Pertemuan 8 — UTS: Tugas Besar

“Satu sistem, tiga cara bicara” — Capaian A: Sistem Terdistribusi

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. Tujuan dan kaitan dengan Capaian A
2. Skenario contoh dan arsitektur acuan
3. Kebutuhan wajib
4. Protokol pengukuran
5. Skenario uji kegagalan
6. Struktur repositori dan laporan
7. Presentasi dan demo
8. Rubrik penilaian
9. Aturan main, jadwal, dan *checklist*
10. Pertanyaan yang akan muncul saat demo

Bobot dan cakupan

UTS bernilai **35%** nilai akhir, mengukur CPMK-1, CPMK-2, dan CPMK-3, dan mencakup seluruh materi pertemuan 1–7: protokol, gaya API, arsitektur, *concurrency*, *replication*, dan Kubernetes.

Tujuan

Mengapa tugas ini berbentuk sistem, bukan ujian tertulis

- Di dunia kerja tidak ada yang bertanya “sebutkan delapan *fallacies*”. Yang ditanya: **mengapa layanan kita lambat**, dan **apa yang terjadi kalau layanan pembayaran mati**.
- Tugas ini memaksa Anda mengalami sendiri setiap panah di diagram arsitektur: memilih protokolnya, mengukurnya, lalu memutusnya.
- Hasilnya berupa kode beserta **bukti terukur**: angka *latency*, *throughput*, dan waktu pulih.

Ringkasan

Satu kalimat sasaran: *bangun sistem kecil yang berbicara dengan tiga cara, ukur harganya masing-masing, lalu buktikan ia tetap hidup ketika sebagian mati.*

Kaitan dengan CPMK

CPMK	Kemampuan yang diukur	Bukti di tugas besar
CPMK-1	Protokol komunikasi, *latency*, *availability*, *fallacies*	Pilihan HTTP/1.1 vs HTTP/2, TLS, angka *latency*, perhitungan *availability*
CPMK-2	Gaya API dan arsitektur	Kontrak REST, `.proto` gRPC, jalur *async*, diagram dan alasan arsitektur
CPMK-3	*Concurrency* dan sistem tahan gagal	*Worker* atau *consumer concurrent*, *replica*, *timeout*/*retry*, uji kegagalan

Setiap komponen rubrik nanti menunjuk ke salah satu baris tabel ini. Kalau suatu bagian tidak punya bukti, nilainya tidak bisa diberikan, sebagus apa pun penjelasan lisannya.

Skenario dan arsitektur acuan

Skenario contoh: toko daring sederhana

- **Layanan pesanan** — menerima pesanan dari klien lewat REST, menyimpan ke basis datanya sendiri.
- **Layanan pembayaran** — dipanggil oleh layanan pesanan lewat **gRPC**; harus cepat dan bertipe ketat.
- **Layanan notifikasi** — menerima *event* “pesanan dibayar” lewat *message queue*, mengirim surel atau *message* secara *asynchronous*.

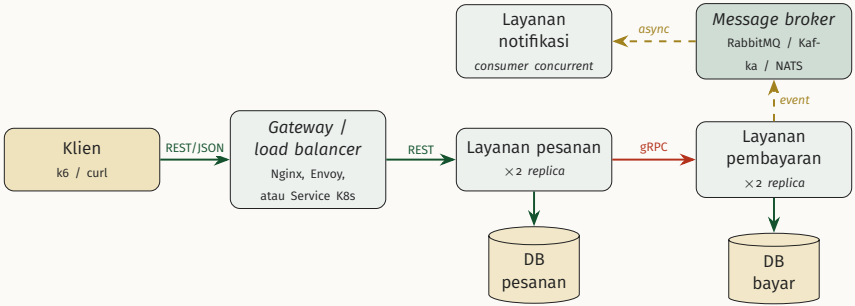
Boleh mengganti domain

Domain lain boleh dipakai selama strukturnya setara: tiga layanan atau lebih, satu panggilan REST dari luar, satu panggilan gRPC antar layanan, dan satu jalur *async*. Contoh: pemesanan ojek, peminjaman buku perpustakaan, pendaftaran kegiatan kampus, pencatatan emisi.

