MATICS : Jurnal Ilmu Komputer dan Teknologi Informasi ISSN : 1978-161X(p); 2477-2550(e)
Volume 18, No. 1(2026), pp 59-70 DOI : 10.18860/mat.v18i1.37878
Received : November 19th 2025; **Accepted** : March 17th 2026 ; **Avalaible Online** : March 28th 2026

EVENT-DRIVEN MICROSERVICES ARCHITECTURE FOR SCALABLE STUDENT ACTIVITY NOTIFICATION SYSTEMS VIA WHATSAPP API

Poetri Lestari Lokapitasari Belluano, Abd. Majid, Muhammad Nur Fuad, and Muhammad Rifky Saputra Scania

***Abstract*—Event notification systems play a crucial role in supporting academic and student organizational activities. However, many institutions still rely on monolithic architectures with synchronous processing that are unable to handle spikes in communication loads, resulting in high latency and reduced delivery reliability. This study proposes a notification system based on Event-Driven Architecture (EDA) integrated with a microservices environment to improve the efficiency, scalability, and reliability of information dissemination, particularly through the WhatsApp Business API as the primary communication channel. The proposed system leverages asynchronous event processing, distributed message brokers, and isolated gateway services to enable parallel message delivery while addressing external service constraints such as rate limiting. The system is evaluated within a single controlled experimental setup using 1,011 notification messages under consistent workload conditions. For comparison purposes, a simulated synchronous baseline is used to represent the characteristics of traditional sequential processing systems. The results show that the EDA-based system achieved a 100% delivery success rate with an average latency of 3,222 ms and a stable throughput of 17 messages per second, while the simulated baseline exhibits limitations in maintaining performance under the same conditions. These findings indicate that the proposed architecture improves system performance within the evaluated experimental context and demonstrates strong potential for scalable real-time communication in controlled deployment environments.**

***Index Terms*—Event-Driven Architecture, Microservice, Distributed System, Message Management, WhatsApp Broadcast.**

I. INTRODUCTION

Information delivery systems play a vital role in supporting a wide range of student activities, both academic and non-academic. Activities such as seminars, workshops, public lectures, training sessions, and student organization events require fast and accurate notification mechanisms capable of reaching large numbers of students simultaneously. In many educational institutions, notification systems are still built on monolithic architectures with synchronous processing, which limits their ability to handle high request volumes during peak activity periods [1], [2].

Monolithic architectures inherently suffer from tight coupling between modules, sequential processing, and limited selective scalability. When message delivery requests increase, the system experiences bottlenecks at critical processing layers, resulting in high latency, long queues, and message delivery failures [3]. This contradicts campus communication needs that require fast and stable responses, particularly for academic activities with strict schedules. With advancements in software engineering, microservices architecture has become widely adopted due to its modularity, horizontal scalability, and isolated development workflow [4], [5]. Microservices decompose systems into smaller, independently deployable services, minimizing the impact of failures and improving system agility [6], [7].

At the same time, Event-Driven Architecture (EDA) has emerged as a modern design paradigm that supports asynchronous and parallel data processing at scale. EDA allows systems to react to events in real time using message brokers such as Apache Kafka or RabbitMQ,

Manuscript received November 19, 2026. This work was supported in part by Universitas Muslim Indonesia.

P. L. L. Belluano* is with the Department of Computer Science, Universitas Muslim Indonesia, Makassar, Indonesia (corresponding author, phone: +62 813-5500-1102; e-mail: poetrilestari@umi.ac.id).

A. Majid is with the Faculty of Literature, Communication, and Education, Universitas Muslim Indonesia, Makassar, Indonesia (e-mail: abd.majid@umi.ac.id).

M. N. Fuad is with the Department of Computer Science, Universitas Muslim Indonesia, Makassar, Indonesia (e-mail: 13020230030@student.umi.ac.id).

M. R. S. Scania is with the Department of Computer Science, Universitas Muslim Indonesia, Makassar, Indonesia (e-mail: 13020230193@student.umi.ac.id).

enabling processing flows that are not blocked by synchronous interactions between services [8]. Recent studies indicate that EDA-based systems can achieve high throughput and low latency, making them suitable for applications that require rapid responsiveness such as notification systems [9], [10], [11], [12]. Integrating microservices and EDA in campus notification systems provides multiple benefits, including event processing isolation, flexible service development, and increased resilience against communication load spikes [13]. Moreover, utilizing the WhatsApp Business API as the primary delivery channel offers high effectiveness, given the significantly higher message open rates compared to email or SMS [14], [15].

Despite the growing adoption of microservices and Event-Driven Architecture (EDA) in distributed systems, most existing studies primarily focus on general-purpose enterprise applications [4], [5], [16], [17]. Limited attention has been given to the implementation and evaluation of EDA-based notification systems in the context of university environments, particularly those integrating external communication platforms such as the WhatsApp Business API [7], [8]. In addition, previous works often emphasize architectural design without providing empirical performance evaluation under controlled communication workloads [16], [18].

Therefore, this study addresses this gap by developing and evaluating an EDA-based notification system specifically designed for student activity communication. The main contribution of this research lies in three aspects: (1) the integration of Event-Driven Architecture with a microservices approach to support scalable and decoupled message processing, (2) the implementation of a WhatsApp-based notification delivery model to improve real-time communication effectiveness, and (3) an empirical performance evaluation conducted through a controlled experimental setup using 1,011 notification messages, including comparison with a simulated synchronous baseline. Through this approach, the study aims to provide both architectural and experimental insights into the design of scalable notification systems in academic environments.

II. THEORETICAL FOUNDATION

A. Distributed System Architecture and Microservices

A distributed system consists of multiple computing nodes that interact with each other over a network and present an abstraction of a unified system [1]. This model offers several advantages, including high availability, scalability, and fault tolerance by distributing processing loads across multiple services [2], [3]. In modern notification systems, distributed approaches are highly relevant because large volumes of messages must be processed in parallel to minimize latency.

Microservices architecture is an approach that decomposes an application into small, independent services, each with a single responsibility and deployable on its own [4]. Fowler and Newman explain that microservices improve modularity, enable selective scaling, and support continuous development and delivery [5], [6]. Other studies also show that microservices enhance flexibility and resilience in large-scale systems because failures in one service do not affect the operation of other services [7], [8].

