PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

WEEKLY REPORT

TOPIC: DEVELOPING A MOBILE MESSAGING APPLICATION

Course Class:	Software Architecture-1-2-25 (N01)	
Instructor:	M.Sc. Vũ Quang Dũng	
Group:	04	
Group Members:	Đỗ Trịnh Lệ Quyên	23010485
	Vũ Viết Huy	23010699
	Nguyễn Mạnh Cường	23010064

Ha Noi, December 27, 2025

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Problem Statement

Following the architectural design phase in Lab 2, where the team established the Layered Architecture for the "Lockmess" application, the next critical step is to define how the system components interact in detail. The challenge in this phase is to transition from a high-level logical view to a concrete component design that supports the project's key features: **Real-time Messaging** and **Intelligent Recommendations**. Specifically, we need to determine how the Mobile Client (Flutter) effectively communicates with the Infrastructure (Supabase) to ensure performance and data integrity without a traditional middleware server.

1.2 Project Objectives

The primary objective of Lab 4 is to decompose the monolithic application into distinct functional modules and establish clear communication protocols. Specifically, the team aims to:

- **Decompose the System:** Map business capabilities to specific Flutter feature modules and Supabase database tables.
- **Define Service Contracts:** Specify the Repository Interfaces (APIs) for critical features like Chat, Groups, and Recommendations.
- **Model Architecture:** Visualize the system using the **C4 Model** across four levels of detail (Context, Container, Component, Code).

2 SERVICE DECOMPOSITION & CONTRACTS

2.1 Architectural Style Selection

Instead of a traditional Microservices architecture which adds unnecessary complexity for this project scale, the team has adopted a **Client-Centric (Serverless)** architecture.

- **Frontend:** Flutter Mobile App (Handles all Business Logic and UI).
- **Backend-as-a-Service (BaaS):** Supabase (Handles Auth, Database, Storage, and Real-time events).

2.2 Decomposition by Business Capability

Based on the functional requirements defined in Lab 1 (FR-01 to FR-12), the system is decomposed into the following feature modules :

Business Capability	Client Module (Flutter Feature)	Data Owned (Supabase Tables)	Responsibility
1. Identity & Authenticatio n	features/login	auth.users, public.profiles.	Manages user registration, login, and secure token handling.
2. Profile Management	features/profile	public.profiles, public.hobbies	Handles personal info updates and interest tags (for recommendati ons).
3. Relationship Management	features/friends	public.friends, public.blocked _users	Manages friend requests, blocking, and contact lists.
4. Direct Messaging	features/chats	public.conversations, public.m essages	Handles real-time 1-on-1 chat, media attachment, and history.
5. Group &Community	features/group	public.groups, public.group_ members	Manages group creation, public channel discovery, and

			member roles.
6. Recommendation System	features/recommendation	(Aggregates data from profiles & groups)	Matches user interest tags with group topic tags to suggest communities.

Table 1: Decomposition by Business Capability

3 Service Contracts Definition (Repository Interfaces)

In the Clean Architecture approach, "Service Contracts" are defined as **Repository Interfaces** within the Domain Layer. These interfaces act as the strict boundary between the Application Logic and the External Data Source (Supabase).

3.1 Identity & Profile Contracts

- **IAuthRepository**
  - **Future<void> signIn(String email, String password):** Authenticate credentials.
  - **Future<void> signUp(String email, String password):** Register new user.
  - **User? getCurrentUser():** Retrieve current session.
- **IProfileRepository**
  - **Future<Profile> getProfile(String userId):** Fetch full user details.
  - **Future<void> updateInterests(List<String> tags):** Update tags for recommendations.

3.2 Chat & Real-time Contracts

- **IChatRepository**
  - **Future<void> sendMessage(Message msg):** Insert message into DB via RPC.
  - **Stream<List<Message>> getMessageStream(String chatId): (Real-time)** Listen for new messages using WebSockets.
  - **Future<void> uploadMedia(File file):** Upload image/video to Storage.

