الجمهورية الجزائرية الديمقراطية الشعبية
Democratic and Popular Algerian Republic
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جـامـعـة الشهيد حمه لخضر الـوادي
Echahide Hamma Lakhdar University – El-Oued

Faculty of Exact Sciences — كلية العلوم الدقيقة

Department of Computer Science — قسم الإعلام الآلي

Thesis

Presented for the attainment of a license
Degree in computer science

Presented By: Bekkari Haithem
Drihem Abdelmoumen

Theme

A mobile app for police to track vehicle status

Presented on ... - ... - 2025

Professor : Meftah Sharaf Eddine	MCA	President
Professor : Bouhamed Mohammed Mounir	MAA	Decided
Professor : Mohammed Amine Yagoub		Supervisor

Academic Year: 2024/2025

PolicePlus

شكر وتقدير

أود أن أعبر عن خالص شكري وتقديري لإدارة الكلية وأعضاء هيئة التدريس الذين قدموا لي المعرفة والإرشاد و البيئة الأكاديمية المناسبة لإعداد هذا البحث. كما أخص بالشكر مشرفي الأكاديمي

الدكتور **محمد أمين يعقوب** على دعمه المستمر وإرشاداته القيمة.

لا يسعني إلا أن أشكر عائلتي العزيزة التي كانت مصدر دعمي وإلهامي خلال هذه الرحلة، وأصدقائي الذين وقفوا بجانبي دائمًا. كما أشكر كل من ساهم في إنجاح هذا البحث، سواء بالنصيحة أو المساندة.

Summary

Our project aims to design and develop a mobile application dedicated to assisting police officers in efficiently tracking and managing vehicle status in real time. The goal is to create a reliable tool that simplifies the verification of vehicle information, facilitates the detection of violations, and supports more effective law enforcement interventions.

The project began with a thorough analysis of existing vehicle tracking and law enforcement systems, identifying key challenges and areas for improvement. This groundwork helped us define the application's core features and user experience. UML diagrams were used to model the architecture and interactions within the system.

We adopted Kotlin with Jetpack Compose for building a modern and intuitive user interface, and utilized a Node.js backend connected to a MySQL database to handle data management and communication. The application provides functionalities such as instant access to vehicle records, anomaly reporting, and the verification of administrative documents.

The proposed solution empowers police officers with a streamlined and responsive tool, enhancing operational efficiency, ensuring quick access to critical data, and contributing to more organized and informed field operations.

Keywords: Mobile Application, Police, Vehicle Management, vehicle Status Tracker, Kotlin, Node.js, Express.

Contents

List of Figures

GENERAL INTRODUCTION

Today, the world is witnessing significant technological advancements in all sectors, particularly in road safety and vehicle tracking. Modern technologies now enable law enforcement agencies to use mobile applications to monitor and manage vehicle status in real time.

Police officers often face numerous challenges, such as quickly identifying vehicles, verifying documents, and detecting infractions. Thanks to computing and web services, it is now possible to enhance the efficiency of these operations by automating the process of vehicle control and tracking.

In this context, we have been led to design, develop, and implement a mobile application dedicated to vehicle management by the police. This application aims to centralize information, provide instant access to vehicle data, and facilitate officers' interventions in the field.

Our application is designed to improve vehicle management and optimize law enforcement operations to enhance the reliability of inspections, reduce response times, and minimize human errors.

This project consists of three chapters presented as follows:

- The first chapter is dedicated to the theoretical presentation and functional requirements.
- Chapter II provides several UML diagrams and specifications.
- Chapter III allows for the visualization and understanding of different system interfaces.

Finally, we will conclude this report with a summary of the work accomplished and an introduction to future perspectives for this project.

SECTION I
SYSTEM CONTEXT

1. Introduction

This chapter provides a general overview of the work context and the objectives of our final year project. We will begin by outlining some of the challenges faced by law enforcement and the importance of an information system for police operations.

1.1.The Importance of Police in Taking Security Measures

The police play a crucial role in ensuring road and vehicle safety by implementing security measures to reduce violations related to vehicle conditions. Ensuring that vehicles are roadworthy is essential to prevent accidents caused by mechanical failures or the deterioration of critical components.

The police conduct regular inspection campaigns to detect vehicles that do not meet legal standards, such as those with faulty brakes, lighting, or tires, which may pose risks to drivers and pedestrians. They also impose fines and penalties on violators to ensure compliance with laws designed to enhance road safety.

In addition, the police contribute to raising awareness among drivers about the importance of regular vehicle maintenance through educational and awareness programs. They also utilize modern technologies, such as smart cameras and electronic inspection devices, to monitor vehicle conditions and effectively detect violations.

Thus, the role of the police in enforcing security measures is not limited to imposing laws but also includes promoting a culture of road safety awareness, which helps reduce accident rates and ensures a safer traffic environment for everyone.

1.2.How the Police Monitor Vehicle Condition and Detect Violations

Monitoring the condition of vehicles is a key responsibility of the police to ensure road safety and reduce accidents. Police officers rely on a set of documents and standards to determine whether a vehicle complies with traffic laws. This document highlights five main factors that help the police monitor vehicle condition and detect potential violations.

1.2.1. Driver's License

The driver's license is an essential document that drivers must vehiclery while driving. It includes important details such as the license number, nickname, and name of the driver. The police verify the validity of the license to ensure that the driver is legally qualified to operate a vehicle. If the license is expired or invalid, it is considered a violation.

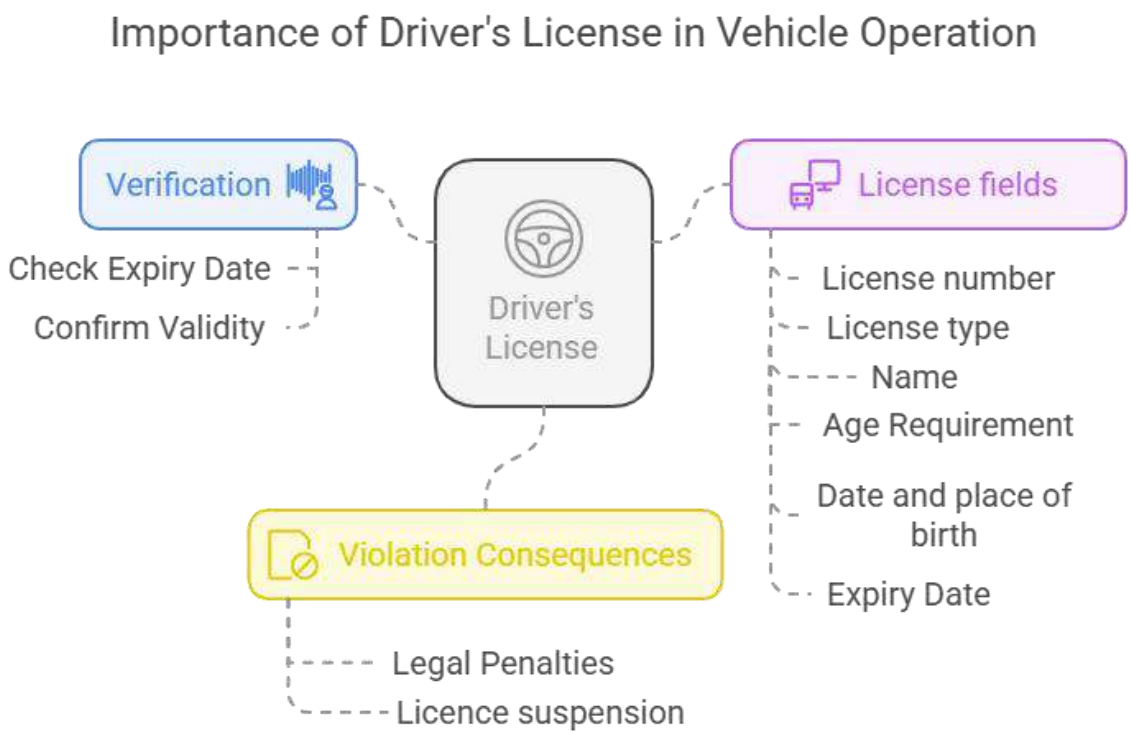


Figure I Driver License Validation Process

Field	Description	Type
License Number	10 digits unique integer number	INTEGER
License type	Driving License Category	VARCHAR
Name and Family Name	The Individual Name and Family Name	VARCHAR
License Expiry Date	it indicates when the license will expire.	DATE

a driver’s license (permis de conduire) has an expiration date and must be renewed periodically, depending on the license category and the driver's age.

1.2.2. License Validity

- Standard Driving Licenses (Category A & B) Valid for 10 years.
- Renewal required every 10 years until age 60. After age 60, renewal may be required more frequently (e.g., every 5 years), often with a medical check-up.
- Professional or Heavy Vehicle Licenses (Category C, D, E) Valid for 5 years. Require regular medical examinations to renew.

1.2.3. Vehicle Registration Certificate

The vehicle registration certificate (also known as the gray vehicled) proves ownership of the vehicle and contains important information such as the matriculation. The police check this document to confirm that the vehicle is legally registered and not stolen. Any discrepancies or tampering with the information can result in fines.

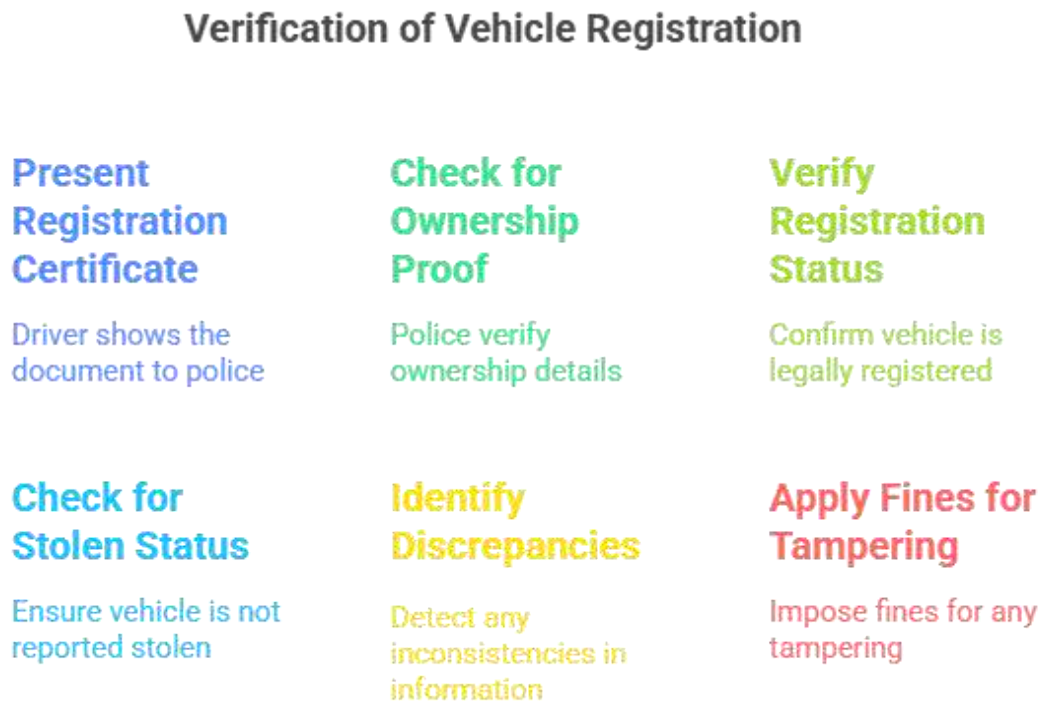


Figure II Figure Showcasing Vehicle Registration verification Process

Field	Description	Type
Date of Issue	When the current certificate was issued.	DATE
Number of Seats	Total number of authorized seats in the vehicle.	INTEGER
Chassis Number	Same as the VIN, sometimes shown separately.	VARCHAR
Use Type	Private, commercial, rental, taxi, etc.	VARCHAR
Inspection Validity	Date until which the vehicle's technical inspection is valid.	DATE

Registration Number	Unique matriculation number (e.g., 1234 - 116 - 25) identifying the vehicle.	INTEGER
Owner's Full Name	Name of the person or entity that owns the vehicle.	VARCHAR
Owner's Address	address of the vehicle owner.	VARCHAR
Vehicle Identification Number (VIN)	A unique 17-character code identifying the vehicle.	VARCHAR
Make and Model	Manufacturer and model of the vehicle	VARCHAR
Type of Vehicle	Classification (e.g., passenger, truck, motorcycle).	VARCHAR
Engine Power	Measured in horsepower (HP) or kilowatts (kW).	INTEGER
Fuel Type	Type of fuel used (e.g., petrol, diesel, LPG).	VARCHAR
Date of First Registration	When the vehicle was first registered (locally or abroad).	DATE
Date of Issue	When the current certificate was issued.	DATE
Use Type	Private, commercial, rental, taxi, etc.	VARCHAR
Inspection Validity	Date until which the vehicle’s technical inspection is valid.	DATE

Figure I.3: Figure Showcasing Vehicle Registration Certificate (Gray vehicled) – Key Fields

1.2.4. Insurance Certificate

A valid insurance certificate is a legal requirement for drivers. It includes important details such as the validity period, specified as "Valid from Date to Date." The police verify the presence of an active insurance policy, as driving without insurance is a legal violation. Insurance protects both drivers and pedestrians in the event of an accident.

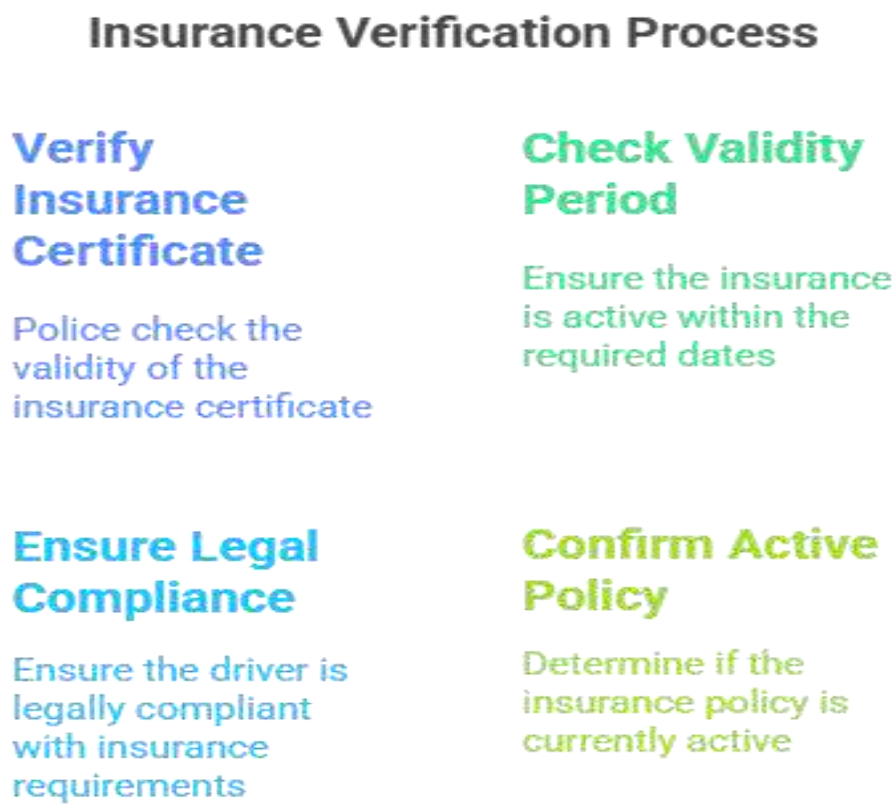


Figure III Showcasing Insurance Verification Process

Field	Description	Type
Registration Number	A unique identifier assigned to the vehicle	VARCHAR
Vehicle Category	Indicates the type of vehicle	VARCHAR
Year of Manufacture	The year the vehicle was manufactured.	DATE
Wilaya Code	A two-digit code representing	INTEGER
Owner's Full Name	The registered owner's name.	VARCHAR
Owner's Address	Residential address of the owner.	VARCHAR
Vehicle Identification Number (VIN)	A unique 17-character code used to identify the vehicle.	VARCHAR
Chassis Number	A unique identifier for the vehicle's chassis.	INTEGER
Engine Number	A unique identifier for the vehicle's engine.	VARCHAR

Number of Seats	Total seating capacity of the vehicle.	INTEGER
Use Type	Specifies whether the vehicle is for private or commercial use.	VARCHAR
Technical Inspection Validity	The date until which the vehicle's technical inspection is valid.	DATE
Registration Date	The date when the vehicle was first registered.	DATE
Issuing Authority	The government body that issued the registration.	VARCHAR

Figure I.4: Figure Showcasing key Fields for Insurance Certificate

1.2.4.1. Legal Importance

Driving without valid insurance in Algeria is illegal and can lead to significant penalties, including fines and vehicle impoundment. The police routinely check for active insurance coverage during traffic stops. Moreover, insurance protects both drivers and pedestrians in the event of an accident, ensuring that financial compensation is available for damages and injuries.

1.2.5. Periodic Technical Inspection

Laws in many countries require vehicles to undergo regular technical inspections to ensure their safety and efficiency. The police check for a valid technical inspection certificate, which includes details such as the Subsequent Monitoring Date, to confirm that the vehicle meets the required mechanical and safety standards. Failing to undergo periodic inspections can result in penalties.

