ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

MẠNG MÁY TÍNH (CO3093)

Báo cáo bài tập lớn 1

GVHD: Hoàng Lê Hải Thanh
Sinh viên: Võ Văn Hiếu - 2252219
Võ Lý Đắc Duy - 2252125
Phan Thảo Vy - 2252930
Đỗ Trường Khoa - 2252345
Lớp: CN01 - Nhóm 2

TP. HỒ CHÍ MINH, THÁNG 11 NĂM 2024

Mục lục

1. Thành viên nhóm

STT	Họ và tên	MSSV	Nhiệm vụ	Tiến độ hoàn thành
01	Võ Lý Đắc Duy	2252125	Code Backend, Client	100%
02	Phan Thảo Vy	2252930	Code Upload, vẽ diagram	100%
03	Võ Văn Hiếu	2252219	Code Download, viết báo cáo	100%
04	Đỗ Trường Khoa	2252345	Code Frontend, Tracker	100%

2. Mô tả chức năng

2.1. Tracker

Về chức năng, tracker là "trạm" dùng để hoàn thành các tác vụ như lưu trữ thông tin của các peer. Những thông tin gồm các file được lưu trữ dưới dạng vị trí của các peers đang nắm giữ file đó, và làm công tác phản hồi liên quan đến việc seeder đăng kí thông tin của nó, leecher yêu cầu danh sách các peer. Bên cạnh đó cũng phản hồi cho việc thay đổi thông tin.

Check torrent info

- Kiểm tra và cập nhật danh sách torrent hiện tại, nhằm đảm bảo rằng thông tin về mỗi torrent đều được ghi nhận và lưu trữ một cách đầy đủ.
- Điều kiện: khi nhận được yêu cầu từ client, hệ thống sẽ lấy thông tin info hash từ yêu cầu đó để tiến hành kiểm tra.
- Hoạt động: hệ thống sẽ đối chiếu info hash từ client với danh sách các torrent đã có. Nếu info hash chưa tồn tại, tức đây là một torrent mới, hệ thống sẽ bổ sung thông tin này vào danh sách torrent để lưu trữ thông tin chi tiết về torrent đó. Sau khi hoàn tất việc kiểm tra và cập nhật danh sách torrent, hệ thống sẽ tiếp tục thực hiện quy trình kiểm tra thông tin client bằng cách gọi hàm `CHECK_CLIENT_INFO()` nhằm xử lý thông tin của peer mong muốn kết nối.

Check client info

- Kiểm tra và cập nhật danh sách các peer, giúp hệ thống nhận diện và quản lý thông tin của từng peer mong muốn kết nối đến tracker qua một tệp torrent cụ thể.
- Điều kiện: khi hệ thống nhận thông tin từ một peer muốn kết nối, bao gồm info hash và các thông tin như ip, port..., hàm sẽ được kích hoạt để tiến hành xử lý.
- Hoạt động: hệ thống sẽ kiểm tra danh sách các peer đã đăng ký trước đó với info hash cụ thể. Nếu đây là một peer mới, chưa có trong danh sách, hệ thống sẽ lưu trữ thông tin của peer, bao gồm địa chỉ IP, port, và các thuộc tính khác. Nếu peer đã tồn tại trong danh sách, tức là đã từng đăng ký với tracker trước đây, hệ thống sẽ cập nhật lại các thông tin như thời gian cuối cùng kết nối (lastseen), trạng thái kết nối (status), và các thông tin khác cần thiết để đảm bảo dữ liệu luôn chính xác và mới nhất.

Announce Response

- Xử lý và phản hồi các yêu cầu kết nối của các peer, nhằm giúp duy trì sự liên lạc và cập nhật thông tin giữa các peer và tracker.
- Điều kiện: khi nhận được yêu cầu `ANNOUNCE_TO_TRACKER()` từ peer, hàm này sẽ được gọi để thực hiện phản hồi phù hợp dựa trên mục đích của peer gửi yêu cầu.
- Hoạt động: hệ thống sẽ bắt lấy và phân loại các tín hiệu yêu cầu từ các peer dựa trên vai trò của chúng (Seeder, Leecher, hay Peer yêu cầu cập nhật):

- Seeder muốn đăng ký: hệ thống sẽ thực hiện các hàm `CHECK_TORRENT_INFO()` và `CHECK_CLIENT_INFO()` để kiểm tra và cập nhật thông tin của torrent và client. Sau khi đăng ký thành công, hệ thống sẽ gửi phản hồi xác nhận đã hoàn tất quá trình đăng ký cho Seeder.
- Leecher muốn tải xuống: hệ thống sẽ gọi hàm `CHECK_TORRENT_INFO()` để kiểm tra thông tin torrent, sau đó lấy danh sách các peer đang chia sẻ tệp tin này và chuyển danh sách đó sang dạng nhị phân để gửi lại cho Leecher. Phản hồi giúp Leecher kết nối với các peer khác để tải xuống dữ liệu cần thiết.
- Peer gửi yêu cầu cập nhật thông tin: đối với peer chỉ muốn cập nhật thông tin, hệ thống sẽ thực hiện các hàm `CHECK_TORRENT_INFO()` và `CHECK_CLIENT_INFO()` để kiểm tra và ghi nhận các thay đổi từ peer. Sau khi cập nhật thành công, hệ thống sẽ gửi phản hồi xác nhận quá trình thay đổi đã hoàn tất.

2.2. Client

2.2.1. Connection

Announce to tracker

- Peer kết nối với tracker để thực yêu cầu.
- Điều kiện: khi client kết nối với tracker.
- Hoạt động: gửi các yêu cầu kết nối khác nhau như: đăng ký seeder, leecher tải xuống, cập nhật thông tin.

Connect to peer

- Sau khi nhận được danh sách peer từ tracker, peer sẽ chọn một hoặc nhiều peer để kết nối. Thông thường, peer sẽ chọn ngẫu nhiên từ danh sách hoặc dựa trên một số tiêu chí nhất định (như tốc độ tải lên/tải xuống).
- Điều kiện: đã có danh sách peer và lựa chon peer để kết nối.
- Hoạt động: Peer sử dụng socket để thiết lập kết nối TCP với peer khác. Kết nối này thường sử dụng địa chỉ IP và cổng mà peer đã nhận từ danh sách peer.

Sau khi quá trình kết nối thành công, các peer có thể bắt đầu trao đổi dữ liệu.

2.2.2. Download

Upload torrent file

- Khi một leecher muốn tải một tài liệu dựa trên tệp torrent, nó sẽ thực hiện tải tệp .torrent từ thiết bị lên client. Tệp này chứa "info hash" (một mã định danh duy nhất của tệp), danh sách các phần (piece) của tệp cùng kích thước mỗi phần, và địa chỉ của các tracker.
- Điều kiện: đã có tệp .torrent trên thiết bị.
- Hoạt động: client sử dụng địa chỉ tracker trong tệp .torrent để kết nối với tracker bằng ANNOUNCE_TO_TRACKER() để thông báo sẽ trở thành leecher. Đồng thời, tracker gửi lại một danh sách các peers khả dụng mà leecher có thể kết nối để tải tệp.

Download flow

- Chuẩn bị Danh sách Công việc: Lấy danh sách các mảnh dữ liệu (pieces) cần tải và xác định danh sách các peers có thể tải dữ liệu.
- Tải dữ liệu từ nhiều nguồn:
  - Chọn một piece cần tải.
  - Kết nối với một peer để tải piece này về. Nếu tải thất bại (do lỗi mạng hoặc dữ liệu không khớp), chuyển sang peer khác để thử lại.

    - Mỗi piece có một giới hạn số lần thử, đảm bảo không lặp lại vô hạn nếu có lỗi.
- Xác minh Dữ liệu Tải về:
    - Sau khi tải một piece, tính toán mã băm và so sánh với mã băm gốc từ file torrent để xác minh dữ liệu có chính xác hay không.
    - Nếu dữ liệu hợp lệ, piece sẽ được đánh dấu hoàn thành và lưu trữ. Nếu không, thử tải lại từ peer khác.
- Xử lý Khi Tải Thất Bại:
    - Nếu không tải thành công sau khi thử từ nhiều peers, piece đó sẽ bị đánh dấu thất bại và không tiếp tục thử nữa.
- Lặp Lại cho Đến Khi Hoàn Thành:
    - Quá trình tiếp tục cho đến khi toàn bộ các pieces trong danh sách đã được tải thành công hoặc đạt giới hạn thử thất bại.

Create handshake
- Tạo thông điệp handshake và gửi cho seeder, một bước quan trọng để thiết lập kết nối giữa hai peer.
- Điều kiện: đã thiết lập kết nối bằng socket.
- Hoạt động: khởi tạo các thông tin handshake và gửi qua socket. Sau đó đợi phản hồi HANDSHAKE_RESPONSE() từ seeder.

Interested
- Gửi thông báo rằng leecher sẽ tiến hành trao đổi thông tin với seeder.
- Điều kiện: nhận được HANDSHAKE_RESPONSE() từ seeder.
- Hoạt động: kiểm tra thông tin từ phản hồi của seeder. Nếu thông tin đúng gửi thông báo INTERESTED(). Nếu không đúng thông báo lỗi.

Request Piece
- Gửi yêu cầu tải mảnh.
- Điều kiện: nhận được thông báo UNCHOKE() và BITFIELD() từ seeder.
- Hoạt động: dựa vào thông tin trong bitfield để gửi yêu cầu tải mảnh cần thiết.

Handle Piece
- Xử lý dữ liệu của mảnh nhận được từ seeder.
- Điều kiện: nhận được SEND_PIECE() từ seeder.
- Hoạt động: xử lý và kiểm tra với hash.

Handle File
- Kết nối và lưu tệp về máy.
- Điều kiện: Đã tải tất cả các mảnh cần thiết.
- Hoạt động: tổng hợp các mảnh lại thành một file và kiểm tra. Nếu file hợp lệ sẽ tiến hành tải về máy.

2.2.3. Upload

Chức năng upload là chức năng tải file lên để tạo torrent và thông báo cho tracker, trở thành seeder giúp chia sẻ tài nguyên với những người dùng khác trong hệ thống. Để thực hiện việc này thì trải qua các giai đoạn:

Generate torrent file
- Tạo file .torrent cho các tài liệu cần chia sẻ.
- Điều kiện: Người dùng chọn mục “Upload file”.

- Hoạt động: khi người dùng tải file lên client sẽ tạo file .torrent và lưu vào trong bộ nhớ của thiết bị, đồng thời lúc này client sẽ trở thành seeder thông báo với tracker bằng ANNOUNCE_TO_TRACKER() để tiến hành seeding.

Handshake response
- Thiết lập kết nối giữa seeder và leecher bằng cách phản hồi yêu cầu handshake. Điều này đảm bảo rằng cả hai bên đều có thể giao tiếp và xác nhận thông tin cần thiết.
- Điều kiện: Đã kết nối với leecher bằng socket và nhận được HANDSHAKE_REQUEST().
- Hoạt động: Kiểm tra thông tin từ yêu cầu của leecher. Nếu thông tin đúng sẽ gửi lại một handshake cho leecher. Nếu không đúng thông báo lỗi.

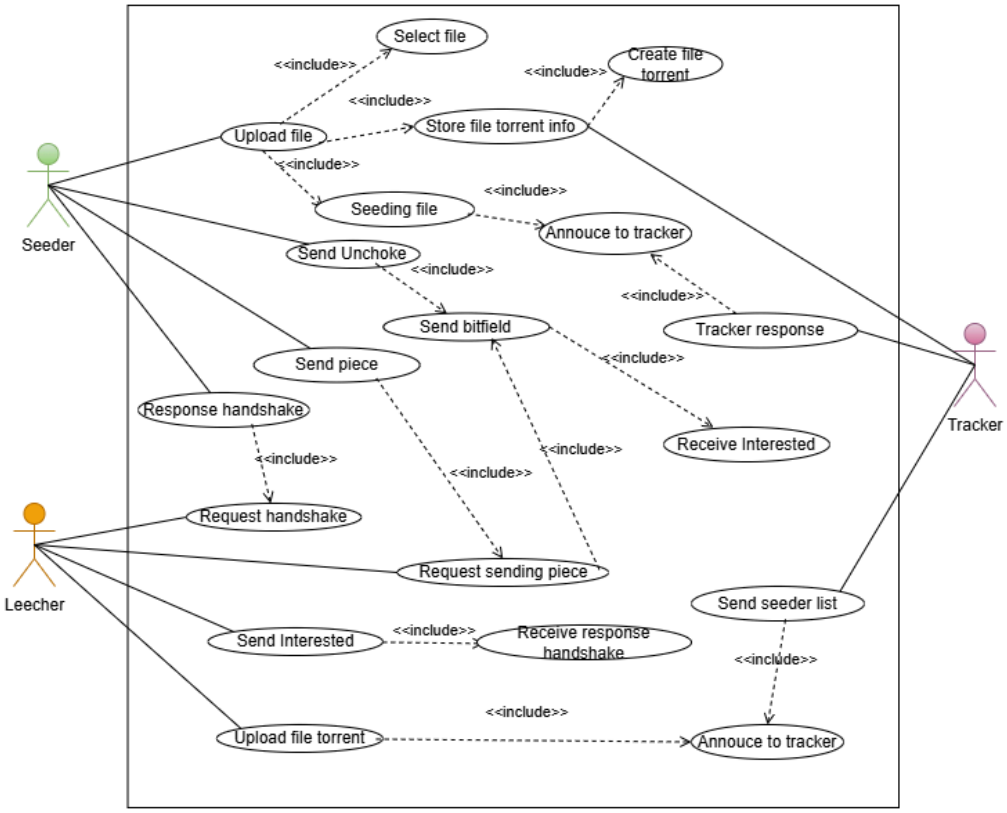
Send Bitfield
- Sau khi hoàn tất quá trình handshake và xác nhận rằng seeder đã thiết lập kết nối với leecher, seeder cần gửi thông điệp bitfield để thông báo về trạng thái của các phần (pieces) mà nó đang sở hữu. Dưới đây là quy trình chi tiết
- Điều hiện: sau khi đã handshake và nhận được thông báo INTERESTED().
- Hoạt động: tạo thông điệp bitfield và gửi qua socket.

