**Chat Application Documentation**It is a client-server chat application that allows users to exchange messages in real time, both publicly and privately. The application stores the message history in a database, supports chat export, notifications and connection management.  
  
**1. Application functionality:**

**1.1 Messaging:**

**Public chat:** Users can send messages that are saved in the database and broadcast to all connected clients.

**Private messages:** Sending private messages via the /pm <username> <message> command is supported. Such messages are saved separately and displayed with a distinctive markup.  
  
**1.2 User Management:**

**Displaying the list of online users:** The client receives updates on the number and names of connected users, and a detailed list is displayed on click.

**Command System:** The client has implemented commands for management (e.g. /who for the list of users, /clear for clearing the screen, /help for help).  
  
**1.3 Working with message history:**

**Storage in the database:** All public messages are stored in the chat\_messages table, and private messages are stored in the private\_messages table using SQLAlchemy and the SQLite database.

**Chat export:** The user can save the message history to a text file using the export function.  
  
**1.4 User Interface (UI):**

**Tkinter GUI:** The client uses a window with a message input field, a chat display area, and a panel with controls.

**Message coloring:** Messages are displayed with different styles and colors depending on their type (system, personal sent/received, and public messages).  
  
**1.5 Connection Management:**

**Automatic Reconnection:** When the connection is lost, the client automatically tries to reconnect.

**Manual Connection Management:** The user can disconnect and reconnect to the server at their own discretion.

**1.6 Additional Features:**

**Sound notifications:** When new messages (especially personal ones) are received, a sound signal is played (winsound function, for the Windows operating system).

**Event logging:** The server maintains detailed logging using the colorlog library for color and informative output.

**2. Requirements for assembling the application into an executable file (.exe)**

**Environment:**Python 3.x (version 3.6 and above recommended).  
  
Operating system: Windows (considering the use of winsound and assembly in .exe).

**Required libraries and packages:**Tkinter: For creating a graphical interface (usually included with Python).  
  
SQLAlchemy: For working with SQLite database.  
  
socket, threading, datetime: Python's standard libraries for networking and multithreading.  
  
winsound: For sound notifications (Windows only).  
  
colorlog: For server side color logging. **Assembly Tool:**PyInstaller: Allows you to compile a Python application into a single executable file (.exe).

**3. Application assembly process  
  
3.1 Install the required packages using pip (In terminal):**  
  
pip install sqlalchemy colorlog pyinstaller

**3.2 Make sure that the project files are in the same directory. This is necessary for the correct operation of the collector.**

**3.3 Open the terminal, in the directory with the source files. And run 2 commands:**

pyinstaller -w --onefile client.py

pyinstaller --onefile server.py

For the rest of the files database, models. You don't need to collect them into an exe file! Since PyInstaller automatically pulls them into server.py, since it uses these files.  
  
**Once completed, a “dist” folder will be created, which will contain ready-to-run .exe files, without dependencies!  
  
4. Internal architecture and mechanisms of the application**

**4.1 General architecture**

**Client part (client.py):**

Implemented in Tkinter, provides a graphical interface.

Uses sockets to connect to the server.

Processes user input and commands (for example, sending messages, personal messages via /pm, commands for displaying help, clearing the chat, etc.).  
  
Works in multithreaded mode: the main thread is responsible for the UI, and a separate thread is responsible for receiving messages from the server.

Applies color marking of messages for ease of perception (system, personal sent, personal received, user messages and public messages).  
  
**4.2 Server part (server.py):**

Runs as a separate process and accepts connections from clients via TCP sockets.

Creates a separate thread for each new client to process its messages.

**Stores dictionaries:**

Client socket <> username.

Username <> timezone offset (for time localization).

Sends the history of public messages from the SQLite database (table chat\_messages) to the client using SQLAlchemy.

**Processes incoming messages:**

Public messages: Stored in the database, formatted with time, and sent to all connected clients.

Private messages: If a message starts with the /pm command, the server fetches the recipient, checks the command for correctness, stores the message in the private\_messages table, and sends it to both the sender and recipient with a distinctive mark.

Sends service messages (e.g. when a user connects/disconnects) and updates information about the current number of online users.

**4.3 Communication and Message Processing Details**

**Session Start:**

When connecting, the client sends a string containing the username and time zone offset.  
  
The server registers the user, stores the offset, and immediately sends the message history to the client.

**Time formatting:**

The server uses UTC and converts timestamps to the client's local time based on the offset passed. This allows the message sending time to be displayed correctly for users in different time zones.

**Database:**The database.py module and models.py model are responsible for connecting to the SQLite database and defining tables:

chat\_messages: stores public messages (username, message text, timestamp).

private\_messages: stores private messages (sender, recipient, message text, timestamp).  
  
**Command processing on the client:**

If a message starts with the / character, the client first tries to process it locally:

The /clear command clears the chat.

The /who command displays a list of currently online users.

The /help command displays help on available commands.

The /pm command is processed by the client as a private message and forwarded to the server for further routing.

**Connection management:**The client has the ability to manually disconnect or restore the connection. If the connection is lost, the automatic reconnection process is launched with an interval (for example, 5 seconds).

When the server detects a connection break, it removes the corresponding socket from the list and informs other users.

**Logging:**

The server outputs detailed information about events (connections, errors, sending messages) using the colorlog library, which simplifies monitoring the application.