A Full-Stack Real-Time Messaging and Media Sharing Platform

Junpeng Li, Zhuangzhuang Cui

Abstract

This project implements a real-time secure communication platform called *Youchat*, designed to support common communication needs such as text messaging, friend and group management, file sharing, and one-to-one video calls. The system follows a front-end–back-end separation model: the front end provides a three-panel interface similar to typical chat applications, while the back end exposes core APIs for user, conversation, and message management, and ensures persistent data storage.

On the technical side, the front end is built with React and Vite, and interacts with the back end through a unified HTTP client and a Socket.IO connection. The back end uses Express and Socket.IO to offer both RESTful endpoints and real-time communication channels. PostgreSQL is used to store users, conversations, messages, file metadata, and logs, with Docker providing a convenient way to launch and manage the database environment. Text messages are delivered in real time via Socket.IO, while one-to-one video calls rely on WebRTC for media transmission, with signaling handled by Socket.IO and connectivity assisted by Google's STUN servers.

For security, the system adopts a multi-factor authentication mechanism based on email verification codes. All protected endpoints use JWT for authentication, passwords are stored using secure hashing, and file upload and download operations include proper login and permission checks. The system also includes a dashboard that displays user counts, conversation counts, message volume, online users, and message trends, making system activity easier to observe and understand.

1 Introduction

In this project, we designed and implemented a real-time communication platform called Youchat, inspired by the functionality of everyday instant messaging tools. Our goal was to create a system that could simultaneously support several common communication needs: one-to-one chats, small group conversations, lightweight file transfers, and one-to-one video calls when needed.

Users begin by registering an account, setting a password, and completing login through an email-based verification step. Once authenticated, they can enter the main interface, add contacts, create conversations, send text messages and files, and initiate video calls within private chats. In addition, the system includes a dashboard that displays statistics such as the number of users, conversations, messages, files, and currently active sessions, providing an accessible overview of system activity for both regular users and those interested in monitoring overall usage.

2 System Features

2.1 Account and Login Process

(1) The system allows new users to register by creating an account and setting a password. Regardless of whether the user is registering or logging in, an additional email verification step is required. This MFA process helps reduce account takeover risks.

(2) Existing users can log in using their credentials. Upon successful authentication, the system issues a time-limited access token, which is then used to identify the user in subsequent requests.

2.2 Contact and Group Management

(1) Users can search for other users by email or user ID.

(2) Users may send and respond to friend requests, including accepting or declining them.

(3) Users can create group chats, invite friends to join.

(4) The conversation list displays both private and group chats, with optional filtering or search tools for easier navigation.

2.3 Text Messaging and Message History

(1) Text messages sent within a conversation appear in real time on the interfaces of participating users.

(2) When users refresh the page or re-enter a conversation, the system retrieves a segment of historical messages and displays them in chronological order.

(3) Message content, sender information, and timestamps are stored persistently, allowing users to review past conversations when needed.

2.4 File Sharing Capabilities

(1) Users can upload files within conversations, and the system enforces a maximum file size limit.

(2) Uploaded files are associated with the corresponding conversation. Other participants can view the file entries in the message list and download them.

(3) Basic metadata—such as file name, size, associated conversation, and storage path—is recorded to support effective management.

2.5 One-to-One Video Calling

(1) Within private chats, users may initiate a video call request.

(2) The system provides the necessary signaling mechanism to help both sides establish a media connection.

(3) During the call, both users can see each other's video streams and communicate through audio.

(4) When the call ends, both the frontend and backend clean up the corresponding connection and interface state.

2.6 Basic Security and System Status Display

(1) The login process adopts multi-factor authentication, combining a password with an email verification code.

(2) Protected resources require authentication checks before access is granted.

(3) File download operations include permission verification to ensure proper access control.

(4) A dashboard presents statistics such as the number of users, conversations, messages, files, and active sessions, enabling a clearer view of overall system activity.

3 System Analysis and Design

3.1 Overall Architecture and Full-Stack Design

The system follows a client–server architecture with a clear separation between the front end and back end.

