International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 14, No. 3, November 2025, pp. 766~784
ISSN: 2089-4864, DOI: 10.11591/ijres.v14.i3.pp766-784

Performance analysis of REST API in a real-time IoT-based vehicle monitoring system

Rizki Ananta Dwiyanto, Giva Andriana Mutiara, Marlindia Ike Sari
Department of Computer Technology, Faculty of Applied Science, Universitas Telkom, Bandung, Indonesia

Article Info

Article history:

Received Mar 17, 2025
Revised Jul 5, 2025
Accepted Oct 9, 2025

Keywords:

Internet of things
Load testing
Performance analysis
REST API
Vehicle monitoring system

ABSTRACT

This study studies the design and implementation of a REST API and its performance analysis for an internet of things (IoT)-based vehicles monitoring system. This system incorporates brake pad sensors, a tire pressure monitoring system (TPMS) for assessing tire pressure and temperature, light detection and ranging (LIDAR) for measuring tire thickness, and radio frequency identification (RFID) for tire identification. Data is gathered using an ESP32 microcontroller and transmitted in real-time to the server via a REST API over a wireless network. The JSON Web Token (JWT) authentication mechanism is employed to ensure data security. Testing indicates that this system has an average response time of 4–11 ms, with optimal performance recorded at 3.93 ms for the RFID sensor and peak performance at 9.19 ms for the LIDAR sensor. Load testing with 100 concurrent users demonstrates that the system maintains stability with a 100% data delivery success rate. Authentication testing demonstrates that the API is accessible solely with a valid token, hence preventing unauthorized access. This study's results demonstrate that integrating REST API with IoT monitoring systems facilitates real-time vehicle monitoring, enhances maintenance efficiency, and offers viable solutions for future predictive maintenance systems.

Corresponding Author:

Giva Andriana Mutiara
Department of Computer Technology, Faculty of Applied Science, Universitas Telkom
St. Telekomunikasi No. 1, Terusan Buah Batu, Bandung 40257, Indonesia
Email: givamz@telkomuniversity.ac.id

1. INTRODUCTION

The integration of a remote monitoring system (RMS) into an early warning system (EWS) has markedly improved early detection abilities, leading to greater safety and risk reduction. Through the use of cutting-edge sensors and data analysis, EWS effectively tracks essential parameters in real-time, allowing for proactive measures to address potential risks before they arise. This technology encompasses several essential components, including the incorporation of high-resolution sensors like fiber bragg grating (FBG) sensors, which are capable of capturing vital data from environmental or infrastructure systems. This data allows for precise forecasting of possible threats or system malfunctions, facilitating swift actions to mitigate more significant consequences [1].

Moreover, deep learning frameworks like Conv-LSTM are employed to analyze sensor data and predict the remaining useful life (RUL) of components, thereby facilitating improved maintenance strategies [2]. In the realm of battery health management, machine learning models are utilized to assess essential battery parameters to forecast potential failures, thereby guaranteeing the safety and reliability of electric vehicles throughout their operation [3]. In the realm of battery health management, machine learning models are utilized to assess essential battery parameters to forecast potential failures, thereby guaranteeing the

safety and reliability of electric vehicles throughout their operation [4]. Moreover, predictive maintenance facilitated by RMS can significantly lower vehicle maintenance expenses, thus enhancing operational efficiency within the electric vehicle sector [5]. Nonetheless, the broad acceptance of RMS continues to encounter various obstacles, such as significant upfront implementation expenses and the necessity for enhanced data security protocols. These challenges represent significant issues that need to be tackled for this technology to achieve broader implementation in the electric vehicle sector.

As the transition from gasoline-powered vehicles to automatic and electric vehicles accelerates, the demand for real-time monitoring systems is growing, highlighting the significance of monitoring technology. One of the parameters to be included on the dashboard of this monitoring system is the brake pads, along with the monitoring of tire parameters such as pressure, temperature, and tire thickness. The four parameters are linked to the ESP32 microcontroller, which then transmits data to a centralized database management system (MySQL) through the REST API. To ensure the protection of vehicle sensor data, the system employs the JSON Web Token (JWT) authentication method, which is accessible only after the user has successfully completed the Sign In process. Furthermore, a static token is implemented to enable the ESP32 to access the system seamlessly, bypassing the authentication process and facilitating the automatic and secure transmission of data.

The implementation of an object-relational mapping (ORM) framework, like Prisma, greatly enhances database management for developers by enabling interaction with the database through object-oriented programming principles. This method streamlines database interaction by substituting traditional SQL query writing with more user-friendly classes and methods, thus minimizing the intricacies of data management [6]. Prisma provides a seamless experience for database migration and is adept at managing intricate data structures, including information from vehicle sensors. This facilitates the smooth incorporation of data gathered by devices like the ESP32 into database tables, enhancing the overall efficiency of the system [7]. The primary benefits of ORM involve streamlining database interactions by utilizing objects and methods, thereby enhancing the ease and speed of data management [6].

In the realm of vehicles, ORMs facilitate the representation of sensor data, including tire pressure and brake conditions, as objects that can be directly manipulated at the application level. Prisma, as an example of an ORM tool, can automate the integration of sensor data into database tables, thereby accelerating data processing and analysis [7]. While ORMs provide numerous advantages, some developers argue that this method may lead to performance overhead, particularly in specific situations. Consequently, selecting ORM or alternative approaches must be customized to align with the specific requirements and scope of the project.

Node.js serves as a robust backend framework, crafted to enhance the creation of scalable and efficient web applications. The event-driven architecture and asynchronous programming model enable the handling of multiple connections at once, making it particularly suitable for environments with high traffic. The scalability is enhanced by a non-blocking I/O model, enabling Node.js to effectively manage substantial amounts of data and traffic [8].

Furthermore, the framework enables horizontal scaling, facilitating the distribution of workloads across various servers, which proves particularly beneficial for extensive applications [9]. In terms of security, Node.js provides safeguards against prevalent vulnerabilities like SQL injection and authentication issues. The framework is frequently integrated with tools like Passport.js and JWT to establish a dependable authentication system [10]. Node.js enhances development efficiency through its comprehensive documentation, tutorials, and robust community support, which greatly accelerates the application development process [9]. The implementation of JavaScript across both client and server sides facilitates code reuse, thereby enhancing the efficiency of the development process [8]. Nonetheless, despite its strengths in various areas, the single-threaded characteristic of Node.js may pose a performance limitation for applications that are CPU-intensive. In these situations, Node.js might not be the optimal selection, since resource-intensive tasks can hinder the performance of other processes [11].

This paper focuses on developing a website monitoring system designed to gather data from multiple vehicle sensors utilizing REST API. The outcomes of this system can aid in future advancements, allowing for enhanced development in predictive vehicle maintenance analysis. This study also results in a monitoring application that is readily available and offers advantages to drivers, enhancing their awareness of various driving safety factors. The paper's structure can be elaborated upon in the methodology section found in section 2. Section 3 presents the results and discussion. Ultimately, section 4 will provide an explanation of the conclusions and outline directions for future inquiry.

2. RESEARCH METHOD

This study employs an experimental and system development methodology to create and deploy an IoT-based monitoring system that incorporates REST API for tracking vehicle brake pads and tires. In the

initial stage, a literature review was conducted on an IoT-based vehicle monitoring system, focusing particularly on the monitoring of brake pads and tire conditions. Architecture for REST API and IoT communication, featuring the implementation of JWT authentication for enhanced security. ESP32 microcontroller technology, featuring the integration of pressure, temperature, and tire thickness sensors. Backend development utilizing Node.js, encompassing data management from sensors to the database through a REST API.

Following that, it proceeded to the system designe phase. At this stage, the monitoring system has been developed with various hardware and software components. The hardware components lack detailed explanation and are viewed merely as a collection of input parameters that transmit data to the dashboard. Utilization of the ESP32 microcontroller for processing sensor data. Incorporating wireless communication (Wi-Fi) for transmitting data to the server. In the meantime, the software architecture for backend development employs Node.js, utilizing REST API as the communication bridge between the ESP32 and the server. Database management employs MySQL. System security employs JWT authentication for users alongside static tokens for IoT devices. Development of the frontend for a web-based monitoring dashboard interface. The system architecture emphasizes the ESP32's role in transmitting sensor data to the REST API. A REST API utilizes MySQL for data storage, while the monitoring dashboard presents real-time information through tables and graphs.

Phase of executing the system setting up and configuring the ESP32 along with various sensors. Creation of a REST API backend utilizing Node.js and Express.js. Creation of the monitoring dashboard's user interface. The testing and evaluation stage is conducted to assess the functionality of each REST API feature, including data sending, authentication, and data retrieval. Evaluating the dashboard interface to confirm that data visualization operates effectively. Subsequently, performance evaluation of the system will include REST API latency testing to assess the response time from the ESP32 to the server. Testing for data stability to evaluate the reliability of sensor data delivery over a specified timeframe, along with security testing to confirm that the implementation of JWT authentication operates effectively.

Next is the data analysis and performance evaluation stage, where an assessment and discussion take place concerning the effectiveness of the REST API in managing requests from IoT devices. In conclusion, it is essential to analyze the outcomes of system testing and offer suggestions for future advancements, including the incorporation of machine learning for predictive maintenance.

