**BỘ CÔNG THƯƠNG**

**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP HỒ CHÍ MINH**

----------------------

**Khoa: Công Nghệ Thông Tin**



**Báo cáo môn kiến trúc phần mềm**

**ĐỀ TÀI: Thu mua linh kiện thiết bị điện tử**

Mã học phần : 422000191403

Tên học phần : Kiến trúc và Thiết kế phần mềm

Giáo viên : Võ Văn Hải

Nhóm : 3

Thành Viên Nhóm:

|  |  |  |
| --- | --- | --- |
| **STT** | **MSSV** | **Họ và Tên** |
| 1 | 20058071 | Bùi Vạn Đăng Quang |
| 2 | 20072631 | Dương Quang Hải |
| 3 | 20073801 | Phan Minh Phú |

**Mục lục**

[CHƯƠNG 1 : GIỚI THIỆU 3](#_Toc167099675)

[1. Tổng quan 3](#_Toc167099676)

[2. Mục tiêu đề tài 3](#_Toc167099677)

[3. Phạm vi đề tài 4](#_Toc167099678)

[1. Chức năng dành cho người dùng: 4](#_Toc167099679)

[*2.* Chức năng dành cho người kỹ thuật viên*:* 4](#_Toc167099680)

[*3.* Chức năng dành cho người kế toán: 5](#_Toc167099681)

[4. Mô tả yêu cầu chức năng 5](#_Toc167099682)

[1. Phân tích yêu cầu của hệ thống 5](#_Toc167099683)

[*2.* Mô tả hoạt động của hệ thống 6](#_Toc167099684)

[5. CƠ SỞ LÝ THUYẾT 7](#_Toc167099685)

[1. Spring Boot Java 7](#_Toc167099686)

[2. MongoDB 9](#_Toc167099687)

[3. Kiến trúc microservice: 11](#_Toc167099688)

[5.3.1. Kiến trúc Microservices: 11](#_Toc167099689)

[5.3.2. Lợi ích khi sử dụng microservices 12](#_Toc167099690)

[5.3.3. Khó khăn 13](#_Toc167099691)

[**5.3.4.** **Một số Khía cạnh Khác:** 14](#_Toc167099692)

[*6.* *Trường hợp nên dùng kiến trúc phân lớp* 15](#_Toc167099693)

[7. Trường hợp không nên dùng kiến trúc phân lớp 16](#_Toc167099694)

[8. Khả năng kiểm thử 17](#_Toc167099695)

[9. Hiệu năng 18](#_Toc167099696)

[10. Khả năng mở rộng 18](#_Toc167099697)

[11. Dễ phát triển 19](#_Toc167099698)

[CHƯƠNG 2 : PHÂN TÍCH VÀ THIẾT KẾ 21](#_Toc167099699)

[1. Phân tích yêu cầu bằng UML 21](#_Toc167099700)

[1.1. Usecase tổng quát 21](#_Toc167099701)

[1.2. Danh sách tác nhân và mô tả 21](#_Toc167099702)

[1.3. Danh sách các tình huống hoạt động (Use cases) 22](#_Toc167099703)

[2. Tình huống hoạt động 23](#_Toc167099704)

[2.1. UC01\_Đăng nhập 23](#_Toc167099705)

[2.2. UC02\_Quên mật khẩu 25](#_Toc167099706)

[2.3. UC03\_Tạo giao dịch mới 27](#_Toc167099707)

[2.4. UC04\_Đánh giá sản phẩm 28](#_Toc167099708)

[2.5. UC05\_Chấp nhận thoả thuận giá cả 29](#_Toc167099709)

[2.6. UC06\_Phân loại thiết bị 30](#_Toc167099710)

[2.7. UC07 \_Thanh toán 32](#_Toc167099711)

[3. Class diagram 34](#_Toc167099712)

[3.1. Chức năng thiết kế theo Kiến trúc *Microservices:* 34](#_Toc167099713)

[- **Quản lý khách hàng (Customers):** 34](#_Toc167099714)

[- **Quản lý thiết bị (Devices):** 35](#_Toc167099715)

[- **Quản lý giao dịch (Transactions):** 35](#_Toc167099716)

[- **Báo cáo (Reports):** 36](#_Toc167099717)

[- **Tìm kiếm và lọc dữ liệu:** 36](#_Toc167099718)

[- **Quản lý người dùng và quyền truy cập:** 36](#_Toc167099719)

[4. : HIỆN THỰC 37](#_Toc167099720)

[4.1. Giao diện của hệ thống 37](#_Toc167099721)

[4.1.1. Giao diện web 37](#_Toc167099722)

[4.1.2. : KẾT LUẬN 38](#_Toc167099723)

[4.2. Kết quả đạt được 38](#_Toc167099724)

[4.3. Hạn chế 38](#_Toc167099725)

[4.4. Hướng phát triển 38](#_Toc167099726)

[4.5. TÀI LIỆU THAM KHẢO 39](#_Toc167099727)

# : GIỚI THIỆU

## Tổng quan

Hệ thống thu mua thiết bị điện tử này cung cấp một nền tảng trực tuyến toàn diện cho cả người dùng (khách hàng) và công ty . Nó cho phép khách hàng dễ dàng kiểm tra giá trị của các thiết bị điện tử cũ trước khi quyết định bán, sau đó gửi chúng đến công ty tái chế thông qua hệ thống. Khi nhận được thiết bị, công ty sẽ thu mua đánh giá tình trạng và giá trị, rồi thực hiện thanh toán cho khách hàng dựa trên kết quả đánh giá này. Khách hàng có thể theo dõi tình trạng thiết bị từ lúc nhận đến khi hoàn tất quá trình mua bán. Sau đó, công ty sẽ xử lý và tái chế các thiết bị này. Hệ thống cung cấp các báo cáo tài chính và hoạt động để giúp công ty quản lý và theo dõi hiệu quả hoạt động. Kiến trúc microservice được sử dụng để triển khai hệ thống, đảm bảo tính linh hoạt, mở rộng và dễ bảo trì.

## Mục tiêu đề tài

Hệ thống thu mua linh kiện điện tử cũ này nhằm mục tiêu tối ưu hóa quy trình thu mua, phân loại và tái bán các linh kiện điện tử đã qua sử dụng. Mục đích chính là thu mua các linh kiện này từ người dùng (khách hàng) với giá tốt nhất, sau đó phân loại, kiểm tra và đánh giá chất lượng. Các linh kiện đạt yêu cầu sẽ được bán lại cho các công ty sản xuất, sửa chữa hoặc tái sử dụng với giá cạnh tranh. Hệ thống cung cấp một kênh giao dịch trực tuyến tiện lợi và minh bạch giữa người bán và công ty thu mua, nhằm tăng cường tái sử dụng linh kiện, giảm thiểu lượng rác thải điện tử, đồng thời mang lại lợi ích cho cả hai bên tham gia. Bằng cách tối ưu hóa quy trình này, hệ thống nhằm tăng lợi nhuận và hiệu quả hoạt động.Với nền tảng website người dùng có thể :

* Yêu cầu báo giá
* Gửi thông tin thiết bị
* Thanh toán
* Báo cáo
* Đánh giá thiết bị
* Tái chế / Bán lại
* Kiểm tra tình trạng thiết bị
* Đăng nhập

## Phạm vi đề tài

## Chức năng dành cho người dùng:

* Đăng ký tài khoản: Khách hàng có thể tạo tài khoản cá nhân trên nền tảng của hệ thống, điều này giúp quản lý giao dịch và lịch sử mua bán dễ dàng hơn.
* Gửi yêu cầu thu mua: Khách hàng có thể đăng tải danh sách các linh kiện điện tử cũ họ muốn bán, kèm theo thông tin chi tiết như tên, mã, số lượng và hình ảnh. Hệ thống sẽ phản hồi với giá thu mua dự kiến.
* Xem giá thu mua: Khách hàng có thể xem mức giá thu mua mà hệ thống đề xuất đối với các linh kiện họ muốn bán.
* Đóng gói và gửi linh kiện: Nếu chấp nhận giá thu mua, khách hàng sẽ được hướng dẫn đóng gói và gửi linh kiện an toàn đến địa chỉ của công ty.
* Theo dõi tình trạng giao dịch: Khách hàng có thể theo dõi quá trình kiểm tra và xử lý linh kiện của hệ thống, từ khi nhận hàng đến khi thanh toán.
* Nhận thanh toán: Sau khi linh kiện được kiểm tra và chấp nhận, khách hàng sẽ nhận được thanh toán theo giá đã thỏa thuận.

