System Design Document: MERN Stack Chat Application with Socket.IO

Project Maker : Harshit Swarnkar

Table of Contents

1. Introduction

This System Design Document outlines the architecture and functionality of a real-time chat application built on the MERN (MongoDB, Express, React, Node.js) stack with Socket.IO. The application allows users to register, log in, and send text messages to other users within the platform.

2. System Overview

The MERN Stack Chat Application is a web-based chat platform that offers the following core features:

- **User Registration:** Users can create accounts by providing their credentials, including a unique username and password.
- **User Authentication:** Registered users can log in securely using their credentials.
- **Real-time Messaging:** Users can send and receive text messages in real-time, enabling instant communication.
- **Group Chat:** Users can create and participate in group chats with multiple members.
- **User Presence:** The application displays the online/offline status of users, allowing for real-time presence information.

3. Architecture

3.1. Frontend

The frontend of the application is built using React. Key components include:

- **User Interface** : The user interface is designed to provide a seamless and intuitive chat experience. Users can view their messages, send messages, and interact with their contacts.
- **State Management** : React's state management or state container (e.g., Redux) is used to manage the application's state, including user authentication and chat messages.
- **Socket.IO Client** : Socket.IO is integrated into the frontend to facilitate real-time communication. It handles events like message sending and receiving.

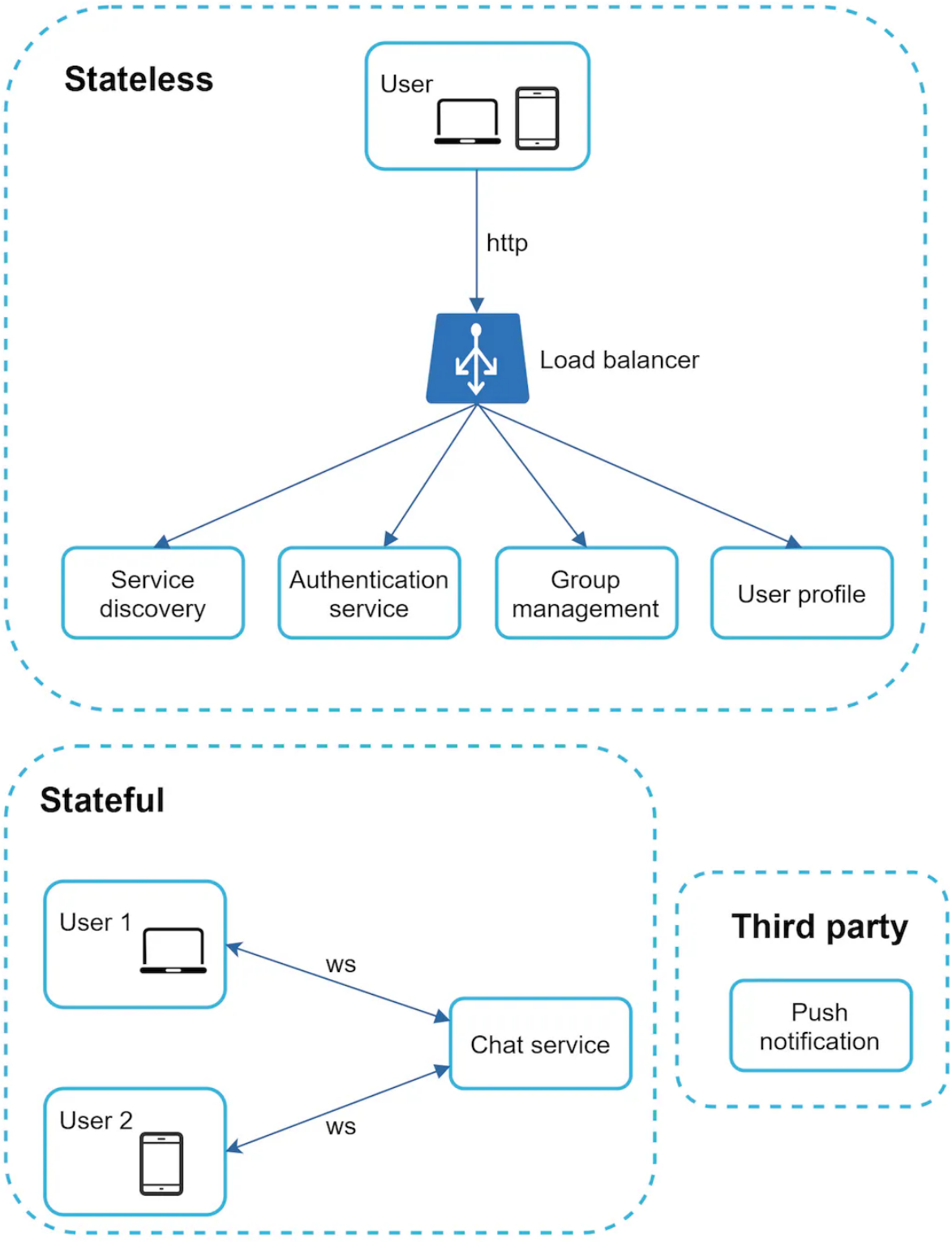
3.2. Backend

The backend is built using Node.js and Express, providing a RESTful API for user authentication and real-time chat functionality. Key components include:

- **User Authentication Middleware :** This component is responsible for user registration, login, and token-based authentication using technologies like JSON Web Tokens (JWT).

