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

BÁO CÁO CUỐI KÌ MÔN NODEJS

HỆ THỐNG BÁN ĐIỆN THOẠI

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

Người thực hiện: **NGUYỄN QUỐC ANH – 52100871**

LỤC MINH HIẾU – 52100889

VÕ PHÚ VINH – 52100947

Lớp **: 21050301**

Khoá **: 25**

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

BÁO CÁO CUỐI KÌ MÔN NODEJS

HỆ THỐNG BÁN ĐIỆN THOẠI

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

Người thực hiện: **NGUYỄN QUỐC ANH – 52100871**

LỤC MINH HIẾU – 52100889

VÕ PHÚ VINH - 52100947

Lớp **: 21050301**

Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Kính thưa thầy Vũ Đình Hồng,

Với lòng kính trọng và biết ơn sâu sắc, nhóm em xin gửi lời cảm ơn chân thành tới thầy, người đã giảng dạy kiến thức và hướng dẫn em hoàn thành bài báo cáo cuối kì này.

Đồng thời em cũng xin được gửi lời cảm ơn đến quý thầy, cô khoa Công Nghệ Thông Tin - những người đã truyền lửa và giảng dạy kiến thức cho em suốt thời gian qua.

Mặc dù đã có những đầu tư nhất định trong quá trình làm bài song cũng khó có thể tránh khỏi những sai sót, nhóm em kính mong nhận được ý kiến đóng góp của thầy để bài báo cáo được hoàn thiện hơn.

Em xin chân thành cảm ơn!

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi / chúng tôi và được sự hướng dẫn của TG Trần Bảo Tín;. 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 đồ 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 đồ 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 tháng 11 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Nguyễn Quốc Anh

Lục Minh Hiếu

Võ Phú Vinh

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

MỤC LỤC

DANH MỤC HÌNH

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Khái quát về đề tài và lý do thực hiện

Trong môi trường bán lẻ nhanh chóng của cửa hàng điện thoại và phụ kiện, việc quản lý và xử lý bán hàng một cách hiệu quả là chìa khóa cho sự thành công kinh doanh. Nhận thức được nhu cầu này, chúng em đã phát triển một ứng dụng web toàn diện, được thiết kế đặc biệt cho những yêu cầu riêng biệt của các cửa hàng bán lẻ điện thoại. Ứng dụng này được tạo ra nhằm mục đích tối ưu hóa các giao dịch bán hàng, quản lý sản phẩm, phối hợp nhân viên, và cung cấp báo cáo chi tiết cũng như thống kê. Công cụ này không chỉ cần thiết để cải thiện hiệu quả hoạt động mà còn nâng cao chất lượng dịch vụ khách hàng trong lĩnh vực yêu cầu dịch vụ nhanh chóng, chính xác và hiệu quả. Loại ứng dụng này đang ngày càng trở thành một phần không thể thiếu trong các ngành bán lẻ tại Việt Nam, bao gồm điện tử, quần áo và siêu thị, phản ánh xu hướng chuyển đổi số ngày càng tăng trong ngành công nghiệp bán lẻ.

1.2 Những yêu cầu của đề tài

1.2.1 Quản lý tài khoản

- Tài khoản quản trị viên được tạo sẵn với thông tin đăng nhập mặc định là admin/admin.
- Quản trị viên có thể thay đổi mật khẩu và tạo tài khoản cho nhân viên bán hàng bằng cách cung cấp thông tin cần thiết như tên đầy đủ và địa chỉ Gmail.
- Liên kết đăng nhập gửi đến nhân viên bán hàng chỉ có hiệu lực trong 1 phút.
- Mỗi người dùng sử dụng phần trước của email làm tên đăng nhập.
- Nhân viên bán hàng phải tạo mật khẩu mới khi đăng nhập lần đầu.

1.2.2 Quản lý danh mục sản phẩm

- Cho phép thêm mới, cập nhật, xem danh sách và xoá sản phẩm.
- Thông tin sản phẩm bao gồm mã vạch, tên, giá nhập, giá bán, ngày tạo, và loại sản phẩm.

1.2.3 Quản lý khách hàng

- Khi thanh toán, nhân viên sẽ hỏi số điện thoại khách hàng và hệ thống tự động hiển thị thông tin nếu khách hàng đã mua hàng trước đó.
- Nhân viên có thể xem thông tin cá nhân và lịch sử mua hàng của khách hàng.

1.2.4 Xử lý giao dịch

- Nhân viên nhập sản phẩm vào hóa đơn qua tìm kiếm hoặc nhập mã vạch.
- Thông tin đơn hàng hiển thị trực quan bao gồm số lượng, đơn giá và tổng cộng.
- Có chức năng nhập thông tin khách hàng mới và hoàn tất quá trình thanh toán.

1.2.5 Báo cáo và phân tích

- Hiển thị kết quả bán hàng theo các khoảng thời gian cụ thể.
- Cung cấp thông tin chi tiết về tổng số tiền thu được, số lượng đơn hàng, số lượng sản phẩm, và danh sách đơn hàng.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Giới thiệu Nodejs

2.1.1 Khái niệm

Node.js là một nền tảng phần mềm mã nguồn mở chạy trên môi trường JavaScript V8 Engine của Google. Nó được thiết kế để xây dựng các ứng dụng mạng nhanh chóng và có khả năng mở rộng dễ dàng. Node.js sử dụng mô hình non-blocking, event-driven, giúp nó trở nên hiệu quả và phù hợp cho các ứng dụng thời gian thực trên các thiết bị phân tán. Điểm độc đáo của Node.js là nó cho phép các nhà phát triển sử dụng JavaScript để viết cả phần mềm phía máy chủ lẫn phía máy khách, qua đó tạo nên một môi trường phát triển đồng nhất và linh hoạt.

2.1.2 Lịch sử hình thành

Node.js được Ryan Dahl tạo ra vào năm 2009, với ý tưởng chính là cung cấp một cách để xây dựng các ứng dụng mạng nhanh chóng và dễ mở rộng. Sự ra đời của Node.js đã mang đến một sự thay đổi lớn trong cách xây dựng phần mềm, đặc biệt là ứng dụng

web, bằng cách sử dụng JavaScript. Node.js nhanh chóng trở nên phổ biến trong cộng đồng phát triển web nhờ vào khả năng xử lý đồng thời nhiều kết nối mà không làm giảm hiệu suất. Sự phát triển liên tục của nó cùng với sự hỗ trợ mạnh mẽ từ cộng đồng mã nguồn mở đã giúp Node.js trở thành một công cụ không thể thiếu trong ngành công nghiệp phần mềm hiện đại.

2.2 Kiến trúc và các đặc điểm kỹ thuật

2.2.1 Kiến trúc chính của Nodejs

2.2.1.1 V8 Engine

- Node.js sử dụng V8 Engine, động cơ JavaScript được phát triển bởi Google cho trình duyệt Chrome. V8 Engine biên dịch JavaScript sang mã máy để tăng tốc độ thực thi.
- Điều này giúp Node.js tận dụng tốc độ và hiệu suất của V8, cho phép thực thi các ứng dụng JavaScript một cách nhanh chóng.

2.2.1.2 Event Loop

- Node.js hoạt động dựa trên mô hình event-driven, bất đồng bộ (asynchronous). Event loop là cơ chế cho phép Node.js thực hiện các tác vụ bất đồng bộ mà không cần đa luồng (multi-threading).
- Event loop xử lý các sự kiện (như I/O) một cách tuần tự, giúp giảm bớt sự phức tạp trong việc quản lý nhiều luồng và tránh tình trạng chặn (blocking).

2.2.1.3 Non-blocking I/O

- Node.js sử dụng mô hình I/O không chặn (non-blocking I/O), cho phép xử lý nhiều yêu cầu cùng một lúc mà không phải chờ đợi hoàn thành từng yêu cầu.
- Điều này làm tăng hiệu quả xử lý, đặc biệt trong các ứng dụng cần xử lý lượng lớn yêu cầu I/O, như ứng dụng web hoặc API dịch vụ.

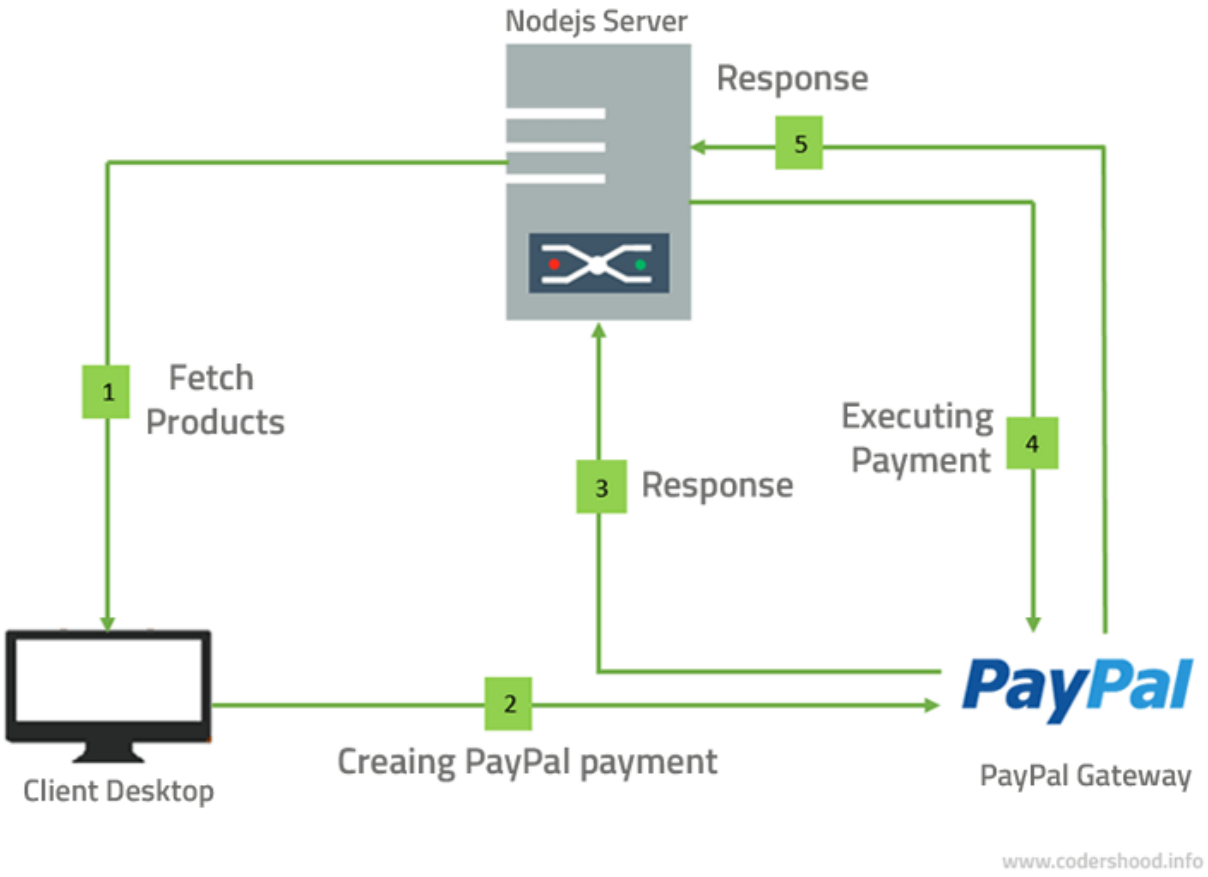
2.2.2 Ngôn ngữ lập trình và cách sử dụng

- Node.js được xây dựng trên ngôn ngữ lập trình JavaScript. Điều này có nghĩa là các nhà phát triển có thể sử dụng cùng một ngôn ngữ để viết mã cho cả phía máy chủ và máy khách, tạo nên sự đồng nhất trong phát triển ứng dụng.
- Sử dụng JavaScript trong Node.js không chỉ giới hạn ở việc xây dựng các ứng dụng máy chủ. Nó cũng mở rộng sang việc xây dựng các công cụ, thư viện, và nhiều ứng dụng khác nhờ vào hệ sinh thái npm (Node Package Manager) phong phú và đa dạng.
- JavaScript trong Node.js cho phép tận dụng các tính năng của ngôn ngữ như hàm callback, Promise, và async/await để xử lý các hoạt động không đồng bộ, làm cho mã nguồn dễ viết, dễ đọc và dễ bảo trì hơn.

