IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 58, No. 6, June, 2025, pp. 1089-1100
Journal homepage: http://iieta.org/journals/jesa

The Dashboard of Remote Monitoring System for Driver Fatigue

Ramanda Nur Hidayat, Prajna Deshanta Ibnugraha*

School of Applied Science, Telkom University, Bandung 40287, Indonesia

Corresponding Author Email: prajna@telkomuniversity.ac.id

https://doi.org/10.18280/jesa.580601

Received: 3 April 2025
Revised: 10 May 2025
Accepted: 20 May 2025
Available online: 30 June 2025

Keywords:
driver monitoring, fatigue detection, real-time notifications, WebSocket, JWT authentication

ABSTRACT

This research presents a real-time driver fatigue monitoring system that integrates computer vision with web-based visualization and notification capabilities. The system utilizes the You Only Look Once version 8 (YOLOv8) algorithm to detect signs of drowsiness, such as closed eyes, mouth condition, head position, and yawning, achieving a detection accuracy of 99.3% based on a dataset from Roboflow. Detection results are transmitted via WebSocket to an interactive web dashboard, enabling live monitoring without the need to reload the page. Additionally, real-time notifications are sent via a Telegram Bot to the driver or operator when risky conditions are detected. To maintain data security, the system implements JSON Web Token (JWT) authentication and password hashing using bcrypt. Testing includes functional, performance, and API evaluations under scenarios simulating driver fatigue and system load. The results show stable response times ranging from 0.8 to 1.1 seconds and efficient API handling of up to 100 iterations. A comparative analysis indicates that the proposed system outperforms traditional and YOLOv5-based methods in terms of accuracy and real-time capability. Although user experience testing has not been conducted, the system has proven to be technically reliable and holds strong potential to reduce traffic accidents.

1. INTRODUCTION

Around the world, driving safety is still very important, especially considering the number of traffic accidents caused by unsafe driver behavior. The World Health Organization (WHO) says that around 1.3 million deaths are caused by traffic accidents each year, human factors such as drowsiness, being distracted by mobile phones, or smoking while driving are part of these causes [1].

According to Korlantas Polri, driver negligence is responsible for about 61% of accidents in Indonesia, with drowsiness as the main cause. Drivers are often unaware of the dangerous conditions they are experiencing, such as drowsiness or other activity distractions, which increase the likelihood of an accident [2].

Conventional methods, such as manual surveillance or vehicle sensors, have proven ineffective in addressing this issue as drivers are often unable to spot danger signs on their own. Moreover, the external sensors used by most current monitoring systems are not always precise, especially in detecting non-physical behaviors such as mobile phone distraction or smoking activity while driving [3]. Therefore, to reduce the possibility of accidents, there is a need for intelligent technology-based products that can detect risky driver behaviors in real-time with low latency and high accuracy.

The aim of this research is to create an automated system based on Artificial Intelligence (AI) and computer vision (CV) that can detect dangerous driver behavior. The YOLOv8 algorithm-also known as You Only Look Once version 8-is able to detect and classify various dangerous behaviors in real-time, especially focusing on drowsiness and parameters such as eyes, mouth, head position, and yawning with higher accuracy in low-light conditions [4].

The developed system will help monitor drivers in real time by displaying easy-to-read data and providing instant feedback without the need to reload the page. The system is also equipped with security technologies to protect user data, including JSON Web Token (JWT) based authentication and API keys to ensure secure communication between the frontend and backend [5].

2. LITERATUR REVIEW

Chapter 2 discusses driving safety, which is a major issue worldwide with many traffic fatalities each year. Human factors such as fatigue and cell phone distraction are often to blame; in Indonesia, driver negligence accounts for 61% of accidents [1]. This chapter also discusses traditional methods of monitoring drivers, such as the use of sensors to determine if drivers are tired. However, these methods often fail under light and cannot spot additional distractions such as cell phones or smoking. AI and YOLOv8 technology can detect risky behavior more accurately [3].

In this system, driver behavior data is obtained through visual detection using the YOLOv8 algorithm, which has proven effective in identifying facial features such as closed

eyes, head tilt, and yawning in real-time. Recent research by Zhang and Zhang [6] demonstrated that an optimized YOLOv8 model can detect driver drowsiness with high accuracy and real-time performance exceeding 50 FPS, making it highly suitable for in-vehicle applications.

The detected data is then integrated into a web-based system using WebSocket technology. WebSocket enables direct, bidirectional communication between the server and the user interface without requiring page reloads, which is essential for delivering instant feedback in driver monitoring systems. Studies have shown that WebSocket is particularly effective in real-time applications due to its low latency and persistent connection capabilities [7].

2.1 Driving safety and traffic accidents

Traffic accidents are still the leading cause of death and injury in the world, making driving safety a growing problem worldwide. According to the World Health Organization (WHO), traffic accidents cause more than 1.3 million deaths each year. Human factors such as smoking, drowsiness, and mobile phone distraction are the main causes of many accidents [1]. According to Korlantas Polri, driver negligence, including fatigue and cell phone distraction, accounts for 61% of traffic accidents in Indonesia. This suggests that drivers increase the risk of accidents as they are often unable to detect or identify danger signs [2].

2.2 Traditional methods of driver monitoring

The use of vehicle-based sensors and manual surveillance are two conventional ways to monitor driver behavior. Sensors based on physical measurements such as Eye Aspect Ratio (EAR) are used to detect driver fatigue. While these systems detect one driver behavior, such as fatigue, they cannot detect other non-physical behaviors, such as mobile phone distraction or smoking. According to research, these sensors often fail in more complex situations or with poor lighting [3]. In addition, manual surveillance requires constant supervision from an operator or supervisor, which can increase workload and increase the likelihood of human error.

Attention will increasingly focus on the use of Artificial Intelligence and computer vision to detect driver behavior. Algorithms such as YOLOv8-also known as You Only Look Once Version-are able to detect and classify objects in images in real-time with a higher degree of accuracy even in low lighting conditions. YOLOv8, which is widely used in computer vision applications, can detect a variety of behaviors that could endanger drivers, such as fatigue, mobile phone use, and smoking [8]. The system utilizes images captured from vehicle cameras to provide feedback and information about the driver's status instantly [9]. Deep learning and Convolutional Neural Networks (CNN) are also used in driver monitoring applications, such as YOLO. CNNs can identify more complex behavioral patterns and symptoms of fatigue or other impairments. Compared to traditional methods, these technologies can be more accurate in finding different types of risky behaviors [10].

2.3 Backend integration and real-time detection system

For driver monitoring systems, real-time integration of data from multiple sensor sources to the backend is a key issue. Many current systems rely on separate external sensors, resulting in delayed data delivery to the driver or vehicle operator. By ensuring fast delivery of feedback, cloud-based systems that use Socket.IO or REST APIs can solve this problem. Socket.IO allows data to communicate between the frontend and backend in real-time without the need to switch pages, improving system responsiveness [11]. This integration is crucial to ensure that drivers receive important information immediately and can take precautions.

Data security and privacy in driver monitoring applications is critical as the data collected can be highly sensitive. To protect user data, the use of technologies such as JSON Web Token (JWT) for authentication and API keys to ensure authorized access is essential. These systems guarantee that only identified persons can access the driver's personal data and related information [5].

3. METHODOLOGY

In this chapter, methods are used to create a YOLO and WebSocket-based driver fatigue monitoring system. The main focus of the system is to detect signs of fatigue in drivers and provide real-time feedback to improve road safety. In this chapter, system design, data transmission, data security, and testing are discussed. Socket.IO technology is used for real-time communication between the frontend and backend, and JWT is used to keep user access secure. This chapter will also discuss how Telegram Bots can be used to notify drivers or vehicle operators if dangerous behavior, such as fatigue, is detected.

