TRƯỜNG ĐẠI HỌC CẦN THƠ

KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG

**BỘ MÔN TIN HỌC ỨNG DỤNG**



**NIÊN LUẬN**

**TÌM HIỂU SPRING FRAMEWORK VÀ XÂY DỰNG ỨNG DỤNG MINH HỌA**

**CÁN BỘ HƯỚNG DẪN SINH VIÊN THỰC HIỆN**

Lê Thị Diễm Trần Quốc Sang

MSSV: B1401085

Khóa: K40

***Cần Thơ, tháng/năm***

MỤC LỤC

[Chương 1: TỔNG QUAN 6](#_Toc513768098)

[1.1 LÝ DO CHỌN ĐỀ TÀI 6](#_Toc513768099)

[1.2 ĐẶT VẤN ĐỀ VÀ HƯỚNG GIẢI QUYẾT 7](#_Toc513768100)

[1.3 PHẠM VI ĐỀ TÀI 9](#_Toc513768101)

[1.4 PHƯƠNG PHÁP NGHIÊN CỨU 9](#_Toc513768102)

[1.5 BỐ CỤC LUẬN VĂN 9](#_Toc513768103)

[Chương 2 CƠ SỞ LÝ THUYẾT 10](#_Toc513768104)

[2.1 NGÔN NGỮ LẬP TRÌNH JAVA 10](#_Toc513768105)

[2.1.1 Java và lịch sử phát triển 10](#_Toc513768106)

[2.1.2 Java Reflection 11](#_Toc513768107)

[2.2 SPRING FRAMEWORK 16](#_Toc513768108)

[2.2.1 Tổng quan về Spring 16](#_Toc513768109)

[2.2.2 Lịch sử phát triển 17](#_Toc513768110)

[2.2.3 Một số khái niệm chính 17](#_Toc513768111)

[2.2.3.1 Inversion of Control (IoC) 17](#_Toc513768112)

[2.2.3.2 Dependency Injection (DI) 21](#_Toc513768113)

[2.2.4 Các module 21](#_Toc513768114)

[2.2.5 Spring Core 23](#_Toc513768115)

[2.2.5.1 Bean 23](#_Toc513768116)

[2.2.5.2 Spring IoC Container 27](#_Toc513768117)

[2.2.5.3 Dependency Injection 31](#_Toc513768118)

[2.2.6 Spring MVC 36](#_Toc513768119)

[2.2.6.1 Tổng quan về Spring MVC 36](#_Toc513768120)

[2.2.6.2 DispathcherServlet 38](#_Toc513768121)

[2.2.6.3 Controllers 39](#_Toc513768122)

[2.2.6.4 Handler Mapping 41](#_Toc513768123)

[2.2.6.5 ViewResolver 43](#_Toc513768124)

[2.2.7 Spring Security 44](#_Toc513768125)

[2.2.7.1 Tổng quan về Spring Security 44](#_Toc513768126)

[2.2.7.2 Các module của Spring Security 46](#_Toc513768127)

[2.6 BOOTSTRAP VÀ RESPONSIVE 47](#_Toc513768128)

[Chương 3 THIẾT KẾ VÀ CÀI ĐẶT CHƯƠNG TRÌNH 48](#_Toc513768129)

[3.1 MÔ TẢ BÀI TOÁN 48](#_Toc513768130)

[3.1.1 MÔ TẢ TỔNG QUAN 48](#_Toc513768131)

[3.1.1.1 Bối cảnh hệ thống 48](#_Toc513768132)

[3.1.1.2 Các chức năng chính 48](#_Toc513768133)

[3.1.1.3 Đặc điểm người dùng 49](#_Toc513768134)

[3.1.2 MỘT SỐ QUY TRÌNH CƠ BẢN 50](#_Toc513768135)

[3.1.2.1 Quy trình đăng câu hỏi: 50](#_Toc513768136)

[3.1.2.2 Quy trình thảo luận câu hỏi 51](#_Toc513768137)

[3.2 SƠ ĐỒ USECASE 51](#_Toc513768138)

[3.2.1 Use case tổng quát cho từng tác nhân 52](#_Toc513768139)

[3.2.1.1 Người dùng khách: 52](#_Toc513768140)

[3.2.1.2 Người dùng là nhân viên 54](#_Toc513768141)

[3.2.2 Phân rã Use case 55](#_Toc513768142)

[3.2.2.1 Người dùng đăng nhập 55](#_Toc513768143)

[3.2.2.2 Người dùng đăng xuất 56](#_Toc513768144)

[3.2.2.3 Người dùng đăng ký 57](#_Toc513768145)

[3.2.2.4 Người dùng tìm kiếm câu hỏi 58](#_Toc513768146)

[3.2.2.5 Người dùng tạo câu hỏi 59](#_Toc513768147)

[3.2.2.6 Người dùng trả lời các câu hỏi 60](#_Toc513768148)

[3.2.2.7 Người dùng bình chọn một số câu trả lời 61](#_Toc513768149)

[3.2.2.8 Quản lý thông tin cá nhân 62](#_Toc513768150)

[3.2.2.9 Nhân viên quản lý trang web 63](#_Toc513768151)

[3.3 THIẾT KẾ CƠ SỞ DỮ LIỆU 64](#_Toc513768152)

[3.7.1 Đặc tả các tập thực thể 64](#_Toc513768153)

[3.7.2 Các mối quan hệ: 67](#_Toc513768154)

[3.7.1 Mô hình ER 72](#_Toc513768155)

[3.4 MÔ HÌNH TUẦN TỰ 73](#_Toc513768156)

[3.4.1 Đăng kí: 73](#_Toc513768157)

[3.4.2 Đăng nhập: 73](#_Toc513768158)

[3.4.3 Tạo câu hỏi 74](#_Toc513768159)

[3.4.4 Trả lời câu hỏi 75](#_Toc513768160)

[3.4.5 Tìm kiếm 75](#_Toc513768161)

[3.4.6 Quản lý thành viên 76](#_Toc513768162)

[3.5 SƠ ĐỒ PHÂN RÃ CHỨC NĂNG 76](#_Toc513768163)

[3.6 CƠ CHẾ BẢO MẬT ỨNG DỤNG 77](#_Toc513768164)

[3.7 GIAO DIỆN VÀ CHỨC NĂNG 77](#_Toc513768165)

[3.7.1 Một số giao diện chính 77](#_Toc513768166)

[3.7.2 Một số chức năng nổi bật của ứng dụng 83](#_Toc513768167)

[Chương 4 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 83](#_Toc513768168)

[4.1 KẾT QUẢ ĐẠT ĐƯỢC 83](#_Toc513768169)

[4.2 HẠN CHẾ 83](#_Toc513768170)

[4.3 HƯỚNG PHÁT TRIỂN 83](#_Toc513768171)

[TÀI LIỆU THAM KHẢO 83](#_Toc513768172)

[PHỤ LỤC 1: HƯỚNG DẪN TRIỂN KHAI ỨNG DỤNG 84](#_Toc513768173)

**TÓM TẮT**

Ngày nay tỷ lệ các dự án phần mềm thất bại hoặc không đáp ứng được các yêu cầu ban đầu là rất cao với hơn 83,8%. Bên cạnh đó chi phí cho việc bảo trì và mở rộng hệ thống luôn lớn hơn nhiều lần so với các pha khác trong quy trình phát triển một hệ thống. Một trong những nguyên nhân cơ bản nhất khiến các dự án thất bại là do các thành phần quá phụ thuộc lẫn nhau, chồng chéo, không có tính sử dụng lại. Vì vậy trong quy trình phát triển phần mềm hiện đại, xây dựng và phát triển một hệ thống bao giờ cũng đặt việc tạo ra các thành phần riêng rẽ độc lập, có sự phân biệt rõ ràng về trách nhiệm, có tính sử dụng lại cao lên làm ưu tiên hàng đầu. Tuy nhiên, điều này là không dễ dàng bởi tính độc lập của các thành phần sẽ bị giảm đi do cách mà chúng liên kết với nhau. Niên luận này sẽ tập trung tìm hiểu và giới thiệu sơ lược về Spring, những đặc tính vượt trội, ưu việt của nó trong việc tạo ra các ứng dụng đòi hỏi sự module hóa và có khả năng sử dụng lại cao. Đồng thời trình bày thêm về một số công nghệ như: MySQL, Javascript, Bootstrap hiện đang được các công ty phần mềm sử dụng để cùng với Spring tạo nên một ứng dụng web enterprise. Sau khi tìm hiểu tôi sẽ vận dụng kết quả tìm hiểu được vào việc xây dựng một ứng dụng nhằm mục đích minh họa cho phần lý thuyết đã trình bày. Tôi sẽ xây dựng một ứng dụng quản lý sản phẩm của một công ty trên nền web. Ứng dụng sẽ được thiết kế thành hai module chính là module cửa hàng và module diễn đàn. Trong luận văn này em sẽ trình bày chi tiết phần thiết kế và xây dựng module diễn đàn. Module cửa hàng sẽ được đề cập trong luận văn của em.

# Chương 1: TỔNG QUAN

## 1.1 LÝ DO CHỌN ĐỀ TÀI

Trong một khảo sát với khoảng 8.000 dự án phần mềm, Standish Group cho biết có chỉ có 16.2% dự án là hoàn thành đúng hạn và nằm trong giới hạn ngân sách, đáp ứng đầy đủ tất cả các tính năng như cam kết ban đầu. Và hơn 83.8% dự án thất bại hoặc không đáp ứng được những yêu cầu ban đầu, trong đó có tới 52.7% dự án được hoàn thành và đi vào hoạt động nhưng không hoàn thành đúng hạn và bội chi, thêm nữa không đáp ứng đầy đủ tính năng và đặt tính như thiết kế ban đầu. Bên cạnh đó chi phí cho việc bảo trì và mở rộng hệ thống luôn lớn hơn nhiều lần so với các pha khác trong quy trình phát triển một hệ thống.

3

3

5

7

15

67

Xác định yêu cầu

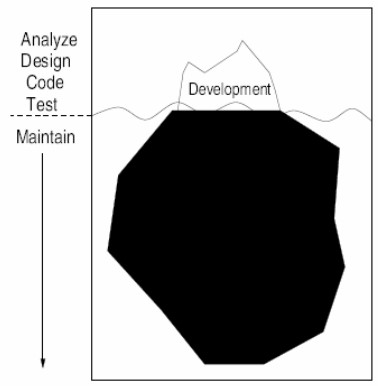
Đặt tả

Thiết kế

Lập trình

Kiểm thử

Bảo trì



###### Hình 1.1: Chi phí cho các pha phát triển một hệ thống

Những rủi ro dẫn đến hủy hoặc đình trệ của các dự án phần mềm tăng nhanh tỉ lệ thuận với việc gia tăng kích thước của dự án; 25% với các dự án > 100.000 LOC (line of code), 50% với các dự án > 500.000 LOC và 65% với các dự án > 1.000.000 LOC. Việc phát triển các hệ thống với hơn 5.000 function point (tương đương 500.000 LOC) được xem là một trong những nhiệm vụ rủi ro nhất.

Một trong những nguyên nhân cơ bản nhất để các dự án thất bại là do các thành phần quá phụ thuộc lẫn nhau, chồng chéo, không có tính sử dụng lại. Vì vậy trong quy trình phát triển phần mềm hiện đại, xây dựng và phát triển một hệ thống bao giờ cũng đặt việc tạo ra các thành phần riêng rẽ độc lập, có sự phân biệt rõ ràng về trách nhiệm, có tính sử dụng lại cao lên làm ưu tiên hàng đầu. Bởi vì một khi đã tạo ra các thành phần như vậy, chúng ta sẽ tiết kiệm được rất nhiều thời gian và công sức trong quá trình bảo trì và mở rộng hệ thống sau này.

Tuy nhiên, điều này quả là không dễ dàng bởi vì tính độc lập của các thành phần sẽ bị giảm đi do cách các thành phần liên kết với nhau.

Mặt khác, ngày nay hầu hết các hệ thống đều được phát triển trên nền web bởi vì các ứng dụng web đem lại khả năng phát triển và triển khai sản phẩm nhanh chóng, giảm thời gian đưa sản phẩm ra thị trường, có khả năng cập nhật tính năng liên tục hàng ngày.

Ứng dụng web có thời gian phát triển nhanh, với chi phí phát triển và triển khai thấp, tin cậy, có thể truy cập từ bất cứ đâu, mọi người có thể sử dụng mà không mất thời gian tìm hiểu nhờ giao diện trực quan.

Hơn nữa công nghệ di động đang ngày càng trở nên phổ biến, trong những năm gần đây số lượng người sử dụng và truy cập qua các thiết bị di động tăng một cách nhanh chóng, xu hướng thiết kế và viết các ứng dụng thích hợp với các thiết bị di động đang trở thành một xu hướng tất yếu. Các hệ thống đã được phát triển để có thể sẵn sàng cho các trang web phù hợp với thiết bị di động và các ứng dụng trên thiết bị di động.

Chính vì những lý do trên nên tôi chọn đề tài “**Tìm hiểu Framework Spring và xây dựng ứng dụng diễn đàn cho công ty kinh doanh nhà**”. Nội dung niên luận này sẽ tập trung tìm hiểu và giới thiệu về Spring, một framework với những đặc tính vượt trội và ưu việt trong việc tạo ra các hệ thống đòi hỏi sự module hóa và có khả năng sử dụng lại cao. Đồng thời tìm hiểu thêm một số framework và kỹ thuật khác hiện đang được các công ty phần mềm sử dụng để tích hợp với Spring tạo nên một ứng dụng web enterprise như: Javascript, Bootstrap, JSP, MySQL.

## 1.2 ĐẶT VẤN ĐỀ VÀ HƯỚNG GIẢI QUYẾT

EJB (Enterprise JavaBeans) là một kiến trúc phát triển dành cho các ứng dụng doanh nghiệp đòi hỏi tính mạnh mẽ và khả năng mở rộng cao. EJB là một thành phần quan trọng của nền tảng Java EE. EJB được tạo ra bởi hai gã khổng lồ công nghệ là IBM và Sun vào những năm cuối thập niên 90. Dưới sự phát triển của hai tập đoàn này, EJB đã trở nên nặng nề hơn. Tiêu chuẩn EJB cũng nhanh chóng được thông qua. Sau tất cả, EJB được ca ngợi là giải pháp tốt nhất để xây dựng các ứng dụng phân tán dành cho doanh nghiệp. Nhưng rồi những chỉ trích về chi tiết kỹ thuật của EJB đã tăng nhanh còn hơn sự nổi tiếng của nó. EJB là một framework lớn với độ phức tạp cao. Các nhà phát triền khi làm việc với EJB đã nhận ra rằng nó không hoạt động như những gì nó đã đề ra. EJB chỉ tuyệt vời trên bàn giấy, nhưng ra thực tế đó lại là một chuyện khác. EJB vẫn được tiếp tục phát triển, dễ sử dụng hơn nhiều so với buổi ban đầu. Nhưng những chỉ trích vẫn còn đó, nỗi thất vọng trong cộng đồng Java ngày càng tăng lên, đòi hỏi cần có một giải pháp thay thế.

Do đó Spring framework ra đời như là một lựa chọn thay thế cho mô hình chuẩn EJB đã đánh dấu một bước ngoặt trong lịch sử phát triển của Enterprise Java. Spring là một trong những framework của Java được sử dụng phổ biến nhất tính đến thời điểm này. Và nó được giới thương mại công nhận như một nền tảng kiến trúc có tầm quan trọng trong chiến lược kiến tạo phần mềm chính là do framework này có cách tạo và liên kết các thành phần rất riêng và hữu ích.

Trong các hệ thống lớn việc xây dựng các thành phần một cách độc lập, riêng rẽ là một yêu cầu quan trọng và nó phụ thuộc rất lớn vào việc lựa chọn công nghệ và kỹ thuật để phát triển hệ thống này. Tính độc lập, riêng rẽ giữa các thành phần được xem xét trên 2 khía cạnh khác nhau.

Một là xét về khía cạnh mã nguồn (code). Giảm sự kết dính của code (writing loosely coupled code) có nghĩa là sự phụ thuộc, liên kết giữa các class, đối tượng trong hệ thống càng “lỏng lẻo” (loosely coupled) càng tốt.

Để giải quyết cho việc “writing loosely coupled code” thì có khá nhiều kĩ thuật nổi tiếng như là: Dependencies Injection, Isolate Dependencies, Reversing Dependencies. Và Spring là một trong những framework mạnh về Dependencies Injection. Ngoài ra Spring là một framework mạnh trong việc xây dựng các ứng dụng enterprise. Nó cũng có thể dễ dàng tích hợp với các framework khác Struts, Hibernate làm cho việc phát triển các ứng dụng enterprise hiệu quả hơn do đó làm giảm sự phụ thuộc và có sự tách biệt rõ ràng giữa các thành phần.

Hai là xét về khía cạnh các thành phần, module trong hệ thống. Nền tảng cơ bản của việc thiết kế hướng đối tượng là việc các đối tượng thao tác với nhau qua việc gửi thông điệp (sending message). Do đó việc thiết kế một phần mềm sẽ xoay quanh việc chúng ta thiết kế sao cho các thành phần, module giao tiếp thông qua một giao diện mà không cần phải biết quá nhiều về nhau và phụ thuộc lẫn nhau.

Để giải quyết vấn đề này tôi áp dụng mô hình hệ thống gửi nhận thông điệp để làm thành phần giao tiếp giữa các module trong các hệ thống. Áp dụng mô hình gửi nhận thông điệp sẽ giúp cho nhiều thành phần dễ dàng giao tiếp với nhau và giảm sự phụ thuộc vào nhau rất nhiều. Trong niên luận này tôi sẽ trình bày về Java Message System (JMS) như là một công nghệ áp dụng cho việc xây dựng hệ thống gửi nhận thông điệp.

Ngày nay khi dịch vụ web bùng nổ phần lớn các hệ thống đều được xây dựng trên nền tảng như một ứng dụng web hơn là các ứng dụng chạy độc lập trên các máy tính cá nhân.

Tuy nhiên một trong những giới hạn của các ứng dụng web là cách thức nó tương tác với người dùng. Khác với các ứng dụng desktop có những khả năng dường như vô tận trong cách thức tương tác với người dùng. Các ứng dụng web tương tác với người dùng chậm hơn so với các ứng dụng desktop bởi chính nguyên lý hoạt động của nó: tất cả các giao dịch phải thực hiện thông qua giao thức HTTP.

Việc áp dụng công nghệ Ajax đóng vai trò cốt lõi thực hiện các tương tác trung gian giữa người dùng và máy chủ sẽ giúp chúng ta khắc phục một phần hạn chế này. Và để mang lại cho người dùng trải nghiệm giống như trên ứng dụng desktop một số framework JavaScript sẽ giúp chúng ta phát triển ứng dụng web dạng SPA. Trong luận văn này tôi sẽ trình bày về Ajax và áp dụng để xây dựng một ứng dụng web minh họa.

Ngày nay với số lượng người dùng các thiết bị di động ngày càng tăng, những hệ thống được triển khai trên nền web luôn đảm bảo yêu cầu là phải thích nghi được trên từng loại thiết bị di động này nhằm tăng thêm thị phần, tạo sự thuận lợi cho người dùng. Viêc hiển thị bố cục phù hợp cho từng loại thiết bị sẽ do CSS hoặc Javascript đảm nhận việc nhận diện kích thước màn hình và hiển thị với thiết lập CSS của giao diện dành cho thiết bị đó. Trong luận văn này tôi sẽ sử dụng Bootstrap để thiết kế giao diện responsive.

## 1.3 PHẠM VI ĐỀ TÀI

Niên luận sẽ tập trung trình bày kết quả nghiên cứu của tôi về các nội dung sau: Spring framework, MySQL, Hibernate, HTML, CSS, JavaScript, jQuery, AJAX, Boostrap và Maven. Mỗi phần tôi sẽ giới thiệu sơ lược và trình bày những nội dung cơ bản nhất, những điểm mạnh hay lợi ích mà nó mang lại cho các nhà phát triển phần mềm.

Cụ thể về Spring framework tôi sẽ tập trung tìm hiểu và trình bày 3 module: Spring core, Spring MVC và Spring Security. Một số module khác được trình bày trong luận văn của tôi.

Sau khi tìm hiểu tôi sẽ vận dụng kết quả tìm hiểu được vào việc xây dựng một ứng dụng nhằm mục đích minh họa cho phần lý thuyết đã trình bày. Tôi sẽ xây dựng một website cho công ty kinh doanh nhà thông minh. Ứng dụng sẽ được thiết kế thành 2 webtise chính là website giới thiệu sản phẩm và website diễn đàn. Trong niên luận này tôi sẽ trình bày phần thiết kế và xây dựng website diễn đàn. Website giới thiệu sản phẩm sẽ được trình bày trong luận văn của tôi.

## 1.4 PHƯƠNG PHÁP NGHIÊN CỨU

Tìm kiếm và nghiên cứu các tài liệu về Spring Framework và các công nghệ có liên quan đến việc phát triển một ứng dụng web enterprise như JMS, jQuery, MySQL, Maven của các tác giả trong và ngoài nước, các bài báo, thông tin trên mạng,… sau đó chọn lọc và sắp xếp lại theo ý tưởng của mình.

Dựa trên kết quả tìm hiểu được để xây dựng một ứng dụng web quản lý diễn đàn có áp dụng tất cả những nội dung đã nghiên cứu nhằm mục đích minh họa cho phần cơ sở lý thuyết sẽ trình bày trong nội dung luận văn này.

## 1.5 BỐ CỤC LUẬN VĂN

Trong phần này tôi sẽ trình bày ngắn gọn về nội dung chính của từng phần trong luận văn nhằm giúp cho người đọc dễ dàng theo dõi.

Chương 1: Giới thiệu tổng quan về cơ quan thực tập; lý do và tính cấp thiết để thực hiện đề tài này. Trong chương này còn trình bày về các vấn đề cụ thể cần phải giải quyết và hướng giải quyết cho từng vấn đề; phạm vi cũng như phương pháp nghiên cứu đề tài.

Chương 2: Trình bày về cơ sở lý thuyết được sử dụng trong đề tài bao gồm các nội dung về ngôn ngữ lập trình Java, Spring framework, Java Message System, AngularJS, MongoDB và Bootstrap:

* Trong phần Spring framework tôi sẽ giới thiệu sơ lược về framework này và một số khái niệm liên quan, tiếp theo sẽ trình bày về 3 module là Spring Core, Spring MVC và Spring Security.
* Phần Java Message System sẽ giới thiệu tổng quan về hệ thống gửi nhận thông điệp, về JMS và 2 mô hình gửi nhận thông điệp trong JMS.
* Phần AgularJS sẽ trình bày tổng quan về framework này và những đặc trưng nổi bật của nó.
* Phần tiếp theo trình bày về NoSQL, các dạng của NoSQL; giới thiệu về MongoDB và ưu điểm của hệ quản trị CSDL này.
* Phần cuối cùng sẽ giới thiệu về Bootstrap và tính năng responsive mà Bootstrap hỗ trợ trong việc xây dựng giao diện người dùng cho ứng dụng web.

Chương 3: Sau khi tìm hiểu về Spring, JMS, AngularJS, MongoDB và Bootstrap chương này trình bày phần phân tích, thiết kế và cài đặt ứng dụng quản lý nhạc phía client sử dụng các công nghệ vừa nêu.

Chương 4: Những kết quả đạt được, thảo luận những vấn đề khó khăn và đưa ra hướng phát triển trong tương lai.

Phần cuối cùng là tài liệu tham khảo và một số phụ lục về hướng dẫn triển khai ứng dụng; phụ lục giới thiệu về Maven.

# Chương 2 CƠ SỞ LÝ THUYẾT

## 2.2 SPRING FRAMEWORK

### 2.2.1 Tổng quan về Spring

Spring framework, hay ngắn hơn là Spring, là một cấu trúc dùng để xây dựng chương trình ứng dụng mã nguồn mở dành cho ngôn ngữ lập trình Java. Phiên bản đầu tiên của nó do Rod Johnson viết và đã được ông xuất bản cùng với quyển sách đầu tay Expert One-on-One JavaEE Design and Development (Nhà xuất bản Wrox Press, Tháng 10 năm 2002).

