



XXXXXXXX | XXXXXX

**INTELLIGENT DRIVER BEHAVIOUR MONITORING SYSTEM
USING ARTIFICIAL INTELLIGENCE FOR ENHANCED SAFETY**

School of Science and Technology, United States International University

APT4099 Final Year Project

Dr. Paul Okanda

06|12|2024

Project Report

DECLARATION

The content of this document is the original work based on my own research and to the best of my knowledge it has not been presented elsewhere for academic purposes.

XXXXXXXXXX XXXXXXXXX

Signed..... Date

This project is submitted as part of the University requirement for the award of the degree of Bachelor of Science in Applied Computer Technology from United States International University Africa

Project supervisor

PROF. PAUL OKANDA.

Signed Date

ACKNOWLEDGEMENT

I would like to extend my sincere appreciation to Prof. Paul M. Okanda for his unwavering support, insightful guidance, and expert advice throughout this project. His mentorship played a crucial role in shaping the direction and successful completion of this work.

I am equally grateful to my colleagues, friends, and family, whose encouragement and moral support kept me motivated along this journey. A special note of thanks to my colleagues for their invaluable technical assistance and collaborative discussions.

Lastly, I would like to recognize USIU-A for providing the essential resources and a conducive environment that enabled me to carry out this project.

ABSTRACT

Road accidents caused by driver fatigue, distraction, and intoxication have become a significant concern. This project aimed to develop a Driver Monitoring and Behavioral Analysis System (DMBAS) to improve road safety by monitoring drivers in real time. Using artificial intelligence (AI), the system analyzes facial expressions, eye movements, and physical signals to detect risky behaviors like drowsiness, stress, or distraction. When such behaviors are identified, the system provides immediate alerts to the driver and, in critical cases, notifies emergency contacts to prevent potential accidents.

Key features of the DMBAS include facial recognition and emotion detection, which use AI models like convolutional neural networks (CNNs) to identify signs of fatigue or stress. The system also uses computer vision and machine learning to detect distractions, such as when a driver's attention drifts away from the road. Additionally, psychological analysis tools help monitor driver behavior patterns, providing feedback that can improve driving habits and overall safety.

The main goal of this project was to enhance driver safety through real-time alerts and support systems. By focusing on early detection of impairment, the DMBAS aims to prevent accidents and provide a safer driving environment. The system was also designed to be user-friendly, with features like voice commands and intuitive interfaces to minimize distraction while driving. Moreover, the data collected can help improve future driver assistance systems and safety technologies.

This project bridges the gap between existing driver assistance tools and the growing need for intelligent monitoring solutions, ultimately reducing road accidents and improving driver awareness.

TABLE OF CONTENTS

DECLARATION	ii
ACKNOWLEDGEMENT	ii
ABSTRACT	iv
CHAPTER 1: INTRODUCTION	1
1.1 History of Driver Monitoring Systems	1
1.2 Problem Statement	2
CHAPTER 2: LITERATURE REVIEW	3
2.1 Overview	3
2.2 Global Scale	3
2.2.1 Magna	4
2.2.2 Smart Eye	5
2.3 Local Scale	6
2.3.1 TrailMyCar	6
2.3.2 Karooooo	7
2.4 Strengths and Weaknesses	8
CHAPTER 3: AIMS AND OBJECTIVES	11
3.1 General Aims and Objectives	11
3.2 Specific Aims and Objectives	12
CHAPTER 4: PROPOSED PROJECT	13
4.1 Project Phases	13
4.2 Program of Work	16
4.2.1 Phase One: Research and Requirements Gathering	16
4.2.2 Phase Two: System Design and Implementation	17
4.2.3 Phase Three: Testing, Documentation and Presentation	18
4.3 Gantt Chart	20
4.4 Requirements	22
4.4.1 Software Requirements	22
4.4.2 Hardware Requirements	22
4.4.3 Server	22
4.4.4 Budget	22
CHAPTER 5: SYSTEM ANALYSIS AND DESIGN	23
5.1 Overview	23
5.2 System Models	23
5.2.1 System Architecture Diagram	24
5.2.2 Use Case Diagram	25
5.2.3 Flowchart Diagram	26

5.2.4 UML Class Diagram	27
5.2.5 Entity Relationship Diagram.....	27
5.2.6 Sequence Diagram	29
5.2.7 Data Flow Diagram.....	30
5.2.8 Network Diagram	33
5.2.9 Wireframes	37
5.3 Functional Requirements	43
5.4 Non-Functional Requirements	45
CHAPTER 6: IMPLEMENTATION	47
6.1 Overview	47
6.2 Coding	47
6.2.1 Backend Coding Implementation.....	47
6.2.2 Frontend Coding Implementation.....	53
6.3 Integration	55
6.4 Application Deployment.....	56
6.5 Summary	58
CHAPTER 7: TESTING & EVALUATION	59
7.1. Introduction	59
7. 2. System Testing	59
7.2.1 Authentication Testing.....	59
7.2.2 Scalability Testing.....	61
7.2.3 Performance Testing	62
7.3 User Acceptance Testing	63
7.3.1 Real-time Driver Monitoring System.....	64
7.3.2 Alert System through SMS	74
7.3.3 System escalation of alerts to secondary emergency contact's	78
7.4 Key Findings	78
7.5 Impact Analysis	79
7.6 Summary	80
CHAPTER 8: CONCLUSION.....	82
8.1 Achievements	83
8.2 Challenges encountered	84
8.3 Future Works	84
REFERENCES.....	84
APPENDICIES.....	86
I. LOGBOOK	86
II. PLAGARISM REPORT	93

LIST OF TABLES AND FIGURES

Table 1: Strengths and Weaknesses between different DMS	8
Table 2: Authentication Test Case	55
Table 3: Scalability Test Case	57
Table 4: Performance Test Case	58
Table 5: Sober Users State Test Case	62
Table 6: Drunk User state Test Case	64
Table 8: No User state Test Case	66
Table 9: No matching state detected	70
Table 10: Alert system through SMS Test Case	71
Table 11: Escalation of alerts to emergency contacts	74
Figure 1: Magna Driver Monitoring System	4
Figure 2: Smart Eye Driver Monitoring System	6
Figure 3: Cartrack Driver Monitoring System	7
Figure 4: Gantt Chart	20
Figure 5: System Architecture Diagram	23
Figure 6: Use Case Diagram	24
Figure 7: Flowchart Diagram	25
Figure 8: UML Class Diagram	26
Figure 9: Entity Relationship Diagram	27
Figure 10: Sequence Diagram	28
Figure 11: Data Flow Diagram Level 0	29
Figure 12: Data Flow Diagram Level 1	29
Figure 13: Data Flow Diagram Level 2	30
Figure 14: Network Diagram	31
Figure 15: Part 2 Network Diagram	32
Figure 16: Startup Page	33
Figure 17: Login Page	34
Figure 18: Sign up Page	35
Figure 19: Get Started Page	36
Figure 20: Home Page	37
Figure 21: Facial Recognition Page	38
Figure 22: Analysis Page	39
Figure 23: Initializing the Camera and Model	44
Figure 24: Camera Initialization	45
Figure 25: Configuring and Starting the Camera	45
Figure 26: Permissions to initialize the camera	46
Figure 27: Initializing the camera	46
Figure 28: Running the AI Model	47
Figure 29: Loading the Model	48
Figure 30: Loading the model	49
Figure 31: UI Development	50
Figure 32: Driver Management System UI	51
Figure 33: Proof of Deployment	53
Figure 34: Test Code Result (login)	56
Figure 35: Test Code Result (Signup)	56
Figure 36: Scalability test code result (Firebase)	58
Figure 37: Performance test code result (Firebase query)	59
Figure 38: Sober person	60
Figure 39: Sober state Test Code	61
Figure 40: Test Code Result for detecting a sober state	62
Figure 41: Drunk person	63
Figure 42: Drunk person Test Code	63
Figure 43: Test Case Result to detect a drunk person	65

Figure 44: No state detected	65
Figure 45: No State Test Code	66
Figure 46: TestCase to not detect a state with confidence	67
Figure 47: No matching state detected	68
Figure 48: No matching state Test Code	69
Figure 49: No matching state is detected	70
Figure 50: Alert system Test code.....	72
Figure 51: Test Code Result I	73
Figure 52:Test Code Result II	73

PROJECT TITLE: Intelligent Driver Behaviour Monitoring System Using Artificial Intelligence for Enhanced Safety

CHAPTER 1: INTRODUCTION

In today's world, road safety remains a significant concern, with numerous accidents caused by driver fatigue, distraction, and intoxication. To combat these issues, there is a growing demand for intelligent driver monitoring systems that can assess driver behaviour and provide real-time feedback to prevent accidents. The Driver Monitoring and Behavioural Analysis System aims to enhance road safety by utilizing advanced technologies, including artificial intelligence (AI) and machine learning, to monitor drivers' physiological signals, facial expressions, and eye movements (Dohun Kim, 2023).

This system focused on detecting signs of stress, distraction, and fatigue through AI-based techniques such as deep learning for facial recognition and computer vision for real-time monitoring. By analyzing the driver's condition and providing actionable feedback, the system was designed to prevent risky behaviours and alert both the driver and relevant emergency responders in case of danger.

While existing driver monitoring systems have made strides in addressing these concerns, they often face challenges in providing comprehensive real-time analysis and integration of diverse data sources. Many systems lack the ability to effectively assess the driver's psychological state and provide timely emergency notifications. This project was designed to overcome these limitations by integrating state-of-the-art AI models, real-time feedback mechanisms, and emergency alert systems to improve driver awareness and overall road safety (Dohun Kim, 2023).

1.1 History of Driver Monitoring Systems

The development of driver monitoring systems (DMS) began with simple technologies aimed at addressing the dangers of drowsy driving. Early systems were designed to monitor basic driving patterns, such as steering behaviour, or employed infrared cameras to track whether the driver's eyes were closing. These systems were reactive and focused on alerting drivers in the event of drowsiness. Although they played a significant role in reducing accidents, their scope was limited to specific behavioural signs without providing a comprehensive analysis of the driver's condition.

With advancements in technology, DMS evolved to become more sophisticated, incorporating AI and machine learning techniques. Modern systems are now capable of analyzing a wide range of behavioural and physiological signals, such as facial expressions, eye movements, and heart rate, to detect signs of fatigue, distraction, and intoxication in real time. Despite the progress, earlier systems often struggled with high rates of false positives and limited data integration. Today's systems are much more advanced, using multiple data points from sensors and cameras to provide a more holistic view of the driver's state and improve overall road safety (Dietrich Manstetten, 2021).

1.2 Problem Statement

Despite significant advancements in DMS, road accidents caused by driver fatigue, distraction, and intoxication remain a major global concern. Current systems often face limitations in accurately detecting and responding to real-time behavioural and physiological changes in drivers, leading to high false-positive rates and incomplete assessments of driver conditions. Additionally, many DMS lack comprehensive integration of psychological analysis, failing to assess stress levels and emotional states, which are critical factors in driving behaviour. As a result, there is a need for an enhanced system that combines AI, machine learning, and realtime data integration to provide a more holistic and accurate assessment of driver behaviour, ensuring safer roads and reducing accident risks.

CHAPTER 2: LITERATURE REVIEW

This literature review explores the advancements, challenges, and opportunities in Driver Monitoring Systems, focusing on the integration of AI, machine learning, and sensor technologies. It examines global and local implementations of Driver Monitoring Systems, providing insights into their effectiveness in enhancing road safety.

2.1 Overview

Driver monitoring systems (DMS) have garnered significant attention in recent years as a means of improving road safety by detecting risky driving behaviours such as fatigue, distraction, and intoxication. Numerous studies have explored the integration of advanced technologies such as artificial intelligence (AI), computer vision, and machine learning to enhance the detection accuracy of these systems. Researchers have focused on using facial recognition, eye tracking, and physiological signal analysis to assess a driver's condition in real time, with the goal of reducing accidents and improving overall driving safety. However, despite the progress in this field, the effectiveness of DMS in addressing all aspects of driver behaviour, including psychological and emotional states, remains a challenge.

The existing body of literature highlights various technological advancements and implementations in DMS, yet gaps persist, particularly in the comprehensive assessment of a driver's mental state and the seamless integration of multiple data streams. Several systems have been developed that focus primarily on detecting physical signs of fatigue or distraction but often overlook emotional factors such as stress, anxiety, and frustration, which can significantly impact driving performance. Moreover, many studies have identified issues with false-positive detections and the complexity of integrating diverse sensor data for real-time monitoring. This literature review aims to assess the strengths and weaknesses of existing systems while identifying opportunities for improvement in future developments.

2.2 Global Scale

Driver Monitoring Systems are gaining widespread adoption globally as advancements in automotive safety technologies become more crucial. These systems utilize AI, computer vision, and machine learning to monitor driver behaviour, prevent accidents, and enhance road safety. Globally, companies like Magna, Smart Eye, and Cartrack have developed sophisticated DMS technologies tailored to detect signs of fatigue, distraction, and intoxication, leading to more secure driving environments and significant reductions in road accidents.

2.2.1 Magna

Magna's DMS is a state-of-the-art technology integrated directly into vehicle mirrors, offering a seamless solution for improving driver safety. The system is equipped with a camera and sensor array designed to capture the driver's eye movements, facial expressions, and head positioning. These data points are processed in real time by artificial intelligence algorithms to determine the driver's state of awareness. If signs of distraction, fatigue, or drowsiness are detected, the system provides immediate alerts to the driver, promoting safer driving practices (Magna, n.d.).

In addition to providing alerts, Magna's DMS can be linked to advanced driver assistance systems (ADAS) to trigger automatic responses, such as reducing the vehicle's speed or pulling the vehicle over to the side of the road in critical situations. By integrating this technology into vehicle mirrors, Magna offers a streamlined solution that doesn't require additional cabin space. The company's approach has been lauded for its unobtrusive design and effectiveness in mitigating risky driving behaviours. Furthermore, Magna's DMS is adaptable to different types of vehicles, ranging from passenger cars to commercial trucks, making it a versatile solution for road safety worldwide.

Magna's DMS also supports semi-autonomous driving technologies by ensuring that the driver remains engaged even when the vehicle is in self-driving mode. The system continually monitors the driver's attentiveness, ensuring that they are ready to take control of the vehicle when necessary. In doing so, Magna's system plays a critical role in bridging the gap between fully manual and autonomous driving, enhancing safety during the transition to automated vehicles (Magna, n.d.).



Figure 1: Magna Driver Monitoring System
Source: (Magna, n.d.)

Figure 1 shows an in-vehicle DMS identifying a distracted driver. The system uses facial recognition technology to track the driver's attention level and highlights signs of distraction, such as looking away from the road, as a key safety feature to prevent accidents (Magna, n.d.).

2.2.2 Smart Eye

Smart Eye offers advanced Driver Monitoring Solutions that utilize artificial intelligence, deep learning, and computer vision technologies to detect driver behaviour in real-time. The core of Smart Eye's system is its eye-tracking technology, which monitors eye movement patterns to assess the driver's level of attention and fatigue. By analyzing these patterns, the system can determine whether a driver is becoming distracted or drowsy, triggering an alert to prevent potential accidents (Smart Eye , n.d.).

One of the key features of Smart Eye's DMS is its ability to operate in a wide range of lighting conditions, ensuring accuracy in monitoring both day and night. Additionally, it is designed to work even if the driver is wearing sunglasses or if the lighting conditions are less than ideal, making it highly reliable under various driving conditions. The system can also differentiate between deliberate distractions, such as looking at a GPS system, and dangerous distractions, such as texting while driving. This level of precision enables Smart Eye's solution to provide more tailored and context-aware alerts to the driver, improving overall road safety (Smart Eye , n.d.).

Smart Eye's DMS is particularly beneficial in semi-autonomous and autonomous vehicles, where monitoring the driver's state is crucial to ensuring safe vehicle operation. The system ensures that drivers remain attentive even during periods of reduced manual input, such as highway driving with cruise control. This capability makes Smart Eye a leader in providing AI-driven solutions that not only enhance driver safety but also complement the growing trend toward vehicle automation (Smart Eye , n.d.).

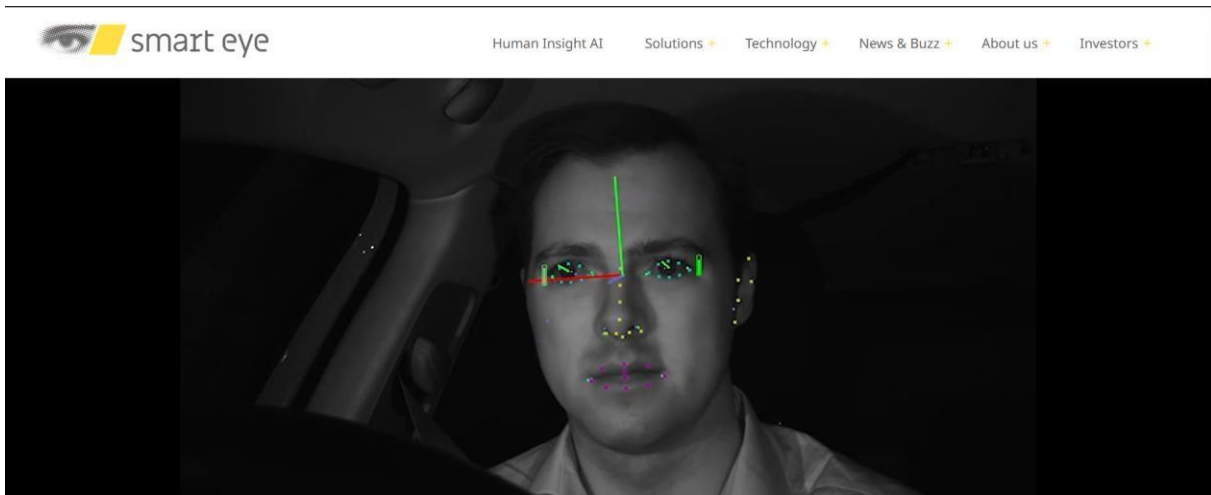


Figure 2: Smart Eye Driver Monitoring System

Source: (Smart Eye , n.d.)

Figure 2 illustrates the advanced facial recognition technology used by Smart Eye's DMS. By mapping key facial landmarks, the system can assess driver attention, emotional state, and signs of drowsiness, contributing to real-time safety measures in modern vehicles (Smart Eye , n.d.).

2.3 Local Scale

At the local scale, driver monitoring systems are gaining traction in regions such as Kenya, where road safety concerns are prompting the adoption of these technologies. Local companies and providers are beginning to implement DMS solutions to address issues of driver distraction, fatigue, and intoxication, particularly in commercial and public transport sectors. These systems not only aim to improve individual driver safety but also enhance the overall safety of passengers and other road users by providing real-time monitoring and alerting capabilities, tailored to the unique needs of the local context.

2.3.1 TrailMyCar

TrailMyCar, a locally-based company in Kenya, provides a comprehensive DMS focused on ensuring driver safety and vehicle security. The system uses telematics, a combination of GPS and diagnostic tools, to monitor driver behaviour in real-time. This technology tracks metrics such as speed, braking patterns, and vehicle handling, alerting fleet managers or drivers if risky driving behaviours are detected. Additionally, the system aims to reduce accidents and optimize vehicle usage by providing actionable insights based on driving data. TrailMyCar DMS is designed to meet the local market's need for cost-effective, scalable solutions for businesses

managing large fleets, and its integration into fleet management has significantly improved driver accountability and safety across Kenya (TrailMyCar, n.d.).

2.3.2 Karooooo

Karooooo, another leading DMS provider in Kenya, focuses on promoting driver safety through advanced monitoring systems that assess driver alertness and behaviour. The system uses realtime data from in-vehicle cameras and sensors to detect signs of driver fatigue, distraction, and poor driving habits. It alerts the driver or the fleet manager when dangerous behaviours such as drowsiness, speeding, or distraction occur. Karooooo's system goes beyond just monitoring to actively prevent accidents by generating early warnings for drivers and providing detailed reports for fleet managers to improve training and ensure adherence to safety protocols. These systems are particularly beneficial in Kenya's commercial transportation sector, where accidents are often caused by overworked or distracted drivers (Karooooo, 2023).

