Universitatea Tehnică a Moldovei

SISTEM INTELIGENT DE CONTROL AL TRAFICULUI SEMAFORIZAT

Student: **Popa Eugeniu**
gr. FAF-202

Coordonator: **Cojuhari Irina**
conf. univ., dr.

Chişinău, 2024

MINISTERUL EDUCAŢIEI, CULTURII ŞI CERCETĂRII
Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică şi Microelectronică
Departamentul Ingineria Software și Automatică

Admis la susţinere
Şef de departament:
Fiodorov Ion dr., conf. univ.

„___” _____ 2024

SISTEM INTELIGENT DE CONTROL AL TRAFICULUI SEMAFORIZAT

INTELLIGENT TRAFFIC LIGHT CONTROL SYSTEM

Proiect de licenţă

Student: _____ **Popa Eugeniu, FAF-202**

Coordonator: _____ **Cojuhari Irina, conf. univ., dr.**

Consultant: _____ **Gavrilița Mihail, asist. univ.**

Consultant: _____ **Catruc Mariana, lect. univ.**

Chişinău, 2024

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

Departamentul Ingineria Software şi Automatică

Programul de studii Ingineria Software

Aprob
Şef de departament:
Fiodorov Ion dr., conf. univ.

_____

"___" _____ ____

CAIET DE SARCINI

pentru proiectul de licenţă al studentului

Popa Eugeniu

1. Tema proiectului de licenţă: **Sistem inteligent de control al traficului semaforizat**
confirmată prin hotărârea Consiliului facultăţii nr. 2 ***din*** *„*31*"* *octombrie* 2023

2. Termenul limită de prezentare a proiectului de licență 20.05.2024

3. Date iniţiale pentru elaborarea proiectului de licență *Elaborarea unui sistem inteligent de monitorizare, analiză și regulare a traficului.*

4. Conţinutul memoriului explicativ

Introducere
1 Analiza domeniului de studiu
2 Modelarea și proiectarea sistemului
3 Implementarea sistemului
4 Documentarea produsului realizat
5 Managementul Proiectului
Concluzii
Bibliografie

5. Conţinutul părţii grafice a proiectului de licență: imagini, figuri grafice.

6. Lista consultanţilor:

Consultant	Capitol	Confirmarea realizării activităţii	
		Semnătura consultantului (data)	Semnătura studentului
Gavrilița Mihail	Standarde tehnologice		
Catruc Mariana	Controlul calității		

7. Data înmânării caietului de sarcini 15.09.2024

Coordonator *Cojuhari Irina* _____
semnătura

Sarcina a fost luată pentru a fi executată de către studentul Popa Eugeniu

_____
semnătura, data

PLAN CALENDARISTIC

Nr. crt.	Denumirea etapelor de proiectare	Termenul de realizare a etapelor	Nota
1	*Elaborarea sarcinii, primirea datelor pentru sarcină*	15.09.23 – 01.03.24	5%
2	*Analiza domeniului de studiu*	04.03.24 - 22.03.24	15%
3	*Proiectarea sistemului*	25.03.24 – 05.04.24	15%
4	*Realizarea sistemului*	08.04.24 – 22.04.24	35%
5	*Descrierea sistemului*	29.04.24 – 03.05.24	10%
6	*Estimarea costurilor sistemului*	06.05.24 – 10.05.24	15%
7	*Finisarea proiectului*	13.05.24 – 17.05.24	5%

Student *Popa Eugeniu* ( )

Coordonator de proiect de licență *Cojuhari Irina* ( )

UNIVERSITATEA TEHNICĂ A MOLDOVEI

FACULTATEA CALCULATOARE, INFORMATICĂ ŞI MICROELECTRONICĂ
DEPARTAMENTUL INGINERIA SOFTWARE ŞI AUTOMATICĂ
PROGRAMUL DE STUDII INGINERIA SOFTWARE

AVIZ
la proiectul de licenţă

Titlul: Sistem inteligent de control al traficului semaforizat

Studentul Popa Eugeniu, gr. FAF-202

1.Actualitatea temei: Gestionarea traficului urban se confruntă cu provocări semnificative din cauza urbanizării crescute și a sistemelor de trafic depășite. Sistemele tradiționale de gestionare a traficului se bazează adesea pe date statice și nu au implementate capacități predictive, deaceia realizarea unui sistem de control inteligent al traficului semaforizat reprezintă o temă de cercetare importantă.

2. Caracteristica proiectului de licenţă: în teza de licență se propune un Sistem Inteligent de Control al Semaforizării folosind o arhitectură de microservicii. În baza datelor obținute în timp real se realizează o analiză și reconfigurarea algoritmului de funcționare a semafoarelor din intersecția.

3. Analiza prototipului: s-au analizat diverse tipuri de sisteme inteligente de control al traficului semaforizat.

4. Estimarea rezultatelor obţinute: sistemul realizat constă din două servicii principale: un serviciu de analiză și un serviciu de reglementare, iar aplicația mobilă realizată oferă informații despre traficul în timp real.

5. Corectitudinea materialului expus: materialul expus corespunde în totalmente standardelor.

6. Calitatea materialului grafic: materialul grafic corespunde standardelor.

7. Valoarea practică a proiectului: Sistemul constă din două servicii principale: un serviciu de analiză și un serviciu de reglementare. Serviciul de analiză colectează și analizează date de trafic în timp real de la multiple intersecții. Serviciul de reglementare ajustează timpii semafoarelor pe baza datelor în timp real și a algoritmilor de predicție.

8. Observaţii şi recomandări: observații nu sunt, ca recomandare este de a face implementarea hardware pe baza automatelor logic programabile PLC.

9. Caracteristica studentului şi titlul conferit : studentul **Popa Eugeniu** este un student muncitor, responsabil, stăruitor, astfel recomand pe studentul **Popa Eugeniu** pentru a obține calificativul de inginer licențiat și propun să fie apreciată lucrarea cu nota 10 (zece).

Lucrarea în forma electronică corespunde originalului prezentat către susținere publică.

Coordonatorul proiectului de licenţă _____Cojuhari Irina, conf. univ., dr.

Abstract

Gestionarea traficului urban se confruntă cu provocări semnificative din cauza urbanizării crescânde și a sistemelor de trafic depășite. Sistemele tradiționale de gestionare a traficului se bazează adesea pe date statice și lipsesc capacități predictive, rezultând într-un flux de trafic ineficient, congestii și emisii mai mari. Această lucrare de licență propune un Sistem Inteligent de Control al Semaforizării folosind o arhitectură de microservicii pentru a aborda aceste probleme prin utilizarea datelor în timp real, analizelor avansate și modelării predictive.

Sistemul constă din două servicii principale: un serviciu de analiză și un serviciu de reglementare. Serviciul de analiză colectează și analizează date de trafic în timp real de la multiple intersecții, generând analize de trafic și modele predictive folosind algoritmi de inteligență artificială. Serviciul de reglementare ajustează timpii semafoarelor pe baza datelor în timp real și a predicțiilor, asigurând un flux de trafic mai fluid și reducând congestiile. O aplicație mobilă oferă informații despre traficul în timp real și analize utilizatorilor și ofițerilor de poliție rutieră, în timp ce camerele de trafic echipate cu tehnologie de detectare a obiectelor colectează date de trafic precise.

Această soluție oferă eficiență îmbunătățită a traficului, reducerea congestiilor și siguranță rutieră sporită prin arhitectura sa modulară și scalabilă.

Teza începe cu o Analiză a Domeniului, explorând provocările actuale în gestionarea traficului urban și limitările sistemelor tradiționale. Acest capitol stabilește necesitatea unor sisteme avansate de gestionare a traficului care utilizează date în timp real și analize predictive.

În Modelarea și Proiectarea Sistemului, teza detaliază proiectarea Sistemului Inteligent de Control al Semaforizării propus, folosind o arhitectură de microservicii. Acest capitol se concentrează pe rolurile serviciilor de analiză și reglementare și pe integrarea lor pentru gestionarea optimă a traficului.

Implementarea Sistemului descrie implementarea tehnică a Sistemului Inteligent de Control al Semaforizării. Aceasta include desfășurarea serviciilor, integrarea camerelor de trafic și dezvoltarea aplicației mobile, împreună cu abordarea provocărilor practice și soluțiilor întâmpinate în timpul procesului de implementare.

Documentația Produsului oferă documentație cuprinzătoare pentru sistem. Include manuale de utilizare, specificații tehnice și ghiduri de întreținere pentru a asigura utilizarea și gestionarea eficientă a sistemului.

În final, Viziunea asupra Managementului Proiectului prezintă perspectiva managementului proiectului. Acest capitol acoperă domeniul de aplicare al proiectului, cronologia, alocarea resurselor și implicarea părților interesate, împreună cu lecțiile învățate și bunele practici pentru proiectele viitoare.

Abstract

Urban traffic management faces significant challenges due to increasing urbanization and outdated traffic systems. Traditional traffic management systems often rely on static data and lack predictive capabilities, resulting in inefficient traffic flow, congestion, and higher emissions. This bachelor thesis proposes an Intelligent Traffic Light Control System utilizing a microservices architecture to address these issues through real-time data, advanced analytics, and predictive modeling.

The system consists of two main services: an analytics service and a regulation service. The analytics service collects and analyzes real-time traffic data from multiple intersections, generating traffic analytics and predictive models using AI algorithms. The regulation service adjusts traffic light timings based on real-time data and predictions, ensuring smoother traffic flow and reduced congestion. A mobile application provides real-time traffic information and analytics to users and traffic police officers, while traffic cameras with object detection gather accurate traffic data.

This solution offers improved traffic efficiency, reduced congestion, and enhanced road safety through its modular, scalable architecture.

The thesis begins with a Domain Analysis, exploring the current challenges in urban traffic management and the limitations of traditional systems. This chapter establishes the need for advanced traffic management systems that utilize real-time data and predictive analytics.

In System Modeling and Design, the thesis details the design of the proposed Intelligent Traffic Light Control System using a microservices architecture. This chapter focuses on the roles of the analytics and regulation services and their integration for optimal traffic management.

System Implementation describes the technical implementation of the Intelligent Traffic Light Control System. This includes the deployment of services, integration of traffic cameras, and development of the mobile application, along with addressing practical challenges and solutions encountered during the implementation process.

Product Documentation provides comprehensive documentation for the system. It includes user manuals, technical specifications, and maintenance guidelines to ensure effective use and management of the system.

Finally, the Project Management View presents the project management perspective. This chapter covers the project's scope, timeline, resource allocation, and stakeholder involvement, along with lessons learned and best practices for future projects.

Table of Contents

List of Abbreviations and Definitions

ITLCS - Intelligent Traffic Light Control System

GDP - Gross Domestic Product

SCATS - Sydney Coordinated Adaptive Traffic System

ITS - Intelligent Transport System

BATCS - Bucharest's Adaptive Traffic Control System

UTC - Urban Traffic Control

PTM - Public Transport Management

ATSC - Adaptive Traffic Signal Control

CAV - Connected and Autonomous Vehicles

SOA - Service Oriented Architecture

API - Application Programming Interface

PLC - Programmable Logic Controller

I/O - Input/Output

UART - Universal Asynchronous Receiver / Transmitter

SPI - Serial Peripheral Interface

I2C - Inter-Integrated Circuit

PC - Personal Computer

UML - Unified Modeling Language

TMS - Traffic Monitoring Service

TAS - Traffic Analytics Service

TRS - Traffic Regulation Service

MA - Mobile Application

DB - Database

YOLO - You Only Look Once

SORT - Simple Online Realtime Tracking

KPI - Key Performance Indicator

ROI - Return of Investment

RSI - Requirements Stability Index

INTRODUCTION

The rapid growth of urbanization and population expansion has caused unprecedented challenges in managing traffic congestion, ensuring road safety, and promoting sustainable urban mobility. As cities continue to expand, the demand for efficient transportation systems grows, becoming increasingly necessary in maintaining residents' quality of life and supporting economic development. However, traditional traffic management systems, although effective to some extent, often struggle to keep pace with the dynamic nature of urban traffic patterns.

In response to these complex challenges, Intelligent Traffic Management Systems have emerged as a promising solution. By leveraging innovative technologies such as real-time data analytics, predictive modeling, and adaptive control algorithms, ITLCS offer the potential to not only optimize traffic flow but also to reduce congestion and improve road safety in urban environments.

This thesis begins on an ambitious journey to design, implement, and evaluate an innovative ITLCS specially customized to meet the unique demands of modern urban landscapes. Drawing inspiration from principles found in industrial automation and software engineering, the proposed solution seeks to revolutionize urban traffic management by introducing a scalable, adaptable, and data-driven approach that can evolve alongside the continuously changing needs of urban environments.

The primary objective of this thesis is to design, develop, and evaluate an ITLCS that utilizes advanced technologies to not only optimize traffic flow and alleviate congestion but also to significantly improve road safety in urban settings. Through comprehensive analysis, comparative studies, and practical implementation, this thesis aspires to make a substantial contribution to the advancement of intelligent transportation systems, ultimately promoting the creation of smarter, more sustainable cities for generations to come.

1 DOMAIN ANALYSIS

1.1 Introduction to Traditional Traffic Light Systems

Traffic lights, also known as traffic signals or stoplights, have been an integral part of urban transportation infrastructure since their inception in the late 19th century. The first manually operated gas-lit traffic signal was installed in London in 1868 to control horse-drawn carriage traffic. However, it wasn't until the early 20th century that electric traffic signals began to replace their gas-lit predecessors, offering more reliable and efficient operation [1].

They are the fundamental components of urban transportation infrastructure designed to regulate the flow of vehicular and pedestrian traffic at intersections. The conventional traffic light system typically consists of a set of signal heads mounted on poles or overhead structures, each equipped with colored lights that indicate when vehicles and pedestrians should stop or proceed.

Within the traditional traffic light system, essential components include the signal heads, controller box, timer and sequencer, and sensors.

The primary components of a traffic light system are the signal heads, which contain colored lights, typically red, yellow/amber, and green-arranged in a vertical or horizontal configuration. These lights serve as visual indicators to drivers and pedestrians, signaling when they should stop, prepare to stop, or proceed with caution.

The controller box, often located at the intersection or nearby, houses the electronic control unit responsible for coordinating the timing and sequencing of the traffic signals. It receives inputs from various sensors, timers, and switches to determine the appropriate signal phases based on traffic conditions.

Within the controller box, a timer and sequencer dictate the duration and sequence of each signal phase. These parameters are typically pre-programmed based on anticipated traffic patterns, with fixed time intervals allocated for green, yellow, and red phases.

Some traditional traffic light systems may incorporate sensors embedded in the road surface or mounted above the intersection to detect the presence of vehicles or pedestrians. These sensors provide real-time data to the controller, allowing it to adjust signal timings dynamically based on traffic demand.

The operation of a traditional traffic light system follows a predetermined sequence of signal phases to facilitate the orderly movement of traffic through the intersection.

During the green phase, vehicles traveling in the direction controlled by the green signal are permitted to proceed through the intersection. Pedestrians may also be granted the right of way to cross the street, depending on the configuration of pedestrian signals.

The yellow or amber phase serves as a transition period between the green and red phases, signaling to drivers and pedestrians that the signal is about to change. Drivers are expected to prepare to stop unless it is unsafe to do so.

In the red phase, vehicles and pedestrians are required to stop. Traffic in the direction controlled by the red signal must yield to cross traffic or pedestrians with the right of way.

While traditional traffic light systems have been effective in managing traffic flow for decades, they do have limitations, particularly in handling dynamic traffic conditions:

Traditional traffic lights operate on fixed timing schedules, which may not adequately accommodate fluctuations in traffic demand throughout the day.

These systems typically lack the ability to adjust signal timings in real-time based on current traffic conditions, leading to potential inefficiencies and congestion, especially during peak hours.

Changes to signal timings often require manual adjustments to the controller settings, making it challenging to respond quickly to changing traffic patterns or unexpected events.

While traditional traffic light systems remain prevalent, advancements in technology have led to the development of intelligent traffic light control systems, which offer increase functionality and adaptability to address the evolving challenges of urban traffic management.

1.2 Importance and Relevance of Intelligent Traffic Light Control Systems

Efficiently managing urban traffic has become an increasingly critical concern for urban planners, policymakers, and technology experts as cities rapidly expand and technology advances. This issue is made worse by increasing urban populations, leading to more road congestion, higher vehicle emissions, and a greater need for improved road safety measures. In response to these complex issues, intelligent traffic light control systems are emerging as transformative solutions that hold the potential to revolutionize how cities approach urban mobility, sustainability, and safety.

Cities worldwide are struggling with the surge in vehicular traffic, causing roadways to become congested, resulting in significant economic losses, reduced quality of life, and increased environmental damage. Traditional traffic lights, which operate on fixed cycles, can't keep up with the changing patterns of urban traffic flow, creating inefficiencies that worsen congestion and pollution. In the EU, congestion costs are around €110 billion annually, or about 1% of its GDP [2]. In the United States, traffic congestion and related delays cost the economy over $120 billion annually [3]. A 2022 study revealed that traffic congestion in the U.S. led to the average driver spending 51 hours in traffic, which translates to about $869 in lost time and increased pollution per person [4].

Intelligent traffic light control systems are a game-changer in solving these challenges. By using real-time traffic data, ITLCS can dynamically adjust signal timings to improve traffic flow and significantly reduce congestion. This flexibility helps keep traffic moving smoothly through intersections, cutting down on delays and preventing congestion buildup. For example, Singapore's Intelligent Transport System has been praised for reducing travel time by up to 20% during peak hours [5].

Beyond congestion management, ITLCS plays a vital role in improving road safety. Over half of all traffic crashes happen at or near intersections. ITLCS can lower this statistic by adjusting signal timings

based on real-time conditions, reducing the chance of collisions and discouraging erratic driving behaviors linked to congested roads. Additionally, ITLCS can dramatically cut vehicle emissions by reducing stop-and-go traffic, contributing to cleaner air and a healthier urban environment.

Traffic congestion is more than just an an inconvenience; it's an economic drain. The cost of congestion, including wasted time, increased fuel consumption, and higher transportation costs, costs major cities billions annually. ITLCS can help reduce these costs by making traffic flow more efficient, proving to be a wise investment for cities looking to improve their economic health and competitiveness.

Integrating Intelligent Traffic Light Control Systems into urban infrastructure is a critical aspect of the broader smart city vision. In smart cities, the strategic use of technology and data aims to elevate various aspects of urban life, including infrastructure, services, and overall quality of life. Among these advancements, ITLCS is fundamental to smart transportation networks. By effectively managing traffic flow and optimizing signal timings based on real-time data, these systems exemplify how technology can manage complex urban challenges head-on, ultimately leading to more efficient and livable cities.

As cities continue to expand and technology progresses, the importance of ITLCS will only grow. The rise of autonomous vehicles, the growth of smart city infrastructure, and a stronger focus on sustainability all highlight a future where intelligent traffic management systems like ITLCS are crucial.

In conclusion, intelligent traffic light control systems are vital for addressing the urgent issues of traffic congestion, road safety, environmental sustainability, and economic efficiency. As we look ahead, the ongoing development and deployment of ITLCS will be key in creating the livable, resilient cities of the future. Your proposed solution, with its innovative features and capabilities, exemplifies the creative drive pushing forward in this field.

1.3 Existing and Similar Solutions: A Comparative Analysis

Urban traffic management presents a significant challenge for cities worldwide, leading to the development and deployment of Intelligent Traffic Light Control Systems. Regarding existing solutions, there are several existing ITLCS and comparable systems, which leverage advanced technologies to adjust traffic signals in real-time, aiming to improve traffic flow, alleviate congestion, and enhance road safety.

SCATS, developed by Australia's Roads and Maritime Services, stands as a paradigm of centralized ITLCS. Operating on a centralized control approach, SCATS utilizes data collected from sensors embedded in roadways to dynamically adjust traffic signal timings. This adaptive system analyzes traffic flow patterns in real-time, allowing for optimal signal coordination and efficient traffic management. SCATS has been deployed globally, including cities like Sydney, Melbourne, and Kuala Lumpur, where it has demonstrated significant reductions in travel time, congestion, and emissions. In SCATS, traffic signal timings are adjusted based on data from sensors installed in roads (see Figure 1.1).

These sensors continuously monitor traffic flow, feeding information to a centralized control center. Algorithms analyze this data in real-time to determine the optimal timing for traffic signals at intersections.

SCATS can adapt to changing traffic conditions, dynamically adjusting signal timings to minimize congestion and maximize traffic flow.

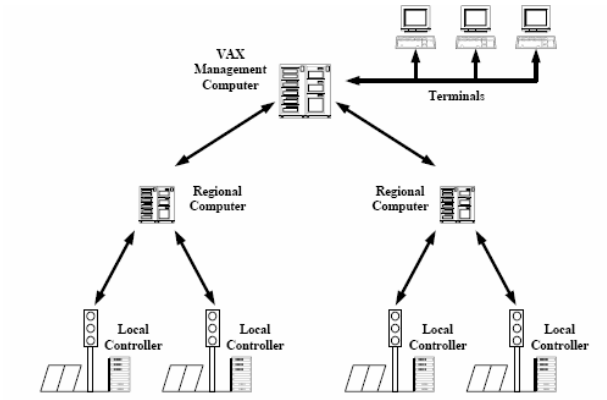

Figure 1.1 – SCATS physical architecture [6]

Surtrac, developed by Traffic21 at Carnegie Mellon University, presents a decentralized approach to traffic management. As Figure 1.2 indicates, this adaptive traffic control system enables individual traffic signals to communicate with neighboring intersections, facilitating coordinated signal timings.

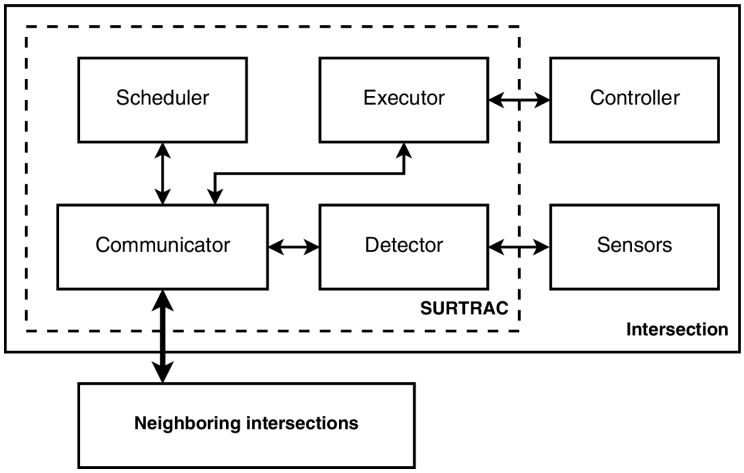

Figure 1.2 – SURTRAC system diagram [7]

By leveraging machine learning algorithms, Surtrac dynamically adjusts signal timings based on real-time traffic data, effectively minimizing delays and congestion. Real-world implementations in cities such as Pittsburgh and Atlanta have yielded impressive results, with studies indicating up to 40% reduction in travel time and significant improvements in overall traffic efficiency. Surtrac employs a decentralized approach where individual traffic signals communicate with each other through a network. Using real-time traffic data, Surtrac's machine learning algorithms predict traffic patterns and adjust signal timings accordingly. By coordinating signals across intersections, Surtrac optimizes traffic flow, reducing delays and improving overall efficiency.