Kiến trúc của Spring framework được ra mắt công chúng lần đầu tiên hồi tháng 6 năm 2003 dưới Giấy phép Apache - phiên bản 2.0. Phiên bản 1.0 đánh dấu mốc thành đạt đầu tiên được xuất bản vào tháng 3 năm 2004 và tiếp đó vào tháng 9 năm 2004, tháng 3 năm 2005.

Tuy Spring framework không bắt buộc người ta phải tuân theo một mô hình lập trình cụ thể nào, song nó lan truyền rộng rải trong cộng đồng những người viết chương trình dùng Java, như một hình thức chủ yếu thay thế cho mô hình Enterprise Java Bean. Theo thiết kế, bộ framework này giải phóng lập trình viên dùng Java, cho phép họ nhiều quyền tự do hơn và đồng thời cung cấp một giải pháp tiện lợi, đầy đủ dẫn chứng bằng tài liệu, dễ dàng sử dụng, phù hợp với những thực hành thông dụng trong công nghệ phần mềm.

Bên cạnh những đặc trưng nền tảng của Spring framework là những cái có thể dùng được trong bất cứ một chương trình ứng dụng Java nào, rất nhiều các mở rộng và tiến bộ trong việc kiến tạo các trình ứng dụng dành cho nền tảng mạng web (web-based application) dựa trên nền Java Enterprise cũng tồn tại nữa. Spring framework nổi tiếng cũng một phần do chính đặc thù kể trên và được giới thương mại công nhận như một nền tảng kiến trúc có tầm quan trọng trong chiến lược kiến tạo phần mềm.

### 2.2.2 Lịch sử phát triển

Phần đầu tiên của Spring framework ban đầu nguyên được Rod Johnson viết vào năm 2000. Vào năm 2001, những mô hình lập trình cho các trình ứng dụng Web được đại đa số sử dụng đều do Java Servlet API và Enterprise Java Bean cung cấp.

Năm 2003 thành lập dự án phát tại Sourceforge để phát triển Spring. Sau khi phát triển trên nền tảng ban đầu hơn một năm họ đã phát hành phiên bản đầu tiên (1.0) vào tháng 3 năm 2004.

Spring framework đã làm cho những kỹ thuật vốn không được biết đến mấy trở nên những kỹ thuật được mọi người ưa chuộng trong một thời gian ngắn ngủi. Một trong những kỹ thuật nổi tiếng hơn cả là kỹ thuật “đảo ngược quyền điều khiển” (Inversion of Control, IoC).

Năm 2005 cho thấy mức độ hưởng ứng nâng cao hơn những năm trước, nguyên do cũng vì những phiên bản mới được giới thiệu với những cột mốc đáng kể và những tính năng mới được thêm vào. Diễn đàn Spring (Spring Forum) ra đời cuối năm 2004 cũng góp phần không nhỏ trong việc nâng cao tính phổ biến của bộ framework và từ đó đến nay đã sinh trưởng trở thành một nguồn thông tin quan trọng, giúp đỡ cho người sử dụng.

Vào tháng 12 năm 2005, hội thảo đầu tiên về Spring Framework đã được tổ chức tại Miami, Florida thu hút 300 nhà phát triển trong 3 ngày và tiếp theo đó cuộc hội thảo ở Antwerp vào tháng 6 năm 2006, thu hút hơn 400 người.

### 2.2.3 Một số khái niệm chính

Hai trong những thành phần chủ chốt và là nền tảng tạo nên sức mạnh của Spring chính là IoC và DI.

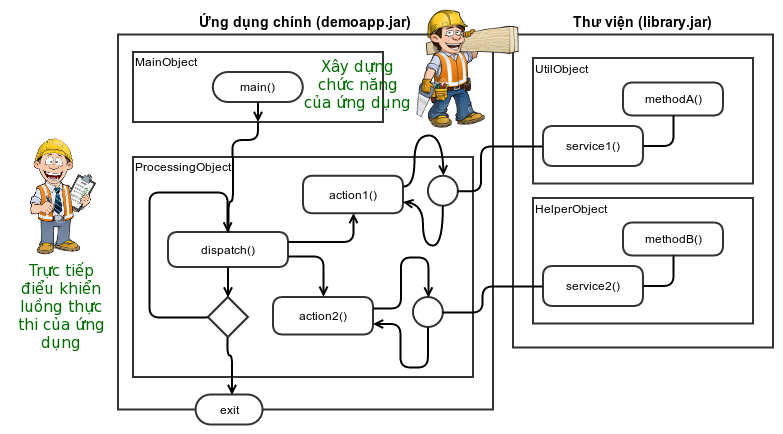
#### 2.2.3.1 Inversion of Control (IoC)

IoC Container trong Spring được xây dựng dựa trên nguyên lý Inversion of Control (đảo ngược điều khiển) đã xuất hiện khá lâu trong các mẫu hình thiết kế (design pattern), và được phổ biến rộng rãi nhờ Robert C. Martin và Martin Fowler. Để hiểu về Spring, trước tiên chúng ta cần hiểu khái niệm IoC là gì? Muốn vậy, chúng ta cần trả lời câu hỏi: Control (điều khiển) trong chương trình phần mềm là gì, và Inversion (sự đảo ngược) đối với điều khiển trong ngữ cảnh này được hiểu như thế nào?

Khái niệm Control Flow (tạm dịch là luồng thực thi) được sử dụng cho trình tự thực hiện các câu lệnh, chỉ thị hoặc lời gọi hàm trong một chương trình, khi chương trình này thực thi.

Do chương trình ngày càng phức tạp, nên các lập trình viên áp dụng phương pháp lập trình hướng đối tượng nhằm phân loại, chia tách các chức năng và gom thành các đối tượng. Các lập trình viên còn tạo dựng các thư viện tạo sẵn để có thể sử dụng lại. Luồng thực thi của chương trình, trong những tình huống cần xem xét ở mức tổng thể, không còn quan tâm đến các bước thực thi câu lệnh cụ thể nữa, mà chỉ xem xét đến quá trình gọi phương thức của các đối tượng trong ứng dụng cũng như các đối tượng của thư viện dựng sẵn.

Các lập trình viên, khi xây dựng ứng dụng từ đầu, đã thực hiện hai nhiệm vụ: trực tiếp điều khiển luồng thực thi của chương trình và xây dựng các chức năng để đáp ứng nghiệp vụ của ứng dụng. Thực tế, có nhiều chương trình hoặc bộ phận trong chương trình có luồng thực thi rất giống nhau, chẳng hạn phần tương tác với HTTP trong các ứng dụng web, phần unit testing trong các ứng dụng,… Việc trực tiếp tạo dựng và kiểm soát luồng thực thi của chương trình lặp đi lặp lại khi xây dựng nhiều ứng dụng sẽ làm mất nhiều công sức, chi phí, tạo ra sự nhàm chán và dễ phát sinh lỗi. Điều này tạo ra động lực cũng như môi trường để nguyên lý đảo ngược điều khiển nảy nở và phát triển.



###### Hình 2.2: Luồng điều khiển của chương trình bình thường

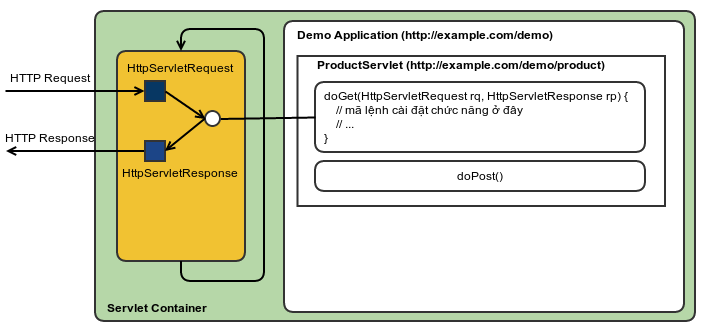
Vậy sự “đảo ngược” (inversion) luồng điều khiển chương trình trong ngữ cảnh này được hiểu như thế nào? Chúng ta hãy xem xét 2 ví dụ dưới đây.

Ví dụ 1: Ứng dụng web trong Java với JavaServlet

Khi lập trình ứng dụng web bằng JavaServlet cần thực hiện các bước lập trình như sau:

* Tạo lớp đối tượng kế thừa từ HttpServlet, override các phương thức doGet(), doPost(),…
* Đăng ký trong file cấu hình Deployment Descriptor tương ứng Servlet này với đường dẫn xác định.
* Lớp đối tượng Servlet chúng ta tạo ra sẽ được gọi đến khi có một truy vấn HTTP có đường dẫn “khớp” với đường dẫn khai báo trong Deployment Descriptor.

Vậy ai kích hoạt ứng dụng chúng ta viết để đáp ứng mỗi khi có HTTP Request gửi đến? Ai chịu trách nhiệm chuyển đổi các thông điệp HTTP (HTTP Request và HTTP Response) thành các đối tượng Java (HttpServletRequest và HttpServletResponse) để truyền cho các hàm doGet(), doPost()? Đó chính là Servlet Container.

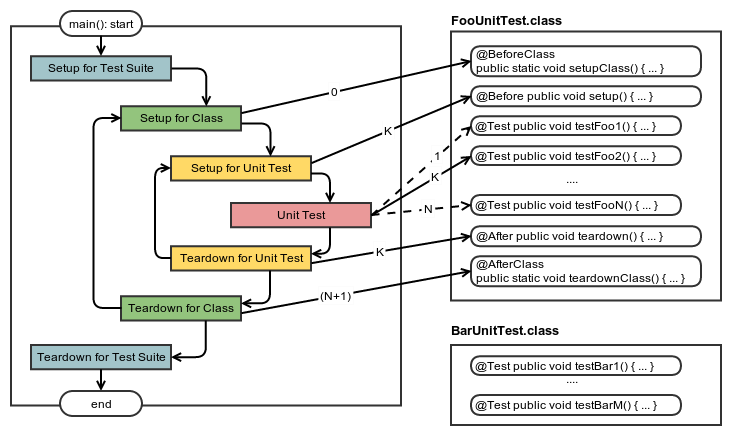


###### Hình 2.3: Servlet Container

Ví dụ 2: Lập trình kiểm thử đơn vị (Unit testing) với Junit

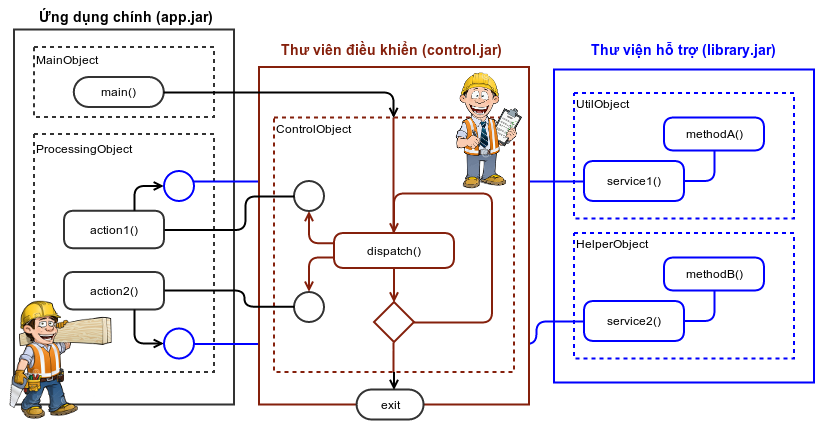
Trong quá trình phát triển các thành phần chức năng của ứng dụng, chúng ta thường áp dụng kiểm thử đơn vị để đảm bảo chức năng đó vẫn chạy đúng trong suốt quá trình ứng dụng được mở rộng và phát triển thêm. Để tạo bộ unit test, chúng ta chỉ cần tạo một lớp đối tượng, định nghĩa các phương thức khởi tạo, phương thức kết thúc và các phương thức test. Sau đó, chúng ta chỉ việc chạy bộ test để kiểm thử.

Việc điều khiển trình tự thực thi các phương thức được giao cho thư viện bên ngoài đảm nhiệm chẳng hạn như TestNG hoặc JUnit.



###### Hình 2.4: Kiểm thử đơn vị với JUnit

Với hai ví dụ trên, chúng ta nhận thấy trong các ứng dụng đã có sự thay đổi vai trò. Ứng dụng không còn ôm đồm vừa trực tiếp tạo dựng và kiểm soát luồng thực thi, vừa xây dựng chức năng nghiệp vụ. Việc kiểm soát luồng thực thi được tách khỏi chức năng nghiệp vụ và bị đẩy ra bên ngoài. Người lập trình đã ủy thác việc kiểm soát luồng thực thi ứng dụng cho một thành phần (thường là thư viện dựng sẵn) bên ngoài đảm nhiệm, chỉ còn tập trung vào chức năng chính của ứng dụng.



###### Hình 2.5: Mô hình ứng dụng áp dụng IoC

Như vậy, khái niệm “đảo ngược” ở đây chính là chuyển nhiệm vụ kiểm soát lưu trình thực thi từ ứng dụng cho một thành phần chuyên trách (thường là một thư viện phần mềm khung – framework – dựng sẵn ở bên ngoài). Ứng dụng chính chúng ta quan tâm phát triển không kiểm soát việc điều khiển luồng thực thi nữa, mà chỉ tập trung vào việc định nghĩa chức năng. Thư viện phần mềm khung chuyên trách kiểm soát điều khiển sẽ dựa trên mô tả trong cấu hình của ứng dụng để thay mặt ứng dụng điều phối luồng thực thi trong chương trình.

Dễ dàng nhận thấy thư viện phần mềm khung này khác với các thư viện thông thường ở chỗ: thư viện thông thường cung cấp các chức năng và chờ được ứng dụng gọi đến, còn thư viện phần mềm khung tạo dựng luồng thực thi và gọi đến các chức năng của ứng dụng.

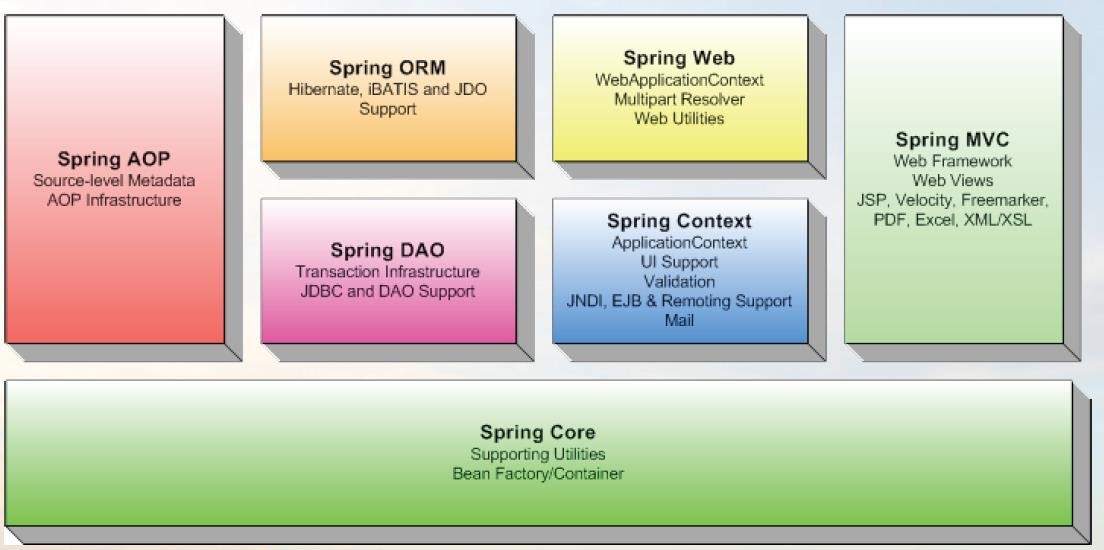
Nói một cách ngắn gọn IoC là một design pattern và tập hợp các kỹ thuật lập trình liên quan, trong đó luồng thực thi của một hệ thống bị đảo ngược so với cách tương tác truyền thống. IoC trong Spring cũng hoàn toàn mang ý nghĩa như trên. Trong Spring các đối tượng chính để xây dựng ứng dụng thì được quản lý bởi Spring IoC container. Và IoC container gọi các đối tượng đó là các bean. Một bean chỉ đơn giản là một đối tượng được khởi tạo và quản lý bởi Spring IoC container.

#### 2.2.3.2 Dependency Injection (DI)

Dependency Injection là khả năng liên kết giữa các thành phần lại với nhau, đó chính là việc các thuộc tính trong một đối tượng được “tiêm chích” (injection) để tham chiếu lần lượt đến các đối tượng khác. Dependency Injection trong Spring hoạt động dựa trên Java Reflection.

### 2.2.4 Các module

Spring được xây dựng với 7 module chính (Hình 2-6).



###### Hình 2.6: Các module của Spring framework

* **Spring Core**: Core package là phần cơ bản nhất của Spring, cung cấp những đặc tính như IoC (Inversion of Control) và DI (Dependency Injection). Khái niệm cơ bản là BeanFactory, một cài đặt của Factory pattern, cho phép “móc nối” sự phụ thuộc giữa các đối tượng trong file cấu hình.
* **Spring Context**: Spring context là một file cấu hình để cung cấp thông tin ngữ cảnh của Spring. Spring context cung cấp các service như JNDI access, EJB integration, e-mail, internalization, validation, và scheduling functionality.
* **Spring AOP (Aspect – Oriented Programming)**: Spring AOP module tích hợp chức năng lập trình hướng khía cạnh vào Spring framework thông qua cấu hình của nó. Spring AOP module cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong bất kỳ ứng dụng nào sử dụng Spring. Với Spring AOP chúng ta có thể tích hợp declarative transaction management vào trong ứng dụng mà không cần dựa vào EJB component. Spring AOP module cũng đưa lập trình metadata vào trong Spring. Sử dụng cái này chúng ta có thể thêm annotation vào source code để hướng dẫn Spring nơi và làm thế nào để liên hệ với aspect.
* **Spring DAO (Data Access Object)**: Tầng JDBC và DAO đưa ra một cây phân cấp exception để quản lý kết nối đến database, điều khiển exception và thông báo lỗi được ném bởi vendor của database. Tầng exception đơn giản điều khiển lỗi và giảm khối lượng code mà chúng ta cần viết như mở và đóng kết nối.

Module này cũng cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong ứng dụng Spring.

* **Spring ORM (Object Relational Mapping):** Spring có thể tích hợp với một vài ORM framework để cung cấp Object Relation tool bao gồm: JDO, Hibernate, OJB và iBatis SQL Maps.
* **Spring Web**: Nằm trên application context module, cung cấp context cho các ứng dụng web. Spring cũng hỗ trợ tích hợp với Struts, JSF và Webwork. Web module cũng làm giảm bớt các công việc điều khiển nhiều request và gắn các tham số của request vào các đối tượng domain.
* **Spring MVC**: MVC Framework thì cài đặt đầy đủ đặc tính của MVC pattern để xây dựng các ứng dụng Web. MVC framework thì cấu hình thông qua giao diện và chứa được một số kỹ thuật view bao gồm: JSP, Velocity, Tiles và generation of PDF và Excel file.

### 2.2.5 Spring Core

#### 2.2.5.1 Bean

Trong Spring các object được quản lý bởi IoC container và được gọi là bean. Một bean thì đơn giản là một đối tượng được khởi tạo, phân phát và được quản lý bởi IoC container. Sự phụ thuộc giữa chúng được phản ánh trong configuration metadata.

Một Spring container sẽ quản lý một hoặc nhiều bean. Các bean được tạo ra nhờ vào thông tin cấu hình được chỉ ra trong thẻ <bean /> nếu sử dụng XML config. Bên trong container các bean được biểu diễn như một đối tượng của lớp BeanDefinition, nó chứa các thông tin sau:

* Class của đối tượng mà bean được cấu hình (bao gồm cả phần package)
* Các thông số cấu hình hành vi của bean như: phạm vi (scope), vòng đời

(lifecycle),…

* Các tham chiếu đến các bean khác, các tham chiếu này được gọi là collaborators hoặc dependencies
* Các thiết lập khác như số connection kết nối đến bean,…

Các thuộc tính sau đây thường được sử dụng để định nghĩa một bean: class (instantiating beans), name (naming beans), scope, constructor arguments, properties, autowiring mode, lazy-initialization mode, initialization method, destruction method

##### 2.2.5.1.1 Tên bean (Naming beans)

Mỗi bean có một hoặc nhiều định danh (indentifiers), các định danh này phải là duy nhất trong một container. Một bean thường chỉ có duy nhất một định danh, tuy nhiên nếu cần nhiều hơn một định danh thì có thể sử dụng bí danh (alias).

Trong cấu hình dạng XML, có thể sử dụng thuộc tính *id* và/hoặc *name* để chỉ định định danh cho bean. Thuộc tính *id* cho phép chỉ định chính xác một định danh duy nhất cho bean.

Chúng ta có thể không cần chỉ định *id* hoặc *name* cho bean, khi đó container sẽ tự tạo ra một định danh duy nhất cho bean. Tuy nhiên nếu muốn sử dụng tham chiếu ở bean khác bằng cách sử dụng thuộc tính *ref* thì cần phải chỉ định một cách rõ ràng.

Thông thường các bean được đặt tên giống như khai báo biến trong Java, theo quy ước camel-cased. Ví dụ như: accountManager, userDao, loginController…Đặt tên cho bean sẽ giúp file cấu hình dễ đọc và hiểu hơn và nếu sử dụng Spring AOP nó sẽ giúp rất nhiều khi thêm các advice vào bean thông qua tên của bean.

##### 2.2.5.1.2 Khởi tạo bean

Một bean thực chất là một “công thức” cho việc tạo ra một hoặc nhiều đối tượng. Nếu sử dụng configuration metadata dạng XML (XML-based configuration metadata) cần phải chỉ định rõ class của đối tượng sẽ được bean quản lý và khởi tạo khi có yêu cầu. Có 3 cách để khởi tạo một bean:

**Khởi tạo bean với hàm dựng (constructor)**

Thông thường khi tạo một bean bằng phương pháp sử dụng constructor, tất cả các class đều có thể dùng được và phù hợp với Spring. Có nghĩa là các class không cần phải thực thi (implement) bất kỳ giao diện (interface) cụ thể nào. Tuy nhiên, tùy thuộc vào loại IoC có thể cần một constructor mặc định (không đối số).

Spring IoC container có thể quản lý hầu như bất kỳ class nào mà chúng ta muốn nó quản lý. Hầu hết người dùng Spring thích sử dụng cách tạo một bean bằng việc sử dụng một constructor mặc định (không có đối số) và các setter và getter.

**Khởi tạo bean với phương thức tĩnh (static factory method)**

Để tạo một bean sử dụng phương thức tĩnh, thuộc tính *class* chỉ ra lớp chứa phương thức tĩnh và thuộc tính *factory-name* chỉ ra tên của phương thức này được định nghĩa bên trong class vừa chỉ định. Ví dụ dưới đây sẽ cho thấy rõ việc tạo một bean bằng cách gọi phương thức tĩnh.

|  |
| --- |
| **public** **class** QuestionService {  **private** **static** QestionDAO questionDAO = **new** QuestionDAO();  **private** QuestionService(){ }  **public static** QestionDAO createInstance() { **return** questionDAO;  }  } |

<bean id=*"*questionDAO*"* class=*"com.ctu.tqsang.questionDAO"* factory-method=*"*createInstance*"* />

Lưu ý thuộc tính *class* trong tag bean không phải chỉ định kiểu (class) của đối tượng mà thật ra chỉ là chỉ định class chứa phương thức tĩnh được chỉ định ở thuộc tính *factoty-method* được gọi để tạo ra đối tượng mà thôi, trong ví dụ này chính là phương thức *createInstance()* trong lớp ***MusicService***.

