Rangkuman Kisi-kisi UAS Interoperabilitas

1. Konsep Interoperabilitas

Interoperabilitas adalah kemampuan dua atau lebih sistem yang berbeda (berbeda platform, bahasa pemrograman, atau arsitektur) untuk saling berkomunikasi, bertukar data, dan menggunakan informasi tersebut secara bermakna tanpa perlu mengetahui detail internal masing-masing sistem. Interoperabilitas memungkinkan sistem yang heterogen tetap dapat bekerja sama secara efektif, meskipun dibangun dengan teknologi, bahasa, dan lingkungan yang berbeda. Dalam sistem terdistribusi modern dan arsitektur microservices, interoperabilitas menjadi sangat krusial karena aplikasi jarang berdiri sendiri, melainkan saling terhubung melalui jaringan, API, dan layanan terpisah yang harus berkoordinasi secara konsisten dan andal.

Interoperabilitas mencakup beberapa aspek utama, yaitu: - **Teknis**: bagaimana data dikirim dan diterima (protokol, format data). - **Semantik**: kesamaan makna data yang dipertukarkan. - **Organisasional**: kesesuaian proses dan aturan antar sistem.

Contoh implementasi interoperabilitas adalah penggunaan REST API, JSON-RPC, XML-RPC, gRPC, dan WebSocket untuk menghubungkan sistem yang heterogen.

---

2. RESTful API

RESTful API adalah layanan web yang menerapkan prinsip **REST (Representational State Transfer)** dengan memanfaatkan protokol HTTP. REST sendiri adalah gaya arsitektur (architectural style), sedangkan RESTful API adalah implementasi nyata dari prinsip-prinsip REST tersebut dalam bentuk API. Dengan kata lain, **REST adalah konsep**, sementara **RESTful API adalah penerapannya**. REST digunakan untuk memungkinkan komunikasi antar sistem secara sederhana, scalable, dan stateless, dengan cara memperlakukan data sebagai resource yang diakses melalui endpoint URL.

Karakteristik utama RESTful API: - Menggunakan metode HTTP: **GET, POST, PUT, DELETE** - Bersifat **stateless** (server tidak menyimpan state klien) - Resource direpresentasikan dalam format **JSON atau XML** - Menggunakan endpoint berbasis URL

RESTful API banyak digunakan karena mudah diimplementasikan, mudah diuji, dan sangat cocok untuk interoperabilitas antar sistem berbasis web maupun mobile.

REST vs RESTful API

REST adalah **gaya arsitektur (architectural style)** yang berisi prinsip-prinsip desain sistem terdistribusi, sedangkan RESTful API adalah **implementasi konkret** dari prinsip REST dalam bentuk API yang dapat digunakan oleh aplikasi.

Dengan kata lain: - **REST** → konsep, aturan, dan prinsip desain - **RESTful API** → penerapan REST menggunakan HTTP dan endpoint API

REST tidak terikat teknologi tertentu, sedangkan RESTful API biasanya menggunakan HTTP, JSON, dan URL.

Kapan Menggunakan REST / RESTful API

RESTful API paling tepat digunakan ketika: - Sistem menggunakan model **client–server** - Aplikasi bersifat **web atau mobile** - Operasi data dominan berupa **CRUD (Create, Read, Update, Delete)** - Dibutuhkan **skalabilitas tinggi** dan komunikasi stateless - Sistem perlu mudah diintegrasikan lintas platform

RESTful API kurang optimal jika aplikasi membutuhkan komunikasi real-time intens atau latensi sangat rendah, karena skenario tersebut lebih cocok menggunakan WebSocket atau gRPC.

Best Practice RESTful API

Beberapa praktik terbaik (best practice) dalam merancang RESTful API antara lain:

1. **Resource-Oriented Design** Endpoint API harus merepresentasikan resource (kata benda), bukan aksi. Contoh:
2. GET /users
3. POST /users
4. GET /users/{id}
5. **Penggunaan HTTP Method yang Tepat** GET untuk membaca data, POST untuk membuat data, PUT/PATCH untuk memperbarui data, dan DELETE untuk menghapus data.
6. **Stateless Communication** Setiap request harus berdiri sendiri tanpa bergantung pada state sebelumnya. Autentikasi biasanya menggunakan token seperti JWT atau OAuth.
7. **Penggunaan HTTP Status Code** Status code digunakan untuk menunjukkan hasil request, seperti 200 (OK), 201 (Created), 400 (Bad Request), 401 (Unauthorized), dan 404 (Not Found).
8. **Format Data Konsisten** Umumnya menggunakan format JSON agar mudah dibaca dan diproses oleh berbagai platform.