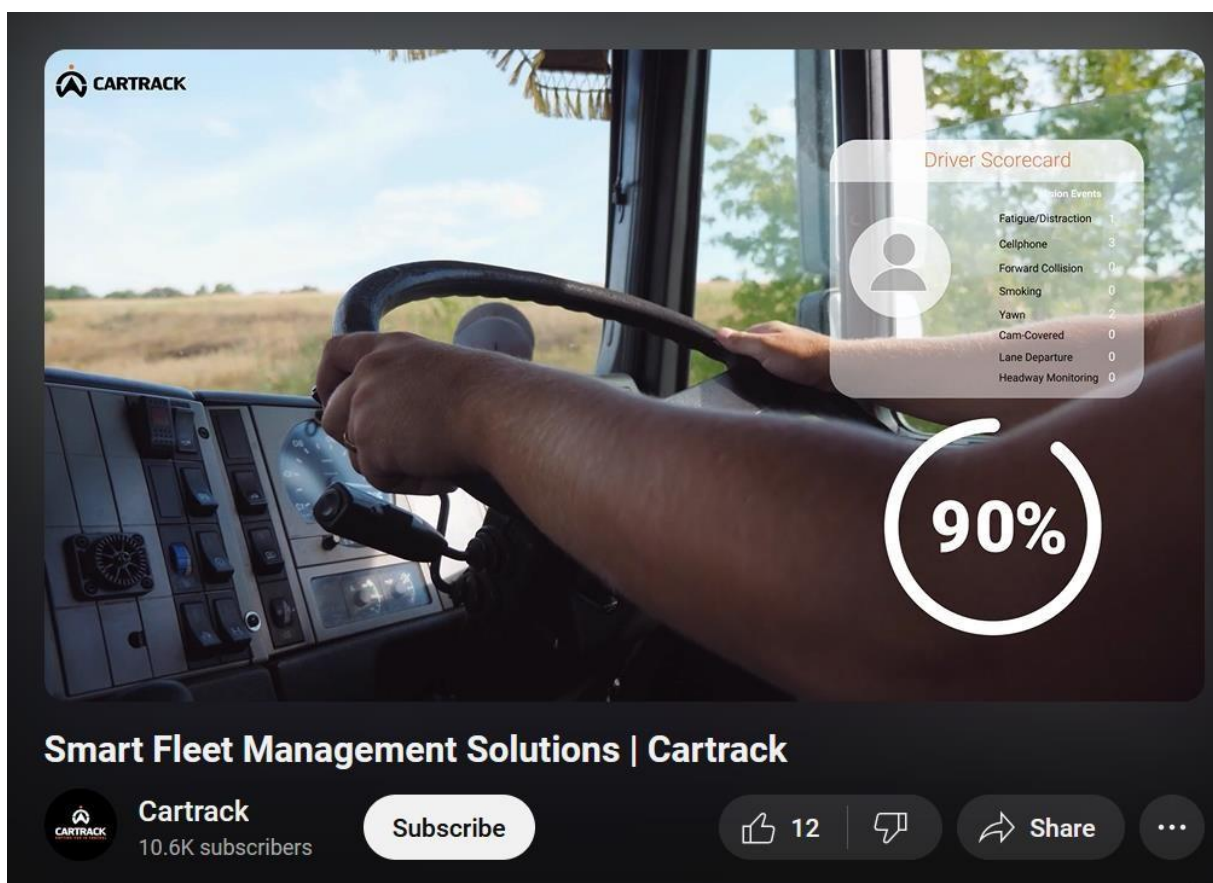


Figure 3: Cartrack Driver Monitoring System

Source: (Karooooo, 2023)

Figure 3 depicts Cartrack's Driver Scorecard system, which is part of their fleet management solution. It evaluates and tracks driver behaviour, including fatigue detection, headway

monitoring, and other safety-related metrics, aimed at improving overall driver performance and road safety (Karo00000, 2023).

2.4 Strengths and Weaknesses

In this section, various Driver Monitoring Systems are evaluated to identify their unique capabilities and areas for improvement. The analysis reveals key technological advancements, alongside the limitations that affect the effectiveness of these systems in both global and local scales.

Table 1: Strengths and Weaknesses between different DMS

DMS	Strengths	Weaknesses
Global Scale (USA)		
Magna Driver Monitoring System	<ul style="list-style-type: none"> - Advanced AI for detecting distracted driving, fatigue and intoxication. - Uses facial recognition for emotion detection. - Real-time driver monitoring. 	<ul style="list-style-type: none"> - High cost of implementation. - Privacy concerns due to extensive driver data collection. - Sensitive to lighting and environmental conditions.
Smart Eye Driver Monitoring System	<ul style="list-style-type: none"> - Highly accurate eyetracking and facial recognition technology. - Effective for detecting drowsiness and distracted driving. - Scalable for different vehicles. 	<ul style="list-style-type: none"> - Complex system integration. - High costs for smaller organizations. - Needs optimal conditions for precise detection.
Local Scale (Kenya)		

TrailMyCar	<ul style="list-style-type: none"> - Real-time monitoring of driver behaviour using telematics. - Cost-effective and scalable for the local market. - Provides actionable insights for fleet management. 	<ul style="list-style-type: none"> - Limited AI-based emotional detection. - Focuses more on fleet management rather than individual drivers. - Basic functionality for small fleets.
Karooooo	<ul style="list-style-type: none"> - Real-time driver behaviour analysis using in-vehicle cameras and sensors. - Alerts for dangerous behaviours like drowsiness and distraction. - Preventive accident measures. 	<ul style="list-style-type: none"> - High dependency on camera and sensor accuracy. Focused more on commercial transportation. - Limited customization for personal vehicle use.

Based on the strengths and weaknesses outlined in the table above, there is an increasing demand for a comprehensive Driver Monitoring System (DMS) that goes beyond conventional solutions, addressing critical gaps in both local and global scales. Current DMS applications, while advanced in certain aspects, often fall short of offering a holistic, cost-effective, and adaptable approach that effectively enhances driver safety and provides real-time monitoring without significant cost or technical barriers.

This project proposed the development of an innovative Driver Monitoring and Behavioural Analysis System that serves as a one-stop solution for individuals, fleet managers, and organizations alike. This system aimed to revolutionize road safety by integrating cutting-edge AI technology, emotional and behavioural monitoring, and real-time alerts to detect signs of driver fatigue, distraction, or intoxication. Additionally, it provided immediate feedback and intervention options to ensure timely preventive measures are taken to avoid accidents.

The DMS features a combination of AI-powered facial recognition, emotion detection, and real-time data tracking to monitor driver behaviour in various conditions. Unlike existing systems, it offers a scalable, cost-effective platform that can be used by both individual drivers and large fleets. The system included personalized monitoring, emergency notifications, and even location-based services to ensure driver safety at all times. By combining data-driven insights and advanced monitoring techniques, this project empowers users to improve driving habits, enhance road safety, and reduce accidents in an efficient and effective manner.

CHAPTER 3: AIMS AND OBJECTIVES

The primary aim of the Driver Monitoring and Behavioural Analysis System (DMS) project was to serve as an innovative and comprehensive platform that enhances road safety by monitoring driver behaviour, detecting signs of fatigue, distraction, and other risky behaviours, and providing real-time alerts and interventions to prevent accidents.

3.1 General Aims and Objectives

The primary aim of the DMS project was to create an innovative platform that significantly enhances road safety by integrating advanced monitoring technologies to track driver behaviour. Based on insights from the literature review, our system will focus on detecting signs of fatigue, distraction, and other risky behaviours that can lead to accidents. By employing sophisticated artificial intelligence (AI) and machine learning algorithms, as well as computer vision techniques, the DMS provides real-time alerts and interventions tailored to individual drivers, fleet managers, and organizations. This comprehensive approach aimed to address the existing gaps in current systems, ensuring a proactive stance in accident prevention and enhancing overall driving performance.

To guide the design and development of this project, a literature survey was conducted to examine the current landscape of driver monitoring technologies and identify key areas for improvement. This survey reviewed a range of studies and existing solutions, focusing on AI and machine learning models, facial recognition for fatigue and distraction detection, and other sensors used for driver behaviour analysis. It highlighted gaps in affordability, scalability, and adaptability of current systems, particularly for individual users and small fleet operators. Findings from the survey underscore the need for a cost-effective, user-friendly DMS that incorporates real-time monitoring, personalized alerts, and emergency response features. Insights gained from this literature review serve as a foundation for designing a system that addresses both the technological and market challenges identified.

To achieve these aims, the design and development phase involved creating a prototype DMS that encompasses a range of critical features. This included real-time tracking of driver conditions, facial recognition technology for fatigue and distraction detection, and GPS-enabled emergency response services. The alert system notifies drivers and fleet managers of risky behaviours, ensuring timely interventions. The development also focused on creating a userfriendly interface that allowed seamless integration into various vehicles, catering to both individual users and commercial fleets.

The effectiveness of the DMS will be rigorously tested and validated through user testing to ensure it meets the needs identified in the literature review and the expectations of the target audience. This phase involved collecting feedback from users to assess the system's performance in real-world scenarios, focusing on its ability to monitor driver behaviour, enhance safety, and prevent accidents. The ultimate goal was to refine the prototype based on user input and to ensure that it not only aligns with local and global market demands but also provides a sustainable and effective solution for improving road safety.

3.2 Specific Aims and Objectives

The specific aims and objectives of this project were to:

- Conduct a literature survey on existing representative Driver Monitoring Systems (DMSes) in order to identify their strengths and weaknesses.
- Design and develop a prototype DMS that tracks drivers' behavior real-time, has an SMS alert system and provides an escalation of alerts that notifies the driver or fleet manager when risky behaviours are detected.
- Perform user testing and validation of the prototype system in order to assess the extent to which it addresses challenges related to driver safety, behaviour monitoring and accident prevention.

CHAPTER 4: PROPOSED PROJECT

The proposed project focused on the development of a comprehensive Driver Monitoring and Behavioural Analysis System (DMBAS), which operates as a vital component within a broader Driver Assistance Program (DAP). The DMBAS is designed to monitor the driver's condition in real time, analyzing data to categorize the driver's state as alert, drowsy, distracted, or intoxicated. Based on this assessment, the system was coded to notify emergency contacts immediately and alert the driver if they are found to be in a compromised state, ensuring timely intervention to prevent potential accidents.

The system functions by utilizing various sensors and AI technologies to continuously monitor the driver's cognitive and physical states. When risky behaviours, such as drowsiness or distraction, are detected, the DMBAS initiates an automatic alert to designated emergency contacts, providing real-time information about the driver's condition and location.

The system also suggests that the driver stop at a safe spot, like a rest area or gas station, so they can take a break or find another way to get around. This proactive approach aimed to enhance road safety by addressing driver impairment before it escalates into hazardous situations, adding an essential layer of preventive safety within the overall driver assistance framework.

4.1 Project Phases

The project was structured in three main phases: research and requirements gathering, system design and implementation, and testing, documentation, and presentation. These phases were critical to ensure the successful development and deployment of the Driver Monitoring System (DMS), each focusing on distinct but interconnected aspects of the project's progression. Below is a detailed outline of these phases.

4.1.1 Phase One: Research and Requirements Gathering

The research and requirements gathering phase laid the foundation for the development of the Driver Monitoring System. The goal was to ensure that the project aligned with both user needs and technological trends in AI-powered driver safety. This stage focused on understanding the current landscape of driver monitoring technologies, identifying the tools and technologies necessary for the system's implementation, and gathering essential requirements from potential users.

Research

To establish the project's objectives and identify the tools, programming languages, and technologies required for development, a thorough investigation was the initial stage. This research involved analyzing existing driver monitoring systems to identify their strengths and weaknesses, as well as understanding the technological trends in AI and driver safety applications.

In this phase, key decisions were made regarding the use of Python for AI and machine learning algorithms, TensorFlow for deep learning models, and AWS for backend services and data storage. The research provided a clear understanding of the technical and functional requirements for building the platform.

Requirements Gathering

In this phase, the project gathered essential requirements by consulting potential users such as fleet managers, drivers, and safety experts to understand their needs. The project also detailed technical specifications such as real-time monitoring, data storage, driver behaviour analytics, and alerting systems. This step ensured that the DMS would be user-friendly, highly secure, and capable of delivering personalized, real-time driver behaviour assessments.

Proposal

The project proposal outlined the most effective solution for building the DMS. This proposal included a description of the system's key features, such as driver fatigue and distraction detection, emergency response alerts, and AI-powered behaviour analysis. Additionally, it highlighted the system architecture, database schema, user interface, and security protocols to ensure data privacy and real-time operation. The proposal functioned as the project development process' road map.

4.1.2 Phase Two: System Design and Implementation

With a clear understanding of the project requirements, the system design phase began. This phase was pivotal in translating the gathered research and user inputs into a tangible system architecture. The goal was to structure the system in a way that would facilitate real-time data processing, secure storage, and user-friendly interactions. Key components such as the user interface, backend architecture, and database schemas were meticulously crafted, laying the groundwork for the development of a robust and scalable system.

System Design

The system design phase focused on making the main components of the Driver Monitoring System. This included designing the user interface, defining the system's backend architecture, and detailing the interactions between the different components such as the DMS web dashboard and the APIs. Wireframes and mock-ups were created to visualize how the system would function, and database schemas were structured to store user data, driver behaviour metrics, and system alerts securely.

Software Implementation

The actual coding and component integration into a working system were the tasks of this phase. Python was used to develop machine learning models for detecting driver fatigue and distractions using video input from cameras. AWS services were employed for data storage and real-time processing. The AI-driven dashboard was developed to give real-time feedback and notifications, while the backend was designed to manage secure authentication and storage of driver data.

4.1.3 Phase Three: Testing, Documentation and Presentation

After the software implementation phase, the focus shifted to rigorous testing to ensure the system met all specified requirements and functions correctly under real-world conditions. This phase also involved compiling comprehensive documentation to record the development process and outcomes, as well as preparing for project presentations. The testing phase was crucial for identifying potential issues, fine-tuning the system's functionality, and ensuring its readiness for deployment.

Software Testing and Evaluation

A rigorous testing phase ensured that the system met all the specified requirements. Unit tests, integration tests, and user acceptance testing were conducted to validate the functionality of the system. Automated testing tools were used to check for bugs, and user feedback was gathered to improve the user experience. Through continuous performance testing it ensured the platform could efficiently handle high loads of data and real-time monitoring.

First Project Presentation

The first project presentation was delivered to gather feedback from the supervisor. This presentation covered the system's core features, design, and initial implementation. Based on

the feedback received, refinements were made to the system, including improvements to the user interface and system functionalities to better meet user expectations.

Project Write-Up

The project write-up documented the entire development process, from research and requirements gathering to implementation and testing. This comprehensive document detailed the methodologies, challenges, and solutions encountered throughout the project, ensuring a thorough understanding of the project's evolution.

Final Project Presentation and Submission of Report

The final project presentation highlighted the completed DMS, showcasing its success in realtime driver behaviour monitoring, AI-driven alerts, and overall user experience. The presentation showed how the system was improved by incorporating input from the first presentation. A final report that summarized the project's accomplishments, assessed its effect on road safety, and offered suggestions for further advancements was sent in.

4.2 Program of Work

The program of work describes the methodical strategy used to carry out the Driver Monitoring System project successfully. A methodical progression from research and requirements collection to final testing and deployment was ensured by the meticulous planning and execution of each phase. Timelines for every stage were included in the program, along with important benchmarks and deliverables to monitor development and guarantee adherence to the project's objectives. Each phase's work plan was broken down below.

4.2.1 Phase One: Research and Requirements Gathering

Objective: Define the project requirements and gather detailed specifications from stakeholders.

i. Research

Tasks:

- Identify programming languages, tools, and technologies for development.
- Conduct research to understand technical and functional requirements.
- Review existing driver monitoring systems.

Deliverables:

Technical and Functional Requirements

Timeline: 1 week **ii.**

Proposal

Tasks:

- Develop the project proposal outlining the best solution.
- Define key features and functionalities.
- Present the proposal to the lecturer for approval.

Deliverables:

Project Proposal Document

Timeline: 2 weeks

4.2.2 Phase Two: System Design and Implementation

Objective: Design the system, implement the components, and ensure the platform meets all requirements.

i. System Design

Tasks:

- Design the user interface and experience.
- Develop the database schema and architecture.
- Define interactions between system components.

Deliverables:

- Design diagrams
- Wireframes and Mock-ups
- Database Schema

Timeline: 1 weeks

ii. Software Implementation Tasks:

- Set up the development environment.
- Implement key features like driver monitoring, alerts, and AI analytics.

Deliverables:

- Source Code
- Live web application

Timeline: 3 weeks

4.2.3 Phase Three: Testing, Documentation and Presentation

Objective: Test the project, present it, document the process, and submit the final report.

i. Software Testing and Evaluation

Tasks:

- Perform unit and integration testing.
- Fix bugs and issues identified during testing.

Deliverables:

- Test Reports
- Bug Reports

Timeline: 2 weeks

ii. First Project Presentation

Tasks:

- Present the project to the lecturer.
- Gather feedback for further improvements.

Deliverables:

- Presentation Slides
- Lecturer Feedback

Timeline: 1 week

iii. Project Write-Up

Tasks:

- Document the entire project process.
- Compile a comprehensive final report.

Deliverables:

Final Project Report

Timeline: 1 week

iv. Final Project Presentation and Submission of Report

Tasks:

- Present the final version of the project.
- Submit the final project report.

Deliverables:

Final Project Report

Timeline: 1 week

4.3 Gantt Chart

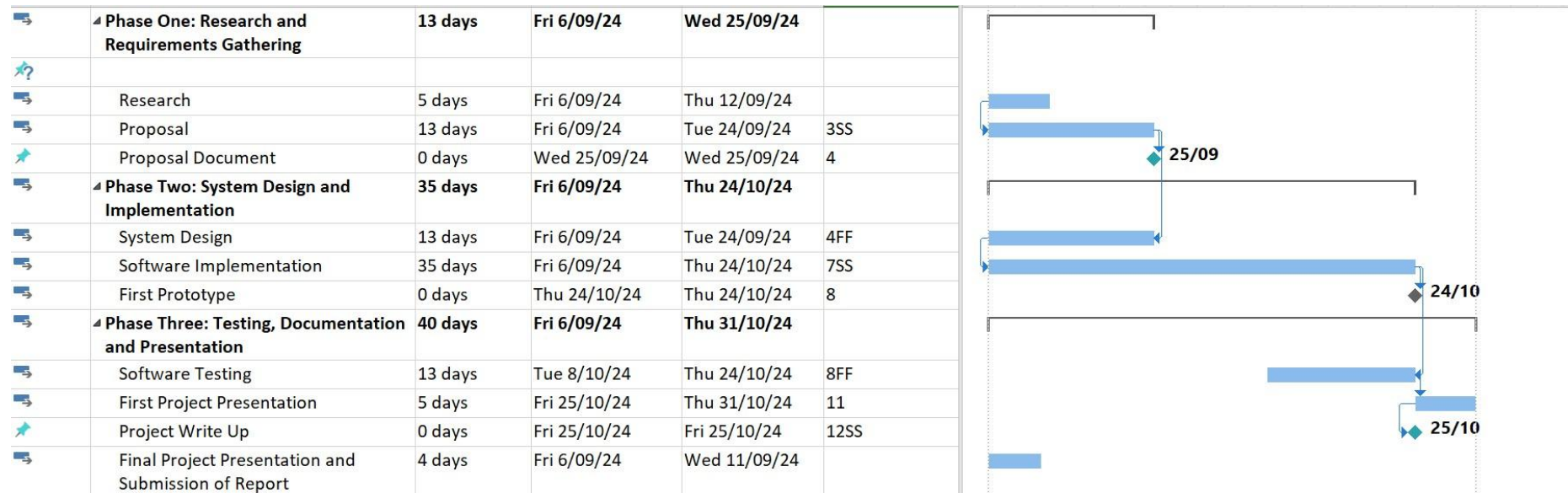


Figure 4: Gantt Chart

Figure 4 above illustrates a Gantt chart for the Driver Monitoring and Behavioural Analysis System project. The project's timeline, which runs from September 6, 2024, to December 6, 2024, is graphically represented by the Gantt chart, which also lists the main tasks and phases. In order to ensure a systematic and orderly movement through the project phases, the chart shows that each task has been allocated a specific duration, start date, and dependencies. In order to successfully complete the DMS project by the end date, each phase was distinguished by significant milestones, such as the creation of the facility locator, emergency notification system, and driver condition monitoring module. ensuring a structured and sequential progression through the project phases.

4.4 Requirements

Below is the software, hardware, server and budget requirements for the project:

4.4.1 Software Requirements

Code Editor:

- Visual Studio Code
- Android Studio (for mobile platform support)

Backend & Authentication:

- Firebase

4.4.2 Hardware Requirements

- Cameras: High-resolution cameras for facial recognition and behaviour monitoring.
- Sensors: Physiological sensors for monitoring heart rate and blink rate; additional sensors for detecting alcohol or drug levels if applicable.
- Personal Computer.
- Mobile Device.

4.4.3 Server

Database and Authentication Server:

- Firebase

4.4.4 Budget

Development Costs: No additional costs.

CHAPTER 5: SYSTEM ANALYSIS AND DESIGN

5.1 Overview

This chapter presents the system analysis and design for the proposed Driver Monitoring and Behavioural Analysis System. It includes various design models that illustrate the system's functionality, architecture, and data flow. Key diagrams such as Use Case, UML Class Diagram, Data Flow Diagrams (DFD Level 0, 1, and 2), Entity Relationship Diagram (ERD), System Architecture, UML Sequence Diagram, and Network Diagram are developed to provide a comprehensive understanding of how the system operates and interacts with its components. These designs serve as a blueprint for the system's development, ensuring a structured and efficient implementation.

5.2 System Models

System models are essential tools in the design and development of complex systems, as they provide a structured framework for understanding and visualizing the various components and their interactions. In the context of the Driver Monitoring System (DMS), system models serve as blueprints that define the system architecture, data flow, and functionality. These models help in illustrating the system's overall structure, user interface, and key processes, ensuring that the development team and stakeholders have a clear understanding of how the system operates. The following section will explore the various system models used in the DMS project, including the architectural model, data flow diagrams, and interaction models, which collectively form the foundation for both development and testing.

