Jurnal Rekursif, Volume 13 Nomor 1, Maret 2025, ISSN 2303-0755, e-ISSN 2777-0427
https://ejournal.unib.ac.id/index.php/rekursif/

Rancang Bangun Arsitektur *Microservices* untuk *Massive Open Online Courses*

Martin Mulyo Syahidin[1], Funny Farady Coastera[2], Tiara Eka Putri[3]

[1,3] Informatika, Teknik, Universitas Bengkulu, Jl. WR. Supratman, Kandang Limun, Bengkulu (38271A) +62-736-21170 dan 21884; +62-736-22105

[1]martinms.za@gmail.com
[2]ffaradyc@unib.ac.id
[3]tiaraekaputri@unib.ac.id

Abstrak: Perkembangan teknologi informasi telah mengubah cara pembelajaran, memungkinkan kegiatan belajar mengajar dilakukan jarak jauh dengan waktu fleksibel. MOOCs adalah model pembelajaran baru yang memungkinkan belajar dari mana saja dan kapan saja dengan jumlah peserta tak terbatas. Dalam pembangunan MOOCs, arsitektur *microservices* digunakan, seperti pada studi kasus Vocasia.id yang menghasilkan layanan *authentication*, *catalog*, *course*, *email*, *enrollment*, *finance*, *instructor*, *order*, dan payment dengan metode dekomposisi berdasarkan *business capabilities*. Kontainerisasi dilakukan menggunakan Docker untuk portabilitas aplikasi, serta Kubernetes untuk *deployment* dan *scaling*. *Benchmarking* dilakukan untuk menentukan konfigurasi *deployment* yang efisien, dengan hasil penggunaan CPU maksimum 28 dan memori 13 GiB. Sistem dilengkapi horizontal pod autoscaler dengan target penggunaan CPU dan memori masing-masing 60%, serta konfigurasi minimum 2 pod dan maksimum 5 pod, memastikan sistem berjalan optimal dengan sumber daya yang efisien.
Kata Kunci: vocasia.id, *microservices*, docker, kubernetes, *horizontal pod autoscaler*

Abstract: The advancement of information technology has revolutionized learning, enabling remote teaching with flexible schedules. MOOCs, a new learning model, offer accessible education anytime, anywhere, with no participant limits. In developing MOOCs, microservices architecture can be applied, as shown in the Vocasia.id case study, which includes services like authentication, catalog, course, email, enrollment, finance, instructor, order, and payment using business capability decomposition. Docker is used for containerization to ensure application portability, while Kubernetes handles orchestration for efficient deployment and scaling. Benchmarking identifies optimal configurations, with maximum CPU usage reaching 28 cores and memory usage at 13 GiB. To ensure system stability, a horizontal pod autoscaler is configured with a 60% target for CPU and memory usage, supporting a minimum of 2 pods and a maximum of 5 pods.

Keywords: vocasia.id, microservices, docker, kubernetes, horizontal pod autoscaler

I. PENDAHULUAN

Yayasan Adipurna Inovasi Asia adalah lembaga pelatihan kompetensi yang telah menyediakan pelatihan tatap muka sejak tahun 2015. Seiring perkembangan teknologi informasi, yayasan meluncurkan layanan *Massive Open Online Courses* (MOOCs) bernama Vocasia pada tahun 2020. MOOCs menawarkan kursus yang dapat diakses kapan saja dan di mana saja oleh banyak peserta secara bersamaan, memberikan kemudahan bagi peserta untuk belajar mandiri dan mengelola waktu secara efisien [1]. Namun, Vocasia.id masih menggunakan arsitektur monolitik yang menimbulkan beberapa masalah, seperti akses lambat pada jam tertentu akibat keterbatasan skalabilitas dan tidak adanya sistem *monitoring* atau *logging* yang memadai.

Arsitektur monolitik menggabungkan semua komponen sistem dalam satu entitas tunggal, sehingga jika terjadi kesalahan pada satu modul,

dapat memengaruhi seluruh [2]. Selain itu, monolitik hanya memungkinkan scaling secara vertikal dengan menambah kapasitas server, yang tidak efisien untuk menangani fluktuasi jumlah pengguna [3]. Pendekatan ini juga menyebabkan *downtime* lebih lama saat melakukan perubahan pada sistem. Masalah ini mendorong perlunya migrasi ke arsitektur *microservices* untuk meningkatkan performa, efisiensi, dan skalabilitas sistem.

