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

Khoa Khoa học và Kỹ thuật Máy tính

ĐỒ ÁN TỔNG HỢP
CÔNG NGHỆ PHẦN MỀM

HỆ THỐNG FILE SHARING

Nhóm: Backend Web
GVHD: Thầy Lê Đình Thuận

Thành phố Hồ Chí Minh, tháng 11 năm 2025

CHƯƠNG 1

TỔNG QUAN DỰ ÁN

1.1 Mô tả dự án

Hệ thống **File Sharing and Management System** là một nền tảng web hỗ trợ người dùng tải lên, quản lý và chia sẻ tệp tin một cách an toàn, tiện lợi và linh hoạt.

Hệ thống được xây dựng theo kiến trúc dịch vụ REST với backend phát triển bằng **Golang**, giao tiếp thông qua API chuẩn hoá, và frontend dưới dạng ứng dụng web (SPA) do nhóm frontend triển khai.

Hệ thống cho phép người dùng tải tệp dưới hai hình thức: *ẩn danh* hoặc *đăng nhập*. Mỗi tệp được gán một *share token* duy nhất giúp người nhận có thể truy cập nhanh, đồng thời hỗ trợ các cơ chế bảo vệ như mật khẩu, whitelist theo email và giới hạn thời gian hiệu lực.

Thông tin tệp (metadata), lịch sử truy cập và các chính sách hệ thống được quản lý tập trung trong cơ sở dữ liệu. Bên cạnh đó, hệ thống hỗ trợ cơ chế dọn dẹp tệp tự động (cleanup) thông qua tác vụ cron với mã xác thực `X-Cron-Secret`. Tính năng thống kê lượt tải và xem lịch sử download hỗ trợ quá trình theo dõi và kiểm toán.

Mục tiêu tổng thể của dự án là xây dựng một nền tảng lưu trữ tệp hiệu quả, mở rộng tốt, đảm bảo bảo mật và dễ tích hợp với các dịch vụ phụ trợ như email service hoặc cloud storage trong tương lai.

1.2 Yêu cầu chức năng và phi chức năng

1.2.1 Yêu cầu chức năng

FR1 – Quản lý người dùng

- FR1.1: Hệ thống cho phép người dùng đăng ký và đăng nhập thông qua email và mật khẩu; xác thực bằng JWT.
- FR1.2: Người dùng có thể kích hoạt và xác minh TOTP (Time-based One-Time Password) để tăng cường bảo mật cho tài khoản (2FA). Sau khi bật TOTP, các lần đăng nhập tiếp theo sẽ yêu cầu mã 6 chữ số từ ứng dụng authenticator.
- FR1.3: Người dùng ẩn danh được phép tải lên tệp nhưng chỉ được upload file public (`isPublic=true`); không thể chỉnh sửa, quản lý hoặc xóa tệp sau đó.

- FR1.4: Người dùng đăng nhập có thể truy cập danh sách tệp cá nhân qua endpoint `/files/my`, xem chi tiết metadata qua `/files/stats/{id}`, thay đổi bảo mật, hoặc xóa tệp.
- FR1.5: Quản trị viên có thể truy cập các API quản lý hệ thống, bao gồm cập nhật chính sách (`/admin/policy`) và thực thi tác vụ cleanup (`/admin/cleanup`).

FR2 – Upload và chia sẻ tệp

- FR2.1: Hệ thống hỗ trợ tải tệp bằng phương thức multipart thông qua endpoint `/files/upload`.
- FR2.2: Sau khi tải thành công, hệ thống sinh ra *share token* duy nhất và liên kết chia sẻ cho phép tải xuống qua endpoint `/files/{shareToken}/download`.
- FR2.3: Người dùng có thể thiết lập các thuộc tính:
  - Khoảng thời gian hiệu lực `availableFrom` – `availableTo` (validate theo system policy: FROM $\leq$ TO, TO không nằm trong quá khứ, tổng thời gian không vượt `maxValidityDays`).
  - Danh sách email được phép tải (*whitelist* `sharedWith`) - yêu cầu authenticated upload.
  - Mật khẩu bảo vệ (tối thiểu 8 ký tự) - yêu cầu authenticated upload.
  - Trạng thái công khai hoặc riêng tư (`isPublic`) - anonymous upload luôn bị ép = true.
- FR2.4: Các tệp hết hạn sẽ không còn khả dụng và trả về mã lỗi HTTP 410. Tệp chưa đến thời gian hiệu lực (pending) trả về mã 423, trừ khi requester là owner (có thể preview để kiểm thử).

