



INSTITUTO TECNOLÓGICO DE CULIACÁN

Ingeniería en Sistemas Computacionales

Tópicos de Inteligencia Artificial

Hora: 12:00 – 01:00 PM

Sistema de Detección de Placas vehiculares

Equipo:

Peña López Miguel Ángel

Robles Rios Jacquelin

Docente:

Mora Félix Zuriel Dathan

Culiacán, Sinaloa

15/11/2025

1 Índice

2	<i>Introducción.....</i>	3
3	<i>Objetivo.....</i>	4
3.1	Objetivos específicos	4
4	<i>Descripción del problema</i>	5
5	<i>Justificación</i>	6
6	<i>Desarrollo del sistema.....</i>	7
6.1	Base de datos	7
6.1.1	Esquema de la BD usando un diagrama EMR:	7
6.2	Modelo de Visión Artificial.....	8
6.2.1	Obtención.....	8
6.2.2	Pre-procesamiento.....	8
6.2.3	Selección de la arquitectura para el modelo.....	8
6.2.4	Definir parámetros de entrenamiento	8
6.2.5	Evaluar	9
6.3	Web Services.....	9
6.3.1	Proyecto de Python.....	9
6.3.2	Modelos.....	9
6.3.3	Servicios	10
6.3.4	Inicialización del proyecto	10
6.3.5	Despliegue en la web	11
6.4	MODEL SERVER	11
6.4.1	Objetivo y arquitectura.....	11
6.4.2	Servicios	11
6.4.3	Flujo de procesamiento y Optimización	12
6.4.4	Inicialización de Servidor	13
6.5	Aplicación con Android Studio	14
6.5.1	Propósito del Proyecto	14
6.5.2	Estructura y funcionalidad.....	14
6.5.3	Funcionamiento de la Aplicación:	15
6.5.4	Generación de APK o App.....	15
6.5.5	Prototipo de aplicación en Figma:.....	16
6.5.6	Aplicación ya desarrollada por Android Studio	18
7	<i>Conclusión</i>	19

2 Introducción

El Sistema de Detección de Matrículas es un proyecto integral el cual debe contener varias capas de aplicaciones para su funcionalidad, cuyo objetivo es desarrollar una solución tecnológica capaz de identificar y reconocer automáticamente las matrículas de los vehículos a partir de imágenes o transmisiones de video. Una vez detectada la matrícula, el sistema la asocia con la información de su respectivo propietario, almacenada en una base de datos.

3 Objetivo

Crear un sistema integral y eficiente de detección y reconocimiento de matrículas capaz de identificar los números de matrícula de los vehículos y vincularlos instantáneamente con la información de sus propietarios en una base de datos.

3.1 Objetivos específicos

- ❖ Creación de una base de datos que contenga los datos de los vehículos, usuarios, y propietarios de un reporte o incidencia que se realice en la aplicación
- ❖ Implementación de un sistema de vinculación o mejor dicho de un WebServices que funcione como el servidor de peticiones que se hagan de parte de los clientes que manejen la aplicación en cada uno de sus dispositivos
- ❖ Creación de un modelo de CNN (Convolutionan Neuronal Network) donde este tenga el objetivo de la detección de placas vehiculares al momento de tomar imagen.
- ❖ Creación de un servidor donde alojar el modelo del modelo para recibir peticiones y estar vinculado a su vez con la aplicación móvil
- ❖ Uso de extensiones o herramientas que ayuden a la decodificación de una imagen tomada por el modelo detector de placas para así utilizar un OCR (Optical Character Recognition) y saber cual es el valor de cada placa y vincularla con su propietario en la BD
- ❖ Creación de una aplicación móvil para que todos los usuarios puedan tener acceso a ella sin ningún problema ya contando dentro
- ❖ Elaborar documentación exhaustiva de: Manual de usuario y documentación técnica de instalación, así como su desarrollo del sistema para garantizar su entendimiento, usabilidad, mantenimiento y futura expansión del sistema

4 Descripción del problema

Actualmente en las ciudades con poblaciones donde los vehículos transitan en las vías públicas ya sean calles, autopistas, avenidas, se suele infringir diferentes tipos de incidencias vehiculares que alteran el orden público, por ejemplo en un estacionamiento el uso de lugares exclusivos para personas discapacitadas, Invadir zonas verdes para utilizarlas como estacionamiento o utilizar banquetas para estacionarse este tipo de incidencias para reportarlas suele ser un proceso nulo o manual y lento donde usualmente no se le aplica una sanción plena en sí. A la hora de la verificación rápida del propietario, control de accesos, gestión de estacionamientos o apoyo a la aplicación de la ley no es del todo eficaz, además no suelen ser escalables o los dejan pasar con el tiempo.