**Khởi tạo bean sử dụng phương thức của đối tượng (instance factory method)**

Tương tự như việc sử dụng phương thức tĩnh, với phương pháp này sử dụng một phương thức non-static của một bean để tạo ra bean mới. Để sử dụng cách này thuộc tính *class* của tag bean sẽ không được chỉ định, thay vào đó thuộc tính *factory-bean* chỉ ra tên của bean mà có chứa phương thức dùng để tạo ra đối tượng. Tên của phương thức này được chỉ định trong thuộc tính *factory-method*. Ví dụ dưới đây sẽ chỉ ra cách sử dụng phương pháp này.

|  |
| --- |
| **public** **class** QuestionService {  **private** **static** AnswerService answerService = **new** AnswerService();  **private** QuestionService() { }  **public** AnswerService createAnswerServiceInstance() {  **return** answerService;  }  } |

*<!-- factory bean -->*

<bean id=*"*questionService*"* class=*"com.ctu.tqsang.QuestionService"*>

*<!-- các cấu hình khác cho bean này ở đây -->*

</bean>

*<!-- bean được tạo thông qua factory bean -->*

<bean id=*"*answerService*"* factory-bean=*"*questionService*"*

Factory-method=*"*createAnswerServiceInstance*"* />

##### 2.2.5.1.3 Phạm vi của bean (Bean scopes)

Khi định nghĩa một bean chúng ta đã đưa ra một “công thức” cho việc tạo một thể hiện thực sự của một class. Việc khởi tạo một hay nhiều thể hiện, khi nào thì một thể hiện được tạo ra sẽ phụ thuộc vào phạm vi của bean. Spring đưa ra năm phạm vi của một bean, trong đó ba phạm vi chỉ có khi xây dựng các ứng dụng web.

Năm phạm vi của một bean được trình bày tóm tắt trong bảng dưới đây:

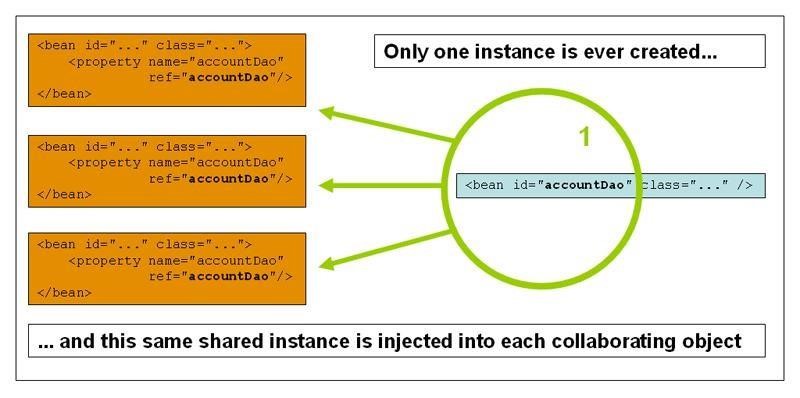
Bảng 2-1: Phạm vi của bean

|  |  |
| --- | --- |
| **Phạm vi** | **Giải thích** |
| Singleton | Chỉ có một thể hiện duy nhất được tạo ra cho bean trong mỗi Spring IoC container. Đây là phạm vi mặc định của mỗi bean |
| Prototype | Ngược lại với singleton, prototype cho phép nhiều hơn một đối tượng được tạo ra trên mỗi container. |
| Request | Một thể hiện cụ thể sẽ được tạo ra cho mỗi HTTP request. Phạm vi này chỉ có trong container của các ứng dụng web  (WebApplicationContext) |
| Session | Tương tự như request nhưng, một thể hiện cụ thể sẽ được tạo ra cho mỗi HTTP session. Phạm vi này chỉ có trong container của các ứng dụng web (WebApplicationContext) |
| Global session | Một thể hiện cụ thể sẽ được tạo ra cho một global HTTP session. Phạm vi này chỉ có trong container của các ứng dụng web. |
| Appplication | Một thể hiện cụ thể sẽ được tạo ra cho một Servlet context. Phạm vi này chỉ có trong container của các ứng dụng web. |

Thông thường thì singleton và prototype là hai phạm vi của bean được sử dụng nhiều nhất. Dưới đây xin trình bày chi tiết về 2 phạm vi này.

**Singleton scope.**

Khi khai báo một bean với phạm vi là singleton điều này có nghĩa là chỉ có duy nhất một thể hiện của class được chỉ ra trong bean trong một container. Thể hiện này sẽ được lưu trữ trong cache của Spring IoC container, tất cả các yêu cầu tiếp theo tham chiếu đến bean này thì thể hiện được lưu trữ trong cache sẽ được trả về.



###### Hình 2.7: Singleton scope

Ví dụ dưới đây cho thấy cách khai báo một bean với phạm vi singleton trong XML:

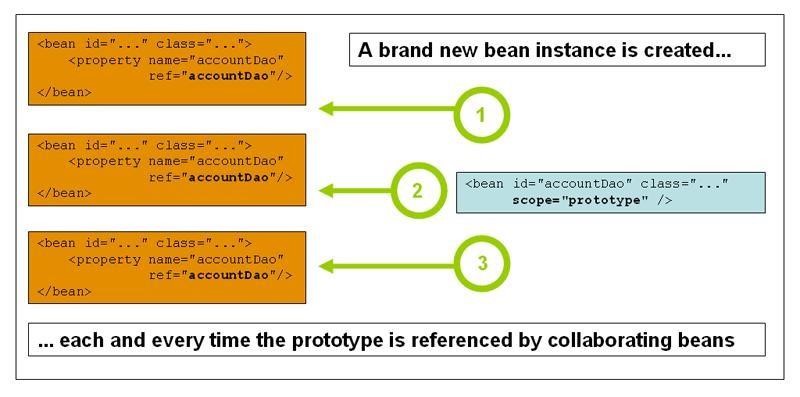
<bean id=*"*userDao*"* class=*"com.ctu.tqsang.UserDao"*></bean>

*<!-- hoặc -->*

<bean id=*"*userDao*"* class=*"com.ctu.tqsang.UserDao"* scope=*"*singleton*"*></bean>

**Prototype scope**

Ngược lại với singleton, một bean với phạm vi là prototype thì mỗi khi có một yêu cầu tham chiếu đến bean này thì một thể hiện cụ thể của bean sẽ được tạo ra.



*.*

*:*

###### Hình 2.8: Prototype scope

Khai báo một bean với phạm vi prototype trong XML như sau:

<bean id=*"*userDao*"* class=*"com.ctu.tqsang.UserDao"* scope=*"*prototype*"*></bean>

#### 2.2.5.2 Spring IoC Container

Spring là một framework thực thi theo nguyên tắc Inversion of Control, IoC cũng được biết đến như Dependency Injection. Nó là một quá trình xử lý ở nơi các object định nghĩa sự phụ thuộc (dependency). Khi các đối tượng hoạt động với nhau chỉ thông qua các tham số của constructor, tham số của các method hoặc các property để thiết lập thể hiện sau khi được khởi tạo. Container sau đó “tiêm” các đối tượng phụ thuộc khi nó được tạo ra từ các bean. Quá trình này về cơ bản là sự đảo ngược, các bean sẽ điều khiển các thể hiện hoặc vị trí phụ thuộc bằng cách khởi tạo trực tiếp từ class của chúng.

Package *org.springframework.beans* và *org.springframework.context* là 2 package cơ bản cho IoC container của Spring. Interface BeanFactory cung cấp kỹ thuật để cấu hình nâng cao và quản lý bất kỳ loại object nào. ApplicationContext kế thừa BeanFactory, ApplicationContext thêm vào một số tính năng như tích hợp để dễ dàng hoạt động với các tính năng của Spring AOP (Aspect Oriented Programming) như Message resource handling, event publication; và một số lớp context đặc biệt như WebApplicationContext được sử dụng trong các ứng dụng web. Nói một cách ngắn gọn, BeanFactory cung cấp cơ chế cấu hình cho framework và các chức năng cơ bản. ApplicationContext thêm vào một số chức năng nâng cao. Trong phần này sẽ sử dụng ApplicationContext để để mô tả cho IoC container của Spring.

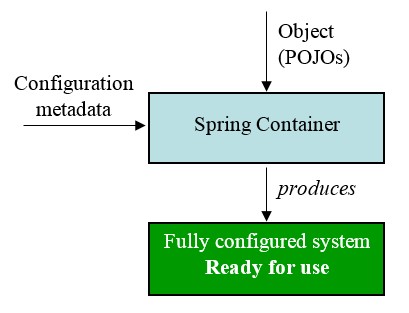
Interface *org.springframework.context.ApplicationContext* chịu trách nhiệm khởi tạo, cấu hình và phân phát các bean. Container lấy các chỉ dẫn cho việc khởi tạo, cấu hình và phân phát bằng cách đọc các configuration metadata. Các configuration metadata được trình bày bằng nhiều cách khác nhau như trong file XML, Java annotaion hoặc trong Java code. Điều này cho phép biểu diễn các đối tượng và sự phụ thuộc lẫn nhau giữa chúng trở nên phong phú hơn.

Một số class thực thi (implementation) interface ApplicationContext như Class

PathXmlApplicationContext, FileSystemXmlApplicationContext, AnnotationConfigApplicationContext,… XML là cách truyền thống và thường được sử dụng để định nghĩa các configuration metadata, chúng ta cũng có thể khai báo để container sử dụng các annotation như configuration metadata hoặc trực tiếp trong code Java bằng cách cấu hình thêm trong file XML cho phép sử dụng các annotation như sau: <context:annotation-config/>

Hình 2-9 mô tả cách thức làm việc của Spring. Các class trong ứng dụng sẽ kết hợp với các configuration metadata để sau khi ApplicationContext được tạo ra và khởi tạo chúng ta có được một cấu hình đầy đủ và có thể thực thi được.

Như hình chúng ta thấy các configuration metadata sẽ nói với Spring container khởi tạo, cấu hình và phân phối các object trong ứng dụng của chúng ta như thế nào.



###### Hình 2.9: Spring IoC Container

Thông thường các configuration metadata được biểu diễn dưới dạng XML, vì vậy trong phần này sẽ sử dụng configuration metadata dưới dạng này để nói về các khái niệm và tính năng của IoC container.

Lưu ý rằng không phải chỉ có các configuration metadata dưới dạng XML mới được Spring container chấp nhận, như đã nói ở phần trên ngoài các metadata dạng XML chúng ta còn có thể sử dụng Java annotaion hoặc trực tiếp trong code Java (Java config). Kể từ phiên bản 2.5 Spring đã hổ trợ kiểu cấu hình dựa trên các Java annotation. Và kể từ Spring 3.0 nhiều tính năng đã được cung cấp bằng cách sử dụng Java config và nó trở thành một phần của Spring core. Vì vậy chúng ta có thể định nghĩa bean bên ngoài các class của ứng dụng bằng cách sử dụng Java code hơn là file XML. Để sử dụng tính năng này xem thêm các annotation như @Configuration, @Bean, @Import và @DependOn. Cấu hình trong Spring bao gồm ít nhất là một và thường là nhiều hơn một định nghĩa bean mà container cần phải quản lý [6, tr28]. Trong metadata dạng XML để khai báo và cấu hình một bean ta sử dụng thẻ <bean /> bên trong thẻ <beans />. Nếu sử dụng dạng Java config thông thường sẽ sử dụng annotaion @Bean bên trong một class với annotaion @Configuration.

Ví dụ dưới đây là sẽ cho thấy cấu trúc cơ bản của một file XML trình bày các configuration metadata.

<?xml version=”1.0” encoding=”UTF-8”?>

<beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi[=*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)  xsi:schemaLocation=*"http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>

<bean id=*"bean1"* class=*"com.ctu.tqsang.Bean1"*>

*<!-- cấu hình cho bean ở đây -->*

</bean>

<bean id=*"bean2"* class=*"com.ctu.tqsang.Bean2"*>

*<!-- cấu hình cho bean ở đây -->*

</bean>

</beans>

Thuộc tính id chỉ ra định danh của bean đó, thuộc tính class chỉ ra class của bean này (lưu ý sử dụng tên đầy đủ của class bao gồm cả phần package). Giá trị của thuộc tính id sẽ được các bean khác sử dụng nếu cần thiết.

##### 2.2.5.2.1 Khởi tạo container

Việc khởi tạo một Spring container đơn giản như việc chúng ta khởi tạo một đối tượng, cần chỉ rõ đường dẫn của file XML cấu hình trong khi khởi tạo một container. Ví dụ dưới đây sẽ cho chúng ta thấy rõ hơn:

ApplicationContext context = **new** ClassPathXmlApplicationContext(**new** String[] {*"*services.xml*"*,

*"*daos.xml*"*} );

Trong đó file config “services.xml” như sau:

|  |
| --- |
| <?xml version=”1.0” encoding=”UTF-8”?>  <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi[=*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)  xsi:schemaLocation=*"http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>  <bean id=*"questionService"* class=*"com.ctu.tqsang.QuestionServicesImpl"*>  <property name=*"user*Dao” ref=”userDao*"* />  <property name=*"answer*Dao*"* ref=*"answer*Dao*"* />  *<!-- các cấu hình khác ở đây -->*  </bean>  </beans> |

Và file config “daos.xml” như sau:

|  |
| --- |
| <?xml version=”1.0” encoding=”UTF-8”?>  <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi[=*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)  xsi:schemaLocation=*"http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>  <bean id=*"user*Dao*"* class=*"com.ctu.tqsang.UserDao"*>  </bean>    <bean id=*"answer*Dao*"* class=*"com.ctu.tqsang.AnswerDao"*>  </bean>  </beans> |

Ở ví dụ trên class ***QuestionServicesIpml*** được cấu hình trong file *“services.xml”* và có hai thuộc tính truy cập dữ liệu thuộc hai lớp là ***UserDao*** và ***AnswerDao***. Thẻ

*<property />* dùng để tham chiếu đến một bean khác, trong đó thuộc tính *name* là tên của thuộc tính trong class ***QuestionServicesIpml*** và giá trị của *ref* là *id* của bean cần tham chiếu đến. Thường thì mỗi file cấu hình XML sẽ cấu hình cho một tầng hoặc một module trong ứng dụng. Việc chia nhỏ như thế giúp dễ dàng kiểm soát hơn.

##### 2.2.5.2.2 Sử dụng container

Ví dụ dưới đây sẽ chỉ ra cách đọc các định nghĩa của bean và truy cập đến chúng.

|  |
| --- |
| ApplicationContext context =  **new** ClassPathApplicationContext(**new** String[] {*"*services.xml*"*,  *"*daos.xml*"*} );  *//nhận về đối tượng đã được config*  QuestionServicesImpl services = (QuestionServicesImpl)context.getBean(*"questionService"*);    //sử dụng đối tượng  List<Answer> answers = services.getAnswerList(); |

#### 2.2.5.3 Dependency Injection

##### 2.2.5.3.1 Constructor – based dependency injection

Constructor – based DI là phương pháp “tiêm sự phuộc” cho các đối tượng bằng cách gọi hàm tạo (constructor method) với các tham số chính là các “sự phụ thuộc” (dependentcies) của đối tượng. Ví dụ sau đây sẽ đưa ra một class chỉ có thể tiêm sự phụ thuộc bằng phương pháp này.

|  |
| --- |
| **public** **class** MessageSender {    **private** Message message;    **public** MessageSender(Message message) { **this**.message = message;  }  } **public** **class** Message {    **public** Message() {    }  } |

Khi đó để tiêm sự phụ thuộc cho lớp MessageSender cần khai báo các bean như sau:

|  |
| --- |
| <bean id=*"message"* class=*"com.ctu.tqsang.Message"* />    <bean id=*"sender"* class=*"com.ctu.tqsang.MessageSender"*>  <constructor-arg ref=*"message"* /> </bean> |

Thẻ <constructor-arg /> trong ví dụ này sẽ cho Spring container biết đây là một bean mà các dependentcy sẽ được tiêm vào bằng cách gọi hàm tạo của lớp này. Đối với các hàm tạo với nhiều đối số thì thứ tự của các thẻ <constructor-arg /> sẽ giống với thứ tự các đối số của hàm, giống như ví dụ dưới đây

**public** **class** A {

**private** B b;  **private** C c;

**public** A(B b, C c) { **this**.b = b;

**this**.c = c;

}

}

<bean id=*"a"* class=*"com.ctu.tqsang.A"*>

<constructor-arg ref=*"b"* />

<constructor-arg ref=*"c"* />

</bean>

<bean id=*"b"* class=*"com.ctu.tqsang.B"* />

<bean id=*"c"* class=*"com.ctu.tqsang.C"* />

Ở các ví dụ trên các dependentcy là những kiểu dữ liệu do người dùng định nghĩa nên thẻ <constructor-arg /> sử dụng thuộc tính *ref* để tham chiếu đến bean khác. Riêng đối với các kiểu dữ liệu nguyên thủy (int, boolean, string,…) thì khai báo bean đơn giản hơn.

|  |
| --- |
| **public** **class** Z {    **private** **int** b;  **private** String c;  **private** **boolean** d;    **public** A(B b, C c, D d) {  **this**.b = b;  **this**.c = c;  **this**.d = d;  }  } |

Các khai báo bean dưới đây là tương tự nhau.

|  |
| --- |
| <bean id=*"z"* class=*"com.ctu.tqsang.Z"*>  <constructor-arg value=*"123"* />  <constructor-arg value=*"abc"* />  <constructor-arg value=*"true"* />  </bean>    <!-- hoặc -->  <bean id=*"z"* class=*"com.ctu.tqsang.Z"*>  <constructor-arg value=*"123"* type=*"int"*/>  <constructor-arg value=*"abc"* type=*"java.lang.String"*/>  <constructor-arg value=*"true"* type=*"boolean"*/>  </bean>    <!-- hoặc -->  <bean id=*"z"* class=*"com.ctu.tqsang.Z"*>  <constructor-arg index=*"0"* value=*"123"* />  <constructor-arg index=*"1"* value=*"abc"* />  <constructor-arg index=*"2"* value=*"true"* />  </bean>    <!-- hoặc -->  <bean id=*"z"* class=*"com.ctu.tqsang.Z"*>  <constructor-arg name=*"b"* value=*"123"* />  <constructor-arg name=*"c"* value=*"abc"* />  <constructor-arg name=*"d"* value=*"true"* /> </bean> |

##### 2.2.5.3.2 Setter – based dependency injection

Thông thường với mỗi thuộc tính XXX trong một lớp, sẽ có các phương thức getXXX() và setXXX() tương ứng với nó. Qua phương thức setXXX(), thuộc tính XXX sẽ được gán cho một giá trị nào đó (giá trị này có thể thuộc kiểu primitive hay kiểu tham chiếu). Spring cũng cung cấp cho chúng ta phương pháp để gán giá trị đến một thuộc tính qua phương thức setter của nó.

Setter – based DI là phương pháp tiêm sự phuộc cho các đối tượng bằng cách gọi các setter của một class sau khi hàm dựng mặc định (không đối số) được gọi để khởi tạo bean. Ví dụ dưới đây sẽ đưa ra một class chỉ có thể tiêm sự phụ thuộc bằng các setter.

**public** **class** Person {

**private** String fullName;

**private** **int** age;

**public** **void** setFullName(String fullName) { **this**.fullName = fullName;

}

**public** **void** setAge(**int** age) { **this**.age = age;

}

}

Khi đó để tiêm sự phụ thuộc cho lớp Person cần khai báo các bean như sau:

|  |
| --- |
| <bean id=*"person"* class=*"com.ctu.tqsang.Person"*>  <property name=*"fullName"*>  <value>Nguyen Thien An</value>  </property>  <property name=*"age"*>  <value>22</value>  </property>  </bean>    <!-- hoặc ngắn gọn hơn -->  <bean id=*"person"* class=*"com.ctu.tqsang.Person"*>  <property name=*"fullName"* value=*"Nguyen Thien An"* />  <property name=*"age"* value=*"22"* /> </bean> |

Thẻ <property> được dùng để biểu diễn cho một phương thức setter. Thuộc tính *name* của thẻ <property /> chỉ định thuộc tính mà setter của nó sẽ được gọi để gán giá trị được chỉ định trong thẻ <value /> (hoặc thuộc tính *value* như cách khai báo bean thứ 2). Thuộc tính *name* phải khớp với phần đuôi của phương thức setter. Ví dụ ta có phương thức là *setAField()*, thì lúc khai báo sẽ tương ứng là *name=”aField”*.

Chúng ta có thể kết hợp cả 2 phương pháp DI này trong khai báo một bean. Thông thường Constructor – based DI được sử dụng cho những dependency bắt buộc và Setter – based DI cho những dependency tùy chọn.

Dễ dàng nhận thấy rằng tất cả các class ở những ví dụ trên đều là những POJOs (những class Java bình thường), chúng hoàn toàn không kế thừa từ bất cứ một class hoặc thực thi một interface đặc biệt nào, đều này cho thấy được sức mạnh của Spring trong việc xây dựng một ứng dụng từ các POJOs là nhờ vào DI và IoC container.

Spring container xử lý việc tiêm sự phụ thuộc cho các bean như sau:

* Spring container ApplicationContext được tạo ra và khởi tạo những siêu dữ liệu cấu hình (configuration metadata) cho tất cả các bean. Các configuration metadata có thể là XML, code Java hoặc các annotaion.
* Đối với từng bean các dependency của nó được thể hiện dưới dạng là các property, các đối số của hàm tạo. Những dependency này sẽ phải được cung cấp cho bean khi bean thực sự được khởi tạo.
* Đối với từng property hoặc các đối số của hàm được gán giá trị để thiết lập cho bean hoặc được tham chiếu đến một bean khác trong container.
* Giá trị của từng property và đối số của hàm tạo sẽ được chuyển thành kiểu dữ liệu phù hợp, Spring sẽ chuyển chúng từ kiểu String sang các kiểu dữ liệu khác như int, long, String, boolean,…

Spring container sẽ kiểm tra tính hợp lệ của các thông tin cấu hình cho mỗi bean khi container được tạo. Tuy nhiên các thuộc tính của bean sẽ không được thiết lập cho đến khi bean thật sự được tạo ra và một bean chỉ được tạo khi có một yêu cầu đến nó.

##### 2.2.5.3.3 Autowiring

Khi một bean A cần tham chiếu đến bean B (B là một dependency của A) chúng ta cần phải cấu hình cho bean A để có thể “tiêm” B vào thông qua Constructor – based DI hoặc Setter – based DI. Tuy nhiên nếu chúng ta cấu hình cho bean A autowire thì Spring container có thể thực hiện việc này một cách “tự động” bằng cách Spring container sẽ kiểm tra trong ApplicationContext để tìm ra B và “tiêm” cho A.

Autowiring thực sự hữu ích khi phát triển một ứng dụng mà việc chỉnh sửa mã nguồn là thường xuyên, ví dụ khi thêm một thuộc tính mới cho một class, nếu sử dụng autowire thì cấu hình bean của class này không cần phải thay đổi.

Để cấu hình cho một bean có thể autowire chỉ cần thêm thuộc tính *autowire* trong thẻ <bean />. Autowire có 4 loại vì vậy cần phải chỉ rõ loại của autowire đối với mỗi bean. Dưới đây là 4 loại của autowire:

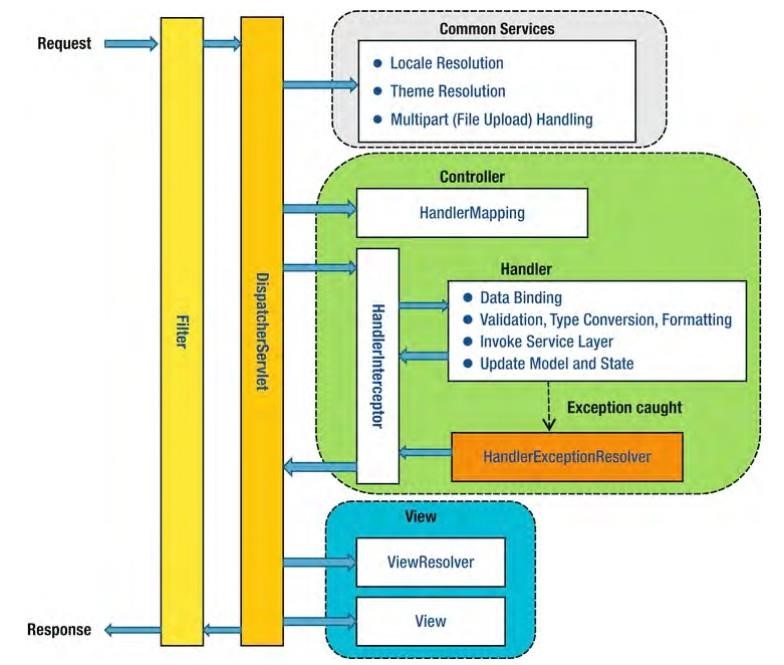
* Autowire by name.Spring sẽ tìm kiếm bean có tên giống như tên của property cần autowire. Ví dụ nếu bean A được chỉ định là autowire by name có property tên *master* (class A cần phải có phương thức setMasster(…)) , Spring sẽ tìm một bean có tên (id) là *master* và tiêm nó vào bean A.
* Autowire by type.Autowire by type cho phép một property được autowire nếu có một bean có kiểu (class) là kiểu của property này. Nếu có nhiều hơn một bean thì một exception được ném ra, ngược lại nếu không tồn tại bất cứ bean nào phù hợp thì không có exception nào được ném ra và property này không được thiết lập (set) giá trị.
* Autowire by constructor. Tương tự như autowire by type, tuy nhiên autowire by constructor chỉ áp dụng cho các đối số của hàm tạo. Nếu không có chính xác một bean có kiểu là kiểu của các đối số trong hàm tạo thì một exception sẽ được ném ra.
* No autowire. Không autowire, đây là thiết lập mặc định của các bean. Các bean tham chiếu đến nhau thông qua thuộc tính *ref.* No autowire được khuyến cáo nên sử dụng, bởi vì việc chỉ định các dependency một cách rõ ràng sẽ tốt hơn và ở một mức độ nào đó nó được xem như tài liệu cho cấu trúc của hệ thống.