2.1. Literature review

This chapter examines pertinent literature concerning EWS technology and IoT-based vehicle monitoring systems, with a particular focus on the monitoring of brake pads and tire conditions. REST API and IoT communication architecture, featuring the application of the JWT authentication method to enhance security. EWSs are progressively incorporating cutting-edge technologies like IoT and AI, greatly provide efficiency and security via real-time monitoring. The integration of IoT enables sensors to gather data in real-time, offering insights that facilitate predictive maintenance and enhance responses to potential risks or system failures [12]. The data undergoes analysis through AI algorithms to enhance the precision of early detection and refine preventive measures across different situations, including natural disasters, industrial equipment failures, or traffic accidents [13]. From an architectural framework perspective, choosing the appropriate software architecture is crucial for the success of IoT applications in EWSs. Methods like microservices and service-oriented architectures offer essential flexibility and scalability, vital for the support of IoT applications [14].

Conversely, client-server architecture proves to be highly efficient in facilitating communication between sensors and backend systems. This architecture facilitates effective data management and instantaneous updates, establishing it as a favored option for IoT-driven applications in EWSs. Nonetheless, although client-server architecture presents numerous benefits, decentralized architectures like peer-to-peer merit consideration as well. These architectures provide enhanced resilience and lower latency, adding significant value to the implementation of EWSs across diverse domains. Taking into account the requirements and scope of the application, the most suitable strategy can be adopted to enhance the efficiency and precision of early identification of possible risks.

The client-server model serves as a fundamental framework that facilitates multi-sensor applications by establishing the server as the central hub for data processing and storage. This architecture enables communication between edge devices, functioning as clients, and servers that may be located in the cloud or within on-premises data centers. This model enables the server to handle requests from numerous clients at the same time, ensuring efficient simultaneous access to resources [15]. Furthermore, the server is capable of handling substantial amounts of data produced by various sensors, guaranteeing both data integrity and availability [16].

The integration of edge devices involves connecting sensors to servers, facilitating data transmission and preprocessing, which enhances overall system efficiency. Frameworks like SEMAR are essential for enhancing the efficiency of edge devices through the implementation of organized initialization, service, and update phases [17]. Furthermore, the collaboration between cloud and edge devices enhances the capabilities of the client-server architecture via a cloud-edge framework that facilitates adaptable task migration between the two components. This facilitates enhanced response time and more efficient resource management [16]. This architecture additionally facilitates real-time data analysis, an essential component for informed decision-making in IoT applications [18]. Nonetheless, while presenting numerous benefits, the client-server architecture is not without its drawbacks, including issues related to latency and bandwidth, particularly in situations that demand real-time data processing. Integrating cloud and edge resources presents a viable approach to addressing these limitations, thereby enhancing overall system performance.

Figure 1 illustrates that the interaction between the client and server employs the standard HTTP protocol methods, including GET, POST, PUT, and DELETE commands. This architecture is frequently utilized due to its adaptability and straightforward implementation, facilitating effective communication among various layers in edge, fog, and cloud systems.

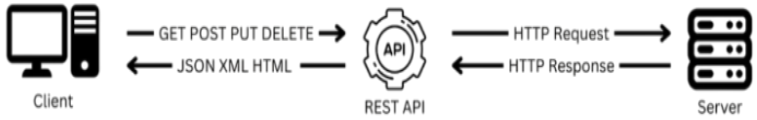

Figure 1. REST API on client-server architecture

In Figure 1, the REST API is crucial for facilitating communication between clients and servers, particularly in IoT systems that utilize microcontrollers like the ESP32. The stateless nature of REST API streamlines server design and enhances scalability, allowing for effective management of sensor data. Furthermore, REST API facilitates create, read, update, and delete (CRUD) operations, which are essential for efficient data management, particularly when handling sensor data that demands real-time updates and accessibility [19]. The efficiency of REST API performance has been demonstrated, showcasing an average response time of approximately 31 ms, which guarantees timely data processing [20].

Nonetheless, in spite of their numerous benefits, REST APIs encounter security issues. Approximately 35% of developers emphasized the importance of enhancing authentication and encryption techniques to achieve more robust data security in IoT applications. Enhanced security measures, including the implementation of OAuth, are strongly advised to ensure the confidentiality and integrity of data [19]. While REST APIs enjoy widespread popularity for their straightforwardness and efficiency, other communication protocols like gRPC and WebSockets may provide superior performance, particularly for applications that demand real-time interaction with minimal latency. Consequently, choosing the appropriate communication protocol should be customized to meet the unique requirements of the IoT application in development [21]. As illustrated in Figure 2, the REST API works in conjunction with the backend framework to organize sensor data systematically, facilitating effective and uniform data management throughout IoT systems.

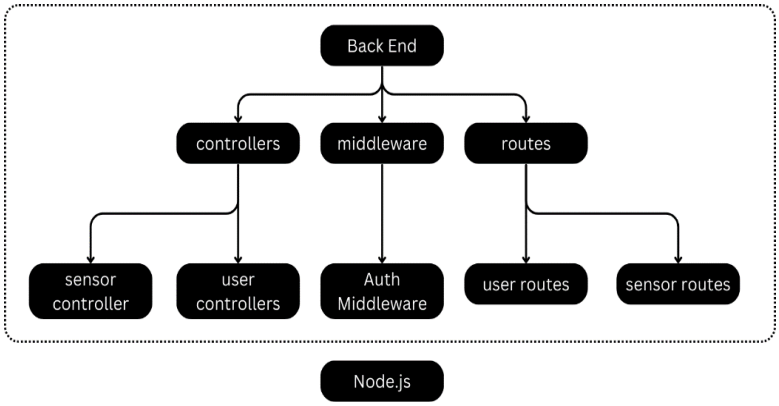

Figure 2. Backend architecture with Node.js and Prisma

The combination of Node.js with Prisma and MySQL for backend architecture in IoT systems presents several benefits regarding scalability, performance, and database management. Node.js, featuring a design that excels in managing concurrency, enables the simultaneous handling of numerous connections, a crucial capability for IoT applications that produce substantial data volumes [8].

The event-driven architecture of Node.js facilitates asynchronous operations, allowing for real-time data processing and notifications, a feature that is particularly advantageous in the realm of efficient sensor data monitoring and processing systems [8]. Streamlines the process of crafting SQL queries, thereby minimizing the potential for errors that can occur with manual SQL coding. Furthermore, Prisma streamlines the process of database migration, enabling developers to implement schema changes seamlessly without interrupting system operations [22]. MySQL serves as a data store, offering efficient and structured data management, which makes it a suitable option for IoT applications that demand reliability and efficiency in sensor data storage [23]. Furthermore, information housed in MySQL can be seamlessly aligned with visualization platforms like Grafana, enhancing the clarity of data analysis and oversight [22].

This combination of technologies presents numerous benefits; however, it is crucial to acknowledge potential challenges, including the learning curve linked to new technologies and the necessity for robust security measures to safeguard sensitive data from IoT devices. Consequently, it is essential for developers to implement suitable security measures to uphold the confidentiality and integrity of the data handled by these systems.

Figure 3 illustrates that the implementation of robust encryption and authentication measures is crucial for upholding the integrity and confidentiality of information obtained from IoT devices, thereby ensuring that the processed and stored data remains secure and safeguarded against potential threats.

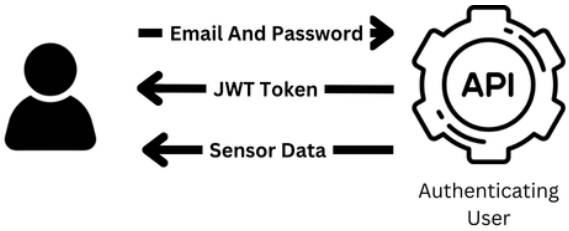

Figure 3. Data security with JWT

Data security in IoT systems, especially for vehicle monitoring, is critical given the sensitive nature of the data involved. Utilizing JWTs for user authentication and employing static tokens for device communication can significantly bolster security measures. JWTs offer a concise and secure method for transmitting information between parties, guaranteeing that only authenticated individuals can access sensitive data [24]. Furthermore, static tokens are essential for enabling secure communication between IoT devices and servers, thereby minimizing the risk of data interception during transmission [25].

Data encryption, employing methods like elliptic curve cryptography (ECC), is essential for safeguarding the integrity and confidentiality of information in IoT settings. The incorporation of blockchain technology significantly boosts security by offering a tamper-proof, decentralized structure for data transactions, guaranteeing that information remains unaltered without detection [25]. Nonetheless, in spite of notable advancements in enhancing security, weaknesses persist, especially in the areas of credential management and network assaults. This underscores the necessity for a comprehensive strategy regarding IoT security [26]. Future investigations should concentrate on creating scalable and energy-efficient solutions to tackle these challenges, while also guaranteeing strong data protection as the IoT ecosystem progresses [24]. Consequently, although JWTs and data encryption have significantly enhanced security, the ever-changing landscape of threats in IoT systems necessitates ongoing innovation and adaptation of security protocols to uphold data integrity and user trust.

The sensor data gathered is utilized for predictive analysis, enabling the identification of potential issues or patterns that can aid in improved decision-making and early damage prevention. In this context, the significance of data security is heightened, as predictive analysis depends on data that upholds its integrity and confidentiality.