5.2.1 System Architecture Diagram

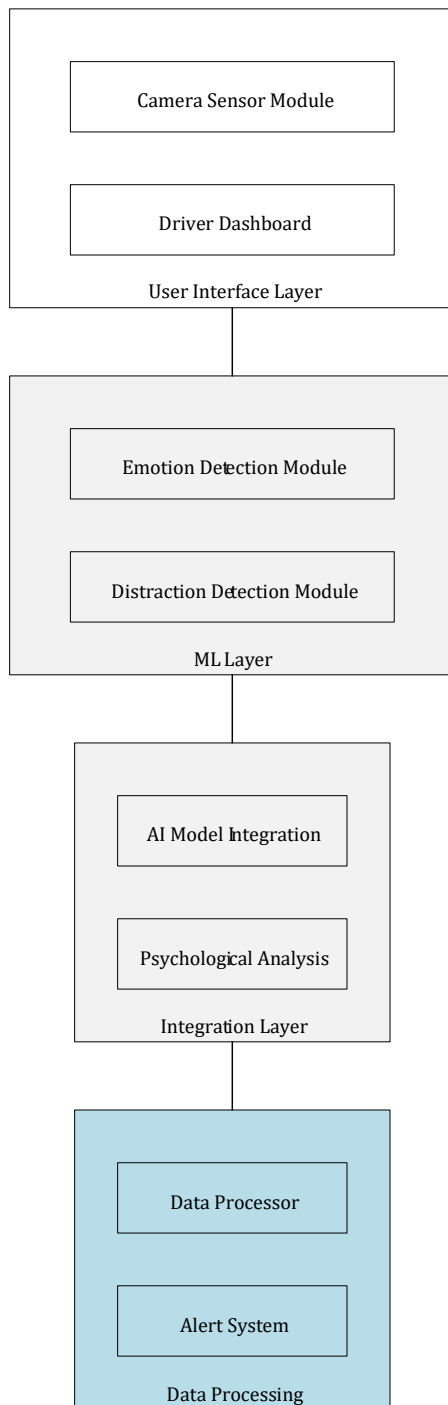


Figure 5: System Architecture Diagram

Figure 5 details the components of a Driver Assistance System broken down by functionality. The User Interface Layer includes modules like Camera Sensor Module and Driver Dashboard, responsible for capturing data and displaying alerts. The ML Layer has modules for Emotion

Detection and Distraction Detection, analyzing driver behaviour. The Integration Layer includes AI Model Integration and Psychological Analysis to synthesize and interpret the data further. Finally, the Data Processing Layer has a Data Processor and Alert System, which handle the processing and generation of alerts based on the analyzed data. This structure illustrates a multi-layered approach to driver monitoring, where each layer contributes to effective real-time analysis and feedback.

5.2.2 Use Case Diagram

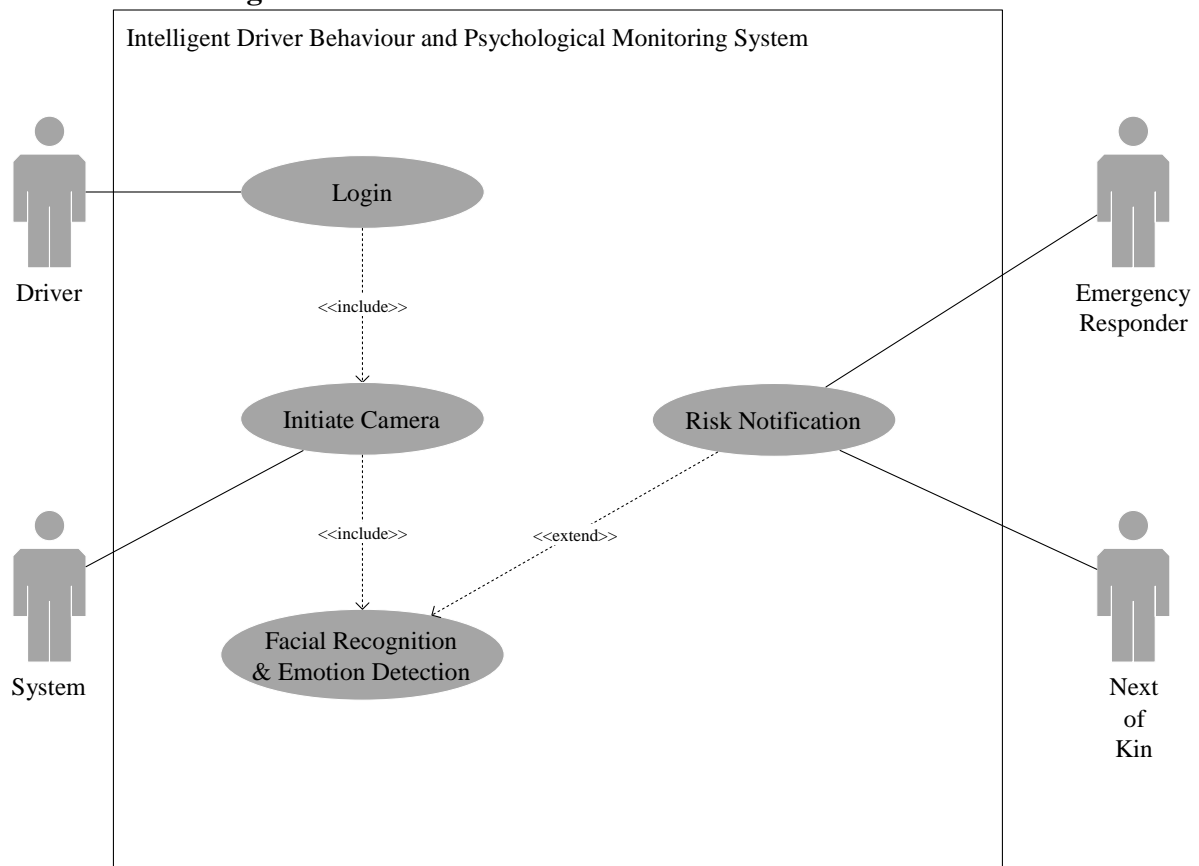


Figure 6: Use Case Diagram

Figure 6 outlines the interactions between different actors and the system. It shows how a driver interacts with the system by logging in and initiating a camera. The system then performs facial recognition and emotion detection. If any risks are detected (e.g., fatigue, distraction), the system sends notifications to emergency responders and the driver's next of kin. The relationships between the use cases (login, risk notification) are defined using "include" and "extend" associations.

5.2.3 Flowchart Diagram

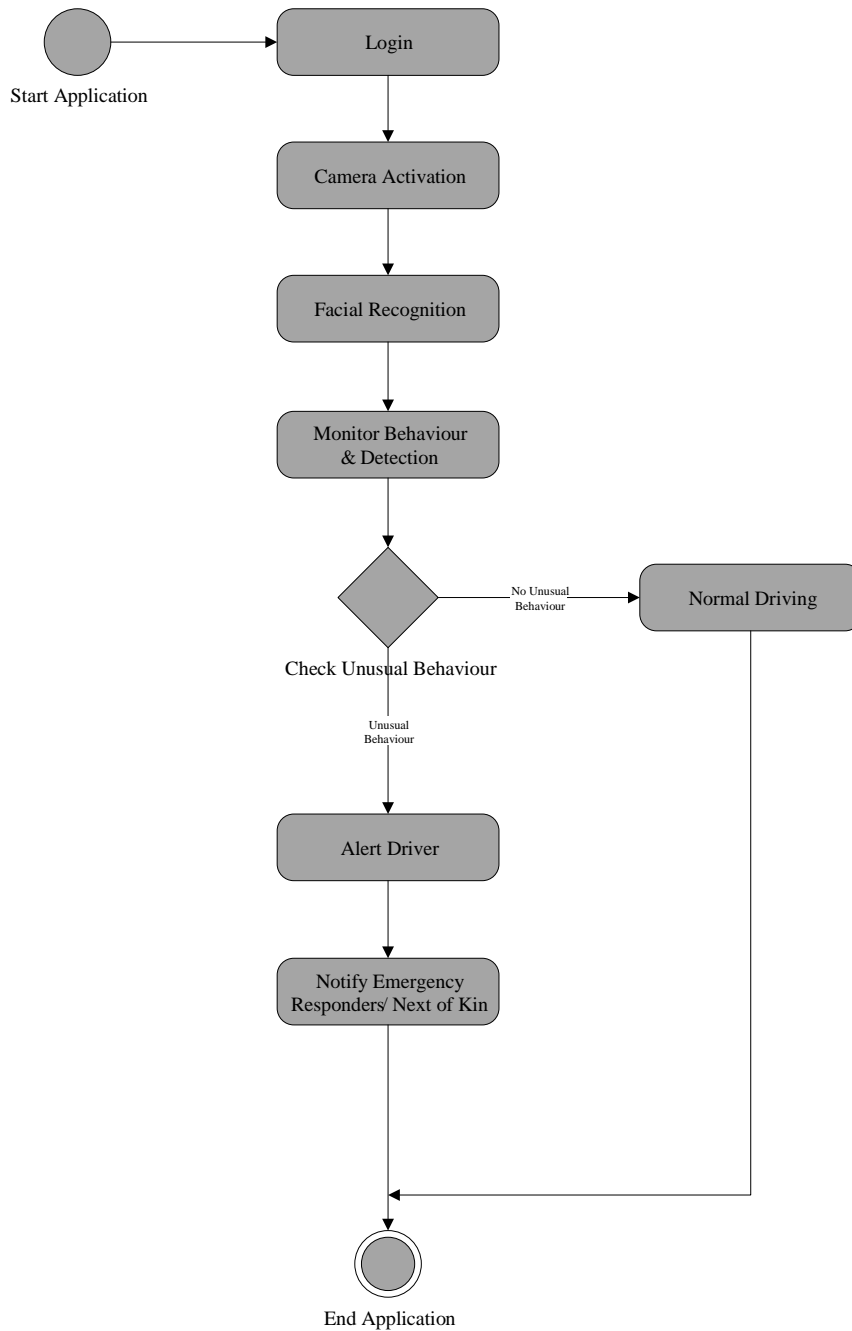


Figure 7: Flowchart Diagram

Figure 7 shows the workflow of the intelligent driver monitoring system. The process starts with the driver logging in and the camera activating. After the system performs facial recognition and monitors driver behaviour, it checks for unusual behaviour. If no unusual behaviour is detected, driving continues as normal. If unusual behaviour is detected, the system

alerts the driver and notifies emergency responders or next of kin. The application ends after this notification process.

5.2.4 UML Class Diagram

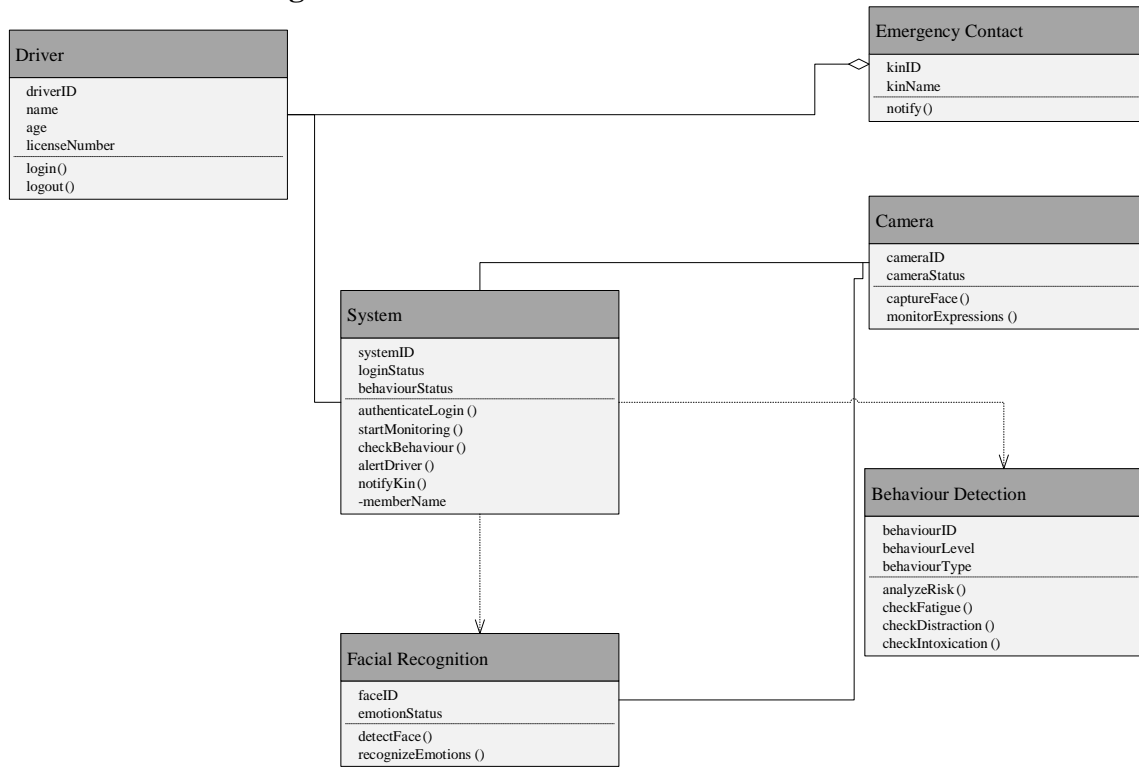


Figure 8: UML Class Diagram

Figure 8 illustrates the structure of the system, with classes representing different components such as the driver, system, facial recognition, camera, behaviour detection, and emergency contact. Each class includes attributes (e.g., driverID, name, behaviourID) and methods (e.g., login (), detectFace (), notify ()). The relationships between the classes are shown using associations, and the system's interactions with different modules like behaviour detection and facial recognition are highlighted.

5.2.5 Entity Relationship Diagram

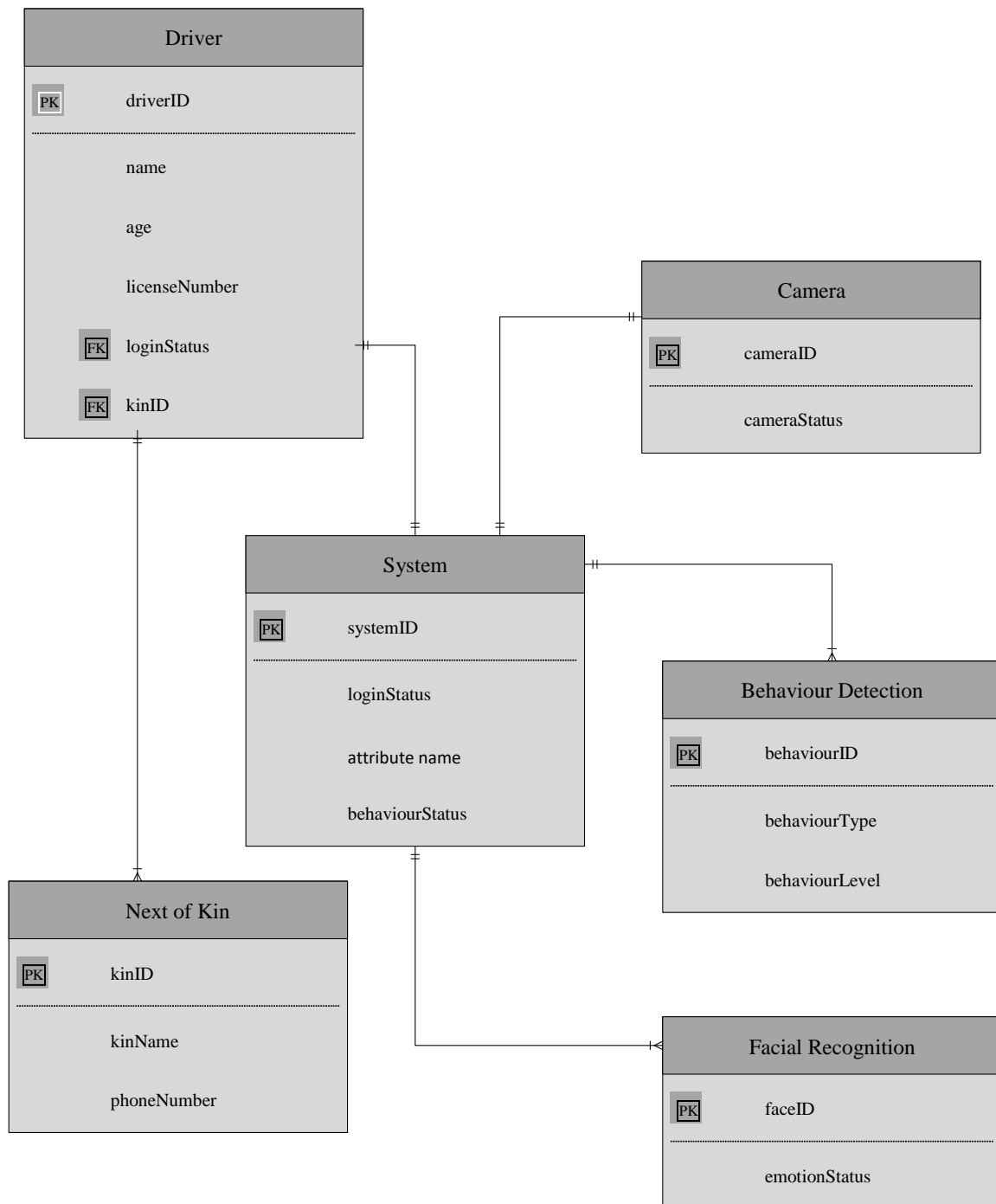


Figure 9: Entity Relationship Diagram

Figure 9 illustrates the relationships between various entities in the Driver Monitoring and Behavioural Analysis System. It includes entities such as Driver, Camera, System, Behaviour Detection, Facial Recognition, and Next of Kin. Each entity displays its attributes and identifies primary keys (PK) and foreign keys (FK), indicating how they are interconnected. This diagram helps visualize the database structure and how different components interact with each other.

5.2.6 Sequence Diagram

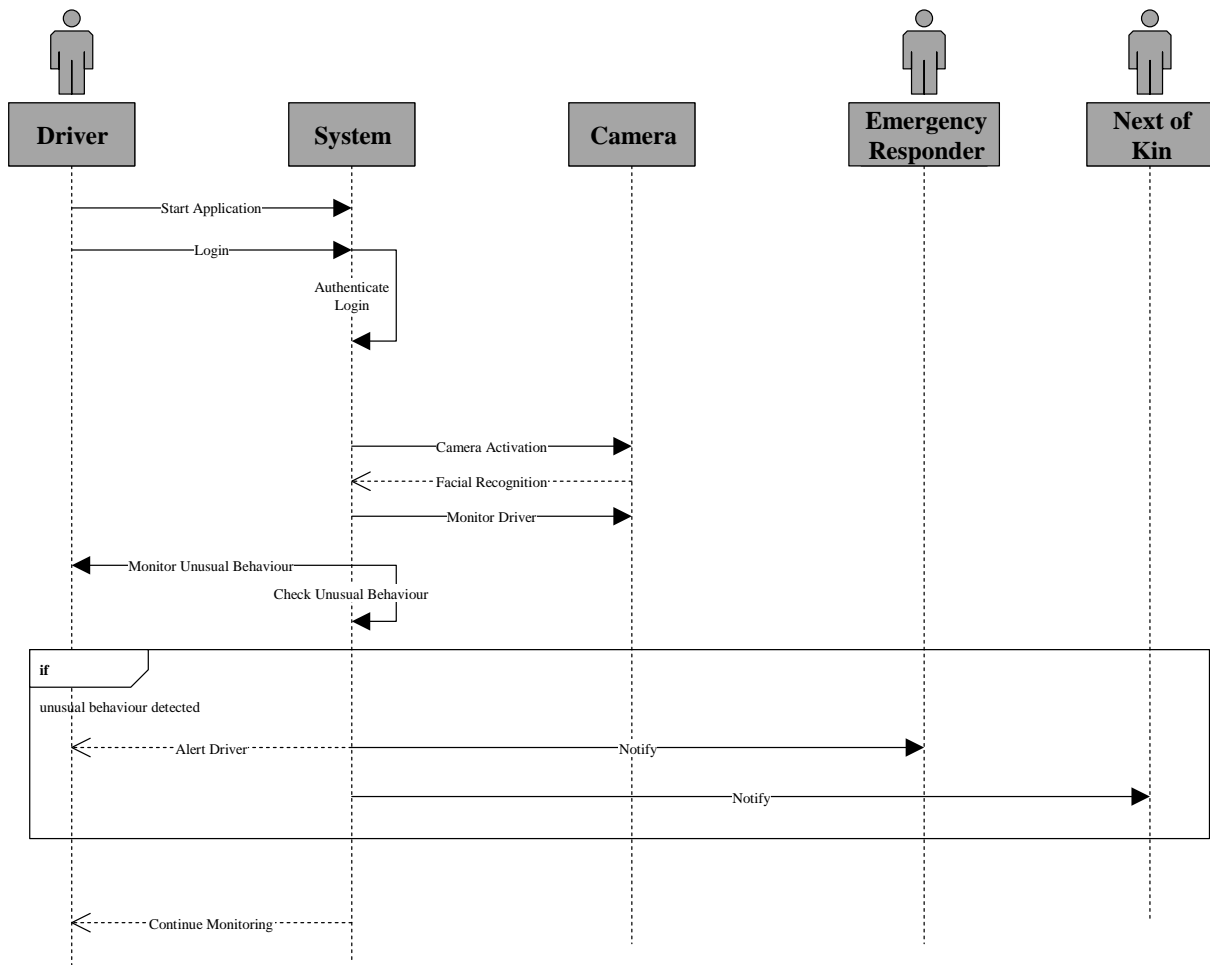


Figure 10: Sequence Diagram

Figure 10 outlines the interaction between the Driver, System, Camera, Emergency Responder, and Next of Kin during the monitoring process. It shows the sequence of events starting from the driver's application login, camera activation, behaviour monitoring, and notification processes if any unusual behaviour is detected. This diagram effectively communicates the flow of information and the timing of events within the system.

