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

MÃ TRƯỜNG QUANG – 52100925
NGUYỄN NGỌC HƯƠNG GIANG – 52100019
HUỲNH THỊ MỘNG TRINH – 52100132

HỆ THỐNG BÁN HÀNG CHO CỬA HÀNG ĐIỆN THOẠI

BÁO CÁO CUỐI KỲ
PHÁT TRIỂN ỨNG DỤNG WEB VỚI NODEJS

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

MÃ TRƯỜNG QUANG – 52100925
NGUYỄN NGỌC HƯƠNG GIANG – 52100019
HUỲNH THỊ MỘNG TRINH – 52100132

HỆ THỐNG BÁN HÀNG CHO CỬA HÀNG ĐIỆN THOẠI

BÁO CÁO CUỐI KỲ
PHÁT TRIỂN ỨNG DỤNG WEB VỚI NODEJS

Người hướng dẫn
ThS.NCS. Vũ Đình Hồng

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn ThS.NCS. Vũ Đình Hồng – Giảng viên khoa Công nghệ thông tin – Trường đại học Tôn Đức Thắng, đã hỗ trợ và giúp đỡ nhiệt tình trong quá trình thực hiện Dự án này.

Chúng em trân trọng cảm ơn Thầy Cô giảng viên Trường đại học Tôn Đức Thắng nói chung cũng như Thầy Cô giảng viên khoa Công nghệ thông tin nói riêng đã giảng dạy và truyền đạt nhiều kinh nghiệm quý trong suốt quá trình học tập tại trường.

Cuối cùng, xin cám ơn gia đình, bạn bè đã luôn động viên và đồng hành trong quá trình học tập cũng như quá trình thực hiện Dự án này.

Mặc dù rất cẩn thận trong quá trình thực hiện đồ án cũng như viết báo cáo nhưng cũng không tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự góp ý từ các Thầy/Cô để đồ án được hoàn thiện hơn.

Xin chân thành cám ơn!

TP. Hồ Chí Minh, ngày 08 tháng 12 năm 2023

Tác giả

(Ký tên và ghi rõ họ tên)

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

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của Thầy ThS.NCS. 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 Dự án 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 tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án 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 08 tháng 12 năm 2023

Tác giả

(Ký tên và ghi rõ họ tên)

TÓM TẮT

Phát triển website cung cấp chức năng Điểm bán hàng nhằm mục đích bán hàng tại cửa hàng bán lẻ điện thoại và phụ kiện. Người dùng ứng dụng web này là nhân viên bán hàng và quản trị viên tại một cửa hàng điện thoại. Người mua điện thoại không phải là người dùng ứng dụng web này. Về cơ bản, ứng dụng web này cần cung cấp các chức năng như: giao dịch bán hàng, quản lý sản phẩm, quản lý nhân viên, xem báo cáo và thống kê. Loại ứng dụng này thường thấy ở các cửa hàng bán lẻ tại Việt Nam như cửa hàng điện tử, quần áo, thực phẩm, đồ uống hay siêu thị.

ABSTRACT

....

....

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC CÁC CHỮ VIẾT TẮT

CHƯƠNG 1. MỞ ĐẦU VÀ TỔNG QUAN ĐỀ TÀI

1.1 Lý do chọn đề tài

Trong bối cảnh ngành bán lẻ điện thoại và phụ kiện ngày càng phát triển tại Việt Nam, việc xây dựng một hệ thống Point of Sale (POS) trực tuyến sẽ mang lại nhiều lợi ích cho cửa hàng, từ quản lý bán hàng đến theo dõi hiệu suất kinh doanh.
Trong bối cảnh ngành bán lẻ điện thoại và phụ kiện ngày càng phát triển tại Việt Nam, việc xây dựng một hệ thống Point of Sale (POS) trực tuyến sẽ mang lại nhiều lợi ích cho cửa hàng, từ quản lý bán hàng đến theo dõi hiệu suất kinh doanh.

1.2 Yêu cầu của đề tài

Xây dựng một ứng dụng web có đầy đủ các chức chức năng như:quản lý tài khoảng, giao dịch bán hàng, quản lý sản phẩm, quản lý nhân viên, xem báo cáo và thống kê.

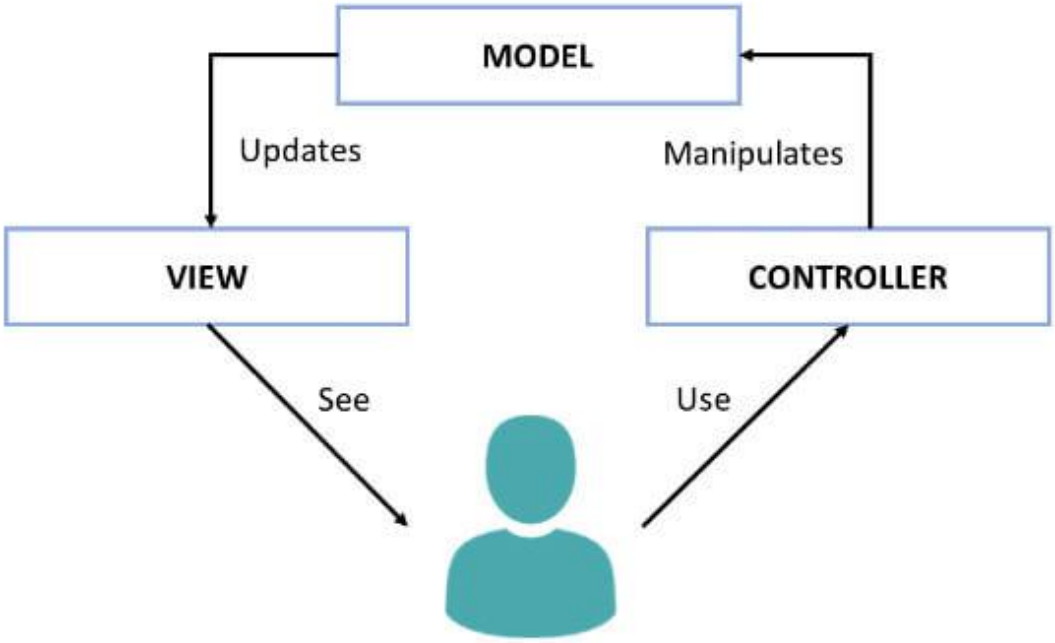
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Mô hình MVC

2.1.1 Khái niệm:

MVC là viết tắt của cụm từ “Model-View-Controller“. Đây là mô hình thiết kế được sử dụng trong kỹ thuật phần mềm. MVC là một mẫu kiến trúc phần mềm để tạo lập giao diện người dùng trên máy tính. MVC chia thành ba phần được kết nối với nhau và mỗi thành phần đều có một nhiệm vụ riêng của nó và độc lập với các thành phần khác. Tên gọi 3 thành phần:

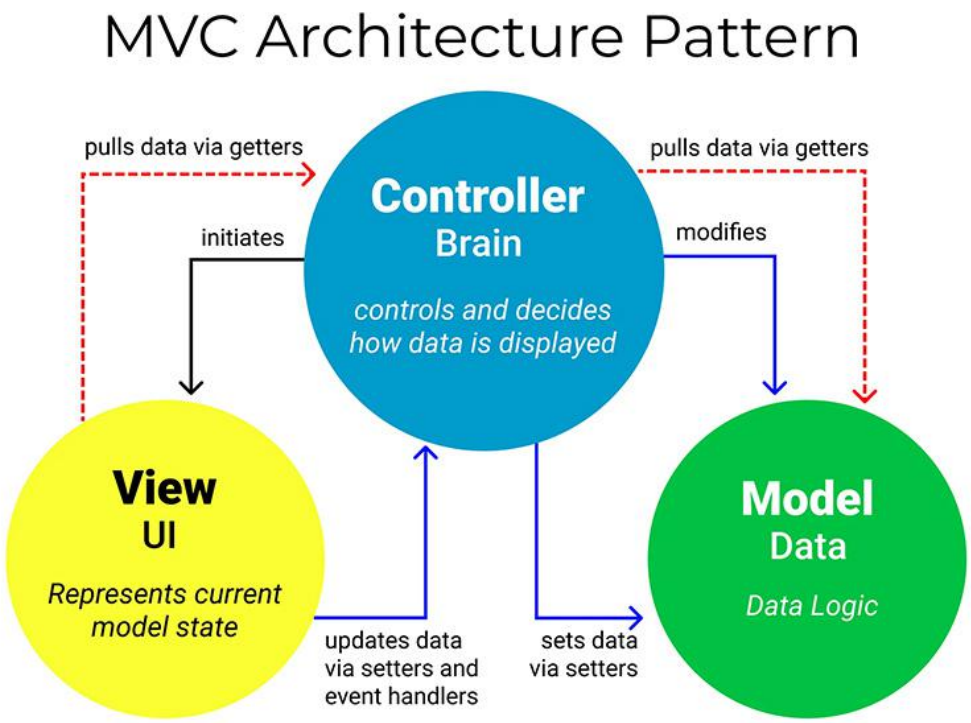
- Model (dữ liệu): Quản lí xử lí các dữ liệu.
- View (giao diện): Nới hiển thị dữ liệu cho người dùng.
- Controller (bộ điều khiển): Điều khiển sự tương tác của hai thành phần Model và View.



Hình 1Mô hình MVC

Mô hình MVC (MVC pattern) thường được dùng để phát triển giao diện người dùng. Nó cung cấp các thành phần cơ bản để thiết kế một chương trình cho máy tính hoặc điện thoại di động, cũng như là các ứng dụng web.

2.1.2 Tìm hiểu các thành phần trong mô hình MVC



Hình 2 Thành phần trong MVC

Mô hình MVC gồm 3 loại chính là thành phần bên trong không thể thiếu khi áp dụng mô hình này:

- Model: Model dùng để cung cấp dữ liệu, thực hiện kết nối, trích lọc, chèn, chỉnh sửa dữ liệu trong database, tương tác với file system, network.
- View:

  + Mỗi view là một trang web hiển thị dữ liệu gì đó. Dữ liệu mà view hiển thị do controller cung cấp (controller lấy từ model để đưa cho view).

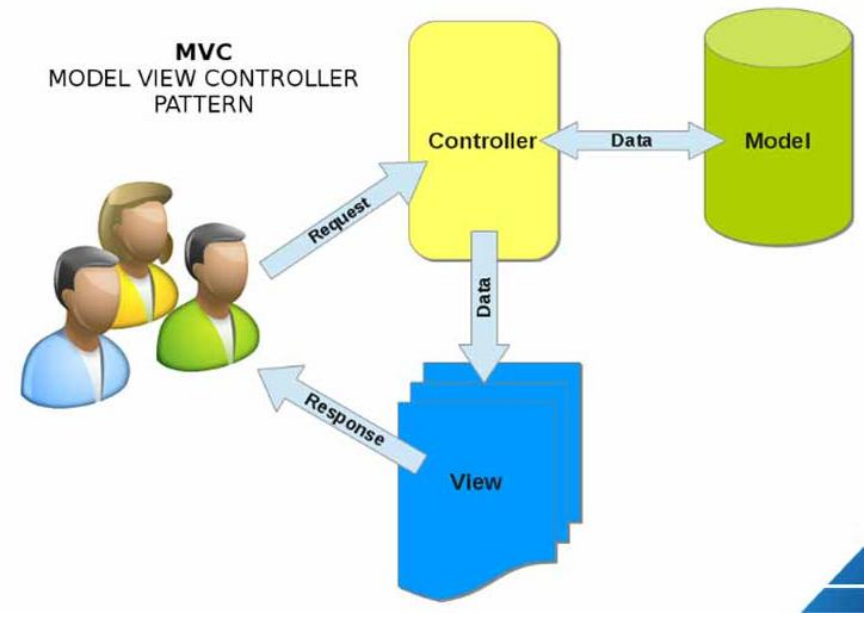
  + Việc tách riêntg vai trò của Controler, Model và View giúp cho team phân định rõ ràng các công việc xử lý nghiệp vụ, xử lý dữ liệu và trình bày dữ liệu. Do vậy việc cập nhật chỉnh sửa một thành phần không làm ảnh hưởng đến các thành phần khác.
- Controller: Controller được tạo ra để thực thi các yêu cầu từ user, controller thực hiện tiếp nhận tham số, gọi các hàm trong model, nạp các view cần thiết… Trong NodeJS , routes đóng vai trò như

controller.Ví dụ: một Controller có thể cập nhật một Model bằng cách thay đổi các thuộc tính của nhân vật trong game. Và nó có thể sửa đổi view bằng cách hiển thị nhân vật được cập nhật trong game đó.

2.1.3 Tìm hiểu các thành phần trong mô hình MVC

Luồng xử lý trong của mô hình MVC, bạn có thể hình dung cụ thể và chi tiết qua từng bước dưới đây:

- Khi một yêu cầu của từ máy khách (Client) gửi đến Server. Thì bị Controller trong MVC chặn lại để xem đó là URL request hay sự kiện.
- Sau đó, Controller xử lý input của user rồi giao tiếp với Model trong MVC.
- Model chuẩn bị data và gửi lại cho Controller.
- Cuối cùng, khi xử lý xong yêu cầu thì Controller gửi dữ liệu trở lại View và hiển thị cho người dùng trên trình duyệt.



