PHENIKAA UNIVERSITY

SCHOOL OF COMPUTING

Lab Report: Lab 04 – Microservices Decomposition & Communication
Microservice Architecture for Real-Time Chat Application using Event-Driven and gRPC

Student Details: Hoàng Lê Đức Huy - 23010298 - Group 8

Nguyễn Hà Nguyên - 23010310 - Group 8

Nguyễn Đức Minh - 23010302 - Group 8

Course Info: Software Architecture

Ha Noi, December 25, 2025

Table of Contents

List of Tables

List of Figures

1. Abstract

This report presents the design outcomes of Lab 04 – Microservices Decomposition & Communication, which focuses on transforming a realtime chat application from a monolithic architectural style into a microservices-based architecture.

The primary objective of this lab is to apply the principle of decomposition by business capability, define clear service boundaries, and establish explicit communication contracts among independent services. Based on the proposed realtime chat system, the application is decomposed into several domain-focused microservices, including User Service, Chat Service, Notification Service, API Gateway, and WebSocket Gateway.

Through this design, each microservice owns its data and communicates exclusively through well-defined HTTP APIs, gRPC interfaces, and asynchronous messaging mechanisms such as RabbitMQ. The resulting architecture improves scalability, modifiability, and maintainability, while effectively supporting real-time communication requirements.

This lab provides a foundational blueprint for implementing a distributed realtime chat system, ensuring loose coupling, independent deployment, and reliable inter-service communication.

2. Lab Specific Section: Microservices Decomposition & Communication

This section documents the three activity practices required in Lab 04, based on the realtime chat system defined by the project draft.

2.1 Activity Practice 1 – Microservices Decomposition

2.1.1 Decomposition by Business Capability

The realtime chat system is decomposed based on business capabilities, ensuring that each microservice represents a cohesive domain responsibility. This approach promotes autonomy, independent scaling, and clear data ownership.

Instead of grouping services by technical concerns, the system is divided according to user-related functionality, messaging functionality, notification delivery, and communication infrastructure.

2.1.2 Identified Business Capabilities and Microservices

Table 1: Business Capability–Based Microservice Decomposition for the Realtime Chat System

Business Capability	Microservice Name	Entities / Data Owned
User Management & Social Relationship	User Service	User, FriendRequest

Conversation & Messaging Management	Chat Service	Conversation, ConversationMember, Message
Notification Delivery	Notification Service	Notification
Client Request Routing	API Gateway	No persistent data
Realtime Communication Handling	WebSocket Gateway	Connection metadata, socket sessions
Persistent Data Storage	MongoDB	Service-specific databases

Each microservice maintains exclusive ownership of its data. Direct database access across services is strictly prohibited, enforcing microservice boundaries and preventing tight coupling.

2.1.3 External Systems and Dependencies

The system interacts with several external systems to support non-core functionalities:

Table 2: External Systems and Dependencies of the Realtime Chat Microservices System

External System	Purpose
SMTP Email Server	Sending email notifications (via Notification Service)
Identity Provider	User authentication and credential validation
Redis	Socket presence management, connection metadata, and caching support
Client Applications	Web or Mobile chat clients
RabbitMQ	Event-driven messaging between Chat Service and Realtime Gateway

2.2 Activity Practice 2 – Service Contracts

This section defines the public interfaces exposed by each microservice. All interactions between services occur through HTTP APIs, gRPC methods, or asynchronous messaging, without direct database access.

2.2.1 User Service – gRPC API

The User Service manages authentication, user profiles, and social relationships.

Table 3: gRPC Service Contracts of the User Service

Method	Input	Output	Description
register	UserRegisterRequest	UserRegisterResponse	Registers a new user account
login	LoginRequest	LoginResponse	Authenticates a user
makeFriend	MakeFriendRequest	MakeFriendResponse	Sends a friend request
updateStatusMakeFriend	UpdateStatusRequest	UpdateStatusResponse	Accepts or rejects a friend request
listFriends	ListFriendsRequest	ListFriendsResponse	Retrieves the friend list

Producer–Consumer Relationship:

- Producer: User Service
- Consumers: API Gateway, Chat Service (indirectly via user identity)

All authentication and friend management logic is centralized within this service. Authentication is handled at the API Gateway using JWT, and authenticated user context (userId) is propagated to downstream services via gRPC metadata.

2.2.2 Chat Service – gRPC API and Realtime Messaging

The Chat Service handles conversations, participants, and message persistence.

