Dokumen Desain Arsitektur Perangkat Lunak

LMS KALAM UMI

DOSEN PENGAMPUH: Mardiyyah Hasnawi

Disusun Oleh :
KELOMPOK 2

Rifky Fahreza	**13020230050**
Jumadil	**13020230052**
Andrian Sutrisman Laturi	**13020210074**
Muh. Ichwan Ardi Arif	**13020210289**

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS MUSLIM INDONESIA
MAKASSAR

2025

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi yang semakin pesat telah mendorong perguruan tinggi untuk mengadopsi sistem pembelajaran digital. Universitas Muslim Indonesia (UMI) merupakan salah satu institusi yang telah menerapkan sistem e-learning melalui platform **Kalam UMI** sebagai media utama untuk mendukung kegiatan belajar mengajar daring maupun hybrid. Platform ini bertujuan mempermudah akses terhadap materi, tugas, dan interaksi akademik antara dosen dan mahasiswa. Namun, dalam praktiknya, efektivitas sistem masih menghadapi kendala pada aspek penggunaan dan keandalan sistem.

Hasil observasi terhadap dosen pengguna Kalam UMI menunjukkan bahwa sebagian besar merasa cukup mudah dalam mengunggah dan mengelola materi perkuliahan, tetapi fitur interaktif seperti kuis, latihan, dan forum diskusi dinilai masih kurang optimal. Sebagian dosen bahkan lebih memilih menggunakan platform komunikasi eksternal seperti WhatsApp atau Telegram untuk menunjang interaksi kelas. Di sisi lain, mahasiswa melaporkan kendala teknis seperti *loading* lambat, *error* saat mengunggah tugas, serta kesulitan mengakses sistem melalui perangkat seluler. Hal ini menunjukkan bahwa desain antarmuka dan performa sistem masih perlu ditingkatkan .

1.2 Kebutuhan Sistem

Dari hasil tersebut, dapat diidentifikasi tiga kebutuhan utama dalam pengembangan sistem pembelajaran daring di UMI. Pertama, kebutuhan fungsional untuk meningkatkan kemudahan akses dan pengelolaan konten pembelajaran agar dosen dapat mengunggah dan memperbarui materi dengan cepat. Kedua, kebutuhan interaktivitas dan kolaborasi melalui forum diskusi, kuis daring, serta umpan balik yang lebih komunikatif. Ketiga, kebutuhan akan kinerja sistem yang stabil dengan dukungan non-fungsional seperti keamanan, skalabilitas, dan performa optimal .

1.3 Stakeholders dan Scope Sistem

Stakeholder utama dalam sistem ini mencakup dosen sebagai penyedia konten, mahasiswa sebagai pengguna, serta unit teknologi informasi kampus sebagai pengembang dan pengelola. Lingkup sistem meliputi seluruh aktivitas pembelajaran digital di kampus, termasuk pengelolaan materi, evaluasi, dan komunikasi akademik daring. Penelitian sebelumnya banyak berfokus pada platform global seperti Moodle atau Google Classroom, sementara riset terhadap platform internal kampus seperti Kalam UMI masih terbatas. Oleh karena itu, penelitian ini penting untuk menggali persepsi pengguna, kebutuhan sistem, serta merumuskan rekomendasi peningkatan yang sesuai dengan konteks kampus.

Penelitian ini bertujuan untuk menganalisis pengalaman pengguna terhadap sistem pembelajaran daring di lingkungan Universitas Muslim Indonesia, mengidentifikasi faktor-faktor yang memengaruhi efektivitas penggunaan sistem, serta memberikan rekomendasi pengembangan agar sistem lebih efisien, aman, dan ramah pengguna.

BAB II

ARCHITECTURE THINKING

2.1 Business Drivers

Dalam konteks pengembangan sistem pembelajaran daring di lingkungan Universitas Muslim Indonesia (UMI), terdapat sejumlah faktor utama yang menjadi pendorong (*business drivers*) dalam perancangan arsitektur perangkat lunak. Tiga *business driver* utama tersebut adalah: (1) efisiensi dan aksesibilitas pembelajaran, (2) peningkatan kualitas interaksi akademik, dan (3) keandalan serta keamanan sistem akademik.

