OPEN ACCESS
Indo-JC
Indonesia Journal of Computing
ISSN 2460-9056
socj.telkomuniversity.ac.id/indojc

Ind. Journal on Computing
Vol. 6, Issue. 3, December 2021. pp. 47-62
doi:10.34818/indojc.2021.6.3.597

Design of API Gateway as Middleware on Platform as a Service

Dita Oktaria [#1], Joel Andrew M. K. Ginting [#2], Maman Abdurohman [#3] Rahmat Yasirandi [#4]

 # School of Computing, Telkom University
Telekomunikasi 1, Ters. Buah Batu, Bandung, Jawa Barat, Indonesia

[1] dioktaria@telkomuniversity.ac.id
[2] joelandrew@students.telkomuniversity.ac.id
[3] abdurohman@telkomuniversity.ac.id
[4] batanganhitam@telkomuniversity.ac.id

Abstract

The process of building a platform consists of various stages. Until now there has been no awareness in utilizing data and information sources that have been available to be used as a basis for developing a platform that is able to improve quality a system of integrity. For this reason, Platform as a Service (PaaS) architecture was built which provides application development and deployment services to process data and information with case studies obtained from practicum activities during the lecture period based on cloud computing using Service Oriented Architectural (SOA) method. Each service that is available is loose coupling where a service can be called by another service without the calling program needing to pay attention to where the location of the service being called is and what platform is used by that service. In making this platform architecture, API gateway is a good middleware system to be used on a microservice-based platform. Analysis results prove that the architecture using API gateway as a built-in middleware can be considered to develop Telkom University lab service system. From the test results, it produces an RTT of 2.081 seconds, 45 MB of memory, and 8% CPU for each user in 100 users.

Keywords: Platform as a Service, Service Oriented Architectural, Platform, Cloud Computing, Loose Coupling

Abstrak

Proses membangun platform terdiri dari berbagai tahapan. Hingga saat ini belum ada kesadaran dalam memanfaatkan sumber data dan informasi yang telah tersedia untuk dijadikan dasar dalam mengembangkan platform yang mampu meningkatkan kualitas suatu sistem yang berintegritas. Untuk itu dibangunlah arsitektur Platform as a Service (PaaS) yang menyediakan layanan pengembangan dan penyebaran aplikasi untuk mengolah data dan informasi dengan studi kasus yang diperoleh dari kegiatan praktikum selama masa perkuliahan berbasis cloud computing dengan menggunakan metode Service Oriented Architectural (SOA). Setiap layanan yang tersedia adalah loose coupling dimana suatu layanan dapat dipanggil oleh layanan lain tanpa perlu memperhatikan program pemanggilan dimana lokasi layanan yang dipanggil dan platform apa yang digunakan oleh layanan tersebut. Dalam pembuatan arsitektur platform ini, API gateway merupakan sistem middleware yang baik untuk digunakan pada platform berbasis microservice. Hasil analisis membuktikan bahwa arsitektur dengan menggunakan API gateway sebagai built-in middleware dapat dipertimbangkan untuk mengembangkan sistem layanan lab Telkom University. Dari hasil pengujian, menghasilkan RTT 2,081 detik, memori 45 MB, dan CPU 8% untuk setiap pengguna dalam 100 pengguna.

Kata Kunci: Platform as a Service, Service Oriented Architectural, Platform, Cloud Computing, Loose Coupling

I. Introduction

Received on October, 2021. Accepted on December, 2021

CLOUD computing is a computing model. Where resources such as computing, storage, networking, and software are provided as services on the internet. Cloud computing also has several services, such as Infrastructure as a Service (IaaS), which provides storage services, Platform as a Service (PaaS), which provides platform services for building applications, and Software as a Service (SaaS) which offers the finished applications product to the client [1][2][3][4].

Cloud computing can also be applied in various ways, one of which is processing and developing practicum data. Telkom University is a university that has many laboratories and practicum activities. In a period of 6 months or once a year there is usually a lot of practicum data produced, one of which is the value of each practicum module that is carried out. But from all the data in the form of practicum values generated, until now there is no platform used to regulate each existing service so that it is able to connect and integrate with one another to work together to provide good services to every user who uses the platform. This is due to the unavailability of a platform that has middleware to make each service work with one another with integrity.

Therefore, a platform in the form of a web application was created using the PaaS concept. PaaS is a cloud-based data processing concept, which provides its users with all the functions to develop, deploy and manage services, without the burden of installing, configuring and managing middleware, operating systems, and underlying hardware [5][6][7][8].

With the existence of a platform in the form of a web application using the PaaS concept which aims to provide cloud computing-based data storage and processing services that can be accessed by every Telkom University student, a data storage container that is more efficient, low budget, and every service can be created. Provided can run well, and with the creation of a platform in the form of a web application, it is hoped that in the future both students or practicum assistants can develop existing applications or create new applications that can improve the quality and performance of each practicum laboratory at Telkom University, through service provided on the platform to be created.

The PaaS concept used to create a new platform in the form of a web application is a good step where users of the services provided by the platform are able to access available data, build applications, upload applications, perform application testing, or manage configurations that are available needed in the application development process.

II. Related Work

PaaS is a suite of cloud-based services that allows business users and developers to build applications at a speed that an on-premise solution cannot match. PaaS can take the cost and hassle out of evaluating, purchasing, configuring, and managing all the hardware and software required for custom-built applications. PaaS has both technical and business advantages. By using PaaS, system providers make most of the choices that determine how the application infrastructure operates, such as the type of OS used, API, programming language, and management capabilities [9]. With the advancement of IoT, PaaS is needed as a Middleware service provider. Until now, PaaS is one of the most needed services to develop software or hardware to be updated and improve its performance [10].

