A logo with blue text

Description automatically generated

**FASTCars**

**An Automated Car Overspeeding and License Plate Detector**

**PROJECT SUPERVISOR**

**Mr. Kashan Hussain**

**PROJECT TEAM**

**Usman Rasheed 21K-3225**

**Talha Minhaj 21K-3347**

**Muneeb Ali 21K-4924**

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

FAST SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES KARACHI CAMPUS

May 2025

|  |  |
| --- | --- |
| Project Supervisor | Mr. Kashan Hussain |
| Project Team | Usman Rasheed K21-3225  Talha Minhaj K21-3347  Muneeb Ali K21-4924 |
| Submission Date | Dec 12, 2024 |

Mr. Kashan Hussain

**Supervisor**

Dr. Ghufran Ahmed

**Head of Department**

FAST SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES KARACHI CAMPUS

**Acknowledgement:**

All the thanks to the supervisor Mr. Kashan Hussain, the members of the project Usman Rasheed, Talha Minhaj, Muneeb Ali, and people who cooperated in providing the data for the project. All the mentioned people have contributed to the project and helped us on this journey. Without their help we could not have done this.

Table of Contents

[Abstract 4](#_Toc197987482)

[Introduction 6](#_Toc197987483)

[Overview 6](#_Toc197987484)

[Functionalities 6](#_Toc197987485)

[Related Work 7](#_Toc197987486)

[Requirements 9](#_Toc197987487)

[Functional Requirements 9](#_Toc197987488)

[Use Case Diagram 10](#_Toc197987489)

[Use Cases 11](#_Toc197987490)

[Non-Functional Requirements: 14](#_Toc197987491)

[Design 15](#_Toc197987492)

[Local Device Layer 16](#_Toc197987493)

[AI Processing Server 16](#_Toc197987494)

[Database Layer 16](#_Toc197987495)

[Web Development Layer 17](#_Toc197987496)

[Implementation 20](#_Toc197987497)

[Test cases and Results 22](#_Toc197987498)

[Future Work 25](#_Toc197987499)

[Conclusion 25](#_Toc197987500)

[References 26](#_Toc197987501)

# Abstract

FASTCars is an intelligent traffic enforcement system designed to address the persistent problem of overspeeding in semi-controlled environments such as university campuses, corporate zones, and gated communities. The system combines hardware-based speed detection, computer vision for license plate recognition, and web-based interfaces for managing violations and fines. It operates autonomously to monitor vehicle speed, capture license plate details, and log overspeeding incidents in a centralized database. Registered vehicle owners are automatically notified of violations, and administrators are provided with tools to oversee system activity and manage user data. The project integrates hardware components, machine learning techniques, and modern web technologies to deliver a scalable, user-friendly, and cost-effective solution that enhances safety and accountability. FASTCars offers an open, modular, and replicable approach that is accessible for smaller institutions and communities seeking to implement smart traffic management systems. To view FASTCars in more details or to replicate its contents you can use the provided Github Repository: <https://github.com/syedmuneeb-17/FASTCars>.

# Introduction

## Overview

Overspeeding in high-traffic zones such as universities, office complexes, and gated communities presents a significant threat to pedestrian and vehicular safety. These environments often experience repeated vehicle entries and exits, making consistent monitoring and enforcement essential. However, existing traffic management systems often lack automation and rely heavily on manual processes, resulting in delayed detection, inefficient record-keeping, and inconsistent fine enforcement.

**FASTCars** addresses these shortcomings by providing an automated solution for speed monitoring, license plate recognition, and fine management. By integrating hardware-based sensing and intelligent software systems, FASTCars ensures real-time detection of traffic violations and streamlines the enforcement process. The system is designed to operate seamlessly across different types of users—registered and unregistered—ensuring fairness, accountability, and improved safety in closed or semi-public traffic environments.

## Functionalities

FASTCars offers a range of integrated functionalities categorized into four main areas:

**Admin Functions:**

* Add, update, and delete user data, including vehicle number plates, owner names, and contact details.
* View detailed overspeeding records, such as speed values, timestamps, and associated license plate images.
* Update fine payment statuses manually for users who pay in cash.
* Oversee system performance and ensure adherence to privacy and traffic compliance policies.

**Vehicle Monitoring Functions:**

* Detect vehicle speeds in real-time.
* Identify and log vehicles exceeding predefined speed limits.
* Record critical violation details such as vehicle number, speed, timestamp, and license plate snapshot.
* Categorize drivers as "Registered" or "Unregistered" based on database records and update logs accordingly.

**Notification and Fine Management Functions:**

* Automatically send fine notifications via email to registered users, including payment instructions and deadlines.
* Maintain detailed records of each violation's fine status (paid/unpaid).
* Support administrators in managing the lifecycle of each fine.

**Web Interface Functions:**

* Display an organized list of overspeeding events, complete with timestamps, vehicle details, and fine amounts.
* Provide visibility into fine payment statuses for easier tracking and follow-up.
* Deliver a user-friendly interface for managing both user and violation data efficiently.

# Related Work

Many traffic enforcement systems currently in operation—particularly those involving speed detection and license plate recognition—are **closed-source** and developed as part of **government contracts or private sector surveillance technologies**. These solutions often operate as black-box systems with **limited or no public access to their architecture, methodology, or source code**, making it difficult for educational institutions or smaller communities to adopt or replicate them.

Some of the most notable closed-source systems include:

* **Vigilant Solutions (Motorola Solutions)**: Used widely by law enforcement for automatic license plate recognition and vehicle location analytics. It is tightly integrated with law enforcement databases but is inaccessible to non-government users.
* **Sensys Gatso Group**: Specializes in speed enforcement cameras and automatic ticketing systems. Deployed across various city infrastructures, it provides robust speed monitoring but is costly and closed-source.
* **Genetec AutoVu**: A comprehensive license plate recognition system used in parking enforcement and campus security, offering high-accuracy recognition and tracking capabilities.
* **PlateSmart**: Provides license plate recognition as part of broader security analytics solutions for businesses and municipalities.

While these systems offer strong performance in their respective domains, they are often **not customizable**, **require large budgets**, and **lack transparency**, which limits their applicability for smaller institutions like universities or private campuses.

In contrast, several **academic and open community projects** have attempted to build smaller-scale systems with limited functionality:

* **Speed Detection with IR Sensors and Arduino**: Common in academic prototypes, this approach measures vehicle speed using timed IR sensor inputs. While functional for basic detection, these projects typically lack plate recognition or integration with web-based fine management.
* **Smart Campus Vehicle Logging Systems**: Some institutions have implemented basic systems to log vehicle entries and exits using RFID or manual registration, but these lack automated enforcement or real-time processing capabilities.
* **AI-Based Traffic Violation Detection (Research Prototypes)**: Various research efforts explore computer vision to detect red-light violations or unauthorized turns, but they rarely include fully integrated platforms for fines, user roles, or real-time notifications.

# Requirements

Functional requirements and the diagrams are given below:

## Functional Requirements

**1. Speed Detection and Overspeed Logging**

1.1. Speed Detection

* 1.1.1. Detect vehicle speed in real-time using IR sensors.

1.2. Overspeed Logging

* 1.2.1. Identify vehicles exceeding the speed limit.
* 1.2.2. Log overspeeding details, including:
  + 1.2.2.1. Number plate
  + 1.2.2.2. Speed value
  + 1.2.2.3. License plate image
  + 1.2.2.4. Timestamp

**2. License Plate Recognition**

2.1. Optical Character Recognition (OCR)

* 2.1.1. Use computer vision and EasyOCR to read license plates.
* 2.1.2. Log recognized license plate details into the database.

**3. Database Management**

3.1. Overspeeding Records

* 3.1.1. Store overspeeding records, including registered and unregistered users.