From a functional standpoint, the front end is responsible for interface rendering, route transitions, state management, and initiating both HTTP requests and long-lived connections to the server. The back end handles business logic, database operations, authentication and authorization, email verification, real-time connection management, and exposes the APIs required by the front end.

From a full-stack design perspective, the front end is built as a single-page application using React and Vite. All HTTP requests to backend /api/... routes are issued through a unified client wrapper, and real-time communication is established through a socket.io client, which is used for receiving live messages and exchanging video-call signaling data.

The backend is implemented using Node.js with Express to provide REST endpoints, while Socket.IO maintains real-time channels. PostgreSQL serves as the persistent data store, and Docker is used to manage and bootstrap the database environment, ensuring easy deployment and consistent setup across machines.

The following figure shows the architecture of the project:

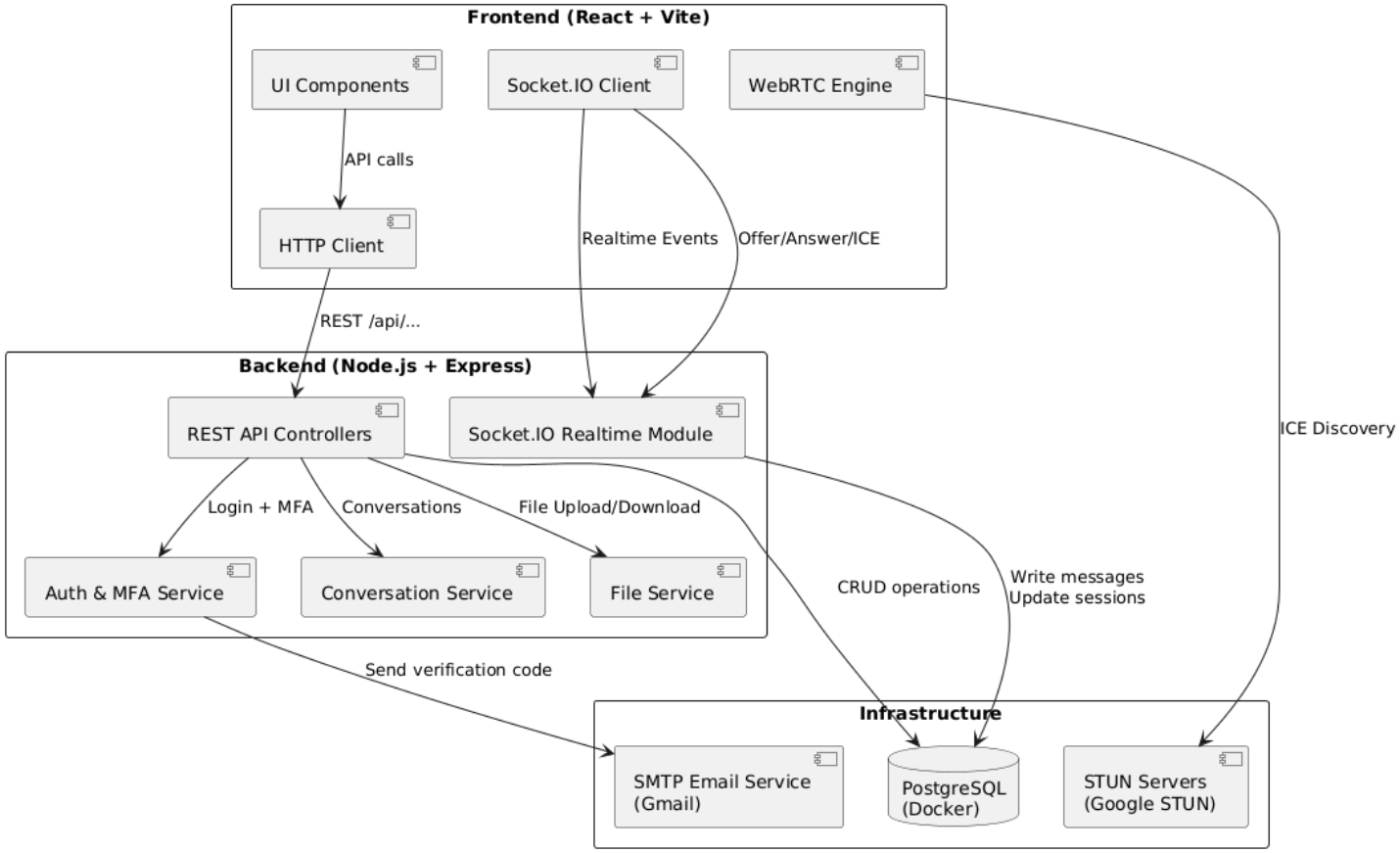

Figure 1 Youchat System Architecture

3.2 Backend Architecture Design

In designing the backend, our goal was to maintain a clear and structured request-handling flow rather than into one layer. The final architecture adopts a layered model:

(1) the outermost layer receives requests and returns responses,
(2) the middle layer organizes business logic according to functional domains
(3) the foundational layer interacts with the database and real-time communication infrastructure.

When the front end issues an HTTP request, it first passes through a unified middleware that checks authentication status and validates basic request information. The request then enters the appropriate controller, which selects a corresponding business module based on the API path and parameters. These modules implement functions such as user registration and login, friend and conversation management, file uploading and downloading, and dashboard statistics. Inside each module, the system accesses the database to read or update users, friends, conversations, messages, files, and logs, and notifies the real-time layer whenever updates need to be propagated to online clients.
