2021 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS)

978-1-6654-3709-7

Microservices-based IoT Monitoring Application with a Domain-driven Design Approach

Alam Rahmatulloh
Department of Informatics
Siliwangi University
Tasikmalaya, Indonesia
alam@unsil.ac.id

Dewi Wulan Sari
Department of Informatics
Siliwangi University
Tasikmalaya, Indonesia
dewiws27@gmail.com

Rahmi Nur Shofa
Department of Informatics
Siliwangi University
Tasikmalaya, Indonesia
rahmi.shofa@unsil.ac.id

Irfan Darmawan
Department of Information System
Telkom University
Bandung, Indonesia
irfandarmawan@telkomuniversity.ac.id

2021 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS) | 978-1-6654-3709-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICADEIS52521.2021.9701966

***Abstract*—The growth in the use of the Internet of Things (IoT) is increasingly massive. Along with the continuous development of the IoT platform, there are obstacles in the number of nodes that continue to increase. In addition, there are growing issues of availability, scalability, and functionality of applications that will lead to dead code. Microservices architectural pattern emerges as an alternative. However, the service decomposition process and data management on services in microservice applications require special attention. Based on the issues described above, in this study, the microservices architecture paradigm with a domain-driven design (DDD) approach is applied to develop an IoT Monitoring application that can handle various IoT projects on one platform. The results show that a definition of a service is designed to be more accurate. The application of the DDD concept in breaking down application services helps in mapping each domain. Therefore, it can produce adaptive software products and generate easy-to-maintain code. The microservice architecture with a REST API-based approach applied to the IoT monitoring application has worked well, tested at the unit testing, integration, and performance stages. Based on performance testing results, the number of nodes (with three sensors per node) that can access simultaneously reaches 75 nodes. The total sensors in one node can have up to 10 sensors per node with a response time of less than 100ms. System development can be done without overhauling the entire system and does not interfere with the performance of other services.**

Keywords—Domain-Driven Design, Internet of Things, Microservices, Monitoring

I. INTRODUCTION

Currently, the growth in the use of the Internet of Things (IoT) is increasingly massive. During the COVID-19 pandemic, there were more connections to IoT devices than non-IoT devices. According to current estimates, there will be 30.9 billion interconnected IoT devices by 2025 [1]. The development of hardware and information technology has accelerated the deployment of billions of interconnected, intelligent, and adaptive devices in the critical infrastructure of various fields [2]. The IoT has proven its potential to transform society and has attracted the attention of both academia and industry.

The characteristic of IoT service is real-time data change. However, research [3] creates gaps or technical challenges such as heterogeneity of devices, networks, and operating systems, interoperability between applications and services, scalability, and continuous integration.

Furthermore, along with the continuous development of the IoT platform, there are obstacles in the number of nodes that continue to increase. Moreover, an increasing number of application functional and availability issues lead to dead code as application code grows more extensive and more complex, resulting in applications that are difficult to maintain properly. [4]–[6]. As a result, a method to improve system performance is required to overcome this issue.

The solution proposed in this study is to deal with performance requirements and system availability in IoT device monitoring applications. Microservices architectural pattern emerges as an alternative for the evolution of monolithic IoT monitoring apps. Microservices consist of small parts that handle specific activities and communicate through a light mechanism [7]. However, the service decomposition process and data management on services in microservice applications require special attention. An approach that can facilitate the definition of services to be more accurate is Domain-Driven Design (DDD) [8]. In previous research, no one has applied the DDD approach to IoT monitoring applications case study. The strategic part of DDD integrates the activities of software development teams with the business's goals. It aids in determining what to concentrate on by identifying one core domain.

The majority of pre-existing IoT platforms are designed for specific and limited uses. In contrast to previous studies [9]–[12], a platform for monitoring multi-project IoT devices will be developed in this study. The platform is capable of handling multiple IoT device monitoring projects in one platform. The IoT platform in this study can visualize sensor data in real-time and analyze data for a certain period. Moreover, the platform provides an open API to achieve integration of more applications quickly. The platform is expected to support interoperability and have a better capacity that can accommodate heterogeneous IoT devices.

II. RELATED WORK

Several previous studies that have tried to implement the microservices architecture on the IoT platform include

research [9], which has developed a parking system by breaking the system into several small services by applying the API gateway pattern with a database per service. This research has carried out system testing using the black-box method and performance testing on parking machines. However, the performance testing of several requests simultaneously on the system server has not been carried out. In addition, there was no comprehensive explanation of the decomposition service technique in this study. Research [10] developed an IoT platform using a microservices architecture approach and serverless computing. The use of microservice architecture can improve the scalability and reusability of the IoT platform. However, in this study, the IoT monitoring platform was only focused on the agricultural sector. Finally, research [13] builds an IoT platform by applying microservice architecture and Object-Oriented Analysis and Design (OOAD) to service solutions. The test results show that eight of the nine tested endpoints were completed with a success rate of 100%, while one endpoint was completed with a success rate of less than 50%.

