Software Requirements Specification

for

Chat Web Application

Prepared by

Moaz Adly
Omar Hussein
Rahma Ali
Salma Hazem

Digital Egypt Pioneers Initiative - DEPI

21-10-2024

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to specify the software requirements for the Real-Time Chat Application. This document covers the detailed functionality, features, and design specifications for the development of a fully functional chat application. The current version is the initial release (v1.4) and outlines the entire system, including both the frontend and backend components.

This Software Requirements Specification (SRS) covers the following scope:
Frontend: Implemented using React to provide users with a responsive and interactive interface for real-time messaging, managing chat rooms, direct messaging, and notifications.

Backend: Developed using Node.js and Socket.IO to enable real-time communication and message delivery, along with user authentication and room management.

System Features: Includes real-time messaging, typing indicators, user authentication, chat room management (both public and private), direct messaging, add friends, and a notification system.

This SRS outlines the entire system from user interface design to back-end communication and does not describe a subsystem or partial implementation. The goal of this project is to deliver a fully operational chat platform that meets the requirements for real-time communication with secure and scalable functionalities.

1.2 Document Conventions

This Software Requirements Specification (SRS) follows a standardized format to ensure clarity and consistency throughout the document. The following conventions and typographical standards are used:

Font:
The main body of the document is written in Times New Roman, 12pt for readability.
Headings and subheadings are in bold to indicate section hierarchy.

Highlighting:
Bold text is used to emphasize important terms or titles (e.g., "Real-Time Chat Application").
Italics are used for special emphasis or for technical terms when first introduced (e.g., Socket.IO).

Requirement Prioritization:
Each requirement is either Functional (F) or Non-Functional (NF). This designation is listed before each requirement to clarify the type.

Priorities are indicated as:
High (H) – Critical for system functionality and must be included in the final version.
Medium (M) – Important, but not critical; should be included if time permits.
Low (L) – Nice-to-have features that will be implemented if resources allow.

Numbering:

Sections and subsections are numbered for easy referencing (e.g., 1.1, 2.1, 3.1.1).
Requirements are labeled sequentially (e.g., R1, R2, R3).

Terminology:
When referring to specific technologies or components, their full names are used the first time, followed by acronyms in subsequent references (e.g., WebSocket, then WS).
This document ensures that all requirements are clearly stated and easily traceable, with each requirement having its own designated priority, rather than inheriting priorities from higher-level statements.

1.3 Intended Audience and Reading Suggestions

1.3.1 This Software Requirements Specification (SRS) is intended for the following audiences:

- **Developers:** Responsible for implementing the system according to the requirements outlined. Developers should focus on the **functional** and **non-functional requirements**, **system architecture**, and **API documentation.**

- **Project Managers:** Oversee the progress and ensure that the project meets its deadlines and objectives. Project managers should focus on the **project scope**, **milestones**, and **deliverables**.

- **Testers/Quality Assurance:** Responsible for verifying that the application meets the specified requirements. Testers should focus on the **functional requirements**, **non-functional requirements**, and the **testing plan** sections to design test cases.

- **Users:** End-users who will interact with the application. Users may refer to the **overview** and **system features** to understand the functionalities of the application.

- **Documentation Writers:** Responsible for creating user manuals or help documents. They should review the entire document, particularly the **UI/UX design**, **system features**, and **API documentation** sections.

1.3.2 Document Organization and Reading Suggestions

Section 1: Introduction
Overview of the project, its purpose, and audience. All readers should begin with this section to get a high-level understanding of the system and its context.

Section 2: System Overview
Describes the overall system architecture and key components. Developers and project managers should start here to understand how the system will be built.

Section 3: Functional Requirements
Detailed breakdown of the system's functional behavior. Developers and testers should read this section in detail, as it contains the core requirements for system functionality.

Section 4: Non-Functional Requirements
Covers performance, scalability, and other non-functional aspects. Project managers and developers should focus on this section to ensure the system meets performance expectations.

Section 5: System Design and Architecture
Provides detailed design decisions, including technical architecture, backend, and frontend frameworks. This section is critical for developers and project managers to understand how to implement the system.

Section 6: Testing Plan
Outlines the approach to testing the system's functionality. Testers and developers should review this section to prepare for the testing phase.

Section 7: Deployment and Maintenance
Covers deployment strategies, cloud hosting, and post-launch support. Developers, project managers, and documentation writers should consult this section for information on application deployment and ongoing maintenance.

1.3.3 Suggested Reading Sequence:

Introduction – All readers
System Overview – Developers, project managers
Functional Requirements – Developers, testers
Non-Functional Requirements – Project managers, developers
System Design and Architecture – Developers
Testing Plan – Testers, developers
Deployment and Maintenance – Project managers, developers, documentation writers

Each reader type is encouraged to prioritize the sections that are most pertinent to their role but should skim through other sections for a comprehensive understanding of the project.

1.4 Project Scope

The **Real-Time Chat Application** is a software system designed to provide users with a seamless and interactive communication platform that supports public and private chat rooms, direct messaging, and real-time notifications. The purpose of this application is to facilitate instantaneous communication between users, offering both personal and group interactions in a secure and user-friendly environment.

1.4.1 Key Benefits:

Instantaneous Communication: Users can send and receive messages in real-time using ***WebSockets***, ensuring minimal latency.

User Authentication and Security: Secure access to chat rooms and private messages through authentication systems (***JWT***).

Customizable Chat Rooms: Support for creating and managing both public and private chat rooms, allowing users to communicate based on their preferences.

Enhanced User Experience: Responsive design, mobile-friendly layout, typing indicators, and notifications to keep users informed of new messages or friend requests and other activities.

Scalable Architecture: A robust backend built with ***Node.js*** and ***Socket.IO***, capable of handling multiple users and chat rooms simultaneously.

1.4.2 Objectives and Goals:

Efficient Real-Time Messaging: Ensure smooth real-time communication for individual users and groups, with efficient handling of ***WS*** connections.

Scalability: Support large numbers of concurrent users without sacrificing performance.

User Engagement: Provide features like typing indicators and notifications to keep users engaged and informed.

Cross-Platform Compatibility: A responsive design that functions seamlessly across desktop and mobile devices.

Security: Implement strong authentication mechanisms to protect user data and secure chat rooms.

1.4.3 Alignment with Business Strategy:

The application is part of a broader strategy to develop tools that enhance communication and collaboration in digital spaces. By offering a scalable and secure chat platform, this project aligns with corporate goals of increasing user engagement, enhancing productivity, and providing innovative communication solutions in a rapidly evolving digital landscape.

This SRS describes the complete scope of the system for its initial release (v1.4), detailing both frontend and backend features. It outlines the core functionalities such as real-time messaging, user authentication, chatroom management, and notifications. Future updates and enhancements will build upon this foundation to add more advanced features like media sharing, encryption, and additional user management capabilities.

1.5 References

This Software Requirements Specification (SRS) refers to the following documents and resources, which provide additional context, standards, or technical specifications necessary for understanding and implementing the Real-Time Chat Application. Each reference is included with sufficient details to allow easy access:

1- React Documentation
Title: React – A JavaScript library for building user interfaces
Author: Facebook, Inc.
Version: Latest
Date: Accessed October 2024
URL: https://reactjs.org/docs/getting-started.html
Description: Official documentation for using React to build the frontend of the application.

2- Node.js Documentation
Title: Node.js JavaScript Runtime
Author: OpenJS Foundation
Version: Latest
Date: Accessed October 2024
URL: https://nodejs.org/en/docs/
Description: Official documentation for setting up and using Node.js for the backend of the application.

3- Socket.IO Documentation
Title: Socket.IO Real-time Engine
Author: Socket.IO contributors
Version: Latest
Date: Accessed October 2024
URL: https://socket.io/docs/
Description: Documentation for implementing real-time, bidirectional, event-based communication using ***WebSockets***.

