Journal System Information Business 03(2025)
Copyright ©2024, JSINBIS, p-ISSN: 2502-2377, e-ISSN: 2088-3587
On-line: http://ejournal.undip.ac.id/index.php/jsinbis/article/view/72832

Event Driven Architecture Approach for Synchronization *Real Time* NeoFeeder PDDIKTI

Sopingi[a*] , Sri Sumarlinda[b]

[a,b] Universitas Duta Bangsa Surakarta

Submitted April 30th, 2025; Revised: June 12th, 2025;
Accepted: June 19th, 2025; Available Online: September 29th, 2025
DOI: 10.21456/vol15iss1pp21-337

Abstract

This research aims to produce an Event-Driven Architecture based integration model that can increase the efficiency of data synchronization between the internal academic system and PDDIKTI NeoFeeder. This research uses a Rapid Application Development approach . The research results show that the system is capable of sending more than 1,000 data per second with an average latency of 4.05 ms and a response time of under 40 ms. In conclusion , the Event Driven Architecture approach is capable of helping synchronize academic data to NeoFeeder in real time, but this study has note discussed data consistency between academic data and data in NeoFeeder. The Higher Education Database (Pangkalan Data Pendidikan Tinggi - PDDIKTI) plays a vital role as the central information hub for higher education in Indonesia. *NeoFeeder* serves as middleware designed to bridge the differences between academic information systems across universities and the centralized PDDIKTI database. Currently, NeoFeeder operates using a batch system or manual triggers, which may result in delays in data updates particularly when handling large datasets or sudden changes. To address these limitations, there is a need for a system architecture that is more adaptive, scalable, and responsive to data changes. This research aims to develop an integration model based on an Event-Driven Architecture (EDA) to enhance the efficiency of data synchronization between internal academic systems and NeoFeeder PDDIKTI. The research adopts the Rapid Application Development (RAD) Approach. The results demonstrate that the system is capable of transmitting over 1,000 data records per second, with an average latency of 4.05 milliseconds and a response time of under 40 milliseconds. In conclusion, the Event-Driven Architecture approach proves effective in enabling real-time synchronization of academic data with NeoFeeder. However, this study does not yet address issues related to data consistency between academic systems and the NeoFeeder database.

Keywords : *Event Driven*, NeoFeeder, *Kafka Apache*, *Synchronization*, *Real Time*

Abstrak

Pangkalan Data Pendidikan Tinggi memiliki peran penting sebagai pusat informasi pendidikan tinggi di Indonesia. NeoFeeder hadir sebagai middleware untuk menjembatani perbedaan sistem informasi akademik antar perguruan tinggi dengan database terpusat PDDIKTI. Implementasi NeoFeeder saat ini masih menggunakan sistem batch atau pemicu manual, yang dapat menyebabkan keterlambatan pembaruan data, terutama jika data besar atau ada perubahan mendadak. Untuk mengatasi hal ini, diperlukan arsitektur sistem yang lebih adaptive, scalable dan responsive terhadap perubahan data. Penelitian ini bertujuan untuk menghasilkan model integrasi berbasis Event-Driven Architecture yang dapat meningkatkan efisiensi sinkronisasi data antara sistem akademik internal dan NeoFeeder PDDIKTI. Penelitian ini menggunakan pendekatan Rapid Application Development. Hasil penelitian menunjukkan bahwa sistem mampu mengirimkan lebih dari 1.000 data per detik dengan latensi rata-rata 4,05 ms dan response time di bawah 40 ms. Kesimpulannya, pendekatan Event Driven Architecture mampu dalam membantu sinkronisasi data akademik ke NeoFeeder secara real time, tetapi dalam penelitian ini belum membahas terkait konsistensi data antara data akademik dengan data di NeoFeeder.

Kata kunci: *Event Driven, NeoFeeder, Kafka Apache, Sinkronisasi, Waktu Nyata.*

1. Introduction

The Higher Education Database constitutes a critical component of Indonesia's higher education infrastructure, functioning as a centralized system for the storage and dissemination of comprehensive information pertaining to the administration and delivery of higher education. Within this framework, NeoFeeder has been introduced as a middleware solution specifically designed to address the heterogeneity of academic information systems across higher education institutions and to facilitate seamless integration with the centralized PDDIKTI database (Auliana & Nurasiah, 2018).

