosesion'
7     IdInicioSesion = db.Column(db.Integer, primary_key=True)
8     Nom = db.Column(db.String(100), nullable=False)
9     Apell = db.Column(db.String(100), nullable=False)
10    Email = db.Column(db.String(200), nullable=False)
11    Contraseña = db.Column(db.String(100), nullable=False)
12    NumTelf = db.Column(db.BigInteger, nullable=False)
13    NumCel = db.Column(db.BigInteger, nullable=False)
14    Direccion = db.Column(db.Text, nullable=False)
15    PerfilDueño_IdPerfilDueño = db.Column(db.Integer, db.ForeignKey('perfildueño.IdPerfilDueño'), nullable=False)
16
```

```
from flask_sqlalchemy import SQLAlchemy
```

```
db = SQLAlchemy()
```

```
class InicioSesion(db.Model):
    __tablename__ = 'iniciosesion'
    IdInicioSesion = db.Column(db.Integer, primary_key=True)
    Nom = db.Column(db.String(100), nullable=False)
    Apell = db.Column(db.String(100), nullable=False)
    Email = db.Column(db.String(200), nullable=False)
    Contraseña = db.Column(db.String(100), nullable=False)
    NumTelf = db.Column(db.BigInteger, nullable=False)
    NumCel = db.Column(db.BigInteger, nullable=False)
    Direccion = db.Column(db.Text, nullable=False)
    PerfilDueño_IdPerfilDueño = db.Column(db.Integer, db.ForeignKey('perfildueño.IdPerfilDueño'), nullable=False)
```

Vista

Vista, En una API REST, la vista está representada por las respuestas que el servidor Flask envía al cliente Flutter.



The image shows a mobile application interface for a registration screen. At the top, there is a back arrow and the title "REGISTRO". Below the title is a circular icon containing a stylized cat face. The screen contains five input fields: "Correo", "Nombre", "Apellido", "Contraseña", and "Repita Contraseña". The "Contraseña" and "Repita Contraseña" fields have a small eye icon to the right, indicating a password toggle. At the bottom, there is a dark button labeled "Registrarse".

Controlador

```

128 @app.route('/register', methods=['GET', 'POST'])
129 def register():
130     if request.method == 'POST':
131         Nom = request.form['userName']
132         Apell = request.form['userLastName']
133         Email = request.form['userEmail']
134         Contrasena = request.form['userPassword']
135         NumTelf = request.form['userPhone']
136         Direccion = request.form['userAddress']

```

Manual Técnico

```

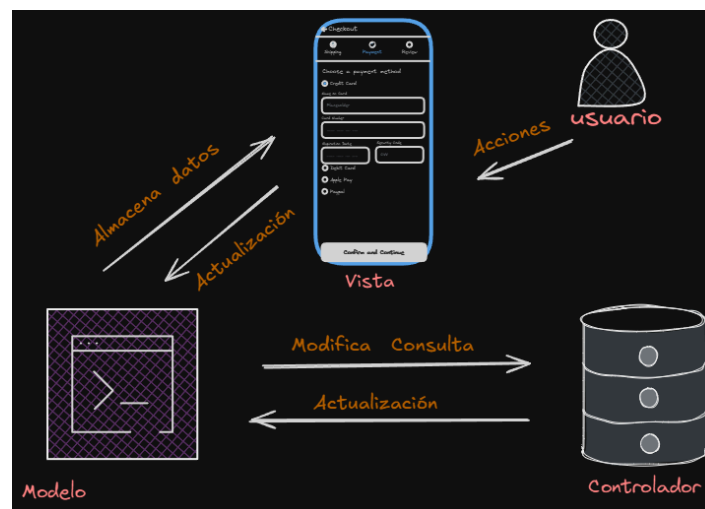
Direccion = request.form['userAddress']

hashed_password = generate_password_hash(Contrasena, method='pbkdf2:sha256')

