D3F.IO

DRIVER DROWSINESS DETECTION & FEEDBACK

The Pennsylvania State University
The College of Engineering
Senior Capstone Project

D3F.io Final Report

Prepared by

Lekh Shetty
Sreeman Etikyala
Shrey Sharma

ABSTRACT

Distracted driving continues to be a significant contributor to traffic accidents worldwide. In an effort to mitigate this problem, we developed a novel distracted driving feedback system in the form of an app that tracks eye and mouth movements to alert drivers when they exhibit signs of distraction. This report presents the development and evaluation of the app, which employs advanced computer vision algorithms to monitor drivers' eye and mouth movements in real time. The app specifically identifies instances of yawning and closed eyes, which can signify fatigue or inattention, and subsequently triggers an auditory alarm to alert the driver of their distraction.

The report outlines the methodology employed in designing and implementing the app, including data collection, algorithm development, and integration with smartphone platforms. It also discusses the results of a comprehensive evaluation of the system's effectiveness in reducing distracted driving behaviors. Our findings indicate a significant reduction in instances of yawning and closed eyes among drivers using the app, demonstrating the potential for this technology to enhance road safety. Further research is recommended to investigate the long-term impact of the app on driver behavior and its applicability across various demographics and driving conditions.

TABLE OF CONTENTS

INTRODUCTION

Although the actual number of crashes caused by drowsy driving is likely to be much higher, according to the National Highway Traffic Safety Administration (NHTSA) (Rivelli, 2022), drowsy driving is a contributing factor in an estimated 100,000 police-reported crashes each year in the United States, resulting in approximately 1,550 deaths, 71,000 injuries, and $12.5 billion in monetary losses. The actual number of crashes caused by drowsy driving is difficult to determine as fatigue is often difficult to detect and underreported; however, Artificial Intelligence has the potential to revolutionize the industry by providing vehicles with the ability to detect driver drowsiness and improve road safety. It can be used to monitor driver behavior, analyze facial expressions and eye movements, and detect signs of fatigue or distraction in real time. This technology, deployed through Streamlit, can alert drivers through the use of an alarm to remind them to stay focused on the road, thereby reducing the risk of accidents caused by drowsy driving. The application of AI in this area has the potential to save lives and prevent road accidents, making driving safer and more efficient for everyone.

To mitigate the risks associated with drowsy and distracted driving, feedback systems for drivers have been developed to alert drivers and prevent accidents. These systems use various facial sensors to monitor the driver's behavior and provide real-time feedback to help them stay alert and focused on the road. The proposed system uses machine learning algorithms to analyze driver behavior, including facial expressions, eye movements, and physiological signals, to detect signs of drowsiness and distraction. Once detected, the system provides real-time alerts to the driver, such as audible alarms or vibrations, to increase their attention and focus on the road.

LITERATURE SURVEY

Literature Review

Artificial Intelligence already plays an important role in promoting road safety both in cars and road cameras; such as adaptive cruise control, lane departure warning, and collision avoidance systems improve road safety by providing real-time warnings and taking corrective actions when necessary. AI algorithms can process real-time data from vehicles to quickly respond to road incidents and other emergencies.

Although the actual number of crashes caused by drowsy driving is likely to be much higher, according to the National Highway Traffic Safety Administration (NHTSA) (Rivelli, 2022), drowsy driving is a contributing factor in an estimated 100,000 police-reported crashes each year in the United States, resulting in approximately 1,550 deaths, 71,000 injuries, and $12.5 billion in monetary losses. The actual number of crashes caused by drowsy driving is difficult to determine as fatigue is often difficult to detect and underreported; however, Artificial Intelligence has the potential to revolutionize the industry by providing vehicles with the ability to detect driver drowsiness and improve road safety. It can be used to monitor driver behavior, analyze facial expressions and eye movements, and detect signs of fatigue or distraction in real time. This technology, deployed through Streamlit, can alert drivers through the use of an alarm to remind them to stay focused on the road, thereby reducing the risk of accidents caused by drowsy driving. The application of AI in this area has the potential to save lives and prevent road accidents, making driving safer and more efficient for everyone.

Assessment of Available Solutions and Techniques

The main approach used across a multitude of projects for drowsiness detection is to apply computer vision to track the face and then employ AI/ML algorithms to analyze and detect drowsiness. The method proposed by Phan et al. (2021) to combat drowsy driving by using deep learning techniques with two adaptive neural networks, based on MobileNet-V2 and ResNet-50V2, which would analyze the video of the driver's faces and detect the driver's activities in each frame to learn all features automatically. This approach can achieve an accuracy as high as 97% (Phan et al., 2021).

The approach proposed by Singh (2022) uses the Mediapipe API to make a drowsiness detector. Mediapipe provides an ML-based solution for producing a face mesh. Singh's (2022) solution uses the Mediapipe ML's API to detect eye landmarks, then applies the Eye Aspect Ratio (EAR) technique to detect drowsiness. His solution (Singh, 2022) then uses the Streamlit web-application development tool to stream the real-time video/audio feed and apply the detector on the feed.

Pros and Cons

During our search for available solutions we came across a lot of projects that have a fully functioning drowsiness detector, but these detectors lack a user-friendly front-end application. The solution proposed by Phan et al. (2021) has very high accuracy, making it a very strong drowsiness detection tool. However, it does not have a reliable front-end application attached to it. In addition, the project also lacks a real-time detection aspect.

Singh's (2022) approach solves the real-time aspect through the use of a Streamlit tool, which also doubles up as a working front-end. However, the research conducted by Phan et. al (2021) suggests that the ANN model they created outperforms the EAR technique. The Approach proposed by Singh (2022) can thus be improved through the use of ML.

Approach

The Distracted Driver Detection project is designed to address the growing issue of driver distraction and promote road safety. The project uses cutting-edge technologies, such as computer vision and machine learning, to analyze real-time video and audio feeds from a vehicle and detect instances of drowsiness or distraction. A Streamlit front-end facilitates real- time video/audio feed analysis and also provides an intuitive and user-friendly interface for the end-user, while the Mediapipe API is used to detect and track faces in the video feed. The project leverages a state-of-the-art ML/AI algorithm, like the one proposed by Phan et. al (2021), to analyze the video feed and provide real-time feedback to the driver if drowsiness or distraction is detected.

The project team assumes full access to the necessary APIs and a stable development environment. A potential risk during development is the possibility of the algorithm producing false positives or negatives. The project faces several constraints, a virtual-only workspace, and a dependence on data from external sources.

The Distracted Driver Detection project is poised to benefit society by reducing the number of accidents caused by driver distraction. The solution is aimed at promoting road safety and preventing distracted driving, thereby improving the safety of drivers, passengers, and other road users. The project has the potential to be adopted by vehicle manufacturers, insurance companies, and governments to promote road safety and prevent distracted driving.

REQUIREMENT SPECIFICATIONS

Stakeholder Analysis

Table 1: Stakeholders

Stakeholder	**Power**	**Interest**	**Legitimacy**
Production Team	High	High	High
Truck Drivers	Low	High	Low
Insurance Companies	Low	High	High
Corporations that employ truck drivers for deliveries	Low	High	High
General Public	Low	High	Low

The production team is the number one stakeholder with high power, high interest, and high legitimacy which overall makes them the highest priority. Corporations that hire truck deliveries to carry their product across the country, and insurance companies have low power but high interest in the product and real legitimacy to invest and utilize the product. Whilst the general public has low power and legitimacy for this product, their high interest is what can get the product to be picked up by businesses to make use of.

Market Analysis

With strong demand for safety and stricter road safety standards, the use of artificial intelligence in sleepiness driving detection is a business that is expanding quickly. AI-based solutions leverage tools like computer vision, machine learning, and deep learning algorithms to identify drowsiness in real time and warn the driver. This is particularly important for the commercial vehicle sector, where driver weariness is a big issue.

Due to reasons like rising drowsy driving-related traffic accidents, government attempts to promote road safety, and the expansion of the auto sector, the market is predicted to rise. Market expansion will also be fueled by rising interest in autonomous vehicles and rising demand for sophisticated driver support systems in automobiles.

The market must also overcome obstacles like high implementation costs, worries about data security and privacy, and a lack of sleepiness detection system standards. Nevertheless, the market for AI in drowsy driving detection is anticipated to increase greatly in the future years due to the continuing advancement of AI technology and its incorporation into a variety of industries.

Businesses that hire trucks to be driven across the country, delivering their products, can benefit from the use of AI. Ensuring that their drivers are getting to their destination as safely and efficiently as possible. Insurance companies that promote safer driving for better rates can benefit from this product too. Promoting their customers to use the website to provide real-time diagnostics whilst driving as well as uphold safe driving practices.

Constraints

We have a few manageable concerns relating to the final development of our application and having it work efficiently. These constraints include Streamlit imposing restrictions on our service due to an overload on their server. This might hinder the ML model from providing quick alerts to distracted drivers. Additionally, the ML model is derived from external sources and it might not be reliable enough. The model may pose issues when it is updated to account for the data logging aspect. We also have to take into account getting consent from the driver of the vehicle to be recorded throughout their car journeys. It's important for us to get this permission from our users to help us avoid breaking any personal data infringement laws. Finally, our April 18th deadline also poses a constraint for us as that only leaves us with nine weeks to work on this project. It's vital for the schedules of all 3 of us to align properly for us to successfully work on this project.

Assumptions

In preparation for this project, the team assumes that smartphones will have access to fast and stable internet and modern processors capable of providing quick feedback. We also make the assumption of users' consent to be recorded while they are driving as we do not want to break any personal data infringement and privacy laws. Assumption of Streamlit not being overloaded and imposing restrictions on our service when trying to provide feedback to the user at the end of their journey. We are also assuming that the model is running without any issues and not giving out any false alerts.

Risks

There is a risk of false positives and false negatives in detecting drowsiness. False positives can lead to unnecessary alerts and can cause distractions while driving, and false negatives can result in not detecting actual drowsiness, leading to a higher risk of accidents. This however can be addressed by adapting the thresholds to the facial features of the driver.

The use of the drowsiness detection system requires the collection and analysis of personal data, such as facial expressions and eye movements, which can raise privacy concerns. If the data is not stored securely, it can be misused and lead to identity theft or other cybercrimes.

