PHENIKAA UNIVERSITY

SCHOOL OF COMPUTING

Lab Report: Lab 02 – Layered Architecture Design (Logical View)

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 13, 2025

Table of Contents

List of Tables

List of Figures

1. Abstract

This lab focuses on designing a Microservice Architecture for the Real-Time Chat Application. Building on the requirements analyzed in Lab 1, the system is analyzed and oriented towards a microservice-based architecture to improve scalability, maintainability, and performance. The microservices communicate internally using gRPC for high-speed RPC calls, while RabbitMQ is used for asynchronous, event-driven communication to decouple core messaging logic from real-time delivery. This report defines the service boundaries, identifies components, specifies gRPC interfaces and RabbitMQ events, and presents the overall microservice component diagram.

2. Lab Specific Section: II. Layered Architecture Design (Logical View)

2.1 Activity Practice 1: Defining Layers and Responsibilities

The Chat Application is logically decomposed into major functional layers and services, which later evolve into independent microservices. Each service owns its data and business logic, and communicates with others via gRPC or event messages.

2.1.1 Service Descriptions

Table 1: Microservices and their responsibilities

Service	Purpose / Responsibilities	Communication
API Gateway	Handles client HTTP requests, token validation, request routing.	HTTP
User Service	Manages user accounts, authentication, and online/offline status.	gRPC
Conversation Service	Manages conversations, membership, and access control.	gRPC
Message Service	Processes incoming messages, stores message data, and publishes events.	gRPC + RabbitMQ
Realtime Gateway	Handles WebSocket connections and real-time message delivery to clients.	WebSocket + RabbitMQ

2.1.2 Data Flow Between Services

Flow for “View Messages in a Conversation”:

1. Client sends GET /conversations/{conversationId}/messages to the API Gateway.
2. API Gateway calls MessageService.GetMessages(conversationId, userId) via gRPC.
3. Message Service calls ConversationService.IsMember(conversationId, userId) to validate access.

4. Message Service retrieves message records from its own database.
5. Message Service returns message data to the Gateway, which forwards it to the client.

Flow for “Send Message”:

1. Client sends SendMessage via WebSocket or REST.
2. Gateway calls MessageService.SendMessage() via gRPC.
3. Message Service stores the message.
4. Message Service publishes a RabbitMQ event message.created.
5. Realtime Gateway consumes the message.created event and delivers the message to connected clients via WebSocket.

2.2 Activity Practice 2: Component Identification

2.2.1 Components of Each Service

Table 2: Key components inside each microservice

Service	Component	Responsibilities
API Gateway	AuthMiddleware	Validates tokens for incoming requests
Realtime Gateway	WebSocketHub	Maintains real-time WebSocket connections
User Service	UserController (gRPC)	User CRUD and token verification
Conversation Service	ConversationController (gRPC)	Conversation membership checks and access control
Message Service	MessageController (gRPC)	Handles sending and retrieving messages
Message Service	RMQPublisher	Publishes message events to RabbitMQ

2.2.2 gRPC and RabbitMQ Interfaces

Table 3: gRPC and RabbitMQ interfaces

Interface	Type	Method Signature / Event Payload	Purpose
GetMessages(conversationId, userId)	gRPC	returns List<MessageDTO>	Retrieves messages in a conversation
SendMessage(MessageDTO)	gRPC	returns Ack	Sends and stores a new message

IsMember(conversationId, userId)	gRPC	returns bool	Validates user membership
message.created	RabbitMQ Event	{conversationId, senderId, content, timestamp}	Notifies new message creation
user.status.updated	RabbitMQ Event	{userId, status}	Updates user presence status

Message events are published by the Message Service and consumed by the Realtime Gateway to deliver real-time updates to clients.