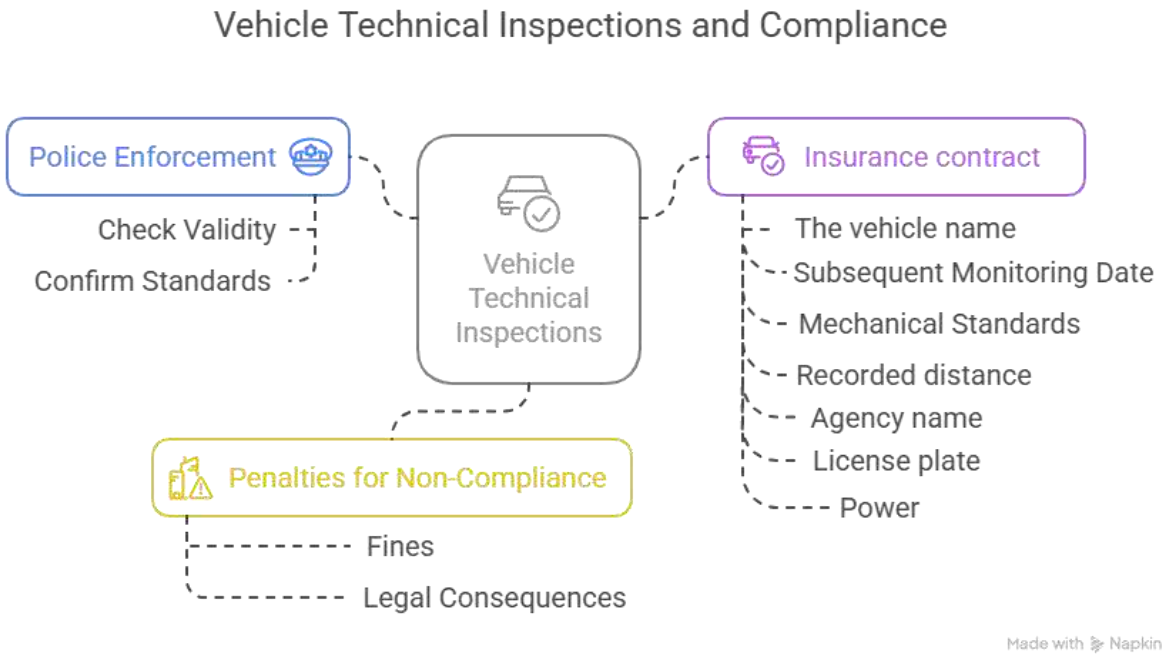


Figure IV Overview for Technical Inspection and Compliance

1.2.5.1. Description of the Periodic Technical Inspection (Contrôle Technique)

The Periodic Technical Inspection (PTI) in Algeria is a mandatory vehicle examination required by law to ensure that vehicles operating on public roads meet established safety and environmental standards. As illustrated in the diagram, this inspection is a central component of vehicle compliance.

- **Purpose**: The primary goal is to verify the roadworthiness of a vehicle, checking critical components like brakes, steering, suspension, lights, tires, emissions, and chassis integrity against defined "Mechanical Standards." This contributes to road safety and environmental protection.
- **Mandate & Frequency**: Inspections are typically required periodically (e.g., annually for older vehicles or commercial vehicles, biennially for newer private vehicles after an initial period). The specific frequency depends on the vehicle's age and type.

1.2.5.2. Enforcement & Compliance

- **Police Enforcement**: Law enforcement authorities conduct checks to "Check Validity" of the inspection certificate and "Confirm Standards" are met. Driving without a valid PTI certificate is illegal.
- **Insurance**: A valid PTI is often a prerequisite for obtaining or renewing an "Insurance contract." Insurance companies may verify the inspection status ("The vehicle,"

"Subsequent Monitoring Date," "Mechanical Standards") as part of their risk assessment and contractual obligations.

- **Penalties**: Failure to comply with PTI requirements ("Non-Compliance") can lead to significant "Penalties," including "Fines" and potentially other "Legal Consequences" like vehicle immobilization.
- **The Certificate**: Upon successfully passing the inspection, a certificate (and often a windscreen sticker) is issued, detailing the vehicle, the inspection date, the expiry date ("Subsequent Monitoring Date"), and confirming it meets the required standards. If major faults are found, the vehicle fails, and a re-inspection is required after repairs.

Section	Field (French Term)	Description	Type
Vehicle Information	Registration Number	The vehicle's official license plate number.	INTEGER
	Chassis Number	Unique Vehicle Identification Number stamped on the chassis.	INTEGER
	Make (Marque)	Vehicle manufacturer (e.g., Peugeot, Renault, Toyota).	VARCHAR
	Model	Specific model of the vehicle (e.g., 208, Clio, Hilux).	VARCHAR
	Date of First Registration	When the vehicle was first registered	DATE
	Energy Type	Type of fuel used: Essence (Petrol), Gasoil etc.	VARCHAR
	Odometer Reading	Mileage at the time of the inspection.	FLOAT
Inspection Details	Inspection Center	Name, address, and official authorization number of the inspection station.	VARCHAR
	Inspector ID (Identifiant du Contrôleur)	ID of the certified inspector who performed the inspection.	VARCHAR
	Date of Inspection (Date du Contrôle)	Date the technical inspection was performed.	DATE
Inspection Results	Expiry Date (Date de Validité)	Date by which the next inspection must be done (validity period).	DATE
	Result of Inspection (Résultat du Contrôle)	Pass (Favorable) or Fail (Défavorable - re-inspection needed).	VARCHAR
	List of Checked Points (Points de	Components inspected: brakes, steering, lights,	VARCHAR

	Contrôle)	emissions, etc.	
	Observed Defects (Défauts Constatés)	Noted defects, often marked as minor, major, or critical.	VARCHAR
	Re-inspection Required (Contre-Visite Prescrite)	Indicates if the vehicle must return for re-inspection due to critical issues.	VARCHAR

Figure I.6: Table Showcasing different Information About the PIC

1.2.6. Tax Receipt

In Algeria, the **vehicle tax receipt**, often referred to as the **"vignette automobile"**, is an official document confirming the payment of the annual vehicle tax. This tax is mandatory for all vehicle owners and is used to fund public infrastructure and transportation projects[17]

1.2.6.1. Purpose of the Vehicle Tax Receipt:

- **Proof of Tax Payment:** The vignette serves as evidence that the vehicle owner has fulfilled their tax obligations for the year.
- **Legal Requirement:** Possessing a valid vignette is legally required. Failure to present it during inspections can result in penalties, including fines and potential vehicle impoundment. [17]

1.2.6.2. Usage and Verification:

- **During Inspections:** Law enforcement officers may request to see the vignette and its payment receipt to ensure compliance with tax regulations. As of 2025, motorists must present both the digital vignette and its payment receipt during such checks. [17]
- **Online Purchase and Digital Vignette:** Recent amendments facilitate the online purchase of the vehicle tax sticker. The digital vignette and its payment receipt can be downloaded, eliminating the need to affix a physical sticker to the windshield. However, both documents must be presented during inspections.[18]

1.2.6.3. Consequences of Non-Compliance:

- **Penalties:** Failure to present a valid vignette and its payment receipt during inspections can lead to the withdrawal of the vehicle's registration vehicled. A provisional authorization valid for seven days may be issued, and a tax fine amounting to 50% of the sticker's value may be imposed.

It's essential for vehicle owners in Algeria to stay informed about the latest regulations regarding the vehicle tax to ensure compliance and avoid legal issues.

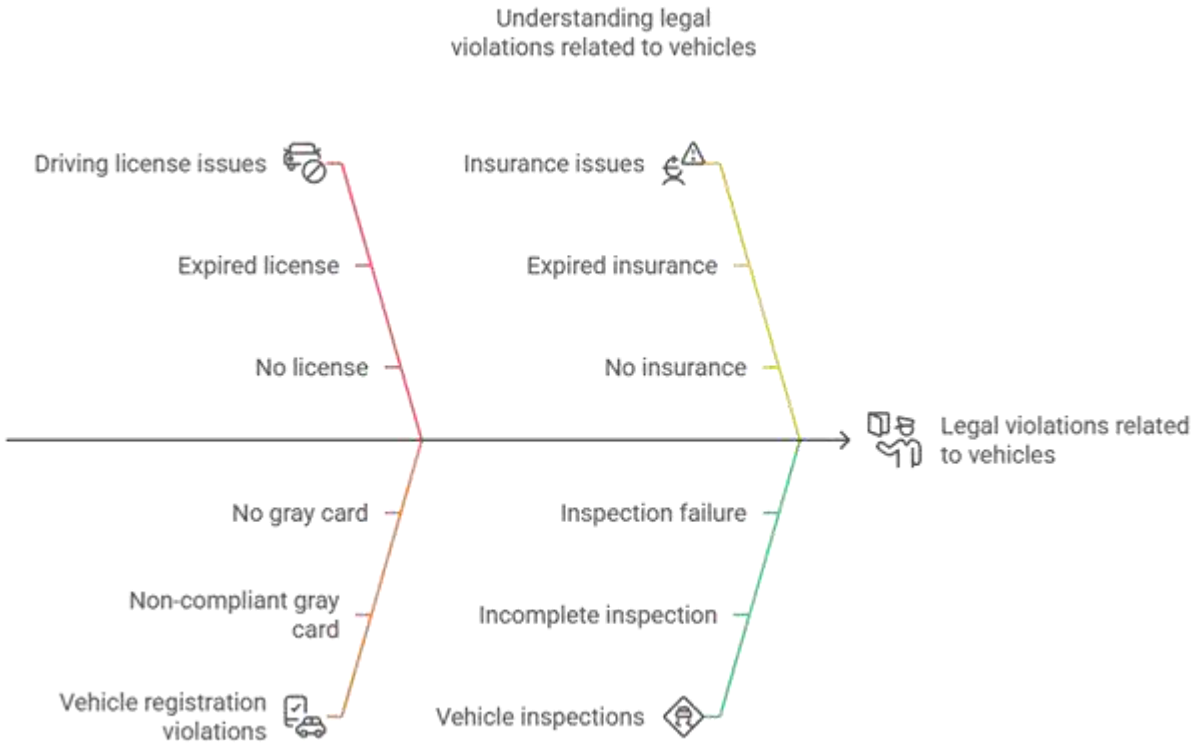


Figure V Diagram of different possible drivers violations

The police rely on these five key documents and standards to maintain road safety and ensure that all drivers comply with traffic laws. It is essential for drivers to keep their documents valid and up to date to avoid violations and penalties.

This diagram shows the common ways drivers can end up facing legal violations related to their vehicles. It breaks these violations down into four main problem areas:

1. **Driving License Problems**: You can get into legal trouble if there's an issue with your driver's license. This includes:
    - Driving with an Expired license.
    - Driving with No license at all.
2. **Vehicle Registration Problems**: Issues with the vehicle's official papers (often called the "gray vehicled" or "vehiclete grise") can lead to violations. This includes:
    - Not having the gray vehicled for the vehicle.
    - Having a Non-compliant gray vehicled, meaning the document is incorrect, out-of-date, or doesn't match the vehicle properly.

3. **Insurance Problems**: Lack of proper insurance is a major cause of legal violations. This includes:
    - Driving with Expired Insurance.
    - Driving with No Insurance coverage for the vehicle.
4. **Vehicle Inspection Problems**: Failure to meet mandatory technical inspection requirements is another source of violations. This includes:
    - The vehicle having an Inspection failure (meaning it didn't pass the required checks).
    - Having an Incomplete inspection, which could mean the inspection wasn't done, isn't valid anymore, or the certificate is missing.

In simple terms: The diagram explains that to avoid legal trouble with your vehicle, you (the driver) need a valid license, the vehicle needs correct registration papers and valid insurance, and it must pass its required technical safety checks. Any failure in these areas can lead to legal violations.

1.3. Challenges in Police Operations

Law enforcement agencies often encounter various difficulties when managing vehicle-related data, including:

- Difficulty in searching for vehicle records manually.
- Time-consuming processes leading to delays in operations.
- Risk of document loss or damage in paper-based systems.
- Limited storage capacity for physical records.
- Inaccurate or outdated record-keeping.
- Challenges in statistical analysis and reporting.
- Difficulty in accessing real-time vehicle information.
- Lack of an efficient system for verifying license plates and retrieving related data.

Vehicle inspections are still largely conducted manually, resulting in reduced efficiency.

1.4. Role of our System for Law Enforcement

Our digital system is designed to modernize and streamline police operations through intelligent automation, secure data handling, and seamless integration. Its primary goal is to assist law enforcement officers in making faster, more informed decisions while maintaining a high standard of data integrity and security. Below are the key functionalities and benefits:

- **Automatic License Plate Recognition (ALPR):**
  Enables real-time scanning and recognition of vehicle license plates using the device camera and OCR technology, minimizing manual input and speeding up identification.

- **Quick Vehicle Information Retrieval:**
  Instantly fetches detailed vehicle data including registration status, insurance validity, reported thefts, and outstanding tickets—helping officers make informed decisions on the spot.
- **Secure and Encrypted Data Storage:**
  All collected records (e.g., scans, officer notes, timestamps) are encrypted to ensure data confidentiality and integrity, reducing risks of unauthorized access.
- **Integration with External Databases:**
  Seamlessly connects to third-party databases such as insurance providers, stolen vehicle registries, and ticketing systems to provide complete, up-to-date information in one place.
- **Real-time Synchronization:**
  Automatically syncs data across devices and departments, ensuring that all users operate with the latest, consistent information.
- **Enhanced Interagency Communication:**
  Facilitates smoother collaboration between police departments and external entities, such as government agencies, insurance companies, and traffic control authorities.
- **Digital Evidence Capture:**
  Allows officers to attach images, notes, and location data to each vehicle interaction, improving accountability, record-keeping, and legal preparedness.
- **Operational Efficiency:**
  Streamlines workflows by reducing paperwork and manual lookups, giving officers more time to focus on patrol duties and public safety.

1.5. Conclusion

In this first chapter, we provided an overview of the challenges associated with manual vehicle inspection. We then focused on identifying key issues that hinder efficiency and accuracy.

In the next chapter, we will analyze the requirements and explore how our proposed solution can address these challenges.

SECTION II
SYSTEM DESIGN

1. Introduction

In this chapter, we present the design of our system. The modeling is done using UML (Unified Modeling Language).

The main diagrams used are:

- Use Case Diagrams
- Sequence Diagrams
- Class Diagrams

1.1 UML Definition:

UML (Unified Modeling Language) is a standard object-oriented modeling language used to specify, visualize, construct, and document the components and structure of an information system.

1.1.1 2TUP Definition

We used the 2TUP (Two-Tiered Unified Process) modeling standard in our project. It is a software development methodology that incorporates UML and provides structured guidelines for modeling, documenting, and managing different phases of a software system's design.

2. Design

In this section, we model the most important features of the **mobile application**, which is designed to help police officers track vehicle status using license plate recognition.

We use UML diagrams to:

- Illustrate user interactions with the system
- Describe internal system behavior
- Visualize data flow and database relations

2.1.1 Identification of Actors

Actor	Role
Police Officer	- Register and log in to the app - Check vehicle status - File a ticket - Report a stolen vehicle
Traffic management	- Manage police database - Search vehicle data and update violations
Vehicle Owner	- Checking Vehicle Status(vehicle documents) - Report a stolen vehicle

2.1.2 Use Case Diagrams:

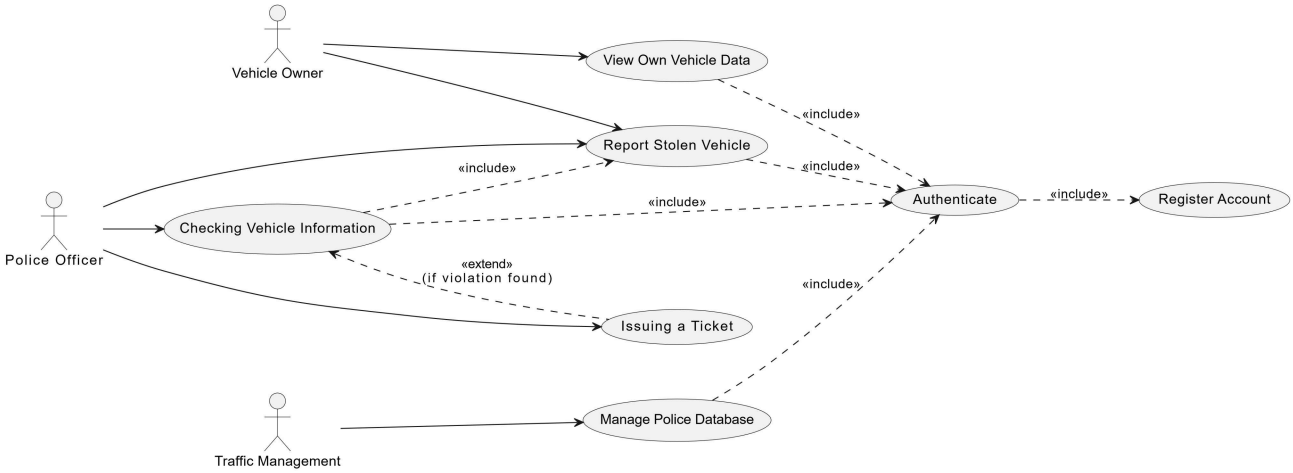


Figure VI Use Case Diagram "System"

a) Use Case: Authenticate (Common for All Actors)

Objective: Verify identity of users before granting access to system features

Actors: Police Officer, Vehicle Owner

Summary:

All users (police officers and vehicle owners) must log in with an email and password. This use case is a prerequisite for all other operations and is always assumed to have already occurred before performing other tasks in sequence diagrams.

Preconditions:

- User has a registered account
- User has access to the app

Normal Flow:

1. User enters email and password into the app.
2. The app sends credentials to the API.
3. The API checks credentials against the database.
4. If valid, user is logged in and granted access to the app.
5. If invalid, an error message is returned.

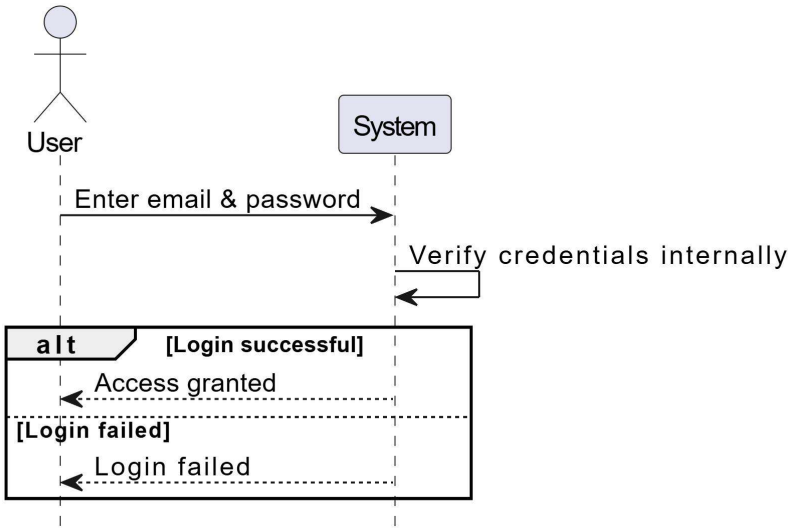


Figure VIII User Login Sequence Diagram

b) . Use Case: Register Police Officer

Actors Involved: Police Officer
Objective: Allow an authorized officer to register a new police account.
Preconditions:

- Only authorized police officers can register new accounts.
- The officer is authenticated.