Singapore's ITS showcases a comprehensive city-wide implementation of ITLCS. With a growing population, Singapore has anticipated the future transport challenges and through innovative use of

technology and policy decisions has planned to ensure that the small city state meets the needs of the population and that the infrastructure is fit for purpose. Utilizing a network of sensors, cameras, and traffic monitoring systems, Singapore collects detailed real-time traffic data, enabling its advanced traffic management system to dynamically adjust signal timings. The impact of Singapore's ITS is substantial, with studies indicating significant reductions in travel time, congestion, and emissions, underscoring its role in fostering sustainable urban mobility. As Figure 1.3 illustrates, Singapore's ITS utilizes a vast network of sensors and cameras installed throughout the city to collect real-time traffic data. This data is analyzed by sophisticated algorithms, which adjust signal timings at intersections based on current traffic conditions. By dynamically optimizing signal timings, ITS improves traffic flow, reduces congestion, and enhances overall road safety.

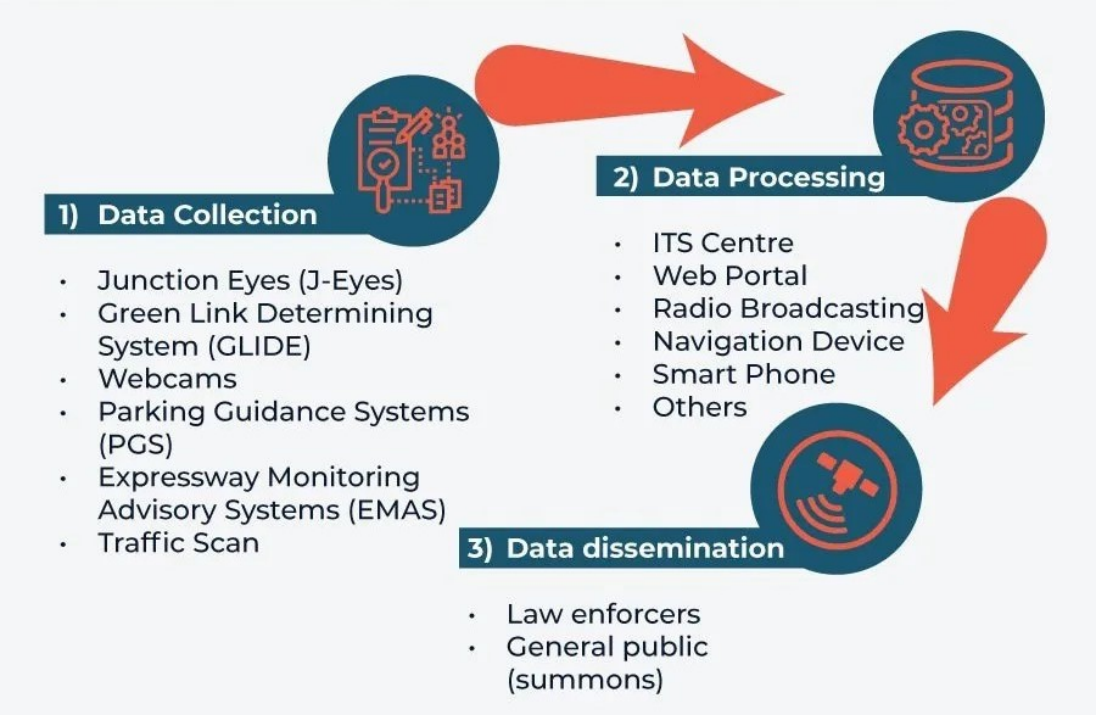

Figure 1.3 – Real time traffic information within the Intelligent Transport System (ITS) [8]

BATCS represents a sophisticated approach to addressing urban traffic congestion, aligning with the city's commitment to leveraging technology for improved urban mobility. The system operates based on a hybrid model that combines features of both centralized and decentralized systems, allowing for localized adjustments at individual intersections within a framework that aligns with broader traffic management objectives set by a central control system.

BATCS employs a sophisticated algorithm to adjust traffic signals in real-time, relying on a mix of sensor data such as flow rates and vehicle occupancy. This data is collected from various sources, including sensors embedded in roadways, cameras, and other monitoring systems. By analyzing real-time traffic conditions, BATCS optimizes traffic light cycles to improve traffic flow and reduce congestion.

The system's hybrid model ensures that traffic optimization at one intersection does not inadvertently cause bottlenecks or inefficiencies at another. It allows for seamless coordination between different intersections while maintaining adaptability to localized traffic conditions. Implemented initially on the city's main ring road and the North-South axis, BATCS has expanded to incorporate more intersections, currently encompassing 172 signalized intersections in Bucharest [9] . The municipality's intention is to integrate all signalized intersections into the system eventually.

BATCS also integrates with other subsystems to enhance its effectiveness. For example, as Figure 1.4 shows, the Subsystem of UTC operates adaptively, allowing real-time coordination of traffic signal timings based on traffic flow patterns.

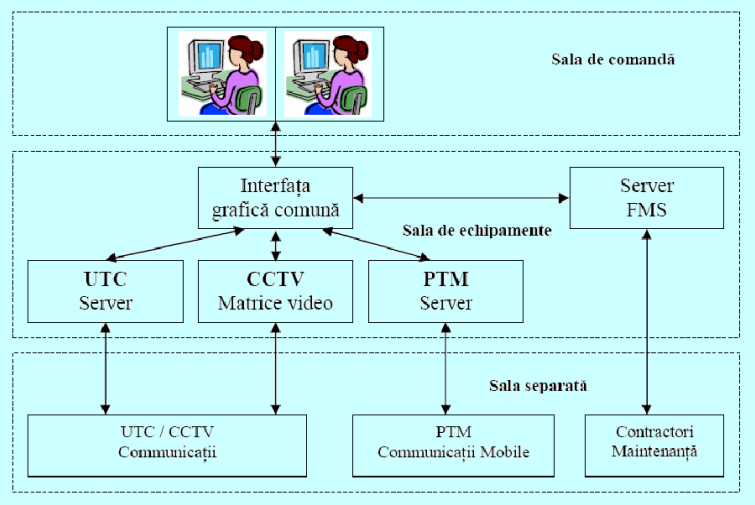

Figure 1.4 – BATCS system architecture [9]

Additionally, the Subsystem of PTM prioritizes public transport vehicles by changing signal colors to green as they approach intersections, reducing delays and improving adherence to schedules (see Figure 1.5).

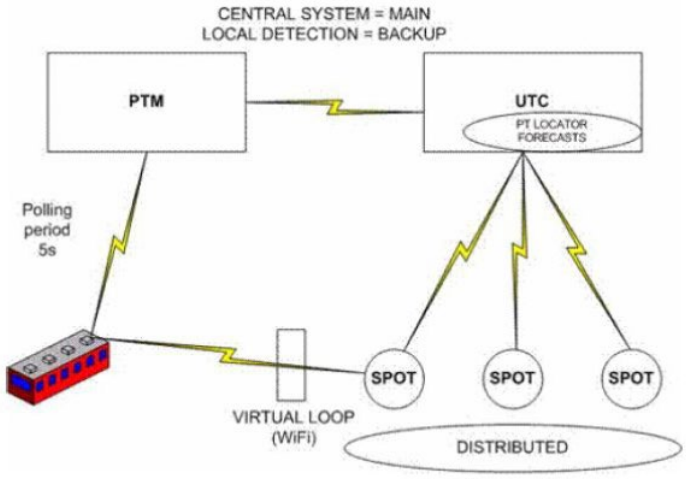

Figure 1.5 – Interconnection and collaboration of subsystems management of traffic and urban transport [9]

The system's integration and collaboration between traffic management and public transport subsystems contribute to its overall efficacy in mitigating traffic congestion and improving urban mobility in Bucharest. Initial reports indicate significant improvements in travel times and reductions in vehicular queue lengths across the city, highlighting the effectiveness of BATCS in urban traffic management.

The city of Chișinău has implemented intelligent traffic light control systems at some intersections to optimize traffic flow and enhance road safety [10]. The system comprises two main elements: the electronic control and command unit and sensors in the form of video cameras for acquiring signals from each axis of the intersection. These cameras continuously monitor traffic flows and transmit information to the traffic lights, which adjust signal timings accordingly, granting priority to vehicles on main axes and expediting green light allocation based on real-time traffic demands. The system's flexibility allows for dynamic adjustments to accommodate varying traffic volumes, minimizing delays and enhancing overall traffic efficiency. The system has effectively eliminated dangerous maneuvers from, ensuring safer intersections. Additionally, by dynamically adjusting signal timings based on traffic fluctuations on each arm of the intersection, the system has reduced overall delays. The implemented system tracks traffic flows on each lane group of every intersection arm, enabling precise optimization of signal timings compared to previous fixed timing plans. Additionally, it facilitates real-time traffic flow monitoring and vehicle type identification (e.g., cars, buses, trucks), aiding in the planning and development of future transportation systems in Chișinău. Integration and Control: The system offers integration potential into a centralized command and control system, enabling adjustment of signal timings based on sensor-detected traffic values transmitted to the traffic control center. Specialized software analyzes collected data to determine optimal signal timings, ensuring continuous vehicle flow and preventing congestion on priority traffic routes throughout the day. The system utilizes video cameras as sensors for real-time and recorded traffic monitoring, aiding in incident investigation and resolution by capturing detailed road conditions.

Table 1.1 provides a comparative analysis of five Intelligent Traffic Light Control Systems: SCATS, SURTRAC, Singapore's ITS, BATCS, and Chisinau's system, across various criteria including deployment locations, cost, scalability, machine learning integration, infrastructure requirements, advantages, and disadvantages.

SCATS is deployed globally, while SURTRAC is implemented in Pittsburgh and Atlanta, Singapore's ITS is located in Singapore, BATCS operates in Bucharest, and Chisinau's system is implemented in Chisinau. In terms of cost, SCATS and SURTRAC both incur high costs, whereas Singapore's ITS requires a very high investment. BATCS and Chisinau's system show moderate costs. Regarding scalability, SURTRAC, Singapore's ITS, BATCS, and Chisinau's system exhibit high scalability compared to SCATS, which shows moderate scalability.

SURTRAC integrates machine learning, offering an advantage over SCATS and Singapore's ITS, which have limited machine learning integration. Additionally, SURTRAC's decentralized approach allows

for local optimization, providing a distinct advantage over SCATS and Singapore's ITS. BATCS and Chisinau's system offer flexibility and adaptability to local traffic conditions, similar to SURTRAC. However, Singapore's ITS provides detailed real-time traffic data for precise adjustments, which may be advantageous over other systems.

In terms of disadvantages, SCATS and SURTRAC both face moderate scalability issues, while Singapore's ITS suffers from a high cost of implementation. BATCS and Chisinau's system have their own challenges, such as limited historical data utilization and the need for careful calibration to ensure system-wide efficiency.

Table 1.1 – Comparative analysis

Criteria	**SCATS**	**SURTRAC**	**Singapore's ITS**	**Chișinău's System**	**BATCS**
Deployment Locations	Global	Pittsburgh, Atlanta	Singapore	Chișinău	Bucharest
Cost	High	High	Very high	Moderate	Moderate
Scalability	Moderate	Moderate	High	High	High
Machine Learning Integration	Limited	Yes	Limited	Limited	Yes
Infrastructure Requirements	Requires sensors and centralized control center	Requires sensors and communication	Requires extensive sensor and camera newtork	Requires sensors and video cameras	Requires sensors, hybrid control system
Advantages	Significant reduction in travel time, congestion, emissions	Decentralized approach allows local optimization	Detailed real-time traffic data for precise adjustments	Dangerous maneuver elimination, global delay reduction	Hybrid approach balances local flexibility with centralized oversight; Cost-effective
Disadvantages	Moderate scalability	Mederate scalability	High cost for implementation	Limited historical data utilization	Requires careful calibration to ensure system-wide efficiency

While ITLCS offers innovative traffic management solutions, other systems with comparable objectives exist.

ATSC systems utilize algorithms to adjust signal timings based on real-time traffic data, although they may lack the centralized coordination and advanced features of ITLCS.

Integration of CAVs with existing traffic management systems presents opportunities for optimizing traffic flow. Through communication with traffic infrastructure, CAVs can coordinate movements to reduce delays. Pilot projects in cities like Columbus, Ohio, and San Francisco have shown promising results in enhancing traffic efficiency and safety.

In conclusion, the adoption of Intelligent Traffic Light Control Systems like SCATS, Surtrac, and Singapore's ITS illustrates the transformative potential of ITLCS in urban traffic management. By dynamically adjusting signal timings based on real-time data, these systems enhance traffic flow, alleviate

congestion, and promote road safety. As cities continue to address urban traffic challenges, ITLCS and similar systems will play a pivotal role in shaping the future of transportation.

1.4 Scope and Objectives

Following our comparison of existing traffic management systems, we present the scope and objectives of our proposed solution. This will help us address the complex challenges cities face with traffic congestion, emissions, and road safety.

The proposed solution incorporates the development and deployment of an ITLCS aimed at revolutionizing urban traffic management. It includes advanced features such as real-time traffic monitoring, predictive analytics, dynamic signal control, user-friendly mobile access, hardware integration, and scalability. By leveraging these capabilities, the ITLCS seeks to optimize traffic flow, reduce congestion, enhance road safety, and contribute to environmental sustainability in urban environments.

The objectives of the proposed solution outline specific goals that each module aims to achieve in order to fulfill the broader scope.

Traffic Monitoring Module would comprise a network of cameras deployed across intersections and roadways to monitor traffic conditions in real-time. It would collect data on vehicle counts. The main objectives of the module would be to enhance real-time monitoring of traffic conditions to provide accurate and up-to-date data for traffic management decisions and to detect traffic incidents or congestion hotspots promptly to facilitate timely interventions and adjustments.

The Traffic Analytics Module would employ advanced analytics algorithms to analyze the data collected by the Traffic Monitoring Module. It would generate insights into traffic patterns and trends. The objectives of this module would be to implement advanced analytics capabilities to analyze historical and real-time traffic data and predict future traffic flow and congestion patterns to enable proactive signal adjustments and traffic management strategies.

The Traffic Control Module would be responsible for dynamically adjusting signal timings at intersections based on the insights provided by the Traffic Monitoring and Analytics Modules. It would optimize signal timings to manage traffic flow efficiently. The main objectives would be to dynamically adjust signal timings based on real-time traffic data and predictive analytics to optimize traffic flow and reduce congestion and to minimize delays and queue lengths at intersections by synchronizing signal timings and prioritizing traffic based on demand.

The User Application Module would constitute a user-friendly application accessible to stakeholders, including traffic authorities and simple users. It would provide real-time access to traffic information, analytics, and system controls. It’s objectives would be to provide stakeholders with user-friendly access to real-time traffic information, including current traffic conditions, congestion alerts, and suggested alternative routes and to enable traffic authorities to remotely monitor traffic flow and analyze the traffic analytics and predictions.

The Hardware Module would include physical components such as PLCs or other hardware devices installed at intersections to facilitate signal control and communication with the central system. The objectives of the module would be to integrate hardware components with the ITLCS to ensure reliable signal control and communication and to support the functionalities of the ITLCS and ensure continuous operation.

By achieving these objectives, the proposed solution aims to revolutionize urban traffic management, improve road safety, and enhance the overall quality of life in urban environments. It represents a significant step forward in leveraging technology to address the complex challenges of modern urban mobility.

1.5 Existing Software Architectures for Traffic Management Systems

In the sphere of urban traffic control, the efficacy and flexibility of the software solutions employed are pivotal in determining the overall performance and adaptability of traffic management systems.

Traditionally, monolithic architectures have dominated this landscape, offering a singular, unified software solution where all components are tightly integrated and interdependent.

In these monolithic systems, all functionalities, such as traffic monitoring, signal control, data analysis, and reporting, are bundled together within a single application. This approach simplifies initial development and deployment, as there is a centralized codebase and deployment process. However, as urban environments evolve and demand more sophisticated traffic management strategies, the limitations of monolithic architectures become increasingly apparent.

While monolithic systems can deliver reliable performance under consistent operating conditions, they often struggle with scalability and adaptability. As urban populations grow and traffic patterns evolve, these systems may face challenges in accommodating increasing data volumes and processing demands. Additionally, the tightly coupled nature of monolithic architectures makes it difficult to incorporate new technologies or methodologies without significant redevelopment and disruption to the entire system. An example of monolithic architecture is illustrated in Figure 1.6.

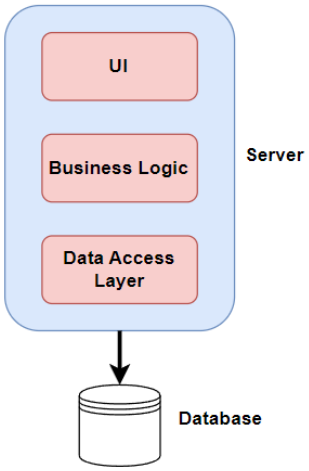

Figure 1.6 – Monolithic architecture [11]

The introduction of service-oriented architectures marked a significant evolution, emphasizing modular components that communicate over a network. This approach facilitated better scalability and the potential for more flexible system integration. However, SOAs often still require substantial overhead for communication protocols and can be challenging to manage due to the complex dependencies between services. These challenges can hinder the system's ability to rapidly adapt to changing traffic conditions or integrate innovative traffic management strategies. An example of SOA is shown in Figure 1.7.

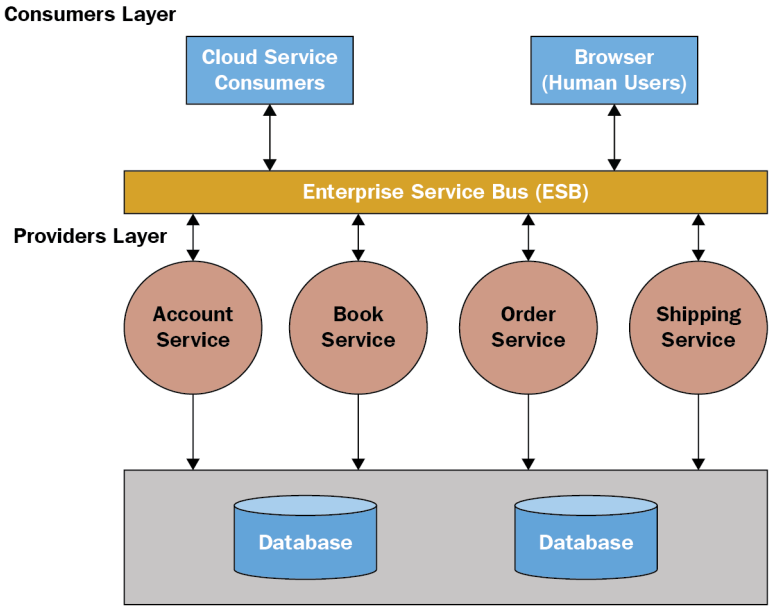

Figure 1.7 – Service-oriented architecture [12]

In contrast, the microservices architecture emerges as a highly compelling solution for traffic management systems, addressing many of the limitations inherent in monolithic and SOA frameworks.

Microservices architecture decomposes a system into a collection of small, autonomous services, each responsible for a specific function and communicating through well-defined APIs. An example of microservices architecture is shown in Figure 1.8.

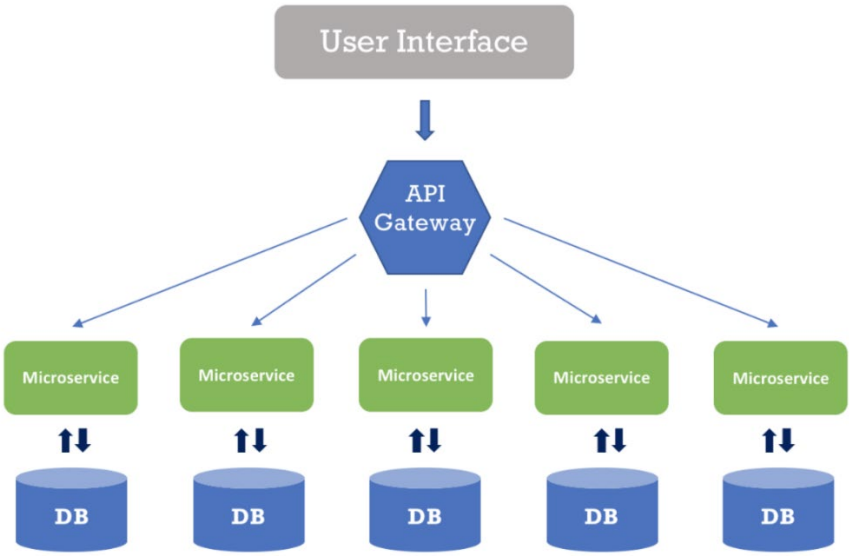

Figure 1.8 – General microservices architecture [13]

This design principle brings up a high degree of agility, allowing for the independent development, deployment, and scaling of individual services. In the context of urban traffic management, this means that specific components of the system, such as traffic monitoring, data analytics, control algorithms, and user interfaces, can evolve independently.

For example, advancements in traffic prediction algorithms can be integrated into the system with minimal impact on other services. This modular approach not only accelerates the adoption of new technologies but also significantly enhances the system's resilience. Failures in one service can be isolated, reducing the risk of system-wide failures. Moreover, microservices support dynamic scaling, where services experiencing high demand can be scaled independently to meet the need without over-provisioning resources for the entire system.

Another notable advantage of microservices lies in their facilitation of continuous integration and deployment practices. These practices enable traffic management systems to rapidly adapt to changing urban dynamics, such as the introduction of new traffic regulations, public events, or the construction of new infrastructure. By leveraging containerization technologies, microservices can be efficiently deployed across diverse computing environments, from cloud platforms to on-premise servers, offering unparalleled flexibility in how traffic management solutions are implemented and evolved.

The transition to a microservices architecture also aligns with the growing trend towards intelligent traffic management systems that leverage real-time data analytics, machine learning, and IoT technologies. By compartmentalizing functionalities, microservices allow for the seamless integration of these advanced technologies, enabling smarter, data-driven decision-making processes that can dynamically adapt to real-time conditions, thereby optimizing traffic flow and enhancing road safety.

In summary, while traditional software solutions for traffic management have provided solid foundations, the unique demands of modern urban environments necessitate a more adaptable, scalable, and resilient approach. The microservices architecture represents a paradigm shift in designing and deploying traffic management systems, offering the flexibility and efficiency required to address the complex challenges of contemporary urban traffic scenarios. This approach not only enhances the capability to manage traffic more effectively but also paves the way for integrating future advancements in traffic management technologies, ensuring that urban centers can continue to evolve towards smarter, safer, and more sustainable mobility solutions.

1.6 Existing Industrial Automation and Control Technologies

When considering the hardware implementation of ITLCS, various solutions are available, each with its advantages and limitations. In this section, we analyze existing solutions for hardware implementation, conduct a comparative analysis, and discuss why PLCs are often chosen for ITLCS.

Microcontrollers are compact, low-cost devices with embedded processors, memory, and I/O capabilities. They offer flexibility in design and programming, making them suitable for small-scale traffic

control applications. An example of a microcontroller commonly used in industrial automation and control applications, including traffic light control systems, is the Arduino platform (see Figure 1.9).

These microcontrollers feature an ATmega series microcontroller chip, along with input/output pins, analog-to-digital converters (ADCs), digital I/O ports, and communication interfaces like UART, SPI, and I2C [14].

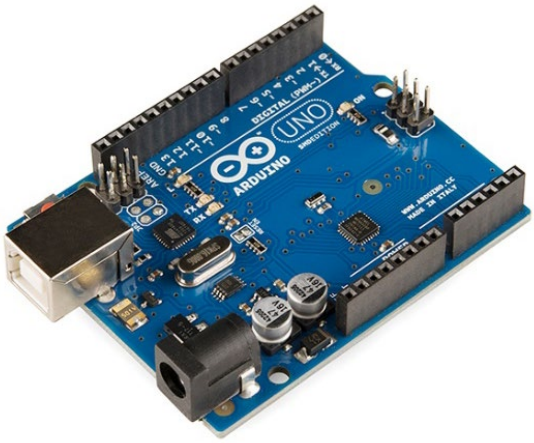

Figure 1.9 – Arduino Uno [15]

In traffic light control systems, Arduino microcontrollers can be utilized to manage the operation of traffic lights at intersections. Engineers can program the Arduino microcontroller to read input from sensors detecting vehicle presence, pedestrian crossings, and other traffic parameters. Based on this input, the Arduino can then execute logic to control the timing and sequencing of the traffic light phases, ensuring efficient traffic flow and safety. However, they may lack the processing power and scalability required for larger and more complex intersections.