NeoFeeder is expected to facilitate and accelerate the data reporting process while simultaneously improving the accuracy and consistency of the reported data. A well-developed information system enables an institution to operate effectively, manage various data processing tasks through the use of

*) Corresponding author: sopingi@udb.ac.id

On-line: http://ejournal.undip.ac.id/index.php/jsinbis/article/view/72832

information technology, and meet users' needs by providing a clear understanding of the system design and its implementation (Rafiezah Rizcha & Yaakub, 2023). According to the NeoFeeder web service manual, data synchronization from academic information systems to NeoFeeder is carried out on a per-record basis and does not yet support batch synchronization. As a result, the synchronization process between an institution's internal systems and NeoFeeder faces several challenges, particularly regarding latency, data inconsistency, and the lack of real-time integration capabilities.

The current integration of NeoFeeder remains batch-oriented or depends on manual triggering mechanisms, which may result in delayed data updates, particularly when handling large-scale datasets or sudden modifications in academic records. Such a non-responsive integration architecture introduces potential risks, including data reporting inaccuracies and latency in fulfilling reporting obligations to the national higher education database (PDDIKTI).

To address these challenges, a more adaptive, scalable, and data-responsive system architecture is required-one that can automatically respond to changes in data without manual intervention. *Event-Driven Architecture (EDA)* has emerged as a relevant and promising solution. In an EDA-based system, each data change is treated as an event that triggers specific services to handle synchronization automatically and in real time, eliminating the need for manual triggers or scheduled synchronization routines. An event-based microservices architecture combines key attributes such as scalability**,** maintainability**,** ease of deployment**,** resilience**,** and reusability (Ubur, 2023).

Through research and development of this system, it is hoped that an *Event - Driven based integration model or prototype can be produced. Architecture* that is able to increase efficiency and effectiveness in the data synchronization process between the internal academic system and NeoFeeder PDDIKTI. Thus, the higher education information system can move towards a more modern, *real time* and responsive digital transformation to the dynamics of higher education management.

2. Theoretical Framework

2.1. Event-Driven Architecture

Event-Driven Architecture (EDA) is a software architecture pattern in which systems are designed around the production, detection, consumption, and reaction to events. Each event represents a significant change in state, such as user input or a modification to the database (Amazon Web Services, 2024). According to a global survey conducted by Solace in 2021, 72% of organizations worldwide had adopted EDA at varying levels of maturity. However, only 13% had reached full maturity, where EDA is implemented comprehensively across the organization. The key benefits reported include improved application responsiveness (46%), enhanced customer experience (44%), and the ability to respond to changes in real time (43%) (Solace Corporation, 2021).

2.2. Web Service

A *web service* is a software component that facilitates communication and interaction between applications over a network, typically using the HTTP protocol. HTTP enables interoperability among heterogeneous systems by leveraging open standards such as XML, JSON, SOAP, and REST (Sopingi, Setyowati, & Purnomo, 2020). Functioning as a request-response protocol within the client-server model, HTTP supports data transfer, request and response handling, web access, and Hypertext functionality. Web services enable diverse applications or systems to exchange data seamlessly and can be accessed from various devices, anytime and anywhere, as long as an internet connection is available.

2.3. NeoFeeder

NeoFeeder is an application launched on February 25, 2022, to assist higher education institutions in managing and synchronizing data with the national higher education database (*Pangkalan Data Pendidikan Tinggi* – PDDIKTI). The application is essential for ensuring that data reporting complies with the Indonesian Ministry of Research, Technology, and Higher Education Regulation No. 61 of 2016 (Brawijaya & Widodo, 2023). *NeoFeeder* is used to manage and transmit structured and verified data—including information on students, lecturers, curricula, courses, study plans (KRS), academic transcripts (KHS), and more—from universities to PDDIKTI. *NeoFeeder* is equipped with data validation features that enable institutions to ensure the accuracy and compliance of submitted data with PDDIKTI standards, preventing formatting errors and meeting government-mandated reporting requirements. Additionally, *NeoFeeder* provides a *Web Service API* that facilitates semester-based reporting through integration with academic information systems (*Sistem Informasi Akademik* – SIAKAD).

