**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN 1**



****

**BÀI TẬP LỚN**

**PHÁT TRIỂN CÁC HỆ THỐNG THÔNG MINH**

**Đề tài: Hệ thống tóm tắt văn bản tự động**

| **STT** | **Họ và tên** | **Mã sinh viên** |
| --- | --- | --- |
| **1** | **Trương Hoàng Nguyên** | **B22DCCN597** |
| **2** | **Nguyễn Phong Nhã** | **B22DCCN573** |
| **3** | **Phạm Thị Hương Nhài** | **B22DCCN574** |

| **Giảng viên hướng dẫn** | **: Nguyễn Mạnh Hùng** |
| --- | --- |
| **Nhóm** | **: 06** |

***Hà Nội – 2025***

# Mục lục

[**Mục lục 2**](#_kvnibfz3c74v)

[**Chương I: Thiết kế hệ thống chung 4**](#_6hcx8e4a8h4e)

[1. Kiến trúc tổng thể: 4](#_f661wdcabic9)

[2. Mô tả chi tiết các module 5](#_tl3upjr9j9gq)

[1.1. Quản lý chức năng admin 5](#_albv7vn1hv5z)

[1.2. Quản lý thông tin người dùng 6](#_q4sdfbq2501q)

[1.3. Tóm tắt văn bản 6](#_gs50zruv9m0d)

[1.4. Lịch sử tóm tắt 7](#_whs7n8dexbvl)

[1.5. Tìm kiếm văn bản tóm tắt 7](#_fpesbain49pb)

[3. Phân chia công việc cá nhân 7](#_di0hopfkazbl)

[**Chương II: Thiết kế hệ thống theo cá nhân 8**](#_y77hddczweww)

[1. Phạm Thị Hương Nhài - B22DCCN574 8](#_9cp489g9lhe5)

[1.1. Thiết kế cơ sở dữ liệu liên quan đến module 8](#_wxntsnhp4dwt)

[1.1.1. Module quản lý thông tin người dùng 8](#_pzgq2gqqu8vu)

[1.1.2. Module lịch sử tóm tắt 8](#_25fbkka1njxy)

[1.2. Thiết kế lớp thực thể liên quan đến module 9](#_d8vr0guwzc41)

[1.2.1. Module quản lý thông tin người dùng 9](#_y6r5h89oqza9)

[1.2.2. Module lịch sử tóm tắt 9](#_yg66ze4rixpi)

[1.3. Thiết kế chi tiết các chức năng 10](#_a9sgjylq6xl7)

[1.3.1. Chức năng đăng ký tài khoản mới: 10](#_cjj3bvr8x6x6)

[a. Thiết kế giao diện 10](#_nf849xqberxj)

[b. Thiết kế biểu đồ lớp chi tiết 11](#_aczznqskjjql)

[c. Thiết kế biểu đồ hoạt động 12](#_vuaupbupwztz)

[d. Thiết kế biểu đồ tuần tự 13](#_lkc7ceaveoor)

[1.3.2. Chức năng đăng nhập/ đăng xuất: 16](#_hlys3zv36256)

[a. Thiết kế giao diện 16](#_n4h4fmg098m)

[b. Thiết kế biểu đồ lớp chi tiết 18](#_aitpx29jn8bm)

[c. Thiết kế biểu đồ hoạt động 21](#_m7mhb6nydlvb)

[d. Thiết kế biểu đồ tuần tự 21](#_wexxm0xws048)

[1.3.3. Chức năng chỉnh sửa thông tin cá nhân: 24](#_9qs9jcdn8m07)

[a. Thiết kế giao diện 24](#_5bl8lp3rl4jp)

[b. Thiết kế biểu đồ lớp chi tiết 25](#_nd5ma16svf64)

[c. Thiết kế biểu đồ hoạt động 27](#_v4nks4ftl9ef)

[d. Thiết kế biểu đồ tuần tự 28](#_wzmfwp9v3eg7)

[1.3.4. Chức năng xem danh sách lịch sử tóm tắt: 31](#_94iixu5lici2)

[a. Thiết kế giao diện 31](#_y9cb8dn56p9o)

[b. Thiết kế biểu đồ lớp chi tiết 33](#_kxhj10gj9fbn)

[c. Thiết kế biểu đồ hoạt động 36](#_rh4mnebngkgd)

[d. Thiết kế biểu đồ tuần tự 37](#_ufkp1f3x49uz)

[1.3.5. Chức năng xóa lịch sử tóm tắt: 39](#_xp02qbfe6jbr)

[a. Thiết kế giao diện 39](#_cekhn7xdzl7q)

[b. Thiết kế biểu đồ lớp chi tiết 40](#_q4kxeufj9gj7)

[c. Thiết kế biểu đồ hoạt động 43](#_vrnv7eyjln4y)

[d. Thiết kế biểu đồ tuần tự 44](#_e25yb91jrgdh)

[2. Trương Hoàng Nguyên - B22DCCN597 47](#_cfisghn0hgac)

[2.1. Thiết kế cơ sở dữ liệu liên quan đến module 47](#_ws62n7onwo4i)

[2.2. Thiết kế lớp thực thể liên quan đến module 47](#_drf4bzl8b85a)

[2.3. Các chức năng phụ trách 48](#_47ajxx82i1ho)

[2.3.1. Chức năng tóm tắt văn bản, file: 48](#_ij791ysi3me0)

[a. Thiết kế giao diện 48](#_v7cd6ba6hcwy)

[b. Sơ đồ lớp chi tiết 48](#_gdwh6gnpp08a)

[c. Sơ đồ hoạt động 51](#_wa2kekem065t)

[d. Sơ đồ tuần tự 51](#_u8lsifihue2t)

[2.3.2. Chức năng tìm kiếm văn bản: 53](#_y8ydkmxj9bmo)

[a. Thiết kế giao diện 53](#_l7ibstntki1z)

[b. Sơ đồ lớp chi tiết 54](#_8ts1hc87da5c)

[c. Sơ đồ hoạt động 55](#_k25bipew79t)

[d. Sơ đồ tuần tự 56](#_wpjhcemlgp6d)

[2.3.3. Chức năng chỉnh sửa văn bản tóm tắt: 56](#_z9krkvmi2ift)

[a. Thiết kế giao diện 56](#_72wnbr2fyay8)

[b. Sơ đồ lớp chi tiết 57](#_3fq1tg9akey9)

[c. Sơ đồ hoạt động 59](#_p9lw82aqf41v)

[d. Sơ đồ tuần tự 59](#_68ccih3e4uir)

[3. Nguyễn Phong Nhã - B22DCCN573 60](#_o25lmwz028mc)

[3.1. Thiết kế cơ sở dữ liệu liên quan đến module 60](#_a0ffdjrwa6ts)

[3.2. Thiết kế lớp thực thể liên quan đến module 60](#_o7qmym273am4)

[3.3 Các chức năng phụ trách 61](#_pdkjs1kmnrtr)

[3.3.1 Chức năng Finetune Model 61](#_flvwqfni321e)

[3.3.2 Chức năng xóa dữ liệu mẫu 65](#_a3ozs769xl51)

[3.3.3 Chức năng Thêm mới Dataset 70](#_ak5sg1ush7wj)

# 

# 

# Chương I: Thiết kế hệ thống chung

## **1. Kiến trúc tổng thể:**

# 

Hệ thống tóm tắt văn bản tự động được thiết kế theo mô hình **Client–Server phân tầng**, bao gồm 2 cặp client–server phục vụ cho 2 nhóm người dùng chính:

* **Quản lý (Admin)**: thực hiện điều hành, theo dõi và cấu hình hệ thống.
* **Người dùng (User)**: sử dụng tính năng tóm tắt, tra cứu và quản lý lịch sử.

Cụ thể gồm các phần:

| **Thành phần** | **Vai trò chính** | **Mô tả** |
| --- | --- | --- |
| **Client 1 (Website quản lý)** | Giao diện dành cho quản trị viên | Cho phép admin quản lý bộ dữ liệu, quản lý, finetune mô hình |
| **Server 1 (Thuật toán)** | Xử lý và triển khai mô hình tóm tắt | Chứa các mô hình TF–IDF, TextRank, PhoBART, ViT5; tiếp nhận yêu cầu tóm tắt từ Server 2, xử lý và trả kết quả. |
| **CSDL** | Lưu trữ dữ liệu huấn luyện & tham số mô hình | Gồm trọng số, tập huấn luyện, kết quả so sánh đánh giá mô hình. |
| **Server 2 (Ứng dụng)** | Xử lý logic nghiệp vụ và điều phối dữ liệu | Nhận yêu cầu từ người dùng, xác thực, gọi API tới Server 1, lưu kết quả vào CSDL ứng dụng. |
| **Client 2 (Website người dùng)** | Giao diện sử dụng chính | Cho phép người dùng đăng nhập, nhập văn bản, xem bản tóm tắt, lưu/xóa lịch sử, tìm kiếm theo từ khóa. |
| **CSDL Ứng dụng** | Lưu trữ thông tin người dùng, lịch sử, log | Lưu thông tin tài khoản, các bản tóm tắt, lịch sử thao tác, từ khóa tìm kiếm. |

## **2. Mô tả chi tiết các module**

Hệ thống tóm tắt văn bản tự động được chia thành 5 nhóm module chính, tương ứng với các nhóm chức năng nghiệp vụ mà người dùng và quản trị viên có thể thực hiện.

| **Nhóm module** | **Đối tượng sử dụng** | **Vai trò chính** |
| --- | --- | --- |
| **Quản lý chức năng admin** | Quản trị viên | Quản lý bộ dữ liệu & fine tune mô hình |
| **Quản lý thông tin người dùng** | Người dùng | Đăng nhập, cập nhật và bảo mật thông tin |
| **Tóm tắt văn bản** | Người dùng | Thực hiện thao tác tóm tắt, chỉnh sửa tóm tắt |
| **Lịch sử tóm tắt** | Người dùng | Quản lý và xóa lịch sử tóm tắt |
| **Tìm kiếm văn bản tóm tắt** | Người dùng | Tra cứu nhanh các bản tóm tắt đã lưu |

### ***1.1. Quản lý chức năng admin***

**Mục đích:** Cho phép quản trị viên quản lý bộ dữ liệu, quản lý, fine tune mô hình.

Chức năng chi tiết:

* Quản lý mô hình tóm tắt:
  + - Chọn mô hình mặc định
    - Cập nhật trọng số, huấn luyện lại mô hình.
* **Thống kê hiệu năng:** Hiển thị chỉ số đánh giá mô hình (Precision, Recall, F1, ROUGE).
* **Quản lý bộ dữ liệu:** Thêm sửa xóa bộ dữ liệu.

### ***1.2. Quản lý thông tin người dùng***

**Mục đích:** Hỗ trợ người dùng đăng nhập, chỉnh sửa và bảo mật thông tin cá nhân.

**Chức năng chi tiết:**

* **Đăng ký tài khoản mới:** Nhập thông tin cơ bản (họ tên, email, mật khẩu).
* **Đăng nhập / Đăng xuất:** Xác thực người dùng, lưu trạng thái phiên đăng nhập.
* **Chỉnh sửa thông tin cá nhân:** Thay đổi tên hiển thị, mật khẩu, email, ảnh đại diện.
* **Khôi phục mật khẩu:** Gửi email khôi phục hoặc xác thực OTP.

### ***1.3. Tóm tắt văn bản***

**Mục đích:** Cho phép người dùng nhập hoặc tải văn bản để hệ thống xử lý và sinh ra bản tóm tắt.

**Chức năng chi tiết:**

* **Thêm mới bản tóm tắt:**
  + - Nhập trực tiếp văn bản hoặc tải file .txt, .docx, .pdf.
    - Chọn mô hình tóm tắt mong muốn (hoặc dùng mặc định).
* **Xử lý và hiển thị kết quả:** Hiển thị nội dung tóm tắt, độ dài, và mô hình sử dụng.
* **Chỉnh sửa bản tóm tắt:** Cho phép người dùng hiệu chỉnh, lưu lại phiên bản cập nhật.

### ***1.4. Lịch sử tóm tắt***

**Mục đích:** Lưu trữ và quản lý các bản tóm tắt mà người dùng đã thực hiện.

**Chức năng chi tiết:**

* **Xem danh sách lịch sử:** Hiển thị toàn bộ bản tóm tắt theo thời gian.
* **Lọc lịch sử theo ngày:** Tìm nhanh các bản tóm tắt theo khoảng thời gian cụ thể.
* **Xóa từng bản tóm tắt:** Cho phép xóa riêng từng bản khi không cần thiết.
* **Xóa toàn bộ lịch sử:** Dọn sạch dữ liệu lịch sử để tiết kiệm dung lượng.

### ***1.5. Tìm kiếm văn bản tóm tắt***

**Mục đích:** Hỗ trợ người dùng tra cứu nhanh các bản tóm tắt đã thực hiện.

**Chức năng chi tiết:**

* **Tìm kiếm theo từ khóa:** Nhập từ khóa, hệ thống tìm trong tiêu đề
* **Gợi ý tìm kiếm:** Hiển thị danh sách tóm tắt gần đây hoặc có nội dung tương tự.
* **Xem chi tiết bản tóm tắt:** Mở nhanh bản tóm tắt được chọn từ kết quả tìm kiếm.

## **3. Phân chia công việc cá nhân**

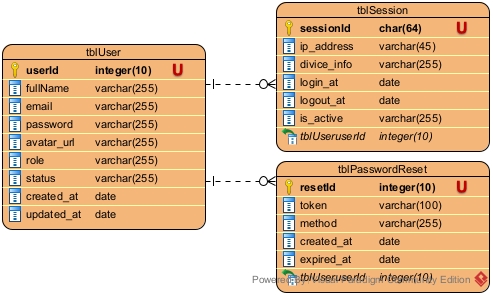
| Tên | MSV | Module phụ trách |
| --- | --- | --- |
| Phạm Thị Hương Nhài | B22DCCN574 | - Quản lý thông tin người dùng  - Lịch sử tóm tắt |
| Trương Hoàng Nguyên | B22DCCN597 | - Tóm tắt văn bản, chỉnh sửa tóm tắt  - Tìm kiếm văn bản tóm tắt |
| Nguyễn Phong Nhã | B22DCCN573 | - Quản lý chức năng admin |

# Chương II: Thiết kế hệ thống theo cá nhân

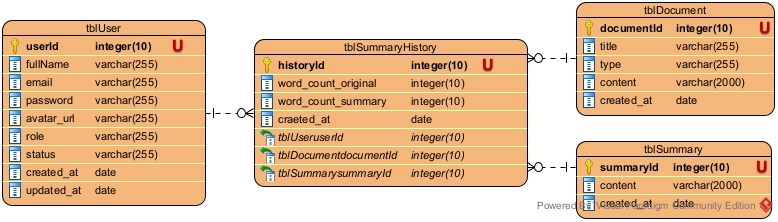
## **1. Phạm Thị Hương Nhài - B22DCCN574**

### ***1.1. Thiết kế cơ sở dữ liệu liên quan đến module***

#### 1.1.1. Module quản lý thông tin người dùng

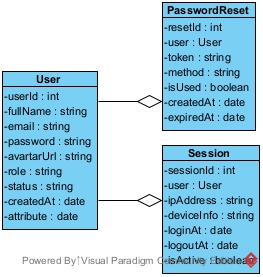


#### 1.1.2. Module lịch sử tóm tắt

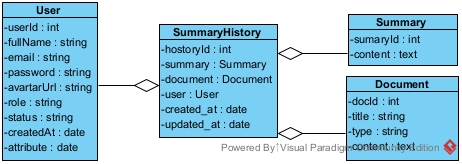


### ***1.2. Thiết kế lớp thực thể liên quan đến module***

#### 1.2.1. Module quản lý thông tin người dùng



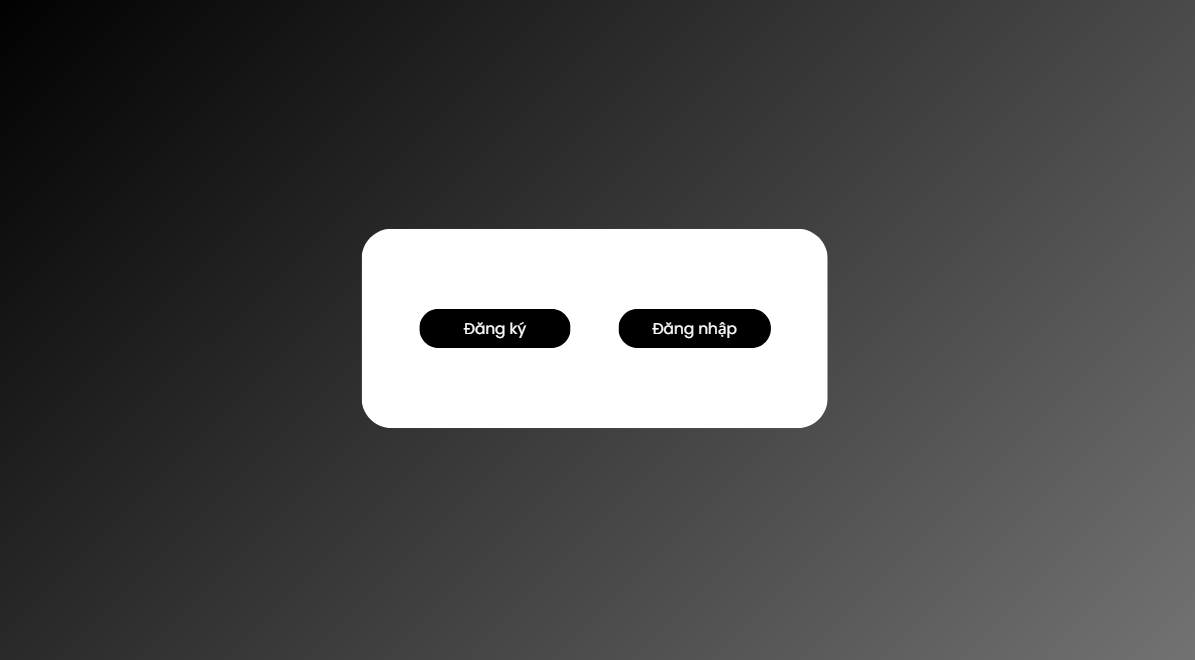
#### 1.2.2. Module lịch sử tóm tắt



### ***1.3. Thiết kế chi tiết các chức năng***

#### *1.3.1. Chức năng đăng ký tài khoản mới:*

##### Thiết kế giao diện

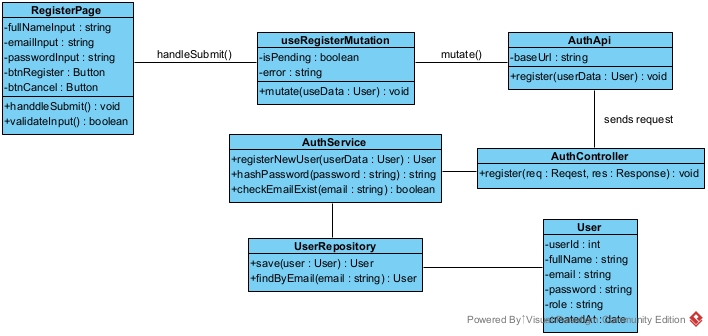


Chọn “Đăng ký” hệ thống hiển thị giao diện đăng ký:



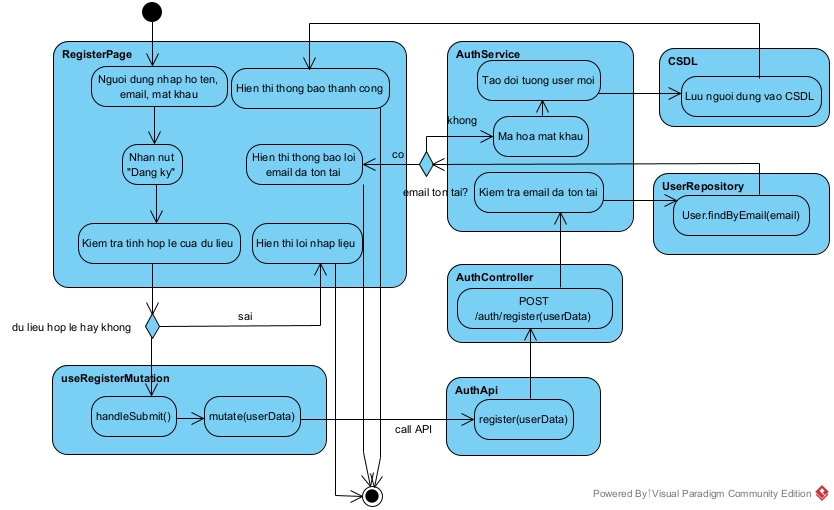
Điền đầy đủ thông tin và chọn nút “Đăng ký” để hoàn thành đăng ký tài khoản.

