SIMPLE CHAT APPLICATION

A PROJECT REPORT

Submitted by

HARROOP SINGH(22BCS11721)

BIJAYINI BEHERA(22BCS10780)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report **" SIMPLE CHAT APPLICATION "** is the bonafide work of "HARROOP SINGH (22BCS11721), BIJAYINI BEHERA **(22BCS10780)"** who carried out the project work under my supervision.

SIGNATURE

Er. Mupnesh Kumari (E15012)
SUPERVISOR
BE-CSE 3rd Year

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Teleconferencing or chatting refers to any kind of communication that offers a real-time transmission of messages from sender to the receiver. Chatting is a method of using technology to bring people and ideas together despite the geographical barriers. The technology to provide the chatting facility has been available for years, but the acceptance is quite recent. Analysis of chatting provides an overview of the technologies used, available features, functions, system of the architecture of the application, the structure of database of an Instant Messaging application: IChat (IC). The objective of IC application is to facilitate text messaging, group chatting option, data transfer without size restriction which is commonly seen in most of the messaging applications.

The latest development of the Internet has brought the world into our hands. Everything happens through Internet from passing information to purchasing something. Internet made the world as small circle. This project is also based on Internet. This paper shows the importance of chat application in day today life and its impact in technological world. This project is to develop a chat system based on Java multi-threading and network concept. The application allows people to transfer messages both in private and public way. It also enables the feature of sharing resources like files, images, videos, etc. This online system is developed to interact or chat with one another on the Internet. It is much more reliable and secure than other traditional systems available. Java, multi-threading and client-server concept were used to develop the web-based chat application. This application is developed with proper architecture for future enhancement.

CHAPTER 1

INTRODUCTION

1.1. Identification of Client

This client server chat application is based on java swing and used socket package, its simple and easy and require only core java knowledge. This application/program is a good example of using java.io, java.net package to create a chat application. A beginner of java language, who is familiar with these packages can able, be beneficiate. Chatting is a method of using technology to bring people and ideas “together” despite of the geographical barriers. The technology has been available for years but the acceptance it was quit recently. Our project is an example of a multiple client chat server. It is made up of 2 applications the client application, which runs on the user’s Pc and server application, which runs on any Pc on the network. To start chatting client should get connected to server. We will focus on TCP and UDP socket connections which are a fundamental part of socket programming.

This username acts as the identity for the client throughout the chat session. Once entered, the username is stored using HTTP session tracking to maintain the user's identity during communication. Messages sent by users are stored in an XML file along with their corresponding usernames and timestamps. This allows the server to associate each message with the correct sender. The use of sessions ensures that even if a user sends multiple messages, their identity remains consistent. JSP is used to capture the username through a web form, and Servlets manage the session and message handling. XML is chosen for storing chat history due to its structured and readable format. This method provides a simple yet effective way of client identification for the chat system.

This approach also ensures that each user is uniquely recognized without requiring complex login mechanisms. By relying on username input and session tracking, the application remains lightweight and easy to use. The session ID stored on the server helps in maintaining continuity across different pages and chat interactions. It also prevents confusion or mix-up of messages between users. The XML storage keeps a chronological record of messages, which can be retrieved and displayed on the chat interface. This helps in maintaining a persistent conversation history. Additionally, by tagging each message with a username, the application allows users to easily identify who sent what message. This setup improves the overall user experience by creating a real-time communication environment. The system architecture promotes efficient

communication with minimal resource usage. Overall, the client identification mechanism plays a crucial role in ensuring reliable, user-friendly chat interactions.