3.2. User Classification

* 3.2.1. Classify users as "Registered" or "Unregistered."
* 3.2.2. Maintain registered user details:
  + 3.2.2.1. Number plate
  + 3.2.2.2. Name
  + 3.2.2.3. Email

**4. Admin Features**

4.1. User Management

* 4.1.1. Add new user details (number plates, names, and emails).
* 4.1.2. Delete user records.

4.2. Record Management

* 4.2.1. View overspeeding records.

4.3. Payment Updates

* 4.3.1. Update fine payments for manual cash transactions.

**5. Notification System**

5.1. Email Notifications

* 5.1.1. Send email notifications to license plate holders for fines.
* 5.1.2. Include payment details and options in the email.

**6. Payment Tracking**

6.1. Fine Status Management

* 6.1.1. Maintain a status of fine payments.
* 6.1.2. Display fine payment statuses on the webpage.

**7. Webpage Features**

7.1. Record Display

* 7.1.1. Display overspeeding records with timestamps.
* 7.1.2. Display user information for both registered and unregistered users.

7.2. Fine Management Interface

* 7.2.1. Provide an interface for tracking fine payment statuses.

## Use Case Diagram

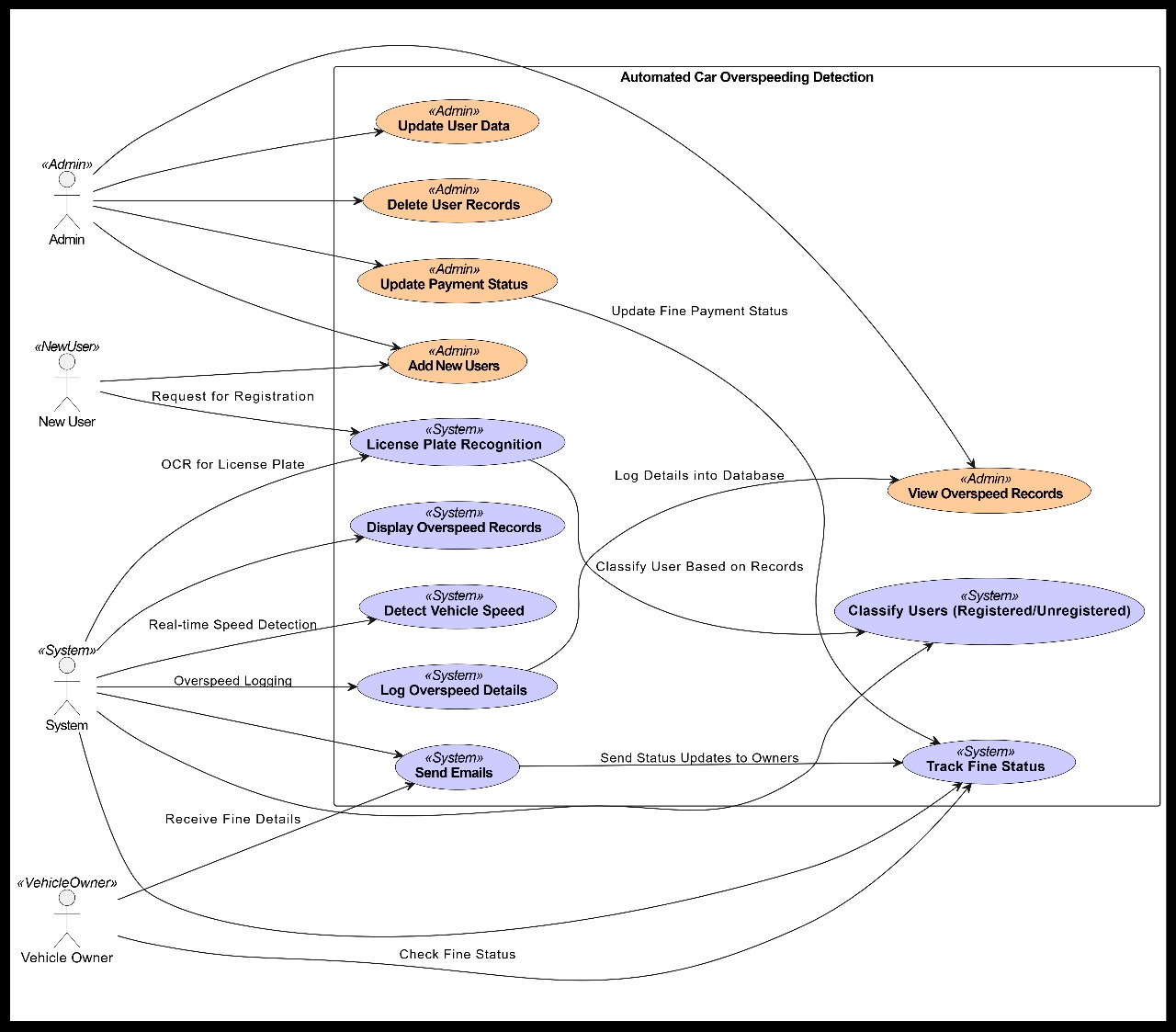


Figure 1: Use Case Diagram

## Use Cases

***Admin - User Management***

|  |  |  |
| --- | --- | --- |
| Use Case Id: | UC-AU-01 | |
| Actors: | Admin | |
| Feature: | Add, Update, and Delete User Records | |
| Pre-condition: | Admin must log in successfully. | |
| Scenarios |  | |
| Step# | **Action** | Software Reaction |
| 1 | Admin selects "Manage Users" from the menu. | System displays a list of existing users and options to add, update, or delete users. |
| 2 | Admin selects "Add User" and enters details (e.g., name, email, license plate). | System validates the input for completeness and duplicate entries. |
| 3 | Admin submits the form. | System saves the new user record and confirms the addition. |
| Alternate Scenarios: | 1a: If required fields are missing, the system highlights the missing fields with error messages.  2a: If the vehicle plate number is already registered, the system notifies the admin and prevents duplication. | |
| Post Conditions: | | |
| Step# | **Description** | |
|  | User details are successfully added, updated, or deleted in the database. | |
| Use Case Cross-Referenced: | UC-OS-01: Overspeed Logging, UC-VO-01: Receive Fine Notification. | |

***Admin - Fine Payment Update***

|  |  |  |
| --- | --- | --- |
| Use Case Id: | UC-AU-02 | |
| Actors: | Admin | |
| Feature: | Update Fine Payment Status | |
| Pre-condition: | Admin must have access to payment records. | |
| Scenarios |  | |
| Step# | **Action** | Software Reaction |
| 1 | Admin selects "Fine Management" from the menu. | System displays a categorized list of fines (e.g., Paid, Unpaid). |
| 2 | Admin selects an unpaid fine to view details. | System displays the fine details, including timestamp and amount. |
| 3 | Admin marks the fine as "Paid." | System updates the payment status and reflects it in the records. |
| Alternate Scenarios: | 1a:If the database connection fails, an error message is displayed, prompting the admin to retry later. | |
| Post Conditions: | | |
| Step# | **Description** | |
|  | Fine payment status is successfully updated in the database. | |
| Use Case Cross-Referenced: | UC-VO-01: Receive Fine Notification, UC-OS-01: Overspeed Logging. | |

