

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



## MẠNG MÁY TÍNH (CO3093)

---

### Bài tập lớn 1

# *Hiện thực server HTTP và ứng dụng chat*

---

Giảng viên hướng dẫn: Bùi Xuân Giang  
Lớp: L02 - L06

STT	Thành viên nhóm	MSSV	Ghi chú
1	Tạ Gia Bảo	2110795	Nhóm trưởng
2	Đoàn Thị Hương Lan	2311806	
3	Trần Lê Nam Linh	2311875	

Thành phố Hồ Chí Minh, Tháng 11 2025



## Danh sách thành viên và phân công công việc

STT	Thành viên nhóm	MSSV	Nhiệm vụ	Hoàn thành
1	Trần Lê Nam Linh	2311875	Step 6 - 7	100%
2	Đoàn Thị Hương Lan	2311806	Step 4 - 5	100%
3	Tạ Gia Bảo	2110795	Step 1 - 2 - 3	100%



## Mục lục

<b>1</b>	<b>HTTP Server với Cookie Session</b>	<b>3</b>
1.1	Mô tả . . . . .	3
1.2	Yêu cầu . . . . .	3
1.3	Cơ chế xác thực Cookie . . . . .	3
1.3.1	Xử lý đăng nhập . . . . .	3
1.3.2	Truy cập tài nguyên được bảo vệ . . . . .	4
1.4	Thực thi backend . . . . .	4
1.5	Kết quả thực nghiệm . . . . .	4
<b>2</b>	<b>Implement Hybrid Chat Application</b>	<b>5</b>
2.1	Mô tả . . . . .	5
2.2	Yêu cầu . . . . .	5
2.3	Mục tiêu . . . . .	6
2.4	Tổng quan kiến trúc . . . . .	6
2.5	Luồng hoạt động Client-Server . . . . .	7
2.5.1	1. Peer Registration . . . . .	7
2.5.2	2. Peer Discovery . . . . .	7
2.5.3	3. Channel Management . . . . .	7
2.6	Luồng hoạt động Peer-to-Peer . . . . .	7
2.6.1	Peer Node (TCP) . . . . .	7
2.7	REST API của WebApp (WeApRous) . . . . .	8
2.8	Giao diện Web . . . . .	9
2.9	Kết quả thực nghiệm . . . . .	11
2.9.1	Chat P2P . . . . .	11
2.9.2	Broadcast . . . . .	11
2.9.3	Kênh chat (Channels) . . . . .	11
<b>3</b>	<b>Put It All Together</b>	<b>11</b>
3.1	Kiến trúc tổng thể . . . . .	11
3.2	Luồng hoạt động end-to-end . . . . .	12
<b>4</b>	<b>Tổng kết</b>	<b>15</b>

# 1 HTTP Server với Cookie Session

## 1.1 Mô tả

Triển khai cơ chế phiên làm việc (session) dựa trên cookie trong một HTTP server cơ bản sử dụng mô-đun `socket` và `threading` của Python. Mục tiêu là xác thực người dùng thông qua biểu mẫu đăng nhập và duy trì trạng thái đăng nhập bằng HTTP Cookie, từ đó chỉ cho phép truy cập các tài nguyên được bảo vệ (ví dụ: trang chủ `index`) khi người dùng đã được xác thực.

## 1.2 Yêu cầu

- **Task 1A – Authentication Handling:**
  - Khi nhận POST `/login`, server phải kiểm tra thông tin đăng nhập (`username=admin`, `password=password`).
  - Nếu hợp lệ: phản hồi trang `index.html` và gửi kèm header `Set-Cookie: auth=true`.
  - Nếu không hợp lệ: phản hồi mã lỗi 401 `Unauthorized`.
- **Task 1B – Cookie-based Access Control:**
  - Khi nhận GET `/`, server kiểm tra cookie `auth=true`.
  - Nếu cookie hợp lệ: phục vụ nội dung trang `index.html`.
  - Nếu cookie thiếu hoặc sai: trả về 401 `Unauthorized`.
- Hệ thống cần có:
  - **Header Parsing:** phân tích header HTTP.
  - **Session Management:** duy trì phiên người dùng qua cookie.
  - **Concurrency:** hỗ trợ nhiều client cùng lúc (đa luồng).
  - **Error Handling:** xử lý lỗi cơ bản (yêu cầu sai, truy cập trái phép).