3.3 Group & Recommendation Contracts

- **IGroupRepository**
  - **Future<List<Group>> getPublicGroups():** Fetch groups where **is_public = true**.

        - **Future<void> joinGroup(String groupId):** Add user to group members.
- **IRecommendationRepository**
    - **Future<List<Group>> getRecommendedGroups(String userId):**
        - *Process:* Fetch User Tags -> Fetch Public Groups -> Run Matching Algorithm (Client-side) -> Return sorted list.

4 SYSTEM ARCHITECTURE & COMMUNICATION (C4 MODEL)

4.1 Level 1: System Context Diagram

Shows the high-level interactions between the User, the Lockmess App, and the Supabase Platform.

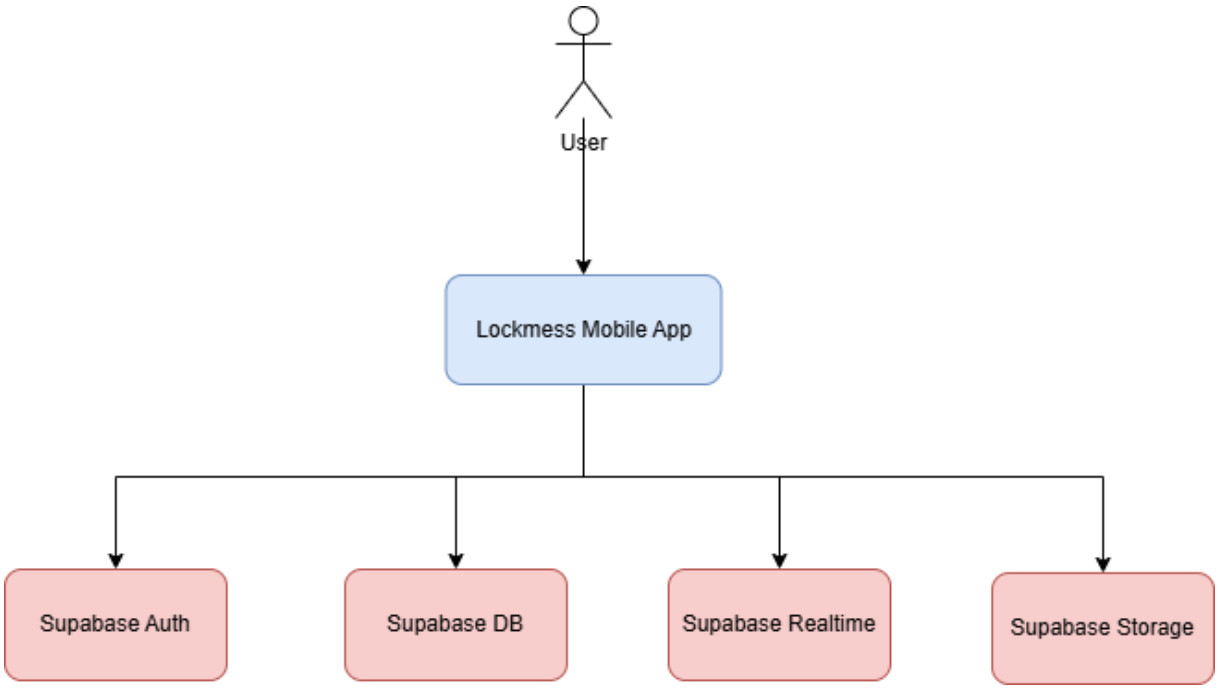

Figure 1: System Context Diagram

4.2 Level 2: Container Diagram

Zooming in to show the specific technology containers: Flutter App and Supabase Services.

