KHOA KỸ THUẬT VÀ CÔNG NGHỆ

BỘ MÔN CÔNG NGHỆ THÔNG TIN

THỰC TẬP ĐỒ ÁN CHUYÊN NGÀNH
HỌC KỲ I, NĂM HỌC 2023 - 2024
NGHIÊN CỨU RESTFUL API VỚI JAVA SPRINGBOOT ĐỂ XÂY DỰNG MODULE BACKEND CHO ỨNG DỤNG QUẢN LÝ QUÁN CAFE

Giáo viên hướng dẫn:

TS. Nguyễn Bảo Ân

Sinh viên thực hiện:

Họ và tên - MSSV:

Lâm Ngọc Tài – 110120152

Lớp: DA20TTB

Trà Vinh, tháng 12 năm 2023

KHOA KỸ THUẬT VÀ CÔNG NGHỆ

BỘ MÔN CÔNG NGHỆ THÔNG TIN

THỰC TẬP ĐỒ ÁN CHUYÊN NGÀNH
HỌC KỲ I, NĂM HỌC 2023 - 2024
NGHIÊN CỨU RESTFUL API VỚI JAVA SPRINGBOOT ĐỂ XÂY DỰNG MODULE BACKEND CHO ỨNG DỤNG QUẢN LÝ QUÁN CAFE

Giáo viên hướng dẫn:

TS. Nguyễn Bảo Ân

Sinh viên thực hiện:

Họ và tên - MSSV:

Lâm Ngọc Tài – 110120152

Lớp: DA20TTB

Trà Vinh, tháng 12 năm 2023

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

LỜI CẢM ƠN

Kính thưa quý thầy cô

Đầu tiên, em xin bày tỏ lòng tri ân sâu sắc đến các thầy cô giáo của Trường Đại học Trà Vinh, đặc biệt là quý thầy cô thuộc Khoa Kỹ thuật & Công nghệ, bộ môn Công nghệ Thông tin, đã tận tình hỗ trợ và tạo mọi điều kiện thuận lợi để con có thể hoàn thành bài đồ án chuyên ngành này một cách tốt nhất.

Em cũng xin gửi lời biết ơn chân thành đến thầy Nguyễn Bảo Ân - Giảng viên Khoa Kỹ thuật & Công nghệ, người đã hết lòng hướng dẫn, truyền đạt kiến thức và kinh nghiệm quý báu, giúp con hoàn thiện đồ án này.

Em nhận thức được rằng bản thân còn nhiều hạn chế và đồ án vẫn còn những sai sót nhất định do kinh nghiệm còn hạn chế. Em kính mong nhận được sự thông cảm và những lời góp ý quý báu từ quý thầy cô để em có thể rút kinh nghiệm và tiếp tục hoàn thiện mình trong những nghiên cứu sau này.

Cuối cùng, em xin kính chúc quý thầy cô sức khỏe dồi dào, hạnh phúc và thành công trong cuộc sống.

Với tất cả sự kính trọng và biết ơn

Ký tên

MỤC LỤC

DANH MỤC VIẾT TẮC

Từ viết tắc	Nghĩa của từ
API	Application Programming Interface
RESTful	Representational State Transfer - ful
SQL	Structured Query Language
HTTP	HyperText Transfer Protocol
XML	eXtensible Markup Language
JWT	JSON Web Token
JSON	JavaScript Object Notation

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

TÓM TẮT ĐỒ ÁN

Vấn đề nghiên cứu

Hiệu quả quản lý thấp: Quán cafe truyền thống thường phụ thuộc vào quản lý thủ công, làm cho việc theo dõi đơn hàng, quản lý kho, và tài chính trở nên phức tạp và dễ phát sinh sai sót.

Quản lý tài chính: Việc theo dõi doanh thu, chi phí, và quản lý dòng tiền thường không được thực hiện một cách bài bản và minh bạch.

Thách thức trong việc phân tích dữ liệu: Việc thiếu hệ thống dữ liệu tổng hợp khiến cho việc phân tích xu hướng tiêu dùng và đưa ra quyết định kinh doanh trở nên khó khăn.

Khách hàng chờ đợi lâu: Quy trình phục vụ truyền thống không hiệu quả, gây mất thời gian cho khách hàng và nhân viên.

Cách giải quyết vấn đề

Phát triển hệ thống quản lý quán cafe dựa trên nền tảng web và mobile: Sử dụng Java Spring Boot để tạo RESTful API, tạo cơ sở cho ứng dụng web và mobile giúp tự động hóa các quy trình quản lý.

Quản lý tài chính minh bạch: Tích hợp các công cụ để theo dõi doanh thu, chi phí và quản lý dòng tiền một cách minh bạch và tự động.

Phân tích dữ liệu thông minh: Xây dựng chức năng phân tích dữ liệu để nhận diện xu hướng tiêu dùng, hỗ trợ quyết định kinh doanh.

Cải thiện trải nghiệm khách hàng: Tích hợp hệ thống đặt hàng và thanh toán trực tuyến, giảm thời gian chờ đợi của khách hàng.

Một số kết quả đạt được:

Tăng cường hiệu quả quản lý: Hệ thống giúp quản lý quán cafe một cách hiệu quả hơn, giảm thiểu sai sót và tối ưu hóa quy trình làm việc.

Quản lý tài chính tốt hơn: Tăng cường khả năng theo dõi và quản lý tài chính, giúp quán cafe có cái nhìn rõ ràng hơn về tình hình tài chính.

Cải thiện quyết định kinh doanh: Cung cấp dữ liệu và phân tích giúp nhà quản lý đưa ra quyết định chính xác và kịp thời.

Tăng sự hài lòng của khách hàng: Cải thiện trải nghiệm khách hàng thông qua việc giảm thời gian chờ đợi và tăng cường tương tác khách hàng.

MỞ ĐẦU

Lí do chọn đề tài:

Một trong những giải pháp công nghệ tiên tiến và phù hợp với nhu cầu thực tế của quản lý quán cafe là Java Spring Boot. Đây là một công nghệ được sử dụng rộng rãi để xây dựng các ứng dụng doanh nghiệp hiệu năng cao, bảo mật tốt, và dễ dàng mở rộng. Với Java Spring Boot, quản lý quán cafe có thể tạo ra một hệ thống quản lý đơn hàng, tồn kho, nhân viên, và khách hàng một cách hiệu quả và tiện lợi. Hệ thống này sẽ giúp quản lý quán cafe theo dõi được tình hình kinh doanh, tối ưu hóa chi phí, và nâng cao chất lượng dịch vụ. Đồng thời, việc sử dụng Java Spring Boot cũng giúp quản lý quán cafe bắt kịp xu hướng số hóa đang diễn ra trong ngành dịch vụ, tạo ra sự khác biệt và cạnh tranh trên thị trường.

Mục tiêu nghiêm cứu:

Xây dựng các RESTful API sử dụng Java Spring Boot để quản lý các hoạt động chính của quán cafe. Tích hợp các tính năng như quản lý đơn hàng, quản lý tồn kho, quản lý nhân viên, và tương tác với khách hàng. Đảm bảo an toàn thông tin và hiệu suất tốt trong quá trình vận hành.

Đối tượng nghiêm cứu:

RESTful API: Các nguyên lý và thực hành xây dựng API theo mô hình REST.

Java Spring Boot: Khám phá các tính năng của Spring Boot như Spring Data JPA, Spring Security.

Ứng dụng quản lý quán cafe: Nghiên cứu về các yêu cầu và quy trình nghiệp vụ trong quản lý quán cafe.

Phạm vị nghiêm cứu:

Tập trung vào việc xây dựng và tích hợp API, xây dựng API theo kiến trúc Restful API

Sử dụng môi trường phát triển dựa trên Java và các công cụ như Intellij IDEA, Mysql Workbench, Post Man.

Các nghiên cứu và phát triển sẽ được thực hiện trong khuôn khổ thời gian và nguồn lực đã xác định trước.

CHƯƠNG 1: TỔNG QUAN

1.1 Mô tả

Trong dự án này, Ta hướng đến việc phát triển một hệ thống quản lý cho một quán cafe vừa và nhỏ, nơi mà việc tổ chức và quản lý hoạt động hàng ngày đóng vai trò quan trọng. Hệ thống này sẽ tập trung vào ba khía cạnh chính: quản lý sản phẩm, tương tác với khách hàng, và quản lý nhân viên.

Đối với sản phẩm, mỗi món ăn và đồ uống được chế biến từ nguyên liệu mua từ chợ sẽ được quản lý một cách cẩn thận, từ thông tin cơ bản như tên và giá cả đến các chi tiết như loại sản phẩm. Quy trình sẽ giúp cho chủ quán ca phê sẽ dễ dàng tạo ra chương tình khuyến mãi.

Tương tác với khách hàng là yếu tố then chốt trong việc cung cấp trải nghiệm tốt tại quán cafe. Từ việc chọn bàn đến gọi món, mỗi tương tác đều được thiết kế để đảm bảo sự thuận tiện và hài lòng cho khách hàng, qua đó góp phần xây dựng mối quan hệ lâu dài và đáng giá với họ.

Quản lý nhân viên cũng là một phần không thể thiếu. Thông tin từ cơ bản đến chi tiết của mỗi nhân viên, từ tên, địa chỉ email, ngày sinh, đến địa chỉ và giới tính, sẽ được theo dõi và quản lý chặt chẽ. Mục đích không chỉ là để quản lý nhân sự hiệu quả mà còn để tạo điều kiện làm việc tốt nhất cho đội ngũ này, qua đó cải thiện chất lượng dịch vụ và nâng cao hiệu suất làm việc chung của quán.

Khi khách hàng đến quán cà phê, nhân viên sẽ xem bàn nào còn trốn sẽ hướng dẫn khác ngồi. Nhân viên sẽ ghi các đồ uống khách yêu cầu thông tin gồm tên đồ uống số lượng. Yêu cầu sẽ được chuyên để quầy pha chế.

Khách hàng yêu cầu thanh toán sẽ được lập hoá đơn ghi nhận việc mua hàng. Thông tin trên hoá đơn gồm có mã hoá đơn, ngày mua, các đồ uống, trạng thái, ghi chú, thành tiền, tên nhân viên lập hoá đơn.

1.2 Phạm vi đề tài

Đề tài sẽ tập trung trình bày kết quả nghiên cứu của tôi về các nội dung sau: Spring framework, MySQL, 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 sẽ tập trung tìm hiểu và trình bày 2 module: Spring boot và Spring Security.

Sau khi tìm hiểu tôi sẽ vận dụng kết quả tìm hiểu được vào việc xây dựng một module được kiểm thử Postman.

1.3 Hướng giải quyết

❖ Khám Phá và Phân Tích:

Đầu tiên, tiến hành nghiên cứu về Spring framework để hiểu rõ kiến trúc cũng như cách thức hoạt động của nó. Điều này bao gồm việc tìm hiểu các khái niệm cơ bản, cấu hình, và quy trình vận hành.

Phân tích và đánh giá MySQL, một hệ quản trị cơ sở dữ liệu quan hệ, để hiểu cách nó có thể được tích hợp và sử dụng hiệu quả trong ứng dụng.

Nghiên cứu Maven như một công cụ quản lý và tự động hóa dự án, đặc biệt là cách quản lý phụ thuộc và xây dựng dự án.

❖ Tìm hiểu Spring Boot và Spring Security:

Tập trung nghiên cứu kỹ lưỡng về Spring Boot, một module của Spring framework, để tạo ra các ứng dụng độc lập với cấu hình mặc định tối ưu.

Nguyên cứu Spring Security, để bảo đảm tính năng bảo mật cho ứng dụng thông qua xác thực và ủy quyền người dùng.

❖ Ứng Dụng Thực Tiễn:

Áp dụng kiến thức thu được để thiết kế và xây dựng một module backend cho ứng dụng Quản lý Quán Cafe, sử dụng Spring Boot để cung cấp cơ sở vững chắc và Spring Security để đảm bảo tính bảo mật.

Tích hợp MySQL như là hệ quản trị cơ sở dữ liệu để quản lý dữ liệu liên quan đến đơn hàng, tồn kho, và thông tin người dùng.

❖ Kiểm Thử và Đánh Giá:

Tiến hành kiểm thử chức năng của module sử dụng Postman, một công cụ kiểm thử API, để đánh giá chức năng và tìm ra các vấn đề tiềm ẩn.