Although there are no definite rules regarding service solutions in the design of a system, data management in services requires special attention because it can be one of the challenges and become an obstacle to the system in the future [6]. Therefore, the design stage in applying microservice architecture in IoT monitoring applications needs to be done carefully. In contrast to the previous research mentioned, the focus of this research will be on implementing a microservices architecture with an API Gateway database per service pattern using a lumen micro-framework with the implementation of a Domain-Driven Design (DDD) approach to service solutions at the system design stage. An analysis of the effect is carried out—implementation of microservice architecture along with DDD. The DDD approach is to design applications divided into different contexts according to the domain expert's view. Systems with microservices architectures that do not apply the concept of DDD will be more challenging to determine how the system should be broken down and challenging to define how small the service should be [14].

III. Methodology

The research stages consisted of five main stages: problem identification, data collection with literature studies and interviews, implementation stage by implementing agile software development methodologies, evaluation, and concluding as shown in Figure 1. In addition, to quickly adapt to changing needs, there are five stages: requirements analysis, design, system implementation (coding), testing, and deployment.

A. Identification of Problems

Along with the continuous development of the Internet of Things platform [15]–[18], there are obstacles in the number of nodes that continue to increase. Therefore, a way is needed to improve system performance to overcome this. In addition, more availability, system scalability, and functional application problems lead to dead code when the application code gets more extensive and more complex, causing applications that cannot be adequately maintained. In addition, the process of solving services and managing data requires special attention because it can become one of the challenges and become an obstacle to the system in the future.

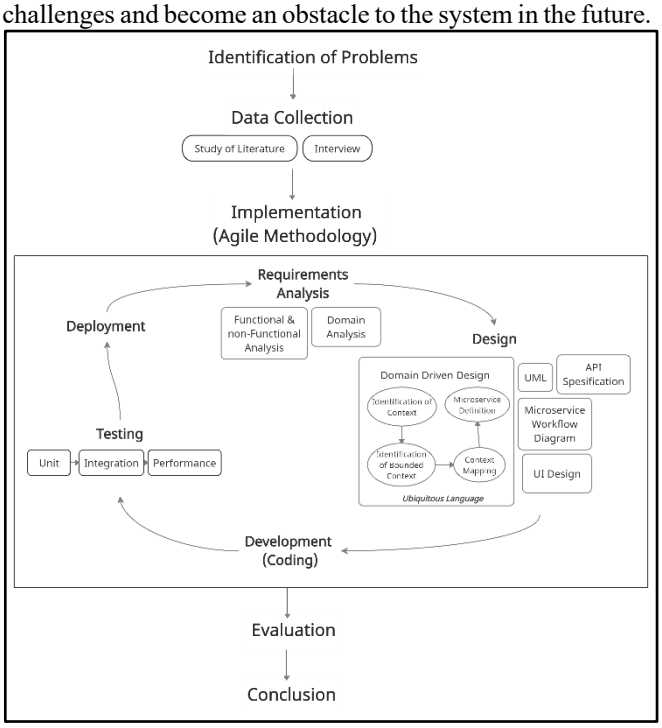

Fig. 1 Methodology

B. Data Collection

The data collection stage was carried out by conducting a literature study and interview methods. In the literature study stage, reference studies are carried out from books, national and international journals between 2016 and 2021, and documents related to research. We select 44 primary studies from 135 potentially relevant papers. In addition, interviews were also conducted with domain experts at the research location, namely Synapsis.id, a startup engaged in the Internet of Things, intended to get an accurate picture of research problems and the system to be built. We conducted at least three in-depth interviews with software professionals at Synapsis.id.

C. Implementation

The implementation phase in this research uses an agile software development methodology. According to research [19], an agile methodology is fast in adapting to changing needs. The Agile method is most suitable for tight deadlines and small budgets.

IV. Result and Analysis

A. Analysis of Requirements

There are business process identification, functional and non-functional analysis, and domain analysis during the requirements analysis phase.

1. Business Process Identification

The business process or flow in the IoT monitoring application that will be built starts with the user registering. Next, the user creates a project, groups it into groups, and registers devices and sensor types. After that, the user can start sending and retrieving sensor data from the IoT platform. Finally, monitoring incoming sensor data is possible.

2. Functional Analysis

The primary functional analysis in the to-be-built IoT monitoring application is a system capable of performing user management, IoT project management, group management, management of IoT devices and their sensors, capable of receiving data from sensors and processing sensor data, also able to display sensor data in the form of visualizations. Graphically or in tabular form so that end-users can understand both in real-time and over a specific time based on the user's preferences. The platform is also providing an open API for third-party applications to monitor specific IoT projects.

3. Non-Functional Analysis

Non-functional requirements needed include usability, the IoT monitoring application has a display that is easy to understand by users, equipped with easy-to-understand icons, and alerts or information for users when doing something such as a question when going to delete data as information. Status when adding or updating data. In terms of portability, IoT monitoring applications can be run on devices with a minimum available RAM of 2GB, installed a web browser application, available Wifi / LAN network to access the application server. In terms of reliability, there is user authentication with passwords and available levels of users with different functional requirements. There is security in the database equipped with passwords, and there is middleware in every microservice application to prevent unauthorized requests or requests from outside.

4. Domain Analysis

The main thing that must be done in domain-driven design is defining the domain in the system. Domain refers to the subject area where the application will be applied. In this study, the domain in question is the IoT monitoring application or IoT platform.

B. Design

At this stage, the design stage is carried out by applying a domain-driven design (DDD) approach [8], [20], in which there is a context identification process, bounded context identification, context mapping, and service definition. However, the interaction between domain experts and developers is often hampered because of fundamental differences in communication. For example, domain experts are generally not familiar with programming concepts like developers. Therefore, we need a unifying language where everyone involved can understand the terms or language used in the system development, known as Ubiquitous Language in the application of DDD [21].