When compared with Software as a Service (SaaS) from the end-user side, PaaS is more flexible because users do not only use the applications provided but can develop an application at the same time [11][12]. With PaaS services created, an architecture can be created to build a platform that is suitable to meet the needs of the client using the Service Oriented Architectural (SOA) system, where SOA is a series of services that communicate with each other to create an application software, by reusing existing services in order to be used for new uses [13][14][15][16]. SOA is an open standards-based framework architecture that allows a company or agency to integrate data that previously existed and was only stored tightly at the headquarters of its customers, partners or suppliers. In other words, SOA is an architecture that supports business integration as a connected service and a path to innovation. In short, SOA is a paradigm for organizing and utilizing and

distributing capabilities that may be under the control of the ownership domain. With the existence of SOA that can create and build a combined architecture, distributed and scalable, through a web management system that is used to allow integration and interoperability through distributed sensor networks and the internet wireless [17].

As defined above, SOA is a way of designing an application using existing components or services. In other words, an application is built on a modular basis. Actually, this modular is not something new. Today's programming techniques, such as object-oriented programming, have put forward a modular approach in building applications. However, what makes SOA different is that these components or services are built and interact with each other freely and loosely (loose coupled).

By being loose coupled, a service can be called by other programs/services without the calling program needing to pay attention to where the location of the called service is located and what platform/technology is used by the service [18]. Loose coupling is very important for SOA because it means that a service call by another service can be done at run-time.

SOA breaks down a system functionality into smaller unit logic called services. These services are independent from each other but have the ability to interact with each other through certain communication mechanisms. SOA can be divided into four main components, namely as service, operation, process, and message [18]. SOA has several characteristics in its type of service, which are organized into two things: Service Interface and Service Implementation. The service interface states how the service can be called, such as input/output parameters and where it is located. Service implementation is how the logic of the customer lookup service is executed. Service implementation is closely related to the programming technology used. SOA does not need to care about how a service is implemented. Whether it's written in Java or Python, what matters is how the service can be called and provides information according to the service interface.

By using the SOA method, the platform to be built can simplify the complexity of integration between applications, where the platform to be built will be a service provider to meet the needs of data transactions between applications. There are several advantages that can also be enjoyed by using the SOA method on the platform to be built, as follows:

1) Service Reusability: In SOA, an application is built by combining several services and functions that are small, independent, and loosely coupled in terms of functionality. Thus services can be reused in several applications independent of their interactions with other services.

2) Easy Maintainability: Because each service is an independent entity, each service can be updated and managed easily without having to worry about other services. This means that a large and complex application can easily be managed properly.

3) Greater Reliability: SOA-based applications are more reliable because small and independent services are easier to test and debug, compared to large chunks of code in non-SOA applications.

4) Location Independence: Services are usually published in a directory where all consumers can see them. This approach allows a service to change its location at any time. However, consumers will always be able to find the service they want by searching the directory referred to above.

5) Improved Scalability and Availability: Several processes of service can run on several different servers at the same time. This can increase the scalability and availability of these services.

6) Improved Software Quality: Because a service can be reused, it will not find redundant functions. This will help reduce errors due to data inconsistencies, thereby improving the quality of the software.

7) Platform Independence: SOA facilitates the development of a complex process by integrating several products from independent vendors that differ from the platform and technology used.

8) Increased Productivity: Developers can reuse legacy applications and build additional functions without having to develop everything from scratch. This situation increases the productivity of the developers. And at the same time, it will reduce the development costs of an application substantially.

III. Service System

A. System Architecture

The system to be built in this research is for the practicum laboratory at the Faculty of Informatics Telkom University, by building a platform in the form of a web application using the PaaS service, which has the ability to store and process practicum data, and is able to make each service provided integrated and work the same in improving services on the platform that was built and being able to develop a new application using the services provided on the platform that was built.

By using the SOA method, the platform to be created can be built properly and efficiently [17]. The process carried out to build a platform in the form of a web application in this research will use components that interact freely and independently of one another, by being loose coupled a service can be used by other programs or services without the need to pay attention to the platform or technology used by the service.

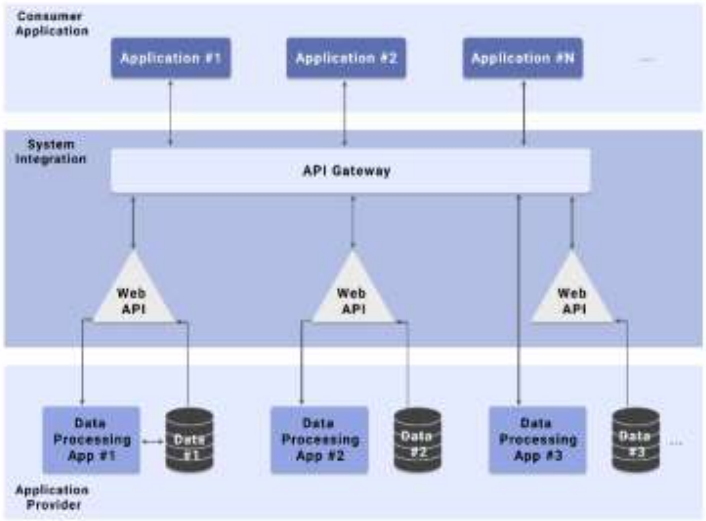

Fig. 1. Platform Architecture with API Gateway as Middleware

In Figure 1, the architecture that is built will use the API gateway as a middleware, where each service provided has its own database and is connected to one another. So that communication between the user and the service can run well. For each service provided on the platform, it will be adjusted to the needs of each Faculty of Informatics practicum laboratory.

