PingMe

Izabela Camaj, Christina Carvalho, Malik Issani

izabelacamaj@oakland.edu, ccarvalho@oakland.edu, mnissani@oakland.edu

Computer Science and Engineering Department

School of Engineering and Computer Science

CSI 2470: Intro to Computer Networks

Prof. Dafer Alali

April 6, 2025

Abstract

PingMe is an application that allows users to chat with any concurrent registered users. PingMe prides itself in providing effective security for all users. Many applications that offer chat features have been known to leak user data, but PingMe has considered many factors to best protect its users while allowing connections between multiple users within the same server. PingMe is a well constructed client-server application that enables a simple chat feature with the use of Transmission Control Protocol (TCP). Users can connect to the server by using an IP address and specific port, send messages and data to the server, and receive response or data from the server accordingly. The server listens on a specific port, accepts multiple client connections, and most importantly processes the client's requests and responses.

Introduction

My team was tasked with creating a client-server application that allows communication over a network. We initially selected the Authentication Server and promised to provide unique services, such as a user registration and login system, a password recovery mechanism, and token-based authentication for secure access. While security remains a priority, our project guarantees secure storage of user information in a SQLite database, processing of authentication requests, and issuing of session tokens. So, registered users can safely access the PingMe chat feature.

The goal of this project is to provide an authentication system that allows user communication. In order to deliver the best product, or application, that meets these requirements, our team applied a lot of focus on our approach to the product development process.

Development Methodology

Approach: SCRUM Agile

Our group utilized SCRUM, an agile methodology, to manage the development of this project. SCRUM maintains different development phases, but we have modified them due to given time and goals that had to be met. SCRUM is a value driven planning tool and our team predicted that we would face some challenges that would require changes to our plan as we embraced this learning journey. Scrum allows modification based on needs (Hron et al., 2022). Our team was determined to deliver the best value that met all requirements, so our phases consisted of Requirements, Design, Implementation, and Testing. Each phase consisted of 13 days, so each member maintained a SCRUM leader position for 13 days in rotation until the testing phase. During the testing phase we simply collaborated to resolve any issues.

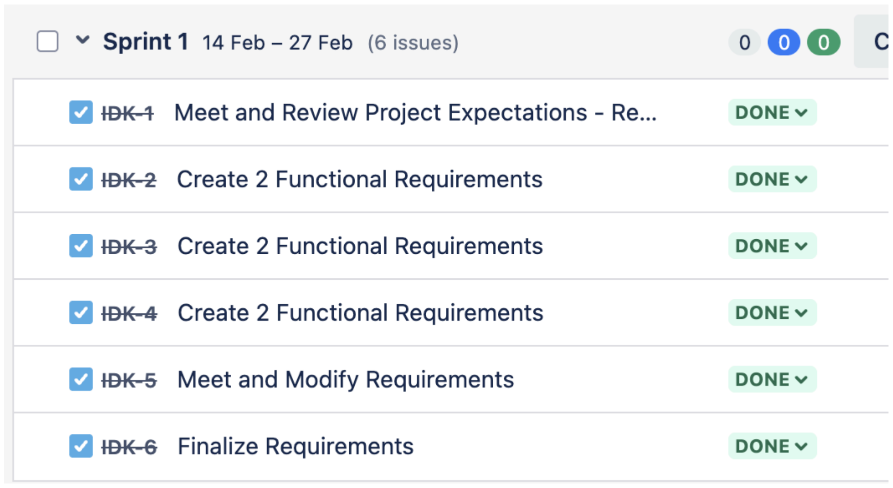
Our team met once a week and maintained communication over messages and Discord (Discord, 2025). The assigned SCRUM leader would be responsible for planning the day and time of these meetings, assigning tasks, and tracking progress. Specifically, the completion of tasks was tracked by Jira (Atlassian, 2025). Jira is a project management tool that is mainly used by software development teams. Each SCRUM leader could create tasks and assign them to different members.

Sprint Breakdown

Sprint 1: Requirement phase. Our group initially met to review the project requirements. After this process, the SCRUM Leader, Izabela Camaj, assigned each member with the task of writing 2 functional requirements. Our team would continue to communicate over Discord and meet once a week to ensure the requirements were modified and finalized by the end of the sprint. All completed tasks can be seen on Figure 1.

