North South University

Department of Electrical & Computer Engineering

PROJECT REPORT

Course Code: CSE215

Course Title: Programming Language II

Faculty: Mohammad Shifat-E-Rabbi (MsRB)

Project Name:

Online Chat Application – GlobalHub

Date of Submission: 3rd December 2024

Section: 8

Group Number: 3

Team Members:-

1. Fazly Fardin Chowdhury - 2322480642
2. Nayer Ali Ahsan - 2321749642
3. Md Shamiullah Shek Raiyan - 2321180642

Introduction

This report presents the Global Hub, a chat application designed to facilitate real-time communication among users. The application features user registration, authentication, and messaging functionalities, while ensuring data persistence through serialization. This document outlines the architecture, functionality, implementation details, and future enhancements of the application.

Keywords: Chat application, user authentication, data persistence, serialization, Java Swing.

Technologies Used

• Programming Language: Java

• GUI Framework: Swing

• Serialization: Java Serialization for data persistence

• Development Environment: VS Code

• Version Control: Git

• Database: binary I/O file

In the digital age, effective communication is paramount. Chat applications have emerged as a vital tool for personal and professional interactions. The Global Hub aims to provide a robust platform for real-time communication, allowing users to send messages, register accounts, and manage their profiles seamlessly. This report details the design and implementation of the Global Hub, highlighting its architecture, features, and user interface, as well as discussing potential future enhancements.

System Architecture

The architecture of Global-Hub consists of the following components:

• **Hub Class:** Manages user and message data, handles authentication, and performs data persistence.

• **User Class:** Represents user entities with attributes such as username,

password, email, name, and age.

• **Message Class:** Represents messages sent by users.

• **GUI Class:** Manages the user interface and user interactions.

The system follows a Model-View-Controller (MVC) pattern, separating the data management logic from the user interface. The User Interface is developed using Java Swing, which provides a rich graphical interface for user interaction. The UI is designed to be intuitive and user-friendly, comprising the following views:

Welcome View: The initial screen that presents options for user login and registration.

Login View: A dedicated interface for users to enter their credentials to access their accounts.

Registration View: An interface that allows new users to create an account by providing necessary details.

Chat Hub View: The main interface for users to send and receive messages in real-time.

User List View: Displays a list of registered users, enhancing community interaction.

The backend of the Global Hub is implemented in Java and is responsible for the core functionalities of the application:

User Management: This module handles user registration, authentication, and profile management.

Messaging System: Manages the creation, storage, and retrieval of messages between users, ensuring a smooth communication experience.

State Management: Maintains the current state of the application, allowing users to navigate seamlessly between different views based on their interactions.

Data persistence is a critical aspect of the Global Hub. The application employs Java's serialization mechanism to store user information and messages in a binary file (data.bin). This approach ensures that user data is retained even after the application is closed, enabling users to resume their sessions without loss of information.

The Global Hub application is structured into several key classes, each responsible for specific functionalities. Below is a detailed overview of the main classes and their responsibilities.

The App class serves as the entry point of the application. It initializes the Hub and Gui objects and manages the application's state through a continuous loop. The state transitions include:

Welcome: Displays the welcome view and prompts users to log in or register.

Login: Handles user authentication and transitions to the chat hub upon successful login.

Hub: Displays the chat interface for real-time messaging.

Register: Manages user registration and validates input data.

User List: Shows a list of registered users to facilitate interaction.

The Hub class is responsible for managing users and messages within the application. Key methods include:

create_user(String username, String password, String name, String email, int age): Registers a new user after

validating the input data against predefined criteria.

authenticate(String username, String password): Validates user credentials and retrieves the corresponding user object if authentication is successful.

create_message(User user, String message): Creates a new message object and adds it to the list of messages, ensuring that all messages are associated with the correct user.

save(): Serializes the current state of the Hub to the binary file, preserving user and message data for future sessions.

load(): Loads user and message data from the binary file, allowing the application to restore its state upon startup.

The User class encapsulates all relevant user details, including:

Attributes: username, password, email, name, age, which are essential for user identification and authentication.

Methods: Includes validation methods to ensure that usernames and emails are correctly formatted and that the age meets the minimum requirement for registration.

The Message class represents a message sent by a user. It contains:

Attributes: User user, String message, which link the message to the sender and store the content of the message.

Constructor: Initializes the message with the sender's user object and the message content, ensuring that each message is associated with the correct user.

