**PROJECT ABSTRACT &**

**SOFTWARE HIGH-LEVEL DESIGN DOCUMENT:**

# SMART TRAFFIC ANALYSIS


TEAM 1

**Vijayraj Shanmugaraj 20171026**

**Sumaid Syed 20171092**

**Harshita Sharma 20171099**

**Eesha Dutta 20171104**

**Paryul Jain 20171083**

**Kasa Pranay 2019900081**

# PROJECT ABSTRACT

## 1. Project Overview:

Everyday a lot of accidents in the world take place due to recklessness of drivers on the road. Getting details about the parties involved in the accident manually takes time. Hence, we propose using CCTV cameras and applying computer vision techniques in order to monitor traffic situations. Previous attempts at this have been restricted to one or two aspects of rule-breaking. We want to build a unified platform which will detect cars which don't adhere to a set of rule-breaking aspects.

## 2. Proposed Solution:

- Building a **highly scalable web application** with features to upload video or embed live stream directly to the application. Once the user uploads a video/opens the live stream to the platform, he can see the output video stream on the application.

- The application will also **capture photos of vehicles breaking following rules** and store them in a database for future analysis. The app will be built such that it will be flexible to the addition of various other rules if required as well.

  The following scenarios will be taken care of by the app implemented in the project:

  a. Vehicles should stop for pedestrians.

b. Vehicles should stop if the signal is red..

c. Vehicles should be under the speed limit.

d. Crash detection.

# HIGH-LEVEL DESIGN DETAILS

## 1. Requirements to be fulfilled by system:

Automatically detect the traffic violations of rules being considered:

1) Compliance with traffic lights
2) Not stopping while pedestrians cross the road
3) Compliance with speed limit
4) Detection of a crash

1.1:

Violation detection should be done through the traffic data that is already available i.e. from

a) Video data streamed from cameras, or pre-recorded data fed by user
b) Real-time traffic signal data, that is available in an existing server.

1.2:

Traffic violation detection should happen automatically for all selected junctions and snapshots are to be stored in a database server.

1.2.1:

There should be a junction/camera selection which can be configured by the traffic police/ concerned authorities.

1.3:

Traffic violation detection should be available for monitoring by the client.

1.3.1:

They should be able to choose a particular junction/camera feed to monitor

3.3.2:

Client should be able to feed in custom video data into the system

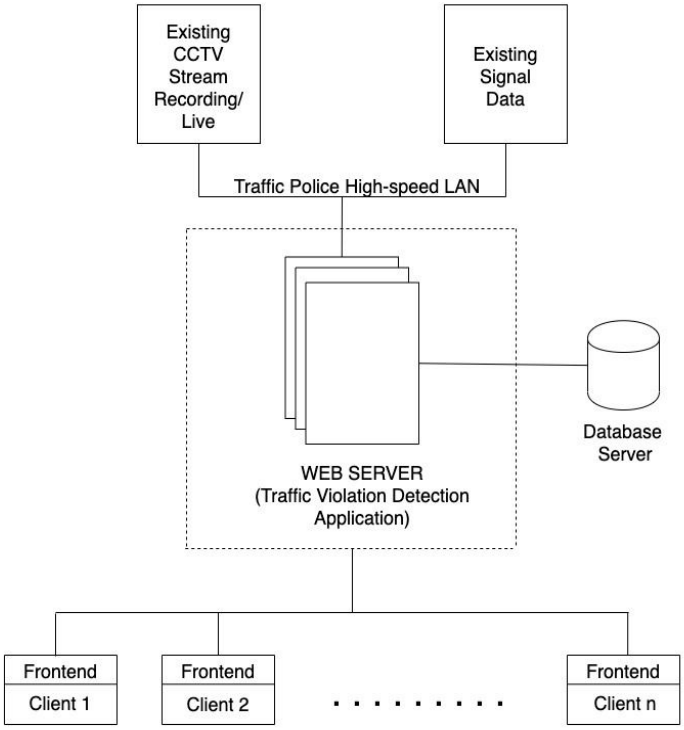
3.3.3

Live processed output should be available if the user demands for the same (GUI to be built for the same)

