**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



**ĐỒ ÁN TỐT NGHIỆP**

Đề tài

**Xây dựng trang quản lý sinh viên**

Giảng viên hướng dẫn: ThS. Phạm Xuân Tích

Sinh viên thực hiện: Trần Minh Hiệp

Mã sinh viên: 160704493

Lớp: CNTT1

Khóa: 57

**Hà Nội - 01/2021**

# LỜI CẢM ƠN

# MỤC LỤC

# DANH MỤC BẢNG BIỂU

# DANH MỤC HÌNH ẢNH

# MỞ ĐẦU

# CHƯƠNG 1: TỔNG QUAN

## Yêu cầu chức năng:

## Yêu cầu giao diện:

Giao diện cần được thiết kế đơn giản, thân thiện, dễ sử dụng. Có đầy đủ các trang, form để hiển thị đầy đủ các thông tin:

* Danh sách đơn hàng, sản phẩm, tài khoản của nhân viên.
* Trang/form chi tiết đơn hàng, sản phẩm, tài khoản.
* Trang/form thêm, sửa.
* Trang đăng nhập hệ thống.

## 1.3. Công nghệ sử dụng:

### 1.3.1. Ngôn ngữ lập trình Java:

**Java** là một nền tảng phát triển các ứng dụng [phần mềm](https://vi.wikipedia.org/wiki/Ph%E1%BA%A7n_m%E1%BB%81m) có vị trí rất lớn trong những năm cuối [thế kỉ 20](https://vi.wikipedia.org/wiki/Th%E1%BA%BF_k%E1%BB%B7_20), đầu [thế kỉ 21](https://vi.wikipedia.org/wiki/Th%E1%BA%BF_k%E1%BB%B7_21). Đánh dấu sự trưởng thành của mô hình lập trình hướng đối tượng, nó được coi là một nền tảng mang tính cách mạng trong ngành phần mềm. Mô hình máy ảo [Virtual Machine](https://vi.wikipedia.org/w/index.php?title=Virtual_Machine&action=edit&redlink=1) đã cho phép các ứng dụng viết bằng Java có thể chạy trên nhiều hệ điều hành khác nhau.

Ban đầu, Java chủ yếu dùng để phát triển các [applet](https://vi.wikipedia.org/w/index.php?title=Applet&action=edit&redlink=1), các ứng dụng nhúng vào trình duyệt, góp phần làm sinh động các [trang web](https://vi.wikipedia.org/wiki/Website) tĩnh vốn hết sức tẻ nhạt. Tuy nhiên, cùng với sự phát triển của công nghệ thông tin và nhu cầu của xã hội, Java applet đã dần mất đi vị trí của nó và thay vào đó, các công ty, cộng đồng ủng hộ Java đã phát triển nó theo một hướng khác. Hiện nay, công nghệ Java được chia làm ba nhánh:

* [**J2SE**](https://vi.wikipedia.org/wiki/J2SE)**:** Gồm các đặc tả, công cụ, [API](https://vi.wikipedia.org/wiki/Giao_di%E1%BB%87n_l%E1%BA%ADp_tr%C3%ACnh_%E1%BB%A9ng_d%E1%BB%A5ng) của nhân Java giúp phát triển các ứng dụng trên desktop và định nghĩa các phần thuộc nhân của Java.
* [**J2EE**](https://vi.wikipedia.org/wiki/J2EE)**:** Gồm các đặc tả, công cụ, API mở rộng J2SE để phát triển các ứng dụng quy mô xí nghiệp, chủ yếu để chạy trên máy chủ (server). Bộ phận hay được nhắc đến nhất của công nghệ này là công nghệ Servlet/[JSP](https://vi.wikipedia.org/wiki/JSP): sử dụng Java để làm các ứng dụng web.
* [**J2ME**](https://vi.wikipedia.org/wiki/J2ME)**:** Gồm các đặc tả, công cụ, API mở rộng để phát triển các ứng dụng Java chạy trên điện thoại di động, thẻ thông minh, thiết bị điện tử cầm tay, robo và những ứng dụng điện tử khác

Java bao gồm các bộ phận:

* Máy ảo Java: JVM
* Bộ công cụ phát triển: J2SDK
* Các đặc tả chi tiết kĩ thuật (specifications)
* Ngôn ngữ lập trình (programming language)
* Có 6 mục tiêu chính trong việc xây dựng ngôn ngữ Java:
* Đơn giản, hướng đối tượng và quen thuộc.
* Mạnh mẽ và an toàn.
* Kiến trúc trung lập và di động.
* Thực thi với hiệu suất cao.
* Dịch ra bytecode, phân luồng và năng động.
* Dễ sử dụng cho người dùng Java.

### 1.3.2. Spring framework:

Spring là một Framework phát triển các ứng dụng Java được sử dụng bởi hàng triệu lập trình viên. Nó giúp tạo các ứng dụng có hiệu năng cao, dễ kiểm thử, sử dụng lại code…

Spring nhẹ và trong suốt (nhẹ: kích thước nhỏ, version cơ bản chỉ khoảng 2MB; trong suốt: hoạt động một cách trong suốt với lập trình viên)

Spring là một mã nguồn mở, được phát triển, chia sẻ và có cộng đồng người dùng rất lơn.

Spring Framework được xây dựng dựa trên 2 nguyên tắc design chính là: [Dependency Injection](https://stackjava.com/design-pattern/dependency-injection-di-la-gi.html) và Aspect Oriented Programming.

Những tính năng core (cốt lõi) của Spring có thể được sử dụng để phát triển Java Desktop, ứng dụng mobile, Java Web. Mục tiêu chính của Spring là giúp phát triển các ứng dụng J2EE một cách dễ dàng hơn dựa trên mô hình sử dụng POJO (Plain Old Java Object)

Spring được chia làm nhiều module khác nhau, tùy theo mục đích phát triển ứng dụng mà ta dùng 1 trong các module đó.

Kiến trúc tổng thể của Spring Framework:



Hình 1.1. Mô hình kiến trúc Spring Framework

* **Test:** Tầng này cung cấp khả năng hỗ trợ kiểm thử với JUnit và TestNG.
* **Spring Core Container:** Bao gồm các module spring core, beans, context và expression languate (EL)
* Spring core, bean cung cấp tính năng [IOC và Dependency Injection](https://stackjava.com/design-pattern/dependency-injection-di-la-gi.html).
* Spring Context hỗ trợ đa ngôn ngữ (internationalization), các tính năng Java EE như EJB, JMX.
* Expression Language được mở rộng từ Expresion Language trong JSP. Nó cung cấp hỗ trợ việc setting/getting giá trị, các method cải tiến cho phép truy cập collections, index, các toán tử logic…
* **Web:** gồm Web, Web-Servlet… hỗ trợ việc tạo ứng dụng web, gọi là Spring MVC.
* **Data Access/Integration:** gồm JDBC, ORM, OXM, JMS và module Transaction, những module này cung cấp khả năng giao tiếp với database.
* **AOP, Aspects, Instrumentation:** Những module này hỗ trợ cài đặt lập trình hướng khía cạnh (Aspect Oriented Programming), hỗ trợ tích hợp với AspectJ.

Các lợi ích của Spring:

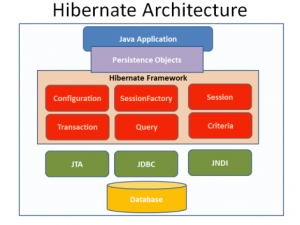
* Spring cho phép lập trình viên sử dụng POJOs. Việc sử dụng POJOs giúp không phải làm việc với EJB, ứng dụng, các luồng chạy, cấu hình… đơn giản hơn rất nhiều.
* Spring được tổ chức theo kiểu mô đun. Số lượng các gói và các lớp khá nhiều, nhưng chỉ cần quan tâm đến những gì cần và không cần quan tâm đến phần còn lại.
* Spring hỗ trợ sử dụng khá nhiều công nghệ như [ORM Framework](https://stackjava.com/uncategorized/orm-la-gi-tong-quan-ve-orm-framework.html), các logging framework, JEE, các thư viện tạo lịch trình (Quartz và JDK timer)…
* Module Web của Spring được thiết kế theo mô hình MVC nên nó cung cấp đầy đủ các tính năng giúp thay thế các web framework khác như Struts.

### 1.3.3. Hibernate Framework

Hibernate là 1 [ORM Framework](https://stackjava.com/uncategorized/orm-la-gi-tong-quan-ve-orm-framework.html): thực hiện mapping cơ sở dữ liệu quan hệ sang các object trong ngôn ngữ hướng đối tượng. Hibernate là 1 Framework cho persistence layer: Thực hiện giao tiếp giữa tầng ứng dụng với tầng dữ liệu (kết nối, truy xuất, lưu trữ…)

Kiến trúc hibernate framework:

* Mỗi table trong database là một object trong Hibernate. Do đó, bạn cần có một java bean cho mỗi table trong database. Các java bean này sẽ có các getters/setters và một số ngoại lệ theo quy ước của Hibernate.
* Mỗi file mapping (ánh xạ) có dạng \*\*\*.hbm.xml có nhiệm vụ đặc tả mỗi quan hệ giữa các thuộc tính của đối tượng và các trường trong bảng CSDL (cột nào ứng với thuộc tính nào của class, kiểu giá trị là gì, quan hệ giữa các bảng chuyển thành quan hệ giữa các lớp ra sao…).



Hình 1.2. Mô hình kiến trúc Hibernate Framework

* File Hibernate.cfg.xml: Đây là tập tin được load đầu tiên khi khởi chạy ứng dụng Hibernate. Nó chứa các thông tin về kết nối (database kết nối là gì (MySQL, Oracle, SQL server…), username và password kết nối với database là gì, tên database được kết nối), driver kết nối.

Ưu điểm của Hibernate Framework:

* Dễ sử dụng: dễ dàng quản lí các kết nối database và dễ fix bug, cung cấp sẵn nhiều API truy vấn.
* Tính độc lập: không cần quan tâm tới cơ sở dữ liệu sử dụng khi viết câu lệnh SQL.
* Tính hướng đối tượng: tập trung xử lý theo hướng đối tượng, phù hợp sử dụng trong các case CRUD (Create, Read, Update, Delete).
* Tính tin cậy: Hibernate đã được kiểm thử và khá an toàn trong các truy vấn.

