**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP.HCM**

**VIỆN ĐÀO TẠO CHẤT LƯỢNG CAO**

∙∙•🙞🖎🕮✍🙜•∙∙

A black and red text

AI-generated content may be incorrect.

**MÔN: LẬP TRÌNH MẠNG**

**BÁO CÁO NGHIÊN CỨU:**

**“LẬP TRÌNH ỨNG DỤNG UPLOAD/DOWNLOAD MULTI FILE (PAUSE, STOP, RESUME)”**

**GVHD**: Bùi Dương Thế

**Nhóm thực hiện**: Nhóm 6

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** |
| 1 | Hoàng Công Minh | 038205013332 |
| 2 | Nguyễn Đức Minh | 068205010134 |
| 3 | Châu Hồng Vũ | 082205006232 |
| 4 | Nguyễn Tiến Vươn | 052205011004 |

**TP. Hồ Chí Minh, tháng 8 năm 2025**

**Mục Lục**

[**Lời mở đầu** 4](#_Toc207310460)

[**1. Giới thiệu đề tài** 5](#_Toc207310461)

[**2. Phạm vi & Giới hạn** 5](#_Toc207310462)

[**2.1. Phạm vi** 5](#_Toc207310463)

[**2.2. Giới hạn** 5](#_Toc207310464)

[**3. Mục tiêu đề tài** 5](#_Toc207310465)

[**3.1. Mục tiêu tổng quát** 5](#_Toc207310466)

[**3.2. Mục tiêu cụ thể** 6](#_Toc207310467)

[**4. Mục đích chọn dự án** 6](#_Toc207310468)

[**5. Cơ sở lý thuyết** 6](#_Toc207310469)

[**5.1.** **Kiến trúc hệ thống** 6](#_Toc207310470)

[**5.2.** **Backend Technologies** 7](#_Toc207310471)

[**5.3.** **Frontend Technologies** 7](#_Toc207310472)

[**5.4.** **Database Schema Users: thông tin người dùng.** 7](#_Toc207310473)

[**5.5.** **Bảo mật Password Hashing (PBKDF2 + salt).** 8](#_Toc207310474)

[**6. Thiết kế & Kiến trúc** 8](#_Toc207310475)

[**6.1.** **Sơ đồ kiến trúc** 8](#_Toc207310476)

[**6.2.** **API Flask – File Manager** 9](#_Toc207310477)

[**6.3. Luồng Upload** 9](#_Toc207310478)

[**6.4.** **Luồng Download** 10](#_Toc207310479)

[**6.5.** **Logging & Theo dõi** 10](#_Toc207310480)

[**6.6.** **Edge Cases** 11](#_Toc207310481)

[**7. Thực hành / Thực nghiệm** 11](#_Toc207310482)

[**7.1. Thiết kế cơ sở dữ liệu** 11](#_Toc207310483)

[**7.2. Xây dựng server** 12](#_Toc207310484)

[**7.3. Xây dựng giao diện frontend** 12](#_Toc207310485)

[7.3.1 Giao diện chức năng sản phẩm 13](#_Toc207310486)

[7.3.2 Giao diện quản lý file và folder 13](#_Toc207310487)

[**7.4. Tích hợp chức năng quản lý file** 13](#_Toc207310488)

[**7.5. Các kịch bản thử nghiệm chính** 14](#_Toc207310489)

[**7.6. Đánh giá hiệu năng** 14](#_Toc207310490)

[**7.7. Nhật ký thử nghiệm** 14](#_Toc207310491)

[**7.8. Sơ đồ kiến trúc triển khai** 15](#_Toc207310492)

[**8. Kinh nghiệm rút ra** 16](#_Toc207310493)

[**8.1. Về kỹ thuật** 16](#_Toc207310494)

[**8.2. Về kỹ năng làm việc nhóm** 16](#_Toc207310495)

[**8.3. Bài học quan trọng** 16](#_Toc207310496)

[**9. Kết quả đạt được** 17](#_Toc207310497)

[**9.1. Hoàn thiện chức năng chính** 17](#_Toc207310498)

[**9.2. Đảm bảo tính ổn định** 17](#_Toc207310499)

[**9.3. Toàn vẹn và an toàn dữ liệu** 17](#_Toc207310500)

[**9.4. Kinh nghiệm học được** 17](#_Toc207310501)

[**10. Kết quả chưa đạt** 17](#_Toc207310502)

[**10.1. Về chức năng** 18](#_Toc207310503)

[**10.2. Về hiệu năng** 18](#_Toc207310504)

[**10.3. Về bảo mật** 18](#_Toc207310505)

[**10.4. Về khả năng mở rộng** 18](#_Toc207310506)

[**11. Hướng phát triển** 18](#_Toc207310507)

[**11.1. Hoàn thiện chức năng** 18](#_Toc207310508)

[**11.2. Nâng cao bảo mật** 19](#_Toc207310509)

[**11.3. Cải thiện giao diện** 19](#_Toc207310510)

[**11.4. Tối ưu hiệu năng** 19](#_Toc207310511)

[**11.5. Khả năng mở rộng** 19](#_Toc207310512)

[**Tài liệu tham khảo** 20](#_Toc207310513)

# **Lời mở đầu**

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, nhu cầu truyền tải dữ liệu nhanh chóng, an toàn và ổn định ngày càng trở nên cấp thiết. Người dùng không chỉ cần tải xuống những tệp dữ liệu nhỏ mà còn thường xuyên phải xử lý các tập tin dung lượng lớn, đồng thời yêu cầu hệ thống có thể hoạt động liên tục, không bị gián đoạn. Các công cụ truyền tải truyền thống thường bộc lộ nhiều hạn chế, như thiếu chức năng tạm dừng hoặc tiếp tục, khó quản lý nhiều tập tin cùng lúc, hay không đảm bảo toàn vẹn dữ liệu khi có sự cố mạng.