Figure 1

Sprint 1: Jira Sprint Tasks


Sprint 1 14 Feb – 27 Feb (6 issues) 0 0 0

~~IDK-1~~	Meet and Review Project Expectations - Re...	DONE
~~IDK-2~~	Create 2 Functional Requirements	DONE
~~IDK-3~~	Create 2 Functional Requirements	DONE
~~IDK-4~~	Create 2 Functional Requirements	DONE
~~IDK-5~~	Meet and Modify Requirements	DONE
~~IDK-6~~	Finalize Requirements	DONE

Note. This specific image captures tasks assigned to each team member. These tasks were completed within the 13 day long phase dedicated to crafting requirements for our product.

Software Requirement Specification

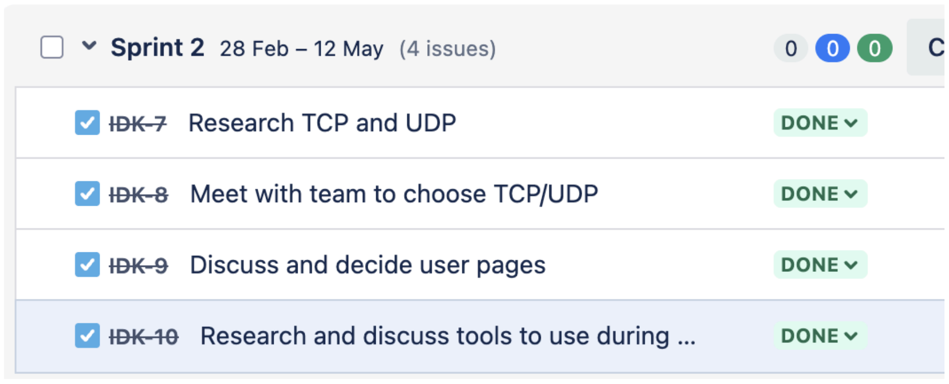
1. The app shall allow new users to register a new account.
   1.1. The user shall create a unique username and password.
   1.2. The user shall answer two security questions.
   1.3. The user shall be given a 2 Factor Authentication code.
   1.4. The app shall store user information in a secure database.
2. The app shall authenticate users by verifying their username, password, and 2FA code before allowing the user to have access to the application.
3. The app shall issue a unique 2 Factor Authentication code to users upon successful login.
   3.1. The user shall utilize this same 2FA code when logging into the app.
4. The app shall allow users to recover their account password.
   4.1. The user shall share their username and answer 2 security questions in order to recover their 2FA code and password.
5. The app shall issue a session token after each successful login.
6. The app shall allow users to send chat messages to other concurrent users.
   6.1. The app shall only allow logged in users to send chat messages.

Sprint 2: Design phase. During the Design Phase, our team focused on creating a client-server architecture to ensure efficient communication. The SCRUM leader, Christina Carvalho, instructed each team member to perform research on UDP and TCP. So, in our first meeting we discussed the benefits of using each protocol. After this discussion, our team planned a TCP communication protocol to facilitate reliable data exchange between the client and server, while using sockets to handle the network interactions.

Throughout this phase, we discussed the user interaction flow to best plan the user experience. Our application would prompt a main welcome page, which gives the user the option to register and log in. The register page would prompt the user to create their username, password, and answer two security questions. The log in page would ask the user to input their registered information, such as username, password, and 2FA code. After successfully logging in, the user can see a chat page where they can send messages and communicate with other users who are logged in the same server. Figure 2 represents all tasks that team members completed during this sprint to produce the best design.

Figure 2

Sprint 2: Jira Sprint Tasks



~~IDK-7~~	Research TCP and UDP	DONE
~~IDK-8~~	Meet with team to choose TCP/UDP	DONE
~~IDK-9~~	Discuss and decide user pages	DONE
~~IDK-10~~	Research and discuss tools to use during ...	DONE

Note. This specific image captures tasks, dedicated to designing the architecture of our product, as assigned to team members. These tasks were completed within the 13 day long phase.

