



ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN SOCKET
PHẦN MỀM MyDRIVE UPLOAD & DOWNLOAD
FILE MỘT CÁCH ĐA LUỒNG

Môn học: Mạng máy tính

Thực hiện bởi:

23127023 - Phan Nhựt Anh
23127373 - Nguyễn Đình Thái Hưng

Giáo viên hướng dẫn:

Mr. Đỗ Hoàng Cường
Mrs. Huỳnh Thụy Bảo Trân

Hồ Chí Minh, 01 tháng 08 năm 2024

Mục lục

I THÔNG TIN CHUNG	2
1 Thông tin thành viên	2
2 Thông tin khái quát về đồ án	2
II CHƯƠNG TRÌNH	3
1 Phần đăng nhập & đăng ký	3
1.1 Phần đăng nhập	3
1.2 Phần đăng ký	6
2 Phần trang chủ	8
2.1 Giao diện	8
2.2 Chức năng upload file	9
2.3 Chức năng download file	13
III ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH	15
1 Xây dựng chương trình	15
2 Viết báo cáo và tìm kiếm tài liệu	15
3 Quay video demo	15

I. THÔNG TIN CHUNG

1. Thông tin thành viên

MSSV	Họ và Tên	Tài khoản Github	Email
23127023	Phan Nhựt Anh	andreeNewbie	pnanh23@clc.fitus.edu.vn
23127373	Nguyễn Đình Thái Hưng	z3nz3nn	ndthung23@clc.fitus.edu.vn

2. Thông tin khái quát về đồ án

Ngôn ngữ lập trình: HTML, CSS, JavaScripts và Python

API: WebSocket

Giao thức: HTTP và TCP

Database: MongoDB Atlats

Thư viện hỗ trợ: SocketIO, Flask-SocketIO, jwt, pymongo

Mô tả:

- Xây dựng phần mềm MyDrive dựa trên API WebSocket để upload và download file đa luồng dựa trên cơ chế bất đồng bộ của JavaScripts và chức năng chạy tác vụ nền (threads) của thư viện SocketIO phía Python.
- Sử dụng kết hợp HTML, CSS, JavaScripts để xây dựng giao diện trên WebSocket. Trong đó, HTML CSS đóng vai trò chính trong việc xây dựng giao diện GUI. JavaScript vừa xây dựng UX cho giao diện, vừa là phần back-end chạy phía client. Python đóng vai trò server chủ đạo, là nơi giao tiếp với database.
- Giữa server và client kết nối giao tiếp với nhau nhờ API WebSocket và thư viện hỗ trợ FlaskSocketIO, SocketIO thông qua giao thức HTTP và TCP trên đường dẫn LocalHost ở port 3000.
- Sử dụng database MongoDB Atlats để kết nối với server thông qua thư viện hỗ trợ pymongo để lưu trữ tài khoản người dùng và files trên hệ thống MyDrive.

II. CHƯƠNG TRÌNH

1. Phần đăng nhập & đăng ký

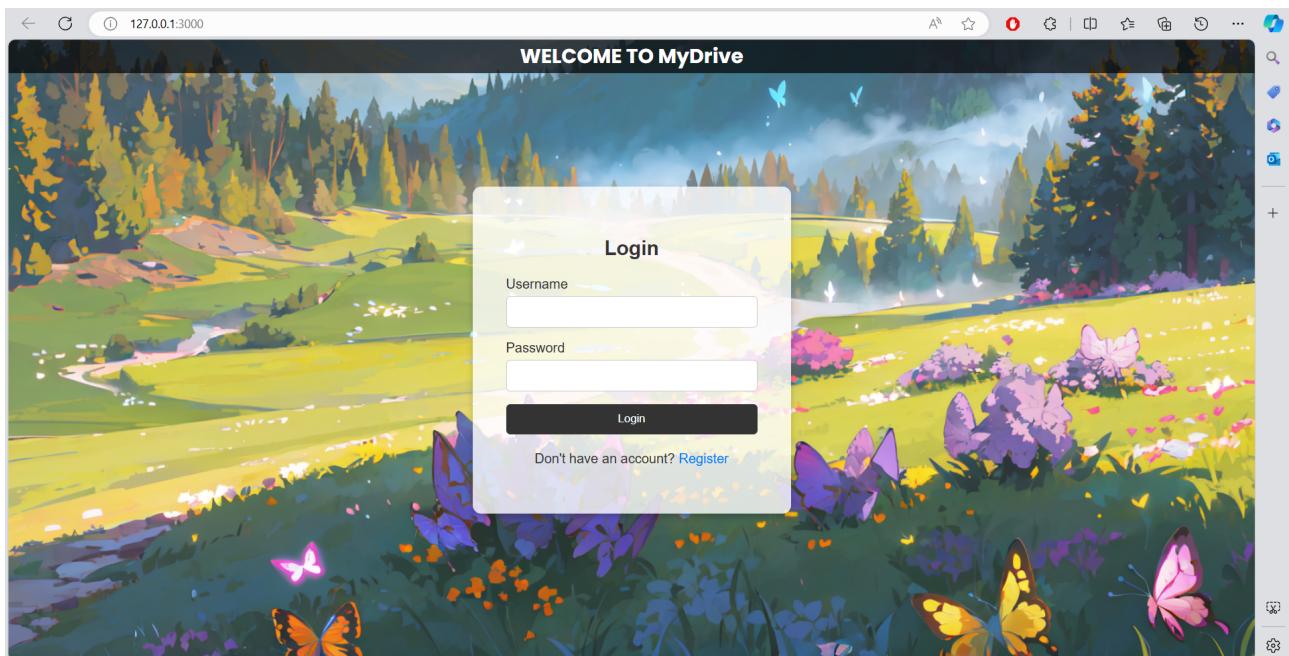
Các file thực hiện: app.py, index.html, style.css và script.js (ở folder static/login)