Single-board computers, such as Raspberry Pi or BeagleBone, offer more processing power and flexibility compared to microcontrollers. They can run full-fledged operating systems and support a wide range of programming languages and frameworks. An example of a single-board computer is illustrated in Figure 1.10. SBCs are suitable for mid-scale traffic control applications but may face challenges with real-time responsiveness and reliability.

Figure 1.10 – Raspberry PI [16]

Industrial PCs provide high computing power, robustness, and reliability, making them suitable for large-scale and complex traffic control systems. They offer extensive connectivity options, support for multiple I/O modules, and compatibility with industrial-grade software. However, industrial PCs can be expensive and may require additional maintenance and support.

An example of an Industrial PC commonly used in industrial automation and control applications, including traffic light control systems, is the Advantech UNO series, shown in Figure 1.11.

Figure 1.11 – Advantech UNO-2484G V2 [17]

Programmable Logic Controllers (PLCs): are specialized industrial computers designed for real-time control and automation tasks. And example of PLCs is represented in Figure 1.12.

Figure 1.12 – Siemens SIMATIC S7-1200 PLC [18]

They offer a balance of reliability, scalability, and ease of programming, making them ideal for ITLCS applications. PLCs are rugged, have deterministic processing capabilities, and support a wide range of I/O modules. They are widely used in traffic control systems due to their proven track record, robustness, and suitability for real-time operation.

Table 1.2 provides a comparative analysis of the existing solutions for hardware implementation.

Table 1.2 – Comparative analysis

Solution	Advantages	Disadvantages
Microcontrollers	Low cost, compact size, flexibility in design and programming	Limited processing power, scalability, and reliability
Single-board Computers	More processing power, flexibility, support for full-fledged OS	Challenges with real-time responsiveness and reliability
Industrial PCs	High computing power, robustness, extensive connectivity options	Expensive, may require additional maintenance and support
Programmable Logic Controllers	Reliability, scalability, real-time control, ease of programming	Limited processing power compared to some industrial PCs

In conclusion, while various hardware solutions exist for implementing ITLCS, PLCs stand out due to their reliability, scalability, real-time control capabilities, ease of programming, and proven track record in industrial applications. PLCs offer a robust foundation for building efficient and dependable traffic control systems, making them a preferred choice for ITLCS implementations.

1.7 Proposed Solution

Existing solutions in urban traffic management often face several challenges and limitations. One of the primary issues is the lack of real-time data utilization and predictive analytics. Many current systems rely on static or outdated data, leading to inneficient traffic light timings and ineffective congestion management. Moreover, centralized systems like SCATS may struggle to adapt to rapidly changing traffic conditions across multiple intersections, resulting in inefficient traffic flow and increased congestion.

Moreover, traditional ITLCS typically lack integration with analytics and predictive modeling capabilities. These systems often focus solely on adjusting traffic signal timings based on current conditions without leveraging predictive analytics to anticipate future traffic patterns. As a result, they may fail to proactively address traffic congestion or optimize traffic flow during peak hours.

Furthermore, existing solutions often lack flexibility and scalability, making it challenging to adapt to the evolving needs of urban traffic management. Centralized systems may face difficulties in integrating data from disparate sources or expanding to cover larger geographic areas effectively. This limitation hinders the ability to implement comprehensive traffic management strategies tailored to specific intersections or regions.

In contrast, the proposed solution introduces a novel approach to urban traffic management by leveraging a microservices architecture. This architecture enables the system to be modular and scalable, allowing for the seamless integration of various services, including analytics, regulation, client applications, and object detection.

In the proposed solution, the analytics service plays a crucial role in generating traffic analytics and predictive models by utilizing real-time traffic data from multiple intersections. Leveraging advanced AI algorithms, this service forecasts traffic patterns proactively. By analyzing historical and current traffic data, as well as incorporating various external factors, the analytics service can predict future traffic

conditions with high accuracy. This predictive capability allows for more effective traffic management and congestion reduction, ultimately reducing travel time and increasing overall road safety.

Furthermore, the regulation service complements the analytics service by integrating with it to dynamically adjust traffic light timings based on both real-time data and predictive analytics. Unlike traditional ITLCS that solely rely on current traffic conditions, the regulation service takes a dynamic approach by incorporating predictive modeling. By anticipating future traffic patterns and optimizing signal timings accordingly, this service ensures smoother traffic flow and minimizes congestion, even during peak hours.

The decision-making algorithm within the regulation service uses inputs from both traffic data and analytics to make informed adjustments to traffic light timings. It evaluates real-time traffic conditions alongside predictions generated by the analytics service to determine the most optimal signal timings for each intersection. This holistic approach enables the system to adapt dynamically to changing traffic patterns and optimize traffic flow in real-time.

Overall, the integration of the analytics service and the regulation service allows for an extensive and dynamic approach to traffic management. By combining real-time data analysis with predictive modeling, the system can effectively foresee and respond to traffic conditions, ultimately improving traffic flow and enhancing road safety for people.

The proposed solution also includes a mobile application that provides real-time information about each intersection, including current traffic conditions, predictions, and analytics. Users, including traffic police officers, can access this information remotely, enabling more informed decision-making and proactive traffic management strategies.

Furthermore, each traffic light is equipped with a camera which utilizes object detection technology to gather real-time traffic data. This data is then transmitted to the analytics and regulation services.

Overall, the proposed solution offers several advantages over existing solutions in urban traffic management. By using a microservices architecture, advanced analytics, predictive modeling, and real-time data utilization, the system enables more efficient traffic management, reduced congestion, and increased road safety. Its scalability, flexibility, and integration capabilities make it adapted for addressing the evolving challenges of traffic management in a dynamic and rapidly growing urban environment.

2 SYSTEM MODELING AND DESIGN

In the process of developing a computer system, effective modeling and design are essential steps to ensure the successful realization of project objectives. Careful consideration was given to selecting the most appropriate design methodology. Considering the dynamic nature of the project, the need for frequent collaboration among stakeholders, and the emphasis on iterative development, an Agile methodology was considered as the most suitable approach.

For this project, Agile methodology, specifically Scrum, was chosen as the primary design methodology. Agile methodologies prioritize adaptability, collaboration, and incremental delivery, making them well-suited for projects with evolving requirements and complex system architectures.

UML serves as the primary design language for this project. UML offers a standardized notation for representing system structures, behaviors, and interactions, making it an invaluable tool for communicating design concepts effectively. Through UML diagrams such as class diagrams, sequence diagrams, and use case diagrams, the design team can visualize and communicate intricate system details to stakeholders with clarity and precision.

For the modeling and design phase, PlantUML is selected as the preferred tool [19]. PlantUML is an open-source tool that enables the creation of UML diagrams using a simple and intuitive text-based syntax. Its lightweight nature and compatibility with various text editors make it ideal for collaborative design sessions and version control integration. Additionally, PlantUML offers extensive support for a wide range of UML diagram types, ensuring that the design team can accurately represent all aspects of the system architecture and functionality.

In the subsequent sections of this chapter, the application of the chosen design methodology, language, and tools will be demonstrated through the creation of UML diagrams. These diagrams will serve as blueprints for the ITLCS, capturing its architecture, components, interactions, and behavior in a concise and structured manner. Through diligent modeling and design, the foundation for the system's development and implementation will be laid, setting the stage for subsequent phases of the project.

2.1 Behavioral Description of the System

This section will provide a detailed overview of the behavioral description of the system. This cover will encompass various aspects, including the general system architecture, visual modeling of flows, transaction states represented through Statechart Diagrams, application usage scenarios depicted via Sequence Diagrams, and message flows between system components illustrated by Collaboration Diagrams. It will serve as a comprehensive exploration of the system's behavioral aspects, facilitating a deeper understanding of its functioning and interactions.

The system architecture consists of several key components, including the Traffic Monitoring Service, Traffic Analytics Service, Traffic Regulation Service, Gateway, Traffic Data Service, Mobile

Application, and other supporting infrastructure. These components work together to collect, analyze, and regulate traffic data to optimize traffic flow and enhance road safety.

2.1.1 System Overview

Use case diagrams provide a high-level view of the system's functionalities and interactions with external actors. It presents a visual representation of the various use cases or tasks that users (actors) can perform within the system. Each use case describes a specific functionality or feature offered by the system, along with the actors involved in performing those actions.

They illustrate the relationships between actors and use cases, showcasing how different actors interact with the system to achieve specific goals. They help stakeholders understand the scope of the system, its primary functionalities, and the roles of different users or external entities.

Figure 2.1 depicts the interaction between various system components for updating traffic signal timings in the traffic light module. The process begins with the traffic light module initiating a request to update the signal timings. This module sends a request to the server to obtain the latest traffic light rules. The server, which houses and manages the traffic light rules, processes this request and checks for any updates to the rules. If new rules are available, the server retrieves these updated rules and sends them back to the traffic light module. The module then updates the signal timings based on the new rules to ensure optimized traffic flow. However, if no new rules are present, the server responds by indicating that the existing rules should be maintained. The traffic light module then continues to operate using the current set of rules.

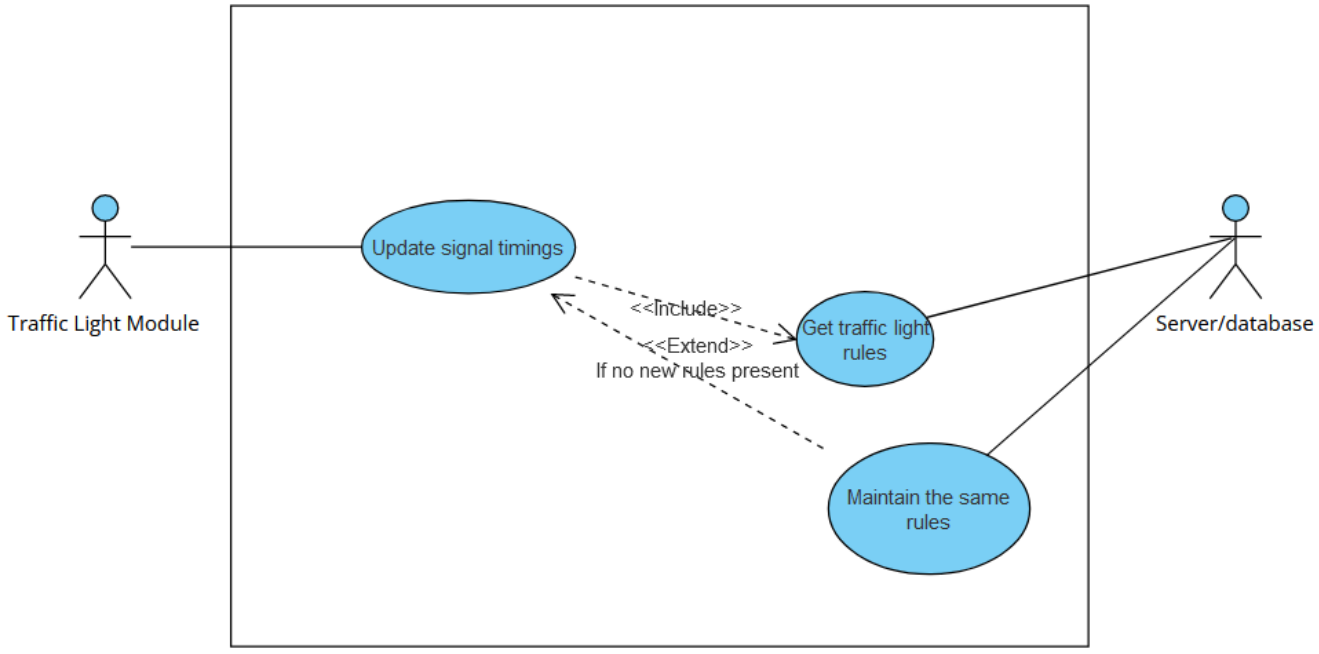

Figure 2.1 – Use Case Diagram for Updating Signal Timings

Figure 2.2 illustrates the interaction between a user and system components to view today's traffic analytics. The user accesses today's traffic analytics through the mobile application, which sends a data

request to the gateway. The gateway then requests traffic data from the server, which in turn requests today's analytics from the database.

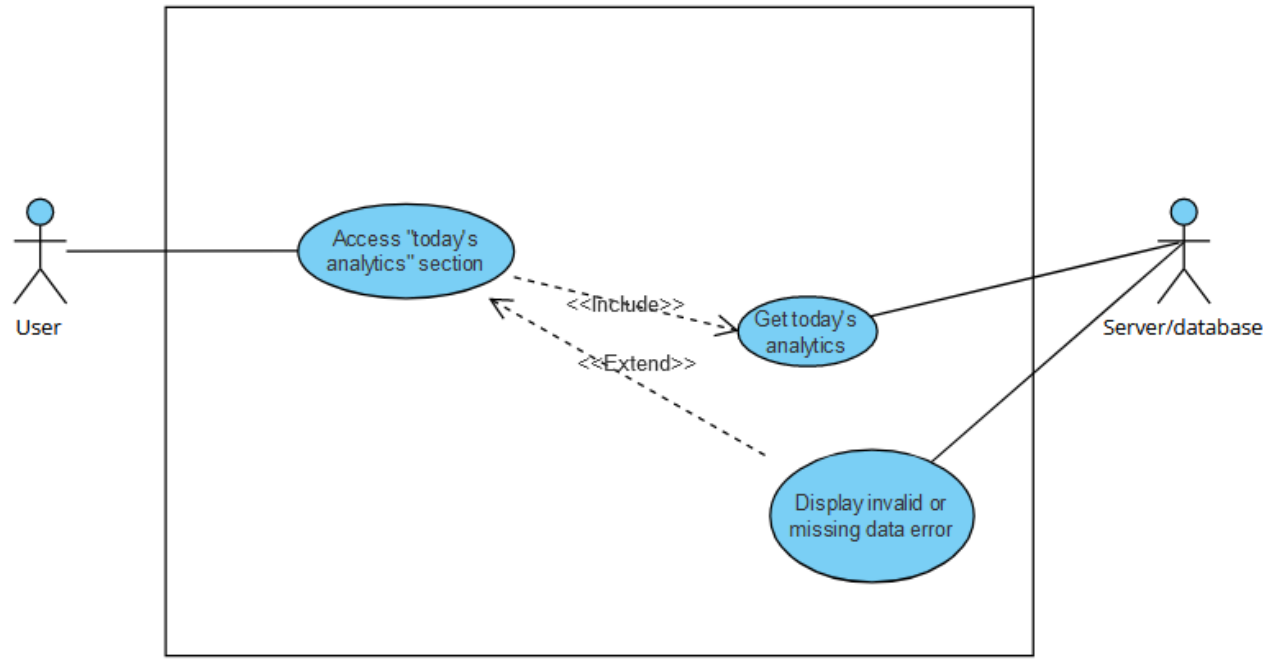

Figure 2.2 – Use Case Diagram for Viewing Today's Traffic Analytics

Figure 2.3 showcases the interaction between a user and system components to access traffic predictions. The traffic police officer requests traffic predictions through the mobile application, which sends a data request to the gateway. The gateway then requests traffic data from the server, which in turn requests predictions from the database.

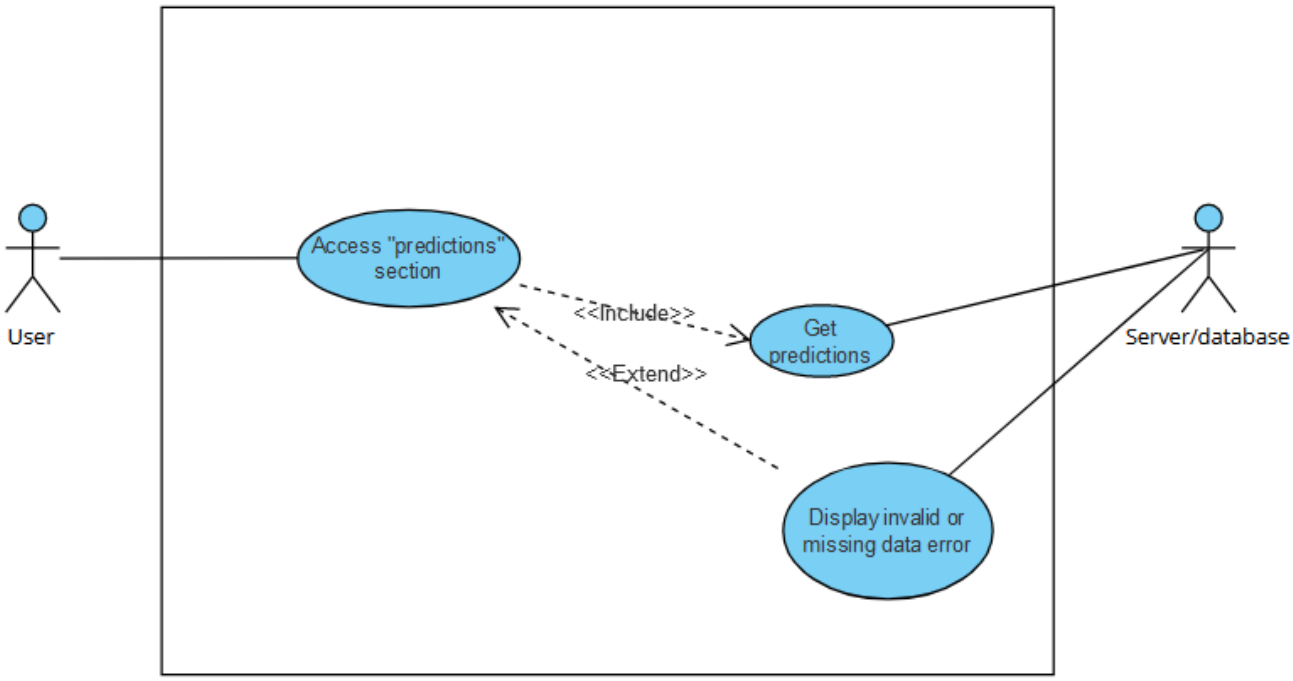

Figure 2.3 – Use Case Diagram for Viewing Traffic Predictions

Figure 2.4 illustrates the process of logging into a mobile app using an account. The primary actors involved are the User, seeking to access the website, and the Mobile App itself.

The main use case depicted is "Login with Account," detailing the steps taken when a user attempts to log in. It begins with the initiation of login by the user, followed by entering their credentials, sending an authentication request to the web application server, and verifying the login credentials.

Upon successful authentication, the user is logged in, granting access to their account or the website's functionalities. In the case of invalid login attempts, the website displays an error message, prompting the user to retry.

Additionally, alternative flows include options for users to sign up for a new account directly on the website or initiate password recovery if they forget their login credentials.

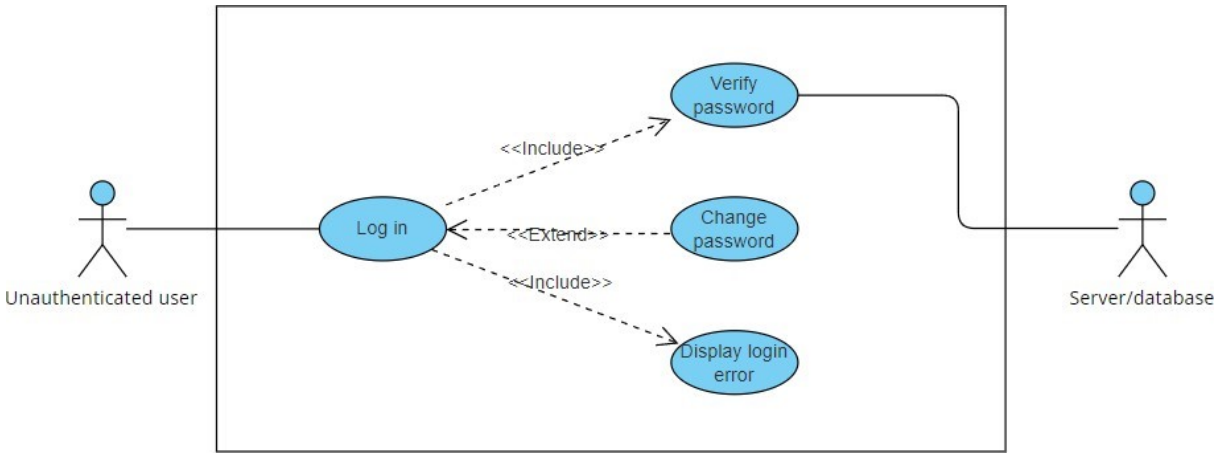

Figure 2.4 – Use Case Diagram for User Login

2.1.2 Visual Modeling of Flows

Activity diagrams provide a visual depiction of the sequence of actions or steps involved in a particular process or use case. These diagrams help to illustrate the workflow and interaction between various components or actors within the system.

Through activity diagrams, we aim to capture the dynamic behavior of the system, showcasing how different elements interact and transition between states or actions. Each activity diagram depicts a specific scenario or use case, breaking down the process into manageable steps and highlighting the sequence of events.

By employing activity diagrams, we gain insights into the flow of operations within the system, including decision points. This visual representation improves our understanding of the system's behavior and improves communication among stakeholders by providing a clear and structured overview of the process flow.

Figure 2.5 showcases the flow of activities when a traffic police officer requests traffic predictions. The process begins with the traffic police officer initiating the request, followed by the mobile application initiating the prediction request. The gateway then sends the request to the Traffic Data Service, which

provides the data to the Traffic Analytics Service. The Traffic Analytics Service generates predictions, which are provided to the gateway. Finally, the gateway forwards the predictions to the mobile application, which displays them to the traffic police officer.

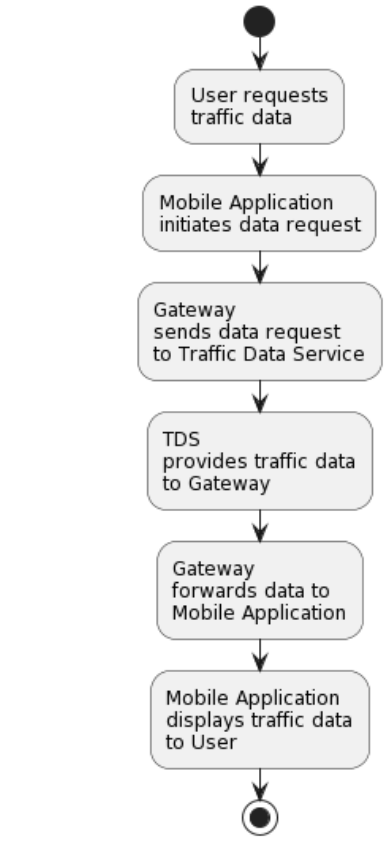

Figure 2.5 – Activity Diagram for Requesting Traffic Predictions

Figure 2.6 depicts the flow of activities involved in analyzing traffic patterns and generating traffic analytics. The process begins with receiving traffic data, followed by the analysis of traffic patterns. Traffic analytics are then generated based on the analysis and provided to the regulation service for further processing.