2.4. Event Streaming

Event streaming refers to the practice of capturing data in real time from a variety of sources. This concept enables organizations to access and process data as it is generated, rather than after it has been stored in a database (Apache, 2020). The core components of event streaming include: producers**,** which may consist of applications, sensors, or users; brokers, which function as event bus applications that

On-line: http://ejournal.undip.ac.id/index.php/jsinbis/article/view/72832

manage the flow of events; consumers, which receive and process the events; and stream processors, which are responsible for analyzing and transforming streaming data.

One of the application platforms that is capable of performing event streaming is *Apache Kafka*. *Apache Kafka* is an open-source solution that can be developed for real-time data distribution in environments requiring high availability and low fault tolerance (Bucur, Stan, & Miclea, 2020). Apache Kafka is a distributed, publish-subscribe messaging system that enables data transmission from producers to consumers, stores data in the form of logs, and supports high scalability.

Figure 1 illustrates the architecture of Apache Kafka, which consists of a *Kafka cluster* a group of brokers that work collaboratively to provide a distributed messaging service (Hesse, Matthies, & Uflacker, 2020). Within this architecture, *producers* act as message senders, while *consumers* serve as message receivers.

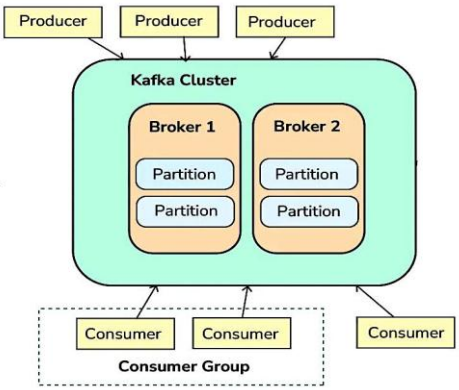

Figure 1. Kafka architecture
(Sharma, 2025)

3. Method

This study focuses on the synchronization between existing academic information systems and the NeoFeeder PDDIKTI application using an *Event-Driven Architecture (EDA)* approach. The research adopts the *Rapid Application Development (RAD)* methodology. RAD facilitates the accelerated transformation of user requirements into software solutions by integrating design, development, and testing into shorter, iterative phases (Živanović, Popović, Vorkapić, Pjević, & Slavković, 2020). The research stages are presented in Figure 2.

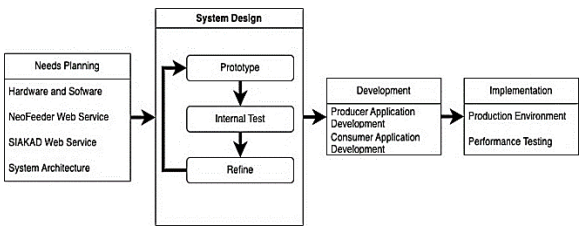

Figure 2. Stage study with RAD approach

There are four main phases in this study, the phrases are:

a. *Planning Need*

This study utilizes three servers: a server for the academic information system (SIAKAD), a server for *NeoFeeder*, and a server for Apache Kafka as the event streaming platform. Each server is interconnected via the internet.

b. *Design System*

A prototype is developed to conduct early testing of the core functionality of the event streaming system in achieving real-time synchronization. This allows potential errors to be identified and corrected at an early stage. At this phase, the prototype is executed using console commands from Apache Kafka, functioning both as a producer and a consumer.

c. *Development*

The applications for the producer and consumer are developed using the Node.js programming language. The producer is integrated with SIAKAD through direct database connections, while the consumer is integrated with NeoFeeder via web service calls

d. *Implementation*

The application is deployed in a real-world environment to enable automatic and real-time synchronization from SIAKAD to NeoFeeder. At this stage, performance testing is also conducted using the tools *org.apache .kafka.tools.ProducerPerformance* and *org.apache.kafka.tools.ConsumerPerformance.*

