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

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 Thực hiện và kết quả

Thực hiện triển khai HTTP server theo các bước sau:

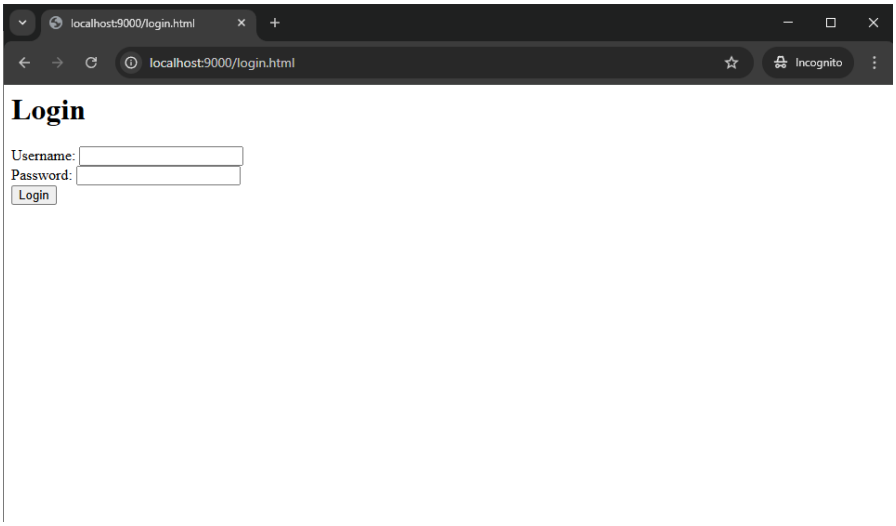
1. Chạy server backend:

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

2. Chạy proxy:

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

3. Mở trình duyệt và truy cập:
   - http://127.0.0.1:9000/login.html hoặc http://127.0.0.1:8080/login.html

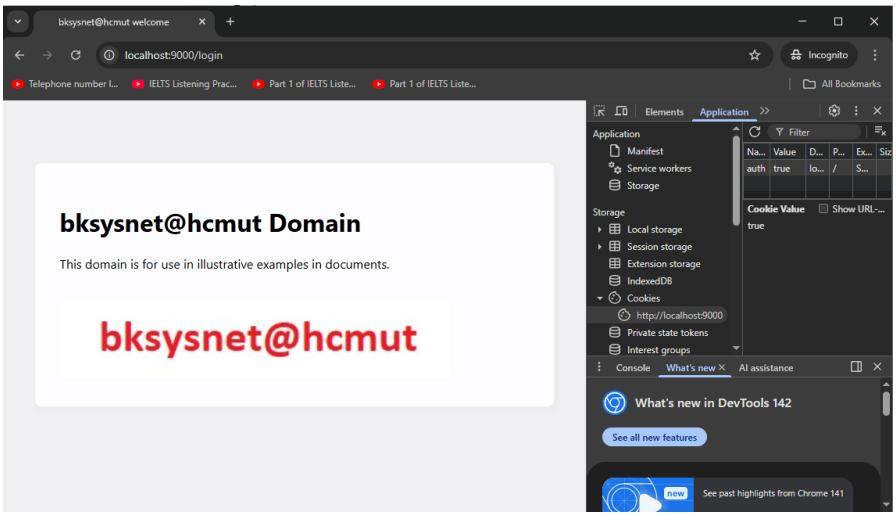


4. Nhập thông tin:

```
username: admin
password: password
```

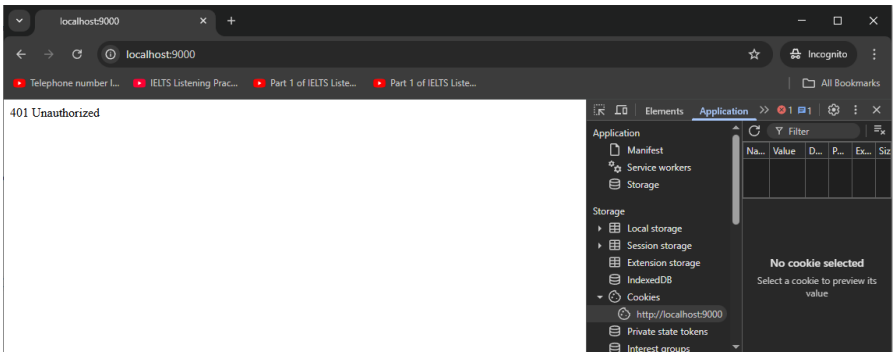
5. Sau khi đăng nhập thành công, server phản hồi:

```
HTTP/1.1 200 OK
Set-Cookie: auth=true
```



6. Khi truy cập lại trang `/`:
   - Nếu có cookie hợp lệ → hiển thị trang `index.html`.

- Nếu xóa cookie hoặc đăng nhập sai → phản hồi `401 Unauthorized`.