Các hàm chức năng chung:

- Phía app.py (server):
 - def serve_static(): Định nghĩa một route để phục vụ các tệp tĩnh từ thư mục static.
 - def serve_index(): Định nghĩa một route để phục vụ trang index (trang đăng nhập và đăng ký).
 - def token_required(): Định nghĩa một decorator để bảo vệ các route yêu cầu xác thực token. Hàm này được sử dụng để bảo vệ các route, chỉ cho phép truy cập nếu người dùng cung cấp token hợp lệ.
- Phía script.js (client):
 - registerLink.addEventListener('click', function(event) { ... }): giúp chuyển từ giao diện đặt nhập sang đăng ký khi người dùng click vào link chuyển đổi.
 - loginLink.addEventListener('click', function(event) { ... }): giúp chuyển từ giao diện đăng ký sang đăng nhập khi người dùng click vào link chuyển đổi.
 - body.classList.add('login-mode'): đặt giao diện mặc định là Login khi người dùng vừa truy cập vào Web.

1.1. Phần đăng nhập

Giao diện:



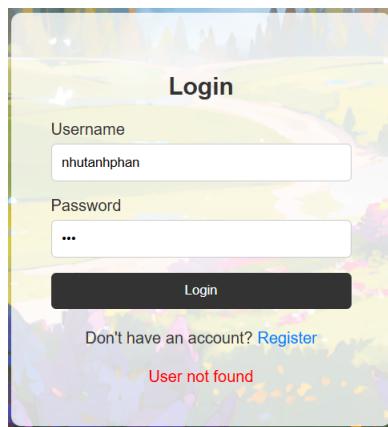
Hình 1: Giao diện đăng nhập

Gồm background và 01 biểu mẫu đăng nhập. Ở biểu mẫu đăng nhập gồm: 002 trường nhập liệu là **Username** và **Password**, 01 button **Login** và 01 link **Register** dẫn qua phần *Dăng ký*.

Các hàm chức năng:

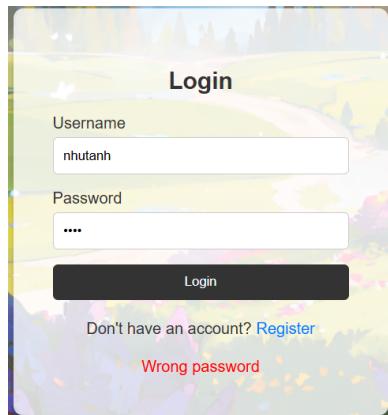
Phía app.py (server):

- def login(): Định nghĩa một route để xử lý yêu cầu đăng nhập bằng phương thức POST.
Lấy dữ liệu (tên đăng nhập, mật khẩu) từ yêu cầu và tiến hành kiểm tra với database người dùng:
 - Nếu người dùng không tồn tại: trả về thông báo lỗi "User not found" và mã trạng thái 400.



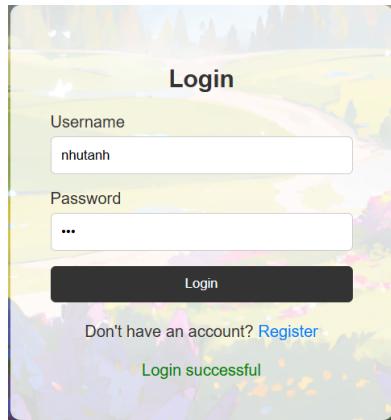
Hình 2: Thông báo lỗi User not found

- Nếu mật khẩu sai: trả về thông báo lỗi "Wrong password" và mã trạng thái 400.



Hình 3: Thông báo lỗi Wrong password

- Nếu tài khoản tồn tại: Trả về thông báo thành công cùng với token và mã trạng thái 200.