B. System Design

The system that we want to design from this research is a platform architecture that can perform and provide service development applications through practicum data provided on the platform in the form of a web application. The services that will be provided are in accordance with Table 1.

TABLE I
SERVICE PLATFORM AND FUNCTIONS

Service Platform	Function
Allocation of Data	With the data allocation service, where the data obtained from the results of practicum implementation will be allocated to the platform, so that it can be accessed by every student who wants to know the value of each practicum that is followed.
Database Server	With the data storage server in the form of a cloud database on the platform that was built, data can be stored safely and can be used to create or develop new and existing applications.
Processing Data	Data processing service is one of the services available on the platform to be built, where this service functions to change each data input into an API so that any data entered into the database can be accessed and processed through the API.
Get Data	Data processing service is one of the services available on the platform to be built, where this service functions to convert each data input into an API so that any data entered into the database can be accessed and processed through the API.
Display Data	The large amount of data stored on the platform must be able to be displayed and seen by the user, so that the existing data can be analyzed to see the accuracy of any stored data, whether it is in accordance with the type and location of each.

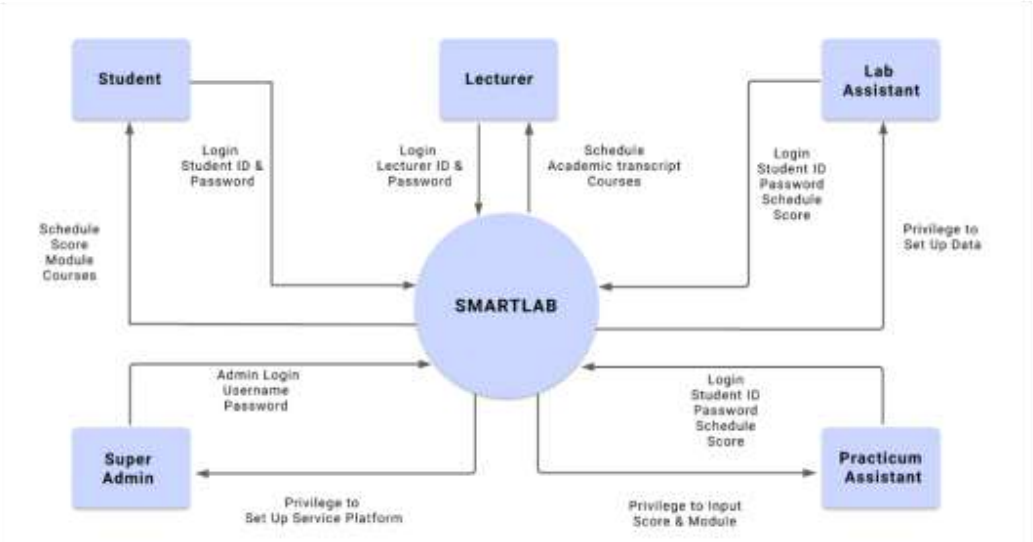

Fig. 2. Data Flow Diagram Smartlab Platform

In Figure 2, it can be seen that there are 5 users who are connected to each other on the SmartLab Platform, where each user has their respective functions and rights. With a platform architecture that is able to make every function of the 5 users, the platform that will be implemented can work optimally to carry out each function.

C. SOA and Microservices Functionally

According to Xiao et al. (2016) [19] the SOA function with the microservice application is divided based on four functionalities according to the existing SOA and microservice components. The functionality of SOA with microservice applications is illustrated in Table 2.

TABLE II
Functions and Components of SOA and Microservice

Function	Component SOA	Component (Microservice)
management center point; registered services/APIs can be searched and found	service registry, service repository	enterprise API registry; enterprise microservice repository
Where the service/proxy API is available or available for use	Enterprise service bus	API management layer (API proxy)
End point of service or microservice service	Service provider	Microservice provider
Other applications that require functionality provided by an API or service	Service consumer	Microservice consumer
Service Runtime/API monitoring and management	BAM, service monitoring	Enterprise monitoring and tracking

D. System Stakeholder

In Figure 3, we uses an integrated architecture with middleware as a liaison between the user and the services provided on the platform. Each service used on the platform has its own API service that is connected to each respective database service. Every user, namely students, lecturers, and lab assistants can access services made on the platform. The architecture below makes each service connected to one another according to the needs desired by the user.

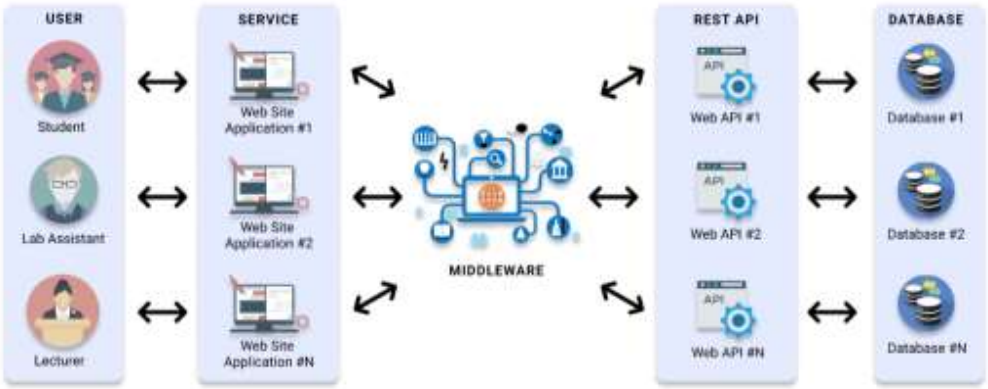

Fig. 3. System Stakeholders on the Platform

E. System Architecture Model

By implementing PaaS services, namely providing an API that is connected to each other in the middleware system using 5 appropriate modules to fulfill services in the application layer. The middleware system provides a simple, structured and well-defined infrastructure for performing data requests and notifications, physical device discovery and processing, communication problems, and rule-based data management [20]. From the services contained in the application layer, according to the related paper that has been read, there are several modules that are suitable to assist in realizing the service.

