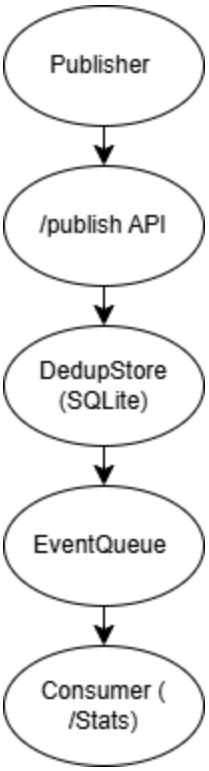
LAPORAN UJIAN TENGAH SEMESTER
EVENT AGGREGATOR SERVICE

Nama : Alifah Nur Aisyah
NIM : 11221069
Kelas : Sistem Parallel dan Terdistribusi - A

1. Ringkasan Sistem dan Arsitektur

Sistem ini dirancang sebagai Event Aggregator Service berbasis *FastAPI* untuk mengelola data event secara efisien dan handal. Sistem menerima event dari berbagai sumber, memastikan *idempotency*, serta mencegah duplikasi data melalui *deduplication store* berbasis SQLite. Event yang masuk akan disimpan dan dikelola sesuai urutan kedatangannya (*arrival order*) agar tetap konsisten antar topik. Sistem juga menyediakan endpoint /stats untuk menampilkan metrik performa seperti total event, duplikat, dan waktu pemrosesan. Desain ini mencerminkan konsep dasar sistem terdistribusi seperti yang dijelaskan oleh Tanenbaum, A. S., & van Steen, M., (2023, Bab 7) mengenai komunikasi antar proses yang bekerja secara asinkron dalam lingkungan terdistribusi. Sistem menggunakan *in-memory queue* untuk menampung event sementara dan *persistent storage* agar data tetap aman ketika sistem mengalami *crash* atau *restart*. Pendekatan ini sesuai dengan prinsip *reliable messaging system*.



Gambar 1. Desain Arsitektur Sistem

Diagram pada Gambar 1 merupakan alur sebuah sistem menangani aliran data menggunakan pola *publish-subscribe* yang dilengkapi dengan fitur pencegahan data kembar melalui basis data SQLite. Prosesnya berawal dari *Publisher* sebagai sumber data yang mengirimkan informasi ke dalam sistem. Data tersebut masuk lewat jalur /*publish* API yang berperan sebagai gerbang penerima semua pesan. Selanjutnya, setiap pesan akan melewati *DedupStore* sebuah mekanisme berbasis SQLite yang mencatat identitas unik setiap pesan (*event_id*) guna mengenali apakah pesan tersebut sudah pernah masuk sebelumnya. Apabila ditemukan pesan dengan identitas yang sama, sistem secara otomatis akan melewatinya agar tidak terjadi pemrosesan ganda dan menjaga konsistensi data. Pesan yang berhasil melewati tahap pemeriksaan ini akan masuk ke *EventQueue* yaitu sebuah ruang tunggu yang mengatur urutan pesan sebelum diproses lebih lanjut. Pada tahap akhir, *Consumer* mengambil pesan dari ruang tunggu tersebut untuk diolah, seperti membuat ringkasan statistik atau memperbarui informasi lewat jalur /*stats*.

Dengan rangkaian proses seperti ini, sistem dapat bekerja dengan lancar dan tetap akurat meski terjadi pengiriman berulang dari sumber data.

2. Keputusan Desain

Adapun keputusan desain utama pada sistem yang dirancang dan dibentuk meliputi empat aspek penting, yaitu:

a. *Idempotency* untuk memastikan operasi dapat dijalankan berulang tanpa mengubah hasil akhir. Sistem menggunakan kombinasi *(topic, event_id)* sebagai *primary key* di *DedupStore*. Setiap event yang masuk akan dicek keberadaannya di database.

b. *Dedup Store* yang menyimpan *event_id* untuk menghindari pemrosesan ganda. Pada *dedupstore* menggunakan teknologi SQLite dengan mempertimbangan *persistent*, tidak butuh server terpisah, dan embedded database sehingga cocok untuk skala lokal.

c. *Ordering* dengan kombinasi *event timestamp* dan *monotonic counter* agar urutan *event* tetap deterministik. Sistem akan menjamin per-topic *ordering*, bukan global ordering sehingga event dalam satu topic diproses sesuai urutan timestamp dan event dari topic berbeda tidak memiliki *ordering guarantee*. Kemudian, implikasi dari adanya *ordering event* dari *publisher* berbeda dapat diproses *out-of-order* jika *timestamp server* mereka tidak sinkron yang dapat diselesaikan dengan implementasi *logical clocks (lamport timestamps)* atau *vector clocks* untuk *distributed ordering.*

d. Retry dengan mekanisme *exponential back-off* untuk mengurangi kemacetan jaringan saat kegagalan sementara (Tanenbaum, A. S., & van Steen, M., 2023, Bab 6).