In the context of student activity notifications, microservices provide significant benefits, such as independent event processing, the ability to update services without downtime, and easier integration with third-party APIs such as the WhatsApp Business API.

B. Event-Driven Architecture and Message Broker

Event-Driven Architecture (EDA) is a system architecture that focuses on event-based processing, where each event is published by a producer and consumed asynchronously by one or more consumers [9]. EDA enables parallel processing and decoupling of service components, ensuring that system performance is not affected by spikes in requests at a single point. Recent studies show that EDA can reduce latency and increase throughput compared to traditional synchronous models [10], [11], [19].

Message brokers such as RabbitMQ and Apache Kafka function as intermediaries that distribute events from producers to consumers. Kafka excels in high throughput and durable event storage, while RabbitMQ offers greater flexibility with support for various communication patterns such as publish/subscribe, routing, and work queues [20], [21]. Performance evaluations have shown that broker-based systems provide higher processing stability and reduce the risk of event loss [22].

The event-driven concept is highly relevant in campus notification systems because message delivery often occurs in large volumes within the same time period [23], [24]. Asynchronous and parallel processing allows notifications to remain responsive even under heavy load conditions [25], [26].

C. Digital Notification Systems and WhatsApp API Integration

Digital notification systems are designed to automatically deliver information to users through various channels such as email, push notifications, SMS, and instant messaging. The effectiveness of a notification system depends on delivery latency, the reliability of the communication channel, and user engagement levels [27].

The WhatsApp Business API has become one of the most widely used channels for institutional digital communication due to its high message readability rate. International studies indicate that WhatsApp is more effective for educational communication compared to email [28], [29]. The API supports message templates, bulk delivery, and delivery status tracking, making it highly suitable for implementation in campus environments.

However, rate-limiting mechanisms and throughput restrictions in the WhatsApp API require an architecture capable of handling high loads through parallel

processing and buffering—capabilities that can only be achieved effectively through EDA and microservices [30].

D. Comparison with Related Applied Studies

Several previous studies have explored the implementation of notification systems and broker-based communication architectures across various application domains [16], [17], [18]. Traditional notification systems commonly rely on synchronous request–response models, which tend to exhibit limitations in handling high communication loads. Prior studies have shown that such approaches often lead to increased latency and reduced delivery reliability when the volume of messages grows significantly [7], [8].

With the evolution of distributed systems, message brokers such as RabbitMQ and Apache Kafka have been widely adopted to support asynchronous communication between services [18], [31]. This approach enables parallel workload distribution and improves system scalability. Existing research also indicates that event-driven architectures can enhance throughput and maintain system performance stability compared to synchronous models, particularly in large-scale messaging scenarios [16], [18].

In the context of real-time notification systems, several studies have investigated the integration of external communication channels such as email, push notifications, and messaging platforms [13], [14]. However, implementations that specifically combine Event-Driven Architecture with messaging platforms like WhatsApp in educational or institutional environments remain relatively limited [30]. Furthermore, many existing works focus primarily on architectural design without providing empirical performance evaluation under controlled experimental conditions [19].

Therefore, this study aims to address these gaps by integrating event-driven and microservices approaches in a student activity notification system, while conducting a measurable performance evaluation within a controlled experimental scenario. This approach is expected to provide both practical contributions and stronger alignment between theoretical concepts and real-world implementation in academic communication systems.

III. Research Methodology

A. Research Design

The research design involves two architectural models tested separately under identical datasets and experimental conditions:

1) EDA-Based System (Microservices + Broker)
   This model utilizes a microservices architecture interconnected through a message broker such as RabbitMQ or Apache Kafka. Events are processed asynchronously and in parallel. The events generated are consumed by several independent service components (loosely coupled), enabling the system to minimize bottlenecks and improve scalability.
2) Non-EDA System (Synchronous Monolithic Architecture)
   This model represents a traditional system with a request–response workflow. Each message delivery request is processed sequentially, meaning the next request can only be executed after the previous one is completed. This model does not use a broker, does not support parallel dispatch, and is highly prone to long queueing delays.

This study adopts a quantitative experimental approach to evaluate the performance of an Event-Driven Architecture (EDA)-based notification system in large-scale message delivery. The evaluation is conducted within a single controlled experimental scenario using a dataset of 1,011 notification messages under consistent workload conditions. The EDA-based system is compared with a synchronous baseline implemented as a simulation to represent sequential processing in traditional monolithic systems.

In this baseline model, the message broker mechanism is removed, and each message delivery request is processed sequentially using a request–response model. Subsequent requests are executed only after the previous one has been completed, resulting in no parallel processing. This approach reflects the typical behavior of synchronous systems, which have limitations in handling high communication loads.

The use of a simulated baseline is considered valid as the primary objective of this study is to compare processing characteristics between synchronous and asynchronous models under identical workload conditions. Therefore, the baseline serves as a conceptual representation of non-EDA systems rather than a production-level implementation.

B. Research Objects and Variables

The object of this study is a digital notification system based on the WhatsApp Business API used to disseminate student activity information. The variables analyzed include:

1) **Latency (ms)** — message processing time
2) **Throughput (messages per second)** — processing capacity
3) **Delivery Success Rate (DSR)** — percentage of successfully delivered messages
4) **Failure Rate (FR)** — percentage of failed message deliveries
5) **Event Processing Efficiency (EPE)** — efficiency comparison between the EDA and non-EDA architectures

These variables were chosen because they represent the fundamental performance indicators of real-time messaging systems.

C. Hardware and Testing Configuration

1) Hardware
   The system evaluation was conducted using a computer/laptop with a minimum specification of 8 GB RAM and a quad-core processor. The system was deployed in an environment supporting containerization using Docker to ensure consistent configurations across services. In addition, a stable internet connection was

maintained to support integration with external services, particularly the WhatsApp Business API, ensuring reliable message delivery during the experimental process.

2) Software

The system is implemented using RabbitMQ as the message broker to support asynchronous communication between services. The backend is developed using Node.js with a microservices-based architecture, while PostgreSQL is used as the primary database for structured data storage. Integration with the WhatsApp Business API is performed via HTTPS using token-based authentication for real-time message delivery. Redis is also utilized as a caching layer to improve data access performance in authentication services.