FR3 – Bảo mật và kiểm soát truy cập

- FR3.1: Mật khẩu tệp được băm bằng **bcrypt**; mật khẩu user cũng được hash bằng bcrypt. Secret TOTP của user được lưu trữ an toàn trong cơ sở dữ liệu.
- FR3.2: Quy trình kiểm tra truy cập download bao gồm theo thứ tự: (1) trạng thái tệp (expired $\rightarrow$ 410, pending $\rightarrow$ 423), (2) kiểm tra whitelist (nếu có `sharedWith` $\rightarrow$ yêu cầu JWT và email nằm trong danh sách), (3) yêu cầu mật khẩu (nếu file có password $\rightarrow$ phải gửi query parameter `password`).
- FR3.3: Người dùng ẩn danh không thể chỉnh sửa thông tin tệp đã tải lên; chỉ người dùng đăng nhập (owner) mới có quyền quản lý file của mình.
- FR3.4: Endpoint `/admin/cleanup` yêu cầu token admin hoặc header `X-Cron-Secret` (với cơ chế rotation định kỳ, rate limiting và logging đầy đủ).

FR4 – Quản lý vòng đời tệp

- FR4.1: Metadata tệp được lưu trong cơ sở dữ liệu bao gồm thuộc tính (fileName, fileSize, mimeType), thời hạn (availableFrom/To, status: pending/active/expired), thông tin bảo mật (password hash, whitelist) và trạng thái.

- FR4.2: Người dùng có thể xem thống kê tệp qua `/files/stats/{id}`, bao gồm tổng lượt tải (`downloadCount`), số người download khác nhau (`uniqueDownloaders`) và thời điểm tải gần nhất (`lastDownloadedAt`). Anonymous uploads không có statistics.
- FR4.3: Hệ thống lưu lịch sử tải xuống chi tiết qua `/files/download-history/{id}` để phục vụ mục đích kiểm toán. Với anonymous download, chỉ ghi nhận "Anonymous" + timestamp + trạng thái, không lưu IP/User-Agent để đảm bảo privacy.
- FR4.4: Tệp có thể bị xóa bởi chủ sở hữu qua `DELETE /files/info/{id}` hoặc bị dọn dẹp tự động khi hết hạn thông qua `/admin/cleanup`.

1.2.2 Yêu cầu phi chức năng

Hiệu năng

- NFR1: Thời gian phản hồi đăng nhập không vượt quá 5 giây.
- NFR2: Mã TOTP phải được sinh và trả về (qua QR code) trong vòng 30 giây (tối đa 60 giây khi tải cao).
- NFR3: Thời gian tải lên tệp dưới 100 MB không vượt quá 30 giây.
- NFR4: Việc tải xuống hỗ trợ streaming HTTP đảm bảo ổn định với tệp dung lượng lớn, không chiếm nhiều RAM trên server.

Bảo mật

- NFR5: Toàn bộ giao tiếp giữa FE và BE sử dụng HTTPS (TLS 1.2 trở lên).
- NFR6: JWT có thời hạn và hỗ trợ refresh token (hoặc yêu cầu đăng nhập lại khi token hết hạn).
- NFR7: Share token được sinh ngẫu nhiên, chống dò quét, không thể đoán trước.
- NFR8: Mật khẩu user và file được hash bằng bcrypt; secret TOTP của user được lưu trữ an toàn; thông tin nhạy cảm không được log hoặc expose trong response.

Khả năng mở rộng và bảo trì

- NFR9: Backend được thiết kế theo mô-đun, dễ mở rộng theo chức năng (ví dụ: tích hợp cloud storage, email service).
- NFR10: CSDL hỗ trợ đánh index và phân trang cho dữ liệu lớn (pagination với `page` và `limit`).
- NFR11: Sẵn sàng tích hợp cloud storage (S3, GCS) và email service (SMTP, provider bên thứ ba) trong tương lai.

Chịu lỗi

- NFR12: Metadata và dữ liệu tệp (binary) được tách biệt, tránh mất dữ liệu metadata khi lỗi lưu trữ file binary.
- NFR13: Các thông báo lỗi tuân theo chuẩn JSON trong tài liệu OpenAPI (schema `Error` với `error`, `message`, `code`).

Giao diện và trải nghiệm người dùng

- NFR14: Dashboard hiển thị danh sách tệp qua `/files/my` với pagination, filter theo status, và summary statistics (active/pending/expired/deleted files).
- NFR15: Hệ thống hỗ trợ owner preview file trong giai đoạn pending (bypass 423) để chủ file có thể kiểm thử link trước khi phát hành.
- NFR16: Khuyến nghị cấu hình background job gửi thông báo (email/SMS/webhook) khi file chuyển từ pending sang active cho owner và whitelist (tính năng này không được triển khai trong phạm vi hiện tại nhưng được thiết kế sẵn cho tương lai).
- NFR17: Hệ thống tự động xoá các tệp upload ẩn danh hoặc tệp tạm sau thời gian tồn tại vượt quá x giờ nhằm tránh chiếm dụng dung lượng và đảm bảo vệ sinh dữ liệu.