After the domain-driven design stage has been completed, the next step is to design the Unified Modeling Language (UML) [22], including use case, sequence, class, and deployment diagrams. Use case diagram used to model systems created with object-based concepts. Sequence diagrams describe interactions that detail how an operation is performed. Class diagrams show system classes and their logical relationships. Deployment diagrams describe the various processes in the system and how the relationships that exist in these processes. Next, making microservices workflow diagrams and designing API specifications to be used. They are finally designing the interface or system display.

1. Identify Context

At this stage, similar functions are grouped into the same context. Furthermore, the domain is broken down into subdomains which are logically differentiated based on the model and functional application of the results of the analysis of similar functional requirements, which is called context. Based on the results of context identification, the IoT monitoring application consists of 6 contexts listed in table 1.

TABLE I. CONTEXT IDENTIFICATION OF IOT MONITORING APPLICATIONS

No	Nama *Context*
1	User
2	Project
3	Group
4	Node
5	Sensor
6	Data

2. Identify Bounded Context

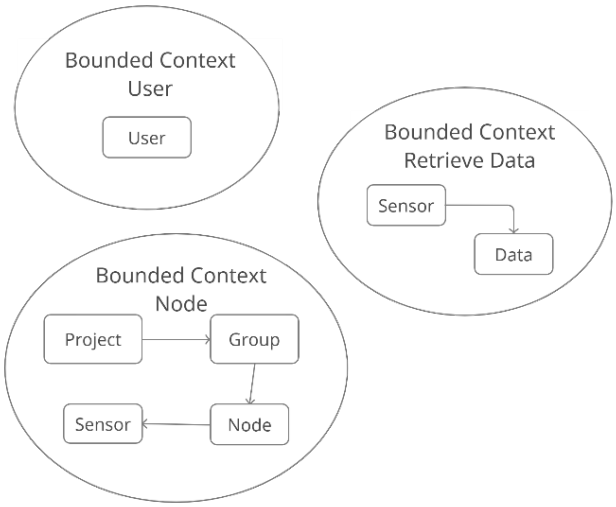

Fig. 2 Identification of the Bounded Context of IoT Monitoring Applications

Identification of bounded context determines the boundaries contained in the context [23]–[25]. One bounded context will be a small service and is the beginning of microservices formed [26], [27]. Microservices are derivatives of stateless web services [28], [29]. There are three bounded contexts identified. First, the bounded context of the user consists of one context that is the user context. Its functionality is closely related to user management functions. Second, the bounded context of node consists of Project, Group, Node, and Sensor contexts. Its functionality is to handle master data. Finally, the bounded context retrieves data consists of context Sensors and Data, which have functionality directly related to handling incoming data from the device. The description of the identification of the bounded context of the IoT monitoring application can be seen in Figure 2.

3. Context Mapping

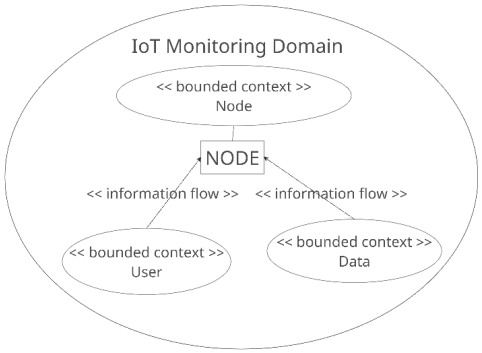

Fig. 3 Context Mapping (Context Mapping) IoT Monitoring Application

4. Definition of Service

Based on bounded context identification and context mapping, the IoT monitoring application service solution is defined into three services: user, node, and data. User-service consists of a user context in which there are user data management and authentication functionalities. Node-service consists of project context, context group, context node, and context sensor, in which there is a project data management function, group, nodes (devices), and sensors. Finally, the data service consists of context sensor data. There is a function to receive sensor data from the device directly, process it, and then send data to the client for monitoring and analysis. The three services identified by the bounded context are independent and perform their respective functional tasks.

5. Use Case Diagram

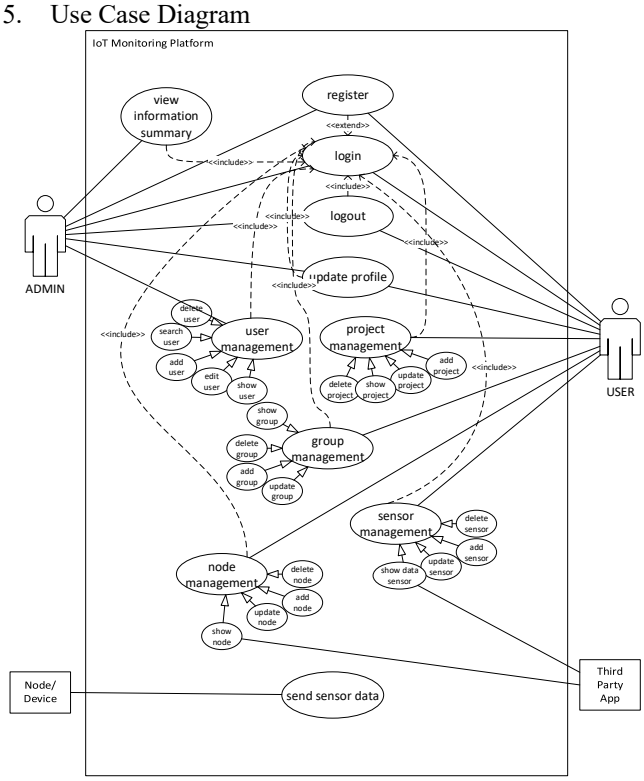

Fig. 4 IoT Monitoring Application Use Case Diagram

The use case diagram shown in Figure 4 describes the interaction between actors, namely admins and guest users, devices or nodes, or third-party applications with IoT monitoring applications created.