Tinh chỉnh và cải tiến module dựa trên kết quả kiểm thử để đảm bảo ứng dụng hoạt động hiệu quả và ổn định.

Các công cụ hỗ trợ nghiên cứu:

1. Microsoft Word: dùng để viết tài liệu báo cáo
2. Draw.io: dùng để vẽ các biểu đồ
3. IntelliJ IDEA: dùng để code back end
4. Postman: kiểm tra các API đã viết ra.

5. MySQL Workbench: dùng để lưu trữ cơ sở dữ liệu

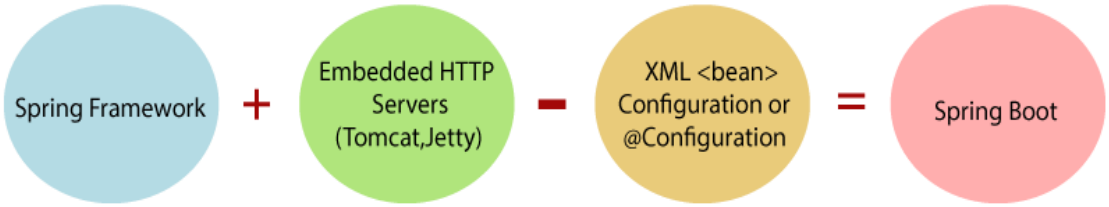
Sau quá trình nghiêm cứu và xây dựng đã xây dựng thành công bộ API đủ phục vụ cho người lập trình viên xây dựng ứng dụng trên các nền tản khác nhau.

CHƯƠNG 2: NGHIÊN CỨU VỀ LÝ THUYẾT

2.1 Tìm hiểu về Java Spring Boot

2.1.1 Java Spring Boot là gì?

Java Spring Boot là một dự án phụ của Spring Framework, được thiết kế để đơn giản hóa quá trình cài đặt và phát triển các ứng dụng dựa trên Spring. Đây là một framework mạnh mẽ, linh hoạt, được sử dụng rộng rãi trong việc phát triển ứng dụng web và microservices trong môi trường Java [1].



Hình 1. Mô tả quá trình tạo nên Spring Boot từ Spring Framework.

Một số điểm nổi bậc của Java Spring Boot:

- Spring Boot tự động cấu hình ứng dụng của dựa trên các thư viện có trong classpath. Điều này giúp giảm thiểu công việc cấu hình và bootstrap ứng dụng.

- Nó tạo ra các ứng dụng có thể chạy độc lập, với embedded Tomcat, Jetty, hoặc các server web khác, không cần một servlet container riêng biệt.

- Các ứng dụng truyền thống của Spring Framework thường yêu cầu cấu hình XML phức tạp hoặc sử dụng annotation @Configuration để định nghĩa beans và các phụ thuộc của chúng. Spring Boot giảm bớt nhu cầu này bằng cách cung cấp tự động cấu hình và các cài đặt mặc định hợp lý [2].

Spring Boot được tạo nên bằng cách kết hợp Spring Framework với các server HTTP tích hợp và giảm bớt đi sự phức tạp của cấu hình XML hoặc annotation. Điều này dẫn đến một nền tảng dễ dàng thiết lập và sẵn sàng chạy trong môi trường riêng của nó, lý tưởng cho các microservices và ứng dụng web [1].

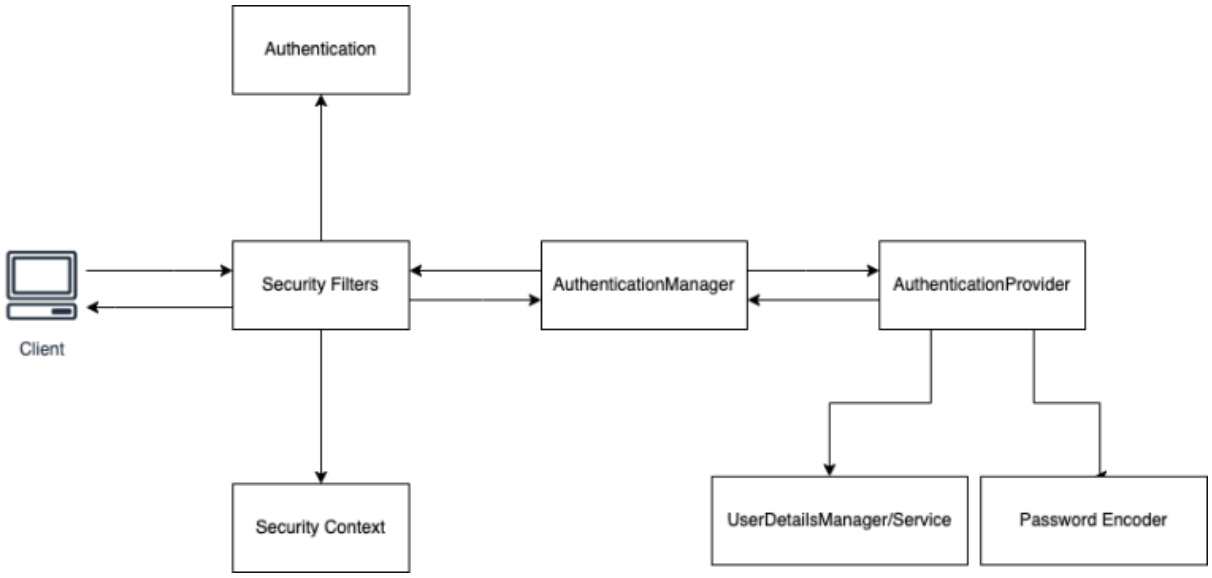
2.1.2 Tổng quan Spring Security

2.1.1.1 Spring Security là gì?

Spring Security là một framework bảo mật chuyên sâu cho các ứng dụng dựa trên Spring Framework. Nó nhằm mục đích cung cấp cả xác thực và phân quyền cũng như các cơ chế bảo vệ chống lại các tấn công thông dụng trong ứng dụng web. Spring Security đặc biệt chú trọng vào việc cung cấp một giải pháp bảo mật dễ dàng tích hợp nhưng cũng đủ mạnh mẽ để đáp ứng được các yêu cầu bảo mật phức tạp [3] [4].

2.1.1.2 Cơ chế hoạt động Spring Security

Spring Security hoạt động theo mô hình client-server. Khi một client gửi một request đến server, server sẽ xác thực người dùng và phân quyền để đảm bảo rằng người dùng chỉ có thể truy cập vào những tài nguyên mà họ được phép truy cập [5].



Hình 2. Cơ chế Filter của Spring Security.

Spring Security là một framework bảo mật mạnh mẽ dành cho các ứng dụng Java, đặc biệt là những ứng dụng được xây dựng trên nền tảng Spring. Nó cung cấp một cơ chế bảo mật toàn diện thông qua một chuỗi các bộ lọc (filters) được xác định để kiểm soát và quản lý quyền truy cập vào ứng dụng. Dưới đây là quy trình hoạt động chi tiết của Spring Security khi xử lý xác thực thông tin người:

- Spring Security Filters

  Khi một yêu cầu HTTP được gửi từ client đến server, chuỗi bộ lọc của Spring Security bắt đầu xử lý yêu cầu đó. Trong số các bộ lọc này, UsernamePasswordAuthenticationFilter đóng vai trò quan trọng trong việc lấy thông tin từ yêu cầu đăng nhập - thường là tên người dùng và mật khẩu - và tạo ra một đối tượng Authentication, thường là UsernamePasswordAuthenticationToken. Bộ lọc

này kế thừa từ AbstractAuthenticationProcessingFilter, một lớp trừu tượng cung cấp cơ sở cho việc xử lý xác thực.

❖ AuthenticationManager và AuthenticationProvider

UsernamePasswordAuthenticationToken được tạo ra sau đó được chuyển đến AuthenticationManager, một giao diện trung tâm trong Spring Security quản lý quá trình xác thực. AuthenticationManager có một phương thức authenticate() được gọi để xử lý đối tượng Authentication. Trong thực tiễn, ProviderManager là một implementation của AuthenticationManager, quản lý một danh sách các AuthenticationProvider.

Mỗi AuthenticationProvider có một phương thức authenticate() riêng biệt và chịu trách nhiệm xác thực thông tin người dùng. Nếu một AuthenticationProvider không thể xác thực thông tin người dùng, quá trình xác thực sẽ tiếp tục với AuthenticationProvider tiếp theo trong danh sách, nếu có [3].

❖ UserDetailsService và PasswordEncoder

DaoAuthenticationProvider, một trong những AuthenticationProvider phổ biến, sử dụng UserDetailsService để tải thông tin chi tiết của người dùng dựa trên tên người dùng. UserDetailsService có một phương thức quan trọng là loadUserByUsername() để tìm kiếm và trả về một đối tượng UserDetails chứa thông tin như tên người dùng, mật khẩu và các quyền hạn (authorities).

PasswordEncoder là một thành phần quan trọng khác, nó chịu trách nhiệm mã hóa mật khẩu khi lưu trữ và so sánh mật khẩu đã mã hóa với mật khẩu được cung cấp bởi người dùng khi đăng nhập. Việc mã hóa này giúp bảo vệ thông tin mật khẩu người dùng ngay cả khi có sự cố dữ liệu bị rò rỉ.

❖ Security Context

Sau khi xác thực thành công, thông tin người dùng và các quyền hạn được lưu trong SecurityContextHolder, trong một đối tượng gọi là SecurityContext. Điều này cho phép thông tin người dùng dễ dàng được truy cập trong suốt quá trình xử lý yêu cầu mà không cần phải xác thực lại. SecurityContext tồn tại trong phạm vi của một yêu cầu và thường được liên kết với một phiên người dùng [5].

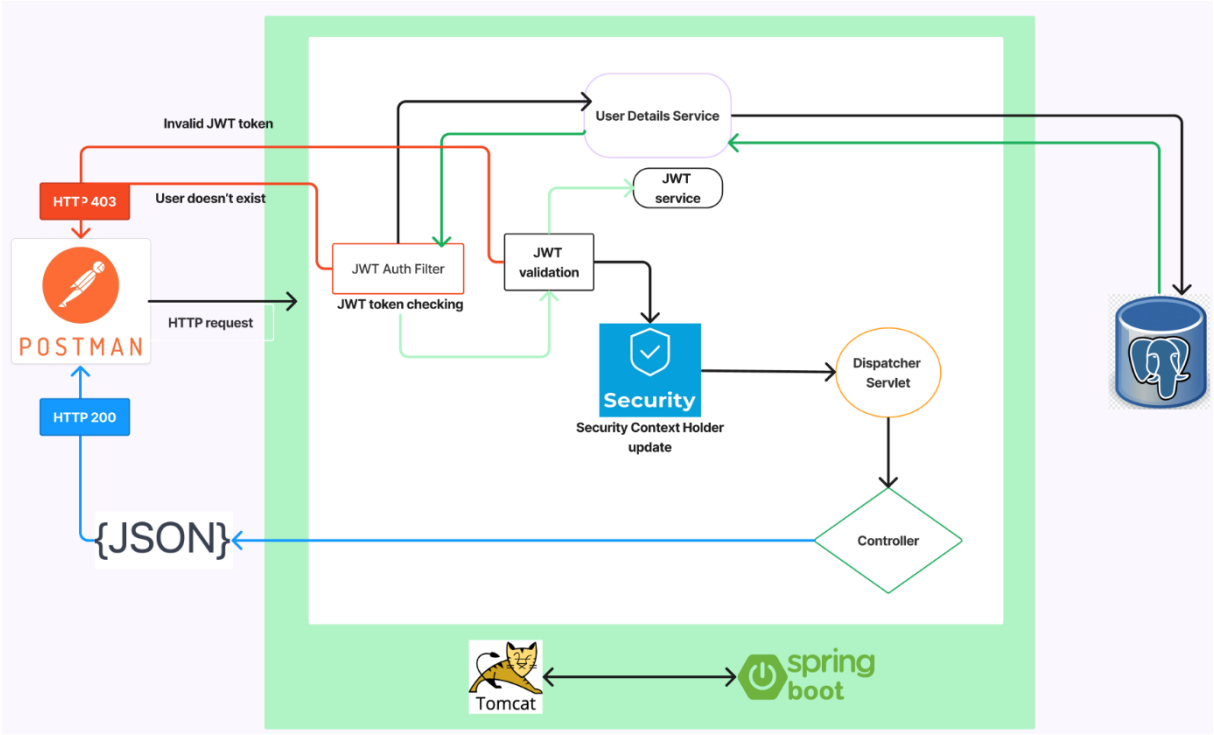
Spring Security xử lý xác thực người dùng thông qua một quy trình nhiều lớp, từ việc lấy thông tin người dùng từ yêu cầu đến xác định và xác thực thông tin đó, rồi cuối cùng lưu trữ trong môi trường an toàn để sử dụng sau này. Quy trình này không chỉ đảm bảo

rằng chỉ những người dùng hợp lệ mới có thể truy cập vào ứng dụng, mà còn bảo vệ thông tin người dùng và ứng dụng khỏi các mối đe dọa bảo mật.

