ConnectSphere Development Plan

ConnectSphere is a secure, enterprise-grade real-time communication platform for organizations, combining chat, audio/video calls, scheduling, and smart availability into one interface. This document outlines the complete development roadmap, architecture, and learning milestones to build the application from scratch using React (frontend), Node.js (backend), MongoDB (database), and WebRTC (for real-time audio/video).

# 1. Project Goals

- Develop a secure and scalable real-time communication platform for employees.  
- Support text chat, file sharing, audio/video calls, and scheduled meetings.  
- Implement smart features such as availability detection, offline messaging, and reminders.  
- Create a professional-grade portfolio project demonstrating full-stack expertise.

# 2. Tech Stack

Frontend:  
• React (Vite)  
• Tailwind CSS  
• React Router  
• Zustand / Redux Toolkit  
• Socket.IO Client  
• WebRTC  
  
Backend:  
• Node.js + Express  
• Socket.IO  
• MongoDB + Mongoose  
• JWT Authentication  
• Bcrypt for password hashing  
• Zod for validation  
  
Other:  
• Docker for deployment  
• Nginx or Caddy for reverse proxy  
• Coturn for TURN server  
• AWS S3 or local storage for file uploads

# 3. Core Features

1. Secure employee login with JWT authentication.  
2. Real-time messaging (1:1 and group rooms).  
3. File sharing (images, documents, PDFs).  
4. Audio and video calls via WebRTC.  
5. Presence indicators (online, away, in-call).  
6. Scheduled calls with reminders (push, email, in-app alerts).  
7. Calendar integration for meetings.  
8. Notifications 10 minutes before scheduled calls.

# 4. Unique Features

- Smart Availability: Working hours, Do Not Disturb, auto-suggest scheduling if unavailable.  
- Offline Messaging & Video Mail.  
- AI-powered Meeting Summarizer (transcription + highlights).  
- Threaded conversations and reactions.  
- Read receipts and delivery reports.  
- Searchable history with filters.  
- Multi-device sync with push notifications.  
- Admin dashboard with analytics and audit logs.  
- Quick Drop Rooms for temporary meetings.

# 5. Development Milestones

1. 1) Scaffold project structure for frontend and backend.
2. 2) Implement authentication (register, login, JWT).
3. 3) Create user and room models in MongoDB.
4. 4) Build real-time messaging with Socket.IO.
5. 5) Add file upload & storage system.
6. 6) Implement audio/video calling via WebRTC.
7. 7) Create scheduling system with reminders.
8. 8) Integrate push and email notifications.
9. 9) Build search, threads, reactions, and presence.
10. 10) Develop admin dashboard with analytics.
11. 11) Add AI meeting summarization.
12. 12) Testing (unit, integration, E2E).
13. 13) Dockerize and deploy to production.

# 6. Database Schema (High Level)

Collections:  
• Users: { \_id, email, name, passwordHash, avatarUrl, role, lastSeenAt, status }  
• Rooms: { \_id, name, isGroup, participants[], lastMessageAt }  
• Messages: { \_id, roomId, senderId, type, text, fileMeta, createdAt, readBy[] }  
• Calls: { \_id, roomId, callerId, calleeIds[], status, startedAt, endedAt }  
• Schedules: { \_id, title, participants[], callType, scheduledTime, reminderSent }

# 7. Learning Outcomes

- Master React for building complex SPAs with real-time capabilities.  
- Learn state management with Redux Toolkit or Zustand.  
- Understand WebSocket communication via Socket.IO.  
- Gain experience with WebRTC for real-time media.  
- Learn MongoDB schema design and indexing.  
- Implement authentication and authorization best practices.  
- Handle file uploads and storage securely.  
- Deploy full-stack apps with Docker, Nginx, and SSL.  
- Build notification systems (push, email).  
- Write tests for backend and frontend.

# 8. Deployment Plan

1. Set up a VPS (e.g., Hetzner, DigitalOcean).  
2. Install Docker & Docker Compose.  
3. Configure MongoDB container.  
4. Deploy backend API with pm2 inside Docker.  
5. Serve frontend via Nginx container.  
6. Configure Caddy or Nginx reverse proxy with SSL.  
7. Set up TURN server (coturn) for WebRTC.  
8. Set up CI/CD pipeline for automated deployment.  
9. Monitor logs and server health.