Teknologi yang Menggunakan RESTful API

RESTful API dapat digunakan oleh berbagai teknologi, di antaranya:

Backend: - Node.js (Express, NestJS) - Laravel - Spring Boot - Django / Flask - ASP.NET Core

Frontend: - React / Next.js - Vue - Angular - Aplikasi mobile (Android, iOS, Flutter)

Tools Pendukung: - OpenAPI / Swagger - Postman - API Gateway (Kong, AWS API Gateway) - Sistem autentikasi (JWT, OAuth 2.0)

---

3. Middleware Cloud-Native

Middleware cloud-native adalah perangkat lunak perantara yang memfasilitasi komunikasi, integrasi, dan koordinasi antar layanan dalam arsitektur cloud-native dan microservices. Middleware ini berperan

sebagai **lapisan abstraksi** yang memisahkan logika bisnis aplikasi dengan mekanisme komunikasi dan integrasi, sehingga setiap service tidak saling bergantung secara langsung (loose coupling).

Middleware cloud-native **dibutuhkan ketika** sistem terdiri dari banyak service yang saling berinteraksi, terutama pada arsitektur microservices, di mana komunikasi langsung antar service akan meningkatkan kompleksitas dan risiko kegagalan. Dengan middleware, service dapat berkomunikasi melalui mekanisme yang terstandarisasi seperti messaging, API gateway, atau event streaming.

Middleware juga sangat penting saat sistem memerlukan **skalabilitas, keandalan, dan ketahanan (resilience)**. Misalnya, message broker seperti Kafka memungkinkan komunikasi asynchronous sehingga kegagalan satu service tidak langsung mematikan service lain. Cache seperti Redis digunakan ketika aplikasi membutuhkan respon cepat tanpa selalu mengakses database.

Secara praktis, middleware cloud-native digunakan untuk kebutuhan seperti routing request, load balancing, autentikasi, manajemen session/token, komunikasi asynchronous, serta integrasi data antar service. Dengan adanya middleware, setiap layanan dapat dikembangkan, diuji, di-deploy, dan diskalakan secara independen tanpa harus mengetahui detail implementasi internal layanan lain, sehingga sistem menjadi lebih modular, scalable, dan mudah dipelihara.

Ciri utama middleware cloud-native: - Ringan dan stateless - Mendukung container dan Kubernetes - Mendukung skalabilitas horizontal - Memiliki observability (logging, tracing, metrics)

Contoh middleware cloud-native meliputi API Gateway, message broker (Kafka), cache (Redis), dan sistem autentikasi. Middleware ini membantu aplikasi berkomunikasi tanpa harus memahami detail teknis layanan lain.

---

4. Service Mesh

Service mesh adalah **lapisan infrastruktur jaringan** yang mengatur komunikasi **antar layanan (service-to-service)** pada arsitektur microservices, terutama di Kubernetes. Tujuan utamanya bukan menambah fitur bisnis, tetapi membuat komunikasi antar service menjadi **lebih aman, terkontrol, mudah dipantau, dan lebih tahan gangguan**—tanpa harus menulis kode tambahan di setiap service.

Kenapa Service Mesh Dibutuhkan

Pada microservices, jumlah service bisa puluhan bahkan ratusan. Jika setiap service mengatur sendiri hal-hal seperti TLS, retry, timeout, tracing, dan policy akses, maka: - konfigurasi jadi tidak konsisten - debugging makin sulit - perubahan policy harus dilakukan di banyak tempat - risiko keamanan meningkat

Service mesh menyelesaikan ini dengan **memindahkan tanggung jawab komunikasi** ke layer jaringan.

Cara Kerja (Gambaran Mudah)

Service mesh biasanya memakai pola **sidecar proxy**. Artinya, setiap pod/service ditemani proxy (misalnya Envoy). Semua traffic masuk/keluar service melewati proxy tersebut.