The YOLOv8 model employed in this system was pre-trained to recognize driver behaviors such as eye closure, yawning, and head posture. The training dataset was sourced from the publicly available Roboflow platform, comprising 3,474 annotated images of drivers captured under diverse conditions, including varying lighting, camera angles, and the presence of accessories such as glasses and masks. The dataset was partitioned into 87% for training, 8% for validation, and 4% for testing. To enhance model generalization, data augmentation techniques were applied, including rotation, zoom, shear, color adjustments, blur, and noise. The model was trained using an input resolution of 416×416 pixels and achieved a mean Average Precision (mAP), precision, and recall of 99.3%. These results indicate that the detection model provides highly reliable input for the web-based monitoring system, enabling accurate real-time visualization and alert delivery via the integrated dashboard and Telegram notification service.

3.1 System design

In this chapter, methods are used to create a YOLO and WebSocket-based driver fatigue monitoring system. The main focus of the system is to detect signs of fatigue in drivers and provide real-time feedback to improve road safety. In this chapter, system design, data transmission, data security, and testing are discussed. Socket.IO technology is used for real-time communication between the frontend and backend, and JWT is used to keep user access secure. This chapter will also discuss how Telegram Bots can be used to notify drivers or vehicle operators if dangerous behavior, such as fatigue, is detected.

The system uses an interactive and easy-to-use web interface to monitor driver behavior in real-time [11]. The displayed data, including important information about the driver's status during the trip, can be viewed by both the driver

and the vehicle operator through this system. A login process that uses JSON Web Token (JWT) based authentication allows any user, both drivers and operators, to access the system [12]. This ensures that only authorized users can access sensitive data.

The displayed data includes various parameters, such as the identification of the monitoring session, the identity of the driver identified as the driver being monitored, and the driver's status, which includes the driver's eye condition, the driver's mouth condition, the driver's head position, and yawing (detection of yawning, which can indicate fatigue) [13]. In addition, the driver's fatigue (drowsiness) status will also be displayed based on the analysis of the driver's physical condition. In addition, the system records the Start Time and End Time for each monitoring session, thus providing an overview of the driver's time in a given situation.

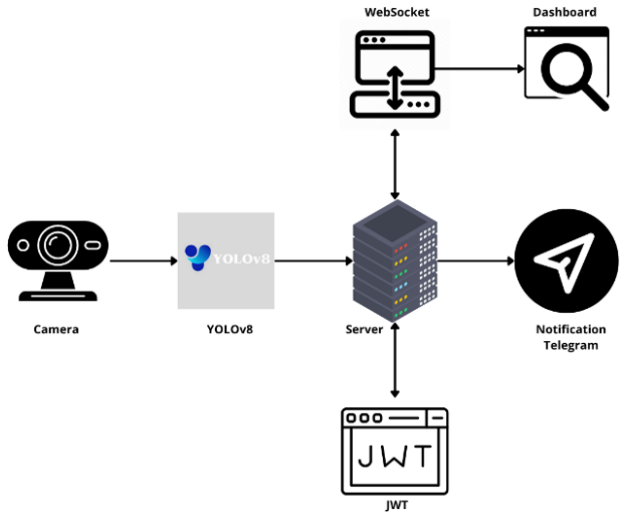

Figure 1. Architecture of real-time driver monitoring system

In Figure 1 illustrates the system architecture of the driver monitoring system. The process begins with a camera capturing real-time video of the driver, which is analyzed by the YOLOv8 detection module to identify signs of fatigue such as eye closure, yawning, and head tilt. The detection results are sent to a centralized server via HTTP for further processing.

The server plays a central role in managing data flow and system logic. It communicates with the WebSocket module to push real-time updates to the Website Dashboard, allowing monitoring personnel to visualize driver behavior without needing to refresh the page [14]. If risky behaviors such as drowsiness or distraction are detected, the server also triggers the Telegram notification system to send alerts to the driver or vehicle operator.

To ensure secure access, the system implements JSON Web Token (JWT) authentication. Users must log in to the system, and their credentials are verified before a token is issued. This token is then used to authorize access to protected resources and data.

This architecture ensures low-latency communication, secure data handling, and timely alerts, making it highly suitable for real-time driver monitoring applications.

Figure 2. Block diagram of driver fatigue detection system

In Figure 2, this flow, the process starts by activating YOLO (You Only Look Once), a computer vision algorithm used to identify the driver's fatigue level. When YOLO finds signs of fatigue, data such as the driver's eye condition, mouth condition, head position, yawning, and fatigue status are sent to the server via HTTP. Once the data is sent, the server will receive and process it. Furthermore, the processed data is automatically displayed on the web interface by the driver or vehicle operator. In addition, the system uses Telegram bots to send notifications to the driver or operator regarding dangerous situations such as driver fatigue or other distractions. In this way, the system provides real-time feedback that allows the driver or operator to take immediate precautions to improve road safety. The Start Time and End Time of the monitoring session are included in the data that shows the driver's condition throughout the journey.

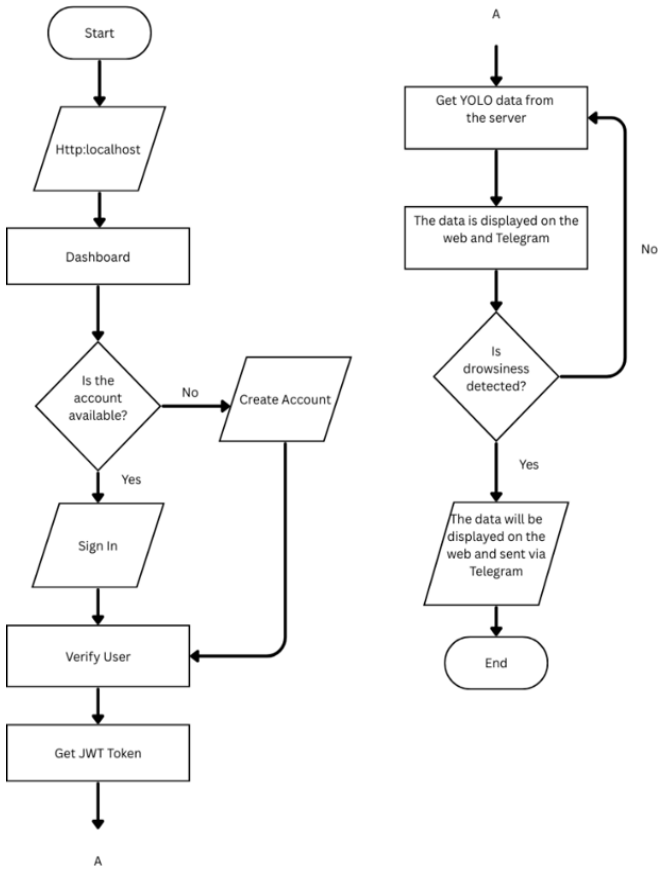

Figure 3. Flowchart system

In Figure 3, the process starts by accessing http://localhost, which takes the user to the main page or dashboard of the application. If a user account is available in the system, the user can log in and start the user verification process. Upon successful verification, the system will provide a JWT token for additional authentication, which allows the user to access the application's data and services [15]. After that, the process will continue. The system will prompt the user to create a new account if the user account is inaccessible.