Xuất phát từ thực tế đó, nhóm chúng em đã lựa chọn đề tài “Lập Trình Ứng Dụng Download/Upload Multi File có Pause / Resume / Start / Stop”. Đây vừa là một cơ hội để áp dụng kiến thức đã học về lập trình mạng, vừa là một thách thức giúp chúng em rèn luyện kỹ năng thiết kế và phát triển một ứng dụng có tính thực tiễn cao. Đề tài không chỉ dừng lại ở mức độ minh họa lý thuyết mà còn được triển khai thành sản phẩm cụ thể, có khả năng ứng dụng trong các tình huống thực tế.

# **1. Giới thiệu đề tài**

Trong kỷ nguyên công nghệ số, dữ liệu ngày càng đóng vai trò quan trọng trong công việc, học tập và đời sống. Người dùng không chỉ làm việc với những file nhỏ, mà còn thường xuyên phải xử lý nhiều file dung lượng lớn cùng lúc. Các công cụ truyền thống thường gặp nhiều hạn chế: nếu mất kết nối mạng thì phải tải lại từ đầu, tốc độ thiếu ổn định, khó quản lý khi số lượng file nhiều, hoặc không hỗ trợ các chức năng tiện ích như tạm dừng (pause), tiếp tục (resume), dừng hẳn (stop), hay hiển thị tiến trình chi tiết.

Đề tài của nhóm nhằm khắc phục những nhược điểm này. Chúng em đã xây dựng một ứng dụng web cho phép người dùng Upload/Download nhiều file đồng thời với khả năng Pause/Resume/Stop từng file riêng biệt, hiển thị tiến độ theo thời gian thực, và có cơ chế bảo toàn dữ liệu cũng như cách ly theo người dùng để đảm bảo an toàn.

Ứng dụng được thiết kế theo mô hình Client–Server, trong đó phía client là giao diện web thân thiện, phía server gồm hai phần: WebSocket Server để quản lý upload theo chunk bất đồng bộ, và Flask File Manager để quản lý API, xác thực, và chức năng download. Thông tin metadata và người dùng được lưu trữ trong cơ sở dữ liệu SQLite.

# **2. Phạm vi & Giới hạn**

## **2.1. Phạm vi**

**Ứng dụng tập trung vào 3 khía cạnh chính:**

1. **Upload nhiều file:** có khả năng Pause/Resume/Stop, hiển thị chi tiết tiến trình từng file.
2. **Download nhiều file:** hỗ trợ tải đồng thời, quản lý trạng thái tải, tuy nhiên chưa có Pause/Resume cho download.
3. **Quản lý file cá nhân:** đăng nhập/đăng ký, quản lý file và thư mục, thùng rác, preview nhiều định dạng, thống kê dung lượng.

## **2.2. Giới hạn**

**Giới hạn kỹ thuật hiện tại:**

* **Download**: theo dõi trạng thái và kết quả; *chưa hỗ trợ Pause/Resume/Stop*.
* **Bảo mật truyền tải**: *chưa bật* SSL/TLS theo mặc định (kiến trúc sẵn sàng).
* **Hiệu năng file rất lớn (>1GB)**: cần tối ưu chunk-size/concurrency và I/O nâng cao.
* **Chia sẻ liên-user & phân quyền chi tiết**: chưa triển khai; hiện tại dữ liệu cách ly theo user.

# **3. Mục tiêu đề tài**

## **3.1. Mục tiêu tổng quát**

Xây dựng một ứng dụng web hỗ trợ upload nhiều tệp tin dung lượng lớn một cách nhanh chóng, ổn định, có khả năng tạm dừng, tiếp tục và quản lý file thân thiện.

## **3.2. Mục tiêu cụ thể**

* Hỗ trợ upload nhiều file cùng lúc với Start/Pause/Resume/Stop.
* Resume upload từ đúng offset khi bị gián đoạn.
* Cho phép người dùng quản lý file riêng, có thư mục, thùng rác, preview và thống kê.
* Đảm bảo bảo mật cơ bản qua password hashing, token, user isolation.
* Kiến trúc hệ thống dễ mở rộng (hỗ trợ cloud, UI hiện đại, storage backend mới).

# **4. Mục đích chọn dự án**

Nhóm lựa chọn đề tài này vì nhận thấy nhu cầu thực tế rất lớn về một hệ thống truyền tải file tiện lợi. Ngoài việc giúp ích cho người dùng, dự án còn mang lại giá trị học thuật cao. Đây là cơ hội để vận dụng kiến thức về lập trình mạng, bất đồng bộ, bảo mật, thiết kế database, logging, và thực hành teamwork qua GitHub.

Mục đích còn là để tạo ra một sản phẩm thực tế có thể triển khai thử nghiệm trong môi trường làm việc nhóm hoặc lớp học, phục vụ cả nhu cầu cá nhân lẫn tổ chức.

Ý nghĩa học thuật: Vận dụng socket/WebSocket, AsyncIO, thiết kế client–server.Thực hành teamwork, Git, logging, quản trị phiên.Rèn kỹ năng viết báo cáo, làm việc nhóm và quản lý dự án phần mềm.