Hình 3 Thành phần trong MVC

Ở đây, **View không giao tiếp trực tiếp với Model**. Sự tương tác giữa **View và Model** sẽ chỉ được xử lý bởi **Controller.**

2.1.4 Ưu và nhược điểm của MVC

- Ưu điểm:

- Đầu tiên, nhắc tới ưu điểm mô hình MVC thì đó là băng thông nhẹ vì không sử dụng viewstate nên khá tiết kiệm băng thông. Việc giảm băng thông giúp website hoạt động ổn định hơn.
- Kiểm tra đơn giản và dễ dàng, kiểm tra lỗi phần mềm trước khi bàn giao lại cho người dùng.
- Một lợi thế chính của MVC là nó tách biệt các phần Model, Controller và View với nhau.
- Sử dụng mô hình MVC chức năng Controller có vai trò quan trọng và tối ưu trên các nền tảng ngôn ngữ khác nhau
- Ta có thể dễ dàng duy trì ứng dụng vì chúng được tách biệt với nhau.
- Có thể chia nhiều developer làm việc cùng một lúc. Công việc của các developer sẽ không ảnh hưởng đến nhau.
- Hỗ trợ TTD (test-driven development). Chúng ta có thể tạo một ứng dụng với unit test và viết các won test case.
- Phiên bản mới nhất của MVC hỗ trợ trợ thiết kế responsive website mặc định và các mẫu cho mobile. Chúng ta có thể tạo công cụ View của riêng mình với cú pháp đơn giản hơn nhiều so với công cụ truyền thống.

- Nhược điểm:

Bên cạnh những ưu điểm MVC mang lại thì nó cũng có một số nhược điểm cần khắc phục.

- MVC đa phần phù hợp với công ty chuyên về website hoặc các dự án lớn thì mô hình này phù hợp hơn so với với các dự án nhỏ, lẻ vì khá là cồng kềnh và mất thời gian.
- Khó triển khai

2.1.5 Mô hình MVC trong Express:

Lập trình MVC trong NodeJS tức bạn sẽ tạo nên các chức năng cho website quy ước của MVC. Theo đó, các việc xử lý request, hiển thị dữ liệu , xử lý dữ liệu phải tách bạch ra theo quy định.

Module express-generator giúp bạn tạo project đã gần giống với tổ chức MVC, như folder views chứa các file view để hiện dữ liệu, folder routes dùng để xử lý các đường path chính là thành phần controller trong MVC. Bạn cần tạo thêm thành phần model nữa để xử lý dữ liệu.

2.2 MongoDB:

2.2.1 MongoDB là gì ?

MongoDB là một cơ sở dữ liệu tài liệu với khả năng mở rộng và linh hoạt mà bạn muốn với truy vấn và lập chỉ mục mà bạn cần. Nó được sử dụng rất rộng rãi bởi khả năng phát triển ứng dụng dễ dàng hơn.

Mô hình lưu trữ dữ liệu của MongoDB rất đơn giản để các nhà phát triển tìm hiểu và sử dụng, trong khi vẫn cung cấp tất cả các khả năng cần thiết để đáp ứng các yêu cầu phức tạp nhất ở mọi quy mô. MongoDB cung cấp các module cho hơn 10 ngôn ngữ trong đó có Node.js, và cộng đồng cũng đã xây dựng thêm hàng tá ngôn ngữ.

2.2.2 Một số điểm nổi bật của MongoDB

- MongoDB lưu trữ dữ liệu trong các tài liệu linh hoạt, giống như JSON, có nghĩa là các trường có thể thay đổi từ tài liệu này sang tài liệu khác và cấu trúc dữ liệu có thể được thay đổi theo thời gian.

- Mô hình dữ liệu tương đồng với các object dữ liệu trong quá trình lập trình, giúp dữ liệu dễ dàng làm việc với các dữ liệu.

- Truy vấn đặc biệt, lập chỉ mục và tổng hợp thời gian thực cung cấp các cách mạnh mẽ để truy cập và phân tích dữ liệu của bạn

- MongoDB là một cơ sở dữ liệu phân tán ở nhân của nó, vì vậy tính sẵn sàng cao, khả năng mở rộng cao (theo chiều ngang), bạn có thể mổ rộng mô hình

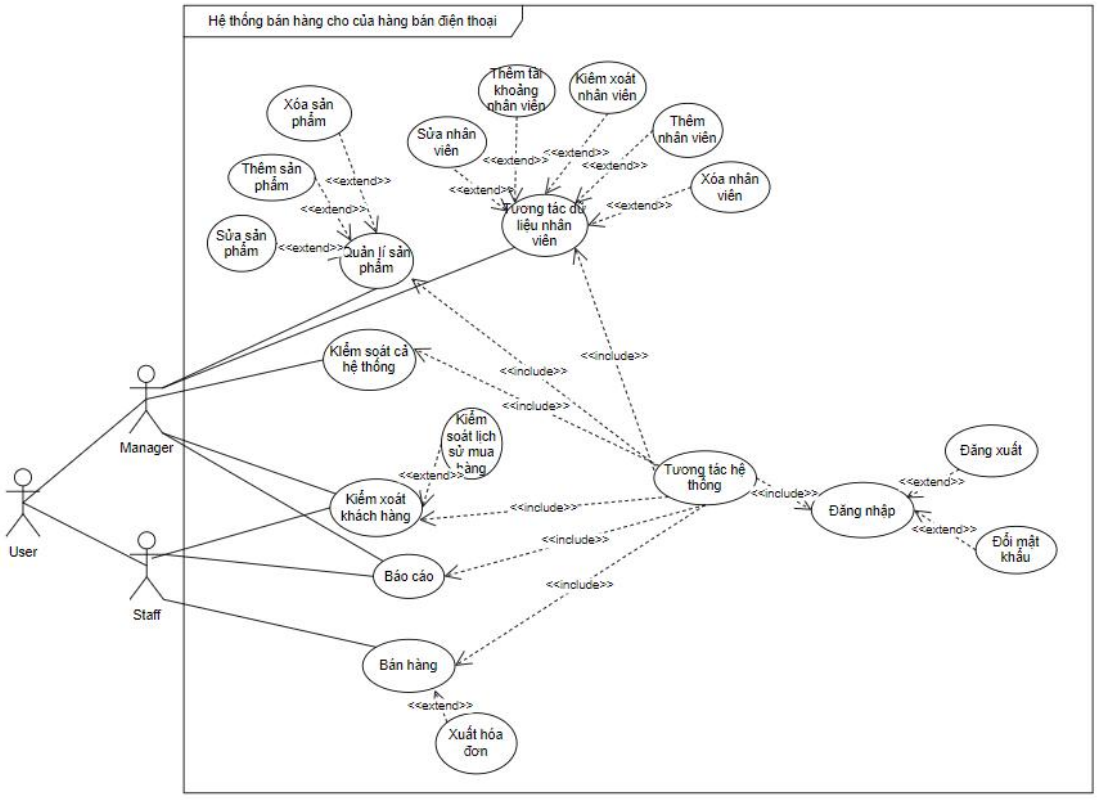
lưu trữ bằng cách thêm các máy chủ khác nhau ở nhiều địa điểm mà không phải nâng cấp phần cứng của server duy nhất như các hệ quản trị cơ sỡ dữ liệu MySQL.

- MongoDB là miễn phí để sử dụng.

⇨ MongoDB lưu trữ dữ liệu một cách linh hoạt theo dạng JSON, trong khi đó MySQL lưu trữ dữ liệu theo các bảng cấu trúc. Mô hình lưu trữ khác nhau bởi vậy với việc xử lý các dữ liệu lớn người ta thường dùng các hệ quản trị cơ sở dữ liệu NoSQL (như MongoDB, Redis).

CHƯƠNG 3. PHÂN TÍCH THIẾT KẾ

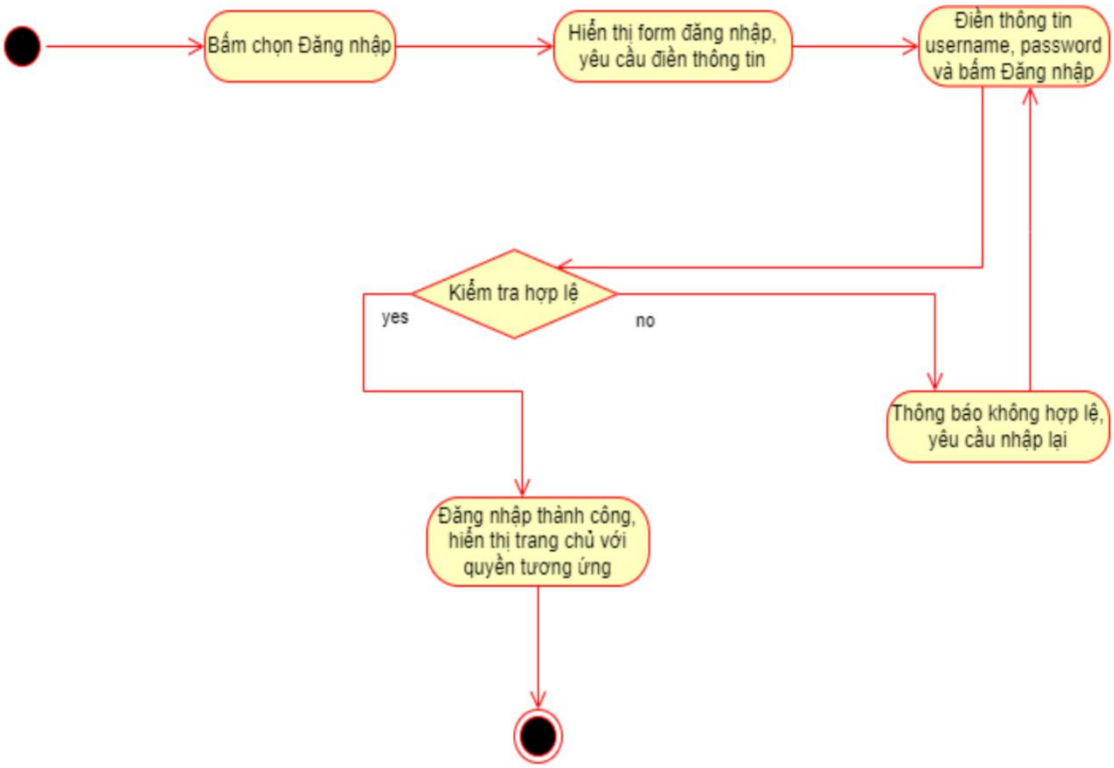
3.1 Usecase



Hình 4 Usecase tổng quan

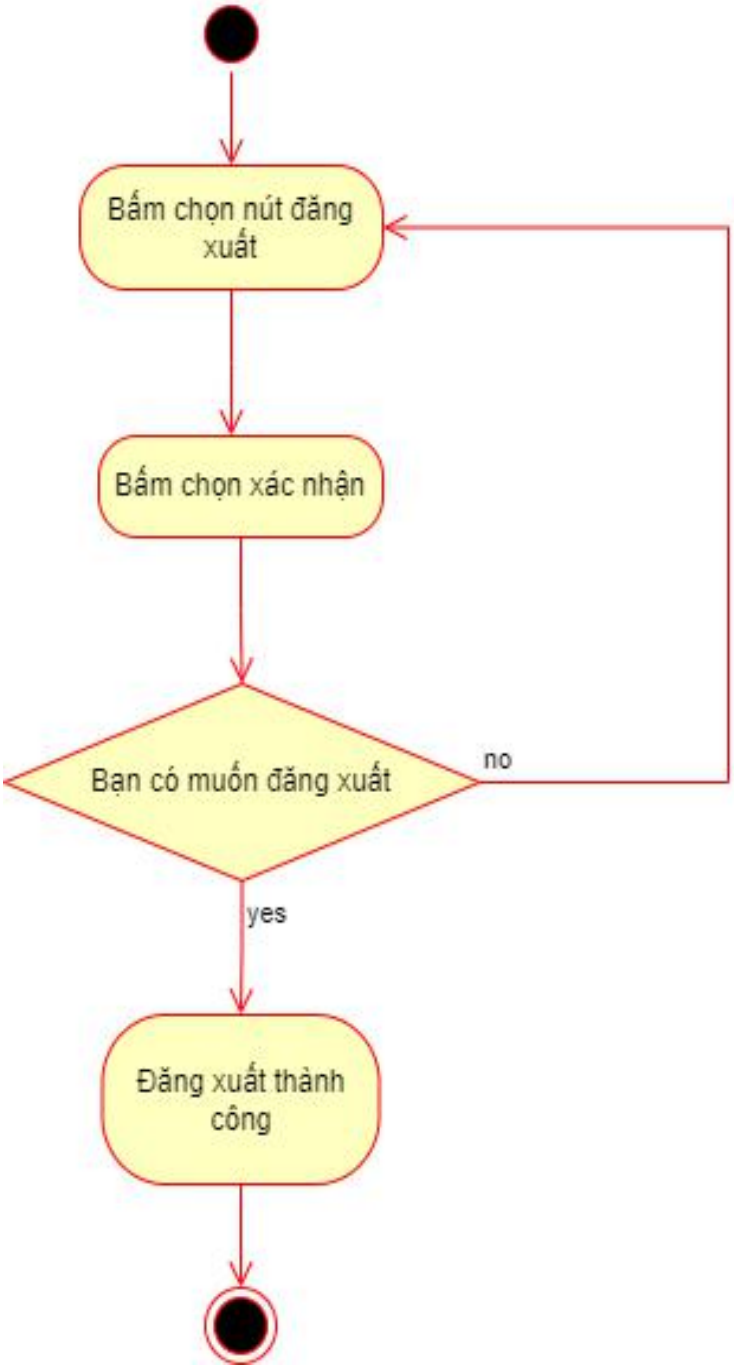
3.2 Sơ đồ trạng thái

3.2.1 Đăng nhập:



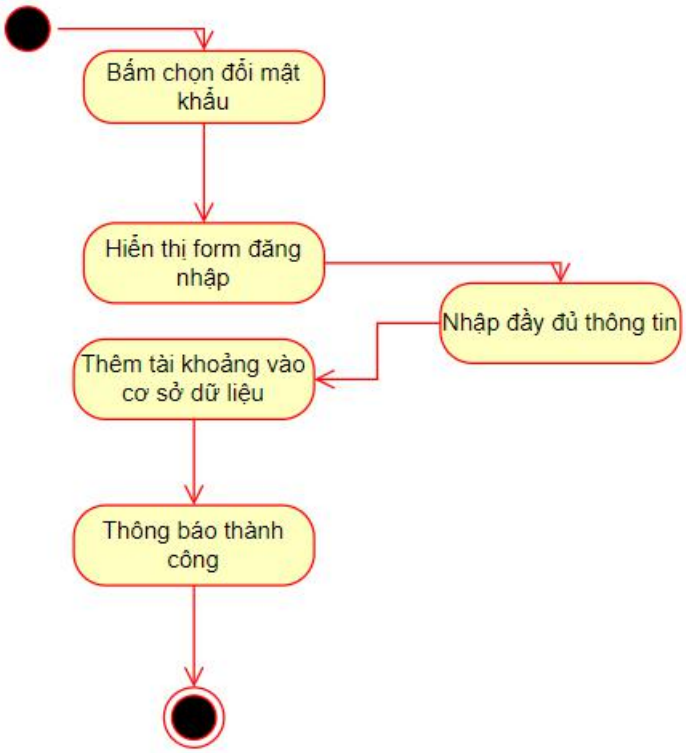
Hình 5 Sơ đồ trạng thái đăng nhập

3.2.2 Đăng xuất :



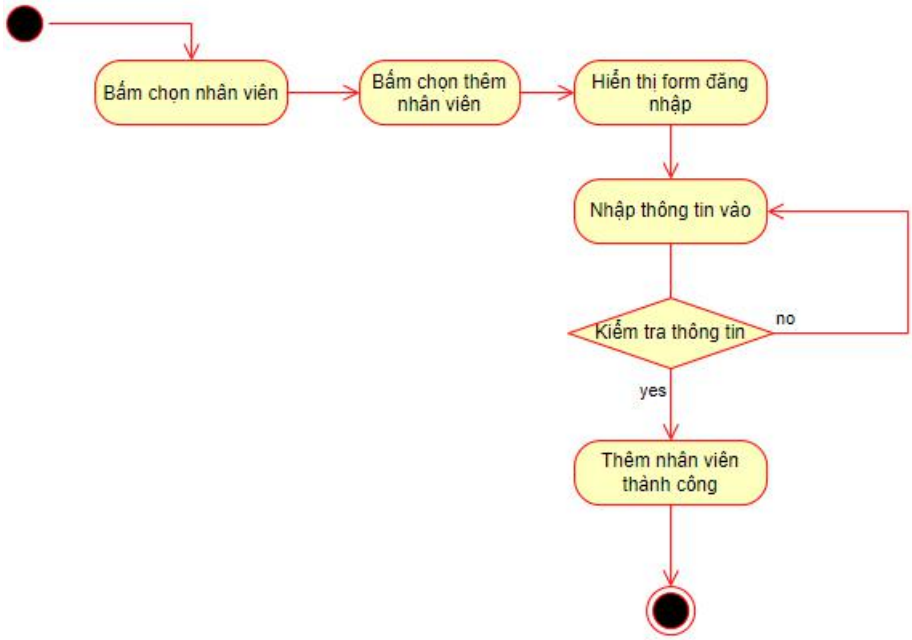
Hình 6 Sơ đồ đăng xuất

3.2.3 Đổi mật khẩu:



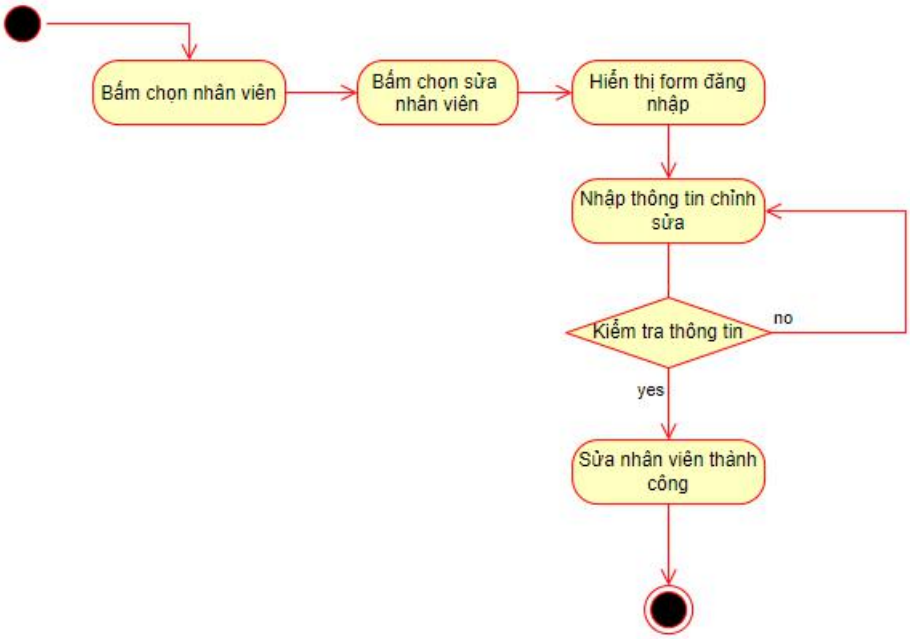
Hình 7 Sơ đồ trang thái đổi mật khẩu

3.2.4 Thêm nhân viên:



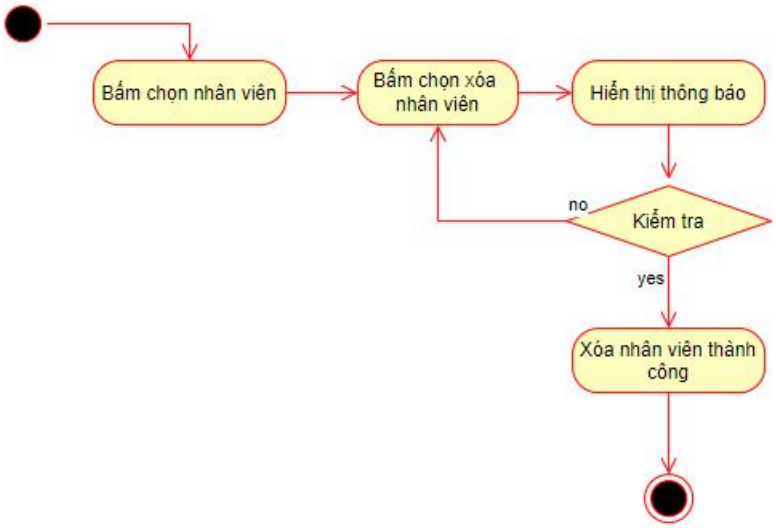
Hình 8 Sơ đồ trạng thái thêm nhân viên

3.2.5 Sửa nhân viên:



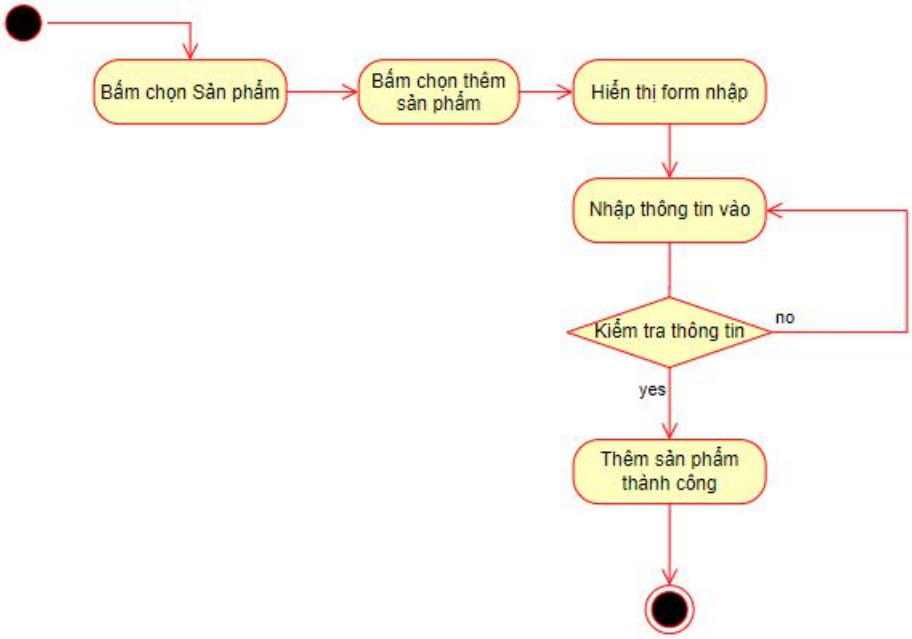
Hình 9 Sơ đồ trạng thái Sửa nhân viên

3.2.6 Xóa nhân viên:



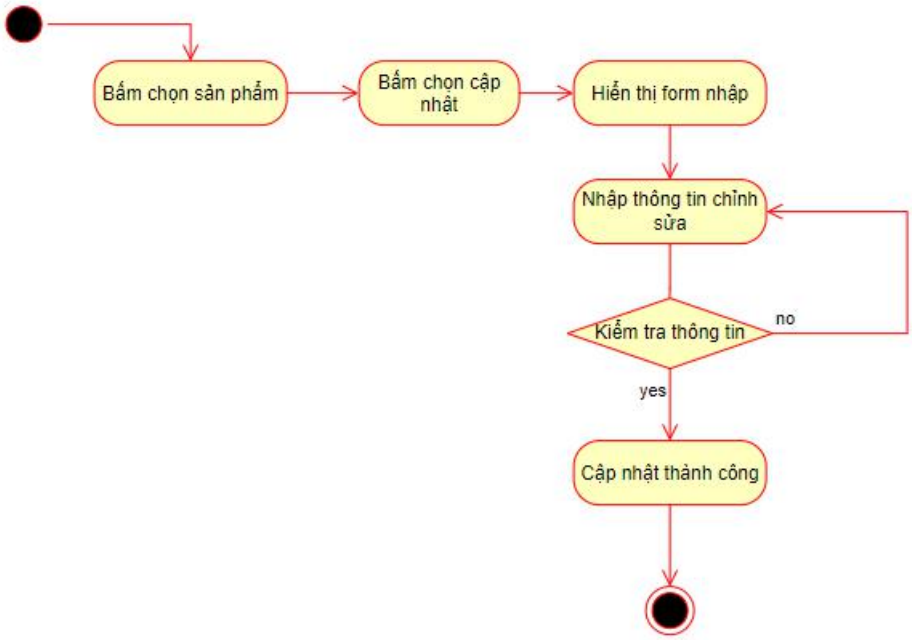
Hình 10 Sơ đồ trạng thái Xóa nhân viên

3.2.7 Thêm sản phẩm:



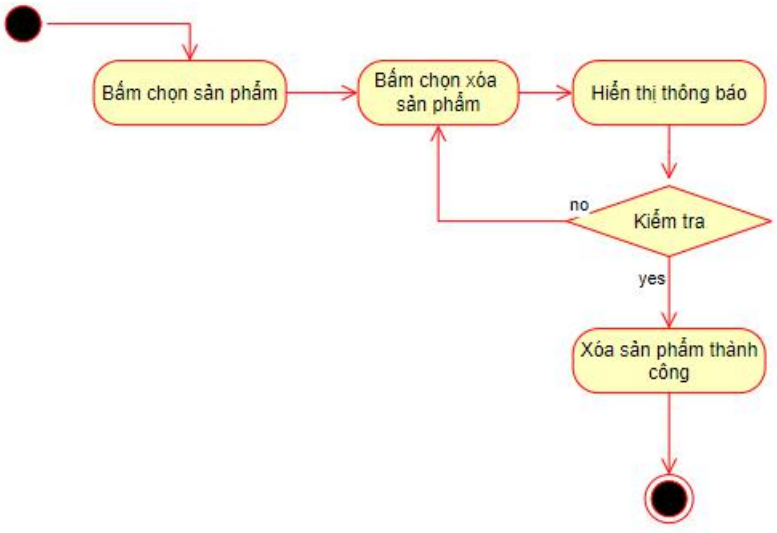
Hình 11 Sơ đồ trạng thái thêm sản phẩm

3.2.8 Cập nhật sản phẩm:



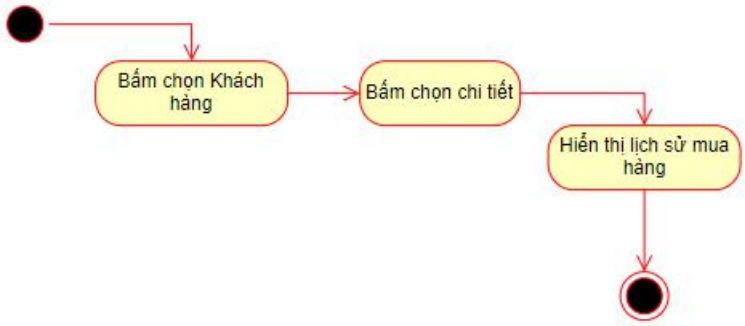
Hình 12 Sơ đồ trạng thái cập nhật sản phẩm

