Microservice Based Architecture: The Development of Rapid Prototyping Supportive Tools for Project Based Learning

Irwan Alnarus Kautsar
Informatics Department
Universitas Muhammadiyah Sidoarjo
Sidoarjo, Indonesia
irwan@umsida.ac.id

M. Ruslianor Maika
Sharia Banking Department
Universitas Muhammadiyah Sidoarjo
Sidoarjo, Indonesia
mr.maika@umsida.ac.id

Agoes Nur Budiman
Research Partner
PT. Ijabqobul Muamalah Indonesia
Sidoarjo, Indonesia
mitra@ijabqabul.id

Arik Bagus Setyawan
Informatics Department
Universitas Muhammadiyah Sidoarjo
Sidoarjo, Indonesia
setyawanarik@gmail.com

Jagad Yudha Awali
Informatics Departments
Universitas Muhammadiyah Sidoarjo
Sidoarjo, Indonesia
me@jagad.dev

***Abstract*—This paper presents the migration of rapid prototyping supportive tools systems from monolith into microservice architecture that will be used as the implementation of Project Based Learning. As in early development, the developed supportive tool was the monolith architecture and web based platform. As the growth of the students as users and addition of the rapid prototyping framework modules that will be used, the monolith architectures are urged to decompose its' services into a more modular way of web services. As a result, the newest version will take advantage of a number of benefits offered by microservice-based architecture, including modularity, scalability and maintainability. The future features that are needed as the implementation of the learning based systems will be more easy to integrate as the beneficial of the microservices-based architectures.**

Keywords—microservice, rapid prototyping, supportive tool, project based learning

I. INTRODUCTION

Numerous web services that are constantly and continually updated have been created and are being operated as a result of the development of cloud computing and web technology. A multi-layered architecture with a monolithic design is typically used to build several web-based applications or web services [1], [2]. The internal implementation of these layers is becoming more challenging, and changing the system can necessitate extensive rebuilds and redeployments [3], [4]. For a system that needs to change often and continuously (like agile software development), there must be a lot of changes that make development and operation difficult [5]. As a result, it is required to localize the area of influence on the modified software module. In light of this, the utility of microservice architecture is being assessed. This architecture builds a system by combining software components from several microservices.

The constructed web application is based on a conventional monolithic design consisting of three fundamental tiers: the persistent layer, middleware, and front-end code. Monolithic design has the disadvantages of not being scalable and reducing modularity. The app's dashboard is a website that serves all static files, front-end HTML, CSS, and JavaScript, acts as the REST API, and serves as the data persistence tier, authentication, and notification. Any changes to any of the three tiers will necessitate a significant amount of team effort to launch a new release—a significant amount of effort for a little change.

Microservices can be viewed as a technique for designing software applications that, by inheriting the principles and concepts of the Service-Oriented Architecture (SOA) style, allows a service-based application to be structured as a collection of very small and connected software services. Microservices architecture can be viewed as a new paradigm for building applications by composing small services, each with its own procedures and lightweight techniques for communication. Microservices are called "micro" not because of the sum of the lines of code, but because of their specific roles for the sake of platform reliability.

Several studies [6], [7] have shown similar efforts to transition monoliths to microservices architecture. Some research has proposed repackaging the program, refactoring the code, and then refactoring the data [8], [9]. To create scalable microservices, the reference [10] advocated multiple stages, beginning with employing unsupervised machine learning techniques to examine monolithic application log files in order to discover candidates for microservices migrations. Next, the reference [10] will determine which portions of the application receive more requests (higher loads) and construct new microservices for these features so that they may be automatically scaled and routed by a load balancer. In a case study of transforming a monolith into a cloud-native application, [11]–[13] recommended multiple techniques depending on the type of existing monolithic applications. The reference [14], [15] propose a recovery strategy to support model-driven engineering for the creation of microservices, whereas the references [16], [17] propose a domain-driven design to complete the migration to a microservices architecture.

This paper is organized as follows: The prior work on rapid prototyping supportive tools as the lecturer's companion while implementing project-based learning (PBL) in monolithic architecture is presented in Section II. In Section III, the microservice architecture as the proposed method is

explored. Section IV discusses the implementations. Section V discusses the results and outlines several future research directions.