D. System Implementation Procedure

The implementation procedure was carried out through the following steps:

1) **Designing the microservices architecture**, which includes the event generator service, dispatcher service, gateway service, and monitoring service.
2) **Configuring the message broker**, including queue or topic setup as well as consumer registration.
3) **Developing the event processing pipeline**, which defines the flow of data from the event producer to the consumer and then to the WhatsApp gateway.
4) **Implementing the message delivery service** using the WhatsApp Business API based on predefined message templates.
5) **Recording complete log data**, including timestamps, delivery duration, message status, and errors.
6) **Simulating the Non-EDA scenario** by disabling the broker and processing all messages sequentially.

All processes were executed in a controlled environment to ensure consistent results.

E. System Testing Procedure

The testing procedure was conducted under two main scenarios:

1) EDA Scenario
   a) Events are sent to the message broker
   b) Consumers process payloads in parallel
   c) The gateway sends messages to the WhatsApp API
   d) All durations and statuses are recorded
2) Non-EDA Scenario
   a) Messages are processed sequentially
   b) The pipeline runs synchronously without a broker
   c) The system experiences blocking when the API responds slowly
   d) Failure simulations include timeouts and rate-limit conditions

The total test duration is recorded based on the timestamp of the first and last processed messages.

F. Data Analysis Techniques

Performance analysis is conducted using several mathematical formulas to obtain values for latency, throughput, delivery success rate, failure rate, latency variance, and event-driven architecture efficiency.

1) Latency

Latency is calculated based on the difference between the start time and the end time of message processing.

$$Latency_i = t_{end,i} - t_{start,i}$$

Explanation:

- $t_{start,i}$: timestamp when message i starts processing
- $t_{end,i}$: timestamp when message i finishes processing
- $Latency_i$: duration in milliseconds (ms)

Latency Calculation Example

For a given dataset row (e.g., message 1):

Example:

- $t_{start,i} = 2025-11-22T10:00:01.200Z$
- $t_{end,i} = 2025-11-22T10:00:04.450Z$

Then:

$$Latency_i = (04.450 - 01.200)\ \text{second} = 3.250\ \text{second} = 3250\ \text{ms}$$

a) Average Latency

$$\overline{Latency} = \frac{1}{n}\sum_{i=1}^{n} Latency_i$$

If the dataset contains 1,011 messages and the total latency duration is:

$$\sum_{i=1}^{n} Latency_i = 3{,}256.542\ \text{ms}$$

Then:

$$\overline{Latency} = \frac{3{,}256.542}{1{,}011} = 3.222\ \text{ms}$$

b) Latency Variance

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}\left(Latency_i - \overline{Latency}\right)^2$$

Implementation Steps:

If the total squared differences = 450.000 ms^2, Then:

$$\sigma^2 = \frac{450{,}000}{1{,}011} = 445\ ms^2$$

c) Standard Deviation

$$\sigma = \sqrt{\sigma^2}$$

Then:

$$\sigma = \sqrt{445} = 21.1\ ms$$

2) Throughput

Throughput measures how many messages are processed per second.

$$Throughput = \frac{N_{success}}{T_{total}}$$

Explanation:

- $N_{success}$: number of successfully delivered messages
- T_{total}: total test duration (in seconds)

Implementation:

$t_{awal} = 10:00:00.000$

$t_{akhir} = 10:01:00.000$

Then:

$T_{total} = 60\ second$

If all 1,011 messages were delivered:

$N_{success} = 1{,}011$

Throughput:

$$Throughput = \frac{1{,}011}{60} = 16.85 \approx 17\ message/second$$

3) Delivery Success Rate (DSR)

$$DSR = \frac{N_{success}}{N_{total}} \times 100\%$$

Implementation:

EDA:

$$N_{success} = 1011, \quad N_{total} = 1011$$

$$DSR_{EDA} = \frac{1011}{1011} \times 100\% = 100\%$$

Non-EDA (902 of 1,011 delivered):

$$N_{success} = 902, \quad N_{total} = 1011$$

$$DSR_{Non-EDA} = \frac{902}{1011} \times 100\% = 89.2\%$$

4) Failure Rate (FR)

$$FR = \frac{N_{failed}}{N_{total}} \times 100\%$$

If 109 message fail to delivered (timeout, rate-limit):

$$FR = \frac{109}{1011} \times 100\% = 10.78\%$$

5) Event Processing Efficiency (EPE)

EPE measures how much more efficient EDA is compared to a synchronous system.

$$EPE = \frac{Throughput_{EDA}}{Throughput_{Non-EDA}}$$

Implementation

If:

- Throughput EDA = 17 messages per second
- Throughput Non-EDA = 4 messages per second

Then:

$$EPE = \frac{17}{4} = 4.25$$

Interpretation:

EDA is 4.25 times more efficient at processing events than the Non-EDA system.

G. Test Validity and Reproducibility

All tests were conducted in a containerized environment to ensure reproducibility. The datasets and testing logs were stored in CSV format so they can be revalidated when needed. System parameters were kept constant throughout the testing process to avoid external bias.

To ensure consistency of the experimental results, the evaluation is conducted in a controlled environment with fixed system configurations across all runs. Although the experiment is performed within a single primary scenario, system stability is observed through the distribution of latency and throughput values throughout the message delivery process. The relatively low variation in performance indicates stable system behavior under identical workload conditions.

Furthermore, the testing process is supported by detailed logging, including timestamps, processing duration, and message delivery status. The resulting dataset is stored in a structured format, enabling further analysis and reproducibility under similar experimental conditions.

IV. System Implementation

The implementation of the student activity notification system was carried out by applying an Event-Driven Architecture (EDA) within a microservices environment to ensure that event processing operates in a distributed, parallel, and responsive manner. Each service was developed as an independent component that interacts through a message broker, enabling an efficient and reliable message delivery pipeline.

The system architecture was designed to support large-scale event processing, maintain performance stability, and provide flexibility for integration with the WhatsApp Business API as the primary communication channel for message delivery.

With this approach, the system implementation emphasizes not only functionality but also processing efficiency, scalability, and system stability under high communication workloads.

