Instituto Tecnologico de Culiacan

Sistema de detección de placas vehiculares

Materia:

Topicos de ia

Grupo:

12:00 – 13:00

Alumnos:

Jesus Alberto Barraza Castro

Jesus Guadalupe Wong Camacho

Profesor:

Zuriel Dathan Mora Felix

Documentación técnica del proyecto de detección de placas vehiculares:

Este documento está dirigido a desarrolladores o administradores de sistemas que deseen realizar mantenimientos o cambios al proyecto.

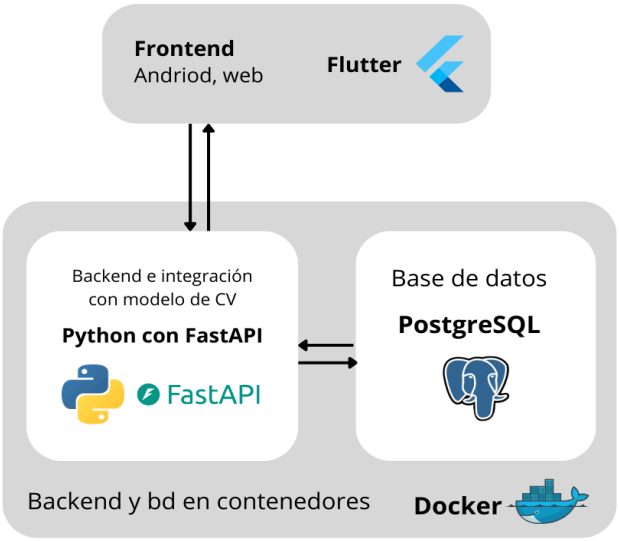
Arquitectura del sistema:

La aplicación fue diseñada con una arquitectura moderna y modular, separando claramente la capa de presentación de la lógica de negocio y la persistencia de datos. El sistema está diseñado para ser escalable y fácilmente desplegable mediante el uso de contenedores.

La arquitectura se compone de tres capas principales que interactúan entre sí:

1. **Frontend (Capa de Presentación):** Gestiona la interfaz de usuario y la interacción con el usuario final.
2. **Backend y Lógica de Negocio (Capa de Aplicación/Procesamiento):** Contiene la lógica central, incluyendo la integración del modelo de Visión por Computadora (CV) para la detección de placas.
3. **Base de Datos (Capa de Datos):** Responsable de la persistencia y gestión de la información.

Arquitectura de la aplicación



Stack de tecnologías utilizadas:

Módulo	Propósito	Tecnología Principal
Frontend	Interfaz de usuario para la interacción con el sistema. Permite el acceso desde múltiples plataformas.	Flutter (para desarrollo multiplataforma, se puede desplegar a la web, ios y android)
Backend	Servidor de aplicación que maneja las solicitudes del Frontend, ejecuta el modelo de CV y se comunica con la base de datos.	Python con FastAPI

Base de Datos	Almacena información persistente, como registros de placas, eventos de detección y datos de usuarios.	PostgreSQL
Contenerización	Contenerización y despliegue estandarizado y portable del ambiente y las dependencias del Backend y la Base de Datos.	Docker

Flujo de interacción:

1. El Frontend (Flutter) realiza peticiones al Backend (Python con FastAPI) a través de su API.
2. El Backend (FastAPI) procesa la solicitud:
    a. Si la solicitud implica el procesamiento de una imagen o video, activa el modelo de Visión por Computadora (CV) para detectar, segmentar y/o reconocer la placa.
    b. Para registrar un evento o consultar datos, se comunica directamente con la Base de Datos (PostgreSQL).
3. La Base de Datos (PostgreSQL) devuelve la información solicitada.
4. El Backend (FastAPI) formula la respuesta y la envía de vuelta al Frontend (Flutter).

