Feedback API Documentation cho Team BE

Ngày: 15/07/2025

Từ: Kiến Hưng (Lead of BA Team)

Mục đích: Cung cấp phản hồi và các điểm cần làm rõ/cập nhật trong tài liệu API WEBHUB - API DOC.pdf để đảm bảo tính nhất quán, chính xác và đầy đủ với các tài liệu yêu cầu và thiết kế cơ sở dữ liệu mới nhất (SRS.pdf, Thiết kế Cơ sở Dữ liệu.pdf, và các tài liệu Yêu cầu Chức năng module cụ thể).

Disclaimer: Tài liệu này được viết với sự tham khảo của AI nhằm mục đích tìm ra vấn đề trong API-DOC cũng như những điểm không nhất quán với tài liệu của BA Team viết. Tài liệu nhằm mục đích tham khảo, không phải văn bản chính thức đề xuất sửa lỗi.

---

1. Nhận xét chung về cấu trúc và định dạng API Doc

- **Tính nhất quán trong tên Endpoint:** Hầu hết các endpoint sử dụng danh từ số nhiều (`/users`, `/projects`, `/blog-posts`). Tuy nhiên, có một số ngoại lệ (`/api/v1/event/create`, `/api/v1/event/{title}`, `/api/v1/evnt/delete`).
    - **Đề xuất:** Chuẩn hóa tất cả các endpoint thành danh từ số nhiều (ví dụ: `/events`) và sử dụng các động từ HTTP phù hợp (ví dụ: `POST /events`, `PUT/PATCH /events/{id}`, `DELETE /events/{id}`).
- **Tham số path và query:** Cần rõ ràng hơn về định dạng nếu tham số là danh sách (ví dụ: `tag=id1,id2` hay `tag=[id1,id2]`).
- **Header `Authorization` cho Public Endpoints:** Nhiều endpoint public (ví dụ: `GET /api/v1/projects`, `GET /api/v1/projects/{id}`, `GET /api/v1/documents/public`, `GET /api/v1/documents/public/{id}`, `GET /api/v1/events/views`, `GET /api/v1/events/search`, `GET /tags`) vẫn yêu cầu Header `Authorization: Bearer <access_token>` là `Có` (Bắt buộc).
    - **Đề xuất:** Đối với các endpoint có `Quyền: GUEST` hoặc `Tất cả người dùng (bao gồm cả không đăng nhập)`, header `Authorization` nên là `Không` (Không bắt buộc).
- **Định nghĩa Response `Success` và `Failure`:** Mẫu chung cung cấp (`<args[1]> : <type>`) nhưng không được áp dụng nhất quán trong các response cụ thể (ví dụ: thiếu kiểu dữ liệu, các trường thêm/bớt).
    - **Đề xuất:** Cung cấp kiểu dữ liệu rõ ràng cho tất cả các trường trong response, đặc biệt là các mảng hoặc đối tượng lồng nhau.

---

2. Phản hồi chi tiết theo từng Module

2.1. II.1. Authentication & Authorization (Phúc)

- **1.1 Đăng nhập (POST /api/v1/auth/login):**
  - API Doc sử dụng `username: String`, trong khi FRD M08 UC-AUTH-01 quy định đăng nhập bằng `email`.
  - **Đề xuất:** Đổi `username` thành `email` trong request body để đồng bộ với FRD và cơ sở dữ liệu.
- **1.3 Quên mật khẩu (POST /api/v1/auth/forgot-password) và Đặt lại mật khẩu (GET/POST /api/v1/auth/reset-password):**
  - Endpoint `GET /api/v1/auth/reset-password?token={token}` trong API Doc chỉ ghi `Mô tả: Chuyển sang trang đổi mật khẩu` và không cung cấp chi tiết response.
  - **Đề xuất:** Làm rõ nếu endpoint `GET` này trả về HTML hay một JSON payload để frontend xử lý. Nếu là frontend redirect, không cần API endpoint chi tiết.
- **1.4 Làm mới token (GET /api/v1/auth/refresh):**
  - API Doc không cung cấp bất kỳ chi tiết nào (Mô tả, Headers, Request body, Response, Failure). FRD M08 không đề cập cụ thể về chức năng refresh token, mặc dù đây là một tính năng bảo mật phổ biến.
  - **Đề xuất:** Cần bổ sung đầy đủ mô tả, headers (có thể là `Authorization` với refresh token), request/response body (thường trả về `accessToken` và `refreshToken` mới), và các trường hợp lỗi. Đồng thời, xác nhận lại với BA xem tính năng này có thuộc phạm vi của phiên bản 1.1 hay không.

