# System Design Document: QChat Messaging Service Prototype

## Project Title: QChat

### Table of Contents

1. [Introduction](#Xe3d0fc0bea9a42ce7605565d0964033d7f6ee47)
2. [System Architecture](#Xadffa0c657789d188b7bad67251e99502d9d94d)
3. [Technologies Used](#X956fb656d66dbbc0beb72b5d9ea5cb5b3c4b617)
4. [Database Design](#Xe7d51658d8f2379c0969f4580b5bec6ca5fc228)
5. [User Authentication and Authorization](#X9a30c72e50ed78edd672b6c3c1661d52bf8153c)
6. [Real-Time Messaging](#Xa37a79c7c6875f4fc8358a411515668a878ab39)
7. [User Interface](#X70cb1363718d2a5dec7a5356f80378e78e20c18)
8. [Notifications](#X9520674923d9734e2e3b9830818978910ff20e1)
9. [Security and Privacy](#Xffa8541f91d87b76a8ff69509e89d335e0a7e1f)
10. [Deployment](#Xaf1235adaeaaedd33fb0fd7d1acf40cb35a4e49)

### 1. Introduction

#### Purpose of the Document

This document outlines the design and architecture of the Messaging Service Prototype developed by Sarthak Choudhary for the internship opportunity with ImBesideYou. The prototype, named “QChat,” aims to create a basic messaging service with real-time messaging capabilities, user authentication, and a user-friendly interface.

#### Scope of the Messaging Service Prototype

The scope of the prototype includes the following key features: - Real-time messaging using WebSocket and Socket.io - User authentication with JWT - User-friendly interface using React and Chakra UI - MongoDB for data storage

#### Developer Information

* Developer: Sarthak Choudhary

#### Deployment Link

* The prototype is deployed on Render.com and can be accessed at <https://qchat-bu1g.onrender.com>.

### 2. System Architecture

#### High-Level Architecture

The system follows a client-server architecture where the client is a React-based web application, and the server is implemented using Node.js and Express.js. Socket.io is used for real-time communication between clients and the server.

#### Frontend Architecture

* **Frontend Framework:** React
* **UI Framework:** Chakra UI
* **WebSocket Implementation:** Socket.io-client for real-time messaging
* **Routing:** React Router

#### Backend Architecture

* **Backend Framework:** Node.js with Express.js
* **Authentication:** JWT (JSON Web Tokens)
* **Database:** MongoDB
* **WebSocket Server:** Socket.io

### 3. Technologies Used

#### Programming Languages

* JavaScript (Node.js and React)

#### Backend Dependencies

* bcryptjs for password hashing
* dotenv for environment variable management
* express for web server
* express-async-handler for handling asynchronous operations
* jsonwebtoken for JWT authentication
* mongoose for MongoDB database interaction
* nodemon for development server auto-reloading
* socket.io for real-time WebSocket communication

#### Frontend Dependencies

* @chakra-ui/icons for Chakra UI icon components
* @chakra-ui/react for Chakra UI component library
* @emotion/react for Emotion styling library
* @emotion/styled for styled components with Emotion
* axios for making HTTP requests
* react for building user interfaces
* react-chips for creating chip input components
* react-dom for React DOM rendering
* react-lottie for animation rendering
* react-notification-badge for displaying notification badges
* react-router-dom for client-side routing
* react-scripts for development scripts
* react-scrollable-feed for scrollable feed components
* socket.io-client for WebSocket client-side implementation

#### Database

* MongoDB: A NoSQL database used for storing user data, conversations, and messages.

### 4. Database Design

#### Database Schema

The database includes three main models: User, Chat, and Message. These models are linked through ObjectIds to establish relationships between them.

