

MAKALAH

ARSITEKTUR PERANGKAT LUNAK

Online Examination berbasis Web

Dosen Pengampuh : Mardiyyah Hasnawi. S.Kom., M.T., MTA.



Disusun Oleh :

M. Rizwan 13020230100

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS MUSLIM INDONESIA
MAKASSAR
2025**

Daftar Isi

BAB 1: PENDAHULUAN	2
1.1 Latar belakang.....	2
1.2 Kebutuhan Bisnis	2
1.2.1 Kebutuhan utama.....	2
1.2.2 Kebutuhan Fungsional	2
1.2.4 Kebutuhan Non-Fungsional	3
1.3 Stakeholders.....	3
BAB 2: ARCHITECTURE THINKING	3
2.1 Business Drivers	3
2.2 Trade-offs Analysis	4
2.3 Tanggung Jawab Keputusan.....	5
2.4 Keseimbangan (Balance)	5
BAB 3: KARAKTERISTIK ARSITEKTUR	6
Karakteristik yang Tidak Diprioritaskan & Alasannya.....	7
BAB 4 — GAYA ARSITEKTUR	7
4.1 Proses Pemilihan Gaya Arsitektur	7
4.2 Diagram Arsitektur.....	8
4.3 Best Practices yang Diterapkan.....	9
4.4 Trade-offs yang Diterima.....	9
BAB 5: PRINSIP ARSITEKTUR	10
5.2 Separation of Concerns	10
5.3 SOLID Principles	11
BAB 6: POLA ARSITEKTUR & DESIGN PATTERN.....	12
6.1 Pola Arsitektur	12
6.2 Design Pattern.....	13
Pattern 1: Factory Pattern (Creational).....	13
Pattern 2: Observer Pattern (Behavioral).....	14
Pattern 3: Strategy Pattern (Behavioral)	15
BAB 7: KESIMPULAN & REKOMENDASI.....	15
Ringkasan Keputusan Arsitektur	15
Metrics Keberhasilan	17
Kesimpulan Akhir.....	17

BAB 1: PENDAHULUAN

1.1 Latar belakang

Proses ujian di kampus saat ini masih banyak dilakukan secara konvensional sehingga membutuhkan waktu, biaya, dan tenaga yang cukup besar, terutama dalam distribusi soal dan koreksi hasil ujian. Selain itu, pengawasan ujian manual masih memiliki risiko terjadinya kecurangan.[1]

Untuk mendukung transformasi menuju Smart Campus, Universitas Muslim Indonesia (UMI) memerlukan sistem ujian berbasis web yang lebih efisien, aman, serta mampu mengelola bank soal dan penilaian secara terpusat. Dengan penerapan Computer-Based Testing (CBT) dan proctoring berbasis kamera, pelaksanaan ujian dapat menjadi lebih terkontrol, cepat, dan mendukung penyediaan hasil evaluasi yang akurat bagi dosen maupun kampus.[1]

1.2 Kebutuhan Bisnis

1.2.1 Kebutuhan utama

1. Efisiensi pelaksanaan ujian tanpa proses manual yang memakan waktu dan biaya.[1]
2. Keamanan dan integritas ujian melalui mekanisme pencegahan kecurangan.[2]
3. Penyediaan hasil evaluasi yang cepat, akurat, dan terdokumentasi sebagai dasar keputusan akademik.[1]

1.2.2 Kebutuhan Fungsional

Sistem ini harus mampu menyediakan fungsi-fungsi sebagai berikut:

1. Memfasilitasi pembuatan, pengelolaan, dan pembaruan bank soal secara terpusat oleh dosen atau admin.
2. Memberikan kemudahan bagi mahasiswa untuk mengikuti ujian berbasis web dengan autentikasi yang valid.[2]
3. Melakukan pemantauan perilaku peserta ujian melalui kamera (proctoring) untuk mendeteksi aktivitas mencurigakan.[2]
4. Melakukan penilaian otomatis untuk soal berbentuk objektif seperti pilihan ganda.
5. Menyediakan laporan hasil ujian yang dapat diakses oleh dosen dan pihak akademik untuk evaluasi lebih lanjut.[1]

1.2.4 Kebutuhan Non-Fungsional

Sistem harus memperhatikan aspek-aspek pendukung berikut:

1. Keamanan, mencakup enkripsi data soal dan autentikasi pengguna.[3]
2. Skalabilitas, agar mampu menangani jumlah peserta dalam skala besar secara bersamaan.[1], [3]
3. Performa, waktu respon sistem harus cepat untuk menjamin kenyamanan pengguna.
4. Ketersediaan, terutama ketika sistem digunakan dalam waktu ujian yang kritis.[1]
5. Kemudahan penggunaan, baik bagi mahasiswa maupun dosen.[4]

1.3 Stakeholders

Para pihak yang terlibat dalam sistem ini meliputi:

- **Mahasiswa**, sebagai pengguna utama yang mengikuti ujian.
- **Dosen atau Pengaji**, yang menyusun soal dan mengakses hasil penilaian.
- **Admin akademik**, yang mengelola data ujian, peserta, dan jadwal.
- **Tim Teknologi Informasi**, yang bertanggung jawab terhadap pemeliharaan dan keamanan sistem.[2]
- **Pimpinan Fakultas atau Kampus**, sebagai pengambil keputusan dari hasil evaluasi akademik.

1.4 Scope Sistem

Sistem ini mencakup fungsi pelaksanaan ujian berbasis web, pengelolaan bank soal, proctoring berbasis kamera, penilaian otomatis soal objektif, serta pelaporan hasil ujian.[1]

Sistem ini tidak mencakup integrasi langsung dengan SIAKAD secara penuh pada tahap awal, dan belum menerapkan penilaian otomatis untuk soal esai berbasis pemrosesan bahasa alami.

BAB 2: ARCHITECTURE THINKING

2.1 Business Drivers

No	Driver	Penjelasan	Dampak ke Arsitektur
1	Integritas dan Keamanan Ujian	Ujian harus terhindar dari kecurangan dan kebocoran soal	Perlu proctoring kamera, autentikasi ketat, enkripsi soal
2	Skalabilitas saat Ujian Massal	Ribuan mahasiswa dapat ujian bersamaan	Arsitektur harus scalable, load balancing, cloud ready

3	Efisiensi Penilaian & Laporan Nilai	Nilai didapat cepat dan akurat untuk dosen dan kampus	Auto-grading, data analytics, central result reporting
---	-------------------------------------	---	--

2.2 Trade-offs Analysis

Trade-off 1 — Monolithic vs Microservices

Aspek	Monolithic	Microservices	Keputusan	Alasan
Deployment	1 aplikasi, mudah	Banyak service, lebih kompleks	✓ Microservices	Ujian harus tetap berjalan tanpa downtime saat update modul
Development	Cepat di awal	Butuh tim lebih matang	⚠ Risiko	Jangka panjang lebih fleksibel dikembangkan per fitur
Scalability	Scale seluruh sistem	Scale sesuai service	✓ Microservices	Server untuk proctoring perlu scale lebih besar dibanding reporting
Cost	Lebih murah di awal	Infrastruktur lebih mahal	⚠ Risiko	Kampus siap investasi bertahap untuk kebutuhan puncak ujian

Keputusan: Microservices karena scalability dan independent deployment sangat krusial untuk ujian massal yang terjadwal ketat.

Trade-off 2 — Proctoring Otomatis vs Manual Monitoring

Aspek	Manual (Tanpa Kamera)	Kamera + Activity Tracking	Keputusan	Alasan
Deteksi Kecurangan	Rendah	Tinggi	✓ Proctoring Kamera	Integritas akademik meningkat
Infrastruktur	Sederhana	Butuh bandwidth & device	⚠ Risiko	Diujicoba dulu untuk kelas besar
Pengawasan	Banyak pengawas	Sistem otomatis	✓	Pengawas cukup memonitor dashboard alert

Keputusan: Proctoring kamera, tetapi bertahap menyesuaikan kesiapan infrastruktur mahasiswa.

Trade-off 3 — Database Terpusat vs Database per Layanan

Aspek	1 DB untuk semua	DB per service	Keputusan	Alasan
Kemandirian Service	Coupling tinggi	Mandiri	<input checked="" type="checkbox"/> DB per Service	Modul ujian dan proctoring dapat bekerja walaupun modul nilai bermasalah
Konsistensi Data	Konsisten kuat	Eventual Consistency	<input type="checkbox"/> Risiko	Risiko <i>eventual consistency</i> adalah <i>trade-off</i> umum dalam Microservices. Data kritis dapat diatasi dengan pola seperti <i>Event Sourcing</i> .[3]
Query Analisis	Lebih mudah	Perlu integrasi	<input type="checkbox"/>	Laporan dapat dibuat dengan mencatat <i>events</i> menggunakan pola <i>Event Sourcing</i> , yang mencatat semua perubahan <i>state</i> .[3]

Keputusan: Database per service, agar sistem tetap berjalan meskipun satu modul bermasalah.