El problema que se aborda es la falta de un sistema automatizado, preciso y en tiempo real que integre la tecnología de visión artificial con una base de datos relacional para vincular una matrícula de un vehículo visible con la identidad de su propietario.

5 Justificación

El desarrollo del Sistema de Detección de Matrículas se justifica por su alto impacto potencial ya que en Seguridad y Aplicación de la Ley Permite una identificación vehicular rápida y precisa en casos de infracciones, vehículos robados o seguimiento de sospechosos, mejorando la capacidad de respuesta y la eficiencia de las autoridades.

El uso de procesos manuales siempre ha sido uno de los principales cambios que se busca con la tecnología, gracias al desarrollo de este reduce la dependencia de la intervención humana, minimizando los errores y liberando recursos que pueden destinarse a otras tareas.

La Innovación Tecnológica que se usa en el proyecto sirve como una excelente demostración práctica de la integración de tecnologías de **Inteligencia Artificial (Visión Artificial)** con sistemas de **Gestión de Bases de Datos**, aportando valor técnico y educativo.

6 Desarrollo del sistema

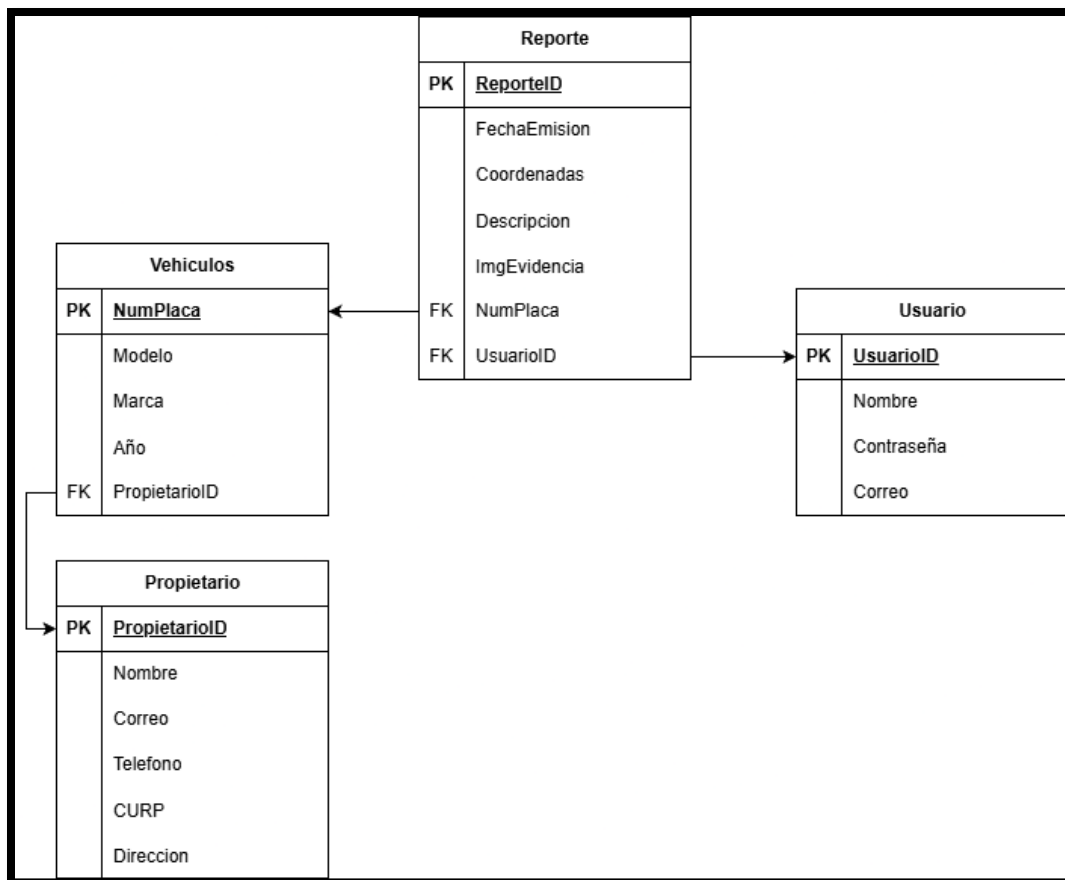
Para el desarrollo del software detector de placas se necesitan 3 capas importantes a abordar en este sistema:

6.1 Base de datos

Se necesita una correcta selección de un buen SGBD, para ello se decidió utilizar el padre de todas las consultas de queries que son óptimas para la web, PostgreSQL por su robustez, estabilidad y amplio soporte para hosteo de páginas web

Para que la BD funcione para todos los usuarios se ha buscado el hosteo de la BD dentro de la web usando páginas dedicadas al hosteo de proyectos, en este caso Render, donde maneja protocolos de certificación de seguridad de datos a la hora de hacer comunicación con otros componentes de proyectos

6.1.1 Esquema de la BD usando un diagrama EMR:



6.2 Modelo de Visión Artificial

6.2.1 Obtención

Para la obtención de un dataset de entrenamiento donde se encuentre fotos de placas vehiculares se toma en cuenta el uso de plataformas como Roboflow, donde ahí mismo se descargaron varios datasets donde venían alrededor de 800 imágenes de placas vehiculares

6.2.2 Pre-procesamiento

El dataset obtenido de parte de roboflow venia ya con aumentación de datos usando diferentes PDI o capas convolucionales como filtros y algunas rotaciones mínimas, cabe decir que ya venia ya con un archivo de “_anottations.txt” donde aquí se alojan las cajas delimitadoras de cada imagen donde se encuentran las placas, esto sirviendo como detector de características principales de las imágenes

6.2.3 Selección de la arquitectura para el modelo

Hay muchas arquitecturas pero la idea para el proyecto es la conocida como la de “Detector de objetos” donde hay ciertas arquitecturas y librerías en los lenguajes de programación de alto nivel que ayudan a enfocarnos en nuestros propósitos de proyectos: En nuestro caso se tomó la decisión de usar “YOLO (You Only Look Once)”, Esta arquitectura es la mas recomendada para aplicaciones en tiempo real o casi real, como la detección de matrículas, debido a su velocidad y eficacia.

6.2.4 Definir parámetros de entrenamiento

Parámetros Clave del Entrenamiento:

- **data='data.yaml':** Le dice a YOLOv8 dónde encontrar tu *dataset* de entrenamiento, validación y prueba. Este parámetro apunta al archivo de configuración YAML que creamos anteriormente, donde se definen las rutas de tus carpetas (train/images, valid/images, etc.) y el nombre de tu clase (placa).
- **epochs=20:** Especifica el número de **épocas** o ciclos completos de entrenamiento. El modelo revisará, aprenderá y ajustará sus pesos utilizando todas las imágenes de tu *dataset* de entrenamiento exactamente 20 veces.
- **imgsz=640: Función:** Define el tamaño al que todas las imágenes de tu *dataset* serán redimensionadas antes de ser alimentadas a la red. Usar un tamaño estandarizado (640x640 píxeles, por defecto en YOLO) es necesario para que el modelo funcione correctamente.

6.2.5 Evaluar

Para la correcta evaluación del modelo hay que tomar en cuenta las siguientes variables y seguir un estándar para ver si un modelo es correcto:

- mAP50: Mide si el modelo está logrando detectar la presencia de la placa de manera efectiva
- mAP50-95: Mide tanto la detección como la localización precisa de la caja delimitadora. Es la métrica más importante para el OCR
- P: precisión de todas las cajas delimitadoras del modelo que predijo que eran "placas"
- R: De todas las placas reales que existen en la imagen de validación, ¿Cuántas logró encontrar el modelo?

6.3 Web Services

Un **Web Service** es un método de comunicación que permite a dos sistemas de software, escritos en diferentes lenguajes y ejecutándose en diferentes plataformas, intercambiar datos a través de una red utilizando protocolos estándar como **HTTP** y **HTTPS**; Para nuestro proyecto de detección de placas y registro de reportes de placas de vehículos, el Web Service es necesario ya que permite enfocarse en atender peticiones de parte de la Aplicación, Para ello hay diferentes tecnologías que facilitan este propósito, He aquí la estructura del Web Service.

6.3.1 Proyecto de Python

Para dar comienzo al Web Services es necesario crear una aplicación que inicie el servidor y siempre esté escuchando a un puerto específico para estar abierto a peticiones que vengan de afuera y llegue por el puerto establecido como comunicación.