4. Results And Discussion

4.1. Results

The results of the study are as follows:

a. *Architecture System*

To illustrate the synchronization process between *NeoFeeder* and the academic system using an *Event-Driven Architecture (EDA)* approach, the author developed a system architecture model, as presented in Figure 3 below.:

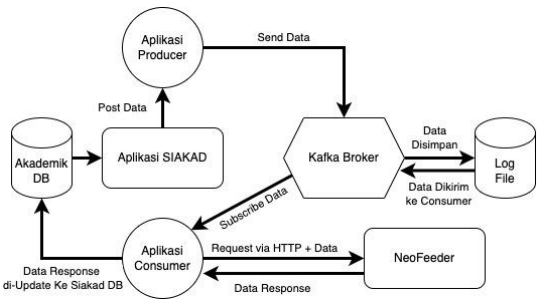

Figure 3. Architecture synchronization

On-line: http://ejournal.undip.ac.id/index.php/jsinbis/article/view/72832

Whenever a change occurs in the academic database, the SIAKAD system sends the data to the Kafka broker via the producer application, without the need to establish a direct connection to NeoFeeder. The Kafka broker stores the data in a log file and queues it for delivery to the consumer application. Upon receiving the data, the consumer application automatically forwards it to NeoFeeder. After receiving a response from NeoFeeder, the consumer application then updates the academic database accordingly.

This mechanism significantly improves synchronization performance in SIAKAD, as it eliminates the need for SIAKAD to wait for a response from NeoFeeder. The entire synchronization process is delegated to the consumer application. Moreover, SIAKAD is capable of synchronizing large volumes of data simultaneously, even though the consumer application still processes and transmits one record per request to NeoFeeder.

b. *Development System*

The producer and consumer applications were developed using the ExpressJS framework, which offers efficient capabilities for routing, handling HTTP requests, and implementing middleware-based security features (Grudniak & Dzieńkowski, 2021). To connect as a Kafka client, the author utilized the *KafkaJS* library, which is known for its lightweight architecture, flexibility, and lack of external dependencies (Ornelas, 2023).

The following is a code snippet illustrating the initialization of the *Kafka producer client*:

```
const { Kafka } = require('kafkajs');

const kafka=new Kafka({
  clientId: 'sopingi-kafka-producer',
  brokers: ['ip_server_kafka:9094']
})
```

Script for Sending Data to the Kafka Broker from the Producer Application:

```
const producer_siakad = kafka.producer ()
const topic_siakad = 'sync-siakad'

const connectKafka = async () => {
  await producer_siakad.connect ()
};
connectKafka().catch(console.error);

router.post('/',async function(rq,rs,n)
{

try {
  var data = JSON.stringify (rq.body)
  await producer_siakad.send ({
    topic: 'sync-siakad ',
    compression: CompressionTypes.GZIP,
    messages: [{ value: data }],
  });
    rs.send ({ message :'Success'})
 } catch (e) {
  rs.status(500).send({message:'failed'})
 }
}
```

Script for Initializing the Kafka Consumer Client:

```
const { Kafka } = require('kafkajs');
const kafka = new Kafka({
  clientId: 'sopingi-kafka-consumer',
  brokers: ['ip_server_kafka:9094']
})
```

Script for Subscribing to Data from the Kafka Broker in the Consumer Application:

```
const consumer_siakad = kafka.consumer({
groupId : 'siakad' });
const topic_siakad = 'sync-siakad'

const connectKafka = async () => {
await consumer_siakad.subscribe ({
    topic:topic_siakad ,
    fromBeginning : true
});

await consumer_siakad.run ({
  eachMessage : async ({ data }) => {
  var json = JSON.parse ( data.value );
  var neoFeeder = await runWs(json);
})
};

connectKafka ().catch( console.error );
```

The synchronization results can be monitored through a Kafka UI application, as shown in *Figure 4*. The Kafka UI provides a visual interface to track the history of academic data that has been synchronized with NeoFeeder.