YOLO data is collected from servers that track driver behavior. This information is displayed on the internet and sent via Telegram notifications to provide immediate feedback to the driver or vehicle operator after the user logs in and gets authenticated [16].

The data collected is then checked by the system to determine if the driver is experiencing drowsiness. If

drowsiness is detected, an alert will be immediately displayed on the web and sent to the driver via Telegram. If it is not detected, the system will return to the previous step or wait for new input to be processed. Once this process is complete, the user will receive feedback on the driver's status and can take the necessary actions to ensure the driver's safety on the road [17].

3.2 Data transmission system

For this project, WebSocket technology is used to connect the frontend and backend with this data delivery system enabling real-time two-way communication, which is essential for tracking driver behavior live and providing feedback quickly without interrupting or repeating requests. The data delivery process starts with the frontend client connecting to the server via WebSocket in the /api/detection namespace. After a successful connection, the server starts sending driver detection data, such as eye status, mouth, head position, and drowsiness status. This data is collected from YOLO processing on the backend [18].

The backend sends detection data to the client via WebSocket whenever a new detection is made. Immediately afterward, the data is processed and displayed on the web dashboard. Conversely, if the driver's drowsiness or fatigue status is detected, the system sends a notification via Telegram Bot. This bot notifies the driver or vehicle operator to take immediate precautions [19].

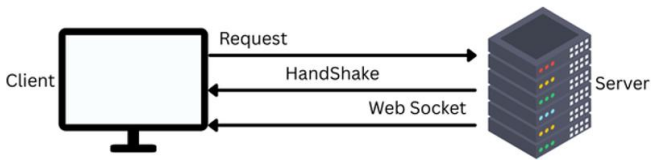

Figure 4. Communication between client and server

In Figure 4, WebSocket plays an important role in ensuring instant and efficient communication between the frontend and backend. Prior to the use of WebSocket, the communication process between the client and server relied on separate HTTP requests, which often resulted in delays and data inaccuracies in applications such as this driver monitoring application that require a quick response [20].

With WebSocket, data can be sent and received instantly without the need to initiate a new request every time the data changes. This reduces latency and provides real-time feedback that is essential for driver monitoring. Socket.io is used to manage various events, such as “disconnect”, which records the connection status between client and server, and to handle more stable WebSocket connections [18].

In addition, WebSocket enables full-duplex communication, meaning that both the server and client can transmit data at any time, which makes the interaction more dynamic and responsive. For example, this improves driver safety and convenience as drivers can instantly receive information about their fatigue status without having to wait or perform manual actions [19].

3.3 Data security

JSON Web Token (JWT) is a method used in web-based applications to authenticate users and maintain secure communication between client and server [15]. JWT is a string-shaped token that contains encoded information used to identify users and grant limited access to them without having to store the session on the server. Due to its ease of implementation, JWT is often used in RESTful API applications and real-time applications.

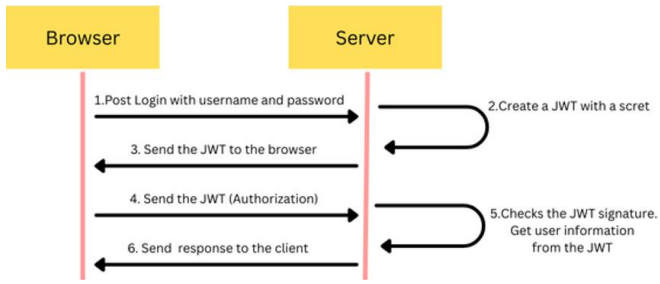

Figure 5. JWT authentication flow between browser and server

In Figure 5, the process starts when the user sends their username and password via HTTP POST to login to the server. Upon receiving the login request, the server verifies the user's credentials. If the credentials are valid, the server then creates a JWT containing the encrypted user information and signs it with the secret key. This token is then sent back to the user's browser. The browser stores this JWT for use in subsequent requests.

Furthermore, when a user accesses a page or data that requires authentication, the browser sends the JWT that has been stored in the HTTP authentication header to the server. The server then checks the received JWT signature by using a secret key to ensure that the token is valid and has not been altered. If valid, the server can retrieve user information from the token and grant access to the requested resource [21].

```
// If the password matches, generate a JWT token
const token = jwt.sign({ id: user.id }, process.env.JWT_SECRET, {
  expiresIn: "1h",
});
```

Figure 6. JWT token generator logic in the authentication module

In Figure 6 presents the core logic used to generate a JWT token in the authentication module of the system. Upon successful password verification, the server utilizes the jsonwebtoken library to create a token that includes the user's ID as its payload.

This token is signed using a secret key stored securely in the environment variable (process.env.JWT_SECRET) and is configured to expire within one hour (expiresIn: "1h"). This expiration setting ensures that the token has a limited validity period, reducing the risk of misuse or replay attacks [22].

The use of environment variables for storing the secret key enhances security by preventing hardcoded credentials in the source code. Additionally, the token generation process is triggered only after the user's credentials are validated, ensuring that only legitimate users receive access tokens. This implementation supports a stateless authentication model, where the server does not need to maintain session data, thereby improving scalability and reducing server load [22].

3.4 Telegram notification

Telegram Bot is a way to use Telegram bot to send automated messages to users through the Telegram app [23]. Telegram bots are accounts that users can interact with and are usually used to send alerts or important information. To use this bot, we have to create a bot in Telegram and get an API token that allows our application to use the bot to send messages through it.

Driver detection data such as eye status, mouth status, head position, drowsiness status, and detection time (start_time and end_time) are sent to the POST server endpoint to start the process. Then the data is stored in the database. After the data is saved successfully, the system uses WebSocket to check if there are any events that should be sent to the frontend. If there is new detection data, the server sends a "new_detection" event to the frontend to update the user display in real-time. In addition to sending data to the frontend, the system also uses the TelegramAlert send function, which sends a message to Telegram Bot containing driver behavior detection information, such as eye, mouth, and drowsiness status, as well as the detection time. The Telegram Bot then sends an instant notification to the driver or vehicle operator to alert them so that precautions can be taken immediately. This allows the driver or operator to respond immediately to risky conditions, such as when a driver is detected to be drowsy, to improve travel safety [24].

Figure 7. Drowsiness detection and notification flow to telegram bot

In Figure 7, this process is clearly depicted in the Block Diagram: (visual detection) detects the driver's drowsiness status and then sends the data to the server via HTTP. Once the data is received by the server, a message containing the detection information is sent to the server's Telegram bot. In addition, the Telegram bot notifies the driver or vehicle operator of the notification. In this way, Telegram Bot becomes an excellent tool for conveying important information quickly and directly without the need to interact with users manually. This ensures that any important alerts are received quickly, lowering the risk of accidents caused by driver negligence. With this integration, the system can provide instant notifications to drivers or operators via telegram to ensure a quick response to dangerous conditions such as drowsy drivers to maintain safe travel [25].

3.5 Hashing implementation for user data security

Hashing is used as the main method to protect data security in this project, especially to protect user passwords. The process of converting input data, such as passwords, into a shorter, consistent form is called hashing. This hash is created using a specific mathematical algorithm that is one-way, or one-way, and cannot be returned to the original data. Therefore, sensitive information such as passwords are not stored in the database in their original form, but only as hashes [26].

In Figure 8, Hashing is used in the figure to secure the user's password. Using an algorithm called "hash function", the password entered by the user will be hashed by the system. This process produces a hash text, which is an encrypted version of the original password [27]. One of the benefits is that the original password remains secure even if the database is compromised because the hash cannot be restored to its original form. Hashing protects sensitive data such as passwords.