Lưu ý là các dependency được chỉ định một cách tường minh sẽ ghi đè lên lên autowire. Đối với các property có kiểu dữ liệu nguyên thủy như String, Classes… thì không thể autowire.

### 2.2.6 Spring MVC

#### 2.2.6.1 Tổng quan về Spring MVC

Spring MVC là một module con trong Spring framework, cung cấp kiến trúc Model-View-Controller và các components sẵn có để sử dụng và phát triển ứng dụng web một cách linh hoạt.



###### Hình 2.10: Kiến trúc module Spring MVC

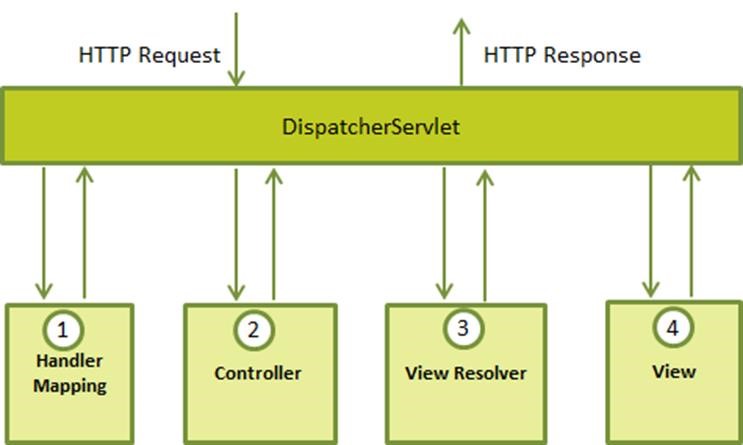
Mô hình MVC là kết quả của việc tách các khía cạnh khác nhau của ứng dụng (logic đầu vào, các xử lý logic, UI) trong khi đó cung cấp một sự kết hợp giữa các thành phần đó một cách “lỏng lẻo”.

Model: đóng gói dữ liệu ứng dụng và bao gồm các POJO.

View: Chịu trách nhiệm nhận giá trị của model và vẽ ra trang HTML mà trình duyệt có thể hiển thị được.

Controller: Chịu trách nhiệm nhận và xử lý các yêu cầu từ người dùng và tạo các model phù hợp và trả về cho view.

Spring MVC được thiết kế xung quanh DispatcherServlet để xử lý tất cả các HTTP request và HTTP response. Luồng xử lý các yêu cầu của DispatcherServlet được minh họa theo hình:



###### Hình 2.11: Sơ đồ luồng xử lý của Spring MVC

Đây là chuỗi sự kiện tương ứng khi nhận một yêu cầu HTTP gửi đến

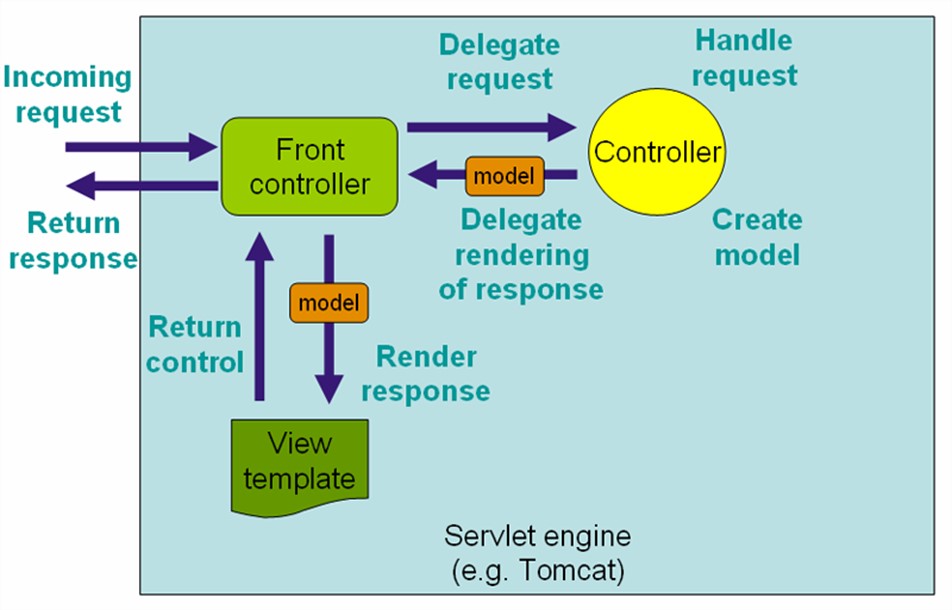
DispatcherServlet:

1. Sau khi nhận một HTTP request, DispatcherServlet gửi yêu cầu đến HandlerMapping (một bản đồ cấu hình URL) để xác định controller nào sẽ xử lý yêu cầu này.
2. Controller sẽ nhận các request và gọi các phương thức dịch vụ (service methods) phù hợp dựa trên việc sử dụng các method GET/POST… Nếu yêu cầu đó cần truy xuất cơ sở dữ liệu thì Controller sẽ ủy nhiệm cho một business logic hay nhiều hơn một model để lấy thông tin và gửi dữ liệu về cho Controller. Lúc này Controller đóng gói mô hình dữ liệu và tên của một view sẽ được tải lên thành đối tượng ModelAndView và gửi trả về cho DispathcherServlet
3. DispatcherServlet gửi gói ModelAndView cho ViewResolver để tìm xem view (thông thường là một trang JSP, HTML) nào sẽ được load lên.
4. Sau khi hoàn tất, DispatcherServlet sẽ chuyển dữ liệu từ model đến view và cuối cùng hiển thị trên trình duyệt.

#### 2.2.6.2 DispathcherServlet

DispathcherServlet là thành phần tiếp nhận tất cả các request từ browser, điều khiển luồng xử lý và trung chuyển giữa các thành phần (components) trong MVC. Hơn thế nữa, nó còn tích hợp các Spring IoC container, cho phép ta sử dụng đến tất cả các tính năng của Spring.

Hình 2-11 sẽ cho chúng ta thấy luồng xử lý một request trong Spring MVC và DispatcherServlet chính là Front controller.



###### Hình 2.12: Luồng xử lý một request trong Spring MVC

DispatcherServlet thực sự là một Servlet (nó thừa kế từ lớp cơ sở HttpServlet), và được khai báo trong tập tin web.xml của ứng dụng web. Các requests muốn DispatcherServlet xử lý sẽ phải được ánh xạ bằng cách sử dụng một bộ ánh xạ URL trong cùng một tập tin web.xml. Dưới đây là ví dụ về khai báo và định nghĩa DispatcherServlet

|  |
| --- |
| <?xml version=”1.0” encoding=”UTF-8”?>  <webapp>  <servlet>  <servlet-name>question-manager</servlet-name>  <servlet-class> org.springframework.web.servlet.DispatcherServlet  </servlet-class>  <load-on-startup>1</load-on-startup> </servlet>    <servlet-mapping>  <servlet-name>question-manager</servlet-name>  <url-pattern>/question/\*</url-pattern>  </servlet-mapping> </webapp> |

Như ta thấy, tất cả các requests bắt đầu bằng **/question** sẽ được DispatcherServlet **‘question-manager’** xử lý.

Khi DispatcherServlet được khởi tạo, Spring sẽ gọi file có tên **[tên-servlet]servlet.xml** trong thư mục **WEB-INF**. Với ví dụ trên, Spring sẽ gọi file **musicmanager-servlet.xml.**

#### 2.2.6.3 Controllers

Controllers là các thành phần được DispatcherServlet gọi để xử lý các nghiệp vụ business. Tất cả các Controller trong Spring đều implement từ Controller interface trong package org.springframework.web.servlet.mvc

**package** org.springframework.web.servlet.mvc;

**public** **interface** Controller {

ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) **throws** Exception;

}

Controller interface định nghĩa ra 1 phương thức chịu trách nhiệm xử lí 1 request và trả về model và view thích hợp. Ba khái niệm này chính là nền tảng của Spring MVC framework : Model – View – Controller.

Sau đây là các Controller sử dụng được trong Spring:

* SimpleFormController
* AbstractController
* AbstractCommandController
* CancellableFormController
* AbstractCommandController
* MultiActionController
* ParameterizableViewController
* ServletForwardingController
* ServletWrappingController
* UrlFilenameViewController

Kể từ khi Spring 2.5 giới thiệu và đưa vào mô hình lập trình cho các MVC controller dựa trên các annotation như @Controller, @RequestMapping, @RequestParam, @PathVariable,… thì việc tạo ra một controller trở nên vô cùng đơn giản.

Để tạo ra một controller chúng ta không cần kế thừa từ một lớp cơ sở hoặc thực thi bất kỳ một interface nào. Hãy xét 2 ví dụ dưới đây để thấy rõ hơn.

Ví dụ 1: Tạo controller bằng cách thực thi giao diện AbstractController

Khi sử dụng AbstractController làm baseclass cho các controller, chỉ cần override hàm *handleRequestInternal (HttpServletRequest, HttpServletResponse),* viết code xử lý, và trả về một đối tượng ModelAndView

|  |
| --- |
| **public** **class** HomeController **implements** AbstractController {  @Override  **public** ModelAndView handleRequest(HttpServletRequest request,  HttpServletResponse response) **throws** Exception { ModelAndView modelAndView = **new** ModelAndView(); model.addObject("message", "This is welcome page!");  model.setViewName("hello");  **return** modelAndView;  }  } |

Controller này sẽ trả về một view “hello” được code cứng.

Ví dụ 2: Tạo controller sử dụng các annotation

|  |
| --- |
| **package** com.ctu.tqsang.controller;    @Controller  @RequestMapping(value = "/home") **public** **class** HomeController {    @RequestMapping(value = "/welcome", method = RequestMethod.***GET***) **public** ModelAndView hello() {  ModelAndView model = **new** ModelAndView(); model.addObject("message", "This is welcome page!");  model.setViewName("hello");  **return** model;  }  } |

Như chúng ta thấy với 2 annotation @Controller và @RequestMapping đã có thể tạo ra 1 controller như ví dụ 1 một cách dễ dàng. Ngoài ra dễ nhận thấy khi có một request với phương thức GET được gửi đến URL “/home/welcome” thì hàm hello( ) của lớp HomeController sẽ được gọi để xử lý và trả kết quả về cho browser. Trong ví dụ này sẽ trả về view “hello” cho browser.

Nếu chúng ta tạo ra các controller như ví dụ 1 thì cần phải khai báo và định nghĩa các HandlerMapping.

#### 2.2.6.4 Handler Mapping

Sử dụng Handler Mapping để có thể ánh xạ từng requests đến các handler thích hợp. Các chức năng cơ bản của một HandlerMapping là đưa ra một Handler Execution Chain (chuỗi hành động), và có thể chứa một số bộ chặn. Khi requests đến, các DispatcherServlet sẽ trao cho các HandlerMapping để nó kiểm tra và xuất ra HandlerExecutionChain thích hợp. Sau đó, các DispatcherServlet sẽ gọi handler và bộ chặn trong hệ thống (nếu có).

Một Handler Mapping đơn giản nhưng rất mạnh mẽ là

Bean Name Url Handler Mapping, nó ánh xạ các HTTP requests đến tên của các beans, được định nghĩa trong bối cảnh ứng dụng web. Ví dụ chúng ta muốn cho phép một người dùng tạo một tài khoản, ở đây ta đã có một form controller thích hợp và một trang JSP để hiển thị.

Khi sử dụng Bean Name Url Handler Mapping, ta có thể ánh xạ HTTP requests với địa chỉ http://samples.com/editaccount vào form Controller như sau:

<beans>

<bean id="handlerMapping" class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<bean name="/edituser" class="org.springframework.web.servlet.mvc.SimpleFormController">

<property name="formView" value="user"/>

<property name="successView" value="user-created"/>

<property name="commandName" value="user"/>

<property name="commandClass" value="samples.User"/>

</bean>

<beans>

Tất cả requests gửi đến yêu cầu địa chỉ /editaccount bây giờ sẽ được xử lý bởi form Controller ở trên.

Cơ chế handler mapping của Spring có khái niệm xử lý đánh chặn, nó có thể cực kỳ hữu ích khi chúng ta muốn áp dụng các chức năng cụ thể cho một request xác định, ví dụ kiểm tra điều kiện chẳng hạn.

Ví dụ dưới đây tạo ra một bộ chặn, nó chặn tất cả các requests và redirect người dùng đến một trang xác định nếu thời gian truy cập không nằm từ 9h sáng đến 6h tối.

<beans>

<bean id=*"handlerMapping"* class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>

<property name=*"interceptors"*>

<list>

<ref bean=*"officeHoursInterceptor"*/>

</list>

</property>

<property name=*"mappings"*>

<value>

/\*=editUserFormController

</value>

</property>

</bean>

<bean id=*"officeHoursInterceptor"*

class=*"samples.TimeBasedAccessInterceptor"*> <property name=*"openingTime"* value=*"9"*/>

<property name=*"closingTime"* value=*"18"*/>

</bean>

<beans>

|  |
| --- |
| **public** **class** TimeBasedAccessInterceptor **extends** HandlerInterceptorAdapter {    **private** **int** openingTime; **private** **int** closingTime;    **public** **void** setOpeningTime(**int** openingTime) { **this**.openingTime = openingTime;  }  **public** **void** setClosingTime(**int** closingTime) { **this**.closingTime = closingTime;  }  **public** **boolean** preHandle(  HttpServletRequest request,  HttpServletResponse response,  Object handler) **throws** Exception {    Calendar cal = Calendar.*getInstance*(); **int** hour = cal.get(0);  **if** (openingTime <= hour && hour < closingTime) { **return** **true**;  } **else** {  response.sendRedirect("http://host.com/outsideOfficeHours"); **return** **false**;  }  }  } |

#### 2.2.6.5 ViewResolver

Views trong Spring được gán “view name” và được phân giải bởi ViewResolver. Sau đây là các ViewResolver trong Spring:

* BeanNameViewResolver
* FreeMarkerViewResolver
* InternalResourceViewResolver
* JasperReportsViewResolver
* ResourceBundleViewResolver
* UrlBasedViewResolver
* VelocityLayoutViewResolver
* VelocityViewResolver
* XmlViewResolver
* XsltViewResolver

Ở đây xin trình bày Internal Resource View Resolver vì nó thường được sử dụng nhiều nhất. View Resolver này sẽ ánh xạ Logical name của Resource (ModelAndView object được trả về bởi Controller) đến một Physical View (myView.jsp,…). Chúng ta có controller như sau:

|  |
| --- |
| **public** **class** MyController { **public** ModelAndView handle(){ **if**(condition1()){  **return** **new** ModelAndView("myView1");  }**else** **if** (condition2()){  **return** **new** ModelAndView("myView2");  }  **return** **new** ModelAndView("myView3");  }  } |

Chúng ta cần phải khai báo một Internal Resource View Resolver trong file web.xml như sau:

<bean id=*"viewResolver"*

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"*>

<value>/WEB-INF/views</value>

</property>

<property name=*"suffix"*>

<value>.jsp</value>

</property>

</bean>

Khi đó nếu như request của người dùng thỏa ***condition1()***, thì view “**myView.jsp**” trong thư mục /WEB-INF/views sẽ được hiển thị, nếu không thì hoặc “**myView2.jsp**” hoặc “**myView3.jsp**” sẽ được hiển thị.

### 2.2.7 Spring Security

#### 2.2.7.1 Tổng quan về Spring Security

Spring Security cung cấp một giải pháp bảo mật toàn diện cho các ứng dụng enterprise dựa trên J2EE, Spring Security cố gắng để cung cấp cho chúng ta một hệ thống bảo mật hữu ích và dễ dàng để cấu hình cho các dự án được phát triển dựa trên Spring framework.

Vào năm 2003, Spring Security ban đầu chỉ là một dự án nhỏ và độc lập với Spring với tên gọi là “The Acegi Security System for Spring”. Đến tháng 3/2004 một dự án chính thức được thành lập để phát triển nó. Khoảng một năm sau khi dự án được giới thiệu Acegi Security chính thức trở thành một dự án con (subproject) của Spring

Framework. Phiên bản 1.0.0 được giới thiệu vào tháng 5/2006 sau hơn hai năm rưỡi phát triển với hàng trăm cải tiến đáng kể. Cuối năm 2007 Acegi Security chính thức trở thành một project của Spring và được đổi tên thành Spring Security.

Hai nhiệm vụ chính của việc bảo mật ứng dụng là "xác thực" (authentication) và

"ủy quyền" (authorization) hay "kiểm soát truy cập" (access-control), đây cũng chính là hai phần chính của Spring Security.

Xác thực là quá trình xác nhận và chứng thực một principal (một principal thường có nghĩa là một người sử dụng, thiết bị hoặc một số hệ thống khác mà có thể thực hiện một hành động trong ứng dụng).

Uỷ quyền đề cập đến quá trình quyết định có một principal được phép thực hiện những hành động gì trong phạm vi ứng dụng của chúng ta [2, tr2]. Để được ủy quyền thì quá trình xác thực là cần thiết và phải được thực hiện trước, ủy quyền chỉ được thực hiện khi xác thực thành công. Đây là những khái niệm rất phổ biến cho hầu hết các ứng dụng và nó không phải là riêng của Spring Security.

Ở một mức độ xác thực, Spring Security hỗ trợ một loạt các mô hình xác thực. Hầu hết các mô hình xác thực này hoặc được cung cấp bởi bên thứ ba, hoặc được phát triển bởi các cơ quan như IETF (Internet Engineering Task Force). Ngoài ra, Spring Security còn cung cấp các mô hình xác thực riêng cho mình. Cụ thể, Spring Security hiện đang hỗ trợ tích hợp xác thực với tất cả các công nghệ này:

* HTTP BASIC authentication headers (an IEFT RFC-based standard)
* HTTP Digest authentication headers (an IEFT RFC-based standard)
* HTTP X.509 client certificate exchange (an IEFT RFC-based standard)
* LDAP (một phương pháp xác thực cross-platform phổ biến thường được sử dụng trong các môi trường lớn)
* Form-based authentication
* OpenID authentication
* Authentication based on pre-established request headers (such as Computer Associates Siteminder)
* JA-SIG Central Authentication Service (hay còn gọi là CAS, một mã nguồn mở single sign)
* Transparent authentication context propagation for Remote Method Invocation

(RMI) and HttpInvoker (a Spring remoting protocol)

* Automatic "remember-me" authentication
* Anonymous authentication
* Run-as authentication
* Java Authentication and Authorization Service (JAAS)
* JEE container autentication
* Kerberos
* Java Open Source Single Sign On (JOSSO)
* OpenNMS Network Management Platform
* AppFuse
* AndroMDA
* Mule ESB
* Direct Web Request (DWR)
* Grails
* Tapestry
* Jtrac
* Jasypt - Roller
* Elastic Path
* Atlassian Crowd

Nhiều công ty phần mềm áp dụng Spring Security vì nó cung cấp những mô hình xác thực một cách linh hoạt. Làm như vậy cho phép các công ty phần mềm nhanh chóng tích hợp các giải pháp của họ với bất cứ khách hàng nào, mà không thực hiện rất nhiều kỹ thuật hoặc yêu cầu khách hàng phải thay đổi môi trường. Nếu không có cơ chế xác thực phù hợp với nhu cầu của khách hàng, Spring Security là một nền tảng mở và nó thực sự đơn giản để viết cơ chế xác thực của riêng chúng ta.

Đôi khi chỉ xác thực là không đủ, chúng ta cũng cần phải quan tâm đến việc nó tích hợp vào ứng dụng như thế nào. Ví dụ, chúng ta có thể muốn đảm bảo rằng tất cả các request chỉ đến qua HTTPS, để bảo vệ mật khẩu khỏi bị nghe trộm hay từ các cuộc tấn công man-in-the-middle. Điều này đặc biệt hữu ích để bảo vệ quá trình phát hiện mật khẩu từ các cuộc tấn công brute force, hoặc chỉ đơn giản là để làm cho nó khó khăn hơn cho người dùng để sao chép nội dung quan trọng của ứng dụng. Để giúp chúng ta đạt được những mục tiêu này, Spring Security hỗ trợ "channel security", cùng với JCaptcha để giải quyết những yêu cầu trên.

#### 2.2.7.2 Các module của Spring Security

Spring Security được chia thành 7 module nhỏ dựa trên chức năng của từng module và các thành phần yêu cầu từ bên thứ 3:

* ***spring-security-core.jar:***Bao gồm các class và interface cơ bản cho quá trình xử lý xác thực và ủy quyền, hổ trợ các API cơ bản mà các ứng dụng sử dụng Spring Security cần. Bao gồm các package sau:

+ org.springframework.security.core

+ org.springframework.security.access

+ org.springframework.security.authentication

+ org.springframework.security.provisioning

+ org.springframework.security.remoting

* ***spring-security-web.jar:*** Module này chứa các filter và những thành phần khác có liên quan đến việc bảo mật một ứng dụng web, bất cứ thành phần nào có liên quan đến Servlet. Chúng ta cần phải sử dụng module này nếu muốn xác thực trên các dịch vụ web và điều khiển truy cập thông qua URL. Package chính của module này là: org.springframework.security.config
* ***spring-security-config.jar:*** Module này là cần thiết nếu chúng ta cấu hình cho

Spring Security bằng file XML. Nó chứa các namspace dùng để parse chúng sang code. Package chính của lớp này org.springframework.security.config

* **s*pring-security-ldap.jar:*** Đây là module dành cho việc xác thực theo mô hình LDAP (Lightweight Directory Access Protocol). Package chính của lớp này là

org.springframework.security.ldap

* ***spring-security-acl.jar:*** ACL (Access Control List) là một danh sách các quyền gắn liền với một đối tượng. Một ACL xác định người dùng hoặc các quá trình của hệ thống được cấp quyền truy cập, cũng như những gì được cho phép hoạt động trên các đối tượng nhất định. Mỗi mục trong một ACL điển hình xác định một chủ thể và một hoạt động. Ví dụ, nếu một tập tin có chứa ACL (Alice: read, write; Bob: read), điều này sẽ cho phép Alice đọc và ghi các file và Bob chỉ đọc nó. Module này sẽ cho phép chúng ta tạo ra một mô hình xác thực như trên.

Package chính của module này là

org.springframework.security.acls

* ***spring-security-cas-client.jar:*** Module này chứa các thành phần cần thiết để tích hợp Spring Security và CAS client (Central Authentication Service) nếu chúng ta sử dụng một CAS single sign-on server. Pakage chính chính của module này là: org.springframework.security.cas
* ***spring-security-openid.jar:*** Module này hổ trợ mô hình xác thực OpenID, sử dụng để chứng thực người dùng bên ngoài hệ thống với một OpenID server.

Pakage chính chính của module này là: org.springframework.security.openid

## 2.6 BOOTSTRAP VÀ RESPONSIVE

Bootstrap là front-end framework, là một bộ sưu tập miễn phí các công cụ để tạo ra các trang web và các ứng dụng web. Bootstrap bao gồm HTML và CSS dựa trên các mẫu thiết kế cho kiểu chữ, hình thức, các button và các thành phần giao diện khác, cũng như mở rộng tùy chọn JavaScript.

Boostrap định nghĩa sẵn các class CSS giúp người thiết kế giao diện website tiết kiệm rất nhiều thời gian. Các thư viện Bootstrap có những đoạn mã sẵn sàng cho chúng ta áp dùng vào website của mình mà không phải tốn quá nhiều thời gian để tự viết.