Send Unchoke
- Sau khi hoàn tất quá trình handshake và nhận được thông điệp bitfield từ leecher, seeder cần gửi thông điệp "unchoke" để thông báo rằng nó đã sẵn sàng cung cấp dữ liệu cho leecher.
- Điều kiện: sau khi đã handshake và đã gửi bitfield.
- Hoạt động: tạo thông điệp unchoke và gửi qua socket.

Send piece
- Khi một leecher yêu cầu một mảnh dữ liệu cụ thể từ seeder, seeder cần gửi mảnh đó nếu nó có.
- Điều kiện: khi leecher yêu cầu một mảnh nào đó.
- Hoạt động: khi nhận được yêu cầu seeder sẽ tạo dữ liệu mảnh đó và gửi dữ liệu bằng socket.

3. Sơ đồ

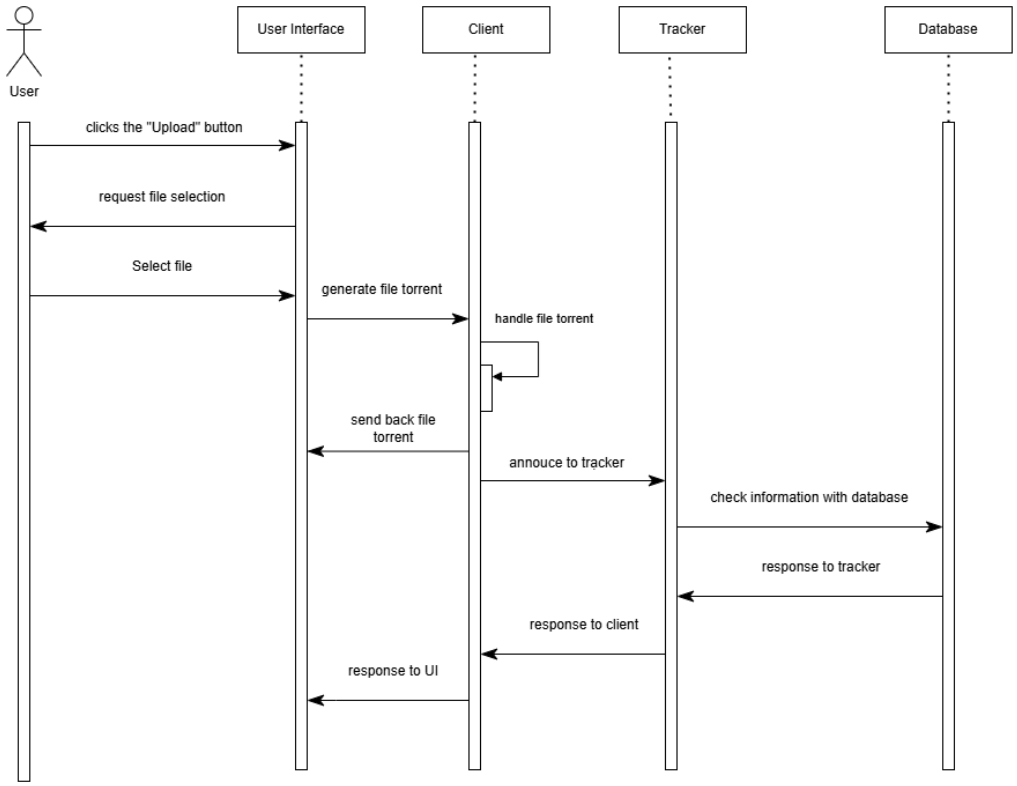
3.1. Use case diagram cho toàn bộ hệ thống



Mô tả: Sơ đồ này mô tả quy trình trao đổi tệp tin giữa seeder và leecher:

- Seeder: Người dùng tải lên tệp tin và chia sẻ nó với người khác. Quá trình bắt đầu khi Seeder chọn tệp tin, tạo file torrent và thông báo cho Tracker.
- Leecher: Người dùng tải về tệp tin từ các Seeder. Leecher gửi yêu cầu tải về tệp tin cho Tracker, sau đó tải về tệp tin từ các Seeder và gửi thông tin về quá trình tải về cho Tracker.
- Tracker: Quản lý và điều phối việc trao đổi tệp tin giữa các người dùng. Tracker kiểm tra thông tin tệp tin với cơ sở dữ liệu, sau đó cung cấp thông tin về các Seeder đang chia sẻ tệp tin cho Leecher.
- Quy trình cụ thể bao gồm các bước như lựa chọn tệp tin, tạo file torrent, thông báo cho Tracker, yêu cầu tải về, tải về từ Seeder, gửi thông tin về quá trình tải về, quản lý danh sách Seeder, và các tương tác khác giữa các thành phần.

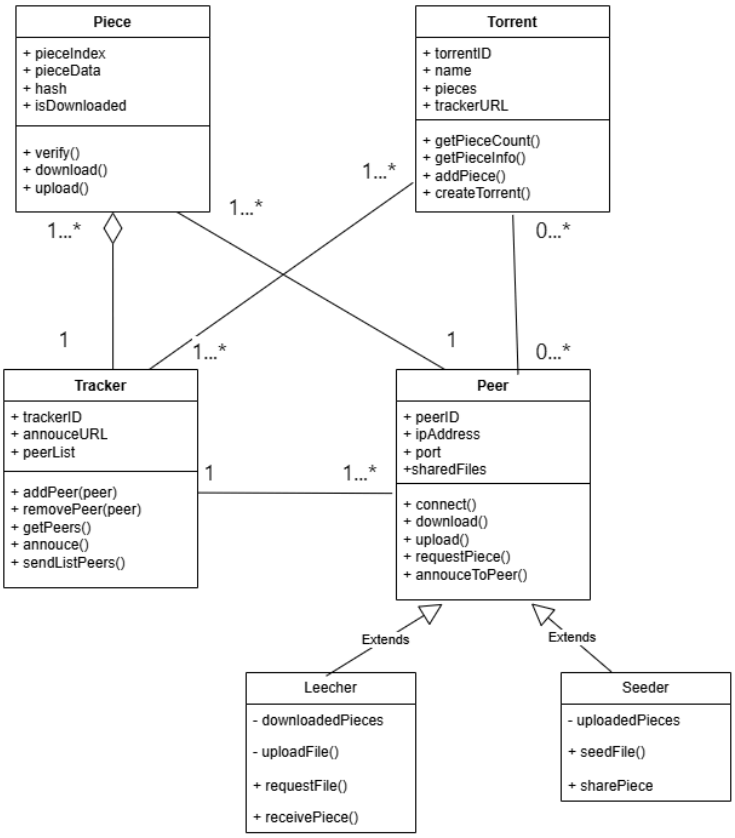
3.2. Sequence diagram cho upload



Mô tả: Sequence diagram mô tả các tương tác giữa các thành phần trong quá trình tải lên (upload) file trong một hệ thống P2P với kiến trúc Client-Tracker. Các bước diễn ra như sau:

- Người dùng (User) nhấn nút "Upload" trên giao diện người dùng (User Interface).
- Giao diện người dùng yêu cầu người dùng chọn file để tải lên.
- Sau khi chọn file, giao diện người dùng tạo ra file torrent.
- Client tiếp nhận file torrent và gửi lại cho giao diện người dùng.
- Client sau đó thông báo cho Tracker về file được tải lên.
- Tracker kiểm tra thông tin file với Database.
- Database trả lời Tracker về thông tin file.
- Tracker gửi phản hồi lại cho Client.
- Cuối cùng, Client gửi phản hồi về giao diện người dùng, và hiển thị kết quả cho người dùng.

3.3. Class diagram



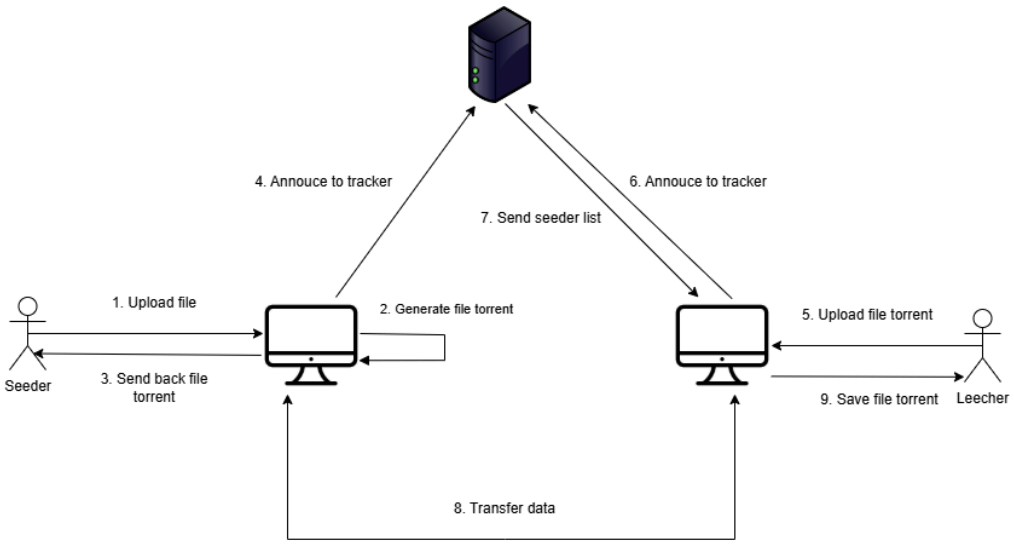
Mô tả: Đây là sơ đồ lớp mô tả các thành phần và mối quan hệ giữa chúng trong một ứng dụng chia sẻ tập tin ngang hàng. Các lớp chính bao gồm:

- Piece: Biểu diễn một phần của tập tin có thể được tải xuống. Nó có các thuộc tính như pieceIndex, pieceData, hash và isDownloaded.
- Torrent: Biểu diễn một tập tin torrent chứa thông tin về một tập tin được chia sẻ. Nó có các thuộc tính như torrentID, name, pieces và trackURL.
- Tracker: Chịu trách nhiệm quản lý các kết nối giữa các đối tượng Peer và cung cấp thông tin về các phần tập tin có sẵn. Nó có các thuộc tính như trackerID, announceURL và peerList.
- Peer: Biểu diễn một đối tượng ngang hàng trong mạng. Nó có các thuộc tính như peerID, ipAddress, port và sharedFiles.
- Leecher: Mở rộng lớp Peer và biểu diễn một đối tượng ngang hàng đang tải về tập tin. Nó có thêm các thuộc tính như downloadedPieces và uploadedFiles.
- Seeder: Mở rộng lớp Peer và biểu diễn một đối tượng ngang hàng đang tải lên tập tin. Nó có thêm một thuộc tính uploadedPieces.

Trong sơ đồ lớp này, các mối quan hệ multiplicity được mô tả như sau:

- Piece và Torrent: Mỗi Torrent chứa nhiều Piece.
- Torrent và Tracker: Mỗi Torrent có liên kết với 1 Tracker.
- Tracker và Peer: Mỗi Tracker quản lý nhiều Peer
- Peer và Piece: Mỗi Peer có thể chia sẻ nhiều Piece.
- Peer và Leecher/Seeder: Mỗi Peer có thể là một Leecher hoặc Seeder.
- Leecher/Seeder và Piece: Mỗi Leecher có thể tải về nhiều Piece. Mỗi Seeder có thể chia sẻ nhiều Piece.

