Video Conferencing App (Mern)

Introduction:

In an era defined by remote collaboration, digital classrooms, and global connectivity, video conferencing has become a cornerstone of communication. Whether for business, education, or personal interactions, users demand platforms that are fast, secure, and easy to use. Video Connect is a full-stack web application developed to meet these expectations by offering seamless video conferencing through the power of the MERN stack (MongoDB, Express.js, React, Node.js).

This project leverages real-time technologies and modern web development practices to create a robust, scalable, and interactive video communication platform. It enables users to create or join virtual meeting rooms, share screens, send real-time messages, and manage participants — all within a browser.

🔍 Key Objectives:

- To build a scalable video conferencing application using the MERN stack.
- To enable secure, authenticated access to virtual meeting rooms.
- To support real-time audio/video communication using WebRTC and Socket.IO.
- To provide a user-friendly interface for managing meeting participants, chat, and screen sharing.

🎯 Motivation:

- Growing demand for custom and lightweight alternatives to platforms like Zoom or Google Meet.
- The need for educational institutions and small teams to host secure, private meetings.
- Aiming to explore full-stack development practices with real-time media and messaging capabilities.

By integrating JWT-based authentication, real-time sockets, WebRTC, and a modular frontend interface, Video Connect delivers a smooth and responsive conferencing experience. This report details the system design, architecture, implementation flow, and key learnings from the project lifecycle.

Project Description

Video Connect is a real-time video conferencing web application designed to support secure, low-latency communication between users. Developed using the MERN stack, the system integrates frontend interactivity with backend logic and real-time media protocols to deliver a full-featured virtual meeting experience directly in the browser.

At its core, the application enables users to register, authenticate, create meeting rooms, and invite others to join using secure tokenized links. Once inside a room, users can participate in video/audio conversations, share their screens, exchange text messages, and manage session settings like audio/video toggling or raising hands.

📦 **Key Functional Modules:**

- Authentication System
  - JWT-based login and registration
  - Role-based access (Host, Guest)
- Meeting Room Management
  - Create and join rooms with unique IDs
  - Support for persistent and temporary rooms
- Invite Flow
  - Generate short-lived, secure invite tokens
  - Allow guests to join from any device or session
- Media & Communication
  - WebRTC-based video and audio streaming
  - Socket.IO-powered real-time messaging
  - Dynamic video grid and participant status updates
- User Interface
  - Built with React and Tailwind CSS
  - Responsive layout with control bar, modals, and overlays

💡 **Highlights:**

- Fully functional video + audio streaming with peer connections.
- Real-time text chat with optional side panel.
- Screen sharing with presenting mode toggle.
- Participant list with host control indicators.
- Token-based multi-user simulation on localhost (for development testing).

The platform is designed to be modular, extensible, and production-ready. It can serve as the foundation for building internal communication tools, virtual classrooms, or even SaaS conferencing platforms.

Use Case Scenario

To illustrate the functionality and practical value of Video Connect, consider the following real-world scenario involving a remote design review between two professionals:

📘 Scenario: Remote Design Review Between Two Team Members

Priya, a UX designer based in Bengaluru, needs to review a new mobile app interface with Daniel, a frontend developer located in Tokyo. Instead of using commercial video conferencing tools, Priya initiates a secure, private meeting using the Video Connect platform.

1. Priya logs in to the application using her credentials and creates a new meeting room titled “Mobile UI Review.”
2. She generates an invite link for Daniel by entering his email ID. The system issues a one-time secure token link.
3. Daniel receives the invite and opens the link in an incognito browser window on the same device.
4. He is automatically authenticated using the invite token and joins the meeting as a guest user.
5. Both participants enable video and audio, and communicate in real-time.
6. Priya shares her screen to walk Daniel through the new layout using the screen sharing feature.
7. Daniel uses the chat panel to ask questions while keeping the meeting distraction-free.
8. After completing the review, both users leave the meeting, and the room is marked inactive.