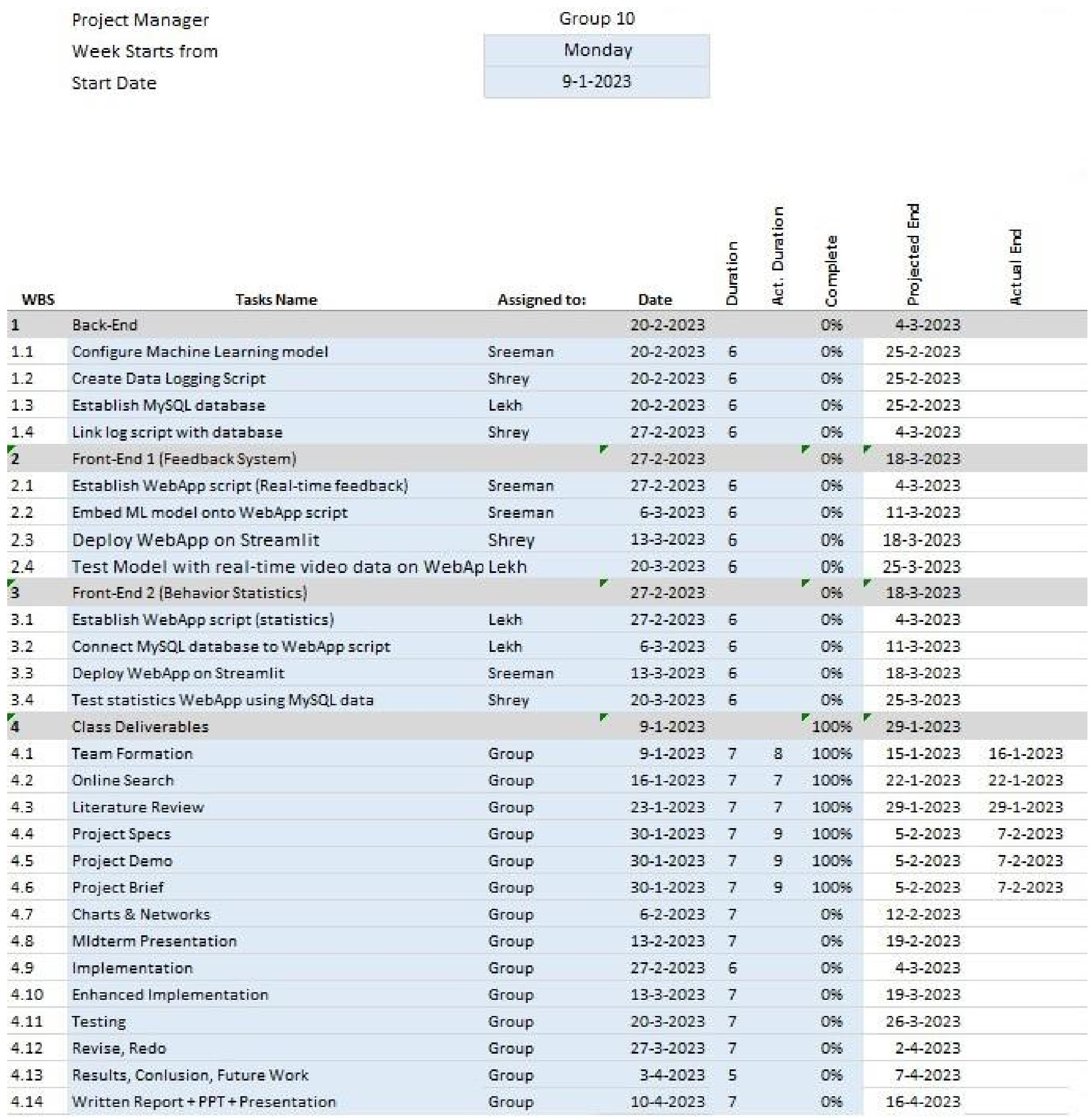
Another risk associated with the applications is the ambiguity in detecting and reading whether the driver is wearing accessories or apparel that cover the face, such as masks or shades. This problem stems from the facial detection logic of the mediapipe library, which the team will no be able to update as it is beyond the team’s capabilities. Another risk to consider is how the feedback system reacts to passenger faces being seen in the background.

SYSTEM DEVELOPMENT

System Planning

With our general idea in mind, we set out to plan our development efforts. Looking at examples of work breakdown structures helped us conceptualize and break down the required tasks into a few overall categories. We knew the system would be a web application, so it would require a front-end user interface and a back-end MySQL database. After brief consideration, we created the first breakdown of our work, shown below in the *First Project Gantt Chart.* The four categories of work were front-end 1, front-end 2, and back-end. The subtasks in these categories shifted over time as the system came to fruition, but the four categories remained the same throughout. This is shown in the final iteration of the same chart.

Table 2: Initial Gantt Chart

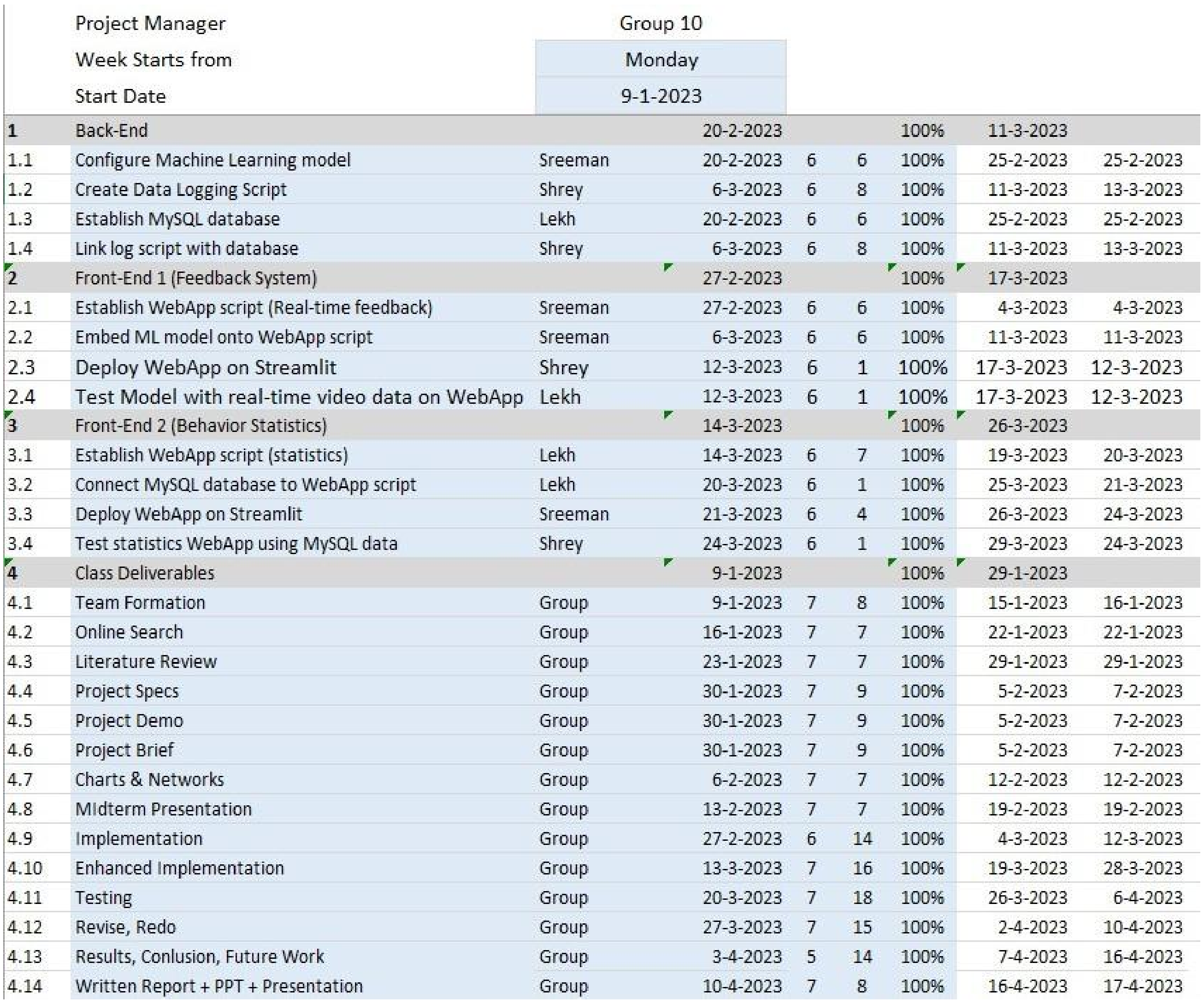


Project Manager	Group 10
Week Starts from	Monday
Start Date	9-1-2023

WBS	Tasks Name	Assigned to:	Date	Duration	Act. Duration	Complete	Projected End	Actual End
1	Back-End		20-2-2023			0%	4-3-2023	
1.1	Configure Machine Learning model	Sreeman	20-2-2023	6		0%	25-2-2023	
1.2	Create Data Logging Script	Shrey	20-2-2023	6		0%	25-2-2023	
1.3	Establish MySQL database	Lekh	20-2-2023	6		0%	25-2-2023	
1.4	Link log script with database	Shrey	27-2-2023	6		0%	4-3-2023	
2	Front-End 1 (Feedback System)		27-2-2023			0%	18-3-2023	
2.1	Establish WebApp script (Real-time feedback)	Sreeman	27-2-2023	6		0%	4-3-2023	
2.2	Embed ML model onto WebApp script	Sreeman	6-3-2023	6		0%	11-3-2023	
2.3	Deploy WebApp on Streamlit	Shrey	13-3-2023	6		0%	18-3-2023	
2.4	Test Model with real-time video data on WebAp	Lekh	20-3-2023	6		0%	25-3-2023	
3	Front-End 2 (Behavior Statistics)		27-2-2023			0%	18-3-2023	
3.1	Establish WebApp script (statistics)	Lekh	27-2-2023	6		0%	4-3-2023	
3.2	Connect MySQL database to WebApp script	Lekh	6-3-2023	6		0%	11-3-2023	
3.3	Deploy WebApp on Streamlit	Sreeman	13-3-2023	6		0%	18-3-2023	
3.4	Test statistics WebApp using MySQL data	Shrey	20-3-2023	6		0%	25-3-2023	
4	Class Deliverables		9-1-2023			100%	29-1-2023	
4.1	Team Formation	Group	9-1-2023	7	8	100%	15-1-2023	16-1-2023
4.2	Online Search	Group	16-1-2023	7	7	100%	22-1-2023	22-1-2023
4.3	Literature Review	Group	23-1-2023	7	7	100%	29-1-2023	29-1-2023
4.4	Project Specs	Group	30-1-2023	7	9	100%	5-2-2023	7-2-2023
4.5	Project Demo	Group	30-1-2023	7	9	100%	5-2-2023	7-2-2023
4.6	Project Brief	Group	30-1-2023	7	9	100%	5-2-2023	7-2-2023
4.7	Charts & Networks	Group	6-2-2023	7		0%	12-2-2023	
4.8	MIdterm Presentation	Group	13-2-2023	7		0%	19-2-2023	
4.9	Implementation	Group	27-2-2023	6		0%	4-3-2023	
4.10	Enhanced Implementation	Group	13-3-2023	7		0%	19-3-2023	
4.11	Testing	Group	20-3-2023	7		0%	26-3-2023	
4.12	Revise, Redo	Group	27-3-2023	7		0%	2-4-2023	
4.13	Results, Conlusion, Future Work	Group	3-4-2023	5		0%	7-4-2023	
4.14	Written Report + PPT + Presentation	Group	10-4-2023	7		0%	16-4-2023	