##### Thiết kế biểu đồ lớp chi tiết

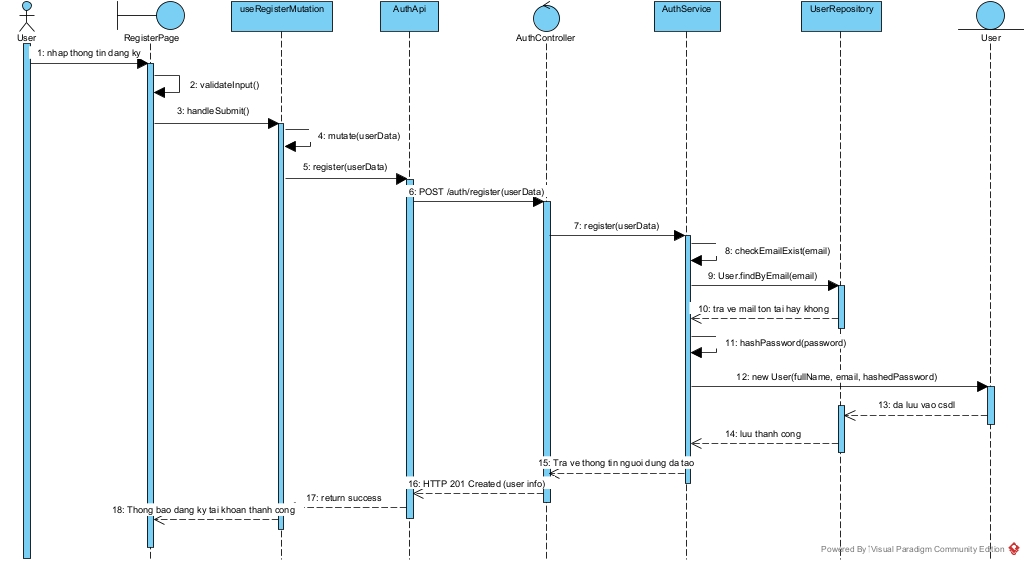


* Lớp RegisterPage (Client): Giao diện đăng ký tài khoản người dùng, cho phép nhập thông tin và gửi yêu cầu đến server.
  + Thuộc tính:
    - fullNameInput, emailInput, passwordInput: giá trị nhập liệu từ form.
    - btnRegister, btnCancel: nút xử lý hành động.
  + Phương thức: handleSubmit():
    - gom dữ liệu từ form, kiểm tra hợp lệ, gọi useRegisterMutation.
    - validateInput(): kiểm tra ràng buộc (email hợp lệ, mật khẩu đủ mạnh, không rỗng...).
* Lớp useRegisterMutation (Client): Trung gian giữa giao diện và API, quản lý trạng thái gọi API (đang chờ, lỗi, thành công...).
  + Thuộc tính:
    - isPending: trạng thái đang gửi request.
    - error: lưu lỗi trả về nếu có.
  + Phương thức:
    - mutate(userData): gọi AuthApi.register() để gửi thông tin lên server, xử lý kết quả.
* Lớp AuthApi (Client): Thực hiện các HTTP request đến server.
  + Thuộc tính:
    - baseUrl: địa chỉ endpoint của server.
  + Phương thức: register(userData):
    - gọi API /auth/register với dữ liệu đăng ký người dùng.
* Lớp AuthController (Server): Xử lý request từ client, tách biệt logic xử lý của tầng service và HTTP.
  + Phương thức:
    - register(req, res): nhận dữ liệu từ client, gọi AuthService, trả kết quả HTTP phù hợp.
* Lớp AuthService (Server): Chứa logic nghiệp vụ chính liên quan đến đăng ký người dùng.
  + Phương thức:registerNewUser(userData):
    - kiểm tra email, mã hóa mật khẩu, lưu thông tin.
    - hashPassword(password): mã hóa mật khẩu bằng thuật toán bảo mật (bcrypt, v.v...).
    - checkEmailExist(email): kiểm tra xem email đã tồn tại trong hệ thống chưa.
* Lớp UserRepository (Server): Làm việc trực tiếp với cơ sở dữ liệu để lưu hoặc truy vấn người dùng.
  + Phương thức:
    - save(user): lưu thông tin người dùng vào bảng.
    - User.findByEmail(email): kiểm tra xem email đã tồn tại.
* Lớp User (Entity): Đại diện cho thực thể người dùng trong hệ thống.
  + Thuộc tính:
    - userId: khóa chính.
    - fullName, email, password, role, createdAt: các thông tin cần thiết để xác định và quản lý người dùng.

##### Thiết kế biểu đồ hoạt động



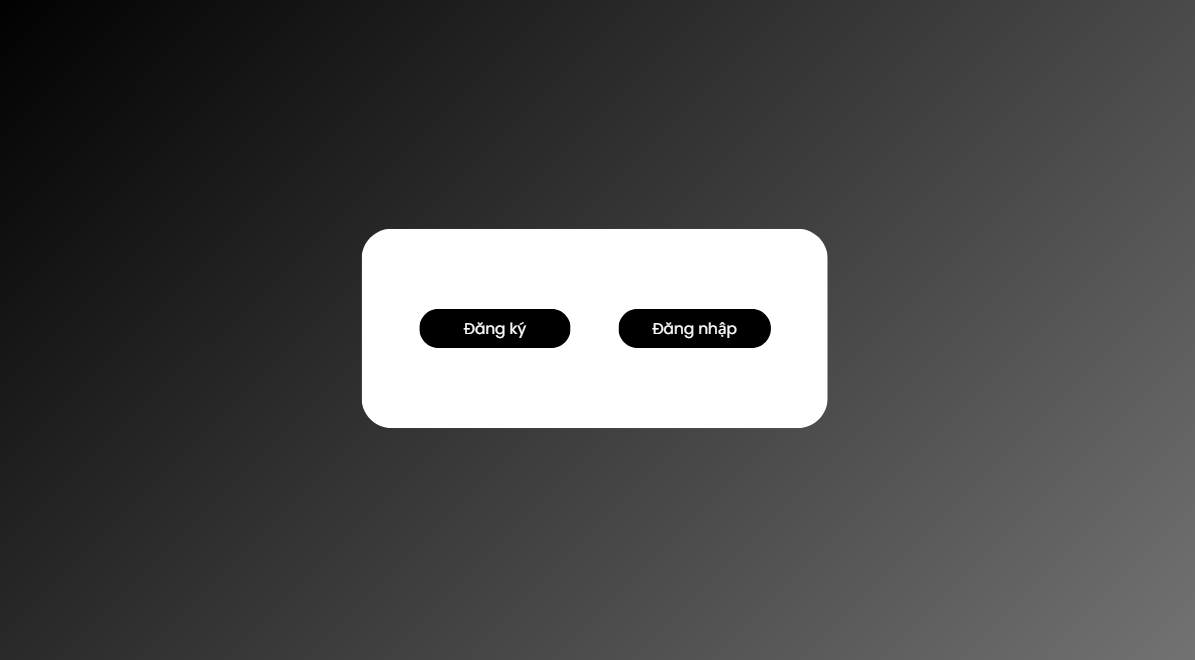
##### Thiết kế biểu đồ tuần tự



| Ký hiệu | Diễn giải chi tiết |
| --- | --- |
| 1. User -> RegisterPage : Nhập thông tin và nhấn "Đăng ký" | Người dùng thao tác trên giao diện đăng ký, nhập họ tên, email và mật khẩu, sau đó nhấn nút “Đăng ký”. |
| 2. RegisterPage -> RegisterPage : validateInput() | Giao diện RegisterPage kiểm tra tính hợp lệ của dữ liệu đầu vào (định dạng email, mật khẩu không rỗng, họ tên hợp lệ...). |
| 3. RegisterPage -> useRegisterMutation : handleSubmit(userData) | Giao diện gọi hàm gửi dữ liệu đăng ký sang hook useRegisterMutation để thực hiện logic đăng ký (handleSubmit()). |
| 4. useRegisterMutation mutate(userData) | Hook useRegisterMutation gọi hàm mutate(). |
| 5. useRegisterMutation -> AuthApi : register(userData) | Hook useRegisterMutation gọi hàm register(userData) để gửi dữ liệu đăng ký tới lớp AuthApi. |
| 6. AuthApi -> AuthController : POST /auth/register(userData) | AuthApi thực hiện yêu cầu HTTP POST đến API /auth/register trên server. |
| 7. AuthController -> AuthService : register(userData) | Controller tiếp nhận yêu cầu đăng ký, gọi đến service AuthService để xử lý nghiệp vụ. |
| 8. AuthService checkEmailExist(email) | AuthService gọi hàm checkEmailExist(email) để kiểm tra email người dùng. |
| 9. AuthService -> UserRepository : findByEmail(email) | Service truy vấn cơ sở dữ liệu để kiểm tra email người dùng có tồn tại hay chưa. |
| 10. UserRepository --> AuthService : return result | Repository trả về kết quả kiểm tra (email tồn tại hoặc chưa). |
| 11. AuthService -> AuthService : hashPassword(password) | Mã hóa mật khẩu người dùng trước khi lưu vào cơ sở dữ liệu. |
| 12. AuthService -> User : new User(fullName, email, hashedPassword) | Tạo đối tượng User mới với thông tin người dùng và mật khẩu đã mã hóa. Lưu đối tượng người dùng mới vào cơ sở dữ liệu. |
| 13. User -> Repository: return success | Phản hồi kết quả lưu thành công |
| 14. Repository--> AuthService : return success | Repository phản hồi kết quả lưu thành công. |
| 15. AuthService --> AuthController : return createdUser | Trả về thông tin người dùng mới vừa được tạo. |
| 16. AuthController --> AuthApi : HTTP 201 Created | Controller gửi phản hồi thành công về phía client. |
| 17. AuthApi --> useRegisterMutation : return success | Client nhận phản hồi thành công từ API. |
| 18. useRegisterMutation --> RegisterPage : show success message | Giao diện hiển thị đang kí thành công.. |

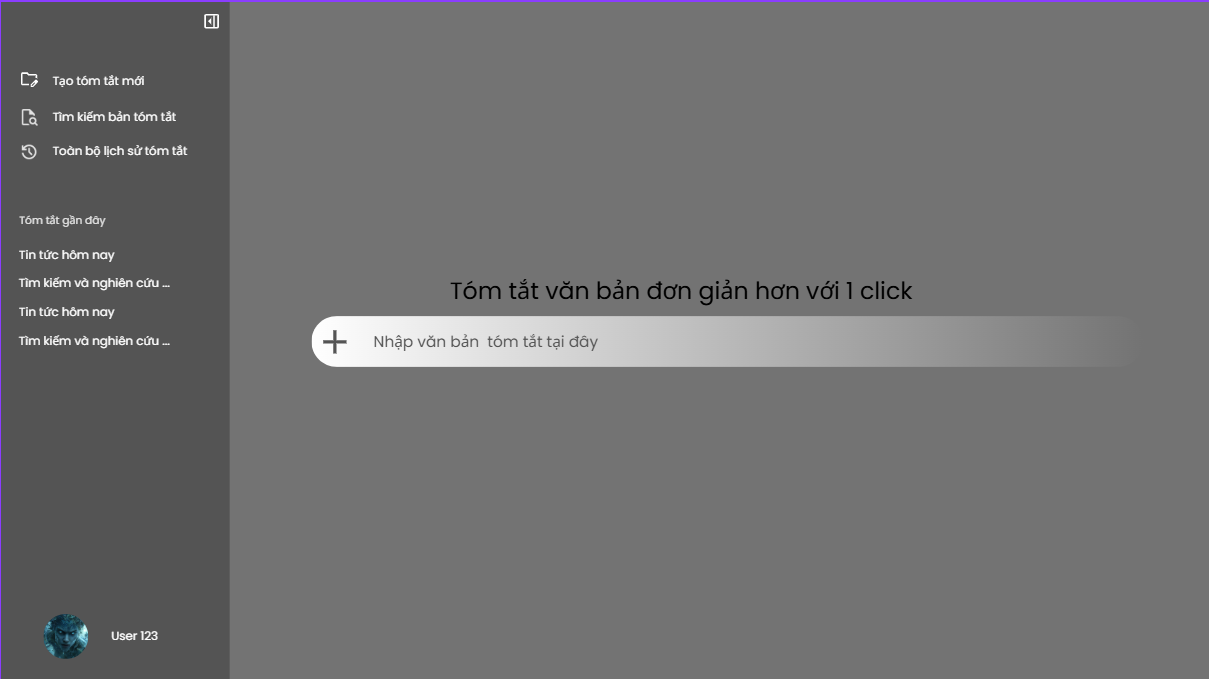
#### *1.3.2. Chức năng đăng nhập/ đăng xuất:*

##### Thiết kế giao diện

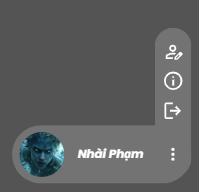
Chọn “Đăng nhập” để đăng nhập tài khoản cá nhân của người dùng, màn hình hiển thị giao diện “Đăng nhập”:



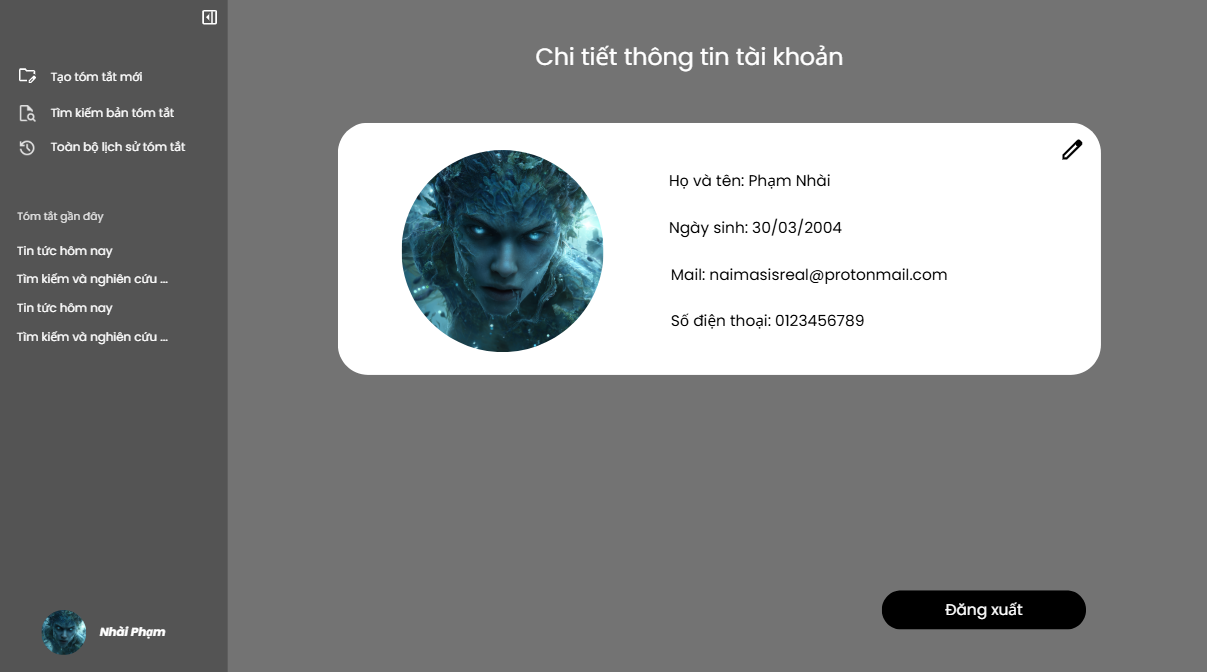
Điền chính xác thông tin đăng nhập sau đó chọn “Đăng nhập” hệ thống chuyển hướng đến trang chủ của người dùng:



Để kết thúc đăng xuất tài khoản di con trỏ đến avatar người dùng hệ thống hiển thị:

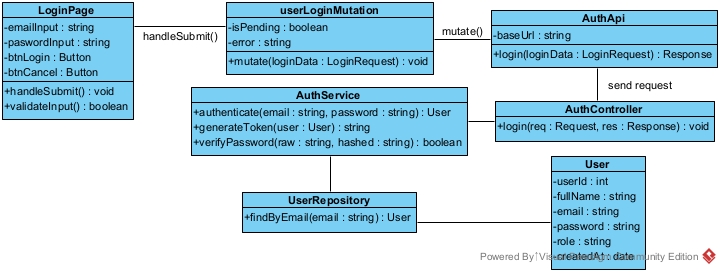


Chọn nút để đăng xuất.

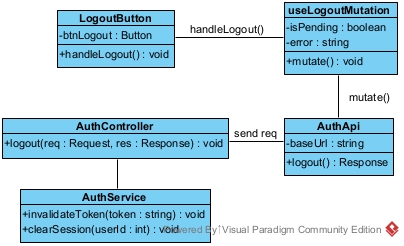
Ngoài ra còn cách đăng xuất khác: click vào tên người dùng hoặc avatar hệ thống hiển thị giao diện:

Chọn nút “Đăng xuất” để đăng xuất tài khoản.

##### Thiết kế biểu đồ lớp chi tiết

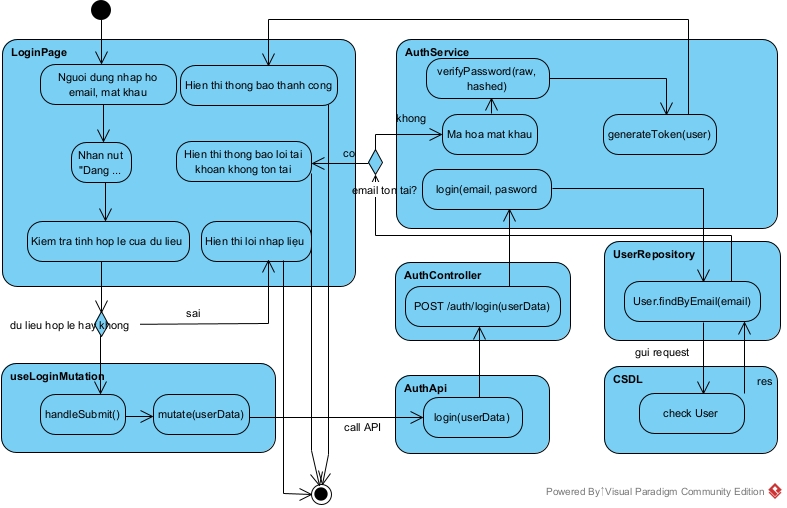


* Lớp LoginPage (Client): Giao diện đăng nhập, nhận email & mật khẩu.
  + Thuộc tính: emailInput, passwordInput, btnLogin, btnCancel
  + Phương thức:
    - handleSubmit(): thu thập thông tin, gọi useLoginMutation.
    - validateInput(): kiểm tra hợp lệ đầu vào.
* Lớp useLoginMutation (Client): Quản lý trạng thái và xử lý gọi API đăng nhập.
  + Thuộc tính: isPending, error
  + Phương thức:
    - mutate(loginData): gọi AuthApi.login(), xử lý kết quả và lỗi.
* Lớp AuthApi (Client): Thực hiện request HTTP đến server.
  + Phương thức: login(loginData): gửi dữ liệu đăng nhập đến /auth/login.
* Lớp AuthController (Server): Nhận request đăng nhập, gọi AuthService.
  + Phương thức: login(req, res): kiểm tra dữ liệu, gọi service, trả response (token, thông tin user).
* Lớp AuthService (Server): Xử lý logic xác thực.
  + Phương thức:
    - authenticate(email, password): xác thực thông tin người dùng.
    - generateToken(user): tạo JWT token hoặc session.
    - verifyPassword(raw, hashed): so sánh mật khẩu.
* Lớp UserRepository (Server): Truy xuất người dùng theo email.
  + Phương thức: findByEmail(email): tìm kiếm user trong DB.
* Lớp User (Entity): Đại diện cho thực thể người dùng.

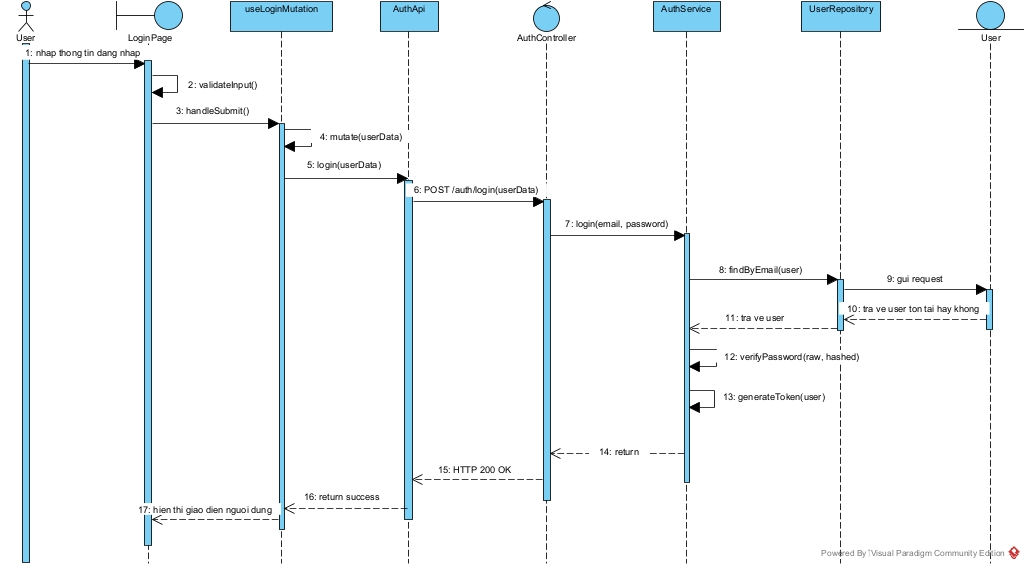


* Lớp LogoutButton (Client): Giao diện nút đăng xuất.
  + Phương thức:
    - handleLogout(): gọi useLogoutMutation, xóa local token, điều hướng về trang login.
* Lớp useLogoutMutation (Client): Trung gian xử lý việc gọi API đăng xuất, quản lý trạng thái.
  + Thuộc tính:
    - isPending, error
  + Phương thức:
    - mutate(): gọi AuthApi.logout().
* Lớp AuthApi (Client)
  + Phương thức:
    - logout(): gửi request /auth/logout để hủy token hoặc session.
* Lớp AuthController (Server)
  + Phương thức:
    - logout(req, res): nhận token từ client, gọi service để vô hiệu hóa.
* Lớp AuthService (Server)
  + Phương thức:
    - invalidateToken(token): xóa hoặc đánh dấu token không hợp lệ.
    - clearSession(userId): xóa session trong server cache hoặc DB.