1) Resource Manager: Responsible for handling data source configuration that allows the data source infrastructure in a logical structure according to the application used [20].

2) Resource Handler: Responsible for handling data source configuration that allows the data source infrastructure in a logical structure according to the application used [20].

3) Event Handler: Responsible for data management and operations requested by data collection and operation services. In data collection is done through the middleware from one or more logical resources, which occurs with the support of the resource handler module. The data collected will be checked again so that any existing data is relevant to the application used [20].

4) Communication Service: Receiving information on data allocation, which will later inform the resource manager to realize the information obtained [20].

5) Service Manager: Responsible for managing all the functionality of each service provided, at the application layer [21].

F. Testing Scenarios

In this research, testing was carried out so that the functionality of each service in the system could run and be used. The test that will be carried out is testing the platform mechanism system in the form of a built web application. To prove that each service contained in the application layer can be run with the module used in the middleware layer. The test to be carried out includes 3 stages of testing:

1) The first test in Table 3, by implementing the architecture created, the architecture created uses the API gateway as middleware [22].

TABLE III
ARCHITECTURAL TESTING SCENARIOS

Type of Test	Architectural implementation
Pre-Condition	Creating a platform architectural design
Criteria	Determine a good architecture to build a laboratory service system for practicum
Test Procedure	Implement the architecture and view the communication process on each service in both architectures
Expectation	Get a good architectural form for practicum laboratory services

2) After obtaining a good and appropriate architecture in building a laboratory service system, it is followed by a second test, ensuring that every available service can be integrated with each other and meet the microservice components in Table 2. by using API Gateway as the middleware platform [23]. The test scenario carried out is shown in Table 4.

TABLE IV
MICROSERVICE COMPONENT TESTING SCENARIOS

Type of Test	Microservice component testing
Pre-Condition	Build a microservice-based platform
Criteria	The platform built meets the microservice component
Test Procedure	Each component listed in Table 2 is available on the platform
Expectation	The built platform meets every microservice component listed in Table 2

3) Compatibility testing, scalability testing, connectivity testing, and reliability testing [24] [25]:

- The first test in Table 5, namely compatibility testing, this test focuses on the level of possible service from the platform to be accepted by various types of programming languages used by the user. The test procedure will be tested through several programming languages used to access services on the platform [24].

TABLE V
COMPATIBILITY TESTING SCENARIO

Type of Test	Compatibility testing scenario
Pre-Condition	Server status is running
Criteria	User requests service according to the programming language used
Test Procedure	Each server service will request user service
Expectation	Service requests can be accessed through several programming languages

- The second test in Table 6 is a performance of scalability testing, this test aims to see the resilience of the system by testing the network performance in data distribution [24].

TABLE VI
PERFORMANCE SCALABILITY TESTING SCENARIOS

Type of Test	Performance scalability testing
Pre-Condition	Server status is running
Criteria	See the value of RTT (Round Time Trip), Memory, and CPU usage
Test Procedure	Increase the number of users accessing the service
Expectation	The average value that can be obtained will increase gradually as the user increases with an increase in the average value that will not change too significantly at each test

- The third test is connectivity testing in Table 7, this test aims to find out how the response received by the user if there is a problem on the connectivity platform used [25].

TABLE VII
CONNECTIVITY TESTING SCENARIOS

Type of Test	Connectivity testing
Pre-Condition	Server status is running
Criteria	The connection on the server has been disabled
Test Procedure	The server that was previously running immediately shuts down
Expectation	All existing services cannot be accessed

- The fourth test is the reliability testing in Table 8, this test aims to find out that each service can be used in any situation [25].

TABLE VIII
RELIABILITY TESTING SCENARIO

Type of Test	Reliability testing
Pre-Condition	Server status is running
Criteria	The service is running and the system is restarted/reloaded
Test Procedure	When the user is accessing the platform the system restarts/reloads
Expectation	The still system can be used when the server performs restart / reload process

IV. RESULTS AND DISCUSSION

A. Architectural Testing Results

1) Comparison of 2 Architectural Testing Results: In Figure 4, it can be seen with the middleware on the platform architecture built using the API gateway, where the Web API is a service provider and a liaison for each user. API gateway functions to integrate each service on the platform so that it can be accessed by users. API gateway will manage the flow of communication nets between users to each existing service according to

the needs of the user itself [22]. With the API gateway, access by multiple users can be controlled because every user who wants to access the services provided on the platform must go through the API gateway first, then the API gateway will direct each user request to the service that is available regularly and efficiently [22].