4- JWT (JSON Web Tokens) Documentation
Title: JSON Web Tokens (JWT) for Secure Authentication
Author: Auth0
Version: Latest
Date: Accessed October 2024
URL: https://jwt.io/introduction/
Description: Documentation on using JSON Web Tokens for secure user authentication.

5- W3C HTML5 Specification
Title: HTML5 – A Vocabulary and Associated APIs for HTML and XHTML
Author: World Wide Web Consortium (W3C)
Version: Latest
Date: Accessed October 2024
URL: https://www.w3.org/TR/html5/
Description: Official specification for HTML5 used in structuring the frontend of the chat application.

6- CSS3 Specification
Title: Cascading Style Sheets Level 3 (CSS3)
Author: World Wide Web Consortium (W3C)
Version: Latest
Date: Accessed October 2024
URL: https://www.w3.org/Style/CSS/Overview.en.html
Description: The specification of CSS3 used for styling and layout in the chat application's frontend.

7- Deployment Guide for Heroku
Title: Deploying Node.js Apps on Heroku
Author: Heroku
Version: Latest
Date: Accessed October 2024
URL: https://devcenter.heroku.com/articles/deploying-nodejs
Description: Guide on how to deploy the Node.js backend for the chat application to Heroku.

8- JWT (JSON Web Tokens) Documentation
Title: JSON Web Tokens (JWT) for Secure Authentication
Author: Auth0
Version: Latest
Date: Accessed October 2024
URL: https://jwt.io/introduction/
Description: Documentation on using JSON Web Tokens for secure user authentication and authorization.

2. Overall Description

2.1 Product Perspective

The Real-Time Chat Application is a new, self-contained product developed to facilitate real-time communication among users via public or private chat rooms, as well as through direct messaging (DMs). This application is not a follow-on member of an existing product family, but rather a standalone solution designed to offer seamless, interactive messaging with real-time updates, user authentication, and notification systems.

This product integrates various components and technologies to deliver an efficient user experience, such as React for building the frontend user interfaces, ***Node.js*** with ***WebSockets*** (specifically ***Socket.IO***) for real-time message transmission, and ***JWT (JSON Web Tokens)*** for user authentication.

In the broader context, the Real-Time Chat Application could be integrated with other systems or platforms, such as a social media network or an enterprise communication tool. The main components of this chat application are independent but can communicate with external APIs or services for additional functionalities such as notifications or advanced user management.

System Components and Interfaces:

Frontend (Client-side): Built using ***React***, the frontend handles all user interactions, chat interfaces, and displays notifications in real time.

Backend (Server-side): The backend is built using ***Node.js*** and ***Socket.IO*** to manage chat room creation, user authentication, and real-time messaging between users.

WebSocket Interface: This facilitates the bidirectional communication between the client and server for instant message delivery and reception.

Database: Used for storing user data, message history, chat room details, and authentication tokens.

Authentication System: JWT-based system for secure user login and access control to different chat rooms.

2.2 Product Features

The Real-Time Chat Application offers a comprehensive set of features to provide users with seamless and interactive communication experiences. Below is a high-level summary of the major functionalities of the product:

Real-Time Messaging:
Users can send and receive messages instantly in real-time, ensuring that conversations are dynamic and up-to-date. The use of ***WebSockets*** facilitates this instant message transmission.

Public and Private Chat Rooms:
Users can join or create both public and private chat rooms. Public chat rooms are accessible to all users, while private chat rooms require invitations or authentication to join.

Direct Messaging (DMs):
The application supports direct messaging, allowing users to have private, one-on-one conversations outside of group chat rooms.

User Authentication:
Secure user login and access control are managed via JSON Web Tokens (JWT). This ensures that only authorized users can access chat rooms and send messages.

Notifications:
Users receive notifications for new messages, friend request, or important updates in their chat rooms, helping them stay informed without constantly monitoring the chat window.

Typing Indicators:
The application provides real-time typing indicators, allowing users to see when others are actively typing a message in the same chat room.

Message History:
The application stores message history, allowing users to review past conversations when they re-enter a chat room.

Mobile Responsiveness:
The chat interface is optimized for different screen sizes, providing an engaging experience on both desktop and mobile devices.

Error Handling and Reconnection:
The system includes error handling for WebSocket disconnections, with automatic reconnection to ensure a seamless user experience during network issues.

These features collectively provide a robust and engaging communication platform for users, with a focus on real-time interaction, security, and accessibility. Detailed explanations of each feature will be provided in Section 3, including their underlying architecture and how they interact with other components.

2.3 User Classes and Characteristics

The Real-Time Chat Application is designed to accommodate a diverse range of users, each with distinct characteristics and needs. Below are the identified user classes, along with their pertinent characteristics:

2.31 General Users
Frequency of Use: Regular users who engage frequently in conversations.
Functions Used: All primary features, including public and private chat rooms, direct messaging, and notifications.
Technical Expertise: Moderate; users should have basic knowledge of web applications but do not need advanced technical skills.
Characteristics: General users are typically individuals looking for a straightforward way to communicate with friends, family, or colleagues. They value real-time messaging and ease of use.

2.3.2 Occasional Users
Frequency of Use: Infrequent; users who may use the application sporadically for specific conversations or events.
Functions Used: Basic features such as viewing messages and participating in group chats or direct messages.
Technical Expertise: Low to moderate; basic familiarity with messaging applications.
Characteristics: These users may not rely on the chat application for regular communication but appreciate the ability to join discussions when necessary. They may require a user-friendly interface.

2.3.3 Developers/Technical Users
Frequency of Use: Variable; primarily during development or maintenance phases.
Functions Used: Backend functionalities, API integrations, and testing features.
Technical Expertise: Very high; developers need comprehensive knowledge of the underlying technologies (React, Node.js, Socket.IO, etc.).
Characteristics: This class includes individuals involved in enhancing the application, troubleshooting issues, and implementing new features. They are less concerned with the user interface but prioritize functionality and performance.

2.3.4 Business/Corporate Users
Frequency of Use: Regular; users from organizations using the application for internal communication.
Functions Used: Features supporting team collaboration, such as group chats, direct messaging, and notifications.
Technical Expertise: Moderate; familiarity with workplace communication tools is common.
Characteristics: These users value security and efficiency. They might require specific features like advanced user authentication and an organized way to manage communication within their teams.

2.4 Operating Environment

The Real-Time Chat Application will operate in a multi-platform environment, ensuring accessibility and usability across various devices and operating systems. Below is a detailed description of the operating environment:

2.4.1 Hardware Platform

Client-Side Devices:

- Desktops and Laptops: Users can access the application via standard desktop and laptop computers equipped with modern hardware (e.g., Intel i3 or higher, 4GB RAM or more).
- Mobile Devices: Smartphones and tablets with screen sizes ranging from 5 inches to 12 inches, supporting modern web browsers.

Server-Side:

A cloud-based server infrastructure, such as Heroku, capable of handling multiple concurrent connections and real-time data processing. Recommended specifications include:
CPU: At least 2 vCPUs
RAM: Minimum of 4GB
Storage: SSD with a minimum of 20GB

2.4.2 Operating System

Client-Side:
The application will be accessible through major operating system (Windows 10 and later)

Mobile Operating Systems:
Android 8.0 (Oreo) and later.

2.4.3 Software Components

Frontend:
The chat application is developed using:
React (latest stable version)
JavaScript (ES6 and beyond)
HTML5 , CSS3 and bootstrap for markup and styling.

Backend:
The server-side component uses:
Node.js (version 14 or later)
Express for routing and middleware handling
Socket.IO for WebSocket communication.
MongoDB as the database for storing user data and message history.

Web Browsers:
The application will be compatible with modern web browsers, including:
Google Chrome (latest version)
Mozilla Firefox (latest version)
Microsoft Edge (latest version)

2.4.4 Other Software Components

Version Control:
The development team will use Git for version control, facilitating collaboration and code management.

Deployment Tools:
Continuous integration and deployment may be facilitated through tools like GitHub Actions.