##### User Model

* Fields:
  + name: User’s name (String, required)
  + email: User’s email (String, unique, required)
  + password: User’s hashed password (String, required)
  + pic: User’s profile picture URL (String, required, default)
  + isAdmin: User’s admin status (Boolean, required, default: false)

##### Chat Model

* Fields:
  + chatName: Name of the chat (String, trim)
  + isGroupChat: Indicates if it’s a group chat (Boolean, default: false)
  + users: Array of user ObjectIds participating in the chat
  + latestMessage: ObjectId of the latest message in the chat
  + groupAdmin: ObjectId of the chat’s admin user

##### Message Model

* Fields:
  + sender: ObjectId of the message sender (ref: “User”)
  + content: Message content (String, trim)
  + chat: ObjectId of the chat to which the message belongs (ref: “Chat”)
  + readBy: Array of user ObjectIds who have read the message (ref: “User”)

### 5. User Authentication and Authorization

#### Authentication Mechanisms

* User authentication is implemented using JWT (JSON Web Tokens). Users obtain tokens upon successful login.

#### Authorization Levels

* The system distinguishes between “regular users” and “admins” with varying levels of access:
  + **Regular Users:** These users have standard access to the messaging service, including user registration, sending and receiving text messages between users, and group chat functionality.
  + **Admins:** Admin users have elevated privileges, such as managing group chats and potentially moderating user interactions.

#### Authentication Middleware

* Authentication middleware is implemented using the authMiddleware function to protect routes that require user authentication.

#### Route Protection

* The protect function is used in route definitions to protect routes that require authentication.

### 6. Real-Time Messaging

#### Messaging Protocol

* WebSocket is used for real-time messaging.
* Socket.io facilitates WebSocket communication between clients and the server.
* Real-time message updates are a core feature, allowing users to send and receive text messages in real-time.

### 7. User Interface

#### User Registration and Authentication

* User registration and authentication features are integrated into the user interface.

#### Sending and Receiving Text Messages

* The user interface enables users to send and receive text messages between each other in real-time.

#### Group Chat Functionality

* Users can create and participate in group chats through the user interface.

### 8. Notifications

#### Real-Time Message Notifications

* Users receive real-time message notifications through WebSocket and Socket.io, ensuring that they are instantly notified when new messages arrive.
* The notification system enhances the user experience by providing immediate updates on incoming messages.

### 9. Security and Privacy

#### Data Security Measures

* Password Hashing: User passwords are securely hashed using bcrypt before storing them in the database, ensuring that plain text passwords are not stored.
* JWT Authentication: User authentication is implemented using JSON Web Tokens (JWT), which are securely signed and verified to ensure the authenticity of user requests.

### 10. Deployment

Online deployed prototype: <https://qchat-bu1g.onrender.com>

#### Prerequisites

* Node.js and npm installed locally.

#### Setup Instructions

1. **Clone the Repository:**
   * Clone the QChat repository from <https://github.com/Cenult/QChat>
2. **Backend Setup:**
   * In root directory, run ‘npm install’ to install backend dependencies.
   * Create a .env file with the following environment variables:
     + PORT=4000
     + DATABASE\_URL=MongoDB\_URL
     + JWT\_SECRET\_KEY=<your-secret-key>
     + NODE\_ENV=development
3. **Frontend Setup:**
   * Navigate to the frontend directory.
   * Run ‘npm install’ to install frontend dependencies.
   * Alternatively – run ‘npm run build’ in root directory to install all dependencies.
4. **Start the Application:**
   * In the root directory, run ‘npm start’ to start the Node.js server.
   * In the frontend directory, run ‘npm start’ to start the React frontend.
5. **Access the Application:**
   * Open your web browser and navigate to http://localhost:3000 to access the QChat messaging service.
6. **Deployment Instructions (Production):**
   * When deploying in a production environment, set NODE\_ENV=production in the .env file.

### 11. Conclusion

This System Design Document provides an overview of the QChat Messaging Service Prototype, detailing its architecture, technologies used, database design, authentication, real-time messaging, user interface, notifications, security measures, and deployment instructions. The prototype aims to deliver a user-friendly and real-time messaging experience for users.

For more information, refer to the project’s GitHub repository and README.md file.