The Gui class manages the graphical user interface and user interactions. Key methods include:

welcome_view(Hub hub): Displays the welcome screen with options for login and registration, and shows the total number of users.

login_user(Hub hub): Manages the login process, including input validation and error handling for incorrect credentials.

register_view(Hub hub): Facilitates user registration by collecting necessary information and validating it before creating a new user.

hub_view(Hub hub): Displays the main chat interface, allowing users to send and receive messages in real-time.

userlist_view(Hub hub): Presents a list of registered users, enhancing community interaction and engagement.

The user experience is a critical aspect of the Global Hub application. The design focuses on simplicity and ease of use, ensuring that users can navigate through the application without confusion. The following features enhance the user experience:

Intuitive Navigation: Clear transitions between different views (welcome, login,

registration, chat) allow users to easily find their way through the application.

Real-Time Messaging: Users can send and receive messages instantly, creating a dynamic and engaging chat environment.

Error Handling: Comprehensive error messages guide users in correcting input mistakes, such as invalid usernames or passwords, enhancing the overall usability of the application.

User List: The ability to view other registered users fosters a sense of community and encourages interaction among users.

To ensure the reliability and functionality of the Global Hub application, a thorough testing process was implemented. This included:

Unit Testing: Individual components, such as user registration and message creation, were tested to verify that each function performs as expected.

Integration Testing: The interaction between different components (e.g., user interface and backend logic) was tested to ensure seamless operation.

User Acceptance Testing: Feedback from potential users was gathered to identify areas for improvement and to validate the overall user experience.

While the Global Hub application provides a solid foundation for real-time communication, several enhancements could be implemented to improve functionality and user experience:

Group Chat Functionality: Allowing users to create and participate in group chats would enhance the application's versatility and appeal.

File Sharing: Implementing a feature for users to share files and images would enrich the communication experience.

Mobile Application: Developing a mobile version of the Global Hub would increase accessibility and user engagement, catering to a broader audience.

Enhanced Security: Implementing additional security measures, such as two-factor authentication, would further protect user accounts and data.

The Global Hub chat application represents a significant step forward in facilitating real-time communication among users. With its user-friendly interface, robust backend logic, and data persistence capabilities, the application is well-equipped to meet the needs of its users. Future enhancements will further expand its functionality and improve the overall user experience, ensuring that Global Hub remains a relevant and valuable tool in the realm of digital communication.

The development of the Global Hub application would not have been possible without the support and guidance of mentors and peers. Special thanks to [insert names or institutions] for their invaluable contributions and feedback throughout the project.

Database

The GlobalHub project uses a binary file named data.bin as its database to persist user and message data. It stores serialized instances of the Hub class, which includes User and Message objects. The application utilizes Java's ObjectOutputStream and ObjectInputStream for saving and loading data, ensuring that the latest state is maintained across sessions. Error handling is incorporated to manage scenarios where the file may not exist or is empty, allowing the application to function smoothly without data loss.

References

1. **GeeksforGeeks:** *Message Dialogs in Java GUI.* Retrieved from https://www.geeksforgeeks.org/message-dialogs-java-gui/
2. **Chatgpt:** *Multithreading Implementation Using the Thread Class.* Developed with assistance from ChatGPT.
3. **Javatpoint:** *Binary Input/Output Serialization in Java.* Retrieved from https://www.javatpoint.com/java-io

Contributors

1. **Fazly Fardin Chowdhury** - 2322480642
   - Led the development of the user authentication system, implementing login and registration functionalities.
   - Designed and developed the graphical user interface (GUI) for the chat application, ensuring a user-friendly experience.
   - Conducted extensive testing and debugging to enhance performance and reliability, contributing approximately 70% of the overall effort in the project.
2. **Nayer Ali Ahsan** - 2321749642
   - Created the PowerPoint presentation to effectively communicate the project's objectives, features, and functionalities to stakeholders.
   - Assisted in the design of the messaging system, enabling users to send and receive messages in real-time.
3. **Md Shamiullah Shek Raiyan** - 2321180642
   - Compiled the project report, summarizing the development process, challenges faced, and solutions implemented.
   - Contributed to the overall architecture of the application, integrating various components seamlessly.

Conclusion

GlobalHub successfully implements a real-time chat application that allows users to communicate effectively. The project demonstrates the ability to manage user data securely while providing an interactive user interface. The application is modular, making it easy to extend and maintain.