##### Thiết kế biểu đồ hoạt động



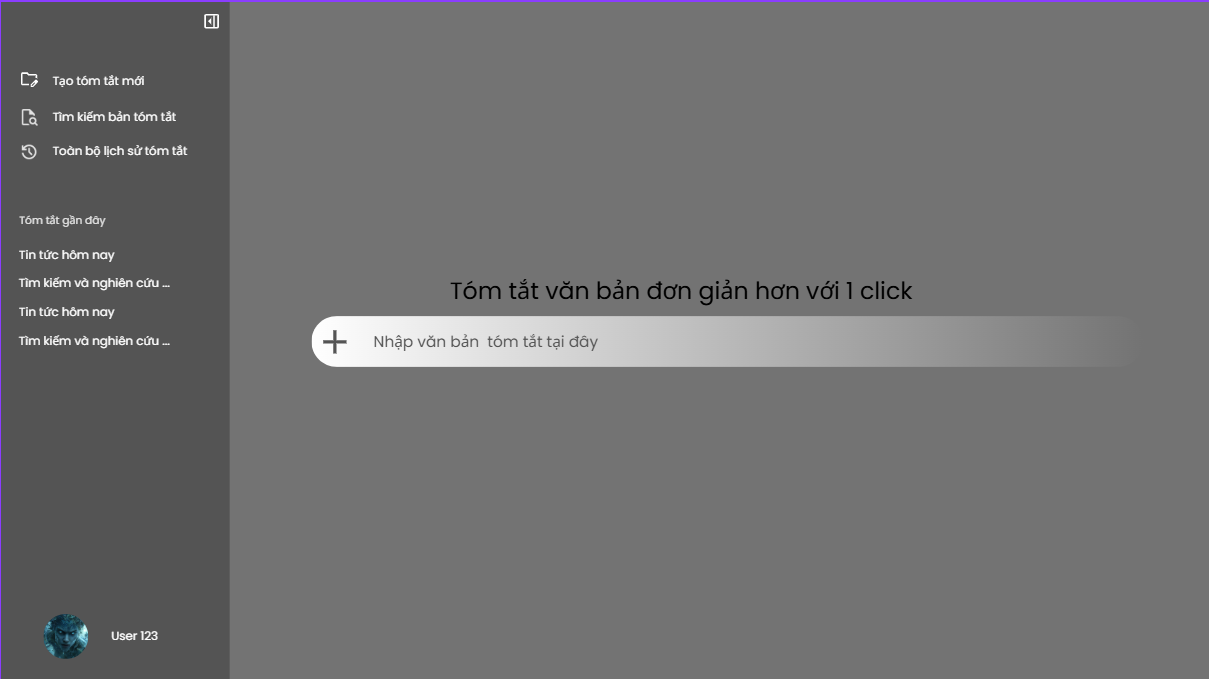
##### Thiết kế biểu đồ tuần tự

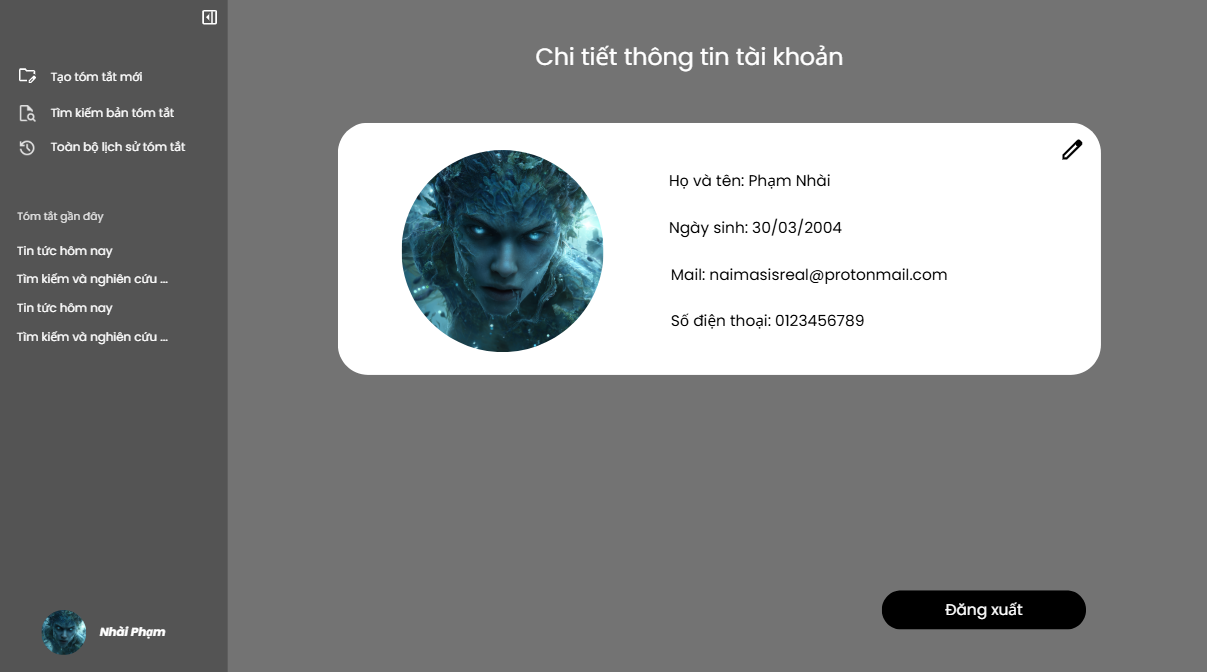


| Ký hiệu | Diễn giải chi tiết |
| --- | --- |
| 1. User -> LoginPage : Nhập email & mật khẩu, nhấn "Đăng nhập" | Người dùng thao tác trên giao diện đăng nhập, nhập email và mật khẩu, sau đó nhấn nút “Đăng nhập”. |
| 2. LoginPage -> LoginPage : validateInput() | Giao diện LoginPage thực hiện kiểm tra hợp lệ dữ liệu đầu vào (định dạng email, mật khẩu không rỗng…). |
| 3. LoginPage -> useLoginMutation : handleSubmit(userData) | Gọi hàm xử lý gửi dữ liệu đăng nhập sang hook useLoginMutation. |
| 4. useLoginMutation: mutate(userData) | userLoginMutation gọi hàm mutate(userData) |
| 5. useLoginMutation -> AuthApi: login(userData) | Hook trung gian này gọi đến lớp AuthApi để thực hiện yêu cầu HTTP. |
| 6. AuthApi -> AuthController : POST /auth/login(userData) | Gửi yêu cầu đăng nhập đến API /auth/login trên server. |
| 7. AuthController -> AuthService : login(email, password) | Tầng Controller tiếp nhận request, gọi đến AuthService để xử lý nghiệp vụ xác thực. |
| 8. AuthService -> UserRepository : findByEmail(email) | Service truy vấn cơ sở dữ liệu để tìm thông tin người dùng theo email. |
| 9. UserRepository -> User | Lớp UserRepository gửi request trung gian lấy dữ liệu từ CSDL. |
| 10. User -> UserRepository | Phản hồi tồn tại tài khoản người dùng hay không. |
| 11. UserRepository --> AuthService | Repository trả về thông tin người dùng tìm thấy. |
| 12. AuthService -> AuthService : verifyPassword(raw, hashed) | So sánh mật khẩu người dùng nhập với mật khẩu đã mã hóa trong cơ sở dữ liệu. |
| 13. AuthService: generateToken(user) | Tạo mã token (JWT hoặc session) cho người dùng. |
| 14. AuthService --> AuthController : return | Trả kết quả xác thực thành công gồm token và thông tin người dùng. |
| 15. AuthController --> AuthApi : HTTP 200 OK | Controller trả phản hồi HTTP thành công về phía client. |
| 16. AuthApi --> useLoginMutation : return success | API client nhận kết quả thành công. |
| 17. useLoginMutation --> LoginPage : show dashboard | Hiển thị giao diện người dùng. |

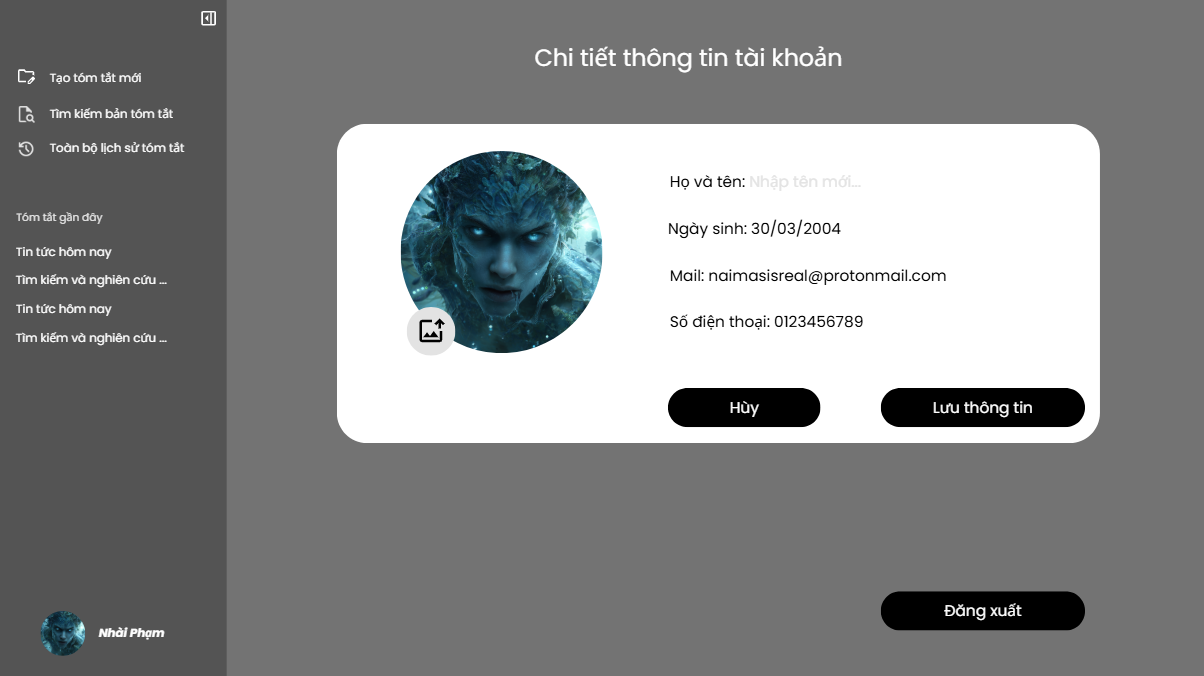
#### 1.3.3. Chức năng chỉnh sửa thông tin cá nhân:

##### Thiết kế giao diện

Sau khi đăng nhập thành công từ trang chủ click vào tên người dùng hoặc avatar hệ thống hiển thị giao diện:

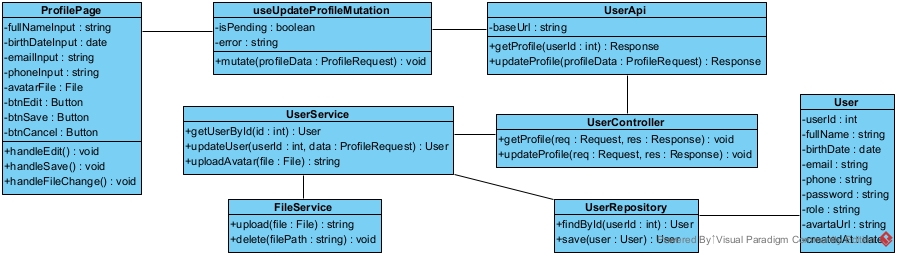


Chọn biểu tượng để chỉnh sửa thông tin cá nhân:



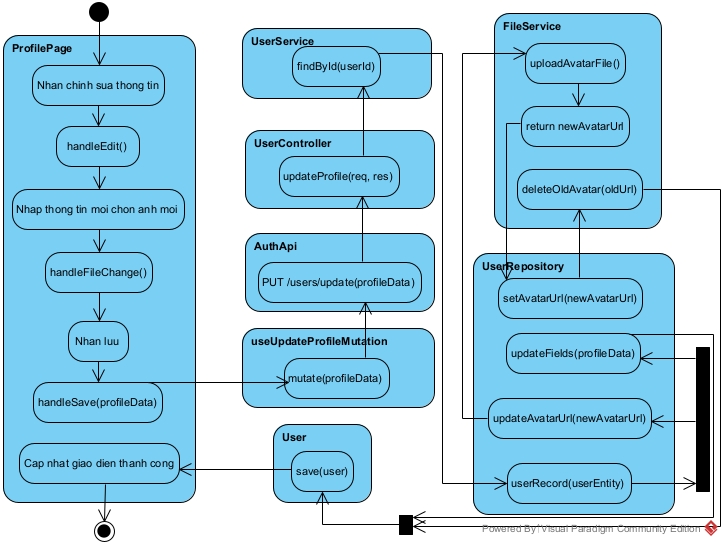
Chỉnh sửa thông tin cá nhân, upload ảnh đại diện mới sau đó chọn “Lưu thông tin” đã hoàn thành chỉnh sửa.

##### Thiết kế biểu đồ lớp chi tiết

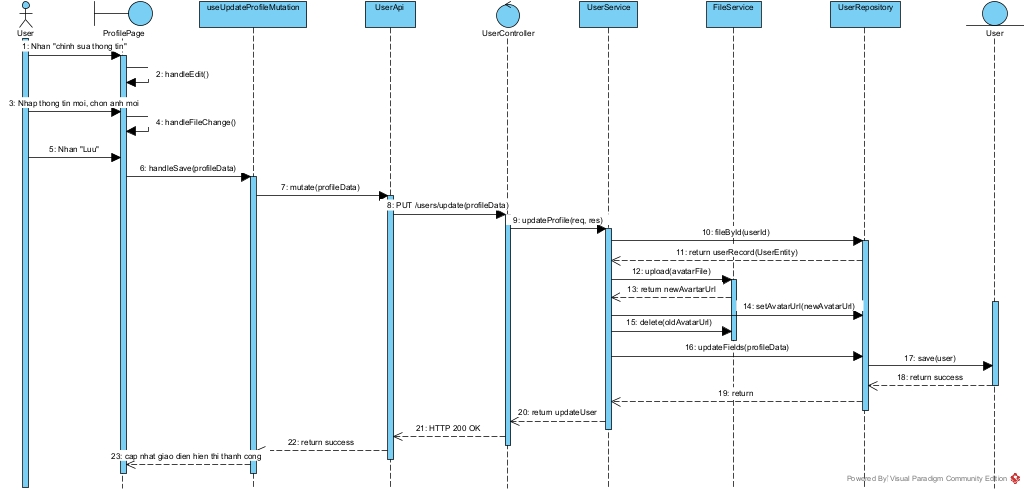


* Lớp ProfilePage (Client): Giao diện người dùng hiển thị và cho phép chỉnh sửa thông tin cá nhân.
  + Thuộc tính:
    - fullNameInput, birthDateInput, emailInput, phoneInput, avatarFile, btnEdit, btnSave, btnCancel
  + Phương thức:
    - handleEdit(): chuyển form sang chế độ chỉnh sửa.
    - handleFileChange(): xử lý khi người dùng chọn ảnh đại diện mới.
    - handleSave(): thu thập dữ liệu và gọi mutation để cập nhật thông tin.
* Lớp useUpdateProfileMutation (Client): Quản lý logic gọi API cập nhật thông tin người dùng.
  + Thuộc tính:
    - isPending, error
  + Phương thức:
    - mutate(profileData): gửi dữ liệu đến UserApi.updateProfile() và theo dõi trạng thái thành công/thất bại.
* Lớp UserApi (Client): Cầu nối HTTP giữa frontend và backend.
  + Phương thức:
    - getProfile(userId): lấy thông tin cá nhân hiện tại.
    - updateProfile(profileData): gửi request cập nhật thông tin.
* Lớp UserController (Server): Nhận request từ client, gọi UserService để xử lý logic.
  + Phương thức:
    - getProfile(req, res): trả về thông tin người dùng.
    - updateProfile(req, res): kiểm tra dữ liệu, gọi service cập nhật, và trả response.
* Lớp UserService (Server): Chứa toàn bộ nghiệp vụ cập nhật thông tin người dùng.
  + Phương thức:
    - getUserById(id): lấy thông tin người dùng theo ID.
    - updateUser(userId, data): kiểm tra hợp lệ và lưu thông tin mới.
    - uploadAvatar(file): xử lý upload ảnh đại diện mới.
* Lớp FileService (Server): Xử lý upload, xóa hoặc quản lý file avatar.
  + Phương thức:
    - upload(file): tải ảnh lên server.
    - delete(filePath): xóa file cũ (nếu người dùng thay ảnh).
* Lớp UserRepository (Server): Truy xuất và lưu thông tin người dùng vào cơ sở dữ liệu.
  + Phương thức:
    - findById(userId): tìm người dùng trong DB.
    - save(user): cập nhật thông tin mới và lưu vào DB.
* Lớp User (Entity): Đại diện cho thực thể người dùng trong hệ thống.
  + Thuộc tính chính:
    - userId, fullName, birthDate, email, phone, avatarUrl, role, createdAt

##### Thiết kế biểu đồ hoạt động



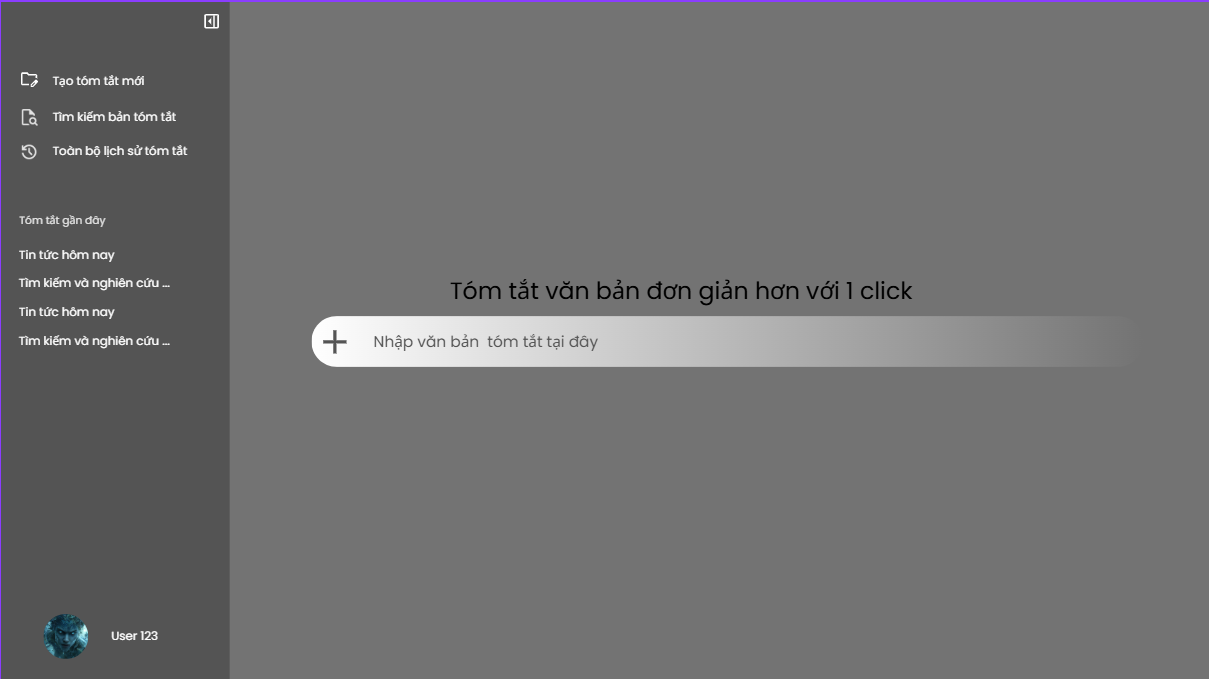
##### Thiết kế biểu đồ tuần tự

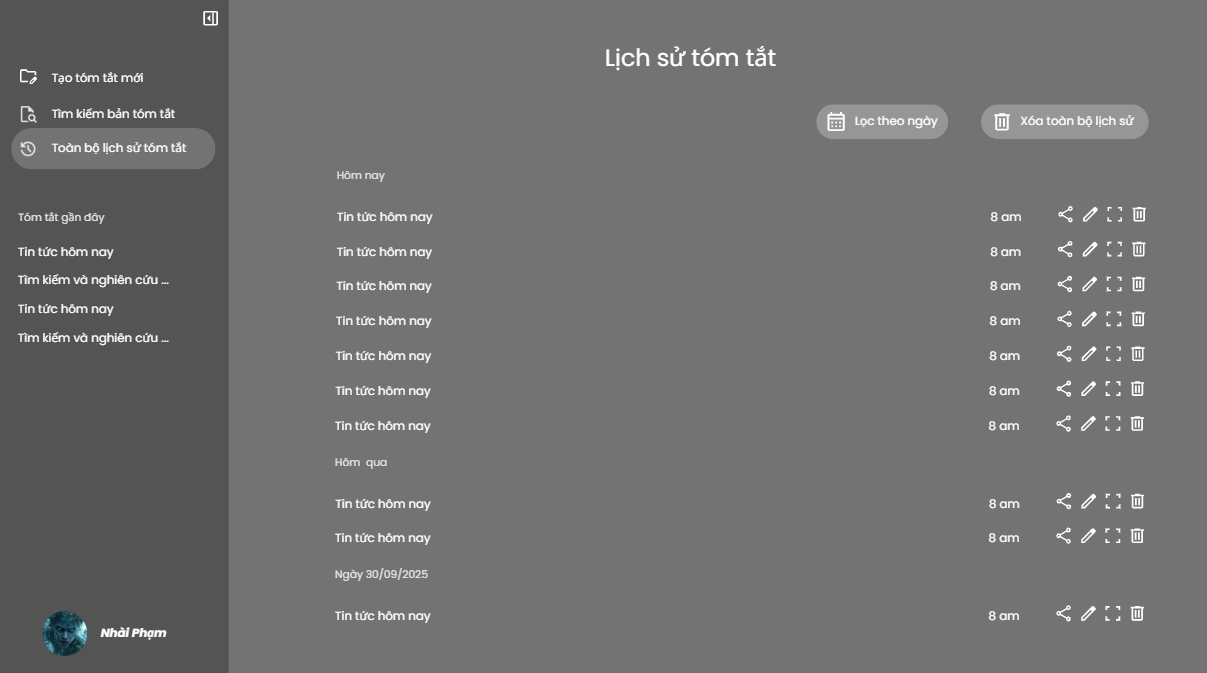


| Ký hiệu | Diễn giải chi tiết |
| --- | --- |
| 1. User -> ProfilePage : Nhấn "Chỉnh sửa thông tin" | Người dùng thao tác để chuyển form sang chế độ chỉnh sửa. |
| 2. ProfilePage -> ProfilePage : handleEdit() | Giao diện bật trạng thái cho phép nhập liệu. |
| 3. User -> ProfilePage : Nhập thông tin mới / chọn ảnh mới | Người dùng thay đổi thông tin cá nhân hoặc chọn ảnh đại diện mới. |
| 4. ProfilePage -> ProfilePage : handleFileChange() | Lưu file ảnh tạm thời và hiển thị bản xem trước ảnh đại diện. |
| 5. User -> ProfilePage : Nhấn "Lưu" | Người dùng gửi yêu cầu cập nhật thông tin cá nhân. |
| 6. ProfilePage -> useUpdateProfileMutation : handleSave(profileData) | Gọi hàm xử lý lưu, thu thập dữ liệu và truyền vào hook mutation. |
| 7. useUpdateProfileMutation -> UserApi : mutate(profileData) | Hook mutation gửi dữ liệu đến tầng API để thực hiện request. |
| 8. UserApi -> UserController : PUT /users/update(profileData) | API client gửi yêu cầu HTTP cập nhật thông tin đến server. |
| 9. UserController -> UserService : updateProfile(req, res) | Controller gọi tầng Service để xử lý logic cập nhật hồ sơ. |
| 10. UserService -> UserRepository : findById(userId) | Truy vấn cơ sở dữ liệu để lấy thông tin người dùng hiện tại. |
| 11. UserRepository --> UserService : return userRecord (UserEntity) | Repository trả về một đối tượng UserEntity chứa dữ liệu người dùng. |
| 12. UserService -> FileService : upload(avatarFile) | Gọi FileService để tải ảnh đại diện mới lên máy chủ. |
| 13. FileService --> UserService : return newAvatarUrl | Trả về đường dẫn ảnh đại diện mới sau khi upload thành công. |
| 14. UserService -> UserEntity : setAvatarUrl(newAvatarUrl) | Cập nhật lại trường avatarUrl của entity User. |
| 15. UserService -> FileService : delete(oldAvatarPath) | Xóa ảnh đại diện cũ khỏi hệ thống (nếu có). |
| 16. UserService -> UserRepository: updateFields(profileData) | Service gọi phương thức của entity để cập nhật các thuộc tính cơ bản như họ tên, ngày sinh, email, số điện thoại. |
| 17. UserRepository-> UserEntity: save(UserEntity) | Lưu thực thể UserEntity đã được chỉnh sửa xuống cơ sở dữ liệu. |
| 18.UserEntity → UserRepository: return success | Phản hồi thành công. |
| 19. UserRepository --> UserService : return success | Repository phản hồi lưu thành công. |
| 20. UserService --> UserController : return updatedUser | Service trả về đối tượng người dùng mới đã được cập nhật. |
| 21. UserController --> UserApi : HTTP 200 OK (updatedUser) | Controller phản hồi về client với dữ liệu người dùng mới. |
| 22. UserApi --> useUpdateProfileMutation : return success | API client nhận kết quả thành công từ server. |
| 23. useUpdateProfileMutation --> ProfilePage : update UI + show success message | Giao diện hiển thị thông tin người dùng mới và thông báo “Cập nhật thành công”. |