6. Class Diagram

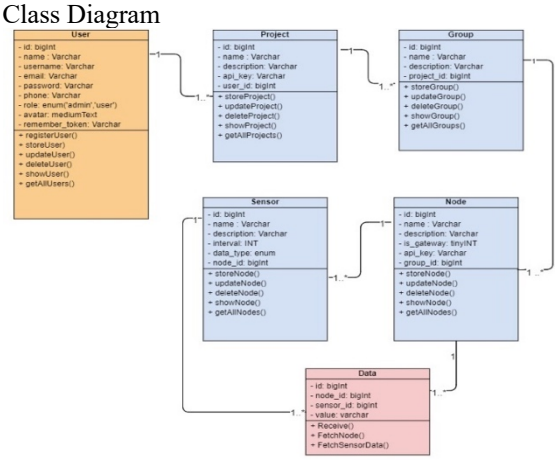

Fig. 5 *Diagram* Class *Aplikasi* Monitoring *IoT*

7. Sequence Diagram

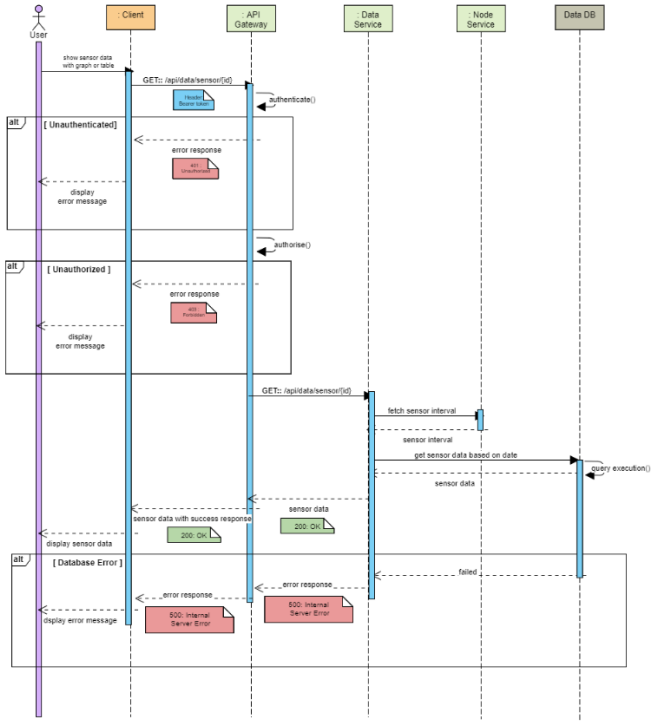

Fig. 6 Sequence Diagram showing sensor data to the client on IoT Monitoring Application

8. Deployment Diagram

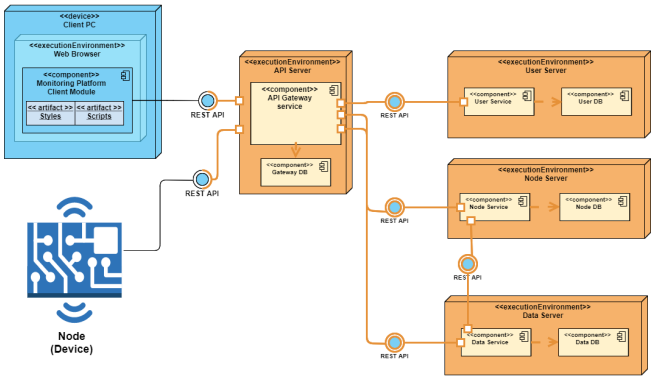

Fig. 7 IoT Monitoring Application Deployment Diagram

9. Microservice Architecture Workflow Diagram

The workflow diagram of the microservice architecture in the IoT monitoring application can be seen in Figure 8. Clients and IoT devices do not communicate directly with the service master on the back-end, but clients and IoT devices communicate through the service gateway. Client communication with the service gateway and communication between services uses a REST API with JavaScript Object Notation (JSON) format because the code structure of the JSON format is more concise and easier to understand. At the service gateway, Lumen Passport is implemented to authenticate user and node or device logins. The communication between services requires a secret key whose configuration is stored in each service master to avoid requests from outside the service that is not recognized. As a result, the service master, namely service users, service nodes, and service data, can be more secure.

