BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO TIẾN ĐỘ ĐÒ ÁN TỐT NGHIỆP  
NGHIÊN CỨU SPRINGBOOT VÀ SPRINGSERCURITY XÂY DỰNG ỨNG DỤNG QUẢN LÝ CHUỖI CỬA HÀNG   
TOUS LES TEMPS**

|  |
| --- |
| **GVHD**: Th.S Mai Cường Thọ  **Sinh viên**: Nguyễn Lê Thành Tâm |
| **MSSV**: 61134311 |

*Khánh Hòa – 2/2023*

**MỤC LỤC**

[LỜI CẢM ƠN 3](#_Toc128471518)

[PHẦN MỞ ĐẦU 4](#_Toc128471519)

[I. SƠ LƯỢC VỀ SPRING BOOT 5](#_Toc128471520)

[1. MÔ HÌNH BA LỚP 5](#_Toc128471521)

[2. MÔ HÌNH SPRING MVC 5](#_Toc128471522)

[3. SPRING BOOT 6](#_Toc128471523)

[4. CÁC THÀNH PHẦN CỦA RESTful API 10](#_Toc128471524)

[5. TỔNG QUAN VỀ JAVA PERSISTENCE API – JPA 13](#_Toc128471525)

[6. GIỚI THIỆU VỀ HIBERNATE 16](#_Toc128471526)

[7. JSON WEB TOKEN 18](#_Toc128471527)

[8. MỘT SỐ ANOTATION TRONG SPRING 20](#_Toc128471528)

[II. XÂY DỰNG HỆ THỐNG 29](#_Toc128471529)

[1. PHÂN TÍCH HỆ THỐNG, MÔ TẢ NGHIỆP VỤ 29](#_Toc128471530)

[2. YÊU CẦU HỆ THỐNG 33](#_Toc128471531)

[3. MÔ TẢ USECASE VÀ SƠ ĐỒ HOẠT ĐỘNG 35](#_Toc128471532)

[1. CƠ SỞ DỮ LIỆU 38](#_Toc128471533)

[2. BẢNG DỮ LIỆU 38](#_Toc128471534)

[3. GIAO DIỆN KẾT QUẢ 38](#_Toc128471535)

[III. KẾT LUẬN 40](#_Toc128471536)

[IV. TÀI LIỆU THAM KHẢO 40](#_Toc128471537)

**PHẦN MỞ ĐẦU**

**Nội dung đề tài**

* Tìm hiểu Java Spring và Spring Security
* Xây dựng ứng dụng quản lý chuỗi cửa hàng Tous les temps

**Báo cáo tiến độ**

* Thời gian thực hiện: từ ngày 20/2/2023 đến ngày 03/06/2023

|  |  |
| --- | --- |
| Thời gian | Nội dung báo cáo |
| Tuần 1 (Từ ngày 20/02/2023 đến ngày 26/02/2023 | * Xây dựng mục lục * Tổng quan về Java 8 và Spring booot * Tìm hiểu về RESTful API web service * JWT token * Phân tích thiết kế hệ thống   + Usecase tổng quát hệ thống  + Yêu cầu chức năng phi chức năng  + Mô tả nghiệp vụ |
| Tuần 2 (Từ ngày 27/02/2023 đến ngày 05/03/2023 |  |
| Tuần 3 (Từ ngày 06/03/2023 đến ngày 12/03/2023 |  |
| .. |  |

1. SƠ LƯỢC VỀ SPRING BOOT
2. MÔ HÌNH BA LỚP

Đây là mô hình tổ chức source code rất phổ biến trong Spring Boot. Cụ thể, ứng dụng được chia làm 3 tầng (tier hoặc layer) như sau:

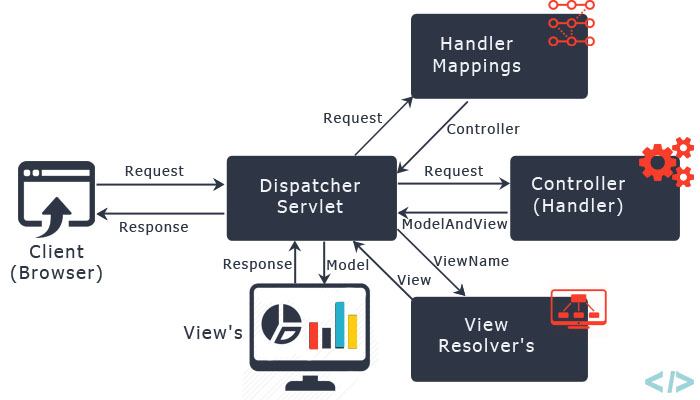
* Presentation layer: tầng này tương tác với người dùng, bằng View, Controller (trong MVC) hoặc API (nếu có).
* Business logic layer: Chứa toàn bộ logic của chương trình, các đa số code nằm ở đây
* Data access layer: Tương tác với database, trả về kết quả cho tầng business logic

Trong Spring Boot, thì có một số thành phần đại diện cho từng lớp:

Service: chứa các business logic code

Repository: đại diện cho tầng data access

1. MÔ HÌNH SPRING MVC



Các thành phần trong mô hình Spring MVC

* Controller: trả về View (có chứa data sẵn, dạng trang HTML), hoặc Model thể hiện dưới dạng API cho View (View viết riêng bằng React, Vue, hoặc Angular).
* Service: chứa các code tính toán, xử lý. Khi Controller yêu cầu, thì Service tương ứng sẽ tiếp nhận và cho ra dữ liệu trả cho Controller (trả về Model). Controller sẽ gửi về View như trên.
* Repository: Service còn có thể tương tác với service khác, hoặc dùng Repository để gọi DB. Repository là thằng trực tiếp tương tác, đọc ghi dữ liệu trong DB và trả cho service.

Luồng đi của ứng dụng MVC qua các tầng

* Người dùng điền vào tên website mình cần truy cập trên browser sau đó bấm enter. Lúc này mình gửi 1 request lên server nơi mà mình triển khai ứng dụng Spring.
* Thành phần DispatcherServlet của Spring MVC sẽ nhận được request (yêu cầu) của người dùng ở bước 1. Dispatcher là thành phần quan trọng nhất trong springmvc. Nó sẽ là nơi đầu tiên nhận request từ client sau đó sẽ chuyển request đó tới các controller tương ứng, đồng thời sẽ là chốt chặn cuối cùng trả về kết quả cho client.
* Sau khi nhận được request(yêu cầu) DispatcherServlet sẽ chuyển yêu cầu đó tới Controller bằng các cơ chế mapping mà ta khai báo trong Handler Mapping.

1. SPRING BOOT

**3.1 Spring framework**

Spring là một framework được ra đời để giúp các nhà phát triển có thể xây dựng hệ thống và chạy ứng dụng trên JVM một cách thuận tiện, đơn giản và nhanh chóng. Đây là một mã nguồn mở được phát triển và rất nhiều người sử dụng. Trên thực tế, Spring framework là tập hợp gồm rất nhiều các dự án nhỏ khác nhau như: Spring MVC (sử dụng để xây dựng các ứng dụng trên nền tảng web), Spring Data, Spring Boot.