***Vehicle Owner - Receive Fine Notification***

|  |  |  |
| --- | --- | --- |
| Use Case Id: | UC-VO-01 | |
| Actors: | Vehicle Owner | |
| Feature: | Receive Fine Notification | |
| Pre-condition: | Vehicle owner is registered and has committed an overspeeding violation. | |
| Scenarios |  | |
| Step# | **Action** | Software Reaction |
| 1 | System sends a fine notification email to the owner. | Email contains details like timestamp, speed, fine amount, and payment options. |
| 2 | Vehicle owner reviews the email. | Owner sees detailed instructions for fine payment. |
| Alternate Scenarios: | 1a: If email delivery fails, the system retries sending the email or notifies the admin. | |
| Post Conditions: | | |
| Step# | **Description** | |
|  | Fine notification is successfully delivered to the vehicle owner. | |
| Use Case Cross-Referenced: | UC-AU-02: Update Fine Payments, UC-OS-01: Overspeed Logging. | |

***System - Speed Detection***

|  |  |  |
| --- | --- | --- |
| Use Case Id: | UC-SYS-01 | |
| Actors: | System | |
| Feature: | Detect Speed of Vehicles | |
| Pre-condition: | IR sensors and speed detection system must be operational. | |
| Scenarios |  | |
| Step# | **Action** | **Software Reaction** |
| 1 | A vehicle passes through the IR sensor's detection zone. | System measures the vehicle's speed in real-time. |
| 2 | System compares the measured speed with the set speed limit. | If speed exceeds the limit, an overspeed record is created. |
| Alternate Scenarios: | 1a: If IR sensors malfunction, the system logs an error and notifies the admin. | |
| Post Conditions: |  | |
| Step# | **Description** | |
|  | Speed data is logged, and overspeeding records are created for violators. | |
| Use Case Cross-Referenced: |  | |
| - | UC-OS-01: Overspeed Logging, UC-SYS-02: License Plate Recognition. | |

***System - License Plate Recognition***

|  |  |  |
| --- | --- | --- |
| **Use Case Id:** | **UC-SYS-02** | |
| **Actors:** | System | |
| **Feature:** | License Plate Recognition using OCR | |
| **Pre-condition:** | Vehicle must pass in the detection zone for OCR to capture the plate. | |
| **Scenarios** |  | |
| **Step#** | **Action** | **Software Reaction** |
| 1 | System captures the vehicle's license plate image. | OCR processes the image to extract license plate details. |
| 2 | OCR reads and saves the license plate data to the database. | System associates the data with the overspeeding record. |
| **Alternate Scenarios:** | 1a:If OCR fails, the image is flagged for manual verification. | |
| **Post Conditions:** | | |
| **Step**# | **Description** | |
|  | License plate details are stored in the database and linked to the overspeeding record. | |
| **Use Case Cross-Referenced:** | UC-OS-01: Overspeed Logging, UC-AU-01: Admin Updates User Records. | |

## Non-Functional Requirements:

**Performance Requirements**

* The system should detect and log overspeeding with minimal delay
* The system should handle multiple data without any performance issues
* The system should be able to log and retrieve results in under 2 seconds
* The Database should be able to support 10,000 records

**Safety Requirements**

* The IR Sensors and camera should be placed such that they do not face any damage
* Data should be kept encrypted to prevent unauthorized access
* The system should include error handling in case of any malfunctions

**Security Requirements**

* All Data including license plate details and user information must be encrypted
* Only those with administrator privileges should have access to view the database and update it
* The system must be protected from unauthorized access such as SQL injections
* All the api keys should be securely stored
* Relevant Data must be salted and hashed

**User Documentation**

* A Detailed user manual should be created outlining how to operate and setup the system

**Operating Environment**

Anyone that uses the system must be aware that the project is broken into 4 modules, the user must ensure proper guidelines are followed for setting up each module:

**The Hardware Module:** The user must ensure that the hardware module has an OS that can run python and can connect to an Arduino and a camera, the settings for the Arduino must ensure that the speed is set according to the requirements of that particular area, the user must ensure the camera and the IR Sensors are placed such that they can perform their tasks respectively when it comes to capturing speed and license plates.

**The AI Server:** The user must ensure all API’s such as gmail’s smtp api, MongoDB’s Api, and the apis of other tools used in the project is set up correctly. The user must ensure to upload the Ai codebase to a server that is able to handle their specific needs, and ensure the model is installed.

**Database:** User must ensure to enter relevant data according to their domain is entered with proper rights given to proper users, if they choose to update any documents then they have to update all other code accordingly.

**Website:** The user must ensure that they have a stable internet.

**System Constraints**

* The IR sensors and camera must be positioned in locations where they can effectively detect vehicle speed and capture clear images of license plates.
* The Sensors and camera must be placed with their range in mind
* The Camera connected has a high display quality and a high frame rate
* The speed limit of the location is configured properly by the admin.
* The system requires a stable internet connection for communication between the project components.
* The AI Servers processes can process the license plate detection effectively.
* The Gmail, MongoDB and other relevant API keys are set up properly by the system administrator.
* The Hardware of the system is connected to a proper power supply.
* The system may require maintenance in the case of any hardware damage

**Assumptions & Dependencies**

* Any vehicle that passes through the system has a license plate on it.
* Every user has provided correct details
* The distance between each IR Sensor has been set up correctly
* The System Admin correctly enters paid fines

# Design

FASTCars adopts a **modular and layered architecture** to ensure flexibility, scalability, and efficient task distribution. The project is divided into four primary layers: **Local Device**, **AI Processing Server**, **Database**, and **Web Development**. Each module communicates with the others through a combination of RESTful APIs and asynchronous data handling techniques, enabling seamless coordination across hardware and software components.

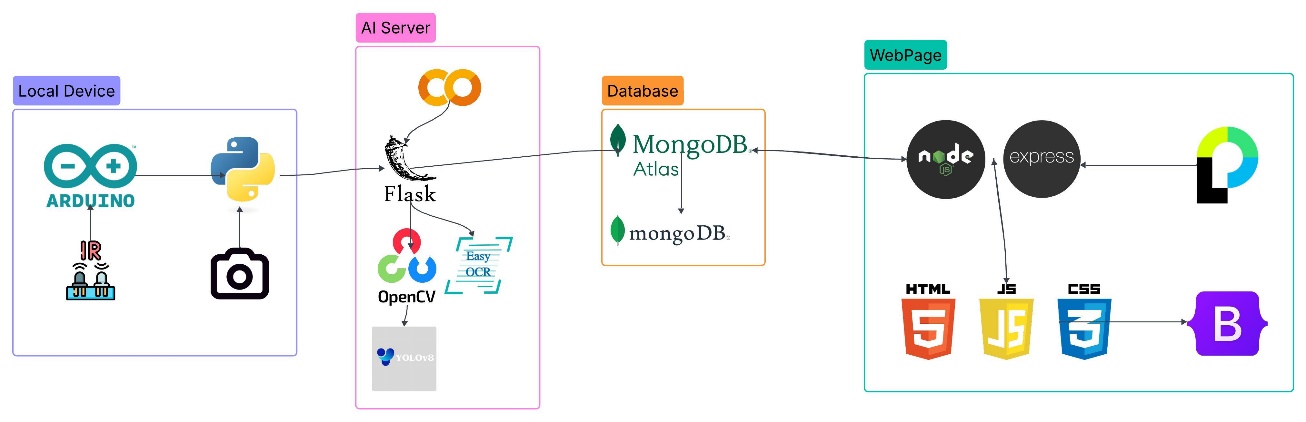


Figure 2: System Architecture Diagram

## Local Device Layer

The local layer serves as the system's hardware interface, integrating IR sensors and a Python-based script to detect vehicle movement and record video segments. This code is responsible for:

* Capturing and forwarding video footage of overspeeding incidents.
* Sending collected footage and sensor data to the AI Processing Server for analysis.  
  This layer runs on physical hardware located at the deployment site, providing real-time interaction with traffic activity.

## AI Processing Server

The AI Processing Server is currently hosted on Google Colab and is responsible for processing videos and analyzing speed violations using various AI and computer vision tools. It handles:

* Speed analysis and overspeeding detection.
* License plate extraction using OCR and a pre-trained object detection model.
* Bi-directional communication with the database to store and retrieve user and violation information.