# **5. Cơ sở lý thuyết**

## **5.1.** **Kiến trúc hệ thống**

Hệ thống được thiết kế theo hướng **tách biệt thành hai dịch vụ chính** thay vì gói gọn trong một tiến trình duy nhất. Cụ thể:

* **WebSocket Server** chịu trách nhiệm quản lý việc upload file theo thời gian thực theo cơ chế chunk + offset.
* **File Manager Server** (Flask) xử lý các API liên quan đến quản lý dữ liệu, tải file về máy người dùng, và xác thực người dùng.

Việc tách biệt này giúp ứng dụng dễ bảo trì và mở rộng từng phần mà không ảnh hưởng toàn bộ hệ thống. Tuy chưa phải là “microservices hoàn chỉnh” nhưng đã đảm bảo tính độc lập giữa upload và quản lý dữ liệu.

Bên cạnh đó, hệ thống tuân theo mô hình **Client–Server Architecture**: frontend (giao diện web) giao tiếp với backend qua hai kênh chính:

* **HTTP REST API**: dùng cho các thao tác dữ liệu thông thường như đăng nhập, quản lý file, tạo thư mục, xóa/khôi phục.
* **WebSocket**: dùng cho việc truyền tải dữ liệu thời gian thực như tiến độ upload và các sự kiện pause, resume, stop.

Ngoài ra, kiến trúc còn dựa theo mô hình **Event-driven Architecture**. Các hành động như bắt đầu upload, tạm dừng, tiếp tục, hoàn tất được phát sinh dưới dạng sự kiện. Các sự kiện này được WebSocket server gửi ngay tới client, giúp giao diện luôn được cập nhật realtime mà không cần reload trang.

## **5.2.** **Backend Technologies**

Hệ thống backend được phát triển dựa trên các công nghệ hiện đại, đảm bảo vừa nhẹ, vừa linh hoạt trong triển khai:

* **Python 3.x**: ngôn ngữ chính để xây dựng backend, dễ đọc, dễ bảo trì, hệ sinh thái thư viện phong phú.
* **Flask Framework**: đóng vai trò RESTful API backend, xử lý các yêu cầu HTTP như đăng nhập, đăng ký, quản lý file metadata, xóa/khôi phục file. Flask được chọn vì đơn giản, dễ mở rộng bằng extensions.
* **WebSocket + AsyncIO**: cho phép server xử lý upload file bất đồng bộ. AsyncIO giúp nhiều tiến trình upload song song, giảm độ trễ và nâng cao hiệu năng.
* **SQLite Database**: cơ sở dữ liệu nhẹ, dễ triển khai mà vẫn đủ tính năng. Rất phù hợp với môi trường demo hoặc ứng dụng vừa – nhỏ, không cần cài đặt phức tạp.

## **5.3.** **Frontend Technologies**

Giao diện người dùng được xây dựng với mục tiêu đơn giản, trực quan, dễ sử dụng, nhưng vẫn đảm bảo các tính năng nâng cao:

* **HTML5 và CSS3:** dùng để xây dựng bố cục, định dạng và tạo phong cách cho giao diện. CSS3 được tận dụng với Flexbox và Grid để bố trí linh hoạt, cùng với animations để tạo hiệu ứng hiện đại.
* **JavaScript ES6+:** đảm nhận toàn bộ phần xử lý logic phía client, bao gồm gọi API, quản lý upload và cập nhật trạng thái giao diện.
* **WebSocket API:** cho phép trình duyệt giao tiếp trực tiếp với server theo thời gian thực.
* **File API:** hỗ trợ tính năng kéo – thả (drag & drop), đọc thông tin file trước khi upload.
* **LocalStorage API:** lưu trữ dữ liệu tạm thời trên client, giúp duy trì trạng thái người dùng hoặc hỗ trợ tính năng resume upload khi bị gián đoạn.

## **5.4.** **Database Schema Users: thông tin người dùng.**

Cơ sở dữ liệu của hệ thống được thiết kế với bốn bảng chính, đảm bảo việc quản lý người dùng và file rõ ràng, có quan hệ ràng buộc chặt chẽ:

* **Users:** lưu trữ thông tin người dùng gồm user\_id, username, password\_hash và role (admin hoặc user). Đây là bảng trung tâm để xác thực và phân quyền.
* **Sessions:** quản lý các phiên đăng nhập, gồm session\_id, user\_id, token và expiry. Cách làm này giúp người dùng duy trì đăng nhập an toàn mà không cần nhập lại mật khẩu nhiều lần.
* **Files:** chứa metadata của từng file như file\_id, filename, size, upload\_date, đồng thời liên kết với user\_id và folder\_id để đảm bảo mỗi file thuộc về đúng người dùng và đúng thư mục.
* **Folders:** hỗ trợ cấu trúc thư mục lồng nhau, với trường folder\_id, parent\_id và user\_id. Nhờ đó, người dùng có thể sắp xếp file khoa học thay vì chỉ để chung một nơi.

Thiết kế database này đảm bảo tính tách biệt người dùng (user isolation), tránh trường hợp người dùng này nhìn thấy dữ liệu của người dùng khác.