3.2.9 Xóa sản phẩm:



Hình 13 Sơ đồ trạng thái Xóa sản phẩm

3.2.10 Lịch sử mua hàng:

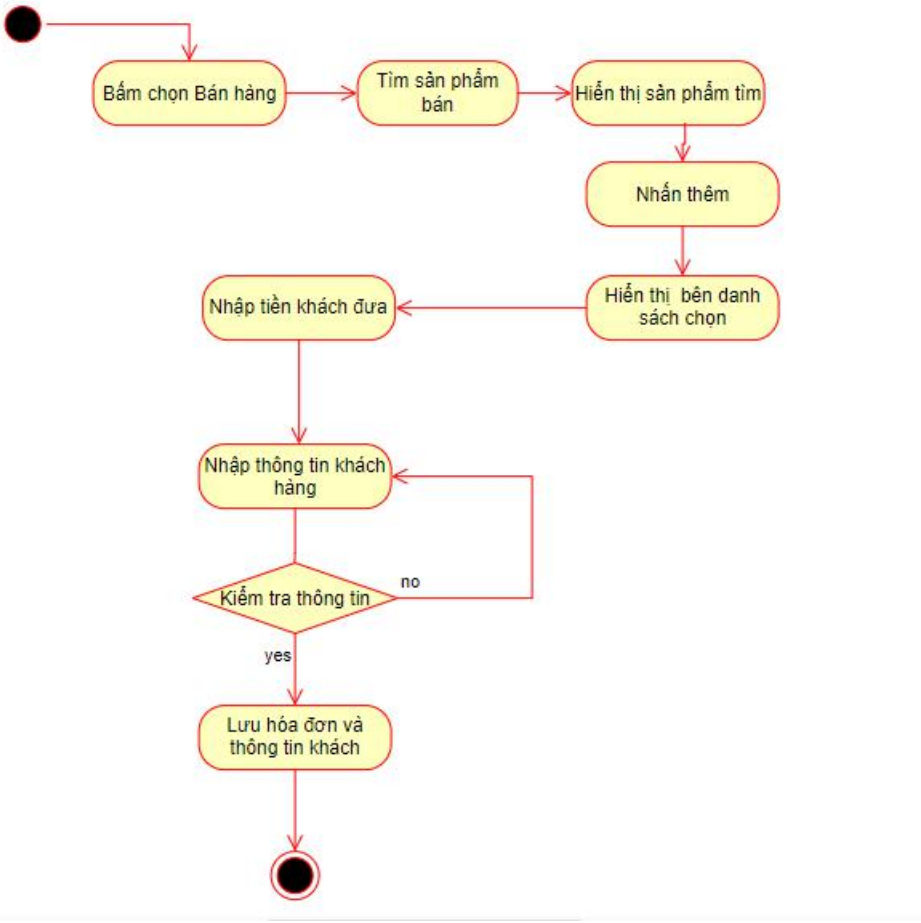


Hình 14 Sơ đồ trạng thái xem lịch sử mua hàng

3.2.11 Báo cáo doanh thu:

Hình 15 Sơ đồ trạng thái Báo cáo doanh thu

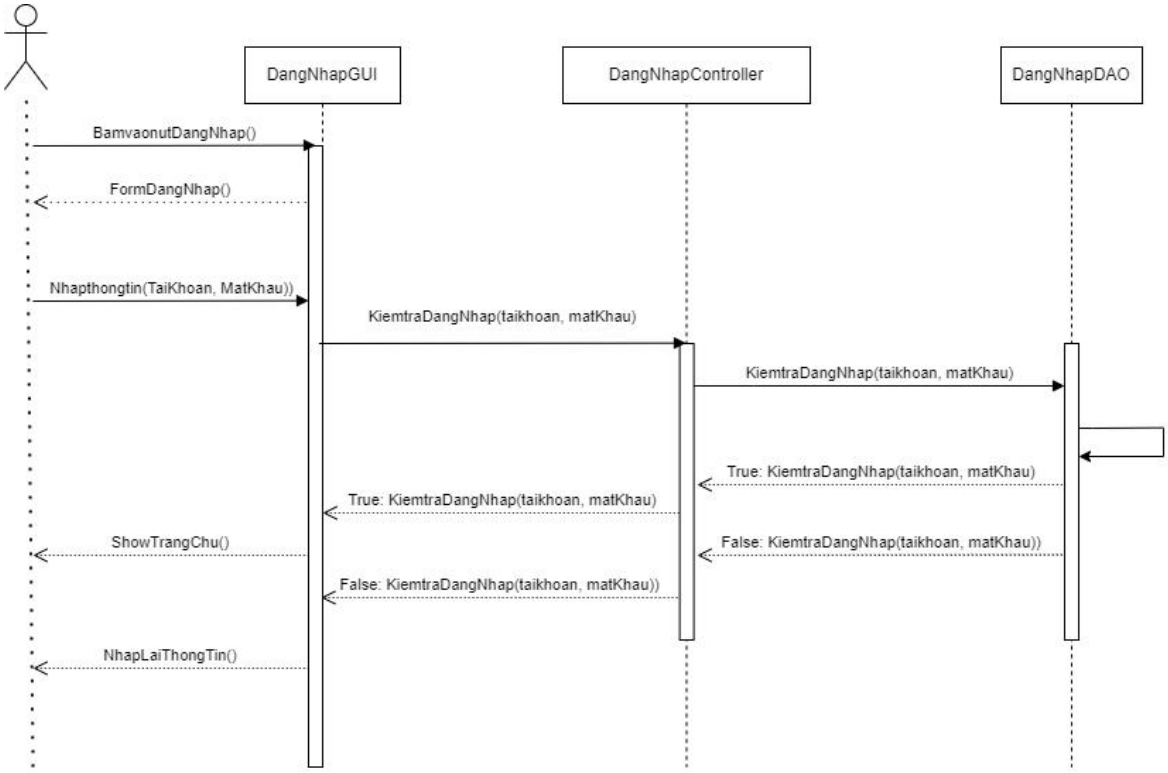
3.2.12 Bán hàng:



Hình 16 Sơ đồ trạng thái Bán hàng

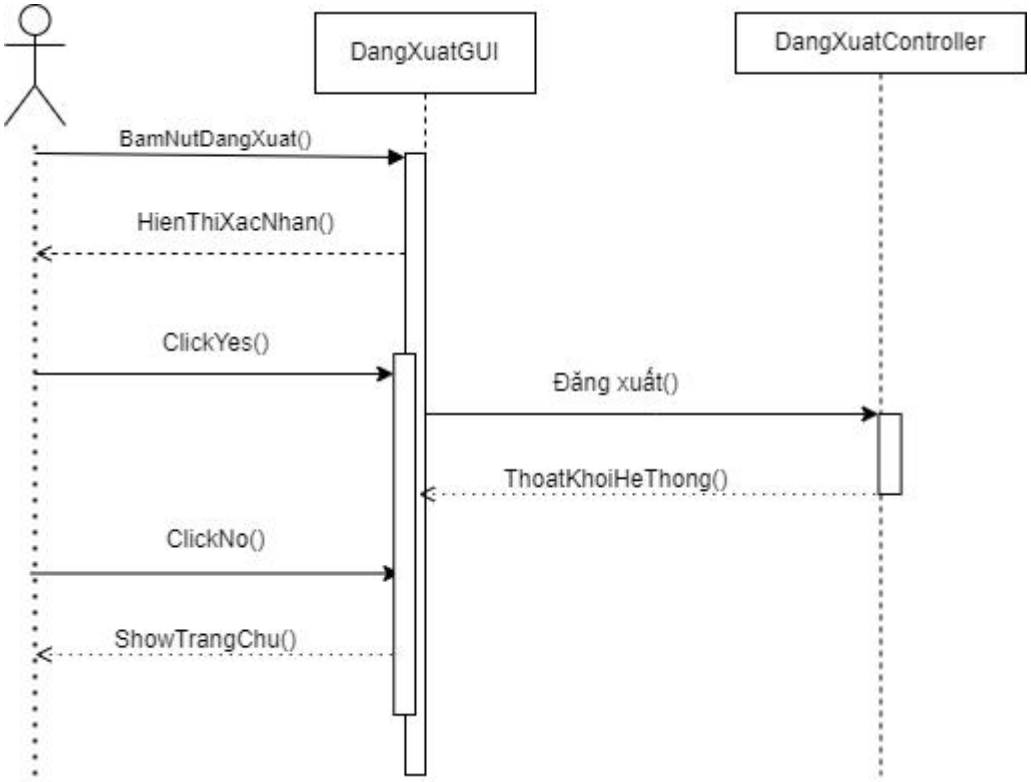
3.3 Sơ đồ hoạt động

3.3.1 Đăng nhập:



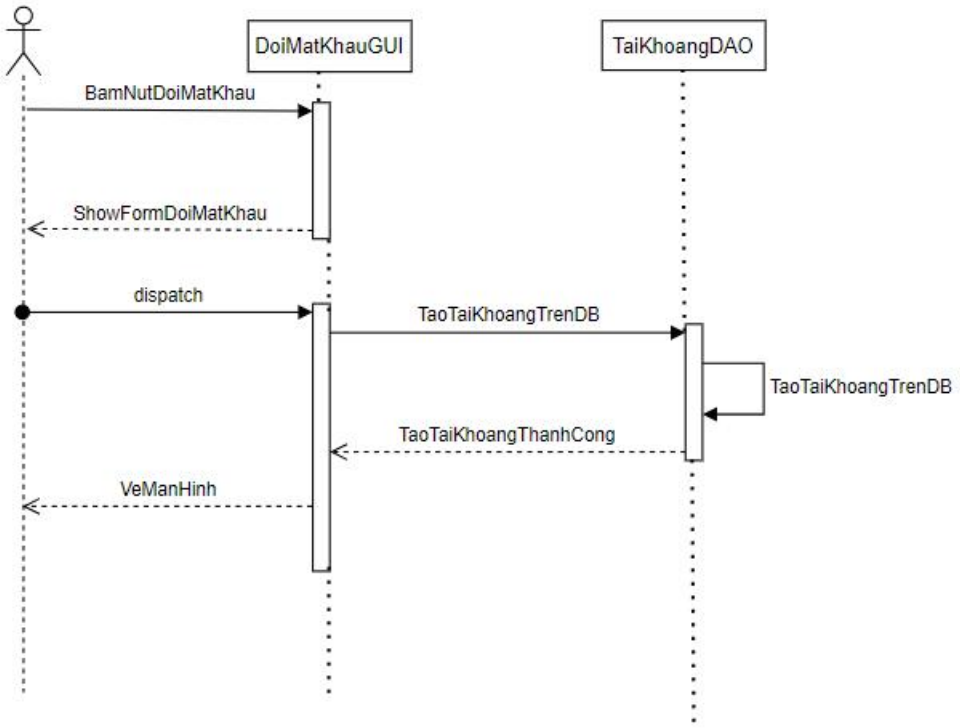
Hình 17 Đăng nhập

3.3.2 Đăng xuất:



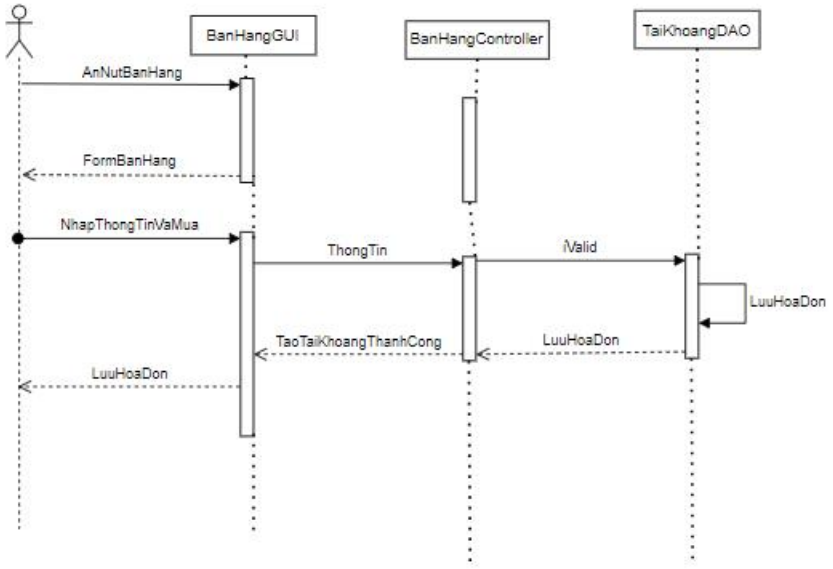
Hình 18 Đăng xuất

3.3.3 Đổi mật khẩu:



Hình 19 Đổi mật khẩu

3.3.4 Bán hàng:



Hình 20 Bán hàng

CHƯƠNG 4. HIỆN THỰC HỆ THỐNG

4.1 Backend

Được tổ chức dựa theo mô hình MVC (Model-View-Controller)

4.1.1 Config

Trong mô hình MVC, phần "Config" đề cập đến cấu hình hệ thống, nơi định nghĩa và thiết lập các tham số và tùy chọn cần thiết để ứng dụng hoạt động đúng cách. Điều này có thể bao gồm cấu hình cơ sở dữ liệu, cấu hình bảo mật, các biến môi trường, và các tùy chọn khác để tinh chỉnh hành vi của ứng dụng.

