Software Requirements Specification

for

Enhanced Security and Blockchain Based Online Meeting Platform

Version 1.0 approved.

Prepared by - Kishan Agrawal

- Rishika Agarwal

- Shubham Singh

- Yashasvi Saxena

KIET Group of Institutions

11th April 2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This document specifies the software requirements for the Enhanced Security and Blockchain-Based Online Meeting Platform. The purpose of this platform is to provide users with a highly secure, decentralized solution for conducting virtual meetings. The system utilizes blockchain technology to ensure data integrity, privacy, and transparency while preventing unauthorized access and tampering. This SRS focuses on the core modules, including user authentication, meeting creation, blockchain logging, and encrypted communication.

1.2 Document Conventions

The document follows standard IEEE SRS formatting. Section headings are numbered hierarchically (e.g., 1, 1.1, 1.2) for clarity. All functional requirements are presented in numbered lists and use clear, unambiguous language. Key technical terms and system modules are italicized. All security-related features are highlighted to emphasize their significance.

1.3 Intended Audience and Reading Suggestions

This document is intended for developers, testers, project managers, cybersecurity consultants, blockchain engineers, and client stakeholders. Readers unfamiliar with blockchain-based applications may begin with the system overview before reviewing detailed requirements. Developers and testers are advised to focus on the functional and non-functional requirements sections, while project managers and stakeholders may prioritize the scope and use case sections.

1.4 Product Scope

The Enhanced Security and Blockchain-Based Online Meeting Platform is designed to offer a secure alternative to conventional video conferencing systems. It leverages blockchain technology to record meeting metadata immutably, offers encrypted communication channels, and enforces strong user authentication protocols. The platform aims to eliminate vulnerabilities common in traditional meeting tools, such as unauthorized access, data leaks, and tampering of logs. Key features include user registration, meeting scheduling, blockchain-based logging, and end-to-end encrypted media exchange

2. Overall Description

2.1 Product Perspective

The decentralized online meeting platform described in the Software Requirements Specification (SRS) is a product designed to provide secure, private, and scalable communication via video conferencing. It is developed from the ground up to address the growing demand for secure and privacy-compliant online meetings, especially in sectors like education, healthcare, government, and businesses.

2.2 Product Functions

The decentralized online meeting platform is designed to provide users with a secure, scalable, and user-friendly environment for hosting and attending virtual meetings. It leverages blockchain for authentication and immutable logging, WebRTC for real-time communication, and IPFS for decentralized storage. The system supports various user roles and ensures data privacy through end-to-end encryption. Key functions include:

- User authentication and role-based access control
- Scheduling and managing meetings.
- Real-time video/audio communication
- Secure file sharing and storage
- Immutable meeting logs
- Notifications and alerts
- Access to recordings and meeting data
- Multi-device support with intuitive UI

2.3 User Classes and Characteristics

The decentralized online meeting platform is intended for a wide range of users with varying levels of expertise, responsibilities, and usage patterns. Users are classified based on their roles, technical skills, and frequency of use. Each class interacts with specific platform features tailored to their needs and privilege levels.

1. Administrative Users

- **Role:** System administrators and IT staff
- **Characteristics:** High technical expertise, full access to all system features, including user management, logs, configuration, and compliance settings
- **Importance:** Critical for setup, maintenance, and security

2. Organizational Users

- **Role:** Managers, faculty, team leads
- **Characteristics:** Moderate technical expertise, frequent use for scheduling and hosting meetings, access to recordings and logs

- **Importance:** High, as they ensure smooth platform operation and drive engagement

3. General Participants

- **Role:** Students, employees, guests
- **Characteristics:** Low to moderate technical knowledge, limited access (join meetings, view shared files, participate in chats)
- **Importance:** Moderate, must be ensured a seamless and accessible user experience

4. Guest Users

- **Role:** One-time or external attendees
- **Characteristics:** Minimal interaction, temporary access with strict permissions
- **Importance:** Low, but their experience must still meet usability and security standards

2.4 Operating Environment

The decentralized online meeting platform is designed to operate in a cross-platform environment, web applications. It is built for compatibility, performance, and secure communication across a range of devices and systems. The software must coexist with various external services and libraries to deliver its full functionality.