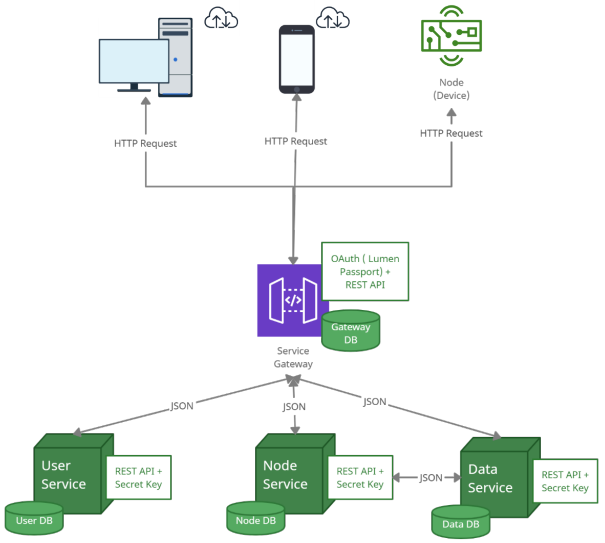

Fig. 8 IoT Monitoring Application Microservice Workflow Diagram

10. API Specification Design

Having an API specification in a team development environment can speed up the team's integration between back-end developers and front-end developers. Front-end developers do not need to wait to complete the back-end developer's work because the API specification has been agreed upon. A total of 43 URI or endpoint specifications were created. The following are some API specifications for IoT monitoring applications, as shown in Table 2.

TABLE II. API SPECIFICATION DESIGN OF IOT MONITORING APPLICATION

Information	*Method*	URI
Displays a list of all users	GET	/api/user
Displays a list of all projects	GET	/api/project
Displays a list of all groups	GET	/api/group?project_id={project_id}
Displays a list of all nodes	GET	/api/node?group_id={group_id}
Displays a list of all sensors	GET	/api/sensor?node_id={node_id}
Receive sensor data from device	POST	/api/data
Retrieve sensor data values	GET	/api/data/sensor/{id}

C. System Implementation

The system is built using a microservice architecture by implementing an API-driven communication pattern. Each service in developing the IoT monitoring application back-end is built using the Lumen micro-framework v.8.0. with a minimum development environment specification using PHP v.7.3. The client or front end is made web-based with the Vue JS framework. Hardware and Software specifications used in System Development are listed in table 3.

The services built on the IoT monitoring application consist of service gateways, service users, service nodes, and service data. The service gateway acts as a request traffic gateway to all services. Every request from the client must go through the service gateway, so the route for all services must be registered at the service gateway. When there is a request from the client, the service gateway will validate the request and then forward the request to the service concerned. In addition to request validation, the service gateway plays a role in handling API authentication. The service user is the service master to handle requests or requests from the service gateway to manage user data to the database. Data is sent to the service gateway in a REST API using the HTTP Request protocol. A service node is a service master whose role is to handle the project, group, node, and sensor data management. Finally, service data is a service that functions to manage sensor data, including receiving sensor data from the device, processing it, and producing sensor data output in real-time and timescales. On service users, service nodes, and service data, a middleware called AuthenticateAccess is implemented using the key for each service to prevent unrecognized requests from outside the service gateway.

TABLE III. HARDWARE AND SOFTWARE SPECIFICATIONS USED IN SYSTEM DEVELOPMENT

Hardware Specifications	
Processor	Intel Core i3 8th Gen
RAM	12 GB
HDD	500 GB
Software Specifications	
Operating System (OS)	Windows 10 Home Single Language 64-bit
Web Browser	Google Chrome Version 91.0.4472.77 (Official Build) (64-bit)
Modeling	Figma, Visual Paradigm Online, Microsoft Visio
Code Editor	Visual Studio Code Versi 1.56.2
Version Control System	Git
DBMS	MySQL Version 5.7.24
Web Server	Apache
Universal Development Environment	Laragon Versi 4.0.12
Testing Tools REST API	Apache JMeter, Postman

The results of the IoT monitoring application interface implementation can be seen in the Figure 9 admin dashboard, and Figure 10 is a sensor display.

