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

Lớp TN01 - Nhóm 8386

Bài tập lớn 1

Mạng máy tính (CO3094)

"Implement HTTP server and chat application"

Giảng viên hướng dẫn: Hoàng Lê Hải Thanh

Sinh viên thực hiện: Lê Nguyễn Kim Khôi - 2311671
Nguyễn Công Minh - 2312080
Bùi Ngọc Phúc - 2312665
Lê Trọng Thiện - 2313233
Phạm Trần Minh Trí - 2313622

THÀNH PHỐ HỒ CHÍ MINH, 10/2025

Mục lục

Danh sách hình ảnh

Danh sách bảng

Danh sách thành viên

nào xong sửa sau

STT	Họ và tên	MSSV	Phân công	% hoàn thành
1	Lê Nguyễn Kim Khôi	2311671	- abc - xyz	100%
2	Nguyễn Công Minh	2312080	- -	100%
3	Bùi Ngọc Phúc	2312665	- -	100%
4	Lê Trọng Thiện	2313233	- -	100%
5	Phạm Trần Minh Trí	2313622	- -	100%

Bảng 1: Danh sách thành viên và phân công

1 Mở đầu

2 HTTP server with cookie session

2.1 Implement authentication handling

Hệ thống WeApRous thực hiện việc xác thực thông qua sự phối hợp của 4 tệp: start_app.py (định nghĩa logic), httpadapter.py (điều phối), request.py (xử lý body), và response.py (tạo phản hồi).

2.1.1 Routing & Validation

- Tệp `start_app.py` định nghĩa route: `@app.route('/login', methods=['POST'])` và liên kết nó với hàm `login_post`.
- `request.py` (trong `prepare_body`) phân tích cú pháp `body` của yêu cầu POST (đã được `httpadapter.py` truyền vào) để lấy `username` và `password`.
- Hàm `login_post` thực hiện kiểm tra logic: `if (username == "admin" and password == "password") ...`. Điều này đáp ứng yêu cầu xác thực.

2.1.2 Invalid

- Nếu `if` thất bại, hàm `login_post` trả về một từ điển: `{"auth": "false"}`.
- `httpadapter.py` (trong `handle_client`) nhận kết quả này (`hook_result`). Nó kiểm tra `if hook_result["auth"] == "false":`.
- Nó ngay lập tức gọi `response = resp.build_unauthorized()`.
- Hàm `build_unauthorized` trong `response.py` tạo ra một trang phản hồi `HTTP/1.1 401 Unauthorized` hoàn chỉnh.
- **Kết luận:** Yêu cầu "401 Unauthorized" được đáp ứng chính xác.

2.1.3 Valid

- Nếu `if` thành công, hàm `login_post` tạo một `session_id` và trả về một từ điển phức tạp hơn: `{"auth": "true", "redirect": "/", "session_id": session_id}`.
- `httpadapter.py` nhận kết quả này. Nó **không** trả về trang `index` ngay lập tức. Thay vào đó, nó phát hiện khóa `"redirect"`.
- Nó gọi hàm `response = resp.build_redirect(redirect, req, new_session_id)`.
- Hàm `build_redirect` trong `response.py` tạo ra một phản hồi `HTTP/1.1 302 Found` (chuyển hướng).
- **Thiết lập Cookie:** Quan trọng nhất, bên trong `build_redirect`, nó chèn các header `Set-Cookie` vào phản hồi:
  - Set-Cookie: auth=true; Path=/
  - Set-Cookie: session_id=...; Path=/
- **Kết luận:** Yêu cầu được đáp ứng. Máy chủ đặt cookie `auth=true` thành công. Thay vì trực tiếp phục vụ trang `index`, nó sử dụng một thông báo chuyển hướng (Redirect), điều này khiến trình duyệt của máy khách tự động thực hiện một yêu cầu `GET /` mới, và yêu cầu này sẽ được xử lý bởi logic của Task 1B.