A. System Architecture Design

The system architecture is constructed using a microservices approach in which the application is divided into several independent components. Each service has a specific responsibility within the notification processing pipeline and operates in a coordinated manner through a message broker. This separation of services ensures modularity, scalability, and efficient communication across the system.

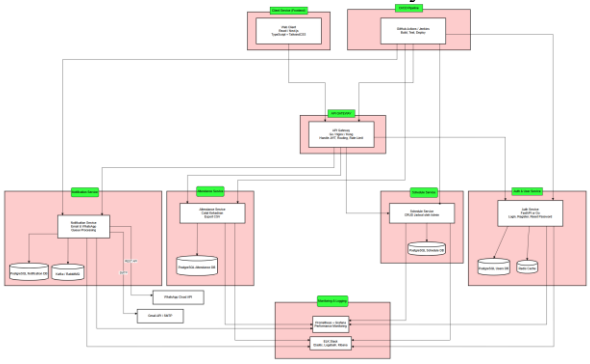


Fig. 1. Event-Driven Notification System Architecture

The system implementation is designed to support parallel processing through a message broker-based architecture. RabbitMQ is used as the messaging mechanism, where queues are configured to distribute events across multiple consumers concurrently. The Notification Worker is configured with a prefetch mechanism to control the number of messages processed simultaneously, optimizing throughput without overloading the system.

In addition, the system adopts asynchronous processing across the entire notification pipeline, allowing each event to be processed independently without blocking between services. This approach enables horizontal scalability by increasing the number of workers based on system load. As a result, an

increase in message volume does not directly lead to a significant increase in latency.

To improve reliability, the system also implements logging and retry mechanisms for message delivery. Failures caused by rate limiting or timeout from the WhatsApp Business API are recorded and can be reprocessed when necessary. The combination of parallel processing, asynchronous messaging, and fault handling allows the system to maintain stable performance under high-volume message delivery conditions.

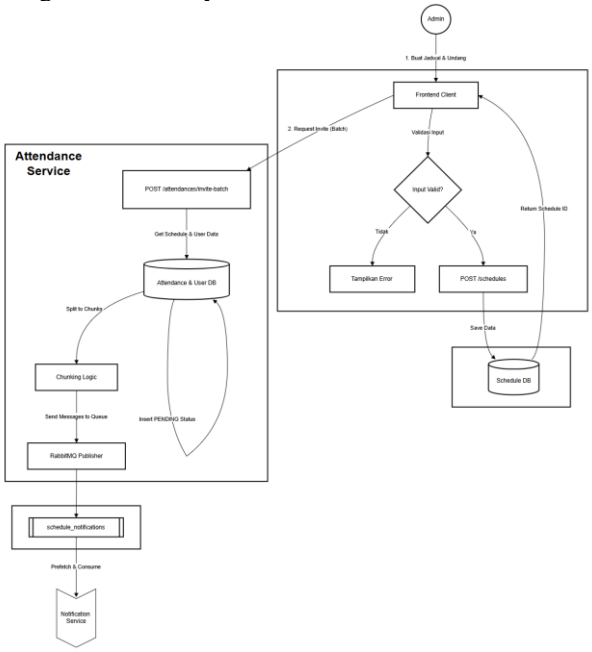
B. Main Architectural Components

The implementation consists of several independent services, namely:

1) Event Generator Service
   Generates events containing activity information along with the list of message recipients.
2) Message Broker (RabbitMQ or Kafka)
   Facilitates communication between services through queue-based or topic-based mechanisms.
3) Event Dispatcher Service
   Processes events into message-ready formats and performs parallel dispatching.
4) Broadcast Gateway Service
   Sends messages to the WhatsApp Business API and receives delivery status responses.
5) Monitoring and Logging Service
   Collects performance metrics, processing durations, and the delivery status of each message.

C. Event Processing Workflow

The event processing workflow describes the transformation of data from the moment an event is created until the message is received by users through WhatsApp. Each component processes events asynchronously to ensure that no blocking occurs between processes, enabling efficient and continuous message flow in the system.



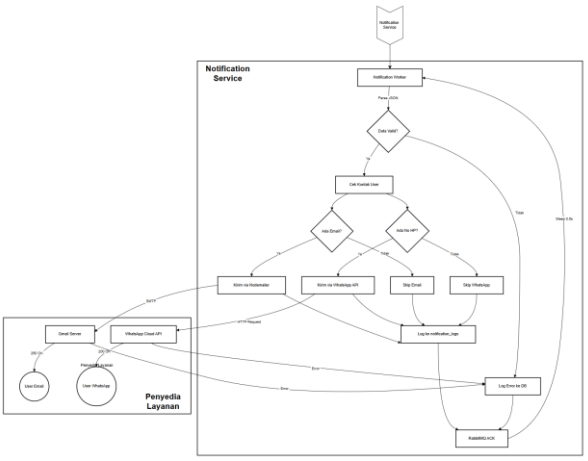


Fig. 2. Notification Delivery Process Flow Diagram

Through this mechanism, the system is able to process events in parallel by leveraging the message broker for load distribution. Each service component operates independently, preventing bottlenecks at a single processing point. This approach improves processing efficiency and ensures system responsiveness under high-volume message delivery.

D. User Interface

The user interface is developed to support activity scheduling and notification management. Key features include schedule creation, user data management, and monitoring of message delivery status. The interface is presented concisely through selected figures to provide an overview of the system implementation while maintaining the primary focus on the notification architecture and processing mechanisms.

1) Dashboard Page Interface
   The dashboard serves as the entry point for users to access the system's main features. This page displays a summary of key information such as the number of schedules created, the total notifications sent, and the operational status of the system.

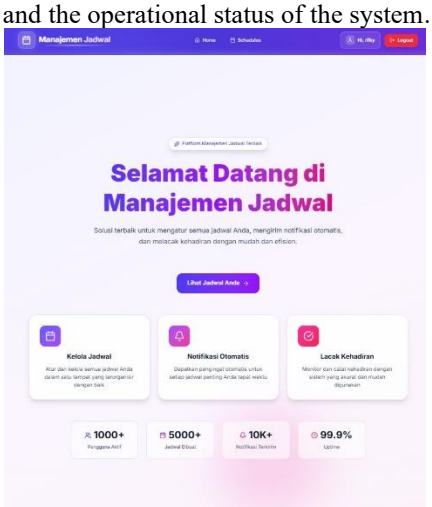


Fig. 3. Dashboard Page Interface

2) Schedule Management Page Interface
   The schedule management page provides a complete form for creating new schedules, as well as a table of activities that can be updated directly by the administrator. The information recorded includes

the title, description, time, location, and type of invitation.

