

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



PHÁT TRIỂN ỨNG DỤNG WEB VỚI NODEJS
ĐỒ ÁN CUỐI KÌ WEBSITE ĐIỂM BÁN
HÀNG ĐIỆN THOẠI VÀ PHỤ KIỆN

Người hướng dẫn: **ThS. VŨ ĐÌNH HỒNG**

Người thực hiện: **CAO NGUYỄN BÌNH – 52000185**

NGUYỄN KHẮC NGHIÊM – 520H0557

LÊ PHẠM ANH TRÍ – 52000153

Nhóm : 3

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN**



**PHÁT TRIỂN ỨNG DỤNG WEB VỚI NODEJS
ĐỒ ÁN CUỐI KÌ WEBSITE ĐIỂM BÁN
HÀNG ĐIỆN THOẠI VÀ PHỤ KIỆN**

Người hướng dẫn: ThS. VŨ ĐÌNH HỒNG

Người thực hiện: CAO NGUYỄN BÌNH – 52000185

NGUYỄN KHẮC NGHIÊM – 520H0557

LÊ PHẠM ANH TRÍ – 52000153

Nhóm : 3

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Kính thưa thầy Vũ Đình Hồng,

Chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất vì sự hỗ trợ và sự dạy dỗ quý báu mà thầy đã mang đến cho chúng em trong suốt thời gian qua. Đặc biệt là, chúng em muốn thể hiện lòng biết ơn đặc biệt đối với những kiến thức và kỹ năng mà thầy đã truyền đạt trong môn phát triển ứng dụng web với nodejs mà chúng em học.

Thầy đã cho chúng em cái nhìn tổng quan về cách vận hành những trang web dùng nodejs, giúp chúng em nắm vững các khái niệm cơ bản và áp dụng chúng vào thực tế. Không chỉ vậy, thầy còn đã hướng dẫn sinh viên về cách tiếp cận vấn đề một cách chủ động để sinh viên không phụ thuộc vào những bài giảng có sẵn mà tự thân vận động, từ đó giúp các bạn sinh viên xây dựng những trang web cũng như ứng dụng đáng chú ý và thú vị.

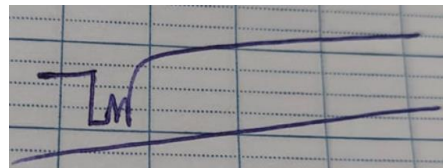
Chúng em tự hào được là học trò của thầy và hy vọng có thể tiếp tục nhận được sự hỗ trợ và chỉ dẫn từ thầy trong tương lai.

Trân trọng,

TP. Hồ Chí Minh, ngày 15 tháng 11 năm 2023

Đại diện kí tên

(ký tên và ghi rõ họ tên)



Cao Nguyên Bình

Nguyễn Khắc Nghiêm

Lê Phạm Anh Trí

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Chúng tôi xin cam đoan đây là công trình nghiên cứu của riêng chúng tôi và được sự hướng dẫn khoa học của Ths. Vũ Đình Hồng. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

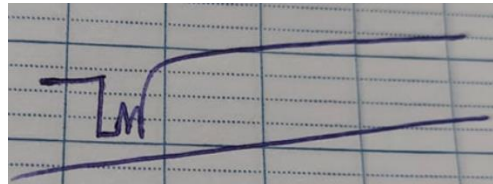
Ngoài ra, trong luận văn còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào chúng tôi xin hoàn toàn chịu trách nhiệm về nội dung luận văn của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 15 tháng 11 năm 2023

Đại diện kí tên

(ký tên và ghi rõ họ tên)



Cao Nguyên Bình

Nguyễn Khắc Nghiêm

Lê Phạm Anh Trí

TÓM TẮT

Đề tài: phát triển website quản lý bán hàng và điểm bán hàng cho cửa hàng điện thoại và phụ kiện

Mô tả:

I. Thông tin chung

Chức năng chính:

- Quản lý tài khoản: Admin và nhân viên bán hàng.
- Quản lý sản phẩm.
- Quản lý khách hàng.
- Giao dịch bán hàng.
- Báo cáo và thống kê.

II. Danh sách tính năng yêu cầu

Quản lý tài khoản:

- Tài khoản admin được tạo sẵn với thông tin đăng nhập là admin/admin.
- Admin có thể thay đổi mật khẩu bất kỳ lúc nào.
- Nhân viên bán hàng không thể tự tạo tài khoản, chỉ admin mới có quyền tạo.
- Khi tạo tài khoản cho nhân viên, admin cần cung cấp tên đầy đủ và địa chỉ Gmail. Nhân viên sẽ nhận email thông báo với đường link đăng nhập, có hiệu lực trong 1 phút.
- Sau khi đăng nhập lần đầu, nhân viên sẽ được bắt buộc đổi mật khẩu.
- Tất cả nhân viên có thể xem và cập nhật thông tin cá nhân và ảnh đại diện.

Quản lý sản phẩm (chỉ admin):

- Admin có thể thực hiện các thao tác cơ bản trên sản phẩm như xem danh sách, thêm mới, cập nhật, xóa.
- Sản phẩm có ít nhất các thông tin như mã vạch, tên sản phẩm, giá nhập, giá bán, danh mục, ngày tạo.

- Sản phẩm chỉ có thể xoá khi không thuộc bất kỳ đơn hàng nào.

Quản lý khách hàng:

- Khi thanh toán, nhân viên sẽ nhập số điện thoại khách hàng. Nếu đã mua hàng trước đó, tên và địa chỉ sẽ tự động hiển thị. Nếu mới, nhân viên cần nhập thông tin để tạo tài khoản khách hàng.
- Nhân viên có thể xem thông tin khách hàng và lịch sử mua hàng, bao gồm tổng số tiền, số tiền khách đưa, số tiền thừa, ngày mua và số lượng sản phẩm.

Xử lý giao dịch:

- Nhân viên sẽ nhập sản phẩm bằng cách tìm kiếm hoặc quét mã vạch.
- Sản phẩm sẽ hiển thị ngay trong danh sách với thông tin như số lượng, giá đơn vị, tổng tiền.
- Nhân viên có thể nhập số điện thoại khách hàng hoặc tạo mới nếu là khách mới.
- Sau khi nhập đủ thông tin, quá trình thanh toán sẽ hoàn thành và hóa đơn sẽ được in (hoặc xuất file PDF).

Báo cáo và thống kê:

- Hiển thị kết quả bán hàng theo khoảng thời gian như hôm nay, hôm qua, 7 ngày qua, tháng này hoặc khoảng thời gian cụ thể.
- Thông tin bao gồm tổng số tiền nhận, số đơn hàng, số lượng sản phẩm và danh sách đơn hàng theo thứ tự thời gian.
- Admin có thể xem thêm thông tin về lợi nhuận toàn bộ.

MỤC LỤC

LỜI CẢM ƠN.....	i
TÓM TẮT	1
MỤC LỤC.....	3
DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT	7
DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ	8
CHƯƠNG 1 – GIỚI THIỆU ĐỀ TÀI VÀ CƠ SỞ LÝ THUYẾT.....	10
1.1 Tổng quan về đề tài.....	10
1.2 Tại sao cần thực hiện đề tài.....	10
1.3 Những yêu cầu cần thực hiện:	11
1.4 Cơ sở lý thuyết:	12
1.4.1 Node.js.....	12
1.4.2 MongoDB.....	12
1.4.3 HTML, CSS, và JavaScript	12
1.4.4 Handlebars.....	12
1.4.5 JWT (JSON Web Token)	13
1.4.6 Middlewares	13
1.5 Ý nghĩa cơ sở lý thuyết:.....	13
CHƯƠNG 2 – PHÂN TÍCH THIẾT KẾ.....	14
2.1 Sơ đồ lớp.....	14
2.1.1 Vẽ sơ đồ	14
2.1.2 Mô tả sơ đồ.....	15
2.2 Sơ đồ use case	17

2.2.1 Vẽ sơ đồ.....	17
2.2.2 Mô tả sơ đồ.....	18
2.3 Sơ đồ ERD và sơ đồ quan hệ.....	21
2.3.1 Vẽ sơ đồ ERD.....	21
2.3.2 Vẽ sơ đồ quan hệ.....	21
2.3.3 Mô tả hai sơ đồ trên.....	22
2.4 Sơ đồ tuần tự.....	23
2.4.1 Sơ đồ quản lý tài khoản.....	23
2.4.2 Sơ đồ quản lý nhân viên.....	24
2.4.3 Sơ đồ quản lý sản phẩm.....	25
2.4.4 Sơ đồ quản lý khách hàng.....	26
2.4.5 Sơ đồ giao dịch bán hàng.....	27
2.4.6 Sơ đồ báo cáo và thống kê.....	28
2.5 Sơ đồ hoạt động.....	29
2.5.1 Sơ đồ quản lý tài khoản.....	29
2.5.2 Sơ đồ quản lý nhân viên.....	30
2.5.3 Sơ đồ quản lý sản phẩm.....	31
2.5.4 Sơ đồ quản lý khách hàng.....	32
2.5.5 Sơ đồ giao dịch bán hàng.....	33
2.5.6 Sơ đồ báo cáo và thống kê.....	34
CHƯƠNG 3 – HIỆN THỰC HỆ THỐNG.....	35
3.1 Chức năng chính của ứng dụng: xử lý giao dịch và chức năng báo cáo thống kê của admin.....	35
3.1.1 Code của phần OrderController.....	35

3.1.2 Giải thích code của OrderController.....	42
3.1.3 Code của phần OrderDetailController.....	45
3.1.4 Giải thích code của phần OrderDetailController	50
3.2 Chức năng gửi mail.....	51
3.2.1 Code của phần MailController.....	51
3.2.2 Giải thích code của phần MailController.....	52
3.3 Chức năng quản lý tài khoản.....	53
3.3.1 Code của phần AccountController.....	53
3.3.2 Giải thích code của phần AccountController.....	66
3.4 Chức năng quản lý khách hàng.....	70
3.4.1 Code của phần CustomerController.....	70
3.4.2 Giải thích code của phần CustomerController	74
3.5 Chức năng quản lý sản phẩm.....	76
3.5.1 Code của phần ProductController.....	76
3.5.2 Giải thích code của phần ProductController.....	86
3.6 Chức năng phụ: phân trang	89
CHƯƠNG 4 – KẾT QUẢ VÀ KẾT LUẬN	91
4.1 Màn hình giao diện của ứng dụng.....	91
4.1.1 Màn hình đăng nhập.....	91
4.1.2 Màn hình đăng ký.....	92
4.1.3 Màn hình trang chủ 1 với giao diện nhân viên.....	93
4.1.4 Màn hình trang chủ 2 với giao diện nhân viên.....	93
4.1.5 Màn hình thanh toán với giao diện nhân viên.....	94

4.1.6 Hóa đơn bán hàng với giao diện nhân viên.....	94
4.1.7 Màn hình danh sách khách hàng với giao diện nhân viên	95
4.1.8 Màn hình danh sách đơn hàng với giao diện nhân viên.....	95
4.1.9 Màn hình danh sách chi tiết đơn hàng với giao diện nhân viên	96
4.1.10 Màn hình báo cáo thống kê với giao diện nhân viên.....	96
4.1.11 Hồ sơ cá nhân của nhân viên với giao diện nhân viên.....	97
4.1.12 Màn hình quản lý sản phẩm 1 với giao diện quản lý.....	98
4.1.13 Màn hình quản lý sản phẩm 2 với giao diện quản lý.....	98
4.1.14 Màn hình quản lý tài khoản với giao diện quản lý.....	99
4.1.15 Màn hình báo cáo thống kê với giao diện quản lý	99
4.1.16 Màn hình xem chi tiết đơn hàng với giao diện quản lý	100
4.2 Kết luận	101
4.2.1 Ưu điểm của hệ thống.....	101
4.2.2 Khuyết điểm của hệ thống.....	102
4.2.3 Những điều đã làm được	102
4.2.4 Những điều chưa làm được.....	103
4.2.5 Hướng phát triển trong tương lai.....	103
TÀI LIỆU THAM KHẢO.....	104

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

CÁC KÝ HIỆU

CÁC CHỮ VIẾT TẮT

JS

JWT

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

Hình 2. 1: Sơ đồ lớp cửa hàng bán lẻ điện thoại và phụ kiện	14
Hình 2. 2: Sơ đồ use case cửa hàng bán lẻ điện thoại và phụ kiện.....	17
Hình 2. 3: Sơ đồ ERD cửa hàng bán lẻ điện thoại và phụ kiện.....	21
Hình 2. 4: Sơ đồ quan hệ các thực thể của cửa hàng bán lẻ điện thoại và phụ kiện	21
Hình 2. 5: Sơ đồ tuần tự quản lý tài khoản	23
Hình 2. 6: Sơ đồ tuần tự quản lý nhân viên.....	24
Hình 2. 7: Sơ đồ tuần tự quản lý sản phẩm.....	25
Hình 2. 8: Sơ đồ tuần tự quản lý khách hàng	26
Hình 2. 9: Sơ đồ tuần tự giao dịch bán hàng.....	27
Hình 2. 10: Sơ đồ tuần tự báo cáo và thống kê.....	28
Hình 2. 11: Sơ đồ hoạt động quản lý tài khoản	29
Hình 2. 12: Sơ đồ hoạt động quản lý nhân viên.....	30
Hình 2. 13: Sơ đồ hoạt động quản lý sản phẩm.....	31
Hình 2. 14: Sơ đồ hoạt động quản lý khách hàng	32
Hình 2. 15: Sơ đồ hoạt động giao dịch bán hàng.....	33
Hình 2. 16: Sơ đồ hoạt động báo cáo và thống kê	34
Hình 4. 1: Màn hình đăng nhập.....	91
Hình 4. 2: Màn hình đăng ký	92
Hình 4. 3: Màn hình trang chủ 1 với giao diện nhân viên.....	93
Hình 4. 4: Màn hình trang chủ 2 với giao diện nhân viên.....	93
Hình 4. 5: Màn hình thanh toán với giao diện nhân viên.....	94

Hình 4. 6: Hóa đơn thanh toán với giao diện nhân viên	94
Hình 4. 7: Màn hình danh sách khách hàng với giao diện nhân viên	95
Hình 4. 8: Màn hình danh sách đơn hàng với giao diện nhân viên	95
Hình 4. 9: Màn hình chi tiết đơn hàng với giao diện nhân viên.....	96
Hình 4. 10: Màn hình báo cáo thống kê với giao diện nhân viên.....	96
Hình 4. 11: Xem hồ sơ cá nhân của nhân viên với giao diện nhân viên	97
Hình 4. 12: Màn hình quản lý sản phẩm 1 với giao diện quản lý	98
Hình 4. 13: Màn hình quản lý sản phẩm 2 với giao diện quản lý	98
Hình 4. 14: Màn hình quản lý tài khoản với giao diện quản lý.....	99
Hình 4. 15: Màn hình báo cáo thống kê với giao diện quản lý.....	99
Hình 4. 16: Chi tiết giao diện chi tiết đơn hàng với giao diện quản lý	100

DANH MỤC BẢNG

CHƯƠNG 1 – GIỚI THIỆU ĐỀ TÀI VÀ CƠ SỞ LÝ THUYẾT

1.1 Tổng quan về đề tài

Đề tài tập trung vào phát triển một trang web quản lý bán hàng và điểm bán hàng dành cho cửa hàng điện thoại và phụ kiện. Sử dụng các công nghệ như Node.js, MongoDB, HTML, CSS và JavaScript, trang web này được thiết kế với chức năng Point of Sale (POS), hỗ trợ quản lý tài khoản, sản phẩm, khách hàng, giao dịch bán hàng, cùng với khả năng xem báo cáo và thống kê.

1.2 Tại sao cần thực hiện đề tài

- Tối ưu hóa quy trình kinh doanh: Việc triển khai một hệ thống POS sẽ giúp tối ưu hóa và tự động hóa quy trình bán hàng, giảm thiểu sai sót và tăng cường hiệu suất làm việc.
- Nâng cao trải nghiệm mua hàng của khách hàng: Quản lý thông tin khách hàng và lịch sử mua hàng sẽ giúp tạo ra trải nghiệm mua sắm cá nhân hóa, tăng cơ hội giữ chân khách hàng và tăng doanh số bán hàng.
- Quản lý hiệu quả nhân sự: Hệ thống tài khoản và quản lý nhân viên sẽ giúp quản trị viên theo dõi và kiểm soát hoạt động của nhân viên, đồng thời tạo điều kiện cho sự chuyên nghiệp hóa trong công việc.
- Phân tích thông tin đa chiều: Bảng báo cáo và thống kê sẽ cung cấp thông tin chi tiết về doanh số bán hàng, lợi nhuận, và xu hướng tiêu dùng, giúp quản lý đưa ra quyết định chiến lược dựa trên dữ liệu phân tích.
- Hiện đại hóa hệ thống bán lẻ: Sử dụng Node.js cho backend giúp ứng dụng chạy mạnh mẽ và hiệu quả, đồng thời có khả năng mở rộng tốt, phù hợp với yêu cầu của cửa hàng điện thoại.
- Cơ sở dữ liệu hiệu quả: MongoDB, một cơ sở dữ liệu NoSQL, được tích hợp để lưu trữ dữ liệu linh hoạt và phản hồi nhanh chóng, giúp quản lý sản phẩm và giao dịch một cách hiệu quả.

- Trải nghiệm người dùng tốt: Sử dụng HTML, CSS, và JavaScript để xây dựng giao diện người dùng thân thiện, tương tác, giúp nhân viên bán hàng dễ dàng thao tác và nâng cao trải nghiệm người dùng.

1.3 Những yêu cầu cần thực hiện:

Quản lý tài khoản:

- Xây dựng tài khoản admin với khả năng thay đổi mật khẩu và quản lý nhân viên.
- Tạo tài khoản cho nhân viên thông qua Node.js, với quy trình xác nhận qua email.
- Node.js sẽ hỗ trợ việc tạo và quản lý các tài khoản, đồng thời đảm bảo tính bảo mật.

Quản lý sản phẩm (Node.js và MongoDB):

- Node.js sẽ xử lý logic quản lý sản phẩm, bao gồm thêm, sửa, xóa sản phẩm.
- MongoDB sẽ lưu trữ thông tin sản phẩm và đảm bảo tính toàn vẹn dữ liệu.

Quản lý khách hàng (HTML, CSS, JS):

- Giao diện khách hàng sẽ được xây dựng bằng HTML, CSS và JavaScript để thuận tiện và dễ sử dụng.
- Node.js sẽ xử lý việc tìm kiếm và hiển thị thông tin khách hàng.

Xử lý giao dịch (Node.js, HTML, CSS, JS):

- Node.js sẽ thực hiện logic xử lý giao dịch khi thêm sản phẩm vào giỏ hàng.
- Giao diện thanh toán sẽ được xây dựng bằng HTML, CSS và JavaScript để đảm bảo tính tương tác và trực quan.

Báo cáo và thống kê (Node.js):

- Node.js sẽ thực hiện việc truy vấn cơ sở dữ liệu để lấy thông tin cho báo cáo và thống kê.

- Dữ liệu sẽ được hiển thị trên giao diện người dùng bằng HTML, CSS, và JavaScript.

Thực hiện đầy đủ các yêu cầu này giúp xây dựng một ứng dụng quản lý bán hàng toàn diện và hiệu quả sử dụng công nghệ đa dạng để đáp ứng nhu cầu của cửa hàng điện thoại và phụ kiện.

1.4 Cơ sở lý thuyết:

1.4.1 Node.js

Node.js là một môi trường thực thi JavaScript dựa trên Chrome's V8 Engine. Nó được chọn cho backend vì khả năng xử lý không đồng bộ và mô hình sự kiện, giúp xây dựng các server hiệu suất cao.

1.4.2 MongoDB

MongoDB là một hệ quản trị cơ sở dữ liệu NoSQL, lưu trữ dữ liệu dưới dạng JSON-like documents. Nó được sử dụng để lưu trữ thông tin sản phẩm, khách hàng và giao dịch, với ưu điểm về tính linh hoạt và khả năng mở rộng.

1.4.3 HTML, CSS, và JavaScript

HTML (HyperText Markup Language): Dùng để tạo cấu trúc của trang web, đặt các phần tử như form, nút và ô nhập liệu.

CSS (Cascading Style Sheets): Định dạng và trang trí cho các phần tử HTML, tạo giao diện người dùng thẩm mỹ và thân thiện.

JavaScript: Sử dụng để thêm tính năng tương tác vào trang web. Trong dự án, nó hỗ trợ việc thêm sản phẩm vào giỏ hàng, xử lý thanh toán và tương tác với backend thông qua Node.js.

1.4.4 Handlebars

Handlebars là một hệ thống template giúp tạo ra các template HTML để duy trì và sử dụng trong ứng dụng web. Nó giúp tách rời mã JavaScript và HTML, tạo sự rõ ràng và dễ quản lý.

1.4.5 JWT (JSON Web Token)

JSON Web Token (JWT) là một phương thức đơn giản và an toàn để truyền thông tin giữa các bên dưới dạng JSON. Trong dự án, JWT có thể được sử dụng để xác thực người dùng khi họ đăng nhập vào hệ thống.

1.4.6 Middlewares

Middlewares là các chương trình con có nhiệm vụ xử lý các yêu cầu trước khi chúng đến đích cuối cùng. Chúng có thể được sử dụng để thực hiện xác thực, quản lý phiên làm việc, và nhiều tác vụ khác. Trong Node.js, middlewares giúp quản lý luồng xử lý của ứng dụng.