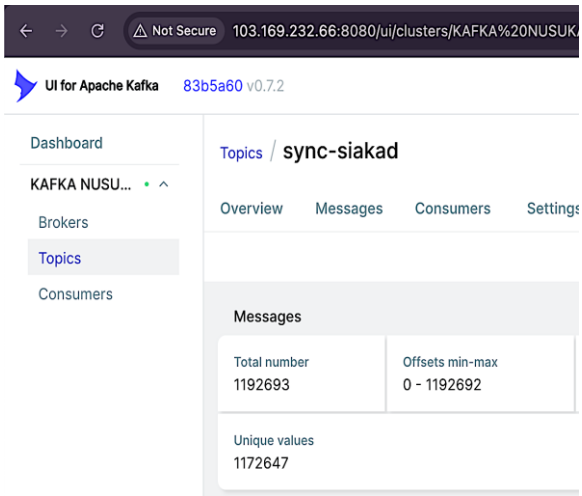

Figure 4. Kafka UI application

On-line: http://ejournal.undip.ac.id/index.php/jsinbis/article/view/72832

c. Testing Performance

The testing performance consists of three key aspects: producer latency in data transmission, the volume of data that can be delivered to the consumer, and the response time of ExpressJS in executing data delivery from the SIAKAD system. Producer latency testing was conducted using org.apache.kafka.tools.ProducerPerformance, a built-in class provided by Apache Kafka. The test was performed with *10,000 records* at a throughput rate of 200 records per second. The results of the producer performance test are presented in Figure 5.

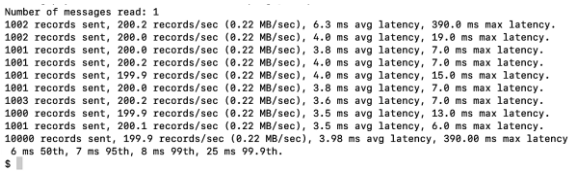

Figure 5. Test results latency *producer*

Based on the results, when the producer transmitted 10,000 records with a throughput of 200 records per second, the average latency was recorded at 4.05 milliseconds. The consumer performance was evaluated using the org.apache.kafka.tools.ConsumerPerformance class to measure the number of records received over a 5-second interval. The results of this evaluation are presented in Table 1.

Table 1. Test Results *Customer Performance*

Header	Value
data.consumed.in.MB	1,9913
MB.sec	0.5654
data.consumed.in.nMsg	10372
nMsg.sec	2944,9177
rebalance.time.ms	3418
fetch.time.ms	104
fetch.MB.sec	19,1475
fetch.nMsg.sec	99730,7692

As shown in Table 1, the consumer was able to receive an average of 2,944.9 messages per second, with a total of 10,372 messages received over a 5-second period.

The performance of ExpressJS in executing data transmission from the SIAKAD system was evaluated using Apache JMeter, a tool capable of testing web service response time performance (Sopingi & Wulandari, 2023). The response time test was conducted with 100 requests over a 30-second period, and the resulting performance graph is presented in Figure 6.

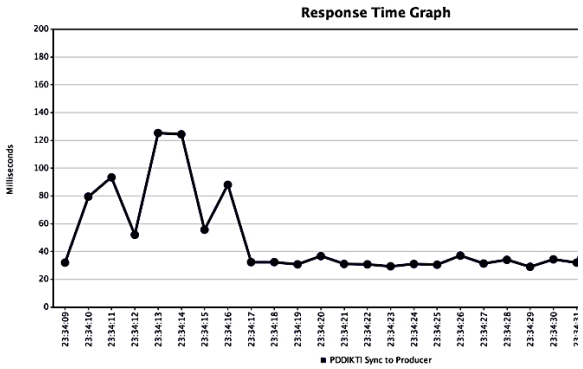

Figure 6. Graph testing *response time*

Based on the graph shown in Figure 6**,** the maximum response time recorded was 125 milliseconds, while the average response time remained below 40 milliseconds.

