**Project Report: Global Chat Application – GlobalHub**

Table of Contents

1. Introduction

2. Project Overview

3. Technologies Used

4. System Architecture

5. Key Features

6. Implementation Details

7. User Interface Design

8. Conclusion

9. Appendix

**1. Introduction**

The Global Chat Application, referred to as GlobalHub, is designed to facilitate real-time communication among users through a user-friendly interface. This project aims to provide an efficient platform for users to interact, share messages, and manage their accounts securely.

**2. Project Overview**

ChatHub allows users to register, log in, and participate in chat discussions. The application supports functionalities such as message creation, user management, and the ability to view a list of registered users. The application is built using Java, leveraging the Swing library for the graphical user interface (GUI).

**3. Technologies Used**

• Programming Language: Java

• GUI Framework: Swing

• Serialization: Java Serialization for data persistence

• Development Environment: VS Code

• Version Control: Git

**4. System Architecture**

The architecture of ChatHub 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.

**5. Key Features**

• User Registration: New users can create accounts with validation checks for username, password, email, name, and age.

• User Authentication: Existing users can log in to access the chat hub.

• Real-time Messaging: Users can send and receive messages in real-time.

• User List Display: A feature to view all registered users.

• Logout Functionality: Users can log out securely from their sessions.

**6. Implementation Details**

**6.1 User Registration**

The registration process includes validations to ensure the integrity of user data. Users must provide a username, password, full name, email, and age. The username must be unique and adhere to specific rules (e.g., no spaces, minimum length).

**6.2 User Authentication**

The authentication method checks the provided username and password against stored user data. If valid, the user gains access to the chat interface.

**6.3 Message Handling**

Messages are created and stored using the Message class, which associates each message with the user who sent it. The system supports continuous loading of messages to ensure users can view the latest communications.

**7. User Interface Design**

The user interface is designed to be intuitive and straightforward. Key components include:

• Welcome Screen: Provides options to log in or register.

• Login Dialog: Prompts users for their credentials.

• Chat Hub View: Displays messages and allows users to send new messages.

• User List Dialog: Shows all registered users in a table format.

The layout utilizes GridBagLayout and FlowLayout for responsive design, ensuring a consistent user experience across different screen sizes.

**8. Conclusion**

ChatHub 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.

**9. Appendix**

**9.1 Code Explanation**

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

**9.1.1 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).

**9.1.2 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.

**9.1.3 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.

**9.2 Multithreading in hub\_view()**

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:



* **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 ChatHub application.