CS/SE/CE 3354 Software Engineering

Final Project Deliverable 2

SMS Messenger Software Project (Sibyl)

Benjamin Welch

Harry Seo

Jay Jhamb

Justin Pham

Mingxuan Wei

Naman Mishra

Saharsha Magar

Samantha Boden

1. Delegation of tasks

- Benjamin Welch was responsible for creating the case diagram, which will map out the interactions between users and the system, as well as doing the cost estimations for our product.
- Harry Seo managed the delegations of tasks for the project and ensured that responsibilities were evenly distributed among the team, and helped develop the test case for our project with Mingxuan Wei, which tested our peer-to-peer messaging system.
- Jay Jhamb created the sequence diagram, which outlines the sequence of processes in the system, and did the comparisons between our product and ones of similar design that are on the market.
- Justin Pham was responsible for the class diagram, which defined the structure of the system (including classes, their attributes, and relationships), as well as wrote the conclusion of the project's work, describing any changes and evaluating the work done.
- Mingxuan Wei specified the non-functional requirements that focused on system performance, scalability, security, and usability, and was the other person that helped Harry Seo with the test case for our project.
- Naman Mishra outlined the functional software requirements which identified and specified what the system should do from a user's perspective, as well as working on the powerpoint presentation with Saharsha Magar.
- Saharsha Magar defined the software process model, which guided us on how the project will be developed and maintained, as well as helping out Naman Mishra with the powerpoint presentation.
- Samantha Boden designed the overall system architecture, detailing how the components of the system will interact with each other, documenting the scheduling of the project, and helped edit the deliverables to be ready to be turned in.

2. Deliverable 1 Content:

2.1 For our project, we will be creating an SMS messenger service called Sibyl. With the implementation of P2P (peer-to-peer) networking, local database storage, and quantum cryptography, we plan to create a messaging service that allows all the generic features of mainstream SMS applications and integrate quantum cryptography to ensure safe and secure messaging. Our team consists of members with experience ranging in cybersecurity to understanding of distributed application architecture, and with all members having experience in using SMS messaging software.

2.2 URL: https://github.com/bukoj/3354-sibyl

2.3 Individual Task Delegation

- Benjamin Welch is responsible for creating the case diagram, which will map out the interactions between users and the system.
- Harry Seo is managing the delegations of tasks for the project, ensuring that responsibilities are evenly distributed among the team.
- Jay Jhamb will create the sequence diagram, which outlines the sequence of processes in the system.
- Justin Pham will be responsible for the class diagram, which will define the structure of the system (including classes, their attributes, and relationships).
- Mingxuan Wei will specify non-functional requirements, focusing on system performance, scalability, security, and usability.
- Naman Mishra will outline the functional software requirements, identifying and specifying what the system should do from a user's perspective.
- Saharsha Magar will define the software process model, which will guide how the project will be developed and maintained.
- Samantha Boden will design the overall system architecture, detailing how the components of the system will interact with each other.

2.4 Address the feedback provided in the Final Project Proposal

- For our peer-to-peer SMS Messenger Software (Sibyl), the spiral model combined with agile methodologies will be employed as peer-to-peer networks involve several hazards, including network stability, message delivery dependability, security (for example, message encryption), and data synchronization among users. The Spiral Model allows us to address these risks early on by constructing prototypes or performing experiments during each iteration to evaluate alternate networking protocols, encryption methods, and database formats while agile practices such as customer collaboration, simplicity, and responsiveness, can be added within each cycle. If any of the risks prove to be unmanageable, the project can be redirected.

This model also enables us to construct essential capabilities, such as basic messaging and database storage, in early rounds before gradually improving features such as message drafts, editing, querying, and encryption in succeeding spirals. Each iteration builds on the previous one, refining the system and incorporating feedback or solutions to problems encountered along the way.

2.5 Describe which software process model is employed in your project and why it was the choice.

- For Sibyl, it is critical that functional demands are clearly defined and leave no room for misinterpretation. These are the designated requirements:

  2.5.1 As previously defined, a peer-to-peer networking system should be put in place which would allow Sibyl to cut the latency between messages and provide added privacy as information would not have to be stored in a central server (which could pose a security risk).

  2.5.2 As a text-messaging application, users shall be able to send and receive messages including attachments (photos/videos/links) with a negligible time delay between messages thanks to P2P.

  2.5.3 The system shall allow the users to sign up and login (if needed, on default the user should be logged in) for the service using their phone numbers. A password will not be required as long as 2FA is set up with an email account and text messaging capabilities (so those functionalities will need to be implemented).

  2.5.4 A user notification system shall be able to provide real-time updates regarding new messages being sent from added friends, new requests to connect and software updates should be implemented.

  2.5.5 The system shall utilize cryptographic methods such as Quantum Key Distribution (QKD) to secure messages between users providing real-time protection against unauthorized access from bad actors.

  2.5.6 In terms of adding contacts, users shall be able to send requests for adding, removing, and restricting friends based on their needs. The system shall also allow users to block any contact with users they choose to refrain from conversing with.

  2.5.7 Users shall be able to improve their interface experience with the added features of changing themes and colors of both the actual application and within direct messages as well (both users must be able to make/see any updates to the current theme which starts as a default picture).

3. Requirements

3.1 Product Requirements

The SMS Messenger system must deliver messages promptly, with an average response time under 2 seconds to ensure real-time communication. It should be capable of supporting multiple devices and platforms without compromising performance, allowing users to send and receive messages seamlessly across desktop and mobile environments.

3.2 Organizational Requirements

The system should be easily maintainable, allowing updates and bug fixes to be deployed quickly without disrupting user experience. Compatibility with existing IT infrastructure and integration into the organization's software development lifecycle is necessary to streamline future enhancements and support.

3.3 External Requirements

The SMS Messenger system must adhere to data privacy regulations like GDPR and HIPAA, implementing encryption and secure access protocols to protect user data during transmission and storage. This compliance is crucial to maintaining trust and legal accountability while interacting with external systems and services.

3.4 Security Requirements

The system should have robust security measures, including rate limiting and anomaly detection, to protect against spamming and brute-force attacks. All sensitive user data, including messages and contact information, should be encrypted both in transit and at rest using AES-256 encryption. Additionally, user sessions should time out after 15 minutes of inactivity to prevent unauthorized access in case of unattended devices.

3.5 Usability Requirements

The interface should be intuitive and user-friendly, allowing users to send, receive, and search messages with minimal effort. Users should be able to customize their message notifications, such as muting specific chats or scheduling "Do Not Disturb" times. The design should be responsive and provide a consistent experience across different screen sizes and resolutions, including smartphones, tablets, and desktops.

3.6 Compatibility Requirements

The system should be compatible with all major web browsers (Chrome, Firefox, Safari, and Edge) and mobile operating systems (iOS and Android). It should function correctly on devices with varying screen resolutions and orientations. The app should support integration with external services like email clients or cloud storage solutions, allowing users to back up their message history if desired.

