B.Sc. (Hons) in Software Development

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

TrafficVision

By
Rohan Sikder

April 26, 2024

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

Contents

List of Figures

Chapter 1

Introduction

This dissertation presents TrafficVision, a new web application developed to show bus system reliability and punctuality for public transport. Consistent bus schedules result in issues for commuters, the infrastructure and the environment of cities in urbanized areas where efficient transportation is a core element of everyday life. TrafficVision is built for these challenges with computer vision and web technologies, providing real-time information about bus service schedules and quality.

1.1 Context and Relevance

Urban mobility relies on public transportation, particularly bus systems. Their effective operation impacts the environmental sustainability and the daily life of the population. However, bus services often have delays and unpredictability that inconvenience passengers and lead to higher carbon emissions and congestion. Such inefficiencies hinder sustainable urban living. TrafficVision hopes to ease these issues by offering real time bus info so commuters, transportation authorities and urban planners can make the best choice possible.

1.2 Project Objectives

The primary objectives of TrafficVision are:

- To develop a web-based platform that offers real-time tracking and analysis of bus services.
- To enhance the punctuality and reliability of public transportation through data-driven insights.

- To improve the daily commuting experience by minimizing uncertainty and wait times for bus passengers.
- To assist in reducing traffic congestion and carbon footprint through efficient bus service management.

These objectives form the benchmarks against which the project's success will be evaluated, with specific metrics detailed in the subsequent chapters.

1.3 Dissertation Overview

This dissertation reports on the development of TrafficVision from concept to implementation, highlighting its challenges and creative solutions. It is structured into several key sections:

- **Chapter 1: Introduction** This chapter introduces TrafficVision, a web application aiming to improve public transport reliability by providing real-time bus service data. It outlines the urban mobility challenges and the project's objectives to enhance bus service punctuality, improve commuting experiences and support sustainable urban living.
- **Chapter 2: Methodology** The methodology chapter details the structured approach to software development adopted for the TrafficVision project, including the use of Kanban boards and Gantt charts for project management, Agile methodologies for iterative development and the use of GitHub for collaboration and version control.
- **Chapter 3: Technology Review** In this chapter, the technology choices for TrafficVision are justified, including the use of MongoDB for its flexible data schema and cloud integration, Python for computer vision tasks, YOLOv8 over TensorFlow for efficient real-time object detection, React for UI development and Node.js for server-side scripting.
- **Chapter 4: System Design** This chapter covers the system architecture and design, explaining the use of React for front-end deployment, serverless architecture for the processing server, Kafka for data flow management, MongoDB for data storage and the integration of these technologies to create a scalable and responsive system.
- **Chapter 5: System Evaluation** The system evaluation chapter assesses TrafficVision against its initial objectives, presenting results and discussing

the system's strengths and limitations. It examines real-time traffic monitoring capabilities, the reliability of public transportation data provided, user engagement and interaction with the system and areas for future development.

- **Chapter 6: Conclusion** The conclusion revisits the project's context and objectives, summarizing key findings from the system evaluation, the challenges faced and the innovative solutions implemented. It encapsulates the project's achievements and the potential for future work to enhance the TrafficVision system.

1.4 Project Resources

The project's source code and resources are available on GitHub, which served as the code repository throughout the development process. The repository URL is:

`https://github.com/ConorPadraigMurphy/FYP`

The repository contains the following main elements:

- **Codebase** - The complete source code for the TrafficVision application, including frontend, backend and computer vision components.
- **Documentation** - A comprehensive set of documentation detailing the system's setup, deployment and usage instructions.
- **Issue Tracker** - A record of issues, enhancements and tasks managed throughout the project lifecycle.

Chapter 2

Methodology of Software Development

2.1 Roadmaps and Kanban Boards

The workflow was effectively managed and visualized using Kanban boards in the project. From initial development phases to final testing, each task was represented by a card on the Kanban board in columns labelled To Do," In Progress," Testing," and Done." This method allowed an overview of project progress and fast identification of bottlenecks. Additionally, roadmaps were created to describe the overall strategy and project milestones. These roadmaps defined the steps towards short-term and long-term goals and ensured that development followed the project goals. The Kanban board used for the TrafficVision project can be accessed at `https://rohansikder4.atlassian.net/jira/software/projects/KAN/boards/1`, providing a real-time view of project management and task progression.

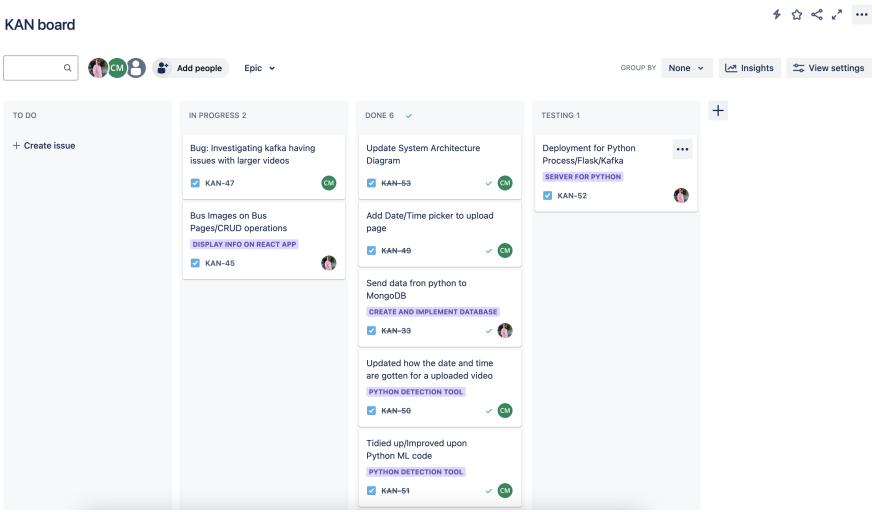

Figure 2.1: A Screenshot of the Kanban Board.

2.2 Gantt Chart Integration

To complement the agile project management approach and enhance visibility into the project timeline, a Gantt chart was employed. This tool was essential in planning, coordinating and tracking specific tasks against time. The Gantt chart for TrafficVision, available at `https://rohansikder4.atlassian.net/jira/software/projects/KAN/boards/1/timeline`, illustrates the project's timeline, including start and end dates for tasks, dependencies and milestones. This visual representation aided in ensuring that project deliverables were completed on schedule and that any potential delays were promptly addressed.

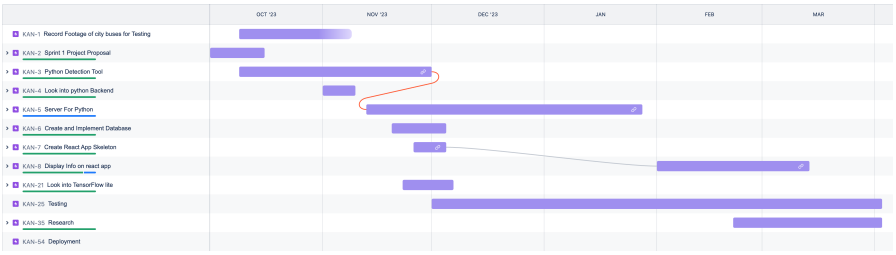

Figure 2.2: A Gantt Chart which was set out in the beginning of the development proccess.

2.3 Agile Methodologies

While primarily Kanban-based, the project also included elements of Agile[3] methodology with emphasis on iterative development and adaptability. Sprint meetings regularly reviewed progress, solved problems and planned future tasks, allowing for rapid iterations on the project scope based on real-time information and feedback.

2.4 Meetings and Communication

Effective communication was essential to the project. Regularly scheduled meetings facilitated continuous project problem-solving and iterative development.

- **Weekly Supervisory Meetings:** Weekly Meetings were held with the project supervisor to review progress and plan future tasks and address immediate issues. These meetings ensured the project met its objectives and offered opportunities for mentorship and guidance. Discussions often revolved around both strategic decisions and technical challenges for a comprehensive project oversight.
- **Weekly Planning Sessions:** Along with supervisory meetings, the team conducted Planning Sessions to outline work for the week. These sessions were critical in clarifying goals, identifying roadblocks and reallocating resources as needed to meet project milestones.
- **Code Review Meetings:** When new Code was written, team members scheduled special Meetings to Review the written Code. These sessions focused on assessing code quality, checking functionality meets project requirements and adding for knowledge sharing and learning. Such reviews helped to maintain high code quality and allowed for group ownership of the project.
- **Issue Resolution Discussions:** Any issues noted during the week or in code reviews were promptly addressed in private discussion sessions. These discussions were essential to troubleshooting issues, brainstorming solutions and delivering fixes on time to avoid project delays.