Real-time communication is handled by a dedicated Socket.IO module that manages connections and events. When a user establishes a Socket.IO connection, the back end validates the provided token during the handshake phase and attaches the associated user identity to the connection. Afterwards, incoming events—such as new messages, conversation updates, or video-call signaling—are routed through this module.
Even though REST endpoints and socket events are two different interaction channels, they share the same authentication rules and data sources, ensuring a unified backend behavior.

3.3 Frontend Interface and Interaction Design

The frontend is built as a single-page application, presenting the entire system as a continuous environment in which different functional areas are displayed through route transitions. React organizes the interface into modular components, such as the login form, verification-code input page, chat window, conversation list, contact list, and dashboard elements. These components are combined into complete pages depending on the current route.
The main pages include:

(1) a login page for entering account credentials,
(2) an MFA page for email verification,
(3) the primary chat interface for daily communication, and
(4) a dashboard view for observing overall system activity.

The chat interface adopts a three-panel layout: navigation on the left, conversation and contact lists in the center, and the active chat window on the right. Users can browse message history, send new messages, upload files, and initiate video calls within private conversations.

The dashboard presents statistics such as user count, conversation count, message count, file count, and the number of online users, along with basic trends over time.

To maintain consistent data across pages, the front end uses a centralized state store that keeps track of the logged-in user, authentication token, conversation list, and selected conversation. All HTTP requests automatically include the token through a unified client wrapper. Real-time behavior is supported by a socket.io client: whenever new messages, friend updates, or video-call signaling events arrive, the affected components receive updates and re-render accordingly.

4 Implementation Details and Methodology

4.1 MFA Email Configuration

The multi-factor authentication process relies on sending a verification code to the user's email, so the backend must provide a reliable email-delivery channel. In this project, we configured Gmail's SMTP service and set up a dedicated

Youchat email account as the sender. A separate configuration file on the backend is used to store the necessary SMTP parameters.

