

VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING



Web Application Development - IT093IU

Final Report

Student ID	Name	Email
ITCSIU23056	Phạm Hoàng Phương	ITCSIU23056@student.hcmiu.edu.vn
ITCSIU23059	Mai Long Thiên	ITCSIU23059@student.hcmiu.edu.vn
ITCSIU23033	Đặng Ngọc Thái Sơn	ITCSIU23033@student.hcmiu.edu.vn

I. INTRODUCTION

1. ABOUT THE PROJECT

The Vehicle Fault Detection system is an AI-powered platform designed to automate the detection and reporting of traffic violations in real-time, addressing the challenges of manual monitoring in high-traffic urban environments, particularly in Vietnam. By integrating advanced computer vision techniques with a flexible rule-based engine, the system enables accurate identification of common violations such as wrong-lane driving, illegal parking, prohibited area entry, helmet non-compliance, and red-light running.

Developed using a modern full-stack architecture — FastAPI for the backend, PostgreSQL for data persistence, YOLO-based object detection and tracking for video processing, and a custom Domain-Specific Language (DSL) for configurable violation rules — the platform supports role-based access control (admin, police, and customer) and provides realtime alerts via WebSocket, comprehensive violation reporting, and administrative tools for managing cameras, zones, and user assignments.

This report presents the system's functional requirements, architecture, key components, API design, and implementation details, demonstrating how the solution effectively enhances traffic enforcement efficiency while maintaining scalability and extensibility for future enhancements.

2. DEVELOPMENT ENVIRONMENT

Since this is a full-stack web application, the project is developed using a modern architecture with a clear separation between frontend and backend, following the principles of modular design and API-first development.

The following technologies and programming languages are used within our system:

1. **Frontend (React.js):** To build a responsive, interactive user interface for the dashboard, real-time monitoring, configuration tools, and report viewing. React is chosen for its component-based structure, enabling efficient rendering of real-time video streams, WebSocket alerts, and dynamic forms for zone drawing and rule management.
2. **Backend (FastAPI - Python):** To handle API requests, authentication (JWT), business logic, and integration with the AI engine. FastAPI is selected for its high performance, automatic OpenAPI documentation (Swagger UI), and native support for asynchronous operations and WebSockets.
3. **Database (PostgreSQL):** To store persistent data, including users, sources (cameras/videos), zones, rules, assignments, and violation records. PostgreSQL is used for its robustness, support for JSONB fields (used in zone coordinates), and reliable transaction management.

4. **AI Processing Engine (Python with Ultralytics YOLOv10):** To perform real-time object detection, tracking, attribute recognition (e.g., helmet detection), and frame visualisation (drawing zones and bounding boxes).
5. **Real-time Communication (WebSocket):** Implemented via FastAPI WebSockets to push violation alerts instantly from the backend to connected frontend clients (police and admin dashboards).
6. **Development and Collaboration Tools:**
 - **Docker & Docker Compose:** For containerization and a consistent development environment across team members.
 - **Git & GitHub:** For version control and collaborative development.
 - **Google Docs/Drive, Notion:** For documentation, meeting notes, and shared resources
 - **Diagramming Tools:** Mermaid online for creating Entity Relationship Diagrams, Use Case diagrams, and Sequence diagrams.
 - **Project Management:** Trello to track overall progress and individual team member tasks.

The system follows a clean architecture with clear separation of concerns: frontend (React) consumes REST APIs and WebSocket from the backend (FastAPI), which interacts with PostgreSQL and runs background AI processing workers.

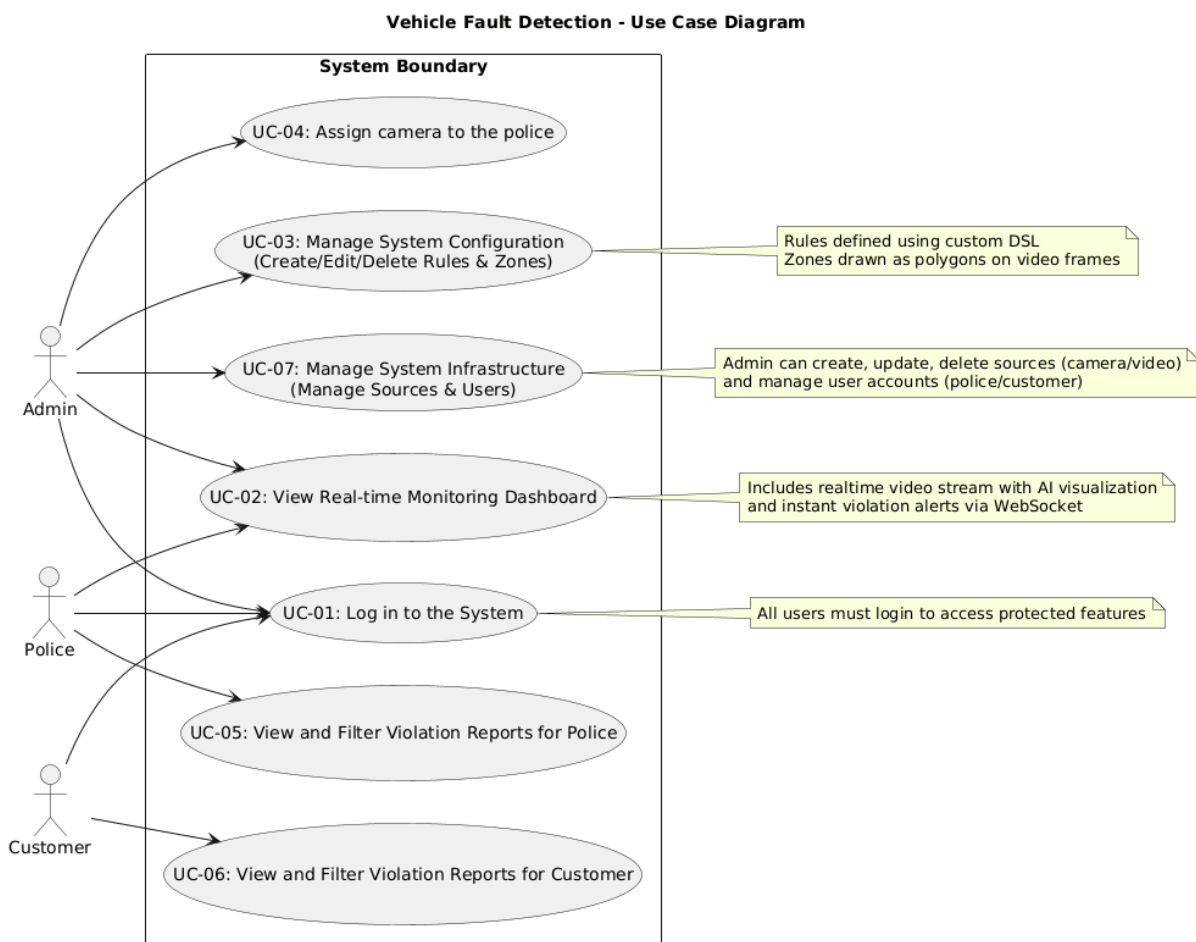
This modern stack ensures the system is scalable, maintainable, and capable of delivering the real-time performance required for traffic violation monitoring.

II. REQUIREMENT ANALYSIS AND DESIGN

This section outlines the requirement analysis and design process for the Vehicle Fault Detection. The design is based on detailed functional and non-functional requirements gathered from stakeholders, ensuring the system meets real-world needs in traffic enforcement. The implementation follows an iterative approach, with continuous refinement of specifications to track project progress and adapt to new insights.

1. REQUIREMENT ANALYSIS

Use case diagrams



Use Case 1: Login to the system

- **Name:** Log in to the System
- **Identifier:** UC-01
- **Inputs:**
 1. Email/User name
 2. Password

- **Outputs:**
 1. The Dashboard page with the User's authorisation [If successful]
 2. The Login page with an error message [If fail]
- **Basic Course:**

Actor: Police/Admin/Customer	System
1. Open the login page	1.1. Display the login page
2. Enter email and password	
3. Submit	3.1. Check the User's info against the database - If successful, return the Dashboard page - Else return the login page with an error

- **Precondition:**
 1. User has a registered account that was created earlier (email and password).
- **Post condition:**
 1. User is successfully authenticated, and a session token (JWT) is established in the client.
- **User story:** As a system user, I want to log in to the system so that I can access the monitoring and configuration features.

Use Case 2: View dashboard

- **Name:** View Monitoring Dashboard
- **Identifier:** UC-02
- **Inputs:**
 1. None
- **Outputs:**
 1. A dashboard displaying AI-processed video streams.
- **Basic Course:**

Actor: Police/Admin	System
1. Navigate to the Dashboard page	1.1. Display the main dashboard layout

- **Precondition:**
 1. Police/admin has successfully logged in to the system (UC-01).
- **Post condition:**
 1. None

- **User story:** As a traffic operator, I want to view live camera feeds and violation alerts in real-time so that I can monitor the current traffic situation effectively.