2.2 Json Web Token (JWT)

2.2.1 Tìm hiểu về JWT

JSON Web Token (JWT) là một tiêu chuẩn mở (RFC 7519) được sử dụng để truyền thông tin an toàn giữa hai bên dưới dạng một đối tượng JSON. Thông tin này có thể được xác thực và đáng tin cậy vì nó được ký số điện tử. JWT thường được sử dụng trong ứng dụng web để quản lý xác thực người dùng và trao đổi thông tin [6].



Hình 3. Xác thực người dùng bằng json web token.

Dưới đây là giải thích từng bước của quy trình xác thực:

JWTAuthFilter: Đây là bộ lọc xác thực JWT, chịu trách nhiệm kiểm tra token JWT trong mỗi yêu cầu HTTP đến.

Validate JWT: JWTAuthFilter gọi JwtService để kiểm tra sự hợp lệ của token JWT.

UserDetailsService: Nếu token hợp lệ, JwtService sẽ liên lạc với UserDetailsService để lấy thông tin chi tiết của người dùng.

Security: Sau khi thông tin người dùng được xác nhận, hệ thống bảo mật của Spring (Security) sẽ cập nhật SecurityContextHolder với thông tin xác thực của người dùng.

DispatcherServlet: Khi thông tin xác thực được cập nhật, yêu cầu sẽ được chuyển đến DispatcherServlet, đây là một thành phần trung tâm trong Spring MVC điều hướng yêu cầu đến các Controller phù hợp.

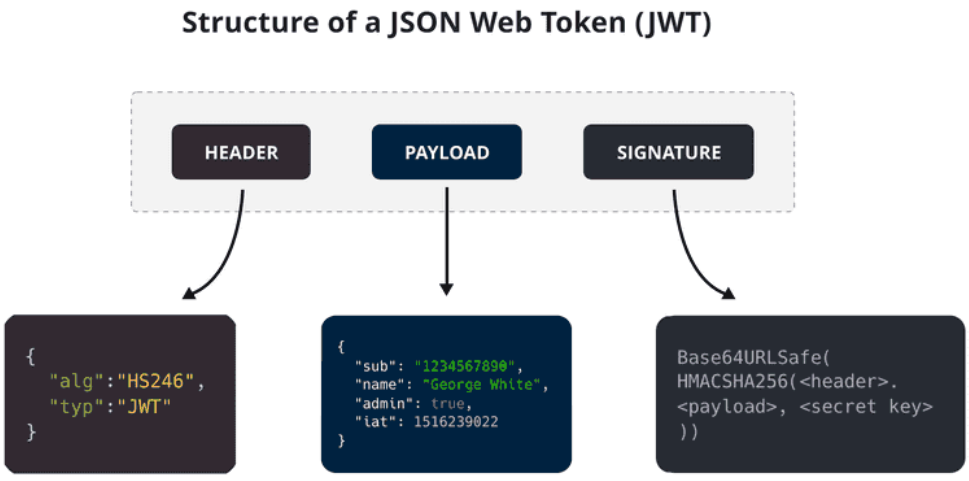
Controller: Controller sau đó xử lý yêu cầu và trả về phản hồi, thường là dạng JSON.

HTTP Response: Phản hồi sẽ được gửi trở lại cho client. Nếu xác thực thành công, client sẽ nhận được mã trạng thái HTTP 200. Nếu token không hợp lệ hoặc thiếu, client sẽ nhận được mã lỗi 403. Nếu thông tin người dùng không tồn tại, hệ thống cũng sẽ trả về mã lỗi 403.

Apache Tomcat: Là máy chủ web và servlet container mà Spring Boot sử dụng, được triển khai dưới dạng một container trong Docker, một nền tảng để triển khai ứng dụng dưới dạng containers, giúp việc triển khai và quản lý ứng dụng trở nên linh hoạt và dễ dàng hơn.

JWT gồm ba phần, tách biệt bởi dấu chấm (.):

- Header (Tiêu đề): Header thường chứa loại token (JWT) và thuật toán mã hóa được sử dụng (như HMAC SHA256 hoặc RSA).
- Payload (Dữ liệu): Payload chứa các tuyên bố (claims), như thông tin người dùng, thời gian hết hạn token, phát hành, v.v.
- Signature (Chữ ký): Để tạo chữ ký, thuật toán mã hóa trong header được sử dụng cùng với base64-url encoded header và payload cùng với một 'secret' chỉ biết bởi người phát hành token [2].



Hình 4. Kiến trúc của JWT

2.2.2 Cơ chế ủy quyền

Ủy quyền JWT (JSON Web Token) là một quy trình quan trọng trong các hệ thống ứng dụng web và di động, cho phép kiểm soát quyền truy cập vào các tài nguyên và dịch vụ dựa trên các thông tin (claims) chứa trong token JWT. Dưới đây là mô tả cách thức JWT được sử dụng trong quá trình ủy quyền:

- Tạo JWT với Claims Ủy Quyền:
    - Khi JWT được tạo sau quá trình xác thực, nó có thể bao gồm các claims liên quan đến ủy quyền. Các claims này thường chứa thông tin về vai trò (role) của người dùng và các quyền (permissions) liên quan.
    - Ví dụ, một JWT có thể chứa thông tin như "role": "admin" hoặc "permissions": ["create", "delete"].
- Gửi JWT trong Mọi Yêu Cầu:
    - Sau khi đăng nhập, người dùng sẽ gửi JWT trong mỗi yêu cầu đến server, thường là thông qua header Authorization.
    - Điều này giúp server xác định danh tính (xác thực) cũng như quyền truy cập (ủy quyền) của người dùng.
- Server Kiểm Tra JWT:
    - Khi nhận yêu cầu, server sẽ xác minh JWT, không chỉ kiểm tra tính hợp lệ của nó mà còn xem xét các claims ủy quyền.
    - Server sau đó xác định xem người dùng có quyền truy cập vào tài nguyên hoặc thực hiện hành động yêu cầu hay không.
- Truy Cập Dựa trên Ủy Quyền:
    - Nếu JWT cho thấy người dùng có quyền thích hợp, yêu cầu sẽ được xử lý. Nếu không, server sẽ từ chối truy cập và có thể trả về một thông báo lỗi như 403 Forbidden.

❖ Ưu điểm của Ủy Quyền JWT

- Trung Tâm và Nhất Quán: Quản lý ủy quyền trở nên dễ dàng hơn khi mọi thông tin cần thiết đều được chứa trong JWT.
- Khả Năng Mở Rộng: JWT hỗ trợ việc mở rộng hệ thống mà không cần thay đổi cơ sở dữ liệu hoặc cấu trúc backend.
- Bảo Mật: Các claims trong JWT được bảo vệ bởi signature, giảm thiểu nguy cơ giả mạo hoặc thay đổi.

❖ Hạn Chế

- Quản Lý Token: Việc quản lý và hủy bỏ JWT có thể phức tạp, đặc biệt nếu token bị lộ hoặc cần được thu hồi trước thời hạn.
- Lưu Trữ Thông Tin Nhạy Cảm: Không nên lưu trữ thông tin nhạy cảm trong JWT vì chúng có thể bị đọc nếu token bị tiết lộ.

Ủy quyền dựa trên JWT là một phương pháp hiệu quả và linh hoạt trong việc quản lý quyền truy cập của người dùng trong ứng dụng, nhưng cần được xử lý cẩn thận để đảm bảo an ninh và hiệu quả [7].

2.2.3 Cơ chế xác thực

Xác thực thường sau khi người dùng đăng nhập, mỗi yêu cầu tiếp theo sẽ bao gồm JWT, cho phép người dùng truy cập các dịch vụ và tài nguyên được bảo vệ.

Quá Trình Xác Thực JWT sẽ diễn ra như sau:

- Đăng Nhập và Tạo Token:
    - Khi người dùng đăng nhập vào hệ thống bằng thông tin xác thực (Tên người dùng và mật khẩu), server sẽ xác minh thông tin này.
    - Nếu thông tin xác thực là hợp lệ, server tạo ra một JWT. JWT này chứa các thông tin (claims) về người dùng và một số metadata khác, như thời gian hết hạn của token.

PAYLOAD: DATA


```
{
  "iat": 1532351326,
  "nbf": 1532351326,
  "jti": "a8c043ae-ffcd-4162-9ff7-779671b58c87",
  "exp": 1532352226,
  "sub": 229,
  "fresh": false,
  "type": "access",
  "user_claims": {
    "device_id": "1a7c0b31-f5b8-49be-83f0efae72273d8d",
    "user_type_id": 1,
    "is_trial": true
  }
}
```


Hình 5.Page load của JWT

- Gửi Token cho Người Dùng:
    - JWT sau khi được tạo sẽ được gửi trở lại cho client. Client này có thể là trình duyệt web, ứng dụng di động, hoặc bất kỳ loại client nào khác.
- Lưu Trữ Token:
    - Người dùng lưu trữ JWT ở phía client, thường là trong local storage của trình duyệt hoặc trong bộ nhớ của ứng dụng.
- Gửi Token trong các Yêu Cầu Tiếp Theo:
    - Trong mỗi yêu cầu đến server, JWT sẽ được đính kèm, thường là trong header Authorization của HTTP request.
    - Cách thức này giúp server biết rằng yêu cầu đang được thực hiện bởi người dùng đã xác thực [5].
- Xác Thực và Ủy Quyền từ Server:
    - Khi server nhận một yêu cầu với JWT, nó sẽ kiểm tra tính hợp lệ của token này.
    - Server xác minh JWT bằng cách kiểm tra signature để đảm bảo rằng nó không bị thay đổi hoặc giả mạo từ lúc được tạo.
    - Nếu JWT hợp lệ, yêu cầu được xử lý; nếu không, server sẽ trả về lỗi xác thực.

Ưu Điểm của JWT trong Xác Thực:

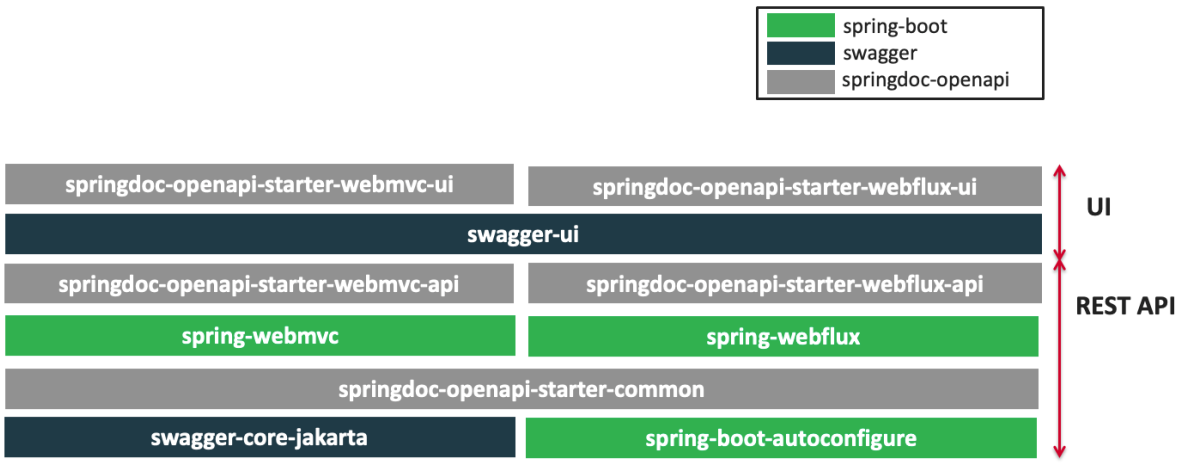
- Không Trạng Thái (Stateless): Server không cần lưu trữ thông tin xác thực hoặc phiên làm việc của người dùng, giúp giảm tải cho hệ thống.
- Bảo Mật và Hiệu Quả: JWT chứa signature để đảm bảo rằng nó không bị thay đổi. Nó cũng cho phép truyền thông tin một cách an toàn giữa client và server.

