Concord

A closed chat for enterprise environments where privacy is essential.

Authors:
Yousef Noufal
Gunnar Johansson
Simon Johansson
Jonathan Alm
Alexander Lisborg

Introduction

The modern-day information driven digital environment is filled with actors who use data as a means to get by. Most often by selling data for advertisement or research purposes. It turns out, however, that

the systems built with the intention of collecting data, storing it and later sharing it for a price comes with increased risk for the users. Malicious actors thrive in this environment. Collections of data illegally gathered by malicious actors can be sold for a fortune, which is exactly what happened in a recent leak of discord messages from millions of users (CONSTANTINESCU, 2024 Discord leak). Getting a hold of chat messages of programmers discussing critical bugs in a banking system would be like getting the keys to a vault. Getting the messages of someone sharing their steam password to their friend would be like getting the keys to their locker, not as big of a threat but indeed one of the many possibilities that emerge in large, open communication systems. What if we narrow down the possibilities? What if there were no opportunity for theft in the first place? This is the purpose of our project.

We are a team of five students collaborating to develop a chat application designed for professional programming environments, with additional functionality for personal use. The application is built on a server-client architecture, enabling multiple clients to connect to a centralized server. The server efficiently manages and distributes communications between users. Key features include sending and receiving messages, creating, joining and leaving channels and connecting and disconnecting from a server. A unique feature inspired by user feedback is a notification window that displays updates, such as new messages or changes within channels. There is no shared data between servers, and the fact that this is not a commercially available platform that is targeted by hackers eliminates the risk of being part of large-scale leaks. Our application is not directly linked to other services such as Gmail, Outlook or other third-party applications that could introduce security flaws.

The main challenge of our project will be prioritizing features with our limited resources in mind. The time span of the project is 8 weeks in total, where much of the time is spent planning, documenting and testing. Facing these limitations, we've decided to not aim for a complete, secure and encrypted program as our end product, but to develop the application with the intention of making it easy for future implementation of complete encryption and other complex functionalities. We approach this problem with the mindset that future developers may take over from where we left off and aim to include sufficient documentation and modularity to make that possible. One of the technical challenges we will face during this project is ensuring that a server can handle numerous simultaneous users without performance issues or errors.

Another technical difficulty will be the implementation of a feature that automatically colors code sent in chat. The feature is thought to allow users to easily read code sent to them inside the interface. These functionalities are designed to create a secure, user-friendly, and efficient platform for professional collaboration.

Current state of affairs

All of the currently top-rated applications used for communication inside businesses that we've inspected have had security issues of some sort. Specifically, these include Discord, Slack(Mimecast Slack Data Breaches), Github(Uy, 2023 Github Security Incident) and Gitlab(CONSTANTINESCU, 2024 GitLab vulnerability). It seems that in spite of companies making the greatest effort to secure their programs, security incidents do happen. Which is only natural in large scale complex applications with millions of users where there is a clear incentive for hackers to find exploits. If you are a smaller group or company looking for programs for communication, the risks of being affected by leaks or other exploits are always there.

With our application, however, there is no risk of being affected by a leak of some global database, there would have to be a targeted attack, specifically targeting your company or your connected computers, which in the case of a smaller company or group, the risk of that would probably be quite small. (Depending on what kind of business you're running) Our product is not made to be commercially spread or used by many. We aim to be a simple alternative to using targeted commercial communication platforms. With further development of this application, our product has the potential to become a highly secure messaging platform for use in enterprise environments.

Mockup

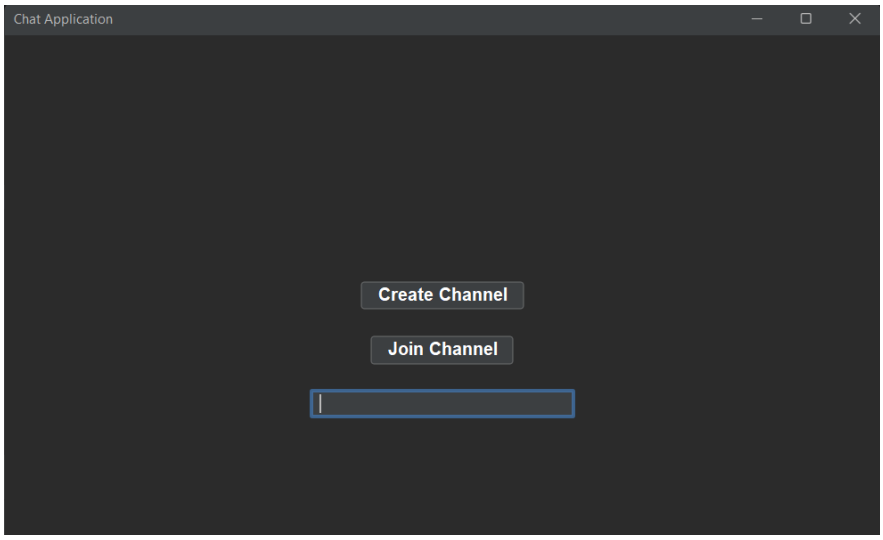


Fig. 1. This window allows users to create a unique username and then either join an existing channel or create a new one to get started. It serves as the first step in establishing a connected communication experience.