Pertama, efisiensi dan aksesibilitas menjadi faktor penting agar sistem dapat diakses lintas perangkat dan waktu. Kedua, sistem harus mendukung interaksi dua arah melalui fitur seperti forum, kuis, dan umpan balik *real-time*. Ketiga, keandalan dan keamanan diperlukan untuk melindungi data sensitif dan menjaga stabilitas sistem, yang mencerminkan reputasi institusi.

2.2 Trade-offs

Setiap keputusan arsitektural memerlukan kompromi antara berbagai aspek. Berikut tiga *trade-off* utama yang perlu dipertimbangkan dalam pengembangan sistem pembelajaran digital di UMI:

No	Trade-off	Deskripsi	Pertimbangan Arsitektural
1	Kinerja vs Biaya Infrastruktur	Peningkatan performa memerlukan investasi lebih besar pada server dan bandwidth.	Menggunakan arsitektur *cloud hybrid* agar menyeimbangkan kinerja dan efisiensi biaya.
2	Fleksibilitas Fitur vs Kompleksitas Sistem	Penambahan fitur baru meningkatkan nilai guna namun memperumit sistem.	Menerapkan desain modular agar fitur dapat ditambah tanpa mengubah keseluruhan struktur.
3	Skalabilitas vs Keamanan	Sistem yang mudah diperluas meningkatkan potensi risiko keamanan.	Menggunakan *API gateway* dan autentikasi berlapis untuk menjaga keseimbangan.

2.3 Tanggung Jawab: Architect vs Developer

Dalam proses pembangunan sistem pembelajaran daring, terdapat pembagian tanggung jawab yang jelas antara *software architect* dan *developer*. Architect berperan dalam perancangan tingkat tinggi, pemilihan gaya arsitektur, serta standar integrasi antar komponen. Developer berfokus pada implementasi teknis sesuai rancangan dan memastikan efisiensi serta *maintainability* sistem. Kolaborasi antara keduanya menjadi kunci keberhasilan proyek perangkat lunak.

2.4 Balance: Performa, Biaya, Kompleksitas, dan Maintainability

Dalam perancangan sistem, keseimbangan antara performa, biaya, kompleksitas, dan *maintainability* merupakan hal penting. Sistem dengan performa tinggi membutuhkan infrastruktur besar, sehingga solusi berbasis *cloud* dengan skala dinamis dapat menjadi pilihan efisien. Selain itu, pendekatan *modular* dan *service-oriented architecture (SOA)* dapat menjaga kompleksitas tetap terkendali tanpa mengorbankan kemampuan pemeliharaan.

Keseimbangan ini memastikan sistem pembelajaran daring tetap efisien, mudah dikelola, dan mampu memberikan pengalaman belajar optimal bagi civitas akademika Universitas Muslim Indonesia.

BAB III

KARAKTERISTIK ARSITEKTUR

Dalam pengembangan sistem Kalam UMI, karakteristik arsitektur berperan penting untuk memastikan sistem tidak hanya memenuhi kebutuhan pengguna, tetapi juga mampu bertahan dan berkembang dalam jangka panjang. Berdasarkan kebutuhan bisnis dan hasil observasi, lima karakteristik kritikal dipilih sebagai fondasi rancangan sistem.

Karakteristik	Kategori	Prioritas	Justifikasi	Metrik Target
Availability	Operational	Tinggi	Sistem harus selalu tersedia agar proses belajar tidak terganggu.signifikan. Downtime berdampak langsung pada aktivitas akademik.	Uptime ≥ 99% selama periode akademik berjalan.