2.5 Collaboration Tools

The project used various tools to improve communication and productivity. GitHub[4] became the central repository for version control and source code management

and was used for reviewing work, facilitating collaboration among team members, regardless of their physical location. This integration of GitHub within the development process underscores its importance as a tool for code collaboration and version control in modern software development.

2.6 Rationale for Technology Stack

The project's choice of MongoDB, Python, React and Node.js as its technology stack was driven by their capabilities to deliver a high-performance web application. Each technology was selected for its strengths in specific areas of the project, from data management to user interface development, ensuring both efficiency and scalability.

2.7 Validation and Testing

Ensuring the quality and reliability of TrafficVision was carried throughout the development process. To achieve this, a comprehensive testing strategy was adopted, employing a mix of manual and automated testing approaches to cover the application's frontend, backend and integration points comprehensively.

2.7.1 API Testing with Postman

Postman[5] was especially important for backend validation, specifically the APIs that execute data transactions between the frontend and the database. A popular API test tool, Postman, allowed to simulate client-side requests to the server and check responses for correctness and performance. By writing structured tests in Postman to ensure all endpoints provided expected functionality, returned appropriate status codes and handled edge cases gracefully. This testing ensured that the server-side logic was robust, fault tolerant and ready to deliver intended user interactions.

Benefits and Outcomes

The use of Postman for API testing brought several benefits:

- **Rapid Feedback:** Quick feedback to reduce the time to detect and fix issues.
- **Improved Quality:** Consistent API testing led to higher quality software.

- **Enhanced Reliability:** Load and security testing ensured the APIs could handle real-world usage effectively.

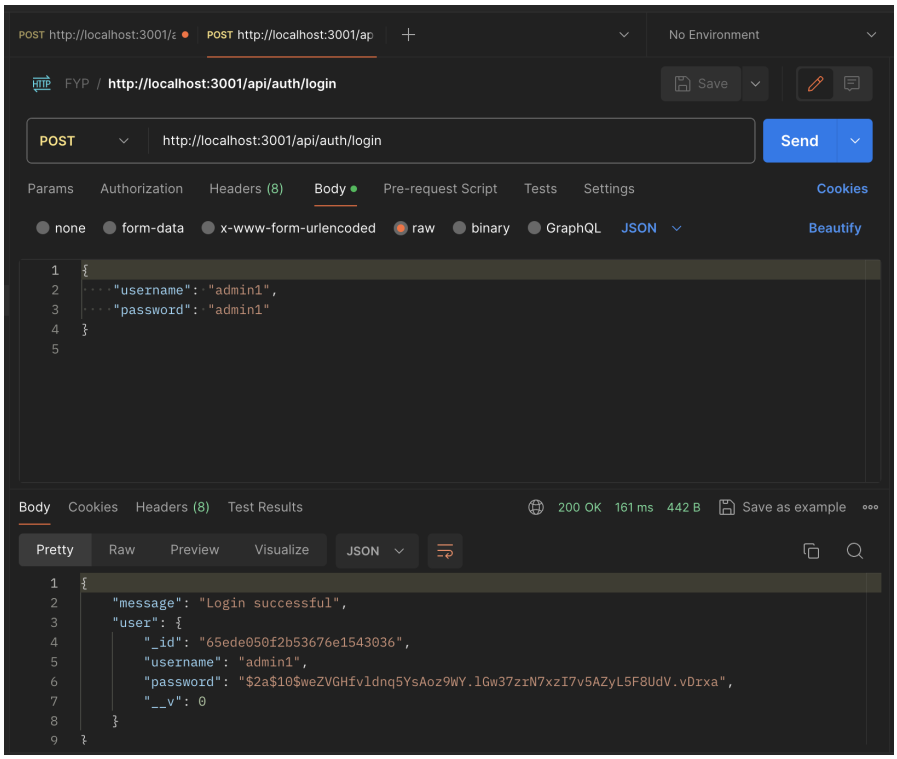

Figure 2.3: A Postman test demonstrating successful user login API call.

2.7.2 Developer-Led Frontend Testing

The frontend was developed with React and tested extensively by the developers that created it. This phase was primarily user testing focusing on the responsiveness, functionality and usability of the user interface. Developers carried out user interactions, navigated the application's features and examined the system behavior in real time. This approach allowed for fast detection and resolution of UI bugs or inconsistencies to ensure intuitive and seamless user experience.

This hands-on testing was complemented by incorporating the frontend and backend services to validate that data flow and processing across system layers worked as intended. This involved testing the real time data display, user authentication and traffic data submission and retrieval.

2.7.3 Ensuring Application Robustness

The combination of Postman for API testing and developers hands-on testing methodology for the frontend meant both critical parts of the TrafficVision application were tested. This methodology allowed for early detection of issues and created quality control.

The project's commitment to testing methodologies emphasized the need to provide a robust and bug-free application capable of enhancing public transportation systems efficiency and reliability. Future enhancements will include additional automated testing frameworks to expand the testing process and coverage to ensure TrafficVision continues to exceed user expectations.

Chapter 3

Technology Review

3.1 Deeper dive into Technology Selection

3.1.1 MongoDB: Flexibility and Cloud Integration

MongoDB's[6] schema-less nature offers the flexibility required for managing diverse data types, including geographical information and user profiles. Its horizontal scalability and robust cloud integration, especially with MongoDB Atlas, made it the preferred choice over traditional SQL databases like MySQL, despite their reliability lacked the same level of flexibility and required additional management for the data.

3.1.2 Python: Optimal for Computer Vision

Python[7] was chosen primarily for its extensive support in computer vision and machine learning tasks, thanks to libraries like OpenCV[8] and PyTorch. This made it exceptionally suited for implementing the YOLOv8 algorithm for real-time vehicle detection and tracking. While C++ could offer performance advantages, Python's balance of performance, ease of use and large library ecosystem presented the best fit for the project's needs.

3.1.3 Choosing YOLOv8[1] over TensorFlow for Object Detection

For the task of real-time vehicle detection and tracking, the project chose YOLOv8, based on the algorithm's faster speed and good object detection accuracy. YOLOv8, the latest YOLO series release, provides faster image processing for real-time applications. This feature is critical for TrafficVision, where timely data processing

affects the system performance for traffic monitoring. Though TensorFlow with its object detection API is a powerful and flexible framework that provides several pre-trained object detection models, including some based on the YOLO architecture, YOLOv8 was chosen due to its native video feed processing efficiency. YOLOv8's optimized model architecture minimizes computational consumption without sacrificing the sensitivity required for vehicle identification and tracking under varying lighting and weather conditions. Moreover, YOLOv8's simplified integration with the Python ecosystem made it easier to integrate into the project backend than TensorFlow's custom object detection setup. This ease of integration, combined with Python's deep support for other project requirements, justified moving forward with YOLOv8 for TrafficVision.

Summary: YOLOv8 was selected over TensorFlow due to the project's focus on real-time processing performance, performance and integration ease. Using the strengths of YOLOv8, TrafficVision achieves its goal of real time traffic monitoring, thereby increasing public transportation reliability.

3.1.4 React: Industry Standard for UI Development

The decision to use React[9] for developing the user interface was influenced by its widespread adoption and the extensive ecosystem surrounding it. Its component-based architecture simplifies the development of dynamic and responsive web applications. Although alternatives such as Angular and Vue.js were considered, React's flexibility, supported by a large community provided a more suitable solution for TrafficVision's interactive features.

3.1.5 Node.js: Efficient Server-Side Scripting

Node.js[10] was selected for its non-blocking, event-driven architecture, ideal for handling TrafficVision's real-time data processing requirements. Its seamless integration with JavaScript for both client and server-side development offers a unified development experience. Despite the availability of other backend frameworks like Django and Spring, Node.js stood out for its lightweight nature and scalability.

Summary: This carefully selected technology stack ensures that TrafficVision leverages modern, efficient solutions to address the challenges of real-time traffic monitoring and enhancing public transportation systems. The combination of MongoDB, Python, React and Node.js not only caters to the current project requirements but also sets a solid foundation for future scalability and enhancements.

3.2 Advanced Urban Traffic Monitoring with OpenCV and YOLOv8 in TrafficVision

TrafficVision's application of OpenCV and YOLOv8 represents a forefront approach in enhancing urban traffic surveillance through computer vision. This section includes the specific use of these technologies within the TrafficVision framework that showcase the project's innovative capabilities in traffic analysis and management.

3.2.1 Using OpenCV: The Foundation for Video Processing

At the core of TrafficVision's video data processing lies OpenCV, which is utilized to carry out preliminary video processing tasks essential for preparing data for detailed analysis.

- **Frame Extraction and Resizing**: Video frames are extracted from the uploaded streams using OpenCV and resized to a uniform resolution of 640x420 pixels. This standardization ensures consistent analysis across varying video inputs and optimizes the YOLOv8 model's performance by balancing detail against computational efficiency.