2.2. II.2. User management & User profile management (Trần kiệt)

- **Cơ sở dữ liệu `users`:** Cột `role` trong DB là `VARCHAR` (`'ROOT'`, `'ADMIN'`, `'MEMBER'`), nhưng trong API Doc (2.2 Tạo tài khoản người dùng mới), `role` lại là `boolean`.
  - **Đề xuất (Quan trọng):** Đồng bộ kiểu dữ liệu của `role` thành `String` (hoặc `VARCHAR`) với các giá trị rõ ràng (`"MEMBER"`, `"ADMIN"`, `"ROOT"`) trong tất cả các request/response liên quan đến người dùng.
- **2.2 Tạo tài khoản người dùng mới (POST /api/v1/users):**
  - Request body thiếu trường `password`. FRD M12 UC-UM-02 quy định hệ thống sẽ `Tạo một mật khẩu ngẫu nhiên, an toàn cho người dùng mới` và `Gửi một email đến người dùng mới, chứa thông tin đăng nhập`.
  - **Đề xuất:** Làm rõ trong API Doc rằng `password` không cần gửi từ frontend vì nó được tạo tự động bởi backend, hoặc nếu frontend cần gửi một mật khẩu khởi tạo, thì phải thêm trường `password` vào request body.

- **2.3 Xem và Chỉnh sửa thông tin người dùng và vô hiệu hóa tài khoản (PATCH /api/v1/users/{id}):**
    - Request body liệt kê `avatar_url:string` và `is_active:boolean`. Nếu `avatar_url` được gửi trực tiếp, `Content-Type` nên là `application/json`. Nếu là file upload, `Content-Type` là `multipart/form-data` và cần có trường `file` thay vì `avatar_url` trong request body. API Doc liệt kê `multipart/form-data` nhưng `avatar_url` trong body.
    - Endpoint này được dùng để vô hiệu hóa tài khoản (M12 UC-UM-04), nhưng trong API Doc `PATCH` lại kết hợp với `Chỉnh sửa các thông tin của user`. Điều này không rõ ràng.
    - Response bao gồm `email`, nhưng FRD M16 UC-UP-01 và M12 UC-UM-03 đều ghi `email` là `Chỉ đọc`.
    - **Đề xuất:**
        - Tách biệt rõ ràng cách gửi dữ liệu: nếu gửi file avatar, request body nên có trường `file` (dạng `multipart/form-data`) và các trường khác dạng JSON; nếu gửi URL avatar, tất cả nên là `application/json`.
        - Tạo endpoint riêng hoặc mô tả rõ ràng cách gửi request để `vô hiệu hóa/kích hoạt` tài khoản (ví dụ: `PATCH /api/v1/users/{id}/status` với `{"is_active": boolean}`).
        - Làm rõ `email` trong response là `read-only`.
- **2.5 Xóa tài khoản người dùng (chỉ Root) (DELETE /api/v1/users):**
    - API Doc yêu cầu `email:string` trong request body để xác nhận xóa. FRD M12 UC-UM-05 cũng xác nhận bằng email. Response chứa `id` và `title` (không phải trường của user).
    - **Đề xuất:**
        - Response cho hành động xóa tài khoản nên trả về `id` và `message` xác nhận thành công. Trường `title` là không phù hợp.
        - Xác nhận lại việc dùng `email` trong body cho DELETE request; thường `id` trong path param là đủ.
- **Tổng thể User Management & Profile:** Kiểm tra lại tất cả các trường `String` và `boolean` cho cột `role` và `is_active` để đảm bảo nhất quán với DB (`VARCHAR` và `BOOLEAN`).

2.3. II.3 Blog management & Public Blog view (Lợi)

- **RẤT QUAN TRỌNG: MÔ HÌNH DỮ LIỆU KHÔNG KHỚP VỚI THIẾT KẾ CSDL MỚI NHẤT!**
    - Tài liệu "Thiết kế Cơ sở Dữ liệu – WebHub HCMUTE RTIC.pdf" (v1.1, ngày 09/07/2025) đã chỉ rõ:
        - `Xóa bảng blog_categories; Xóa cột category_id khỏi bảng blog_posts.`

