

⚡️BÁO CÁO TIẾN ĐỘ THỰC TẬP⚡️

🌱Phạm Hoàng Tuấn - Trần Lê Tuyết Hương🌱

## [**TUẦN 1: 23/05/2022 - 29/05/2022**](https://github.com/hoangtuandev/report-vnptit)

### 🐟 ***Câu 1: Kiến trúc Microservice ?***

* Một ứng dụng được chia thành nhiều “thành phần” độc lập về mặt dữ liệu, đủ nhỏ về kích cỡ và đủ độ ảnh hưởng trong hệ thống.
* Mỗi microservice có khả năng độc lập, đảm báo lắp vào hoặc tháo ra khỏi hệ thống không làm ảnh hưởng đến các thành phần khác
* Có 4 đặc trưng: micro-service, tính độc lập, tính chuyên biệt, phòng chống lỗi.

### 🐠 ***Câu 2: So sánh kiến trúc Ứng dụng nguyên khối và Microservice ?***

| **Mô hình Microservice** | **Mô hình nguyên khối** |
| --- | --- |
| - Các microservice hoạt động tách biệt nhau trong hệ thống, do vậy việc build một microservice cũng độc lập với việc build các microservice khác. | - Toàn bộ ứng dụng là một khối lớn, trong khối lớn ấy có chia thành các mô đun nhỏ, mỗi mô đun thực hiện một nhiệm vụ riêng và các mô đun thường gọi nhau qua function call. |
| - Mỗi microservice là một dịch vụ chuyên biệt, có thể hoạt động độc lập. | - Việc phát triển và triển khai ứng dụng với kiến trúc này khá đơn giản khi mà các IDE hỗ trợ rất tốt việc kiểm tra và chạy ứng dụng với chỉ một cú click chuột hay một phím tắt. |
| -Kiến trúc microservice sinh ra là để dành cho các hệ thống từ lớn đến vô cùng lớn. | Kiến trúc này cũng đặc biệt phù hợp với các công ty outsource. |

### 🐋 ***Câu 3: So sánh SQL Database và NoSQL Database***

|  | **CSDL quan hệ** | **CSDL không quan hệ** |
| --- | --- | --- |
| **Khối lượng công việc tối ưu** | Cơ sở dữ liệu quan hệ được thiết kế dành cho các ứng dụng xử lý giao dịch trực tuyến (OLTP). | Các cơ sở dữ liệu NoSQL được thiết kế cho các mẫu truy cập dữ liệu, bao gồm các ứng dụng có độ trễ thấp. |
| **Mô hình dữ liệu** | Mô hình quan hệ chuẩn hóa dữ liệu vào bảng được hình thành từ hàng và cột. | Các cơ sở dữ liệu NoSQL cung cấp nhiều mô hình dữ liệu khác nhau như khóa-giá trị, tài liệu và biểu đồ, được tối ưu hóa để đạt hiệu năng và quy mô tối ưu. |
| **Thuộc tính ACID** | Cơ sở dữ liệu quan hệ có các thuộc tính mang tính nguyên tố, nhất quán, tách biệt và bền vững (ACID) | Cơ sở dữ liệu NoSQL thường phải đánh đổi bằng cách nới lỏng một số thuộc tính ACID này của cơ sở dữ liệu quan hệ để có mô hình dữ liệu linh hoạt hơn có khả năng thay đổi quy mô theo chiều ngang. |
| **Hiệu năng** | Hiệu năng thường phụ thuộc vào hệ thống con của ổ đĩa. | Hiệu năng thường được xem là chức năng của kích cỡ cụm phần cứng ngầm, độ trễ mạng và ứng dụng đưa ra lệnh gọi. |
| **Quy mô** | Cơ sở dữ liệu quan hệ thường tăng quy mô bằng cách tăng năng lực điện toán của phần cứng hoặc tăng quy mô bằng cách thêm bản sao của khối lượng công việc chỉ đọc. | Cơ sở dữ liệu NoSQL thường có tính phân mảnh cao do các mẫu truy cập khóa-giá trị có khả năng tăng quy mô bằng cách sử dụng kiến trúc được phân phối để tăng thông lượng, đem đến hiệu năng ổn định với quy mô gần như không giới hạn. |
| **API** | Yêu cầu lưu trữ và truy xuất dữ liệu được truyền đạt bằng cách sử dụng các truy vấn nhất quán với ngôn ngữ truy vấn có cấu trúc (SQL). | API trên cơ sở đối tượng cho phép các nhà phát triển ứng dụng dễ dàng lưu trữ và truy xuất cấu trúc dữ liệu trong bộ nhớ. |

### 🐬 ***Câu 4: Cơ sở dữ liệu phân tán***

Cơ sở dữ liệu phân tán phân tán là một tập hợp dữ liệu có liên quan (về logic) được dùng chung và phân tán về mặt vật lí trên một mạng máy tính.

### 🐳 ***Câu 5: Tìm hiểu về Angular***

[**Angular**](https://angular.io/) là một Javascript Framework dùng để viết giao diện Web, xây dựng các Single Page Application(SPA)

### [Ưu điểm của Angular](https://angular.io/)

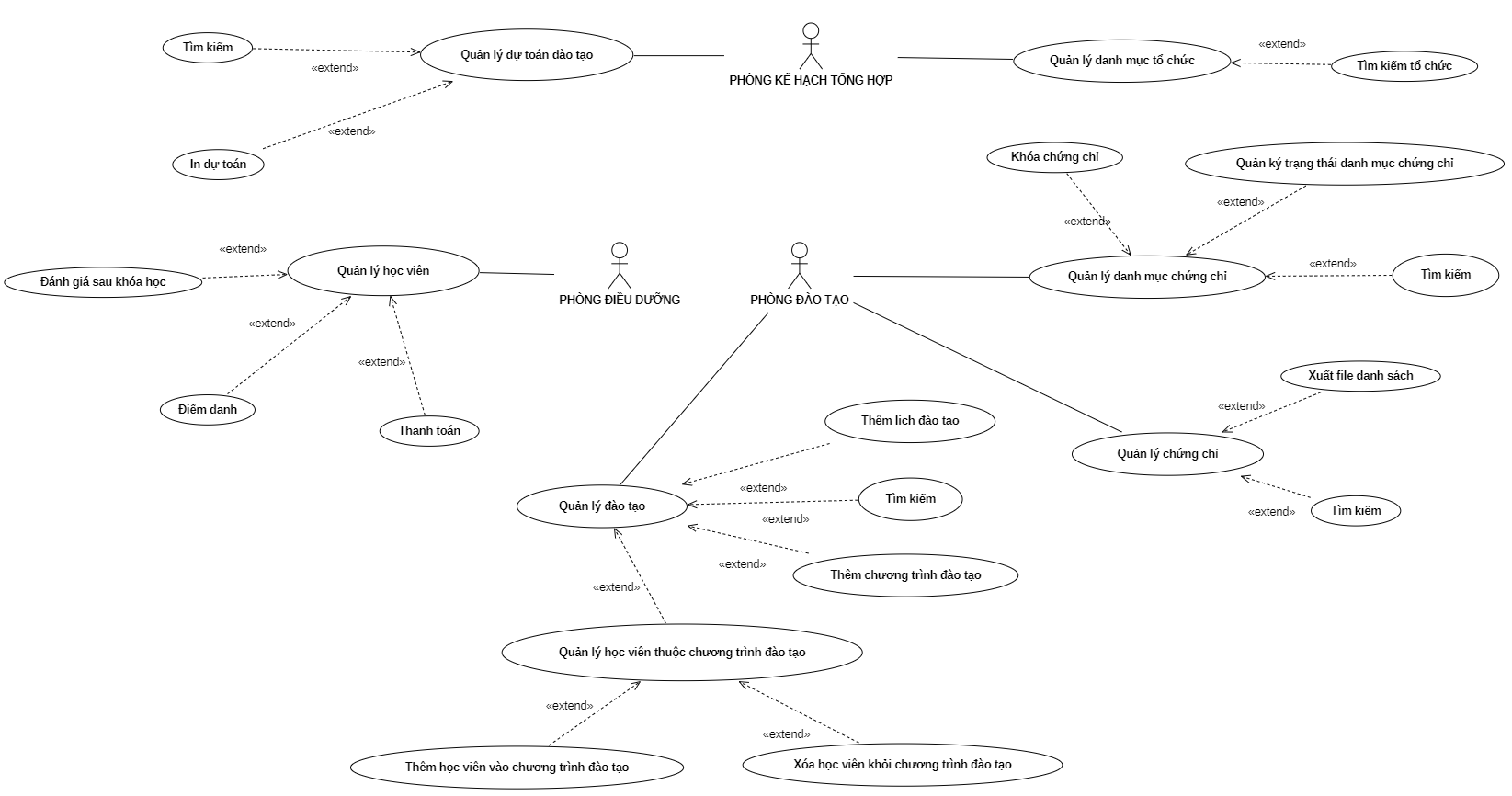
* Cú pháp mã lệnh đơn giản hơn Javascript thuần
* Phân chia layout thành các component, tăng khả năng tái sử dụng và sửa lỗi
* Tạo ra các Single Page Application (SPA).

### 🍃 ***Câu 6: Tìm hiểu về Spring Boot***

* [**Spring Boot**](https://spring.io/projects/spring-boot) là một dự án phát triển bởi JAV (ngôn ngữ java) trong hệ sinh thái Spring framework. Nó giúp cho các lập trình viên chúng ta đơn giản hóa quá trình lập trình một ứng dụng với Spring, chỉ tập trung vào việc phát triển business cho ứng dụng.
* [**Spring Boot**](https://spring.io/projects/spring-boot) là một giải pháp được sử dụng rộng rãi ngày nay để phát triển ứng dụng web Java. Nó có một quy ước cố định về cách tiếp cận cấu hình. Nó hoàn toàn được điều khiển bởi cấu hình và làm cho việc sử dụng Spring Framework và nhiều thư viện của bên thứ ba khác trở nên thú vị. Các ứng dụng Spring Boot là cấp sản xuất và chỉ có thể chạy trong bất kỳ môi trường nào có cài đặt JVM. Nó tự động cấu hình Spring bất cứ khi nào có thể với các giá trị mặc định hợp lý và có POM khởi động cho nhiều mô-đun và thư viện của bên thứ ba. Nó không yêu cầu bất kỳ cấu hình XML nào và cho phép bạn tùy chỉnh các bean tự động định cấu hình bằng cách sử dụng cấu hình Java.

## [**TUẦN 2: 30/05/2022 - 05/06/2022**](https://github.com/hoangtuandev/report-vnptit)

### 🌲 ***SƠ ĐỒ USECASE***



SoDoUseCase

🌴**1. Chức năng Quản lý danh mục tổ chức**

* Tìm kiếm tổ chức: tìm kiếm theo: mã tổ chức, tên tổ chức.

🌴**2. Chức năng Quản lý danh mục chứng chỉ**

* Tìm kiếm: tìm kiếm theo tên danh mục chứng chỉ.
* Quản lý trạng thái danh mục chứng chỉ: mỗi danh mục chứng chỉ có trạng thái hoặc “Đang sử dụng” hoặc “Ngưng sử dụng”
* Khóa chứng chỉ:
  + Thay dổi trạng thái danh mục chứng chỉ sang “Ngưng sử dụng”.
  + Đảm bảo lưu trữ chứng chỉ của nhân viên dù chứng chỉ đã ngưng sử dụng.

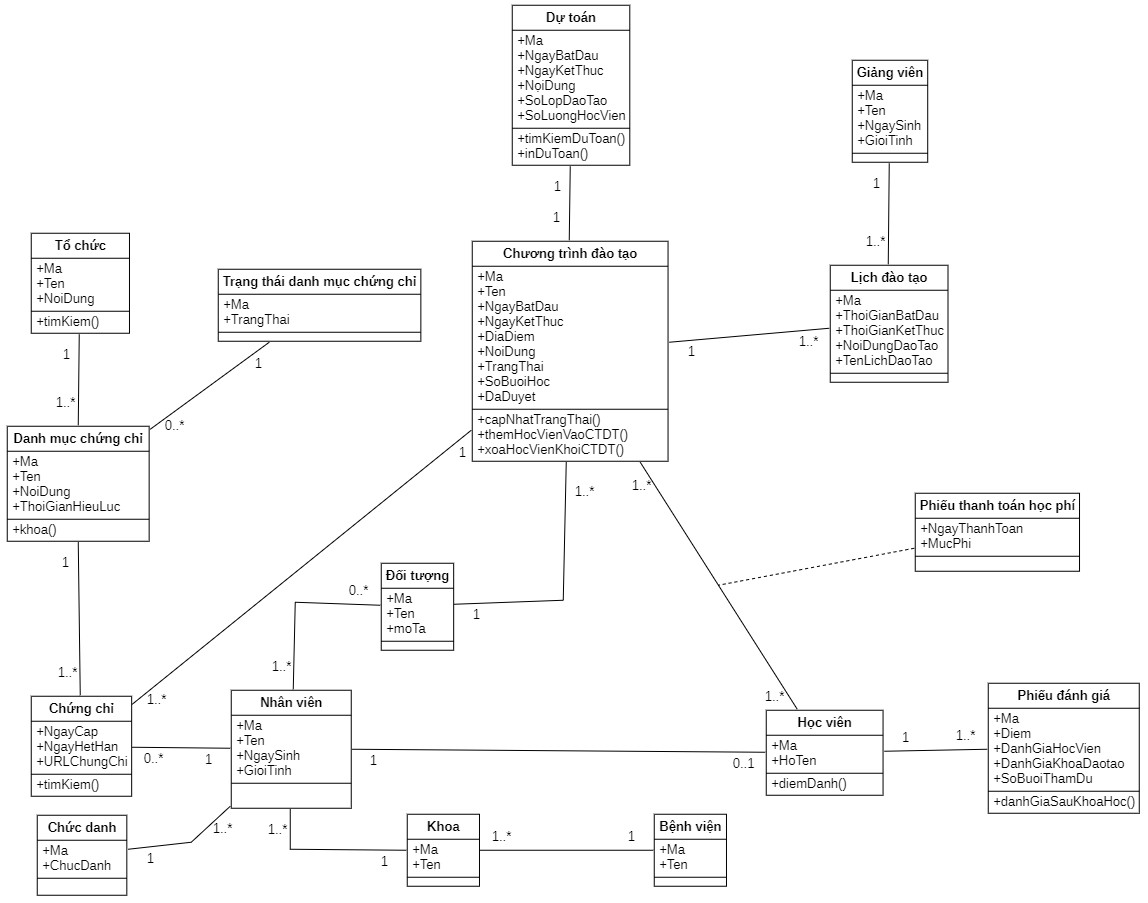
🌴**3. Chức năng Quản lý chứng chỉ** - Tìm kiếm: tìm kiếm chứng chỉ của nhân viên theo: mã nhân viên, tên nhân viên, tên đơn vị. - Xuất file danh sách: xuất file danh sách chứng chỉ của nhân viên.

🌴**4. Chức năng Quản lý đào tạo** - Thêm chương trình đào tạo. - Thêm lịch đào tạo: mỗi Chương trình đào tạo sẽ có một hoặc nhiều lịch đào tạo theo: mã đào tạo, tên đào tạo, ngày bắt đầu, ngày kết thúc, trạng thái. - Tìm kiếm: tìm kiếm chương trình đào tạo. - Quản lý học viên thuộc chương trình đào tạo: thêm học viên vào chương trình đào tạo, xóa học viên khỏi chương trình đào tạo.

🌴**5. Chức năng Quản lý học viên** - Đánh giá học viên sau khóa học. - Điểm danh: điểm danh các học viên. - Thanh toán: đánh dấu các học viên đã thanh toán học phí.

🌴**6. Chức năng quản lý dự toán** - Tìm kiếm: tìm kiếm dự toán đào tạo theo: mã dự toán, số lượng học viên. - In dự toán.

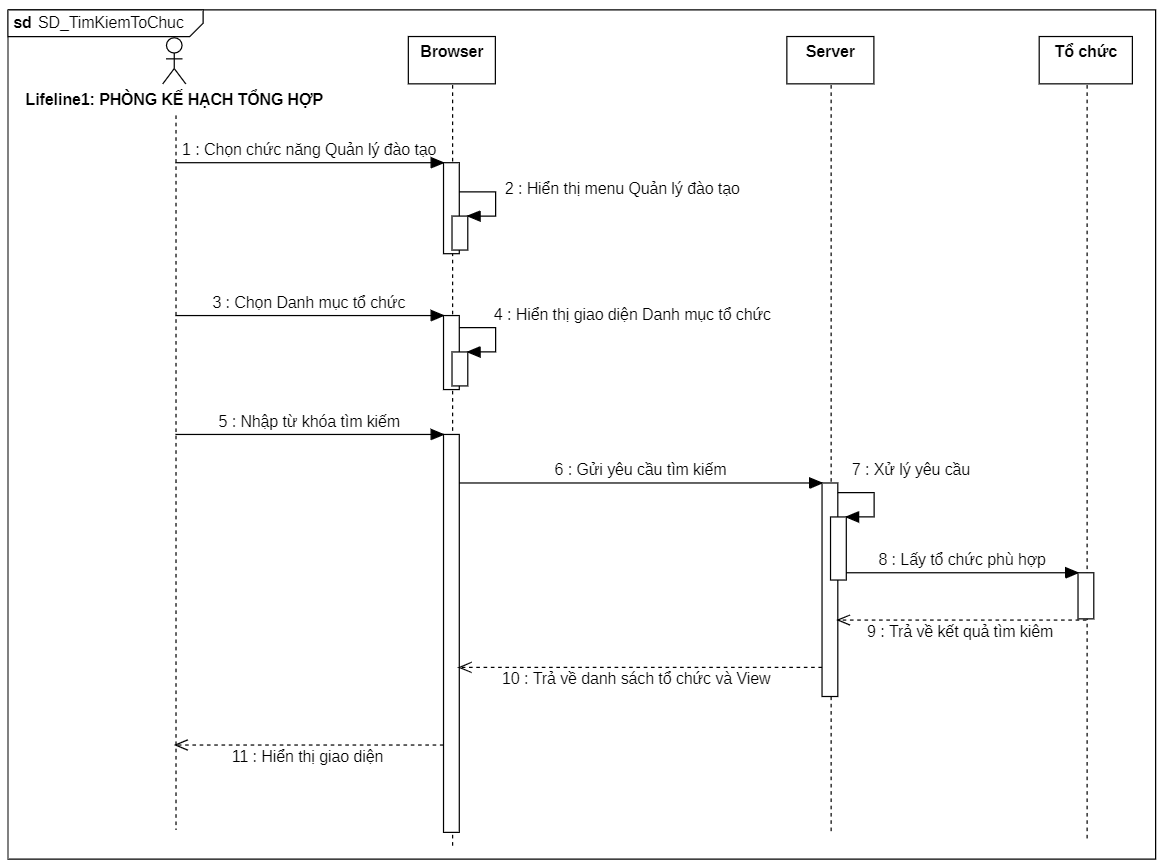
### ✏️ ***SƠ ĐỒ CLASS***



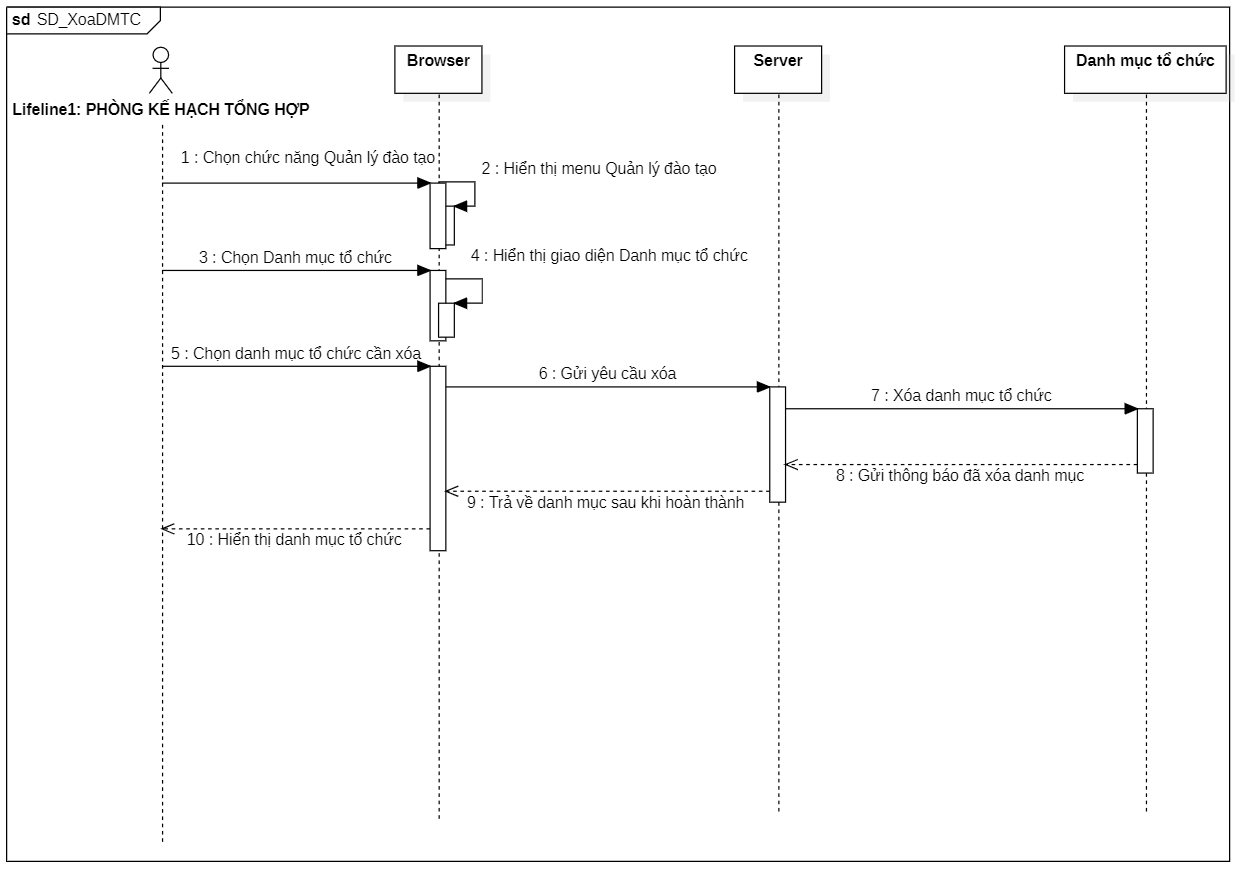
SoDoUseClass

### 🍀 ***SƠ ĐỒ SEQUENCE***

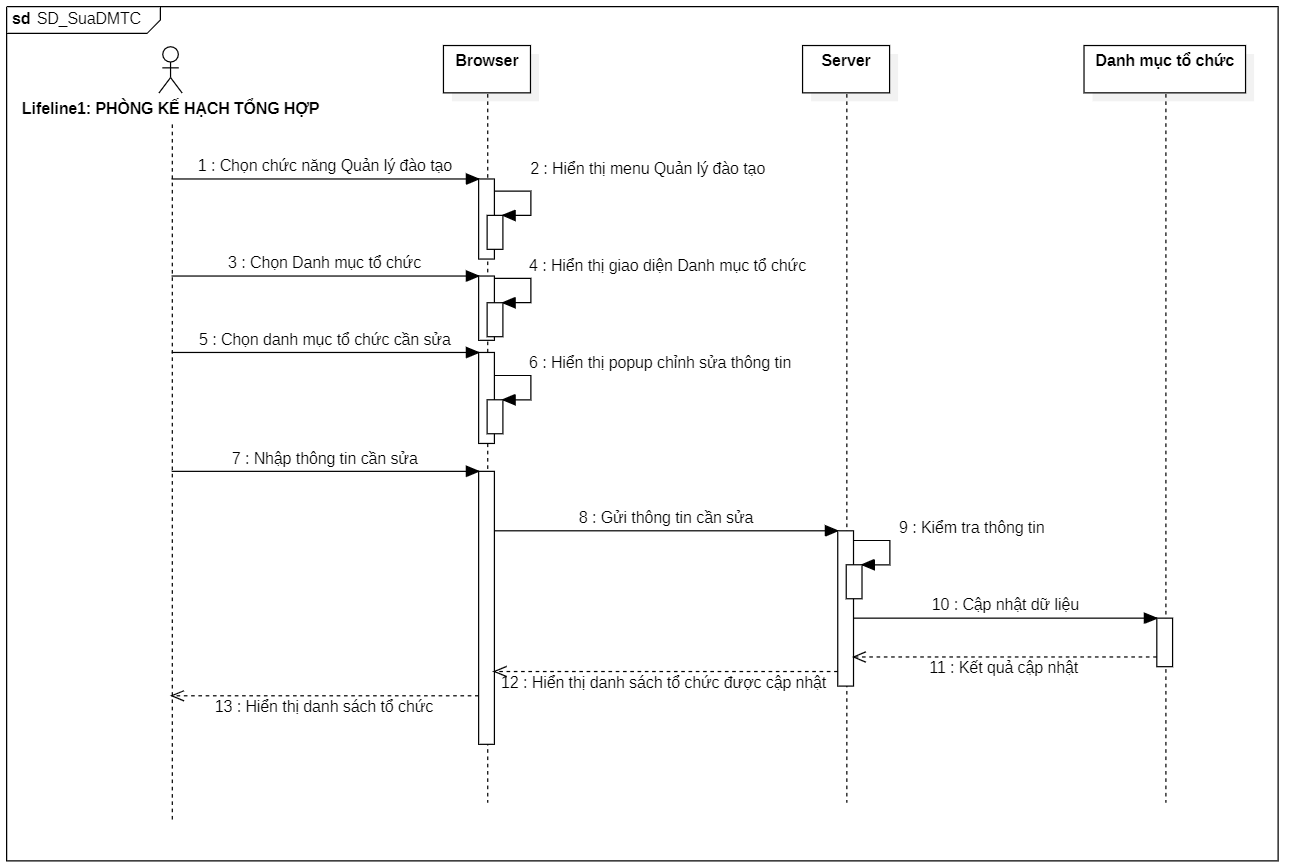
🌿**1. Tìm kiếm tổ chức**



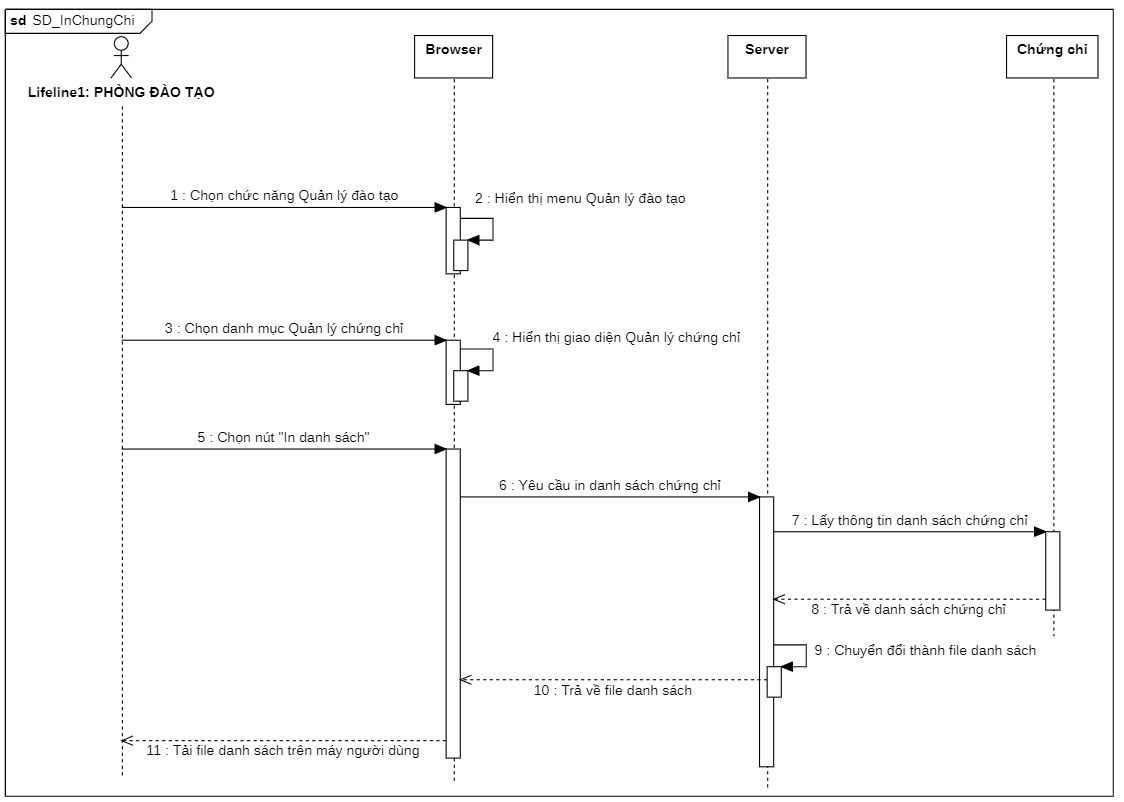
🌿**2. Xóa danh mục tổ chức**



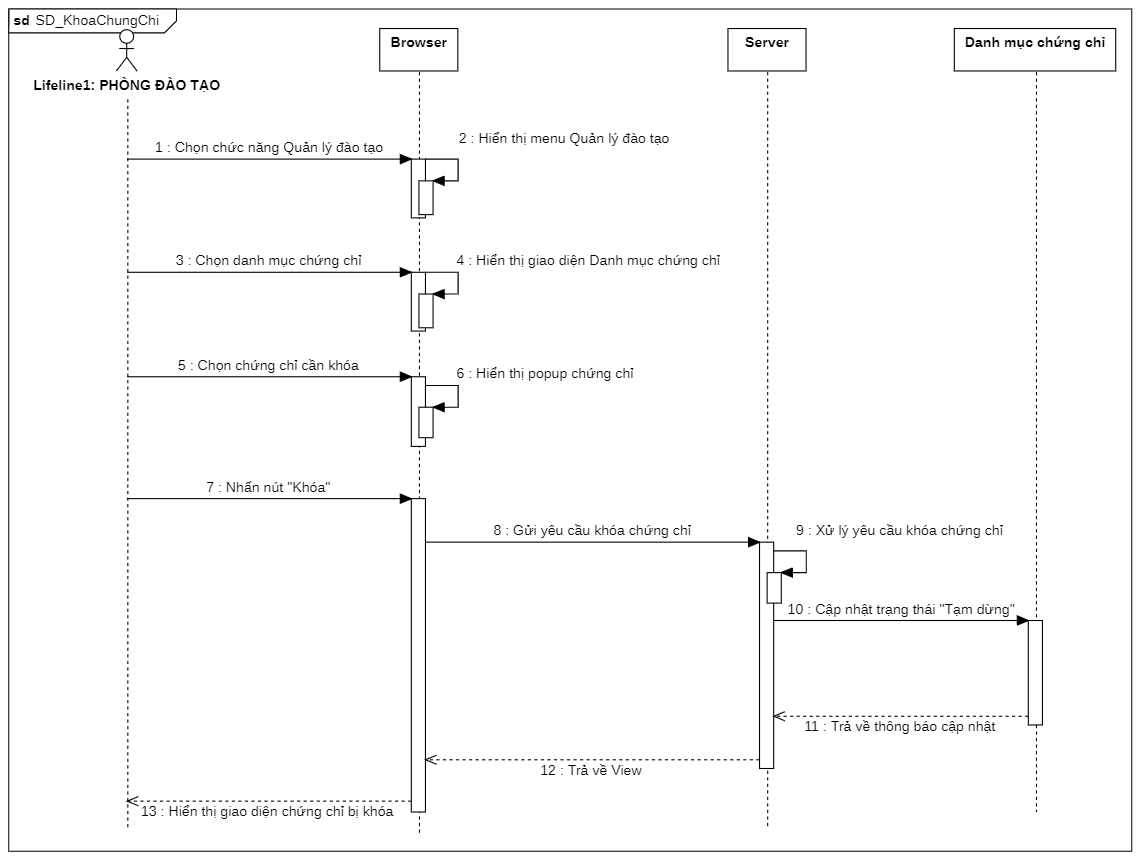
🌿**3. Sửa danh mục tổ chức**



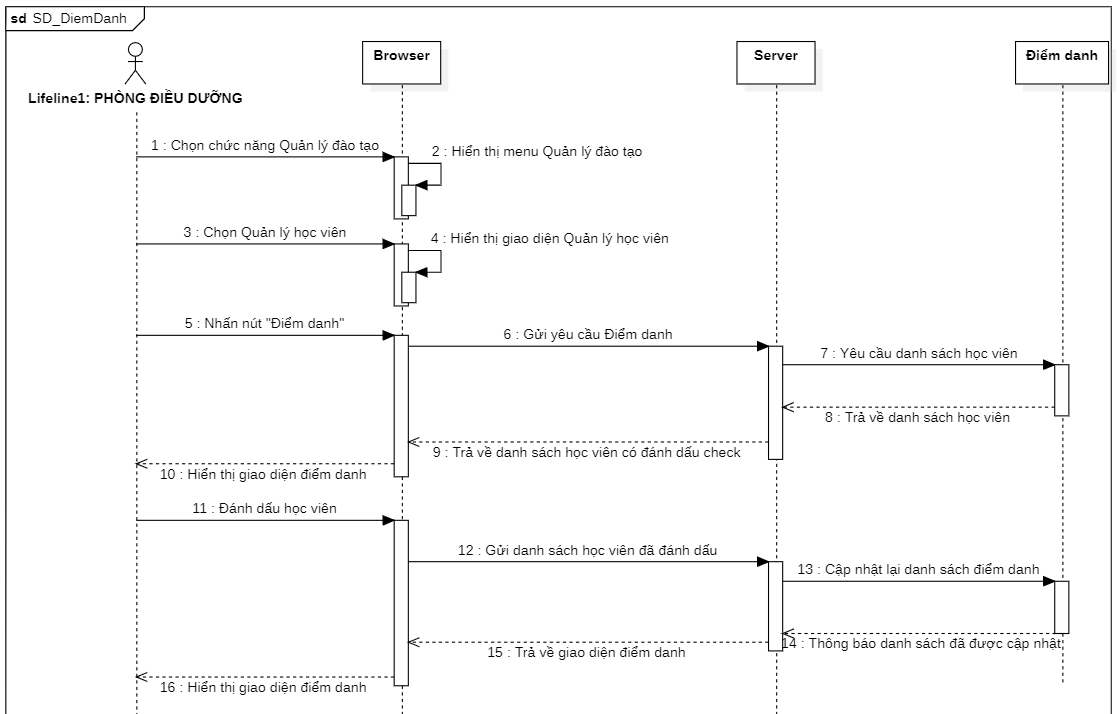
🌿**4. In chứng chỉ**



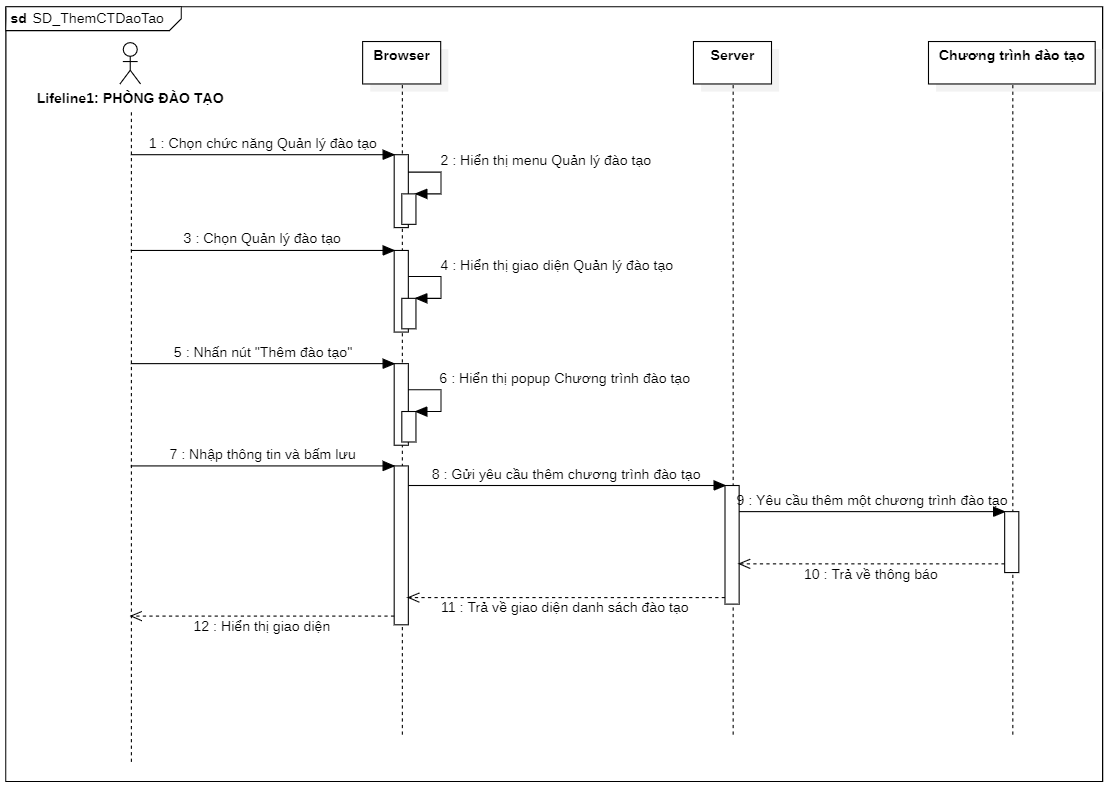
🌿**5. Khóa chứng chỉ**



🌿**6. Điểm danh**



🌿**7. Thêm chương trình đào tạo**



## [**TUẦN 3: 06/06/2022 - 12/06/2022**](https://github.com/hoangtuandev/report-vnptit)

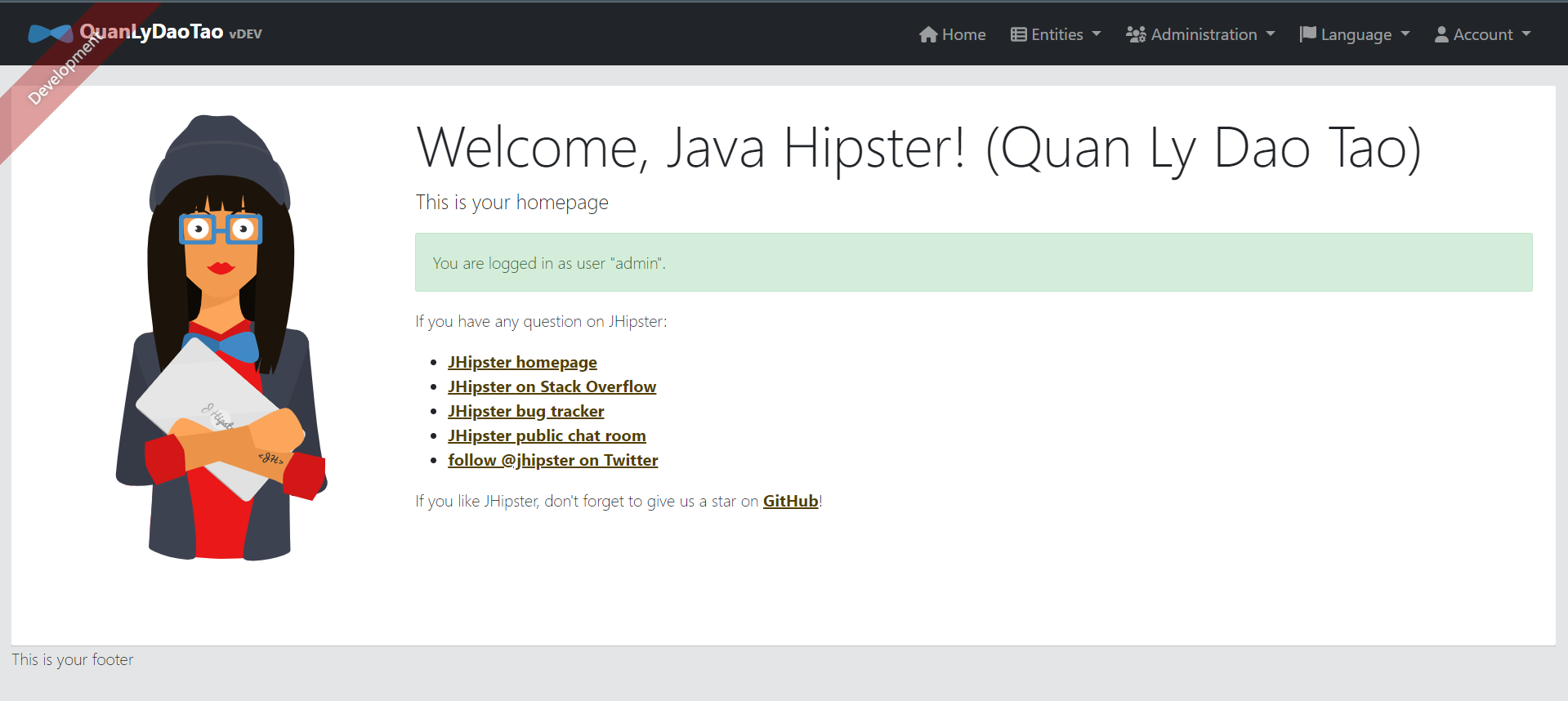
### ✏️ ***Câu 1: Tìm hiểu về Jhipster***

* [**Jhipster**](https://www.jhipster.tech/) là một nền tảng để tạo, phát triển và triển khai các ứng dụng web hiện đại và kiến trúc Microservice.
* Về frontend, chúng hỗ trợ nhiều công nghệ: Angular, React và Vue. Chúng thậm chí có hỗ trợ ứng dụng mobile với React Native.
* Về backend, chúng hỗ trợ Spring Boot (với Java và Kotlin), Micronaut, Quarkus, Node.js và .NET.
* Về triển khai, chúng nắm vững các nguyên tác hoạt động của đám mây với Docker và Kubernetes. Hỗ trợ triển khai trên AWS, Azure, Cloud Foundry, Google Cloud Platform, Heroku và Open Shift.

### ✏️ ***Câu 2: Angular trong dự án Jhipster***

* [**Angular**](https://angular.io/) được sử dụng để xây dựng giao diện người dụng trong dự án Jhipster
* Thư viện ng-jhipster chứa các chức năng tiện ích và các thành phần phổ biến.
* Angular CLI được sử dụng để xây dựng và thử nghiệm các ứng dụng JHipster.
* Cấu trúc dự án được sử dụng theo kiểu Angular.
* Sử dụng bộ định tuyến Angular để tổ chức các phần khác nhau của ứng dụng khách.

🌺🌺🌺[**Trước khi sử dụng Angular để thay đổi giao diện dự án**](https://angular.io/)🌺🌺🌺



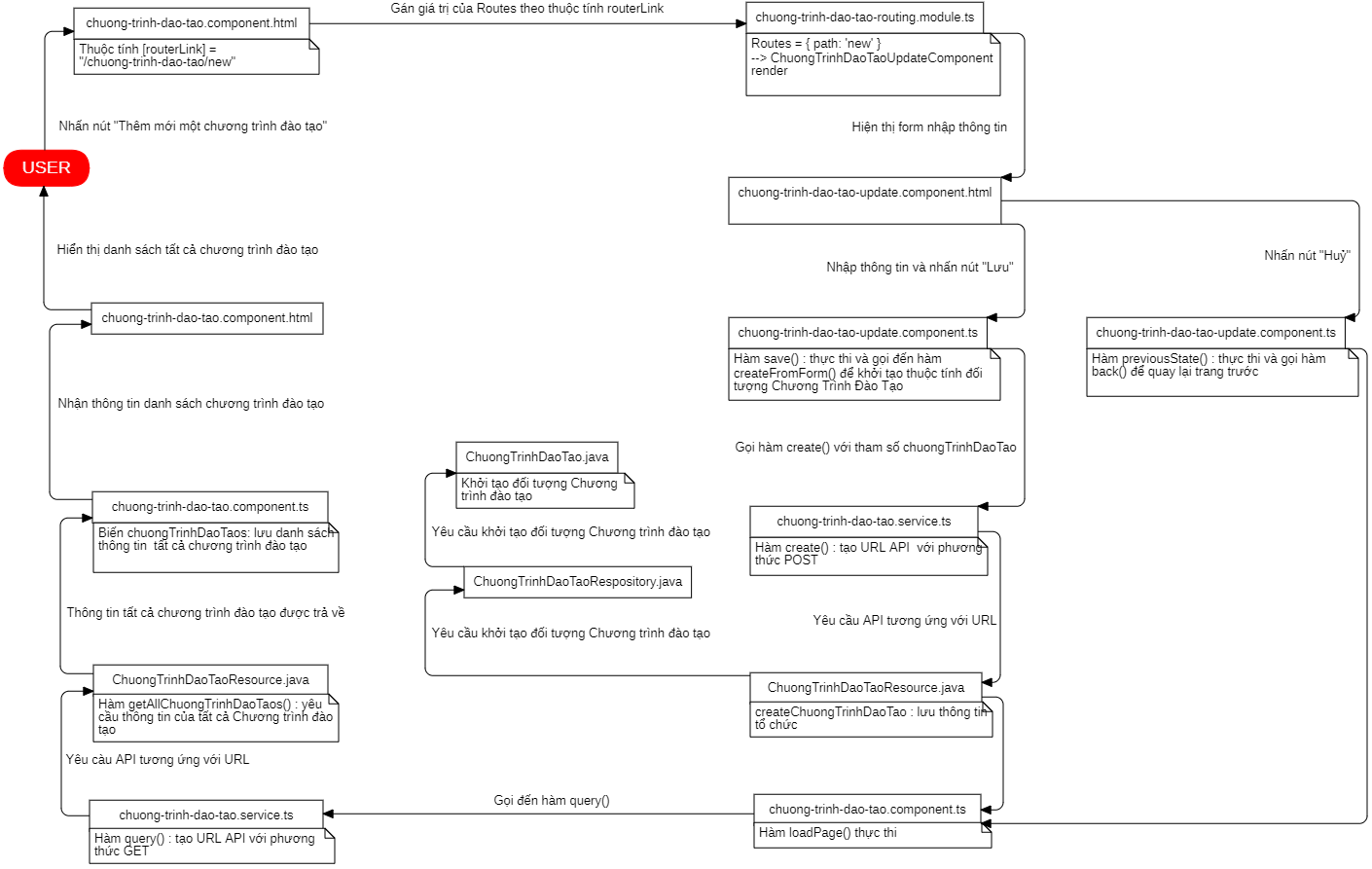
🌺🌺🌺[**Sau khi sử dụng Angular để thay đổi giao diện dự án**](https://angular.io/)🌺🌺🌺



## [**TUẦN 4: 13/06/2022 - 19/06/2022**](https://github.com/hoangtuandev/report-vnptit)

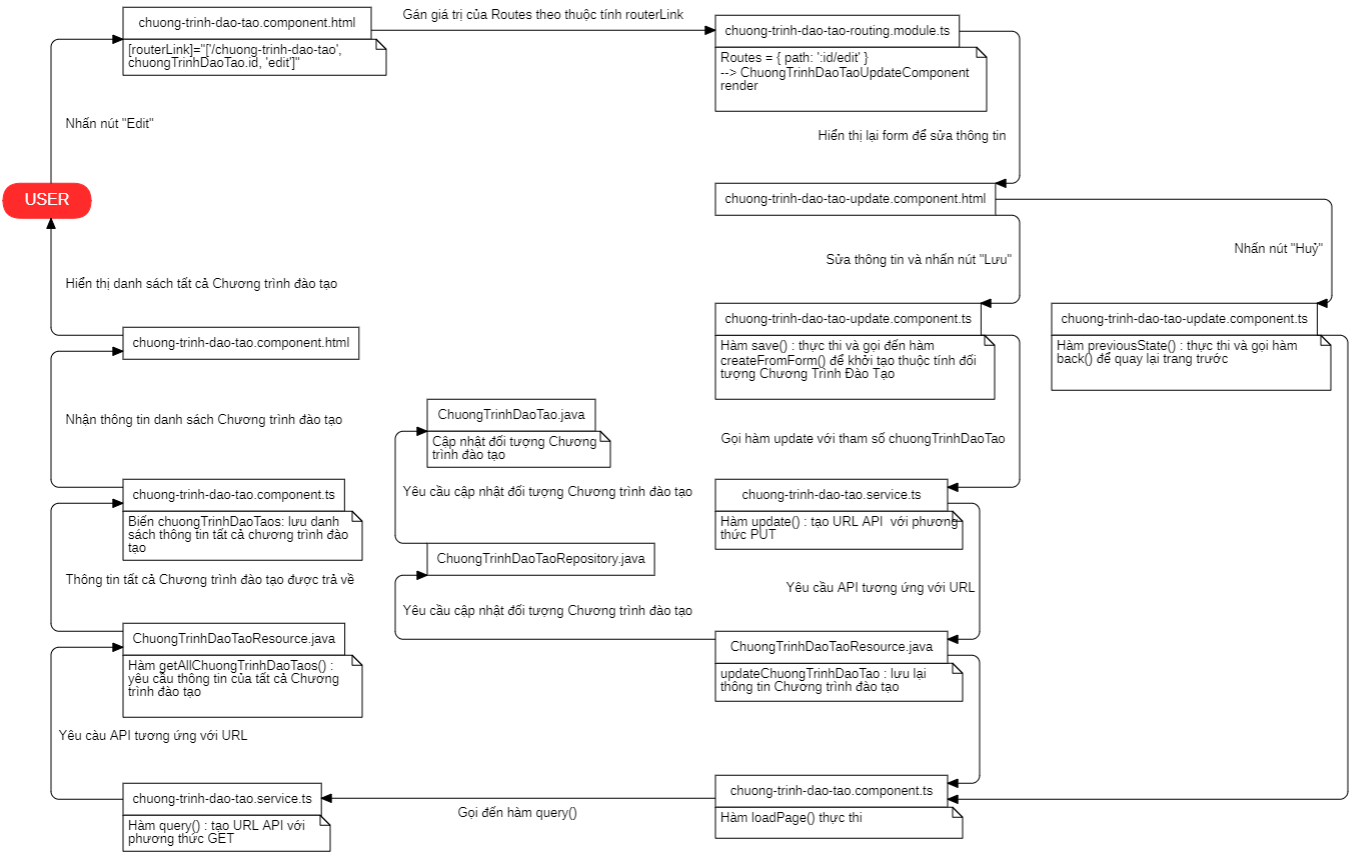
### 💐 **SƠ ĐỒ LUỒNG CÁC CHỨC NĂNG**

🌼**1. Chức năng “Thêm Chương Trình Đào Tạo”**



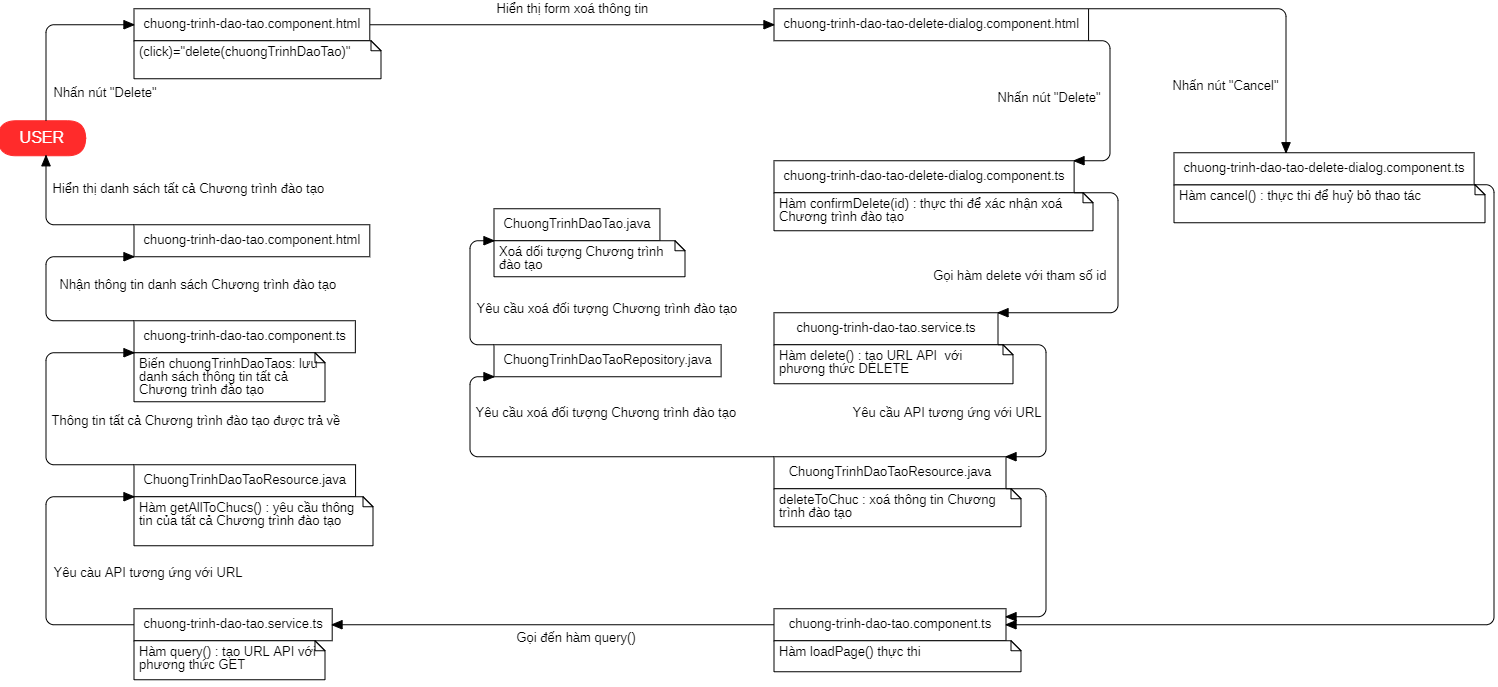
ThemChuongTrinhDaoTao.png

🌹**2. Chức năng “Sửa Chương Trình Đào Tạo”**



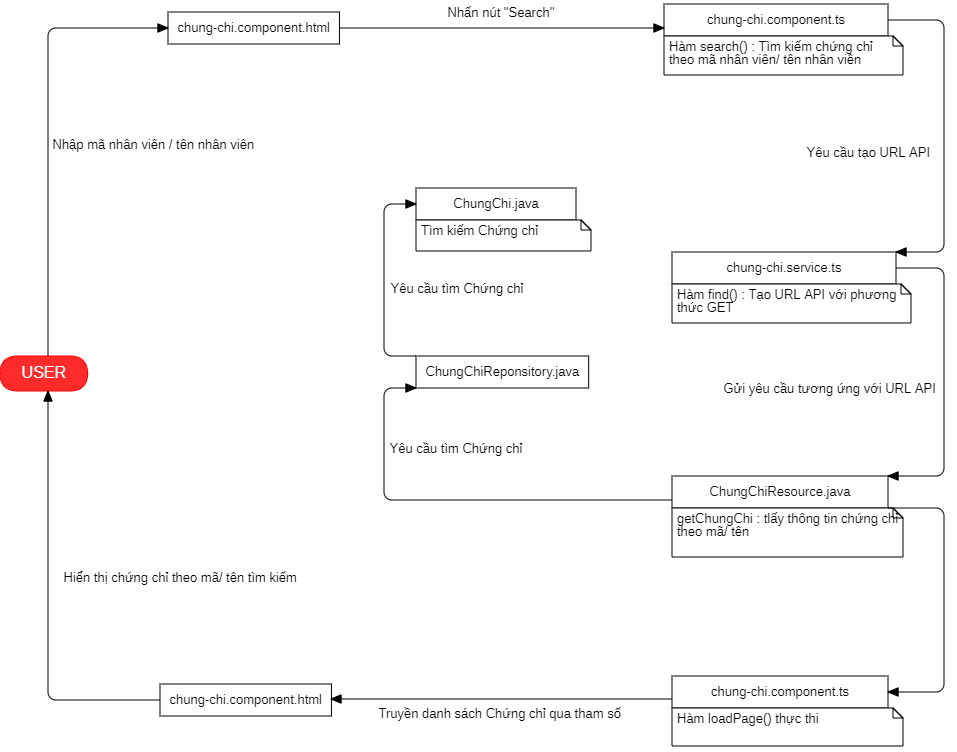
SuaChuongTrinhDaoTao.png

🌻**3. Chức năng “Xoá Chương Trình Đào Tạo”**



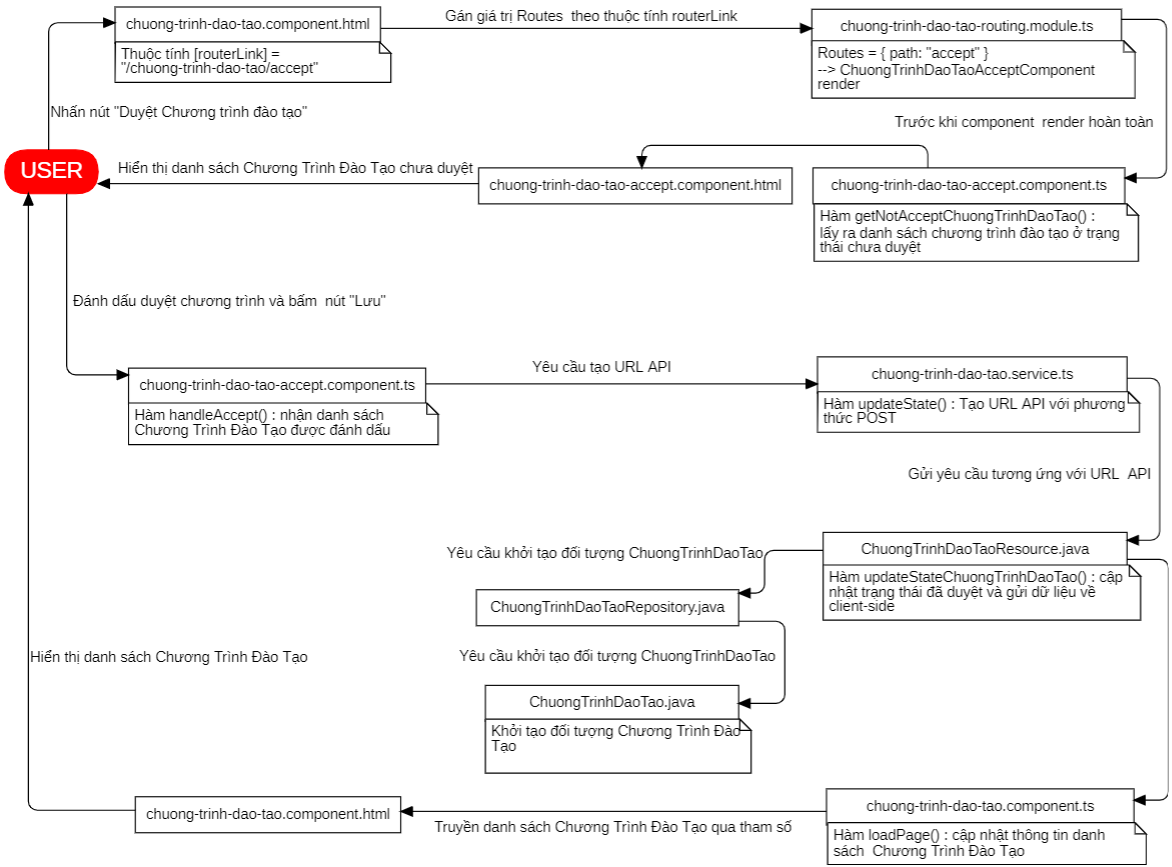
XoaChuongTrinhDaoTao.png

🌷**4. Chức năng “Tìm Kiếm Chứng Chỉ”**



TimKiemChungChi.png

🌸**5. Chức năng “Duyệt Chương Trình Đào Tạo”**



TimKiemChungChi.png

### ✏️ **Chức năng “Thêm mới Chương trình đào tạo”**

➡️ *Người dùng nhấn vào nút “Thêm mới Chương trình đào tạo”*

**src/main/webapp/app/entities/chuong-trinh-dao-tao/list/chuong-trinh-dao-tao.component.html**

 - Là giao diện chính của Quản lý Chương trình đào tạo - <button [routerLink]="/chuong-trinh-dao-tao/new">Thêm mới Chương trình Đào tạo</button> - Sử dụng giá trị “/chuong-trinh-dao-tao/new” của thuộc tính [routerLink] để làm đường dẫn cho component tiếp theo được render.

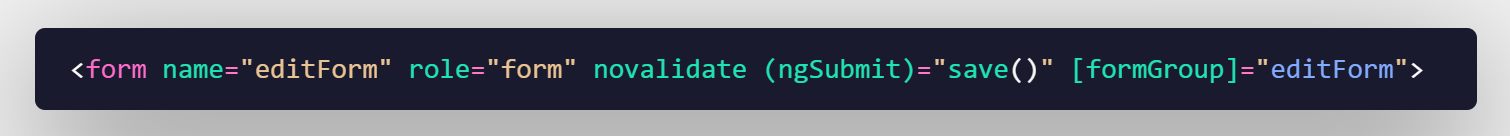
➡️ *Giá trị /chuong-trinh-dao-tao/new được sử dụng cho việc định tuyến*

**src/main/webapp/app/entities/chuong-trinh-dao-tao/list/chuong-trinh-dao-tao-routing.module.ts**

 - Là nơi chứa các Route, với mỗi “đường dẫn”(path) các component tương ứng sẽ được render - Với Routes = { path: '/chuong-trinh-dao-tao/new' }, component ChuongTrinhDaoTaoUpdate sẽ được render. - **NOTE:** canActivate

➡️ *Form nhập thông tin Chương trình đào tạo mới được hiển thị*

**src/main/webapp/app/entities/chuong-trinh-dao-tao/list/chuong-trinh-dao-tao-update.component.html**

 - Là giao diện form thêm mới/cập nhật thông tin của Chương trình đào tạo - <form (ngSubmit)="save()"></form>- Hàm save() sẽ được kích hoạt khi người dùng nhấn nút “Lưu”

➡️ *Người dùng nhập thông tin Chương trình đào tạo và nhấn nút “Lưu”*

**src/main/webapp/app/entities/chuong-trinh-dao-tao/list/chuong-trinh-dao-tao-update.component.ts**

 - Là nơi chứa các hàm xử lý các sự kiện(event), xử lý logic cho việc thêm mới/cập nhật thông tin Chương trình đào tạo - Hàm save() tạo một biến chuongTrinhDaoTao(object type) để lưu thông tin đối tượng và gọi đến hàm create() với tham số chuongTrinhDaoTao để hậu xử lý tạo mới Chương trình đào tạo.

➡️ *Yêu cầu dịch vụ create()*

**src/main/webapp/app/entities/chuong-trinh-dao-tao/sevice/chuong-trinh-dao-tao.service.ts**

 - Là nơi chứa các dịch vụ, phương thức dùng chung cho toàn bộ chức năng Quản lý Chương trình đào tạo - Hàm create() sử dụng phương thức POST của HTTP Method và yêu cầu khởi tạo một URL đến API Thêm mới Chương trình đào tạo

➡️ *Yêu cầu khởi tạo một URL dẫn đến API Thêm mới Chương trình đào tạo*

**src/main/webapp/app/core/config/application-config.service**

 - Nơi cấu hình các dịch vụ(service) cho toàn bộ ứng dụng - Hàm getEndpointFor() khởi tạo một URL dẫn đến API Thêm mới Chương trình đào tạo theo yêu cầu

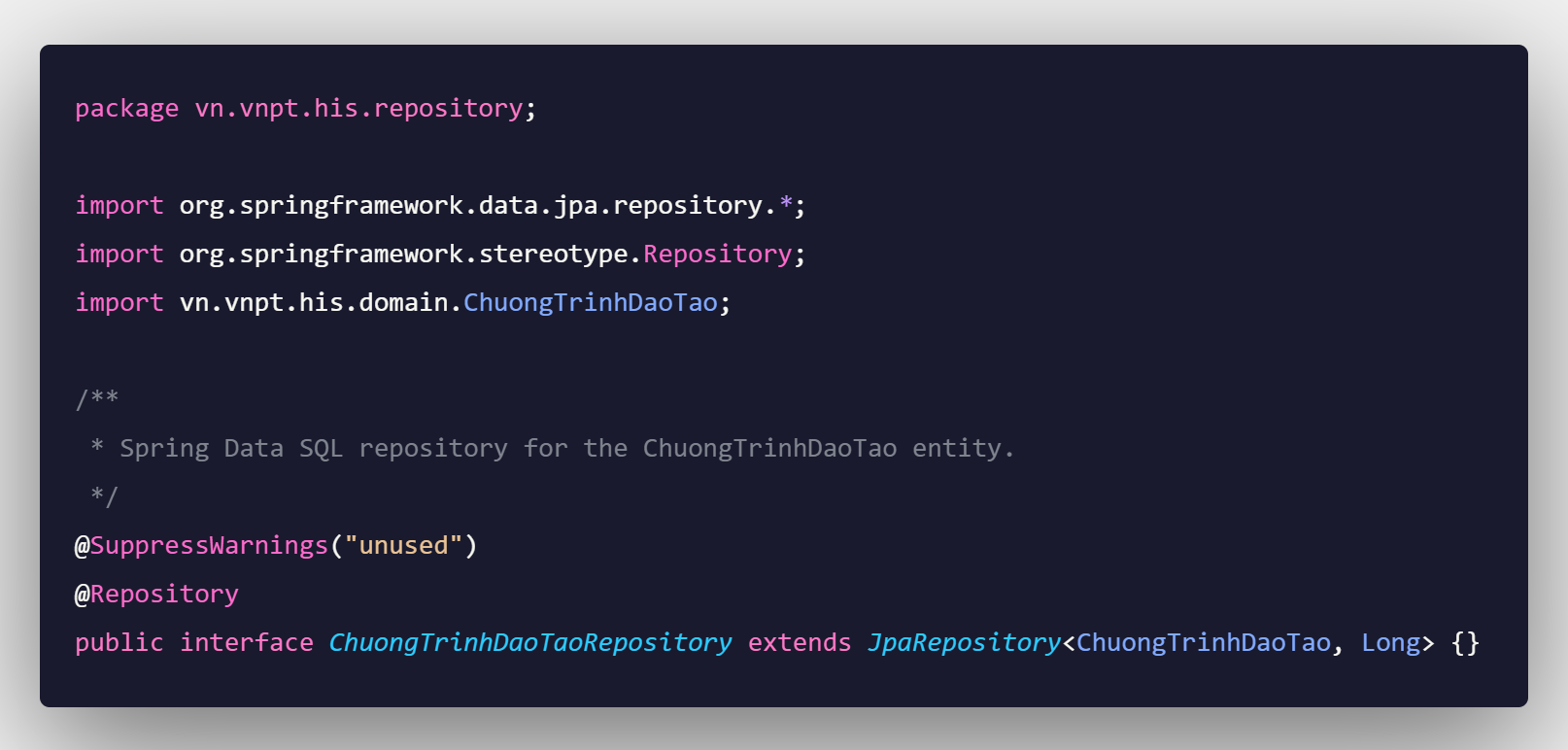
➡️ *Gửi yêu cầu Thêm mới Chương trình đào tạo tương ứng với URL đã được khởi tạo đến Server*

**src/main/java/vn/vnpt/his/web/rest/ChuongTrinhDaoTaoResource.java**

 - Nơi chứa các mã nguồn, các yêu cầu ánh xạ cho các phương thức dùng chung cho Quản lý Chương trình đào tạo - Hàm createChuongTrinhDaoTao() : tạo và lưu thông tin Chương trình đào tạo - @RequestBody: đây là nơi chứa data chính để gửi lên. Thường thì request body sẽ ở dạng JSON hoặc form-data, khi vào controller sẽ được tự động parse ra thành Object. - URI : xác định tài nguyên bởi chính xác nơi lấy nó hoặc tên của nó. - .created tạo link chứa URI ResponseEntity với trạng thái HTTP. Ví dụ như 201 là tạo thành công, còn 400 là Bad Request nếu id đã tồn tại. - .headers

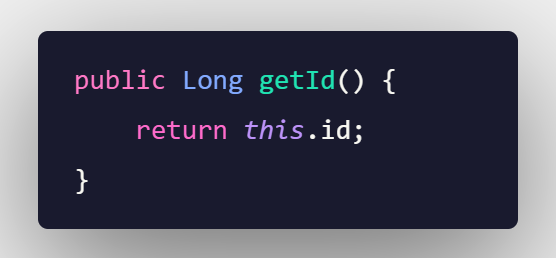
➡️ *Yêu cầu khởi tạo đối tượng Chương trình đào tạo*

**src/main/java/vn/vnpt/his/repository/ChuongTrinhDaoTaoRepository.java**

 - Kho lưu trữ Spring Data SQL cho thực thể Chương trình đào tạo. - @SuppressWarnings : Thông báo cho trình biên dịch biết là không được in các câu cảnh báo nào đó.  
- @Repository là để áp dụng trên các DAO (Data Access Object) class dùng để thao tác với database.

➡️ *Yêu cầu khởi tạo đối tượng Chương trình đào tạo*

**src/main/java/vn/vnpt/his/domain/ChuongTrinhDaoTao.java**

 - Khởi tạo đối tượng Chương trình đào tạo. - Chứa các thuộc tính của đối tượng Chương trình đào tạo. - getId() dùng để lấy giá trị thuộc tính id trong ChuongTrinhDaoTao.

➡️ *Client thực hiện tải lại trang web(Load Page)*

**src/main/webapp/app/entities/chuong-trinh-dao-tao/list/chuong-trinh-dao-tao.component.ts**

 - Là nơi chứa các hàm xử lý các sự kiện(event), xử lý logic chung cho Quản lý Chương trình đào tạo - Hàm loadPage() gọi đến dịch vụ query() yêu cầu lấy thông tin tất cả Chương trình đào tạo

➡️ *Yêu cầu dịch vụ query()*

**src/main/webapp/app/entities/chuong-trinh-dao-tao/sevice/chuong-trinh-dao-tao.service.ts**

 - Là nơi chứa các dịch vụ, phương thức dùng chung cho toàn bộ chức năng Quản lý Chương trình đào tạo - Hàm query() sử dụng phương thức GET của HTTP Method và yêu cầu khởi tạo một URL đến API lấy thông tin tất cả Chương trình đào tạo

➡️ *Yêu cầu khởi tạo một URL dẫn đến API lấy thông tin tất cả Chương trình đào tạo*

**src/main/webapp/app/core/config/application-config.service**

 - Nơi cấu hình các dịch vụ(service) cho toàn bộ ứng dụng - Hàm getEndpointFor() khởi tạo một URL dẫn đến API lấy thông tin tất cả Chương trình đào tạo theo yêu cầu

➡️ *Gửi yêu cầu lấy thông tin tất cả Chương trình đào tạo tương ứng với URL đã được khởi tạo đến Server*

**src/main/java/vn/vnpt/his/web/rest/ChuongTrinhDaoTaoResource.java**

 - Nơi chứa các mã nguồn, các yêu cầu ánh xạ cho các phương thức dùng chung cho Quản lý Chương trình đào tạo - Hàm getAllChuongTrinhDaoTaos() yêu cầu lấy thông tin của tất cả Chương trình đào tạo

➡️ *Thông tin tất cả Chương trình đào tạo được trả về Client*

**src/main/webapp/app/entities/chuong-trinh-dao-tao/list/chuong-trinh-dao-tao.component.ts**

 - Là nơi chứa các hàm xử lý các sự kiện(event), xử lý logic chung cho Quản lý Chương trình đào tạo - Biến chuongTrinhDaoTaos lưu thông tin tất cả Chương trình đào tạo - **NODE:** sort()

➡️ *Truyền thông tin tất cả Chương trình đào tạo cho việc hiển thị với người dùng*

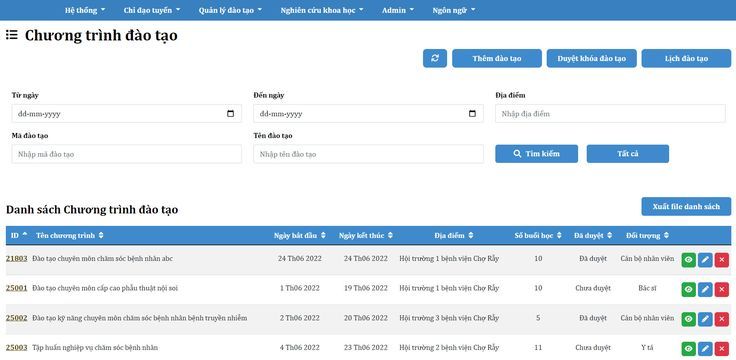
**src/main/webapp/app/entities/chuong-trinh-dao-tao/list/chuong-trinh-dao-tao.component.html**

 - Là giao diện chính của Quản lý Chương trình đào tạo - Sử dụng chuongTrinhDaoTao.[attributes] để truyền dữ liệu vào danh sách Chương trình đào tạo

➡️ *Danh sách thông tin tất cả Chương trình đào tạo được hiển thị trên giao diện người dùng*

## [**TUẦN 5, 6: 14/06/2022 - 03/07/2022**](https://github.com/hoangtuandev/report-vnptit)

### ☀️GIAO DIỆN QUẢN LÝ ĐÀO TẠO



➡️ Chương trình đào tạo sẽ quản lý danh sách các chương trình đào tạo bao gồm các chức năng như Thêm mới chương trình đào tạo, Cập nhật chương trình đào tạo, Xoá chương trình đào tạo, Xem thông tin chi tiết chương trình đào tạo, Lịch đào tạo, Duyệt khoá đào tạo, Tìm kiếm và Xuất file danh sách.

❄️ Chức năng [**Thêm mới chương trình đào tạo**](https://www.google.com)

[](https://i.pinimg.com/originals/b5/89/f4/b589f4e98a3422246a8997db2c43c61b.jpg

➡️ Thêm mới chương trình đào tạo bao gồm các thuộc tính Id, Tên chương trình, Ngày bắt đầu, Ngày kết thúc, Địa điểm, Số buổi học, Đã duyệt, Nội dung và có khoá ngoại là Đối tượng.

➡️ Trong đó, có hai thuộc tính là Tên chương trình và Đã duyệt là bắt buộc nhập. Thuộc tính Id là có sẵn và tự động tăng.

➡️ Ta chỉ cần nhập những dữ liệu cần thiết, sau đó nhấn vào nút Lưu. Ngược lại, nhấn vào nút Huỷ để huỷ bỏ thao tác.

❄️ Chức năng [**Cập nhật chương trình đào tạo**](https://www.google.com)



➡️ Cập nhật chương trình đào tạo sẽ lấy lại các thuộc tính Tên chương trình, Ngày bắt đầu, Ngày kết thúc, Địa điểm, Số buổi học, Đã duyệt, Nội dung và Đối tượng từ Chương trình đào tạo.

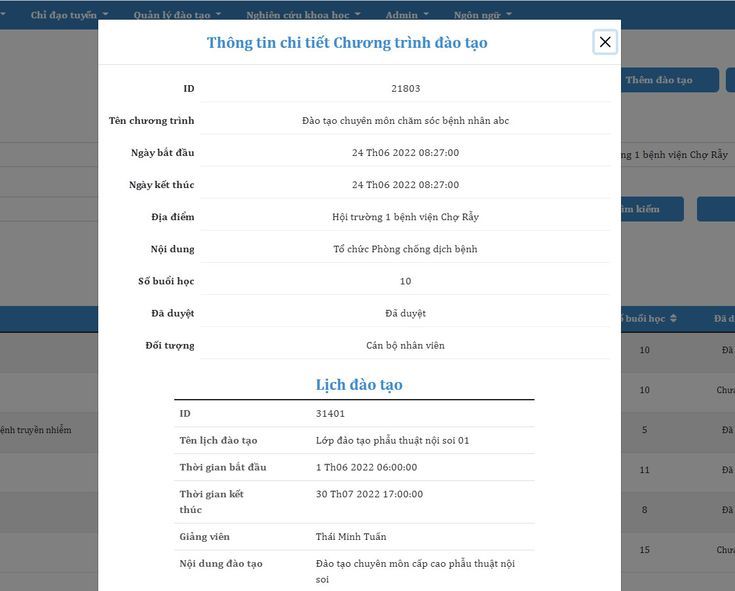
➡️ Khi cần cập nhật lại thuộc tính nào đó thì ta chỉ cần xoá và nhập lại, sau đó nhấn vào nút Lưu. Ngược lại, nhấn vào nút Huỷ để huỷ bỏ thao tác.

❄️ Chức năng [**Xoá chương trình đào tạo**](https://www.google.com)



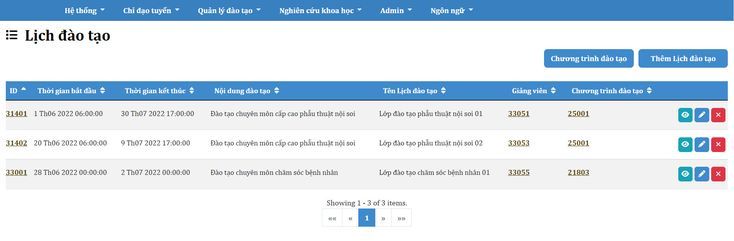
➡️ Khi click vào icon ❌ thì sẽ hiển thị thông báo xác nhận xoá chương trình đào tạo. Nếu muốn xoá sẽ click vào nút Xoá, còn không muốn xoá nữa sẽ click vào nút Huỷ.

❄️ Chức năng [**Xem thông tin chi tiết chương trình đào tạo**](https://www.google.com)



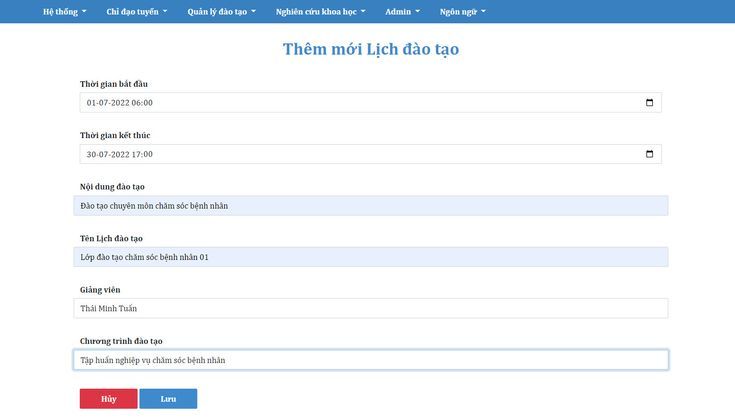
➡️ Khi click vào icon 👀 thì sẽ hiển thị một form chứa đầy đủ thông tin của một Chương trình đào tạo. Khi bạn muốn thoát chỉ cần click vào nút Đóng.

❄️ Chức năng [**Lịch đào tạo**](https://www.google.com)



➡️ Hiển thị danh sách lịch đào tạo.

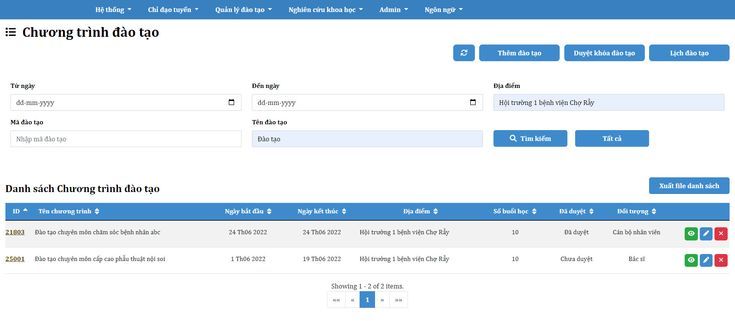
❄️ Chức năng [**Thêm mới Lịch đào tạo**](https://www.google.com)



➡️ Lịch đào tạo có các thuộc tính Thời gian bắt đầu, Thời gian kết thúc, Nội dung đào tạo, Tên Lịch đào tạo, Giảng viên, Chương trình đào tạo.

➡️ Trong thuộc tính Chương trình đào tạo, ta chỉ chọn được các Chương trình đào tạo Đã được duyệt.

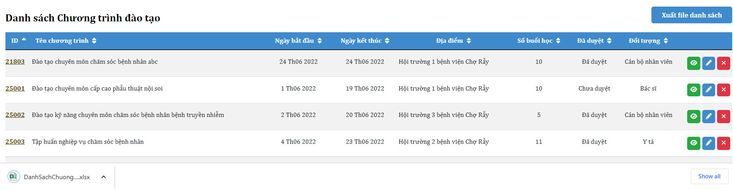
❄️ Chức năng [**Tìm kiếm**](https://www.google.com)



➡️ Tìm kiếm bao gồm các thuộc tính Từ ngày, Đến ngày, Địa điểm, Mã đào tạo và Tên đào tạo. Khi Tìm kiếm, chúng ta có thể nhập dữ liệu cho một hoặc nhiều thuộc tính để tiến hành tìm kiếm.

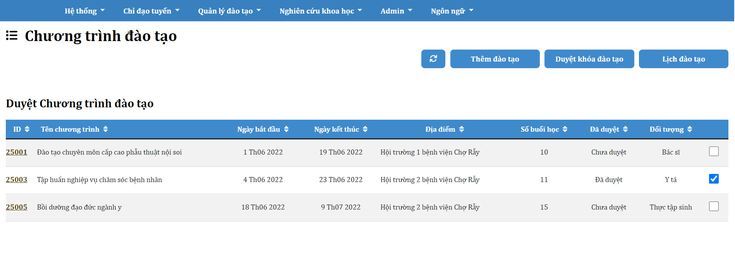
➡️ Nếu muốn hiển thị lại danh sách chương trình đào tạo, ta sẽ click vào nút Tất cả thì sẽ hiển thị lại danh sách.

❄️ Chức năng [**Xuất file danh sách**](https://www.google.com)



➡️ Khi click vào nút Xuất file danh sách thì chúng ta sẽ xuất được một file excel với đầy đủ thông tin của tất cả chương trình đào tạo.

❄️ Chức năng [**Duyệt chương trình đào tạo**](https://www.google.com)



➡️ Khi click vào nút Duyệt khoá đào tạo thì sẽ hiển thị được danh sách chứa các chương trình đào tạo chưa được duyệt.

➡️ Muốn duyệt một chương trình đào tạo nào đó, ta chỉ cần check vào checkbox thì nó sẽ chuyển sang trạng thái là "Đã duyệt".

➡️ Nếu không muốn duyệt nữa, ta có thể check lại vào ô checkbox để loại bỏ thao tác và trạng thái sẽ được chuyển lại là "Chưa duyệt".

➡️ Sau đó ta chỉ cần click lại vào nút Duyệt khoá đào tạo là được.

### 🎃

### 🎐 **SWAGGER**

**Swagger** là một bộ công cụ mã nguồn mở để xây dựng OpenAPI specifications giúp chúng ta có thể thiết kế, xây dựng tài liệu và sử dụng REST APIs.

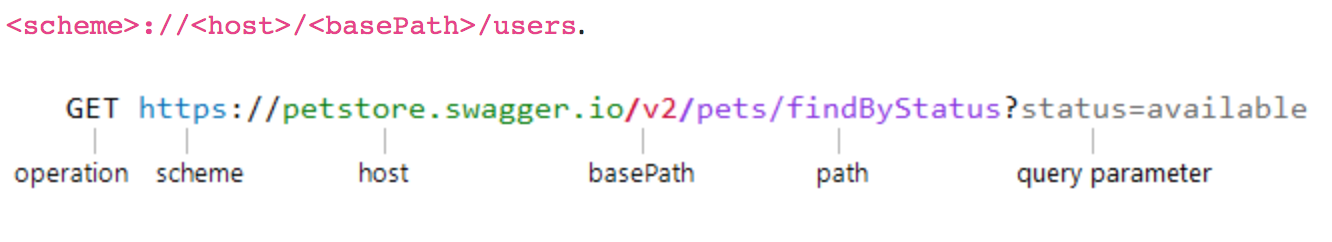
**Swagger** cung cấp 3 tools chính cho các developers : - ***Swagger-Editor*** : dùng để design lên các APIs hoàn toàn mới hoặc edit lại các APIs có sẵn thông qua 1 file config. - ***Swagger-Codegen*** : dùng để generate ra code từ các file config có sẵn - ***Swagger-UI*** : dùng để generate ra file html,css,… từ 1 file config.

**Cấu trúc cơ bản của file Swagger**

Đầu tiên 1 file swagger có thể viết bằng JSON hoặc YAML.

* ***Metadata:*** Mọi thông số kỹ thuật của Swagger đều bắt đầu với phiên bản Swagger . Phiên bản Swagger xác định cấu trúc tổng thể của đặc tả API - những gì bạn có thể ghi lại và cách bạn ghi lại nó. Ngoài ra các thông tin chi tiết như tiêu đề, mô tả hay version của bản api hiện tại cũng được khai báo tại đây.
* ***Base Url:*** Nơi bạn sẽ định nghĩa host của server, đường dẫn cơ bản cũng như giao thức https hoặc http.
* ***Consumes, Produces:*** xác định các loại MiME được API hỗ trợ.
* ***Paths:*** xác định các điểm cuối riêng lẻ (đường dẫn) trong API của bạn và các phương thức HTTP (hoạt động) được hỗ trợ bởi các điểm cuối này. Và đây là phần quan trọng chứa thông tin API của bạn sẽ như thế nào bằng đường dẫn API, phương thức (GET, POST, PUT…), request (query, path, body..), response API.

REST APIs có một URL cơ sở mà các đường dẫn điểm cuối được nối vào. Đường url này được định nghĩa bởi schema, host, basePath.

 - ***Schema*** là giao thức truyền được API sử dụng. Swagger hỗ trợ 2 giao thức là http và https. - ***host*** là tên miền hoặc địa chỉ IP (IPv4) của máy chủ lưu trữ cung cấp API. Nó có thể bao gồm số cổng nếu khác với cổng mặc định của lược đồ (80 cho HTTP và 443 cho HTTPS). - ***basePath*** là tiền tố URL cho tất cả các đường dẫn API, liên quan đến gốc máy chủ. Nó phải bắt đầu bằng một dấu gạch chéo /. Nếu basePath không được chỉ định, nó sẽ mặc định là /, nghĩa là, tất cả các đường dẫn đều bắt đầu từ máy chủ gốc.

### 🎍 **DTO** (Data Tranfer Object)

🎏 **DTO** là cấu trúc dữ liệu tối giản thường là class không có method chỉ dùng để truyền dữ liệu giữa các layer và service.

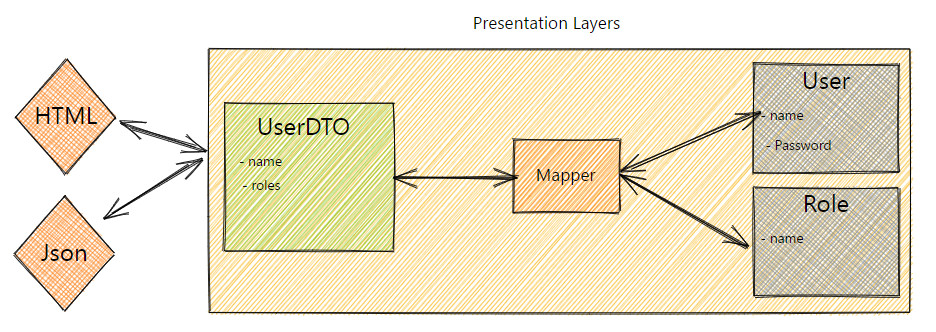
🎏 **DTO** có thể serialize ra JSON hoặc XML hoặc ngược lại từ JSON, XML deserialize thành DTO.

🎏 **DTO** có tác dụng che dấu hoặc lược bỏ những trường nhạy cảm trong table như password.

🎏 Hoặc đôi khi **DTO** là kết quả join từ nhiều bảng.

🎏 Mục đích tạo ra **DTO** là để giảm bớt lượng info không cần thiết phải chuyển đi, và cũng tăng cường độ bảo mật.

* Chúng ta cần phân biệt giữa ***Domain model*** và ***DTO*** để tránh nhầm lẫn. ***Domain model*** là các Entity class dùng để ánh xạ một table trong database còn ***DTO*** là một object kết hợp nhiều tham số thành một đặt trong một DTO class.

 ➡️ **DTO** là một cấu trúc dữ liệu phẳng và không chứa business logic trong đó chỉ dùng để lưu trữ dữ liệu, các method cho phép cập dữ liệu và sử dụng trong quá trình serialization or deserialization. Dữ liệu được ánh xạ từ domain model sang DTO và ngược lại thông qua một thành phần gọi là Mapper được đặt trong presentation hoặc facade layer.

### 🎋 **MAPPER**

Là lớp trung gian dùng để chuyển đổi qua lại giữa DTO và Domain model.

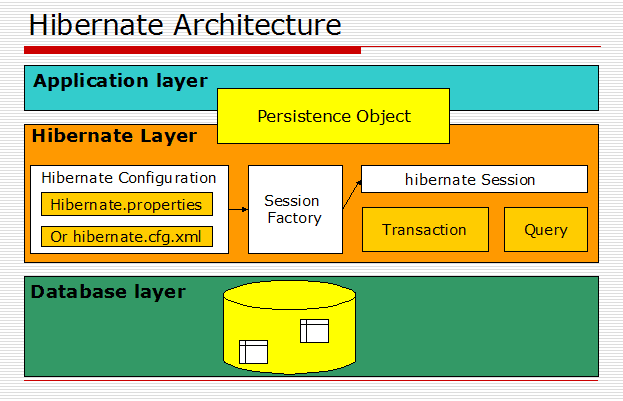
### 🎈 **DAO**

* DAO là Data Access Object. Một class có những method Create, Update, Delete, Query tác động lên một bảng trong CSDL.
* Tại sao lại có DAO bởi DAO giúp lập trình viên ở tầng Controller và Service không phải bận tâm viết câu lệnh SQL nữa. Hoặc ở một số framework DAO tự động chuyển đổi tên class, property thành câu lệnh SQL tương ứng.
* Nhờ có DAO mà chúng ta có thể viết logic một lần nhưng có thể triển khai trên nhiều loại database khác nhau.

### 🔆**HIBERNATE**

🔮 **Hibernate** là một thư viện ORM (Object Relational Mapping) mã nguồn mở giúp lập trình viên viết ứng dụng Java có thể map các objects (pojo) với hệ quản trị cơ sở dữ liệu quan hệ, và hỗ trợ thực hiện các khái niệm lập trình hướng đối tượng với cớ dữ liệu quan hệ.

🔮 Hiểu ngắn gọn thì Hibernate sẽ là một layer đứng trung gian giữa ứng dụng và database, và chúng ta sẽ giao tiếp với Hibernate thay vì giao tiếp với database

 ### **1. Domain** - Một Entity (Domain) là ánh xạ của 1 bảng ở db vào java, với @Column(name = "tencotdb") - Các thuộc tính đi kèm trong @Column như nullable, length sẽ được tự tạo khi file jhipster của entity được generate.

/\*\*  
 \* A ChuongTrinhDaoTao.  
 \*/  
@Entity // Đánh dấu đây là một Entity, chịu sự quản lý của Hibernate  
@Table(name = "chuong\_trinh\_dao\_tao") // Entity này đại diện cho table chuong\_trinh\_dao\_tao trong db  
@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)  
public class ChuongTrinhDaoTao implements Serializable {  
  
 private static final long serialVersionUID = 1L;  
  
 @Id // Đánh dấu biến ở dưới là primary key của table này  
 @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sequenceGenerator")  
 @SequenceGenerator(name = "sequenceGenerator")  
 @Column(name = "id") // trường id ở dưới đại diện cho cột id trong database  
 private Long id;  
  
 @NotNull // Thiết lập cột không được nhận giá trị NULL  
 @Column(name = "ten\_chuong\_trinh", nullable = false) // trường tenChuongTrinh ở dưới đại diện cho cột ten\_chuong\_trinh trong database  
 private String tenChuongTrinh;  
  
 @Column(name = "ngay\_bat\_dau")  
 private Instant ngayBatDau;  
  
 @Column(name = "ngay\_ket\_thuc")  
 private Instant ngayKetThuc;  
  
 @Column(name = "dia\_diem")  
 private String diaDiem;  
  
 @Column(name = "noi\_dung")  
 private String noiDung;  
  
 @Column(name = "so\_buoi\_hoc")  
 private Integer soBuoiHoc;  
  
 @NotNull  
 @Enumerated(EnumType.STRING) //sẽ chuyển đổi giá trị của enum sang string để lưu xuống database.  
 @Column(name = "da\_duyet", nullable = false)  
 private TrangThaiDuyet daDuyet;  
  
 @OneToMany(mappedBy = "chuongTrinhDaoTao")  
 @Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)  
 @JsonIgnoreProperties(value = { "chuongTrinhDaoTao", "danhMucChungChi", "nhanViens" }, allowSetters = true) // dùng để loại bỏ các thuộc tính  
 private Set<ChungChi> chungChis = new HashSet<>();

@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE) - ***READ\_WRITE:*** Cơ chế này đảm bảo tính nhất quán dữ liệu cao bằng việc sử dụng ‘soft lock`. Khi một thực thể đã được cache bị update, một ’soft lock’ được lưu lại trong cache cho entity và nó sẽ được giải phóng (release) khi transaction được commit. Tất cả các transaction nếu truy cập vào các đối tượng đang bị softblock sẽ được lấy trực tiếp từ cơ sở dữ liệu.

### **2. Respository**

/\*\*  
 \* Spring Data SQL repository for the ChuongTrinhDaoTao entity.  
 \*/  
@SuppressWarnings("unused")  
@Repository // Đánh dấu một Class Là tầng Repository, phục vụ truy xuất dữ liệu.  
public interface ChuongTrinhDaoTaoRepository extends JpaRepository<ChuongTrinhDaoTao, Long>, JpaSpecificationExecutor<ChuongTrinhDaoTao> {}

* @SuppressWarnings : Thông báo cho trình biên dịch biết là không được in các câu cảnh báo nào đó. Cụ thể ở đây là ***“unused”***.
* @Repository là để áp dụng trên các DAO (Data Access Object) class dùng để thao tác với database. Repository là 1 kho lưu trữ, truy xuất dữ liệu trung gian giữa các entity và database.

➡️ ChuongTrinhDaoTaoRepository là một interface, chỉ chứa các phương thức trừu tượng.

➡️ ChuongTrinhDaoTaoRepository kế thừa từ lớp JpaRepository và lớp JpaSpecificationExecutor.

### **3.Criteria**

* Criteria là 1 class sử dụng thư viên criteria do jhipster hỗ trợ. Các thuộc tính sẽ tương ứng với các thuộc tính trong DTo nhưng kiểu dữ liệu sẽ kèm theo Filer ví dụ: Long = LongFilter
* Các Filter sẽ hỗ trợ tùy theo như setEquals, setContains(StringFilter), greaterThanOrEquals(LocalDateFilter, ZonedDateTimeFilter)... các thuộc tính này sẽ được sử dụng trong phần createSpecification ở QueryService để build các câu truy vấn sẽ được đề cấp sau đây

/\*\*  
 \* Criteria class for the {@link vn.vnpt.his.domain.ChuongTrinhDaoTao} entity. This class is used  
 \* in {@link vn.vnpt.his.web.rest.ChuongTrinhDaoTaoResource} to receive all the possible filtering options from  
 \* the Http GET request parameters.  
 \* For example the following could be a valid request:  
 \* {@code /chuong-trinh-dao-taos?id.greaterThan=5&attr1.contains=something&attr2.specified=false}  
 \* As Spring is unable to properly convert the types, unless specific {@link Filter} class are used, we need to use  
 \* fix type specific filters.  
 \*/  
@ParameterObject  
public class ChuongTrinhDaoTaoCriteria implements Serializable, Criteria {  
  
 /\*\*  
 \* Class for filtering TrangThaiDuyet  
 \*/  
 public static class TrangThaiDuyetFilter extends Filter<TrangThaiDuyet> {  
  
 public TrangThaiDuyetFilter() {}  
  
 public TrangThaiDuyetFilter(TrangThaiDuyetFilter filter) {  
 super(filter);  
 }  
  
 @Override  
 public TrangThaiDuyetFilter copy() {  
 return new TrangThaiDuyetFilter(this);  
 }  
 }  
  
 private static final long serialVersionUID = 1L;  
  
 private LongFilter id;  
  
 private StringFilter tenChuongTrinh;  
  
 private InstantFilter ngayBatDau;  
  
 private InstantFilter ngayKetThuc;  
  
 private StringFilter diaDiem;  
  
 private StringFilter noiDung;  
  
 private IntegerFilter soBuoiHoc;  
  
 private TrangThaiDuyetFilter daDuyet;  
  
 private LongFilter chungChiId;  
  
 private LongFilter lichDaoTaoId;  
  
 private LongFilter phieuDanhGiaId;  
  
 private LongFilter phieuThanhToanHocPhiId;  
  
 private LongFilter duToanId;  
  
 private LongFilter doiTuongId;  
  
 private LongFilter hocVienId;  
  
 private Boolean distinct;  
  
 public ChuongTrinhDaoTaoCriteria() {}  
  
 public ChuongTrinhDaoTaoCriteria(ChuongTrinhDaoTaoCriteria other) {  
 this.id = other.id == null ? null : other.id.copy();  
 this.tenChuongTrinh = other.tenChuongTrinh == null ? null : other.tenChuongTrinh.copy();  
 this.ngayBatDau = other.ngayBatDau == null ? null : other.ngayBatDau.copy();  
 this.ngayKetThuc = other.ngayKetThuc == null ? null : other.ngayKetThuc.copy();  
 this.diaDiem = other.diaDiem == null ? null : other.diaDiem.copy();  
 this.noiDung = other.noiDung == null ? null : other.noiDung.copy();  
 this.soBuoiHoc = other.soBuoiHoc == null ? null : other.soBuoiHoc.copy();  
 this.daDuyet = other.daDuyet == null ? null : other.daDuyet.copy();  
 this.chungChiId = other.chungChiId == null ? null : other.chungChiId.copy();  
 this.lichDaoTaoId = other.lichDaoTaoId == null ? null : other.lichDaoTaoId.copy();  
 this.phieuDanhGiaId = other.phieuDanhGiaId == null ? null : other.phieuDanhGiaId.copy();  
 this.phieuThanhToanHocPhiId = other.phieuThanhToanHocPhiId == null ? null : other.phieuThanhToanHocPhiId.copy();  
 this.duToanId = other.duToanId == null ? null : other.duToanId.copy();  
 this.doiTuongId = other.doiTuongId == null ? null : other.doiTuongId.copy();  
 this.hocVienId = other.hocVienId == null ? null : other.hocVienId.copy();  
 this.distinct = other.distinct;  
 }  
 //Getter, setter, equals, hasCode, toString

### **4. QueryService**

/\*\*  
 \* Service for executing complex queries for {@link ChuongTrinhDaoTao} entities in the database.  
 \* The main input is a {@link ChuongTrinhDaoTaoCriteria} which gets converted to {@link Specification},  
 \* in a way that all the filters must apply.  
 \* It returns a {@link List} of {@link ChuongTrinhDaoTao} or a {@link Page} of {@link ChuongTrinhDaoTao} which fulfills the criteria.  
 \*/  
@Service //Đánh dấu một Class là tầng Service, phục vụ các logic nghiệp vụ.  
@Transactional(readOnly = true)  
public class ChuongTrinhDaoTaoQueryService extends QueryService<ChuongTrinhDaoTao> {  
  
 private final Logger log = LoggerFactory.getLogger(ChuongTrinhDaoTaoQueryService.class);  
  
 private final ChuongTrinhDaoTaoRepository chuongTrinhDaoTaoRepository;  
  
 public ChuongTrinhDaoTaoQueryService(ChuongTrinhDaoTaoRepository chuongTrinhDaoTaoRepository) {  
 this.chuongTrinhDaoTaoRepository = chuongTrinhDaoTaoRepository;  
 }  
  
 /\*\*  
 \* Return a {@link List} of {@link ChuongTrinhDaoTao} which matches the criteria from the database.  
 \* @param criteria The object which holds all the filters, which the entities should match.  
 \* @return the matching entities.  
 \*/  
 @Transactional(readOnly = true)  
 public List<ChuongTrinhDaoTao> findByCriteria(ChuongTrinhDaoTaoCriteria criteria) {  
 log.debug("find by criteria : {}", criteria);  
 final Specification<ChuongTrinhDaoTao> specification = createSpecification(criteria);  
 return chuongTrinhDaoTaoRepository.findAll(specification);  
 }  
 protected Specification<ChuongTrinhDaoTao> createSpecification(ChuongTrinhDaoTaoCriteria criteria) {  
 Specification<ChuongTrinhDaoTao> specification = Specification.where(null);  
 if (criteria != null) {  
 // This has to be called first, because the distinct method returns null  
 if (criteria.getDistinct() != null) {  
 specification = specification.and(distinct(criteria.getDistinct()));  
 }  
 if (criteria.getId() != null) {  
 specification = specification.and(buildRangeSpecification(criteria.getId(), ChuongTrinhDaoTao\_.id));  
 }  
 if (criteria.getTenChuongTrinh() != null) {  
 specification =  
 specification.and(buildStringSpecification(criteria.getTenChuongTrinh(), ChuongTrinhDaoTao\_.tenChuongTrinh));  
 }  
 if (criteria.getNgayBatDau() != null) {  
 specification = specification.and(buildRangeSpecification(criteria.getNgayBatDau(), ChuongTrinhDaoTao\_.ngayBatDau));  
 }  
 if (criteria.getNgayKetThuc() != null) {  
 specification = specification.and(buildRangeSpecification(criteria.getNgayKetThuc(), ChuongTrinhDaoTao\_.ngayKetThuc));  
 }  
 if (criteria.getDiaDiem() != null) {  
 specification = specification.and(buildStringSpecification(criteria.getDiaDiem(), ChuongTrinhDaoTao\_.diaDiem));  
 }  
 if (criteria.getNoiDung() != null) {  
 specification = specification.and(buildStringSpecification(criteria.getNoiDung(), ChuongTrinhDaoTao\_.noiDung));  
 }  
 if (criteria.getSoBuoiHoc() != null) {  
 specification = specification.and(buildRangeSpecification(criteria.getSoBuoiHoc(), ChuongTrinhDaoTao\_.soBuoiHoc));  
 }  
 if (criteria.getDaDuyet() != null) {  
 specification = specification.and(buildSpecification(criteria.getDaDuyet(), ChuongTrinhDaoTao\_.daDuyet));  
 }  
 if (criteria.getChungChiId() != null) {  
 specification =  
 specification.and(  
 buildSpecification(  
 criteria.getChungChiId(),  
 root -> root.join(ChuongTrinhDaoTao\_.chungChis, JoinType.LEFT).get(ChungChi\_.id)  
 )  
 );  
 }  
 if (criteria.getDoiTuongId() != null) {  
 specification =  
 specification.and(  
 buildSpecification(  
 criteria.getDoiTuongId(),  
 root -> root.join(ChuongTrinhDaoTao\_.doiTuong, JoinType.LEFT).get(DoiTuong\_.id)  
 )  
 );  
 }  
 if (criteria.getHocVienId() != null) {  
 specification =  
 specification.and(  
 buildSpecification(  
 criteria.getHocVienId(),  
 root -> root.join(ChuongTrinhDaoTao\_.hocViens, JoinType.LEFT).get(HocVien\_.id)  
 )  
 );  
 }  
 }  
 return specification;  
 }

* Specificaion là một cách để định nghĩa các Predicate(một mệnh đề điều kiện trong câu lệnh truy vấn.) có thể tái sử dụng được. Bản chất Specificaion là một function interface với 1 hàm duy nhất.
  + Specification.where() để xây dựng cho mình tập các điều kiện để query.
  + root: root là khai báo đối tượng bạn sẽ sử dụng trong query, tương đương với đối tượng sau mệnh đề FROM.
  + Một Specification<TenEntity> sẽ được sử dụng trong Repository với hàm findAll().
  + Có thể ghép nhiều Specification lại với nhau thông qua .and hoặc .or  
    ### **5. Service**

/\*\*  
 \* Service Interface for managing {@link ChuongTrinhDaoTao}.  
 \*/  
public interface ChuongTrinhDaoTaoService {  
 /\*\*  
 \* Save a chuongTrinhDaoTao.  
 \*  
 \* @param chuongTrinhDaoTao the entity to save.  
 \* @return the persisted entity.  
 \*/  
 ChuongTrinhDaoTao save(ChuongTrinhDaoTao chuongTrinhDaoTao);  
  
 /\*\*  
 \* Updates a chuongTrinhDaoTao.  
 \*  
 \* @param chuongTrinhDaoTao the entity to update.  
 \* @return the persisted entity.  
 \*/  
 ChuongTrinhDaoTao update(ChuongTrinhDaoTao chuongTrinhDaoTao);  
  
 /\*\*  
 \* Partially updates a chuongTrinhDaoTao.  
 \*  
 \* @param chuongTrinhDaoTao the entity to update partially.  
 \* @return the persisted entity.  
 \*/  
 Optional<ChuongTrinhDaoTao> partialUpdate(ChuongTrinhDaoTao chuongTrinhDaoTao);  
  
 /\*\*  
 \* Get all the chuongTrinhDaoTaos.  
 \*  
 \* @param pageable the pagination information.  
 \* @return the list of entities.  
 \*/  
 Page<ChuongTrinhDaoTao> findAll(Pageable pageable);  
  
 /\*\*  
 \* Get the "id" chuongTrinhDaoTao.  
 \*  
 \* @param id the id of the entity.  
 \* @return the entity.  
 \*/  
 Optional<ChuongTrinhDaoTao> findOne(Long id);  
  
 /\*\*  
 \* Delete the "id" chuongTrinhDaoTao.  
 \*  
 \* @param id the id of the entity.  
 \*/  
 void delete(Long id);  
}

**Service** là một interface, nó chỉ chứa các phương thức trừu tượng.

Tại đây chúng ta sẽ cần dùng đến ***Mapper*** để chuyển đổi từ DTO sang User. Tiến hành lưu xuống database và lại dùng Mapper để chuyển đổi User object đã được lưu xuống database sang DTO và trả về cho client.

### **6. ServiceImpl**

* ***ServiceImpl:*** kế thừa từ interface Service.

/\*\*  
 \* Service Implementation for managing {@link ChuongTrinhDaoTao}.  
 \*/  
@Service // Đánh dấu một Class là tầng Service, phục vụ các logic nghiệp vụ.  
@Transactional  
public class ChuongTrinhDaoTaoServiceImpl implements ChuongTrinhDaoTaoService {  
  
 private final Logger log = LoggerFactory.getLogger(ChuongTrinhDaoTaoServiceImpl.class);  
  
 private final ChuongTrinhDaoTaoRepository chuongTrinhDaoTaoRepository;  
  
 public ChuongTrinhDaoTaoServiceImpl(ChuongTrinhDaoTaoRepository chuongTrinhDaoTaoRepository) {  
 this.chuongTrinhDaoTaoRepository = chuongTrinhDaoTaoRepository;  
 }  
  
 @Override  
 public ChuongTrinhDaoTao save(ChuongTrinhDaoTao chuongTrinhDaoTao) {  
 log.debug("Request to save ChuongTrinhDaoTao : {}", chuongTrinhDaoTao);  
 return chuongTrinhDaoTaoRepository.save(chuongTrinhDaoTao);  
 }  
  
 @Override  
 public ChuongTrinhDaoTao update(ChuongTrinhDaoTao chuongTrinhDaoTao) {  
 log.debug("Request to save ChuongTrinhDaoTao : {}", chuongTrinhDaoTao);  
 return chuongTrinhDaoTaoRepository.save(chuongTrinhDaoTao);  
 }  
  
 @Override  
 public Optional<ChuongTrinhDaoTao> partialUpdate(ChuongTrinhDaoTao chuongTrinhDaoTao) {  
 log.debug("Request to partially update ChuongTrinhDaoTao : {}", chuongTrinhDaoTao);  
  
 return chuongTrinhDaoTaoRepository  
 .findById(chuongTrinhDaoTao.getId())  
 .map(existingChuongTrinhDaoTao -> {  
 if (chuongTrinhDaoTao.getTenChuongTrinh() != null) {  
 existingChuongTrinhDaoTao.setTenChuongTrinh(chuongTrinhDaoTao.getTenChuongTrinh());  
 }  
 if (chuongTrinhDaoTao.getNgayBatDau() != null) {  
 existingChuongTrinhDaoTao.setNgayBatDau(chuongTrinhDaoTao.getNgayBatDau());  
 }  
 if (chuongTrinhDaoTao.getNgayKetThuc() != null) {  
 existingChuongTrinhDaoTao.setNgayKetThuc(chuongTrinhDaoTao.getNgayKetThuc());  
 }  
 if (chuongTrinhDaoTao.getDiaDiem() != null) {  
 existingChuongTrinhDaoTao.setDiaDiem(chuongTrinhDaoTao.getDiaDiem());  
 }  
 if (chuongTrinhDaoTao.getNoiDung() != null) {  
 existingChuongTrinhDaoTao.setNoiDung(chuongTrinhDaoTao.getNoiDung());  
 }  
 if (chuongTrinhDaoTao.getSoBuoiHoc() != null) {  
 existingChuongTrinhDaoTao.setSoBuoiHoc(chuongTrinhDaoTao.getSoBuoiHoc());  
 }  
 if (chuongTrinhDaoTao.getDaDuyet() != null) {  
 existingChuongTrinhDaoTao.setDaDuyet(chuongTrinhDaoTao.getDaDuyet());  
 }  
  
 return existingChuongTrinhDaoTao;  
 })  
 .map(chuongTrinhDaoTaoRepository::save);  
 }  
  
 @Override  
 @Transactional(readOnly = true)  
 public Page<ChuongTrinhDaoTao> findAll(Pageable pageable) {  
 log.debug("Request to get all ChuongTrinhDaoTaos");  
 return chuongTrinhDaoTaoRepository.findAll(pageable);  
 }  
  
 @Override  
 @Transactional(readOnly = true)  
 public Optional<ChuongTrinhDaoTao> findOne(Long id) {  
 log.debug("Request to get ChuongTrinhDaoTao : {}", id);  
 return chuongTrinhDaoTaoRepository.findById(id);  
 }  
  
 @Override  
 public void delete(Long id) {  
 log.debug("Request to delete ChuongTrinhDaoTao : {}", id);  
 chuongTrinhDaoTaoRepository.deleteById(id);  
 }  
}

### **7. Resource(Controller)**

Tầng này sẽ nhận ***request*** từ client và chuyển xuống cho tầng Service xử lý.

/\*\*  
 \* REST controller for managing {@link vn.vnpt.his.domain.ChuongTrinhDaoTao}.  
 \*/  
@RestController  
@RequestMapping("/api")  
public class ChuongTrinhDaoTaoResource {  
  
 private final Logger log = LoggerFactory.getLogger(ChuongTrinhDaoTaoResource.class);  
  
 private static final String ENTITY\_NAME = "chuongTrinhDaoTao";  
  
 @Value("${jhipster.clientApp.name}")  
 private String applicationName;  
  
 private final ChuongTrinhDaoTaoService chuongTrinhDaoTaoService;  
  
 private final ChuongTrinhDaoTaoRepository chuongTrinhDaoTaoRepository;  
  
 private final ChuongTrinhDaoTaoQueryService chuongTrinhDaoTaoQueryService;  
  
 public ChuongTrinhDaoTaoResource(  
 ChuongTrinhDaoTaoService chuongTrinhDaoTaoService,  
 ChuongTrinhDaoTaoRepository chuongTrinhDaoTaoRepository,  
 ChuongTrinhDaoTaoQueryService chuongTrinhDaoTaoQueryService  
 ) {  
 this.chuongTrinhDaoTaoService = chuongTrinhDaoTaoService;  
 this.chuongTrinhDaoTaoRepository = chuongTrinhDaoTaoRepository;  
 this.chuongTrinhDaoTaoQueryService = chuongTrinhDaoTaoQueryService;  
 }  
  
 /\*\*  
 \* {@code POST /chuong-trinh-dao-taos} : Create a new chuongTrinhDaoTao.  
 \*  
 \* @param chuongTrinhDaoTao the chuongTrinhDaoTao to create.  
 \* @return the {@link ResponseEntity} with status {@code 201 (Created)} and with  
 \* body the new chuongTrinhDaoTao, or with status  
 \* {@code 400 (Bad Request)} if the chuongTrinhDaoTao has already an ID.  
 \* @throws URISyntaxException if the Location URI syntax is incorrect.  
 \*/  
 @PostMapping("/chuong-trinh-dao-taos")  
 public ResponseEntity<ChuongTrinhDaoTao> createChuongTrinhDaoTao(@Valid @RequestBody ChuongTrinhDaoTao chuongTrinhDaoTao)  
 throws URISyntaxException {  
 log.debug("REST request to save ChuongTrinhDaoTao : {}", chuongTrinhDaoTao);  
 if (chuongTrinhDaoTao.getId() != null) {  
 throw new BadRequestAlertException("A new chuongTrinhDaoTao cannot already have an ID", ENTITY\_NAME, "idexists");  
 }  
 ChuongTrinhDaoTao result = chuongTrinhDaoTaoService.save(chuongTrinhDaoTao);  
 return ResponseEntity  
 .created(new URI("/api/chuong-trinh-dao-taos/" + result.getId()))  
 .headers(HeaderUtil.createEntityCreationAlert(applicationName, true, ENTITY\_NAME, result.getId().toString()))  
 .body(result);  
 }  
  
 /\*\*  
 \* {@code PUT /chuong-trinh-dao-taos/:id} : Updates an existing  
 \* chuongTrinhDaoTao.  
 \*  
 \* @param id the id of the chuongTrinhDaoTao to save.  
 \* @param chuongTrinhDaoTao the chuongTrinhDaoTao to update.  
 \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body  
 \* the updated chuongTrinhDaoTao,  
 \* or with status {@code 400 (Bad Request)} if the chuongTrinhDaoTao is  
 \* not valid,  
 \* or with status {@code 500 (Internal Server Error)} if the  
 \* chuongTrinhDaoTao couldn't be updated.  
 \* @throws URISyntaxException if the Location URI syntax is incorrect.  
 \*/  
 @PutMapping("/chuong-trinh-dao-taos/{id}")  
 public ResponseEntity<ChuongTrinhDaoTao> updateChuongTrinhDaoTao(  
 @PathVariable(value = "id", required = false) final Long id,  
 @Valid @RequestBody ChuongTrinhDaoTao chuongTrinhDaoTao  
 ) throws URISyntaxException {  
 log.debug("REST request to update ChuongTrinhDaoTao : {}, {}", id, chuongTrinhDaoTao);  
 if (chuongTrinhDaoTao.getId() == null) {  
 throw new BadRequestAlertException("Invalid id", ENTITY\_NAME, "idnull");  
 }  
 if (!Objects.equals(id, chuongTrinhDaoTao.getId())) {  
 throw new BadRequestAlertException("Invalid ID", ENTITY\_NAME, "idinvalid");  
 }  
  
 if (!chuongTrinhDaoTaoRepository.existsById(id)) {  
 throw new BadRequestAlertException("Entity not found", ENTITY\_NAME, "idnotfound");  
 }  
  
 ChuongTrinhDaoTao result = chuongTrinhDaoTaoService.update(chuongTrinhDaoTao);  
 return ResponseEntity  
 .ok()  
 .headers(HeaderUtil.createEntityUpdateAlert(applicationName, true, ENTITY\_NAME, chuongTrinhDaoTao.getId().toString()))  
 .body(result);  
 }  
  
 /\*\*  
 \* {@code PATCH /chuong-trinh-dao-taos/:id} : Partial updates given fields of  
 \* an existing chuongTrinhDaoTao, field will ignore if it is null  
 \*  
 \* @param id the id of the chuongTrinhDaoTao to save.  
 \* @param chuongTrinhDaoTao the chuongTrinhDaoTao to update.  
 \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body  
 \* the updated chuongTrinhDaoTao,  
 \* or with status {@code 400 (Bad Request)} if the chuongTrinhDaoTao is  
 \* not valid,  
 \* or with status {@code 404 (Not Found)} if the chuongTrinhDaoTao is  
 \* not found,  
 \* or with status {@code 500 (Internal Server Error)} if the  
 \* chuongTrinhDaoTao couldn't be updated.  
 \* @throws URISyntaxException if the Location URI syntax is incorrect.  
 \*/  
 @PatchMapping(value = "/chuong-trinh-dao-taos/{id}", consumes = { "application/json", "application/merge-patch+json" })  
 public ResponseEntity<ChuongTrinhDaoTao> partialUpdateChuongTrinhDaoTao(  
 @PathVariable(value = "id", required = false) final Long id,  
 @NotNull @RequestBody ChuongTrinhDaoTao chuongTrinhDaoTao  
 ) throws URISyntaxException {  
 log.debug("REST request to partial update ChuongTrinhDaoTao partially : {}, {}", id, chuongTrinhDaoTao);  
 if (chuongTrinhDaoTao.getId() == null) {  
 throw new BadRequestAlertException("Invalid id", ENTITY\_NAME, "idnull");  
 }  
 if (!Objects.equals(id, chuongTrinhDaoTao.getId())) {  
 throw new BadRequestAlertException("Invalid ID", ENTITY\_NAME, "idinvalid");  
 }  
  
 if (!chuongTrinhDaoTaoRepository.existsById(id)) {  
 throw new BadRequestAlertException("Entity not found", ENTITY\_NAME, "idnotfound");  
 }  
  
 Optional<ChuongTrinhDaoTao> result = chuongTrinhDaoTaoService.partialUpdate(chuongTrinhDaoTao);  
  
 return ResponseUtil.wrapOrNotFound(  
 result,  
 HeaderUtil.createEntityUpdateAlert(applicationName, true, ENTITY\_NAME, chuongTrinhDaoTao.getId().toString())  
 );  
 }  
  
 /\*\*  
 \* {@code GET /chuong-trinh-dao-taos} : get all the chuongTrinhDaoTaos.  
 \*  
 \* @param pageable the pagination information.  
 \* @param criteria the criteria which the requested entities should match.  
 \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and the list  
 \* of chuongTrinhDaoTaos in body.  
 \*/  
 @GetMapping("/chuong-trinh-dao-taos")  
 public ResponseEntity<List<ChuongTrinhDaoTao>> getAllChuongTrinhDaoTaos(  
 ChuongTrinhDaoTaoCriteria criteria,  
 @org.springdoc.api.annotations.ParameterObject Pageable pageable  
 ) {  
 log.debug("REST request to get ChuongTrinhDaoTaos by criteria: {}", criteria);  
 Page<ChuongTrinhDaoTao> page = chuongTrinhDaoTaoQueryService.findByCriteria(criteria, pageable);  
 HttpHeaders headers = PaginationUtil.generatePaginationHttpHeaders(ServletUriComponentsBuilder.fromCurrentRequest(), page);  
 System.out.println(page.getContent());  
 return ResponseEntity.ok().headers(headers).body(page.getContent());  
 }  
  
 /\*\*  
 \* {@code DELETE /chuong-trinh-dao-taos/:id} : delete the "id"  
 \* chuongTrinhDaoTao.  
 \*  
 \* @param id the id of the chuongTrinhDaoTao to delete.  
 \* @return the {@link ResponseEntity} with status {@code 204 (NO\_CONTENT)}.  
 \*/  
 @DeleteMapping("/chuong-trinh-dao-taos/{id}")  
 public ResponseEntity<Void> deleteChuongTrinhDaoTao(@PathVariable Long id) {  
 log.debug("REST request to delete ChuongTrinhDaoTao : {}", id);  
 chuongTrinhDaoTaoService.delete(id);  
 return ResponseEntity  
 .noContent()  
 .headers(HeaderUtil.createEntityDeletionAlert(applicationName, true, ENTITY\_NAME, id.toString()))  
 .build();  
 }  
}

* @RestController: trả về dữ liệu dưới dạng JSON.
* @PathVariable: lấy ra thông tin trong URL, dựa vào tên của thuộc tính đã định nghĩa trong ngoặc kép {id}
* @RequestBody nói với Spring Boot rằng hãy chuyển Json trong request body thành đối tượng Todo.
* @PostMapping: có nhiệm vụ đánh dấu hàm xử lý POST request trong Controller. Một yêu cầu ***POST*** được sử dụng để gửi dữ liệu tới Server
* @GetMapping: có nhiệm vụ đánh dấu hàm xử lý GET request trong Controller. ***GET*** được sử dụng để lấy lại thông tin từ Server đã cung cấp bởi sử dụng một URI đã cung cấp. Các yêu cầu sử dụng GET chỉ nhận dữ liệu và không có ảnh hưởng gì tới dữ liệu.
* @PatchMapping: có nhiệm vụ đánh dấu hàm xử lý PATCH request trong Controller. ***PATCH*** thay đổi 1 phần các đại diện hiện tại của nguồn mục tiêu với nội dung được tải lên..
* @PutMapping: có nhiệm vụ đánh dấu hàm xử lý PUT request trong Controller. ***PUT*** thay đổi tất cả các đại diện hiện tại của nguồn mục tiêu với nội dung được tải lên.
* @DeleteMapping: có nhiệm vụ đánh dấu hàm xử lý DELETE request trong Controller. ***DELETE*** : gỡ bỏ tất cả các đại diện hiện tại của nguồn mục tiêu bởi URI.