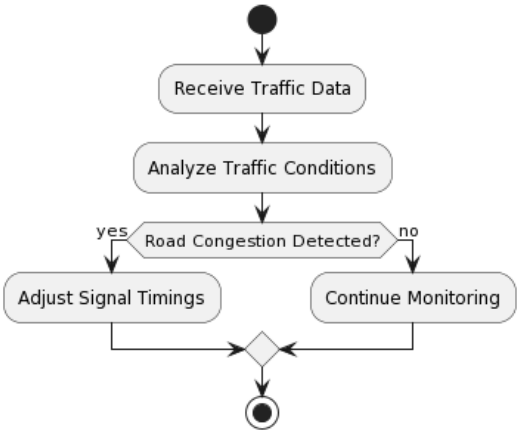

Figure 2.6 – Activity Diagram for Traffic Analytics

Figure 2.7 demonstrates the flow of activities involved in regulating traffic based on analyzed traffic conditions. The process begins with receiving traffic data, followed by the analysis of traffic conditions. If road congestion is detected, signal timings are adjusted accordingly. The process continues monitoring traffic until stopped.

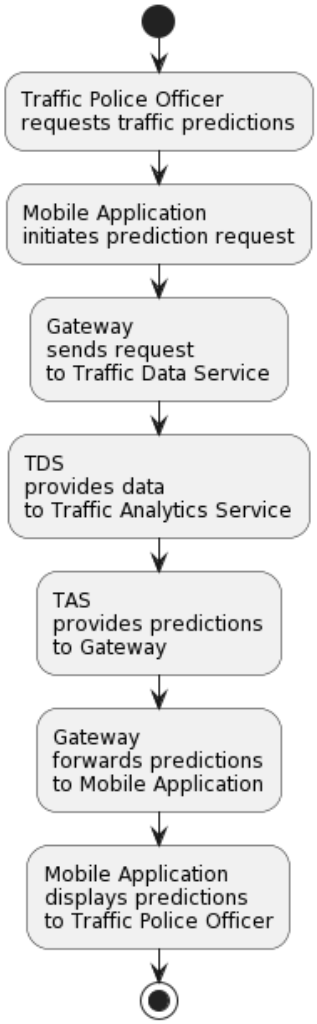

Figure 2.7 – Activity Diagram for Traffic Regulation

2.1.3 System Transaction States

Statechart diagrams provide a visual depiction of the system's behavior over time, illustrating how it responds to different inputs and conditions.

These diagrams capture the dynamic nature of the system, depicting states as nodes and transitions as arrows between these nodes. Each state represents a specific condition or mode of operation, while transitions depict the events or actions that trigger a change in state.

Statechart diagrams are particularly useful for modeling complex systems with multiple states and intricate state transitions. They allow us to visualize the system's behavior in a structured manner, highlighting the sequence of states and transitions that occur during its operation.

By analyzing statechart diagrams, we can gain insights into the system's behavior under different scenarios and conditions. We can identify critical states, understand the conditions that lead to state transitions, and ensure that the system behaves as intended in various situations.

Figure 2.8 illustrates the state transitions within a traffic monitoring system. Initially, the system starts in the TrafficMonitoring state. It proceeds to Getting Traffic Data upon receiving new traffic data. Once the data is structured, it transitions to SendingData and subsequently returns to the initial state after completion.

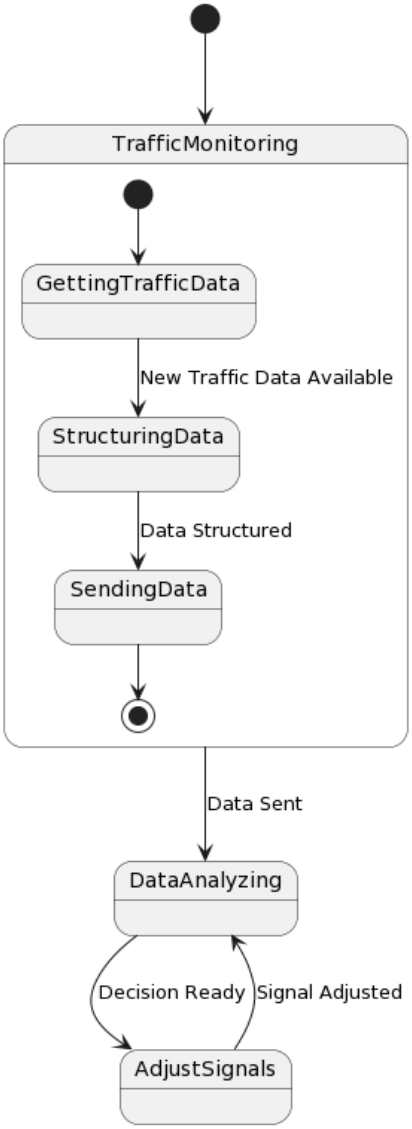

Figure 2.8 – Statechart Diagram for Traffic Monitoring and Data Analysis

After sending data, the system transitions to Data Analyzing where the traffic data is analyzed. When the analysis is complete and a decision is ready, the system moves to AdjustSignals to adjust traffic signals. The system may loop between DataAnalyzing and AdjustSignals as needed for continuous monitoring and adjustments.

Figure 2.9 represents the flow of a traffic monitoring system focusing on traffic data analysis. The system starts in the TrafficMonitoring state and transitions through the stages of receiving, structuring, and sending traffic data. Upon sending data, the system enters TrafficDataAnalyzing where traffic conditions are analyzed (AnalyzingTraffic). If traffic congestion is detected, the system proceeds to Making Decision to initiate appropriate actions. The system then loops back to the initial state for continuous monitoring and data analysis.

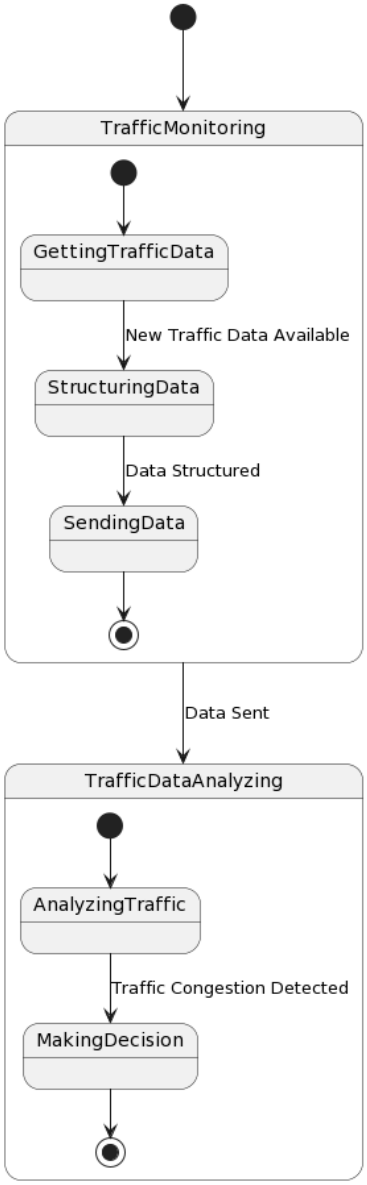

Figure 2.9 – Statechart Diagram for Traffic Congestion Detection

Description: Figure 2.10 depicts the operational states of a traffic monitoring system for object detection. The system begins in the TrafficMonitoring state and initializes its monitoring process (Initializing). It then transitions to Monitoring upon detecting an object event.

If no objects are detected, the system returns to the initial state. However, upon detecting objects (DetectingObjects), it proceeds to (Notifying) to inform relevant components. After notifying, the system resumes monitoring (Monitoring) to continue detecting objects.

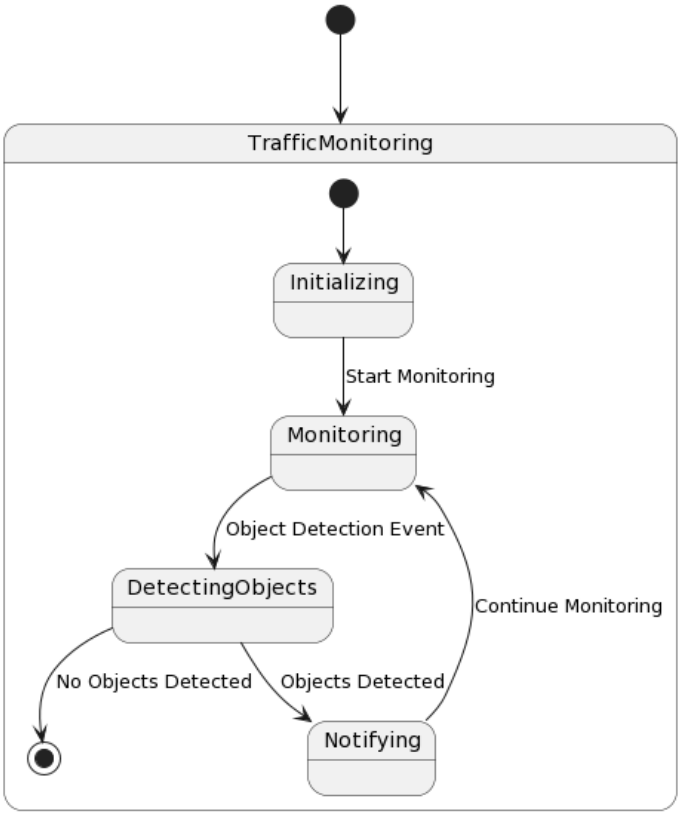

Figure 2.10 – Statechart Diagram for Object Detection

2.1.4 Description of Application Use Cases

The description of application use scenarios through Sequence Diagrams offers a brief and visual narrative of system operations and user interactions. These diagrams serve to illustrate how system components collaborate and communicate to accomplish specific tasks or respond to user inputs over time.

Sequence Diagrams are involved in depicting the dynamic flow of events within the system, showcasing the sequence of messages exchanged between objects. They provide a step-by-step representation of interactions, illustrating the chronological order of method calls, data transmissions, and system responses.

These diagrams serve as a valuable tool for visualizing system behavior in response to user actions, highlighting specific use cases or scenarios. They aid in system analysis and design validation by demonstrating how the system architecture aligns with functional requirements and user expectations.

The sequence diagram depicted in Figure 2.11 outlines the process of user account creation within the mobile application. Users begin by entering their login credentials, triggering an authorization request to the server. The server checks its database for existing account data. If valid credentials are provided, the website logs the user in; otherwise, it notifies the user of an unsuccessful attempt. Users without an account

can create one by accessing the signup feature, entering new account information, and subsequently gaining immediate access upon successful creation.

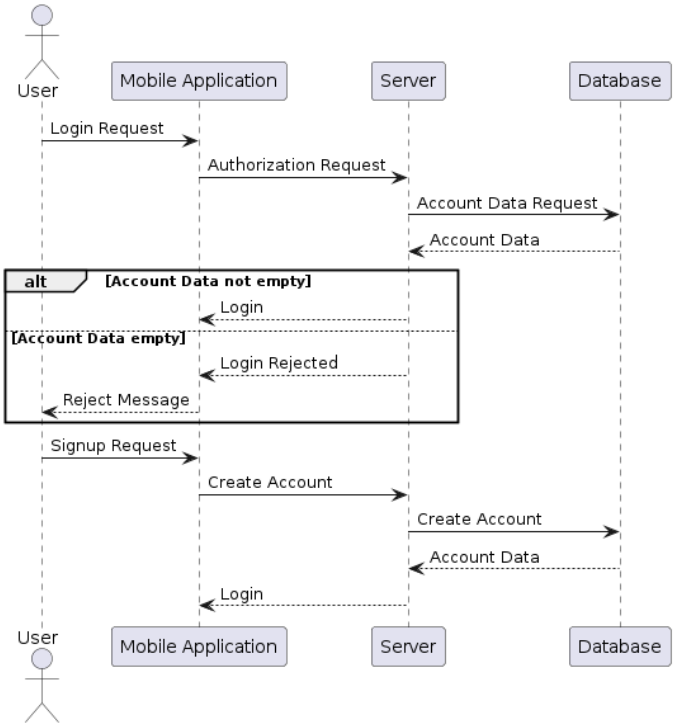

Figure 2.11 – Sequence Diagram for Login/Signup

Figure 2.12 illustrates the scenario where a user accesses traffic data via the mobile application. The interaction begins with the user (Actor) initiating the action of viewing traffic data. This action triggers a communication to the Mobile Application, where the user interface is located.

Once the Mobile Application receives the user's request, it sends a data request to the Gateway, which serves as an intermediary for communication between different services. The Gateway then proceeds to request traffic data from the Traffic Data Service, indicating a need for traffic information.

This data exchange ensures that the Mobile Application receives the requested traffic data for the current day, which can then be displayed to the user within the application interface.

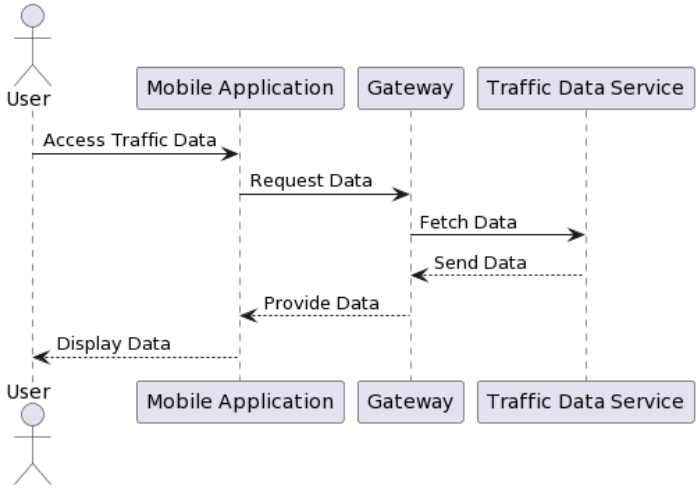

Figure 2.12 – Sequence Diagram for Accessing Traffic Data via Mobile Application

Figure 2.13 illustrates the mobile application's retrieval of analytics data for the current day. A user requests analytics data, triggering a server response to provide the requested information. Upon receipt, the application displays the analytics data for the user. If data is unavailable or restricted, an alternative path is followed, displaying an empty dataset to the user.

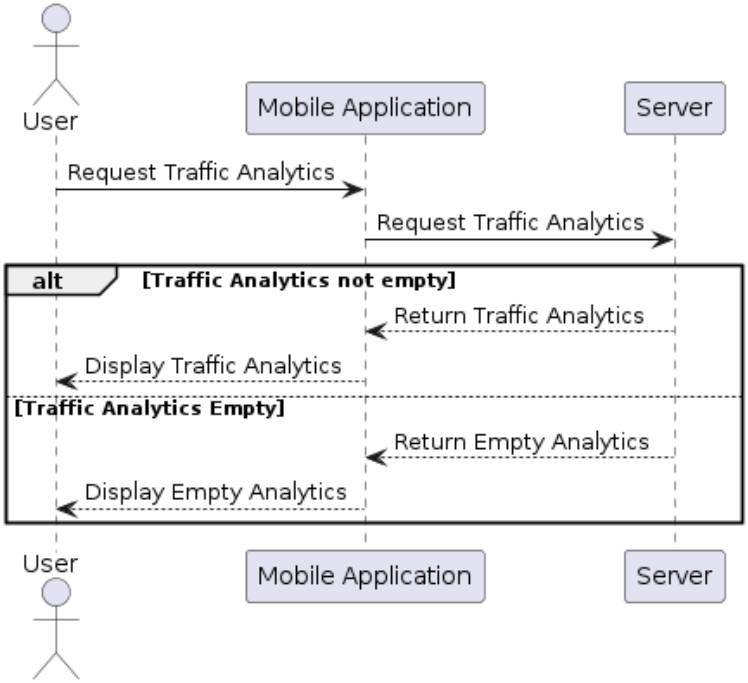

Figure 2.13 – Sequence Diagram for Accessing Traffic Analytics

Figure 2.14 represents the retrieval of predictive analytics data. A user requests predictive insights, and the server responds with relevant predictions based on historical data. The mobile application then displays the predictive analytics to the user. If data is unavailable, the application notifies the user accordingly.

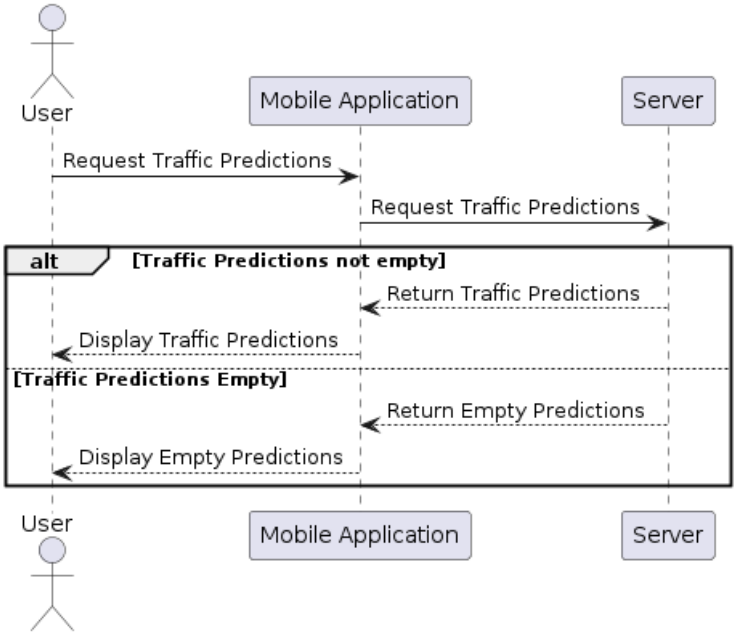

Figure 2.14 – Sequence Diagram for Accessing Traffic Predictions

Figure 2.15 depicts the process of traffic monitoring and analytics within the system. It shows the Traffic Monitoring Service sending traffic data to both the Traffic Analytics Service and the Traffic Regulation Service. The Traffic Data Service then requests historical data from the Traffic Analytics Service.

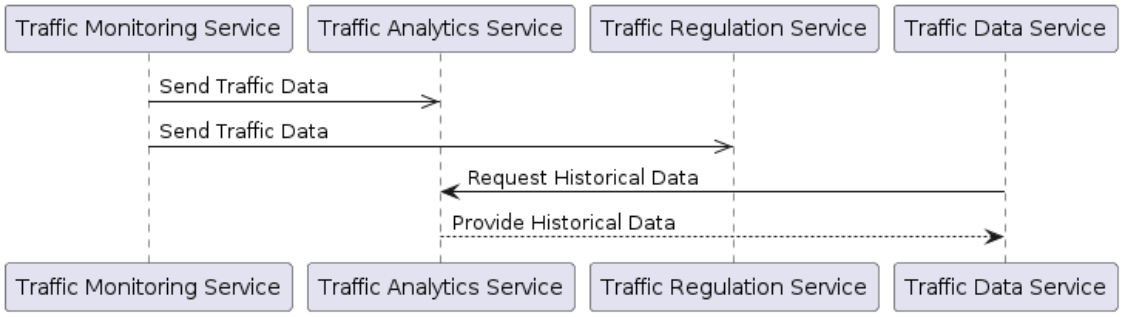

Figure 2.15 – Sequence Diagram for Traffic Flow Process

2.1.5 Message Flows and Links between System Components

Collaboration diagrams illustrate how different components within a system interact by exchanging messages or information. They show the flow of communication between system elements, such as actors (users or external systems) and system components (services, modules, or databases). These diagrams use shapes and arrows to represent the components and the flow of messages between them, providing a visual overview of how the system functions in terms of communication and interaction.

Figure 2.16 depicts the collaboration between the Traffic Monitoring Service, Traffic Analytics service, Traffic Regulation Service, and the PLC module within the traffic management system. The TMS monitors traffic flow and sends data to both the TAS and TRS. The TAS analyzes the traffic data and sends its analysis to the TRS, which then regulates traffic and adjusts signal timings based on the analysis and real-time data. Finally, the TRS communicates with the PLC module to implement the signal timing adjustments as commanded by the regulation service.

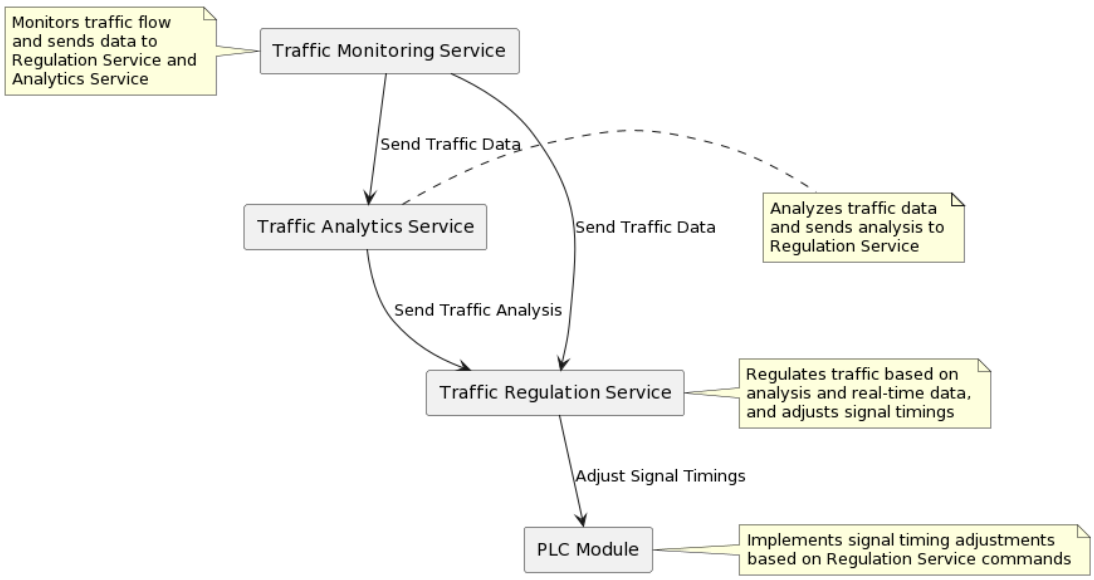

Figure 2.16 – Collaboration Diagram for Traffic Management Services

Figure 2.17 represents the collaboration between users, the mobile application, gateway, traffic data service, and traffic analytics service for retrieving traffic data. Users and traffic police officers send data requests to the mobile application, which forwards them to the gateway. The gateway requests traffic data from the traffic data service, which in turn requests analytics data from the traffic analytics service. Finally, the traffic analytics service provides analytics data to the traffic data service, which returns it to the gateway and subsequently to the mobile application for user access.

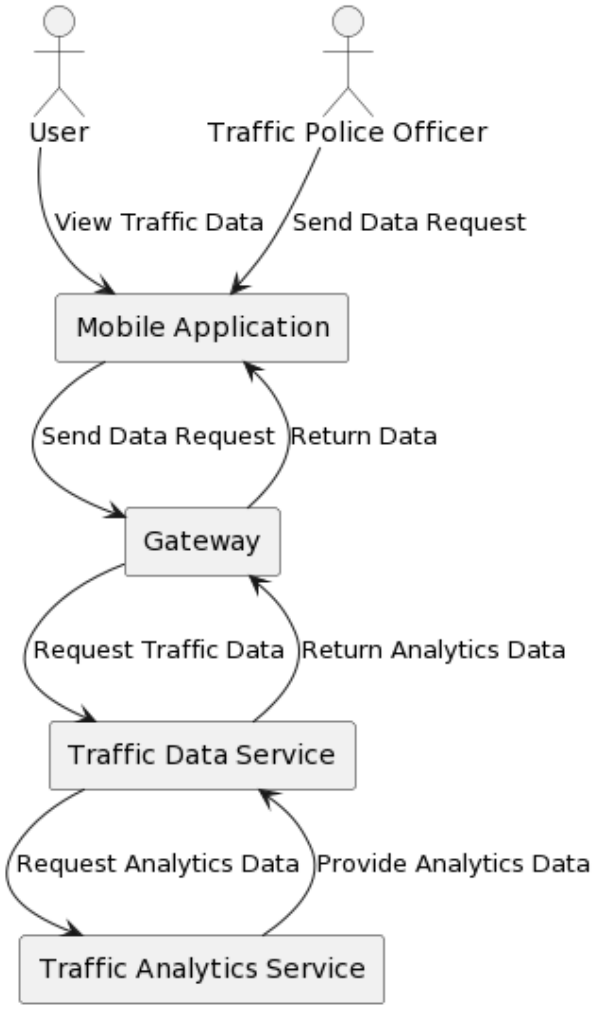

Figure 2.17 – Collaboration Diagram for Traffic Data Retrieval

2.2 Structural Description of the System

This section goes through the structural description of the system, providing an overview of its static components and their relationships. It encompasses three key aspects: description of the static structure, dependency relationships between components, and modeling the equipment of the implementation environment.