db = None
cursor = None
try:
    db = conectar_db()
    if db is None:
        return jsonify({'message': 'Error interno del servidor (DB Connection)'}), 500
    cursor = db.cursor()

```

A continuación adjunto un diagrama visual de (MVC)



Tecnologías Utilizadas

- **Frontend:** Dart 3.6.1 (Flutter 3.27.2)
- **Backend:** Python 3.11+ (Flask)
- **Base de Datos:** MySQL (local)
- **Android Studio:** (con Android SDK)
- **Control de Versiones:** Git y GitHub
- **Docker Desktop** para Windows
- **Infraestructura:** Servidores en la nube (futuro)

Herramientas DevOps

- **GitHub** → Control de versiones
- **GitHub Actions** → Automatización CI/CD
- **Docker** → Contenedores para backend y frontend
- **Prometheus** → Monitoreo de métricas en Flask

Flujo de datos

(Explicación de como funciona el sistema)

Requisitos Técnicos del Sistema

Requisitos de hardware y software para instalación

General

- Sistema operativo: Windows (recomendado) o Linux
- Git instalado
- Conexión a internet

Frontend (Flutter)

- Flutter SDK `3.5.0` o superior
- Android Studio o VS Code con extensiones de Flutter y Dart
- Emulador Android o dispositivo físico
- Configuración de AVD (Android Virtual Device)

Backend (Python)

- Python `3.10` o superior
- XAMPP (Apache y MySQL)
- pip

Requisitos de red y servicios

- Abrir XAMPP y arrancar MySQL
- Ir a <http://localhost/phpmyadmin>
- Crear una base de datos llamada `dacky`
- Importar el archivo `dacky.sql` incluido en el proyecto

Dependencias del sistema

Archivo `requirements.txt` con las siguientes dependencias:

```
Flask==3.0.3
Flask-Bcrypt==1.0.1
flask-cors==5.0.1
Flask-Login==0.6.3
Flask-SQLAlchemy==3.1.1
Flask-WTF==1.2.2
mysql-connector-python==9.0.0
mysqlclient==2.2.4
PyMySQL==1.1.1
```

Estructura de la Base de Datos

Diagrama Entidad Relación

En este diagrama Entidad-Relación se representa cada tabla con sus respectivos atributos y como se relacionan entre si dentro del sistema de manera visual

IdDispositivoGPS	PK	1:1 mascota
Mascota_IdMascota	FK	

Instalación y Configuración

1. Clonar el repositorio

```
git clone https://github.com/smiling011/ProyectoDacky.git
cd "Proyecto Dacky"
```

Asegurarse de que las carpetas y archivos estén organizados así:

```
DackyApp/
├── backend/
│   ├── dacky.py
│   ├── config.py
│   ├── models.py
│   └── ...
├── frontend/
│   └── dacky_app/
│       ├── lib/
│       ├── assets/
│       └── pubspec.yaml
└── docker/
    ├── backend.Dockerfile
    ├── flutter.Dockerfile
    └── docker-compose.yml
```

2. Instalación y Configuración de Flutter

- Descargar Flutter desde el sitio oficial:
<https://docs.flutter.dev/get-started/install>
- Extraer la carpeta **flutter** en una ruta permanente, por ejemplo:

```
C:\src\flutter
```

- Agregar Flutter al **PATH del sistema**:
 - Buscar "Variables de entorno" en Windows.
 - Editar la variable **Path** del sistema.
 - Agregar:

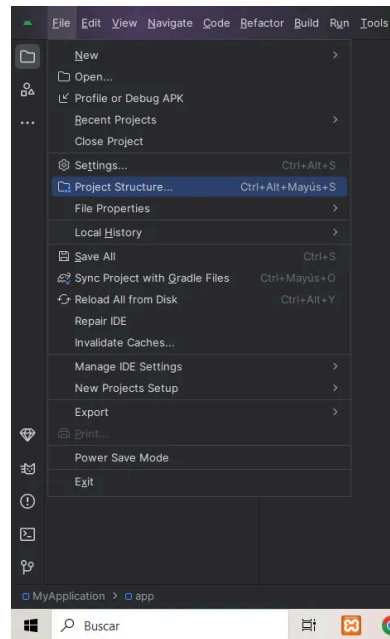

```
C:\src\flutter\bin
```
- Verificar instalación:

```
flutter doctor
```

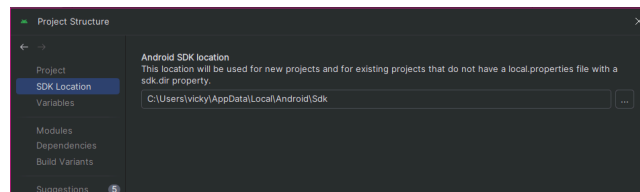
3. Instalación y Configuración de Android Studio

- Descargar desde:
<https://developer.android.com/studio>
- Durante la instalación, seleccionar los siguientes componentes:
 - Android SDK
 - Android SDK Platform-Tools

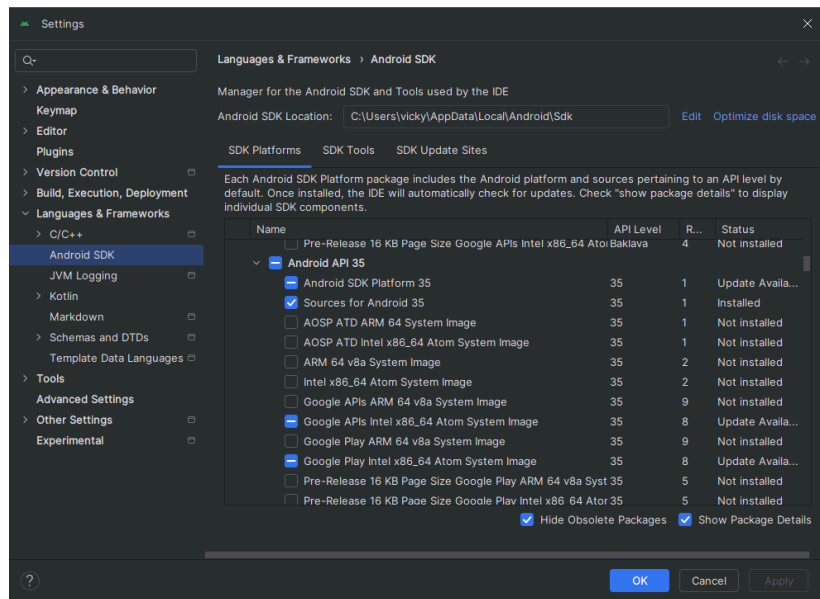
- Android Virtual Device (AVD)
- Una vez instalado:
 - Abrir Android Studio
 - **Verifica el SDK de Android**
- ▼ Despliega en Menú > File > Project Structure:



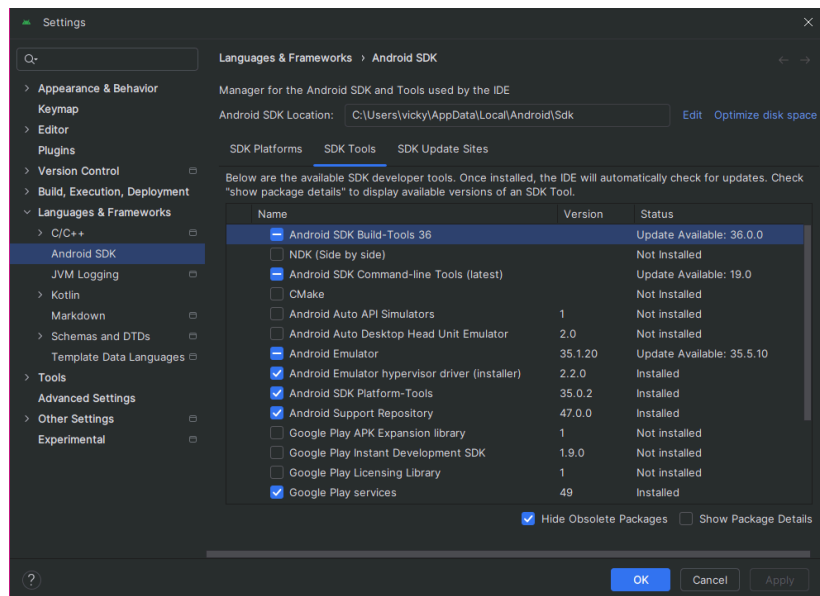
- ▼ Verifica si la ruta del SDK Location es correcta:



- Luego en Android Studio > **Settings** > **Appearance & Behavior** > **System Settings** > **Android SDK**
- Instala lo siguiente que:
- **Android API 35 o inferior**



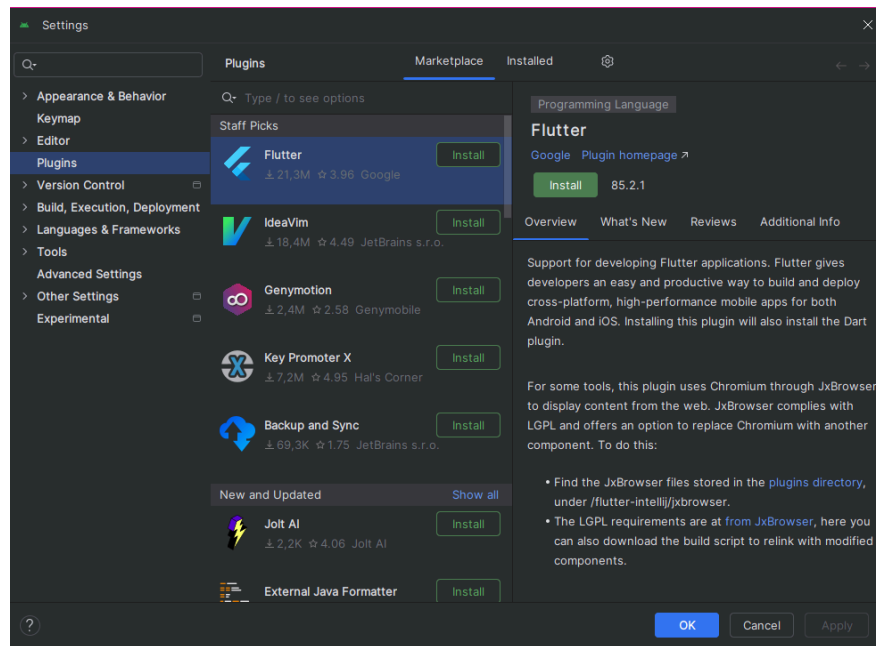
■ Android SDK Command-line Tools



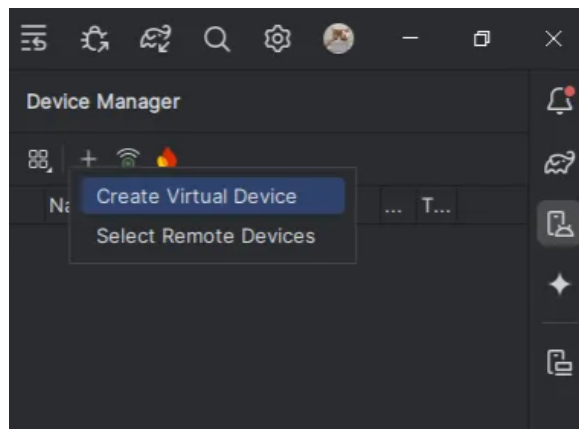
o Ir a: **Settings > Plugins > Marketplace**

o **Instalar el plugin Flutter**

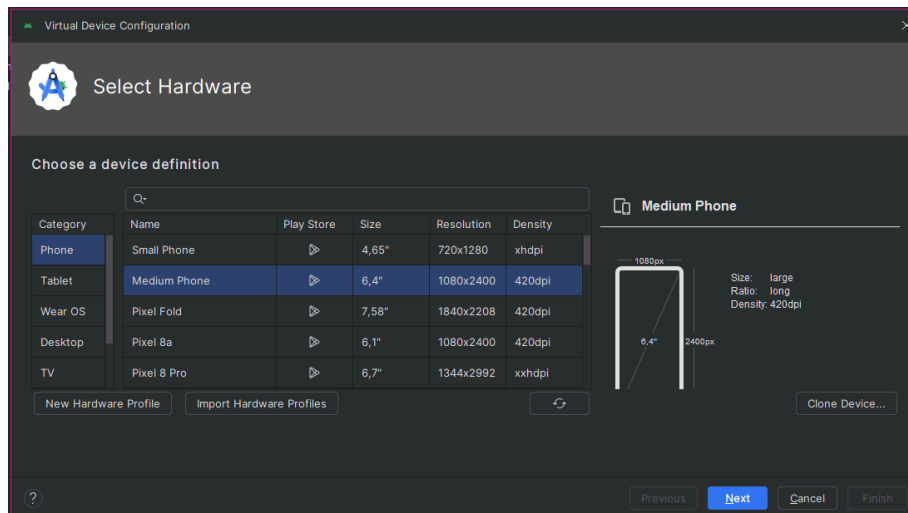