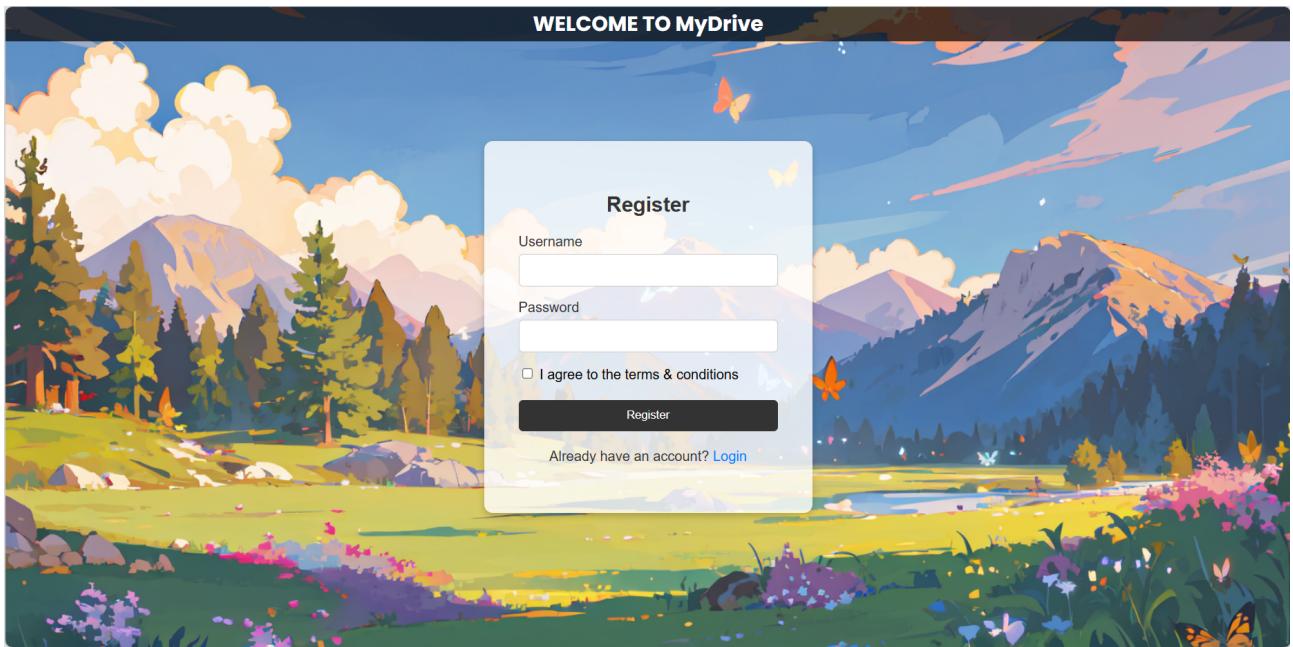
Hình 4: Thông báo Đăng nhập thành công

Phía script.js (client):

- const loginForm = document.getElementById('login-form'): lấy phần tử HTML có id là login-form và gán nó cho biến loginForm
- loginForm.addEventListener('submit', async function(event) { ... }): Gán một hàm xử lý sự kiện cho sự kiện 'submit' của biểu mẫu đăng nhập. Hàm này sẽ được gọi khi người dùng nhấn nút gửi biểu mẫu. Khi hàm được gọi:
 - Lấy giá trị nhập vào từ các trường 'username' và 'password' trong biểu mẫu và gán chúng cho các biến tương ứng.
 - Gửi một yêu cầu POST tới đường dẫn '/login' trên máy chủ cục bộ (localhost) với dữ liệu là tên người dùng và mật khẩu. (Phía máy chủ trên localhost nhận yêu cầu là app.py)
 - Dợi, nhận, chuyển đổi và xử lí phản hồi từ máy chủ:
 - * Nếu yêu cầu thành công: hiển thị thông báo đăng nhập thành công. Lưu token và tên người dùng vào local storage. Chuyển hướng người dùng đến trang chủ. (Minh họa ở Hình 4.)
 - * Nếu yêu cầu không thành công: hiển thị thông báo lỗi từ phản hồi máy chủ. (Minh họa ở Hình 2, Hình 3)
 - * Nếu có lỗi xảy ra trong quá trình gửi yêu cầu hoặc xử lý phản hồi: hiển thị lỗi lên console để lập trình viên xử lí.

1.2. Phần đăng ký

Giao diện:



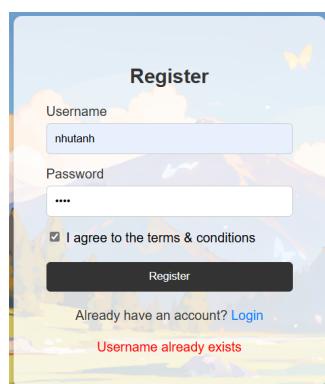
Hình 5: Giao diện đăng ký

Gồm background và 01 biểu mẫu đăng ký. Ở biểu mẫu đăng kí gồm: 01 checkbox, 02 trường nhập liệu là **Username** và **Password**, 01 button **Register** và 01 link **Login** dẫn qua phần *Dăng nhập*.

Các hàm chức năng:

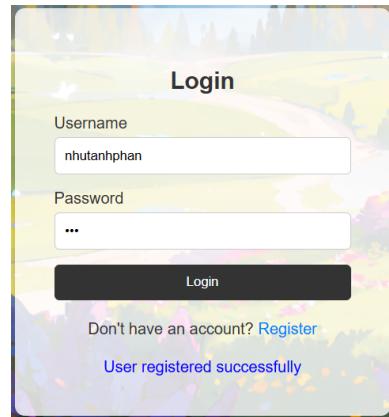
Phía app.py (server):

- def register(): Định nghĩa một route để xử lý yêu cầu đăng nhập bằng phương thức POST. Lấy dữ liệu (tên đăng nhập, mật khẩu) từ yêu cầu và tiến hành kiểm tra với database người dùng:
 - Nếu thiếu tên người dùng hoặc mật khẩu: trả về phản hồi với thông báo "Missing username or password" và mã trạng thái 400.
 - Nếu người dùng đã tồn tại: trả về phản hồi với thông báo "Username already exists" và mã trạng thái 400.



Hình 6: Thông báo Người dùng đã tồn tại

- Nếu đăng ký thành công (tài khoản chưa có trên database): trả về phản hồi với thông báo "User registered successfully", cùng với tên người dùng và mật khẩu, và mã trạng thái 201.



Hình 7: Thông báo Đăng kí thành công và chuyển sang giao diện đăng nhập

Phía script.js (client):

- const registerForm = document.getElementById('register-form'): Lấy phần tử HTML có id là register-form và gán nó cho biến registerForm.
- registerForm.addEventListener('submit', async function(event) { ... }): Gán một hàm xử lý sự kiện cho sự kiện 'submit' của biểu mẫu đăng kí. Hàm này sẽ được gọi khi người dùng nhấn nút gửi biểu mẫu. Khi hàm được gọi:
 - Lấy giá trị nhập vào từ các trường 're-username' và 're-password' trong biểu mẫu và gán chúng cho các biến tương ứng.
 - Gửi một yêu cầu POST tới đường dẫn '/register' trên máy chủ cục bộ (localhost) với dữ liệu là tên người dùng và mật khẩu. (Phía máy chủ trên localhost nhận yêu cầu là app.py)
 - Dợi, nhận, chuyển đổi và xử lý phản hồi từ máy chủ:
 - * Nếu yêu cầu thành công: hiển thị thông báo thành công và cập nhật giao diện để chuyển sang chế độ đăng nhập. (Tham khảo Hình 7)
 - * Nếu yêu cầu không thành công: Hiển thị thông báo lỗi từ phản hồi máy chủ.
 - * Nếu có lỗi xảy ra trong quá trình gửi yêu cầu hoặc xử lý phản hồi: Hiển thị lỗi lên console để lập trình viên xử lí.