II. Problem Analysis

A. Project-based Learning and Prototyping Framework

Project-based Learning (PBL) is one type of learning model that challenges students to solve real-world issues [18]–[21]. Key components of the PBL method include presenting students with the need for some systems or an incomplete existing digital service, then encouraging them to complete it, promoting self-discipline and self-regulation by allowing students to define their working hours, timeline, and outcome, and encouraging teamwork and interdisciplinary collaboration.

Learning models such as project-based learning are output-based learning models, also known as outcome-based education (OBE). When adapting OBE, informatics students must create an outcome from ideas into usable applications as learners [18,19]. For this reason, students need to be introduced to several prototyping tools. The problem starts rising: "What are the suitable prototyping frameworks, not only for freshman year but also for 3rd and 4th year bachelor students?" These questions are urged to be answered since the Indonesian Higher Education Ministry implemented a renowned curriculum called Merdeka Belajar Kampus Merdeka (MBKM) [22]. The meaning of "Merdeka" can be translated as "freedom." As an Indonesian citizen, the word "Merdeka" is meaningful since the word played an important role in Indonesia's independence. The word "belajar" means "to learn" or "learning." This is why the main focus in those MBKM curricula is on students being pushed to have collaboration not only with their peers in the same department but also with their seniors and other students from outside departments (adapting interdisciplinary learning). So, in our perspective, introducing a framework that can be used not only by informatics students but also other students from outside the informatics department seems urgently needed.

B. Supportive Tools and Prototyping Framework

A supportive tool is a platform that is used to support educators in implementing a learning model [23], [24]. Previously, various prototyping frameworks had been chosen in order to prototype adaptation in project-based learning models. Also, we develop supportive tool platforms that adapt to the chosen framework [25], [26]. The platform created not only allows students to publish their final project. But it also helps lecturers monitor, control, and evaluate the learning process [20] .

However, the supportive platform that started with small modules eventually became a bigger project [27]. Along with the adaptation of the new selected prototyping frameworks. Other problems arise since the chosen prototyping framework is divided into different levels. For example, the Cause-Effect-Solution framework and Funtional/Non Functional (F/NF) adoption are for first- and second-year students, respectively. And the business model canvas and platform design canvas are for the 3rd and 4th years, respectively. Because they will be using the same platform and the same supportive tool, which is built on a monolithic architecture, the load on the supportive tool and server response time will quickly become an issue. Figure 1 depicts the monolithic architecture of the developed supportive tools.

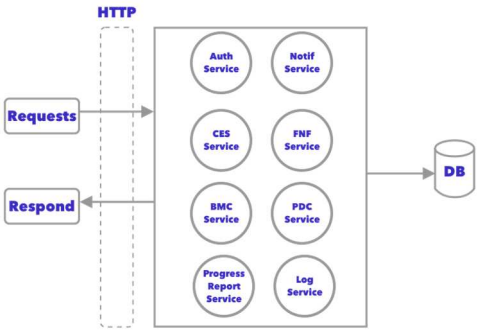

Fig. 1. Supportive tool with monolith architecture

III. Proposed Method

A. Microservices Based Architecture (MBSA)

Microservice is an imprecise phrase that is supposed to refer to an architectural approach that separates a system into small, lightweight services [28]. This service is purposefully designed to execute a highly interdependent business function; it is an extension of the classic service-oriented architecture [29] and a well-mapped implementation in [28], [30].

According to [31], several microservices are combined to create a single application. These microservices run in their own processes and frequently connect with one another using a lightweight communication protocol, such as the REST API (Representational State Transfer Application Programming Interface) [32]. In addition, these microservices are based on business capabilities and can be independently delivered by completely automated procedures [33]. The degree of centralization for these services is limited, and each service is able to utilize distinct programming languages and data storage technologies.

In practice, the idea of the microservice is to examine the offered functionality [34]. As a result, it is clear that microservices go beyond the separation of services in a monolith [29], [31], [35]. As seen in Figure 1, all services are still tied to a single database. Each service with its own database should migrate to a microservice-based architecture. It provided the REST API as the interface in addition to separating the database to achieve independence. Figure 2 depicts the current development of rapid prototyping supportive tools' microservice-based architectures.