Nhược điểm của Hibernate Framework:

* Không hỗ trợ các câu truy vấn phức tạp.
* Một số trường hợp vẫn phải dùng native SQL do Hibernate không thể cover hết tất cả các cú pháp của các hệ quản trị cơ sử dữ liệu.
* Bị hạn chế sự can thiệp vào câu lệnh SQL do nó được tự động sinh ra.

### 1.3.4. Hệ quản trị cơ sở dữ liệu MySql:

**MySQL** là [hệ quản trị cơ sở dữ liệu](https://vi.wikipedia.org/wiki/H%E1%BB%87_qu%E1%BA%A3n_tr%E1%BB%8B_c%C6%A1_s%E1%BB%9F_d%E1%BB%AF_li%E1%BB%87u) [mã nguồn mở](https://vi.wikipedia.org/wiki/Ph%E1%BA%A7n_m%E1%BB%81m_ngu%E1%BB%93n_m%E1%BB%9F) phổ biến nhất thế giới và được các nhà phát triển rất ưa chuộng trong quá trình phát triển ứng dụng. Vì MySQL là hệ quản trị cơ sở dữ liệu tốc độ cao, ổn định và dễ sử dụng, có tính khả chuyển, hoạt động trên nhiều hệ điều hành cung cấp một hệ thống lớn các hàm tiện ích rất mạnh. Với tốc độ và tính bảo mật cao, MySQL rất thích hợp cho các ứng dụng có truy cập CSDL trên internet.

Máy chủ cơ sở dữ liệu MySQL đáp ứng nhiều tính năng linh hoạt, nó có sức chứa để xử lý các ứng dụng được nhúng sâu với 1MB dung lượng để chạy kho dữ liệu khổng lồ lên đến hàng terabytes thông tin. Đặc tính này của MySQL là sự linh hoạt về flatform với tất cả các phiên bản của Windows, Unix và Linux đang được hỗ trợ. Và đương nhiên, tính chất mã nguồn mở của MySQL cho phép tùy biến theo ý muốn để thêm các yêu cầu phù hợp cho database server. Với kiến trúc storage-engine cho phép các chuyên gia cơ sở dữ liệu cấu hình máy chủ cơ sở dữ liệu MySQL đặc trưng cho các ứng dụng chuyên biệt. Dù ứng dụng là website dung lượng lớn phục vụ hàng triệu người/ngày hay hệ thống xử lý giao dịch tốc độ cao thì MySQL đều đáp ứng được khả năng xử lý khắt khe của mọi hệ thống. MySQL đảm bảo sự tin cậy và có thể sử dụng ngay. MySQL đưa ra nhiều tùy chọn có thể dùng ngay từ cấu hình tái tạo server/client tốc độ cao, để các nhà phân phối thứ 3 có thể đưa ra những điều hướng có thể dùng ngay duy nhất cho server cơ sở dữ liệu MySQL. MySQL là nơi cho các website trao đổi thường xuyên bởi nó có engine xử lý tốc độ cao, khả năng chèn dữ liệu cực nhanh và hỗ trợ mạnh các chức năng chuyên dụng của web. Các tính năng này cũng được sử dụng cho môi trường lưu trữ dữ liệu mà MySQL tăng cường đến hàng terabyte cho các server đơn. Những tính năng khác như chỉ số băm, bảng nhớ chính, bảng lưu trữ và cây nhị phân đã được gói lại để giảm các yêu cầu lưu trữ. MySQL tích hợp các tính năng bảo mật an toàn tuyệt đối. Với việc xác nhận truy cập cơ sở dữ liệu, MySQL trang bị các kỹ thuật mạnh, chỉ có người sử dụng đã được xác nhận mới truy cập được vào cơ sở dữ liệu. SSH và SSL cũng được hỗ trợ để đảm bảo kết nối an toàn và bảo mật. Tiện ích backup và recovery cung cấp bởi MySQL và các hãng phần mềm thứ 3 cho phép backup logic và vật lý cũng như recovery toàn bộ hoặc tại một thời điểm.

### 1.3.5. Angular:

Angular là một Javascript Framework được phát triển bởi google để xây dựng các Single Page Application (SPA) bằng JavaScript, HTML và TypeScript. Angular cung cấp các tính năng tích hợp cho animation, http service, materials và có các tính năng như auto-complete, navigation, toolbar, menus, … Code được viết bằng TypeScript, biên dịch thành JavaScript và hiển thị giống nhau trong trình duyệt.

Phiên bản đầu tiền của angular là AngularJS được bắt đầu từ năm 2009 và đc ra mắt vào 20/10/2010 bởi lập trình viên Misko Hevery tại Google. Lúc đó angular js được viết theo mô hình MVC. Tháng 3 năm 2015 phiên bản bản angular 2 ra đời nhằm thay thế Angular Js với các khái niệm mới nhằm đơn giản hóa và tối ưu cho quá trình phát triển sử dụng framework này. Angular 2 thay đổi hoàn toàn so với angular js bằng việc thay Controllers và $scope (Angular js) bằng components và directives. Components = directives + template, tạo nên view của ứng dụng và xử lí các logic trên view. Angular 2 hoàn toàn được viết bằng Typescript. Angular 2 nhanh hơn angular js, hỗ trợ đa nền tảng đa trình duyệt, cấu trúc cdoe được tổ chức đơn giản và dễ sử dụng hơn. Angular 4 ra mắt vào tháng 3/2017 đây là một phiên bản nâng cấp từ Angular 2 nên kiến trúc không thay đổi nhiều ngoài việc giảm thiểu code được tạo ra từ đó giảm kích thước tệp được đóng gói, đẩy nhanh quá trình phát triển ứng dụng. Angular 5 được phát hành vào 1/11/2017 với mục tiêu thay đổi về tốc độ và kích thước nên nó nhanh hơn và nhỏ hơn angular 4. Các tính năng mới so với angular 4:

* **Sử dụng HTTPClient**thay vì sử dụng HTTP: bởi vì nó nhanh, an toàn và hiệu quả hơn.
* Với phiên bản Angular 5 mặc định sử dụng RxJs 5.5
* **Multiple export aliases**: Một component có thể được xuất bằng nhiều bí danh (aliases) để giảm bớt quá trình di chuyển.
* **Internationalized Pipes for Number, Date, and Currency:** Các pipe mới được giới thiệu để tiêu chuẩn hóa tốt hơn.
* Tối ưu hóa build production bằng việc sử dụng công cụ build optimizer được tích hợp sẵn vào trong CLI. Công cụ này tối ưu tree shark và loại bỏ code dư thừa.
* Cải thiện tốc độ biên dịch bằng việc dùng TypeScript transforms, giờ đây khi build sẽ sử dụng lệnh “ng serve –aot”. AOT sẽ cải thiện performace khi load page và nó được dùng để deploy app lên production.

Angular 6 được phát hành vào 4/5/2018 với những thay đổi so với Angular 5:

* **Cập nhật CLI, command line interface**: thêm 1 số lệnh mới như ng-update để chuyển từ version trước sang version hiện tại; ng-add để thêm các tính năng của ứng dụng để trở thành một ứng dụng web tiến bộ.
* **Angular Element:** Cho phép các component của **Angular** được triển khai dưới dạng component web, sau đó có thể được sử dụng trong bất kỳ trang HTML nào một cách dễ dàng.
* **Multiple Validators:** cho phép nhiều Validators được áp dụng trên form builder.
* **Tree-shakeable providers:** giúp loại bỏ mã code chết.
* Sử dụng RxJS 6 với syntax thay đổi.

Angular 7 được phát hành vào 18/10/2018 với những thay đổi như:

* Hỗ trợ TypeScript 3.1
* ScrollingModule: Để scroll load dữ liệu.
* Drag and Drop: Chúng ta có thể dễ dàng thêm tính năng kéo và thả vào một mục.
* Angular 7.0 đã cập nhật RxJS 6.3.

Angular 8 được phát hành vào 3/4/2019, được nâng cấp hỗ trợ TypeScript 3.4.

Angular được load với tính năng Power-packaged. Một số tính năng như:

* Cơ chế Two-Way Data Binding: Đây là tính năng cool nhất của Angular. Data binding tự động và rất nhanh tức là bất cứ thay đổi nào trên view đều được tự động cập nhật vào component class và ngược lại.
* Hỗ trợ cơ chế Routing mạnh mẽ: Angular có cơ chế routing tải trang một cách bất đồng bộ trên cùng một trang cho phép chúng ta tạo SPA.
* Mở rộng HTML: Angular cho phép chúng ta sử dụng cấu trúc lập trình giống như điều kiện if, vòng lặp for...để render các control.
* Thiết kế module hoá: Angular tiếp cận theo hướng thiết kế module hoá. Bạn phải tạo các Angular Module để tổ chức tốt hơn và quản lý source code.
* Hỗ trợ làm việc với hệ thống Backend: Angular được xây dựng hỗ trợ làm việc với backend server và thực thi bất cứ logic nào và nhận dữ liệu về.
* Cộng đồng tốt: Angular được hỗ trợ bởi Google và cộng đồng.

# CHƯƠNG 2: THIẾT KẾ ỨNG DỤNG

## 2.1. Đặc tả tình huống sử dụng:

### 2.1.1. Nhóm chức năng nhân viên:

#### 2.1.1.1. Mô tả:

Hệ thống cho phép thực hiện các chức năng:

* Xem danh sách / chi tiết / tìm kiếm đơn hàng.
* Tạo mới đơn hàng và in hóa đơn cho khách hàng.
* Sửa thông tin đơn hàng.
* Xem danh sách / chi tiết / tìm kiếm sản phẩm.