3.2.2 YOLOv8: Pioneering Object Detection and Tracking

Following initial processing with OpenCV, YOLOv8 undertakes the task of vehicle detection and tracking in the video frames, demonstrating significant advancements in real-time object recognition and tracking.

- **Real-time Object Recognition**: YOLOv8 excels in fast and accurate detection of vehicles within each frame, capable of distinguishing between different vehicle types. This ability allows TrafficVision to accurately classify and enumerate various vehicles, contributing to a comprehensive assessment of traffic composition.

- **Vehicle Tracking Through Frames**: YOLOv8 extends its functionality to track detected vehicles across frames, thereby facilitating an analysis of traffic flow patterns, speeds and potential congestion points. This continuous tracking is integral to understanding traffic dynamics and aids in effective traffic data.

3.2.3 Integration with Kafka for Real-Time Analysis

Kafka[11] is key in managing TrafficVision's data flow, creating the backbone for real-time data streaming and processing. The integration of Kafka demonstrates how TrafficVision leverages event-driven architectures for efficient video data management.

- **Event-driven Architecture**: Upon video upload, the associated metadata is encapsulated in a JSON object and transmitted to a Kafka topic named "incoming-videos". This event-driven methodology underscores Kafka's strength in facilitating scalable and real-time data pipelines.

- **Decoupling of Data Ingestion and Processing**: Through Kafka, TrafficVision achieves a decoupled architecture, allowing video ingestion and processing tasks to scale independently. This separation ensures the system's resilience and adaptability to fluctuating video data volumes.

3.2.4 Storing Analysis Results in MongoDB

The analytical outcomes, including vehicle counts, types and direction, are stored in MongoDB. Opting for this NoSQL database aligns with the project's need for scalable and flexible data storage solutions, capable of accommodating schema-less traffic analysis results.

3.2.5 YOLOv8 Conclusion

The integration of OpenCV and YOLOv8 within TrafficVision showcases a high-performance framework for real-time urban traffic analysis. By combining these cutting-edge computer vision technologies with Kafka's data management capabilities and MongoDB's dynamic storage solutions, TrafficVision presents a comprehensive approach to addressing urban traffic challenges. This detailed examination of TrafficVision's technological framework highlights its utilization of leading-edge tools to revolutionize traffic monitoring and management.

3.3 Flask Framework in TrafficVision's Backend Infrastructure

The Flask framework serves as a cornerstone in the development of TrafficVision's backend infrastructure, facilitating the seamless creation and management

of RESTful API services. This section dives into how Flasks functionality of TrafficVision, from handling video uploads to integrating with Kafka and MongoDB for real-time data streaming and storage.

3.3.1 Flask - Streamlining Video Uploads

At the heart of TrafficVision's Flask application is the capability to handle video uploads, a critical feature for the traffic data pool.

- **Video Upload Endpoint**: Flask's robust routing enables the definition of the '/upload' endpoint, where incoming video files are validated and processed. This ensures that each file is correctly formatted and present, showcasing Flask's efficiency in managing user inputs.

- **Data Management**: Utilizing Python's uuid module, Flask assigns a custom video ID to each upload, demonstrating the framework's ability for intricate data management tasks essential for applications dealing with substantial user inputs.

3.3.2 Implementing Kafka for Real-Time Data Handling

The integration of Kafka with Flask illustrates a sophisticated approach to real-time data streaming and processing within TrafficVision.

- Flask facilitates the encapsulation of video metadata into JSON objects, which are then published to a specific Kafka topic, initiating the asynchronous video data analysis pipeline. This highlights Flask's versatility in supporting microservices architectures and maintaining a scalable, decoupled system design.

3.3.3 Enhancing Data Management with MongoDB

Flask's adaptability extends to its integration with MongoDB, emphasizing the framework's capacity to support varied backend functionalities.

- By acting as the intermediary between video processing logic and MongoDB, Flask enables the efficient storage and querying of traffic analysis output. This exemplifies Flask's role in projects that require advanced data management capabilities for handling and retrieving extensive datasets.

3.3.4 Supporting Large Backend Solutions

TrafficVision's backend architecture, powered by Flask, showcases the framework's strengths in developing scalable and maintainable solutions.

- The project leverages Flask's lightweight and modular nature to achieve high efficiency and flexibility in backend services. Flask's design ensures that complexity does not compromise the simplicity of implementing API endpoints.

3.3.5 Flask Conclusion

The deployment of Flask within TrafficVision underscores the framework's efficacy in creating agile, scalable and efficient backend services for web applications. By using Flask for video upload functionalities, real-time data streaming with Kafka and data storage with MongoDB, TrafficVision establishes a robust backend infrastructure that not only fulfills present requirements but is also made for future expansion. This utilization of Flask highlights its critical role in the development and success of the TrafficVision project.

3.4 Apache Kafka Integration in TrafficVision

Apache Kafka's integration into TrafficVision exemplifies the application of real-time data streaming and event-driven architectures in urban traffic management systems. By leveraging Kafka, TrafficVision addresses complex requirements for data ingestion, processing and distribution, enhancing the system's ability to manage and interpret vast amounts of traffic data efficiently. This section outlines the specific use cases of Kafka within TrafficVision, emphasizing its critical role in facilitating real-time analytics and integration with other key technologies such as Flask, MongoDB and YOLOv8.

3.4.1 Kafka for Data Ingestion and Distribution

Kafka acts as the cornerstone of TrafficVision's data ingestion and distribution system, channeling video data and analytical results across various application components. Implemented within a Flask-based backend the Kafka producer interfaces with new video streams, recording essential metadata such as video ID, geolocation coordinates and timestamps alongside the actual video content. This setup showcases Kafka's capability to handle high-throughput data streams, allowing TrafficVision to scale its data ingestion efforts in response to fluctuations in traffic video data volumes.

3.4.2 Data Processing Workflows: Decoupling

A distinguishing characteristic of Kafka is its ability to decouple data producers from consumers, a feature that TrafficVision capitalizes on. By using Kafka topics as queues for incoming video data, the architecture allows the video processing unit to function independently of the data ingestion process. This separation enhances the system's availability and resilience, facilitating independent scaling of the video processing workload from data ingestion, thereby optimizing computational resource utilization.

3.4.3 Integrating Kafka with Flask and MongoDB

In TrafficVision, Kafka is utilized not only for data ingestion but also to initiate downstream processing activities. Upon the upload of a video, its metadata is dispatched to a Kafka topic, triggering the activation of the video processing unit. This unit, leveraging YOLOv8 for object detection and tracking, subsequently stores the processed data in MongoDB. The synergy between Kafka, Flask and MongoDB underscores Kafka's role as a robust message broker for asynchronous task execution, contributing to a modular and scalable architectural design.

3.4.4 Traffic Analysis in Real Time with YOLOv8

The real-time traffic analysis capability of TrafficVision is dependent on the YOLOv8 object detection model, complemented by Kafka's real-time data streaming services. As Kafka conveys video data to the processing unit, YOLOv8 analyzes each frame to identify and track vehicles, compiling crucial data on vehicle count, direction and speed. These insights are important for monitoring real-time traffic flow and congestion, showing Kafka's contribution to the enhancement of data analysis and decision-making processes.

3.4.5 Confluent Cloud Enhancement for Kafka in TrafficVision

Confluent[12] Cloud is a fully managed, cloud-native service that enhances Kafka functionality in TrafficVision. This integration solves the challenges of managing a high throughput data streaming infrastructure in the cloud for real-time urban traffic management

Cloud-Based Management and Scalability:

- **Automated Kafka Operations:** Confluent Cloud simplifies the operational management of Kafka clusters with automated provisioning, scaling

and management, ensuring that TrafficVision can efficiently handle varying data loads without manual intervention.

- **Enhanced Security and Compliance:** Utilizing Confluent Cloud, TrafficVision benefits from built-in security features like encryption, audit logs and compliance controls essential for handling sensitive traffic data securely.

Integrated Monitoring and Operations:

- **Confluent Control Center:** This tool from Confluent provides a comprehensive overview of Kafka's performance and stream data flows within TrafficVision. It helps in active monitoring and management of the system, ensuring high availability and reliability of the service.

Strategic Advantages:

- **Global Scalability and Reliability:** Confluent Cloud's multi-region deployment capability can improve TrafficVision by leaving room for expansion, offering low latency and high data throughput across geographical locations.

- **Resource Optimization:** The serverless nature of Confluent Cloud allows TrafficVision to optimize resource usage, adapting to traffic data demands in real-time, therefore reducing operational costs and enhancing system efficiency.

By using Confluent Cloud, TrafficVision not only strengthens its data streaming architecture but also aligns with modern cloud practices that enhance scalability, reliability and security. This strategic choice supports TrafficVision's commitment to delivering advanced, technology-driven solutions for urban traffic challenges, showing an innovative approach to traffic data management.