#### 1.3.4. Chức năng xem danh sách lịch sử tóm tắt:

##### Thiết kế giao diện

Từ giao diện chính sau khi đăng nhập chọn để xem toàn bộ lịch sử tóm tắt của cá nhân:

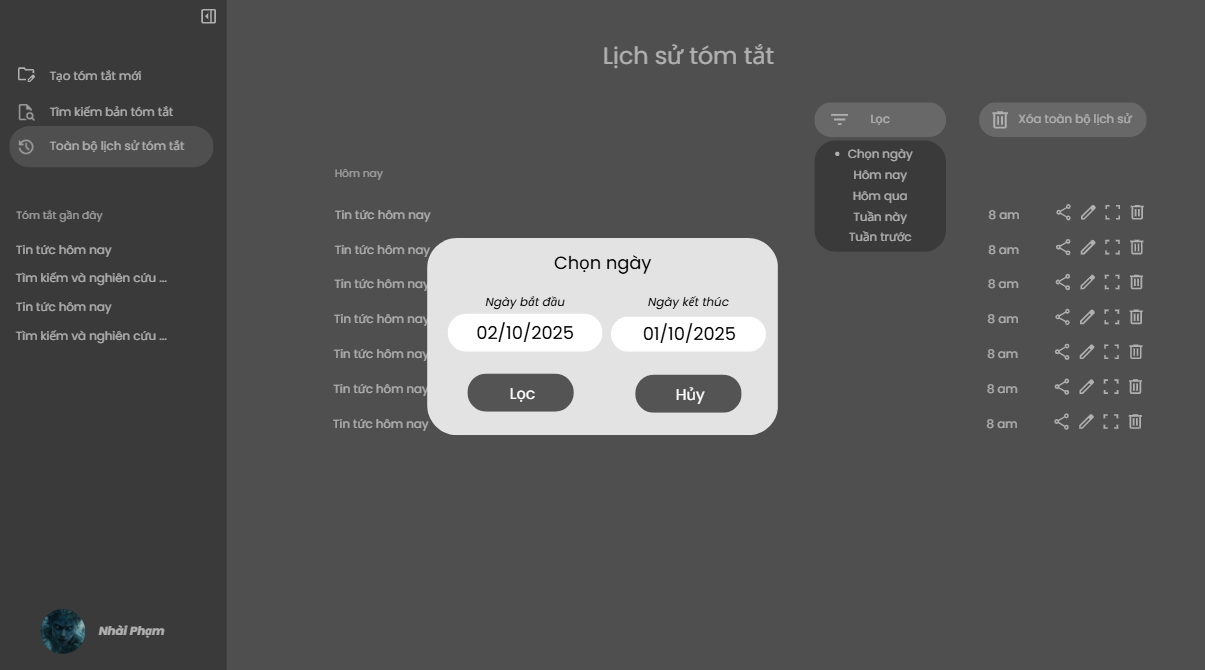


Lịch sử tóm tắt hiển thị theo thời gian (gần nhất rồi đến xa nhất theo danh sách). Để hiển thị lịch sử theo thời gian cụ thể (ngày bắt đầu, ngày kết thúc) chọn  hệ thống hiển thị các mặc định chọn như:



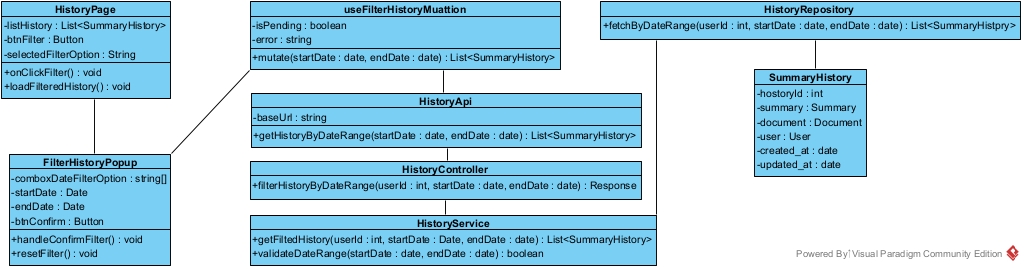
Click chọn mục cụ thể (chọn ngày là “Hôm nay”) giao diện hiển thị danh sách lịch sử theo ngày như sau:  


Ngoài ra nếu chọn khoảng ngày khác có thể chọn mục giao diện sẽ hiển thị:



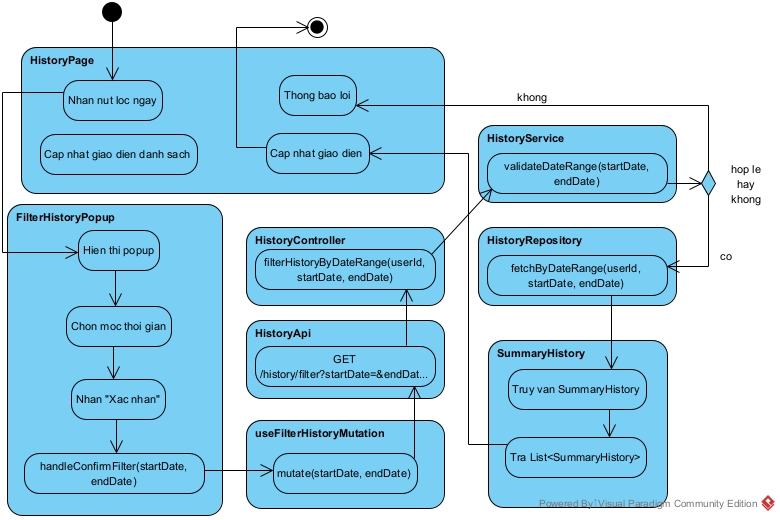
Người dùng chọn ngày bắt đầu và kết thúc sau đó lọc để xem lịch sử theo khoảng chọn.

##### Thiết kế biểu đồ lớp chi tiết

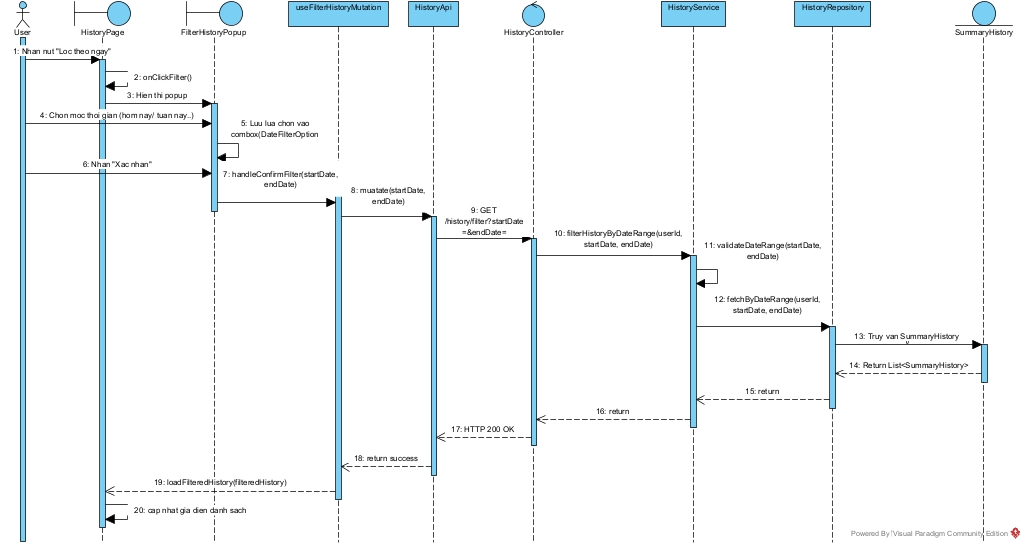


* Lớp HistoryPage (Client): Đây là lớp giao diện chính hiển thị danh sách lịch sử tóm tắt cho người dùng. Người dùng có thể xem toàn bộ lịch sử tóm tắt hoặc sử dụng bộ lọc để hiển thị danh sách theo thời gian cụ thể.
  + Các thuộc tính:
    - listHistory: danh sách các bản tóm tắt hiển thị.
    - btnFilter: nút bấm để mở popup lọc lịch sử.
    - selectedFilterOption: giá trị lựa chọn bộ lọc hiện tại (ví dụ: “Hôm nay”, “Tuần này”, hoặc “Tùy chỉnh”).
  + Các phương thức:
    - onClickFilter(): được gọi khi người dùng nhấn nút Lọc theo ngày, hiển thị popup FilterHistoryPopup.
    - loadFilteredHistory(): thực hiện tải danh sách lịch sử sau khi lọc bằng cách gọi useFilterHistoryMutation.
* Lớp FilterHistoryPopup (Client): Đây là lớp giao diện phụ được hiển thị khi người dùng chọn chức năng Lọc theo ngày cụ thể. Cho phép người dùng chọn mốc thời gian (Hôm nay, Hôm qua, Tuần này, Tuần trước, hoặc Khoảng ngày tùy chọn).
  + Các thuộc tính:
    - comboxDateFilterOption: danh sách lựa chọn bộ lọc thời gian.
    - startDate: ngày bắt đầu (khi chọn “Chọn ngày”).
    - endDate: ngày kết thúc (khi chọn “Chọn ngày”).
    - btnConfirm: nút xác nhận lọc lịch sử.
  + Các phương thức:
    - handleConfirmFilter(): thu thập thông tin người dùng chọn và gọi useFilterHistoryMutation để truy vấn dữ liệu từ server.
    - resetFilter(): đặt lại bộ lọc về mặc định (toàn bộ lịch sử).
* Lớp useFilterHistoryMutation (Client): Là lớp trung gian giữa lớp giao diện và lớp gọi API, được dùng để quản lý các state bất đồng bộ (pending, loading, success, error).
  + Các thuộc tính:
    - isPending: cho biết việc gọi API có đang được thực hiện hay không.
    - error: chứa thông tin lỗi nếu việc truy vấn lịch sử thất bại.
  + Các phương thức:
    - mutate(): gọi lớp HistoryApi để thực hiện truy vấn danh sách lịch sử dựa trên tham số ngày bắt đầu và ngày kết thúc.
* Lớp HistoryApi (Client): Là lớp phía Client, chịu trách nhiệm gửi HTTP request đến server để lấy danh sách lịch sử tóm tắt.
  + Các thuộc tính:
    - baseUrl: là đường dẫn đến server.
  + Các phương thức:
    - getHistoryByDateRange(startDate: Date, endDate: Date): gửi request đến server để lấy danh sách các bản tóm tắt nằm trong khoảng thời gian được chọn.
* Lớp HistoryController (Server): Là tầng điều khiển phía server, chịu trách nhiệm nhận request từ client, xác thực dữ liệu và gọi lớp HistoryService để xử lý nghiệp vụ.
  + Các phương thức:
    - filterHistoryByDateRange(userId, startDate, endDate): nhận dữ liệu từ client, xác thực thông tin, gọi service xử lý và trả kết quả về cho client.
* Lớp HistoryService (Server): Là lớp xử lý logic nghiệp vụ chính của module lịch sử tóm tắt.
  + Các phương thức:
    - getFilteredHistory(userId, startDate, endDate): xử lý nghiệp vụ lọc lịch sử, gọi lớp repository để lấy dữ liệu theo ngày.
    - validateDateRange(startDate, endDate): kiểm tra tính hợp lệ của khoảng thời gian lọc.
* Lớp HistoryRepository (Server): Là lớp quản lý các phương thức truy xuất cơ sở dữ liệu (Data Access Layer).
  + Các phương thức:
    - fetchByDateRange(userId, startDate, endDate): thực hiện câu truy vấn lấy lịch sử tóm tắt trong khoảng thời gian cụ thể từ cơ sở dữ liệu.
* Lớp SummaryHistory (Entity): Là lớp thực thể đại diện cho từng bản ghi lịch sử tóm tắt trong hệ thống.
  + Các thuộc tính:
    - historyId: định danh của bản ghi.
    - userId: mã người dùng sở hữu bản ghi.
    - originalText: nội dung gốc được tóm tắt.
    - summaryText: nội dung văn bản sau khi tóm tắt.
    - createdAt: thời gian tóm tắt được thực hiện.
    - isShared: trạng thái chia sẻ (true/false).

##### Thiết kế biểu đồ hoạt động



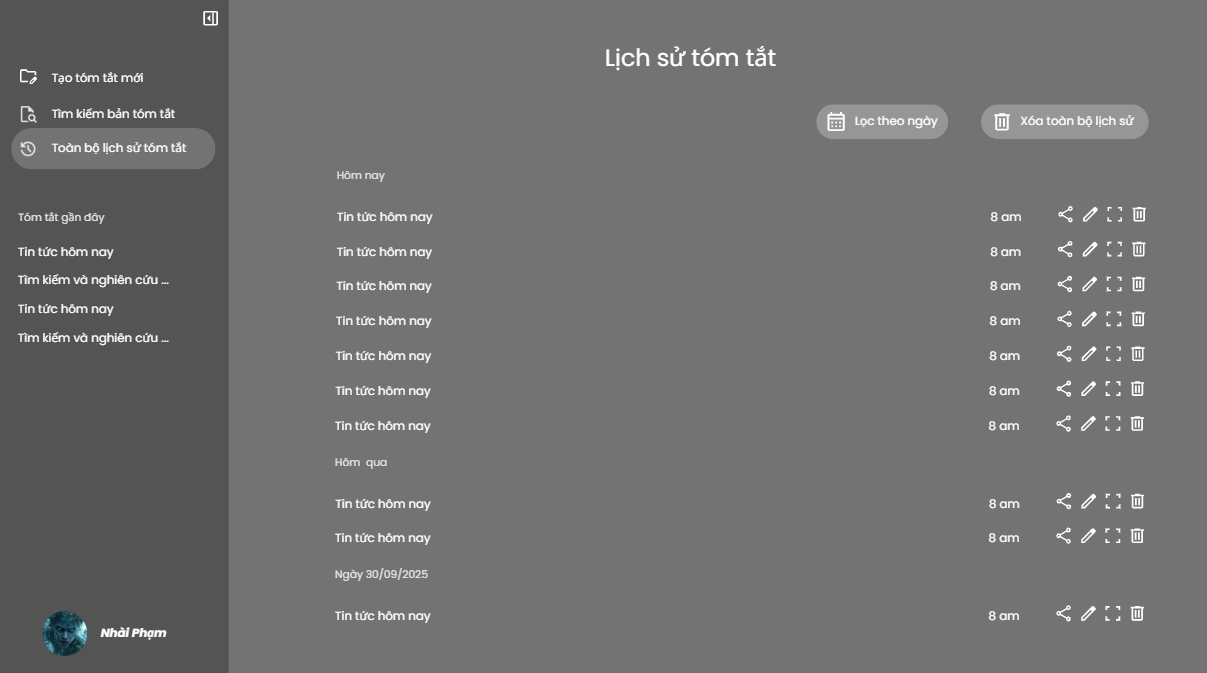
##### Thiết kế biểu đồ tuần tự



| Ký hiệu | Diễn giải chi tiết |
| --- | --- |
| 1. User -> HistoryPage : Nhấn nút "Lọc theo ngày" | Người dùng bắt đầu thao tác lọc lịch sử tóm tắt bằng cách nhấn nút lọc trên giao diện chính. |
| 2. HistoryPage -> HistoryPage : onClickFilter() | Click chọn ngày để hiển thị cửa sổ popup cho phép chọn mốc thời gian cần lọc. |
| 3. HistoryPage -> FilterHistoryPopup : Hiển thị popup | hiển thị cửa sổ popup cho phép chọn mốc thời gian cần lọc. |
| 4. User -> FilterHistoryPopup : Chọn mốc thời gian (Hôm nay / Tuần này /…) | Người dùng chọn tùy chọn thời gian mong muốn trong popup lọc. |
| 5. FilterHistoryPopup -> FilterHistoryPopup : Lưu lựa chọn vào combox(DateFilterOption) | Lưu lựa chọn đã chọn ở popup tùy chỉnh thời gian vào combox(DateFilterOption) |
| 6. User -> FilterHistoryPopup: Nhấn “Xác nhận” | Xác nhận lại thời gian để bắt đầu tiến hành lọc |
| 7. FilterHistoryPopup -> useFilterHistoryMutation : handleConfirmFilter(startDate, endDate) | Khi nhấn “Xác nhận lọc”, popup gửi dữ liệu mốc thời gian đã chọn sang mutation để xử lý. |
| 8. useFilterHistoryMutation -> HistoryApi : mutate(startDate, endDate) | Mutation gọi đến API để gửi request truy vấn dữ liệu lịch sử theo ngày. |
| 9. HistoryApi -> HistoryController : GET /history/filter?startDate=&endDate= | API gửi yêu cầu HTTP GET đến backend, chuyển tiếp tham số ngày bắt đầu và kết thúc. |
| 10. HistoryController -> HistoryService : filterHistoryByDateRange(userId, startDate, endDate) | Controller gọi tầng Service để xử lý logic lọc lịch sử theo người dùng và thời gian. |
| 11. HistoryService -> HistoryService : validateDateRange(startDate, endDate) | Service kiểm tra tính hợp lệ của khoảng thời gian lọc. |
| 12. HistoryService -> HistoryRepository : fetchByDateRange(userId, startDate, endDate) | Service yêu cầu Repository truy vấn danh sách lịch sử tóm tắt trong cơ sở dữ liệu. |
| 13. HistoryRepository -> SummaryHistory : Truy vấn SummaryHistory | Repository gửi truy vấn đến cơ sở dữ liệu, ánh xạ dữ liệu thu được thành các đối tượng thực thể SummaryHistory. |
| 14. SummaryHistory --> HistoryRepository : return danh sách đối tượng SummaryHistory | Các đối tượng SummaryHistory được khởi tạo hoàn chỉnh và trả về cho Repository. |
| 15. HistoryRepository --> HistoryService : return | Repository trả về danh sách bản ghi lịch sử đã được lọc cho Service. |
| 16. HistoryService --> HistoryController : return \ | Service chuyển danh sách lịch sử đã lọc về cho Controller. |
| 17. HistoryController --> HistoryApi : HTTP 200 OK (filteredHistory) | Controller gửi phản hồi thành công về phía client thông qua API. |
| 18. HistoryApi --> useFilterHistoryMutation : return success | API chuyển dữ liệu lịch sử đã lọc cho mutation để xử lý hiển thị. |
| 19. useFilterHistoryMutation --> HistoryPage : loadFilteredHistory(filteredHistory) | Mutation gửi danh sách lịch sử đã lọc về giao diện để hiển thị. |
| 20. HistoryPage -> HistoryPage : Cập nhật giao diện danh sách | Giao diện người dùng hiển thị danh sách lịch sử tóm tắt tương ứng với bộ lọc đã chọn. |

#### 1.3.5. Chức năng xóa lịch sử tóm tắt:

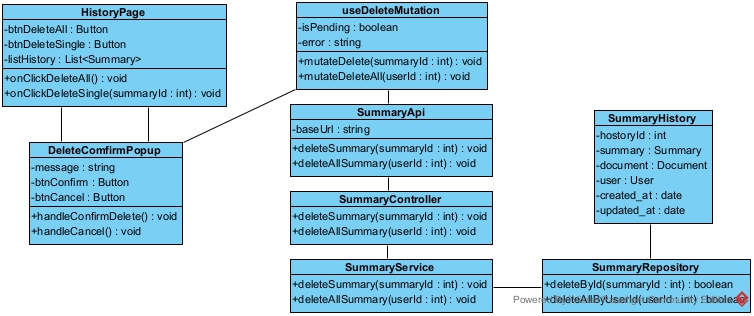
##### Thiết kế giao diện

Tại giao diện chung lịch sử tóm tắt chọn  để xóa toàn bộ lịch sử tóm tắt của cá nhân. Hoặc chọn biểu tượng  để xóa các lịch sử tóm tắt cụ thể. Giao diện sẽ hiển thị giao diện xác nhận:



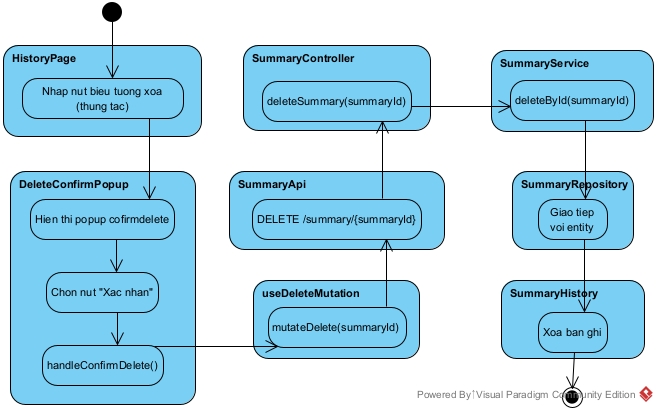
Xác nhận chọn “Xóa” để hoàn thành xóa lịch sử tóm tắt.