3.4. Kiến trúc hệ thống



Mô tả: Đây là một mô tả kiến trúc hệ thống các tương tác trong một ứng dụng chia sẻ tập tin ngang hàng (peer-to-peer file sharing). Các bước chính trong quy trình này bao gồm:

- Seeder (người chia sẻ) tải lên tập tin torrent.
- Seeder tạo ra một tập tin torrent từ tập tin được chia sẻ.
- Seeder gửi lại tập tin torrent đã tạo.
- Seeder thông báo cho Tracker về tập tin torrent.
- Leecher (người tải về) tải lên tập tin torrent.
- Leecher thông báo cho Tracker về tập tin torrent.
- Tracker gửi danh sách Seeder cho Leecher.
- Dữ liệu được truyền giữa Seeder và Leecher.
- Leecher lưu tập tin torrent đã tải về.

4. Thực hiện

4.1. Tracker

Trong tệp `tracker.py` được sử dụng các thư viện bao gồm

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import urllib.parse
import bencodepy
import time
```

```
import hashlib
import signal
import sys
```

- `http.server`: Dùng để tạo máy chủ HTTP.
- `urllib.parse`: Xử lý và phân tích URL.
- `bencodepy`: Mã hóa và giải mã dữ liệu bằng Bencode (chuẩn BitTorrent).
- `time`: Xử lý thời gian (timestamp).
- `hashlib`: Tạo hash (dùng SHA-1)
- `signal`: Bắt tín hiệu hệ thống
- `sys`: Thao tác với hệ thống

Đầu tiên tạo dictionary nhằm lưu trữ các danh sách torrent, tạo mã hash SHA-1 giả định cho một torrent mẫu và khởi tạo danh sách peer rỗng cho torrent mẫu.

```
torrents = {}
infoHash = hashlib.sha1(b"fake_torrent_data").hexdigest()
torrents[infoHash] = []
```

Class TrackerHandler

Class `TrackerHandler` kế thừa lớp `BaseHTTPRequestHandler` để xử lý yêu cầu HTTP. Bên trong lớp gồm có các hàm sau:

- Hàm `do_GET` phân tích URL yêu cầu, nếu đường dẫn là `/announce` thì gọi hàm `AnnounceResponse` để xử lý. Nếu không thì trả về lỗi "404 Not found".

```
def do_GET(self):
    parsed_url = urllib.parse.urlparse(self.path)
    if parsed_url.path == '/announce':
        self.AnnounceResponse(parsed_url)
        print(torrents)
    else:
        self.send_error(404, "Not Found")
```

- Hàm `AnnounceResponse` phân tích tham số truy vấn từ URL (`info_hash` , `peer_id` , `port` ,), sau đó lấy địa chỉ IP của client `self.client_address[0]` nhằm xác định sự kiện từ client và loại kết nối

```
def AnnounceResponse(self, parsed_url):
    query = urllib.parse.parse_qs(parsed_url.query)
    info_hash = query.get('info_hash', [None])[0]
    peer_id = query.get('peer_id', [None])[0]
    port = query.get('port', [None])[0]
    ip = self.client_address[0]
    event = query.get('event', [None])[0]
    type = query.get('type', [None])[0]
```

Nếu type là leecher thì tracker trả về danh sách peer ở dạng compact. Sau đó gọi `CheckClientInfo` để cập nhật thông tin peer và phản hồi bao gồm khoảng thời gian cập nhật cũng như danh sách peer.

```
if info_hash:
    if type == 'leecher':
        peer_list = self.get_peers(info_hash)
        compact_peer_list = bytes(self.compact_peers(peer_list))
        self.CheckClientInfo(info_hash, peer_id, ip, port)
        response = {
            b'interval': 1800,
            b'peers': compact_peer_list,
        }
```

Sau đó gửi phản hồi HTTP thành công với dữ liệu được mã hóa bằng Bencode

```
self.send_response(200)
self.send_header('Content-Type', 'application/x-bittorrent')
self.end_headers()
self.wfile.write(bencodepy.encode(response))
```

- **CHECK_CLIENT_INFO**

Nếu torrent không tồn tại , tạo một danh sách rỗng. Nếu peer tồn tại, cập nhật thông tin và nếu là peer mới thì thêm vào danh sách.

```
def CheckClientInfo(self, info_hash, peer_id, ip, port):
    if info_hash not in torrents:
        torrents[info_hash] = []
    existing_peer = next((peer for peer in torrents[info_hash] if
peer['peer_id'] == peer_id), None)
    if existing_peer:
        existing_peer.update({'ip': ip, 'port': int(port), 'last_seen':
time.time()})
    else:
        torrents[info_hash].append({'peer_id': peer_id, 'ip': ip, 'port':
int(port), 'last_seen': time.time()})
```

- **CHECK_TORRENT_INFO**

Kiểm tra xem torrent (xác định bởi info_hash) đã tồn tại trong danh sách torrents chưa. Nếu chưa tồn tại thì tạo một mục mới trong danh sách torrents với `info_hash` làm khóa. Giá trị tương ứng là một danh sách rỗng (`[]`), nơi sẽ lưu danh sách các peer liên quan đến torrent đó.

```
def CheckTorrentInfo(self, info_hash):
    if info_hash not in torrents:
        torrents[info_hash] = []
```

- Hàm `get_peers` Lấy danh sách các peer (client) liên quan đến một torrent được xác định bởi `info_hash`. Nếu torrent không tồn tại, trả về danh sách rỗng (`[]`)

```
def get_peers(self, info_hash):
    return torrents.get(info_hash, [])
```

- Hàm `compact_peers` Chuyển danh sách các peer từ định dạng dictionary (JSON-like) sang định dạng compact. Mỗi peer được biểu diễn bằng 6 byte: 4 byte đầu là địa chỉ IP (IPv4) và 2 byte sau là cổng (port) của peer.
- Hàm khởi động máy chủ: Khởi động máy chủ HTTP trên cổng 8080 với lớp xử lý TrackerHandler. Đăng ký trình xử lý tín hiệu để tắt máy chủ khi nhấn Ctrl+C

```
with HTTPServer(("", PORT), TrackerHandler) as httpd:
    def signal_handler(sig, frame):
        print('Shutting down server...')
        httpd.server_close()
        sys.exit(0)
    signal.signal(signal.SIGINT, signal_handler)
    print(f"BitTorrent tracker running on port {PORT}")
    httpd.serve_forever()
```

4.2. Client

4.2.1. Connection

Để tiến hành trao đổi thông tin giữa 2 phía Leecher và Seeder thì `ANNOUNCE_TO_TRACKER()` và `SOCKET_CONECTION()` là 2 hàm chức năng quan trọng không thể thiếu.

ANNOUNCE_TO_TRACKER

Giữa leecher và seeder đều cần `ANNOUNCE_TO_TRACKER()` để thông báo tới tracker nhưng tồn tại một số nhỏ chi tiết khác nhau trong lúc thực hiện.

Ở phía leecher, hàm nhận vào các giá trị `tracker_url: str`, `info_hash`, `file_length: int`, `peer_id: bytes`, `type`, `port=6060` tiến hành chuẩn hóa, xây dựng chuỗi tham số `params` bằng cách tạo một object chứa đầy đủ thông tin cần thiết của file torrent để gửi tới tracker với mục tiêu được trả về danh sách thông tin của các peer seeder.

```
params = urllib.parse.urlencode({
        'info_hash': info_hash.hex(), #bytes.fromhex(info_hash),
        'peer_id': peer_id,
        'port': port,
```

```
        'uploaded': 0,
        'downloaded': 0,
        'left': file_length,
        'compact': 1,
        'event': 'started',
        'type': type,
    })
```

Sau đó gửi yêu cầu HTTP đến Tracker gồm tracker_url và chuỗi tham số (sử dụng aiohttp để gửi yêu cầu HTTP GET).

```
url = f"{tracker_url}?{params}"

async with aiohttp.ClientSession() as session:
    async with session.get(url) as response:
```

Sau khi gửi thì cần đợi phản hồi từ tracker và xử lí: kiểm tra trạng thái HTTP, nếu không phải 200, thông báo lỗi.

```
if response.status != 200:
    raise ValueError(f"Tracker error: {response.status}")
```

Dữ liệu từ tracker được giải mã bằng bencodepy (chuẩn mã hóa trong giao thức BitTorrent).

```
data = await response.read()
decoded_response = bencodepy.decode(data)
```

Nếu phản hồi chứa failure reason, tracker đã báo lỗi.

```
if b'failure reason' in decoded_response:
    raise ValueError(decoded_response[b'failure reason'].decode())
```

Ở chế độ compact, tracker trả về danh sách peer dưới dạng một chuỗi byte: 6 byte cho mỗi peer: 4 byte đầu là địa chỉ IP, 2 byte cuối là port. Địa chỉ IP được tách thành 4 phần, mỗi phần tương ứng 1 byte. Port được tính từ hai byte cuối bằng cách dịch trái byte đầu 8 bit và cộng byte sau.

```
peers = decoded_response[b'peers']
return [
    {
        'ip': f"{peers[i]}.{peers[i + 1]}.{peers[i + 2]}.{peers[i + 3]}",
        'port': (peers[i + 4] << 8) + peers[i + 5],
    }
    for i in range(0, len(peers), 6)
     if f"{peers[i]}.{peers[i + 1]}.{peers[i + 2]}.{peers[i + 3]}" != ""
#public_ip
]
```

Mỗi peer được biểu diễn dưới dạng một dictionary chứa: `ip`, `port` và loại trừ peer có cùng IP với peer hiện tại (public_ip).

Về phía Seeder, khi nó thông báo tới tracker rằng mình muốn seed một file bất kì, cung cấp cho leecher khác những thông tin của nó để có thể tiến hành download file dễ dàng hơn.

Hàm nhận đầu vào `announce_to_tracker(torrent_data, port, peer_id, type)` lưu ý trong torrent_data có nghĩa là cần các tham số khác trong nó (info, announce,...)

Đầu tiên dựa vào tham số `info` bên trong `torrent_data` để tính toán `info_hash` (hash SHA-1 của thông tin tệp tin)

```
info_hash = calculate_info_hash(torrent_data['info'])
```

Lấy `url` của tracker cần gửi đến, là trường `announce` trong `torrent_data`

```
tracker_url = torrent_data['announce']
```

Xây dựng tham số yêu cầu

```
params = {
    'info_hash': info_hash.hex(),
    'peer_id': peer_id,
    'port': port,
    'uploaded': 0,
    'downloaded': 0,
    'left': torrent_data['info']['length'],
    'compact': 1,
    'event': 'started',
    'type': type,
}
```

Gửi yêu cầu HTTP GET đến tracker với URL và tham số đã xây dựng sau đó kiểm tra phản hồi, nếu không phải mã 200 sẽ gây ra lỗi `RequestException`.

```
response = requests.get(tracker_url, params=params)
response.raise_for_status()
```

Bước tiếp theo đợi phản hồi từ tracker trực tiếp real-time, và giải mã dữ liệu phản hồi đã được mã hóa từ tracker và gán cho `tracker_response`

```
tracker_response = bencodepy.decode(response.content)
```

Kiểm trắc nội dung trong `tracker_response` nếu tracker trả về failure reason, in thông báo lỗi và không trả về dữ liệu, ngược lại trả về phản hồi đã giải mã từ tracker (chứa thông tin như danh sách peer, thời gian chờ, v.v.).

```
if b"failure reason" in tracker_response:
    print("Thông báo đến tracker thất bại:", tracker_response[b"failure
reason"].decode())
else:
    print("Đã thông báo đến tracker thành công")
    return tracker_response
except requests.RequestException as e:
    print("Lỗi khi thông báo đến tracker:", e)
```

SOCKET_CONECTION

- Bước 1: tạo kết nối TCP bằng socket
  - Socket (TCP)
  - Tham số:
    - Port: cổng kết nối.
    - IP: địa chỉ IP của peer.
- Bước 2: tiến hành trao đổi thông tin dựa trên Peer Wire Protocol:
  - Handshake: xác định xem hai peer có chia sẻ cùng một tệp torrent hay không.
  - Gửi thông điệp (Messages): mỗi thông điệp bao gồm: 4 byte đầu tiên: cho biết kích thước của thông điệp; 1 byte tiếp theo: xác định ID thông điệp và cuối là nội dung thông điệp. Sau khi handshake thành công, các peer có thể gửi nhiều loại thông điệp khác nhau:
    - Choke: Thông báo rằng peer không muốn nhận dữ liệu từ peer kia.
    - Unchoke: Cho phép peer gửi dữ liệu.
    - Interested: Thông báo rằng peer muốn nhận dữ liệu từ peer kia.
    - Not Interested: Thông báo rằng peer không quan tâm đến dữ liệu từ peer kia.
    - Have: Thông báo rằng peer đã tải xuống một phần tệp.
    - Bitfield: Cung cấp thông tin về các phần mà peer đã tải xuống.
    - Request: Yêu cầu một phần dữ liệu cụ thể từ peer.
    - Piece: Gửi một phần dữ liệu mà peer đã yêu cầu.
- Bước 3: kết thúc trao đổi thông tin. Đóng socket.

4.2.2. Download

Tệp `download.py` có tác dụng như là một chương trình "leecher", có chức năng tải xuống các phần của tệp từ các peers trong mạng.

Trong tệp có các hàm nhằm hỗ trợ việc tải dữ liệu như sau:

- Hàm `bencode_decode`: có chức năng giải mã một giá trị bencoded (sử dụng trong tệp torrent) thành các kiểu dữ liệu Python tương ứng như chuỗi (`str`), số nguyên (`int`), danh sách (`list`), và từ điển (`dict`).