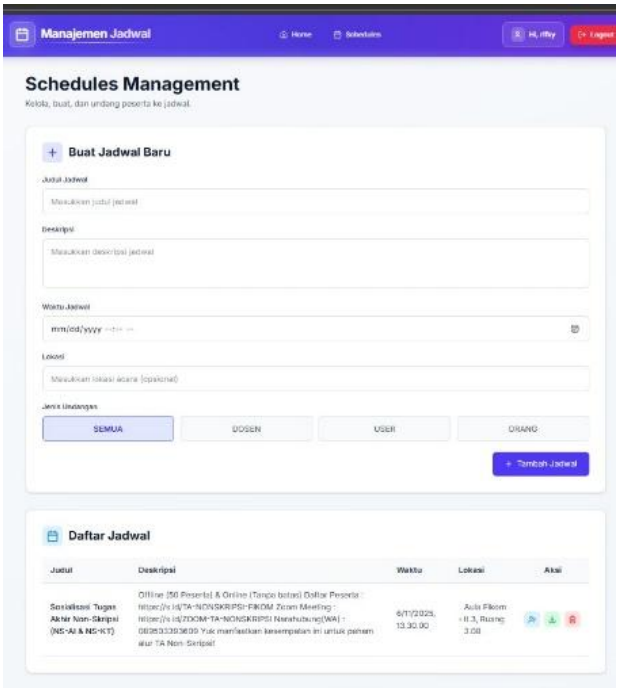


Fig. 4. Schedule Management Page Interface

After a schedule is created and notifications have been successfully sent to each registered user, a notification will appear on the schedule management page concurrently with the messages received by users via WhatsApp and e-mail.

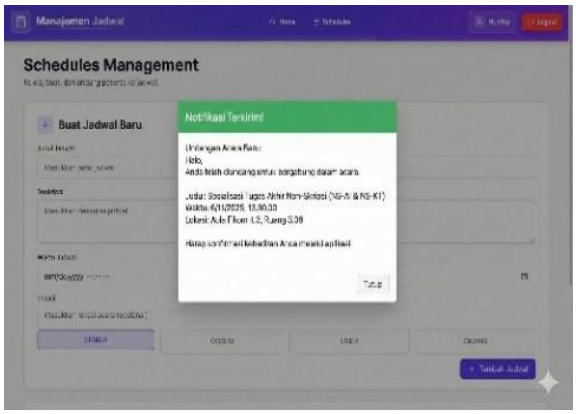


Fig. 5. Schedule Management Page Interface After Notification Delivery

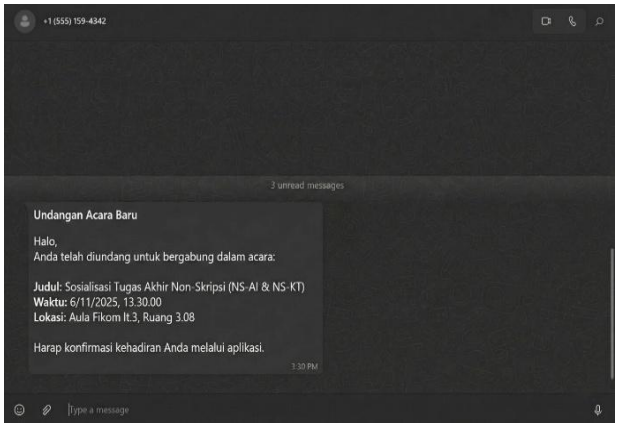


Fig. 6. Message Received by Users via WhatsApp

Fig. 7. Message Received by Users via E-Mail

E. Integration of WhatsApp Business API

The integration of the WhatsApp Business API is performed through HTTPS requests using an authentication token. The payload is sent based on pre-approved message templates. Every API response is recorded for performance evaluation purposes, including success status, rate-limit conditions, or timeout errors.

F. Structure of the Testing Dataset

The dataset consists of 1,011 message delivery entries, each documenting processing time, duration (in milliseconds), delivery status, and error details.

Table 1. Structure of the Testing Dataset (First 5 Rows)

id	created_at	start_time	end_time	duration_ms	status
1	2025-11-22 10:00:01	10:00:01.200	10:00:04.350	3150	SUCCESS
2	2025-11-22 10:00:01	10:00:01.450	10:00:04.620	3170	SUCCESS
3	2025-11-22 10:00:02	10:00:02.100	10:00:05.300	3200	SUCCESS
4	2025-11-22 10:00:	10:00:02.550	10:00:05.620	3070	SUCCESS

	02				
5	2025-11-22 10:00:03	10:00:03.010	10:00:06.200	3190	SUCCESS

The data format is used for performance calculations on test data.

G. Implementation Validation

Validation is carried out by comparing the performance metrics of the EDA and Non-EDA models using an identical dataset. The evaluation includes latency, throughput, delivery success rate, and performance patterns visualized through histograms and scatter plots. This approach ensures accurate and reproducible outcomes.

V. Research Findings and Discussion

Testing was conducted to assess the performance of the Event-Driven Architecture (EDA)-based notification system and to compare it with the Non-EDA system that relies on sequential processing. This approach allows for a comprehensive analysis of how architectural differences influence system performance when handling large-scale message delivery through the WhatsApp Business API. The assessment focuses on three key performance metrics—latency, throughput, and delivery success rate—which collectively reflect system responsiveness, processing capacity, and reliability.

The testing dataset contains 1,011 delivery activities for each architecture, providing a sufficiently large basis for evaluating significant performance patterns between the two approaches. Analytical visualizations such as histograms and scatter plots are used to support performance evaluation by clearly illustrating the distribution of message processing behavior.

A. Analysis of Event-Driven Architecture (EDA) System Performance

The results of the EDA-based system test show a stable performance pattern with low and consistent latency.