##### Thiết kế biểu đồ lớp chi tiết

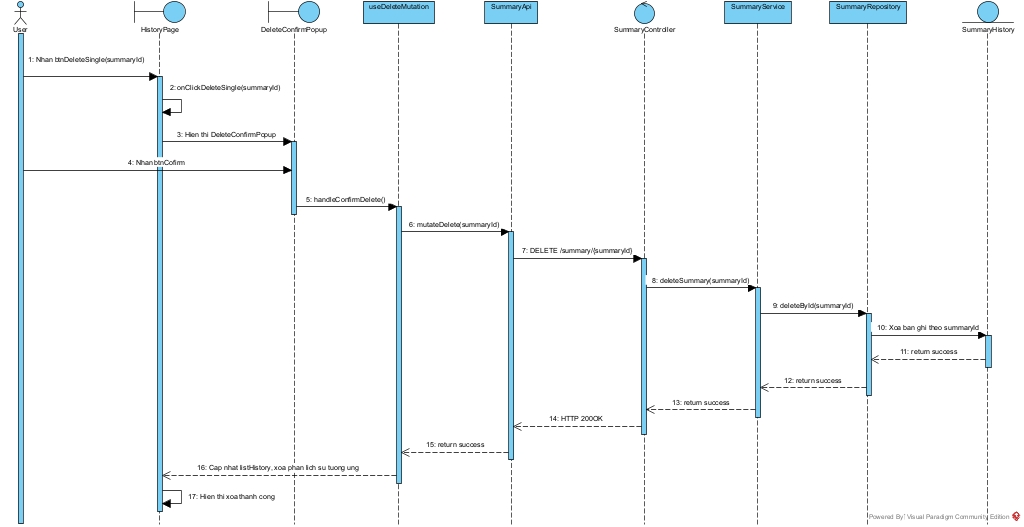


* Lớp HistoryPage (Client): Đây là lớp giao diện chính hiển thị danh sách lịch sử tóm tắt của người dùng. Khi người dùng muốn xóa lịch sử, lớp này cung cấp nút btnDeleteAll để xóa toàn bộ lịch sử, hoặc biểu tượng btnDeleteSingle để xóa từng lịch sử cụ thể.
  + Các thuộc tính:
    - btnDeleteAll: nút bấm để xóa toàn bộ lịch sử tóm tắt.
    - btnDeleteSingle: biểu tượng xóa một lịch sử tóm tắt cụ thể.
    - listHistory: danh sách các bản tóm tắt hiển thị trên giao diện.
  + Các phương thức:
    - onClickDeleteAll(): được gọi khi người dùng chọn “Xóa toàn bộ”, sẽ mở hộp thoại xác nhận.
    - onClickDeleteSingle(summaryId): được gọi khi người dùng chọn biểu tượng thùng rác, mở hộp thoại xác nhận xóa lịch sử cụ thể.
* Lớp DeleteConfirmPopup (Client): Đây là popup hiển thị hộp thoại xác nhận khi người dùng thực hiện thao tác xóa lịch sử tóm tắt.
  + Các thuộc tính:
    - message: chuỗi thông báo xác nhận xóa (“Xác nhận xóa toàn bộ lịch sử?” hoặc “Xác nhận xóa lịch sử này?”).
    - btnConfirm: nút xác nhận thao tác xóa.
    - btnCancel: nút hủy thao tác.
  + Các phương thức:
    - handleConfirmDelete(): thực hiện thao tác xóa, gọi lớp mutation để gửi yêu cầu xóa đến server.
    - handleCancel(): đóng popup xác nhận và không thực hiện thao tác xóa.
* Lớp useDeleteMutation (Client): Là lớp trung gian giữa giao diện và API, quản lý trạng thái bất đồng bộ trong quá trình gọi API xóa dữ liệu.
  + Các thuộc tính:
    - isPending: cho biết quá trình gọi API đang xử lý.
    - error: chứa thông tin lỗi nếu quá trình gọi API thất bại.
  + Các phương thức:
    - mutateDelete(summaryId): gọi lớp SummaryApi để thực hiện xóa lịch sử cụ thể.
    - mutateDeleteAll(userId): gọi lớp SummaryApi để thực hiện xóa toàn bộ lịch sử của người dùng.
* Lớp SummaryApi (Client): Là lớp chịu trách nhiệm thực hiện HTTP request đến server để thao tác với dữ liệu lịch sử tóm tắt.
  + Các thuộc tính:
    - baseUrl: đường dẫn cơ bản của API server.
  + Các phương thức:
    - deleteSummary(summaryId): gửi request DELETE đến endpoint xóa một lịch sử tóm tắt cụ thể.
    - deleteAllSummaries(userId): gửi request DELETE đến endpoint xóa toàn bộ lịch sử của người dùng.
* Lớp SummaryController (Server): Là tầng controller trên server, tiếp nhận các request xóa từ phía client, kiểm tra dữ liệu và gọi tầng service để xử lý.
  + Các phương thức:
    - deleteSummary(summaryId): tiếp nhận yêu cầu xóa lịch sử cụ thể, kiểm tra hợp lệ và gọi SummaryService.deleteSummary(summaryId).
    - deleteAllSummaries(userId): tiếp nhận yêu cầu xóa toàn bộ lịch sử của người dùng.
* Lớp SummaryService (Server): Chứa logic xử lý chính của nghiệp vụ “xóa lịch sử tóm tắt”.
  + Các phương thức:
    - deleteSummary(summaryId): gọi SummaryRepository để xóa một lịch sử cụ thể, đồng thời cập nhật log hoặc thông báo cho người dùng.
    - deleteAllSummaries(userId): gọi repository để xóa toàn bộ lịch sử tóm tắt của người dùng.
* Lớp SummaryRepository (Server): Là lớp thao tác trực tiếp với cơ sở dữ liệu, thực hiện truy vấn xóa dữ liệu theo yêu cầu từ tầng service.
  + Các phương thức:
    - deleteById(summaryId): xóa bản ghi lịch sử tóm tắt theo ID.
    - deleteAllByUserId(userId): xóa tất cả các bản ghi lịch sử thuộc về một người dùng cụ thể.
* Lớp SummaryHistory (Entity): Là lớp thực thể đại diện cho từng bản ghi lịch sử tóm tắt trong hệ thống.
  + Các thuộc tính:
    - historyId: định danh của bản ghi.
    - userId: mã người dùng sở hữu bản ghi.
    - originalText: nội dung gốc được tóm tắt.
    - summaryText: nội dung văn bản sau khi tóm tắt.
    - createdAt: thời gian tóm tắt được thực hiện.
    - isShared: trạng thái chia sẻ (true/false)

##### Thiết kế biểu đồ hoạt động



##### Thiết kế biểu đồ tuần tự



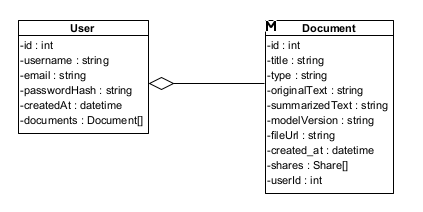
| Ký hiệu | Diễn giải chi tiết |
| --- | --- |
| 1. User -> HistoryPage : Nhấn btnDeleteSingle(summaryId) | Người dùng chọn nút xóa (Delete Button) cho một bản tóm tắt cụ thể (được xác định bởi summaryId) trên trang lịch sử. |
| 2. HistoryPage -> HistoryPage: onClickDeleteSingle(summaryId) | Giao diện chuyển tiếp yêu cầu xóa đến thành phần popup xác nhận xóa mang theo Id của bản tóm tắt cần xóxóa. |
| 3. HistoryPage -> DeleteConfirmPopup : Hiển thị DeleteConfirmPopup | Popup xác nhận hiện ra trên màn hình để hỏi lại người dùng trước khi thực hiện xóa. |
| 4. User-> DeleteConfirmPopup: Nhấn nút Confirm | Sự kiện xác nhận xóa được gửi đến hook useDeleteMutation để xử lý logic xóa. |
| 5. DeleteConfirmPopup -> useDeleteMutation: handleConfirmDelete() | Sự kiện xác nhận xóa được gửi đến hook useDeleteMutation để xử lý logic xóa. |
| 6. useDeleteMutation-> SummaryApi : mutateDelete(summaryId) | Hook mutation gọi hàm API để gửi yêu cầu xóa đến server. |
| 7. SummaryApi -> SummaryController : DELETE /summaries/{summaryId} | Client gửi yêu cầu HTTP DELETE đến endpoint xóa tóm tắt theo ID. |
| 8. SummaryController -> SummaryService : deleteSummary(summaryId) | Controller gọi tầng Service để thực thi nghiệp vụ xóa dữ liệu. |
| 9. SummaryService -> SummaryRepository : deleteById(summaryId) | Service gọi Repository để thao tác trực tiếp với cơ sở dữ liệu. |
| 10. SummaryRepository -> SummaryHistory: xóa bản ghi SummaryHistory tương ứng | Repository tìm đối tượng SummaryHistory có summaryId tương ứng và thực hiện xóa. |
| 11. SummaryHistory--> SummaryRepository : Xác nhận xóa thành công | Entity phản hồi kết quả xóa thành công về Repository. |
| 12. SummaryRepository --> SummaryService: return success | Repository trả về kết quả thành công cho Service. |
| 13. SummaryService --> SummaryController : return success | Service trả về kết quả thành công cho Controller. |
| 14. SummaryController --> SummaryApi : HTTP 200 OK | Controller phản hồi HTTP 200 về phía client. |
| 16. SummaryApi --> useDeleteMutation : return success | API trả về phản hồi thành công cho hook mutation. |
| 17. useDeleteMutation --> HistoryPage : Cập nhật lại danh sách lịch sử | Client cập nhật lại danh sách tóm tắt sau khi xóa. |

## **2. Trương Hoàng Nguyên - B22DCCN597**

### ***2.1. Thiết kế cơ sở dữ liệu liên quan đến module***



### ***2.2. Thiết kế lớp thực thể liên quan đến module***



### ***2.3. Các chức năng phụ trách***

#### 2.3.1. Chức năng tóm tắt văn bản, file:

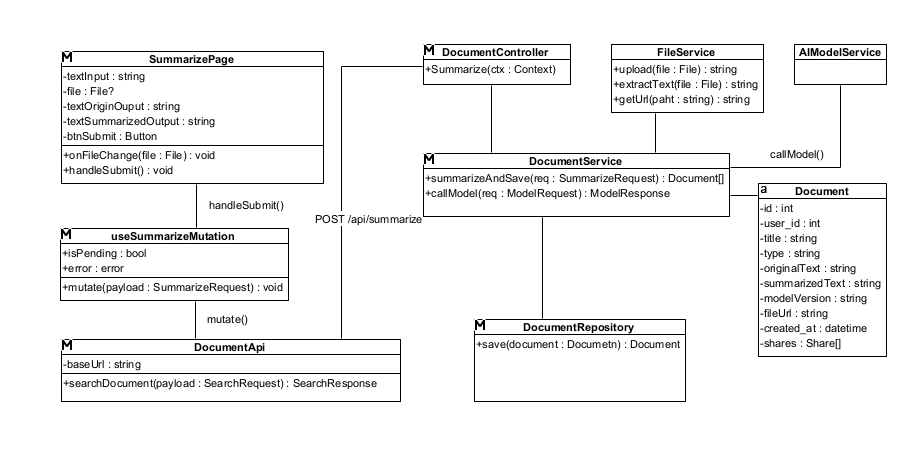
##### Thiết kế giao diện



Giao diện chính bao gồm

* Thanh nhập văn bản, hoặc tải file lên
* Nút submit
* Ô text văn bản gốc và văn bản tóm tắt

##### Sơ đồ lớp chi tiết



Lớp SummarizePage (Client)

* Là lớp giao diện chính cho người dùng thao tác tóm tắt văn bản. Giúp tách biệt phần UI ra khỏi các logic khác.
* Các thuộc tính
  + textInput: nhận vào văn bản tóm tắt.
  + file : nhận file tóm tắt được tải lên.
  + textOriginOutput và textSummarizedOutput: hiển thị kết quả văn bản gốc và văn bản tóm tắt được.
  + btnSubmit: nút submit textInput hoặc file để tóm tắt.
* Các phương thức:
* onFileChange(file : File): phương thức chứa logic đọc file, có thể tái sử dụng khi người dùng thay đổi file tải lên.
* handleSubmit(): phương thức chứa logic gom và đóng gói dữ liệu thành Request và gọi lớp mutation để tóm tắt, tập trung các bước xử lý trong cùng một hàm.

Lớp useSummarizeMutation (Client)

* Là trung gian giữa lớp giao diện và lớp gọi API, được dùng để quản lý các state bất đồng bộ (pending, loading, success, error).
* Các thuộc tính:
  + isPending: cho biết việc gọi API đã hoàn thành hay chưa.
  + error: cho biết việc gọi API có lỗi không.
* Các phương thức:
* mutate(payload : SummarizeRequest): gọi lớp DocumentApi để gọi server. Có thể tùy chỉnh xử lý theo sự kiến onSuccess, onError.

Lớp DocumentApi (Client)

* Là lớp thuộc phía Client, thực hiện việc gọi HTTP request đến server
* Các thuộc tính:
* baseUrl: là đường dẫn gọi đến server.
* Các phương thức
* summarize(): gom thông tin (url, headers, parse JSON) vào và thực hiện gọi API.

Lớp DocumentController (Server)

* Là tầng giao tiếp nhận request từ phía client. Tách logic xử lý các nghiệp vụ của tầng Service với logic request/response và HTTP Status Code.
* Các phương thức:
  + Summarize(ctx : Context): thực hiện phân giải dữ liệu từ ctx thành Request và kiểm tra, gọi DocumentService, nhận kết quả xử lý và dựng response trả về kết quả HTTP tuân theo mô hình MVC.

Lớp DocumentService (Server)

* Chứa logic xử lý chính của các nghiệp vụ liên quan đến Document
* Các phương thức:
  + summarizeAndSave(req : SummarizeRequest): gom logic chính: nhận request, gọi các Service khác để xử lý dữ liệu, gọi model AI, đóng gói Document, lưu vào cơ sở dữ liệu và trả về kết quả
  + callModel(req : ModelRequest): hàm thực hiện gọi modelAI, tách riêng nhằm dễ theo dõi, kiểm thử, tách riêng với logic xử lý chính.

Lớp DocumentRepository (Server)

* Là lớp quản lý phương thức truy xuất cơ sở dữ liệu.
* Các phương thức:
  + save(document : Document): nhận vào đối tượng Document đã đóng gói từ Service và lưu vào cơ sở dữ liệu, trả về đối tượng đã có id.

Lớp FileService (Server)

* Lớp thuộc phía Server ,chứa các hàm chuyên biệt liên quan đến xử lý file.

Các phương thức:

* + upload(file : File): tải văn bản lên kho lưu trữ.
  + getUrl(path : string): lấy đường dẫn truy cập file.
  + extractText(file : File): trích xuất văn bản từ file.

Lớp AIModelService

* Là lớp dịch vụ ngoài bên phía server thuật toán AI.

Lớp Document (Entity)

* Là lớp thực thể hệ thống ,đại diện cho một tài liệu được tóm tắt
* Các thuộc tính: gồm đầy đủ các thuộc tính như trong sơ đồ lớp thực thể.

##### Sơ đồ hoạt động

##### Sơ đồ tuần tự

#### 

1. Người dùng tải lên file hoặc nhập nội dung văn bản, sau đó nhấn Submit tại giao diện SummarizePage.
2. SummarizePage gọi hàm handleSubmit() để xử lý.
3. Hàm handleSubmit() gọi đến lớp useSummarizeMutation để thực hiện mutation gửi yêu cầu tóm tắt.
4. useSummarizeMutation gọi hàm mutate() để gửi yêu cầu đến tầng API.
5. Hàm mutate() gọi đến lớp DocumentApi để bắt đầu quá trình giao tiếp với server.
6. DocumentApi gọi hàm searchDocument() để kiểm tra hoặc tìm kiếm tài liệu cần tóm tắt.
7. Hàm searchDocument() gửi yêu cầu đến server thông qua API POST /api/summarize.
8. DocumentController trên server nhận yêu cầu và gọi hàm Summarize().
9. Hàm Summarize() trong DocumentController gọi đến lớp DocumentService để xử lý logic nghiệp vụ.
10. DocumentService gọi hàm summarizeAndSave() để thực hiện toàn bộ quy trình tóm tắt và lưu tài liệu.
11. Bên trong hàm summarizeAndSave(), DocumentService gọi đến FileService để bắt đầu quá trình xử lý file.
12. FileService thực hiện hàm upload() để tải file người dùng lên hệ thống lưu trữ
13. FileService trả kết quả lại cho DocumentService.
14. DocumentService gọi hàm extractText() trong FileService để trích xuất nội dung văn bản từ file.
15. FileService xử lý việc trích xuất nội dung và trả kết quả cho DocumentService.
16. Sau khi có nội dung văn bản, DocumentService chuẩn bị dữ liệu gửi cho AI để tóm tắt.
17. DocumentService gọi hàm callModel() đến lớp AIModelService để gửi dữ liệu cần tóm tắt.
18. AIModelService xử lý yêu cầu, thực hiện mô hình AI và trả kết quả tóm tắt về cho DocumentService.
19. DocumentService sau đó gọi hàm getUrl() của FileService để lấy đường dẫn (URL) của file vừa xử lý.
20. FileService thực hiện lấy đường dẫn file
21. FileService trả về đường dẫn cho DocumentService.
22. DocumentService gọi hàm Document() để đóng gói thông tin thành đối tượng Document
23. Lớp Document đóng gói dữ liệu tài liệu
24. Lớp Document trả về cho DocumentService.
25. DocumentService gọi lớp DocumentRepository để lưu đối tượng Document vào cơ sở dữ liệu.
26. DocumentRepository thực hiện thao tác lưu trữ thông qua hàm gọi hàm save()
27. DocumentRepository trả kết quả thao tác lưu cho DocumentService
28. DocumentService trả về kết quả cho DocumentController.
29. DocumentController trả về kết quả cho lớp DocumentApi.
30. DocumentApi trả kết quả cho useSummarizeMutation.
31. useSummarizeMutation trả kết quả về cho SummarizePage.
32. SummarizePage hiển thị nội dung tóm tắt lên giao diện.

#### 

#### 2.3.2. Chức năng tìm kiếm văn bản:

##### Thiết kế giao diện

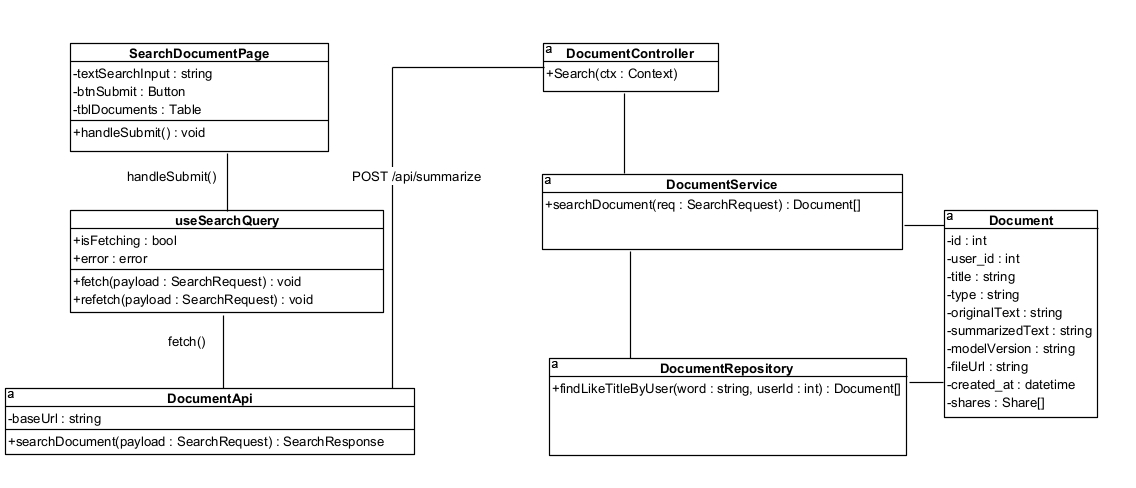




Giao diện chính bao gồm:

* Thanh nhập từ tìm kiếm
* Nút submit
* Bảng kết quả

##### Sơ đồ lớp chi tiết

Lớp SearchDocumentPage (Client)

* Là lớp giao diện chính của tính năng tìm kiếm văn bản tóm tắt. Tách biệt với các phần xử lý logic.
* Các thuộc tính:
  + textSearchInput: ô nhập nội dung tìm kiếm.
  + btnSubmit: nút submit nội dung tìm kiếm
  + tblDocuments: bảng kết quả các Document phù hợp với nội dung tìm kiếm.
* Các phương thức:
  + handleSubmit(): đóng gói nội dung tìm kiếm thành request, gọi lớp query để thực hiện tìm kiếm.