2.3 Nodejs trong phát triển Web

- Xây dựng API và Backend: Node.js rất phổ biến để xây dựng API cho các ứng dụng web và di động. Nó hỗ trợ JSON (JavaScript Object Notation) một cách tự nhiên, làm cho việc trao đổi dữ liệu giữa client và server trở nên mượt mà.
- Ứng dụng thời gian thực: Node.js là sự lựa chọn lý tưởng cho các ứng dụng cần xử lý thời gian thực như trò chuyện trực tuyến, trò chơi trực tuyến và các ứng dụng cộng tác trực tuyến, nhờ vào khả năng xử lý đồng thời và hiệu suất cao.
- Single Page Applications (SPA): Node.js phù hợp cho việc phát triển các ứng dụng trang đơn (SPA), nơi mà một lượng lớn trao đổi dữ liệu diễn ra giữa client và server mà không cần tải lại trang.

Ví dụ web phổ biến sử dụng Nodejs:



Hình 2.1 Paypal

PayPal chuyển đổi một phần lớn của ứng dụng web của họ sang Node.js, điều này giúp cải thiện tốc độ phản hồi của ứng dụng và giảm thời gian phát triển.

2.4 Giới thiệu Mô-đun và thư viện phổ biến

2.4.1 Hệ sinh thái NPM (Node Package Manager)

- NPM là trình quản lý gói mặc định cho Node.js, cho phép người dùng cài đặt, cập nhật, và quản lý các thư viện và công cụ phần mềm.
- Với hơn một triệu gói (packages) khác nhau, NPM là một trong những hệ thống quản lý gói lớn nhất thế giới, cung cấp một kho tàng lớn các thư viện và công cụ cho các nhà phát triển.

- NPM giúp dễ dàng chia sẻ và tái sử dụng mã nguồn, giúp quá trình phát triển trở nên nhanh chóng và hiệu quả hơn.

2.4.2 Một số mô-đun và thư viện phổ biến

- Express.js: Đây là một framework cho Node.js, được sử dụng rộng rãi để xây dựng ứng dụng web và API. Nó cung cấp một cách đơn giản để tạo ra các server HTTP và xử lý các yêu cầu từ client.
- Mongoose: Mongoose là một ODM (Object Data Modeling) library cho MongoDB và Node.js. Nó giúp quản lý mối quan hệ giữa dữ liệu, cung cấp cơ sở dữ liệu schema validation và được sử dụng để làm việc với MongoDB.
- Async: cung cấp các hàm tiện ích để làm việc với hoạt động không đồng bộ trong JavaScript. Nó giúp quản lý chuỗi các hoạt động không đồng bộ một cách dễ dàng.

Các mô-đun và thư viện này chỉ là một phần nhỏ của hệ sinh thái phong phú mà Node.js cung cấp, giúp các nhà phát triển tùy chỉnh và mở rộng các ứng dụng của họ theo nhu cầu cụ thể.

2.5 Ưu điểm và nhược điểm

2.5.1 Ưu điểm

- Hiệu suất cao: Node.js sử dụng V8 JavaScript Engine của Google, giúp biên dịch mã JavaScript sang mã máy nhanh chóng. Điều này mang lại hiệu suất cao cho ứng dụng. Cơ chế xử lý không đồng bộ và event-driven của Node.js giúp tăng cường khả năng xử lý nhiều kết nối cùng lúc một cách hiệu quả.
- Mô hình Non-blocking: Node.js xử lý các yêu cầu một cách không đồng bộ, giúp tránh hiện tượng chặn (blocking) và giảm thời gian chờ đợi, đặc biệt hữu ích trong các ứng dụng cần xử lý một lượng lớn yêu cầu đồng thời.
- Cộng đồng mạnh mẽ và hệ sinh thái phong phú: Node.js có một cộng đồng lớn và năng động cùng với NPM, cung cấp hàng triệu thư viện và công cụ, giúp tăng tốc độ phát triển và hỗ trợ nhiều tính năng.

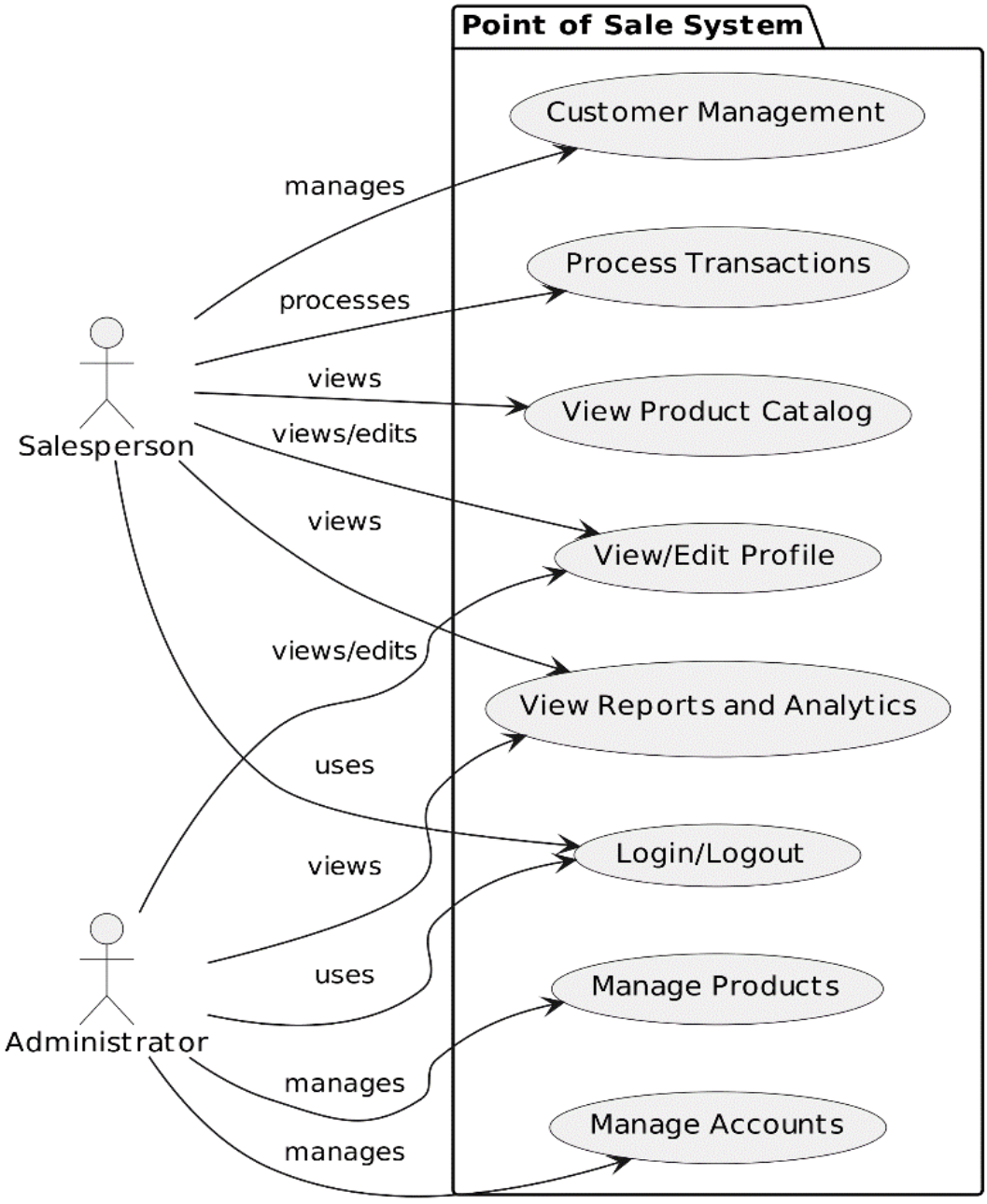
2.5.2 Nhược điểm

- Quản lý bộ nhớ: Vì Node.js sử dụng JavaScript, nó kế thừa các hạn chế của JavaScript về quản lý bộ nhớ. JavaScript không phải là ngôn ngữ tối ưu nhất cho việc xử lý các tác vụ yêu cầu bộ nhớ lớn.
- Xử lý các tác vụ tính toán nặng: Node.js không phải là lựa chọn tốt nhất cho các ứng dụng yêu cầu xử lý tính toán nặng và đồng bộ, vì mô hình non-blocking của nó có thể dẫn đến việc chậm trễ trong việc xử lý các tác vụ đó.
- Độ ổn định của một số mô-đun: Mặc dù có một hệ thống module phong phú, nhưng không phải tất cả các module trong NPM đều được duy trì và cập nhật thường xuyên. Điều này có thể gây ra các vấn đề về ổn định và bảo mật.