2.2 Implement cookie-based access control

2.2.1 Routing

Tệp `start_app.py` định nghĩa route: `@app.route('/', methods=['GET'])` và liên kết nó với hàm `index`.

2.2.2 Cookie Check

- Hành động đầu tiên của hàm `index` là gọi `if not authenticate(headers): return {"auth": "false"}`. Hàm `authenticate` (trong `start_app.py`) là cơ chế kiểm soát truy cập.
- **Phân tích** `authenticate(headers)`**:**
  - Nó không đọc trực tiếp header `Cookie:` Thay vào đó, nó dựa vào `request.py`.
  - Khi yêu cầu `GET /` đến, `request.prepare` được gọi trước. `request.prepare` gọi `prepare_cookies`.
  - `prepare_cookies` đọc chuỗi cookie thô (ví dụ: `"auth=true; session_id=123"`) và phân tích nó thành một từ điển Python: `{'auth': 'true', 'session_id': '123'}`. Từ điển này được lưu trong `headers["cookie-pair"]`.
  - Hàm `authenticate` sau đó đọc từ điển đã được xử lý này: `cookie = headers.get("cookie-pair", None)`.
  - Nó kiểm tra cụ thể: `auth = cookie.get("auth", "")` và sau đó kiểm tra `if auth == "true" and session_id in session_to_account`.

2.2.3 Invalid cookie

- Nếu `authenticate` trả về `False` (do `cookie` không tồn tại, `auth` không phải là `"true"`, hoặc `session_id` không hợp lệ), hàm `index` sẽ trả về `{"auth": "false"}`.
- `httpadapter.py` bắt được giá trị này và nó gọi `resp.build_unauthorized()` để trả về `401 Unauthorized`.
- **Kết luận:** Yêu cầu được đáp ứng chính xác.

2.2.4 Valid cookie

- Nếu `authenticate` trả về `True`, hàm `index` tiếp tục thực thi.
- Nó trả về một từ điển: `{"auth": "true", "content": "index.html", "placeholder": ...}`.
- `httpadapter.py` nhận kết quả này. Nó thấy khóa `"content"` (và `"placeholder"`).
- Nó gọi `response = resp.build_content_placeholder(req, content, placeholder)`.
- Hàm này trong `response.py` sẽ tải tệp `index.html` từ thư mục `www/`, thay thế bất kỳ nội dung động nào (nếu có), và xây dựng một phản hồi `HTTP/1.1 200 OK` hoàn chỉnh với nội dung trang.
- **Kết luận:** Yêu cầu "phục vụ trang index" được đáp ứng.

3 Hybrid chat application

Thiết kế này áp dụng nguyên tắc phân tách trách nhiệm (Separation of Concerns) rõ ràng giữa hai thành phần chính:

- Web Server (start_app.py): Đóng vai trò là "người điều phối". Server chịu trách nhiệm quản lý việc xác thực (authentication) người dùng, phân quyền (authorization), tạo channel, quản lý thành viên (ai được tham gia) và lưu trữ mật khẩu (dưới dạng hash).
- P2P Client (start_p2p.py): Chịu trách nhiệm xử lý toàn bộ logic nhắn tin (messaging). Sau khi được Web Server "giới thiệu" và cung cấp danh sách thành viên, P2P Client sẽ thực hiện broadcast tin nhắn trực tiếp đến các thành viên khác trong channel mà không cần thông qua server.

3.1 Chatting between 2 peer

3.2 Channel management

Tính năng Channel Chat được thiết kế để cho phép nhiều người dùng tham gia vào một phòng chat chung , song song với tính năng chat 1-1 hiện có. Mỗi channel được xác định duy nhất bằng tên (channel name) và được bảo vệ bằng mật khẩu để kiểm soát truy cập.