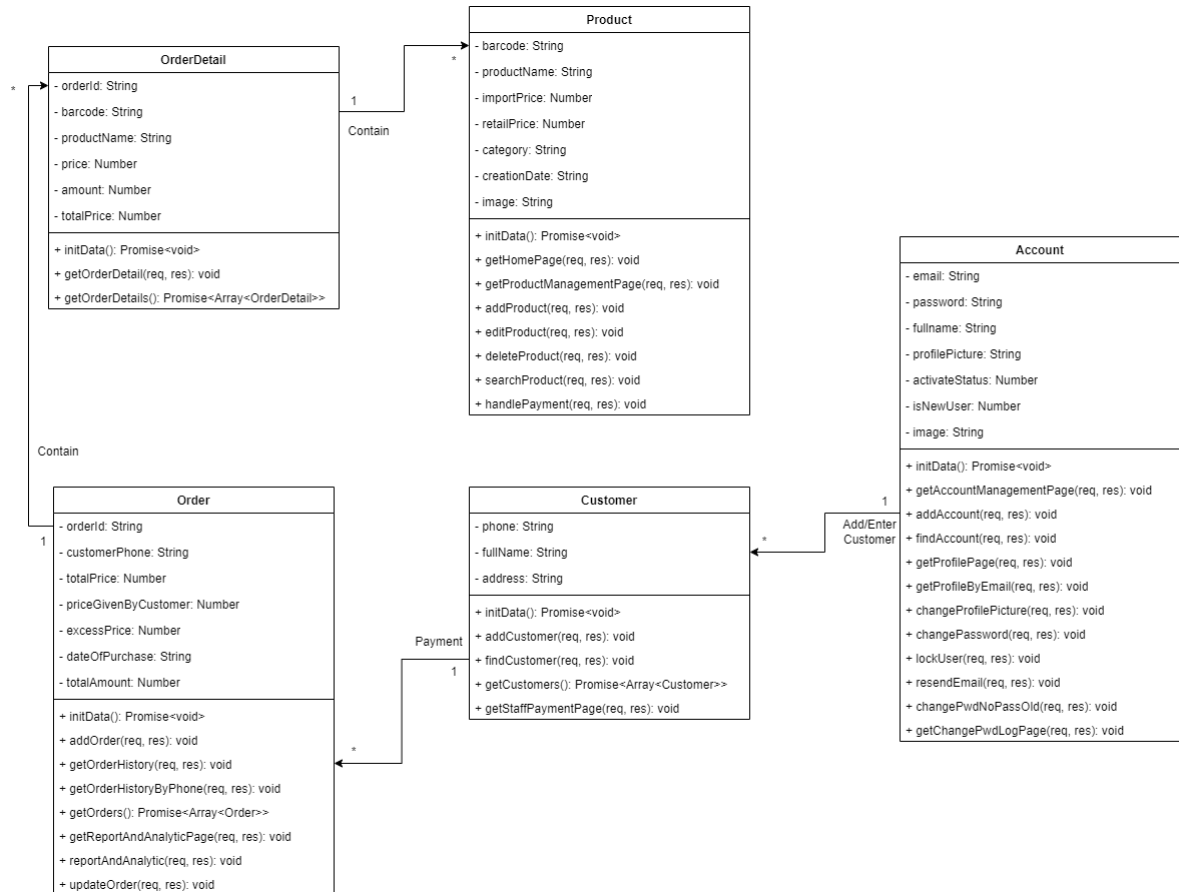
1.5 Ý nghĩa cơ sở lý thuyết:

Sự kết hợp của Node.js, MongoDB, HTML, CSS, và JavaScript trong việc phát triển ứng dụng quản lý bán hàng mang lại hiệu suất và trải nghiệm người dùng tốt. Handlebars giúp tạo ra giao diện dễ duy trì, trong khi JWT đảm bảo an toàn trong xác thực người dùng. Middlewares làm nhiệm vụ quan trọng trong quá trình xử lý yêu cầu, tăng tính ổn định và bảo mật cho hệ thống. Tổng hợp, cơ sở lý thuyết này cung cấp nền tảng vững chắc cho việc xây dựng ứng dụng quản lý bán hàng và điểm bán hàng cho cửa hàng điện thoại và phụ kiện.

CHƯƠNG 2 – PHÂN TÍCH THIẾT KẾ

2.1 Sơ đồ lớp

2.1.1 Vẽ sơ đồ



Hình 2. 1: Sơ đồ lớp của hàng bán lẻ điện thoại và phụ kiện

2.1.2 Mô tả sơ đồ

Hệ thống có các thành phần chính bao gồm các controllers và models liên quan đến quản lý sản phẩm, khách hàng, đơn đặt hàng, chi tiết đơn đặt hàng và tài khoản nhân viên.

Controllers:

- ProductController:
 - Chịu trách nhiệm quản lý sản phẩm, bao gồm hiển thị trang quản lý sản phẩm, trang chủ, thêm, sửa, xóa sản phẩm, tìm kiếm sản phẩm và xử lý thanh toán.
 - Sử dụng và liên kết với model Product để thực hiện các thao tác trên dữ liệu sản phẩm.
- CustomerController:
 - Quản lý thông tin khách hàng, bao gồm thêm mới khách hàng, tìm kiếm khách hàng và hiển thị trang thanh toán nhân viên.
 - Sử dụng và liên kết với model Customer để thực hiện các thao tác trên dữ liệu khách hàng.
- OrderController:
 - Điều khiển quá trình đặt hàng và lịch sử đặt hàng, bao gồm hiển thị trang lịch sử thanh toán, thêm mới đơn hàng và lấy thông tin đơn hàng theo số điện thoại khách hàng.
 - Sử dụng và liên kết với model Order và Customer để quản lý dữ liệu đơn hàng.
 - Báo cáo thống kê theo ngày: hiển thị tổng doanh thu, các đơn hàng theo ngày hoặc khoảng thời gian.
- OrderDetailController:
 - Quản lý chi tiết đơn hàng, bao gồm hiển thị trang chi tiết đơn hàng và khởi tạo dữ liệu mẫu.

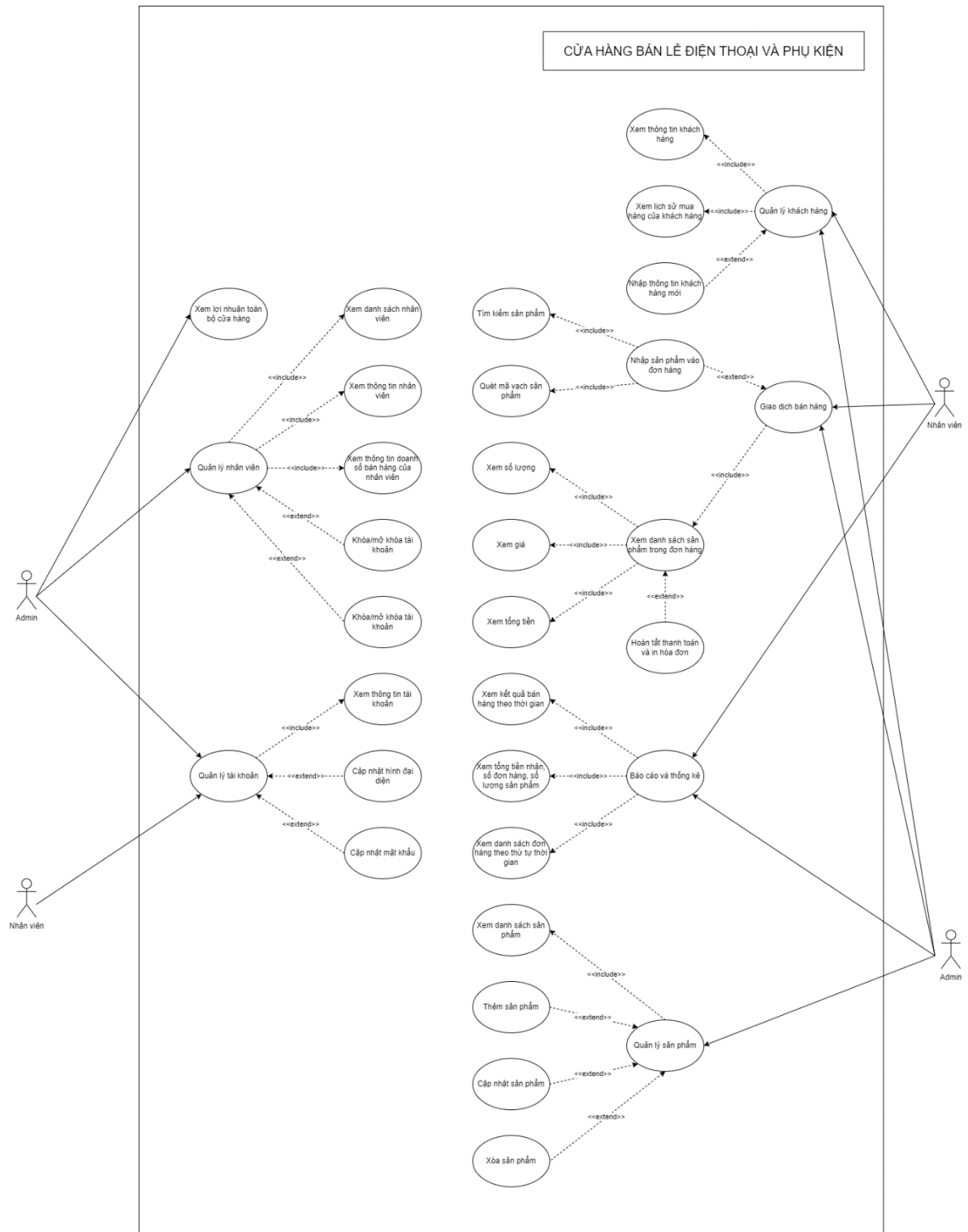
- Sử dụng và liên kết với model OrderDetail để thực hiện các thao tác trên dữ liệu chi tiết đơn hàng.
- AccountController:
 - Điều khiển quản lý tài khoản người dùng, bao gồm hiển thị trang quản lý tài khoản, đăng ký, đăng nhập, thay đổi mật khẩu và các chức năng liên quan.
 - Sử dụng và liên kết với model Account để thực hiện các thao tác trên dữ liệu tài khoản người dùng.

Models:

- Product:
 - Đại diện cho thông tin sản phẩm, bao gồm các thuộc tính như tên sản phẩm, giá, và mã vạch.
- Customer:
 - Lưu trữ thông tin về khách hàng như số điện thoại, địa chỉ và tên đầy đủ.
- Order:
 - Biểu diễn đơn hàng, bao gồm các thuộc tính như số điện thoại khách hàng, giá trị đơn hàng và ngày đặt hàng.
- OrderDetail:
 - Chứa thông tin chi tiết của đơn hàng, bao gồm mã đơn hàng, mã vạch sản phẩm, tên sản phẩm, giá, số lượng và tổng giá.
- Account:
 - Đại diện cho tài khoản người dùng với các thuộc tính như email, mật khẩu, tên đầy đủ và trạng thái khóa tài khoản.
 - Các controllers và models tương tác với nhau để hỗ trợ các chức năng chính của hệ thống quản lý bán hàng.

2.2 Sơ đồ use case

2.2.1 Vẽ sơ đồ



Hình 2. 2: Sơ đồ use case cửa hàng bán lẻ điện thoại và phụ kiện

2.2.2 Mô tả sơ đồ

Dưới đây là mô tả chi tiết hơn về các chức năng trong sơ đồ Use Case, bao gồm người thực hiện mỗi chức năng:

Quản lý tài khoản (account management):

- Xem thông tin tài khoản (view account information):
 - Người thực hiện: tất cả người dùng (admin và nhân viên).
 - Mô tả: cho phép người dùng xem thông tin chi tiết về tài khoản của mình, bao gồm hình đại diện, tên đầy đủ, và thông tin khác.
- Cập nhật hình đại diện và mật khẩu (update profile and password):
 - Người thực hiện: tất cả người dùng (admin và nhân viên).
 - Mô tả: cho phép người dùng cập nhật hình đại diện và thay đổi mật khẩu của mình.
- Quản lý nhân viên (manage employees):
 - Người thực hiện: admin.
 - Mô tả: cho phép admin xem danh sách nhân viên, xem chi tiết thông tin của mỗi nhân viên, và thực hiện các hành động như gửi lại email đăng nhập trong vòng 1 phút, khoá/mở khoá tài khoản, xem thông tin doanh số bán hàng của nhân viên.

Quản lý sản phẩm (product management):

- Xem danh sách sản phẩm (view product list):
 - Người thực hiện: tất cả người dùng (admin và nhân viên).
 - Mô tả: cho phép xem danh sách sản phẩm trong cửa hàng, bao gồm thông tin như tên sản phẩm, giá, và danh mục.
- Thêm mới sản phẩm (add new product):
 - Người thực hiện: admin.
 - Mô tả: cho phép admin thêm mới sản phẩm vào danh sách, nhập thông tin như mã vạch, tên sản phẩm, giá nhập, giá bán, danh mục, và ngày tạo.

- Cập nhật thông tin sản phẩm (update product information):
 - Người thực hiện: admin.
 - Mô tả: cho phép admin cập nhật thông tin chi tiết về sản phẩm, bao gồm cả việc thay đổi giá và danh mục.
- Xóa sản phẩm (delete product):
 - Người thực hiện: admin.
 - Mô tả: cho phép admin xóa sản phẩm khỏi danh sách, nhưng chỉ khi sản phẩm không thuộc đơn hàng nào.

Quản lý khách hàng (customer management):

- Xem thông tin khách hàng (view customer information):
 - Người thực hiện: nhân viên.
 - Mô tả: cho phép nhân viên xem thông tin chi tiết về khách hàng, bao gồm tên, số điện thoại, địa chỉ.
- Xem lịch sử mua hàng của khách hàng (view customer purchase history):
 - Người thực hiện: nhân viên.
 - Mô tả: cho phép nhân viên xem lịch sử mua hàng của khách hàng, bao gồm thông tin về đơn hàng, tổng tiền, và sản phẩm đã mua.
- Nhập thông tin khách hàng mới (add new customer):
 - Người thực hiện: nhân viên.
 - Mô tả: cho phép nhân viên nhập thông tin mới về khách hàng khi khách hàng thanh toán lần đầu tiên.

Giao dịch bán hàng (sales transactions):

- Nhập sản phẩm vào đơn hàng (add product to order):
 - Người thực hiện: nhân viên.
 - Mô tả: cho phép nhân viên thêm sản phẩm vào đơn hàng thông qua tìm kiếm hoặc quét mã vạch.
- Xem danh sách sản phẩm trong đơn hàng (view order items):
 - Người thực hiện: nhân viên.

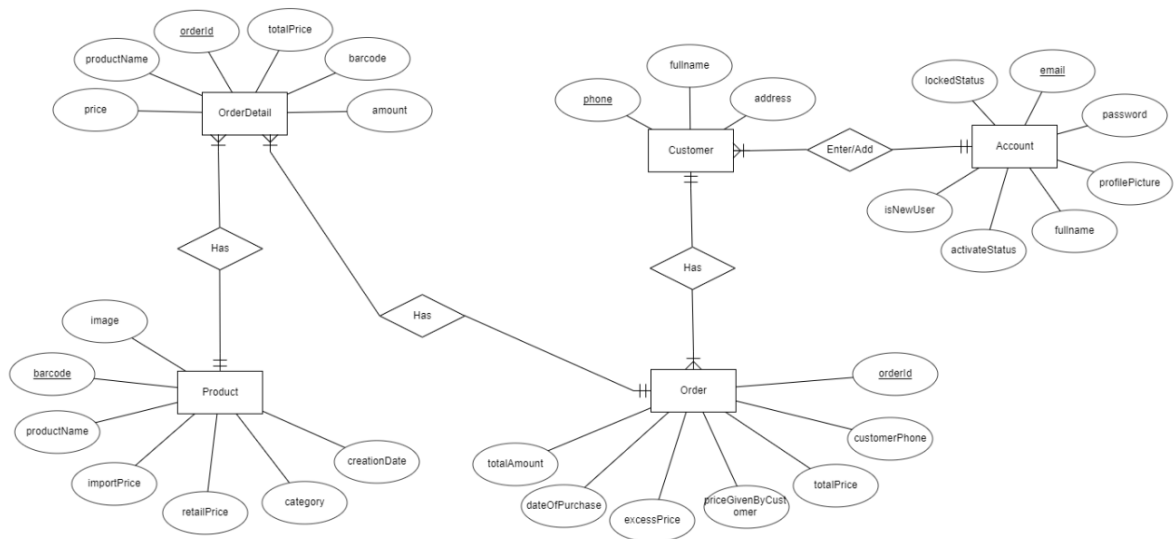
- Mô tả: hiển thị danh sách sản phẩm trong đơn hàng, bao gồm thông tin về số lượng, giá đơn vị và tổng tiền.
- Nhập thông tin khách hàng (enter customer information):
 - Người thực hiện: nhân viên.
 - Mô tả: cho phép nhân viên nhập thông tin khách hàng hoặc tạo mới nếu là khách hàng mới.
- Hoàn tất thanh toán và in hóa đơn (complete payment and print invoice):
 - Người thực hiện: nhân viên.
 - Mô tả: kết thúc quá trình thanh toán, in hóa đơn và cập nhật thông tin liên quan.

Báo cáo và thống kê (reporting and analytics):

- Xem kết quả bán hàng theo khoảng thời gian (view sales results by timeframe):
 - Người thực hiện: tất cả người dùng (admin và nhân viên).
 - Mô tả: hiển thị kết quả bán hàng dựa trên khoảng thời gian, bao gồm tổng số tiền, số đơn hàng, và số lượng sản phẩm.
- Xem danh sách đơn hàng (view order list):
 - Người thực hiện: tất cả người dùng (admin và nhân viên).
 - Mô tả: hiển thị danh sách các đơn hàng, sắp xếp theo thời gian.
- Xem thông tin lợi nhuận toàn bộ (view total profit):
 - Người thực hiện: admin.
 - Mô tả: hiển thị thông tin chi tiết về lợi nhuận toàn bộ.

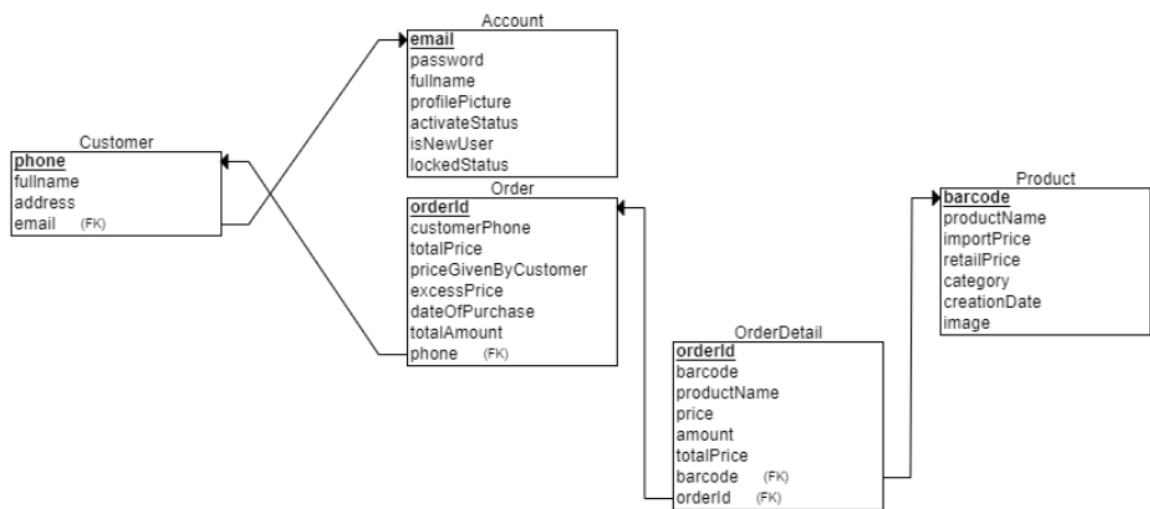
2.3 Sơ đồ ERD và sơ đồ quan hệ

2.3.1 Vẽ sơ đồ ERD



Hình 2. 3: Sơ đồ ERD của hàng bán lẻ điện thoại và phụ kiện

2.3.2 Vẽ sơ đồ quan hệ



Hình 2. 4: Sơ đồ quan hệ các thực thể của cửa hàng bán lẻ điện thoại và phụ kiện

2.3.3 Mô tả hai sơ đồ trên

Hệ thống cửa hàng bán lẻ điện thoại và phụ kiện có các thực thể như sau: tài khoản, khách hàng, sản phẩm, đơn hàng và chi tiết đơn hàng.

Mỗi tài khoản là một nhân viên có các thuộc tính như: email (khóa chính), mật khẩu, thông tin hình ảnh, họ và tên, trạng thái, thuộc tính kiểm tra có phải người mới hay không, thuộc tính khóa (khi nhân viên đó chưa xác thực email). Nhân viên có thể thêm khách hàng vào hệ thống hoặc nhập thông tin khách hàng khi khách hàng thực hiện đặt hàng hoặc hoàn tất hóa đơn cho khách hàng. Mỗi khách hàng chỉ được nhập hoặc thêm bởi một nhân viên còn mỗi nhân viên có thể nhập hoặc thêm được nhiều khách hàng. Thuộc tính của khách hàng gồm: số điện thoại (khóa chính), họ và tên, địa chỉ.

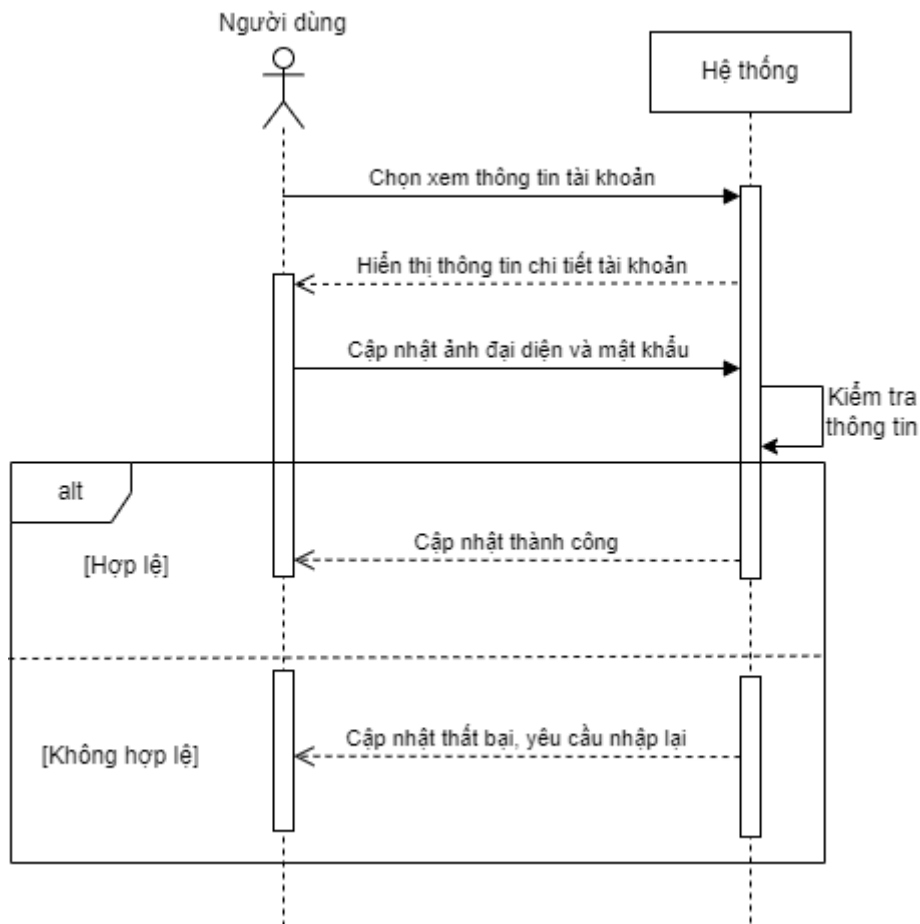
Mỗi khách hàng có thể có một hoặc nhiều hóa đơn, tuy nhiên mỗi hóa đơn chỉ có thể thuộc bởi một khách hàng. Thuộc tính của hóa đơn là: mã hóa đơn (khóa chính), số điện thoại khách hàng sở hữu hóa đơn, tổng tiền, số tiền mà khách hàng đưa, số tiền trả lại cho khách hàng, ngày mua, tổng số lượng sản phẩm mua.

