Detailed Backend Documentation: LAN Voice Call System

This document summarizes the complete backend architecture of a LAN-based real-time voice calling system using Node.js, Express, Socket.io, WebRTC, and MongoDB. The goal is to allow users connected on the same local area network (LAN) to authenticate and make peer-to-peer voice calls from their browsers without internet connectivity.

# 1. Project Structure

- server.js → Main entry point of the server.  
- /routes/  
 - authRoutes.js → Routes for user signup, login, OTP verification.  
 - callRoutes.js → REST APIs for call history (if needed).  
- /models/  
 - User.js → User schema and methods.  
 - Call.js → Call schema (used for logging call history).  
- /public/  
 - test\_client.html → A basic browser UI to register, initiate and accept calls.  
 - client.js → WebRTC and socket handling logic.  
- /config/  
 - db.js → MongoDB connection setup.  
- .env → Environment variables like PORT, JWT\_SECRET, DB\_URI etc.

# 2. models/User.js

Schema:  
- scholarId (String): Unique ID of student/user.  
- otp (String): Generated OTP for login.  
- otpExpiresAt (Date): Expiry time of OTP.  
- macAddress (String): Client's MAC address.  
- ipAddress (String): Client's IP address.  
  
Methods:  
- isOTPValid(enteredOTP): Validates OTP and expiry.

# 3. models/Call.js

Schema:  
- callerId (String): ID of caller.  
- receiverId (String): ID of receiver.  
- timestamp (Date): Time of call.  
- status (String): "missed", "accepted", "ended".

# 4. routes/authRoutes.js

POST /auth/signup  
- Accepts scholarId, macAddress.  
- Generates OTP, stores it with expiry.  
  
POST /auth/verify-otp  
- Accepts scholarId, otp.  
- Validates OTP and returns JWT.  
  
GET /auth/profile  
- Returns user profile if valid JWT token is present.

# 5. routes/callRoutes.js

These are optional REST endpoints for storing or retrieving call history.  
  
POST /call/initiate  
POST /call/accept  
POST /call/end  
GET /call/history/:userId

# 6. server.js

- Express + HTTP + Socket.io server setup.  
- CORS enabled for LAN use.  
- Registers API routes and serves static files from /public.  
- Socket.io Events:  
 - register-user → Maps userId to socketId.  
 - callRequest → Sends "incomingCall" event.  
 - offer → Sends WebRTC offer.  
 - answer → Sends WebRTC answer.  
 - candidate → Sends ICE candidates.  
 - callEnded → Notifies other user.

# 7. public/client.js

- Handles WebRTC peer connection setup.  
- Emits register-user, callRequest, offer, answer, candidate.  
- Accepts incomingCall, offer, answer, candidate via socket.on handlers.  
- Uses navigator.mediaDevices.getUserMedia for microphone access.  
- Plays remote audio in an <audio> tag.

# 8. public/test\_client.html

- Provides basic interface for registration, calling, accepting, and ending calls.  
- Inputs for userId and receiverId.  
- Buttons linked to client.js functions.  
- Contains <audio autoplay> element for remote stream playback.

# 9. .env Variables

PORT=3000  
JWT\_SECRET=your\_secret\_key  
MONGODB\_LAN\_URL=mongodb://localhost:27017/lan\_communication

# 10. Summary

This backend supports LAN-based peer-to-peer voice calling with full signaling using Socket.io and WebRTC. OTP-based authentication is built-in with MongoDB and Mongoose schemas. A lightweight test client in HTML + JS helps simulate full call flow across two browser tabs.