Methods like clustering and deep learning are employed to recognize failure patterns and assess the remaining usable life (RUL) of the components [27], [28]. This method not only aids in minimizing the total maintenance expenses but also diminishes the likelihood of accidents by proactively tackling potential

failures [29]. Predictive maintenance allows vehicle operators to enhance maintenance schedules through predictive analytics, preventing severe failures and escalating repair expenses. This literature review demonstrates that an IoT-based monitoring system featuring REST API integration, sensor technology, and data security holds significant promise for use in contemporary vehicle monitoring. The forthcoming sub-chapter will detail the design employed in the development of this system.

2.2. Proposed system

This sub-chapter outlines the methodology employed, focusing on an experimental and development approach to create a real-time vehicle condition monitoring system. This system incorporates a range of sensors, including light detection and ranging (LIDAR) for tire thickness detection, TPMS for tire pressure monitoring, radio frequency identification (RFID) for vehicle identification, and brake pad sensors for assessing brake wear. Information gathered from the sensors is analyzed to offer understanding regarding the state of the vehicle. This information is accessible via a web-based interface, enabling remote monitoring of vehicles. This system is crafted to be adaptive and proficient in handling sensor data instantaneously, thereby enhancing the precision of decision-making processes.

The system provides a versatile and widespread monitoring solution through the use of wireless communication technology. Information gathered from sensors is transmitted to a server, where it is processed and stored in a database, ultimately being presented to users through clear reports or visualizations. This process guarantees efficient monitoring of any alterations in vehicle conditions, contributing to enhanced safety and performance of the vehicle. This method facilitates a more proactive management of vehicles, with the system tailored to accommodate various vehicle types and diverse monitoring requirements.

The client-server approach to IoT offers advantages in terms of management, data integrity, and system security. This architecture sets the server as a central hub, allowing for easier management and control of all IoT devices, increasing efficiency and structure in management processes. Centralizing data storage guarantees uniformity of information, minimizes the chances of data duplication or loss, and facilitates more thorough data analysis. This method enhances security by allowing for data encryption and enabling the monitoring of user access via a centralized security protocol. This system enables IoT devices to function with enhanced control, assured data access, and superior protection against security threats, positioning it as an excellent solution for extensive IoT systems. The system that has been proposed is illustrated in Figure 4.

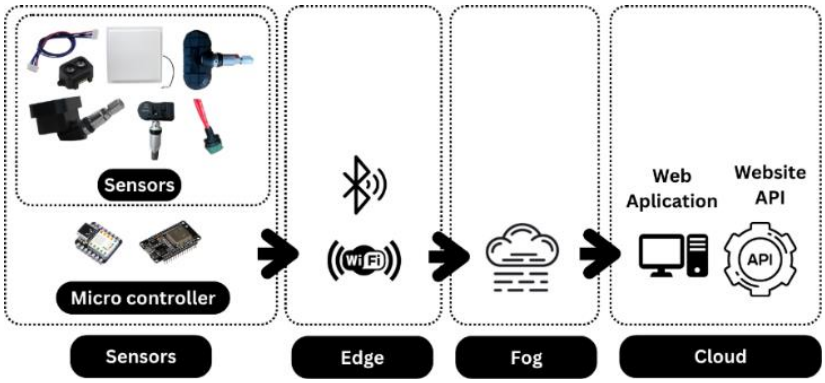

Figure 4. Proposed system

The observation environment comprises LIDAR, TPMS, RFID, and brake pad sensors that relay data in real time through HTTP. The REST API facilitates the management of sensor data by utilizing the POST method for database interactions and the GET method for presenting information on the website in JSON format. This system is engineered to maintain peak performance while managing numerous execution requests. The client-oriented process is segmented into data services and user services to ensure effective access to the database. Sensor data is captured along with their identities through the POST method and transmitted to the website using the GET method for clear and accessible information display.

The diagram depicting the system's layer-by-layer relationship is presented in Figure 5, showcasing the data flow from sensors to the user interface, thereby facilitating effective data monitoring and management.

This system features two distinct categories of administrators: Company Admin and Website Admin. The Company Admin oversees the configuration of APIs and the establishment of sensor connections with the server. They offer distinct endpoints for every sensor to maintain an organized data delivery process. Information collected from sensors is transmitted using the POST method and recorded in a MySQL database in accordance with the designated columns. Meanwhile, the Website Admin oversees the management and updates of the website interface that showcases sensor data in a graphical format.

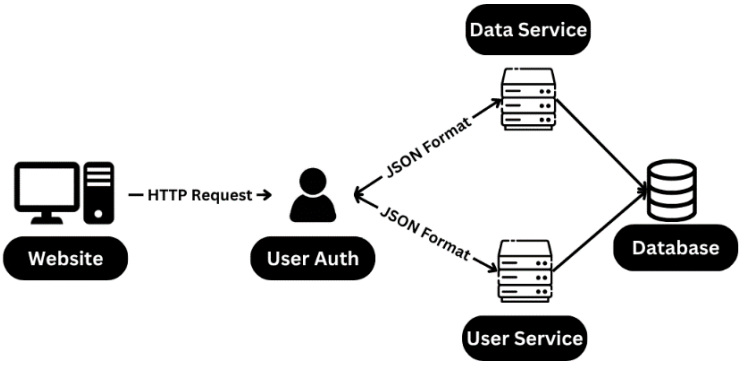

Figure 5. User-side client-based diagram

The GET and POST methods are employed to retrieve and send data to the API, ensuring synchronization between sensor data and the database. The API serves as a crucial link between the server and the client, facilitating seamless data communication and ensuring that updates are displayed in real-time.

Figure 6 illustrates the database tables associated with the four primary sensors (TPMS, LIDAR, RFID, and brake pads), along with supplementary tables for users, Prisma migrations, account sessions, and vehicle data (CangoLog). The user table contains information about users, whereas the ban_tpms, ban_lidar, rfid_log, and brake_pads tables document data from their respective sensors. The prisma_migrations table meticulously tracks data migrations, the sessions table efficiently manages account sessions, and the cangolog table accurately logs vehicle status. The GET method is employed for data retrieval to access the APIs of each sensor. Postman is a valuable tool for evaluating these requests.

Figure 6. Database

Figure 7 illustrates a react script that employs useEffect to asynchronously retrieve data from the API, save it in state, and present it in a graph on the website. The data refreshes every 5 seconds through the use of setInterval, facilitating continuous real-time monitoring. Using clearInterval for cleanup helps avoid memory leaks when the component is unmounted. All data undergoes conversion to JSON format prior to being presented in an accessible and user-friendly manner.

```
useEffect(() => {
  const fetchData = async () => {
    setKampasRemData(await getKampasRem()); //Fetch break pads data
    setTpmsData(await getBanTpms()); // Fetch Tpms data
    setLidarData(await getBanLidar()); // Fetch Lidar data
    setRfidData(await getRfidSensor()); // Fetch RFID data
  };

  const intervalId = setInterval(fetchData, 5000);
  fetchData();
  return () => clearInterval(intervalId);
}, []);
```

Figure 7. GET sensor data to website

The system architecture encompasses interactions among sensors, an API Gateway, data services, and databases. The diagram in Figure 8 illustrates the communication flow within the REST API system, detailing the journey from data delivery by sensors to the subsequent storage and processing in the backend.

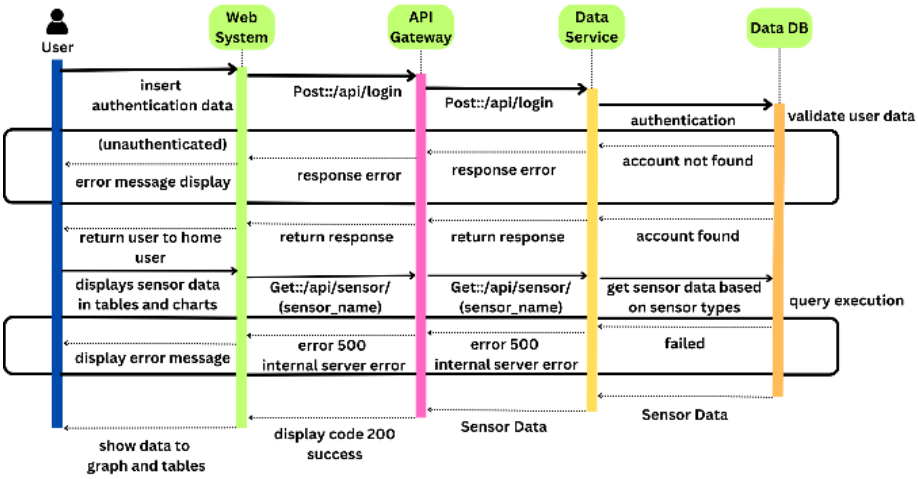

Figure 8. Sequence diagram

Figure 9 presents a use case diagram that demonstrates the interaction of users with the system for accessing sensor data. Meanwhile, Figure 10 presents an application flowchart that outlines the user authentication process utilizing JWT, data retrieval from the API, and the mechanisms for handling errors. This method guarantees precise, effective, and safe monitoring of vehicles.

