Problem Statement

Modern distributed systems require efficient, reliable communication for real-time messaging and notifications. Challenges in distributed programming include ensuring message consistency, managing communication across services, enabling independent scaling, and minimizing latency in real-time scenarios. This project addresses these challenges to build a robust **Chat app** by leveraging distributed programming principles.

Proposed Solution:

In this project, I particularly design and develop a full stack app with ***Service oriented architecture (SOA)***, where each service serves a different purpose and is decoupled from others. Each can be run separately, and one's failure will not halt the complete system. The communication between services happens via ***AMQP*** using ***RabbitMQ*** broker. I use a ***SQLite*** database with a ***repository pattern***. For real-time communication, I use Go's ***web sockets***, along with ***REST APIs*** for certain services.

Proposed Architecture:

1. **Authentication Service:** A distributed ***REST API*** ensuring secure login, registration, and JWT-based authentication, decoupled for independent scaling.
2. **Messaging Service:** Real-time ***WebSocket-based*** messaging backed by ***RabbitMQ*** for asynchronous queuing, ensuring reliable delivery across nodes.
3. **Notification Service:** Handles real-time notifications using ***RabbitMQ*** for decoupled, scalable delivery to multiple clients.
4. **API Gateway:** Acts as the centralized entry point for routing client requests to backend services, simplifying communication in a distributed environment.
5. **Database:** ***SQLite*** with the ***repository pattern*** to manage user data, messages, and contact lists, ensuring abstraction and scalability.
6. **Frontend**: Go html/templates

Features

- **Real-Time Messaging:** One-to-one chat with WebSocket support, ensuring low-latency communication.
- **Contact Management:** Handling of contact requests and approvals, along with contact search features.
- **File Transfer:** Support for chunk-based file sending within chat
- **User Management:**Login, registration, and logout functionalities
- **Chat UI:** Server-rendered interface for real-time interactions.

Tech Stack

- **Backend:** Go programming language for efficient and scalable service development.
- **Database:** SQLite with repository pattern for structured, abstracted data handling.
- **Frontend Templates:** Go templates for dynamic, server-rendered UI.
- **Message Broker:** RabbitMQ for asynchronous messaging and queue management.
- **Architecture:** Service-Oriented Architecture (SOA) for modular, distributed services.
- **Protocols:** WebSockets for real-time communication and REST APIs for auxiliary service interactions.