5.2.7 Data Flow Diagram

5.2.7.1 Level 0

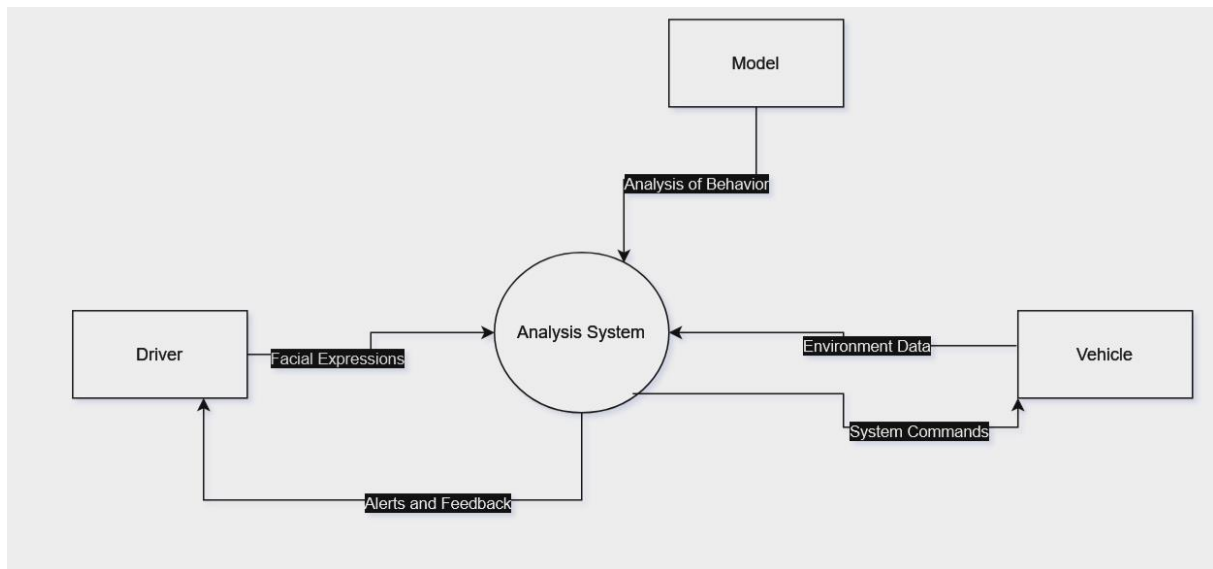


Figure 11: Data Flow Diagram Level 0

Figure 11 simplifies the flow of data within an Analysis System. The Driver provides Facial Expressions as input, while Environment Data is fed in from the Vehicle. The Analysis System leverages a Model to process and analyze the driver's behaviour, sending Alerts and Feedback to the driver and issuing System Commands to the vehicle when necessary. This setup represents a high-level overview of how driver behaviour and environmental factors are monitored, with a focus on real-time feedback and interaction between the driver and vehicle.

5.2.7.1 Level 1

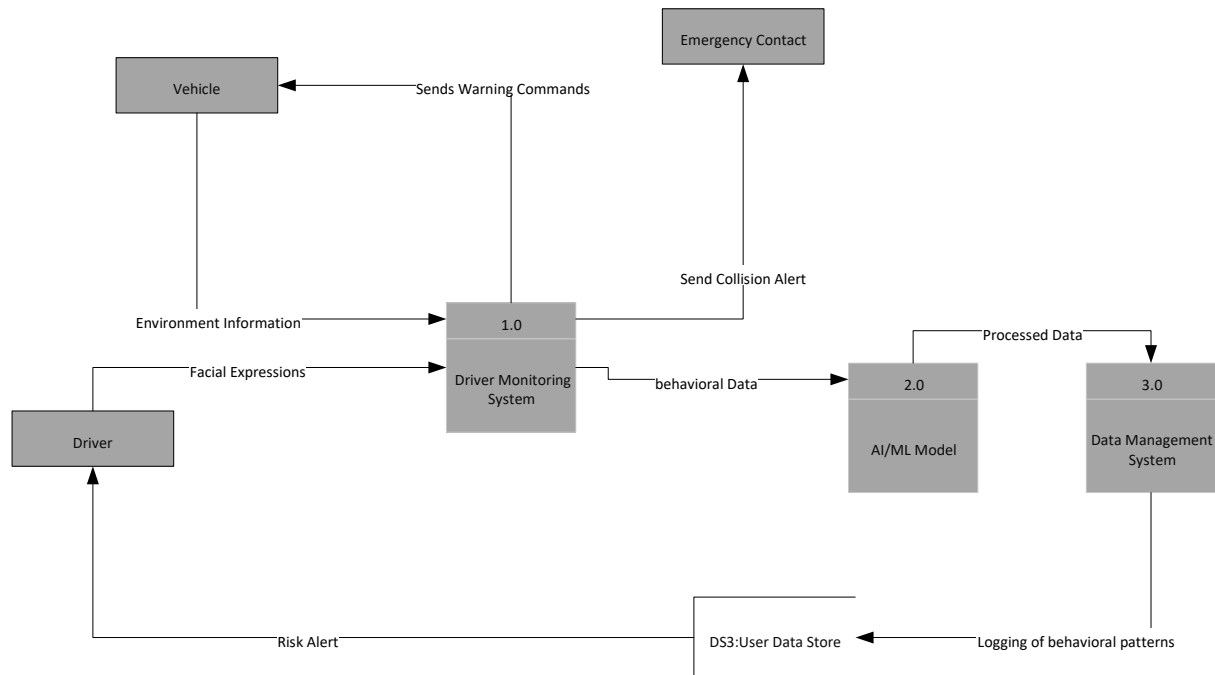
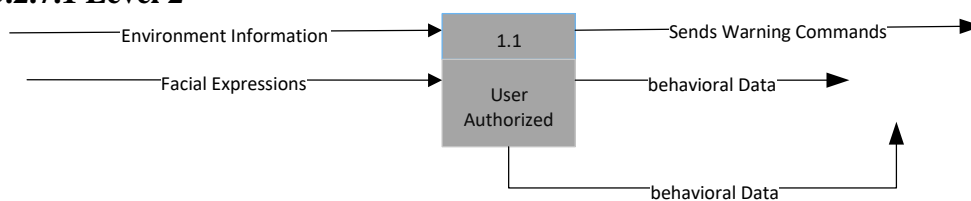


Figure 12: Data Flow Diagram Level 1

Figure 12 represents a Driver Monitoring System for real-time monitoring of a driver's behaviour and environment. It shows data flow from the Driver and Vehicle to the Driver Monitoring System, which collects Facial Expression and Environment Information. The system analyzes this data to assess potential risks and issues alerts (such as Risk Alert to the driver and Warning Command to the vehicle). It also sends Collision Alerts to an Emergency Contact if necessary. Processed Behavioural Data is sent to an AI/ML Model for further analysis, which then feeds into a Data Management System, responsible for storing behavioural patterns and generating reports.

5.2.7.1 Level 2



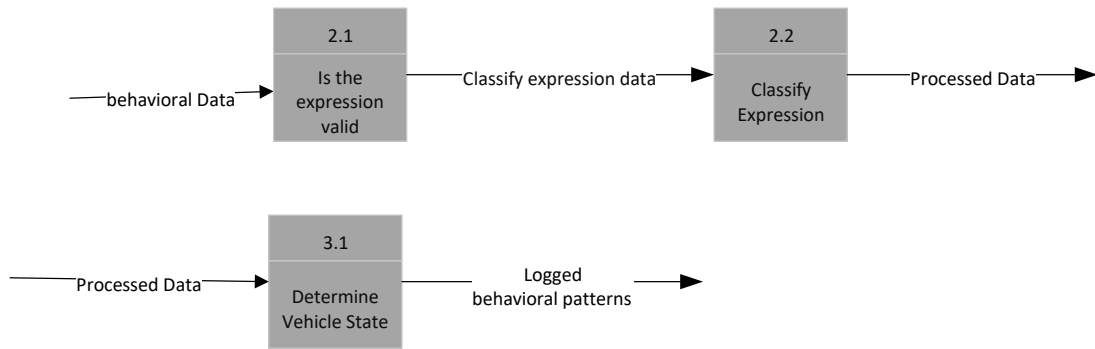


Figure 13: Data Flow Diagram Level 2

Figure 13 represents the workflow of a Driver Monitoring System, specifically focusing on the analysis and response to driver behaviour. It begins with gathering environmental information and facial expressions as inputs to authorize the user. If authorized, the system sends warning commands or processes the behavioural data further. In Step 2.1, the system evaluates whether the expression is valid, filtering out irrelevant or incorrect data. Valid expressions are then classified to interpret driver behaviour, producing processed data that reflects the driver's state. Finally, the system determines the vehicle state based on the processed data and logs any significant behavioural patterns, providing a continuous feedback loop that helps enhance driver safety by detecting and addressing risky behaviours in real time.

5.2.8 Network Diagram

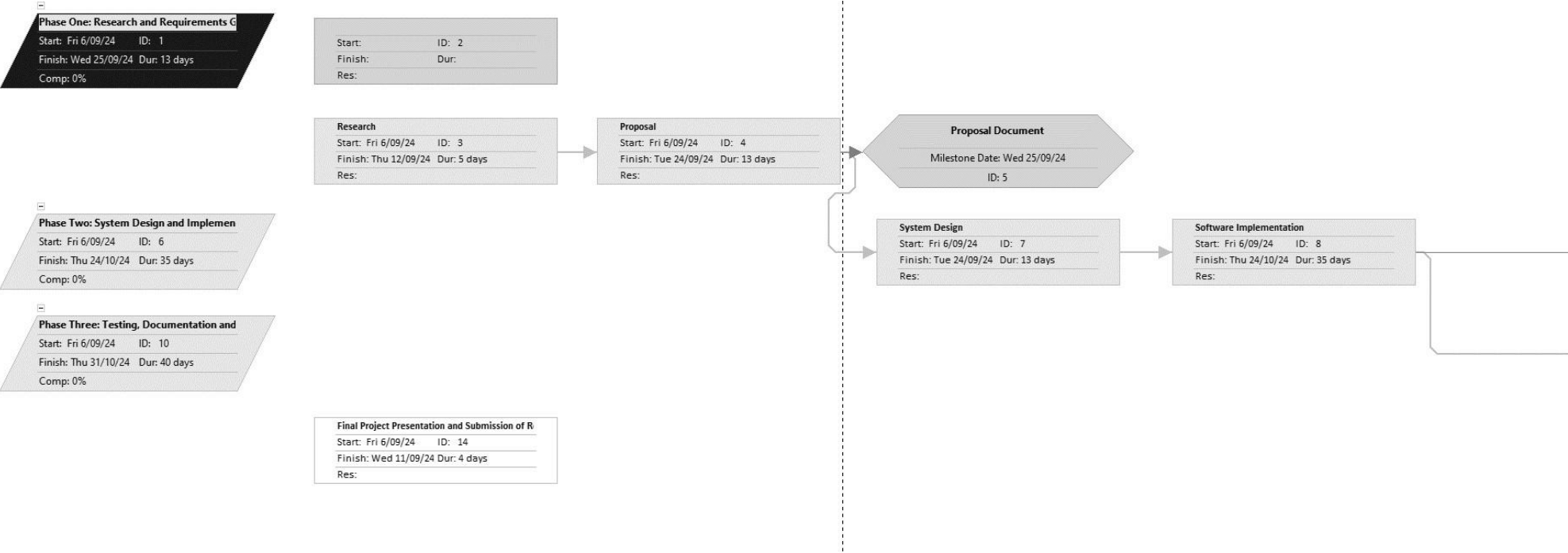


Figure 14: Network Diagram

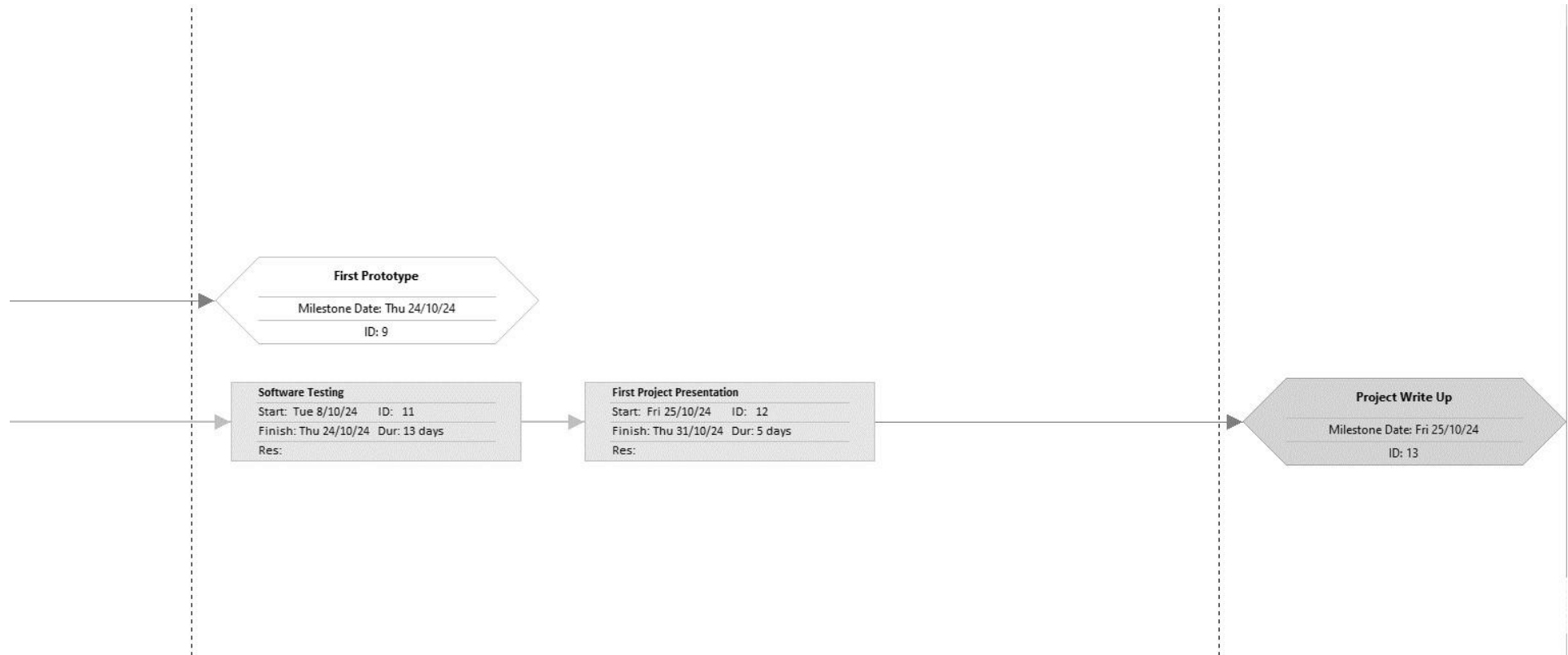


Figure 15:Part 2 Network Diagram

The diagram above shows the logical flow of tasks and their dependencies in a structured format. It emphasizes task relationships, illustrating the sequence of activities and how each task depends on previous ones. This layout highlights critical paths, or sequences of tasks that directly impact the project's completion date, helping project managers identify key dependencies and potential bottlenecks in the workflow. It provides a comprehensive view of the project schedule and workflow dependencies.

5.2.9 Wireframes

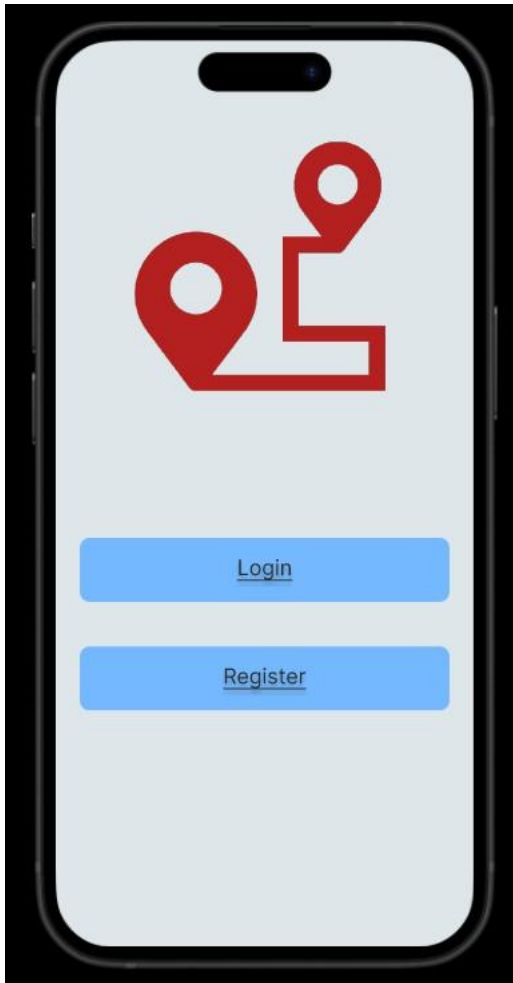


Figure 16: Startup Page

The application's first startup page is depicted in this page diagram. The driver will have the choice to register if this is their first time using the application, or log in if they are already a user. They are then taken to the appropriate pages based on their preferences.

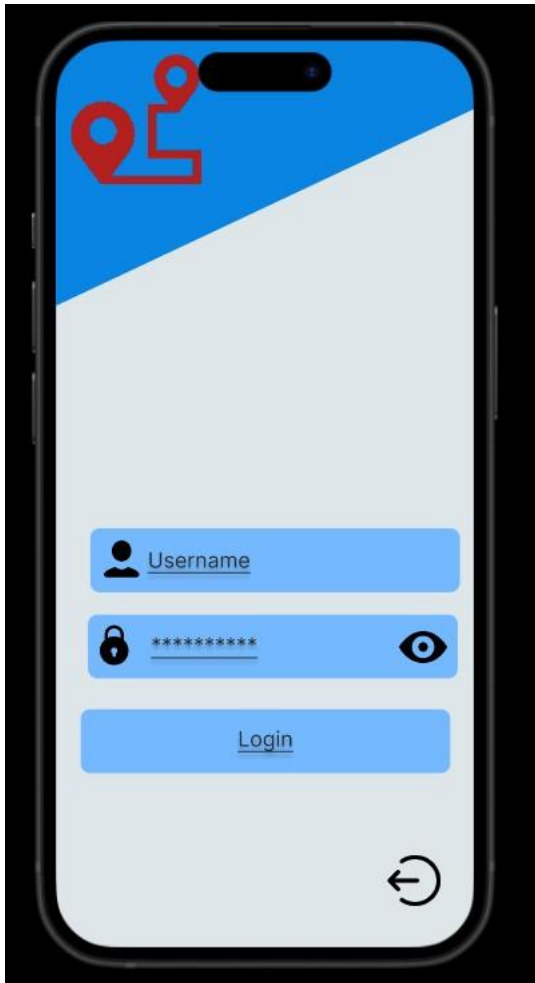


Figure 17: Login Page

Figure 17 above illustrates the required credentials that are needed before access to the application is granted. They will be required to input their username and password, which acts as an authentication method and can be retrieved from a Firebase data store or via localized user authentication.

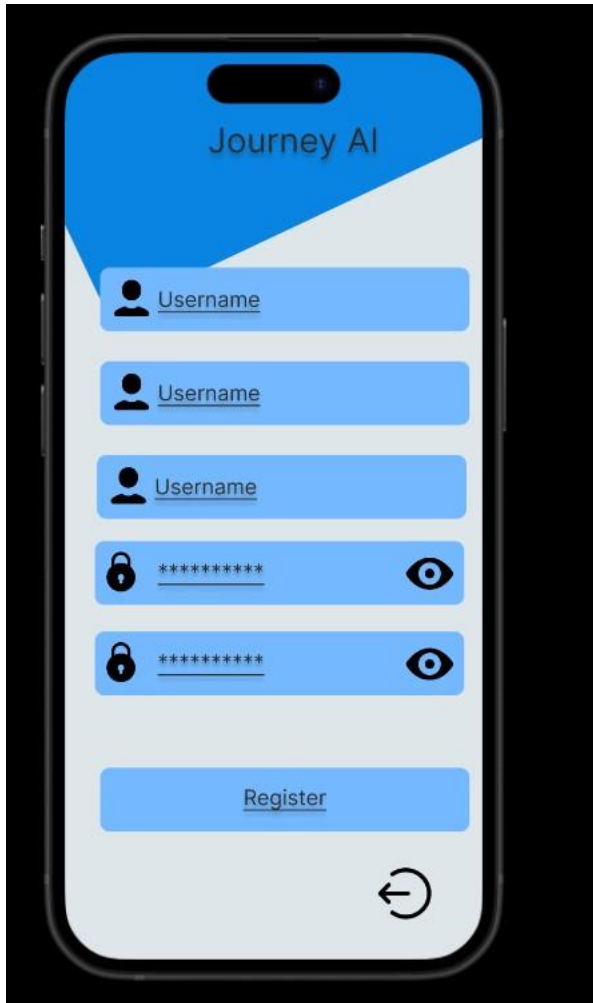


Figure 18: Sign up Page

The sign up page above demonstrates the credentials required to register the new user. The user will provide the necessary credentials to register them so as to be able to access the application.