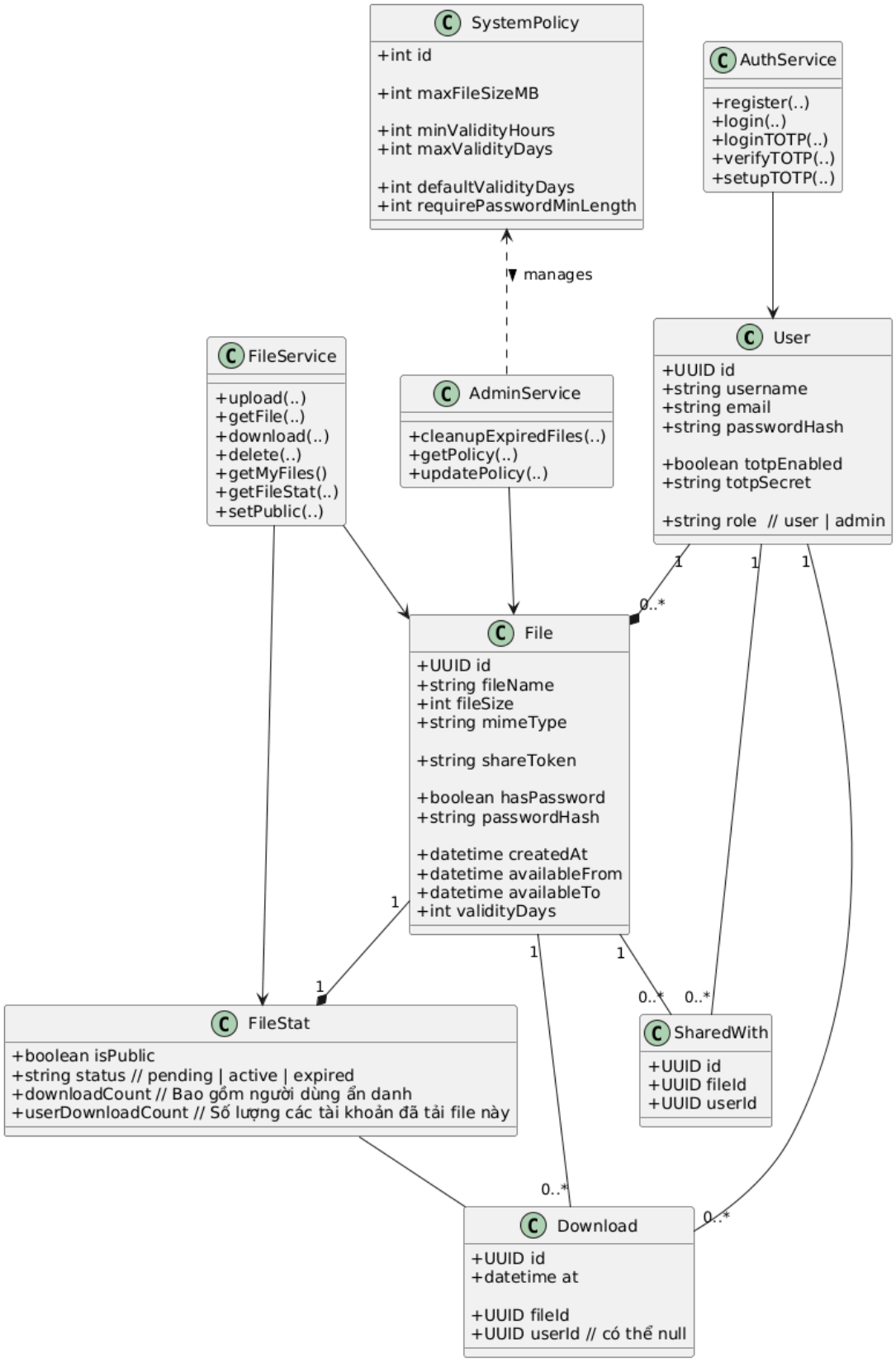
1.3 Use-case diagram cho toàn bộ hệ thống

Hình 1.1: Sơ đồ Use Case của hệ thống chia sẻ file

No.	Use-case	Job Description
1	Create Account	Dùng để người dùng tạo tài khoản mới trong hệ thống.
2	Login	Cho phép người dùng đăng nhập để truy cập các chức năng của hệ thống.
3	Logout	Đăng xuất khỏi hệ thống, kết thúc phiên làm việc.
4	Enable TOTP	Bật tính năng xác thực hai lớp (2FA) cho tài khoản người dùng.
5	Setup TOTP	Sinh mã QR và secret key để người dùng đăng ký Google Authenticator.
6	Verify TOTP	Kiểm tra mã xác thực sáu chữ số được tạo từ ứng dụng xác thực.
7	Upload files	Cho phép người dùng hoặc khách ẩn danh tải file lên hệ thống và tạo link chia sẻ.
8	Set password	Thiết lập mật khẩu bảo vệ file để chỉ người có mật khẩu mới tải được.
9	Share with	Chia sẻ file cho danh sách người dùng cụ thể qua email.
10	Download files	Tải file về nếu thỏa mãn điều kiện truy cập.
11	Download History	Ghi lại lịch sử tải file.
12	Delete files	Cho phép chủ sở hữu file xóa file khỏi hệ thống.
13	Update files	Cập nhật thông tin hoặc thuộc tính của file.
14	List my files	Hiển thị danh sách file của người dùng.
15	Validate Period	Kiểm tra thời gian hiệu lực file.
16	Set attribute file input	Thiết lập thuộc tính file.
17	System Policy	Xem và quản lý cấu hình hệ thống.
18	Cleanup Expired Files	Xóa các file đã hết hạn.