3.4.6 Kafka Conclusion

The deployment of Apache Kafka within TrafficVision shows the application of real-time data processing in traffic data. Through facilitating high-volume data ingestion, supporting a scalable and decoupled architectural framework and ensuring seamless integration with Flask, MongoDB and YOLOv8, Kafka empowers TrafficVision to deliver cutting-edge traffic intelligence. This comprehensive examination of Kafka's application in TrafficVision not only highlights its utility in contemporary web applications but also the project's commitment to advancing technology-driven solutions in urban problems.

3.5 MongoDB in TrafficVision's Data Architecture

MongoDB plays a big role in TrafficVision's data architecture, providing a robust and scalable solution to manage the platform's diverse and dynamic data sets. Its document-oriented and schema-less design affords the flexibility required to accommodate various data types, from user profiles to extensive traffic data.

3.5.1 Exploiting MongoDB's Document Model

TrafficVision utilizes on MongoDB's document model for its capacity to store different types of data within BSON documents, a choice that aligns seamlessly with the unstructured nature of traffic monitoring data.

BSON document

The following table illustrates an example of a BSON document used in TrafficVision to store traffic data, highlighting how MongoDB's document model supports complex and varied data structures:

Field	Value
_id	660acbd3867bb1af95cfdb4f
ObjectId	object_id
Int32	class_id
Bus	45
String	entered_time
Date	2024-01-0T12:00:44.500+00:00
direction	Left
address	Dublin Rd (Dawn Dairies), Galway, Co. Galway, Ireland
latitude	53.277507671611616
longitude	-9.013417301094194

Table 3.1: Example of Traffic Data Entry in MongoDB

This example underscores the flexibility of MongoDB's document structure, accommodating a variety of data types and formats essential for comprehensive traffic analysis.

- The document model is particularly adept at handling geospatial information and timestamped details of traffic videos, offering a hierarchical structure that captures the complexity of traffic data.

- Embedded documents and arrays within single documents enable structured recording of traffic incidents, vehicle movements and user interactions, enhancing data organization and accessibility.

3.5.2 Dynamic Data Schema

Using MongoDB's dynamic schema capability, TrafficVision efficiently adapts its data model in response to evolving platform needs without the need for costly database migrations.

- This feature supports the seamless introduction of new data types and the continuous innovation of TrafficVision's features, ensuring the platform remains at the forefront of traffic analysis technology.

3.5.3 Integration with Kafka and Express

The integration of MongoDB with Kafka for real-time data processing and Express for server-side logic forms a cohesive and scalable backend architecture.

- TrafficVision's backend, built on Express, utilizes MongoDB for efficient CRUD operations, leveraging its rich query language and indexing capabilities.
- Kafka serves as the wires in this architecture, ensuring a seamless data flow between frontend and backend components, thereby creating a reactive user experience.

3.5.4 Performance and Scalability

MongoDB's scalability features, such as sharding and replication, are crucial to TrafficVision given the voluminous traffic data it manages.

- These features enable TrafficVision to distribute data across multiple servers, maintaining high availability and consistency while facilitating horizontal scaling.

3.5.5 MongoDB Conclusion

MongoDB's flexible data model, dynamic schema and advanced querying capabilities are instrumental in meeting TrafficVision's complex data management requirements. By using MongoDB, TrafficVision gives insightful, real-time traffic analytics, showcasing the database's potential in modern web applications dealing with dynamic and varied data sets.

3.6 Secure Authentication in TrafficVision

In TrafficVision, ensuring the security of user data and maintaining system integrity is crucial. A key aspect of this security strategy is the implementation of bcrypt, a robust password hashing algorithm known for its effectiveness in securing passwords. The incorporation of bcrypt into the authentication flow underscores TrafficVision's following to industry-standard security practices, significantly avoiding the risk of data breaches.

3.6.1 bcrypt in Action: Enhancing Password Security

TrafficVision utilizes bcrypt to hash user passwords prior to their storage in the database, converting plain text passwords into hashes that are extremely difficult to decrypt. This section details the use of bcrypt and its impact on password security.

- **Password Hashing**: By transforming plain text passwords into hashed strings, bcrypt ensures that passwords stored in the database are protected against unauthorized access and decryption attempts.
- **Salting**: The addition of salt, a random value, to each password before hashing enhances security by preventing precomputed hash attacks, like rainbow table attacks. This method ensures that even if two users have the same password, their stored password hashes will be distinct.

3.6.2 Integration with Express and MongoDB

TrafficVision's backend, powered by Express, integrates bcrypt within its authentication routes for secure password handling during user registration and login processes.

- **User Registration**: During registration, bcrypt adds a salt to the password hash function automatically. The resulting hashed password, along with the username, is stored in MongoDB, guarding against potential database breaches.
- **User Login**: The bcrypt compare function is used to check if the submitted password matches the stored hashed password by encrypting the submitted password with the original salt and comparing the two hashes. Successful matches grant access, while mismatches result in authentication failure.

3.6.3 Secure Authentication Workflow

The secure authentication workflow implemented in TrafficVision leverages bcrypt, Express and MongoDB to protect user data effectively.

- **Secure Hash Generation**: Upon registration, a unique hash of the user's password is generated by bcrypt and stored in MongoDB, with each hash being uniquely salted.
- **Password Verification**: At login, the submitted password is hashed and compared with the stored hash, ensuring secure server-side verification without exposing plain text passwords.
- **User Data Protection**: bcrypt's hashing and salting capabilities enhance the security of TrafficVision, safeguarding user data from unauthorized access and fostering a secure user experience.

3.6.4 Secure Authentication Conclusion

The integration of bcrypt[13] in TrafficVision's authentication system reflects the project's strong commitment to security and privacy. By securely hashing and salting user passwords in conjunction with Express and MongoDB, TrafficVision not only protects users from potential data breaches but also shows its dedication to offering a secure, reliable traffic analysis and management platform.

3.7 Development Workflow Automation

In TrafficVision's development workflow, the utilization of script[14] files significantly contributes to time-saving and error reduction by automating repetitive tasks and ensuring consistency in command execution. The **start_app.sh** script serves as a example of how automation streamlines the setup and initialization of the development environment, providing several benefits:

- **Time-saving Automation**: By encapsulating multiple setup tasks within a single script, such as starting Kafka, creating Kafka topics, launching the Flask app and initiating the React frontend, developers can initiate the entire development environment with a single command (`./start_app.sh`). This eliminates the need for manual execution of individual commands saving valuable time and streamlining the development process.
- **Error Reduction**: The script ensures consistency in command execution, minimizing the likelihood of human errors. Each time the script is run it

executes the same sequence of commands in the same manner, eliminating variability and reducing the risk of configuration errors. This standardized approach enhances reliability and consistency across development environments.

- **Improved Developer Efficiency**: With the automation provided by the script, developers can focus their efforts on coding and problem-solving rather than repetitive setup and configuration tasks. This boosts overall developer efficiency and productivity enabling faster iteration and development cycles.
- **Enhanced Onboarding Experience**: For new team members joining the project, the **start_app.sh** script acts as a guide to initializing the development environment. By simply running the script new developers can quickly set up their local environment without the need for manual configuration, reducing the onboarding process.

The automation facilitated by script files like **start_app.sh** not only saves time and reduces errors but also gives developers efficiency, consistency and onboarding experience.

3.8 Dockerfile: Deployment Workflow Automation

In TrafficVision's deployment workflow, the Dockerfile [15] plays a big role in automating the setup of the production environment giving several advantages:

- **Streamlined Deployment Process**: The Dockerfile encapsulates the configuration and setup tasks required for deploying the TrafficVision application into a production environment. By creatings the necessary steps to build the Docker image including installing dependencies, setting up the Python environment and configuring application settings. The deployment process is streamlined and simplified.
- **Consistency Across Environments**: The Dockerfile ensures consistency in the deployment process across different environments from development to production. By specifying the exact dependencies and environment configurations required for the application, discrepancies between environments are minimized reducing the risk of deployment errors and inconsistencies.
- **Enhanced Scalability and Portability**: Docker containers created from the Dockerfile are lightweight, portable and scalable. The Docker image contains all the dependencies and configurations needed to run the TrafficVision

application allowing it to be easily deployed and scaled across different infrastructure environments including on-premises servers and cloud platforms.

- **Improved Deployment Reliability**: By automating the deployment process through the Dockerfile the likelihood of deployment errors and misconfigurations is significantly reduced. Each time the Docker image is built and deployed the same set of commands is executed consistently ensuring reliability and predictability in the deployment process.

Note: It's important to know that this Dockerfile was utilized during a trial deployment of the Python backend through Railway, although close was unsuccessful. This experience provides valuable insights for refining the Dockerfile and addressing any deployment challenges encountered in future deployment attempts.

