ITE 18 – Application Development and Emerging Technologies

Web Application – Final Project Submission

Student Name(s): Christian Angelo M. Gonzaga
Project Title: TraMo (Track Your Motor)
Date of Submission: December 22, 2025
Course/Section: ITE 18 – EJ1

1. PROJECT OVERVIEW

1.1 Project Description

TraMo (Track Your Motor) is a comprehensive, web-based transportation monitoring and vehicle management system designed to digitally manage, organize, and monitor motor vehicle information in a structured, secure, and centralized environment. The system was developed to address common challenges faced by organizations and institutions that rely on traditional manual methods such as paper records, logbooks, or spreadsheets to manage transportation data.

Manual record-keeping systems often result in data redundancy, inaccuracies, loss of records, limited accessibility, and inefficient monitoring processes. TraMo resolves these issues by providing a centralized database-driven solution that ensures data consistency, security, and ease of access. The system allows authorized users to store vehicle details, record monitoring activities, and retrieve historical transportation data efficiently.

By utilizing a modern web architecture composed of a Laravel backend, React-based frontend, and a MySQL relational database, TraMo delivers a scalable, maintainable, and efficient solution. The application improves operational efficiency, enhances accountability, and supports informed decision-making through organized and accessible vehicle monitoring records.

1.2 Target Users

Intended Users of TraMo

The intended users of TraMo include individuals and organizations involved in transportation and vehicle management processes, specifically:

- **Transportation Administrators** – responsible for overseeing vehicle inventories, monitoring usage, and maintaining accurate records.
- **System Administrators** – responsible for managing user accounts, roles, and system access permissions.

- **Authorized Staff or Personnel** – tasked with recording vehicle monitoring data, reviewing vehicle status, and maintaining transportation logs.
- **Vehicle Owners / Clients** – able to view information related to their registered vehicles, including owner, registration details, fines, insurance, and monitoring history.

These users require a system that is reliable, secure, intuitive, and capable of handling structured data efficiently with minimal training. TraMo is designed to reduce administrative workload while improving data accuracy, traceability, and overall system usability.

1.3 Key Features

- Secure user authentication and session management
- Role-based authorization and controlled access to system features
- Vehicle management module with full Create, Read, Update, and Delete (CRUD) functionality
- Transportation monitoring (TraMo module) for recording vehicle activity and status updates
- Centralized MySQL database for structured and persistent data storage
- RESTful API-based architecture for clean frontend-backend separation
- Responsive and user-friendly web interface accessible across devices

1.4 Technology Stack

Frontend:

- HTML5 – page structure and semantic markup
- CSS3 – layout, styling, and responsive design
- JavaScript – client-side interactivity
- React – component-based user interface development

Backend:

- Laravel (PHP Framework) – server-side logic, routing, authentication, and security
- RESTful API architecture – standardized communication between frontend and backend

Database:

- MySQL – relational database for persistent data storage

Other Tools:

- Git – version control and source code management
- Postman – API testing and debugging
- Visual Studio Code – development environment
- XAMPP – local Apache and MySQL server setup

2. APP PLAN

2.1 Project Scope

In Scope

- User authentication, session handling, and access control
- Vehicle information management (create, update, view, delete)
- Transportation monitoring records linked to vehicles
- Secure API-driven communication between frontend and backend
- Normalized relational database design using MySQL

Out of Scope

- Real-time GPS or map-based vehicle tracking
- Native Android or iOS mobile applications
- Online payment, billing, or subscription system

2.2 Objectives & Goals

1. To design and develop a fully functional web-based transportation monitoring and vehicle management system
2. To implement secure authentication and role-based authorization mechanisms
3. To ensure data integrity, accuracy, and scalability through proper database normalization
4. To provide an intuitive user interface that minimizes training requirements and improves usability

2.3 User Stories & Use Cases

User Story 1

As an administrator, I want to securely log in to the system so that sensitive vehicle and monitoring data is protected from unauthorized access.

Acceptance Criteria:

- The system allows access only with valid credentials
- Invalid login attempts display appropriate error messages
- User sessions are terminated upon logout

User Story 2

As an authorized user, I want to add and manage vehicle records so that transportation data remains accurate and up to date.

Acceptance Criteria:

- New vehicle records can be created
- Existing vehicle records can be edited
- Vehicle records can be deleted with confirmation

User Story 3

As a user, I want to record monitoring information for vehicles so that historical tracking data is maintained.

Acceptance Criteria:

- Monitoring records can be linked to specific vehicles
- Monitoring history is viewable per vehicle

2.4 System Architecture

TraMo follows a **client-server architecture**. The frontend is built using React and runs in the user's web browser. It communicates with the backend through RESTful API endpoints using HTTP requests. The Laravel backend handles authentication, authorization, validation, business logic, and database operations. All persistent data is stored in a MySQL relational database.

This separation of concerns improves system maintainability, scalability, and security while enabling independent frontend and backend development.

2.5 Development Timeline

Phase	Description	Timeline
Phase 1	Requirements Analysis & Planning	Week 1–2
Phase 2	Database Design & Setup	Week 2–3
Phase 3	Frontend Development	Week 3–5
Phase 4	Backend API Development	Week 4–6
Phase 5	System Integration & Testing	Week 6–7

3. UI/UX DESIGN

3.1 Design Philosophy

The UI/UX design of TraMo follows **user-centered design principles** with an emphasis on simplicity, clarity, and consistency. The interface is structured to reduce cognitive load, present information clearly, and enable users to complete tasks efficiently. Consistent layouts, predictable navigation patterns, and clear visual hierarchy are used throughout the system.

3.2 Color Scheme

- **Primary Color:** Blue (#1E88E5) – navigation and primary actions
- **Secondary Color:** Gray (#546E7A) – secondary elements and icons
- **Accent Color:** Green (#43A047) – success indicators and confirmations
- **Background Color:** Light Gray (#F5F5F5) – improved readability
- **Text Color:** Dark Gray (#212121) – optimal contrast and legibility

3.3 Typography

- **Font Family:** Arial / Sans-serif
- **Heading Size:** 24–32px
- **Body Text Size:** 14–16px
- **Line Height:** 1.5

3.4 UI Components

- Buttons – consistent styling for actions
- Input Fields – validated inputs with inline error messages
- Navigation Bar – centralized navigation menu
- Cards – grouped vehicle and monitoring information
- Modal Dialogs – confirmations and alerts

3.5 Wireframes & Mockups

Link to Wireframe:
https://miro.com/welcomeonboard/QnRNM0lDWko0ZzZkbkQ3eHhGRkdFeGRub1Q1ZktoWElZb TZzbFhHV0JhOXVKQitOR0NlN0NrenlFeXhQbW83Yk9aWFFpQkZBZUxCUUJEbXprblhNME4 yR2padTRGVzdqKzVYSkNTWXpjM2NzT2JobHB5YVNaZTFWaWxxYS1h0bE5BS2NFMDFkcUN FSnM0d3FEN050ekl3PT0hdjE=?share_link_id=418258415733

This section outlines the wireframes and high-fidelity mockups developed for the **TraMo (Traffic Monitoring & Management System)**. The wireframes illustrate the full user journey, including public access, vehicle search, detailed record viewing, and administrative management screens. The flow demonstrates how users and administrators interact with the system from entry to data management.

Screen 1: Landing Page / Welcome to TraMo

Wireframe/Mockup: *(Top-center – "Welcome to TraMo" screen)*

Description:
This is the system's entry point for all users. It introduces the TraMo platform and provides access to core services. Users can navigate to vehicle search, fines lookup, or administrator login from this page.