Normal Flow:

- The officer opens the registration screen.
- The officer enters:
  - Full Name
  - Badge Number
  - Department
  - Email
  - Password
  - Rank

1. The app sends this data to the API.
2. The API checks the database to ensure no duplicate badge or email exists.
3. If all fields are valid, the new account is saved to the database.
4. The system confirms successful registration.

Postconditions:

A new police officer account is created and can now authenticate.

b.a) Sequence Diagram

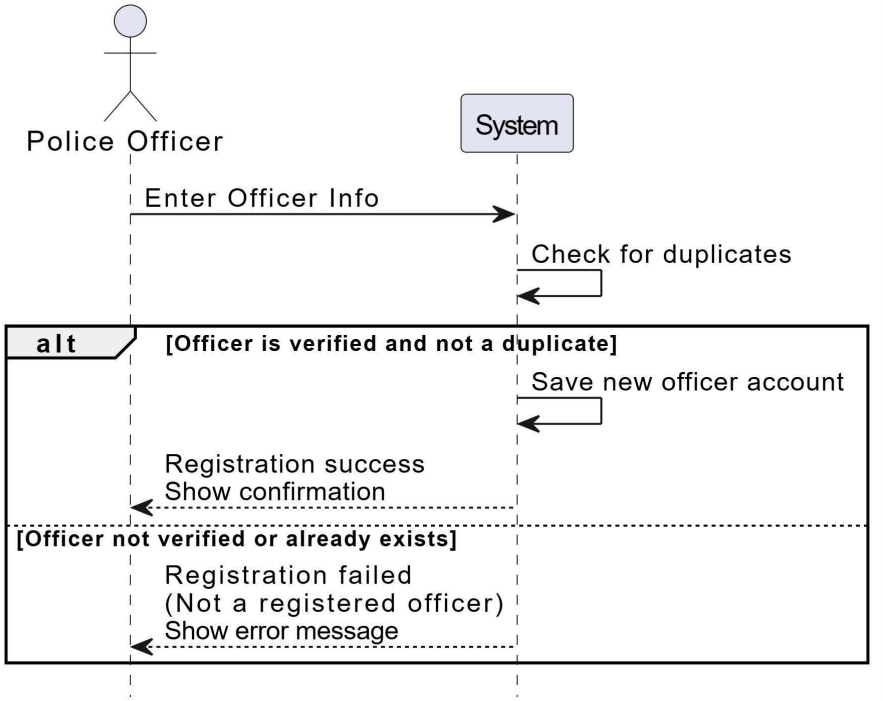


Figure IX Officer Registration Sequence Diagram

B.a) Class Diagram

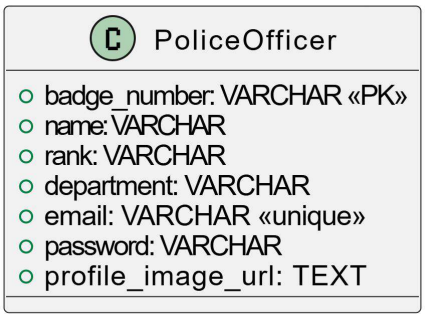


Figure X Officer Table

c) Use Case: Register Account for Vehicle Owners

Title: Register Vehicle Owner
Actors Involved: Vehicle Owner
Objective: Allow a Vehicle owner to create an account to access and manage vehicle data.
Preconditions:

- The Vehicle owner is not yet registered.

Normal Flow:

- The user opens the registration screen.
- The user enters:
  - Name
  - Email
  - Password
  - Vehicle License Plate

1) The app sends this data to the API.
2) The API checks if the email or plate already exists in the database.
3) If valid, a new account is created and saved in the database.
4) The user is informed of successful registration.

Postconditions:

- A new Vehicle owner account is created and can now authenticate.

c.a) Sequence Diagram

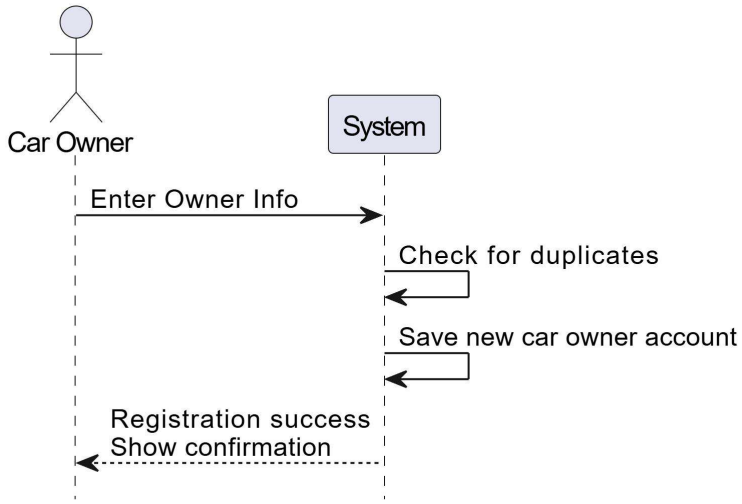


Figure XI Vehicle Owner Registration Sequence Diagram

c.b) Class Diagram

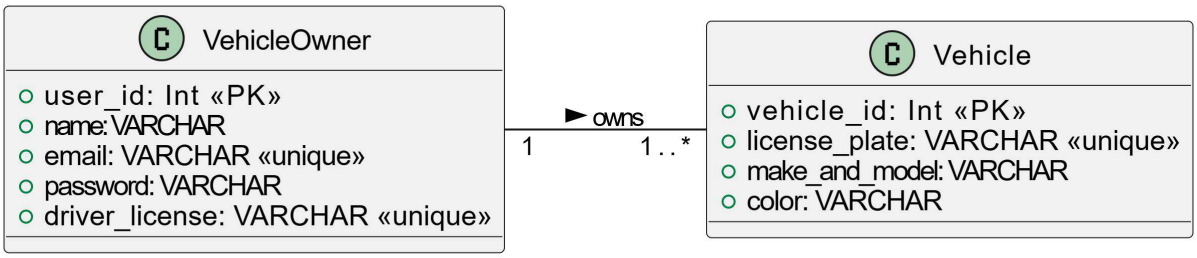


Figure XII Tables Related To Vehicle Owner Registration

d) Use Case: Check Vehicle Information

Actor: Police Officer

Objective: To verify details of a specific vehicle by searching using its license plate.

Preconditions:

- Police officer is authenticated.
- Officer has access to the app interface.

Scenario Description (Normal Flow):

1. Officer inputs or scans a license plate using the app.
2. The app sends a request to the API to retrieve vehicle data.
3. The API queries the database.
4. The database returns vehicle details (owner, insurance, violations, etc.).
5. The API responds with the formatted vehicle data.
6. The app displays the retrieved vehicle information to the officer.

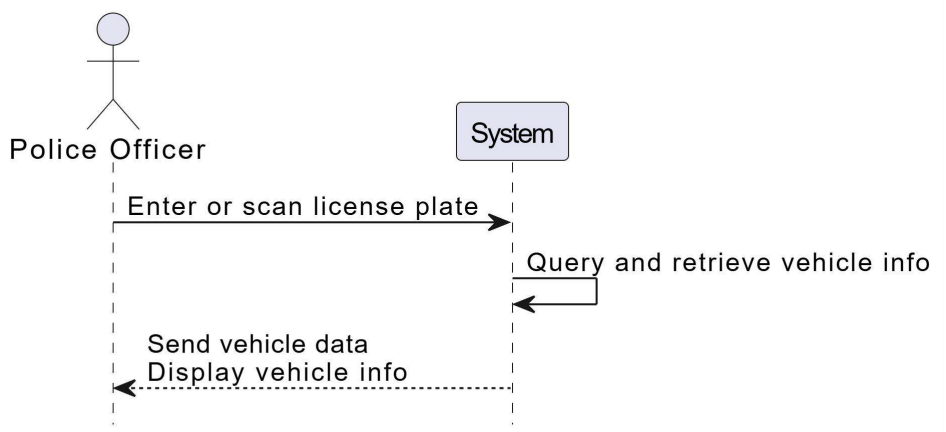


Figure XIII Checking Vehicle Info For Police Officers Sequence Diagram

e) Use Case: Check Vehicle Information(Vehicle Owner)

Actor: vehicle Owner

Objective: Retrieve and display the vehicle's details when opening the app.

Preconditions:

- The user is authenticated and logged in.
- The user has already registered a vehicle in the system.

Scenario Description (Normal Flow):

1. The vehicle owner opens the app (home or profile screen).
2. The app automatically triggers a request to fetch vehicle data linked to the logged-in user.
3. The app sends a request to the API with the user's identifier.
4. The API queries the database for the user's registered vehicle(s).
5. The database returns the relevant vehicle information.
6. The API sends the data back to the app.
7. The app displays the vehicle details to the user.

e.a) Sequence Diagram

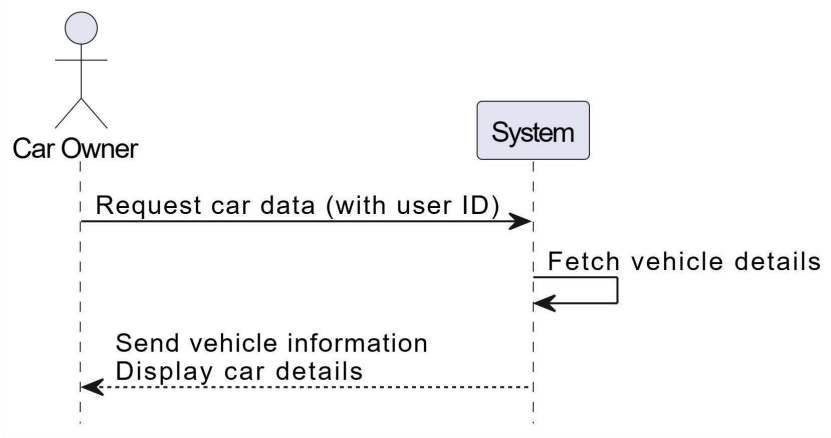


Figure XIV Checking Vehicle Info For Vehicle Owners

e.b) Class Diagram

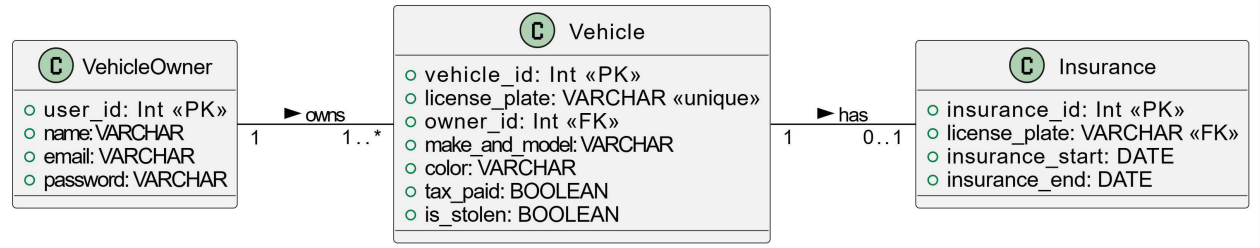


Figure XV Class Diagram for Vehicles

f) Use Case: Report Stolen Vehicle

Actor: Vehicle Owner

Objective: To notify the authorities that their vehicle has been stolen.

Preconditions:

- The vehicle owner must be authenticated.
- The Vehicle must already exist in the system (previously added by the user or verified by the system).

Scenario Description (Normal Flow):

1. The Vehicle owner opens the app and navigates to the "Report Stolen" section.
2. They select one of their registered vehicles.
3. The app prompts the user to confirm the report and optionally provide extra details (e.g., last seen location, time).
4. The report is submitted to the system through the API.
5. The backend verifies the vehicle belongs to the reporting user.
6. The report is saved to the database with a "stolen" flag and timestamp.
7. A confirmation message is sent back to the app.
8. The app displays a success message and may disable future edits to that Vehicle until resolved.

f.a) Sequence Diagram

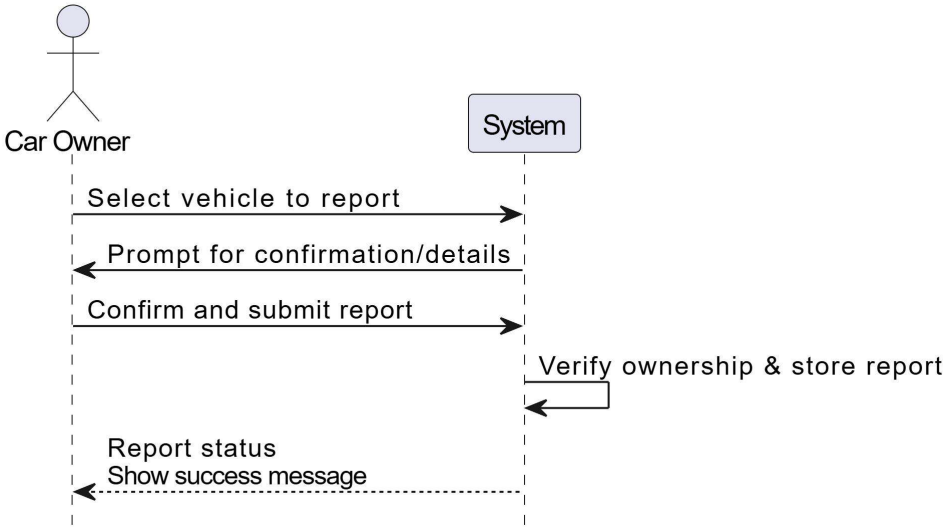


Figure XVI Reporting Stolen vehicle Sequence Diagram

f.b) Class Diagram

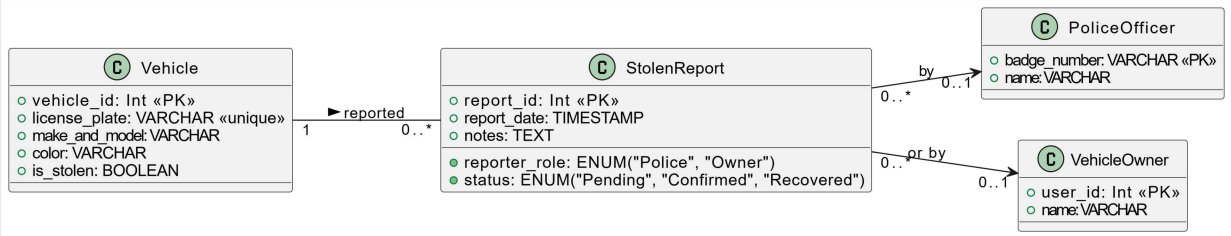


Figure XVII Class Diagram Related to Reporting Stolen vehicle

g) Use Case: Issue Ticket

Actor: Police Officer

Objective: To issue a traffic violation ticket after identifying a vehicle violation.

Preconditions:

- Officer has already checked the vehicle info.
- A violation has been detected or observed.

Scenario Description (Normal Flow):

1. Officer taps to issue a ticket within the app.
2. Officer fills out ticket details (violation type, location, comments, etc.).
3. The app sends the ticket information to the API.
4. The API validates and stores the ticket data.
5. The database confirms the successful insertion.
6. The app receives a success response and displays a confirmation.

a) Sequence diagram

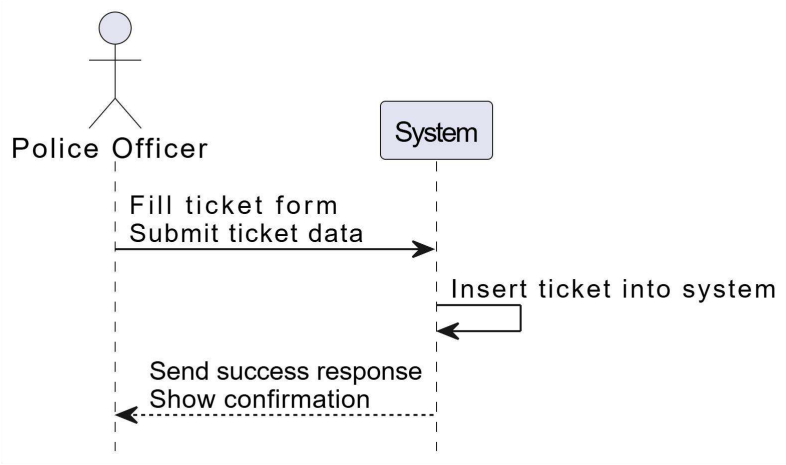


Figure XVIII Filing a ticket Sequence Diagram

b) Class Diagram

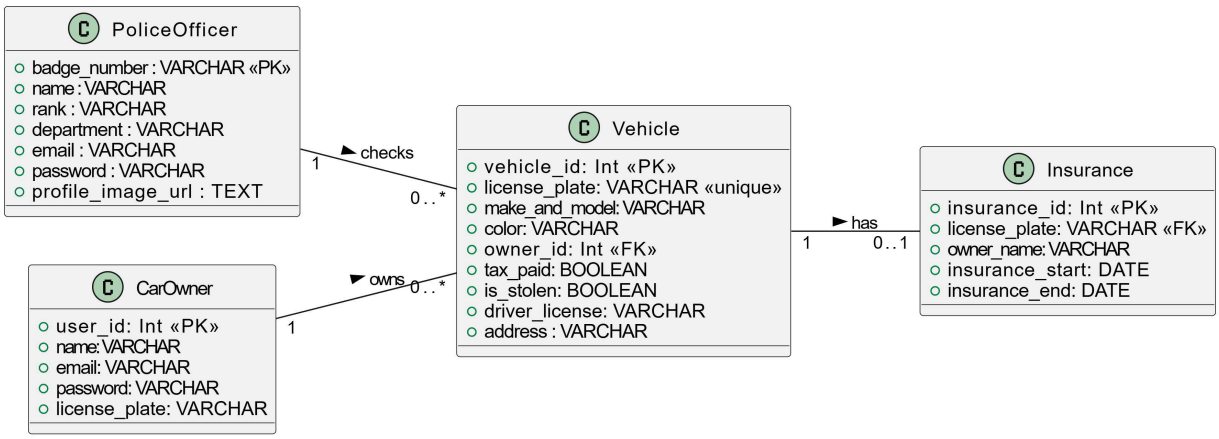


Figure XIX Class Diagram for Filing a Ticket

h) Use Case: Manage Police Database

Actor: Traffic Management

Objective: Add, edit, or delete police officer accounts in the system.