6.3.2 Modelos

Hay que darle la estructura a nuestro proyecto para que reconozca los objetos con los que maneja para hacer uso de la BD y a su vez recibirlo de la web. Para ello hay que crear los archivos siguientes dentro de la carpeta de models:

- ❖ Propietario.py
- ❖ Reporte.py
- ❖ Usuario.py
- ❖ Vehiculos.py

El objetivo principal de estos archivos es la representación de las tablas de la BD y guardarlas en clases que se puedan usar para la creación de API's en el Web Services en lugar de usar queries SQL como cadenas de texto, este término se le conoce **como ORM**

(Object Relational Mapping) Estos objetos son accesibles para Python, en lugar de escribir sentencias SQL puedes usar estos objetos que tienen propiedades SQL

6.3.3 Servicios

Carpeta donde irán aquellas API's o Servicios que brindará nuestro web services, estos deben ir separados por sus objetivos de servicio:

- ❖ propietarioService.py
- ❖ reporteServices.py
- ❖ usuarioServices.py
- ❖ vehiculoServices.py

Aquí dentro van aquellas responsabilidades de peticiones CRUD para cada clases, desde creación de usuarios, hasta levantamiento de reportes, además dejando el proyecto con la potencialidad de ser escalable agregando mas servicios o API's que ayuden a la aplicación,

Para la creación de API's se usa la librería "Flask" y "Database" donde estas permiten crear "planos o blueprints" que son el esqueleto de las peticiones o plantilla de los servicios para posteriormente hacerlos llamar cuando la app.py se vaya a iniciar y que estos servicios estén levantados y funcionando y a la espera de ser invocados por peticiones

6.3.4 Inicialización del proyecto

La clase más importante: "**app.py**" donde se ligán todos los servicios y se establece la cadena de conexión a la BD, junto con la acción de "abrir el puerto destino" donde estará a la espera de peticiones de la web

❖ Inicialización

Flask el framework principal para la construcción de la aplicación, se le asigna a una variable app, a esta misma se le establece que pueda recibir peticiones de navegadores externos con Flask_CORS, y la conexión y funcionalidad del ORM

❖ codificaciones de JSON UTF-8

Estándar para aceptar cualquier carácter con tildes o ñ

❖ Configuración de la BD

Usa la cadena de conexión que es llamada por la variable DATABASE_URL esta se establece en la pagina de hosteo para que pueda ser alojada la BD ahí mismo y conectarse a la BD Web

❖ Configuración ORM:

Carga la URL de conexión en la configuración de SQLAlchemy y con db puede inicializar el objeto ligado a la app usando: "db.init_app(app)"