## **Chức năng dành cho người kỹ thuật viên***:*

* Kiểm tra và phân loại linh kiện:
  + Khi nhận được linh kiện từ khách hàng, kỹ thuật viên sẽ tiến hành kiểm tra và đánh giá chất lượng, tình trạng hoạt động của từng linh kiện.
  + Phân loại linh kiện thành các nhóm dựa trên độ hư hỏng, khả năng tái sử dụng và giá trị thu mua.
* Hủy bỏ và tái chế linh kiện:
  + Đối với các linh kiện còn sử dụng được, kỹ thuật viên sẽ thực hiện các công việc sửa chữa, làm sạch, bảo dưỡng để chuẩn bị cho việc tái bán.
  + Các linh kiện không thể sửa chữa sẽ được tái chế để thu hồi các thành phần, nguyên liệu có giá trị.
* Định giá và lập kế hoạch bán hàng:
  + Dựa trên kết quả kiểm tra và phân loại, kỹ thuật viên sẽ đề xuất giá thu mua phù hợp cho từng loại linh kiện.
  + Lập kế hoạch bán hàng, tìm kiếm và liên hệ với các công ty, cơ sở sửa chữa hoặc tái sử dụng để bán các linh kiện đã được phục hồi.
* Quản lý kho và vận chuyển:
  + Quản lý kho lưu trữ các linh kiện đã được kiểm tra, phân loại và chuẩn bị để bán.
  + Tổ chức vận chuyển an toàn các linh kiện đến khách hàng mua.

## 3. Chức năng dành cho người kế toán:

- Thống kê và Báo cáo : Thống kê và báo cáo doanh thu, số lượng thu mua được trong tháng

- Thực hiện thanh toán cho khách hàng.

## Mô tả yêu cầu chức năng

## Phân tích yêu cầu của hệ thống

- Người dùng có thể thay đổi thông tin cá nhân như tên, giới tính…

- Người dùng thực hiện cung cấp thông tin hình ảnh.

- Người dùng có thể hủy, không muốn giao dịch này nếu số tiền không mong muốn.

- Từ mô tả trên, có thể đưa ra yêu cầu của hệ thống với hai đối tượng chính tương tác với hệ thống như sau:

• Đối với hệ thống:

- Quản lý các giao dịch

- Theo dõi giao dịch của cá nhân (khách hàng).

- Tìm kiếm thông tin của giao dịch

- Quản lý thông tin của người dùng

- Thống kê doanh thu

• Đối với người dùng:

* Liên hệ và thương lượng với kỹ thuật viên về việc cung cấp/bán linh kiện cũ trên hệ thống.
* Đàm phán giá cả và điều kiện mua bán.
* Cung cấp thông tin chi tiết về tình trạng và đặc điểm của linh kiện.
* Vận chuyển linh kiện đến địa điểm thu mua.
* Nhận thanh toán từ kế toán trên hệ thống.

## **Mô tả hoạt động của hệ thống** *Top of Form*

* Quản lý giao dịch:
  + Thêm, sửa đổi, hủy bỏ các giao dịch mua bán linh kiện điện tử.
  + Lưu trữ đầy đủ thông tin về từng giao dịch như: ngày tháng, loại linh kiện, số lượng, giá trị, trạng thái, người thực hiện giao dịch, giá cả.
* Theo dõi giao dịch của khách hàng:
  + Theo dõi và lưu trữ lịch sử giao dịch của từng khách hàng.
  + Cung cấp chức năng xem chi tiết các giao dịch của khách hàng.
  + Phân tích hành vi, nhu cầu của từng khách hàng dựa trên lịch sử giao dịch.
* Tìm kiếm thông tin giao dịch:
  + Cho phép tìm kiếm, lọc thông tin giao dịch theo các tiêu chí như: khách hàng, loại linh kiện, thời gian, trạng thái.
  + Cung cấp các tùy chọn để người dùng dễ dàng tìm kiếm và truy xuất thông tin giao dịch cần thiết.
* Quản lý thông tin người dùng:
  + Lưu trữ và quản lý thông tin cá nhân của khách hàng như: tên, địa chỉ, số điện thoại, email, số thẻ ngân hàng.
  + Cho phép khách hàng cập nhật, chỉnh sửa thông tin cá nhân.
  + Đảm bảo an toàn, bảo mật thông tin khách hàng.
* Thống kê doanh thu:
  + Tổng hợp, phân tích doanh thu theo các tiêu chí như: thời gian, loại linh kiện, khách hàng.
  + Cung cấp các báo cáo, thống kê về tình hình kinh doanh, hiệu quả hoạt động.

## CƠ SỞ LÝ THUYẾTTop of Form

## Spring Boot Java



Hình 2‑1 String boot Java development

**Tổng quan về Spring Boot:**

Spring Boot là một framework trong hệ sinh thái Spring, được phát triển bởi Pivotal Software. Mục đích chính của Spring Boot là đơn giản hóa quá trình phát triển các ứng dụng Spring, giúp lập trình viên nhanh chóng xây dựng và chạy các ứng dụng độc lập, ứng dụng microservices. Một trong những đặc điểm nổi bật của Spring Boot là khả năng tự động cấu hình, giúp lập trình viên không phải cấu hình nhiều. Ngoài ra, Spring Boot còn tích hợp sẵn các web servers như Tomcat, Jetty, Undertow, cho phép chạy ứng dụng web độc lập mà không cần triển khai trên các web server bên ngoài. Quản lý các thư viện, dependencies và phiên bản của chúng cũng là một ưu điểm của Spring Boot, giúp lập trình viên tập trung vào phát triển logic nghiệp vụ. Bên cạnh đó, Spring Boot cung cấp các tính năng như Actuator để theo dõi và quản lý ứng dụng, cũng như framework testing tích hợp sẵn để đơn giản hóa quá trình viết unit test và integration test. Một ưu điểm khác của Spring Boot là sự phù hợp với việc xây dựng các ứng dụng microservices, cung cấp các tính năng như Service Discovery, Load Balancing, Circuit Breaker. Ngoài ra, Spring Boot còn mang đến các tính năng sẵn có để triển khai ứng dụng vào production, như logging, external configuration, security, metrics. Nhờ những tính năng này, Spring Boot đã trở thành một framework rất phổ biến trong cộng đồng lập trình Java, giúp các lập trình viên rút ngắn đáng kể thời gian phát triển và triển khai ứng dụng.

Ưu điểm:

1. Tự động cấu hình: Spring Boot có thể tự động cấu hình ứng dụng dựa trên các thư viện và dependencies đã được thêm vào. Điều này giúp giảm đáng kể thời gian cấu hình và cấu trúc cần thiết.

2. Starter Dependencies: Spring Boot cung cấp các "starter dependencies" tiêu chuẩn, cho phép developers nhanh chóng thêm các tính năng phổ biến như web, security, data access, messaging, v.v. vào ứng dụng.

3. Embedded Servers: Spring Boot có thể chạy ứng dụng với các web server như Tomcat, Jetty hoặc Undertow một cách nội bộ, giúp việc triển khai và vận hành trở nên đơn giản hơn.

4. Spring Actuator: Actuator là một module cung cấp các tính năng quản lý và monitoring ứng dụng một cách sâu rộng, như health checks, metrics, info, dump, etc.

5. Production-ready Features: Spring Boot được xây dựng với nhiều tính năng sẵn có hướng tới môi trường production, như external configuration, logging, profile-specific properties, etc.

6. Microservices: Spring Boot rất phù hợp với việc xây dựng các ứng dụng microservices, với các tính năng như service discovery, load balancing, circuit breakers, etc.

7. Tính mở rộng và linh hoạt: Spring Boot có thể tích hợp dễ dàng với các thư viện và frameworks khác như Spring Security, Spring Data, Spring Batch, v.v. Nó cũng rất linh hoạt trong việc cấu hình và tùy biến.

Nhờ những lợi ích trên, Spring Boot đã trở thành một lựa chọn phổ biến trong cộng đồng Java, giúp developers xây dựng các ứng dụng Java hiện đại một cách nhanh chóng và hiệu quả hơn.