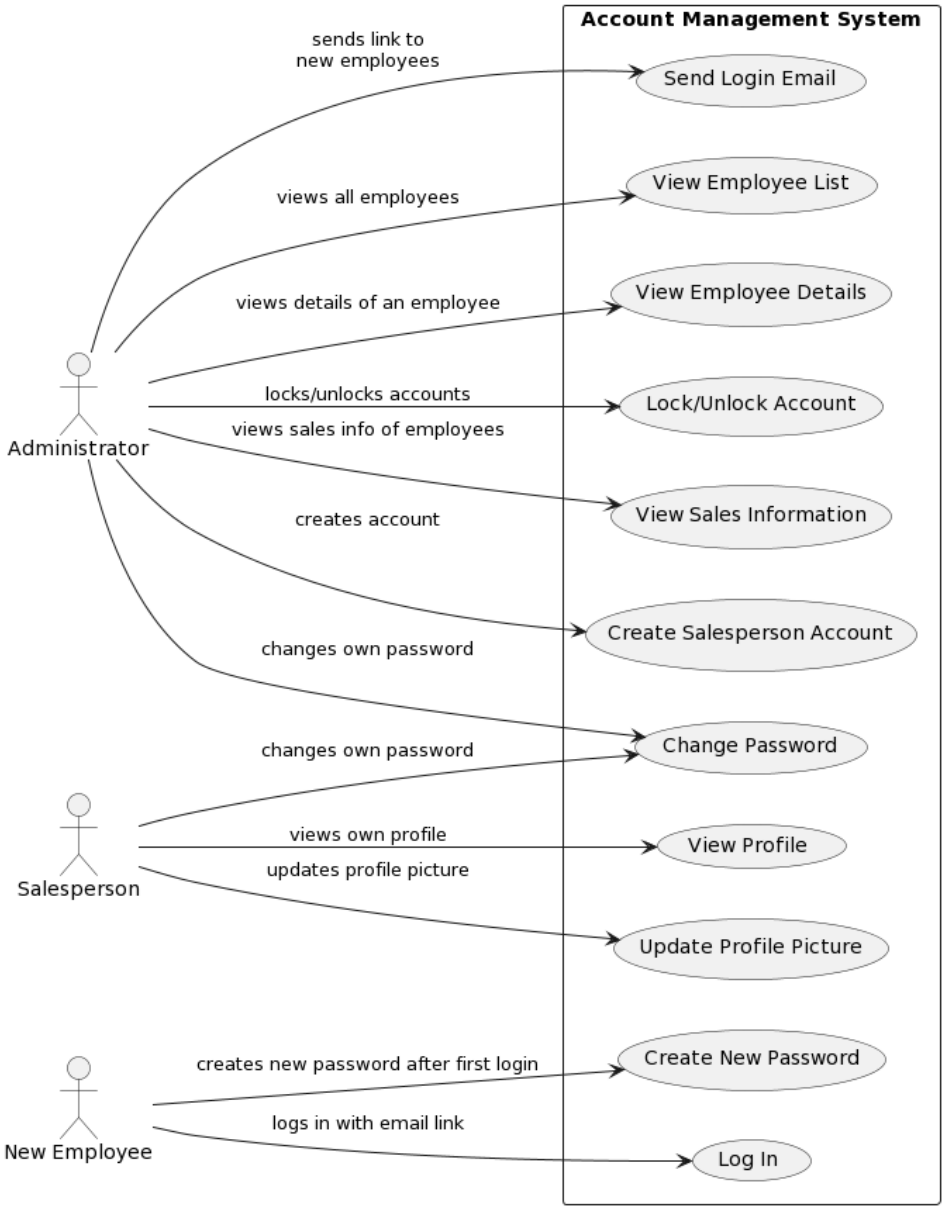
CHƯƠNG 3. PHÂN TÍCH THIẾT KẾ

3.1 ERD Database

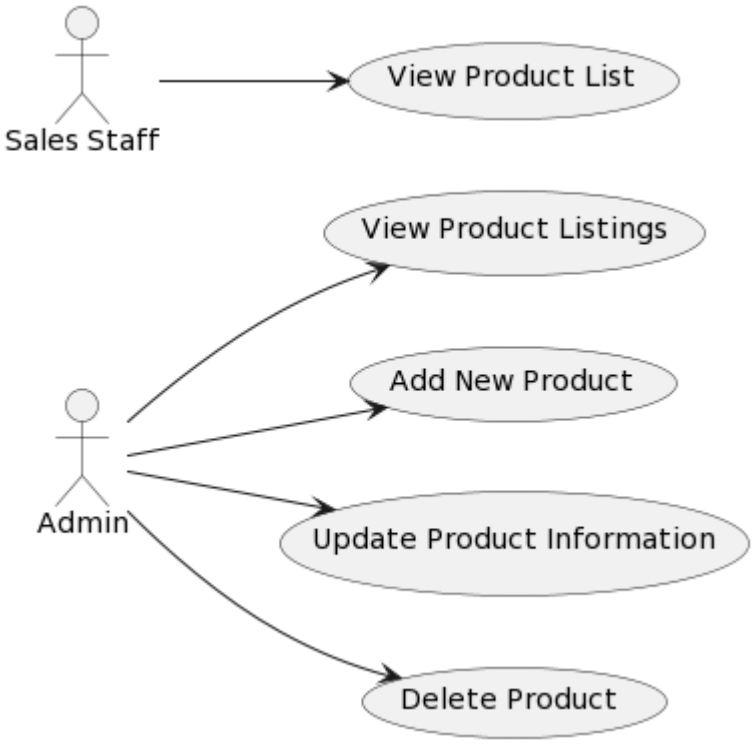
3.2 Usecase diagram



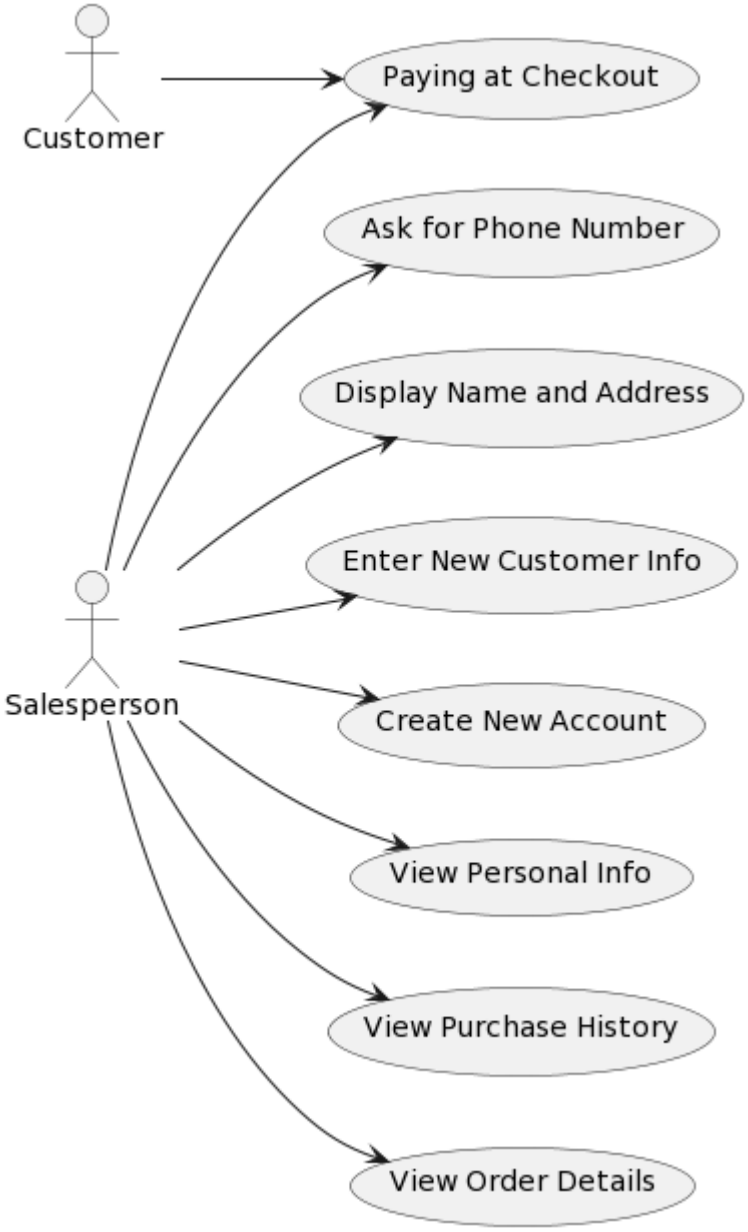
Hình 3.1 Usecase Diagram POS System



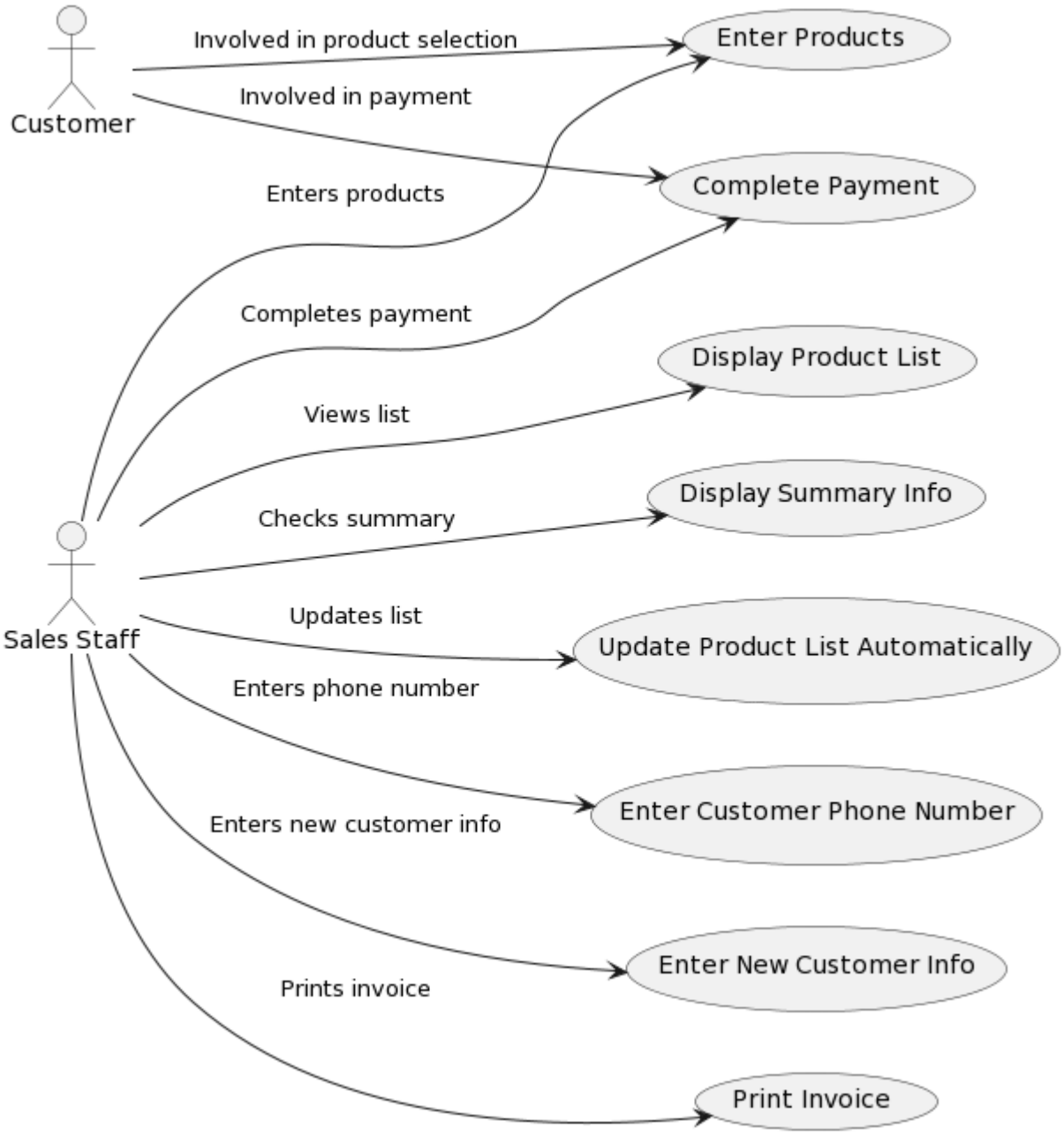
Hình 3.2 Usecase Diagram Account Management



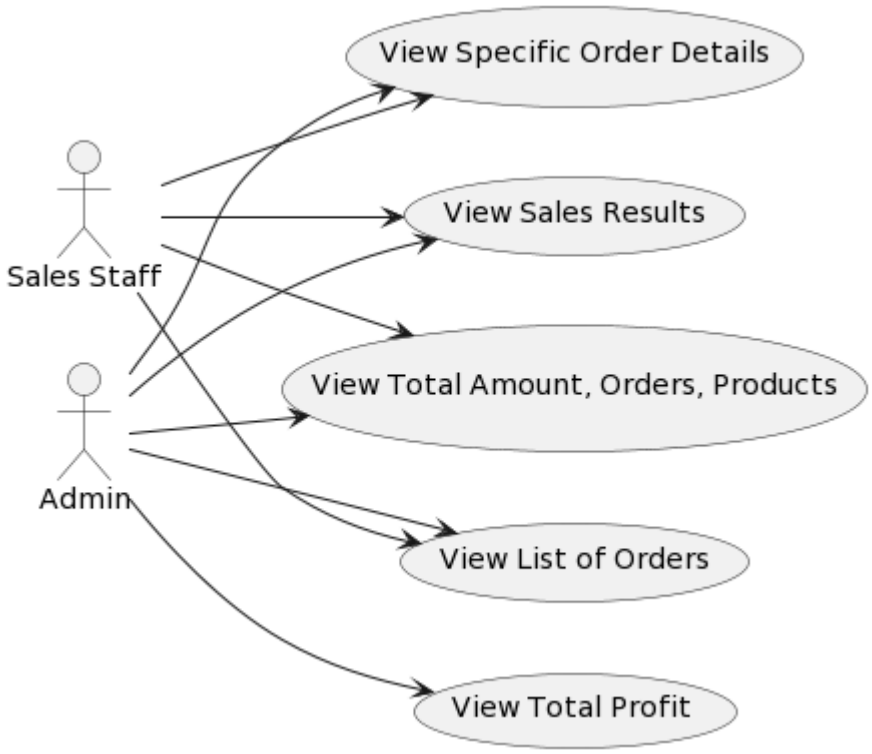
Hình 3.3 Usecase Diagram Product Catalog Management