Table 4: gRPC Service Contracts of the Chat Service

Method	Input	Output	Description
createConversation	CreateConversationRequest	CreateConversationResponse	Creates a new conversation
addMemberToConversation	AddMemberRequest	AddMemberResponse	Adds a user to a conversation

getConversations	GetConversationsRequest	GetConversationsResponse	Lists conversations of a user
getMessagesByConversationId	GetMessagesRequest	MessageListResponse	Loads messages with pagination

Realtime Messaging:

- Messages are delivered to online users via WebSocket Gateway
- RabbitMQ is used as the message broker to publish message events, which are consumed by the Realtime Gateway

Producer–Consumer Relationship:

- Producer: Chat Service
- Consumers: WebSocket Gateway, Client Applications

2.2.3 Notification Service – gRPC API

The Notification Service is responsible for generating and delivering notifications, including email-based notifications.

Table 5: Service Contracts of the Notification Service

Method	Input	Output	Description
createNotification	CreateNotificationRequest	NotificationResponse	Creates a notification record
markAsRead	NotificationId	Acknowledgement	Marks notification as read

Integration:

- SMTP Email server for email delivery
- Triggered by events such as friend acceptance or system alerts

Producer–Consumer Relationship:

- Producer: User Service, Chat Service
- Consumer: Notification Service

Notifications are processed asynchronously to avoid blocking core workflows.

2.3 Activity Practice 3 – C4 Model (Level 1: System Context)

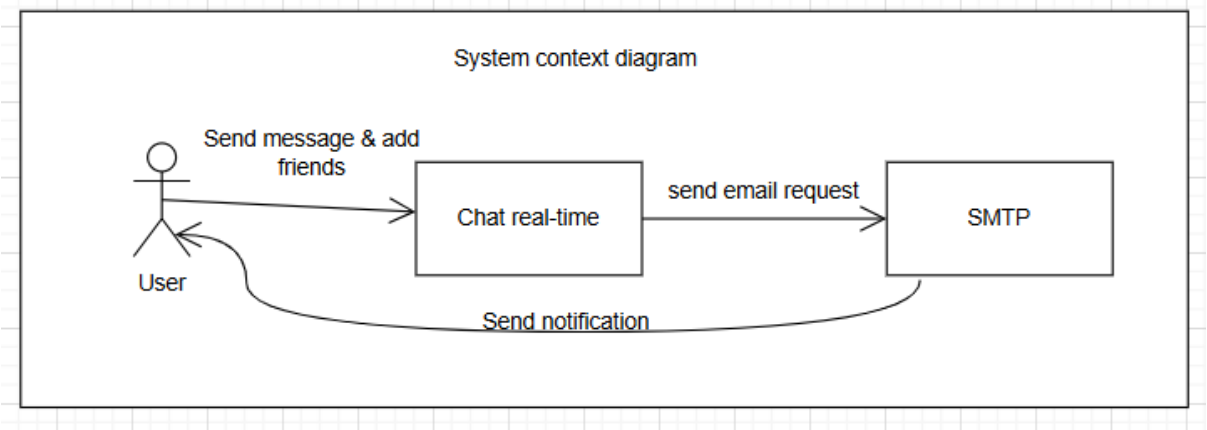

Figure 1: Level 1 Context

The system separates the API Gateway (HTTP + gRPC orchestration) and the Realtime Gateway (WebSocket handling), enabling independent scaling and clear responsibility boundaries.