Trong bài sẽ có 2 file thuộc config là config.js và db.js

- config.js

```
const dotenv = require("dotenv") ;

dotenv.config();


const { SECRET_SESSION, PORT, MONGODB_URI, SECRET_KEY, HOST } = process.env

const config = {
    secret_session: SECRET_SESSION || "",
    secret_key: SECRET_KEY || "",
    port: PORT || 8080,
    mongodb_url: MONGODB_URI,
    host: HOST
};

module.exports = config;
```

Phần này sử dụng thư viện dotenv để load các biến môi trường từ file .env vào biến process.env. Điều này giúp bạn duy trì và quản lý các cấu hình của ứng dụng mà không cần trực tiếp đặt trực tiếp giá trị vào mã nguồn.

- dotenv.config(): sử dụng thư viện dotenv để load các biến môi trường từ file .env vào process.env.

- Destructuring Assignment: SECRET_SESSION, PORT, MONGODB_URI, SECRET_KEY, và HOST là các biến môi trường được

đọc từ process.env. Dòng này sử dụng destructuring assignment để gán giá trị của các biến này từ process.env.

- Object Literal:
    - Một object config được tạo, chứa các cấu hình của ứng dụng.
    - secret_session, secret_key, port, mongodb_url, và host là các thuộc tính của object config.
    - Các giá trị được gán từ các biến môi trường hoặc giá trị mặc định nếu biến môi trường không tồn tại.
    - PORT có giá trị mặc định là 8080 nếu không có giá trị từ biến môi trường.

- module.exports = config: doạn mã cuối cùng xuất object config để có thể sử dụng nó ở các file khác trong ứng dụng. Điều này giúp tránh việc trực tiếp sử dụng process.env và tạo ra một cách truy cập thuận tiện đến cấu hình của ứng dụng từ các module khác.

- db.js

```
const mongoose = require("mongoose") ;
const config = require('./config');
const User = require('../models/user');
const Customer = require('../models/customer');
const { mongodb_url } = config;


const dbConnected = async () => {
  try {
    mongoose
      .connect(mongodb_url)
      .then(() =>
        console.info(`MongoDB Connected to ${mongodb_url}`)
      )
      .catch(err => console.error(err));
  } catch (error) {
    console.info(`MongoDB failed to connect to ${mongodb_url}`,error);
    return null;
  }
};

User.autoCreateAdmin();
Customer.autoCreateCustomer();
module.exports = { dbConnected };
```

Thực hiện các bước liên quan đến kết nối MongoDB và khởi tạo dữ liệu mẫu cho các mô hình User và Customer.

- Import thư viện và Modules: Import các thư viện và modules cần thiết, bao gồm mongoose để kết nối với MongoDB, config để lấy thông tin cấu hình, và các mô hình User và Customer.

- Hàm kết nối MongoDB:

- dbConnected là một hàm async được thiết kế để kết nối đến MongoDB sử dụng URL từ biến môi trường mongodb_url.
- Sử dụng mongoose.connect để kết nối và log thông báo kết nối thành công hoặc lỗi nếu có.
- Sử dụng try-catch để bắt lỗi trong quá trình kết nối và log thông báo nếu kết nối thất bại.

- Gọi hàm kết nối và tạo dữ liệu mẫu: gọi các hàm autoCreateAdmin và autoCreateCustomer từ các mô hình User và Customer để tự động tạo một người dùng admin và một khách hàng mẫu nếu chúng không tồn tại trong cơ sở dữ liệu.

- Xuất Module: Cuối cùng, xuất hàm dbConnected để có thể sử dụng nó từ các module khác trong ứng dụng.

4.1.2 Models

Hệ thống có 4 mô hình dữ liệu gồm có user(chỉ nhân viên), product, order và customer.

- User (Người dùng- chỉ nhân viên):

```
const bcrypt = require("bcrypt");
const mongoose = require("mongoose");

//Định nghĩa user
const userSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    unique: true,
    match: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/,
  },
  password: {
    type: String,
    required: true
  },
  role: { type: String, enum: ['ADMIN', 'STAFF'], default: 'STAFF' },
  fullName: { type: String, default: "" },
  urlAvatar: { type: String, default: "" },
  gender: { type: String, default: "" },
  birthday: { type: String, default: "" },
  isActive: { type: Boolean, default: false },
  isLock: { type: Boolean, default: false },
  isFirstLogin: { type: Boolean, default: true },
  soldList: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Order' }]
}, { timestamps: true });
```

+ Thuộc tính:

- email: Địa chỉ email của người dùng, phải là một chuỗi không trống, duy nhất và phải phù hợp với định dạng email.
- password: Mật khẩu của người dùng, là một chuỗi không trống.

- role: Quyền hạn của người dùng, chỉ nhận giá trị là 'ADMIN' hoặc 'STAFF', mặc định là 'STAFF'.
- fullName: Họ và tên của người dùng, mặc định là chuỗi trống.
- urlAvatar: Đường dẫn đến hình đại diện của người dùng, mặc định là chuỗi trống.
- gender: Giới tính của người dùng, mặc định là chuỗi trống.
- birthday: Ngày sinh của người dùng, mặc định là chuỗi trống.
- isActive: Trạng thái kích hoạt tài khoản, mặc định là false.
- isLock: Trạng thái khóa tài khoản, mặc định là false.
- isFirstLogin: Trạng thái đăng nhập lần đầu, mặc định là true.
- soldList: Danh sách các đơn hàng đã bán của người dùng, mỗi đơn hàng được đại diện bởi một ObjectId của đối tượng Order.

+ Phương thức:

- autoCreateAdmin: Phương thức tạo tự động một người dùng với quyền 'ADMIN' nếu chưa có. Sử dụng địa chỉ email admin@gmail.com và mật khẩu admin.
- pre('save'): Middleware được sử dụng trước khi lưu một người dùng mới để mã hóa mật khẩu bằng bcrypt.
- checkPassword: Phương thức kiểm tra tính đúng đắn của mật khẩu đã nhập so với mật khẩu đã mã hóa của người dùng.

+ Tính chất khác:

- timestamps: true: Tự động thêm hai trường createdAt và updatedAt để theo dõi thời gian tạo và cập nhật của mỗi bản ghi.

- Product (Sản phẩm)

```
const mongoose = require("mongoose");

const productSchema = new mongoose.Schema({
    barcode: {
        type: String,
        index: true,
        unique: true
    },
    productName: { type: String, default: "" },
    imageUrl: { type: String, default: "" },
    importPrice: Number,
    retailPrice: Number,
    category: String,
    inventory: Number
}, { timestamps: true });


module.exports = mongoose.model('Product', productSchema);

```

+Thuộc tính:

- barcode: Mã vạch của sản phẩm, là một chuỗi được đánh dấu là chỉ số (index) và phải là duy nhất trong cơ sở dữ liệu.
- productName: Tên của sản phẩm, mặc định là chuỗi trống.
- imageUrl: Đường dẫn đến hình ảnh của sản phẩm, mặc định là chuỗi trống.
- importPrice: Giá nhập của sản phẩm, là một số.
- retailPrice: Giá bán lẻ của sản phẩm, là một số.
- category: Danh mục mà sản phẩm thuộc về, là một chuỗi.
- inventory: Số lượng tồn kho của sản phẩm, là một số.

+ Tính chất khác:

- { timestamps: true }: Tự động thêm hai trường createdAt và updatedAt để theo dõi thời gian tạo và cập nhật của mỗi bản ghi.

+ Tính năng:

- Mô hình này được sử dụng để đại diện cho thông tin của mỗi sản phẩm trong cơ sở dữ liệu.

- Các trường như barcode và imageUrl được thiết lập để đảm bảo tính duy nhất và dễ tra cứu.
- Thông tin về giá nhập, giá bán lẻ và số lượng tồn kho giúp quản lý thông tin liên quan đến quá trình bán hàng và quản lý kho.
- Trường timestamps ghi lại thời gian tạo và cập nhật để theo dõi lịch sử thay đổi của sản phẩm.

- Order (Đơn hàng)

```
const mongoose = require("mongoose");

const orderSchema = new mongoose.Schema({
  saleId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  customerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Customer' },
  products: [{
    productId: {type: mongoose.Schema.Types.ObjectId, ref: 'Product'},
    quantity: Number,
    totalPrice: Number
  }],
  totalAmount: Number,
  moneyReceived: Number,
  moneyBack: Number,
  purchaseDate: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Order', orderSchema);
```

+ Thuộc tính:

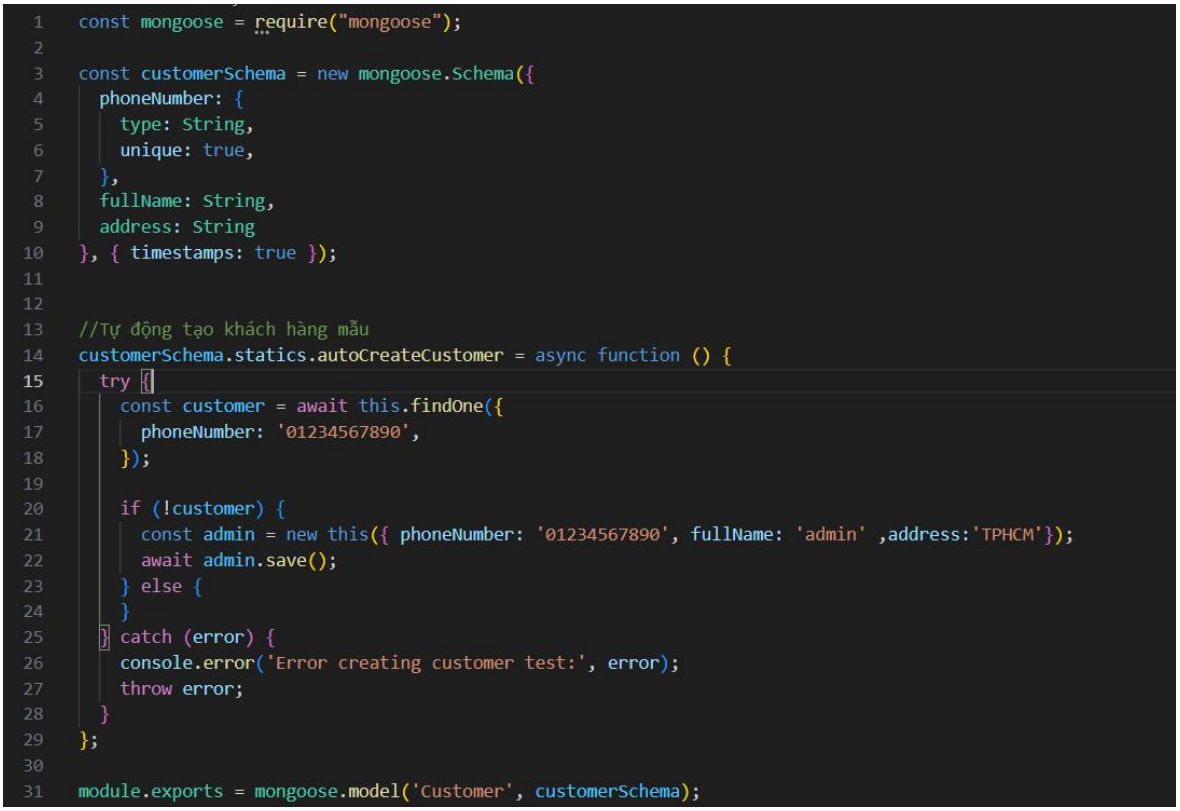
- saleId: Liên kết với người bán hàng (người tạo đơn) thông qua ObjectId và tham chiếu đến mô hình User.
- customerId: Liên kết với thông tin của khách hàng thông qua ObjectId và tham chiếu đến mô hình Customer.
- products: Một mảng chứa thông tin về các sản phẩm trong đơn hàng, mỗi phần tử gồm:
- productId: Liên kết với thông tin của sản phẩm thông qua ObjectId và tham chiếu đến mô hình Product.
- quantity: Số lượng của sản phẩm trong đơn hàng, là một số.

