MACAU UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Computer Science and Engineering

Faculty of Innovation Engineering

<<Software Project for Course Software Engineering>>

Homework ID	: Task2-User Requirements Specifications
Report Title	: Link-Space-Chat by Do It Dui Team URS
Student Name	: 游翔宇，孙宇轩，邢天舒
Student No.	: 1230006152，1230019445，1230002381
Date	: October 27, 2025

Amendment History

Version	Date	Modified By	Reviewed By	Amendment Details
1.0	2025/10/23	You Xiangyu	You Xiangyu	Initial version
1.01	2025/10/25	Sun Yuxuan	You Xiangyu	Optimize translation
1.02	2025/10/26	You Xiangyu	You Xiangyu	Supplemented non-functional requirements and interface details.
1.03	2025/10/27	Xing Tianshu	You Xiangyu	layout & formatting, and correction of sentence-level and formatting errors.

Table of Contents

1. Introduction

1.1. Purpose

This document aims to define in detail the user requirements for the "Cross-Network Instant Messaging Platform" (hereinafter referred to as "this system"). It will serve as the primary basis for subsequent system design, development, testing, and acceptance, ensuring that the final project deliverables meet the user's core expectations and business objectives. This document is intended for the project development team, testing team, and project managers.

1.2. Notation

This document follows established principles for requirements engineering, aiming for clarity, precision, and testability. System interaction and structure diagrams (such as context diagrams) follow the UML 2.0 specification.

1.3. Scope

This system is a lightweight, self-deployable instant messaging solution focused on meeting the needs of temporary, privacy-sensitive, and cross-network (LAN/public) communication. Its core feature is a frictionless chat experience, allowing users to quickly create and join chats without registering or installing a client.

The core scope of the local iteration includes:

1. Host: Enables the chat room service to be launched with a simple command on a personal device.

2. Participant: Enables anonymous chat access via a browser link.

3. Network Adaptability: The system can generate links for both local area networks (LANs) and wide area networks (WANs).

4. Core Chat Functionality: Supports the sending of text, emojis, images, and files.

5. Temporariness and Privacy: Chat data is temporary and destroyed upon service shutdown. No personally identifiable information is collected.

1.4. Context Diagram

This system (Link-Space-Chat service) serves as the core and interacts with the following external entities:

1. Host User: Starts and stops the service and obtains access links through the command-line interface (CLI).

2. Participant User: Performs all chat operations in a browser through the web user interface (UI).

3. Ngrok Service: An external third-party service. This system uses its local API to establish a public network access tunnel, enabling intranet penetration.

4. User Browser: The software environment that runs the client code, responsible for rendering the UI and handling WebSocket communication.