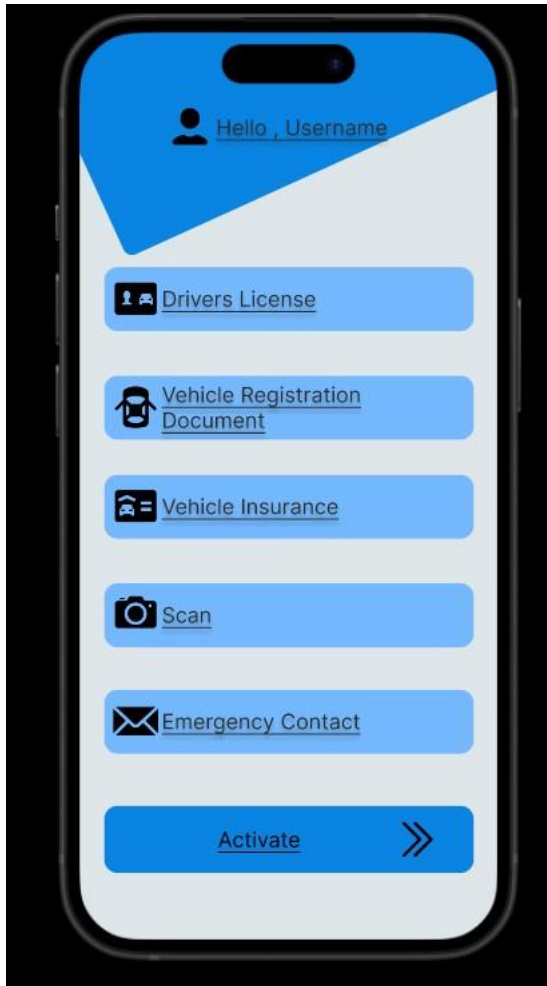


Figure 19: Get Started Page

The page above serves as a pre-activation interface for the app, where users must upload key documents like their driver's license, vehicle registration, and insurance, along with adding an emergency contact. Each section likely provides options to upload files or use the device's camera to scan documents, ensuring all essential information is collected for verification. Once completed, users can press "Activate" to finalize the setup and access the app's main features.

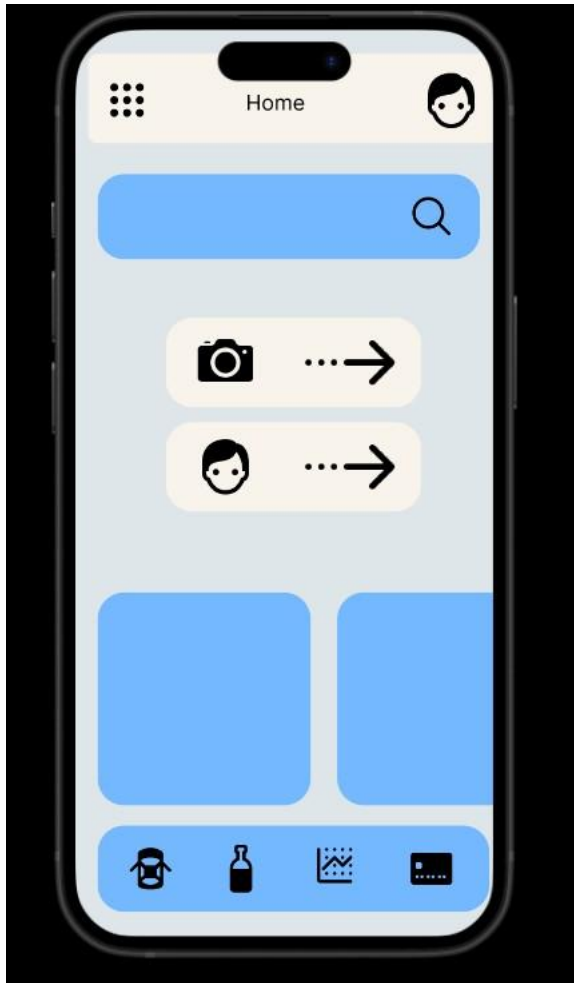


Figure 20: Home Page

Figure 20 illustrates the various features available in the application, allowing the user to see metrics on the vehicle status, begin scanning of the environment or begin scanning of the driver. Additionally, they have the ability to view activity data and make payments as needed.

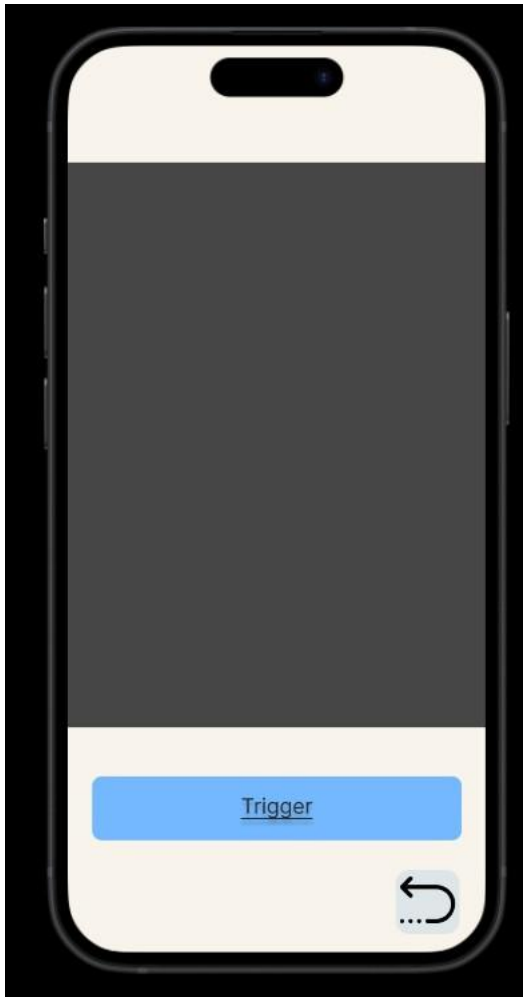


Figure 21: Facial Recognition Page

The page illustrated above represents a simple interface for a facial recognition feature within an app. The main section likely functions as a camera feed area where the user's face is displayed for analysis. Below the camera view, a "Trigger" button initiates the facial recognition process, which may capture the user's face for identification or analyze facial expressions to detect signs of fatigue or distraction.

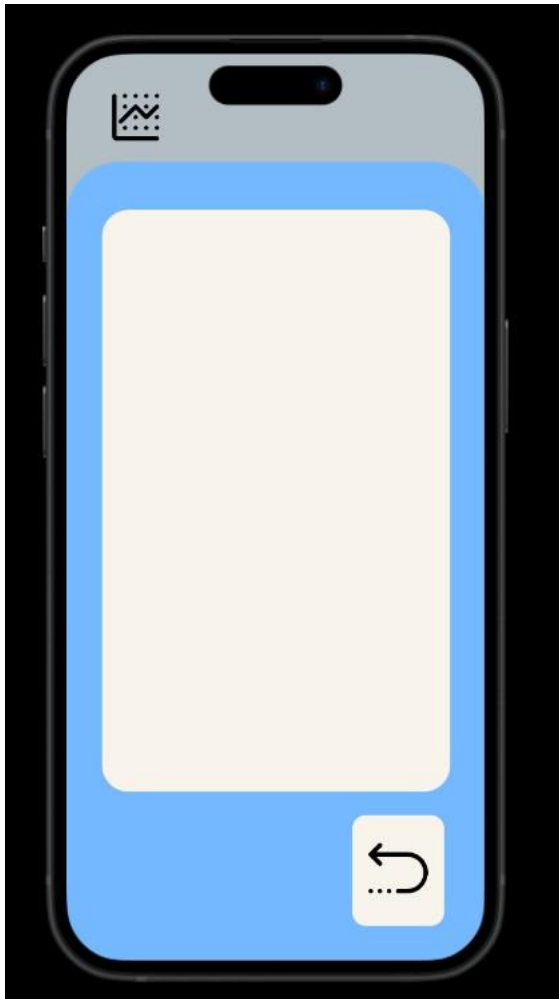


Figure 22: Analysis Page

The page above will demonstrate comprehensive insights based on the data collected from the user. This page would typically include detailed information on the user's recent facial recognition sessions, such as timestamps, detection accuracy, and behaviour patterns observed (e.g., signs of fatigue, distraction, or emotional states like stress or calmness). Visualizations such as graphs or charts may be incorporated to represent trends over time, helping users or administrators quickly interpret the results.

5.3 Functional Requirements

The functional requirements of the DMS outline the essential capabilities needed to fulfil its objectives of enhancing road safety, detecting risky driver behaviour's, and providing real-time alerts and interventions. These requirements specify the actions the system must perform to monitor driver conditions effectively and provide timely feedback.

FR1: Real-Time Driver Monitoring

The system shall continuously monitor the driver's facial expressions, eye movements, and physiological indicators (e.g., heart rate or head position) to detect signs of fatigue, distraction, intoxication, or other risky behaviours in real time.

FR2: Facial Recognition for Fatigue and Distraction Detection

The system shall use AI-powered facial recognition technology to analyze the driver's facial features and movements, identifying potential signs of fatigue or distraction, such as prolonged eye closure, yawning, or head tilting.

FR3: Emotion Detection and Behavioural Analysis

The system shall employ emotion detection algorithms to assess the driver's emotional state (e.g., stress, anger, calmness) and detect behaviours that may indicate impaired driving, providing context to the driver's condition and potential safety risks.

FR4: Real-Time Alerts and Notifications

The system shall issue real-time audio or visual alerts to the driver when risky behaviours, such as signs of fatigue or distraction, are detected. Additionally, the system shall notify fleet managers or designated contacts if critical thresholds are met, ensuring timely interventions.

FR5: User Interface for Real-Time Data Visualization

The system shall provide an intuitive user interface that displays real-time monitoring data, including the driver's current status, alerts, and notifications. This interface shall allow drivers and fleet managers to review ongoing assessments and receive feedback on risky behaviours.

FR5: Personalized Driver Profiles

The system shall support personalized driver profiles, allowing it to adapt to individual driver behaviours and provide tailored monitoring and alert thresholds based on historical data and specific risk factors.

FR6: Data Privacy and Security

The system shall implement security protocols to protect driver data, ensuring that all collected information is securely stored and access is restricted to authorized users only.

5.4 Non-Functional Requirements

The non-functional requirements specify the quality attributes, system constraints, and operational standards that the DMS must meet to ensure reliable, efficient, and user-friendly performance. These requirements address aspects such as system performance, usability, reliability, and security.

NFR1: Performance and Responsiveness

- The system shall process driver monitoring data and provide real-time alerts with minimal delay (less than 2 seconds).
- The system shall maintain efficient resource usage to avoid slowing down vehicle operations or draining power excessively.

NFR2: Reliability and Availability

- The system shall operate reliably under various conditions (e.g., different lighting, weather, and driver behaviours).
- The system shall have a minimum uptime of 99%, ensuring continuous operation.

NFR3: Scalability

- The system shall support scalability to handle individual drivers and large fleet operations without performance degradation.
- The system architecture shall enable future updates and feature additions.

NFR4: Usability and Accessibility

- The user interface shall be intuitive and accessible, requiring minimal training.
- The system shall support multiple languages to cater to diverse user bases.

NFR5: Accuracy and Precision

- The system shall achieve a false positive rate of less than 5% and a false negative rate of less than 3%.
- Emotion detection and behavioural analysis algorithms shall deliver consistent and reliable results.

NFR6: Security and Privacy

- The system shall encrypt stored and transmitted data to protect driver information.
- Access control mechanisms shall restrict data access to authorized personnel only.

NFR7: Compliance and Standards

- The system shall comply with regulations like GDPR for data privacy and ISO 26262 for automotive functional safety.
- It shall adhere to ethical AI standards ensuring fairness and transparency.

NFR8: Maintainability and Modularity

- Modular architecture shall support easy upgrades, maintenance, and feature addition.
- Comprehensive documentation shall be provided for developers to facilitate testing and debugging.

NFR9: Compatibility

- The system shall integrate with different vehicle makes, models, and operating systems (e.g., Android, iOS).
- Compatibility with fleet management software and standard infotainment systems shall be ensured.

NFR10: Environmental Resilience

- The system shall maintain performance under varied lighting, temperature, and vibration conditions.
- Hardware components shall be durable and withstand automotive environments, including prolonged exposure to heat and dust.

CHAPTER 6: IMPLEMENTATION

6.1 Overview

The implementation phase of the Driver Monitoring and Behavioural Analysis System (DMS) involved transforming the system's design and requirements into a fully operational application. This stage encompassed coding, integration, and deployment to ensure the system effectively monitored driver behaviour in real-time, provided actionable alerts, and adhered to the outlined functional and non-functional requirements. Each step was critical in delivering a robust, secure, and scalable solution for enhancing road safety.

6.2 Coding

The coding phase focused on translating the design specifications into functional modules using the chosen technologies and frameworks. The primary technologies included:

- TFLite (TensorFlow Lite): Utilized for running pre-trained models for driver behaviour and emotion detection on mobile devices. TFLite's lightweight nature ensured the AI models worked efficiently in a real-time environment with low resource consumption.
- Flutter: Used for building a cross-platform, user-friendly interface for drivers and fleet managers, ensuring a seamless experience across mobile devices.

Key coding tasks included:

- Driver Behaviour and Emotion Analysis: Integrated TFLite models for detecting driver states, such as drowsiness, distraction, fatigue, and emotional stress, using real-time video feeds. Optimized the pre-trained models to ensure high accuracy and low latency on mobile devices.
- UI Development: Designed and implemented a responsive user interface in Flutter to display live video feeds, driver behaviour summaries.
- Performance Optimization: Refined the integration of TFLite models with the Flutter application to reduce resource consumption and ensure smooth real-time monitoring. Conducted extensive testing to identify and fix bottlenecks in the system for seamless operation under different conditions.

6.2.1 Backend Coding Implementation

The backend code implementation focuses on preparing the system to process driver behaviour and emotion detection efficiently. It ensures seamless integration between the application's AI

model and its real-time functionalities. The backend is optimized to handle tasks such as loading pre-trained TFLite models, processing camera input streams, and generating predictions. This implementation is designed to operate directly on the device, prioritizing low latency and high performance to support the real-time requirements of the DMS.

The following code snippets demonstrate the implementation of the core functionality described above. Each block highlights key aspects of the system's development, including driver behaviour detection, emotion analysis, and real-time alerts.

6.2.1.1 Initializing the Camera and Model

This section initializes the camera and the TensorFlow Lite model when the widget is created. It also sets up a periodic timer to analyze the camera feed regularly.

```
22 | @override
23 | void initState() {
24 |   super.initState();
25 |   initializeCamera();
26 |
27 |   // Set up a periodic timer to run every 7 seconds
28 |   timer = Timer.periodic(Duration(seconds: 7), (Timer t) {
29 |     runModel();
30 |   });
31 | }
```

Figure 23: Initializing the Camera and Model

Line 23: This line calls the parent class's *initState()* method to initialize the state of the widget before custom initialization.

Line 24: This line calls the *initializeCamera()* method to set up the camera for capturing images.

Line 28: This line creates a periodic timer that triggers every 7 seconds to repeatedly run a function, in this case, *runModel()*.

Line 29: Calls the *runModel()* function every 7 seconds to process the camera frame and generate predictions.

6.2.1.2 Camera Initialization

This section fetches available cameras, starts the camera feed, and loads the machine learning model. It acts as the starting point for the system.

```
33 | void initializeCamera() async {
34 |   cameras = await availableCameras(); // Fetch available cameras
35 |   loadCamera();
36 |   loadModel();
37 | }
```


Figure 24: Camera Initialization

Line 34: Asynchronously fetches the list of available cameras on the device and assigns it to the *cameras* variable.

Line 35: Calls the *loadCamera()* method to initialize and start the camera with the fetched camera data.

Line 36: Calls the *loadModel()* method to load the pre-trained machine learning model for inference.

6.2.1.3 Configuring and Starting the Camera

This section sets up the camera stream, specifying the resolution and camera index (e.g., front camera). It also handles live frame updates.

```
39 | loadCamera() {  
40 |   cameraController = CameraController(cameras![1], ResolutionPreset.medium);  
41 |   cameraController!.initialize().then((_) {  
42 |     if (!mounted) return;  
43 |  
44 |     setState(() {  
45 |       cameraController!.startImageStream((CameraImage image) {  
46 |         cameraImage = image;  
47 |       });  
48 |     });  
49 |   });  
50 | }
```

Figure 25: Configuring and Starting the Camera

Line 40: Initializes the *cameraController* with the second available camera (*cameras![1]*) and sets the resolution to *medium* for capturing images.

Line 41: Asynchronously initializes the camera, and once initialization is complete, it proceeds with the following actions inside the *then()* callback.

Line 42: Checks if the widget is still mounted in the widget tree; if not, it returns early to avoid performing actions on an unmounted widget.

Line 44: Triggers a rebuild of the widget by calling *setState()*, indicating that the state has changed and needs to be updated.

Line 45: Starts streaming camera frames by using *startImageStream()* and provides a callback function to handle each frame.

Line 46: Assigns the current camera frame (*image*) to the *cameraImage* variable for further processing.