```
def decode_bencode(bencoded_value):
    if chr(bencoded_value[0]).isdigit():
        first_colon_index = bencoded_value.find(b":")
        if first_colon_index == -1:
            raise ValueError("Invalid encoded value")
```

```
        length = int(bencoded_value[:first_colon_index])
        return (
             bencoded_value[first_colon_index + 1 : first_colon_index + 1
+ length],
            bencoded_value[first_colon_index + 1 + length :],
        )
    elif chr(bencoded_value[0]) == "i":
        end_index = bencoded_value.find(b"e")
        if end_index == -1:
            raise ValueError("Invalid encoded value")
         return int(bencoded_value[1:end_index]), bencoded_value[end_index
+ 1 :]
    elif chr(bencoded_value[0]) == "l":
        list_values = []
        remaining = bencoded_value[1:]
        while remaining[0] != ord("e"):
            decoded, remaining = decode_bencode(remaining)
            list_values.append(decoded)
        return list_values, remaining[1:]
    elif chr(bencoded_value[0]) == "d":
        dict_values = {}
        remaining = bencoded_value[1:]
        while remaining[0] != ord("e"):
            key, remaining = decode_bencode(remaining)
            if isinstance(key, bytes):
                key = key.decode()
            value, remaining = decode_bencode(remaining)
            dict_values[key] = value
        return dict_values, remaining[1:]
    else:
        raise NotImplementedError(
            "Only strings, integers, lists, and dictionaries are supported
at the moment"
        )
```

- Hàm `calculate_info_hash` nhận một từ điển info từ tệp torrent, mã hóa nó thành dạng bencoded bằng `bencodepy`, sau đó tính toán mã băm SHA-1 của dữ liệu `bencoded` và trả về kết quả dưới dạng chuỗi nhị phân (20 byte). Mã băm này (gọi là `info_hash`) được sử dụng để nhận dạng duy nhất tệp torrent trong giao thức BitTorrent.

```
def calculate_info_hash(info):
    bencoded_info = bencodepy.encode(info)
    return hashlib.sha1(bencoded_info).digest()
```

- Hàm `get_public_ip` là một hàm bất đồng bộ sử dụng thư viện `aiohttp` để gửi yêu cầu HTTP đến API của https://api.ipify.org và nhận về địa chỉ IP công cộng của máy khách dưới dạng chuỗi JSON. Hàm này trả về địa chỉ IP công cộng (chuỗi) của máy khi thực thi thành công.

```
async def get_public_ip() -> str:
    async with aiohttp.ClientSession() as session:
```

```
        async with session.get("https://api.ipify.org?format=json") as
response:
            data = await response.json()
            return data['ip']
```

- Hàm `announce_to_tracker` là một hàm bất đồng bộ gửi thông báo đến tracker (URL tracker từ tệp torrent) để lấy danh sách các peers có sẵn để tải tệp. Hàm này xây dựng URL truy vấn bằng cách thêm các tham số như `info_hash`, `peer_id`, `port`, và trạng thái tải (đã tải, đang tải, còn lại), sau đó sử dụng `aiohttp` để gửi yêu cầu HTTP GET. Nếu thành công, hàm giải mã phản hồi từ tracker và trả về danh sách các peers (bao gồm địa chỉ IP và cổng), hoặc ném lỗi nếu gặp vấn đề (ví dụ: tracker từ chối yêu cầu).

```
async def announce_to_tracker(tracker_url: str, info_hash, file_length: int,
peer_id: bytes, type, port=6060) -> List[Dict]:
    public_ip = await get_public_ip()

    params = urllib.parse.urlencode({
        'info_hash': info_hash.hex(), #bytes.fromhex(info_hash),
        'peer_id': peer_id,
        'port': port,
        'uploaded': 0,
        'downloaded': 0,
        'left': file_length,
        'compact': 1,
        'event': 'started',
        'type': type,
    })
    url = f"{tracker_url}?{params}"

    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            if response.status != 200:
                raise ValueError(f"Tracker error: {response.status}")
            data = await response.read()
            decoded_response = bencodepy.decode(data)

            if b'failure reason' in decoded_response:
                raise ValueError(decoded_response[b'failure reason'].decode())

            peers = decoded_response[b'peers']
            return [
                {
                    'ip': f"{peers[i]}.{peers[i + 1]}.{peers[i + 2]}.{peers[i
+ 3]}",
                     'port': (peers[i + 4] << 8) + peers[i + 5],
                }
                for i in range(0, len(peers), 6)
                  if f"{peers[i]}.{peers[i + 1]}.{peers[i + 2]}.{peers[i +
3]}" != "" #public_ip
            ]
```

- Hàm `download_piece` là một hàm bất đồng bộ chịu trách nhiệm tải xuống một mảnh (piece) của tệp từ một peer cụ thể.Hàm thiết lập kết nối với peer bằng giao thức BitTorrent, gửi thông điệp handshake để xác thực, và yêu cầu tải xuống mảnh tệp. Dữ liệu của mảnh được tải xuống thành từng khối nhỏ (block), kiểm tra tính toàn vẹn bằng cách so sánh mã băm SHA-1 của mảnh với mã băm mong đợi từ tệp torrent. Nếu mảnh tải xuống thành công và hợp lệ, nó sẽ được lưu vào tệp đầu ra; nếu không, hàm sẽ xử lý các lỗi liên quan như kết nối hoặc dữ liệu không hợp lệ. Hàm sử dụng cơ chế giữ kết nối (keep-alive) và quản lý dữ liệu đồng bộ qua mạng với peer. Trong hàm `download_piece` cũng bao gồm các hàm chức năng như:

```
async def download_piece(file_content, piece_index, output_path, peer,
peer_id, work_queue):
    torrent_data,_ = decode_bencode(file_content)
    info_hash = calculate_info_hash(torrent_data['info'])
        print(f"Downloading piece {piece_index} from peer {peer['ip']}:
{peer['port']}")

    piece_length = torrent_data['info']['piece length']
      last_piece_length = torrent_data['info']['length'] % piece_length or
piece_length
         current_piece_length = last_piece_length if piece_index ==
(torrent_data['info']['length'] // piece_length) else piece_length
    # print(f"piece_length: {piece_length}" )
    # print(f"last_piece_length: {last_piece_length}" )
    # print(f"current_piece_length: {current_piece_length}" )
    reader, writer = await asyncio.open_connection(peer['ip'], peer['port'])
    keep_alive_interval = 30  # seconds
    handshake_msg = create_handshake(info_hash, peer_id)
    writer.write(handshake_msg)
    await writer.drain()
    async def keep_alive():
        while True:
            await asyncio.sleep(keep_alive_interval)
            writer.write(struct.pack('!I', 0))  # Send keep-alive message
            await writer.drain()

    asyncio.create_task(keep_alive())
    buffer = bytearray()
    received_length = 0
    requests_sent = 0
    block_size = 16 * 1024  # 16 KiB
    piece_data = bytearray(current_piece_length)
    bitfield_received = False
    peer_bitfield = bytearray()
    handshake_received = False

    def cleanup():
        if writer:
            writer.close()


    def send_request(begin, length):
        nonlocal requests_sent
        request_msg = struct.pack('>IBIII', 13, 6, piece_index, begin, length)
```

```
        writer.write(request_msg)
        requests_sent += 1

    def request_pieces():
        if not has_piece(piece_index):
            print(f"Peer doesn't have piece {piece_index}, skipping requests")
            return

        if piece_index in work_queue.completed_pieces:
            print(f"Piece {piece_index} already downloaded, skipping requests")
            return

        while requests_sent * block_size < current_piece_length:
            begin = requests_sent * block_size
            length = min(block_size, current_piece_length - begin)
            send_request(begin, length)

    def has_piece(index):
        byte_index = index // 8
        bit_index = 7 - (index % 8)
       return byte_index < len(peer_bitfield) and (peer_bitfield[byte_index]
& (1 << bit_index)) != 0

    def handle_piece_message(payload):
        nonlocal received_length
        block_index = struct.unpack('>I', payload[:4])[0]
        block_begin = struct.unpack('>I', payload[4:8])[0]
        block_data = payload[8:]

        piece_data[block_begin:block_begin + len(block_data)] = block_data
        received_length += len(block_data)
          print(f"Download progress piece {piece_index}: {received_length /
current_piece_length * 100:.2f}%")

        if received_length == current_piece_length:
            piece_hash = hashlib.sha1(piece_data).hexdigest()
           expected_hash = torrent_data['info']['pieces'][piece_index * 20:
(piece_index + 1) * 20].hex()
            if piece_hash == expected_hash:
                with open(output_path, 'wb') as f:
                    f.write(piece_data)
                print(f"Piece {piece_index} downloaded successfully")
                cleanup()
            else:
                print(f"Piece {piece_index} hash mismatch")

    while True:
        data = await reader.read(4096)
        if not data:
            break
        buffer.extend(data)
        while len(buffer) >= 4:
            if not handshake_received:
                if buffer[1:20] == b'BitTorrent protocol':
                    handshake_received = True
```

```
                    print("Handshake received")
                    buffer = buffer[68:]  # Remove handshake
                    interested_msg = struct.pack('>IB', 1, 2)
                    writer.write(interested_msg)
                    await writer.drain()
                    print("Sent interested message")
                else:
                    break
            else:
                message_length = struct.unpack('>I', buffer[:4])[0]
                if len(buffer) < 4 + message_length:
                    break
                message_id = buffer[4]
                payload = buffer[5:5 + message_length]
                if message_id == 0:  # Choke
                    print("Choke received")
                elif message_id == 1:  # Unchoke
                    print("Unchoke received")
                    request_pieces()
                elif message_id == 5:  # Bitfield
                    peer_bitfield = payload
                    print("Bitfield received")
                elif message_id == 7:  # Piece
                    handle_piece_message(payload)
                else:
                    print(f"Unhandled message type: {message_id}")

                buffer = buffer[4 + message_length:]
    writer.close()
    await writer.wait_closed()
    return
```

- `cleanup()`: Đóng kết nối với peer bằng cách gọi phương thức `close()` trên `writer`, đảm bảo giải phóng tài nguyên mạng.
- `send_request(begin, length)`: Tạo và gửi yêu cầu tải một khối (block) từ một mảnh với thông tin chỉ số mảnh, vị trí bắt đầu, và độ dài khối.
- `request_pieces()`: Gửi yêu cầu tải tất cả các khối của mảnh từ peer nếu peer có mảnh này và mảnh chưa được tải.
- `has_piece(index)`: Kiểm tra xem peer có sở hữu mảnh cần tải bằng cách sử dụng thông điệp bitfield và tính toán bitmask.
- `handle_piece_message(payload)`: Xử lý dữ liệu khối nhận được, cập nhật tiến độ tải, kiểm tra mã băm SHA-1 của mảnh và lưu vào tệp nếu hợp lệ.
- `keep_alive()`: Duy trì kết nối với peer bằng cách gửi định kỳ thông điệp "keep-alive" để ngăn kết nối bị ngắt.
- Hàm `create_handshake` tạo thông điệp handshake theo chuẩn giao thức BitTorrent, bao gồm các thành phần: chiều dài của chuỗi giao thức (`pstrlen`), tên giao thức (`pstr` - "BitTorrent protocol"), một chuỗi 8 byte dự phòng (`reserved`), mã băm info_hash để xác định tệp torrent, và ID peer (`peer_id`) để nhận dạng client. Thông điệp này được đóng gói bằng `struct.pack` và trả về dưới dạng chuỗi nhị phân, dùng

để gửi đến peer trong bước kết nối ban đầu. Đây là bước thiết yếu để peer xác nhận tệp torrent và thiết lập giao tiếp với client.

```
def create_handshake(info_hash, peer_id):
    pstr = b"BitTorrent protocol"
    pstrlen = len(pstr)
    reserved = b'\x00' * 8
    return struct.pack('!B', pstrlen) + pstr + reserved + info_hash + peer_id
```

- Lớp `WorkQueue` được thiết kế để quản lý trạng thái và phân phối công việc tải các mảnh tệp (pieces) trong quá trình tải xuống từ mạng ngang hàng. Nó theo dõi các mảnh tệp chưa tải, đang tải, và đã tải xong, đồng thời cung cấp cơ chế kiểm tra tiến độ và phân phối các mảnh một cách hợp lý cho các worker.