Để phát triển một ứng dụng web cơ bản sử dụng [Spring framework](https://itnavi.com.vn/blog/spring-la-gi-spring-framework-cua-java/), bạn cần trải qua ít nhất 5 công đoạn sau:

* Tạo một project sử dụng Maven với các dependency cần thiết.
* Tạo một tập tin web đuôi (.xml) để khai báo DispatcherServlet
* Một tập tin có cấu hình của Spring MVC.
* Trả về một class Controller khi có request đến.
* Cuối cùng là phải có một web server phục vụ triển khai ứng dụng lên chạy.

Để khởi tạo một dự án Spring khá là vất vả và tốn nhiều công sức. Người lập trình phải khai báo những dependency trong pom.xml, cấu hình XML hoặc các annotation cực kỳ phức tạp. Giờ đây, việc tạo ra các ứng dụng này sẽ được thực hiện một cách đơn giản, nhanh chóng hơn rất nhiều chỉ với Spring Boot.

**3.2 Khái niệm về Spring Boot**

Từ những thông tin ở trên, ta có thể khẳng định Spring Boot chính là một Java framework siêu to và khổng lồ và có nhiều khả năng hữu ích vì nó có thể giúp lập trình viên giải quyết rất nhiều vấn đề. So với framework Spring thông thường, Spring Boot tỏ ra những lợi thế vượt trội. Khi sử dụng Spring Boot, rất nhiều thứ được cải tiến hỗ trợ lập trình viên như:

* Auto config: tự động cấu hình thay lập trình viên, bạn chỉ cần viết code và tiến hành chạy hệ thống là được.
* Dựa trên các Annotation để tạo lập các bean thay vì XML.
* Server Tomcat có thể được nhúng ngay trong file JAR build ra và có thể chạy ở bất kì đâu mà java chạy được.

Khi sử dụng Spring Boot, lập trình viên chỉ cần:

* Sử dụng Spring Initializr: nhập các thông tin của dự án (project), chọn thư viện (Library) rồi tải code về máy.
* Mở mã nguồn (source code) và bắt đầu viết code.
* Có thể chạy ngay trong IDE, hoặc build thành file JAR mà không cần cấu hình config cho server nữa.

**Ưu điểm nổi bật của Spring Boot**

* Phát triển web một cách đơn giản và nhanh chóng.
* Config an toàn
* Có thể hỗ trợ YAML
* Có tính quản trị cao
* Dễ dàng ứng dụng Spring và các sự kiện.
* Có thể cấu hình ở bên ngoài và tạo ra những tệp thuộc tính.
* Tính bảo mật cao.
* Khả năng ghi log

3.3 Maven



Maven là một công cụ mã nguồn mở, thực hiện chức năng quản lý, xây dựng và triển khai cho các dự án, được phát triển bởi Apache Software Foundation. Công cụ này được viết bằng ngôn ngữ Java để xây dựng các dự án được viết bằng C#, Ruby, Scala và các ngôn ngữ khác. **Một số tính năng của Maven:**

* Liên tục cập nhật và phát triển kho lưu trữ thư viện người dùng lớn.
* Khả năng thiết lập dự án dễ dàng, ưu tiên sử dụng các phương pháp đem lại hiệu suất tối ưu nhất cho dự án.
* Quản lý phần phụ thuộc, có tính năng cập nhật tự động.
* Tương thích ngược với các phiên bản trước.
* Báo cáo các lỗi nghiêm trọng và tính toàn vẹn.
* Thiết lập tính nhất quán được đảm bảo trên tất cả các dự án.
* Maven có thể mở rộng, bạn dễ dàng viết các phiên bản bổ trợ bằng ngôn ngữ sẵn có hoặc Java.
* **Các giai đoạn hoạt động của Maven**

Hoạt động của Maven thực hiện được tính theo mỗi chu kỳ. Trong một chu kỳ, Maven cần đảm bảo thực hiện một số giai đoạn cụ thể để dễ dàng trong việc kiểm soát và mang lại hiệu quả tích cực.

* **Xác thực:** Xác thực dự án là chính xác và tất cả các thông tin cần thiết đều có sẵn.
* **Biên dịch:** Biên dịch mã nguồn của dự án.
* **Kiểm tra:** Kiểm tra mã nguồn đã biên dịch bằng cách sử dụng khung kiểm đơn vị phù hợp.
* **Gói:** Lấy mã ở biên dịch và đóng mã ở định dạng có thể phân phối thành các tệp JAR hoặc ZIP.
* **Xác minh:** Thực hiện quy trình kiểm nghiệm đối với kết quả của các bài kiểm tra tích hợp để đảm bảo các tiêu chí về mặt chất lượng được đám ứng
* **Cài đặt:** Cài đặt gói và kho lưu trữ cục bộ để sử dụng như một phần phụ thuộc trong các dự án khác cục bộ.
* **Triển khai:** Sao chép gói cuối cùng vào kho lưu trữ từ xa để chia sẻ với các nhà phát triển hoặc các dự án khác.
* **Thành phần của Maven**

**Maven POM**

Mọi dự án sử dụng Maven đều có tệp POM nằm trong thư mục gốc. pom.xml mô tả các phần phụ thuộc của dự án và cho bạn biết cách xây dựng như thế nào, phần phụ thuộc là phần mềm của bên thứ ba sử dụng trong dự án.

**Thư mục Maven**

Thư mục Maven thực hiện những gì được gọi là quy ước về cấu hình, một giải pháp sử dụng để cấu hình. Thay vì yêu cầu các nhà phát triển xác định bố cục và cấu hình thủ công các thành phần cho mỗi dự án mới, Maven thiết lập một cấu trúc dự án chung và cung cấp một định dạng tệp tiêu chuẩn để mô tả cách hoạt động.

**Kho lưu trữ**

Maven sử dụng kho lưu trữ tập trung để tìm và xuất các gói dự án dưới dạng phần phụ thuộc. Khi bạn tham chiếu đến phần phụ thuộc trong dự án, Maven sẽ phát hiện và tải các gói dự án xuống kho lưu trữ cục bộ và cài đặt vào dự án của bạn.

1. CÁC THÀNH PHẦN CỦA RESTful API

4.1 API là gì ?

API (Application Programming Interface) là một tập hợp các định nghĩa và giao thức để xây dựng, tích hợp các phần mềm ứng dụng. Khái niệm này đôi khi còn được xem như một “hợp đồng” giữa người cung cấp thông tin và người sử dụng thông tin – thiết lập nội dung được yêu cầu từ người dùng (call) và từ producer (response)

Nói cách khác, nếu muốn tương tác với một máy tính hay hệ thống để truy xuất thông tin thì API sẽ giúp người dùng giao tiếp với hệ thống để nó hiểu và đáp ứng các yêu cầu.

Dựa vào đặc điểm của API, các tổ chức có thể chia sẻ tài nguyên và thông tin mà vẫn có thể kiểm soát bảo mật, quá trình xác thực và quyền truy cập vào các tài nguyên đó. Bên cạnh đó, việc sử dụng API cũng không yêu cầu chi tiết về caching, tức là cách tài nguyên được truy xuất hoặc nguồn gốc của các tài nguyên này.

4.2 RESTful là gì?

REST là một tập hợp các ràng buộc về kiến trúc. Các developer API có thể triển khai REST theo nhiều cách khác nhau. Khi một client request được tạo thông qua RESTful API, nó sẽ chuyển một biểu diễn trạng thái của tài nguyên đến người yêu cầu (hoặc endpoint). Thông tin này sau đó sẽ được chuyển ở một trong số các định dạng như: JSON, HTML, XLT, Python, PHP hay plain text. JSON là định dạng file phổ biến nhất vì là một ngôn ngữ bất khả tri, có thể được đọc bởi cả con người lẫn máy.