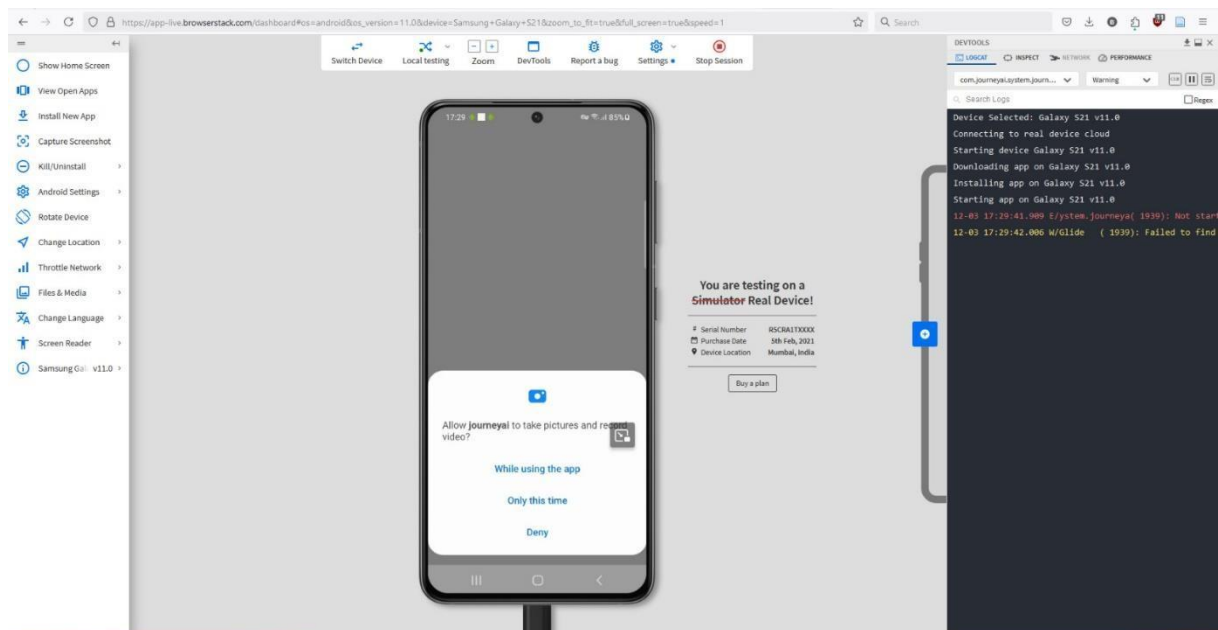


Figure 26: Permissions to initialize the camera

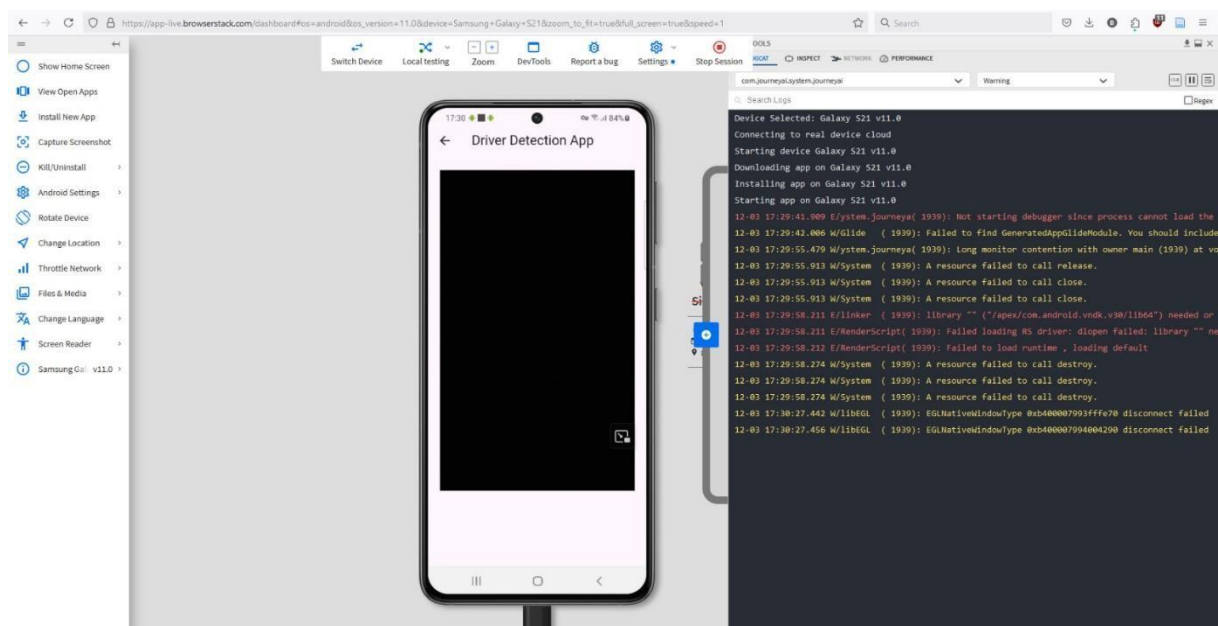


Figure 27: Initializing the camera

6.2.1.4 Running the AI Model

This section processes the live camera feed using the TensorFlow Lite model to generate predictions about driver behaviour or emotions.

```

52 | runModel() async {
53 |   if (cameraImage != null) {
54 |     var predictions = await Tflite.runModelOnFrame(
55 |       |   bytesList: cameraImage!.planes.map((plane) => plane.bytes).toList(),
56 |       |   imageHeight: cameraImage!.height,
57 |       |   imageWidth: cameraImage!.width,
58 |       |   imageMean: 127.5,
59 |       |   imageStd: 127.5,
60 |       |   rotation: 90,
61 |       |   numResults: 2,
62 |       |   threshold: 0.1,
63 |       |   async: true);
64 |
65 |     predictions?.forEach((element) {
66 |       |   setState(() {
67 |         |   output = element['label'];
68 |       |   });
69 |     });
70 |   }
71 | }

```

Figure 28: Running the AI Model

Line 53: Checks if *cameraImage* is not null before proceeding to run the model, ensuring there is a valid image to process.

Line 54: Runs the pre-trained model on the current camera frame using TFLite's *runModelOnFrame()* method and waits for the result asynchronously.

Line 55: Converts the camera frame (*cameraImage*) into a list of byte data (*bytesList*) by mapping over each plane in the image and extracting its bytes.

Line 56: Passes the height of the camera image to the model for processing.

Line 57: Passes the width of the camera image to the model for processing.

Line 58: Sets the mean value (127.5) for normalizing the image data before feeding it into the model.

Line 59: Sets the standard deviation value (127.5) for normalizing the image data before feeding it into the model.

Line 60: Specifies the rotation angle (90 degrees) to apply to the image before feeding it into the model for accurate predictions.

Line 61: Limits the model to return only the top 2 results (predictions) based on the model's output.

Line 62: Sets a threshold of 0.1, meaning the model will return results that have a confidence level greater than 10% (0.1).

Line 63: Indicates that the model inference should be performed asynchronously to avoid blocking the UI thread.

Line 65: Iterates over the prediction results, if they exist (*predictions?*), to process and display the predicted labels.

Line 66: Calls *setState()* to update the widget's state and trigger a UI rebuild.

Line 67: Sets the *output* variable to the predicted label (such as "drunk," "drowsy," etc.) from the current *element* in the predictions.

6.2.1.5 Loading the Model

This section loads the TensorFlow Lite model and its associated label file into memory, preparing the app for inference.

```
73 | loadModel() async {  
74 | |   await Tflite.loadModel(  
75 | | |   model: "assets/model.tflite", labels: "assets/labels.txt");  
76 | | }  
77 |
```

Figure 29: Loading the Model

Line 74: Calls the *loadModel()* method from the Tflite library, which loads a TensorFlow Lite model asynchronously.

Line 75: Specifies the path to the TensorFlow Lite model file (*model.tflite*) located in the assets folder. Also, specifies the path to the labels file (*labels.txt*), which contains the class labels corresponding to the model's output, also located in the assets folder.

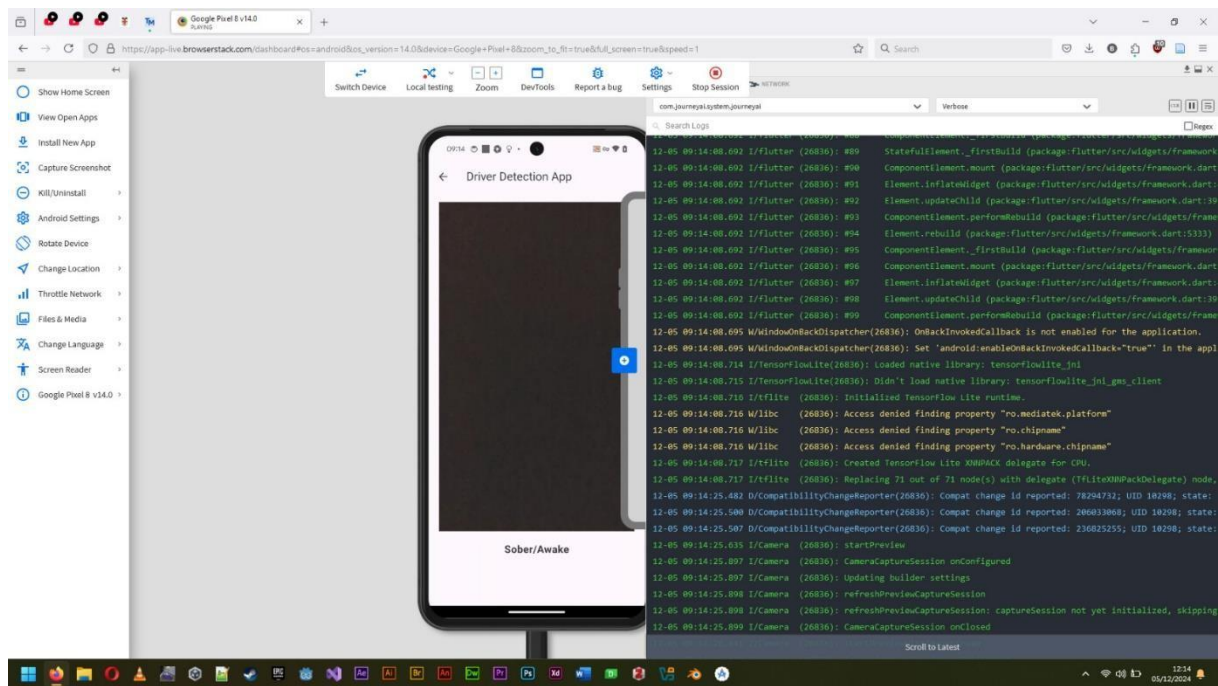


Figure 30: Loading the model

6.2.2 Frontend Coding Implementation

The frontend code implementation utilized Flutter to create a cross-platform interface that was user-friendly and highly responsive. Flutter's flexibility enabled the development of a seamless user experience for drivers and fleet managers, displaying real-time monitoring data and alerts from the backend. The interface facilitates smooth interactions between the driver and the system, with clear visual feedback of AI inferences displayed in real time.

6.2.2.1 UI Development

This section focuses on building the user interface (UI) for the app. The UI is designed to display the live camera feed and the output predictions from the TensorFlow Lite model. It uses Flutter's Scaffold, AppBar, and other layout widgets for a structured and user-friendly experience.

```

86  @override
87  Widget build(BuildContext context) {
88    return Scaffold(
89      appBar: AppBar(title: Text('Facial Recognition App')),
90      body: Column(children: [
91        Padding(
92          padding: EdgeInsets.all(20),
93          child: Container(
94            height: MediaQuery.of(context).size.height * 0.7,
95            width: MediaQuery.of(context).size.width,
96            child: !cameraController!.value.isInitialized
97              ? Container()
98              : AspectRatio(
99                aspectRatio: cameraController!.value.aspectRatio,
100               child: CameraPreview(cameraController!),
101             ),
102          ),
103        ),
104        Text(
105          output,
106          style: TextStyle(fontWeight: FontWeight.bold, fontSize: 20),
107        ),
108      ]),
109    );
110  }
111 }

```

Figure 31: UI Development

Line 87: The *build()* method is called to build the widget tree of the screen. It takes the *BuildContext* as an argument.

Line 88: Returns a *Scaffold* widget, which provides basic structure elements like an app bar, body, etc.

Line 89: Creates an *AppBar* with the title *Facial Recognition App*.

Line 90: The body of the *Scaffold* contains a *Column* widget that allows for vertical arrangement of its child widgets.

Line 91: Adds padding around the child widget inside the *Padding* widget.

Line 92: Specifies 20 units of padding on all sides of the widget.

Line 93: The child of the *Padding* widget is a *Container*, which can hold other widgets.

Line 94: Sets the height of the container to 70% of the screen height using *MediaQuery*.

Line 95: Sets the width of the container to the full width of the screen using *MediaQuery*.

Line 96: If the camera is not initialized (*cameraController!.value.isInitialized* is false), an empty *Container* is displayed.

Line 98: If the camera is initialized, an *AspectRatio* widget is used to maintain the aspect ratio of the camera preview.

Line 99: Sets the aspect ratio of the *AspectRatio* widget to match the camera's aspect ratio.

Line 100: Displays the camera preview using the *CameraPreview* widget, which renders the live camera feed.

Line 104: A *Text* widget is added to display the result of the model's prediction.

Line 105: The *output* variable, which holds the result of the model's prediction, is displayed in the *Text* widget.

Line 106: Applies bold text styling with a font size of 20 to the displayed *output*.



Figure 32: Driver Management System UI

6.3 Integration

Several system modules had to be combined during the integration phase, and the components created during the coding phase had to work together seamlessly. The focus was on achieving smooth communication between the AI-powered modules, the user interface, and the backend services to deliver a cohesive and functional system.

Essential Integration Tasks:

1. **TFLite and Flutter Integration:** Successfully integrated TFLite models into the Flutter application for real-time detection of driver behaviours such as fatigue, distraction, and emotional states. Ensured the smooth processing of video input from the camera and realtime inference results display on the user interface.
2. **Database and Cloud Integration:** Linked Firebase with the Flutter app to store and retrieve driver data, such as logged behaviours, timestamps, and safety alerts. Enabled real-time data syncing to ensure all logged incidents were accessible for live monitoring and postevent analysis.
3. **Cross-Platform Support:** Tested the system on both Android and iOS platforms to ensure consistent performance and interface behaviour across devices.
4. **Testing Interactions:** Conducted thorough integration testing to ensure that all components worked seamlessly together, particularly the interaction between real-time TFLite detections and backend data logging.

6.4 Application Deployment

The deployment phase ensured the application was fully operational and accessible to end users, including drivers and fleet managers. This phase involved configuring the hosting environment, implementing automated deployment pipelines, and monitoring system performance postlaunch. Deployment Tasks:

1. **Hosting Configuration:**
 - Used Firebase Hosting to deploy the Flutter application, ensuring fast and reliable access for all users.
 - Optimized hosting for low-latency data exchange, critical for real-time monitoring and notifications.
2. **Continuous Integration/Continuous Deployment (CI/CD):**
 - Set up CI/CD pipelines using tools such as GitHub Actions to automate build, testing, and deployment processes.
 - Ensured that future updates or patches to the app could be deployed seamlessly with minimal downtime.
3. **Mobile App Store Distribution:**

- Prepared the app for publication on Google Play Store and Apple App Store, ensuring compliance with platform requirements.
 - Included detailed documentation and app descriptions to guide users on installation and usage.
4. Post-Deployment Monitoring:
- Configured Firebase Analytics to monitor app performance and track user interactions, such as the frequency of unsafe behaviour alerts.
 - Implemented tools like Sentry for real-time error tracking and resolution to ensure smooth operation post-launch.
5. User Feedback Collection:
- Incorporated feedback mechanisms within the app to allow drivers and fleet managers to report issues or suggest improvements.
 - Used feedback data to plan subsequent updates and feature enhancements.
6. Maintenance and Support:
- Set up a support system for troubleshooting and resolving technical issues encountered by users.
 - Scheduled periodic updates to maintain compatibility with evolving hardware, operating systems, and industry standards.

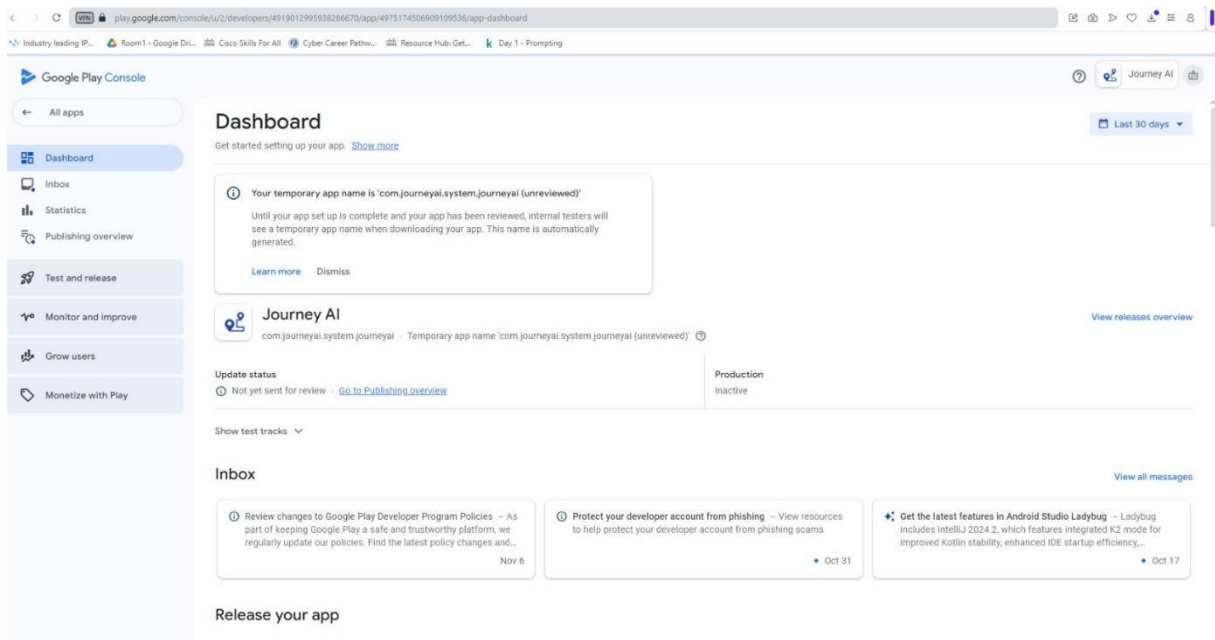


Figure 33: Proof of Deployment

Figure 29 showcases the Google Play Console dashboard for the "Journey AI" application. It draws attention to important details like the app's interim, unreviewed name

(com.journeyai.system.journeyai) and management options for publishing, testing, user growth, and revenue. While the "Inbox" section shows messages regarding policy updates, phishing protection, and new features in Android Studio, the "Update status" part shows that the app has not yet been submitted for review. This gives developers a summary of the resources and upgrades they may use to manage and enhance their app on the platform.

6.5 Summary

In this chapter, we explored the complete process of developing a facial recognition application, from coding the backend to building the frontend interface. The chapter focused on the integration of camera functionality and machine learning models to detect and classify images in real-time, with a particular emphasis on ensuring the smooth interaction between the app's various components. We discussed the implementation of both the backend logic, which handles image processing and model execution, and the frontend, which displays the live camera feed and model predictions to the user. Additionally, the chapter covered the integration and deployment phases, ensuring that all elements work together efficiently, leading to a fully functional application ready for deployment.

CHAPTER 7: TESTING & EVALUATION

7.1. Introduction

The evaluation chapter focuses on assessing the effectiveness, functionality, and reliability of the Driver Monitoring System (DMS) application. This chapter outlines the testing methodologies, test cases, and evaluation metrics used to ensure the system meets its intended goals. The chapter also highlights the feedback collected from users, the challenges faced during testing, and the overall impact of the application on its stakeholders, including fleet managers and drivers. The testing process aimed to verify the system’s performance under realworld conditions, focusing on key functionalities such as real-time monitoring, accurate detection of driver behaviours, and seamless integration of AI models with the mobile application. Furthermore, the system's ability to provide timely and actionable feedback to ensure safe driving practices was rigorously evaluated.

7. 2. System Testing

System testing is an essential phase of the development process to ensure that the DMS works as expected under various conditions and meets the functional and non-functional requirements. In this section, we outline the primary issues tested, including Performance, Authentication, and Scalability. For each issue, we will break it down into subsections with detailed test cases, code, and screenshots.

7.2.1 Authentication Testing

Authentication tests verify that only authorized users can access the system, and the credentials are verified securely.

Table 2: Authentication Test Case

Test ID	Description	Preconditions	Test Steps	Expected Results	Test Score
TC01	Verify user can sign up with valid credentials	User is on the Sign-Up page	1. Enter valid email. 2. Enter valid password. 3. Enter driver	User account is created, and the user is redirected to the dashboard	Pass

			details. 4. Click Sign up button.		
TC02	Verify user can log in with valid credentials	User is already registered and is on the login page	1. Enter valid email. 2. Enter valid password. 3. Click "Login" button.	User is logged in and redirected to the dashboard.	Pass

Table 2 shows two test cases used to test for authentication. Verifying that a user may successfully register using legitimate credentials is the goal of the first test case, TC01. Assuming the user is on the Sign-Up page, the test runs. The process entails hitting the SignUp button after providing a working email address, password, and driving information. It is anticipated that the user will be taken to the dashboard when their account has been created. This test was successfully completed. The login functionality is the main focus of test case number two, TC02. The user is presumed to be on the Login page and already registered. The process entails clicking the Login button after providing a working email address and password. The user should log in and be taken to the dashboard as the anticipated outcome. This test also passed successfully.

```
E:\JourneyAI
flutter test test/login_page_test.dart
00:05 +0: LoginPage login test
Login test passed!
00:05 +1: All tests passed!
```

Figure 34: Test Code Result (login)

```
E:\JourneyAI
flutter test test/signup_test.dart

A new version of Flutter is available!
To update to the latest version, run "flutter upgrade".

00:15 +0: RegistrationPage registration test
Registration test passed!
00:15 +1: All tests passed!
```

Figure 35: Test Code Result (Signup)

7.2.2 Scalability Testing

Scalability testing ensures that the DMS can handle increased workload demands, such as processing a higher number of recognition requests without compromising performance. This test evaluates the system's ability to scale up seamlessly under different load conditions.

Table 3: Scalability Test Case

Test ID	Description	Preconditions	Test Steps	Expected Result	Test score
TC03	Verify multiple user's emergency contacts can be fetched from firebase all at once	Multiple users are logged in at the same time All the logged in users must have added emergency contacts	1. Check for the currently logged in users' ID 2.query firebase for the currently logged in multiple users emergency contacts and their details	All the emergency contacts for the currently logged in users are fetched in less than ten seconds	Pass

Table 3 shows a single test case for testing scalability of the system. The system's capacity to retrieve emergency contacts for numerous users at once from Firebase is evaluated in the test case TC03. Several users must be logged in at the same time and have emergency contacts added for each user. Finding the IDs of the users who are currently logged in and requesting the emergency contact information for each of them from Firebase are the test steps. In less than ten seconds, the system should retrieve the emergency contact details for any person who is logged in. This test passed successfully.

```

W/ytes.journeys(25467): Accessing hidden field Ljava/net/Socket;=>ispt:Ljava/net/SocketImpl; (unsupported, reflection, allowed)
W/ytes.journeys(25467): Accessing hidden method Ljava/security/spec/ECParameterSpec;=>setCurveName(Ljava/lang/String;)V (unsupported, reflection, allowed)
D/TrafficStats(25467): tagSocket(506) with statsTag=0xffffffff, statsUid=-1
W/FinalizerDaemon(25467): type=1400 audit(0.0:13270137): avc: denied { getopt } for path="/dev/socket/usap_pool_primary" scontext=u:r:untrusted_app:s0:c128,c258,c512,c768 tcontext=u:r:zygote:s0
tclass=unix_stream_socket permissive=0 app=com.assaf.carrefouruae
I/flutter (25467): Scalability test completed: All 100 users fetched emergency contacts successfully.
I/flutter (25467): 00:04 +2: (tearDownAll)
I/flutter (25467): 00:04 +3: All tests passed!
All tests passed.