- ❖ Importación de modelos


```
from models.Propietario import Propietario
from models.Vehiculos import Vehiculos
from models.Reporte import Reporte
from models.Usuario import Usuario
```
- ❖ Importación y registro de Servicios


```
from services.reporteServices import reporte_bp
from services.usuarioServices import user_bp
from services.vehiculoServices import vehiculo_bp
from services.propietarioServices import propietario_bp
```
- ❖ Ejecución del servidor


```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

6.3.5 Despliegue en la web

Para que el servicio de hosteo en Render funcione puede variar según la pagina de hosteo, en este caso solamente con la conexión del repositorio en Git y dos archivos llamados

- ❖ **Procfile**
Define el comando de inicio y el tipo de proceso principal de tu aplicación. Es esencial para entornos de producción, ya que le indica a Render qué comando exacto debe usar para que tu servidor Flask se ponga en marcha y escuche las peticiones en el puerto.
- ❖ **Requirements.txt**
 Aquí se le indica a Render que antes de arrancar debe instalar las dependencias que se encuentren dentro de este archivo

6.4 MODEL SERVER

6.4.1 Objetivo y arquitectura

El servidor que aloja un modelo de detector de objetos en la web se conoce generalmente como un **Servidor de Inferencia** o **Servidor de Modelos (Model Server)**.

Función: Cuando recibe una imagen a través de una solicitud HTTP/REST, ejecuta el modelo YOLOv8 y devuelve el resultado (las coordenadas de la placa y su etiqueta). El servicio implementa un flujo de **Detección de Dos Etapas** que combina la visión artificial con el Reconocimiento Óptico de Caracteres (EasyOCR)

6.4.2 Servicios

Solo tiene una única API

MÉTODO	API	FUNCION	ENTRADA	SALIDA
POST	/analizar	Procesa la imagen subida, detecta placa y extrae texto	Imagen que se agrega a temp: multipart/form-data	JSON: { "placa": "VLD817D", "confianza": 0.85 }

6.4.3 Flujo de procesamiento y Optimización

El código esta diseñado para ser eficiente en entornos de producción, especialmente en la gestión de memoria VRAM/RAM minimizando la latencia de la predicción.

FASE A: Recepción y Preprocesamiento de la Imagen para API

Paso	Código Clave	Explicación
Recepción	request.files['imagen']	El servidor recibe el archivo binario de la imagen a través de la petición POST.
Decodificación	cv2.imdecode(npimg, ...)	La imagen binaria se decodifica en un formato legible por OpenCV (NumPy Array), preparándola para el procesamiento visual.
Optimización	if width > 640: ... cv2.resize(...)	Optimización Crítica: Las imágenes de celulares son grandes y consumen mucha memoria. El código redimensiona la imagen a un ancho máximo de 640 píxeles. Esto reduce el consumo de RAM en órdenes de magnitud y acelera la inferencia de YOLO.

FASE B: Detección de objetos YOLOv8

Paso	Código Clave	Explicación
Inferencia	model = YOLO('best.pt'); results = model(frame)	Se carga el modelo YOLOv8 entrenado (best.pt) y se ejecuta la detección en la imagen pre-procesada. El modelo devuelve las coordenadas exactas de las placas detectadas.
Recorte (Crop)	crop = frame[y1:y2, x1:x2]	Se recorren los resultados y se utiliza OpenCV para recortar las regiones de interés (ROIs), aislando la placa.

Liberación de Memoria	del model; gc.collect()	Optimización Crítica: Las librerías de ML como ultralytics pueden retener la memoria de la GPU/RAM. Las llamadas explícitas a del y al <i>Garbage Collector</i> (gc.collect()) liberan los recursos de la GPU inmediatamente después de la detección, evitando saturación del servidor.
------------------------------	----------------------------	--

FASE C: Reconocimiento de Caracteres con EasyOCR

Paso	Código Clave	Explicación
Inicialización OCR	reader = easyocr.Reader(['es'], gpu=False)	Se carga la librería EasyOCR, configurada para el idioma español ('es').gpu=False para entornos de <i>hosting</i> compartido sin GPUs dedicadas.
Lectura de Texto	ocr_result = reader.readtext(crop)	EasyOCR lee los caracteres de la placa recortada.
Filtrado	if len(limpio) > 3 and conf > confianza_max	Se filtra el texto detectado, priorizando las detecciones de alta confianza y los fragmentos de texto con más de 3 caracteres (para descartar leyendas pequeñas), tomando el resultado con la confianza más alta de YOLO como definitiva.

6.4.4 Inicialización de Servidor

Así como el Web Services este también utilizara el uso de la librería como Flask para que este pueda inicializar un servidor y abrir sus puertos, haciendo posibles peticiones externas

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=7860)
```

6.5 Aplicación con Android Studio

6.5.1 Propósito del Proyecto

La aplicación móvil Android actúa como el Cliente de la solución completa. Su función principal es servir como interfaz de usuario para capturar la imagen del vehículo y enviarla al Model Server para capturar y decodificar la placa con los valores de la placa para así después crear un reporte y mandarlo al web services con la finalidad de que se refleje en la BD

El proyecto está diseñado para ser **modular** y **escalable**, siguiendo una estructura común de desarrollo en Android que separa las responsabilidades:

- **UI (Interfaz de Usuario):** Se encarga de mostrar la pantalla de captura, los resultados y el login.
- **Networking:** Se encarga de comunicarse con el Model Server y el Web Services hacia sus servicios.
- **Data:** Se encarga de la lógica de negocio y el manejo de datos

6.5.2 Estructura y funcionalidad

Carpeta/Archivo	Contenido Principal	Propósito en el Proyecto
data	ApiModels.kt	Contiene las clases de datos (Data Classes) de Kotlin que definen la estructura de la información que se intercambia con la API REST (ej: el formato JSON de la placa y la confianza).
network	ApiService.kt, RetrofitClient.kt, IService.kt	Es el módulo de comunicación. Utiliza Retrofit para manejar las peticiones HTTP (POST, GET) a tu Web Service alojado en Render. Define las interfaces para enviar la imagen y recibir el resultado ({ "placa": "..."}).
ui	Carpeta de theme, y la mayoría de los archivos .kt	Contiene todo lo relacionado con la Interfaz de Usuario y la lógica de presentación.

ui/theme	Color.kt, Theme.kt	Define la paleta de colores, tipografía y estilos visuales de la aplicación.
ui/...Activity.kt	HomePageActivity.kt, LoginActivity.kt, ReportActivity.kt, etc.	Son los Controladores de la Interfaz . Contienen la lógica para gestionar la interacción del usuario: tomar fotos, iniciar sesión, navegar entre pantallas y mostrar los resultados de la detección.
SessionManager.kt		Maneja el estado de la sesión del usuario (ej: ¿está logueado?).
res		Contiene todos los recursos no-código: <i>layouts</i> (XML para las vistas), imágenes, <i>strings</i> de texto y estilos.
AndroidManifest.xml		El archivo de configuración principal de Android. Define permisos (ej: acceso a la cámara y a internet) y los componentes de la aplicación.

