Messaging Service Prototype System Design Document

Prepared by **Abdul Mannan**

September 22, 2024

Version: 1.0

Confidential

Contents

1 Introduction

1.1 Purpose

This document provides a comprehensive system design for the **Messaging Service Prototype**, a scalable, responsive, and real-time messaging application. It outlines the architecture, components, technologies, and strategies employed to build a robust platform capable of handling high traffic and delivering an exceptional user experience.

1.2 Scope

The **Messaging Service Prototype** aims to replicate and enhance the core functionalities of modern messaging applications, including real-time messaging, voice and video calls, media sharing, and seamless user authentication. This document covers the high-level architecture, detailed component design, data models, API specifications, scalability considerations, security measures, and maintenance plans.

1.3 Audience

This document is intended for software architects, developers, project managers, and stakeholders involved in the development and maintenance of the **Messaging Service Prototype**.

2 System Overview

2.1 High-Level Architecture

The system adopts a serverless and microservices-oriented architecture, leveraging Next.js capabilities for both frontend and backend functionalities. This approach allows seamless integration of API routes within the Next.js application, eliminating the need for a separate Express.js server. The primary components include:

- **Frontend and Backend**: Unified within Next.js, utilizing both server-side rendering (SSR) and API routes for backend functionalities.
- **Database**: Prisma ORM connected to a MongoDB database for flexible and scalable data management.
- **Real-time Messaging**: Pusher manages real-time events such as chat messages, online statuses, and notifications.
- **Voice and Video Calls**: ZegoCloud facilitates real-time voice and video call services.
- **Authentication**: NextAuth.js integrates with Google, GitHub, and a custom email/password flow using Nodemailer.
- **File Storage**: Media messages like images and voice clips are stored on Cloudinary.
- **Search**: Custom messaging service prototype implemented for search functionality.

2.2 Architecture Diagram

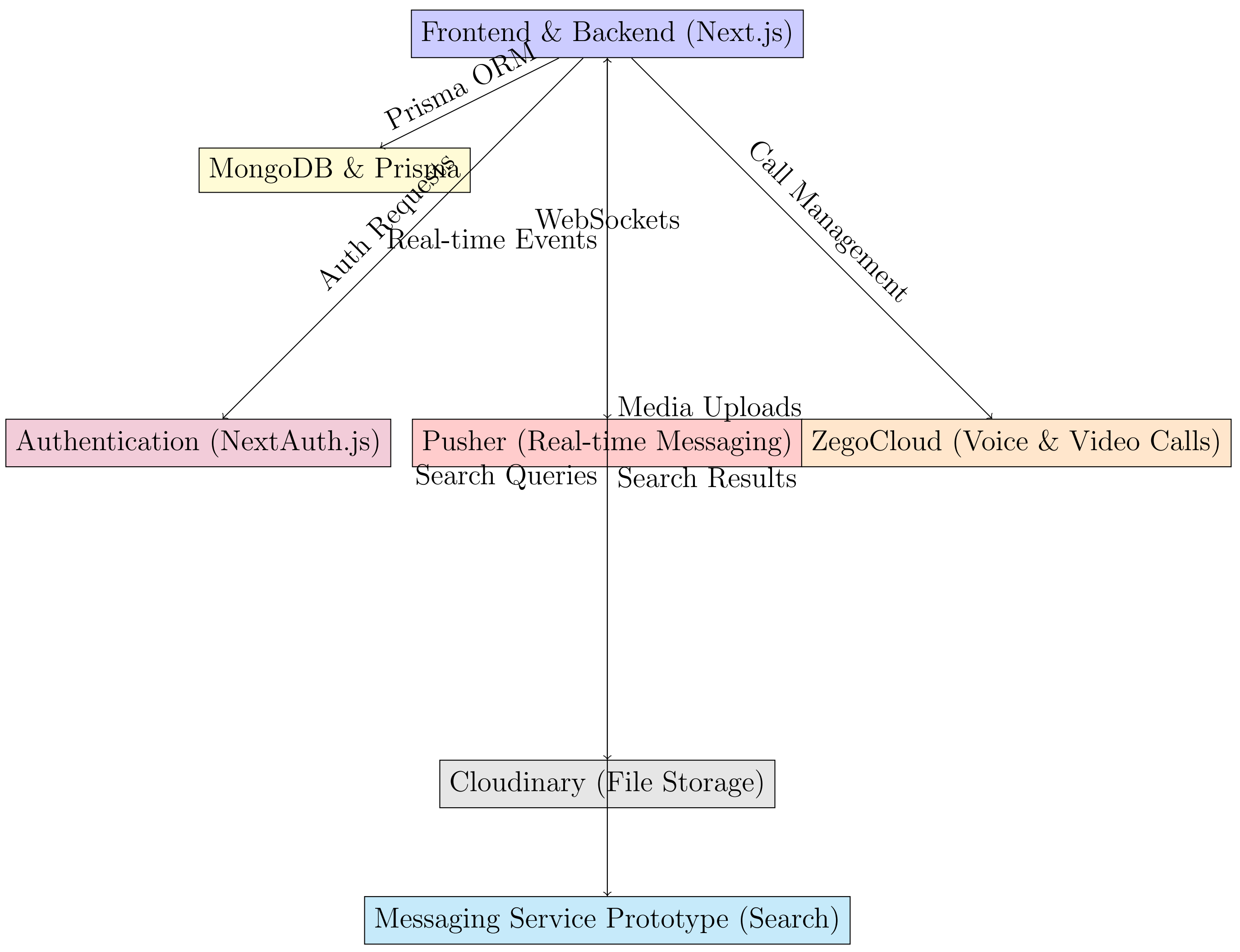
Frontend & Backend (Next.js)
Prisma ORM
MongoDB & Prisma
Auth Requests
Real-time Events
WebSockets
Call Management
Media Uploads
Authentication (NextAuth.js)
Pusher (Real-time Messaging)
ZegoCloud (Voice & Video Calls)
Search Queries
Search Results
Cloudinary (File Storage)
Messaging Service Prototype (Search)