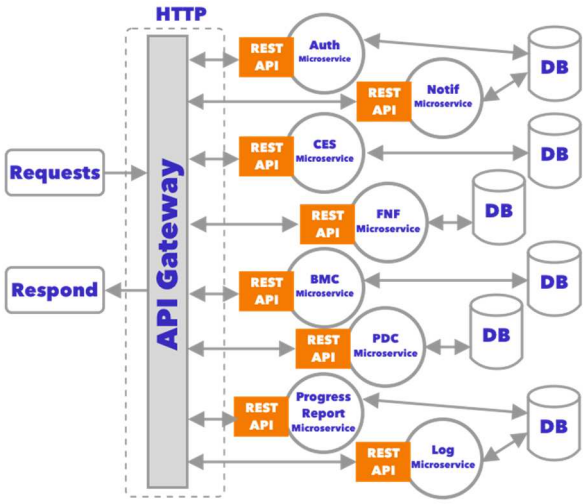

Fig. 2. Supportive tool microservice architecture

Figure 1 depicts a monolithic architecture in which system modules such as authentication, notification, progress reporting, and log services are centralized at a single database engine. The same database engine stores the prototyping frameworks: cause-effect-solution (CES), function-non-functional (FNF), business model canvas (BMC), and platform design canvas (PDC). Figure 2 shows the services are decomposed into separate REST API services along with a decentralized database engine.

B. Interprocess Communications

The key to using supportive tools is to enable students to collaborate on the prototyping process as part of project-based learning. This means the prototyping microservices (CES, FNF, BMC, and PDC) and system modules (authentication, notification, progress report, and log services) must have the ability to have interservice communications. In monolithic business logic, microservice interprocess communication occurs. It needs to be deployed at intelligent endpoints, also known as business logic layers (BLL). Direct point-to-point communication is the most straightforward approach to invoking the service. Each microservice represents a REST API, and a microservice or external client can import other microservices using its REST API, as depicted in Figure 3.

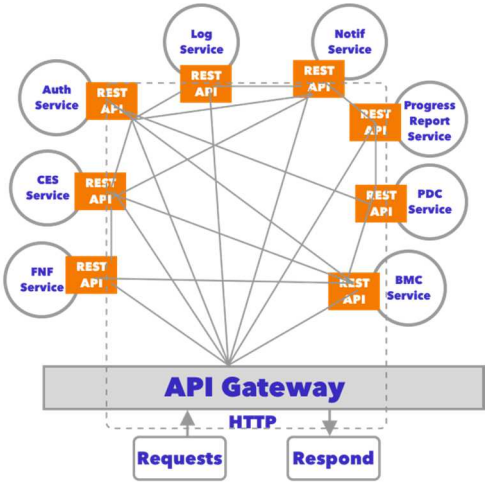

Fig. 3. Interprocess Communication between microservices

Representational State Transfer (REST), which offers a straightforward communications style implemented with HTTP request-response and is based on resource API style, is the overwhelming choice. Synchronous messaging is a REST API. While they are required to simulate asynchronous messaging protocols like ZeroMQ, RabbitMQ, or Matrix for various microservice scenarios. We implemented microservice using Flask and Nameko framework for the microservices.

As the number of microservices increases, point-to-point communication will become more difficult. At each and every microservices level, the non-functional requirement must be implemented. This may result in redundant common functionality and a complete lack of control over the communication between microservices and clients. This form of direct communication is considered an antipattern for large-scale microservice implementation [29]. In this case, an API-Gateway design is used. The concept is to employ a lightweight message gateway as the primary entry point for all clients, and to integrate non-functional requirements such as security, monitoring, and control at the gateway level. The alternative style may be a message broker style for asynchronous messaging technologies like RabbitMQ and ZeroMQ.

Due to the high density of microservices in microservice architecture and the possibility of continuous request changes as part of agile development, the service registry concept will provide a solution. The locations of the microservice instances will be stored in the service registry. It indicates that the service registry registers each microservice instance during startup and deregisters it upon shutdown. The introduction of a service discovery is used to locate the accessible microservices. Next, load balancer will control and serve the incoming request as part of service discovery mechanisms. Figure 4 illustrates the role of service registry.