Mỗi hóa đơn như thế thì gồm nhiều chi tiết hóa đơn, nhưng mỗi chi tiết hóa đơn thì chỉ nằm trong một hóa đơn. Chi tiết hóa đơn gồm các thuộc tính như: mã hóa đơn (khóa chính), tên sản phẩm, giá cả, tổng tiền, số lượng sản phẩm, barcode.

Mỗi chi tiết hóa đơn như thế thì chứa một sản phẩm nhưng mỗi sản phẩm lại có thể nằm trong nhiều chi tiết hóa đơn, thuộc tính của sản phẩm gồm: barcode (khóa chính), tên sản phẩm, hình ảnh, giá nhập, giá bán, phân loại, ngày tạo.

2.4 Sơ đồ tuần tự

2.4.1 Sơ đồ quản lý tài khoản

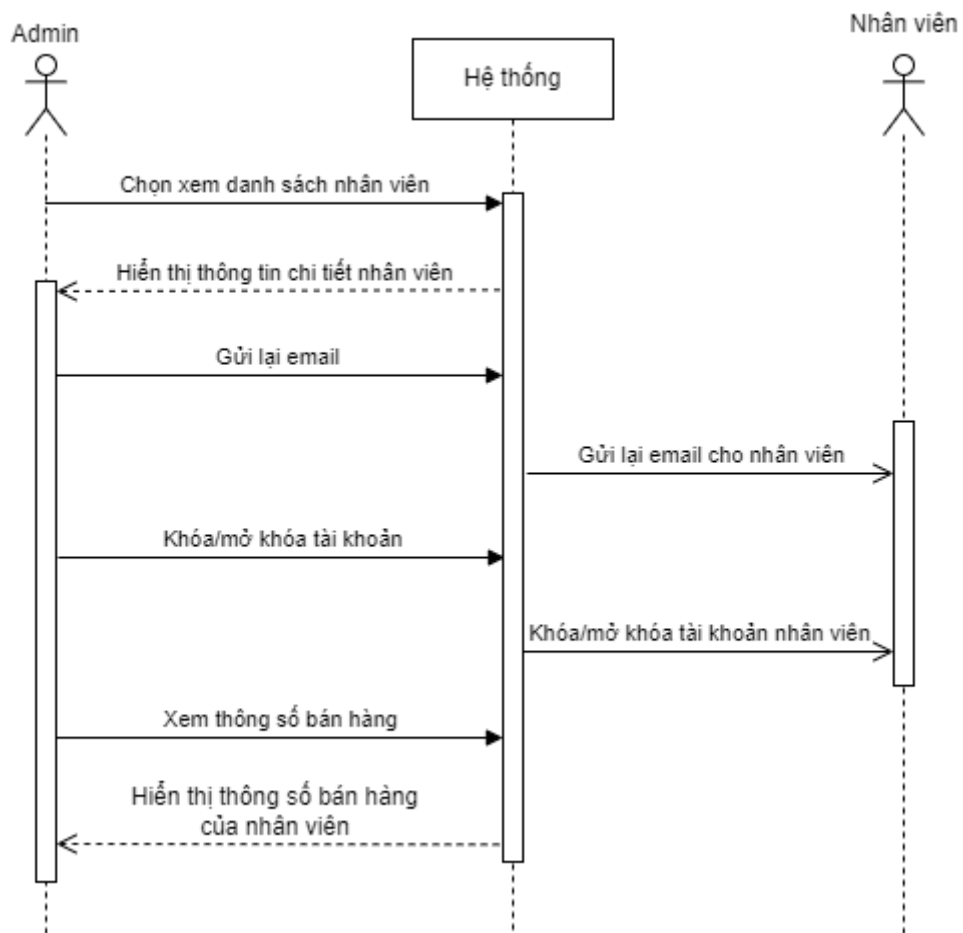


Hình 2. 5: Sơ đồ tuần tự quản lý tài khoản

Mô tả sơ đồ tuần tự quản lý tài khoản:

- Người dùng chọn chức năng xem thông tin tài khoản
- Hệ thống hiển thị thông tin chi tiết tài khoản
- Người dùng xem và chọn chức năng cập nhật ảnh đại diện hoặc mật khẩu
- Hệ thống kiểm tra thông tin cập nhật:
 - Nếu hợp lệ sẽ thông báo thành công.
 - Nếu không hợp lệ thông báo thất bại và yêu cầu người dùng nhập lại.

2.4.2 Sơ đồ quản lý nhân viên

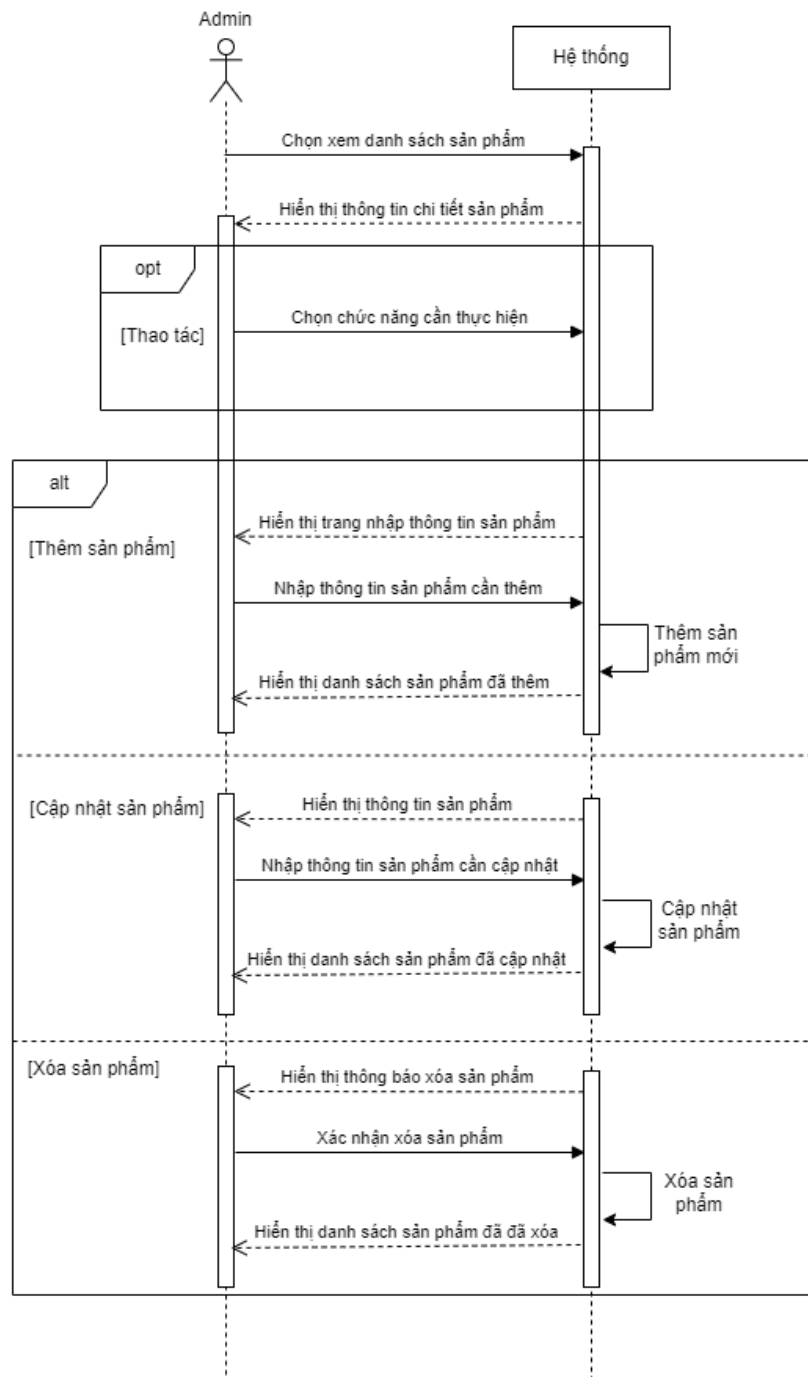


Hình 2. 6: Sơ đồ tuần tự quản lý nhân viên

Mô tả sơ đồ tuần tự quản lý nhân viên:

- Admin chọn chức năng xem danh sách nhân viên.
- Hệ thống hiển thị thông tin chi tiết nhân viên
- Admin xem và chọn các chức năng:
 - Gửi lại email: Hệ thống sẽ gửi lại email xác nhận cho nhân viên.
 - Khóa/mở khóa tài khoản: Hệ thống sẽ khóa/mở tài khoản của nhân viên đã chọn
 - Xem thông số bán hàng: Hệ thống sẽ hiển thị thông số bán hàng của nhân viên.

2.4.3 Sơ đồ quản lý sản phẩm

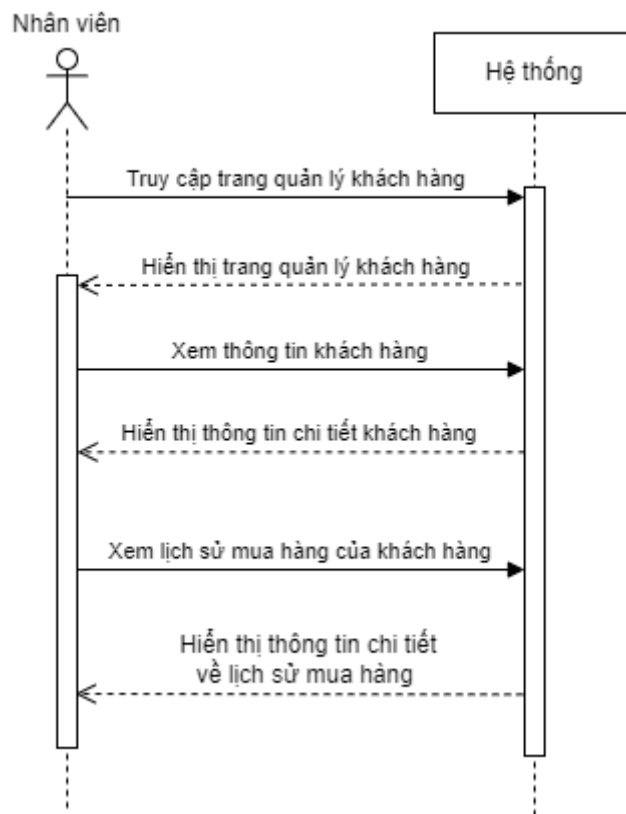


Hình 2. 7: Sơ đồ tuần tự quản lý sản phẩm

Mô tả sơ đồ tuần tự quản lý sản phẩm:

- Admin chọn chức năng xem danh sách sản phẩm.
- Hệ thống hiển thị chi tiết sản phẩm.
- Admin chọn các chức năng thêm, cập nhật và xóa sản phẩm.

2.4.4 Sơ đồ quản lý khách hàng

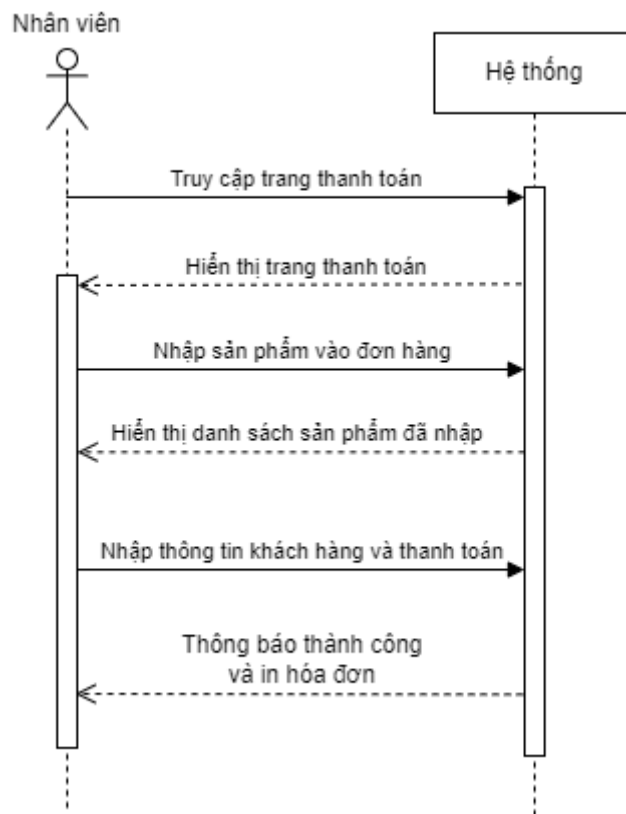


Hình 2. 8: Sơ đồ tuần tự quản lý khách hàng

Mô tả sơ đồ tuần tự quản lý khách hàng:

- Nhân viên truy cập vào trang quản lý khách hàng.
- Hệ thống hiển thị trang quản lý khách hàng.
- Nhân viên xem và chọn chức năng:
 - Xem thông tin khách hàng: Hệ thống hiển thị thông tin chi tiết khách hàng.
 - Xem lịch sử mua hàng của khách hàng: Hệ thống hiển thị thông tin chi tiết về lịch sử mua hàng.

2.4.5 Sơ đồ giao dịch bán hàng

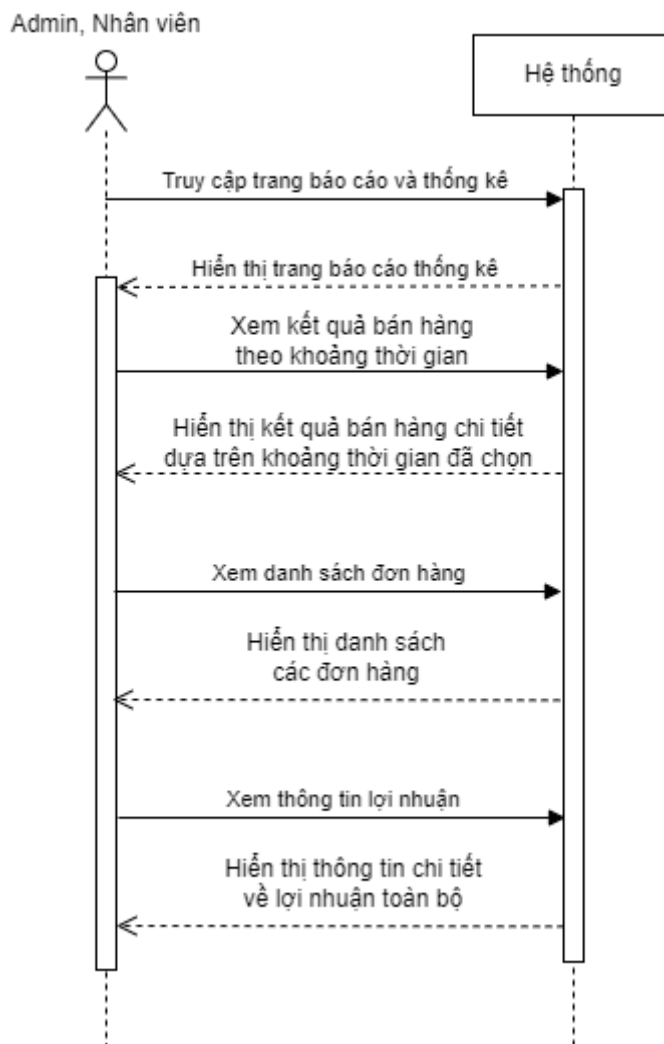


Hình 2. 9: Sơ đồ tuần tự giao dịch bán hàng

Mô tả sơ đồ tuần tự giao dịch bán hàng:

- Nhân viên truy cập vào trang thanh toán.
- Hệ thống hiển thị trang thanh toán.
- Nhân viên nhập sản phẩm vào đơn hàng.
- Hệ thống hiển thị danh sách sản phẩm đã nhập.
- Nhân viên nhập thông tin khách hàng và thanh toán.
- Hệ thống thông báo thành công và in hóa đơn.

2.4.6 Sơ đồ báo cáo và thống kê



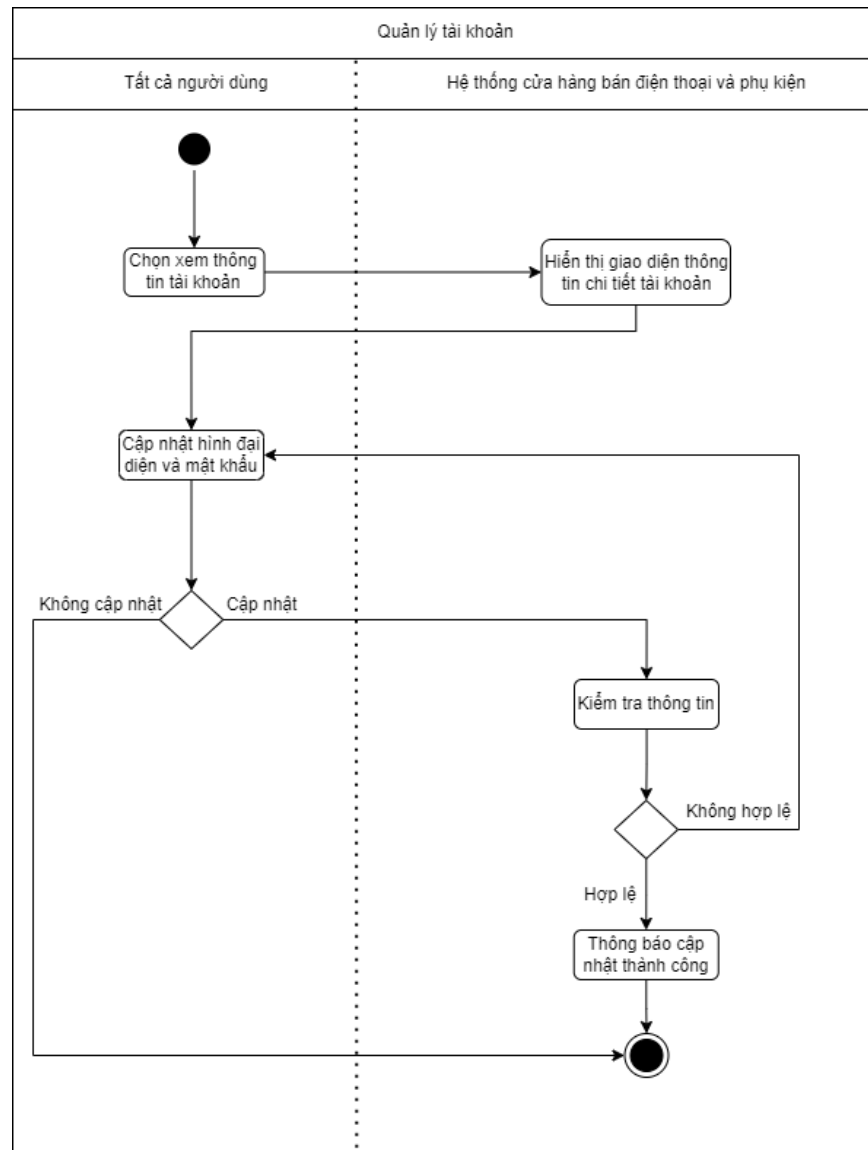
Hình 2. 10: Sơ đồ tuần tự báo cáo và thống kê

Mô tả sơ đồ tuần tự báo cáo và thống kê:

- Admin, Nhân viên truy cập vào trang báo cáo và thống kê.
- Hệ thống hiển thị trang báo cáo và thống kê.
- Admin, Nhân viên xem và chọn chức năng:
 - Xem kết quả bán hàng theo khoảng thời gian: Hệ thống hiển thị kết quả bán hàng chi tiết dựa trên khoảng thời gian đã chọn.
 - Xem danh sách đơn hàng: Hệ thống hiển thị danh sách các đơn hàng.
 - Xem thông tin lợi nhuận: Hệ thống hiển thị thông tin chi tiết về lợi nhuận toàn bộ.

2.5 Sơ đồ hoạt động

2.5.1 Sơ đồ quản lý tài khoản



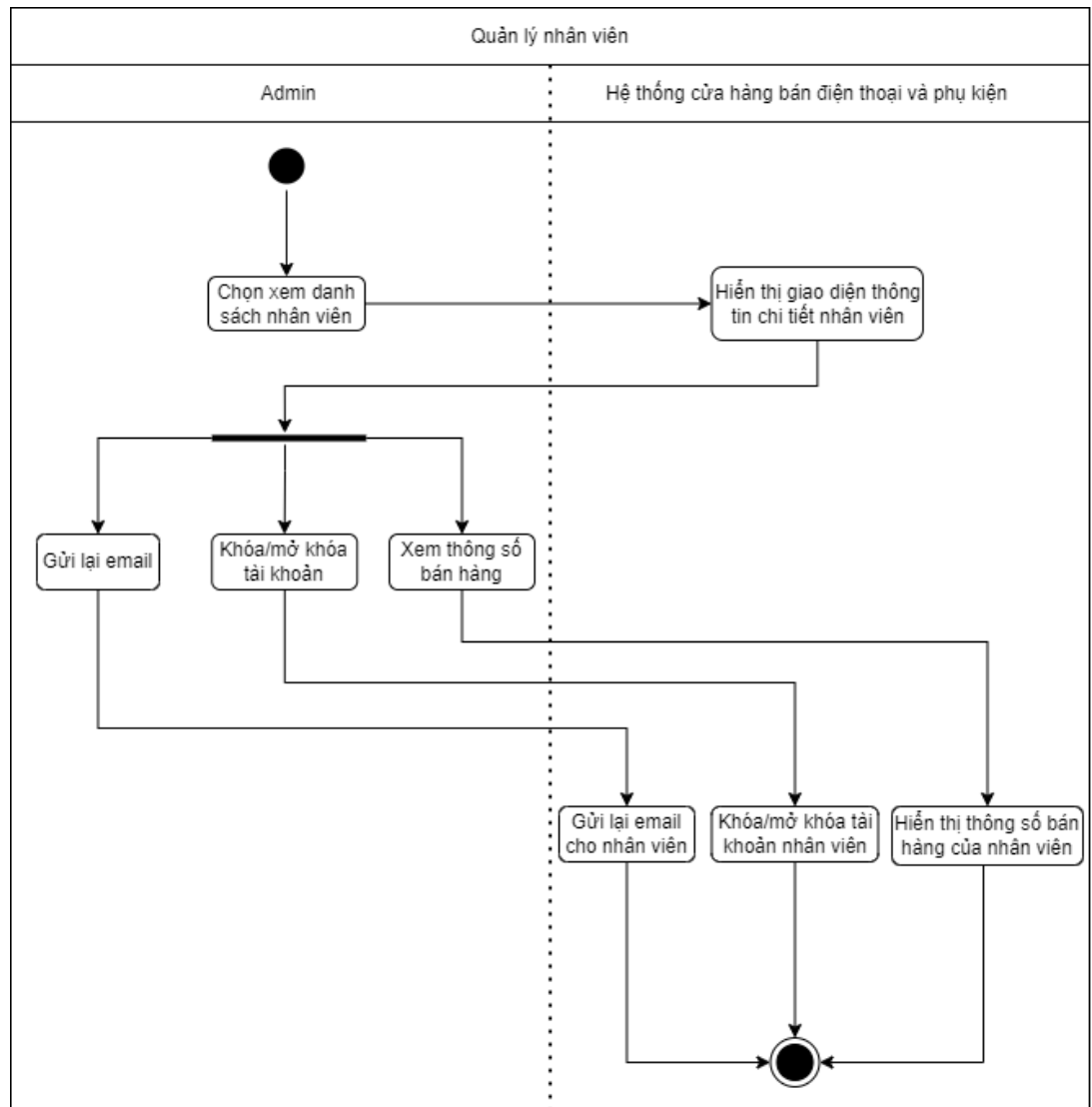
Hình 2. 11: Sơ đồ hoạt động quản lý tài khoản

Mô tả sơ đồ hoạt động quản lý tài khoản:

Tất cả người dùng có thể:

- Xem thông tin chi tiết về tài khoản của mình, bao gồm hình đại diện, tên đầy đủ và thông tin khác.
- Cập nhật hình đại diện và thay đổi mật khẩu của mình.