## 1.3 Cơ chế xác thực Cookie

### 1.3.1 Xử lý đăng nhập

Khi client gửi:

POST `/login`

Lớp `HttpAdapter` đọc body dạng form:

`username=admin&password=password`

Nếu hợp lệ:

- server trả mã 302 Found
- gửi kèm `Set-Cookie: auth=true`
- chuyển hướng người dùng sang trang `/`

### 1.3.2 Truy cập tài nguyên được bảo vệ

Tại route `/index.html`, server kiểm tra:

- Nếu cookie chứa `auth=true` → trả file HTML thật.
- Nếu không có cookie → trả về trang `401.html` với mã lỗi 401.

## 1.4 Thực thi backend

Backend chạy tại cổng 9000:

```
python start_backend.py --server-ip 0.0.0.0 --server-port 9000
```

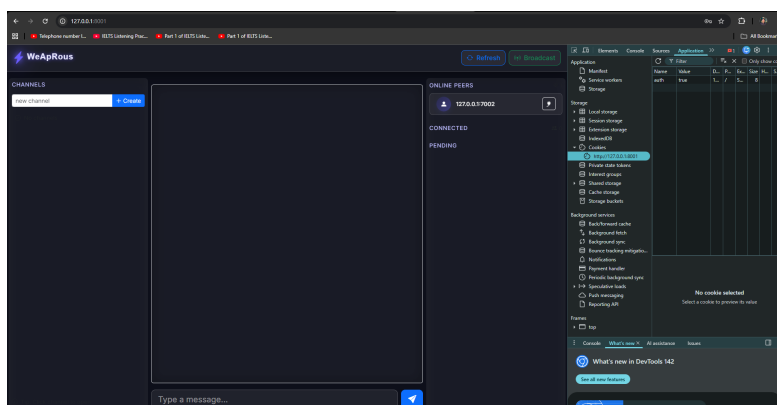
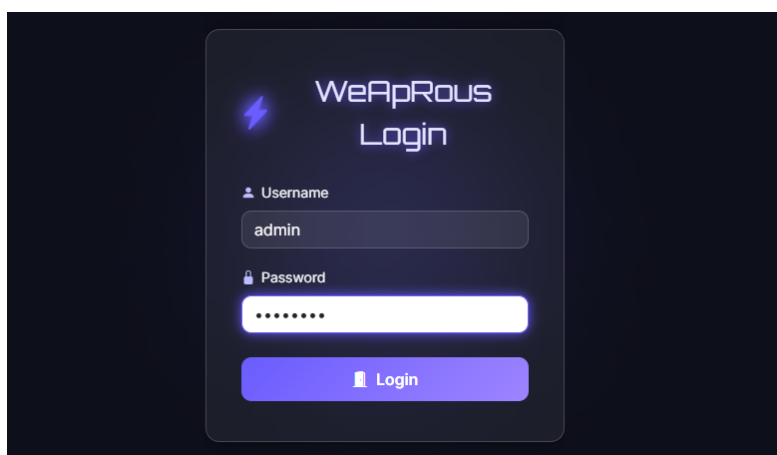
Kết hợp với proxy tại port 8080:

```
python start_proxy.py --server-ip 0.0.0.0 --server-port 8080
```