Key Elements:

- System branding and welcome message
- Primary navigation buttons
- "Our Services" section highlighting key features
- Clean, user-friendly layout for first-time users

Screen 2: Vehicle Search Dashboard

Wireframe/Mockup: *(Top-center-right – "Vehicle Search Dashboard")*

Description:
This screen allows users to search for vehicle records using a registration number. It acts as the main lookup interface for public users and redirects them to detailed results once a valid record is found.

Key Elements:

- Vehicle registration input field
- Search button
- Status indicators for search results
- Navigation back to the home screen

Screen 3: Authentication Screens (Login & Validation States)

Wireframe/Mockup: *(Top-right screens with login forms and error/success states)*

Description:
These screens handle secure access for administrators. They include login input fields and validation feedback such as incorrect credentials or successful authentication.

Key Elements:

- Username and password fields
- Login button
- Error and success feedback messages
- Secure access flow to admin features

Screen 4: Vehicle Details Overview

Wireframe/Mockup: *(Middle-left – "Vehicle Details" screen)*

Description:
This screen displays comprehensive vehicle information after a successful search. It consolidates vehicle data, owner information, insurance details, and violation summaries in a structured layout.

Key Elements:

- Vehicle specifications (VIN, make, model, year)
- Owner details section
- Insurance and registration status
- Summary cards for fines and violations

Screen 5: Fines and Violations Information (Scrollable Page)

Wireframe/Mockup: *(Bottom-left – "Fines and Violations Information" screen)*

Description:
This screen presents a scrollable list of all fines and violations associated with a vehicle. Each entry includes clear categorization, descriptions, and status indicators to enhance readability.

Key Elements:

- Scrollable single-page layout
- Violation categories and descriptions
- Status indicators (paid/unpaid)
- Due dates and penalty information

Screen 6: Admin Dashboard

Wireframe/Mockup: *(Center – "Admin Dashboard" screen)*

Description:
This screen serves as the control center for administrators. It provides quick access to vehicle management, insurance records, owner details, and fine administration functions.

Key Elements:

- Administrative navigation panels
- Overview of system modules
- Quick-access management buttons
- Clear separation of admin-only features

Screen 7: Vehicle & Owner Management (Admin)

Wireframe/Mockup: *(Right-side panels with editable forms)*

Description:
These screens allow administrators to view and update vehicle and owner records. Editable forms ensure accurate data maintenance and record integrity.

Key Elements:

- Editable vehicle detail fields
- Owner personal and address information
- Update and save buttons
- Form validation indicators

Screen 8: Insurance Policy Management

Wireframe/Mockup: *(Mid-right – insurance form screens)*

Description:
This screen enables administrators to add, update, and manage insurance policies linked to vehicles. It supports compliance tracking and insurance verification.

Key Elements:

- Policy number and serial number fields
- Insurance provider details
- Coverage description
- Policy start and end dates

Screen 9: Fines and Violations Management (Admin)

Wireframe/Mockup: *(Lower-right – fine creation and management screens)*

Description:
This screen allows administrators to issue new fines or violations. It supports structured entry of violation reasons, amounts, and deadlines.

Key Elements:

- Violation reason input
- Fine amount field
- Issue and due date selectors
- Add and manage fine actions

Screen 10: System Feedback & Confirmation States

Wireframe/Mockup: *(Multiple linked screens showing success/error flows)*

Description:
These screens provide system feedback during user and admin actions, such as successful updates, errors, or confirmation of submitted data. They enhance usability and system transparency.

Key Elements:

- Success and error messages

- Confirmation prompts
- Navigation continuation options

3.6 User Flows

This section describes the complete step-by-step user flows for both **Administrators** and **Public Users** within the TraMo (Traffic Monitoring & Management System). Each flow outlines the actions taken by the user and the corresponding system behavior.

A. Administrator User Flow

Admin Authentication & System Access

1. Admin navigates to the login screen.
2. Admin enters username and password.
3. System validates credentials against the database.
4. If credentials are valid, the admin is redirected to the **Admin Dashboard**.
5. If credentials are invalid, an error message is displayed, and the admin is prompted to retry.

Admin Vehicle Search Flow

1. Admin selects **Search Vehicle** from the dashboard.
2. Admin enters a vehicle registration number.
3. System retrieves the vehicle record from the database.
4. Vehicle details, owner details, insurance records, and violation summaries are displayed.

Admin Vehicle & Owner Management Flow

1. Admin views vehicle and owner information.
2. Admin chooses to **Add**, **Edit**, or **Delete** vehicle details.
3. Admin updates vehicle data (VIN, model, year, status, etc.).
4. Admin updates owner data (name, contact details, address).
5. System validates the entered data.
6. System saves changes to the database.
7. A confirmation message is displayed, and the UI refreshes.

Admin Insurance Management Flow

1. Admin navigates to the **Insurance Policies** section.
2. Admin adds or edits insurance policy details.
3. System validates policy information.
4. Insurance data is stored in the database.
5. Updated insurance information is displayed in the vehicle record.

Admin Fines & Violations Management Flow

1. Admin navigates to **Fines and Violations**.
2. Admin adds a new fine or edits an existing violation.
3. Admin enters violation reason, amount, issued date, and due date.
4. System validates the inputs.
5. Fine information is saved to the database.
6. The fines list updates automatically with confirmation feedback.

B. Public User Flow

User Vehicle Search Flow

1. User accesses the TraMo landing page.
2. User selects **Search Vehicle**.
3. User enters a vehicle registration number.
4. System validates the input and searches the database.
5. If the record exists, vehicle details are displayed.

User Vehicle Information Viewing Flow

1. User views vehicle details (make, model, registration status).
2. User scrolls through **Fines and Violations Information**.
3. User sees fine amounts, violation reasons, and due dates.
4. Paid and unpaid fines are clearly indicated.

User Feedback & Error Handling Flow

1. If an invalid vehicle number is entered, the system displays an error message.
2. User is prompted to re-enter the vehicle number.
3. No editing or administrative actions are available to the user.

C. Key Experience Differences (Admin vs User)

Feature	Admin	Public User
Login Required	Yes	No
Search Vehicle	Yes	Yes
View Vehicle Details	Yes	Yes
Update Vehicle Information	Yes	No
Manage Insurance Records	Yes	No
Add/Edit Fines & Violations	Yes	No
View Fines & Violations	Yes	Yes

4. Database Architecture (ERD)

4.1 Entity Relationship Diagram

The ERD contains the following entities:

- address
- owners
- vehicles
- registrations
- fines
- insurance
- admin_roles
- admins
- admin_actions

Each entity includes all attributes, primary keys, foreign keys, and relationships.