👥 Use Case Interaction Summary

Role	Action	Result
Host (Priya)	Creates meeting room	Room ID generated with host as owner
	Generates invite link for guest	Secure token shared via email or chat
	Shares screen	Frontend interface shown to other users
Guest (Daniel)	Joins via invite link	Authenticated temporarily as guest
	Participates in video chat	WebRTC connects peers through Socket.IO
	Sends chat messages	Chat updates in real time

This scenario reflects the core design goals of the project: ease of access, real-time communication, and secure, temporary collaboration, even across accounts and devices. It also demonstrates the platform's readiness for real-world remote use cases like interviews, design reviews, webinars, and online classrooms.

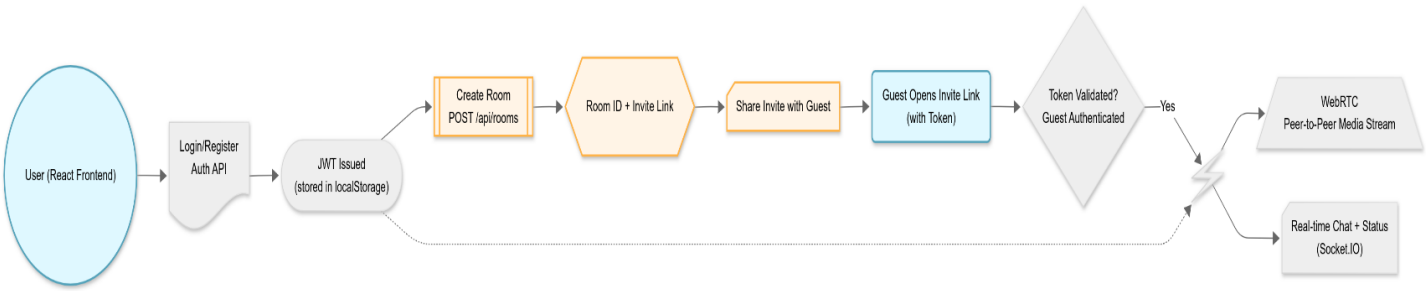
Technical Architecture

Video Connect is structured around the MERN stack with added real-time communication support via WebRTC and Socket.IO. The system follows a modular, service-oriented architecture that separates client logic, backend APIs, and real-time media signaling.

🧱 Architecture Components

Layer	Technology	Responsibility
Frontend	React, Vite, Tailwind CSS	UI rendering, routing, media handling
Auth System	JWT, bcryptjs	User registration, login, token issuance
Backend API	Express.js, Mongoose	Room/user management, REST API, token validation
Database	MongoDB	User data, room info, meeting state persistence
RTC Signaling	Socket.IO	Real-time signaling for peer discovery/connection
Media	WebRTC	Peer-to-peer audio / video / screen streaming

🔁 System Flow (Login → Create Room → Join Meeting)



🔌 Real-Time Communication Flow

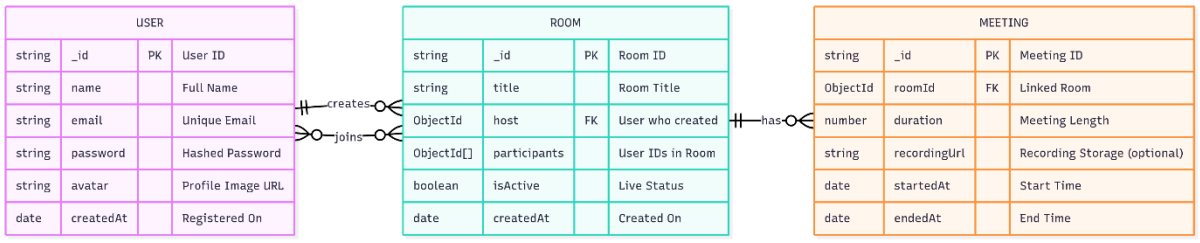
1. Socket.IO establishes a signaling channel between all users in the same room.
2. When a user joins a room:
    a. A join-room event is emitted with roomId and userId.
    b. The server adds the socket to a room and notifies others via user-joined.
3. WebRTC peer connection is established between all users using:
    a. ICE candidates
    b. SDP offer/answer exchange
4. Audio/video streams are attached to video elements.
5. If screen sharing starts, a second media stream replaces the video track dynamically.