Jangan terjebak *feature*

Nilai tidak datang dari banyaknya *feature* bisnis. Satu alur yang lengkap dan terukur jauh lebih berharga daripada sepuluh halaman CRUD.

Arsitektur acuan



Garis hijau: *synchronous* REST. Garis merah: *synchronous* gRPC. Garis putus-putus: *asynchronous*. Tiap layanan memiliki datanya sendiri; tidak ada layanan yang membaca *database* layanan lain.

Ringkasan

Gambar ini acuan, bukan cetakan. Yang wajib adalah **tiga cara bicara** dan **alasan** mengapa tiap panah memakai cara itu.

Bahasa dan alat yang disarankan

Kebutuhan	Pilihan aman	Catatan
Bahasa layanan	Go, Python (FastAPI), Java/Kotlin (Spring), Scala (Akka)	Boleh berbeda antar layanan (*polyglot*); kontrak gRPC menjembataninya
gRPC	`protoc` + plugin bahasa masing-masing	*File* `.proto` disimpan di satu folder bersama
Message broker	RabbitMQ, Kafka, atau NATS	Satu saja; jelaskan alasannya
Database	PostgreSQL, SQLite, atau Redis	Satu per layanan
Orkestrasi	Docker Compose; Kubernetes (kind, k3s, minikube) untuk nilai lebih	Lihat rubrik
Uji beban	k6 (REST), ghz (gRPC), wrk/hey (REST)	Jalankan dari mesin atau *container* terpisah bila bisa

Kebutuhan wajib

Kebutuhan fungsional dan teknis

Wajib

1. Minimal **tiga layanan** yang berjalan sebagai proses atau *container* terpisah.
2. Satu jalur **REST** dari klien ke sistem.
3. Satu jalur **gRPC** antar layanan, dengan kontrak `.proto`.
4. Satu jalur ***asynchronous***: *message queue*, *log* terdistribusi, atau *actor*.
5. Minimal satu layanan berjalan dengan $\geq$**2 *replica*** di belakang *load balancer*.
6. Sistem dinyalakan dengan **satu perintah**: `docker compose up` atau `kubectl apply -f k8s/`.

Wajib di dalam kode

7. Setiap panggilan jaringan punya ***timeout***.
8. *Retry* hanya untuk operasi yang ***idempotent***, dengan *backoff*.
9. Satu bagian *concurrent* yang disengaja: *worker pool*, *consumer* paralel, *goroutine*, atau *actor*.
10. *Health check* (`/healthz`) di setiap layanan.
11. *Log* terstruktur dengan *request id* yang diteruskan antar layanan.

Bila salah satu kebutuhan wajib tidak dipenuhi, nilai maksimum komponen terkait dibatasi setengahnya.

Nilai lebih (tidak wajib)

- **Kubernetes** — *deployment* dengan *liveness* dan *readiness probe*, *horizontal pod autoscaler*.
- **GraphQL** — satu *endpoint* GraphQL sebagai alternatif REST untuk klien, beserta perbandingannya.
- **HTTPS dan mTLS** — TLS di depan *gateway*, mTLS antar layanan.
- ***Circuit breaker*** — layanan pesanan berhenti memanggil pembayaran sementara ketika pembayaran sedang gagal.
- ***Observability*** — *metric* Prometheus dan *tracing* OpenTelemetry, ditunjukkan saat demo.
- ***Idempotency key*** — pesanan yang dikirim dua kali tidak tertagih dua kali.

Ringkasan

Kerjakan dulu semua yang wajib sampai terukur. Nilai lebih dari bagian opsional tidak menutup kekurangan bagian wajib.

Protokol pengukuran

Apa yang diukur

Eksperimen E1: REST vs gRPC

Operasi yang **sama** (misalnya "cek status pembayaran") diekspos dua kali: sekali lewat REST/JSON, sekali lewat gRPC/Protobuf. Ukur keduanya dengan beban yang sama.