The Gantt chart before project development

Table 3: Final Project Gantt Chart

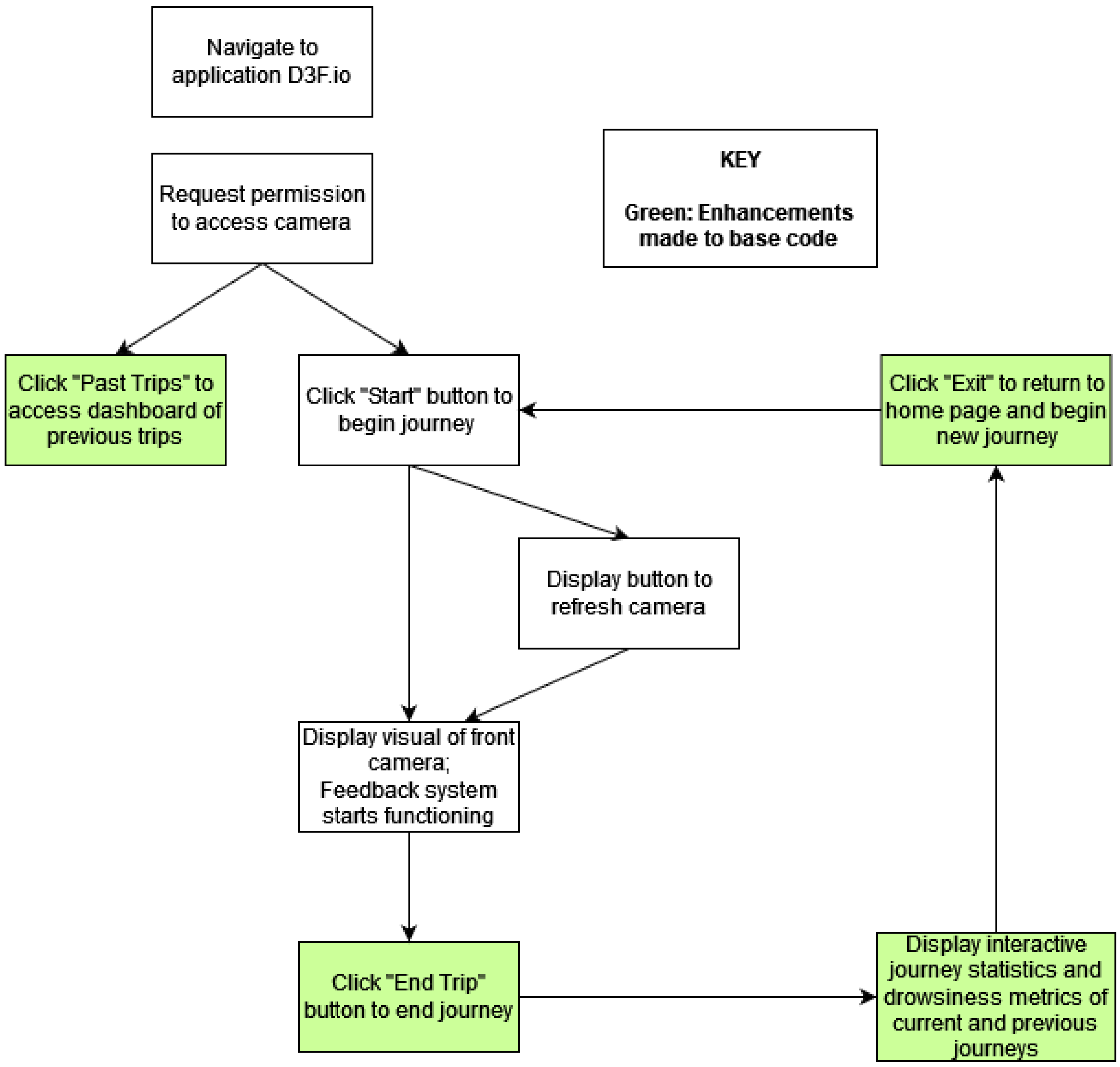


	Project Manager		Group 10					
	Week Starts from		Monday					
	Start Date		9-1-2023					
1	Back-End		20-2-2023			100%	11-3-2023	
1.1	Configure Machine Learning model	Sreeman	20-2-2023	6	6	100%	25-2-2023	25-2-2023
1.2	Create Data Logging Script	Shrey	6-3-2023	6	8	100%	11-3-2023	13-3-2023
1.3	Establish MySQL database	Lekh	20-2-2023	6	6	100%	25-2-2023	25-2-2023
1.4	Link log script with database	Shrey	6-3-2023	6	8	100%	11-3-2023	13-3-2023
2	Front-End 1 (Feedback System)		27-2-2023			100%	17-3-2023	
2.1	Establish WebApp script (Real-time feedback)	Sreeman	27-2-2023	6	6	100%	4-3-2023	4-3-2023
2.2	Embed ML model onto WebApp script	Sreeman	6-3-2023	6	6	100%	11-3-2023	11-3-2023
2.3	Deploy WebApp on Streamlit	Shrey	12-3-2023	6	1	100%	17-3-2023	12-3-2023
2.4	Test Model with real-time video data on WebApp	Lekh	12-3-2023	6	1	100%	17-3-2023	12-3-2023
3	Front-End 2 (Behavior Statistics)		14-3-2023			100%	26-3-2023	
3.1	Establish WebApp script (statistics)	Lekh	14-3-2023	6	7	100%	19-3-2023	20-3-2023
3.2	Connect MySQL database to WebApp script	Lekh	20-3-2023	6	1	100%	25-3-2023	21-3-2023
3.3	Deploy WebApp on Streamlit	Sreeman	21-3-2023	6	4	100%	26-3-2023	24-3-2023
3.4	Test statistics WebApp using MySQL data	Shrey	24-3-2023	6	1	100%	29-3-2023	24-3-2023
4	Class Deliverables		9-1-2023			100%	29-1-2023	
4.1	Team Formation	Group	9-1-2023	7	8	100%	15-1-2023	16-1-2023
4.2	Online Search	Group	16-1-2023	7	7	100%	22-1-2023	22-1-2023
4.3	Literature Review	Group	23-1-2023	7	7	100%	29-1-2023	29-1-2023
4.4	Project Specs	Group	30-1-2023	7	9	100%	5-2-2023	7-2-2023
4.5	Project Demo	Group	30-1-2023	7	9	100%	5-2-2023	7-2-2023
4.6	Project Brief	Group	30-1-2023	7	9	100%	5-2-2023	7-2-2023
4.7	Charts & Networks	Group	6-2-2023	7	7	100%	12-2-2023	12-2-2023
4.8	MIdterm Presentation	Group	13-2-2023	7	7	100%	19-2-2023	19-2-2023
4.9	Implementation	Group	27-2-2023	6	14	100%	4-3-2023	12-3-2023
4.10	Enhanced Implementation	Group	13-3-2023	7	16	100%	19-3-2023	28-3-2023
4.11	Testing	Group	20-3-2023	7	18	100%	26-3-2023	6-4-2023
4.12	Revise, Redo	Group	27-3-2023	7	15	100%	2-4-2023	10-4-2023
4.13	Results, Conlusion, Future Work	Group	3-4-2023	5	14	100%	7-4-2023	16-4-2023
4.14	Written Report + PPT + Presentation	Group	10-4-2023	7	8	100%	16-4-2023	17-4-2023

The final Gantt chart after project completion

Principle Operation

Figure 1: Operational Flow


Navigate to application D3F.io
Request permission to access camera
KEY
Green: Enhancements made to base code
Click "Past Trips" to access dashboard of previous trips
Click "Start" button to begin journey
Click "Exit" to return to home page and begin new journey
Display button to refresh camera
Display visual of front camera; Feedback system starts functioning
Click "End Trip" button to end journey
Display interactive journey statistics and drowsiness metrics of current and previous journeys

The operational flow chart displays the steps the user takes to use the application.

The principle operation begins when the user navigates to d3f.io on their web browser. As shown in Figure one, after entering the site, they have one of two options: to view an interactive dashboard of a previous trip or to start a new trip and launch the real-time feedback system.

When the user clicks the start trip button, the user will be able to access the real-time feedback system. Here, after the user has granted permission for the camera, the user will be able to see the web player display their face with landmarks on their eyes and mouth. These landmarks are used to track the user's behavior like blinks and yawns through Eye-Aspect-Ratio (EAR) and Mouth-Aspect-Ratio (MAR). The web player will display these ratios and a timer to count how long the user's eyes have been shut. Once the timer reaches a threshold number of seconds, the feedback system will ring an alarm to alert the user. This alarm has both a visual and audible aspect. The web player will display flashing text and also play audio of a person saying "Wake Up." While using the app the web player also allows the user to stop using the camera or switch to a different camera.