        - Thêm liên kết từ taggables đến documents. (có nghĩa là taggables là bảng trung gian cho blog_posts và content_tags cũng).
    - Tuy nhiên, API Doc vẫn đang sử dụng category_id và categories trong tất cả các endpoint liên quan đến blog.
    - **Đề xuất (Ưu tiên cao nhất):**
        - **Xóa bỏ hoàn toàn mục 5.2. Quản lý Danh mục Blog (/blog-categories)** vì bảng blog_categories đã bị xóa.
        - **Cập nhật tất cả các endpoint Blog** (5.1. Quản lý Bài viết Blog) để sử dụng tag_ids (mảng các ID integer từ bảng content_tags) thay cho category_ids.
        - Response cho Blog Post Object cần trả về tags (mảng các đối tượng {id: integer, name: string} hoặc mảng string tên tag) thay vì categories.
        - Đảm bảo 5.3. Quản lý Thẻ (/tags) (dựa trên M27) là cách duy nhất để quản lý các tag chung và cách gắn tag vào blog posts (và các loại nội dung khác) qua bảng taggables.
- **5.1.1 Lấy danh sách tất cả bài viết blog (GET /blog-posts):**
    - category_id trong query parameters cần đổi thành tag_id hoặc tag_name.
    - Quyền truy cập: M20 FRD (UC-PUB-BLOG-01) ghi Guest có thể xem, API Doc ghi Member/Admin.
    - **Đề xuất:** Cần làm rõ quyền truy cập cho Guest. Guest nên được xem các bài viết is_public=true. Nếu có filter is_public=false, thì mới cần Authorization và quyền cao hơn.
- **5.1.2 Lấy thông tin chi tiết bài viết blog theo ID (GET /blog-posts/{id}):**
    - Quyền truy cập: API Doc ghi Member/Admin. M20 FRD (UC-PUB-BLOG-03) ghi Guest có thể đọc chi tiết bài viết.
    - **Đề xuất:** Guest cần được quyền đọc các bài viết công khai. Header Authorization cho Guest nên là Không bắt buộc.
- **5.1.3 Tạo bài viết blog mới (POST /blog-posts):**
    - Trong Request body, cần đổi category_ids thành tag_ids (mảng các ID integer). Trường tags (mảng các string) cũng xuất hiện, gây nhầm lẫn.
    - **Đề xuất:** Chuẩn hóa: Chỉ nên dùng tag_ids (array of integers) trong request body để gắn các tag đã tồn tại. Nếu muốn tạo tag mới qua endpoint này, cần mô tả rõ ràng.
- **5.1.6 Cập nhật trạng thái bài viết blog (Ẩn/Xuất bản) (PATCH /blog-posts/{id}/status):**
    - is_public và is_featured là các trường boolean riêng biệt trong Blog Post Object. API Doc chỉ đề cập status thay đổi is_public tự động (FR-STATUS-003). Chức năng Ghim/Bỏ ghim (is_pinned/is_featured - UC-BLOG-06) không có trong endpoint này.
    - **Đề xuất:**

        - Làm rõ cách cập nhật `is_featured` và `is_public`. Nếu chúng được cập nhật thông qua `status`, cần mô tả chi tiết logic này trong API Doc.
          Nếu `is_featured` là một trường có thể cập nhật riêng biệt, cần bổ sung vào request body hoặc một endpoint riêng.
        - Đồng bộ `is_pinned` và `is_featured`: CSDL dùng `is_pinned`, FRD dùng `is_featured`. Cần thống nhất một tên.
- **5.3. Quản lý Thẻ (/tags):**
    - API Doc hiện tại mô tả các endpoint (`POST, PUT, DELETE /api/v1/projects/{id}/tags`) trong mục Project Management nhưng không có trong mục Blog Management.
    - **Đề xuất (Quan trọng):**
        - Tạo một phần riêng cho "Quản lý Tag chung" (dựa trên M27 FRD) với các endpoint `GET /tags`, `POST /tags`, `PUT /tags/{id}`, `DELETE /tags/{id}`.
        - Các endpoint gắn/gỡ tag cho từng loại nội dung (`blog-posts`, `projects`, `documents`, `events`) nên sử dụng cấu trúc thống nhất (ví dụ: `POST /api/v1/blog-posts/{id}/tags` với `tag_id` trong body, và `DELETE /api/v1/blog-posts/{id}/tags/{tag_id}`). Tránh gửi `name` trong body cho DELETE request.