Alur sederhana: 1. Service A ingin memanggil Service B 2. Request dari A **tidak langsung** ke B, tetapi ke sidecar proxy A 3. Proxy A menerapkan policy (mTLS, routing, retry, rate limit, dsb) 4. Proxy A meneruskan ke proxy B 5. Proxy B meneruskan ke Service B, lalu response kembali lewat jalur yang sama

Komponen Utama

Service mesh bekerja dengan dua komponen utama: - **Data Plane**: sidecar proxy (misalnya Envoy) yang menangani traffic antar service, melakukan enkripsi, retry, load balancing, dan telemetry - **Control Plane**: pusat pengaturan yang mendistribusikan konfigurasi/policy ke semua proxy (contoh pada Istio: `istiod`)

Fitur Utama (Yang Sering Muncul di Soal)

1. **Security (mTLS)**
2. komunikasi antar service otomatis terenkripsi
3. identitas service dapat diverifikasi (service identity)
4. cocok untuk lingkungan zero-trust
5. **Traffic Management**
6. routing cerdas (misal canary release: 90% v1, 10% v2)
7. load balancing
8. mirroring traffic (untuk testing)
9. **Resilience / Fault Tolerance**
10. retry otomatis dengan batas
11. timeout standar
12. circuit breaker untuk mencegah cascading failure
13. fault injection (simulasi delay/error) untuk uji ketahanan
14. **Observability (Monitoring, Logging, Tracing)**
15. metrics (Prometheus), dashboard (Grafana)
16. tracing (Jaeger)
17. membantu mencari bottleneck dan error antar service

Kapan Service Mesh Dipakai (Kriteria Praktis)

Service mesh biasanya **dibutuhkan** ketika: - microservices sudah banyak dan traffic antar service kompleks - butuh **mTLS internal** tanpa menambah kode di tiap service - perlu observability end-to-end (trace request lintas service) - butuh traffic routing canggih (canary, blue-green, A/B) di level jaringan - sering ada masalah reliability (timeout, retry, cascading failure)

Service mesh **biasanya belum perlu** ketika: - sistem masih monolith atau microservices sedikit - traffic antar service sederhana - overhead proxy dirasa terlalu berat untuk skala kecil

Best Practice Service Mesh

- Mulai dari **observability dulu** (monitoring + tracing), baru aktifkan policy yang kompleks
- Terapkan **mTLS bertahap** (per namespace/service) agar tidak mengganggu sistem
- Atur **timeout dan retry** dengan benar (retry tanpa batas bisa memperparah beban)
- Pastikan resource cukup karena sidecar proxy menambah overhead CPU/RAM
- Dokumentasikan policy routing dan security agar tim paham perubahan jaringan

Contoh Implementasi

Contoh implementasi service mesh adalah **Istio** dan **Linkerd**. Istio umumnya dipilih saat butuh fitur lengkap (mTLS, routing kompleks, integrasi telemetry), sedangkan Linkerd sering dipilih karena lebih ringan dan sederhana.

---

5. gRPC

gRPC (gRPC Remote Procedure Call) adalah framework komunikasi open-source yang dikembangkan oleh Google untuk komunikasi antar layanan dengan performa tinggi dalam sistem terdistribusi dan arsitektur microservices. gRPC memungkinkan sebuah service memanggil fungsi di service lain seolah-olah fungsi tersebut dijalankan secara lokal, sehingga komunikasi antar layanan menjadi lebih terstruktur, efisien, dan mudah dikelola dibandingkan mekanisme berbasis HTTP tradisional.

gRPC menggunakan: - Protokol **HTTP/2** - Format data **Protocol Buffers (Protobuf)**

Keunggulan gRPC: - Lebih cepat dan efisien dibanding REST karena menggunakan HTTP/2 dan format biner - Overhead jaringan lebih kecil sehingga cocok untuk sistem dengan traffic tinggi - Mendukung komunikasi streaming (real-time data flow) - Kontrak API jelas melalui Protobuf sehingga mengurangi miskomunikasi antar service - Cocok untuk microservices skala besar dan komunikasi internal

Kapan gRPC Dibutuhkan

gRPC **dibutuhkan** ketika: - komunikasi antar service sangat sering dan sensitif terhadap latensi - sistem terdiri dari banyak microservices internal - dibutuhkan performa tinggi dan efisiensi bandwidth - perlu komunikasi streaming (misalnya monitoring, data sensor, atau live update antar service) - tim ingin kontrak API yang ketat (strongly typed)

gRPC **kurang cocok** ketika: - API bersifat publik dan diakses langsung oleh browser - sistem sederhana dengan kebutuhan CRUD biasa - developer membutuhkan fleksibilitas tanpa Protobuf

Jenis komunikasi gRPC: - Unary RPC → satu request satu response - Server streaming → satu request banyak response - Client streaming → banyak request satu response - Bidirectional streaming → client dan server saling mengirim data secara real-time

---

6. WebSocket

WebSocket adalah protokol komunikasi **full-duplex** yang memungkinkan klien dan server saling mengirim dan menerima data **secara real-time** melalui **satu koneksi yang tetap terbuka**. Berbeda dengan HTTP biasa yang berbasis request–response (klien harus meminta dulu baru server

merespons), WebSocket membuat koneksi persisten sehingga server dapat **mendorong (push)** data kapan saja tanpa menunggu request baru. Umumnya koneksi dimulai dengan **HTTP handshake** (upgrade dari HTTP ke WebSocket), lalu setelah berhasil, komunikasi berjalan dua arah dengan overhead yang jauh lebih kecil dibanding polling/long-polling, sehingga latensi lebih rendah dan lebih efisien untuk aplikasi yang membutuhkan update cepat dan berkelanjutan.