Tools such as OpenCV and EasyOCR are employed to support video analysis and license plate recognition. Once a vehicle is flagged for overspeeding, relevant data—including the license plate, speed, and a photo is compiled and sent to the central database.

## Database Layer

The project uses **MongoDB Atlas** to store user information, speed violations, and session data. The database schema is designed for extensibility and real-time updates.

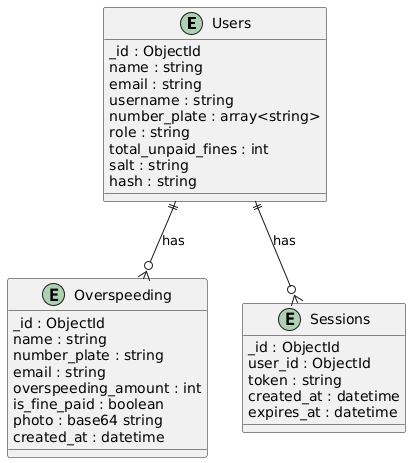


Figure 3: ER Diagram

## Web Development Layer

This is the **presentation and control** layer built using a **Model-View-Controller (MVC)** pattern. It includes:

* **Backend:** Node.js and Express.js for server-side logic and RESTful API creation.
* **Frontend:** Embedded JavaScript (EJS) and Bootstrap for responsive design.
* **Deployment:** Hosted online for access by admins and users.

Key features:

* Admin dashboard for managing users and fines.
* User dashboard for viewing personal overspeeding incidents and fine statuses.
* Mobile and desktop-friendly interface.
* Real-time communication with the database.

Below are some attached images of the Web Interface.