Use Case 3: Manage System Configuration (Rules and Zones)

- **Name:** Manage System Configuration (Rules and Zones)
- **Identifier:** UC-03
- **Inputs:**
 1. DSL rule text for creating or updating a rule.
 2. Zone name and polygon coordinates for creating or updating a zone.
- **Outputs:**
 1. A success message confirming the configuration has been saved.
 2. An error message if the input is invalid (e.g., DSL syntax error).
 3. The UI is updated to reflect the new or modified configuration.
- **Basic Course:**

Actor: Admin	System
1. Navigate to the "Configuration" page	1.1. Display the interface for managing Rules and Zones
2. Choose to create/edit a Rule or a Zone	2.1. Display the appropriate form (a text editor for a Rule, a drawing interface for a Zone)
3. Enter the required information and submit	3.1. Validate the submitted input (check DSL syntax, validate coordinates) - If valid, save the configuration to the database. Return a success response and update the UI - If invalid, return an error response with details

- **Precondition:**
 1. Admin has successfully logged in to the system (UC-01).
- **Post condition:**
 1. A new or updated Rule/Zone is persisted in the database and becomes active for the Rule Engine to use.
- **User story:** As a system admin, I want to define custom violation rules and draw detection zones so that I can tailor the system's logic to my specific monitoring needs.

Use Case 4: Assign Camera to Police

- **Name:** Assign Camera to Police
- **Identifier:** UC-04
- **Inputs:**
 1. Police user ID
 2. A list of Camera IDs to be assigned
- **Outputs:**
 1. A success message confirming the assignment.
 2. An error message if the Police user or Camera does not exist.
- **Basic Course:**

Actor: Admin	System
1. Navigate to the "Police Management" or "Camera Assignment" page	1.1. Display lists of Police users and available Cameras
2. Select a Police user	
3. Select one or more cameras to assign	
4. Submit the assignment	4.1. Validate that the user has 'Admin' privileges
4.2. Check if the selected Police user and Cameras exist	
4.3. Create/update records in the police_camera_assignments table	
4.4. Return a success message	

- **Precondition:**
 1. User has successfully logged in as an 'Admin' (UC-01).
 2. At least one 'Police' user and one 'Camera' exist in the system.
- **Post condition:**
 1. The association between the specified Police user and the selected cameras is created or updated in the database.
- **User story:** As an administrator, I want to assign specific cameras to each police officer so that they can only monitor the areas relevant to their duties.

Use Case 5: View and Filter Violation Reports for Police

- **Name:** View and Filter Violation Reports for Police
- **Identifier:** UC-05
- **Inputs:**
 1. Filter criteria (optional): Date range, rule name.
- **Outputs:**
 1. A list of historical violation records that occurred on the cameras assigned to the logged-in Police user.
 2. Detailed information and evidence for a selected violation.
- **Basic Course:**

Actor: Police	System
1. Navigate to the "Reports" page	1.1. Identify the logged-in Police user
1.2. Retrieve the list of cameras assigned to this user	
1.3. Display an initial list of recent violations from only the assigned cameras	
2. Apply filter criteria and submit	2.1. Query the database for violations matching the criteria AND belonging to the assigned cameras
2.2. Display the updated, filtered list of violations	

- **Precondition:**
 1. User has successfully logged in as a 'Police' user (UC-01).
 2. The logged-in Police user has been assigned at least one camera by an Admin (UC-04).
- **Post condition:**

1. None (this is a read-only operation with data authorisation).
- **User story:** As a police officer, I want to review and filter violation reports from my assigned cameras so that I can process infringements within my jurisdiction.

Use Case 6: View Violation for Customer

- **Name:** View Violation Reports for Customer
- **Identifier:** UC-06
- **Inputs:**
 1. Filter criteria (optional): Date range, rule name.
- **Outputs:**
 1. A list of historical violation records associated with the customer's registered license plate.
 2. Detailed information and evidence for a selected violation.
- **Basic Course:**

Actor: Customer	System
1. Navigate to the "My Violations" page	1.1. Identify the logged-in Customer
1.2. Retrieve the customer's license plate from their profile	
1.3. Query the violations table for records where detected_license_plate matches the customer's plate	
1.4. Display the list of their personal violations	
2. Apply filter criteria and submit	2.1. Re-query the database with the added filter conditions (date, etc.)
2.2. Display the updated, filtered list of personal violations	

- **Precondition:**
 1. User has successfully logged in as a 'Customer' (UC-01).
 2. The customer's account has a registered license plate.
 3. The AI system is capable of performing Automatic Number Plate Recognition (ANPR).
- **Post condition:**
 1. None (this is a read-only operation).

- **User story:** As a vehicle owner, I want to view a list of my own traffic violations so that I can be aware of them and handle the fines accordingly.

Use Case 7: Manage System Infrastructure

- **Name:** Manage System Infrastructure
- **Identifier:** UC-07
- **Inputs:**
 1. Camera details (name, source URL).
 2. User account details (email, role: 'Police' or 'Customer', license plate for customers).
- **Outputs:**
 1. A new/updated Camera is active in the system.
 2. A new/updated User account is created in the system.
 3. A confirmation message of the action.
- **Basic Course:**

Actor: Admin	System
1. Navigate to the "Admin Panel"	1.1. Display options to manage Cameras and Users (Police, Customers)
2. Choose to create/edit a Camera or a User	2.1. Display the corresponding form
3. Enter the required details and submit	3.1. Validate the user's 'Admin' privileges
3.2. Validate the input data (e.g., unique email, valid URL)	
3.3. Create or update the record in the cameras or users table	
3.4. Return a success response and refresh the UI	

- **Precondition:**
 1. User has successfully logged in as an 'Admin' (UC-01).
- **Post condition:**
 1. A new Camera is available for assignment and processing.
 2. A new Police or Customer account is created and can be used to log in.
- **User story:** As an administrator, I want to manage the system's core assets like cameras and user accounts to maintain and scale the platform.

2. FUNCTIONAL REQUIREMENTS

Use Case 1: Log in to the System

1. **Scope of the Product:** This functionality serves as the entry gateway to the system, ensuring that only authenticated users can access protected features.
2. **Functional and Data Requirements:**
 - **Functional Requirements:**
 - The system shall provide a login endpoint for the user to submit credentials (email and password).
 - The system shall validate the user's credentials against the records stored in the database.
 - The system shall use the bcrypt hashing mechanism to compare passwords securely.
 - The system shall generate and return a JSON Web Token (JWT) containing the user ID and role upon successful authentication.
 - The system shall return an appropriate error message (401 Unauthorised) upon failed authentication.
 - **Data Requirements:**
 - The user's credentials (email, hashed_password) must exist in the users table.
 - The input email must be in a valid format (validated by Pydantic EmailStr).

Use Case 2: View Monitoring Dashboard

1. **Scope of the Product:** This is the primary workspace for Police and Admin users, providing live video streams processed by AI and immediate violation alerts.
2. **Functional and Data Requirements:**
 - **Functional Requirements:**
 - The system shall provide REST endpoints to list sources and start/stop AI processing for each source.
 - The system shall support MJPEG streaming of processed frames (with bounding boxes, zones, and violation highlights) via `/api/v1/sources/{source_id}/stream`.
 - The system shall push new violation events to connected clients via WebSocket as soon as they are detected.
 - The frontend shall render the MJPEG stream and display a list of recent violation alerts.
 - **Data Requirements:**
 - Processed frame data shall be encoded as JPEG and sent via a multipart stream.
 - Violation alert data shall be formatted in JSON and broadcast via WebSocket.

Use Case 3: Manage System Configuration

1. **Scope of the Product:** This functionality allows administrators to customise violation detection logic without modifying code.
2. **Functional and Data Requirements:**
 - **Functional Requirements (Rule Management):**
 - The system shall provide CRUD endpoints for violation rules using a custom Domain-Specific Language (DSL).
 - The system shall store DSL rules in the database and apply them during frame evaluation.
 - The system shall allow creation, update, and deletion of rules via API.
 - **Functional Requirements (Zone Management):**
 - The system shall provide CRUD endpoints for polygonal zones associated with specific sources.
 - The system shall store zone coordinates as JSONB in the database.
 - The system shall render drawn zones on the real-time video stream for visual verification.
 - **Data Requirements:**
 - A rule's DSL content must follow the defined DSL grammar.
 - A zone must have at least 3 vertices and be linked to a valid source.

Use Case 4: Assign Sources to Police

1. **Scope of the Product:** This feature enables administrators to control which sources (cameras/videos) each police officer can monitor.
2. **Functional and Data Requirements:**
 - **Functional Requirements:**
 - The system shall provide endpoints for administrators to assign multiple sources to a police user.
 - The system shall enforce access control so that police users can only view and process their assigned sources.
 - The system shall provide endpoints for police to retrieve their assigned sources.
 - **Data Requirements:**
 - Assignments are stored in the `police_source_assignments` table (many-to-many relationship).

Use Cases 5 & 6: View and Filter Violation Reports

1. **Scope of the Product:** This feature supports post-event analysis for both Police (assigned sources) and Customers (personal license plate).
2. **Functional and Data Requirements:**
 - **Functional Requirements:**
 - The system shall provide endpoints to list violations with filtering by date range, rule type, and source.
 - The system shall restrict police access to violations from their assigned sources.

- The system shall restrict customer access to violations matching their license plate.
- The system shall provide a detailed view of individual violations, including the evidence image URL.
- **Data Requirements:**
 - Each violation record must include `source_id`, `rule_id`, `timestamp`, `detected_license_plate`, `evidence_url`, and metadata (including `bbox`).