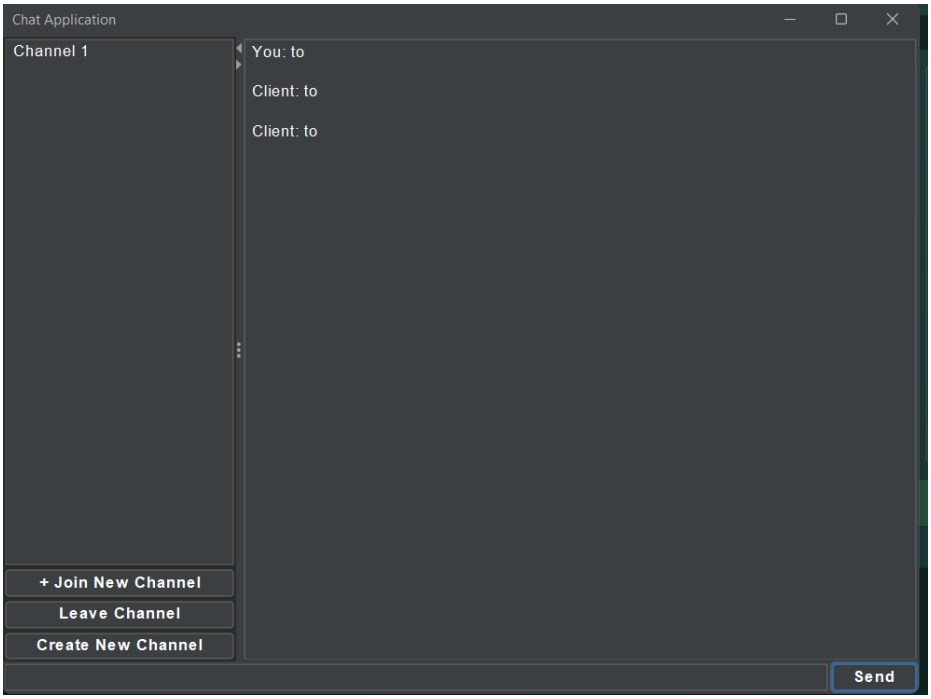


Fig. 2. This window interface will allow the client to seamlessly view and interact with its features. Functionalities include joining or leaving channels and switching between them. In addition to the ability to leave the channel.

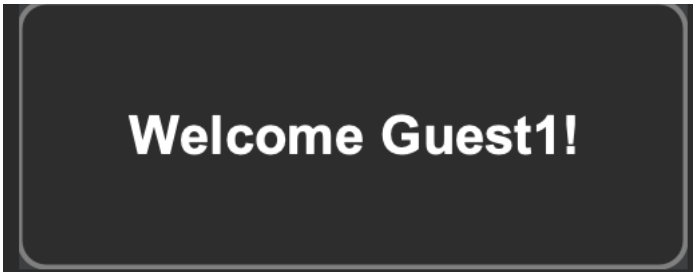


Fig 3. Notification. This window will be shown for the user to be able to show notification to the user like joining/leaving.

The functional requirements of our application can be divided into must haves (representing our minimal viable product), should haves and could haves as listed below:

Must haves:
Client application:
- Functioning user interface.
- Connect & disconnect from server.
- Join & leave channel.
- Send & receive messages in a channel.

Server application:
- Start & stop correctly
- Communicate with clients.
- Handle multiple users.

Should haves:
Multiple channel support. (Users being able to join and chat in multiple channels)
Password protected channels.
User login functionality. (Having an account that you can login to via credentials)
Multiple server support. (Users being able to join and leave different servers)
Encrypted messages.
Easy to read code in chat. (Color highlighted code for one programming language)
Secure handling of server data. (Storing of data in a secure, encrypted manner)