## MongoDB - AgileSparksMongoDB

SQL (Structured Query Language) cũng có thể được sử dụng để xây dựng và quản lý các microservices, không chỉ riêng MongoDB. Dưới đây là một số cách SQL có thể được tích hợp vào kiến trúc microservices:

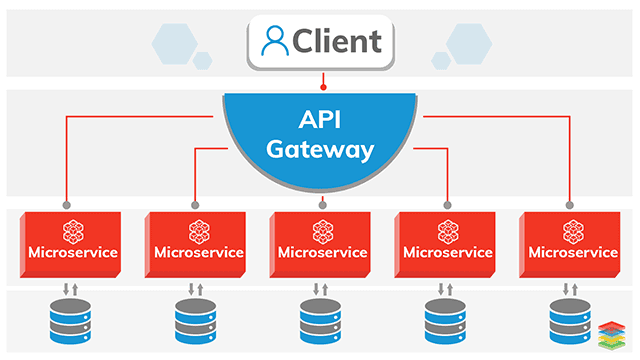
**Sử dụng SQL Database trong Microservices:**

* Spring Data MongoDB:
  + Spring Boot tích hợp sẵn Spring Data MongoDB, một module của Spring Data, giúp cung cấp các tính năng như CRUD repository, query method, pagination, và các tính năng khác để làm việc với MongoDB.
* Auto-configuration:
  + Khi sử dụng Spring Boot, việc cấu hình kết nối với MongoDB được tự động hóa. Lập trình viên chỉ cần cung cấp các thông tin kết nối cơ bản như host, port, database name trong application.properties và Spring Boot sẽ tự động cấu hình và khởi tạo kết nối.
* Reactive Programming:
  + Spring Boot 2.x hỗ trợ lập trình reactive, cho phép tận dụng các ưu điểm của MongoDB như non-blocking I/O và horizontal scaling. Điều này rất phù hợp với các ứng dụng có lưu lượng truy cập lớn.
* Microservices:
  + Khi xây dựng kiến trúc microservices, MongoDB là một lựa chọn tuyệt vời để làm database, nhờ tính linh hoạt, khả năng mở rộng và phù hợp với các yêu cầu của các microservices độc lập.
* Testability:
  + Spring Boot cung cấp các công cụ để mock MongoDB trong quá trình testing, giúp đơn giản hóa việc viết unit test và integration test cho ứng dụng.

## Kiến trúc microservice:

### Kiến trúc Microservices:

* là một dạng của kiến trúc hướng dịch vụ, trong đó các ứng dụng được xây dựng như một tập hợp các dịch vụ nhỏ hơn khác nhau chứ không phải toàn bộ một ứng dụng. Thay vì ứng dụng nguyên khối, bạn có một số ứng dụng độc lập có thể tự chạy và có thể được tạo bằng các ngôn ngữ lập trình hoặc mã hóa khác nhau. Các ứng dụng lớn và phức tạp có thể được tạo thành từ các chương trình đơn giản và độc lập được thực thi bởi chính chúng. Các chương trình nhỏ hơn này được nhóm lại với nhau để cung cấp tất cả các chức năng của ứng dụng lớn, nguyên khối.



**Ngoài các service, trong kiến trúc microsercies còn có những thành phần đặc trưng khác.**

* Management/orchestration: Đây là thành phần chịu trách nhiệm đặt các service vào các node, xác định lỗi, cân bằng service giữa các node, ... Thông thường, thành phần này là một công nghệ có sẵn như Kubernetes chứ ko phải là một thứ chúng ta tự xây dựng.
* API Gateway: API gateway là cổng truy cập cho client. Thay vì gọi trực tiếp service, client gọi API gateway sau đó API gateway sẽ gọi đến các service thích hợp.
  + Ưu điểm khi sử dụng API gateway:
  + Tách client khỏi service. Các service có thể nâng cấp lên phiên bản mới hoặc tái cấu trúc mà không cần cập nhật tât cả client.
  + Các service có thể sử dụng các giao tiếp protocols như là AMQP.
  + API gateway có thể thực hiện liên tục các chức năng như xác thực, ghi nhật kí, kết thúc SSL và cân bằng tải.

### Lợi ích khi sử dụng microservices

* Agility: Vì các service triển khai đôc lập nên có thể dễ dàng sửa lỗi và phát hành tính năng mới. Bạn có thể cập nhật service mà không cần triển khai lại toàn bộ ứng dụng và có thể khôi phục bản cập nhật trước đó nếu có lỗi xảy ra. Trong cách triển khai truyền thống, nếu có lỗi nó có thể chặn toàn bộ quá. trình triển khai ứng dụng. Các tính năng mới có thể được giữ lại để chờ bản sửa lỗi được tích hợp, kiểm thử và phát hành.
* Small, focused teams: Một microservice phải đủ nhỏ để một team có thể xây dựng, kiểm thử và triển khai nó. Quy mô team nhỏ nên có thể phát triển nhanh hơn. Các team lớn thường kém năng suất hơn, vì giao tiếp chậm hơn, chi phí quản lý cao nên giảm đi sự nhanh nhẹn.
* Small code base: Trong một ứng dụng monolithic, code ngày càng phụ thuộc vào nhau và trên nên rối hơn. Mỗi khi thêm tính năng mới sẽ phải sửa code ở nhiều nơi. Bằng cách không chia sẻ code hoặc data, kiến trúc microservices giảm thiểu sự phụ thuộc, giúp việc triển khai tính năng mới trở nên dễ dàng hơn.
* Mix of technologies: Có thể chọn công nghệ phù hợp nhất hoặc sử dụng kết hợp nhiều công nghệ khác để triển khai một service;.
* Fault isolation: Nếu một microservice không hoạt động, nó cũng sẽ không làm gián đoạn ứng dụng, miễn là các microservice được thiết kế để xử lý lỗi một cách chính xác.
* Scalability: Các service có thể mở rộng một cách độc lập, cho phép bạn mở rộng các hệ thống con yêu cầu nhiều tài nguyên hơn mà không cần mở rộng toàn bộ ứng dụng. Sử dụng Kubernetes hoặc Service Fabric, bạn có thể đóng gói mật độ service cao hơn vào trong một máy chủ duy nhất, điều này cho phép sử dụng tài nguyên hiệu quả hơn.
* Data isolation: Cập nhật kho dữ liệu dễ dàng hơn vì chỉ có một service nhỏ bị ảnh hưởng. Trong ứng dụng monolithic, cập nhật dữ liệu rất khó khăn, vì các phần khác nhau của ứng dụng có thể đang dùng chung một kho dữ liệu, bất kì thay đổi nào đều có thể xảy ra rủi ro.

### Khó khăn

* Complexity: Một ứng dụng microservices có thể có nhiều thành phần hơn so với ứng dụng monolithic tương đương. Mỗi service thì đơn giản hơn nhưng toàn bộ hệ thống thì phức tạp hơn.
* Development and testing: Việc viết một service nhỏ dựa trên các service phụ thuộc khác yêu cầu một cách tiếp cận khác so với monolithic hoặc layered application. Các công cụ hiện tại không phải lúc nào cũng được thiết kế để hoạt động với các phụ thuộc service. Việc tái cấu trúc qua lại giữa các service có thể khó khăn. Việc kiểm tra sự phụ thuộc của các service cũng là một thách thức, đặc biệt là khi ứng dụng đang phát triển nhanh chóng.
* Lack of governace: Cách tiếp cận phi tập trung để xây dựng microservices có những ưu điểm, nhưng nó cũng có thể dẫn đến nhiều vấn đề. Nếu sử dụng nhiều ngôn ngữ và framework khác nhau có thể khiến ứng dụng trở nên khó duy trì.
* Network congrestion and latency: Tắc nghẽn mạng và độ trễ. Việc sử dụng nhiều microservice có thể dẫn đến giao tiếp giữ các service nhiều hơn. Ngoài ra, nếu các service gọi qua lại lẫn nhau nhiều lần độ trễ bổ sung có thể trở thành một vấn đề. Bạn sẽ cần thiết kế API một cách cẩn thận
* Data integrity: Mỗi microservice chịu trách nhiệm với dữ liệu của chính nó. Do đó, tính thống nhất của dữ liệu có thể là một thách thức.
* Management: Để thành công với microservices đòi hỏi một nền tảng DevOps lớn. Việc log các hoạt động liên quan đến service có thể là một thách thức. Thông thường việc log các hoạt động phải gọi đến nhiều service cho một thao tác của người dùng.
* Versioning: Các bản cập nhật service không được phá vỡ các service phụ thuộc vào nó. Nhiều service có thể cập nhật bất kỳ lúc nào, vì vậy nếu không thiết kế cẩn thận bạn có thể gặp vấn đề về khả năng tương thích ngược hoặc chuyển tiếp.
* Skill set: Microservices là những hệ thống có tính phân tán cao. Phải đánh giá cẩn thận xem team bạn có đủ kỹ năng và kinh nghiệm để triển khai nó không.
  + 1. **Một số Khía cạnh Khác:**
* Độc lập về dữ liệu:
  + Mỗi microservice quản lý riêng biệt dữ liệu của mình, sử dụng cơ sở dữ liệu phù hợp. Điều này giúp tăng tính linh hoạt và khả năng mở rộng.
* Độc lập về công nghệ:
  + Các microservice có thể sử dụng các công nghệ, ngôn ngữ lập trình khác nhau, phù hợp với từng dịch vụ. Điều này giúp giảm sự phụ thuộc vào một công nghệ duy nhất.
* Tự động hóa:
  + Cấu trúc microservices thường được thiết kế để tự động hóa các quá trình như build, test, deploy và giám sát. Điều này giúp tăng tốc độ và độ tin cậy của quá trình phát triển.
* Khả năng chịu lỗi:
  + Nếu một microservice gặp sự cố, nó sẽ không ảnh hưởng đến toàn bộ ứng dụng. Mỗi microservice có thể có các cơ chế cách ly lỗi và nâng cao khả năng chịu lỗi của hệ thống.
* Quản lý phân tán:
  + Với nhiều microservice độc lập, việc quản lý, triển khai và vận hành trở nên phân tán, đòi hỏi các công cụ và quy trình quản lý phù hợp.
* Hợp tác và giao tiếp:
  + Do tính chất phân tán, việc hợp tác và giao tiếp giữa các nhóm phát triển trở nên quan trọng hơn, cần có các quy chuẩn và phương thức giao tiếp rõ ràng.
  + Những khía cạnh này đều góp phần tạo nên những lợi ích của cấu trúc microservices như linh hoạt, mở rộng, khả năng chịu lỗi cao và giảm sự phụ thuộc vào công nghệ.