Hạn Chế:

- Bảo Mật: Thông tin trong JWT dễ bị đọc nếu token bị bắt giữ. Do đó, không nên lưu trữ thông tin nhạy cảm trong JWT.
- Quản Lý Phiên: JWT không thể bị hủy bỏ một cách dễ dàng trước khi hết hạn, vì vậy việc quản lý phiên có thể trở nên phức tạp nếu token bị chiếm đoạt [7].

2.3 Springdoc-openapi

Thư viện springdoc-openapi trong Java là một công cụ hữu ích giúp tự động hóa việc tạo tài liệu API cho các dự án Spring Boot. Thư viện này hoạt động bằng cách phân tích ứng dụng ở thời gian chạy để suy luận về ngữ nghĩa của API dựa trên cấu hình Spring, cấu trúc lớp và các annotation khác nhau [8].



Hình 6. Mô tả chung về springdoc-openai.

Springdoc-openapi là một thư viện được sử dụng trong các ứng dụng Spring Boot để tự động tạo ra tài liệu cho các API dựa trên OpenAPI 3 Specification. OpenAPI (trước đây được gọi là Swagger) là một tiêu chuẩn ngành công nghiệp cho việc mô tả, sản xuất và sử dụng RESTful services [8].

Tự Động Tạo Tài Liệu: springdoc-openapi có khả năng tự động phát hiện các endpoint trong ứng dụng và tạo ra tài liệu API tương ứng.

Hỗ Trợ OpenAPI 3: thư viện hỗ trợ phiên bản mới nhất của OpenAPI, cho phép mô tả API một cách chi tiết và đầy đủ hơn so với các phiên bản trước đây.

Tích Hợp UI Để Xem Tài Liệu: springdoc-openapi cung cấp tích hợp với Swagger UI, cho phép ta xem tài liệu API đã được tạo ra qua một giao diện người dùng trực quan và thân thiện.

Hỗ Trợ Đa Dạng: có khả năng hỗ trợ cả các ứng dụng WebFlux và Servlet, cũng như hỗ trợ cho các dịch vụ Reactive.

Cấu Hình Dễ Dàng: springdoc-openapi có thể được cấu hình một cách linh hoạt thông qua properties hoặc cấu hình Java.

Tích Hợp với Spring Security: có khả năng tích hợp với Spring Security để đưa ra các thông tin về xác thực và ủy quyền trong tài liệu API.

Tùy Chỉnh Tài Liệu: có thể tùy chỉnh tài liệu của mình thông qua việc sử dụng các annotation hoặc cung cấp thông tin bổ sung trong tài liệu OpenAPI.

Để sử dụng springdoc-openapi trong một ứng dụng Spring Boot, chỉ cần thêm dependency của springdoc-openapi vào file pom.xml (nếu sử dụng Maven) hoặc build.gradle (nếu sử dụng Gradle) của dự án. Sau khi thêm thành công, Spring Boot sẽ tự động cấu hình springdoc-openapi và có thể truy cập tài liệu API qua một URL mặc định, thường là /v3/api-docs cho tài liệu JSON và /swagger-ui.html để xem Swagger UI [8].

2.4 Tổng quan về MySQL

MySQL là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mã nguồn mở phổ biến, sử dụng ngôn ngữ truy vấn cơ sở dữ liệu SQL.

Hình 7. Logo MySQL

MySQL có mã nguồn mở, cho phép người dùng tự do sử dụng và sửa đổi theo nhu cầu của họ.

Dễ sử dụng: MySQL dễ cài đặt và sử dụng. Nó cung cấp một loạt các công cụ và giao diện để quản lý dữ liệu và cấu hình cơ sở dữ liệu.

Hiệu suất cao: MySQL được tối ưu hóa cho hiệu suất với các tính năng như bộ nhớ đệm truy vấn và bảng hash.

Bảo mật: MySQL cung cấp các tính năng bảo mật như xác thực, phân quyền và mã hóa để giúp bảo vệ dữ liệu.

Khả năng mở rộng: MySQL có thể xử lý một lượng lớn dữ liệu và được thiết kế để dễ dàng mở rộng [9] [9] [10].

CHƯƠNG 3: THỰC HÓA NGHIÊM CỨU

3.1 Lược đồ use-case

Hình 8. Sơ đồ use-case.

Nhân viên có trách nhiệm với các nhiệm vụ liên quan đến bán hàng và tạo hóa đơn, trong khi quản lý có quyền truy cập vào các thông tin chi tiết hơn và có thể thực hiện các quyết định dựa trên thống kê doanh thu và sản phẩm. Sơ đồ cung cấp cái nhìn tổng quan về cách phân chia quyền hạn và trách nhiệm giữa các vai trò khác nhau trong tổ chức hoặc hệ thống đó.

Nhân Viên:

- Tạo Hóa Đơn: Nhân viên có thể tạo hóa đơn cho các giao dịch.
- Thanh Toán: Có thể thực hiện việc thanh toán cho các hóa đơn đã tạo.
- Cập Nhật Trạng Thái Bàn: Nhân viên có trách nhiệm cập nhật trạng thái của bàn của quán cafe.

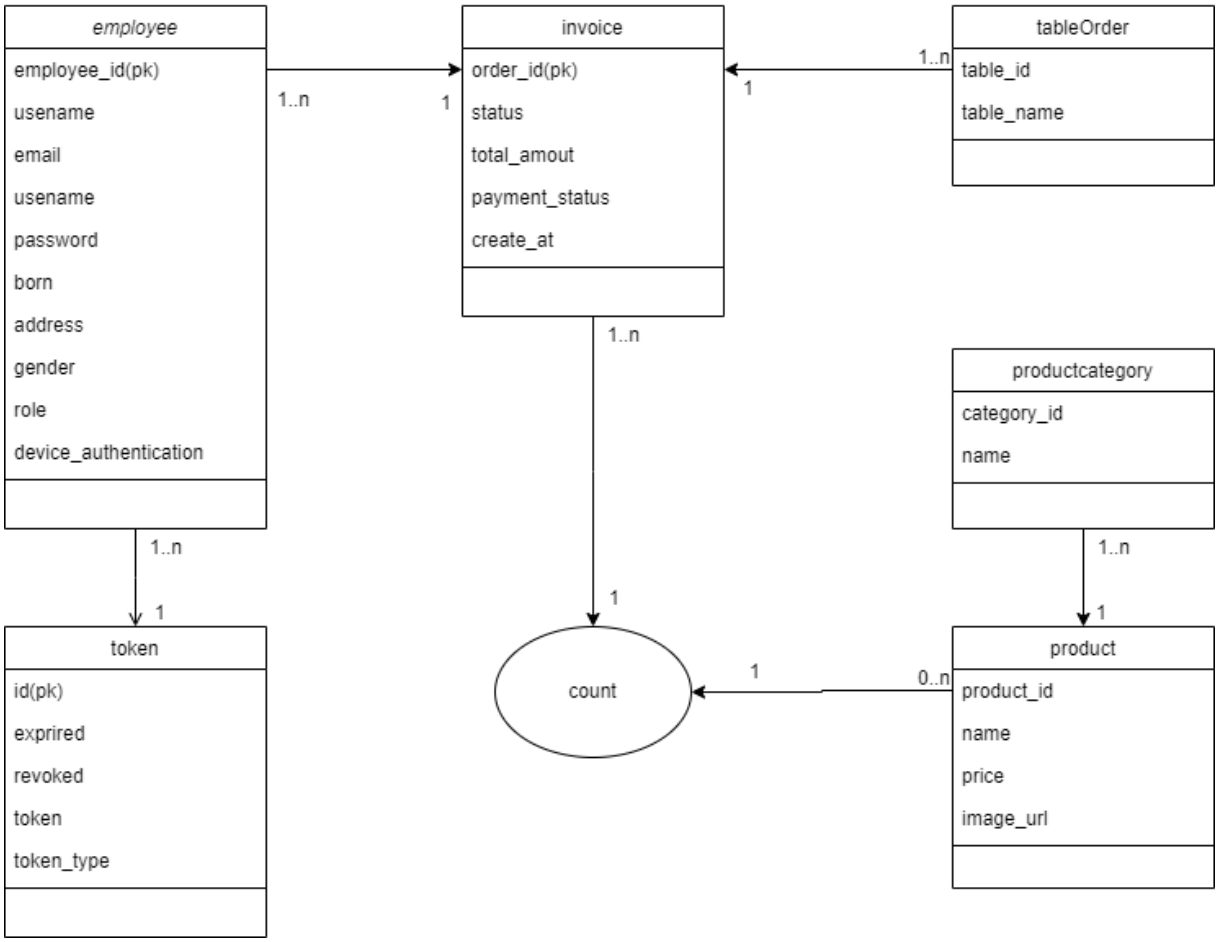
Quản Lý:

- Duyệt Hóa Đơn: Quản lý có thẩm quyền duyệt các hóa đơn được tạo bởi nhân viên.

- Thống Kê Doanh Thu: Quản lý thực hiện các báo cáo và thống kê doanh thu từ hoạt động bán hàng.
- Thống Kê Sản Phẩm: Quản lý có khả năng thực hiện thống kê về các sản phẩm, có thể liên quan đến việc theo dõi tồn kho, xu hướng bán hàng, v.v.

Biểu đồ cũng cho thấy rằng cả nhân viên và quản lý đều có thể thực hiện các hoạt động như "tạo hóa đơn" và "thanh toán", nhưng chỉ có quản lý mới có quyền "duyệt hóa đơn" và thực hiện các hoạt động thống kê. Điều này phản ánh một mô hình phân quyền trong đó nhân viên xử lý các nhiệm vụ hàng ngày trong khi quản lý giám sát và đưa ra các quyết định dựa trên dữ liệu tổng hợp.

3.2 Mô hình thực thể kết hợp

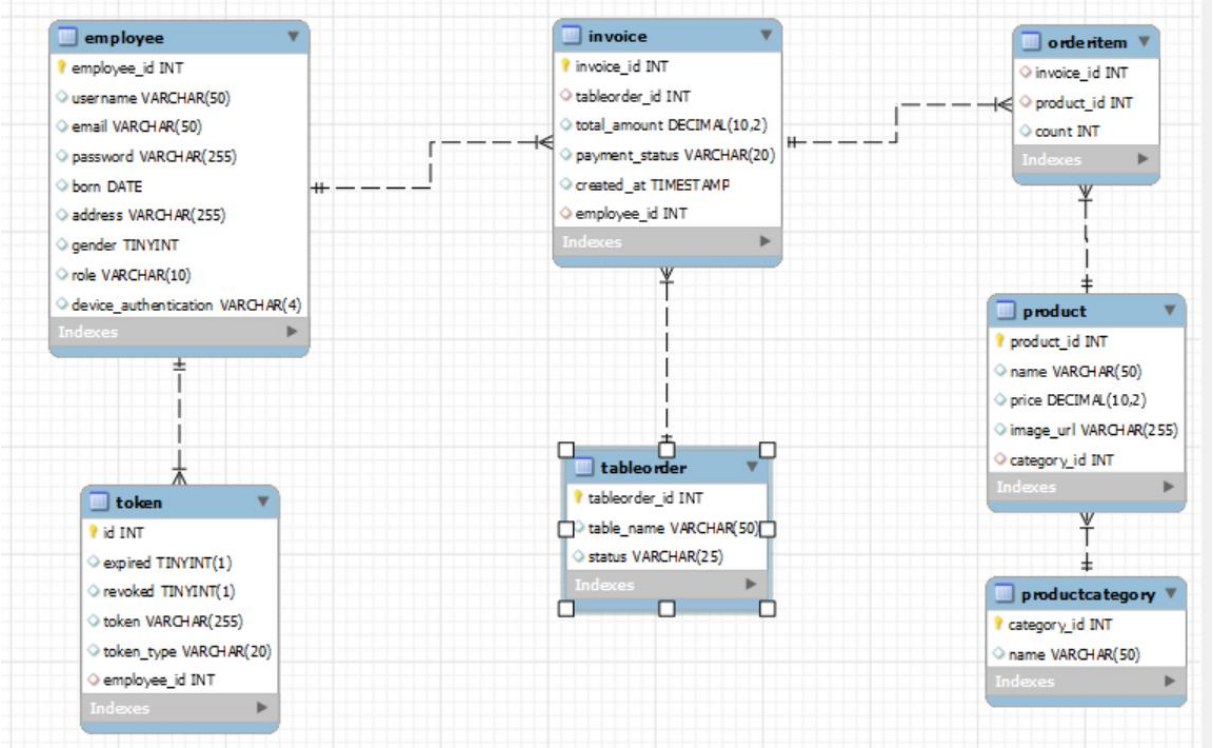


Hình 9 Mô hình thực thể kết hợp

Hình ảnh trên mô tả một mô hình cơ sở dữ liệu quan hệ, thường được sử dụng trong thiết kế và lập trình cơ sở dữ liệu.