Operating Environment:

- **Hardware Platforms:**
  - User devices: Smartphones, tablets, laptops, desktops
  - Requirements: Camera, microphone, speaker/headphones, internet connectivity
- **Operating Systems:**
  - Web: Compatible with latest versions of Chrome, Firefox, Safari, Edge
- **Software Dependencies:**
  - WebRTC for real-time communication
  - IPFS for decentralized storage
  - Blockchain SDKs (e.g., Solana Web3.js) for identity and logging
  - OAuth 2.0 for authentication
  - MongoDB Atlas for structured data storage
  - SRTP and AES-256 for secure media transmission
  - RESTful APIs for integration with third-party apps like Google Calendar

The platform must function reliably in environments with varied bandwidth and device performance while maintaining high security and compliance standards.

2.5 Design and Implementation Constraints

The development of the decentralized online meeting platform is subject to several constraints that influence design choices, technology selection, and implementation strategies. These constraints ensure the system remains secure, compliant, and maintainable while meeting user and organizational expectations.

Key Constraints:

- **Regulatory Compliance:**
  - Must adhere to GDPR and ISO/IEC 27001 for data privacy and security.
  - Use encryption standards such as AES-256.
- **Security Considerations:**
  - Mandatory end-to-end encryption for all media and data.
  - Role-based access control and blockchain-based identity verification.
  - Secure handling of user credentials and audit logging.
- **Technology Stack Requirements:**
  - Use of specific open-source libraries and technologies: WebRTC, IPFS, OAuth, Solana blockchain, MongoDB Atlas.
  - Development in JavaScript using frameworks like React.js and Node.js.
- **Hardware Limitations:**
  - Must operate on low-power or older devices with limited processing capability.
  - Minimal local storage usage; preference for cloud and decentralized networks.
- **Communication Protocols:**
  - Real-time media over SRTP and data transmission via HTTPS and WebSocket.
  - Peer-to-peer communication using WebRTC with STUN/TURN servers.
- **Interface Requirements:**
  - Seamless integration with third-party APIs (e.g., Google Calendar, OAuth providers).
  - RESTful API design standards for interoperability.
- **Maintenance and Coding Standards:**
  - Code must be modular, well-documented, and follow standard naming conventions.
  - Should allow easy handover and updates by internal IT or external developers.

These constraints shape both the architecture and implementation of the platform, ensuring it is secure, reliable, and compliant from the ground up.

2.6 Assumptions and Dependencies

These factors are considered while defining the system's functionality and performance expectations. Any deviation from these assumptions could impact project delivery or system performance.

Assumptions:

1. **Platform Availability**
   It is assumed that the required blockchain network (e.g., Solana), IPFS nodes, and WebRTC-supported devices will be available throughout development and deployment.

2. **Third-Party Integration Stability**
   It is assumed that third-party APIs and services such as OAuth (for authentication), Calendly and Google Calendar (for scheduling), and Cloudinary (for file handling) will remain stable and maintain backward compatibility.

3. **Reliable Infrastructure**
   It is assumed that users will have access to stable, high-speed internet and that reliable cloud infrastructure will be available for system hosting and data synchronization.

4. **User Competency**
   It is assumed that end users possess a basic level of digital literacy, including the ability to operate devices with video/audio capabilities and navigate web/mobile interfaces.

5. **Device Compatibility**
   It is assumed that users will access the platform using hardware with adequate processing power, updated browsers, and necessary features (e.g., camera, microphone, encryption support).

Dependencies:

1. **Blockchain Network**
   The system depends on the availability and responsiveness of the blockchain network for identity verification, immutable logging, and access control.

2. **IPFS and Decentralized Storage**
   System performance and storage security rely on IPFS or compatible decentralized file systems for handling meeting recordings and shared files.

3. **WebRTC and Media Encryption Protocols**
   Real-time communication features depend on the WebRTC protocol stack and SRTP for secure transmission of audio/video streams.
4. **OAuth Authentication Providers**
   The authentication flow is dependent on OAuth-based identity providers and the security of the credential validation process.