Hình 3.4 Usecase Diagram Customer Management

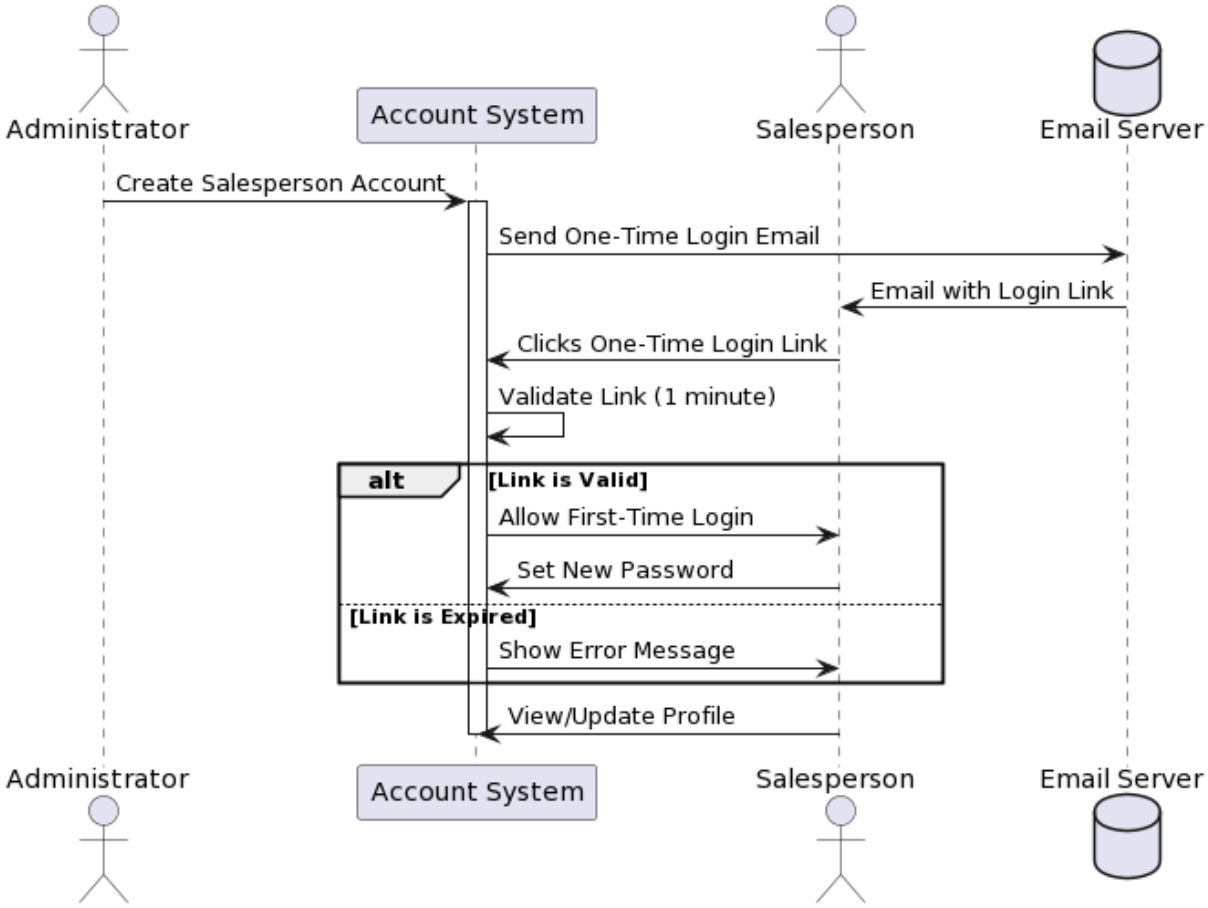


Hình 3.5 Usecase Diagram Transaction Processing

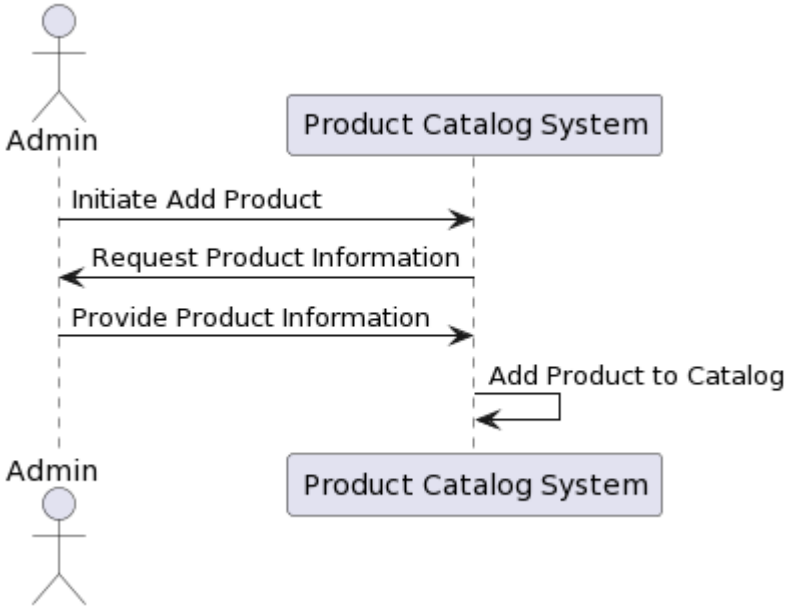


Hình 3.6 Usecase Diagram Reporting & Analytics

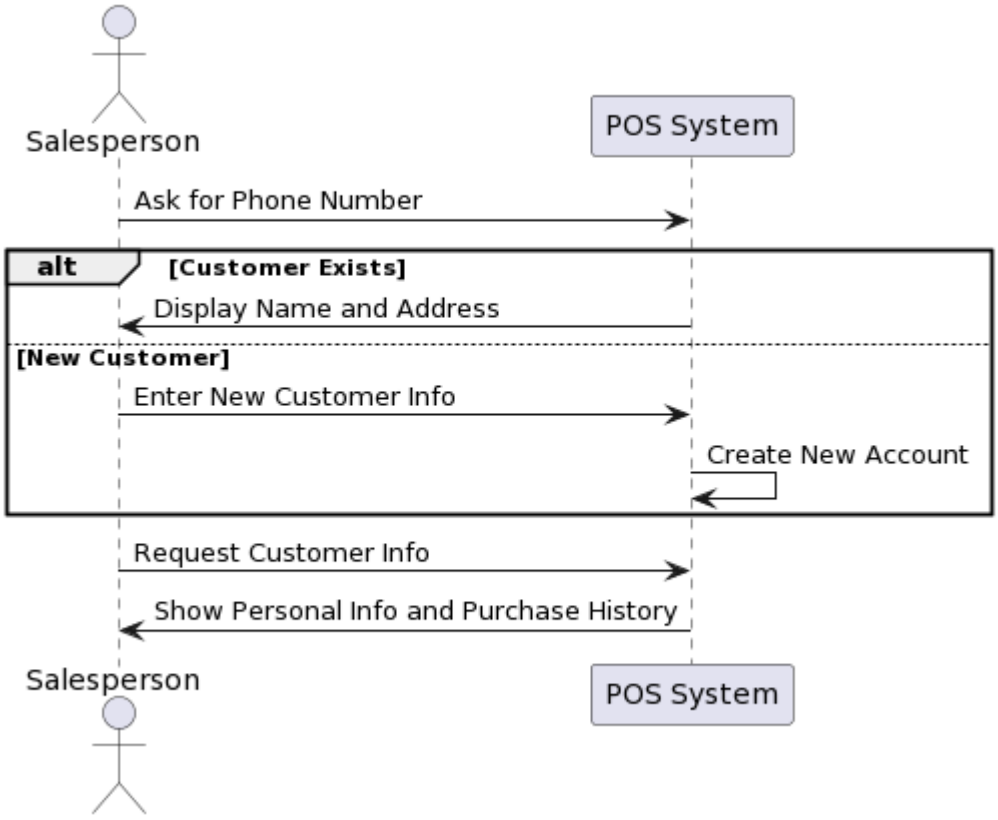
3.3 Sequence Diagram



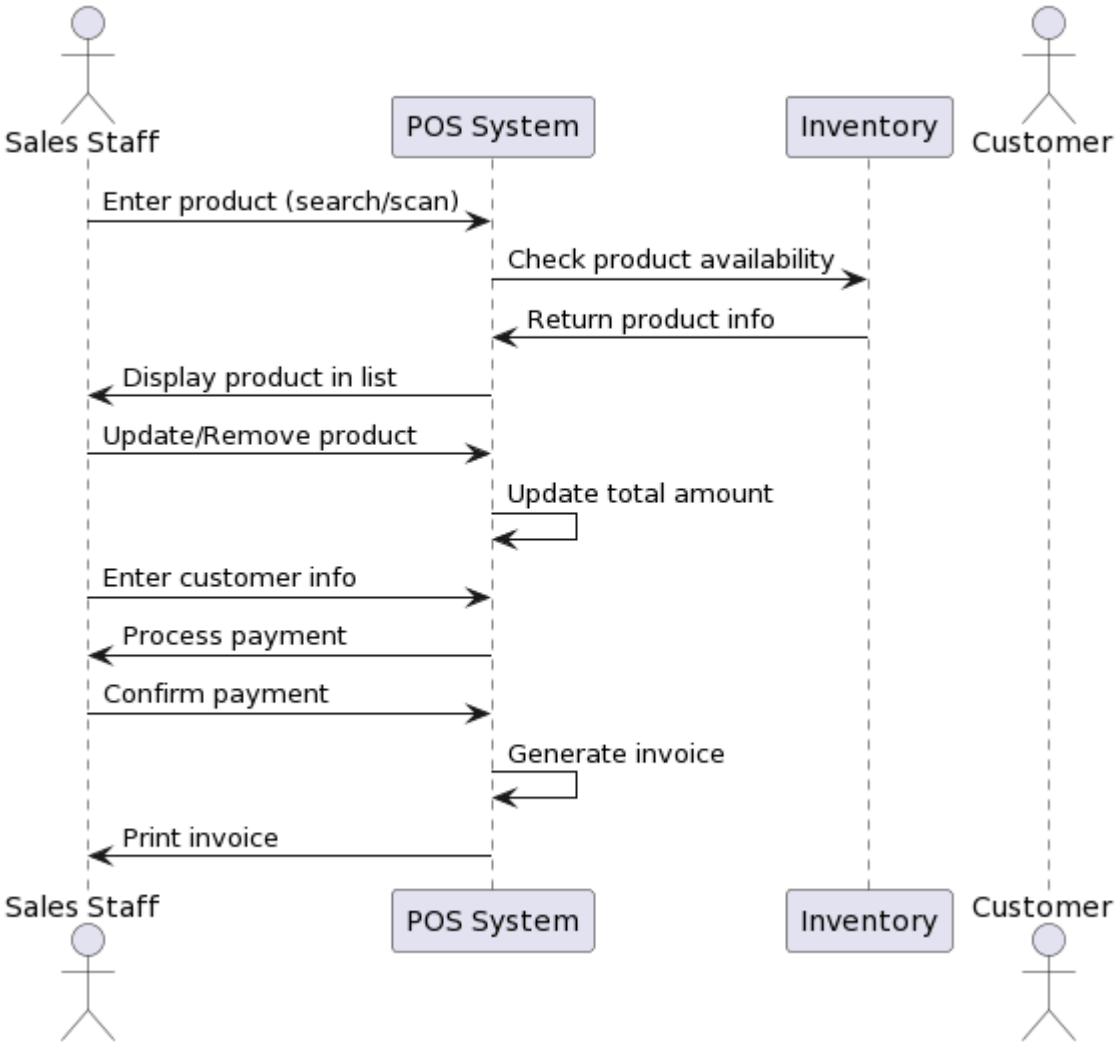
Hình 3.7 Sequence Diagram Account Management