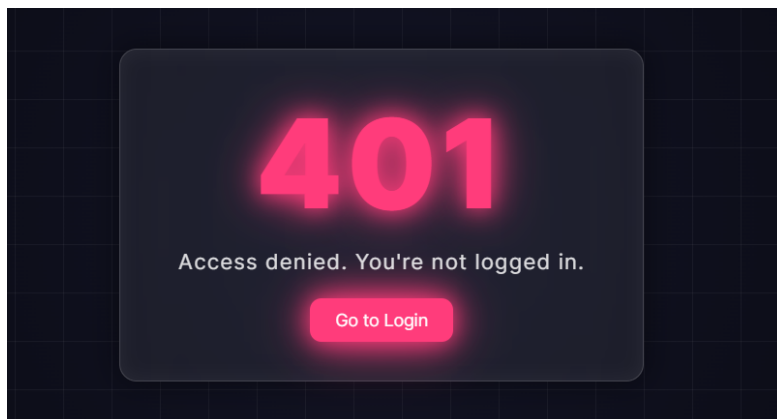
Proxy đọc config/proxy.conf và điều hướng request theo hostname.

## 1.5 Kết quả thực nghiệm

- Đăng nhập thành công → cookie được set và truy cập trang chủ.



- Xoá cookie → bị chặn đúng với 401.



- HTTP server cung cấp file tĩnh (HTML/CSS/JS).

## 2 Implement Hybrid Chat Application

### 2.1 Mô tả

Hiện thực và triển khai ứng dụng chat hybrid kết hợp cả hai mô hình client-server và peer-to-peer. Ứng dụng hỗ trợ giao diện RESTful API sử dụng các phương thức HTTP chuẩn như GET, POST, PUT, DELETE để thao tác tài nguyên (users, peers, messages, channels). Tất cả các endpoint được triển khai bằng framework tự xây dựng WeApRous, hoạt động độc lập không dùng bất kỳ thư viện web nào khác.

### 2.2 Yêu cầu

- **Mô hình Client–Server:**
  - *Peer registration:* Khi một peer mới tham gia, nó phải gửi thông tin IP và port của mình đến server trung tâm qua API `/add-list`.
  - *Tracker update:* Server trung tâm duy trì danh sách các peer đang hoạt động.
  - *Peer discovery:* Các peer có thể yêu cầu danh sách peer hiện tại từ server qua API `/get-list`.
  - *Connection setup:* Các peer sử dụng danh sách này để thiết lập kết nối P2P trực tiếp.
- **Mô hình Peer-to-Peer:**
  - *Broadcast connection:* Một peer có thể gửi tin broadcast đến tất cả các peer đang kết nối.
  - *Direct peer communication:* Các peer trao đổi tin nhắn trực tiếp mà không qua server trong phiên chat đang hoạt động.
- **Quản lý kênh:**
  - *Channel listing:* Người dùng có thể xem danh sách các channel mà họ đã tham gia.

- *Message display*: Mỗi channel có một cửa sổ hiển thị tin nhắn dạng cuộn.
- *Message submission*: Giao diện hỗ trợ nhập và gửi tin nhắn.
- *No edit/delete*: Tin nhắn không thể chỉnh sửa hoặc xóa sau khi gửi.
- *Notification system*: Có thông báo khi có tin nhắn mới.
- *Access control (Optional)*: Channel có thể định nghĩa quyền truy cập riêng.

- **Task requirements:**

- Client-server process programming: Cài đặt tiến trình server/client theo mô hình TCP/IP.
- Protocol design: Thiết kế và triển khai giao thức trao đổi tin nhắn giữa client-server và giữa các peer.
  - \* `http://IP:port/login`
  - \* `http://IP:port/submit-info`
  - \* `http://IP:port/add-list`
  - \* `http://IP:port/get-list`
  - \* `http://IP:port/connect-peer`
  - \* `http://IP:port/broadcast-peer`
  - \* `http://IP:port/send-peer`
- Concurrency: Hỗ trợ nhiều client/peer hoạt động đồng thời.
- Error Handling: Xử lý các lỗi như mất kết nối, định dạng dữ liệu sai, hoặc lỗi socket.