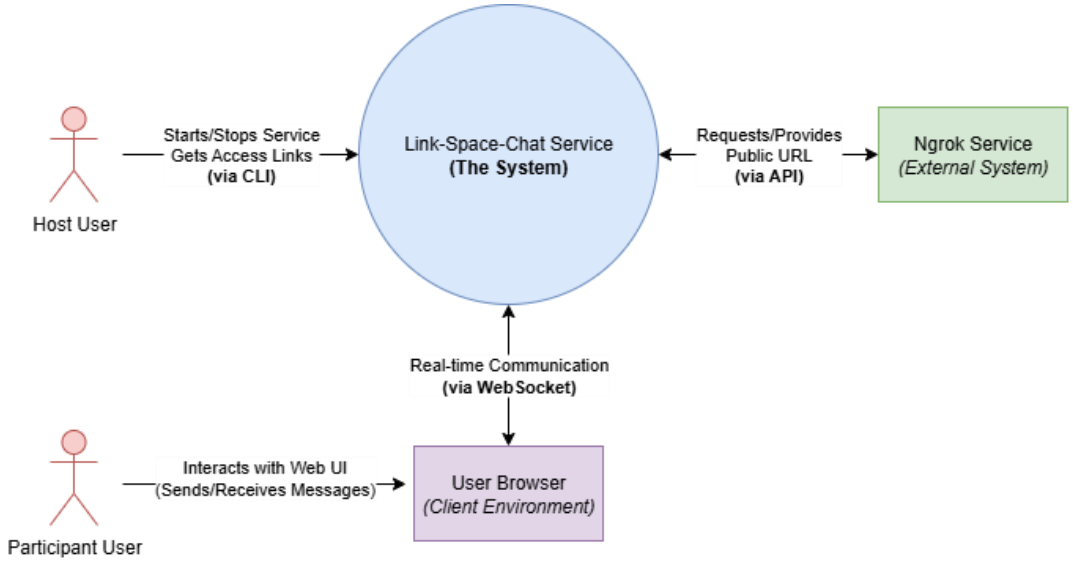

Figure 1 context diagram

1.5.Definitions and Acronyms

This section describes any terms used:

1. URS: User Requirements Specification
2. CLI: Command Line Interface
3. UI: User Interface
4. LAN: Local Area Network
5. PII: Personally Identifiable Information
6. WSS: WebSocket Secure
7. Host User: The user who creates and runs the chat room service.
8. Participant User: An ordinary user who joins the chat room via a link.

1.6.References

1. Project Proposal: "Cross-Network Instant Messaging Platform (Proposal)"
2. User Needs Survey and Results
3. Current System Code: https://github.com/YouXiangyu/Link-Space-Chat
4. Source Project: https://github.com/socketio/chat-example?tab=readme-ov-file
5. Technical Guidance: https://socket.io/docs/v4/tutorial/introduction
4. URS Standard Template

1.7.Overview

This document begins with a general description of the system's functionality and users in Section 2. Sections 3 and 4 define the project's constraints, assumptions, and dependencies. Section 5 outlines all core functional requirements in a table format. Section 6 provides a detailed breakdown of each functional requirement in Section 5. Section 7 defines the system's external interface requirements. Finally, Section 8 details non-functional requirements such as performance, security, and availability.

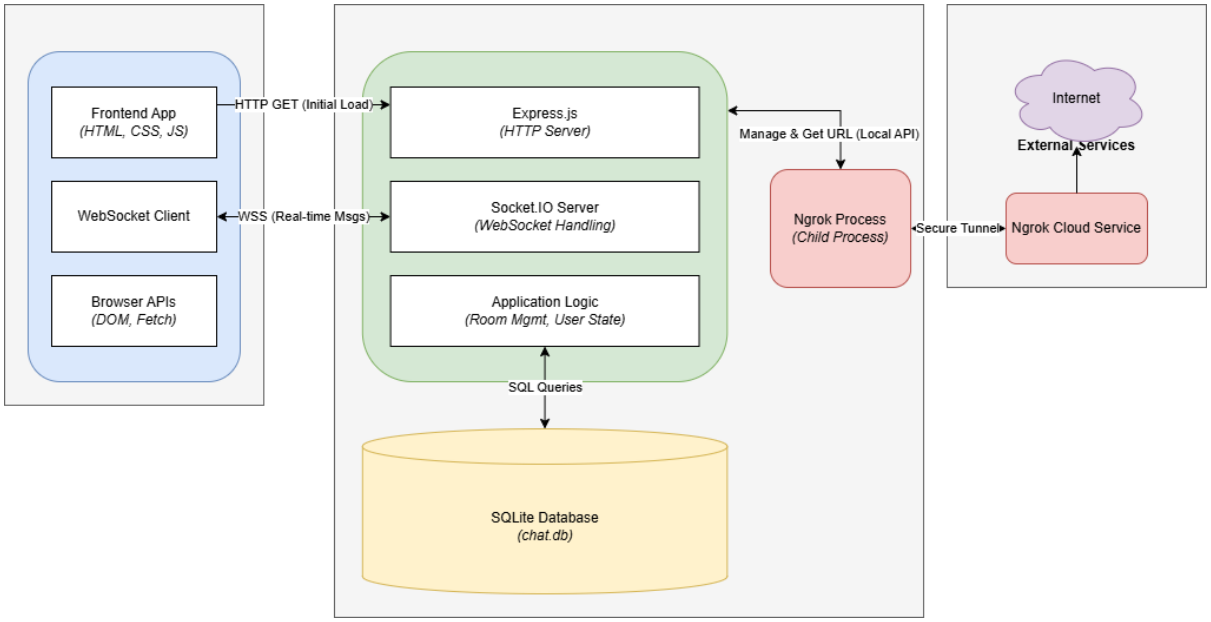

Figure 2 Key Technologies

2. General Description

2.1. System Functions

This system aims to address the shortcomings of traditional IM software in handling temporary communication scenarios. User surveys clearly indicate that users are troubled by the cumbersome "add friends first, then communicate" process, the reluctance to expose private social media accounts, and the difficulty of deleting temporary group chats. This system addresses these pain points through the following core features:

1. One-click room creation: Host users can instantly launch a temporary chat room on their local device.

2. Cross-network accessibility: The system automatically generates a LAN link for offline or internet-free scenarios, and also uses NAT traversal technology (Ngrok) to generate a public internet link for easy remote user access.

3. Browser-only access: Participants can join conversations using only a web browser, without the need to register, log in, or install any client software, significantly lowering the barrier to entry.

4. Temporary and secure communication: The system focuses on "ephemeral" conversations, a direction strongly supported by our User Survey results.

2.2. User Characteristics

According to user surveys, the target users are mainly students (especially computer/IT-related majors) and young professionals who have graduated. They are highly receptive to new technologies and value efficiency and personal privacy.

1. Host users (room creators):

(1) Characteristics: Usually individuals with a certain technical background, such as IT students, developers, organizers of technical conferences or offline events. They pursue efficient, controllable and highly private communication methods.

(2) Goal: Quickly establish a private or semi-public temporary communication channel with minimal setup cost, and easily share access methods with others.

2. Participating users (room members):

(1) Characteristics: Anyone who receives a chat link, who may be a friend, colleague, or stranger in a specific scenario (such as a classroom or technology sharing session).

(2) Goal: Be able to join the conversation instantly and seamlessly without worrying about personal information leakage or being forced to install new applications.

3. General Constraints

3.1. Software Constraints

1. Host: A runtime environment that supports Node.js (v16 or higher) is required.

2. Participant: A mainstream web browser that supports WebSocket and modern JavaScript (ES6+) is required, such as Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, etc.

3. External Dependencies: The system's public network access functionality relies on the ngrok npm module and the services it provides.

3.2. Hardware Constraints

1. Host: Any PC or server capable of running a Node.js server. No special hardware is required.

2. Participant: Any device capable of running a modern web browser, including desktops, laptops, smartphones, or tablets.

4. Assumptions and Dependencies

1. Assumptions:

(1) The host user has basic capabilities to launch an application from the command line.

(2) The participating users' network environments allow for establishing WebSocket connections with the host server.

2. Dependencies:

(1) The system's public network accessibility is entirely dependent on the stability and availability of the third-party Ngrok service.

(2) The availability of the chat room directly depends on whether the host user's device is online and whether the service process is running.

5. Functional Requirements Master List

Req. ID	Requirement Name	Requirement Description
REQ-1.1	Create a room with one click	A host user must be able to start a new chat room by executing a single script or command.
REQ-1.2	Generate access link	After the room is created, the system should automatically generate and display LAN and public network access links.
REQ-1.3	Automatic data destruction	When the last user leaves the chat room, the system should automatically delete all chat records in the room.
REQ-2.1	Text message	Users must be able to send and receive text messages in real time.
REQ-2.2	Emoji support	The system should support the input and correct rendering of standard Emoji expressions.
REQ-2.3	Photo Sharing	Users must be able to upload and send image files, and images should be displayed inline in the chat window.
REQ-2.4	File Sharing	Users must be able to upload and send generic files, for which download links will be provided.
REQ-2.5	Markdown support	The system should support basic Markdown syntax to format text messages.
REQ-3.1	Set a temporary nickname	When a user enters a chat room for the first time, he/she must set a temporary nickname and the nickname must be unique within that room.
REQ-3.2	Online user list	The user interface should display a list of nicknames of all online users in the current chat room in real time.
REQ-3.3	Loading historical messages	When a new user joins, the system should automatically load and display the most recent 20 historical messages.