2. Phần trang chủ

2.1. Giao diện

The screenshot shows a web browser window with the URL 127.0.0.1:3000/static/homepage/homepage.html. The page title is "MyDrive" and the main heading is "Welcome to MyDrive". On the left, there is a button labeled "+ New". The central part of the page is a table listing files:

Name	Owner	Action
starting out with cplusplus (1).pdf	nhutanh	<button>Download</button>
DSA.pdf	nhutanh	<button>Download</button>
TestAudio.mp3	abcd	<button>Download</button>
Chung khảo (7).jpg	abcd	<button>Download</button>
Chung khảo (7).jpg	abcd	<button>Download</button>
Chung khảo (6).jpg	abc	<button>Download</button>
Chung khảo (6).jpg	abc	<button>Download</button>
Chung khảo (7).jpg	nhutanh	<button>Download</button>

Hình 8: Giao diện Trang chủ

Thành phần giao diện:

- Tên web **MyDrive** (góc phải, phía trên) và tên người dùng (góc trái, phía trên).
- Nút **+New**: người dùng ấn vào đây để upload file.
- Giao diện hiển thị file đã được upload từ nhiều người dùng. Ở mỗi file sẽ hiển thị tên file, người upload file (owner) và nút **Download** - nơi mà người dùng ấn vào để tải file tương ứng.

Các hàm chức năng:

Phía app.py (server):

- def handle_get_files(data): định nghĩa một handler cho sự kiện 'get_files'. Hàm này sẽ xử lý sự kiện 'get_files' được gửi từ client:
 - Tìm và lấy tất cả các tệp được lưu trữ trên database
 - Tạo một danh sách chứa thông tin về các tệp, bao gồm tên tệp (filename), chủ sở hữu (owner), và ID của tệp (_id).
 - Gửi danh sách tệp tới client thông qua sự kiện 'file_list' cùng với mã phiên request.sid

Phía homepage.js (client):

- function fetchFiles(): hàm này sẽ được gọi để gửi yêu cầu lấy danh sách các tệp từ máy chủ thông qua sự kiện 'get_file'.
- socket.on('file_list', (files) => { ... }):
 - Lắng nghe sự kiện 'file_list' từ server và nhận danh sách các tệp (files) được gửi từ server.
 - Tạo ra bảng tệp (hiển thị GUI) thông qua danh sách files nhận được từ server.
 - Thêm vào các button Download ứng với mỗi file và tạo sự kiện 'click' giúp người dùng download file.

Name	Owner	Action
starting out with cplusplus (1).pdf	nhutanh	<button>Download</button>
DSA.pdf	nhutanh	<button>Download</button>
TestAudio.mp3	abcd	<button>Download</button>
Chung khảo (7).jpg	abcd	<button>Download</button>
Chung khảo (7).jpg	abcd	<button>Download</button>
Chung khảo (6).jpg	abc	<button>Download</button>
Chung khảo (6).jpg	abc	<button>Download</button>
Chung khảo (7).jpg	nhutanh	<button>Download</button>

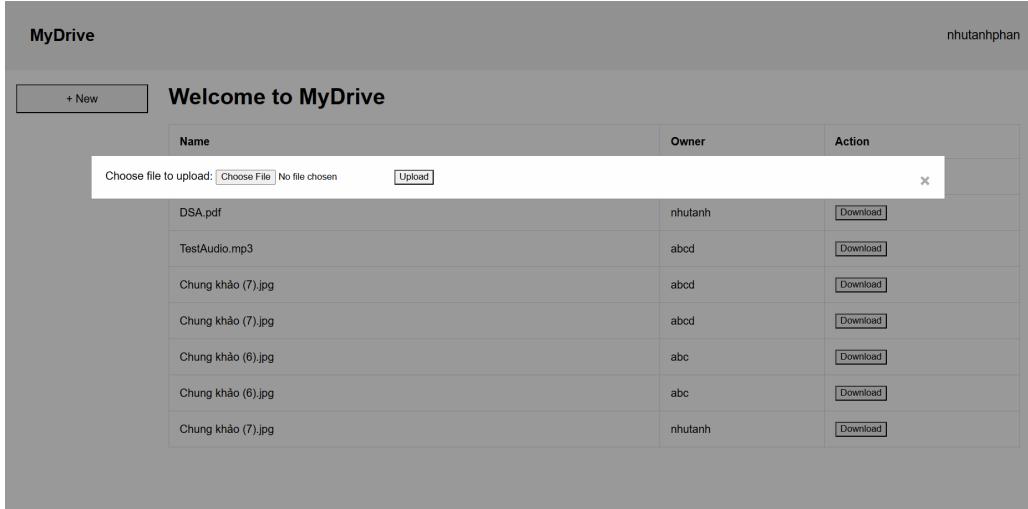
Hình 9: Minh họa cho hàm socket.on('file_list', (files) => { ... })