## **5.5.** **Bảo mật Password Hashing (PBKDF2 + salt).**

Vì hệ thống liên quan trực tiếp đến dữ liệu cá nhân và file của người dùng, nhóm đã chú trọng đến việc triển khai các cơ chế bảo mật ngay từ đầu:

* **Password Hashing (PBKDF2 + salt)**: mật khẩu không được lưu trực tiếp trong database. Thay vào đó, hệ thống sử dụng thuật toán PBKDF2 với muối (salt) để tăng cường khả năng chống lại các cuộc tấn công brute-force.
* **Token-based Authentication**: khi người dùng đăng nhập, hệ thống cấp một token thay vì duy trì session truyền thống. Token này được gửi kèm trong các yêu cầu API, giúp tăng tính bảo mật và tiện lợi.
* **Role-based Access Control (RBAC):** hiện tại mới dừng ở mức cơ bản. User chỉ thao tác file của mình; admin có thể xem toàn bộ, nhưng tính năng phân quyền nâng cao chưa triển khai chi tiết.
* **Input Validation & CORS**: tất cả dữ liệu đầu vào (đặc biệt là tên file, đường dẫn) đều được kiểm tra để tránh injection. Đồng thời, cấu hình CORS giúp kiểm soát nguồn truy cập từ frontend đến backend, ngăn chặn tấn công cross-origin.

Nhờ các cơ chế trên, hệ thống vừa đảm bảo tính an toàn dữ liệu, vừa cung cấp trải nghiệm mượt mà cho người dùng.

# **6. Thiết kế & Kiến trúc**

Trong phần này, nhóm sẽ trình bày chi tiết hơn về thiết kế tổng thể và kiến trúc của hệ thống. Không chỉ dừng lại ở sơ đồ tóm tắt, chúng em phân tích từng thành phần, cách chúng tương tác với nhau, cũng như lý do lựa chọn công nghệ.

## **6.1.** **Sơ đồ kiến trúc**

Hệ thống được xây dựng theo mô hình Client–Server, trong đó Client (giao diện web) kết nối với Backend qua hai kênh: REST API và WebSocket. Sự phân tách này giúp các chức năng quản lý file và truyền tải dữ liệu hoạt động độc lập nhưng vẫn đồng bộ với nhau. WebSocket Server chịu trách nhiệm cho toàn bộ quy trình upload file theo cơ chế chia nhỏ thành chunk, trong khi Flask File Manager đảm nhiệm các tác vụ như quản lý metadata, xác thực, thống kê, và cung cấp file cho người dùng tải xuống. CSDL SQLite là nơi lưu trữ thông tin người dùng, file, session, và trạng thái recycle bin. File thực tế được lưu trong các thư mục riêng biệt của từng người dùng để đảm bảo tính cách ly.

A diagram of a computer

AI-generated content may be incorrect.

Hình 6.1 Sơ đồ kiến trúc tổng thể

## **6.2.** **API Flask – File Manager**

File Manager cung cấp nhiều nhóm API:

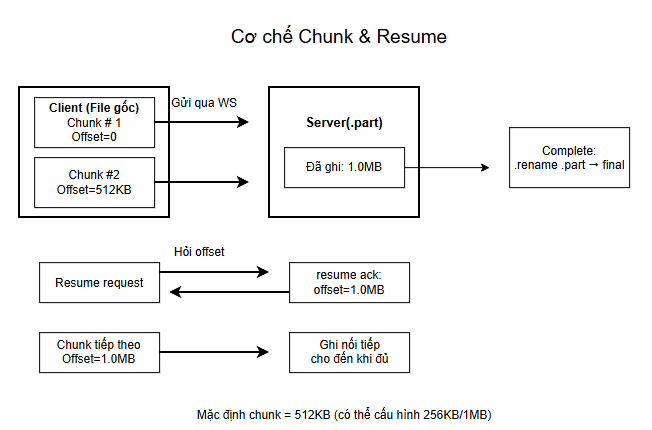
* **Auth**: /login, /register, /logout.
* **File/Folder**: tải về, đổi tên, xóa mềm, khôi phục, di chuyển, tạo thư mục mới.
* **Preview**: trả về nội dung xem nhanh các file như ảnh, PDF, video, text.
* **Stats**: thống kê số file, dung lượng, số thư mục.
* **Recycle Bin**: quản lý xóa mềm, restore, xóa vĩnh viễn. Mọi API đều yêu cầu session token hợp lệ để đảm bảo bảo mật.

**HìnhA diagram of a software application

AI-generated content may be incorrect. 6.2** Bản đồ API Flask

## **6.3. Luồng Upload**

Quy trình upload hoạt động như sau: Client gửi yêu cầu tạo session mới, server trả về session\_id. Sau đó client chia file thành nhiều chunk nhỏ (thường 512KB), gửi từng chunk kèm offset. Server kiểm tra offset, ghi dữ liệu vào file tạm .part và gửi ack. Khi toàn bộ dữ liệu đã nhận xong, server đổi tên file tạm thành file hoàn chỉnh và cập nhật thông tin trong DB. Ưu điểm là nếu kết nối bị gián đoạn, người dùng chỉ cần resume từ offset cuối cùng thay vì phải upload lại từ đầu.



**Hình 6.3** Cơ chế Chunk & Resume

## **6.4.** **Luồng Download**

Người dùng có thể chọn file để tải về từ trang quản lý. Flask API sẽ cung cấp endpoint trả về file trực tiếp. Trong phiên bản hiện tại, download chưa hỗ trợ resume, tuy nhiên vẫn đảm bảo tốc độ ổn định và an toàn. Đây là điểm nhóm dự kiến mở rộng trong tương lai.