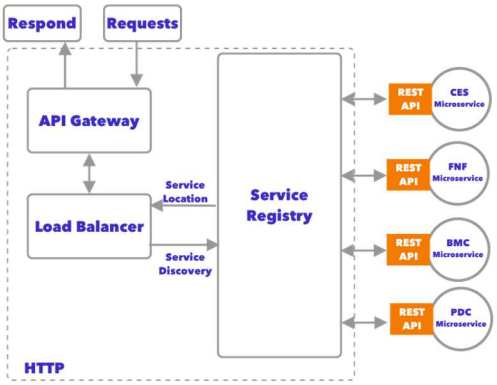

Fig. 4. Service registry

Because microservices are self-contained services that are directly connected to the database, a secure communication interface is required. Furthermore, OAuth2 and OpenID are implemented in the microservices architecture as API security standards. OAuth2 will authenticate the client with the authorization server and return an access token. An access token is an obscure token with no user or client information. It contains only a reference to the user's information, which can only be retrieved by the authorization server, and it will be saved as a "by-reference token." In addition to the access token, the authorization server uses an OpenID token that contains information about the user in the form of a JSON web token (JWT) that is signed by the authorization server. This will ensure that the authorization server and mobile client are trustworthy. Figure 5 shows the implementation of OAuth2 and OpenID as part of the authentication process.

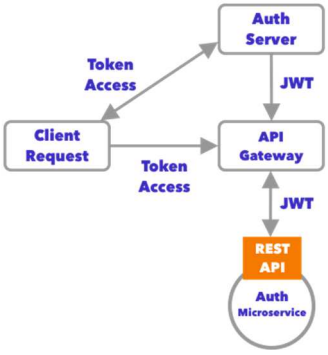

Fig. 5. Rapid prototyping suppotive tool architeture design

To guarantee the management of the system's health, we adopted self-managing atomic services [36]. The simplified instantiation and de-instantiation sequence diagram is depicted in Figure 7. The orchestrator initiates the deployment of services first. Secondly, the load balancer (LB) registers the request as a new identification of an endpoint. Also, the LB monitors the registration Application Service (AS) and other pertinent events with the information stored at Oh-My-PickleDB (OMPDB) [37]. OMPDB is an open-source key-value store using Python's JSON module. OMPDB updates configuration settings (reconfiguration parts for Application Service and Cache). The orchestrator will set the service to "active" as soon as all initial components have been deployed. The produced component's monitoring data is continuously saved at the OMPDB. Periodically, each component checks the status of the service. If the service is running and the OMPDB cluster leader node is discovered, auto-scale and health management will begin. As an alternative, the automatic scaling and the health management components can be launched based on the load of the requested microservices. Figure 7 shows the sequence diagram of the service instantiation and de-instantiation mechanisms.