6. Functional Requirement REQ-1.1: One-Click Room Creation

6.1. Description

The hosting user must be able to instantly start a new chat room service by executing a simple command on the local computer.

6.2. System Input

The user executes the startup script (e.g. node server.js) in the operating system's command line/terminal, optionally appending the port number and parameters to start Ngrok.

6.3. Display

After a successful startup, the system should clearly display the following information on the host user's command line/terminal screen:

1. The local service running address.
2. The LAN access address.
3. (If enabled) Ngrok's public network access address and a room link example.

6.4. System Processing

Include: Upon system startup, the following business rules will be executed:

1. Start an Express web server and listen on the specified port.
2. Call the os module to automatically detect and determine the local network IP address.
3. Check the startup parameters. If public network access is required, start the Ngrok child process using child_process.spawn.
4. Access Ngrok's local management API to obtain the dynamically generated public URL.

6.5. System Output

1. All available access links printed to the command-line interface.
2. A running Node.js server process in a listening state, ready to accept HTTP and WebSocket connections.

6.6. Error Handling

1. When a user terminates the service using Ctrl+C, the system will capture the SIGINT signal and prompt the user to clear all chat history databases, achieving a graceful shutdown.
2. If the Ngrok service cannot be connected, the system will print a warning message, but local and LAN services will continue to run.

7. Functional Requirement REQ-1.2: Generate Access Links

7.1. Description

After the service is started, the system should automatically generate and display the LAN IP address link and (if configured) the public URL link for the host user to share.

7.2. System Input

This functionality is triggered automatically as part of REQ-1.1, without separate direct user input.

7.3. Display

Displays formatted LAN and public URLs in the host user's command line interface, including example room link formats (e.g., Example Room Link: http://<URL>/r/your-room-id).

7.4. System Processing

1. LAN connection: Use the getLanAddress function to iterate over all network interfaces, prioritizing IPv4 addresses that fall within the private IP address range.

2. Public network connection: Obtain a dynamically generated HTTPS public network URL by interacting with the Ngrok child process's API.

7.5. System Output

Prints two or three URL strings to the terminal that can be copied and shared.

8. Functional Requirement REQ-1.3: Automatic Data Deletion

8.1. Description

When the last user in a chat room leaves, the system should automatically delete all related chat records in the room.

8.2. System Input

The user's disconnect or leave_room Socket.IO event.

8.3. Display

There is no direct display for participating users. A confirmation log message is printed to the server's command line interface (for example, History for empty room <roomId> cleared.).

8.4. System Processing

In the removeUserFromRoom function, when a user disconnects, the system checks the number of users in the corresponding room. If the number of users reaches 0, the system calls the db.clearHistoryForRoom(roomId) function, which executes a transaction and atomically deletes all data related to the roomId from the messages and rooms tables.

8.5. System Output

The data of the corresponding room in the database is physically deleted.

8.6. Data Handling

Use SQL transactions to ensure the integrity of the delete operations, either all succeed or all rollback in case of errors.

8.7. Error Handling

If the database deletion fails, the transaction will be rolled back and the data will not be partially deleted. The error will be captured and logged to the server console.

9. Functional Requirement REQ-2.1: Text Messaging

9.1. Description

Users must be able to send and receive text messages in real time.

9.2. System Input

The client sends a chat_message event via WebSocket with the text content as a parameter.

9.3. Display

The sent message should appear in the message display area of all users in the room in real time, with the sender's nickname and timestamp.

9.4. System Processing

The server-side socket.on("chat_message", ...) listener receives the message and calls db.saveMessage to save the message to the SQLite database. If successful, the complete message object is broadcast to all users in the room.

9.5. System Output

All room member clients will receive the chat_message event with the complete message object for rendering in the UI.

9.6. Data Handling

Text message text is processed as a standard UTF-8 string.

9.7. Error Handling

If the message fails to be saved to the database successfully, the server will not broadcast the message and return an error message to the sender through the callback function ack.

10. Functional Requirement REQ-2.2: Emoji Support

10.1. Description

The system should support the input and correct rendering of standard Emoji expressions.

10.2. System Input

The user enters text containing Emoji characters in the text input box.