- **MongoDB Database** : MongoDB is used to store user data, chat messages, and other application data. The database schema will be detailed in the following section.

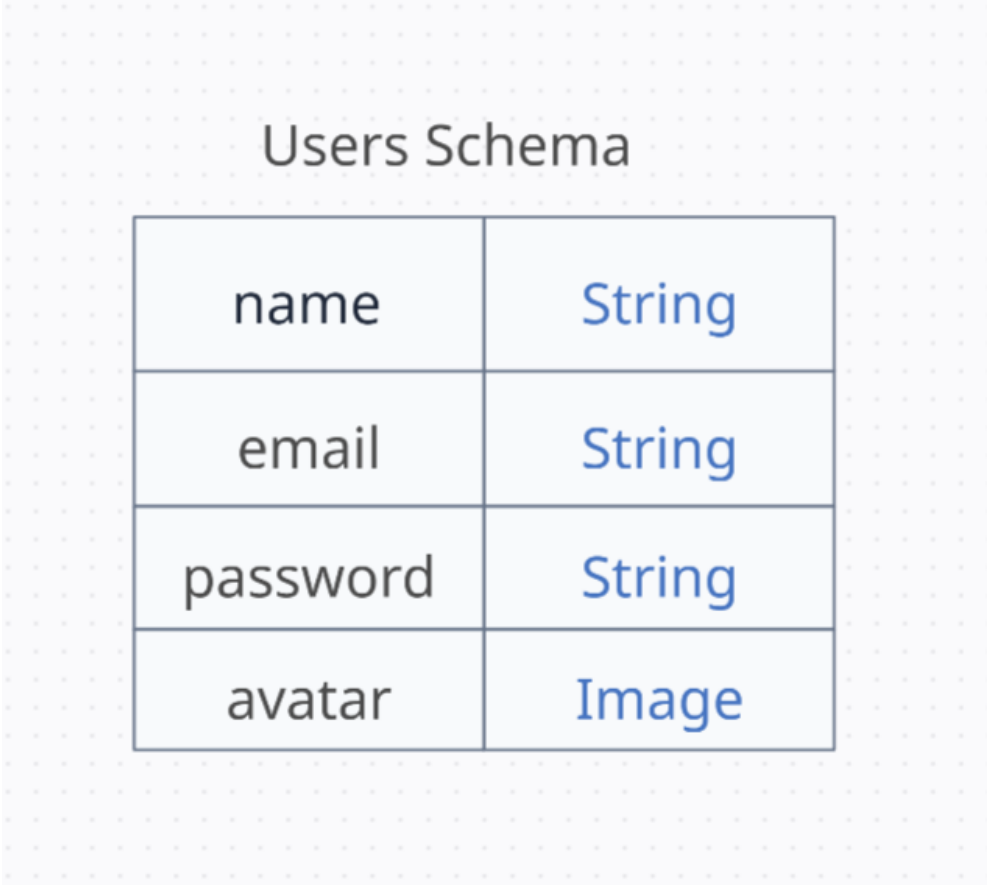
- **Socket.IO Server** : The Socket.IO server handles real-time messaging. It enables users to send and receive messages instantly. Messages are stored in the database for future retrieval.



4. Database Schema

The MongoDB database is used to store user information and chat messages. The schema includes the following collections:

- **Users Collection**: Stores user information such as username, password (hashed and salted), and user-related metadata.



- **Conversations Collection**: Represents conversations between users. It includes the participants and an array of message objects with sender, receiver, and message content.

Message Schema

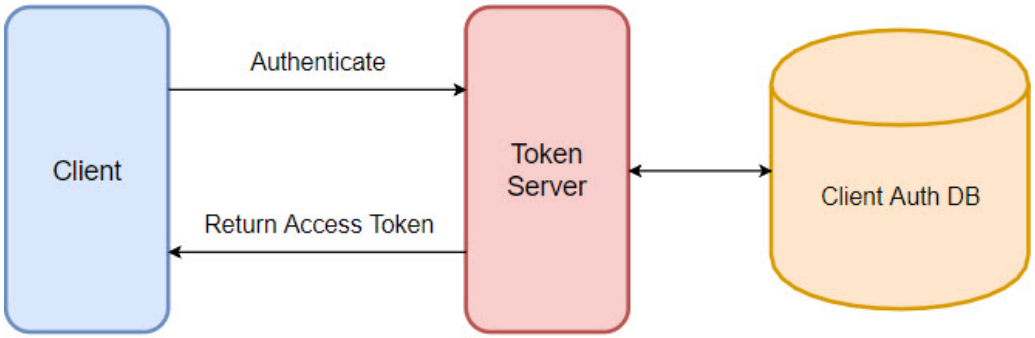
sender	ObjectId
content	String
chat	ObejctId
timestamps	String