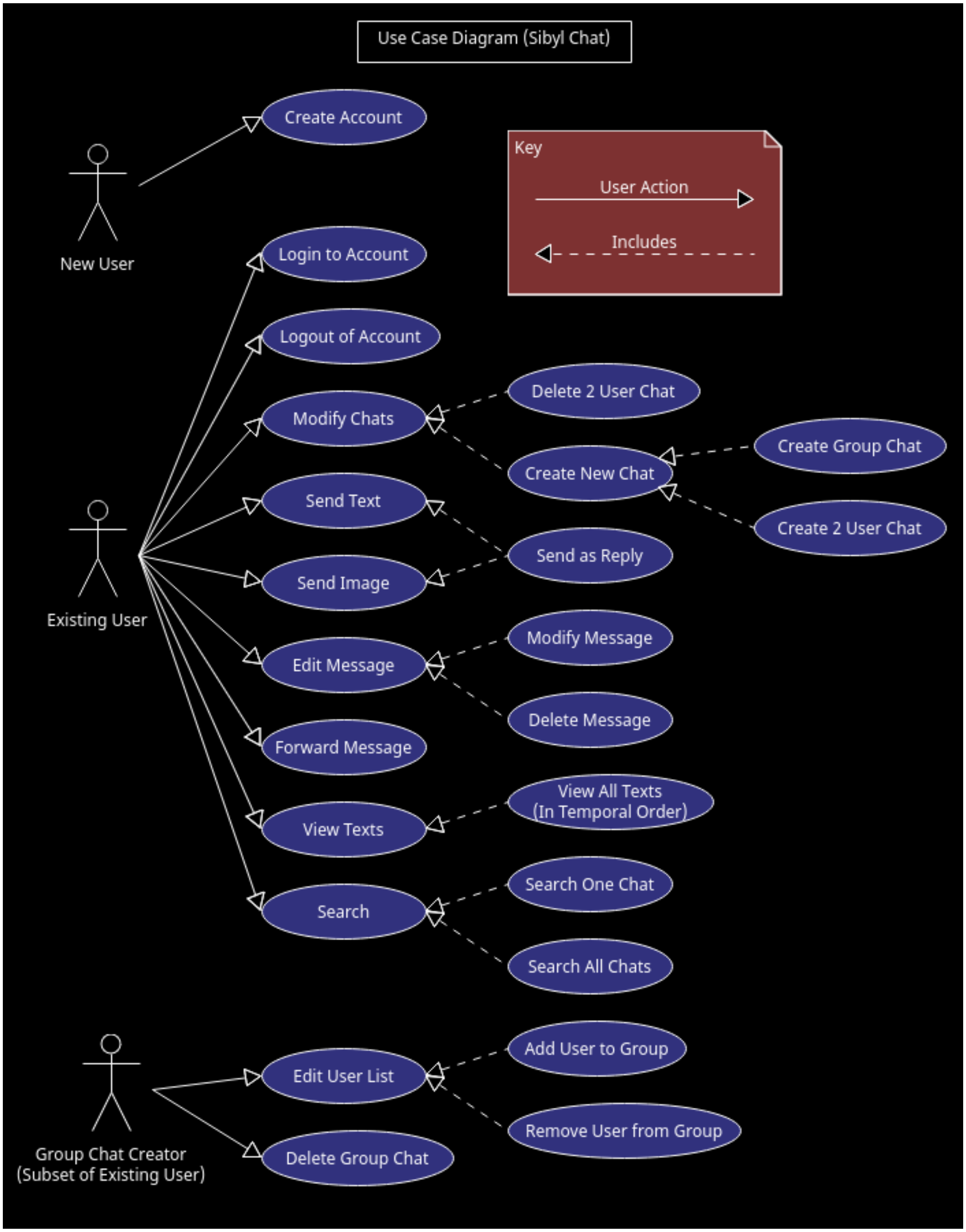
3.7 Availability Requirements

The system should be available and working almost all the time, with very little downtime each month. Planned maintenance should be done when fewer people are using it, and users should be informed ahead of time. If something goes wrong unexpectedly, the system should be able to go back to normal within 15 minutes.

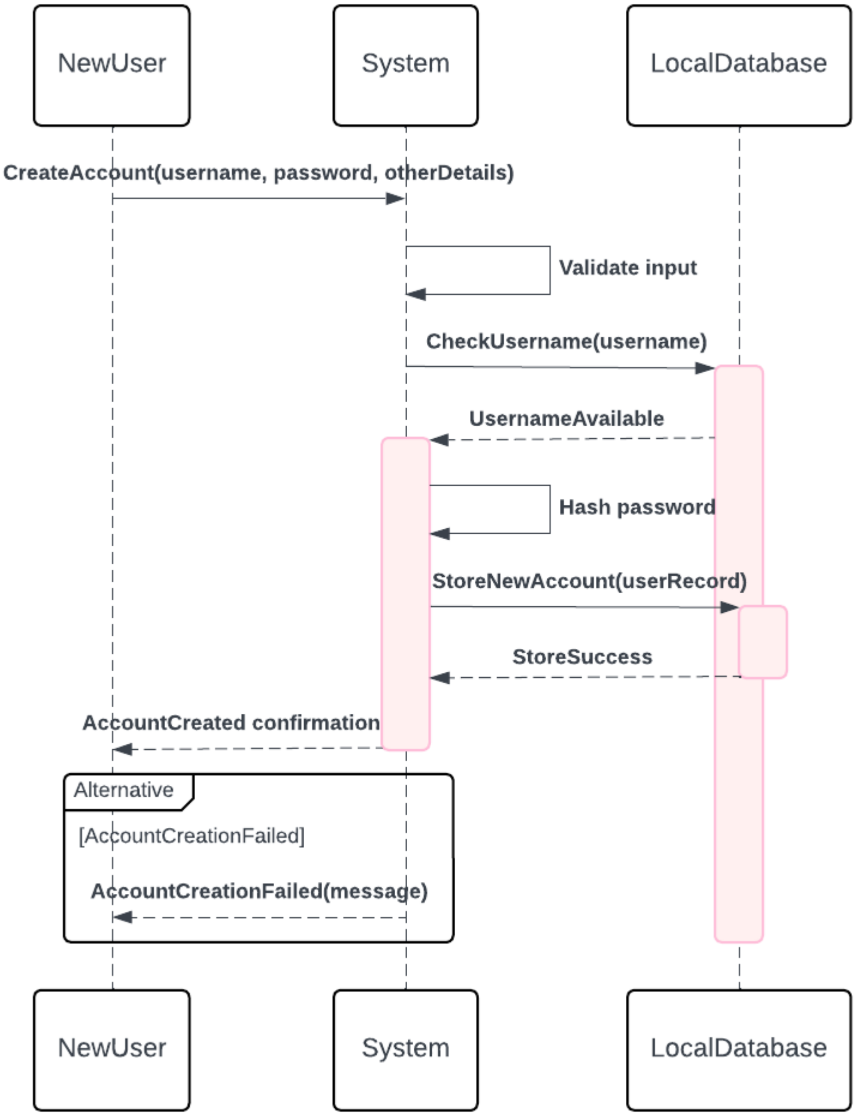
3.8 Maintainability Requirements

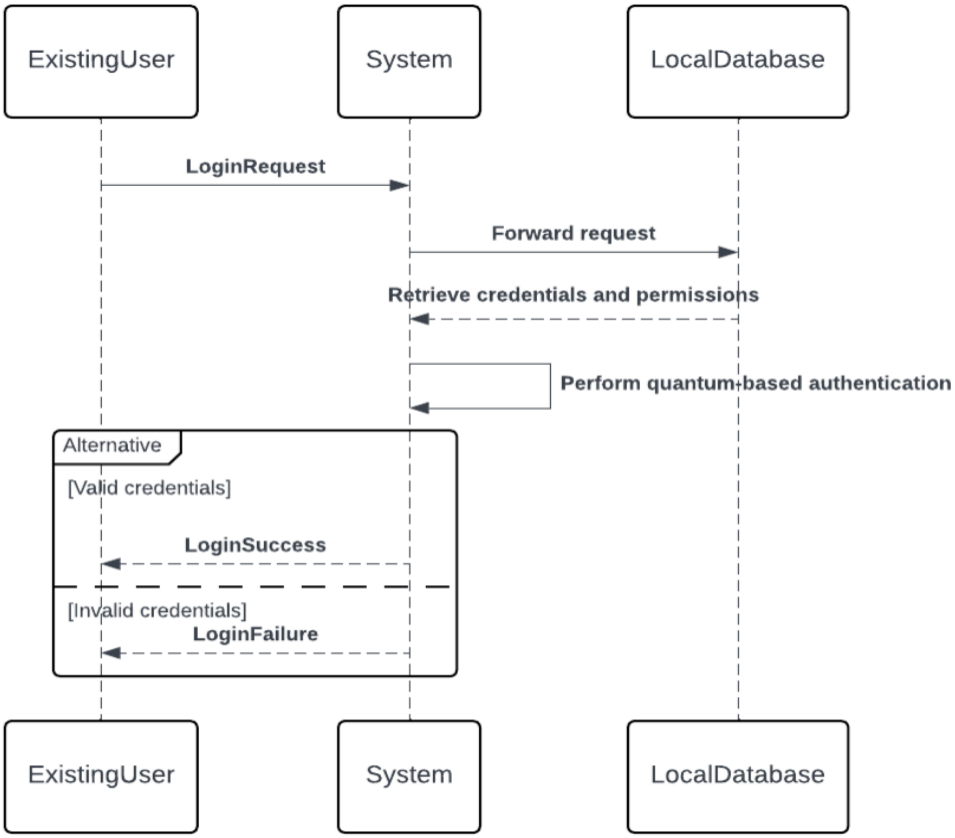
The system should be easy to update and fix. The code should be simple to understand so that new developers can quickly learn and make changes. Automated tests should be used to catch most errors early, and every major change should have a plan to undo it if something goes wrong.