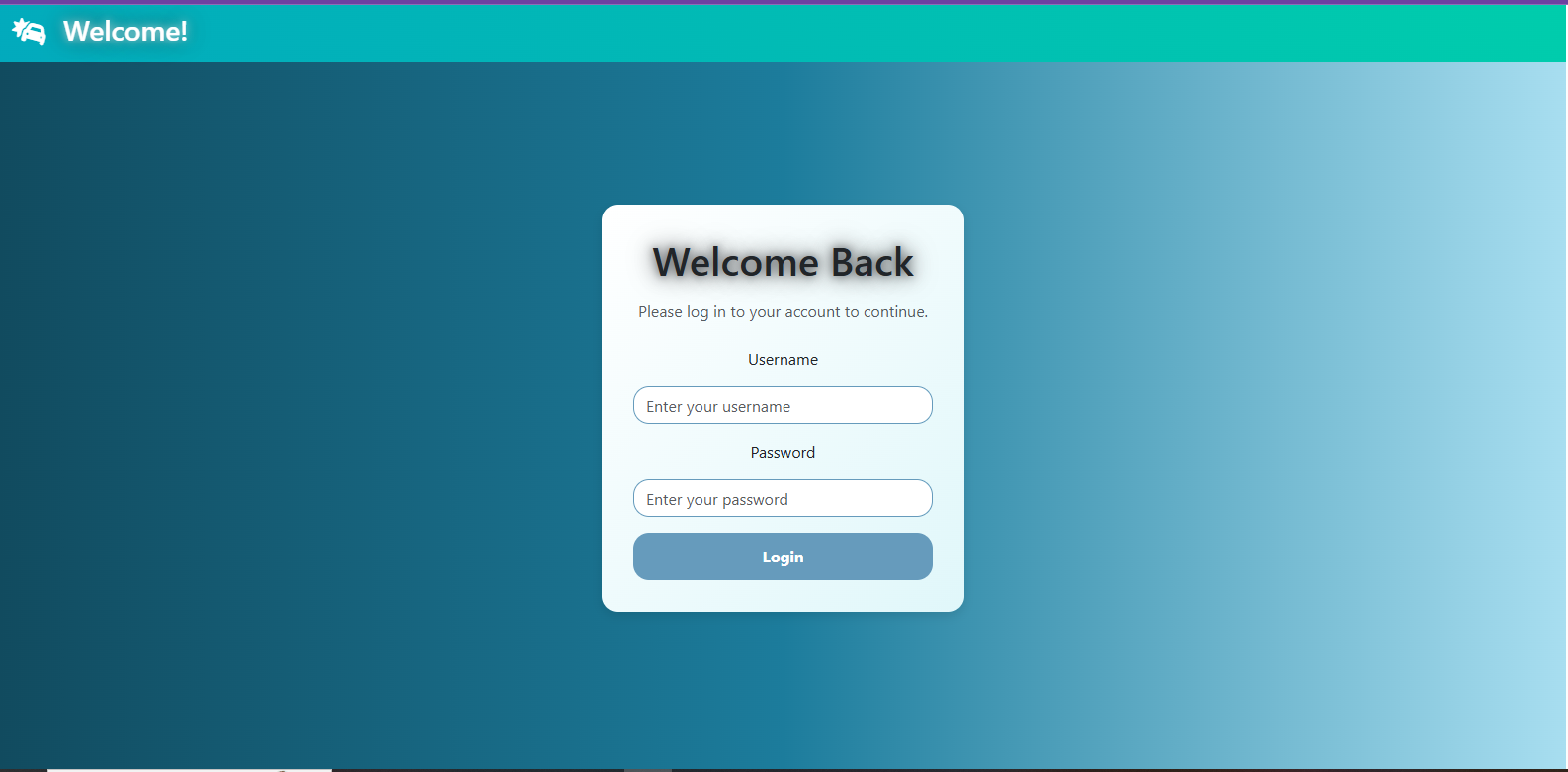


Figure 4.1: Web Interface Login Page

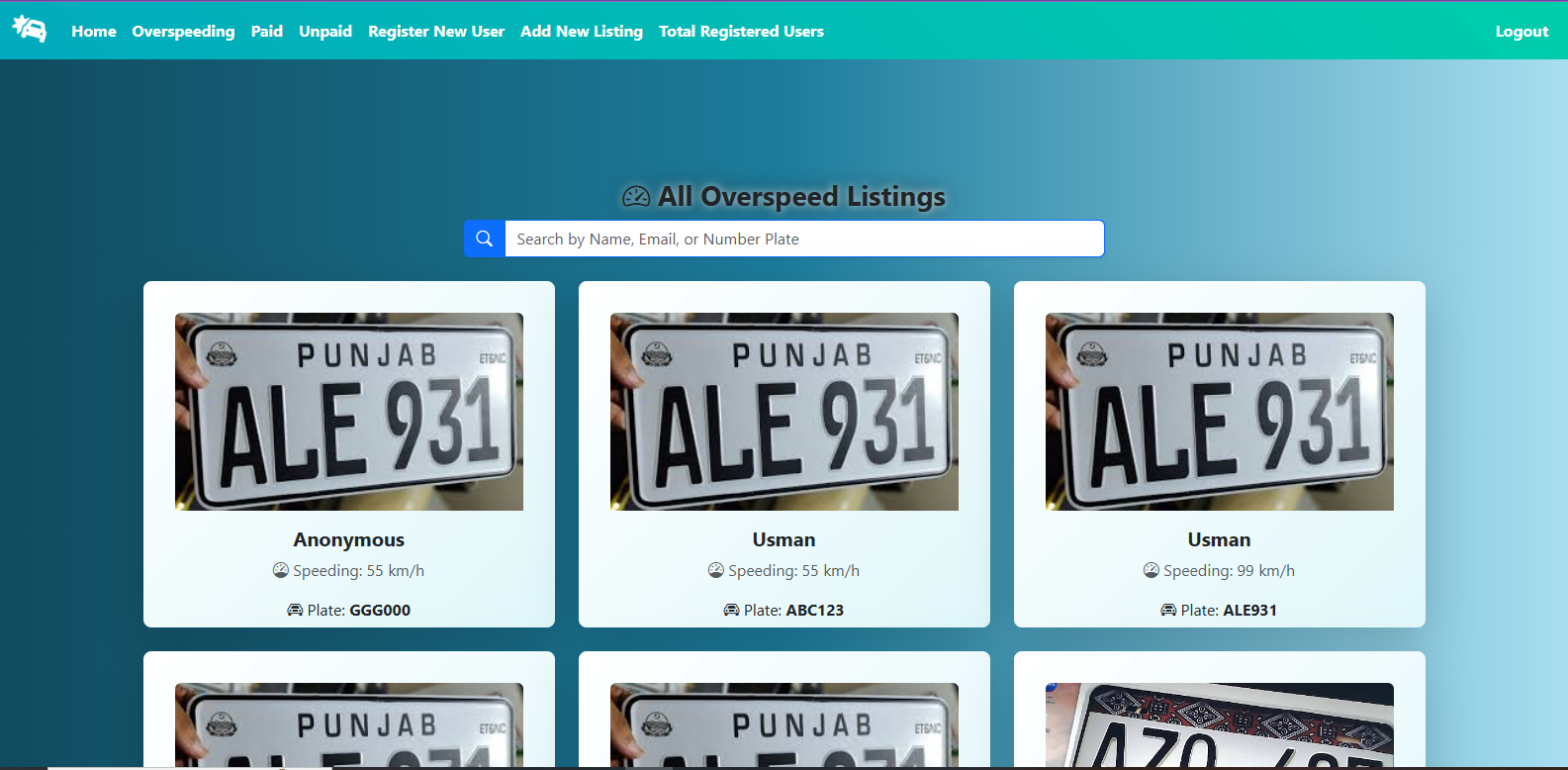


Figure 4.2: Web Interface records page

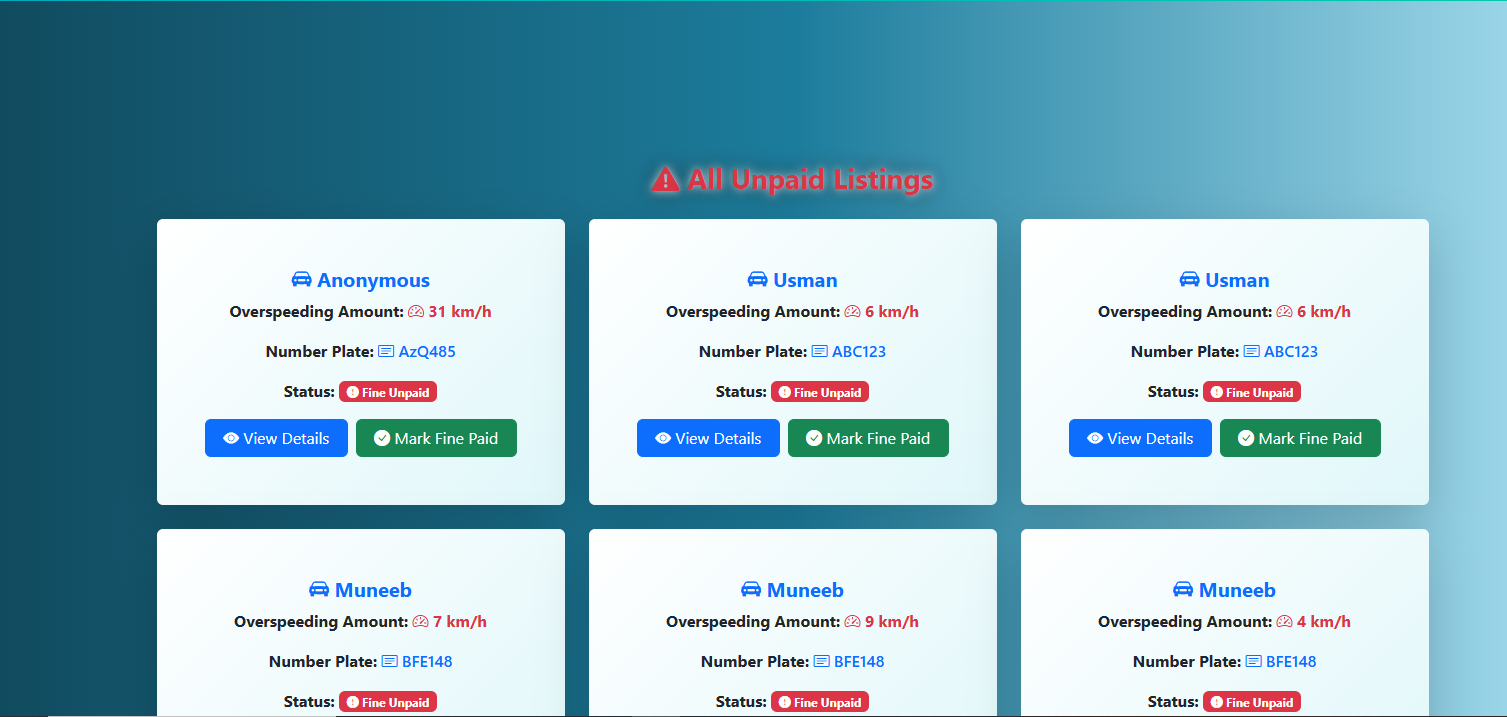


Figure 4.3: Web Interface Unpaid Listings Page

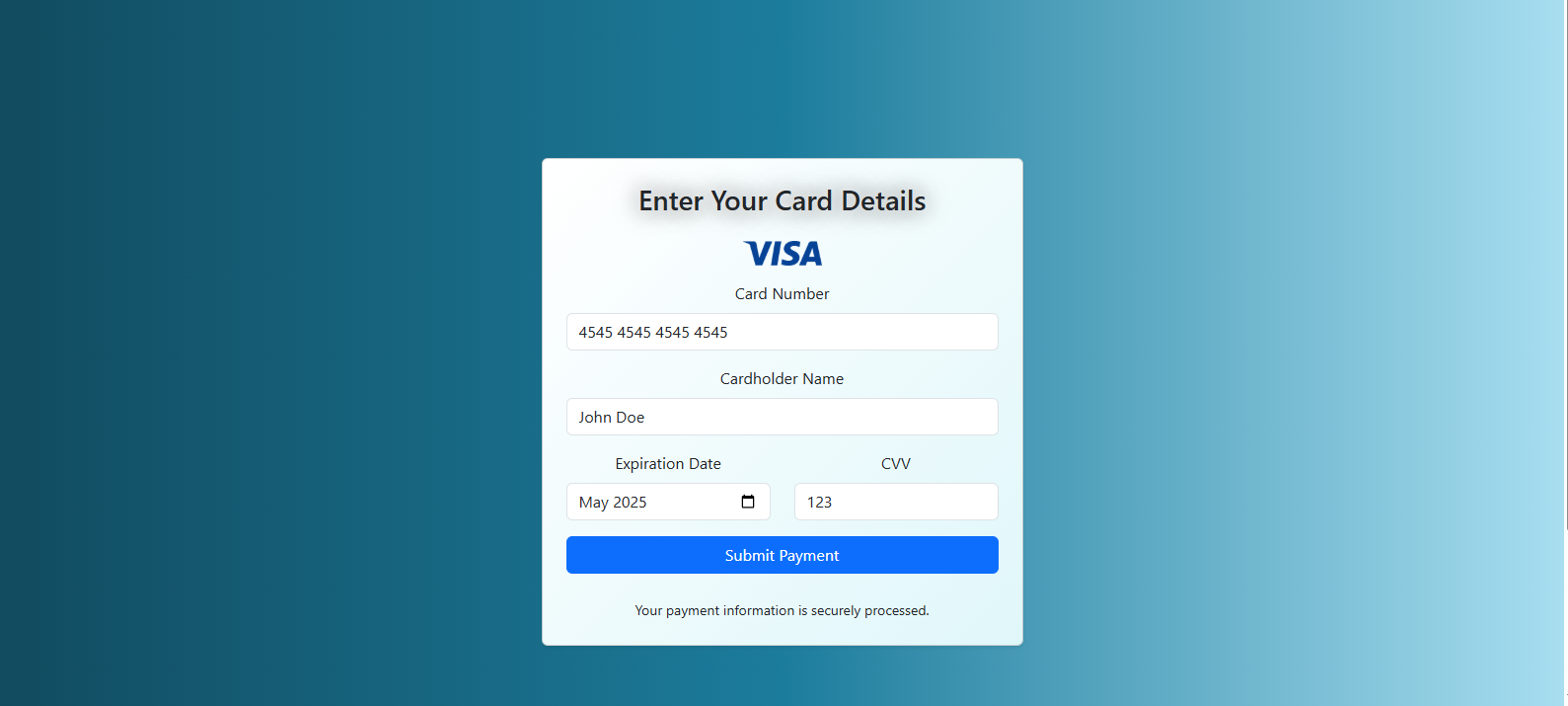


Figure 4.4: Web Interface Card Input form

# Implementation

The implementation of FASTCars translates its modular and layered design into a functioning system composed of hardware, AI-based detection, server APIs, database management, and a responsive web interface. Each component plays a specific role in achieving the system’s end-to-end automation for overspeed detection and fine management.

**Local Hardware & Detection System**

The local hardware unit consists of an **Arduino microcontroller** connected to **two IR sensors** placed a known distance apart. The Arduino continuously monitors the time it takes for a vehicle to pass between the sensors and calculates its speed. If the measured speed exceeds the defined threshold:

* The Arduino sends a signal to a **Python script running in a Jupyter notebook** on a local machine.
* The Python script activates a **camera** to record a short video of the vehicle.
* The script timestamps the footage and sends all relevant data (video, timestamp, speed value) to a RESTful API.

**AI Processing Server**

The AI server is hosted on **Google Colab**, and it performs the following steps:

* A **Flask API** running on Colab receives POST requests from the local system. The API is exposed to the public using **Ngrok**.
* The uploaded video is processed using a **YOLOv8 model** trained specifically to detect license plates. The trained weights are stored on **Google Drive** and loaded into Colab for inference.
* After license plate detection, **EasyOCR** is applied to extract the alphanumeric text from the plate.
* The system checks MongoDB to determine if the license plate belongs to a registered user:
  + If found, the user’s name and email are fetched.
  + If not found, the plate is marked as belonging to an **anonymous (unregistered)** user.