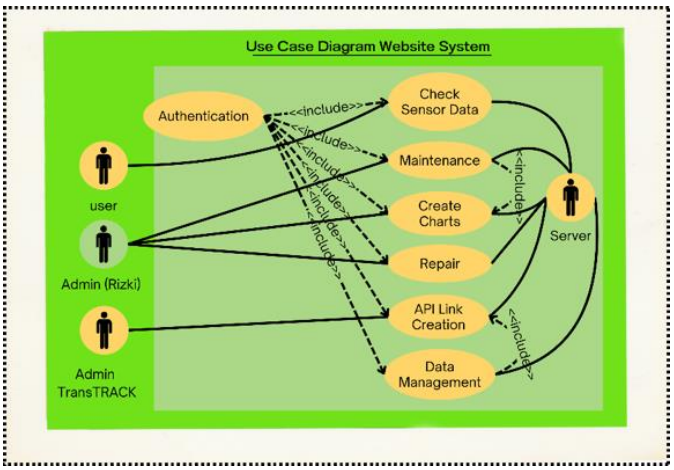

Figure 9. Use case diagram

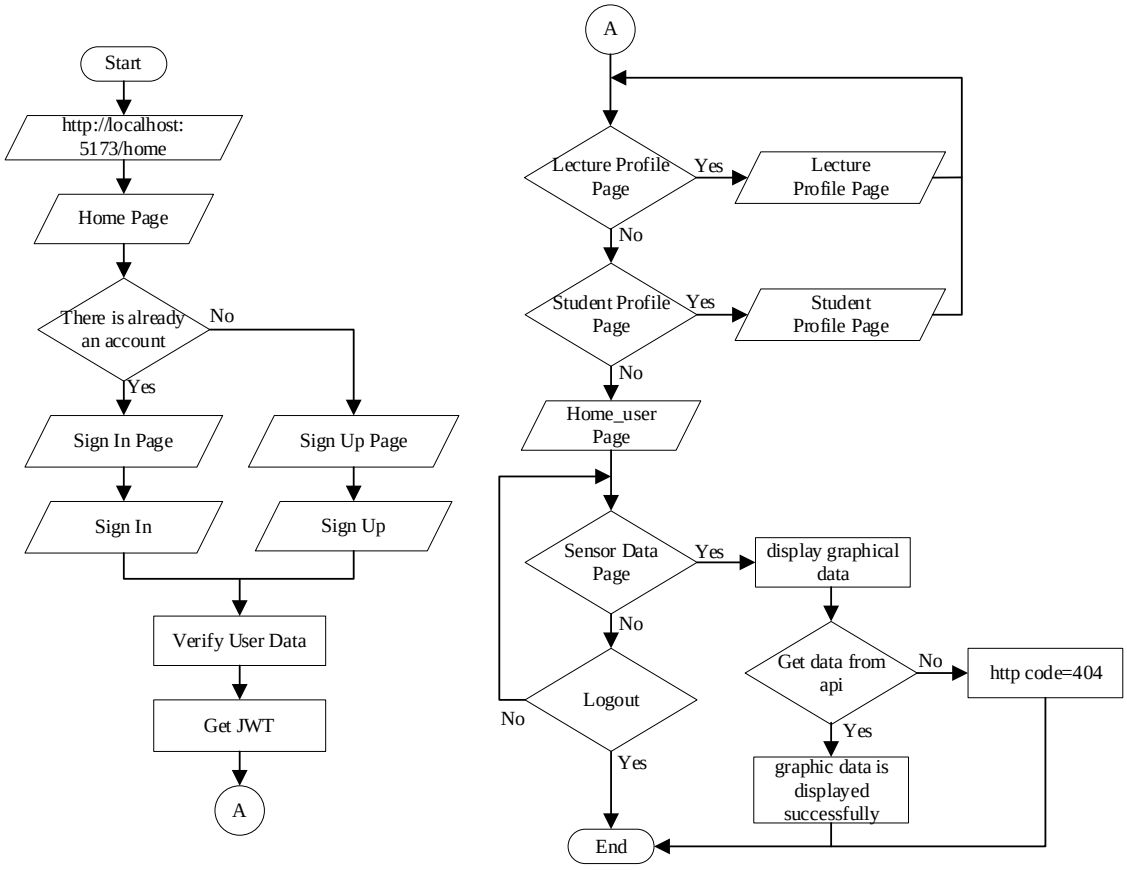

Figure 10. Flowchart system

2.3. Testing scenario

The subsequent phase entails choosing a test scenario to assess the application built on REST API principles. This seeks to guarantee that the API operates correctly, securely, and in line with the anticipated functionality. The design of client-server architecture presents various vulnerabilities, such as reliance on the server, risks of overload, and challenges related to network latency. API testing encompasses several categories, including functionality testing, performance testing, and security testing.

The first phase of testing includes manual verification of the API endpoint's functionality prior to moving on to automated testing. One approach employed is to make use of Postman. In this scenario, Postman sends an HTTP request to the API to analyze the response and conveniently adjust the request parameters. Subsequently, a GET and POST data request is initiated to one of the endpoints. Once the data is collected, an HTTP response will be produced with a status code of 200, signifying "Client Success." Alongside the explanation of the 200 code, Postman offers details like the time taken for data transmission, the size of the data in bytes, and additional status codes including 404 for "Not Found" and 400 for "Bad Request."

This test utilizes data derived from sensors, encompassing details like tire thickness, temperature, air pressure, and tire ID sourced from RFID sensors. The data will be transmitted to the server through the API endpoint utilizing the POST method, followed by verification with the GET method to confirm that the data is properly stored and accessible. Tables 1 to 5 present the POST and GET scenarios executed on sensor data, encompassing information such as tire thickness, air pressure, temperature, and the tire ID linked to each sensor. This test will assess the functionality of all sensors to verify the system's integration and precision in handling data in real-time. Table 6 provides an overview of the various sensors, devices, protocols, and APIs utilized.

Table 1. POST data using Postman

Id	Sensor id	Pressure (kPa)	Temperature (°C)
Auto increment	1	0	12

Table 2. GET data for canvas parameters using Postman

Id	Canvas thickness (%)	Timestamp
5370	100	2025-03-07T06:45:19.000Z
5369	100	2025-01-15T15:01:24.000Z
5368	100	2025-01-15T15:01:20.000Z
5367	100	2025-01-15T15:01:17.000Z
5366	100	2025-01-15T15:01:14.000Z

Table 3. GET data TPMS using Postman

Id	Sensor_id	Pressure (kPa)	Temperature (°C)	Timestamp
4705	1	0	28	2025-03-07T02:18:37.000Z
4704	1	0	13	2025-03-07T02:16:53.000Z
4703	1	0	12	2025-01-15T15:01:25.000Z
4702	1	0	25	2025-01-15T15:01:21.000Z
4701	1	0	29	2025-01-15T15:01:18.000Z

Table 4. GET data LIDAR using Postman

Id	Sensor_id	Thickness (cm)	Description	Timestamp
13546	2	4	Good	2025-01-15T15:07:02.000Z
13544	1	4	Good	2025-01-15T15:06:51.000Z
13545	1	4	Good	2025-01-15T15:06:51.000Z
13543	1	4	Good	2025-01-15T15:06:50.000Z
13542	1	4	Good	2025-01-15T15:06:50.000Z

Table 5. GET data RFID using Postman

Id	Tag_ID	Detected_at
980	010CE28014700000219059E891038	2025-01-15T07:52:59.878Z
979	010CE28014700000219059E891038	2025-01-15T07:52:59.045Z
978	010CE28014700000219059E891038	2025-01-15T07:52:58.234Z
977	010CE28014700000219059E891038	2025-01-15T07:52:57.408Z

Table 6. GET data TPMS using Postman

Sensor types	Device	Protocol	API
MS580314BA	TPMS	SPI/I2C	http://localhost:5001/api/sensor/ban/tpms
TF Mini	LIDAR	UART	http://localhost:5001/api/sensor/ban/lidar
AWG 18	Brake pad (*kampas rem*)	GPIO	http://localhost:5001/api/sensor/kampas_rem
Ommni directional UHF	RFID	ISO 18000-6C	http://localhost:5001/api/sensor/rfid

In this project, API testing is carried out through various scenarios to verify that the system's performance, reliability, security, and functionality align with the anticipated specifications. The initial scenario involves conducting manual testing to confirm the functionality of the API endpoints. This testing is conducted with tools like Postman to verify that each API endpoint responds as intended. During this evaluation, GET and POST requests are dispatched to designated endpoints, including /api/sensor/, along with the necessary parameters. The response from the API is analyzed, noting the status code (200 indicates a successful request, 400 signifies an invalid request, and 404 means the endpoint is not found).

This testing additionally confirms the structure of the response data, ensuring it aligns with the anticipated JSON format. The second scenario involves performance testing, which focuses on assessing the response time and stability of the API under specific load conditions. This testing employs tools like JMeter, specifically crafted for conducting load and stress assessments. The system undergoes rigorous testing by dispatching hundreds to thousands of requests at the same time to evaluate various parameters, including latency and potential failure points. The calculation of latency is derived from the average response time of the API to incoming requests. This testing aids in pinpointing API capacity thresholds and evaluating system performance when the load exceeds its typical limits.

The third scenario involves load testing, aimed at verifying that the API can manage numerous user queries at the same time without suffering from notable performance decline. This test involves evaluating the system with a substantial volume of query requests to analyze fluctuations in response time and identify any bottlenecks or vulnerabilities that may hinder API performance. The findings from this test illuminate the vulnerabilities within the system and offer direction for enhancement.

The fourth scenario encompasses security testing, designed to uncover vulnerabilities that could be leveraged by unauthorized individuals. Security testing is carried out with Postman, focusing on the process

of attempting to send data via a POST request while omitting a valid authentication token. The token utilized in the API may either be a static token or a dynamic token, which can solely be acquired following the login procedure. In this test, simulations for data transmission are conducted both with and without the appropriate token to verify that the API endpoint is exclusively accessible to users possessing valid credentials.