## 2.3 Mục tiêu

Hệ thống chat phải kết hợp cả hai mô hình:

- **Client-Server**: dùng backend tập trung để lưu trữ danh sách peer, quản lý channel và lịch sử tin nhắn.
- **Peer-to-Peer (P2P)**: các peer kết nối trực tiếp qua TCP socket để trò chuyện theo thời gian thực.

## 2.4 Tổng quan kiến trúc

Hệ thống bao gồm 3 thành phần chính:

### 1. Tracker Backend (port 9000)

- Duy trì danh sách peer online trong `db/peers.json`.
- Quản lý danh sách channel và lưu lịch sử tin nhắn.
- Cung cấp API client-server cho webapp.

### 2. Peer Node (TCP server)

- Mỗi peer là một tiến trình độc lập lắng nghe kết nối TCP.
- Hỗ trợ: `connect-request`, `connect-accept`, `message`, `broadcast`.
- Lưu pending requests, danh sách các peer đã kết nối, tin nhắn P2P.

### 3. WebApp – WeApRous

- Cung cấp API REST cho UI.
- Kết nối với tracker backend và với peer node.
- Phục vụ giao diện web cho người dùng.

## 2.5 Luồng hoạt động Client–Server

### 2.5.1 1. Peer Registration

Khi chạy ứng dụng chat qua `start_chatapp.py`, webapp tự gửi yêu cầu đăng ký:

```
POST /submit-info
{
  "ip": <ip_peer>,
  "port": <peer_port>
}
```

Backend ghi nhận vào `db/peers.json`.

### 2.5.2 2. Peer Discovery

Webapp gọi:

```
GET /get-list
```

Tracker kiểm tra xem peer nào còn sống bằng cách thử kết nối TCP tới từng peer. Các peer không còn online sẽ bị loại khỏi danh sách.

### 2.5.3 3. Channel Management

- POST `/create-channel` – tạo channel mới.
- POST `/post-channel` – gửi tin nhắn vào channel.
- POST `/channel-history` – đọc lại lịch sử.

Dữ liệu được lưu tại `db/channels.json`.  
Hệ thống không hỗ trợ sửa/xoá tin nhắn.

## 2.6 Luồng hoạt động Peer–to–Peer

### 2.6.1 Peer Node (TCP)

Mỗi peer chạy:

```
PS C:\Users\Admin\Downloads\C03094-weaprous\C03094-weaprous> python start_chatapp.py --ui-port 8002 --peer-port 7002 --my-ip 127.0.0.1
[PeerNode] Listening on 127.0.0.1:7002
[ChatBackend] Listening on 8002
[ChatApp] Registered to tracker
ChatApp running, click on http://127.0.0.1:7002/login
```

Peer xử lý các loại gói tin TCP sau:

1. **connect-request** Gửi từ peer A  $\rightarrow$  peer B:

```
{
  "action": "connect-request",
  "from": "<A_id>",
  "host": "<A_ip>",
  "port": <A_port>
}
```

$\rightarrow$  Peer B lưu vào `pending_requests`.

2. **connect-accept** Khi B chấp nhận:

```
{
  "action": "connect-accept",
  "from": "<B_id>",
  "host": "<B_ip>",
  "port": <B_port>
}
```

3. **message**

```
{
  "action": "message",
  "from": "<peer_id>",
  "message": "<text>"
}
```

4. **broadcast**

```
{
  "action": "broadcast",
  "from": "<peer_id>",
  "message": "<text>"
}
```

Toàn bộ đều thuần TCP, không có server trung gian.

## 2.7 REST API của WebApp (WeApRous)

Các route được định nghĩa trong `routes.py`. Webapp đóng vai trò trung gian UI giữa trình duyệt và PeerNode/Tracker.