```
class WorkQueue:
    """A queue to manage the download of pieces."""
    def __init__(self, total_pieces: int):
        self.pending_pieces = set(range(total_pieces))
        self.in_progress_pieces = set()
        self.completed_pieces = set()

    def get_next_piece(self) -> int:
        """Retrieve the next piece to download."""
        if not self.pending_pieces:
            return None
        piece = self.pending_pieces.pop()
        self.in_progress_pieces.add(piece)
        return piece

    def mark_piece_complete(self, piece_index: int):
        """Mark a piece as downloaded and verified."""
        self.in_progress_pieces.discard(piece_index)
        self.completed_pieces.add(piece_index)

    def is_complete(self) -> bool:
        """Check if all pieces are downloaded."""
        return not self.pending_pieces and not self.in_progress_pieces
```

- Phương thức `__init__(self, total_pieces: int)` khởi tạo lớp WorkQueue, nhận vào tổng số mảnh tệp (`total_pieces`) và tạo tập hợp `pending_pieces` chứa tất cả các mảnh cần tải (từ `0` đến `total_pieces - 1`). Khởi tạo hai tập hợp rỗng `in_progress_pieces` và `completed_pieces` để theo dõi các mảnh đang tải và đã tải xong.
- Phương thức `get_next_piece(self) -> int` lấy chỉ số của mảnh tệp tiếp theo cần tải từ danh sách `pending_pieces`. Loại bỏ mảnh khỏi `pending_pieces` và thêm mảnh đó vào `in_progress_pieces` sau đó trả về chỉ số của mảnh. Nếu không còn mảnh nào trong danh sách, trả về `None`.
- Phương thức `mark_piece_complete(self, piece_index: int)` đánh dấu một mảnh tệp là đã tải xuống và xác minh thành công. Hàm loại bỏ mảnh đó khỏi `in_progress_pieces` sau đó thêm mảnh vào .

- Phương thức `is_complete(self) -> bool` kiểm tra xem quá trình tải xuống tất cả các mảnh tệp đã hoàn tất hay chưa. Hàm trả về `True` nếu cả hai tập hợp `pending_pieces` và `in_progress_pieces` đều trống, ngược lại trả về `False`.
- Hàm `download_worker` chịu trách nhiệm tải xuống các mảnh (pieces) từ một peer cụ thể. Nó phối hợp với WorkQueue để lấy các mảnh cần tải, xử lý tải xuống từng mảnh, và đảm bảo mỗi mảnh được xác minh trước khi đánh dấu hoàn thành.

```
async def download_worker(peer, file_content, torrent_data, info_hash,
peer_id, work_queue, downloaded_pieces, peers):

    max_retries = 50
    max_retries_per_piece = {}
    current_peer_index = peers.index(peer) if peer in peers else 0

    def get_next_peer():
        nonlocal current_peer_index
        current_peer_index = (current_peer_index + 1) % len(peers)
        return peers[current_peer_index]

    while not work_queue.is_complete():
        piece_index = work_queue.get_next_piece()
        if piece_index is None:
            print("No more pieces to download")
            break

        current_peer = peer
        retry_count = max_retries_per_piece.get(piece_index, 0)
        success = False

        while not success and retry_count < max_retries:
            try:
                temp_path = Path(os.path.join(
                    tempfile.gettempdir(),
                  f"piece_{piece_index}_{int(time.time())}_{random.randint(0,
10000)}"
                ))

                print(f"Attempting to download piece {piece_index} from peer
{current_peer['ip']}:{current_peer['port']}")
                await download_piece(
                    file_content,
                    piece_index,
                    temp_path,
                    current_peer,
                    peer_id,
                    work_queue
                )

                if temp_path.exists():
                    with temp_path.open("rb") as temp_file:
                        piece_data = temp_file.read()
                        piece_hash = hashlib.sha1(piece_data).hexdigest()
                             expected_hash = torrent_data["info"]["pieces"]
[piece_index * 20: piece_index * 20 + 20].hex()
```

```
                        if piece_hash == expected_hash:
                            downloaded_pieces[piece_index] = piece_data
                            work_queue.mark_piece_complete(piece_index)
                                print(f"Piece {piece_index} verified and
stored successfully")
                            success = True
                        else:
                            raise ValueError(f"Hash verification failed for
piece {piece_index}")

                    temp_path.unlink()  # Xóa tệp tạm sau khi sử dụng

            except Exception as error:
                    print(f"Error downloading piece {piece_index} from peer
{current_peer['ip']}:{current_peer['port']}: {error}")

                if isinstance(error, ConnectionResetError):
                        print(f"Connection reset while downloading piece
{piece_index}. Retrying...")
                    retry_count += 1
                    max_retries_per_piece[piece_index] = retry_count
                    current_peer = get_next_peer()
                    await asyncio.sleep(2)
                else:
                    current_peer = get_next_peer()
                    max_retries_per_piece[piece_index] = retry_count + 1
                    await asyncio.sleep(1 * (retry_count + 1))

        if not success:
                print(f"Failed to download piece {piece_index} after trying
all peers")
```

Cách hoạt động:

- Lấy mảnh tiếp theo từ WorkQueue: Sử dụng `work_queue.get_next_piece()` để lấy mảnh cần tải. Nếu không còn mảnh nào, hàm dừng lại.
- Quản lý peer hiện tại: Bắt đầu tải mảnh từ peer đã được chỉ định. Nếu tải không thành công, chuyển sang peer tiếp theo trong danh sách peers.
- Tải xuống và xác minh mảnh: Sử dụng `download_piece` để tải một mảnh từ peer. Sau khi tải xong, kiểm tra tính toàn vẹn của mảnh bằng cách so sánh mã băm SHA-1 với mã băm mong đợi trong tệp torrent. Nếu mảnh hợp lệ, lưu trữ mảnh vào danh sách `downloaded_pieces` và đánh dấu mảnh đó là hoàn thành trong `work_queue`.
- Xử lý lỗi: Nếu việc tải xuống thất bại (do lỗi mạng, kết nối bị ngắt, hoặc hash không khớp), hàm sẽ chuyển sang peer khác. Hàm có giới hạn số lần thử lại (retry) cho mỗi mảnh để tránh vòng lặp vô tận.
- Lặp lại cho đến khi hoàn tất: Tiếp tục tải các mảnh mới từ `work_queue` cho đến khi tất cả các mảnh đã được tải xuống hoặc danh sách peers không còn sử dụng được.
- Hàm `download_file` là một hàm bất đồng bộ chịu trách nhiệm quản lý toàn bộ quá trình tải xuống tệp từ mạng BitTorrent. Nó xử lý từ việc đọc thông tin tệp torrent,

kết nối với tracker để lấy danh sách peers, phân phối công việc tải các mảnh tệp, đến việc ghép nối và lưu trữ tệp đã tải.

```
async def download_file(file_content, output_path):
    torrent_data, _ = decode_bencode(file_content)
    info = torrent_data['info']
    file_length = info['length']
    piece_length = info['piece length']
    total_pieces = (file_length + piece_length - 1) // piece_length

    print(f"Downloading {total_pieces} pieces...")
    info_hash = calculate_info_hash(info)
    peer_id = generate_peer_id()
    tracker_url = torrent_data['announce'].decode()
    peers = [{'ip':'192.168.1.9', 'port': 5143}]
    if not peers:
        raise ValueError("No peers found.")

    print(f"Found {len(peers)} peers.")
    downloaded_pieces = {}
    work_queue = WorkQueue(total_pieces)
    tasks = [
       download_worker(peer, file_content, torrent_data, info_hash, peer_id,
work_queue, downloaded_pieces, peers)
        for peer in peers
    ]

    await asyncio.gather(*tasks)

    if work_queue.is_complete():
        print("Download complete.")
    else:
        print("Download incomplete. Some pieces are missing.")

    final_data = b''.join(
        data for _, data in sorted(downloaded_pieces.items())
    )
    # Lấy tên tệp từ metadata
    file_name = torrent_data["info"]["name"]
    if isinstance(file_name, bytes):
        file_name = file_name.decode("utf-8")

    output_file_path = os.path.join(output_path, file_name)

    Path(output_path).mkdir(parents=True, exist_ok=True)
    with open(output_file_path, "wb") as f:
        f.write(final_data)

    print(f"Download completed: {output_file_path}")
    return str(output_file_path)
```

- Cách hoạt động của hàm `download_file`:
  - Giải mã tệp torrent: Giải mã dữ liệu bencoded từ tệp torrent để lấy thông tin cần thiết, bao gồm:

  - Tổng kích thước tệp (length).
  - Kích thước mỗi mảnh (piece length).
  - URL tracker (announce).

- Tính toán số lượng mảnh: Dựa trên tổng kích thước tệp và kích thước mỗi mảnh, tính số lượng mảnh cần tải (`total_pieces`).
- Tạo `info_hash` và : Tính toán `info_hash` từ phần info của tệp torrent để xác định duy nhất tệp. Đồng thời tạo một `peer_id` ngẫu nhiên để nhận dạng client trong mạng.
- Liên lạc với tracker: gửi yêu cầu đến tracker bằng hàm `announce_to_tracker` để nhận danh sách các peers có sẵn để tải tệp.
- Khởi tạo các thành phần quản lý: Tạo một đối tượng `WorkQueue` để quản lý trạng thái các mảnh tệp. Khởi tạo một dictionary `downloaded_pieces` để lưu trữ dữ liệu các mảnh đã tải.
- Tải xuống mảnh từ các peers: Sử dụng asyncio.gather để khởi chạy nhiều worker (hàm `download_worker`) tải các mảnh từ danh sách peers đồng thời.
- Kiểm tra hoàn thành: Sau khi các worker kết thúc, kiểm tra trạng thái của `WorkQueue`. Nếu hoàn tất, tiếp tục ghép nối các mảnh lại thành tệp hoàn chỉnh. Nếu chưa hoàn thành, thông báo có mảnh bị thiếu.
- Lưu trữ tệp đã tải: Lấy tên tệp từ `metadata` trong `info`. Lưu các mảnh ghép nối thành tệp hoàn chỉnh vào đường dẫn đầu ra (`output_path`).

Kết quả sau khi thực hiện hàm `download_file` là khi thành công, tệp đã tải xuống được lưu tại vị trí chỉ định. Nếu quá trình tải xuống không hoàn tất (thiếu mảnh), chương trình sẽ thông báo lỗi.

- Hàm `start_leecher` khởi động quá trình tải xuống tệp từ mạng BitTorrent bằng cách đọc nội dung tệp torrent qua hàm `read_file` và gọi hàm bất đồng bộ `download_file` để xử lý toàn bộ quá trình tải xuống. Hàm nhận vào đường dẫn tệp torrent (`torrent_file`) và thư mục đầu ra (`output_path`), thực hiện việc tải các mảnh tệp từ mạng ngang hàng và lưu tệp đã tải hoàn chỉnh vào thư mục chỉ định.

```
async def start_leecher(torrent_file, output_path):
    file_content = read_file(torrent_file)
    await download_file(file_content, output_path)
```

4.2.3. Upload

Tệp `upload.py` được thiết kế với các chức năng chính bao gồm tạo tệp torrent, giao tiếp với tracker, và triển khai seeder. Chương trình cho phép tạo tệp .torrent từ một tệp dữ liệu, tính toán các mã băm SHA-1 cho từng phần (piece) của tệp và lưu thông tin như tên tệp, kích thước, và URL tracker. Nó có khả năng thông báo trạng thái của peer (seeder hoặc leecher) với tracker để duy trì kết nối mạng ngang hàng. Bên cạnh đó, chương trình còn triển khai seeder, lắng nghe các kết nối từ peer, xử lý xác thực bằng handshake, gửi thông báo trạng thái (bitfield, unchoke) và phục vụ dữ liệu theo yêu cầu của peer. Với các công cụ tiện ích như tạo ID peer và tính toán băm SHA-1,

tệp này cung cấp các thành phần cốt lõi để xây dựng hệ thống chia sẻ tệp thông qua mạng BitTorrent.

- Hàm `generate_torrent` có chức năng tạo một tệp .torrent từ một tệp cụ thể.