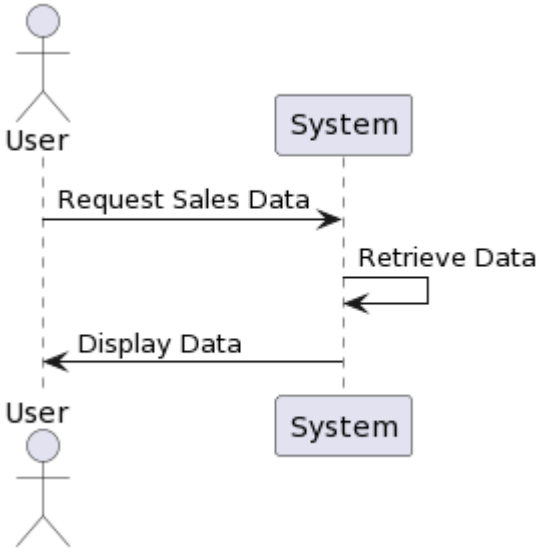
Hình 3.8 Sequence Diagram Product Catalog Management



Hình 3.9 Sequence Diagram Customer Management

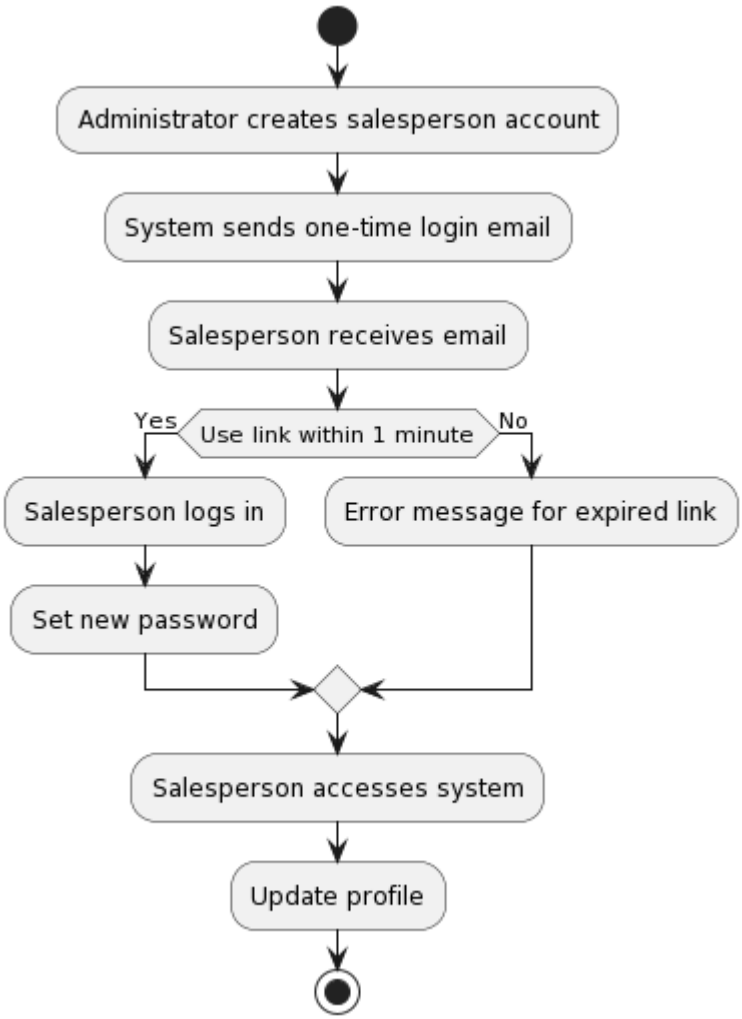


Hình 3.10 Sequence Diagram Transaction Processing

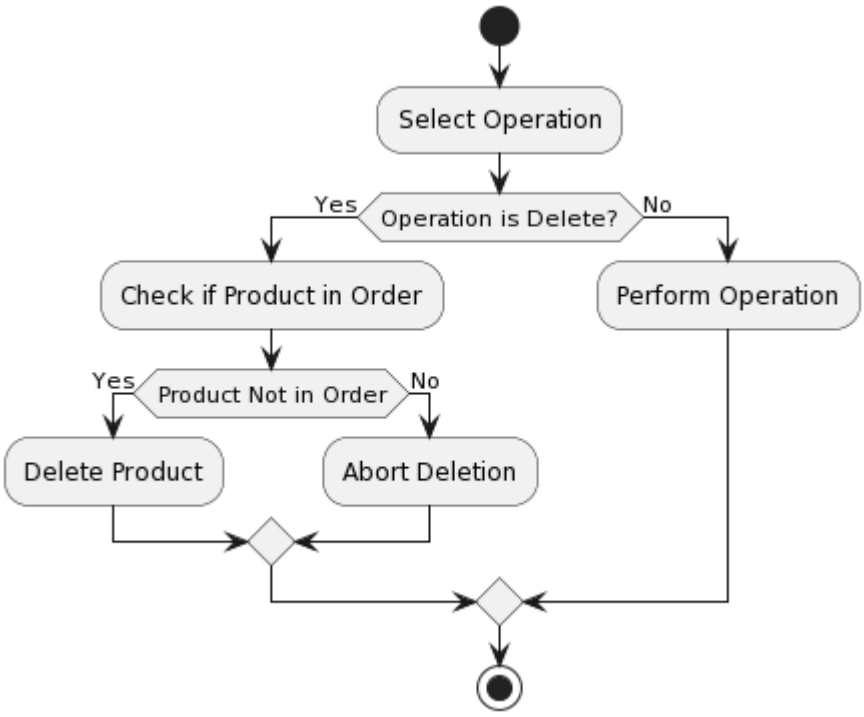


Hình 3.11 Sequence Diagram Reporting & Analytics

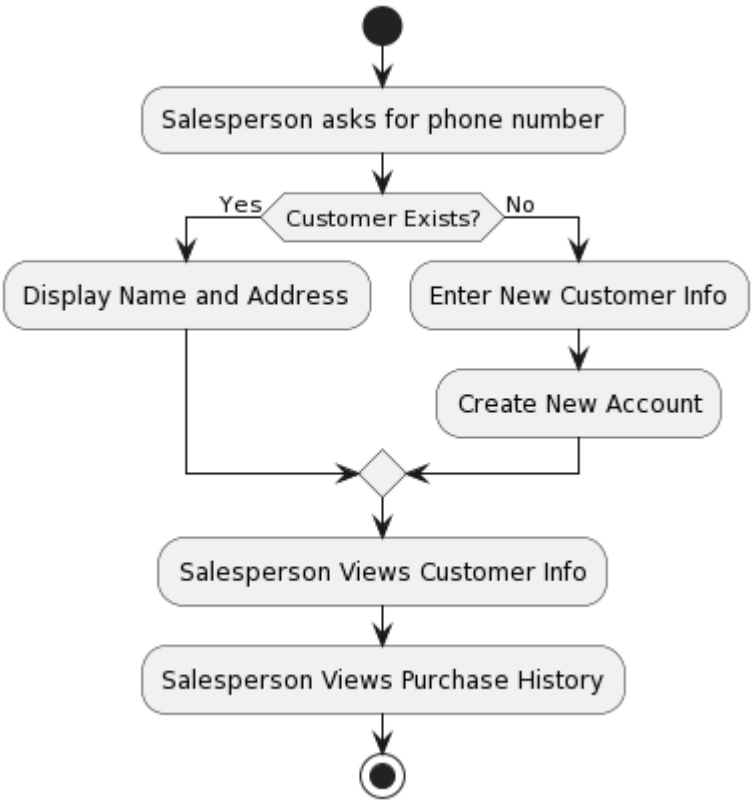
3.4 Activity Diagram



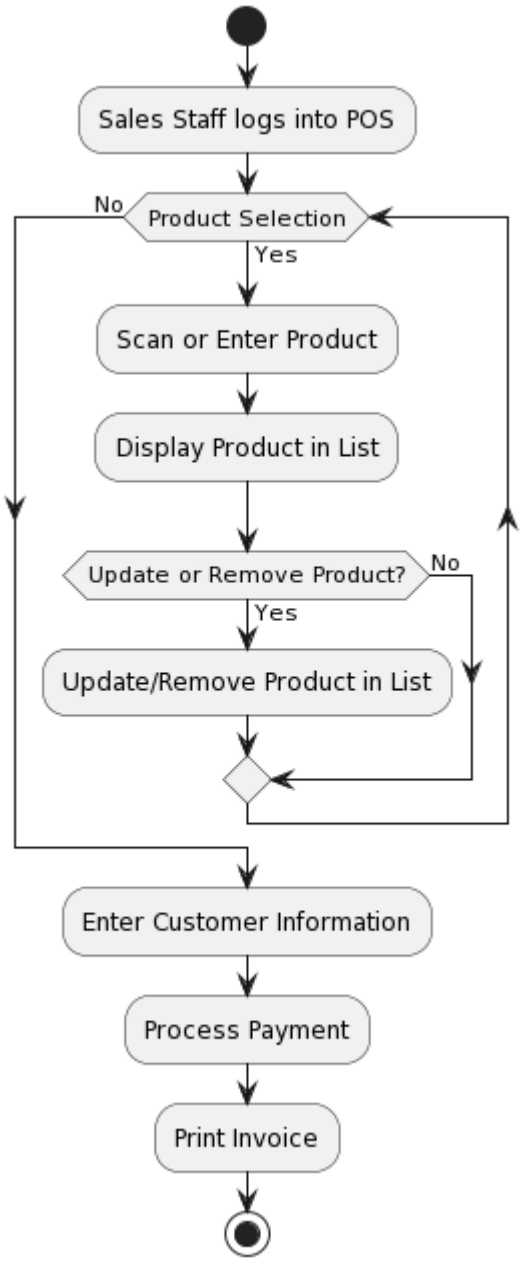
Hình 3.12 Activity Diagram Account Management



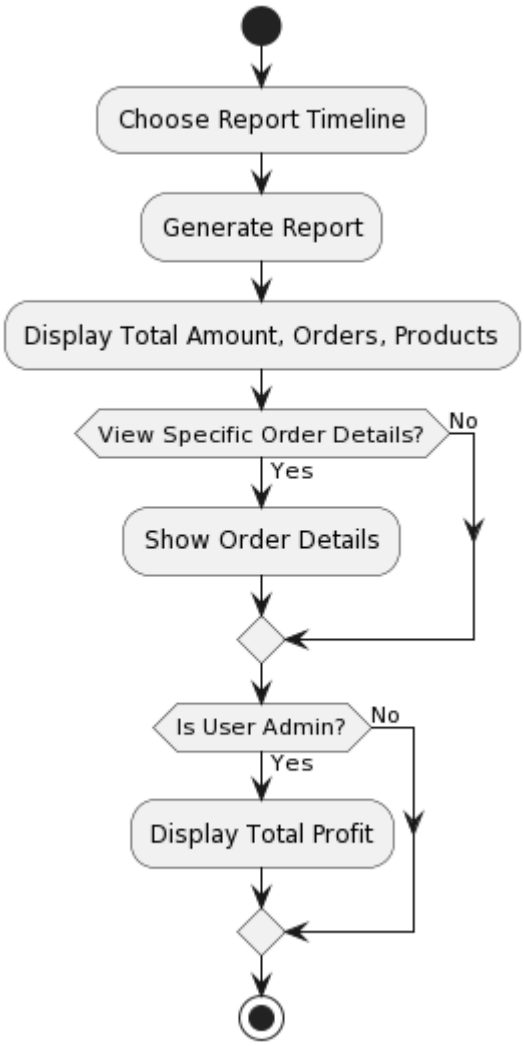
Hình 3.13 Activity Diagram Product Catalog Management



Hình 3.14 Activity Diagram Customer Management



Hình 3.15 Activity Diagram Transaction Processing



Hình 3.16 Activity Diagram Reporting & Analytics

CHƯƠNG 4. HIỆN THỰC HỆ THỐNG

4.1 Quản lý tài khoản

```
} from '../controller/userController.js'

const userRouter = express.Router();

userRouter.get('/', getUserView)

userRouter.get('/api', getListUsers);

userRouter.post('/register', userRegister);

userRouter.get('/edit/:id', getUserDetail);

userRouter.post('/update/:id', updateUser);

userRouter.delete('/delete/:id', userRemove)

userRouter.post('/block/:userId', blockUser);

userRouter.get('/profile/:id', getUserProfile);

userRouter.post('/resend-activation/:userId', resendEmail);
```

1. **Hiển Thị Thông Tin Người Dùng (getUserView, getUserProfile):** Các hàm này hiển thị các trang liên quan đến người dùng.
2. **Quản Lý Đăng Ký Người Dùng (userRegister):** Cho phép người dùng đăng ký và xử lý thông tin đăng ký.
3. **Kích Hoạt Tài Khoản (activateUser):** Xử lý kích hoạt tài khoản thông qua token.
4. **Cập Nhật và Xóa Người Dùng (updateUser, userRemove):** Cung cấp chức năng cập nhật và xóa tài khoản người dùng.
5. **Quản Lý Mật Khẩu (getSetPasswordView, setUserPassword):** Xử lý việc thiết lập và cập nhật mật khẩu cho người dùng.
6. **Gửi Lại Email Kích Hoạt (resendEmail):** Cho phép gửi lại email kích hoạt nếu người dùng chưa kích hoạt.