Use Case 7: Manage System Infrastructure

1. **Scope of the Product:** Administrative functions for managing foundational components of the system.
2. **Functional and Data Requirements:**
 - **Functional Requirements (Source Management):**
 - The system shall allow administrators to create, update, and soft-delete sources (video files or camera URLs).
 - **Functional Requirements (User Management):**
 - The system shall allow administrators to create, update, and soft-delete user accounts (police and customer roles).
 - Passwords shall be hashed using `bcrypt` before storage.
 - **Data Requirements:**
 - Sources must have either `file_path` (for video files) or `camera_url` (for RTSP streams).
 - User emails must be unique across the system.

These functional requirements reflect the current implementation state of the system, with real-time processing handled via background workers triggered by API calls, and visualisation provided through MJPEG streaming with AI overlays.

3. NON-FUNCTIONAL REQUIREMENTS

1. Security Requirements

These requirements ensure the integrity and confidentiality of the system and its data.

- **Authentication & Authorisation:**
 - All API endpoints (except for public endpoints like login/register) **must** be protected and require a valid JSON Web Token (JWT) for access.
 - The system **must** enforce role-based access control (RBAC), ensuring that only users with 'Admin' privileges can access infrastructure management APIs.
- **Data Security:**
 - User passwords **must** be securely hashed using a strong algorithm (e.g., `bcrypt`) before being stored in the database.

- All communication between the frontend and backend **must** be encrypted using HTTPS in a production environment.
- **Attack Prevention:**
 - The system **must** incorporate baseline defences against common web vulnerabilities such as SQL Injection (mitigated by using an ORM) and Cross-Site Scripting (XSS) (mitigated by frontend framework features).

2. Usability Requirements

These requirements relate to the ease and efficiency with which a user can interact with the system.

- **User Interface (UI):**
 - The UI **shall** be designed to be intuitive, consistent, and easy to navigate.
 - The system **shall** provide clear feedback for user actions (e.g., "Rule saved successfully," "An error occurred").
 - Error messages **must** be informative, helping the user understand the problem and how to resolve it (e.g., "DSL Syntax Error on Line 5: Invalid keyword 'THEN'").
- **Domain-Specific Language (DSL):**
 - The DSL syntax **shall** be designed to be highly readable and as close to natural language as possible, enabling technical users to express business logic effectively.

3. Maintainability Requirements

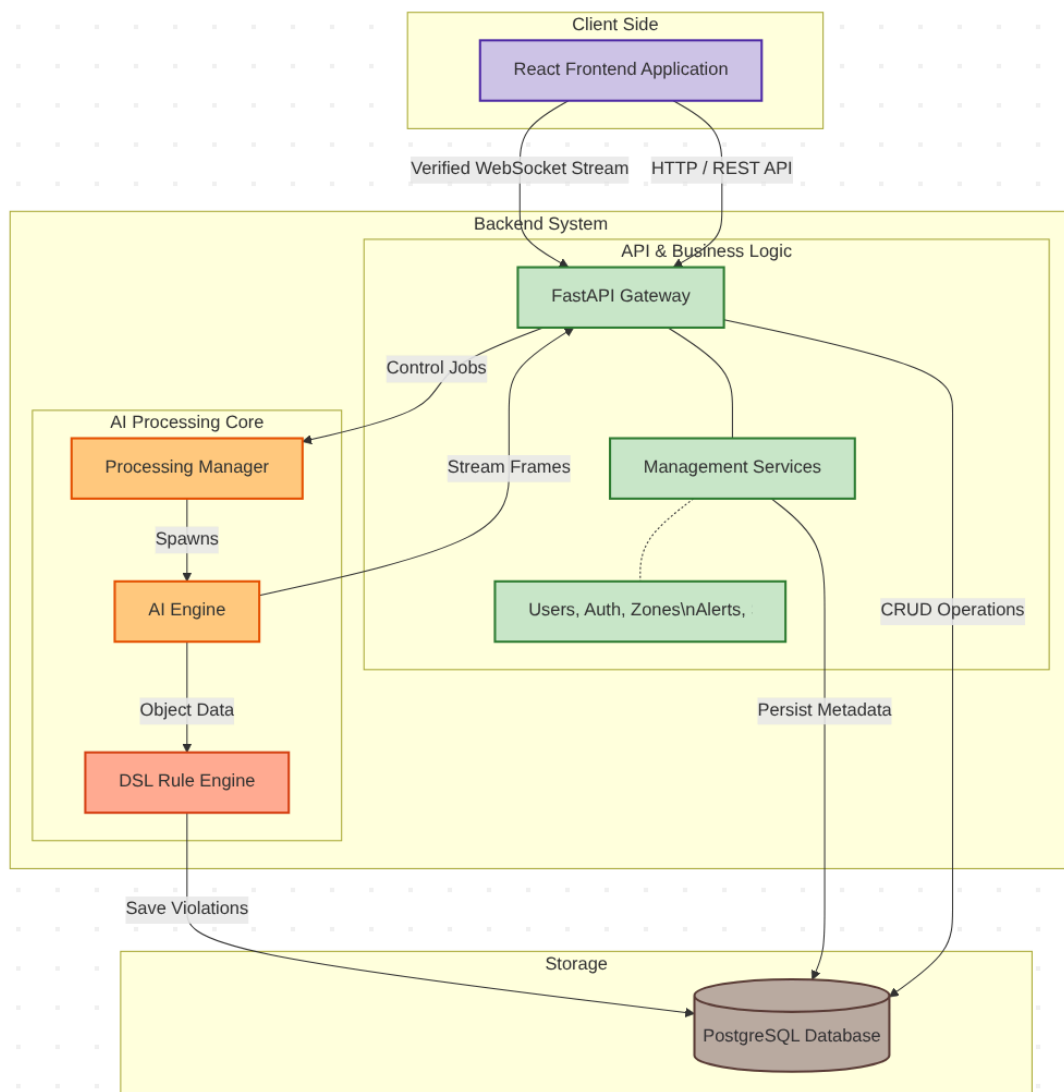
These requirements ensure that the system can be easily modified, updated, and debugged in the future.

- The source code **must** be well-organised and adhere to established coding conventions for each respective language.
- Complex functionalities **must** be documented with explanatory comments or external documentation.
- The use of Docker and Docker Compose **must** ensure that any developer can set up and run the entire system on their local machine with minimal effort.

SYSTEM ARCHITECTURE DESIGN

The Vehicle Fault Detection adopts a modern, modular full-stack architecture with clear separation of concerns between the client-side and backend components. The design follows an API-first approach, where the frontend consumes RESTful APIs and real-time streams from the backend, while the backend handles business logic, AI processing, and data persistence.

The overall architecture is illustrated in the diagram below:

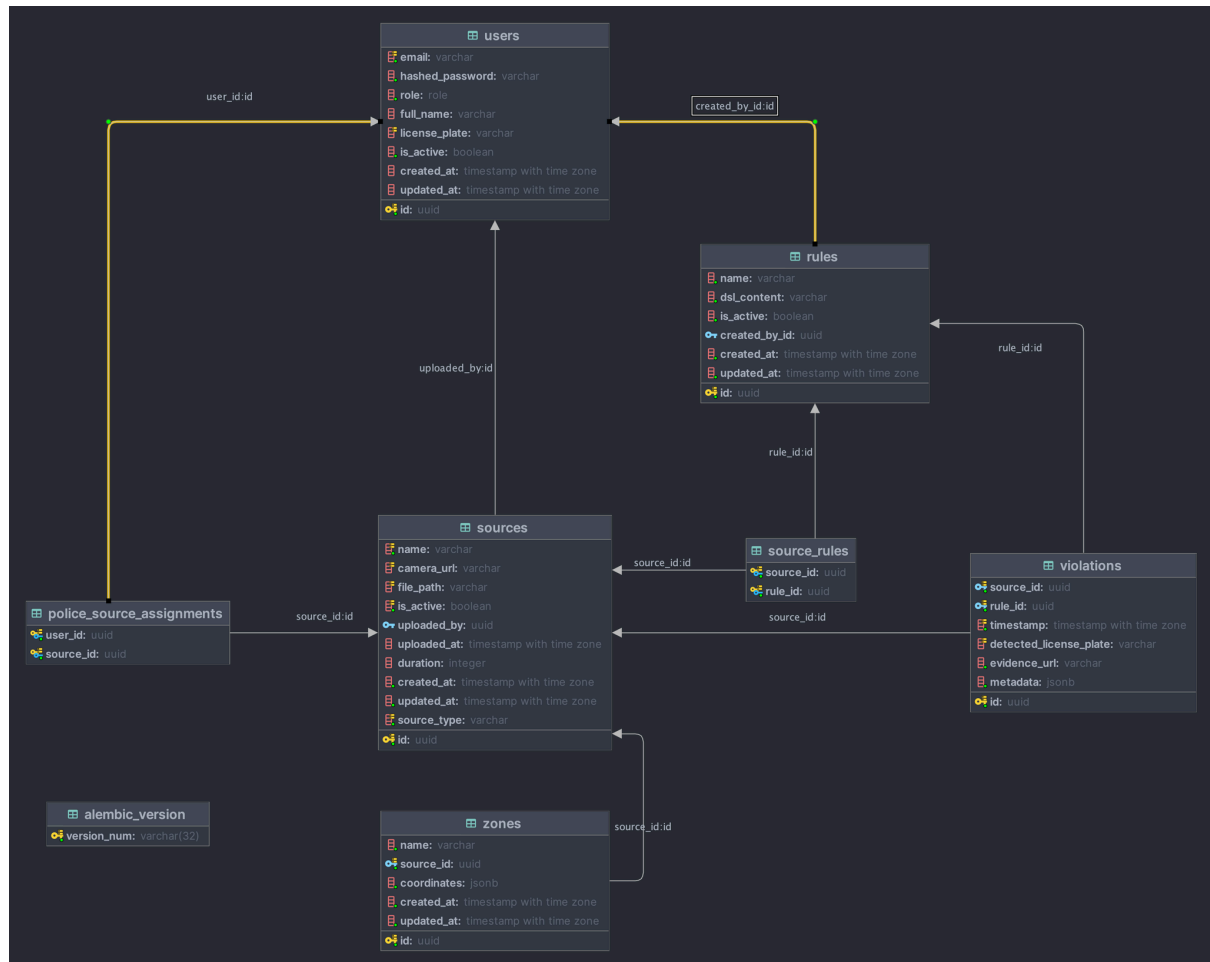


Key Components and Data Flow:

1. **Client Side – React Frontend Application** The user interface is built using React.js, providing a responsive dashboard for real-time monitoring, configuration, and reporting. It communicates with the backend via:
 - **HTTP/REST API:** For authentication, CRUD operations (users, sources, rules, zones, assignments), and fetching violation reports.
 - **Verified WebSocket Stream:** For receiving instant violation alerts pushed from the backend.
2. **Backend System – FastAPI Gateway** FastAPI serves as the central API gateway and orchestrator:
 - Exposes REST endpoints for all business logic (authentication, management services).
 - Manages WebSocket connections for pushing violation alerts.
 - Coordinates background processing jobs through the Processing Manager.
3. **AI Processing Core**
 - **Processing Manager:** Controls the lifecycle of AI workers (start/stop based on API requests).
 - **AI Engine:** Performs real-time video analysis using YOLOv8 for object detection and ByteTrack for tracking. It generates object metadata, applies zone containment checks, and visualises results (bounding boxes, zone polygons, violation highlights) on each frame.
 - **DSL Rule Engine:** Evaluates detected objects against administrator-defined rules using the custom Domain-Specific Language (DSL) to determine violations.
4. **Data Flow**
 - Processed frames with visualisation are streamed back to the frontend.
 - Detected violations are saved to the database and simultaneously pushed to connected clients via WebSocket.
 - Management operations (CRUD on users, sources, zones, rules) persist metadata in the database.
5. **Storage Layer**
 - **PostgreSQL Database:** Stores all persistent data, including user accounts, source configurations, zones (coordinates as JSONB), rules (DSL strings), assignments, and violation records with metadata.
 - **Evidence Storage:** Stores snapshot images of violations for evidentiary purposes.

This architecture ensures high performance, scalability (through background workers), real-time responsiveness (WebSocket + MJPEG streaming), and flexibility (DSL-based rules and configurable zones), making it suitable for both development/testing with video files and production deployment with live camera streams.

DATABASE ARCHITECTURE DESIGN



This database schema is designed for a **Vehicle Fault & Traffic Violation Detection System**. It manages the lifecycle of traffic monitoring, from defining data sources (cameras/files) and detection rules to assigning officers and recording specific violations. The architecture prioritises scalability (via UUIDs), flexibility (via JSONB), and auditability.

1. Core Tables & Schema Definitions

- users Table:

- **Purpose:** Acts as the central authority for system authentication and identity management
- **Key Attributes:**
 - **Id: Primary Key:** Utilising **UUIDv7** (instead of v4) provides time-ordered sortability. This significantly reduces B-Tree index fragmentation and improves insert performance compared to random UUIDs.

- Email: **varchar** (unique, not null) - Serves as the primary login identifier; the unique constraint prevents duplicate account creation.
- hashed_password: **varchar** - Stores only cryptographic hashes Argon2. Plain-text passwords are never persisted.
- Role: **ENUM** (ADMIN, POLICE, USER). Hardcodes authorisation levels directly into the database schema to strictly enforce access boundaries.
- fullname: **varchar** - Stores the display name for dashboard UI and reporting purposes.
- license_plate: **varchar** (nullable) - Suggests the user might be a patrol officer associated with a specific vehicle.
- is_active: **boolean** - Implements **Soft Delete** to allow administrators to revoke access without deleting the row, preserving referential integrity for historical logs.
- create_at, update_at: **timestampz** - auto-managed audit columns to track when the user was onboarded and when their profile was last modified.

- **Sources Table:**

- **Purpose:** Represents the input streams for detection. These can be live feeds or static files.
- **Key Attributes:**
 - id: **Primary Key:** Utilising **UUIDv7**
 - camera_url / file_path: **varchar** - Locators for the media.
 - source_type: **varchar** - Differentiates between inputs (camera, video)
 - uploaded_by: **uuid** (FK) - Links to the users table, tracking who introduced the source.
 - duration
 - is_active: **boolean** - Soft switch to enable/disable monitoring on this source.
 - create_at, update_at: **timestampz** - auto-managed audit columns to track when the sources were onboarded and last modified.

- **Rules Table:**

- **Purpose:** Defines the logic for what constitutes a violation (e.g., "Red Light Crossing", "Speeding > 80km/h").
- **Key Attributes:**
 - id: **Primary Key:** Utilising **UUIDv7**
 - name: **varchar** - A human-readable identifier for the rule
 - dsl_content: **varchar** - Stands for **Domain Specific Language**. This stores the actual logic or code snippet used by the detection engine to process the video feed.
 - is_active: Setting this to **False** instantly disables this rule across *all* assigned sources. Useful for temporarily suspending a specific rule type (e.g., during sensor maintenance or policy changes).

- **created_by_id: uuid (FK)** - Audit trail for who created the rule. Since these rules trigger legal penalties (violations), it is critical to track the specific administrator who defined or altered the logic.
 - **create_at, update_at: timestampz** - auto-managed audit columns to track when the rules were onboarded and last modified.
- **violations Table**
 - **Purpose:** The transactional record of a detected event. This is the "output" of the system.
 - **Key Attributes:**
 - **id: Primary Key:** Utilising **UUIDv7**
 - **source_id:** Identifies the specific camera or file stream where the event occurred, enabling location-based reporting
 - **rule_id:** Categorises the type of infraction (e.g., "Red Light"), linking this specific event back to the legal logic defined in the system.
 - **Timestamp: timestamp with time zone** - Exact time of occurrence.
 - **detected_license_plate: varchar** - The OCR result of the vehicle involved.
 - **evidence_url: varchar** - Link to a snapshot or video clip proving the violation.
 - **Metadata: jsonb** - Stores unstructured data, likely containing confidence scores, vehicle colour, make/model, or bounding box coordinates.
- **zones Tables**
 - **Purpose:** Defines specific Regions of Interest (ROI) within a source's view (e.g., a specific lane or intersection box).
 - **Key Attributes:**
 - **id: Primary Key:** Utilising **UUIDv7**
 - **name: varchar:** A descriptive label for the specific area
 - **source_id: uuid (FK)** - Links the zone to a specific camera/file.
 - **Coordinates: jsonb** - Likely stores an array of (x, y) points defining a polygon overlay on the video feed.
 - **create_at, update_at: timestampz** - auto-managed audit columns to track when the zones were onboarded and last modified.

2. Association & Join Tables (Relationships)

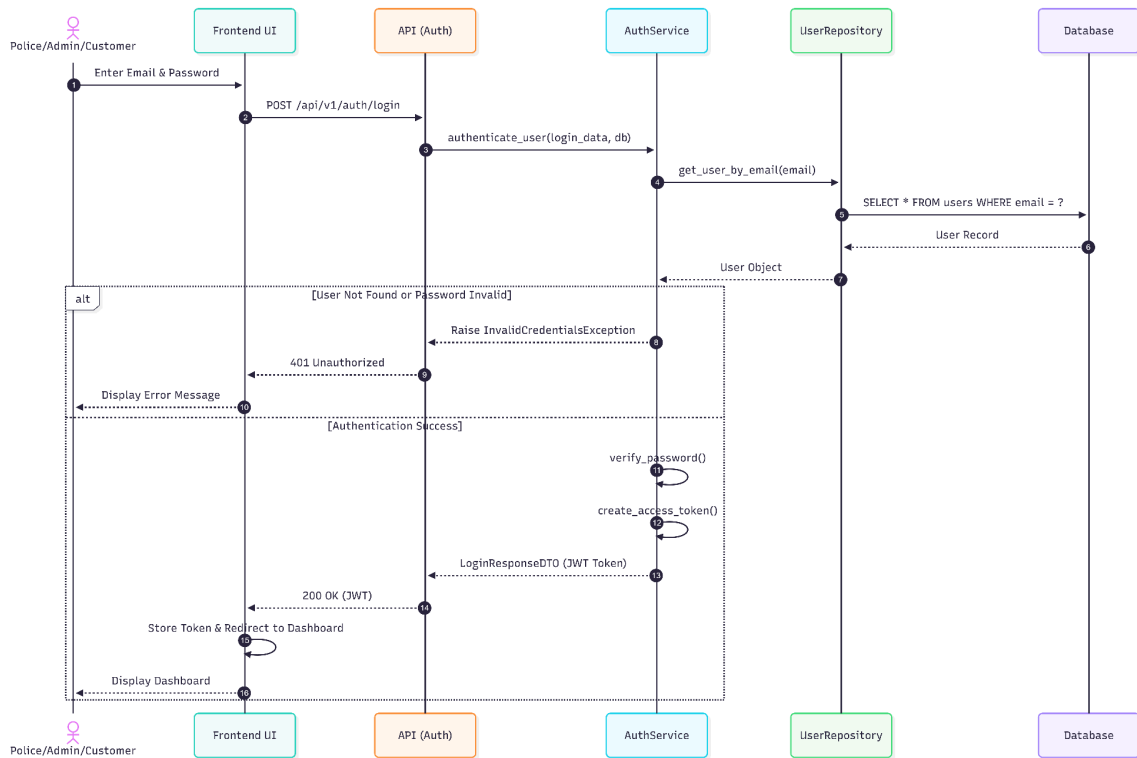
This schema relies heavily on **Many-to-Many (M:N)** relationships to maintain flexibility.