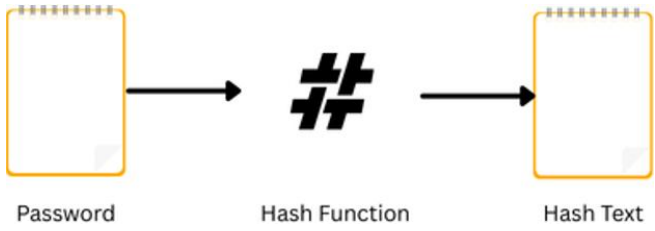

Figure 8. Transformation of password to hash text using hash function

3.6 Hashing implementation for user data security

The database system used on this website is MySQL, a relational database management system that is widely used because of its stability and ability to manage large amounts of data. Data storage in MySQL uses an SQL (Structured Query Language) structure, which means data is stored in the form of structured and interrelated tables. Each table has columns that define specific types of data, making it easier to manage, search, and analyze data. In addition to MySQL, the system also allows integration with NoSQL-based storage, such as MongoDB, which stores data in JSON format [28]. This approach is beneficial for data that is more flexible and does not always have a fixed structure, depending on the needs of the system.

The main data stored in the database comes from the detection results of driver behavior analyzed in real-time using YOLO v8 (You Only Look Once version 8) technology, which is a deep learning-based object detection model. This data reflects the driver's physical state and facial expressions, which are then processed to determine whether the driver is in a drowsy state or not. This is very important in the context of driving safety, as the system can provide early warnings if it detects signs of fatigue.

No	Name	Type	Collation	Null	Default	Extra
1	id	int(11)		No	None	AUTO_INCREMEN
2	driver_id	varchar(50)	utf8mb4_general_ci	No	None	
3	eye_state	varchar(50)	utf8mb4_general_ci	Yes	NULL	
4	mouth_state	varchar(50)	utf8mb4_general_ci	Yes	NULL	
5	head_pose	varchar(50)	utf8mb4_general_ci	Yes	NULL	
6	yawning	tinyint(1)		Yes	NULL	
7	drowsiness_status	varchar(50)	utf8mb4_general_ci	Yes	NULL	
8	start_time	datetime		Yes	NULL	
9	duration	int(50)		Yes	NULL	
10	end_time	datetime		Yes	NULL	

Figure 9. Table structure for driver behavior and fatigue monitoring

In Figure 9, To store driver status data, a special table is used with the following column structure: driver_id, eye_state, mouth_state, head_pose, yawning, drowsiness_status,

start_time, duration, and end_time. These columns represent various important parameters that are observed while the driver is operating the vehicle. The determination of drowsiness status is based on the analysis of several key parameters, such as eye_state, head_position, mouth_state, and yawning. If the system detects a certain combination of these parameters indicating fatigue, the drowsiness_status will be changed to “drowsy”, and the time and duration will be recorded in the database.

3.7 Testing design

System testing is carried out to ensure that each component of the YOLO and WebSocket-based driver fatigue monitoring system runs according to the functions that have been designed. Testing starts from the client side, namely the fatigue detection module that runs on the driver's device. In this test, various driver conditions are simulated, such as closed eyes, tilted head position, and yawning, to see if the system can detect correctly based on the YOLO model used. The success of detection is seen from whether the status data is sent to the backend server in real-time using the HTTP method [29].

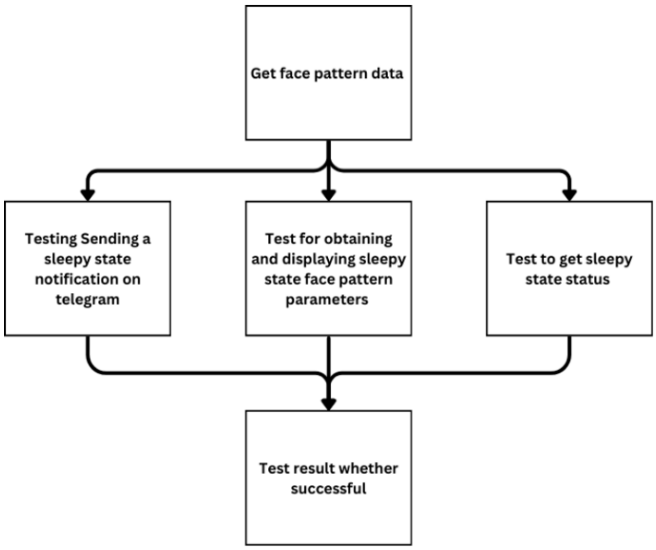

Figure 10. Workflow of testing the driver monitoring and telegram notification system

In Figure 10, Next, testing is done on the backend server. The server is tested to process the data sent from the client, save it to the database, and send it to the frontend using WebSocket. At this stage, the speed and accuracy of data transmission are the main focus. The backend is also tested to ensure data security with the implementation of JWT-based authentication [20]. Access from unauthorized users is denied, and only users who have logged in with a valid token can access the driver status information. This ensures that the user's personal data remains secure.

The frontend or user interface was tested to display real-time driver status data without the need to reload the page. The display was tested under various conditions to ensure responsiveness and clarity of information. In addition, the Telegram notification feature was tested to send automatic alerts when the system detects signs of fatigue. In addition, tests were conducted on the data storage system, especially on the user password storage mechanism [30]. Passwords were tested to be stored in hashed form using the bcrypt algorithm to avoid sensitive data leakage. All of these tests aim to ensure that the system is reliable, real-time, secure, and can be used effectively in monitoring the driver's condition to prevent fatigue accidents.

In this study, User Acceptance Testing (UAT) was not conducted due to the subjective nature of this type of evaluation, which typically requires the involvement of representative end-users. At the current stage of development, the system remains in its prototype phase, and access to a sufficiently diverse and representative user base was not available. UAT is generally employed to assess the usability and satisfaction of the system from the perspective of actual users; however, such an evaluation is more appropriate once the system reaches a more mature stage and is ready for broader deployment.

To compensate for the absence of UAT, the research focused on objective and quantifiable testing methodologies. Functional testing was carried out to ensure that each component of the system performed according to its intended specifications. Performance testing was also conducted to evaluate the system’s responsiveness and stability under varying loads, particularly in scenarios involving real-time data transmission and notification delivery. Furthermore, the accuracy of the fatigue detection mechanism was validated by comparing the system’s output with ground truth data, thereby ensuring the reliability of the detection model.

These testing approaches were selected to provide a robust technical evaluation of the system’s capabilities. While user-centered evaluations such as UAT are valuable, they are planned for future stages of development when the system is deployed in real-world environments. At this stage, the emphasis remains on verifying the technical soundness and operational effectiveness of the system under controlled conditions.

4. RESULT AND DISCUSSION

This chapter presents the test results of the web-based driver behavior detection and notification system through the Telegram application. The tests were conducted in a laboratory using simulated drivers to test the efficacy of the system in detecting signs of driver fatigue and non-compliance. The system being tested is YOLO v8, a deep learning-based object detection model that can detect various important parameters, including eye condition, mouth movements, head position, and yawning, which is the term for nodding or yawning. In addition, the test will include Telegram alerts that will be sent to drivers and vehicle operators in real-time if the system finds signs of negligence or drowsiness in driving behavior.

It should be noted that these tests have not been conducted in the field, so additional testing on drivers in real-life situations is needed to more accurately assess the system's performance. Testing in real-life situations will help assess the system's performance in the face of external variables such as environmental disturbances or more complex driver behaviors. The results of these laboratory tests show that the web-based driver fatigue detection and real-time notification technology has great potential for use. However, additional testing is needed to ensure that the system can operate well in more complex and dynamic work.