7. **Thông Báo cho Quản Trị Viên (notifyAdmin):** Gửi thông báo đến quản trị viên với các yêu cầu cụ thể.

```
const activateUser = async (req, res) => {
  try {
    const token = req.body.token;
    const decoded = jwt.verify(token, config.secret_key);
    const email = decoded.email;

    const result = await activateUserByEmail(email);

    if (result) {
      req.flash('success_msg', 'Activation successful.');
      return res.redirect(`/login`);
    }

    return res.redirect(`/activate/${req.body.token}`);

  } catch (error) {
    if (error instanceof jwt.TokenExpiredError) {
      // Token hết hạn, redirect đến /resend-request
      console.log("Activation link has expired. Please request a new one.")
      req.flash('error_msg', 'Activation link has expired. Please request a new one.');
      return res.redirect(`/resend-request`); // Thay đổi ở đây
    } else if (error instanceof jwt.JsonWebTokenError) {
      req.flash('error_msg', 'Invalid activation link.');
      return res.redirect(`/activate/${req.body.token}`);
    } else {
      console.error('Activation error', error);
      req.flash('error_msg', 'An error occurred.');
      return res.redirect(`/activate/${req.body.token}`);
    }
  }
};
```

Hàm **activateUser** rất quan trọng trong quá trình quản lý người dùng vì nó xử lý kích hoạt tài khoản, một bước cần thiết để đảm bảo người dùng có thể truy cập vào hệ thống sau khi đăng ký.

- Xác Thực Token: Hàm sử dụng thư viện jsonwebtoken để xác thực token được gửi qua yêu cầu POST. Nếu token hợp lệ, email được trích xuất từ token và sử dụng để kích hoạt tài khoản.

- Xử Lý Lỗi: Hàm xử lý các trường hợp lỗi như token hết hạn hoặc không hợp lệ và gửi phản hồi thích hợp.
- Kích Hoạt Tài Khoản: Nếu không có lỗi, tài khoản được kích hoạt thông qua hàm activateUserByEmail.

4.2 Quản lý danh mục sản phẩm

Quản lý danh mục sản phẩm

```
export const deleteProduct = async (req, res) => {
  try {
    const { id } = req.params
    const prod = await productService.getProductById(id);
    if (prod.isBought) {
      req.flash('msg', `Delete product failed (This product is bought).`);
      req.flash('status', 'Failed');
      return res.redirect('/product');
    }
    const result = await productService.deleteProductById(id);
    req.flash('msg', `Delete product successfully.`);
    req.flash('status', 'Success');
    return res.redirect('/product');
  } catch (err) {
    console.log("Delete Product Error: ", err);
    req.flash('msg', `Delete product failed.`);
    req.flash('status', 'Failed');
    return res.redirect('/product');
  }
}
```

Đây là hàm để xử lý request xóa sản phẩm. Trong hàm trên sẽ kiểm tra xem sản phẩm đã được mua hay chưa nếu chưa mới có thể xóa được. Còn nếu đã có lượt mua thì không thể xóa được.

```
export const create = async (req, res) => {
  const { barcode, productName, importPrice, retailPrice, category, inventory } = req.body;
  try {
    const checkExist = await Product.exists({ barcode });
    if (checkExist) {
      req.flash('msg', `Create product ${productName} failed. Duplicate Barcode.`);
      req.flash('status', 'Failed');
      return res.redirect('/product');
    }
    const result = uploadFirebase(req.file)
    if ((await result).downloadURL) {
      const product = new Product({
        barcode,
        productName,
        category,
        inventory,
        importPrice: Number.parseInt(importPrice),
        retailPrice: Number.parseInt(retailPrice),
        thumbnailUrl: (await result).downloadURL
      });
      await product.save();
      req.flash('msg', `Create product ${productName} successfully.`);
      req.flash('status', 'Success');
      return res.redirect('/product');
    } else {
      throw new Error((await res).message)
    }

  } catch (err) {
    req.flash('msg', `Create product ${productName} failed.`);
    req.flash('status', 'Failed');
    return res.redirect('/product');
  }
}
```

Đây là hàm xử lý request tạo sản phẩm. Trong hàm này ta cần kiểm tra xem barcode có được sử dụng hay chưa nếu chưa thì mới có thể tạo được. Đồng thời sẽ gửi hình ảnh của sản phẩm upload lên trên dịch vụ lưu trữ đám mây FireStorage.s

Config cho việc upload ảnh lên Fire Storage

src > controller > JS firebaseController.js > ...

You, 2 weeks ago | 2 authors (Quoc Anh Nguyen and others)

```
1  import { initializeApp } from "firebase/app";
2  import { getStorage, ref, getDownloadURL, uploadBytesResumable, deleteObject } from "firebase/storage";
3
4  import dotenv from 'dotenv';
5
6  dotenv.config();
7
8  const firebaseConfig = {
9      apiKey: process.env.FIREBASE_API_KEY,
10     authDomain: process.env.AUTH_DOMAIN,
11     projectId: process.env.PROJECT_ID,
12     storageBucket: process.env.STORAGE_BUCKET,
13     messagingSenderId: process.env.SENDER_ID,
14     appId: process.env.APP_ID
15 };
16
17 // Initialize Firebase
18 const app = initializeApp(firebaseConfig);
19
20
21 // Initialize Cloud Storage and get a reference to the service
22 const storage = getStorage(app);
```

Đây là đoạn code để kết nối với Firebase với thư viện firebase/app. Cùng với các biến môi trường đã config trước.

```
const uploadFirebase = async (file) => {
    try {
        const dateTime = giveCurrentDateTime();
        console.log(file);

        const storageRef = ref(storage, `files/${file.originalname + "_" + dateTime}`);

        // Create file metadata including the content type
        const metadata = {
            contentType: file.mimetype,
        };

        // Upload the file in the bucket storage
        const snapshot = await uploadBytesResumable(storageRef, file.buffer, metadata);
        //by using uploadBytesResumable we can control the progress of uploading like pause, resume, cancel

        // Grab the public url
        const downloadURL = await getDownloadURL(snapshot.ref);

        console.log('File successfully uploaded.');
        return {
            success: true,
            message: 'file uploaded to firebase storage',
            name: file.originalname,
            type: file.mimetype,
            downloadURL: downloadURL
        }
    } catch (error) {
        console.log("Firebase error: ", error.message);
        return { success: false, message: error.message }
    }
};
```

Hàm `uploadFirebase`: Hàm này dùng để tải lên tập tin lên Firebase Storage. Nó nhận đối số là một tập tin, tạo một tham chiếu lưu trữ với tên tập tin và thời gian hiện tại, sau đó tải lên tập tin này. Hàm này cũng trả về URL để tải xuống tập tin sau khi tải lên thành công.

```
const deleteImageFromFirebase = async (downloadURL) => {
    const fileName = refFromURL(downloadURL);
    const storageRef = ref(storage, fileName);
    try {
        await deleteObject(storageRef);
        return true;
    } catch (error) {
        return false;
    }
}
```

Hàm `deleteImageFromFirebase`: Hàm này dùng để xóa tập tin từ Firebase Storage dựa trên URL tải xuống của tập tin.

```
rootRouter.use("/user", isAuthenticated, checkUserActivation, isFirstLogined, checkUserBlocked, fetchNotifications, requireRole(['ADMIN']), userRouter);
rootRouter.use("/product", isAuthenticated, checkUserActivation, isFirstLogined, checkUserBlocked, fetchNotifications, requireRole(['ADMIN', 'SALE']), productRouter);
rootRouter.use("/customer", isAuthenticated, checkUserActivation, isFirstLogined, checkUserBlocked, fetchNotifications, requireRole(['ADMIN', 'SALE']), customerRouter);
rootRouter.use("/order", isAuthenticated, checkUserActivation, isFirstLogined, checkUserBlocked, fetchNotifications, requireRole(['ADMIN', 'SALE']), orderRouter);
rootRouter.use("/cart", isAuthenticated, checkUserActivation, isFirstLogined, checkUserBlocked, fetchNotifications, requireRole(['SALE']), cartRouter);
rootRouter.use("/notification", isAuthenticated, checkUserActivation, isFirstLogined, checkUserBlocked, fetchNotifications, requireRole(['ADMIN']), notificationRouter);
rootRouter.use("/statistic", isAuthenticated, checkUserActivation, isFirstLogined, checkUserBlocked, fetchNotifications, requireRole(['ADMIN', 'SALE']), statisticRouter);
rootRouter.use('/profile', isAuthenticated, checkUserActivation, isFirstLogined, checkUserBlocked, fetchNotifications, requireRole(['ADMIN', 'SALE']), profileRouter);

export default rootRouter;
```

Đây là phần điều hướng các request. Nơi khai báo các middleware để check xem user có thể gọi đến route đó không.

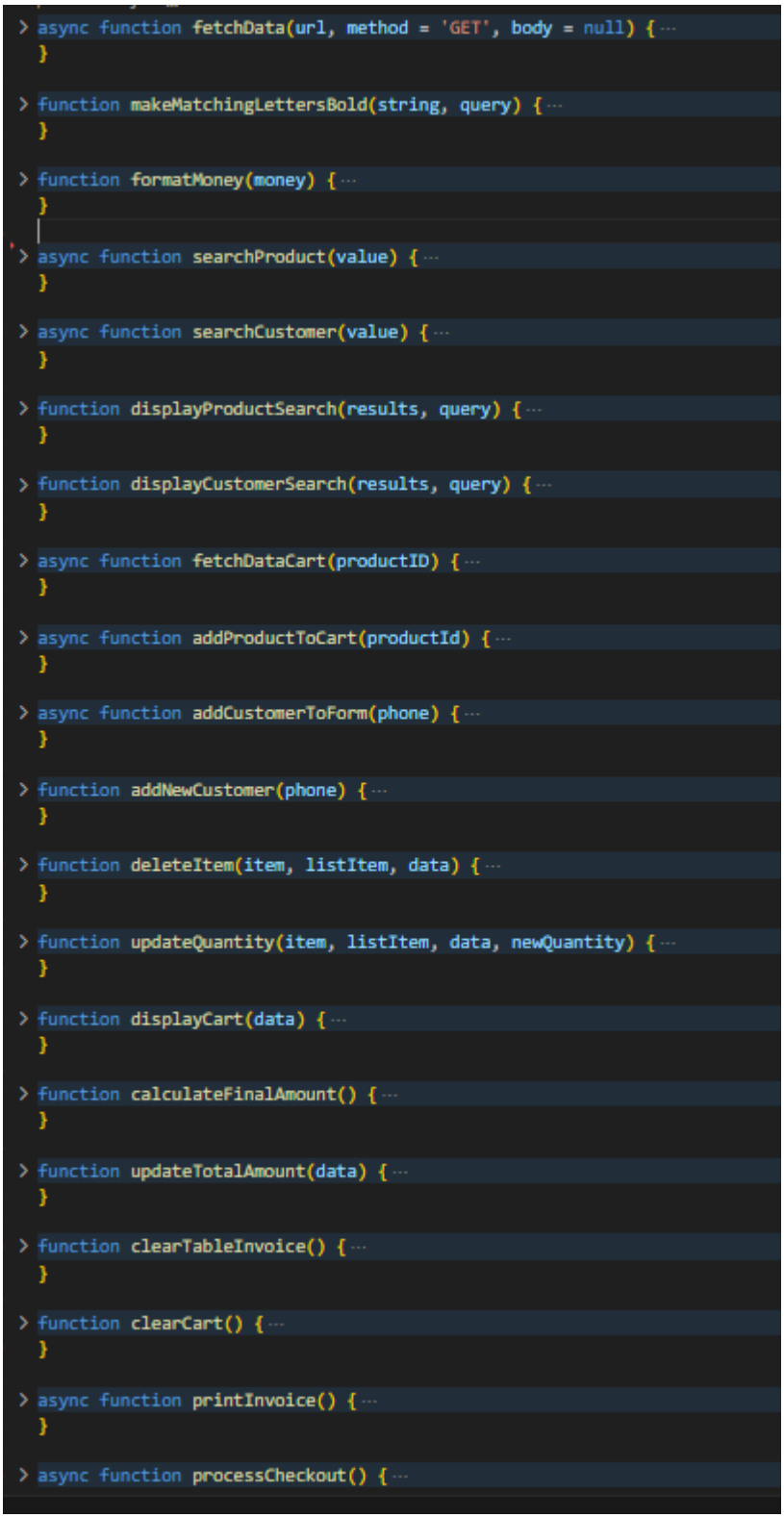
```
const calculateProfitByDate = async (startDate, endDate) => {
  const tempEndDate = new Date(endDate);
  tempEndDate.setDate(tempEndDate.getDate() + 1)
  const orders = await order.aggregate([
    {
      $match: {
        purchaseDate: { $gte: new Date(startDate), $lte: tempEndDate }
      }
    },
    {
      $group: {
        _id: { $dateToString: { format: "%Y-%m-%d", date: "$purchaseDate" } },
        totalRevenue: { $sum: "$totalAmount" },
        totalCost: { $sum: "$cost" }
      }
    },
    {
      $project: {
        date: "$_id",
        _id: 0,
        profit: { $subtract: ["$totalRevenue", "$totalCost"] }
      }
    },
    { $sort: { date: 1 } }
  ]);

  return orders.map(order => ({
    date: order.date,
    profit: order.profit
  }));
};
```