Eksperimen E2: pengaruh *replica*

Jalankan layanan pesanan dengan 1, 2, dan 4 *replica*. Ukur *throughput* maksimum dan *latency* pada beban tetap.

Eksperimen E3: *sync* vs *async*

Bandingkan waktu respons ke klien bila notifikasi dikirim *synchronous* (ditunggu) vs *asynchronous* (lewat *queue*).

Metric wajib

- *Latency* P_{50}, P_{95}, P_{99} (milidetik)
- *Throughput* (*request* per detik)
- *Error rate* (%)
- Ukuran *payload* (byte) untuk E1
- Pemakaian CPU dan memori per layanan (`docker stats` atau `kubectl top`)

Rata-rata saja **tidak cukup**. Pengguna merasakan ekor distribusi, bukan rata-ratanya.

Cara mengukur yang jujur

1. **Catat lingkungan** — CPU, RAM, sistem operasi, versi bahasa, apakah semua berjalan di satu laptop.
2. **Pemanasan** — jalankan beban 30–60 detik dan buang hasilnya (JIT, koneksi, *cache*).
3. **Durasi cukup** — tiap pengukuran minimal 60 detik.
4. **Ulangi** — minimal **3 kali**; laporkan median dan rentang (min–maks).
5. **Satu variabel per eksperimen** — jangan mengubah protokol dan jumlah *replica* sekaligus.
6. **Simpan data mentah** — *output* k6/ghz dalam JSON atau CSV di folder `bench/results/`, plus skrip yang menghasilkannya.

Ringkasan

Angka yang tidak bisa diulang orang lain hanyalah anekdot. Sertakan perintah persisnya.

Contoh perintah dan bentuk tabel yang diminta

```
# REST: 50 virtual user, 60 detik
k6 run --vus 50 --duration 60s \
  --summary-export bench/results/rest-1.json \
  bench/rest.js

# gRPC: 50 koneksi konkuren, 60 detik
ghz --insecure --proto proto/payment.proto \
  --call payment.Payment/GetStatus \
  -d '{"order_id":"A-1"}' -c 50 -z 60s \
  -O json -o bench/results/grpc-1.json \
  localhost:50051
```

Tabel E1 (isi dengan median dari 3 kali ulang):

	P_{50}	P_{99}	req/s	*payload*
REST/JSON	...	...	...	...
gRPC	...	...	...	...

Grafik yang diminta:

- E1: *bar chart* P_{50} dan P_{99} REST vs gRPC
- E2: *throughput* terhadap jumlah *replica* (1, 2, 4)
- E3: distribusi *latency sync* vs *async*
- Uji kegagalan: *error rate* terhadap waktu

Skenario uji kegagalan

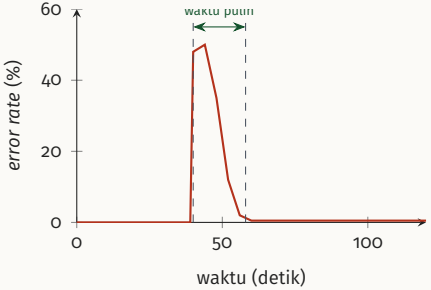
Putuskan panahnya, lalu amati

No	Gangguan	Cara melakukan	Yang dilaporkan
F1	Satu *replica* layanan pesanan mati	`docker kill` / `kubectl delete pod` saat beban berjalan	*Error rate*, waktu sampai normal
F2	Layanan pembayaran mati total	Hentikan semua *replica*-nya 30 detik	Perilaku *timeout*, *message error* ke klien
F3	Layanan notifikasi mati	Hentikan *consumer* 2 menit, lalu nyalakan	Apakah *message* menumpuk dan terkirim setelah pulih?
F4	Jaringan lambat	`tc netem delay 200ms` pada satu *container*	Dampak ke P_{99}, apakah *retry* memperparah

Wajib: F1, F2, F3. F4 untuk nilai lebih. Untuk setiap skenario tuliskan **dugaan sebelum mencoba**, hasil yang terjadi, dan penjelasan bila keduanya berbeda.