#### 2.1.1.2. Chi tiết usecase:

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Nhân viên xem danh sách /chi tiết / tìm kiếm đơn hàng** | |
| **Mô tả Usecase** | | Cho phép nhân viên xem danh sách / chi tiết / tìm kiếm đơn hàng. | |
| **Tác nhân** | **Chính** | Nhân viên | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là nhân viên | |
| **Kết quả thực hiện** | | Thực hiện xem danh sách / chi tiết / tìm kiếm đơn hàng thành công. | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Trên giao diện hệ thống nhân viên chọn xem danh sách đơn hàng. | | | Hiển thị danh sách đơn hàng. |
| Nhân viên chọn đơn hàng cần xem chi tiết. | | | Hiển thị chi tiết đơn hàng được chọn. |
| Nhân viên nhập thông tin tìm kiếm đơn hàng. | | | Hiển thị danh sách đơn hàng theo thông tin tìm kiếm. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Nhân viên tạo mới đơn hàng và in hóa đơn cho khách hàng** | |
| **Mô tả Usecase** | | Cho phép nhân viên tạo mới đơn hàng, in đơn hàng cho khách hàng. | |
| **Tác nhân** | **Chính** | Nhân viên | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là nhân viên | |
| **Kết quả thực hiện** | | Đơn hàng tạo mới thành công. | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Trên giao diện hệ thống nhân viên chọn tạo mới đơn hàng. | | | Hiện thị form đơn hàng. |
| Nhân viên nhập các thông tin đơn hàng. | | | Lưu thông tin vào cơ sở dữ liệu và xuất thông tin ra file văn bản/excel để in. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Nhân viên sửa thông tin đơn hàng** | |
| **Mô tả Usecase** | | Cho phép nhân viên sửa thông tin đơn hàng. | |
| **Tác nhân** | **Chính** | Nhân viên | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là nhân viên.  Đơn hàng ở trạng thái “sửa”. | |
| **Kết quả thực hiện** | | Thông tin đơn hàng sửa thành công | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Nhân viên chọn đơn hàng cần sửa. | | | Hiển thị form sửa thông tin đơn hàng. |
| Nhân viên sửa thông tin đơn hàng. | | | Lưu lại đơn hàng sau khi sửa thông tin. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Nhân viên xem danh sách / chi tiết / tìm kiếm sản phẩm** | |
| **Mô tả Usecase** | | Cho phép nhân viên xem danh sách / chi tiết / tìm kiếm sản phẩm. | |
| **Tác nhân** | **Chính** | Nhân viên | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là nhân viên. | |
| **Kết quả thực hiện** | | Thực hiện xem danh sách / chi tiết / tìm kiếm sản phẩm thành công. | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Trên giao diện hệ thống nhân viên chọn xem danh sách sản phẩm. | | | Hiển thị danh sách sản phẩm. |
| Nhân viên chọn sản phẩm cần xem chi tiết. | | | Hiển thị chi tiết sản phẩm được chọn. |
| Nhân viên nhập thông tin tìm kiếm sản phẩm. | | | Hiển thị danh sách sản phẩm theo thông tin tìm kiếm. |

### 2.1.2. Nhóm chức năng quản lý:

#### 2.1.2.1. Mô tả:

Hệ thống cho phép thực hiện các chức năng:

* Xem danh sách / chi tiết / tìm kiếm đơn hàng.
* Thay đổi trạng thái đơn hàng nếu đơn hàng cần sửa đổi thông tin.
* Xem danh sách / chi tiết / tìm kiếm sản phẩm.
* Sửa thông tin sản phẩm.
* Xem danh sách / chi tiết / tìm kiếm tài khoản.
* Thêm mới / xóa tài khoản.

#### 2.1.2.2. Chi tiết usecase:

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Quản lý xem danh sách / chi tiết / tìm kiếm đơn hàng** | |
| **Mô tả Usecase** | | Cho phép nhân viên xem danh sách / chi tiết / tìm kiếm đơn hàng. | |
| **Tác nhân** | **Chính** | Quản lý | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là quản lý. | |
| **Kết quả thực hiện** | | Thực hiện xem danh sách / chi tiết / tìm kiếm đơn hàng thành công. | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Trên giao diện hệ thống nhân viên chọn xem danh sách đơn hàng. | | | Hiển thị danh sách đơn hàng. |
| Nhân viên chọn đơn hàng cần xem chi tiết. | | | Hiển thị chi tiết đơn hàng được chọn. |
| Nhân viên nhập thông tin tìm kiếm đơn hàng. | | | Hiển thị danh sách đơn hàng theo thông tin tìm kiếm. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Quản lý thay đổi trạng thái đơn hàng nếu đơn hàng cần sửa đổi thông tin.** | |
| **Mô tả Usecase** | | Cho phép nhân viên thay đổi trạng thái đơn hàng. | |
| **Tác nhân** | **Chính** | Quản lý | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là quản lý. | |
| **Kết quả thực hiện** | | Thay đổi trạng thái đơn hàng thành công. | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Trên giao diên hệ thống nhân viên tìm đơn hàng cần thay đổi trạng thái. | | | Hiển thị đơn hàng theo thông tin tìm kiếm. |
| Nhân viên thực hiện chọn trạng thái cần thay đổi. | | | Lưu lại trạng thái đã thay đổi. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Quản lý xem danh sách / chi tiết / tìm kiếm sản phẩm** | |
| **Mô tả Usecase** | | Cho phép nhân viên xem danh sách / chi tiết / tìm kiếm sản phẩm. | |
| **Tác nhân** | **Chính** | Quản lý | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là quản lý. | |
| **Kết quả thực hiện** | | Thực hiện xem danh sách / chi tiết / tìm kiếm sản phẩm thành công. | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Trên giao diện hệ thống nhân viên chọn xem danh sách sản phẩm. | | | Hiển thị danh sách sản phẩm. |
| Nhân viên chọn sản phẩm cần xem chi tiết. | | | Hiển thị chi tiết sản phẩm được chọn. |
| Nhân viên nhập thông tin tìm kiếm sản phẩm. | | | Hiển thị danh sách sản phẩm theo thông tin tìm kiếm. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Quản lý sửa thông tin sản phẩm** | |
| **Mô tả Usecase** | | Cho phép nhân viên sửa thông tin sản phẩm. | |
| **Tác nhân** | **Chính** | Quản lý | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là quản lý. | |
| **Kết quả thực hiện** | | Thông tin sản phẩm sửa thành công | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Nhân viên chọn sản phẩm cần sửa. | | | Hiển thị form sửa thông tin sản phẩm. |
| Nhân viên sửa thông tin sản phẩm. | | | Lưu lại sản phẩm sau khi sửa thông tin. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Quản lý xem danh sách / chi tiết / tìm kiếm tài khoản** | |
| **Mô tả Usecase** | | Cho phép nhân viên xem danh sách / chi tiết / tìm kiếm tài khoản. | |
| **Tác nhân** | **Chính** | Quản lý | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là quản lý. | |
| **Kết quả thực hiện** | | Thực hiễn xem danh sách / chi tiết / tìm kiếm tài khoản thành công. | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Trong giao diện hệ thông nhân viên chon xem danh sách tài khoản. | | | Hiển thị danh sách tài khoản. |
| Chọn tài khoản cần xem chi tiết. | | | Hiển thị thông tin tài khoản. |
| Nhân viên nhập thông tin tài khoản cần tìm. | | | Hiện thị danh sách tài khoản theo thông tin tìm kiếm. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Tên UseCase: Quản lý thêm mới / xóa tài khoản** | |
| **Mô tả Usecase** | | Cho phép nhân viên thêm / xóa tài khoản. | |
| **Tác nhân** | **Chính** | Quản lý | |
| **Phụ** |  | |
| **Điều kiện thực hiện** | | Nhân viên đăng nhập hệ thống và được cấp quyền.  Nhân viên có chức danh là quản lý. | |
| **Kết quả thực hiện** | | Thực hiện thêm / xóa tài khoản thành công. | |
| **Thao tác** | | | **Hệ thống hoạt động** |
| Trong giao diện hệ thống nhân viên chọn thêm mới tài khoản. | | | Hiển thị form thêm mới tài khoản. |
| Nhân viên nhập thông tin tài khoản cần thêm. | | | Lưu tài khoản mới và hệ thống. |
| Nhân viên tìm tài khoản cần xóa và thực hiện thao tác xóa tài khoản. | | | Tài khoản được chọn bị xóa khỏi hệ thống. |

## 2.2. Mô tả biểu đồ tuần tự ( Sequence diagram ) theo chức năng:

### 2.2.1. Biểu đồ tuần tự theo nhóm chức năng nhân viên:

#### 2.2.1.1. Xem danh sách / chi tiết / tìm kiếm đơn hàng:



Hình 2.1. Biểu đồ tuần tự Nhân viên xem danh sách / chi tiết / tìm kiếm

đơn hàng

**\* Mô tả biểu đồ:**

* Bước 1:
  + Nhân viên yêu cầu mở form cho <FormDSDH>.
  + <FormDSDH> gửi yêu cầu lấy danh sách đơn hàng đến <DonHangRepository>.
  + <DonHangRepository> trả dữ liệu, hiển thị dữ liệu lên <FormDSDH>.
* Bước 2:
  + Nhân viên yêu cầu xem chi tiết đơn hàng.
  + <FormDSDH> gửi yêu cầu hiển thị cho <FormChiTietDH>.
  + <FormChiTietDH> gửi yêu cầu lấy thông tin đơn hàng đến <DonHangRepository>
  + <DonHangRepository> trả dữ liệu, hiển thị lên <FormChiTietDH>.
* Bước 3:
  + Nhân viên thực hiện bước 1 và nhập thông tin tìm kiếm.
  + <FormDSDH> gửi yêu cầu tìm kiếm đơn hàng đến <DonHangRepository>.
  + <DonHangRepository> trả dữ liệu theo thông tin tìm kiếm, hiển thị dữ liệu lên <FormDSDH>.

#### 2.2.1.2. Tạo mới đơn hàng và in hóa đơn cho khách hàng:



Hình 2.2. Biểu đồ tuần tự Nhân viên tạo đơn hàng

**\* Mô tả biểu đồ:**

* Nhân viên yêu cầu tạo đơn hàng đến <FormDSDH>.
* <FormDSDH> gửi yêu cầu hiển thị <FormTaoDH>, nhân viên nhập thông tin đơn hàng, gửi yêu cầu tạo đơn hàng cho <DonHangRepository>.
* <DonHangRepository> thực hiện tạo đơn hàng.
* <DonHangRepository> trả dữ liệu đơn hàng mới cho <FormChiTietDH>.
* Nhân viên thực hiện in hóa đơn cho khách hàng từ <FormChiTietDH>.

#### 2.2.1.3. Sửa thông tin đơn hàng:



Hình 2.3. Biểu đồ tuần tự Nhân viên sửa đơn hàng

**\* Mô tả biểu đồ:**

* Nhân viên yêu cầu sửa thông tin đơn hàng.
* <FormChiTietDH> yêu cầu hiển thị <FormSuaDH>, nhân viên thay đổi thông tin cần sửa.
* <FormSuaDH> gửi yêu cầu sửa cho <DonHangRepository>.
* <DonHangRepository> thực hiện sửa đơn hàng.
* <DonHangRepository> trả dữ liệu đơn hàng sau khi sửa cho <FormChiTietDH>.