During login, once the user's email and password pass initial validation, the backend generates a six-digit verification code, stores it as a "challenge record" in the database, and sends the same code to the user's email. The following figure shows the email with verification code:

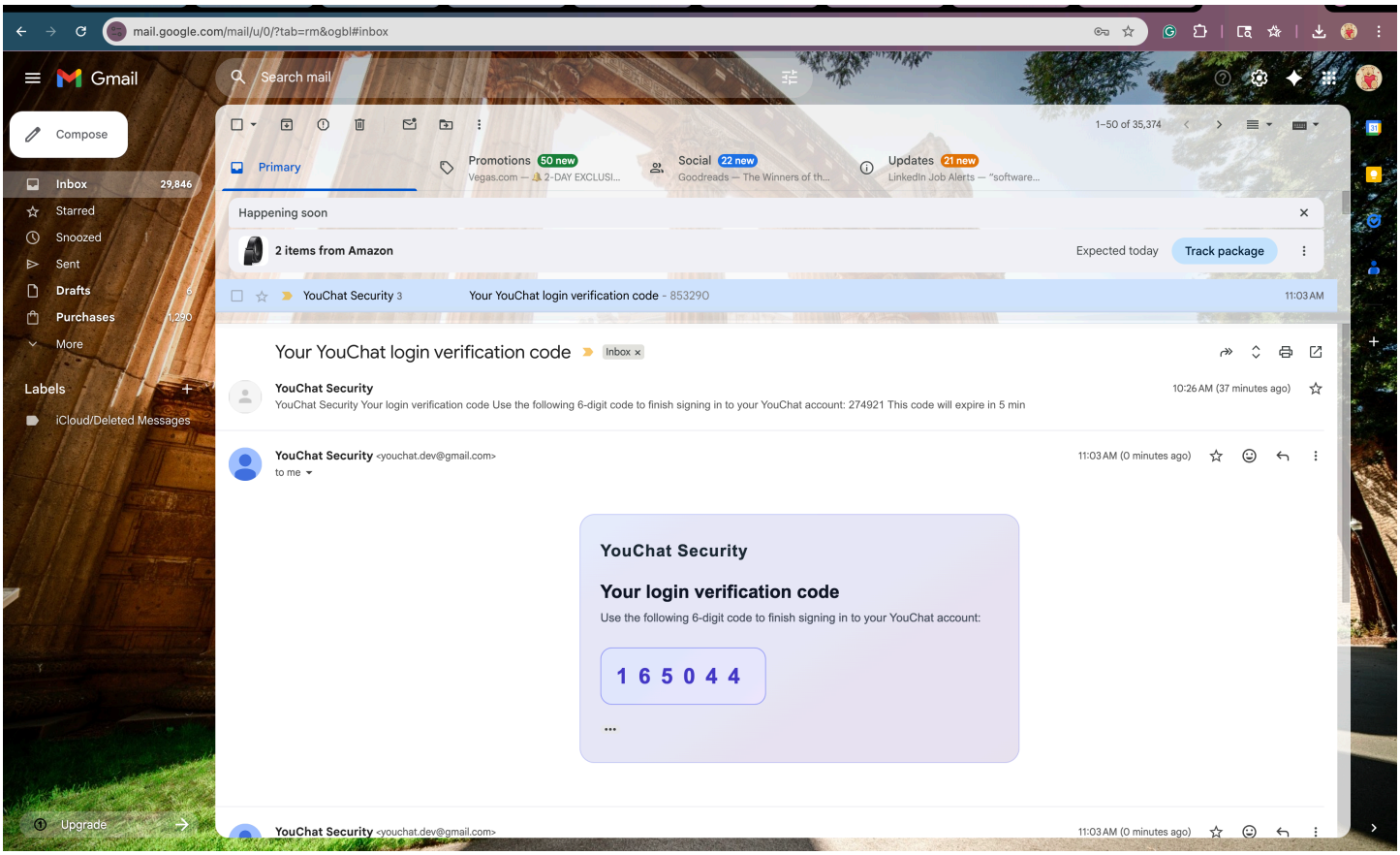

Figure 2 Youchat System Architecture

When the user submits the code on the MFA page, the backend checks whether the code matches the stored value and whether it is still valid.