Bảng 1.1: Bảng mô tả Use Case của hệ thống chia sẻ file

1.4 Class Diagram



Hình 1.2: Sơ đồ Class của hệ thống chia sẻ file

No.	Class	Giải thích
1	User	Lưu trữ thông tin về tài khoản người dùng.
2	File	Lưu các metadata về các file được tải lên.
3	FileStat	Chứa các thông tin về tình trạng và thống về của file.
4	SharedWidth	Cho biết file này được chia sẻ với các người dùng nào.
5	Download	Ghi nhận những lượt tải về (bao gồm tải về ẩn danh).
6	FileService, AdminService và AuthService	Các interface để quản lý, tương tác với File, User, FileStat.
7	SystemPolicy	Các ràng buộc về tải lên và chia sẻ file.

Bảng 1.2: Bảng mô tả về Class diagram

CHƯƠNG 2

API SPEC

2.1 Giới thiệu chung

- **OpenAPI:** 3.0.4
- **Tên:** File Sharing System API – hệ thống chia sẻ file tạm thời cho phép upload, đặt thời gian hiệu lực, chia sẻ qua link, giới hạn người nhận, bảo vệ bằng mật khẩu hoặc TOTP, tự xoá khi hết hạn.
- **Phiên bản:** 1.0.0
- **Servers:**
  - `http://localhost:8080/api` (môi trường phát triển)
  - `https://api.filesharing.com/api` (môi trường production)
- **Tags chính:**
  - **Authentication**: Đăng ký, đăng nhập, TOTP/2FA, lấy thông tin user.
  - **Files**: Upload, xem, xoá, thống kê và lịch sử download, download qua share token.
  - **Admin**: Cleanup file hết hạn, cấu hình system policy.
- **Security global:** Mặc định tất cả endpoint (trừ các endpoint public như đăng ký, login, xem metadata/shareToken, download public) đều yêu cầu Bearer JWT. Một số endpoint admin cho phép xác thực bằng `X-Cron-Secret`.

2.2 Authentication

2.2.1 POST /auth/register

- **Mục đích**: Tạo tài khoản user mới, phục vụ upload/manage file có gắn owner.
- **Auth**: Không yêu cầu.
- **Request body** (`RegisterRequest`):

  - `username` (string, bắt buộc) – duy nhất.
  - `email` (string, email, bắt buộc) – duy nhất.
  - `password` (string, ≥ 8 ký tự, bắt buộc).
- **Response 200** (`RegisterResponse`):
  - `message`: *User registered successfully.*
  - `userId`: UUID của user vừa tạo.
- **Mã lỗi**:
  - 400 – thiếu trường bắt buộc hoặc format không hợp lệ (ví dụ email sai định dạng).
  - 409 – email hoặc username đã tồn tại.

2.2.2 POST /auth/login

- **Mục đích**: Đăng nhập bằng email/password để lấy JWT, với luồng hỗ trợ TOTP.
- **Auth**: Không yêu cầu.
- **Request body** (`LoginRequest`):
  - `email` (string, email).
  - `password` (string).
- **Response 200**:
  - Nếu user *chưa bật TOTP*:
    - `LoginResponse`: `accessToken` + `user` (id, username, email, role, totpEnabled).
  - Nếu user *đã bật TOTP*:
    - `TOTPRequiredResponse`: `requireTOTP = true`, `message` = *TOTP verification required.*
- **Mã lỗi**:
  - 401 – email hoặc password không đúng.

2.2.3 POST /auth/login/totp

- **Mục đích**: Hoàn tất đăng nhập cho user đã bật TOTP (sau khi /auth/login trả requireTOTP).
- **Auth**: Không yêu cầu Bearer token (vì user chưa có token).
- **Request body** (`TOTPVerifyRequest`):
  - `email` (string, email).
  - `code` (string, pattern 6 chữ số).
- **Response 200**: `LoginResponse` – access token + thông tin user.
- **Mã lỗi**:
  - 401 – mã TOTP sai hoặc đã hết hạn / phiên login hết hạn.