4.1 Notifications delivery time testing

In the first test, measurements were taken to measure the time to send notifications to the system when the driver's status

on the web changed to “sleepy”. The purpose of this test was to measure the responsiveness of the system in alerting the driver and vehicle operator in real time. Parameters such as eye condition, head position, mouth movement, and yawning- or nodding or yawning-were used in the tested system to detect signs of driver fatigue. If the driver system detects drowsiness, the status on the web interface will change to “drowsy”, and a notification will be sent via the Telegram app. This test measured and recorded the time taken for the notification to be sent. As part of the system evaluation, a series of ten trials were conducted to verify the consistency of notification delivery under various conditions. The test results showed that the speed of notification delivery and the accuracy of the system to identify the driver's status were excellent. Furthermore, the test data was evaluated and compared with the previous test results discussed in section 4.2 which tested the system's performance in more complex situations. As such, this test also provides an overview of the stability of the system in operating under real-time conditions as well as potential enhancements needed to improve the responsiveness of the system.

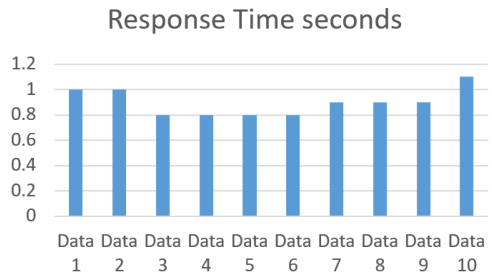

Figure 11. Response time for telegram notification delivery

Figure 11, In the first test, the main objective was to measure the notification delivery time when the status is changed to “sleepy” in the system. In this test, the system is programmed to detect certain conditions, such as a change in user status, and then send the relevant notification. The graph depicts the response time in seconds at 10 different data points, where each data point represents the time taken by the system to send and process the notification. The test results show that the response time is relatively consistent from Data 1 to Data 9, ranging from 0.8 to 1 second. This reflects the efficiency of the system in managing notification requests under normal conditions.

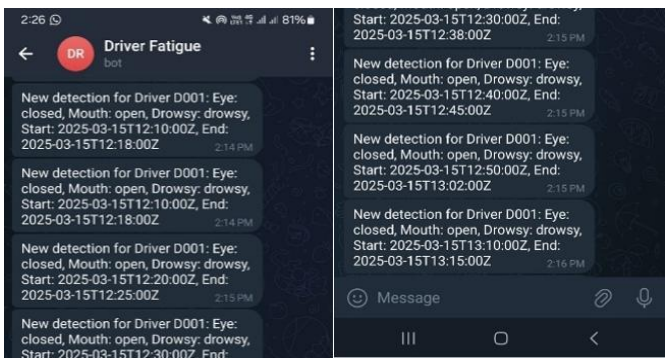

Figure 12. Telegram notification

Figure 12, however, in Data 10 there is a significant spike in response time, reaching more than 1 second. This spike could be caused by several factors, such as increased system load, network delays, or less efficient management of the notification queue. Overall, although the system showed stable response times in most of the tests, the spike in Data 10 indicates a bottleneck or technical issue that needs to be analyzed further.

To improve the efficiency of the system, it is necessary to further analyze the cause of the spike on Data 10, including evaluating server availability, queue management, and network conditions. Future tests could involve simulations with a larger number of notifications to test how the system handles higher loads.

```
// Make sure the detectionend is initialized before emitting.
if (detectionend) {
  detectionend.emit("new_detection", data); // Emit events to the frontend
} else {
  console.warn("Socket.io not initialized, skipping emit.");
}

const message = `New detection for Driver
${driver_id}: Eye: ${eye_state},
 Mouth: ${mouth_state},
 Status: ${drowsiness_status},
 Start: ${start_time},
 End: ${end_time},
 Duration: ${duration}`;
sendTelegramAlert(message);
```

Figure 13. Notification logic flow from detection to telegram alert delivery

Figure 13, the telegram notification mechanism is activated after the detection data is processed and successfully stored in the database. The system first verifies whether the WebSocket connection (detectionend) has been properly initialized. If the connection is active, a new_detection event is emitted to the frontend to update the dashboard interface in real time. If the WebSocket is not available, the system logs a warning and skips the emission process to prevent runtime errors.

Subsequently, the system constructs a structured message containing key detection parameters, including the driver ID, eye and mouth conditions, drowsiness status, and the start and end timestamps of the detection event. This message is then passed to the sendTelegramAlert() function, which delivers the alert to a designated Telegram chat using the Telegram Bot API.

This logic ensures that notifications are only sent when valid detection data is available and the system is in a stable operational state. Furthermore, it prevents unnecessary alerts by verifying the readiness of the communication channel before attempting to emit or notify. This approach enhances the reliability of the alert system and minimizes the risk of false or redundant notifications.

4.2 Testing to obtain drowsiness condition parameter responses

This test measures driver drowsiness by monitoring the following parameters: Driver ID for driver identification, Eye State for eye state (open or closed), Mouth State for mouth state (open or closed), Head Pose for head position (normal or tilted), Yawning to measure whether the driver yawns, Drowsiness Status to indicate whether the driver is sleepy, and Start Time, End Time, and Duration to record the start, end, and duration times of the test. These parameters are used to assess the drowsiness level of the driver.

In Table 1 present the result of the test, which recorded in this table aims to measure the response time of sending data

related to the drowsiness state of driver D001. Each row records data about the driver indicating drowsiness, with parameters such as eyes closed, mouth open, yawning (true), and drowsy status. The data also records varying head positions of the driver, such as normal, tilted, and slightly tilted head positions, which can indicate fatigue or inattention. The test duration for each data varies from 5 to 12 seconds.

Table 1. Data sending test results

Data	Response (ms)
D001, closed, open, normal, true, drowsy, 2025-03-15T12:00:00Z, 2025-03-15T12:05:00Z, 5s	35
D001, closed, open, tilted, true, drowsy, 2025-03-15T12:10:00Z, 2025-03-15T12:18:00Z, 8s	53
D001, closed, open, normal, true, drowsy, 2025-03-15T12:20:00Z, 2025-03-15T12:25:00Z, 5s	
D001, closed, open, slightly tilted, true, drowsy, 2025-03-15T12:30:00Z, 2025-03-15T12:38:00Z, 8s	10
D001, closed, open, normal, true, drowsy, 2025-03-15T12:40:00Z, 2025-03-15T12:45:00Z, 5s	54
D001, closed, open, normal, true, drowsy, 2025-03-15T12:50:00Z, 2025-03-15T13:02:00Z, 12s	
D001, closed, open, tilted, true, drowsy, 2025-03-15T13:10:00Z, 2025-03-15T13:15:00Z, 5s	9
D001, closed, open, normal, true, drowsy, 2025-03-15T13:20:00Z, 2025-03-15T13:28:00Z, 8s	13
D001, closed, open, normal, true, drowsy, 2025-03-15T13:30:00Z, 2025-03-15T13:37:00Z, 7s	
D001, closed, open, normal, true, drowsy, 2025-03-15T13:40:00Z, 2025-03-15T13:45:00Z, 5s	12

Each line shows a “Success” status after successful data transmission. For data transmission, milliseconds (ms) are used to indicate the response time, which ranges from 9 ms to 106 ms. A faster response time, such as 9 ms, indicates faster processing, while a longer response time, such as 106 ms, indicates that data transmission takes a little more time, although it remains within an acceptable range. Overall, these tests were successful; they demonstrated the system's ability to send and process data in a relatively quick time, which suggests that the system is quite effective at identifying driver drowsiness.