Cần lưu ý rằng các header và tham số cũng vô cùng quan trọng trong phương thức HTTP của một request HTTP RESTful API. Sở dĩ vì chúng chứa thông tin định danh quan trọng đối với metadata, authorization, URI, caching, cookies,… của request.

Header gồm có hai loại: Request header và response header, mỗi loại đều có thông tin kết nối HTTP và status code riêng.

* Cách thức hoạt động của RESTful API
* RESTful API chia một transaction (giao dịch) ra thành nhiều module nhỏ, mỗi module giải quyết một phần cơ bản của transaction. Việc này giúp tăng tính linh hoạt nhưng đôi khi lại tương đối khó khăn cho các developer khi muốn thiết kế REST API từ đầu
* Một **RESTful API** sử dụng các câu lệnh để lấy tài nguyên, trạng thái của tài nguyên ở bất kỳ timestamp nào được gọi là một biểu diễn của tài nguyên đó. Các phương thức HTTP mà RESTful API sử dụng được xác định bởi giao thức RFC 2616:
* **GET:**Trả về một tài nguyên.
* **PUT:**Thay đổi trạng thái hoặc cập nhật tài nguyên (có thể là đối tượng, file hay block).
* **POST:**Tạo tài nguyên.
* **DELETE:**Xoá một tài nguyên.
* Một số định dạng dữ liệu được RESTful API hỗ trợ:
* application/json.
* application/xml.
* application/x-wbe+xml.
* application/x-www-form-urlencoded.
* multipart/form-data.

4.3 HTTP Method

HTTP request : là khi client truy cập vào trang web client sẽ gửi một yêu cầu (request) lên server để nói server là client cần những thông tin gì Khi server nắm được yêu cầu (request) của client thì nó sẽ trả lại thông tin tương ứng với yêu cầu của client.

* **GET**:được sử dụng để lấy thông tin từ server theo URL đã cung cấp, sẽ không có phần body
* **POST**: thêm mới, gửi thông tin tới server thông qua các biểu mẫu http
* **PUT**: cập nhật tài nguyên
* **DELETE**: xóa tài nguyên trên server.
* STATUS CODE
* **200 OK**: Trả về thành công cho những phương thức GET, PUT, PATCH hoặc DELETE.
* **201 Created:** Trả về khi tạo thành công một Resouce.
* **204 No Content:** Trả về khi xóa thành công Resource.
* **304 Not Modified:** Client có thể sử dụng dữ liệu cache.
* **400 Bad Request:** Request không hợp lệ.
* **401 Unauthorized:**Request cần phải có auth.
* **403 Forbidden:** Bị từ chối không cho phép.
* **404 Not Found:** Không tìm thấy resource từ URI.
* **405 Method Not Allowed:** Phương thức không cho phép với user hiện tại.
* **410 Gone:** Resource không còn tồn tại, version cũ không còn hỗ trợ.
* **415 Unsupported Media Type:** Không hỗ trợ kiểu Resource này.
* **422 Unprocessable Entity:** Dữ liệu không được xác thực.
* **429 Too Many Requests:**Request bị từ chối do bị giới hạn.

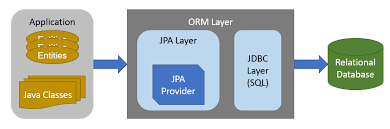
1. TỔNG QUAN VỀ JAVA PERSISTENCE API – JPA

5.1 Giới thiệu JPA

JPA là viết tắt của **Java Persistence API**, nó là một đặc tả Java cho việc ánh xạ giữa các đối tượng Java với cơ sở dữ liệu quan hệ sử dụng công nghệ phổ biến là ORM (Object Relational Mapping).

JPA cung cấp đầy đủ các công cụ cho phép chúng ta có thể thao tác với cơ sở dữ liệu một cách đơn giản và nhanh chóng. JPA có thể dùng để persist một đối tượng Java (POJO – Plain Old Java Object) vào trong cơ sở dữ liệu hoặc lấy dữ liệu từ cơ sở dữ liệu và ánh xạ (mapping) ra các đối tượng Java một cách đơn giản.

JPA hoạt động như một cầu nối giữa các table/ các mối quan hệ giữa các table trong database và các class/ mối quan hệ giữa các object. Ví dụ: table USER với các column (Id, username, password) sẽ tương ứng với class User.java với các field Id, username, password. Từ đó mỗi khi truy vấn table hay các column ta sẽ gọi trực tiếp các phương thức trên các class, các field của class mà không cần quan tâm tới việc đang dùng loại database nào, kiểu dữ liệu database ra sao

****

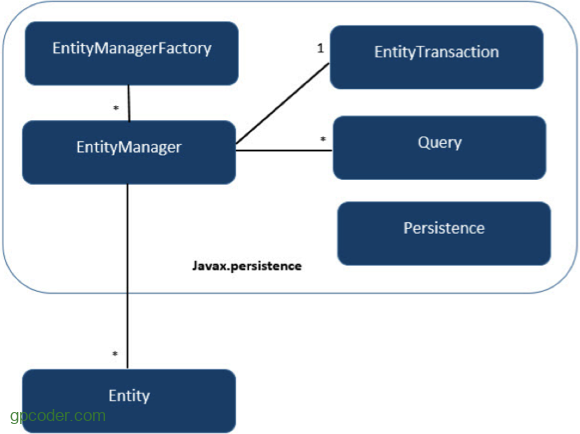
ORM là viết tắt của Object Relational Mapping, là một công nghệ/ khái niệm/ quá trình chuyển đổi dữ liệu từ ngôn ngữ hướng đối tượng sang Database quan hệ và ngược lại. Trong Java nó được thực hiện với sự trợ giúp của Reflection và JDBC.

ORM có khả năng xử lý các thao tác của nhiều loại cơ sở dữ liệu khác nhau một cách dễ dàng mà không quan tâm đến loại database sử dụng (SQL Server, MySQL, PostgreSQL, …) hay loại thao tác sử dụng (INSERT, UPDATE, DELETE, SELECT, …)

* Lợi ích của việc sử dụng JPA
* Đơn giản hóa công nghệ cho tầng persistence (tầng dữ liệu).
* Không phụ thuộc vào các framework ORM.
* Có nhiều nhà cung cấp hỗ trợ cài đặt JPA.
* Dữ liệu có thể được lưu trữ thông qua việc ORM.