2.2.4 POST /auth/totp/setup

- **Mục đích**: Khởi tạo secret TOTP và QR code cho user, để chuẩn bị bật 2FA.
- **Auth**: Bắt buộc Bearer token.
- **Request**: Không có body.
- **Response 200** (`TOTPSetupResponse`):
  - `message`: *TOTP secret generated.*
  - `totpSetup.secret`: secret dạng Base32.
  - `totpSetup.qrCode`: chuỗi hình ảnh QR dạng data URL (base64).
- **Mã lỗi**:
  - 401 – thiếu hoặc token không hợp lệ/hết hạn.

2.2.5 POST /auth/totp/verify

- **Mục đích**: Verify mã TOTP được sinh từ secret, chính thức bật 2FA cho tài khoản.
- **Auth**: Bearer token bắt buộc.
- **Request body**:
  - `code` (string, 6 chữ số) – mã từ app authenticator.
- **Response 200**:
  - `message`: *TOTP verified successfully.*
  - `totpEnabled`: `true`.
- **Mã lỗi**:
  - 400 – mã không đúng hoặc hết hạn.
  - 401 – thiếu/sai Bearer token.

2.2.6 POST /auth/logout

- **Mục đích**: Đăng xuất logic phía server (client tự xoá JWT).
- **Response 200**: `message = User logged out`.

2.2.7 GET /user

- **Mục đích**: Lấy thông tin profile user hiện tại để hiển thị trang tài khoản.
- **Auth**: Bearer token bắt buộc.
- **Response 200** (`UserProfileResponse`):
  - `user`:
    - `id` (uuid), `username`, `email`, `role` (user/admin), `totpEnabled` (bool).
- **Mã lỗi**: 401 – chưa đăng nhập hoặc token hết hạn.

2.3 Files

2.3.1 POST /files/upload

- **Mục đích**: Upload file mới và sinh share link.
- **Auth**:
  - Optional – không gửi token thì anonymous upload.
  - Tuy nhiên anonymous chỉ được upload file public (`isPublic=true`).
- **Request body** (multipart, `FileUploadRequest`):
  - `file` (binary, bắt buộc).
  - `isPublic` (boolean, mặc định true) – anonymous luôn bị ép = true.
  - `password` (string, min 8) – password bảo vệ file (chỉ khi đã auth).
  - `availableFrom` (date-time) – thời điểm bắt đầu hiệu lực.
  - `availableTo` (date-time) – thời điểm kết thúc hiệu lực.
  - `sharedWith` (array email) – whitelist email được phép tải (chỉ khi đã auth).
- **Validation thời gian**:
  - `availableFrom` $\leq$ `availableTo`.
  - `availableTo` không nằm trong quá khứ.
  - Khoảng FROM $\rightarrow$ TO không vượt quá `system_policy.maxValidityDays`.
  - Nếu thiếu FROM/TO:
    - FROM + TO: dùng trực tiếp.
    - Chỉ TO: từ hiện tại đến TO.
    - Chỉ FROM: FROM đến FROM + defaultValidityDays (mặc định 7).
    - Không có: hiện tại đến +7 ngày.
- **Response 201** (`FileUploadResponse`):
  - `success` (bool).
  - `message`: *File uploaded successfully.*
  - `file`: đối tượng `File` với id, fileName, fileSize, mimeType, shareToken, shareLink, isPublic, hasPassword, status, owner, ...
- **Mã lỗi**:
  - 400 – thiếu file, password quá ngắn, thời gian hiệu lực không hợp lệ.
  - 401 – thiếu/sai token khi cố upload private/whitelist/password (case *privateRequiresAuth*).
  - 413 – file vượt giới hạn dung lượng.

2.3.2 GET /files/my

- **Mục đích**: Dashboard "My files" – liệt kê các file do user hiện tại upload.
- **Auth**: Bearer token bắt buộc.
- **Query params**:
  - `status`: `active|expired|pending|deleted|all`, mặc định `all`.
  - `page`: số trang, `default=1`, `min=1`.
  - `limit`: số file/trang, `default=20`, `min=1`, `max=100`.
  - `sortBy`: `createdAt|fileName`, mặc định `createdAt`.
  - `order`: `asc|desc`, mặc định `desc`.
- **Response 200** (`UserFilesResponse`):
  - `files`: danh sách `File` (id, fileName, status, validity, owner, ...).
  - `pagination`: currentPage, totalPages, totalFiles, limit.
  - `summary`: activeFiles, pendingFiles, expiredFiles, deletedFiles.
- **Mã lỗi**: 401 – chưa đăng nhập.

2.3.3 GET /files/info/{id}

- **Mục đích**: Lấy metadata chi tiết bằng file ID (UUID) để hiển thị trong dashboard owner/admin.
- **Auth**: Bearer token; chỉ owner hoặc admin được truy cập.
- **Path param**: `id` (uuid).
- **Response 200** (`FileInfoResponse`):
  - `file`: đối tượng `File` đầy đủ (bao gồm thông tin nhạy cảm như whitelist, link, thời gian hiệu lực, ...).
- **Mã lỗi**: 401 (chưa đăng nhập), 403 (không phải owner/admin), 404 (file không tồn tại).