- **source_rules**
 - **Relationship: Sources ↔ Rules**
 - **Logic:** A single camera (source) can check for multiple infractions (e.g., Speeding AND Seatbelt). Conversely, a single rule (e.g., "Red Light") can be applied to thousands of cameras.
 - **Keys:** Composite Primary Key made of **source_id** and **rule_id**.

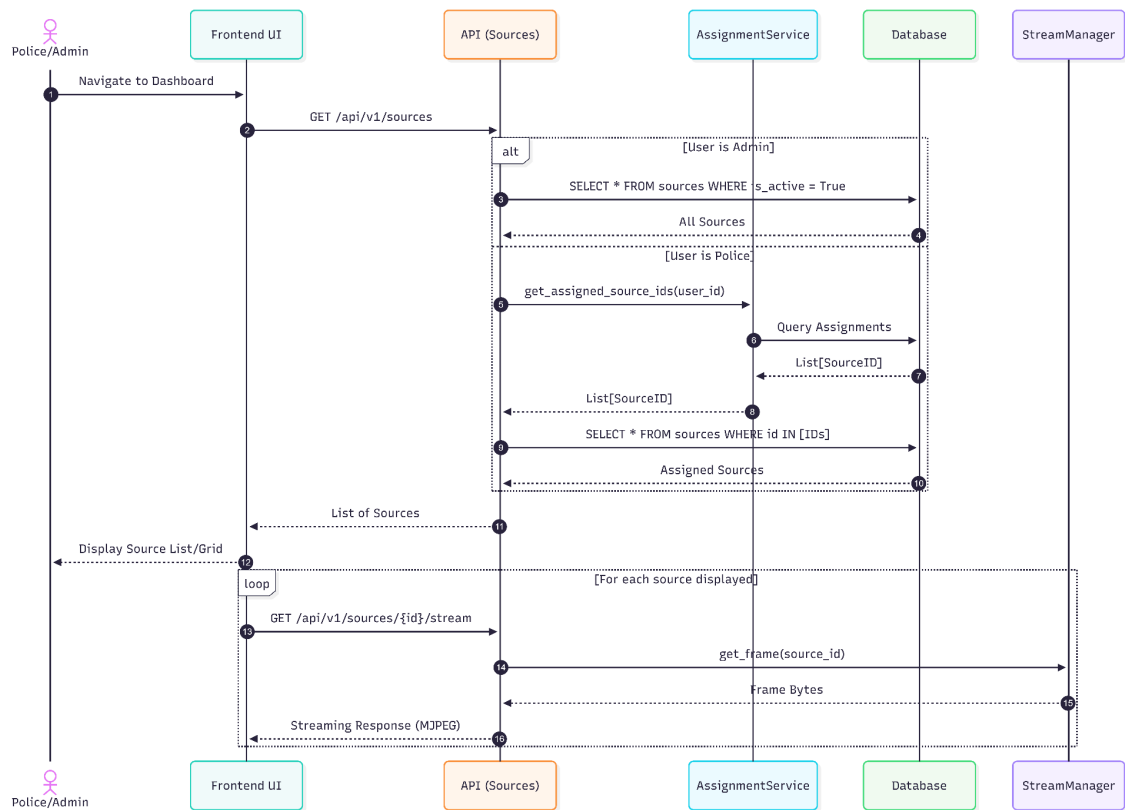
- **police_source_assignments**
 - **Relationship:** Users (Police) ↔ Sources
 - **Logic:** Assigns specific officers to monitor specific feeds. An officer can watch multiple cameras, and a major intersection camera might be monitored by multiple officers.
 - **Keys:** Composite Primary Key made of user_id and source_id.