4.2 Entity Descriptions

Entity 1: address

Field	Data Type	Constraints	Description
address_id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique address identifier
street	VARCHAR(50)	NOT NULL	Street name or number
city	VARCHAR(50)	NOT NULL	City name
province	VARCHAR(50)	NOT NULL	Province name
postal_code	VARCHAR(50)	NOT NULL	Postal or ZIP code
created_by	VARCHAR(100)	NULL	User who created the record
updated_by	VARCHAR(100)	NULL	User who last updated the record
deleted_by	VARCHAR(100)	NULL	User who deleted the record
timestamps	DATETIME+DATETIME	NOT NULL	Laravel automatic timestamp generation
deleted_at	DATETIME	NULL	Soft delete marker

Entity 2: owners

Field	Data Type	Constraints	Description
owner_id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique owner identifier
FName	VARCHAR(100)	NOT NULL	Owner's first name
LName	VARCHAR(100)	NOT NULL	Owner's last name
address_id	BIGINT	FOREIGN KEY (address_id)	Links to address table
PhoneNumber	VARCHAR(20)	NOT NULL	Owner phone number
LicenseNumber	BIGINT	NOT NULL	Government license ID
created_by	VARCHAR(100)	NULL	User who created the record
updated_by	VARCHAR(100)	NULL	User who last updated the record
deleted_by	VARCHAR(100)	NULL	User who deleted the record
timestamps	DATETIME+DATETIME	NOT NULL	Laravel automatic timestamp
deleted_at	DATETIME	NULL	Soft delete marker

Entity 3: vehicles

Field	Data Type	Constraints	Description
vehicle_id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique vehicle identifier
owner_id	BIGINT	FOREIGN KEY → owners	Points to vehicle owner
vin	VARCHAR(50)	NOT NULL	Vehicle Identification Number
make	VARCHAR(50)	NOT NULL	Vehicle manufacturer
model	VARCHAR(50)	NOT NULL	Vehicle model
color	VARCHAR(50)	NOT NULL	Vehicle color
year	INT	NOT NULL	Manufacturing year
vehicle_status	VARCHAR(100)	NOT NULL	Active, inactive, etc.
created_by	VARCHAR(100)	NULL	User who created the record

Field	Data Type	Constraints	Description
updated_by	VARCHAR(100)	NULL	User who last updated the record
deleted_by	VARCHAR(100)	NULL	User who deleted the record
timestamps	DATETIME+DATETIME	NOT NULL	Laravel timestamp feature
deleted_at	DATETIME	NULL	Soft delete marker

Entity 4: registrations

Field	Data Type	Constraints	Description
registration_id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique registration ID
vehicle_id	BIGINT	FOREIGN KEY → vehicles	References vehicle table
plate_number	VARCHAR(20)	NOT NULL	License plate number
registration_date	DATE	NOT NULL	Date of registration
expiration_date	DATE	NOT NULL	Expiry of registration
status	VARCHAR(50)	NOT NULL	Active / expired / pending
created_by	VARCHAR(100)	NULL	User who created the record
updated_by	VARCHAR(100)	NULL	User who last updated the record
deleted_by	VARCHAR(100)	NULL	User who deleted the record
timestamps	DATETIME+DATETIME	NOT NULL	Laravel timestamps
deleted_at	DATETIME	NULL	Soft delete marker

Entity 5: fines

Field	Data Type	Constraints	Description
fine_id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique fine identifier
vehicle_id	BIGINT	FOREIGN KEY → vehicles	Vehicle penalized

Field	Data Type	Constraints	Description
issued_date	DATE	NOT NULL	Date penalty issued
due_date	DATE	NOT NULL	Payment deadline
amount	DECIMAL(10,2)	NOT NULL	Fine amount
reason	VARCHAR(100)	NOT NULL	Violation type
created_by	VARCHAR(100)	NULL	Record creator
updated_by	VARCHAR(100)	NULL	Last record updater
deleted_by	VARCHAR(100)	NULL	Deleted record user
timestamps	DATETIME+DATETIME	NOT NULL	Laravel timestamps
deleted_at	DATETIME	NULL	Soft delete marker

Entity 6: insurance

Field	Data Type	Constraints	Description
insurance_id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique insurance record
vehicle_id	BIGINT	FOREIGN KEY → vehicles	Covered vehicle
provider	VARCHAR(50)	NOT NULL	Insurance company
policy_number	VARCHAR(50)	NOT NULL	Policy identifier
coverage	TEXT	NOT NULL	Coverage description
start_date	DATE	NOT NULL	Insurance start date
end_date	DATE	NOT NULL	Insurance expiration
created_by	VARCHAR(100)	NULL	User who created the record
updated_by	VARCHAR(100)	NULL	User who last updated the record
deleted_by	VARCHAR(100)	NULL	User who deleted the record
timestamps	DATETIME+DATETIME	NOT NULL	Laravel timestamps
deleted_at	DATETIME	NULL	Soft delete marker

Entity 7: admin_roles

Field	Data Type	Constraints	Description
admin_role_id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique admin role
role_name	VARCHAR(100)	NOT NULL	Role name
role_level	INT	NOT NULL	Access level
permissions	VARCHAR(255)	NULL	Role privilege flags
timestamps	DATETIME+DATETIME	NOT NULL	Laravel timestamps

Entity 8: admins

Field	Data Type	Constraints	Description
admin_id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique admin account
role_id	BIGINT	FOREIGN KEY → admin_roles	Admin role assignment
username	VARCHAR(100)	UNIQUE, NOT NULL	Login username
password_hash	VARCHAR(255)	NOT NULL	Password hash
email	VARCHAR(150)	UNIQUE, NOT NULL	Login email
timestamps	DATETIME+DATETIME	NOT NULL	Laravel timestamps
deleted_at	DATETIME	NULL	Soft delete marker

Entity 9: admin_actions

Field	Data Type	Constraints	Description
action_id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique action identifier
admin_id	BIGINT	FOREIGN KEY → admins	Admin performing the action
action_type	VARCHAR(100)	NOT NULL	Type of action (Insert, Update, Delete, etc.)
description	TEXT	NULL	Summary of admin activity
timestamps	DATETIME+DATETIME	NOT NULL	System generated time logs

4.3 Relationships

Relationship 1

address (1) → (∞) owners
Type: One-to-Many
Description: A single address can be linked to multiple owners.

Relationship 2

owners (1) → (∞) vehicles
Type: One-to-Many
Description: Each owner can register multiple vehicles.

Relationship 3

vehicles (1) → (∞) registrations
Type: One-to-Many
Description: Each vehicle can have multiple registration records across time.

Relationship 4

vehicles (1) → (∞) fines
Type: One-to-Many
Description: A vehicle may incur multiple fines.

Relationship 5

vehicles (1) → (∞) insurance
Type: One-to-Many
Description: A vehicle may have multiple insurance entries.

Relationship 6

admin_roles (1) → (∞) admins
Type: One-to-Many
Description: Admins inherit access rights and permissions from assigned roles.

Relationship 7

admin_actions (∞) ↔ (∞) owners
Type: Many-to-Many

Description: Admin actions may involve multiple owners, and owners can appear in multiple admin actions.

Relationship 8

admin_actions (∞) ↔ (∞) vehicles
Type: Many-to-Many
Description: Admin actions can be linked to one or more vehicles, and vehicles may be involved in multiple admin actions.

Relationship 9

admin_actions (∞) ↔ (∞) registrations
Type: Many-to-Many
Description: Admin actions may involve one or more registration records, and registrations may connect to multiple admin actions.

Relationship 10

admin_actions (∞) ↔ (∞) fines
Type: Many-to-Many
Description: Admin actions may resolve or assign multiple fines, and fines may be associated with multiple admin actions.

Relationship 11

admin_actions (∞) ↔ (∞) insurance
Type: Many-to-Many
Description: Admin actions may approve or modify multiple insurance entries, and insurance records may be linked to multiple admin actions.

Relationship 12

admins (1) → (∞) admin_actions
Type: One-to-Many
Description: A single admin can perform multiple actions recorded in the system.