The final assessment involves automating API testing and overseeing the dashboard. In this scenario, testing is carried out with the Postman Collection Runner, enabling the automated execution of functional tests across different API endpoints. Test results are documented in the monitoring dashboard to capture essential metrics including the number of queries, response time, success rate, and more. This series of tests guarantees that the API operates at peak performance while remaining secure against threats, ensuring its reliability in managing data requests across diverse conditions.

3. RESULTS AND DISCUSSION

3.1. Functional performance testing

As detailed in the preceding chapter, the initial assessment involves conducting functional testing using Postman. The results of the test are presented in Table 7. According to Table 7, manual testing with the Postman tool on multiple sensors operates effectively. Four types of sensors have been evaluated, each demonstrating an average response time for the GET and POST methods. This test permits two HTTP methods on the API: GET and POST. The GET method serves to present sensor data from the database on the dashboard, whereas the POST method is utilized for inputting sensor data into the database. The test results indicate that the average response time for the GET method across all sensors is under 10 milliseconds, with the minimum response time recorded at 4 milliseconds for both the brake pad and RFID sensors.

Table 7. Client-side test results for Postman tools

No	Sensor	Average GET response time (ms)	Average POST response time (ms)
1	Brake pad	4	11
2	TPMS	5	7
3	RFID	4	10
4	LIDAR	7	10

The average response time for the POST method ranges from 7 to 11 milliseconds, with the brake pad sensor recording the longest response time at 11 milliseconds. This indicates that the process of retrieving data (GET) is more efficient than the process of sending data (POST). The variation in response time is affected by the volume of data presented and the payload transmitted. An increase in the number of payloads or the complexity of the parameters leads to an extended server processing time. Nonetheless, the average response time recorded in this test indicates that the API manages requests with great proficiency and effectiveness.

3.2. Performance testing

In the second scenario, Figure 11 illustrates the test results. The purpose of performance testing is to assess the response time and stability of the API under specific load conditions utilizing JMeter. Evaluations are currently being conducted on the client side.

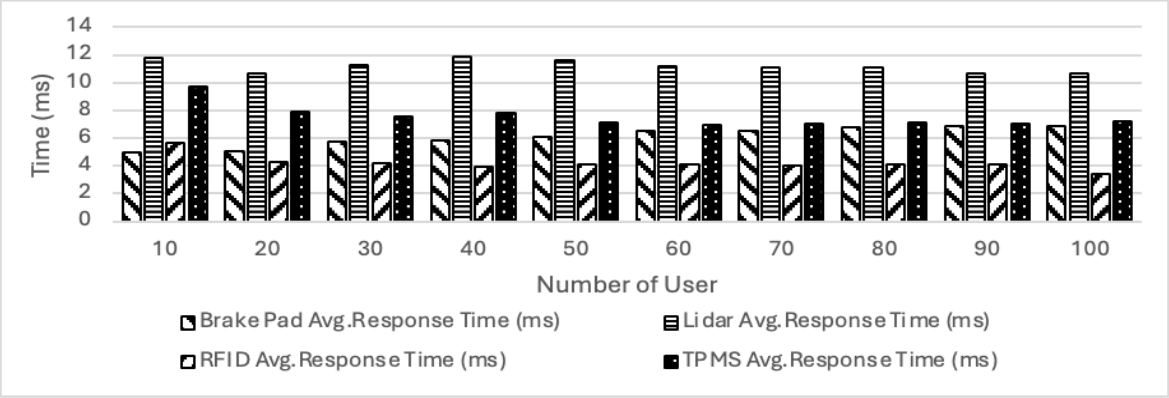

Figure 11. Client-side performance result

An evaluation was carried out to assess the effectiveness of the proposed system on several sensors, as indicated by the performance testing table image. Testing was carried out utilizing Apache JMeter, employing test parameters that included 5 fixed iterations and a gradual increase in the user count from 10 to 100. The test results indicate that the average response time and latency fluctuate based on the user count and the sensor type. The brake pad sensor demonstrates a consistent average response time and latency, remaining stable under 15 milliseconds up to 100 users. The LIDAR sensor exhibits greater latency compared to other sensors across all user points, whereas the RFID sensor demonstrates the most stable and low response time, remaining under 10 milliseconds in the majority of scenarios. Overall, these findings suggest that the increase in user numbers affects the system's response time and latency, although variations among sensors may arise from the intricacies of data processing or differing payload sizes. This system has demonstrated the capability to manage loads of up to 100 users without exhibiting notable performance decline. Figure 12 illustrates that the data delivery success rate to the client consistently achieves 100% while handling 100 threads.

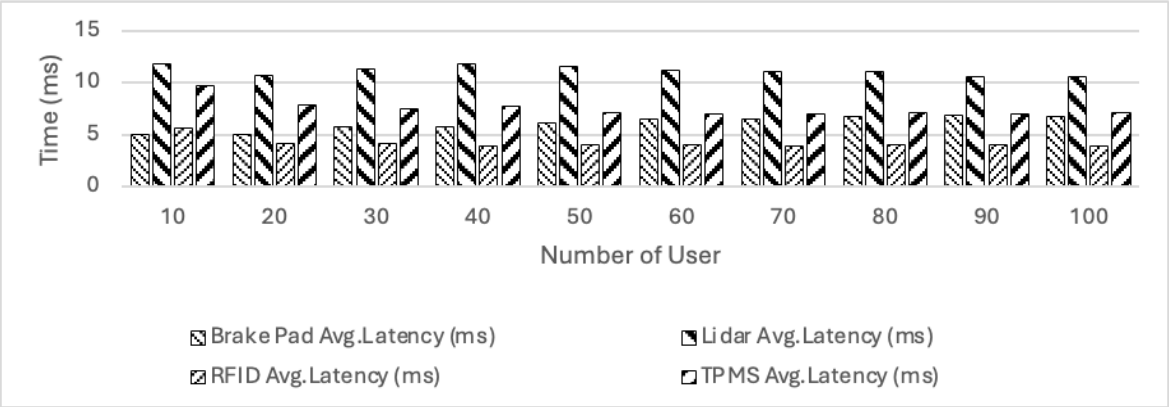

Figure 12. Client-side performance result for success rate

3.3. Load testing performance

A load test was performed for a duration of 5 minutes using 100 virtual users to evaluate the performance of the API, as illustrated in Figure 13. The /sensor/ban/lidar endpoint exhibited a higher response time, experiencing fluctuations of up to 15 ms, whereas the /sensor/rfid and /sensor/kampas_rem endpoints maintained a more stable response in the range of 5-7 ms. In summary, the system demonstrates consistent reliability, exhibiting no notable errors or spikes, even under increased load conditions.

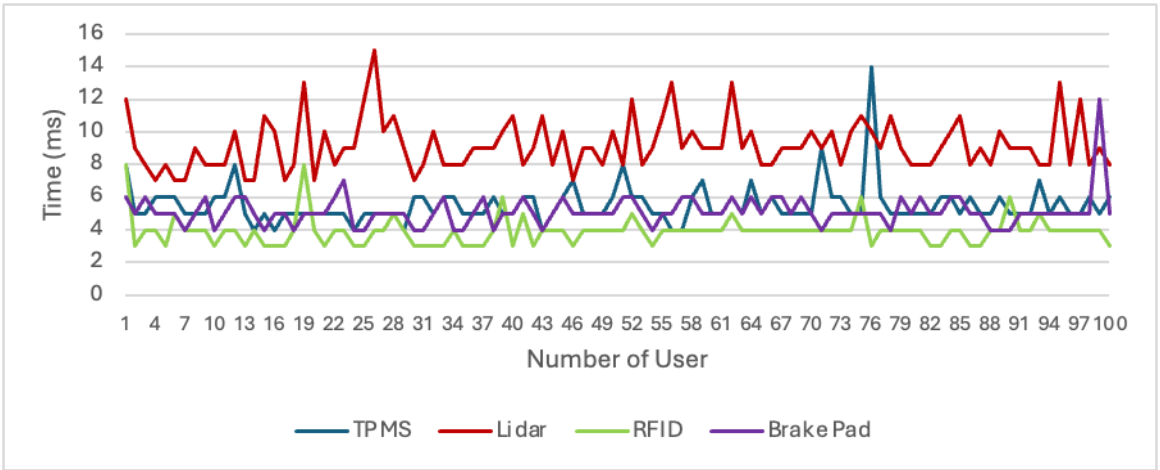

Figure 13. Load testing

In the meantime, Table 8 presents the detailed test outcomes for each sensor associated with the GET and POST methods. The test results indicate that the LIDAR sensor exhibits the highest average response time for both methods, recorded at 9.19 ms, with a maximum response time of 15 ms and a minimum response time of 7 ms. The RFID sensor demonstrates optimal performance, achieving the lowest average response time for both the GET and POST methods at 3.93 ms. It records a maximum response time of just 8 ms and a minimum of 3 ms. Additional sensors, including TPMS and brake pads, exhibit average response times of approximately 5.52 ms and 5.14 ms, respectively, while their maximum response times can peak at 14 ms and 12 ms, respectively. The test results indicate that a bottleneck arises in the LIDAR sensor during data retrieval via the GET method, particularly when the user count hits maximum capacity. In these circumstances, the peak response time may extend to 15 ms. The findings demonstrate that the server continues to manage requests effectively, despite the discrepancies in response times among the sensors.