2.3 Tanggung Jawab Keputusan

Keputusan	Architect	Developer
Pemilihan Monolithic vs Microservices	<input checked="" type="checkbox"/>	<input type="checkbox"/> X
Desain integrasi modul	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Implementasi fitur CBT	<input type="checkbox"/> X	<input checked="" type="checkbox"/>
Kebijakan keamanan (OAuth, data encryption)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Pemilihan teknologi proctoring	<input checked="" type="checkbox"/>	<input type="checkbox"/> X
Optimasi performa database	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

2.4 Keseimbangan (Balance)

♦ Performa vs Biaya

- Proctoring kamera → butuh server & bandwidth lebih besar[3]
- Balance: Only-trigger recording (hanya capture saat suspicious event)

◆ Kompleksitas vs Maintainability

- Microservices kompleks saat awal
- Balance: Mulai dengan 3 core services:
 1. Ujian & Bank Soal
 2. Proctoring
 3. Penilaian & Laporan

◆ Time-to-Market vs Quality

- Rilis cepat berisiko bug saat ujian
- Balance: Pilot testing pada beberapa program studi dulu sebelum full release. *Quality* harus didorong dengan praktik DevSecOps dan pengujian yang kuat (*SAST/DAST*) sejak fase desain.[2]

BAB 3: KARAKTERISTIK ARSITEKTUR

Tabel karakteristik :

Karakteristik	Kategori	Prioritas	Justifikasi	Metrik Target
Availability	Operational	CRITICAL	Ujian memiliki jadwal ketat. Jika sistem down saat ujian berlangsung, dapat mengganggu kelulusan dan jadwal akademik.[1]	99.9% uptime (maks 43 menit downtime/bulan) saat periode ujian[1]
Security	Cross-cutting	CRITICAL	Soal ujian dan data nilai sangat sensitif. Kebocoran soal dapat merusak integritas akademik.[1]	Zero soal bocor, autentikasi berlapis, enkripsi data ujian[1]
Scalability	Operational	HIGH	Ribuan mahasiswa bisa ujian serentak. Sistem harus tetap responsif saat peak load ujian massal.[1]	Support 10,000+ concurrent exam participants[1]
Performance	Operational	HIGH	Proses loading soal dan submit jawaban harus cepat agar ujian tidak terganggu.[1]	Response API < 500 ms, submit < 2 detik[1]
Usability	Structural	MEDIUM	Mahasiswa dan dosen membutuhkan antarmuka yang mudah dipahami tanpa pelatihan teknis khusus.[1]	95% user success rate tanpa bantuan[1]

Karakteristik yang Tidak Diprioritaskan & Alasannya

- **Portability**
Sistem dirancang untuk berjalan stabil di satu platform cloud yang sudah ditentukan, tidak perlu sering dipindahkan.[1]
- **Customizability**
Fokus awal pada standar ujian kampus, fitur lanjutan dapat menyusul setelah sistem stabil.[1]

BAB 4 — GAYA ARSITEKTUR

4.1 Proses Pemilihan Gaya Arsitektur

Langkah 1 — Eliminasi Gaya yang Kurang Cocok

Gaya Arsitektur	Cocok?	Alasan
Layered (N-Tier)	✗	Skalabilitas per modul rendah; saat ujian massal bisa bottleneck. Kelemahan arsitektur monolitik yang sering menggunakan <i>layered</i> adalah kesulitan dalam penskalaan individu komponen[4]
SOA	✗	Terlalu heavy untuk scope kampus; butuh ESB & cost tinggi[4]
Microservices	✓	Setiap modul (CBT, proctoring, nilai) dapat scale sesuai kebutuhan. Arsitektur ini unggul dalam scalability dan independent deployment.[4]
Event-Driven	✓	Proctoring dan logging ujian butuh real-time event streaming. EDA sangat berguna untuk aplikasi yang membutuhkan respons terhadap peristiwa secara <i>real-time</i> dan berskala besar.[4]
Serverless	✗	Tidak cocok untuk core session exam yang long-running.

