**LAN Messenger – Full Project Documentation (JavaFX)**

**🧾 Table of Contents**

1. [Introduction](#introduction)
2. [Project Structure](#project-structure)
3. [Features](#features)
4. [Client-Side Architecture](#client-side-architecture)
5. [Server-Side Architecture](#server-side-architecture)
6. [Networking Protocol](#networking-protocol)
7. [Controllers Overview](#controllers-overview)
8. [Threading & Performance](#threading--performance)
9. [Known Issues & Limitations](#known-issues--limitations)
10. [Future Improvements](#future-improvements)

**🧩 Introduction**

**LAN Messenger** is a JavaFX-based real-time chat application designed for local area networks. It supports user registration, authentication, private and group messaging (with image sharing), profile management, and real-time friend request handling.

**🗂️ Project Structure**

bash

Copy code

src/

├── client/

│ ├── controllers/ # All JavaFX controllers (Login, Signup, Dashboard, etc.)

│ ├── network/ # Singleton ClientConnection

│ ├── model/ # User, Message, Group models

│ ├── utils/ # Session, Database, ImageLoader, etc.

├── server/

│ ├── ServerMain.java # Entry point, starts thread pool and listens

│ ├── ClientHandler.java # Handles each connected client

│ ├── data/ # Chat logs, user data, group data

**✨ Features**

**🔐 Authentication**

* Sign up / Login using username & password.
* Session-based login state and singleton connection.

**👤 User Dashboard**

* View profile photo and username.
* Search users, send/view friend requests.
* Edit profile (photo, name, bio, birthdate).
* Navigate to chatbox or logout.

**👥 Social Features**

* Send/accept/decline friend requests.
* View other users’ profiles.
* Friendship status shown on profile pages.

**💬 Chat**

* Private chats (supports text + images + video calling).
* Group chats (supports text + images).
* Persistent chat history saved per user/group.
* Offline message support (queued and shown on login).

**📷 Profile Editing**

* Upload and preview profile photo.
* Save changes to name, bio, and birthdate.

**🖧 Client-Side Architecture**

**ClientConnection – Singleton Class**

Handles all communication with the server:

* Persistent socket connection.
* Background thread for reading incoming messages.
* Dispatches messages to registered controllers using listener interfaces:
  + MessageListener for chat updates
  + FriendListener for friend events
* Queueing of messages if no listeners are available.
* Serializes messages using a dispatchPool to ensure thread safety.

**Lifecycle:**

* On successful login/signup: connect() opens socket.
* Sends ONLINE|username to the server.
* Starts a listener thread that routes incoming messages using dispatch() and dispatchRequests().

**Session and Database**

* Session: Stores current logged-in user state.
* Database: Loads/saves User object, images, and local metadata.

**🖧 Server-Side Architecture**

**ServerMain**

* Listens on port 12346.
* Uses ExecutorService with fixed thread pool (10 threads).
* Accepts connections and hands each client to a ClientHandler.

**ClientHandler**

* Prepares I/O streams from socket.
* Infinite loop in run() to handle commands from client.
* Uses maps:
  + ONLINE\_WRITERS: username → PrintWriter
  + GROUP\_MEMBERS: group → set of members
* Persists chats in CHAT\_FOLDER and group data in GROUP\_FOLDER.

**Supported Commands:**

* SIGNUP|username|password
* LOGIN|username|password
* PRIVATE|from|to|timestamp|msg
* GROUP|groupName|from|msg
* FRIEND\_REQUEST|from|to
* ACCEPT|from|to, DECLINE|...
* UPLOAD\_PHOTO|username|photoBytes
* GET\_PROFILE|username

**🔌 Networking Protocol**

Custom text-based command protocol:

| **Command** | **Description** |
| --- | --- |
| `SIGNUP | u |
| `LOGIN | u |
| `PRIVATE | a |
| `GROUP | g |
| `FRIEND\_REQUEST | a |
| `ACCEPT | a |
| `DECLINE | a |
| `OFFLINE\_MSG | a |
| `CHAT\_HISTORY | ...` |

**🎮 Controllers Overview**

| **Controller** | **Role** |
| --- | --- |
| WelcomeController | Entry screen navigation to login/signup |
| LoginController | Login form, validates credentials, calls ClientConnection.connect() |
| SignUpController | Sends `SIGNUP |
| DashboardController | Central hub: profile view, search, chat, requests, logout |
| EditProfileController | Uploads photo, name, bio, and birthday |
| FriendRequestsController | View and respond to requests using accept/decline |
| ViewProfileController | View other user info, send friend request |
| ChatBoxController | Private + Group chat UI, loads/stores chat history, shows bubbles |
| GroupChatController | Manages group creation, members, sending group messages |

**🧵 Threading & Performance**

* Server:
  + Fixed thread pool (Executors.newFixedThreadPool(10)) for scalability.
  + Each client gets a ClientHandler thread.
* Client:
  + Background thread for receiving server messages (listen()).
  + Dispatching via ExecutorService ensures serialized message handling.

**Design Benefits:**

* Prevents UI freeze due to blocking I/O.
* Guarantees message order and avoids race conditions.
* Efficient server resource usage.