Arsitektur *microservices* membagi sistem menjadi modul-modul kecil dengan fungsi spesifik, memungkinkan komunikasi antar modul menggunakan API [4]. Pendekatan ini mendukung *scaling* yang lebih fleksibel dan isolasi kesalahan. Proses *monitoring* dan *logging* pada setiap modul dilakukan secara spesifik sehingga pengembang dapat lebih mudah mengidentifikasi masalah. Pembagian modul juga mendukung sistem untuk menghadapi fluktuasi pengguna dengan menambah atau mengurangi replika modul sesuai beban kerja [5].

Microservices menggunakan metode *deployment* berbasis container dengan teknologi seperti Docker, yang mengisolasi setiap modul dalam container terpisah [6]. Teknologi *container* memungkinkan pengelolaan sumber daya yang lebih efisien dan meminimalkan dampak kerusakan antar modul. Dalam konteks ini, Kubernetes menjadi alat orkestrasi yang mendukung pengelolaan aplikasi berbasis container, termasuk scaling otomatis, *load balancing*, dan *self-healing* [7].

Kubernetes menyediakan mekanisme Horizontal Pod Autoscaler (HPA) untuk scaling otomatis menggunakan metrik CPU dan memori. Namun, penentuan *threshold* CPU dan memori memerlukan analisis yang tepat agar proses *scaling* berjalan optimal [8]. Dengan HPA, Kubernetes dapat menyesuaikan jumlah pod berdasarkan beban kerja, memastikan sistem tetap responsif bahkan dalam kondisi beban tinggi. Selain itu, fitur *load balancing* dan *self-healing* memastikan ketersediaan dan reliabilitas sistem.

Melalui penerapan arsitektur *microservices* dan Kubernetes, Vocasia.id dapat mengatasi keterbatasan arsitektur monolitik dan meningkatkan efisiensi sistem. Solusi ini memungkinkan sistem berjalan lebih fleksibel, responsif, dan andal, mendukung pengalaman belajar *online* yang optimal bagi para penggunanya.

II. LANDASAN TEORI

A. *Arsitektur Microservices*

Microservices adalah arsitektur perangkat lunak berbasis sistem terdistribusi yang membagi sistem besar menjadi layanan-layanan kecil dengan fungsi spesifik dan sifat independen, sehingga memudahkan pengembangan, *upgrade*, dan penggantian komponen [9]. Berbeda dari arsitektur monolitik yang menggabungkan seluruh komponen dalam satu kesatuan, *microservices* memisahkan komponen dengan *interface* berupa API [10]. Keuntungan *microservices* meliputi fleksibilitas teknologi, resiliensi terhadap kegagalan, skalabilitas yang efisien, kemudahan pengembangan, penyelarasan organisasi, kemudahan komposisi layanan, dan kemampuan mengganti komponen dengan mudah [11]. Namun, *microservices* juga menghadapi tantangan, seperti identifikasi komponen yang kompleks, batasan antar komponen yang sulit ditentukan, asumsi berbeda terkait *state* dan konsistensi dalam sistem terdistribusi, risiko kegagalan baru, serta kesulitan memahami perilaku sistem secara keseluruhan [12].

B. *RESTful API*

RESTful API adalah komponen sistem yang berfungsi sebagai mediator untuk mengakses dan menggunakan sumber daya dalam sistem melalui protokol HTTP, dengan sumber daya diidentifikasi menggunakan URI (*Uniform Resource Identifier*) sebagai jembatan komunikasi data [13]. API, atau *Application Programming Interface*, merupakan antarmuka yang terdiri atas kumpulan instruksi dalam *library* yang memungkinkan sistem saling berkomunikasi, dengan spesifikasi struktur data, objek, dan fungsi yang dapat diakses melalui URL [14]. RESTful API mendukung operasi CRUD (*create, read, update, delete*) melalui berbagai HTTP method, seperti GET, POST, PUT, PATCH, dan DELETE, untuk mengelola data secara efisien di bagian *back-end*.