2.5.2 Sơ đồ quản lý nhân viên

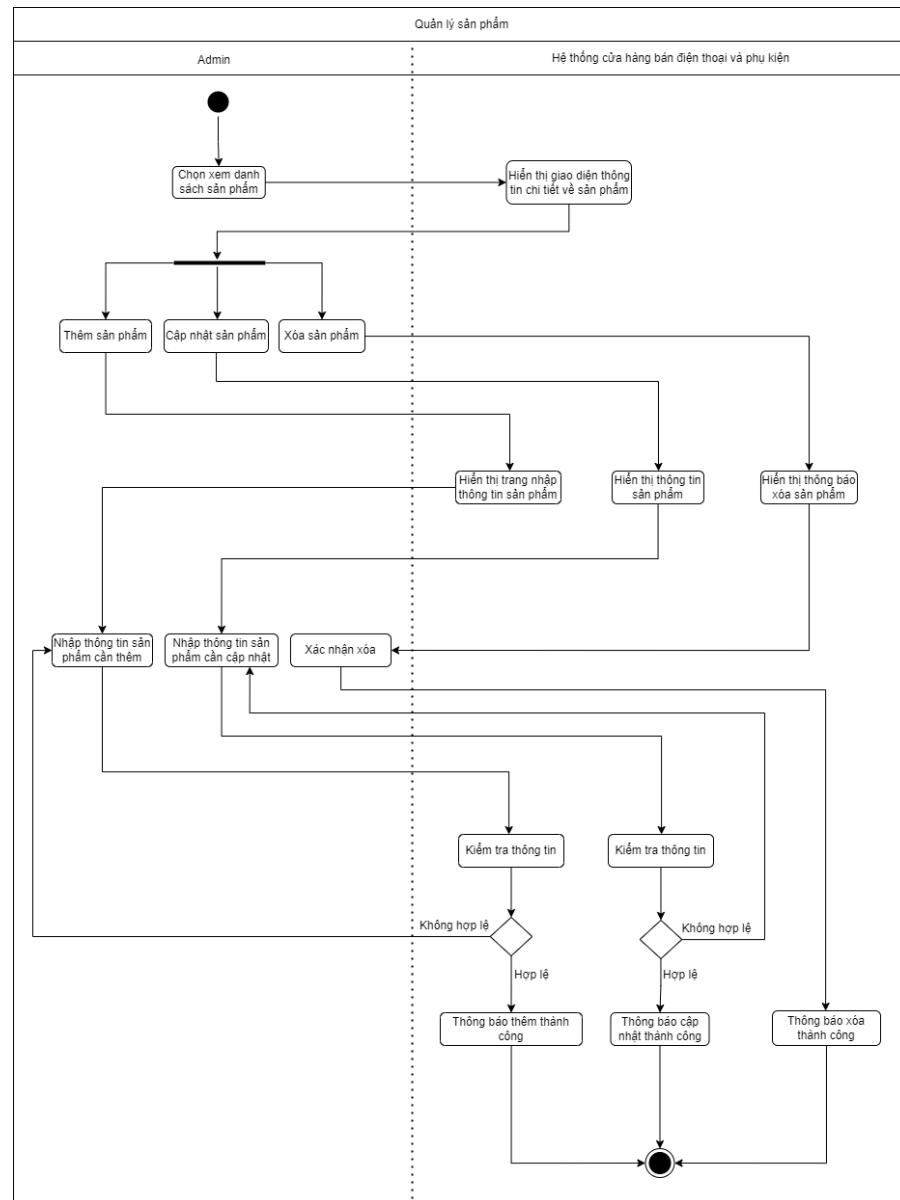


Hình 2. 12: Sơ đồ hoạt động quản lý nhân viên

Mô tả sơ đồ hoạt động quản lý nhân viên:

Admin có thể xem danh sách nhân viên, xem chi tiết thông tin của mỗi nhân viên, và thực hiện các hành động như gửi lại email đăng nhập trong vòng 1 phút, khoá/mở khoá tài khoản, xem thông tin doanh số bán hàng của nhân viên.

2.5.3 Sơ đồ quản lý sản phẩm



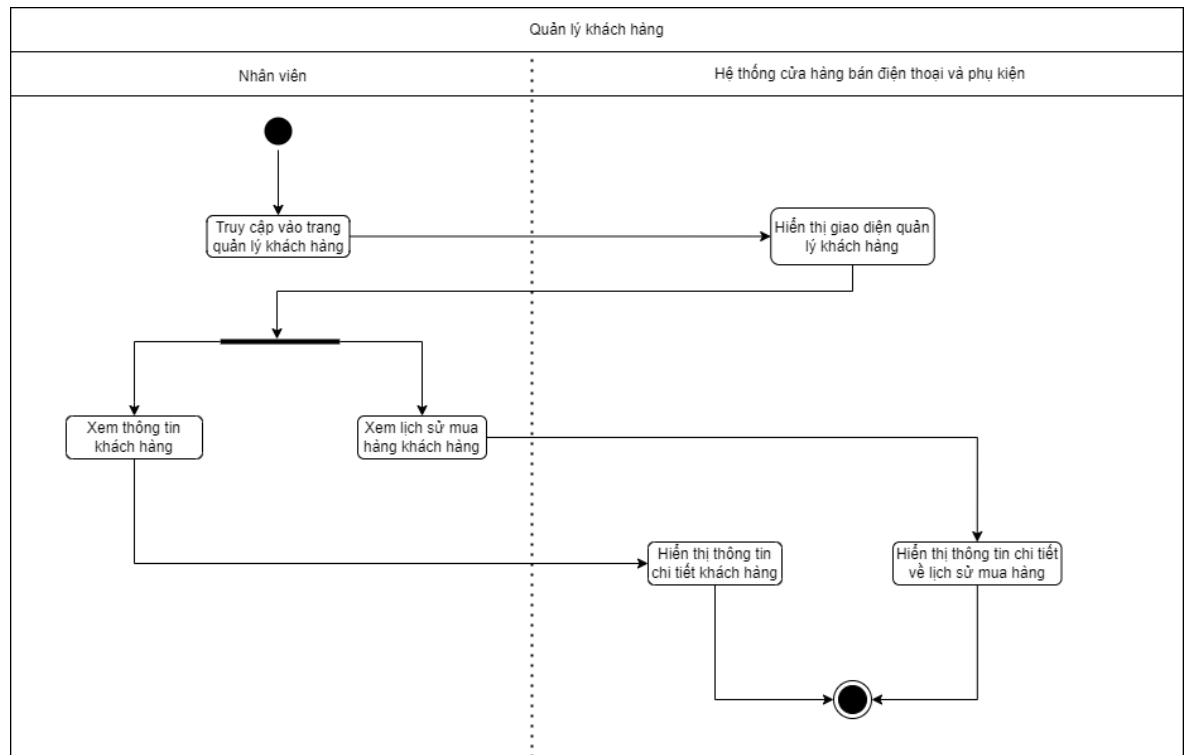
Hình 2. 13: Sơ đồ hoạt động quản lý sản phẩm

Mô tả sơ đồ hoạt động quản lý sản phẩm:

Admin có thể:

- Thêm mới sản phẩm vào danh sách, nhập thông tin như mã vạch, tên sản phẩm, giá nhập, giá bán, danh mục, và ngày tạo.
- Cập nhật thông tin chi tiết về sản phẩm, bao gồm cả việc thay đổi giá và danh mục.
- Xóa sản phẩm khỏi danh sách.

2.5.4 Sơ đồ quản lý khách hàng



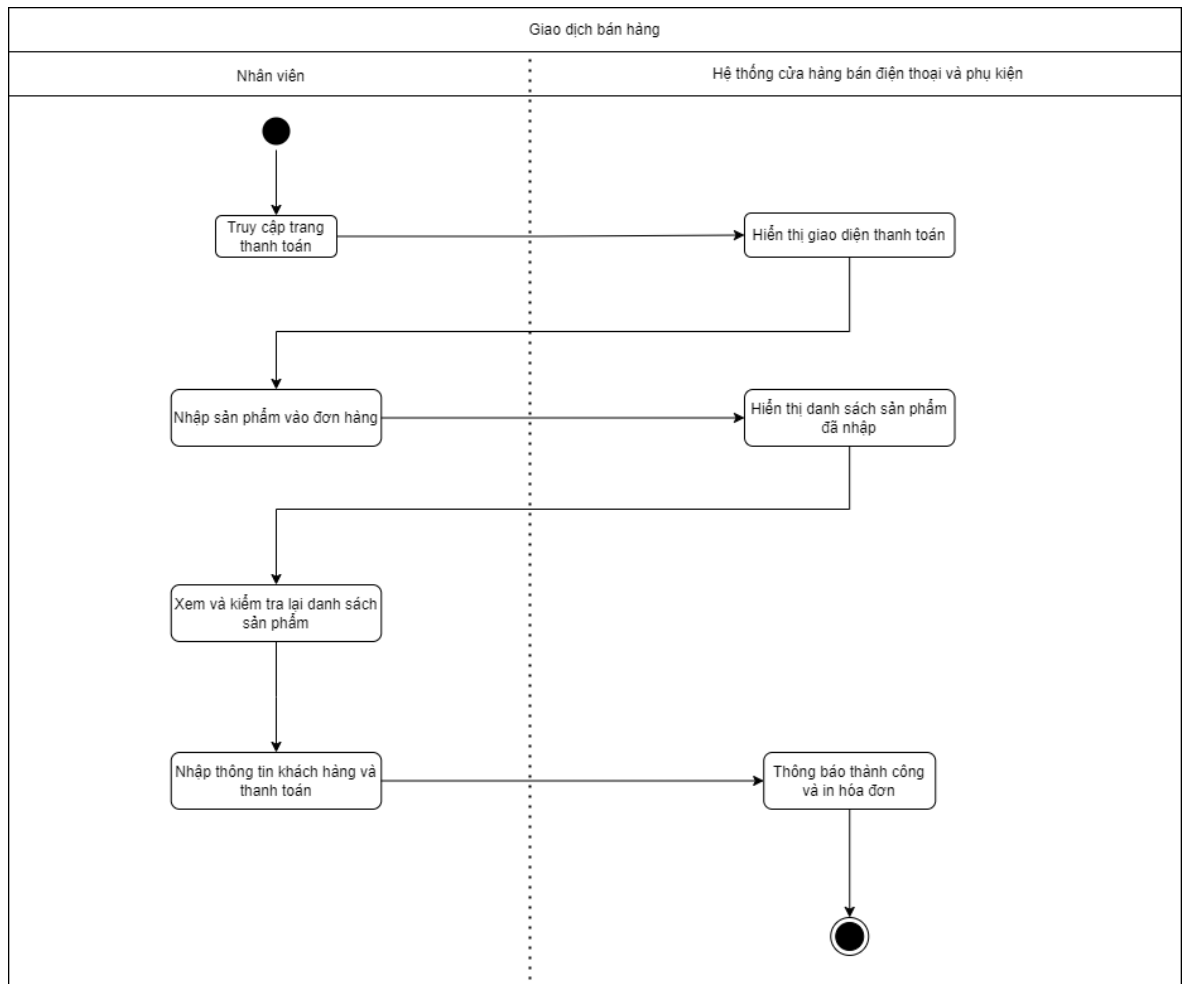
Hình 2. 14: Sơ đồ hoạt động quản lý khách hàng

Mô tả sơ đồ hoạt động quản lý khách hàng:

Nhân viên có thể:

- Xem thông tin chi tiết về khách hàng, bao gồm tên, số điện thoại, địa chỉ.
- Xem lịch sử mua hàng của khách hàng, bao gồm thông tin về đơn hàng, tổng tiền, và sản phẩm đã mua.

2.5.5 Sơ đồ giao dịch bán hàng



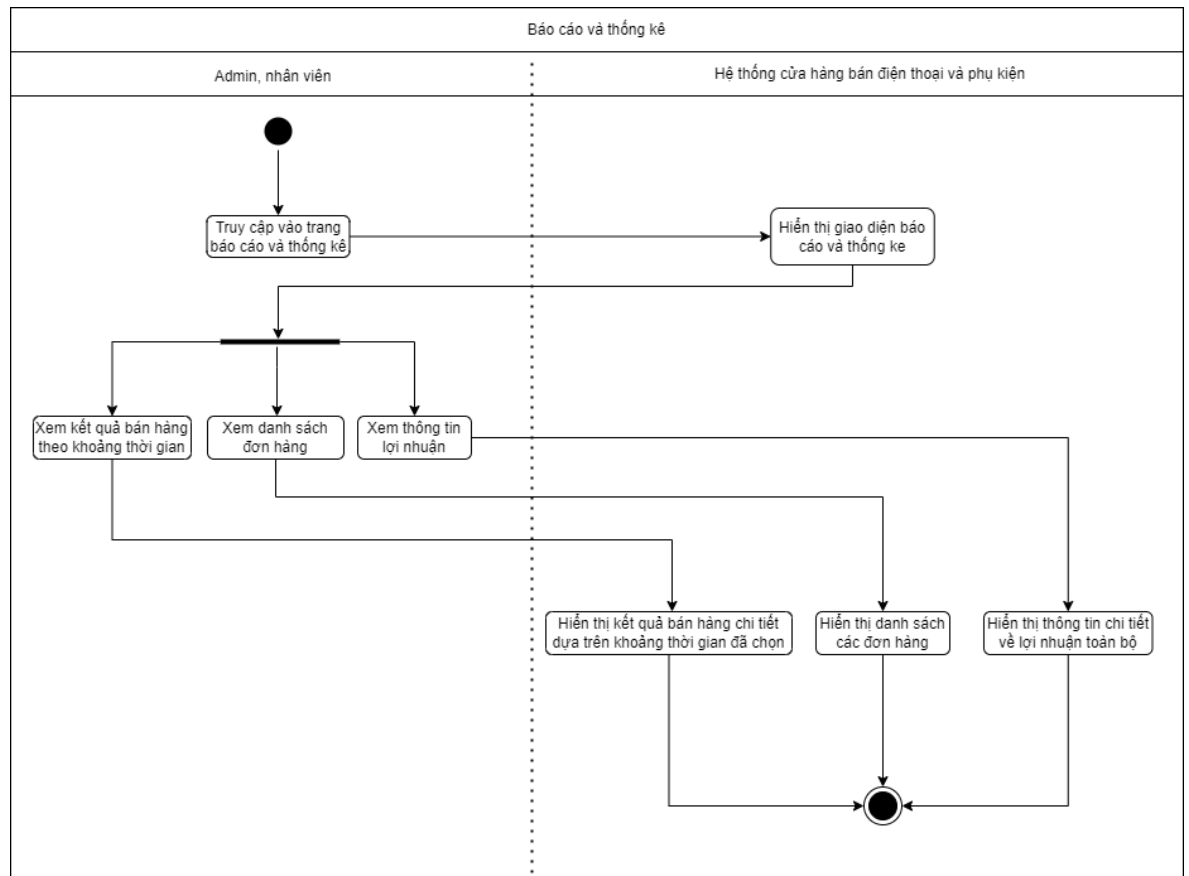
Hình 2. 15: Sơ đồ hoạt động giao dịch bán hàng

Mô tả sơ đồ hoạt động giao dịch bán hàng:

Nhân viên có thể:

- Thêm sản phẩm vào đơn hàng thông qua tìm kiếm hoặc quét mã vạch.
- Xem danh sách sản phẩm trong đơn hàng, bao gồm thông tin về số lượng, giá đơn vị và tổng tiền.
- Nhập thông tin khách hàng hoặc tạo mới nếu là khách hàng mới.
- Xác nhận thanh toán và in hóa đơn.

2.5.6 Sơ đồ báo cáo và thống kê



Hình 2. 16: Sơ đồ hoạt động báo cáo và thống kê

Mô tả sơ đồ hoạt động báo cáo thống kê:

Admin và nhân viên có thể:

- Xem kết quả bán hàng dựa trên khoảng thời gian, bao gồm tổng số tiền, số đơn hàng, và số lượng sản phẩm.
- Xem danh sách các đơn hàng, sắp xếp theo thời gian.
- Xem thông tin chi tiết về lợi nhuận toàn bộ.

CHƯƠNG 3 – HIỆN THỰC HỆ THỐNG

3.1 Chức năng chính của ứng dụng: xử lý giao dịch và chức năng báo cáo thống kê của admin

3.1.1 Code của phần OrderController

```
let Order = require("../models/order");
let Customer = require("../models/customer");

async function getOrderHistory(req, res) {
  const ITEMS_PER_PAGE = 10; // Số lượng item mỗi trang
  const page = parseInt(req.query.page) || 1; // Lấy số trang hiện tại
  const nextPage = page + 1;
  const prevPage = page - 1;
  const skip = (page - 1) * ITEMS_PER_PAGE;

  try {
    const orders = await getOrders();
    const totalOrders = orders.length;
    const totalPages = Math.ceil(totalOrders / ITEMS_PER_PAGE);
    const pages = Array.from({ length: totalPages }, (_, index) => index
+ 1);

    const paginatedOrders = orders.slice(skip, skip + ITEMS_PER_PAGE);

    res.render('payment-history', {
      orders: paginatedOrders,
      success: req.flash("success"), error: req.flash("error"),
      email: req.session.email,
      currentPage: page,
      nextPage: nextPage,
      prevPage: prevPage,
      totalPages: totalPages,
      pages: pages,
    });
  }
}
```

```

    } catch (error) {
      console.error('Error fetching products:', error);
      res.status(500).send('Internal Server Error');
    }
  }
}

async function getOrders() {
  try {
    let orders = await Order.find().lean();
    return orders;
  } catch (error) {
    throw new Error("Đã xảy ra lỗi khi lấy dữ liệu từ cơ sở dữ liệu.");
  }
}

async function getOrderHistoryByPhone(req, res) {
  let phone = req.params.customerPhone;

  const ITEMS_PER_PAGE = 10; // Số lượng item mỗi trang
  const page = parseInt(req.query.page) || 1; // Lấy số trang hiện tại
  const nextPage = page + 1;
  const prevPage = page - 1;
  const skip = (page - 1) * ITEMS_PER_PAGE;

  try {
    const orders = await Order.find({ customerPhone: phone }).lean();
    const totalOrders = orders.length;
    const totalPages = Math.ceil(totalOrders / ITEMS_PER_PAGE);
    const pages = Array.from({ length: totalPages }, (_, index) => index
+ 1);

    const paginatedOrders = orders.slice(skip, skip + ITEMS_PER_PAGE);

    let options = {
      orders: paginatedOrders,
      success: req.flash("success"),

```



```

        error: req.flash("error"),
        email: req.session.email,
        currentPage: page,
        nextPage: nextPage,
        prevPage: prevPage,
        totalPages: totalPages,
        pages: pages,
    });

    res.render("payment-history", options);
} catch (error) {
    console.error('Error fetching order history by phone:', error);
    res.status(500).send('Internal Server Error');
}
}

// Khởi tạo 1 số dữ liệu mẫu để chạy chương trình
async function initData() {
    // Trước khi khởi tạo dữ liệu mẫu thì ta cần xóa các dữ liệu hiện có
    await Order.deleteMany()

    let order = new Order({
        orderId: "101220230940410909000000",
        customerPhone: "0909000000",
        totalPrice: 4658000,
        priceGivenByCustomer: 5000000,
        excessPrice: 342000,
        dateOfPurchase: "10/12/2023",
        totalAmount: 2,
    });

    await order.save()

    let order2 = new Order({
        orderId: "091220230941030909000000",
        customerPhone: "0909000000",
    });

```

```
        totalPrice: 25590000,  
        priceGivenByCustomer: 26000000,  
        excessPrice: 410000,  
        dateOfPurchase: "09/12/2023",  
        totalAmount: 1,  
    });  
  
    await order2.save()  
  
    let order3 = new Order({  
        orderId: "021220230941200909111111",  
        customerPhone: "0909111111",  
        totalPrice: 315000,  
        priceGivenByCustomer: 320000,  
        excessPrice: 5000,  
        dateOfPurchase: "02/12/2023",  
        totalAmount: 1,  
    });  
  
    await order3.save()  
  
    let order4 = new Order({  
        orderId: "041220230942120909222222",  
        customerPhone: "0909222222",  
        totalPrice: 22000000,  
        priceGivenByCustomer: 23000000,  
        excessPrice: 1000000,  
        dateOfPurchase: "04/12/2023",  
        totalAmount: 1,  
    });  
  
    await order4.save()  
  
    let order5 = new Order({  
        orderId: "251120230942340909111111",  
        customerPhone: "0909111111",
```

```

        totalPrice: 621000,
        priceGivenByCustomer: 621000,
        excessPrice: 0,
        dateOfPurchase: "25/11/2023",
        totalAmount: 1,
    });

    await order5.save()
}

async function addOrder(req, res) {
    let currentDate = new Date();
    let phone = req.body.phone;
    let formattedDate = `${currentDate.getDate().toString().padStart(2, '0')}/${currentDate.getMonth() + 1).toString().padStart(2, '0')}/${currentDate.getFullYear()}`;
    let formattedOrderId = `${currentDate.getDate().toString().padStart(2, '0')}${currentDate.getMonth() + 1).toString().padStart(2, '0')}${currentDate.getFullYear()}${currentDate.getHours().toString().padStart(2, '0')}${currentDate.getMinutes().toString().padStart(2, '0')}${currentDate.getSeconds().toString().padStart(2, '0')}${phone}`;

    if (req.body.phone === "" || req.body.fullname === "Không tìm thấy khách hàng" || req.body.address === "Không tìm thấy khách hàng" || req.body.fullname === "" || req.body.address === "") {
        req.flash("error", "Thông tin khách hàng không hợp lệ");
        return res.render("product-payment", { error: req.flash("error") });
    }

    let order = new Order({
        orderId: formattedOrderId,
        customerPhone: req.body.phone,
        totalPrice: req.body.totalAmountInput,
        priceGivenByCustomer: 0,
        excessPrice: 0,
        dateOfPurchase: formattedDate,
    });