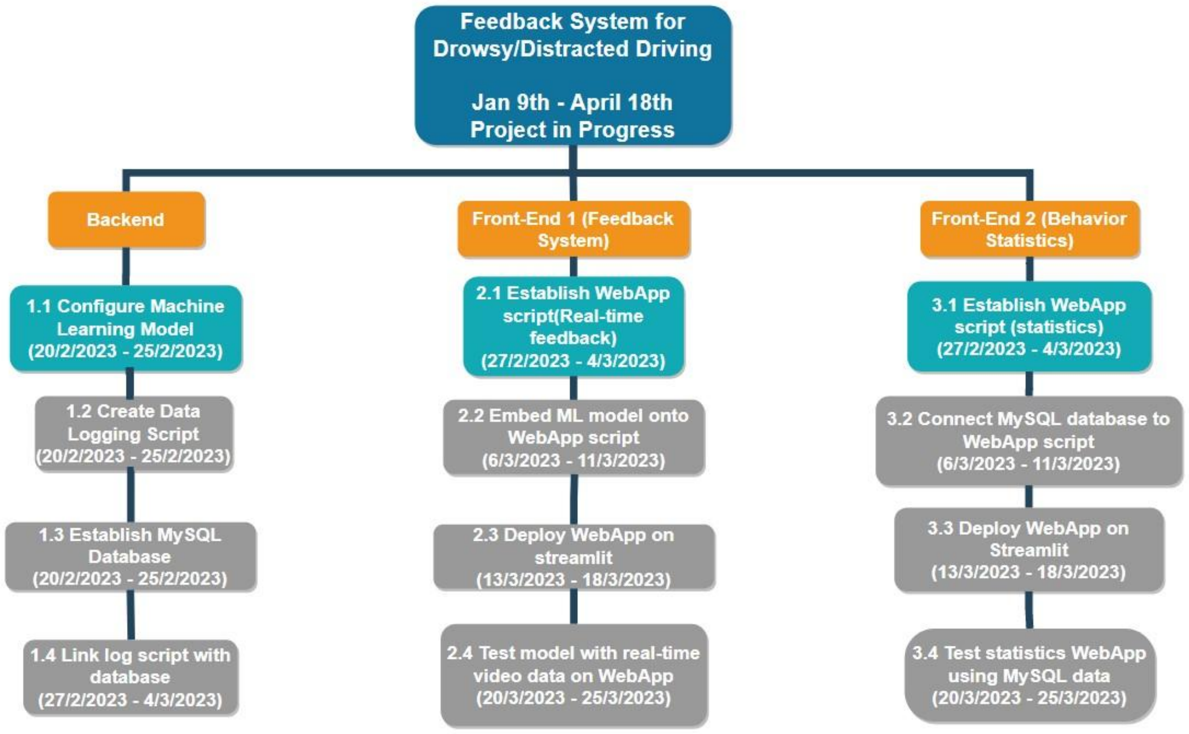
Once the user chooses to end the trip, they may click the end trip button, which will take them to the trip information page. This page is also accessible from the "view previous trips" button on the main page. This page contains a drop-down menu of all the logged trips. Once the user selects a trip from the menu, an interactive dashboard is generated. This dashboard has graphs pertaining to the EAR, MAR, and Alarm behavior over the duration of the trip. The user can interact with these graphs by zooming into specific timeframes, panning around the timeframes, and also downloading the graphs. The dashboard also has information like the number of times the user yawned, blinked, and triggered the alarm. This page also allows the user to delete any trip records that are either empty or that the user may no longer find to be useful. From here the user can return to the main page by using the return home button.

System Design

For both front-end components, the team decided to use Streamlit for development. The team was not the strongest when it came to web application development, so Streamlit was the best option as it was Python-friendly, and easier to use and deploy than other traditional tools like Flask or Django. Streamlit also had great support for data visualization, so it was a clear choice for creating the graphs in front-end 2's dashboard. For the back-end, MySQL was a clear choice. A MySQL could comfortably handle the amount and the speed at which data was produced by each frame of the live video. Also, Streamlit's support for MySQL ensured the smooth creation of the dashboard using the data from the MySQL database.

The steps the team followed to develop each part of the application are shown in *Work Breakdown Structure* below.

Figure 2: Work Breakdown Structure


Feedback System for Drowsy/Distracted Driving
Jan 9th - April 18th Project in Progress
Backend
Front-End 1 (Feedback System)
Front-End 2 (Behavior Statistics)
1.1 Configure Machine Learning Model (20/2/2023 - 25/2/2023)
1.2 Create Data Logging Script (20/2/2023 - 25/2/2023)
1.3 Establish MySQL Database (20/2/2023 - 25/2/2023)
1.4 Link log script with database (27/2/2023 - 4/3/2023)
2.1 Establish WebApp script(Real-time feedback) (27/2/2023 - 4/3/2023)
2.2 Embed ML model onto WebApp script (6/3/2023 - 11/3/2023)
2.3 Deploy WebApp on streamlit (13/3/2023 - 18/3/2023)
2.4 Test model with real-time video data on WebApp (20/3/2023 - 25/3/2023)
3.1 Establish WebApp script (statistics) (27/2/2023 - 4/3/2023)
3.2 Connect MySQL database to WebApp script (6/3/2023 - 11/3/2023)
3.3 Deploy WebApp on Streamlit (13/3/2023 - 18/3/2023)
3.4 Test statistics WebApp using MySQL data (20/3/2023 - 25/3/2023)

The Work Breakdown Structure reveals how the project is broken down by date and section.

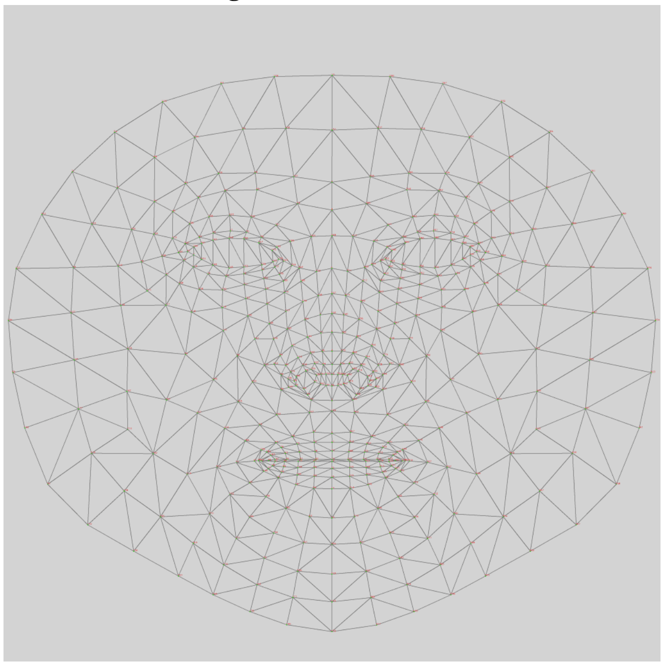
The entire application is built using three python scripts, namely drowsy_detection.py, audio_handler.py, and streamlit_app.py.

drowsy_detection.py:

This script handles the logic behind the visual aspect of the feedback system. It uses the VideoFrameHandler class to accept new frames from the live stream and analyzes them by using function like calculate_ear_mar(), and process().

The process() function is where the frame is analyzed. It accepts the current frame and a thresholds dictionary. It first checks the frame for a face, and when a face is detected, it plots landmarks around the eyes and the mouth by using the plot_landmarks() function. This function uses Mediapipe's face mesh to determine the landmarks' location. It also uses the calculate_ear_mar() function to get the Eye Aspect Ratio (EAR) and the Mouth Aspect Ratio (MAR) for the current frame.

Figure 3: Face Model



Mediapipe's canonical face model, where each vertex is a number representing a specific face landmark

The process() function uses the thresholds in the dictionary for EAR, MAR and Wait Time to check if eyes are shut, yawn is triggered, etc. By checking against these thresholds the function is able to selectively update several variables like yawn_counter, alarm_counter, play_alarm, drwsy_time, etc. Some of these variables are stored in the row_dict dictionary so that they can be used later to make the interactive dashboard. The update to the play_alarm variable allows the feedback system to determine if the system should ring the alarm. If the eyes are determined to be shut for more than the "Wait_Time" threshold, then play_alarm is set to true, thus allowing the system to sound the alarm using the alarm_handling.py script.

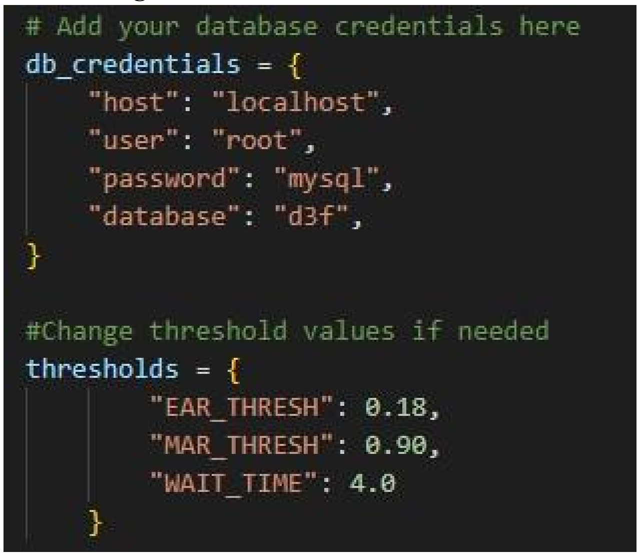
audio_handling.py:

The audio_handling.py script handles the logic behind the audio aspect of the feedback system. The file is structures similar to the drowsy_detection.py file, comprising of the AudioFrameHandler class with a process() function. This file is used by the feedback system to play the alarm to alert the driver. Here the process() function accepts the current audio frame and a boolean variable play_alarm. This function checks the audio frame based on the play_alarm variable and decides whether to play the alarm or not.

streamlit_app.py:

The streamlit_app.py file is the main file that both creates the front-end and also connects it to the back-end. This page comprises of several functions, out of which the ones named main(), page2(), and page3() are the 3 main pages of the web application. The script also contains two important dictionary at the start. The first is the db_credentials dictionary that contains information regarding connection to the backend MySQL database. The other dictionary is thresholds. This contains EAR and MAR thresholds along with Wait_Time. These thresholds are used by the feedback system to alert the driver.