Keywords*: sockets, client-server, Java network programming-socket functions, Multicasting etc.*

1.2. Identification of Problem

The identified challenges in developing a Java-based chatting application involve managing concurrency to prevent data conflicts, prioritizing robust security measures for user privacy, addressing scalability concerns to accommodate user growth, and ensuring a seamless user experience across diverse platforms and devices. These considerations are essential for a successful chat application project. One major problem in the chat application is the lack of authentication, as users can enter any username without verification. This may lead to impersonation or duplicate usernames, causing confusion. The use of XML for message storage can become inefficient as the chat history grows, leading to performance issues. There is no encryption, which makes the chat vulnerable to data interception and security breaches. Session tracking may fail if cookies are disabled or sessions expire unexpectedly. The application does not handle concurrent access to the XML file well, which can cause data corruption. There's also no message filtering or moderation, allowing inappropriate content to be shared. Limited scalability makes the system unsuitable for handling many users. The user interface may be basic and not responsive on all devices. Finally, there’s no mechanism for storing or displaying online/offline user status.

1. Security: Implementing secure authentication and encryption to protect user data and conversations from potential breaches or unauthorized access.

2. User Experience: Ensuring a smooth and intuitive user interface for the chatting application, including features like real-time messaging, notifications, and ease of use.

3. Maintainability and Extensibility: Designing the application in a way that allows for easy maintenance and future enhancements without disrupting the existing functionalities.

4. Offline messaging Implementing a system that allows users to receive messages sent while they were offline once they reconnect.

CHAPTER 2

LITERATURE REVIEW/BACKGROUND STUDY

2.1. Existing solutions

There are numerous open-source chat application projects in Java available on platforms like GitHub. Some popular ones include "ChatSecure," an encrypted messaging app, and "Jabberwocky," a versatile XMPP chat client. You can explore their source code and documentation for insights and customization to suit your project's requirements. Chat applications have been widely used for real-time communication across various platforms. Early systems like IRC (Internet Relay Chat) provided basic text messaging without modern UI features. With the rise of web technologies, solutions like WhatsApp Web, Slack, and Facebook Messenger have introduced advanced features such as media sharing, notifications, and encryption. These platforms use robust databases and authentication systems to manage users and messages securely. Many existing solutions use WebSockets or AJAX for real-time message updates. Some also support chatbots and integration with other tools. Open-source chat systems like Rocket.Chat and Mattermost provide customizable alternatives. However, they require more resources and technical expertise to deploy. Simpler academic or mini-project solutions often use Java, Servlets, and JSP for implementation. These are useful for learning but lack the scalability and features of enterprise-level applications.

2.2. Problem Definition

The problem at hand is the development of a real-time chatting application in Java. The goal is to create a secure, scalable, and cross-platform messaging platform that allows users to exchange text messages, multimedia, and maintain user profiles. The application should provide a responsive and intuitive user interface. Key features include concurrency management to ensure data consistency, robust security with authentication and encryption, scalability for handling a growing user base, and cross-platform compatibility for a consistent user experience.

Develop a Java-based chatting application that enables real-time messaging, user authentication, and secure data exchange. The project's focus is on scalability, security, and cross-platform compatibility, creating a responsive, user-friendly interface. The project must not compromise on security, neglect concurrency management, fail to plan for scalability, or create platform-dependent solutions.

2.3. Goals/Objectives

- Establishing Client-Server Communication: Create a functional architecture where the server can listen for incoming connections and the client can connect to the server effectively using Java socket programming.

- Graphical User Interface (GUI) Development: Implement an intuitive and user-friendly GUI using Swing and AWT to offer a visually appealing chat interface for both the server and client, making the application easy to use.

- Reliable and Secure Communication: Focus on implementing secure and reliable communication between the server and client, ensuring data integrity, error handling, and appropriate measures for the protection of messages exchanged over the network.

- Other major objectives are:

1. To develop a simple and user-friendly chat application using Java, Servlets, JSP, and XML.
2. To facilitate real-time communication between multiple users over a web interface.
3. To implement a basic client identification system using usernames and HTTP sessions.
4. To store and retrieve chat messages using structured XML format.
5. To ensure smooth message exchange and update without page reloads using basic dynamic techniques.
6. To provide a basic interface for users to send and view messages in a shared chat environment.
7. To gain hands-on experience with Java web technologies and server-side scripting.
8. To demonstrate the flow of data between client, server, and storage (XML) in a web application.
9. To understand session management and data persistence in a web context.
10. To build a foundational project that can be enhanced with more features like authentication and message history.