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 lí 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 Cấu trúc mã nguồn

Các phần chính của ứng dụng được định nghĩa trong `apps/chatApp.py`. Một route trong mô hình Client–Server:

Listing 1: Route /add-list trong chatApp.py

```
@app.route("/add-list", methods=["POST"])
def add_peer(body):
    peer = json.loads(body)
    added = tracker.add_peer(peer)
    return {"status": "ok" if added else "exists",
            "peers": tracker.get_peers()}
```

Phần peer-to-peer được hiện thực bằng trong `peer_node.py` có các chức năng:

- Đăng ký với tracker qua HTTP.
- Lấy danh sách peers khác và duy trì kết nối TCP lâu dài.
- Gửi và nhận tin nhắn trực tiếp (broadcast hoặc send).

2.4 Thực hiện và kết quả

2.4.1 Client–Server paradigm

1. Khởi động WebApp (tracker):

   ```
   python start_chatapp.py --server-ip 0.0.0.0 --server-port 8000
   ```

2. Thêm peer mới vào tracker:

```
Invoke-RestMethod -Method Post -Uri http://127.0.0.1:8000/add-list ‘
-Headers @{ ’Content-Type’ = ’application/json’ } ‘
-Body ’{"ip":"127.0.0.1","port":9999}’
```

```
PS C:\Users\Admin> Invoke-RestMethod -Method Post -Uri http://127.0.0.1:8000/add-list `
>>   -Headers @{ 'Content-Type' = 'application/json' } `
>>   -Body '{"ip":"127.0.0.1","port":9999}'
>>

status peers
------ -----
ok     {@{ip=127.0.0.1; port=9999}}
```

3. Lấy danh sách peers hiện tại:

```
Invoke-RestMethod -Uri http://127.0.0.1:8000/get-list
```

```
PS C:\Users\Admin> Invoke-RestMethod -Uri http://127.0.0.1:8000/get-list

peers
-----
{@{ip=127.0.0.1; port=9999}}
```

4. Tạo channel mới và gửi tin nhắn:

```
Invoke-RestMethod -Method Post ‘
   -Uri http://127.0.0.1:8000/create-channel ‘
   -Headers @{ ’Content-Type’=’application/json’ } ‘
   -Body ’{"channel":"general"}’
Invoke-RestMethod -Method Post ‘
   -Uri http://127.0.0.1:8000/send-channel ‘
   -Headers @{ ’Content-Type’=’application/json’ } ‘
   -Body ’{"channel":"general","sender":"alice","message":"hello"}’
```

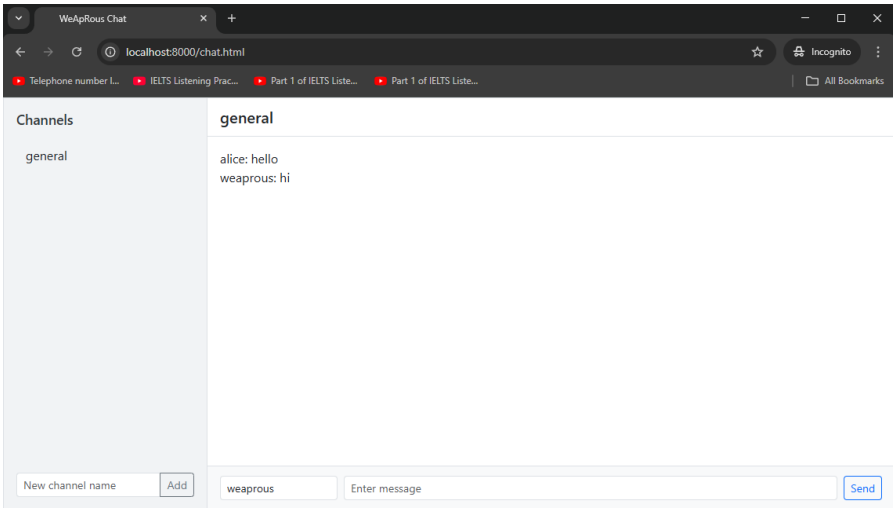
```
PS C:\Users\Admin> Invoke-RestMethod -Method Post `
>>       -Uri http://127.0.0.1:8000/create-channel `
>>       -Headers @{ 'Content-Type'='application/json' } `
>>       -Body '{"channel":"general"}'

status message
------ -------
ok     Joined channel general


PS C:\Users\Admin> Invoke-RestMethod -Method Post `
>>       -Uri http://127.0.0.1:8000/send-channel `
>>       -Headers @{ 'Content-Type'='application/json' } `
>>       -Body '{"channel":"general","sender":"alice","message":"hello"}'

status message
------ -------
ok     Message sent
```

5. Mở trình duyệt truy cập:
   - http://127.0.0.1:8000/chat.html
6. Gửi và nhận tin nhắn giữa nhiều tab hoặc trình duyệt khác nhau trên giao diện web.