Figure 4: Initialization Dictionaries



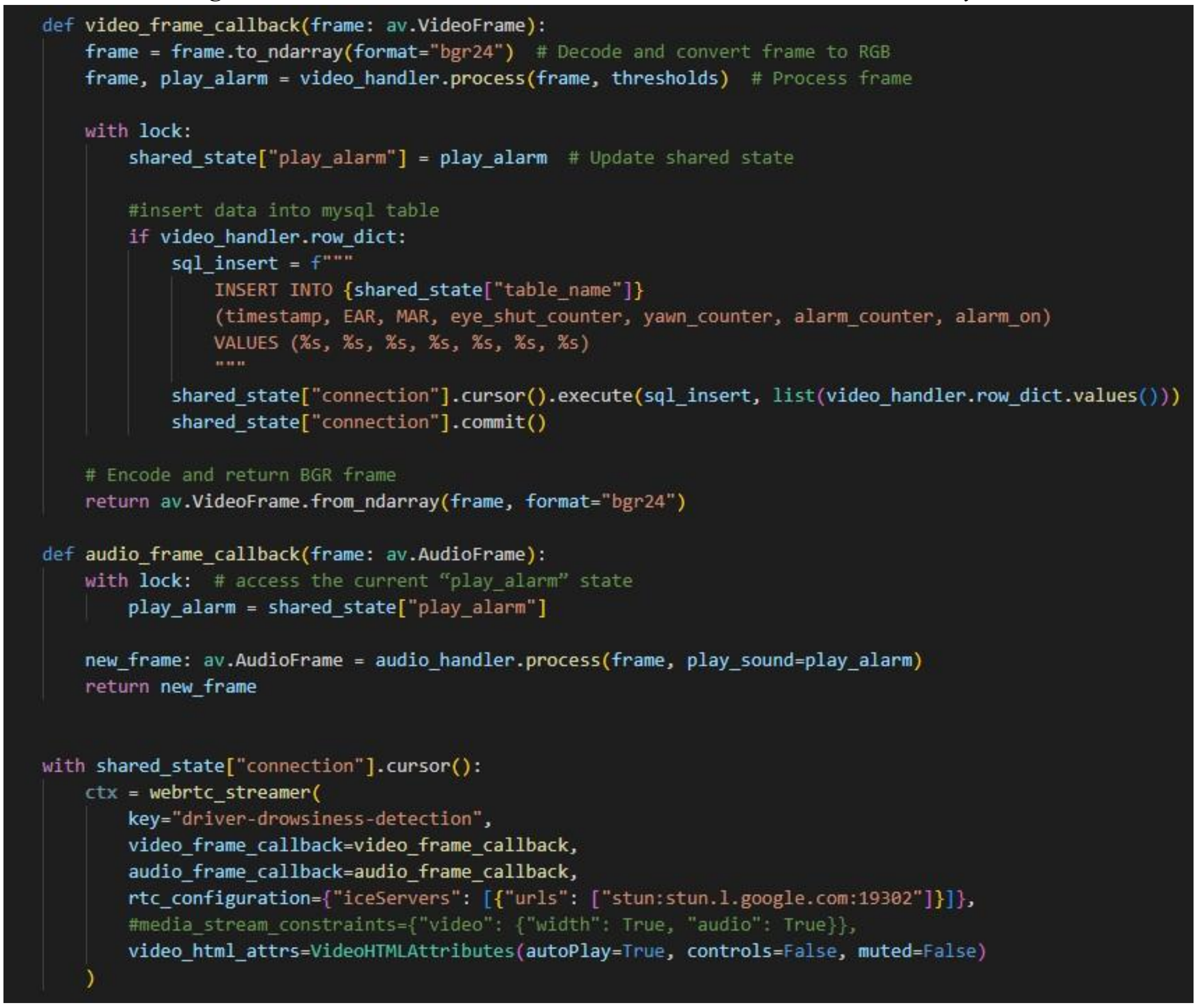
```python
# Add your database credentials here
db_credentials = {
    "host": "localhost",
    "user": "root",
    "password": "mysql",
    "database": "d3f",
}

#Change threshold values if needed
thresholds = {
        "EAR_THRESH": 0.18,
        "MAR_THRESH": 0.90,
        "WAIT_TIME": 4.0
    }
```

These python dictionaries contain details about connecting to the MySQL database and the thresholds used by the feedback system, respectively

The main() function houses code for the home page with two buttons, “start trip” and “view previous trips”, that each lead to the other two pages. This function is the first to be executed when the web app loads. It calls the create_database function that creates a new MySQL if it doesnt already exist. It also calls the create_table() function when the start_trip button is clicked to create a new MySQL table with an automatically generated name to collect data for the current trip.

Page2() contains code for the feedback system. Here the objects from the classes VideoFrameHandler() (from drowsy_detection.py) and AudioFrameHandler() (from audio_handling.py) are utilized in the video_frame_callback() and audio_frame_callback functions respectively. These two functions are executed for each frame of the live stream. The video_frame_callback() function executes the process() function (from drowsy_detection.py) to process each video frame. Here the row_dict dictionary that is created when process() is run is appended to a back-end MySQL table. The audio_frame_callback() function also executes the process() function (from audio_handling.py) to process each audio_frame. These two functions are repeatedly called in the webrtc_streamer function. This function is for the web player on the web application that accepts the video from the camera.

Figure 5: Video and Audio Frame Callback Functions and Web Player



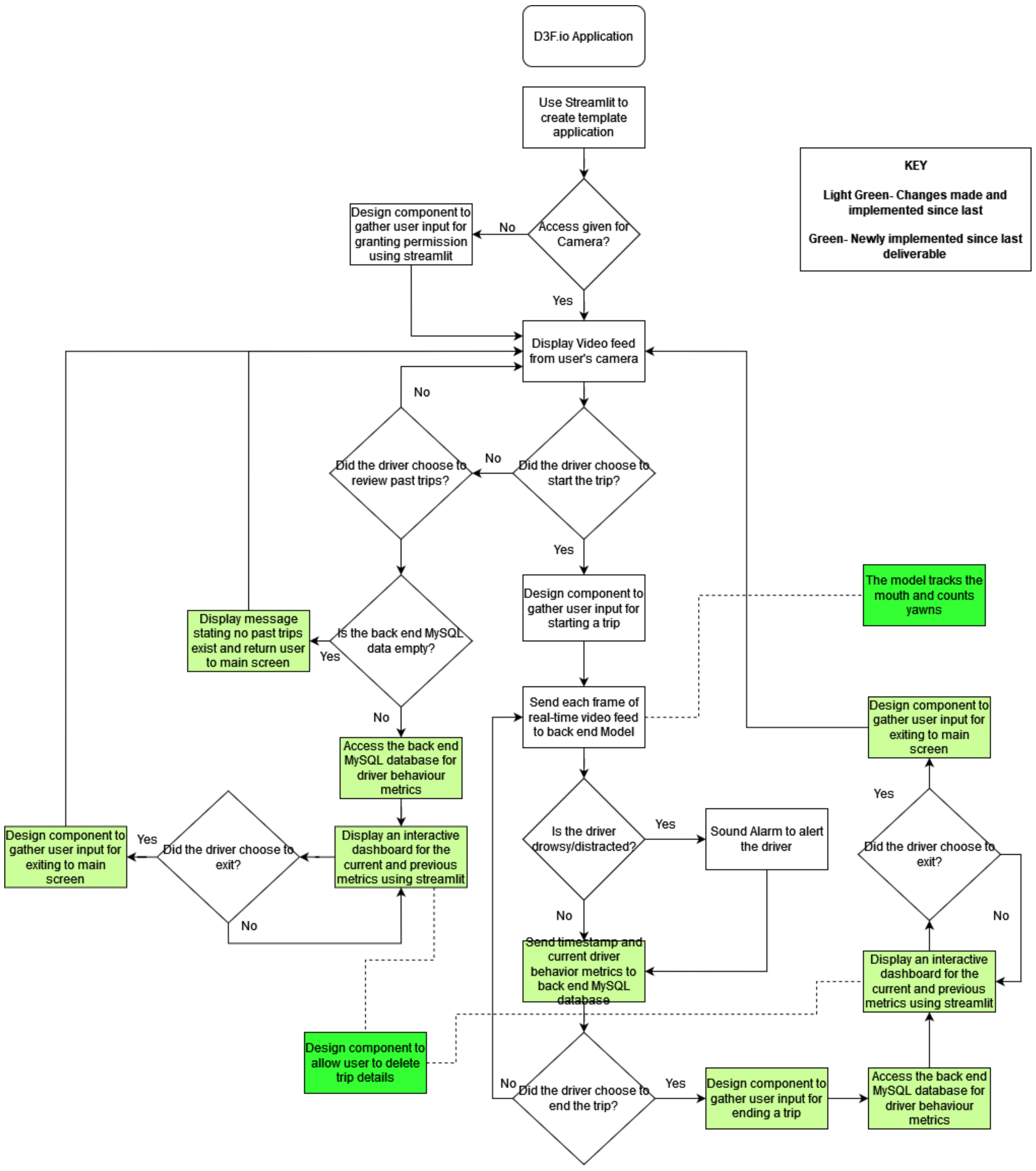
```python
def video_frame_callback(frame: av.VideoFrame):
    frame = frame.to_ndarray(format="bgr24")  # Decode and convert frame to RGB
    frame, play_alarm = video_handler.process(frame, thresholds)  # Process frame

    with lock:
        shared_state["play_alarm"] = play_alarm  # Update shared state

        #insert data into mysql table
        if video_handler.row_dict:
            sql_insert = f"""
                INSERT INTO {shared_state["table_name"]}
                (timestamp, EAR, MAR, eye_shut_counter, yawn_counter, alarm_counter, alarm_on)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
                """
            shared_state["connection"].cursor().execute(sql_insert, list(video_handler.row_dict.values()))
            shared_state["connection"].commit()

    # Encode and return BGR frame
    return av.VideoFrame.from_ndarray(frame, format="bgr24")

def audio_frame_callback(frame: av.AudioFrame):
    with lock:  # access the current “play_alarm” state
        play_alarm = shared_state["play_alarm"]

    new_frame: av.AudioFrame = audio_handler.process(frame, play_sound=play_alarm)
    return new_frame


with shared_state["connection"].cursor():
    ctx = webrtc_streamer(
        key="driver-drowsiness-detection",
        video_frame_callback=video_frame_callback,
        audio_frame_callback=audio_frame_callback,
        rtc_configuration={"iceServers": [{"urls": ["stun:stun.l.google.com:19302"]}]},
        #media_stream_constraints={"video": {"width": True, "audio": True}},
        video_html_attrs=VideoHTMLAttributes(autoPlay=True, controls=False, muted=False)
    )
```

These two callback functions are repeatedly called by the webrtc_streamer function for each frame collected by the web player from the camera

Page3() contains code for collecting data from the back-end MySQL database in order to create the interactive dashboard. This file has code a drop down menu that contains names of all the tables in the database where each table pertains to a specific trip. These names are obtained using the get_table_names() function. After a table is selected, data from that table is loaded into a pandas dataframe using the load_data_from_table() function. This dataframe is then used to create EAR, MAR, and Alarm behavior graphs through the create_dashboard() function. This page also has a “delete trip details” button that uses the delete_table() function to drop the specified table from the back-end database.

All these scripts together allow the d3f system to function. The final project flow made possible using the python files is detailed below.