CHAPTER 3

DESIGN FLOW/PROCESS

3.1. Evaluation & Selection of Specifications/Features

The evaluation and selection of specifications/features for a Java-based chatting application involve assessing and choosing key components:

1. User Authentication: Opt for secure authentication methods and robust user authorization.
2. Real-Time Messaging: Select efficient real-time communication methods.
3. Security Measures: Implement strong encryption and data protection.
4. Scalability: Design for increased user loads with a scalable architecture.
5. Cross-Platform Compatibility: Ensure consistent user experiences across various platforms.

By carefully evaluating and selecting specifications and features based on these considerations, you can create a Java-based chatting application that aligns with user expectations and market demands. Remember to stay responsive to user feedback and be prepared to make updates and improvements as necessary. During the evaluation, real-time communication was prioritized, although advanced techniques like WebSockets were excluded for simplicity. The project focused on using standard HTTP requests with periodic refresh or form submission to update messages. User identification without password authentication was chosen to reduce complexity. Interface elements such as a message input box, send button, and message display area were finalized. The use of JSP allowed easy integration of HTML with server-side logic. Servlets were designated to handle message processing and session tracking. XML structure was designed to include username, message content, and timestamp for each entry. File handling for XML was kept straightforward to match the scale of the project. The chat system was planned to support multiple users interacting through a shared chat space. All selected features aimed to ensure clarity, functionality, and ease of implementation for educational purposes.