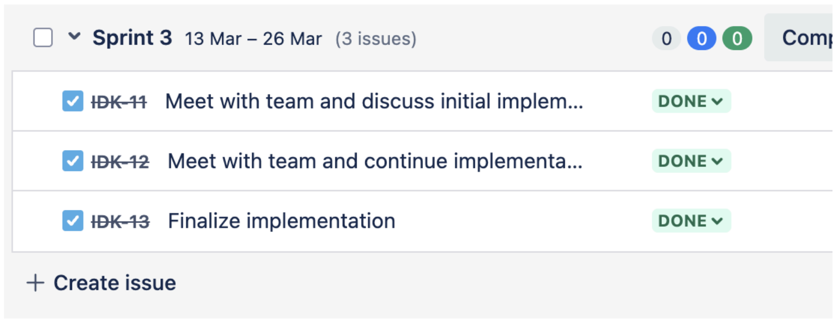
Sprint 3: Implementation phase. During the implementation phase, our team created the server and client files. We considered necessary requirements, such as utilizing the correct programming language (Python), networking protocol (TCP), implementing key features, efficient concurrency, error handling, and security. The SCRUM leader, Malik Issani, assigned tasks during each meeting and in communication over Discord. Figure 3 displays the sprint tasks dedicated to meeting product requirements. Specific tasks had to be altered over the course of this sprint, so they were assigned in communication.

Our team designed a client-server architecture to ensure secure and efficient communication between users and the centralized authentication server. The server manages user authentication, data storage, and request handling, while the client interacts with the server to complete registration, login, and retrieve data. The server specifically integrates with a SQLite database in order to store user information. Communication between the client and server is established by utilizing a TCP-based protocol over sockets. This provides an efficient and reliable connection for data exchanges. Our team prioritized security in this design through token-based authentication and password hashing.

Our team integrated a token system for secure authentication and session management. The user will not see this token, but it can be seen in the terminal for security measures. We implemented password hashing to protect user credentials while registering or logging in. There is also a recovery feature to allow users to recover their passwords and 2FA codes, which are necessary to log in.

Figure 3

Sprint 3: Jira Sprint Tasks



Sprint 3 13 Mar – 26 Mar (3 issues) 0 0 0 Comp

~~IDK-11~~	Meet with team and discuss initial implem...	DONE
~~IDK-12~~	Meet with team and continue implementa...	DONE
~~IDK-13~~	Finalize implementation	DONE

+ Create issue

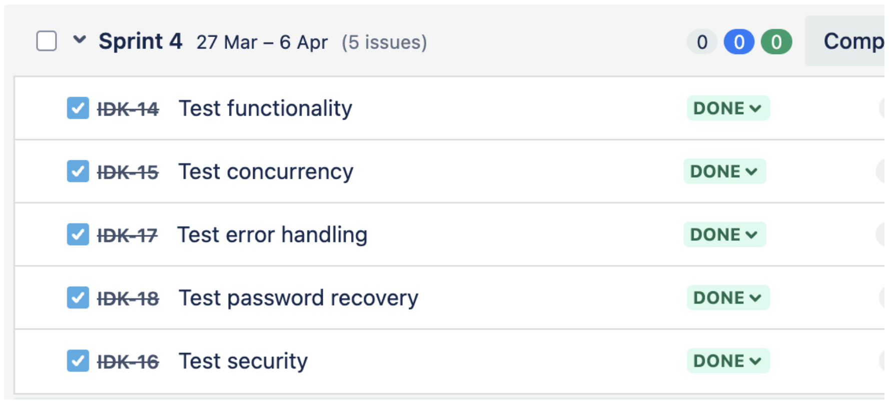
Note. This image displays general tasks over the course of Sprint 3. Tasks assigned over communication were completed within the 13 day long phase, dedicated to creating the source code for our product.

Sprint 4: Testing phase. During the testing phase, our team ran server and client files multiple times to test the features of our app. After confirming the functionality of our features, we tested whether the server could handle concurrent users without a decrease in performance. After confirming this as well, our team ensured the functionality of invalid inputs.

Furthermore, we checked the security features, including the token system, password hashing, and recovery feature. While meeting these requirements, we also maintained a user friendly experience to enhance the user experience.

Figure 4

Sprint 4: Jira Sprint Tasks



Sprint 4 27 Mar – 6 Apr (5 issues) 0 0 0 Comp

~~IDK-14~~	Test functionality	DONE
~~IDK-15~~	Test concurrency	DONE
~~IDK-17~~	Test error handling	DONE
~~IDK-18~~	Test password recovery	DONE
~~IDK-16~~	Test security	DONE

Note. This picture represents different testing tasks that each member completed during the testing phase. These tasks were completed within the 13 day long phase to best test out required features and functionality of PingMe.