* **SMTP (Gmail API)** is used to automatically send a fine notification email to the vehicle owner if they are registered.
* All violation data is logged in the database, including speed, timestamp, license plate text, and an image.
* **Cloudinary** is used to upload and manage license plate images, allowing base64 image links to be saved in the database.
* **Mongoose** serves as the object data modeling (ODM) layer to interact with MongoDB.

**Database Integration**

The backend utilizes **MongoDB Atlas** for cloud-based data storage. It includes three main collections:

* **Users:** Information about registered users, vehicle numbers, emails, and login credentials.
* **Overspeeding Records:** Detailed logs of overspeeding events including photo evidence, speed data, and payment status.
* **Sessions:** Temporary authentication tokens for secure user and admin access.

Mongoose schemas are used to define and enforce data structure across all interactions with the database.

**Web Application**

The **web interface** is implemented using:

* **Node.js** and **Express.js** for server-side logic and API routing.
* **EJS (Embedded JavaScript)** templating for dynamic rendering.
* **Bootstrap** for responsive UI/UX design, ensuring compatibility on mobile and desktop.

The web app supports **two types of logins**:

* **Admin Login:**
  + Full access to all overspeeding records.
  + Ability to update fine statuses manually.
  + Manage registered users and their vehicle data.
* **User Login:**
  + View their own overspeeding history.
  + Check fine payment statuses.
  + See photo and timestamp proof of violations.
  + Perform fine payment through their debit/credit card

Session handling and authentication are implemented using the **Passport.js** library, which works in tandem with **Mongoose** for persistent session management.

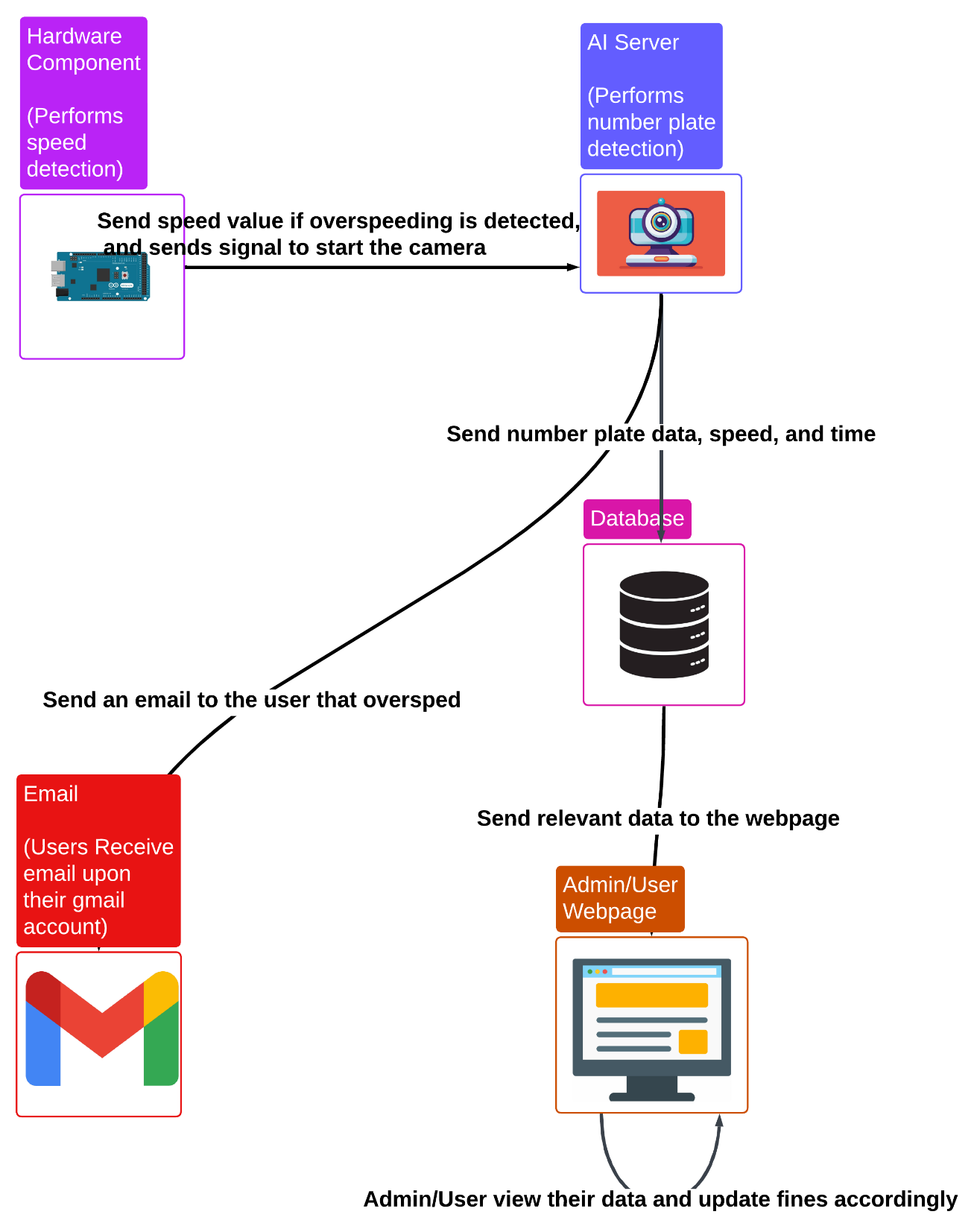


Figure 5: System WorkFlow Diagram

# Test cases and Results

The AI model for license plate detection was trained using Yolov8 on a model from Roboflow with a dataset of 2046 images for 15 epochs and achieved an accuracy of 97%. Similar results were observed during testing with real-world number plates.