#### 2.2.1.4. Xem danh sách / chi tiết / tìm kiếm sản phẩm:



Hình 2.4. Biểu đồ tuần tự Nhân viên xem danh sách / chi tiết / tìm kiếm

sản phẩm

**\* Mô tả biểu đồ:**

* Bước 1:
  + Nhân viên yêu cầu mở form cho <FormDSSP>.
  + <FormDSSP> gửi yêu cầu lấy danh sách sản phẩm đến <SanPhamRepository>.
  + <SanPhamRepository> trả dữ liệu, hiển thị dữ liệu lên <FormDSSP>.
* Bước 2:
  + Nhân viên yêu cầu xem chi tiết sản phẩm.
  + <FormDSSP> gửi yêu cầu hiển thị cho <FormChiTietSP>.
  + <FormChiTietSP> gửi yêu cầu lấy thông tin sản phẩm đến <SanPhamRepository>
  + <SanPhamRepository> trả dữ liệu, hiển thị lên <FormChiTietSP>.
* Bước 3:
  + Nhân viên thực hiện bước 1 và nhập thông tin tìm kiếm.
  + <FormDSSP> gửi yêu cầu tìm kiếm sản phẩm đến <SanPhamRepository>.
  + <SanPhamRepository> trả dữ liệu theo thông tin tìm kiếm, hiển thị dữ liệu lên <FormDSSP>.

### 2.2.2. Biểu đồ tuần tự theo nhóm chức năng quản lý:

#### 2.2.2.1. Xem danh sách / chi tiết / tìm kiếm đơn hàng:



Hình 2.5. Biểu đồ tuần tự Quản lý xem danh sách / chi tiết / tìm kiếm

đơn hàng

**\* Mô tả biểu đồ:**

* Bước 1:
  + Quản lý yêu cầu mở form cho <FormDSDH>.
  + <FormDSDH> gửi yêu cầu lấy danh sách đơn hàng đến <DonHangRepository>.
  + <DonHangRepository> trả dữ liệu, hiển thị dữ liệu lên <FormDSDH>.
* Bước 2:
  + Quản lý yêu cầu xem chi tiết đơn hàng.
  + <FormDSDH> gửi yêu cầu hiển thị cho <FormChiTietDH>.
  + <FormChiTietDH> gửi yêu cầu lấy thông tin đơn hàng đến <DonHangRepository>
  + <DonHangRepository> trả dữ liệu, hiển thị lên <FormChiTietDH>.
* Bước 3:
  + Quản lý thực hiện bước 1 và nhập thông tin tìm kiếm.
  + <FormDSDH> gửi yêu cầu tìm kiếm đơn hàng đến <DonHangRepository>.
  + <DonHangRepository> trả dữ liệu theo thông tin tìm kiếm, hiển thị dữ liệu lên <FormDSDH>.

#### 2.2.2.2. Thay đổi trạng thái đơn hàng nếu đơn hàng cần sửa đổi thông tin:



Hình 2.6. Biểu đồ tuần tự Quản lý thay đổi trạng thái đơn hàng

**\* Mô tả biểu đồ:**

* Quản lý yêu cầu thay đổi trường trạng thái đơn hàng.
* <FormChiTietDH> gửi yêu cầu thay đổi trường trạng thái cho <DonHangRepository>.
* <DonHangRepository> thực hiện sửa trường trạng thái.

#### 2.2.2.3. Xem danh sách / chi tiết / tìm kiếm sản phẩm:



Hình 2.7. Biểu đồ tuần tự Quản lý xem danh sách / chi tiết / tìm kiếm

sản phẩm

**\* Mô tả biểu đồ:**

* Bước 1:
  + Quản lý yêu cầu mở form cho <FormDSSP>.
  + <FormDSSP> gửi yêu cầu lấy danh sách sản phẩm đến <SanPhamRepository>.
  + <SanPhamRepository> trả dữ liệu, hiển thị dữ liệu lên <FormDSSP>.
* Bước 2:
  + Quản lý yêu cầu xem chi tiết sản phẩm.
  + <FormDSSP> gửi yêu cầu hiển thị cho <FormChiTietSP>.
  + <FormChiTietSP> gửi yêu cầu lấy thông tin sản phẩm đến <SanPhamRepository>
  + <SanPhamRepository> trả dữ liệu, hiển thị lên <FormChiTietSP>.
* Bước 3:
  + Quản lý thực hiện bước 1 và nhập thông tin tìm kiếm.
  + <FormDSSP> gửi yêu cầu tìm kiếm sản phẩm đến <SanPhamRepository>.
  + <SanPhamRepository> trả dữ liệu theo thông tin tìm kiếm, hiển thị dữ liệu lên <FormDSSP>.

#### 2.2.2.4. Sửa thông tin sản phẩm:



Hình 2.8. Biểu đồ tuần tự Quản lý sửa sản phẩm

**\* Mô tả biểu đồ:**

* Quản lý yêu cầu sửa thông tin sản phẩm.
* <FormChiTietSP> yêu cầu hiển thị <FormSuaSP>, quản lý thay đổi thông tin cần sửa.
* <FormSuaSP> gửi yêu cầu sửa cho <SanPhamRepository>.
* <SanPhamRepository> thực hiện sửa sản phẩm.
* <SanPhamRepository> trả dữ liệu sản phẩm sau khi sửa cho <FormChiTietSP>.

#### 2.2.2.5. Xem danh sách / chi tiết / tìm kiếm tài khoản:



Hình 2.9. Biểu đồ tuần tự Quản lý xem danh sách / chi tiết / tìm kiếm

tài khoản

**\* Mô tả biểu đồ:**

* Bước 1:
  + Quản lý yêu cầu mở form cho <FormDSTK>.
  + <FormDSTK> gửi yêu cầu lấy danh sách tài khoản đến <TaiKhoanRepository>.
  + <TaiKhoanRepository> trả dữ liệu, hiển thị dữ liệu lên <FormDSTK>.
* Bước 2:
  + Quản lý yêu cầu xem chi tiết tài khoản.
  + <FormDSTK> gửi yêu cầu hiển thị cho <FormChiTietTK>.
  + <FormChiTietTK> gửi yêu cầu lấy thông tin tài khoản đến <TaiKhoanRepository>
  + <TaiKhoanRepository> trả dữ liệu, hiển thị lên <FormChiTietTK>.
* Bước 3:
  + Quản lý thực hiện bước 1 và nhập thông tin tìm kiếm.
  + <FormDSTK> gửi yêu cầu tìm kiếm tài khoản đến <TaiKhoanRepository>.
  + <TaiKhoanRepository> trả dữ liệu theo thông tin tìm kiếm, hiển thị dữ liệu lên <FormDSTK>.

#### 2.2.2.6. Thêm mới / xóa tài khoản:



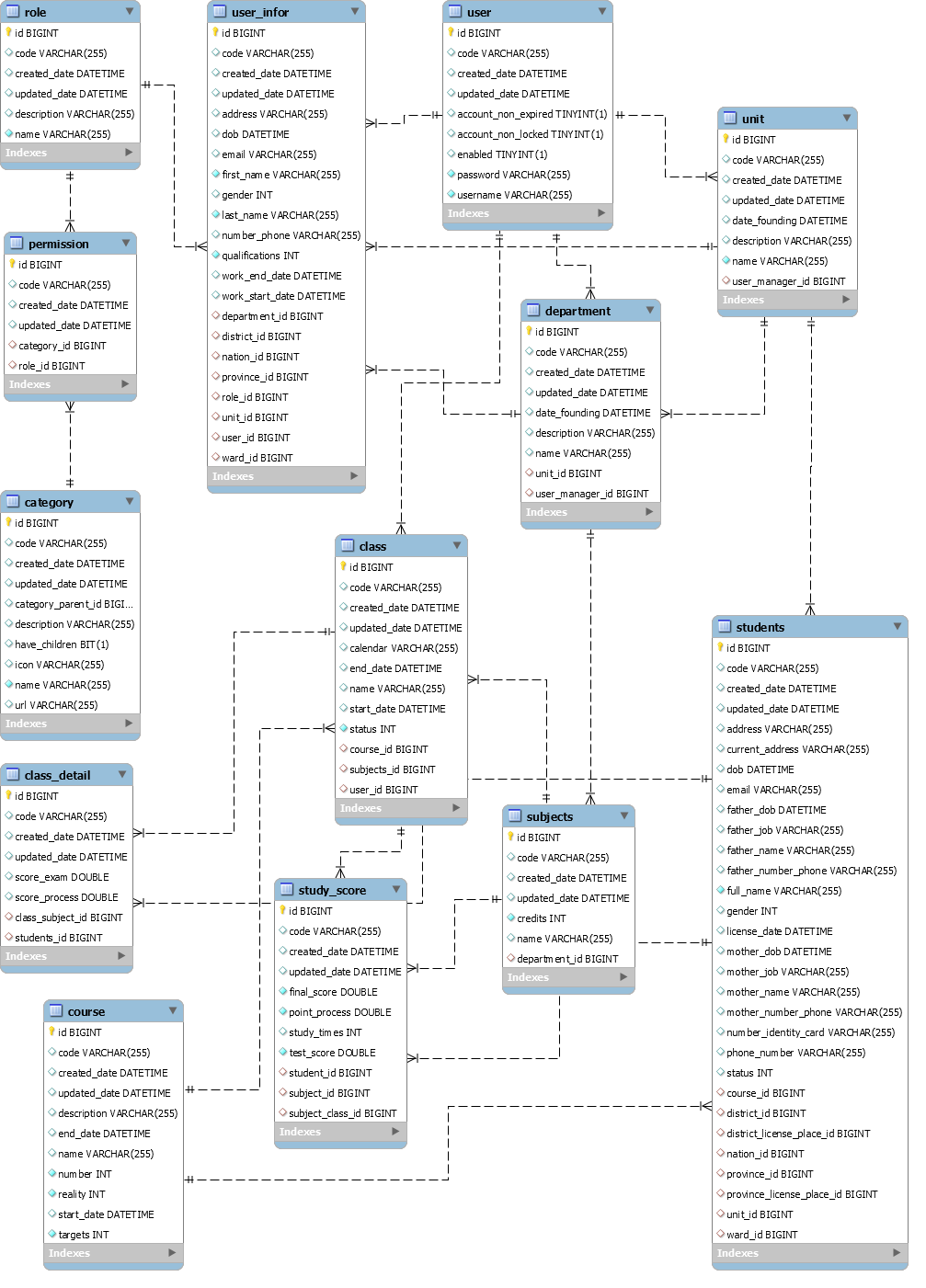
Hình 2.10. Biểu đồ tuần tự Quản lý thêm / xóa tài khoản

**\* Mô tả biểu đồ:**

* Bước 1:
  + Quản lý yêu cầu tạo tài khoản đến <FormDSTK>.
  + <FormDSTK> gửi yêu cầu hiển thị <FormTaoTK>, quản lý nhập thông tin tài khoản, gửi yêu cầu tạo cho <TaiKhoanRepository>.
  + <TaiKhoanRepository> thực hiện tạo tài khoản.
  + <TaiKhoanRepository> trả dữ liệu tài khoản mới cho <FormChiTietTK>.
* Bước 2:
  + Quản lý yêu cầu xóa tài khoản đến <FormDSTK>.
  + <FormDSTK> gửi yêu cầu xóa đến <TaiKhoanRepository>.
  + <TaiKhoanRepository> thực hiện xóa tài khoản.