Preconditions:

- Traffic management staff is authenticated.

- The staff has proper administrative access.

Scenario Description (Normal Flow):

1. Traffic Management opens the database management interface (e.g., phpMyAdmin, admin panel, or SQL client).
2. They locate the `officers` table.
3. To **Add**, they insert a new officer record with all required fields: name, badge number, department, rank, and email.
4. To **Edit**, they select an existing record and update the fields.
5. To **Delete**, they remove a selected officer row.
6. The changes are applied directly to the database.
7. Success or error messages are shown within the DB interface.

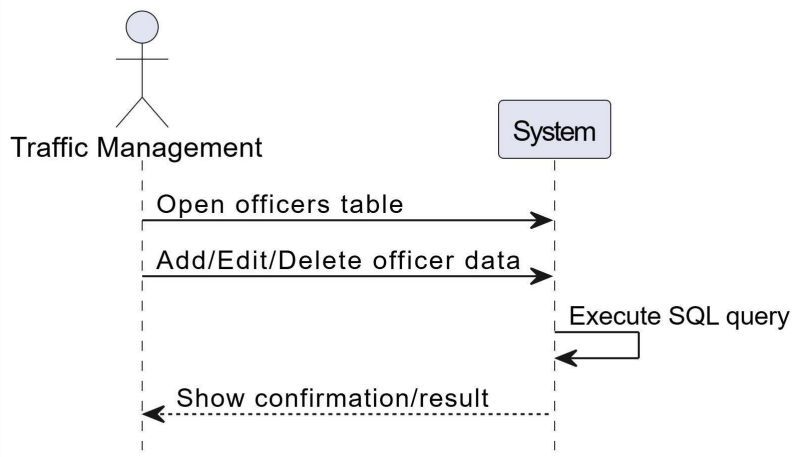


Figure XX Editing Officers Account Sequence Diagram

k) Search and Consult Vehicle Data

Actor: Traffic Management

Objective: Retrieve specific or general vehicle data from the database for verification or analytics.

Preconditions:

- Traffic Management has authenticated DB access.
- The database is available and properly indexed.

Scenario Description (Normal Flow):

1) Traffic Management opens the database tool.
2) They navigate to the `vehicles` table.
3) They execute a query to:

- Find a specific vehicle by license plate, or
- Browse/filter by owner, model, insurance status, etc.

4) The system returns the matching records.
5) The Traffic Manager reviews the vehicle information directly.

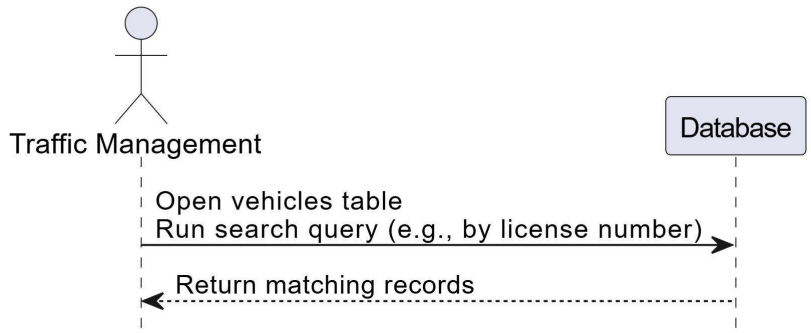


Figure XXI Searching and Consulting vehicle Data Sequence Diagram

l) Search and Update Violations

Actor: Traffic Management

Objective: Search and update traffic violation records, such as removing resolved tickets after payment.

Preconditions:

- Traffic Management is logged into the database system.
- The `violations` table exists and contains indexed data.

Scenario Description (Normal Flow):

1. Traffic Management logs into the database management tool.
2. They access the `violations` table.
3. They execute a **search query** using vehicle license plate or ticket number.
4. The system returns related violation entries.
5. Traffic Management identifies violations that were resolved (e.g., fines paid).
6. They either **mark them as resolved** or **delete them** depending on policy.
7. Database is updated accordingly.

l.a) Sequence Diagram

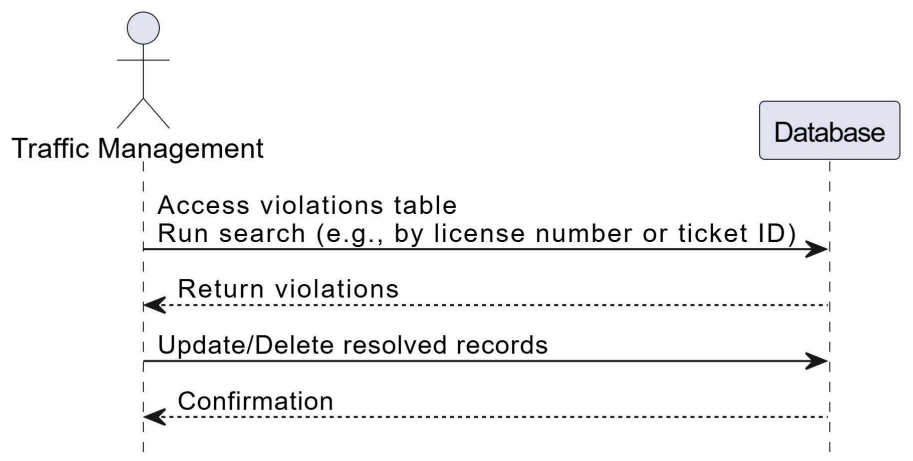


Figure XXII Checking And Updating Violations Sequence Diagram

l.b) Class Diagram

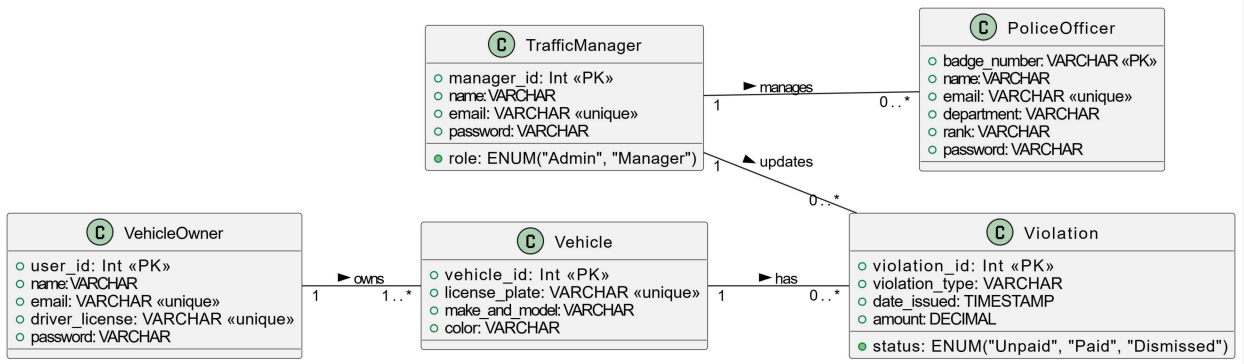


Figure XXIII Traffic Management Class Diagram

2.2 Technical Requirements Identification

The gathering of technical requirements, which complements the collection of functional requirements, encompasses all constraints not directly related to describing user actions or application features. The specification model is defined from two perspectives: software specifications (technical system functions) and the system architecture, including the physical infrastructure utilized. This section details these technical requirements for the "Smart Vehicle Monitoring" application, reflecting the chosen technology stack: Kotlin with Jetpack Compose for the mobile frontend, Node.js with Express for the backend API, MySQL as the primary database, Room for local mobile persistence, Google ML Kit for OCR, and various supporting libraries and cloud services (Render, Clever Cloud).

2.2.1 New System Architecture Diagram

The "Smart Vehicle Monitoring" system adopts a multi-tier architecture. This architecture is optimized for mobile application performance, scalability, and maintainability. It integrates local device capabilities, cloud-based services, and interactions with external data systems for specialized information like insurance and vehicle inspection status.

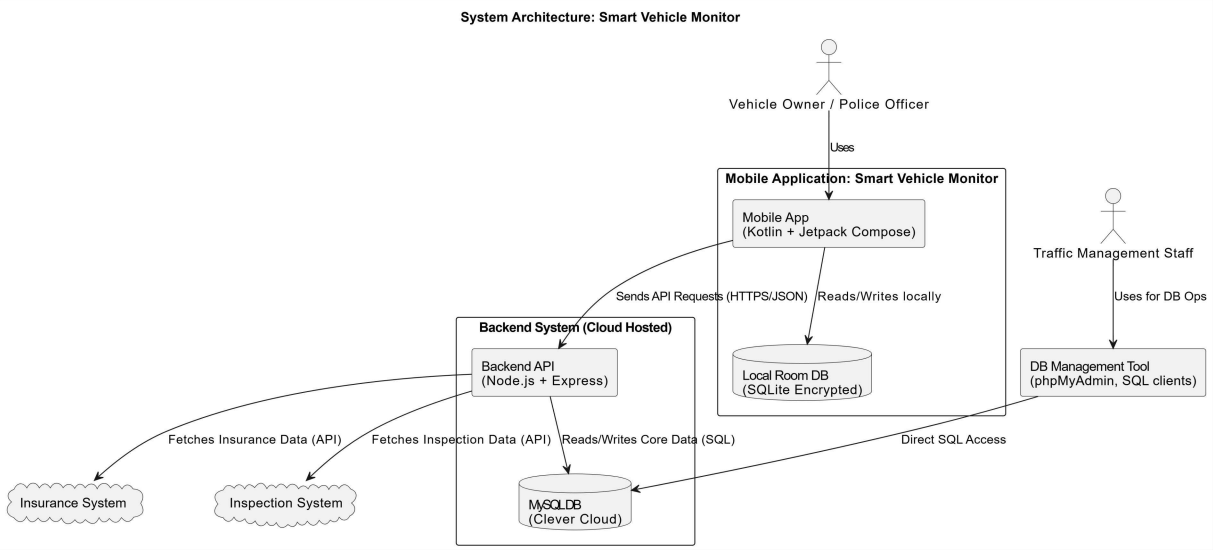


Figure XXIV New System Architecture Diagram

2.2.2 Description of the New System Architecture (Multi-tier with external integrations)

The system architecture is distributed as follows:

- **Mobile Client (User Device):**
  - **Mobile Application:** Developed using **Kotlin** with **Jetpack Compose** for a modern, native UI. Network communication with the Backend API is handled efficiently and type-safely using **Retrofit**. Dependency injection is managed by **Dagger Hilt** to promote a clean architecture. Local data persistence for offline capabilities (like scan history) is managed by the **Room** persistence library over an encrypted SQLite database. **Google ML Kit** is integrated for on-device OCR functionality.

- **Backend System (Cloud Hosted):**
  - **Backend API (Application Server):** Implemented with **Node.js** and the **Express** framework, providing a robust and scalable backend. It is hosted on **Render**. This API is responsible for all business logic, processing mobile app requests, user authentication (as detailed in your sequence diagrams), and orchestrating data flow.
  - **Primary Database Server (MySQL):** Hosted on **Clever Cloud**, this relational database stores core application data: user accounts (`PoliceOfficer`, `VehicleOwner`), vehicle information, traffic `Tickets`, and `StolenReport` records.

- **External Data Systems:**
  - **External Insurance Data System:** An external service providing authoritative insurance information, accessed by the Backend API.
  - **External Vehicle Inspection System:** An external service providing authoritative vehicle inspection records, accessed by the Backend API.

- **Traffic Management Interface:**
  - **Database Management Tool:** Traffic Management staff utilize standard database tools (e.g., phpMyAdmin, DBeaver, or other SQL clients) for direct interaction with the **MySQL** database for administrative tasks like managing officer accounts and updating violation statuses, as per their defined use cases.
- **Communication:**
  - Mobile App to Backend API: HTTPS/JSON, facilitated by Retrofit.
  - Backend API to Internal MySQL: SQL queries via a Node.js MySQL driver.
  - Backend API to External Systems: Secure HTTPS API calls.
  - Authentication for mobile app users is token-based (JWTs), managed by the Backend API.

This architecture ensures a separation of concerns, leverages specialized external services for data accuracy, and provides a robust platform for the application's functionalities.

2.2.3 Identification of Technical Use Cases

To establish the software specification model, a focus is placed on the system's specific technical functions, distinct from the user-facing functional use cases.

2.2.3.1 Beneficiaries of the System (from a technical perspective):

- **End User (Mobile App User: Police Officer, Vehicle Owner):** The primary interactor with the system via the Kotlin-based mobile application. Their actions trigger various technical processes like data persistence through Room and API calls via Retrofit.
- **Traffic Management Staff:** Interacts with the system at a data level, directly accessing the MySQL database for administrative functions.
- **System Administrator/Developer (Operations Engineer in the example):** Responsible for deploying the Node.js backend on Render, managing the MySQL database on Clever Cloud, monitoring system health, managing API keys for external services, and troubleshooting.

2.2.3.2 Operational Expectations for each Beneficiary Category:

End Users (Mobile App):

- Will interact with data entities (Vehicles, Tickets, Stolen Reports) as objects within the mobile app. This necessitates robust **Data Persistence** mechanisms (Room for local caching, Node.js API for backend MySQL storage, and integration with external Insurance/Inspection APIs).

- Expect responsive UI interactions (Jetpack Compose) and efficient data fetching (Retrofit, optimized Node.js API).
- Police Officers require **Offline Access** to critical data like scan history, managed by Room.

Traffic Management Staff:

- Requires reliable and secure direct access to the **MySQL** database to perform administrative tasks on officer accounts and violation records.
- **Multiple Users (Mobile and Traffic Management):**
- The system must handle concurrent operations. The Node.js backend, with its event-driven, non-blocking I/O, is well-suited for concurrent API requests. The MySQL database must manage concurrent connections from the API and direct access tools. **Data Integrity** is paramount, preventing issues from simultaneous updates.

The System (as a whole):

- Must securely **Authenticate** mobile users via the Node.js API and **Authorize** their actions based on roles. Traffic Management staff are authenticated at the database level.
- Must provide mechanisms for **Error Handling & Logging** across the mobile app (Kotlin), backend (Node.js), and interactions with external services.
- The system must be **Maintainable and Deployable**, facilitated by the chosen modular technologies (Kotlin/Jetpack Compose, Node.js/Express, cloud hosting on Render/Clever Cloud).

2.2.4 Security Rules Governing System Operations:

- **Authentication:** Mobile users via email/password against Node.js API, leading to JWT issuance. Traffic Management via MySQL database credentials.
- **Authorization (Habilitation):** Role-based access control within the Node.js API for mobile users. Database-level permissions for Traffic Management staff.
- **Encryption (Cryptage):** HTTPS for all API communication (Retrofit to Node.js, Node.js to external APIs). Encryption for sensitive data in the local Room database. Secure configuration of MySQL on Clever Cloud.
- **Data Integrity:** Ensured via database constraints, input validation in the Node.js API, and careful handling of data from external services.
- **Audit :** Comprehensive logging by the Node.js backend on Render for API requests, critical operations, and errors. MySQL logs on Clever Cloud.
- **Non-Repudiation:** Achieved through secure authentication and audit trails linking actions to specific users.

- **Input Validation:** Both client-side (basic in Kotlin app) and server-side (thorough in Node.js/Express API) to prevent common vulnerabilities.
- **Secure API Key Management:**** For accessing External Insurance and Inspection Systems.

The technical constraints and operational expectations lead to a software specification model represented by the following technical use case diagram, adapted from the provided example:

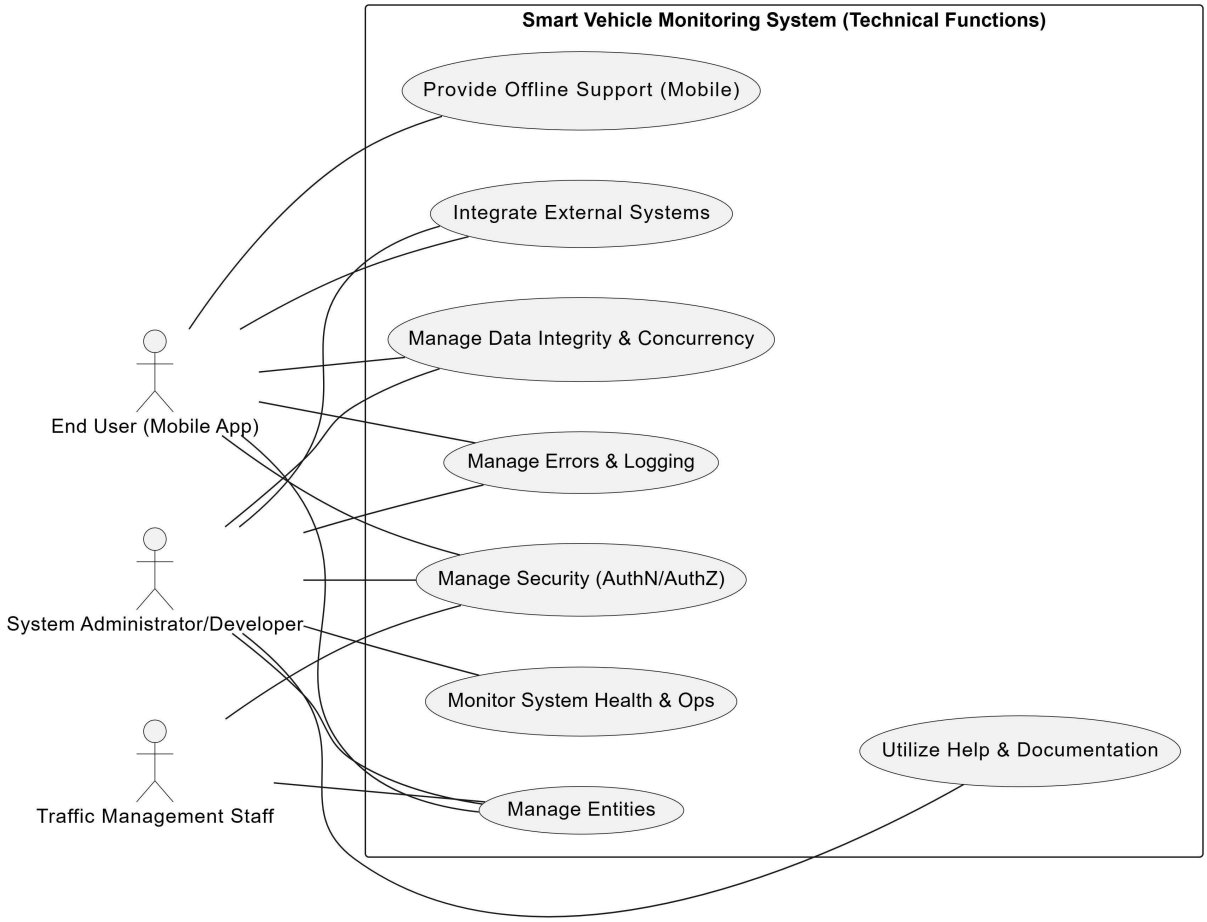


Figure XXV Technical Use Case Diagram for the System

2.2.5 Description of Technical Use Cases (with Activity Diagrams)

A. Use Case: "Manage Entities & Persistence"

Goal: The system must allow End Users (via the mobile app and Node.js API) and Traffic Management Staff (via direct DB access) to interact with data entities (Users, Vehicles, Tickets, StolenReports, Insurance/Inspection data summaries). This involves creating, reading,

updating, and deleting these entities, ensuring data persistence in Room (local cache), MySQL (core data), and proper interaction with external Insurance/Inspection APIs for relevant data.