Communication Protocol

As our group was tasked with building an authentication server, we chose to use TCP (Transmission Control Protocol) as our networking protocol. TCP guarantees the delivery of packets by checking for any data loss, duplication, or reorganization. Therefore, we can be assured that the server reliably receives any credentials the client sends to it. Additionally, TLS (Transport Layer Security) runs on top of TCP to provide secure encryption methods. TCP was also necessary to handle multiple client connections, as multiple users would need to log in simultaneously to chat with each other. This feature was implemented using threading.

Features & Implementation

Server Build and Client Connection

Python was chosen as our language since our group was familiar with it and its versatility to support sockets, hashing, databases, JSON, and threading libraries (Campbell, 2024). We chose to use SQLite as our database for storing and retrieving user information, as it's relatively easy to implement, it's reliable, and it stores the database in one file (Yegulalp, 2024). To build the client application, we wanted to use a Python app layout library called Tkinter. This library allows us to cleanly customize the format of each screen and interact with user input (Amos, 2024).

The server first creates a socket for IPV4 over a TCP connection. Additionally, the server makes sure that the socket can be reused after terminating the connection. Then, the server binds the port to a specified host IP address and port number. If the bind is unsuccessful due to the port being in use, then it attempts to bind it to the next available port within a certain number of attempts. If an available port is not found or if a different type of error occurs, the server closes.

If the bind is successful, then the server listens for requests from clients sent to that IP and port number (Tripathi, 2023).

When a client connects, the server stores the client's connection to send requests back to the client. Additionally, a new thread is made for each client that connects. This separates all client connections, allowing them to be handled concurrently (Brownlee, 2023). Client sessions are validated using tokens from the secrets library. The server generates a 32-character hexadecimal string that serves as a unique session identifier. It is stored in the database and a dictionary that keeps track of all client sessions. The client also stores the session token so that it is included in the request messages it sends to the server. A new token is generated each time any user logs in (GeeksForGeeks, 2018a).

Overview of Client Application

When a client opens the app, they are greeted with the home page. If the user hasn't made an account yet, they click the register button. On the register page, the user can create a username and password and provide answers for two security questions: "What is your pet's name?" and "What is your favorite color?" When the user clicks the register button, it stores the information in a database and provides a 2-factor authentication code. When the user goes to log in, they need to provide their username, password, and the provided 2-factor code. A successful login will direct the user to the main chat page, where they can send chat messages that other concurrent users can see. If the user forgets their password when they try to log in, they click the forgot password button. After the user shares their username, they need to answer their two security questions to retrieve their password and 2-factor code. On the main chat page, users can type out messages and send them for any other concurrently logged-in users to view.

Registering. When a user enters their username, password, and security questions answers, the code first checks that the fields aren't empty. Then, a JSON file is built with this information and sent to the server over the established socket connection (Python Software Foundation, n.d.). When the server receives the information, it places it into a buffer until the complete message is received. The message is extracted, and if there's any missing information, the server sends an error. The server connects to the SQLite database to check if the username exists and sends back an error if so (SQLite, 2025). The password is then hashed using the SHA-256 encryption method (GeeksForGeeks, 2018b). A unique 6-digit 2-factor authentication code is generated as an extra security measure to prove that its actually the user trying to sign in. The information is then stored in the database (SQLite, 2025). The server sends a success message back to the client with the generated 2-factor code.

Logging In. Similar to registering, when a user enters their username, password, and 2-factor code, the code first checks if there are any empty fields. It builds a JSON request with the information, sends it to the server, and places it into a buffer (Python Software Foundation, n.d.). Then, it extracts the message, connects to the database, and queries the database to check if the information matches (SQLite, 2025). If all information matches, then the user is given a generated session token using the "secrets" Python library, and it is stored in the database. The new connection is stored in order to track current connections. The server sends a success response to the client that the user has successfully logged in. The session token is stored and the user is redirected to the main chat page. Now, whenever the client sends a request, the session token is appended to the message to verify a real connection (GeeksForGeeks, 2018a).