2

**📘 JavaFX LAN Messenger – Full Documentation**

**📌 Overview**

This is a JavaFX-based **LAN Messenger** that enables real-time communication between users on the same network. It supports:

* User authentication (login/signup)
* Profile editing (photo, name, bio, birthdate)
* Friend system (search, send/accept/decline friend requests)
* Private and group messaging with text and image support
* Video calling (private chats only)
* Persistent chat history and friend data
* Fully multithreaded TCP server with a client connection manager

**🏗️ Project Structure**

**🖥️ Client (JavaFX)**

| **Module/Page** | **Description** |
| --- | --- |
| **WelcomePage.fxml** | Entry screen; navigation to login/signup |
| **LoginController.java** | Handles login logic and error reporting |
| **SignUpController.java** | Sends signup info, creates local user data |
| **Dashboard.fxml** | Main hub after login: shows profile, navigation buttons |
| **EditProfileController** | Allows user to update photo, name, bio, birthdate |
| **ViewProfileController** | View another user's profile; send friend request |
| **FriendRequestsController** | See incoming requests; accept or decline |
| **ChatBoxController** | Displays private and group chat UI, message bubbles, image sending, etc. |
| **VideoCallController** | Manages video call (P2P or simulated) |

**🖧 Server**

| **Component** | **Description** |
| --- | --- |
| **Server.java** | Main entry; starts server on TCP port 12346 using a fixed thread pool |
| **ClientHandler.java** | Handles commands from each connected client via TCP |

**⚙️ Functional Features**

**✅ Login & Signup**

* LOGIN|username|password and SIGNUP|username|password
* Server validates credentials and responds:
  + SUCCESS
  + ERROR|reason
* On success, ClientConnection is initialized and the Dashboard is shown.

**✅ Dashboard Features**

Layout includes:

* Profile photo + username
* Friend search input
* Friend request viewer
* Buttons:
  + Edit Profile
  + Go to Chatbox
  + Logout

**✅ Profile Editing**

Editable fields:

* Name
* Bio
* Date of birth
* Profile image (uploaded and sent to server)

**✅ View Another User's Profile**

* Shows their profile picture and username
* Button: Send friend request

**✅ Friend System**

1. **Search for Friend**
   * Sends SEARCH|username to server
   * Server returns user info if found
2. **Send Friend Request**
   * Sends FRIEND\_REQUEST|from|to
   * Server updates user data and notifies target if online
3. **Accept/Decline Friend Request**
   * Accept: FRIEND\_ACCEPT|from|to
   * Decline: FRIEND\_DECLINE|from|to

**💬 Chat System**

**💌 Private Chat**

* Message format: PRIVATE|from|to|timestamp|message
* Saved in both users’ chat files
* Supports sending text and images

**👥 Group Chat**

* CREATE\_GROUP|groupname|member1,member2,...
* Messages: GROUP|groupname|from|timestamp|message
* Server dispatches message to all group members
* Group chat history is saved to disk

**📷 Image Support**

* Encoded image bytes are sent in custom format:
  + IMAGE\_PRIVATE|from|to|filename|base64bytes
  + IMAGE\_GROUP|from|group|filename|base64bytes
* Images are displayed inline in the chat

**📹 Video Calling (Private Only)**

* Initiated by CALL|from|to
* Peer-to-peer or local simulation via UI (implementation choice)

**🧠 Architecture and Design**

**🧩 Singleton: ClientConnection**

java

Copy code

private static ClientConnection INSTANCE;

Handles:

* Socket connection
* Message sending
* Incoming dispatch to listeners (e.g., chat, friend UI)

Why singleton?

* Ensures one active socket
* Controllers all share this for consistent state

**🔁 Message Listener Dispatching**

* registerListener() → for message events
* registerFriendListener() → for friend events
* dispatch() → routes chat messages
* dispatchRequests() → routes friend actions

All dispatches use:

java

Copy code

private final ExecutorService dispatchPool = Executors.newSingleThreadExecutor();

This ensures that:

* All messages/events are processed **in order**
* No race conditions in UI threads

**🖧 Server Design**

**🔌 Server.java**

* Uses ExecutorService with fixed thread pool (10 threads)
* Accepts connections on port 12346
* Launches ClientHandler for each new connection

**👤 ClientHandler.java**

* Reads and parses lines:
  + LOGIN
  + SIGNUP
  + PRIVATE
  + GROUP
  + FRIEND\_REQUEST
* Each command triggers helper methods and server state changes

**📂 Storage**

**🧍 User Data**

* Stored as users/{username}/profile.ser
* Includes name, bio, birthdate, photo, friends, friend requests

**💬 Chat Logs**

* chats/{username}/{friend}.txt for private
* groups/{groupname}.txt for group

**🖼️ Images**

* Saved in images/ folder
* File references stored in chat logs

**💾 Data Flow Summary**

**▶️ On Startup**

* Welcome.fxml loaded
* On login/signup success, Dashboard is shown
* ClientConnection.connect() maintains a live socket

**💬 On Sending a Message**

* Controller calls ClientConnection.sendPrivateMessage(...)
* Message sent over socket as formatted line
* Server receives → dispatches to target → both sides log it

**📥 On Receiving a Message**

* Server sends line to client
* Client listen() loop reads it
* dispatch() finds relevant controller
* Controller updates chat UI and saves message

**🔐 Concurrency, Threads, and Safety**

**Client:**

* UI: JavaFX Application Thread
* Background:
  + Socket listener thread
  + dispatchPool for ordered message routing

**Server:**

* Thread-per-client (ClientHandler)
* ConcurrentHashMap for shared structures:
  + ONLINE\_WRITERS → user to PrintWriter
  + GROUP\_MEMBERS → group to members

**✅ Summary of Features**

| **Feature** | **Supported** |
| --- | --- |
| Login / Signup | ✅ |
| Profile Edit | ✅ |
| Search + View Profile | ✅ |
| Friend Requests (Send/Accept/Decline) | ✅ |
| Private Chat | ✅ |
| Group Chat | ✅ |
| Image Messaging | ✅ |
| Video Calling (Private) | ✅ |
| Persistent Storage | ✅ |

**🛠️ TODO / Extensions**

* Encrypted message transmission (TLS)
* Online presence indicator in chat
* Group admin/moderator roles
* Message reactions or replies
* Real-time typing indicators
* Notification sounds or toasts