Below are images attached with the results.

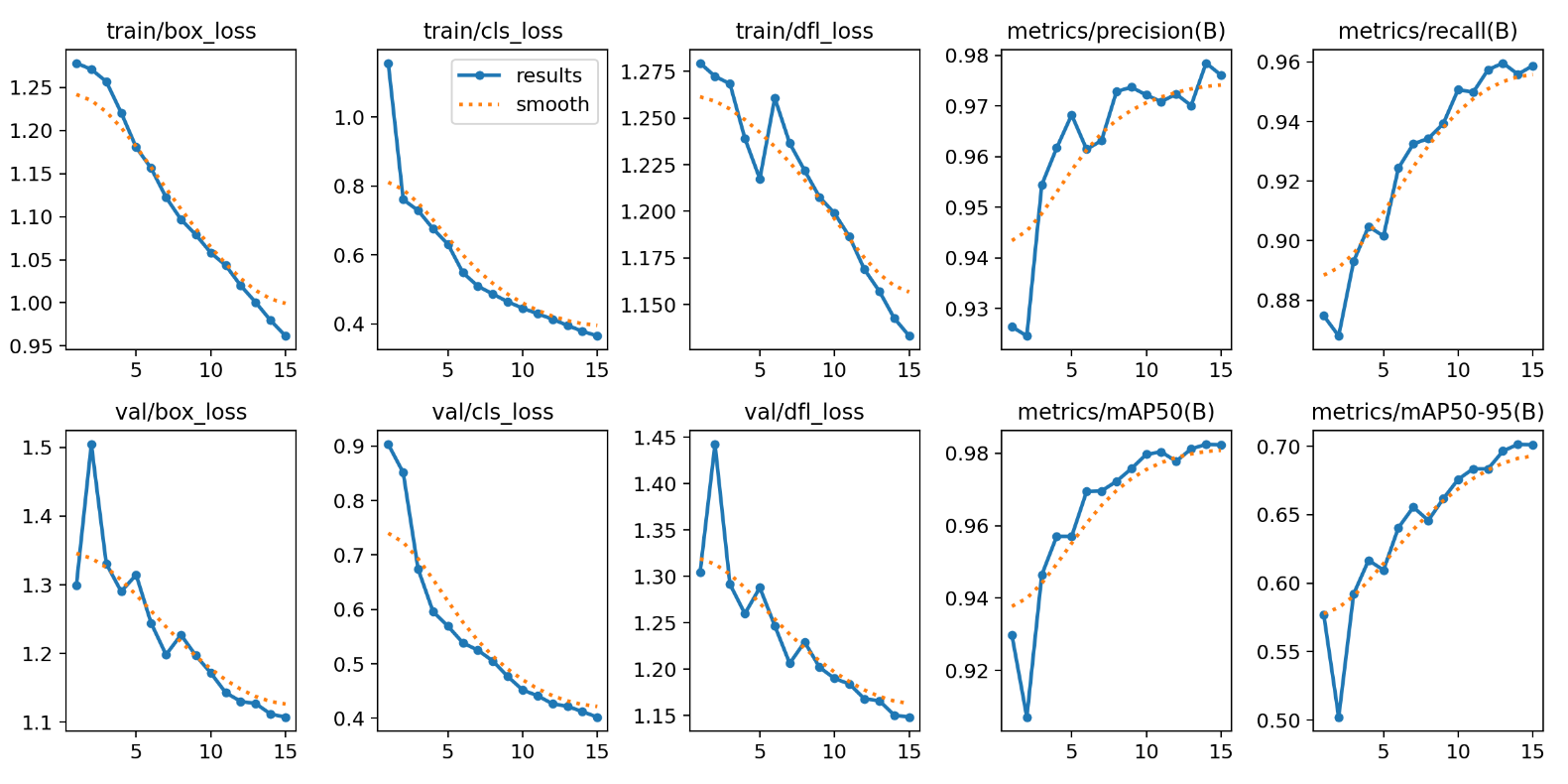


Figure 6.1: Test Results

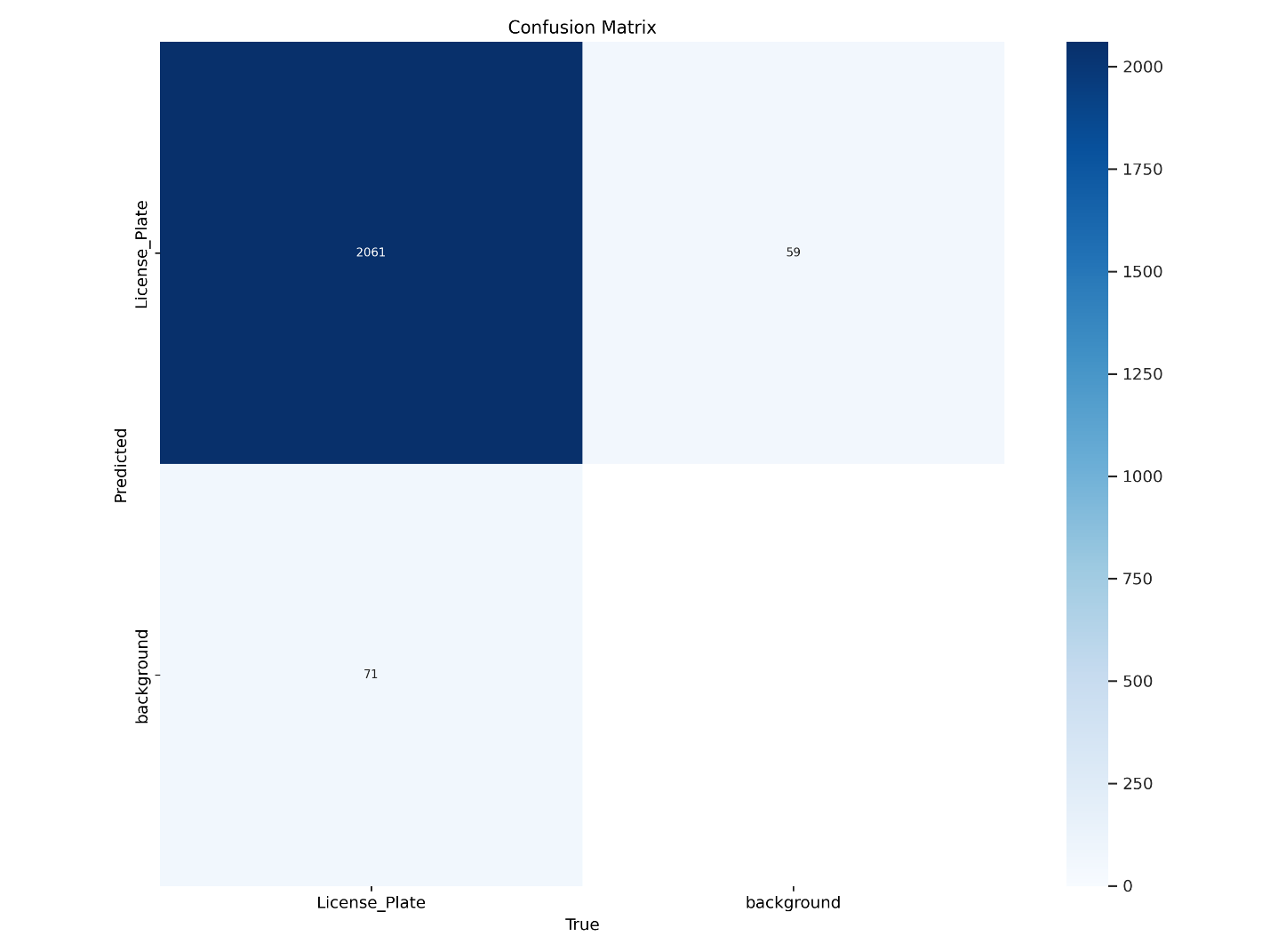


Figure 6.2: Test Results

Manual tests were performed upon the project to check if it was working properly, below is a table with the tests.