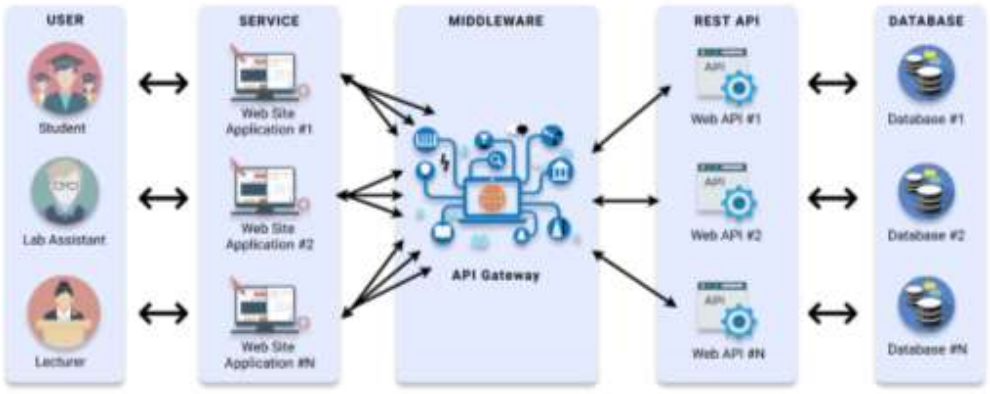

Fig. 4. Platform Architecture Implementation with API Gateway as Middleware

2) Microservice Component Testing Results: In the microservice components testing, the built architecture must meet the component standards listed in Table 9.

TABLE IX
MICROSERVICE COMPONENT TESTING

Function	Component (Microservice)	Status
management center point; registered services/APIs can be searched and found	enterprise API registry; enterprise microservice repository	Fulfilled
Where the service/proxy API is available or available for use	API management layer (API proxy)	Fulfilled
End point of service or microservice service	Microservice provider	Fulfilled
Other applications that require functionality provided by an API or service	Microservice consumer	Fulfilled
Service Runtime/API monitoring and management	Enterprise monitoring dan tracking	Fulfilled

- Enterprise API Registry
  Each service on the platform already has its API, which is registered and can be searched or found as seen in Figure 5. We use Swagger as a design for each existing API to be registered and usable.

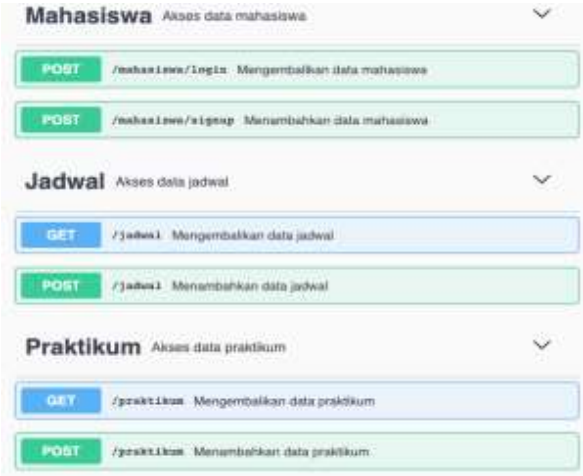

Fig. 5. Enterprise API Registry

- API Management Layer (API Proxy)
  In the architecture that was built, the API management layer used is the API gateway as a middleware that is tasked with managing and distributing every user request for services provided on the platform. API gateway will set the route for each existing service request according to user needs.
- Microservice Provider
  In Figure 6, each available service already has its endpoint according to Table 10:

TABLE X
ENDPOINT FOR EACH SERVICE

Endpoint	Information
/college student	Endpoint for using student services
/schedule	Endpoint for using the schedule service
/practice	Endpoint for using practicum services

- Microservice Consumer
  We created three applications, each of which uses the programming languages PHP, Python, and Golang. By utilizing all API services available through the API gateway that has been provided.
- Enterprise Monitoring and Tracking
  The platform that is built can be continuously monitored using the monitoring tools in Figure 6, to view information such as run time, memory usage, server status, CPU usage, and incoming service requests.

Fig. 6. Monitoring Results on the Server Platform

3) Results for Compatibility, Scalability, Connectivity, and Reliability Testing: In this test, we carried out several testing stages which will be explained per-point as follows:

- Compatibility Testing
  The results of the tests conducted show that the platform created can be accessed with several different programming languages.

TABLE XI
RESULT OF COMPATIBILITY TESTING

Type os Test	Architectural implementation
PHP (1)	Yes
Python (2)	Yes
Golang (3)	Yes

Fig. 7. Display Platform (1), (2), and (3) that can be Accessed with Several Different

It can be seen in Figure 7, we made three applications using different programming languages, namely PHP, Python, and Golang. Applications are made by utilizing every service available on the platform.

- Performance Scability Testing

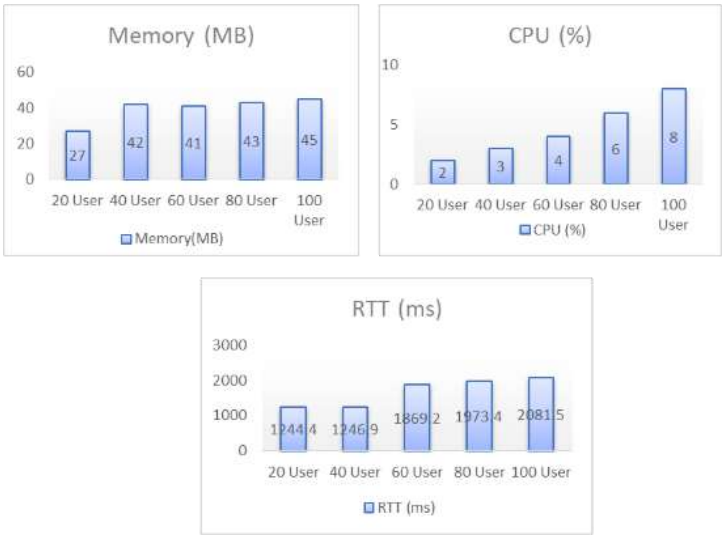

Fig. 8. Performance Scalability Testing Results

The test scenario can be seen in Table 6. The results of performance scalability testing are divided into 3 results in the form of RTT (Round Time Trip), Memory, and CPU usage. From these three results, it can be seen that the increase in the average value at each additional user for a multiple of a certain number will not change too significantly. RTT aims to see the travel time of connection requests between each user on the available services. The resulting memory and CPU usage also

display an increase in the average value that is not too far away when the number of users accessing the platform increases to a certain multiple of numbers. To get these three values, we use the Httping tool on each service simultaneously. In testing the performance scalability testing, the specifications of the server device that are running are shown in Table 12:

TABLE XII
SERVER DEVICE SPESIFICATIONS

Server Spesification	
Type	MacBook Pro (13-inch, Mid 2012)
Processor	2.5 GHz Dual-Core Intel Core i5
Memory	4 GB 1600 MHz DDR3
Disk	Macintosh SSD

It can be seen in Figure 8 when the number of users has increased gradually the resulting RTT value for each user for 100 users is still 2.081 seconds. The travel time category is quite good for a network that is run on a server with the specifications in Table 10. The memory usage generated by each user for 100 users is 45 MB. From the total memory capacity in Table 10, the memory usage generated when a user accesses the server is quite small and the platform can still be accessed normally. The resulting CPU usage per user for 100 users is 8%. From the value of CPU usage, platform usage is still running normally and can be accessed properly.

- Connectivity Testing

In accordance with the test scenario in Table 7, the expectation is every service available on the platform cannot be run. When the server is running in accordance with Figure 9 and the platform that is made can be accessed in Figure 10, then the server that was initially turned on is suddenly shut down in Figure 11. By turning off the server on the platform created, it makes any running service unable to accessible and the platform cannot be used as in Figure 12.

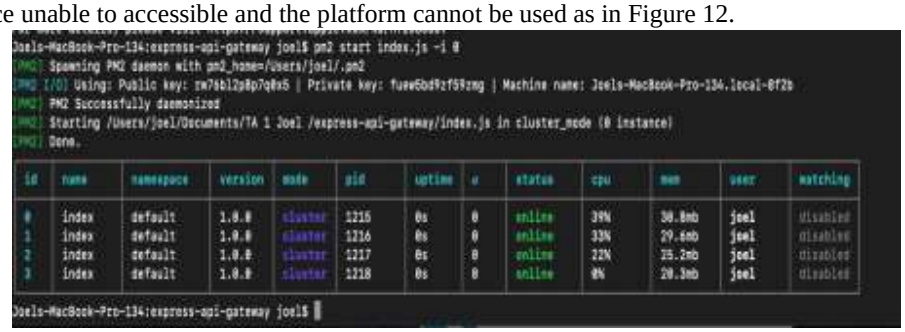

Fig. 9. The Results of the Server Status Test are Running

Fig. 10. Platform Test Results Can be Accessed

Fig. 11. The Test Results of the Server are Shut Down

Fig. 12. Platform Test Results are not Accessible

- Reliability Testing
  In accordance with the test scenario in Table 8, the expectation is that every service available on the platform can still be run even though it is restarted or reloaded on the server. When the server is running in accordance with Figure 13, then the server that is running will be restarted or reloaded in accordance with Figure 14 by restarting or reloading the server platform that is made, will not significantly affect the platform that is being accessed by the user as in Figure 15.

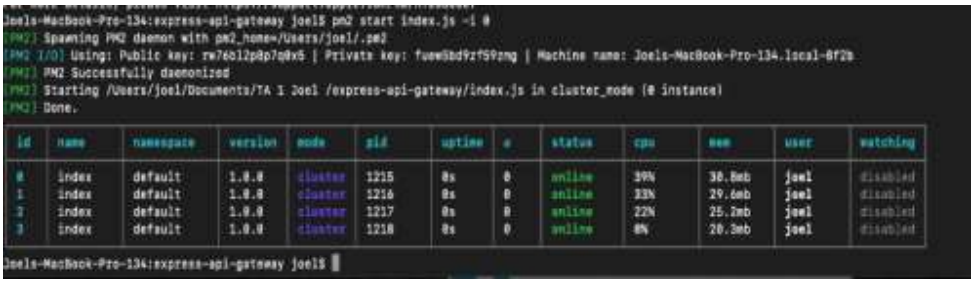

Fig. 13. The Results of the Running Server Test

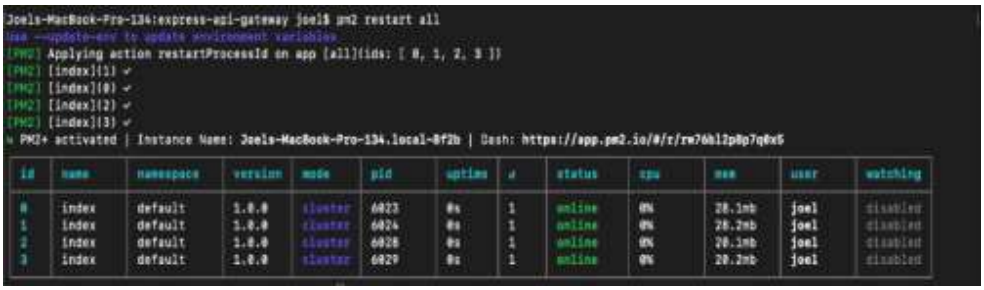

Fig. 14. Test Results Restart/Reload on the Server

Fig. 15. Platform Test Results While the Server is Restarting/Reloading

B. Analysis of Architectural Testing Results

In the test results, we created a platform architecture using the API gateway as middleware. The architecture that is built is good architecture and suitable for use in building a laboratory service system at Telkom University.