Password Recovery. When the user requests that they forgot their password, they need to submit their username. Similar to the register and login process, the server queries the database to check if the username is valid. If a valid username is provided, the user is redirected to another page to enter the answers to their secret questions. Yet again, the user enters their information, the server queries the database, and returns a success response if the information matches (SQLite, 2025). The success response includes the user's password and 2-factor authentication code needed for logging in.

Chat Functionality. After a user successfully logs in, they are directed to the main chat page. When the user submits a message, the client side builds a JSON object and sends it over the TCP socket, including the user's current session token (Python Software Foundation, n.d.). If the user is somehow disconnected from the server, it will try to reestablish the connection to send the message. On the server side, it extracts the session token from the request and verifies it by returning the corresponding username from the database. All current user sessions are stored in a dictionary. The server creates another JSON object that contains the username of the sender and their message. This object is then sent to all users who have an active connection. When each client receives this request, it is extracted, and the username and message are shown in the main chat area (Python Software Foundation, n.d.).

Logging Out. If the user wants to disconnect, they click the logout button. On the server side, their session token is deleted from the database, and their current session is deleted from the user session dictionary (SQLite, 2025). The server sends a success message that the user has been logged out. On the client side, their session token is deleted and the user is redirected to the home page.

Security Measures. There are several security measures this application implements. When binding the host IP and port number, the server ensures that it binds to an unused port (Tripathi, 2023). If there are any other errors when starting the server, it terminates. When users login and register, each request checks if all fields are provided. Then, if the fields do not match what is provided in the database, the action will not be performed, such as a wrong password or invalid session token (SQLite, 2025). Additionally, when users register, no two users can have the same username. Passwords are hashed using SHA-256 to make it difficult for hackers to recover the passwords (GeeksForGeeks, 2018b). Session tokens are generated to ensure that the requests are only processed by authenticated users . When a user logs out, the session is terminated to prevent token exploitation (GeeksForGeeks, 2018a). A 2-factor authentication code is provided when first registering to add an additional layer of security when signing in.

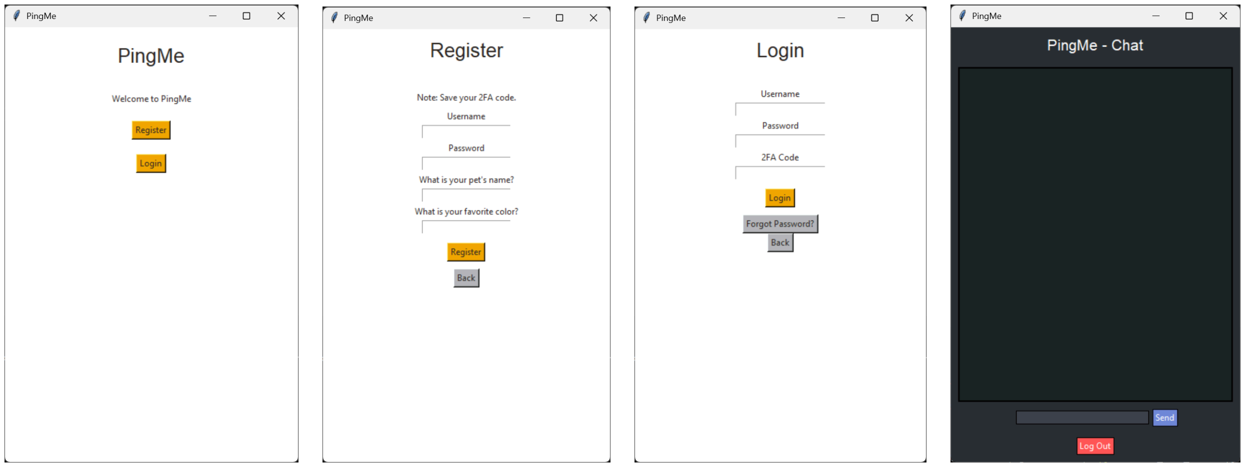
Challenges and Solutions

Transition from Web Interface to Tkinter Desktop Application

The project initially aimed to provide a web-based interface. However, based on user feedback, we transitioned to a desktop application using Tkinter. This shift presented challenges, including adapting the user interface design to a desktop environment and ensuring that the application could handle multiple clients effectively (Amos, 2024). Despite these challenges, Tkinter proved to be a suitable choice for building responsive and user-friendly desktop applications, offering a straightforward approach to GUI development in Python, as seen on Figure 5.