Table 8. Server-side response time load test results

Sensor	Get			Post		
	AVG (ms)	Max (ms)	Min (ms)	AVG (ms)	Max (ms)	Min (ms)
TPMS	5.52	14	4	5.51	14	4
LIDAR	9.19	15	7	9.19	15	7
RFID	3.93	8	3	3.93	8	3
Canvas	5.14	12	4	5.14	12	4

3.4. Security performance system

The next assessment, the fourth assessment, is security testing. Security testing seeks to identify deficiencies or vulnerabilities in the API that may be exploited by unauthorized entities. The initial stage of security testing involves the authentication and authorization process, the steps of which are illustrated in Figure 14.

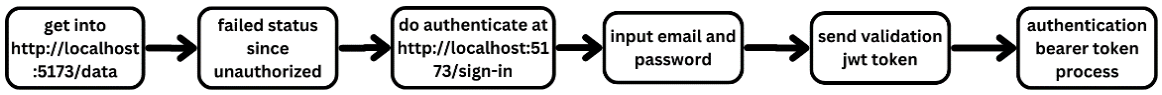

Figure 14. Authentication and authorization process

Figure 14 illustrates the process of obtaining data from https://api.stas-rg.com/sensor, which is unsuccessful because an authentication token is missing. The authentication procedure is accessible at https://api.stas-rg.com/auth/signin. The authentication process requires the user to input the correct email and password. Upon successful entry, a validated code (JWT) is generated, which remains valid for one hour. If this token has been acquired, Postman is capable of accepting input in the Authentication - Bearer Token section. Access to the system is denied to users who enter an incorrect email address or password. Upon entering the correct email and password, access to the previously restricted webpage is granted after the token is provided. The upcoming phase of evaluation involves utilizing Postman for basic security assessments. Testing is conducted to verify that the API is protected against unauthorized access by assessing its responses to various authentication scenarios. This test involves executing multiple token validation scenarios to assess the security mechanism of the API.

In the initial test, a request is dispatched to the API omitting the authentication token from the Authorization header. The findings from this test indicate that the API delivers an error message stating "Token not provided," accompanied by a status code of 401 Unauthorized. This suggests that the API is securely safeguarded and restricts access to authenticated users only. Figure 15 illustrates the API response for the test conducted without an authentication token.

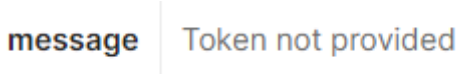

Figure 15. Security testing without token

In the second test, a request is dispatched utilizing an invalid or incorrect authentication token. A random token is utilized, which is placed within the Authorization header. The API response indicates an error message stating, "Invalid or expired token," accompanied by a status code of 403 Forbidden. This finding indicates that the API is capable of identifying invalid tokens and restricting access to secured resources. Figure 16 presents the outcomes of security testing conducted with an invalid token. The results from both tests indicate that the API security mechanism has been effectively implemented to thwart unauthorized access. The API is capable of differentiating between scenarios involving no token, a valid token, and an invalid token, delivering a suitable response for each case. These testing steps are crucial for confirming that the API is sufficiently safeguarded against unauthorized access, a key element in application security testing.

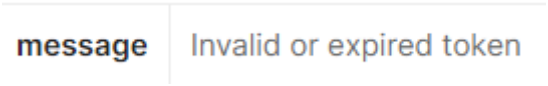

Figure 16. Security testing using wrong token

3.5. API dashboard monitoring testing

The final assessment involves testing the API automation and monitoring dashboard. This testing aims to reproduce functional testing using the Postman Collection Runner, which automates test generation through its features. To execute the API automation using the Collection Runner in Postman, the first step is to set up the API Collection (pre-request) to automate the sensor type and sensor ID. The SENSOR_NAME variable is designated to hold the type of sensor, whereas the SENSOR_URL variable is intended to store the URL associated with each sensor. These variables will serve to enhance dynamic testing. Figure 17 provides an illustration of the script.

Figure 17. Testing script on Postman collection runner

In Figure 17, the script executed for automation testing sequentially evaluates each sensor to confirm that the response code received is 200, indicating successful input and retrieval of sensor data from the database, while also verifying whether the sensor type associated with the URL is consistent. Figure 18 illustrates the outcomes of the GET method test. The figure illustrates that the total execution time for the API using the GET method is 25 seconds, accompanied by an average response time of 4 milliseconds. The POST method test is illustrated in Figure 19. The total execution duration of the API utilizing the POST method is 25 seconds, accompanied by an average response time of 5 milliseconds.

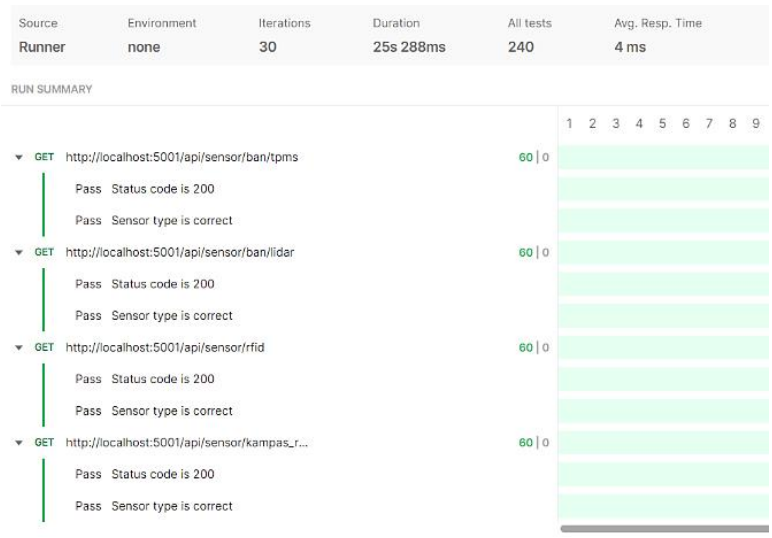

Figure 18. GET method testing

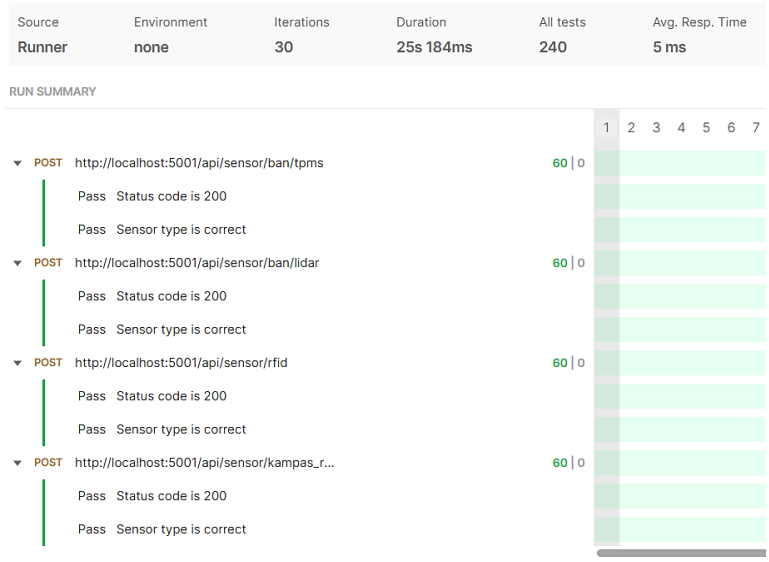

Figure 19. POST method testing

The results of the POST and GET method tests indicate that the dashboard display effectively allows for proper observation and monitoring of the data, as illustrated in Figure 20. Presenting the outcomes of the data dashboard that showcases the findings from the observations of the TPMS, LIDAR, and brake pad sensors through graphical representations.

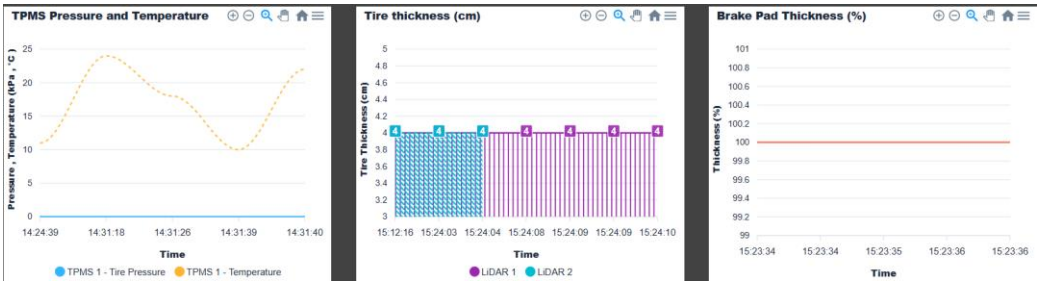

Figure 20. Dashboard testing TPMS, tire thickness, and brake pad

This study develops a vehicle monitoring website. This study presents a vehicle monitoring website that incorporates sensor integration and a REST API to effectively manage and display real-time data on vehicle component conditions. This system incorporates four primary sensors and utilizes 14 APIs, achieving an average response time ranging from 4 to 11 milliseconds. Testing indicates that all sensors effectively transmit data with precision, even in environments lacking Wi-Fi through Bluetooth. Conducting load testing with 100 users demonstrates consistent latency and response times, attributed to the system's low complexity. In alternative scenario evaluations, response times vary between 3.93 and 9.19 milliseconds, reaching a maximum of 15 seconds under full load conditions. Regarding security, authentication testing demonstrates that the system is capable of denying access in the absence of a token or when presented with an invalid token. While no significant security flaws were identified, it is advisable to implement OAuth 2.0 along with a token refresh mechanism to enhance data security. Ultimately, automated testing with Postman Collection Runner (PCR) facilitates efficient API validation, enabling developers to oversee and assess API performance with greater ease.