Appendix

1. Code Explanation

This section provides an overview of the key classes and methods implemented in the GlobalHub application.

1.1 App Class

- **Purpose:** The entry point of the application.
- **Key Methods:**
  - **main(String[] args)**: Initializes the Hub and GUI, and manages the application state through a loop that switches between different views (welcome, login, hub, register, userlist).

1.2 Hub Class

- **Purpose:** Manages user and message data, including authentication and data persistence.
- **Key Methods:**
  - **create_user(...)**: Validates and creates a new user.
  - **authenticate(...)**: Checks if the provided username and password match an existing user.
  - **create_message(...)**: Creates a new message and stores it.
  - **load()**: Loads user and message data from a serialized file.

1.3 GUI Class

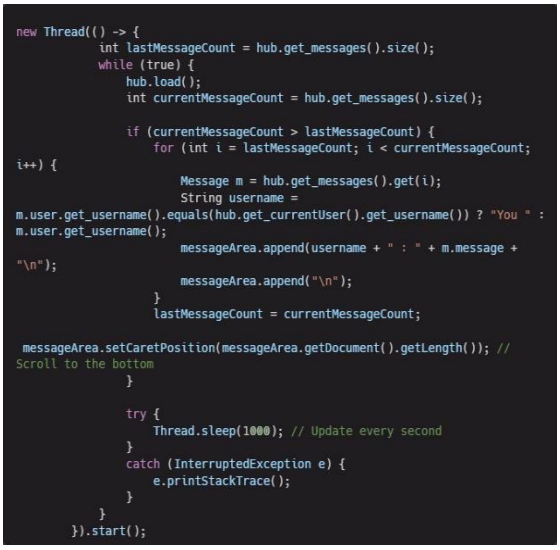
- **Purpose:** Manages the graphical user interface and user interactions.
- **Key Methods:**
  - **login_user(Hub hub)**: Displays a login dialog and handles user authentication.
  - **hub_view(Hub hub)**: Displays the chat hub interface where users can send and receive messages.
  - **register_view(Hub hub)**: Displays a registration dialog for new users.
  - **userlist_view(Hub hub)**: Displays a dialog showing all registered users.

2. Multithreading

The **hub_view(Hub hub)** method in the **GUI** class contains a key multithreading implementation that enables real-time message updates in the chat interface. Here's a detailed explanation of how it works:

- **Purpose of Multithreading:** The primary goal of using a separate thread in this method is to allow the user interface to remain responsive while continuously checking for new messages. Without multithreading, the GUI would freeze while waiting for new messages to load, resulting in a poor user experience.

- **Thread Implementation:** Inside the **hub_view()** method, a new thread is created using the following code:



```
new Thread(() -> {
            int lastMessageCount = hub.get_messages().size();
            while (true) {
                hub.load();
                int currentMessageCount = hub.get_messages().size();

                if (currentMessageCount > lastMessageCount) {
                    for (int i = lastMessageCount; i < currentMessageCount;
i++) {
                        Message m = hub.get_messages().get(i);
                        String username =
m.user.get_username().equals(hub.get_currentUser().get_username()) ? "You " :
m.user.get_username();
                        messageArea.append(username + " : " + m.message +
"\n");
                        messageArea.append("\n");
                    }
                    lastMessageCount = currentMessageCount;

 messageArea.setCaretPosition(messageArea.getDocument().getLength()); //
Scroll to the bottom
                }

                try {
                    Thread.sleep(1000); // Update every second
                }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }).start();
```

- **Explanation of the Code:**
    - A new thread is started that runs an infinite loop (**while (true)**), which continuously checks for new messages.
    - The **hub.load()** method is called to refresh the messages from the data store.
    - It compares the current message count with the last recorded message count (**lastMessageCount**). If new messages are detected, they are appended to the **messageArea** (the text area displaying messages).
    - The **messageArea.setCaretPosition(...)** method ensures that the view scrolls to the bottom to show the latest messages.
    - The thread sleeps for one second (**Thread.sleep(1000)**) to limit the frequency of updates, preventing excessive resource usage while still providing timely updates.
- **Benefits of Multithreading in this Context:**
    - **Responsiveness:** The chat interface remains responsive, allowing users to interact with it (e.g., sending messages) while new messages are being loaded in the background.
    - **Real-time Updates:** Users receive new messages in real-time without needing to refresh or reload the interface manually.

This multithreading implementation is crucial for providing a smooth and interactive user experience in the GlobalHub application.