```

Figure 36: Scalability test code result (Firebase)

7.2.3 Performance Testing

Performance testing evaluates the system's efficiency, including response time, under various conditions. This ensures that the DMS provides timely alerts and recognitions for real-time driver monitoring.

Table 4: Performance Test Case

Test ID	Description	Preconditions	Test Steps	Expected Result	Test score
TC04	Verify location is fetched in less than 10 seconds for efficiency in sending the alert	User has the app running in foreground or background with location settings turned on	<ol style="list-style-type: none"> 1. Check if user's location is enabled 2. if location is disabled ask for permission to access location 3. query the user's current location and show the street and city 	User accurate location is fetched in less than ten seconds and show the users street and city	Pass

TC05	Verify firebase querying of emergency contacts during alert periods is of less than ten seconds	User must be logged in User must have added emergency contacts	1. query firebase for the currently logged in user's emergency contacts and their details	Currently logged in users emergency contacts are fetched in less than ten seconds	Pass
------	---	--	---	---	------

Table 4 shows two test cases used to test for performance of the emergency response notification system mobile application. In order to provide timely alerts, the first test case, TC04, assesses how well it can retrieve the user's location in less than ten seconds. The test makes the assumption that the application is operating in the background or foreground with location settings enabled. Verifying whether the user's location services are enabled, asking for consent if they are not, and getting the user's current location in order to display their street and city are the procedures involved. It is anticipated that the user's street and city will be displayed along with the precise location in less than ten seconds. This test was successfully completed. The second test case, TC05, evaluates how quickly emergency contact details can be retrieved from Firebase during alert times. The user must be logged in and have emergency contacts added in order to meet the preconditions. The test involves querying Firebase for the logged-in user's emergency contact details. The expected result is that the emergency contact information is fetched in less than 10 seconds. This test also passed successfully.

```

W/system.journal(22303): Accessing hidden method Ldalvik/system/VMTask;->getStackClass2()Ljava/lang/Class; (unsupported, reflection, allowed)
W/system.journal(22303): Accessing hidden method Ljava/security/spec/ECParameterSpec;->getCurveName()Ljava/lang/String; (unsupported, reflection, allowed)
I/ProviderInstaller(22303): Installed default security provider GmsCore_OpenSSL
D/Performance(22303): TagSocket(1a3) with stateTag=00000000, stateId=0
W/system.journal(22303): Accessing hidden field Ljava/net/Socket;->impl:Ljava/net/SocketImpl; (unsupported, reflection, allowed)
W/system.journal(22303): Accessing hidden method Ljava/security/spec/ECParameterSpec;->setCurveName(Ljava/lang/String;)V (unsupported, reflection, allowed)
D/Performance(22303): TagSocket(1a4) with stateTag=00000000, stateId=0
I/Flutter (22303): No widgets found at Offset(107.9, 519.3).
W/FinalizerDaemon(22303): type=1400 audit(0.0:13265725): avc: denied { getapt } for path="/dev/socket/usap_pool_primary" context=u:r:untrusted_app:s0:c126,c256,c512,c768 ts=contextu:r:zygote:s0
tclass=unix_stream_socket permission=app-com.journal.system.journal
D/PlusInputMethodManagerInternal(22303): get InputMethodManager extension: com.android.internal.view.IInputMethodManager$Stub$Proxy@6a7e0aef
I/Flutter (22303): Time taken to fetch emergency contacts: 7274 ms
I/Flutter (22303): 00:07 *2: (tearDownAll)
I/Flutter (22303): 00:07 *3: All tests passed!
All tests passed.

```

Figure 37: Performance test code result (Firebase query)

7.3 User Acceptance Testing

User Acceptance Testing (UAT) plays a critical role in ensuring that the DMS meets the objectives and requirements outlined during the design and development phases. UAT ensures

that the system performs effectively in real-world conditions, effectively monitoring driver behaviour, detecting fatigue, distraction, and other risky behaviours, and providing real-time alerts to prevent accidents. The test will be based on the specific aims and objectives of the project, which include designing and developing a cost-effective, user-friendly system that integrates AI, machine learning algorithms, and advanced sensors for real-time monitoring.

7.3.1 Real-time Driver Monitoring System

This subsection tests the functionality of real-time tracking and facial recognition for identifying a driver's state. The tests focus on verifying the accuracy and responsiveness of the system in detecting sober and active conditions and triggering alerts when unsafe behaviour is identified.

7.3.1.1 Test Case: Detect a sober person with confidence

This test case checks the system's ability to accurately detect a sober person with a confidence level greater than 0.5 from facial recognition data. The system should flag the person as sober when the confidence is sufficiently high.

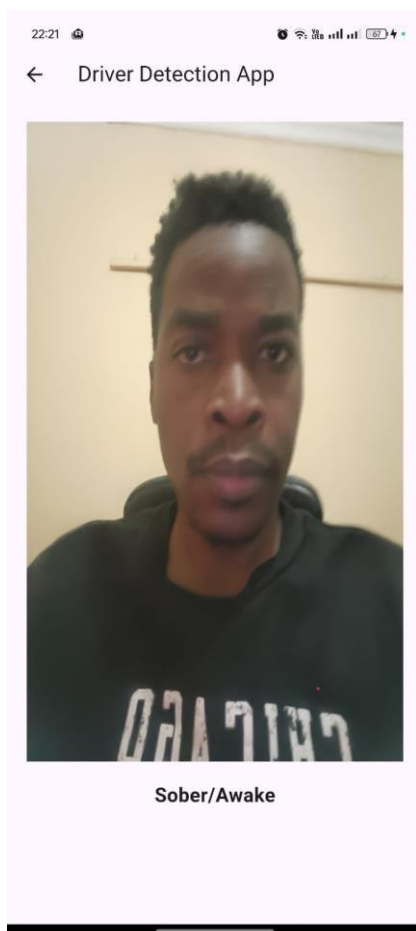


Figure 38: Sober person

Figure 30 illustrates the detection of a sober person by the system, successfully recognizing the state with a confidence level greater than 0.5.

Code:

```
7 | test('should detect a sober person with confidence > 0.5', () {
8 |   final recognitions = [
9 |     {'label': 'sober', 'confidence': 0.7},
10 |    {'label': 'active', 'confidence': 0.6}
11 |   ];
12 |
13 |   final result = FacialRecognitionService.checkForState(recognitions, 'sober');
14 |   print('Test 1: Detect sober with confidence > 0.5 - Result: $result');
15 |   expect(result, isTrue);
16 | });
```

Figure 39: Sober state Test Code

Line 7: This line defines a test case using the test function from the Flutter testing library. The description of the test, 'should detect a sober person with confidence > 0.5', indicates the functionality being tested: verifying that the system correctly identifies a sober person if their confidence score exceeds 0.5.

Line 8 - 11: This line initializes some variable recognitions, which simulates the input from the facial recognition system. It contains a list of maps where each map represents a detection result with two keys: 'label' (the state detected) and 'confidence' (the confidence score of the detection). In this test, two states—'sober' with a confidence score of 0.7 and 'active' with a confidence score of 0.6—are provided as sample inputs.

Line 13: This line calls the checkForState method from the FacialRecognitionService class, passing the recognitions list and the target state 'sober' as arguments. The method processes the input to determine if the specified state ('sober') is detected with a confidence score greater than 0.5. The result of the check (true or false) is stored in the result variable.

Line 14: This line prints the test description and the result of the method call to the console. It provides useful debugging information by showing whether the system correctly identified a sober person based on the input.

Line 15: This line is the assertion for the test. It checks if the result variable is true, meaning that the method successfully detected the state 'sober' with a confidence score above 0.5. If the assertion passes, the test is considered successful; otherwise, it fails.

Table 5: Sober State Test Case

Test ID	Description	Preconditions	Test Steps	Expected Results	Test Score
TC06	Detect a sober person with confidence > 0.5	The system should have facial recognition capabilities.	1. Input recognitions with label "sober" having confidence 0.7, and "active" having confidence 0.6.	The system should detect the "sober" state with confidence > 0.5 and return true.	Pass

This test case ensures that the system can accurately detect when the driver is in a "sober" state, provided the confidence level exceeds 0.5. The simulated input includes a 'sober' label with a confidence of 0.7, which meets the condition. The method successfully identifies the state and returns true. This verifies the system's ability to recognize safe driving states, which is essential for monitoring driver conditions in real-time.

```
PS C:\Program Files\AndroidStudioProjects\dms_facial_recognition> flutter test
00:02 +0: FacialRecognitionService Tests should detect a sober person with confidence > 0.5
Test 1: Detect sober with confidence > 0.5 - Result: true
```

Figure 40: Test Code Result for detecting a sober state

7.3.1.2 Test Case: Detect a drunk person with confidence

This test case ensures that the system correctly detects a drunk person based on the facial recognition data when the confidence level is above 0.5.

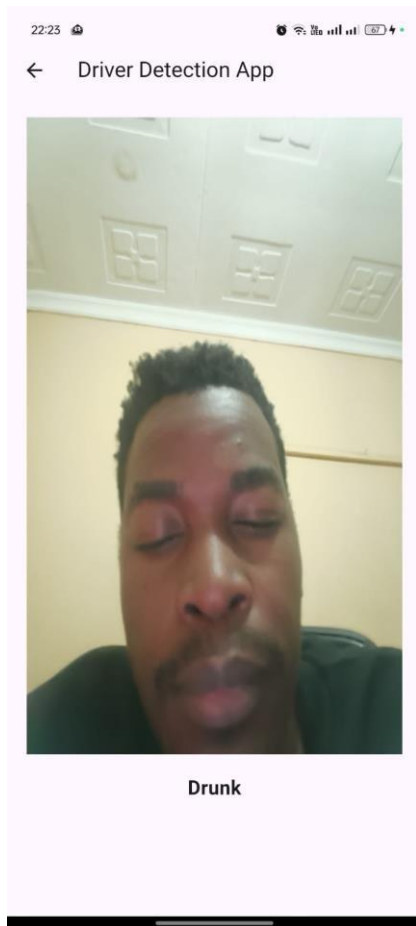


Figure 41: Drunk person

Figure 32 illustrates the detection of a drunk person by the system, successfully recognizing the state with a confidence level greater than 0.5.

Code:

```
18 | test('should detect a drunk person with confidence > 0.5', () {
19 |     final recognitions = [
20 |         {'label': 'drunk', 'confidence': 0.8},
21 |         {'label': 'active', 'confidence': 0.4}
22 |     ];
23 |
24 |     final result = FacialRecognitionService.checkForState(recognitions, 'drunk');
25 |     print('Test 2: Detect drunk with confidence > 0.5 - Result: $result');
26 |     expect(result, isTrue);
27 | });
28 |
```

Figure 42: Drunk person Test Code

Line 18: This line defines a test case named 'should detect a drunk person with confidence > 0.5'. It aims to test whether the system can correctly identify a drunk person if their confidence score exceeds 0.5. The test function is used to group this specific test with a descriptive title.

Line 19 – 22: A recognitions list is initialized with sample input data representing detections made by the facial recognition system. It contains two maps: one indicates the 'drunk' state with a confidence score of 0.8, and the other indicates the 'active' state with a confidence score of 0.4. These inputs simulate a scenario where the system has detected different states with varying confidence levels.

Line 24: The checkForState method of the FacialRecognitionService class is invoked, passing the recognitions list and the target state 'drunk' as arguments. This method checks if the target state 'drunk' exists in the input and whether its confidence score exceeds the threshold of 0.5. The method's return value (true or false) is stored in the result variable.

Line 25: This line prints the description of the test and the result obtained from the checkForState method to the console. It helps in debugging by displaying whether the system successfully identified the 'drunk' state based on the input.

Line 26: The expect function asserts that the result variable is true. This confirms that the checkForState method correctly identified the 'drunk' state with a confidence score greater than 0.5. If this assertion fails, the test will report an error, indicating a problem in the detection logic. If it passes, the test is considered successful.

Table 6: Drunk state Test Case

Test ID	Description	Preconditions	Test Steps	Expected Results	Test Score
TC07	Detect a drunk person with confidence > 0.5.	The system should have facial recognition capabilities	1. Input recognitions with label "drunk" having confidence 0.8, and "active" having confidence 0.4.	The system should detect the "drunk" state with confidence > 0.5 and return true.	Pass

This test case evaluates the system's ability to detect a "drunk" state when the confidence level is above 0.5. The input includes a 'drunk' label with a confidence value of 0.8, satisfying the

threshold. The method accurately identifies this state and returns true, highlighting its capability to detect risky behaviours and initiate appropriate alerts to ensure driver and road safety.

```
00:02 *1: FacialRecognitionService Tests should detect a drunk person with confidence > 0.5  
Test 2: Detect drunk with confidence > 0.5 - Result: true
```

Figure 43: Test Case Result to detect a drunk person

7.3.1.3 Test Case: Not detect a state with confidence

This test case ensures that the system does not detect any state when the confidence level of the recognition data is less than or equal to 0.5.

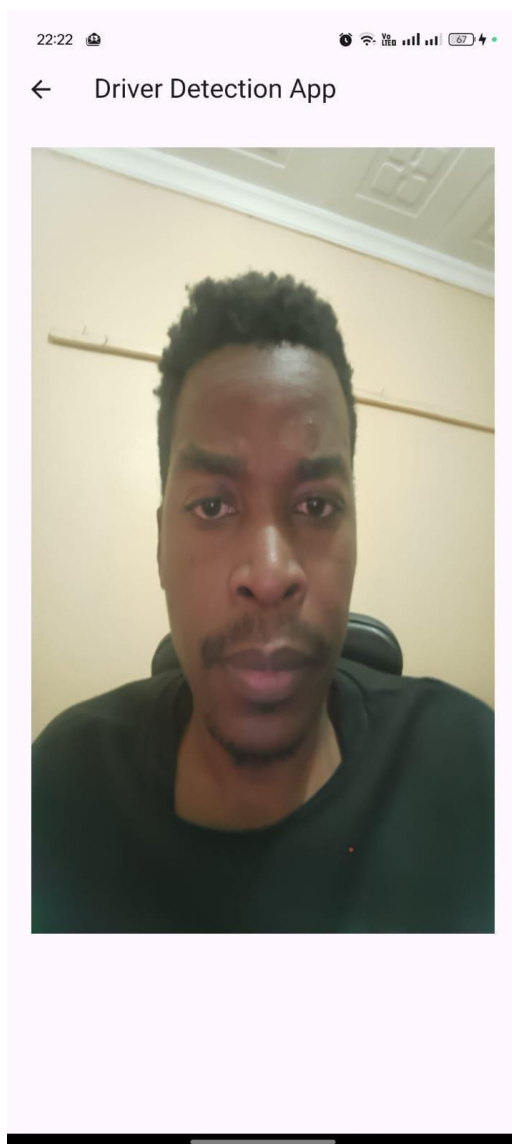


Figure 44: No state detected

Figure 35 illustrates the no state detected by the system, successfully recognizing the state with a confidence level greater than 0.5.

Code:

```
41 | test('should not detect a state when confidence <= 0.5', () {
42 |     final recognitions = [
43 |         {'label': 'sober', 'confidence': 0.4},
44 |         {'label': 'drunk', 'confidence': 0.3}
45 |     ];
46 |
47 |     final result =
48 |         FacialRecognitionService.checkForState(recognitions, 'sober');
49 |     print('Test 4: Do not detect sober with confidence <= 0.5 - Result: $result');
50 |     expect(result, isFalse);
51 | });
```

Figure 45: No State Test Code

Line 41: This is the definition of a test case. It describes what the test will check, in this case, ensuring that a state (e.g., "sober") is not detected when the confidence level is less than or equal to 0.5. The description inside the test function helps to identify what the test does.

Line 42: This defines a variable recognition that holds a list of dictionaries, where each dictionary represents a recognition result from a facial recognition model. Each entry contains a label (such as "sober" or "drunk") and the corresponding confidence score.

Line 43: This dictionary represents a recognition result indicating that the model detected "sober" with a confidence score of 0.4. Since this is below the threshold of 0.5, it should not be considered as a valid detection.

Line 45: Another recognition result, this time for "drunk" with a confidence score of 0.3. Like the previous entry, this score is also below the threshold for detection.

Line 47 – 48: This line invokes the method checkForState from the FacialRecognitionService class, passing in the recognitions list and the string 'sober'. The method checks if any of the recognition results have a confidence score above 0.5 for the specified state (in this case, "sober").

Line 49: This line prints a message to the console, showing the result of the test case. It helps to display whether the detection logic correctly identified that "sober" was not detected due to the low confidence score (0.4). The printed result will be either true or false based on the actual output.

Line 50: This is the key assertion in the test case. The expect function is used to compare the actual output (result) with the expected outcome (isFalse).

Table 7: No state Test Case

Test ID	Description	Preconditions	Test Steps	Expected Results	Test Score

TC09	Do not detect a state	The system should have	1. Input recognitions	The system should not	Pass
	when confidence is ≤ 0.5	facial recognition capabilities	with label "sober" having confidence 0.4, and "drunk" having confidence 0.3.	detect "sober" due to low confidence and return false.	

This test case assesses the system's ability to avoid false positives when the confidence level is 0.5 or below. The simulated input includes a 'sober' label with a confidence of 0.4, which does not meet the threshold. The method correctly returns false, ensuring that only states with high confidence are flagged. This helps maintain the system's accuracy and reduces unnecessary alerts.

```
Test 4: Do not detect sober with confidence <= 0.5 - Result: false
00:02 +4: FacialRecognitionService Tests should return false when no matching state is detected
```

Figure 46: TestCase to not detect a state with confidence

7.3.1.4 Test Case: No Matching State Detected

This test case verifies that the system correctly returns false when no matching states (such as "sober," "drunk," or "active") are detected from the input recognition data.

Driver Detection App

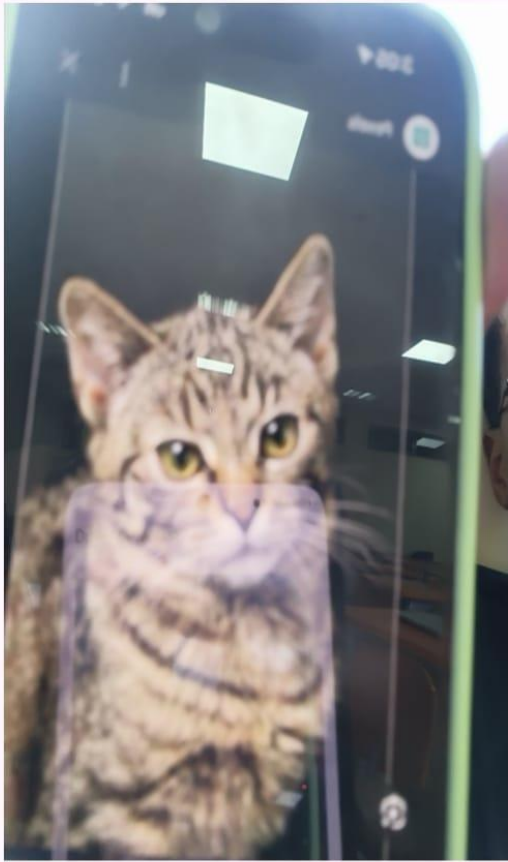


Figure 47: No matching state detected

Figure 43 illustrates an image of a cat therefore no matching state detected by the system, successfully recognizing the state with a confidence level greater than 0.5.

Code:


```

53 | test('should return false when no matching state is detected', () {
54 |     final recognitions = [
55 |         {'label': 'cat', 'confidence': 0.9},
56 |         {'label': 'dog', 'confidence': 0.8}
57 |     ];
58 |
59 |     final result =
60 |         FacialRecognitionService.checkForState(recognitions, 'sober');
61 |     print('Test 5: No matching state detected - Result: $result');
62 |     expect(result, isFalse);
63 | });
64 | });
65 | }

```

Figure 48: No matching state Test Code

Line 53: This defines the test case with a description. It specifies that the test will check whether the system returns false when there is no recognition result matching the requested state (in this case, "sober"). The description helps identify the test's intent.

Line 54: This line declares the variable `recognitions` and initializes it with a list containing dictionaries. Each dictionary represents a facial recognition result, with the label describing what was detected (e.g., "cat" or "dog") and the confidence value representing the model's certainty in that label.

Line 55: The first dictionary in the list indicates that the recognition model detected the label "cat" with a confidence score of 0.9. This is a high confidence value, but it is irrelevant to the test since the test is checking for the label "sober."

Line 56: The second dictionary indicates that the recognition model detected the label "dog" with a confidence score of 0.8. Like the first entry, this is also irrelevant to the test, as we are testing for the "sober" label, which is absent in the recognitions list.

Line 59 - 60: This line invokes the `checkForState` method of the `FacialRecognitionService` class. It passes the `recognitions` list (which contains "cat" and "dog") and the target state, 'sober'. The method will check whether any of the recognitions in the list match the "sober" label and whether their confidence score is above the threshold (0.5). Since there is no "sober" label in the list, the method should return false.