The following figure shows the MFA login flow:

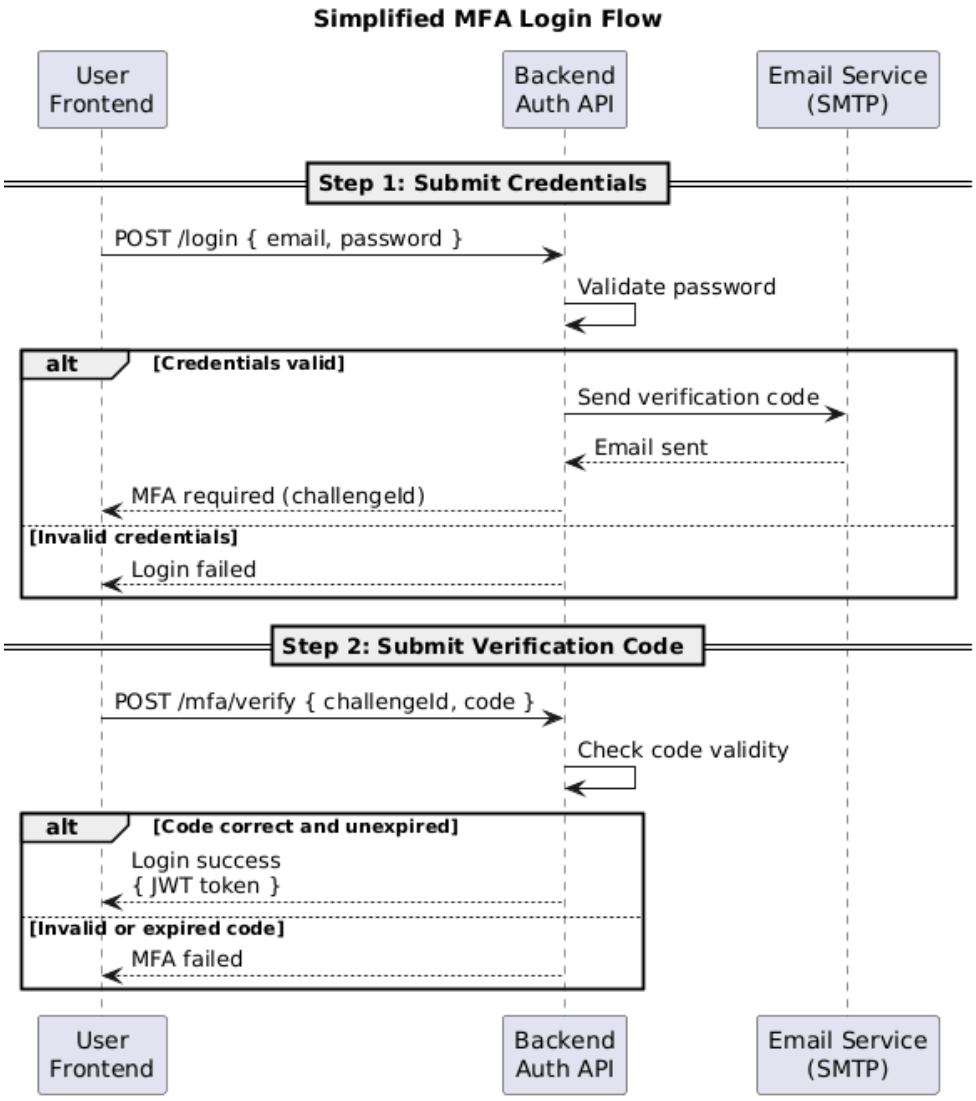

Figure 3 Youchat System Architecture

4.2 Deploying and Managing PostgreSQL with Docker

For database management, we chose to run PostgreSQL inside a Docker container, using a *docker-compose.yml* file to describe the container configuration. This approach offers two key advantages:

(1) PostgreSQL does not need to be manually installed on every development machine.
(2) As long as Docker and the compose file are available, the same database environment can be started consistently.

Database files are stored in Docker volumes, ensuring that data persists across container restarts or removals.

The *docker-compose.yml* file specifies details such as the PostgreSQL image, initialization variables, port mappings, and volume mappings.