Karakteristik WebSocket: - Full-duplex (dua arah) - Low latency - Stateful connection - Berbasis TCP

WebSocket banyak digunakan pada aplikasi real-time seperti chat, dashboard monitoring, game online, dan sistem IoT.

Kapan WebSocket Dibutuhkan

WebSocket **dibutuhkan** ketika aplikasi memerlukan: - update data secara real-time tanpa polling berulang - komunikasi dua arah yang terus-menerus - latensi sangat rendah - server perlu mengirim data ke klien secara aktif (server push)

Contoh kasus: - aplikasi chat dan notifikasi - dashboard monitoring (CPU, traffic, sensor) - game multiplayer online - aplikasi kolaborasi real-time

WebSocket **kurang cocok** ketika: - aplikasi hanya membutuhkan request–response sederhana - data jarang berubah - beban server perlu dijaga tetap ringan tanpa koneksi persisten

Best Practice WebSocket

- Gunakan WebSocket hanya untuk data yang benar-benar real-time
- Tetap gunakan REST API untuk operasi CRUD
- Kelola koneksi dengan baik (disconnect, heartbeat, timeout)
- Amankan koneksi dengan autentikasi (token saat handshake)
- Batasi jumlah koneksi aktif agar tidak membebani server

---

7. Kaitan dengan Tugas PBL

Dalam tugas PBL (Project-Based Learning), konsep interoperabilitas diwujudkan melalui integrasi berbagai layanan dan teknologi.

Contoh penerapan: - REST API atau gRPC untuk komunikasi antar modul aplikasi - Middleware (API Gateway, Redis, Kafka) untuk integrasi dan manajemen data - Service mesh untuk pengelolaan komunikasi microservices di Kubernetes - WebSocket untuk fitur real-time (notifikasi, monitoring)

Dengan menerapkan konsep-konsep ini, sistem PBL menjadi lebih scalable, modular, dan mudah dikembangkan.

---

8. Perbandingan REST API vs gRPC vs WebSocket vs Service Mesh

Bagian ini bertujuan untuk membantu memahami **perbedaan peran, fungsi, dan kapan masing-masing teknologi digunakan**, karena topik ini sering menjadi soal jebakan pada UAS.

Tabel Perbandingan

Aspek	REST API	gRPC	WebSocket	Service Mesh
Jenis	Gaya arsitektur API	Framework RPC	Protokol komunikasi	Infrastruktur jaringan
Pola Komunikasi	Request–Response	RPC + Streaming	Full-duplex real-time	Service-to-service
Protokol	HTTP/1.1	HTTP/2	TCP (via HTTP Upgrade)	Berbasis jaringan
Format Data	JSON / XML	Protobuf (biner)	Bebas (JSON, binary)	Tidak mengatur format
Latensi	Sedang	Rendah	Sangat rendah	Bergantung traffic
State	Stateless	Stateless	Stateful	Stateless
Skalabilitas	Tinggi	Sangat tinggi	Terbatas koneksi	Sangat tinggi
Digunakan Oleh	Client ↔ Server	Service ↔ Service	Client ↔ Server	Antar service internal
Kompleksitas	Rendah	Menengah	Menengah	Tinggi

Kapan Menggunakan Masing-Masing

- **REST API** digunakan ketika aplikasi membutuhkan integrasi lintas platform, operasi CRUD, dan API publik.
- **gRPC** digunakan untuk komunikasi internal microservices yang membutuhkan performa tinggi dan latensi rendah.
- **WebSocket** digunakan ketika aplikasi memerlukan komunikasi real-time dua arah secara terus-menerus.
- **Service Mesh** digunakan ketika jumlah microservices sudah besar dan dibutuhkan kontrol komunikasi, keamanan, dan observability di level jaringan.