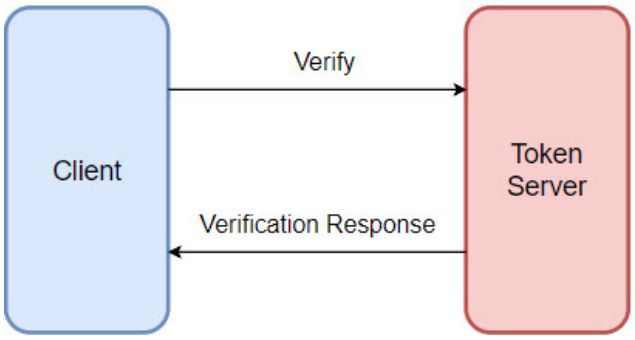
- **Chat Model Collection**: represents if chat is group chat or not , user type , latest message and timestamps.

```
const chatModel = mongoose.Schema(
  {
    chatName: {
      type: String,
      trim: true,
    },
    isGroupChat: {
      type: Boolean,
      default: false,
    },
    latestMessage: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Message',
    },
    users: [
      {
        type: mongoose.Schema.Types.ObjectI
        ref: 'User',
      },
    ],
    groupAdmin: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
    },
  },
  { timestamps: true }
);
```

5. User Authentication

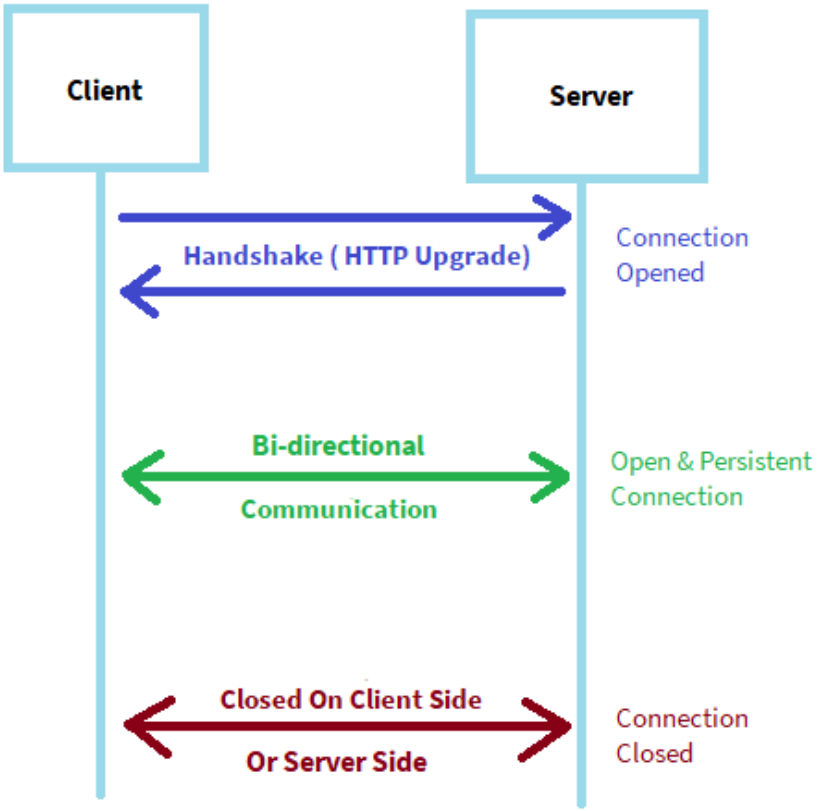
User authentication is implemented using JWT (JSON Web Tokens). Upon successful registration or login, the server issues a token that must be included in the headers of subsequent requests to access protected routes.





6. Real-time Messaging with Socket.IO

Socket.IO is used to enable real-time messaging. The server emits and listens for events related to message sending and receiving. Messages are stored in the database to ensure that chat history is preserved.



7. Security Considerations

- User passwords are securely hashed and salted before storage in the database.

- User sessions and authentication tokens are managed with JWT to ensure secure access to protected resources.

- Data validation and sanitization are implemented to prevent common security vulnerabilities like SQL injection and cross-site scripting.

8. Scalability and Performance

To enhance scalability and performance, the system can be scaled horizontally by deploying multiple instances of the server and database. Load balancing and caching mechanisms can also be implemented to handle a large number of concurrent users.

9. Deployment

The application is deployed on a cloud infrastructure provider Render. Continuous integration and continuous deployment (CI/CD) pipelines are also set up for automated testing and deployment.

10. Conclusion

The MERN Stack Chat Application with Socket.IO provides users with a real-time chat experience. By using the MERN stack and Socket.IO, the application ensures a modern and scalable architecture that can be further extended and enhanced with additional features and capabilities. Careful consideration of security and performance aspects ensures a robust and reliable application for users to communicate in real-time.