2.2. Chức năng upload file

Phía client: gồm homepage.js và uploader.js

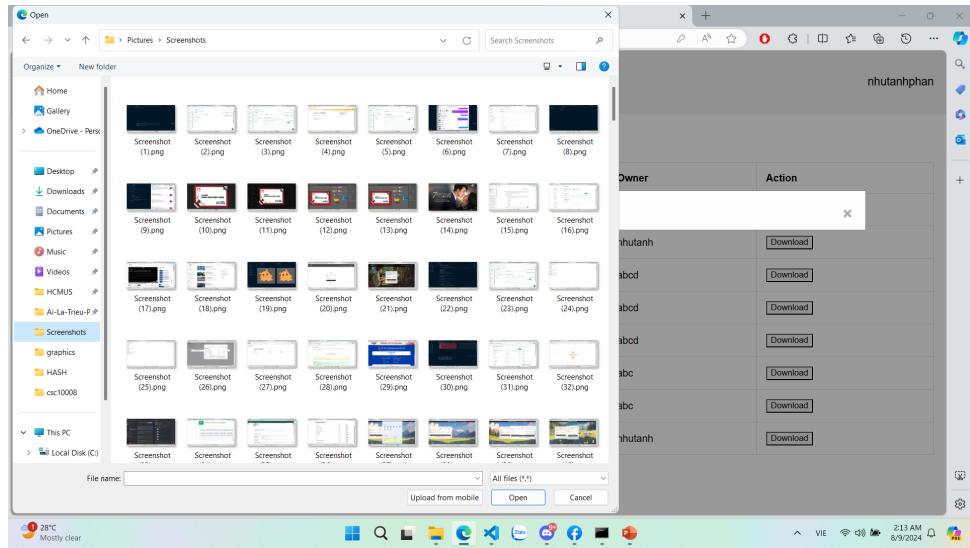
homepage.js:

- newFileButton.addEventListener('click', () => { ... }): khi người dùng ấn vào nút +New thì Upload Modal sẽ hiển thị lên.



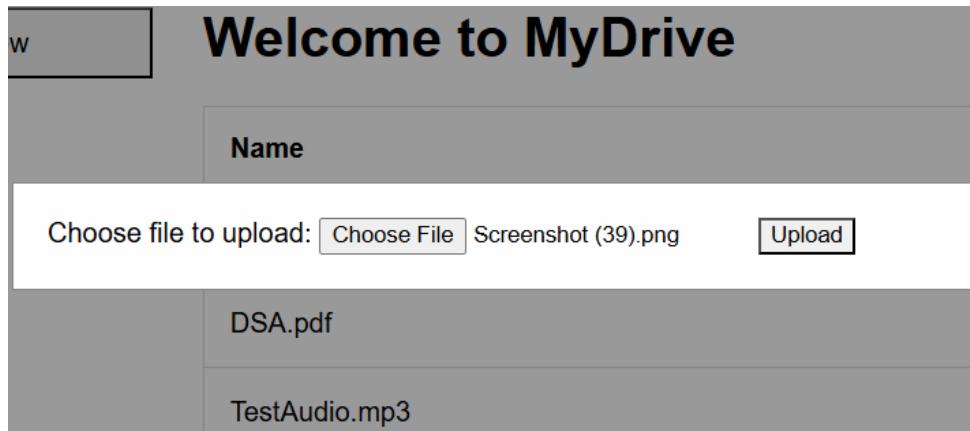
Hình 10: Hiển thị Upload Modal

- closeModal.onclick = () => { ... }): khi người dùng ấn vào nút close (đầu x cuối bên phải Upload Modal) hoặc ấn bấm kì đâu không thuộc phạm vi của Upload Modal thì Upload Modal sẽ ẩn xuống.
- uploadForm.addEventListener('submit', (event) => { ... }): Hàm sẽ hoạt động khi Upload Modal hiện lên.
 - Khi người dùng ấn nút **Choose File**: hiển thị danh sách file từ File Explorer và chặn người dùng click vào nơi khác ngoài phạm vi File Explorer.



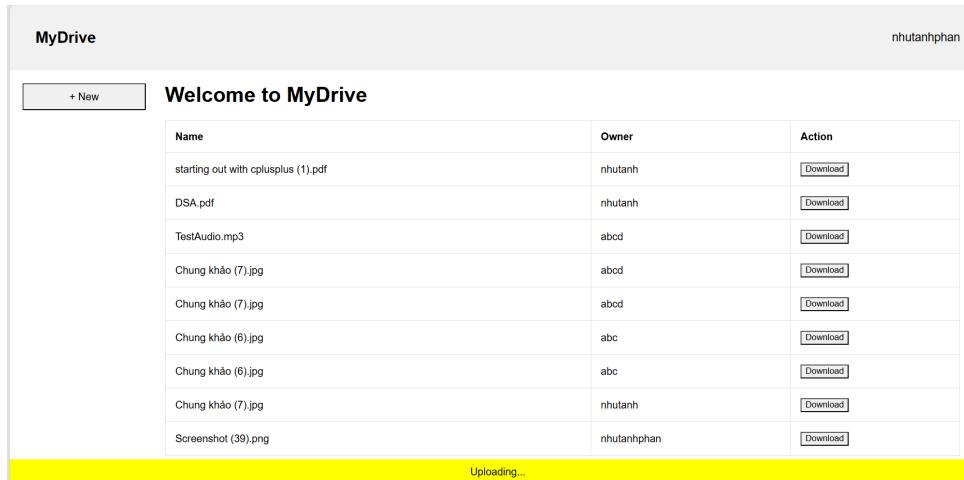
Hình 11: Hiển thị File Explorer từ Upload Modal

- Khi người dùng chọn file xong, tệp sẽ được lấy xuống và hiển thị trên Upload Modal