Với Bootstrap, việc phát triển giao diện website để phù hợp với đa thiết bị trở nên dễ dàng hơn bao giờ hết. Đây là xu hướng phát triển giao diện website đang rất được ưu chuộng hiện nay. Bootstrap cung cấp tính năng responsive và mobile first, nghĩa là làm cho trang web có thể tự co giãn để tương thích với mọi thiết bị khác nhau, từ điện thoại di động đến máy tính bảng, máy tính xách tay, máy tính để bàn,...

Một khía cạnh khác là responsive web design làm cho trang web cung cấp được trải nghiệm tuyệt vời cho người dùng trên nhiều thiết bị, kích thước màn hình khác nhau. Đây là một đặc điểm vô cùng quan trọng vì chúng ta không thể lường trước được các thiết bị, kích thước màn hình sẽ sử dụng để truy cập vào trang web. Một trang có thể hoạt động tốt bất kể sự biến đổi sẽ cung cấp một trải nghiệm người dùng tốt và nhất quán hơn một trang được thiết kế cho một loại thiết bị và kích thước màn hình cụ thể.

# Chương 3 THIẾT KẾ VÀ CÀI ĐẶT CHƯƠNG TRÌNH

## 3.1 MÔ TẢ BÀI TOÁN

### 3.1.1 MÔ TẢ TỔNG QUAN

#### 3.1.1.1 Bối cảnh hệ thống

Hiện nay, song song với việt xây dựng một website, một trang web cho cá nhân hoặt tổ chức, doanh nghiệp, công ty, .. đi đôi với việt xây đựng một trang web quản lý các sản phẩm thì sẽ xuất hiện một số yêu cần tất yếu là nhu cầu của khách hàng là trao đổi thông tin. Càng nhận được nhiều thông tin thuộc về một chủ đề cụ thể thì người đó càng hiểu sâu vấn đề hơn. Thông tin được trao đổi, có nghĩa là phải có người nói, người nghe và thảo luận. Thảo luận sẽ kích thích tư duy, trí tò mò và mong muốn chia sẻ kiến thức.

Đó là lý do tại sao khi học bất cứ thứ gì mới, nếu ta có một nhóm người cùng thảo luận thì sẽ đẩy mạnh tốc độ học tập của cả nhóm. Đấy là lý do ta cần diễn đàn.

#### 3.1.1.2 Các chức năng chính

Người dùng(user) là những người đã được cấp tài khoản để đăng nhập vào trang web dùng lưu trữ các thông tin như: mã người dùng(userid), tên đăng nhập(username), mật khẩu(password), ảnh đại diện(avatar), điểm hoạt động(point), ngày tạo(datecreate), lần cuối đăng nhập vào hệ thống(latelogin), lần cuối cập nhật(lateupdated).

Người dùng khi được tạo ra sẽ được phân quyền(role) cụ thể, mỗi người dùng có thể có nhiều quyền khác nhau. Có 2 loại người dùng là nhân viên(employee) và khách hàng(customer).

Các thông tin cơ bản của nhân viên và khách hàng sẽ được lưu lại vào người(person) chứa các thông tin: mã(personid), họ(firstname), chữ lót(middlename), tên(lastname), giới tính(gender), ngày sinh(birthday), địa chỉ email(email), số điện thoại(phone), và địa chỉ cụ thể(address).

Địa chỉ(address) bao gồm: vị trí(location), nơi(place), khu vực(ward), huyện(district), tỉnh(province).

Về phần câu hỏi(question), mỗi câu hỏi sẽ có mã câu hỏi(questionid), nội dung(content), ngày tạo(datecredat), tiêu đề(title), liên kết tiêu đề(lug), ngày sữa đỗi(updateat), số lược xem(views). Mỗi câu hỏi sẽ thuộc một loại câu hỏi(categoryquestion), và sẽ có thể có nhiều tag(tag), nhiều câu trả lời(answer), đồng thời cũng sẽ được đánh giá(vote) từ người dùng.

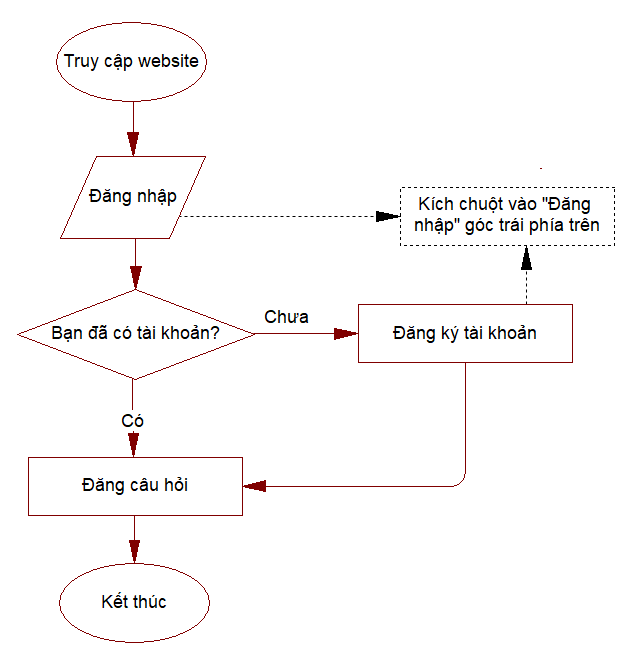
Câu trả lời sẽ có: mã số(answered), có tốt nhất cho câu hỏi chưa(isbest), nội dung(content), ngày trả lời(createdate), số lược bình chọn(votes).

#### 3.1.1.3 Đặc điểm người dùng

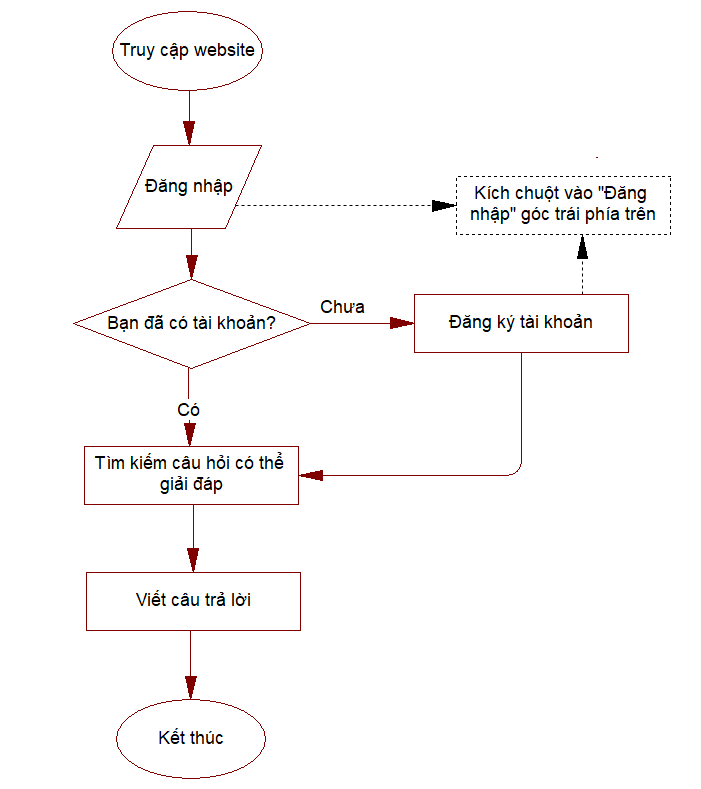
* Khách (Guest): Người dùng vãng lai, có thể vào diễn đàn để xem các chuyên mục, tìm kiếm các câu hỏi, và tham khảo các câu trả lời có sẵn, nếu có nhu cầu có thể tạo tài khoản.
* Thành viên (Customer): Có thể xem, trả lời, đánh giá câu hỏi, ngoài ra còn có thể đặc câu hỏi, truy cập trang cá nhân, chĩnh sữa thông tin cá nhân.
* Nhân viên (Employee): Ngoài những hành động của thành viên, nhân viên còn có thể truy cập vào trang admin để quản lý và thống kê về câu hỏi, thành viên, câu trả lời,…

### 3.1.2 MỘT SỐ QUY TRÌNH CƠ BẢN

#### **3.1.2.1 Quy trình đăng câu hỏi**:



#### 3.1.2.2 Quy trình thảo luận câu hỏi



## 3.2 SƠ ĐỒ USECASE

Bảng 3-1: Danh sách các actor

|  |  |
| --- | --- |
| **Actor** | **Vai trò** |
| User | User là đối tượng chính tác động trực tiếp và nhiều nhất vào hệ thống. User có thể thực hiện hầu như các chức năng của ứng dụng. |
| Admin | Admin là actor kế thừa actor user và thêm chức năng quản lý |

Bảng dưới đây trình bày các chức năng chính của ứng dụng.

Bảng 3-2: Danh sách các usecase

|  |  |
| --- | --- |
| **Chức năng** | **Mô tả** |
| Đăng nhập | Xác thực và phân quyền cho người dùng khi họ sử dụng diễn đàn |
| Đăng xuất | Thoát khỏi phiên làm việc. |
| Đăng ký | Đăng ký tài khoản để sử dụng một số chức năng yêu cầu |
| Thay đổi mật khẩu | Thay đổi mật khẩu người dùng |
| Thêm câu hỏi | Thêm câu hỏi mới vào ứng dụng |
| Sửa câu hỏi | Chỉnh sửa thông tin câu hỏi |
| Xóa câu hỏi | Xóa câu hỏi |
| Thêm câu trả lời | Thêm câu trả lời mới cho câu hỏi |
| Chỉnh sữa câu trả lời | Chỉnh sửa thông tin trả lời |
| Xóa câu trả lời | Xóa câu trả lời |
| Quản lý tất cả câu hỏi và trả lời của người dùng | Quản lý câu hỏi và câu trả lời của người dùng như lọc nội dung, xóa các câu trả lời không đúng chủ đề, xấp xếp lại câu hỏi. |
| Quản lý tất cả người dùng | Quản lý người dùng, chỉnh sữa thay đổi, phân quyền tài khoản, xóa tài khoản. |
| Quản lý các loại câu hỏi | Quản lý các loại câu hỏi, thêm sửa, xóa |
| Quản lý các thẻ của câu hỏi | Quản lý các thẻ của câu hỏi, thêm, sửa, xóa |

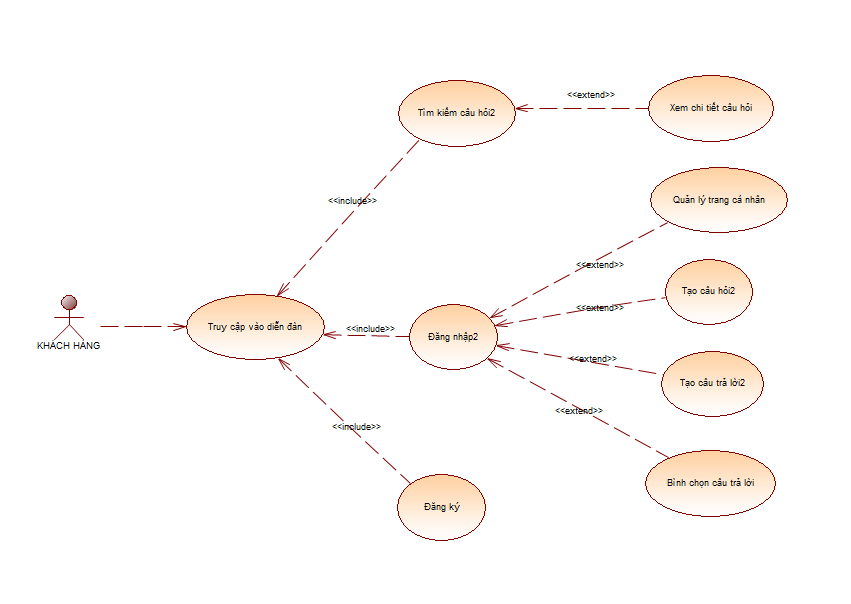
Dưới đây là sơ đồ mối quan hệ giữa các actor với actor, actor với use case và use case và use case.

### 3.2.1 Use case tổng quát cho từng tác nhân

#### 3.2.1.1 Người dùng khách:

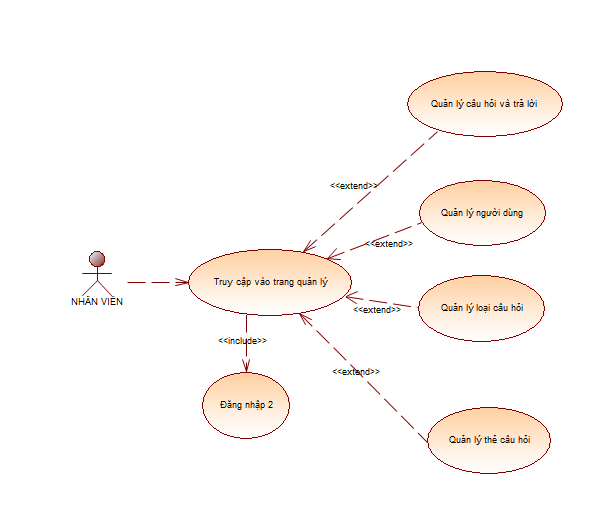
Tìm kiếm câu hỏi cần giải đáp hoặt đăng nhập vào hệ thống

Khách là những người sử dụng trang web có nhu cầu tìm kiếm thông tin



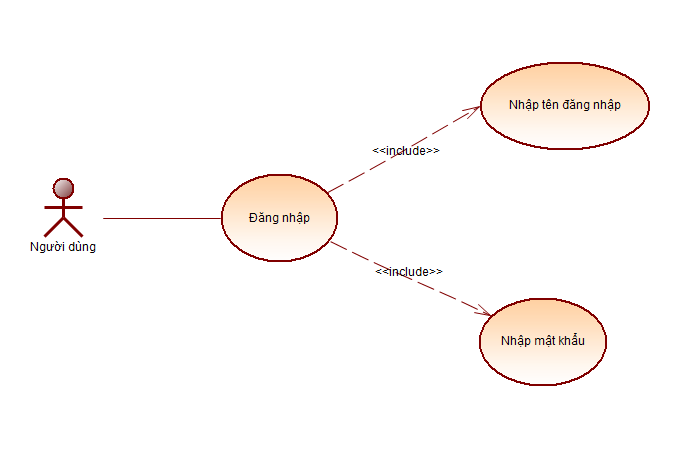
Hình 3.10: Sơ đồ use case của actor user

#### 3.2.1.2 Người dùng là nhân viên



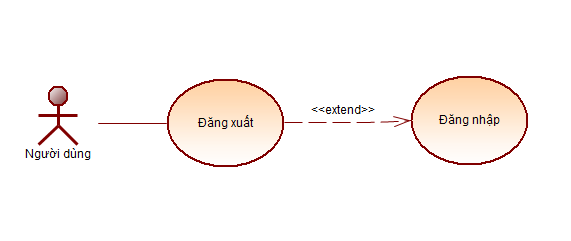
### 3.2.2 Phân rã Use case

#### 3.2.2.1 Người dùng đăng nhập



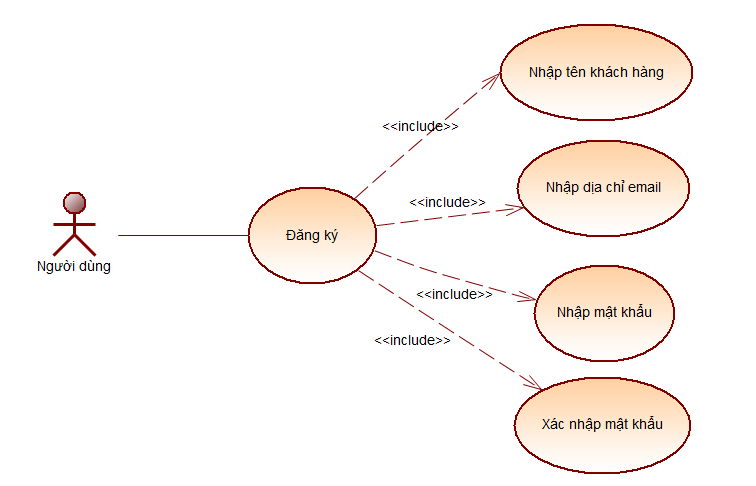
|  |  |
| --- | --- |
| Tên chức năng | Người dùng đăng nhập |
| Đối tượng sử dụng | Nhân viên hoặt khách hàng |
| Điều kiện sử dụng | Truy cập vào được diễn đàn và đã có tài khoảng người dùng |
| Cách xử lý | B1: Truy cập vào trang web chọn đăng nhập  B2: Nhập các thông tin tên truy cập và mật khẩu  B3: Chọn nút đăng nhập   * Nếu không có sự cố ngoài ý muốn thì sẽ đăng nhập thành công và trở về trang chủ * Ngược lại, nếu có sự cố ngoài ý muốn. Ví dụ như sai tên truy cập hay mật khẩu thì sẽ hiển thị thông báo, yêu cầu nhập lại cho đến khi thành công hoặt hủy bỏ đăng nhập. |
| Kết quả | Đăng nhập người dùng thành công hay thất bại |
| Ghi chú | Các thông tin bắt buộc nhập đầy đủ và chính sát: tên truy cập và mật khẩu. |

#### 3.2.2.2 Người dùng đăng xuất



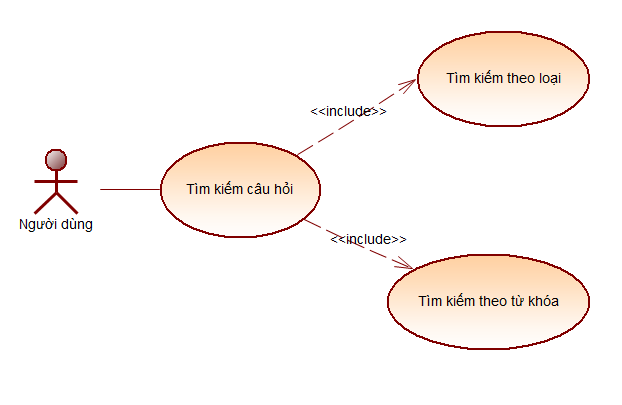
|  |  |
| --- | --- |
| Tên chức năng | Người dùng đăng xuất |
| Đối tượng sử dụng | Nhân viên hoặt khách hàng |
| Điều kiện sử dụng | Truy cập vào được diễn đàn và đã đăng nhập thành công |
| Cách xử lý | Chọn đăng xuất để thoát người dùng khỏi hệ thống |
| Kết quả | Đăng xuất thành công và chuyển về trang chủ diễn đàn |
| Ghi chú | Sau khi đăng xuất thành công thì tên của người dùng sẽ thay đổi thành đăng nhập |

#### 3.2.2.3 Người dùng đăng ký



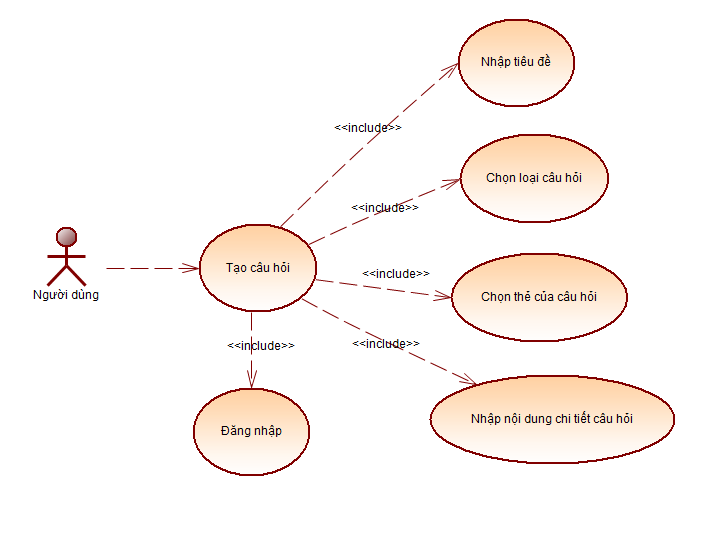
|  |  |
| --- | --- |
| Tên chức năng | Người dùng đăng ký |
| Đối tượng sử dụng | Khách hàng |
| Điều kiện sử dụng | Truy cập vào được diễn đàn |
| Cách xử lý | B1: Truy cập vào trang web chọn đăng ký  B2: Nhập các thông tin yêu cầu  B3: Chọn nút đăng ký   * Nếu không có sự cố ngoài ý muốn thì sẽ đăng nhập thành công và trở về trang chủ * Ngược lại, nếu có sự cố ngoài ý muốn. Ví dụ như tài khoản đã tồn tại thì sẽ hiển thị thông báo, yêu cầu nhập lại cho đến khi thành công hoặt hủy bỏ đăng ký. |
| Kết quả | Đăng ký người dùng thành công hay thất bại |
| Ghi chú | Các thông tin bắt buộc nhập đầy đủ và chính sát và không được trùng lập. |

#### 3.2.2.4 Người dùng tìm kiếm câu hỏi



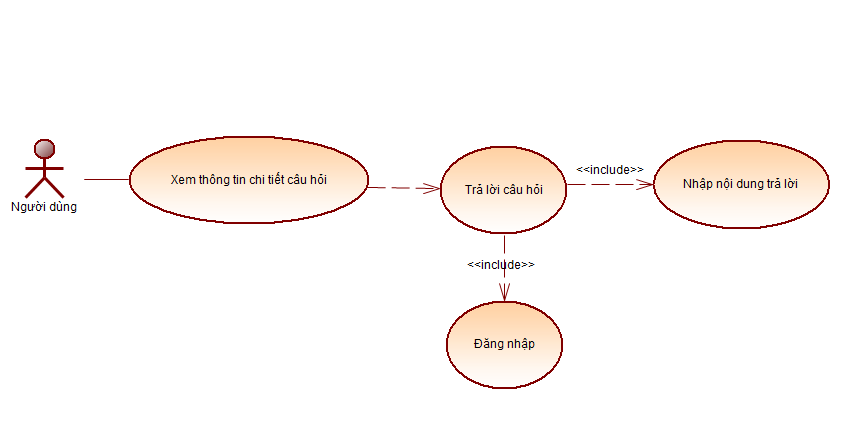
|  |  |
| --- | --- |
| Tên chức năng | Người dùng tìm kiếm câu hỏi |
| Đối tượng sử dụng | Tất cả những người sử dụng trang web |
| Điều kiện sử dụng | Truy cập vào được diễn đàn |
| Cách xử lý | Cách 1: Tìm kiếm theo loại câu hỏi:   * B1: Truy cập vào trang web chọn loại câu hỏi đã có * B2: Duyệt qua danh sach câu hỏi đã hiển thị   Cách 2: Tìm kiếm theo từ khóa   * B1: Truy cập vào trang web chọn thanh tìm kiếm * B2: Gõ vào các từ khóa cần tìm kiếm, duyệt qua danh sách kết quả |
| Kết quả | Tìm thấy được kết quả người dùng cần thành công hay thất bại |
| Ghi chú | Một số thông tin về từ khóa cần phải chính sát |

#### 3.2.2.5 Người dùng tạo câu hỏi



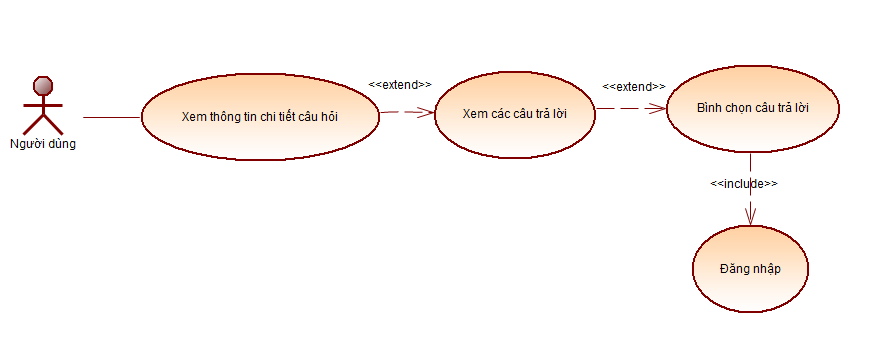
|  |  |
| --- | --- |
| Tên chức năng | Người dùng tạo câu hỏi |
| Đối tượng sử dụng | Nhân viên hoặt khách hàng |
| Điều kiện sử dụng | Truy cập vào được diễn đàn và đã đăng nhập thành công vào trang web, và không tìm thấy câu hỏi đã tồn tại từ diễn đàn |
| Cách xử lý | B1: Truy cập vào trang web chọn tạo câu hỏi  B2: Nhập, tùy chọn các thông tin cần thiết  B3: Chọn nút đăng câu hỏi   * Nếu không có sự cố ngoài ý muốn thì sẽ đăng câu hỏi thàng công và hiện thông báo kết quả * Ngược lại, nếu có sự cố ngoài ý muốn. Ví dụ như cung cấp chưa đủ thông tin thì sẽ hiển thị thông báo, yêu cầu nhập lại cho đến khi thành công hoặt hủy bỏ đăng câu hỏi. |
| Kết quả | Đăng câu hỏi thành công hay thất bại |
| Ghi chú | Các thông tin bắt buộc nhập đầy đủ và chính sát, kiểm tra kỹ trước khi đăng câu hỏi |