## **6.5.** **Logging & Theo dõi**

Hệ thống sử dụng module logging riêng, log chi tiết tất cả sự kiện: client kết nối mới, trạng thái upload, pause/resume/stop, lỗi chunk, thao tác xóa hoặc restore file. Log có nhiều mức: INFO, DEBUG, ERROR. Thông tin log hỗ trợ quá trình kiểm thử và gỡ lỗi.

**HìnhA diagram of a logistic timeline

AI-generated content may be incorrect. 6.4** Logging Timeline

## **6.6.** **Edge Cases**

Trong thực tế, người dùng có thể gặp nhiều tình huống bất thường. Hệ thống đã xử lý:

* **File rỗng**: từ chối upload, báo lỗi.
* **Trùng tên**: tự động thêm hậu tố an toàn.
* **File vượt dung lượng cho phép**: từ chối ngay từ đầu.
* **Tên file chứa ký tự đặc biệt**: normalize để không gây lỗi hệ thống.
* **Ứng dụng bị crash giữa chừng**: khi mở lại sẽ kiểm tra file .part còn nguyên vẹn và resume tiếp tục.

A diagram of a flowchart

AI-generated content may be incorrect.

Hình 6.5 Edge Case Handling Flow

# **7. Thực hành / Thực nghiệm**

Để kiểm chứng tính đúng đắn và hiệu quả của hệ thống, nhóm đã triển khai nhiều hoạt động thực hành và thực nghiệm trên môi trường phát triển thực tế.

## **7.1. Thiết kế cơ sở dữ liệu**

Ngay từ giai đoạn đầu, nhóm tập trung thiết kế cơ sở dữ liệu với mục tiêu đảm bảo quản lý người dùng, phiên đăng nhập và thông tin file một cách khoa học. Hệ thống xây dựng trên SQLite với các bảng chính: **Users, Sessions, Files, Folders, Recycle Bin**. Mối quan hệ được định nghĩa qua khóa chính – khóa ngoại, giúp duy trì tính toàn vẹn dữ liệu và tách biệt dữ liệu từng user.

* **Users**: lưu username, mật khẩu đã mã hóa (PBKDF2+salt), role.
* **Sessions**: lưu token đăng nhập và thời hạn, hỗ trợ duy trì phiên làm việc an toàn.
* **Files**: metadata file gồm tên, dung lượng, ngày tải lên, trạng thái, liên kết với user và thư mục.
* **Folders**: cấu trúc thư mục lồng nhau, hỗ trợ người dùng sắp xếp khoa học.
* **Recycle Bin**: quản lý xóa mềm, cho phép restore trong thời hạn.

## **7.2. Xây dựng server**

Server được chia thành hai tiến trình chính:

* **Flask Server**: quản lý REST API cho các thao tác đăng ký, đăng nhập, quản lý metadata file, phân quyền người dùng. Ưu điểm: gọn nhẹ, dễ tích hợp, phù hợp hệ thống vừa và nhỏ.
* **WebSocket Server**: phát triển dựa trên AsyncIO, quản lý tiến trình upload theo thời gian thực. Server cho phép truyền tải hai chiều, gửi thông báo ngay về trạng thái upload tới client.

Sự kết hợp Flask + WebSocket giúp vừa xử lý yêu cầu HTTP, vừa đảm bảo realtime cho tiến độ upload.

Ứng dụng được chạy thử trên Python 3.10+, với backend gồm hai thành phần: Flask File Manager (REST API) chạy trên cổng 5000 và WebSocket Server chạy trên cổng 8765. Frontend được phục vụ qua HTTP server (cổng 8000). Cấu hình đơn giản này đủ để mô phỏng một hệ thống truyền tải file hoàn chỉnh.

## **7.3. Xây dựng giao diện frontend**

Frontend được phát triển bằng **HTML5, CSS3, JavaScript ES6+**. Các tính năng chính:

* **Drag & Drop**: kéo thả file trực tiếp vào giao diện để upload.
* **Progress Bar**: thanh tiến độ hiển thị realtime.
* **Responsive UI**: tương thích trên cả PC và thiết bị di động.

Thiết kế giao diện hiện đại, trực quan, dễ tiếp cận nhiều đối tượng người dùng.

### 7.3.1 Giao diện chức năng sản phẩm

A screenshot of a computer

AI-generated content may be incorrect.

### 7.3.2 Giao diện quản lý file và folder

A screenshot of a computer

AI-generated content may be incorrect.

## **7.4. Tích hợp chức năng quản lý file**

Sau khi hoàn thiện upload, nhóm bổ sung thêm các chức năng quản lý file:

* Tạo thư mục mới để tổ chức dữ liệu.
* Tìm kiếm file theo tên hoặc loại để tiết kiệm thời gian.
* Thống kê số lượng file, dung lượng để quản lý lưu trữ hiệu quả.  
  Những chức năng này giúp hệ thống trở thành một **trình quản lý file trực tuyến** thực sự thay vì chỉ là công cụ upload.

