Multi-Client Chat in Python – Dokumentation (OOP, Konsoleanwendung)

Autor: Nikolina Nišić-Quinones

Datum: 20.08.2025

E-Mail: nikolinanisic@protonmail.com

1. Überblick

Dieses Projekt implementiert einen Multi-Client-Chat in Python, bei dem mehrere Benutzer über ein lokales Netzwerk miteinander kommunizieren können. Die Anwendung verwendet das Socket-Modul und Threading zur Verwaltung paralleler Verbindungen.

Das Interface der Benutzer befindet sich über die Konsole. Es handelt sich um eine lokale Anwendung, bei der alle Teilnehmer im selben Netzwerk verbunden sind.

2. Programmlogik / Menu-Ablauf

Hauptmenü (in Konsolenvariante):

Nach dem Start des Programms stehen folgende Optionen zur Verfügung:

Optionen:

- ***a)*** *Server starten*
- ***b)*** *Chat starten (Client auswählen)*
- ***q)*** *Programm beenden*

Das Menü ist alphabetisch organisiert und bietet einfache Navigation.

Objektorientierte Struktur (OOP)

1. Server-Klasse

Zweck:

Die Server-Klasse verwaltet mehrere Clients und ermöglicht das Broadcasting von Nachrichten an alle verbundenen Clients.

Wichtige Methoden:

startServer():

Diese Methode akzeptiert eingehende Client-Verbindungen und startet für jede Verbindung einen neuen Thread. Jeder Thread ist dafür zuständig, die Kommunikation mit einem bestimmten Client zu übernehmen.

add_new_client():

Diese Methode verarbeitet Nachrichten eines einzelnen Clients. Sie überwacht die Kommunikation und sorgt dafür, dass der Client bei einer Trennung korrekt aus der Client-Liste entfernt wird.

broadcast_message():

Diese Methode sendet eine Nachricht an alle verbundenen Clients, mit Ausnahme des Absenders. Sie wird verwendet, um Nachrichten an alle anderen Clients im Netzwerk zu verteilen.

2. Client-Klasse

Zweck:

Die Client-Klasse repräsentiert einen einzelnen Benutzer, der Nachrichten an den Server senden und Nachrichten vom Server empfangen kann.

Wichtige Methoden:

send_messages():

Diese Methode erlaubt es dem Nutzer, Nachrichten einzugeben, die an den Server gesendet werden. Zusätzlich kann der Client den Chat mit dem Server durch Eingabe von menu oder exit verlassen.

receive_messages():

Diese Methode hört kontinuierlich auf eingehende Nachrichten vom Server. Sobald eine Nachricht empfangen wird, wird sie sofort in der Konsole des Clients ausgegeben.

3. Datenbank-Klasse

Zweck:

Die Datenbank-Klasse speichert alle gesendeten Nachrichten zusammen mit einem Zeitstempel in einer SQLite-Datenbank.

Wichtige Methoden:

create_table():

Diese Methode erstellt die Tabelle chat_messages in der SQLite-Datenbank, falls diese noch nicht existiert. Die Tabelle speichert den Namen des Absenders, die Nachricht und einen Zeitstempel.

save_messages():

Diese Methode speichert jede empfangene Nachricht zusammen mit dem aktuellen Zeitpunkt in der Datenbank.

get_messages():

Diese Methode ruft alle gespeicherten Nachrichten ab, sortiert nach dem Zeitstempel. Sie ermöglicht das Abrufen und Exportieren aller Nachrichten aus der Datenbank.

4. Menü-Klasse

Zweck:

Die Menü-Klasse bietet ein interaktives Konsolen-Menü, das dem Benutzer ermöglicht, den Server oder den Client zu starten und den Ablauf des Programms zu steuern.

Wichtige Methoden:

main_menu():

Diese Methode zeigt das Hauptmenü an, in dem der Benutzer zwischen den Optionen wählen kann, den Server zu starten, einen Client zu verbinden oder das Programm zu beenden.

start_server():

Diese Methode startet den Server in einem separaten Thread. So kann der Server parallel zum Client laufen und mehrere Verbindungen gleichzeitig akzeptieren.

start_client():

Diese Methode startet einen Client und verbindet ihn mit dem Server. Der Client kann dann Nachrichten senden und empfangen.

close_application():

Diese Methode schließt den Server und beendet alle offenen Client-Verbindungen. Sie sorgt für eine ordnungsgemäße Trennung und beendet die Anwendung.

5. Netzwerkaufbau

Der Server läuft auf einer festen lokalen IP-Adresse (z. B. 127.0.0.1) und einem frei wählbaren Port (z. B. 8095).

Alle Clients verbinden sich über dieselbe IP-Adresse.

Die Kommunikation erfolgt über das TCP-Protokoll, das für eine zuverlässige und geordnete Nachrichtenübertragung sorgt.