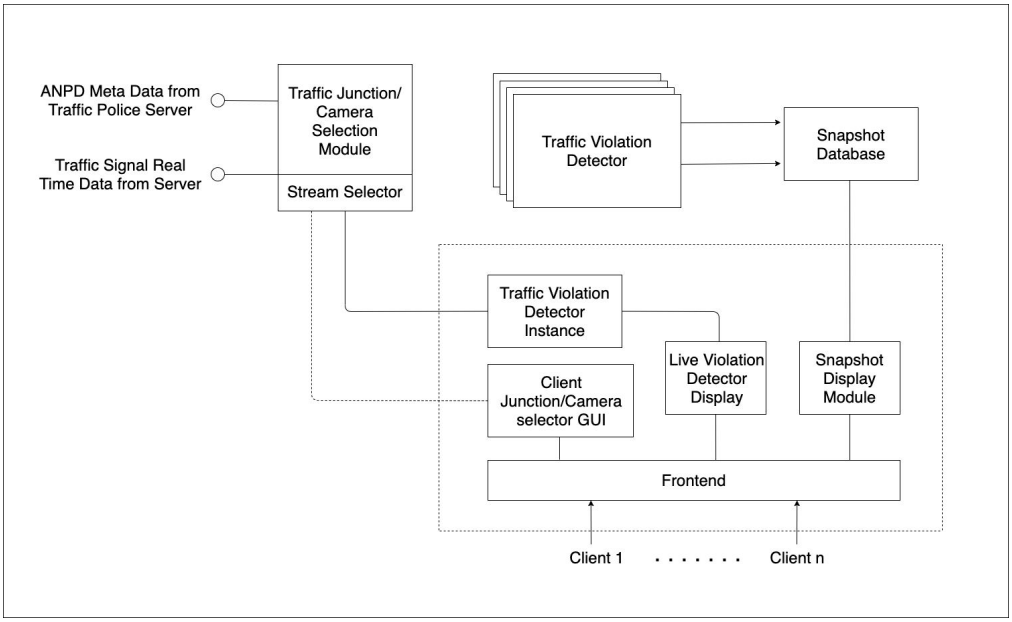
3.4:

Traffic violations/car crashes should be alarmed on the monitoring displays in the control room.

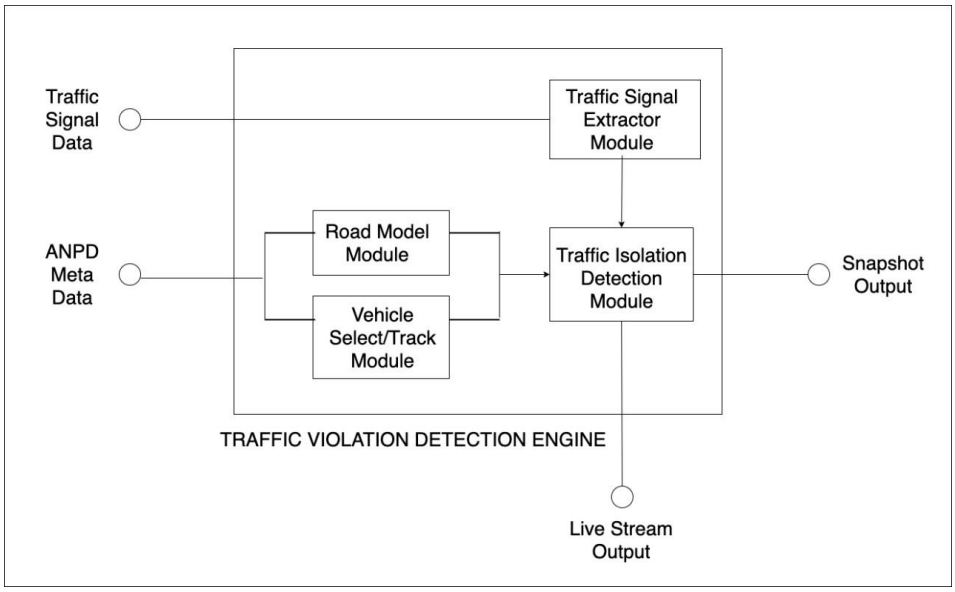
# 2. Design:




**Overall high level architecture**




**Software Architecture**




**Architecture of Traffic Violation Detection Engine**

# 3. Assumptions:

1. Availability of a camera stream/pre recorded data and live streams are connected to the police server. **(NOTE: For this project, we will be implementing a Minimum Viable Product(MVP) only the video upload and processing, since we don't have enough hardware and time to learn and implement the product for a CCTV scenario)**
2. In the case that the vehicle details should be detected, we assume that it is done by a separate system which performs automatic number plate detection, and extracts vehicle registration data from an already existing server (this is not in our project scope).
3. Traffic light data is available from all the junctions required as real-time values.

# 4. Design Rationale:

The design is based on a distributed architecture, since traffic violation detection is a resource demanding application; Not because of the algorithms involved, but due to the volume of data to be processed. The amount of data processed depends on the following factors:

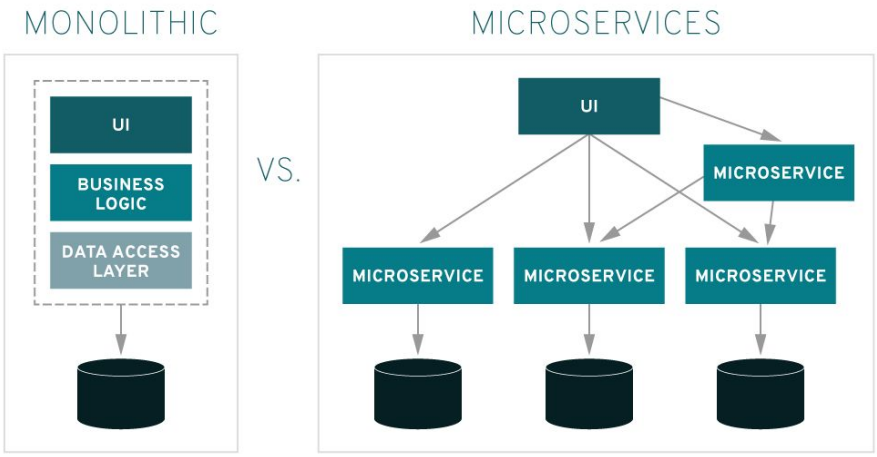
- Traffic camera/junctions to be monitored simultaneously.
- The amount of data processed depends on the day, time of the day etc., due to various traffic behaviors.