```

```

        totalAmount: req.body.totalQuantityInput
    });

    order.save()
        .then(newOrder => {
            res.redirect("invoice");
        })
        .catch(error => {
            req.flash("error", "Có lỗi khi tạo hóa đơn");
            res.render("product-payment", { error: req.flash("error") });
        });
}

// Cập nhật lại priceGivenByCustomer và excessPrice
async function updateOrder(req, res) {
    let orderId = req.body.orderId;
    let priceGivenByCustomer = req.body.priceGivenByCustomer;
    let excessPrice = req.body.excessPrice;

    let order = await Order.findOne({ orderId: orderId });
    if (order === null)
        return res.json({ code: 1, error: "Hóa đơn không tồn tại" });

    order.priceGivenByCustomer = priceGivenByCustomer;
    order.excessPrice = excessPrice;
    await order.save();

    res.json({ code: 0, success: "In hóa đơn bán hàng thành công!" });
}

async function getReportAndAnalyticPage(req, res) {
    let email = req.session.email;

    if(email === "admin@gmail.com")
        res.render('report-analytic', { layout: "admin" });
    else

```

```

        res.render('report-analytic', { email });
    }

    async function reportAndAnalytic(req, res) {
        let generalTimeline = req.body.generalTimeline;
        let fromTimeLine = req.body.fromTimeLine;
        let toTimeLine = req.body.toTimeLine;
        let orders;

        // mặc định -> lấy hết
        if (!generalTimeline && !fromTimeLine && !toTimeLine) {
            orders = await Order.find().lean();
            return res.json({ code: 0, orders: orders });
        }

        // trường hợp mốc thời gian
        if (generalTimeline) {
            orders = await Order.find({ dateOfPurchase: generalTimeline
        }).lean();
            return res.json({ code: 0, orders: orders });
        }

        // trường hợp from - to
        orders = await Order.find({
            dateOfPurchase: {
                $gte: fromTimeLine,
                $lte: toTimeLine
            }
        })
        .lean();

        res.json({ code: 0, orders: orders });
    }

    module.exports = {
        getOrderHistory,
    }

```

```

getOrderHistoryByPhone,
initData,
addOrder,
updateOrder,
getReportAndAnalyticPage,
reportAndAnalytic
};

```

3.1.2 Giải thích code của OrderController

Mô tả chức năng của các module trong OrderController:

- Hàm getOrderHistory:
 - Chức năng: lấy danh sách đơn đặt hàng và render trang 'payment-history' với thông tin đơn hàng và thông báo.
 - Thực hiện:
 - Gọi hàm getOrders để lấy danh sách đơn đặt hàng.
 - Render trang 'payment-history' với danh sách đơn đặt hàng, và các thông báo success và error.
- Hàm getOrders:
 - Chức năng: lấy danh sách đơn đặt hàng từ cơ sở dữ liệu.
 - Thực hiện:
 - Sử dụng Order.find() để lấy danh sách đơn đặt hàng từ MongoDB.
 - Trả về danh sách đơn đặt hàng.
 - Nếu có lỗi, throw một lỗi với thông báo lỗi.
- Hàm getOrderHistoryByPhone:
 - Chức năng: lấy thông tin đơn đặt hàng dựa trên số điện thoại của khách hàng và render trang 'payment-history'.
 - Thực hiện:
 - Sử dụng Order.findOne để tìm đơn đặt hàng dựa trên số điện thoại được truyền vào.
 - Nếu tìm thấy, tạo một object options chứa thông tin của đơn đặt hàng.

- Render trang 'payment-history' với thông tin của đơn đặt hàng và thông báo success và error.
- Hàm initData:
 - Chức năng: khởi tạo dữ liệu mẫu đơn đặt hàng để chạy chương trình.
 - Thực hiện:
 - Trước khi khởi tạo, xóa tất cả đơn đặt hàng hiện có.
 - Tạo và lưu hai đơn đặt hàng mẫu sử dụng model Order.
- Hàm addOrder:
 - Chức năng: thêm một đơn đặt hàng mới vào cơ sở dữ liệu và chuyển hướng đến trang 'invoice'.
 - Thực hiện:
 - Lấy thông tin cần thiết từ request và thời gian hiện tại để tạo một đơn đặt hàng mới.
 - Kiểm tra tính hợp lệ của thông tin khách hàng. Nếu không hợp lệ, hiển thị thông báo lỗi và render lại trang 'product-payment'.
 - Nếu thành công, chuyển hướng đến trang 'invoice'.
 - Nếu có lỗi, hiển thị thông báo lỗi và render lại trang 'product-payment'.
- Hàm updateOrder:
 - Mục đích: Cập nhật giá được đưa bởi khách hàng (priceGivenByCustomer) và số tiền thừa (excessPrice) cho một đơn đặt hàng cụ thể.
 - Thực hiện:
 - Trích xuất orderId, priceGivenByCustomer, và excessPrice từ request.
 - Tìm đơn đặt hàng với orderId cụ thể.
 - Cập nhật các trường priceGivenByCustomer và excessPrice.
 - Lưu đơn đặt hàng đã cập nhật vào cơ sở dữ liệu.
 - Phản hồi:

- Nếu không tìm thấy đơn đặt hàng, trả về một phản hồi JSON với mã lỗi và thông báo lỗi.
- Nếu cập nhật thành công, trả về một phản hồi JSON với mã thành công và thông báo thành công.
- Hàm `getReportAndAnalyticPage`:
 - Mục đích: Render trang 'report-analytic', với giao diện khác nhau cho người dùng admin và người dùng thông thường.
 - Thực hiện:
 - Kiểm tra email của người dùng trong session để xác định liệu họ có phải là admin không.
 - Render trang 'report-analytic' với layout phù hợp.
- Hàm `reportAndAnalytic`:
 - Mục đích: Lấy thông tin đơn đặt hàng dựa trên các dải thời gian chỉ định cho báo cáo và phân tích.
 - Thực hiện:
 - Phân tích các tham số như `generalTimeline`, `fromTimeLine`, và `toTimeLine` từ request.
 - Nếu không có dải thời gian cụ thể được chỉ định, lấy tất cả các đơn đặt hàng.
 - Trong trường hợp có dải thời gian cụ thể, lấy đơn đặt hàng trong khoảng thời gian đó.
 - Trả về một phản hồi JSON với danh sách đơn đặt hàng.

3.1.3 Code của phân *OrderDetailController*

```
let OrderDetail = require("../models/orderdetail");

async function getOrderDetail(req, res) {
  const ITEMS_PER_PAGE = 10; // Số lượng item mỗi trang
  const page = parseInt(req.query.page) || 1; // Lấy số trang hiện tại
  const nextPage = page + 1;
  const prevPage = page - 1;
  const skip = (page - 1) * ITEMS_PER_PAGE;

  try {
    const orderdetails = await getOrderDetails();

    const totalOrderDetails = orderdetails.length;
    const totalPages = Math.ceil(totalOrderDetails / ITEMS_PER_PAGE);
    const pages = Array.from({ length: totalPages }, (_, index) => index
+ 1);

    const paginatedOrderDetails = orderdetails.slice(skip, skip +
ITEMS_PER_PAGE);

    res.render('detail-order', {
      orderdetails: paginatedOrderDetails,
      success: req.flash("success"),
      error: req.flash("error"),
      email: req.session.email,
      currentPage: page,
      nextPage: nextPage,
      prevPage: prevPage,
      totalPages: totalPages,
      pages: pages,
    });
  } catch (error) {
    console.error('Error fetching products:', error);
    res.status(500).send('Internal Server Error');
  }
}
```

```

    }
  }

  async function getOrderDetails() {
    try {
      let orderdetails = await OrderDetail.find().lean();
      return orderdetails;
    } catch (error) {
      throw new Error("Đã xảy ra lỗi khi lấy dữ liệu từ cơ sở dữ liệu.");
    }
  }

  async function getOrderDetailById(req, res) {
    let orderId = req.params.orderId;

    const ITEMS_PER_PAGE = 10; // Số lượng item mỗi trang
    const page = parseInt(req.query.page) || 1; // Lấy số trang hiện tại
    const nextPage = page + 1;
    const prevPage = page - 1;
    const skip = (page - 1) * ITEMS_PER_PAGE;

    try {
      const orderDetails = await OrderDetail.find({ orderId: { $regex: `^${orderId.slice(0, -3)}` } }).lean();
      let email = req.session.email;

      const totalOrderDetails = orderDetails.length;
      const totalPages = Math.ceil(totalOrderDetails / ITEMS_PER_PAGE);
      const pages = Array.from({ length: totalPages }, (_, index) => index + 1);

      const paginatedOrderDetails = orderDetails.slice(skip, skip + ITEMS_PER_PAGE);

      if(email === "admin@gmail.com")
        res.render("detail-order", {

```

```

        layout: "admin",
        orderDetails: paginatedOrderDetails,
        success: req.flash("success"),
        error: req.flash("error"),
        currentPage: page,
        nextPage: nextPage,
        prevPage: prevPage,
        totalPages: totalPages,
        pages: pages,
    });
    else
        res.render("detail-order", {
            email: email,
            orderDetails: paginatedOrderDetails,
            success: req.flash("success"),
            error: req.flash("error"),
            currentPage: page,
            nextPage: nextPage,
            prevPage: prevPage,
            totalPages: totalPages,
            pages: pages,
        });

    } catch (error) {
        console.error('Error fetching order details by order id:', error);
        res.status(500).send('Internal Server Error');
    }
}

// Khởi tạo 1 số dữ liệu mẫu để chạy chương trình
async function initData() {
    // Trước khi khởi tạo dữ liệu mẫu thì ta cần xóa các dữ liệu hiện có
    await OrderDetail.deleteMany()

    let orderdetail3 = new OrderDetail({
        orderId: "1012202309404109090000000001",

```

```
        barcode: "00000000",
        productName: "Iphone 15 Pro Max 256Gb",
        price: 40990000,
        amount: 1,
        totalPrice: 40990000,
    });

    await orderdetail3.save()

    let orderdetail4 = new OrderDetail({
        orderId: "1012202309404109090000000002",
        barcode: "00112233",
        productName: "Xiaomi Redmi Note 12 4G",
        price: 5590000,
        amount: 1,
        totalPrice: 5590000,
    });

    await orderdetail4.save()

    let orderdetail5 = new OrderDetail({
        orderId: "0912202309410309090000000001",
        barcode: "11111111",
        productName: "Ipad Air 7",
        price: 25590000,
        amount: 1,
        totalPrice: 25590000,
    });

    await orderdetail5.save()

    let orderdetail6 = new OrderDetail({
        orderId: "021220230941200909111111001",
        barcode: "55556666",
        productName: "Ốp lưng MagSafe iPhone 14",
        price: 315000,
```

```

        amount: 1,
        totalPrice: 315000,
    });

    await orderdetail6.save()

    let orderdetail7 = new OrderDetail({
        orderId: "041220230942120909222222001",
        barcode: "11223344",
        productName: "Iphone 15 Plus 256Gb",
        price: 22000000,
        amount: 1,
        totalPrice: 22000000,
    });

    await orderdetail7.save()

    let orderdetail8 = new OrderDetail({
        orderId: "251120230942340909111111001",
        barcode: "77778888",
        productName: "Ốp lưng Magsafe iPhone 15",
        price: 621000,
        amount: 1,
        totalPrice: 621000,
    });

    await orderdetail8.save()
}

module.exports = {
    getOrderDetailById,
    getOrderDetail,
    initData
};

```

3.1.4 Giải thích code của phần *OrderDetailController*

Mô tả chức năng của các module trong OrderController:

- Hàm `initData`:
 - Chức năng: khởi tạo dữ liệu mẫu chi đặt hàng để chạy chương trình.
 - Thực hiện:
 - Trước khi khởi tạo, xóa tất cả chi tiết đơn đặt hàng hiện có.
 - Tạo và lưu hai đơn đặt hàng mẫu sử dụng model `OrderDetail`.
- Hàm `getOrderDetail`:
 - Chức năng: lấy danh sách chi tiết đơn đặt hàng và render trang 'detail-order' với thông tin chi tiết đơn hàng và thông báo.
 - Thực hiện:
 - Gọi hàm `getOrderDetails` để lấy danh sách chi tiết đơn đặt hàng.
 - Render trang 'detail-order' với danh sách chi tiết đơn đặt hàng, và các thông báo success và error.
- Hàm `getOrderDetails`:
 - Chức năng: lấy danh sách chi tiết đơn đặt hàng từ cơ sở dữ liệu.
 - Thực hiện:
 - Sử dụng `OrderDetail.find()` để lấy danh sách chi tiết đơn đặt hàng từ MongoDB.
 - Trả về danh sách chi tiết đơn đặt hàng.
 - Nếu có lỗi, throw một lỗi với thông báo lỗi.

3.2 Chức năng gửi mail

3.2.1 Code của phần MailController

```
const nodeMailer = require('nodemailer')
const emailExistence = require('email-existence');

function isEmail(email) {
  const regex = /^[a-zA-Z0-9._%+-]+@gmail\.com$/;
  return regex.test(email);
}

function checkEmailExistence(email) {
  return new Promise((resolve, reject) => {
    emailExistence.check(email, (error, exists) => {
      if (error) {
        reject(error);
      } else {
        resolve(exists);
      }
    });
  });
}

function sendMail(to, subject, content) {
  const transport = nodeMailer.createTransport({
    host: process.env.MAIL_HOST,
    port: process.env.MAIL_PORT,
    secure: false,
    auth: {
      user: process.env.MAIL_USERNAME,
      pass: process.env.MAIL_PASSWORD,
    }
  });

  const options = {
    from: process.env.MAIL_FROM_ADDRESS,
    to: to,
    subject: subject,
    html: content
  }

  return transport.sendMail(options);
}

module.exports = {
```

```

    sendMail,
    checkEmailExistence,
    isEmail
  };

```

3.2.2 Giải thích code của phần MailController

Mô tả module EmailService:

- Hàm isEmail Function:
 - Chức năng: kiểm tra xem một địa chỉ email có đúng định dạng của Gmail hay không.
 - Thực hiện:
 - Sử dụng biểu thức chính quy để so sánh địa chỉ email với định dạng Gmail.
 - Trả về true nếu đúng định dạng, ngược lại trả về false.
- Hàm checkEmailExistence:
 - Chức năng: kiểm tra sự tồn tại của một địa chỉ email bằng cách thực hiện kiểm tra kết nối tới máy chủ email của địa chỉ đó.
 - Thực hiện:
 - Trả về một Promise, trong đó:
 - Nếu có lỗi trong quá trình kiểm tra, reject với lỗi tương ứng.
 - Nếu kiểm tra thành công, resolve với kết quả exists là true nếu địa chỉ email tồn tại, false nếu không tồn tại.
- Hàm sendMail:
 - Chức năng: gửi email sử dụng thông tin cấu hình được cung cấp.
 - Thực hiện:
 - Sử dụng thư viện nodemailer để tạo một đối tượng transport với thông tin cấu hình máy chủ email.
 - Tạo một đối tượng options chứa thông tin về email gửi và nội dung email.
 - Sử dụng transport.sendMail để gửi email với các tùy chọn đã được thiết lập.

- Trả về Promise của quá trình gửi email.

3.3 Chức năng quản lý tài khoản

3.3.1 Code của phần AccountController

```
let Account = require("../models/account");
let mailController = require('./MailController')
let multiparty = require('multiparty') // upload file
let fsx = require('fs-extra'); // upload file
const path = require('path');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

// Load user lên trang quản lí
function getAccountManagementPage(req, res) {
  Account.find()
    .lean() // convert Mongoose Object Array thành Javascript Object Array
    .then(accounts => {
      const ITEMS_PER_PAGE = 8; // Số lượng item mỗi trang
      const page = parseInt(req.query.page) || 1; // Lấy số trang hiện tại
      const nextPage = page + 1;
      const prevPage = page - 1;
      const skip = (page - 1) * ITEMS_PER_PAGE;

      // Lọc ra các account mà không phải là admin
      const accountsNotAdmin = accounts.filter(a => a.email !==
"admin@gmail.com");
      const totalAccountsNotAdmin = accountsNotAdmin.length;
      const totalPages = Math.ceil(totalAccountsNotAdmin /
ITEMS_PER_PAGE);
      const pages = Array.from({ length: totalPages }, (_, index) => index
+ 1);

      const paginatedAccountsNotAdmin = accountsNotAdmin.slice(skip, skip
+ ITEMS_PER_PAGE);
```

```

        res.render('accountManagement', {
            layout: "admin",
            accounts: paginatedAccountsNotAdmin,
            currentPage: page,
            nextPage: nextPage,
            prevPage: prevPage,
            totalPages: totalPages,
            pages: pages
        });
    })
}

// Đăng ký
async function addAccount(req, res) {
    // Kiểm tra tính hợp lệ của dữ liệu
    if(req.body.email === "" || req.body.fullname === "") {
        req.flash("error", "Vui lòng không bỏ trống thông tin");
        return res.render("signup", {error: req.flash("error"), email:
req.body.email, fullname: req.body.fullname});
    }

    if(!mailController.isEmail(req.body.email)) {
        req.flash("error", "Email không hợp lệ");
        return res.render("signup", {error: req.flash("error"), email:
req.body.email, fullname: req.body.fullname});
    }

    if((await mailController.checkEmailExistence(req.body.email)) === false)
    {
        req.flash("error", "Địa chỉ email không tồn tại");
        return res.render("signup", {error: req.flash("error"), email:
req.body.email, fullname: req.body.fullname});
    }

    let account = new Account({
        email: req.body.email,

```

```

        password: (req.body.email).split("@")[0], // mật khẩu khi mới tạo sẽ
        là username (phần trước dấu @)
        fullname: req.body.fullname,
        profilePicture: "default-avatar.png",
        activateStatus: 0, // tài khoản mới tạo mặc định chưa được activate
        (tức là chưa click vào đường dẫn trong email)
        isNewUser: 1, // tài khoản mới tạo mặc định là user mới
        lockedStatus: 0 // tài khoản mới tạo mặc định chưa bị khóa
    });

    account.save()
    .then(newAccount => {
        resendEmail(req, res);

        req.flash("success", "Đăng ký tài khoản thành công. Vui lòng kiểm
        tra email của bạn.");
        res.render("signup", {success: req.flash("success")});
    })
    .catch(error => {
        req.flash("error", "Email này đã tồn tại");
        res.render("signup", {error: req.flash("error"), email:
        req.body.email, fullname: req.body.fullname});
    });
}

// Đăng nhập
function findAccount(req, res) {
    let username = req.body.username;
    let password = req.body.password;

    if(username === "" || password === "") {
        req.flash("error", "Vui lòng không bỏ trống thông tin")
        return res.render("login", {error: req.flash("error"), username:
        username, password: password})
    }
}

```

```

let email = username + "@gmail.com";

Account.findOne({
  email: email,
  password: password
})
.then(async account => {
  if(!account) {
    req.flash("error", "Tài khoản hoặc mật khẩu không chính xác")
    return res.render("login", {error: req.flash("error"), username:
username, password: password, token: req.body.token})
  }

  if(account.lockedStatus === 1) {
    req.flash("error", "Tài khoản của bạn đã bị khóa.")
    return res.render("login", {error: req.flash("error")})
  }

  // User chưa kích hoạt tài khoản
  if(account.activateStatus === 0) {
    // Người dùng KHÔNG truy cập trang login thông qua đường link
    trong email
    if(!req.body.token || !bcrypt.compareSync(email,
req.body.hashEmail)) {
      req.flash("error", "Vui lòng nhấn vào đường link được gửi
đến email của bạn.")
      return res.render("login", {error: req.flash("error")})
    }

    await Account.updateOne({email: email}, {$set: {activateStatus:
1}}, { new: true })

    req.session.email = email;
    return res.redirect("changepwd_logout")
  }

```

```

// user mới
if(account.isNewUser === 1)
    return res.redirect("changepwd_logout")

req.session.email = email;

if(email === "admin@gmail.com")
    res.redirect("/product-management")
else
    res.redirect("/")
})
.catch(error => {
    req.flash("error", "Có lỗi xảy ra trong quá trình đăng nhập. Vui lòng thử lại sau.")
    res.render("login", {error: req.flash("error"), username: username, password: password})
});
}

// Load trang profile dựa vào session
function getProfilePage(req, res) {
    Account.findOne({
        email: req.session.email,
    })
    .then(account => {
        let options = {
            email: account.email,
            fullname: account.fullname,
            profilePicture: account.profilePicture,
            success: req.flash("success"),
            error: req.flash("error")
        };

        res.render("profile", options)
    })
}

```

```

// Cập nhật avatar
function changeProfilePicture(req, res) {
  let form = new multiparty.Form()

  form.parse(req, (error, data, files) => {
    if (error) {
      req.flash("error", "Thay đổi ảnh thất bại.")
      return res.redirect('profile')
    }

    let file = files.file[0];

    // Validate file
    if(file.size > 1048576) {
      req.flash("error", "Không chấp nhận ảnh có kích thước lớn hơn
1MB.")
      return res.redirect('profile')
    }

    // Đổi tên file là username + extension
    let newFileName = (req.session.email).split("@")[0] +
path.extname(file.originalFilename);

    // Lưu file đã được upload vào server
    let tempPath = file.path;
    let savePath = path.join(__dirname, "../public/uploads/avatars/",
newFileName);

    fsx.copy(tempPath, savePath, (err) => {
      if (err) {
        req.flash("error", "Thay đổi ảnh thất bại.")
        return res.redirect('profile')
      }
    });
  });
}