4.1. Discussion

PDDIKTI (Pangkalan Data Pendidikan Tinggi) serves as the national higher education data center managed by the Indonesian Ministry of Education, Culture, Research, and Technology (Kemendikbudristek). Its main function is to collect and manage data related to all higher education activities in Indonesia. The PDDIKTI business process model consists of the following stages: 1) Data collection, which includes student records, lecturer profiles, curriculum structures, course information, class schedules, grades, student activities, and enrollment status from higher education institutions. 2) Data processing and validation, conducted by both the respective institutions and the PDDIKTI system. 3) Data reporting and utilization, which supports official reporting, accreditation (e.g., BAN-PT), tracer studies, and national higher education policy-making.

NeoFeeder, on the other hand, is a supporting tool implemented at the institutional (campus) level to manage and report data to PDDIKTI in a standardized and routine manner. The NeoFeeder business process includes: 1) Data synchronization from the academic information system (SIAKAD) to the NeoFeeder application. 2) Data management, including students, lecturers, curricula, courses, class schedules, grades, student activities, and academic status, either directly or through API integration with the institution's information system. 3) Local data validation, followed by uploading the verified data to the central PDDIKTI system through a synchronization mechanism. 4) Data status monitoring, which involves tracking synchronization success, identifying errors, and performing data corrections when necessary.

Based on the results of this study and performance testing, the *Event-Driven Architecture (EDA)* approach demonstrated excellent performance in enabling real-time and automated synchronization of academic data without requiring manual triggers. The

system achieved a data transmission rate of over 1,000 records per second, with an average latency of 4.05 milliseconds and an average response time of less than 40 milliseconds. These findings are supported by previous research, which has shown that event streaming platforms such as Apache Kafka offer high throughput performance (Köstler, Reiser, Habiger, & Hauck, 2021). Similarly, Luan Lazzari concluded that event-driven applications provide faster responsiveness to data changes (Lazzari & Farias, 2023). Moreover, the response time testing using the ExpressJS framework also demonstrated strong performance for synchronization tasks, consistent with the findings of Prayogi, who reported that ExpressJS could achieve 100% throughput under high-load conditions (Prayogi, Niswar, Indrabayu, & Rijal, 2020).

In contrast to previous research conducted by Irfan Darmawan *et al.*, which also focused on synchronization of PDDIKTI reporting, their study utilized GraphQL and REST-based methods. The results indicated that the response time for 1,000 requests using GraphQL reached 11,862 milliseconds, while REST recorded a response time of 18,202 milliseconds (Darmawan, Rahmatulloh, & Gunawan, 2022).

Compared to their findings, the results of the present study demonstrate significantly better performance, with faster response times and lower latency, highlighting the efficiency of the Event-Driven Architecture and event streaming approach implemented in this research.

5. Conclusion

The *Event-Driven Architecture (EDA)* approach has proven effective in supporting real-time synchronization of academic data to NeoFeeder, achieving an average latency of 4.05 milliseconds and an average response time below 40 milliseconds. However, the author acknowledges that this study did not address the issue of data consistency between the academic database and NeoFeeder. Therefore, it is recommended that future research focuses on developing a synchronization mechanism that ensures data consistency between the institutional academic system and the NeoFeeder platform.

Acknowledgement

The writers would like to express sincere appreciation and gratitude to Universitas Duta Bangsa Surakarta for providing the necessary facilities, including server infrastructure, internet access, and data resources used for system testing.

Bibliography

Amazon Web Services, I. (2024, November 22). What is EDA (Event-Driven Architecture)? Retrieved March 15, 2025, from https://aws.amazon.com/what-is/eda/

Apache, K. (2020, June 25). What is event streaming. Retrieved March 15, 2025, from Kafka website: https://kafka.apache.org/intro

Auliana, S., & Nurasiah, I. (2018). Penerapan Knowledge Management pada Proses Pelaporan Data Perguruan Tinggi (Studi Kasus di STIE Bina Bangsa). *Indonesian Journal of Strategic Management*, *1*(1), 3–12. https://doi.org/10.25134/ijsm.v1i1.838

Brawijaya, H., & Widodo, S. (2023). Implementation of PDDIKTI Neo Feeder Web Service in Recording of Independent Campus Activities. *Jurnal Riset Informatika*, *5*(2), 203–210.