Because of this, we need a system that automatically scales up as well as down as per the requirement. Also, the system should support live monitoring by a number of officers (clients), which may also increase due to certain events happening in the city. Considering all these the distributed architecture is selected.

When we go for distributed architecture, the next question is: “What is the basic architecture like?”. The nodes are not geographically distributed, and the network between the nodes are high speed LAN and are very reliable. Hence, the network partition is not considered. So, only node failure is considered. Hence we need a CA system.

# 5. Application Architecture:

1. Instead of having a monolithic architecture with layers stacked on one another, we are designing our application based on microservices architecture. Monolithic architecture only allows for vertical scaling, but with microservices architecture it's possible to achieve horizontal scaling easily.




2. Numerous benefits of Microservices architecture include :

    a. Modularity : This makes the application easier to understand, develop, test, and become more resilient to architecture erosion.

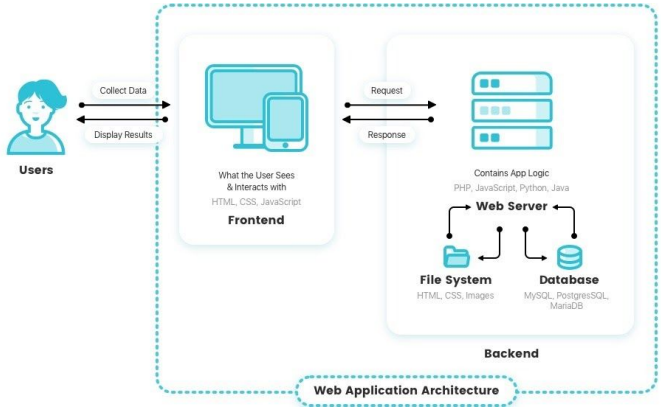
    b. Scalability : Since microservices are implemented and deployed independently of each other, i.e. they run within independent processes, they can be monitored and scaled independently.

c. Distributed development: it parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.

3. First option with microservices is to have a VM for each microservice. For eg, if our app has three microservices : frontend, backend and db then each microservice will run on a seperate VM. However, a virtual machine has a substantial overhead, like performance is poorer, it consumes a lot of memory in RAM & it occupies a lot of hard disk space. The idea solution would be a lightweight alternative to virtualization which brings us to docker.

   Hence for our application, each microservice will be containerized with docker. Instead of having the entire operating system within a docker container, docker makes sure there's only enough kernel pieces to run the corresponding microservice.

4. Our web application will have seperate microservices for front end, database and each of the features of the backend. For example, for detecting overspeeding vehicles in a video, we will have a seperate microservice running which will take input as video and return output as snapshots of overspeeding cars.




5. While deploying the web application, it's necessary to have container orchestration. Container orchestration is all about managing the life cycles of containers, especially in large, dynamic environments. Software teams use container orchestration to control and automate many tasks including provisioning and deployment of containers, redundancy and availability of containers, Scaling up or removing containers to spread application load evenly across host infrastructure, Movement of containers from one host to another if there is a shortage of resources in a host, or if a host dies, Allocation of resources between containers, etc.

# 6. Tools :

## 1. Front End:

a. We chose Angular as the front end framework.

b. Angular is considered a full MVC framework because it offers strong opinions as to how your application should be structured. It also has much more functionality “out-of-the-box”. You don’t need to decide which routing libraries to use or other such considerations.

## 2. Back End:

We chose Flask as the back end framework, since it provides simplicity, flexibility and fine-grained control. It is un-opinionated (it lets you decide how you want to implement things).

## 3. Database:

PostgresSQL was chosen for of its scalability.

## 4. Object Detection:

**YOLO** : You only look once (YOLO) is a state-of-the-art, real-time object detection system.

## 5. Container Orchestration:

**Kubernetes** is considered as the gold standard for container orchestration. Once applications are developed using docker images, it will be easy to deploy those containers on Kubernetes, since it provides fault tolerance, scalability like no other platform. One of the amazing things about Kubernetes is that we can add new features to the application with zero downtime deployment which is really amazing.

**Docker swarm** is also another container orchestration platform. But Docker swarm has limited functionality and fault tolerance. In docker swarm, services need to be scaled manually, while in Kubernetes this can be easily automated.

## 7. Use Case Diagram :