4.4 Database Normalization

This database is fully normalized to **Third Normal Form (3NF)**:

✔ First Normal Form (1NF)

- All data values are atomic
- No repeating groups or multivalued attributes

- Each record is uniquely identifiable via a primary key

✔ Second Normal Form (2NF)

- All non-key attributes depend on the whole primary key
- No partial dependencies exist due to single-column primary keys

✔ Third Normal Form (3NF)

- No transitive dependencies
- Every attribute depends only on its table's primary key
- Reference tables (address, admin_roles, owners) remove redundancy

Normalization benefits:

- Eliminates update anomalies
- Prevents redundant storage
- Improves data integrity
- Strengthens referential consistency

5. APPLICATION FEATURES & FUNCTIONALITY

5.1 User Authentication & Session Management

Handles secure login, logout, and session lifecycle.
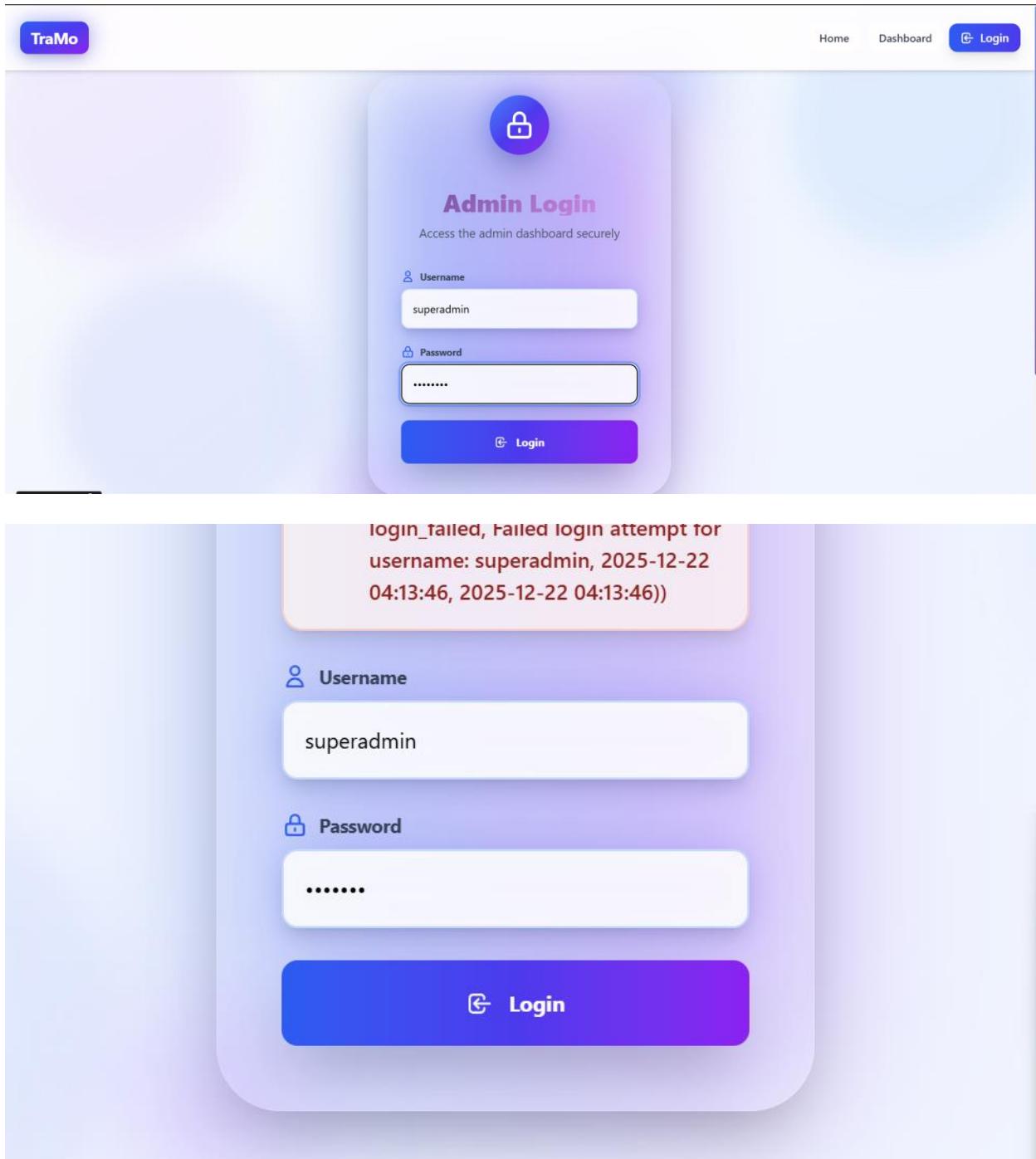
Functionality:

- Email/username and password login (admin)
- Secure session creation
- Session invalidation on logout
- Route protection using middleware

Implementation:
Laravel authentication with hashed passwords and middleware-protected routes.

Screenshots:

login_failed, Failed login attempt for username: superadmin, 2025-12-22 04:13:46, 2025-12-22 04:13:46))

