BÁO CÁO CUỐI KỲ

ĐIỆN TOÁN ĐÁM MÂY VÀ PHÁT TRIỂN ỨNG DỤNG HƯỚNG DỊCH VỤ - SE360.Q11

GVHD: TS. Lê Văn Tuấn
Hồ Nguyễn Minh Sang - 23521338
Nguyễn Thanh Kiệt - 22520720
Huỳnh Chí Hên - 23520455

MỤC LỤC

INTRODUCE

INTRODUCE

UIT-Go là backend phân tán mô phỏng nền tảng gọi xe, xử lý nghìn request/giây.

Mục tiêu chính của Module A Thiết kế Kiến trúc cho Scalability & Performance:

- Mở rộng ngang thông qua các microservice không trạng thái và điều phối container (Docker + ECS).
- Giao tiếp bất đồng bộ theo sự kiện để hấp thụ các đợt tăng tải đột ngột.
- Độ trễ thấp nhờ cơ chế cache và truy vấn địa lý trong bộ nhớ.
- Cân bằng hợp lý giữa hiệu năng, chi phí và độ phức tạp

INTRODUCE

UIT-Go theo hướng domain-driven architecture

Service	Responsibility	Data Layer	Communication
API Gateway	Central entry, routing, JWT validation	—	REST
User Service	Manage passengers & drivers	PostgreSQL	REST
Driver Service	Real-time location updates & availability	Redis Geo	REST + RabbitMQ
Trip Service	Trip creation, driver matching, lifecycle management	PostgreSQL	REST + RabbitMQ
Notification Service	Push events to users in real time	—	RabbitMQ + WebSocket