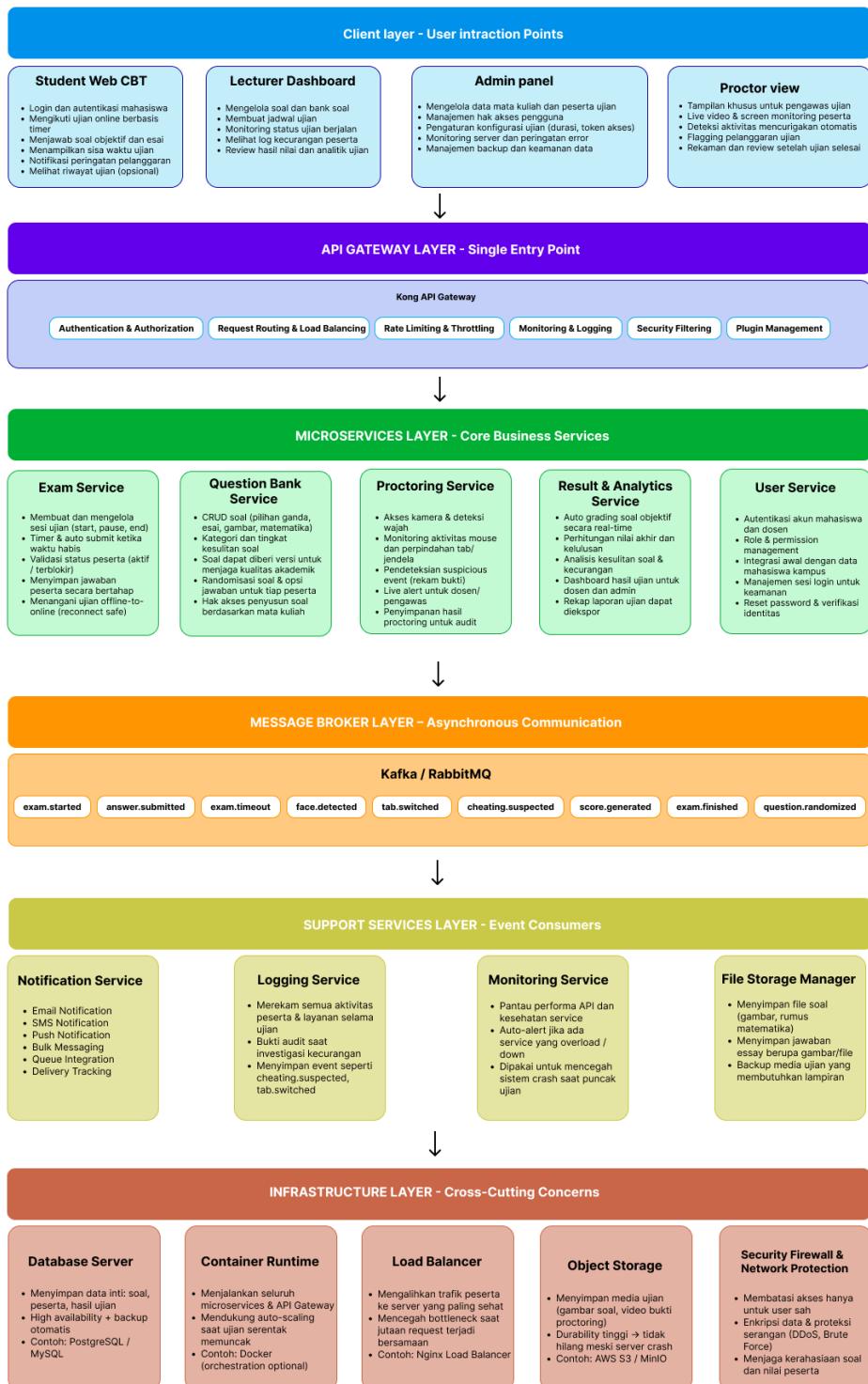
Langkah 2 — Evaluasi Pilihan Tersisa

Microservices vs Event-Driven

- Bukan "VS" → Bisa kombinasi!
- Microservices → struktur modul terpisah & scalable evcdas
- Event-Driven → proctoring alert & logging real-time

Keputusan Final : Hybrid → Microservices + Event-Driven Architecture untuk mendukung scalability + real-time proctoring

4.2 Diagram Arsitektur



Penjelasan Komponen

1. **API Gateway** → Single entry point; autentikasi & traffic routing
2. **Exam Service** → Handle pelaksanaan ujian real-time
3. **Question Bank Service** → CRUD bank soal terpusat
4. **Proctoring Service** → Kamera, activity tracking, fraud detection
5. **Result & Analytics Service** → Auto-grading, score processing, reporting
6. **User Service** → Manajemen akun mahasiswa & dosen
7. **Message Broker (Kafka/RabbitMQ)** → Event syncing antar service
8. **Notification Service** → Reminder & info hasil
9. **Monitoring Service** → Track status ujian & alert error