Procedures (Technical Flow - Mobile User retrieving full vehicle data):

1. Kotlin app (using Retrofit) sends an API request to the Node.js/Express backend.
2. Node.js API authenticates the request (JWT).
3. Node.js API queries internal MySQL for core vehicle data.
4. Node.js API makes separate, secure API calls to External Insurance and Inspection Systems.
5. Node.js API aggregates all retrieved data.
6. Node.js API sends a consolidated JSON response to the Kotlin app.
7. Kotlin app updates its local Room database cache and displays info using Jetpack Compose.

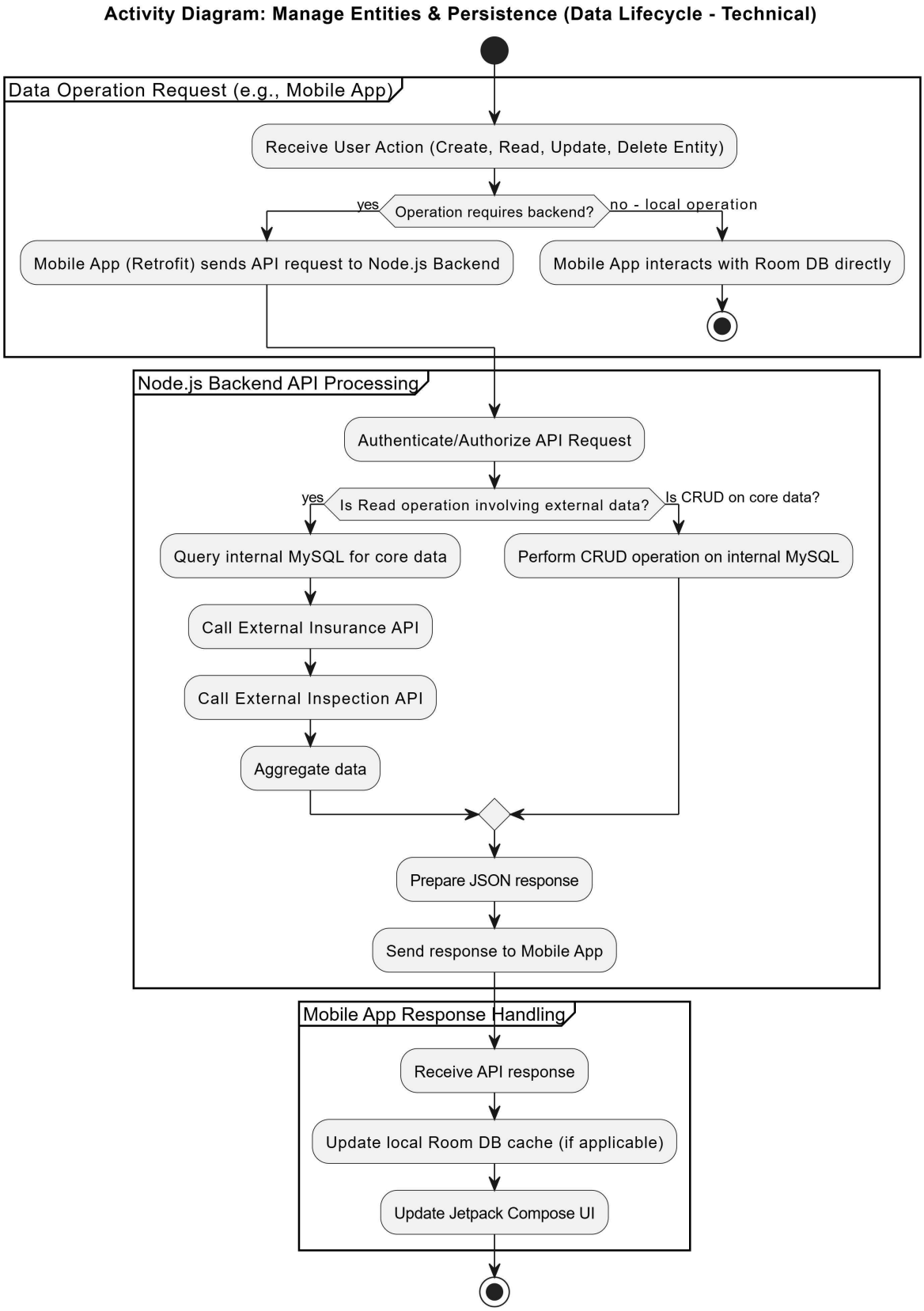


Figure XXVI Activity Diagram for "Manage Entities & Persistence

B. Use Case: "Manage Data Integrity & Concurrency"

Goal: The system must allow multiple users (mobile app users via Node.js API, Traffic Management via direct DB access) to interact with data concurrently without compromising data integrity. This prevents issues like conflicting updates.

Procedures (Technical Flow):

1. Node.js API handles concurrent requests from mobile clients.
2. MySQL database uses transaction management and appropriate isolation levels to ensure ACID properties for operations initiated by the API or direct DB tools.
3. For operations modifying data, optimistic or pessimistic locking might be employed at the Node.js or MySQL level if high contention is anticipated for specific resources.
4. Validation rules in the Node.js API ensure data consistency before database writes.

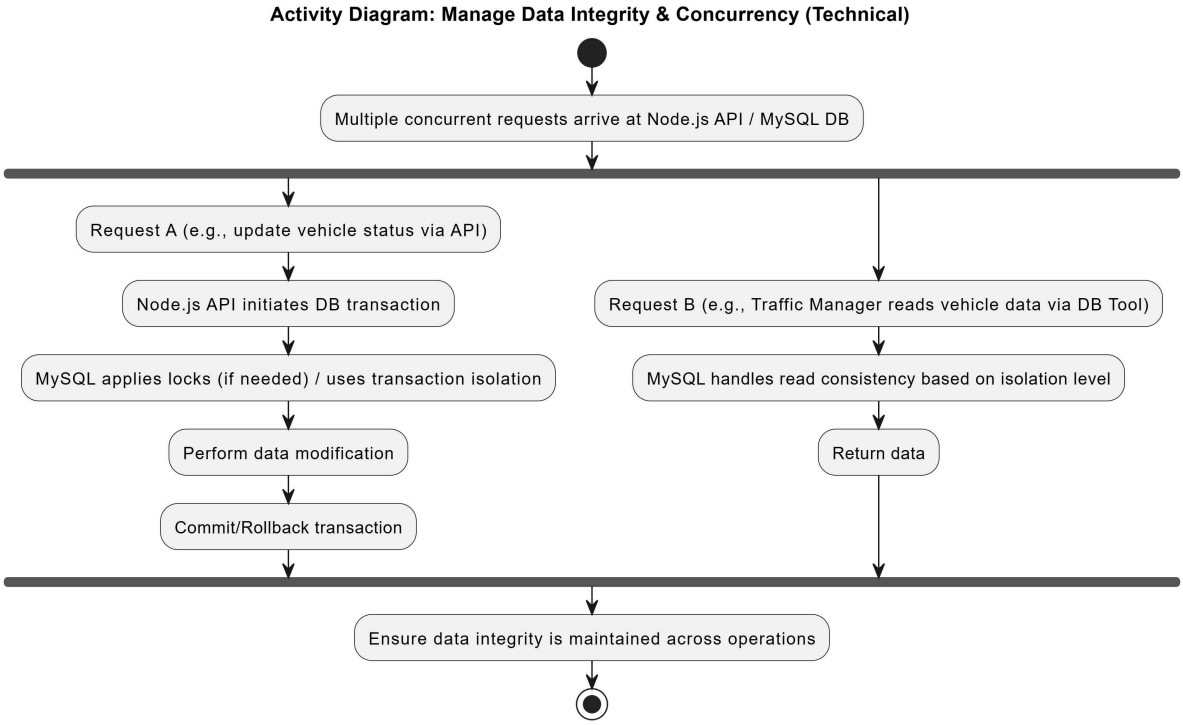


Figure XXVII Activity Diagram for Manage Data Integrity & Concurrency

C. Use Case: "Manage Errors & Logging"

Goal: The system must be able to manage errors generated during its operation, providing clear messages to users and comprehensive logs for administrators.

Procedures (Technical Flow):

1. Kotlin mobile app uses try-catch blocks for Retrofit calls and Room operations, displaying user-friendly errors via Jetpack Compose.
2. Node.js/Express backend uses middleware and try-catch blocks to handle errors from its logic, database interactions, or calls to external APIs.
3. Standardized error responses (HTTP status codes, JSON error objects) are sent to the mobile app.
4. Detailed error logs (stack traces, request context, timestamps) are generated by Node.js and stored via Render's logging services or configured log files. MySQL errors are logged by Clever Cloud.

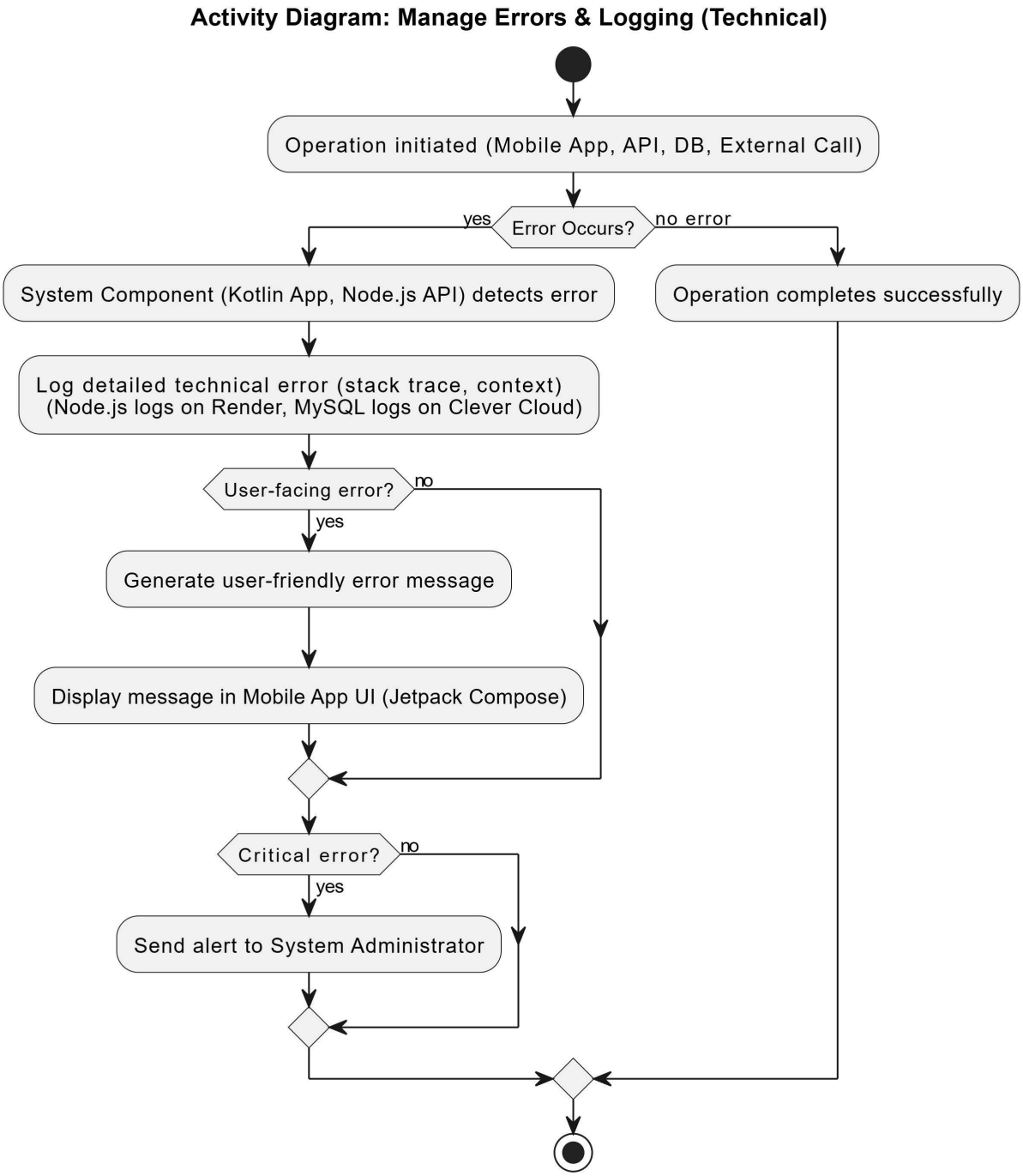


Figure XXVIII Activity Diagram for Manage Errors & Logging

D. Use Case: "Manage Security (Authentication & Authorization)"

Goal: To provide the system with security mechanisms to protect it from intrusions and malicious access, verifying the identity of each user (mobile or Traffic Management) and their assigned roles and permissions.

Procedures (Technical Flow - Mobile User Authentication, as per your sequence diagram):

1. User enters credentials in the Kotlin/Jetpack Compose app.
2. Mobile app (using Retrofit) sends credentials securely (HTTPS) to the Node.js/Express API.
3. Node.js API validates credentials against the `Users` table in MySQL (passwords should be hashed).
4. If valid, API generates a JWT and returns it to the mobile app.
5. Mobile app stores the JWT securely (e.g., Android Keystore or encrypted SharedPreferences) and includes it in headers of subsequent Retrofit calls.
6. Node.js API middleware validates the JWT for authorization on protected routes.

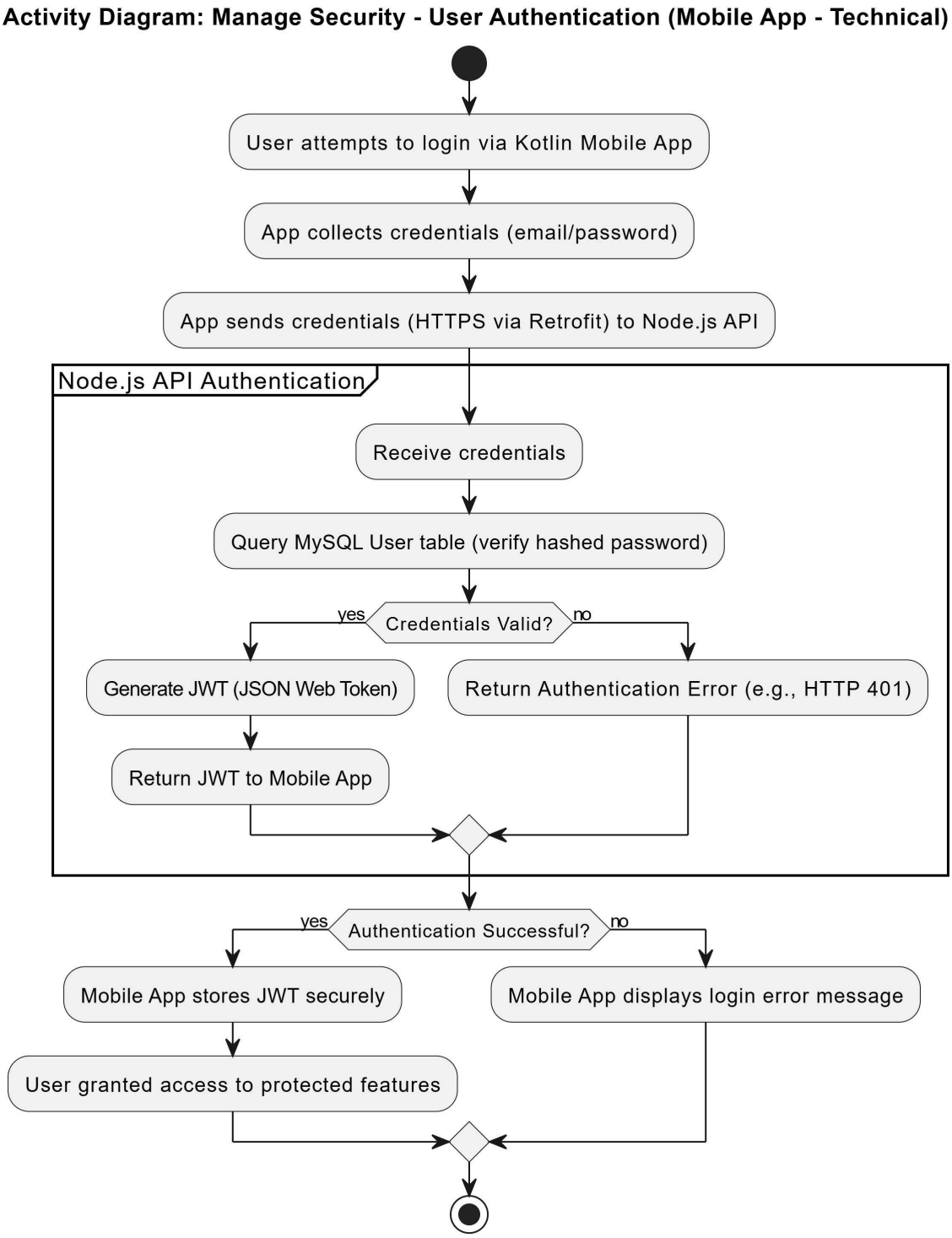


Figure XXIX Activity Diagram for Manage Security (Authentication)

E. Use Case: "Utilize Help & Documentation"

Goal: To facilitate the use of the system for different users, particularly System Administrators/Developers, by providing contextual help and comprehensive documentation.