Figure 5

All Featured Pages



Note. First image represents the PingMe main page. According to preference, the user can be guided to the registration page or login page. After registering and logging in, the user can access the chat page.

Race Conditions and Multi-Client Handling

Developing a multi-client chat application using Tkinter presented challenges related to race conditions and managing multiple client connections. Race conditions occur when multiple threads access and modify shared resources simultaneously, leading to inconsistent or unexpected behavior (Kumar, 2025). In our application, this was particularly evident when multiple clients interacted concurrently, potentially causing data inconsistencies. To target these issues, we employed thread synchronization mechanisms such as locks to ensure that only one thread could access shared resources at a time, thereby preserving data integrity and providing a stable user experience.

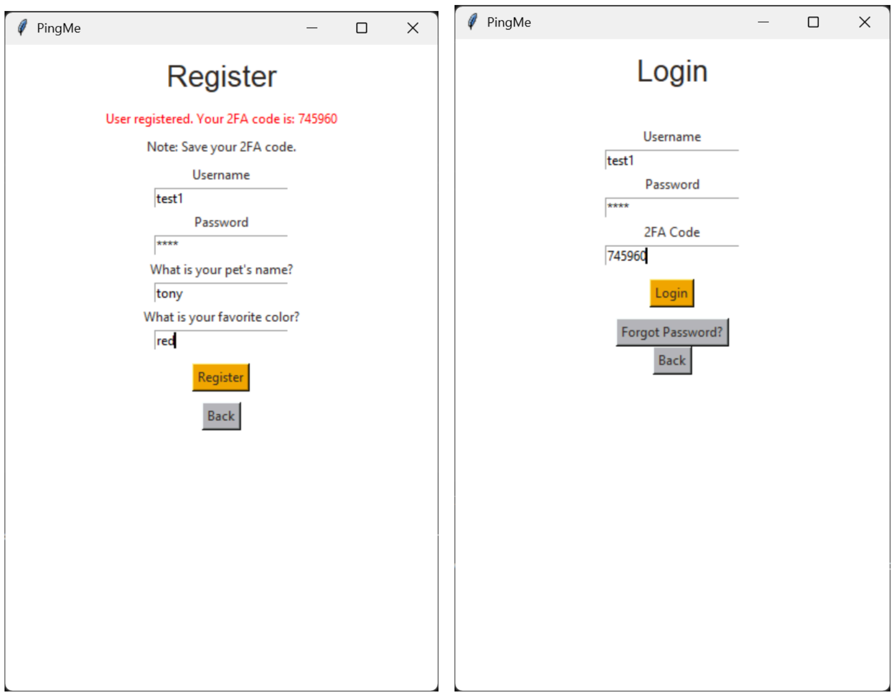
2-Factor Authentication Implementation

Initially, we planned to incorporate two-factor authentication (2FA) using email or text messages to enhance security. However, it presented various internal and external challenges. Implementing 2FA using email or SMS proved challenging due to the need for integrating with third-party services like Twilio or SMTP servers, which introduced technical complexity (Hoffman, 2017). These services also require managing API keys, handling rate limits, and ensuring secure communication, adding to the overall difficulty. Furthermore, potential issues like message delivery delays, spam filters, and network congestion posed reliability concerns, making the implementation more difficult and prone to external service-related challenges.

Nonetheless, feedback and further research indicated a preference for a different approach. In response, we adapted our strategy by implementing 2FA through security questions, such as "What is your pet's name?" and "What is your favorite color?" as seen on Figure 6. This change simplified the user experience while maintaining a reasonable level of security, aligning with user preferences and feedback.

Figure 6

PingMe Registration Page and Login Page Displaying the 2FA Approach



PingMe

Register

User registered. Your 2FA code is: 745960

Note: Save your 2FA code.