4.3 Dashboard testing

In Figure 14, In the dashboard section, tests were conducted to monitor the driver's drowsiness condition using several parameters, which were then displayed in tabular form on the dashboard of the driver behavior detection system. The data displayed in the table includes various attributes related to the driver's drowsiness status, such as Driver ID, Eye State, Mouth State, Head Pose, Yawning, Drowsiness Status, Start Time, End Time, and Duration. From the table shown, it can be seen that driver D001 experienced a drowsy condition in each test conducted. In each row, the drowsiness_status shows the value “Drowsy”, which means the driver is in a drowsy condition. This can be seen from several parameters such as Eye State which shows closed eyes, and yawning which indicates the driver is yawning.

The test duration varies between 5 to 12 seconds. This shows that the system can detect driver drowsiness in a relatively short time. Although there is a slight variation in duration, the overall system shows good responsiveness to the driver's drowsy state.

In Figure 15, Testing on the POST detection endpoint was performed with 100 iterations, resulting in a total test duration of 8 seconds 720 milliseconds (8s 720ms). During the test, all requests were successfully processed with a response status of 200, indicating that there were no errors in any of the API requests. The average response time for each request was 20 milliseconds, indicating that the API responded very quickly. The time distribution diagram also shows that most requests completed within a short time, with little variation in response duration. These test results show that the API works efficiently, can handle the test load well, and provides fast and stable results.

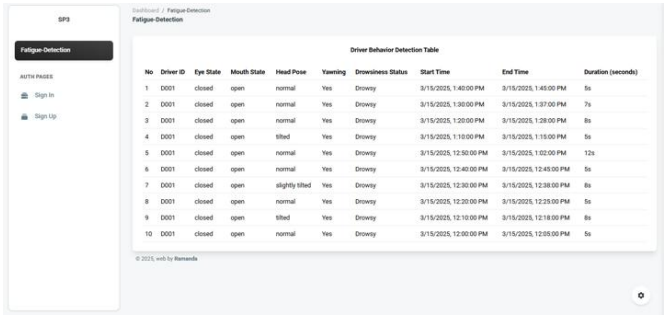

Figure 14. Dashboard Monitoring

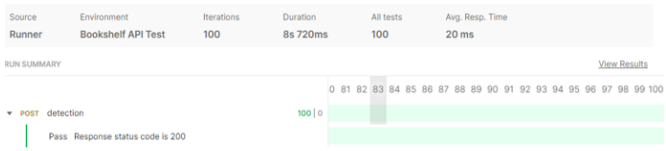

Figure 15. Post-detection API test summary and response times

In Figure 16, Testing on the GET detection endpoint was conducted with 100 iterations, resulting in a total test duration of 8 seconds 794 milliseconds (8s 794ms). During the test, all requests were successfully processed with a response status of 200, indicating that the API worked well and as expected. The average response time for each request was 7 milliseconds, indicating that the API is very efficient in providing responses. The time distribution diagram shows that most requests completed with very fast and stable response times, with no major fluctuations in duration. These test results show that the API on the GET detection endpoint has excellent performance, responsiveness, and stability.

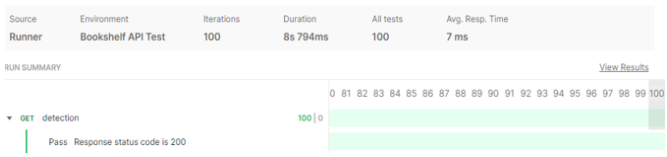

Figure 16. Get-detection API test summary and response times

4.4 User registration process with data security using hashing

In Figure 17, figure explanation the application of hashing is very important in securing user data in the sign-in and sign-up system. In the view shown in figure, this is the sign-up page for a new user, in this case an admin. The user is required to enter a name, email, and password to create an account. After registering, the data entered will be automatically sent to the

server. One important step is that the entered password will be converted into a hash value using a secure algorithm such as bcrypt or SHA-256. This hashing process converts the original password into a random string that cannot be restored to its original form. This aims to keep in mind that even if the database is accessed by unauthorized parties, they will only see the hash value and not the original password which can be misused.

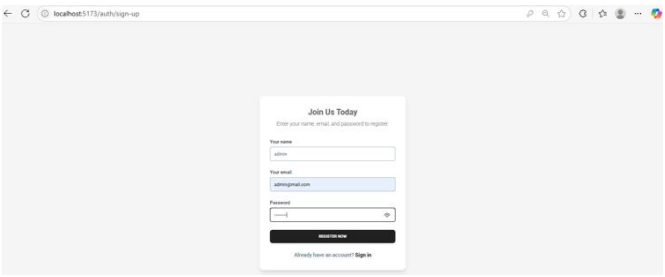

Figure 17. User registration page for secure data access

In Figure 18, shows a view of the database that stores registered user data. The password column indicates that the value stored is not the original password entered by the user, but the hash value generated from the hashing process. For example, in the first row, the password entered by the user has been converted into a long random string. This is the result of the hashing algorithm applied at the time of sign-up. With this method, even if this database is accessed by unauthorized parties, they will not be able to know the user's original password as only the hash value is visible. This strengthens the security of user data by preventing the theft of sensitive data such as passwords. The created_at field indicates the time of account creation, which can be used to track when the account was created.

name	email	password	created_at
admin	admin@mail.com	$2b$10$Rl/n/ryqrqvGXRnn8Su6cujDUZOT/Je3yd3k0B.qSE0...	2025-04-18 15:35:12

Figure 18. Database entry for user encrypted password

The application of this hashing technique ensures that the data stored in the database is safe from unauthorized access, maintains user privacy, and prevents potential misuse of personal data.

In Figure 19, shows a comparison of user data that has a password in plain text or plaintext without hashing technology. In this case, the password in the password field is the same as the user entered, i.e. 12345678. This shows that this password is no longer protected. Since the database is stored in an easily accessible format, the password can be easily read and misused if an unauthorized party is able to access it.

id	name	email	password	create_at
1	admin	admin@mail.com	12345678	2025-04-18 18:46:56

Figure 19. Database entry for user with plaintext password

4.5 Performance testing

In Figure 20 illustrates the response time graph obtained during a 10-minute performance testing session. This test was designed to evaluate the responsiveness of both the REST API endpoints and the WebSocket communication channel under varying virtual user loads. The number of simulated users was gradually increased from 20 to 100 and then decreased back to 20, while monitoring the system's ability to handle concurrent requests and real-time data transmission.

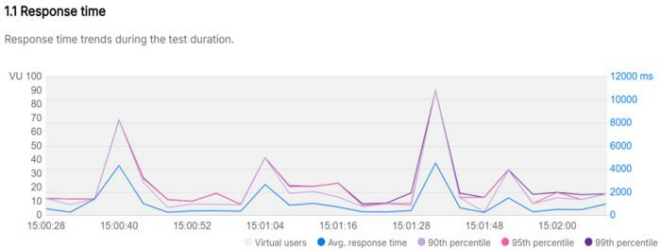

Figure 20. Performance

The graph shows that the average response time (represented by the blue line) remained relatively stable during most of the test duration. However, significant spikes in response time were observed as the number of concurrent users increased. The first noticeable spike occurred around 15:00:40, coinciding with the initial rise in user load. The most substantial spike was recorded at 15:01:28, when the system reached its peak load of 100 users. At this point, the response time exceeded 10,000 milliseconds (10 seconds), as indicated by the sharp rise in the 95th and 99th percentile lines.