Chapter 4

System Design

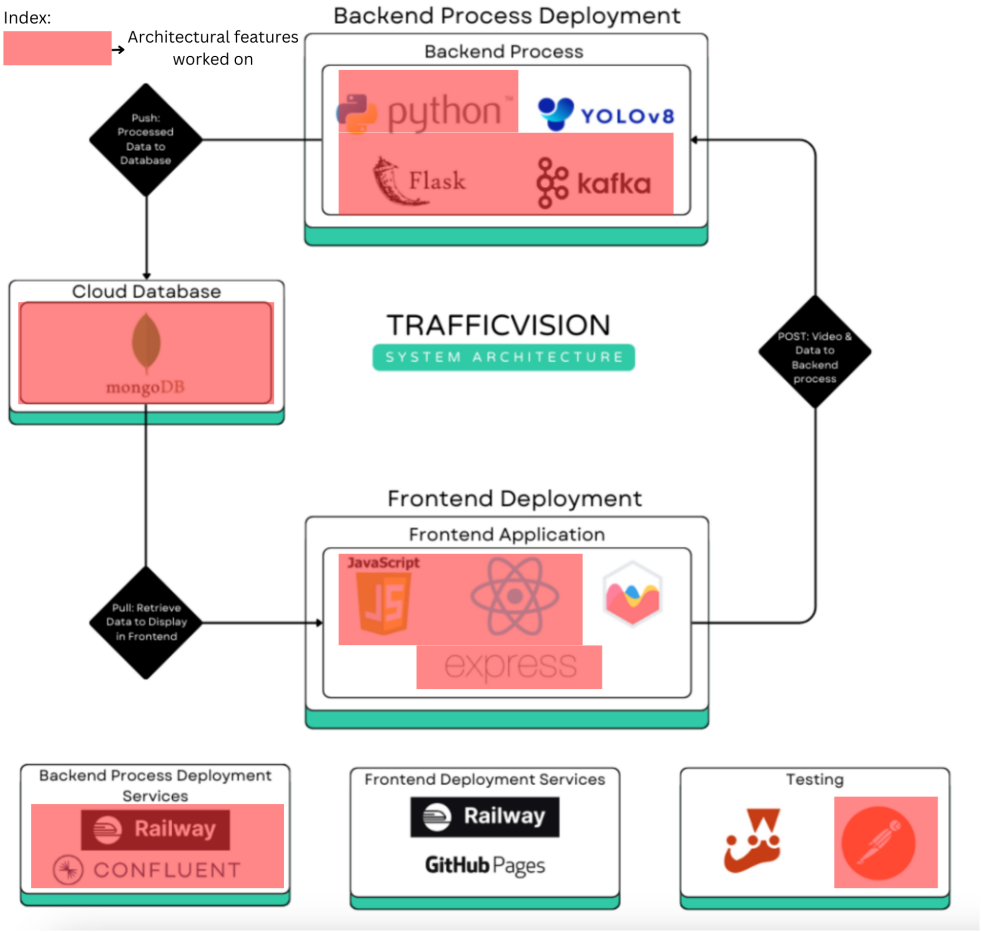

Figure 4.1: System Architecture.

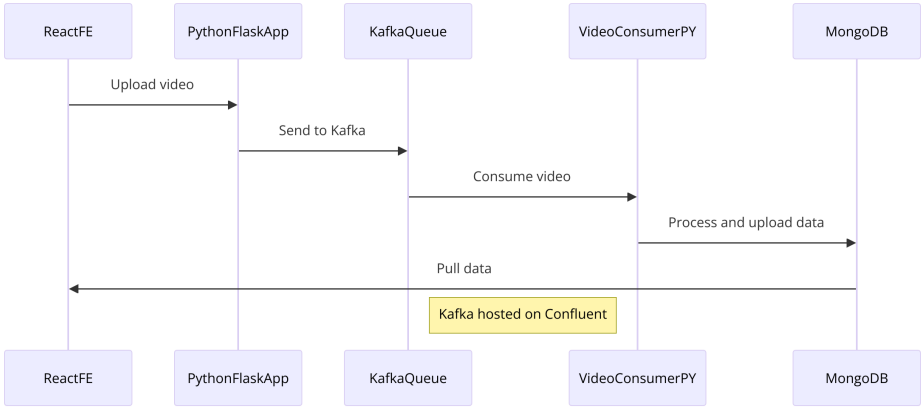

Figure 4.2: Flowchart[2] of TrafficVision System Operations.

4.1 TrafficVision System Architecture

4.1.1 Frontend Deployment

The frontend of TrafficVision is constructed using React, a JavaScript library for building user interfaces. The choice of React is attributed to its component-based architecture, which facilitates the development of interactive UIs.

- Users use this frontend to upload traffic videos and additional data into the system, initiating the data processing workflow.

4.1.2 Serverless Deployment: Processing Server

Upon user interaction with the frontend, video and data uploads trigger actions within a serverless processing server.

- The serverless model, functioning as a function-as-a-service (FaaS), dynamically executes backend code in response to events, in this scenario, video uploads, circumventing the need for a persistent server environment.
- The processing server operates on a Python runtime, renowned for its extensive libraries that support a multitude of web and data processing functionalities.
- Within this server, two technologies are utilized:
  - **YOLOv8 (You Only Look Once):** This state-of-the-art deep learning algorithm specializes in object detection within video frames in real-time, identifying vehicles and other objects promptly and accurately.

- **Apache Kafka:** A robust distributed event streaming platform that orchestrates real-time data feeds. Kafka manages the flow of data throughout the system components, particularly by queuing video processing tasks efficiently.

4.1.3 Cloud Database: MongoDB

Post-processing of video data, MongoDB, a NoSQL cloud database, is employed to manage the data output.

- MongoDB is chosen for its adaptable, schema-less data structure and powerful scalability, which is ideal for handling the vast and varied datasets of TrafficVision.

- The database serves as a repository for processed traffic data, which can be later fetched by the frontend for display and analysis purposes.

4.1.4 Kafka Deployment on Confluent

Apache Kafka's deployment within the TrafficVision system is facilitated through Confluent, which is a fully managed Kafka service that simplifies the process of running Kafka.

The following snippet of code simplifies the Kafka configuration for connecting to the Confluent service:

```
kafka_config = {
    "bootstrap.servers": os.getenv("KAFKA_BROKER_URL"),
    "security.protocol": "SASL_SSL",
    "sasl.mechanisms": "PLAIN",
    "sasl.username": os.getenv("KAFKA_API_KEY"),
    "sasl.password": os.getenv("KAFKA_API_SECRET"),
    ...
}
```

This configuration uses the Producer and Consumer components ensuring they communicate efficiently with the Kafka cluster hosted on Confluent. The TrafficVision system uses Confluent's managed Kafka solution to ensure high-throughput data processing and streamlining the real-time video analytics pipeline.

4.1.5 Data Flow

The operation of TrafficVision encompasses a series of coordinated processes:

- **Video Upload:** Utilizing the frontend application, users upload videos to the processing server, instigating an event that forwards the video and associated data to the server for processing.
- **Data Processing:** Leveraging serverless deployment, video data is processed in real-time. Kafka's queuing system allows for scalable and parallel processing of multiple video streams.
- **Data Storage:** Upon processing the data is stored in MongoDB, making it accessible for subsequent retrieval.
- **Data Retrieval:** The React-based frontend retrieves processed video data from MongoDB, presenting users with comprehensive traffic analysis and insights.

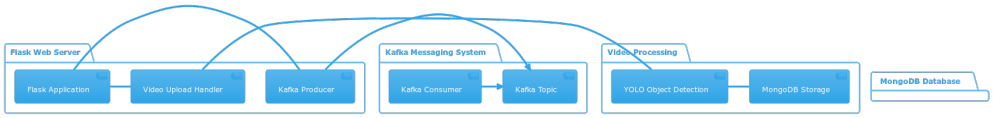

Figure 4.3: Map Diagram of the TrafficVision Data Flow.

4.2 Component Interaction and Code Correlation

The TrafficVision system's UML[16] diagram provides a detailed overview of the component interactions within the architecture. The diagram is essential in understanding the system's modularity and connectivity between components.