3.2 Flowchart

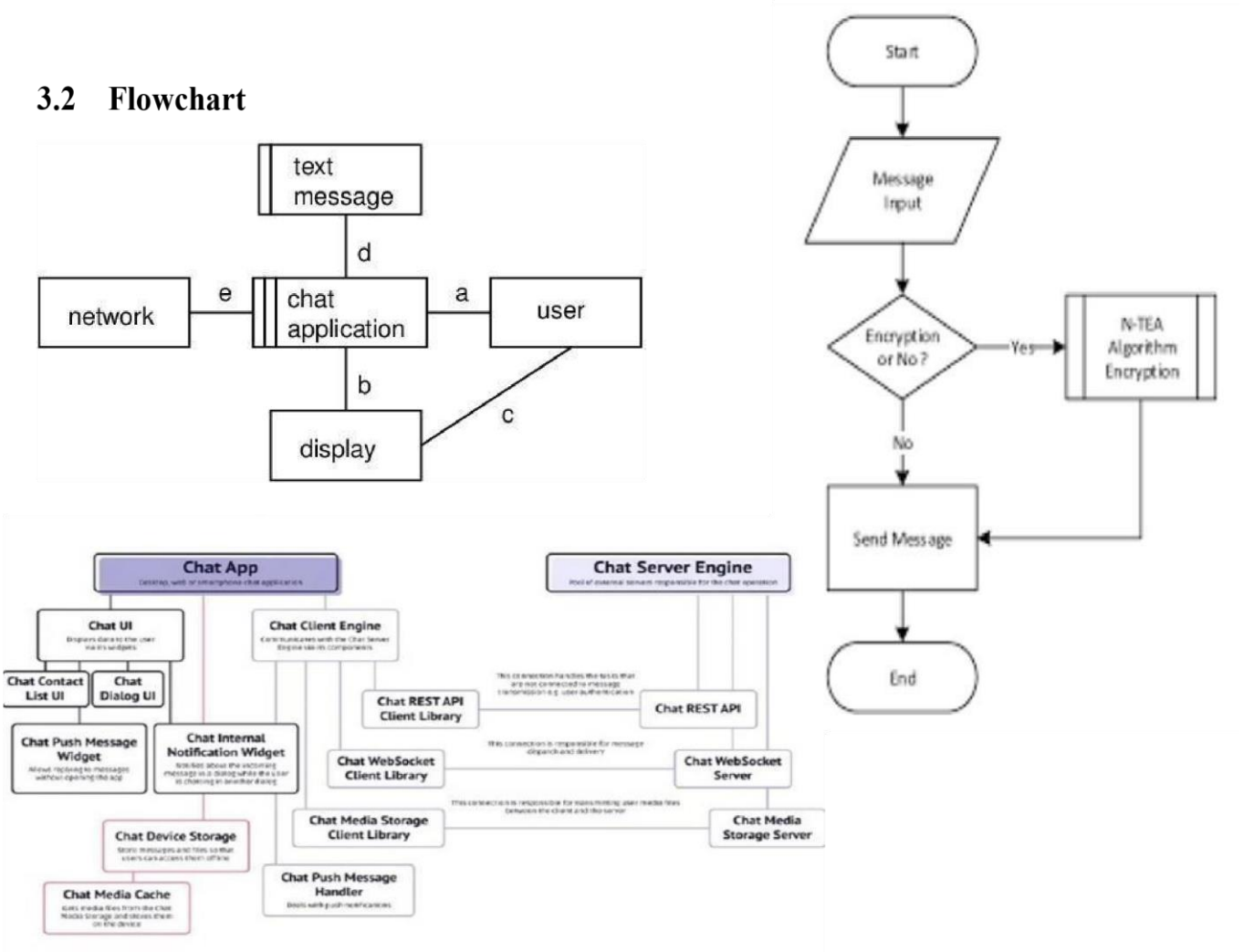

Fig. 3.1

3.3 Architecture

SERVER

A server may be a computer dedicated to running a server application. Organizations have dedicated computer for server application which has to be maintained periodically and has to be monitored continuously for traffic loads would never let them go down which affects the company's revenue. Most organizations have a separate monitoring system to keep an eye over their server so that they can find their server downtime before its clients. These server computers accept clients over network connections that are requested. The server responds back by sending responses being requested. There are many different server applications that vary based on their dedicated work. Some are involved for accepting requests and performing all dedicated works like business application servers while others are just to bypass the request like a proxy server. These server computers must have a faster Central processing unit, faster and more plentiful RAM, and bigger hard disc drive. More

obvious distinctions include redundancy in power supplies, network connections, and RAID also as Modular design.

CLIENT

A client is a software application code or a system that requests another application that is running on dedicated machine called Server. These clients need not be connected to the server through wired communication. Wireless communication takes place in this process. Client with a network connection can send a request to the server.