6.5.3 Funcionamiento de la Aplicación:

- ❖ **Inicio de Sesión (LoginActivity):** El usuario se autentica.
- ❖ **Captura de Imagen (HomePageActivity):** El usuario utiliza la cámara del dispositivo para tomar una foto del vehículo.
- ❖ **Envío a la Nube:** La Activity llama a la capa **network**, que encapsula la imagen y la envía a la URL de tu **Servicio Web de Inferencia** (la API /analizar).
- ❖ **Procesamiento Remoto:** El servidor web (Flask con YOLOv8 y EasyOCR) procesa la imagen y devuelve la placa.
- ❖ **Recepción y Registro:** La capa **network** recibe el resultado JSON, lo convierte en un objeto Kotlin (ApiModels.kt), y la Activity lo muestra al usuario y gestiona el registro en la Base de Datos a través de los servicios de la API (si aplica).

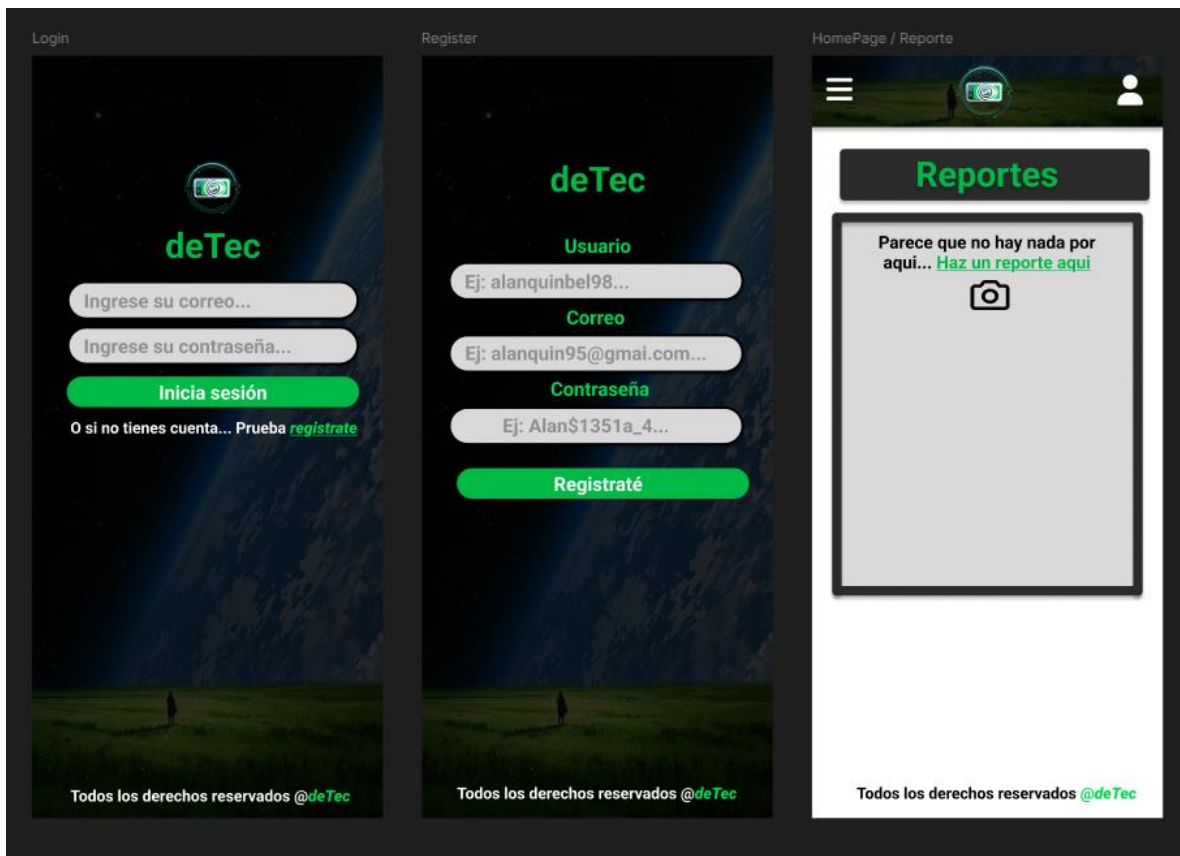
6.5.4 Generación de APK o App

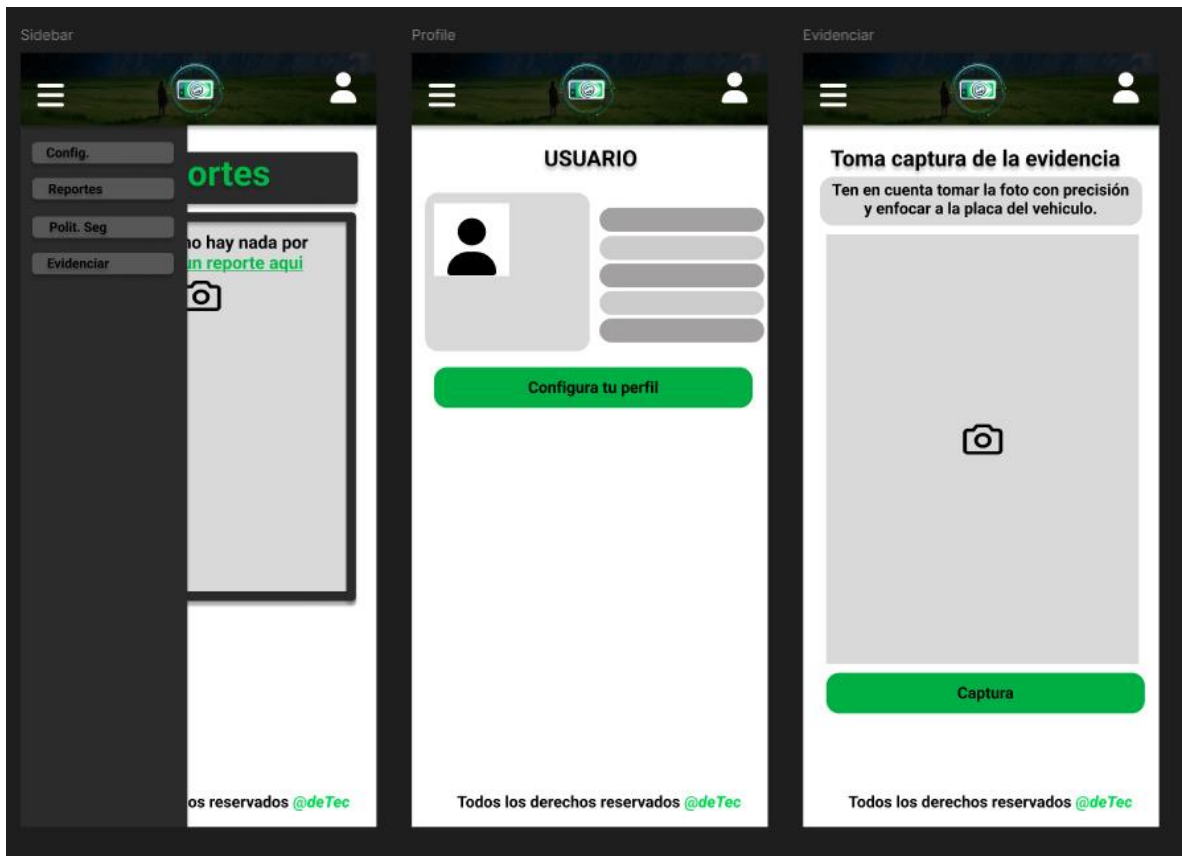
La principal ventaja de Android Studio es su capacidad para crear un paquete de instalación único y ejecutable: la **APK (Android Package Kit)**. En Android Studio, se utiliza el menú **Build > Build Bundles(s) / APK(s) > Build APK(s)**. Esto compila todo el código Kotlin, los recursos y las dependencias en un solo archivo binario.