Luồng hoạt động chi tiết như sau:

- Tạo Channel mới - **@app.route("/create-channel", methods=["POST"])**: Người dùng khởi tạo một phòng chat mới.
  1. Đầu tiên, User cung cấp channel_name và password qua form.
  2. Tiếp theo Server kiểm tra channel_name có tồn tại trong hệ thống chưa. Nếu đã tồn tại thì báo lỗi cho người dùng.
  3. Nếu chưa thì tuần tự Server thực hiện hash(password) để lưu trữ an toàn, sau đó tạo một mục mới trong cấu trúc dữ liệu channels và cuối cùng thêm địa chỉ của người dùng vào danh sách thành viên đầu tiên.
- Xem danh sách và Tham gia Channel - **@app.route("/channel", methods=["GET"])** và **@app.route("/join-channel", methods=["POST"])**: Cho phép người dùng thấy các channel có sẵn và tham gia chúng. Với xem luồng danh sách thì khi truy cập channel, server hiển thị 2 danh sách bao gồm joined channels - Các channel mà người dùng đã là thành viên; và available channels - Các channel khác mà người dùng có thể tham gia. Còn đối với luồng tham gia:
  1. User chọn một channel từ "Available channels" và nhập password.
  2. Server lấy password người dùng nhập, thực hiện hash và so sánh với stored_password_hash của channel đó.
  3. Nếu hash(password) == stored_password_hash, thêm địa chỉ của user (user_address) vào address_list của channel kèm thông báo tham gia thành công. Nếu sai, báo lỗi mật khẩu không chính xác.
- Kết nối vào Channel Chat (Handoff to P2P) - **@app.route("/connect-channel", methods=["POST"])**: Khởi tạo phiên chat P2P sau khi người dùng đã được xác thực là thành viên.

1. User nhấp vào "Connect" trên một channel đã tham gia.
2. Web Server lấy address_list (danh sách địa chỉ IP/Port) của tất cả thành viên hiện tại trong channel đó.
3. Server thực hiện một POST redirect (chuyển hướng POST) sang ứng dụng P2P Client (start_p2p.py), mang theo address_list này.
4. P2P Client nhận được danh sách, lưu vào channel_members[channel_name] và thực hiện broadcast một thông báo "join" đến tất cả các thành viên trong danh sách.

- Chat trong Channel: Luồng gửi và nhận tin nhắn P2P. Đây là luồng gửi tin nhắn:

  1. User gõ tin nhắn trong giao diện (UI) của channel.
  2. P2P App (của người gửi) xác định channel hiện tại (channel_name).
  3. App lấy danh sách thành viên từ channel_members[channel_name].
  4. App thực hiện broadcast (gửi đồng thời) tin nhắn đến tất cả địa chỉ trong danh sách đó.

  Còn đây là luồng nhận tin nhắn:

  1. P2P Client (của người nhận) lắng nghe và nhận được một tin nhắn.
  2. Client kiểm tra định dạng tin nhắn.
  3. Client phân tích (parse) tin nhắn để lấy sender_address, timestamp, channel_name, và message.
  4. Client xác định đây là tin nhắn của channel_name và hiển thị nội dung message lên UI của channel tương ứng.

Để cụ thể hơn về định dạng tin nhắn được nhắc ở trên thì cần phân tích sâu hơn ở đặc điểm chính là để phân biệt tin nhắn. Để phân biệt tin nhắn 1-1 và tin nhắn channel, một định dạng prefix đặc biệt được sử dụng ở trong này. Client P2P sẽ dựa vào prefix này để xử lý tin nhắn đúng cách.

Định dạng này được định nghĩa như sau: [sender_address] [timestamp] [Channel]: channel_name message. Tiền tố [Channel]channel_name là dấu hiệu nhận biết (flag) cho P2P Client rằng đây là tin nhắn thuộc về một channel cụ thể.

4 Kết luận

Tài liệu tham khảo

[1] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. 8th. Boston, MA: Pearson, 2021. ISBN: 978-0136681557.

[2] PEP Editors. *Python Enhancement Proposals (PEPs) — PEP Index*. https://peps.python.org/. Accessed: 2025-10-12.