4.3 Best Practices yang Diterapkan

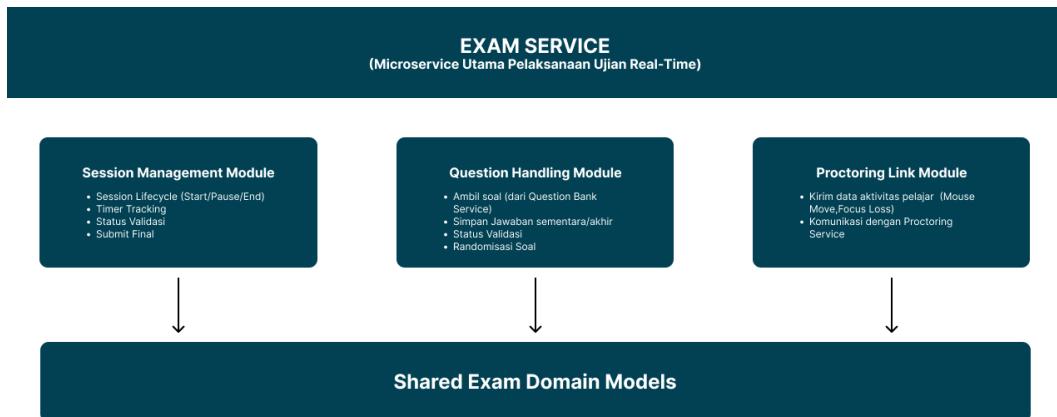
Best Practice	Implementasi	Alasan
Database per Service	CBT DB, Proctoring DB, Analytics DB	Loose coupling , setiap modul independen, mendukung <i>Decentralized Data Management</i> .[4]
API Gateway	Authentication, logging, rate limit	Security & centralized access . Bertindak sebagai <i>security checkpoint</i> dan <i>broker</i> koneksi.[4]
Circuit Breaker	Jika proctoring gagal → ujian tetap berjalan	Mencegah <i>cascade failure</i> . Pola ini meningkatkan resiliency sistem[4]
Event Sourcing (partial)	Simpan event submit jawaban & suspicious activity	Audit trail & replay jika error . Menyediakan catatan lengkap perubahan <i>state</i> .[4]
Centralized Logging	Kamera log & activity log	ebugging saat sengketa ujian. Log (<i>traces, metrics</i>) sangat penting untuk observability di sistem terdistribusi.[4]
Health Check Endpoint	Auto failover saat service down	Menjaga sistem tetap berjalan saat ujian. Ini adalah bagian dari <i>Microservice Resiliency and Availability Overlay</i> [4]

4.4 Trade-offs yang Diterima

Trade-off	Konsekuensi	Mitigasi
Eventual Consistency	Nilai & status ujian tidak selalu real-time	Acceptable delay < 5 detik
Network Latency	Inter-service call lebih lambat dari in-process	Minimize sync calls + caching
Operational Complexity	Butuh DevOps untuk manage service banyak	Managed cloud service + pelatihan
Distributed Transaction	ACID sulit antar service	Saga Pattern untuk konsistensi proses

BAB 5: PRINSIP ARSITEKTUR

Diagram Modular :

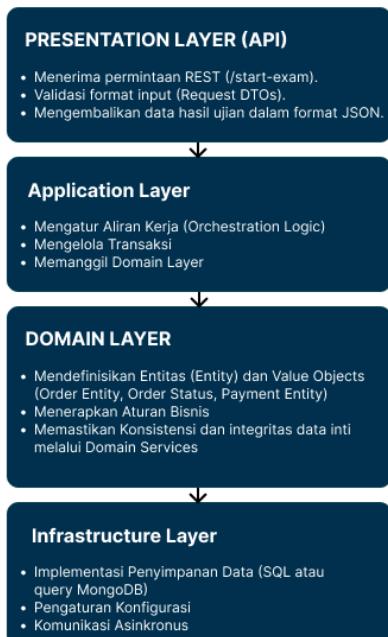


Prinsip Modularitas:

- Low Coupling : Antar service berkomunikasi melalui **Message Broker (RabbitMQ)** sehingga ketergantungan langsung berkurang
- High Cohesion : Setiap service mengelola **satu domain**, contoh: Exam Service hanya menangani sesi ujian[5]

5.2 Separation of Concerns

Contoh di Exam Service :



5.3 SOLID Principles

Contoh 1: Single Responsibility Principle (SRP)

Kasus:

Dalam sistem CBT, ExamService awalnya menangani pembuatan ujian, penilaian jawaban, dan pengiriman notifikasi. Hal ini membuat kode sulit diuji dan diperbarui.