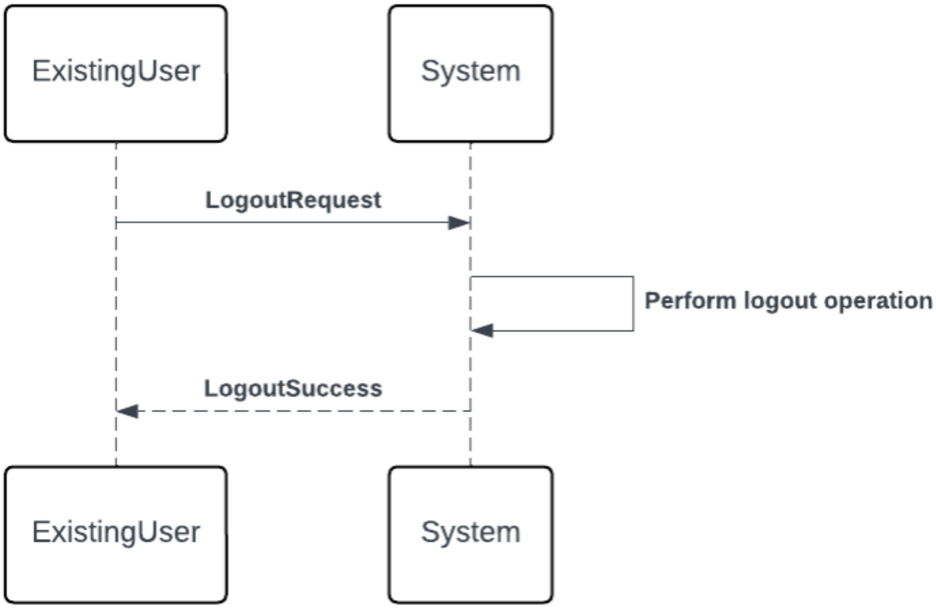
4. Use case diagram



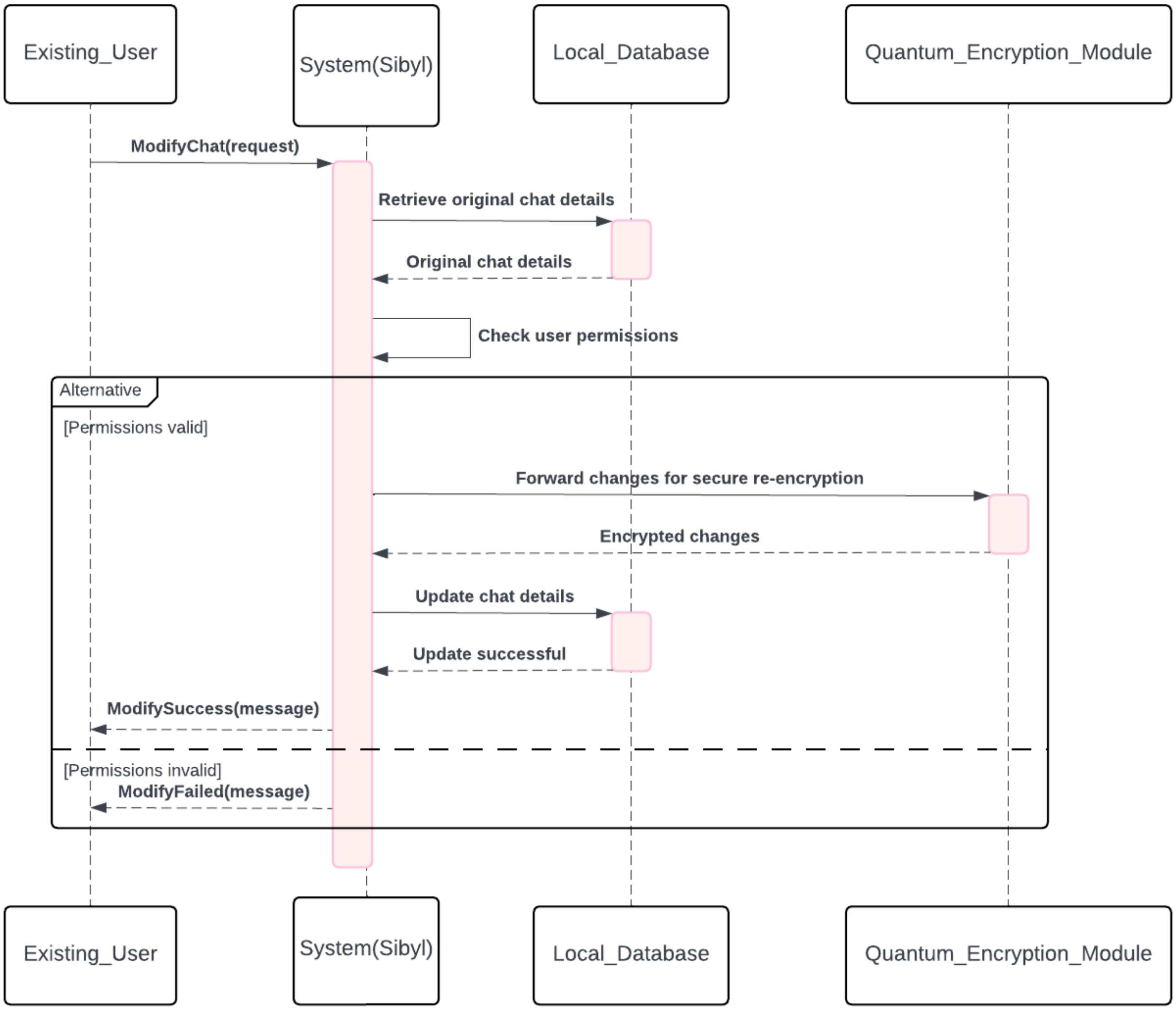
5. Sequence diagram



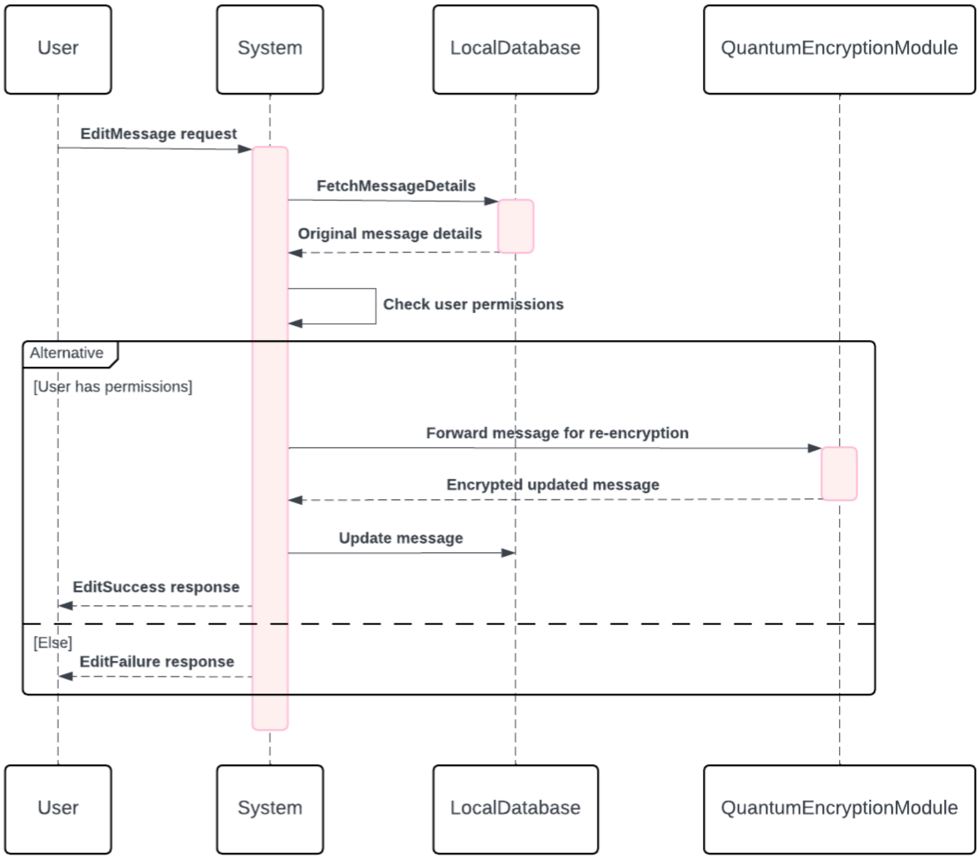

ExistingUser
System
LocalDatabase
LoginRequest
Forward request
Retrieve credentials and permissions
Perform quantum-based authentication
Alternative
[Valid credentials]
LoginSuccess
[Invalid credentials]
LoginFailure
ExistingUser
System
LocalDatabase


ExistingUser
System
LogoutRequest
Perform logout operation
LogoutSuccess
ExistingUser
System