Procedures:

1. **For End Users (Mobile App):** Intuitive UI design (Jetpack Compose), clear error messages, and potentially in-app guides for complex features.
2. **For System Administrators/Developers:** Well-documented Node.js API (e.g., using Swagger/OpenAPI via Postman or integrated tools). Clear documentation for deployment on Render/Clever Cloud, database schema, and troubleshooting guides. Code comments in Kotlin and Node.js. Version control history via Git/GitHub.

2.2.6 Relationships

Police Database:

- **PoliceOfficer**: Contains information about police officers, including their badge number, name, email, password, rank, and department. Officers can issue tickets and report stolen vehicles.
- **vehicleOwner**: Represents registered vehicle owners. Stores name, email, password, and a unique driver's license number.
- **TrafficManager**: Users with administrative roles ("Admin" or "Manager") responsible for managing police officers.

Core Domain:

- **Vehicle**: Represents vehicles in the system. Each vehicle has a unique license plate, make and model, color, owner (linked to a vehicleOwner), and a flag indicating whether it is stolen.
- **Ticket**: Represents a traffic ticket issued for a vehicle. Includes the issuing officer, violation type, date, status, fine amount, and optional details.
- **StolenReport**: Tracks reports of stolen vehicles, including who reported it (Police or Owner), the report status, and additional notes.
- **Insurance**: Stores vehicle insurance information, including policy duration, provider name, and policy number. Linked to both the vehicle and its owner.

Relationships:

- **Each vehicle** is owned by **one vehicleOwner** (One-to-Many: vehicleOwner → Vehicle).
- **Each vehicle** can have **multiple tickets** issued against it (One-to-Many: Vehicle → Ticket).
- **Each ticket** is issued by **one PoliceOfficer**, identified by their badge number (Many-to-One: Ticket → PoliceOfficer).
- **Each PoliceOfficer** can be managed by a **TrafficManager** (One-to-Many: TrafficManager → PoliceOfficer).
- **Each vehicle** can have **at most one insurance record** (One-to-One: Vehicle → Insurance).
- **Each stolen report** is associated with a **single vehicle** (One-to-Many: Vehicle → StolenReport).

- **A stolen report** is created either by a **PoliceOfficer** or a **vehicleOwner**, depending on the reporter's role (Many-to-One: StolenReport → [PoliceOfficer | vehicleOwner]).
- **Each insurance record** is linked to a specific **vehicle and owner**, enforcing ownership consistency.

2.2.7 Class Diagram :

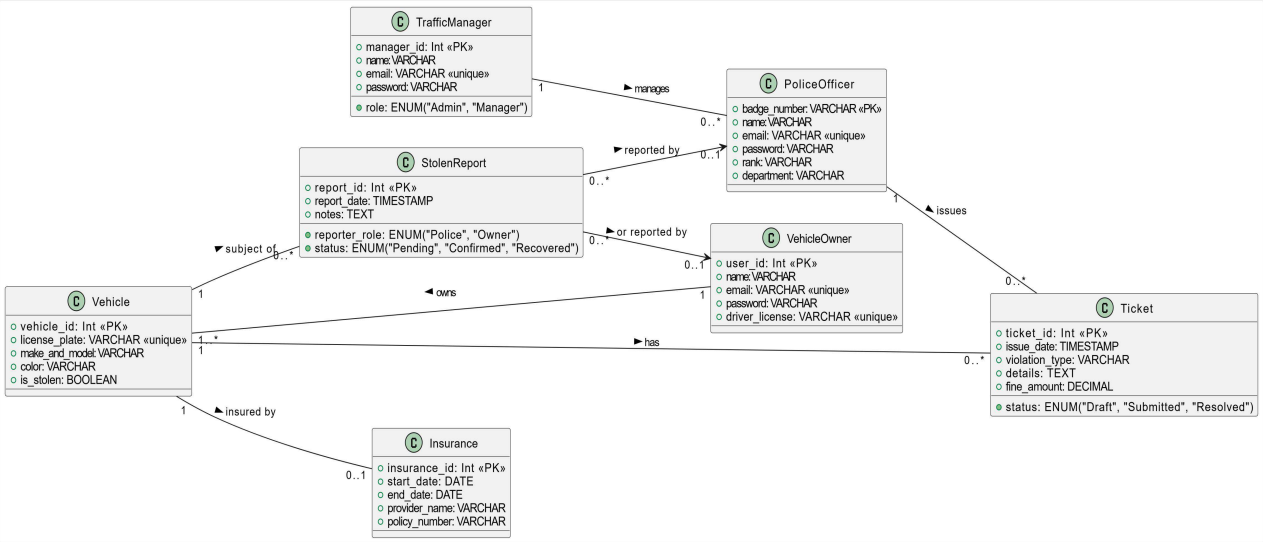


Figure XXX Entire Class Diagram for Our System

2.2.7 Moving from class schema to databases :

users(id, email, password, name, rank, department, badge_number, vehicles_scanned, officer_image, created_at)

tickets(ticket_id, driver_license, ticket_type, ticket_details, issued_by_badge, issued_by_name, issue_date,# driver_license, #issued_by_badge)

officers(id, name, badge_number)

normal_users(id, name, email, password, license_number, created_at)

vehicles(id, license_plate, owner_name, inspection_start, inspection_end, tax_paid, stolen_vehicle, make_and_model, color, driver_license, address, #license_number)

Insurance(id, licence_plate, owner, insurance_start, insurance_end, #licence_plate)

2.2.8 Data dictionary

Category	Element	Coding	Data Type	Notes
PoliceOfficer	Badge Number	badge_number	VARCHAR(50)	PK
	Name	name	VARCHAR(100)	
	Email	email	VARCHAR(255)	Unique
	Password	password	VARCHAR(255)	
	Rank	rank	VARCHAR(50)	
	Department	department	VARCHAR(100)	

Category	Element	Coding	Data Type	Notes
vehicleOwner	User ID	user_id	INT	PK
	Name	name	VARCHAR(100)	
	Email	email	VARCHAR(255)	Unique
	Password	password	VARCHAR(255)	
	Driver License	driver_license	VARCHAR(50)	Unique

Category	Element	Coding	Data Type	Notes
TrafficManager	Manager ID	manager_id	INT	PK
	Name	name	VARCHAR(100)	
	Email	email	VARCHAR(255)	Unique
	Password	password	VARCHAR(255)	
	Role	role	ENUM	Values: "Admin", "Manager"

Category	Element	Coding	Data Type	Notes
Vehicle	Vehicle ID	vehicle_id	INT	PK
	License Plate	license_plate	VARCHAR(20)	Unique
	Make and Model	make_and_model	VARCHAR(100)	
	Color	color	VARCHAR(50)	
	Owner ID	owner_id	INT	FK to vehicleOwner.user_id
	Is Stolen	is_stolen	BOOLEAN	

Category	Element	Coding	Data Type	Notes
Ticket	Ticket ID	ticket_id	INT	PK
	Vehicle ID	vehicle_id	INT	FK to Vehicle.vehicle_id
	Issued By	issued_by	VARCHAR(50)	FK to PoliceOfficer.badge_number
	Issue Date	issue_date	TIMESTAMP	
	Violation Type	violation_type	VARCHAR(100)	
	Details	details	TEXT	

	Status	status	ENUM	Values: "Draft", "Submitted", "Resolved"

Category	Element	Coding	Data Type	Notes
StolenReport	Report ID	report_id	INT	PK
	Vehicle ID	vehicle_id	INT	FK to Vehicle.vehicle_id
	Reported By ID	reported_by_id	INT	FK to PoliceOfficer.badge_number or vehicleOwner.user_id
	Reporter Role	reporter_role	ENUM	Values: "Police", "Owner"
	Report Date	report_date	TIMESTAMP	
	Status	status	ENUM	Values: "Pending", "Confirmed", "Recovered"
	Notes	notes	TEXT	

Category	Element	Coding	Data Type	Notes
Insurance	Insurance ID	insurance_id	INT	PK
	Vehicle ID	vehicle_id	INT	FK to Vehicle.vehicle_id
	Owner ID	owner_id	INT	FK to vehicleOwner.user_id
	Start Date	start_date	DATE	
	End Date	end_date	DATE	
	Provider Name	provider_name	VARCHAR(100)	
	Policy Number	policy_number	VARCHAR(50)	

3. Conclusion

This chapter allowed us to present the approach followed to address the problem in question, whether through the description of the system hierarchy or the diagrams outlining the structure of the app. However, the practical implementation still remains — which will be the focus of the next chapter.

SECTION III
SYSTEM IMPLEMENTATION

1. Introduction

In this chapter, we will discuss the tools and specific programming languages used in the development and implementation of the website. We will also present the graphical interfaces of this application.

2. Tools Used

2.1 Room database

Room is a persistence library introduced as part of the Android Jetpack suite to simplify working with SQLite databases. It provides:
A structured approach to manage data.
Compile-time checks for SQL queries.
Seamless integration with modern Android components like LiveData and Coroutines. [1]

2.2 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development. Based on the powerful code editor and developer tools from IntelliJ IDEA , Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Live Edit to update composables in emulators and physical devices in real time
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks. [2]

3. Programming languages and technologies used

3.1 Kotlin Language

Kotlin is an open-source, statically-typed programming language that supports both object-oriented and functional programming. Kotlin provides similar syntax and concepts from other languages, including C#, Java, and Scala, among many others. Kotlin does not aim to be unique, instead, it draws inspiration from decades of language development. It exists in variants that target the JVM (Kotlin/JVM), JavaScript (Kotlin/JS), and native code (Kotlin/Native). [3]

3.2 Mysql

"SQL", the acronym for Structured Query Language. A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. SQL is a language that programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups. [4]

3.3 Jetpack Compose

Jetpack Compose is Android's recommended modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs. [5]

3.4 Dagger Hilt

Dagger Hilt is a dependency injection library for Android, built on top of Dagger 2. Hilt is designed to simplify the implementation of dependency injection in Android apps by reducing boilerplate code and providing seamless integration with Android components such as activities, fragments, services, and ViewModels. It leverages the power of Dagger under the hood while offering a more concise and intuitive API for developers. [6]

3.5 Retrofit

Retrofit is a type-safe HTTP client for Android and Java, library that simplifies this process by providing a clean and efficient way to make API calls. [7]

3.6 Nodejs

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project!
Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant.
[8]

3.7 Express

Express is an open-source web application framework for Node.js. It provides a robust set of features for building web and mobile applications, including routing, middleware, template engines, seamless database integration, and a wealth of features for developing advanced features and functions. [9]

3.8 Render

Render is a cloud-based platform that simplifies the process of deploying and hosting web applicatoins, APIs, static sites, and more. It automates tasks like building, deploying, and scaling applications, making it easier for us to bring our project to production.[10]

3.9 Clever Cloud

Clever Cloud provides an automated hosting platform for developers. Deploy your app easily and launch dependencies without having to worry about the infrastructure set up. Follow this guide to get ready to deploy quickly as you learn the basics of Clever Cloud. [11]

3.10 Postman

Postman is an all-in-one API platform for building and working with APIs. It takes the pain out of every stage of the API lifecycle—from designing and testing to delivery and monitoring. Built for teams, Postman makes it easy to collaborate, stay organized, and build secure, reliable APIs faster. [12]

3.11 Figma

Figma is a collaborative web application for interface design, with additional offline features enabled by desktop applications for macOS and Windows. The feature set of Figma focuses on user interface and user experience design. [13]

3.12 Git & GitHub

Git is a version control system that intelligently tracks changes in files. Git is particularly useful when you and a group of people are all making changes to the same files at the same time.
GitHub is a cloud-based platform where you can store, share, and work together with others to write code. [14]

4. Summary table and

Component	Technology Used
Mobile App	Kotlin, Jetpack Compose
Backend APIs	JavaScript (Node.js + Express)
Database	MySQL
UI/UX Design	Adobe Figma
Deployment	Render (APIs), Clever Cloud (Databases)
Local Data Storage	Room (Android SQLite ORM)
Dependency Injection	Dagger Hilt
Network Communication	Retrofit, Axios
Security	Bcrypt (password hashing), JWT (token

	authentication)
IDEs	Android Studio, Visual Studio Code
Version Control	Git, GitHub
API Testing	Postman

5. Api Secutiry Mesures

To ensure the robustness, integrity, and privacy of data exchanged through our API, we have implemented a comprehensive set of security measures. These practices are designed to protect against common attack vectors and unauthorized access while maintaining the usability of the API for legitimate clients.

5.1 JWT-Based Authentication

Our API uses JSON Web Tokens (JWT) for secure and stateless authentication. When a user logs in, a signed token is issued and must be included in the Authorization header for all subsequent requests. This ensures that only authenticated users can access protected resources. Tokens are signed with a secret key stored in environment variables to prevent tampering.

5.2 HTTPS-Only Communication

All communication with the API is enforced over HTTPS to protect against man-in-the-middle (MITM) attacks. HTTPS encrypts both request and response traffic, ensuring data confidentiality and integrity during transit.

5.3 CORS Configuration

We implemented Cross-Origin Resource Sharing (CORS) to restrict API access to trusted clients only. This is configured to allow requests only from specific origins—such as our Android application's domain—effectively preventing unauthorized external web applications from interacting with the API.

5.4 Rate Limiting

To prevent abuse, brute-force attempts, and denial-of-service (DoS) attacks, we've added rate limiting. This restricts the number of requests an IP address can make within a defined time window, ensuring fair usage and safeguarding API availability.

5.5 Input Validation & Sanitization

All incoming requests are subjected to strict validation and sanitization rules to mitigate risks such as SQL injection, NoSQL injection, and cross-site scripting (XSS). This is enforced using middleware that checks the format, type, and integrity of request parameters and body fields.

5.6 Secure HTTP Headers (Helmet)

We use the **Helmet** middleware to automatically set HTTP headers that protect the API against common vulnerabilities such as:

- Clickjacking (X-Frame-Options)
- MIME-type sniffing (X-Content-Type-Options)
- Cross-site scripting (Content-Security-Policy)
- HTTP Strict Transport Security (Strict-Transport-Security)

These headers help enforce secure browser behavior and reduce attack surfaces.

5.7 Loging of Sensitive Info

Sensitive actions and incoming requests are logged using Winston, a robust logging library. This allows for effective monitoring, debugging, and incident investigation without exposing sensitive data in logs. Logs are structured and stored securely for audit purposes.

5.8 Preventing Sensitive Data Exposure

To avoid leaking internal implementation details, the API is configured to return generic error messages for unexpected server-side errors. Detailed stack traces and error logs are not exposed to clients, reducing the risk of attackers gaining insights into the internal logic or structure.

5.9 Use of Environment Variables

Secrets such as database credentials, API keys, and JWT secrets are stored securely using environment variables. This ensures that no sensitive information is hardcoded into the codebase, supporting both security and maintainability across environments.

5.10 Summary

These combined measures significantly enhance the security posture of our API while maintaining performance and developer usability. Our approach ensures:

- Only authorized users can access sensitive endpoints.
- Data is protected in transit and at rest.
- Malicious input is filtered out.
- The system is resilient to abuse and misconfiguration.

Security is a continuous process, and we will continue to monitor, update, and improve our protections as the API evolves.

6. Presentation of the Developed Application

Before starting the development, it was necessary to prepare the essential tools. After familiarizing ourselves with the development environment, we first integrated the necessary libraries and then defined the usage scenario of the application. The different stages are described below:

6.1 Scenario:

Our application is designed for police officers to facilitate the management and tracking of vehicles. It allows officers to quickly access vehicle information by scanning or entering a license plate number.
The application is connected to a centralized database where vehicle information is stored, including the registration certificate, the owner's driver's license, insurance details, technical inspection status, and road tax payment.
With this application, law enforcement can verify the administrative status of vehicles in real time, detect potential violations, and optimize their field interventions. The application requires an internet connection to query the remote database and ensure real-time updates of information.

Figure XXXI Application

6.2 Description of the Application Interface:

In the following, we present the different interfaces of the application, detailing each screenshot.

