**LAN-Based Communication System: Backend Development Roadmap**

**Introduction**

This document provides a **detailed roadmap** for developing the backend of a **LAN-based communication system** where students at NIT Silchar can **call, chat, and share files** over a high-speed LAN network without internet dependency. The backend will be built using **Node.js, Express, MongoDB**, and **WebSockets** while leveraging **computer networking principles** to enable seamless data exchange.

**Project Scope and Features**

**Authentication & Authorization**

* **Login Mechanism:**
  + Upon connecting to LAN, a login page will automatically open.
  + Students will enter a **unique identifier (Scholar ID)**.
  + An **OTP** will be generated and displayed on the screen.
  + Upon **OTP verification**, access is granted.
* **Security Measures:**
  + **JWT-based authentication** for secure session management.
  + **MAC Address & IP Binding** for device authentication.

**Voice Calling Feature (Phase 1 Priority)**

* **Real-time voice calling over LAN** using **WebRTC**.
* **Direct peer-to-peer (P2P) connections** for efficient data transfer.
* **Signaling mechanism** to establish and manage calls.
* **Call notifications and status updates** using **Socket.io**.
* **Mute, hold, and end call functionalities**.

**Upcoming Features (Phase 2 & Beyond)**

* **Chat System (Real-time messaging over LAN using Socket.io).**
* **File Sharing (P2P file transfer using WebRTC DataChannel).**

**Technology Stack**

**Backend Technologies**

* **Node.js** → Runtime for backend logic.
* **Express.js** → Web framework for API handling.
* **MongoDB** → NoSQL database for storing users and call history.
* **Socket.io** → Enables real-time notifications and call signaling.
* **WebRTC** → Handles peer-to-peer voice calls over LAN.

**Networking Concepts Used**

* **LAN-based IP Addressing** → Devices communicate using assigned local IPs (e.g., 192.168.x.x).
* **WebRTC Peer-to-Peer** → Establishes direct media streams for voice communication.
* **NAT Traversal & STUN/TURN Servers** → Handles direct connections within LAN.
* **Network Ports and Sockets** → WebRTC for calls, Socket.io for signaling.

**Backend Development Roadmap**

**Step 1: Project Setup**

**Initialize Project**

1. Create project directory: mkdir lan-call-system && cd lan-call-system.
2. Initialize Node.js: npm init -y.
3. Install dependencies:
4. npm install express mongoose dotenv cors body-parser jsonwebtoken socket.io webrtc

**Project Structure**

lan-call-system/

│── models/ # Database Schemas

│── routes/ # API Endpoints

│── config/ # Server & Database Configuration

│── middleware/ # Authentication & Security

│── server.js # Main Entry Point

│── package.json # Dependencies

│── .env # Environment Variables

**Step 2: Database Setup (Mongo DB)**

**Schema Design**

* **Users Schema(User.js)**
* const userSchema = new mongoose.Schema({
* scholarId: { type: String, required: true, unique: true },
* macAddress: { type: String, required: true, unique: true },
* ipAddress: { type: String, required: true },
* isVerified: { type: Boolean, default: false },
* lastLogin: { type: Date, default: Date.now }
* });
* **Call History Schema (Call.js)**
* const callSchema = new mongoose.Schema({
* callerId: { type: String, required: true },
* receiverId: { type: String, required: true },
* status: { type: String, enum: ['ongoing', 'ended', 'missed'], default: 'ongoing' },
* timestamp: { type: Date, default: Date.now }
* });

**Step 3: API Development for Voice Calling**

**1. Signaling Server Setup (server.js)**

* Handles WebRTC connection establishment.
* Uses **Socket.io** for signaling.
* API Endpoints:
  + POST /call/initiate → Initiates a call request.
  + POST /call/accept → Accepts an incoming call.
  + POST /call/reject → Rejects an incoming call.
  + POST /call/end → Ends an active call.
  + GET /call/history/:userId → Fetches call history for a user.

**2. Socket.io Event Handlers (call.js)**

* socket.emit('callRequest', { callerId, receiverId })
* socket.on('callAccepted', { callerId, receiverId })
* socket.on('callRejected', { callerId })
* socket.on('callEnded', { callerId })

**Step 4: WebRTC Implementation for Voice Calls**

**1. WebRTC Peer Connection Setup**

* Establishes direct **peer-to-peer media streaming**.
* Uses **STUN/TURN servers** (if needed for NAT traversal in LAN).
* Ensures **low-latency communication**.

**2. Handling Audio Streams**

* Capture user microphone input.
* Establish WebRTC **media connection**.
* Encode and transmit **audio packets** over LAN.

**3. Managing Call States**

* Maintain call status in **MongoDB (ongoing, ended, missed).**
* Update UI via **real-time Socket.io events.**

**Step 5: Call Notifications & User Presence**

**1. Incoming Call Notification System**

* WebSockets push notifications for incoming calls.
* Displays **real-time caller ID** on recipient’s UI.

**2. Call Status Tracking**

* If unanswered → Mark **missed call** in MongoDB.
* If answered → Update call history.

**Next Steps**

* **Implement frontend integration for voice calls.**
* **Test WebRTC connections in different LAN environments.**
* **Optimize voice call performance (buffering, latency).**

🚀 **Phase 1 complete: Voice calls over LAN using WebRTC & Socket.io!**

lan-voice-call/

│── config/ # Server & Database Configuration

│ ├── db.js # MongoDB Connection

│── models/ # Database Schemas

│ ├── User.js # User Schema

│ ├── Call.js # Call Session Schema

│── routes/ # API Endpoints

│ ├── authRoutes.js # Authentication APIs

│ ├── callRoutes.js # Call Management APIs

│── middleware/ # Middleware for Authentication

│ ├── authMiddleware.js

│── public/ # Static Files (if needed)

│── server.js # Main Entry Point

│── package.json # Dependencies

│── .env # Environment Variables

│── README.md # Project Documentation