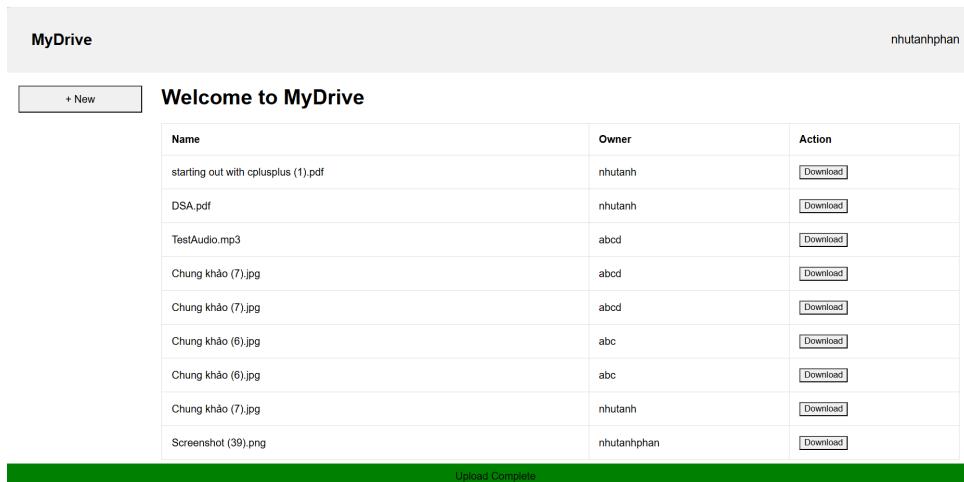


Hình 12: Hiển thị tên file đã được chọn trên Upload Modal

- Khi người dùng ấn nút **Upload** thì file mới được lấy sau đó gửi thông tin file qua server, hiển thị thanh trạng thái (Hình 13 và Hình 14 bên dưới) và gọi *uploader.js* gửi file theo từng segment qua server (*app.py*)
- `socket.on('upload_response', (data) => { ... })`: Lắng nghe sự kiện 'upload_response' từ máy chủ thông qua Socket.IO.
 - Nếu thông báo từ máy chủ là "File uploaded successfully", ẩn thanh trạng thái sau 2 giây, hiển thị hộp thông báo với thông báo từ máy chủ, và gửi yêu cầu lấy danh sách tệp mới từ máy chủ.
 - Nếu có lỗi, hiển thị cảnh báo với thông báo lỗi.
- `function showStatusBar()`: khi người dùng upload file, hiển thị thanh trạng thái thông báo về quá trình.

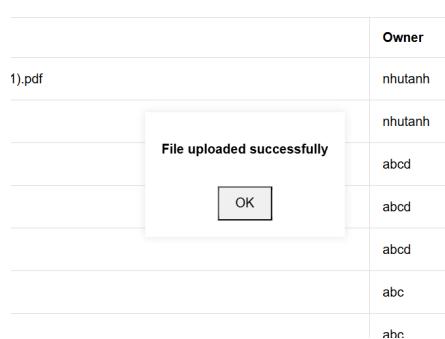


Hình 13: Uploading (Thanh trạng thái màu vàng)



Hình 14: Uploading (Thanh trạng thái màu xanh lá)

- function hideStatusBar(): ẩn thanh trạng
- function showMessageBox(): hiện hộp thoại thông báo khi upload file thành công. Khi người dùng ấn nút OK thì ẩn thanh báo.



Hình 15: Hộp thoại thông báo

- function hideMessageBox(): ẩn hộp thoại thông báo

uploader.js: gửi file đa luồng dựa trên cơ chế bất đồng bộ của ngôn ngữ JavaScript

- function Uploader(): Khởi tạo các thuộc tính gồm:
 - segmentSize: Kích thước của mỗi đoạn tệp. (Nhóm em cài đặt 300KB)
 - file: Đối tượng tệp cần tải lên.
 - fileId: ID duy nhất của tệp.
 - number_of_segments: Số lượng đoạn tệp. (= Kích thước file / kích thước segment và làm tròn lên)
 - threadsQuantity: Số lượng kết nối đồng thời. (Nhóm em cài đặt là 20 luồng 1 lúc để tránh tắt nghẽn)
 - aborted: Biến cờ để kiểm tra xem quá trình tải lên có bị hủy không.
 - uploadedSize: Kích thước đã được tải lên.
 - progressCache: Bộ nhớ đệm cho tiến trình tải lên.
 - activeConnections: Các kết nối đang hoạt động.
 - retryQueue: Hàng đợi cho các đoạn cần tải lại.
- Uploader.prototype.start = function (): Bắt đầu quá trình tải tệp lên:
 - Kiểm tra xem tệp đã được thiết lập chưa.
 - Tạo hàng đợi các đoạn tệp cần gửi.
 - Bắt đầu gửi các đoạn tệp.
- Uploader.prototype.sendNext = function (): Kiểm soát và điều khiển quá trình gửi segment đa luồng thông qua cơ chế bất đồng bộ bằng việc gọi đệ quy chính nó liên tục:
 - Kiểm tra xem quá trình tải lên có bị hủy không. Nếu có thì dừng quá trình tải lên.
 - Kiểm tra số lượng kết nối đang hoạt động. Nếu vượt qua 20 luồng tại một thời điểm thì tạm ngừng gửi segment hiện tại.
 - Nếu không còn đoạn nào trong hàng đợi (chunksQueue) và hàng đợi tải lại (retryQueue), và không còn kết nối đang hoạt động, hoàn thành quá trình tải lên. Còn không lấy segment từng khác hàng đợi.
 - Cắt segment từng file gốc.
 - Gọi phương thức sendChunk để gửi segment vừa cắt.
 - * Nếu segment gửi thành công thì gọi đệ quy chính nó để gửi tiếp.
 - * Nếu segment gửi không thành công thì lưu segment vào retryQueue và gọi đệ quy chính nó để gửi tiếp.
 - Gọi đệ quy chính nó để gửi tiếp.
- Uploader.prototype.sendChunk = function (chunk, id): gửi segment qua server
 - Sử dụng FileReader để đọc đoạn segment cần gửi từ biến chunk đưa vào.
 - Khi đọc xong, gửi đoạn tệp lên máy chủ qua Socket.IO.
 - Lắng nghe phản hồi từ máy chủ.
 - Nếu thành công, đánh dấu là segment đã gửi.
 - Nếu thất bại, đánh dấu là segment chưa được gửi.
- Uploader.prototype.complete = function (error): xử lý các lỗi (nếu segment gửi đi bị lỗi thì sẽ gửi lại) và hoàn thành quá trình upload.
- Uploader.prototype.abort = function (): hủy quá trình tải lên (do lỗi hoặc đã tải lên đủ segment).