## **7.5. Các kịch bản thử nghiệm chính**

Nhóm đã xây dựng nhiều kịch bản để kiểm tra đầy đủ chức năng:

* **Upload nhiều file song song**: chọn đồng thời 5–10 file có dung lượng từ nhỏ (ảnh vài trăm KB) đến lớn (video 100MB). Kết quả: tất cả file được upload thành công, không ảnh hưởng lẫn nhau.
* **Pause/Resume:** trong lúc upload, nhấn pause để dừng và resume để tiếp tục. Kết quả: file tiếp tục đúng từ offset cuối, không bị mất dữ liệu.
* **Stop một file:** khi đang upload nhiều file, dừng một file riêng. Kết quả: file đó dừng, các file khác vẫn tiếp tục.
* **Mất kết nối mạng:** ngắt kết nối trong lúc upload, sau đó kết nối lại và resume. Kết quả: resume thành công, dữ liệu chính xác.
* **Xóa/Restore:** xóa một file và kiểm tra trong Recycle Bin, sau đó restore. Kết quả: file khôi phục nguyên vẹn.
* **Download nhiều file:** chọn nhiều file tải về cùng lúc. Kết quả: tất cả file tải thành công, tuy hiện tại chưa có resume.
* **Preview dữ liệu:** kiểm tra preview ảnh, PDF, video, text. Kết quả: hiển thị đúng trong giao diện.
* **User isolation:** đăng nhập bằng hai tài khoản khác nhau. Kết quả: mỗi user chỉ thấy file của chính mình.

## **7.6. Đánh giá hiệu năng**

Trong thử nghiệm LAN, hệ thống xử lý tốt 10–15 file song song, tiến trình realtime rõ ràng. Thời gian phản hồi trung bình dưới 200ms cho mỗi chunk. CPU và RAM ổn định, chứng minh AsyncIO hiệu quả. Đây là cơ sở để nhóm tin tưởng rằng hệ thống có thể mở rộng thêm mà vẫn giữ hiệu suất.

## **7.7. Nhật ký thử nghiệm**

Hệ thống ghi log chi tiết từng sự kiện: bắt đầu upload, pause, resume, stop, lỗi chunk, delete, restore. Các log hỗ trợ kiểm chứng hoạt động và debug nhanh

A screenshot of a computer

AI-generated content may be incorrect.

## **7.8. Sơ đồ kiến trúc triển khai**

Phần này mô tả **cách triển khai thực tế** các thành phần chạy trên máy phát triển, thể hiện rõ **tiến trình**, **cổng**, **đường dẫn lưu trữ** và **luồng dữ liệu**. Cấu hình đề xuất:

* **Frontend Static Server**: http://localhost:8000 (chạy bằng python -m http.server 8000) – phục vụ HTML/CSS/JS tĩnh.
* **Flask File Manager (REST)**: http://localhost:5000 – cung cấp API /login, /register, /files, /download, /preview, /stats (yêu cầu token hợp lệ).
* **WebSocket Upload Server**: ws://localhost:8765/ws – nhận chunk + offset, gửi ACK, hỗ trợ Pause/Resume/Stop cho từng file.
* **Storage**: remote\_uploads/{user}/ (file hoàn chỉnh) và temp\_uploads/\*.part (file tạm trong lúc upload).
* **SQLite Database**: các bảng users, sessions, files, folders, recycle\_bin.

**Luồng chính**:

1. Trình duyệt tải static assets từ Frontend server (cổng 8000).
2. Người dùng đăng nhập và thao tác file qua REST API (cổng 5000).
3. Upload sử dụng WebSocket (cổng 8765) gửi chunk + offset; server ghi vào temp\_uploads/\*.part và khi xong sẽ đổi tên thành file hoàn chỉnh trong remote\_uploads/{user}/.
4. Flask quản lý metadata và phục vụ download từ remote\_uploads/{user}/; đồng thời truy vấn SQLite để kiểm tra quyền và cập nhật trạng thái.

Tóm lại, phần thực hành/thực nghiệm chứng minh hệ thống vận hành đúng thiết kế, ổn định trong nhiều tình huống, đáp ứng yêu cầu đề tài.

# **8. Kinh nghiệm rút ra**

Trong quá trình triển khai đề tài, nhóm đã thu được rất nhiều kinh nghiệm quý báu cả về mặt kỹ thuật lẫn kỹ năng làm việc nhóm. Đây không chỉ là một bài tập cuối kỳ mà còn là một dự án thực tế, giúp các thành viên rèn luyện cách vận dụng kiến thức vào giải quyết vấn đề.

## **8.1. Về kỹ thuật**

* **Quản lý trạng thái bất đồng bộ**: Qua AsyncIO, nhóm hiểu sâu hơn cách xử lý hàng loạt kết nối mà không cần nhiều luồng, tiết kiệm tài nguyên nhưng vẫn đảm bảo hiệu năng. Việc áp dụng vào WebSocket giúp thấy rõ sự khác biệt so với cách xử lý tuần tự truyền thống.
* **Truyền tải file theo chunk**: Đây là một cơ chế phức tạp, đòi hỏi phải kiểm soát offset, xử lý ACK và đảm bảo dữ liệu không trùng hoặc thiếu. Quá trình triển khai giúp nhóm nhận thức rõ tầm quan trọng của việc kiểm chứng từng bước nhỏ để đảm bảo toàn vẹn dữ liệu.
* **Logging và giám sát hệ thống**: Nhóm học được rằng việc ghi log chi tiết (INFO, DEBUG, ERROR) không chỉ giúp kiểm thử mà còn cực kỳ hữu ích khi demo hoặc phát hiện lỗi bất thường.
* **Bảo mật ứng dụng**: Việc triển khai password hashing, token-based authentication và kiểm tra dữ liệu đầu vào cho nhóm thêm kinh nghiệm về bảo mật, hiểu rõ rằng nếu bỏ qua thì hệ thống có thể bị khai thác dễ dàng.
* **Tích hợp giao diện với backend**: Quá trình kết nối frontend với API Flask và WebSocket cho nhóm cái nhìn thực tế về những khó khăn khi đồng bộ hai bên, từ CORS đến xử lý JSON và cập nhật realtime.