5. **Cloud Service Providers**
   The backend relies on cloud services for real-time synchronization, scalability, and resilience.

3. External Interface Requirements

3.1 User Interfaces

1. Design Approach

- Clean, responsive UI using Tailwind CSS.
- Consistent layout across devices (desktop, tablet, mobile).
- Optional dark/light mode for accessibility.

2. Common UI Elements

- Navbar: Home, Start Call, Wallet Connect, Call History, Help, Logout.
- Buttons: Start Call, End Call, Connect Wallet, Submit, Back, Cancel.

3. Key Screens

- Login: Connect wallet (MetaMask, Phantom).
- Dashboard: User info, recent activity, call options.
- Call Screen: WebRTC video, mute/camera/end call, blockchain call verification.
- Call History: List of previous calls with timestamps and hashes.
- Popups: Toasts for success, warnings, errors.

4. GUI Standards

- Consistent header/footer.
- Icons from Lucide icons.
- Status indicators (network, call, blockchain).
- Tooltips.

5. Error Display

- Toast or Sonner-style alerts.
- Clear messages with possible solutions.

3.2 Hardware Interfaces

The following are the logical and physical characteristics of the interfaces between the software and the hardware components:

1. Supported Device Types

- Desktops and Laptops (Windows, macOS, Linux)
- Mobile Devices (Android, iOS – via compatible browsers)
- Devices must have:
  - Webcam
  - Microphone
  - Stable internet connection

2. Nature of Interaction

- The software interacts with hardware (camera/mic) using WebRTC APIs to capture and transmit real-time audio and video data.
- Access to these devices will require user permission via the browser.

3. Data and Control Flow

- Input Devices: Webcam and mic stream data into the application during calls.
- Output Devices: Speakers/headphones and screen used for video/audio output.
- Software controls start/stop/mute actions through browser-level hardware APIs.

4. Communication Protocols

- WebRTC: For peer-to-peer media communication between devices.
- HTTPS & WebSockets: For secure signaling, metadata exchange, and blockchain interactions.
- Blockchain Protocols (e.g., Solana RPC or Ethereum JSON-RPC): For logging/verifying call data on-chain.

3.3 Software Interfaces

This section outlines the connections between this product and other software components, including tools, databases, libraries, and protocols.

1. Operating Systems

- Compatible with:
  - Windows 10+
  - macOS
  - Linux (Ubuntu preferred)
  - Mobile OS: Android (Chrome), iOS (Safari)

2. Databases

- NoSQL Database: MongoDB

    - Used to store user profiles, call metadata, and call history (non-sensitive data).
    - Incoming data: user info, call start/end times, wallet address.
    - Outgoing data: fetched for display in user dashboards and call history.

3. Blockchain Network

- Solana (preferred)
    - Used to store hashed metadata of calls for security and verification.
    - Interacts via Solana Web3.js.
    - Purpose: call verification, user identity verification.

4. Web Libraries / APIs

- WebRTC
    - Handles real-time audio/video communication between peers.
    - Used for stream capture, peer connection, and media control.
- MetaMask / Phantom Wallet
    - Browser wallet extensions for authentication and signing blockchain transactions.
- Solana Web3.js
    - Used for blockchain interaction (send/read transactions, connect to wallet).
- React (v18+), Node.js, Tailwind CSS
    - Core frontend and backend technologies.

5. Communication Flow

- Incoming Messages:
    - Wallet connection request
    - Video/audio stream data via WebRTC
    - Call metadata from users (e.g., call start time)
- Outgoing Messages:
    - On-chain transactions (call log, verification)
    - Real-time media to peers via WebRTC
    - Status updates via WebSockets

6. Shared Data

- Shared data includes:
    - Wallet address

    - Call metadata (duration, timestamps, hash)
    - Blockchain transaction hashes
- Shared via:
    - REST API (internal)
    - WebSocket (real-time updates)
    - Blockchain RPC endpoints (external)

7. Implementation Constraints

- WebRTC access to hardware must follow browser permission standards.
- Blockchain interaction must use async non-blocking calls.
- Wallet data is accessed only with user consent.
- Secure communication must be over HTTPS/WebSockets only.