Line 61: This line prints a message to the console, displaying the result of the test case. It helps the developer understand what the output of the test is. The result should be false because there is no "sober" label in the recognitions list, so the system correctly returns false.

Line 62: This is the assertion in the test case. The expect function checks whether the result returned by checkForState matches the expected value (isFalse).

Table 8: No matching state detected

Test ID	Description	Preconditions	Test Steps	Expected Results	Test Score
TC010	Return false when no matching state is detected	The system should have facial recognition capabilities	1. Input recognitions with labels "cat" having confidence 0.9, and "dog" having confidence 0.8.	The system should return false as no matching "sober" label exists.	Pass

This test case examines how the system behaves when the input does not include the specified state. The input contains labels unrelated to the target state (e.g., 'cat' and 'dog'), both with high confidence levels. The method correctly returns false, demonstrating that it does not falsely detect a state when it is not present in the data. This ensures the robustness of the system by preventing incorrect classifications.

```
00:02 +4: FacialRecognitionService Tests should return false when no matching state is detected
Test 5: No matching state detected - Result: false
```

Figure 49: No matching state is detected

7.3.2 Alert System through SMS

This section focuses on the implementation and testing of the SMS-based alert system in the Driver Monitoring System (DMS). The alert system is designed to send real-time notifications to drivers or fleet managers when risky behaviours, such as fatigue or distraction, are detected. By leveraging SMS as a communication medium, the system ensures that critical alerts are delivered promptly and reliably, even in areas with limited internet connectivity. The following subsections include the code implementation, screenshots, and detailed test cases to validate the functionality of the SMS alert system.

Table 9: Alert system through SMS Test Case

Test ID	Description	Preconditions	Test Steps	Expected Result	Test Score
TC11	Verify alert SMS is sent when driver is detected as drunk/drowsy	Driver is detected as drunk/drowsy	1. Detect drunk/drowsy condition. 2. Verify SMS is sent to the first emergency contact. 3. Reply "2" to escalate and verify next contact is alerted. 4. Verify information is sent to admin web app. 5. Check admin web app for updates.	Alert SMS is sent to the first emergency contact.	Pass

Table 10 shows the sending of alerts through SMS test case. This test scenario confirms that when a driver is identified as intoxicated or sleepy, an alert SMS with the relevant information can be sent. The prerequisite is that the motorist must have been found to be intoxicated or sleepy. Identifying the driver's condition, validating that the first emergency contact receives an alert SMS, and confirming that the SMS was received are the test phases. The test also includes making sure that the information is transmitted to the admin web app, checking the admin web app for updates, and responding with "2" to escalate the alarm to the next emergency contact. The anticipated outcome is that the admin web app is updated with the pertinent data and the alert SMS is delivered as specified. This test passed successfully.

```

44: group('sendAlert', () {
45:     late MockTelephony mockTelephony;
46:     late TestDriverStateService testDriverStateService;
47:
48:     setUp(() {
49:         mockTelephony = MockTelephony();
50:         testDriverStateService = TestDriverStateService(mockTelephony);
51:     });
52:
53:     test('should send alert with correct message and location', () async {
54:         final contact = {'name': 'John Doe', 'phone': '1234567890'};
55:         final message = 'This is a test alert';
56:
57:         await testDriverStateService.sendAlert('test', message, contact);
58:
59:         // Verify that the SMS was sent with the correct message
60:         expect(
61:             () async => await mockTelephony.sendSms(
62:                 to: contact['phone'],
63:                 message: '$message\nLocation: Test Location\nReply 1=Ack, 2=Esc',
64:             ),
65:             returnsNormally,
66:         );
67:     });
68:
69:     test('should handle SMS sending failure', () async {
70:         final contact = {'name': 'John Doe', 'phone': 'fail'};
71:         final message = 'This is a test alert';
72:
73:         await testDriverStateService.sendAlert('test', message, contact);
74:
75:         // Verify that the SMS sending failure is handled
76:         expect(
77:             () async => await mockTelephony.sendSms(
78:                 to: contact['phone'],
79:                 message: '$message\nLocation: Test Location\nReply 1=Ack, 2=Esc',
80:             ),
81:             throwsException,

```

Figure 50: Alert system Test code

Line 46: Defines a test group named 'sendAlert' to logically group related test cases for the sendAlert functionality.

Line 47: Declares a late variable mockTelephony of type MockTelephony (a mock object for testing SMS functionality).

Line 48: Declares another late variable testDriverStateService, which will be the service under test.

Line 50: The setUp () function ensures this code runs before each test in the group.

Line 51: Initializes the mockTelephony object before each test.

Line 52: Instantiates testDriverStateService with mockTelephony injected, setting up the dependency for the service.

Line 54: Starts a test case with a descriptive name, 'should send alert with correct message and location'.

Line 55: Defines a contact object containing the recipient's name and phone number.

Line 56: Defines a test message string.

Line 58: Calls the `sendAlert` method on `testDriverStateService` with the alert type ('test'), message, and contact details. This simulates sending the alert.

Line 61: The `expect` function asserts that:

Line 62: The `mockTelephony.sendSms()` method is called with:

- The phone number from the contact.
- A formatted message string that includes location and reply options.

Line 65: The operation completes without throwing exceptions (`returnsNormally`).

Line 67: Starts another test case, 'should handle SMS sending failure'.

Line 68: Creates a contact object with a phone value of 'fail' (likely a test condition to simulate an error).

Line 69: Defines the test message string.

Line 71: Calls the `sendAlert` method, similar to the previous test, but with a failing contact.

Line 75: The `mockTelephony.sendSms()` method is called with the failing phone number and formatted message.

Line 79: The method throws an exception (`throwsException`), confirming proper error handling.

```
flutter test test/sendalert_test.dart
00:04 +0: sendAlert should send alert with correct message and location
Full message: This is a test alert
Location: Test Location
Reply 1=Ack, 2=Esc
Full message length: 63 characters
Sending alert to John Doe at 1234567890
```

Figure 51: Test Code Result I

```
flutter test test/sendalert_test.dart
00:08 +0: sendAlert should handle SMS sending failure
Full message: This is a test alert
Location: Test Location
Reply 1=Ack, 2=Esc
Full message length: 63 characters
Sending alert to John Doe at fail
Failed to send SMS: type 'MockTelephony' is not a subtype of type 'Telephony' in type cast
00:08 +1: All tests passed!
```

Figure 52: Test Code Result II

7.3.3 System escalation of alerts to secondary emergency contact's

Table 10: Escalation of alerts to emergency contacts

Test ID	Description	Preconditions	Test Steps	Expected Result	Test Score
TC12	Verify emergency contact can acknowledge/escalate alert	Emergency contact receives alert SMS	1. Reply "1" to acknowledge and check Firebase.	Acknowledgment and escalation are logged in Firebase.	Pass

Table 11 shows the Escalation of alerts to secondary emergency contact's test. The test scenario intended to confirm that an emergency contact is capable of acknowledging or elevating an alert is displayed in this table. The alarm SMS must have been received by the emergency contact. In the test phases, the emergency contact acknowledges the alarm by answering "1" and then verifies that the escalation and acknowledgment are recorded in Firebase. It is anticipated that Firebase would correctly register the acknowledgment and any escalation. This test passed successfully.

7.4 Key Findings

The evaluation of the Driver Monitoring System (DMS) yielded several key insights into the system's performance, usability, and user experience.

1. Functionality

- The DMS successfully implemented its core features, including real-time driver behaviour monitoring, facial recognition, fatigue detection, and AI-driven alerts.
- Users (drivers and fleet managers) found the interface easy to use, with clear instructions and effective feedback on driver behaviour and performance.
- The integration with real-time camera feeds and AI model predictions worked as expected, detecting distractions, fatigue, and other safety-critical behaviours.

2. Performance

- The system showed strong performance, with the AI model providing quick and accurate predictions in real-time without significant delays.
- The platform demonstrated robust performance under various environmental conditions, including different lighting scenarios and vehicle movements.

- Stress testing indicated that the system could handle multiple drivers and simultaneous camera streams, maintaining consistent processing speeds.

3. User Satisfaction

- Feedback from both drivers and fleet managers was overwhelmingly positive, especially regarding the real-time monitoring capabilities and the ability to receive immediate feedback on driving behaviour.
- Drivers appreciated the non-intrusive nature of the system, with alerts delivered in a timely manner without disrupting their driving.
- Fleet managers found the system valuable for monitoring fleet-wide behaviour and improving driver safety.

4. Issues and Improvements

- A few drivers reported occasional inconsistencies in the facial recognition accuracy, particularly when wearing face masks or in low-light conditions. These issues will be addressed with improvements to the model and camera settings.
- Some fleet managers suggested incorporating a feature that would allow them to customize alert thresholds based on driver performance, which could be useful for tailored interventions.

7.5 Impact Analysis

The Driver Monitoring System had a significant impact on its intended users, helping enhance road safety and driver performance while meeting its objectives of real-time monitoring and AI-driven insights.

1. Drivers

- Drivers benefited from immediate feedback regarding unsafe behaviours such as fatigue or distraction, helping them to adjust their driving habits for improved safety.
- The system's real-time monitoring and alerts helped reduce instances of accidents caused by fatigue or lack of attention, leading to a safer driving environment.
- Several drivers noted that the system's non-invasive nature allowed them to stay focused on the road without being distracted by the monitoring process.

2. Fleet Managers

- Fleet managers found the DMS invaluable for monitoring driver behaviour across multiple vehicles, providing actionable insights that helped improve fleet safety and reduce incidents.
- The system helped them identify high-risk drivers who needed additional training or intervention, which directly contributed to a decrease in safety-related incidents and enhanced operational efficiency.

3. Vehicle Manufacturers and Insurance Companies

- Vehicle manufacturers found the integration of advanced monitoring technologies a valuable selling point, allowing them to offer enhanced safety features in their vehicles.
- Insurance companies showed interest in the system as a potential tool for reducing premiums and improving claims management through better risk management based on real-time data on driver behaviour.

7.6 Summary

This chapter focused on the comprehensive evaluation of the Driver Monitoring System (DMS) prototype, encompassing critical aspects such as scalability, performance, and facial recognition accuracy. Detailed test cases were created and executed to validate the system's functionality under various conditions. The results demonstrated the system's ability to handle real-time driver condition monitoring effectively, including detecting sober, active, or risky states with high accuracy.

The scalability tests confirmed that the system could manage multiple recognition requests simultaneously without any significant decline in performance. Performance testing verified the system's quick response times, making it suitable for real-time applications. Additionally, specific test cases provided insights into the system's reliability when processing varied input data, ensuring robust and consistent behaviour.

Through screenshots, code snippets, and detailed explanations of the test outcomes, this chapter highlighted the system's readiness for deployment in real-world scenarios. The evaluation also identified opportunities for future improvements, such as optimizing response times further or expanding recognition capabilities to include more driver states.

In conclusion, the tests validated that the DMS prototype successfully meets its core objectives of enhancing driver safety through real-time monitoring, ensuring scalability, and maintaining high performance in demanding conditions.

CHAPTER 8: CONCLUSION

This chapter provides an overview of the entire project, summarizing the key objectives, processes, and results, and offering insights into how the project was implemented and its outcomes. The project began with an introduction to the history of driver monitoring systems (DMS), highlighting their growing importance in improving road safety. The problem statement outlined the need for effective monitoring solutions to reduce accidents caused by distracted or fatigued driving, setting the stage for the project's development.

A comprehensive literature review was conducted, examining existing DMS solutions both globally and locally. International companies like Magna and Smart Eye, as well as local solutions such as TrailMyCar and Karooooo, were discussed. The strengths and weaknesses of these systems were analyzed, providing valuable context for the project's objectives and identifying gaps that the proposed system aimed to address. This review laid the foundation for understanding the state of the industry and where improvements were needed.

The project's aims and objectives were clearly defined in the following chapter. The general aim was to enhance driver safety through a monitoring system, while the specific objectives focused on designing, implementing, and evaluating a system that would be functional, reliable, and user-friendly. These objectives helped guide the project, ensuring that it met the necessary requirements for success.

The proposed project was structured into three key phases: research and requirements gathering, system design and implementation, and testing, documentation, and presentation. A detailed program of work outlined the software, hardware, and budgetary requirements, providing a roadmap for the project's execution. A Gantt chart was included to manage the timeline and ensure timely completion of tasks, helping the team stay on track throughout the project.

System analysis and design were thoroughly explored through the presentation of various system models, including architecture diagrams, use case diagrams, flowcharts, UML class diagrams, and sequence diagrams. These visual aids helped clarify the structure and operation of the system. Functional and non-functional requirements were also defined, ensuring the system would meet performance expectations, scalability, and usability standards.

The implementation chapter focused on the coding, integration, and deployment of the driver monitoring system. It provided details of both backend and frontend code implementation, outlining how the system was developed and integrated. The deployment strategy ensured the

system could be effectively launched and used in real-world conditions. This chapter highlighted the practical application of the system's design and addressed the challenges encountered during implementation.

In the testing and evaluation chapter, the system was rigorously tested to ensure it met the outlined requirements. Various system tests and test cases were presented, verifying that all components worked as expected. Key findings were discussed, and the impact of the system was evaluated to ensure its effectiveness in real-world applications. The importance of robust testing was emphasized to guarantee the system's reliability and success before deployment.

The conclusion summarized the entire project, revisiting the key objectives and outcomes. It reflected on the challenges faced during the development and implementation phases while acknowledging the success of the project in contributing to road safety. The project demonstrated the potential of driver monitoring systems to enhance safety through advanced technologies like computer vision and real-time analytics. Finally, the chapter suggested areas for future improvements and further research, highlighting the system's adaptability to evolving technological advancements and needs in the transportation sector.

8.1 Achievements

Throughout the DMS, significant progress was made in both the technical implementation and the achievement of project milestones. Key accomplishments include the successful integration of real-time facial recognition capabilities to assess driver states such as sobriety, alertness, and activity levels. Additionally, the alert system, which notifies fleet managers and drivers about risky behaviours, was implemented, enhancing safety measures within the system. The system's scalability and performance were optimized to handle multiple vehicle data streams efficiently, ensuring that the system performs well under heavy load conditions. These technical achievements were complemented by rigorous testing, which ensured the system's accuracy and reliability in real-world scenarios.

Furthermore, the project demonstrated successful collaboration among team members, each contributing their expertise in machine learning, software development, and testing to ensure the project's success. The incorporation of user feedback, especially in improving the user interface and experience, further refined the system's usability. These achievements culminated in a system that not only meets the technical specifications but also provides valuable tools for improving road safety and fleet management.

8.2 Challenges encountered

During the course of the project, several challenges were encountered, particularly in the training of the model. One of the key difficulties was ensuring the accuracy of the machine learning model used for facial recognition and driver monitoring. Training the model required a large, diverse dataset that was not always readily available, making it difficult to achieve high levels of accuracy. Additionally, the environmental variability, such as lighting conditions, driver position, and camera quality, impacted the performance of the model in real-world scenarios. The limited processing power of certain hardware also posed challenges when trying to run the model in real-time. Another significant challenge was the integration of the various components, such as the camera system, model, and user interface, which required careful synchronization to ensure smooth and efficient performance. Despite these challenges, iterative testing and refinement were employed to overcome these obstacles, resulting in a functional system, though there remains room for further improvements.

8.3 Future Works

The future work for this project revolves around refining the current system to address the challenges encountered during implementation. One area for improvement is the accuracy of the facial recognition model. This can be achieved by incorporating more diverse datasets and exploring advanced techniques like transfer learning or fine-tuning existing pre-trained models to improve performance in varied real-world environments. Additionally, the system can be enhanced by integrating more sophisticated sensors and cameras that offer better image quality, which would help mitigate issues related to lighting and resolution. Another area for future development is expanding the system's capabilities to monitor additional aspects of driver behaviour, such as monitoring heart rate, body posture, and other physiological indicators, which could improve the system's ability to detect driver fatigue or intoxication. Finally, user experience improvements, such as more intuitive interfaces and personalized feedback for drivers, would make the system more accessible and effective in promoting safe driving practices.

REFERENCES

Dietrich Manstetten, F. B.-J. (2021). The Evolution of Driver Monitoring Systems: A Shortened Story on Past, Current and Future Approaches How Cars Acquire Knowledge About the Driver's State. *In 22nd International Conference on HumanComputer Interaction with Mobile Devices and Services (MobileHCI '20)*, 1-6.

Dohun Kim, H. P. (2023). Real-time driver monitoring system with facial landmark-based eye closure detection and head pose recognition. *Scientific Reports*, 1-14.

Karooooo. (2023). Retrieved from Karooooo.


Magna. (n.d.). Retrieved from Magna.

Smart Eye. (n.d.). Retrieved from Smart Eye:
<https://www.smarteye.se/solutions/automotive/driver-monitoring-system/>

TrailMyCar. (n.d.). Retrieved from TrailMyCar.

APPENDICIES



I. LOGBOOK

Logbook for Final Project	
 United States International University-Africa	
Student ID No.	664723
Name	Onchiew Michelle
Email	marchiew@ustu.ac.ke
Concentration	Forensic Information Technology and Cybercrime
Year and Semester	Fall 2024
Supervisor	Dr. Paul Okanda

Page 1 of 8

Notes on Use of the Project Logbook

1. The student and supervisor must arrange regular supervisory meetings to review progress and make plans for the project. It is the purpose of the Project Logbook to document these meetings and therefore build up a record of the student's progress throughout the project.
2. The student should prepare for the supervisory meeting by deciding which questions he or she needs to ask the supervisor and what progress has been made since the last meeting (if applicable) and noting these in the relevant sections of the sheet, effectively forming an agenda for the meeting.
3. The business of the meeting should be noted briefly as items in the relevant section of the sheet. There will be one sheet for each supervisory meeting and the actions on the student (and perhaps the supervisor) which should be carried out before the next meeting should be noted briefly in the relevant section of the sheet.
4. The Project Logbook is one of the deliverables of the final project and is an important record of the student's organisation and learning experience. The student should ensure that it is handed in at the end of the semester to their supervisor, with sheets dated and signed to show a consistent record of the supervisory meetings.

<p>Meeting 1</p> <p><u>Focus of Discussion:</u></p> <p>Review on the Project Proposal</p> <p>Current week's Activities</p> <p>Worked on writing my project proposal</p> <p>Next week's Activities</p> <p>Design the system architecture, including DBs and UI prototyping.</p> <p>Incorporating the changes on the project proposal.</p> <p><u>Supervisor's Comment On Student's Progress:</u></p> <ul style="list-style-type: none"> - Change structure & content of proposal. - Present design dgms. at next meeting. <p>Student's Signature : </p> <p>Supervisor's Signature : </p> <p>Date : 20/09/2024</p>
--

Meeting 2

Focus of Discussion:

Reviewing the changes made to project proposal and system architecture.

Current week's Activities

Incorporating proposal changes.

Designing system architecture.

Next week's Activities

Supervisor's Comment On Student's Progress:

- Tweak proposal to reflect changes discussed.
- Complete designs to include Level 1 & 2 DFDs, Network Diagrams.
- Proceed to begin coding.

Student's Signature : 

Supervisor's Signature : 

Date : 27/9/24.

Meeting 3

Focus of Discussion:

Review system design and documentation.

Code Review.

Current week's Activities

Training and Testing Models.


System design.


Coding.

Next week's Activities

Supervisor's Comment On Student's Progress:

- Add Functional & Non-functional require
- Tweak Aims & Objs. chapter, work on Gantt Chart to include milestones & dependencies.
- Complete training & export of facial

Student's Signature:  recognition models, test camera resolution, integrate work with other grp. members (90% or complete) & cloud hosted.

Supervisor's Signature:  Finalize prototype for demo. purposes.

Date: 31/10/2024

Meeting 4

Focus of Discussion:

Integrated Application by Other Team Members.

Proposal changes

Current week's Activities

Incorporating proposal changes.

Fixing code bugs.

Next week's Activities


Chapter 6 and 7; Draft.

Proposal changes.

Supervisor's Comment On Student's Progress:

- Logs, code punct. & non punct. rgmts.
- Gantt chart in landscape.
- Draft Implem + Testing & Eval. chapters

Student's Signature: 

Supervisor's Signature: 

Date: 12/11/2024

Meeting 5

Focus of Discussion:

Chapter 6 and 7 Draft.

Proposal changes.

Current week's Activities

Drafting of chapter 6 and 7.

Incorporating proposal changes.

Next week's Activities

Chapter 8.

Supervisor's Comment On Student's Progress:

- Tweak Implem + Testing & Eval chapter.
- Finalize Conclusion chapter.

Student's Signature : 

Supervisor's Signature : 

Date : 21/11/24.

Meeting 6

Focus of Discussion:

Project Report: Chapter 6, 7, & (Implementation)

Current week's Activities

Drafting chapter 6, 7 and 8.

Next week's Activities

Project Report, Documentation and Presentation.

Supervisor's Comment On Student's Progress:

- Change all bkg to white.
 - Add UI screenshots
 - Only significant details
 - 110 pages max.
 - No code in Chapter 7
 - Table explanation
- Achievements [x-3]

Student's Signature: 

Supervisor's Signature: 

Date: 27/11/2024

II. PLAGARISM REPORT