💬 One-to-One Chat App using FastAPI + WebSockets

🛞 Goal

We want to build a simple real-time chat system where:

- A user creates a **room code**
- Another user joins using the same code
- Both users can chat in real-time
- Uses WebSockets (not HTTP)

How It Works (Architecture)

Client 1 → WebSocket → FastAPI Server → WebSocket → Client 2

FastAPI keeps track of: - Active rooms - Connected users in each room

We will use: - FastAPI - WebSockets - In-memory storage (dictionary)

♒ Install Dependencies

```
pip install fastapi uvicorn
```

⭕ Project Structure

```
app.py
```

🔹 Step 1: Basic FastAPI Setup

```
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
from typing import Dict, List
```

```
app = FastAPI()
```

🔹 Step 2: Room Manager

We maintain a dictionary:

```
rooms: Dict[str, List[WebSocket]] = {}
```

Structure:

```
{
    "room123": [websocket1, websocket2]
}
```

🔹 Step 3: WebSocket Endpoint

```
@app.websocket("/ws/{room_code}")
async def websocket_endpoint(websocket: WebSocket, room_code: str):
    await websocket.accept()

    # Create room if not exists
    if room_code not in rooms:
        rooms[room_code] = []

    # Allow only 2 users (one-to-one)
    if len(rooms[room_code]) >= 2:
        await websocket.send_text("Room is full")
        await websocket.close()
        return

    rooms[room_code].append(websocket)

    try:
        while True:
            data = await websocket.receive_text()

            # Broadcast to other user in room
            for connection in rooms[room_code]:
                if connection != websocket:
                    await connection.send_text(data)

    except WebSocketDisconnect:
```

```
        rooms[room_code].remove(websocket)

        if len(rooms[room_code]) == 0:
            del rooms[room_code]
```

🚫 Run the Server

```
uvicorn app:app --reload
```

🌐 Simple HTML Client (Testing)

Create a file `chat.html`:

```
<!DOCTYPE html>
<html>
<body>
    <input id="room" placeholder="Enter room code" />
    <button onclick="connect()">Join</button>

    <div id="chat"></div>

    <input id="message" placeholder="Type message" />
    <button onclick="sendMessage()">Send</button>

<script>
let socket;

function connect() {
    const room = document.getElementById("room").value;
    socket = new WebSocket(`ws://localhost:8000/ws/${room}`);

    socket.onmessage = function(event) {
        const chat = document.getElementById("chat");
        chat.innerHTML += "<p>" + event.data + "</p>";
    }
}

function sendMessage() {
    const input = document.getElementById("message");
    socket.send(input.value);
    input.value = "";
}
</script>
```

```
</body>
</html>
```

Open the file in **two different browsers**, use same room code, and chat.

How Read/Write Happens

Action	What Happens
User sends message	websocket.receive_text()
Server processes	Loop through room users
Other user receives	connection.send_text()

🚫 Improvements (Production Ideas)

- Add user names
- Store chat history in database
- Add authentication
- Use Redis for scaling
- Add typing indicator
- Use UUID for secure room codes

🅾️ Scaling Architecture (Important)

Current version is:

✔️ Works for single server ❌ Not scalable (rooms stored in memory)

For production:

- Use Redis Pub/Sub
- Store connections in distributed cache
- Use load balancer

☑️ Summary

We built:

- One-to-one chat
- Room based connection

- Real-time messaging
- WebSocket based architecture

If you want next: - Multi-user group chat - Persistent messages using SQLAlchemy - JWT authenticated chat - Chat app with React frontend - Deploy to Docker

Tell me what level you want 🚀