2.4.5 Coexistence

The application must coexist peacefully with other applications that may be running on the client devices, such as antivirus software, firewalls, and browser extensions. It should maintain compatibility with common network configurations, ensuring smooth operation in various user environments.

This diverse operating environment ensures that the Real-Time Chat Application is accessible to a wide audience while maintaining robust performance and reliability.

2.5 Assumptions and Dependencies

The successful development and implementation of the Real-Time Chat Application are based on several assumptions and dependencies that could significantly impact the project if they prove to be incorrect or change during the development process.

2.5.1 Assumptions

User Connectivity: It is assumed that users will have reliable internet connectivity to ensure smooth real-time messaging and application functionality. Variability in user internet speed or availability could affect user experience.

Browser Compatibility: The application is assumed to be accessed using modern web browsers. If users attempt to use outdated or unsupported browsers, they may encounter functionality issues that could affect their experience.

Device Specifications: It is assumed that users will have devices that meet minimum hardware specifications (e.g., adequate RAM and CPU) to run the application effectively. Users with lower-spec devices may experience performance issues.

Security Compliance: It is assumed that users will adhere to best practices regarding password security and account management. If users do not follow security guidelines, it could increase the risk of unauthorized access to their accounts.

Third-Party Services: The application may depend on third-party services for features like notifications or authentication. It is assumed that these services will be reliable and maintain compatibility with the chat application throughout its lifecycle.

Development Environment Stability: It is assumed that the development environment (Node.js, Express, MongoDB, etc.) will remain stable without significant changes that could introduce breaking changes or compatibility issues during development.

2.5.2 Dependencies

Socket.IO Library: The project depends on the Socket.IO library for real-time messaging capabilities. Any changes or issues with this library could directly impact the core functionality of the chat application.

Database Integration: The application relies on MongoDB for data storage. Any issues with MongoDB (such as outages or changes in API) could affect data retrieval and storage processes.

Authentication Service: The use of JWT for user authentication creates a dependency on the implementation of secure token handling mechanisms. If the JWT library or its usage guidelines change, it may necessitate a redesign of the authentication process.

Cloud Hosting Provider: The application is expected to be hosted on a cloud platform (e.g., AWS, Heroku). Any service disruptions or changes in pricing models from the cloud provider could affect the deployment and operational costs of the application.

Team Skill Sets: The development team's familiarity with the chosen technologies (React, Node.js, MongoDB) is crucial for project success. If team members lack the necessary skills or if key personnel leave, it could impact project timelines and quality.

User Adoption and Feedback: The success of the application may depend on user feedback and adoption rates. If initial users provide critical feedback that necessitates significant changes, it could impact the project timeline and resource allocation.

These assumptions and dependencies must be monitored throughout the development process to mitigate risks and ensure the successful delivery of the Real-Time Chat Application. Regular communication with the development team and stakeholders will help identify any changes to these factors and allow for timely adjustments to project plans.

3. System Features

3.1 User Authentication

3.1.1 Description and Priority

The User Authentication feature allows users to securely sign up, log in, and manage their accounts within the Real-Time Chat Application. This feature is of High priority as it ensures secure access to the chat rooms and personal messages, protecting user data and privacy.

Benefit: 9
Penalty: 7
Cost: 5
Risk: 6

3.1.2 Stimulus/Response Sequences

User Signup:
Stimulus: The user clicks on the "Sign Up" button and enters their details (username, password, email).
Response: The system validates the input, creates a new user account, and displays a confirmation message.

User Login:
Stimulus: The user enters their credentials (username and password) and clicks "Log In".
Response: The system verifies the credentials. If valid, the user is directed to the chat interface; if invalid, an error message is displayed.

Password Recovery:
Stimulus: The user clicks on "Forgot Password" and provides their email address.
Response: The system sends a password recovery link to the user's email address.

3.1.3 Functional Requirements

REQ-1: The system shall allow users to create an account by providing a unique username, password, and email address.

REQ-2: The system shall validate user credentials during login, ensuring that the username and password match an existing account.

REQ-3: The system shall allow users to log out securely, invalidating their session and redirecting them to the login page.

REQ-4: The system shall implement password recovery by sending an email with a password reset link to the registered email address.

REQ-5: The system shall enforce password complexity requirements (e.g., minimum length, special characters) during account creation and password updates.

REQ-6: The system shall provide an error message for invalid login attempts, specifying whether the username or password was incorrect.

REQ-7: The system shall limit login attempts to prevent brute force attacks, locking the account for a specified duration after multiple failed attempts.

3.2 Real-Time Messaging

3.2.1 Description and Priority

The Real-Time Messaging feature enables users to send and receive messages instantly within chat rooms and direct messages. This feature is of High priority as it is the core functionality of the chat application.

Benefit: 9
Penalty: 8
Cost: 6
Risk: 5

3.2.2 Stimulus/Response Sequences

Send Message:
Stimulus: The user types a message in the input box and clicks "Send".
Response: The system sends the message to the designated chat room or user, and it appears in the chat interface for all participants.

Receive Message:
Stimulus: A message is sent to the user from another participant.
Response: The system immediately displays the incoming message in the chat interface.

Typing Indicator:
Stimulus: A user starts typing in the message input box.
Response: The system displays a "User is typing..." indicator in the chat interface.

3.2.3 Functional Requirements

REQ-8: The system shall enable users to send messages in real time to chat rooms and individual users.

REQ-9: The system shall display incoming messages in the chat interface immediately after they are received.

REQ-10: The system shall show typing indicators for users currently typing in a chat room.

REQ-11: The system shall allow users to delete their own messages, removing them from the chat history.

REQ-12: The system shall implement message history, allowing users to scroll back and view previous messages in a chat room.

REQ-13: The system shall handle message delivery failures gracefully, providing error messages to users when applicable.

3.3 Chat Room Management

3.3.1 Description and Priority

The Chat Room Management feature allows users to create, join, and manage both public and private chat rooms. This feature is of High priority as it facilitates group communication and enhances user interaction within the application.

Benefit: 9
Penalty: 7
Cost: 6
Risk: 5

3.3.2 Stimulus/Response Sequences

Create Chat Room:
Stimulus: The user clicks on "Create Room," enters a room name and selects visibility (public/private), and clicks "Create."
Response: The system creates the chat room and adds the user as a participant, displaying the new room in the user's list of active rooms.

Join Chat Room:
Stimulus: The user selects a chat room from the list and clicks "Join."
Response: The system adds the user to the chat room, and the chat interface for that room is displayed.

Leave Chat Room:
Stimulus: The user clicks "Leave Room."
Response: The system removes the user from the chat room and updates the user's interface accordingly.

3.3.3 Functional Requirements

REQ-14: The system shall allow users to create public and private chat rooms, specifying room name and visibility.

REQ-15: The system shall enable users to join existing chat rooms based on visibility settings and user permissions.

REQ-16: The system shall allow users to leave chat rooms, removing them from the list of active rooms.

REQ-17: The system shall display a list of all active chat rooms, indicating the number of participants in each room.

REQ-18: The system shall allow the room creator to delete the chat room, removing it and all associated messages from the system.

3.4 Direct Messaging

3.4.1 Description and Priority

The Direct Messaging feature allows users to send private messages to each other outside of chat rooms. This feature is of medium priority as it enhances personal communication but is not essential for basic functionality.

Benefit: 8
Penalty: 5
Cost: 5
Risk: 4

3.4.2 Stimulus/Response Sequences

Send Direct Message:
Stimulus: The user selects a contact and types a message in the direct message input field, then clicks "Send."
Response: The system sends the message and displays it in the conversation thread between the two users.

Receive Direct Message:
Stimulus: A direct message is sent to the user from another participant.
Response: The system immediately displays the incoming direct message in the user’s chat interface.

View Direct Message History:
Stimulus: The user selects a contact from their contact list.
Response: The system retrieves and displays the conversation history with that contact.

3.4.3 Functional Requirements

REQ-19: The system shall allow users to initiate direct messages with other users who are online.