SEQUENCE DIAGRAMS

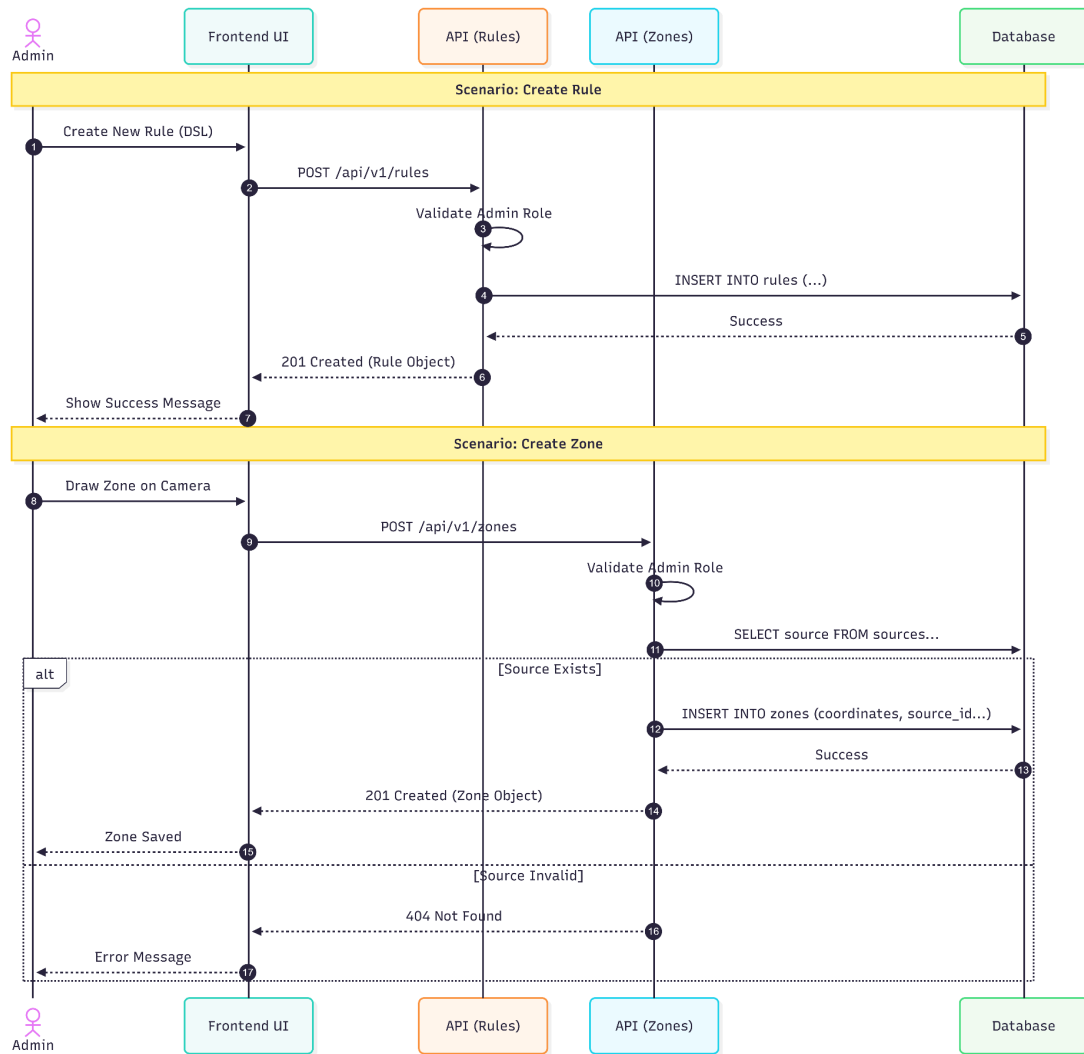
1. Log in to the System



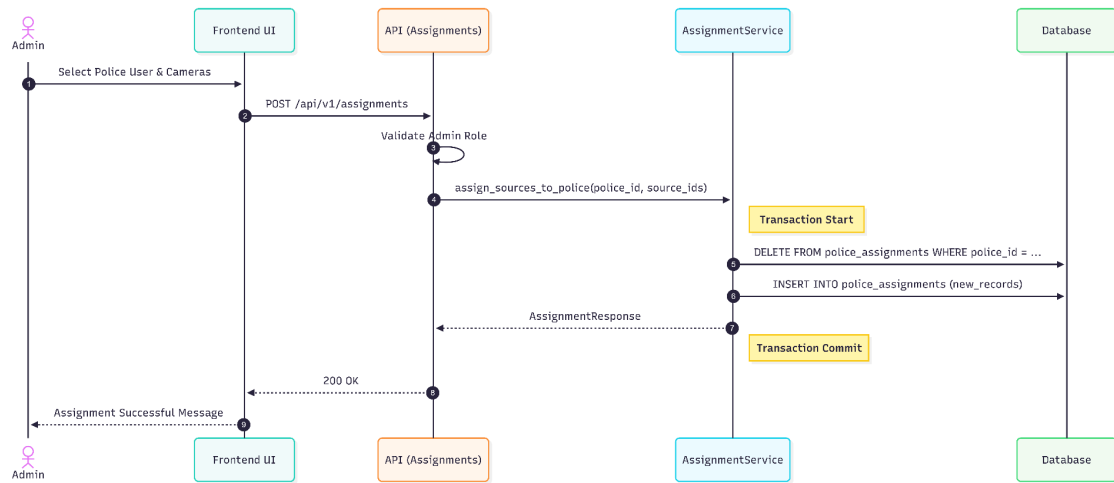
2. View Real-time Monitoring Dashboard



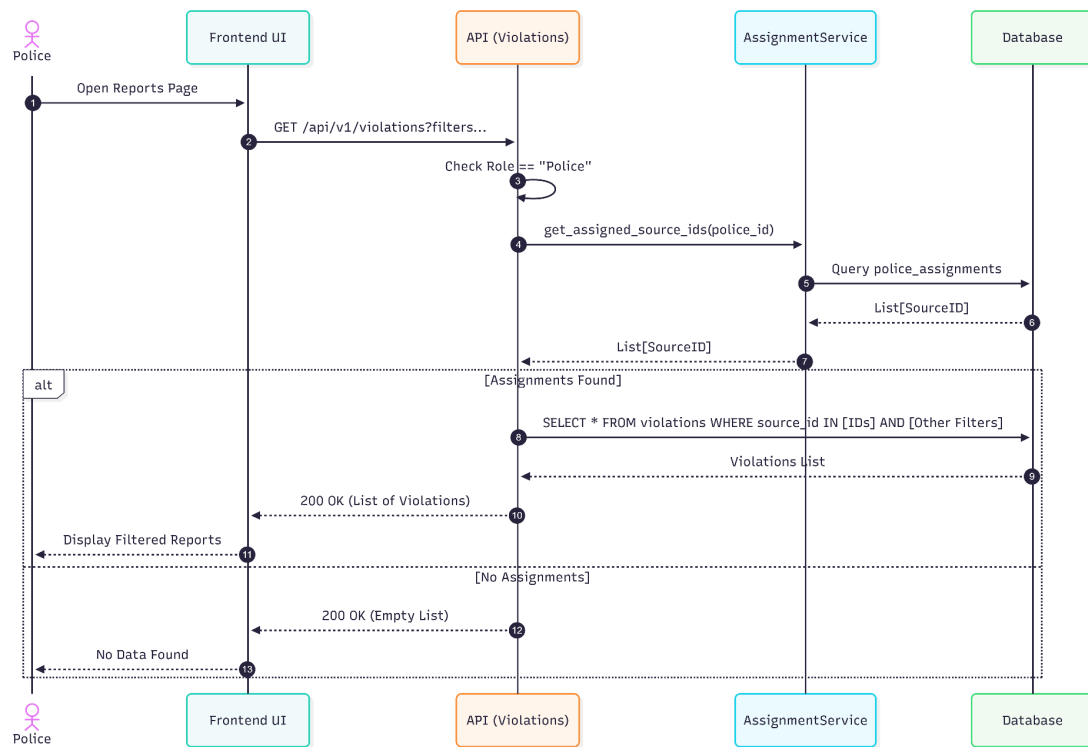
3. Manage System Configuration



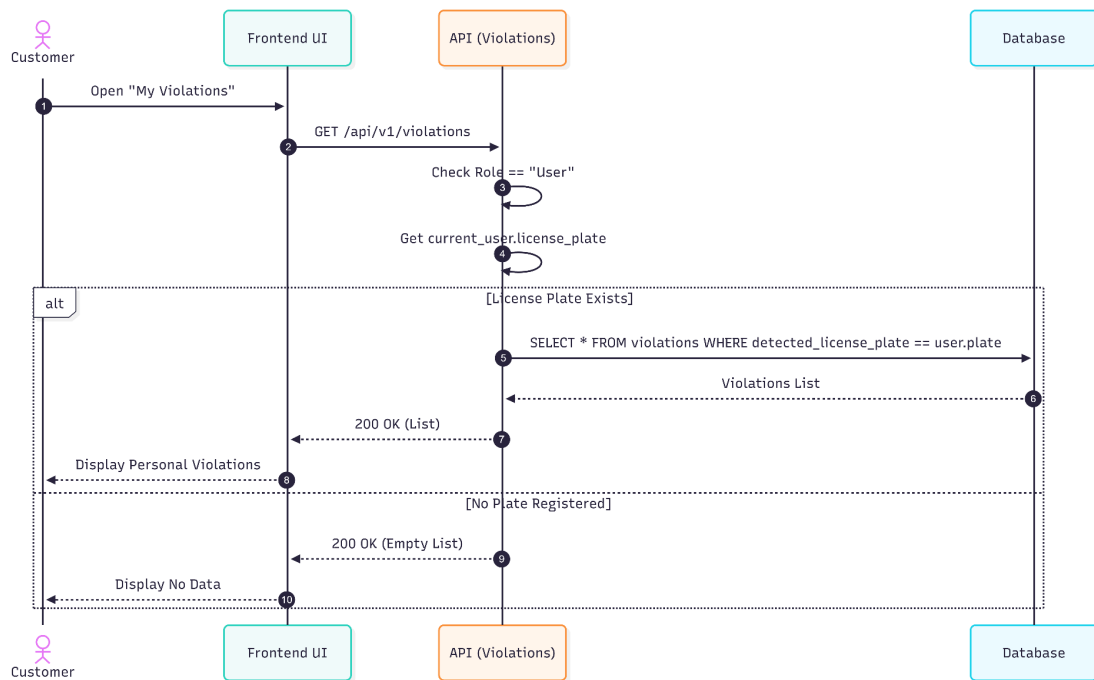
4. Assign a camera to the police



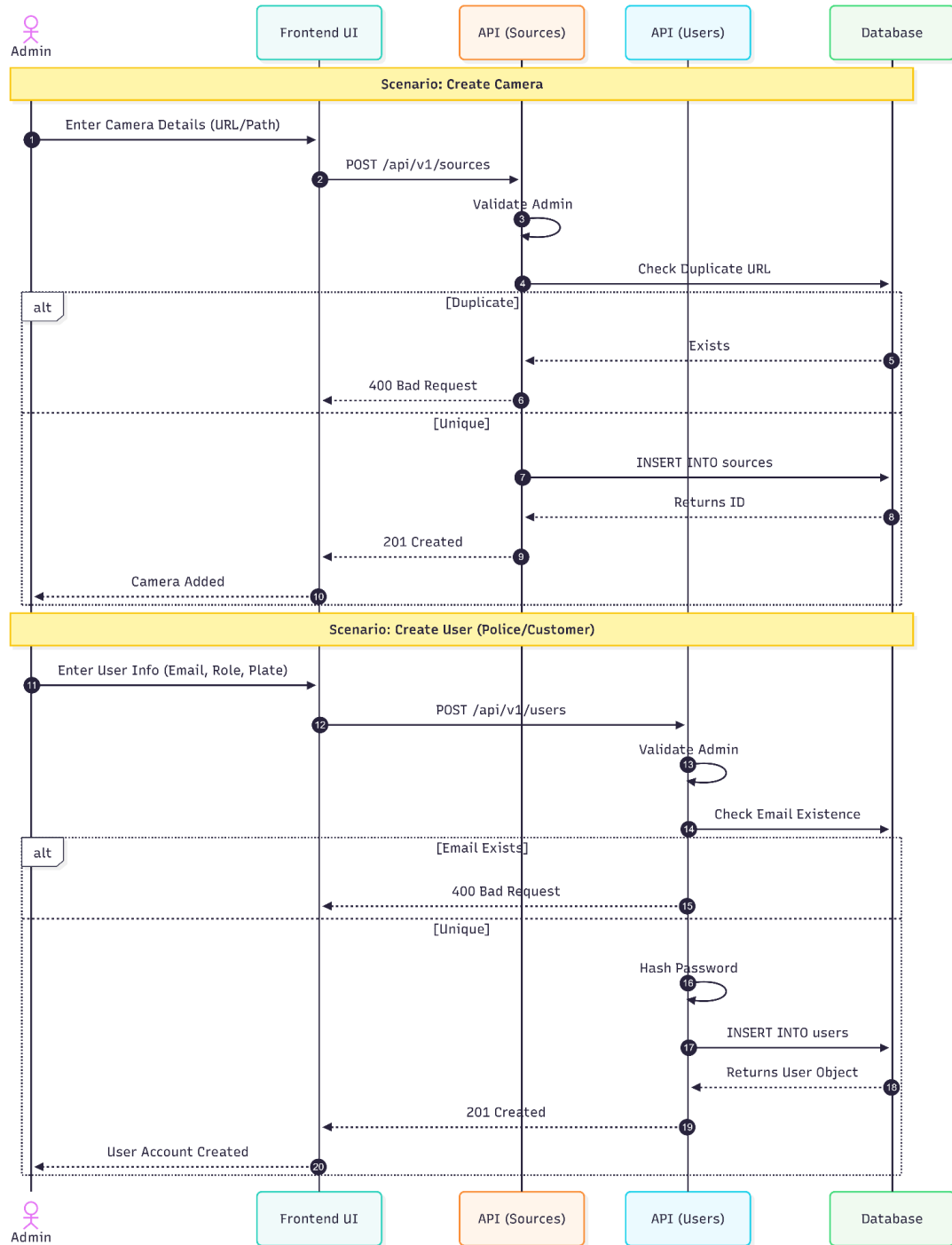
5. View and Filter Violation Reports for Police



6. View and Filter Violation Reports for Customer



7. Manage System Infrastructure



III. Implementation

1. Login

- Input the user credentials (email and password) → Login Button.
- If the credentials are valid, the user can log in to the system.
- If the credentials are not valid, the user will get an error.