2.4. II.4 Project management & Public project view (Khoa & Phương)

- **4.1.1 Xem danh sách dự án (GET /api/v1/projects):**
    - Query param `status` không được M14 FRD UC-PROJ-01 đề cập làm bộ lọc cho view public. FRD chỉ ghi `type` và `is_public`.
    - **Đề xuất:** Kiểm tra lại liệu có hỗ trợ filter `status` cho public view không. Nếu có, cần cập nhật FRD.
- **4.2.1 Tạo dự án mới (POST /api/v1/projects):**
    - Request body có các trường `slug`, `creator_id`, `created_at` (thường do backend tự sinh/quản lý) và `url` (không có trong schema DB, có thể là `git_url` hoặc `production_url`). Thiếu trường `type` (INTERNAL, COLLAB, RESEARCH) theo DB và FRD.
    - **Đề xuất:**
        - Bỏ `slug`, `creator_id`, `created_at` khỏi request body.
        - Đổi `url` thành `git_url` và `production_url` hoặc làm rõ mục đích của `url`.
        - Thêm trường `type: String` vào request body.
        - Làm rõ mối quan hệ giữa `status` và `is_public` trong request, và cách chúng được lưu trong DB. DB có cả 2 trường.
- **4.2.2 Cập nhật dự án (PUT /api/v1/projects/{id}):**

    - Request body chứa `id` (redundant với path param) và `updated_at` (do backend quản lý). Thiếu nhiều trường có thể cập nhật (`type`, `short_description`, `start_date`, `is_public`, `git_url`, `production_url`, `tags`).
    - **Đề xuất:**
        - Bỏ `id` và `updated_at` khỏi request body.
        - Kiểm tra lại tất cả các trường có thể cập nhật và thêm chúng vào request body. Sử dụng `PATCH` thay `PUT` nếu chỉ hỗ trợ cập nhật từng phần.
- **4.2.3 Xóa dự án (DELETE /api/v1/projects/{id}):**
    - Query param `status` trong endpoint `GET /api/v1/projects/{id}?status=status.` là không hợp lý cho DELETE. Request body chứa `title` trong response (không phù hợp).
    - **Đề xuất:**
        - Loại bỏ query param `status` khỏi endpoint DELETE.
        - Response nên chỉ bao gồm `message` và `id` của dự án đã xóa.
- **4.3 Quản lý tags của project:**
    - Như đã đề cập ở mục 2.3, phần này cần được xem xét lại để đồng bộ với cách quản lý tag chung (module M27) và việc sử dụng bảng `taggables`.

2.5. II.5 Document management & Public document view (Anh Hưng)

- **Endpoint `public`:** Nhiều endpoint `GET` public (`/documents/public` và `/documents/public/{id}`) vẫn yêu cầu `Authorization: Bearer` là `Có`.
    - **Đề xuất:** Đổi `Authorization` thành `Không` cho các endpoint public.
- **Mô hình dữ liệu `documents`:** DB schema không có trường `is_public` cho bảng `documents`, nhưng API Doc lại có trong request body và response của endpoint `POST/PATCH /api/v1/documents`.
    - **Đề xuất (Quan trọng):** Đồng bộ lại. Nếu tài liệu có thể là public, cần thêm trường `is_public` vào schema `documents`. Nếu không, cần xóa khỏi API Doc. Tương tự với trường `status` trong response của POST/PATCH, cần làm rõ `status` này trong DB schema của `documents`.
- **5.2.1 Tạo tài liệu (POST /api/v1/documents):**
    - M24 FRD UC-MDS-01 đề cập `(Tùy chọn) Gắn thẻ cho tài liệu.` Trường `tags` không có trong request body của API Doc.
    - **Đề xuất:** Thêm khả năng gắn tag (ví dụ: `tag_ids: array of integers`) vào request body.
- **5.2.3 Xóa tài liệu (DELETE /api/v1/documents/{id}):**
    - M24 FRD UC-MDS-02 nói về soft delete (`Tạm ẩn`) và full delete (`Xóa vĩnh viễn`). API Doc không có tham số để phân biệt 2 loại xóa này.

    - **Đề xuất:** Thêm một query parameter (ví dụ: `?force_delete=true`) hoặc trường trong request body để chỉ định loại xóa.
- **5.2.4 Quản lý phân loại tài liệu (UC-DM-05):**
    - Tương tự như module Blog/Project, các endpoint gắn/gỡ tag cần đồng bộ với hệ thống quản lý tag chung (M27) và bảng `taggables`. Quyền truy cập cũng cần làm rõ: M24 FRD UC-MDS-06 ghi Member có thể gắn tag cho tài liệu của họ. API Doc chỉ ghi Admin.
    - **Đề xuất:**
        - Đồng bộ các endpoint quản lý tag cho tài liệu với cách quản lý tag chung.
        - Cập nhật quyền truy cập cho Member nếu họ có thể gắn tag cho tài liệu của mình.
        - Endpoint `GET /api/v1/documents/{id}/tags` (lấy danh sách tag của tài liệu) nên có quyền Guest nếu tài liệu là public.

