Chat Application

Full Documentation

Blue Matrix

Contents

Chat Application – Full Documentation

A full-stack real-time chat application built using Next.js, NestJS, WebSockets, and PostgreSQL. This system includes user authentication, contact listing, real-time messaging, and clean architecture following best practices.

You can find the project code here: https://github.com/BinuMDX/Chat-Application

1 Tech Stack

This project demonstrates a production-ready real-time chat system with authentication, user management, and messaging. It uses:

- NestJS for backend APIs & WebSockets
- Next.js for frontend UI & client-side socket connection
- Prisma + PostgreSQL to store users & messages

2 Features

2.1 Authentication Features

2.1.1 User Registration

- New users can create an account with:
    - Name, Email, Password
- Backend validates:
    - Email format, Password strength (optional), Email uniqueness
- Passwords are hashed using bcrypt.
- Upon successful registration:
    - JWT token is generated
    - User details (except password) returned

2.1.2 User Login

- Existing users can log in with:
    - Email, Password
- Backend verifies:
    - User existence, Password correctness
- Responds with:
    - JWT access token, Authenticated user info (id, name, email)

2.2 User Management Features

2.2.1 List All Users

- Authenticated user can fetch all registered users.
- List excludes the logged-in user.
- Response includes:
    - User ID, Name, Email (optional)

2.2.2 View User Profile

- Fetch profile of any user by ID.
- Includes:
    - Name, Email, CreatedAt
- Does **not** return password or sensitive data

2.3 Messaging Features

2.3.1 Send Message (Realtime)

- User sends a message with: { "receiverId": "...", "content": "Hello" }
- Backend handles:
    - Stores message in database
    - Sends real-time event to receiver using WebSocket
    - Updates chat window instantly for both users

2.3.2 Chat History (per user)

- User can open a chat with someone and retrieve history: GET chat/messages/receiverId
- Messages sorted by timestamp ascending.
- Attributes:
    - Text messages
    - Timestamps
    - Sender/receiver id

2.3.3 Message Timestamps

- Each message includes:
    - CreatedAt (ISO format)
- Frontend shows:
    - Time (e.g., 10:30 AM)

2.3.4 Real-time Delivery

- As soon as sender sends message:Receiver sees the message instantly

2.4 WebSocket Realtime Features

2.4.1 Join Personal Room

- After login, client emits: socket.emit("join", { userId })
- Backend creates a dedicated room for the user.

2.4.2 Real-time Message Event

- Backend emits: "receive_message" with sender info + content.

2.5 Frontend Features (Next.js UI)

2.5.1 Login Page

- Email and password fields
- Error handling
- Redirects to chat page on success
- Uses API via Axios

2.5.2 Sign Up Page

- Name, email, password fields
- Validations
- Success toast notification
- Redirect to login

2.5.3 Chat App Layout

- Sidebar (user list)
- Chat window
- Message box

- Real-time updates

2.5.4 User List Sidebar

- All users
- Clicking a user opens chat

2.5.5 Chat Window Features

- Displays entire conversation
- Auto-scrolls to bottom on new message
- Shows sender/receiver differently (left/right align)
- Timestamp under each message
- Loading indicators

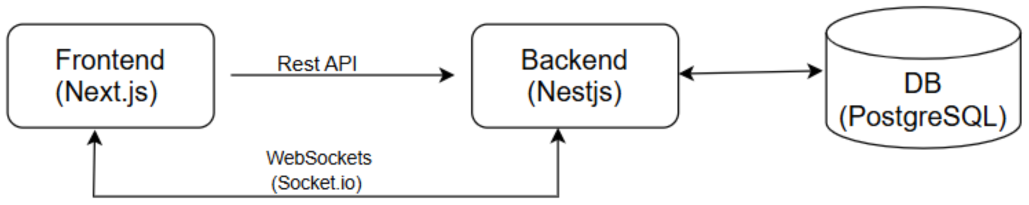
2.5.6 Message Input Box

- Textbox + Send button
- Enter key to send
- Auto-clears after sending

2.5.7 Responsive UI

- Mobile-friendly
- Sidebar collapses on small screens
- Optimized for both desktop and mobile

3 Architecture



4 Database Schema (Prisma Schema)

```
model User {
  id            Int                       @id @default(autoincrement())
  username      String                    @unique
  email         String                    @unique
  hashpassword  String
  conversations ConversationParticipant[]
  messages      Message[]
  createdAt     DateTime                  @default(now())
  updatedAt     DateTime                  @updatedAt
  hashedRt      String?

  @@map("users")
}

model Conversation {
  id            String                     @id @default(cuid())
  participants  ConversationParticipant[]
  messages      Message[]
  createdAt     DateTime                   @default(now())
  updatedAt     DateTime                   @updatedAt
}

model ConversationParticipant {
  id             Int          @id @default(autoincrement())
  userId         Int
  conversationId String

  user          User          @relation(fields: [userId], references: [id])
  conversation  Conversation  @relation(fields: [conversationId], references: [id])

  @@unique([userId, conversationId])
}

model Message {
  id             Int          @id @default(autoincrement())
  text           String
  senderId       Int
  conversationId String
  readAt         DateTime?

  sender         User         @relation(fields: [senderId], references: [id])
  conversation   Conversation @relation(fields: [conversationId], references: [id])
  createdAt      DateTime     @default(now())
  updatedAt      DateTime     @updatedAt
}
```

5 API Endpoints

5.1 Auth Route

POST	/auth/signup	Register a new user
POST	/auth/login	Login user & return access token

5.2 User Routes (Protected)

Requires Authorization header:
Authorization: Bearer <token>

GET	/users	Get all users except the logged-in user

5.3 Message Routes (Protected)

Requires Authorization header:
Authorization: Bearer <token>

POST	chat/messages/:conversationId	deprecated
GET	chat/messages/:conversationId	Get chat history between logged user & :userId
GET	chat/conversations	Join to a conversation
POST	chat/conversations	Create or get conversation