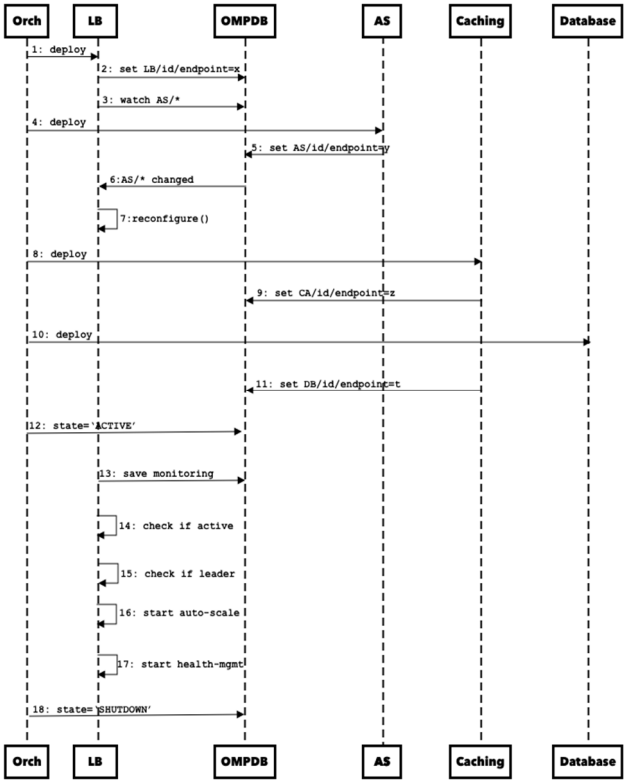

Fig. 6. Self-managing sequence diagram microservice-based architecture

IV. Result and Discussion

A. Load Test

In this section, we compare the load tests of the monolithic and microservice-based architectures. The load of each architecture has been evaluated using Locust (http://locust.io). The load test was applied to four prototyping frameworks (CES, FNF, BMC, and PDC) in both architectures. The results are shown at Table I, Table II, and Figure 7.

TABLE I. Monolith Based Load Test Results

No	MS Code	Req. Count	Min Resp. Time (ms)	Max Resp. Time (ms)	Avg. Resp. Time (ms)	Avg. Size (Byte)
1	CES	4826	8.72	316.15	62.81	96.13
2	FNF	5921	8.24	288.72	49.29	980
3	BMC	4829	6.49	340.23	32.12	63.46
4	PDC	5102	6.45	182.47	26.92	86.17

TABLE II. Microservice Based Load Test Results

No	MS Code	Req. Count	Min Resp. Time (ms)	Max Resp. Time (ms)	Avg. Resp. Time (ms)	Avg. Size (Byte)
1	CES	4983	4.87	207.53	21.12	82.27
2	FNF	5125	4.93	256.06	30.26	1350
3	BMC	5069	4.77	395.41	22.64	81.79
4	PDC	5093	5.05	192.68	21.87	84.07

From Tables 1 and 2, the FNF prototyping framework chose the comparison of both architectures. That is because the FNF in both architectures has the highest average content size (ACS), which is 1350 bytes for microservices and 980 bytes for monoliths at each request. Figure 7 shows that, when FNF is compared to monolith-based architecture, the implementation of microservice-based architecture has a faster response time. The microservices-based architecture has the lowest average response time (ART), with a value of 30.26 ms. Compared with ART on monolithic bases with 49.29 ms. Furthermore, when looking at overall response time (ms) results from all prototyping framework load tests with microservice-based architecture, the ART value has decreased over time. This means the system health management implementation was successfully implemented.

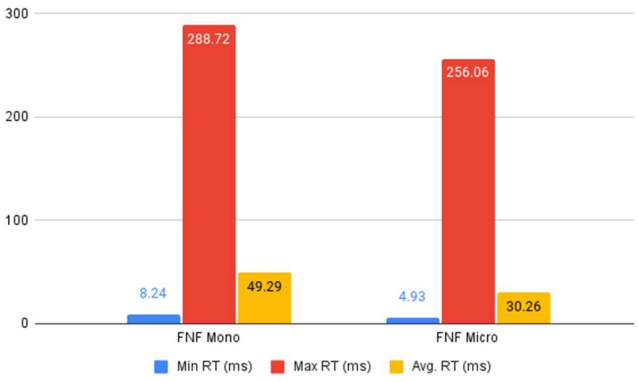

Fig. 7. The load test results of FNF ptrototyping framework

B. User Acceptance Test

We conduct experiments with the following scenarios to determine the level of acceptance of students as users when using monolithic and microservice-based supportive tools:

1. Students from the first, second, third, and fourth years formed a group.Each group has been given two assignments as a case study. They asked for the first assignment to analyze problems in the crowdfunding sector.The second assignment is to investigate health-care issues.
2. We established guidelines stating that first-year students should use the CES Framework, second-year students should use the F/NF Framework, third-year students should use the BMC Framework, and fourth-year students should use the PDC Framework.
3. Each group was given one week to complete the two assignments.Following that, we instruct students to complete their first assignment on Server A, where we prepared the supporting tools using a monolithic architecture. And finish the second assignment on server B, which already use microservice architecture to deploy supportive tools.