The static structure of the system outlines the composition of its components, including modules, services, and external interfaces. We examine the architectural elements that form the backbone of the system, emphasizing their roles and interconnections.

We analyze the dependency relationships between system components, explaining how different modules interact and rely on one another to fulfill specific functionalities. This includes exploring dependencies at both the design and implementation levels.

Understanding the implementation environment is vital for system deployment. We model the hardware and software components of the implementation environment, considering factors like servers, databases, communication protocols, and supporting infrastructure.

2.2.1 Description of the Static Structure of the System

Class diagrams illustrate how classes interact, showcasing attributes, operations, and associations between components. They serve as a blueprint for understanding the system's architecture and organization, facilitating design decisions and system maintenance. Through these diagrams, stakeholders gain insights into the system's static structure, promoting clarity and helping in system development and management.

Figure 2.18 illustrates the static structure of the Traffic Monitoring System, focusing on key classes and their relationships. It includes classes such as TrafficMonitor, TrafficCamera, TrafficData, and TrafficEvent, which represent components of the monitoring system.

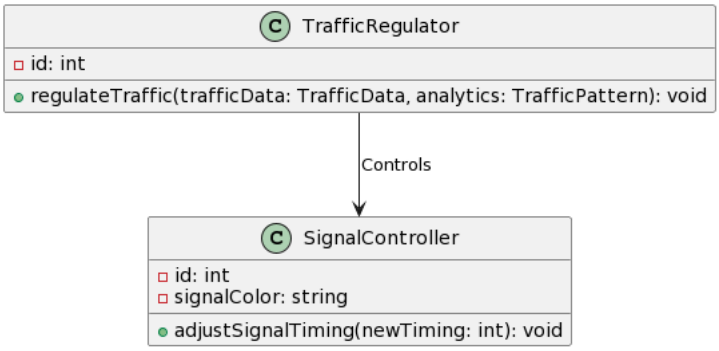

Figure 2.18 – Class Diagram For Traffic Monitoring Service

Figure 2.19 represents the static structure of the Traffic Analytics Service. It outlines classes like TrafficAnalyzer, TrafficPattern, and TrafficPrediction. The diagram shows how these classes collaborate to analyze traffic data and generate predictions based on historical patterns.

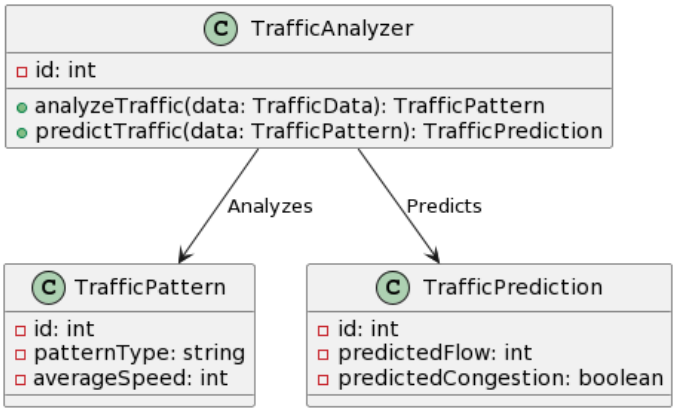

Figure 2.19 – Class Diagram For Traffic Analytics Service

Figure 2.20 illustrates the static structure of the Traffic Regulation Service. It includes classes such as TrafficRegulator and SignalController, showcasing their attributes and interactions. This diagram highlights how the service regulates traffic based on analytics and adjusts signal timings.

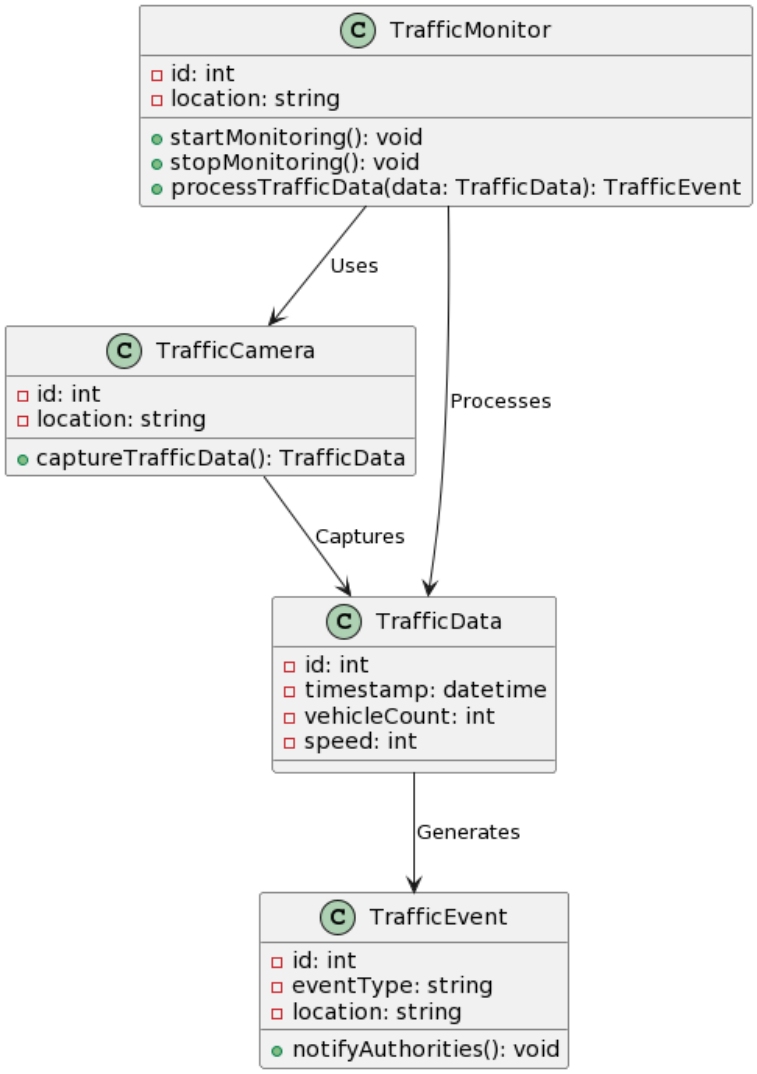

Figure 2.20 – Class Diagram For Traffic Regulation Service

2.2.2 Dependency Relationships between System Components

Component diagrams visually represent the structural dependencies and interactions between various components within a software system. They illustrate how system components are composed, showing dependencies, interfaces, and interactions. These diagrams emphasize the organization of components into logical layers and modules, promoting system decomposition, abstraction, and understanding of system architecture. Overall, component diagrams provide a concise and graphical overview of the system's structural design, helping in effective communication and decision-making during software development and maintenance.

Figure 2.21 represents the component diagram of the general architecture of the system.

Mobile Application represents the user interface accessed via mobile application, enabling users to access real-time traffic information.

Traffic Monitoring Service collects and processes traffic data from sensors or data sources, facilitating analysis.

Gateway Service acts as a central communication hub, routing requests between various system services and managing traffic flow.

Traffic Regulation Service implements traffic regulations based on collected data, controlling traffic lights or devices.

Traffic Analytics Service analyzes traffic data to derive insights and identify traffic patterns or anomalies.

Traffic Data Center manages retrieval of analytics data for the mobile application.

Databases store configurations or rules for the Traffic Regulation Service and analytics data generated by the Traffic Analytics Service.

The system employs a modular structure, assigning specific tasks to distinct components for better maintainability and scalability. Separate databases for different functions ensure data isolation, possibly for security or performance reasons. The API-based architecture provides effective communication between system components, providing flexibility and potential integration with other systems.

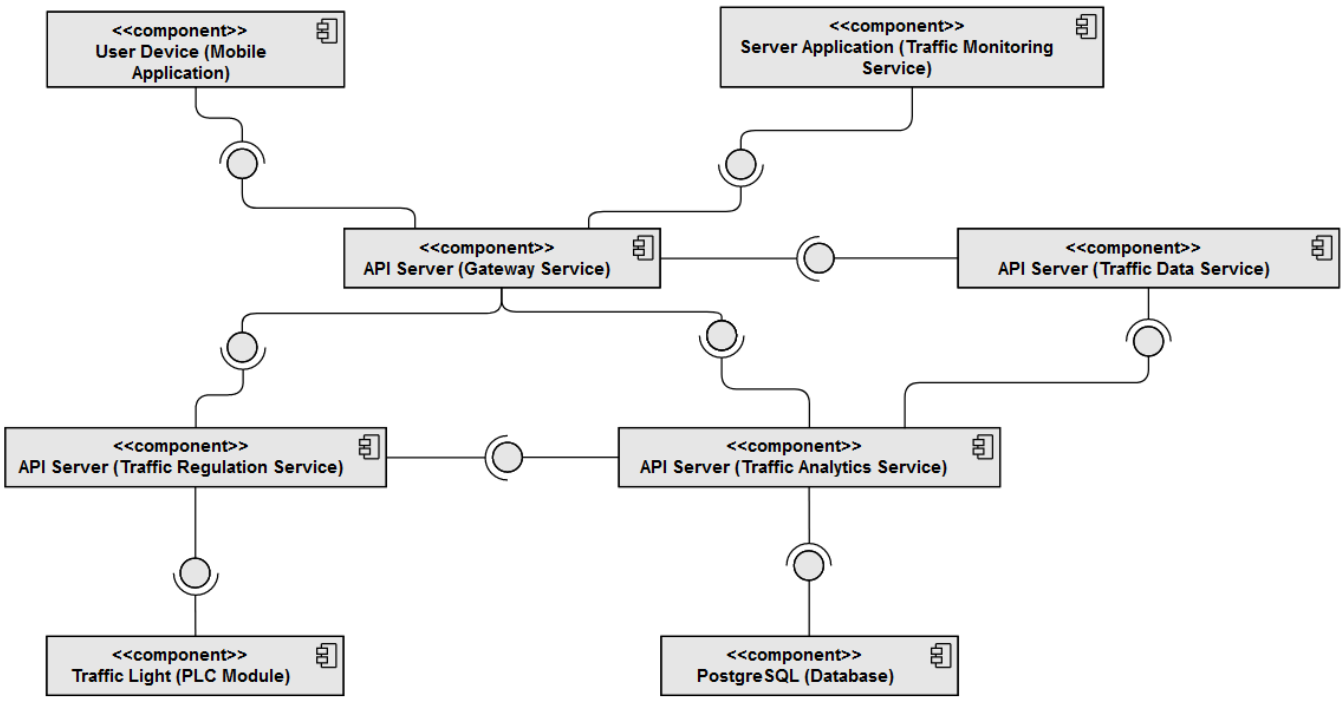

Figure 2.21 – Component Diagram for System Architecture

Figure 2.22 illustrates the flow of real time traffic data within the traffic management system.

Traffic Monitoring Service monitors traffic flow and sends traffic-related data to the gateway. Gateway acts as a central hub, facilitating communication between different system components. Traffic

Regulation Service receives traffic regulation commands from the gateway and adjusts traffic signals. Traffic Analytics Service analyzes traffic data received from the gateway and collaborates with the database.

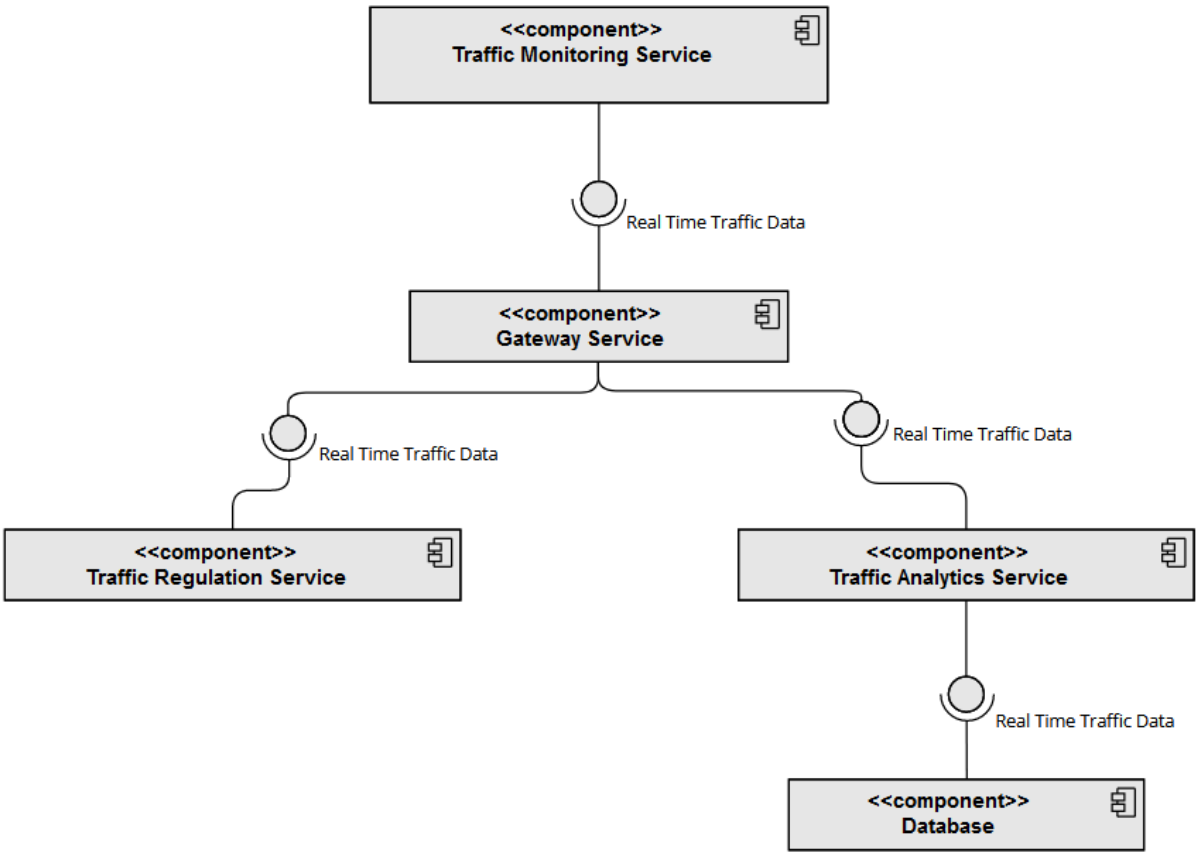

Figure 2.22 – Component Diagram for Real Time Data Flow

The connections mean the exchange of information:

- Traffic Monitoring Service → Gateway provides real time traffic data to the gateway for further processing;
- Gateway → Traffic Regulation Service sends real time traffic data for traffic regulation adjustments;
- Gateway → Traffic Analytics Service sends real time traffic data exchange for analysis;
- Traffic Data Service → Traffic Analytics Service provides historical data for analysis purposes;
- Traffic Analytics Service → Database sends traffic data for storing;

Figure 2.23 depicts the interaction between components involved in user interactions within the system. Mobile Application represents the user interface for interacting with the traffic management system. Gateway acts as an intermediary between the mobile application and various backend services. Traffic Data Service handles requests and responses related to traffic data retrieval. Traffic Analytics Service manages traffic analysis tasks and provides analytical insights.

The connections mean the flow of interactions:

- Mobile Application → Gateway initiates requests and receives responses through the gateway;

- Gateway → Traffic Data Service requests traffic-related data from the data service;
- Traffic Data Service → Traffic Analytics Service requests analytical insights and receives traffic analysis results;

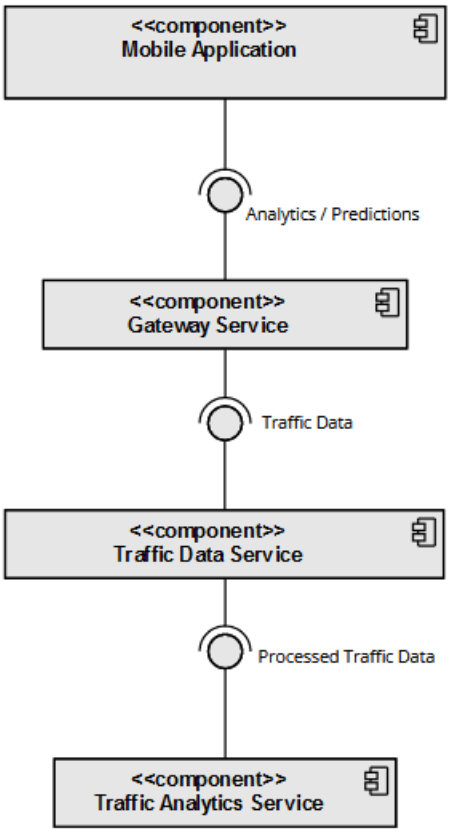

Figure 2.23 – Component Diagram for User Interaction

2.2.3 Modeling the Equipement of the Implementation Environment

The deployment diagram, a type of UML diagram, illustrates the physical deployment of software components across hardware nodes within the system's implementation environment. It showcases the relationships and interactions between software components and the hardware infrastructure. This diagram is particularly useful for visualizing how software applications are deployed on servers, devices, or virtual machines.

Figure 2.24 illustrates the physical architecture of a traffic monitoring system, showcasing the distribution and relationships of software components across different server nodes within the deployment environment.

The User Device hosts the Mobile Application component, which serves as the user interface for accessing traffic-related information.

The Traffic Monitor hosts the Video Camera and Traffic Monitoring Service, which serves as the real time traffic data provider.

The Traffic Management Server serves for critical components:

- Traffic Analytics Service analyzes traffic data to derive insights;
- Traffic Analytics Database stores real time traffic data, analytics data and predictions data;
- Traffic Regulation Service implements traffic regulations based on analytics and real time data;
- Traffic Data Service handles data retrieval and communication tasks;

The Traffic Light hosts the PLC Module, which controls the actual traffic light.

Deployment Relationships:

- The MA communicates with TDS on the API Server to retrieve and display traffic-related information;
- TDS interacts with TAS to exchange traffic data and analytics;
- TMS communicates with both TAS and TRS to provide data for analysis and regulation;
- TAS accesses Traffic Analytics Database to store and retrieve analytics data;
- TRS interacts with TAS to exchange traffic analytics.

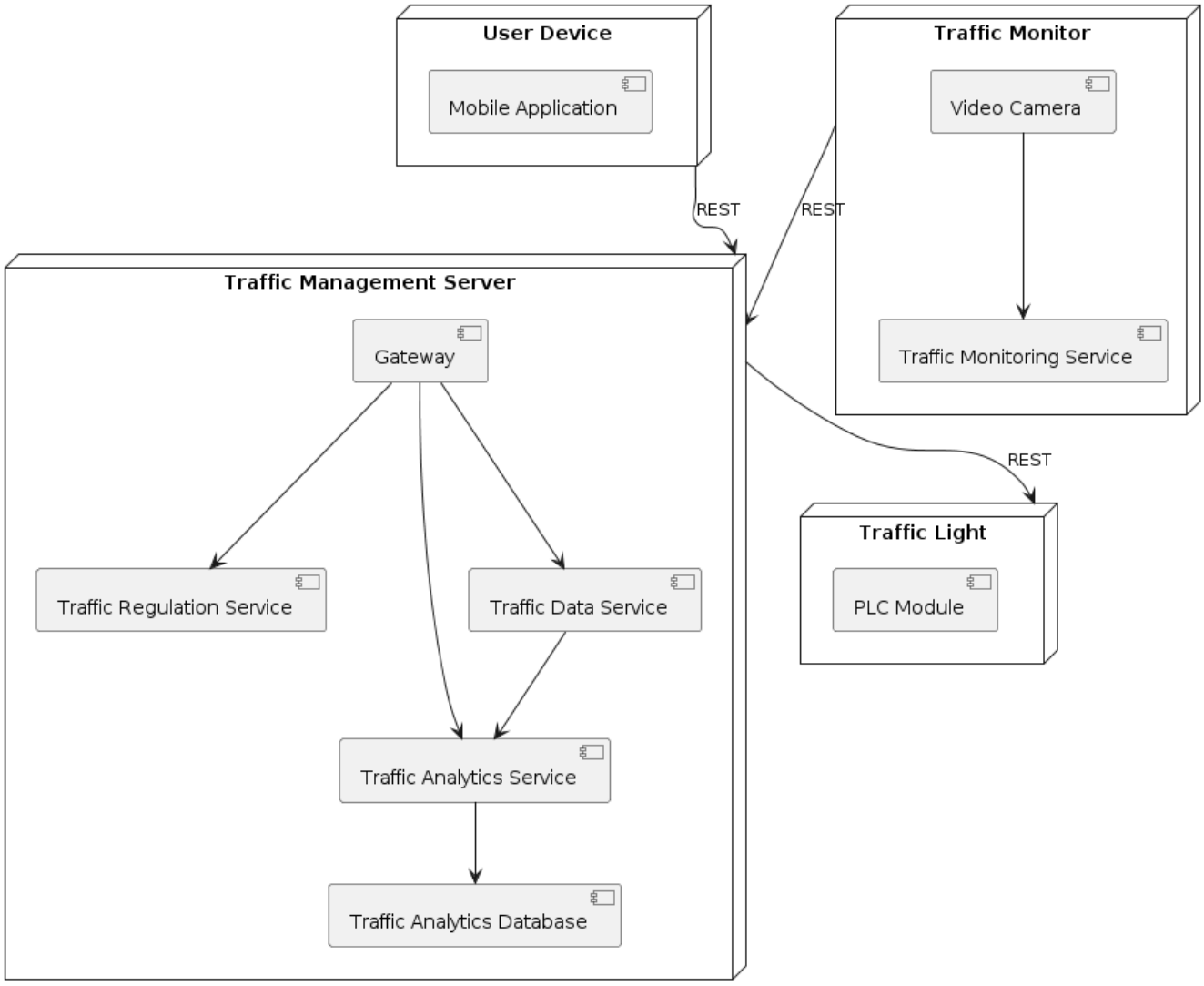

Figure 2.24 – Deployment Diagram for Traffic Management System

3 SYSTEM IMPLEMENTATION

3.1 Traffic Monitoring Service

The Traffic Monitoring service is built using the Python programming language. It is a sophisticated system built using computer vision techniques to monitor traffic conditions in real-time from video input. This system integrates various components, including object detection, object tracking, data analysis within defined road sections, and interaction with external services for traffic analysis and regulation.

The YOLO deep learning model, initialized with the `yolov8n.pt' pre-trained weights in this module (see Listing 3.1), represents a recent approach to real-time object detection [20].

```
model = YOLO("yolo-weights/yolov8n.pt")
results = model(img_region, stream=True)
```

Listing 3.1 – Object Detection with YOLO

YOLO operates by partitioning the input image into a grid and making predictions directly from this grid, enabling efficient detection of multiple object classes including vehicles, pedestrians, bicycles, and traffic signs within each frame of the video (see Figure 3.1). This model stands out for its single-shot detection capability, processing images in a single pass through the network without the need for complex region proposal mechanisms. By influencing anchor boxes for precise localization and multi-class classification, YOLO demonstrates high accuracy across various object scales and shapes. The use of pre-trained weights improves the model's efficiency and effectiveness, making it well-suited for applications requiring rapid and accurate object detection, such as traffic monitoring.