5.2 Kiến trúc JPA

JPA sử dụng metadata để ánh xạ các đối tượng persistence với các bảng trong cơ sở dữ liệu. JPA hỗ trợ SQL như là một ngôn ngữ truy vấn để dễ dàng xử lý các truy vấn cơ sở dữ liệu. Ngôn ngữ truy vấn JPA có thể dùng thực thi cả truy vấn tĩnh và truy vấn động. JPA bao gồm các thành phần chính là:

* **Entity**: là các đối tượng thể hiện tương ứng 1 table trong cơ sở dữ liệu. Entity thường là các class POJO đơn giản, chỉ gồm các phương thức getter, setter.
* **EntityManager**: EntityManager là một interface cung cấp các API cho việc tương tác với các Entity.
* **EntityManagerFactory**: EntityManagerFactory được dùng để tạo ra một instance của EntityManager
* **EntityTransaction**: Một Transaction là một tập hợp các thao tác trong đó tất cả các thao tác phải được thực hiện thành công hoặc tất cả thất bại. EntityTransaction có quan hệ 1-1 với EntityManager. Bất kỳ thao tác nào được bắt đầu thông qua đối tượng EntityManager đều được đặt trong một Transaction. Đối tượng EntityManager giúp tạo EntityTransaction.
* **Persistence**: Một Persistence định nghĩa một tập hợp các Entity class được quản lý bởi 1 instacne của EntityManager trong ứng dụng.Persistence (javax.persistence.Persistence) class bao gồm các phương thức static để lấy instance của EntityManagerFactory.
* **Query**: là một interface, được mỗi nhà cung cấp JPA implement để có được các đối tượng quan hệ đáp ứng các tiêu chí (criteria) truy vấn.
* Tính năng của JPA
* JPA hỗ trợ plugable, tức là có thể sử dụng nhiều 3rd khác nhau như Hibernate hay MyBatis.
* Hỗ trợ cấu hình thông qua annotation và xml.
* Giảm bớt số lớp yêu cầu cho việc phát triển persistence.
* Không cần phải viết các mô tả triển khai trong XML. Các Annotation dựa trên metadata đã hỗ trợ trong các ứng dụng JPA.
* Đã chuẩn hóa ORM và dễ dàng phát triển hơn.
* JPA hỗ trợ truy vấn động và tĩnh.
* Nhiều IDE hỗ trợ phát triển ứng dụng JPA và có thể tự động sinh code ánh xạ từ cơ sở dữ liệu thành các entity và ngược lại.

1. GIỚI THIỆU VỀ HIBERNATE

6.1 Hibernate Framework

Hibernate framework là một giải pháp ORM (Object Relational Mapping) mã nguồn mở, gọn nhẹ. Hibernate giúp đơn giản hoá sự phát triển của ứng dụng java để tương tác với cơ sở dữ liệu.

Tool ORM giúp đơn giản hoá việc tạo ra dữ liệu, thao tác dữ liệu và truy cập dữ liệu. Đó là một kỹ thuật lập trình để ánh xạ đối tượng vào dữ liệu được lưu trữ trong cơ sở dữ liệu.

6.2 Kiến trúc Hibernate

Kiến trúc Hibernate bao gồm nhiều đối tượng như đối tượng persistent, session factory, transaction factory, connection factory, session, transaction. Có 4 tầng trong kiến trúc hibernate đó là tầng ứng dụng java, tầng hibernate framework, tầng backhand api và tầng database. Dưới đây là sơ đồ kiến trúc hibernate:|



**Đối tượng Configuration**

Đối tượng Configuration là đối tượng Hibernate đầu tiên bạn tạo trong bất kỳ ứng dụng Hibernate nào và chỉ cần tạo một lần trong quá trình khởi tạo ứng dụng. Nó đại diện cho một tập tin cấu hình hoặc thuộc tính yêu cầu của Hibernate. Đối tượng Configuration cung cấp hai thành phần chính:

* Database Connection: Thao tác này được xử lý thông qua một hoặc nhiều tệp cấu hình được Hibernate hỗ trợ. Các tệp này là hibernate.properties và hibernate.cfg.xml.
* Class Mapping Setup: Thành phần này tạo ra kết nối giữa các lớp Java và các bảng cơ sở dữ liệu.

**Đối tượng SessionFactory**

Đối tượng Configuration được sử dụng để tạo ra một đối tượng SessionFactory mà cấu hình Hibernate cho ứng dụng bằng cách sử dụng tệp tin cấu hình được cung cấp và cho phép một đối tượng Session được khởi tạo. SessionFactory là một đối tượng luồng an toàn và được sử dụng bởi tất cả các luồng của một ứng dụng.

SessionFactory là đối tượng nặng nên thường nó được tạo ra trong quá trình khởi động ứng dụng và lưu giữ để sử dụng sau này. Bạn sẽ cần một đối tượng SessionFactory cho mỗi cơ sở dữ liệu bằng cách sử dụng một tập tin cấu hình riêng biệt. Vì vậy, nếu bạn đang sử dụng nhiều cơ sở dữ liệu thì bạn sẽ phải tạo nhiều đối tượng SessionFactory.

**Đối tượng Session**

Một session được sử dụng để có được một kết nối vật lý với một cơ sở dữ liệu. Đối tượng Session là nhẹ và được thiết kế để được tạo ra thể hiện mỗi khi tương tác với cơ sở dữ liệu. Các đối tượng liên tục được lưu và truy xuất thông qua một đối tượng Session.

Các đối tượng Session không nên được mở trong một thời gian dài bởi vì chúng thường không phải là luồng an toàn và chúng cần được tạo ra và được đóng khi cần thiết.

**Đối tượng Transaction**

Một Transaction đại diện cho một đơn vị làm việc với cơ sở dữ liệu và hầu hết các RDBMS hỗ trợ chức năng transaction. Các transaction trong Hibernate được xử lý bởi trình quản lý transaction và transaction (từ JDBC hoặc JTA).

**Đối tượng Query**

Các đối tượng Query sử dụng chuỗi truy vấn SQL hoặc Hibernate Query Language (HQL) để lấy dữ liệu từ cơ sở dữ liệu và tạo các đối tượng. Ví dụ truy vấn được sử dụng để ràng buộc các tham số truy vấn, giới hạn số lượng kết quả được trả về bởi truy vấn và cuối cùng thực hiện truy vấn.

**Đối tượng Criteria**

Đối tượng Criteria được sử dụng để tạo và thực hiện truy vấn các tiêu chí định hướng đối tượng để lấy các đối tượng.

1. JSON WEB TOKEN

7.1 Khái niệm

JSON Web Token (JWT) là 1 tiêu chuẩn mở (RFC 7519), định nghĩa cách thức truyền tin an toàn giữa các ứng dụng bằng một đối tượng JSON. Dữ liệu truyền đi sẽ được mã hóa và chứng thực, có thể được giải mã để lấy lại thông tin và đánh dấu tin cậy nhờ vào “chữ ký” của nó. Phần chữ ký của JWT sẽ được mã hóa lại bằng HMAC hoặc RSA.

Đặc điểm

* Gọn nhẹ (compact): JWT có thể được truyền đi thông qua URL, hoặc qua giao thức POST, hay gán vào bên trong phần HTTP Header. Kích thước nhỏ hơn ứng với công việc truyền tải sẽ nhanh hơn. Dưới đây là cách thức truyền token vào trong HTTP Header sử dụng Bearer Schema:

**Authorization: Bearer <token>**

* Tự đóng gói (self-contained): Payload của JWT đã chứa các thông tin cần thiết về user, thay vì phải truy vấn cơ sở dữ liệu nhiều lần.

7.2 Thành phần của JWT

**Header**

Gồm 2 phần: type của token, giá tri là JWT, và phương thức mã hóa (HMAC SHA256 hay RSA). JSON object này sau đó được mã hóa Base64URL.

**Payload**

Payload chức các thành phần gọi là claim. Các claim này chứa các thông tin về đối tượng (thường là user), và các meta data của token. Server sẽ dùng các thông tin từ các claim này để có thể thực hiện authentication. Có 3 loại claim: reserved, public và private.