- `/whoami` – trả về ID của peer hiện tại.
- `/get-list` – lấy danh sách peer từ tracker.
- `/connect-peer` – gửi yêu cầu kết nối TCP.
- `/accept-request` – chấp nhận yêu cầu vào peer node.
- `/deny-request` – từ chối kết nối.

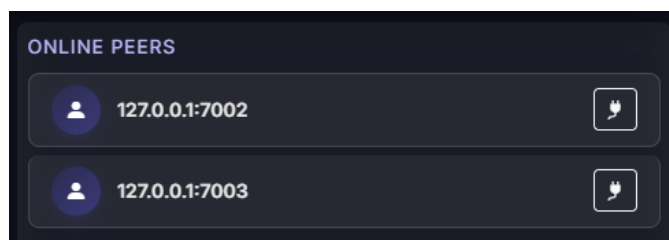
- `/disconnect-peer` – xoá peer khỏi danh sách kết nối.
- `/send-peer` – gửi tin P2P qua TCP.
- `/broadcast-peer` – broadcast tin nhắn qua TCP.
- `/get-pending` – danh sách yêu cầu kết nối đến.
- `/get-connected` – danh sách peer đã kết nối.
- `/get-messages` – log tin nhắn P2P để UI hiển thị.

Tất cả API đều chạy qua `HttpAdapter`.

## 2.8 Giao diện Web

UI nằm trong `www/` và file chính là `chat.js`. Các thành phần giao diện:

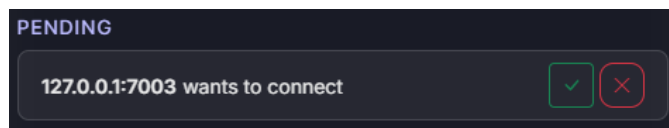
- Danh sách peer đang online (tracker).



- Danh sách peer đã kết nối TCP.



- Danh sách yêu cầu kết nối đến.

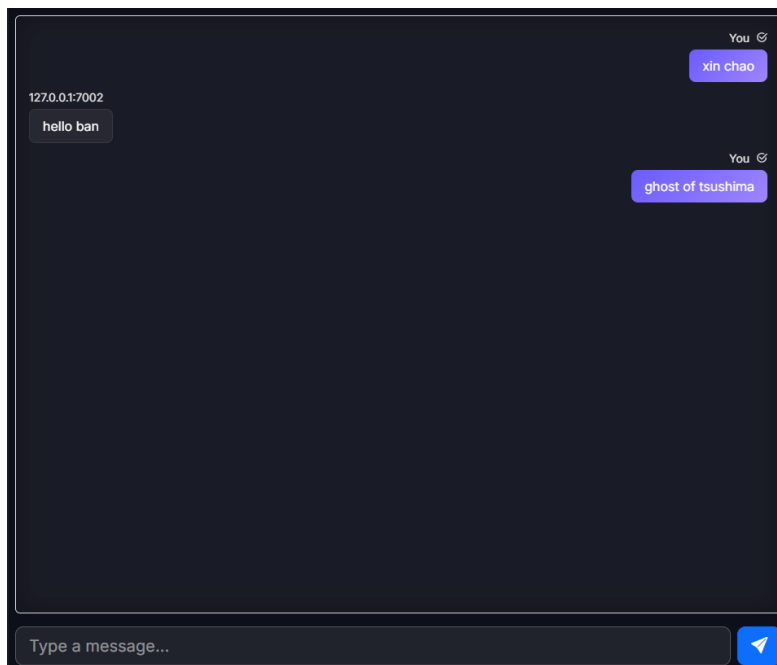


- Notification.