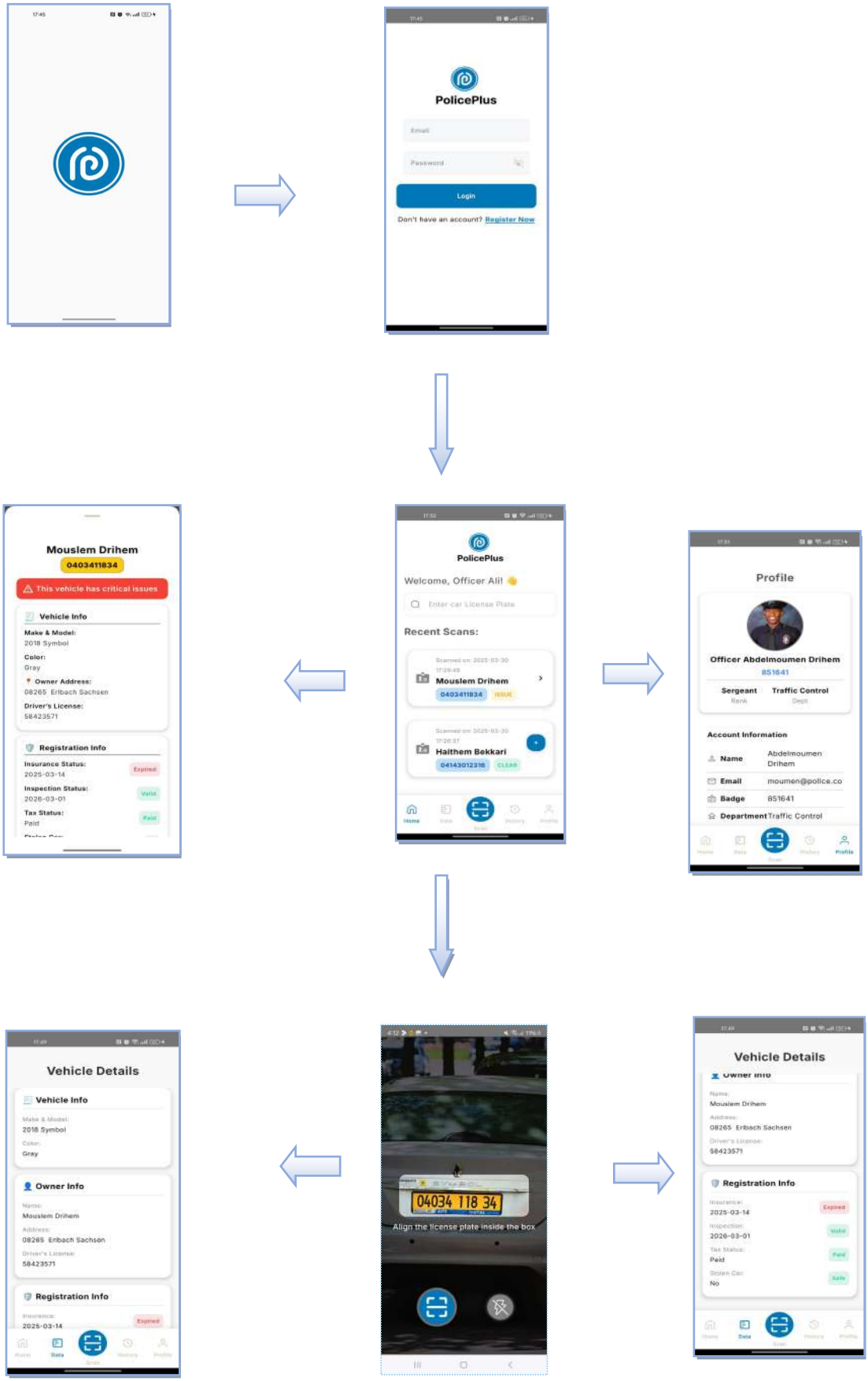


Figure XXXII General Presentation of the Parent Application

The application provides two login options either 'Police Officer' or 'User':

To begin, the officer is required to enter a 'Username' and 'Password' when launching the application. If no account exists, registration must be completed by providing a Badge Number, Rank, Department, and Email address.

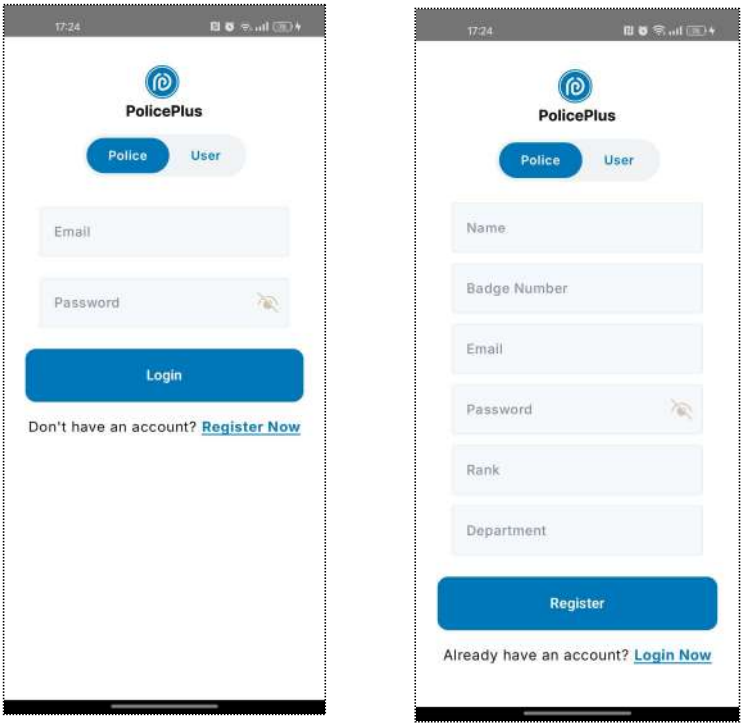


Figure XXXIII Police Officer Login & Register page

a. «Main» Interface:

This is the main interface. It is displayed after verifying the 'Username' and 'Password' and shows the user's activities. A message will appear asking for permission to receive notifications from the application. The interface allows the user to choose from the options: 'Home,' 'Data,' 'Scan,' 'History,' and 'Profile'.

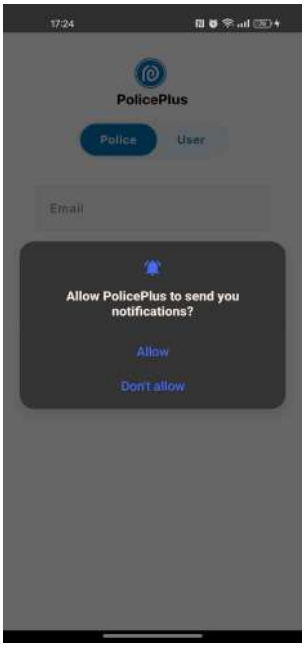


Figure XXXIV permission to receive notifications

b. «Home» Interface:

Use the search bar for quick vehicle lookups by entering the vehicle number and clicking 'Search' to display the vehicle's details. Additionally, the last three scanned vehicles and the total number of scanned vehicles are displayed.

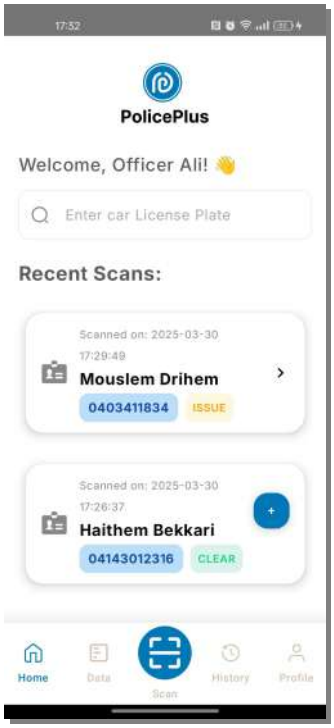


Figure XXXV Main menu 'Home'

c. «Scan» Interface:

License Plate Recognition and Data Retrieval: The officer can scan the vehicle number and confirm the scanned details.

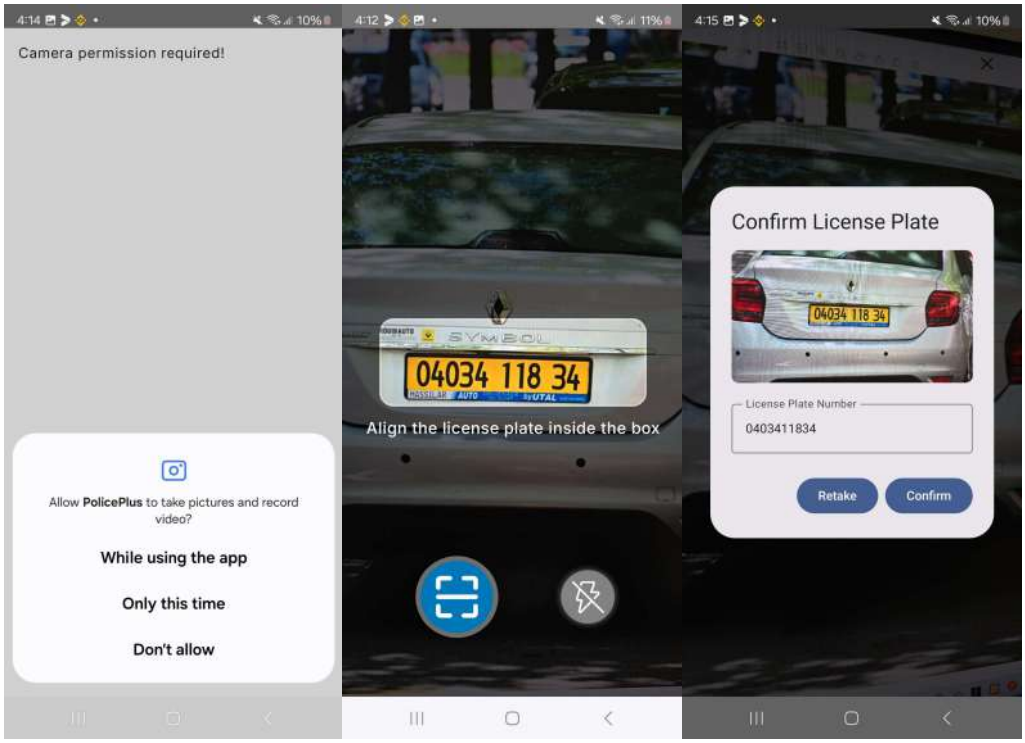
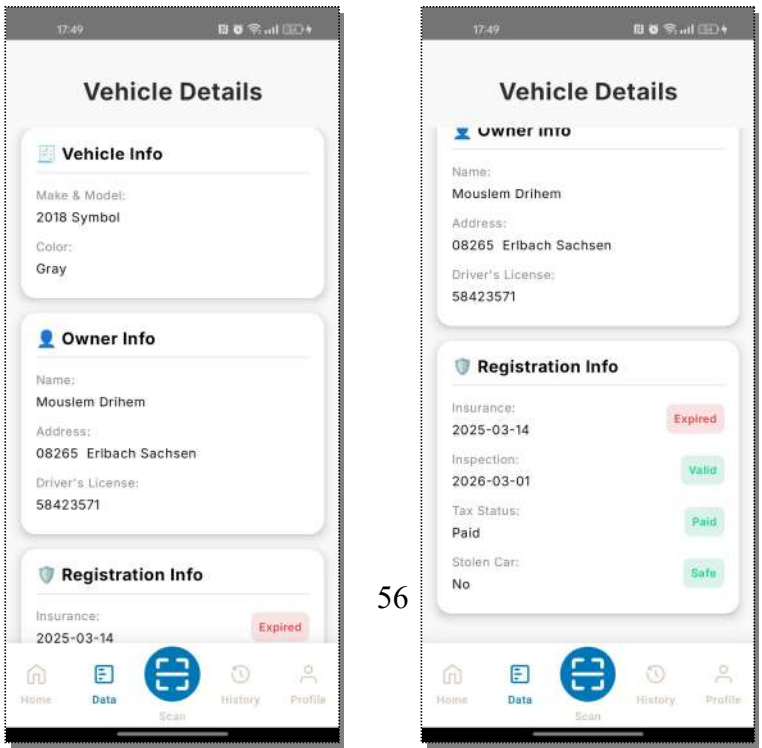


Figure XXXVI Scan Interface

d. «Data» Interface:

Here, all information is displayed, including owner details, insurance start and end dates, stolen vehicle status, and any unpaid tickets.

Figure XXXVII Vehicle Details Screen



e. «History» Interface:

The app allows officers to view previous scans with full details and filter records by selecting 'Stolen,' 'Issue,' or 'All.

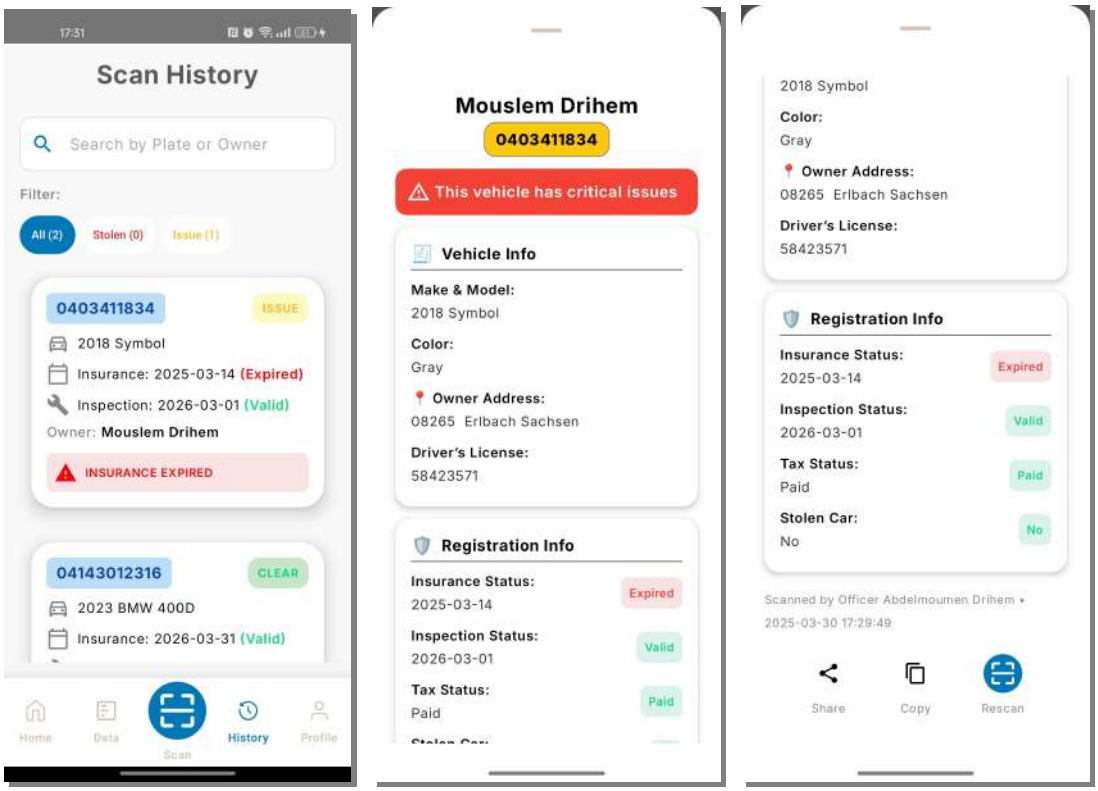


Figure XXXVIII Scan History Interface

f. «Profile» Interface:

The officer can view account information and log out when needed.

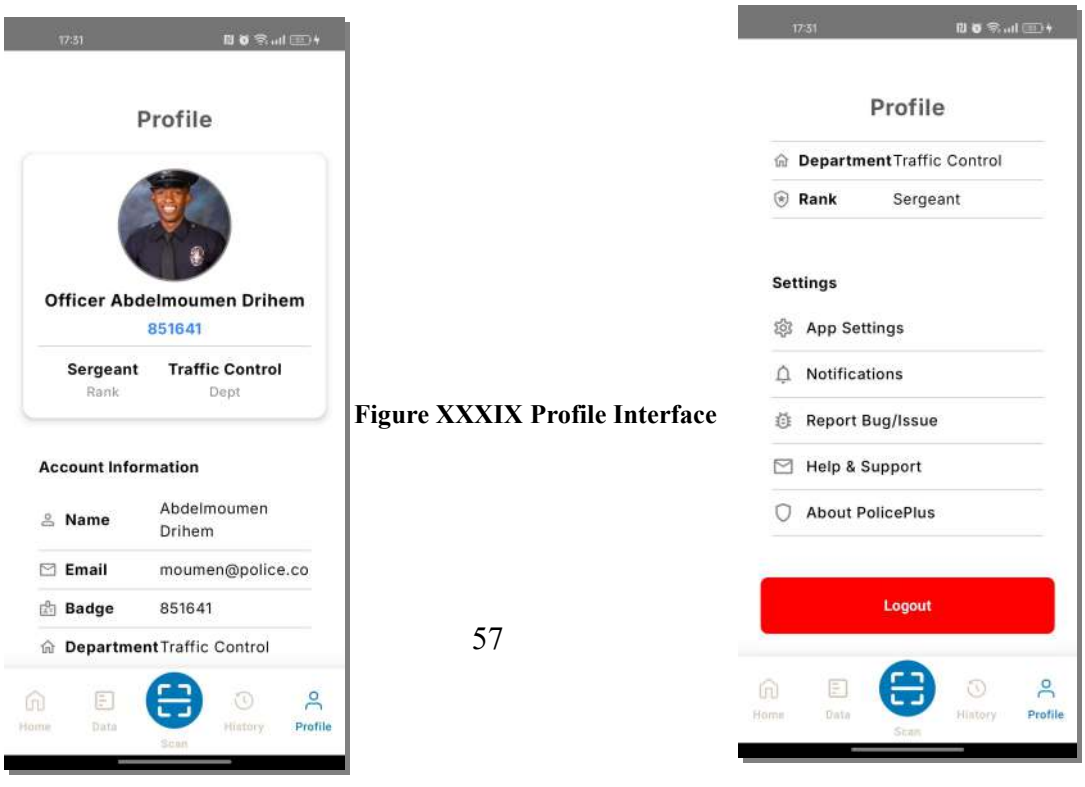


Figure XXXIX Profile Interface

g. Ticket Interface:

The officer can issue a ticket or report a stolen vehicle(for users too).