Security	Cross-cutting	Sangat Tinggi	Melindungi data akun, nilai, dan dokumen dari akses tidak sah.	Enkripsi data, autentikasi ganda, audit
Scalability	Structural	Tinggi	Sistem harus mampu menangani lonjakan pengguna saat ujian atau registrasi.	≥ 2000 koneksi simultan, respon < 2 detik
Usability	Cross-cutting	Sedang	UI intuitif meningkatkan efisiensi dan keterlibatan pengguna.	Skor *System Usability Scale (SUS)* ≥ 80.
Maintainability	Structural	Sedang	Sistem harus mudah diperbarui tanpa gangguan layanan.	Waktu perbaikan bug kritis < 4 jam; kompatibilitas penuh setelah update.

Kelima karakteristik ini saling mendukung: Availability dan Security menjaga stabilitas operasional, Scalability memastikan sistem tetap responsif saat beban tinggi, Usability meningkatkan kenyamanan pengguna, dan Maintainability menjamin keberlanjutan sistem. Pemilihan karakteristik ini didasarkan pada business drivers dan konteks kampus, serta menjadi acuan dalam pengambilan keputusan arsitektural di bab-bab selanjutnya.

BAB IV

GAYA ARSITEKTUR

Pemilihan gaya arsitektur dalam pengembangan sistem perangkat lunak menjadi aspek penting yang menentukan bagaimana komponen-komponen sistem dirancang, berinteraksi, dan dipelihara. Dalam konteks pengembangan **sistem pembelajaran daring Kalam UMI**, gaya arsitektur yang dipilih adalah **Layered Architecture**. Gaya ini memberikan pemisahan tanggung jawab yang jelas antar lapisan sistem, memudahkan

pemeliharaan (*maintainability*), meningkatkan keamanan (*security*), serta menjaga fleksibilitas dalam pengembangan dan integrasi fitur baru.

4.1 Alasan Pemilihan Layered Architecture

Layered Architecture dipilih karena cocok untuk sistem pembelajaran daring yang memiliki alur data terstruktur dan tanggung jawab terpisah. Gaya ini memudahkan pengembangan bertahap, pengujian modul, dan pemeliharaan sistem. Dibandingkan microservices atau event-driven, pendekatan ini lebih sederhana dan sesuai dengan kapasitas tim pengembang kampus.

4.2 Struktur Lapisan dan Komponen

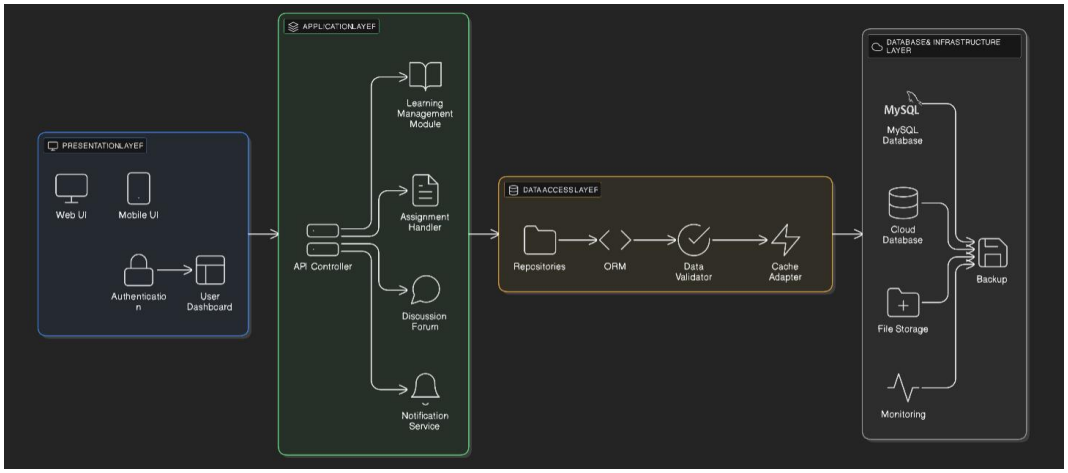
Sistem Kalam UMI dibagi menjadi empat lapisan utama:

- **Presentation Layer**
  Antarmuka pengguna web dan mobile, autentikasi, dan dashboard. Bertugas menerima input dan menampilkan hasil proses.
- **Application Layer**
  Logika bisnis: manajemen kelas, tugas, penilaian, notifikasi, dan API controller. Menghubungkan UI dengan data.
- **Data Access Layer**
  Repositori data, ORM, validasi, dan cache adapter. Bertugas mengelola komunikasi antara aplikasi dan database.
- **Database & Infrastructure Layer**
  MySQL/Cloud DB, file storage, monitoring, dan backup. Menjamin ketersediaan, keamanan, dan performa sistem.

Alur data mengalir dari pengguna ke UI, diproses oleh Application Layer, lalu disimpan atau diambil melalui Data Access Layer ke Database Layer. Integrasi antar lapisan menggunakan RESTful API.

4.3 Diagram Arsitektur Tingkat Tinggi

Diagram berikut menggambarkan alur interaksi antar lapisan pada arsitektur sistem Kalam UMI:



Alur data dimulai dari pengguna melalui antarmuka web atau mobile, diteruskan ke *Application Layer* untuk diproses oleh modul logika bisnis, lalu data disimpan atau diambil dari *Database Layer* melalui *Data Access Layer*. Integrasi antar lapisan menggunakan *RESTful API* untuk memastikan fleksibilitas, keamanan, dan kemudahan pengujian.

4.4 Best Practices

- **Separation of Concerns**: Tiap lapisan hanya menangani satu jenis tugas.
- **Caching & Asynchronous Processing**: Materi dan jadwal disimpan sementara, notifikasi massal diproses async.
- **CI/CD & Monitoring**: Automasi deployment dan observasi performa dengan tools seperti Prometheus dan ELK Stack.

4.5 Trade-offs

Walaupun memiliki banyak keunggulan, Layered Architecture juga menimbulkan beberapa kompromi desain:

No.	Trade-off	Dampak	Solusi Mitigasi
1	**Sederhana tapi Kurang Fleksibel**	Sulit diubah menjadi layanan independen saat sistem tumbuh besar.	Dapat diadaptasi menjadi *modular monolith* sebagai langkah awal menuju *microservices*.
2	**Komunikasi Antar Lapisan**	Setiap pemanggilan API menambah	Gunakan *caching* dan *batch processing* untuk

	Tambah Overhead	latensi waktu respon.	mengurangi beban komunikasi.
3	**Keterbatasan Skalabilitas Independen**	Semua modul terikat dalam satu unit deploy.	Gunakan *load balancing* dan *containerization* untuk memperluas kapasitas sistem.

BAB V

PRINSIP ARSITEKTUR

Dalam perancangan sistem pembelajaran daring **Kalam UMI**, penerapan prinsip arsitektur perangkat lunak bertujuan untuk memastikan sistem memiliki struktur yang mudah dipelihara, dikembangkan, dan diperluas di masa depan. Bab ini membahas tiga prinsip utama yang menjadi dasar rancangan, yaitu **modularitas**, **separation of concerns**, dan **penerapan prinsip SOLID** pada komponen inti sistem.

5.1 Modularitas

Modularitas diterapkan dengan membagi sistem menjadi beberapa modul independen berdasarkan proses bisnis: Learning Module, Assignment Module, Notification Module, dan User Management. Setiap modul memiliki tanggung jawab spesifik dan berkomunikasi melalui API. Pendekatan ini memungkinkan pengembangan paralel, isolasi kesalahan, dan pembaruan tanpa memengaruhi modul lain.

Contoh: Assignment Module terhubung ke User Management untuk mengambil data dosen/mahasiswa, dan ke Notification Module untuk mengirim pengingat tugas.