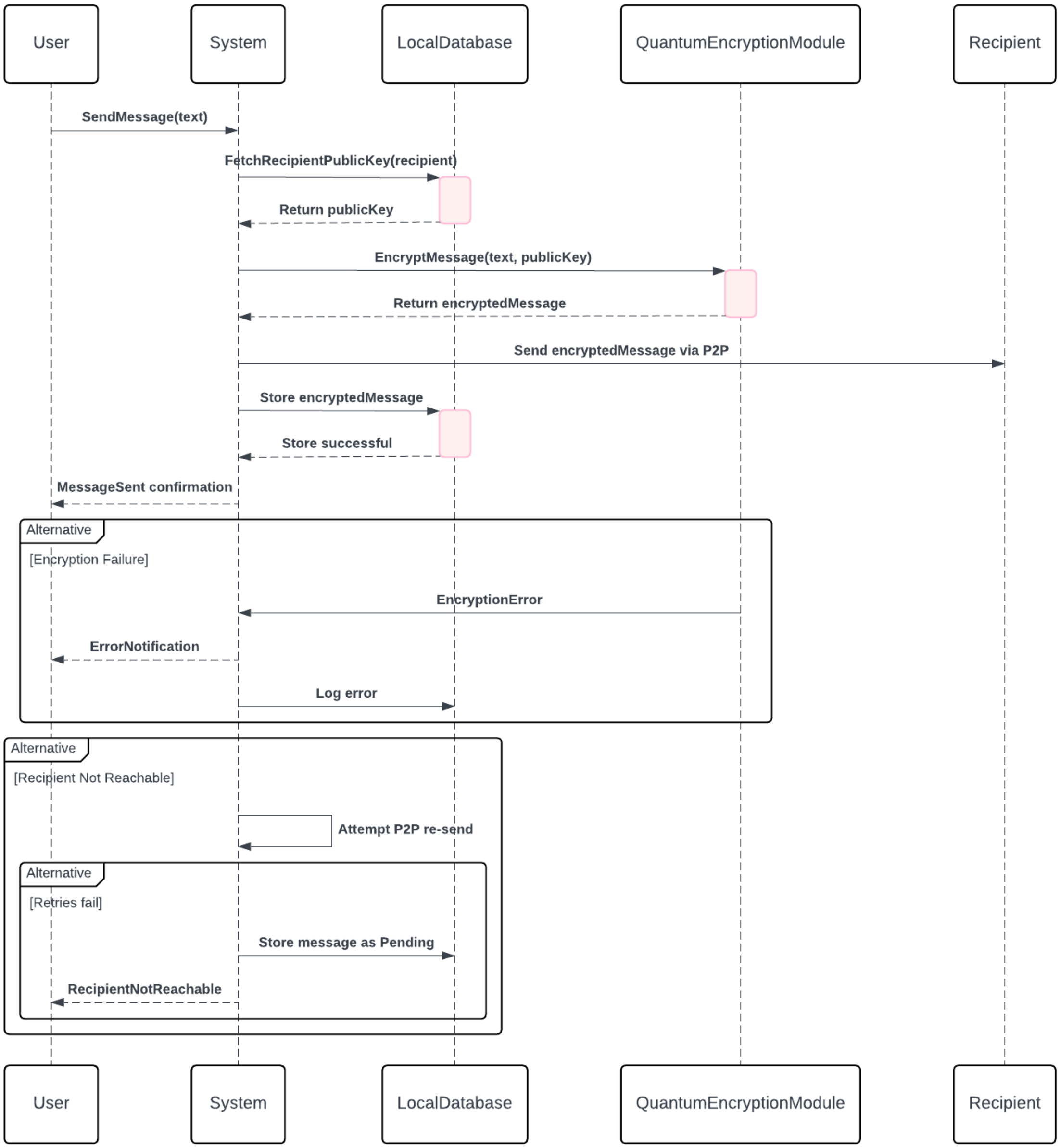
Modify Chat use case:



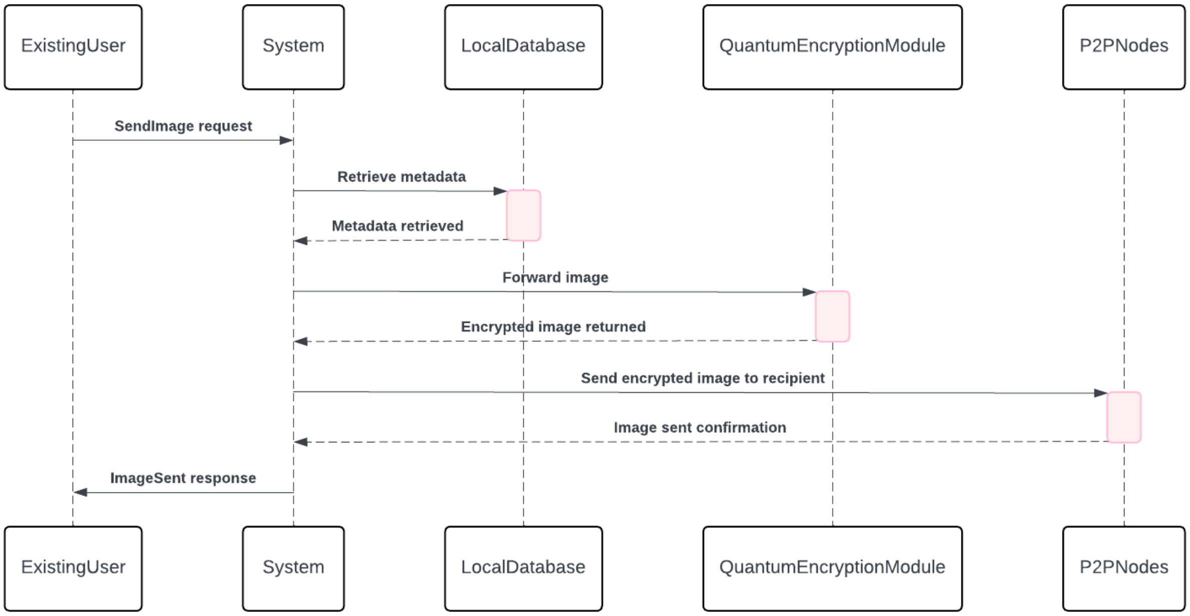
Edit Message use case:



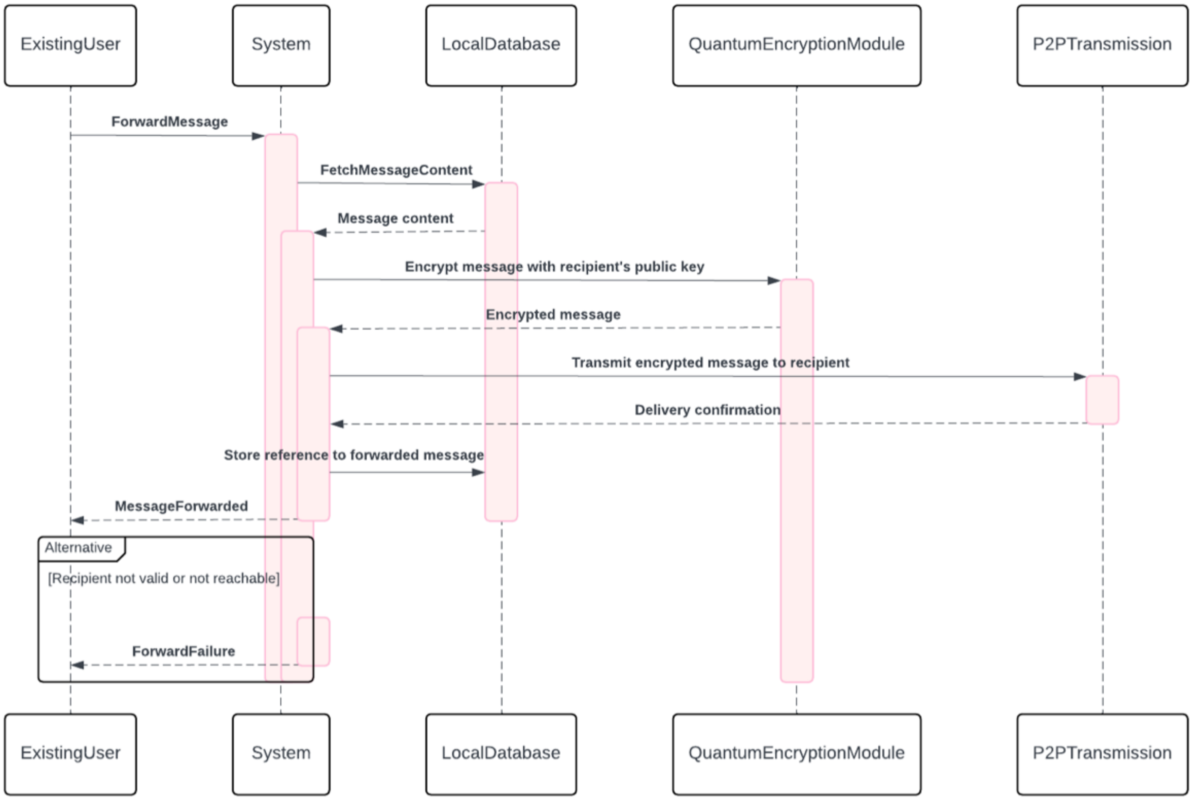
SendMessage use case:



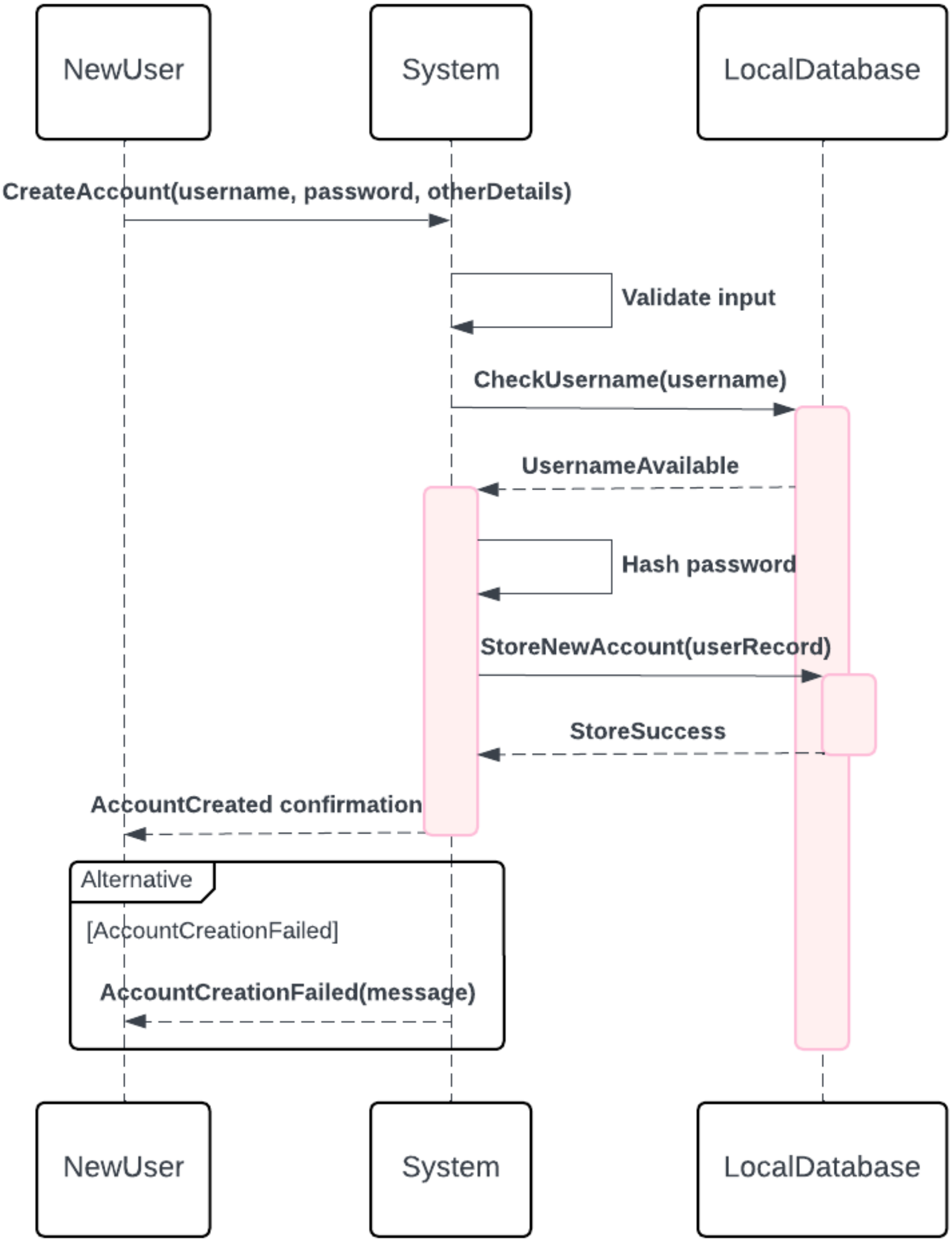
Send image use case:



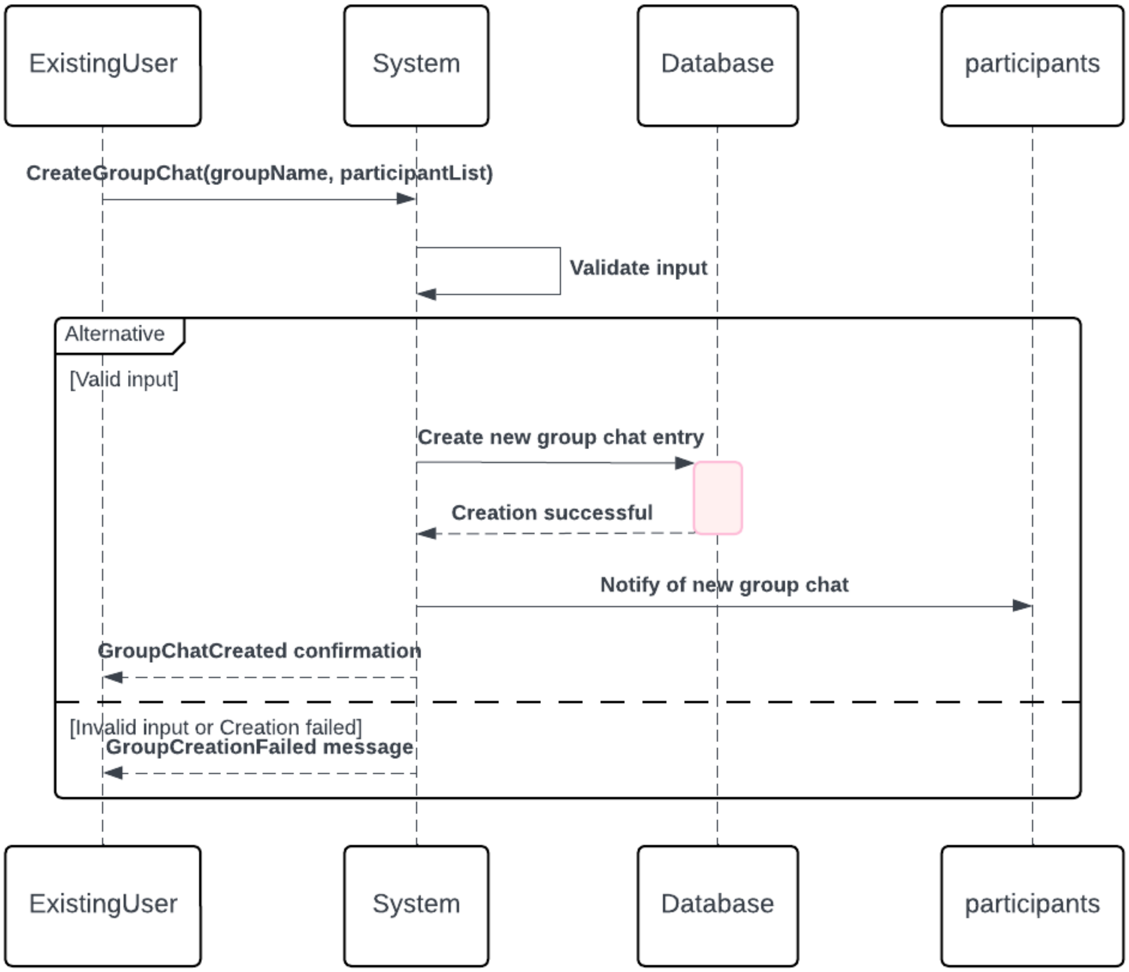
Forward Message use case:



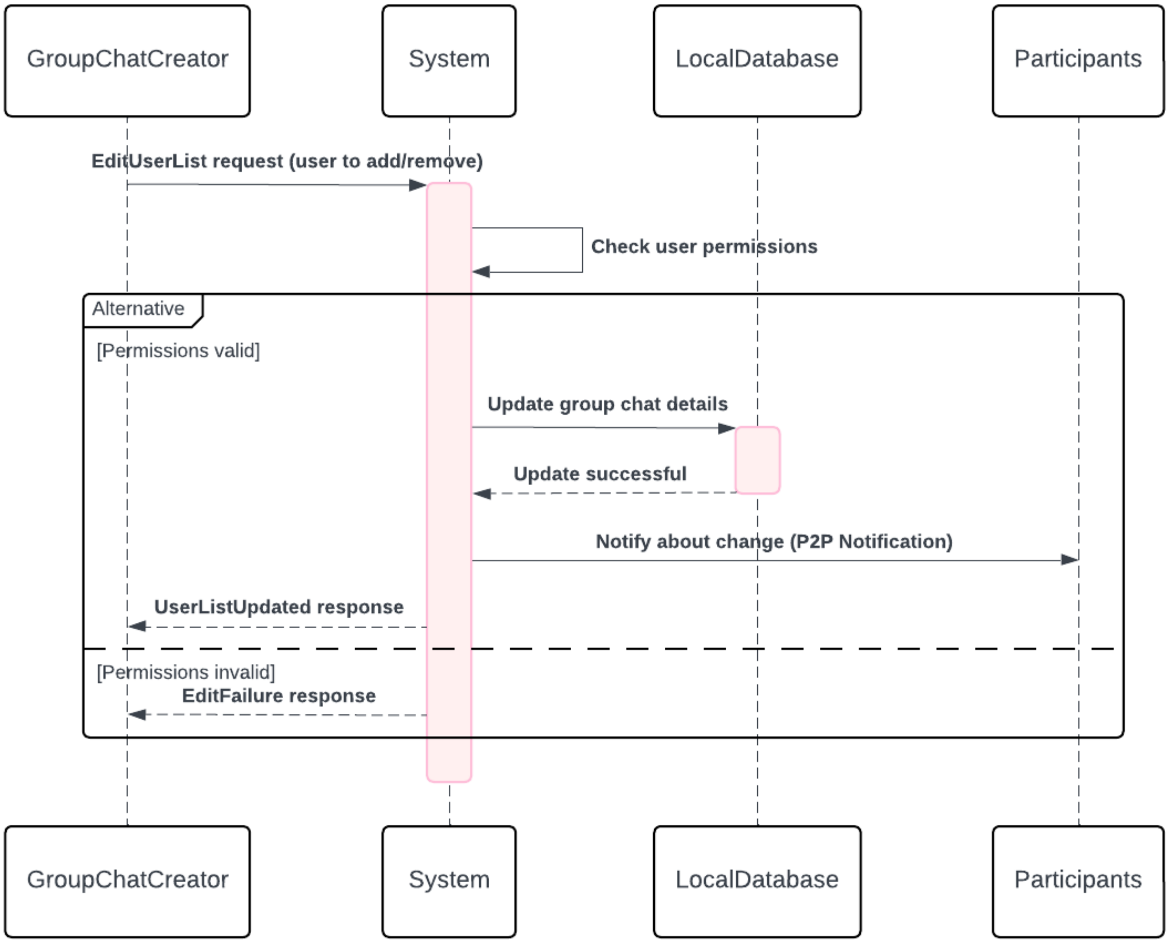
View Texts use case:



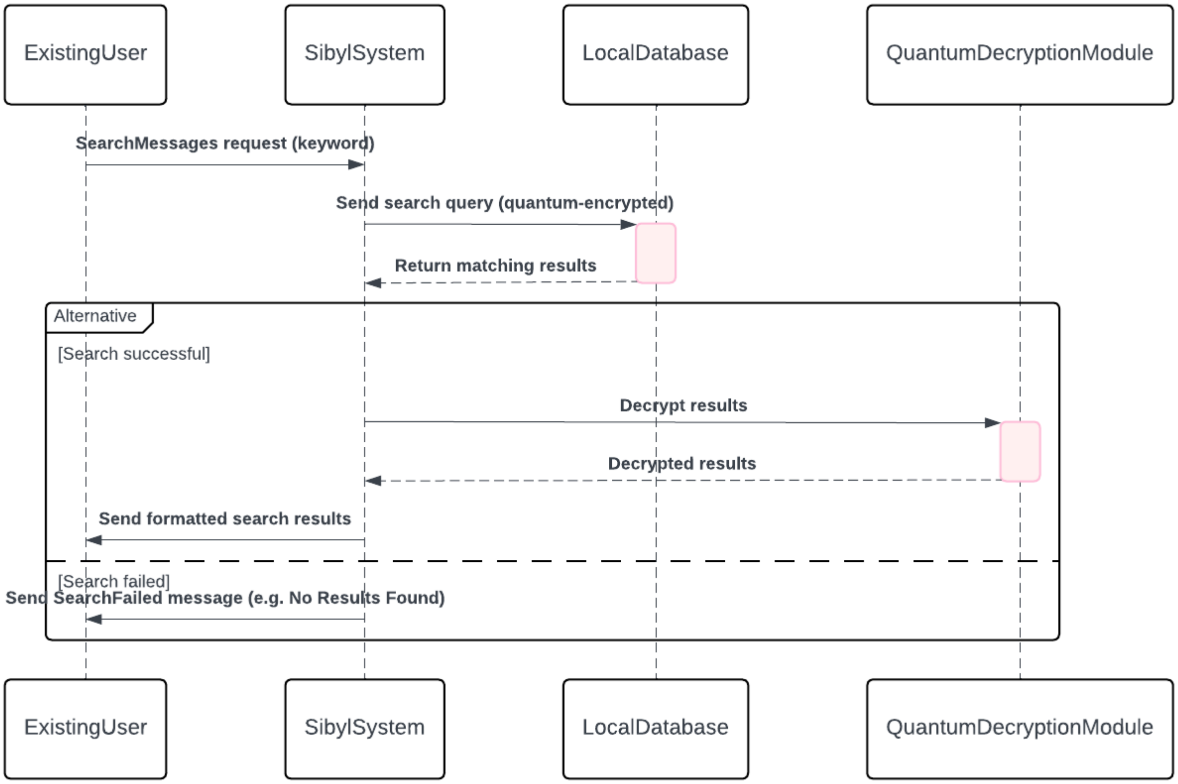
Create group chat use case:


ExistingUser
System
Database
participants
CreateGroupChat(groupName, participantList)
Validate input
Alternative
[Valid input]
Create new group chat entry
Creation successful
Notify of new group chat
GroupChatCreated confirmation
[Invalid input or Creation failed]
GroupCreationFailed message
ExistingUser
System
Database
participants

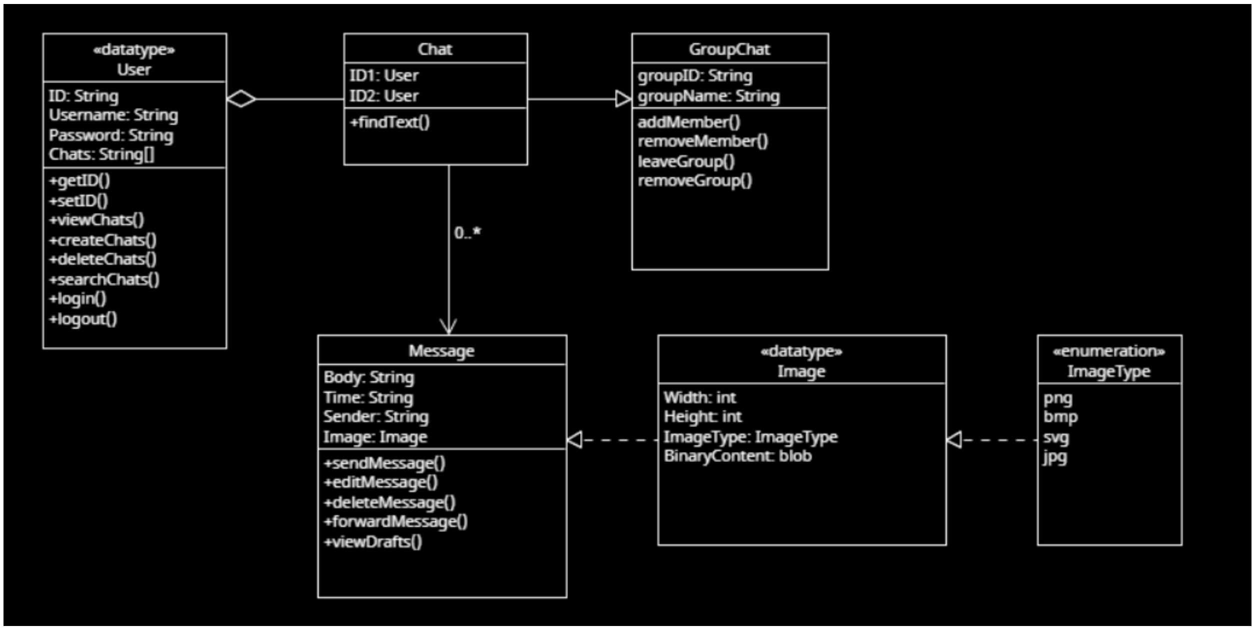
Edit Group chat use case:



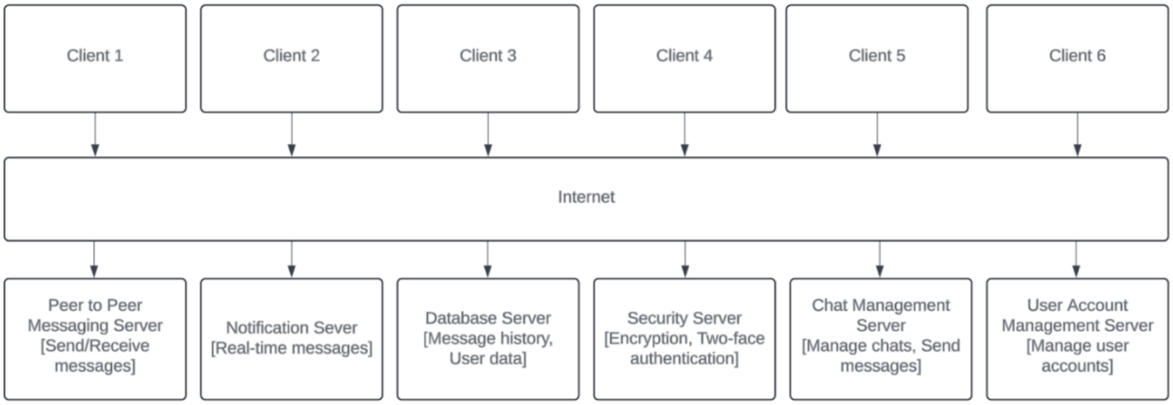
Search use case:



6. Class diagram



7. Client Server Architectural design


Client 1
Client 2
Client 3
Client 4
Client 5
Client 6
Internet
Peer to Peer Messaging Server [Send/Receive messages]
Notification Sever [Real-time messages]
Database Server [Message history, User data]
Security Server [Encryption, Two-face authentication]
Chat Management Server [Manage chats, Send messages]
User Account Management Server [Manage user accounts]

------- Deliverable 1 Content Ends here -------

7. Project Scheduling

Project Duration:

- Start Date: 9/14
- Due Date for Deliverable 2 (Code Demonstration): 11/10
- Full Implementation Date (Complete App): 12/10 (Project extension beyond initial deliverable)

The project is scheduled over 8 weeks, from 9/14 to 11/10, with an additional month for complete app implementation. This schedule includes code demonstration for Deliverable 2, where the basic application functionality will be demonstrated, but not fully developed. The full implementation is scheduled for completion by 12/10.

Working Hours Per Day:

- Working Days: Monday to Friday (Weekends are excluded)
- Working Hours per Day: 2 hours per developer, with a total of 16 hours per week for the team (8 developers x 2 hours/day x 5 days/week).

This is in line with the team's availability and is designed to be minimal yet efficient enough to keep development on track.

Key Milestones and Deliverables:

- Prototype (Early Spiral Cycle): 9/14 - 9/28
  - Core functionalities: P2P messaging, user sign-up, and login.
  - Basic encryption functionality integrated (without full quantum cryptography).
  - Estimated hours: 40 hours (8 developers, 5 days/week).
- Code Demonstration (Deliverable 2): 9/29 - 11/10
  - Demonstration with key features including successful sent and received messages.
  - Estimated hours: 68 hours (team focus on finalizing messaging system, testing, and integration).
- Full Implementation (App Completion): 11/11 - 12/10
  - Full implementation including advanced encryption features (Quantum Key Distribution), comprehensive testing, bug fixing, and user interface development.

    - Final release preparation and complete app validation.
    - Estimated hours: 80 hours (team continues work on encryption, UI, and testing).

Justifications for Estimation:

- The team is composed of 8 developers with varying expertise, which should facilitate quick progress.
- The workload per day (2 hours) is designed to keep the project manageable without overwhelming the team, allowing time for learning and iteration in the spiral model.
- Key tasks are broken into phases to allow for iterative progress, where each phase builds on the previous one.

7.2 Function Point Calculation:

	Category	Count	Complexit y			Count x Complexity
			Simple	Average	Complex	
1	User Inputs	7	3	4	6	7*3
2	User Outputs	2	4	5	7	2*4
3	User Queries	3	3	4	6	3*3
4	Data / Relational	3	7	10	15	3*7
5	External Interfaces	4	5	7	10	4*5

GFP = 79

Processing Complexity Adjustment:

PCA = .65 + .01(4 + 5 + 0 + 1 + 0 + 4 + 0 + 1 + 1 + 0 + 0 + 1 + 0 + 3) = .85

PCA * GFP = FP

.85 * 79 = 67.15

(Function Point Sum) 67.15 ≈ 68 (Analogous Estimate)

I promise I didn't modify the values, the near complete match between FP Sum, and Analogous Estimation was extremely surprising to me too. I will round up to 68 for future calculations.

Scheduling Estimation Sanity Check via Analogous Estimation:

- Core Server/Client Infrastructure - 3 Dev Hours (External Interface)
- Account Creation System - 2 Dev Hours (User Input)
- Account Logs, and Password Salting - 3 Dev Hours (Data Files)
- Login/Logout & Main Menu - 2 Dev Hours (User Query)
- Chat With Text Logs - 4 Dev Hours (User Input / User Query)
- Multiuser Chats - 2 Dev Hours (External Interface)
- Add & Remove Users on Multiuser Chats - 2 Dev Hours (User Input)
- Chat Reply System - 2 Dev Hours (User Input)
- Image Texting & Logs - 5 Dev Hours (User Input / Data Files)
- Message Editing & Deletion - 4 Dev Hours (User Input / User Output)
- Message Forwarding - 1 Dev Hour (User Input / User Output)
- Message Search - 3 Dev Hours (User Query)
- Encryption Implementation - 5 Dev Hours (Data Files)
- Debugging Time - 15 Dev Hours (External Interfaces)
- UI Implementation (or more debugging) - 15 Dev Hours (External Interfaces)
- Total Dev Work - 68 Points
- 1 Function Point is roughly equivalent to 1 dev hour

Function Point Assignment Logic:

Function points were assigned to components via the gross function point method, and then modified for process complexity, and the sanity checked via analogous estimation using personal experience. For the components which I didn’t know how to develop, I deferred to the experience of my team members for a time estimate.

Regarding Integration:

The integration of the diverse functions into a cohesive whole is factored into the function point dev time, however, assuming any issues appear, which can’t be resolved without interrupting function development, we have set aside 15 hours for debugging, and integration, and yet another 15 hours which would be dedicated to UI implementation in the best case scenario would also be dedicated to debugging, so we can ensure we hit our time targets, if only with a CLI interface.

Work Distribution & Costs:

- 8 Person Team
- Each Dedicates 2 Hours Per Week
- Average Software Intern Pay in Texas = $23/hr. [1]
- Approximate Dev Time = (8 * 2 * 4.25 = 68) = 4.25 Weeks
- Approximate Personnel Cost = 68 * 23 = $1564

Hardware Costs:

Server must be active 24/7, and estimated server load is low, but not trivially so due to the need for a fast encryption/decryption process. For text this would not be necessary, however, images are much larger, and they could require up to a minute to encrypt if on an out-of-date device. Given these requirements I have decided to use a dedicated x86 VM instance with four cores, and 16 GB RAM, the cheapest service for such a server is Oracle Cloud, which costs just $54 per month, and provides a free 50 gigabytes of block storage, which won't fill up easily due to the vast majority of stored data being texts. Google, Azure, and AWS are at least twice the cost per month, and all other services aren't as large or reliable. [2]

Software Costs:

For Android development the team will use Android Studio, which is free, and deals with most of the boilerplate code automatically, thus saving time over other development environments. To maintain a consistent development environment across platforms for server-side development, we will use another JetBrains based IDE, IntelliJ, which is not free for commercial use. It costs $16.9 monthly per user, and rounding up to 2 months, and 8 users the total software cost is…

16.9 * 8 * 2 = $270.4 [3]

Total Costs:

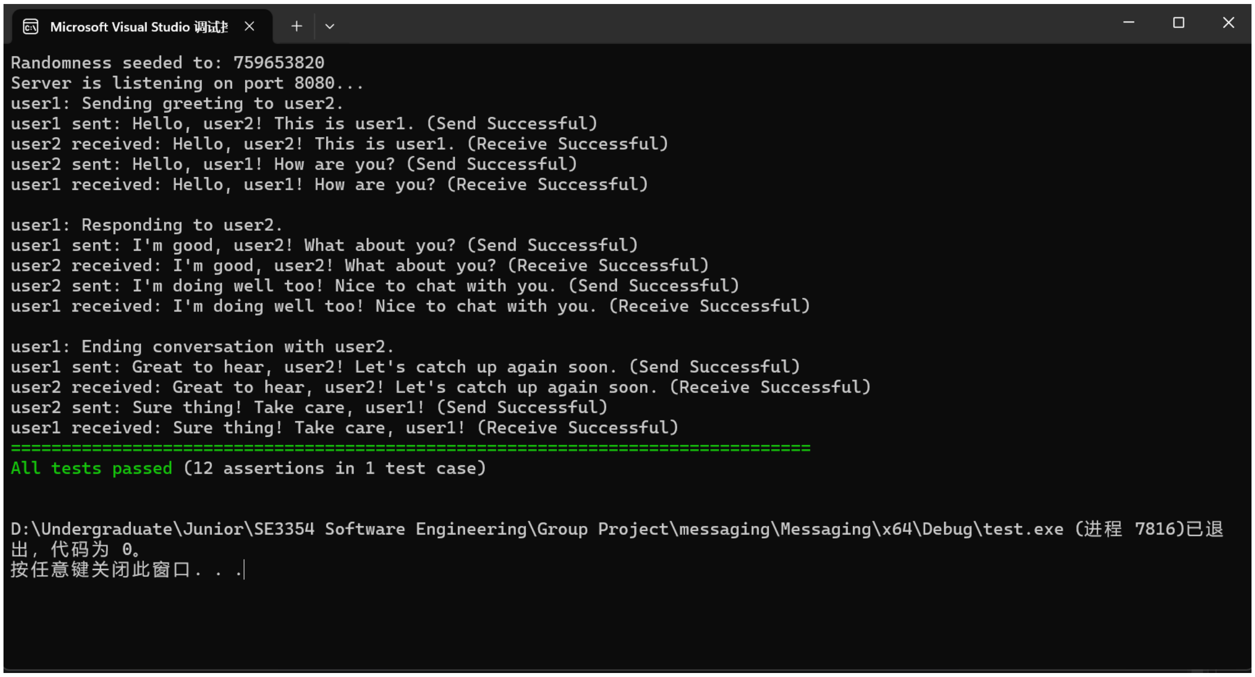
(Server Cost Paid for 2 Development Months)

1564 + 270.4 + 54 * 2 = $1942.4

8. Test Plan

For the test plan of a messenger app, it would only make sense to test the messaging code as it is a crucial component of the entire application. The following code includes both server side and client-side applications with the implementation of TCP (Transmission Control Protocol). TCP is not used for SMS but, it still allows one or more users to connect with one another through the server to communicate. This is why direct messaging applications have been taking over regular SMS as direct messaging utilizes TCP. Anyone with access to the internet can chat freely with the use of direct messaging.

Our test case is an example of a conversation of two users, with the code clearly expressing and simulating how messages are delivered to each user. A total of one test case was executed, with 6 unique messages being verified by the program when it is sent and received.



```
Randomness seeded to: 759653820
Server is listening on port 8080...
user1: Sending greeting to user2.
user1 sent: Hello, user2! This is user1. (Send Successful)
user2 received: Hello, user2! This is user1. (Receive Successful)
user2 sent: Hello, user1! How are you? (Send Successful)
user1 received: Hello, user1! How are you? (Receive Successful)

user1: Responding to user2.
user1 sent: I'm good, user2! What about you? (Send Successful)
user2 received: I'm good, user2! What about you? (Receive Successful)
user2 sent: I'm doing well too! Nice to chat with you. (Send Successful)
user1 received: I'm doing well too! Nice to chat with you. (Receive Successful)

user1: Ending conversation with user2.
user1 sent: Great to hear, user2! Let's catch up again soon. (Send Successful)
user2 received: Great to hear, user2! Let's catch up again soon. (Receive Successful)
user2 sent: Sure thing! Take care, user1! (Send Successful)
user1 received: Sure thing! Take care, user1! (Receive Successful)
===============================================================================
All tests passed (12 assertions in 1 test case)


D:\Undergraduate\Junior\SE3354 Software Engineering\Group Project\messaging\Messaging\x64\Debug\test.exe (进程 7816)已退
出，代码为 0。
按任意键关闭此窗口. . .
```

9. Comparison of your work with similar designs

9.1 Peer-To-Peer Networking and Security: “Sibyl” uses a peer-to-peer (P2P) networking model which is not as common in most SMS applications. Telegram and WhatsApp focus on end-to-end encryption for security and their message delivery depends on centralized servers which handle user connections. [4] “Sibyl” uses P2P with Quantum Key Distribution (QKD), a cutting-edge encryption technique that offers a higher level of security than end-to-end encryption in most SMS apps. [6] Compared to the Signal protocol that most applications use, QKD is designed to be more resilient against future computing attacks making it a more viable long-term solution.

9.2 Storage System: “Sibyl” is going to use a local storage system instead of cloud-based servers. Most SMS competitors like Telegram use cloud-based systems, however, there are tradeoffs for both types of storage systems. [7] Storing locally is great for privacy and security concerns as it keeps the message histories on the user’s device, but this means the user's device storage will be taken up with the messages. [5] Cloud storage stores them online where there is essentially unlimited storage, however, the data is easier to access for a third-party agent. [4]

9.3 Platform Compatibility and Real-Time Notification System: Signal offers cross-platform compatibility which ensures a seamless experience across mobile and desktop environments. [8] Much like this, “Sibyl” aims to deliver a consistent user experience across all devices with responsive design and <2-second average response time benchmark. Furthermore, implementing an advanced real-time notification system would provide a more integrated notification experience. This means that there will be real-time updates on messages, friend requests, and software updates. This is different from our competitors because they handle all these features separately without a system that handles all notification types. In all, the notification system adds to the app’s usability by allowing users to manage communication and updates in one place.

9.4 Privacy and Theme Customization: “Sibyl” aims to offer many options for managing privacy, namely being online status or disabling read receipts. [5] This aligns closely with the goals of Signal, which offers similar privacy controls. Signal has a system of auto-delete timers and managing read receipts which we aim to include. [8] Furthermore, we would like to have a more customizable experience making our app artsy. Allowing users to change the color theme of the app which is less common in mainstream SMS applications.

9.5 Compliance and Data Privacy: SMS applications like WhatsApp and Signal are GDPR-compliant. This ensures data protection and user rights over personal data. [4] “Sibyl” aims also to be GDPR and HIPAA-compliant, using AES-256 encryption for data security, the current industry standard. This will allow “Sibyl” to have legal accountability for prioritizing user privacy.

9.6 Chat Management Controls: One of the main goals of “Sibyl” is to have many options when managing contacts, creating chats, using block/hide features, or limiting communication from specific users. [4] We want to create an application that can allow the user to do anything they can do when it comes to privacy and security. Many other competitors use block features however, “Sibyl” will have much more detailed aspects. [5] The user might want to block someone for a specific amount of time. Sibyl allows that. The user might want to create a group chat with a blocked user. Sibyl allows that. Additionally, group chats in “Sibyl” come with user permissions which are not as accessible in other apps. “Sibyl” will give the user complete user and privacy control.

6. Conclusion

Our project didn’t deviate from our initial plans, which was due in part to our pre-planning and risk assessment that we have outlined within our diagram and models, therefore, there was not much change at all in our work. In terms of our software development, we utilized a spiral model as our software development life cycle model of choice, and function points to estimate how long developing a component of the product would take. With a team of 8 developers working 2 hours a week, using the average software intern pay in Texas, we estimated $1564 as the approximate personnel cost. Our software planning was based on peer-to-peer messaging, whereas shown in our test case, we were able to send 6 unique messages between two users. Overall, our work was a success, with few little roadblocks encountered along the way.

References

[1] “Software Engineer Intern,” *ZipRecruiter*, 2024. https://www.ziprecruiter.com/Salaries/Software-Engineer-Intern-Salary--in-Texas

[2] “Oracle Cloud Pricing and Costs,” *Oracle.com*, 2021. https://www.oracle.com/cloud/pricing/

[3] “Buy IntelliJ IDEA Ultimate: Pricing and Licensing, Discounts - JetBrains Toolbox Subscription,” *JetBrains*. https://www.jetbrains.com/idea/buy/?section=personal&billing=monthly

[4] S. Chekanov, “Top 10 encrypted messaging apps for 2024 - Brosix,” *BROSIX*, Nov. 07, 2024. https://www.brosix.com/blog/encrypted-messaging-apps/

[5] N. Lewis, “Why you should use Peer-to-Peer messaging Apps,” *How-To Geek*, Apr. 10, 2022. [Online]. Available: https://www.howtogeek.com/790612/why-you-should-use-peer-to-peer-messaging-apps/

[6] A. S. Gillis, “Quantum key distribution (QKD),” *Search Security*, Nov. 30, 2022. https://www.techtarget.com/searchsecurity/definition/quantum-key-distribution-QKD

[7] B. Stephenson, “What is the Telegram app?,” *Lifewire*, Apr. 09, 2024. [Online]. Available: https://www.lifewire.com/what-is-telegram-8627901?utm_source=chatgpt.com

[8] J. Centers, “Signal provides secure Cross-Platform replacement for WhatsApp | VMUG,” Jan. 18, 2021. https://vmug.bc.ca/signal-provides-secure-cross-platform-replacement-for-whatsapp/