On the platform architecture, it can be seen that each existing service uses its respective API, where the API gateway is in charge of middleware in directing user needs to each existing service so that communication between the user and each service provided can run properly according to the user want. The gateway API is responsible for determining the routing of user requests for each service, composition, and translation of each protocol. All service requests from the user must go through the API gateway first, then they are routed according to the appropriate service. API gateway will handle each user request using the existing microservice and combine the results.

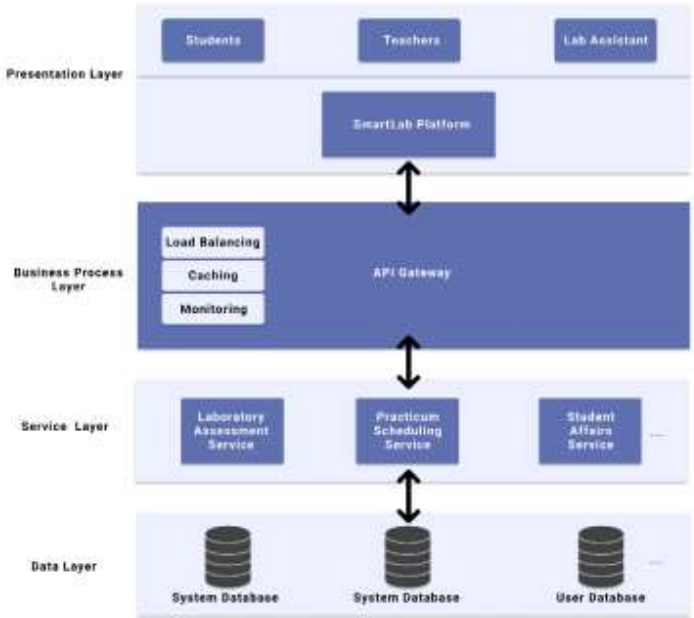

Fig. 16. Platform Architecture Model with API Gateway as Middleware

With the API gateway as the middleware in Figure 16, each service is integrated. The user does not have to know how to communicate between each service, the user only needs to choose what service is needed so that the API gateway will create a communication route for each existing service so that the user gets the desired results. The functions of the API Gateway on the platform that we make are:

1) Load Balancing: Load Balancing can be used to balance the load on each service access on the platform by distributing various workloads to compute resources. So that there is no accumulation of workload on just one resource.

2) Caching: Caching is a useful mechanism for storing frequently accessed data in temporary storage that has fast access times, to improve the performance of the platform being built.

3) Monitoring: Monitoring is a function that is useful for viewing information such as run time, memory usage, server status, CPU usage, and incoming service requests.

In the security aspect, with the API gateway, every user who wants to access the services provided on the platform must go through the API gateway first, then the API gateway will direct each user request to the service. It authenticates that a particular consumer has permission to access the API, using a predefined set of credentials. Basic Authentication checks the username-password combination against the Authorization and Proxy-Authorization request headers, which hold the credentials required to authenticate the user with the server.

Moreover, this middleware increasing the business process of the study activity in laboratory. In the previous condition, practicum handled by actors that using several silos system. This condition often creates several problems in study activity because of data duplication, inconsistent data, and slow progress of data update because some of them still need to be updated manually. However, with this middleware, the existing service will be able to connect and integrate with one another to work together to provide good services for the study activity in laboratory. For example, in the current system, a lecturer will need to ask through an email to the laboratory assistant to get the grades of the students or need to wait until the head of laboratory assistant send the grades report to the lecturer, and then the lecturer will manually enter the grade to the grading system. This middleware will increase the business process by integrating the grading system of the laboratory assistant as a service provider to provide for the lecturer's grading system as the service consumer. Thus, lecturer can actively show the grading progress for the students so that they can monitored their study performances.

This study is focusing on technical aspects of the architectural design to use API Gateway as middleware. The study in information management aspect should be explored more, start from defining the business process, decomposing business proses until getting good design of IT services that interact in the system. Hopefully the detailed of the business process integration could help the people in Academic Management to adjust in academic management process.

V. Conclusion

From testing the architectural design that was built, the following conclusions were obtained. From performance scalability testing on the platform architecture that was built, it resulted in RTT 2.081 seconds, 45 MB memory, and 8% CPU usage. The results obtained make the architecture built using the API gateway as a middleware to be considered to develop a Telkom University practicum laboratory service system. Despite adjustments in resources and needs, the purpose of building this architecture has generally been realized. Based on the results of tests that have been done with an architecture that uses the API gateway as middleware. It can be concluded that an architecture that uses an API gateway as a middleware can provide benefits such as monitoring, caching, and offers flexibility in the use and management of services on the platform.

References

[1] Andi, "Kupas Tuntas Bermacam Aplikasi Generasi Cloud Computing," Semarang: Wahana, Indonesia, 2011.