## *Trường hợp nên dùng kiến trúc phân lớp*

* Ứng dụng có quy mô lớn, phức tạp và yêu cầu tính mở rộng cao:
  + Microservices phù hợp với ứng dụng có nhiều tính năng và cần được mở rộng theo từng phần.
  + Chia nhỏ ứng dụng thành các microservices độc lập giúp tăng khả năng mở rộng.
* Ứng dụng cần được phát triển và triển khai liên tục, với các đội phát triển độc lập:
  + Microservices cho phép các đội phát triển làm việc độc lập, dễ dàng tích hợp và triển khai.
  + Việc phát triển và triển khai riêng biệt các dịch vụ giúp tăng tốc độ và linh hoạt.
* Ứng dụng có nhiều tính năng khác nhau, không tightly coupled với nhau:
  + Microservices phù hợp khi các tính năng trong ứng dụng không phụ thuộc chặt chẽ vào nhau.
  + Chia tách các tính năng độc lập giúp dễ dàng phát triển, bảo trì và nâng cấp.
* Ứng dụng cần tính linh hoạt cao, khả năng thích ứng với thay đổi:
  + Microservices cho phép dễ dàng thay đổi, cập nhật từng dịch vụ mà không ảnh hưởng đến toàn bộ ứng dụng.
  + Kiến trúc linh hoạt, khả năng thích ứng với các yêu cầu mới.
* Ứng dụng cần khả năng chịu lỗi cao, ít ảnh hưởng khi một phần bị lỗi:
  + Nếu một microservice bị lỗi, nó sẽ không ảnh hưởng đến toàn bộ ứng dụng.
  + Cách ly lỗi giữa các dịch vụ giúp tăng khả năng chịu lỗi của hệ thống.

## Trường hợp không nên dùng kiến trúc phân lớp

* Ứng dụng nhỏ, đơn giản và không yêu cầu tính mở rộng:
  + Với ứng dụng nhỏ và đơn giản, kiến trúc monolithic có thể đáp ứng đủ.
  + Việc chia tách thành microservices có thể gây ra sự phức tạp không cần thiết.
* Ứng dụng có các tính năng chặt chẽ, tightly coupled với nhau:
  + Nếu các tính năng trong ứng dụng phụ thuộc chặt chẽ vào nhau, việc chia tách thành microservices sẽ gặp nhiều khó khăn.
  + Kiến trúc monolithic có thể phù hợp hơn.
* Ứng dụng có thời gian phát triển và triển khai ngắn:
  + Với thời gian phát triển và triển khai ngắn, việc áp dụng microservices có thể làm tăng thêm sự phức tạp và thời gian.
  + Kiến trúc monolithic có thể nhanh chóng triển khai hơn.
* Ứng dụng không yêu cầu tính linh hoạt hoặc khả năng chịu lỗi cao:
  + Nếu ứng dụng không yêu cầu tính linh hoạt và khả năng chịu lỗi cao, kiến trúc monolithic có thể đáp ứng đủ.
  + Việc áp dụng microservices có thể gây ra sự phức tạp không cần thiết.
* Ứng dụng có nguồn lực phát triển và vận hành hạn chế:
  + Áp dụng microservices đòi hỏi các nguồn lực, công cụ và quy trình quản lý phức tạp hơn.
  + Với nguồn lực hạn chế, kiến trúc monolithic có thể là lựa chọn phù hợp hơn.

## Khả năng kiểm thử

* Kiểm thử đơn vị (unit testing):
  + Với microservices, việc kiểm thử đơn vị từng dịch vụ độc lập trở nên dễ dàng hơn.
  + Mỗi microservice có phạm vi nhỏ, đơn giản hơn để kiểm thử đúng chức năng.
* Kiểm thử tích hợp (integration testing):
  + Khi thay đổi một microservice, chỉ cần kiểm thử lại các dịch vụ liên quan, thay vì phải kiểm thử toàn bộ ứng dụng.
  + Phạm vi kiểm thử tích hợp nhỏ hơn, dễ dàng hơn.
* Kiểm thử end-to-end (E2E):
  + Với microservices, việc triển khai và kiểm thử từng dịch vụ độc lập giúp tăng tốc độ kiểm thử E2E.
  + Có thể kiểm thử từng dịch vụ riêng lẻ thay vì phải chờ triển khai toàn bộ ứng dụng.
* Độc lập hóa môi trường:
  + Mỗi microservice có thể có môi trường riêng, giúp dễ dàng tái tạo và kiểm thử trong các môi trường khác nhau.
* Nhược điểm của microservices về khả năng kiểm thử:
* Độ phức tạp tăng:
  + Việc quản lý và kiểm thử nhiều microservices độc lập có thể phức tạp hơn so với kiến trúc monolithic.
  + Cần có các công cụ, quy trình quản lý phức tạp hơn.
* Khó khăn trong kiểm thử end-to-end:
  + Với nhiều microservices, việc kiểm thử end-to-end toàn bộ ứng dụng trở nên phức tạp hơn.
  + Cần phải mô phỏng và giả lập các dịch vụ phụ thuộc.
* Yêu cầu cao về kỹ năng và nguồn lực:
  + Áp dụng microservices đòi hỏi các kỹ năng và nguồn lực quản lý, vận hành phức tạp hơn.
  + Cần các công cụ, quy trình kiểm thử phù hợp.

## Hiệu năng

* Tính mở rộng (scalability):
  + Với microservices, các dịch vụ có thể được mở rộng độc lập, giúp tăng hiệu năng của ứng dụng.
  + Có thể mở rộng các dịch vụ riêng lẻ mà không ảnh hưởng đến toàn bộ ứng dụng.
* Độc lập về công nghệ:
  + Mỗi microservice có thể sử dụng công nghệ phù hợp, giúp tối ưu hóa hiệu năng.
  + Có thể lựa chọn ngôn ngữ, framework, cơ sở dữ liệu tối ưu cho từng dịch vụ.
* Tính sẵn sàng (availability):
  + Khi một microservice bị hỏng, chỉ ảnh hưởng đến chức năng tương ứng, không làm toàn bộ ứng dụng ngừng hoạt động.
  + Có thể nâng cấp, sửa chữa từng dịch vụ độc lập mà không ảnh hưởng đến toàn bộ hệ thống.