🔐 Security Highlights

- All APIs are protected via Authorization: Bearer <token> headers.
- JWT tokens have expiry durations (JWT_EXPIRES_IN) and are signed with a secure secret.
- Invite tokens for guests are short-lived (10 minutes) and scoped to a specific room.
- User data is encrypted at rest via MongoDB's schema-level protections.

E-R Diagram Overview

The Video Connect application manages users, meeting rooms, and real-time sessions. The following entities and relationships model how users interact with rooms and meetings.



🧩 Entities and Attributes

- 👱 User

Field	Type	Description
_id	ObjectId	Unique identifier
name	String	User's full name
email	String	Unique email address

password	String	Hashed password
avatar	String	URL of profile picture
createdAt	Date	Account creation timestamp

- 📁 Room

Field	Type	Description
_id	ObjectId	Unique room ID
title	String	Meeting room title
host	ObjectId	Reference to User
participants	ObjectId	Array of User references
isActive	Boolean	Whether the room is currently live
createdAt	Date	Timestamp of room creation

- 📹 Meeting

Field	Type	Description
_id	ObjectId	Unique meeting ID
roomId	ObjectId	Reference to associated Room
duration	Number	Duration of the meeting (minutes)
recordingUrl	String	URL to saved recording (if supported)
startedAt	Date	Meeting start time
endedAt	Date	Meeting end time

🔗 Relationships

- A User can:
  - Host multiple rooms (1:N)
  - Join many rooms as a participant (M:N)
- A Room must be created by one User (1:1)
- A Room can have many participants (users)
- A Meeting is linked to one Room and may store session metadata

Pre-requisites

Before setting up or running the Video Connect application, make sure the following tools, environments, and system configurations are available. These tools ensure smooth development, testing, and deployment of the MERN-based video conferencing application.

💻 Environment Requirements

- A system with Node.js (v16 or higher)
- A local or remote MongoDB instance
- A modern browser (preferably Chrome or Firefox) that supports WebRTC
- An IDE or code editor (e.g., Visual Studio Code)
- Git (for cloning the repository)

🔧 Tools and Versions

Tool / Tech	Recommended Version	Purpose
Node.js	16.x or above	JavaScript runtime for backend and frontend builds
npm / yarn	npm 8.x or yarn 1.22+	Package management for dependencies
MongoDB	5.x or Atlas	Database for storing user, room, and session data
Vite	Latest	Fast React frontend bundler
React	18.x	Frontend library for UI components
Socket.IO	4.x	Real-time communication for chat/video signaling
WebRTC Support	Native in browser	Peer-to-peer video/audio communication
VS Code	Latest	Development and debugging
Postman / Thunder Client	Any	API testing and debugging (optional)

Setup and Installation

Follow the steps below to set up the project on your local development environment. This guide assumes you have already installed the required tools listed in the Pre-requisites section.

📄 **Step-by-Step Instructions**

1. Clone the Repository

```
git clone https://github.com/SKChauhan17/Video-Conferencing-App.git
cd Video-Conferencing-App
```

2. Backend Setup

```
cd server
npm install              # Install backend dependencies
cp .env.example .env     # Create .env file
```

🖍️ Edit the .env file with your MongoDB URI and JWT secret. Example:

```
MONGO_URI=mongodb://localhost:27017/video-connect
JWT_SECRET=your-secure-secret
JWT_EXPIRES_IN=1d
PORT=5000
CLIENT_URL=http://localhost:5173
```