* **Reserved claim**

Là những thông tin đã được quy định ở trong [IANA JSON Web Token Claims registry](https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32#section-10.1). Các claim này là không bắt buộc phải có trong token, nhưng có thể cung cấp các thông tin hữu ích. Các claim này chỉ gồm 3 ký tự vì mục đích giảm kích thước của Token.

**iss** (issuer): tổ chức phát hành token.

**sub**(subject): chủ đề của token.

**aud**(audience): đối tượng sử dụng token.

**exp**(expired time): thời điểm token sẽ hết hạn.

**nbf**(not before time): token sẽ chưa hợp lệ trước thời điểm này.

**iat**(issued at): thời điểm token được phát hành, tính theo UNIX time.

**jti**: JWT ID.

* **Public claim**

Là những claim được định nghĩa 1 cách công khai bởi những bên sử dụng JWT. Chúng nên được quy định ở trong IANA JSON Web Token Registry hoặc là 1 URI có chứa không gian tên không bị trùng lặp

**Signature**

Signature được tạo ra bằng cách dùng phương pháp mã hóa được chỉ định ở header để mã hóa nội dung encode của header, payload, cùng với chuỗi khóa bí mật.

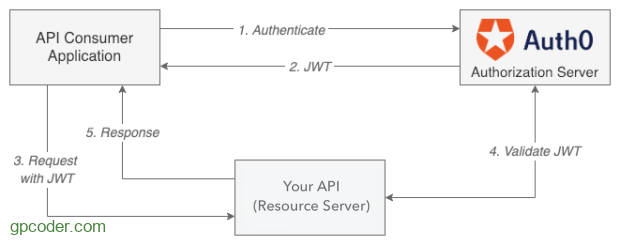
Phương thức mã hóa có thể là HMAC hay RSA:

**HMAC**: đối tượng khởi tạo JWT (token issuer) và đầu nhận JWT (token verifier) sử dụng chung 1 mã bí mật để mã hóa và kiểm tra.

**RSA**: sử dụng 1 cặp key, đối tượng khởi tạo JWT sử dụng Private Key để mã hóa, đầu nhận JWT sử dụng Public Key để kiểm tra.

7.3 Cách thức hoạt động của JWT

* Application hoặc client gửi thông tin chứng thực lên server (có thể là username/ password).
* Khi đã chứng thực thành công, authorization server sẽ trả về một access token (JWT) cho client.
* Phía Client sẽ gửi token này kèm theo request để truy cập resource. Thông thường token được gửi trong header: **Authorization: Bearer <token>**



1. MỘT SỐ ANOTATION TRONG SPRING

[Spring dependency injection](https://shareprogramming.net/gioi-thieu-inversion-of-control-va-dependency-injection-trong-spring/): là một trong những tính năng cốt lỗi của Spring Framework cho phép khởi tạo, quản lý và lắp ráp các bean trong toàn ứng dụng. Để sử dụng những tính năng này chúng ta phải thông qua các annotation được cung cấp trong org.springframework.beans.factory.annotation and org.springframework.context.annotation package

**@Configuration**

Được sử dụng để chỉ ra rằng class khai báo sử dụng annotation @Configuration sẽ khai báo một hoặc nhiều @Bean method trong class đó. Những class khai báo với @Configuration sẽ được Spring container quản lý và tạo bean trong lúc chương trình đang chạy. Thông thường các bean cấu hình cho dự án ta để trong này. Ví dụ cấu hình thymeleaf, đa ngôn ngữ , và nhiều cấu hình khác cho ứng dụng.

|  |
| --- |
| **@Configuration**  **public class WebConfig implements WebMvcConfigurer {**  **@Bean**  **@Description("Thymeleaf template resolver serving HTML 5")**  **public ClassLoaderTemplateResolver templateResolver() {**  **var templateResolver = new ClassLoaderTemplateResolver();**  **templateResolver.setPrefix("mytemplates/");**  **templateResolver.setCacheable(false);**  **templateResolver.setSuffix(".html");**  **templateResolver.setTemplateMode("HTML5");**  **templateResolver.setCharacterEncoding("UTF-8");**  **return templateResolver;**  **}** |

**@Bean**

Method (phương thức) sử dụng @Bean ở phía trên mình để chỉ ra rằng . Method đó sẽ sản xuất ra đối tượng bean và được quản lý bởi spring container . Bean annotation có thể sử dụng với các tham số như name, initMethod hoặc destroyMethod, dùng Bean để tạo ra object Spring Template:

|  |
| --- |
| **@Bean**  **@Description("Thymeleaf template engine with Spring integration")**  **public SpringTemplateEngine templateEngine() {**  **var templateEngine = new SpringTemplateEngine();**  **templateEngine.setTemplateResolver(templateResolver());**  **return templateEngine;**  **}** |

**@ComponentScan**

Sử dụng **@ComponentScan** để thông báo Spring Container biết phải vào package nào trong dự án để quét các **Annotation** và tạo **Bean**. Như ví dụ bên dưới. Spring sẽ quét tất cả các file trong package levunguyen.spring. Tìm các Class có annotation để tạo bean và các **@Autowire** để nhúng bean ở trong container vào các Class sử dụng autowire

|  |
| --- |
| **@ComponentScan(basePackages = "levunguyen.spring ")**  **@Configuration**  **public class SpringComponentScanApp {**  **// ...**  **}** |

**@Component**

Khi một class được đánh dấu là @Component thì sẽ được tạo thành 1 bean. Khi Spring start thì nó quét qua các annotation có đánh dấu là @Component thì nó sẽ tạo bean cho class đó. Ví dụ ta có class Contact và ta đánh dấu nó là @Component thì Spring khi đọc qua class này nó sẽ tạo 1 bean có tên là contact trong container của nó. Nếu có class nào dùng thì nó sẽ nhúng bean này vào. Dùng @component là để tạo ra một bean

|  |
| --- |
| **@Component**  **@Scope("request")**  **public class Contact {**  **//…**  **}** |

**@PropertySource và @Value**

Sử dụng **@PropertySource** để cho Spring biết tìm các file properties cấu hình cho hệ thống ở đâu đồng thời sử dụng **@Value** để lấy các giá trị trong file properties

Ví dụ bên dưới ta sử dụng classpath để khai báo file properties ta đặt ở đâu trong dự án. Tiếp đến ta sử dụng @Value để lấy các giá trị trong file properties với key tương ứng và gán vào biến mà ta sẽ sử dụng.

|  |
| --- |
| **@Configuration**  **@ComponentScan(basePackages = { "demoAnotation.\*" })**  **@PropertySource("classpath:config.properties")**  **public class AppConfigMongoDB {**  **@Value("${mongodb.url}")**  **private String mongodbUrl;**  **@Value("${mongodb.db}")**  **private String defaultDb;** |

**@Autowired**

Tự động nhúng các bean được Spring Container sinh ra vào Class có khai báo **@Autowired**. Khi Spring nó sẽ tìm kiếm bean có tên là BookDao trong container của nó, sau đó nhúng vào lớp BookServiceImple. Đây chính là cơ chế **DI** (dependency injection). Khi Spring bắt đầu chạy nó sẽ quét các lớp có sử dụng annotation để tạo bean đồng thời nó cũng quét bên trong các bean xem có khai báo @Autowired không nếu có sẽ tìm kiếm bean tương ứng mà nó quản lý và nhúng vào.

|  |
| --- |
| **@Service**  **public class BookServiceImpl implements BookService {**  **@Autowired**  **private BookDao bookDao;**  **@Autowired**  **private CustomerDao customerDao;**  **...**  **}** |