Esto también instalará el plugin Dart



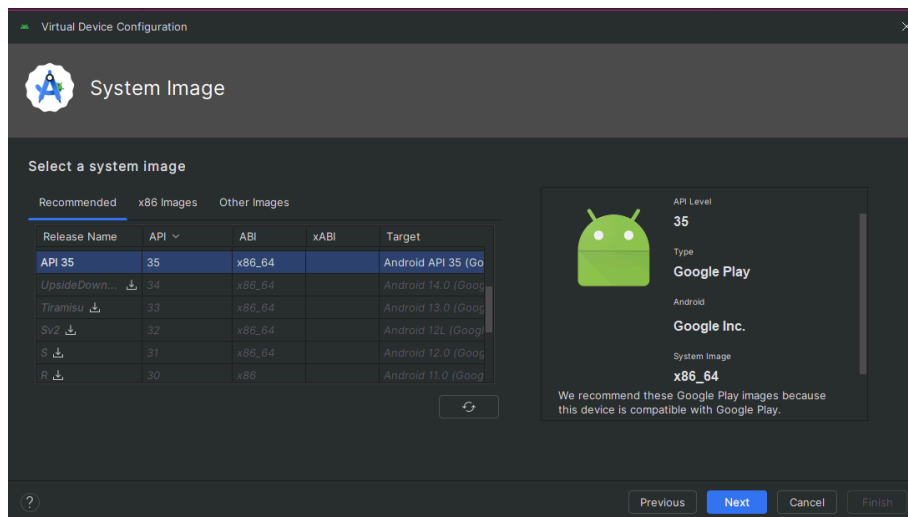
- Ir a: [More Actions > SDK Manager](#)
 - Verifica que tienes una versión reciente de SDK, por ejemplo: [Android 33](#)
- Crear un emulador:
 - Ir a [Device Manager > Create Device](#)



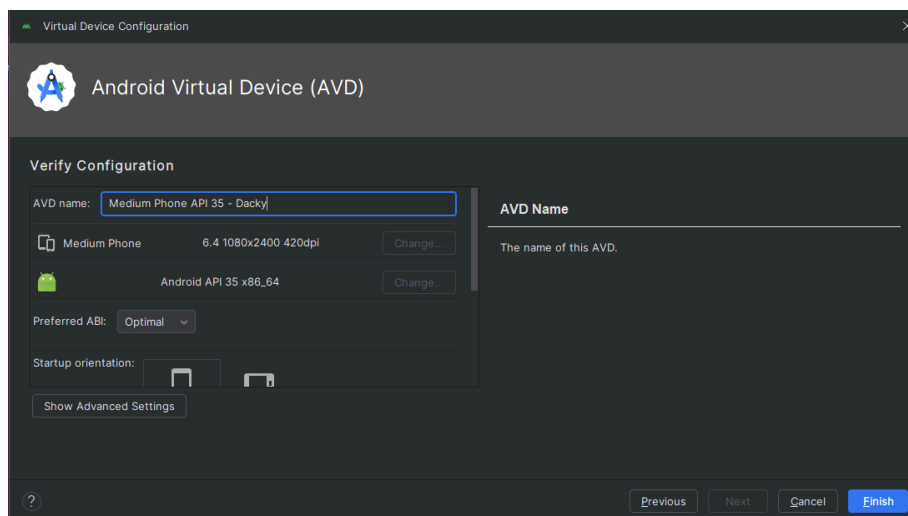
- Escoger el modelo (Medium Phone)



- Descargar imagen del sistema (API 30 o superior)



- Por ultimo nombrar a gusto AVD name y finalizar



4. Configurar Backend (Flask)

a. Crear entorno virtual (opcional)