Implementasi :

```
public class ExamService {  
    public void createExam(Exam exam) { // Hanya membuat ujian}  
}  
  
public class EvaluationService {  
    public void evaluateAnswers(List<Answer> answers) { // Hanya menghitung nilai}  
}  
  
public class NotificationService {  
    public void sendNotification(String message) { // Hanya mengirim notifikasi}  
}
```

Benefit:

- Setiap service fokus pada satu tugas.
- Perubahan di satu fungsi tidak memengaruhi fungsi lain.
- Kode lebih mudah diuji dan dikelola.

Contoh 2 — Dependency Inversion Principle (DIP)

Kasus:

Saat sistem CBT ingin menambah jenis ujian baru (AdaptiveExam), developer tidak boleh mengubah kode yang sudah stabil di ExamFactory.

Implementasi :

```
public abstract class Exam {  
    abstract void startExam();
```

```

}

public class AdaptiveExam extends Exam {

    void startExam() {System.out.println("Starting adaptive exam..."); }

}

public class ExamFactory {

    public static Exam createExam(String type) {

        switch(type.toLowerCase()) {

            case "regular": return new RegularExam();

            case "adaptive": return new AdaptiveExam();

            default: throw new IllegalArgumentException();

        }
    }
}

```

Benefit:

- Penambahan fitur baru tanpa mengganggu kode lama.
- Risiko bug lebih kecil.
- Sistem lebih fleksibel dan mudah dikembangkan.

BAB 6: POLA ARSITEKTUR & DESIGN PATTERN

6.1 Pola Arsitektur

Pola 1 : Backend for Frontend (BFF)

Masalah	Solusi	Implementasi	Benefit
UI peserta dan proctor memiliki kebutuhan data serta hak akses yang berbeda. Jika langsung memanggil microservice, logika autentikasi	Buat backend khusus untuk setiap jenis client (BFF peserta dan BFF proctor). BFF bertugas menangani autentikasi, agregasi data, dan	- BFF Peserta: handle start exam, submit jawaban, timer. - BFF Proctor: handle live monitoring, log aktivitas, cheating alert. - Gunakan Node.js atau Laravel	- Response lebih ringan dan efisien untuk tiap client. - Logika UI lebih sederhana. - Security lebih terkontrol per role. - Perubahan UI tidak

dan transformasi data menjadi duplikat di client.	menyesuaikan payload ke kebutuhan UI.[5]	sebagai lapisan BFF sebelum API Gateway.	berdampak langsung ke microservice.[5]
---	--	--	--

Pola 2 : CQRS (Command Query Responsibility Segregation)

Masalah	Solusi	Implementasi	Benefit
Aplikasi CBT memiliki beban tulis tinggi saat peserta mengirim jawaban, dan beban baca tinggi saat proctor melihat dashboard live. Jika satu database digunakan untuk keduanya, performa menurun.[5]	Pisahkan operasi tulis (Command) dan baca (Query). Command fokus ke transaksi cepat, Query ke laporan dan analytics. Sinkronisasi data dilakukan melalui event stream.[5]	- Command: menyimpan jawaban ke DB utama (PostgreSQL). - Query: dashboard membaca data dari cache/read replica (Redis/ElasticSearch). - Event broker (RabbitMQ/Kafka) mengirim event answersubmitted.[5]	- Meningkatkan performa baca & tulis. - Meningkatkan skalabilitas dashboard real-time. - Struktur data analytics dapat dioptimalkan tanpa mengganggu proses ujian.[5]

Kedua pola ini saling melengkapi: **Microservices sebagai struktur, Event-driven sebagai komunikasi utama.[5]**

6.2 Design Pattern

Pattern 1: Factory Pattern (Creational)

1. Nama & Kategori:

Factory Pattern — *Creational Pattern*

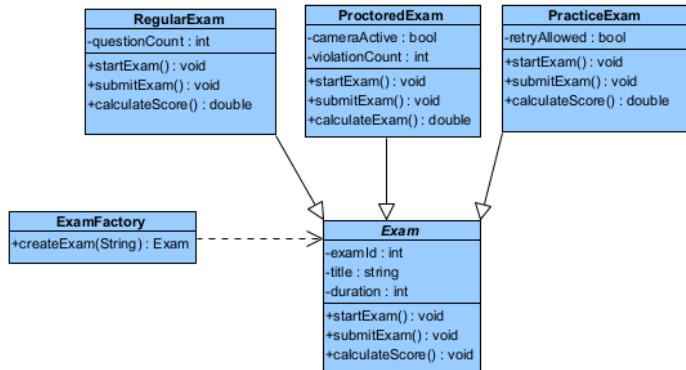
2. Masalah yang Diselesaikan:

Sistem CBT memiliki berbagai tipe ujian: Regular, Proctored, dan Practice.