#### 3.2.2.6 Người dùng trả lời các câu hỏi



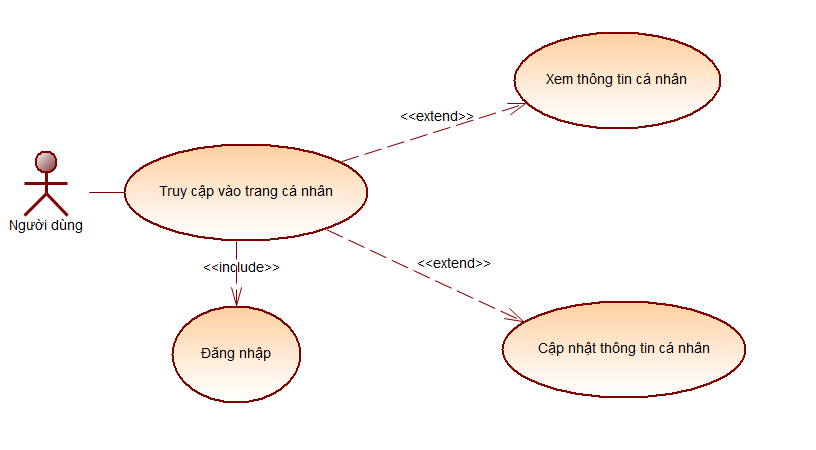
|  |  |
| --- | --- |
| Tên chức năng | Người dùng trả lời các câu hỏi |
| Đối tượng sử dụng | Nhân viên hoặt khách hàng |
| Điều kiện sử dụng | Truy cập vào được diễn đàn và đã đăng nhập thành công vào trang web |
| Cách xử lý | B1: Truy cập vào trang web chọn câu hỏi  B2: Người dùng để lại ý kiếm hay thảo luận của mình ở phần trả lời của câu hỏi  B3: Chọn nút đăng câu trả lời |
| Kết quả | Câu trả lời của người dùng được ghi nhận và hiển thị ở mục trả lời của câu hỏi |
| Ghi chú | Người dùng trả lời phải đúng mục đích của câu hỏi, nhằm giúp người đã đặc câu hỏi |

#### 3.2.2.7 Người dùng bình chọn một số câu trả lời



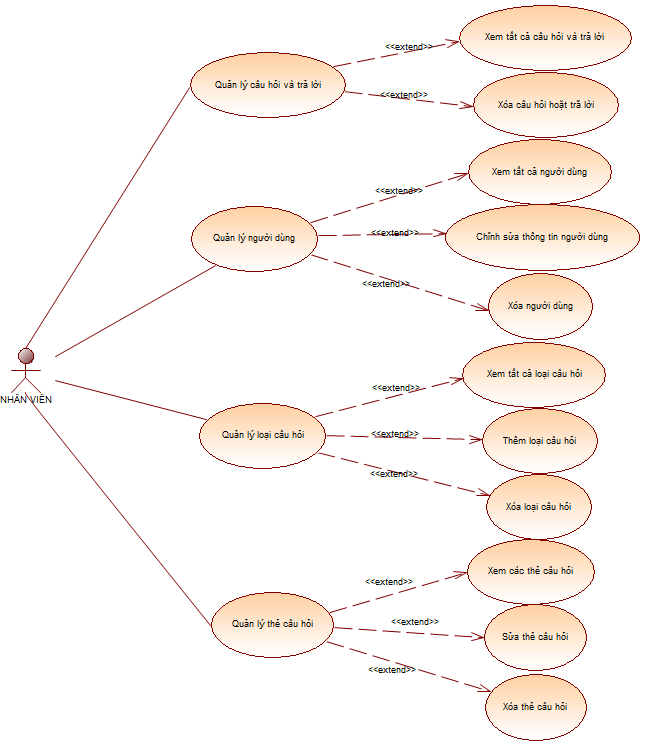
|  |  |
| --- | --- |
| Tên chức năng | Người dùng đánh giá câu trả lời |
| Đối tượng sử dụng | Nhân viên hoặt khách hàng |
| Điều kiện sử dụng | Truy cập vào được diễn đàn và đã đăng nhập thành công vào trang web |
| Cách xử lý | B1: Truy cập vào trang web chọn cẩu hỏi  B2: Xem các câu trả lời đã có và đánh giá mức độ khả thi của câu trả lời, nếu người dùng truy cập vào câu hỏi của mình và đã được giải đáp thành công thì có thể chọn câu trả lời đã giải quyết được vấn đề. |
| Kết quả | Đánh giá được độ trực quan của các câu trả lời |
| Ghi chú | Người dùng có thể thích hoặt không thích câu trả lời, nếu người dùng bình chọn nhiều lần sẽ hiệ thông báo hạn chế |

#### 3.2.2.8 Quản lý thông tin cá nhân



|  |  |
| --- | --- |
| Tên chức năng | Quản lý thông tin cá nhân |
| Đối tượng sử dụng | Nhân viên hoặt khách hàng |
| Điều kiện sử dụng | Truy cập vào được diễn đàn và đã đăng nhập thành công vào trang web |
| Cách xử lý | B1: Truy cập vào trang web chọn người quản lý tài khoảng  B2: Chọn xem thông tin cá nhân hoặt thay đổi thông tins |
| Kết quả | Xem được các thông tin cá nhân, tài nguyên của người dùng |
| Ghi chú | Người dùng có thể thay đổi thông tin cá nhân, mật khẩu, nhưng không thay đổi được tên đăng nhập |

#### 3.2.2.9 Nhân viên quản lý trang web



|  |  |
| --- | --- |
| Tên chức năng | Quản lý trang web |
| Đối tượng sử dụng | Nhân viên |
| Điều kiện sử dụng | Đăng nhập thành công và truy cập vào trang quản lý |
| Cách xử lý | B1: Đăng nhập và truy cập vào bảng điều khiển quản trị viên  B2: Xem thông tin tổng hợp ở trang chủ quản lý  B3: Chọn những phần cần quản lý ở danh mục như: quản lý người dùng, quản lý câu hỏi, quản lý thể loại, quản lý các thẻ |
| Kết quả | Xem được các hoạt động của trang web, quản lý các nội dung |
| Ghi chú | Về phần quản lý cần phải cẩn thận và chi tiết, lọc các nội dung không cần thiết. |

## 3.3 THIẾT KẾ CƠ SỞ DỮ LIỆU

### 3.7.1 Đặc tả các tập thực thể

3.7.1.1 Bảng tỉnh thành (province)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | provinceid | Integer |  | X | X | Mã tỉnh |
| 2 | name | Varchar | 100 |  |  | Tên tỉnh |
| 3 | type | Varchar | 30 |  |  | Loại(Thành phố hoặc tỉnh) |

3.7.1.2 Bảng huyện (district)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | districtid | Integer |  | X | X | Mã huyện |
| 2 | name | Varchar | 100 |  |  | Tên huyện |
| 3 | type | Varchar | 30 |  |  | Loại(Thành phố hoặc huyện) |
| 4 | location | Varchar | 30 |  |  | Tọa độ |

3.7.1.3 Bảng khu vực (ward)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | wardid | Integer |  | X | X | Mã khu vực |
| 2 | name | Varchar | 100 |  |  | Tên khu vực |
| 3 | type | Varchar | 30 |  |  | Loại(phường hoặc xã) |
| 4 | location | Varchar | 30 |  |  | Tọa độ |

3.7.1.3 Bảng địa chỉ (address)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | addressid | Integer |  | X | X | Mã địa chỉ |
| 2 | name | Varchar | 100 |  |  | Tên địa chỉ(Đường, số nhà) |
| 3 | place | Varchar |  |  |  | Nơi cụ thể(Tầng, lầu, phòng) |

3.7.1.1 Bảng người dùng (person)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | personid | Integer |  | X | X | Mã người dùng |
| 2 | firstname | Varchar | 50 |  |  | Họ người dùng |
|  | middlename | Varchar | 50 |  |  | Chữ lót người dùng |
|  | lastname | Varchar | 50 |  |  | Tên người dùng |
|  | gender | Integer |  |  |  | Giới tính |
|  | birthday | Varchar | 10 |  |  | Ngày sinh |
|  | email | Varchar | 100 |  |  | Chịa chỉ email |
|  | phone | Varchar | 20 |  |  | Số điện thoại |

3.7.1.1 Bảng khách hàng (customner)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | customerid | Integer |  | X | X | Mã khách hàng |
| 2 | deleted | Integer |  |  |  | Đã xóa hay chưa |

3.7.1.1 Bảng nhân viên (employee)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | employee | Integer |  | X | X | Mã nhân viên |
| 2 | type | Varchar | 30 |  |  | Loại(quản lý hay nhân viên) |

3.7.1.1 Bảng người dùng (user)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | userid | Integer |  | X | X | Mã người dùng |
| 2 | username | Varchar | 30 |  |  | Tên đăng nhập |
| 3 | password | Varchar | 60 |  |  | Mật khẩu |
| 4 | avatar | Text |  |  |  | Ảnh đại diện |
| 5 | point | Integer |  |  |  | Điểm hoạt động |
| 6 | datecreate | Varchar | 10 |  |  | Ngày tạo |
| 7 | lastlogin | Varchar | 10 |  |  | Lần đăng nhập cuối |
| 8 | lastupdated | Varchar | 10 |  |  | Lần cập nhật cuối |

3.7.1.1 Bảng quyền (role)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | roleid | Integer |  | X | X | Mã quyền |
| 2 | name | Varchar | 30 |  |  | Tên quyền |

3.7.1.1 Bảng câu hỏi (question)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | questionid | Integer |  | X | X | Mã câu hỏi |
| 2 | content | Varchar | 30 |  |  | Nội dung câu hỏi |
|  | createat | Date & Time |  |  |  | Thời gian tạo |
|  | slug | Text |  |  |  | Tiêu đề trên thanh url |
|  | title | Text |  |  |  | Tiêu đề |
|  | updatedat | Date & Time |  |  |  | Ngày cập nhật |
|  | views | Integer |  |  |  | Số lược xem |

3.7.1.1 Bảng trả lời (answer)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | answerid | Integer |  | X | X | Mã câu trả lời |
| 2 | isbest | Boolean |  |  |  | Câu trả lời chính sát |
| 3 | content | Text |  |  |  | Nội dung câu trả lời |
|  | createdat | Date&Time |  |  |  | Ngày tạo |
|  | votes | Integer |  |  |  | Số lược bình chọn |

3.7.1.1 Bảng đánh giá câu trả lời (vote)

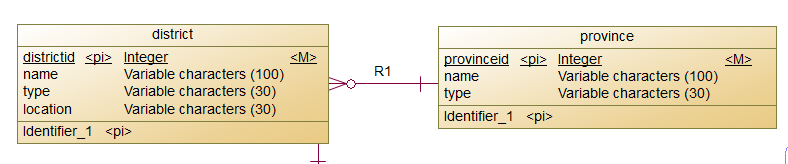
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | votequestionid | Integer |  | X | X | Mã đáng giá câu hỏi |

3.7.1.1 Bảng loại câu hỏi (categoryquestion)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STT | Tên trường | Kiểu | Kích thước | Khóa chính | Not null | Diễn giải |
| 1 | cqid | Integer |  | X | X | Mã loại câu hỏi |
| 2 | name | Varchar | 30 |  |  | Tên loại câu hỏi |

### 3.7.2 Các mối quan hệ:

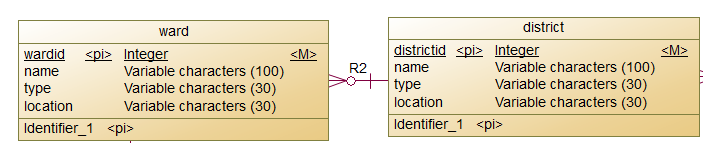
3.7.2.1 Mối quan hệ của tỉnh – huyện



Lớp tham gia: Tỉnh (province) và Huyện (district)

Diễn giải: Một tỉnh có chứa nhiều huyện, nhưng một huyện chỉ thuộc duy nhất một tỉnh

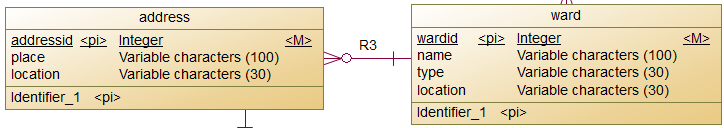
3.7.2.1 Mối quan hệ của khu vực – huyện



Lớp tham gia: Khu vực (ward) và Huyện (district)

Diễn giải: Một huyện có chứa nhiều khu vưch, nhưng một khu vực chỉ thuộc duy nhất một huyện

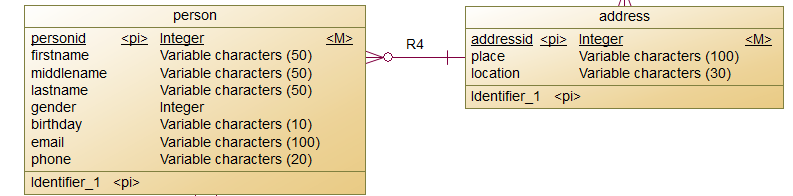
3.7.2.3 Mối quan hệ của địa chỉ - khu vực



Lớp tham gia: Khu vực (ward) và Địa chỉ (address)

Diễn giải: Một khu vực có chứa nhiều địa chỉ, nhưng một địa chỉ thuộc duy nhất một khu vực

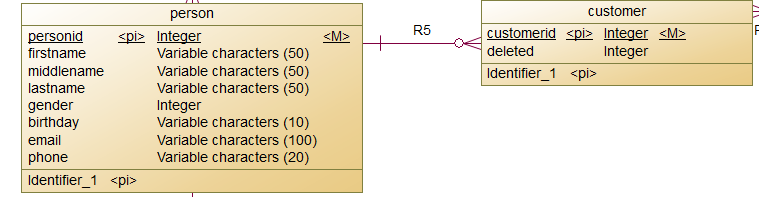
3.7.2.4 Mối quan hệ người dùng – địa chỉ



Lớp tham gia: Người dùng (person) và Địa chỉ (address)

Diễn giải: Một địa chỉ có thể chứa nhiều người dùng, nhưng người dùng chỉ thuộc duy nhất một địa chỉ

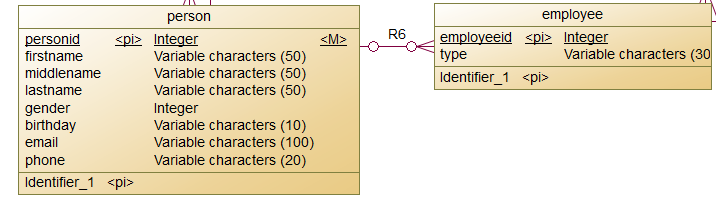
3.7.2.5 Mối quan hệ người dùng và khách hàng



Lớp tham gia: Người dùng (person) và khách hàng (customer)

Diễn giải: Khách hàng có thể chứa nhiều người dùng, nhưng người dùng chỉ có thể thuộc duy nhất một khách hàng hoặt không thuộc khách hàng

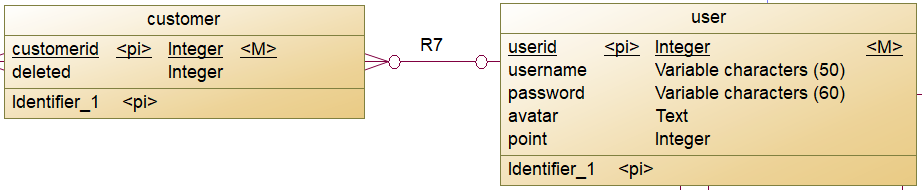
3.7.2.6 Mối quan hệ người dùng và nhân viên



Lớp tham gia: Người dùng (person) và nhân viên (employee)

Diễn giải: Nhân viên có chứa nhiều người dùng, nhưng người dùng chỉ có thể thuộc duy nhất một nhân viên hoặt không thuộc nhân viên

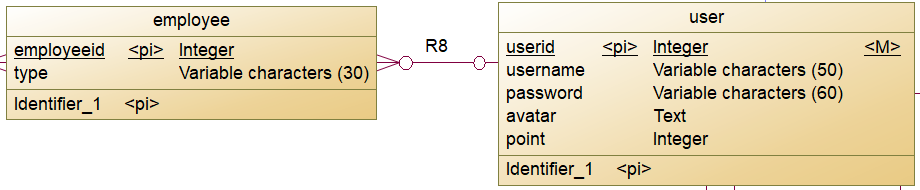
3.7.2.7 Mối quan hệ khách hàng – thành viên



Lớp tham gia: Thành viên (user) và khách hàng (customer)

Diễn giải: Khách hàng có chứa nhiều thành viên, nhưng thành viên chỉ có thể thuộc duy nhất một khách hàng hoặt không thuộc khách hàng

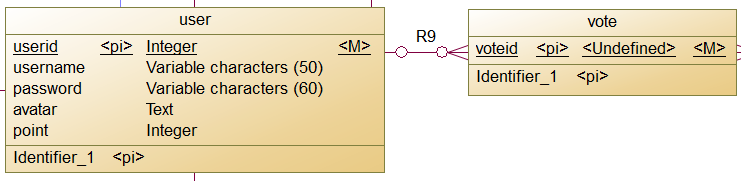
3.7.2.8 Mối quan hệ nhân viên – thành viên



Lớp tham gia: Thành viên (user) và nhân viên (employee)

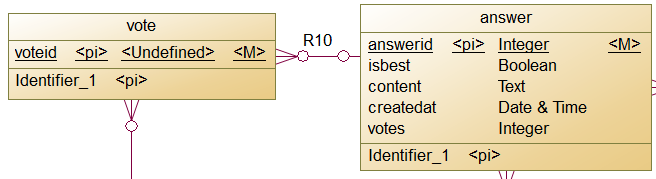
Diễn giải: Nhân viên có chứa nhiều thành viên, nhưng thành viên chỉ có thể thuộc duy nhất một nhân viên hoặt không thuộc nhân viên

3.7.2.11 Mối quan hệ thành viên – đánh giá

 Lớp tham gia: Thành viên (user) và đánh giá (vote)

Diễn giải: Thành viên có thể có nhiều đánh giá, nhưng mỗi lược đánh giá chỉ có thể do duy nhất một thành viên đánh giá

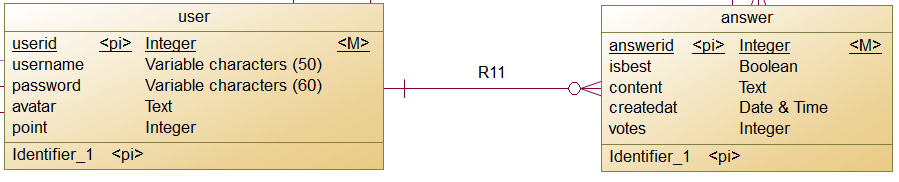
3.7.2.10 Mối quan hệ câu trả lời – đánh giá



Lớp tham gia: Câu trả lời (answer) và đánh giá (vote)

Diễn giải: Câu trả lời có thể có nhiều đánh giá, nhưng mỗi lược đánh giá chỉ có thể đánh gái một câu trả lời

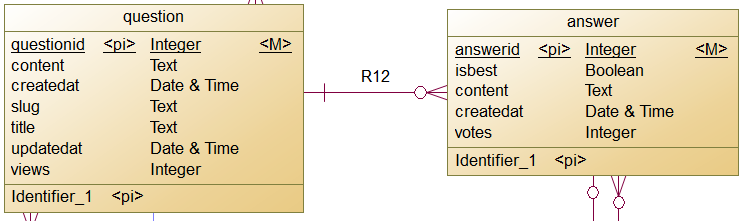
3.7.2.11 Mối quan hệ thành viên – trả lời



Lớp tham gia: Thành viên (user) và câu trả lời (answer)

Diễn giải: Thành viên có thể có nhiều câu trả lời, nhưng câu trả lời chỉ có thể do duy nhất một thành viên trả lời

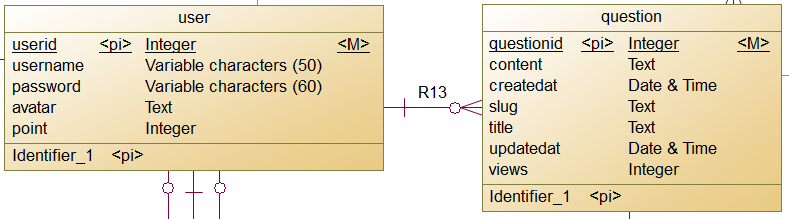
3.7.2.12 Mối quan hệ câu hỏi – câu trả lời



Lớp tham gia: Câu hỏi (question) và câu trả lời (answer)

Diễn giải: Câu hỏi có thể có nhiều câu trả lời, nhưng mỗi câu trả lời chỉ có thể thuộc duy nhất một câu hỏi

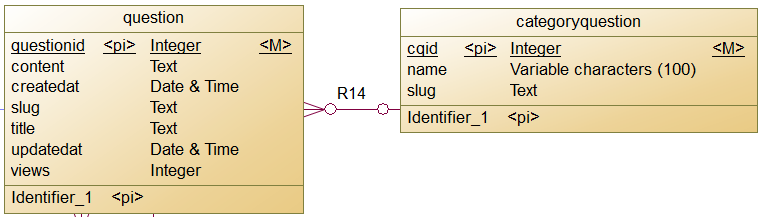
3.7.2.8 Mối quan hệ thành viên – câu hỏi



Lớp tham gia: Thành viên (user) và câu hỏi (question)

Diễn giải: Thành viên có thể đăng nhiều câu hỏi, nhưng câu hỏi chỉ có thể do duy nhất một thành viên đăng

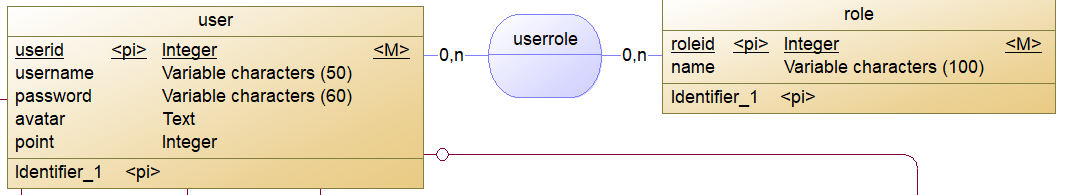
3.7.2.11 Mối quan hệ câu hỏi – loại câu hỏi



Lớp tham gia: Câu hỏi (question) và loại câu hỏi (categoryquestion)

Diễn giải: Câu hỏi chỉ có thể thuộc một loại câu hỏi, nhưng mỗi loại câu hỏi có thể chứa nhiều câu hỏi

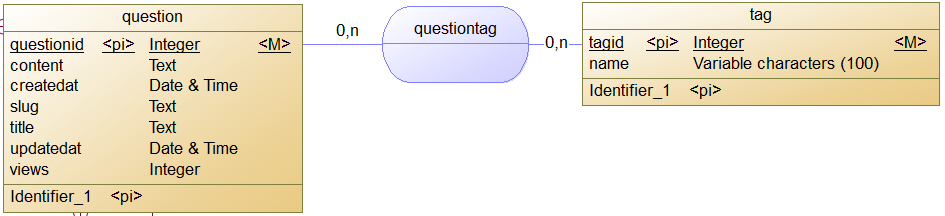
3.7.2.8 Mối quan hệ thành viên – quyền



Lớp tham gia: Thành viên (user) và quyền (role)