```
python -m venv venv
venv\Scripts\activate # En Windows
```

b. Instalar dependencias

```
pip install -r requirements.txt
```

c. Configurar base de datos

- Abrir XAMPP y arrancar MySQL
- Ir a <http://localhost/phpmyadmin>
- Crear una base de datos llamada `dacky`
- Importar el archivo `dacky.sql` incluido en el proyecto

d. Archivo config.py (ejemplo)

```
SQLALCHEMY_DATABASE_URI = 'mysql+pymysql://root:(tu_contraseña)@localhost/dacky'
SQLALCHEMY_TRACK_MODIFICATIONS = False
```

e. Ejecutar servidor Flask

```
cd C:\Proyecto Dacky\DackyApp\backend (o ruta donde clonaste el Repositorio)
python dacky.py
```

Deberías ver: `Running on http://127.0.0.1:5000/`

5. Configurar Frontend (Flutter)

a. Abrir VS Code y entrar en la sección de **extensiones** (`Ctrl + Shift + X`):

- Instalar:
 - **Flutter**
 - **Dart**

b. Ir al directorio del proyecto Flutter

```
cd C:\Proyecto Dacky\DackyApp\frontend\dacky_app (o ruta donde clonaste el Repositorio)
```

c. Obtener paquetes

```
flutter pub get
```

d. Verificar emulador o dispositivo

```
flutter devices
```

e. Ejecutar la app

```
flutter run
```

Comunicación entre Flutter y Flask

- Si usas emulador Android: `http://10.0.2.2:5000`
- Si usas un dispositivo físico: usa tu IP local (ej. `http://192.168.1.10:5000`)

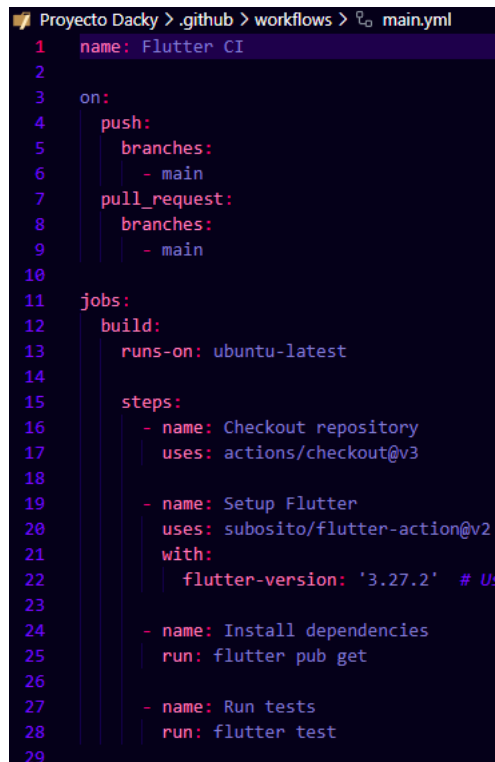
Pruebas Técnicas

En progreso

Durante el desarrollo de Dacky se realizaron **pruebas manuales** y algunas pruebas básicas con **Postman** para validar el correcto funcionamiento de las rutas del backend.

Tipos de pruebas realizadas:

- **Pruebas unitarias (manuales):** validación de funciones como registro, inicio de sesión y consulta de datos.
- **Pruebas de integración:** pruebas entre Flutter y Flask asegurando que las solicitudes HTTP respondan correctamente.
- **Pruebas funcionales:** asegurando que el flujo de registro de mascotas, visualización de la tarjeta de vacunas y lectura del código QR funcionen como se espera.
- **Integración Continua (CI) con GitHub Actions:** Se configuró un flujo de trabajo en GitHub Actions para verificar automáticamente que el backend se ejecute correctamente y que se instalen las dependencias necesarias en cada push o pull request. Esto permite detectar errores temprano y mantener la estabilidad del proyecto.