5.2 Separation of Concerns

Prinsip ini diimplementasikan melalui Layered Architecture yang memisahkan tanggung jawab sistem ke dalam empat lapisan:

- **Presentation Layer**: Menangani UI dan input pengguna
- **Application Layer**: Menjalankan logika bisnis
- **Data Access Layer**: Mengelola komunikasi dengan database
- **Infrastructure Layer**: Menyediakan penyimpanan, keamanan, dan monitoring

Dengan pemisahan ini, perubahan pada satu lapisan tidak memengaruhi lapisan lain, sehingga sistem lebih mudah diuji dan dikembangkan.

5.3 Prinsip SOLID

1. Single Responsibility Principle (SRP)

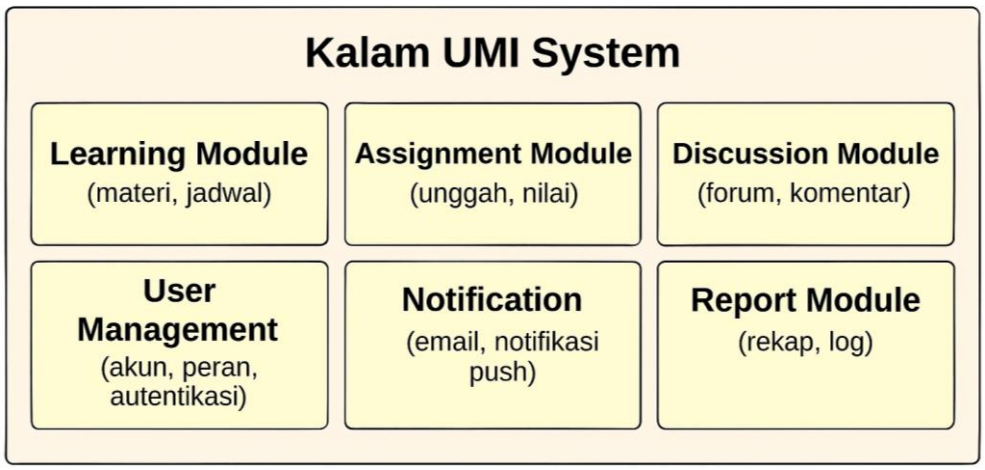
Setiap kelas hanya memiliki satu tanggung jawab.
Contoh: AssignmentService hanya menangani logika tugas, sedangkan FileValidator menangani validasi file.

2. Dependency Inversion Principle (DIP)

Modul tingkat tinggi bergantung pada abstraksi, bukan implementasi langsung.
Contoh: NotificationService menggunakan interface INotificationProvider, sehingga mudah menambahkan jenis notifikasi baru tanpa mengubah logika utama.

Penerapan prinsip-prinsip ini meningkatkan fleksibilitas, mengurangi ketergantungan antar komponen, dan memudahkan pengembangan fitur baru secara berkelanjutan

Struktur pembagian modul sistem digambarkan sebagai berikut:



Setiap modul memiliki tanggung jawab spesifik namun tetap dapat berinteraksi melalui *Application Layer* dengan pola *service-to-service communication*. Misalnya, Assignment Module berinteraksi dengan User Management untuk mengakses data dosen dan mahasiswa yang relevan, sementara Notification Module terhubung ke Assignment Module untuk mengirimkan pengingat tenggat waktu tugas.

Penerapan modularitas ini meningkatkan *maintainability* dan *scalability* sistem karena pembaruan pada satu modul tidak memengaruhi modul lainnya selama antarmuka tetap konsisten.

BAB VI

POLA DAN DESIGN PATTERN

6.1 Pola Arsitektur

1. API Gateway Pattern
Masalah: Klien harus berinteraksi langsung dengan banyak layanan, menyebabkan duplikasi logika dan kompleksitas.
Solusi: API Gateway sebagai pintu masuk tunggal, menangani autentikasi, routing, dan agregasi data.
Implementasi: Kalam UMI menggunakan gateway untuk menghubungkan web/mobile client dengan layanan internal seperti Assignment dan Notification.
Benefit: Sederhana, aman, dan scalable.