3.5. Discussion

The test results indicate that the developed IoT-based REST API system demonstrates stable performance when managing up to 100 simultaneous users, with response times varying from 3.93 to 9.19 ms under normal conditions. Load testing indicates that the peak response time hits 15 ms in scenarios with a significant user load, yet remains within acceptable thresholds for real-time monitoring applications. Furthermore, security testing demonstrates that the JWT-based authentication mechanism has been effectively implemented, safeguarding against unauthorized access to the vehicle monitoring system.

This study aligns with earlier investigations that have examined the application of REST APIs in monitoring systems based on IoT technology. For instance, a study conducted by [20] indicates that REST API exhibits an average response time of 31 ms in a web service-based system, which remains higher than the system developed in this investigation, where the response time ranges from 3.93 to 9.19 ms. A recent investigation by [19] emphasized that REST APIs are favored for their stateless characteristics, yet encounter issues related to latency unless optimized through caching and load balancing techniques.

From an authentication and security perspective, previous findings by [25] indicated that OAuth 2.0 and token refresh mechanisms are more advisable than static JWTs, as JWTs that are not updated regularly have a higher risk of token theft. This indicates that while the implemented system utilizes JWT, it is essential to explore the potential of OAuth 2.0 or blockchain-based authentication to enhance system security moving forward. Furthermore, regarding IoT-based vehicle monitoring, a study conducted by [29] on predictive maintenance in IoT-monitored systems indicates that employing machine learning for analyzing component wear patterns can enhance the efficiency of vehicle maintenance systems. The system created in this study continues to emphasize real-time monitoring; however, the integration of predictive maintenance models could represent the next advancement in enhancing system reliability.

The findings of this study carry substantial importance for the automotive sector and IoT systems. Leveraging REST APIs in vehicle monitoring systems allows for real-time tracking of vehicle conditions, enhances driving safety, and supports predictive maintenance strategies aimed at minimizing the likelihood of component failure. This system offers greater automation and accessibility when contrasted with traditional vehicle monitoring techniques that depend on manual diagnosis or regular inspections. Furthermore, the effective combination of ESP32, Node.js, Prisma ORM, and MySQL in this study demonstrates that IoT-based microcontrollers can seamlessly integrate with cloud-based systems, paving the way for advancements in applications like fleet management, electric vehicle monitoring, and AI-driven maintenance systems.

While the test results indicate strong performance, it is important to acknowledge certain limitations that warrant consideration. A significant limitation is the dependence on Wi-Fi connectivity, which may lead to communication disruptions if the network is unreliable. Options like LoRa, NB-IoT, or 5G may be explored to enhance connectivity. Moreover, the test scale remains confined to 100 users, which means that the system's performance in more extensive scenarios, like thousands of connected IoT devices, has yet to be thoroughly assessed. The test results indicate that the LIDAR sensor exhibits the fastest response time among the various sensors, highlighting the necessity for enhancements in the processing of more intricate sensor data. Furthermore, while JWT authentication has been put in place, the system currently lacks a token refresh mechanism or OAuth 2.0, which could enhance data security against threats like token theft or replay attacks.

To enhance the effectiveness and scalability of this system, future development directions may involve the integration of machine learning to bolster predictive maintenance capabilities, by examining brake pad and tire wear patterns derived from historical data. Enhancing communication infrastructure through the implementation of 5G or LoRaWAN can provide more stable connectivity than traditional Wi-Fi. Moreover, enhancing the REST API through caching with Redis can significantly decrease latency and accelerate the retrieval of sensor data. Creating a mobile application as an alternative monitoring interface

can enhance user accessibility and enable remote vehicle monitoring. When considering security measures, the implementation of OAuth 2.0 and refresh tokens represents a crucial advancement in safeguarding user data. Additionally, investigating the potential of blockchain technology for securing data transactions could serve as a sustainable approach to thwarting manipulation or unauthorized access to sensor information.

Overall, this study has effectively developed an IoT-based REST API for vehicle monitoring, demonstrating improved response times in comparison to various earlier studies. Furthermore, testing indicates that this system is capable of managing 100 concurrent users while achieving a 100% success rate in data delivery. Nonetheless, for large-scale applications, it is essential to focus on performance optimization, enhance security measures, and incorporate machine learning for predictive maintenance. With further development, this system could emerge as the primary solution in vehicle health monitoring technology, aiding the automotive sector in creating smarter vehicles that prioritize efficiency and safety.

4. CONCLUSION

This study has effectively shown that incorporating REST API into an IoT-based vehicle monitoring system offers a robust solution for real-time monitoring of vehicle conditions. This system integrates an ESP32 as the primary microcontroller, utilizing TPMS sensors for monitoring tire pressure and temperature, LIDAR for assessing tire thickness, RFID for tire identification, and brake pads. It effectively gathers data and transmits it to the server through a REST API using wireless connectivity. The conducted tests revealed that the system maintains an average response time ranging from 3.93 to 9.19 milliseconds, achieving a data delivery success rate of 100%, even under conditions involving 100 simultaneous users. Thus, the development of a website monitoring system designed to collect data from multiple vehicle sensors via REST API was successful, and it can serve as the basis for future predictive vehicle maintenance research.

The use of JWT for authentication has demonstrated effectiveness in safeguarding sensor data from unauthorized access. Nonetheless, this study also uncovered various obstacles, particularly regarding reliance on Wi-Fi connectivity, which may lead to data loss if the network experiences instability. Furthermore, while the system has undergone testing with 100 users, its ability to scale to a larger user base requires additional assessment. The test revealed that the LIDAR sensor exhibits a higher response time relative to other sensors, highlighting the necessity for additional optimization in handling more intricate sensor data.

The implications of this study are extensive, particularly for the automotive sector that is progressively embracing IoT-based technologies. This system has the potential to serve as a foundation for advancing vehicle monitoring solutions, including fleet management, electric vehicle oversight, and predictive maintenance analytics. The findings of this study demonstrate that REST API serves as a dependable, adaptable, and scalable option for vehicle monitoring systems, enabling users to retrieve data with increased speed and precision.

While it has demonstrated encouraging outcomes, there remains potential for additional advancement in the future. One of the primary areas for enhancement is the implementation of more reliable communication technologies, like 5G or LoRa, to address the shortcomings of Wi-Fi in data transmission. Furthermore, employing machine learning for predictive maintenance enables the analysis of wear patterns in vehicle components, facilitating early warnings of possible damage. Regarding security, implementing OAuth 2.0 and refresh tokens can enhance the protection of sensor data from cyber threats, while investigating blockchain technology can offer improved transparency and integrity in managing sensor data.

This study demonstrates that REST API serves as a practical and effective solution for vehicle monitoring systems based on IoT technology. Through continued advancements in network communication, data security, and predictive analysis, this system has the potential to evolve into a more intelligent and adaptive vehicle monitoring platform, paving the way for a safer, more efficient, and more sustainable vehicle ecosystem in the future

ACKNOWLEDGEMENTS

We would like to express our gratitude to thank the Directorate of Research and Community Service (PPM) of Telkom University and CoE Smart Technology and Applied Sciences (STAS) of Telkom University, Bandung, Indonesia for their invaluable support in this research endeavor.

REFERENCES

[1] S. H. Grandhi, H. M. Al-Jawahry, D. S, B. V. Kumar, and M. K. Padhi, "A Quantum Variational Classifier for Predictive Maintenance and Monitoring of Battery Health in Electric Vehicles," in *2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS)*, Hassan, India, Aug. 2024, pp. 1–4, doi: 10.1109/iacis61494.2024.10721715.

[2] W. Li, Z. Zhang, Y. Jiang, and K. Wang, "Prediction and Degradation Analysis Revolutionizing Electric Vehicle Reducer

Prognostics: An Advanced Deep Learning Framework for Accurate RUL Prediction and Degradation Analysis," in *2024 6th International Conference on Internet of Things, Automation and Artificial Intelligence (IoTAAI)*, Guangzhou, China, Jul. 2024, pp. 58–64, doi: 10.1109/iotaai62601.2024.10692468.

[3] S. M. *et al.*, "IoT-based Battery Health Management System for Electric Vehicles: A Predictive Approach," in *2024 IEEE 4th International Conference on Sustainable Energy and Future Electric Transportation (SEFET)*, Hyderabad, India, Jul. 2024, pp. 1–6, doi: 10.1109/sefet61574.2024.10718077.

[4] D. L. Sampath, "Harnessing AI and Predictive Analytics: Transforming the Electric Vehicle Market in India," *International Journal of Scientific Research in Engineering and Management*, vol. 08, no. 008, pp. 1–7, Sep. 2024, doi: 10.55041/ijsrem37444.

[5] I. Kamusiime, "A remote vehicle health monitoring system: a case study of the kayoola electric vehicles," M.S. thesis, The Nelson Mandela African Institution of Science and Technology, Arusha, Tanzania, 2023, doi: 10.58694/20.500.12479/2586.