## 2.3. Thiết kế cơ sở dữ liệu:

### 2.3.1. Mô hình ERD:



Hình 2.11. Mô hình ERD

### 2.3.2. Chi tiết các bảng:

#### 2.3.2.1. Bảng user

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| account\_non\_locked | Tinyint | 1 | Tài khoản không bị khóa |
| account\_non\_expired | Tinyint | 1 | Tài khoản không hết hạn |
| enabled | Tinyint | 1 | Tài khoản đang được dùng |
| password | Varchar(50) |  | Mật khẩu |
| username | Varchar(50) |  | Tên đăng nhập |

Bảng 2.1. Mô tả chi tiết các trường bảng user

#### 2.3.2.2. Bảng user\_infor (thông tin tài khoản người dùng)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| adress | Varchar(255) | Null | Địa chỉ |
| dob | Datetime | Null | Ngày sinh |
| email | Varchar(50) | Null | Địa chỉ email |
| gender | Int | 2 | Giới tính: 0 – nam, 1 – nữ, 2 - khác |
| number\_phone | Varchar(50) | Null | Số điện thoại |
| role\_id | Bigint |  | Id của chức danh |
| first\_name | Varchar(50) |  | Họ |
| last\_name | Varchar(50) |  | Tên |
| unit\_id | Bigint |  | Id khoa / phòng ban |
| user\_id | Bigint |  | Id tài khoản đăng nhập |
| qualifications | Int | Null | Trình độ chuyên môn |
| work\_start\_date | Datetime | Null | Ngày bắt đầu làm việc |
| work\_end\_date | Datetime | Null | Ngày kết thúc làm việc |
| nation\_id | Bigint | Null | Quốc tịch |
| district\_id | Bigint | Null | Quận/huyện/thành phố |
| proince\_id | Bigint | Null | Tỉnh/thành phố |
| ward\_id | Bigint | Null | Phường/xã |
| department\_id | Bigint | Null | Id bộ môn |

Bảng 2.2. Mô tả chi tiết các trường bảng user\_infor

#### 2.3.2.3. Bảng students

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| address | Varchar(255) | Null | Địa chỉ |
| dob | Datetime | Null | Ngày sinh |
| email | Varchar(50) | Null | Địa chỉ email |
| phone\_number | Varchar(50) | Null | Số điện thoại |
| nation\_id | Bigint | Null | Quốc tịch |
| father\_dob | Datetime | Null | Ngày sinh của bố |
| father\_name | Varchar(50) | Null | Tên cuả bố |
| father\_job | Varchar(50) | Null | Nghề nghiệp của bố |
| father\_number\_phone | Varchar(50) | Null | Số điện thoại của bố |
| mother\_dob | Datetime | Null | Ngày sinh của mẹ |
| mother\_name | Varchar(50) | Null | Tên cuả mẹ |
| mother\_job | Varchar(50) | Null | Nghề nghiệp của mẹ |
| mother\_number\_phone | Varchar(50) | Null | Số điện thoại của mẹ |
| license\_date | Datetime | Null | Ngày cấp CMND/CCCD/hộ chiếu |
| number\_identity\_card | Varchar(50) | Null | Số CMND/CCCD/hộ chiếu |
| district\_id | Bigint | Null | Quận/huyện/thành phố |
| district\_license\_place\_id | Bigint | Null | Quận/huyện/thành phố - nơi cấp CMND/CCCD |
| province\_id | Bigint | Null | Tỉnh/thành phố |
| province\_license\_place\_id | Bigint | Null | Tỉnh/thành phố - nơi cấp CMND/CCCD |
| ward\_id | Bigint | Null | Phường xã |
| status | Int | 1 | Trạng thái học: 0 – nghỉ học, 1 – đang học, 2 – hoàn thành |
| gender | Int | 2 | Giới tính: 0 – nam, 1 – nữ, 2 - khác |
| current\_address | Varchar(255) | Null | Địa chỉ hiện tại |
| course\_id | Bigint |  | Khóa học |
| unit\_id | Bigint |  | Khoa |
| full\_name | Varchar(50) |  | Tên đầy đủ |

Bảng 2.3. Mô tả chi tiết các trường bảng students

#### 2.3.2.4. Bảng study\_score (điểm sinh viên)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| final\_score | Double | Null | Điểm tổng kết học phần |
| point\_process | Double |  | Điểm quá trình |
| study\_times | Int |  | Lần học |
| test\_score | Double |  | Điểm thi |
| student\_id | Bigint |  | Id của sinh viên |
| subject\_id | Bigint |  | Id của học phần |
| subject\_class\_id | Bigint |  | Id cảu lớp học phần |

Bảng 2.4. Mô tả chi tiết các trường bảng study\_score

#### 2.3.2.5. Bảng category (danh mục)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| Description | Varchar(255) | Null | Mô tả danh mục |
| Name | Varchar(50) |  | Tên danh mục |
| url | Varchar(50) |  | Đường dẫn truy cập |
| Category\_parent\_id | Bigint | Null | Id danh mục cha |
| Have\_children | Tinyint | Null | Có danh mục con |
| icon | Varchar(50) | Null | Biểu tượng |

Bảng 2.5. Mô tả chi tiết các trường bảng category

#### 2.3.2.6. Bảng role (chức danh)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) |  | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| name | Varchar(50) |  | Tên chức danh |
| description | Varchar(255) |  | Mô tả chức danh |

Bảng 2.6. Mô tả chi tiết các trường bảng brand

#### 2.3.2.7. Bảng permission (quyền truy cập danh mục ứng với chức danh)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| role\_id | Bigint |  | Id chức danh |
| category\_id | Bigint |  | Id danh mục |

Bảng 2.7. Mô tả chi tiết các trường bảng permission

#### 2.3.2.8. Bảng class (lớp học phần)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| calendar | Varchar(255) | Null | Lịch học |
| end\_date | Datetime | Null | Ngày kết thúc |
| start\_date | Datetime | Null | Ngày bắt đầu |
| name | Varchar(50) |  | Tên lớp học phần |
| status | Int | 0 | Trạng thái: 0 – chuẩn bị, 1 – đang học, 2 – kết thúc |
| course\_id | Bigint |  | Khóa học |
| subjects\_id | Bigint |  | Học phần |
| user\_id | Bigint | Null | Giảng viên phụ trách |

Bảng 2.8. Mô tả chi tiết các trường bảng class

#### 2.3.2.9. Bảng class\_detail

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| Score\_exam | Double |  | Điểm thi |
| Score\_process | Double |  | Điểm quá trình |
| Class\_subject\_id | Bigint |  | Lớp học phần |
| Student\_id | Bigint |  | Id sinh viên |

Bảng 2.9. Mô tả chi tiết các trường bảng class\_detail

#### 2.3.2.10. Bảng subjects (học phần)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) |  | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| credits | Int |  | Số tín chỉ |
| name | Varchar(50) |  | Tên học phần |
| department\_id | bigint |  | Id bộ môn |

Bảng 2.10. Mô tả chi tiết các trường bảng subjects

#### 2.3.2.11. Bảng unit (khoa / phòng ban)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| date\_founding | Datetime | Null | Ngày thành lập |
| name | Varchar(50) |  | Tên khoa/phòng ban |
| description | Varchar(255) | Null | Mô tả |
| user\_manager\_id | Bigint | Null | Id người quản lý |

Bảng 2.11. Mô tả chi tiết các trường bảng unit

#### 2.3.2.12. Bảng department (bộ môn)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| date\_founding | Datetime | Null | Ngày thành lập |
| name | Varchar(50) |  | Tên bộ môn |
| description | Varchar(255) | Null | Mô tả |
| user\_manager\_id | Bigint | Null | Id người quản lý |
| unit\_id | Bigint |  | Id khoa |

Bảng 2.12. Mô tả chi tiết các trường bảng department

#### 2.3.2.13. Bảng course (khóa học)

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Giá trị mặc định** | **Mô tả** |
| id | Bigint | Tự tăng | Khóa chính của bảng |
| code | Varchar(50) | null | Mã tài khoản người dùng |
| created\_date | Datetime | Null | Ngày tạo bản ghi |
| updated\_date | Datetime | Null | Ngày cập nhật bản ghi |
| description | Varchar(255) | Null | Mô tả |
| name | Varchar(50) |  | Tên khóa học |
| end\_date | Datetime | Null | Ngày kết thúc |
| start\_date | Datetime | Null | Ngày bắt đầu |
| targets | Bigint | Null | Chỉ tiêu tuyển sinh |
| reality | Bigint | Null | Thực tế tuyển sinh |
| number | Int | Null | Số khoa |

Bảng 2.13. Mô tả chi tiết các trường bảng course

# CHƯƠNG 3: XÂY DỰNG VÀ PHÁT TRIỂN

## 3.1. Cấu hình Security:

### 3.1.1. Server:

Trong file “*pom.xml*” khai báo các thư viện cho security như sau:

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-security</**artifactId**>  
</**dependency**>

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-security</**artifactId**>  
</**dependency**>

<**dependency**>  
 <**groupId**>org.springframework.security</**groupId**>  
 <**artifactId**>spring-security-test</**artifactId**>  
 <**scope**>test</**scope**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>io.jsonwebtoken</**groupId**>  
 <**artifactId**>jjwt</**artifactId**>  
 <**version**>0.9.0</**version**>  
</**dependency**>

Trong package “*com.datn.config”* tạo các class *SecurityConfig, JwtConfig, JwtAuthEntryPoint, JwtAuthTokenFilter* để xác thực quyền truy cậpcác đường link API lấy dữ liệu trong ứng dụng.