Lớp useSearchQuery (Client)

* Là lớp trung gian giữa giao diện và API, quản lý các state khi thực hiện query.
* Các thuộc tính:
  + isFetching: cho biết quá trình gọi API đã hoàn thành hay chưa.
  + error: chứa thông tin lỗi nếu có lỗi xảy ra khi gọi API.
* Các phương thức:
  + fetch(payload : SearchRequest): thực hiện gọi lớp API để gọi đến server.
  + refetch(payload : SearchRequest): cho phép gọi lại API do các lớp hook trung gian chỉ được gọi một lần.

Lớp DocumentApi (Client)

* Là lớp thuộc phía Client, thực hiện việc gọi HTTP request đến server

Các thuộc tính:

* baseUrl: là đường dẫn gọi đến server.

Các phương thức

* searchDocument(payload : SearchRequest): gom thông tin (url, headers, parse JSON) vào và thực hiện gọi API, đóng gói và trả về kết quả.\

Lớp DocumentController (Server)

* Là tầng giao tiếp nhận request từ phía client. Tách logic xử lý các nghiệp vụ của tầng Service với logic request/response và HTTP Status Code.

Các phương thức:

* + Search(ctx : Context): thực hiện phân giải dữ liệu từ ctx thành Request và kiểm tra, gọi DocumentService, nhận kết quả xử lý và đóng gói response trả về kết quả HTTP tuân theo mô hình MVC.

Lớp DocumentService (Server)

* Chứa logic xử lý chính của các nghiệp vụ liên quan đến Document

Các phương thức:

* + SearchDocument(req : SearchRequest): gọi lớp repository để thực hiện tìm kiếm, đóng gói và trả về kết quả

Lớp DocumentRepository (Server)

* Là lớp quản lý phương thức truy xuất cơ sở dữ liệu.

Các phương thức:

* + FindLikeTitleByUser(word : string, userId : int): nhận vào một từ do người dùng nhập và id của người dùng từ Service, tìm kiếm các Document, gọi lớp Document để đóng gói và trả về kết quả.

Lớp Document (Entity)

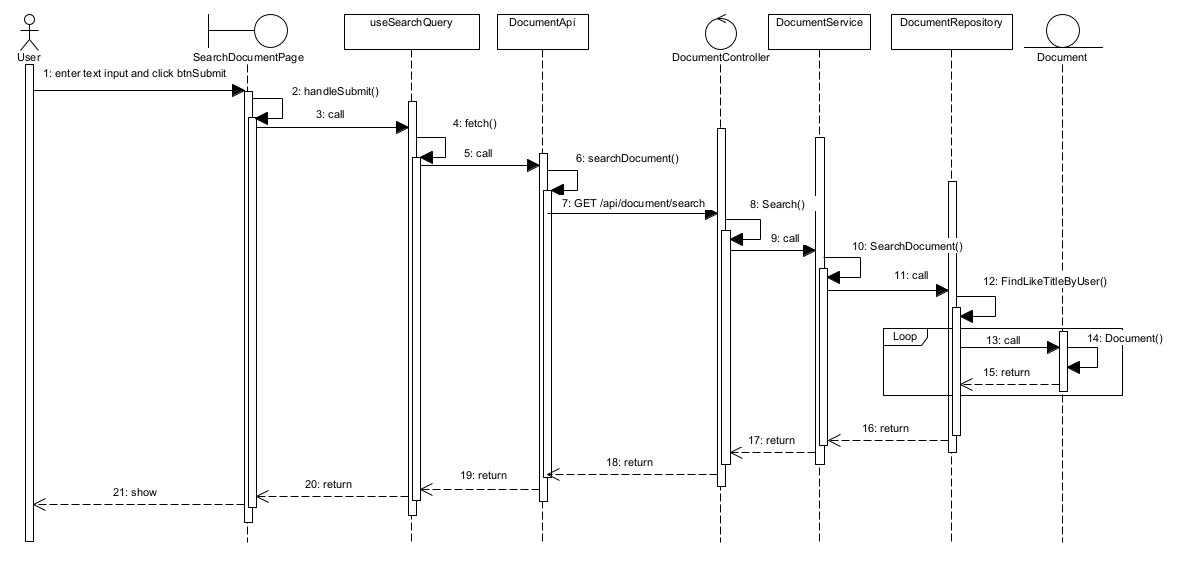
* Là lớp thực thể hệ thống ,đại diện cho một tài liệu được tóm tắt

Các thuộc tính: gồm đầy đủ các thuộc tính như trong sơ đồ lớp thực thể.

##### Sơ đồ hoạt động



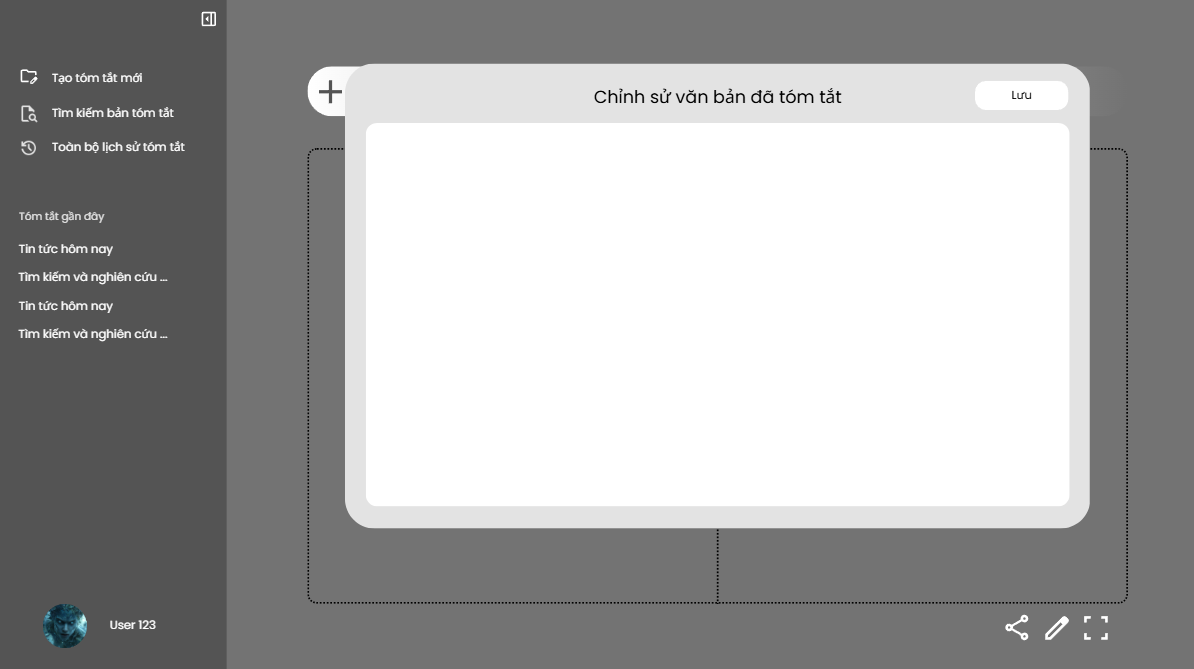
##### Sơ đồ tuần tự

****

1. Tại giao diện SearchDocumentPage, người dùng nhập từ khóa và nhấn submit
2. Lớp SearchDocumentPage gọi hàm handleSubmit()
3. Hàm handleSubmit() gọi đến lớp useSearchQuery để gửi yêu cầu tìm kiếm và theo dõi trạng thái yêu cầu
4. Lớp useSearchQuery gọi hàm fetch()
5. Hàm fetch() gọi đến lớp DocumentApi yêu cầu gọi đến server.
6. Lớp DocumentApi gọi hàm searchDocument()
7. Hàm searchDocument() gọi đến server qua API GET /api/document/search
8. Lớp DocumentController nhận lời gọi và thực hiện hàm Search()
9. Hàm Search() gọi lớp DocumentService yêu cầu thực hiện logic xử lý tìm kiếm văn bản
10. Lớp DocumentService gọi hàm SearchDocument()
11. Hàm SearchDocument() gọi đến lớp DocumentRepository để tìm kiếm trong cơ sở dữ liệu
12. Lớp DocumentRepository gọi hàm FindLikeTitleByUser()
13. Hàm FindLikeTitleByUser() thực hiện tìm kiếm trong cơ sở dữ liệu và lặp gọi đến lớp Document đóng gói dữ liệu thành đối tượng.
14. Lớp Document đóng gói dữ liệu
15. Lớp Document trả về kết quả cho lớp DocumentRepository
16. Lớp DocumentRepository trả về kết quả cho lớp DocumentService
17. Lớp DocumentService trả về kết quả cho lớp DocumentController
18. Lớp DocumentController trả kết quả về cho lớp DocumentApi
19. Lớp DocumentApi trả về kết quả cho lớp useSearchQuery
20. Lớp useSearchQuery trả về kết quả cho lớp SearchDocumentPage
21. Lớp SearchDocumentPage hiển thị kết quả lên giao diện cho người dùng.

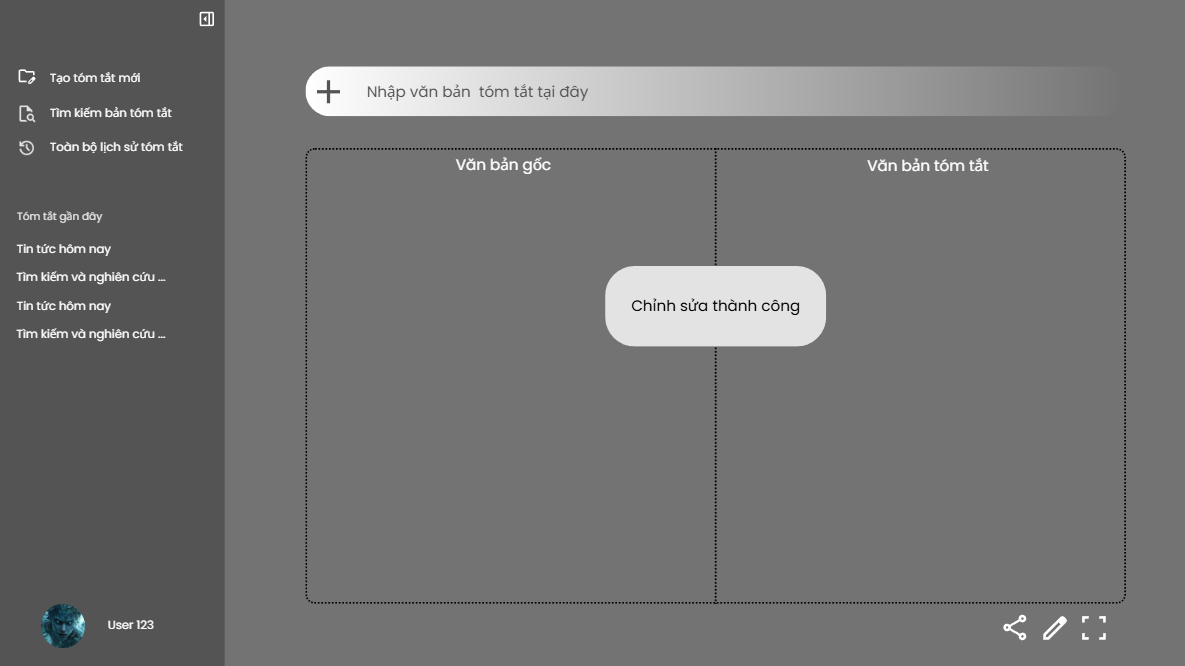
#### 2.3.3. Chức năng chỉnh sửa văn bản tóm tắt:

##### Thiết kế giao diện

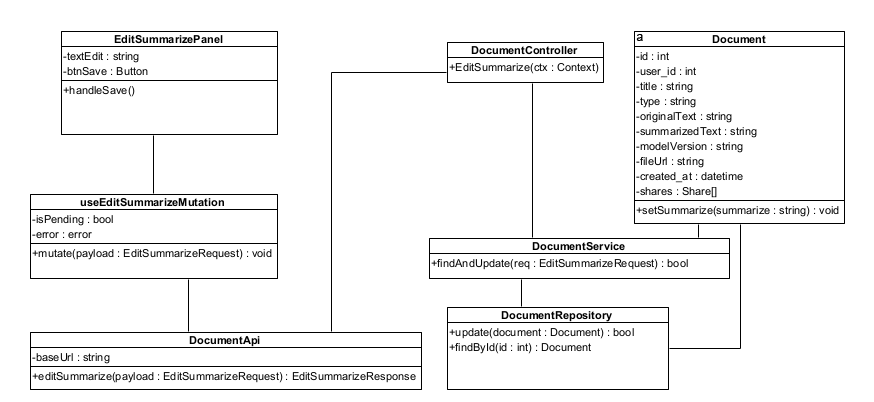


Giao diện chính gồm:

* Ô text chứa văn bản tóm tắt có thể chỉnh sửa được
* Nút “Lưu” để submit



##### Sơ đồ lớp chi tiết

Lớp EditSummarizePanel (Client)

* Là lớp giao diện chính của chỉnh sửa văn bản tóm tắt. Tách biệt với các phần xử lý logic.
* Các thuộc tính:
  + textEdit: chứa nội dung tóm tắt và cho phép người dùng chỉnh sửa
  + btnSubmit: nút submit để lưu lại vẳn bản tóm tắt đã chính sửa.
* Các phương thức:
  + handleSave(): đóng gói nội dung tìm kiếm thành request, gọi lớp mutation để thực hiện lưu nội dung đã chỉnh sửa.

Lớp useEditSummarizeMutation (Client)

* Là lớp trung gian giữa giao diện và API, quản lý các state khi thực hiện query.
* Các thuộc tính:
  + isPending: cho biết quá trình gọi API đã hoàn thành hay chưa.
  + error: chứa thông tin lỗi nếu có lỗi xảy ra khi gọi API.
* Các phương thức:
  + mutation(payload : EditSummarizeRequest): gọi lớp DocumentApi để gọi server. Có thể tùy chỉnh xử lý theo sự kiến onSuccess, onError.

Lớp DocumentApi (Client)

* Là lớp thuộc phía Client, thực hiện việc gọi HTTP request đến server

Các thuộc tính:

* baseUrl: là đường dẫn gọi đến server.

Các phương thức

* editSummarize(payload : SearchRequest): gom thông tin (url, headers, parse JSON) vào và thực hiện gọi API, đóng gói và trả về kết quả.

Lớp DocumentController (Server)

* Là tầng giao tiếp nhận request từ phía client. Tách logic xử lý các nghiệp vụ của tầng Service với logic request/response và HTTP Status Code.

Các phương thức:

* + EditSummarize(ctx : Context): thực hiện phân giải dữ liệu từ ctx thành Request và kiểm tra, gọi DocumentService, nhận kết quả xử lý và đóng gói response trả về kết quả HTTP tuân theo mô hình MVC.

Lớp DocumentService (Server)

* Chứa logic xử lý chính của các nghiệp vụ liên quan đến Document

Các phương thức:

* + findAndUpdate(req : EditSummarizeRequest): gọi lớp repository để thực hiện tìm kiếm, chỉnh sửa, đóng gói, cập nhật và trả về kết quả.

Lớp DocumentRepository (Server)

* Là lớp quản lý phương thức truy xuất cơ sở dữ liệu.

Các phương thức:

* + findById(id : int): nhận vào id và tìm kiếm bản ghi phù hợp, gọi lớp Document để đóng gói và trả về kết quả.
  + update(document : Document): nhận vào một Document, thực hiện cập nhật vào cơ sở dữ liệu và trả về kết quả

Lớp Document (Entity)

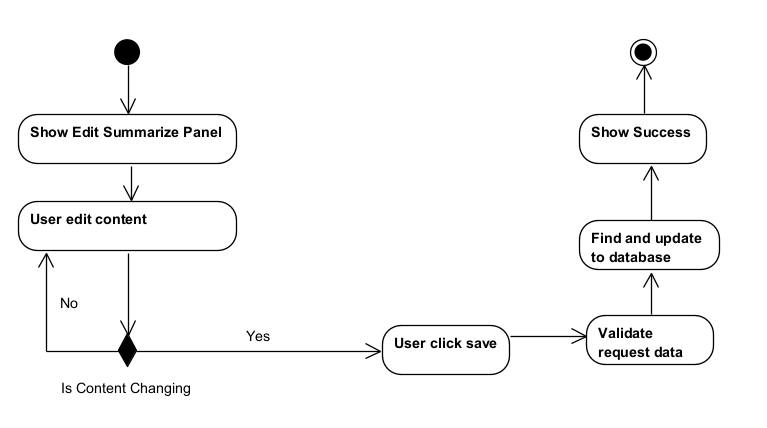
* Là lớp thực thể hệ thống ,đại diện cho một tài liệu được tóm tắt

Các thuộc tính: gồm đầy đủ các thuộc tính như trong sơ đồ lớp thực thể.

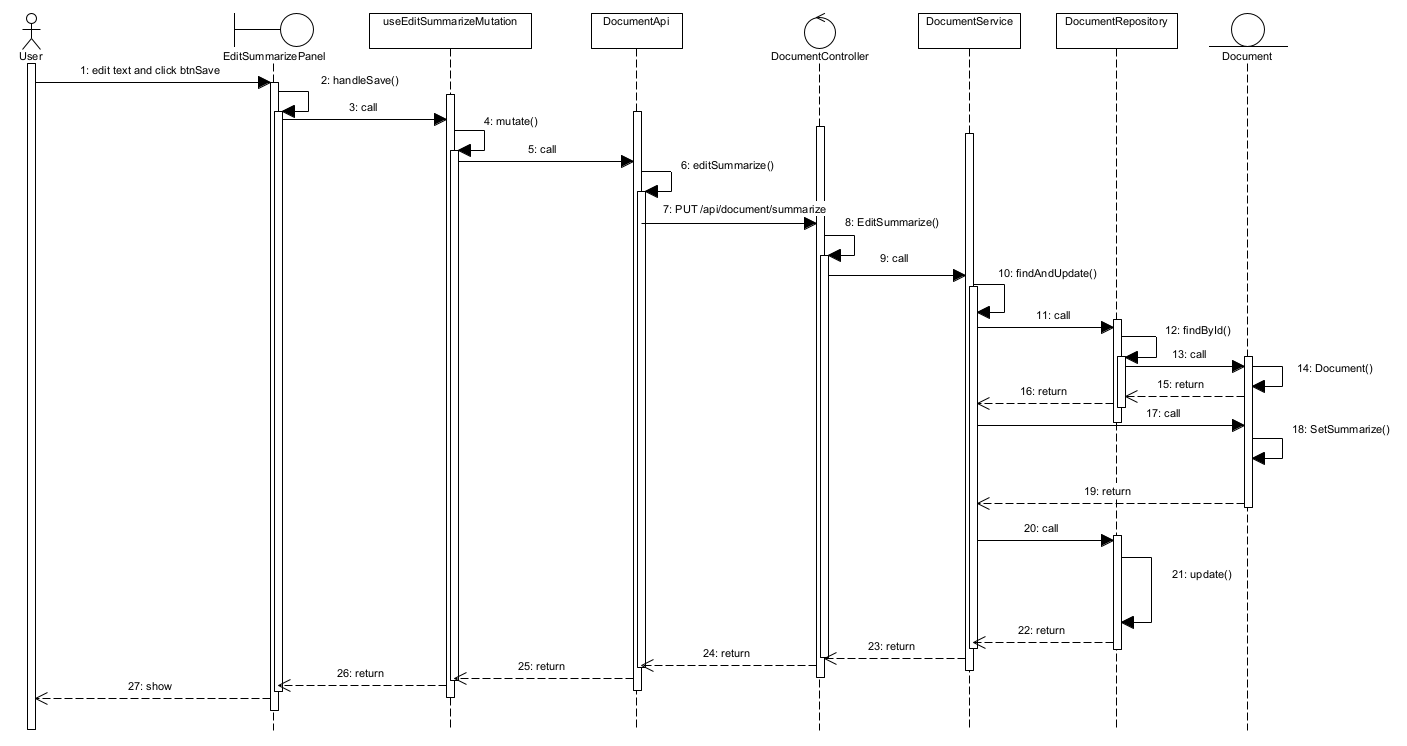
Các phương thức:

* setSummarize(summarize : string): cập nhật trường summarizedText cho đối tượng.