## Khả năng mở rộng

* Tính mở rộng độc lập của các dịch vụ:
  + Với microservices, các dịch vụ riêng lẻ có thể được mở rộng một cách độc lập, không ảnh hưởng đến các dịch vụ khác.
  + Điều này cho phép ứng dụng có thể mở rộng theo nhu cầu sử dụng của từng chức năng cụ thể.
* Khả năng phân chia tài nguyên:
  + Các dịch vụ độc lập có thể sử dụng tài nguyên (CPU, RAM, v.v.) phù hợp với nhu cầu của từng dịch vụ.
  + Không phải phân bổ tài nguyên một cách chung chung cho toàn bộ ứng dụng như trong kiến trúc monolithic.
* Tính linh hoạt trong triển khai:
  + Với microservices, có thể triển khai, nâng cấp hoặc thay thế từng dịch vụ độc lập mà không ảnh hưởng đến các dịch vụ khác.
  + Điều này giúp ứng dụng có thể mở rộng và cập nhật một cách linh hoạt.
* Tính độc lập công nghệ:
  + Mỗi microservice có thể sử dụng công nghệ phù hợp với yêu cầu của chính nó.
* Điều này giúp tăng tính mở rộng và khả năng thích ứng với các công nghệ mớ

## Dễ phát triển

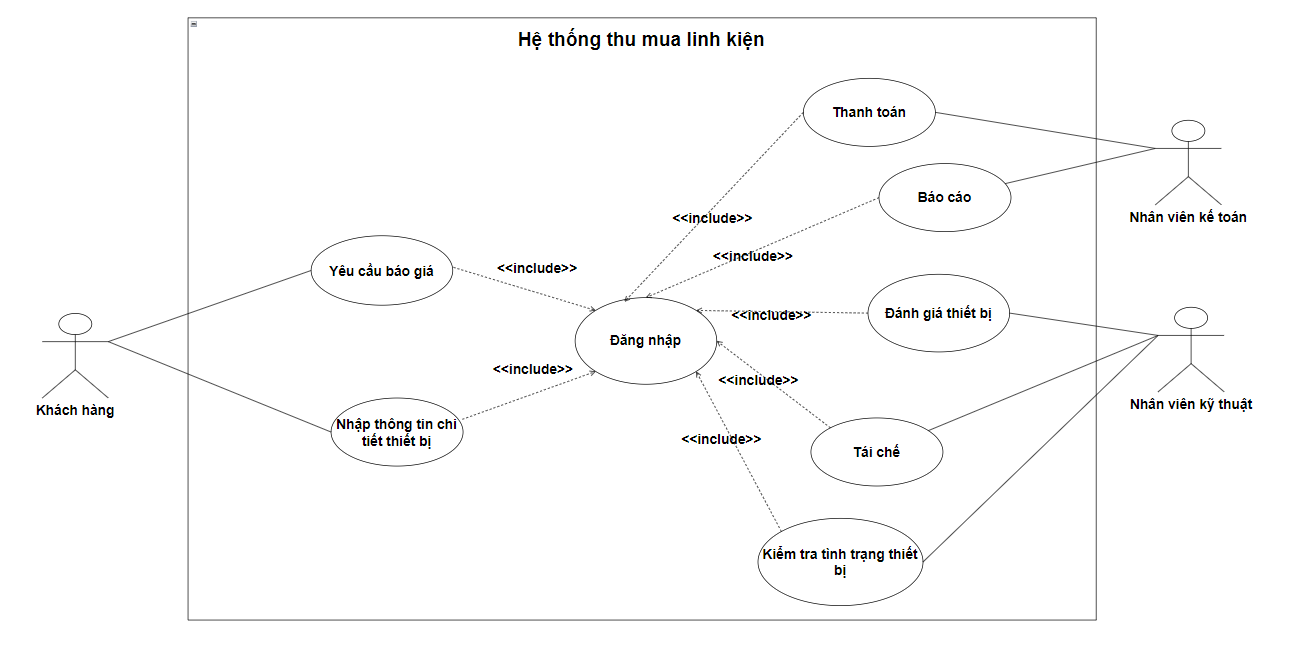
Tính dễ phát triển nhận được điểm tương đối cao, chủ yếu là do mô hình này quá nổi tiếng và không quá phức tạp để thực hiện. Bởi vì hầu hết các công ty phát triển ứng dụng bằng cách tách các microservice (trình diễn, nghiệp vụ, cơ sở dữ liệu), kiến trúc này trở thành lựa chọn tự nhiên cho hầu hết việc phát triển ứng dụng kinh doanh.Có những đặc điểm tốt:

* Phát triển và triển khai độc lập:
  + Với microservices, các nhóm phát triển có thể làm việc độc lập trên các dịch vụ riêng biệt.
  + Điều này giúp tăng tốc độ phát triển, vì các đội nhỏ hơn có thể tập trung và nhanh chóng hoàn thành các tính năng của từng dịch vụ.
* Linh hoạt trong thay đổi công nghệ:
  + Mỗi microservice có thể sử dụng công nghệ thích hợp, không bị ràng buộc với công nghệ toàn cục.
  + Khi có công nghệ mới, chỉ cần thay đổi trong phạm vi từng dịch vụ, thay vì phải thay đổi toàn bộ ứng dụng.
* Dễ dàng thử nghiệm và triển khai:
  + Với microservices, việc triển khai, thử nghiệm và phát triển các tính năng mới trở nên dễ dàng hơn.
  + Có thể triển khai từng dịch vụ một, quan sát tác động và dần dần mở rộng.
* Tái sử dụng các dịch vụ:
  + Các dịch vụ độc lập có thể được tái sử dụng giữa các ứng dụng khác nhau.
  + Điều này giúp tiết kiệm thời gian và chi phí phát triển.
* Khả năng mở rộng và thích ứng:
  + Như đã thảo luận ở trên, microservices cung cấp khả năng mở rộng tuyệt vời.
  + Điều này cho phép ứng dụng dễ dàng thích ứng với các thay đổi về nhu cầu và yêu cầu.

# : PHÂN TÍCH VÀ THIẾT KẾ

## Phân tích yêu cầu bằng UML

### Usecase tổng quát



Hình 3.3 Quản lý hệ thống thu mua

### Danh sách tác nhân và mô tả

Danh sách tác nhân và mô tả xem ở Bảng 3-1.

|  |  |
| --- | --- |
| Tác nhân | Mô tả tác nhân |
| User | * Đăng nhập * Đăng kí * Tạo giao dịch mỡi: * Thêm, sửa xóa giao dịch. * Chấp nhận thoải thuận giá cả * Đánh giá sản phẩm * Phân loại thiết bị * Tìm kiếm * Xem Thông Báo * Xem và theo dõi thông tin giao dịch. * Sửa password * Quản lí thông tin * Quản lí kỹ thuật viên * Quản lí kế toán * Quản lí thống kê * Quản lí thông tin khách hàng * Thoát |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Bảng 3‑1 Danh sách tác nhân và mô tả

### Danh sách các tình huống hoạt động (Use cases)

|  |  |
| --- | --- |
| **ID use case** | **Tên Use case** |
| **U1** | Đăng nhập |
| U2 | Thay Đổi mật khẩu |
| U3 | Tạo giao dịch mới |
| U4 | Đánh giá sản phẩm |
| U5 | Chấp nhận thoải thuận giá cả |
| U6 | Phân loại thiết bị |
| U7 | Thanh toán |

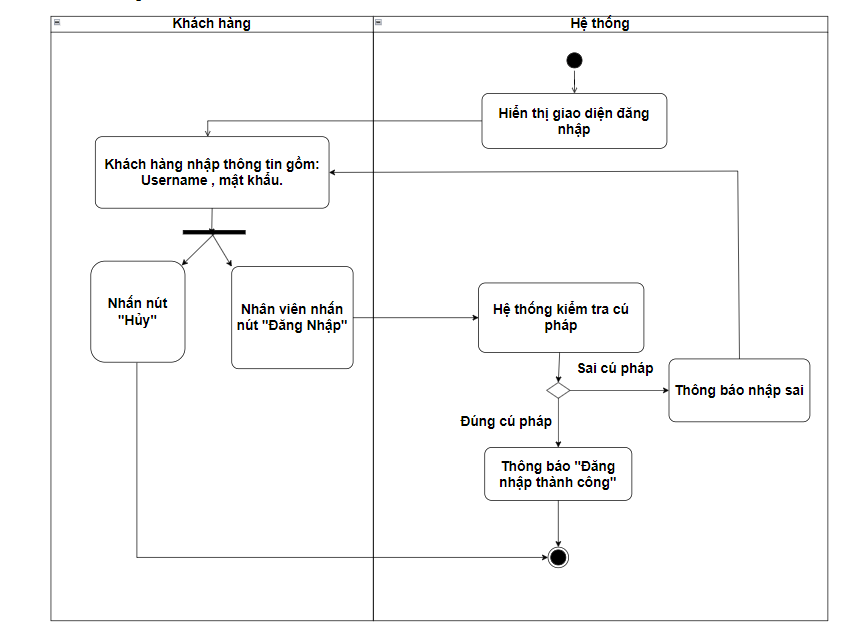
Bảng 3‑2 Danh sách các tình huống trong hệ thống