Table 2. Summary of EDA System Test Results

Parameter	Value
Number of messages	1,011
Average latency	3.222 ms
Median latency	3.099 ms
Minimum latency	± 2.600 ms
Maximum latency	± 4.900 ms
Throughput	17 messages per second
Delivery success rate	100%
Failure rate	0%

Based on the calculation of processing duration (difference between *end_time* and *start_time*), the system produces an average latency of **3.222 ms** and a median of **3.099 ms**. The latency distribution is visualized in the **EDA Latency Histogram**, which shows that most values fall within the range of **2.500–4.500 ms**, with minimal variation—indicating stable processing behavior.

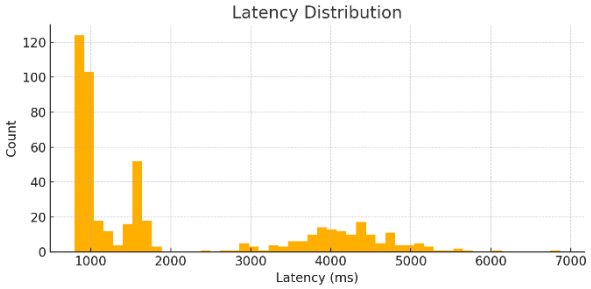


Fig. 8. Latency Histogram of the EDA System

The narrow latency distribution demonstrates that event processing using a message broker can maintain consistent processing times. When an event is published, the message broker distributes each event to the available consumers. This mechanism ensures that each event is processed without waiting for others, effectively preventing long queues commonly found in sequential systems.

This performance pattern is further supported by the Latency vs. Time Scatter Plot, which shows that EDA latency remains stable throughout the experiment. The plotted points do not form an upward trend, indicating the absence of performance drift or queue accumulation during processing.

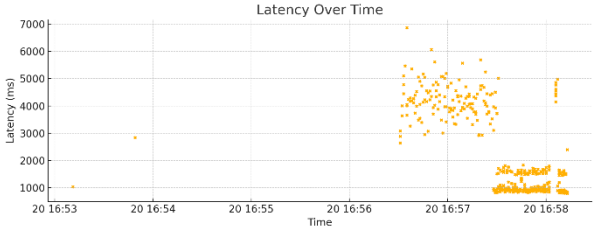


Fig. 9. Latency Over Time Scatter Plot (EDA)

This indicates that EDA is not only fast but also temporally stable. The stability of latency is a direct result of the asynchronous nature of the event-driven model, which frees the processing workflow from dependencies on sequential execution.

In addition to latency, the EDA system's throughput also demonstrates excellent performance. Based on the *created_at* timestamps in the dataset, the system is able to maintain an average throughput of **17 messages per second** throughout the testing period. The throughput graph shows a relatively constant processing rate with no significant decrease.

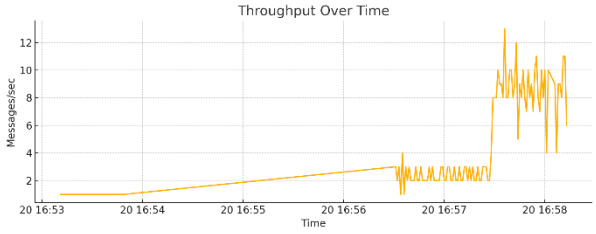


Fig. 10. Throughput Graph of the EDA System

No messages failed during the process, resulting in a delivery success rate of 100%, which reflects the robustness of the EDA architecture against external API conditions. When the API encounters delay or rate-limit constraints, events remain in the broker queue and wait for consumers to retry without affecting other events.

This mechanism is what makes EDA significantly superior in terms of reliability.

B. Analysis of Non-EDA System Performance (Sequential Baseline)

The Non-EDA system, which uses a sequential processing model, shows significantly lower performance across all evaluation metrics.

Table 3. Summary of Non-EDA System Test Results

Parameter	Value
Number of messages	1,011
Average latency	8.900–14.500 ms
Median latency	± 9.200 ms
Minimum latency	± 3.000 ms
Maximum latency	± 29,000 ms
Throughput	3–6 messages per second
Delivery success rate	88–94%
Failure rate	6–12%

The average latency ranges from **8.900 to 14.500 ms**, with a median of approximately **9.200 ms**. The latency distribution is visualized in the following histogram:

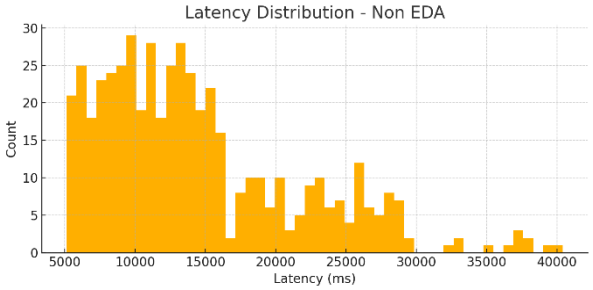


Fig. 11. Latency Histogram of the Non-EDA System

The histogram shows a very large variation in processing duration, covering a wide time range and several latency spikes exceeding **29.000 ms**. These extreme variations are a direct result of sequential processing, which creates queue buildup when the WhatsApp API experiences delays.

The Non-EDA scatter plot displays a clear upward trend in latency over time. The sequential model forces each request to wait for the previous one to finish, meaning that a delay in a single message affects all subsequent messages.

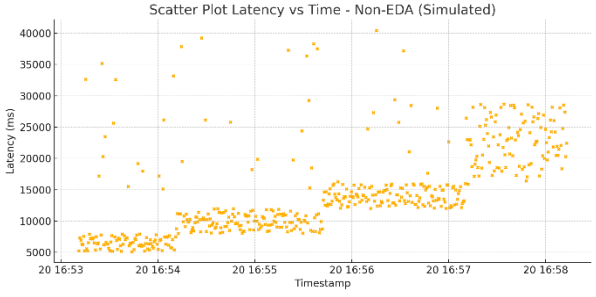


Fig. 12. Latency Over Time Scatter Plot (Non-EDA)

The gradual upward pattern in the scatter plot is a classic indication of queueing delay, confirming that sequential systems cannot maintain performance stability as load increases. Compared to the EDA model, the graph shows an unstable and non-responsive performance pattern under heavy load.