- totalPrice: Tổng giá trị của sản phẩm (số lượng * giá bán lẻ), là một số.
- totalAmount: Tổng giá trị của đơn hàng, là tổng giá trị của tất cả các sản phẩm trong đơn hàng.
- moneyReceived: Số tiền khách hàng trả khi thanh toán đơn hàng, là một số.
- moneyBack: Số tiền trả lại khách hàng (nếu có) sau khi thanh toán, là một số.
- purchaseDate: Ngày và giờ mà đơn hàng được tạo, mặc định là ngày hiện tại.

+ Tính chất khác:

- Mô hình này được thiết kế để theo dõi thông tin chi tiết của mỗi đơn hàng, bao gồm thông tin người bán, thông tin khách hàng và danh sách sản phẩm.
- Các trường saleId, customerId, và productId sử dụng tham chiếu để tạo mối quan hệ giữa các mô hình User, Customer, và Product.
- Trường purchaseDate sử dụng kiểu dữ liệu Date để ghi lại thời gian tạo đơn hàng.

- Customer (khách hàng)



```
const mongoose = require("mongoose");

const customerSchema = new mongoose.Schema({
  phoneNumber: {
    type: String,
    unique: true,
  },
  fullName: String,
  address: String
}, { timestamps: true });


//Tự động tạo khách hàng mẫu
customerSchema.statics.autoCreateCustomer = async function () {
  try {
    const customer = await this.findOne({
      phoneNumber: '01234567890',
    });

    if (!customer) {
      const admin = new this({ phoneNumber: '01234567890', fullName: 'admin' ,address:'TPHCM'});
      await admin.save();
    } else {
    }
  } catch (error) {
    console.error('Error creating customer test:', error);
    throw error;
  }
};

module.exports = mongoose.model('Customer', customerSchema);
```

+ Thuộc tính:

- phoneNumber: Số điện thoại của khách hàng, là một chuỗi và duy nhất trong cơ sở dữ liệu.
- fullName: Họ và tên của khách hàng, là một chuỗi.
- address: Địa chỉ của khách hàng, là một chuỗi.

+ Tính chất khác:

- { timestamps: true }: Tự động thêm hai trường createdAt và updatedAt để theo dõi thời gian tạo và cập nhật của mỗi bản ghi.

+ Phương thức:

- autoCreateCustomer: Phương thức tạo tự động một khách hàng mẫu nếu chưa có trong cơ sở dữ liệu. Sử dụng số điện thoại 01234567890, tên là 'admin' và địa chỉ là 'TPHCM'.

Mô hình Customer này được thiết kế để lưu trữ thông tin về khách hàng trong hệ thống. Các trường như phoneNumber được thiết lập để đảm

bảo tính duy nhất và dễ tra cứu thông tin. Phương thức autoCreateCustomer có thể hữu ích trong quá trình phát triển và kiểm thử để tự động tạo một khách hàng mẫu khi cần thiết.

4.1.3 Services

Trong hệ thống, Services chịu trách nhiệm thực hiện logic kinh doanh cụ thể và tương tác với cơ sở dữ liệu thông qua các model. Các services được thiết kế để đóng gói và tái sử dụng logic, giúp Controllers giữ gọn và tập trung vào việc xử lý yêu cầu từ người dùng. Dưới đây là ví dụ về Services liên quan đến đơn hàng trong file orderService.js.

Đầu tiên cần Import Các Models và thư viện liên quan

```
const Order = require('../models/order');
const moment = require('moment');
```

- Order Model: Đại diện cho cấu trúc dữ liệu của đơn hàng.
- Moment Library: Được sử dụng để xử lý và định dạng thời gian.

- Hàm getOrderService:

```
const getOrderService = async () => {
    try{
        return await Order.find();
    }catch(error) {
        console.error('Error getting list of orders:', error);
        throw error;
    }
}
```

- Mô Tả: Truy vấn cơ sở dữ liệu để lấy danh sách đơn hàng.
- Cơ Chế Hoạt Động: Sử dụng phương thức find của Model Order để lấy tất cả đơn hàng.
- Phản Hồi: Nếu có lỗi, in ra console và ném ra lỗi để Controllers xử lý.

- Hàm createOrderService:

```
const createOrderService = async (orderData) => {
    try{
        console.log(JSON.parse(orderData));
        const order = new Order(JSON.parse(orderData));
        if(await order.save()){
            return order;
        }
        else{
            return false;
        }
    }catch(error) {
        console.error('Error creating order:', error);
        throw error;
    }
}
```

- Mô Tả: Tạo một đơn hàng mới và lưu vào cơ sở dữ liệu.
- Cơ Chế Hoạt Động: Sử dụng phương thức save của Model Order để lưu đơn hàng mới. Trả về đơn hàng nếu thành công, ngược lại trả về false.
- Phản Hồi: Nếu có lỗi, in ra console và ném ra lỗi để Controllers xử lý.

- Hàm getOrderByCustomerIdService:

```
const getOrderByCustomerIdService = async (id) => {
    try{
        const history = await Order.find({customerId: id})
            .populate('saleId')
            .populate('customerId')
            .populate('products.productId')
            .sort({purchaseDate : 1});
        if(history){
            return history
        }else{
            return false;
        }

    }catch(error) {
        console.error('Error getting history purchase:', error);
        throw error;
    }
}
```

- Mô Tả: Lấy lịch sử mua hàng của một khách hàng dựa trên ID.

- Cơ Chế Hoạt Động: Sử dụng phương thức find của Model Order với điều kiện customerId: id. Sử dụng populate để lấy thông tin liên quan từ các model khác. Sắp xếp theo thời gian mua hàng.
- Phản Hồi: Nếu có lỗi, in ra console và ném ra lỗi để Controllers xử lý.

- Hàm getOrderByTimeService:

```
const getOrderByTimeService = async (start,end) => {
    try{
        const startTime = moment(start).startOf('day').toISOString();
        const endTime = moment(end).endOf('day').toISOString();
        return await Order.find({
            purchaseDate: {
                $gte: startTime,
                $lte: endTime
            }
        })
            .populate('saleId')
            .populate('customerId')
            .populate('products.productId')
            .sort({purchaseDate : 1});
    }catch(error) {
        console.error('Error getting list of orders by time:', error);
        throw error;
    }
}
```

- Mô Tả: Lấy danh sách đơn hàng trong khoảng thời gian cụ thể.
- Cơ Chế Hoạt Động: Sử dụng phương thức find của Model Order với điều kiện purchaseDate nằm trong khoảng thời gian được chỉ định. Sử dụng populate để lấy thông tin liên quan từ các model khác. Sắp xếp theo thời gian mua hàng.
- Phản Hồi: Nếu có lỗi, in ra console và ném ra lỗi để Controllers xử lý.

- Xuất Module:

```
module.exports = {getOrderService, createOrderService, getOrderByTimeService, getOrderByCustomerIdService};
```

Tương tự như vậy với các services còn lại.

4.1.4 Controllers

Controllers đóng vai trò quan trọng trong việc xử lý logic kinh doanh và tương tác với dữ liệu từ các services. Controllers là nơi chia nhỏ logic ứng dụng thành các phần có ý nghĩa và dễ bảo trì. Ví dụ về Controllers chịu trách nhiệm cho các chức năng liên quan đến đơn hàng trong file orderController.js.

Cũng giống như route, đầu tiên cần import các dịch vụ liên quan đến đơn hàng cụ thể là orderService.

```
const {getOrderService, createOrderService, getOrderByTimeService, getOrderByCustomerIdService} = require('../services/orderService');
```

Tiếp đến là xây dựng các hàm thể thực các chức năng cụ thể:

- Hàm getOrder:

```
const getOrder = async (req,res) =>{
    try {
        res.json(await getOrderService());
    }catch (error) {
        console.error(error);
        res.redirect('/');
    }
}
```

- Mô Tả: Xử lý yêu cầu GET để lấy danh sách đơn hàng.
- Cơ Chế Hoạt Động: Gọi hàm getOrderService từ dịch vụ để nhận danh sách đơn hàng và trả về dưới dạng JSON.
- Phản Hồi: Nếu có lỗi, chuyển hướng đến trang chủ.

- Hàm createOrder:

```
const createOrder = async (req,res) =>{
    try {
        console.log("áđá")
        console.log(req.body.order)
        res.json( await createOrderService(req.body.order));
    }catch (error) {
        console.error(error);
        res.redirect('/');
    }
}
```

- Mô Tả: Xử lý yêu cầu POST để tạo đơn hàng mới.

- Cơ Chế Hoạt Động: Gọi hàm createOrderService từ dịch vụ và truyền thông tin đơn hàng từ req.body. Trả về kết quả tạo đơn hàng dưới dạng JSON.
- Phản Hồi: Nếu có lỗi, chuyển hướng đến trang chủ.

- Hàm getOrderByTime:

```
const getOrderByTime = async (req,res) =>{
    try {
        res.json(await getOrderByTimeService(req.params.start, req.params.end));
    }catch (error) {
        console.error(error);
        res.redirect('/');
    }
}
```

- Mô Tả: Xử lý yêu cầu GET để lấy danh sách đơn hàng dựa trên khoảng thời gian cụ thể.
- Cơ Chế Hoạt Động: Gọi hàm getOrderByTimeService từ dịch vụ và truyền tham số start và end. Trả về danh sách đơn hàng dưới dạng JSON.
- Phản Hồi: Nếu có lỗi, chuyển hướng đến trang chủ.

- Hàm getHistoryPurchase:

```
const getHistoryPurschase = async (req,res) => {
    try {
        const history = await getOrderByCustomerIdService(req.params.id)
        res.render('history',{user : req.session.user, history : history, moment : require("moment")});
    }catch (error) {
        console.error(error);
        res.redirect('/');
    }
}
```

- Mô Tả: Xử lý yêu cầu GET để hiển thị lịch sử mua hàng của một khách hàng dựa trên ID.
- Cơ Chế Hoạt Động: Gọi hàm getOrderByCustomerIdService từ dịch vụ và truyền tham số id. Nhận lịch sử mua hàng và render trang 'history' sử dụng template engine.
- Phản Hồi: Nếu có lỗi, chuyển hướng đến trang chủ.

Sau khi định nghĩa các hàm thì cần xuất module ra.

Xuất Module: cung cấp các hàm controllers cho việc sử dụng trong routes hoặc các module khác của ứng dụng.

```
module.exports = {getOrder, createOrder, getOrderByTime, getHistoryPurschase}
```

Tương tự như vậy với các controllers còn lại. Các controllers này giúp duy trì tính tổ chức và tái sử dụng của mã nguồn, đồng thời tách biệt logic xử lý và dữ liệu để làm cho hệ thống dễ bảo trì và mở rộng hơn.

4.1.5 Route:

Mỗi Route được định nghĩa trong các tệp tin route, chúng liên kết với một Controller cụ thể. Điều này giúp tách biệt logic xử lý và các endpoint cụ thể của ứng dụng. Mỗi Route định nghĩa một hoặc nhiều endpoint, đại diện cho các đường dẫn URL. Khi một yêu cầu đến, Express sẽ xác định Route tương ứng và kích hoạt các hàm xử lý trong Controller.

Một ví dụ là trong orderRouter.js

```
1  const express = require("express");
2  const {getOrder, createOrder, getOrderByTime} = require('../controllers/orderController');
3
4  const orderRouter = express.Router();
5
6  orderRouter.get('/', (req, res) => {
7      res.render('/order', {user : req.session.user});
8  });
9
10 orderRouter.get('/list', getOrder);
11 orderRouter.get('/:start/:end', getOrderByTime);
12 orderRouter.post('/create', createOrder);
13
14 module.exports = orderRouter;
```

Đầu tiên cần import các hàm xử lý từ orderController để điều hướng các yêu cầu. Tiếp theo tạo một router Express để xử lý các yêu cầu liên quan đến đơn hàng. Sẽ có các route sau:

- route GET đơn giản để hiển thị trang đặt hàng (/order).

  - res.render('/order', { user: req.session.user }); sử dụng template engine (có thể là ejs, pug,...) để render trang và truyền một đối tượng { user: req.session.user } để sử dụng thông tin người dùng trong template.

- route GET để lấy danh sách đơn hàng: yêu cầu sẽ được xử lý bởi hàm getOrder trong orderController.

- route GET để lấy danh sách đơn hàng dựa trên khoảng thời gian cụ thể: yêu cầu sẽ được xử lý bởi hàm getOrderByTime trong orderController.