Mỗi mối quan hệ giữa các bảng được thể hiện bằng các đường nối có chú thích chỉ ra mối quan hệ số lượng (ví dụ: 1-n tức là một đến nhiều). Điều này chỉ ra cách các bảng liên kết với nhau.

3.4 Mô hình mức dữ liệu logic



Hình 10 Mô hình mức dữ liệu logic

Bảng 1. Bảng "productcategory"

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	Category_id	Int	Khóa chính	Mã thể loại
2	name	Varchar(50)		Tên thể loại

Bảng 2. Bảng "product"

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	product_id	Int	Khóa chính	Mã sản phẩm
2	name	Varchar(50)		Tên sản phẩm
3	price	Decimal(10, 2)		Giá của sản phẩm
4	image_url	Varchar(255)		Đường dẫn của ảnh

5	Category_id	int	Khóa ngoại	Mã thể loại

Bảng 3. Bảng "OrderItem"

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	Category_id	Int	Khóa ngoại	Mã thể loại
2	product_id	int	Khóa ngoại	Mã sản phẩm
3	count	Int		Số lượng sản phẩm

Bảng 4. Bảng "Invoice"

Stt	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	Invoice_id	Int	Khóa chính	Mã hóa đơn
2	Tableorder_id	Int	Khóa ngoại	Mã bàn
3	Employee_id	Int	Khóa ngoại	Mã nhân viên
4	Payment_status	Varchar(20)		Trạng thái hóa đơn
5	Total_amount	Decimal(10, 2)		Tổng giá trị đơn hàng
6	Create_at	Timetemp		Ngày lập hóa đơn

Bảng 5. Bảng "productcategory"

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	employee_id	Int	Khóa chính	Mã người dùng
2	usename	Varchar(50)		Tên người dùng
3	email	Varchar(50)		Email của người dùng
4	password	Varchar(255)		Mật khẩu của người dùng
5	Born	Date		Ngày sinh cảu người dùng
6	address	Varchar(255)		Địa chỉ của người dùng
7	gender	Tinyint		Giới tính của người dùng

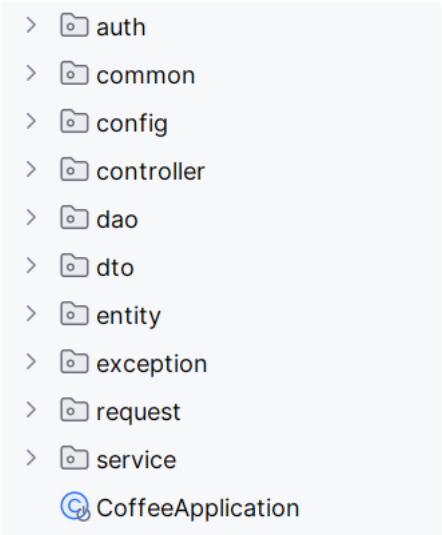
8	role	Varchar(10)		Vai trò của người dùng
9	device_authentication	Varchar(10)		Mã xác thực

Bảng 6. Bảng "token"

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	id	Int	Khóa chính	Mã của token
2	expired	Tinyint		Token đã hết hạn chưa
3	revoked	Tinyint		Token đã thu hồi chưa
4	token_type	Varchar(20)		Loại của token
5	employee_id	int	Khóa ngoại	Mã của người dùng

3.5 Kiến trúc hệ thống

Dự án được xây dựng kiến trúc như sau



Hình 11. Kiến trúc dự án.

Dự án của tôi được tổ chức thành các package chức năng, điều này giúp tăng khả năng bảo trì và mở rộng trong tương lai. Dưới đây là một số giả định về mục đích và chức năng của từng package:

- auth: có thể chứa các class liên quan đến xác thực và ủy quyền người dùng, bao gồm cơ chế đăng nhập, JWT filters và các security configurations.

- common: chúng có thể được xem như là các tiện ích hoặc thành phần chung được sử dụng xuyên suốt ứng dụng.
- config: thường chứa các class cấu hình cho ứng dụng, như cấu hình bean cho Spring, cấu hình bảo mật, database hoặc bất kỳ cấu hình nào cần thiết cho ứng dụng.
- controller: chứa các class Controller, nơi xử lý các yêu cầu HTTP, chuyển đổi chúng thành các hành động trong service.
- dao (Data Access Object): cung cấp một abstraction layer cho việc truy cập dữ liệu, các interface cho JPA repositories hoặc các phương thức truy cập dữ liệu cụ thể khác.
- dto (Data Transfer Object): chứa các object được sử dụng để chuyển dữ liệu giữa các lớp và các hệ thống mà không cần dùng đến entity models trực tiếp.
- entity: chứa các class Java Persistence API (JPA) entities, đại diện cho các bảng trong cơ sở dữ liệu.
- exception: có thể chứa các class xử lý ngoại lệ, như custom exception handlers.
- request: có thể chứa các class định nghĩa các đối tượng yêu cầu được gửi từ client đến server.
- service: chứa business logic của ứng dụng, nơi các nghiệp vụ được thực hiện.

Cấu trúc dự án này được tổ chức theo tính năng giúp việc tổ chức code trong một ứng dụng Spring Boot, giúp việc bảo trì và mở rộng ứng dụng trở nên dễ dàng hơn.

3.6 Thiết kế API

Việc xây dựng các API trong một ứng dụng Spring Boot thường tuân theo một quy trình phát triển nhất định. Dưới đây là các bước cơ bản để xây dựng các API, từ việc thiết kế cơ sở dữ liệu cho đến triển khai các endpoints API:

❖ Thiết kế Cơ Sở Dữ liệu và Mô hình Entity

Bước Đầu Tiên: Xác định các bảng cần thiết trong cơ sở dữ liệu và mối quan hệ giữa chúng (ví dụ: Invoice, OrderItem, Product, v.v.).

Mô hình Entity: Dựa trên thiết kế cơ sở dữ liệu, tạo các lớp entity trong Spring Boot. Sử dụng JPA để map các lớp này với bảng cơ sở dữ liệu.

❖ Tạo Data Transfer Objects (DTOs)

Mục Đích: DTOs là đối tượng dùng để chuyển dữ liệu giữa client và server mà không cần phải sử dụng trực tiếp các entity.

Thực Hiện: Tạo các DTOs cho các thao tác như tạo mới, cập nhật dữ liệu.

- ❖ Xây dựng Repository Layer

Repository: Sử dụng Spring Data JPA để tạo các repository, cung cấp các phương thức truy vấn cơ sở dữ liệu mặc định và tùy chỉnh.

- ❖ Tạo Service Layer: Convert dữ liệu giữa các DTOs và entity trong service layer.
- ❖ Xây dựng Controller Layer

Endpoint API: Tạo các controller để xử lý các yêu cầu HTTP. Mỗi controller sẽ tương tác với một hoặc nhiều service.

Mapping: Sử dụng @GetMapping, @PostMapping, @PutMapping, @DeleteMapping để map các hành động CRUD đến các phương thức tương ứng trong controller.

- ❖ Xác thực và Phân quyền

Bảo mật: Sử dụng Spring Security để thêm xác thực và phân quyền cho các API.

- ❖ Tài liệu API

Swagger: Tạo tài liệu cho các API để dễ dàng tham khảo và kiểm thử.

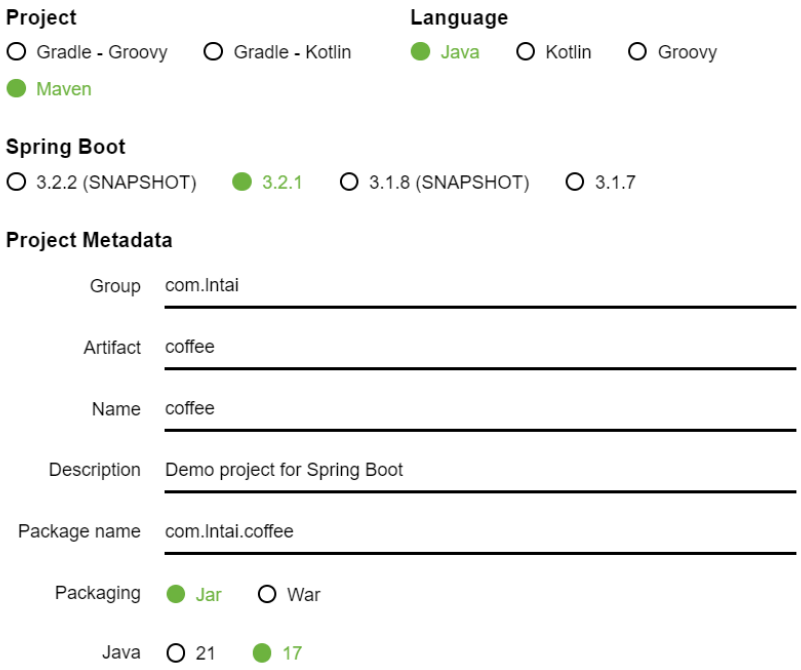
Bằng cách theo dõi các bước này, Tôi có thể xây dựng một tập hợp các API hiệu quả và bảo mật cho ứng dụng của mình. Mỗi bước đều quan trọng và đóng góp vào việc tạo ra một hệ thống API hoàn chỉnh.

CHƯƠNG 4: KẾT QUẢ NGHIÊM CỨU

4.1 Khởi tạo dự án

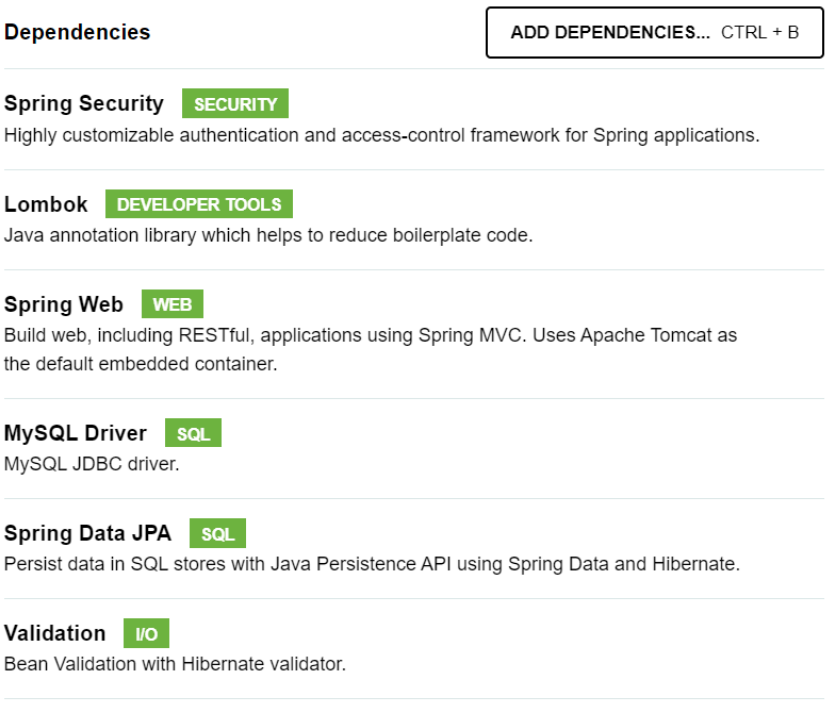
Truy cập vào https://start.spring.io/

Chọn phiên bản và ngôn ngữ như sau:



Hình 12. Khởi tạo dự án Java Spring Boot.

Và chọn các Dependencies phù hợp



Hình 13. Một số dependencies của dự án.

Các dependencies này cho phép ta xây dựng một ứng dụng web an toàn, có khả năng kết nối với cơ sở dữ liệu MySQL, có tài liệu API tự động và hỗ trợ việc thực thi test.