4.2.1 Flask Application

The `FlaskApplication` acts as the interface for client interactions. It handles HTTP requests including video uploads and assigns tasks to other components. The following code snippet represents the video upload handling within the Flask application, corresponding to the `route` method in the UML class:

```
@app.route("/upload", methods=["POST"])
def upload_video():
    # [Code to handle video upload]
```

4.2.2 Kafka Producer

The `Producer` sends messages to the Kafka cluster. Upon a successful video upload the producer sends a message containing the video ID and metadata, as shown in the code below:

```
try:
    producer.produce("incoming-videos", key=str(uuid.uuid4()), value=message)
    producer.flush()
except Exception as e:
    # [Error handling code]
```

4.2.3 YOLO Object Detection

The `YOLO` class represents the object detection component using the YOLOv8 algorithm. The code snippet below shows the model loading and video processing, which correlates with the UML diagram:

```
model = YOLO("yolov8n.pt")
# [Code to process video and detect objects]
```

4.2.4 MongoDB Storage

The `MongoClient` handles data storage in MongoDB. The process of inserting data into the database is represented in the following snippet:

```
if info_list:
    collection.insert_many(info_list)
    # [Code to handle database insertion]
```

4.2.5 Integration and Workflow

Integrating these components allows for a seamless flow from video upload to object detection and data storage. The provided code offers a concrete implementation of the conceptual UML structure.

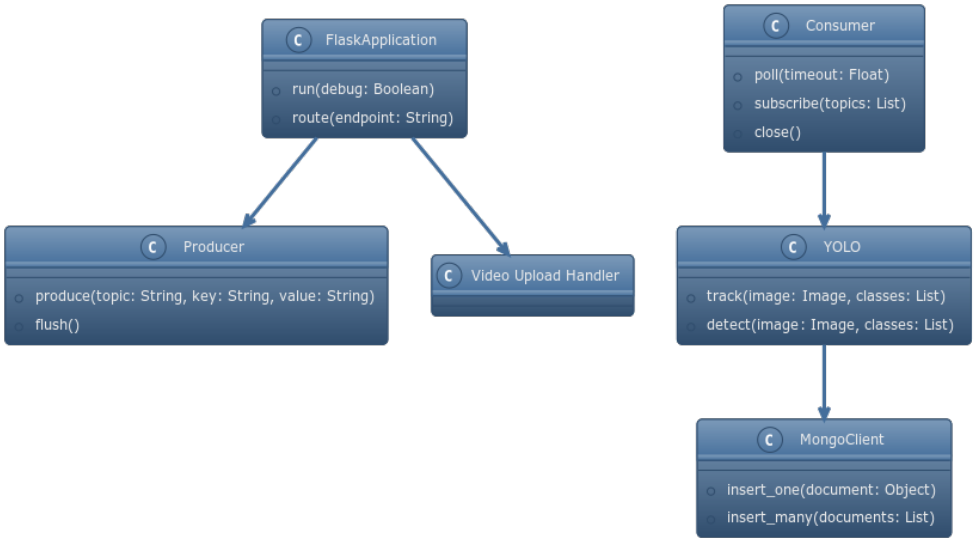

Figure 4.4: UML Diagram of TrafficVision Components.

This UML diagram showcases the individual responsibilities of each system component and its modularity within TrafficVision for efficient traffic video processing and analysis.

4.3 Environment Configuration with `.env` Files

4.3.1 Purpose and Benefits

The TrafficVision system uses `.env`[17] files to manage environment variables, which are crucial for defining the runtime environment configuration outside of the codebase. This approach separates configuration from code, which is a principle of a twelve-factor app methodology[18] leading to several advantages:

- **Security:** Sensitive information such as API keys and database credentials are kept out of the version control system reducing the risk of security breaches.
- **Portability:** Developers can create multiple `.env` files for different environments like development, testing and production making it easier to manage and deploy the application across various environments.
- **Customization:** Each instance of the application can be easily customized by changing the environment variables without altering the code.

4.3.2 Implementation in TrafficVision

The TrafficVision system utilizes the `dotenv` library for Python to load the configuration variables from the `.env` file when the application starts. The following code snippet shows how the environment variables are loaded:

```
from dotenv import load_dotenv
DOTENV_PATH = "./.env"
load_dotenv(dotenv_path=DOTENV_PATH)
```

This simple yet effective setup allows the application to access the environment variables as needed, for instance:

```
kafka_config = {
    "bootstrap.servers": os.getenv("KAFKA_BROKER_URL"),
    ...
    "sasl.password": os.getenv("KAFKA_API_SECRET"),
}
```

In this example the Kafka connection configuration is built using values sourced from the environment which allows for flexibility and security.

4.3.3 Implication

TrafficVision's system design utilizes cutting edge technologies and practices to make a flexible and effective traffic management solution. Front and center of its frontend is React's user friendly user interface, where traffic information can be uploaded. This user interaction launches a backend response in the backend which handles video data uploads dynamically.

YOLOv8's real time object detection and Apache Kafka's information streaming are central to the system processing abilities, all hosted in the scalable Confluent managed services environment. These technologies work together to provide quick and precise processing of traffic videos, queuing activities and smartly managing information flows.

Persistence and results of processed information are handled by MongoDB, a NoSQL cloud database with adaptable data structure and scalability which successfully handles the different produced data sets.

TrafficVision's environment configuration approach based on.env documents highlights the system's operational flexibility and security. By externalizing configuration detail, TrafficVision secures very sensitive information and offers flexibility to adapt to various deployment environments without refactoring the core codebase.

To sum up, TrafficVision's system architecture is a mix of intuitive user interfaces, serverless backend computing, information processing, and configuration management security. This sophisticated integration allows TrafficVision to offer effective traffic analysis.

Chapter 5

System Evaluation

This chapter evaluates the TrafficVision project against the objectives set out in the introduction. Here presenting the results obtained from the system's operation and discuss its strengths and weaknesses. This evaluation serves as a platform to demonstrate critical thinking in relation to the project and its outcomes.

5.1 Objective Alignment

The primary objectives of TrafficVision were to provide real-time traffic monitoring, enhance public transportation system reliability and improve user engagement through interactive features. The following sections detail how the system has met these objectives and highlight areas for further development.

5.2 Real-Time Traffic Monitoring

One of TrafficVision's core objectives was to implement real-time traffic monitoring. The system's ability to process and display live traffic conditions is illustrated in Figure 5.1.

Figure 5.1: TrafficVision's real-time traffic monitoring interface demonstrating vehicle detection and tracking.

The system successfully identifies and tracks vehicles with high accuracy and low latency, as shown in the terminal outputs in Figure 5.2. However, one weakness observed is the dependency on good lighting conditions for optimal performance, which could be addressed with enhanced image preprocessing techniques or the integration of infrared camera feeds for night-time monitoring. Future enhancements for the TrafficVision real-time monitoring system may include the integration of additional data sources such as traffic cameras across different locations, which would provide a more comprehensive traffic analysis. Additionally, machine learning techniques could be employed to predict traffic conditions based on historical data, providing users with not only the current state of traffic but also an intelligent forecast of short-term changes.

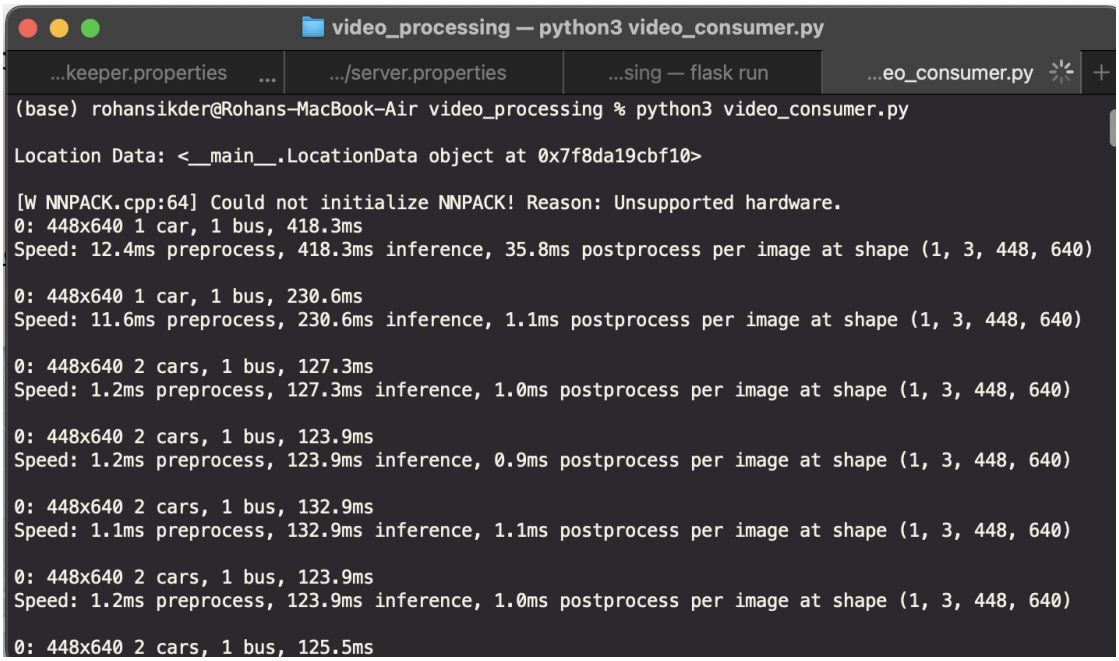

Figure 5.2: Performance metrics of the TrafficVision system captured from the terminal output.