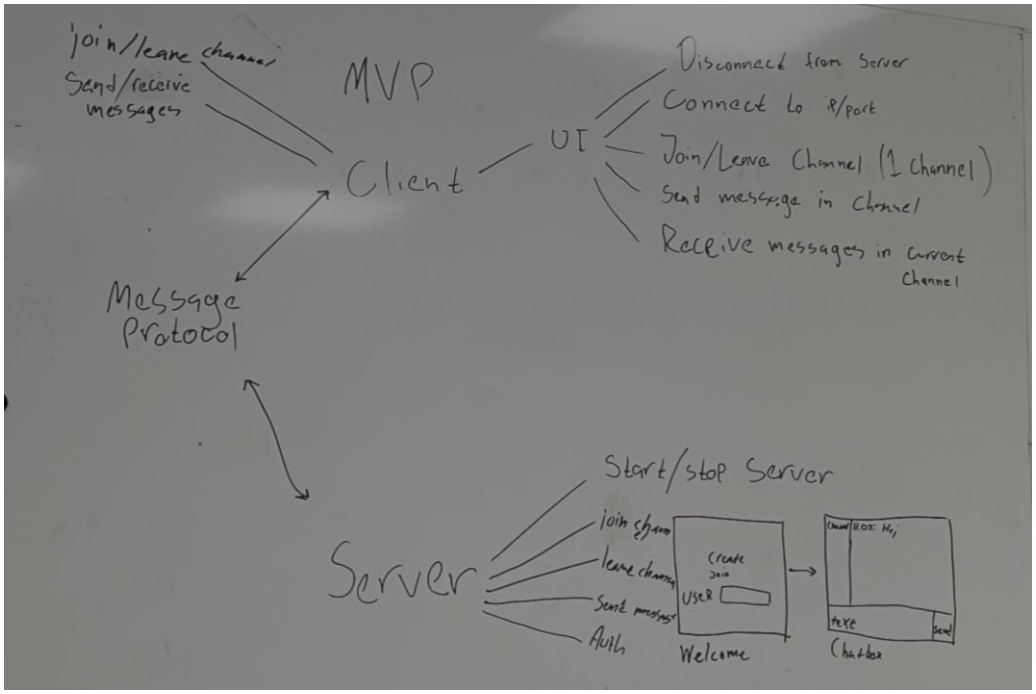
Could haves:
User profiles.
Easy to read code support for multiple programming languages.
Secure login authentication method.

User stories

The value of the functionality lies within its useability. We envision our end users to be workers in IT enterprises and the technically curious who want private communication. To act as representatives of our end users, we have chosen several people in computer science education on the University of Gothenburg campus and conducted interviews with them to figure out their needs and gain their perspective on our application. These interviews have served as direct inspiration in the definition of our project and have helped us in the construction of user stories. These user stories are made up stories told from the perspective of our end users, having a need for certain functionality in order to use the program in a practical scenario. Ideally, these user stories would be discussed with our stakeholders to reach conclusions about how we should define acceptance criteria and how we should approach a solution, but since we don’t have those, we’ve simulated discussions.

Requirements Analysis Document

Epics:
MVP



As a user, I want to open a client application, connect to a server, and create or join a channel to facilitate communication.

Acceptance Criteria:

- The client can successfully connect to the server.
- The user can join a channel.
- The user can leave a channel.
- The user can send and receive messages within a channel.

User Stories 1, Create channel, let users join - Server side:

As a programmer, I want to be able to create and join channels.

Discussion:
The server should handle requests to create and join channels.
If the logic validates successfully, a new channel will be created, and the requesting user will be added as the first member.
If other users are connected to the server, they should be able to join existing channels.

Acceptance Criteria:

- The logic for creating, joining, and leaving channels must be correct.
- Each channel should have a unique identifier, and duplicate channels should not be allowed.
- If a channel already exists, other users should only be able to join it, not create it again.

User Stories 2, Create a client:

As a programmer, I want to create a client application to interact with the server so that users can connect to channels and communicate.

Discussion
The client should be able to send requests to the server for creating, joining, and leaving channels.
It should provide a user-friendly interface for interacting with the server's features.

Acceptance Criteria

- The client can connect to the server and establish a stable connection.
- Users can issue requests
- Server response are displayed clearly to the user.