## **8.2. Về kỹ năng làm việc nhóm**

* **Phân chia công việc và phối hợp**: Mỗi thành viên chịu trách nhiệm một phần (frontend, backend, database, tài liệu), nhưng phải liên tục trao đổi để đảm bảo hệ thống chạy thống nhất.
* **Sử dụng Git/GitHub**: Ban đầu nhóm gặp khó khăn khi merge code và xử lý conflict, nhưng dần dần thành thạo cách tạo branch, pull request và review, nhờ đó quy trình làm việc trở nên chuyên nghiệp hơn.
* **Thuyết trình và viết báo cáo**: Quá trình chuẩn bị slide, viết báo cáo, và tập thuyết trình giúp nhóm rèn luyện khả năng trình bày mạch lạc và thuyết phục.

## **8.3. Bài học quan trọng**

* Hệ thống phần mềm chỉ thực sự hoàn thiện khi được **kiểm thử với nhiều tình huống**: mất mạng, file trùng tên, file rỗng… Đây là kinh nghiệm thực tế mà sách vở khó thể hiện.
* **Thiết kế ban đầu quan trọng nhưng không tránh khỏi việc điều chỉnh**. Nhóm đã thay đổi một số ý tưởng sau khi kiểm thử để phù hợp hơn với thực tế (ví dụ: bổ sung Recycle Bin, tối ưu resume).
* **Sự kiên nhẫn và kỹ năng gỡ lỗi**: Việc xử lý lỗi offset sai, chunk trùng hay token hết hạn cho thấy cần kiên nhẫn và có phương pháp tiếp cận bài bản.

Tóm lại, qua dự án này, mỗi thành viên không chỉ nâng cao kiến thức lập trình mạng mà còn trưởng thành hơn trong kỹ năng làm việc nhóm, quản lý dự án và tư duy giải quyết vấn đề.

# **9. Kết quả đạt được**

Trong suốt quá trình thực hiện, nhóm đã đạt được nhiều kết quả quan trọng cả về mặt kỹ thuật lẫn giá trị học tập. Những kết quả này chứng minh sản phẩm không chỉ đáp ứng yêu cầu đề tài mà còn có thể mở rộng để ứng dụng thực tế.

## **9.1. Hoàn thiện chức năng chính**

* Hệ thống hỗ trợ **Upload/Download nhiều file đồng thời**, mỗi file được quản lý riêng biệt.
* Các nút **Pause, Resume, Stop** hoạt động ổn định trên từng file upload.
* Quản lý file với các tính năng **rename, move, delete, restore**, cùng **Recycle Bin** để tránh mất dữ liệu ngoài ý muốn.
* **Preview file**: ảnh, PDF, văn bản, video/audio có thể xem nhanh trên trình duyệt.

## **9.2. Đảm bảo tính ổn định**

* Hệ thống chạy mượt trong môi trường LAN với 10–15 file song song.
* Thời gian phản hồi thấp, tiến trình hiển thị realtime rõ ràng.
* AsyncIO chứng minh hiệu quả trong việc xử lý nhiều kết nối cùng lúc.

## **9.3. Toàn vẹn và an toàn dữ liệu**

* Cơ chế chia nhỏ file theo chunk + offset và xác nhận ACK đảm bảo **không mất hoặc trùng dữ liệu**.
* Hash kiểm tra sau upload cho thấy file đích trùng khớp với file nguồn.
* Bảo mật: mật khẩu được hash (PBKDF2+salt), token auth hoạt động đúng, người dùng chỉ truy cập file của mình.

## **9.4. Kinh nghiệm học được**

* Nhóm nắm vững cách sử dụng WebSocket, AsyncIO và Flask trong dự án thực tế.
* Thành thạo hơn trong thiết kế database, xây dựng REST API, và đồng bộ frontend–backend.
* Rèn kỹ năng teamwork: phân chia nhiệm vụ, sử dụng GitHub, viết báo cáo và thuyết trình.

# **10. Kết quả chưa đạt**

Bên cạnh những thành công đạt được, hệ thống vẫn còn tồn tại một số hạn chế nhất định cần được nhìn nhận để cải tiến trong tương lai.

## **10.1. Về chức năng**

* **Resume cho Download**: hiện tại chỉ mới triển khai resume cho upload. Khi tải xuống file, nếu bị ngắt kết nối thì người dùng phải tải lại từ đầu, gây lãng phí thời gian và băng thông.
* **Giao diện người dùng (UI)**: mới dừng ở mức cơ bản, chủ yếu để minh họa chức năng. Chưa có nhiều tính năng nâng cao về UI/UX, cũng chưa hỗ trợ đa ngôn ngữ.
* **Chức năng quản lý nâng cao**: hệ thống chưa hỗ trợ quota theo user, chưa có phân quyền chi tiết (chỉ mới dừng ở mức user và admin cơ bản).

## **10.2. Về hiệu năng**

* **File cực lớn**: hệ thống hoạt động ổn định với file đến hàng trăm MB, nhưng với file vài GB thì cần tối ưu thêm để tránh chiếm dụng quá nhiều RAM hoặc gây nghẽn I/O.
* **Tối ưu mạng**: chưa có cơ chế nén dữ liệu hay điều chỉnh tự động kích thước chunk theo tốc độ mạng, do đó hiệu suất truyền tải có thể chưa tối ưu trong mọi điều kiện.