4.3 Backend Support for One-to-One Video Calls

In one-to-one video calls, the actual audio and video streams are transmitted directly between browsers through WebRTC, but the backend still plays an important role in signaling and identity verification.

After the user logs in successfully, the browser establishes a Socket.IO connection with the backend, during which the server validates the user's token and associates the user ID with the connection. When a user initiates a video call in a private chat, the frontend sends an event containing the target user's ID and the local SDP description. The backend then locates the corresponding socket for the target user and forwards the signaling message.

Throughout the call setup, both users exchange SDP descriptions and ICE candidates through Socket.IO, enabling WebRTC to determine a viable connection path.

The frontend uses Google STUN servers when creating the WebRTC connection, helping each browser discover its public-facing network address. When the call ends, the frontend sends a termination event to the backend, which forwards it to the other participant so both sides can properly close their WebRTC sessions and update their interfaces.

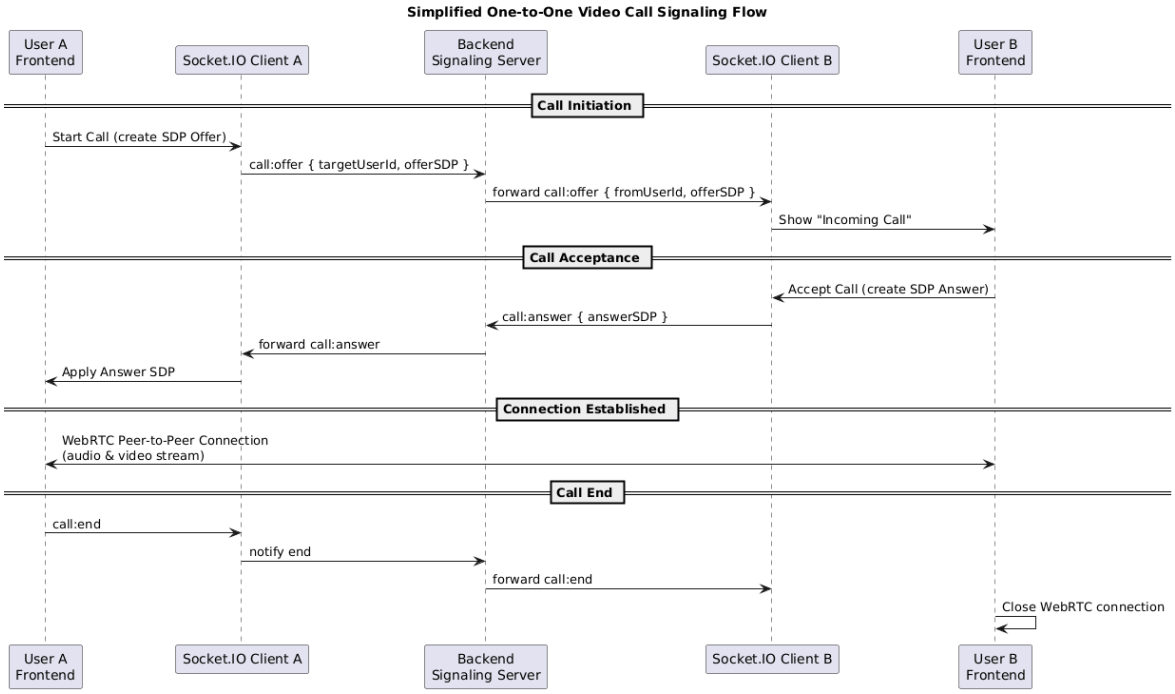

Figure 3 One to one Video Call

And the video call demo is as following:

Figure 4 One to one Video Call

5 Testing and User Manual

For the testing phase, we primarily adopted a black-box approach to verify whether the system behaved as expected. Backend endpoint testing was conducted using **supertest** to issue HTTP requests directly, with **vitest** serving as the testing framework. Each test case involved sending actual requests and examining the returned payload and status codes to determine whether the corresponding functionality operated correctly. For example, in authentication-related tests, we constructed scenarios with correct credentials, incorrect passwords, valid verification codes, invalid codes, and expired codes, and checked whether the login and MFA endpoints responded appropriately.