```

```

    // Cập nhật lại giá trị của profilePicture trong database
    Account.updateOne({email: req.session.email}, {$set:
{profilePicture: newFileName}}, { new: true })
    .then(updatedAccount => {
        if (!updatedAccount)
            req.flash("error", "Thay đổi ảnh thất bại.")
        else
            req.flash("success", "Thay đổi ảnh thành công.")

        res.redirect('profile')
    })
    .catch(error => {
        req.flash("error", "Thay đổi ảnh thất bại.")
        res.redirect('profile')
    });
})
}

// Đổi mật khẩu
function changePassword(req, res) {
    Account.findOne({
        email: req.session.email,
    })
    .then(async account => {
        let currentPassword = req.body.currentPassword;
        let newPassword = req.body.newPassword;
        let confirmPassword = req.body.confirmPassword;

        if(currentPassword === "" || newPassword === "" || confirmPassword
=== "")
            req.flash("error", "Vui lòng không bỏ trống thông tin")
        else if(currentPassword !== account.password)
            req.flash("error", "Mật khẩu hiện tại không chính xác");
        else if(newPassword !== confirmPassword)
            req.flash("error", "Nhập lại mật khẩu mới không chính xác");
        else {

```

```

        // Cập nhật lại mật khẩu mới trong database
        await Account.updateOne({email: req.session.email}, {$set:
{password: newPassword}}, { new: true })
        .then(updatedAccount => {
            if (!updatedAccount)
                req.flash("error", "Đổi mật khẩu thất bại.")
            else
                req.flash("success", "Đổi mật khẩu thành công. Vui lòng
đăng nhập lại.");
        })
        .catch(error => {
            req.flash("error", "Đổi mật khẩu thất bại.")
        });
    }

    let options = {
        currentPassword: currentPassword,
        newPassword: newPassword,
        confirmPassword: confirmPassword,
        error: req.flash("error"),
        success: req.flash("success")
    };

    res.json(options)
})
}

```

// Khởi tạo 1 số dữ liệu mẫu để chạy chương trình

```

async function initData() {
    // Trước khi khởi tạo dữ liệu mẫu thì ta cần xóa các dữ liệu hiện có
    await Account.deleteMany()

    // Tài khoản admin
    let account = new Account({
        email: "admin@gmail.com",
        password: "admin",
    })
}

```



```
        fullname: "Administrator",
        profilePicture: "default-avatar.png",
        activateStatus: 1,
        isNewUser: 0,
        lockedStatus: 0
    });

    await account.save()

    let account2 = new Account({
        email: "nghiem7755@gmail.com",
        password: "nghiem7755",
        fullname: "Nguyễn Khắc Nghiêm",
        profilePicture: "default-avatar.png",
        activateStatus: 1,
        isNewUser: 1,
        lockedStatus: 0
    });

    await account2.save()

    let account3 = new Account({
        email: "anhtri000@gmail.com",
        password: "anhtri000",
        fullname: "Lê Phạm Anh Trí",
        profilePicture: "default-avatar.png",
        activateStatus: 1,
        isNewUser: 0,
        lockedStatus: 0
    });

    await account3.save()

    let account4 = new Account({
        email: "caonguyenbinh12@gmail.com",
        password: "caonguyenbinh12",
```

```

        fullname: "Cao Nguyên Bình",
        profilePicture: "default-avatar.png",
        activateStatus: 0,
        isNewUser: 1,
        lockedStatus: 0
    });

    await account4.save()
}

// Load profile user theo email tại trang quản lí
function getProfileByEmail(req, res) {
    Account.findOne({
        email: req.params.email,
    })
    .then(account => {
        let options = {
            layout: "admin",
            email: account.email,
            fullname: account.fullname,
            profilePicture: account.profilePicture,
            success: req.flash("success"),
            error: req.flash("error")
        };

        res.render("profileid", options)
    })
}

function lockUser(req, res) {
    Account.findOne({
        email: req.body.email,
    })
    .then(async account => {
        let updatedField;

```

```

    if(account.lockedStatus === 0) {
      updatedField = {
        $set: {
          lockedStatus: 1
        }
      }
    }
    else {
      updatedField = {
        $set: {
          lockedStatus: 0
        }
      }
    }
  }

  await Account.updateOne({email: req.body.email}, updatedField, {
new: true })
    .then(updatedAccount => {
      res.end();
    })
  })
}

function resendEmail(req, res) {
  let email = req.body.email;
  let subject = "Xác thực tài khoản";

  // Tạo token với thời gian hết hạn là 60 giây
  const token = jwt.sign({ email }, process.env.JWT_SECRET, { expiresIn:
60 });

  // Sử dụng bcrypt để tạo hash của email
  const hashedEmail = bcrypt.hashSync(email, 3);

  // Tạo đường link với token và hash của email

```

```

    let content = `<a
href="${process.env.APP_URL}/login?token=${token}&hashedEmail=${hashedEmail}"
"> Vui lòng nhấn vào đây để hoàn tất thủ tục tài khoản</a>`;
    mailController.sendMail(email, subject, content);
}

function getChangePwdLoginPage(req, res) {
    Account.findOne({
        email: req.session.email
    })
    .then(account => {
        if(account.isNewUser === 1)
            res.render('changepwd_logout', {layout: null, email:
req.session.email});
        else
            res.redirect('/login');
    })
}

// Đổi mật khẩu không cần pass cũ và cập nhật isNewUser
function changePwdNoPassOld(req, res) {
    Account.findOne({
        email: req.session.email,
    })
    .then(async account => {
        let currentPassword = account.password;
        let newPassword = req.body.newPassword;
        let confirmPassword = req.body.confirmPassword;
        let isNewUser = account.isNewUser;

        if(newPassword === "" || confirmPassword === "")
            req.flash("error", "Vui lòng không bỏ trống thông tin")
        else if(newPassword !== confirmPassword)
            req.flash("error", "Nhập lại mật khẩu mới không chính xác");
        else {
            let updatedField;

```

```

        updatedField = {
          $set: {
            isNewUser: 0,
            password: newPassword
          }
        }

        // Cập nhật lại mật khẩu mới và isNewUser trong database
        await Account.updateOne({email: req.session.email},
updatedField, { new: true })
        .then(updatedAccount => {
          if (!updatedAccount)
            req.flash("error", "Đổi mật khẩu thất bại.")
          else
            req.flash("success", "Đổi mật khẩu thành công. Vui lòng
đăng nhập lại.");
        })
        .catch(error => {
          req.flash("error", "Đổi mật khẩu thất bại.")
        });
      }

      let options = {
        currentPassword: currentPassword,
        newPassword: newPassword,
        confirmPassword: confirmPassword,
        isNewUser: isNewUser,
        error: req.flash("error"),
        success: req.flash("success")
      };

      res.json(options)
    })
  }

```

```
module.exports = {
  getAccountManagementPage,
  addAccount,
  getProfilePage,
  getProfileByEmail,
  changeProfilePicture,
  findAccount,
  changePassword,
  initData,
  lockUser,
  resendEmail,
  changePwdNoPassOld,
  getChangePwdLogPage
};
```

3.3.2 Giải thích code của phần AccountController

Mô tả module AccountController

- Hàm getAccountManagementPage:
 - Chức năng: load danh sách tài khoản và hiển thị trên trang quản lý.
 - Thực hiện:
 - Sử dụng model Account để lấy danh sách tài khoản từ cơ sở dữ liệu.
 - Lọc bỏ tài khoản admin.
 - Render trang accountManagement với danh sách tài khoản đã lọc.
- Hàm addAccount:
 - Chức năng: đăng ký tài khoản mới.
 - Thực hiện:
 - Kiểm tra tính hợp lệ của dữ liệu đầu vào như email, tên đầy đủ.
 - Kiểm tra email có tồn tại và có định dạng hợp lệ không.
 - Tạo một tài khoản mới với mật khẩu mặc định (username là phần trước dấu @ của email).
 - Lưu tài khoản mới vào cơ sở dữ liệu.

- Gửi email xác nhận đăng ký và render trang signup với thông báo thành công.
- Hàm findAccount:
 - Chức năng: xác thực thông tin đăng nhập và điều hướng người dùng đến trang tương ứng.
 - Thực hiện:
 - Kiểm tra tính hợp lệ của tên đăng nhập và mật khẩu.
 - Nếu tài khoản chưa được kích hoạt, kiểm tra token và hash của email để xác nhận đường link trong email.
 - Nếu thông tin đăng nhập chính xác, thiết lập session và điều hướng người dùng đến trang tương ứng (changepwd_logout, trang chính, hoặc trang quản lý sản phẩm).
- Hàm getProfilePage:
 - Chức năng: load thông tin tài khoản của người dùng và hiển thị trên trang cá nhân.
 - Thực hiện:
 - Sử dụng session để xác định người dùng hiện tại.
 - Render trang profile với thông tin tài khoản.
- Hàm changeProfilePicture:
 - Chức năng: thay đổi ảnh đại diện của người dùng.
 - Thực hiện:
 - Sử dụng thư viện multipart để xử lý upload file.
 - Kiểm tra kích thước file và lưu file vào thư mục uploads/avatars.
 - Cập nhật tên ảnh đại diện trong cơ sở dữ liệu và render trang profile với thông báo thành công hoặc lỗi.
- Hàm changePassword:
 - Chức năng: thay đổi mật khẩu của người dùng.
 - Thực hiện:
 - Kiểm tra tính hợp lệ của thông tin mật khẩu.

- Cập nhật mật khẩu mới trong cơ sở dữ liệu và trả về kết quả (thông báo thành công hoặc lỗi) dưới dạng JSON.
- Hàm initData:
 - Chức năng: khởi tạo dữ liệu mẫu trong cơ sở dữ liệu.
 - Thực hiện:
 - Xóa dữ liệu hiện có và thêm tài khoản admin và một số tài khoản mẫu khác vào cơ sở dữ liệu.
- Hàm getProfileByEmail:
 - Chức năng: load thông tin tài khoản dựa trên địa chỉ email và hiển thị trang cá nhân.
 - Thực hiện:
 - Sử dụng địa chỉ email từ yêu cầu và render trang profileid với thông tin tài khoản.
- Hàm lockUser:
 - Chức năng: khóa hoặc mở khóa tài khoản người dùng.
 - Thực hiện:
 - Xác định tài khoản dựa trên địa chỉ email từ yêu cầu và cập nhật trạng thái khóa trong cơ sở dữ liệu.
- Hàm resendEmail:
 - Chức năng: gửi lại email xác nhận đăng ký với đường link kích hoạt tài khoản.
 - Thực hiện:
 - Sử dụng thư viện jwt để tạo token với hạn chế thời gian.
 - Sử dụng bcrypt để tạo hash của địa chỉ email.
 - Gửi email xác nhận đăng ký với đường link chứa token và hash của email.
- Hàm getChangePwdLoginPage:
 - Chức năng: load trang đổi mật khẩu khi người dùng đăng nhập lần đầu.

- Thực hiện:
 - Kiểm tra xem người dùng có phải là người dùng mới hay không và render trang changepwd_logout hoặc điều hướng đến trang đăng nhập.
- Hàm changePwdNoPassOld:
 - Chức năng: đổi mật khẩu không yêu cầu mật khẩu cũ khi người dùng đăng nhập lần đầu.
 - Thực hiện:
 - Kiểm tra tính hợp lệ của thông tin mật khẩu.
 - Cập nhật mật khẩu mới và trạng thái isNewUser trong cơ sở dữ liệu và trả về kết quả (thông báo thành công hoặc lỗi) dưới dạng JSON.

3.4 Chức năng quản lý khách hàng

3.4.1 Code của phần CustomerController

```
let Customer = require("../models/customer");

async function addCustomer(req, res) {
  if(req.body.phone === "" || req.body.fullname === "Không tìm thấy khách hàng" || req.body.address === "Không tìm thấy khách hàng" || req.body.fullname === "" || req.body.address === "") {
    req.flash("error", "Thông tin khách hàng không hợp lệ");
    return res.render("product-payment", { error: req.flash("error") });
  }

  let customer = new Customer({
    phone: req.body.phone,
    fullname: req.body.fullname,
    address: req.body.address
  });

  const existingCustomer = await Customer.findOne({ phone: req.body.phone });

  if (existingCustomer) {
    return res.redirect("invoice");
  }

  customer.save()
    .then(newCustomer => {
      res.redirect("invoice");
    })
    .catch(error => {
      req.flash("error", "Người dùng đã tồn tại");//tồn tại rồi thì in hóa đơn luôn
      res.render("product-payment", {error: req.flash("error"), email: req.body.email, fullname: req.body.fullname, address: req.body.address});
    });
}
```

```

}

function findCustomer(req, res) {
  let phone = req.params.phone;

  Customer.findOne({
    phone: phone
  })
  .then(customer => {
    if(!customer)
      return res.json({ fullname: "Không tìm thấy khách hàng",
address: "Không tìm thấy khách hàng" })

    res.json({ fullname: customer.fullname, address: customer.address })
  })
  .catch(error => {
    res.json({ fullname: "Lỗi không tìm thấy khách hàng", address: "Lỗi
không tìm thấy khách hàng" })
  });
}

async function getStaffPaymentPage(req, res) {
  const ITEMS_PER_PAGE = 10; // Số lượng item mỗi trang
  const page = parseInt(req.query.page) || 1; // Lấy số trang hiện tại
  const nextPage = page + 1;
  const prevPage = page - 1;
  const skip = (page - 1) * ITEMS_PER_PAGE;

  try {
    const customers = await getCustomers();
    const totalCustomers = customers.length;
    const totalPages = Math.ceil(totalCustomers / ITEMS_PER_PAGE);
    const pages = Array.from({ length: totalPages }, (_, index) => index
+ 1);

```

```

        const paginatedCustomers = customers.slice(skip, skip +
ITEMS_PER_PAGE);

    res.render('staff-payment', {
        customers: paginatedCustomers,
        success: req.flash("success"),
        error: req.flash("error"),
        email: req.session.email,
        currentPage: page,
        nextPage: nextPage,
        prevPage: prevPage,
        totalPages: totalPages,
        pages: pages
    });
} catch (error) {
    console.error('Error fetching products:', error);
    res.status(500).send('Internal Server Error');
}
}

async function getCustomers() {
    try {
        let customers = await Customer.find().lean();
        return customers;
    } catch (error) {
        throw new Error("Đã xảy ra lỗi khi lấy dữ liệu từ cơ sở dữ liệu.");
    }
}

// Khởi tạo 1 số dữ liệu mẫu để chạy chương trình
async function initData() {
    // Trước khi khởi tạo dữ liệu mẫu thì ta cần xóa các dữ liệu hiện có
    await Customer.deleteMany()

    let customer = new Customer({
        phone: "0909000000",

```

```
        fullname: "Nguyễn Văn A",
        address: "Cà Mau"
    });

    await customer.save()

    let customer2 = new Customer({
        phone: "0909111111",
        fullname: "Nguyễn Văn B",
        address: "Yên Bái"
    });

    await customer2.save()

    let customer3 = new Customer({
        phone: "0909222222",
        fullname: "Nguyễn Văn C",
        address: "Hà Nội"
    });

    await customer3.save()

    let customer4 = new Customer({
        phone: "0909333333",
        fullname: "Nguyễn Văn D",
        address: "Hà Nội"
    });

    await customer4.save()

    let customer5 = new Customer({
        phone: "0909444444",
        fullname: "Nguyễn Văn E",
        address: "Hà Nội"
    });
```

```

    await customer5.save()
}

module.exports = {
  addCustomer,
  findCustomer,
  getStaffPaymentPage,
  initData
};

```

3.4.2 Giải thích code của phần CustomerController

Mô tả module CustomerController

- Hàm addCustomer:
 - Chức năng: thêm thông tin khách hàng mới hoặc in hóa đơn nếu khách hàng đã tồn tại.
 - Thực hiện:
 - Kiểm tra tính hợp lệ của thông tin khách hàng (số điện thoại, tên, địa chỉ).
 - Tạo một đối tượng khách hàng mới.
 - Kiểm tra xem khách hàng đã tồn tại hay chưa bằng cách tìm kiếm theo số điện thoại.
 - Nếu khách hàng đã tồn tại, chuyển hướng đến trang invoice.
 - Nếu chưa tồn tại, lưu thông tin khách hàng mới vào cơ sở dữ liệu và chuyển hướng đến trang invoice.
 - Nếu có lỗi, hiển thị thông báo lỗi trên trang product-payment.
- Hàm findCustomer:
 - Chức năng: tìm kiếm thông tin khách hàng theo số điện thoại.
 - Thực hiện:
 - Lấy số điện thoại từ yêu cầu.
 - Tìm kiếm khách hàng trong cơ sở dữ liệu theo số điện thoại.

- Nếu tìm thấy, trả về thông tin khách hàng (tên và địa chỉ), ngược lại trả về thông báo không tìm thấy.
- Hàm `getStaffPaymentPage`:
 - Chức năng: load trang thanh toán cho nhân viên với danh sách khách hàng.
 - Thực hiện:
 - Lấy danh sách khách hàng từ cơ sở dữ liệu.
 - Render trang `staff-payment` với danh sách khách hàng và thông báo thành công hoặc lỗi.
- Hàm `getCustomers`:
 - Chức năng: lấy danh sách khách hàng từ cơ sở dữ liệu.
 - Thực hiện:
 - Sử dụng model `Customer` để lấy danh sách khách hàng.
 - Trả về danh sách khách hàng.
- Hàm `initData`:
 - Chức năng: khởi tạo dữ liệu mẫu về khách hàng trong cơ sở dữ liệu.
 - Thực hiện:
 - Xóa tất cả dữ liệu khách hàng hiện có.
 - Thêm một số khách hàng mẫu vào cơ sở dữ liệu.

3.5 Chức năng quản lý sản phẩm

3.5.1 Code của phần *ProductController*

```
let Product = require("../models/product");
let Customer = require("../models/customer");
let Order = require("../models/order");
let OrderDetail = require("../models/orderdetail");
let multiparty = require('multiparty') // upload file
let fsx = require('fs-extra'); // upload file
const path = require('path');

async function getProductManagementPage(req, res) {
    let products = await getProducts();
    res.render('product-management', { products, success:
req.flash("success"), error: req.flash("error") });
}

async function getHomePage(req, res) {
    let products = await getProducts();
    res.render('index', { products });
}

// Khởi tạo 1 số dữ liệu mẫu để chạy chương trình
async function initData() {
    // Trước khi khởi tạo dữ liệu mẫu thì ta cần xóa các dữ liệu hiện có
    await Product.deleteMany()

    let product = new Product({
        barcode: "00000000",
        productName: "Iphone 15 Pro Max 256Gb",
        importPrice: 20000000,
        retailPrice: 30000000,
        category: "Điện thoại",
        creationDate: "03/08/2023",
        image: "iphone15.png"
    });
}
```



```
await product.save()

let product2 = new Product({
  barcode: "11111111",
  productName: "Ipad Air 7",
  importPrice: 15000000,
  retailPrice: 22000000,
  category: "Máy tính bảng",
  creationDate: "15/12/2022",
  image: "ipadair7.png"
});

await product2.save()

let product3 = new Product({
  barcode: "33333333",
  productName: "Iphone 14 Pro 128Gb",
  importPrice: 15000000,
  retailPrice: 22000000,
  category: "Điện thoại",
  creationDate: "15/12/2022",
  image: "iphone14.png"
});

await product3.save()

let product4 = new Product({
  barcode: "44444444",
  productName: "Iphone 15 Pro 64Gb",
  importPrice: 15000000,
  retailPrice: 22000000,
  category: "Điện thoại",
  creationDate: "15/12/2022",
  image: "iphone15.png"
});
```

```

    await product4.save()

    let product5 = new Product({
      barcode: "55555555",
      productName: "Iphone 14 Pro 1Tb",
      importPrice: 15000000,
      retailPrice: 22000000,
      category: "Điện thoại",
      creationDate: "15/12/2022",
      image: "iphone14.png"
    });

    await product5.save()
  }

  async function addProduct(req, res) {
    let form = new multiparty.Form();

    form.parse(req, async (error, data, files) => {
      if (error) {
        console.log("Error File upload: " + error);
        const products = await Product.find().lean();
        req.flash("error", "Đã xảy ra lỗi khi xử lý upload file");
        return res.render("product-management", { error :
req.flash("error"), products });
      }

      if (data.barcode[0] === '' || data.productName[0] === '' ||
data.importPrice[0] <= 0 || data.retailPrice[0] <= 0 || data.category[0] ===
'' || data.creationDate[0] === '' || files.image[0].originalFilename === '')
{
        const products = await Product.find().lean();
        req.flash("error", "Vui lòng nhập đầy đủ thông tin và chọn file
ảnh.");

```

```

        return res.render("product-management", { error :
req.flash("error"), products });
    }

    try {
        let file = files.image[0];
        let tempPath = file.path;
        let savePath = path.join(__dirname,
"../public/uploads/products", file.originalFilename);

        await fsx.copy(tempPath, savePath);

        let formattedDate = formatDate(data.creationDate[0]);

        let product = new Product({
            barcode: data.barcode[0],
            productName: data.productName[0],
            importPrice: data.importPrice - 0, // Conversion to float
            retailPrice: data.retailPrice - 0, // Conversion to float
            category: data.category[0],
            creationDate: formattedDate,
            image: file.originalFilename
        });

        product.save()
            .then(newProduct => {
                req.flash("success", "Thêm sản phẩm thành công.");
                return res.redirect("/product-management");
            })
            .catch(error => {
                req.flash("error", "Sản phẩm này đã tồn tại hoặc có lỗi
khi lưu.");

                res.redirect("/product-management");
            });
    } catch (error) {
        req.flash("error", "Đã xảy ra lỗi khi xử lý thêm sản phẩm.");
    }

```

```

        return res.redirect("/product-management");
    }
});
}

async function editProduct(req, res) {
    let form = new multiparty.Form();

    form.parse(req, async (error, data, files) => {
        if (error) {
            console.log("Error File upload: " + error);
            const products = await Product.find().lean();
            req.flash("error", "Đã xảy ra lỗi khi xử lý upload file");
            return res.render("product-management", { error:
req.flash("error"), products });
        }

        if (data.barcode[0] === '' || data.productName[0] === '' ||
data.importPrice[0] <= 0 || data.retailPrice[0] <= 0 || data.category[0] ===
'' || data.creationDate[0] === '') {
            req.flash("error", "Vui lòng nhập đầy đủ thông tin.");
            return res.redirect("/product-management");
        }

        try {
            const barcode = data.barcode[0];

            let product = await Product.findOne({ barcode }).exec();

            if (!product) {
                req.flash("error", "Không tìm thấy sản phẩm để chỉnh sửa.");
                return res.redirect("/product-management");
            }

            product.productName = data.productName[0];
            product.importPrice = data.importPrice[0] - 0;