##### Sơ đồ hoạt động



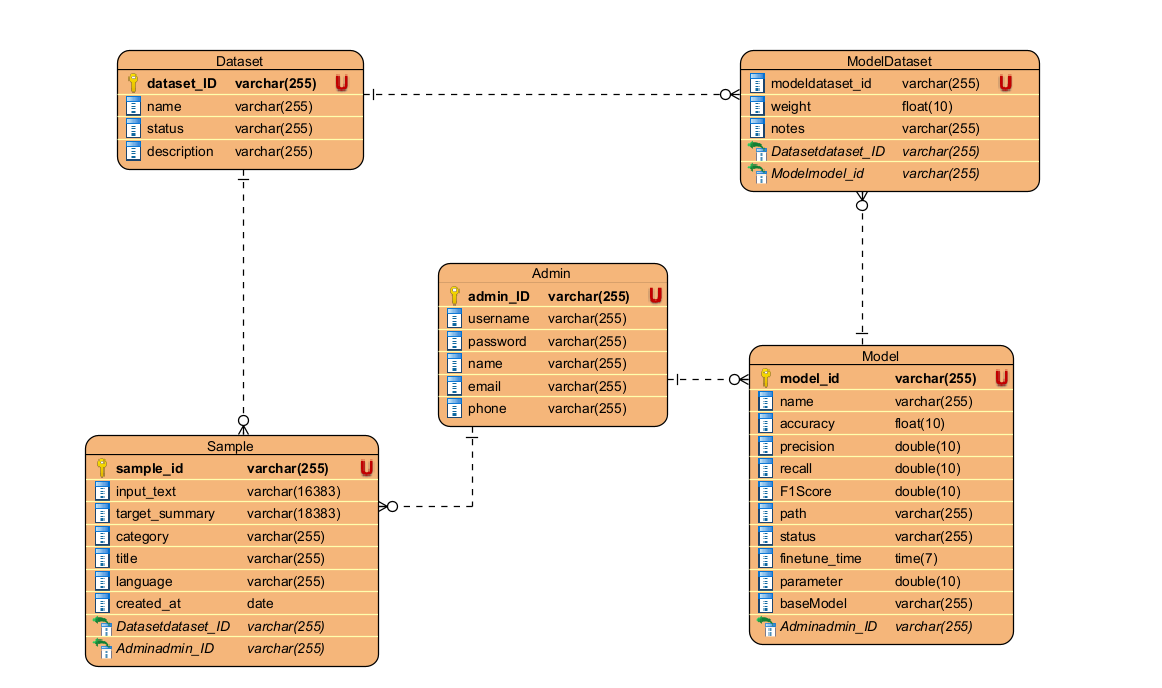
##### Sơ đồ tuần tự



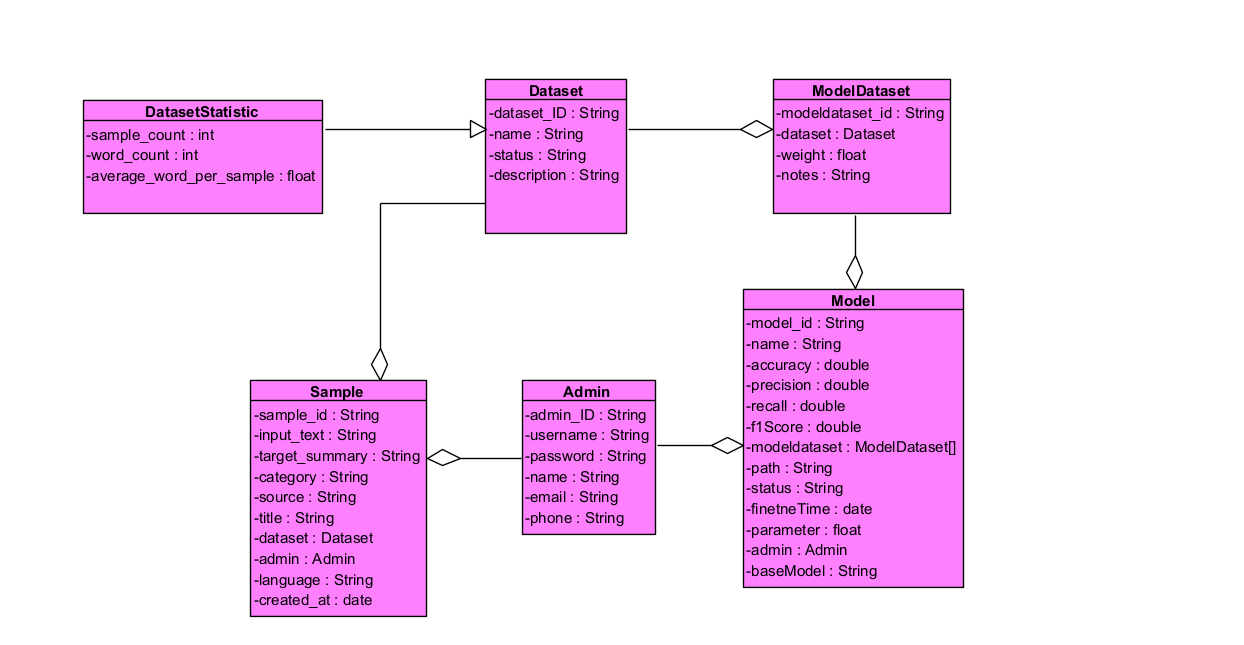
1. Tại giao diện sửa nội dung tóm tắt EditSummarizePanel, người dùng chỉnh sửa nội dung tóm tắt và bấm nút save.
2. Lớp EditSummarizePanel gọi hàm handleSave().
3. Hàm handleSave() gọi đến lớp useEditSummarizeMutation để gửi yêu cầu chỉnh sửa và theo dõi trạng thái yêu cầu.
4. Lớp useEditSummarizeMutation gọi hàm mutate().
5. Hàm mutate gọi đến lớp DocumentApi để thực hiện gọi đến server.
6. Lớp DocumentApi gọi hàm editSummarize().
7. Hàm editSummarize() gọi đến lớp DocumentController ở server qua api PUT /api/document/summarization.
8. Lớp DocumentController gọi hàm EditSummarize()
9. Hàm EditSummarize() gọi đến lớp DocumentService thực hiện logic xử lý
10. Lớp DocumentService gọi hàm FindAndUpdate()
11. Hàm FineAndUpdate() gọi đến lớp DocumentRepository để tìm kiếm trong cơ sở dữ liệu
12. Lớp DocumentRepositoru gọi hàm FindById().
13. Hàm FindById gọi đến lớp Document yêu cầu đóng gói dữ liệu
14. Lớp Document đóng gói dữ liệu thành thực thể.
15. Lớp Document trả về kết quả cho lớp DocumentRepository.
16. Lớp DocumentRepository trả về kết quả.
17. Hàm FineAndUpdate() gọi lớp Document để chỉnh sửa nội dung.
18. Lớp Document gọi hàm SetSummarize().
19. Lớp Document trả về kết quả đã chỉnh sửa cho lớp DocumentService .
20. Lớp FineAndUpdate() gọi lớp DocumentRepository thực hiện cập nhật vào cơ sở dữ liệu.
21. Lớp DocumentRepository gọi hàm update().
22. Lớp DocumentRepository trả về kết quả sau khi cập nhật
23. Lớp DocumentService trả về kết quả cho lớp DocumentController.
24. Lớp DocumentController trả về kết quả cho lớp DocumentApi
25. Lớp DocumentApi trả về kết quả cho lớp useEditSummarizeMutation.
26. Lớp useEditSummarizeMutation trả về kết quả cho lớp EditSummarizePanel.
27. Lớp EditSummarizePanel hiển thị kết quả cập nhật cho người dùng.

## **3. Nguyễn Phong Nhã - B22DCCN573**

### ***3.1. Thiết kế cơ sở dữ liệu liên quan đến module***



### ***3.2. Thiết kế lớp thực thể liên quan đến module***

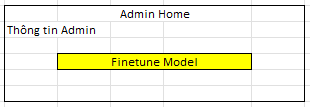


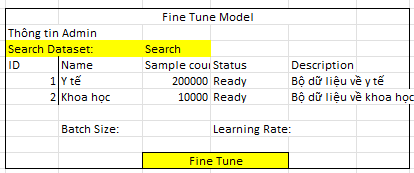
### 

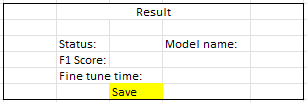
### ***3.3 Các chức năng phụ trách***

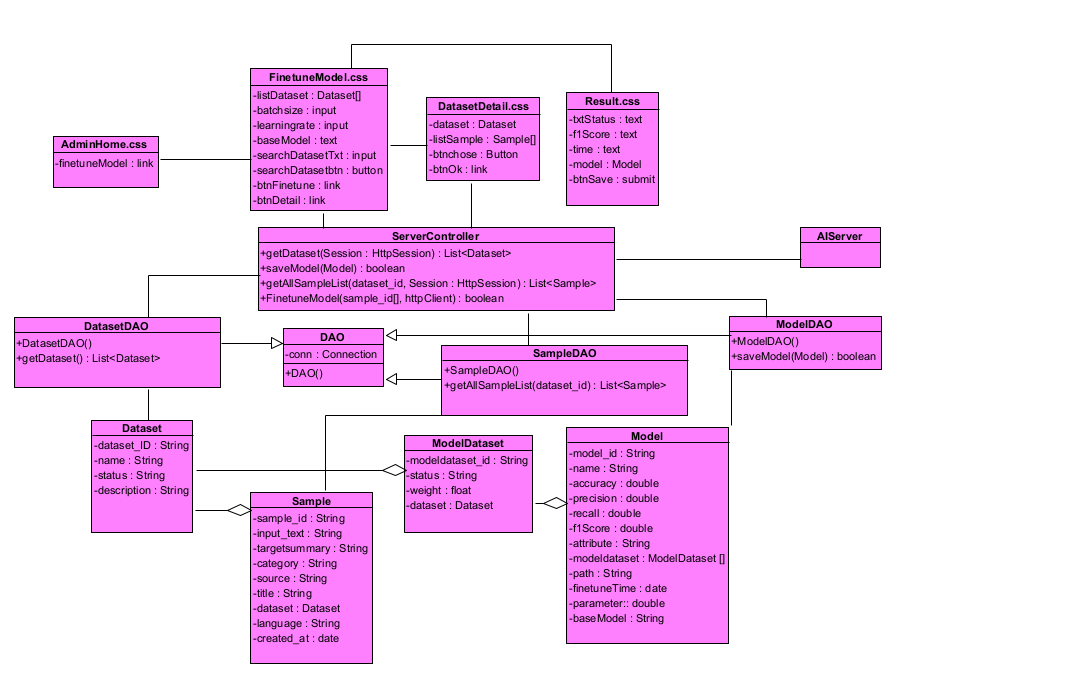
#### 3.3.1 Chức năng Finetune Model

Thiết kế giao diện







**Biểu đồ lớp chi tiết**

**Tầng giao diện**

AdminHome.html là giao diện chính dành cho admin.

- Cần ít nhất 1 thẻ link để truy cập chức năng Fine-tune mô hình.

FinetuneModel.html là giao diện cho phép admin lựa chọn dataset và thiết lập tham số để fine-tune mô hình chatbot. Giao diện chứa ít nhất các trường sau:

- Thông tin danh sách dataset

- Một thẻ input searchDatasetTxt chứa thông tin cần tìm kiếm dataset

- Một nút searchDatasetbtn để gửi yêu cầu tìm kiếm dataset

- Một bảng (table) để chứa thông tin danh sách các dataset bao gồm: ID, tên dataset, mô tả, trạng thái.

- Các thẻ input để nhập thông số huấn luyện:

· batchsize: nhập kích thước batch

· learningrate: nhập tốc độ học

- baseModel: hiển thị tên mô hình gốc

- Một nút btnFinetune để gửi yêu cầu bắt đầu fine-tune mô hình.

Result.html là giao diện hiển thị kết quả sau khi quá trình fine-tune mô hình hoàn tất. Giao diện chứa ít nhất các trường sau:

- txtStatus: hiển thị trạng thái quá trình fine-tune

- f1Score: hiển thị chỉ số F1 của mô hình sau huấn luyện

- time: hiển thị thời gian thực hiện fine-tune

- model: hiển thị tên mô hình đã fine-tune xong

- Một nút btnSave để lưu mô hình vào cơ sở dữ liệu.

**Tầng truy xuất dữ liệu**

**Lớp điều khiển**

ServerController.java:

- getDataset(session: HttpSession): List<Dataset>

· Gọi lớp DatasetDAO để lấy danh sách dataset từ cơ sở dữ liệu.

· Lưu danh sách dataset vào session để hiển thị trên FinetuneModel.html.

- saveModel(model: Model): boolean

· Lấy thông tin model từ giao diện Result.html.

· Gọi lớp ModelDAO để thực hiện lưu mô hình đã fine-tune vào cơ sở dữ liệu.

· Trả về true nếu lưu thành công, ngược lại trả về false.

Lớp giao tiếp với cơ sở dữ liệu

DAO là một lớp chung DAO, có cấu trúc để kết nối với cơ sở dữ liệu và cung cấp kết nối chung cho tất cả các lớp DAO kế thừa trong hệ thống.

DatasetDAO là lớp thao tác với cơ sở dữ liệu liên quan đến đối tượng Dataset. Lớp này có các phương thức:

- getDataset(): List<Dataset> dùng để lấy danh sách dataset từ DB

Output: danh sách Dataset

ModelDAO là lớp thao tác với cơ sở dữ liệu liên quan đến đối tượng Model. Lớp này có các phương thức:

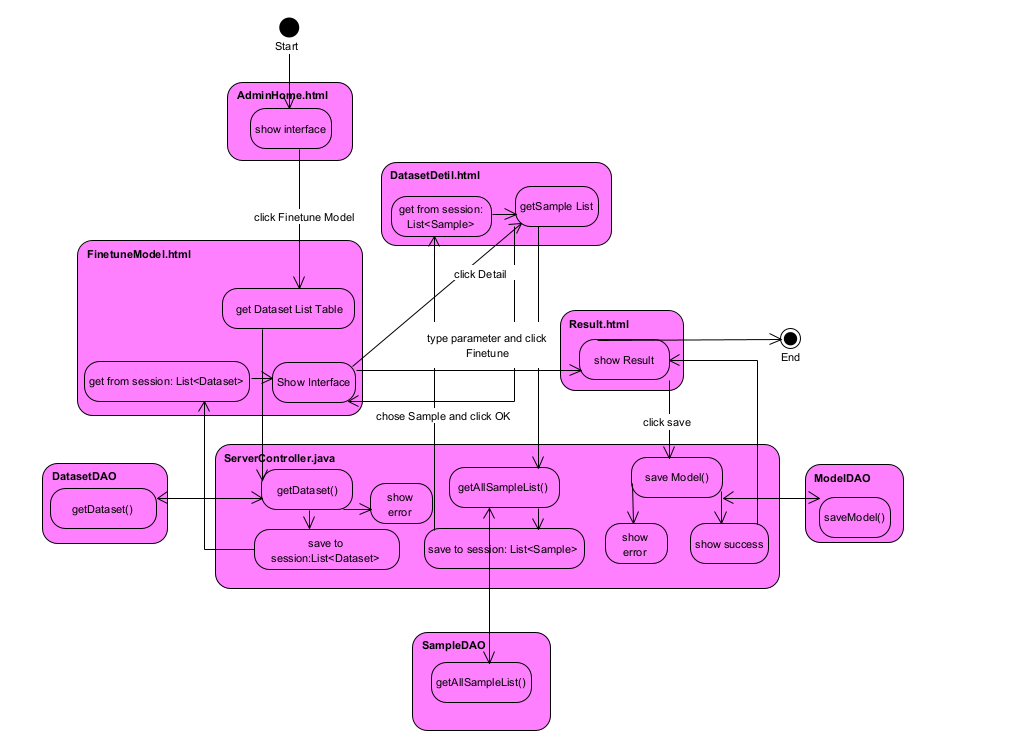
- saveModel(model: Model): boolean – dùng để lưu thông tin mô hình đã fine-tune vào DB

· Input: Model

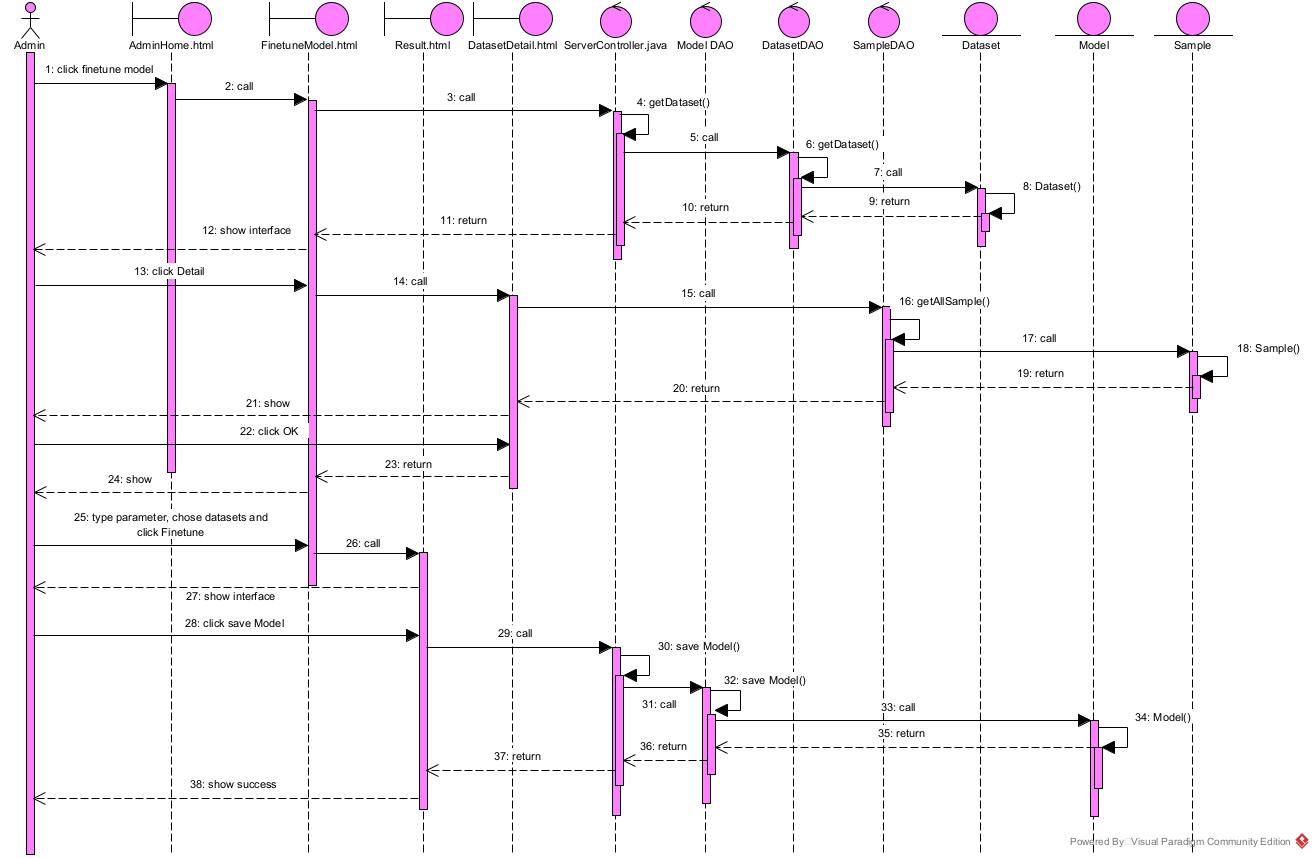
· Output: boolean (true nếu lưu thành công, false nếu thất bại)

**Tầng thực thể**: Gồm các thực thể: Dataset, Sample, ModelDataset, Model, Admin

**Biểu đồ hoạt động:**

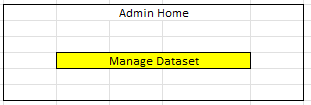
****

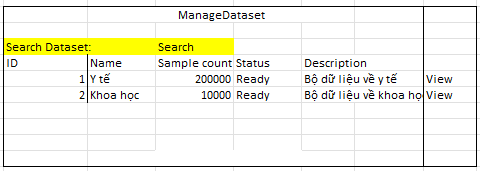
**Biểu đồ tuần tự:**

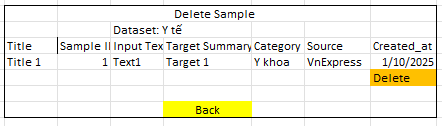
****

1. Tại trang AdminHome.html, Admin chọn chức năng click finetune model.
2. Trang AdminHome.html gọi trang FinetuneModel.html.
3. Trang FinetuneModel.html gọi lớp ServerController.java yêu cầu lấy dữ liệu.
4. Lớp ServerController.java gọi hàm getDataset().
5. Hàm getDataset() gọi lớp DatasetDAO yêu cầu lấy danh sách dataset.
6. Lớp DatasetDAO gọi hàm getDataset().
7. Hàm getDataset() thực hiện và gọi lớp Dataset đóng gói thông tin.
8. Lớp Dataset đóng gói thông tin thực thể.
9. Lớp Dataset trả kết quả về cho hàm getDataset().
10. Hàm getDataset() trả kết quả về cho hàm getDataset().
11. Hàm getDataset() trả kết quả về cho trang FinetuneModel.html.
12. Trang FinetuneModel.html hiển thị show interface cho Admin.
13. Tại trang FinetuneModel.html, Admin click Detail.
14. Trang FinetuneModel.html gọi trang DatasetDetail.html.
15. Trang DatasetDetail.html gọi lớp SampleDAO yêu cầu lấy thông tin chi tiết.
16. Lớp SampleDAO gọi hàm getAllSample().
17. Hàm getAllSample() thực hiện và gọi lớp Sample đóng gói thông tin.
18. Lớp Sample đóng gói thông tin thực thể.
19. Lớp Sample trả kết quả lại cho hàm getAllSample().
20. Hàm getAllSample() trả kết quả lại cho trang DatasetDetail.html.
21. Trang DatasetDetail.html show cho Admin.
22. Admin click OK.
23. Trang DatasetDetail.html trả kết quả về cho trang FinetuneModel.html.
24. Trang FinetuneModel.html show lại cho Admin.
25. Tại trang FinetuneModel.html điền các tham số.
26. Trang FinetuneModel.html gọi trang Result.html.
27. Trang Result.html show interface cho Admin.
28. Tại trang Result.html, Admin click save Model.
29. Trang Result.html gọi lớp ServerController.java yêu cầu lưu model.
30. Lớp ServerController.java gọi hàm save Model().
31. Hàm save Model() gọi lớp Model DAO yêu cầu lưu model.
32. Lớp Model DAO gọi hàm save Model().
33. Hàm save Model() thực hiện và gọi lớp Model đóng gói thông tin.
34. Lớp Model đóng gói thông tin thực thể.
35. Lớp Model trả kết quả lại cho hàm save Model().
36. Hàm save Model() trả kết quả cho hàm save Model().
37. Hàm save Model() trả kết quả cho trang Result.html.
38. Trang Result.html show success cho Admin.