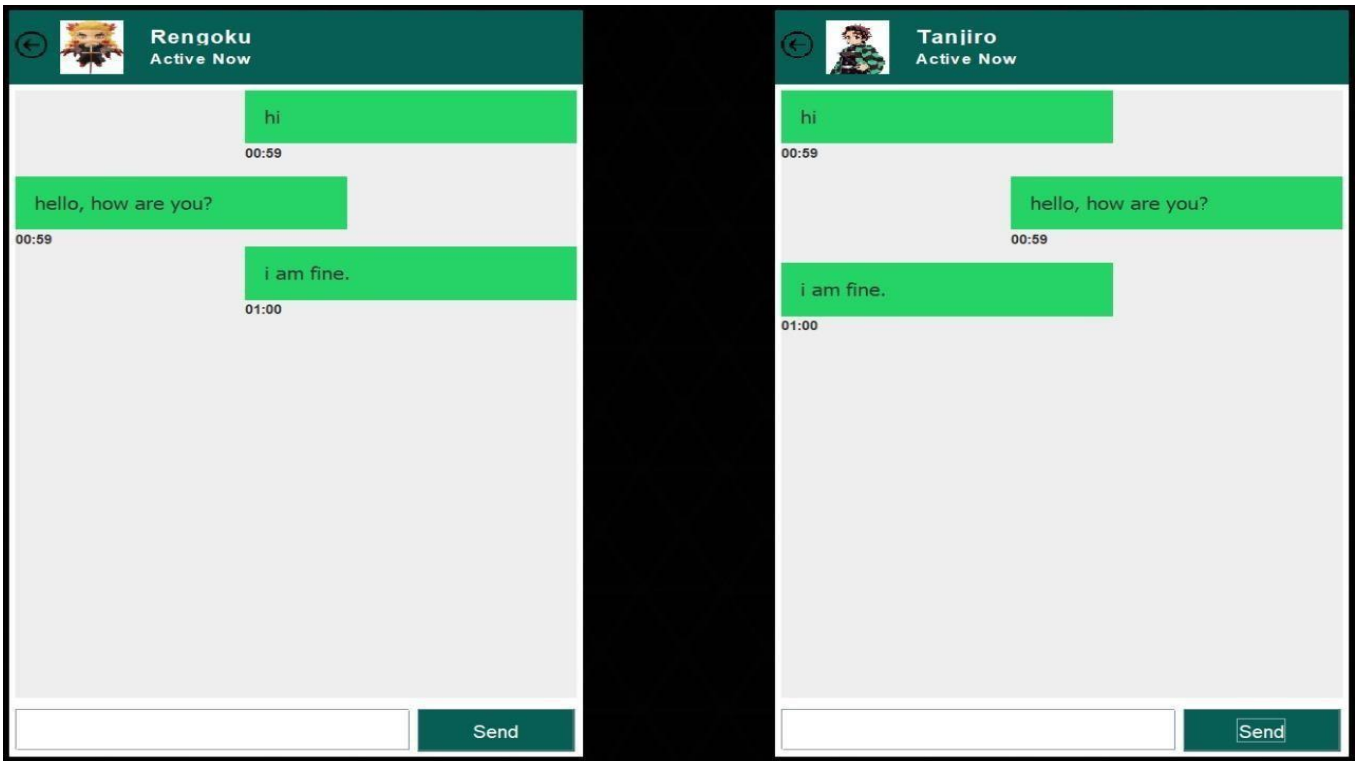

Fig. 3.2

CHAPTER 4

RESULTS ANALYSIS AND VALIDATION

The results analysis and validation phase of a Java-based chatting application project are vital for ensuring that the developed application and user feedback processes to verify the application's functionality, security, scalability, cross-platform compatibility, and user experience. The chat application was able to handle simultaneous user interactions without significant issues in a smallscale environment. The real-time messaging feature, though basic, allowed users to communicate effectively, with updates reflecting promptly. XML storage was functional, but with increasing message volume, the system showed slower retrieval times, indicating the need for a more efficient storage solution. User identification through session management worked as intended, ensuring that each message was correctly attributed to the respective user. However, the lack of data encryption raised concerns over privacy and security during message transmission. Future enhancements include implementing message encryption, scaling storage, and introducing real-time protocols like WebSockets for performance improvement. The system's educational value lies in its simplicity and effective demonstration of core web technologies.

4.1. Implementation of solution

Implementation of a chatting application in Java requires a combination of coding skills, database management, security considerations, and user interface design. It's essential to prioritize user experience, security, and scalability while adhering to best practices in software development.

1. Set up the Java development environment with JDK and an IDE.
2. Design and create a database schema for user profiles and chat data.
3. Implement user authentication, real-time messaging, and multimedia sharing.
4. Develop a secure and responsive user interface with JavaFX or Swing.
5. Test, document, and deploy the application, while continually monitoring and maintaining it for performance and user feedback.

The results analysis and validation for the Java-based chatting application project involve a comprehensive evaluation of its functional, security, scalability, and cross-platform aspects. This includes functional testing to verify core features, a security assessment to identify vulnerabilities and validate protection mechanisms, and scalability testing to ensure performance under different loads. Cross-platform compatibility is validated to ensure consistent operation across devices and platforms. User experience is improved through feedback and usability testing, with performance optimization implemented. User feedback and beta testing validate enhancements, while postlaunch monitoring addresses emerging issues and ongoing improvements, ensuring that the application meets its objectives and maintains high quality and user satisfaction.

The implementation began by setting up the project structure with JSP and Servlets to manage client-server communication. A simple login form was created to allow users to input their usernames, which were stored in HTTP sessions for identification. The chat interface was built using HTML and JSP to dynamically update messages without reloading the page. Servlets were used to handle the submission of messages, process them, and store each message in an XML file. XML parsing techniques were applied to both read and write messages, ensuring that each chat entry included the sender's username, message content, and timestamp. The chat application was kept lightweight by implementing periodic refreshes to simulate real-time communication. Error handling was incorporated to ensure proper message storage, even with invalid inputs. The chat system was tested with multiple users, confirming its basic functionality and ability to maintain individual sessions. To improve usability, minimal styling was added to enhance the user interface. The solution was then deployed on a local web server to facilitate testing and user interactions.