Diễn giải: Thành viên có thể có nhiều quyền, mỗi quyền cũng có thể thuộc vào nhiều thành viên

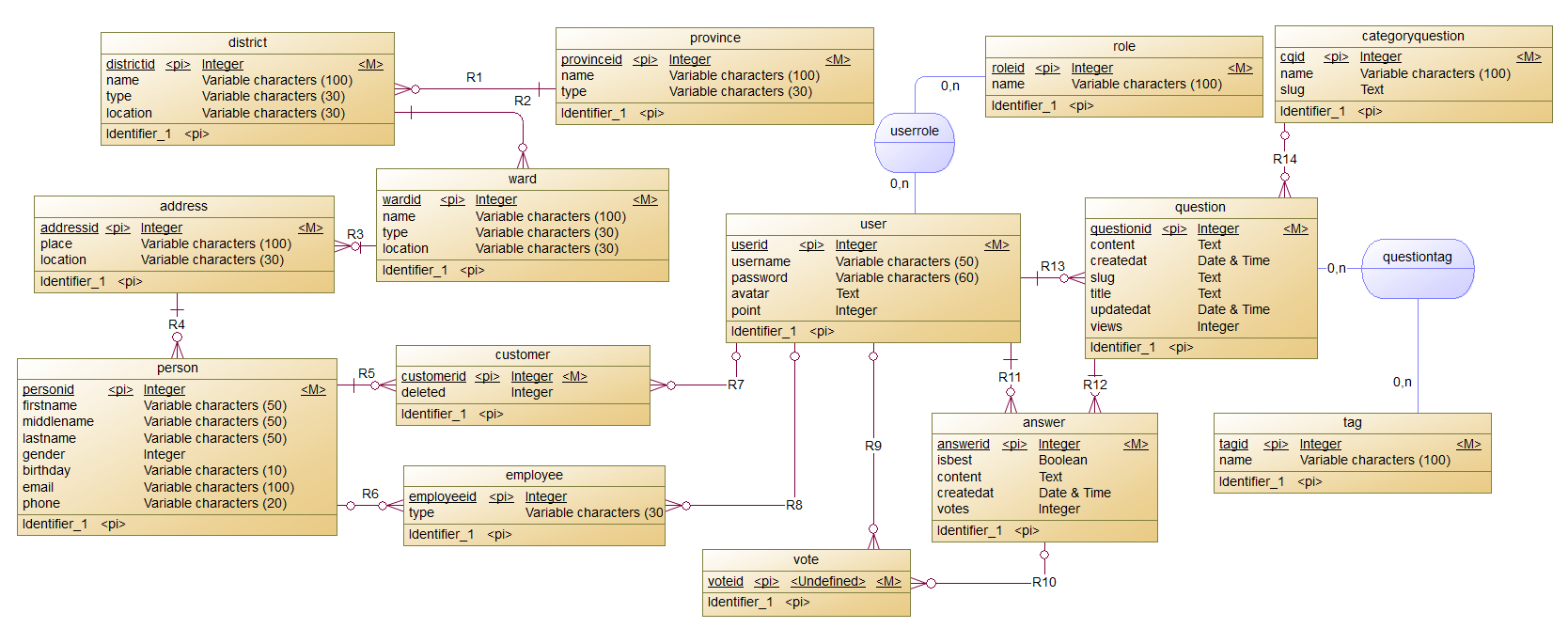
3.7.2.11 Mối quan hệ câu hỏi – thẻ câu hỏi



Lớp tham gia: Câu hỏi (question) và thẻ câu hỏi (tag)

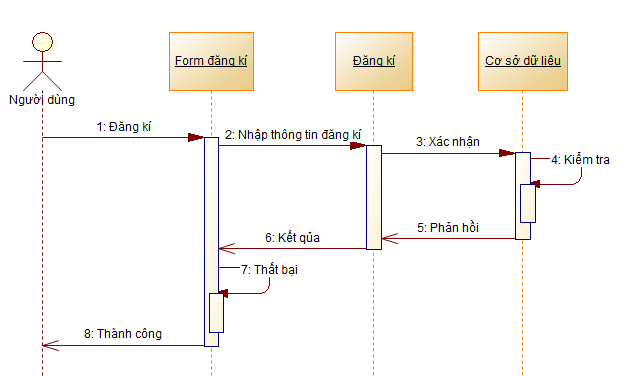
Diễn giải: Câu hỏi có thể có chứa nhiều thẻ liên quan, và mỗi thẻ cũng có thể thuộc nhiều câu hỏi

### 3.7.1 Mô hình ER

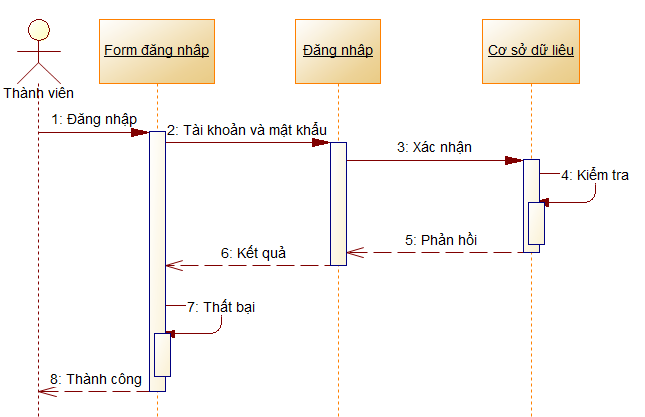


## 3.4 MÔ HÌNH TUẦN TỰ

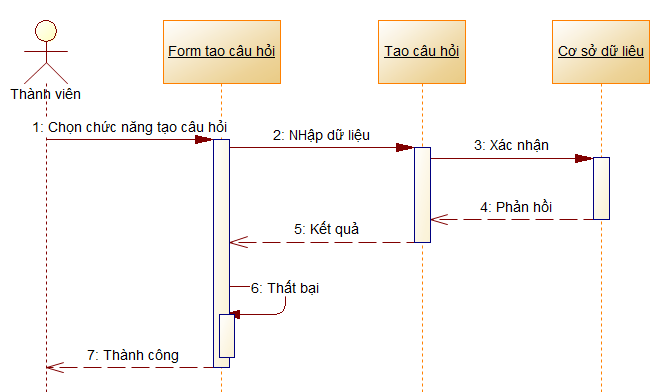
### 3.4.1 Đăng kí:



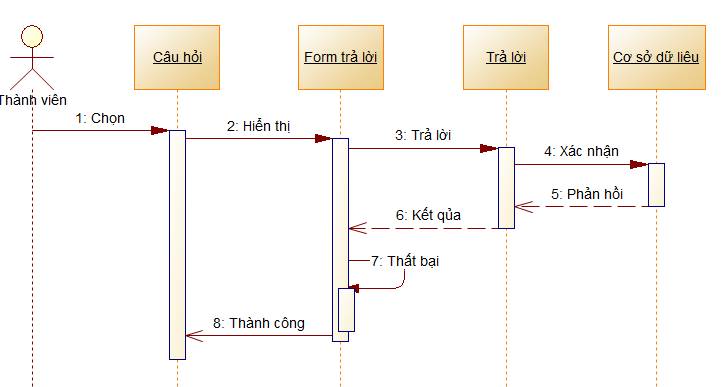
### 3.4.2 Đăng nhập:



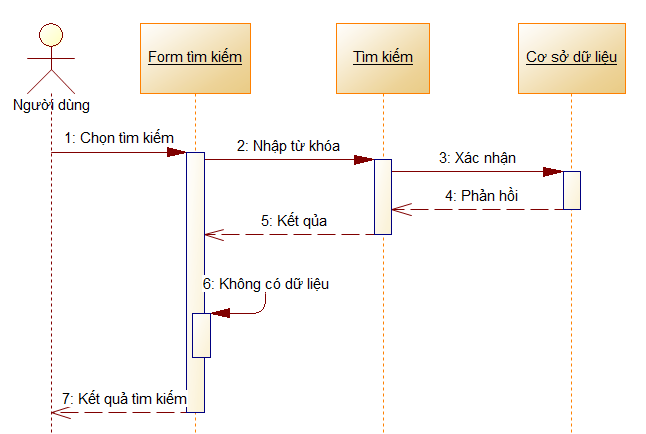
### 3.4.3 Tạo câu hỏi



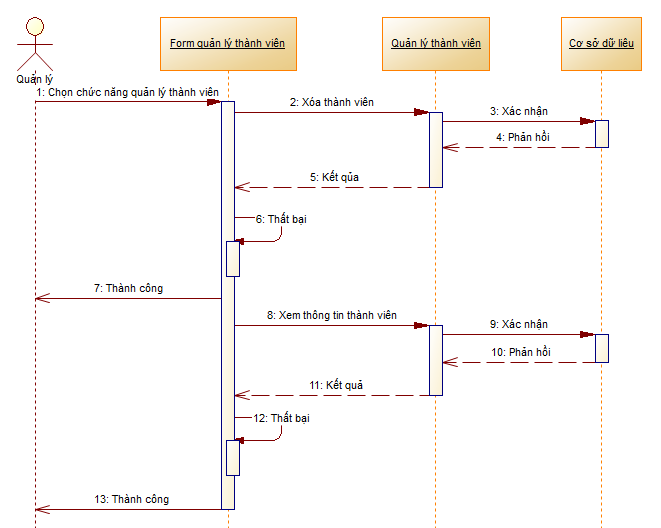
### 3.4.4 Trả lời câu hỏi



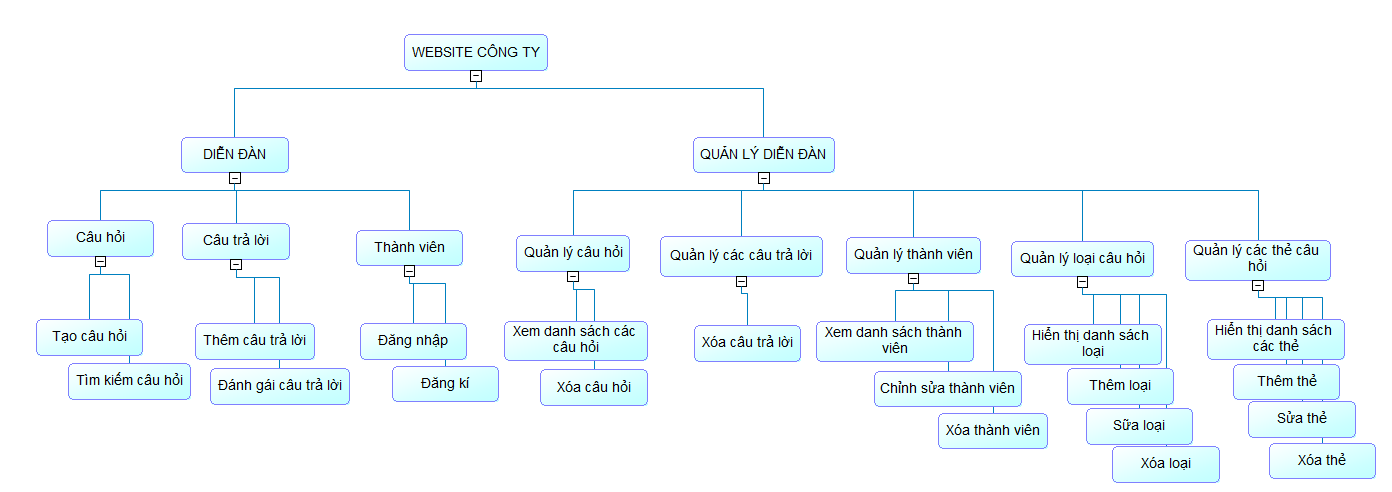
### 3.4.5 Tìm kiếm



### 3.4.6 Quản lý thành viên



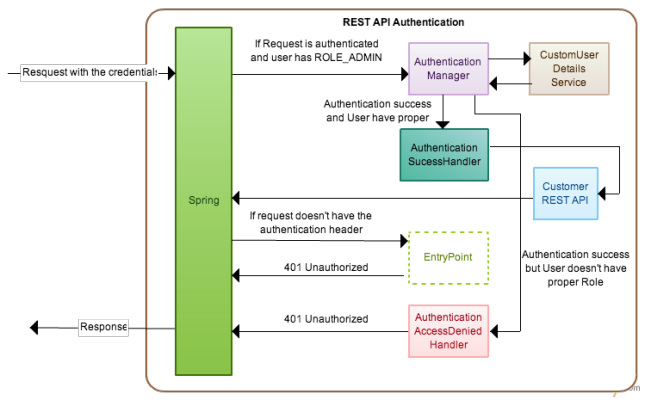
## 3.5 SƠ ĐỒ PHÂN RÃ CHỨC NĂNG



## 3.6 CƠ CHẾ BẢO MẬT ỨNG DỤNG

Việc bảo mật ứng dụng sẽ tập trung vào việc bảo vệ các RESTful API bởi vì tất cả các thao tác nghiệp vụ đều được xử lý thông qua các API này.

Tôi sẽ sử dụng module Spring Security để xây dựng cơ chế bảo vệ các RESTful API. Hình 3.9 sẽ cho thấy chi tiết cơ chế bảo mật của ứng dụng.

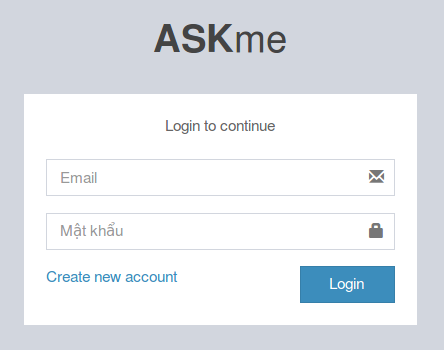


*Hình : Cơ chế bảo mật ứng dụng*

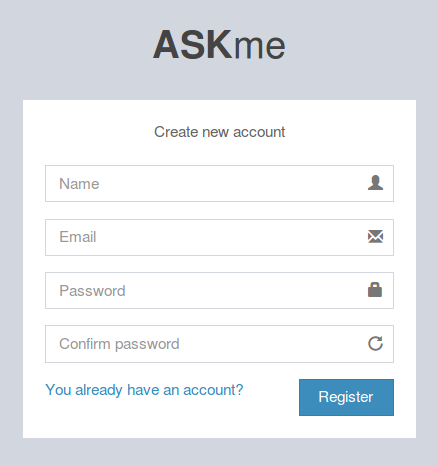
## 3.7 GIAO DIỆN VÀ CHỨC NĂNG

### 3.7.1 Một số giao diện chính

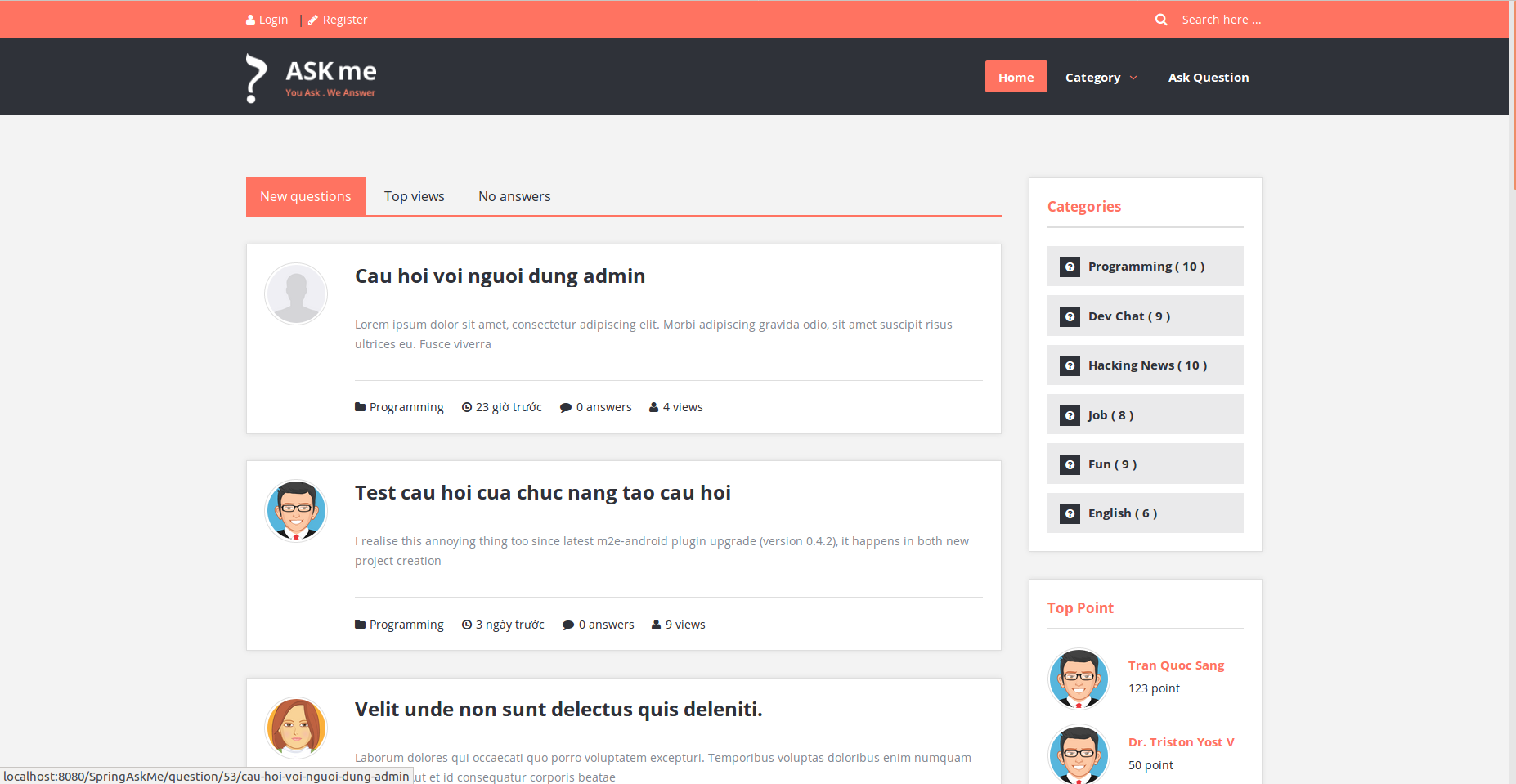
Dưới đây là một số giao diện và chức năng chính của ứng dụng.



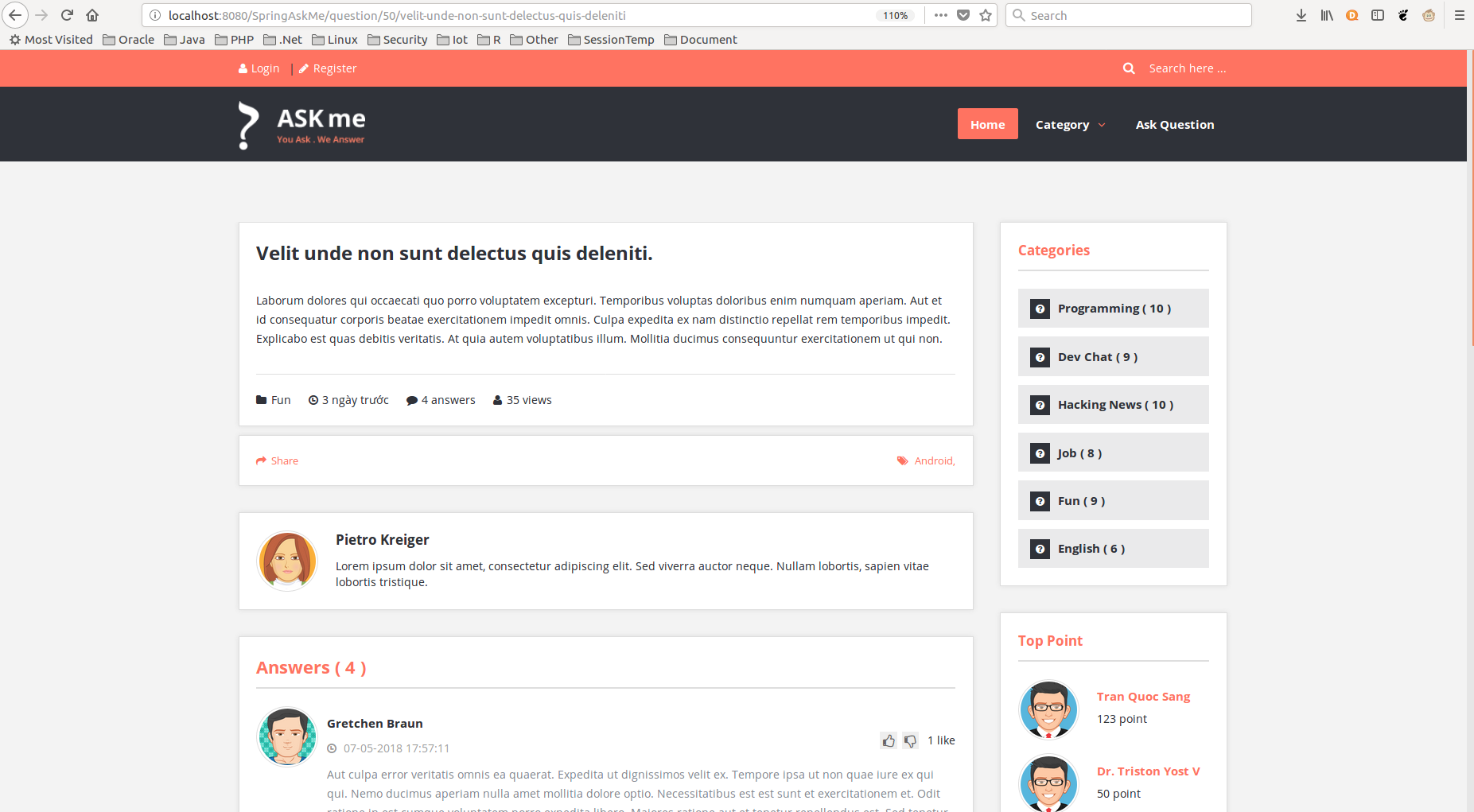
Hình 3.14: Giao diện đăng nhập



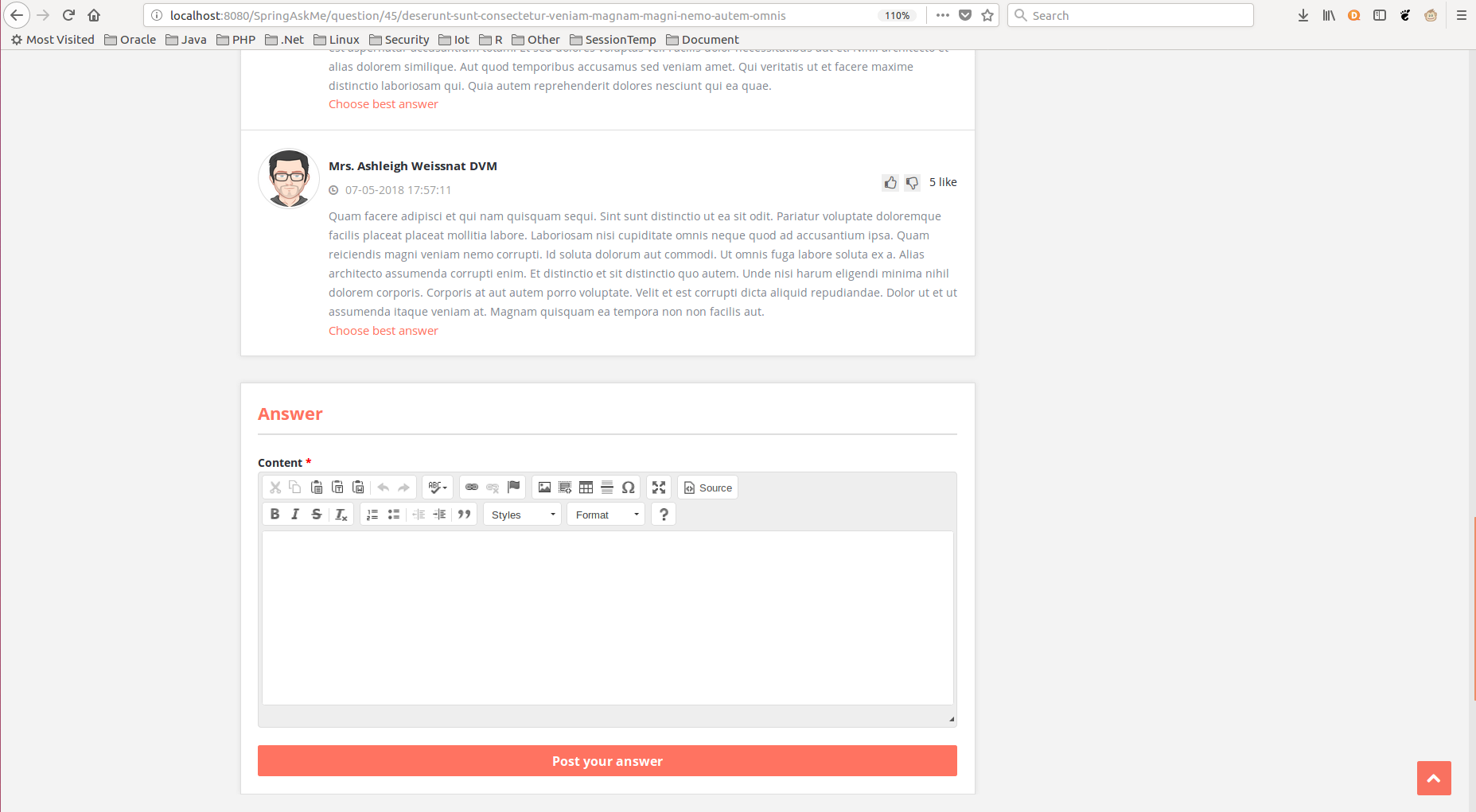
##### Hình 3.13: Giao diện đăng ký người dùng