Threading wird eingesetzt, um parallele Verbindungen zu ermöglichen, sodass mehrere Clients gleichzeitig mit dem Server kommunizieren können, ohne die Performance des Servers zu beeinträchtigen.

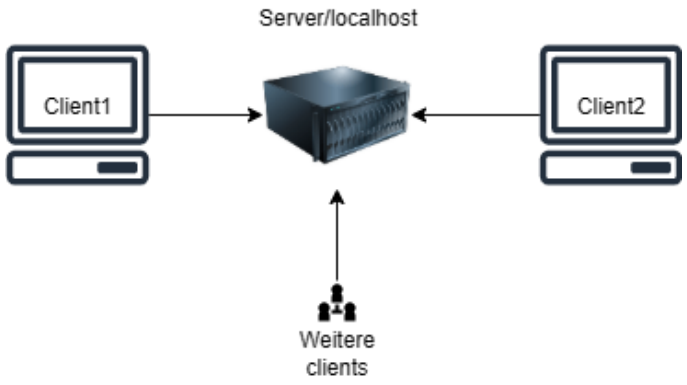

Abbildung 1. Multi-Client Chat

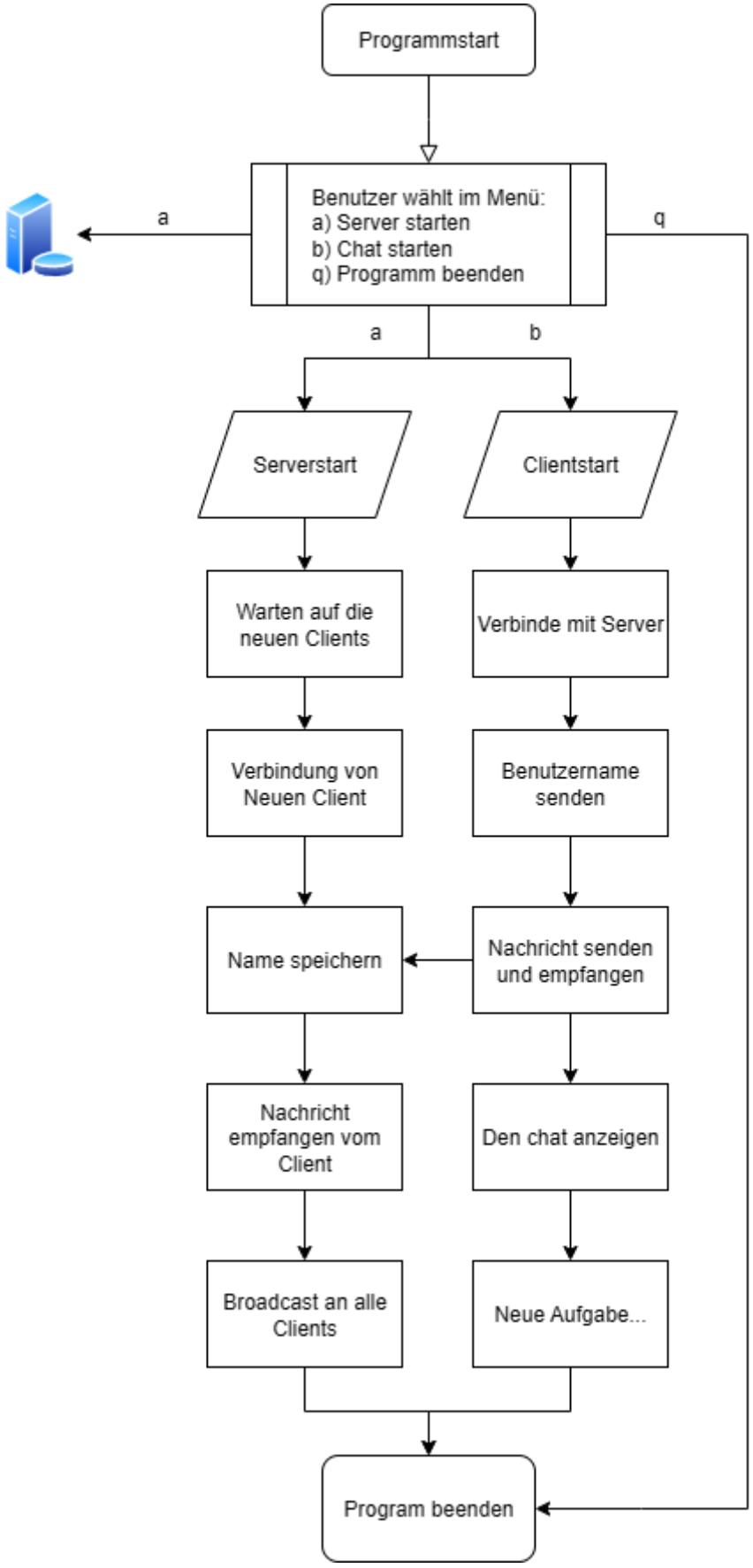

Abbildung 2. Flowchart des Programmes