REQ-20: The system shall display incoming direct messages immediately upon receipt.

REQ-21: The system shall provide a direct message history, allowing users to view previous conversations with other users.

REQ-22: The system shall allow users to delete direct messages, removing them from the chat history.

3.5 Notification System

3.5.1 Description and Priority

The Notification System feature alerts users to new messages and activities within the application. This feature is of medium priority as it improves user engagement and responsiveness.

Benefit: 8
Penalty: 4
Cost: 4
Risk: 3

3.5.2 Stimulus/Response Sequences

New Message Notification:
Stimulus: A user receives a new message in a chat room or direct message.
Response: The system triggers a notification alerting the user of the new message.

Typing Notification:
Stimulus: Another user is typing in a chat room.
Response: The system displays a typing indicator in the chat interface.

Missed Message Notification:
Stimulus: The user has missed messages while the application is minimized.
Response: The system displays a badge indicating the number of missed messages when the user returns to the application.

3.5.3 Functional Requirements

REQ-23: The system shall provide real-time notifications for new messages in chat rooms and direct messages.

REQ-24: The system shall display a visual indicator when another user is typing.

REQ-25: The system shall track missed messages and notify the user when they return to the application.

REQ-26: The system shall allow users to enable or disable notifications based on their preferences.

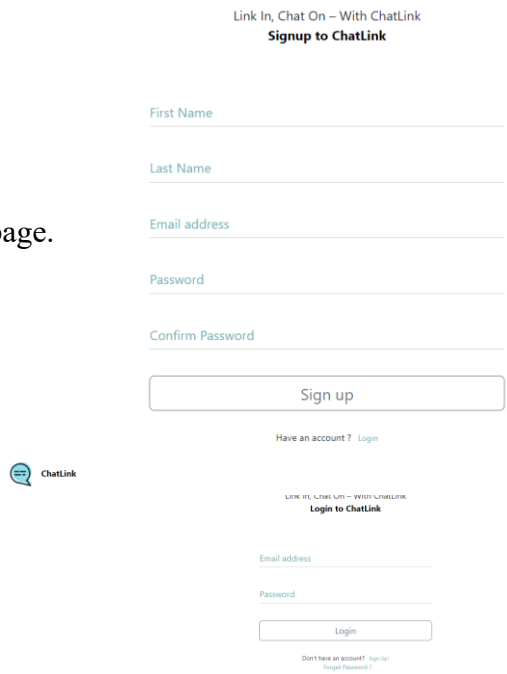
4. External Interface Requirements

4.1 User Interfaces

4.1.1 Overview

The user interfaces for the Real-Time Chat Application are designed to provide an intuitive and engaging user experience. This section describes the logical characteristics of each interface between the software product and the users, including GUI standards, screen layout constraints, standard buttons and functions, and other relevant details.

4.1.2 Interface Characteristics

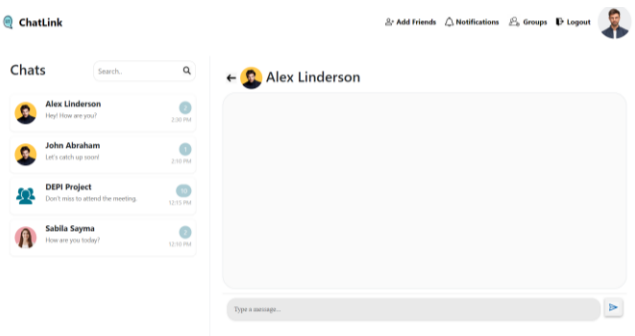


Sign Up Page:
Purpose: Allows new users to create an account.
Key Features:
Input fields for username, email, and password.
"Sign Up" button to submit the form.
"Already have an account? Log In" link redirecting to the login page.
Screen Elements:
Standard buttons: Submit, Clear.
Error message display for invalid input.

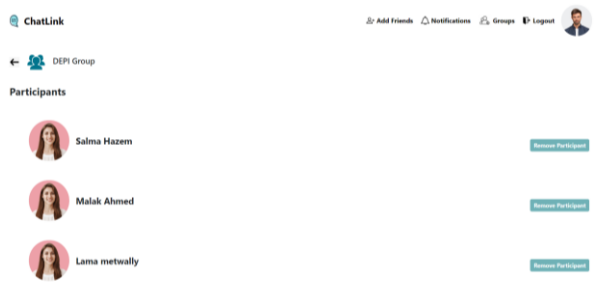
Log In Page:
Purpose: Allows existing users to log into their account.
Key Features: Input fields for email and password.
"Log In" button to submit the credentials.
"Forgot Password?" link for password recovery.
Screen Elements:
Standard buttons: Submit, Clear.
Error message display for invalid login attempts.

Main Page with Chat List:
Purpose: Displays the list of active chat rooms and direct messages.
Key Features:
Navigation bar for accessing different sections (e.g., Notifications, Add Friend, Settings).
Chat list showing recent conversations with unread message indicators.
"Create Room" button to initiate a new chat room.
Screen Elements:
Standard buttons: Create Room, Log Out.
Help button available for assistance.

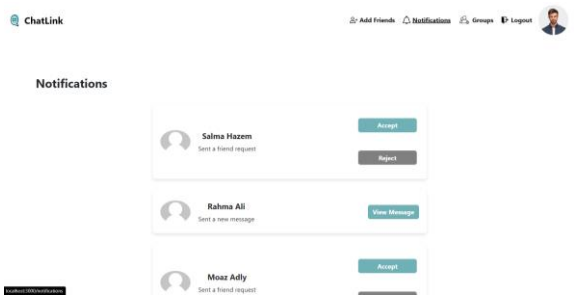
Chat Page:
Purpose: Displays the chat interface for a selected chat room or direct message.
Key Features:
Message input area for typing and sending messages.
Display of message history with timestamps.
Typing indicator for active participants.
Screen Elements:
Standard buttons: Send, Attach Files, Emoji Picker.
Error message display for failed message sends.



Member Participants Page in Any Group:
Purpose: Displays the list of members in a chat group.
Key Features:
Search bar to find specific members.
Option to add or remove participants.
Screen Elements:
Standard buttons: Add Member, Remove Member.
Member status indicators (online/offline).

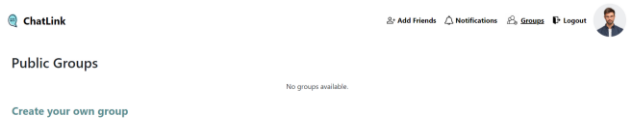


Notification Page:
Purpose: Displays all notifications related to messages and activities.
Key Features:
List of notifications with timestamps and message previews.
Options to mark notifications as read or delete them.
Screen Elements:
Standard buttons: Mark All as Read, Clear All Notifications.

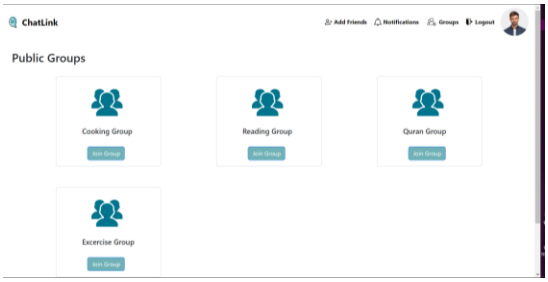


Add Friend Page:
Purpose: Allows users to search for and add friends.
Key Features:
Search bar for finding users by username or email.
List of friend requests with options to accept or decline.
Screen Elements:
Standard buttons: Send Friend Request, Accept, Decline.

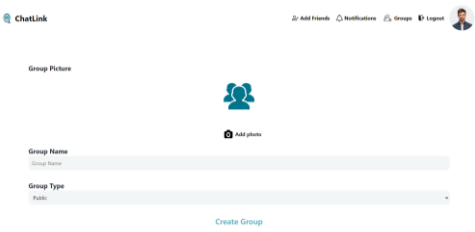
Group Page:
Purpose: Manages group-related functions.
Key Features:
Displays a list of public groups and options to create a new group.
"Create Your Own Group" page with settings for group name and photo.
Screen Elements:
Standard buttons: View Group, Create Group.
Error message display for invalid group settings.