Phía server: app.py

- def handle_upload_file_info(data): Xử lý thông tin tệp được gửi từ client trước khi bắt đầu tải lên các đoạn tệp.
 - Kiểm tra token và giải mã nó để lấy thông tin người dùng.

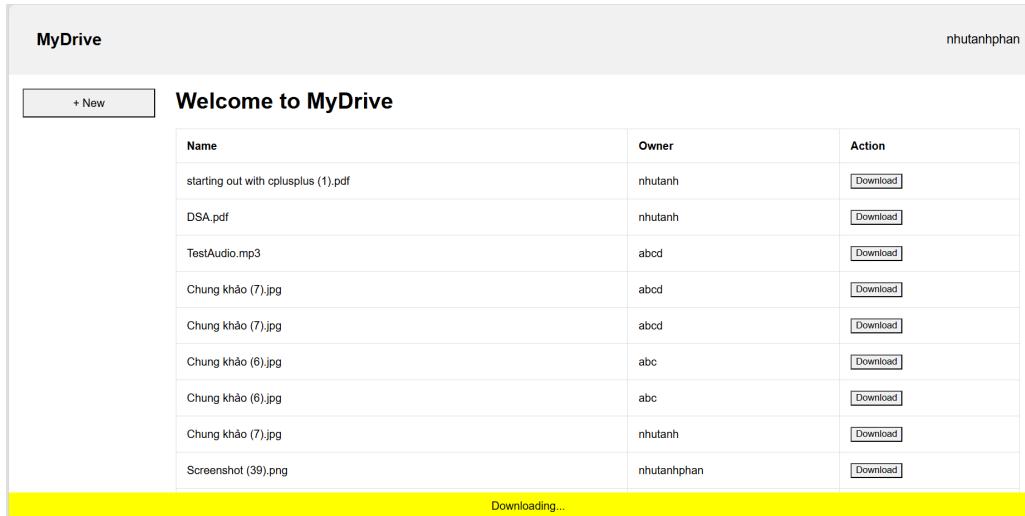
- Lưu trữ thông tin tệp vào file_info.
- Nếu có lỗi, gửi phản hồi lỗi về client.
- def handle_upload_segment(data): Xử lý từng đoạn tệp được gửi từ client.
 - Lấy dữ liệu segment, segment index, và file ID từ dữ liệu nhận được.
 - Kiểm tra xem thông tin file đã được nhận chưa. Nếu chưa, gửi phản hồi lỗi về client. Nếu có thì nhận các segment này. Nếu segment đã được nhận, gửi phản hồi thành công về client.
 - Nếu chưa có thông tin về các segment của file này, khởi tạo danh sách file_segments với kích thước tương ứng với số lượng segment.
 - Bắt đầu tác vụ nền (luồng) để lưu segment vào dữ liệu tạm thời.
 - Kiểm tra nếu tất cả các segment đã được nhận, ghép nối và lưu file vào cơ sở dữ liệu, gửi phản hồi thành công về client, và xóa thông tin tạm thời về các segment và thông tin file.
- def save_segment(segment_index, segment_data, file_id): Lưu trữ đoạn tệp vào bộ nhớ tạm thời tại vị trí tương ứng với segment_index.
- def write_file(info, segments): Ghép nối tất cả các đoạn tệp thành dữ liệu tệp hoàn chỉnh. Lưu trữ tệp hoàn chỉnh vào cơ sở dữ liệu GridFS với tên tệp và thông tin chủ sở hữu.

2.3. Chức năng download file

Phía client: Bao gồm homepage.js và downloader.js

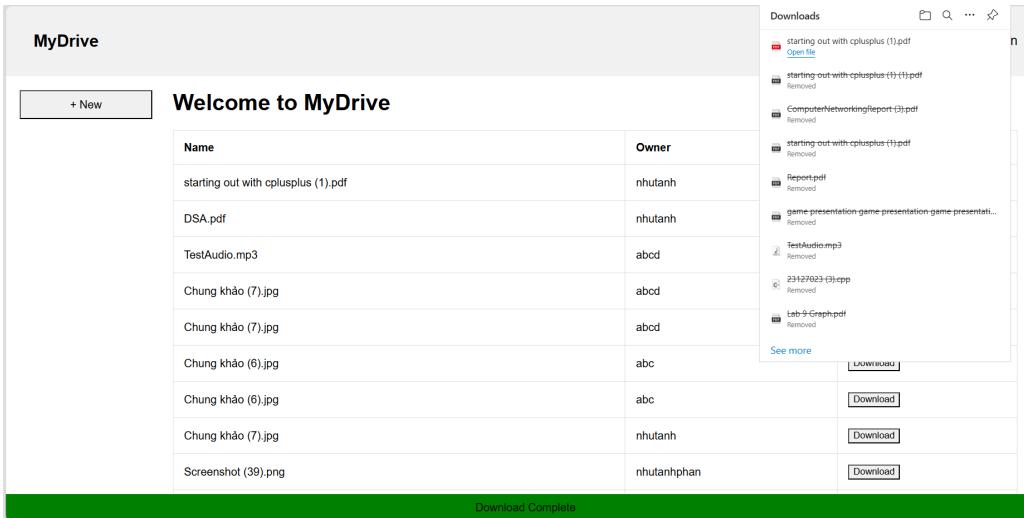
homepage.js

- Khi người dùng click vào nút download ứng với mỗi file đã nói ở phần. Lệnh `socket.emit ('download_file_info', token: token, file_id: fileId)`; sẽ chạy và gửi thông tin file cần download cho server, khởi động quá trình tải xuống. Hiển thị thanh trạng thái khi quá trình bắt đầu bằng hàm `showStatusBar()` đã nói ở phần 2.2



Hình 16: Downloading (Thanh trạng thái màu vàng)

- `socket.on('download_file_info', (data) => { ... })`: lắng nghe sự kiện 'download_file_info' từ server, nhận thông tin file tải xuống và gọi `downloader.js` để tải file đa luồng theo từng segment.
- `socket.on('download_response', (data) => { ... })`: Nhận phản hồi download thành công từ máy chủ và cập nhật thanh trạng thái, ẩn sau đó 2 giây. Kết thúc quá trình tải xuống.