```
def generate_file_torrent(file_path, announce_list, torrent_name, path):
  #files = [os.path.abspath(file_path)]
  torrent_name_without_extension = os.path.splitext(torrent_name)[0]
  torrent_data = {
      "announce": announce_list[0],  # Chọn tracker đầu tiên
      "info": {
          "name": torrent_name,
          "length": os.path.getsize(file_path),
          "piece length": 16384,  # Kích thước mỗi phần (byte)
          "pieces": bytes(bencode_pieces(file_path, 16384))  # Tính toán
các mã băm của từng phần
      }
  }

output_path = os.path.join(path, f"
{torrent_name_without_extension}.torrent")
  with open(output_path, "wb") as torrent_file:
      torrent_file.write(bencodepy.encode(torrent_data))

  return output_path
```

  - Khởi tạo dữ liệu torrent:

    `announce`: Chỉ định tracker URL (lấy URL đầu tiên từ danh sách `announce_list`).

    `info`: `name`: Tên của torrent, thường là tên tệp hoặc mô tả ngắn.

    `length`: Kích thước của tệp gốc (tính bằng byte) bằng cách sử dụng `os.path.getsize(file_path)`.

    `piece length`: Kích thước của mỗi phần (piece), cố định ở 16KB (16384 byte).

    `pieces`: Danh sách các mã băm SHA-1 cho từng phần (piece) của tệp, được tạo bởi hàm `bencode_pieces`.

  - Tính toán mã băm các phần (pieces): Hàm `bencode_pieces` chia tệp thành các phần có kích thước 16KB. Từng phần được băm bằng SHA-1, kết quả là một chuỗi byte chứa tất cả các mã băm nối liền.

  - Ghi dữ liệu torrent ra tệp: Tạo đường dẫn tệp torrent, lưu ở thư mục Downloads với tên {`torrent_name`}.torrent. Mã hóa dữ liệu torrent bằng `bencodepy.encode`. Ghi dữ liệu bencoded vào tệp .torrent.

  - Trả về đường dẫn tệp: Sau khi hoàn tất, hàm trả về đường dẫn đầy đủ của tệp .torrent vừa được tạo.

- Hàm `bencode_pieces` có nhiệm vụ tính toán các mã băm SHA-1 cho từng phần (`piece`) của tệp gốc. Các mã băm này được sử dụng trong giao thức BitTorrent để đảm bảo tính toàn vẹn của dữ liệu khi chia sẻ giữa các peer.

```
def bencode_pieces(file_path, piece_length):

  pieces = bytearray()
  with open(file_path, "rb") as f:
      while True:
          piece = f.read(piece_length)
          if not piece:
              break
          pieces.extend(hashlib.sha1(piece).digest())
  return pieces
```

- Khởi tạo: Một `bytearray` rỗng (`pieces`) được tạo ra để lưu trữ tất cả các mã băm SHA-1 của các phần.
- Mở tệp gốc: Hàm mở tệp ở chế độ nhị phân (`rb`) để đảm bảo xử lý được dữ liệu gốc của tệp, bao gồm cả dữ liệu không phải văn bản.
- Đọc từng phần: Dữ liệu từ tệp được đọc thành từng phần nhỏ (kích thước được chỉ định bởi piece_length, mặc định thường là 16KB = 16384 byte). Nếu không còn dữ liệu để đọc (tệp đã hết), vòng lặp dừng lại.
- Tính mã băm SHA-1: Sử dụng `hashlib.sha1(piece).digest()` để tính mã băm SHA-1 của phần vừa đọc. Kết quả là một chuỗi byte 20 byte được thêm vào pieces.
- Trả về: Sau khi tất cả các phần được băm, hàm trả về một chuỗi byte chứa toàn bộ các mã băm SHA-1 được nối liền.

- Hàm `send_handshake` có nhiệm vụ gửi thông điệp handshake đến một peer khác trong mạng BitTorrent. Đây là bước đầu tiên trong giao tiếp giữa các peer, được sử dụng để xác thực và thiết lập kết nối.
  - Xây dựng các thành phần của handshake:

```
async def send_handshake(writer, info_hash, peer_id):
pstr = b"BitTorrent protocol"
pstrlen = len(pstr)
reserved = b'\x00' * 8
handshake = struct.pack('!B', pstrlen) + pstr + reserved + info_hash
+ peer_id
writer.write(handshake)
await writer.drain()
```

`pstr`: Chuỗi cố định "BitTorrent protocol" chỉ định giao thức được sử dụng.

`pstrlen`: Độ dài chuỗi `pstr`, tính bằng `len(pstr)`, sẽ là 19.

`reserved`: 8 byte dành riêng, thường để trống (`b'\x00' * 8`).

`info_hash`: Mã băm SHA-1 của trường `info` từ tệp torrent, dùng để xác định tệp mà các peer muốn chia sẻ.

`peer_id`: ID của peer (20 byte), là một chuỗi ngẫu nhiên dùng để nhận diện peer.

  - Đóng gói thông điệp: Sử dụng `struct.pack` để xây dựng thông điệp theo định dạng:

`!B`: Một số nguyên không dấu 1 byte (độ dài của pstr). Kết hợp tất cả các phần theo thứ tự:

`\text{<pstrlen><pstr><reserved><info_hash><peer_id>}`

Kết quả là một chuỗi byte hoàn chỉnh.

- Gửi thông điệp:

  Dùng `conn.send(handshake)` để gửi thông điệp qua kết nối socket đã mở với peer.

- Cấu trúc của thông điệp handshake:

  `pstrlen` (1 byte): Độ dài của chuỗi pstr. `pstr` (19 byte): Chuỗi giao thức "BitTorrent protocol". `reserved` (8 byte): Dành riêng (hiện tại là toàn bộ 0x00). `info_hash` (20 byte): Mã băm SHA-1 xác định tệp torrent. `peer_id` (20 byte): ID của peer gửi thông điệp.

  Tổng độ dài: $1 + 19 + 8 + 20 + 20 = 68$ byte.

- Hàm `send_bitfield` có nhiệm vụ gửi thông điệp bitfield đến một peer trong mạng BitTorrent. Bitfield là một thông điệp quan trọng trong giao thức BitTorrent, dùng để thông báo cho peer biết seeder (hoặc leecher) hiện có những phần (pieces) nào của tệp đang chia sẻ.

  - Tính toán chiều dài của bitfield:

```python
async def send_bitfield(writer, total_pieces):

bitfield_length = (total_pieces + 7) // 8
bitfield = bytearray([0xFF] * bitfield_length) # All pieces available
bitfield_message = bytearray(4) + bytearray([5]) + bitfield
struct.pack_into('!I', bitfield_message, 0, len(bitfield_message) - 4)
print("Sent bitfield message")
writer.write(bitfield_message)
await writer.drain()
```

  `bitfield_length`: Xác định số byte cần thiết để biểu diễn tất cả các phần. Vì mỗi bit trong bitfield đại diện cho một phần (piece), số byte cần thiết là:

  $$\text{bitfield\_length} = \frac{\text{total\_pieces}}{8}$$

  - Tạo bitfield:

    `bytearray([0xFF] * bitfield_length)`: Bitfield là một chuỗi byte, trong đó mỗi bit có giá trị 1 (giả định seeder có đầy đủ tất cả các phần). Mỗi byte có 8 bit, biểu diễn trạng thái của 8 phần.

  - Đóng gói thông điệp: Cấu trúc thông điệp bitfield:

    – Length prefix (4 byte): Độ dài của toàn bộ thông điệp (tính từ `Message ID` trở đi).

    – Message ID (1 byte): Giá trị cố định là 5, biểu thị loại thông điệp là bitfield.

– Bitfield: Danh sách trạng thái các phần (1 = có sẵn, 0 = không có).

Sử dụng `struct.pack_into` để ghi length prefix vào đầu thông điệp.

- Gửi thông điệp: Dùng `conn.send(bitfield_message)` để gửi thông điệp qua kết nối socket.

- Hàm `send_unchoke` có nhiệm vụ gửi thông điệp unchoke đến một peer. Thông điệp này thông báo rằng seeder (hoặc leecher) đã bỏ chặn (unchoke) peer đó, cho phép peer bắt đầu gửi yêu cầu tải xuống các phần dữ liệu (pieces).

  - Tạo thông điệp unchoke: `length prefix` (4 byte): Độ dài của phần thông điệp từ `ID` trở đi, cố định là 1 byte (vì chỉ có `message ID`).

    `message ID` (1 byte): Giá trị 1, biểu thị đây là thông điệp unchoke.

  - Đóng gói thông điệp: Sử dụng `struct.pack_into` để ghi `length prefix` vào đầu thông điệp. Ghi `message ID` vào byte thứ 5 của thông điệp (vị trí 4, do index bắt đầu từ 0).

  - Gửi thông điệp: Sử dụng `conn.send(unchoke_message)` để gửi thông điệp unchoke qua kết nối socket đến peer.

  - In trạng thái: In ra thông báo xác nhận rằng thông điệp unchoke đã được gửi.

- Hàm `send_piece_message` có nhiệm vụ gửi thông điệp chứa một phần dữ liệu (piece) từ seeder đến một peer trong mạng BitTorrent. Đây là một trong những thông điệp quan trọng, vì nó thực sự truyền dữ liệu của tệp mà các peer đang tải xuống.

  - Tính chiều dài thông điệp: `message_length`: Tổng chiều dài của thông điệp, bao gồm:

    9 byte cố định: Gồm `message ID`, `index`, và `begin offset`.

    Dữ liệu thực tế (piece_data).

  - Tạo thông điệp piece:

    Cấu trúc thông điệp: `length prefix` (4 byte): Độ dài toàn bộ thông điệp (không tính `length prefix`).

    `message ID` (1 byte): Giá trị 7, biểu thị đây là thông điệp piece.

    `piece index` (4 byte): Chỉ số của phần dữ liệu (piece) trong tệp torrent.

    `begin offset` (4 byte): Vị trí bắt đầu trong phần (block) mà peer yêu cầu.

    `block` (n byte): Dữ liệu thực tế được gửi.

    Sử dụng `struct.pack_into` để đóng gói các giá trị như `length prefix`, `piece index`, và `begin offset`.

  - Gửi thông điệp: Dùng `conn.send(piece_message)` để gửi thông điệp piece qua kết nối socket.

- In trạng thái: Sau khi gửi, hàm in ra thông tin về piece đã gửi để kiểm tra và debug.

- Hàm `start_seeder` chịu trách nhiệm khởi chạy một seeder trong mạng BitTorrent. Seeder này lắng nghe các kết nối từ các peer khác, xử lý giao tiếp thông qua giao thức BitTorrent, và phục vụ dữ liệu (pieces) theo yêu cầu.
  - Khởi tạo thông tin seeder: Đọc và giải mã tệp torrent (`torrent_file`) để lấy thông tin như `info_hash`, kích thước tệp, độ dài từng phần (piece), và tổng số phần. Đồng thời tạo một `peer_id` ngẫu nhiên để định danh seeder.
  - Giao tiếp với tracker: Gửi thông báo đến tracker về trạng thái seeder (`event='started'`).
  - Tải tệp vào bộ nhớ: Đọc toàn bộ tệp cần chia sẻ (`file_buffer`) để sẵn sàng phục vụ dữ liệu.
  - Thiết lập socket: Tạo một socket để lắng nghe kết nối từ các peer trên cổng được chỉ định (`port`).
  - Xử lý kết nối từ peer:
    - Chấp nhận kết nối từ peer.
    - Nhận và phân tích thông điệp từ peer:
      - Handshake: Peer gửi handshake để xác thực tệp torrent đang được chia sẻ.
      - Bitfield: Seeder gửi trạng thái các phần dữ liệu hiện có.
      - Unchoke: Thông báo rằng seeder sẵn sàng chia sẻ dữ liệu.
      - Yêu cầu dữ liệu (request): Peer yêu cầu một phần dữ liệu cụ thể, và seeder phản hồi bằng cách gửi thông điệp chứa phần dữ liệu đó (`piece`).
  - Quản lý kết nối: Xử lý từng thông điệp của peer cho đến khi kết nối kết thúc hoặc bị ngắt.

4.3. Multi-direction data transfering (MDDT)

Tệp `view.py` đảm nhận nhiệm vụ xây dựng các API để quản lý tệp torrent, bao gồm tải lên, tải xuống (leecher), và tạo seed (seeder). Nó cung cấp các endpoint RESTful để liệt kê danh sách torrent, xử lý chi tiết từng torrent (lấy thông tin, cập nhật, hoặc xóa), đồng thời hỗ trợ điều khiển các nhiệm vụ bất đồng bộ như tải xuống hoặc seeding thông qua các lệnh nhận từ client. Tệp cũng sử dụng threading để thực hiện các tác vụ nền và theo dõi trạng thái nhiệm vụ. Ngoài ra, các chức năng như kiểm tra trạng thái nhiệm vụ hoặc xử lý lệnh tùy chỉnh được thiết kế để nâng cao khả năng kiểm soát và tương tác với hệ thống tải/seed torrent.

- Hàm `torrents_list` có nhiệm vụ cung cấp API để liệt kê tất cả các tệp torrent hoặc thêm một tệp torrent mới và bắt đầu tải xuống.