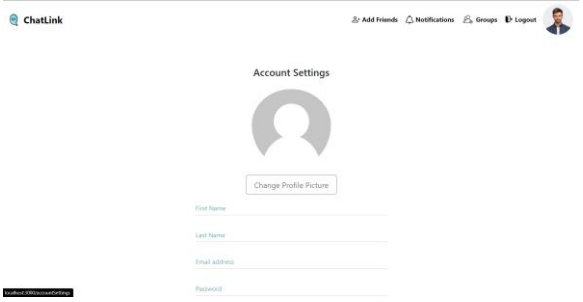
Public Groups Page:
Purpose: Displays a list of available public groups to join.
Key Features:
Search functionality to find groups.
Join button for each group.
Screen Elements:
Standard buttons: Join Group.



Create Your Own Group Page:
Purpose: Allows users to create a custom group.
Key Features:
Input fields for group name and upload group photo.
"Create Group" button to finalize group creation.
Screen Elements:
Standard buttons: Create Group, Cancel.



Account Settings Page:
Purpose: Allows users to manage their account settings.
Key Features:
Options to change password, email, and privacy settings.
Account deletion option.
Screen Elements:
Standard buttons: Save Changes, Cancel.



4.2 Software Interfaces

4.2.1 Overview

The Real-Time Chat Application interfaces with various software components to ensure functionality and efficiency. This section outlines the connections between the application and other specific software components, including databases, operating systems, libraries, and integrated commercial components.

4.2.2 Software Components and Versions

Frontend Framework:
Name: React
Version: 18.0.0
Purpose: Used for building the user interface of the chat application, enabling dynamic rendering and component-based architecture.

Backend Framework:
Name: Node.js
Version: 16.0.0
Purpose: Serves as the runtime environment for executing server-side JavaScript code, handling requests and responses for the application.

WebSocket Library:
Name: Socket.IO
Version: 4.0.0
Purpose: Facilitates real-time, bidirectional communication between the client and server for instant message delivery.

Database:
Name: MongoDB
Version: 5.0.0
Purpose: NoSQL database used for storing user data, chat messages, and notification data.

Authentication Library:
Name: jsonwebtoken (JWT)
Version: 8.5.1
Purpose: Used for securely generating and verifying tokens for user authentication.

4.2.3 Data Items and Messages

Incoming Data Items:

User Input:
- Username, email, and password during registration.
- Messages sent by users in chat rooms or direct messages.
- Friend requests and group management requests.

Purpose: Collects user data for account creation, message exchange, and social interactions.
Outgoing Data Items:

Chat Messages:
Sent messages are broadcasted to connected users in real-time.

Notifications:
Notifications about new messages, friend requests, and group activity.

Authentication Tokens:
JWT tokens issued upon successful login for subsequent requests.

Purpose: Communicates user actions, updates, and access permissions to ensure a responsive user experience.

4.2.4 Services and Communication

Real-Time Messaging Service:
Communication via WebSocket using Socket.IO to handle message exchanges between clients.
User Authentication Service:
RESTful API endpoints for user registration and login that utilize JWT for secure sessions.

Database Operations:
CRUD operations performed on MongoDB to manage user accounts, messages, and notifications.

4.2.5 Data Sharing Across Components

User Data: User profiles, including usernames, email addresses, and authentication tokens, are shared between the frontend and backend for user management.

Message Data: Chat messages are stored in MongoDB and retrieved by clients when they connect to chat rooms or direct message threads.

Notification Data: Notifications are stored in the database and fetched in real-time to keep users informed about new messages and friend requests.

4.2.6 Implementation Constraints

Data Sharing Mechanism: The application must use the MongoDB driver for Node.js to interface with the database. This must be implemented following MongoDB's connection and query protocols to ensure proper communication.

WebSocket Connection: The application must establish and maintain persistent WebSocket connections to enable real-time messaging, with fallback mechanisms for users on unsupported browsers.

JWT Token Expiry: The application must implement a token expiry mechanism to ensure that users re-authenticate after a specified period to maintain security.

4.2.7 References

MongoDB connection and query protocols are described in the MongoDB Documentation.

4.3 Communications Interfaces

The communications interfaces for this real-time chat application require several protocols and technologies to ensure smooth and secure operations:

WebSocket Protocol: The primary protocol for real-time communication in this application, used to create a persistent connection between the client and server, enabling low-latency message exchange.

HTTP/HTTPS: Used for initial connections, API requests, and data fetching. HTTPS ensures secure data transfer by encrypting communication between the client and server.

Message Formatting: JSON will be used as the message format for exchanging structured data between the frontend and backend.

Communication Security: TLS (Transport Layer Security) will be used to secure the WebSocket connections and prevent eavesdropping or man-in-the-middle attacks.

Synchronization Mechanisms: WebSocket's built-in mechanisms will be used to synchronize real-time messages and user interactions.

This section specifies how the application communicates with the server and ensures data security and efficiency.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The performance requirements for the chat application ensure that it runs smoothly under different scenarios, particularly when multiple users interact with the system in real time. Key performance

metrics include:

Response Time:
Messages must be delivered to the recipient within 1 second of being sent, ensuring real-time communication.
The user interface should respond to user interactions (e.g., sending a message, opening a chat window) in under 300 milliseconds.

Scalability:
The system should be able to support up to 10,000 concurrent users without a noticeable decline in performance.
It should handle up to 1,000 messages per second across all active users without delay.

Latency:
Network latency for message delivery should not exceed 150 milliseconds under normal conditions and 500 milliseconds under peak loads.

Data Throughput:
The system should support high-throughput operations to accommodate large volumes of messages, such as 50 MB of data transfers per second across the entire application.

Resource Utilization:
Server CPU usage should not exceed 75% under peak load.
Memory usage should remain below 80% capacity during peak times.

Error Handling:
The system should recover from network failures within 2 seconds and retry sending unsent messages within 5 seconds.

These requirements guide the design decisions and infrastructure choices, ensuring efficient and scalable performance across different user loads and scenarios.

5.2 Safety Requirements

The safety requirements for the chat application focus on ensuring the protection of users, data, and system integrity.

Key considerations include:

Data Loss Prevention:
All user messages and data must be backed up regularly to prevent loss in the event of system failure.

A recovery plan should be in place to restore lost data within 24 hours of any incident.

User Privacy and Security:
Personal data, including messages, should be encrypted (using protocols such as TLS/SSL) both in transit and at rest to prevent unauthorized access.
Users should be able to report and block harmful behavior or content.

Prevention of Harmful Actions:
The system should prevent the sending or receiving of malicious files (e.g., viruses, malware).
Ensure that no harmful scripts or actions can be triggered by user inputs within chats (e.g., XSS attacks).

Compliance with Regulations:
The application must adhere to GDPR (General Data Protection Regulation) for handling user data, including providing options for data deletion and consent management.
Follow local regulations regarding the storage and transmission of sensitive user data, especially in countries with strict privacy laws.

Safety Certifications:
If applicable, the software must be certified by relevant cybersecurity organizations or pass standard safety tests to ensure it operates securely in any environment. This could include certifications like ISO/IEC 27001 (information security) or SOC 2 compliance (data management).

By adhering to these requirements, the system minimizes risks such as data breaches, harmful content, and system failures, ensuring user safety and compliance with global regulations.

5.3 Security Requirements

The security requirements for the chat application focus on ensuring the protection of user data, maintaining user privacy, and preventing unauthorized access.

Key aspects include:

User Authentication and Authorization:
JWT (JSON Web Tokens) must be used to authenticate users. Each user must receive a token upon login that is securely verified with every interaction.
Multi-factor authentication (MFA) is required for sensitive operations (e.g., changing account settings, accessing administrative functionalities).
Role-based access control (RBAC) should be implemented to restrict access to specific features based on user roles (e.g., admin, regular user).