Figure 1: High-Level Architecture Diagram

3 Functional Requirements

3.1 Authentication & Signup

- **Social Login**: Users can log in using Google and GitHub via OAuth.
- **Email Signup**: Custom login/signup using email verification through Nodemailer.
- **Session Management**: User sessions managed using JWT stored in HTTP-only cookies.

3.2 Messaging

- **One-to-One Chats**: Facilitate direct messaging between two users.
- **Group Chats**: Support for group conversations with multiple participants.

- **Media Support**: Sending and receiving text, images, and voice messages.
- **Real-Time Updates**: Instant message delivery and status updates powered by Pusher.
- **Search**: Ability to search through messages and contacts using the messaging service prototype.

3.3 Notifications

- **Real-Time Notifications**: For new messages, events, and call requests.
- **Web Push Notifications**: Alert users of messages and calls even when the app is inactive.

3.4 Voice & Video Calls

- **Real-Time Communication**: Enable voice and video calls using ZegoCloud SDK.
- **Call Management**: Handle call invitations, acceptance, and termination events.

3.5 User Interface

- **Themes**: Support for both dark mode and light mode.
- **Responsive Design**: Mobile-first UI ensuring usability across all devices.

4 Non-Functional Requirements

4.1 Scalability

- The system must handle high traffic (100,000+ daily active users), especially during peak hours.
- Microservices architecture allows individual components to scale independently.

4.2 Performance

- Real-time updates must have message delivery times under 100ms.
- Efficient data handling strategies to minimize latency.

4.3 Security

- Data encryption at rest and in transit using TLS/SSL.
- Secure storage of user passwords using bcrypt with salting.
- Protection against common vulnerabilities such as XSS, CSRF, and injection attacks.

4.4 Availability

- Target of 99.9% uptime through redundant services and failover mechanisms.
- Load balancing and auto-scaling to handle traffic spikes.

4.5 Maintainability

- Modular codebase following best practices for ease of maintenance and updates.
- Comprehensive logging and monitoring for quick issue detection and resolution.

5 Detailed Component Design

5.1 Authentication Module

5.1.1 Overview

Handles user authentication and authorization, supporting both social and email-based signups.