2.3 Activity Practice 3: UML Component Diagram

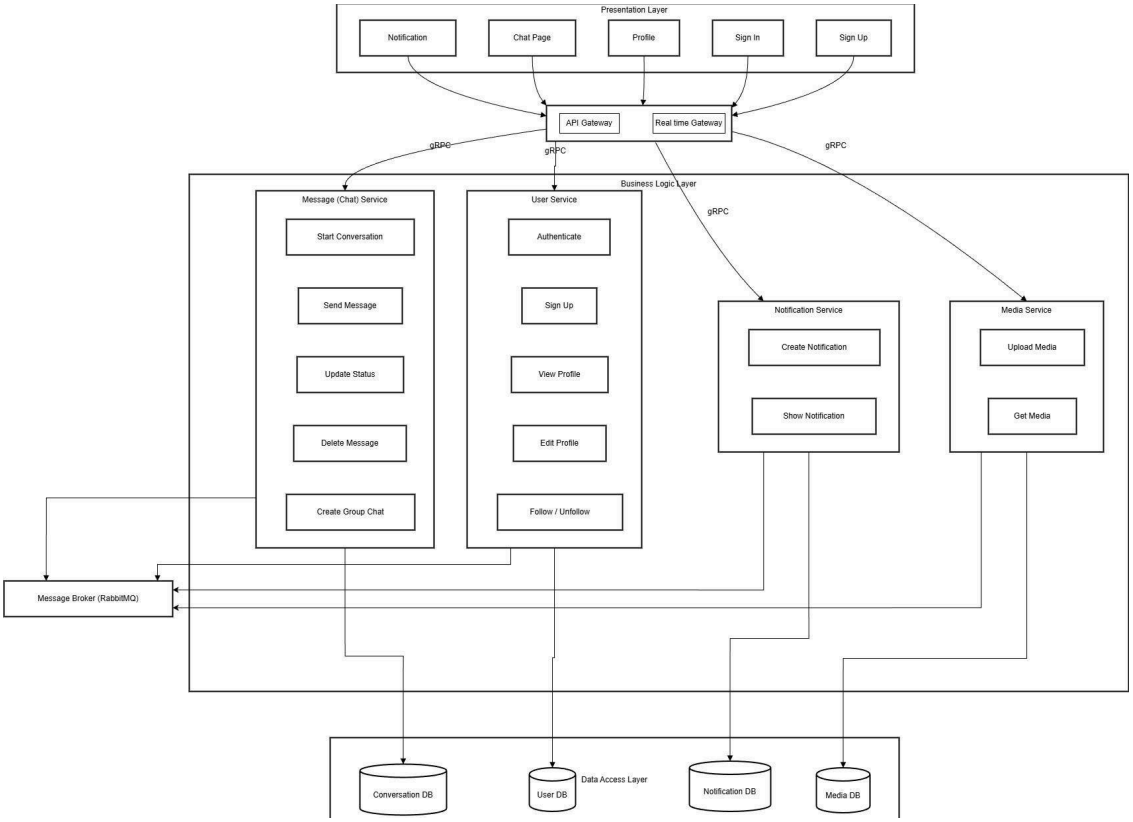

Figure 1: Microservice Component Diagram with API Gateway, Realtime Gateway, gRPC, and RabbitMQ

3. Architectural Design

3.1 Problem Statement

The system must support high-throughput real-time messaging, maintain low latency, and ensure scalability as the number of users and conversations grows. The architecture must allow independent development, deployment, and scaling of functionalities such as user management, conversation management, and messaging.

3.2 Rationale: Why Microservices + gRPC + RabbitMQ

- Microservices
    - Allows independent scaling (Message Service can scale horizontally).
    - Enables isolated failure handling and independent deployment.
    - Supports clear separation of responsibilities and domain boundaries.

- gRPC
    - High-performance RPC framework using HTTP/2 and Protobuf.
    - Lower latency and overhead compared to REST.
    - Ideal for inter-service communication in real-time systems.

- RabbitMQ
    - Handles asynchronous event-driven communication.
    - Offloads real-time broadcasting tasks from core services.
    - Ensures reliable delivery of chat events even under high load.

4. Conclusion and Reflection

Lab 2 redesigns the Chat Application from a layered model into a microservice architecture using gRPC and RabbitMQ. The resulting architecture offers strong scalability, efficient communication, and robust real-time message handling. This foundation enables more flexible development and will serve as the basis for implementation in Lab 3. The separation between API Gateway and Realtime Gateway ensures clean responsibility boundaries and enables independent scaling of real-time communication.