Khi chọn đầy đủ các dependencies cũng như chọn các option cho dự án. Nhấn "GENERATE" sẽ tạo ra file zip, giải nén tiếp tục phát triển chương trình

start.spring.io là một trang web được cung cấp bởi Spring, nơi chúng ta có thể dễ dàng tạo ra các dự án mới sử dụng Spring Boot. Trang này được thiết kế để giúp các nhà phát triển khởi tạo cấu trúc cơ bản của một ứng dụng, bao gồm cấu hình Maven hoặc Gradle, cấu trúc thư mục, các file cấu hình cơ bản và một tập hợp các dependencies cần thiết để bắt đầu một ứng dụng.

4.2 Cấu hình dự án

Việc cấu hình trong dự án Spring Boot được thực hiện trên file application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/CoffeeShopDB?useSSL=false&useUnicode=yes&characterEncoding=UTF-8&al
spring.datasource.username=root
spring.datasource.password=123456

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.data.rest.base-path=/api

application.security.jwt.secret-key=f2649f5654eca5d1a8fbad9130b2e8b62c98f9539be4abc9dde37800b396efca
# 1 ngày
application.security.jwt.expiration= 86400

# 7 ngày
application.security.jwt.refresh-token.expiration=604800000
```

Hình 14. Cấu hình dự án

Có một số điều cần chú ý như sau:

- Kết nối với database: tên database "CoffeeShopDB", tài khoản kết nối "root".
- "secret-key": đây là khóa bí mật của dự án.
- Gán giá trị thời gian hết hạn token.

4.2 Bảo mật chương trình

4.2.1 Xây dựng Security Filters Chain

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
            .csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(req ->
                    req.requestMatchers(WHITE_LIST_URL) AuthorizedUrl
                            .permitAll() AuthorizationManagerRequestMat...
                            .requestMatchers("/api/v1/management/**").hasAnyRole(ADMIN.name(), MANAGER.name())
                            .requestMatchers(GET, "/api/v1/management/**").hasAnyAuthority(ADMIN_READ.name(), MANAGER_READ.name())
                            .requestMatchers(POST, "/api/v1/management/**").hasAnyAuthority(ADMIN_CREATE.name(), MANAGER_CREATE.name())
                            .requestMatchers(PUT, "/api/v1/management/**").hasAnyAuthority(ADMIN_UPDATE.name(), MANAGER_UPDATE.name())
                            .requestMatchers(DELETE, "/api/v1/management/**").hasAnyAuthority(ADMIN_DELETE.name(), MANAGER_DELETE.name())
                            .anyRequest() AuthorizedUrl
                            .authenticated()
            )
            .sessionManagement(session -> session.sessionCreationPolicy(STATELESS))
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
            .logout(logout ->
                    logout.logoutUrl("/api/v1/auth/logout")
                            .addLogoutHandler(logoutHandler)
                            .logoutSuccessHandler((request, response, authentication) -> SecurityContextHolder.clearContext())
            )
    ;
```

Hình 15. Xây dựng security filters.

Đây là một phần của phương thức securityFilterChain trong một lớp cấu hình. Dưới đây là mô tả từng phần của đoạn code:

CSRF Protection Disabled: .csrf(AbstractHttpConfigurer::disable) - Dòng này vô hiệu hóa bảo vệ CSRF (Cross-Site Request Forgery), điều này thường chỉ nên được làm trong các dịch vụ API REST, nơi mà các token CSRF không được yêu cầu.

Authorization Configuration: Phần còn lại của đoạn code cấu hình các quy tắc ủy quyền dựa trên các URL và phương thức HTTP cụ thể:

- Whitelist URLs: Các URL được định nghĩa trong WHITE_LIST_URL sẽ được phép truy cập mà không cần xác thực.

```
private static final String[] WHITE_LIST_URL = {"" +
        "/api/v1/auth/**",
        "/v3/api-docs",
        "/v3/api-docs/**",
        "/swagger-ui/**",
        "/swagger-ui.html"};
```

Hình 16. WHITE_LIST_URL.

- API URLs: Các URL bắt đầu bằng /api/v1/management/** đều yêu cầu quyền ủy quyền cụ thể, dựa trên vai trò (ADMIN hoặc MANAGER) và hành động cụ thể (đọc, tạo, cập nhật, xóa).

- Any Other Requests: Bất kỳ yêu cầu nào không khớp với các quy tắc trên phải được xác thực.
- Session Management: .sessionManagement(session -> session.sessionCreationPolicy(STATELESS)) - Cấu hình này chỉ ra rằng ứng dụng không nên duy trì trạng thái phiên làm việc (session state), điều này phù hợp với các ứng dụng không sử dụng phiên dựa trên cookie mà thay vào đó dùng tokens như JWT.
- Authentication Provider: .authenticationProvider(authenticationProvider) - Cho phép cài đặt một nhà cung cấp xác thực tùy chỉnh.
- JWT Authentication Filter:. addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class) - Thêm một bộ lọc JWT trước bộ lọc xác thực tên người dùng/mật khẩu mặc định. Bộ lọc này sẽ kiểm tra sự hiện diện của JWT và xác thực nó trước khi yêu cầu được xử lý bởi bộ lọc xác thực.
- Logout Configuration: Cấu hình cho quy trình đăng xuất, bao gồm URL đăng xuất và các xử lý khi đăng xuất thành công, như xóa ngữ cảnh bảo mật.

Mỗi khi có một yêu cầu HTTP, cấu hình này đảm bảo rằng yêu cầu đó phải qua các kiểm tra xác thực và ủy quyền dựa trên quy tắc được định nghĩa trước khi tiếp tục đến các bộ phận xử lý yêu cầu khác trong ứng dụng.

4.2.2 Xây dựng xác thực JWT

JwtAuthenticationFilter là một lớp mở rộng từ OncePerRequestFilter của Spring Security, được sử dụng để xác thực người dùng dựa trên JSON Web Tokens (JWT).

```
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;
    private final TokenRepository tokenRepository;

    no usages  taiz
    @Override
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
            @NonNull FilterChain filterChain
    ) throws ServletException, IOException {
        if (request.getServletPath().contains("/api/v1/auth")) {...}
        final String authHeader = request.getHeader( s: "Authorization");
        final String jwt;
        final String userEmail;
        if (authHeader == null ||!authHeader.startsWith("Bearer ")) {...}
        jwt = authHeader.substring( beginIndex: 7);
        userEmail = jwtService.extractUsername(jwt);
        if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {...}
        filterChain.doFilter(request, response);
    }
}
```

Hình 17. Xây dựng JwtAuthenticationFilter

Bỏ qua xác thực đối với đường dẫn đăng nhập: Nếu yêu cầu HTTP đến đường dẫn liên quan đến xác thực (/api/v1/auth), filter sẽ cho phép yêu cầu đi qua mà không cần xác thực.

Lấy và kiểm tra Header Authorization: Filter kiểm tra header Authorization trong yêu cầu HTTP để lấy JWT. Nếu header không tồn tại hoặc không bắt đầu bằng "Bearer ", yêu cầu sẽ tiếp tục mà không cần xác thực.

Trích Xuất và Xác Thực Token: Nếu có JWT, filter trích xuất thông tin người dùng từ token và sử dụng JwtService và TokenRepository để kiểm tra tính hợp lệ của token (không hết hạn và không bị thu hồi).

Tạo và thiết lập Authentication: Nếu token hợp lệ, filter sẽ tạo một UsernamePasswordAuthenticationToken mới với thông tin người dùng và các quyền hạn của họ. Token này sau đó được thiết lập vào SecurityContext, cho phép Spring

Security xác định người dùng đã được xác thực và có thể truy cập vào tài nguyên bảo vệ.

Tiếp tục chuỗi Filter: Cuối cùng, yêu cầu được chuyển đến filter tiếp theo trong chuỗi filter hoặc, nếu không còn filter nào, đến servlet hoặc controller để xử lý.

Tóm lại, JwtAuthenticationFilter là một cơ chế xác thực mạnh mẽ trong Spring Security, đảm bảo rằng chỉ những yêu cầu có JWT hợp lệ mới có thể truy cập vào các tài nguyên được bảo vệ của ứng dụng.

4.2.3 Xác thực người dùng

lớp cấu hình này định nghĩa cách thức xác thực người dùng thông qua UserDetailsService, sử dụng EmployeeRepository để lấy thông tin người dùng và xác định các bean cần thiết để mã hóa và so sánh mật khẩu cũng như quản lý quá trình xác thực.

```
@Configuration
@RequiredArgsConstructor
public class ApplicationConfig {
    private final EmployeeRepository repository;
    taiz
    @Bean
    public UserDetailsService userDetailsService(){
        return username -> repository.findByEmail(username)
                .orElseThrow(()-> new UsernameNotFoundException("user not found"));
    }
    taiz
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }
    taiz
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception{
        return config.getAuthenticationManager();
    }
    taiz *
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

Hình 18. Đoạn code cấu hình liên quan đến xác thực người dùng.

@Configuration: Annotation này chỉ định rằng lớp ApplicationConfig là một lớp cấu hình, nơi Spring Container có thể tìm thấy và đăng ký các bean.

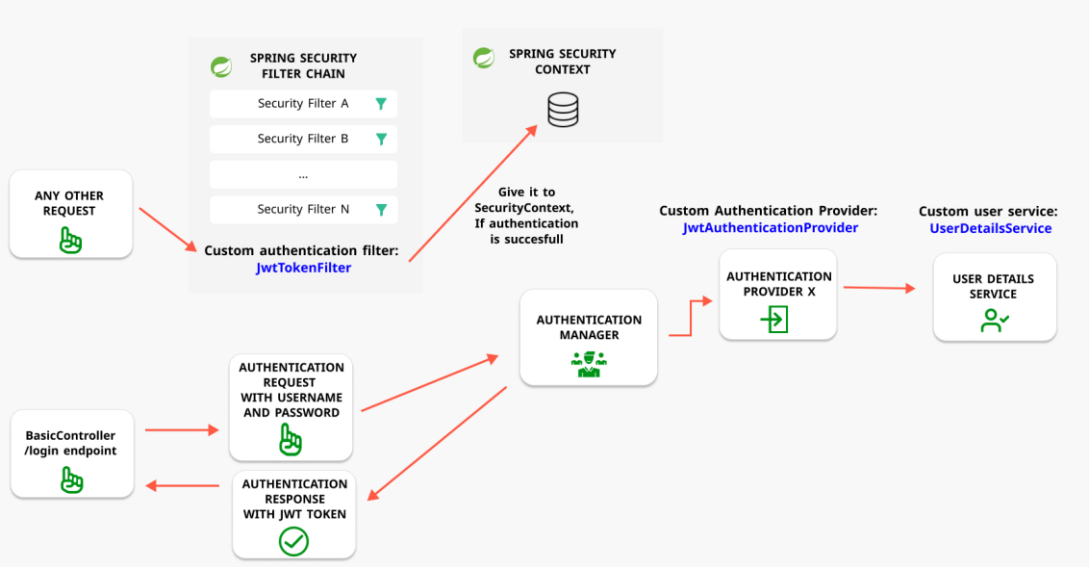
@RequiredArgsConstructor: Đây là một annotation của Lombok, tạo tự động constructor với các tham số cuối cùng cho tất cả các trường được khai báo là final. Ở đây nó sẽ tạo một constructor cho EmployeeRepository repository.

UserDetailsService Bean: Bean userDetailsService() được sử dụng để tải thông tin của người dùng. Trong trường hợp này, nó sử dụng phương thức findByEmail() từ EmployeeRepository để tìm người dùng theo email, nếu không tìm thấy sẽ ném ra ngoại lệ UsernameNotFoundException.

AuthenticationProvider Bean: Bean authenticationProvider() tạo một đối tượng DaoAuthenticationProvider mới. DaoAuthenticationProvider sử dụng UserDetailsService để tải thông tin người dùng và so sánh mật khẩu đã mã hóa lưu trong cơ sở dữ liệu với mật khẩu được cung cấp khi đăng nhập. Nó cũng được cấu hình với một PasswordEncoder để mã hóa mật khẩu.

AuthenticationManager Bean: Bean authenticationManager() lấy AuthenticationManager từ AuthenticationConfiguration đã cung cấp. AuthenticationManager là thành phần trung tâm trong Spring Security để xử lý quá trình xác thực.

PasswordEncoder Bean: Bean passwordEncoder() tạo một đối tượng BCryptPasswordEncoder, là một phương pháp mã hóa mật khẩu dựa trên bcrypt, cung cấp mật khẩu mã hóa mạnh mẽ.



Hình 19. Mô tả quá trình xác thực người dùng.

4.2.4 Xây dựng token

Để xây dựng JWT, ta cần có ba thành phần: header, payload và signature. Header chứa thông tin về thuật toán mã hóa và kiểu token. Payload chứa các thông tin cần xác thực, như id người dùng, quyền truy cập, thời gian hết hạn. Signature là kết quả của việc mã hóa header và payload bằng một khóa bí mật.