5.2 User Management & Authorization

Controls access permissions based on user roles.
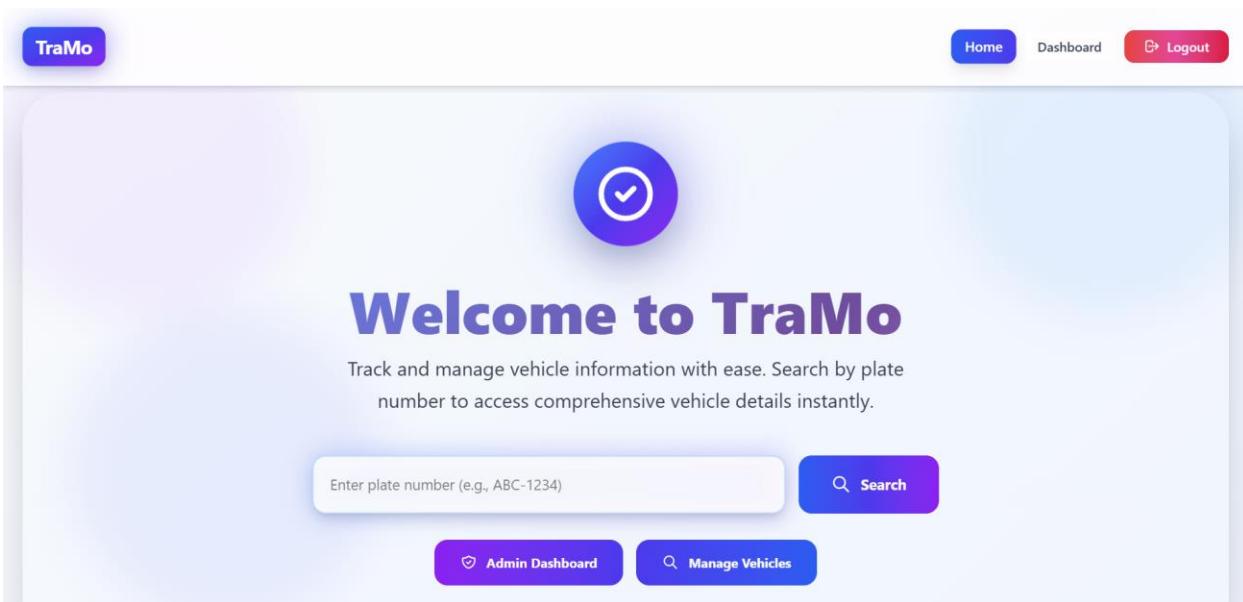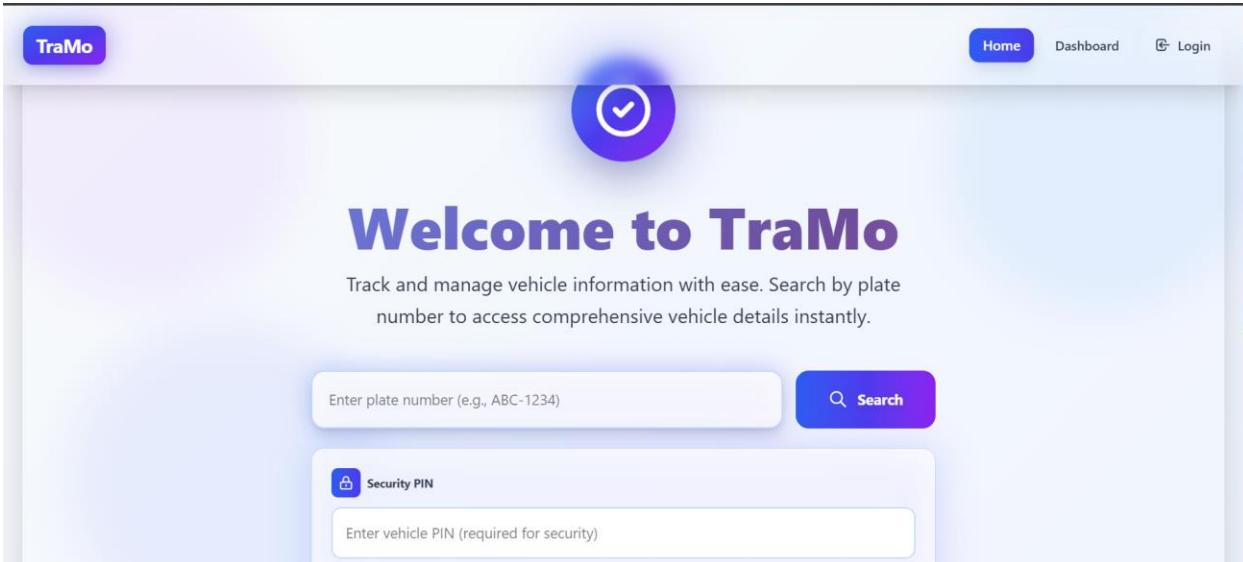
Functionality:
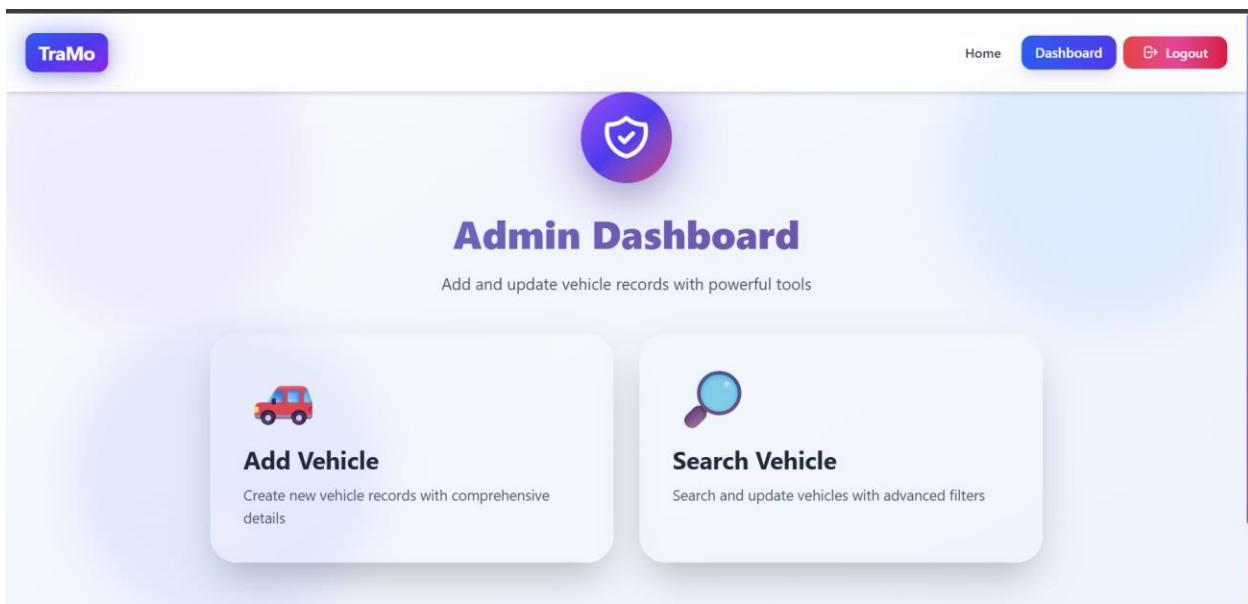
- Assign roles to users
- Restrict feature access

Implementation:
Role checks enforced via Laravel middleware.

Screenshots:

5.3 Vehicle Management Module

Manages all vehicle-related records.

Functionality:

- Add vehicles with all corresponding details

Implementation:
Laravel controllers handle CRUD logic; React updates UI dynamically.

Screenshots:

5.4 Vehicle Information Viewing (Public User Feature)

Description:
This feature allows public users to view detailed vehicle information without requiring authentication. It ensures transparency while maintaining data security through read-only access.

Functionality:
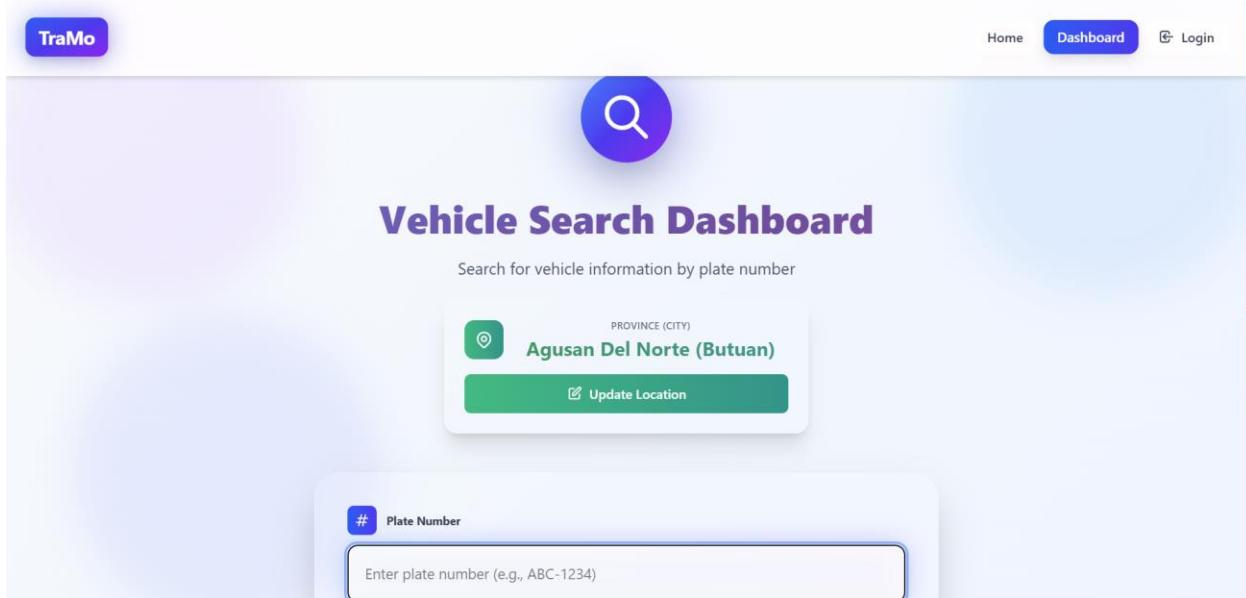
- Search for a vehicle using plate number and security PIN.
- Province Based Search
- View vehicle details including:
  - Vehicle specifications (make, model, year, status)
  - Owner information
  - Registration details