Figure 6: Design Flow Chart

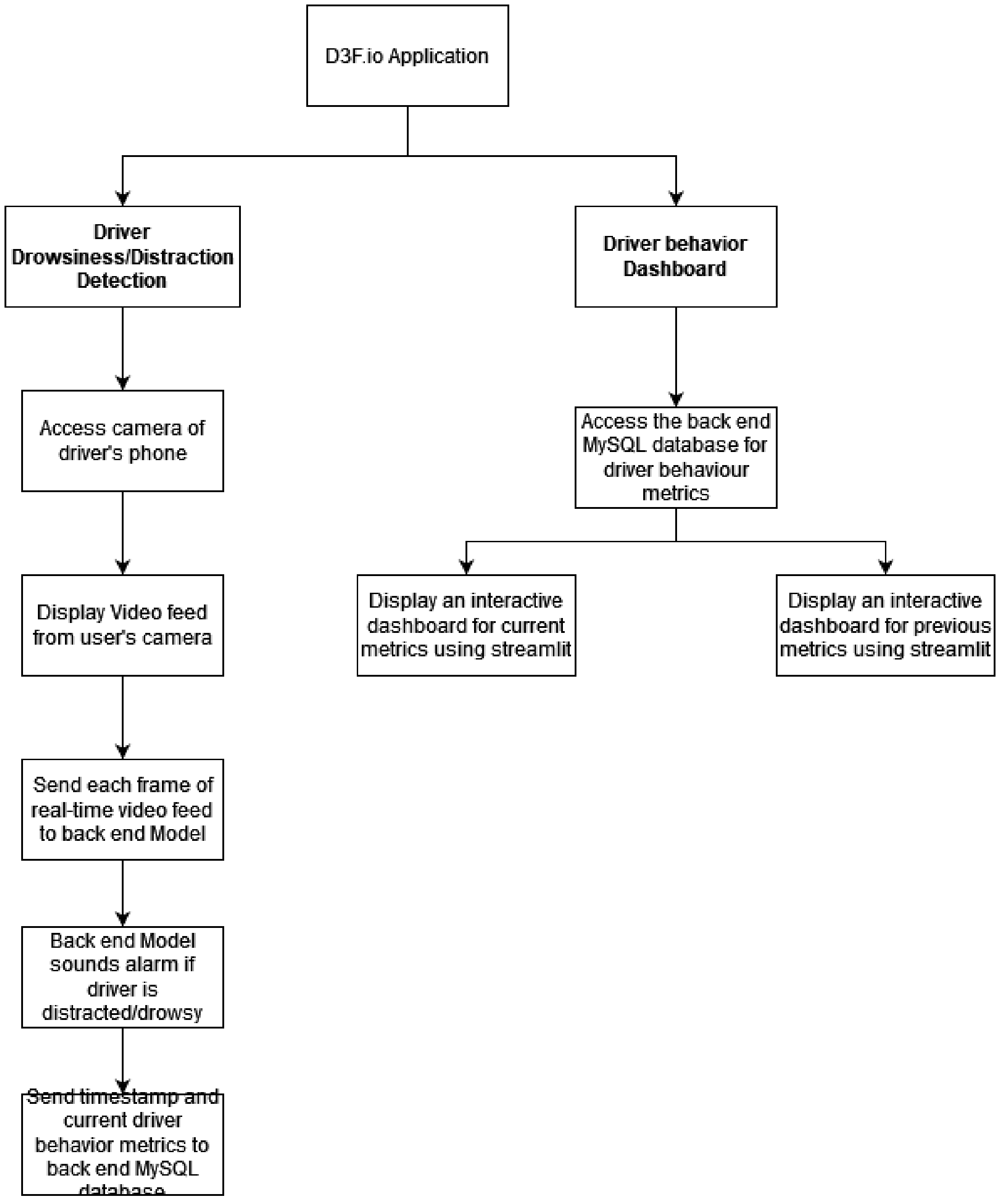

D3F.io Application
Use Streamlit to create template application
Access given for Camera?
No
Design component to gather user input for granting permission using streamlit
Yes
KEY
Light Green- Changes made and implemented since last
Green- Newly implemented since last deliverable
Display Video feed from user's camera
No
Did the driver choose to review past trips?
No
Did the driver choose to start the trip?
Yes
Design component to gather user input for starting a trip
The model tracks the mouth and counts yawns
Display message stating no past trips exist and return user to main screen
Yes
Is the back end MySQL data empty?
No
Send each frame of real-time video feed to back end Model
Design component to gather user input for exiting to main screen
Access the back end MySQL database for driver behaviour metrics
Yes
Design component to gather user input for exiting to main screen
Yes
Did the driver choose to exit?
Display an interactive dashboard for the current and previous metrics using streamlit
Is the driver drowsy/distracted?
Yes
Sound Alarm to alert the driver
Did the driver choose to exit?
No
No
No
Send timestamp and current driver behavior metrics to back end MySQL database
Display an interactive dashboard for the current and previous metrics using streamlit
Design component to allow user to delete trip details
No
Did the driver choose to end the trip?
Yes
Design component to gather user input for ending a trip
Access the back end MySQL database for driver behaviour metrics

The design flow chart shows how the application was designed from start to finish.

Functional Decomposition

The breakdown of the functionality of the application is actually fairly simple. With only two use cases, the functional decomposition of the project is not convoluted or complex, but rather a tree with two branches as seen below.

Figure 7: Functional Decomposition


D3F.io Application
Driver Drowsiness/Distraction Detection
Driver behavior Dashboard
Access camera of driver's phone
Access the back end MySQL database for driver behaviour metrics
Display Video feed from user's camera
Display an interactive dashboard for current metrics using streamlit
Display an interactive dashboard for previous metrics using streamlit
Send each frame of real-time video feed to back end Model
Back end Model sounds alarm if driver is distracted/drowsy
Send timestamp and current driver behavior metrics to back end MySQL database

The Functional Decomposition Diagram shows how the code is broken down into a detailed level of design.

Once the application is started, regardless of use case it will go through the same initial processes. The application will always instantiate the main page that allows the user to navigate to either the feedback system or the interactive dashboard. In the case of selecting one or the other, it will go along the branch until it is complete, and will then allow the user to choose between either function again.

If the user decides to use the feedback system, the web player is displayed that will show the live footage being captures using the user’s camera. When the driver starts the trip, the feedback system will process each frame of the live video, and sound the alarm if the driver is determined to be drowsy. The application also sends important statistics collected in each frame like EAR, MAR, etc. to the back-end MySQL database for later use.

If the user decides to end the current trip, or chooses to view previous trips from the home page, they will be navigated to the dashboard page where they have to choose a completed trip from a drop down menu. Data pertaining to the user selected trip will be queried from the back-end database and then be displayed in the form of an interactive dashboard.

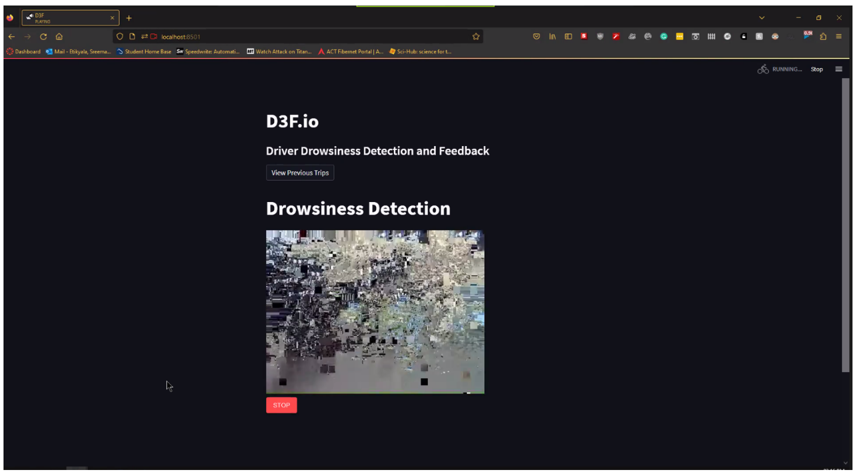
Testing

Case 1: Optimization

The initial project was not performing well on older and weaker hardware, which posed a significant challenge. The goal was to make the application work seamlessly on phones, but this objective would have been compromised if the app couldn't provide real-time feedback. To address this issue, the team focused on optimizing the code so that it could run smoothly on the weakest and oldest laptop available to the group. The aim was to ensure that the application could function effectively on phones while also providing real-time feedback, which was critical to the project's success.

The app initially had performance issues, with frequent slowdowns, crashes, and excessive memory usage. One of the factors contributing to this was the insertion of more than 1 row per frame into a pandas dataframe, resulting in a bloated table with around 17 redundant rows of data per frame. This inefficiency added to the app's performance problems, contributing to the overall suboptimal user experience.

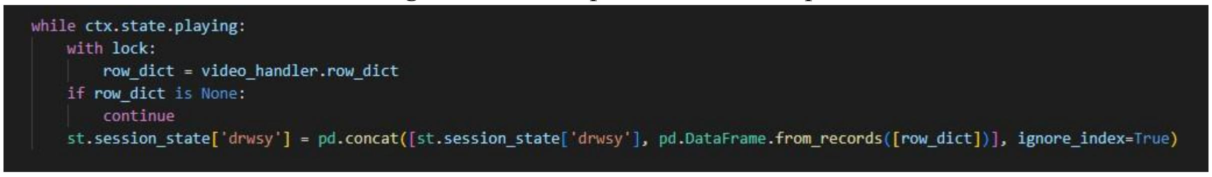
Figure 8: Unoptimized Feedback System



The unoptimized system would routinely freeze up and crash

The possible reasons for the lag and crashes in the code could be due to the way it was initially designed to log data like eye-aspect-ratio, alarm_counter, timestamp, etc. into a pandas dataframe every time a frame from the camera was being processed. The insertion of rows was managed by a while loop, which looped faster than the speed at which the frames were being processed, leading to redundant rows. Another possible cause of latency could be the action of adding rows to the pandas dataframe. The data frame was only sent to the MySQL backend when the user chose to end the trip.