- route POST để tạo đơn hàng mới: yêu cầu sẽ được xử lý bởi hàm createOrder trong orderController.

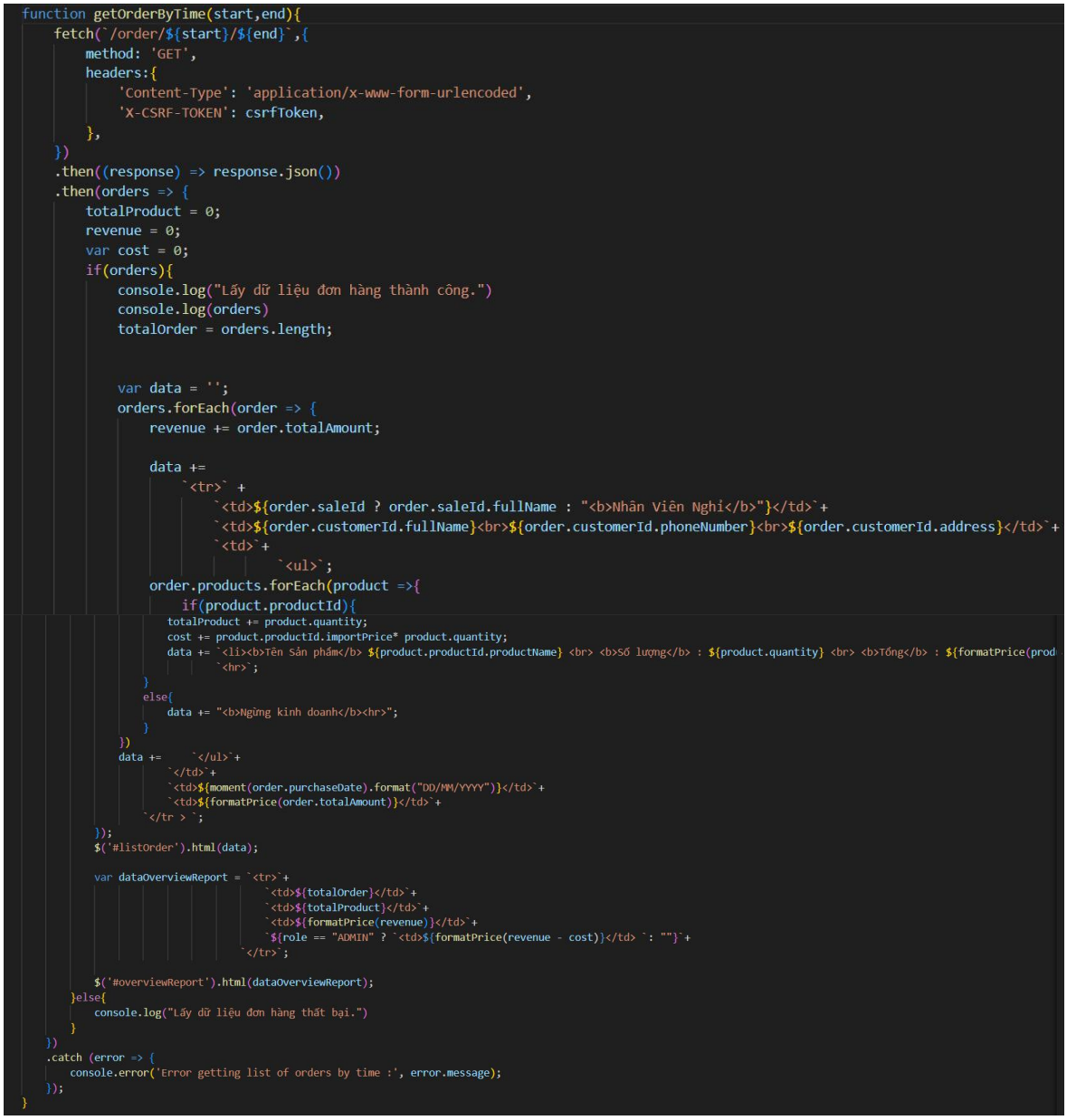
Và dùng module.exports = orderRouter; để router được xuất để có thể sử dụng nó trong tệp tin khác của ứng dụng. Tương tự như vậy với các file trong thư mục route khác trong bài làm.

4.2 Frontend

Phần frontend được viết trong file .ejs kèm với bootstrap giúp giao diện của hệ thống trả nên mượt mà hơn. EJS (Embedded JavaScript) là một ngôn ngữ mẫu (template language) được tích hợp trực tiếp vào JavaScript và được sử dụng để tạo ra HTML động trên máy chủ. Nó cho phép nhúng mã JavaScript vào trong các tệp HTML để thực hiện các logic, vòng lặp, điều kiện, và hiển thị dữ liệu động một cách dễ dàng. Web được chia thành nhiều trang và layout được đặt trong thư mục view và sử dụng fetch API để gọi tới backend.

Phần fetch API của cả hệ thống được lưu trữ trong public/js.

Ví dụ về phần fetch API liên quan đến chức năng thống kê của hệ thống.



```
function getOrderByTime(start,end){
    fetch(`/order/${start}/${end}`,{
        method: 'GET',
        headers:{
            'Content-Type': 'application/x-www-form-urlencoded',
            'X-CSRF-TOKEN': csrfToken,
        },
    })
    .then((response) => response.json())
    .then(orders => {
        totalProduct = 0;
        revenue = 0;
        var cost = 0;
        if(orders){
            console.log("Lấy dữ liệu đơn hàng thành công.")
            console.log(orders)
            totalOrder = orders.length;


            var data = '';
            orders.forEach(order => {
                revenue += order.totalAmount;

                data +=
                    `<tr>` +
                        `<td>${order.saleId ? order.saleId.fullName : "<b>Nhân Viên Nghỉ</b>"}</td>`+
                        `<td>${order.customerId.fullName}<br>${order.customerId.phoneNumber}<br>${order.customerId.address}</td>`+
                        `<td>`+
                                `<ul>`;
                order.products.forEach(product =>{
                    if(product.productId){
                        totalProduct += product.quantity;
                        cost += product.productId.importPrice* product.quantity;
                        data += `<li><b>Tên Sản phẩm</b> ${product.productId.productName} <br> <b>Số lượng</b> : ${product.quantity} <br> <b>Tổng</b> : ${formatPrice(prod
                                `<hr>`;
                    }
                    else{
                        data += "<b>Ngừng kinh doanh</b><hr>";
                    }
                })
                data +=     `</ul>`+
                        `</td>`+
                        `<td>${moment(order.purchaseDate).format("DD/MM/YYYY")}</td>`+
                        `<td>${formatPrice(order.totalAmount)}</td>`+
                    `</tr > `;
            });
            $('#listOrder').html(data);

            var dataOverviewReport = `<tr>`+
                                        `<td>${totalOrder}</td>`+
                                        `<td>${totalProduct}</td>`+
                                        `<td>${formatPrice(revenue)}</td>`+
                                        `${role == "ADMIN" ? `<td>${formatPrice(revenue - cost)}</td> `: ""}`+
                                    `</tr>`;

            $('#overviewReport').html(dataOverviewReport);
        }else{
            console.log("Lấy dữ liệu đơn hàng thất bại.")
        }
    })
    .catch (error => {
        console.error('Error getting list of orders by time :', error.message);
    });
}
```

- Sử dụng fetch để gửi yêu cầu GET đến endpoint /order/${start}/${end} trên máy chủ.

- Thiết lập header của yêu cầu, bao gồm 'Content-Type': 'application/x-www-form-urlencoded' và 'X-CSRF-TOKEN' (CSRF token, giả sử đã được định nghĩa trước đó).

- Xử lý phản hồi từ máy chủ, chuyển đổi nó thành đối tượng JSON.

- Tính toán các thống kê như tổng số sản phẩm, doanh thu, và chi phí từ dữ liệu đơn hàng nhận được.

- Tạo chuỗi HTML để hiển thị thông tin đơn hàng và thông tin tổng hợp báo cáo.
- Sử dụng jQuery để cập nhật nội dung của các phần tử HTML với ID listOrder và overviewReport.

Và tương tự như vậy với các fetch API khác như product, customer

4.3 Run code

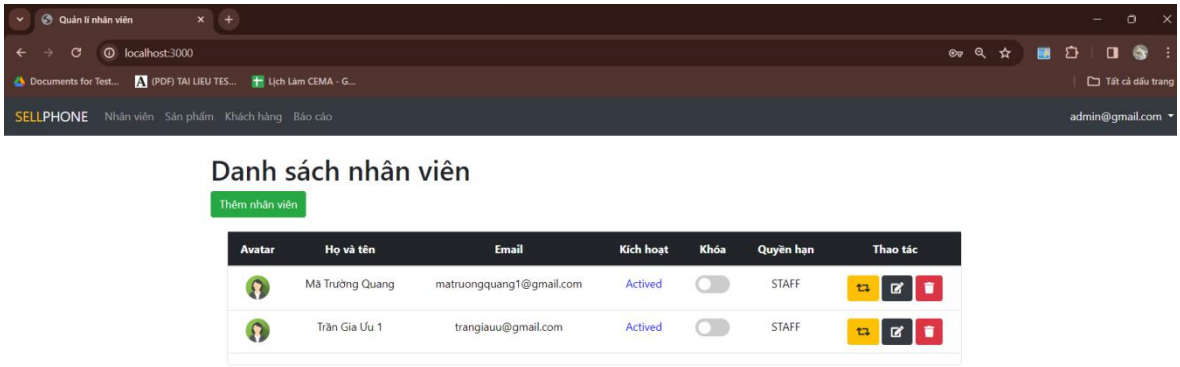
File dùng để chạy code trong hệ thống này là file index.js. file index.js thường đóng vai trò là điểm khởi đầu của ứng dụng. Trong file cần thực hiện các công việc:

- Cài đặt các thư viện và module:
- Kết nối đến cơ sở dữ liệu: import module mongoose để kết nối đến cơ sở dữ liệu MongoDB, sử dụng thông tin cấu hình từ file config.js để xác định các thông số kết nối, kết nối đến cơ sở dữ liệu thông qua hàm dbConnected.
- Khởi tạo ứng dụng Express: Tạo đối tượng ứng dụng Express thông qua hàm express(). Cấu hình ứng dụng như sử dụng middleware, thiết lập template engine (ejs), và cài đặt các cấu hình an ninh.
- Kết nối đến cơ sở dữ liệu và tạo người dùng mẫu: Gọi hàm dbConnected() để kết nối đến cơ sở dữ liệu MongoDB. Gọi các phương thức khởi tạo người dùng mẫu từ các mô hình như autoCreateAdmin và autoCreateCustomer.
- Cấu hình định tuyến (Routing): Định nghĩa các router và liên kết chúng với các đối tượng controllers. Sử dụng các middleware như checkAuth và checkAdmin để kiểm tra quyền truy cập.
- Xử lý lỗi và lắng nghe cổng: Định nghĩa các middleware để xử lý lỗi và chuyển hướng đến các trang 404 và 500. Lắng nghe cổng được xác định trong tệp config.js và in ra console khi ứng dụng được khởi chạy.

CHƯƠNG 5. KẾT QUẢ ĐẠT ĐƯỢC

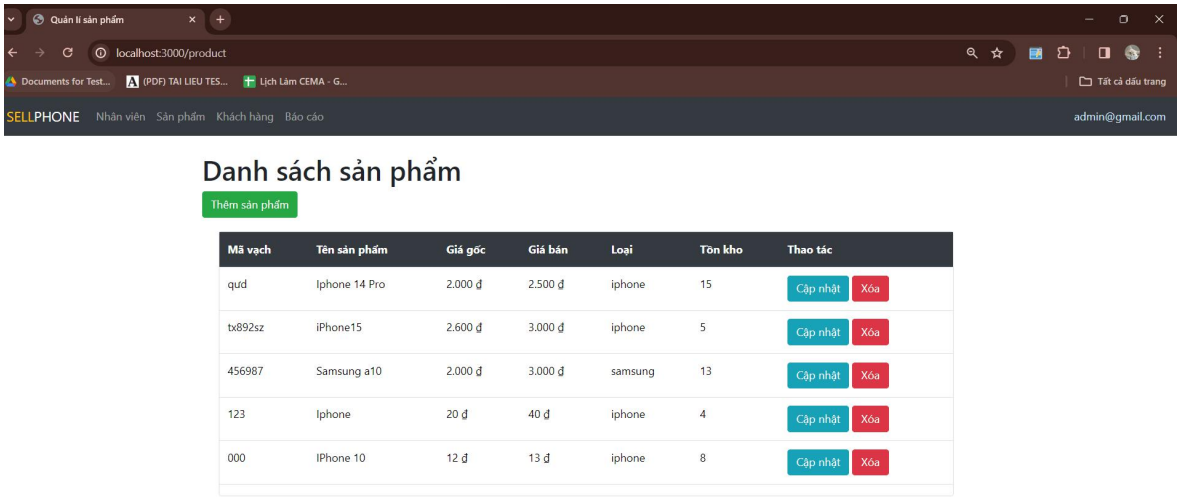
Sau khi hoàn thành xong đồ án thì nhóm chúng em đã thu được rất nhiều kết quả và sau đây là một số kết quả đáng được nới đến:

Nhóm chúng em đã tiến hành được phân chia quyền quản trị cho Admin và Staff. Sau đó chúng em đã sử dụng nó để thực hiện tiếp đồ án của mình là chỉ có Admin mới có quyền quản lí nhân viên, ở đây thì có quyền thêm nhân viên và khi thêm nhân viên thì hệ thống sẽ gửi một mail về cho nhân viên đó và nhân viên đó thông qua mail đó thể đăng nhập vào hệ thống.