```
@api_view(['GET', 'POST'])
def torrents_list(request, format=None):
    """
    List all code torrents, or create a new torrent.
    """
    if request.method == 'GET':
        torrents = Torrents.objects.all()
```

```
        serializer = TorrentsSerializer(torrents, many=True)
        return Response(serializer.data)

    elif request.method == 'POST':
        torrent_file = request.FILES['file']

        # Lưu torrent file
        torrent_instance = TorrentFile.objects.create(file=torrent_file)
        torrent_path = torrent_instance.file.path

        start_leecher(torrent_path, "C:/Users/HP/Downloads/temp")

    return Response({
        'message': 'Downloaded successfully',
     }, status=201)
```

Trong đó:

`GET`: Truy xuất danh sách các tệp torrent từ cơ sở dữ liệu bằng cách sử dụng model Torrents và chuyển đổi chúng thành JSON để trả về.

`POST`: Nhận file torrent từ người dùng, lưu vào model TorrentFile, và bắt đầu quá trình leecher bằng cách đọc nội dung file và tải dữ liệu xuống.

- Hàm `run_async_coroutine` có nhiệm vụ hỗ trợ thực thi các coroutine bất đồng bộ trong môi trường đồng bộ.

```
def run_async_coroutine(coroutine, *args, **kwargs):
asyncio.run(coroutine(*args, **kwargs))
```

Chi tiết: Sử dụng `asyncio.run` để chạy các coroutine, chẳng hạn như `start_leecher` hoặc `start_seeder`, trong các thread được khởi tạo.

- Hàm `start_leecher` có nhiệm vụ thực hiện quá trình tải xuống dữ liệu từ tệp torrent.

```
async def start_leecher(torrent_file, output_path, task_id):
    file_content = read_file(torrent_file)
    await download_file(file_content, output_path)
    task_statuses[task_id] = "completed"
```

Cách hoạt động: Đọc nội dung file torrent bằng cách sử dụng hàm `read_file`. Sau đó tải dữ liệu xuống đường dẫn đầu ra bằng coroutine `download_file`. Cuối cùng cập nhật trạng thái nhiệm vụ tương ứng thành "completed" trong `dictionary task_statuses`.

- Lớp `DownloadView` có nhiệm vụ xử lý yêu cầu tải xuống tệp torrent hoặc dừng các nhiệm vụ leecher đang chạy.

```
class DownloadView(View):
    def post(self, request, *args, **kwargs):
```

```
        torrent_file = request.FILES['file']
        command = request.POST.get('command')
        path_output = request.POST.get('path_output')
        torrent_instance = TorrentFile.objects.create(file=torrent_file)
        torrent_path = torrent_instance.file.path
        if command == 'down':
            print("Received 'down' command")  # Thêm lệnh in để kiểm tra
            task_id = id(threading.current_thread())
            task_statuses[task_id] = "running"
                        task  =  threading.Thread(target=run_async_coroutine,
args=(start_leecher, torrent_path, path_output, task_id))
            task.start()
            tasks_leech[task_id] = task
            return JsonResponse({'message': 'Task started',
              'task_id': task_id})

        elif command == 'stop':
            print("Received 'stop' command")  # Thêm lệnh in để kiểm tra
            task_id = int(request.POST.get('task_id'))
            task = tasks_leech.get(task_id)
            if task:
                print("Stopping thread is unsafe and not recommended")
                task_statuses[task_id] = "stopped"
                del tasks_leech[task_id]
                 return JsonResponse({'message': 'Task stopped', 'task_id':
task_id})
            else:
                return JsonResponse({'error': 'Task not found'}, status=404)

        return JsonResponse({'error': 'Invalid request'}, status=400)
```

Lệnh `down`:

Bắt đầu một thread mới để thực thi coroutine `start_leecher`. Lưu thông tin nhiệm vụ (ID, trạng thái) vào các dictionary tasks_leech và task_statuses.

Lệnh `stop`:

Xóa nhiệm vụ tương ứng khỏi tasks_leech và cập nhật trạng thái thành "stopped".

- `GenerateTorrentView`

  Nhiệm vụ: Xử lý yêu cầu tạo seed (seeder) cho torrent và quản lý các nhiệm vụ seeder.

```
class GenerateTorrentView(View):
    def post(self, request, *args, **kwargs):
        command = request.POST.get('command')
        path_input = request.POST.get('path_input')
        path_output = request.POST.get('path_output')

        if command == 'run':
            print("Received 'run' command")  # Thêm lệnh in để kiểm tra
            port = random.randint(1024, 65535) # Chọn một cổng ngẫu nhiên từ
1024 đến 65535
```

```
            task_id = id(threading.current_thread())
                        task  =  threading.Thread(target=run_async_coroutine,
args=(start_seeder, path_output, path_input, port))
            task.start()
            tasks_seed[task_id] = task
            return JsonResponse({'message': 'Task started',
              'task_id': task_id})

        elif command == 'stop':
            print("Received 'stop' command")  # Thêm lệnh in để kiểm tra
            task_id = int(request.POST.get('task_id'))
            task = tasks_seed.get(task_id)
            if task:
                print("Stopping thread is unsafe and not recommended")
                del tasks_seed[task_id]
                 return JsonResponse({'message': 'Task stopped', 'task_id':
task_id})
            else:
                return JsonResponse({'error': 'Task not found'}, status=404)

        return JsonResponse({'error': 'Invalid request'}, status=400)
```

Chi tiết: Lệnh `run`: Chọn ngẫu nhiên một cổng mạng để seeding. Bắt đầu một thread để chạy coroutine `start_seeder`. Lưu thông tin nhiệm vụ vào `dictionary tasks_seed`. Lệnh stop: Tìm nhiệm vụ tương ứng và xóa khỏi `tasks_seed`.

5. Hướng dẫn sử dụng

Để bắt đầu sử dụng phần mềm P2P để tiến hành chia sẽ file, trước tiên cần chuẩn bị:

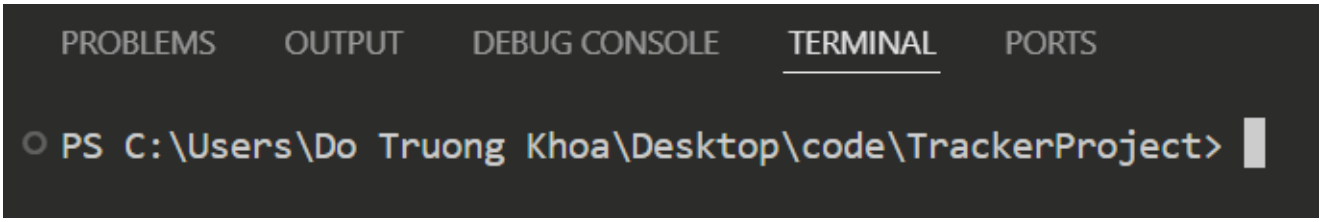
- 1 máy tính Host Tracker
- Ít nhất 2 chiếc máy tính để truyền file
- một đường dẫn mạng chung được kết nối với 3 máy trên

Các bước chuẩn bị setting code:

- tải source code từ github

```
git clone https://github.com/salamander53/TrackerProject.git
```

- Mở terminal **(Các bước kế dành cho máy ngoài host tracker)**



- Và di chuyển tới backend

```
cd .\Web\backend\
```

- Chạy môi trường ảo

```
venv\Scripts\activate
```

- **Sửa anounce_list = http:// + [ip Link-local Ipv6 của máy Host Tracker] + :8080/announce ở trong tệp**

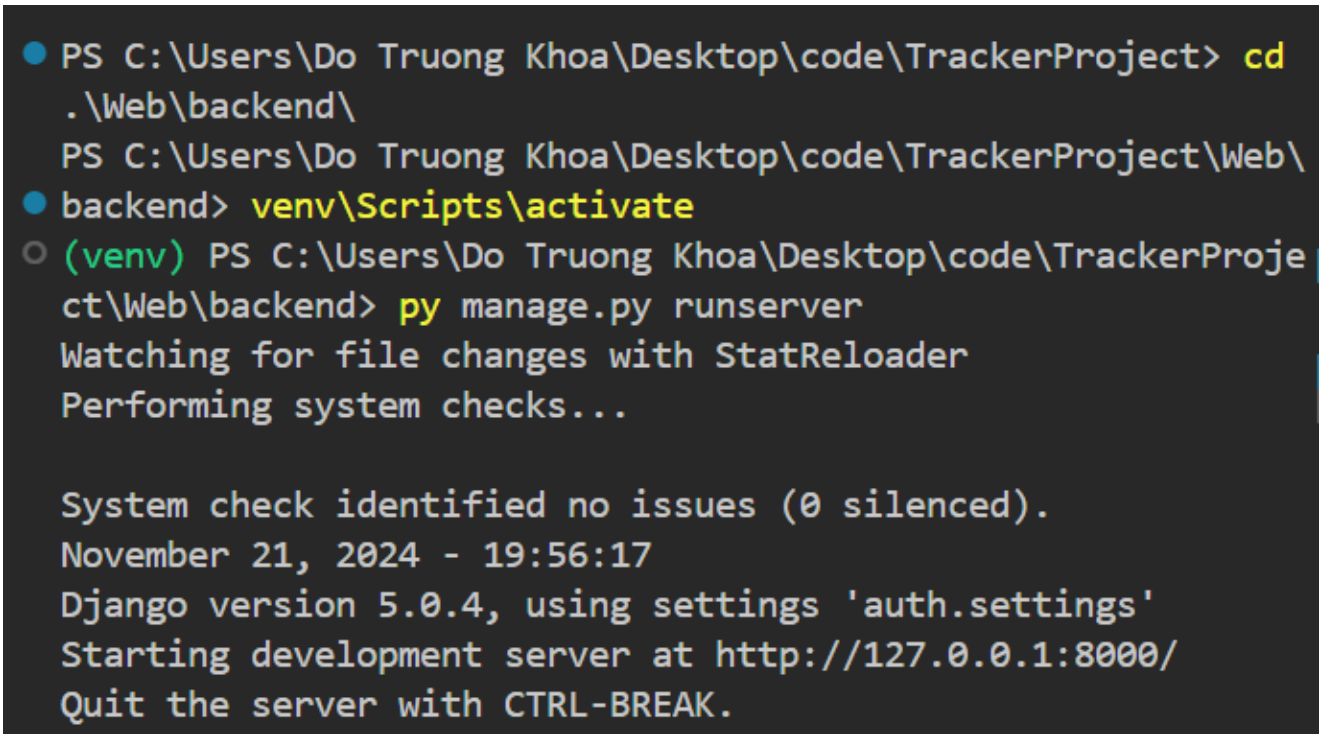
```
.\TrackerProject\Web\backend\torrents\upload.py
```

ip 192.168.43.129 trong hình là ip Link-local Ipv6 của máy tôi định host, nếu không biết ip thì chạy ipconfig trên máy host

- Chạy backend

```
python manage.py runserver
```

- Kết quả



- Chạy Frontend

```
cd .\Web\frontend\
```

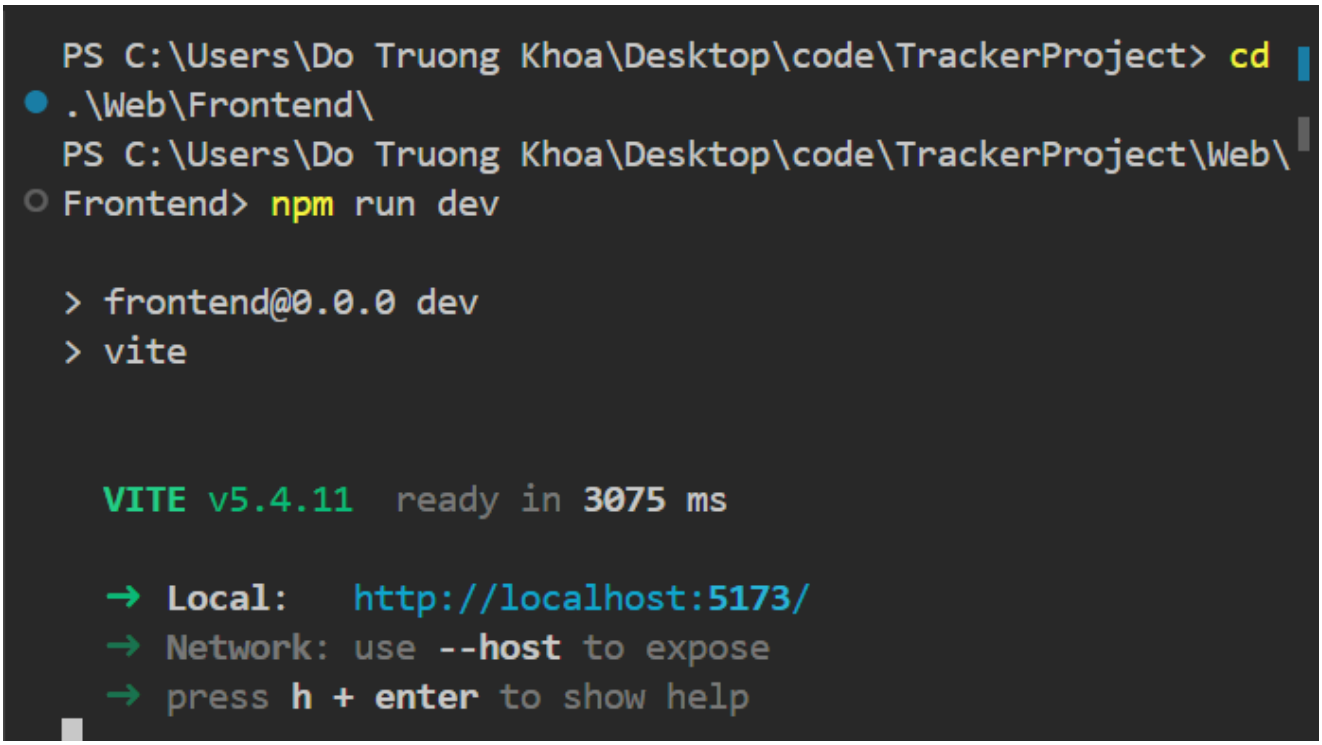
- install về các gói cần thiết