2.3.4 DELETE /files/info/{id}

- **Mục đích**: Xoá file do owner sở hữu.
- **Auth**: Bearer token; chỉ owner được xoá, anonymous upload không thể xoá sau đó.
- **Response 200**: `message` + `fileId`.
- **Mã lỗi**: 403 (không phải owner hoặc anonymous), 404 (file không tồn tại).

2.3.5 GET /files/stats/{id}

- **Mục đích**: Thống kê lượt download ở mức tổng quan.
- **Auth**: Bearer token; chỉ owner/admin.
- **Response 200**:
  - `fileId`, `fileName`.
  - `statistics.downloadCount` – tổng lượt download.
  - `statistics.uniqueDownloaders` – số user khác nhau (không tính anonymous).
  - `statistics.lastDownloadedAt`, `statistics.createdAt`.
- **Ghi chú**: Anonymous uploads không có statistics → có thể trả 404 kèm thông điệp tương ứng.

2.3.6 GET /files/download-history/{id}

- **Mục đích**: Audit trail chi tiết các lượt download.
- **Auth**: Bearer token; chỉ owner/admin.
- **Response 200**:
  - `fileId`, `fileName`.
  - `history[]`:
    - `id`: UUID của bản ghi download.
    - `downloader`:
      - `username`, `email` (nullable) – `null` nếu anonymous.
    - `downloadedAt`: thời điểm download.
    - `downloadCompleted`: bool (false nếu bị gián đoạn).
  - `pagination`: currentPage, totalPages, totalRecords, limit.
- **Privacy**: anonymous download chỉ lưu "Anonymous" + timestamp + trạng thái; không log IP/User-Agent/fingerprint.

2.3.7 GET /files/{shareToken}

- **Mục đích**: Lấy metadata cơ bản của file qua share token để hiển thị trước khi tải.
- **Auth**: Public, không yêu cầu JWT.
- **Path param**: `shareToken`.
- **Response 200**: `FileInfoResponse` với các trường public (id, fileName, status, isPublic, ...).
- **Mã lỗi**: 404 (token không tồn tại), 410 (file đã hết hạn).

2.3.8 GET /files/{shareToken}/download

- **Mục đích**: Download file thực tế thông qua share token.
- **Auth**:
  - Hỗ trợ cả anonymous và authenticated.
  - Nếu có whitelist (`sharedWith`) thì bắt buộc JWT.
- **Logic kiểm tra (theo thứ tự)**:
  1. **File status**:
     - expired → 410 + `expiredAt`.
     - pending → 423 + `availableFrom`, `hoursUntilAvailable` (trừ owner preview).
  2. **Whitelist**:
     - Nếu có `sharedWith[]`:
       * Thiếu JWT → 401 (*missingAuth*).
       * Có JWT nhưng email không nằm trong danh sách → 403 (*notWhitelisted*).
  3. **Password**:
     - Nếu file có password:
       * Thiếu `password` query → 403 (*missingPassword*).
       * Password sai → 403 (*wrongPassword*).
- **Owner preview**: nếu requester là owner (JWT, `sub` = ownerId) thì được phép bypass trạng thái pending để test link; các user khác vẫn nhận 423 cho đến khi đến giờ.
- **Notification**: spec khuyến nghị cấu hình cron/background job gửi email/SMS/webhook khi file chuyển từ pending sang active (không do endpoint tự gửi).
- **Response 200**: trả binary stream (octet-stream) với header `Content-Disposition` và `Content-Length`.
- **Mã lỗi**: 401, 403, 404, 410, 423 như mô tả ở trên.

2.4 Admin

2.4.1 POST /admin/cleanup

- **Mục đích**: Dọn dẹp các file đã hết hạn (cron job hoặc thao tác admin).
- **Auth**:
  - Bearer admin token **hoặc**
  - header `X-Cron-Secret` (CronSecret).
- **Security best practices**:
  - Secret lưu trong env, không commit vào repo.
  - Rotation định kỳ (30/60 ngày hoặc khi nghi ngờ lộ):

        * Tạo secret mới.
        * Cập nhật vào secret manager/CI.
        * Redeploy cron job.
        * Vô hiệu hóa secret cũ.
        * Log thời điểm tạo/thu hồi để audit.
    - Áp dụng rate limiting, IP allowlist và log mọi request (timestamp, source, kết quả).
    - Khuyến khích dùng JWT admin cho thao tác tay; secret chủ yếu cho job nội bộ.
- **Response 200**: message, số lượng file xoá, timestamp.
- **Mã lỗi**:
    - 401 – thiếu/sai Bearer token hoặc X-Cron-Secret.
    - 403 – không phải admin hoặc secret sai.
    - 429 – vượt giới hạn tần suất gọi cleanup.