- Insurance policy status
    - Fines and violations with due dates
- Clearly distinguish paid and unpaid fines.
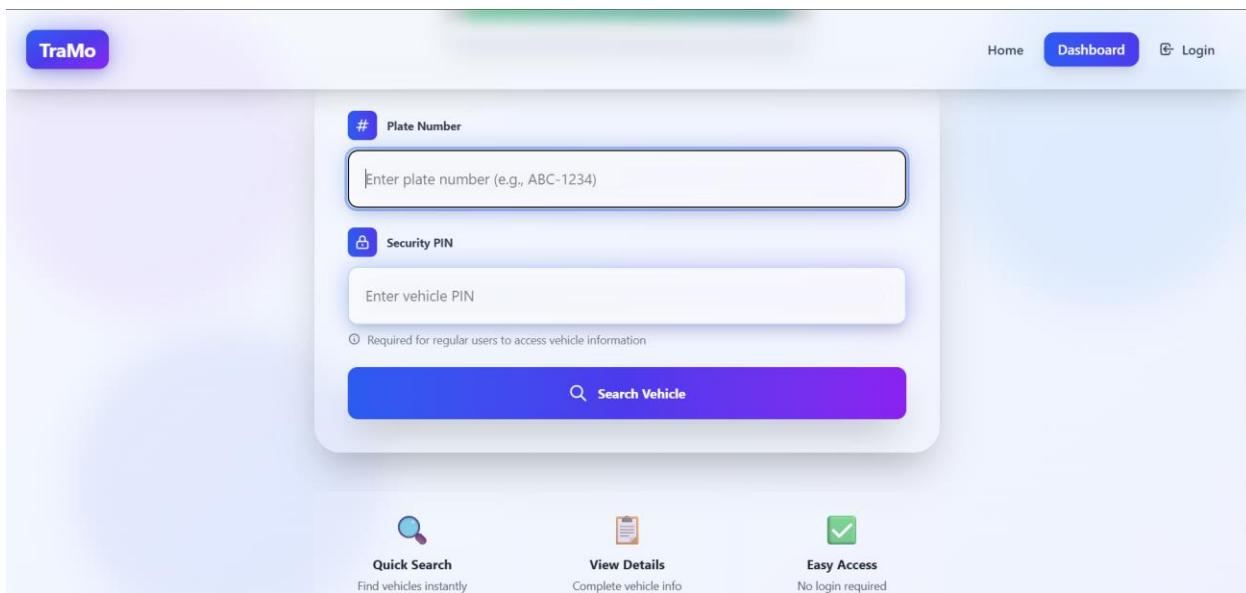
Implementation:
The backend exposes public API endpoints that retrieve vehicle-related data using relational database queries. The React frontend renders this information in a structured, scrollable layout. No update or delete actions are available to public users.

Screenshots:

Vehicle Details

Plate: SCS-3442   Stolen

Vehicle Information

VIN
EB479980RP

MAKE
Bode-Leuschke

MODEL
neque

YEAR
1979

Owner Information

FULL NAME
Hester Ortiz

PHONE NUMBER
325.726.7742
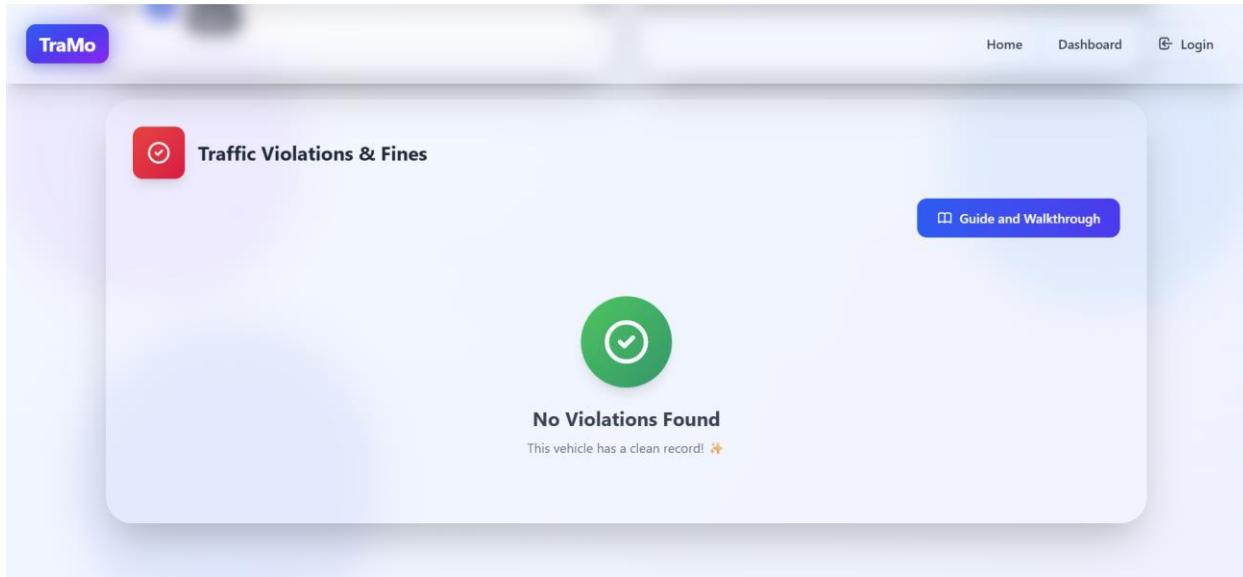
Registration Details

PLATE NUMBER
SCS-3442

REGISTRATION NUMBER
REG-4129-6329

START DATE
2025-11-14

EXPIRATION DATE
2026-03-21

STATUS
active

Insurance Policy

No Insurance Information Available

5.5 Vehicle Update & Management (Administrator Feature)

Description:
This feature enables administrators to manage and maintain accurate vehicle records within the system. It supports full control over vehicle-related data.

Functionality:

- View complete vehicle profiles.
- Update vehicle details such as VIN, make, model, year, color, and status.
- Update linked owner information.
- Modify registration, insurance, and fine records associated with a vehicle.
- Delete vehicle records with confirmation (soft delete).

Implementation:
Laravel controllers handle secured CRUD operations, protected by role-based middleware. Admin-specific API endpoints allow updates to vehicle-related entities. The React interface provides editable forms and confirmation feedback after successful updates.

Screenshots:

5.6 Violation Guidance & Resolution Information (User Support Feature)

Description:

This feature provides vehicle owners with clear guidance on what actions to take when a violation is recorded. It helps users understand the nature of the violation and the steps required for resolution, reducing confusion and improving compliance.

Functionality:

- Display a dedicated **"Violation Guidance"** tab within the vehicle details view.
- Show detailed information for each violation, including:
    - Violation type and description
    - Issuing authority or location (if applicable)
- Provide clear instructions on:
    - Where the violation can be settled (e.g., traffic office, authorized payment centers)
    - Required documents or reference numbers

Implementation:

Violation guidance data is retrieved from the backend alongside fine records using relational queries. The React frontend presents this information in a structured, user-friendly tab layout. Content is displayed in read-only format and is accessible to public users, while administrators can manage and update violation-related instructions through secured admin interfaces.