Definition of Done

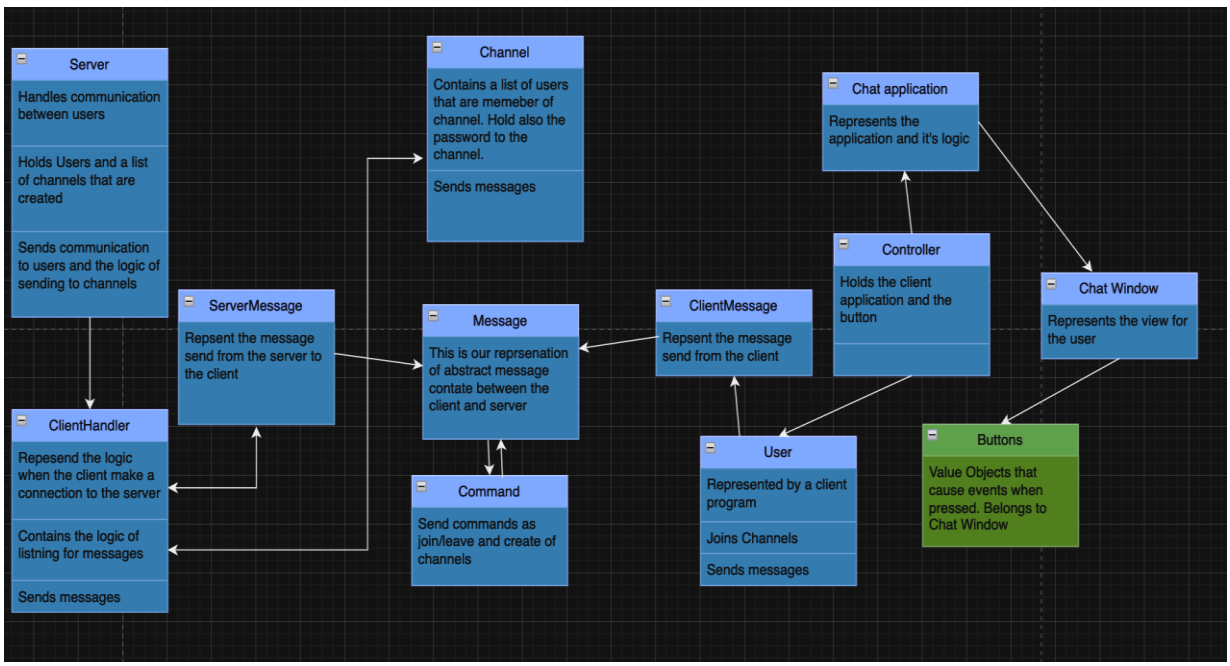
Our process begins by defining high-level epics on the Kanban board, which are then broken down into user stories and further divided into smaller, actionable tasks. These tasks are assigned to team members for implementation.

Feature development occurs on dedicated branches in our version control system, incorporating JUnit testing, collaborative peer reviews, and adherence to the coding standards outlined in our group contract established at project inception. This ensures code quality, alignment with the intended design, and comprehensive testing of all program components where applicable.

Stakeholder feedback is gathered through tools like Discord or in-person meetings. For UX/UI development, which is less directly testable, weekly user interviews and internal testing play a critical role in refining the design and functionality.

Once a feature is completed, it is moved to the "Under Review" column on the Kanban board. Depending on the task's complexity and prior arrangements, it is either thoroughly reviewed or moved to "Done" after confirmation. Finally, the feature is merged into the main branch, completing the development cycle.

Domain model



User

A User represents an individual interacting with the system. Each user has a unique identifier, username, and profile information. Users can join or leave channels, send messages, and receive notifications.

UserHandler
A UserHandler represents a user on the Server end of the connection.

Server
A Server represent the connection manger between the user connected and the channels available. Holding the correct information about the members and then send the correct message or command to the user.

Channel
A Channel is a virtual space where users can engage in conversations. Channels can be public or private, and users must join a channel to participate in its discussions but currently the user needs to know the channel name. Each channel maintains a list of members.

Message
A Message represents the communication exchanged within a channel. Each message is associated with a sender, timestamp, and content.

Notification
Notifications alert users to new messages or changes within their channels, ensuring they stay informed even when switching between channels.

Glossary

Server: "*A server is a hardware device or software* that processes requests sent over a network and replies to them." - https://www.geeksforgeeks.org/what-is-server/

Encryption: "*Data Encryption is a method of preserving data confidentiality by transforming it into ciphertext, which can only be decoded using a unique decryption key produced at the time of the encryption or before it. The conversion of plaintext into ciphertext is known as encryption.*"
- https://www.geeksforgeeks.org/what-is-data-encryption/

Data leak: "*A data leak is when information is exposed to unauthorized people due to internal errors. This is often caused by poor data security and sanitization, outdated systems, or a lack of employee training. Data leaks could lead to identity theft, data breaches, or ransomware installation.*"
- https://abnormalsecurity.com/glossary/data-leak

References

5 Slack Data Breaches you should know about. Mimecast. (n.d.). https://www.mimecast.com/blog/slack-data-breaches/

CONSTANTINESCU, V. (2024a, April 29). *Discord takes down "Spy.pet" website that harvested data from hundreds of millions of users*. Hot for Security. https://www.bitdefender.com/en-

us/blog/hotforsecurity/discord-takes-down-spy-pet-website-that-harvested-data-from-hundreds-of-millions-of-users

CONSTANTINESCU, V. (2024b, September 13). *Gitlab issues emergency patch for pipeline exploitation vulnerability*. Hot for Security. https://www.bitdefender.com/en-us/blog/hotforsecurity/gitlab-issues-emergency-patch-for-pipeline-exploitation-vulnerability

Uy, P. (2023, April 28). *Understanding the github security incident: Implications and preventive measures*. IPV Network. https://ipvnetwork.com/understanding-the-github-security-incident-implications-and-preventive-measures/