PROJECT OBJECTIVE

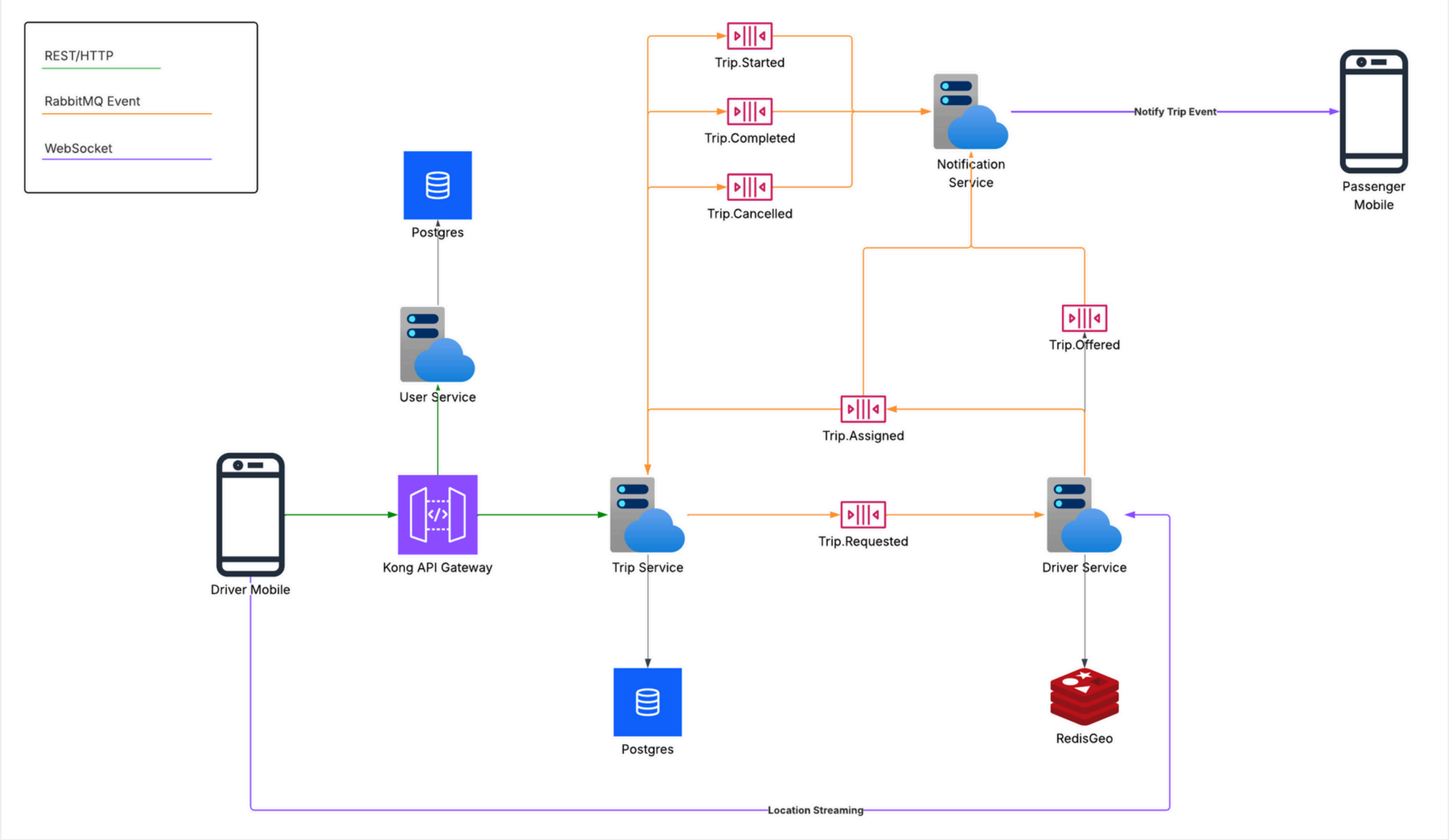
PROJECT OBJECTIVE

Kế hoạch và mục tiêu triển khai cho module A Thiết kế Kiến trúc cho Scalability & Performance

Quyết định	Hướng triển khai	Kỳ vọng	Trade-offs
Messaging bất đồng bộ (RabbitMQ hoặc SQS)	Tách TripService và DriverService bằng message queue	Hấp thụ tải tăng đột ngột, tránh nghẽn dịch vụ	Tăng nhẹ độ trễ phản hồi (~200 ms)
Redis Geo cho tra cứu tài xế realtime	Lưu & truy vấn tọa độ tài xế trong bộ nhớ bằng Redis GEO	Tra cứu địa lý <10 ms, giảm lượng query DB	Tốn RAM hơn, cần TTL và cơ chế dọn dẹp
PostgreSQL Read Replicas	Tách truy vấn đọc khỏi database chính	Tăng throughput khi tải đọc cao	Có độ trễ replication (~100–200 ms)
ECS Auto Scaling	Tự động scale service dựa trên CPU, queue depth hoặc latency	Đảm bảo co giãn khi lưu lượng tăng	Chi phí thay đổi theo mức tải
Idempotent Message Consumers	Đảm bảo mỗi message chỉ được xử lý đúng một lần, kể cả khi retry	Tăng độ an toàn, tránh gán chuyến trùng lặp	Logic xử lý phức tạp hơn

ARCHITECTURE

EVENT-DRIVEN ARCHITECTURE



ARCHITECTURAL DECISION

Quyết Định	Mục Đích Sử Dụng	Lý Do Chính
Redis (thay vì DynamoDB)	Lưu trữ Vị trí Tài xế	Cần **tốc độ truy cập cực cao** (low-latency) và hiệu năng **in-memory** cho dữ liệu thường xuyên thay đổi/truy vấn.
RESTful (thay vì gRPC)	Giao tiếp API bên ngoài	Ưu tiên **tương thích rộng** (Web, Mobile) và **đơn giản hóa công cụ** (tooling) hơn là tốc độ truyền tải tối đa.
RabbitMQ (thay vì Kafka)	Xử lý Sự kiện Nội bộ	Tập trung vào **đảm bảo phân phối** (durable delivery) và ngữ nghĩa **Hàng đợi Tin nhắn** (Queue-based) đơn giản hơn so với luồng dữ liệu (streaming).
Kong Gateway	Quản lý API	Cung cấp các tính năng cần thiết như **Rate Limiting**, **Authentication** và **Observability** tập trung.
PostgreSQL (thay vì Mongo)	Dữ liệu Kinh doanh Cốt lõi	Yêu cầu tính toàn vẹn dữ liệu **ACID** và khả năng xử lý các quan hệ phức tạp (joins, transactions).
Websocket	Thông báo thời gian thực	Cung cấp kết nối **song công (bi-directional)**, liên tục, cần thiết cho việc gửi thông báo ngay lập tức.
Terraform	Quản lý Cơ sở hạ tầng	Đảm bảo tính nhất quán và khả năng **tái tạo cơ sở hạ tầng** thông qua mã nguồn (Infrastructure as Code - IaC).