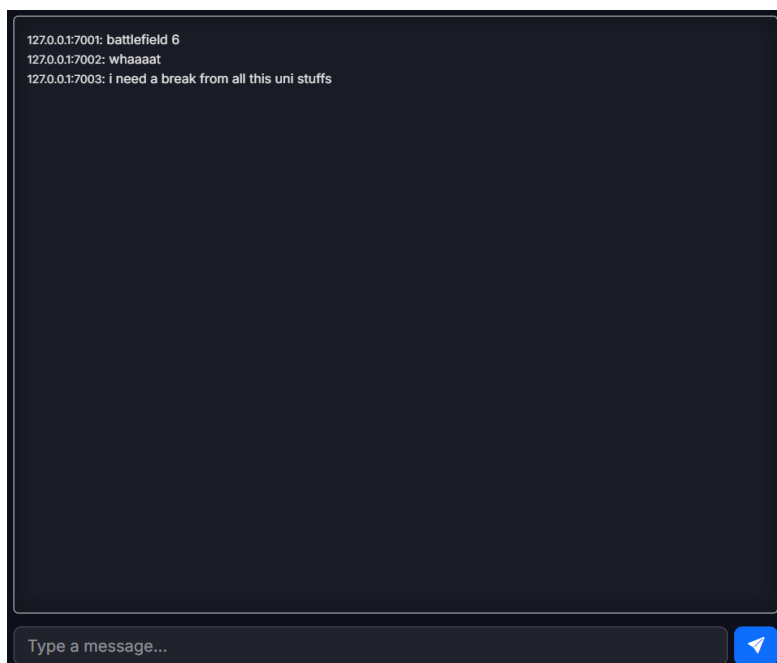




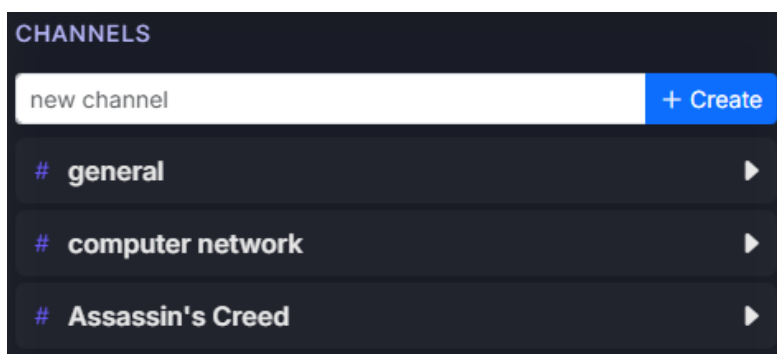
- Cửa sổ chat P2P.



- Cửa sổ chat channel.



- Danh sách các kênh chat.



## 2.9 Kết quả thực nghiệm

### 2.9.1 Chat P2P

- Gửi và nhận tin nhắn trực tiếp giữa hai peer.
- Không đi qua server – chỉ TCP.

### 2.9.2 Broadcast

Tin nhắn broadcast xuất hiện cho tất cả peer:

- UI hiển thị bubble đặc biệt.
- Đồng thời hiện notification.

### 2.9.3 Kênh chat (Channels)

- Tạo channel.
- Gửi tin nhắn.
- Tự động load lịch sử tin nhắn.

## 3 Put It All Together

### 3.1 Kiến trúc tổng thể

Sau khi hoàn thiện từng thành phần (HTTP cookie server, backend tracker, WebApp, peer-to-peer), hệ thống cuối cùng được triển khai dưới dạng nhiều tiến trình độc lập có thể chạy song song:

1. **Proxy Server (port 8080)** Điều hướng request dựa trên cấu hình `config/proxy.conf`. Proxy đóng vai trò lớp front-end có thể mở rộng nếu có nhiều backend.
2. **Backend Tracker (port 9000)** Server trung tâm nhận đăng ký peer, quản lý danh sách peer đang online và quản lý toàn bộ channel.
3. **Peer Node (TCP, mỗi peer một cổng riêng)** Mỗi người dùng chạy một tiến trình TCP server độc lập, cho phép kết nối trực tiếp peer-to-peer.

4. **WebApp UI + REST API** Cung cấp giao diện web, đồng thời là REST controller tương tác với peer node và tracker backend.