C. *Kubernetes*

Kubernetes adalah platform *open-source* untuk *container orchestration* yang dikembangkan oleh Google, dirancang untuk menjalankan aplikasi berbasis container secara *scalable* dan *reliable* melalui *application-oriented APIs* [15]. Kubernetes memungkinkan pengelolaan sistem terdistribusi dengan menyediakan fitur seperti pembuatan, modifikasi, replikasi, dan penghapusan *container*. Sistem ini berfokus pada *cluster*, yaitu kumpulan *virtual machine* yang terdiri dari *master nodes* untuk pengelolaan dan *worker nodes* untuk menjalankan aplikasi. Unit terkecil dalam Kubernetes adalah pod, yang berisi satu atau lebih *container* dengan *resource* yang dialokasikan bersama. Fitur seperti Horizontal Pod Autoscaler (HPA) memungkinkan penyesuaian jumlah pod secara otomatis berdasarkan beban kerja dan metrik *resource*, seperti CPU dan memori, untuk menjaga performa aplikasi [16].

D. *Massive Open Online Courses*

Massive Open Online Courses (MOOCs) merupakan bentuk pendidikan berbasis teknologi dan virtual yang memungkinkan pembelajaran dilakukan secara online dengan peserta yang dapat mencapai ribuan orang [17]. MOOCs menyediakan akses terbuka untuk semua orang, memungkinkan peserta untuk memilih materi yang diminati dan mengaksesnya kapan saja dan di mana saja. Karakteristik utama MOOCs meliputi skala yang tidak terbatas (*massive*), akses yang terbuka bagi siapa saja (*open*), sepenuhnya dilakukan secara online (*online*), dan dikelola sebagai perkuliahan yang utuh dengan berbagai kegiatan seperti membaca materi, mengikuti ujian, mengerjakan tugas, dan berpartisipasi dalam forum diskusi. Beberapa layanan MOOCs juga menawarkan materi secara gratis, sementara yang lainnya mengenakan biaya untuk sertifikat atau ujian.

III. METODOLOGI PENELITIAN

A. *Jenis Penelitian*

Jenis penelitian yang dilakukan adalah *applied research.* Penelitian ini bertujuan untuk mencari solusi dari masalah-masalah tertentu dan melakukan pemecahan masalah praktis yang timbul ataupun menghasilkan suatu produk yang memiliki fungsi praktis lainnya.

B. *Metode Pengumpulan Data*

1. Studi Pustaka

Metode ini dilakukan dengan cara mengumpulkan data dari berbagai literatur seperti buku, jurnal, buku elektronik dan media internet yang berkaitan dengan penelitian yang akan dikerjakan.

2. Studi Lapangan

Metode ini dilakukan dengan cara melakukan pengamatan langsung pada objek yang akan diteliti untuk memperoleh data. Adapun teknik pengumpulan data yang dilakukan adalah sebagai berikut:

- Observasi

  Observasi dilakukan dengan melakukan analisis terhadap proses bisnis yang sudah berjalan di vocasia.id

- Wawancara

  Wawancara dilakukan untuk mengetahui permasalahan dan proses bisnis yang sedang berjalan pada sistem informasi yang sudah berjalan.

3. Desain Penelitian

Metode pengembangan sistem yang digunakan dalam penelitian ini adalah SDLC (*System Development Life Cycle*). Dalam metode SDLC terdapat beberapa model atau jenis, salah satunya model *waterfall* yang mana digunakan dalam penelitian ini. Adapun tahapan model *waterfall* secara garis besar dalam pengembangan sistem yang akan digunakan adalah sebagai berikut:

a. Analisis Kebutuhan

Pada tahap ini dilakukan penetapan kebutuhan-kebutuhan dalam pengembangan sistem informasi. Pada tahapan ini pula terjadi pengidentifikasian tentang data apa saja yang dibutuhkan dalam proses pembuatannya.

b. Perancangan Sistem

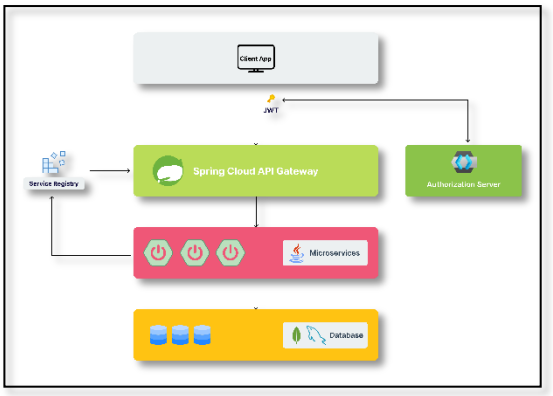
Setelah dilakukan analisis kebutuhan, selanjutnya dilakukan perancangan berdasarkan hasil analisis tersebut. Perancangan ini bertujuan untuk memberikan gambaran mengenai sistem yang akan dibangun dan interaksi pengguna dengan sistem.