```
npm install
```

- Chạy frontend

```
npm run dev
```

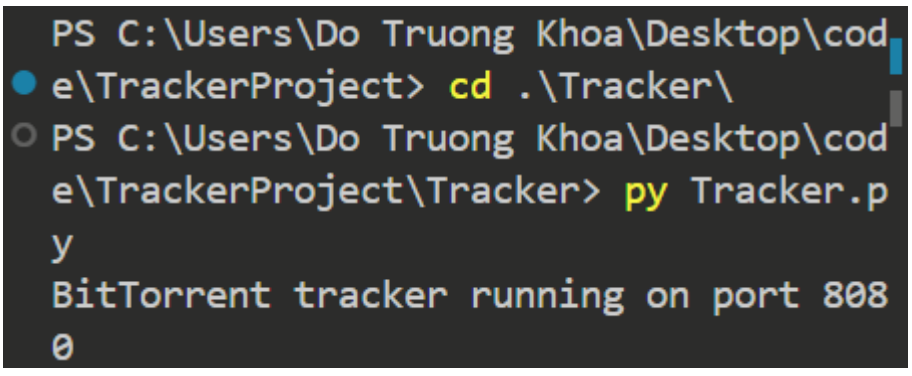
- Kết quả



- Cài đặt frontend và backend tương tự cho những máy khác **(NGOÀI HOST TRACKER)**
- Tiếp theo tiến hành Host Tracker trên 1 máy đã chuẩn bị

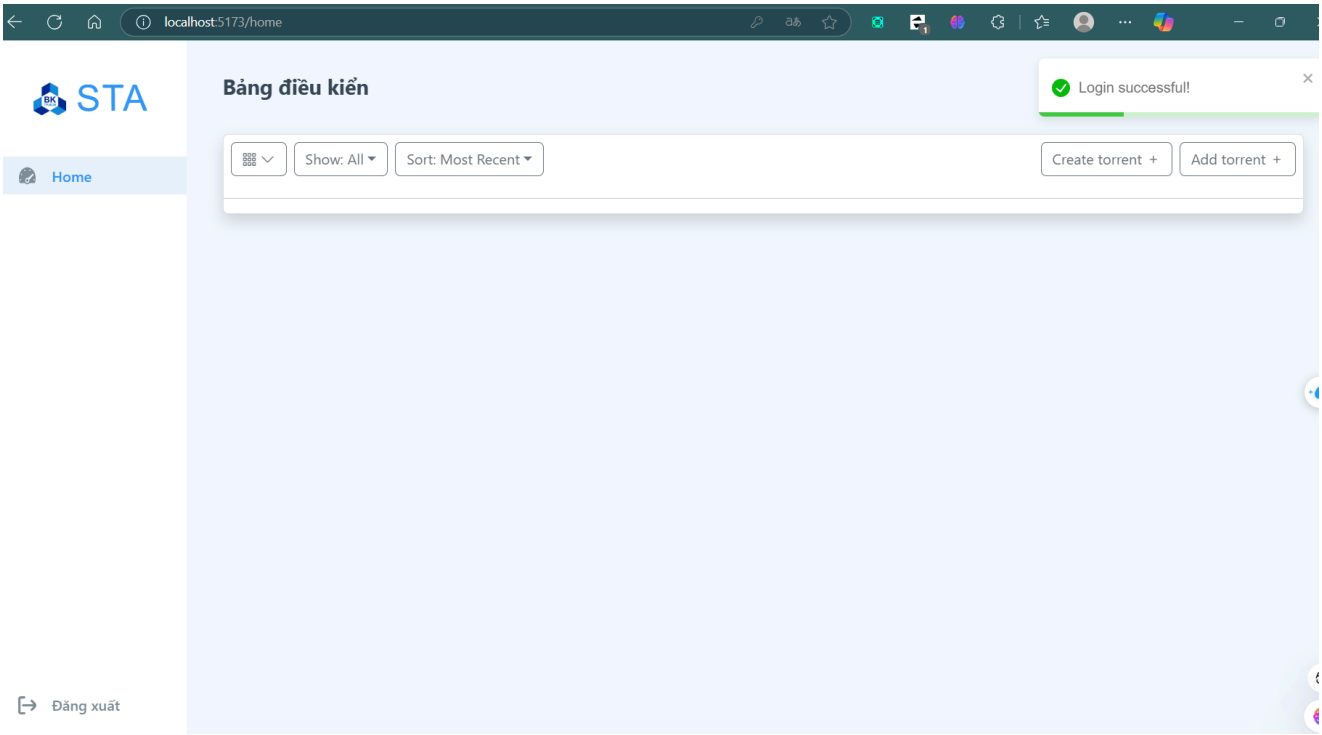
```
cd .\Tracker\
py Tracker.py
```

- Kết quả

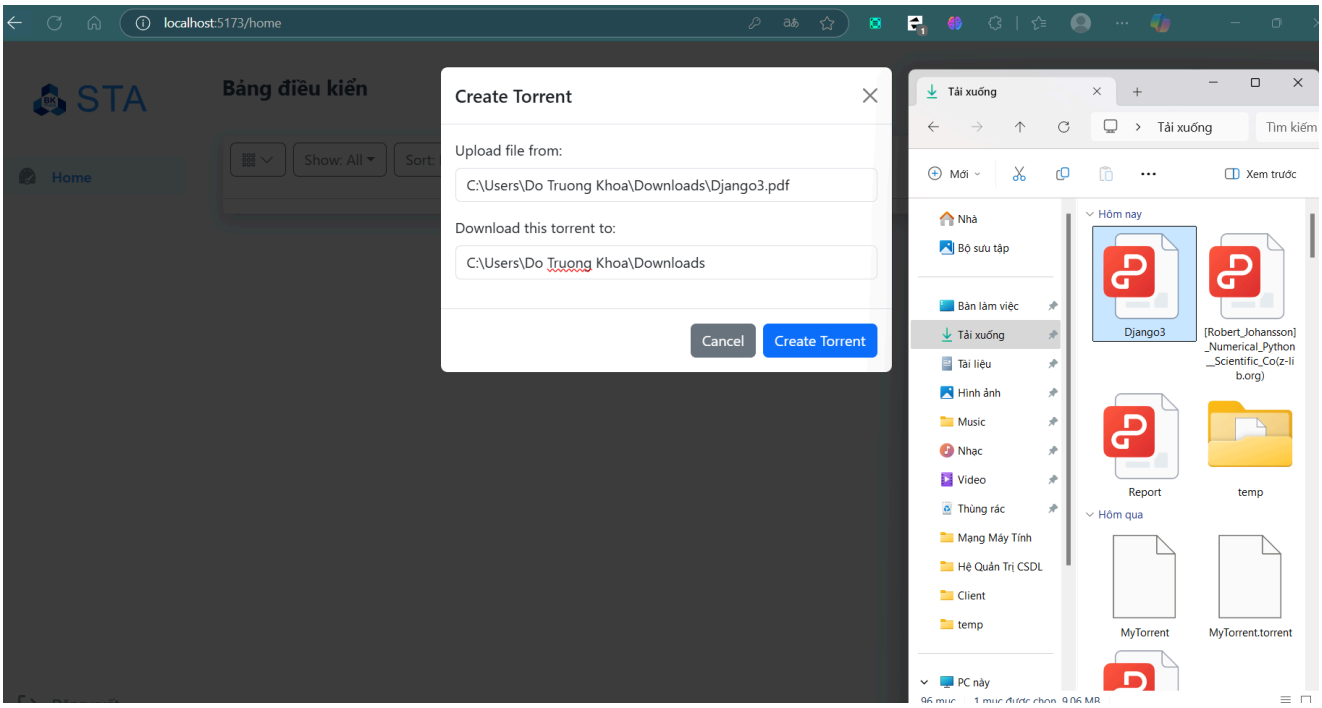


- Tiếp theo mở UI frontend bằng đường link `http://localhost:5173/` trên trình duyệt của bạn
- Nhập tài khoản và mật khẩu

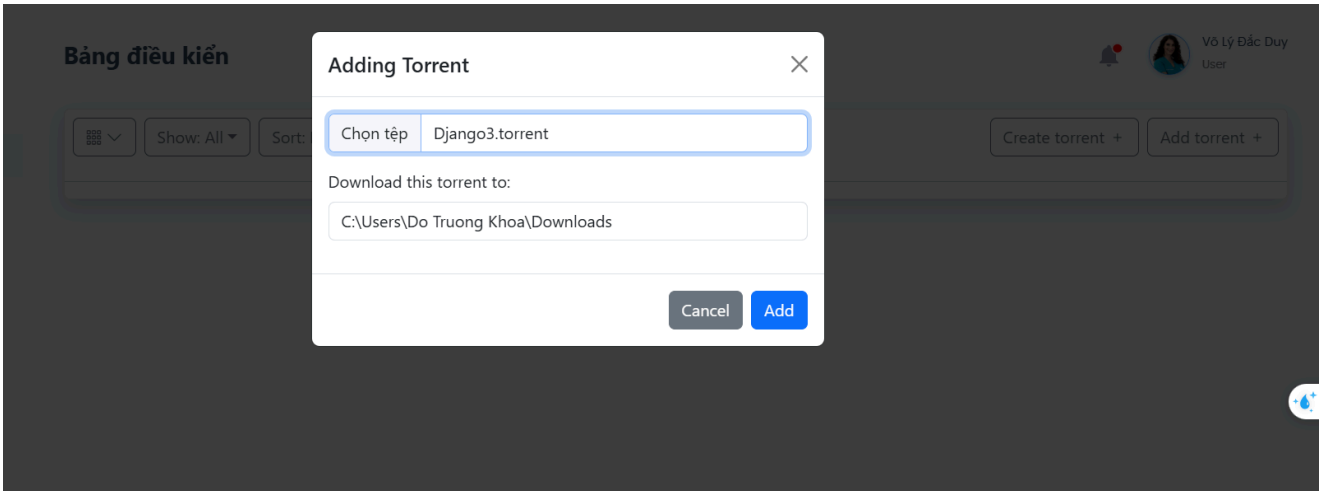
```
Account: 123@gmail.com
Password: 123
```



- Ở góc độ seeder, muốn seed một file bất kỳ bằng cách nhập đường dẫn tới file đó và đường dẫn xuất ra file torrent -> [Nhấn Create torrent] -> nhập đường dẫn tới file -> nhập đường dẫn xuất ra file torrent -> nhấn [Create Torrent]



- Ở máy tôi thì file torrent được tạo ra trong [`C:\Users\Do Truong Khoa\Downloads`]
- gửi file torrent đó cho máy leecher (máy cần tải file).
- Ở góc độ leecher, tải file torrent về và lưu trong máy tính
- Nhấn [Add Torrent] từ trang home và chọn file torrent vừa mới tải về và nhập đường dẫn xuất file sau đó nhấn [Add] và đợi tải về



Lưu ý:

- Đảm bảo rằng máy host đã chạy `py tracker.py` và đang lắng nghe tín hiệu.
- Những peer muốn leech và seed phải được chạy cả frontend lẫn backend như đã hướng dẫn trên.
- Nếu seed thành công thì terminal của seeder sẽ thể thể hiện những cụm từ như `start seeding`..., `generated torrent file at`... và đảm bảo không có lỗi nào hiển thị
- nếu leech thành công thì terminal sẽ hiển thị tiến độ process tải liên tục. `Downloading piece`... cho đến khi hoàn thành

6. Lời cảm ơn

Để hoàn thành đề tài này ngoài sự cố gắng của tập thể nhóm còn có sự hướng dẫn nhiệt tình của thầy Hoàng Lê Hải Thanh. Chúng em chân thành cảm ơn thầy!