2. CQRS Pattern (Command Query Responsibility Segregation)

Masalah: Operasi baca yang intensif mengganggu performa tulis.
Solusi: Pisahkan operasi baca dan tulis ke model dan layanan berbeda.
Implementasi: Command Service untuk unggah tugas, Query Service untuk akses cepat ke data tugas dan notifikasi.
Benefit: Performa baca meningkat, beban database berkurang.

6.2 Design Pattern Berorientasi Objek

1. Factory Pattern
Masalah: Pembuatan objek notifikasi tergantung jenis pengguna.
Solusi: Gunakan Factory untuk membuat objek dinamis tanpa mengubah logika utama.
Implementasi: `NotificationFactory.create("email")` menghasilkan objek sesuai kebutuhan.
Benefit: Mudah diperluas, low coupling.

2. Observer Pattern
Masalah: Mahasiswa perlu mendapat notifikasi otomatis saat dosen mengunggah tugas.
Solusi: Gunakan Observer, di mana AssignmentService memberi tahu NotificationService saat ada perubahan.
Benefit: Efisien, event-driven, minim polling.

3. Strategy Pattern
Masalah: Sistem mendukung berbagai skema penilaian (rata-rata, tertinggi, berbobot).
Solusi: Gunakan Strategy untuk mengganti algoritma penilaian secara dinamis.
Implementasi: `GradeCalculator` menerima strategi seperti `WeightedStrategy`.
Benefit: Fleksibel, mudah diuji, sesuai prinsip DIP.

6.3 Diagram UML — Strategy Pattern

Diagram menunjukkan:

- Interface `GradingStrategy`
- Implementasi: `AverageStrategy`, `HighestStrategy`, `WeightedStrategy`
- `GradeCalculator` bergantung pada abstraksi, bukan implementasi langsung

Pola ini mencerminkan prinsip SOLID dan memungkinkan sistem penilaian yang fleksibel dan mudah diperluas.

Kalau kamu mau, aku bisa bantu ubah semua ringkasan ini jadi versi siap salin ke Word atau bantu lanjut ke bagian lampiran dan referensi. Mau lanjut ke mana?

Diagram berikut menggambarkan hubungan antar kelas dalam **Strategy Pattern** yang digunakan pada modul penilaian Kalam UMI:

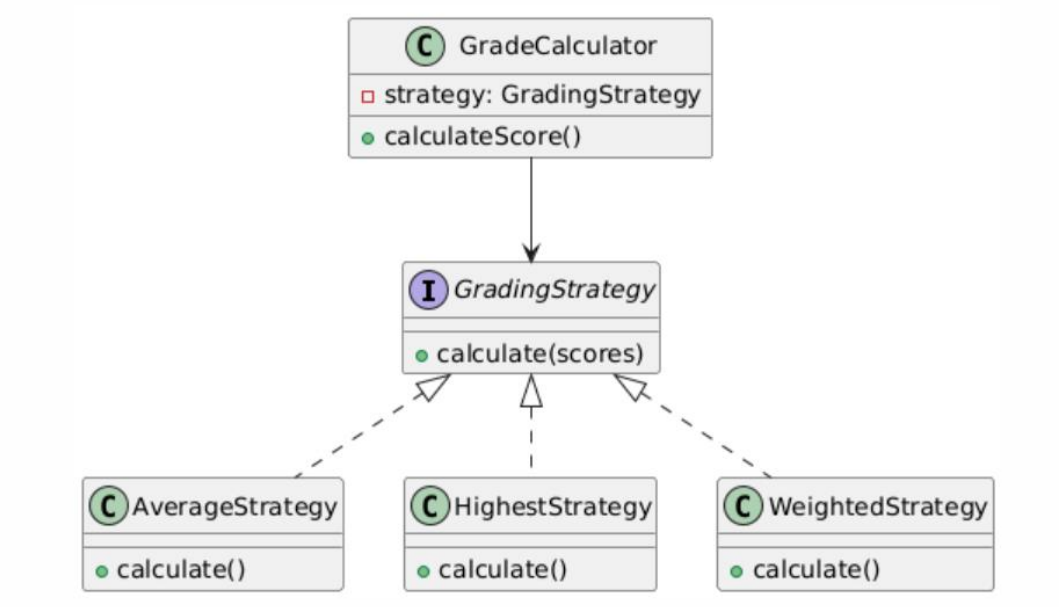


Diagram di atas menunjukkan bahwa GradeCalculator berinteraksi dengan antarmuka GradingStrategy, bukan dengan implementasi konkret. Hal ini mencerminkan prinsip **Dependency Inversion**, yang meningkatkan fleksibilitas sistem dan memudahkan penggantian strategi tanpa memodifikasi kelas utama.

BAB VII

KESIMPULAN

Perancangan arsitektur sistem Kalam UMI menghasilkan model perangkat lunak yang terstruktur dan relevan dengan kebutuhan akademik Universitas Muslim Indonesia. Layered Architecture dipilih karena kesederhanaannya, kemudahan pemeliharaan, dan kesesuaian dengan sistem pembelajaran daring yang membutuhkan stabilitas dan skalabilitas moderat.

Penerapan prinsip modularitas, separation of concerns, dan SOLID meningkatkan fleksibilitas dan maintainability. Pembagian modul seperti Learning, Assignment, dan Notification memungkinkan pengembangan paralel tanpa ketergantungan kompleks.

Pola arsitektur seperti API Gateway dan CQRS, serta design pattern Factory, Observer, dan Strategy, memperkuat efisiensi, skalabilitas, dan konsistensi sistem. Kombinasi ini mendukung interaksi pengguna yang tinggi, integrasi layanan kampus, dan pengembangan berkelanjutan.

Dengan pendekatan ini, Kalam UMI diharapkan menjadi sistem e-learning yang adaptif, aman, dan berorientasi pada pengalaman pengguna, serta mampu mendukung transformasi digital kampus secara menyeluruh.

DAFTAR PUSTAKA

[1] Y. Jumaryadi and D. Mahdiana, "Usability Testing of Budi Luhur University E-Learning System Using System Usability Scale (SUS)," *Jurnal Teknik Informatika Universitas Jenderal Soedirman (JUTIF)*, vol. 3, no. 2, pp. 245–252, 2022.

[2] D. Setiawan and S. L. Wicaksono, "Evaluasi Usability Google Classroom Menggunakan System Usability Scale," *Jurnal Ilmu Komputer dan Teknologi Informasi (JIKTI)*, vol. 5, no. 1, pp. 12–18, 2020.

[3] M. A. Rahman, R. T. Putra, and I. P. Sutanto, "Evaluation of E-Learning Usability Based on ISO 25010 with Hofstede's Cultural Dimensions as Moderation," *Applied Information Systems and Management Journal*, vol. 5, no. 2, pp. 65–76, 2025.

[4] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 4th ed. Boston, MA: Addison-Wesley, 2021.

[5] I. Malavolta, "Software Architecture Principles and Practices for System Design," *IEEE Software*, vol. 39, no. 4, pp. 40–49, 2022.

[6] N. Rozanski and E. Woods, *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*, 3rd ed. Boston, MA: Addison-Wesley, 2021.

[7] A. Alvi, S. Ullah, and M. Hussain, "Security and Scalability Considerations in Modern Cloud-Based E-Learning Platforms," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 13, no. 9, pp. 112–120, 2022.

[8] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*, Hoboken, NJ: John Wiley & Sons, 2020.

[9] S. H. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2021.

[10] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Boston, MA: Addison-Wesley, 2020.

[11] M. Richards and N. Ford, *Fundamentals of Software Architecture: An Engineering Approach*, Sebastopol, CA: O'Reilly Media, 2020.

[12] C. Richardson, "Microservice Patterns: API Gateway, CQRS, and Saga," *IEEE Software Architecture Newsletter*, vol. 8, no. 3, pp. 15–22, 2023.