[2] S. S. Yadav and Z. W. Hua, “Cloud: A Computing Infrastructure on Demand,” in 2nd International Conference on Computer Engineering and Technology, vol. 1, 2010, pp. 423-426, doi: 10.1109/ICCET.2010.5486068.
[3] S. Namasudra, P. Roy, and B. Balusamy, "Cloud Computing: Fundamentals and Research Issues," in Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM), pp. 7-12, 2017, doi: 10.1109/ICRTCCM.2017.49.
[4] I. Odun-Ayo, M. Ananya, F. Agono, and R. Goddy-Worlu, "Cloud Computing Architecture: A Critical Analysis," in 18th International Conference on Computational Science and Applications (ICCSA), pp. 1-7, 2018, doi: 10.1109/ICCSA.2018.8439638.
[5] C. A. Ardagna, E. Damiani, F. Frati, D. Rebeccani, and M. Ughetti, "Scalability Patterns for Platform-as-a-Service," in IEEE Fifth International Conference on Cloud Computing, pp. 718-725, 2012, doi: 10.1109/CLOUD.2012.41.
[6] M. Boniface, et. al, "Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds," in Fifth International Conference on Internet and Web Applications and Services, pp. 155-160, 2010, doi: 10.1109/ICIW.2010.91.
[7] J. Gibson, R. Rondeau, D. Eveleigh, and Q. Tan, "Benefits and Challenges of Three Cloud Computing Service Models,", in Fourth International Conference on Computational Aspects of Social Networks (CASoN), pp. 198-205, 2012, doi: 10.1109/CASoN.2012.6412402.
[8] A. Yousif, M. Farouk, and M. B. Bashir, "A Cloud Based Framework for Platform as a Service," in International Conference on Cloud Computing (ICCC), pp. 1-5, 2015, doi: 10.1109/CLOUDCOMP.2015.7149621.
[9] G. Lawton, "Developing Software Online with Platform as a Service Technology," in IEEE Computer Society, vol. 41, no. 6, pp. 13-15, 2008, doi: 10.1109/MC.2008.185.
[10] L. Carlson, “A Praticial Guide to Coding for Platform-as-a-Service,” Gravenstein Highway North: O'Reilly Media, Inc., 2013.
[11] G. Kulkarni, P. Khatawkar, and J. Gambhir, "Cloud Computing-Platform as Service," International Journal of Engineering and Advanced Technology (IJEAT), vol. 1, no. 2, pp. 6, 2011.
[12] A. Singh, S. Sharma, S. R. Kumar, and S. A. Yadav, "Overview of PaaS and SaaS and Its Application in Cloud Computing," in International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH), pp. 72-176, 2016, doi: 10.1109/ICICCS.2016.7542322.
[13] K. C. Laudon and J. P. Laudon, “Management Information Systems,” Upper Saddle River, New Jersey: Pearson Education, Inc., 2012.
[14] M. Mohammadi and M. Mukhtar, "Service-Oriented Architecture and Process Modeling," in International Conference on Information Technologies (InfoTech), pp. 1-4, 2018, doi: 10.1109/InfoTech.2018.8510730.
[15] N. Serrano, J. Hernantes, and G. Gallardo, "Service-Oriented Architecture and Legacy Systems,", in IEEE Software., vol. 31, no. 5, pp. 15-19, 2014, doi: 10.1109/MS.2014.125.
[16] W. W. Dai, V. Vyatkin, and J. H. Christensen, "The Application of Service-Oriented Architectures in Distributed Automation Systems," in IEEE International Conference on Robotics and Automation (ICRA), pp. 252-257, 2014, doi: 10.1109/ICRA.2014.6906618.
[17] K. Avila, P. Sanmartin, D. Jabba, and M. Jimeno, "Applications Based on Service-Oriented Architecture (SOA) in the Field of Home Healthcare," Sensors (Basel, Switzerland), vol. 17, no. 8, 2017, doi: 10.3390/s17081703.
[18] T. Erl, “Service-Oriented Architecture: Concepts, Technology, and Design,” USA: Prentice Hall International, 2005.
[19] Z. Xiao, I. Wijegunaratne, and X. Qiang, "Reflections on SOA and Microservices," in IEEE 4th International Conference on Enterprise Systems (ES), Melbourne, VIC, Australia, pp. 60-67, 2016, doi: 10.1109/ ES.2016.14.
[20] L. A. Amaral, E. d. Matos, R. T. Tiburski, and F. Hessel, "Cooperative Middleware Platform as a Service for Internet of Things Applications," in Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, pp. 488-493, 2015, doi: 10.1145/2695664.2695799.
[21] B. Eom, C. Lee, C. Yoon, H. Lee, and W. Ryu, "A Platform as a Service for Smart Home," International Journal of Future Computer and Communication, vol. 2, no. 3, 2013, doi: 10.7763/IJFCC.2013.V2.162.
[22] B. L. Putro and Y. Rosmansyah, "Functionality design of enterprise service bus (ESB) as middleware on the smart educational service computing system platform," in International Conference on Information Technology Systems and Innovation (ICITSI), Bandung, Indonesia, pp. 355-360 2017, doi: 10.1109/ICITSI.2017.8267970.
[23] J. Wang, W. Wang, and Q. Zhu, "Design and Implementation of WeChat Mini Program for University Dormitory Based on SOA," Journal of Physics: Conference Series, vol. 1069, p. 012086, 2018, doi: 10.1088/1742-6596/1069/1/012086.
[24] R. Yasirandi, A. Rakhmatsyah, and R. Alifudin, "Perancangan Arsitektur Sistem Digital Signage secara Terpusat pada Negara Berkembang," Techno.COM, vol. 18, pp. 145-153, 2019, doi: 10.33633/tc.v18i2. 2304.
[25] M. Pallot and K. Pawar, "A holistic model of user experience for living lab experiential design," in 18th International ICE Conference on Engineering, Technology and Innovation, Munich, Germany, pp. 1-15, 2012, doi: 10.1109/ICE.2012.6297648.