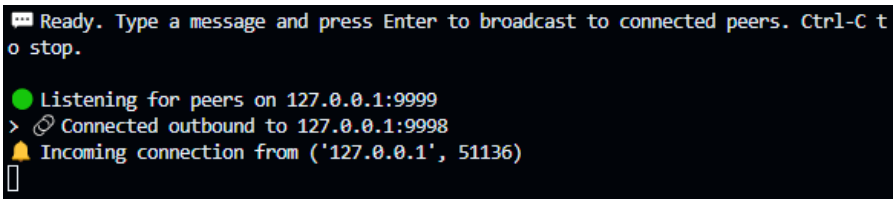


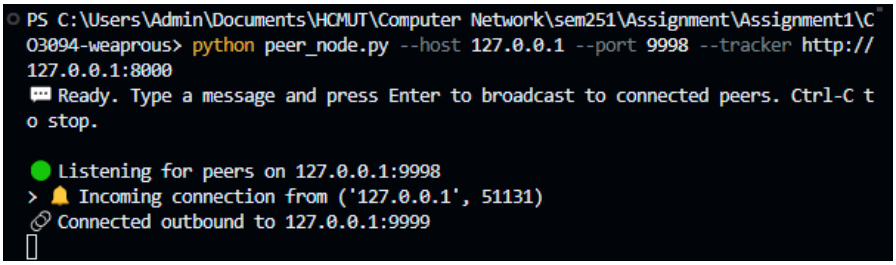
2.4.2 Peer-to-Peer paradigm

1. Mở hai terminal:

```
# Peer A
python peer_node.py --host 127.0.0.1 --port 9999 --tracker
http://127.0.0.1:8000
# Peer B
python peer_node.py --host 127.0.0.1 --port 9998 --tracker
http://127.0.0.1:8000
```

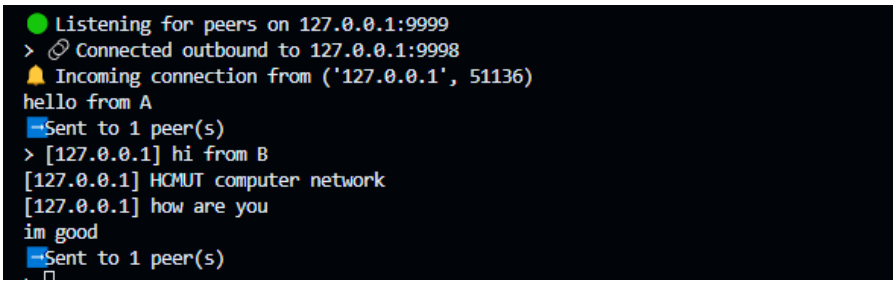
2. Quan sát console:
   - Cả hai peer tự động đăng ký với tracker.



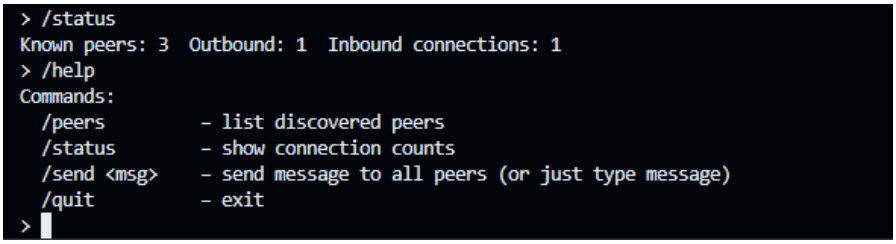


- Tự động phát hiện nhau và kết nối TCP trực tiếp.

3. Gửi tin nhắn:



4. Các lệnh trên CLI:



3 Put It All Together

Kết hợp hai phần chính: (1) HTTP server với cookie session và (2) Ứng dụng chat hybrid thành một hệ thống hoàn chỉnh.

Kiến trúc tổng thể:

- Proxy: vai trò trung gian, nhận yêu cầu HTTP từ client và chuyển tiếp đến backend hoặc webapp phù hợp theo file cấu hình `config/proxy.conf`. Proxy cho phép mở rộng nhiều backend và phân tải (*round-robin*).
- Backend / WebApp Process (WeApRous): xử lý các route RESTful như `/login`, `/add-list`, `/get-list`, `/create-channel`, và cung cấp các trang HTML (`index.html`, `chat.html`). Đồng thời, WebApp đóng vai trò **tracker** quản lý danh sách các peer đang hoạt động.
- Peer Node: các peer đăng ký với tracker, duy trì kết nối TCP trực tiếp và trao đổi tin nhắn P2P trong giai đoạn chat.

Luồng hoạt động tổng quát:

1. Người dùng truy cập giao diện qua `Proxy → Backend`.
2. Thực hiện đăng nhập bằng cookie session (HTTP Server).
3. Sau khi đăng nhập, người dùng vào giao diện `chat.html` để tạo channel, gửi tin nhắn hoặc đăng ký peer mới.
4. Các peer được tracker (WebApp) ghi nhận và tự động thiết lập kết nối trực tiếp với nhau, thực hiện truyền tin qua mô hình P2P.

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).