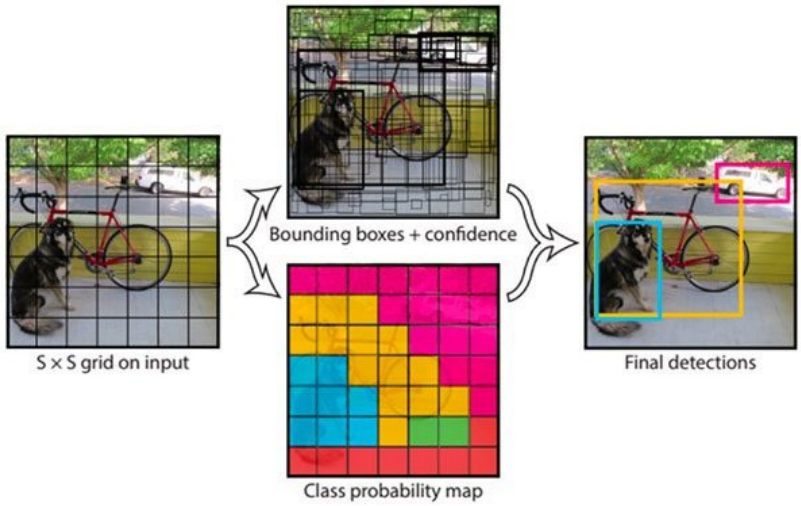

Figure 3.1 – A Simplified Illustration of the YOLO Object Detector Pipeline [20]

To track detected objects across frames and maintain their identities, the module uses the SORT algorithm. This algorithm associates detected objects with existing tracks based on their spatial and temporal characteristics, ensuring reliable tracking even in complex traffic scenarios [21]. By using a combination of motion predictions and matching techniques, SORT maintains continuity in tracking objects across frames, essential for advanced traffic analysis and monitoring applications. The algorithm's efficiency and effectiveness in real-time tracking make it good for scenarios requiring continuous

observation and analysis of moving objects within video streams. Listing 3.2 illustrates the initialization of the tracker.

```
tracker = Sort(max_age=20, min_hits=3, iou_threshold=0.3)
results_tracker = tracker.update(detections)
```

Listing 3.2 – Object Tracking with SORT

In this case, the video frame is divided into specific road sections (`section1`, `section2`, `section3`), each defined by coordinates to represent different areas of interest within the scene. Detected objects, such as vehicles and pedestrians, are categorized based on their positions relative to these road sections. The division is represented in Listing 3.3.

```
# Define road sections
# Road 1 coordinates
section1 = {"x_min": 0, "y_min": 0, "x_max": 540, "y_max": 370}
# Road 2 coordinates
section2 = {"x_min": 560, "y_min": 200, "x_max": 1200, "y_max": 430}
# Road 3 coordinates
section3 = {"x_min": 400, "y_min": 450, "x_max": 1150, "y_max": 710}
```

Listing 3.3 – Road Section Definition and Traffic Analysis

At regular intervals (`interval'), the module collects traffic-related data within each road section, including vehicle counts and pedestrian counts. Listing 3.4 illustrates how data is timestamped and prepared for transmission to external traffic analytics and regulation services via HTTP POST requests.

```
traffic_data = {
     "time": current_time,
     "vehicle_count": len(results_tracker_in_section1) +
len(results_tracker_in_section3),
     "pedestrian_count": len(results_tracker_in_section2),
     "traffic_light_id": 1
}

response_analytics = requests.post(traffic_analytics_service_url, json=traffic_data)
response_regulation = requests.post(traffic_regulation_service_url,
json=traffic_data)
```

Listing 3.4 – Data Collection and Transmission

To provide visual feedback, the module annotates the video frames with bounding boxes around detected objects and overlays tracking IDs. Additionally, road sections are visually highlighted to indicate the categorization of objects based on their locations within the scene. Listing 3.5 represents a code snippet of the data annotation.

```
     # Draw bounding box around tracked object
     cv2.rectangle(img, (x1, y1), (x2, y2), (255, 102, 102), 2)
     # Display tracking ID
     cv2.putText(img, f'ID: {int(car_id)}', (x1, max(30, y1 - 10)),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 102, 102), 2)
     # Highlight road section based on object's location
     if section1["x_min"] <= x1 <= section1["x_max"] and section1["y_min"] <= y1 <=
section1["y_max"]:
          road_name = "Road 1"
          cv2.rectangle(img, (section1["x_min"], section1["y_min"]),
(section1["x_max"], section1["y_max"]), (0, 255, 0), 2)
```

Listing 3.5 – Object Visualization and Annotation

Figure 3.2 illustrates the road sections, bounding boxes, and tracking IDs.

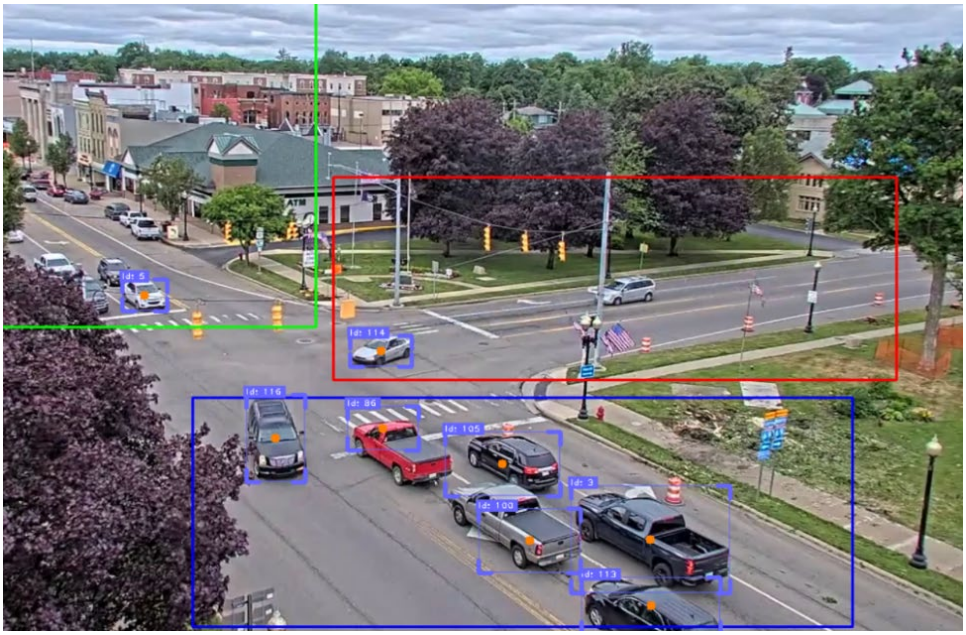

Figure 3.2 – Object Detection and Tracking

3.2 Traffic Analytics Service

The Traffic Analytics Service is a Flask-based application, built using the Python programming language, designed to provide real-time traffic monitoring, analytics, and predictive insights based on traffic-related data stored in a PostgreSQL database. This service integrates various functionalities including database setup, data retrieval, statistical analysis, machine learning model training, and endpoint handling for data access.

The application uses a PostgreSQL database to store traffic-related information such as traffic lights, traffic records, and traffic statistics. The database schema is designed with appropriate tables and relationships to facilitate efficient data storage and retrieval (see Annex A).

The Flask application exposes several endpoints to interact with the traffic-related data stored in the database.

- GET /get_traffic_lights - Retrieves a list of traffic lights with their respective IDs, names, coordinates, and addresses from the database.
- POST /add_traffic_record - Adds traffic records to the database, including details such as traffic light ID, timestamp, vehicle count, and pedestrian count. This endpoint also updates the real-time data cache for monitoring purposes.
- GET /get_statistics/<int:traffic_light_id> - Fetches statistics for a specific traffic light ID, including peak hours, normal hours, light hours intervals, mean vehicle and pedestrian counts, and time range.
- GET /get_predictions/<int:traffic_light_id> - Retrieves predictions for vehicle counts at a specific traffic light ID and time using trained machine learning models.

The Traffic Analytics Service integrates several background processes to manage and analyze traffic-related data efficiently. These processes cover updating real-time data caches, calculating detailed statistics for traffic patterns, and training machine learning models for predictive analytics based on traffic records.

The real-time data cache is crucial for providing immediate access to the latest traffic information without directly querying the database for each request. The background process responsible for updating this cache regularly retrieves traffic records from the database and stores them locally. Listing 3.6 illustrates this process. This ensures that the service can quickly respond to data requests and maintain responsiveness for real-time monitoring and analysis.

```
def update_cache():
    """
    Background thread function to update the real-time data cache.
    Retrieves traffic records from the database and stores them locally for quick
access.
    """
    while True:
        # Retrieve and update traffic records from the database
        # Store the records in a local cache for real-time data access
        # Implement synchronization mechanisms to ensure data consistency
        pass
```

Listing 3.6 – Function for Updating Cache

The calculation of traffic statistics involves analyzing historical traffic data to obtain meaningful insights into traffic patterns and trends. This process includes identifying peak hours, normal traffic intervals, and light traffic periods based on vehicle and pedestrian counts over time. The statistics are computed for each traffic light location and stored for retrieval when requested by clients.

The traffic analytics service uses machine learning models to predict vehicle counts at specific traffic light locations and times. This involves preprocessing historical traffic data, splitting it into training and validation sets, and training regression models such as Random Forest Regressor.

Before training the machine learning models, the traffic records data undergoes preprocessing. This involves extracting features such as hour and minute from the timestamp to represent time-based patterns in the traffic data (see Listing 3.7).

```
def extract_hour_minute(df):
    """
    Extracts hour and minute features from the timestamp in the DataFrame.
    """
    df_copy = df.copy()
    df_copy['time'] = pd.to_datetime(df_copy['time'])
    df_copy['hour'] = df_copy['time'].dt.hour
    df_copy['minute'] = df_copy['time'].dt.minute
    return df_copy
```

Listing 3.7 – Function for Calculating Traffic Statistics

The Random Forest Regressor is a machine learning algorithm used for regression tasks, including predicting vehicle counts based on time-related features [22]. This algorithm works by constructing

multiple decision trees during training and outputting the average prediction of the individual trees. Figure 3.3 represents the decision tree of the algorithm.

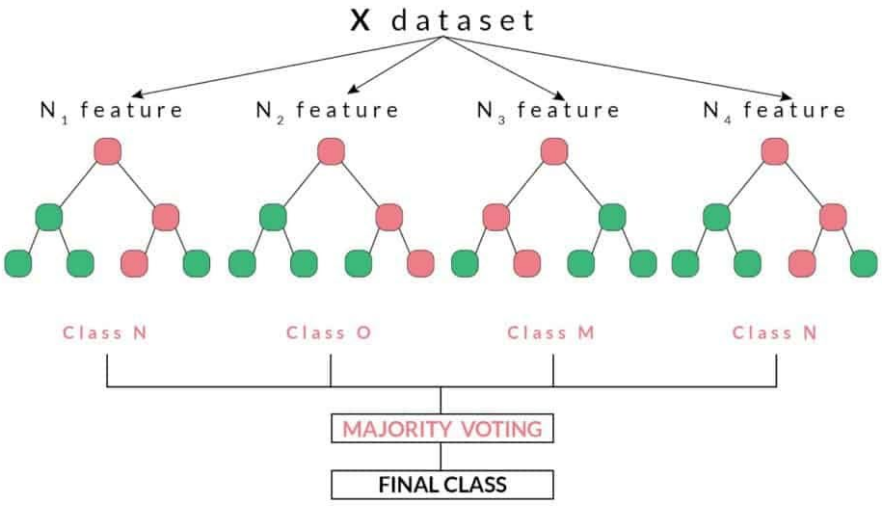

Figure 3.3 – Random Forest Algorithm Decision Tree [23]

Listing 3.7 shows the function that trains the models for predicting vehicle counts.

```
def train_models_by_intersection(df):
    """
    Trains Random Forest Regressor models for each traffic light intersection.
    """
    models = {}
    df_copy = extract_hour_minute(df)
    grouped_data = df_copy.groupby('traffic_light_id')

    for traffic_light_id, data in grouped_data:
        X = data[['hour', 'minute']].values
        y = data['vehicle_count'].values
        X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

        model = RandomForestRegressor(n_estimators=100, random_state=42)
        model.fit(X_train, y_train)

        models[traffic_light_id] = model

    return models
```

Listing 3.7 – Function for Making Vehicle Count Predictions

By training Random Forest Regressor models for each traffic light intersection based on historical traffic data, the service can predict future vehicle counts at specific times. These trained models are updated periodically to adapt to changing traffic conditions and optimize traffic flow efficiently.

Through the integration of real-time data management and machine learning algorithms, the Traffic Analytics Service delivers timely and actionable insights for traffic monitoring, management, and decision-making. The use of statistical analysis and predictive modeling improves the service's capabilities, enabling effective traffic management strategies.

3.3 Traffic Regulation Service

The Traffic Regulation Service is a Flask-based application, built using the Python programming language. It is a key component of the ITLCS, designed to optimize traffic flow and improve safety at intersections through intelligent traffic light control. This service utilizes real-time data, statistical analysis, and machine learning to dynamically adjust traffic light timings based on current traffic conditions and historical patterns.

The service aims to minimize congestion and delays by dynamically adjusting traffic light durations to match changing traffic demands. Using real-time data and historical traffic patterns, the service adapts traffic light timings to varying traffic conditions throughout the day. The decision-making algorithm within the Traffic Regulation Service plays a critical role in dynamically adjusting traffic light timings based on real-time data and historical traffic patterns. This algorithm ensures efficient traffic flow, and minimizes congestion.

Incoming traffic data, including vehicle counts, pedestrian counts, and intersection identifiers, is processed to assess current traffic conditions. The algorithm identifies the current traffic period (peak hours, light hours, normal hours) based on predefined time intervals associated with traffic patterns. Based on the identified traffic period and emergency conditions, the algorithm recommends optimal durations for green and red lights at each intersection. Continuous updates to traffic light rules ensure that intersections adapt to changing traffic demands, optimizing traffic flow throughout the day (see Annex C).

The algorithm begins by processing incoming real-time traffic data to evaluate current traffic conditions at specific intersections. It retrieves vehicle and pedestrian counts from the incoming data. These counts are compared against historical averages to identify deviations and assess traffic intensity.

If vehicle or pedestrian counts exceed predetermined thresholds (e.g., twice the historical average), the system triggers emergency rules to manage traffic accordingly.

Emergency rules prioritize safety by adjusting traffic light timings during unexpected surges in traffic volume.

Based on the assessed conditions, the algorithm recommends updated durations for green and red lights at each intersection.

These recommendations aim to optimize traffic flow, reduce congestion, and improve overall traffic management.

The recommendation of traffic light durations is a critical aspect of the algorithm. The algorithm determines whether the current time falls within peak hours, light hours, or normal hours based on historical traffic patterns and predefined time intervals. Based on the identified traffic period, the algorithm suggests optimal durations for green and red lights to accommodate traffic demands effectively. Different durations are recommended for peak hours, light hours, and normal hours to optimize traffic flow and minimize delays.

The algorithm continuously updates traffic light rules based on real-time data, historical statistics, and predefined rules. The system runs background threads that periodically update traffic light rules for each intersection. This ensures that traffic light timings remain responsive to changing traffic conditions throughout the day. During emergency conditions (e.g., high traffic volumes), the algorithm adjusts traffic light timings to prioritize efficient traffic management. Emergency rules override standard recommendations to address immediate traffic challenges effectively.

3.4 Traffic Insights Mobile Application

The Traffic Insights Mobile Application represents a sophisticated toolset designed to provide users with detailed traffic information and intersection-specific analytics and predictions.

The primary objective of the Traffic Insights App is to offer users, an interactive map interface that displays traffic intersections as markers, allowing users to explore intersections and obtain detailed information, access to data about each intersection, including mean vehicle count, pedestrian count, operational intervals, and predictive analytics for future traffic conditions, a seamless and intuitive user interface that offers easy navigation, search functionality, and actionable insights for urban commuters, transportation planners, and city officials.

The Map Screen uses GPS technology to determine the user's current location and render an interactive map displaying traffic intersections as markers. Users can search for specific intersections by name or address and view detailed information about selected intersections in a modal overlay. The interactive map is represented in Figure 3.4.

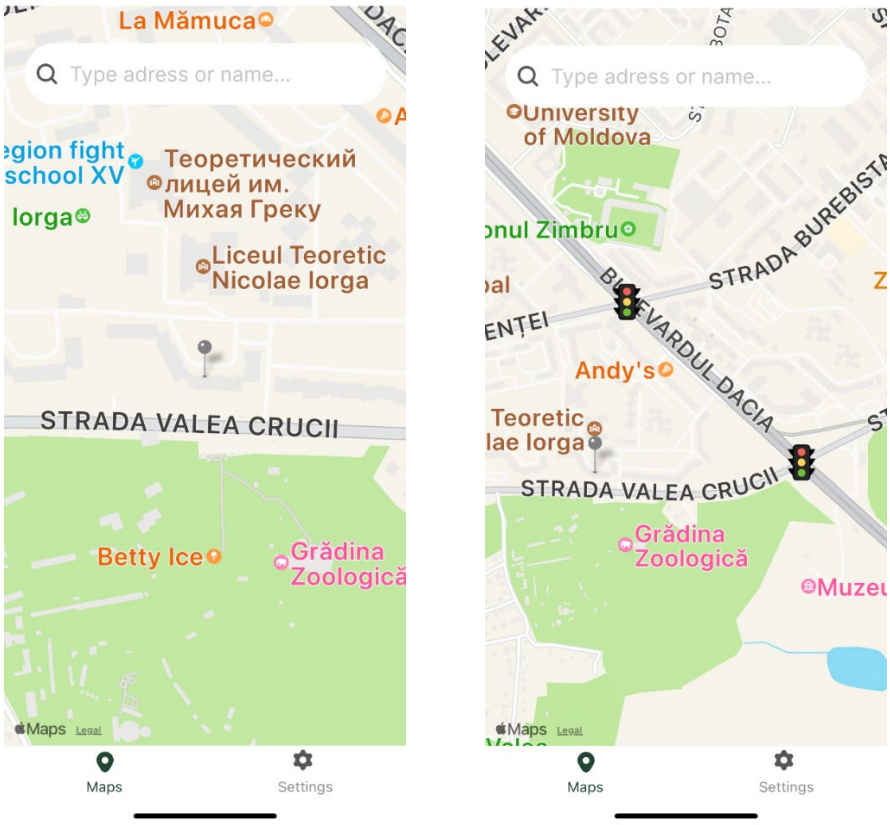

Figure 3.4 – Traffic Insights Map Screen

The Details Screen fetches intersection data from backend APIs and displays mean vehicle count, pedestrian count, and operational intervals for selected intersections. It also provides predictive insights into future traffic conditions based on historical data and machine learning models. The traffic insights are displayed in Figure 3.5.

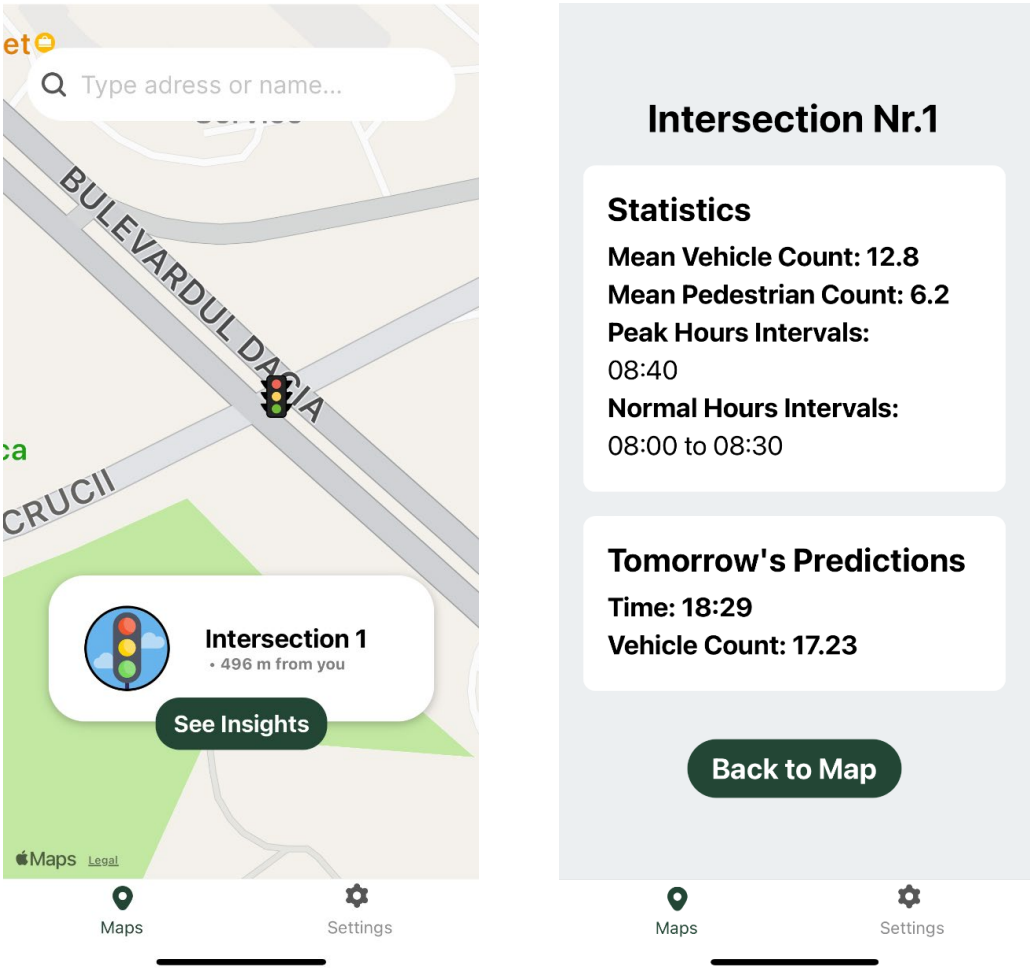

Figure 3.5 – Traffic Insights Details Screen

The key components and functions of the main screens are shown in Annex D.

The Traffic Insights app utilizes the following technologies and integrations:

- React Native - A framework for building cross-platform mobile applications with a native user interface.
- react-native-maps - A library for integrating interactive maps into React Native applications.
- Expo - A development platform and set of tools that simplify the development and deployment of React Native applications.
- Location Services - Integration with device GPS capabilities using the `expo-location` library for accurate location-based features.
- Custom API Integration - Interacts with backend services to fetch traffic data and predictive analytics.

Users of the Traffic Insights App can expect an intuitive experience while interacting with traffic-related data. They can navigate city streets, explore traffic intersections, search for specific intersections by name or address, and gain insights into future traffic conditions to plan routes effectively.

3.5 Traffic Data Service

The Traffic Data Service is a critical component of the traffic insights part, providing essential functionality for retrieving intersection-specific traffic data and predictions.

The primary purpose of the Traffic Data Service is to serve as a middleware layer between the frontend mobile application and backend data sources. It facilitates the retrieval of intersection information, including statistics such as mean vehicle count, mean pedestrian count, and operational intervals, as well as predictive analytics for future traffic conditions.

The `main` function initializes an HTTP server, exposing an endpoint `/get_intersection_info` to handle requests for intersection information (see Listing 3.8).

```
package main

import (
     "fmt"
     "net/http"
     "traffic-data-service/core"
)

function main() {
     http.HandleFunc("/get_intersection_info", core.GetIntersectionInfoHandler)

     fmt.Println("Server is listening on port 9000...")
     if err := http.ListenAndServe(":9000", nil); err != nil {
         fmt.Printf("Error starting core: %s\n", err)
     }
}
```

Listing 3.8 – Main Handler of the Service

The `GetIntersectionInfoHandler` function processes incoming requests for intersection information. It retrieves statistics and predictions for a specified intersection ID and returns the data in JSON format.