Data Encryption:
All user data, including chat messages, must be encrypted in transit using TLS (Transport Layer Security) and at rest using AES-256 encryption.
Sensitive data, such as passwords, must be hashed using a strong hashing algorithm (e.g., bcrypt) before being stored in the database.

Privacy and Data Protection:
The application must adhere to privacy regulations such as GDPR and CCPA to protect user data and ensure user consent before data collection.
Users must be given control over their data, including the ability to delete their accounts and associated data upon request.

Security Monitoring and Incident Response:
The system must log all authentication attempts, successful and unsuccessful, and monitor for suspicious activities such as repeated login failures (brute-force attacks).
A security incident response plan must be in place to handle breaches, including notifying affected users and providing them with support to secure their accounts.

API Security:
All API endpoints must be protected with proper authentication and authorization, ensuring that users can only access data and actions they are authorized to perform.
The system should implement rate limiting and IP blocking mechanisms to prevent DoS (Denial of Service) attacks or API abuse.

Compliance and Certifications:
The software must comply with security standards like ISO/IEC 27001 (information security management) and SOC 2 Type II for securing data in cloud environments, ensuring data integrity, and protecting user privacy.

By meeting these security requirements, the system ensures the safety of user data, prevents unauthorized access, and complies with relevant privacy laws.

5.4 Software Quality Attributes

The software quality attributes for the chat application include the following:

5.4.1 Usability:
The system must be intuitive and easy to navigate for both technical and non-technical users, allowing seamless onboarding and daily use. Usability testing will ensure that users can efficiently perform tasks such as sending messages, creating groups, and adjusting settings without needing extensive guidance.

5.4.2 Reliability:
The system must ensure reliable performance, with an uptime of 99.9%, ensuring that users can access the application at any time without disruption. This includes handling high traffic loads without failure.

5.4.3 Scalability:
The software must be scalable to handle an increasing number of users and messages without significant performance degradation. This includes supporting multiple concurrent users and enabling horizontal scaling through distributed systems or cloud infrastructure.

5.4.4 Security:
Security is paramount, ensuring that user data is protected from breaches, and unauthorized access is prevented. Encryption protocols, secure authentication methods, and real-time security monitoring must be in place.

5.4.5 Maintainability:
The software should be designed with modularity to facilitate easy updates and bug fixes. Code documentation, adherence to coding standards, and automated testing will support efficient maintenance.

5.4.6 Interoperability:
The system must integrate with external APIs and platforms seamlessly, enabling features such as real-time notifications, cloud storage, and third-party authentication services (Google, Facebook).

5.4.7 Performance:
Response times for key actions, such as sending messages or loading chats, must be less than 1 second under normal conditions. The application should handle peak loads without significant delays, especially during group chats or high message traffic.

5.4.8 Adaptability:
The system should allow for the easy addition of new features, such as integrating with new messaging protocols or supporting new device types (smartphones, tablets) without major restructuring.

5.4.9 Testability:
The system must be built with automated testing in mind, allowing for unit, integration, and regression tests. The ability to easily test individual components and overall system functionality is key to maintaining high-quality software.

5.4.10 Robustness:
The system should handle unexpected inputs or errors gracefully, ensuring that it doesn’t crash or expose vulnerabilities. Error handling and validation mechanisms must be in place to ensure system integrity.

By focusing on these attributes, the system ensures a balance of performance, usability, security, and maintainability, delivering a high-quality experience to users and developers alike.

6. Other Requirements

Database Requirements:
The system must use MongoDB for storing user data, messages, notifications, and chat history.
Data redundancy and replication mechanisms must be in place for fault tolerance.
All data must be encrypted at rest.

Internationalization Requirements:
The system should support multiple languages, starting with English and Arabic. This includes displaying text and dates in the appropriate format based on the user's locale.
Right-to-left text support should be included for Arabic.

Legal Requirements:
The system must comply with GDPR (General Data Protection Regulation) for data protection and user privacy.
All user data handling should follow applicable regional laws and regulations concerning personal information security.

Reuse Objectives:
The application should use modular, reusable components wherever possible to allow for future integration with other messaging or social applications.

Code and design should be documented to allow parts of the application (such as notification systems or authentication) to be reused in other projects.

Appendix A: Glossary

- SRS: Software Requirements Specification
- UI: User Interface
- JWT: JSON Web Token, used for user authentication
- API: Application Programming Interface, defines communication between software components
- GDPR: General Data Protection Regulation, an EU regulation for data privacy

Appendix B: Analysis Models

1- Sequence Diagram

1.1. User Authentication:

In the sequence diagram for user authentication, the user interacts with the system by providing their credentials (e.g., username and password).

Steps:

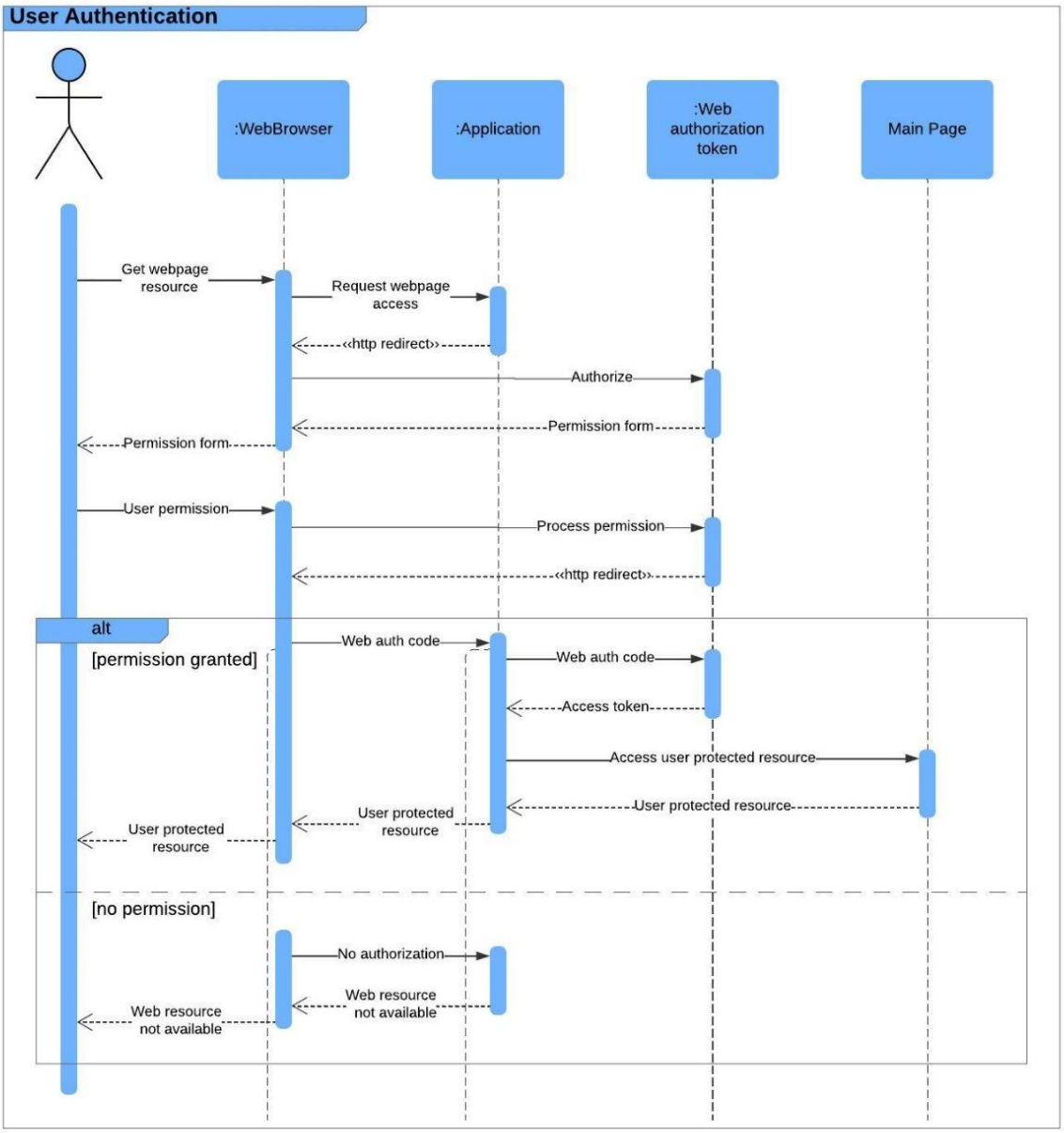
The user submits the login form.