5.3 Public Transportation System Reliability

Enhancing the reliability of public transportation was another key objective. TrafficVision contributes to this by analyzing bus punctuality, as depicted in Figure 5.3. The graph shows the frequency of bus appearances and can be used to assess punctuality against the schedule.

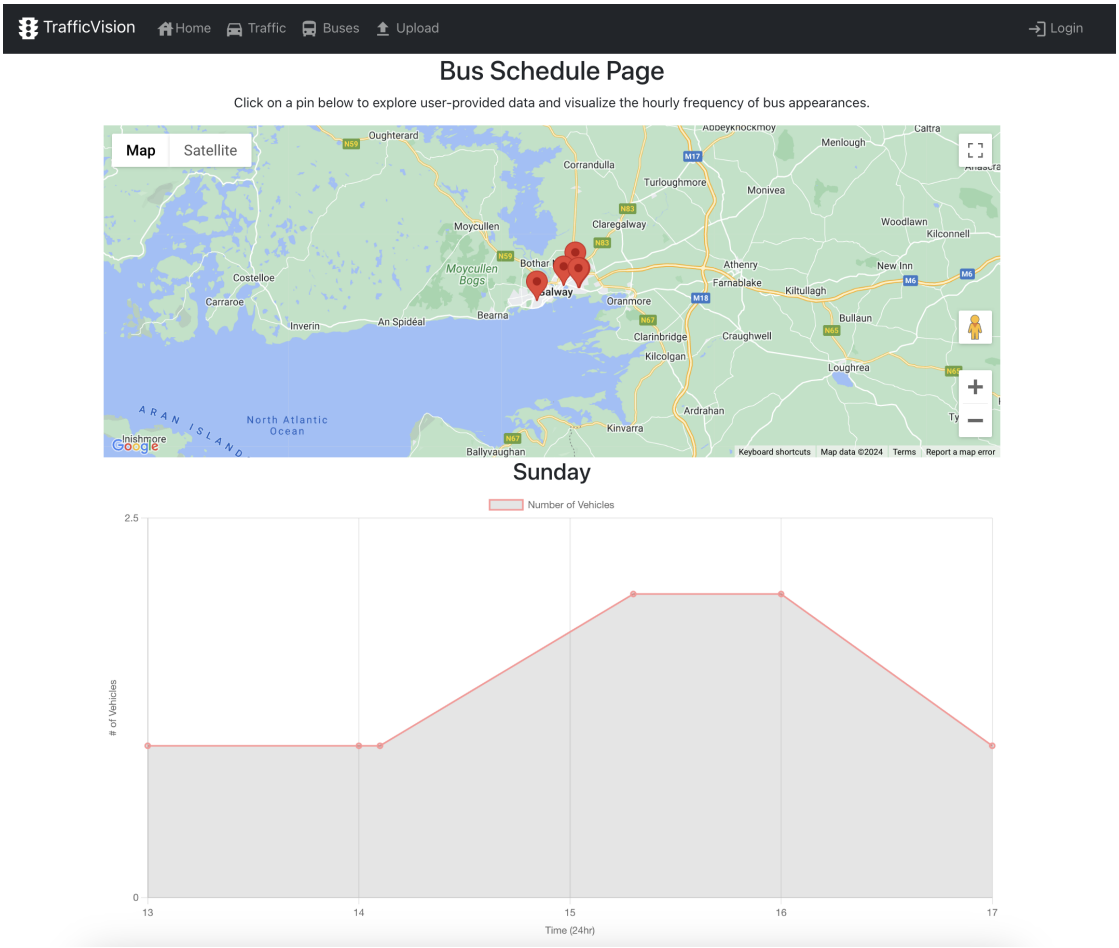

Figure 5.3: Visualization of bus schedule adherence using TrafficVision's data analytics.

While the system effectively records bus frequencies, the strength of the data is dependent on the quantity and quality of user submissions. This represents a limitation in data collection that could potentially be overcome with more widespread user participation or the incorporation of official transit data feeds.

5.4 Integration Challenges with Transport for Ireland API

The implementation of the Transport for Ireland API into the TrafficVision system was not a option due to the lack of documentation and complexity of the API. The

API, which provides real-time bus schedules with arrival and departure times, was critical to compare how late buses were.

After research on attempts to integrate this API due to the significant challenges faced, including the API being oriented towards internal use by Transport for Ireland. Restricted accessibility making it not suitable for the TrafficVision project.

Alternative data sources were looked at that could provide accurate real time bus schedules along with departure and arrival times. However, no suitable alternatives were identified that satisfied the project's specific accessibility, reliability and integration needs.

5.5 User Engagement and Interaction

User engagement is critical for crowd-sourced platforms like TrafficVision. The UI and UX are designed to be intuitive and engaging, as evident in the system's login page (Figure 5.4) and the traffic upload feature (Figure 5.5).

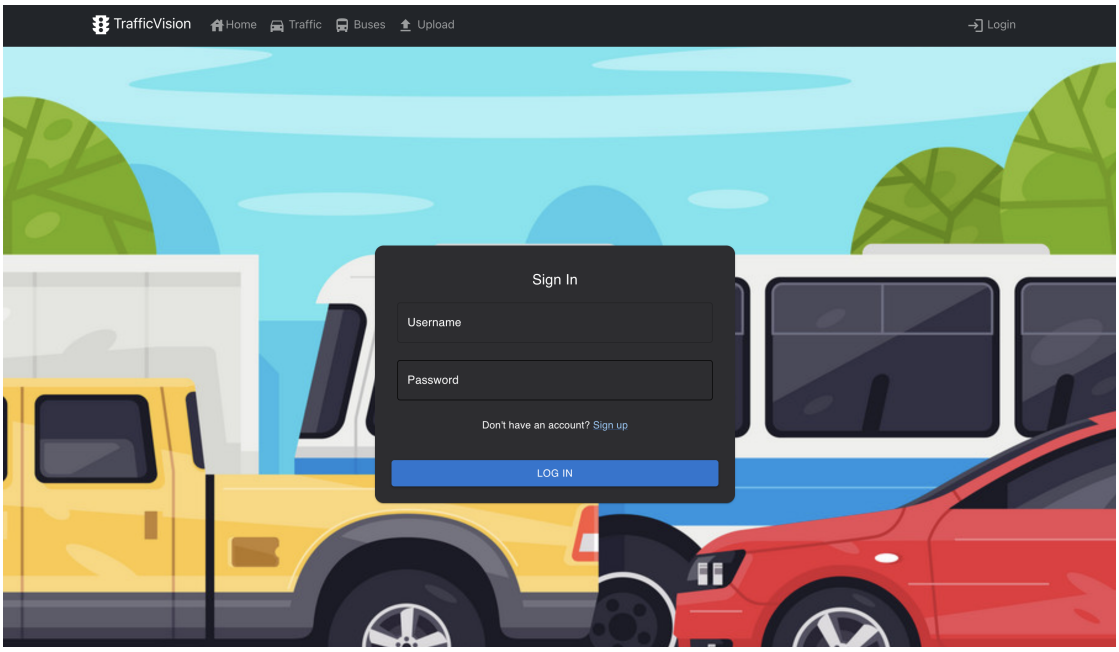

Figure 5.4: The login page of TrafficVision, showcasing the system's user-friendly interface.

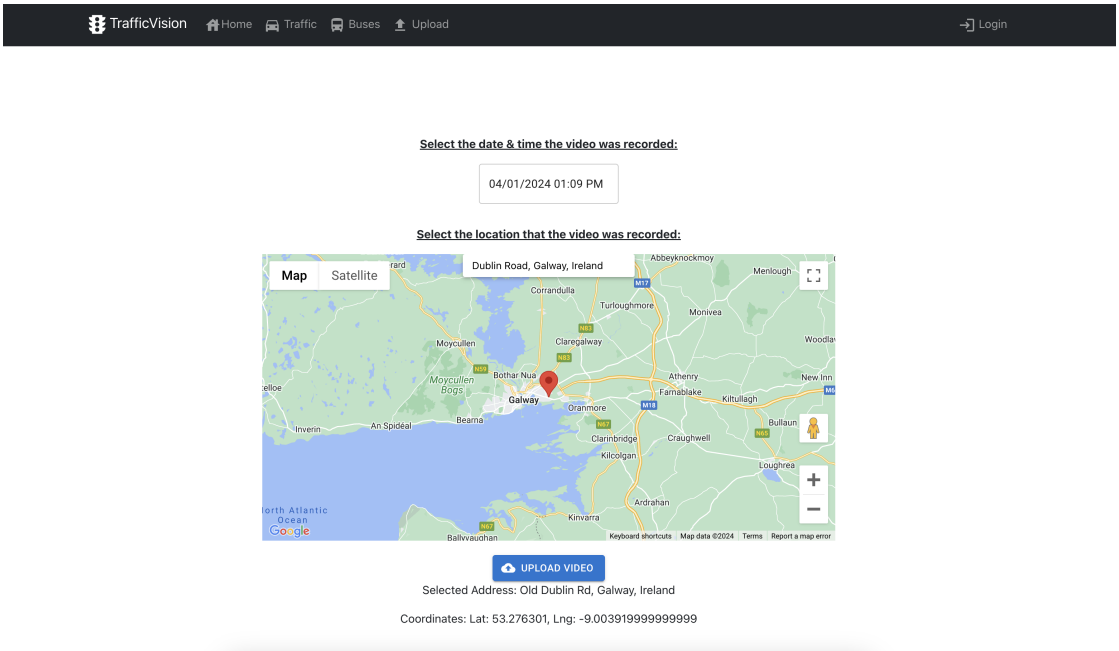

Figure 5.5: TrafficVision's video upload feature, enabling user participation in traffic data crowdsourcing.