10.3. Display

Emoticons should now display correctly graphically in the chat interface on all clients.

10.4. System Processing

The system processes Emoji expressions as standard UTF-8 character text. Both the database and the transport layer support UTF-8 encoding, and no special logic is required.

10.5. System Output

Text messages containing Emoji are broadcast correctly.

10.6. Constraints

Correct display depends on the participating users' operating systems and browsers having the appropriate font and rendering capabilities.

11. Functional Requirement REQ-2.3 & 2.4: Image and File Sharing

11.1. Description

Users should be able to upload and share images and general files.

11.2. System Input

The user selects a local file through the UI.

11.3. Display

Image files should appear inline in the chat stream. Other files should appear as cards with the file name and a download link.

11.4. System Processing

The client sends the file data to the server via WebSocket. After receiving the file, the server does not store it persistently, but directly broadcasts the file data to all other users in the room.

11.5. System Output

After receiving the file data, other clients will render it or provide download according to the file type.

11.6. Constraints

The size of a single file should be limited to 5MB. Supported image formats are JPG, PNG, and GIF.

11.7. Data Handling

File data should not be stored persistently on disk.

11.8. Error Handling

If the user selects an unsupported file type or the file is too large, the front-end should give a clear prompt and prevent the upload.

12. Functional Requirement REQ-2.5: Markdown Support

12.1. Description

The system should support basic Markdown syntax to format text messages (such as bold, italics, code blocks).

12.2. System Input

The user enters text that conforms to Markdown syntax in the chat input box.

12.3. Display

The rendered text should be displayed in the chat window as formatted text, not raw Markdown code.

12.4. System Processing

It is recommended to use libraries such as marked.js for rendering on the client side, and use DOMPurify for XSS security purification before rendering to prevent malicious script injection.

12.5. System Output

There is no server-side output, processing is done on the client side.

13. Functional Requirement REQ-3.1: Temporary Nickname

13.1. Description

When a user enters a chat room for the first time, he/she must set a temporary nickname that is unique in the room.

13.2. System Input

The client sends a join_room event with a { roomId, nickname } object.

13.3. Display

Upon first entering the room, a modal dialog will overlay the main chat interface, requiring the user to enter a nickname.

13.4. System Processing

The server checks the nickname for uniqueness. If there is a conflict, it uses the server-ping event to determine if the old connection is a "zombie connection." If so, it disconnects the old connection and allows the new user to use the nickname. If not, it rejects the new user's request.

13.5. System Output

The callback function ack is used to notify the client of the success or failure of joining and the specific reason.

13.6. Constraints

The nickname cannot be empty and cannot exceed 20 characters.

13.7. Error Handling

If the nickname entered by the user is already taken by an active user, the system will return a clear error message "The nickname is already taken".

14. Functional Requirement REQ-3.2: Online User List

14.1. Description

The user interface should display a list of nicknames of all online users in the current chat room in real time.

14.2. System Input

This function is automatically triggered by the system when a user successfully joins or leaves a room.

14.3. Display

A list of nicknames of all current users is updated and displayed in real time in a dedicated area of the web interface (such as a sidebar).

14.4. System Processing

Whenever a user joins or leaves the room, the system calls the emitRoomUsers function, which gets the latest complete user list from memory and broadcasts it to all members in the room through io.to(roomId).emit("room_users", users).

14.5. System Output

All clients receive the room_users event with a string array containing the nicknames of all users.

15. Functional Requirement REQ-3.3: Load Message History

15.1. Description

When a new user joins, the system should automatically load and display the most recent 20 historical messages so that the new member understands the context.

15.2. System Input

This function is automatically triggered by the system after the user successfully handles the join_room event.

15.3. Display

When a new user joins, the message display area of their chat window will automatically be filled with recent chat history.

15.4. System Processing

After a user successfully joins a room, the system calls db.getRecentMessages(roomId, 20) to query up to 20 historical messages from the database, and then sends them only to the new user through socket.emit("history", history).

15.5. System Output

The newly joined client receives a history event, whose data is an array containing up to 20 historical message objects.

15.6. Constraints

The number of historical messages loaded is hard-coded to 20.