c. Implementasi

Pada tahap ini dilakukan implementasi hasil perancangan menjadi sistem menggunakan bahasa pemrograman Java.

IV. HASIL DAN PEMBAHASAN

A. *Arsitektur Microservices*



Gambar 1. Arsitektur Microservices

Arsitektur sistem yang dihasilkan terdiri dari *client app*, *identifier token* (JWT), *authorization server*, *API gateway*, *service registry*, *service layer*, dan *persistence layer*. *Client app* berfungsi sebagai antarmuka pengguna yang menghubungkan *user* dengan *back-end*, menangani *input*, serta menampilkan data dari *server*, baik melalui aplikasi web, *mobile*, atau aplikasi pihak ketiga. Untuk memastikan keamanan, digunakan JWT token sebagai metode otentikasi dan otorisasi, yang dikeluarkan dan divalidasi oleh *authorization server*, yang juga mengimplementasikan *access policies* berbasis *role* (siswa, instruktur, admin). *API gateway* bertindak sebagai gerbang utama untuk mengelola *routing*, otentikasi, otorisasi, manipulasi data, serta pembagian beban ke *service layer*. *Service registry*, seperti Spring Cloud Kubernetes Discovery Agent, mengelola alamat *instance microservices* dan memastikan komunikasi antar *service* melalui *health check* dan registrasi. Di *service layer*, *microservices* yang dibangun

menggunakan Spring Boot bertanggung jawab memproses permintaan pengguna, mengembalikan data dalam format JSON, serta mendukung *monitoring*, pengujian, dan *autoscaling*. Terakhir, *persistence layer* mengelola data secara permanen menggunakan MySQL, MongoDB, dan Redis sebagai *server cache* untuk mendukung performa sistem.

B. *Logging dan Monitoring*

Gambar 2. Logging dan Monitoring

Proses *monitoring* dan *logging* menggunakan beberapa *tools*, yaitu OpenTelemetry, Loki, Prometheus, Tempo, dan Grafana, untuk memberikan gambaran menyeluruh tentang performa dan log sistem. Loki menjadi sumber data untuk Grafana dan memungkinkan administrator mengakses log dari satu tempat tanpa harus membuka *container* secara langsung. Dengan Loki, administrator dapat melakukan pencarian cepat, filter log sesuai kebutuhan, dan integrasi dengan Tempo untuk *tracing request*. Prometheus melakukan scraping data dari semua service seperti API Gateway dan Eureka Discovery Agent secara berkala. Data Prometheus dapat diakses melalui Grafana untuk memantau metrik sistem seperti penggunaan CPU dan memori, membantu deteksi dini masalah melalui peringatan ambang batas. Tempo digunakan untuk tracing request dengan *trace ID*, memberikan visualisasi detail proses yang terjadi dalam suatu *request*, termasuk metode yang dipanggil, *query* ke *database*, dan komunikasi antar *service*, sehingga memudahkan analisis *bottleneck*. Seluruh data dari Loki, Prometheus, dan Tempo divisualisasikan melalui Grafana, yang memungkinkan pembuatan dasbor kustom untuk menampilkan data dari berbagai sumber secara terpusat. Integrasi keempat tools ini memberikan solusi *monitoring* dan *logging* yang efektif, membantu administrator memelihara stabilitas sistem dan meningkatkan pengalaman pengguna.

C. *Benchmarking*

Benchmarking dilakukan untuk menguji performa sistem dengan memberikan beban kerja secara bertahap guna mengukur kemampuan sistem menangani peningkatan beban. Pengujian menggunakan Grafana k6 dilakukan selama 7 menit dengan jumlah pengguna aktif mulai dari 50 hingga 200, yang merepresentasikan jumlah pengguna aktif berdasarkan hasil wawancara. Setiap *service* diuji dengan konfigurasi yang sama melalui tiga kali pengujian untuk mendapatkan hasil yang konsisten. Jumlah maksimum 200 pengguna dipilih untuk menggambarkan kinerja sistem saat menghadapi fluktuasi beban kerja. Hasil dari *benchmarking* ini memberikan data penting tentang performa sistem yang kemudian digunakan untuk menentukan konfigurasi resource yang optimal.