Đây là hàm để tính toán lợi nhuận theo ngày. Bằng cách group các ngày lại với nhau và tính tổng các chi phí cũng như tổng tiền của hóa đơn.

4.3 Quản lý khách hàng

Tương tự quản lý tài khoản

4.4 Xử lý giao dịch


async function fetchData(url, method = 'GET', body = null) {
}
function makeMatchingLettersBold(string, query) {
}
function formatMoney(money) {
}
async function searchProduct(value) {
}
async function searchCustomer(value) {
}
function displayProductSearch(results, query) {
}
function displayCustomerSearch(results, query) {
}
async function fetchDataCart(productID) {
}
async function addProductToCart(productId) {
}
async function addCustomerToForm(phone) {
}
function addNewCustomer(phone) {
}
function deleteItem(item, listItem, data) {
}
function updateQuantity(item, listItem, data, newQuantity) {
}
function displayCart(data) {
}
function calculateFinalAmount() {
}
function updateTotalAmount(data) {
}
function clearTableInvoice() {
}
function clearCart() {
}
async function printInvoice() {
}
async function processCheckout() {

1.Người Nghe Sự Kiện và Khởi Tạo:

- window.onload khởi tạo hiển thị giỏ hàng và thiết lập các giá trị mặc định.
- Các bộ nghe sự kiện cho ô nhập tìm kiếm sản phẩm và khách hàng, số tiền nhận được, và nút thanh toán.
- document.addEventListener('DOMContentLoaded', ...) xử lý chức năng đăng xuất.

2.Chức Năng Tìm Kiếm (searchProduct và searchCustomer):

- Các hàm này thực hiện tìm kiếm sản phẩm và khách hàng sử dụng hàm fetchData.
- displayProductSearch và displayCustomerSearch được gọi để hiển thị kết quả tìm kiếm.

3. Lấy Dữ Liệu (fetchData function):

- Hàm tổng quát để thực hiện các yêu cầu fetch. Nó xử lý các yêu cầu GET và POST và trả về dữ liệu JSON.
- Sử dụng API fetch với xử lý lỗi.

4. Cập Nhật Nội Dung Động (displayProductSearch, displayCustomerSearch, displayCart):

- Các hàm này cập nhật nội dung HTML dựa trên dữ liệu được fetch.
- Chúng thao tác DOM để hiển thị sản phẩm, khách hàng và các mục trong giỏ hàng.

5. Quản Lý Giỏ Hàng (addProductToCart, addCustomerToForm, deleteItem, updateQuantity):

- Các hàm để thêm sản phẩm vào giỏ hàng, thêm thông tin khách hàng, xóa mục khỏi giỏ hàng và cập nhật số lượng sản phẩm.

6. Thanh Toán và Tạo Hóa Đơn (processCheckout và printInvoice):

- processCheckout xử lý quá trình thanh toán, bao gồm các xác nhận và in hóa đơn.
- printInvoice tạo hóa đơn PDF sử dụng jsPDF và html2canvas.

7. Hàm Tiện Ích (makeMatchingLettersBold, formatMoney):

- makeMatchingLettersBold làm nổi bật các chữ cái khớp trong một chuỗi.
- formatMoney định dạng một số thành tiền tệ.

Giải Thích Chi Tiết về Phần Quan Trọng - Hàm processCheckout:

```
if (newCustomer) {
    const dataCustomer = {
        phoneNumber: phoneNumber.value,
        fullName: fullName.value,
        address: address.value,
    }
    console.log(dataCustomer)
    const response = await fetchData('customer/add', 'POST', dataCustomer);
    newCustomer = false;
}

if (fullName.value && address.value && amountReceived.value && products.length != 0)
    await printInvoice();
    //Remove productName
    const productsToCheckout = products.map(({ productName, ...rest }) => rest);
    const data = {
        customerId: customerId_,
        products: productsToCheckout,
        totalAmount: parseFloat(totalAmountEl.textContent.replace(/[$,]/g, "")),
        discount: parseFloat(discountInput.value),
        finalAmount: parseFloat(finalAmountEl.textContent.replace(/[$,]/g, "")),
        moneyReceived: parseFloat(amountReceived.value),
        moneyBack: parseFloat(changeAmount.textContent.replace(/[$,]/g, ""))
    };

    try {
        const responseData = await fetchData('order/add', 'POST', data);
        console.log('Checkout successful:', responseData);
        clearCart();
    } catch (error) {
        console.error('Checkout error:', error);
    }
}
```

Hình 5.1 Hàm processCheckout

Kiểm Tra Xác Nhận:

- Đảm bảo rằng giỏ hàng không trống, thông tin khách hàng đã được cung cấp và số tiền nhận đủ.
- Hiển thị thông báo lỗi bằng toastr nếu có lỗi xác nhận.

Xử Lý Khách Hàng Mới:

- Nếu thanh toán cho khách hàng mới (newCustomer cờ), nó gửi dữ liệu khách hàng (dataCustomer) đến máy chủ để tạo một bản ghi khách hàng mới.

In Hóa Đơn:

- Gọi printInvoice để tạo và tải xuống hóa đơn PDF.

Chuẩn Bị Dữ Liệu cho Thanh Toán:

- Chuẩn bị dữ liệu cho quá trình thanh toán, bao gồm ID khách hàng, sản phẩm, tổng số tiền, giảm giá, số tiền cuối cùng, tiền nhận và tiền trả lại.

Yêu Cầu Thanh Toán:

- Gửi dữ liệu thanh toán đến máy chủ sử dụng hàm fetchData với yêu cầu POST đến điểm cuối 'order/add'.
- Xóa sạch giỏ hàng sau khi thanh toán thành công và ghi lại phản hồi hoặc lỗi.

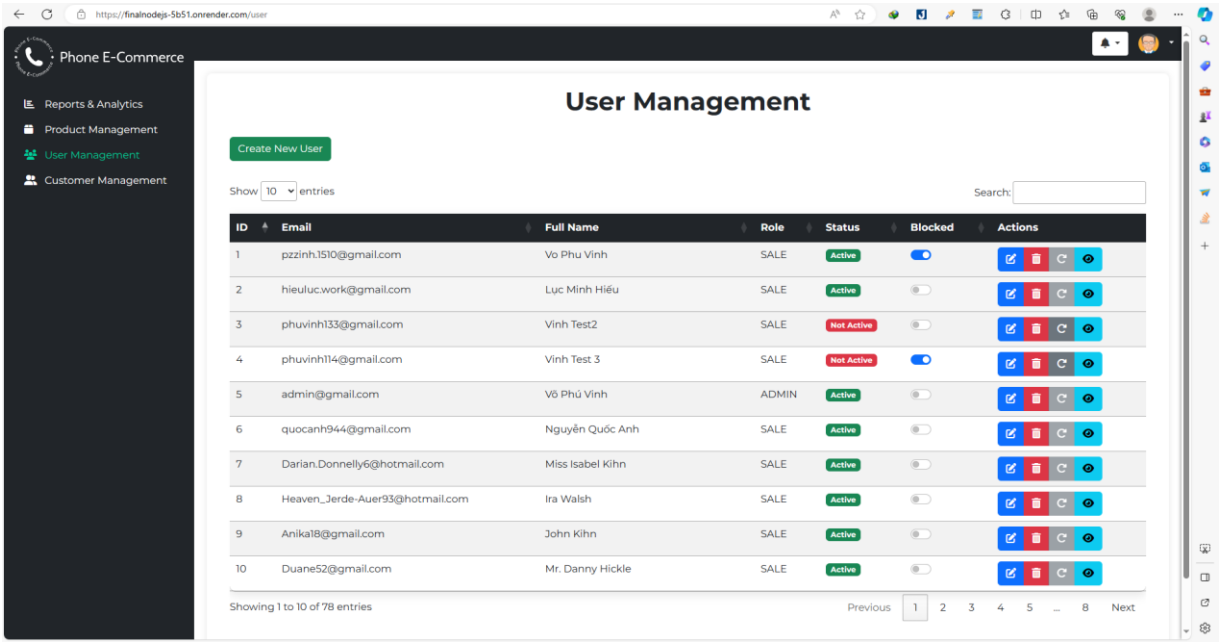
4.5 Báo cáo và phân tích

Sử dụng thư viện ApexCharts, là một thư viện đồ thị JavaScript hiện đại và linh hoạt, thường được sử dụng để tạo ra các biểu đồ đẹp mắt trên các trang web. Khi sử dụng Node.js cho phía máy chủ, ApexCharts thường được tích hợp vào ứng dụng web để hiển thị dữ liệu theo cách trực quan.