For chat- and conversation-related features, we tested by invoking endpoints for creating conversations, sending messages, and retrieving message history, ensuring that message content and quantity aligned with expectations. Additionally, we opened multiple browser windows during manual testing to observe whether real-time messages were synchronized correctly across clients.

File-sharing tests included uploading valid files, attempting to upload files exceeding the size limit, and downloading existing files, allowing us to confirm that file size constraints and permission checks functioned properly.

For one-to-one video calls, the focus was on verifying that call requests reliably reached the other user when both parties were online, that call acceptance or rejection behaved correctly, and that the call interface and WebRTC connection were properly cleaned up when the session ended.

From an end-user perspective, the basic workflow is as follows:
The user begins by registering an account and setting a password. On the login page, they enter their email and password, then complete multi-factor authentication by submitting the six-digit verification code sent to their email. After entering the main interface, users can search for and add contacts, create private or group conversations, send text messages and emojis, and upload files that appear in the shared message list for others to download.

When more direct communication is needed, the user may initiate a one-to-one video call within a private chat. For users or administrators who want to observe overall system activity, the dashboard provides live statistics, including the number of users, conversations, messages, files, and active sessions.

6 Improvements and Future Work

(1) In terms of file management, the current local-disk storage approach could be migrated to a distributed cloud storage system. This would provide better scalability and fault tolerance, especially in larger deployment scenarios.

(2) From a security perspective, beyond the existing login validation and transport-layer protection, the system could explore end-to-end encryption to offer a higher level of confidentiality for both text messages and media content.

(3) Regarding performance and deployment, parts of the system could be separated into independent services, enabling more fine-grained scaling. Combined with load balancing, this approach would help support higher levels of concurrency.

(4) For the frontend interface and user experience, further improvements can be made to refine layout and interaction details, ensuring a smoother and more consistent experience across different devices and screen resolutions.

7 Conclusion

This project set out to build a fully functioning real-time communication platform, covering a complete chain of features from user registration and login to contact management, text and file exchanges, one-to-one video calls, and basic system activity monitoring. Across messaging, file sharing, and video communication, the platform maintains a consistent user interface and interaction flow, allowing users to carry out a wide range of everyday communication tasks within a single environment. Overall, the system demonstrated stable performance and was able to support simultaneous online interactions and lightweight collaboration among multiple users.

8 Reflection

Throughout the development process, we gained a clearer understanding of how network communication, authentication, and data management interact when integrated into the same system. Many concepts that are typically presented separately in textbooks—such as how authentication ties into conversation access, how real-time connections synchronize with existing user and session data, and how persistent storage supports analytics and visualization—came together in a practical and coherent way. By combining these elements into a unified application, we developed a more concrete sense of how different networking and system-design ideas can be organized around a specific use case. This experience also lays a foundation for extending the platform to larger-scale deployments or exploring more advanced architectural designs in the future。

References

[1] Mahmoud H, Abozariba R. A systematic review on WebRTC for potential applications and challenges beyond audio video streaming[J]. Multimedia Tools and Applications, 2025, 84: 2909–2946. DOI:10.1007/s11042-024-20448-9.

[2] Tran-Truong P T, Pham M Q, Son H X, et al. A systematic review of multi-factor authentication in digital payment systems: NIST standards alignment and industry implementation analysis[J]. Journal of Systems Architecture, 2025, 162: 103402. DOI:10.1016/j.sysarc.2025.103402.

[3] WebRTC Project, “WebRTC Standards Documentation,” https://webrtc.org/

[4] Socket.IO Contributors, “Socket.IO Documentation,” https://socket.io/

[5] ReactJS Foundation, “React Official Documentation,” https://react.dev/

[6] Express.js Contributors, “Express Framework Documentation,” https://expressjs.com/

[7] R. Fielding, “Architectural Styles and the Design of Network-Based Software Architectures,” Ph.D. dissertation, UC Irvine, 2000.