```

```

        product.retailPrice = data.retailPrice[0] - 0;
        product.category = data.category[0];
        product.creationDate = formatDate(data.creationDate[0]);

        let file = files.image[0];
        if (file.originalFilename !== "") {
            let tempPath = file.path;
            let savePath = path.join(__dirname,
                "../public/uploads/products", file.originalFilename);

            await fsx.copy(tempPath, savePath);
            product.image = file.originalFilename;
        }

        await product.save();

        req.flash("success", "Chỉnh sửa sản phẩm thành công.");
        return res.render("product-management", { product });
    } catch (error) {
        req.flash("error", "Đã xảy ra lỗi khi chỉnh sửa sản phẩm.");
        return res.redirect("/product-management");
    }
});
}

async function deleteProduct(req, res) {
    try {
        let barcode = req.body.barcode;
        let products;

        // Kiểm tra xem sản phẩm có từng được mua rồi hay không
        let orderDetail = await OrderDetail.findOne({barcode: barcode});

        if (orderDetail) {
            products = await Product.find().lean();

```

```

        req.flash("error", "Không thể xóa sản phẩm này vì đã được mua trước đó.");
        return res.render('product-management', { error: req.flash('error'), products });
    }

    let deletedProduct = await Product.findOneAndDelete({ barcode });

    if (deletedProduct) {
        products = await Product.find().lean();

        req.flash("success", "Xóa sản phẩm thành công.");
        res.render('product-management', { success: req.flash('success'), products });
    } else {
        req.flash("error", "Không thể xóa sản phẩm. Sản phẩm không tồn tại.");
        res.redirect('/product-management');
    }
} catch (error) {
    console.error('Error deleting product:', error);
    req.flash("error", "Lỗi trong quá trình xóa sản phẩm.");
    res.redirect('/product-management');
}

function formatDate(inputDate) {
    const parts = inputDate.split('-');
    return `${parts[2]}/${parts[1]}/${parts[0]}`;
}

async function getProducts() {
    try {
        let products = await Product.find().lean();
        return products;
    } catch (error) {

```

```

        throw new Error("Đã xảy ra lỗi khi lấy dữ liệu từ cơ sở dữ liệu.");
    }
}

async function searchProduct(req, res) {
    try {
        const barcode = req.body.barcode;
        let results;

        if (!barcode || barcode.trim() === '') {
            return res.json([]);
        } else {
            results = await Product.find({ barcode: { $regex: barcode,
$options: 'i' } }).lean();
        }
        res.json(results);
    } catch (error) {
        res.status(500).json({ error: "Đã xảy ra lỗi khi tìm kiếm sản phẩm."
});
    }
}

async function handlePayment(req, res) {
    try {
        const currentDate = new Date();
        const phone = req.body.phone;
        const formattedDate =
`${currentDate.getDate()}/${currentDate.getMonth() +
1).toString().padStart(2, '0')}/${currentDate.getFullYear()}`;
        const formattedOrderId =
`${currentDate.getDate()}${currentDate.getMonth() +
1).toString().padStart(2,
'0')}${currentDate.getFullYear()}${currentDate.getHours().toString().padSt
art(2, '0')}${currentDate.getMinutes().toString().padStart(2,
'0')}${currentDate.getSeconds().toString().padStart(2, '0')}${phone}`;

```

```

        if (req.body.phone === "" || req.body.fullname === "Không tìm thấy
khách hàng" || req.body.address === "Không tìm thấy khách hàng" ||
req.body.fullname === "" || req.body.address === "") {
            req.flash("error", "Thông tin khách hàng không hợp lệ");
            return res.render("product-payment", { error: req.flash("error")
});
        }

        const existingCustomer = await Customer.findOne({ phone:
req.body.phone });

        if (existingCustomer) {
            existingCustomer.fullname = req.body.fullname;
            existingCustomer.address = req.body.address;
            await existingCustomer.save();
        } else {
            const newCustomer = new Customer({
                phone: req.body.phone,
                fullname: req.body.fullname,
                address: req.body.address
            });
            await newCustomer.save();
        }

        const order = new Order({
            orderId: formattedOrderId,
            customerPhone: req.body.phone,
            totalPrice: req.body.totalAmountInput,
            priceGivenByCustomer: 0,
            excessPrice: 0,
            dateOfPurchase: formattedDate,
            totalAmount: req.body.totalQuantityInput
        });
        await order.save();

        const productTable = await req.body.productTable;

```



```

    const parsedProductTable = JSON.parse(productTable);

    if (parsedProductTable.length === 0) {
        req.flash("error", "Chưa có sản phẩm trong giỏ hàng");
        return res.render("product-payment", { error: req.flash("error")
    });
    }

    let orderDetails = [];
    let counter = 1;

    for (const product of parsedProductTable) {
        const uniqueFormattedOrderId =
` ${formattedOrderId}${counter.toString().padStart(3, '0')}`;

        const orderDetail = new OrderDetail({
            orderId: uniqueFormattedOrderId,
            barcode: product.barcode,
            productName: product.productName,
            price: product.retailPrice,
            amount: product.quantity,
            totalPrice: product.total
        });
        orderDetails.push(orderDetail);
        counter++;
    }

    await OrderDetail.insertMany(orderDetails);
    res.redirect("invoice");
} catch (error) {
    req.flash("error", "Không thể thanh toán hóa đơn");
    res.render("product-payment", { error: req.flash("error") });
}

module.exports = {

```

```

getProductManagementPage,
getHomePage,
initData,
addProduct,
editProduct,
deleteProduct,
searchProduct,
handlePayment
};

```

3.5.2 Giải thích code của phần *ProductController*

Mô tả module *ProductController*:

- Hàm *getProductManagementPage*:
 - Chức năng: load trang quản lý sản phẩm với danh sách sản phẩm.
 - Thực hiện:
 - Lấy danh sách sản phẩm từ cơ sở dữ liệu.
 - Render trang *product-management* với danh sách sản phẩm và thông báo thành công hoặc lỗi.
- Hàm *getHomePage*:
 - Chức năng: load trang chính (trang chủ) với danh sách sản phẩm.
 - Thực hiện:
 - Lấy danh sách sản phẩm từ cơ sở dữ liệu.
 - Render trang *index* với danh sách sản phẩm.
- Hàm *initData*:
 - Chức năng: khởi tạo dữ liệu mẫu về sản phẩm trong cơ sở dữ liệu.
 - Thực hiện:
 - Xóa tất cả dữ liệu sản phẩm hiện có.
 - Thêm một số sản phẩm mẫu vào cơ sở dữ liệu.
- Hàm *addProduct*:
 - Chức năng: thêm sản phẩm mới.
 - Thực hiện:

- Xử lý upload file hình ảnh sản phẩm.
 - Kiểm tra tính hợp lệ của thông tin sản phẩm.
 - Lưu hình ảnh vào thư mục uploads/products.
 - Tạo đối tượng sản phẩm mới và lưu vào cơ sở dữ liệu.
 - Hiện thị thông báo thành công hoặc lỗi.
- Hàm editProduct:
 - Chức năng: chỉnh sửa thông tin sản phẩm.
 - Thực hiện:
 - Xử lý upload file hình ảnh sản phẩm (nếu có).
 - Kiểm tra tính hợp lệ của thông tin sản phẩm.
 - Lưu hình ảnh mới vào thư mục uploads/products.
 - Chỉnh sửa thông tin sản phẩm trong cơ sở dữ liệu.
 - Hiện thị thông báo thành công hoặc lỗi.
- Hàm deleteProduct:
 - Chức năng: xóa sản phẩm.
 - Thực hiện:
 - Kiểm tra xem sản phẩm đã được mua trước đó hay chưa.
 - Nếu đã mua, hiện thị thông báo lỗi.
 - Nếu chưa mua, xóa sản phẩm khỏi cơ sở dữ liệu.
 - Hiện thị thông báo thành công hoặc lỗi.
- Hàm formatDate:
 - Chức năng: định dạng ngày theo định dạng "dd/mm/yyyy".
 - Thực hiện:
 - Nhận vào một chuỗi ngày trong định dạng "yyyy-mm-dd".
 - Trả về chuỗi ngày mới định dạng "dd/mm/yyyy".
- Hàm getProducts:
 - Chức năng: lấy danh sách sản phẩm từ cơ sở dữ liệu.
 - Thực hiện:
 - Sử dụng model Product để lấy danh sách sản phẩm.

- Trả về danh sách sản phẩm.
- Hàm searchProduct:
 - Chức năng: tìm kiếm sản phẩm theo mã vạch.
 - Thực hiện:
 - Nhận vào mã vạch từ yêu cầu.
 - Tìm kiếm sản phẩm trong cơ sở dữ liệu theo mã vạch.
 - Trả về kết quả tìm kiếm.
- Hàm handlePayment:
 - Chức năng: xử lý thanh toán hóa đơn.
 - Thực hiện:
 - Tạo đơn hàng mới và lưu vào cơ sở dữ liệu.
 - Cập nhật thông tin khách hàng (thêm mới nếu chưa tồn tại).
 - Lưu chi tiết đơn hàng (các sản phẩm được mua) vào cơ sở dữ liệu.
 - Chuyển hướng đến trang in hóa đơn.
 - Hiển thị thông báo lỗi nếu có vấn đề trong quá trình xử lý.

3.6 Chức năng phụ: phân trang

Các hàm hiện ra một trang trong hệ thống được sử dụng để hiển thị trang với chức năng phân trang cho danh sách đối tượng. Dưới đây là mô tả chung chung cho sự phân trang trong ngữ cảnh của các phần code trong hệ thống:

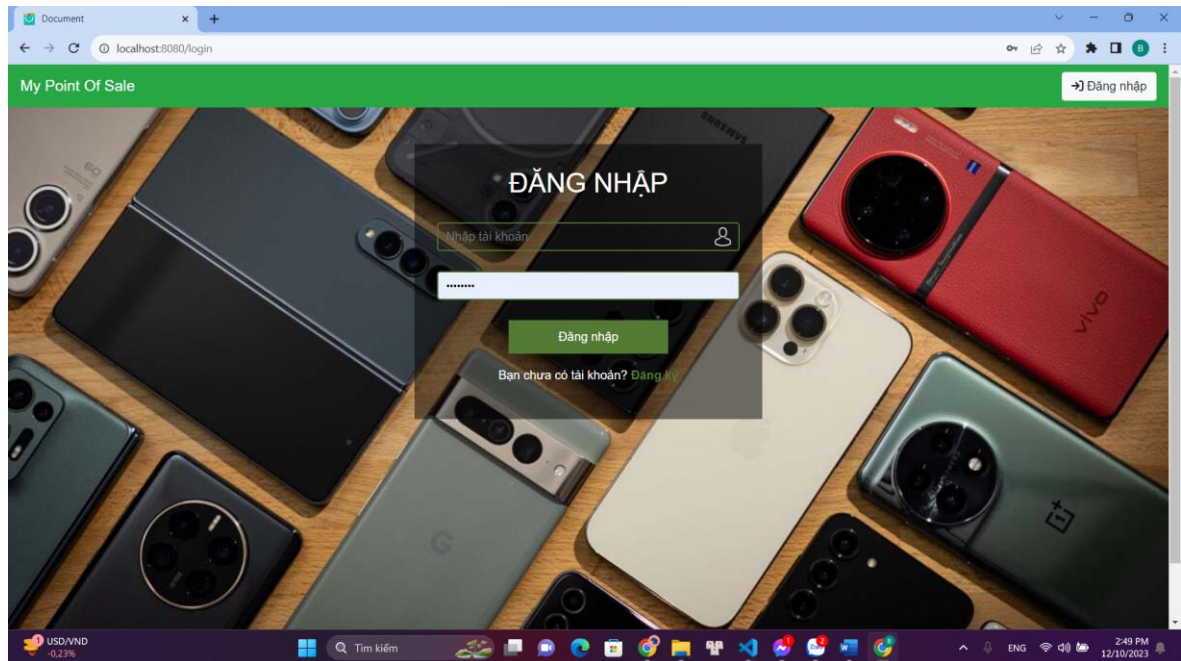
- Khai báo và truy xuất thông tin phân trang:
 - ITEMS_PER_PAGE: Số lượng đối tượng hiển thị trên mỗi trang.
 - page: Số trang hiện tại được trích xuất từ tham số truy vấn của request.
 - nextPage: Số trang tiếp theo.
 - prevPage: Số trang trước đó.
 - skip: Số lượng đối tượng được bỏ qua để đến trang hiện tại.
- Lấy danh sách đối tượng:
 - Gọi các hàm get (đối tượng) () để lấy danh sách đối tượng từ cơ sở dữ liệu. (Hàm này không được định nghĩa trong đoạn các mã)
- Xử lý phân trang:
 - Tính tổng số lượng đối tượng.
 - Tính tổng số trang (totalPages) dựa trên số lượng đối tượng và số lượng đối tượng trên mỗi trang.
 - Tạo một mảng các số trang (pages) để hiển thị trên giao diện người dùng.
- Tạo mảng đối tượng cho trang hiện tại:
 - Sử dụng slice để lấy một phần của danh sách đối tượng dựa trên skip và ITEMS_PER_PAGE.
 - Mảng này chứa danh sách đối tượng cho trang hiện tại.
- Render trang của dữ liệu:
 - Sử dụng res.render để hiển thị trang hiện tại với các thông tin sau:
 - Đối tượng: Danh sách đối tượng cho trang hiện tại.
 - success: Thông báo thành công (nếu có).
 - error: Thông báo lỗi (nếu có).

- email: Địa chỉ email của người dùng đang đăng nhập (nếu có).
 - currentPage, nextPage, prevPage: Các thông tin về trang hiện tại và các trang liền kề.
 - totalPages: Tổng số trang.
 - pages: Mảng số trang để hiển thị nút điều hướng trang trên giao diện người dùng.
- Xử lý lỗi:
 - Trong trường hợp có lỗi khi lấy dữ liệu đối tượng, in thông báo lỗi vào console và trả về một trang lỗi với mã trạng thái 500.
 - Như vậy, hàm này giúp hiển thị danh sách đối tượng trên nhiều trang để giảm độ dài của trang và tối ưu hóa trải nghiệm người dùng.

CHƯƠNG 4 – KẾT QUẢ VÀ KẾT LUẬN

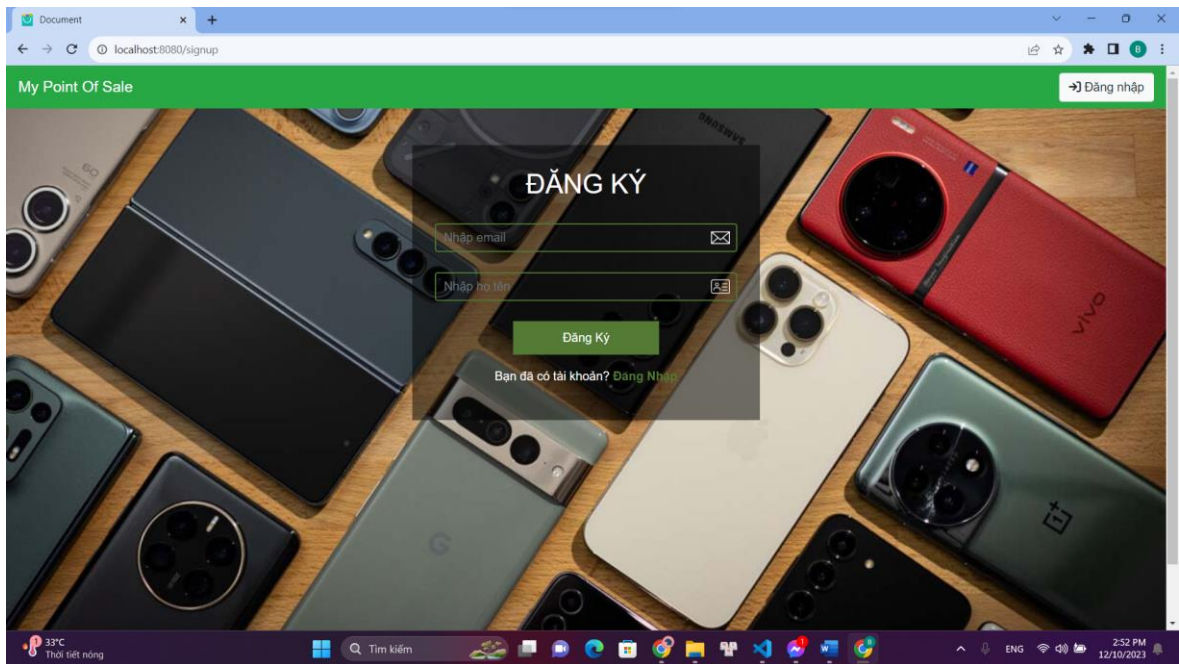
4.1 Màn hình giao diện của ứng dụng

4.1.1 Màn hình đăng nhập



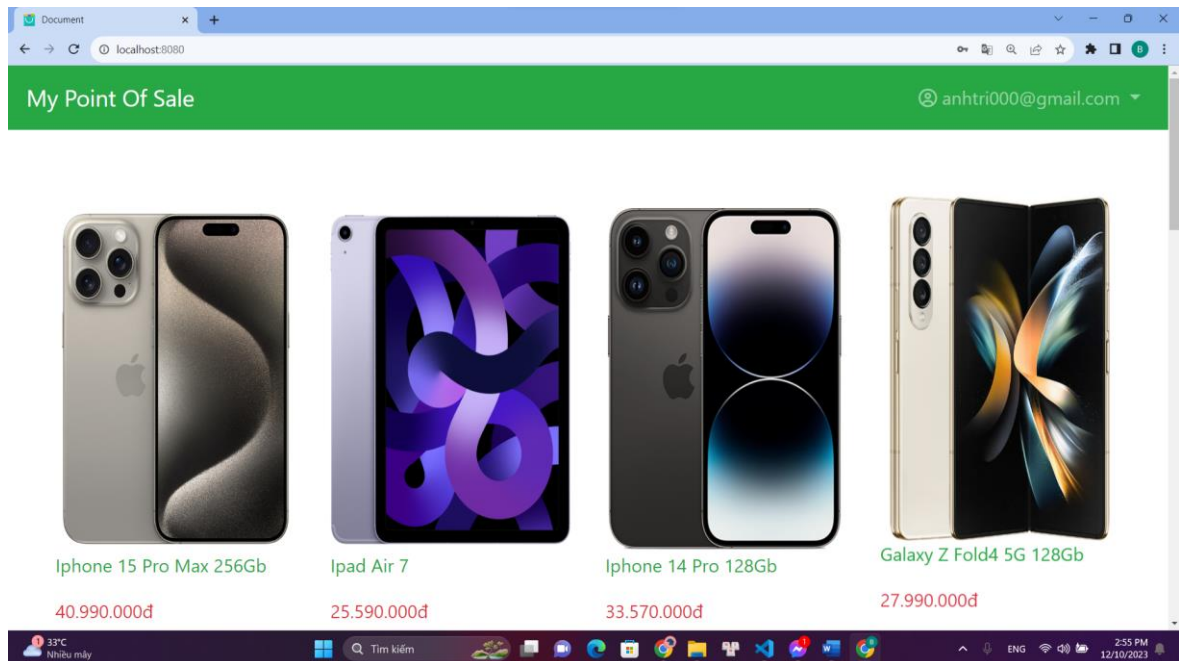
Hình 4. 1: Màn hình đăng nhập

4.1.2 Màn hình đăng ký



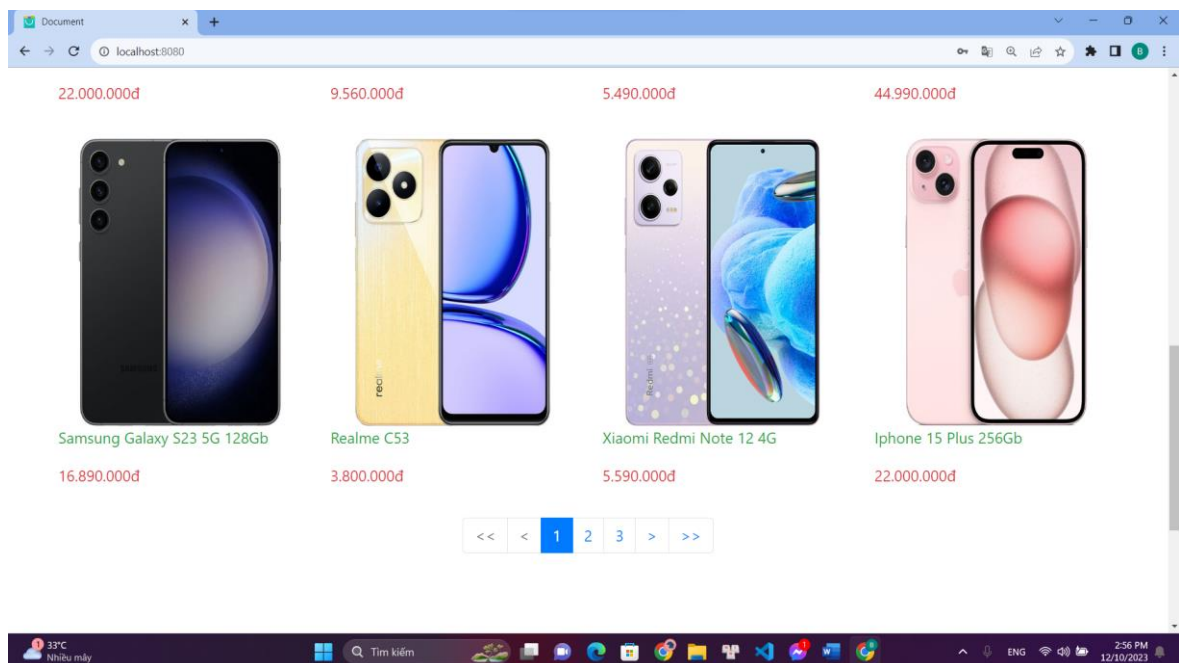
Hình 4. 2: Màn hình đăng ký

4.1.3 Màn hình trang chủ 1 với giao diện nhân viên



Hình 4. 3: Màn hình trang chủ 1 với giao diện nhân viên

4.1.4 Màn hình trang chủ 2 với giao diện nhân viên



Hình 4. 4: Màn hình trang chủ 2 với giao diện nhân viên

4.1.5 Màn hình thanh toán với giao diện nhân viên

Thanh toán

Nhập tên hoặc barcode

Barcode	Hình ảnh	Tên sản phẩm	Số lượng	Đơn giá	Tổng tiền	Xóa
00000000		Iphone 15 Pro Max 256Gb	<input type="text" value="3"/>	40990000	122970000	
11111111		Ipad Air 7	<input type="text" value="1"/>	25590000	25590000	