Setiap tipe membutuhkan logika berbeda (misalnya ProctoredExam memakai kamera).

Tanpa Factory, developer harus membuat if-else panjang untuk setiap tipe.

3. Solusi (Diagram UML) :



4. Benefit:

- Menghindari kode if-else yang panjang.[5]
- Memudahkan penambahan tipe ujian baru.[5]
- Pola ini meningkatkan *modularitas* dan *reusability*.[5]

Pattern 2: Observer Pattern (Behavioral)

1. Nama & Kategori:

Observer Pattern — *Behavioral Pattern*

2. Masalah yang Diselesaikan:

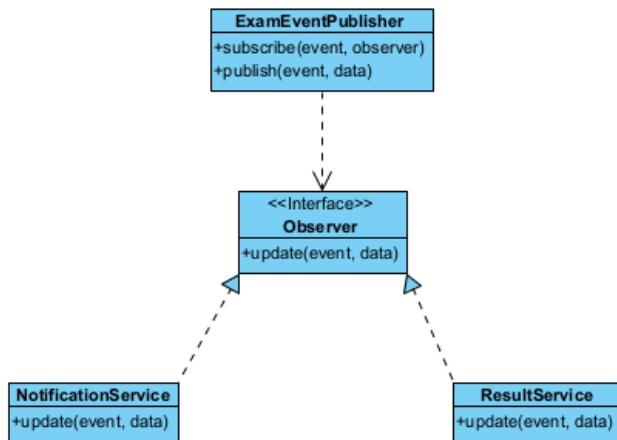
Ketika event exam.started terjadi, beberapa modul (Notification, Result, Analytics) perlu merespons.

Tanpa observer, semua harus dipanggil manual satu per satu.

3. Solusi (Diagram UML):

Setiap listener berlangganan event dari ExamEventPublisher.

Ketika event dikirim, semua listener otomatis dijalankan.



4. Benefit:

- Mengurangi *tight coupling* antar komponen.[5]
- Memudahkan ekspansi sistem event-driven.[5]
- Menambah subscriber baru tanpa ubah kode lama.[5]

Pattern 3: Strategy Pattern (Behavioral)

1. Nama & Kategori:

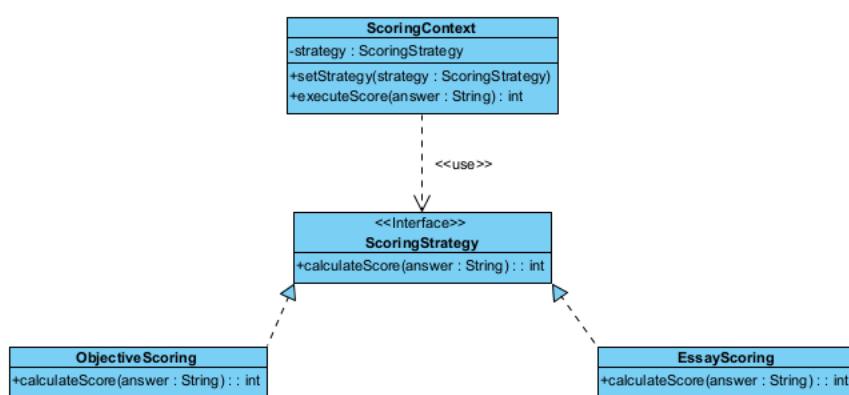
Strategy Pattern — *Behavioral Pattern*

2. Masalah yang Diselesaikan:

Tiap ujian bisa memakai algoritma penilaian berbeda (Objective, Essay, Adaptive). Tanpa Strategy Pattern, semua logika bercampur di satu class.

3. Solusi (Diagram UML)

Gunakan interface ScoringStrategy, dan service ScoringContext memilih algoritma yang tepat.