We collected questionnaires from 146 participants to determine whether there is any performance improvement or experience with the proposed method (microservice-based) compared to the previously developed monolith-based architecture. The questionnaire had five Likert scales with the predicates "Strongly Agree," "Agree," "Neutral," "Disagree," and "Very Disagree," with points 5, 4, 3, 2, and 1, respectively. Also, using Eq. (1) for understanding the respondents' expressions with the questionnaire items.

$$P = \frac{N \times R}{I} \times 100\% \quad (1)$$

Where:
P = Each question percentage value
N = The value of each instruments response
R = The frequency of answered value
I = The number of participants multiplied by the highest value of the answer (146 × 5 = 730)

Table 3 shows the questionnaire item.

TABLE III. QUESTIONNAIRE ITEMS

No	Descriptions
1	Prototyping frameworks help analyze problems.
2	Supportive tools help implement the prototyping framework.
3	Supportive tools help collaboration while prototyping.
4	Both assignments have similar difficulties.
5	Both supportive tools have the same response.
6	Server A appeared to be faster than server B.
7	Server B appeared to be faster than server A.
8	The given instructions are easy to follow.

Figure 8 shows the Likert percentage from Tabel 3 and Eq. (1)

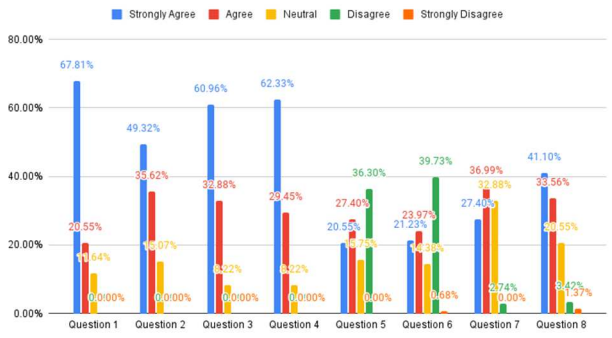

Fig. 8. Likert percentage from questionnaires

From Figure 8, 67% of students express a strong understanding of the use of prototyping frameworks for problem analysis. Furthermore, more than 70% of students (a combination of strongly agree and agree) understand the benefits of supportive tools and use the prototyping framework. More than 83% of respondents express agreement that supportive tools facilitate collaboration while prototyping. Next, students were asked about their experiences and if there were any differences between using server A or B to complete the assignment. It happened that 36% of students felt that there was a difference in response from both servers. 63% express the opinion that Server B appeared to be faster than Server A.

V. CONCLUSION AND FUTURE WORK

The monolithic tools were converted to a microservice architecture. The migrated design includes the OAuth2 and OpenID API security standards. This reduces the security threats since the databases are decentralized to their services. In contrast to monolithic architecture, which stores credential and transactional data in a single database, microservice architecture stores transactional and credential data in separate databases. Furthermore, the platform performance that is being developed with microservice architecture offers a better experience for lecturers and students while using supportive tools for implementing project-based learning. This is because it already has separate services for each student level.However, if the platform has already been decomposed into one prototyping framework and one service, the implementation of the new framework will not disrupt service. The use of microservice-based architecture offers flexibility when adapting new prototyping frameworks. Because when deploying the new service, there is no need to terminate the whole prototyping service. It only needs to reactivate the service registry. Because the platform structure already has an independent API service, it will make development easier when the mobile version of the supporting tools is developed in the near future.

ACKNOWLEDGMENT

The authors thank Kementerian Pendidikan, Kebudayaan, Riset, and Teknologi for funding this work under contract number 073/E5/P6.02.00.PT/2022, 019/SP2H/PT-L/LL7/2022, and 621.06/II.3.AU/14.00/C/PER/VI/2022. The authors also thank Universitas Muhammadiyah Sidoarjo.