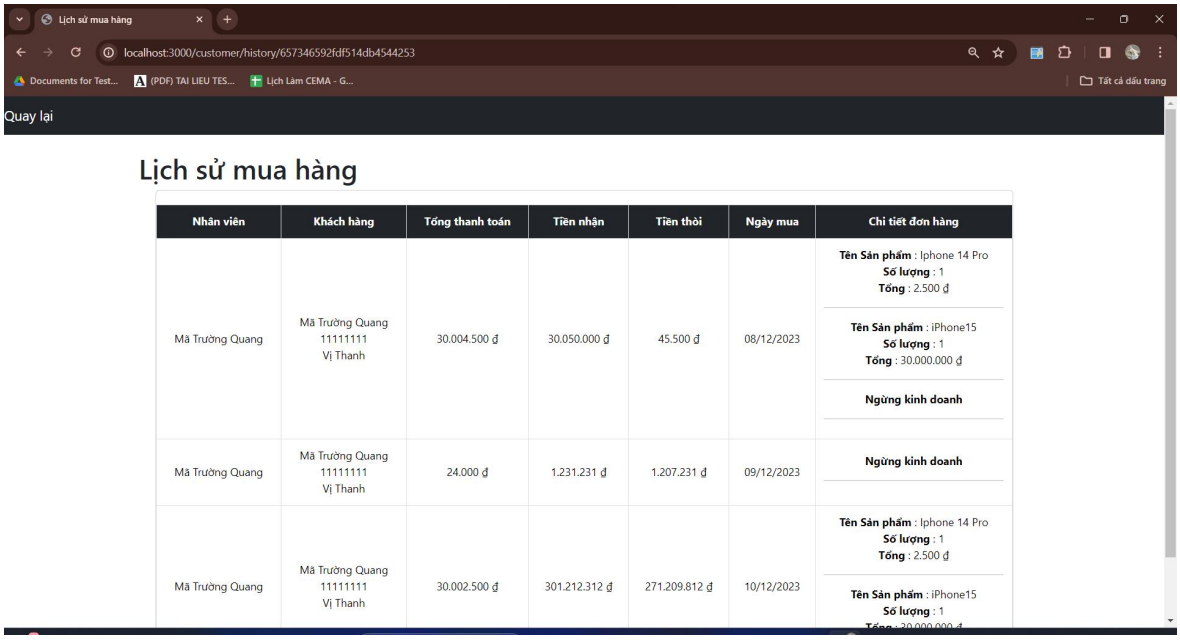
Hình 21 Danh sách nhân viên

Chức năng quản lí sản phẩm, chức năng này cũng chỉ áp dụng cho Admin mới có quyền thao tác thêm, xóa, sửa nó thoi còn Staff chỉ có quyền xem chứ không được thao tác.



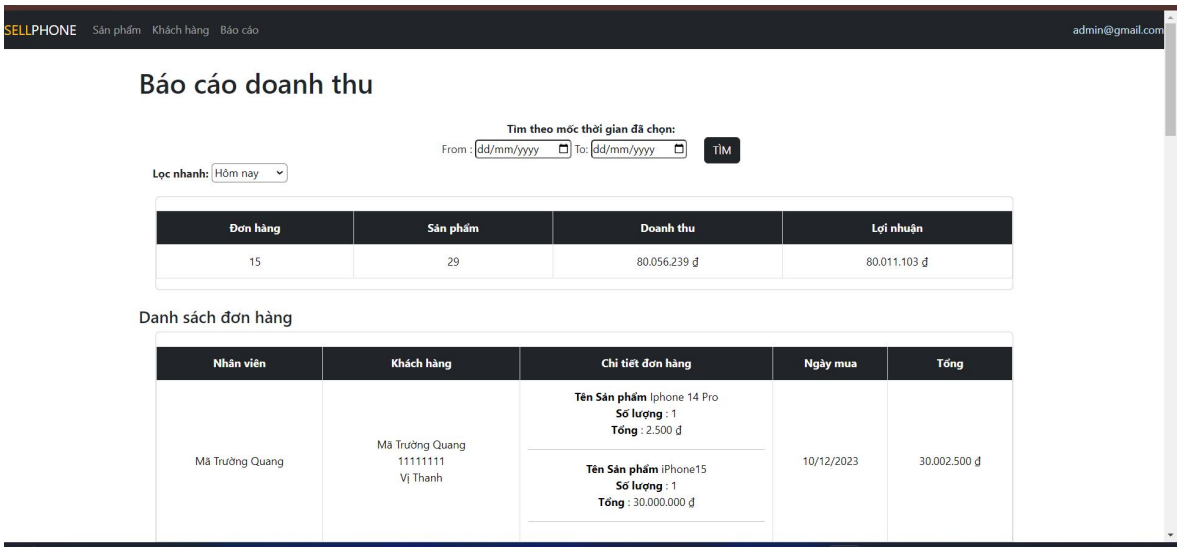
Hình 22 Danh sách sản phẩm

Chức năng lịch sử mua hàng cho từng khách hàng, chúng ta có thể xem cụ thể những đơn hàng mà khách hàng đó đã mua cũng như là khách hàng đó đã mua gì thông qua chức năng lịch sử mua hàng này.



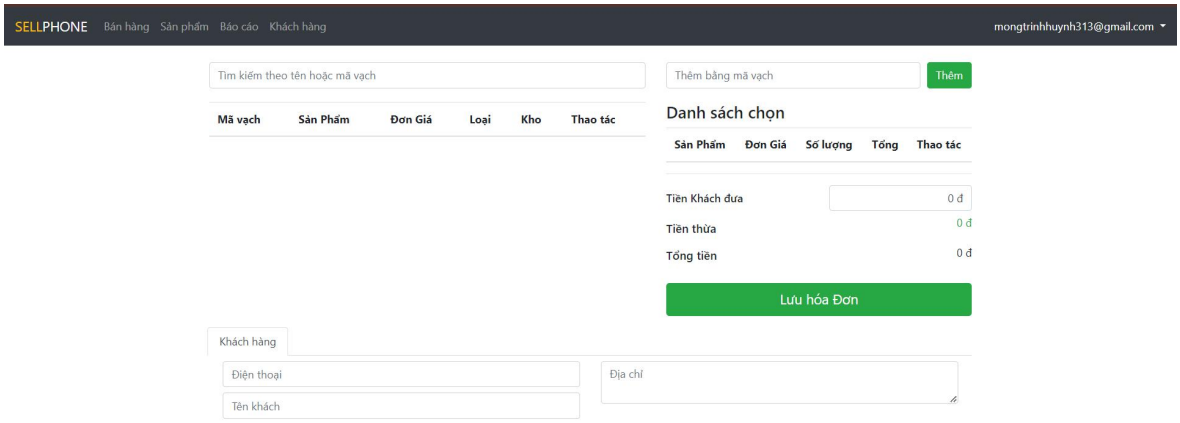
Hình 23 Lịch sử mua hàng

Chức năng báo cáo doanh thu, chắc năng này sẽ cho chúng ta biết doanh thu của cửa hàng mình bán. Ở đây có chức năng lọc nhanh hoặc lọc theo ngày và ở đây cũng có chức năng phân quyền ở đây là chỉ có admin mới có quyền thấy được lợi nhuận thoi còn nhân viên thì sẽ không thấy được.



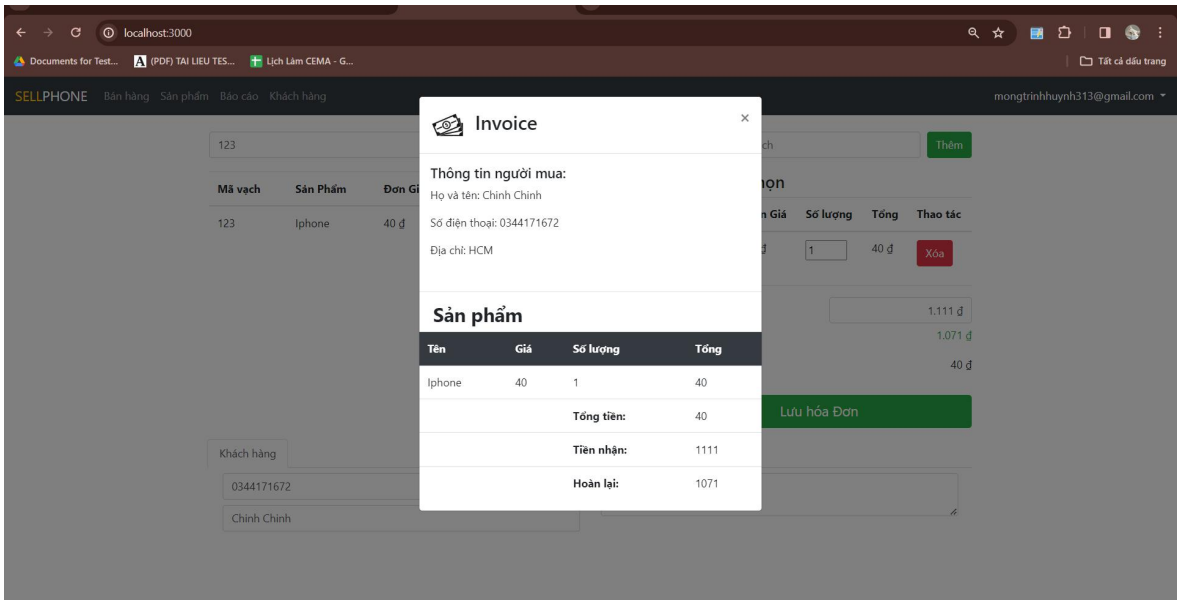
Hình 24 Báo cáo doanh thu

Chức năng bán hàng này chỉ áp dụng cho nhân viên bán hàng, ở đây người bán có thể tìm theo tên sản phẩm hoặc cũng có thể tìm theo mã của sản phẩm đó. Nếu bạn có mã sản phẩm chính xác thì bạn có thể tìm trực tiếp nó có mục thêm trực tiếp thông qua mã vạch. Khi bạn bán hàng sẽ có mục nhập thông tin khách hàng. Ở đây nếu khách hàng đã mua sản phẩm thì mua hàng nó sẽ tự động điền nếu chưa mua thì tiến hành xin thông tin khách hàng và nếu không nhập thì hệ thống sẽ báo lỗi. Có mục nhập số tiền khách đưa nếu khách không đưa hoặc số tiền ít hơn tổng số thì hệ thống cũng sẽ báo lỗi.



Hình 25 Bán hàng

Khi nhấn lưu hóa đơn thì hóa đơn sẽ hiện lên, hóa đơn sẽ chứa đầy đủ thông tin khách hàng cũng như sản phẩm và số tiền của khách hàng.



Hình 26 Hóa đơn

CHƯƠNG 6. KẾT LUẬN

6.1 Ưu điểm:

- Giao diện dễ tiếp cận với người dùng
- Có chức năng xác nhận thông tin đăng nhập

- Phân quyền quản trị để tiện quản lý

6.2 Hạn chế:

- Tốc độ còn chậm
- Layout chưa được đẹp

6.3 Làm được:

- Quản lý tài khoản
- Quản lý người dùng
- Quản lý sản phẩm
- Quản lý khách hàng
- Xử lý giao dịch
- Báo cáo và phân tích

6.4 Hướng phát triển:

- Cải thiện giao diện cho đẹp hơn
- Phát triển ứng dụng cải thiện tốc độ
- Làm nhiều chức năng hơn

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Longnv (17/01/2021) Lập trình MVC trong NodeJS https://longnv.name.vn/lap-trinh-nodejs/lap-trinh-mvc-trong-nodejs

2. Hương Thanh (11/05/2021) Toàn bộ lý thuyết về kiến trúc Node.js https://techmaster.vn/posts/36460/toan-bo-ly-thuyet-ve-kien-truc-node-js

3. Châu IT (21/09/2021) MVC trong Express và ví dụ minh họa https://viblo.asia/p/mvc-trong-express-va-vi-du-minh-hoa-924lJ8nNKPM

Tiếng Anh

1. Emmanuel Etukudo (03/03/2021) Understanding MVC pattern in Nodejs https://dev.to/eetukudo_/understanding-mvc-pattern-in-nodejs-2bdn

2. JavaScript Today (04/03/2023) Understanding MVC Architecture: Beginner Tutorial with Node.js https://blog.javascripttoday.com/blog/model-view-controller-architecture/

3. W3Schools Node.js MongoDB https://www.w3schools.com/nodejs/nodejs_mongodb.asp