| **Test Case ID** | **Test Description** | **Input** | **Expected Output** | **Pass/Fail** | **Remarks** |
| --- | --- | --- | --- | --- | --- |
| TC-001 | Speed detection at threshold | Vehicle crosses IR sensors at 5 km/h | No overspeeding log | Pass |  |
| TC-002 | Speed detection over threshold | Vehicle crosses IR sensors at 10 km/h | Trigger overspeeding event | Pass |  |
| TC-003 | Camera activation | Overspeeding detected | Camera records short video | Pass |  |
| TC-004 | Video transmission | Video and data from local system | API receives and stores data | Pass |  |
| TC-005 | YOLOv8 License Plate Detection | Image with visible license plate | Bounding box around license plate | Pass |  |
| TC-006 | EasyOCR Plate Reading | Image with clear license plate text | Alphanumeric license plate string | Pass | Blurry plates may affect accuracy |
| TC-007 | Database Check for User | License plate from OCR | Matching user info returned | Pass |  |
| TC-008 | Email Notification to Registered User | Registered plate and user email | Email sent to user | Pass |  |
| TC-009 | Email Skipped for Unregistered User | Unregistered plate | No email sent | Pass | "Anonymous" tag added in database |
| TC-010 | Record Creation in MongoDB | Overspeeding event data | MongoDB document inserted | Pass |  |
| TC-011 | Admin Login and Dashboard | Valid admin credentials | Admin dashboard visible | Pass |  |
| TC-012 | User Login and History View | Valid user credentials | User sees their fine history | Pass | Unauthorized access blocked |
| TC-013 | Payment Update by Admin | Admin clicks "Mark as Paid" | Fine marked as paid in system | Pass |  |
| TC-014 | View Overspeeding Logs | Access record listing page | Logs listed with photos, timestamps, and statuses | Pass | Mobile and desktop layouts work as intended |

Table 1: Test cases

# Future Work

The current implementation of FASTCars utilizes a single hardware unit connected to a Windows-based laptop for overspeeding detection and preliminary testing. While this setup was suitable for prototyping and demonstration purposes, it is not ideal for large-scale or real-world deployment. In future iterations, the system can be adapted to use a dedicated embedded Linux device—such as a Raspberry Pi or similar single-board computer—running a lightweight kernel specifically tailored for the project's tasks. This would significantly reduce cost, improve portability, and make the hardware easily replicable for deployment across multiple locations.

Additionally, the project currently relies on Google Colab as the AI processing server, primarily to take advantage of occasional access to free GPU resources during low-traffic testing. For a production environment, a dedicated server or cloud-based infrastructure should be employed to ensure consistent availability, scalability, and responsiveness to handle high data volumes and concurrent requests.

# Conclusion

FASTCars represents a significant step toward smarter and safer traffic systems, particularly in high-density environments such as universities, corporate offices, and gated communities. By combining hardware-based speed detection with AI-powered license plate recognition, FASTCars automates the entire process of identifying, logging, and penalizing overspeeding vehicles. This eliminates the need for manual monitoring and drastically reduces human error and enforcement delays.

One of the most impactful aspects of FASTCars is its commitment to the open-source ecosystem. By leveraging freely available tools and technologies, and making the system architecture, codebase, and design openly accessible. FASTCars encourages innovation, collaboration, and adaptation across communities. This opens up new opportunities for developers, researchers, and civic bodies to tailor and expand upon the solution for their own traffic management needs.

Moreover, FASTCars offers a cost-effective alternative to traditional, proprietary traffic enforcement systems that are often inaccessible to smaller institutions due to high licensing fees and technical barriers. Through its modular design and reliance on affordable hardware and cloud infrastructure, FASTCars ensures that effective traffic safety mechanisms can be implemented even with limited budgets.

In summary, FASTCars not only enhances road safety through automation and real-time monitoring, but also contributes meaningfully to the global push for open, scalable, and inclusive smart city technologies.

# References

[1] G. Jocher, A. Chaurasia, J. Qiu, and A. Stoken, "Ultralytics YOLOv8," *GitHub*, 2023. [Online]. Available: <https://github.com/ultralytics/ultralytics>

[2] JaidedAI, "EasyOCR: Ready-to-use OCR with 80+ languages," *GitHub*, 2020. [Online]. Available: <https://github.com/JaidedAI/EasyOCR>

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. [Online]. Available: <https://opencv.org>

[4] Google, "Colaboratory: Free Jupyter Notebooks Environment," *Google Research*, [Online]. Available: https://colab.research.google.com

[5] Arduino, "Arduino - Open-source electronics platform," [Online]. Available: <https://www.arduino.cc>

[6] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2018.

[7] Ngrok, "Secure introspectable tunnels to localhost," [Online]. Available: <https://ngrok.com>

[8] MongoDB, Inc., "MongoDB Atlas: The Multi-Cloud Database Service," [Online]. Available: <https://www.mongodb.com/cloud/atlas>

[9] Automattic, "Mongoose ODM for MongoDB," [Online]. Available: <https://mongoosejs.com>

[10] Cloudinary Ltd., "Cloudinary: Image and Video Upload, Storage, Optimization and CDN," [Online]. Available: <https://cloudinary.com>

[11] Google Developers, "Gmail SMTP Server," [Online]. Available: https://developers.google.com/gmail

[12] Node.js Foundation, "Node.js," [Online]. Available: <https://nodejs.org>

[13] Express.js Foundation, "Express - Fast, unopinionated, minimalist web framework for Node.js," [Online]. Available: <https://expressjs.com>

[14] EJS Contributors, "EJS - Embedded JavaScript Templates," [Online]. Available: <https://ejs.co>

[15] Bootstrap Team, "Bootstrap: Build fast, responsive sites," [Online]. Available: <https://getbootstrap.com>

[16] J. Hanson, "Passport.js: Simple, unobtrusive authentication for Node.js," [Online]. Available: <https://www.passportjs.org>

[17] Roboflow, "License Plate Recognition Project," *Roboflow*, [Online]. Available: <https://roboflow.com>.

[18] Vigilant Solutions (Motorola Solutions), "Automatic License Plate Recognition and Vehicle Location Analytics," *Vigilant Solutions*, [Online]. Available: <https://www.vigilantsolutions.com>.

[19] Sensys Gatso Group, "Speed Enforcement Cameras and Automatic Ticketing Systems," *Sensys Gatso Group*, [Online]. Available: <https://www.sensysgatso.com>.

[20] Genetec, "AutoVu License Plate Recognition System," *Genetec*, [Online]. Available: <https://www.genetec.com>.

[21] PlateSmart, "License Plate Recognition and Security Analytics Solutions," *PlateSmart Technologies*, [Online]. Available: <https://www.platesmart.com>.