REFERENCES

[1] "Effective Optimization of Web Sites for Mobile Access: The Transition from eCommerce to mCommerce: Journal of Interactive Advertising: Vol 9, No 1."

https://www.tandfonline.com/doi/full/10.1080/15252019.2008.10722149 (accessed Jun. 20, 2021).
[2] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting Microservices: Practical Opportunities and Challenges," *J. Comput. Inf. Syst.*, vol. 60, no. 5, pp. 428–436, Sep. 2020, doi: 10.1080/08874417.2018.1520056.
[3] W. K. G. Assunção, J. Krüger, and W. D. F. Mendonça, "Variability management meets microservices: six challenges of re-engineering microservice-based webshops," in *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*, New York, NY, USA, Oct. 2020, pp. 1–6. doi: 10.1145/3382025.3414942.
[4] T. Killalea, "The Hidden Dividends of Microservices: Microservices aren't for every company, and the journey isn't easy.," *Queue*, vol. 14, no. 3, pp. 25–34, May 2016, doi: 10.1145/2956641.2956643.
[5] F. Li and L. Gelbke, "Microservice architecture in industrial software delivery on edge devices," in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, New York, NY, USA, May 2018, pp. 1–4. doi: 10.1145/3234152.3234196.
[6] O. Al-Debagy and P. Martinek, "Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach," in *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, Jun. 2020, pp. 289–294. doi: 10.1109/SoSE50414.2020.9130466.
[7] S. S. de Toledo, A. Martini, A. Przybyszewska, and D. I. K. Sjøberg, "Architectural technical debt in microservices: a case study in a large company," in *Proceedings of the Second International Conference on Technical Debt*, Montreal, Quebec, Canada, May 2019, pp. 78–87. doi: 10.1109/TechDebt.2019.00026.
[8] M. Tusjunt and W. Vatanawood, "Refactoring Orchestrated Web Services into Microservices Using Decomposition Pattern," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, Dec. 2018, pp. 609–613. doi: 10.1109/CompComm.2018.8781036.
[9] N. Gonçalves, D. Faustino, A. R. Silva, and M. Portela, "Monolith Modularization Towards Microservices: Refactoring and Performance Trade-offs," in *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2021, pp. 1–8. doi: 10.1109/ICSA-C52384.2021.00015.
[10] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices," *J. Syst. Softw.*, vol. 151, pp. 243–257, May 2019, doi: 10.1016/j.jss.2019.02.031.
[11] S. G. Haugeland, P. H. Nguyen, H. Song, and F. Chauvel, "Migrating Monoliths to Microservices-based Customizable Multi-tenant Cloud-native Apps," in *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Sep. 2021, pp. 170–177. doi: 10.1109/SEAA53835.2021.00030.
[12] M. Mishra, S. Kunde, and M. Nambiar, "Cracking the monolith: challenges in data transitioning to cloud native architectures," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, New York, NY, USA, Sep. 2018, pp. 1–4. doi: 10.1145/3241403.3241440.
[13] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May 2016, doi: 10.1109/MS.2016.64.
[14] L. Wu, J. Tordsson, A. Acker, and O. Kao, "MicroRAS: Automatic Recovery in the Absence of Historical Failure Data for Microservice Systems," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, Dec. 2020, pp. 227–236. doi: 10.1109/UCC48980.2020.00041.
[15] A. Power and G. Kotonya, "A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems," in *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, Jun. 2018, pp. 588–599. doi: 10.1109/WoWMoM.2018.8449789.
[16] S. Kapferer and O. Zimmermann, "Domain-Driven Service Design," in *Service-Oriented Computing*, Cham, 2020, pp. 189–208. doi: 10.1007/978-3-030-64846-6_11.
[17] 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, May 2018, doi: 10.1109/MS.2018.2141028.
[18] M. Barak and S. Yuan, "A cultural perspective to project-based learning and the cultivation of innovative thinking," *Think. Ski. Creat.*, vol. 39, p. 100766, Mar. 2021, doi: 10.1016/j.tsc.2020.100766.