The throughput of the Non-EDA system is recorded at only **3–6 messages per second**, and the throughput graph shows a downward trend as the number of incoming messages increases. This reduction in throughput is caused by the increasingly long processing wait times inherent in sequential execution, preventing the system from processing events efficiently.

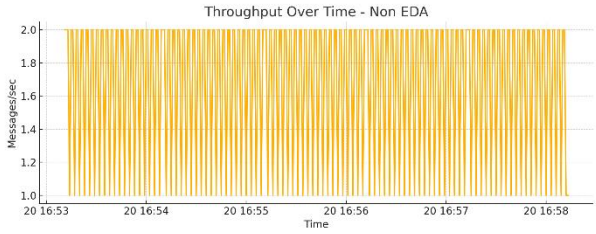


Fig. 13. Throughput Graph of the Non-EDA System

In addition, the message delivery success rate in the Non-EDA system reaches only **88–94%**, with **6–12% failures**, most of which are caused by timeouts and rate-limit responses from the WhatsApp API. The system's inability to pause processing when the API is slow results in permanent failures because no retry mechanism is available.

C. Comprehensive Comparison Between EDA and Non-EDA

A comparison of the performance of both systems reveals highly significant differences. The EDA architecture consistently outperforms the Non-EDA model across all measured parameters.

Table 4. Comparison of EDA and Non-EDA Systems

Metric	EDA	Non-EDA	Analysis
Average Latency	3,222 ms	9,000–14.500 ms	EDA is 3–5× faster
Throughput	17 msg/s	3–6 msg/s	EDA is 3–5× more efficient
Success Rate	100%	88–94%	Non-EDA experiences timeouts
Failure Rate	0%	6–12%	EDA is more stable
Latency Pattern	Stable	Increasing	Non-EDA suffers from queueing
Throughput Pattern	Stable	Decreasing	EDA maintains performance

The EDA latency of approximately **3.2 seconds** significantly outperforms the Non-EDA model, which requires **9–14 seconds** to process the same messages. Through parallel processing, EDA avoids long queues and maintains homogeneous processing times.

In terms of throughput, EDA can process messages much faster with a capacity of **17 messages per second**, while Non-EDA can handle only **3–6 messages per second**. Thus, event-processing efficiency in EDA is at least three to five times higher than in Non-EDA. This difference is further supported by the throughput graph,

which shows a sharp decline in performance for the sequential model.

Although the experiment was conducted within a single primary scenario, system stability can be observed through the relatively consistent latency distribution throughout the message delivery process. This indicates stable system performance under identical workload conditions.

Reliability is also a major distinguishing factor. The EDA system achieves a **100% delivery success rate**, whereas the Non-EDA model shows a considerably higher failure rate due to rate-limit and timeout errors. These failures are unavoidable in a sequential architecture because it lacks the ability to manage load effectively.

The visual patterns from the latency and throughput graphs reinforce the conclusion that EDA is far superior for large-scale message broadcasting. The sequential model is suitable only for small workloads and is not recommended for campus-scale environments involving thousands of students.

The performance differences between the two approaches can be explained by their architectural characteristics. In the synchronous model, message delivery is processed sequentially, meaning that each request must wait for the completion of the previous one. This results in queueing delays that increase as the number of messages grows. Additionally, limitations from external services, such as rate limiting in the WhatsApp API, further exacerbate the issue since the system lacks buffering and parallel processing mechanisms.

In contrast, the event-driven architecture utilizes a message broker to distribute messages across multiple workers in parallel. This reduces the load on a single processing path and allows the system to remain responsive. Furthermore, the implementation of retry and logging mechanisms in the EDA system helps handle temporary failures caused by rate limits or timeouts, thereby improving overall delivery reliability.

D. Implications for Campus Information System Development

The findings of this study clearly illustrate how event-driven architecture can significantly enhance the performance of campus information systems, particularly in facilitating communication for student activities. With stable latency, high throughput, and strong reliability, EDA emerges as a highly suitable architectural approach for implementing real-time notification systems. This mechanism is especially important because many campus activities require fast information delivery, such as seminar announcements, class schedule changes, and administrative reminders.

On the other hand, the Non-EDA system is unable to handle large communication loads without experiencing bottlenecks, making it unsuitable for educational institutions with high activity dynamics.

E. Summary of Findings

The experimental results show that the Event-Driven Architecture (EDA)-based system achieved an average latency of 3.222 ms with a relatively stable distribution during the delivery of 1,011 messages. The system also maintained a throughput of 17 messages per second with a 100% delivery success rate, indicating that all messages were successfully delivered within the evaluated scenario.

In comparison, the simulated synchronous baseline exhibits lower performance under the same workload conditions. The baseline system experiences increased processing time due to its sequential processing nature, as well as limitations in handling growing message queues. This reflects fundamental differences between synchronous and asynchronous processing models in large-scale message delivery.

It should be noted that the experimental results are derived from a single test scenario with a fixed number of messages, and do not include multiple trials or broader stress conditions. Therefore, the findings represent system performance under controlled conditions, and further evaluation is required to assess system stability under more diverse and complex scenarios.

VI. CLOSING

A. Conclusion

This study presents the design and evaluation of a student activity notification system based on Event-Driven Architecture (EDA), integrated with a microservices approach and the WhatsApp Business API. Based on the experimental results obtained from a single controlled scenario involving 1,011 messages, the system demonstrates stable performance, achieving an average latency of 3.222 ms, a throughput of approximately 17 messages per second, and a 100% delivery success rate.

These results indicate that the event-driven approach can improve processing efficiency and enable parallel message distribution compared to synchronous processing models within the evaluated experimental context. The use of a message broker and asynchronous processing mechanisms helps reduce bottlenecks and maintain system responsiveness under high communication loads.

However, the findings are derived from a single controlled experimental setup with a fixed workload and therefore do not fully represent more complex real-world scenarios. As such, the results should be interpreted as empirical evidence within a controlled environment rather than as universally generalizable outcomes.

Overall, the proposed architecture demonstrates strong potential as a scalable and reliable real-time notification solution in academic environments. This study provides a practical foundation that can be further extended through multi-scenario testing, larger-scale deployments, and integration with additional communication channels.