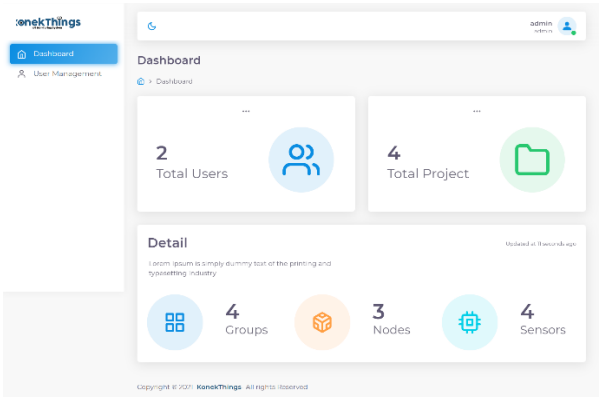

Fig. 9 Admin Dashboard Page Interface

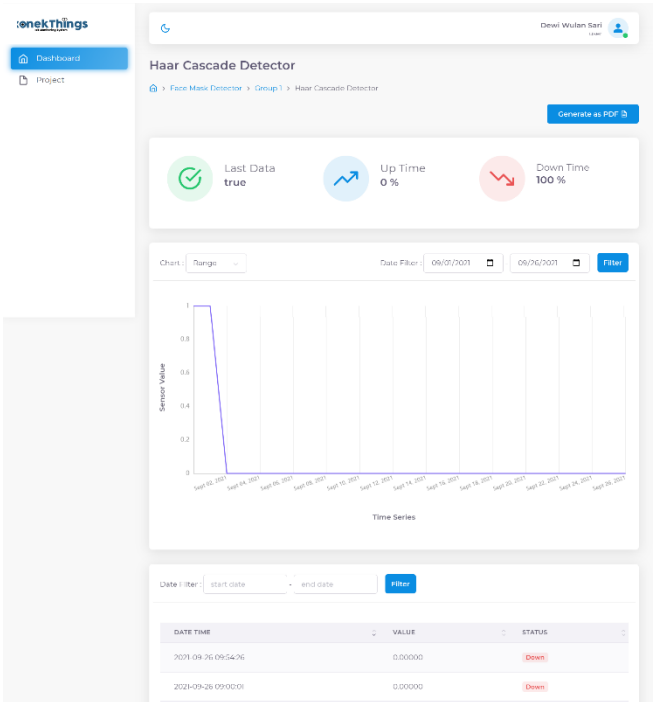

Fig. 10 Sensor Details Page Interface

D. *Testing*

1. Unit Test

Unit testing focuses on testing the smallest unit on the system in the form of features or functions. Tests are carried out to verify that each unit runs as expected under various circumstances. Unit testing is carried out using an automatic test method using a test script by utilizing the PHPUnit library version 9.5.2, integrated into the Lumen project for each service. The results of the unit testing of the IoT monitoring application can be seen in table 4.

TABLE IV. IDENTIFICATION OF IOT MONITORING APPLICATION CONTEXT

Service Name	Test Case	Test Statement	Execution Time	Memory	Status
Service Gateway	Two tests	16 assertions	1.9 s	26 MB	Passed
Service User	20 tests	88 assertions	7.3 s	28 MB	Passed
Service Node	48 tests	139 assertions	59.3 s	30 MB	Passed
Service Data	Nine tests	12 assertions	3.7 s	28 MB	Passed

2. Integration Testing

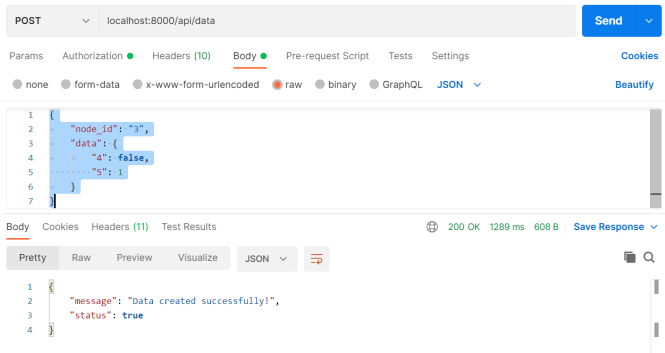

Fig. 11 Endpoint Integration Testing receiving data from nodes with microservice

Figure 11 is an integration test of 43 endpoints receiving sensor data from nodes with microservices. The process of sending data from a node or device using the JSON format.

3. Performance Testing

The test was carried out using Apache Jmeter, which was carried out three times and limited each run to 1 minute so that there were consistent parameters to compare. Tests were carried out using a 64-bit Ubuntu 20.10 LTS Virtual Machine running on Virtual Box with a basic specification of 2 GB RAM as a server.

Table 5 shows the average response time, latency, and throughput of the system in receiving sensor data from several devices or nodes. Testing is done with each node having three sensors that access the system simultaneously to obtain consistent test data. This test is to identify the number of nodes limits that can be accessed simultaneously.

TABLE V. TEST RESULT OF TOTAL NODES LIMITATIONS ACCESSING THE SYSTEM CONCURRENTLY

Total Nodes	Avg. Response Time (ms)	Avg. Latency (ms)	Avg. Throughput (request/minute)	Success Rate (%)
1	62.3	62.3	954.0	100
2	120.7	120.7	989.7	100
3	217.0	217.0	903.0	100
4	252.7	252.7	950.0	100
5	304.3	304.3	985.0	100
10	787.7	787.7	773.0	100
25	1844.7	1844.7	823.3	100
50	3574.7	3574.3	857.0	100
75	5709.7	5706.7	814.0	99.8
100	13101.7	13097.7	484.7	85.6

TABLE VI. TEST RESULTS LIMITATION OF TOTAL SENSORS IN A NODE ACCESSING THE SYSTEM SIMULTANEOUSLY

Total Sensor	Avg. Response Time (ms)	Avg. Latency (ms)	Avg. Throughput (request/minute)
1	52.7	52.7	1123.7
2	57.3	57.3	1037.0
3	61.7	61.7	961.3
4	67.0	67.0	888.3
5	73.3	73.3	810.7
10	99.3	98.7	601.3

Details of the average response time, latency, and throughput of receiving sensor data by the system sent from several sensors that access the system simultaneously to

obtain a limit on the number of sensors in a node can be seen in table 6. as well as representations in figures 12 and 13.