2.6. II.6 Event management & Event page (Chí quốc)

- **Cơ sở dữ liệu `events`:** Các trường `tag`, `register`, `detail`, `member` trong API Doc không nhất quán với DB schema. DB có `signup_link`, `cover_image_url`, `start_time`, `end_time`.
    - **Đề xuất (Quan trọng):**
        - Đổi `tag` thành `tags` (mảng các object/string) trong request/response.
        - Đổi `register` thành `signup_link`.
        - Đổi `image` thành `cover_image_url`.
        - Đổi `startDate` thành `start_time` và `endDate` thành `end_time`.
        - Kiểm tra và loại bỏ hoặc làm rõ mục đích của các trường `detail` và `member` nếu chúng không có trong DB hoặc được lấy từ nguồn khác.
- **6.1 Xem danh sách sự kiện (GET /api/v1/events/views):**
    - Đường dẫn `/events/views` là không chuẩn theo RESTful.
    - **Đề xuất:** Đổi thành `GET /api/v1/events`.
- **6.2 Tìm kiếm sự kiện (GET /api/v1/events/?title=...&year=...&location=...&status=...):**
    - Response hiển thị một object sự kiện, nhưng tìm kiếm thường trả về một *danh sách*.
    - `year` và `location` là query params, nhưng M22 FRD UC-PUB-EV-02 chỉ nói tìm kiếm theo tên.
    - **Đề xuất:**
        - Response nên trả về một *mảng* các sự kiện.
        - Làm rõ việc hỗ trợ tìm kiếm theo `year` và `location` trong FRD.
- **6.3 Tạo sự kiện mới (POST /api/v1/event/create):**
    - Đường dẫn `/api/v1/event/create` không nhất quán.

    - **Đề xuất:** Đổi thành `POST /api/v1/events`.
- **6.4 Cập nhật thông tin sự kiện (PUT /api/v1/event/{title}):**
    - Cập nhật bằng `title` trong path param là không nên (không đảm bảo unique, thay đổi khó quản lý). `PUT` thay vì `PATCH` cũng cần xem xét.
    - **Đề xuất:**
        - Đổi thành `PATCH /api/v1/events/{id}` để cập nhật theo ID và cho phép cập nhật từng phần.
        - Header `Content-Type` cần làm rõ nếu có file upload.
- **6.5 Xóa sự kiện (DELETE /api/v1/evnt/delete):**
    - Đường dẫn `/api/v1/evnt/delete` và việc truyền `id`, `title` trong request body là không chuẩn.
    - **Đề xuất:** Đổi thành `DELETE /api/v1/events/{id}`. Request body nên để trống (hoặc có thể có `confirm: boolean` nếu cần).

2.7. III. Các module nội bộ (Email Service, Log Service)

- API Doc chỉ liệt kê tên module mà không có bất kỳ chi tiết API nào. SRS có đề cập Email Service là Dependency.
    - **Đề xuất:** Làm rõ phạm vi của "internal modules" trong API Doc này.
        - Nếu là API được expose bởi dự án WebHub cho các service nội bộ khác sử dụng, cần có tài liệu API đầy đủ.
        - Nếu chỉ là cách WebHub *gọi* các service khác, thì không cần API doc chi tiết ở đây, nhưng cần ghi chú rõ ràng về việc tích hợp.

---

3. Đề xuất chung để cải thiện tài liệu API:

- **Tổng hợp Data Model:** Tạo một phần "Data Models" hoặc "Objects" riêng biệt ở đầu tài liệu, định nghĩa rõ ràng cấu trúc của các đối tượng (User, Blog Post, Project, Event, Document, Tag) với tất cả các trường, kiểu dữ liệu, ràng buộc (bắt buộc/tùy chọn) và ghi chú. Điều này giúp tránh lặp lại và dễ dàng tham chiếu.
- **Chuẩn hóa Error Responses:** Đảm bảo tất cả các endpoint trả về lỗi theo một định dạng thống nhất (ví dụ: `{"error": "message", "code": 4xx/5xx}`).
- **Minh họa Request/Response:** Sử dụng JSON ví dụ đầy đủ và chính xác cho mọi request body và response body.
- **Vai trò và Quyền truy cập:** Rõ ràng về vai trò nào được phép truy cập từng endpoint và hành động cụ thể, đặc biệt là khi có sự khác biệt giữa Member, Admin, Root và Guest.