Start the backend development server:

```
npm run dev
```

3. Frontend Setup

```
cd ../client npm install            # Install frontend dependencies
npm run dev             # Start frontend dev server
```

The app will be accessible at:

🔗 http://localhost:5173 (Frontend)

🔗 http://localhost:5000 (Backend API)

API & Database Verification

- Register a new user via the UI.
- Use Postman or browser dev tools to verify API routes like:
    - POST /api/auth/register
    - POST /api/auth/login
    - POST /api/rooms
    - POST /api/invite

☑ Success! You're Up and Running

You can now:

- Join/create rooms
- Send and accept invite links
- Enable video/audio
- Share screen
- Chat in real-time

🧪 Developer Tips

- Use Incognito mode or multiple tabs to simulate multiple participants.
- For easier development, install:
    - nodemon for backend: npm install -g nodemon
    - Tailwind CSS IntelliSense in VS Code

Folder Structure

The project follows a standard MERN stack separation between frontend and backend. Below is the high-level directory layout with descriptions of major files and folders.

Video-Conferencing-App/

```
├── client/                 # Frontend (React + Vite + Tailwind CSS)
│   ├── public/             # Static assets and favicon
│   ├── src/                # Application source code
│   │   ├── components/     # Reusable UI components (MeetingRoom, Controls, etc.)
│   │   ├── contexts/       # React context providers (Auth, WebRTC, Meeting)
│   │   ├── api/            # Axios instance for API calls
│   │   ├── pages/          # Page-level components (e.g., Login, Dashboard)
│   │   ├── styles/         # Tailwind and custom styles (if any)
```

```
│   │   └── main.tsx          # App entry point
│   ├── index.html            # Root HTML template
│   ├── vite.config.ts        # Vite configuration
│   ├── tailwind.config.js    # Tailwind CSS configuration
│   ├── tsconfig.json         # TypeScript settings
│   └── package.json          # Frontend dependencies and scripts
│
├── server/                   # Backend (Node.js + Express + MongoDB + Socket.IO)
│   ├── config/               # MongoDB connection setup
│   ├── controllers/          # Route handler functions (auth, rooms, invite)
│   ├── middleware/           # Authentication middleware (JWT protect)
│   ├── models/               # Mongoose schemas (User, Room, etc.)
│   ├── routes/               # Express routes (authRoutes, roomRoutes, inviteRoutes)
│   ├── .env.example          # Example environment config file
│   ├── server.js             # Entry point for the backend server
│   └── package.json          # Backend dependencies and scripts
│
├── .gitignore                # Git ignore rules
├── README.md                 # Project overview and setup instructions
```

📌 Notable Conventions

- client/src/contexts: Centralized state management using React Context API.
- client/src/components/meeting: Contains layout elements like video grid, chat, controls.
- server/routes: Each module (auth, room, invite) has its own Express route file.
- server/server.js: Initializes Express, connects to MongoDB, and binds Socket.IO.
- Tokens, room IDs, and user identities are passed securely between client and server.

Application Flow

The Video Connect application is built around a streamlined user experience that transitions smoothly from authentication to room creation, real-time communication, and session termination. This section outlines the step-by-step flow of how users interact with the system and how different components work together.

🔁 Overall Flow Overview

- User visits application
- Registers or logs in via authentication API
- Frontend receives JWT token and stores it (localStorage)
- User creates or joins a meeting room
- If hosting, user invites others using a one-time secure link
- Guests join using the invite link (token in URL)
- Participants connect via Socket.IO and establish WebRTC P2P streams
- Real-time features enabled:
    - Audio/video sharing
    - Screen sharing
    - Chat
- Participant management
- Meeting ends when all users leave or host terminates the session

🧭 **Role-Based Interaction Summary**

Role	Actions
Host	Login → Create Room → Share Invite Link → Start Meeting → Share Screen
Guest	Open Invite Link → Auto-authenticate via token → Join Meeting
Both	Toggle audio/video, chat, raise hand, view participants