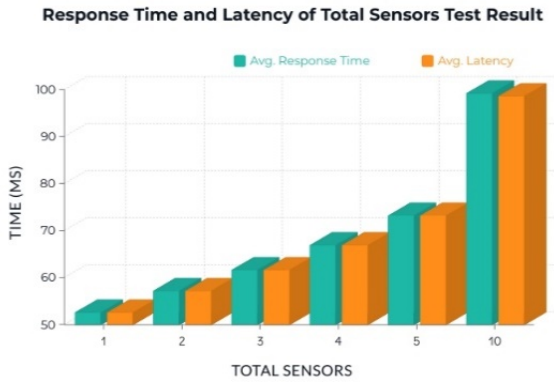

Fig. 12 Graph of response time and latency of the total sensor test results that access simultaneously

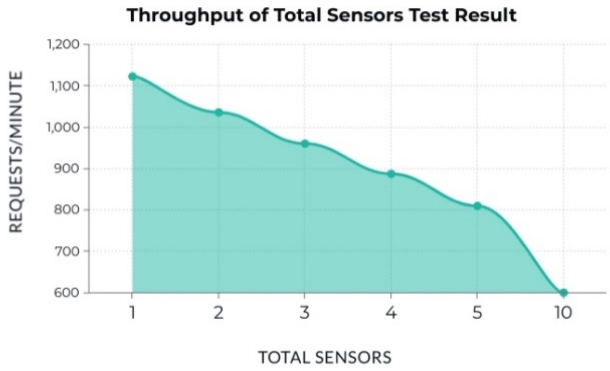

Fig. 13 The throughput graph of the total sensor test results that access simultaneously

Details of the average response time, latency, and throughput of accessing sensor data on the system by users who access simultaneously can be seen in table 7.

TABLE VII. TEST RESULTS OF TOTAL USER LIMITATIONS ACCESSING THE SYSTEM CONTINUOUSLY

Total User	Avg. Response Time (s)	Avg. Latency (s)	Avg. Throughput (request/minute)
1	1.5	1.5	43.0
2	2.2	2.1	59.7
3	2.4	2.4	76.0
4	3.2	3.2	76.0
5	4.1	4.1	74.3
10	8.5	8.4	75.7
25	20.0	19.8	80.7

Based on the results of performance testing, the IoT monitoring application with a microservice architecture based on the REST API built can handle the number of nodes (with three sensors per node) accessing simultaneously at a time, reaching a limit of 75 nodes. The total sensors in one node can have up to 10 sensors per node with a response time of less than 100ms. In testing the user's request to get sensor data simultaneously, the results show that the response time of each request is directly proportional to the number of sensors, so the number of sensors at each node will affect the throughput or the number of requests handled in one minute. As for the number of users who access sensor data simultaneously in the application, there is no definite limit because the response depends on the amount of data called.

E. Deployment

After the testing phase is over, the system is feasible to be launched and implemented. However, the software development cycle will continue. Maintenance, repair, and updates must be carried out continuously for the survival of the system.

F. Evaluation

The application of the Domain-Driven Design (DDD) concept approach positively impacts the resulting design. DDD is more than what Object-Oriented Analysis and Design (OOAD) [13] tries to solve. There are Bounded Contexts on the DDD side, which are delimited the applicability of a specific model. The software made closely related to the business domain, rather than being arbitrary decisions made by the team. Communication with domain experts in the DDD phase using Ubiquitous Language is beneficial to describe the system's interactions in terms of the business problem that is being attempted to solve.

Microservice architecture with a REST API-based approach or, in other words, an API-driven architecture that is applied to IoT monitoring applications works well. The application of microservice architecture in IoT monitoring applications makes all services built to be more independent during the development phase. System development can be carried out without the need to overhaul the entire system. It will not interfere with the performance of other services, thus increasing the scalability aspect of the system. In addition, the development of each service does not depend on the programming language. At the deployment stage, the service can be deployed without waiting for the entire service to be completed, thereby increasing the effectiveness of system deployment.

V. CONCLUSION

Based on the research results, applying the concept of domain-driven design (DDD) makes the definition of boundaries or scope of services more precise and more accurate than OOAD because the decomposed services are organized based on business or domain concepts. So that it can produce adaptive software to changes during the development phase. In addition, it produces effective code with object-oriented techniques, resulting in easier to maintain code and avoiding dead code. That has been tested at the unit testing, integration, and performance stages. Furthermore, the implementation of DDD overcomes the difficulty of defining how small the size of the service is to be split on a microservice architecture if without DDD. Based on the performance testing results, the system can handle the number of nodes (with three sensors per node) that access simultaneously at one time, reaching 75 nodes. The total sensors in one node can have up to 10 sensors per node with a response time of less than 100ms.

The aspects that need to be improved include the dependence of service data on service nodes. For future research, we can try to minimize the dependencies between services and achieve greater scalability by developing event-based microservice architectures using message broker technology and the MQTT protocol. In addition, the

implementation of domain-driven design is not only applied during the design phase, but the domain-driven design layer can also be implemented in code in every service in the development phase.

REFERENCES

[1] K. L. Lueth, "State of the IoT 2020: 12 billion IoT connections," 2020. [Online]. Available: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/. [Accessed: 24-Feb-2021].

[2] A. A. Karia, L. V. Budhwani, and V. S. Badgujar, "IoT-Key Towards Automation," *2018 Int. Conf. Smart City Emerg. Technol. ICSCET 2018*, pp. 1–5, 2018.

[3] L. Sun, Y. Li, and R. A. Memon, "An open IoT framework based on microservices architecture," *China Commun.*, vol. 14, no. 2, pp. 154–162, 2017.

[4] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *J. Syst. Softw.*, vol. 150, pp. 77–97, 2019.

[5] C. Setya Budi and A. M. Bachtiar, "Implementasi Arsitektur Microservices pada Backend Comrades," *Progr. Stud. Tek. Inform. Univ. Komput. Indones.*, pp. 1–6, 2018.