CameraLanguage

Home About Us Blog Documentation

LIGHT Login

CameraLanguage

You're one step away being fined.

A beautifully designed interface to pay for your mistake

- Track and pay bills effortlessly
- Simplify your process
- Hassle-free payment
- Insights into your mistake

"No hidden fees or charges — the price you see is the price you pay. (We promise!)"

CarScript

Twitter Facebook LinkedIn

© 2025 MLT Cooperation. All Rights Reserved. Privacy Policy Terms of Service

Login

Enter your details below to login

Email

admin@system.com

Password

Must be at least 8 characters including uppercase, lowercase, number and special character.

Login

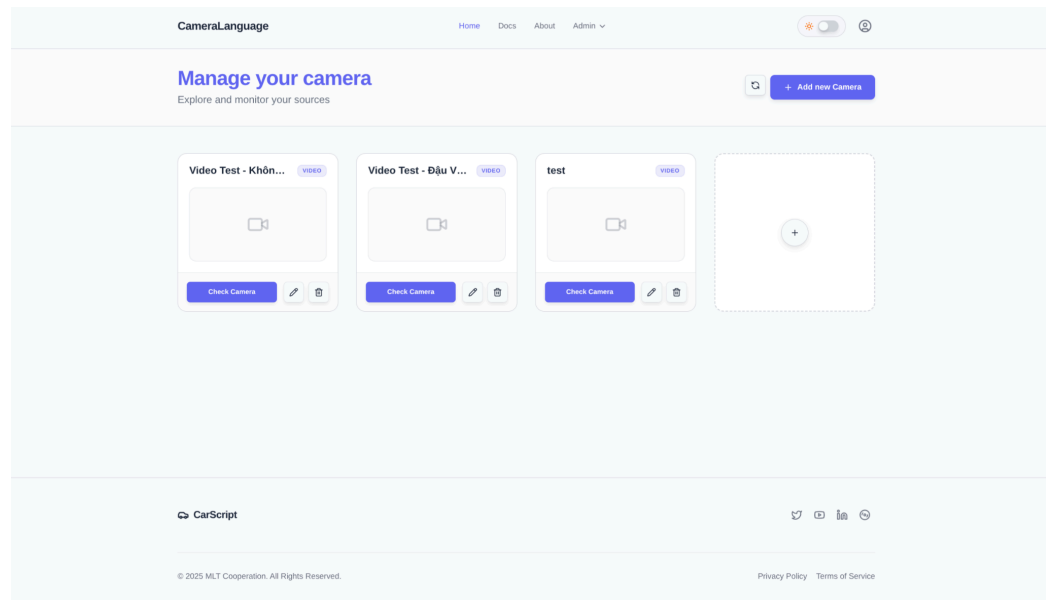
Login with Google

Didn't have an account? [Sign Up](#)

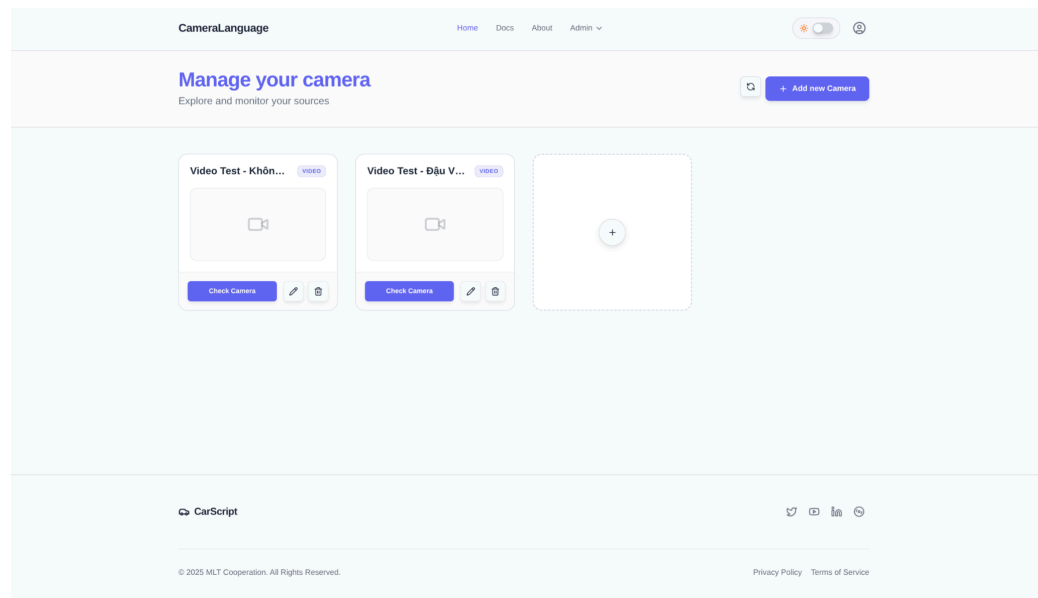
[Forgot your password?](#)

2. View Monitoring Dashboard

- After login, each role will have its own dashboard
 - + Admin with all the sources implemented on the system



- + Police with all the sources that are assigned to them



- + User with their violation if they have

3. Manages system configuration

- We have custom zones and rules to capture the violations in sources.
- After configuring zones and rules, we can use the AI model to detect the violations by clicking the “Bắt đầu AI”, or you can stop at a place.
- After violations are detected, it will store the database.

CameraLanguage

Home Docs About Admin

Video Test - Không Đội Mũ Bảo Hiểm

videos/no_helmet.mp4

Bắt đầu AI

Dừng

Stream \no_helmet_zone

Hướng dẫn Demo

1. Nhấn Bắt đầu AI để chạy phân tích.

2. Thêm Vùng (Zone) ở bên phải (hỗ trợ dán np.array).

3. Thêm Luật (Rule) sử dụng DSL.

4. Các lỗi vi phạm sẽ hiện khung đỏ trên video.

Cấu hình Vùng (Zones)

Tên vùng (ví: Khu vực cấm đỗ)

[[0,0], [100,0], [100,100], [0,100]]

Lưu Vùng

no_helmet_zone

[[129,118],[138,999],[1879,975],[1866,77]]

Luật Kiểm Tra (Rules)

Tên luật (ví: Kiểm tra đồ xe sai quy định)

Nội dung DSL: rule 'MyRule' when object.class_name == 'Car' AND object.current_zone == 'Khu vực cấm'

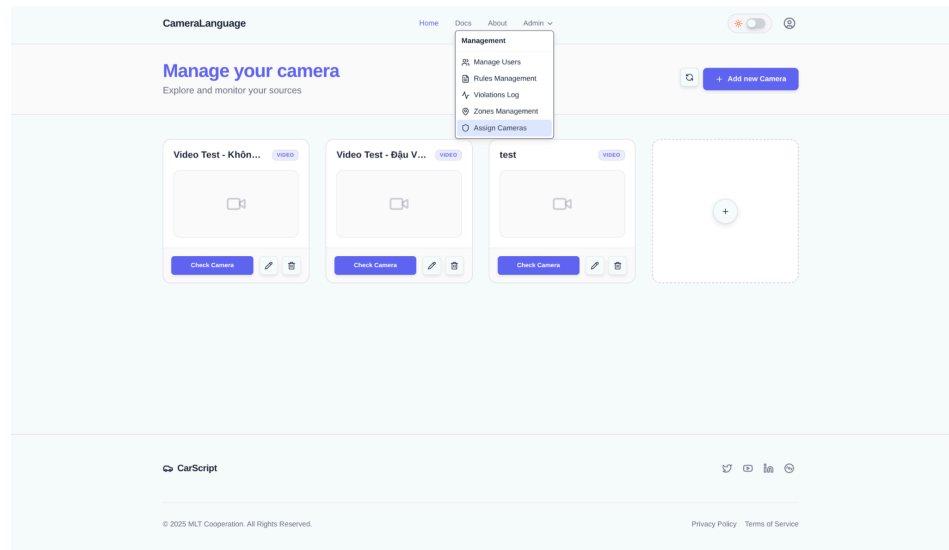
Thêm Luật

Không đội mũ bảo hiểm

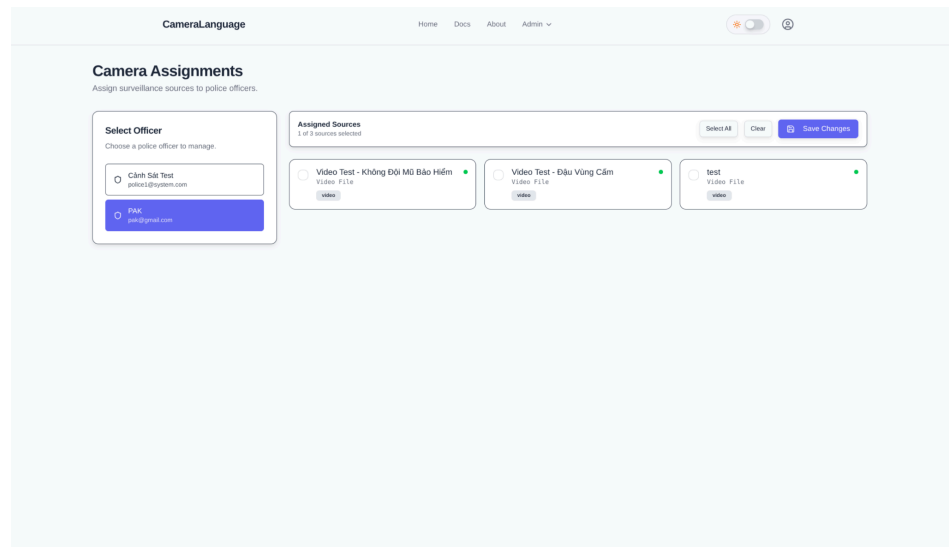
IF object.class_name IN "No Helmet" AND object.current_zone == "motorbike_lane" THEN TRIGGER_VIOLATION