2.4.2 GET /admin/policy

- **Mục đích**: Xem cấu hình system policy hiện tại (giới hạn file size, validity, độ dài password tối thiểu, ...).
- **Auth**: JWT admin bắt buộc.
- **Response 200** (`SystemPolicy`):
    - `maxFileSizeMB`, `minValidityHours`, `maxValidityDays`, `defaultValidityDays`, `requirePas`
- **Mã lỗi**: 401 (thiếu/sai token), 403 (không phải admin).

2.4.3 PATCH /admin/policy

- **Mục đích**: Cập nhật một phần system policy.
- **Auth**: JWT admin.
- **Request body** (`SystemPolicyUpdate`): các trường tuỳ chọn với ràng buộc minimum (ví dụ maxFileSizeMB ≥ 1).
- **Response 200**: `message` + policy mới (`SystemPolicy`).
- **Mã lỗi**:
    - 400 – payload vi phạm ràng buộc (ví dụ maxValidityDays < minValidityHours).
    - 401, 403 – tương tự GET /admin/policy.

2.5 Components chính

2.5.1 Security Schemes

- **BearerAuth**:
  - Loại: HTTP bearer, bearerFormat: JWT.
  - Dùng cho: hầu hết các endpoint authenticated (user, files, admin).
- **CronSecret**:
  - Loại: apiKey, header `X-Cron-Secret`.
  - Dùng cho: `/admin/cleanup` trong bối cảnh cron job.
  - Lưu trong env, có cơ chế rotation và logging như mô tả ở trên.

2.5.2 Schemas tiêu biểu

- **User**: id (uuid), username, email, role (user/admin), totpEnabled.
- **File**: id, fileName, fileSize, mimeType, shareToken, shareLink, isPublic, hasPassword, availableFrom/To, validityDays, status (pending/active/expired), hoursRemaining, sharedWith[], owner, createdAt.
- **Error**: `error`, `message`, `code` – dùng chung cho mọi lỗi chuẩn hoá.
- Các schema khác: `RegisterRequest/Response`, `LoginRequest/Response`, `TOTPRequiredRespons` `TOTPSetupResponse`, `TOTPVerifyRequest`, `FileUploadRequest/Response`, `FileInfoResponse`, `UserProfileResponse`, `UserFilesResponse`, `SystemPolicy`, `SystemPolicyUpdate`.

2.6 Phụ lục

2.6.1 Validity Period Logic

- FROM + TO: hiệu lực từ FROM đến TO.
- Chỉ TO: hiệu lực từ hiện tại đến TO.
- Chỉ FROM: hiệu lực từ FROM đến FROM + defaultValidityDays (mặc định 7).
- Không có: hiệu lực từ hiện tại đến +7 ngày.

Validation bổ sung:

- `availableFrom` $\leq$ `availableTo`.
- `availableTo` không nằm trong quá khứ và không vượt `system_policy.maxValidityDays`.
- Nếu vi phạm: backend trả lỗi validation `invalidValidityRange`.

2.6.2 Security & Privacy Notes

- Password user và các khoá nhạy cảm được hash/bảo vệ an toàn (bcrypt, pattern 6 số cho TOTP).
- Download sử dụng streaming HTTP để hỗ trợ file lớn mà không chiếm nhiều RAM, kết hợp tiêu đề `Content-Length`, `Content-Disposition`.
- Anonymous download chỉ log mức tối thiểu: "Anonymous" + timestamp + trạng thái, không lưu IP/User-Agent để giảm rủi ro privacy.

2.6.3 Future Enhancements

- Email notification khi file gần hết hạn, khi file chuyển từ pending sang active, hoặc khi có lượt download mới (nhất là với file private/whitelist).
- Thiết lập quota theo user (giới hạn số/lượng dung lượng upload), quản lý versioning file (ghi đè hoặc tạo bản mới).
- Tích hợp cloud storage (S3, GCS, ...) và email service (SMTP, provider bên thứ ba).

CHƯƠNG 3

CI/CD PIPELINE

3.1 Tổng quan

Hệ thống CI/CD được xây dựng nhằm tự động hoá quá trình kiểm thử, đóng gói và triển khai ứng dụng *File Sharing Web*. Pipeline được thiết kế chạy hoàn toàn trên **self-hosted GitHub Actions Runner**, giúp nhóm chủ động trong việc kiểm soát môi trường build và triển khai.

Pipeline bao gồm ba giai đoạn chính:

- **Continuous Integration (CI)**: Kiểm thử, build hình ảnh Docker và xác nhận ứng dụng hoạt động ổn định.
- **Continuous Delivery (CD) – Build & Push**: Đóng gói và đẩy image production lên Docker Hub.
- **Continuous Deployment (CD) – Deploy**: Triển khai ứng dụng lên máy chủ production bằng Docker Compose.

Mỗi giai đoạn được định nghĩa như một *job* độc lập trong GitHub Actions, sử dụng chung runner có nhãn `[self-hosted, Linux, X64]`. Việc phân tách này đảm bảo pipeline có thể mở rộng, dễ debug, và tránh ảnh hưởng lẫn nhau giữa các bước build – deploy.