Hình 17: Download thành công (Thanh trạng thái màu xanh)

downloader.js

- function Downloader(): khởi tạo các thuộc tính phụ vụ cho quá trình tải xuống
 - segmentSize: Kích thước của mỗi segment. (300KB)
 - fileId: ID của file cần tải xuống.
 - fileName: Tên của file cần tải xuống.
 - numberOfSegments: Số lượng segment.
 - downloadedSegments: Mảng chứa các segment đã tải xuống.
 - threadsQuantity: Số lượng kết nối đồng thời. (luồng)
 - activeConnections: Các kết nối đang hoạt động.
 - aborted: Biến cờ để kiểm tra xem quá trình tải xuống có bị hủy không.
 - dataFile: Đối tượng Blob để lưu trữ dữ liệu file.
 - chunksQueue: Hàng đợi các segment cần tải xuống.
 - retryQueue: Hàng đợi các segment cần tải lại.
- Downloader.prototype.start = function ()
 - Khởi tạo đối tượng Blob để lưu trữ dữ liệu file.
 - Tạo hàng đợi các segment cần .
 - Bắt đầu tải xuống các segment.
- Downloader.prototype.downNext = function (): thực hiện và quản lý quá trình download segment đa luồng bằng cách gọi đệ quy chính nó liên tục dựa trên cơ chế bất đồng bộ.
 - Kiểm tra xem quá trình tải xuống có bị hủy không. Nếu có thì dừng quá trình tải xuống.
 - Kiểm tra số lượng kết nối đang hoạt động. Điều tiết luồng hoạt động trách tắc nghẽn.
 - Lấy segment từ chunksQueue hoặc retryQueue.
 - Gọi hàm downloadSegment để tải segment xuống và đánh dấu segment đang được tải xuống.
- Downloader.prototype.downloadSegment = function (segmentIndex): tải segment xuống từ
 - Tạo một promise để quản lý quá trình tải xuống segment.
 - Gửi yêu cầu tải xuống segment đến máy chủ thông qua Socket.IO.
 - Lắng nghe phản hồi từ máy chủ.
 - Nếu tải xuống thành công, lưu segment lại, đánh dấu là đã tải segment xuống, resolve promise.

- Nếu tải xuống thất bại, đưa segment vào retryQueue chờ tải lại.
- Downloader.prototype.completeDownload = function (): Hoàn thành quá trình tải xuống và lưu trữ file vào máy khách. (Tham khảo Hình 16 và Hình 17)
 - Sắp xếp các segment theo đúng thứ tự và ghép nối chúng thành đối tượng Blob.
 - Cập nhật thanh trang thái để hiển thị quá trình tải xuống đã hoàn thành.
 - Tạo URL cho file đã tải xuống và tự động tải file về máy khách.
 - Ẩn thanh trạng thái sau 2 giây.

III. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

1. Xây dựng chương trình

NỘI DUNG	NGƯỜI THỰC HIỆN	HOÀN THÀNH
Chương trình đăng nhập và đăng ký	Nguyễn Đình Thái Hưng, Phan Nhựt Anh	100%
Giao diện trang chủ	Nguyễn Đình Thái Hưng	100%
Xây dựng kết nối giữa phía client (JS) và phía server (Py)	Nguyễn Đình Thái Hưng, Phan Nhựt Anh	100%
Chức năng upload và download file	Phan Nhựt Anh, Nguyễn Đình Thái Hưng	100%
Xây dựng code đa luồng	Phan Nhựt Anh	100%
Xây dựng và kết nối database	Phan Nhựt Anh	100%

2. Viết báo cáo và tìm kiếm tài liệu

Người thực hiện: Phan Nhựt Anh

3. Quay video demo

Người thực hiện: Phan Nhựt Anh, Nguyễn Đình Thái Hưng

Video demo: đính kèm link Youtube và Google Drive (dự phòng) trong báo cáo này

Link Youtube: <https://www.youtube.com/VideoDemoMMT>

Link Google Drive: <https://drive.google.com/VideoDemoMMT>

Tài liệu tham khảo

- [1] <https://chat.openai.com>.
- [2] <https://socket.io/>.
- [3] Login Form in HTML & CSS <https://youtu.be/hlwIM4a5rxg?si=diinV1mhGm5YrX1i>
- [4] Intro to MongoDB Atlas in 10 mins | Jumpstart https://youtu.be/xrc7dI0_tXk?si=GzyBptn0oP210k3Z.
- [5] HTML & CSS Full Course - Beginner to Pro <https://youtu.be/G3e-cpL7ofc?si=xJAfndUo4c8QyTVr>
- [6] Python Socket Programming Tutorial <https://youtu.be/3QiPPX-KeSc?si=SpU8SsdfXTEnqoLH>
- [7] Multithreaded Client Server in Python || Socket Programming in Python https://youtu.be/xceTFWy_eag?si=UrpoobuZRcn1mEtU
- [8] Các tài liệu đã được cung cấp bởi giáo viên trong môn học.