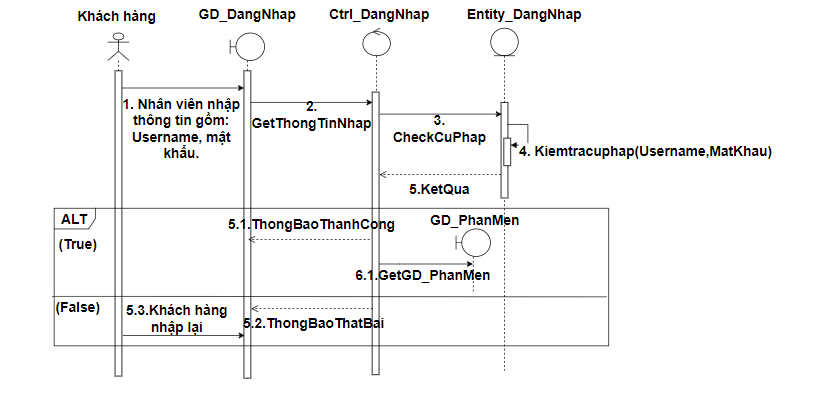
## Tình huống hoạt động

### UC01\_Đăng nhập

|  |  |
| --- | --- |
| * **Tên use case**: UC01\_Đăng nhập | |
| * **Mô tả sơ lược**: Người dùng đăng nhập thành công vào website. | |
| * **Actor chính**: Khách hàng | |
| * **Actor phụ**: không | |
| * **Tiền điều kiện (Pre-condition):** Người dùng đã có tài khoản trước đó. | |
| * **Hậu điều kiện (Post-condition):** Hệ thống thông báo đăng kí thành công. | |
| * **Luồng sự kiện chính (main flow):** | |
| **Actor** | **System** |
|  | 1. Hệ thống hiễn thị giao diện đăng nhập |
| 1. Người dùng nhập tài khoản gồm username, password. |  |
| 1. Chọn nút đăng nhập |  |
|  | 1. Hệ thống kiểm tra thông tin nhập vào |
|  | 1. Hệ thống chuyển hướng người dùng vào giao diện trang chủ. |
|  | 1. Kết thúc UC |
| * **Luồng sự kiện thay thế (alternate flow):** | |
|  | 5.1. Hệ thống thông báo thông tin tài khoản đăng nhập không đúng |
|  | 5.2. Hệ thống quay lại bước 2 |

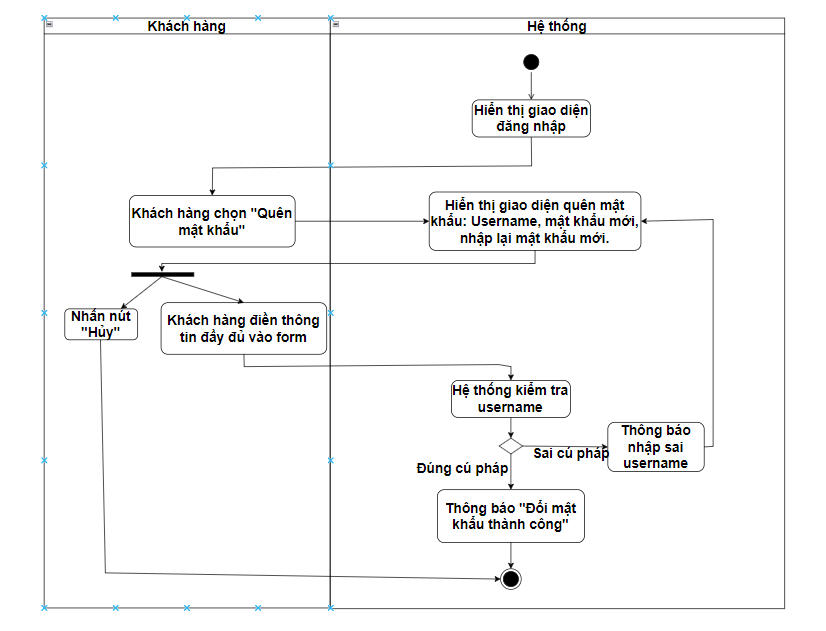
Activity diagram:



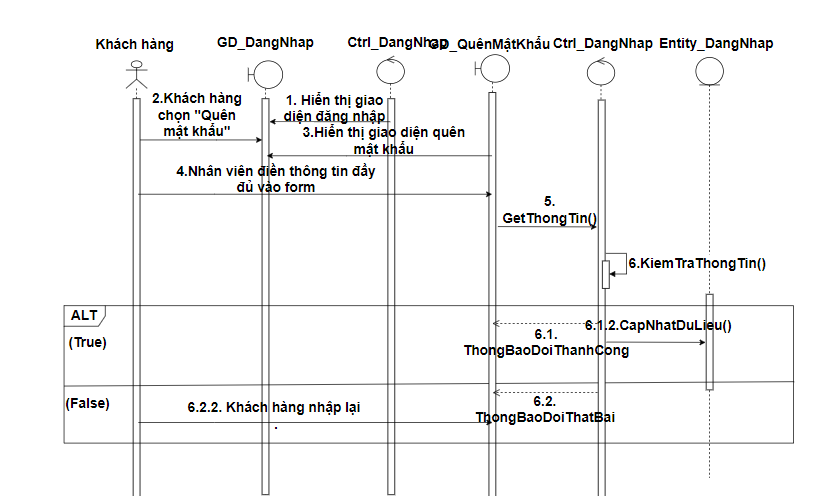
Sequence diagram:

### UC02\_Quên mật khẩu

|  |  |
| --- | --- |
| * **Tên use case**: UC03\_ Quên mật khẩu | |
| * **Mô tả sơ lược**: Người dùng có thể thay đổi mật khẩu, để vào trang chủ trong website | |
| * **Actor chính**: Người dùng | |
| * **Actor phụ**: Không | |
| * **Tiền điều kiện (Pre-condition):** Người dùng đã có tài khoản do trường cấp | |
| * **Hậu điều kiện (Post-condition):** Người dùng thay đổi mật khẩu thành công. | |
| * **Luồng sự kiện chính (main flow):** | |
| **Actor** | **System** |
| * 1. Người dùng chọn chức năng quên mật khẩu |  |
|  | * 1. Hệ thống hiễn thị form, yêu cầu nhập username. |
| * 1. Người dùng nhập vào pass cũ và mới |  |
|  | * 1. Hệ thống thực hiện kiểm tra pass cũ, xác nhận đã sửa passwork |
|  | * 1. Hệ thống thông báo thành công và quay vè màn hình đăng nhập |
|  | * 1. Kết thúc UC |
| * **Luồng sự kiện thay thế (alternate flow):** | |
|  | 4.1 Hệ thống thông báo username không hợp lệ. |
|  | 4.2 quay lại bước 1 |

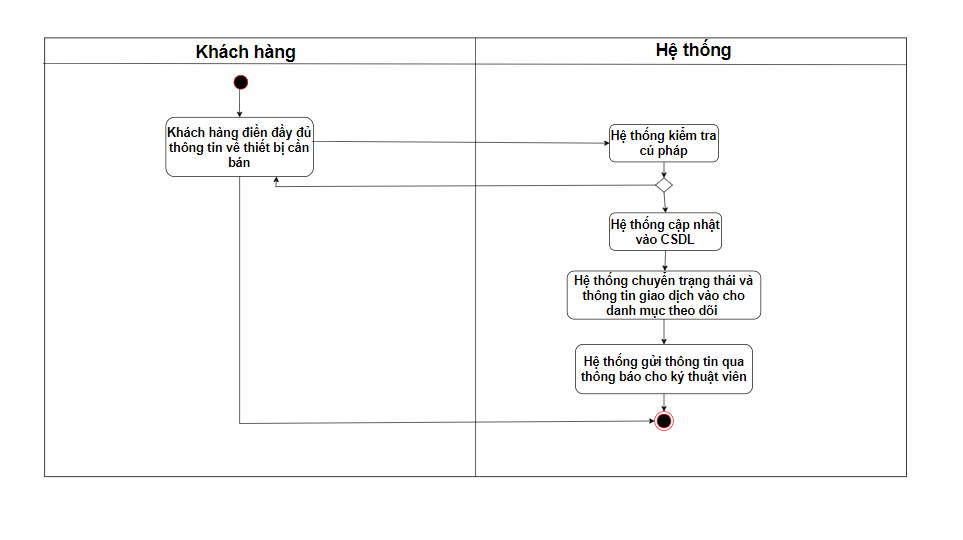


Sequence :