Pipeline được kích hoạt khi có:

- Push lên nhánh `main` hoặc `master`.
- Pull request nhằm kiểm thử trước khi hợp nhất.
- Lệnh thủ công thông qua `workflow_dispatch`.

3.2 CI: Build and Test

Giai đoạn CI đóng vai trò xác minh mã nguồn trước khi được đưa vào môi trường triển khai. Job *CI – Build and Test* được thực hiện trên runner nội bộ nhằm đảm bảo tính nhất quán giữa môi trường phát triển và sản xuất.

3.2.1 Các bước chính của CI

1. **Checkout mã nguồn**: Đảm bảo runner truy cập được toàn bộ repository.
2. **Thiết lập Docker Buildx**: Cho phép build multi-platform và sử dụng cache layer hiệu quả.
3. **Build hình ảnh backend, frontend và production**: Mỗi thành phần được build độc lập nhằm phát hiện lỗi sớm trong pipeline.
4. **Giải mã environment dành cho môi trường dev**: Nếu có công cụ SOPS trong hệ thống, pipeline tự động giải mã `dev.enc` thành `.env`.
5. **Chạy kiểm thử**: Bao gồm kiểm thử backend và frontend (có thể mở rộng tuỳ theo nhu cầu).
6. **Dọn dẹp hình ảnh tạm**: Loại bỏ các Docker image sử dụng cho quá trình kiểm thử.

3.2.2 Ý nghĩa của CI

CI giúp phát hiện lỗi sớm ngay trong quá trình phát triển, giảm thiểu rủi ro khi deploy. Khi tất cả các bước trong CI đều thành công, pipeline mới tiếp tục sang giai đoạn CD.

3.3 CD: Build, Push & Deploy

3.3.1 Build & Push Docker Image

Sau khi job CI pass, pipeline chạy job *Build and Push* (chỉ khi `push` vào `main/master`, `needs: test`):

1. Checkout repository.
2. Thiết lập Docker Buildx (đa tầng, multi-platform, hỗ trợ cache).
3. Đăng nhập Docker Hub qua GitHub Secrets.
4. Build image production từ `Dockerfile.production`, dùng cache-from/cache-to registry.
5. Push image lên Docker Hub với thẻ `:latest`.

3.3.2 Triển khai lên Production

Job *CD – Deploy to Production* thực hiện trên self-hosted runner/server:

1. Thiết lập khoá AGE cho SOPS từ GitHub Secrets.
2. Giải mã `prod.enc` thành `.env`; nếu thiếu `.env` thì fail.
3. Đăng nhập Docker Hub.
4. **Tắt** container cũ: `docker compose down` (bỏ qua lỗi nếu không có).
5. **Pull** image mới: `playernem/dath20251_file-sharing:latest`; in thông tin image và digest.

6. **Khởi chạy** bằng `docker compose up -d -force-recreate -pull always -no-deps app`; kiểm tra container đang dùng image nào (`docker inspect file-sharing-app`); hiển thị `docker compose ps`.
7. **Health-check cục bộ**: vòng lặp tối đa 30 lần, mỗi lần chờ 2s; backend `/health` phải OK, frontend phải trả 200/304 (502/503 được coi là chưa sẵn sàng và sẽ retry).
8. Nếu health-check không đạt: in trạng thái, `docker compose logs -tail=100`, thử `supervisorctl status`, sau đó fail.
9. Bước "Show container status": in `docker compose ps`, log 100 dòng, `supervisorctl status`, kiểm tra HTTP frontend (200/304 vs 502/503).
10. Dọn dẹp image cũ: `docker image prune -f`.
11. Thông báo thành công, kèm quick health check `curl -s http://localhost/health`.

3.3.3 Health-check

- Backend: `/health` phải phản hồi thành công.
- Frontend: yêu cầu HTTP 200/304; nếu 502/503 thì tiếp tục chờ trong giới hạn 30 lần (2s mỗi lần).
- Khi lỗi: log chi tiết từ Docker Compose và `supervisorctl` để hỗ trợ debug.

Nếu bất kỳ thành phần nào không hoạt động đúng, pipeline dừng và thông báo lỗi đến người phát triển.

3.4 Kết luận

Pipeline CI/CD được xây dựng giúp tự động hoá toàn bộ quy trình:

1. Kiểm thử chất lượng ứng dụng.
2. Đóng gói và đẩy lên Docker Hub.
3. Triển khai ổn định lên môi trường production.

Cách tổ chức ba giai đoạn độc lập giúp hệ thống dễ mở rộng, dễ bảo trì và đảm bảo độ an toàn trong triển khai. Việc sử dụng self-hosted runner cũng mang lại lợi ích lớn khi có thể tùy biến môi trường triển khai theo nhu cầu của dự án.