# System Design Documentation:

## 1. Introduction

The Real-Time Chat Application is a platform that facilitates real-time communication between users. It includes features like real-time messaging, responsive UI, user registration and authentication, group chat creation, and real-time typing indicators.

### 1.1 Scope

The application enables users to register, log in, communicate in real-time, and create group chats for collaborative discussions.

## 2. Architectural Overview

### 2.1 High-Level Architecture

The system follows a client-server architecture:

* Client Side: ReactJS with Socket.io for real-time communication and Chakra UI for the responsive UI.
* Server Side: Node.js and Express.js with Socket.io for managing real-time messaging.
* Database: MongoDB for storing user data and chat history.

### 2.2 Key Components

* Frontend (Client Side): ReactJS with Chakra UI for the user interface.
* Backend (Server Side): Node.js and Express.js for handling requests and Socket.io for real-time communication.
* Database (Data Layer): MongoDB for storing user information and chat data.

### 2.3 Data Flow

User interactions on the client-side trigger requests to the server. The server processes the requests and communicates updates in real-time via Socket.io to relevant clients.

### 2.4 Technologies Used

* ReactJS: Provides a dynamic and responsive user interface for seamless user experience.
* Chakra UI: A component library for React to design responsive and accessible UI.
* Node.js and Express.js: Efficiently handles HTTP requests and WebSocket connections for real-time updates.
* Socket.io: Enables real-time bidirectional communication between clients and the server.
* MongoDB: A NoSQL database for storing user data and chat history.

## 3. Detailed Design

### 3.1 Frontend (ReactJS)

#### 3.1.1 Design Considerations

ReactJS was chosen for its component-based architecture, facilitating dynamic UI updates and seamless user interactions.

#### 3.1.2 Dependencies and Libraries

* React Router: For handling client-side routing within the application.
* Chakra UI: Provides a flexible and customizable component library for building the UI.
* Socket.io-client: Enables WebSocket connections to the server for real-time communication.

### 3.2 Backend (Node.js and Express.js)

#### 3.2.1 Design Considerations

Node.js and Express.js were chosen for their efficiency in handling HTTP requests and managing WebSocket connections for real-time messaging.

#### 3.2.2 Dependencies and Libraries

* Express: A minimal and flexible Node.js web application framework.
* Socket.io: Enables real-time bidirectional communication with clients.
* JSON Web Token: For user authentication and creating JSON Web Tokens.

### 3.3 Database (MongoDB)

#### 3.3.1 Design Considerations

MongoDB was chosen for its flexibility in storing and querying unstructured data, making it suitable for chat history and user data.

#### 3.3.2 Dependencies and Libraries

* Mongoose: An ODM (Object-Document Mapper) for MongoDB, simplifying database operations.

## 4. Data Management

### 4.1 Data Storage

MongoDB is used to store user profiles, chat history, and related data.

## 5. Security

### 5.1 Authentication and Authorization

JSON Web Tokens (JWT) are used for user authentication, ensuring secure access to the system.

### 5.2 Data Encryption

Sensitive user data is encrypted using industry-standard encryption algorithms.

## Setup and Run Instructions

To set up and run the prototype of the Real-Time Chat Application, follow these steps:

* a. Navigate to the repository: cd bitchat.  
  b. Set up a MongoDB database (e.g., using MongoDB Atlas) and obtain the connection URI.  
  c. Update the connection URI in the backend configuration. Create a .env file and include the credentials:

PORT=5000

MONGO\_URI="MONGO\_CONNECTION\_URL"

* d. Install dependencies: npm install. #server dependencies
* e. Navigate to the client end: cd client
* f. Install dependencies: npm install. #client dependencies
* g. Open two terminals and run both commands separately:

npm run client

npm start

### Access the Application: Open a web browser and navigate to http://localhost:3000 to access this Chat Application.