[19] M. Marques, S. F. Ochoa, M. C. Bastarrica, and F. J. Gutierrez, "Enhancing the Student Learning Experience in Software Engineering Project Courses," *IEEE Trans. Educ.*, vol. 61, no. 1, pp. 63–73, Feb. 2018, doi: 10.1109/TE.2017.2742989.
[20] I. A. Kautsar and R. Sarno, "A Supportive Tool for Project Based Learning and Laboratory Based Education," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 9, no. 2, pp. 630–639, 2019.
[21] M. Genc, "The project-based learning approach in environmental education," *Int. Res. Geogr. Environ. Educ.*, vol. 24, no. 2, pp. 105–117, Apr. 2015, doi: 10.1080/10382046.2014.993169.
[22] "The Impact of Covid-19 to Indonesian Education and Its Relation to the Philosophy of 'Merdeka Belajar' | Studies in Philosophy of Science and Education," Apr. 2020, Accessed: Sep. 12, 2021. [Online]. Available: https://scie-journal.com/index.php/SiPoSE/article/view/9
[23] I. A. Kautsar, Y. Musashi, S. Kubota, and K. Sugitani, "Synchronizing learning material on Moodle and lecture based supportive tool: The REST based approach," in *2015 International Conference on Information Communication Technology and Systems (ICTS)*, Sep. 2015, pp. 187–192. doi: 10.1109/ICTS.2015.7379896.
[24] I. A. Kautsar, S. Kubota, Y. Musashi, and K. Sugitani, "Lecturer Based Supportive Tool Development and Approaches for Learning Material Sharing under Bandwidth Limitation," *J. Inf. Process.*, vol. 24, no. 2, pp. 358–369, 2016, doi: 10.2197/ipsjjip.24.358.
[25] I. A. Kautsar and M. R. Maika, "The use of User-centered Design Canvas for Rapid Prototyping," *J. Phys. Conf. Ser.*, vol. 1764, no. 1, p. 012175, Feb. 2021, doi: 10.1088/1742-6596/1764/1/012175.
[26] I. A. Kautsar and M. R. Maika, "Platform Design Canvas Adaptation for Rapid Prototyping and Project-based Learning amid Covid-19 Pandemic," in *2022 IEEE World Engineering Education Conference (EDUNINE)*, Mar. 2022, pp. 1–6. doi: 10.1109/EDUNINE53672.2022.9782390.
[27] I. A. Kautsar and R. Sarno, "The use of Microframework for Portable and Distributed ePortfolio Development," in *2019 IEEE International Conference on Engineering, Technology and Education (TALE)*, Dec. 2019, pp. 1–6. doi: 10.1109/TALE48000.2019.9225965.
[28] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *J. Syst. Softw.*, vol. 150, pp. 77–97, Apr. 2019, doi: 10.1016/j.jss.2019.01.001.
[29] A. Kwan, H.-A. Jacobsen, A. Chan, and S. Samoojh, "Microservices in the modern software world," in *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*, USA, Oct. 2016, pp. 297–299.
[30] N. Alshuqayran, N. Ali, and R. Evans, "A Systematic Mapping Study in Microservice Architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Nov. 2016, pp. 44–51. doi: 10.1109/SOCA.2016.15.
[31] N. Dragoni *et al.*, "Microservices: Yesterday, Today, and Tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 195–216. doi: 10.1007/978-3-319-67425-4_12.
[32] R. T. Fielding *et al.*, "Reflections on the REST architectural style and 'principled design of the modern web architecture' (impact paper award)," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, New York, NY, USA, Aug. 2017, pp. 4–14. doi: 10.1145/3106237.3121282.
[33] E. Djogic, S. Ribic, and D. Donko, "Monolithic to microservices redesign of event driven integration platform," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2018, pp. 1411–1414. doi: 10.23919/MIPRO.2018.8400254.
[34] N. C. Mendonca, P. Jamshidi, D. Garlan, and C. Pahl, "Developing Self-Adaptive Microservice Systems: Challenges and Directions," *IEEE Softw.*, vol. 38, no. 2, pp. 70–79, Mar. 2021, doi: 10.1109/MS.2019.2955937.
[35] Q. Xiang *et al.*, "No Free Lunch: Microservice Practices Reconsidered in Industry." arXiv, Jun. 14, 2021. doi: 10.48550/arXiv.2106.07321.
[36] G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, and A. Edmonds, "An architecture for self-managing microservices," in *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, New York, NY, USA, Apr. 2015, pp. 19–24. doi: 10.1145/2747470.2747474.
[37] "Oh-My-PickleDB - JSON Database." https://tory1103.github.io/oh-my-pickledb/ (accessed Aug. 19, 2022).