MODULE A – ARCHITECTURAL APPROACH

Hướng tiếp cận	Chi tiết thực thi
Event-Driven & Decoupled Architecture	• Tách biệt TripService ↔ DriverService qua RabbitMQ • Hạn chế coupling, cho phép xử lý bất đồng bộ & chịu tải tốt hơn
Stateless Microservices	• Tất cả logic xử lý được thiết kế stateless • Cho phép scale-out linh hoạt và dễ mở rộng trong tương lai
Real-Time Geospatial Processing	• Redis GEO lưu toạ độ tài xế, lookup <10ms • Tự động loại tài xế offline/outdated bằng TTL
Distributed Locking for Correct Assignment	• Redis SETNX đảm bảo mỗi chuyến chỉ được nhận 1 lần • Giải quyết race-condition trong môi trường nhiều tài xế cạnh tranh
Observability-Driven Development	• K6 → load injection • Prometheus → metrics scraping • Grafana → end-to-end monitoring • Cho phép đánh giá latency, contention, queue depth theo thời gian thực

DEMO

EVALUATION SCENARIOS

SCENARIO 1 — BASELINE LOAD TEST

SCENARIO 2 — PASSENGER–DRIVER END-TO-END WORKFLOW

SCENARIO 3 — TRIP SPIKE TEST (200 VUS)

SCENARIO 4 — SPIKE E2E TEST (200 DRIVERS)

SCENARIO 5 — RAMP-UP UNLIMITED (0 → 3000 VUS)

RESULTS & VALUATION

RESULTS & VALUATION

SCENARIO 1 — BASELINE LOAD TEST

KẾT QUẢ

- **P95 LATENCY: 12.2 MS**
- **AVG LATENCY: 6.2 MS**
- **ERROR RATE: 0%**
- **THROUGHPUT: ~26 REQ/S**

=>TRIP CREATION ĐƯỢC TỐI ƯU RẤT TỐT VÀ KHÔNG PHẢI ĐIỂM NGHẼN. POSTGRESQL XỬ LÝ CÁC YÊU CẦU GHI LIÊN TỤC MÀ KHÔNG XUẤT HIỆN LOCK CONTENTION HOẶC TRUY VẤN CHẬM.

SCENARIO 2 — PASSENGER–DRIVER END-TO-END WORKFLOW

KẾT QUẢ

- **P95 LATENCY: 13.01 MS**
- **ERROR RATE: 0%**
- **ITERATION DURATION: ~3.56 S**
- **ASSIGNMENT CORRECTNESS: 100%**

=>TOÀN BỘ PIPELINE BẤT ĐỒNG BỘ (RABBITMQ), REDIS GEO VÀ CƠ CHẾ KHÓA SETNX HOẠT ĐỘNG CHÍNH XÁC DƯỚI TẢI TRUNG BÌNH. LATENCY THẤP CHỨNG TỎ CÁC THAO TÁC LIÊN SERVICE, GEOSPATIAL LOOKUP VÀ MESSAGE PUBLISHING CHỈ TẠO ĐỘ TRỄ TỐI THIỂU.

RESULTS & VALUATION

SCENARIO 3 — TRIP SPIKE TEST (200 VUS)

KẾT QUẢ

- **P95 LATENCY: 42.32 MS**
- **AVG LATENCY: 15.64 MS**
- **ERROR RATE: 0%**
- **THROUGHPUT: ~197 REQ/S**

=>TRIPSERVICE XỬ LÝ TỐT SPIKE LỚN, VẪN DUY TRÌ LATENCY THẤP VÀ KHÔNG PHÁT SINH LỖI. POSTGRESQL TIẾP TỤC XỬ LÝ CÁC THAO TÁC GHI ỔN ĐỊNH, KHÔNG BỊ BÃO HÒA.ĐIỀU NÀY KHẲNG ĐỊNH LUỒNG XỬ LÝ ĐỒNG BỘ (SYNCHRONOUS WRITE PATH) ĐỦ MẠNH ĐỂ CHỊU ĐƯỢC BURST TRAFFIC.