These results suggest that while the system performs efficiently under normal conditions, it encounters performance bottlenecks during peak load, particularly in handling simultaneous WebSocket messages and REST API requests. Once the load was reduced, the response time stabilized, indicating the system's ability to recover after high-stress conditions.

This test highlights the need for further optimization, such as implementing load balancing, asynchronous processing, or message queuing, to enhance the system's scalability and ensure consistent performance under high concurrency.

4.6 Comparative analysis with existing

This test highlights the need for further optimization, such as implementing load balancing, asynchronous processing, or message queuing, to enhance the system's scalability and ensure consistent performance under high concurrency.

To evaluate the effectiveness of the proposed driver fatigue monitoring system, a comparative analysis was conducted against existing systems that utilize traditional sensor-based methods and earlier versions of computer vision models such as YOLOv5. The comparison focuses on key performance indicators including detection accuracy, response time, real-time feedback capability, and system scalability.

In Table 2, The proposed system demonstrates competitive performance in terms of detection accuracy and response speed. The YOLOv8 model employed in this system was pre-trained to recognize driver behaviors such as eye closure, yawning, and head posture. The training dataset was sourced from the publicly available Roboflow platform, comprising 3,474 annotated images of drivers captured under diverse conditions, including varying lighting, camera angles, and the presence of accessories such as glasses and masks. The dataset was partitioned into 87% for training, 8% for validation, and 4% for testing. Data augmentation techniques were applied to enhance generalization, including rotation, zoom, shear, color adjustments, blur, and noise. The model was trained using an

input resolution of 416×416 pixels and achieved a mean Average Precision (mAP), precision, and recall of 99.3%, indicating highly reliable detection performance for real-time monitoring and alert delivery via the integrated dashboard and Telegram notification service [31].

Table 2. Comparative analysis

Featur	Proposed System (YOLOv8 + WebSocket)	Traditional Sensor Based System	YOLOv5-Based System
Drowsiness Detection Accuracy	99.3% (based on Roboflow dataset)	75% (EAR-based sensors)	85% (YOLOv5)
Real-Time Notifications	Yes	No	Limited
Response Time	0.8-1.1 Seconds	2-3 Seconds	1.5-2 seconds
Security (JWT+ bcrypt)	Strong	Weak (no encryption)	Moderate

YOLOv5-based systems trained on facial features such as eye closure and yawning achieve approximately 73.7% accuracy in detecting driver drowsiness, but it is limited in responsiveness and adaptability to varying lighting conditions. In contrast, traditional sensor-based systems relying on Eye Aspect Ratio (EAR) or steering pressure typically reach around 70-75% accuracy and are less suitable for real-time applications due to their intrusive nature and lack of contextual awareness [31, 32].

These findings highlight the superior performance of the proposed YOLOv8-based system in both accuracy and responsiveness, making it a more viable solution for real-world deployment in dynamic driving environments.

4.7 Eror source and statical validation

Despite the high accuracy and responsiveness demonstrated by the proposed driver fatigue monitoring system, several factors may introduce variability or error in detection outcomes. These factors must be critically examined to ensure the robustness and reliability of the system in diverse operational environments.

One of the primary sources of error is lighting conditions. Inadequate or uneven illumination can significantly impair the visibility of facial features, particularly the eyes and mouth, which are essential for detecting signs of drowsiness. Similarly, occlusions caused by accessories such as eyeglasses, face masks, or headwear may obstruct key facial landmarks, leading to misclassification or missed detections.

Another contributing factor is the camera angle and resolution. Improper camera placement or low-resolution video input can reduce the precision of the YOLOv8 detection model, especially when subtle facial movements are involved. Furthermore, driver variability—including differences in facial structure, behavioral patterns, and expressions of fatigue—can affect the generalization capability of the model across different individuals.

Table 3, To evaluate the consistency of the system's performance, a statistical analysis was conducted using response time data collected from ten trials of drowsiness detection and notification delivery. The results are presented in Table 3.7. The analysis yielded a mean response time of 0.91 seconds and a standard deviation of 0.09 seconds, indicating a high level of consistency across trials. These findings suggest that the system performs reliably under controlled laboratory conditions.

For future work, it is recommended to incorporate more advanced statistical validation techniques, such as hypothesis testing and confidence interval analysis, to further substantiate the significance of the experimental results. Additionally, field testing under varied environmental conditions and with a broader demographic of drivers will be essential to assess the system's robustness and scalability in real-world applications.

Table 3. Performance consistency test

Trial	Response Time (Seconds)
1	0.8
2	0.9
3	0.8
4	0.9
5	1.0
6	0.8
7	0.9
8	1.1
9	0.9
10	1.0

5. CONCLUSIONS

This research has demonstrated the development of a real-time driver fatigue monitoring system that combines computer vision techniques with interactive web-based visualization and notification features. Through the use of the YOLOv8 algorithm, the system can accurately identify fatigue-related indicators—such as eye state, mouth activity, head orientation, and yawning—achieving a 99.3% accuracy rate based on the Roboflow dataset. Real-time detection data is streamed to a responsive web dashboard via WebSocket, providing continuous updates without requiring page reloads. Simultaneously, alert messages are dispatched instantly through Telegram Bot to inform users when potential danger is detected. System evaluations, including functional and stress testing, revealed consistent response times between 0.8 and 1.1 seconds and efficient API handling under a simulated workload of 100 iterations. Compared to existing traditional systems and those using YOLOv5, the proposed solution delivers improved precision and responsiveness. While user-centered evaluations were not part of this study, the system has proven to be technically sound and scalable. Further research is recommended to evaluate user interaction and refine the user interface design to better support deployment in real-world driving conditions.

REFERENCES

[1] Ahmed, S.K., Mohammed, M.G., Abdulqadir, S.O., El-Kader, R.G.A., El-Shall, N.A., Chandran, D., Ur Rehman, M.E., Dhama, K. (2023). Road traffic accidental injuries and deaths: A neglected global health issue. Health Science Reports, 6(5): e1240. https://doi.org/10.1002/hsr2.1240

[2] Iridiastadi, H., Abdurrahman, I., Puspasari, M., Soetisna, H.R. (2020). Fatigue and sleepiness during long-duration driving: A preliminary study among Indonesian

commercial drivers. Transport Problems, 15(2): 17-24. https://doi.org/10.21307/tp-2020-016

[3] Sun, W., Si, Y., Guo, M., Li, S. (2021). Driver distraction recognition using wearable IMU sensor data. Sustainability, 13(3): 1342. https://doi.org/10.3390/su13031342

[4] Aote, S.S., Tank, K., Khanna, A., Padole, V., Rewatkar, A. (2024). Driver monitoring based on drowsiness and yawning using YOLOv8. In 2024 International Conference on Current Trends in Advanced Computing (ICCTAC), Bengaluru, India, pp. 1-6. https://doi.org/10.1109/ICCTAC61556.2024.10581349

[5] Sakthy, S.S., Sriraman, S., Krishna, R. (2024). DMS-driver monitoring system for license test using machine learning. In 2024 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India, pp. 1-5. https://doi.org/10.1109/ICPECTS62210.2024.10780229

[6] Zhang, M., Zhang, F. (2024). Lightweight YOLOv8 networks for driver profile face drowsiness detection. International Journal of Automotive Technology, 25(6): 1331-1343. https://doi.org/10.1007/s12239-024-00103-w

[7] Sharma, N., Agarwal, R. (2023). HTTP, WebSocket, and signalR: A comparison of real-time online communication protocols. In International Conference on Mining Intelligence and Knowledge Exploration. Cham: Springer Nature, Switzerland. pp. 128-135. https://doi.org/10.1007/978-3-031-44084-7_13