3.4 Communications Interfaces

This section describes the communication-related functions and protocols required for the enhanced video calling platform.

1. Communication Channels & Functions

- Web Browser Communication:
    - All interactions occur via modern browsers supporting WebRTC and JavaScript ES6+.
    - Compatible browsers: Chrome, Firefox, Safari, Edge (latest versions).
- Peer-to-Peer Media Transfer:
    - WebRTC used for real-time audio/video streaming between users.
    - Media stream is direct (peer-to-peer), not stored or routed via central server.
- Blockchain Communication:
    - Uses Solana RPC for sending/fetching transactions.
    - Wallet interactions through MetaMask or Phantom extensions.
- Backend Communication:
    - REST APIs (Node.js-based) used for data fetching, storing metadata, etc.
    - WebSocket protocol used for real-time status updates (e.g., call state, blockchain status).

2. Communication Protocols

- HTTP/HTTPS: All browser-to-server communications must use HTTPS.

- WebRTC (DTLS, SRTP): Secure peer communication via DTLS and SRTP for media encryption.
- WebSocket: Full-duplex communication for real-time sync.
- Blockchain RPC (HTTPS): Secure interaction with blockchain nodes via RPC endpoints.

3. Message Format

- JSON format for all API requests and responses.
- Blockchain transactions follow the structure defined by the respective SDKs (e.g., Solana Web3.js).
- WebRTC uses SDP and ICE candidates for signaling.

4. Security & Encryption

- End-to-End Encryption for video and audio via WebRTC.
- JWT (JSON Web Token) used for user session authentication.
- Blockchain interactions signed using wallet private keys (via MetaMask/Phantom).
- All communication encrypted using TLS/SSL.

5. Data Transfer & Sync

- Video/audio data transfer via WebRTC ranges between 500 kbps to 2 Mbps based on connection quality.
- Blockchain sync is asynchronous, handled via transaction confirmations and hash responses.
- Real-time events (call join/leave) synchronized using WebSockets.

4. System Features

Each feature includes description, priority, behavior, and functional requirements.

4.1 User Authentication & Role-Based Access Control

4.1.1 Description and Priority

A high-priority feature that handles login via OAuth, blockchain-based identity verification, and manages roles (Admin, Host, Participant).

4.1.2 Stimulus/Response Sequences

DeMeet Landing Page → User opens login page → Submits credentials → System verifies with OAuth and blockchain → Token is returned.

Admin logs in → Can assign roles → Blockchain verifies role mapping.

4.1.3 Functional Requirements

- Authenticate users using Google OAuth and blockchain signature verification.
- Assign user roles (admin, host, participant) using smart contracts.
- Deny access to unauthorized users and log attempts.
- Use JWT for access tokens with expiry.

4.2 Meeting Scheduling and Management

4.2.1 Description and Priority

High priority. Enables hosts to create, modify, and cancel meetings. Ensure role-based rights and access to information.

4.2.2 Stimulus/Response Sequences

- Host schedules meeting → Details (chats and recordings) stored in DB → Blockchain transaction is created.
- Participant joins meeting → Access validated against role and time.

4.2.3 Functional Requirements

- Create meetings with title, date/time, participant list.
- Generate blockchain ID for every meeting.
- Send notifications and reminders via email/in-app, integration of Calendly.
- Log changes to meeting data

4.3 Real-Time Audio/Video Communication

4.3.1 Description and Priority

Critical functionality using WebRTC with SRTP encryption for real-time media.

4.3.2 Stimulus/Response Sequences

- Participant joins meeting → Token verified → WebRTC stream starts with SRTP.
- Connection drop → System retries signaling.

4.3.3 Functional Requirements

- Start secure video/audio stream on meeting join.
- Encrypt all media using SRTP and DTLS

- Maintain latency under 300ms.
- Auto-reconnect on network loss.

4.4 Secure File Sharing and Meeting Recordings

4.4.1 Description and Priority

High priority. Allows file sharing and meeting recording, storing via IPFS.