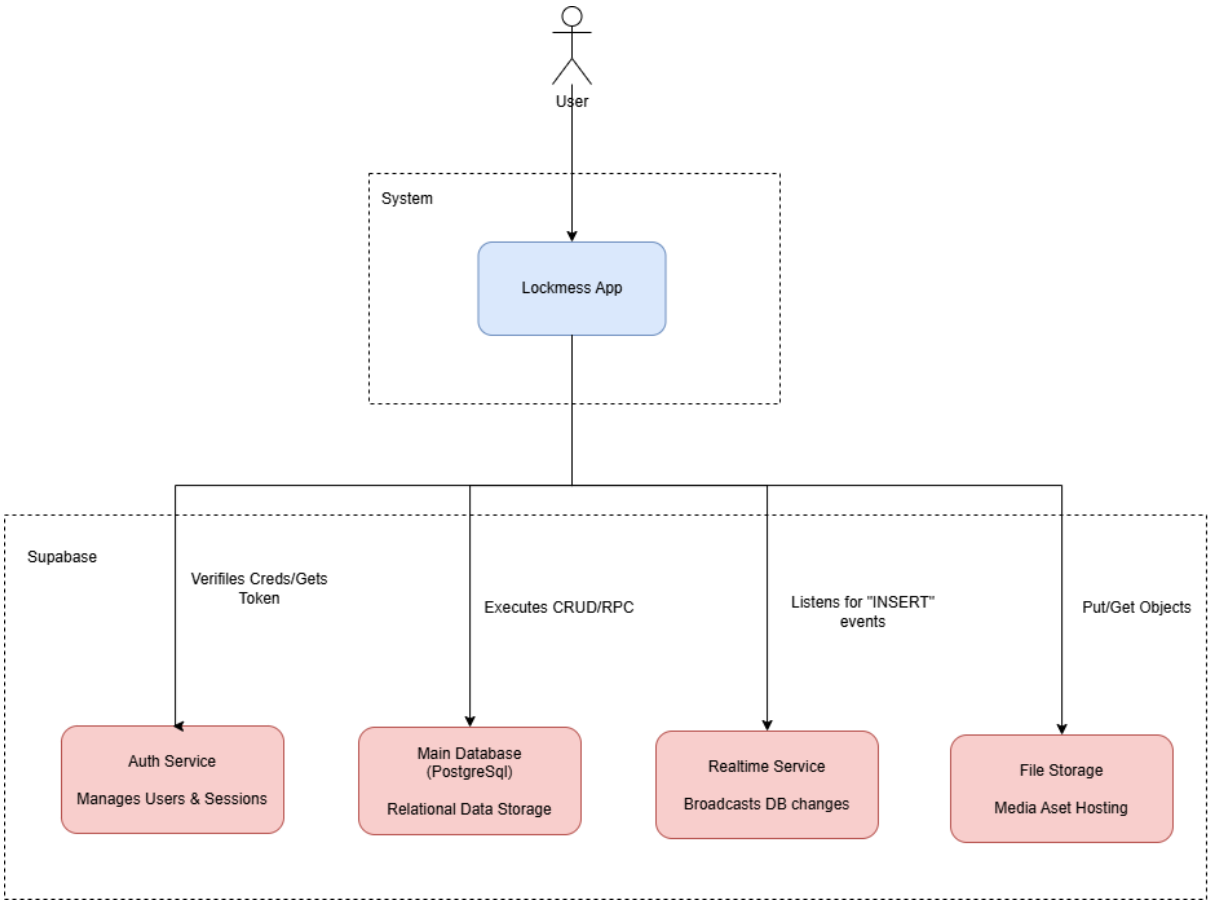

Figure 2: Container Diagram

4.3 Level 3: Component Diagram

Zooming into the Mobile App to show the Modular structure (mapping to lib/features).

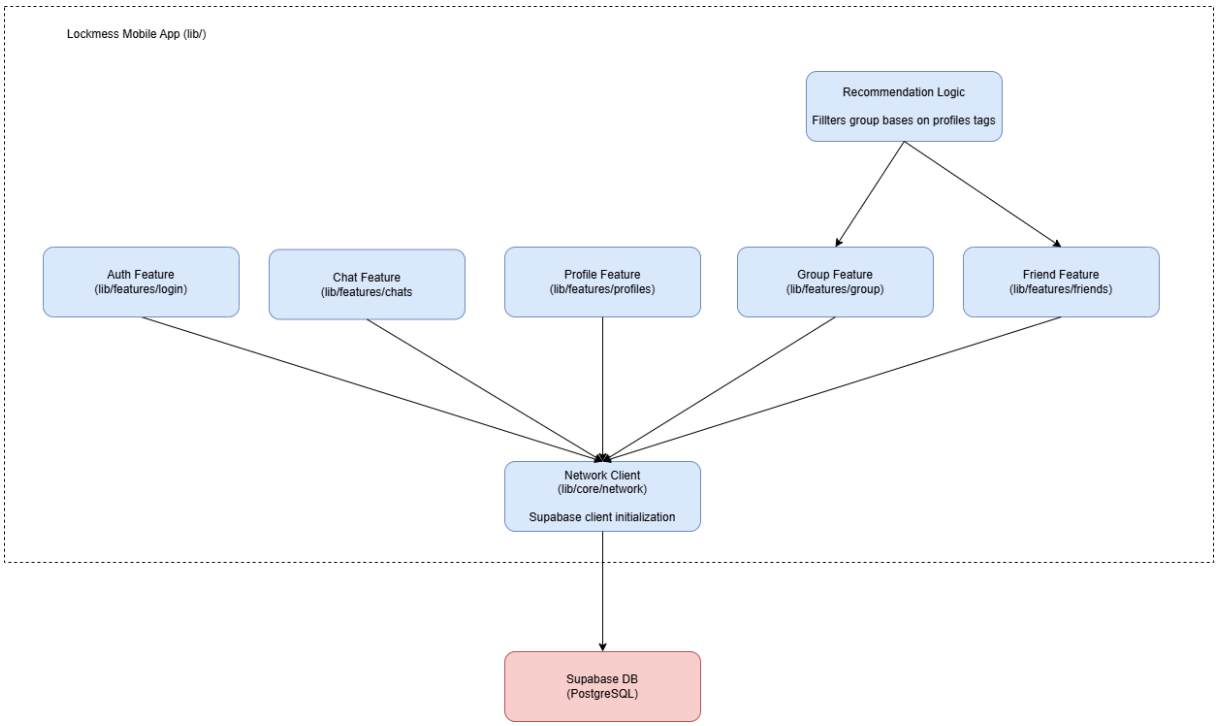

Figure 3: Component Diagram

4.4 Level 4: Code Diagram (Chat Feature)

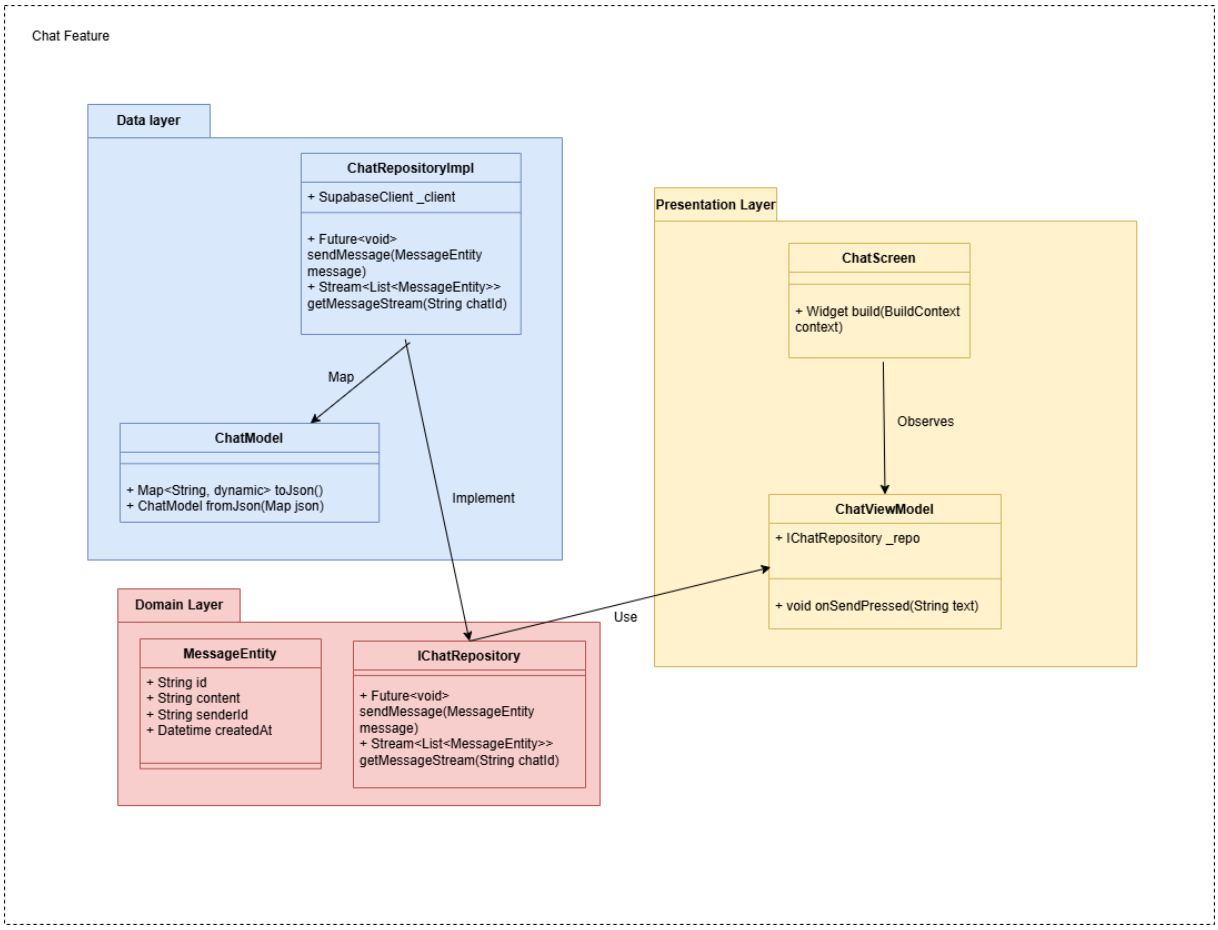

Figure 4: Class Diagram

4.5 Communication Strategy Analysis

To meet the Non-Functional Requirements (NFRs) for Performance and Reliability, we employ a hybrid strategy:

Interaction	Protocol	Communication Type	Rationale
1. Login / Register	HTTPS (REST)	Synchronous	Immediate feedback (Success/Fail) is required for the user to proceed.

2. Receiving Messages	Secure WebSockets (WSS)	Asynchronous	Required for Real-time performance (NFR-01). The app listens to Supabase Realtime channels instead of polling.
3. Sending Messages	HTTPS (RPC)	Synchronous	Ensures data durability (Persisted to DB) before confirming "Sent" status to the UI.
4. Recommendation Engine	Internal Function	Synchronous (Local)	The matching logic runs locally on the device CPU using cached data for instant results.

Table 2: Communication Straegy Analysis

5 CONCLUSION

Through Lab 4, Group 04 has successfully detailed the architectural design of the Lockmess application. By decomposing the system into clear Feature Modules (Auth, Chat, Group, Recommendation) and defining strict Repository Interfaces, we have ensured a high degree of **Separation of Concerns**.

The analysis of the Communication Strategy confirms that using **Supabase Realtime (WebSockets)** is the optimal solution to meet the critical Real-time Performance requirement (ASR-1), while standard REST calls handle transactional data effectively. The use of the C4 Model from Context down to Code level demonstrates a robust and scalable design ready for full implementation.