D. *Container Deployment*

Deployment container dilakukan di Google Kubernetes Engine (GKE) dengan 3 *nodes*, dimana setiap *service* memiliki minimum 2 pod untuk memastikan ketersediaan dan menghindari kegagalan total. Alokasi CPU dan memori ditentukan menggunakan *request* dan *limit*, dengan CPU dalam satuan *milicores* (1 CPU = 1.000 *milicores*) dan memory dalam satuan *mebibytes* (MiB). Horizontal Pod Autoscaler (HPA) secara

otomatis menambah atau mengurangi jumlah pod berdasarkan pemantauan penggunaan CPU terhadap ambang batas yang ditentukan. Jika penggunaan CPU melebihi *threshold*, HPA akan menambah pod hingga batas maksimum yang dikonfigurasi, sedangkan jika menurun di bawah *threshold*, HPA akan mengurangi pod hingga batas minimum. Nilai request didasarkan pada rata-rata penggunaan *resource* selama pengujian, sementara limit ditentukan berdasarkan penggunaan *resource* tertinggi selama pengujian. Konfigurasi *deployment* mencakup jumlah replika maksimum, limit CPU, dan limit memori, serta *threshold* HPA.

Tabel 1. Konfigurasi *Deployment Kubernetes*

No.	*Service*	*Max Replica*	CPU Limit (m)	Memory Limit (MiB)	HPA CPU (%)
1.	*Authentication*	5	1000	500	77
2.	*Catalog*	5	400	150	85
3.	*Course*	5	1000	200	70
4.	*Email*	5	400	150	90
5.	*Enrollment*	5	500	300	73
6.	*Finance*	5	1000	300	81
7.	*Instructor*	5	400	300	82
8.	*Order*	5	700	400	87
9.	*Payment*	5	300	200	83

V. KESIMPULAN DAN SARAN

A. *Kesimpulan*

Perancangan arsitektur *microservices* untuk sistem MOOCs dimulai dengan identifikasi proses bisnis menggunakan metode *decomposition by business capabilities*, menghasilkan sembilan service utama: *authentication*, *catalog*, *course*, *email*, *enrollment*, *finance*, *instructor*, *order*, dan *payment service*. Implementasi *deployment container* menggunakan Kubernetes dengan konfigurasi maksimum 28 CPU dan 13 GiB memori, serta *auto-scaling* melalui horizontal pod autoscaler yang menjaga performa sistem dengan parameter penggunaan CPU dan memori sebesar 60%. Hasil penelitian menunjukkan bahwa arsitektur *microservices* yang diorkestrasi menggunakan Kubernetes mendukung skalabilitas dan efisiensi penggunaan resource dalam pengelolaan sistem MOOCs.

B. *Saran*

Microservices adalah arsitektur yang memerlukan perancangan matang untuk mendukung proses *development* dan pengelolaan sistem secara optimal. Migrasi dari arsitektur monolitik ke *microservices* dapat dilakukan secara bertahap untuk memastikan proses bisnis tetap berjalan lancar. Pemilihan teknologi disesuaikan dengan kebutuhan, seperti PostgreSQL untuk *relational database* dan MongoDB untuk data berbasis dokumen. Penggunaan aplikasi tambahan seperti Istio atau Kong sebagai API Gateway, serta Eureka atau Konsul sebagai *service registry*, membantu memenuhi kebutuhan pengembangan. Selain itu, implementasi *service mesh* seperti Istio dengan mTLS direkomendasikan untuk memastikan keamanan komunikasi antar *microservice*.

VI. REFERENSI

[1] M. Bagus Pambudi and S. Chandra Wibawa, "Pengaruh Model Pembelajaran Massive Open Online Coutses Terhadap Hasil Belajar Peserta Didik," *Jurnal IT-EDU*, vol. 05, no. 1, pp. 294–302, 2020.