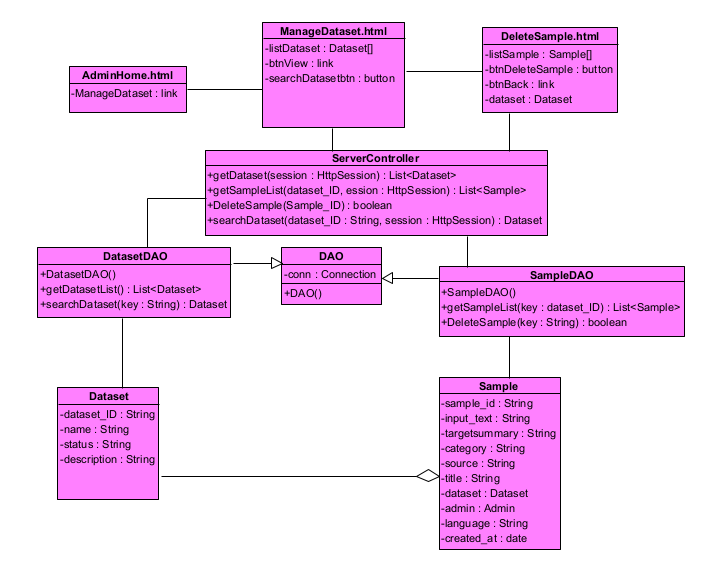
#### 3.3.2 Chức năng xóa dữ liệu mẫu







**Biểu đồ lớp chi tiết:**



**Tầng giao diện**

**AdminHome.html** là giao diện chính dành cho admin.

- Cần ít nhất 1 thẻ **link** để truy cập chức năng **Quản lý Dataset**.

**ManageDataset.html** là giao diện dùng để quản lý danh sách dataset. Giao diện chứa ít nhất các trường sau:

- Thông tin danh sách dataset (listDataset)

- Một nút searchDatasetbtn để gửi yêu cầu tìm kiếm dataset

- Một bảng (table) hiển thị danh sách dataset bao gồm: ID, tên, trạng thái, mô tả

- Tại cuối mỗi dòng của bảng chứa ít nhất một thẻ **link** btnView để xem chi tiết dataset.

**DeleteSample.html** là giao diện cho phép admin xem và xóa các sample thuộc một dataset cụ thể. Giao diện chứa ít nhất các trường sau:

- Thông tin dataset hiện tại

- Thông tin danh sách các sample (listSample) thuộc dataset đó

- Một nút btnDeleteSample để thực hiện xóa sample được chọn

- Một nút btnBack dạng link để quay lại trang **ManageDataset.html**

- Một bảng hiển thị danh sách sample bao gồm: ID, input\_text, targetsummary, category, source, title, created\_at.

**Tầng truy xuất dữ liệu**

**Lớp điều khiển**

**ServerController.java**:

- getDataset(session: HttpSession): List<Dataset>

· Gọi DatasetDAO để lấy danh sách dataset từ cơ sở dữ liệu.

· Lưu danh sách dataset vào session để hiển thị trên **ManageDataset.html**.

- getSampleList(session: HttpSession): List<Sample>

· Gọi SampleDAO để lấy danh sách sample của dataset.

· Lưu danh sách sample vào session để hiển thị trên **DeleteSample.html**.

- DeleteSample(sample\_ID: String): boolean

· Lấy ID của sample cần xóa từ request.

· Gọi SampleDAO để xóa sample tương ứng trong DB.

· Trả về true nếu xóa thành công, ngược lại trả về false.

- searchDataset(dataset\_ID: String, session: HttpSession): Dataset

· Lấy ID dataset từ request.

· Gọi DatasetDAO để tìm kiếm dataset theo ID.

· Nếu tìm thấy: lưu dataset vào session để hiển thị chi tiết.

· Nếu không tìm thấy: trả về null hoặc thông báo lỗi.

**Lớp giao tiếp với cơ sở dữ liệu**

**DAO** là lớp cha cung cấp kết nối cơ sở dữ liệu chung cho các lớp DAO kế thừa trong hệ thống.

* - conn: Connection — đối tượng kết nối DB
* + DAO() — khởi tạo kết nối

**DatasetDAO** là lớp thao tác với cơ sở dữ liệu liên quan đến đối tượng **Dataset**.

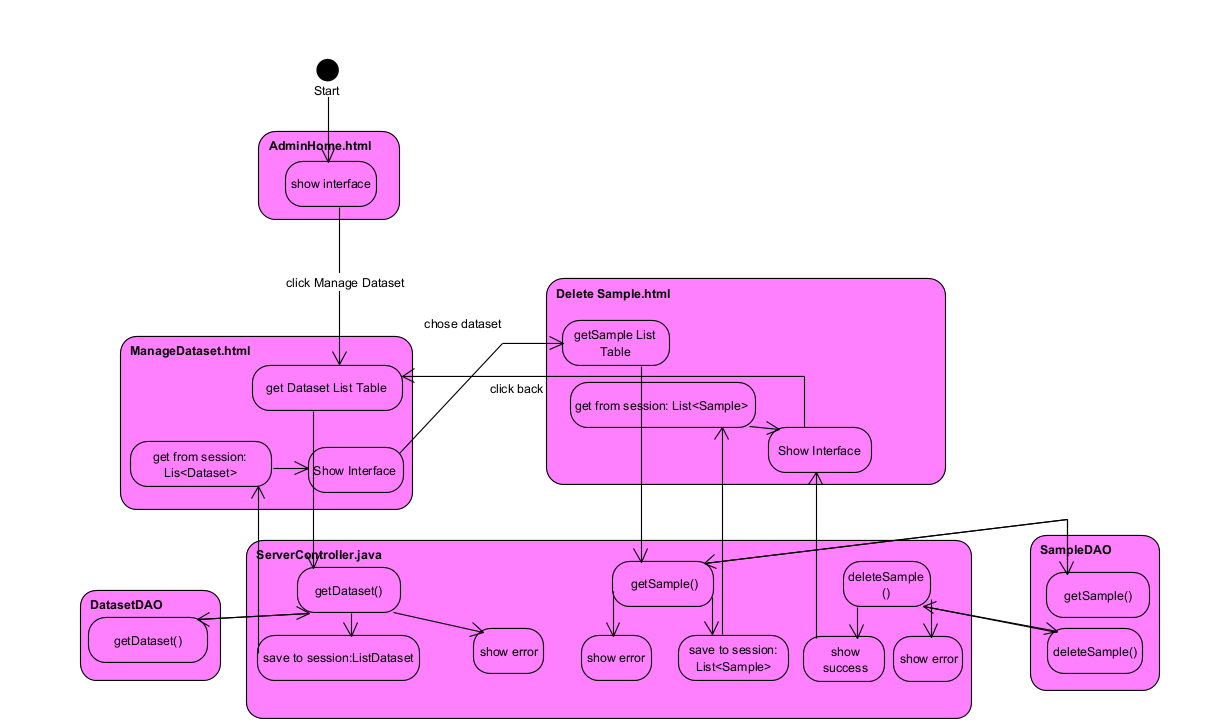
* DatasetDAO() — khởi tạo
* getDatasetList(): List<Dataset> — lấy danh sách tất cả dataset
  + Input: không có
  + Output: danh sách Dataset
* searchDataset(key: String): Dataset — tìm kiếm dataset theo ID
  + Input: key
  + Output: Dataset

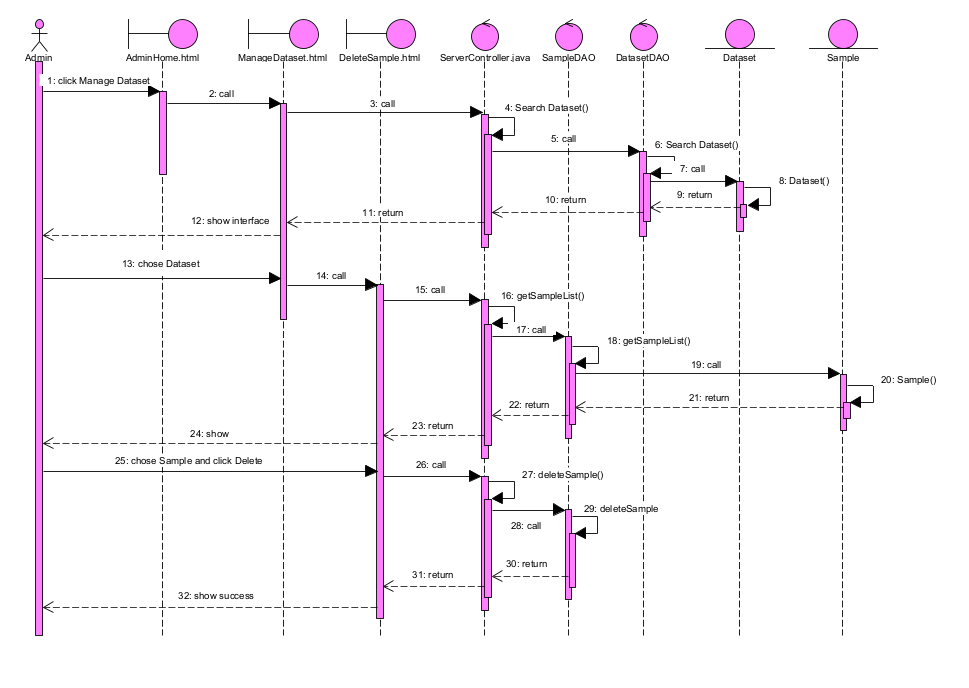
**SampleDAO** là lớp thao tác với cơ sở dữ liệu liên quan đến đối tượng **Sample**.

* SampleDAO(): khởi tạo
* getSampleList(): List<Sample> lấy danh sách sample của một dataset
  + Input: dataset\_ID
  + Output: danh sách Sample
* DeleteSample(key: String): boolean: xóa sample theo ID
  + Input: sample\_id
  + Output: boolean

**Tầng thực thể:** Dataset, Sample

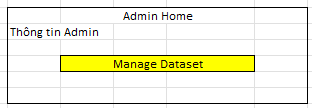
**Biểu đồ hoạt động:**

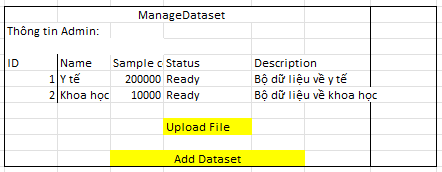
****

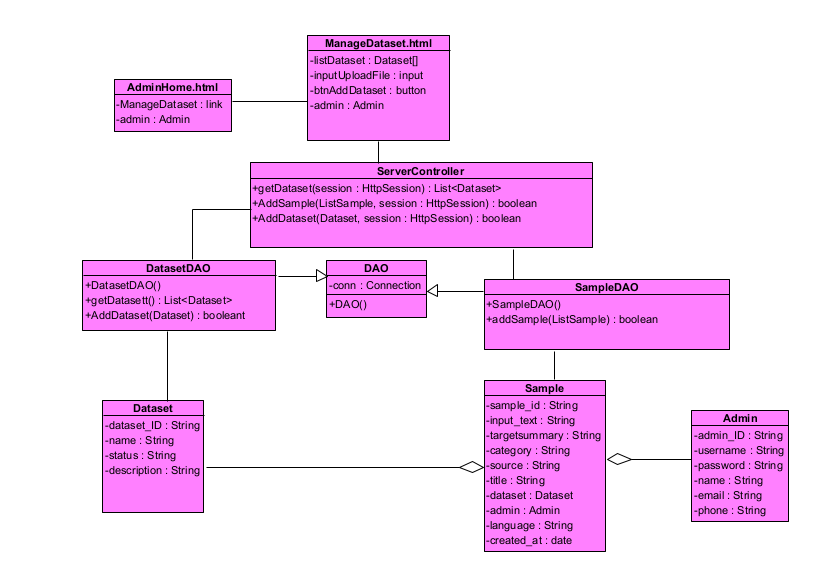
**Biểu đồ tuần tự:  
**

1. Tại trang AdminHome.html, Admin chọn chức năng click Manage Dataset.
2. Trang AdminHome.html gọi trang ManageDataset.html.
3. Trang ManageDataset.html gọi lớp ServerController.java.
4. Lớp ServerController.java gọi hàm Search Dataset().
5. Hàm Search Dataset() gọi lớp DatasetDAO.
6. Lớp DatasetDAO gọi hàm Search Dataset().
7. Hàm Search Dataset() gọi lớp Dataset.
8. Lớp Dataset gọi hàm Dataset().
9. Lớp Dataset trả kết quả về cho lớp DatasetDAO.
10. Lớp DatasetDAO trả kết quả về cho lớp ServerController.java.
11. Lớp ServerController.java trả kết quả về cho trang ManageDataset.html.
12. Trang ManageDataset.html hiển thị show interface cho Admin.
13. Tại trang ManageDataset.html, Admin chọn chose Dataset.
14. Trang ManageDataset.html gọi trang DeleteSample.html.
15. Trang DeleteSample.html gọi lớp ServerController.java.
16. Lớp ServerController.java gọi hàm getSampleList().
17. Hàm getSampleList() gọi lớp SampleDAO.
18. Lớp SampleDAO gọi hàm getSampleList().
19. Hàm getSampleList() gọi lớp Sample.
20. Lớp Sample gọi hàm Sample().
21. Lớp Sample trả kết quả lại cho lớp SampleDAO.
22. Lớp SampleDAO trả kết quả lại cho lớp ServerController.java.
23. Lớp ServerController.java trả kết quả lại cho trang DeleteSample.html.
24. Trang DeleteSample.html show cho Admin.
25. Tại trang DeleteSample.html, Admin chọn chose Sample and click Delete.
26. Trang DeleteSample.html gọi lớp ServerController.java.
27. Lớp ServerController.java gọi hàm deleteSample().
28. Hàm deleteSample() gọi lớp SampleDAO.
29. Lớp SampleDAO gọi hàm deleteSample.
30. Lớp SampleDAO trả kết quả lại cho lớp ServerController.java.
31. Lớp ServerController.java trả kết quả lại cho trang DeleteSample.html.
32. Trang DeleteSample.html hiển thị show success cho Admin.

#### 3.3.3 Chức năng Thêm mới Dataset







**Biểu đồ lớp chi tiết**

**Tầng giao diện**

**AdminHome.html** là giao diện chính dành cho admin.

- Cần ít nhất 1 thẻ **link** để truy cập chức năng **Quản lý Dataset**.

- Thông tin **admin** được hiển thị trên giao diện.

**ManageDataset.html** là giao diện dùng để quản lý danh sách dataset và thêm dataset mới vào hệ thống. Giao diện chứa ít nhất các trường sau:

- Thông tin danh sách dataset (listDataset) hiển thị dưới dạng bảng (table), bao gồm: ID, tên dataset, trạng thái, mô tả.

- Một thẻ **input** UploadFile để admin upload file chứa dữ liệu mẫu (sample) cho dataset mới.

- Một nút btnAddDataset để gửi yêu cầu thêm dataset mới vào hệ thống.

- Thông tin **admin** được hiển thị trên giao diện để xác định người thực hiện thao tác.

**Tầng truy xuất dữ liệu**

**Lớp điều khiển**

**ServerController.java**:

- getDataset(session: HttpSession): List<Dataset>

o Gọi lớp DatasetDAO để truy vấn danh sách dataset từ DB.

o Lưu danh sách dataset vào session để hiển thị trên **ManageDataset.html**.

- AddDataset(dataset: Dataset, session: HttpSession): boolean

o Lấy thông tin dataset mới từ request (được tạo sau khi upload file).

o Gọi DatasetDAO để thêm dataset vào cơ sở dữ liệu.

o Trả về true nếu thêm thành công, ngược lại false.

- AddSample(listSample: List<Sample>, session: HttpSession): boolean

o Sau khi dataset mới được thêm thành công, lấy danh sách sample từ file upload.

o Gọi SampleDAO để thêm toàn bộ sample vào DB và liên kết với dataset vừa thêm.

o Trả về true nếu thêm thành công, ngược lại false.

**Lớp giao tiếp với cơ sở dữ liệu**

**DAO** là lớp chung cung cấp kết nối cơ sở dữ liệu và được kế thừa bởi các lớp DAO khác trong hệ thống.

* - conn: Connection — đối tượng kết nối DB.
* + DAO() — khởi tạo kết nối dùng chung.

**DatasetDAO** là lớp thao tác với cơ sở dữ liệu liên quan đến đối tượng **Dataset**.

- DatasetDAO() — khởi tạo.

- getDataset(): List<Dataset> — lấy danh sách tất cả dataset.

o Output: danh sách Dataset.

- AddDataset(dataset: Dataset): boolean — thêm dataset mới vào DB.

o Input: Dataset.

o Output: boolean.

**SampleDAO** là lớp thao tác với cơ sở dữ liệu liên quan đến đối tượng **Sample**.

- SampleDAO() — khởi tạo.

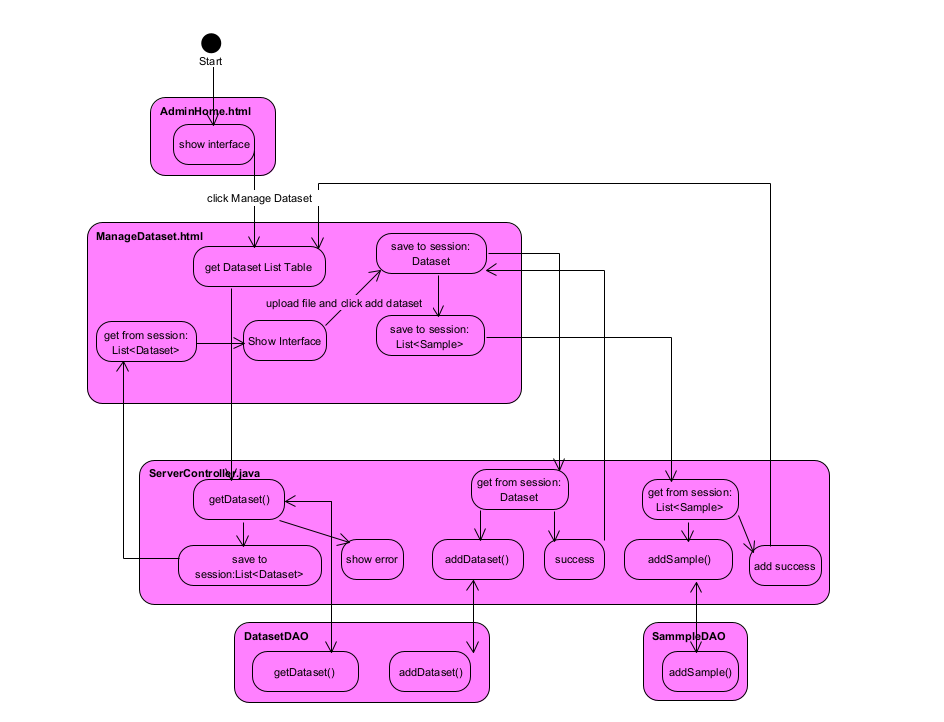
- addSample(Sample: List<Sample>): boolean — thêm nhiều sample cùng lúc vào DB và liên kết với dataset tương ứng.

- Input: listSample.

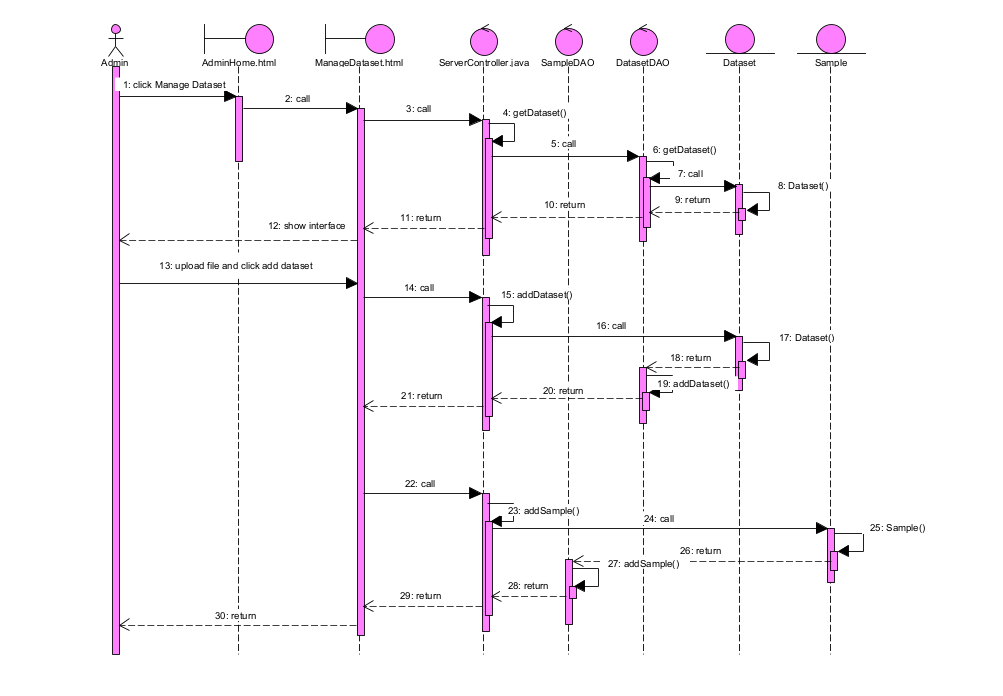
- Output: boolean.

**Tầng thực thể:** Gồm các lớp Dataset, Sample, Admin

**Biểu đồ hoạt động:**

****

**Biểu đồ tuần tự:**

****