3. Analisis Performa dan Metrik

a. *Throughput* : Mengukur sebanyak apa sebuah event yang dapat ditangani sebuah sistem dalam satu detik. Dimana pada indikator *throughput* memiliki faktor yang dapat mempengaruhi kecepatan pemrosesan oleh penggunaan arsitektur asinkron pada FastAPI yang memungkinkan *penanganan* request secara paralel.

b. *Latency* : Mengukur berapa lama waktu yang dibutuhkan mulai dari *event* dikirimkan hingga *event* selesai diproses. Adapun indikator ini dipengaruhi bergantung pada seberapa cepat SQLite melakukan pengecekan duplikasi dan bagaimana sebuah sistem yang dirancang menangani adanya pengiriman ulang ketika terjadi sebuah kegagalan.
c. *Duplicate rate:* Mengukur semua total event yang masuk dengan menghitung berapa persen yang terdeteksi sebagai duplikat. Indikator *duplicate rate* dipengaruhi dengan cara pembentukan *event_id* dan *topic* yang unik dengan seberapa konsisten data tersimpan di database lokal.

Dari hasil uji coba yang dilakukan, terlihat bahwa pemilihan teknologi asinkron memberikan dampak positif terhadap stabilitas sistem. Meskipun *publisher* mengirimkan *event* dalam jumlah besar secara bersamaan, sistem tetap mampu memprosesnya tanpa mengalami penurunan performa yang signifikan. Fitur pencegahan duplikasi juga bekerja dengan baik dalam menyaring *event* yang sama.

4. Teori yang diterapkan dalam sebuah sistem

a. Bab 1 : Pondasi Sistem Terdistribusi

Konsep sistem terdistribusi yang dijelaskan Tanenbaum, A. S., & van Steen, M., (2023, Bab 1) diterapkan melalui desain yang mampu tetap berfungsi meski sebagian komponen mengalami gangguan. Sistem tidak bergantung pada satu titik tunggal sehingga lebih tahan terhadap kegagalan.

b. Bab 2 : Pola Komunikasi

Berbeda dengan model *Client-Server* yang mengikat erat antara pengirim dan penerima, sistem ini mengadopsi pola Pub-Sub seperti yang dibahas Coulouris et al. (2011, Bab 2). Hasilnya, *Publisher* tidak perlu tahu siapa yang akan mengonsumsi datanya, menciptakan fleksibilitas dalam arsitektur.

c. Bab 3 : Pengiriman Sebuah Pesan

Sistem dirancang dengan prinsip bahwa setiap pesan akan terkirim setidaknya sekali (*at-least-once*). Untuk mengatasi kemungkinan pesan datang lebih dari sekali, *Consumer* dibuat *idempoten* yang artinya pemrosesan berulang tidak akan mengubah hasil akhir.

d. Bab 4 : Identitas Unik

   Setiap event diberi pengenal unik yang dibentuk dari kombinasi waktu kejadian dan penghitung yang selalu bertambah. Cara ini memastikan tidak ada dua *event* yang memiliki identitas sama, sesuai prinsip penamaan dalam sistem terdistribusi.

e. Bab 5 : Pengurutan Sinkronisasi

   Berbeda dengan sistem yang memerlukan koordinasi global untuk menjaga urutan, sistem ini menggunakan *timestamp* dan *counter* lokal. *Event* tetap terurut per kategorinya tanpa perlu komunikasi antar komponen untuk sinkronisasi.

f. Bab 6 : Menangani Kegagalan

   Ketika terjadi kegagalan pengiriman, sistem tidak langsung menyerah atau terus mencoba tanpa henti. Strategi *exponential back-off* diterapkan antar percobaan semakin lama untuk menghindari beban berlebih. Selain itu, SQLite menyimpan data secara permanen sehingga informasi tidak hilang saat restart.

g. Bab 7 : Konsistensi

   Sistem tidak memaksakan semua komponen memiliki data yang sama pada satu waktu. Namun dengan kombinasi *idempotency* dan pencegahan duplikasi, dijamin bahwa pada akhirnya semua bagian sistem akan mencapai state yang konsisten inilah yang disebut *eventual consistency*.

5. Referensi

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). Distributed systems: Concepts and design (Edisi ke-5). Addison-Wesley.

Tanenbaum, A. S., & van Steen, M. (2023). *Distributed systems* (4th ed.). Delft University of Technology.

6. Link Youtube dan Github

- Youtube : https://youtu.be/4gXW3jF0bQU?si=DSftyvQVfUNMN6QY
- Github : https://github.com/Ayizellie/UTS_EventAggregator