16. External Interface Requirement

16.1. Data Interfaces

The system interacts with several external components through well-defined data interfaces:

1. WebSocket Interface: The primary communication channel between the client (user's browser) and the host server. All real-time data, including text messages, user join/leave events, and user lists, are transmitted as structured JSON objects over this interface.

2. Ngrok API: The system acts as an HTTP client to the local Ngrok management API (http://127.0.0.1:4040/api/tunnels). It sends an HTTP GET request and receives a JSON response containing the public tunnel URL. This is a read-only interface.

3. SQLite Database Interface: The system interfaces with a local SQLite database file (chat.db) via the sqlite3 Node.js library. It sends standard SQL commands (INSERT, SELECT, DELETE) to persist and retrieve message history and manage room records.

16.2. User Interfaces

The system provides two distinct user interfaces for its different user roles:

1. Web Graphical User Interface (GUI): This is the primary interface for Participant Users, rendered in a standard web browser. It is a single-page application that must be responsive and intuitive. Key components include:

(1) A modal dialog for initial nickname entry.

(2) A main display area for rendering the chronological flow of chat messages.

(3) A text input field with a send button for composing messages.

(4) A side panel that displays a real-time list of all currently online users.

2. Command Line Interface (CLI): This is the exclusive interface for the Host User. It is used to start the server, view real-time status logs (including server startup, user connections, and errors), and obtain the generated LAN and Public URL for sharing.

16.3. Other Interfaces

1. Node.js Runtime Environment: The application interfaces with the Node.js software environment to access underlying system resources. This includes using the os module to query network interfaces and the child_process module to spawn and manage the Ngrok process.

2. Web Browser APIs: The client-side application relies heavily on standard Web Browser APIs, including the WebSocket API for real-time communication, the DOM API for rendering the user interface, and the Fetch API or XMLHttpRequest for retrieving initial data.

17. Non-Functional Requirements

17.1. System Performance

This section describes the system performance requirements. Example response times are suggested though different systems requirements may apply.

Req. Id	Description	Response Time
PERF-1.1	Chat Room Entry Time: Time experienced by a participant from clicking a valid room link to seeing the chat interface and being prompted for a nickname.	3 seconds
PERF-1.2	Message Latency: Time from a user sending a message to it appearing on the screens of all other participants in the same room under normal network conditions.	1 second
PERF-1.3	Usability (Time to First Message): A new user, without any prior instructions, must be able to understand the interface and successfully send their first message.	60 seconds

17.2. Information Security

This section describes the information security requirements.

Req. Id	Description
SEC-1.1	No Personal Data Collection: The system shall not require, collect, or store any Personally Identifiable Information (PII) such as real names, email addresses, or phone numbers from any user. User identity is limited to a temporary, self-assigned nickname.
SEC-1.2	Encrypted Transmission: All communication between the user's browser and the host server must be transmitted over an encrypted network channel (HTTPS/WSS) to prevent eavesdropping.
SEC-1.3	Ephemeral Data Storage: All chat data (messages) shall only be held in the host's database for the duration of an active chat session. Data must be irrevocably deleted from the database once the last participant leaves the room (as defined in REQ-1.3).

17.3. Availability

This section describes the availability requirements in terms of days and permissible planned and unplanned unavailability.

Req. Id	Description
AVA-1.1	Host-Dependent Availability: The availability of any given chat room is directly and entirely dependent on the Host User's machine being powered on, connected to the network, and running the server process. The system itself is not a centrally-hosted, high-availability service.
AVA-1.2	Process Stability: The server application must be designed to run continuously without crashing due to common errors (e.g., invalid user input, sudden disconnections). It should handle exceptions gracefully and log relevant error information to the CLI.

17.4. Capacity

Req. Id	Description
CAP-2.1	Concurrent User Volume: A single chat room instance, running on a standard consumer-grade laptop with a broadband internet connection, must be able to manage and provide stable real-time performance for at least 10 concurrent users.(Note: Long-term data volume requirements (CAP-1.1, CAP-1.2, etc.) are not applicable to this system due to its ephemeral, session-based nature where data is not intended for long-term storage.)