The package includes functions to fetch statistics and predictions from external APIs. The `GetStatistics` and `GetPrediction` functions make HTTP GET requests to the corresponding endpoints and parse the JSON responses.

The Traffic Data Service Package is built using the Go programming language and uses standard HTTP server and client functionalities. It integrates with external APIs to fetch intersection statistics and predictions, enabling seamless data retrieval for the Traffic Insights mobile application.

3.6 Gateway Service

In the traffic monitoring system, the Gateway plays an important role in easing communication between various components and external services. It serves as a centralized interface for handling

incoming requests, orchestrating data flow, and coordinating interactions with traffic analytics and regulation services.

The Gateway is implemented as a Flask application, using its simplicity and flexibility for handling HTTP requests. It consists of two main routes: `/gateway/add_traffic_record` and `/gateway/get_intersection_info`. The former receives incoming traffic records from sensors or other components, while the latter allows external clients to retrieve information about the intersection.

Threading is utilized to concurrently handle requests to different services, ensuring optimal performance and responsiveness.

When a traffic record is received at `/gateway/add_traffic_record`, the Gateway extracts the data and constructs payloads for sending to the traffic analytics and regulation services. Parallel threads are spawned to send requests to each service asynchronously, allowing for efficient utilization of resources and minimizing response times. Error handling mechanisms are implemented to capture and any exceptions that may occur during request processing, ensuring robustness and fault tolerance. The endpoint is illustrated in Listing 3.9.

```
@app.route('/gateway/add_traffic_record', methods=['POST'])
def gateway():
    data = request.get_json()

    service_a_url = 'http://traffic-analytics-service:8000/add_traffic_record'
    service_b_url = 'http://traffic-regulation-service:7000/add_traffic_record'

    responses = [None, None]
    errors = [None, None]

    threads = []
    for i, url in enumerate([service_a_url, service_b_url]):
        thread = threading.Thread(target=send_request, args=(url, data, responses,
errors, i))
        thread.start()
        threads.append(thread)

    for thread in threads:
        thread.join()

    for error in errors:
        if error:
            return jsonify({'error': error}), 500

    return jsonify({'responses': responses}), 200
```

Listing 3.9 – Gateway Traffic Record Handler

The Gateway integrates seamlessly with external traffic analytics and regulation services through HTTP requests. It abstracts away the complexity of communication protocols and data formats, providing a unified interface for interacting with these services. Responses from the external services are collected before being returned to the client, ensuring consistency in the data presented.

The Gateway is designed to be scalable and extensible, capable of accommodating future enhancements and additions to the system. It follows a modular architecture, allowing for easy integration of new services or functionalities without disrupting existing operations.

3.7 Traffic Light Simulator

The Traffic Light Simulator is an important component of our traffic monitoring system, responsible for emulating the behavior of traffic lights at an intersection. It uses modern web technologies, particularly React.js, to create an interactive and dynamic user interface that simulates the operation of traffic lights in real-time.

The Traffic Light Simulator is implemented as a React application, utilizing functional components and hooks for managing state and side effects. It follows a modular and component-based architecture, with separate components for traffic lights, timers, and data management. The application state is managed using React's built-in `useState` and `useReducer` hooks, providing a simple and declarative way to update and synchronize state across components.

The simulator mimics the behavior of traffic lights by cycling through different states, including red, green, and yellow. It maintains an internal state to track the current index of the active traffic light and the remaining time for each phase. A `useEffect` hook is used to transition between traffic light states based on predefined durations for red and green lights. When the timer for a particular phase expires, the simulator automatically switches to the next phase, updating the UI to reflect the change.

The Traffic Light Simulator integrates with external traffic regulation services to fetch traffic rules and configuration parameters dynamically. It periodically sends requests to the traffic regulation service to retrieve updated values for red and green light durations. Upon receiving new traffic rules, the simulator adjusts its internal state accordingly, ensuring that the simulation reflects the latest traffic regulations and timing constraints.

The user interface of the Traffic Light Simulator provides a visual representation of the intersection and the status of each traffic light. It displays colored circles to represent the traffic lights, with the opacity adjusted to indicate the active light. A timer indicates the remaining time for the current phase, updating in real-time as the simulation progresses. The interface is designed to be intuitive and user-friendly, allowing users to easily monitor and interact with the simulated traffic lights. Figure 3.6 shows the visual representation of the traffic light.

One of the key features of the Traffic Light Simulator is its ability to dynamically update the simulation based on external inputs and changes in traffic rules. By periodically fetching and updating traffic rules from the traffic regulation service, the simulator ensures that the simulation remains accurate and up-to-date with the latest regulations. This dynamic behavior enhances the realism and utility of the simulation, providing valuable insights into traffic flow and intersection dynamics.

The Traffic Light Simulator is designed to be scalable and extensible, capable of accommodating additional features and enhancements in the future. The modular architecture allows for easy integration of new components or functionalities, while React's component-based approach facilitates code reuse and maintainability. The simulator can be extended to support more complex traffic scenarios, such as multiple intersections or variable traffic patterns, by enhancing its logic and data management capabilities.

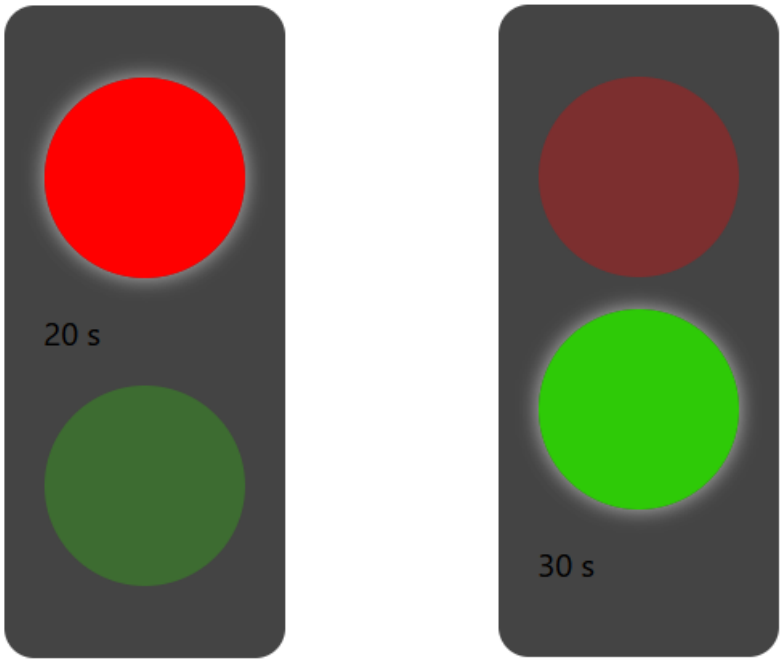

Figure 3.6 – Traffic Light Simulator Visual Representation

4 PRODUCT DOCUMENTATION

This chapter provides a detailed documentation for the Traffic Management System, outlining technical requirements, installation procedures, and user instructions. The Traffic Management System covers various components aimed at optimizing traffic flow, providing analytics insights, and offering real-time traffic updates to users. By following the instructions provided in this documentation, users will gain a thorough understanding of how to deploy, operate, and utilize each aspect of the system effectively.

4.1 Installation Requirements and Instructions

For optimal setup and functionality, it is essential to follow the specified order of installation:

- Run the Traffic Management System Containers
- Start the Traffic Light Simulator
- Launch the Traffic Monitoring Service
- Install and use the Traffic Insights App

The Traffic Regulation Service is responsible for implementing traffic regulations based on real-time data and analytics.

Technical Requirements:

- Docker image: `python:3.10.7`
- Port: `7000`

The Traffic Analytics Service provides analytics and insights into traffic patterns and trends.

Technical Requirements:

- Docker image: `python:3.10.7`
- Port: `8000`

The Traffic Data Service manages traffic data storage and retrieval.

Technical Requirements:

- Docker image: `golang:1.20`
- Port: `6000`
- Dependencies: Go 1.20

The Gateway Service acts as an interface for communication between microservices.

Technical Requirements:

- Docker image: `python:3.10.7`
- Port: `5000`

The Traffic Analytics Database stores traffic analytics data.

Technical Requirements:

- Docker image: `postgres:latest`
- Port: `5433`
- Environment variables: `POSTGRES_PASSWORD=111111`, `POSTGRES_DB=traffic-analytics`

Installation of the containers:

- Clone the repository containing the application source code
- Navigate to the root directory of the cloned repository
- Use Docker Compose to build and run the application: `docker-compose up –build`

The services will be deployed and accessible according to the specified ports

The Traffic Light Simulator is a React application that simulates traffic light behavior for testing and development purposes.

Technical Requirements:

- Development Framework: React Native

Installation:

- Clone the repository containing the application source code
- Navigate to the root directory of the cloned repository
- Install dependencies using npm: `npm install`, `yarn install`
- Run the simulator code to visualize traffic light behavior: `npm start`, `yarn start`

The Traffic Monitoring Service is responsible for real-time monitoring of traffic conditions, including vehicle counts, pedestrian activity, and traffic flow. It collects data from various sources such as cameras, sensors, and traffic light controllers, and provides insights to traffic management authorities.

Technical Requirements:

- Python version: 3.10.7
- Dependencies: Python 3.10.7, specified in requirements.txt

Installation:

- Clone the repository containing the application source code
- Navigate to the root directory of the cloned repository
- Install dependencies using pip and the requirements.txt file
- Run the service using Python

The Traffic Insights App is a mobile application available on both the Google Play Store and the Apple App Store. It provides users with real-time traffic information and predictions.

Technical Requirements:

- Development Framework: React Native
- Deployment Platforms: Google Play Store, Apple App Store

Installation:

- Clone the repository containing the application source code
- Navigate to the root directory of the cloned repository
- Install dependencies using npm: `npm install`, `yarn install`
- Configure deployment settings for both Android and iOS platforms

- Build the app for release
- Submit the app for review and publishing on the respective app stores

Installation on local machine:

- Clone the repository containing the application source code
- Navigate to the root directory of the cloned repository
- Install dependencies using npm: `npm install`
- Run the simulator code to visualize traffic light behavior: `npm start`, `yarn start`

4.2 User Manual

As an administrator, you have access to the following functionalities:

- Traffic Regulation: Visualize traffic regulations based on real-time data and analytics to optimize traffic flow and ensure safety. Access the Traffic Regulation Service through the designated port (7000). Use the provided data and insights to make informed decisions regarding traffic regulations.
- Traffic Analytics: Monitor traffic patterns snd trends to make data-driven decisions for improving traffic management strategies. Access the Traffic Analytics Service through the designated port (8000). Analyze traffic data to identify congestion hotspots, peak traffic hours, and other relevant insights.

As a user, you can interact with the following components:

- Traffic Insights App: Available on both the Play Store and App Store, the Traffic Insights App provides users with real-time traffic updates and predictions.

4.3 Real World Deployment

In a real-world scenario, the Traffic Management System would be deployed in urban areas with high traffic density and complex traffic patterns. Traffic lights equipped with PLC modules would be installed at intersections to regulate traffic flow dynamically based on real-time data from sensors, cameras, and other monitoring devices.

The Traffic Insights App would be available for download on both the Play Store and App Store, allowing users to access traffic updates, plan routes, and receive notifications about road conditions.

4.4 Conclusion

This documentation provides detailed information about the technical requirements, installation process, and user manual for the Traffic Management System. It defines the functionalities available to administrators and users, both in simulation and real-world scenarios, and describes the deployment options for various components of the application.

5 PROJECT MANAGEMENT VIEW

The chosen delivery approach for the traffic management software project is Agile. Agile methodology is well-suited for a dynamic project environment, where traffic conditions and user requirements are subject to continuous changes. This iterative and collaborative approach allows us to adapt to evolving scenarios, engage closely with stakeholders, and prioritize features based on real-time feedback. Agile aligns seamlessly with the project's goal of providing effective traffic management in a rapidly changing urban landscape.

5.1 SMART Objectives

a) Objective 1: Optimize Traffic Flow

- Specific: Develop a traffic management software system using AI-driven real-time data analysis to optimize traffic flow in urban areas.
- Measurable: Achieve a minimum of 15% improvement in traffic flow at monitored intersections post-implementation.
- Achievable: Utilize advanced AI algorithms and real-time data to adapt traffic signals dynamically, responding to changing traffic conditions.
- Relevant: Aligns with the project's goal of mitigating urban traffic congestion and improving traffic management efficiency.
- Time-bound: Complete the optimization feature development and implementation within the first six months of the project.

b) Objective 2: Reduce Congestion

- Specific: Implement features to actively reduce traffic congestion by providing adaptive traffic signal control.
- Measurable: Achieve a 20% reduction in overall congestion levels within targeted urban areas six months after the software's launch.
- Achievable: Utilize predictive modeling and historical data analysis to identify congestion patterns and optimize routes in real-time.
- Relevant: Addresses the core business problem of congestion, contributing to economic savings and improved quality of life.
- Time-bound: Have congestion reduction features fully operational within the first eight months of the project.

c) Objective 3: Enhance User Experience

- Specific: Develop a user-friendly interface accessible to traffic authorities and the general public.
- Measurable: Attain a user satisfaction rating of at least 80% based on feedback three months after the software's launch.

- Achievable: Engage UI/UX designers to create an intuitive interface catering to diverse user needs.
- Relevant: Positive user experience is crucial for widespread adoption and success of the application.
- Time-bound: Complete UI/UX design and implementation before the official software launch.

d) Objective 4: Provide Real-Time Insights

- Specific: Implement features for real-time insights and alerts on traffic conditions to users and authorities.
- Measurable: Ensure real-time insights are delivered within a latency of 10 seconds from data acquisition.
- Achievable: Employ efficient data processing and communication protocols for timely delivery of critical information.
- Relevant: Real-time insights contribute to proactive traffic management and enhance overall software effectiveness.
- Time-bound: Have real-time insights feature fully operational within the first nine months of the project.

e) Objective 5: Achieve High User Adoption

- Specific: Develop and execute a comprehensive marketing campaign to promote the software.
- Measurable: Achieve a user adoption rate of 50% within the first year of the software's launch.
- Achievable: Implement strategic marketing initiatives, collaborate with local authorities for promotion, and address user concerns effectively.
- Relevant: High user adoption is crucial for software success and its impact on traffic management.
- Time-bound: Execute the marketing campaign and monitor adoption rates continuously throughout the project.

5.2 SWOT Analysis

Strengths: The application utilizes innovative technology, uzing AI and real-time data analysis to optimize traffic flow effectively. Its comprehensive features, including real-time traffic data and predictive insights, make it a robust solution. With the potential to significantly reduce congestion and improve commute times, the application offers tangible benefits. Moreover, its user-friendly design ensures accessibility to both traffic authorities and the general public.

Weaknesses: However, the initial development costs for implementing advanced AI-driven technology can be substantial. Ongoing maintenance, data management, and infrastructure upgrades may also pose resource-intensive challenges. Convincing users to adopt a new traffic management tool and addressing data privacy concerns may present initial hurdles.

Opportunities: There are opportunities for market expansion, as the global demand for traffic management solutions grows due to urbanization and congestion issues. Collaborations with local governments, transportation authorities, and technology companies can enhance our application's reach and

impact. Offering customization options for different cities and regions can cater to specific needs and preferences.

Threats: Despite these opportunities, regulatory changes in traffic regulations or data privacy laws could impact our application's operation. Rapid technological advancements may need continuous updates to stay competitive. Additionally, user resistance to adopting new technologies or skepticism about effectiveness could pose threats to widespread adoption.

5.3 Key Performance Indicators

Traffic Flow Improvement Percentage (Stage: Implementation and Ongoing Monitoring): Measures the percentage improvement in traffic flow at monitored intersections post-implementation. Directly reflects the effectiveness of our solution in reducing congestion and enhancing traffic conditions. Regular analysis of traffic data will quantify the impact of our system, showcasing the tangible benefits in terms of reduced wait times and improved traffic flow, contributing to a more efficient urban transportation network.

User Adoption Rate (Stage: Post-launch): Measures the rate at which the application is adopted by the target audience. Gauges the level of acceptance among commuters and authorities, indicating the application's relevance and usability. User adoption is a critical factor in the success of our project. A higher adoption rate signifies that our solution resonates with users, providing them with valuable insights and contributing to safer and more convenient travel experiences.

Reduction in Commute Time (Stage: Ongoing Monitoring): Measures the average reduction in commute time for users. Provides a clear indication of the practical impact of the software on daily lives. Reduced commute times signify the success of our system in optimizing traffic signals and minimizing delays. Ongoing monitoring will capture trends in commute time reductions, demonstrating the sustained benefits our solution brings to urban commuters.

Data Accuracy (Stage: Ongoing Monitoring): Ensures that the data provided by our system is reliable, crucial for both traffic authorities and users. The accuracy of data is important for making informed decisions. Regular checks and validations will be in place to ensure that the traffic information delivered by the system remains accurate and trustworthy.

Customer Satisfaction (Stage: Post-launch): Measures customer satisfaction through user surveys and feedback analysis. Understanding user satisfaction is crucial for continual improvement. Surveys and feedback analysis will provide valuable insights into user experiences, allowing to address concerns, implement enhancements, and ensure sustained user engagement.

Return on Investment (Stage: Post-launch): Quantifies the financial benefits derived from the project against the initial investment. Calculating ROI helps assess the economic viability of our project. It considers the positive impact on traffic management, potential cost savings, and overall efficiency gains, providing stakeholders with a comprehensive view of the project's financial success.

Requirements Stability Index (Stage: Planning and Throughout): Measures the stability of project requirements by comparing changes made to the initial requirements. RSI provides insights into the volatility of project scope. A stable requirements index indicates effective planning and allows the team to manage changes more efficiently, contributing to project success.

Velocity (Stage: Development Iterations in Agile): Measures the amount of work completed in each Agile iteration. Velocity in Agile reflects the team's productivity and helps in planning future iterations. It serves as a valuable metric for optimizing development processes and ensuring continuous progress.

Code Churn Rate (Stage: Development and Testing): Measures the frequency of changes in the source code. Monitoring code churn helps assess code stability and potential quality issues. It guides the team in maintaining code integrity and minimizing disruptions due to frequent changes.

Defect Density (Stage: Testing and Deployment): Measures the number of defects per lines of code. Defect density is a key indicator of software quality. Regular monitoring ensures early detection and correction of defects, contributing to the overall reliability and effectiveness of our traffic management solution.

Deployment Success Rate (Stage: Deployment): Measures the percentage of successful deployments without critical issues or rollbacks. Deployment success rate reflects the efficiency of our deployment processes. A high success rate indicates smooth implementations, minimizing disruptions and ensuring the availability of the system to users.

Post-Implementation Defect Rate (Stage: Post-deployment): Measures the number of defects reported by users after the software has been deployed. Post-implementation defect rate assesses the impact of the system on users. It guides post-deployment support activities and contributes to ongoing improvements.

Technical Debt (Throughout SDLC): Represents the effort required to fix incomplete or suboptimal parts of the code. Monitoring technical debt helps maintain code quality. Regular efforts to address technical debt contribute to the long-term sustainability and extensibility of our traffic management solution.

Sprint Burndown Rate (Agile): Tracks the completion of tasks throughout a sprint in Agile. Sprint burndown rate provides real-time visibility into task completion. It aids the Agile team in assessing progress, identifying potential bottlenecks, and making informed adjustments during each sprint.

5.4 Software Delivery Lifecycle Phases

In the initiation phase, several key deliverables are produced to lay the groundwork for the project. These include the project charter, which formally authorizes the project and outlines its objectives, scope, stakeholders, and high-level requirements. The business case provides a rationale for the project, including the problem statement, solution proposition, and expected benefits. Stakeholder analysis identifies and analyzes project stakeholders, their interests, and their influence. Additionally, the project scope statement defines the project's scope, objectives, deliverables, and constraints. Project management artifacts in this

phase include the project initiation document, which summarizes key project information, and the stakeholder register, which lists all project stakeholders and their relevant information.

During the planning phase, the project plan is developed, detailing the project's scope, objectives, schedule, budget, resources, and risk management plan. Requirements specification outlines the functional and non-functional requirements of the software, while the risk management plan identifies potential risks, assesses their impact and probability, and outlines mitigation strategies. Resource allocation assigns roles and responsibilities to project team members. Project management artifacts in this phase include the project management plan, which guides project execution, control, and closure, the risk register, which logs identified project risks, and the requirements traceability matrix, which links requirements to project objectives and test cases.

In the design and development phase, the main deliverables include functional software developed according to specified requirements, user interface prototypes to visualize the design, and technical documentation detailing the software architecture and codebase. Project management artifacts include the work breakdown structure, a hierarchical breakdown of project tasks and activities, the Gantt chart, a visual representation of project tasks, dependencies, and timelines, and the change request log, which records and tracks requested changes to the project scope.

During the testing and quality assurance phase, key deliverables include test plans outlining the scope, objectives, and approach to testing, test cases detailing test scenarios and scripts, defect reports recording identified software defects, and quality metrics providing quantitative measures of software quality. Project management artifacts include the test plan, which specifies the testing strategy, resources, and schedule, the issue log, which records identified issues and defects, and the quality metrics dashboard, which visually represents software quality metrics for monitoring.

In the deployment phase, the main deliverables include deployed software installed and operational in the production environment, user manuals providing guidance on using the software, and training materials for end-users and support staff. Project management artifacts include the deployment plan, which outlines the process and schedule for deploying the software, and the user acceptance testing sign-off, formal acceptance of the software by end-users after successful testing.

During the operation and maintenance phase, main deliverables include regular software updates, user support documentation, and maintenance reports documenting maintenance activities and their impact on the software. Project management artifacts include release notes communicating changes, updates, and bug fixes to users, the change control log recording and tracking all changes made to the software post-deployment, and incident reports documenting reported incidents and their resolution.

In the monitoring and evaluation phase, main deliverables include performance reports providing regular reports on the software's performance, user feedback analysis analyzing user feedback, surveys, and support tickets, and KPI metrics measuring key performance indicators such as traffic flow improvement

and user adoption. Project management artifacts include the performance dashboard, visually representing project performance metrics, the lessons learned report highlighting project successes, challenges, and recommendations for future projects, and the project closure report, formally closing the project and including lessons learned, final budget, and project evaluation.

These phases and deliverables provide a structured framework for planning, executing, and delivering the project successfully while adhering to best practices in project management. They help ensure that the project progresses systematically, and that key information and documentation are available throughout the project lifecycle.

5.5 Roles and Responsibilities

As the leader of the project, the Project Manager plays a crucial role in scope management. Their responsibilities include leading the overall scope management process, collaborating with stakeholders to gather requirements, overseeing the development of the scope statement, and monitoring and controlling changes to the scope. They ensure that the project stays on track and delivers the intended outcomes within the defined scope.

The Business Analyst engages with stakeholders to elicit and document requirements, ensuring that they align with the project objectives. They analyze these requirements to identify any gaps or discrepancies and contribute to the creation of the scope statement. By bridging the gap between stakeholders and the project team, they ensure that the project scope accurately reflects the needs and expectations of all parties involved.

Stakeholders provide valuable input on project objectives and requirements based on their perspectives and needs. They actively participate in scope validation activities to ensure that the project aligns with their expectations. By engaging with stakeholders throughout the project lifecycle, the project team can maintain alignment with stakeholder interests and priorities, fostering a collaborative approach to scope management.

The Technical Team collaborates in defining the technical aspects of the project scope, using their expertise to assess the feasibility and impact of proposed features. They provide valuable insights into the technical requirements and constraints, ensuring that the scope is realistic and achievable within the project's constraints. By working closely with other team members, they contribute to the development of a robust and feasible project scope.