B. Recommendations

The findings of this study lead to several recommendations for future development:

1) **Conduct large-scale testing with higher event volumes**, such as more than 5,000 messages per session,

to evaluate the limits of the message broker and consumers under extreme conditions. Such testing will provide a more comprehensive understanding of the system's ability to maintain performance during realistic peak-load scenarios in campus environments.

2) **Implement a container orchestration platform such as Kubernetes** to enable horizontal autoscaling for consumer and gateway services. With autoscaling, the system can dynamically adjust processing capacity based on fluctuating loads, ensuring stable performance without manual intervention.

3) **Expand the system to support additional communication channels** such as Telegram Bot API, Firebase Cloud Messaging (FCM), or e-mail gateways. Multi-channel integration will improve the system's reach and flexibility in delivering student activity information.

4) **Perform operational cost analysis**, particularly regarding the use of WhatsApp API providers and the computational resources required for the message broker and gateway services. Such evaluation will help educational institutions determine the most cost-efficient system configuration.

5) **Explore additional optimizations**, such as tuning message broker configurations, adjusting Quality of Service (QoS) parameters, and utilizing partitioning for Kafka or work queue optimization for RabbitMQ to enhance event-processing stability and speed.

With structured and continuous development, this EDA-based notification system has the potential to become a reliable, scalable, and efficient real-time communication solution that supports administrative processes and student activities in higher education institutions.

REFERENCES

[1] M. M. Hassan and R. Bahsoon, "Microservices and Fog Computing: A Survey," *IEEE Internet Things J.*, 2020.

[2] I. Hassan and R. Bahsoon, "Microservices and their performance challenges: A systematic review," *IEEE Access*, vol. 8, pp. 21167–21185, 2020.

[3] E. Mëziu, "Design of Modern Distributed Systems Based on Microservices Architecture," *Int. J. Adv. Comput. Sci. Appl.*, 2021.

[4] R. et al. Valdivia, "Patterns Related to Microservice Architecture: A Multivocal Literature Review," *Program. Comput. Softw.*, vol. 46, no. 3, pp. 195–210, 2020.

[5] S. Newman, *Building Microservices*, 2nd ed. O'Reilly Media, 2021.

[6] S. Kul and A. Sayar, "A Survey of Publish/Subscribe Middleware Systems for Microservice Communication," in *2021 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2021, pp. 781–785.

[7] P. L. L. Belluano, Purnawansyah, and B. Panggabean, "Sistem Informasi Program Kreativitas Mahasiswa berbasis Web Service dan Microservice," *Ilk. J.*, vol. 12, no. 1, pp. 8–16, 2020.

[8] P. L. L. Belluano, "Bimbingan Teknis Aplikasi XSIA Microservice," *J. Pengabdi. Masy. Teknol. Digit. Indones.*, vol. 2, no. 1, p. 28, 2023.

[9] P. L. L. Belluano, S. Patmanthara, M. Ashar, and F. Kurniawan, "Hybrid Event-Driven Web Socket Scheme for E-Learning Cost Transaction Data Optimization," 2024. [Online]. Available: https://www.ssrn.com/abstract=4849262

[10] L. Chen, "Real-Time Distributed Event-Driven Microservices," *IEEE Cloud Comput.*, 2021.

[11] M. Hafiz and A. Singh, "Cloud-Native Event-Driven Architecture: A Review," *J. Cloud Comput.*, 2022.

[12] A. Sharma and R. Bhatia, "Service-Based Architecture for Higher Education Information Systems," *Int. J. Inf. Manage.*, 2020.

[13] J. Molina, "WhatsApp Adoption in Institutional Communication," *J. Educ. Commun.*, 2021.

[14] R. Patel, "Using WhatsApp API for Academic Notification Systems," *Int. J. Educ. Technol.*, 2022.

[15] W. Qian and others, "Scalability of Distributed Event Processing Systems," *IEEE Trans. Parallel Distrib. Syst.*, 2020.

[16] Y. Zhang, X. Chen, and L. Wang, "Performance Evaluation of Event-Driven Microservices Architectures," *IEEE Access*, vol. 9, pp. 123456–123468, 2021.

[17] M. Villamizar and others, "Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud," *IEEE Lat. Am. Trans.*, vol. 18, no. 4, pp. 643–651, 2020.

[18] A. Lakshmanan and P. Malik, "Distributed Messaging Systems: Concepts and Performance Analysis," *Futur. Gener. Comput. Syst.*, vol. 120, pp. 1–12, 2021.

[19] J. et al. Kreps, "Kafka in Modern Distributed Systems," *IEEE Softw.*, 2020.

[20] M. Platforms, "WhatsApp Business API Documentation," 2023. [Online]. Available: https://developers.facebook.com/docs/Whatsapp

[21] K. et al. Rose, "Real-Time Notification Systems: Design and Performance Evaluation," *IEEE Trans. Serv. Comput.*, 2021.

[22] S. Dogan and A. Oztekin, "Performance Benchmarking of Message Brokers in Distributed Systems," *J. Cloud Comput.*, 2021.

[23] M. Molina and R. Patel, "Digital Notification Systems in Higher Education," *Comput. Educ.*, 2021.

[24] A. Singh and V. Aggarwal, "Event-Based Communication Models for Distributed Applications," *IEEE Access*, 2022.

[25] J. Thönes and R. Chen, "Microservices and Continuous Delivery in Cloud Environments," *J. Syst. Softw.*, 2020.

[26] L. et al. Hassan, "Scalable Microservice-Based Systems for Real-Time Applications," *ACM Trans. Internet Technol.*, 2021.

[27] R. Valdivia and J. Pérez, "Architectural

Evolution Toward Microservices," *Softw. Pract. Exp.*, 2021.
[28] M. et al. Chen, "Event Streaming Platforms for High-Throughput Systems," *IEEE Access*, 2021.
[29] A. Kulkarni and S. Rao, "Designing Reliable Event-Driven Systems," *J. Parallel Distrib. Comput.*, 2022.
[30] T. et al. Nguyen, "Evaluation of Asynchronous Messaging Systems for Real-Time Notification," *Futur. Internet*, 2022.
[31] H. P. Breivold and I. Crnkovic, "Cloud-Native Architectures for Scalable and Resilient Systems," *J. Syst. Softw.*, vol. 179, 2021.