## **10.3. Về bảo mật**

* **SSL/TLS**: trong bản demo chưa bật HTTPS/WSS. Điều này hạn chế tính an toàn khi truyền tải dữ liệu qua mạng công cộng.
* **Quản lý phiên**: token xác thực hiện có hạn 24h, nhưng chưa có cơ chế refresh tự động hoặc đăng xuất từ xa.

## **10.4. Về khả năng mở rộng**

* **Tích hợp cloud**: hệ thống chưa kết nối với các dịch vụ lưu trữ đám mây như AWS S3 hoặc Google Drive.
* **Triển khai thực tế**: hệ thống mới chạy thử nghiệm trong môi trường cục bộ (localhost), chưa triển khai trên môi trường sản xuất với nhiều user đồng thời.

Tóm lại, các hạn chế này không ảnh hưởng lớn đến việc chứng minh ý tưởng và mục tiêu của đề tài, nhưng cần được khắc phục nếu muốn hệ thống đạt mức triển khai thực tế.

# **11. Hướng phát triển**

Dựa trên các kết quả đã đạt được và những hạn chế còn tồn tại, nhóm đề xuất một số hướng phát triển trong tương lai nhằm hoàn thiện hệ thống hơn, nâng cao trải nghiệm người dùng và tăng khả năng mở rộng.

## **11.1. Hoàn thiện chức năng**

* **Bổ sung Resume cho Download**: triển khai cơ chế HTTP Range hoặc checkpoint download để khi mạng gián đoạn, người dùng có thể tiếp tục từ vị trí đã tải.
* **Batch Download**: cho phép tải nhiều file cùng lúc dưới dạng ZIP, tiện lợi khi cần lấy toàn bộ thư mục.
* **Quản lý nâng cao**: thêm quota cho từng user, phân quyền chi tiết hơn giữa admin, user thường, và các role khác.

## **11.2. Nâng cao bảo mật**

* **Tích hợp SSL/TLS**: triển khai HTTPS/WSS để bảo mật kênh truyền khi đưa hệ thống lên môi trường thực tế.
* **Cải thiện quản lý phiên**: thêm cơ chế refresh token, đăng xuất từ xa, cảnh báo khi phát hiện đăng nhập bất thường.

## **11.3. Cải thiện giao diện**

* **Responsive UI hoàn thiện**: hỗ trợ đầy đủ cho di động, máy tính bảng.
* **Drag & Drop nâng cao**: cho phép kéo thả cả thư mục.
* **Preview phong phú**: hỗ trợ thêm nhiều định dạng tài liệu (docx, xlsx, pptx).

## **11.4. Tối ưu hiệu năng**

* **Xử lý file cực lớn**: áp dụng cơ chế streaming, mmap hoặc chia chunk động theo tốc độ mạng.
* **Tối ưu mạng**: bổ sung nén dữ liệu, điều chỉnh tự động kích thước chunk.

## **11.5. Khả năng mở rộng**

* **Tích hợp cloud storage**: kết nối với AWS S3, Google Drive để lưu trữ phân tán.
* **Triển khai trên môi trường thực tế**: cài đặt hệ thống trên server thật hoặc cloud, kiểm tra với nhiều user đồng thời.
* **Microservices hóa**: tách biệt hẳn WebSocket, Flask, và Database thành các dịch vụ độc lập, dễ scale.

Tóm lại, những hướng phát triển này sẽ giúp hệ thống từ một bản demo học thuật trở thành một ứng dụng hoàn chỉnh, có thể triển khai cho nhiều mục đích thực tế.

# **Tài liệu tham khảo**

1. Python Software Foundation. *asyncio — Asynchronous I/O*. Python 3.10 Documentation. Truy cập tại: <https://docs.python.org/3/library/asyncio.html>
2. Python Software Foundation. *hashlib — Secure hashes and message digests*. Python 3.10 Documentation. Truy cập tại: <https://docs.python.org/3/library/hashlib.html>
3. Python Software Foundation. *logging — Logging facility for Python*. Python 3.10 Documentation. Truy cập tại <https://docs.python.org/3/library/logging.html>
4. Pallets Project. *Flask Documentation*. Truy cập tại: <https://flask.palletsprojects.com/>
5. AIO-Libs. *aiohttp Documentation*. Truy cập tại: <https://docs.aiohttp.org/>
6. aaugustin. *websockets — WebSocket library for Python*. Truy cập tại: <https://websockets.readthedocs.io/>
7. SQLite Consortium. *SQLite Documentation*. Truy cập tại: <https://www.sqlite.org/docs.html>
8. Mozilla Developer Network (MDN). *WebSockets API*. Truy cập tại: <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API>
9. Internet Engineering Task Force (IETF). *RFC 6455 – The WebSocket Protocol*. [Truy cập tại: https://datatracker.ietf.org/doc/html/rfc6455](file:///D:\Lập%20trình%20mạng\Truy%20cập%20tại:%20https:\datatracker.ietf.org\doc\html\rfc6455)
10. Giáo trình môn *Lập trình mạng*, Khoa CNTT.
11. Giáo trình môn *Hệ điều hành*, Khoa CNTT.
12. Bài báo, blog kỹ thuật về **Resumable Upload** và **File Transfer Optimization** (tham khảo từ Medium, Stack Overflow, Dev.to).