4. Assign sources to the polices

- Press “admin” and press “Assign Camera”

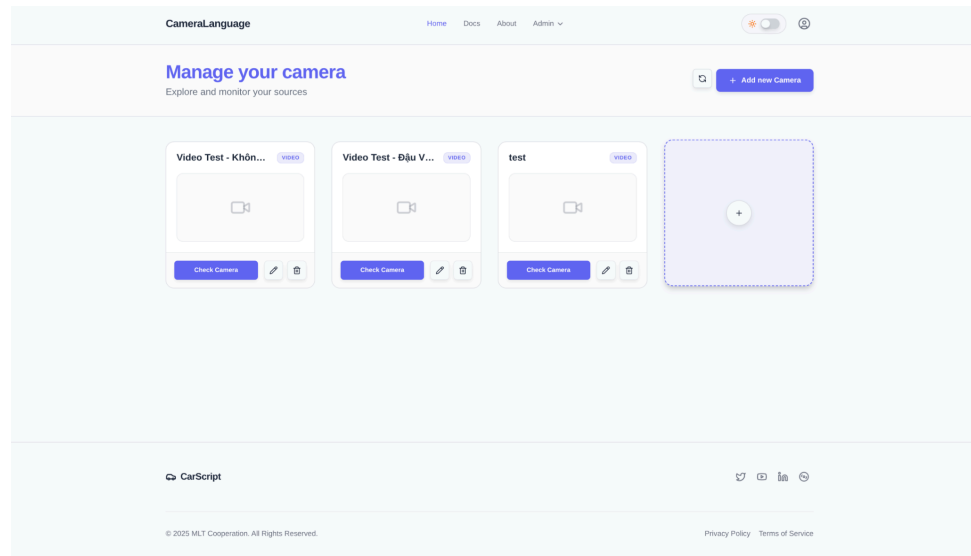


- Then select the officer and a suitable source to assign



5. Create camera

- Press to the “+” button

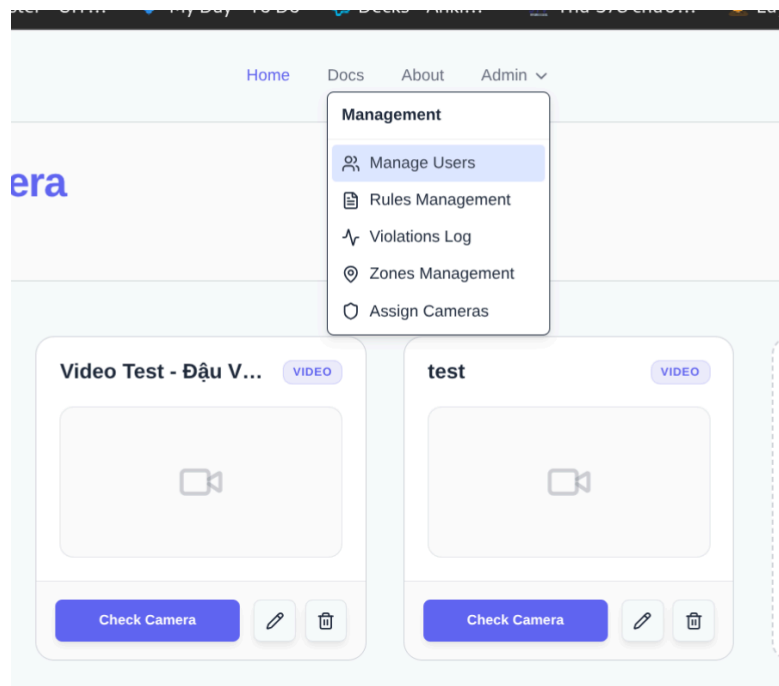


- Type information source

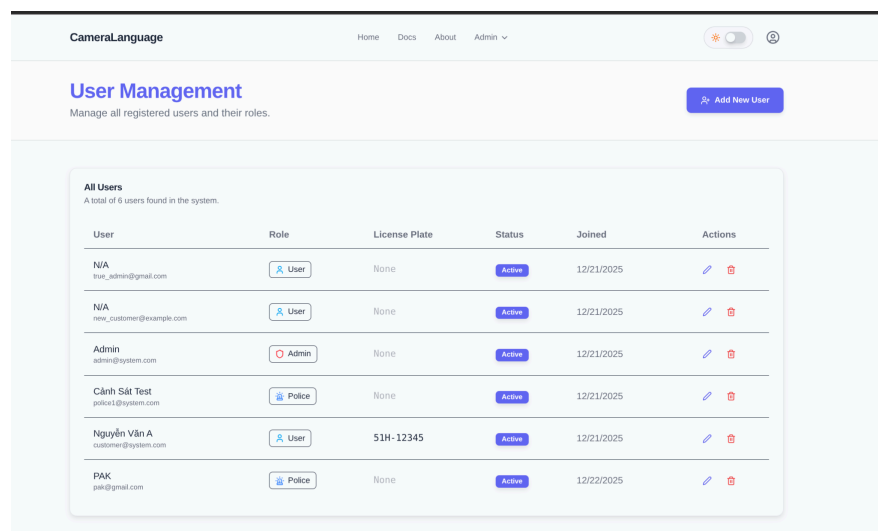
The screenshot shows a modal form titled 'Add New Source' with a close button (X) in the top right corner. The form contains three input fields: 'Source Name' with the placeholder text 'e.g. Front Gate Camera', 'Source Type' with a dropdown menu currently showing 'Camera (Live Stream)', and 'Camera URL' with the placeholder text 'rtsp://... or http://...'. At the bottom of the form are two buttons: 'Cancel' and 'Create Source'.

6. Create a new user

- Press “admin” and press “manage user”



- Press “add new user”



- Type information and press “create user”

Add New User

Enter details to create a new system user.

Full Name

John Doe

OPTIONAL

Email Address

john@example.com

Role

User

License Plate

ABC-1234

OPTIONAL

Password

Confirm Password

Cancel

Create User

IV. Discussion and Conclusion

The seven use cases presented in this project highlight that the Vehicle Fault Detection system serves as more than just an error identification tool; it represents a comprehensive workflow that efficiently connects administrators, law enforcement officers, and drivers. The system is designed with a strong emphasis on security and organised protocols. By implementing secure login procedures and role-based access controls, the platform ensures that sensitive surveillance data and personal records are accessible solely to authorised personnel. This clear delineation of responsibilities—where Administrators oversee the technical framework while Law Enforcement Officers concentrate on enforcement—creates a secure environment, allowing each user to access only the information necessary for their role, thereby mitigating the risk of boundary violations and preserving data privacy.

In addition to security, the system offers substantial operational flexibility through its administrative tools. Rather than being constrained by hard-coded traffic laws that are challenging to modify, the system enables Administrators to customise detection "Rules" and specific "Zones" within the camera feeds manually. This capability ensures that as speed limits or road configurations change, the system can be swiftly adapted to reflect these changes within minutes, without the need for complex software updates. Such adaptability is complemented by a robust camera management feature, facilitating the infrastructure's scalability as urban monitoring needs evolve.

For law enforcement, this system revolutionises the conventional workflow from passive patrolling to proactive digital monitoring. The ability to assign designated cameras to specific officers prevents information overload, allowing personnel to focus exclusively on their assigned districts. In instances of violations, the availability of digital evidence coupled with a transparent review history enhances accountability. This dual-view approach—allowing officers to validate fines with video evidence while granting drivers (when permitted) access to their violation history—diminishes the potential for disputes and ensures that the enforcement process is perceived as both fair and objective.

In summary, the Vehicle Fault Detection system effectively automates the intricate task of traffic monitoring. By translating real-world traffic laws into digital regulations and offering real-time video analysis, the project replaces slow, manual processes with a rapid, accurate AI-driven solution. The design demonstrates that it is feasible to establish a system that serves as a powerful asset for authorities—providing live data and management tools—while also being equitable for the public—delivering clear evidence for each recorded violation. Ultimately, this initiative lays a strong foundation for a modern, smart city infrastructure where traffic management is proactive, transparent, and highly efficient.