Figure 9: The unoptimized while loop

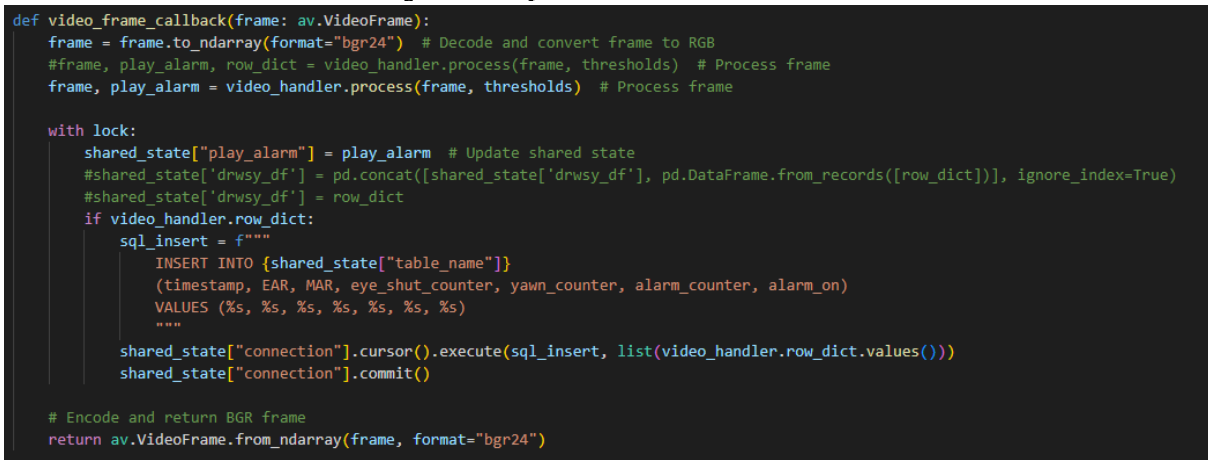


```python
while ctx.state.playing:
    with lock:
        row_dict = video_handler.row_dict
    if row_dict is None:
        continue
    st.session_state['drwsy'] = pd.concat([st.session_state['drwsy'], pd.DataFrame.from_records([row_dict])], ignore_index=True)
```

The while loop would execute multiple times per frame causing slow-downs and data redundancy

The first step in improving the performance of the app was to replace the pandas dataframe with a list of dictionaries, which led to some improvement, but the app was still slow. To address this, the data collection process was completely overhauled. Instead of collecting data in a list and then sending it to the MySQL backend at the end of the trip, the data was now being sent to the database as soon as it was created during frame processing. This made the previously depicted while loop obsolete, so it was since removed. This change not only significantly reduced latency but also eliminated redundant rows, resulting in less memory usage and a more streamlined backend table.

Figure 10: Optimized Data Collection



```python
def video_frame_callback(frame: av.VideoFrame):
    frame = frame.to_ndarray(format="bgr24")  # Decode and convert frame to RGB
    #frame, play_alarm, row_dict = video_handler.process(frame, thresholds)  # Process frame
    frame, play_alarm = video_handler.process(frame, thresholds)  # Process frame

    with lock:
        shared_state["play_alarm"] = play_alarm  # Update shared state
        #shared_state['drwsy_df'] = pd.concat([shared_state['drwsy_df'], pd.DataFrame.from_records([row_dict])], ignore_index=True)
        #shared_state['drwsy_df'] = row_dict
        if video_handler.row_dict:
            sql_insert = f"""
                INSERT INTO {shared_state["table_name"]}
                (timestamp, EAR, MAR, eye_shut_counter, yawn_counter, alarm_counter, alarm_on)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
                """
            shared_state["connection"].cursor().execute(sql_insert, list(video_handler.row_dict.values()))
            shared_state["connection"].commit()

    # Encode and return BGR frame
    return av.VideoFrame.from_ndarray(frame, format="bgr24")
```

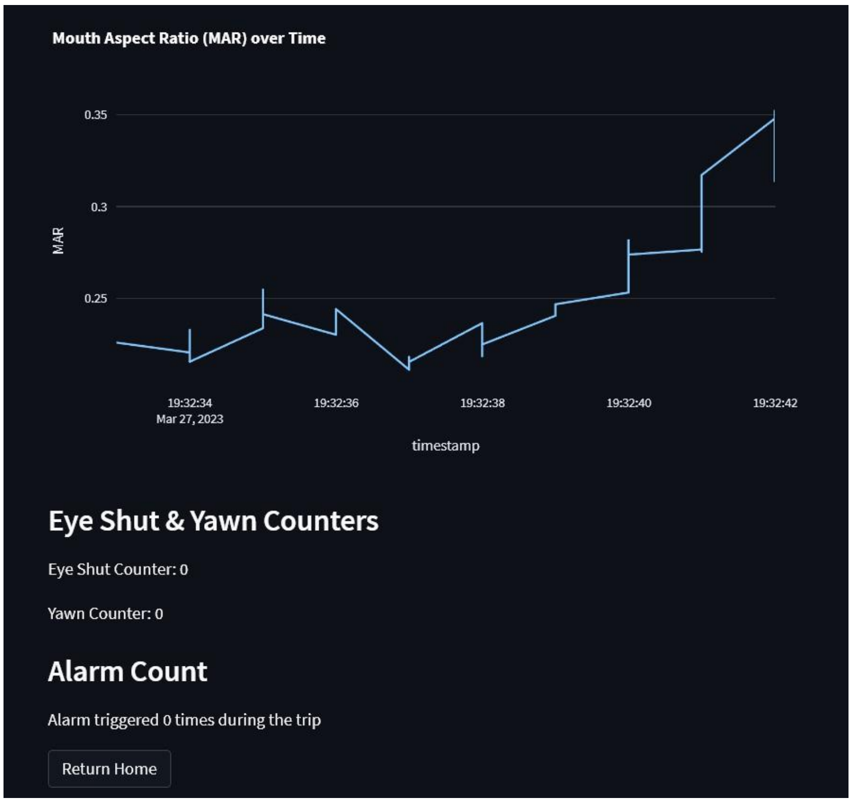
The system ran much faster with the new data collection logic

Case 2: Incorrect graphs

The purpose of the app goes beyond giving immediate feedback to drivers; it also includes an interactive dashboard that utilizes data from the backend database. The dashboard is designed to present information using interactive graphs that display metrics such as eye-aspect-ratio and alarm behavior over time. The intention is to provide drivers with an easy-to-use and visually appealing interface to help them understand their driving habits and make improvements as needed. However, there have been concerns regarding the accuracy of the graphs and their ability to provide reliable data.

The graphs in the dashboard were functional in terms of utilizing the data available. However, they presented an issue of displaying multiple y-values for a single time stamp, which could potentially lead to confusion and inaccuracies. Additionally, there was a possibility of incorrect values being posted on the dashboard if the user stopped and then resumed using the camera during the same trip.

Figure 11: Incorrect graph and counter values



Eye Shut & Yawn Counters

Eye Shut Counter: 0

Yawn Counter: 0

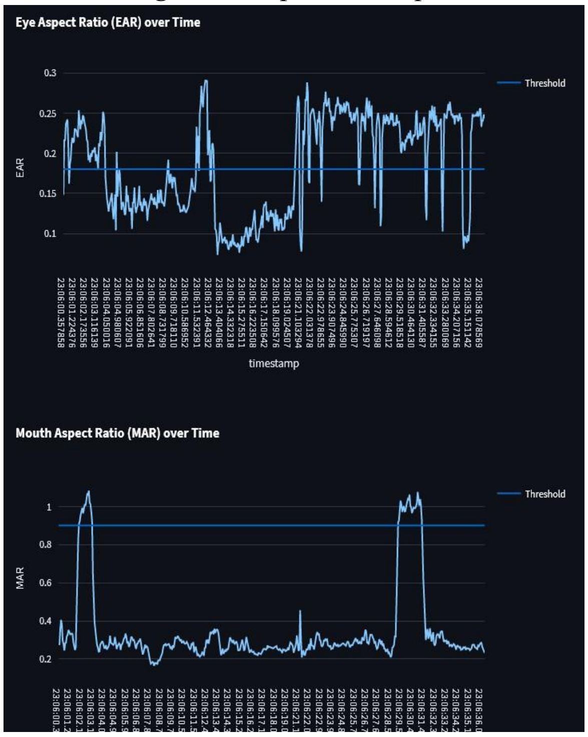
Alarm Count

Alarm triggered 0 times during the trip

There were two potential reasons for the issues with the dashboard. Firstly, it was discovered that the timestamp column in the database was not properly preserving the milliseconds when queried by the front end. As a result, the graphs were only displaying data for each second and ignoring the milliseconds. Secondly, it was observed that when the user refreshed the camera on the dashboard, the script was being rerun by Streamlit, causing some variables to be reset to their default values. This led to incorrect values being displayed on the dashboard.

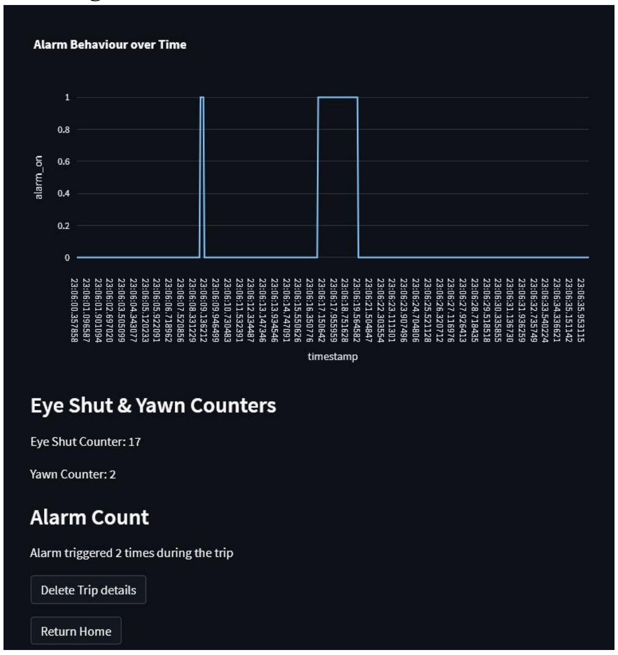
The first modification that was implemented involved changing the format of the timestamp column, which was originally represented as a DateTime object, to a floating-point value measuring the seconds elapsed since the epoch date of January 1st, 1970 UTC. This adjustment was instrumental in preserving the accuracy of the data to the millisecond level and eliminating any vertical lines in the graphical representation. Additionally, the values of the variables were preserved using the "session_state" functionality in Streamlit, which ensured that the information was retained throughout the duration of the session.