4.4.2 Stimulus/Response Sequences

- User uploads file during meeting → File hashed → Stored in IPFS → Metadata saved to blockchain.

4.4.3 Functional Requirements

- Upload file, hash via SHA256, store on IPFS
- Retrieve file via hash and validate access permissions.
- Record meetings and store securely
- Prevent unauthorized downloads.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- 95% of authentication and scheduling responses < 2 sec
- Real-time video latency ≤ 300ms
- Simultaneous support for 100+ participants
- Blockchain write and verify ops ≤ 2 sec per transaction

5.2 Safety Requirements

- Enforce timeout on inactive sessions.
- Limit participants count to prevent server overload.
- Prevent DOS attacks via rate limiting and CAPTCHA.

5.3 Security Requirements

- End-to-end encryption via SRTP
- Use smart contracts for role assignment.
- Multi-factor authentication support

5.4 Software Quality Attributes

- **Reliability**: Auto-reconnect and session recovery
- **Maintainability**: Modular backend with logging and testing hooks
- **Usability**: Easy to Use and Interactive UI
- **Portability**: Works on any browser.

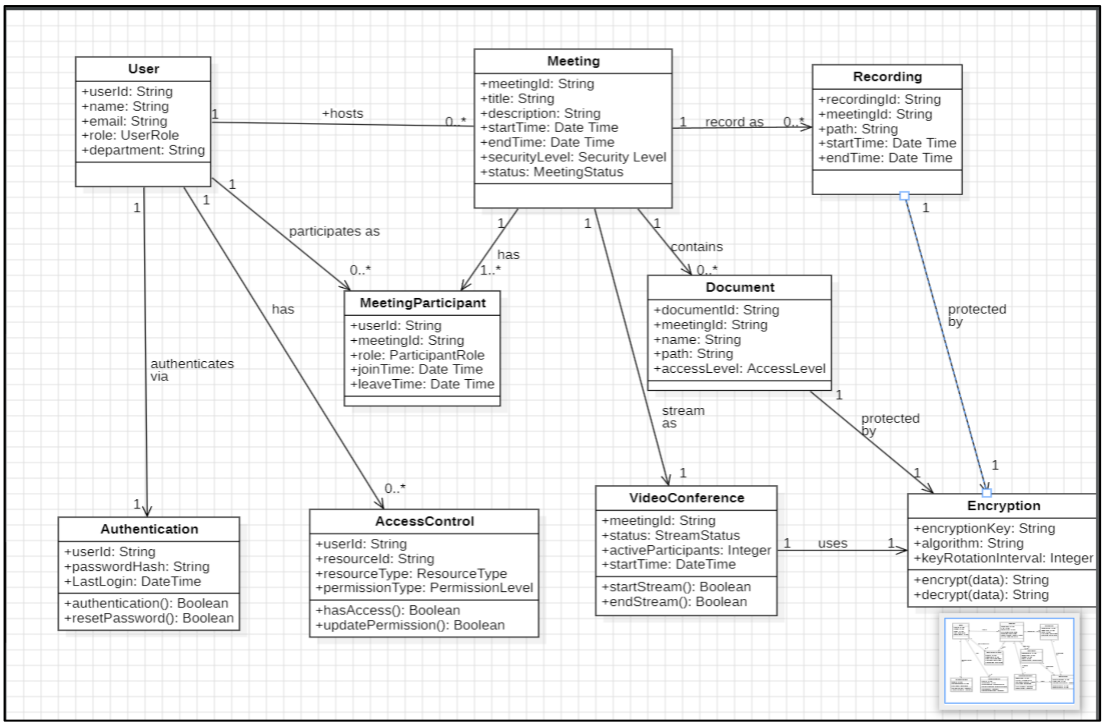
5.5 Business Rules

- Only hosts can start/stop meetings.
- Only admins can assign roles.
- Blockchain log must be created for each critical action (meeting created, file uploaded, etc.)
- File deletion is only logical (actual content remains immutable unless IPFS node purges)

