# Overview

This project is a multi-user chatroom application built using Python, TCP sockets, threading, and SQLite (via SQLAlchemy).
Users can register, log in, and participate in a global chatroom where all messages are broadcast to everyone connected.

The system includes:

- A server that handles networking, authentication, and storing chat history.
- A client that connects over TCP and sends/receives messages.
- Docker support for running both the client and server in containers.

# Features

- User authentication (register + login)
- Password hashing using Werkzeug
- Server-side validation of username and password rules
- Global shared chatroom
- Real-time broadcasting to all connected users
- Message history stored in SQLite
- Polling mechanism to retrieve missed messages
- Full Docker support for the server and client
- Threading-based concurrency

# Running Without Docker

## 1. Install Dependencies

```
pip install sqlalchemy werkzeug
```

## 2. Start the Server

```
python server.py
```

### 3. Start the Client

```
python client.py
```

You can start multiple clients in separate terminals to simulate multiple users.

## Running With Docker

### Build Server Image

```
docker build -t chat-server .
```

### Run Server Container

```
docker run -p 5000:5000 chat-server
```

## How Authentication Works

- User registers with a unique username and a password.
- Server enforces:
    - Username must be at least 3 characters.
    - Passwords must be at least 8 characters.
- Passwords are hashed using Werkzeug before being stored.
- Login verifies the hashed password against stored user credentials.

## How Message Polling Works

- Each message is stored with an auto-increment ID.
- Clients track their `last_message_id`.
- Every second, the client sends:

```
{ "action": "poll", "last_id": <number> }
```

- Server returns all messages with ID greater than `last_id`.
- Client avoids duplicates by comparing message IDs.

This ensures:

- No missed messages
- No repeated messages

## Error Handling

- Invalid commands return error responses.
- Server handles malformed JSON safely.
- Server removes disconnected clients from active list.
- Client gracefully exits on user interrupt (Ctrl+C).
- Database sessions closed properly.

## Credits

**S M Sadid Zarif Prinon**

**Eilia Safarian**

Seneca Polytechnic