```
1 name: Flutter CI
2
3 on:
4   push:
5     branches:
6       - main
7   pull_request:
8     branches:
9       - main
10
11 jobs:
12   build:
13     runs-on: ubuntu-latest
14
15     steps:
16       - name: Checkout repository
17         uses: actions/checkout@v3
18
19       - name: Setup Flutter
20         uses: subosito/flutter-action@v2
21         with:
22           flutter-version: '3.27.2' # Us
23
24       - name: Install dependencies
25         run: flutter pub get
26
27       - name: Run tests
28         run: flutter test
29
```

Herramientas usadas:

- **Postman** para pruebas de las rutas del backend (**GET**, **POST**)
- **flutter run** y dispositivo/emulador para probar funcionalidades visuales

Seguridad del Sistema

En progreso

Actualmente, la app implementa medidas de seguridad básicas, y se tienen consideradas mejoras para futuras versiones.

Implementadas:

- **Validación de entradas** en formularios (correo, contraseñas, datos del perro).
- Separación cliente-servidor: el frontend no accede directamente a la base de datos.
- **Control de rutas seguras:** el backend Flask maneja cada solicitud validando los datos.
- **Contraseñas Cifradas:**

El sistema cifra las contraseñas de los usuarios usando `werkzeug.security` o una función equivalente, aplicando hashing seguro (como una función equivalente, aplicando hashing seguro (como `generate_password_hash`) para evitar almacenar contraseñas en texto plano.

Planeadas (futuras mejoras):

- **Autenticación con JWT** para asegurar el inicio de sesión.
- **Prevención de ataques comunes:**
 - SQL Injection: usando SQLAlchemy para evitar inyecciones.
 - CSRF y XSS: se implementarán al crear la versión web o autenticación avanzada.

Mantenimiento y Actualización

En progreso

Procedimiento para actualizar la app

1. Clonar la última versión del repositorio desde GitHub.
2. Actualizar dependencias:

```
flutter pub get
pip install -r requirements.txt
```

3. Reiniciar el servidor Flask y reiniciar la app en el emulador/dispositivo.

Backup de base de datos

1. Abrir phpMyAdmin.
2. Seleccionar la base `dacky`.
3. Ir a la pestaña "Exportar" → Formato: SQL → Descargar.

Restaurar base de datos

1. Ir a phpMyAdmin > pestaña "Importar".
2. Subir el archivo `.sql` exportado anteriormente.

Logs y depuración

- Flask muestra los errores en la terminal con detalles útiles para depuración.
- Se pueden agregar logs personalizados con `app.logger.info()` o `app.logger.error()`

Control de Versiones

Clonar repositorio (primera vez)

```
git clone https://github.com/smiling011/ProyectoDacky.git
```

Verificar el estado del repositorio

Antes de actualizar, revisa qué archivos han cambiado o no están en seguimiento:

```
git status
```

Agregar todos los archivos modificados y nuevos:

```
git add .
```

Confirmar los cambios (cada cambio debe tener un mensaje descriptivo):

```
git commit -m "Descripción del cambio realizado"
```

Subir los cambios a GitHub en la rama `main`, ejecuta:

```
git push origin main
```

Actualizar el repositorio local *(por si trabajas en varias PCs o en equipo)*

Si otros hicieron cambios en GitHub, antes de empezar a trabajar en tu código, descárgalos con:

```
git pull origin main --rebase
```

Notas Adicionales

- La app usa una base de datos MySQL alojada localmente
- El código QR redirige a una URL que podrá conectarse a un backend online
- Colores y tipografía personalizados según la marca Dacky
- Este manual es exclusivamente para la ejecución y configuración de la aplicación móvil

Futuras Mejoras

- Autenticación segura (JWT)
- Subida de imagen de la mascota
- Integración con GPS real y mapas
- Portal web para dueños y veterinarias
- Planes premium con publicidad personalizada

Contacto

Si tienes alguna duda o sugerencia, no dudes en escribir a:

victoriavielmaromero@gmail.com

Enlace del Repositorio

<https://github.com/smiling011/ProyectoDacky.git>

Enlace de la Documentación Técnica (Notion)

En Notion esta organizada toda la documentación original

 [Proyecto Dacky](#)