Experiment No: 5

Student Name: Rashid Khan
UID: 23BCS12551
Branch: B.E./C.S.E.
Section/Group: KRG-3B
Semester: 6th
Date of Performance: 17/02/2026
Subject Name: System Design
Subject Code: 23CSH-314

1. **Aim:** Design of a Scalable Messenger Application (similar to WhatsApp or Facebook Messenger)

2. **Objective:**

- Enable user registration and login with secure authentication (JWT-based session management).
- Implement real-time messaging using WebSocket connections.
- Support one-to-one messaging with message routing and delivery tracking.
- Integrate Redis/Kafka for message streaming and efficient data flow.
- Implement message search functionality using Elasticsearch.

3. **Tools Required:**

- **React / Flutter** – For building the client (user interface).
- **Node.js / Spring Boot** – For backend service development.
- **JWT** – For secure authentication and session management.
- **WebSocket** – For real-time messaging communication.
- **Nginx (Load Balancer)** – For distributing traffic across servers.
- **MySQL** – For storing user data.
- **PostgreSQL** – For storing group data.
- **MongoDB / Cassandra** – For storing chat messages.
- **Redis** – For caching and tracking online users.
- **Kafka** – For message streaming and queue handling.
- **Amazon S3** – For storing images and media files.
- **Elasticsearch** – For fast message search.
- **Firebase Cloud Messaging (FCM)** – For Android push notifications.
- **APNS** – For iOS push notifications.
- **Docker** – For containerization and deployment.
- **Kubernetes** – For scaling and managing services.

4. SYSTEM DESIGN / SYSTEM SPECIFICATION

4.1 Functional Requirements

- Client should be able to register and login himself.
- System should support one-to-one messaging.
- System should also support, group messaging as well.
- System should support sending message in both texual as well as media messages.
- System should support preserving message history.
- Client should be able to see delivered messages (read receipts)

4.2 Non-Functional Requirements

- Target Scale: 1 Billion users having 100msgs per day.
- CAP Theorem: High Availability (Eventual Consistency).
- Latency: Approx 200-300 ms.
- System should be highly reliable, there should not be any packet loss or message drop.

4.3 Core Entities of the System

- Users
- Groups
- Messages

5. API Endpoints:

1. Register a User

- **POST Call:** https://www.whatsapp.com/user/register

```
{
  "userName": "Rashid Khan",
  "email": "rashid@gmail.com",
  "phoneNumber": "9876543210",
  "password": "123456"
}
```

- Other CRUD Operations:

  GET: /user/{user_ID} → Get user details

  PUT: /user/{user_ID} → Update user profile

  DELETE: /user/{user_ID} → Delete user account

2. Login

- **POST Call:** https://www.whatsapp.com/user/login

```
{
  "email": "rashid@gmail.com",
  "password": "123456"
}
```

- **Response:** Returns JWT token for authentication

3. One-to-One Message

- **WS:** /messages/send
  (WS = WebSocket for real-time messaging)

  ```
  {
    "senderId": 101,
    "receiverId": 202,
    "message": "Hello!"
  }
  ```

- **GET:** /chat/{user_ID} → List<Chat> (Pagination)
  [List of all chats of the user]
- **GET:** /messages/{user_ID}/{receiver_ID}
  [When user clicks a chat → show all messages between both users]

4. Group Messages

- **Create a Group**
  POST: /groups/create

  ```
  {
    "groupName": "Project Team",
    "createdBy": 101
  }
  ```

- **Add Member**
  POST: /groups/{group_ID}/add

  ```
  {
    "userId": 202
  }
  ```

- **Remove Member**
  DELETE: /groups/{group_ID}/remove

  ```
  {
    "userId": 202
  }
  ```

- **WS:** /messages/send
  (Used for sending group messages in real-time)
- **GET:** /groups/{group_ID} → List<Chat> (Pagination)
  [Shows all messages of that group]

6. HLD (High Level Design):