*JwtConfig* là lớp chứa các thông tin chung để tạo mã Jwt (Json Web Token) như thời gian hết hạn, khóa bí mật, v.v… và các phương thức tạo token, kiểm tra token. Các giá trị của thông tin được lưu trong file “*application.properties*” và được tham chiếu các trường trong class thông qua annotation @Value của Spring .

* Khai báo thông tin cấu hình trong “*application.properties”*:

security.jwt.uri=/auth/\*\*  
security.jwt.expiration=60480000  
security.jwt.header=Authorization  
security.jwt.prefix=Bearer  
security.jwt.secret=DoAn160704493

* Khai báo thông tin trong class *JwtConfig*:

@Value("${security.jwt.uri}")  
private String Uri;  
@Value("${security.jwt.header}")  
private String header;  
@Value("${security.jwt.prefix}")  
private String prefix;  
@Value("${security.jwt.expiration}")  
private int expiration;  
@Value("${security.jwt.secret}")  
private String secret;

* Phương thức tạo mã Jwt:

**public** String generateJwtToken(Authentication authentication){  
 Date present = **new** Date();  
 User userPrincipal = (User) authentication.getPrincipal();  
 *//Tạo và trả về một token dạng String* **return** Jwts.*builder*()  
 .setSubject(userPrincipal.getUsername())  
 .setIssuedAt(present)  
 .setExpiration(**new** Date(present.getTime()+**expiration**))  
 .signWith(SignatureAlgorithm.***HS512***, **secret**)  
 .compact();  
}

* Phương thức kiểm tra mã Jwt:

**public boolean** validateJwtToken(String token){  
 **try** {  
 Jwts.*parser*().setSigningKey(**secret**).parseClaimsJws(token);  
 **return true**;  
 }**catch** (SignatureException e){  
 ***logger***.error(**"Invalid JWT signature -> Message: {}"**,e.getMessage());  
 }**catch** (MalformedJwtException e){  
 ***logger***.error(**"Invalid JWT token -> Message: {}"**,e.getMessage());  
 }**catch** (ExpiredJwtException e){  
 ***logger***.error(**"Expired JWT token -> Message: {}"**,e.getMessage());  
 }**catch** (UnsupportedJwtException e){  
 ***logger***.error(**"Unsupported JWT token -> Message: {}"**,e.getMessage());  
 }**catch** (IllegalArgumentException e){  
 ***logger***.error(**"JWT claims string is empty -> Message: {}"**, e.getMessage());  
 }  
 **return false**;  
}

Lớp *JwtAuthTokenFilter* kết thừa lớp *OncePerRequestFilter* của Spring cung cấp dùng để kiểm tra mã Jwt ở header các request gửi về server. Lớp *JwtAuthEntryPoint* thực thi interface *AuthenticationEntryPoint* của Spring cung cấp dùng để xử lý và trả về kết quả cho người dùng nếu quá trình xác thực xảy ra ngoại lệ. Lớp *SecurityConfig* kế thừa lớp *WebSecurityConfigurerAdapter* của Spring dùng để phân quyền truy cập các url API, sử dụng các phương thức kiểm tra và xử lý ngoại lệ các request có trong các lớp *JwtAuthTokenFilter* và *JwtAuthEntryPoint.*

@Override  
**protected void** configure(HttpSecurity http) **throws** Exception {  
 http.cors().and().csrf().disable()  
 .authorizeRequests()  
 *//Các uri không cần xác thực* .antMatchers(**jwtConfig**.getUri()).permitAll()  
 *//Tất cả các request đều phải được xác thực* .anyRequest().authenticated()  
 .and()  
 .exceptionHandling().authenticationEntryPoint(**unauthorizedHandler**)  
 .and()  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.***STATELESS***);  
 *//Kiểm tra header các request trước khi trả về 1 response* http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.**class**);  
}

### 3.1.2. Client:

Trong project angular thêm thư viện @angular-jwt thông qua lệnh terminal “*npm install @auth0/angular-jwt*”. Thư viện này cung cấp một *HttpInterceptor* tự động cài mã Jwt vào header mỗi request của HttpClient. Lớp *AuthInterceptor* thực thi interface *HttpInterceptor* với phương thức intercept gán mã Jwt và header mỗi request gửi về server đồng thời khai báo một providers HTTP\_INTERCEPTOR trong “*app.module.ts*” để ứng dụng thực thi phương thức intercept.

* Phương thức intercept():

intercept(req: HttpRequest<*any*>, next: HttpHandler): Observable<*HttpEvent*<*any*>> {  
 *this*.authReq = req;  
 *const* token = *this*.tokenStorageService.getToken();  
 *// Kiểm tra có token tồn tại trong session storage và gán vào header mỗi request  
 if* (token != *null*) {  
 *this*.authReq = req.clone({headers: req.headers.set(*this*.authConfig.*REQUEST\_HEADER*,  
 *this*.authConfig.*REQUEST\_HEADER\_PREFIX* + ' ' + token)});  
 }  
 *return* next.handle(*this*.authReq);  
}

* Khai báo providers:

{  
 provide: ***HTTP\_INTERCEPTORS***,  
 useClass: AuthInterceptor,  
 multi: true  
}

Trong thư viên angular-jwt còn cung cấp một lớp *JwtHelperService* với các phương thức sẵn có phục vụ cho việc giải mã, kiểm tra các thông tin như thời gian tạo, thời gian hết hạn, v.v… của mã Jwt.

## 3.2. Cấu hình kết nối cơ sở dữ liệu:

Thông tin cấu hình kết nối tới cơ sở dữ liệu được viết trong file “*application.yml*”, lưu trữ ở vị trí “*src/main/resources*” ở phần mã java. Các thông tin kết nối cơ sở dữ liệu bao gồm:

* Url: đường link localhost chỉ đến database sẽ kết nối thuộc hệ quản trị cở sở dữ liệu MySql.
* Username: tài khoản đăng nhập vào hệ quản trị cơ sở dữ liệu MySql với các quyền cần thiết như tạo/sửa bảng, thêm/sửa/xóa bản ghi, v.v….
* Password: mật khẩu tương ứng với username.
* Driver: thư viện của Java để kết nối chương trình viết bằng mã Java với hệ quản trị cơ sở dữ liệu MySql.

Các thông tin được khai báo trong file “*application.yml”* như sau:

* Spring.datasource.url=jdbc:mysql://localhost:3306/do\_an\_db?serverTimeZone=UTC&userLegacyDateTimeCode=false&useUnicode=yes&characterEncoding=UTF-8&reWriteBatchedInserts=true
* Spring.datasource.username=root
* Spring.datasource.password=15011996
* Spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

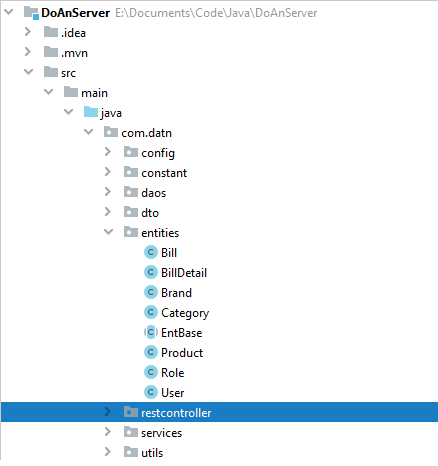
Ngoài các thông tin kết nối tới cơ sở dữ liệu còn có các thông tin cấu hình cho Hibernate và JPA với mục đích như cho ứng dụng biết loại cơ sở dữ liệu đang sử dụng để hibernate chuyển đổi câu truy vấn SQL, thu thập số liệu thống kê để hibernate điều chỉnh hiệu suất, hiển thị và định dạng câu truy vấn SQL ra console hoặc file log, v.v…. Thông tin các cấu hình được mô tả chi tiết như hình dưới đây (Hình 3.1).



Hình 3.1. Các thông tin cấu hình database và Hibernate

trong file application.yml

## 3.3. Entitiy:

Entity là các đối tượng trong ứng dụng, mỗi đối tượng tương ứng với một bảng trong cơ sở dữ liệu. Các đối tượng đều được đánh dấu bằng hai annotation @Entity và @Table thuộc “*javax.persistence*” nhằm thông báo cho hibernate biết rằng đây là một thực thể và là một bảng trong cơ sở dữ liệu. Các trường của đối tượng mặc định tương ứng với trường dữ liệu trong cơ sở dữ liệu, nếu trường thông tin có các cài đặt khác như giá trị default, not null, v.v… sẽ khai báo thông qua anotation @Column. Các trường của đối tượng không phải là một trường thông tin trong cơ sở dữ liệu thì khai báo kèm với annotation @Transient. Khóa chính của thực thể trong cơ sở dữ liệu được đánh với bởi annotation @Id trong trường thông tin của đối tượng. Các đối tượng Entity được khai báo trong package “*com.datn.entities*”.