Bucur, V., Stan, O., & Miclea, L. (2020). An Analysis of the Implementation of Kafka in High-Frequency Electronic Trading Environments. *International Journal of Modeling and Optimization*, *10*(2), 52–56. https://doi.org/10.7763/IJMO.2020.V10.746

Darmawan, I., Rahmatulloh, A., & Gunawan, R. (2022). Web Service Modeling for GraphQL Based College Data Service Access. *2022 International Conference Advancement in Data Science, E-Learning and Information Systems (ICADEIS)*, 1–6. IEEE. https://doi.org/10.1109/ICADEIS56544.2022.10037508

Grudniak, M., & Dzieńkowski, M. (2021). REST API performance comparison of web applications based on JavaScript programming frameworks. *Journal of Computer Sciences Institute*, *19*, 121–125. https://doi.org/10.35784/jcsi.2620

Hesse, G., Matthies, C., & Uflacker, M. (2020). How Fast Can We Insert? An Empirical Performance Evaluation of Apache Kafka. *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, 641–648. IEEE. https://doi.org/10.1109/ICPADS51040.2020.00089

Köstler, J., Reiser, H. P., Habiger, G., & Hauck, F. J. (2021). SmartStream: towards byzantine resilient data streaming. *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 213–222. New York, NY, USA: ACM. https://doi.org/10.1145/3412841.3441904

Lazzari, L., & Farias, K. (2023, August 10). Uncovering the Hidden Potential of Event-Driven Architecture: A Research Agenda. https://doi.org/10.48550/arXiv.2308.05270

Ornelas, T. (2023, February). Kafkajs, a Modern Apache Kafka Client for Node.Js. Retrieved March 28, 2025, from https://kafka.js.org

Prayogi, A. A., Niswar, M., Indrabayu, & Rijal, M. (2020). Design and Implementation of REST API for Academic Information System. *IOP Conference Series: Materials Science and*

Engineering, *875*(1), 012047. https://doi.org/10.1088/1757-899X/875/1/012047

Rafiezah Rizcha, Y., & Yaakub, S. (2023). Sistem Informasi Manajemen Sumber Daya Manusia Pada Universitas Muhammadiyah Jambi. *Jurnal Manajemen Sistem Informasi*, *8*(1), 78–93. https://doi.org/10.33998/jurnalmsi.2023.8.1.765

Sharma, A. (2025, April 4). Apache Kafka: A Deep Dive into Its Architecture and Workflow. Retrieved June 15, 2025, from Medium website: https://medium.com/@minervaaniket/apache-kafka-a-deep-dive-into-its-architecture-and-workflow-510709dff298

Solace Corporation. (2021, August). Results from the Industry's First Event-Driven Architecture Survey. Retrieved March 15, 2025, from https://solace.com/event-driven-architecture-statistics

Sopingi, S., & Wulandari, S. (2023). *Integrasi Sistem Pembelajaran dengan Google Classroom melalui Google Apps Script*. *6*(2), 195–206. https://doi.org/https://doi.org/10.31764/justek.v6i2.15061

Sopingi, Setyowati, R., & Purnomo, S. (2020). Pengembangan Web Service Digital Assessment Test of English for International Communication (TOEIC). *Jurnal E-Komtek (Elektro-Komputer-Teknik)*, *4*(1), 75–90. https://doi.org/10.37339/e-komtek.v4i1.232

Ubur, S. D. (2023, March 2). Reviewing the Scope and Impact of Implementing a Modernised IT Event-Driven Architecture from Traditional Architecture using Agile Frameworks: A Case study of Bimodal operational strategy. https://doi.org/10.48550/arXiv.2303.12082

Živanović, S., Popović, M., Vorkapić, N., Pjević, M., & Slavković, N. (2020). An overview of rapid prototyping technologies using subtractive, additive and formative processes. *FME Transactions*, *48*(2), 246–253. https://doi.org/10.5937/fmet2001246Z