5.1.2 Components

- **NextAuth.js**: Manages OAuth flows for Google and GitHub.
- **JWT**: Manages user sessions with tokens stored in HTTP-only cookies.
- **Nodemailer**: Sends verification emails with tokenized URLs for email-based signup.

5.1.3 Workflow

1. User initiates login/signup via frontend.
2. For social login, NextAuth.js handles OAuth flow with Google/GitHub.
3. For email signup, backend sends verification email via Nodemailer.
4. Upon verification, JWT is issued and stored in an HTTP-only cookie.
5. Subsequent requests validate JWT for authentication.

5.2 Messaging Module

5.2.1 Overview

Facilitates real-time communication between users, supporting one-to-one and group chats with media sharing.

5.2.2 Components

- **MongoDB & Prisma**: Stores messages, chats, and user data.
- **Pusher**: Manages real-time WebSocket connections for message delivery.
- **Cloudinary**: Stores media files such as images and voice clips.

5.2.3 Workflow

1. User sends a message via frontend.
2. Message is sent to Next.js API route and stored in MongoDB via Prisma.
3. If the message contains media, it is uploaded to Cloudinary, and the URL is stored in the message record.
4. Pusher broadcasts the message to the recipient(s) in real-time.
5. Recipient's frontend receives the message via WebSocket and updates the UI instantly.

5.3 Voice & Video Calls Module

5.3.1 Overview

Enables real-time voice and video communication between users using ZegoCloud.

5.3.2 Components

- **ZegoCloud SDK**: Handles media streaming and call functionalities.
- **Pusher**: Manages call invitations and signaling events.

5.3.3 Workflow

1. Caller initiates a call via the frontend.
2. Call invitation is sent via Pusher to the recipient.
3. Recipient receives the invitation and responds (accept/decline).
4. Upon acceptance, ZegoCloud establishes a media session between the caller and recipient.
5. During the call, media streams are handled by ZegoCloud, while call events (mute, end) are managed via Pusher.

5.4 Search Functionality

5.4.1 Overview

Provides search capabilities for messages, chats, and contacts using a custom messaging service prototype.

5.4.2 Components

- **Messaging Service Prototype**: Implements search algorithms to index and retrieve data.
- **Prisma & MongoDB**: Syncs data to the messaging service prototype for indexing.

5.4.3 Workflow

1. User inputs a search query in the frontend.
2. Frontend sends the query to the backend API.
3. Backend queries the messaging service prototype for matching results.
4. Results are returned to the frontend and displayed to the user.

6 Data Storage & Database Schema

6.1 MongoDB Database

Stores the main non-relational data with the following key entities:

- **Users**:
  - `_id`: ObjectId, Primary Key
  - `username`: String
  - `email`: String, Unique
  - `password_hash`: String
  - `profile_picture`: String (URL to Cloudinary)
  - `contacts`: Array of `User_IDs`
  - `created_at`: Date
- **Chats**:
  - `_id`: ObjectId, Primary Key
  - `is_group`: Boolean
  - `participants`: Array of `User_IDs`
  - `created_at`: Date
- **Messages**:
  - `_id`: ObjectId, Primary Key
  - `chat_id`: ObjectId, Foreign Key
  - `sender_id`: ObjectId, Foreign Key
  - `content`: String
  - `media_url`: String (optional)

  - `message_type`: String (e.g., text, image, voice)
  - `created_at`: Date

- **Groups**:
  - `_id`: ObjectId, Primary Key
  - `group_name`: String
  - `created_by`: ObjectId, Foreign Key
  - `participants`: Array of `User_IDs`
  - `created_at`: Date

6.2 Messaging Service Prototype

Used for implementing search functionality within the application.

6.3 Cloudinary (File Storage)

Media files (images, voice clips) are stored in Cloudinary. Each media message in the database contains a reference URL pointing to the stored file in Cloudinary.