[2] S. Waruwu, I. Kadek, and D. Nuryana, "Implementasi Arsitektur Monolitik Pada Rancang Bangun Sistem Informasi," *Journal of Informatics and Computer Science*, vol. 4, no. 4, pp. 399–404, 2023.

[3] D. Putra Permana, H. Ashaury, and P. Nurul Sabrina, "Reengineering Arsitektur

Monolithic ke Microservices pada Website Management Content MQTV," *Jurnal Locus Penelitian dan Pengabdian*, vol. 2, no. 9, pp. 924–931, Oct. 2023, doi: 10.58344/locus.v2i9.1723.

[4] A. Sinambela, Ernawati, and F. Farady Coastera, "Implementasi Arsitektur Microservices Pada Rancang Bangun Aplikasi Marketplace Berbasis Web," *Jurnal Rekursif*, vol. 9, no. 1, pp. 1–13, 2021, [Online]. Available: http://ejournal.unib.ac.id/index.php/rekursif/1

[5] K. Yedutun, A. Noertjahyana, and H. Novianus Palit, "Implementasi Container Kubernetes untuk Mendukung Scalability," *Jurnal Infra*, vol. 7, no. 2, 2019, Accessed: Nov. 10, 2023. [Online]. Available: https://publication.petra.ac.id/index.php/teknik-informatika/article/view/8774

[6] R. Senington, B. Pataki, and X. V. Wang, "Using docker for factory system software management: Experience report," in *Procedia CIRP*, Elsevier B.V., 2018, pp. 659–664. doi: 10.1016/j.procir.2018.03.173.

[7] H. Ahmad, C. Treude, M. Wagner, and C. Szabo, "Smart HPA: A Resource-Efficient Horizontal Pod Auto-scaler for Microservice Architectures," Feb. 2024, doi: 10.1109/ICSA59870.2024.00013.

[8] O. Pozdniakova, D. Mažeika, and A. Cholomskis, "SLA-Adaptive Threshold Adjustment for a Kubernetes Horizontal Pod Autoscaler," *Electronics (Switzerland)*, vol. 13, no. 7, Apr. 2024, doi: 10.3390/electronics13071242.

[9] G. Munawar and A. Hodijah, "Analisis Model Arsitektur Microservice Pada Sistem Informasi DPLK," *Publikasi Jurnal & Penelitian Teknik Informatika*, vol. 3, no. 1, pp. 232–239, 2018.

[10] S. Sharma, Rajesh RV, and D. Gonzalez, *Microservices: Building Scalable Software*, 1st ed. Mumbai: Packt Publishing, 2017. Accessed: Feb. 25, 2024. [Online]. Available: https://arc.cct.ie/fac_books/1/

[11] S. Newman, *Building Microservices*. O'Reilly Media, Inc, 2015. [Online]. Available: http://safaribooksonline.com

[12] M. Bruce and P. A. Pereira, *Microservices in Action*. 2018.

[13] G. N. Tanaem, P. F., Wijaya, A. F., Manuputty, A. D., & Huwae, "Penerapan RESTFul Web Service Pada Disain Arsitektur Sistem Informasi Pada Perguruan Tinggi (Studi Kasus: STARS UKSW)," *JASIEK (Jurnal Aplikasi Sains, Informasi, Elektronika dan Komputer)*, vol. 2, no. 1, pp. 11–20, 2020.

[14] M. F. Ramadhani, "Pembangunan Aplikasi Informasi, Pengaduan, Kritik, Dan Saran Seputar Kota Cimahi Pada Platform Android," *Jurnal Ilmiah Komputer dan Informatika (KOMPUTA)*, p. 9, 2018.

[15] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, *Berg, Omega and Kubernetes*, 5th ed., vol. 59. Queue, 2016. doi: https://doi.org/10.1145/2898442.2898444.

[16] F. Rossi, V. Cardellini, and F. Lo Presti, "Hierarchical Scaling of Microservices in Kubernetes," in *Proceedings - 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2020*, Institute of Electrical and Electronics Engineers Inc., Aug. 2020, pp. 28–37. doi: 10.1109/ACSOS49614.2020.00023.

[17] M. Dihan and A. Mochtar, “Massive Open Online Course (MOOC) Sebagai Sistem Pembelajaran Online yang Efektif dan Efisien,” 2021.