The front-end sends a login request to the authentication server.

The authentication server checks the credentials with the user database.

If credentials are valid, the server generates a JWT token and returns it to the user.

The user is granted access to the system by including the token in subsequent requests.



1.2 Create Group:

The create group sequence diagram outlines how a user creates a new chat group.

Steps:

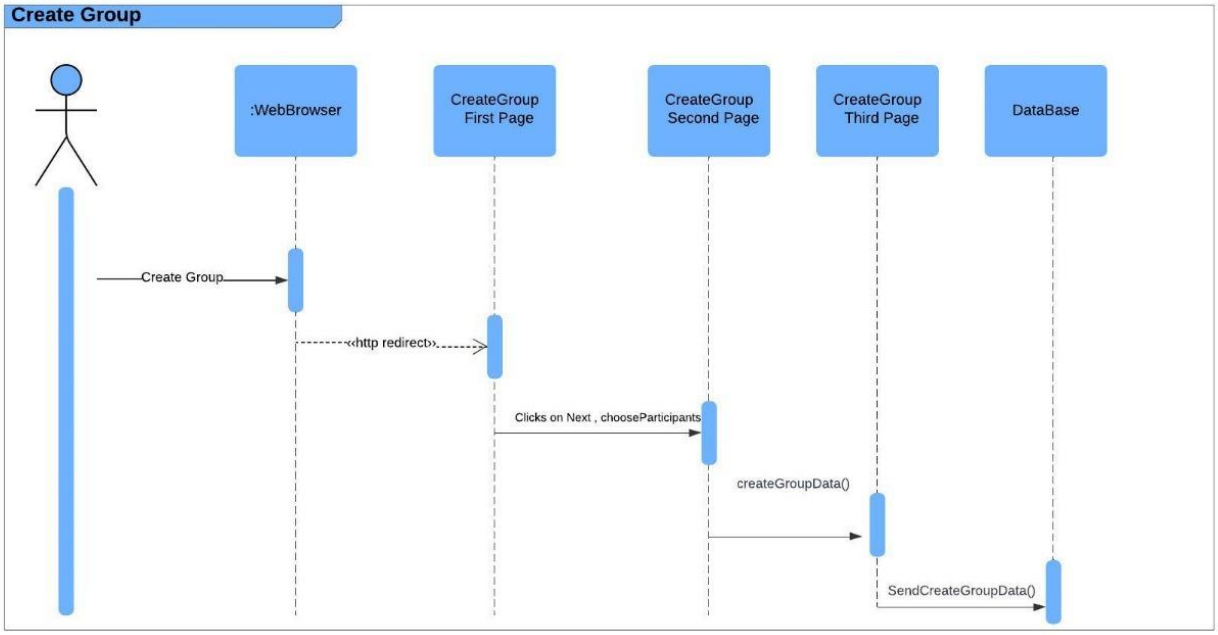
The user selects the option to create a group and provides necessary details (e.g., group name, photo).

The front-end sends a create group request to the server.

The server processes the request, creating a new group entry in the database.

The server may assign the creator as the admin.

A confirmation message is returned to the user, and the new group appears in the group list.



1.3 Remove Participant:

The remove participant sequence diagram shows how an admin removes a participant from a group.

Steps:

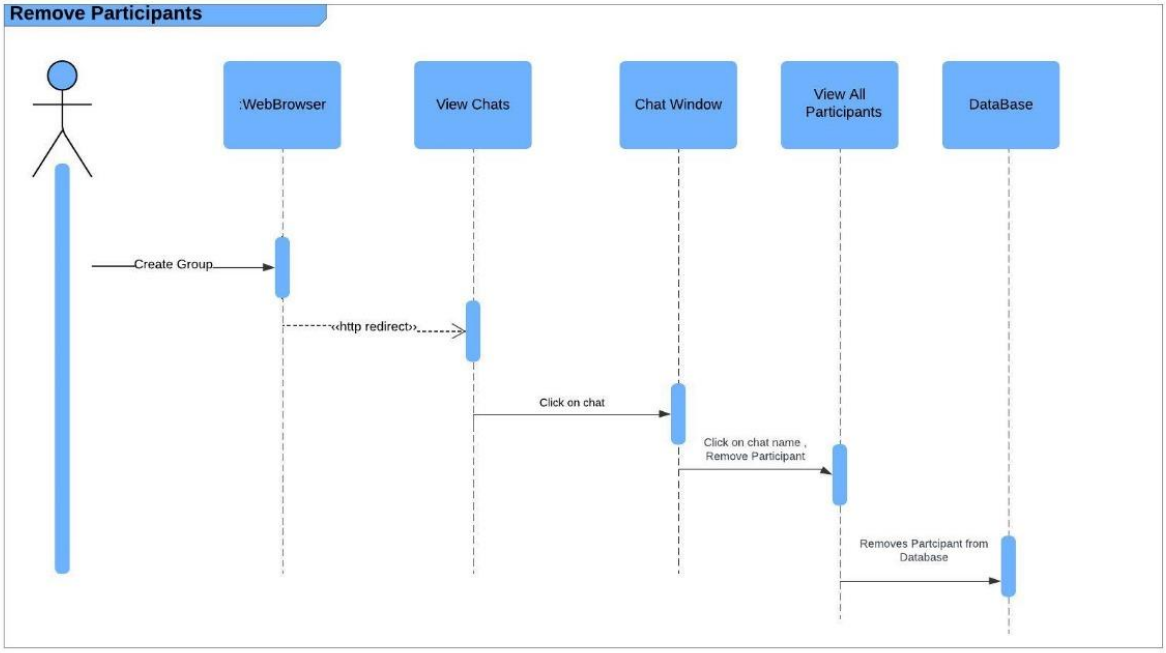
The admin selects the participant to be removed from the group.

The front-end sends a request to the server to remove the participant.

The server checks if the user making the request has admin rights.