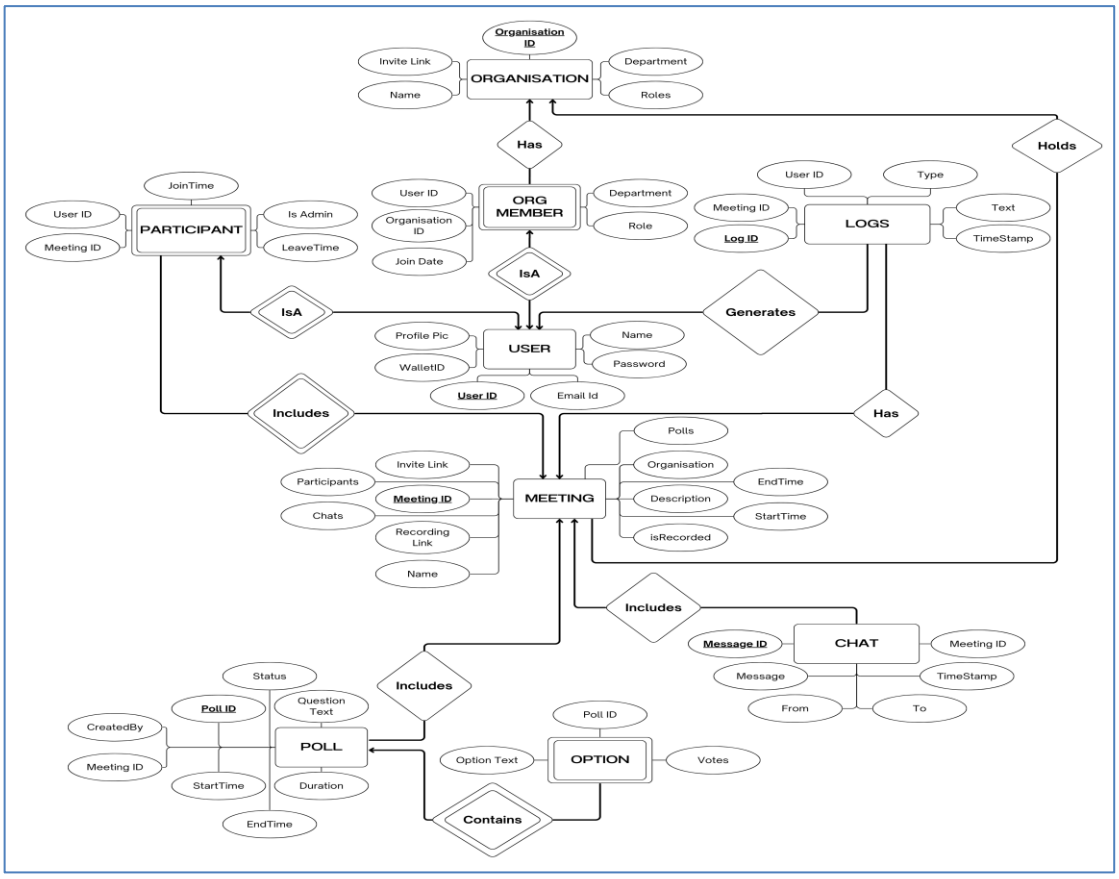
Appendix A: Glossary

Term	Definition
SRS	Software Requirements Specification – A document describing the system's functionality and constraints.
WebRTC	Web Real-Time Communication – A technology for peer-to-peer video/audio communication via web browsers.
Blockchain	A decentralized, distributed ledger technology used for secure and immutable record-keeping.
Solana	A high-performance blockchain platform known for fast and low-cost transactions.
Smart Contract	A self-executing program deployed on the blockchain that automatically enforces rules and agreements.
MetaMask	A popular browser extension wallet used for interacting with the Ethereum blockchain.
Phantom	A browser wallet primarily used for interacting with the Solana blockchain.
WebSocket	A communication protocol for full-duplex communication over a single TCP connection.
RPC	Remote Procedure Call – Used to communicate with blockchain nodes.
UI	User Interface – The visual part of the application that users interact with.
UX	User Experience – The overall experience and ease of use provided to users.
JWT	JSON Web Token – A secure way to transmit user authentication data.
API	Application Programming Interface – A set of functions that allow different software systems to communicate.
HTTPS	Hypertext Transfer Protocol Secure – Protocol for secure communication over the web.
DTLS/SRTP	Datagram Transport Layer Security / Secure Real-time Transport Protocol – Used in WebRTC for secure media transmission.