Distribución a Usuarios:

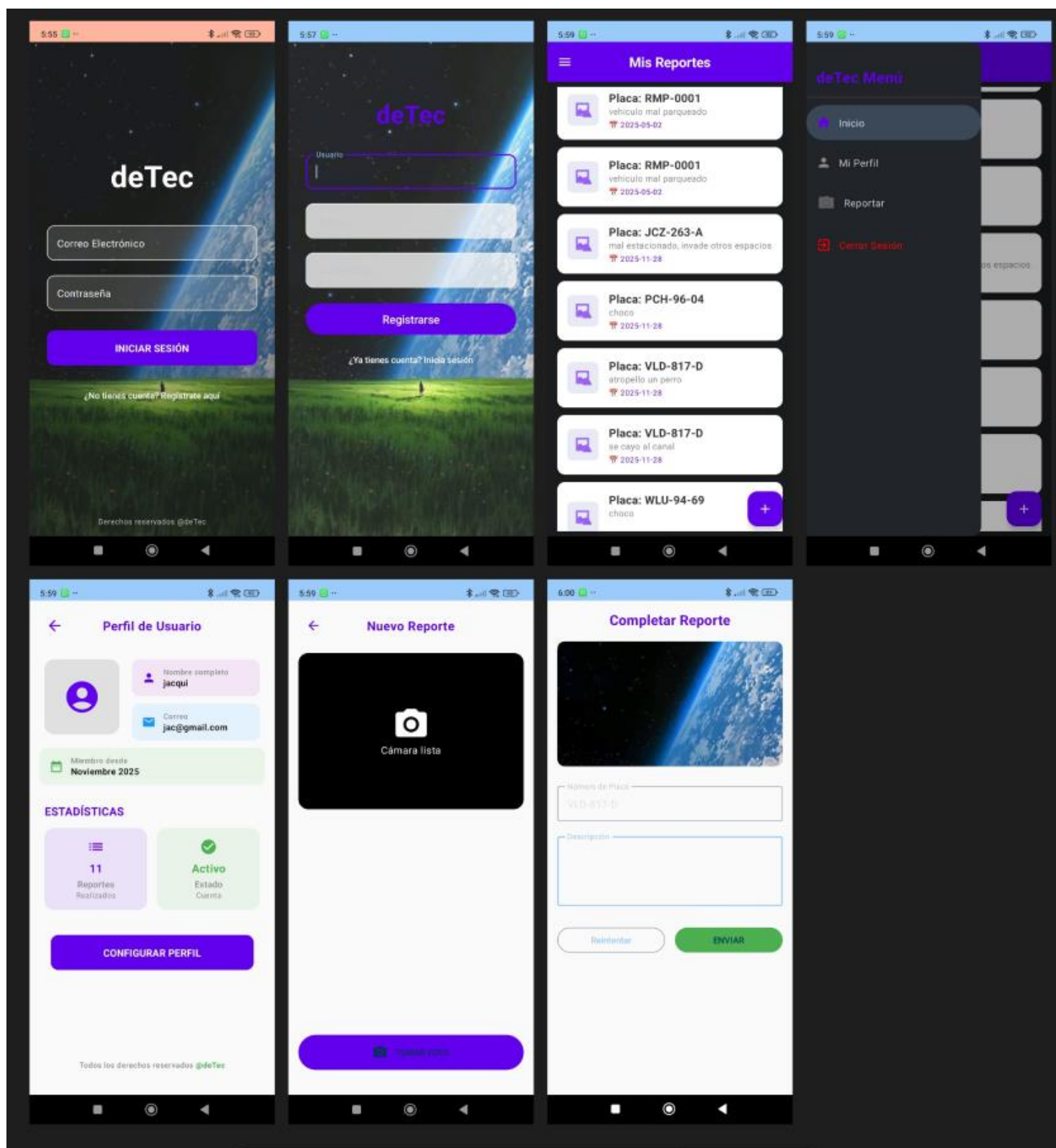
- **Fase de Pruebas (Testeo):** Simplemente se puede compartir el archivo .apk generado con los usuarios para que lo instalen directamente en sus dispositivos.

6.5.5 Prototipo de aplicación en Figma:





6.5.6 Aplicación ya desarrollada por Android Studio



7 Conclusión

El desarrollo del **Sistema de Detección de Placas Vehiculares** no solo cumple su objetivo de automatizar el reconocimiento de matrículas, sino que establece una solución tecnológica completa, eficiente y lista para producción, superando el proceso manual e ineficiente que justificó su creación. A demás de que la aplicación deja una iniciativa escalable para en un futuro la adición a nuevas funcionalidades o estadísticas que se quieran medir de parte de la aplicación, usuarios, vehículos, regiones, ciudades, etc.

El uso de una arquitectura robusta y bien pensada nos hace sentir orgulloso de la buena planificación de este proyecto y como este es un buen punto de partida para la publicación de un buen proyecto que puede ayudar a la resolución de diferentes tipos de problemáticas.