The UI/UX Designer plays a crucial role in defining the user interface requirements of the project scope. They work closely with stakeholders to understand user needs and preferences, ensuring that design elements align with the overall project scope. By focusing on usability, accessibility, and user satisfaction, they contribute to the creation of an intuitive and engaging user experience that enhances the project's overall success.

The QA Tester validates that the defined scope aligns with the testing criteria, ensuring that all requirements are thoroughly tested and validated. They provide insights into the impact of changes on testing efforts, helping to identify potential risks and challenges early in the project lifecycle. By conducting comprehensive testing and quality assurance activities, they contribute to the delivery of a high-quality and reliable project outcome.

5.6 Scope Creation and Validation Process

During the project initiation phase, a collaborative effort involving the Project Manager, Business Analyst, and Stakeholders is undertaken. This phase utilizes methods such as workshops, interviews, and surveys to gather initial project requirements. The objective is to establish a clear understanding of project objectives and initial requirements to lay a solid foundation for the project.

The Planning phase involves the Project Manager, Business Analyst, and Technical Team. It occurs during project planning and focuses on developing a detailed project plan outlining scope, objectives, and constraints. This stage ensures that project planning is based on a well-defined scope, setting the stage for successful execution.

Throughout the development phase, the Project Manager, Business Analyst, Technical Team, UI/UX Designer, and QA Tester collaborate regularly. Their aim is to address potential scope changes or deviations promptly through regular collaboration and communication. This process ensures that the project remains aligned with the defined scope and that changes are managed effectively.

Before deployment, a formal validation process is conducted involving the Project Manager, Business Analyst, Stakeholders, Technical Team, UI/UX Designer, and QA Tester. This includes formal review meetings, demonstrations, and validation sessions with stakeholders. The purpose is to ensure that the developed solution aligns with stakeholder expectations and project objectives before deployment.

Throughout the project lifecycle, the Project Manager, Business Analyst, Technical Team, and Stakeholders are involved in a formal change control process. This process includes the submission of change requests, impact analysis, and approval. The objective is to control changes to the scope, avoiding scope creep, and maintaining project focus.

Stakeholders play a crucial role in providing input on project objectives and requirements. Their involvement ensures that the project meets their expectations and addresses their needs effectively.

Continuous collaboration among team members helps in identifying and addressing scope-related issues promptly. This ensures that the project stays on track and remains aligned with stakeholder expectations.

Formal change control processes ensure that any changes to the scope are thoroughly evaluated, and their impact is understood before implementation. This helps in maintaining project stability and minimizing the risk of scope creep.

Conducting scope validation before deployment ensures that the final product aligns with stakeholder expectations, reducing the risk of post-deployment issues and rework. This ensures that the project delivers the intended outcomes effectively.

5.7 Estimation Plan

In a Planned Iterative methodology, effective project planning and control rely heavily on accurate estimation of work. Several estimation techniques can be employed in this context to ensure thorough estimation of project tasks. Here are some commonly used techniques:

- Expert Judgment: Involves seeking input from individuals or groups with expertise in the specific area of the project, such as project managers, team leads, or industry experts. Useful for tasks where historical data is limited, and expert opinion holds significant value.
- Analogous Estimation: Relies on historical data from similar projects to estimate the duration or effort required for current tasks. Effective when the current project shares similarities with past projects in terms of scope and complexity.
- Parametric Estimation: Involves using statistical relationships between historical data and other variables (parameters) to calculate estimates. Suitable for tasks with clear and quantifiable parameters that correlate with effort or duration.
- Three-Point Estimation (PERT): Utilizes three estimates for each task – optimistic, pessimistic, and most likely – to calculate an expected duration or effort. Useful for tasks with high uncertainty, allowing for a more probabilistic estimate.
- Bottom-Up Estimation: Involves estimating each task individually and then aggregating these estimates to derive overall project estimates. Effective when the project is well-defined, and detailed task-level estimates are possible.
- Delphi Technique: Involves obtaining anonymous input from a panel of experts and iterating until a consensus is reached. Useful for tasks where diverse expert opinions are valuable, and there is a need to reduce bias.
- Reserve Analysis: Involves allocating contingency reserves to account for uncertainties in estimates. Essential for tasks where there is a high degree of uncertainty, helping to manage risks effectively.

For the Traffic Management Software Project, a combination of estimation techniques would be prudent. Given the innovative nature of the project, Expert Judgment and Analogous Estimation can provide valuable insights. Three-Point Estimation (PERT) may be beneficial for tasks with inherent uncertainty, while Bottom-Up Estimation can offer detailed estimates for well-defined components. Additionally, Reserve Analysis should be applied to manage risks associated with uncertainties in the project.

This approach ensures a comprehensive estimation strategy that considers both expert insights and historical data while addressing uncertainties in a dynamic project environment.

CONCLUSIONS

In today's hurried urban environments, managing traffic efficiently and ensuring road safety are critical challenges. Our exploration into the development of an intelligent traffic monitoring system reveals a integral approach that unifies new technologies, real-time analytics, and seamless collaboration with external services to revolutionize urban transportation management.

At the core of the system are advanced algorithms like YOLO for real-time object detection and SORT for multi-object tracking. These technologies empower our system to accurately identify and track vehicles, pedestrians, and traffic signals, providing detailed insights into traffic dynamics and behaviors.

The real-time analysis capabilities of our system enable authorities to make timely decisions for traffic management and regulation. By continuously monitoring traffic activities, the system offers actionable insights into traffic flow, congestion patterns, and safety risks, easing proactive measures to optimize traffic efficiency and improve road safety.

Moreover, the integration of the system with external traffic analytics and regulation services increases its functionality and impact. By exchanging data in real-time with these services, the system contributes valuable insights for traffic planning, optimization, and enforcement.

The deployment of the traffic monitoring system holds immense promise for improving urban mobility and safety. From reducing traffic congestion to increased pedestrian safety, its impact on urban transportation is profound.

Looking ahead, there are numerous opportunities for further research and development in intelligent transportation systems. These include exploring advanced machine learning techniques for improving detection accuracy, integrating additional sensor modalities for increased data collection, and developing predictive analytics models for traffic forecasting and congestion detection.

In conclusion, the traffic monitoring system represents a significant step forward in the quest for smarter, more efficient cities. By utilizing technology, data analytics, and collaboration, it offers a glimpse into the future of urban transportation management.

BIBLIOGRAPHY

[1] L. Wagner, "1868-2019: a Brief History of Traffic Lights", Accessed: Mar. 12, 2024. [Online]. Available: https://www.inclusivecitymaker.com/1868-2019-a-brief-history-of-traffic-lights/

[2] "Urban mobility in the EU." EUROPEAN COURT OF AUDITORS. Accessed: Feb. 25, 2024. [Online]. Available: https://www.eca.europa.eu/lists/ecadocuments/ap19_07/ap_urban_mobility_en.pdf

[3] C. Degliomini, "Traffic Congestion Costs Commuters Valuable Time And Has A Heavy Economic Cost, Costing $120 Billion A Year - Is This Company's A.I. The Solution?", Accessed: Feb. 25, 2024. [Online]. Available: https://www.accesswire.com/757275/traffic-congestion-costs-commuters-valuable-time-and-has-a-heavy-economic-cost-costing-120-billion-a-year-is-this-companys-ai-the-solution

[4] C. Degliomini, "Traffic Congestion Cost The US Over $81 Billion In 2022 - One AI-Driven Company Reports Providing The Necessary Data Analytics To Facilitate Better Transportation and Greenhouse Gas Management", Accessed: Feb. 25, 2024. [Online]. Available: https://www.accesswire.com/752669/traffic-congestion-cost-the-us-over-81-billion-in-2022-one-ai-driven-company-reports-providing-the-necessary-data-analytics-to-facilitate-better-transportation-and-greenhouse-gas-management

[5] "Smart Nation: Singapore's Intelligent Transport System (ITS)." The ASEAN Post Team. Accessed: Feb. 25, 2024. [Online]. Available: https://theaseanpost.com/article/smart-nation-singapores-intelligent-transport-system-its

[6] N. Vogiatzis, H. Ikeda, and W. Wibisono, "Architecture Design and Prototyping of Front-End Component For Integrated Transportation System." Nikolaos Vogiatzis. Accessed: Feb. 25, 2024. [Online]. Available: https://www.researchgate.net/publication/266353695_Architecture_Design_and_Prototyping_of_Front-End_Component_For_Integrated_Transportation_System

[7] S. Smith, G. Barlow, X.-F. Xie, and Z. Rubinstein, "SURTRAC: Scalable Urban Traffic Control." Accessed: Feb. 25, 2024. [Online]. Available: https://www.ri.cmu.edu/pub_files/2013/1/13-0315.pdf

[8] "SINGAPORE'S TRANSPORTATION SYSTEM." Civil Engineering Consortium IIT Roorkee. Accessed: Feb. 25, 2024. [Online]. Available: https://cec-iitr.medium.com/singapores-transportation-system-4a414caf7b34

[9] "Sistemul de Management al Traficului din București." PRIMARIA MUNICIPIULUI BUCURESTI DIRECTIA DE TRANSPORTURI DRUMURI SI SISTEMATIZAREA CIRCULATIEI. Accessed: Mar. 14, 2024. [Online]. Available: https://smartcitiesofromania.ro/wp-content/uploads/2017/02/2015-11.pdf

[10] “Semafoare inteligente, instalate la mai multe intersecții din Chișinău,” *știri.md*. Accessed: Mar. 14, 2024. [Online]. Available: https://stiri.md/article/social/semafoare-inteligente-instalate-la-mai-multe-intersectii-din-chisinau

[11] “Monolithic architecture.” Learnitweb. Accessed: Mar. 15, 2024. [Online]. Available: https://learnitweb.com/microservices-with-spring/monolithic-architecture/

[12] T. Kontio, “Service Oriented Architecture.” Accessed: Mar. 15, 2024. [Online]. Available: https://ttow0130.pages.labranet.jamk.fi/01.-SOA-%26-Microservices/01.soa/

[13] K. Jaiswal, “Microservices Architecture.” Accessed: Mar. 15, 2024. [Online]. Available: https://blog.knoldus.com/microservices-architecture/

[14] “UART vs I2C vs SPI – Communication Protocols and Uses.” yida. Accessed: Mar. 15, 2024. [Online]. Available: https://www.seeedstudio.com/blog/2019/09/25/uart-vs-i2c-vs-spi-communication-protocols-and-uses/

[15] “Arduino.” Wikipedia. Accessed: Mar. 15, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Arduino

[16] “Raspberry Pi.” Wikidata. Accessed: Mar. 15, 2024. [Online]. Available: https://www.wikidata.org/wiki/Q245

[17] “Advantech UNO-2484G V2.” Accessed: Mar. 15, 2024. [Online]. Available: https://www.advantech.com/pt-br/products/1-2mlj9a/uno-2484g-v2/mod_9764124e-aff0-4e44-80d5-6fd87eec5982

[18] “Siemens SIMATIC S7-1200 PLC.” Mantis Systems. Accessed: Mar. 15, 2024. [Online]. Available: https://mantis-sys.co.nz/product/siemens-simatic-s7-1200-plc/

[19] “PlantUML.”

[20] A. Rosebrock, “YOLO object detection with OpenCV”, Accessed: Apr. 25, 2024. [Online]. Available: https://pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

[21] J. Mccarthy, “Object Tracking with YOLOv5 and SORT”, Accessed: Apr. 25, 2024. [Online]. Available: https://medium.com/@jarrodmccarthy12/object-tracking-with-yolov5-and-sort-589e3767f85c

[22] N. Beheshti, “Random Forest Regression”, Accessed: Apr. 24, 2024. [Online]. Available: https://towardsdatascience.com/random-forest-regression-5f605132d19d

[23] V. Lyashenko, “How to use random forest for regression”, Accessed: Apr. 25, 2024. [Online]. Available: https://cnvrg.io/random-forest-regression/

ANNEXES

Annex A

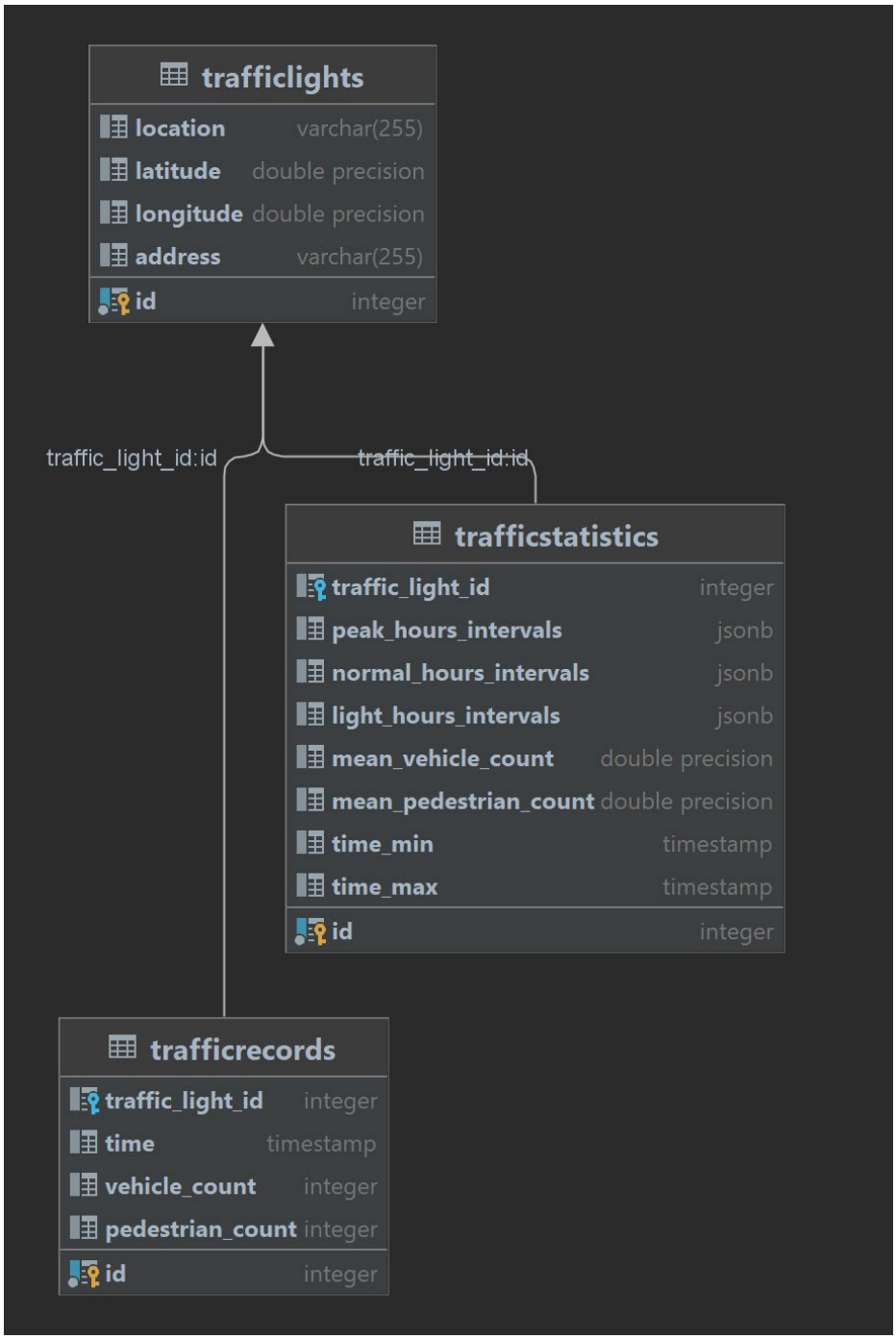

Figure A.1 – Database Schema

```
import psycopg2
import psycopg2.pool

Config class:
     DATABASE = {
         'dbname': 'traffic-analytics',
         'user': 'postgres',
         'password': '111111',
         'host': 'localhost',
         'port': '5433'
```

```
    }

# Connection pool setup for managing database connections
db_pool = psycopg2.pool.SimpleConnectionPool(
    false=1,
    maxconn=10,
    **DATABASE Config
)

def get_connection_from_pool():
    return db_pool.getconn()

def return_connection_to_pool(conn):
    db_pool.putconn(conn)

def setup_database():
    # Function to set up database schema and tables
    drop_tables = """
        DROP TABLE IF EXISTS TrafficRecords;
        DROP TABLE IF EXISTS TrafficStatistics;
        DROP TABLE IF EXISTS TrafficLights;
    """

    create_traffic_light_table = """
        CREATE TABLE IF NOT EXISTS TrafficLights (
            id SERIAL PRIMARY KEY,
            location VARCHAR(255),
            latitude FLOAT,
            longitude FLOAT,
            address VARCHAR(255)
        );
    """

    create_traffic_records_table = """
        CREATE TABLE IF NOT EXISTS TrafficRecords (
            id SERIAL PRIMARY KEY,
            traffic_light_id INTEGER REFERENCES TrafficLights(id),
            time TIMESTAMP,
            vehicle_count INTEGER,
            pedestrian_count INTEGER
        );
    """

    create_traffic_statistics_table = """
        CREATE TABLE IF NOT EXISTS TrafficStatistics (
            id SERIAL PRIMARY KEY,
            traffic_light_id INTEGER REFERENCES TrafficLights(id),
            peak_hours_intervals JSONB,
            normal_hours_intervals JSONB,
            light_hours_intervals JSONB,
            mean_vehicle_count FLOAT,
            mean_pedestrian_count FLOAT,
            time_min TIMESTAMP,
            time_max TIMESTAMP
        );
    """

    try:
        conn = psycopg2.connect(**Config.DATABASE)
        cursor = conn.cursor()
        cursor.execute(drop_tables)
        conn.commit()
        cursor.execute(create_traffic_light_table)
        conn.commit()
```

```
        cursor.execute(create_traffic_records_table)
        conn.commit()
        cursor.execute(create_traffic_statistics_table)
        conn.commit()
        cursor.close()
        conn.close()
        print("Database setup completed.")
    except psycopg2.Error as e:
        print(f"Error setting up database: {e}")
```

Listing A.1 – Database Setup and Configuration

Annex B

```
from flask import Flask, jsonify, request
import threading
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

app = Flask(__name__)

# Endpoint to retrieve traffic light data
@app.route('/get_traffic_lights', methods=['GET'])
def get_traffic_lights():
    # Database query to retrieve traffic lights data
    # Construct JSON response with traffic lights information
    return jsonify(traffic_lights_data), 200

# Endpoint to add traffic records to the database
@app.route('/add_traffic_record', methods=['POST'])
def add_traffic_record():
    # Insert traffic records into the database
    # Update real-time data cache for monitoring
    return jsonify({'message': 'Data added successfully'}), 200

# Endpoint to retrieve statistics for a specific traffic light ID
@app.route('/get_statistics/<int:traffic_light_id>', methods=['GET'])
def get_statistics(traffic_light_id):
    # Retrieve statistics for the specified traffic light ID from cache or database
    return jsonify(statistics_data), 200

# Endpoint to retrieve predictions for vehicle counts
@app.route('/get_predictions/<int:traffic_light_id>', methods=['GET'])
def get_predictions(traffic_light_id):
    # Retrieve predictions for vehicle counts using machine learning models
    return jsonify(predictions_data), 200

if __name__ == '__main__':
    app

.run(host="0.0.0.0", port=8000, debug=False)
```

Listing B.1 – Traffic Analytics Endpoints

Annex C

```
def process_real_time_data(date):
    try:
        vehicle_count = data["vehicle_count"]
        pedestrian_count = data["pedestrian_count"]
        traffic_light_id = data["traffic_light_id"]

        with lock:
```

```
            # Check for emergency conditions based on traffic counts
            if (vehicle_count > 2 *
traffic_intersection_data[traffic_light_id]["mean_vehicle_count"]) or (
                    pedestrian_count > 2 *
traffic_intersection_data[traffic_light_id]["mean_pedestrian_count"]):
                emergency_rules[traffic_light_id] = True
            otherwise:
                emergency_rules[traffic_light_id] = False
                # Determine current traffic period (peak, light, normal)
                current_stats = traffic_intersection_data[traffic_light_id]
                current_period = get_current_time_period(current_stats)
                # Recommend optimal green and red light durations based on current
period
                green_duration, red_duration =
recommend_traffic_light_duration(current_period)

                # Update traffic light rules dynamically
                traffic_light_rules[traffic_light_id] = {"green_duration":
green_duration,
                                                         "red_duration":
red_duration}

        print(f"New Real-Time Data Processed - Intersection {traffic_light_id}\n")

    except Exception as e:
        print(f"Error processing real-time traffic data: {str(e)}")
```

Listing C.1 – Algorithm to Process Real Time Data

Annex D

```
// Example code snippet for Map Screen component
import React, { useEffect, useState } from "react";
import { Dimensions, Image, Modal, StyleSheet, Text, TextInput, View } from "react-
native";
import MapView, { Marker } from "react-native-maps";
import { fetchIntersectionData } from "../api/intersectionApi";

const MapScreen = ({ navigation }) => {
  const [currentLocation, setCurrentLocation] = useState(null);
  const [trafficLights, setTrafficLights] = useState([]);
  const [filteredTrafficLights, setFilteredTrafficLights] = useState([]);
  const [modalVisible, setModalVisible] = useState(false);
  const [selectedTrafficLight, setSelectedTrafficLight] = useState(null);

  useEffect(() => {
    const getLocationAndTrafficLights = async () => {
      // Get current location using GPS
      // Fetch intersection data from API
      const data = await fetchIntersectionData();
      setTrafficLights(data);
      setFilteredTrafficLights(data);
    };

    getLocationAndTrafficLights();
  }, []);

  const renderMarkers = () => {
    return filteredTrafficLights.map((trafficLight) => (
      <Marker
        key={trafficLight.id}
        coordinate={{ latitude: trafficLight.latitude, longitude:
trafficLight.longitude }}
        onPress={() => {
```

```
          setSelectedTrafficLight(trafficLight);
          setModalVisible(true);
        }}
      >
        <Image source={require("./../assets/resources/images/map-pin.png")}
style={styles.markerImage} />
      </Marker>
    ));
  };

  return (
    <View style={styles.container}>
      <MapView style={styles.map} initialRegion={/* Initial region coordinates */}>
        {renderMarkers()}
      </MapView>
      {/* Modal for displaying selected intersection details */}
      <Modal visible={modalVisible} animationType="slide">
        {/* Modal content with intersection details */}
      </Modal>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
  },
  map: {
    width: Dimensions.get("window").width,
    height: Dimensions.get("window").height,
  },
  markerImage: {
    width: 35,
    height: 35,
  },
});

export default MapScreen;
```

Listing D.1 – Key Components and Functionality of the Map Screen

```
// Example code snippet for Details Screen component
import React, { useEffect, useState } from "react";
import { SafeAreaView, StyleSheet, Text, View } from "react-native";
import { fetchIntersectionInfo } from "../api/intersectionApi";

const DetailsScreen = ({ route }) => {
  const { intersectionId } = route.params;
  const [intersectionData, setIntersectionData] = useState(null);

  useEffect(() => {
    const fetchIntersectionData = async () => {
      const response = await fetchIntersectionInfo(intersectionId);
      setIntersectionData(response);
    };

    fetchIntersectionData();
  }, [intersectionId]);

  return (
    <SafeAreaView style={styles.container}>
      <View style={styles.content}>
```

```
        {intersectionData ? (
          <>
            <Text>{`Intersection Nr.${intersectionId}`}</Text>
            {/* Display intersection details */}
          </>
        ) : (
          <Text>Loading...</Text>
        )}
      </View>
    </SafeAreaView>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
  },
  content: {
    flex: 1,
    justifyContent: "center",


 alignItems: "center",
  },
});

export default DetailsScreen;
```

Listing D.2 – Key Components and Functionality of the Details Screen