Figure 12: Updated Graphs



The vertical lines were rectified after milliseconds were preserved

Figure 13: Counters with accurate values



Eye Shut & Yawn Counters

Eye Shut Counter: 17

Yawn Counter: 2

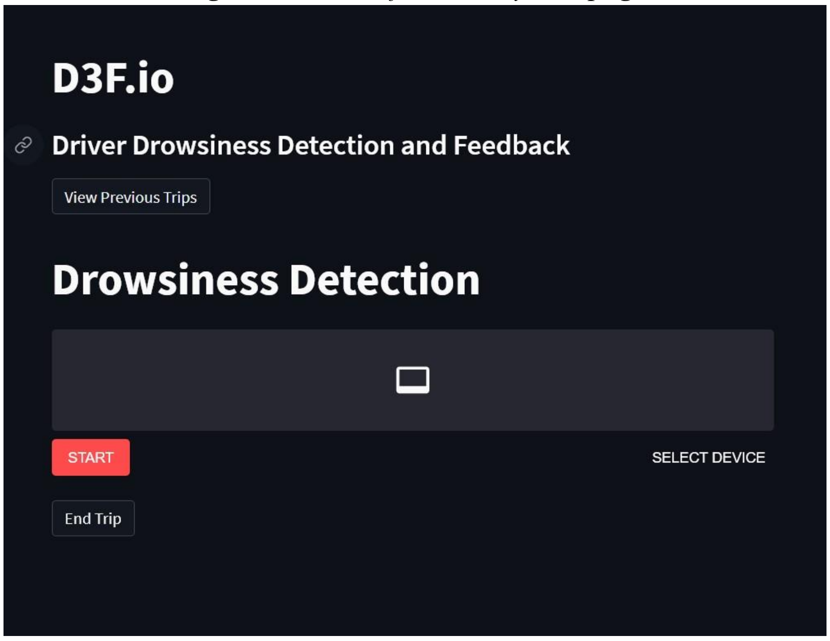
Alarm Count

Alarm triggered 2 times during the trip

Case 3: No trip error

In the scenario where a user ends a trip without having first started it, the dashboard creation function would encounter an error. Additionally, if the database did not contain any tables, clicking the "View Previous Trips" button would also produce an error.

Figure 14: Initial feedback system page

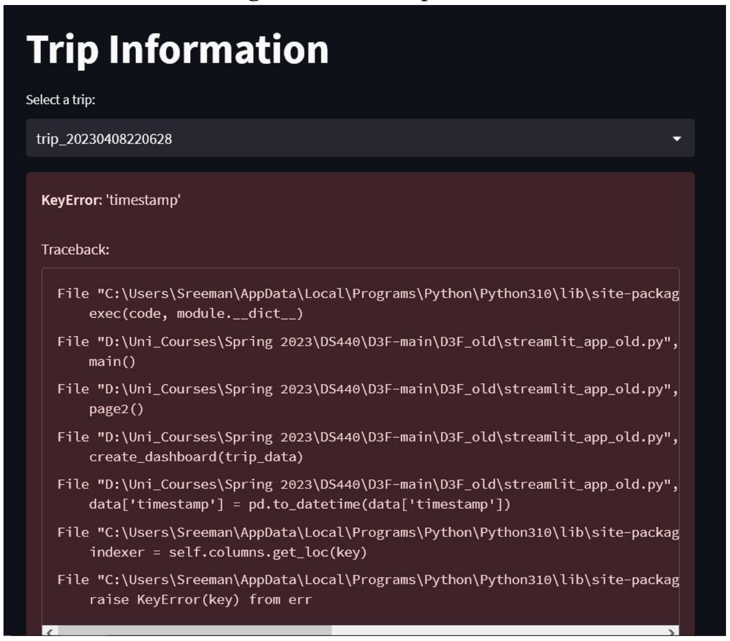


D3F.io

Driver Drowsiness Detection and Feedback

Drowsiness Detection

Figure 15: No trip error



Trip Information

KeyError: 'timestamp'

Traceback:

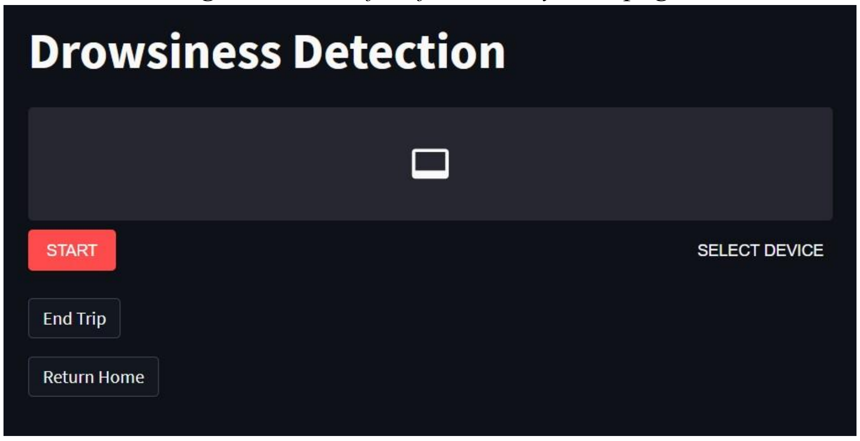
```
File "C:\Users\Sreeman\AppData\Local\Programs\Python\Python310\lib\site-packag
    exec(code, module.__dict__)
File "D:\Uni_Courses\Spring 2023\DS440\D3F-main\D3F_old\streamlit_app_old.py",
    main()
File "D:\Uni_Courses\Spring 2023\DS440\D3F-main\D3F_old\streamlit_app_old.py",
    page2()
File "D:\Uni_Courses\Spring 2023\DS440\D3F-main\D3F_old\streamlit_app_old.py",
    create_dashboard(trip_data)
File "D:\Uni_Courses\Spring 2023\DS440\D3F-main\D3F_old\streamlit_app_old.py",
    data['timestamp'] = pd.to_datetime(data['timestamp'])
File "C:\Users\Sreeman\AppData\Local\Programs\Python\Python310\lib\site-packag
    indexer = self.columns.get_loc(key)
File "C:\Users\Sreeman\AppData\Local\Programs\Python\Python310\lib\site-packag
    raise KeyError(key) from err
```

Error because there are no tables in the back-end to query

There are two reasons why the dashboard creation function could fail. The first reason is that the tables created upon clicking the "end trip" button could be empty, resulting in the function being unable to retrieve the necessary data. The second reason is that there are no tables in the database, which would prevent the database creation function from executing successfully. Either of these scenarios could result in the dashboard creation function failing to work as intended.

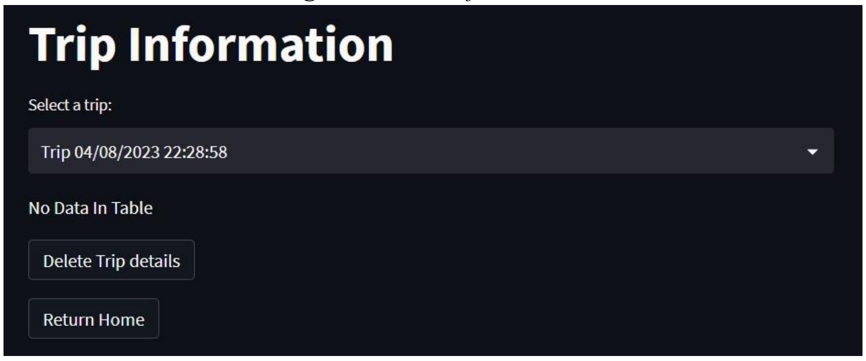
Several corrections have been made to an app's dashboard function. One addition was a "return home" button to stop the dashboard function from running and to remove any empty tables that were created. Another correction was that even if a user clicked the "end trip" button instead of the "return home" button, the app would still return the message "No data in Table." Additionally, a new button called "Delete trip details" was added, which allows users to delete any empty tables that were created, as well as any previous trip details.

Figure 16: Modified feedback system page



The return home button is added

Figure 17: Rectified Error



No Data In Table

An empty table now causes the app to return a string instead of an error

CONCLUSION

The development of the project remained largely on track according to the initial roadmap. However, there were a few challenges that led to the project falling short in certain aspects. One issue was the ambiguity in detecting and reading whether the driver is wearing accessories or apparel that cover the face, such as masks or shades. This problem stemmed from the facial detection logic of the Mediapipe library, which the team was unable to update as it was beyond the team's capabilities.

Another issue was the inability to deploy the app online because the web player, which is a function called "webrtc_streamer" in the code, is still in early development. The WebRTC packets of the web player are frequently dropped by the user's network, causing the web player to malfunction. To fix this issue, a "TURN" server needs to be set up, but this task is beyond the team's capabilities. Consequently, the project cannot be run on a mobile device at present.

Despite these setbacks, the team was eager to incorporate certain features that were not originally part of the roadmap. However, the optimization phase took longer than anticipated, consuming more resources and time. As a result, these additional features did not make it into the final build.

In conclusion, while the project largely adhered to the initial roadmap, some challenges hindered its progress, such as issues with facial detection and web player malfunctions. Nonetheless, the team remained enthusiastic about incorporating additional features, but the optimization phase consumed more time and resources than anticipated, resulting in these features being excluded from the final build.

FUTURE WORK

The proposed future work for the drowsiness detection system includes several features that aim to improve its effectiveness and usability. One such feature is the Variable Threshold, which will shorten the time it takes for the alarm to trigger when the driver yawns a specified number of times in a given time period. This time would also be shortened every time the alarm triggers. This variable threshold will ensure that the driver receives more frequent alerts when they are very drowsy.

Another feature that could enhance the system's accuracy is the use of a more sophisticated model. The current model does not adapt the threshold parameters based on the driver's facial features. However, implementing an ML model that can modify these threshold parameters according to the driver's facial features would provide more accurate feedback.

The system currently only accommodates one user at a time. To support multiple users, a login system can be implemented, which will allocate each user their own tables in the database. This will enhance the user experience and make it easier for the system to track multiple users' data.

The system currently uses Streamlit as its web player, but this sacrifices functionality for ease of use. Switching to Flask, Django, or similar platforms could address this issue, making the app more user-friendly while retaining its full functionality. This would also enable easier deployment of the app on mobile devices.

In the future, the system could integrate with health apps on mobile devices such as Apple Health. This would allow the system to monitor and provide feedback on the driver's drowsy behavior, contributing to a more comprehensive approach to monitoring driver safety.

REFERENCES

Phan, A.-C., Nguyen, N.-H.-Q., Trieu, T.-N., & Phan, T.-C. (2021, September 11). *An efficient approach for detecting driver drowsiness based on Deep Learning*. MDPI. Retrieved from https://www.mdpi.com/2076-3417/11/18/8441

Rivelli, E. (2022, September 13). *Drowsy driving 2021 Facts & Statistics*. Bankrate. Retrieved January 29, 2023, from https://www.bankrate.com/insurance/car/drowsy-driving-statistics/#:~:text=Each%20year%2C%20drowsy%20driving%20accounts,National%20Safety%20Council%20(NSC).

Rivelli, E. (2022, September 13). *Drowsy driving 2021 Facts & Statistics*. Bankrate. Retrieved January 29, 2023, from https://www.bankrate.com/insurance/car/drowsy-driving-statistics/#:~:text=Each%20year%2C%20drowsy%20driving%20accounts,National%20Safety%20Council%20(NSC).

Singh, V. (2022, December 13). *Driver drowsiness detection using Mediapipe in python*. LearnOpenCV. Retrieved from https://learnopencv.com/driver-drowsiness-detection-using-mediapipe-in-python/