While the system provides a straightforward mechanism for users to contribute data, a notable weakness is the reliance on active user engagement, which may lead to data sparsity. Future enhancements could include gamification elements to incentivize regular contributions or the use of passive data collection methods.

5.6 Deployment Challenges

The TrafficVision system deployment presented significant technical challenges, especially in handling the large scale video processing requirements. The initial deployment strategy used cloud services like AWS which provided extensive resources but had cost and scalability issues due to traffic volumes fluctuating.

To address these concerns, the project shifted toward using Railway[19], a platform that supports Docker, aiming to simplify the deployment process through containerization. This way we could have a more manageable configuration of our Python applications and dependency management.

Given challenges the challenges met in researching and developing a deployment, the most effective deployment of the TrafficVision system has been on local machines, where video processing can be managed more reliably.

In the future, optimization of the deployment architecture will be a priority, such as stronger cloud offerings or dedicated hardware to handle high video processing. This includes testing machine learning models optimized for resource efficiency and integrating adaptive streaming technologies.

5.7 Cross-Platform Development Challenges

A small team of two members worked on TrafficVision using two different operating systems: Unix and Windows. This configuration introduced special system compatibility and integration issues when using technologies like Confluent for Kafka streams, MongoDB for database management and Python for backend and video processing.

These compatibility issues highlighted the need for development practices that support both environments. The team considered several ways to coordinate their development efforts including:

- **Standardization of Development Tools:** The team focused on using cross-platform compatible tools and technologies to reduce system incompatibility issues. This approach was vital to ensure that all components of the TrafficVision system worked harmoniously across different operating environments.
- **Version Control Systems:** Strong version control practices became critical for effective change management and maintaining system stability across different operating systems. By employing robust version control, the team could manage code changes more efficiently, ensuring that updates did not disrupt the system's operation on any platform.

In the future, the team sees the potential in virtual environments integrating into their workflow. Future projects using virtual environments could deliver better development efficiency due to the consistent development settings across systems. This would simplify the development cycle, cut down setup time and reduce compatibility issues.

This experience helped to understand how to work together in a mixed system context, preparing the team for future work involving different development platforms.

5.8 Strengths and Weaknesses Summary

The evaluation of TrafficVision against the set objectives has demonstrated strengths in real-time processing, user interface design and traffic data visualization. Weak-

nesses are mainly centered around data dependence on user engagement and environmental conditions affecting data acquisition. Addressing these weaknesses will be crucial for the next phase of development, focusing on system robustness and data diversity.

Overall, TrafficVision has made significant strides toward its envisioned goals, laying a solid foundation for future enhancements in traffic monitoring and analysis.

Chapter 6

Conclusion

This dissertation has explored the development and deployment of TrafficVision, a web application that aims to transform urban mobility through real time traffic monitoring and advanced analytics. The main focus of this large scale project was to improve the network reliability of public transport services, commuter wait times and sustainable urbanization through traffic congestion and emissions reduction.

TrafficVision was developed using OpenCV for video processing, YOLOv8 for object detection, Kafka for data handling and MongoDB for data storage. These technologies were combined to process real-time video data. Database management systems was crucial for analyzing urban traffic patterns and improving public transport reliability and punctuality.

On a closer look, TrafficVision was a success in achieving its initial goals due to its real-time traffic status reports and bus schedule compliance. This critical feature allows commuters and urban planners to make informed decisions that significantly improved commuting efficiency and traffic control. However, TrafficVision currently has a limit on the amount of crowdsourced data it processes. With more data collection the system will become more precise and useful in terms of deep insights and more accurate traffic forecasts.

One of TrafficVision's key functions is providing transportation authorities and government entities with real-time information about bus punctuality and traffic density. This info is clear evidence that could be used to lobby for required reforms in the public transport system. TrafficVision identifies bus service problem areas and peak times with high delays and supports targeted interventions to reduce traffic congestion and improve bus service efficiency.

6.0.1 Future improvements

Future improvements to TrafficVision are numerous. Future developments could mainly focus on extending data collection techniques to cover more passive data

streams and on enhancing system robustness to various environmental conditions. High-end image processing technologies could be combined to work well in different lighting and weather conditions to achieve uniform performance regardless of external conditions.

Furthermore, collaboration with local governments and transportation authorities could enhance data quality and comprehensiveness and extend TrafficVision's reach across geographic and operational boundaries. Such collaborations could also enable TrafficVision to be incorporated into official traffic management and urban planning initiatives as a core technology for smart city initiatives.

TrafficVision provided a solid foundation for the future of traffic management systems and demonstrated the potential of technology in resolving challenging urban mobility issues. The user-contributed data will continue to refine TrafficVision as a useful tool for urban planning and public transportation optimization. With ongoing technological developments and enhanced user engagement, TrafficVision will be considered a key element of future smart cities and will lead to transformation of public transportation.

6.0.2 In conclusion

TrafficVision met its initial objectives of proving it was capable of boosting the public transportation reliability via real time traffic monitoring. However with increased time for deployment and a bigger dataset the application would have further refined during its first functional stage. These enhancements would refine the system's accuracy and user interface, setting the foundation for its subsequent development stages.

Looking forward, machine learning could be used to analyze the massive traffic data collected. This advancement will enable TrafficVision to not just react to actual traffic situations but also predict the real traffic patterns to better inform urbanized transportation methods. Such predictive features, coupled with environmental traffic monitoring (like emissions monitoring),would expand TrafficVision into a more of a tool for city planning and sustainability.

Appendix A

Project Resources

- Video Demo: Project Demo Link
- GitHub Repository: FYP on GitHub
- Kanban Board: Project Kanban Board
- Gantt Chart: Project Timeline

Bibliography

[1] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[2] Lucidchart. Data flow diagram. `https://www.lucidchart.com/pages/data-flow-diagram`, 2023.

[3] Kent Beck et al. Manifesto for agile software development. `https://agilemanifesto.org/`, 2001.

[4] GitHub, Inc. Github: Where the world builds software. `https://github.com/`, 2024.

[5] Postman, Inc. Postman api development. `https://www.postman.com/`, 2021.

[6] MongoDB Inc. Mongodb documentation. `https://docs.mongodb.com/`, 2021.

[7] Guido Van Rossum and Fred L. Jr. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[8] Gary Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[9] Facebook. React: A javascript library for building user interfaces. `https://reactjs.org/`, 2021.

[10] Node.js Foundation. Node.js official documentation. `https://nodejs.org/en/`, 2024.

[11] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: A distributed messaging system for log processing. In *NetDB*, 2011.

[12] Confluent, Inc. Confluent cloud: Stream data across public clouds. `https://www.confluent.io/confluent-cloud/`, 2024.

[13] bcrypt Contributors. bcrypt: A library to help you hash passwords. https://www.npmjs.com/package/bcrypt, 2024.

[14] HPC Wiki Contributors. Sh file. https://hpc-wiki.info/hpc/Sh-file#:~:text=A%20shell%20script%20or%20sh,the%20specified%20commands%20in%20order., Year of publication (if available).

[15] Docker Documentation. Dockerfile reference. https://docs.docker.com/reference/dockerfile/#:~:text=A%20Dockerfile%20is%20a%20text,line%20to%20assemble%20an%20image., n.d.

[16] Miro. What is a uml diagram. https://miro.com/diagramming/what-is-a-uml-diagram/, 2023.

[17] Bitsrc Blog. A gentle introduction to .env files. https://blog.bitsrc.io/a-gentle-introduction-to-env-files-9ad424cc5ff4, n.d.

[18] 12factor.net. The twelve-factor app. https://12factor.net/, n.d.

[19] Railway. Railway: Infrastructure for modern applications. https://railway.app/, 2024.