Bagaimana mengukur waktu pulih

- Jalankan beban tetap (misalnya 20 req/s) sepanjang eksperimen.
- Catat waktu setiap *request* beserta status suksesnya.
- Waktu pulih = selisih antara gangguan dimulai dan saat *error rate* kembali di bawah 1%.
- Hitung *availability* selama eksperimen:
  $A = \dfrac{\text{request sukses}}{\text{total request}}$.



Ilustrasi bentuk grafik, bukan hasil nyata.

Ringkasan

Kaitkan dengan pertemuan 1: $A = \text{MTBF}/(\text{MTBF} + \text{MTTR})$. Yang Anda ukur di sini adalah MTTR sistem Anda.

Repositori, laporan, dan demo

Struktur repositori

```
uts-kelompok-XX/
  README.md          # cara menjalankan
  docker-compose.yml
  k8s/               # opsional
  proto/             # kontrak gRPC
  services/
    order/
    payment/
    notification/
  bench/
    rest.js  run.sh
    results/         # data mentah
  docs/
    arsitektur.png
    laporan.pdf
  AI-USAGE.md        # catatan alat AI
```

- `README.md` memuat: prasyarat, satu perintah untuk menyalakan, satu perintah untuk menjalankan semua *benchmark*, dan pembagian tugas anggota.
- Riwayat *commit* harus memperlihatkan kontribusi **setiap anggota**. Satu *commit* raksasa di malam terakhir adalah tanda bahaya.
- Jangan menyimpan rahasia (*password*, token) di repositori; gunakan `.env.example`.

Struktur laporan (maksimal 12 halaman)

1. **Pendahuluan** — domain, alur utama, dan batasan.
2. **Arsitektur** — diagram, daftar layanan, kontrak API, alasan pemilihan REST, gRPC, dan jalur *async* untuk tiap panah.
3. **Implementasi** — *concurrency*, *timeout*, *retry*, *replica*, cara menjalankan.
4. **Metodologi pengukuran** — lingkungan, alat, beban, durasi, pengulangan.
5. **Hasil** — tabel dan grafik E1–E3.
6. **Uji kegagalan** — F1–F3 (F4): dugaan, hasil, penjelasan.
7. **Diskusi** — kaitkan dengan materi: *fallacies*, HTTP/2, Protobuf, CAP. Apa yang akan diubah bila pengguna naik 100 kali?
8. **Kesimpulan dan keterbatasan**.
9. **Lampiran** — pembagian kerja dan catatan penggunaan AI.

Presentasi dan demo

Format per kelompok: 20 menit

Presentasi arsitektur dan hasil	7 menit
Demo langsung	6 menit
Tanya jawab	7 menit

Satu sesi UTS 100 menit memuat sekitar lima kelompok; kelompok lainnya dijadwalkan di sesi tambahan.

Demo wajib memperlihatkan

1. Sistem dinyalakan dari nol dengan satu perintah
2. Satu alur penuh: REST $\rightarrow$ gRPC $\rightarrow$ *event* $\rightarrow$ notifikasi
3. Satu *benchmark* singkat berjalan
4. Satu uji kegagalan (F1 atau F2) secara langsung

Ringkasan

Setiap anggota wajib berbicara dan menjawab minimal satu pertanyaan. Nilai individu dapat berbeda dari nilai kelompok bila kontribusi tidak seimbang.

Penilaian dan aturan main

Rubrik penilaian

Komponen	Bobot	Nilai penuh bila
Arsitektur dan alasan	15%	Diagram jelas; tiap panah punya alasan protokol yang tepat; data per layanan
Implementasi wajib	20%	Semua 11 kebutuhan wajib berjalan; kode rapi; satu perintah menyalakan sistem
Pengukuran (E1–E3)	20%	Protokol jujur, 3 kali ulang, P_{50} / P_{99}, grafik lengkap, data mentah tersedia
Uji kegagalan (F1–F3)	15%	Dugaan, hasil, waktu pulih, dan penjelasan yang terkait materi
Laporan	15%	Struktur lengkap, analisis yang menjelaskan sebab, keterbatasan jujur
Presentasi dan demo	15%	Demo berhasil; semua anggota menjawab dengan paham
Nilai lebih	*+10%*	*Kubernetes, GraphQL, mTLS, circuit breaker, observability, F4*

Nilai akhir UTS dibatasi 100. UTS bernilai 35% dari nilai mata kuliah.