SCENARIO 4 — SPIKE E2E TEST (200 DRIVERS)

KẾT QUẢ

- **P95 LATENCY: 130.23 MS**
- **AVG LATENCY: 24.1 MS**
- **HTTP ERROR RATE: 38.87%**
- **DRIVER ONLINE SUCCESS: ~99%**
- **TRIP CREATION SUCCESS: 100%**
- **DRIVER ACCEPT SUCCESS HOẶC CONFLICT (200/409): ~93%**

=>TỶ LỆ LỖI CAO CHỦ YẾU CÓ THỂ ĐẾN TỪ CẠNH TRANH KHÓA REDIS SETNX. CÓ THỂ ĐÂY LÀ BOTTLENECK CỦA HỆ THỐNG

RESULTS & VALUATION

SCENARIO 5 — RAMP-UP UNLIMITED (0 → 3000 VUS)

KẾT QUẢ

- **AVG LATENCY: 1.42 S**
- **P95 LATENCY: 6.81 S**
- **HTTP ERROR RATE: 42.73%**
- **CHECKS SUCCEEDED: 91.20%**
- **TRIP CREATION SUCCESS: ~99%**
- **DRIVER ACCEPT SUCCESS HOẶC CONFLICT: ~75%**
- **FATAL TIMEOUT / HTTP 500: KHÔNG XUẤT HIỆN**

=>KHI TẢI TĂNG VƯỢT QUÁ GIỚI HẠN XỬ LÝ HIỆN TẠI, HỆ THỐNG BẮT ĐẦU BIỂU HIỆN ĐÚNG BOTTLENECK CỦA KIẾN TRÚC. ĐIỀU NÀY PHẢN ÁNH RÕ RÀNG RẰNG ĐIỂM NGHẼN CHÍNH CỦA TOÀN HỆ THỐNG NẰM Ở LUỒNG DRIVER ACCEPT, BAO GỒM:

1. **REDIS SETNX LOCK — KHỐI LƯỢNG YÊU CẦU CẠNH TRANH QUÁ LỚN, GÂY TẮC NGHẼN TỰ NHIÊN.**
2. **SINGLE-THREADED REDIS EXECUTION MODEL — XỬ LÝ TUẦN TỰ, NÊN THROUGHPUT GIẢM MẠNH KHI SỐ CẠNH TRANH TĂNG.**
3. **RETRY LOGIC LÀM TĂNG ÁP LỰC LÊN REDIS VÀ KÉO DÀI ITERATION.**

TRONG KHI ĐÓ, LUỒNG TRIP CREATION VẪN DUY TRÌ ~99% SUCCESS RATE, CHỨNG MINH RẰNG BOTTLENECK KHÔNG NẰM Ở TRIPSERVICE HAY POSTGRESQL, MÀ TẬP TRUNG TẠI “ACCEPT PATH” VỐN CÓ ĐỘ PHỨC TẠP VÀ TÍNH CẠNH TRANH CAO.

CONCLUSION

CONCLUSION

1. **KIẾN TRÚC MODULE A ĐẠT HIỆU NĂNG TỐT Ở TẢI THỰC TẾ**
   - LATENCY THẤP (P95 < 150MS) TRONG CÁC KỊCH BẢN TƯƠNG ỨNG VỚI NHU CẦU THỰC.
   - TRIPSERVICE GIỮ 99% TRIP CREATION SUCCESS NGAY CẢ KHI HỆ THỐNG BỊ ĐẨY TỚI CỰC HẠN.
2. **ĐIỂM NGHẼN (BOTTLENECK) NẰM Ở DRIVER ACCEPT PATH**
   - REDIS SETNX LOCK LÀ NƠI XẢY RA CẠNH TRANH CAO NHẤT.
   - TỶ LỆ LỖI TĂNG MẠNH KHI ≥ 1000 VUS.
   - ĐÂY LÀ BOTTLENECK RÕ RÀNG NHẤT CẦN ĐƯỢC TỐI ƯU TRONG TƯƠNG LAI.
3. **KIẾN TRÚC CÓ “GRACEFUL DEGRADATION” TỐT**
   - KHÔNG CRASH, KHÔNG 500, KHÔNG TIMEOUT DÙ QUÁ TẢI.
   - CÁC SERVICE LUÔN PHẢN HỒI, ĐẢM BẢO TÍNH ỔN ĐỊNH VÀ TÍNH NHẤT QUÁN DỮ LIỆU.
4. **EVENT-DRIVEN PIPELINE HOẠT ĐỘNG CHÍNH XÁC**
   - RABBITMQ XỬ LÝ QUEUE ỔN ĐỊNH Ở MỌI KỊCH BẢN.
   - KHÔNG BACKLOG OVERFLOW, KHÔNG CONSUMER STARVATION.

FUTURE WORK

- **Mở rộng khả năng scale của Driver Accept**
- **Tối ưu hiệu năng phân bổ tài xế**
- **Tối ưu hệ thống quan sát**
- **Hỗ trợ scale-out thật**
- **Tách luồng accept thành service độc lập**

:

XIN CẢM ƠN THẦY ĐÃ LẮNG NGHE