Daniel Schuster

Term Project: Instant Messaging App

Architecture Document

Table of Contents

1 Introduction

This document provides an architectural overview of the backend of an instant messaging application built using the MERN stack. It details the system design, architectural goals, database structure, and microservices implementation.

1.1 Purpose

The purpose of this document is to guide developers and stakeholders in understanding the backend architecture of the application. It serves as a reference for future development, ensuring consistency and maintainability.

1.2 Target Audience

The target audience includes backend developers, software architects, and project stakeholders interested in the system design and implementation details.

1.3 Terms and Definitions

- MERN: MongoDB, Express.js, React.js, Node.js
- API: Application Programming Interface
- WebSocket: A protocol for real-time communication
- JWT: JSON Web Token for authentication

2 Architectural Goals and Constraints

This backend architecture is designed to handle real-time messaging efficiently, ensuring secure, scalable, and reliable communications.

2.1 Security

The system implements authentication using JWTs, encrypts stored passwords, and utilizes role-based access control (RBAC) to protect sensitive operations.

2.2 Persistency

Data is stored in MongoDB, ensuring persistence of user accounts, chat history, and chat rooms. Data is indexed for fast retrieval, and backups are scheduled regularly.

2.3 Reliability/Availability

The system employs auto-scaling and load balancing to ensure high availability. WebSockets handle real-time communication with fallbacks to polling mechanisms in case of failure.

3 System Overview

The system consists of a backend server handling client connections, authentication, real-time messaging, and database operations. Users can register, create chat rooms, send messages, and view chat history. The backend is structured in a layered architecture with API endpoints, business logic, and data persistence layers.

The system follows a client-server model where users interact with the application through a frontend interface, which communicates with the backend via RESTful APIs and WebSocket connections. The backend manages message delivery, user authentication, and data storage.

The key components of the system include:

- **User Interface (Frontend)**: The frontend application provides an intuitive interface for users to send and receive messages.
- **API Gateway**: The centralized entry point for all client requests, responsible for routing, authentication, and load balancing.
- **Microservices**: Independent backend services responsible for authentication, messaging, user management, and chat room operations.
- **Database Management**: A MongoDB-based data storage system that handles user accounts, messages, and chat room data.
- **Real-time Communication Engine**: WebSockets facilitate bidirectional communication between clients and the server, ensuring low-latency message delivery.

4 System Architecture

4.1 Front-end Architecture

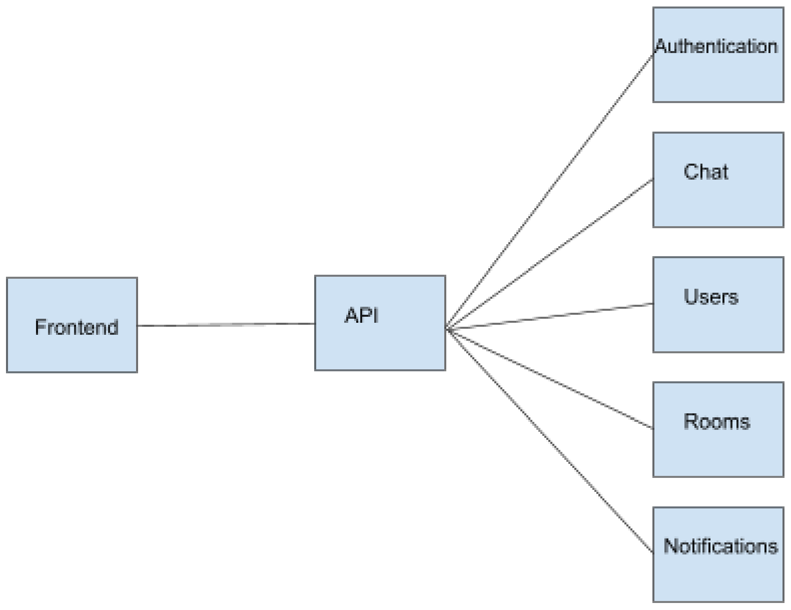
Although this document focuses on the backend, the frontend interacts with the backend via RESTful APIs and WebSocket connections. It uses React.js for the user interface and Redux for state management.

4.2 Back-end Microservices Architecture

The backend follows a microservices-based approach with the following services:

- **Authentication Service**: Handles user registration, login, and token-based authentication.
- **Chat Service**: Manages real-time messaging via WebSockets.
- **User Service**: Stores and retrieves user profile information.
- **Room Service**: Manages chat rooms, including creation and deletion.
- **Notification Service**: Sends push notifications to users for new messages and updates.
- **Dispatch Service**: Coordinates message delivery and ensures messages reach the correct recipients.

Each microservice communicates via HTTP REST APIs and WebSockets. The API Gateway centralizes requests and authentication before routing them to respective services.


Frontend
API
Authentication
Chat
Users
Rooms
Notifications

5 Database Design

The system uses MongoDB as the primary database. Advantages include scalability and efficiency, and there will be collections for users, messages, and chat rooms. The design ensures fast retrieval, indexing, and optimized querying for real-time communication.

Database Schema:

- **Users Collection:**
  - `_id` (ObjectId, Primary Key)
  - `username` (String, Unique)
  - `email` (String, Unique, Encrypted)
  - `password` (String, Hashed)
  - `firstName` (String)
  - `lastName` (String)
  - `createdAt` (Timestamp)
  - `updatedAt` (Timestamp)
- **Messages Collection:**
  - `_id` (ObjectId, Primary Key)
  - `senderId` (ObjectId, Reference to Users)
  - `receiverId` (ObjectId, Reference to Users)
  - `chatRoomId` (ObjectId, Reference to Chat Rooms)
  - `message` (String)
  - `timestamp` (Timestamp)
  - `status` (String, e.g., Sent, Delivered, Read)
- **Chat Rooms Collection:**
  - `_id` (ObjectId, Primary Key)
  - `roomName` (String, Unique)

  - `participants` (Array of ObjectIds, References to Users)
  - `createdAt` (Timestamp)
  - `updatedAt` (Timestamp)

6 Issue and Concerns

- Scalability: Ensuring the WebSocket server scales efficiently under heavy load.
- Data Consistency: Managing synchronization between services.
- Security: Preventing unauthorized access and data breaches.