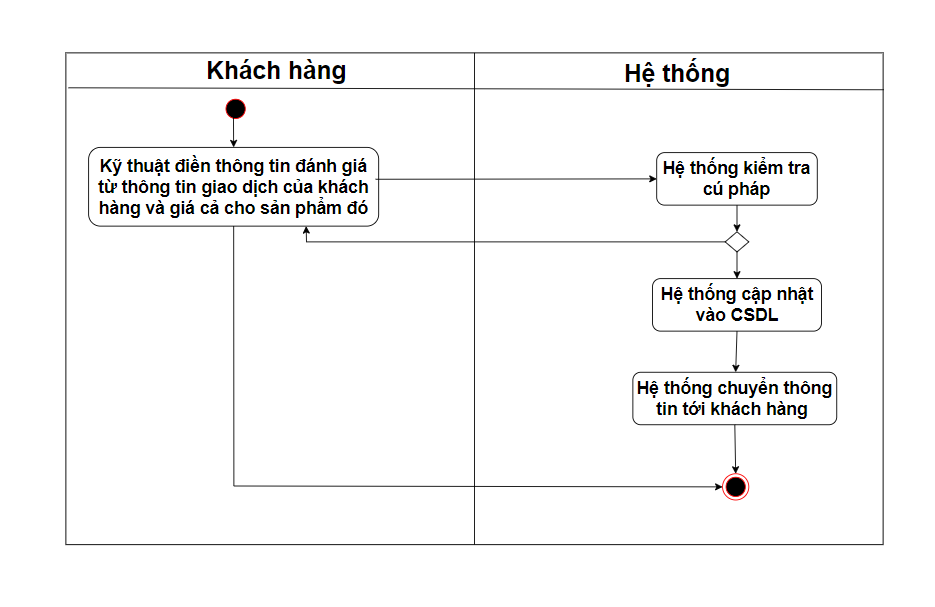
### UC03\_Tạo giao dịch mới

|  |  |
| --- | --- |
| * **Tên use case**: UC04\_ Tạo giao dịch mới | |
| * **Mô tả sơ lược**: Người dùng tạo giao dịch sản phẩm cần bán thành công. | |
| * **Actor chính**: Người dùng | |
| * **Actor phụ**: Không | |
| * **Tiền điều kiện (Pre-condition):** Người dùng chọn tạo giao dịch trên website | |
| * **Hậu điều kiện (Post-condition):** Người dùng tạo được giao dịch thành công và gửi thông tin vào CSDL | |
| * **Luồng sự kiện chính (main flow):** | |
| **Actor** | **System** |
| * 1. Khách hàng điền đầy đủ thông tin (tên sản phẩm, hình ảnh thực tế, màu, mua từ khi nào, số lượng) |  |
|  | * 1. Hệ thống kiểm tra cú pháp |
|  | * 1. Hệ thống cập nhật CSDL |
|  | * 1. Hệ thống chuyển đổi trạng thái và thông tin giao dịch đưa vào danh sách theo dõi của khách hàng. |
|  | * 1. Hệ thống gửi thông tin qua kỹ thuật viên |
|  | * 1. Kết thúc UC |
| * **Luồng sự kiện thay thế (alternate flow):** | |
|  | 2.1. Sai cú pháp |
|  | 2.2 quay lại bước 1 |



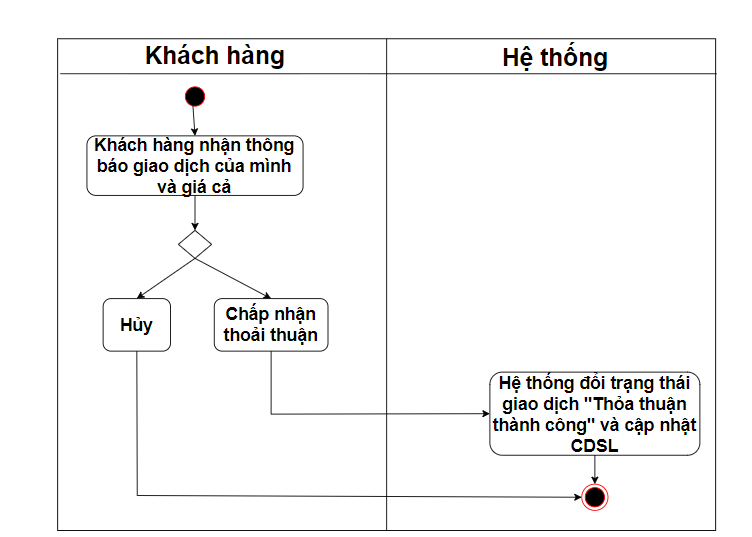
### UC04\_Đánh giá sản phẩm

|  |  |
| --- | --- |
| * **Tên use case**: UC05\_ Đánh giá sản phẩm | |
| * **Mô tả sơ lược**: Kỹ thuật viên đánh giá sản phẩm theo mức độ phần trăm của sản phẩm và đưa ra giá cả hợp lí kèm theo nhận xét cho khách hàng. | |
| * **Actor chính**: Kỹ thuật viên | |
| * **Actor phụ**: Không | |
| * **Tiền điều kiện (Pre-condition):** Kỹ thuật viên đăng nhập thành công vào web site, kỹ thuật viên chọn giao dịch cần xem. | |
| * **Hậu điều kiện (Post-condition):** Kỹ thuật viên đánh giá và đưa ra giá cả thành công. Hệ thống lưu vào CSDL và gửi thông tin tới khách hàng. | |
| * **Luồng sự kiện chính (main flow):** | |
| **Actor** | **System** |
| * 1. Kỹ thuật viên điền đầy đủ thông tin cần thiết (mức độ phần trăm của sản phẩm, giá cả hợp lí, nhận xét) |  |
|  | * 1. Hệ thống kiếm tra cú pháp |
|  | * 1. Hệ thống Cập nhập vào CSDL |
|  | * 1. Hệ thống gửi thông tin tới khách hàng, để khách hàng có thể theo dõi và thỏa thuận. |
|  | * 1. Kết thúc UC |
| * **Luồng sự kiện thay thế (alternate flow):** | |
|  | 2.1 Hệ thống thông báo sai cú pháp  2.2. Hệ thống quay lại 1 |



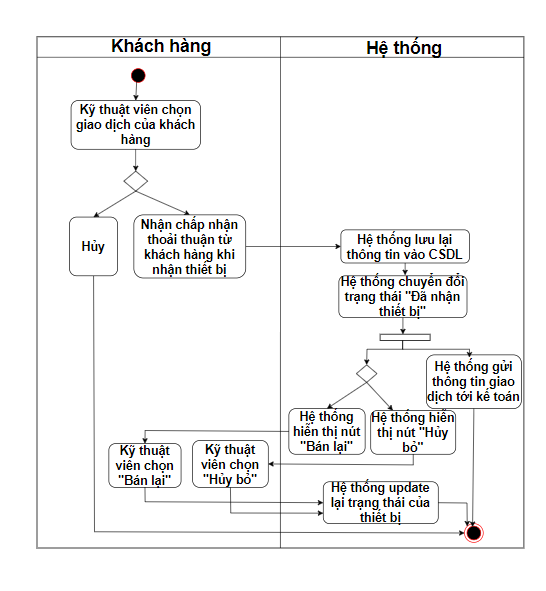
### UC05\_Chấp nhận thoả thuận giá cả

|  |  |
| --- | --- |
| * **Tên use case**: UC06\_ Chấp nhận thoả thuận giá cả | |
| * **Mô tả sơ lược**: Người dùng có thể chấp nhận thỏa thuận với yêu cầu thông tin nhận xét từ kỹ thuật viên. Và thỏa thuận thành công. | |
| * **Actor chính**: Khách hàng | |
| * **Actor phụ**: Không | |
| * **Tiền điều kiện (Pre-condition):** Người dùng đăng nhập thành công vào website, và chọn theo dõi giao dịch cần xem. | |
| * **Hậu điều kiện (Post-condition):** Người dùng chấp nhận thỏa thuận giá cả | |
| * **Luồng sự kiện chính (main flow):** | |
| **Actor** | **System** |
| * + 1. Khách hàng nhận thông báo giao dịch của mình và giá cả |  |
|  |  |
|  |  |
| * **Luồng sự kiện thay thế (alternate flow):** | |
|  | 2.1 Hệ thống thông báo chưa cập nhật điểm  2.2 Hệ thống quay lại giao diện chức năng sinh viên |