7 API Folder Structure

```
api/
    auth/
        route.ts
    call/
        accepted/
            route.ts
        route.ts
    conversations/
        [conversationId]/
        seen/
            route.ts
        route.ts
    messages/
        route.ts
    register/
        route.ts
    settings/
        route.ts
    users/
```

```
        route.ts
    verify/
        route.ts
```

8 Real-Time Communication Design

8.1 Pusher Integration

Pusher is used to manage WebSocket connections for real-time messaging and notifications. Each user maintains a persistent connection to Pusher channels corresponding to their active chats and groups.

8.1.1 Channels

- **User Channels**: Individual channels for one-to-one chats.
- **Group Channels**: Channels for group chats where messages are broadcasted to all participants.

8.1.2 Events

- **Message Sent**: Triggered when a new message is sent.
- **Message Received**: Triggered when a message is received by a user.
- **User Online/Offline**: Updates user presence status.
- **Call Invitation**: Notifies users of incoming call requests.

8.2 ZegoCloud Integration

ZegoCloud manages the media streaming for voice and video calls. It handles the establishment, maintenance, and termination of media sessions.

8.2.1 Call Workflow

1. Caller initiates a call via the frontend.
2. Call invitation is sent via Pusher to the recipient.
3. Recipient receives the invitation and responds (accept/decline).
4. Upon acceptance, ZegoCloud establishes a media session between the caller and recipient.
5. During the call, media streams are handled by ZegoCloud, while call events (mute, end) are managed via Pusher.

9 Scalability and Performance Considerations

9.1 Database Sharding

As user traffic grows, database sharding by user ID ensures faster lookups and prevents bottlenecks. Each shard handles a subset of users, distributing the load evenly across database instances.

9.2 Load Balancing

Utilize Vercel's built-in load balancing capabilities to distribute incoming requests across multiple serverless functions, ensuring no single instance becomes a performance bottleneck.

9.3 Auto-Scaling

Leverage Next.js' serverless architecture on platforms like Vercel, which inherently supports auto-scaling based on traffic patterns, ensuring resources are allocated efficiently.

9.4 Caching Strategies

Implement efficient data handling strategies to minimize latency, such as in-memory caching within serverless functions or leveraging Cloudinary's CDN capabilities for media delivery.

9.5 Content Delivery Network (CDN)

Use a CDN like Cloudinary's built-in CDN to deliver media assets (images, voice clips) quickly to users across different geographical locations, reducing latency and improving load times.

10 Security Considerations

10.1 Data Encryption

- **In Transit**: All data transmitted between clients and servers is encrypted using TLS/SSL.
- **At Rest**: Sensitive data stored in databases and storage services is encrypted to protect against unauthorized access.

10.2 Authentication Security

- Use strong hashing algorithms (bcrypt) with salting for storing user passwords.
- Implement rate limiting on authentication endpoints to prevent brute-force attacks.
- Utilize JWT with short expiration times and refresh tokens for session management.

10.3 Input Validation and Sanitization

Ensure all user inputs are validated and sanitized to prevent SQL injection, Cross-Site Scripting (XSS), and other injection attacks.

10.4 Access Control

Implement role-based access control (RBAC) to restrict access to sensitive operations and data based on user roles and permissions.

10.5 Monitoring and Logging

- Monitor all authentication attempts and unusual activities.
- Log all critical operations and access patterns for auditing and incident response.

11 Technology Stack

11.1 Frontend and Backend

- **Framework**: Next.js
- **Language**: TypeScript
- **Styling**: Tailwind CSS or Styled Components
- **State Management**: Redux or Context API

11.2 Database and Storage

- **Primary Database**: MongoDB
- **ORM**: Prisma
- **File Storage**: Cloudinary

11.3 Real-Time Services

- **Messaging**: Pusher
- **Voice & Video Calls**: ZegoCloud

11.4 Authentication

- **Library**: NextAuth.js
- **Email Service**: Nodemailer

12 API Folder Structure

```
api/
    auth/
        route.ts
    call/
        accepted/
            route.ts
        route.ts
    conversations/
        [conversationId]/
        seen/
            route.ts
        route.ts
    messages/
        route.ts
    register/
        route.ts
    settings/
        route.ts
    users/
        route.ts
    verify/
        route.ts
```

13 Real-Time Communication Design

13.1 Pusher Integration

Pusher is used to manage WebSocket connections for real-time messaging and notifications. Each user maintains a persistent connection to Pusher channels corresponding to their active chats and groups.