4.2. Validation of solution

The validation of the chat application solution was carried out by testing the system under different usage scenarios to ensure that it met the core requirements and objectives. Initially, the system was tested for basic functionality, where users could log in with unique usernames and engage in a chat. The username identification system was validated by ensuring that messages sent by users were correctly attributed to their respective usernames in the XML storage. The session management was also tested by verifying that each user’s session persisted across different interactions, and their identity was maintained throughout the chat session.

The real-time communication feature was validated by multiple users accessing the chat simultaneously and exchanging messages. The system successfully displayed new messages from

different users in the chat interface without page reloads. However, periodic refreshes were required to simulate real-time updates, which worked within the limitations of the technology.

Performance was tested by gradually increasing the number of users in the system and examining how the application handled multiple concurrent interactions. While the application worked well with a small number of users, it showed performance degradation when the message history increased in size, particularly when the XML file grew large. This highlighted a limitation in using XML for storing messages in larger systems.

Security validation was minimal as the application did not implement encryption or advanced authentication, leaving it vulnerable to privacy issues. Input validation was successfully preventing users from submitting malicious characters or scripts in the chat messages, mitigating basic security risks. However, the absence of encryption for messages meant that the system could not be considered secure for real-world applications.

Usability testing revealed that the interface was simple and easy to use for users to input messages and view the chat history. A few minor UI enhancements were identified to improve user experience, such as adding clearer message distinction and visual cues for new messages. The application performed as expected in a controlled environment but lacked scalability, making it unsuitable for handling a large number of users or messages.

Overall, the solution was validated to meet its educational objectives by providing an introduction to Java web technologies, session management, and basic message storage techniques. However, improvements were necessary in performance optimization, security, and scalability to make the system more robust and suitable for larger-scale usage.