Bốn thành phần này tạo thành một hệ thống đầy đủ: **Client truy cập Web UI** → **UI gọi WebApp** → **WebApp gọi Tracker + Peer Node** → **Peer Node giao tiếp TCP trực tiếp**.

### 3.2 Luồng hoạt động end-to-end

Khi hệ thống chạy hoàn chỉnh, các bước hoạt động chính như sau:

#### 1. Khởi động hệ thống

- Chạy proxy:

```
python start_proxy.py --server-ip 0.0.0.0 --server-port 8080
```

- Chạy backend tracker:

```
python start_backend.py --server-ip 0.0.0.0 --server-port 9000
```

- Khởi động ChatApp (UI + Peer Node):

```
python start_chatapp.py --ui-port 8001 --peer-port 7001 --my-ip 127.0.0.1
```

```
python start_chatapp.py --ui-port 8002 --peer-port 7002 --my-ip 127.0.0.1
```

Ngay khi khởi động, WebApp tự động:

- đăng ký peer vào tracker qua `/submit-info`;
- khởi chạy peer node (TCP server);
- khởi chạy backend webapp (UI server);
- in ra URL để người dùng truy cập UI.

#### 2. Đăng nhập vào hệ thống

Người dùng truy cập:

```
http://127.0.0.1:8001/login
```

Lớp `HttpAdapter` kiểm tra thông tin đăng nhập và set cookie `auth=true`. Nếu không có cookie hợp lệ, truy cập trang chủ bị chặn bởi `401.html`.

### 3. Giao diện Web được tải

`index.html` sử dụng JavaScript (`chat.js`) để:

- lấy danh sách peer từ tracker,
- hiển thị danh sách pending request,
- hiển thị peer đã kết nối TCP,
- hiển thị kênh và tin nhắn trong channel,
- hiển thị tin nhắn P2P.

UI làm mới dữ liệu mỗi 3 giây:

```
setInterval(refreshAll, REFRESH_INTERVAL)
```

### 4. Kết nối Peer-to-Peer

Từ danh sách peer online, người dùng bấm “Connect” → WebApp gọi API:

POST `/connect-peer`

WebApp gọi hàm:

```
peer.request_connect(target_ip, target_port, my_id)
```

Peer B nhận gói tin `connect-request` và lưu vào `pending_requests`.

Nếu B bấm “Accept”, WebApp gọi:

POST `/accept-request`

Peer B gửi `connect-accept`, sau đó hai peer được thêm vào danh sách `connected_peers`.

### 5. Gửi tin nhắn P2P

Tại UI:

- Khi người dùng chọn một peer → UI đặt `currentChat.type = "p2p"`.
- Khi gửi tin → gọi API:

POST `/send-peer`

- WebApp gọi hàm:

```
peer.send_message(target_ip, target_port, my_id, msg)
```

- Tin nhắn tới thẳng peer kia qua TCP.

Không có máy chủ trung gian trong luồng P2P — thuần TCP.



## 6. Broadcast đến tất cả peer

Khi người dùng gửi broadcast:

POST /broadcast-peer

WebApp gọi:

```
peer.broadcast(my_id, message)
```

Peer Node gửi tin nhắn tới toàn bộ `connected_peers`. UI hiện bubble broadcast và notification.

## 7. Chat theo kênh (Client-Server)

Các thao tác với channel qua backend tracker:

- tạo channel → /create-channel
- gửi tin → /post-channel
- xem lịch sử → /channel-history

Tin nhắn được lưu trong file JSON, UI tải lại lịch sử mỗi khi chuyển kênh.



## 4 Tổng kết

Weaprous đã được hiện thực các chức năng:

- HTTP Server với cơ chế xác thực bằng Cookie.
- Chat hybrid với hai mô hình Client–Server và Peer–to–Peer.
- Giao diện web quản lý channel.
- Tích hợp toàn bộ các thành phần thông qua kiến trúc nhiều tiến trình (Proxy–Backend–Peer).