Screenshots:

🔒 2. DRIVING WITHOUT LICENSE / EXPIRED LICENSE / WRONG LICENSE CLASS

What?

Operating a motor vehicle without carrying a valid license or using the wrong license type.

Steps:

1. Keep the ticket.
2. Bring ID and (if you have it) your actual license.
3. If expired, renew your license first at LTO.
4. Pay violation at issuing office.
5. Get Official Receipt.

📄 3. NO HELMET (Motorcycle rider or backrider)

What?

Riding without a standard motorcycle helmet.

Steps:

5. Keep receipts for both renewal and violation.

⊘ UNIVERSAL CHECKLIST

(WORKS FOR ALL VIOLATIONS)

1. Keep the ticket (OVR).
2. Identify which agency issued it.
3. Prepare needed documents:
   - Valid ID
   - Driver's License
   - OR/CR
   - Authorization letter if not the owner
4. Go to correct office (LTO / MMDA / City Hall)
5. Pay OR contest within the allowed period
6. Get Official Receipt as proof

6. SECURITY & ERROR HANDLING

6.1 Security Measures Implemented

Authentication:
Username/email and password authentication with bcrypt hashing.

Authorization:
Role-based access control using middleware.

Input Validation:
Server-side validation using Laravel validation rules.

Data Protection:
Hashed passwords and secured environment variables.

Vulnerability Protection:

- CSRF protection
- SQL injection prevention via ORM
- XSS protection through sanitization

6.2 Error Handling

Backend:

- Structured JSON errors
- Proper HTTP status codes
- Exception handling

Frontend:

- Inline validation messages
- Clear user feedback for failures

7. Installation & Setup Instructions

7.1 Prerequisites

List of software and tools required to run TraMo locally:

- Node.js v14+ (for building & running the React frontend)
- npm v6+ or yarn (JavaScript package manager)
- Composer v2+ (PHP dependency manager)
- PHP v8.1+ (Laravel framework requirement)
- MySQL v8.0+ (database server)
- Laravel v10+ (backend framework)
- XAMPP or WAMP/LAMP (for local Apache & MySQL runtime)
- Postman / Thunder Client (API testing tool)
- Git (version control)
- Visual Studio Code (recommended IDE)

7.2 Installation Steps

Steps to set up the TraMo application:

1. Clone the repository from GitHub (https://github.com/CHRISTIAN-GONZAGA/ITE-18-FINAL-PROJECT.git).
2. Navigate to the backend folder.
3. Install backend dependencies using Composer.
4. Create a `.env` file and configure environment variables, including database credentials.
5. Generate the Laravel application key.
6. Run database migrations and seed base data.
7. Start the backend API server.
8. Navigate to the frontend folder and install dependencies.
9. Start the frontend development server.

7.3 Running the Application

Once both servers are running:

- **Frontend URL:** http://localhost:3000
- **Backend API URL:** http://127.0.0.1:8000/api/v1
- **Default Login Credentials (sample):**
  - Username: `admin`
  - Password: `password123`

Users can now access the dashboard, vehicle management screens, TraMo module, and all other system features.

8. Testing

8.1 Test Cases

Testing overview performed to verify functional system behavior:

Test Case	Steps	Expected Result	Actual Result	Status
Login Authentication	Submit valid username & password via admin login	Successful login + token returned	Working as expected	PASS
Create Vehicle	Submit form data to create a vehicle	New vehicle record saved in database	Record successfully saved	PASS
Delete Owner	Delete owner record	Owner removed + database updated	Owner deleted successfully	PASS
Invalid Token Access	Attempt route access without valid token	Access denied + error message displayed	Error handled correctly	PASS
Search VIN	Search for vehicle by VIN	Vehicle record returned	Record returned correctly	PASS

Additional tests included database validation, frontend form validation, and exception handling.

8.2 Known Issues & Limitations

Current limitations identified during testing:

- No real-time push notifications for expiring insurance or registration records.
- No CSV/Excel import-export module; data entry must be manual.
- Minimal dashboard visualization; graphical statistics can be improved.
- Mobile responsiveness incomplete; UI not fully optimized for smartphones.
- Some API endpoints require pagination; large datasets may affect performance.

Planned improvements:

- Add push alert system for compliance deadlines.
- Integrate advanced reporting and analytics dashboards.
- Add mobile UI enhancements.
- Implement caching for performance optimization.

9. Code Quality & Documentation

9.1 Code Structure

Overview of the folder structure:

```
tramo/
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   ├── pages/
│   │   └── utils/
│   ├── public/
│   └── package.json
├── backend/
│   ├── app/
│   │   ├── Http/
│   │   │   ├── Controllers/Api/
│   │   │   └── Middleware/
│   │   ├── routes/
│   │   │   └── api.php
│   │   ├── models/
│   │   └── composer.json
├── database/
│   ├── migrations/
│   ├── seeders/
│   └── schema.sql
└── README.md
```

9.2 Code Standards

- **Naming Conventions:** camelCase for variables & functions; PascalCase for React components & PHP classes; snake_case for database fields.
- **Code Style:** ESLint + Prettier for frontend; PSR-12 for Laravel backend.
- **Comments & Documentation:** Functions include inline comments; API behavior documented inside controllers; README contains setup instructions.
- **Version Control:** GitHub used for repository management; branching strategy: main, dev, feature branches; pull requests required for code merge.

9.3 API Endpoints

Authentication

POST /api/v1/admin/login

- Login using username and password
- **Response:** JWT token + user info
- **Status Codes:** 200, 401

POST /api/v1/admin/logout

- Invalidates user session
- **Status Codes:** 200, 403

GET /api/v1/admin/roles

- Retrieves all admin roles

POST /api/v1/admin/create *(protected)*

- Creates a new admin account

GET /api/v1/admin/actions *(protected)*

- Retrieves all actions performed by admins

GET /api/v1/admin/profile *(protected)*

- Retrieves current admin profile

Vehicles *(protected)*

GET /api/v1/get-vehicle-list-with-owners – Returns all vehicles with owner details
POST /api/v1/create-vehicle – Creates a new vehicle record
PUT /api/v1/update-vehicle/{vehicleId} – Updates a vehicle record
DELETE /api/v1/delete-vehicle/{vehicleId} – Deletes a vehicle record

Owners *(protected)*

GET /api/v1/get-owner-list – Retrieves all owners
GET /api/v1/get-owner-by-id/{ownerId} – Retrieves a specific owner
POST /api/v1/create-owner – Creates a new owner
PUT /api/v1/update-owner/{ownerId} – Updates owner details
DELETE /api/v1/delete-owner/{ownerId} – Deletes an owner

Registrations *(protected)*

GET /api/v1/get-registration-list – Retrieves all registration records
GET /api/v1/get-registration-by-id/{registrationId} – Retrieves a specific registration
GET /api/v1/get-registration-list-expiring-soon – Registrations nearing expiration
POST /api/v1/create-registration – Creates a new registration
PUT /api/v1/update-registration/{registrationId} – Updates registration
DELETE /api/v1/delete-registration/{registrationId} – Deletes registration