Hình 3.2. Package chứa các Entity

Ví dụ đối tượng *User* được khai báo như sau:

@Entity  
@Table  
**public class** User **extends** EntBase **implements** UserDetails {  
 @Column(unique = **true**, nullable = **false**)  
 **private** String **username**;  
 **private** String **password**;  
 **private** String **name**;  
 **private** Date **dob**;  
 **private** String **email**;  
 **private** String **address**;  
 **private** String **phone**;  
 **private** String **avatar**;  
 @Column(columnDefinition = **"boolean default true"**)  
 **private boolean accountNonExpired**;  
 @Column(columnDefinition = **"boolean default true"**)  
 **private boolean accountNonLocked**;  
 @Column(columnDefinition = **"boolean default true"**)  
 **private boolean credentialsNonExpired**;  
 @Column(columnDefinition = **"boolean default true"**)  
 **private boolean enabled**;  
 @ManyToOne  
 @JoinColumn  
 **private** Role **role**;  
 @OneToMany(mappedBy = **"user"**, cascade = CascadeType.***ALL***)  
 **private** List<Bill> **bills**;

}

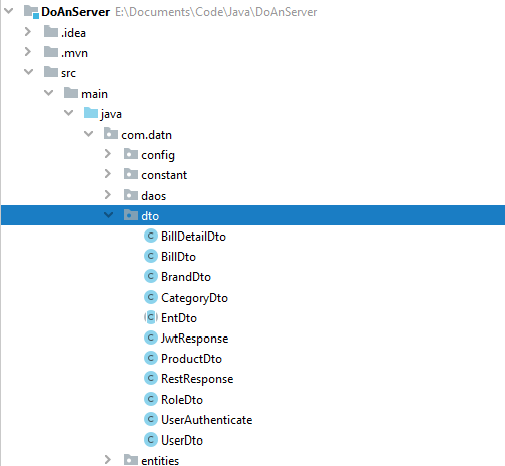
Các đối tượng đều kế thừa một lớp trừu tượng *BaseEnt* chứa các thông tin chung và các phương thức chung của tất cả đối tượng. Lớp *BaseEnt* được mô tả như sau:

@MappedSuperclass  
public abstract class BaseEnt {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String code;  
 private Date createdDate;  
 private Date updatedDate;  
  
 public BaseEnt() { }  
  
 public BaseEnt(Long id, String code, Date createdDate, Date updatedDate) {  
 this.id = id;  
 this.code = code;  
 this.createdDate = createdDate;  
 this.updatedDate = updatedDate;  
 }  
  
 public abstract BaseDto convertToDto();  
  
}

Anotation @MappedSuperClass dùng để chỉ định đây không phải là một thực thể trong cơ sở dữ liệu. Phương thức asDto() chuyển đổi dữ liệu sang Data Transfer Object để trả kết quả về phía client và các phương thức getter , setter.

## 3.4. Data Transfer Object (DTO):

Data Transfer Object (DTO) – đối tượng truyền dữ liệu là một đối tượng mang nhiều dữ liệu trong các quy trình. DTO được sử dụng khi chúng ta muốn truyền dữ liệu với nhiều thông tin trong một lần gửi yêu cầu từ client đến server. Các đối tượng này là các lớp đơn giản chỉ có các phương thức getter, setter, contructor và không có bất kỳ phương thức logic nghiệp vụ nào. Các lớp nghiệp vụ của server lấy dữ liệu từ cơ sở dữ liệu, xử lý dữ liệu và chuyển đổi thành đối tượng DTO và gửi nó đên client. Đối với client các đối tượng DTO chỉ để đọc. Các đối tương DTO được khai báo trong package “*com.datn.dto*”.

Hình 3.3. Package chứa các đối tượng DTO

Ví dụ khai báo đối tượng *UserDto*:

@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@JsonInclude(JsonInclude.Include.***NON\_NULL***)  
@JsonIgnoreProperties(ignoreUnknown = **true**)  
**public class** UserDto **extends** EntDto{  
 @NotNull  
 @NotBlank  
 **private** String **username**;  
 @NotNull  
 @NotBlank  
 **private** String **password**;  
 **private** String **name**;  
 **private** Date **dob**;  
 @Email  
 **private** String **email**;  
 **private** String **address**;  
 **private** String **phone**;  
 **private boolean accountNonExpired** = **true**;  
 **private boolean accountNonLocked** = **true**;  
 **private boolean credentialsNonExpired** = **true**;  
 **private boolean enabled** = **true**;  
 **private** String **avatar**;  
 **private long idRole**;  
  
 @Override  
 **public** User asEnt() {  
 **return** AppUtils.*mapToDtoAndEnt*(**this**,User.**class**);  
 }  
}

Giống như các entity, các đối tượng DTO đều kế thừa lớp trừu tượng *EntDto* có thông tin và phương thức chung của các đối tượng. Phương thức asEnt() để chuyển đổi một đối tượng DTO về đối tượng Entity. Các đối tượng DTO không nhất thiết phải tương ứng với đối tượng Entity vì DTO chỉ chứa các thông tin mà phía client cần đọc, một ứng dụng có rất nhiều đối tượng DTO. Mô tả lớp *EntDto* như sau:

@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
**public abstract class** EntDto {  
 **private long id**;  
  
 **public abstract** EntBase asEnt();  
}

## 3.5. Data Access Object (DAO):

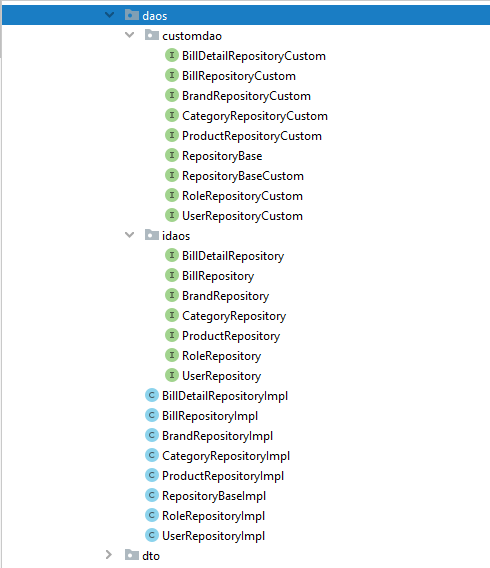
Data Access Object (DAO) – đối tượng truy cập dữ liệu là một trong những Pattern thuộc nhóm cấu trúc. DAO được sử dụng để phân tách logic lưu trữ dữ liệu trong một lớp riêng biệt. Theo cách này, các service được che dấu về cách các hoạt động cấp thấp để truy cập cơ sở dữ liệu được thực hiện thay vì có logic trực tiếp với cơ sở dữ liệu hay bất kỳ cơ chế lưu trữ nào ứng dụng cần sử dụng má sẽ để các logic này giao tiếp với lớp trung gian DAO. Lớp DAO sẽ giao tiếp với cơ sở dữ liệu như thực hiện các công việc liên quan đến lưu trữ và truy vấn dữ liệu. DAO dựa trên nguyên tắc thiết kế abstraction và encapsulation, nó sẽ bảo vệ phần còn lại của ứng dụng khỏi mọi thay đổi trong lớp lưu trữ như thay đổi công nghệ lưu trữ, thay đổi hệ quản trị cơ sở dữ liệu, v.v….

Các thành phần của DAO bao gồm:

* **Data Access Object (DAO)**: là các interface định nghĩa các phương thức trừu tượng việc triển khai truy cập dữ liệu.
* **Data Access Object Concrete**: là các lớp thực thi các interface DAO, lấy dữ liệu từ nguồn dữ liệu.
* **Data Transfer Object (DTO)**: là các lớp đơn giản để lưu trữ dữ liệu được truy xuất bằng các lớp DAO.

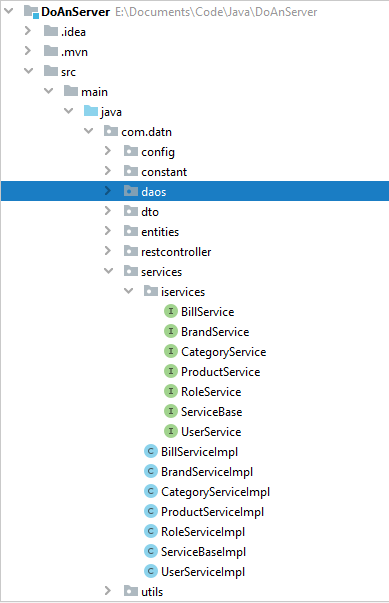
Các interface DAO được khai báo trong package “*com.datn.daos.idaos*” và “*com.datn.daos.customdao*”. Các interface trong *idaos* đều kế thừa interface *JpaRepository* khai báo các phương thức trừu tượng không cần triển khai theo nguyên tắc khai báo của JPA hoặc thông qua annotation @Query nhằm giảm thiểu lượng mã phải viết. Ví dụ *UserRepository:*

**public interface** UserRepository **extends** JpaRepository<User, Long>, UserRepositoryCustom {  
 User findUserByUsername(String username);  
  
 Page<User> findAllByRoleNotIn(Collection<Role> roles, Pageable pageable);  
  
 Page<User> findAllByRoleIn(Collection<Role> roles, Pageable pageable);  
  
 Page<User> findAllByRole\_Id(Long id, Pageable pageable);  
}

Các interface trong *customdao* khai báo các phương thức trừu tượng thực thi việc truy cập dữ liệu với các truy vấn phức tạp và không theo nguyên tắc khai báo của JPA, nó sẽ được triển khai trong các lớp thực thi các interface này trong package “*com.datn.daos*”.

Hình 3.4. Package chứa các lớp DAO

## 3.6. Service:

Service là các interface và các lớp thực thi các interface này gồm các phương thức logic nghiệp vụ. Các phương thức xử lý dữ liệu được truy vấn từ cơ sở dữ liệu, chuyển đổi dạng dữ liệu trả về cho client những dữ liệu được yêu cầu hoặc xử lý các dữ liệu nhận từ client trước khi lưu trữ vào cơ sở dữ liệu.

Hình 3.5. Package chứa các lớp Service

Các service được khai báo trong package “*com.datn.services*”. Các interface đều kế thừa interface *ServiceBase* chứa định nghĩa các phương thức chung như lưu, cập nhật, xóa, lấy tất cả dữ liệu từ một bảng, v.v…. Các ServiceImpl thực hiện triển khai các phương thức trong interface, được đánh dấu bởi annotation @Service để thông báo cho Spring đây là một Service Component, annotation @Transactional để chỉ định các phương thức của service sẽ hoạt động trong một giao dịch đảm bảo tính toàn vẹn dữ liệu nếu có lỗi xảy ra trong trình giao tiếp với cơ sở dữ liệu.