```
public String buildToken(
        Map<String, Object> extraClaims,
        UserDetails userDetails,
        long expiration
){
    return Jwts
            .builder()
            .setClaims(extraClaims)
            .setSubject(userDetails.getUsername())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + expiration))
            .signWith(getSignInKey(), SignatureAlgorithm.HS256)
            .compact();
}
```

Hình 20. Phương thức buildToken.

Phương thức buildToken trong đoạn code này là một phần của một dịch vụ quản lý token JWT trong một ứng dụng Spring Boot. Nó tạo ra và trả về một JWT dựa trên thông tin người dùng và một số claims bổ sung. Dưới đây là mô tả cụ thể của từng bước trong quy trình tạo token:

- setClaims(extraClaims): Thiết lập các claims bổ sung cho token. Các claims này có thể chứa thông tin phiên bản, quyền hạn của người dùng, hoặc bất kỳ thông tin nào khác mà muốn bao gồm trong token.
- setSubject(userDetails.getUsername()): Đặt "subject" của JWT, thường là tên đăng nhập hoặc ID duy nhất của người dùng, lấy từ đối tượng UserDetails.
- setIssuedAt(new Date(System.currentTimeMillis())): Đặt thời gian phát hành (issued at) của token, sử dụng thời gian hiện tại của hệ thống.
- setExpiration(new Date(System.currentTimeMillis() + expiration)): Đặt thời gian hết hạn (expiration) cho token dựa trên thời gian hiện tại cộng với số lượng milliseconds được chỉ định trong tham số expiration.

- signWith(getSignInKey(), SignatureAlgorithm.HS256): Ký token sử dụng thuật toán mã hóa HS256 và khóa bí mật. Phương thức getSignInKey() được sử dụng để lấy khóa đã được mã hóa và chuẩn bị sẵn sàng để sử dụng trong quá trình ký.
- compact(): Nén và tạo chuỗi JWT hoàn chỉnh.

```
public boolean isTokenValid(String token, UserDetails userDetails) {
    final String username = extractUsername(token);
    return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);
}

1 usage  taiz
private boolean isTokenExpired(String token) { return extractExpiration(token).before(new Date()); }

1 usage  taiz
private Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration); }
1 usage  taiz
private Claims extractAllClaims(String token) {
    return Jwts
            .parserBuilder()
            .setSigningKey(getSignInKey())
            .build()
            .parseClaimsJws(token)
            .getBody();
}
2 usages  taiz
private Key getSignInKey() {
    byte[] keyBytes = Decoders.BASE64.decode(secretKey);
    return Keys.hmacShaKeyFor(keyBytes);
}
```

Hình 21. Đoạn code kiểm tra JWT tokens.

Đoạn mã Java này là một phần của lớp dịch vụ JWT trong Spring Boot, chịu trách nhiệm kiểm tra tính hợp lệ của JWT tokens. Các phương thức cụ thể được sử dụng để xác minh tính hợp lệ của token và để xác định xem token đó có hết hạn hay không. Dưới đây là giải thích chi tiết cho từng phương thức:

- isTokenValid: Phương thức công khai này nhận vào token JWT và đối tượng UserDetails. Nó trích xuất tên người dùng từ token và so sánh với tên người dùng từ UserDetails. Nếu cả hai khớp nhau và token chưa hết hạn, phương thức này sẽ trả về true, còn không sẽ trả về false.
- isTokenExpired: Phương thức riêng tư này kiểm tra xem token JWT đã hết hạn chưa bằng cách so sánh thời gian hết hạn của token với thời gian hiện tại.
- extractExpiration: Phương thức này trích xuất thời gian hết hạn từ token JWT.

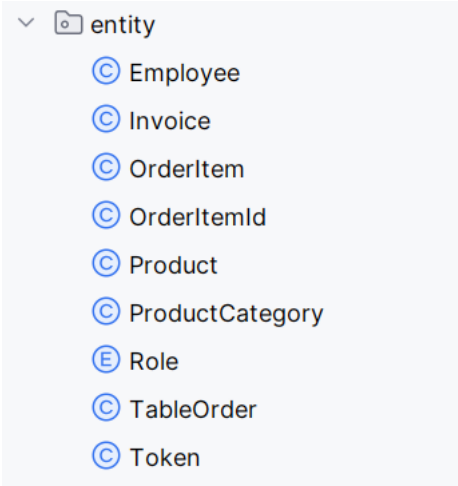
- extractAllClaims: Phương thức này trích xuất tất cả các claims từ token JWT. Nó sử dụng khóa bí mật để giải mã và kiểm tra chữ ký của token.
- getSignInKey: Phương thức này tạo ra một khóa bảo mật từ một chuỗi được mã hóa Base64 (secretKey). Khóa này sau đó được sử dụng để ký và xác minh token JWT.

Các phương thức này làm việc cùng nhau để đảm bảo rằng mỗi token JWT được kiểm tra kỹ càng trước khi cho phép người dùng truy cập vào các tài nguyên được bảo vệ trong ứng dụng.

4.3 Xây dựng các API

4.3.1 Tạo lớp Entity

Tạo các lớp entity tương ứng với các bảng cơ sở dữ liệu. Sử dụng JPA để map các entity với bảng.



Hình 22. Lớp Entity.

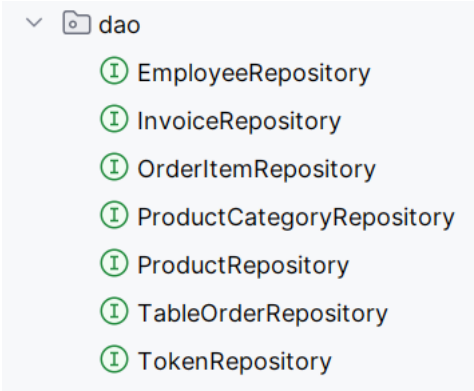
Để tạo các lớp entity và map chúng với các bảng cơ sở dữ liệu sử dụng JPA, ta cần tuân theo những bước cơ bản sau:

- Tạo Lớp Entity: Mỗi lớp trong package entity sẽ tương ứng với một bảng trong cơ sở dữ liệu. Lớp này sẽ chứa các trường dữ liệu tương ứng với các cột trong bảng.
- Sử Dụng Annotation: Dùng các annotation của JPA như @Entity, @Table, @Id, @Column, v.v., để chỉ định mối quan hệ giữa lớp Java và bảng cơ sở dữ liệu.
- Xác Định Khóa Chính: Sử dụng annotation @Id để xác định trường nào là khóa chính của bảng.

- Quan Hệ giữa các Bảng: Định nghĩa mối quan hệ giữa các bảng sử dụng các annotation như @OneToMany, @ManyToOne, @ManyToMany, @OneToOne.
- Cấu Hình JPA: Cấu hình DataSource và EntityManager trong các file cấu hình của Spring Boot để quản lý các entity và giao dịch với cơ sở dữ liệu.

4.3.2 Tạo lớp DAO

Đây là nơi chứa các interface và class truy cập cơ sở dữ liệu, thường là các repository hoặc DAOs sử dụng Spring Data JPA.



Hình 23. Lớp DAO.

Trong xây dựng API, lớp DAO đóng một vai trò quan trọng trong việc tách biệt logic truy cập dữ liệu khỏi tầng logic nghiệp vụ. Đây là những vai trò cụ thể của lớp DAO:

- Trừu Tượng Hóa Truy Cập Dữ Liệu: DAO cung cấp một giao diện trừu tượng để truy cập cơ sở dữ liệu. Điều này cho phép tầng nghiệp vụ tương tác với cơ sở dữ liệu mà không cần biết về chi tiết cách dữ liệu được lưu trữ hay truy vấn.
- Giảm Sự Phức Tạp: DAO xử lý tất cả công việc liên quan đến cơ sở dữ liệu như mở/kết thúc phiên, giao dịch, xử lý ngoại lệ, v.v., giúp giảm bớt sự phức tạp cho các nhà phát triển tập trung vào logic nghiệp vụ.
- Tích Hợp với Spring Data JPA: thực hiện dưới dạng các interface Repository với sự hỗ trợ của Spring Data JPA.
- Ổn Định API: DAO giúp đảm bảo rằng các thay đổi trong cách lưu trữ hoặc truy cập dữ liệu không ảnh hưởng đến các endpoint API đã được công bố, giữ cho API ổn định và nhất quán trong thời gian dài.

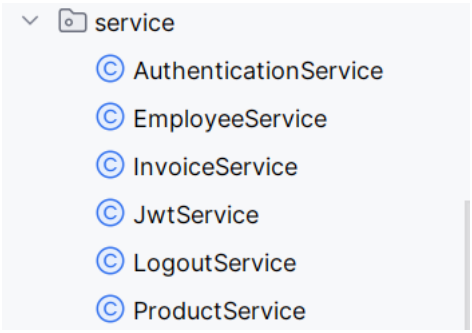
```
public interface ProductRepository extends JpaRepository<Product, Integer> {
    1 usage   taiz
    @Query(value = """
        SELECT p1_0 FROM Product p1_0
        WHERE p1_0.name LIKE %:name%
        """)
    List<Product> findByNameContaining(@Param("name") String name);
```

Hình 24. ProductRepository.

Đây là đoạn code trong class ProductRepository, truy vấn được xác định sử dụng annotation @Query và nó trả về một danh sách các đối tượng Product dựa trên tên sản phẩm mà có chứa chuỗi được cung cấp.

4.3.3 Lớp service

Tầng Service (Service Layer) là một thành phần trung gian giữa tầng trình bày (thường là Controllers trong mô hình MVC) và tầng truy cập dữ liệu (DAO). Dưới đây là những vai trò và trách nhiệm chính của tầng Service



Hình 25. Lớp service.

Logic Nghiệp Vụ: Service Layer chứa logic nghiệp vụ của ứng dụng. Nó định nghĩa các quy tắc và thủ tục cần thiết để thao tác và biến đổi dữ liệu.

Trừu Tượng Hóa và Tái Sử Dụng Code: Tầng này cung cấp một giao diện trừu tượng cho tầng trình bày, giúp tái sử dụng logic nghiệp vụ ở nhiều phần khác nhau trong ứng dụng mà không cần quan tâm đến chi tiết triển khai.

Giao Tiếp với DAO: Service Layer gọi đến DAO để lấy dữ liệu từ cơ sở dữ liệu và sau đó có thể thực hiện các biến đổi hoặc logic nghiệp vụ trước khi trả dữ liệu về cho client.

Validation (Kiểm Tra): Kiểm tra dữ liệu đầu vào và đảm bảo rằng nó đáp ứng các yêu cầu và ràng buộc của nghiệp vụ trước khi thực hiện các thao tác trên dữ liệu.

Security (Bảo Mật): Thực hiện các kiểm tra bảo mật như xác thực và ủy quyền, đảm bảo rằng người dùng chỉ có thể truy cập hoặc thay đổi dữ liệu mà họ có quyền.

Vai trò của Service Layer là rất quan trọng trong việc đảm bảo rằng mô hình thiết kế của ứng dụng được tổ chức tốt, dễ mở rộng, bảo trì và kiểm thử.

Vd: EmployeeService

```
public class EmployeeService {
    private final PasswordEncoder passwordEncoder;
    private final EmployeeRepository repository;
    1 usage   taiz *
    public void changePassword(ChangePasswordRequest request, Principal connectedUser) {

        var user = (Employee) ((UsernamePasswordAuthenticationToken) connectedUser).getPrincipal();