Insurance *(protected)*

GET /api/v1/get-insurance-list – Retrieves all insurance records
GET /api/v1/get-insurance-by-id/{insuranceId} – Retrieves a specific insurance record
GET /api/v1/get-insurance-list-expiring-soon – Insurance expiring soon
POST /api/v1/create-insurance – Creates a new insurance record
PUT /api/v1/update-insurance/{insuranceId} – Updates insurance record
DELETE /api/v1/delete-insurance/{insuranceId} – Deletes insurance record

Fines *(protected)*

GET /api/v1/get-fine-list – Retrieves all fines
GET /api/v1/get-fine-by-id/{fineId} – Retrieves a specific fine
GET /api/v1/get-fine-list-overdue – Retrieves overdue fines
POST /api/v1/create-fine – Adds a new fine for a vehicle
PUT /api/v1/update-fine/{fineId} – Updates a fine
DELETE /api/v1/delete-fine/{fineId} – Deletes a fine

Advanced Queries / Analytics *(protected)*

GET /api/v1/get-vehicle-list-by-owner/{ownerId} – Returns vehicles owned by a specific owner
GET /api/v1/get-vehicle-history-by-id/{vehicleId} – Vehicle monitoring history
GET /api/v1/get-vehicle-search-by-vin – Search vehicle by VIN (query param: `vin`)
GET /api/v1/get-owner-list-with-vehicle-counts – Returns owners with their vehicle counts
GET /api/v1/get-vehicle-list-by-status/{status} – Returns vehicles filtered by status
GET /api/v1/get-dashboard-stats-summary – Dashboard metrics and analytics
GET /api/v1/get-vehicle-compliance-score-by-id/{vehicleId} – Vehicle compliance score

Nested / Combined Endpoint *(protected)*

POST /api/v1/vehicles – Create vehicle + owner in one request

Public Search

GET /api/v1/vehicle/search/{plate} – Search vehicle by plate number (no auth required)

10. REFERENCES

Laravel. (n.d.). *Laravel documentation*. Laravel. Retrieved December 2025, from
https://laravel.com/docs

React. (n.d.). *React documentation*. React. Retrieved December 2025, from
https://reactjs.org/docs

Oracle. (n.d.). *MySQL documentation*. MySQL. Retrieved December 2025, from
https://dev.mysql.com/doc/

W3Schools. (n.d.). *W3Schools references*. W3Schools. Retrieved December 2025, from
https://www.w3schools.com/

Traversy Media. (2023, October 15). *React & Laravel Full Stack Project Tutorial* [Video].
YouTube. https://www.youtube.com/watch?v=bHRe5XNP5l8

Jeffrey Way. (n.d.). *Laracasts*. Laracasts. Retrieved December 2025, from https://laracasts.com/

Mozilla. (n.d.). *MDN Web Docs: HTML, CSS, JavaScript reference*. Mozilla Developer
Network. Retrieved December 2025, from https://developer.mozilla.org/

Stack Overflow. (n.d.). *Stack Overflow – Community Q&A*. Stack Exchange, Inc. Retrieved
December 2025, from https://stackoverflow.com/

11. APPENDIX

11.1. Additional Diagrams

This section includes visual materials that support the TRAMO system. These diagrams provide a better understanding of the system's structure and workflow.

- **Entity-Relationship Diagram (ERD)** – illustrates the database schema, tables, and relationships used in the TRAMO system.
- **Flowcharts** – detailed flow of key processes such as inventory management, request submission, and transaction summaries.
- **Wireframes / UI Layouts** – visual representation of the main user interfaces of the system.

Note: The ERD, flowchart, and wireframes will be provided as separate image files showing the complete diagrams and are not embedded within this document file.

11.2 Supplementary Information

This section provides extended technical, design, and operational details that support the TraMo (Traffic Monitoring & Management System). The information included here enhances understanding of system decisions, constraints, usability considerations, and future scalability.

11.2.1 Role-Based Access Control (RBAC)

The TraMo system implements strict role-based access control to ensure security and data integrity.

- **Public Users**
    - Can search for vehicles using a registration number.
    - Can view vehicle details, insurance status, and fines/violations.
    - Have read-only access with no ability to modify records.
- **Administrators**
    - Must authenticate before accessing the system.
    - Have full CRUD (Create, Read, Update, Delete) permissions.
    - Can manage vehicles, owners, insurance policies, and fines.
    - Can correct or update records in real time.

This separation ensures sensitive data is protected while maintaining transparency for public users.

11.2.2 Data Validation and Integrity Controls

To prevent incorrect or inconsistent data, multiple validation layers are implemented:

- Mandatory fields for all critical forms (vehicle, owner, insurance, fines).
- Format validation for registration numbers, VINs, phone numbers, and dates.
- Logical validation (e.g., insurance end date must be after start date).
- System-level checks preventing orphan records (e.g., fines cannot exist without a vehicle).

These controls improve system reliability and reduce administrative errors.

11.2.3 Error Handling and System Feedback

User experience is enhanced through clear and immediate system feedback:

- Error messages for invalid login credentials.
- Notifications for unsuccessful vehicle searches.
- Confirmation messages after successful updates or record additions.
- Inline validation indicators on forms.

This feedback ensures users and administrators always understand system responses and next steps.

11.2.4 UI/UX Design Principles

The user interface design follows established usability principles:

- **Consistency:** Uniform layout, typography, and color schemes across all screens.
- **Clarity:** Clear labeling of fields, buttons, and sections.
- **Minimal Cognitive Load:** Public users are presented with only essential information.
- **Responsiveness:** Layouts designed to adapt to various screen sizes.
- **Accessibility:** Readable fonts, sufficient contrast, and logical navigation flow.

These principles contribute to a smooth and intuitive user experience.

11.2.5 Performance and Scalability Considerations

The system is designed with scalability in mind:

- Modular architecture allowing independent feature expansion.
- Efficient database queries to handle large volumes of vehicle records.
- UI refresh mechanisms that update only modified components.
- Separation of frontend and backend logic to improve performance.

These considerations support future growth and increased system usage.

11.2.6 Security Considerations

Security is a core design priority in the TraMo system:

- Encrypted credential storage for administrators.
- Restricted admin-only routes and screens.
- Protection against unauthorized data modification.
- Controlled session access with logout functionality.

These measures help safeguard sensitive system data and prevent misuse.

11.2.7 Assumptions and System Constraints

The following assumptions guided system design:

- All administrative users are trained and authorized.
- Vehicle registration numbers are unique identifiers.
- Data entered by administrators is accurate and verified.
- The system operates in an online environment.

Constraints include:

- No integrated payment processing in the current version.
- Manual entry of fines and insurance data.
- Dependence on correct administrative input.

11.2.8 Maintainability and Extensibility

The system is designed to be easily maintained and extended:

- Clear separation of concerns between UI, logic, and data.
- Reusable form components for vehicle, insurance, and fines.
- Consistent naming conventions and modular workflows.
- Documentation-friendly structure supporting future developers.

11.2.9 Auditability and Transparency

TraMo supports accountability through:

- Clear traceability of updates made by administrators.
- Consistent record structures for vehicles and violations.
- Logical workflows that reflect real-world traffic management processes.

This makes the system suitable for institutional or governmental use cases.

11.2.10 Future Enhancement Opportunities

Potential improvements identified for future iterations include:

- Online fine payment and receipt generation.
- Automated fine issuance through traffic monitoring systems.
- SMS/email notifications for fine due dates.
- Advanced reporting and analytics dashboards.
- Multi-language and regional support.
- Integration with national vehicle databases.

Student/Team Signature:

CHRISTIAN ANGELO M. GONZAGA
Date: December 22, 2025