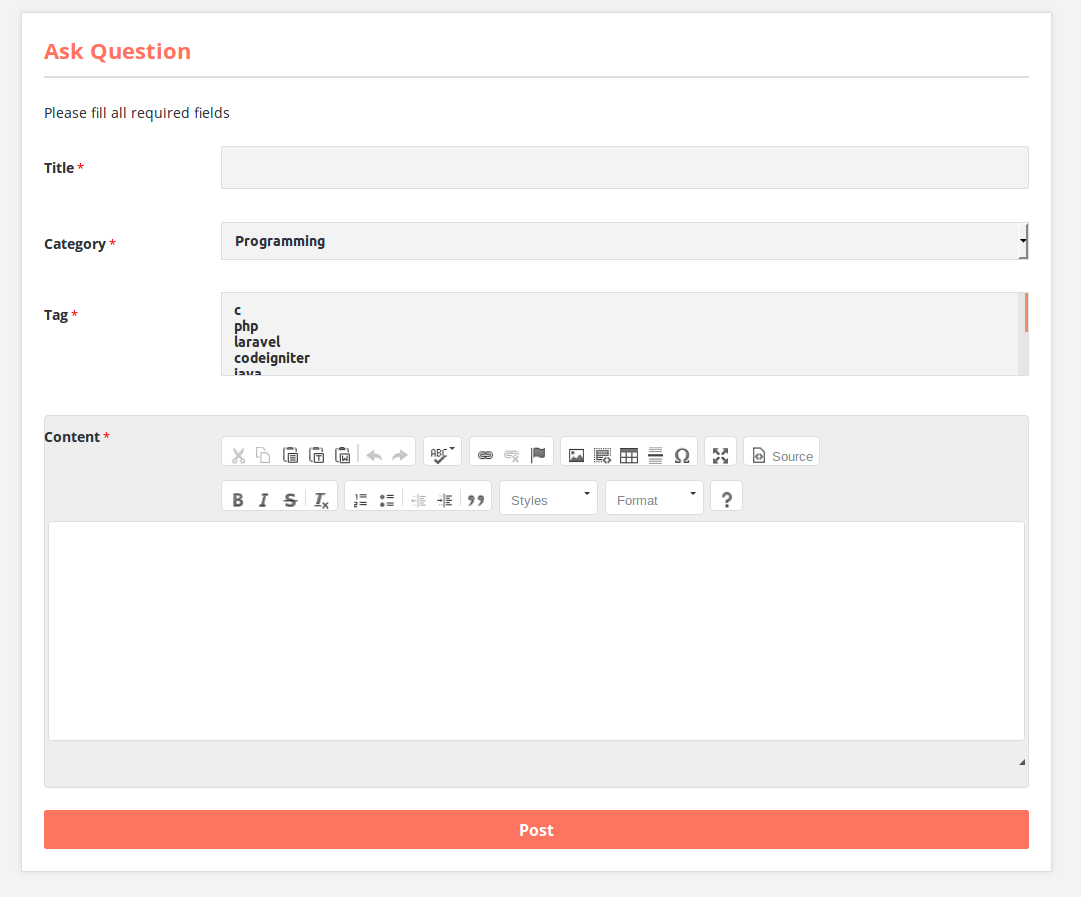
*Hình 3.17: Giao diện trang chủ*



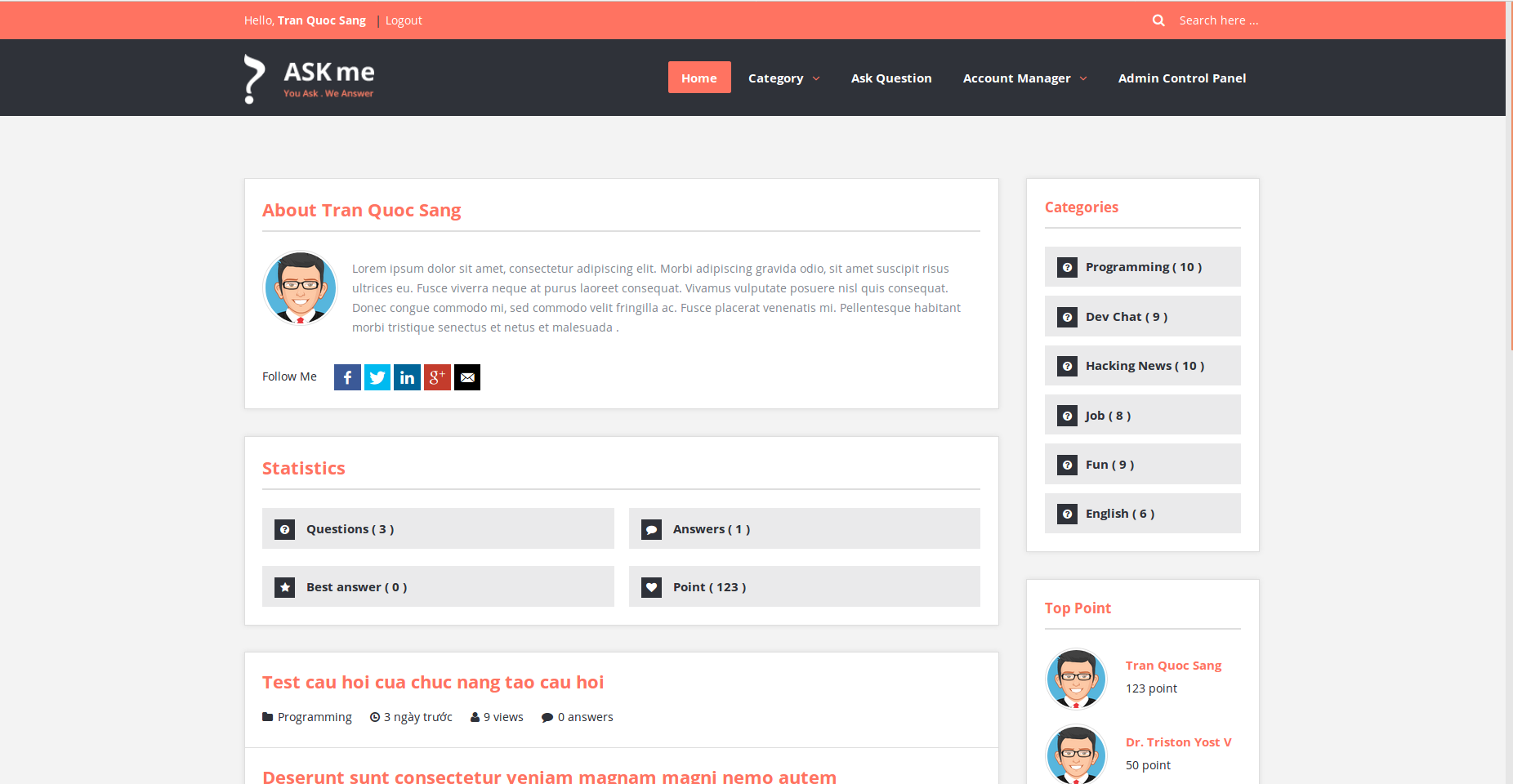
##### Hình 3.15: Giao diện chi tiết câu hỏi



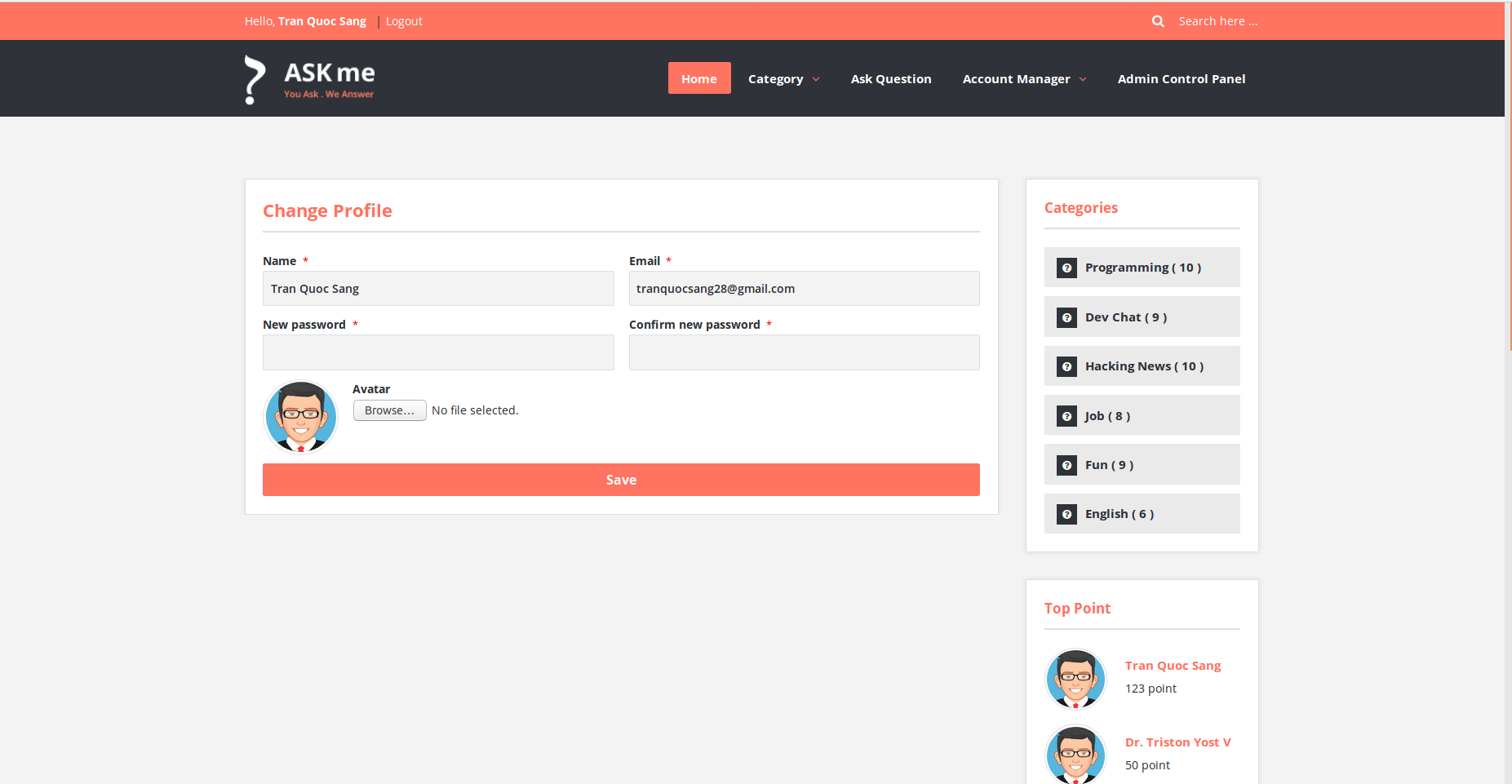
Hình 3.16: Giao diện tạo câu trả lời



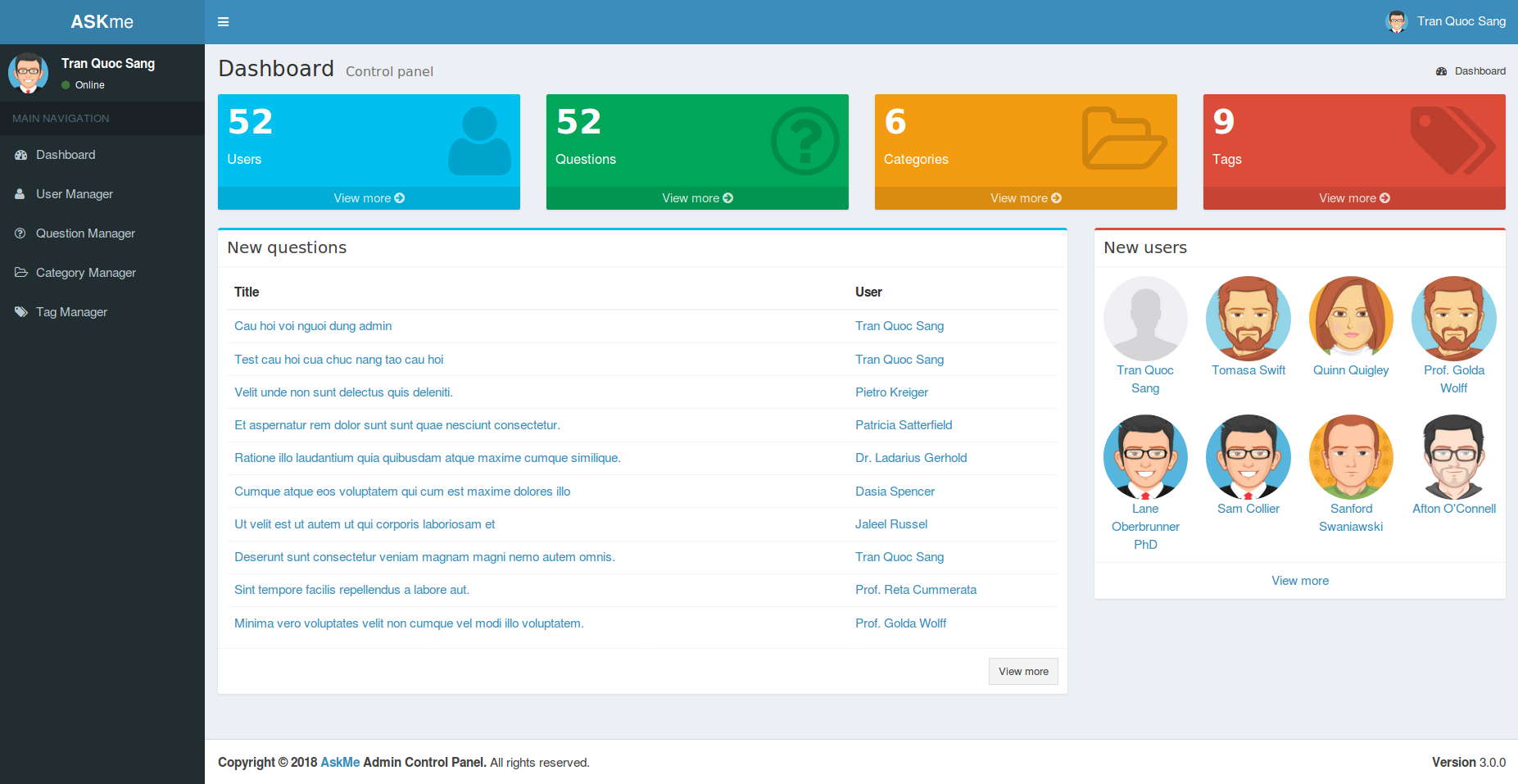
*Hình 3.19: Giao diện đăng câu hỏi*



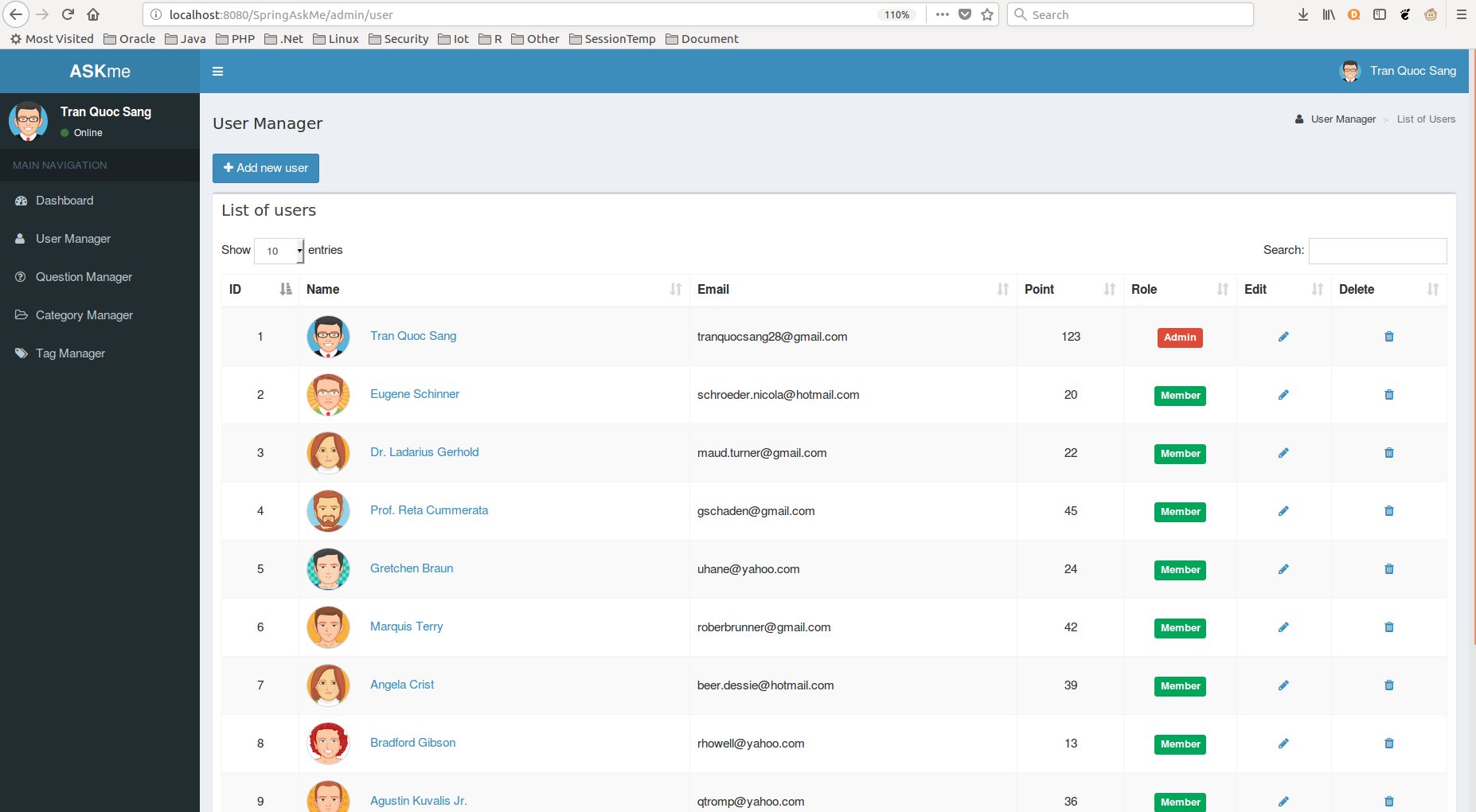
*Hình 3.18: Giao diện trang cá nhân*



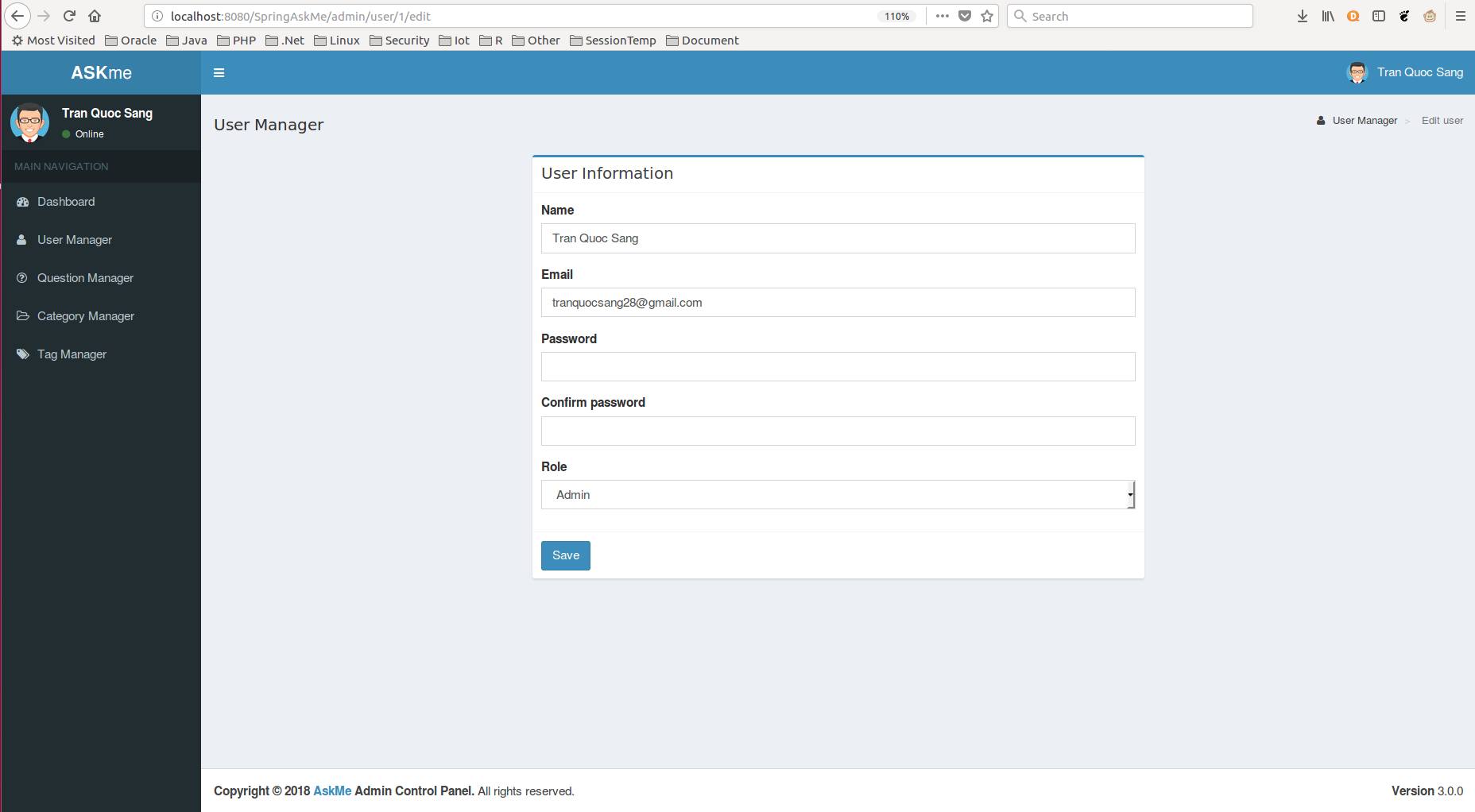
*Hình 3.20: Giao diện sửa thông tin user*



*Hình 3.22: Giao diện thống kê các thông tin*



*Hình 3.21: Giao diện quản lý user*



###### Hình 3.24: Giao diện sửa thông tin user của admin

### 3.7.2 Một số chức năng nổi bật của ứng dụng

# Chương 4 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 4.1 KẾT QUẢ ĐẠT ĐƯỢC

## 4.2 HẠN CHẾ

## 4.3 HƯỚNG PHÁT TRIỂN

# TÀI LIỆU THAM KHẢO

# PHỤ LỤC 1: HƯỚNG DẪN TRIỂN KHAI ỨNG DỤNG