[8] Ibnugraha, P.D., Sani, M.I., Sari, M.I., Rizal, M.F., Hanifa, F.H., Kurniawan, A.P. (2023). Automatic Passenger Counting (APC) for Online Event Data Recorder (EDR). In 2023 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD), Denpasar, Indonesia, pp. 89-93. https://doi.org/10.1109/ICoABCD59879.2023.10390960

[9] Ravichandran, M., Laxmikant, K., Muthu, A. (2023). Efficient vehicle detection and classification using YOLO v8 for real-time applications. In 2023 Global Conference on Information Technologies and Communications (GCITC), Bangalore, India, pp. 1-5. https://doi.org/10.1109/GCITC60406.2023.10426587

[10] Zhou, L., Li, S., Wang, Y. (2023). Fatigue detection and early warning system for drivers based on deep learning. In 2023 IEEE 3rd International Conference on Data Science and Computer Application (ICDSCA), Dalian, China, pp. 1348-1351. https://doi.org/10.1109/ICDSCA59871.2023.10392792

[11] Gote, A. (2024). Real-time interactivity in hybrid applications with web sockets. International Research Journal of Modernization in Engineering Technology and Science, 6(1): 2459-2463. https://www.doi.org/10.56726/IRJMETS48494

[12] Sengar, S.S., Kumar, A., Singh, O. (2024). VigilEye—Artificial Intelligence-based real-time driver drowsiness detection. arXiv preprint arXiv:2406.15646. https://doi.org/10.48550/arXiv.2406.15646

[13] Sharma, K., Rahul. (2023). Vehicle registration using blockchain and JSON web token for authentication on restful web service from India perspective. In International Conference on Soft Computing and Signal Processing. Springer Nature, Singapore, pp. 83-92. https://doi.org/10.1007/978-981-99-8628-6_8

[14] Du, X., Yu, C., Sun, T. (2024). Multi—Parameter fusion driver fatigue detection method based on facial fatigue features. Journal of the Society for Information Display, 32(9): 676-690. https://doi.org/10.1002/jsid.1343

[15] Abdul-Rahaim, L.A., Gheni, H.M., Ameen, H.A. (2022). Real-time traffic violation system vehicles to cloud data exchange based driver behaviour. In 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, pp. 1-6. https://doi.org/10.1109/HORA55278.2022.9800028

[16] Mahindraka, P. (2020). Insights of JSON web token. International International Journal of Recent Technology and Engineering, 8(6): 1707-1710. https://doi.org/10.35940/ijrte.F7689.038620

[17] Pawar, S., Jadhav, D.B., Godse, D., Jadhav, R., Thakur, S. (2024). Vision-based empty shelf detection in retail with real-time telegram notifications for efficient restocking. International Journal of Electronics and Communication Engineering, 11(7): 180-187. https://doi.org/10.14445/23488549/IJECE-V11I7P118

[18] Karthik, C., Praveen, K.N., Clarinda, S.S. (2025). University transport connect: Real-time university bus tracking system using Flask WebSocket, and machine learning predictive analysis. International Journal for Research Trends and Innovation, 10(1): a892-a895. https://www.ijrti.org/papers/IJRTI2501107.pdf.

[19] Navaneethakrishnan, M.M.E., Joel, A.M.A., Raj, M.L., Kanna, K.S. (2021). Developing a cross-platform vehicle tracking system using node.js and WebSocket. International Journal of Advanced Research in Computer Science Engineering and Information Technology, 6(3): 1420-1425.

[20] Dubey, A. (2023). Enhancing real time communication and efficiency with WebSocket. International Research Journal of Engineering and Technology, 10(8): 891-895. https://www.irjet.net/archives/V10/i8/IRJET-V10I8147.pdf.

[21] Sharma, A., Shrivastava, V., Pandey, A., Sharma, E.A. (2024). Providing authentication using JSON web tokens for enhancing user security. International Journal of Research Publication and Reviews, 5(4): 5309-5312. https://ijrpr.com/uploads/V5ISSUE4/IJRPR25377.pdf.

[22] Xu, B.W., Jia, S.J., Lin, J.Q., Zheng, F.Y., Ma, Y., Liu, L.M., Gu, X.Z., Song, L. (2023). JWTKey: Automatic cryptographic vulnerability detection in JWT applications. In European Symposium on Research in Computer Security, Springer Nature, Switzerland, pp. 263-282. https://doi.org/10.1007/978-3-031-51479-1_14

[23] Wahid, A., Parenreng, J.M., Kusnandar, W.C.K., Adi, P.D.P. (2024). Telegram bot-based flood early warning system with WSN integration. ILKOM Jurnal Ilmiah, 16(2): 151-160. https://doi.org/10.33096/ilkom.v16i2.1699.151-160

[24] Satti, S.K., Rajareddy, G.N., Ravipati, N.V., Samanvita, S.G. (2024). Drowsy alert: A system to detect and alert driver's drowsiness for road safety. In 2024 IEEE Students Conference on Engineering and Systems (SCES), Prayagraj, India, pp. 1-6. https://doi.org/10.1109/SCES61914.2024.10652546

[25] Dey, M., Majhi, M., Koda, Y., Maji, B., Chatterjee,, R. (2024). Drowsy driver detection system. International Journal for Research in Applied Science & Engineering Technology, 12(v): 1488-1492.

https://doi.org/10.22214/ijraset.2024.61832
[26] McGiffen, M. (2022). Hashing and salting of passwords. In Pro Encryption in SQL Server 2022: Provide the Highest Level of Protection for Your Data, USA, pp. 269-275. https://doi.org/10.1007/978-1-4842-8664-7_19
[27] Touil, H., El Akkad, N., Satori, K. (2021). Securing the storage of passwords based on the MD5 HASH transformation. In International Conference on Digital Technologies and Applications, Fez, Morocco, pp. 495-503. https://doi.org/10.1007/978-3-030-73882-2_45
[28] Matallah, H., Belalem, G., Bouamrane, K. (2021). Comparative study between the MySQL relational database and the MongoDB NoSQL database. International Journal of Software Science and Computational Intelligence (IJSSCI), 13(3): 38-63. https://doi.org/10.4018/IJSSCI.2021070104
[29] Li, X., Li, X., Shen, Z., Qian, G. (2024). Driver fatigue detection based on improved YOLOv7. Journal of Real-Time Image Processing, 21(3): 75. https://doi.org/10.1007/s11554-024-01455-3
[30] Kumbhar,, A.S.S., Athane, P.R., Nandangi, A.A., Pote, S.S., Chopade, P.M. (2024). Driver drowsiness detection system with real-time monitoring and historical tracking. International Research Journal of Modernization in Engineering Technology and Science, 6(2): 2259-2264. https://www.doi.org/10.56726/IRJMETS49771
[31] El-Nabi, S.A., El-Shafai, W., El-Rabaie, E.S.M., Ramadan, K.F., Abd El-Samie, F.E., Mohsen, S. (2024). Machine learning and deep learning techniques for driver fatigue and drowsiness detection: A review. Multimedia Tools and Applications, 83(3): 9441-9477. https://doi.org/10.1007/s11042-023-15054-0
[32] Lazuardi, M.R., Hadi, M.Z.S., Sudibyo, R.W. (2023). Driver drowsiness detection system using deep learning method to reduce risk accident. In 2023 International Electronics Symposium (IES), Denpasar, Indonesia, pp. 399-404. https://doi.org/10.1109/IES59143.2023.10242431