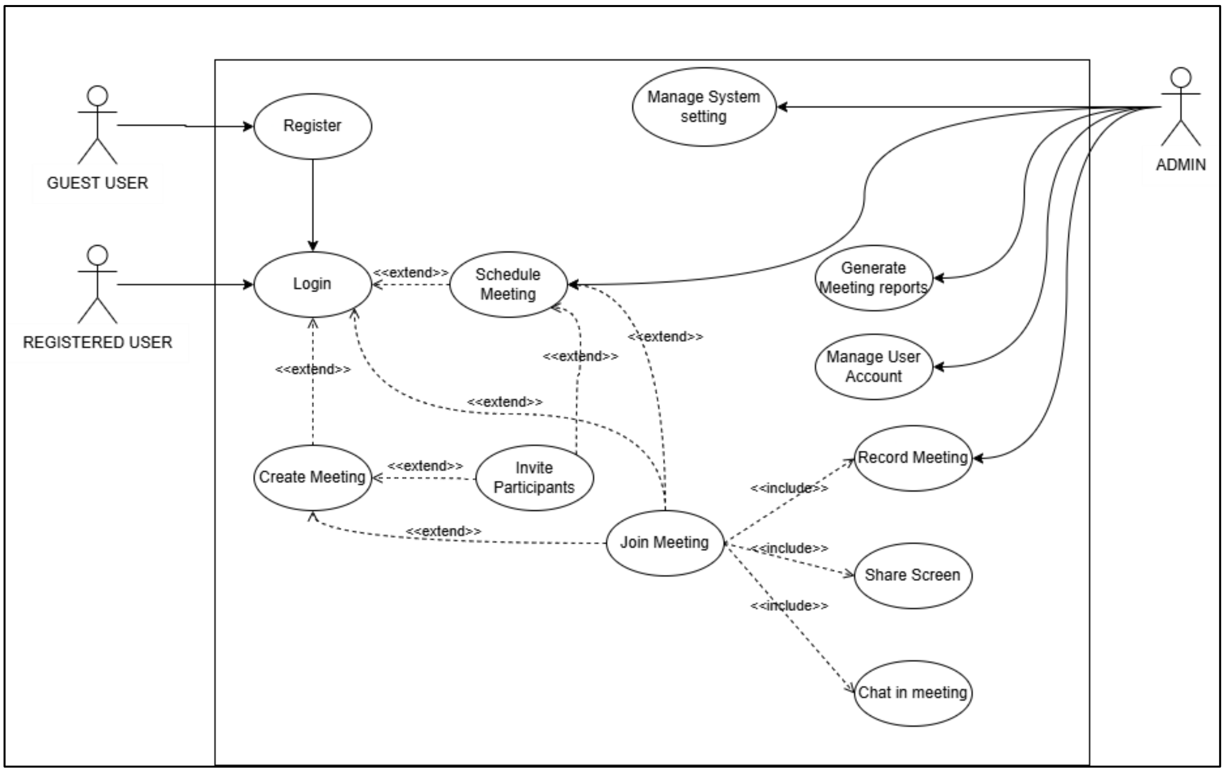
Appendix B: Analysis Model



Class Diagram



Entity Relationship Diagram



Use case Diagram

Appendix C: To Be Determined List

TBD Item	Description	Expected Resolution Time
Wallet Support Scope	Final list of supported wallets (e.g., MetaMask, Phantom, WalletConnect).	Before UI implementation
Call Metadata Structure	Specific metadata fields to be stored on-chain for each video call.	During smart contract design
Cross-Platform Support	Whether a mobile app version will be developed alongside the web version.	To be discussed after MVP
Scalability Tools	Use of external services like Layer-2 solutions for large-scale adoption.	After performance testing