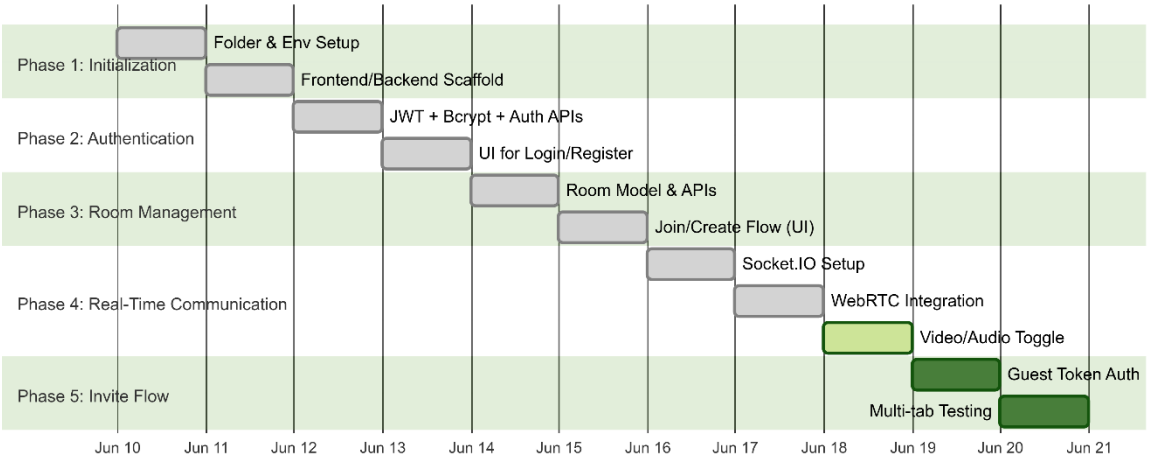
🧠 **Behind the Scenes (Internal Workings)**

- JWT tokens protect all API routes and identify users across tabs/sessions.
- Invite links use short-lived JWT tokens embedded in the URL (?token=...).
- WebRTC manages real-time peer-to-peer streaming.
- Socket.IO enables:
    - Room join/leave notifications
    - Chat broadcasting
    - Participant sync across tabs

- Frontend tracks and updates:
    - Local and remote media streams
    - UI state for audio/video/chat/screen
    - Participant metadata

Project Milestones

The development of Video Connect was structured into iterative milestones to ensure clarity, modularity, and steady progress. Each phase focused on specific features and architectural goals to transform the project from a basic communication system into a functional and scalable video conferencing solution.



☑ Phase 1: Project Initialization

- Set up GitHub repository and folder structure
- Installed core tech stack:
    - React + Vite (frontend)
    - Node.js + Express (backend)
    - MongoDB (database)
- Initialized Tailwind CSS and TypeScript configs
- Created development .env files and environment setup

☑ Phase 2: User Authentication

- Implemented user registration and login APIs with JWT
- Password encryption using bcryptjs
- Created login & register pages with form validation
- Token handling via localStorage
- Middleware: protected routes using authMiddleware.js

☑ Phase 3: Room Management

- Built Room model with host and participants
- API endpoints for:
    - Creating new meeting rooms
    - Joining existing rooms
- Added context-based frontend state to store meeting data
- Integrated React Router for dynamic room navigation

☑ Phase 4: Real-Time Communication

- Integrated Socket.IO on both frontend and backend
- Built WebRTC signaling through sockets
- Enabled:
    - Peer-to-peer video/audio streaming
    - Dynamic video grid layout
    - Remote stream handling
- Setup mute/unmute and video toggle controls

☑ Phase 5: Invite Flow and Guest Join (localhost testing)

- Added secure, short-lived invite link functionality
- Auto-authentication via token in invite URL
- Allowed same-device, multi-user testing (host + guest tabs)