**@Scope**

Được dùng để định nghĩa phạm vi của một **@Component** class hay một **@Bean** method trong Spring. Các phạm vi có thể có là singleton, prototype, request, session, globalSession, và một số phạm vi được tuỳ biến.

|  |
| --- |
| **@Component**  **@Scope("prototype")**  **class Engine {}** |

**@Service**

Nếu một class được đánh dấu là @Service nó là kiểu đặt biệt của @Component. Được dùng để xử lý các nghiệp vụ của ứng dụng. Ví dụ như kế toán thì có nghiệp vụ là kiểm tra chi, quản lý thu. Lớp BookServiceImpl dưới đây được đánh dấu là @Service thì nó sẽ phụ trách xử lý các vấn đề liên quan đến nghiệp vụ.

|  |
| --- |
| **@Service**  **public class BookServiceImpl implements BookService {**  **//…}** |

**@Repository**

Nếu một class được đánh dấu là **@Repository** thì nó là kiểu đặt biệt của **@Component** . Nó được sử dụng để nói bean này dùng để truy cập và thao tác xuống cơ sở dữ liệu. Class BookDaoImpl được đánh dấu với **@Repository** nghĩa là lớp này có nhiệm vụ thực hiện các câu lệnh truy vấn xuống database.

|  |
| --- |
| **@Repository**  **public class BookDaoImpl implements BookDao {**  **//….**  **}** |

**@Controller**

Một class được đánh dấu là **@Controller** thì để khai báo Class đó là một controller và có nhiệm vụ mapping request trên url vào các method tương ứng trong controller. Ví dụ: ta khai báo Class HomeController là một Controller . Khi người dùng gõ vào http://localhost:8080/ thì sẽ được xử lý bởi Class HomeController. Như vậy nhiệm vụ của Controller là điều hướng các request (yêu cầu) người dùng vào method xử lý tương

|  |
| --- |
| **@Controller**  **public class HomeController {**  **@RequestMapping("/")**  **public String homePage() {**  **return "home";**  **}**  **}** |

**@RequestMapping**

Có nhiệm vụ ánh xạ các request (yêu cầu) người dùng vào method tương ứng trong controller. Ví dụ : Khi nhập vào url là http://localhost:8080/method2 thì nó sẽ được xử lý bởi phương thức là public String method2().

Ví dụ : Khi ta nhập vào url là http://localhost:8080/method3 thì nó sẽ được xử lý bởi phương thức là public String method3().

|  |
| --- |
| **/Xử lý cho request có phương thức http là post và url hoặc action method là "/home/method2**  **@RequestMapping(value = "/method2", method = RequestMethod.POST)**  **public String method2() {**  **return "method2";**  **}**  **//Xử lý cho request có phương thức http là post hoặc get có url hoặc**  **// form action method là "/home/method3**  **@RequestMapping(value = "/method3", method = {RequestMethod.POST, RequestMethod.GET})**  **public String method3() {**  **return "method3";**  **}** |

**@PathVariable**

**@PathVariable** được sử dụng để xử lý những URI động, có một hoặc nhiều parameter trên URI.

|  |
| --- |
| **@RequestMapping("/test2/{id}/{name}")**  **public String test2(@PathVariable("id") int id, @PathVariable("name") String name, Model model) {**  **model.addAttribute("id", id);**  **model.addAttribute("name", name);**  **return "test2";**  **}** |

**@RequestParam**

Chúng ta sử dụng **@RequestParame** để bắt các giá trị các tham số mà người dùng truyền vào trên url theo định dạng key và value

Ví dụ với link sau <http://localhost:8080/api/foos?id=abc>, để lấy giá trị abc của tham số id trên url thì sẽ dùng **@RequestParam**. Khai báo giá trị tham số trên URL theo định dạng key = value (id=abc).

|  |
| --- |
| **@GetMapping("/api/foos")**  **@ResponseBody**  **public String getFoos(@RequestParam String id) {**  **return "ID: " + id;**  **}** |

**@Valid**

Dùng để kiểm tra dữ liệu có đúng như mình mong muốn hay không. Ví dụ dưới đây mình mong muốn name là không được rỗng , author không được rỗng. Nếu dữ liệu bị rỗng thì @Validate sẽ bắt lỗi.

|  |
| --- |
| **@Entity**  **public class Book {**  **@Id**  **@GeneratedValue**  **private Long id;**  **@NotEmpty(message = "Please provide a name")**  **private String name;**  **@NotEmpty(message = "Please provide a author")**  **private String author;**  **@NotNull(message = "Please provide a price")**  **@DecimalMin("1.00")**  **private BigDecimal price;**  **//...**  **}**  **@RestController**  **public class BookController {**  **@PostMapping("/books")**  **Book newBook(@Valid @RequestBody Book newBook) {**  **return repository.save(newBook);**  **}**  **//...**  **}** |

**@SessionAttribute**

Chúng ta sử dụng @SessionAttribute để lưu trữ các giá trị trong một phiên làm việc. Giống như mình làm một ứng dụng shopping cart . Khi người dùng chọn 1 sản phẩm thì mình dùng session mình lưu lại. Khi khách hàng thanh toán giỏ hàng thì mình lấy hết tất cả các mặt hàng chứa trong session ra và tính toán

|  |
| --- |
| **@Controller**  **@SessionAttributes("shoppingCart")**  **public class AddToCartController {**  **@PostMapping("/addToCart")**  **public String addToCart(final Model model, @ModelAttribute ShoppingCart shoppingCart, final String productCode) {**  **if (shoppingCart != null) {**  **//add product to the shopping cart list**  **shoppingCart.setProduct(productCode);**  **model.addAttribute("cart", shoppingCart);**  **} else {**  **ShoppingCart cart = new ShoppingCart();**  **cart.setCustomerName("Super customer");**  **cart.setProduct(productCode);**  **model.addAttribute("cart", cart);**  **}**  **return "redirect:" + "product-detail-page";**  **}**  **@ModelAttribute("shoppingCart")**  **public ShoppingCart shoppingCart() {**  **return new ShoppingCart();**  **}**  **}** |

