Sistema de Detección de Placas Vehiculares (LPR)

Alumnos: Omar Bermejo y Diego Araujo

Plataforma: iOS (SwiftUI) & Backend (FastAPI/Render)

1. Introducción

Este documento técnico describe la arquitectura, integración y flujo de datos de la aplicación móvil de Detección de Placas Vehiculares (LPR). La aplicación permite a los usuarios capturar imágenes de vehículos mediante la cámara o la galería, procesarlas a través de un servicio en la nube y gestionar la información resultante localmente.

El sistema se basa en una arquitectura Cliente-Servidor, donde el cliente es una aplicación nativa iOS desarrollada en **SwiftUI** y el servidor es una API RESTful desarrollada en **Python (FastAPI)** alojada en **Render**.

2. Objetivo del Sistema

El objetivo principal es proveer una interfaz móvil eficiente para el reconocimiento automático de matrículas, facilitando:

1. **Captura de Imágenes:** Interfaz nativa para uso de hardware de cámara.
2. **Procesamiento OCR:** Delegación de la carga computacional pesada a un servidor remoto.
3. **Gestión de Datos:** Almacenamiento local de placas detectadas junto con información de contacto del propietario (Nombre, Teléfono, Email).

3. Arquitectura del Backend

El backend está implementado como un microservicio stateless.

- **Hosting:** Render (PaaS).
- **Framework:** FastAPI (Python).
- **Endpoint Principal:**
  - **URL:** https://plate-backend-mcsd.onrender.com/detect
  - **Método:** POST
  - **Autenticación:** Pública (según implementación actual).

Lógica de Procesamiento

Al recibir una imagen, el backend:

1. Valida que el archivo sea una imagen válida.

2. Preprocesa la imagen (escala de grises, umbralización).
3. Ejecuta el modelo de inferencia (OCR/ANPR).
4. Retorna el texto más probable y un puntaje de confianza.

4. Arquitectura del Frontend (SwiftUI)

La aplicación iOS sigue el patrón **MVVM (Model-View-ViewModel)** para asegurar la separación de responsabilidades y la testabilidad.

4.1 Capa de Modelo (Model)

Define las estructuras de datos que coinciden con la respuesta JSON del servidor.

```
struct DetectionResponse: Codable {
   let success: Bool
   let plateText: String?
   let confidence: Double?
   let detail: String?
}
```

4.2 Capa de Vista (View)

Construida enteramente en SwiftUI.

- **ImagePicker:** UIViewControllerRepresentable que envuelve UIImagePickerController para acceder a cámara/galería.
- **ScannerView:** Vista principal que orquesta la captura y muestra estados de carga (ProgressView).
- **FormView:** Formulario para ingresar datos del dueño una vez que success == true.

4.3 Capa de ViewModel

El ScannerViewModel es responsable de la lógica de negocio:

- Mantiene el estado de la UI (@Published var image, @Published var detectionResult).
- Maneja la comunicación de red (URLSession).
- Construye el cuerpo multipart/form-data requerido por el backend.

5. Integración y Consumo del API

Estructura de la Petición

La comunicación se realiza vía HTTPS para garantizar la seguridad de los datos en tránsito.

Request:

- **Header:** Content-Type: multipart/form-data

- **Body:** Archivo de imagen binario, asociado a la clave file (o la clave específica definida por FastAPI, usualmente file o image).

Response (Ejemplo Real):
El backend retorna siempre un código de estado HTTP 200 (OK) si la petición llegó correctamente, delegando el estado del reconocimiento al campo success del JSON.


```
{
  "success": true,
  "plateText": "XYZ-987",
  "confidence": 0.98,
  "detail": "Plate detected"
}
```


6. Flujo de Datos

1. **Input:** Usuario toma foto -> UIImage.
2. **Conversión:** UIImage -> Data (JPEG Compression 0.8).
3. **Transporte:** URLSession envía POST a Render.
4. **Procesamiento:** Render ejecuta OCR.
5. **Output:** JSON recibido en app.
6. **Persistencia:** Si el usuario guarda, los datos se almacenan en UserDefaults o CoreData (según implementación de historial local).

7. Referencias

- Documentación de SwiftUI: Apple Developer
- FastAPI Documentation: FastAPI
- Render Hosting: Render.com