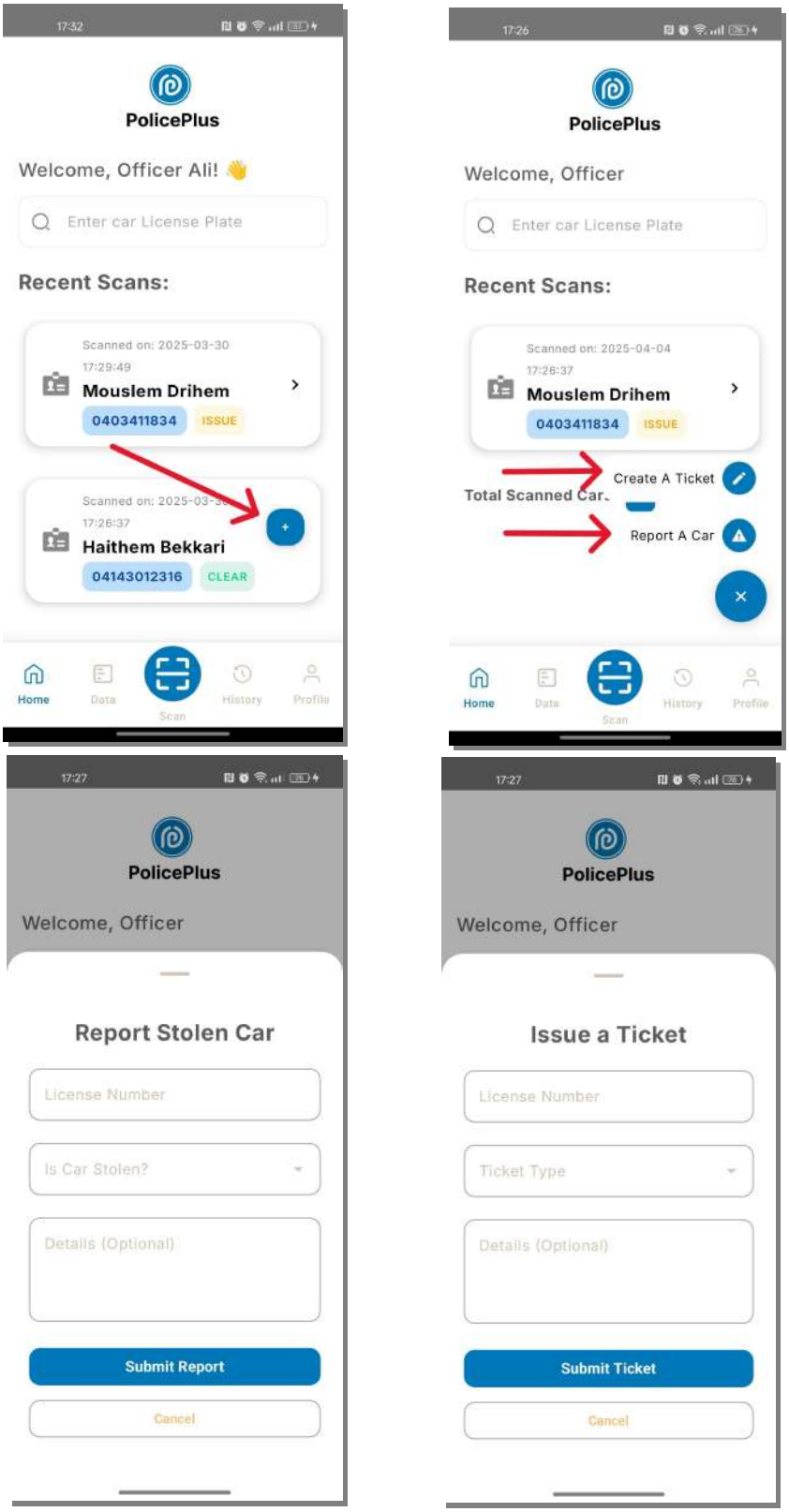


Figure XL issue a ticket(right), Report Stolen vehicle(left)

Alternatively, the user must enter their Email and Password. If they do not already have an account, they will need to register by providing their Name, Email address, Password, and vehicle License Plate.

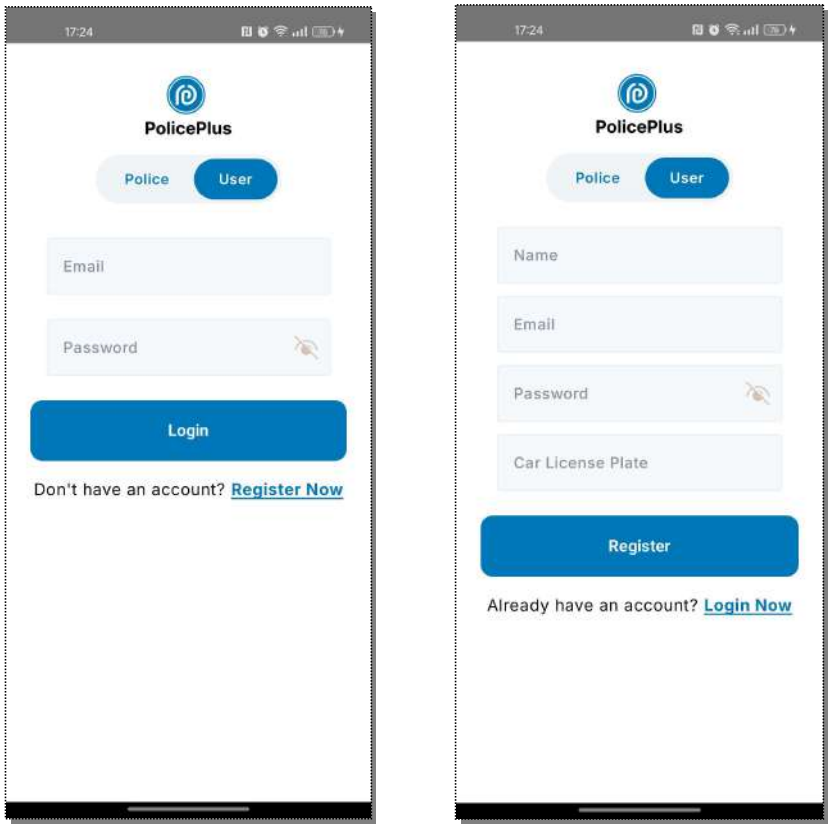


Figure XLI User Login & Register page

«Home» Interface "User":

All vehicle details are displayed here, including the owner, model, color, driving license, and registration information such as insurance, inspection, vehicle registration, and tax.

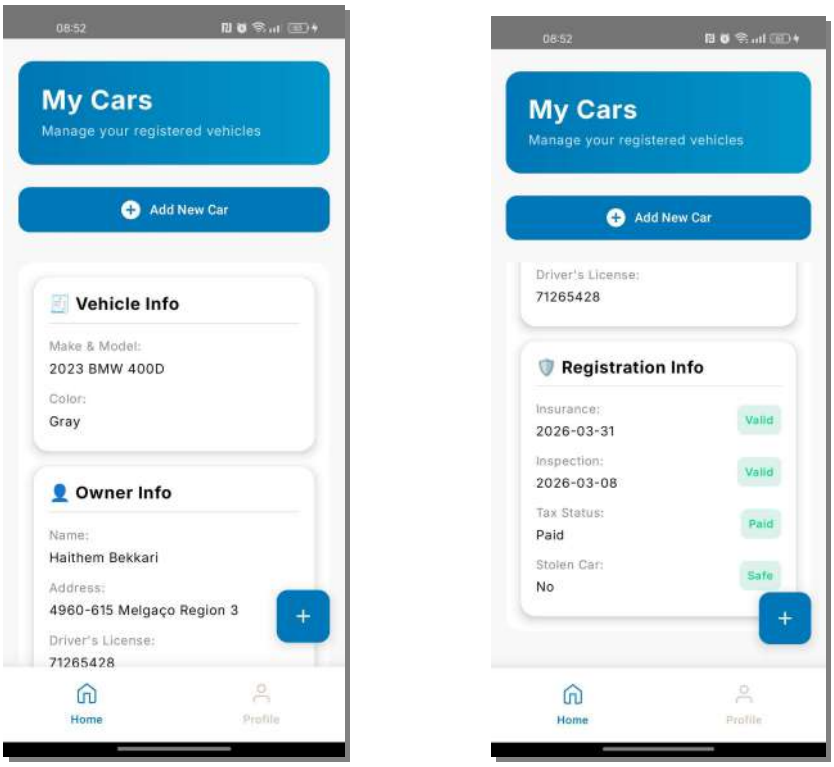


Figure XLII Home page

h. Add new vehicle Interface :

To add a new vehicle, click the button and enter the license plate number.

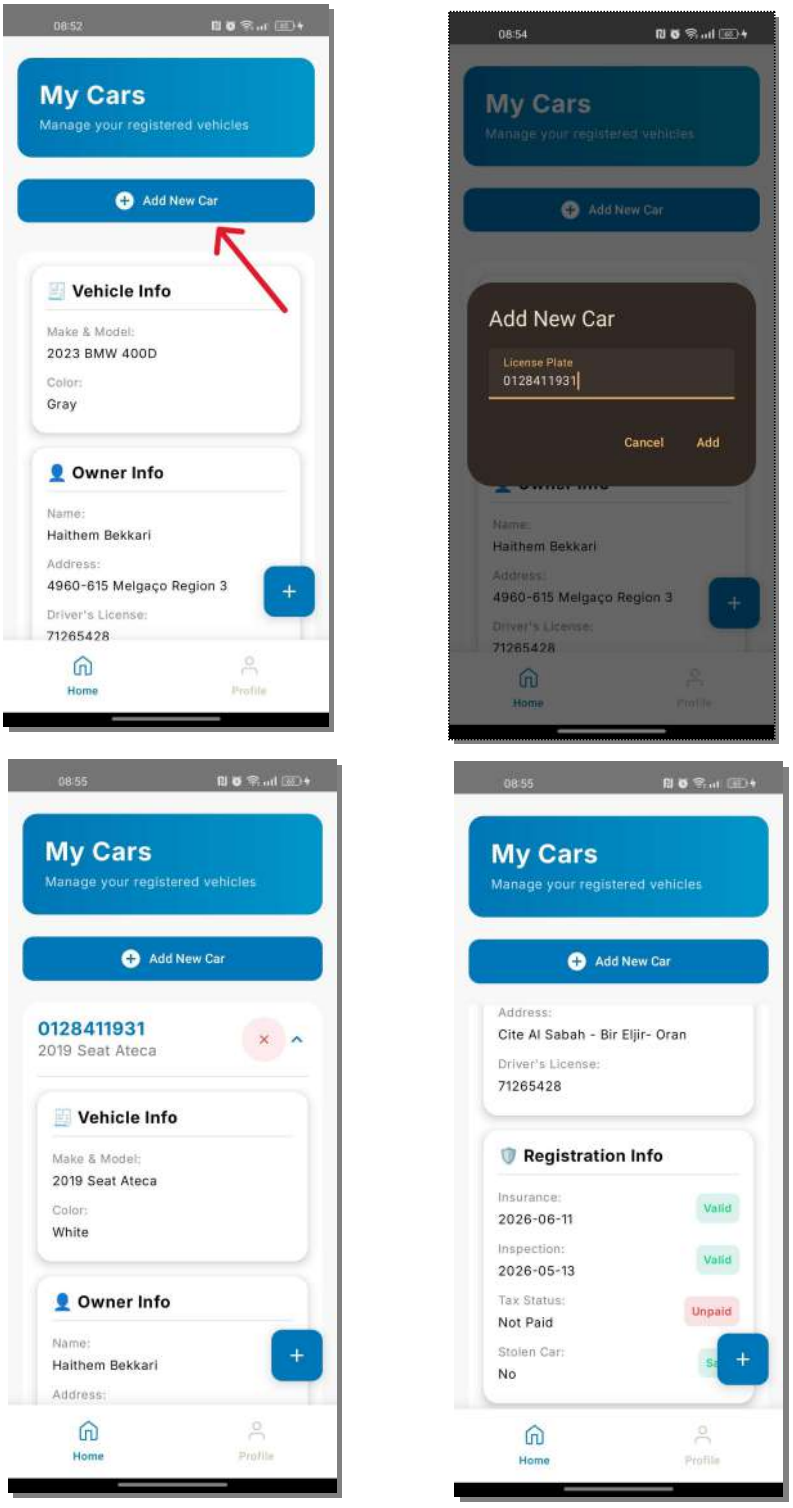


Figure XLIII Adding a new vehicle for drivers

User profile :

The Profile page displays the user's account information, including their name, email address, and driver's license details.

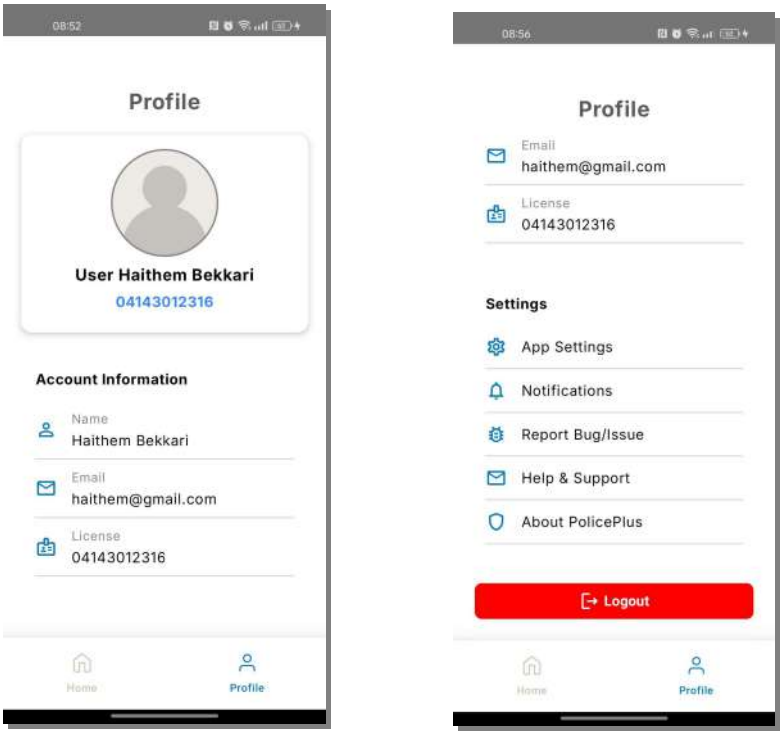


Figure XLIV Driver Profile

i. Report a vehicle

The user can report a vehicle by providing the following information:

- License Number
- Indicate if the vehicle is stolen (Yes/No)
- Additional details (optional)

Then, click submit report.

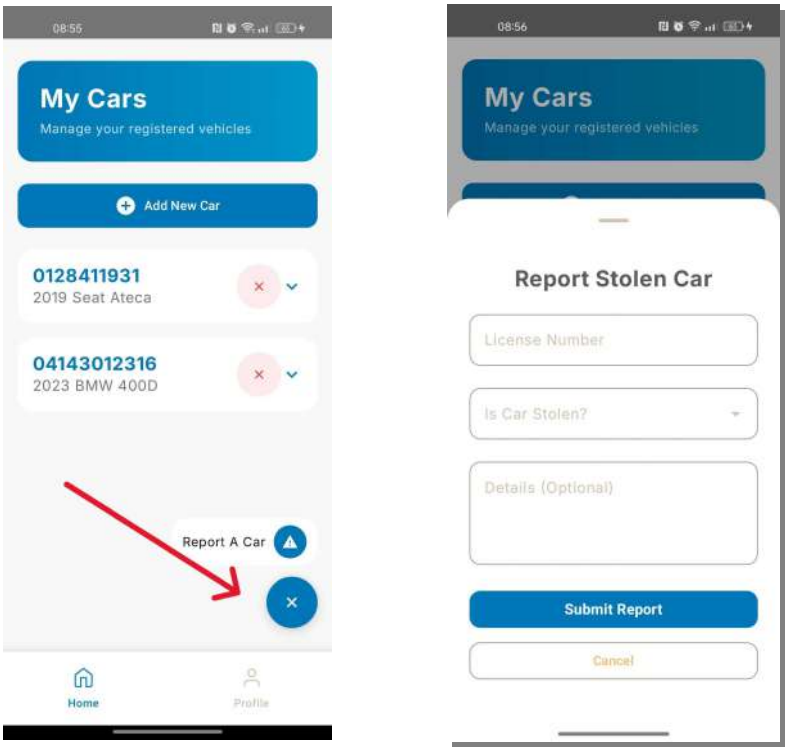


Figure XLV Report a stolen vehicle

7. Conclusion

In this chapter, we presented the development environment and introduced the core user interfaces of our police application. These interfaces were vehicleefully designed to ensure ease of use, responsiveness, and efficiency for all users, particularly police officers. The application provides a comprehensive set of features that support essential tasks such as vehicle tracking, reporting incidents, and accessing critical data in real-time. Whether it's issuing a ticket, verifying vehicle status, or filing a stolen vehicle report, the interfaces streamline the workflow and improve operational productivity. Overall, this chapter highlighted how the system architecture and UI design work together to enhance user experience and ensure that both functionality and performance meet the demands of daily police operations.

General conclusion

This project focuses on monitoring vehicle statuses and supporting police officers in verifying and managing vehicle-related information effectively. We have developed a mobile application tailored for law enforcement agencies, enabling officers to access critical vehicle data in real time. Through the app, police officers can check and validate important documents such as the vehicle registration certificate, driver's license, insurance records, technical inspection dates, and tax receipts. The system also helps in detecting anomalies such as expired documents or reports of stolen vehicles, allowing officers to receive alerts and take immediate action when necessary.

In addition to police functionality, the application also offers a user-friendly interface for vehicle owners. Regular users can log in to view their vehicle details, track the status of their documents, and report a stolen vehicle directly from the app—enhancing transparency and user participation in maintaining public safety.

For development, we utilized Android Studio for mobile application development, Adobe Figma for designing intuitive user interfaces, and MySQL for managing and storing structured vehicle and user data. Looking ahead, we plan to implement offline capabilities and strengthen the system's security infrastructure to ensure higher reliability, better performance, and protection against unauthorized access.

List of sources and references

[1] : Room Database in Android(medium.com)
[2] : Meet Android Studio(developer.android.com)
[3] : Kotlin overview(developer.android.com)
[4] : MySQL(wikipedia)
[5] : Jetpack Compose(developer.android.com)
[6] : Understanding Dagger Hilt: Simplified Dependency Injection for Android(medium.com)
[7] : Retrofit in Android(medium.com)
[8] : Introduction to Node.js(nodejs.org)
[9] : What is Express.js? (medium.com)
[10] : Slow rendering(developer.android.com)
[11] : Clever cloud(clever-cloud.com)
[12] : What is Postman?(postman.com)
[13] : Figma(wikipedia)
[14] : About GitHub and Git (docs.github.com)
[15] : UML (fr.wikipedia.org)
[16] : Two Tracks Unified Process(fr.wikipedia.org)
[17] : Vignette Automobile(algeriainvest.com)
[18] : Significant Developments(taxsummaries.pwc.com)