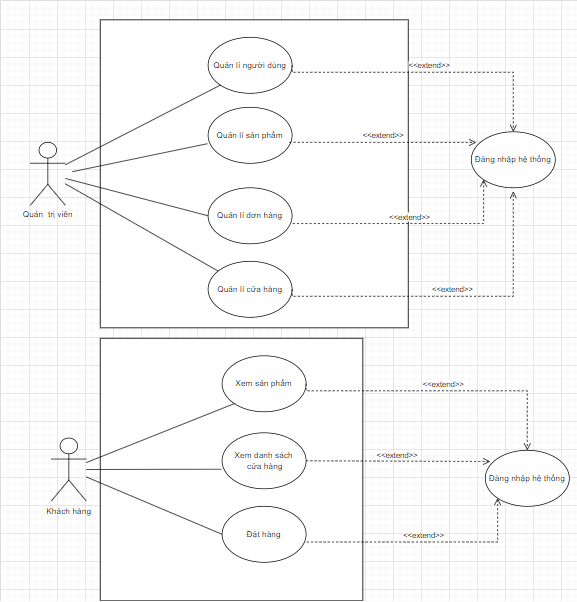
# **XÂY DỰNG HỆ THỐNG**

## PHÂN TÍCH HỆ THỐNG, MÔ TẢ NGHIỆP VỤ

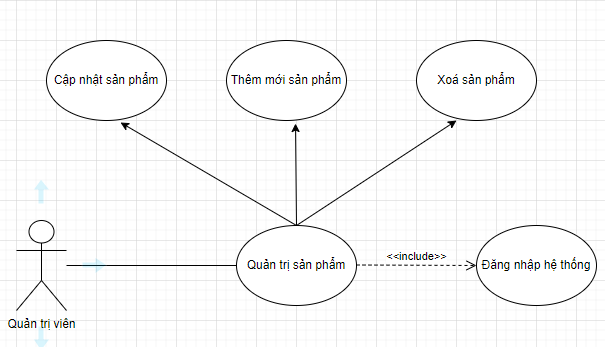
Chức năng quản trị viên

Quản trị viên thực hiện được các công việc như đăng nhập hệ thống, quản lí sản phẩm, loại sản phẩm, đơn hàng, danh sách chuỗi cửa hàng Tous Les Temp. Đồng thời quản trị được danh sách đơn hàng, tình trạng hàng tồn ở mỗi cửa hàng, thống kê doanh thu,…

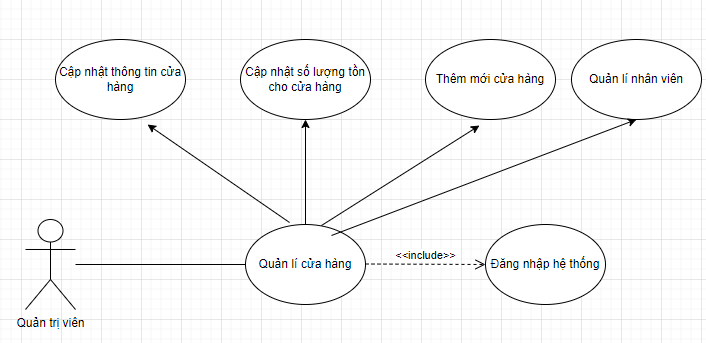
Sơ đồ usecase tổng quát của hệ thống



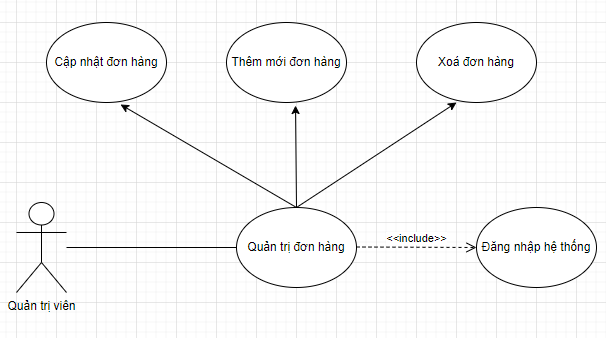
Sơ đồ chức năng quản lí thông tin sản phẩm



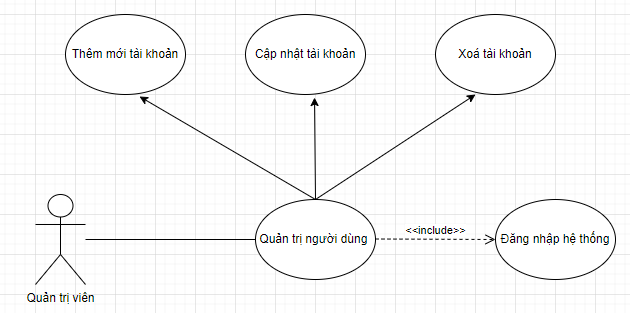
Sơ đồ chức năng quản lí thông tin chuỗi cửa hàng



Sơ đồ chức năng quản lí thông tin đơn hàng



Sơ đồ chức năng quản lí thông tin người dùng



Chức năng người dùng

## YÊU CẦU HỆ THỐNG

2.1 Yêu cầu chức năng

* Đối với quản trị viên
* Xây dựng web application với mục tiêu quản lí chuỗi cửa hàng Tous Les Temp về nghiệp vụ bán hàng, sản phẩm có ở từng cửa hàng, danh sách đơn hàng, danh sách cửa hàng có mặt trên thành phố Nha Trang
* Chức năng xử lí danh sách sản phẩm có ở mỗi cửa hàng: Duyệt xem danh sách, thêm mới, cập nhật số lượng tồn
* Xem tổng số sản phẩm của cửa hàng
* Truy vấn nhanh một số thông tin về sản phẩm bằng các thao tác với bộ lọc
* Tìm một sản phẩm dựa trên tên của sản phẩm đó
* Thực hiện các thao tác hàng loạt với sản phẩm như: xóa sản phẩm, thêm/ xóa sản phẩm trong loại sản phẩm, đăng/ ẩn sản phẩm trong kênh bán hàng, thay đổi giao diện hiển thị chi tiết
* Chức năng quản lí tồn kho: Mỗi một địa chỉ cửa hàng được tạo ra trên hệ thống đều có thể được hiểu là một kho hàng và do đó, chứa các dữ liệu tồn kho tương ứng cho riêng cửa hàng đó.
* Chức năng quản lí đơn hàng: Nhận thông tin đơn hàng và thực hiện xử lí
* Xem tổng số đơn hàng của cửa hàng hiện tại
* Truy vấn nhanh một số thông tin về đơn hàng trên mã đơn hàng, trạng thái đơn hàng
* In danh sách đơn, thống kê
* Chức năng thêm mới khách hàng: Tạo mới tài khoản khách hàng, cho phép đặt hàng,..
* Tìm kiếm sản phẩm: Sau khi tìm kiếm sản phẩm, quản trị viên có thể thực hiện các thao tác cập nhật với sản phẩm đó
* Đối với người dùng
* Chức năng xem tấc cả sản phẩm, xem chi tiết thông tin sản phẩm
* Thực hiện lọc theo một sộ số tiêu chi: Tên sản phẩm, loại sản phẩm,…
* Chức năng đặt hàng thanh toán cơ bản

2.2 Yêu cầu phi chức năng

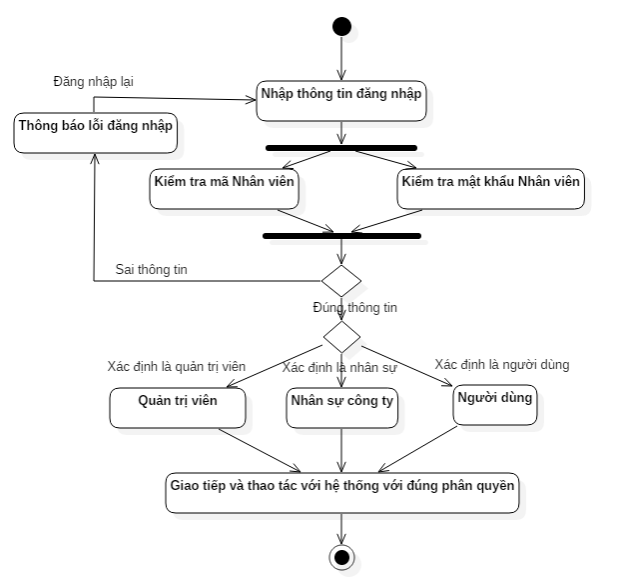
- Dung lượng website vừa phải, tốc độ truy xuất nhanh. Hệ thống cần có chế độ bảo mật, an toàn.