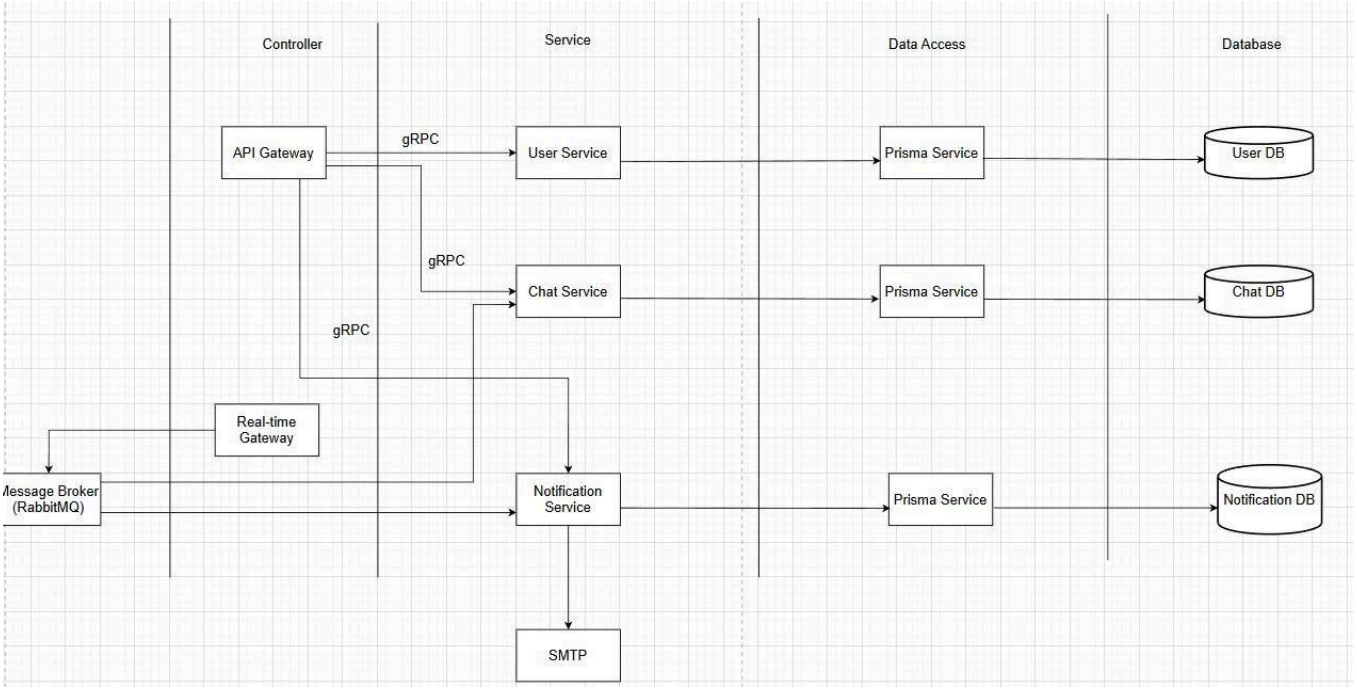

Figure 2: Level 2 Containers

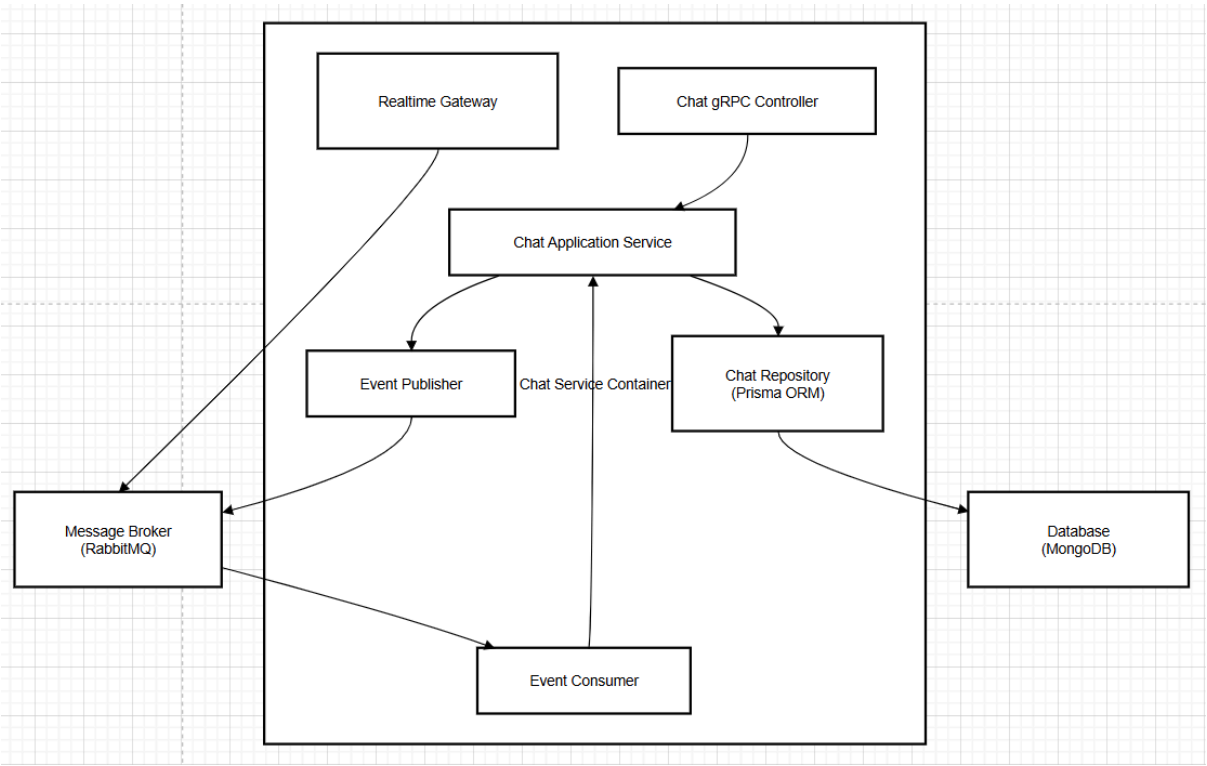

Figure 3: Level 3 Components (Chat Service Container)

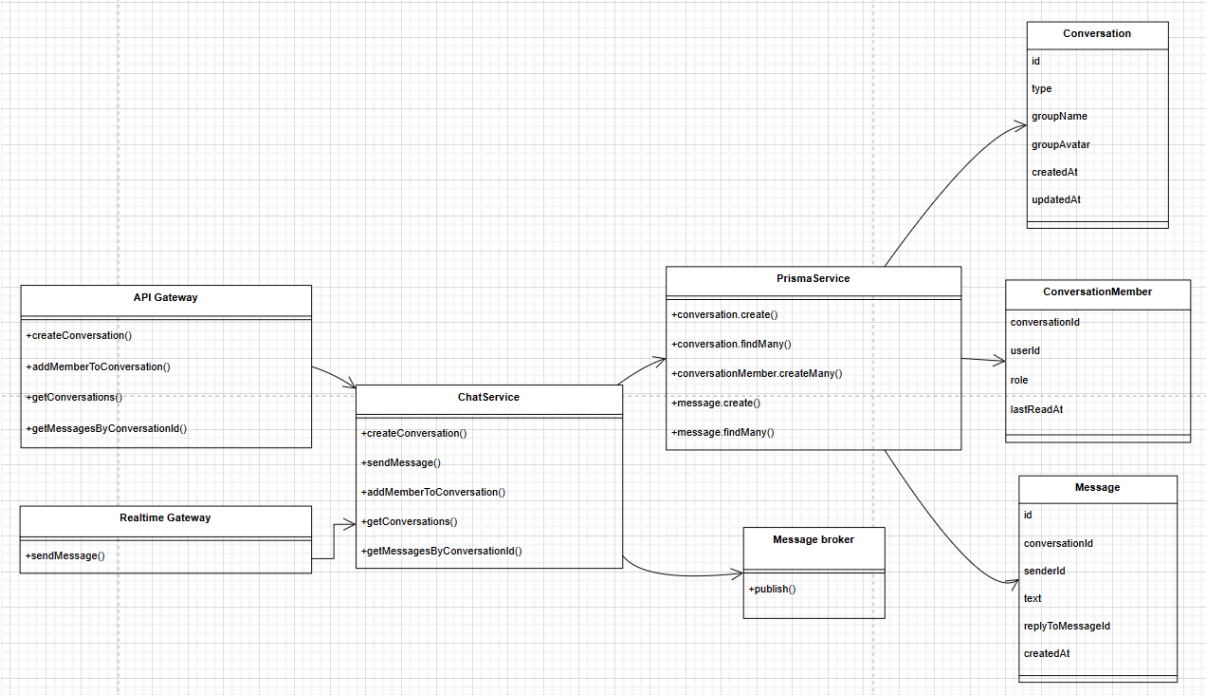

Figure 4: Level 4 Code

3. Conclusion & Reflection

Lab 04 successfully demonstrates the transformation of a realtime chat system into a well-structured microservices architecture. By applying decomposition by business capability, the system achieves clear service boundaries, independent data ownership, and flexible communication patterns.

The defined service contracts ensure loose coupling and prevent cross-service data dependencies. The integration of synchronous gRPC calls and asynchronous event-driven communication allows the system to balance responsiveness and scalability, which is critical for realtime applications.

This architectural blueprint establishes a solid foundation for future implementation phases, enabling independent development, deployment, and scaling of each microservice. The introduction of an event-driven messaging layer via RabbitMQ further decouples core chat logic from realtime delivery concerns.