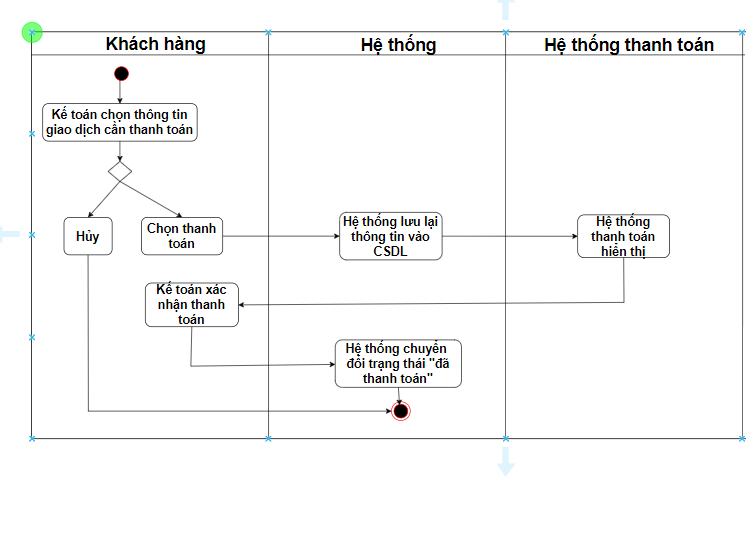
### UC06\_Phân loại thiết bị

|  |  |
| --- | --- |
| * **Tên use case**: UC07\_ Phân loại thiết bị | |
| * **Mô tả sơ lược**: Kỹ thuật viên có thể phân loại thiệt bị “bán lại” hay “tiêu hủy”. | |
| * **Actor chính**: Kỹ thuật viên | |
| * **Actor phụ**: Không | |
| * **Tiền điều kiện (Pre-condition):** Người kỹ thuật viên chọn giao dịch sản phẩm đó. | |
| * **Hậu điều kiện (Post-condition):** | |
| * **Luồng sự kiện chính (main flow):** | |
| **Actor** | **System** |
| 1. Kỹ thuật viên chọn giao dịch của khách hàng |  |
|  | 1. Nhận chấp nhận thoải thuận từ khách hàng khi nhận thiết bị |
|  | 1. Hệ thống lưu lại thông tin vào CSDL |
|  | 1. Hệ thống chuyển đổi trạng thái "Đã nhận thiết bị" |
|  | 1. Hệ thống hiễn thị nút "Bán lại" và “Hủy bỏ“ |
| 1. Kỹ thuật viên chọn chức năng |  |
|  | 1. Hệ thống update lại trạng thái của thiết bị |
|  | 1. Kết thúc UC |
|  |  |
| * **Luồng sự kiện thay thế (alternate flow):** | |
|  | 3.1. Hệ thống gửi thông tin giao dịch tới kế toán |



### UC07 \_Thanh toán

|  |  |  |  |
| --- | --- | --- | --- |
| * **Tên use case**: UC08 \_Thanh toán | | | |
| * **Mô tả sơ lược**: Hệ thống cho phép thanh toán thông qua ngân hàng. | | | |
| * **Actor chính**: Kế toán | | | |
| * **Actor phụ**: Hệ thống thanh toán | | | |
| * **Tiền điều kiện (Pre-condition):** Kế toán đăng nhập thành công. | | | |
| * **Hậu điều kiện (Post-condition):** Kế toán thanh toán thành công cho khách hàng. | | | |
| * **Luồng sự kiện chính (main flow):** | | | |
| **Actor** | **System** | | **Payment System** |
| 1. Kế toán chọn thông tin giao dịch cần thanh toán |  |  | |
|  | 1. Hệ thống lưu lại thông tin vào CSDL |  | |
| 1. Chọn thanh toán |  |  | |
|  |  | 1. Hệ thống   thanh toán hiển thị | |
| 1. Kế toán xác nhận thanh toán |  |  | |
|  | 6. Hệ thống chuyển  đổi trạng thái "đã thanh toán" |  | |
|  | 1. Kết thúc UC. |  | |
| * **Luồng sự kiện thay thế (alternate flow):** | |  | |
| 3.1 Hủy | 3.2. Kết thúc UC |  | |



## Class diagram

## Luồng chức năng thiết kế theo Kiến trúc *Microservices:*

* **Quản lý khách hàng (Customers):**
  + Tạo mới khách hàng: Cho phép thêm mới thông tin khách hàng như tên, email, số điện thoại vào cơ sở dữ liệu.
  + Cập nhật thông tin khách hàng: Cho phép chỉnh sửa thông tin của khách hàng đã có trong hệ thống.
  + Xóa khách hàng: Cho phép xóa thông tin của khách hàng khỏi cơ sở dữ liệu.
  + Xem danh sách khách hàng: Hiển thị danh sách tất cả các khách hàng đang có trong hệ thống.
* **Quản lý thiết bị (Devices):**
  + Tạo mới thiết bị: Cho phép thêm thông tin về các thiết bị điện tử đã thu thập từ khách hàng như model, nhãn hiệu, tình trạng, giá trị ước tính, v.v.
  + Cập nhật trạng thái thiết bị: Cho phép cập nhật trạng thái của thiết bị trong quá trình xử lý (đang chờ, đã nhận, đã đánh giá, đã thanh toán, đã tái chế).
  + Cập nhật thông tin thiết bị: Cho phép chỉnh sửa các thông tin liên quan đến thiết bị.
  + Xóa thiết bị: Cho phép xóa bỏ thông tin của thiết bị khỏi cơ sở dữ liệu.
  + Xem danh sách thiết bị theo khách hàng: Hiển thị danh sách các thiết bị của từng khách hàng.
* **Quản lý giao dịch (Transactions):**
  + Tạo mới giao dịch: Cho phép ghi nhận các giao dịch liên quan đến việc thu mua, thanh toán thiết bị.
  + Cập nhật trạng thái thanh toán: Cho phép cập nhật trạng thái thanh toán của giao dịch (đang chờ, đã thanh toán).
  + Xóa giao dịch: Cho phép xóa bỏ thông tin của giao dịch khỏi cơ sở dữ liệu.
  + Xem danh sách giao dịch theo khách hàng: Hiển thị danh sách các giao dịch của từng khách hàng.
* **Báo cáo (Reports):**
  + Xem danh sách các báo cáo: Hiển thị danh sách tất cả các báo cáo đã được tạo.
* **Tìm kiếm và lọc dữ liệu:**
  + Tìm kiếm khách hàng, thiết bị, giao dịch: Cho phép tìm kiếm dữ liệu dựa trên các tiêu chí như tên, model, trạng thái, v.v.
  + Lọc dữ liệu: Cho phép lọc dữ liệu theo các tiêu chí như trạng thái thiết bị, trạng thái thanh toán giao dịch.
* **Quản lý người dùng và quyền truy cập:**
  + Tạo mới người dùng: Cho phép thêm mới người dùng vào hệ thống.
  + Cập nhật vai trò và quyền truy cập: Cho phép chỉnh sửa vai trò và quyền truy cập của người dùng.
  + Xóa người dùng: Cho phép xóa người dùng khỏi hệ thống.
  + Xác thực người dùng: Đảm bảo an toàn bằng cách yêu cầu người dùng xác thực trước khi sử dụng hệ thống.

# : HIỆN THỰC

## Giao diện của hệ thống

### Giao diện web

### : KẾT LUẬN

## Kết quả đạt được

## Hạn chế

## Hướng phát triển

## TÀI LIỆU THAM KHẢO

1. Fundamentals of Software Architecture - Mark Richards
2. Software Architecture Monday - Mark Richards
3. <https://viblo.asia/p/tao-ung-dung-microservices-don-gian-voi-spring-boot-1Je5Ex2wlnL>
4. <https://spring.io/projects/spring-boot>
5. <https://www.mongodb.com/>
6. [Software Architecture Patterns](file:///C:\Users\ASUS\AppData\Roaming\Microsoft\Word\Software%20Architecture%20Patterns)
7. Geri Schneider, Jason P.Winters (foreword by Jvar Jacobson). Applying use cases: A practical guide. 2006
8. WAZLAWICK, Raul Sidnei. Object-oriented analysis and design for information systems: modeling with UML, OCL, and IFML. Elsevier, 2014.
9. Daniela Berardi , Diego Calvanese , Giuseppe De Giacomo. Reasoning on UML class diagrams, 2005
10. Marco Brambilla, Jordi Cabot, Manuel Wimmer. Model-Driven Software Engineering in Practice: Second Edition, 2010
11. Alan Dennis, Barbara Wixom, David Tegarden. Systems Analysis and Design: An Object-Oriented Approach with UML. 2014