Username: test1

Password: ****

What is your pet's name?: tony

What is your favorite color?: red

PingMe

Login

Username: test1

Password: ****

2FA Code: 745960

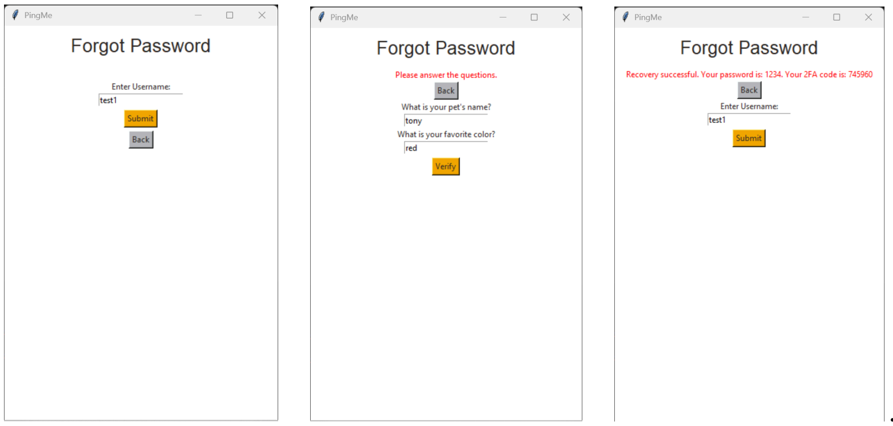
Note. The functionality of the 2FA feature appears on the registration and login page.

Password Recovery Mechanism

The password recovery mechanism in the PingMe app utilizes security questions to ensure a secure and efficient recovery process. Users are required to answer pre-set questions, such as "What is your pet's name?" and "What is your favorite color?" in order to verify their identity. Upon successful validation of the answers, users can reset their password and regain access to their account, as seen on Figure 7. This mechanism is designed to provide an extra layer of security while maintaining a straightforward recovery process for users who may have forgotten their credentials.

Figure 7

PingMe's Password Recovery Mechanism



Enter Username: test1

Please answer the questions.

What is your pet's name? tony

What is your favorite color? red

Recovery successful. Your password is: 1234. Your 2FA code is: 745960

Enter Username: test1

Note. The functionality of the password recovery mechanism appears on the Forgot password prompted pages.

Testing and Results

Manual Testing and Simulated Clients

To evaluate the functionality and reliability of our application, we conducted extensive manual testing. This involved simulating multiple clients to assess the server's ability to handle concurrent connections and message broadcasting. We verified that messages were correctly transmitted between clients and that the user interface responded appropriately to various interactions. This testing approach helped identify and resolve issues related to client-server communication and user experience.

Successful Connections

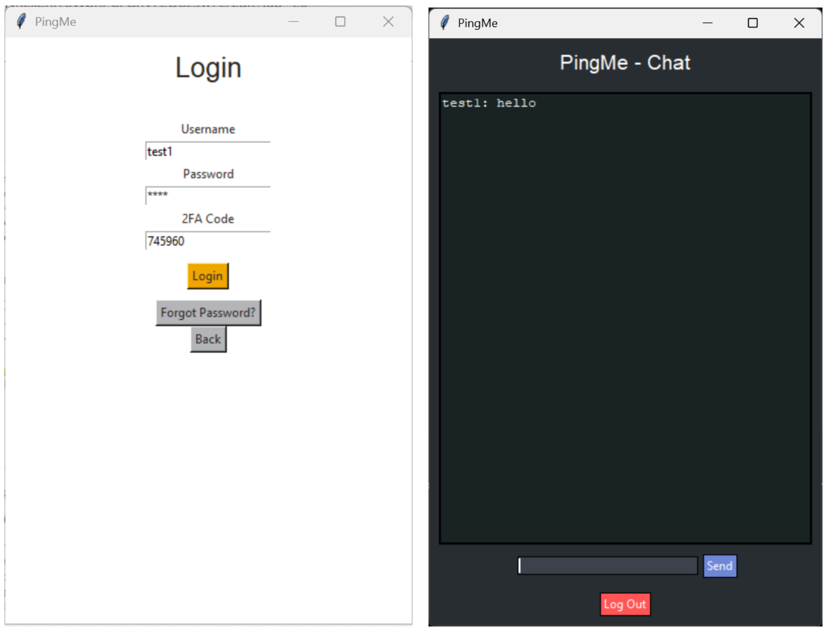
Our testing confirmed that the application could establish and maintain stable connections between clients and the server. Clients were able to log in, participate in chat sessions, and receive real-time updates without significant delays or errors, as seen on Figure 8. This demonstrated the effectiveness of our client-server architecture and the robustness of the implemented features.