Aturan main

Kelompok

- 3–4 orang, ditetapkan paling lambat minggu ke-5
- Pembagian tugas tertulis di `README.md`
- Konflik kelompok dilaporkan ke dosen sebelum minggu ke-7, bukan setelah tenggat

Keterlambatan

Dikurangi 10 poin per hari, maksimal 3 hari. Setelah itu tidak diterima.

Alat bantu AI dan plagiarisme

- Boleh memakai alat bantu AI. Tuliskan di `AI-USAGE.md`: alat apa, untuk bagian mana, dan apa yang Anda ubah.
- Setiap anggota harus mampu menjelaskan **baris mana pun** dari kode kelompoknya. Tidak bisa menjelaskan = bagian itu tidak dinilai.
- Kode dari sumber lain wajib disebut sumbernya.
- Menyalin pekerjaan kelompok lain atau memalsukan angka pengukuran: nilai UTS nol.

Jadwal



Pada pertemuan 5 rancangan arsitektur dinilai sebagai tugas; pada pertemuan 7 progres dinilai sebagai tugas. Keduanya masuk komponen “tugas, praktikum, dan kuis” (20%), bukan nilai UTS.

Checklist pengumpulan

- ☐ Tautan repositori dapat diakses dosen
- ☐ `README.md`: prasyarat, satu perintah jalan, satu perintah *benchmark*
- ☐ Tiga layanan atau lebih, REST + gRPC + *async*
- ☐ *File* `.proto` ada di `proto/`
- ☐ $\geq$2 *replica* di belakang *load balancer*
- ☐ *Timeout*, *retry*, *health check*, *request id*
- ☐ Data mentah *benchmark* di `bench/results/`
- ☐ Grafik E1, E2, E3, dan uji kegagalan
- ☐ Laporan PDF $\leq$ 12 halaman di `docs/`
- ☐ `AI-USAGE.md` terisi
- ☐ Pembagian kerja tertulis
- ☐ Slide presentasi

Pertanyaan yang akan muncul saat demo

1. Mengapa panah ini memakai gRPC dan bukan REST? Bagaimana kalau dibalik?
2. Angka P_{99} Anda jauh lebih besar daripada P_{50}. Apa penyebabnya?
3. Apa yang terjadi bila *request* pembayaran *timeout* padahal sebenarnya sudah diproses? Apakah pelanggan bisa tertagih dua kali?
4. Mengapa notifikasi dibuat *asynchronous*? Apa yang dikorbankan?
5. Kalau *message broker* Anda mati, apa yang hilang?
6. Fallacy mana yang paling terasa saat Anda mengerjakan tugas ini?
7. Menambah *replica* dari 2 ke 4 tidak menggandakan *throughput*. Mengapa?
8. Tunjukkan satu baris kode yang Anda tulis sendiri dan jelaskan.

Ringkasan

Pertanyaan ini sengaja diberikan sekarang. Kelompok yang menyiapkan jawabannya dengan data hampir pasti mendapat nilai baik.

Setelah UTS

Paruh pertama menjawab: bagaimana sistem tetap **benar dan hidup** walau terpecah di banyak mesin.

Paruh kedua berganti pertanyaan: bagaimana memakai banyak prosesor agar **secepat mungkin**. Kita turun ke CPU, *cache*, OpenMP, MPI, dan GPU.

Pertemuan 9

Pengantar *High Performance Computing*: *speedup*, efisiensi, *strong* dan *weak scaling*, serta cara mengukur performa yang jujur.

Selamat mengerjakan

Ukur dulu, baru berpendapat.