1. Problem Statement & Objectives

Problem Statement: In today's fast-paced digital world, users need an efficient and real-time communication platform that allows them to interact seamlessly. The lack of a user-friendly, secure, and scalable chat system can hinder communication and collaboration.

Objectives:

- Develop a web-based chat application for real-time messaging.
- Ensure security through authentication and data encryption.
- Support multiple users with seamless conversation handling.
- Implement a responsive and intuitive user interface.
- Provide a scalable backend to handle high user traffic.

2. Use Case Diagram & Descriptions

Actors:

- **User**: Registers, logs in, and participates in chats.
- **Admin**: Manages users, moderates content, and oversees system operations.

Use Cases:

1. **User Registration & Authentication –** Users sign up and log in securely.
2. **One-on-One Chat –** Users can send and receive private messages.
3. **Group Chat –** Users can participate in group conversations.
4. **Message History –** Users can view chat history.
5. **File Sharing –** Users can send images, documents, and other files.
6. **Notification System –** Users receive notifications for new messages.
7. **Admin Moderation –** Admin can monitor and control conversations if necessary.

3. Functional & Non-Functional Requirements

Functional Requirements:

- Users should be able to register, log in, and log out.
- Users can send and receive messages in real-time.
- Support for private and group chats.

- Ability to send media files (images, documents, etc.).
- Search functionality for past messages.
- Notifications for new messages and friend requests.
- Admin can moderate messages and manage users.

Non-Functional Requirements:

- The system should ensure high availability and scalability.
- Messages should be delivered with minimal latency.
- User data must be encrypted for privacy and security.
- The system should have a responsive UI across different devices.
- Ensure a seamless user experience with minimal downtime.

4. Software Architecture

Architecture Style: Model-View-Controller (MVC) pattern

Components:

1. **Frontend (Client-Side)**
    a. Developed using React.js or Vue.js.
    b. Uses WebSockets for real-time messaging.
    c. Responsive design for mobile and desktop users.
2. **Backend (Server-Side)**
    a. Built with Node.js and Express.js.
    b. Handles authentication, message routing, and database interactions.
3. **Database Layer**
    a. Uses MongoDB or PostgreSQL to store user data and chat history.
    b. Indexed for fast message retrieval.
4. **Real-Time Communication**
    a. WebSockets or Firebase for instant message delivery.
5. **Security Measures**
    a. JWT authentication for secure user sessions.
    b. Data encryption for stored messages and user information.
    c. Secure API endpoints.
6. **Deployment & Scalability**
    a. Cloud hosting on AWS/GCP.
    b. Load balancer for handling multiple users.

c. CDN for faster media file delivery.

5. Database Design & Data Modeling

ER Diagram (Entity-Relationship Diagram): A well-defined ER diagram illustrating the relationships between system entities:

- **User** (UserID, Name, Email, Password, Status, Profile Picture)
- **Chat** (ChatID, ChatType, CreatedAt)
- **Message** (MessageID, ChatID, SenderID, Content, Timestamp, MessageType)
- **File** (FileID, MessageID, FilePath, FileType)
- **Notification** (NotificationID, UserID, Type, Message, Timestamp)
- **Admin** (AdminID, UserID, Role)

Logical & Physical Schema:

1. **Users Table**
   a. UserID (Primary Key, Auto Increment)
   b. Name (VARCHAR, Not Null)
   c. Email (VARCHAR, Unique, Not Null)
   d. Password (VARCHAR, Not Null)
   e. Status (VARCHAR, Default: 'Online')
   f. ProfilePicture (VARCHAR, Nullable)
2. **Chats Table**
   a. ChatID (Primary Key, Auto Increment)
   b. ChatType (ENUM: 'Private', 'Group')
   c. CreatedAt (TIMESTAMP, Default: CURRENT_TIMESTAMP)
3. **Messages Table**
   a. MessageID (Primary Key, Auto Increment)
   b. ChatID (Foreign Key -> Chats.ChatID)
   c. SenderID (Foreign Key -> Users.UserID)
   d. Content (TEXT, Nullable)
   e. Timestamp (TIMESTAMP, Default: CURRENT_TIMESTAMP)
   f. MessageType (ENUM: 'Text', 'Image', 'File')
4. **Files Table**
   a. FileID (Primary Key, Auto Increment)
   b. MessageID (Foreign Key -> Messages.MessageID)
   c. FilePath (VARCHAR, Not Null)

    d. FileType (ENUM: 'Image', 'Document', 'Other')

5. **Notifications Table**
    a. NotificationID (Primary Key, Auto Increment)
    b. UserID (Foreign Key -> Users.UserID)
    c. Type (ENUM: 'Message', 'Friend Request')
    d. Message (TEXT, Not Null)
    e. Timestamp (TIMESTAMP, Default: CURRENT_TIMESTAMP)
6. **Admins Table**
    a. AdminID (Primary Key, Auto Increment)
    b. UserID (Foreign Key -> Users.UserID)
    c. Role (VARCHAR, Not Null)

6. Data Flow & System Behavior

DFD (Data Flow Diagram):

- Context-level and detailed diagrams showing how data moves through the system.

Sequence Diagrams:

- Process flow representation of key interactions between components.

Activity Diagram:

- Visualizing the workflow of processes or user actions within the system.

State Diagram:

- Represents different states of an object and how it transitions between them.

Class Diagram:

- Defines the structure of the system by showing classes, attributes, methods, and relationships.

This structured approach ensures the system is well-documented, optimized, and scalable for real-time chat performance.