Stress Test Outcomes

Under testing conditions, where multiple clients connected simultaneously, the application maintained functionality, though some performance degradation was observed as the number of clients increased. This indicated that while the application could handle multiple clients, optimizations were necessary to improve scalability and performance under high load conditions (Eyalvr12, 2024). These insights guided further development efforts, focusing on enhancing the application's ability to manage a larger number of concurrent users efficiently.

Figure 8

Successful Login and Chat Testing



Username
test1
Password
****
2FA Code
745960

```
test1: hello
```

Conclusion

Recap of Project Goals and Outcomes

The primary goal of our project was to design and implement a secure, multi-client chat application, PingMe, that offers real-time communication while incorporating essential authentication and recovery mechanisms. We aimed to build a user-friendly application with core functionalities such as registration, login, two-factor authentication, password recovery, and seamless client-server messaging. Although the original plan involved creating a web-based platform, user feedback led us to pivot toward a desktop solution using Tkinter. Ultimately, the application successfully supported concurrent users, demonstrated stable performance, and implemented user verification and security.

Ideas for Future Improvements

While PingMe achieved its foundational goals, several enhancements could further elevate the application. Implementing OAuth or third-party login options (e.g., Google or GitHub) would improve security and streamline user authentication. Adding a frontend framework, such as a web interface using Flask or React, would expand accessibility and allow cross-platform usage. We also envision integrating better logging mechanisms for activity monitoring, debugging, and analytics. Finally, adopting secure protocols like SSL/TLS for encrypted messaging and incorporating a notification system would make PingMe more scalable for real-world deployment.

References

Amos, D. (2024, December 7). *Python GUI programming with Tkinter*. Real Python. https://realpython.com/python-gui-tkinter/

Atlassian. (2025). *Great outcomes start with jira*. https://www.atlassian.com/software/jira?campaign

Brownlee, J. (2023, November 1). *Python Threading: The Complete Guide*. SuperFastPython. https://superfastpython.com/threading-in-python/

Campbell, D. (2024, January 12). *Python socket programming: Server-client connection*. Pubnub https://www.pubnub.com/blog/python-socket-programming-client-server/

Discord. (2025). *Group chat that's All Fun & Games*. https://discord.com/

Eyalvr12. (2024, December 2). *Simple multi-client chat using threads*. Python https://discuss.python.org/t/simple-multi-client-chat-using-threads/73043

GeeksForGeeks. (2018a, May 30). *Secrets | Python module to Generate secure random numbers.* https://www.geeksforgeeks.org/secrets-python-module-generate-secure-random-numbers/#

GeeksForGeeks. (2018b, February 14). *SHA in Python*. https://www.geeksforgeeks.org/sha-in-python/

Hoffman, C. (2017, June 12). *Why you shouldn't use SMS for two-factor authentication (and what to use instead)*. How. https://www.howtogeek.com/310418/why-you-shouldnt-use-sms-for-two-factor-authentication/

Hron, M., Obwegeser, N., Bass, J. M., Clarke, P., Cooper, R. G., Dikert, K., Eloranta, V. P., Hoda, R., Kettunen, P., Aamir, M., Abrahamsson, P., Ali, A., Ali, M., Alperowitz, L., Alqudah, M., Angelov, S., ... Farid, W. M. (2022). Why and how is Scrum being adapted in practice: A systematic review. *Journal of Systems and Software*, 182, 111061. https://doi.org/10.1016/j.jss.2021.111061

Kumar, P. (2025, April 3). *Understanding race conditions in python and how to handle them*. Medium. https://medium.com/yavar/understanding-race-conditions-in-python-and-how-to-handle-t hem

Python documentation. (2025). *Tkinter - Python interface to TCL/TK*. https://docs.python.org/3/library/tkinter.html

Python Software Foundation. (n.d.). *json — JSON encoder and decoder.* https://docs.python.org/3/library/json.html

SQLite. (2025). *Documentation*. https://www.sqlite.org/docs.html

Thought. (2022, December 6). *Client/server TCP socket programming*. Python. https://discuss.python.org/t/client-server-tcp-socket-programming/21701

Tripathi, S. (2023, October 2). *Guide to python socket programming*. BuiltIn Builhttps://builtin.com/data-science/python-socket

Yegulalp, S. (2024, May 22). *Why You Should Use Sqlite*. InfoWorld https://www.infoworld.com/article/2337363/why-you-should-use-sqlite-3.html