Tổng số lượng:
4

Tổng tiền: 148.560.000đ

Số điện thoại:

Họ và tên:

Địa chỉ:

Thanh toán

Hình 4. 5: Màn hình thanh toán với giao diện nhân viên

4.1.6 Hóa đơn bán hàng với giao diện nhân viên

Mã hóa đơn: 101220231500080942563747

Số điện thoại KH: Nhập số điện thoại

Họ tên KH:

Địa chỉ KH:

Tổng hóa đơn: 148.560.000đ

Khách đưa: Nhập số tiền khách đưa

Trả khách:

IN HÓA ĐƠN

Hình 4. 6: Hóa đơn thanh toán với giao diện nhân viên

4.1.7 Màn hình danh sách khách hàng với giao diện nhân viên

Họ và tên	Số điện thoại	Địa chỉ	Lịch sử mua hàng
Nguyễn Văn A	0909000000	Cà Mau	Xem
Nguyễn Văn B	0909111111	Yên Bái	Xem
Nguyễn Văn C	0909222222	Hà Nội	Xem
Nguyễn Văn D	0909333333	Hà Nội	Xem
Nguyễn Văn E	0909444444	Hà Nội	Xem
Cao Nguyễn Bình	0942563747	Cao Nguyễn Bình	Xem

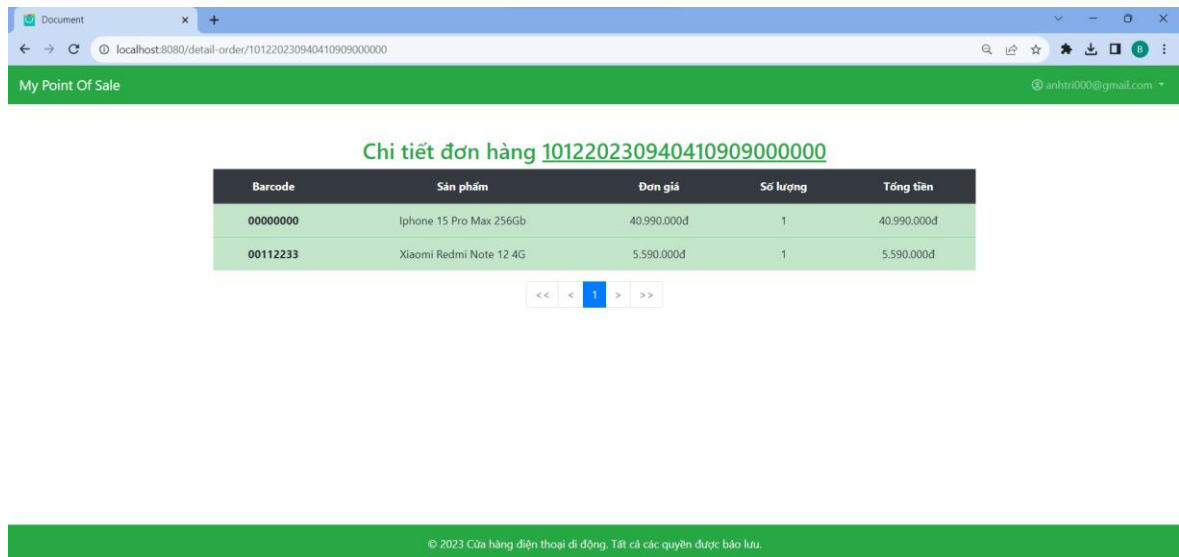
Hình 4. 7: Màn hình danh sách khách hàng với giao diện nhân viên

4.1.8 Màn hình danh sách đơn hàng với giao diện nhân viên

Mã đơn hàng	Tổng tiền	Số tiền đã trả	Số tiền được hoàn	Ngày mua	Số lượng	Chi tiết
101220230940410909000000	46.580.000đ	50.000.000đ	3.420.000đ	10/12/2023	2	Chi tiết
091220230941030909000000	25.590.000đ	26.000.000đ	410.000đ	09/12/2023	1	Chi tiết

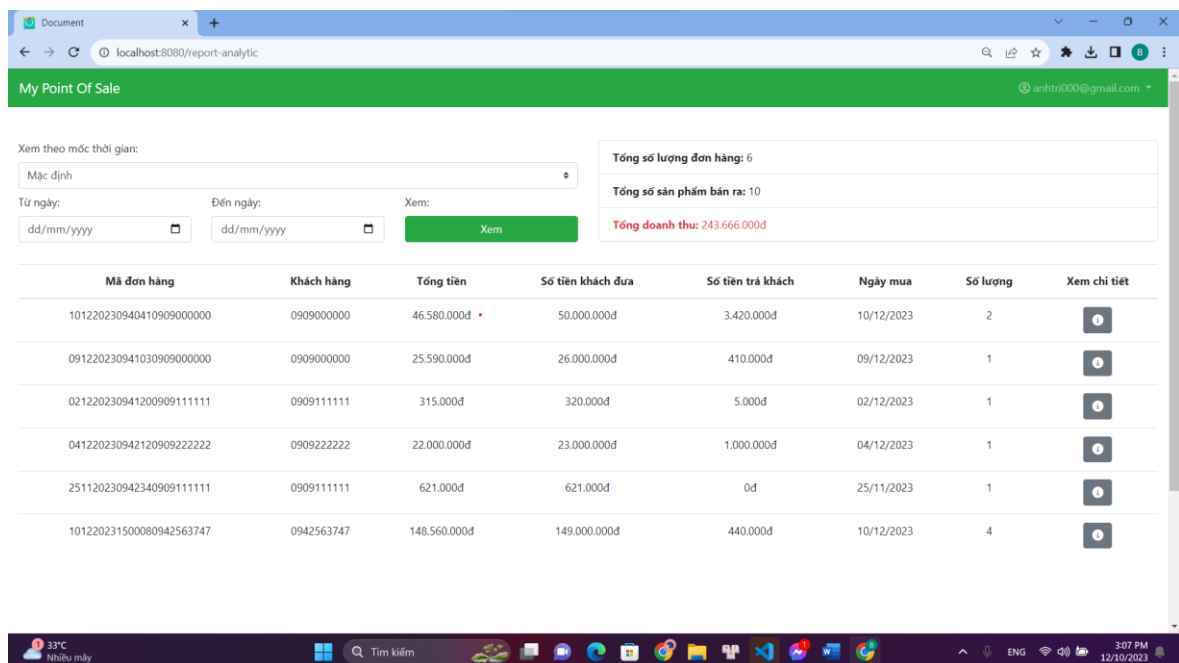
Hình 4. 8: Màn hình danh sách đơn hàng với giao diện nhân viên

4.1.9 Màn hình danh sách chi tiết đơn hàng với giao diện nhân viên



Hình 4. 9: Màn hình chi tiết đơn hàng với giao diện nhân viên

4.1.10 Màn hình báo cáo thống kê với giao diện nhân viên



Hình 4. 10: Màn hình báo cáo thống kê với giao diện nhân viên

4.1.11 Hồ sơ cá nhân của nhân viên với giao diện nhân viên

Thông tin cá nhân



Email:

Họ và tên:

[Đổi mật khẩu](#)

Hình 4. 11: Xem hồ sơ cá nhân của nhân viên với giao diện nhân viên

4.1.12 Màn hình quản lý sản phẩm 1 với giao diện quản lý

My Point Of Sale | Sản Phẩm | Tài Khoản | Báo Cáo Thống Kê | Quản trị viên

Barcode: VD: 91238129 | Tên sản phẩm: VD: Iphone 15 | Danh mục: Điện thoại

Giá nhập: | Giá bán: | Ngày tạo: dd/mm/yyyy

Chọn ảnh | Browse

Thêm | Hủy

HÌNH ẢNH	BARCODE	TÊN SẢN PHẨM	DANH MỤC	GIÁ NHẬP	GIÁ BÁN	NGÀY TẠO	HÀNH ĐỘNG
	00000000	Iphone 15 Pro Max 256Gb	Điện thoại	35.000.000đ	40.990.000đ	03/08/2022	
	11111111	Ipad Air 7	Máy tính bảng	20.380.000đ	25.590.000đ	15/12/2022	
	22222222	Iphone 14 Pro 128Gb	Điện thoại	28.500.000đ	33.570.000đ	15/12/2022	

Hình 4. 12: Màn hình quản lý sản phẩm 1 với giao diện quản lý

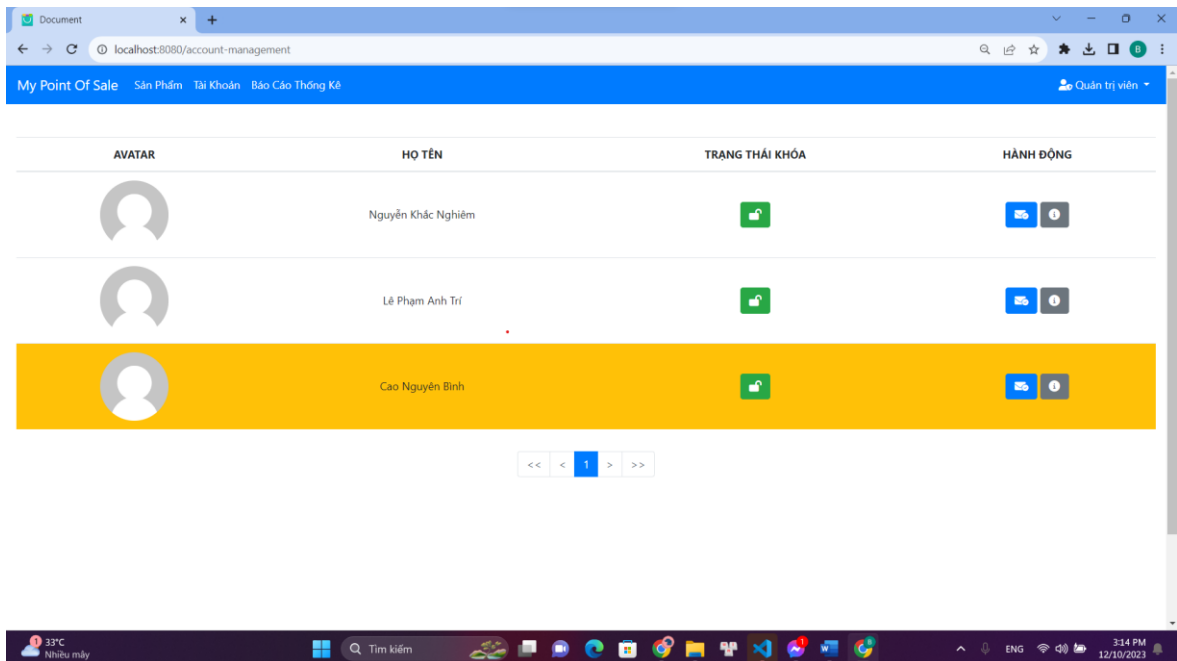
4.1.13 Màn hình quản lý sản phẩm 2 với giao diện quản lý

	44444444	Galaxy Z Flip4 5G 256Gb	Điện thoại	20.000.000đ	22.000.000đ	15/12/2022	
	55555555	Xiaomi 13T 5G 8Gb	Điện thoại	8.000.000đ	9.560.000đ	16/12/2022	
	66666666	Oppo A58 8Gb	Điện thoại	5.000.000đ	5.490.000đ	16/12/2022	
	77777777	Oppo Find N3 5G	Điện thoại	40.000.000đ	44.990.000đ	16/12/2022	
	88888888	Samsung Galaxy S23 5G 128Gb	Điện thoại	13.000.000đ	16.890.000đ	16/12/2022	
	99999999	Realme C53	Điện thoại	2.000.000đ	3.800.000đ	16/12/2022	

<< < 1 2 3 4 > >>

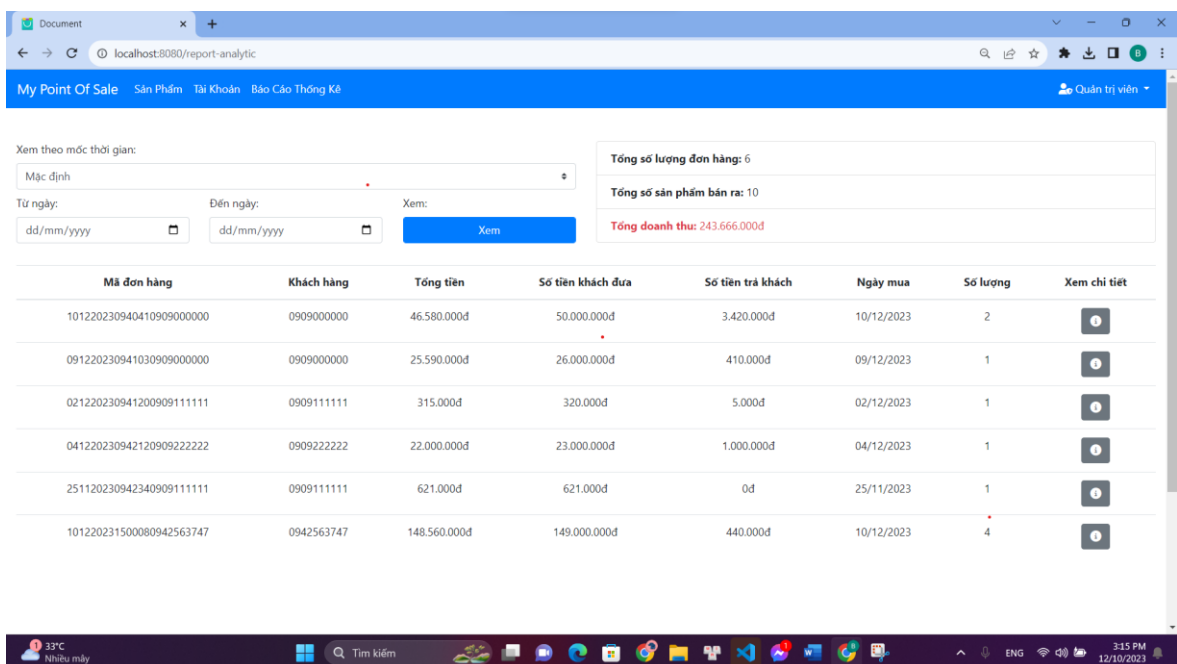
Hình 4. 13: Màn hình quản lý sản phẩm 2 với giao diện quản lý

4.1.14 Màn hình quản lý tài khoản với giao diện quản lý



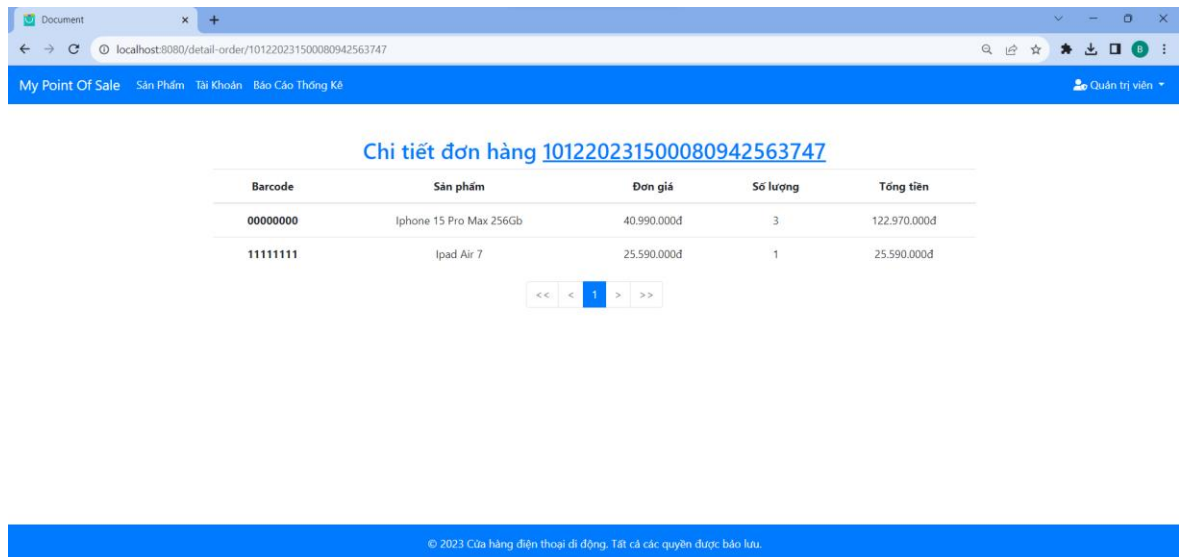
Hình 4. 14: Màn hình quản lý tài khoản với giao diện quản lý

4.1.15 Màn hình báo cáo thống kê với giao diện quản lý



Hình 4. 15: Màn hình báo cáo thống kê với giao diện quản lý

4.1.16 Màn hình xem chi tiết đơn hàng với giao diện quản lý



Chi tiết đơn hàng [101220231500080942563747](#)

Barcode	Sản phẩm	Đơn giá	Số lượng	Tổng tiền
00000000	Iphone 15 Pro Max 256Gb	40.990.000đ	3	122.970.000đ
11111111	Ipad Air 7	25.590.000đ	1	25.590.000đ

<< < 1 > >>

© 2023 Cửa hàng điện thoại di động. Tất cả các quyền được bảo lưu.

Hình 4. 16: Chi tiết giao diện chi tiết đơn hàng với giao diện quản lý

4.2 Kết luận

4.2.1 Ưu điểm của hệ thống

Quản lý tài khoản hiệu quả:

- Hệ thống quản lý tài khoản có tính bảo mật cao, đặc biệt là với khả năng yêu cầu salesperson sử dụng liên kết trong email để đăng nhập lần đầu tiên.
- Quản trị viên có khả năng thực hiện các thao tác quản lý nhân viên một cách linh hoạt, bao gồm việc gửi lại email đăng nhập trong 1 phút và khóa/mở khóa tài khoản.

Quản lý danh mục sản phẩm:

- Tính năng quản lý danh mục sản phẩm cho phép quản trị viên thực hiện các thao tác cơ bản như xem, thêm, cập nhật và xóa sản phẩm.
- Yêu cầu xác nhận trước khi xóa sản phẩm, đảm bảo tính nhất quán trong quá trình quản lý.

Quản lý khách hàng và thanh toán:

- Tính năng quản lý khách hàng thông minh, hiển thị thông tin khách hàng nhanh chóng khi thanh toán, giúp tăng trải nghiệm mua sắm.
- Tự động tạo tài khoản khách hàng khi là lần đầu tiên mua hàng.

Xử lý giao dịch:

- Giao diện thanh toán được thiết kế mạch lạc và thuận tiện, giúp nhân viên bán hàng dễ dàng thêm và xóa sản phẩm trong quá trình giao dịch thông qua việc tìm kiếm hoặc nhập tên barcode/tên sản phẩm.
- Tự động cập nhật thông tin và tổng giá trị đơn hàng khi thêm/xóa sản phẩm.

Báo cáo và phân tích:

- Cung cấp báo cáo và phân tích theo các khoảng thời gian khác nhau, giúp quản trị viên và nhân viên bán hàng nắm bắt tình hình kinh doanh.

4.2.2 Khuyết điểm của hệ thống

Chưa triển khai kiểm thử chéo:

- Chưa triển khai hệ thống trên môi trường thực tế và kiểm thử chéo trên nhiều loại trình duyệt và thiết bị.

Chưa triển khai deploy web:

- Chưa triển khai deploy web, điều này làm giảm khả năng trải nghiệm thực tế của hệ thống.

4.2.3 Những điều đã làm được

Quản lý tài khoản hiệu quả:

- Xây dựng tài khoản quản trị tích hợp sẵn.
- Triển khai cơ chế đặt lại mật khẩu và khóa tài khoản cho quản trị viên.

Quản lý danh mục sản phẩm:

- Xây dựng các chức năng quản lý sản phẩm.

Quản lý khách hàng và thanh toán:

- Hiện thị thông tin khách hàng khi thanh toán.
- Tự động tạo tài khoản khách hàng khi cần thiết.

Xử lý giao dịch:

- Xây dựng giao diện thanh toán thuận tiện và tự động cập nhật thông tin đơn hàng.

Báo cáo và phân tích:

- Cung cấp báo cáo và phân tích theo khoảng thời gian.

4.2.4 Những điều chưa làm được

Triển khai deploy web:

- Chưa triển khai web trên môi trường thực tế.

Một số chức năng phụ không có trong yêu cầu: lọc sản phẩm trong giao diện nhân viên và tìm kiếm trong việc quản lý:

- Chưa triển khai chức năng lọc danh sách sản phẩm và tìm kiếm trong các giao diện quản lý.

4.2.5 Hướng phát triển trong tương lai

Triển khai kiểm thử chéo:

- Triển khai hệ thống trên môi trường thực tế và kiểm thử chéo trên nhiều loại trình duyệt và thiết bị để đảm bảo tính ổn định và tương thích.

Chức năng lọc sản phẩm trong giao diện trang chủ nhân viên và tìm kiếm trong các trang quản lý:

- Triển khai chức năng lọc danh sách sản phẩm cho nhân viên và tìm kiếm trong các giao diện quản lý để tăng hiệu suất làm việc.

Tối ưu hóa trải nghiệm người dùng:

- Tối ưu hóa giao diện người dùng để đảm bảo trải nghiệm tốt nhất cho quản trị viên và nhân viên bán hàng.

Triển khai deploy web:

- Triển khai deploy web để cho phép người dùng sử dụng hệ thống trong môi trường thực tế.

Chức năng mở rộng:

- Xem xét và triển khai các chức năng mở rộng khác theo yêu cầu của doanh nghiệp hoặc người dùng, như chức năng lấy lại mật khẩu tự động,...

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Talan, Dev.to, Middleware trong expressjs là gì? Tác dụng và cách sử dụng như thế nào?, <https://dev.to/tuantnguyen/middleware-trong-expressjs-l-g-tc-dng-v-cch-s-dng-nh-th-no-3i2j>.
2. Nguyễn Xuân Hùng, Viblo, Sử dụng handlebars js, <https://viblo.asia/p/su-dung-handlebars-js-naQZRJP0Zvx>.
3. Trần Vương Minh, Viblo, Tìm hiểu về json web token (JWT), <https://viblo.asia/p/tim-hieu-ve-json-web-token-jwt-7rVRqp73v4bP>.

Tiếng Anh

4. shubhamkquv4, Geeksforgeeks, MongoDB Tutorial, <https://www.geeksforgeeks.org/mongodb-tutorial/>.
5. StrongLoop/IBM, Routing, <https://expressjs.com/en/guide/routing.html>.

PHỤ LỤC