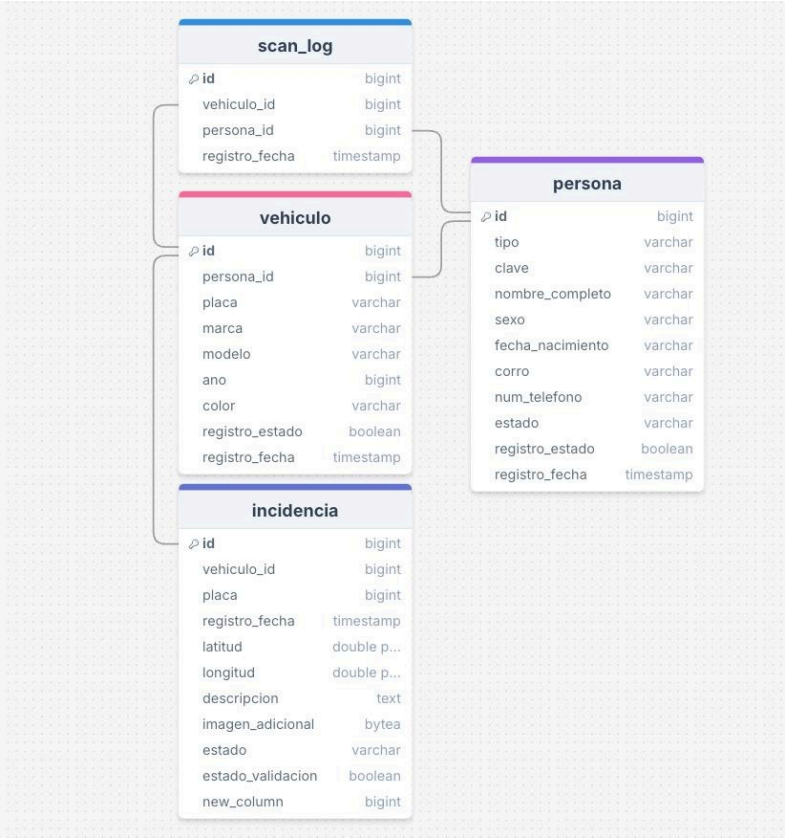
Despliegue:

Tanto el Backend como la Base de Datos se ejecutan dentro de contenedores Docker. Esto asegura que el entorno de ejecución sea idéntico en desarrollo,

pruebas y producción, simplificando la gestión de dependencias y el escalado del sistema.

Esquema de la base de datos:

Nuestro sistema de detección de placas utiliza PostgreSQL como motor de base de datos relacional. El diseño se centra en cuatro entidades principales, garantizando el registro de vehículos, personas, eventos de detección y logs de acceso.



A continuación, se detalla la estructura de las cuatro tablas principales:

Tabla: persona

Esta tabla almacena la información de las personas registradas, que pueden ser propietarios de vehículos.

Columna	Tipo de Dato	Descripción	Clave
id	bigint	Identificador único de la persona.	**PK**
tipo	varchar	Tipo de persona (ej. Alumno, Administrador).	
clave	varchar	Clave de acceso o identificación interna. (numero de control en caso del alumno)	
nombre_completo	varchar	Nombre y apellidos de la persona.	
sexo	varchar	Género de la persona.	
fecha_nacimiento	varchar	Fecha de nacimiento de la persona.	
corro	varchar	Correo electrónico de contacto.	

num_telefono	varchar	Número de teléfono de contacto.	
estado	varchar	Estado actual de la persona en el sistema (ej. Activo, Inactivo).	
registro_estado	boolean	Indicador de si el registro está activo o dado de baja.	
registro_fecha	timestamp	Fecha y hora en que se creó el registro.	

Tabla: vehiculo

Almacena los datos de los vehículos.

Columna	Tipo de Dato	Descripción	Clave
id	bigint	Identificador único del vehículo.	**PK**
persona_id	bigint	ID de la persona propietaria o asociada al vehículo.	**FK** a persona.id

placa	varchar	**Número de placa** del vehículo. Columna clave para la detección.	
marca	varchar	Marca del vehículo.	
modelo	varchar	Modelo del vehículo.	
ano	bigint	Año de fabricación del vehículo.	
color	varchar	Color del vehículo.	
registro_estado	boolean	Indicador de si el registro del vehículo está activo.	
registro_fecha	timestamp	Fecha y hora de registro del vehículo.	

Tabla: scan_log (Registro de Escaneos)

Registra cada intento o evento de escaneo de placa, ya sea exitoso o no, sirviendo como historial de actividad del sistema.

Columna	Tipo de Dato	Descripción	Clave
id	bigint	Identificador único del log.	**PK**
vehiculo_id	bigint	ID del vehículo detectado (si se identificó).	**FK** a vehiculo.id
persona_id	bigint	ID de la persona asociada al vehículo detectado.	**FK** a persona.id
registro_fecha	timestamp	Fecha y hora exacta del escaneo/detección.	

Tabla: incidencia

Esta tabla almacena los registros de incidentes capturados por el personal que usará la aplicación para llevar conteo de incumplimiento en el reglamento del estacionamiento.

Columna	Tipo de Dato	Descripción	Clave
id	bigint	Identificador único de la incidencia.	**PK**
vehiculo_id	bigint	ID del vehículo relacionado con la incidencia.	**FK** a vehiculo.id
placa	varchar	Placa detectada en el momento de la incidencia.	
registro_fecha	timestamp	Fecha y hora en que se registró la incidencia.	
latitud	double precision	Coordenada geográfica (latitud) de la detección.	
longitud	double precision	Coordenada geográfica (longitud) de la detección.	

descripcion	text	Descripción detallada de la incidencia.	
imagen_adicional	bytea	Almacenamiento de datos binarios (la imagen de la placa detectada o evidencia).	
estado	varchar	Estado de la incidencia (ej. Abierta, Cerrada, En Revisión).	
estado_validacion	boolean	Indicador de si la incidencia ha sido validada por un operador.	
new_column	bigint	Columna de propósito indefinido. **Nota:** Deberá ser renombrada o eliminada.	

Lógica de Negocio y Procedimientos Almacenados

Para optimizar el rendimiento de las operaciones complejas y la lógica de negocio crítica, el sistema utiliza funciones y procedimientos almacenados (PL/pgSQL) directamente en la base de datos PostgreSQL. Esto permite ejecutar lógica compleja (como la búsqueda de placas con fuzzy matching) más cerca de los datos.

Parámetro de Entrada	Tipo	Descripción
_data	jsonb	Objeto JSON que contiene la acción (AC) y los parámetros necesarios (ej. placa).

Casos de Uso (AC)	Descripción y Lógica
by_id	Búsqueda inteligente de Vehículo por Placa. Implementa una lógica sofisticada para compensar errores de reconocimiento de placa utilizando múltiples niveles de coincidencia como búsqueda exacta, búsqueda Limpia: Coincidencia de la placa sin caracteres especiales, similitud (Levenshtein): Uso de la función LEVENSHTEIN para encontrar placas con hasta 2 errores de diferencia, similitud por Patrón: Uso de la función SIMILARITY para patrones similares (similar a un *fuzzy search*).

get_logs	Recupera la lista de los últimos 100 registros de escaneo (scan_log), incluyendo la información detallada del vehículo y su propietario asociado.
get_vehicle_list	Devuelve la lista completa de todos los vehículos registrados y sus propietarios.
get_incidencia_list	Devuelve la lista completa de todas las incidencias registradas, ordenadas por fecha de registro descendente.

Estos procedimientos almacenados se encuentran en “database_scripts” en la raíz del repositorio.

Especificaciones de la API

Esta sección documenta la interfaz de comunicación entre el Frontend (Flutter) y el Backend (Python con FastAPI).

El Backend utiliza FastAPI para exponer una API RESTful, lo que garantiza un alto rendimiento.

Endpoints Clave

Módulo	Endpoint (Ruta)	Método HTTP	Descripción
Detección	/api/vehiculos/detect-plate/	POST	Recibe un archivo de imagen/video para el procesamiento por el modelo de CV. Devuelve el resultado del OCR y los datos del vehículo.
Vehículos	/api/vehiculos/read	POST	Llama al procedimiento almacenado read_vehiculos con la acción AC = 'by_id' para la búsqueda inteligente de una placa.
Incidencias	/api/incidencia/write/	POST	Registra una nueva incidencia en la base de datos.

Manual de instalación de entorno de desarrollo:

Este proceso describe los pasos para configurar el proyecto en una máquina local para desarrollo y pruebas, utilizando la contenerización de Docker para el Backend y la Base de Datos.

Requisitos de Software Iniciales

Antes de comenzar, asegúrese de que el entorno de desarrollo local tenga instalados los siguientes componentes:

1. Docker & Docker Compose: Necesario para levantar el Backend de FastAPI y la Base de Datos PostgreSQL en contenedores aislados.
2. Python 3.x: El lenguaje principal del Backend.
3. Flutter SDK: Necesario para compilar, ejecutar y probar el Frontend (Web, Android, iOS, etc.).
4. Git: Para clonar el repositorio del código fuente.

Obtención del Código Fuente

Clonar el Repositorio: Abra su terminal o símbolo del sistema, navegue hasta el directorio de trabajo deseado y clone el proyecto.

Estructura de Directorios: Verifique que la estructura del proyecto esté completa (ej. subdirectorios para backend y frontend).

Configuración y Arranque del Backend (Docker)

El Backend y la Base de Datos se gestionan mediante Docker Compose para garantizar un entorno consistente.

Levantar Contenedores: Desde el directorio que contiene el archivo docker-compose.yml, ejecute el siguiente comando:

docker-compose up -d --build

Ejecución del Frontend (Flutter)

1. Ingresar al directorio del frontend
2. Descargar las dependencias necesarias especificadas en el archivo “pubspec.yaml” utilizando el comando *flutter pub get*
3. Configurar la conexión al backend, ingresar el endpoint en la clase “api_service.dart”
4. Ejecutar la aplicación usando el comando *flutter run*, ya sea en un navegador web o un dispositivo android o ios