- Khi người dùng thực hiện đăng nhập phải thông qua tên đăng nhập mật khẩu, khi đăng kí cần có đầy đủ thông tin về email, địa chỉ, điện thoại

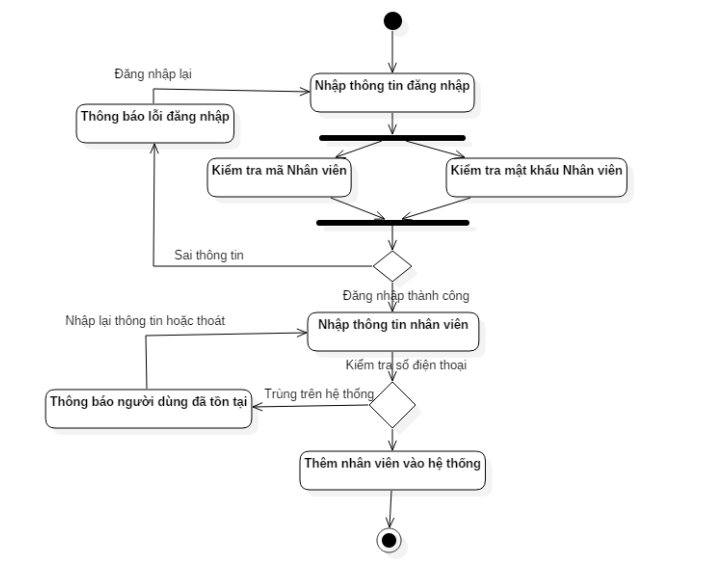
- Cơ sở dữ liệu phải đảm bảo hoạt động khi truy cập hệ thống

## MÔ TẢ USECASE VÀ SƠ ĐỒ HOẠT ĐỘNG

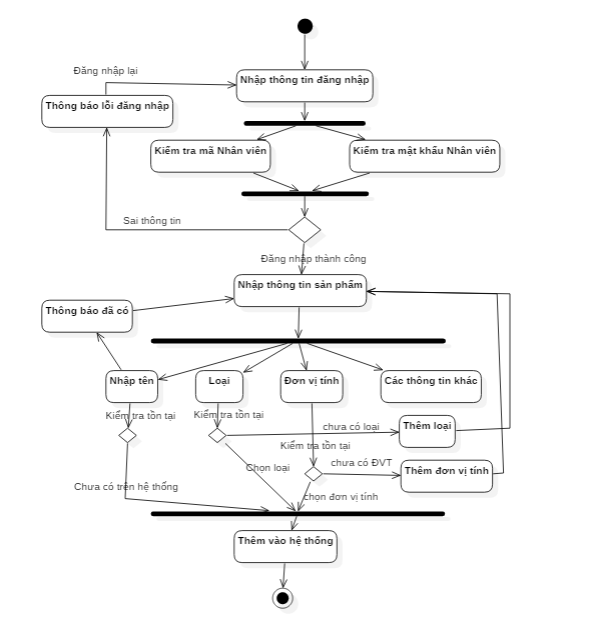
2.1 Sơ đồ hoạt động chức năng đăng nhập



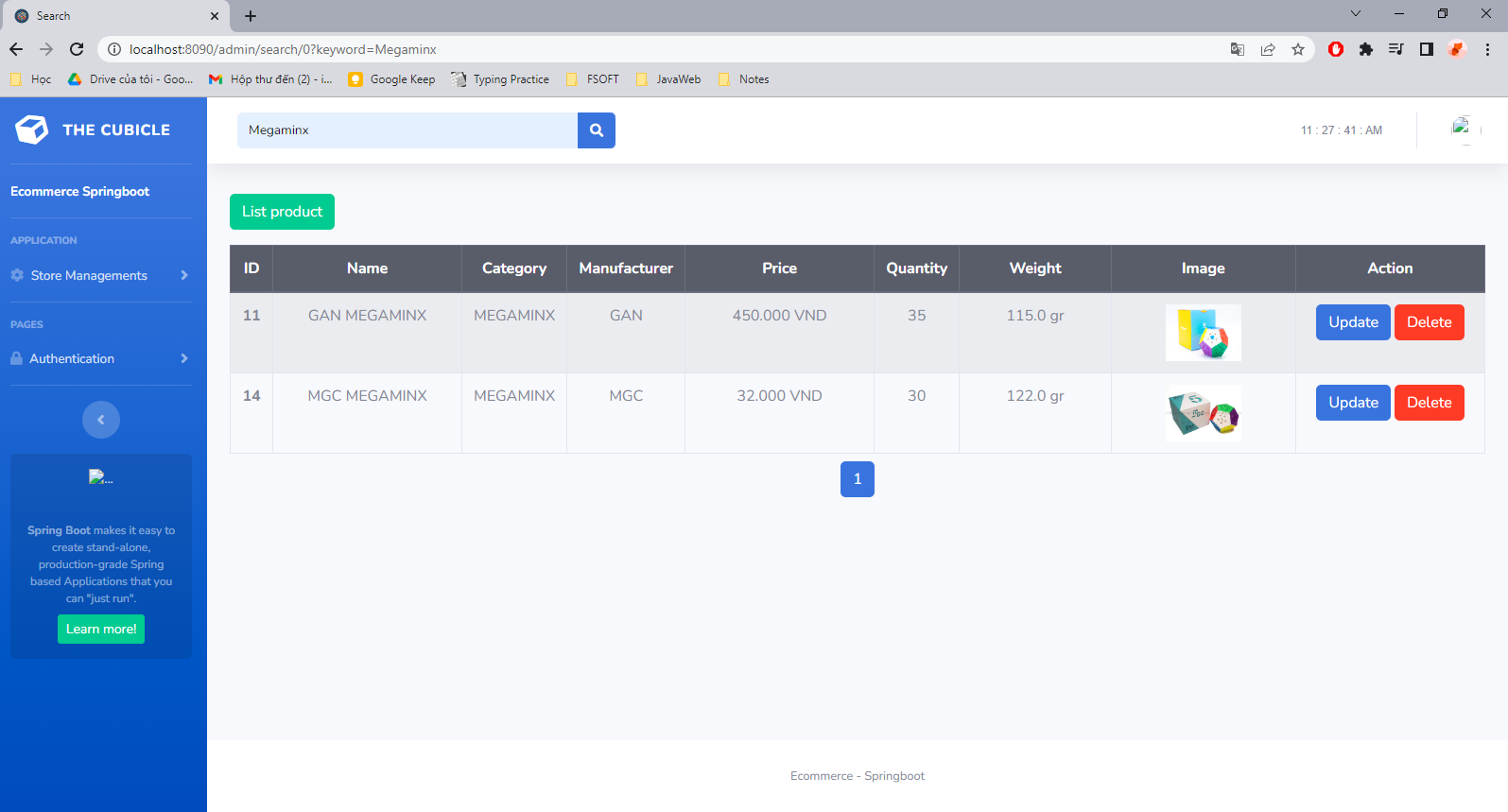
2.2 Chức năng thêm tài khoản



2.3 Chức năng thêm sản phẩm



1. CƠ SỞ DỮ LIỆU
2. BẢNG DỮ LIỆU
3. GIAO DIỆN KẾT QUẢ



Giao diện 9: Tìm kiếm sản phẩm

# **KẾT LUẬN**

# **TÀI LIỆU THAM KHẢO**