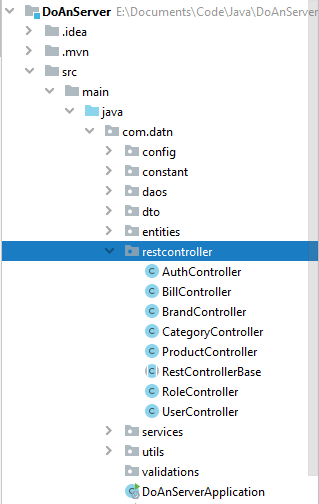
\*Mô tả interface *ServiceBase*:

**public interface** ServiceBase<T **extends** EntBase> {  
 */\*\*  
 \* Lưu 1 đối tượng  
 \** ***@param entity*** *\** ***@return*** *\*/* T save(T entity);  
  
 */\*\*  
 \* Lưu 1 mảng đối tượng  
 \** ***@param entities*** *\** ***@return*** *\*/* List<T> saveAll(List<T> entities);  
  
 */\*\*  
 \* Xóa 1 danh sách đối tượng  
 \** ***@param entities*** *\*/* **void** deleteAll(List<T> entities);  
  
 */\*\*  
 \* Xóa 1 đối tượng theo id  
 \** ***@param id*** *\*/* **void** deleteById(Long id);  
 */\*\*  
 \* Lấy tất cả bản ghi một bảng  
 \** ***@return*** *\*/* List<T> findAll();  
 */\*\*  
 \* Tìm bản ghi dữ liệu theo id  
 \** ***@param id*** *\** ***@return*** *\*/* T findById(Long id);  
  
 */\*\*  
 \* Trả về số lượng bản ghi  
 \** ***@return*** *\*/* **long** count();  
}

\*Mô tả lớp *ServiceBaseImpl*:

@Transactional  
**public class** ServiceBaseImpl<T **extends** EntBase> **implements** ServiceBase<T> {  
 **private** JpaRepository<T, Long> **jpaRepository**;  
  
 **public** ServiceBaseImpl(JpaRepository<T, Long> jpaRepository) {  
 **this**.**jpaRepository** = jpaRepository;  
 }  
  
 @Override  
 **public** T save(T entity) {  
 **return jpaRepository**.save(entity);  
 }  
  
 @Override  
 **public** List<T> saveAll(List<T> entities) {  
 **return jpaRepository**.saveAll(entities);  
 }  
  
 @Override  
 **public void** deleteAll(List<T> entities) {  
 **jpaRepository**.deleteAll(entities);  
 }  
  
 @Override  
 **public void** deleteById(Long id) {  
 **jpaRepository**.deleteById(id);  
 }  
  
 @Override  
 **public** List<T> findAll() {  
 **return jpaRepository**.findAll();  
 }  
  
 @Override  
 **public** T findById(Long id) {  
 **return jpaRepository**.findById(id).orElse(**null**);  
 }  
  
 @Override  
 **public long** count() {  
 **return jpaRepository**.count();  
 }  
}

## 3.7. Rest Controller:

 Rest Controller bao gồm các lớp tiếp nhận những yêu cầu xử lý được gửi từ phía client. Nó gọi tới các service xử lý logic nghiệp vụ để lấy thông tin dữ liệu cần thiết và trả về cho client dữ liệu dưới dạng Json. Các lớp RestController đều kế thừa lớp cơ sở *ResControllerBase* chứa các phương thức chung và định nghĩa các phương thức chung (phương thức trừu tượng) như post, put, delete, v.v…

Hình 3.6. Package chứa các Restcontroller

\*Mô tả lớp *RestControllerBase*:

**public abstract class** RestControllerBase<T **extends** EntDto> {  
  
 **public abstract** ResponseEntity<T> post(T dto);  
  
 **public abstract** ResponseEntity<T> put(T dto);  
  
 **public abstract** ResponseEntity<RestResponse> delete(String id);  
  
 **public abstract** ResponseEntity<T> findById(String id);  
  
 **public abstract** ResponseEntity<Page<T>> findAll(String page, String limit);  
  
 **public** String text(String key) {  
 **return** AppUtils.*text*(key);  
 }  
  
 **public** Pageable getPageable(String page, String limit){  
 **if** (limit == **null** || limit.equals(**""**)) limit = **"20"**;  
 **if** (page == **null** || page.equals(**""**)) page = **"0"**;  
 **return** PageRequest.*of*(Integer.*parseInt*(page), Integer.*parseInt*(limit));  
 }  
}

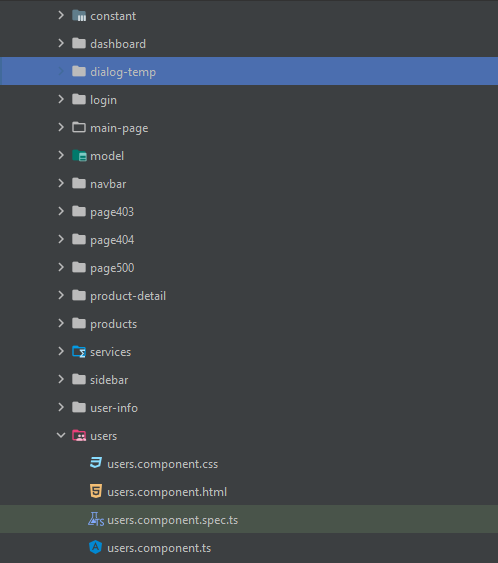
\*Mô tả một restcontroller (UserController):

@RestController  
@CrossOrigin(origins = **"http://localhost:4200"**)  
@RequestMapping(**"/user"**)  
**public class** UserController **extends** RestControllerBase<UserDto>{  
 @Autowired  
 **private** UserService **userService**;  
  
 @Override  
 @PostMapping(**"/add"**)  
 **public** ResponseEntity post(@RequestBody UserDto dto) {  
 **return new** ResponseEntity(**userService**.saveDto(dto),HttpStatus.***OK***);  
 }  
  
 @Override  
 @PutMapping(**"/update"**)  
 **public** ResponseEntity put(@RequestBody UserDto dto) {  
 **return new** ResponseEntity(**userService**.saveDto(dto),HttpStatus.***OK***);  
 }  
  
 @Override  
 @DeleteMapping(**"/delete"**)  
 **public** ResponseEntity<RestResponse> delete(@RequestParam(value = **"id"**) String id) {  
 String message = text(**"Delete account success!"**);  
 **try** {  
 **userService**.deleteById(Long.*valueOf*(id));  
 Map map = **new** HashMap();  
 map.put(**"exception"**, message);  
 RestResponse restResponse = **new** RestResponse(200, map);  
 **return new** ResponseEntity<>(restResponse, HttpStatus.***OK***);  
 } **catch** (Exception e) {  
 message = text(**"Delete account fail!"**);  
 Map map = **new** HashMap();  
 map.put(**"exception"**, message);  
 RestResponse restResponse = **new** RestResponse(400, map);  
 **return new** ResponseEntity<>(restResponse, HttpStatus.***BAD\_REQUEST***);  
 }  
 }  
  
 @Override  
 @GetMapping(**"/detail"**)  
 **public** ResponseEntity findById(@RequestParam(**"id"**) String id) {  
 **return new** ResponseEntity(**userService**.findByIdDto(Long.*parseLong*(id)), HttpStatus.***OK***);  
 }  
  
 @Override  
 @GetMapping(**"/all"**)  
 **public** ResponseEntity<Page<UserDto>> findAll(@RequestParam(value = **"page"**,required = **false**) String page,  
 @RequestParam(value = **"limit"**, required = **false**) String limit) {  
 Pageable pageable = getPageable(page, limit);  
 **return new** ResponseEntity(**userService**.findAllDto(pageable), HttpStatus.***OK***);  
 }

}

## 3.8. Components:

Component là các thành phần tạo nên một ứng dụng hoàn chỉnh. Một component có ba thành phần chính là code TypeScript, template HTML và CSS. Một ứng dụng angular có rất nhiều component, mỗi component được đặt trong một thư mục như hình (Hình 3.7):

Hình 3.7. Cấu trúc ứng dụng Angular

* **[component name].component.css:**được sử dụng để chứa mã style cho giao diện web.
* **[component name].component.html:**chứa các mã giao diện html.
* **[component name].component.ts:**chứa các mã xử lý logic.

Các component được tạo ra đều được khai báo ở mục “*declarations*” trong file “*app.module.ts*” để xác định các component sẽ được sử dụng. Mục “*imports*” khai báo các module ứng dụng sử dụng trong đó có một số module quan trọng như *BrowserModule* – module cần thiết để mọi ứng dụng angular chạy trên trình duyệt, *HttpCientModule* – module cung cấp các phương thức để giao tiếp với server, v.v….

## 3.9. Routing:

Routing – bộ định tuyến dùng để xử lý điều hướng từ chế độ xem này sang chế độ xem khác khi người dùng thực hiện các tác vụ. Các route được khai báo trong file “*app-routing.module.ts*” và được import vào “app.module.ts”. Mỗi route có hai thành phần chính là path – url chỉ dẫn đến view cần xem, component – thành phần view tương ứng với url được khai báo ở path.

const routes: Routes = [  
 { path: '', redirectTo: 'login', pathMatch: 'full'},  
 { path: 'login', component: LoginComponent },  
 { path: 'logout', redirectTo: 'login', pathMatch: 'full' },  
 { path: 'home', component: MainPageComponent, children: [  
 { path: '', redirectTo: 'dashboard', pathMatch: 'prefix'},  
 { path: 'dashboard', component: DashboardComponent },  
 { path: 'users', component: UsersComponent },  
 { path: 'profile', component: UserInfoComponent }  
 ]},  
 { path: '403', component: Page403Component },  
 { path: '404', component: Page404Component },  
 { path: '500', component: Page500Component }  
];  
@NgModule({  
 imports: [RouterModule.*forRoot*(routes)],  
 exports: [RouterModule]  
})  
export class AppRoutingModule { }

Ngoài các routing và component, ứng dụng Angular cũng có các service chứa các phương thức dùng để gửi các yêu cầu lên server. Dữ liệu nhận về ở dạng Json sẽ được tùy biến thành các đối tượng model để hiển thị lên cho người dùng. Các service phải được đánh dấu với annotation @Injectable({providedIn: ‘root’}).

# CHƯƠNG 4: XÂY DỰNG GIAO DIỆN

## 4.1. Giao diện dánh sách sinh viên:

## 4.2. Giao diện quản lý tài khoản:

## 4.3. Giao diện quản lý lớp học phần:

## 4.4. Giao diện quản lý môn học:

## 4.5. Giao diện quản lý khoa / phòng ban:

## 4.6. Giao diện quản lý bộ môn:

## 4.7. Giao diện quản lý khóa học:

## 4.8. Giao diện quản lý giảng viên:

# CHƯƠNG 5: TỔNG KẾT VÀ ĐÁNH GIÁ

## 5.1. Những điểm chương trình thực hiện được:

## 5.2. Những điểm chương trình chưa thực hiện được:

## 5.3. Hướng phát triển: