System Architecture for Anonymous Group Chat Web Application

1. Introduction

This document outlines the **System Architecture** for the **Anonymous Group Chat Web Application**. It describes the key components, their interactions, and the overall data flow of the system to ensure **scalability, security, and real-time communication**.

2. System Overview

The application follows a **Client-Server Architecture** with real-time messaging capabilities. It consists of three main layers:

1. **Frontend (Client-Side UI)** – Handles user interactions.
2. **Backend (Server-Side Logic & API)** – Manages requests, authentication, and chat processing.
3. **Database & Storage** – Stores chat messages, user data, and group information.

3. System Components & Technologies

3.1 Frontend (Client-Side)

- **Technology:** React.js / HTML, CSS, JavaScript
- **Role:** Displays the chat interface, user groups, and handles real-time updates.
- **Key Functions:**
  - User joins/leaves chat groups.
  - Displays real-time messages using WebSockets.
  - Provides options for sending messages, images, and polls.
  - Allows group creators to manage participants.

3.2 Backend (Server-Side)

- **Technology:** Node.js with Express.js
- **Role:** Handles all server-side logic, processes API requests, and manages chat functionality.
- **Key Functions:**
  - User authentication (JWT for group creators).
  - Handles real-time messaging via **Socket.io**.
  - API endpoints for creating/joining groups, sending messages, and managing users.
  - Ensures **end-to-end encryption** for message security.

3.3 Database (Data Storage Layer)

- **Technology:** MySQL (Structured Data)
- **Role:** Stores user details, chat groups, and message history.
- **Key Tables:**
  - `Users` – Stores group creators and authentication details.
  - `Groups` – Stores group metadata (creator, max members, etc.).
  - `Messages` – Stores text messages, timestamps, and sender details.
  - `Polls` – Stores created polls and user responses.

3.4 Real-Time Messaging System

- **Technology:** Socket.io (WebSockets for real-time data exchange)
- **Role:** Enables real-time, low-latency communication between users.
- **Key Functions:**
  - Listens for incoming messages and broadcasts them to the group.
  - Updates chat history in the database asynchronously.

3.5 Security & Encryption

- **Authentication:** JWT-based authentication for group creators.
- **Message Encryption:** End-to-end encryption for messages.
- **Data Privacy:** No logs for anonymous users; only group creators can access stored chat history.
- **Firewall & DDoS Protection:** Ensures security against unauthorized access.

4. System Workflow & Data Flow

Step 1: User Access & Group Creation

1. User opens the chat platform.
2. If a user is a **Group Creator**, they log in and create a new group.
3. A **unique invite link** is generated and shared.

Step 2: Anonymous User Joins a Group

1. A participant joins via the invite link without logging in.
2. A **temporary anonymous identifier** (e.g., User1234) is assigned.

Step 3: Sending & Receiving Messages

1. A user sends a message via the frontend.
2. The message is sent to the backend via **Socket.io**.
3. The backend processes and **encrypts** the message.
4. The message is stored in the database (only for group creators).
5. The message is **broadcasted in real-time** to all participants.

Step 4: Exiting & Data Retention

1. A participant leaves the group (their messages are not saved).
2. Group creators retain chat history for moderation purposes.
3. Temporary user IDs are deleted upon session exit.

5. Deployment & Hosting

Component	Hosting Service
Frontend	Vercel / Netlify
Backend API	Render / Heroku
Database	MySQL Cloud (e.g., PlanetScale, AWS RDS)
WebSocket Messaging	Socket.io Server