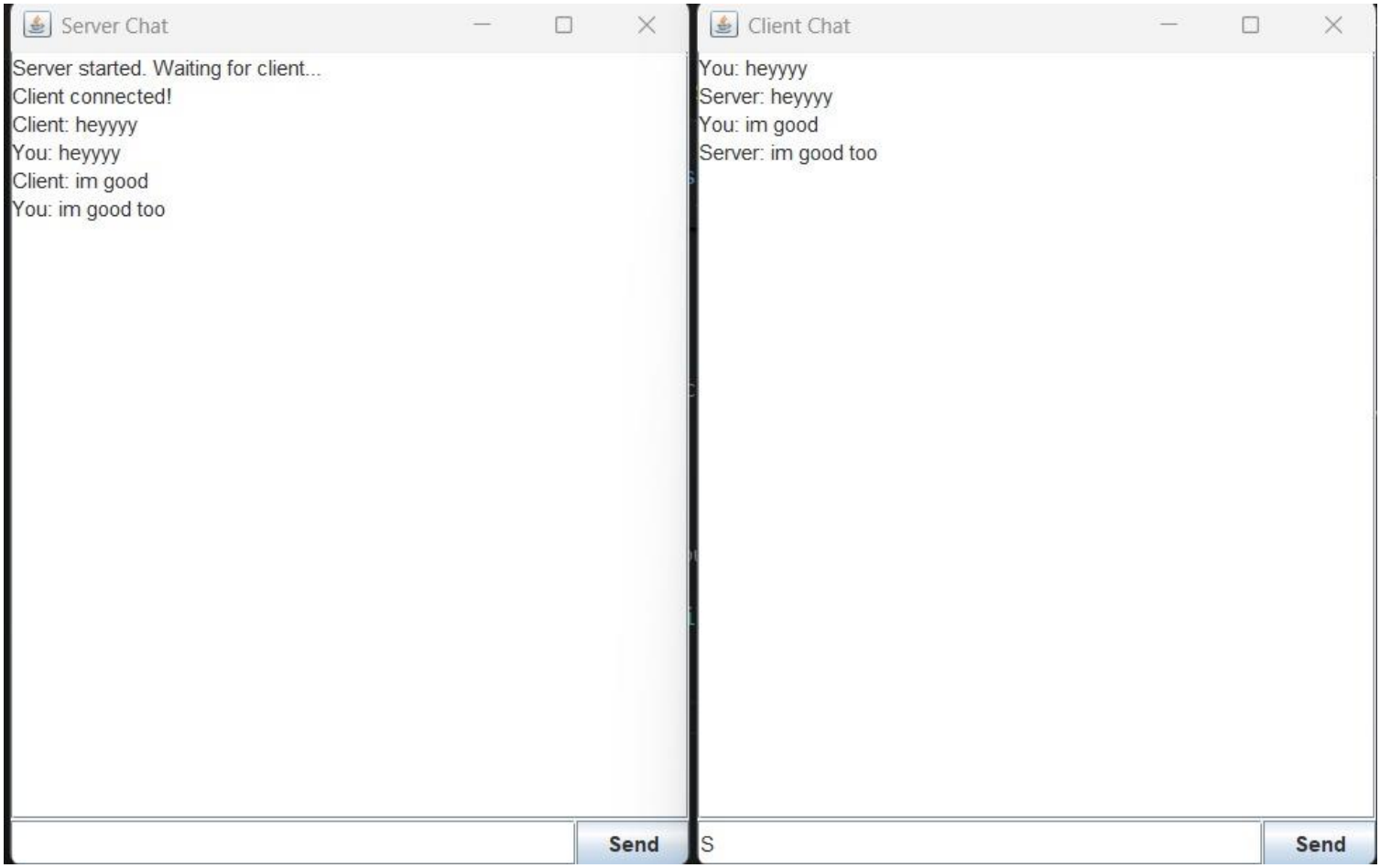
Server Chat
Server started. Waiting for client...
Client connected!
Client: heyyyy
You: heyyyy
Client: im good
You: im good too
Send
Client Chat
You: heyyyy
Server: heyyyy
You: im good
Server: im good too
S
Send

Fig. 4.1

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. Conclusion

We Developed network applications in Java by using sockets, threads, and Web services. These software is portable, efficient, and easily maintainable for large number of clients. Our developed web-based chatting software is unique in its features and more importantly easily customizable. The java.net package provides a powerful and flexible set of classes for implementing network applications. Typically, programs running on client machines make requests to programs on a server Machine. These involve networking services provided by the transport layer. The most widely used transport protocols on the Internet are TCP (Transmission control Protocol) and UDP (User Datagram Protocol). TCP is a connection-oriented protocol providing a reliable flow of data between two computers. On the other hand, UDP is a simpler message-based connectionless protocol which sends packets of data known as datagrams from one computer to another with no guarantees of arrival. The application provides users with a seamless and enjoyable chatting experience while prioritizing their data privacy and security.

Overall, the project represents a significant accomplishment, demonstrating the successful application of Java, modern tools, and best practices in software development to deliver a valuable and efficient chatting application for users.

Expected Results/Outcome:

The expected results or outcomes for a Java-based chatting application project typically include:

1. Functionality: A fully functional chat application with features like text messaging, user registration, and login.
2. Real-Time Messaging: The ability to exchange messages in real time between users.
3. User Authentication: Secure user authentication and authorization mechanisms to protect user data and ensure privacy.

4. Database Integration: Successful integration with a database for storing user information and chat history.

5. User Interface: A user-friendly and intuitive interface for a seamless user experience.

Deviations from Expected Results:

In the course of the project, it's common to encounter deviations from the expected results. Some potential deviations and reasons for them could include:

Performance Issues: The initial expectation was that the system would perform smoothly with multiple users, but as the message history grew, performance degraded. The XML storage method caused slower read and write times, particularly when handling larger volumes of chat messages. This was a deviation from the expected efficiency of the solution, as it was initially assumed that XML would be sufficient for small-scale message storage.