4. Benefit:

- Menambah algoritma baru tanpa ubah class utama.[5]
- Kode lebih fleksibel dan mudah diuji.[5]
- Mendukung prinsip **Open/Closed** dari SOLID.[5]

BAB 7: KESIMPULAN & REKOMENDASI

Ringkasan Keputusan Arsitektur

Perancangan arsitektur sistem **Computer Based Test (CBT)** ini dilakukan dengan pendekatan **microservices architecture** yang dikombinasikan dengan **event-driven communication**.[2]

Pemilihan gaya arsitektur ini didasarkan pada kebutuhan skalabilitas, modularitas, dan fleksibilitas dalam mengelola fitur-fitur utama seperti **Exam Management**, **Question Bank**, **Proctoring**, **Result Processing**, dan **Notification**.[2]

Komponen pendukung seperti **API Gateway (Kong)**, **Message Broker (RabbitMQ)**, dan **Redis Cache** digunakan untuk mendukung komunikasi antar layanan dan meningkatkan performa sistem.[2]

Rekomendasi Implementasi (Fase)

Implementasi sistem CBT ini sebaiknya dilakukan secara bertahap melalui tiga fase utama:

Fase	Deskripsi	Fokus Utama
Fase 1	<i>Core Functionality Implementation</i>	Pengembangan modul dasar: registrasi, pembuatan ujian, dan evaluasi hasil.
Fase 2	<i>Integration & Automation</i>	Integrasi Proctoring Service (face detection, cheating detection) dan event-driven communication.[2]
Fase 3	<i>Optimization & Scaling</i>	Penambahan caching (Redis), auto-scaling microservices, dan analitik performa pengguna.[2]

Pendekatan bertahap ini memastikan sistem dapat diuji dan dievaluasi di setiap tahap tanpa mengganggu keseluruhan arsitektur.[2]

Risiko dan Mitigasi

Risiko	Dampak	Mitigasi
Kegagalan komunikasi antar service	Data tidak sinkron antar modul	Gunakan message queue (RabbitMQ) untuk memastikan reliabilitas event.[2]
Beban server tinggi saat ujian serentak	Penurunan performa atau downtime	Terapkan load balancing dan Redis cache untuk data yang sering diakses.[2]
Kesalahan deteksi pada Proctoring Service	Salah deteksi kecurangan	Gunakan model AI yang terlatih dengan dataset lokal dan tambahkan manual review.[2]
Ketergantungan antar microservices	Kompleksitas integrasi meningkat	Gunakan API Gateway dan service registry untuk memudahkan orkestrasi.[2]

Metrics Keberhasilan

Keberhasilan implementasi sistem ini dapat diukur melalui beberapa metrik berikut:

Aspek	Metrik	Target
Availability	Uptime server	$\geq 99.9\%$
Performance	Waktu respons API	≤ 500 ms
Scalability	Jumlah user bersamaan	≥ 10.000 concurrent users
Security	Jumlah insiden keamanan	0 data breach
User Experience	Tingkat kepuasan pengguna	$\geq 90\%$ dari hasil survei

Kesimpulan Akhir

Dengan arsitektur yang modular, terdistribusi, dan event-driven, sistem Online Examination (CBT) ini mampu menyediakan solusi ujian daring yang handal, aman, dan mudah dikembangkan. Pendekatan microservices memastikan setiap modul dapat ditingkatkan secara independen tanpa memengaruhi keseluruhan sistem.[2] Jika implementasi dilakukan sesuai fase yang direkomendasikan, sistem ini siap dioperasikan secara skala besar dengan performa tinggi dan keamanan terjamin.[2]

REFERENSI

- [1] V. B. Ramu, "Performance Impact of Microservices Architecture," *The Review of Contemporary Scientific and Academic Studies*, vol. 3, no. 6, Jun. 2023, doi: 10.55454/rccas.3.06.2023.010.
- [2] "Microservices Architecture Pattern," 2021.
- [3] Ashwin Chavan, "Exploring event-driven architecture in microservices- patterns, pitfalls and best practices," *International Journal of Science and Research Archive*, vol. 4, no. 1, pp. 229–249, Dec. 2021, doi: 10.30574/ijrsa.2021.4.1.0166.
- [4] R. Cabral, M. Kalinowski, M. T. Baldassarre, H. Villamizar, T. Escovedo, and H. Lopes, "Investigating the impact of solid design principles on machine learning code understanding," in *Proceedings - 2024 IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI, CAIN 2024*, Association for Computing Machinery, Inc, Apr. 2024, pp. 7–17. doi: 10.1145/3644815.3644957.
- [5] V. Velepucha and P. Flores, "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," *IEEE Access*, vol. 11, pp. 88339–88358, 2023, doi: 10.1109/ACCESS.2023.3305687.