[6] J. Juneau and T. Telang, "Object-Relational Mapping," in *Java EE to Jakarta EE 10 Recipes*, Apress, 2022, pp. 333–377, doi: 10.1007/978-1-4842-8079-9_7.

[7] N. Biswas, "Creating an App with Prisma," in *Practical GraphQL*, Apress, 2023, pp. 163–220, doi: 10.1007/978-1-4842-9621-9_5.

[8] R. K. Kannan, M. A. K. T., S. Vairachilai, R. Vijayalakshmi, "NodeJS and Postman for Serverless Computing," in *Serverless Computing Concepts, Technology and Architecture*, IGI Global, 2024, pp. 195–204, doi: 10.4018/979-8-3693-1682-5.ch012.

[9] A. Dubey, G. S. Chauhan, A. Dubey, J. Singh, and P. Girdhar, "Customized Framework for Backend Using Node JS," in *2023 International Conference on Sustainable Emerging Innovations in Engineering and Technology (ICSEIET)*, Ghaziabad, India, Sep. 2023, pp. 01–05, doi: 10.1109/icseiet58677.2023.10303462.

[10] N. Subbulakshmi, S. A. Begum, R. Venkata Vignesh, and R. Chandru, "Asynchronous Event Driven Brain Teaser Using Node.js," in *2024 5th International Conference on Electronics and Sustainable Communication Systems (ICESC)*, Coimbatore, India, Aug. 2024, pp. 113–118, doi: 10.1109/icesc60852.2024.10689905.

[11] D. Otynshin, "Optimizing Node.js application performance through main thread offloading," *International Journal of Information and Communication Technologies*, vol. 4, no. 2(14), pp. 82–93, Jan. 2024, doi: 10.54309/IJICT.2023.14.2.008.

[12] N. Singh, Shikha, A. Gola, and Sidharth, "Empowering tomorrow's mobility: innovations in electric vehicle technology with iot and ai integration," in *Advancing Innovation in Smart Systems, Energy, Materials, and Manufacturing: Unleashing the Potential of IoT, AI, and Edge Intelligence*, Iterative International Publishers, Selfypage Developers Pvt Ltd, 2024, pp. 141–159, doi: 10.58532/nbennuraich8.

[13] R. K. P. and B. Amutha, "Literature Review of Data-Driven Strategies for the Sustainable Growth of Electric Vehicles in Cities," in *2024 1st International Conference on Trends in Engineering Systems and Technologies (ICTEST)*, Kochi, India, Apr. 2024, pp. 1–5, doi: 10.1109/ictest60614.2024.10576091.

[14] V. M. Macharia, V. K. Garg, and D. Kumar, "A review of electric vehicle technology: Architectures, battery technology and its management system, relevant standards, application of artificial intelligence, cyber security, and interoperability challenges," *IET Electrical Systems in Transportation*, vol. 13, no. 2, pp. 1-25, Jun. 2023, doi: 10.1049/els2.12083.

[15] G. Nyabuto, "Client-server Architecture, a Review," *International Journal of Advanced Science and Computer Applications*, vol. 3, no. 1, Dec. 2023, doi: 10.47679/ijasca.v3i1.48.

[16] G. Loseto *et al.*, "A Cloud-Edge Artificial Intelligence Framework for Sensor Networks," in *2023 9th International Workshop on Advances in Sensors and Interfaces (IWASI)*, Jun. 2023, pp. 149–154, doi: 10.1109/iwasi58316.2023.10164335.

[17] Y. Y. F. Panduman *et al.*, "An Edge Device Framework in SEMAR IoT Application Server Platform," *Information*, vol. 14, no. 6, pp. 1-25, May 2023, doi: 10.3390/info14060312.

[18] V. K. Butte and S. Butte, "An End to End Edge to Cloud Data and Analytics Strategy," in *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, Oct. 2022, pp. 1–6, doi: 10.1109/icccnt54827.2022.9984604.

[19] M. Mudassir and M. Mushtaq, "The role of APIs in modern software development," *World Journal of Advanced Engineering Technology and Sciences*, vol. 13, no. 1, pp. 1045–1047, Oct. 2024, doi: 10.30574/wjaets.2024.13.1.0515.

[20] D. Prasetyawan and P. D. Rahmanto, "Development of a Web Service-Based Research Proposal Selection System Using REST API (in Indonesia: Pengembangan Sistem Seleksi Proposal Penelitian Berbasis Web Service Menggunakan REST API)," *JTIM : Jurnal Teknologi Informasi dan Multimedia*, vol. 6, no. 3, pp. 283–295, Sep. 2024, doi: 10.35746/jtim.v6i3.585.

[21] F. A. Akbar, E. P. Mandyartha, and H. Maulana, "An Approach for Automatic Generating RESTFul API Code based on SQL DDL Code," *Technium: Romanian Journal of Applied Sciences and Technology*, vol. 16, pp. 118–123, Oct. 2023, doi: 10.47577/technium.v16i.9969.

[22] K. Anam, D. N. Rofi, and R. Meiyanti, "Monitoring System for Temperature and Humidity Sensors in the Production Room Using Node-Red as the Backend and Grafana as the Frontend," *Journal of Systems Engineering and Information Technology (JOSEIT)*, vol. 2, no. 2, pp. 68–76, Sep. 2023, doi: 10.29207/joseit.v2i2.5222.

[23] H. Sun and L. Chen, "Design of application layer software platform of remote monitoring system," *Frontiers in Computing and Intelligent Systems*, vol. 3, no. 2, pp. 124–126, Apr. 2023, doi: 10.54097/fcis.v3i2.7576.

[24] B. Mondal, I. Arif, T. Barua, and M. R. I. Chowdhury, "Data security in iot devices and sensor networks for robust threat detection and privacy protection," *Academic Journal On Science, Technology, Engineering & Mathematics Education*, vol. 1, no. 01, pp. 19-35, Oct. 2024, doi: 10.69593/ajieet.v1i01.116.

[25] S. Mehta and R. Kumar, "Blockchain-powered Solutions for Ensuring IoVT Data Confidentiality and Integrity," in *2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT)*, Greater Noida, India, Aug. 2024, pp. 1–4, doi: 10.1109/iceect61758.2024.10739157.

[26] A. Ahmad, R. Maulana, and K. Akmal, "Data Privacy and Security in the Age of IoT A Comprehensive Study on Information System Vulnerabilities," *Journal Informatic, Education and Management (JIEM)*, vol. 6, no. 2, pp. 1–7, Jun. 2024, doi: 10.61992/jiem.v6i2.78.

[27] D. N. Mishra, D. A. M. Haval, A. Mishra, and S. S. Dash, "Automobile Maintenance Prediction Using Integrated Deep Learning and Geographical Information System," *Indian Journal of Information Sources and Services*, vol. 14, no. 2, pp. 109–114, Jun. 2024, doi: 10.51983/ijiss-2024.14.2.16.

[28] A. Amune, S. Shahari, S. Kasurde, S. Nimale, S. Surdas, and S. Tayde, "Exploring Predictive Maintenance and Signal Processing Techniques for Automotive Health Monitoring," in *2024 International Conference on Expert Clouds and Applications (ICOECA)*, Bengaluru, India, Apr. 2024, pp. 541–547, doi: 10.1109/icoeca62351.2024.00100.

[29] E. Zero, M. Sallak, and R. Sacile, "Predictive Maintenance in IoT-Monitored Systems for Fault Prevention," *Journal of Sensor and Actuator Networks*, vol. 13, no. 5, pp. 1-20, Sep. 2024, doi: 10.3390/jsan13050057.

BIOGRAPHIES OF AUTHORS

Rizki Ananta Dwiyanto is a Computer Technology student at Telkom University, Faculty of Applied Sciences, who is developing a Website for Early Warning System: Enhancing Vehicle Safety Through Real-Time Monitoring. This project focuses on developing a website-based Early Warning System that allows users to easily check the condition of various vehicle components. By simply accessing the website, users can monitor critical data such as brake pad thickness, tire thickness measured using LIDAR, tire pressure and temperature through TPMS, and tire identification using RFID. This system enhances convenience and efficiency in vehicle maintenance, ensuring real-time monitoring and early detection of potential issues to improve safety and reliability. He can be contacted at email: rizkiadwi@student.telkomuniversity.ac.id.

Giva Andriana Mutiara is an Associate Professor at Telkom University, Department of Applied Science. She completed her Ph.D. in Information and Communication Technology (ICT) from Universiti Teknikal Malaysia Melaka (UTeM) in 2022, and her Master of Engineering in Computer Engineering from Bandung Institute of Technology in 2005. She has participated in several collaborative projects with various industries and received both internal and external grant funding in several research schemes of the Ministry of Research and Technology in recent years. Currently, she is head of Center of Excellence Smart Technology and Applied Science RG and focused her research on Smart Technology and IoT. She holds several intellectual property rights and patents. Her work is reflected in various publications indexed in reputable databases. She can be contacted at email: givamz@telkomuniversity.ac.id.

Marlindia Ike Sari is an Assistant Professor at Telkom University, Department of Applied Science. She completed her Master of Engineering in Electrical-Telecommunication Engineering from Telkom University (formerly as Institute of Technology Telkom) in 2011. She has participated in several collaborative projects with various communities. Currently, she is a member of Center of Excellence Smart Technology and Applied Science RG. Her works is reflected in various publications indexed in reputable databases She can be contacted at email: marlindia@tass.telkomuniversity.ac.id.