**System Design Document for Next.js Real-time Chat App**

Vercel Link : https://chat-app-sarthak-saxena-1609s-projects.vercel.app/

1. Introduction

The application is a fully responsive real-time chat app built with Next.js 13, MongoDB, Tailwind CSS, Pusher, Next-Auth, and Cloudinary. The app includes the following features:

- Real-time chat updates using Pusher

- Group chat functionality

- Ability to delete chat history

- Image hosting via Cloudinary

- Dynamic theme support (Light/Dark mode)

- Fully responsive for both Desktop and Mobile screens

2. System Architecture

The architecture is divided into three main parts:

- Frontend: Built with Next.js for server-side rendering (SSR) and client-side rendering (CSR) using the app router.

- Backend: Node.js/Express with Next.js API routes handling requests. Pusher is used for real-time updates, while Next-Auth is used for authentication (GitHub and Google OAuth).

- Database: MongoDB is used as the main database to store user information, chat history, and metadata related to images.

3. Data Flow

1. User Authentication:

- OAuth with GitHub or Google is used for user login. Next-Auth manages the OAuth flow and stores user sessions in MongoDB.

2. Real-time Messaging:

- Pusher handles real-time updates. When a user sends a message, it triggers an event through Pusher, which notifies other clients to update the chat feed instantly.

3. Image Upload:

- Users can upload images to Cloudinary. Once uploaded, Cloudinary returns a URL, which is stored in MongoDB and sent to the chat feed.

4. Dark/Light Mode:

- The user’s theme preference is stored locally using Zustand (state management) and applied dynamically.

4. Technology Choices

- Next.js: The primary framework for the frontend and backend API routes. It was chosen for its powerful server-side rendering, routing, and performance optimization.

- MongoDB: A NoSQL database that efficiently stores chat messages and user data. It was chosen for its flexibility in handling dynamic, schema-less data like chat messages.

- Tailwind CSS: A utility-first CSS framework that enables quick and responsive UI development with minimal CSS overhead.

- Pusher: Chosen for real-time messaging because of its ease of integration, scalability, and reliability.

- Next-Auth: Used for authentication (Google and GitHub OAuth) to simplify user login and session management.

- Cloudinary: A cloud-based image and video management service that allows for secure and scalable media hosting, making it perfect for storing and retrieving user-uploaded images.

- Zustand: Lightweight state management library used to manage global state like the user’s theme preference (dark/light mode).

5. Scalability and Performance

- Real-time Chat: Pusher provides scalable and low-latency real-time communication. The app can handle multiple users interacting simultaneously without performance issues.

- MongoDB: The database can scale horizontally, making it suitable for handling large volumes of chat messages.

- Image Hosting: Cloudinary manages media storage and retrieval, offloading the burden of hosting images to a specialized service.

Setup Documentation for Next.js Real-time Chat App

Prerequisites

* Node.js (>= 14.x)
* Yarn (>= 1.22)
* MongoDB (cloud-hosted via MongoDB Atlas)
* Pusher (for real-time messaging)
* Cloudinary (for image hosting)
* GitHub OAuth App (for user authentication)
* Google OAuth App (for user authentication)

Step-by-Step Setup Instructions For Deploying on Localhost

**1. Setup MongoDB:**

1. Go to [MongoDB Atlas] (https://www.mongodb.com/atlas/database) and sign up or log in.

2. Create a new cluster with the Free tier.

3. Create a new database user and note down the password.

4. Add `0.0.0.0/0` to your IP Access List to allow all IPs to connect.

5. Copy the connection string, replace `<password>` with your password, and append `/test` to the end of the string.

**2. Setup GitHub Authentication**

1. Go to [GitHub](https://github.com) and navigate to Settings > Developer settings > OAuth Apps.

2. Create a new OAuth App:

- Homepage URL: `http://localhost:3020/`

- Authorization callback URL: `http://localhost:3020/api/auth/callback/github`

3. Copy the Client ID and Client Secret.

**3. Setup Google Authentication**

1. Go to [Google Cloud Console] (https://console.cloud.google.com) and create a new project.

2. Navigate to API & Services > OAuth consent screen and set up the consent screen.

3. Create new OAuth credentials with:

- Authorized redirect URIs: `http://localhost:3020/api/auth/callback/google`

4. Copy the Client ID and Client Secret.

**4. Setup Cloudinary**

1. Go to [Cloudinary] (https://console.cloudinary.com), log in, and copy the Cloud name from your dashboard.

2. Go to Settings > Upload and create a new Upload preset with Unsigned mode.

3. Note down the preset name.

**5. Setup Pusher**

1. Go to [Pusher Channels] (https://dashboard.pusher.com/channels), create a new app with the following settings:

- Frontend: React

- Backend: Node.js

2. Note down the App ID, Key, Secret, and Cluster.

**6. Project Setup**

1. Clone the project from the repository:

bash

git clone https://github.com/your-repo.git

cd your-repo

2. Create an `.env` file:

3. Inside the `.env` file, add the following environment variables with values from the previous steps:

- MongoDB connection string

- Pusher app details (App ID, Key, Secret, Cluster)

- Cloudinary credentials (Cloud name, upload preset)

- GitHub and Google OAuth credentials (Client IDs and secrets)

4. Install the dependencies:

yarn install

5. Push the Prisma schema to your MongoDB:

yarn prisma db push

6. Generate the Prisma client:

prisma generate

7. Run the app locally:

yarn dev

**7. Testing the App**

Once your environment variables are set and the app is running, open `http://localhost:3020/` in your browser. You should be able to sign in using GitHub or Google OAuth, send real-time messages, and upload images.

Name: Sarthak Saxena

University: IIT Roorkee

Department: Civil Engineering