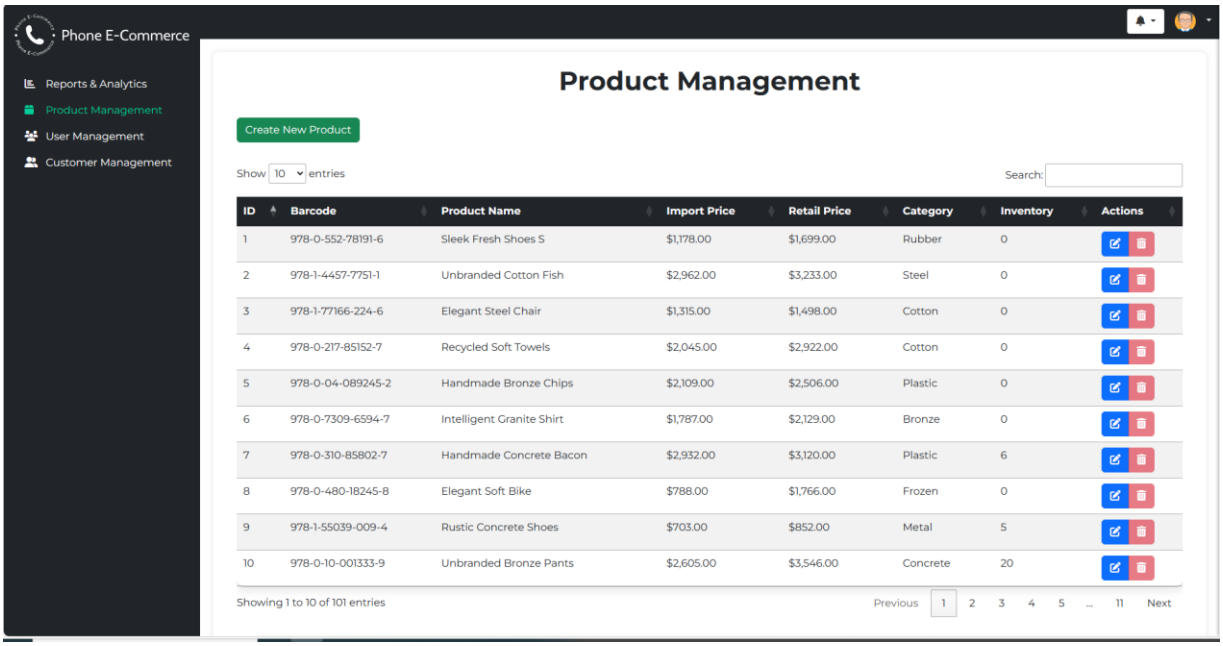
```
async function loadSalesData(start, end) {
  try {
    const response = await axios.get('/statistic/sales-data', {
      params: {
        startDate: start.format('YYYY-MM-DD'),
        endDate: end.format('YYYY-MM-DD')
      }
    });
    console.log(response)
    const data = response.data;
    renderChart(data);
  } catch (error) {
    console.error('Error loading sales data:', error);
  }
}

function loadDashboardData() {
  axios.get('/statistic/dashboard')
    .then(function (response) {
      console.log(response)
      const data = response.data.data;

      if (data) {
        renderDashboardData(data);
      }
    })
    .catch(function (error) {
      console.error('Error loading dashboard data:', error);
    });
}
```

CHƯƠNG 5. KẾT QUẢ ĐẠT ĐƯỢC

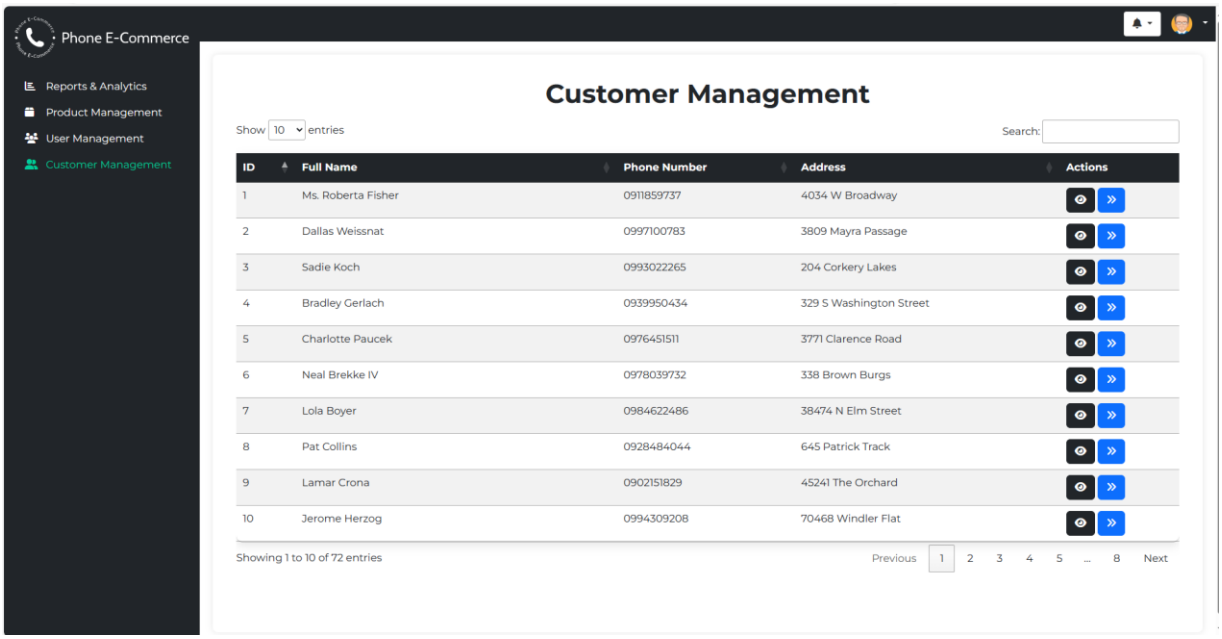
5.1 Quản lý tài khoản



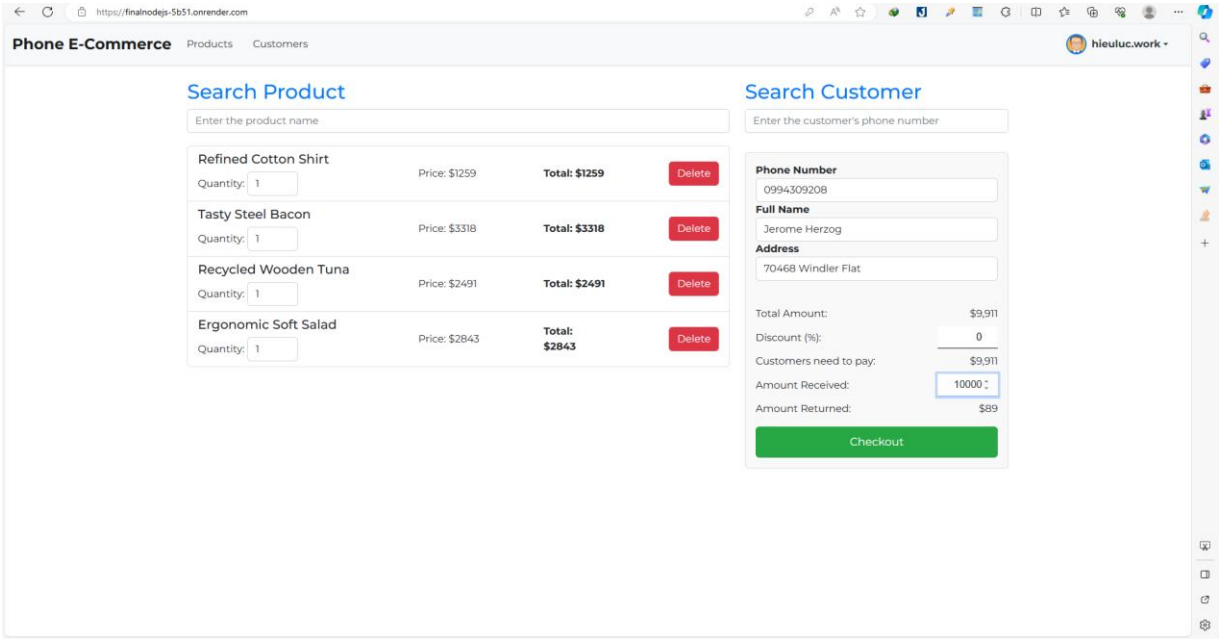
5.2 Quản lý danh mục sản phẩm



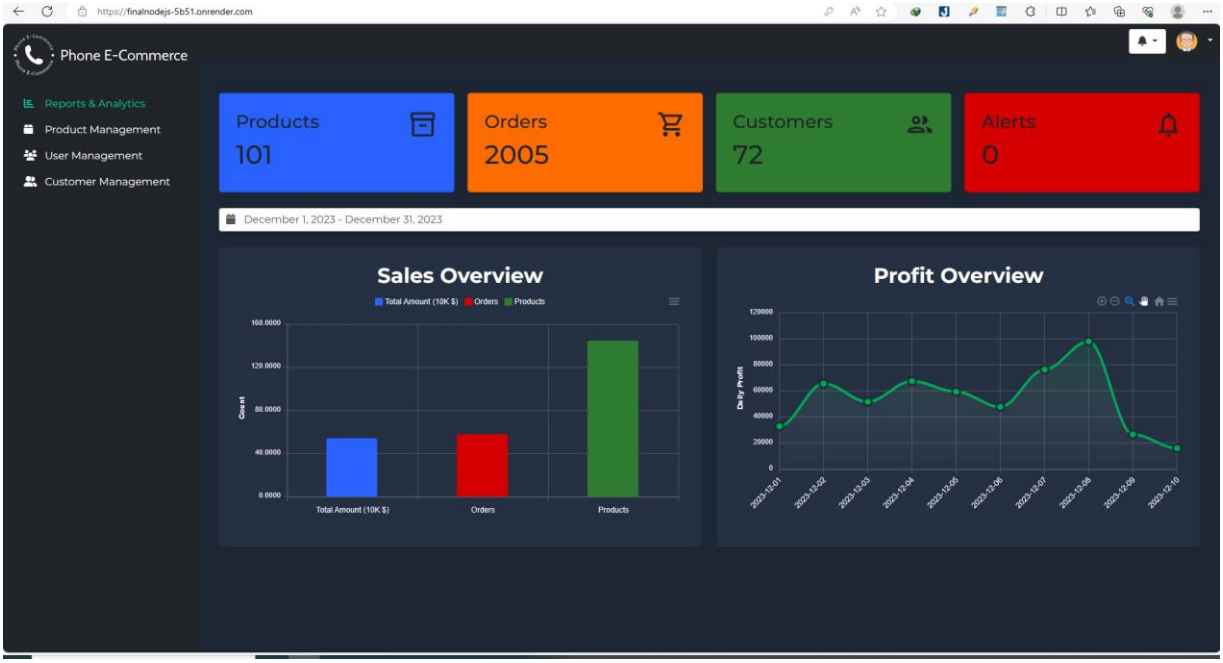
5.3 Quản lý khách hàng



5.4 Xử lý giao dịch



5.5 Báo cáo và phân tích



CHƯƠNG 6. TỔNG KẾT

6.1 Ưu điểm và khuyết điểm của hệ thống

6.1.1 Ưu điểm

- Giao diện thân thiện
- Có thể lưu ảnh trên firebase
- Có chức năng chống DDOS
- Đầy đủ các tính năng cần thiết cho một hệ thống bán hàng
- Có hệ thống báo cáo và thống kê trực quan, dễ sử dụng
- Có nguồn dữ liệu mẫu cho môi trường testing
- Dễ dàng bảo trì và nâng cấp sau này
- Có thể phát triển thêm ứng dụng cho di động dựa trên nền tảng đã có sẵn.

6.1.2 Khuyết điểm

- Dữ liệu mẫu tuy đầy đủ nhưng chưa đúng chủ đề
- Tổ chức code chưa đồng bộ
- Giới hạn dung lượng lưu trữ do mongo và firebase đang sử dụng bản miễn phí

- Giới hạn về mặt tốc độ xử lý request do đang sử dụng mongo và firebase

6.2 Hướng phát triển trong tương lai

- Phát triển thêm ứng dụng cho mobile
- Mở rộng thêm nhiều tính năng hơn như giao hàng, thanh toán online,…
- Chỉnh sửa lại giao diện hoàn hảo hơn, bố trí hợp lí các tính năng hơn cho người dung dễ thao tác
- Tối ưu hóa tốc độ của hệ thống
- Tăng tính bảo mật cho người dùng

TÀI LIỆU THAM KHẢO

1. Lời khuyên trước khóa học Node Express | Học lập trình cơ bản | Tự học lập trình online (youtube.com)
2. https://viblo.asia/p/mot-so-luu-y-de-de-ung-dung-nodejs-an-toan-hon-3P0lPnLoKox
3. https://getbootstrap.com/docs/5.0/components/navbar/
4. https://stackoverflow.com/questions/46155/how-can-i-validate-an-email-address-in-javascript
5. https://apexcharts.com/
6. https://expressjs.com/
7. https://render.com/docs/deploy-node-express-app