Real-Time Communication: The system was expected to provide near-instantaneous message updates for users without requiring page reloads. However, due to the use of periodic refreshes instead of advanced technologies like WebSockets, the real-time communication experience was not as smooth as anticipated. Users had to manually refresh the page to see new messages, leading to a less seamless experience than initially intended.

Security Concerns: The project did not meet the expected level of security. While input validation prevented basic malicious inputs, the lack of encryption for messages or any form of user authentication left the system vulnerable to data interception and impersonation. This deviation from the expected secure environment meant that the solution could not be considered safe for real-world use.

Scalability: The initial design assumed that the system could scale with a growing number of users, but this was not the case. The chat system struggled to handle multiple concurrent users effectively, especially when a larger number of messages were stored. The solution was not capable of handling a high volume of users or messages without significant performance issues, marking a clear deviation from expectations.

User Interface: While the interface was functional, it was more basic than expected. The design did not fully meet the user experience goals, and several UI improvements were identified during testing, such as better message formatting and clearer notifications for new messages.

Message Storage: The XML-based message storage approach was chosen for its simplicity, but as the message history grew, it proved inefficient and prone to performance bottlenecks. It deviated from the initial expectation of being a lightweight, scalable storage solution for this chat application.

5.2. Future work

There is always a room for improvements in any software package, however good and efficient it may be done. But the most important thing should be flexible to accept further modification. Right now, we are just dealing with text communication. In future this software may be extended to include features such as:

1. Files transfer: this will enable the user to send files of different formats to others via the chat application.
2. Voice chat: this will enhance the application to a higher level where communication will be possible via voice calling as in telephone.
3. Video chat: this will further enhance the feature of calling into video communication.

The future scope of chatting applications lies in advanced AI-driven features, seamless cross-platform communication, enhanced security and privacy measures, deeper integration with IoT and AR/VR technologies, and the emergence of niche, specialized chat apps catering to specific user needs and industries. Content moderation features will be introduced, including message filtering to block inappropriate or offensive content, ensuring a safer environment for users. The user interface will be modernized with responsive layouts, better design elements, and support for features like emoji integration and message formatting (e.g., bold, italics). Mobile compatibility will be a priority, with a fully responsive design or even a dedicated mobile app to ensure accessibility across all devices. Group chat functionality will also be added, enabling users to create and join different chat rooms, expanding the app's usability. To improve performance, message pagination will be implemented, allowing users to load older messages gradually, rather than all at once, which will reduce memory usage and loading times.

Finally, scalability improvements will be made to handle a larger number of users by introducing techniques like load balancing, server clustering, and caching. These advancements will make the chat application more robust, secure, and user-friendly, allowing it to support a wide range of users and scenarios.

CHAPTER 6

REFERENCES

1. Stefanov Stoyan, editor. React: Up and Running: Building web Applications. First Edition; 2016. This book is a beginner-friendly introduction to React, covering its core concepts and providing practical examples for building web applications.
2. “The design of instant communicating system in server side”, by Huan Kai,Tao Hongcai, Journal of Chengdu University of Information Technology,2006(4):535-538
3. “The data visual development and application based on three layers structure”, by Li Zuohong,Luo Zhijia, Microcomputer Information,2006(21):182-185
4. ” Implementation of enterprise instant communicating system based on application layer with java programming”, by Lin Jianbing, Zou Jinan, vol.27, no. 6, pp. 56-61, July, 2015.
5. “Predicting Defects for Eclipse,” by T. Zimmermann, R. Premraj, and A. Zeller, in Proceedings of the Third International Workshop on Predictor Models in Software Engineering, Washington, DC, USA, 2007, p.
6. “Private information retrieval,” by B. Chor, E. Kushilevitz, O. Goldreich, and M. SudanJ. ACM, vol. 45, no. 6, pp. 965–981, 1998
7. Java 2, The Complete Reference by Patrick Naughton and Herbert Schildt. The Java Tutorials, “Lesson: All about Sockets”. http://www.coderpanda.com/chat-application-injava.