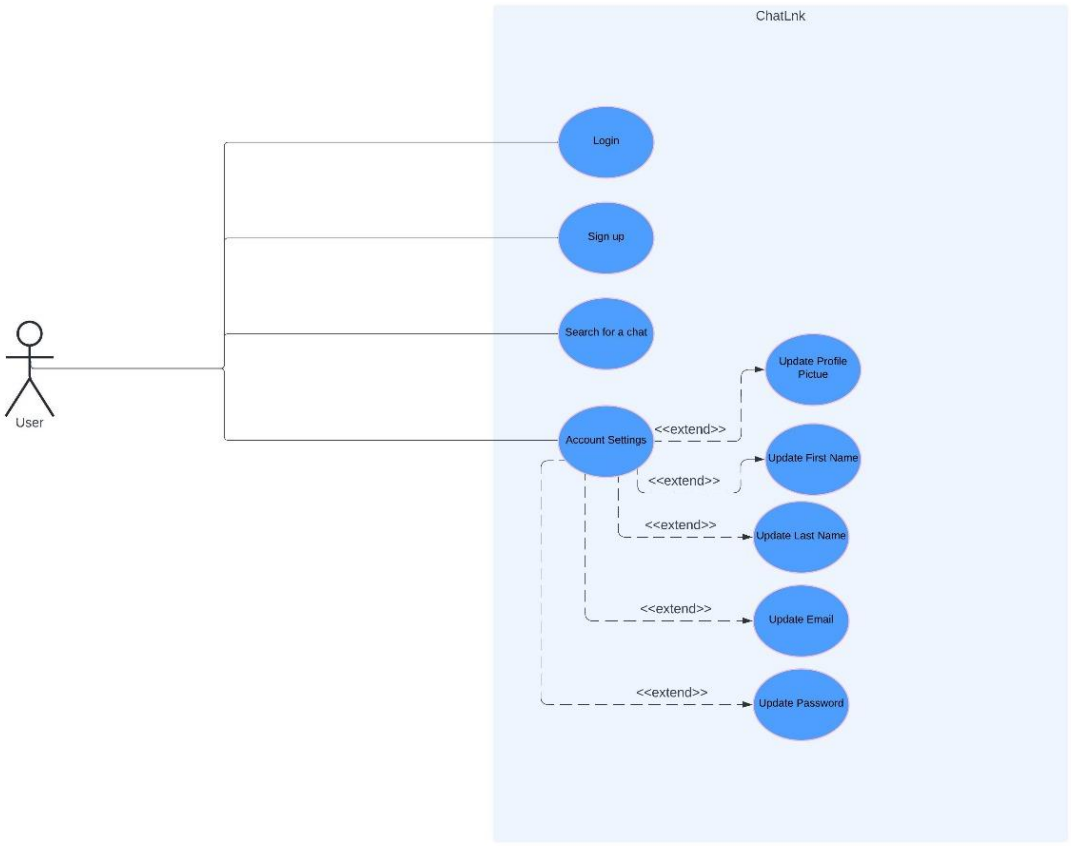
If authorized, the server updates the group participant list in the database, removing the selected user.

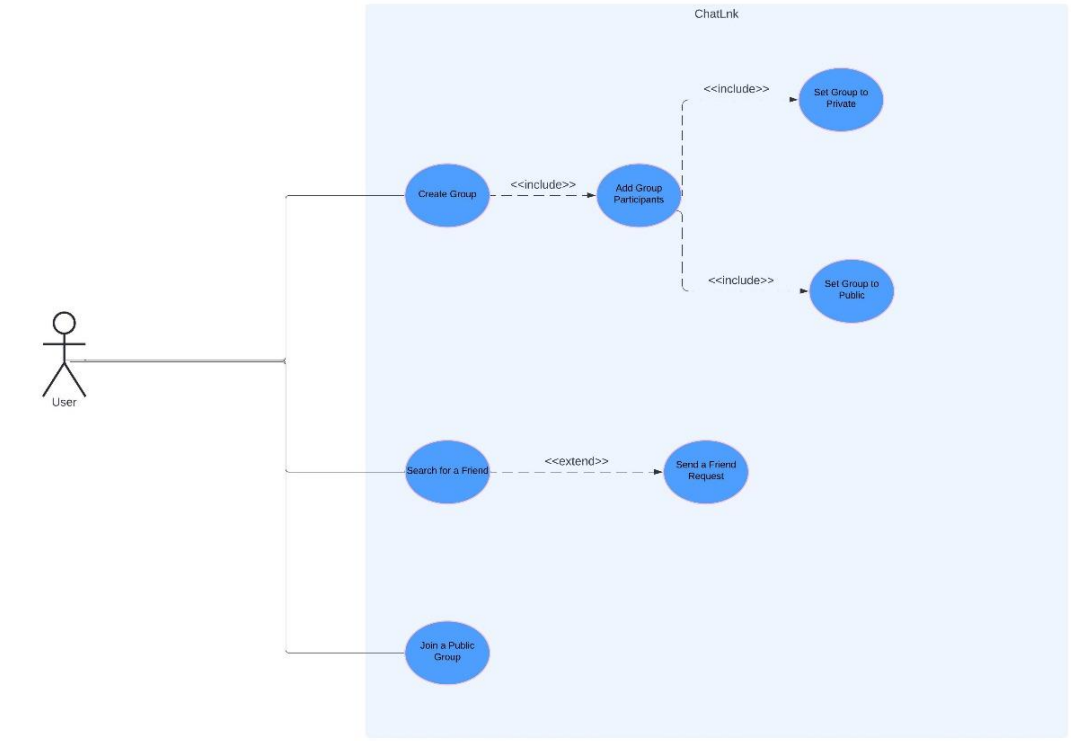
A confirmation is sent to the admin, and the user is removed from the group.

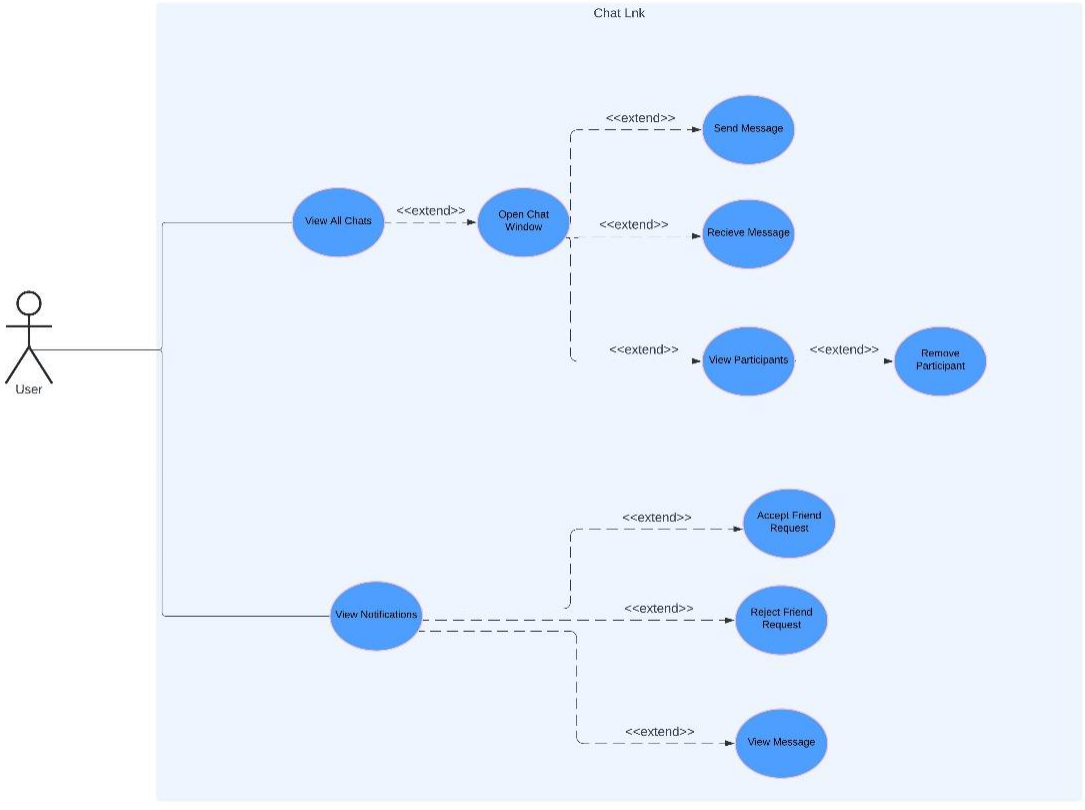


2- Use Case Diagram

A use case diagram illustrates the interactions between users (actors) and the system, highlighting key functionalities (use cases) within the system's scope. It visually represents how different actors engage with the system, providing a clear overview of user goals and system responses.




ChatLnk
<<include>>
Set Group to Private
Create Group
<<include>>
Add Group Participants
<<include>>
Set Group to Public
User
Search for a Friend
<<extend>>
Send a Friend Request
Join a Public Group


Chat Lnk
<<extend>>
Send Message
View All Chats
<<extend>>
Open Chat Window
<<extend>>
Recieve Message
User
<<extend>>
View Participants
<<extend>>
Remove Participant
<<extend>>
Accept Friend Request
View Notifications
<<extend>>
Reject Friend Request
<<extend>>
View Message

3- Data Base schema