[6] A. Messina, R. Rizzo, P. Storniolo, and A. Urso, "A Simplified Database Pattern for the Microservice Architecture," *Eighth Int. Conf. Adv. Databases, Knowledge, Data Appl.*, no. June, pp. 35–40, 2016.

[7] A. Hermawan, "Peningkatan Ketersediaan Aplikasi Web Menggunakan Arsitektur Layanan Mikro Berdasarkan Identifikasi Log Akses," 2017.

[8] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003.

[9] S. Dharma Handayani and U. Uminingsih, "Pengorganisasian Kerja Sistem Parkir Menggunakan Arsitektur Microservice," *J. Teknol.*, vol. 13, no. 1, pp. 27–35, 2020.

[10] S. Trilles, A. González-Pérez, and J. Huerta, "An IoT platform based on microservices and serverless paradigms for smart farming purposes," *Sensors (Switzerland)*, vol. 20, no. 8, 2020.

[11] N. Nikolakis *et al.*, "A microservice architecture for predictive analytics in manufacturing," *Procedia Manuf.*, vol. 51, no. 2019, pp. 1091–1097, 2020.

[12] A. Macías, E. Navarro, and P. González, "A Microservice-Based Framework for Developing Internet of Things and People Applications," *Proceedings*, vol. 31, no. 1, p. 85, 2019.

[13] F. Zaki and S. Adhy, "PENGEMBANGAN INTERNET OF THINGS PLATFORM BERBASIS WEB MENGGUNAKAN METODE OBJECT-ORIENTED ANALYSIS AND DESIGN (OOAD)," 2018.

[14] M. Kalske, N. Makitalo, and T. Mikkonen, "Challenges When Moving from Monolith to Microservice Architecture," *Int. Conf. Web Eng.*, no. February, pp. 32–47, 2018.

[15] V. Sharma and R. Tiwari, "A review paper on 'IOT' & It "s Smart Applications," *Int. J. Sci. Eng. Technol. Res.*, vol. 5, no. 2, pp. 472–476, 2016.

[16] B. Artono and R. G. Putra, "Penerapan Internet Of Things (IoT) Untuk Kontrol Lampu Menggunakan Arduino Berbasis Web," *J. Teknol. Inf. dan Terap.*, vol. 5, no. 1, pp. 9–16, 2019.

[17] F. M. S. Nursuwars and A. Rahmatulloh, "RFID for nurse activity monitoring in the hospital's nurse call system with Internet of Thing (IoT) concept," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 550, no. 1, 2019.

[18] A. Rahmatulloh, F. M. S. Nursuwars, I. Darmawan, and G. Febrizki, "Applied Internet of Things (IoT): The Prototype Bus Passenger Monitoring System Using PIR Sensor," *2020 8th Int. Conf. Inf. Commun. Technol. ICoICT 2020*, 2020.

[19] D. Pal, "Review on Impact of Agile Technologies in Software Development," *SSRN Electron. J.*, 2021.

[20] D. Kurniawan, R. Fadli Isnanto, Syamsuryadi, and Fathoni, "Implementasi Arsitektur Microservice: Studi Kasus Pada Pengembangan Surat Keterangan Pendamping Ijazah di Lingkungan Fakultas Unsri," *Pros. Annu. Res. Semin. 2019 Comput. Sci. ICT*, vol. 5, no. 1, pp. 978–979, 2019.

[21] V. Vernon, *Implementing Domain-Driven Design*. 2013.

[22] D. S. Purnia, A. Rifai, and S. Rahmatullah, "Penerapan Metode Waterfall dalam Perancangan Sistem Informasi Aplikasi Bantuan Sosial Berbasis Android," *Semin. Nas. Sains dan Teknol. 2019*, pp. 1–7, 2019.

[23] F. Rademacher, J. Sorgalla, and S. Sachweh, "Challenges of domain-driven microservice design: A model-driven perspective," *IEEE Softw.*, vol. 35, no. 3, pp. 36–43, 2018.

[24] B. Hippchen, M. Schneider, P. Giessler, and S. Abeck, "Systematic Application of Domain-Driven Design for a Business-Driven Microservice Architecture," vol. 12, no. 3, pp. 343–355, 2019.

[25] A. Diepenbrock, F. Rademacher, and S. Sachweh, "An Ontology-based Approach for Domain-driven Design of Microservice Architectures," *Lect. Notes Informatics (LNI), Proc. - Ser. Gesellschaft fur Inform.*, vol. 275, pp. 1777–1791, 2017.

[26] K. Katuwal, "Microservices : A Flexible Architecture for the Digital Age Version 1 . 1," *Am. J. Comput. Sci. Eng.*, vol. 3, no. 4, pp. 23–28, 2016.

[27] N. Dragoni *et al.*, "Microservices : Yesterday , Today , and Tomorrow," *Present Ulterior Softw. Eng. Springer, Berlin, Ger. 2017*, pp. 195–216, 2017.

[28] R. Rizal and A. Rahmatulloh, "RESTful Web Service untuk Integrasi Sistem Akademik dan Perpustakaan Universitas Perjuangan," *J. Ilm. Inform.*, vol. 7 No 1, 2019.

[29] A. Pamuji, "Rancang Bangun Web Service Menggunakan Representational State Transfer Untuk Pengolahan Data Barang - UTY Open Access," 2020.