Kesimpulan Perbandingan (Pola Jawaban UAS)

REST API, gRPC, WebSocket, dan Service Mesh **bukan saling menggantikan**, melainkan **saling melengkapi**. REST API cocok untuk komunikasi client–server dan API publik. gRPC lebih efisien untuk komunikasi internal antar service. WebSocket unggul pada kebutuhan real-time. Service mesh berperan sebagai lapisan infrastruktur untuk mengelola dan mengamankan komunikasi antar service dalam skala besar.

Pemilihan teknologi harus disesuaikan dengan **kebutuhan sistem, skala aplikasi, dan karakteristik komunikasi**, bukan berdasarkan tren semata.

---

9. Contoh Soal UAS & Pembahasan (Berdasarkan Soal Asli)

Bagian ini berisi **contoh soal UAS Interoperabilitas beserta pembahasan uraian** yang dapat digunakan sebagai latihan dan referensi jawaban saat ujian.

Soal 1

Jelaskan pemahaman Anda mengenai interoperabilitas berdasarkan level hardware, level network, level software, dan level organisasi!

Pembahasan:
Interoperabilitas pada level **hardware** berkaitan dengan kemampuan perangkat keras yang berbeda untuk saling terhubung dan berkomunikasi, misalnya komputer, server, dan perangkat IoT yang menggunakan arsitektur berbeda namun tetap dapat bertukar data melalui standar tertentu.

Pada level **network**, interoperabilitas berhubungan dengan kemampuan sistem yang berada di jaringan berbeda untuk berkomunikasi menggunakan protokol jaringan yang sama, seperti TCP/IP, HTTP, atau HTTPS. Standar protokol ini memungkinkan sistem yang heterogen tetap saling terhubung melalui jaringan.

Pada level **software**, interoperabilitas berkaitan dengan kemampuan aplikasi atau sistem yang dibangun dengan bahasa dan platform berbeda untuk saling berkomunikasi, misalnya melalui REST API, gRPC, atau WebSocket.

Sedangkan pada level **organisasi**, interoperabilitas mencakup keselarasan proses bisnis, kebijakan, dan aturan antar organisasi atau unit kerja agar pertukaran data dapat dimanfaatkan secara efektif dan konsisten.

Soal 2

Jelaskan macam-macam method komunikasi HTTP yang dapat digunakan pada RESTful API!

Pembahasan:
Beberapa method HTTP yang umum digunakan pada RESTful API antara lain **GET** untuk mengambil data, **POST** untuk membuat data baru, **PUT atau PATCH** untuk memperbarui data, serta **DELETE** untuk menghapus data. Penggunaan method HTTP yang tepat mencerminkan prinsip REST dan memudahkan pemahaman serta pengelolaan API.

Soal 3

Jelaskan proses bisnis aplikasi yang tim Anda kembangkan dalam Project Based Learning (PBL) yang menerapkan konsep interoperabilitas!

Pembahasan (contoh umum):
Pada project PBL, konsep interoperabilitas diterapkan dengan memisahkan sistem menjadi beberapa modul atau service yang saling berkomunikasi melalui API. Frontend berkomunikasi dengan backend menggunakan REST API untuk pengelolaan data, sedangkan komunikasi antar service internal dapat menggunakan gRPC atau message broker. Dengan pendekatan ini, setiap modul dapat dikembangkan dan dikelola secara terpisah namun tetap terintegrasi dalam satu sistem.

Soal 4

Buat daftar rancangan URL API pada aplikasi tim Anda dalam Project Based Learning!

Contoh Jawaban:

No	URL	Method	Parameter	Deskripsi
1	/api/register	POST	email, password, nama	Registrasi user
2	/api/login	POST	email, password	Login user
3	/api/users	GET	-	Mengambil data user
4	/api/users/{id}	PUT	nama, email	Update data user
5	/api/users/{id}	DELETE	-	Hapus data user

Soal 5

Jelaskan tugas Anda dalam pengembangan aplikasi PBL dan sebutkan anggota tim yang berperan aktif serta kurang berkontribusi!

Pembahasan:
Dalam pengembangan aplikasi PBL, saya berperan dalam perancangan dan implementasi API serta integrasi antar modul aplikasi. Saya bertanggung jawab memastikan komunikasi antar sistem berjalan dengan baik menggunakan prinsip interoperabilitas. Beberapa anggota tim berperan aktif dalam pengembangan backend dan frontend, sedangkan ada juga anggota yang kontribusinya lebih terbatas pada tahap dokumentasi atau pengujian.

> Catatan: Pada ujian asli, bagian ini disesuaikan dengan kondisi dan pembagian tugas tim masing-masing.