13.1.1 Channels

- **User Channels**: Individual channels for one-to-one chats.
- **Group Channels**: Channels for group chats where messages are broadcasted to all participants.

13.1.2 Events

- **Message Sent**: Triggered when a new message is sent.
- **Message Received**: Triggered when a message is received by a user.
- **User Online/Offline**: Updates user presence status.
- **Call Invitation**: Notifies users of incoming call requests.

13.2 ZegoCloud Integration

ZegoCloud manages the media streaming for voice and video calls. It handles the establishment, maintenance, and termination of media sessions.

13.2.1 Call Workflow

1. Caller initiates a call via the frontend.
2. Call invitation is sent via Pusher to the recipient.
3. Recipient receives the invitation and responds (accept/decline).
4. Upon acceptance, ZegoCloud establishes a media session between the caller and recipient.
5. During the call, media streams are handled by ZegoCloud, while call events (mute, end) are managed via Pusher.

14 Scalability and Performance Considerations

14.1 Database Sharding

As user traffic grows, database sharding by user ID ensures faster lookups and prevents bottlenecks. Each shard handles a subset of users, distributing the load evenly across database instances.

14.2 Load Balancing

Utilize Vercel's built-in load balancing capabilities to distribute incoming requests across multiple serverless functions, ensuring no single instance becomes a performance bottleneck.

14.3 Auto-Scaling

Leverage Next.js' serverless architecture on platforms like Vercel, which inherently supports auto-scaling based on traffic patterns, ensuring resources are allocated efficiently.

14.4 Caching Strategies

Implement efficient data handling strategies to minimize latency, such as in-memory caching within serverless functions or leveraging Cloudinary's CDN capabilities for media delivery.

14.5 Content Delivery Network (CDN)

Use a CDN like Cloudinary's built-in CDN to deliver media assets (images, voice clips) quickly to users across different geographical locations, reducing latency and improving load times.

15 Security Considerations

15.1 Data Encryption

- **In Transit**: All data transmitted between clients and servers is encrypted using TLS/SSL.
- **At Rest**: Sensitive data stored in databases and storage services is encrypted to protect against unauthorized access.

15.2 Authentication Security

- Use strong hashing algorithms (bcrypt) with salting for storing user passwords.
- Implement rate limiting on authentication endpoints to prevent brute-force attacks.
- Utilize JWT with short expiration times and refresh tokens for session management.

15.3 Input Validation and Sanitization

Ensure all user inputs are validated and sanitized to prevent SQL injection, Cross-Site Scripting (XSS), and other injection attacks.

15.4 Access Control

Implement role-based access control (RBAC) to restrict access to sensitive operations and data based on user roles and permissions.

15.5 Monitoring and Logging

- Monitor all authentication attempts and unusual activities.
- Log all critical operations and access patterns for auditing and incident response.

16 Technology Stack

16.1 Frontend and Backend

- **Framework**: Next.js
- **Language**: TypeScript
- **Styling**: Tailwind CSS or Styled Components
- **State Management**: Redux or Context API

16.2 Database and Storage

- **Primary Database**: MongoDB
- **ORM**: Prisma
- **File Storage**: Cloudinary

16.3 Real-Time Services

- **Messaging**: Pusher
- **Voice & Video Calls**: ZegoCloud

16.4 Authentication

- **Library**: NextAuth.js
- **Email Service**: Nodemailer

17 Conclusion

This system design document outlines the architecture, components, technologies, and strategies for building the **Messaging Service Prototype** as a full Next.js application with Prisma connected to MongoDB. By leveraging modern web technologies such as Next.js, Pusher, Prisma, Cloudinary, and ZegoCloud, the system is designed to be scalable, responsive, and capable of delivering real-time interactions efficiently. Emphasis on security, scalability, and maintainability ensures that the platform can handle high traffic while providing a seamless and secure user experience. This comprehensive design serves as a blueprint for the development and deployment of a robust messaging platform, ready to meet the demands of modern users.