        // kiem tra mat khau
        if (!passwordEncoder.matches(request.getCurrentPassword(), user.getPassword())) {
            throw new EmployeeException("Mật khẩu sai");
        }
        // kiem tra mat khau giong nhau
        if (!request.getNewPassword().equals(request.getConfirmationPassword())) {
            throw new EmployeeException("Mật khẩu không giống nhau");
        }
        // cap nhat
        user.setPassword(passwordEncoder.encode(request.getNewPassword()));
        // luu mat khau
        repository.save(user);
    }
}
```

Hình 26. EmployeeService.

Đoạn mã trên đây mô tả phương thức changePassword trong lớp EmployeeService.

Nhận Đối Tượng Người Dùng:

- Biến connectedUser chứa thông tin xác thực của người dùng hiện tại.

Cast đối tượng xác thực sang Employee để có thể làm việc với đối tượng người dùng.

Kiểm Tra Mật Khẩu Hiện Tại:

- Sử dụng PasswordEncoder để kiểm tra xem mật khẩu hiện tại người dùng nhập vào có khớp với mật khẩu được mã hóa lưu trong cơ sở dữ liệu không.
- Nếu không khớp, một ngoại lệ EmployeeException sẽ được ném ra với thông điệp "Mật khẩu sai".

Kiểm Tra Tính Nhất Quán của Mật Khẩu Mới:

- So sánh mật khẩu mới và mật khẩu xác nhận để đảm bảo chúng giống nhau.
- Nếu chúng không khớp, một ngoại lệ EmployeeException được ném ra với thông điệp "Mật khẩu không giống nhau".

Cập Nhật Mật Khẩu:

- Mã hóa mật khẩu mới sử dụng PasswordEncoder và cập nhật cho đối tượng người dùng.

Lưu Thông Tin Người Dùng:

- Sử dụng repository để lưu đối tượng người dùng đã được cập nhật vào cơ sở dữ liệu.

Phương thức này đảm bảo rằng mật khẩu chỉ có thể được thay đổi khi người dùng biết mật khẩu hiện tại của mình và nhập chính xác mật khẩu mới cùng với xác nhận mật khẩu. Đây là một phương thức tiêu chuẩn trong quản lý người dùng để bảo mật thông tin tài khoản.

4.3.4 Lớp Controller

Trong Spring Boot, lớp Controller đóng một vai trò trung tâm trong việc xử lý các yêu cầu HTTP đến và gửi phản hồi HTTP đến client. Nơi Controller xử lý logic điều hướng và phối hợp giữa Model.

```
@RestController
@RequestMapping("/api/v1/change-password")
@RequiredArgsConstructor
public class EmployeeController {

    private final EmployeeService service;

    taiz
    @PatchMapping
    public ResponseEntity<?> changePassword(
            @RequestBody ChangePasswordRequest request,
            Principal connectedUser
    ) {
        service.changePassword(request, connectedUser);
        return ResponseEntity.ok().build();
    }

    taiz
    @ExceptionHandler(EmployeeException.class)
    public ResponseEntity<?> handleCustomException(EmployeeException ex) {
        return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(ex.getMessage());
    }
}
```

Hình 27. Đoạn code EmployeeController.

Ảnh trên mô tả một lớp Controller được thiết kế để xử lý các yêu cầu liên quan đến việc thay đổi mật khẩu của nhân viên. Dưới đây là phân tích chi tiết:

- Lớp EmployeeController: Đây là một lớp Controller được đánh dấu bằng annotation @RestController, cho thấy nó là một thành phần trong kiến trúc RESTful của ứng dụng. Annotation @RequestMapping("/api/v1/change-password") áp đặt một tuyến đường cơ bản mà tất cả các phương thức trong lớp này sẽ sử dụng.
- Constructor Injection: Annotation @RequiredArgsConstructor từ Lombok tự động tạo ra một constructor với các tham số cho tất cả các trường final. Điều này giúp inject EmployeeService vào EmployeeController mà không cần viết code constructor thủ công.
- Phương Thức changePassword: Được đánh dấu bằng @PatchMapping, phương thức này xử lý các yêu cầu HTTP PATCH đến đường dẫn "/change-password". Nó nhận vào đối tượng ChangePasswordRequest và thông tin xác thực Principal từ yêu cầu HTTP, sau đó gọi service.changePassword để xử lý thay đổi mật khẩu. Nếu thay đổi thành công, phương thức trả về một ResponseEntity với trạng thái HTTP 200 OK.
- Exception Handling: Phương thức handleCustomException sử dụng annotation @ExceptionHandler để xác định cách xử lý ngoại lệ EmployeeException. Nếu ngoại lệ này được ném ra từ service.changePassword, phương thức này sẽ xử lý và trả về phản hồi với mã trạng thái HTTP 400 BAD REQUEST cùng với thông điệp lỗi từ ngoại lệ.

4.5 Triển khai

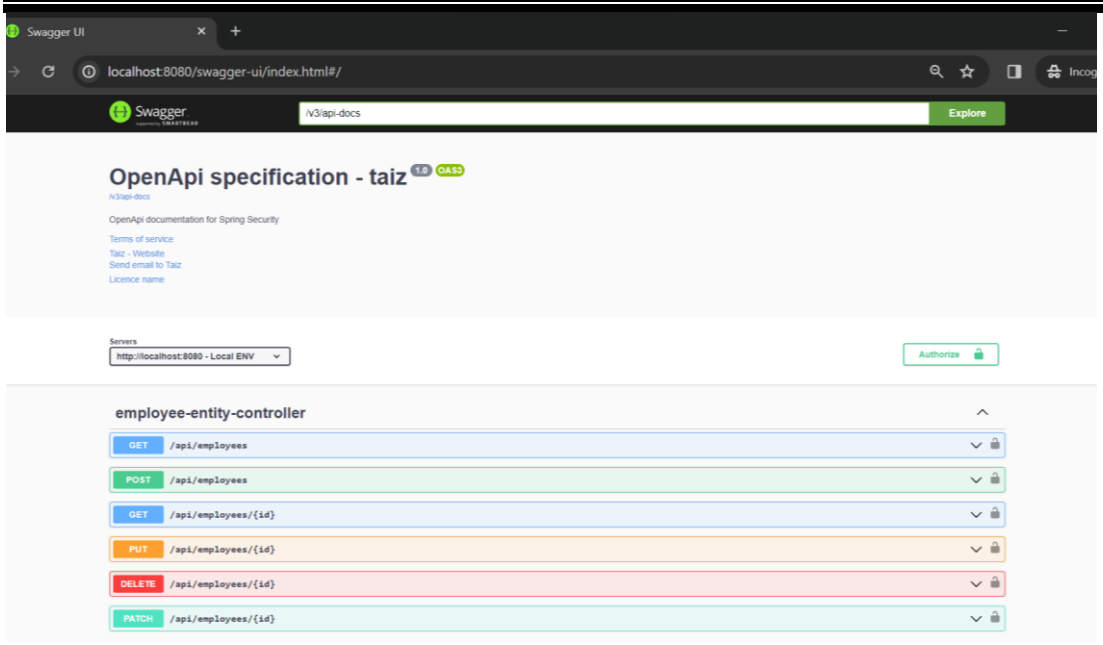
Dự án được triển khi trên Spring Boot 3.1.4 và jdk 21.

```
com.lntai.coffee.CoffeeApplication       : Started CoffeeApplication in 9.25 seconds (process running for

o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet        : Completed initialization in 3 ms
o.springdoc.api.AbstractOpenApiResource  : Init duration for springdoc-openapi is: 1193 ms
```

Hình 28. Đoạn log hiện thị khi chạy dự án.

Ứng dụng Spring Boot của tôi có tên là CoffeeApplication đã khởi động thành công trong vòng 9.25 giây. DispatcherServlet, phần tử quan trọng trong Spring MVC, cũng đã sẵn sàng để điều hướng các yêu cầu đến các controllers. Thêm vào đó, tài liệu API tự động được tạo ra thông qua springdoc-openapi chỉ trong hơn 1 giây.



Hình 29. Giao diện swagger-ui.

Đây là nơi hiển thị cũng như các API và cũng là nơi tương tác.Nó sử dụng các thông tin từ các annotations của Spring như @RestController và @RequestMapping để tạo ra tài liệu OpenAPI một cách tự động.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết quả dạt được

Đề tài đã khảo sát các khái niệm cơ bản của RESTful API và cách sử dụng Java SpringBoot để thiết kế và triển khai một API cho module backend của ứng dụng Quản lý Quán Cafe. Tôi đã tạo ra các endpoint để thực hiện các thao tác CRUD trên các đối tượng như sản phẩm, đơn hàng, khách hàng và nhân viên. Kiểm tra và kiểm thử các endpoint bằng công cụ Postman hoặc Swagger. Tôi cũng đã áp dụng các kỹ thuật như phân trang, bảo mật và kiểm thử để nâng cao chất lượng và hiệu suất của API. Bằng cách sử dụng RESTful API, có thể tạo ra một module backend linh hoạt, mở rộng và dễ tích hợp với các module frontend khác nhau.

5.2 Hạn chế

Nhiên cứu này cũng chỉ ra một số hạn chế và khuyến nghị của việc xây dựng module backend cho ứng dụng Quản lý Quán Cafe bằng Java SpringBoot, như sau:

- Tính năng chưa hoàn thiện: ứng dụng chỉ hỗ trợ một số chức năng cơ bản như đặt bàn, thanh toán, kiểm tra doanh thu, chưa có tính năng quản lý nhân viên, quản lý kho hàng, quản lý khuyến mãi, etc. Cần phát triển thêm các tính năng này để tăng khả năng ứng dụng hóa của ứng dụng.
- Bảo mật chưa cao: ứng dụng chưa có tính năng phân quyền người dùng, cho phép bất kỳ ai cũng có thể truy cập vào module backend qua RESTful API.
- Còn một số lỗi: ứng dụng có thể gặp phải một số lỗi khi xử lý các yêu cầu HTTP từ module frontend. Cần kiểm tra và sửa lỗi kỹ lưỡng trước khi triển khai ứng dụng lên môi trường thực tế.

5.3 Hướng phát triển

Tích Hợp Công Nghệ Mới: Khám phá và tích hợp các công nghệ mới vào hệ thống hiện tại, như Spring WebFlux cho lập trình phản ứng, hoặc container hóa ứng dụng với Docker và Kubernetes để dễ dàng triển khai và quản lý.

Tối Ưu Hóa Hiệu Suất: Đánh giá và tối ưu hóa hiệu suất của các API, đảm bảo việc xử lý đồng thời số lượng lớn yêu cầu một cách nhanh chóng và ổn định.

Bảo Mật Cao Cấp: Tăng cường các biện pháp bảo mật, bao gồm việc cập nhật thường xuyên các phụ thuộc và sử dụng các kỹ thuật mã hóa tiên tiến hơn.

Phát Triển Dựa Trên Đám Mây: Di chuyển hệ thống backend lên môi trường đám mây để cải thiện khả năng mở rộng và độ tin cậy.

Phân Tích Dữ Liệu và AI: Áp dụng phân tích dữ liệu lớn và học máy để cung cấp thông tin chi tiết từ dữ liệu, cải thiện quyết định kinh doanh và tạo trải nghiệm cá nhân hóa cho khách hàng.

Giao Diện Người Dùng và UX: Phát triển và cải tiến giao diện người dùng phía client để đảm bảo trải nghiệm người dùng mượt mà và trực quan.

Tích Hợp Thanh Toán Điện Tử: Nhúng các giải pháp thanh toán điện tử để cung cấp một quy trình thanh toán liền mạch cho khách hàng.

Tích Hợp Thêm Các Mô-đun: Như quản lý lịch đặt bàn, chương trình khách hàng thân thiết, và tự động hóa quản lý tồn kho.

Mỗi hướng phát triển này đều nhằm mục đích tăng cường khả năng cạnh tranh, cải thiện hiệu quả hoạt động và đảm bảo ứng dụng Quản lý Quán Cafe có thể thích ứng linh hoạt với các yêu cầu thay đổi của thị trường.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] M. Heckler, Spring Boot: Up and Running, 2021.

[2] S. Musib, Spring Boot in Practice, 2022.

[3] A. Mankale, Spring Security 3.x Cookbook, Packt, 2013.

[4] L. Spilca, Spring Security in Action, 2020.

[5] M. Knutson, Spring Security, 2017.

[6] N. Madden, API Security in Action, 2020.

[7] D. Moore, Breaking down JSON Web Tokens: From pros and cons to building and revoking, 2022.

[8] "swagger," [Online]. Available: https://swagger.io/. [Accessed 2 1 2024].

[9] T. N. N. Cương, Lý Thuyết Cơ Sở Dữ Liệu Quan Hệ Và Ứng Dụng, 2019.

[10] G. Reese, MySQL Pocket Reference, 2003.

PHỤ LỤC