VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY

WEB APPLICATION DEVELOPMENT PROJECT
E-COMMERCE PLATFORM

By
Nguyễn Hà Thanh - Student ID: ITITIU21144

Advisor: Dr. Nguyen Van Sinh

A report submitted to the School of Computer Science and Engineering in partial fulfillment of the requirements for the Final Project in Web Application Development course - December 2025

Ho Chi Minh city, Vietnam, 2025

TABLE OF CONTENT

1. INTRODUCTION....4

1.1. About the Project....4

1.2. Project Objectives....4

1.3. Development Environment and Tools....5

2. REQUIREMENT ANALYSIS AND DESIGN....5

2.1. Requirement Analysis....5

2.1.1. Overall System Description....5

2.1.2. Functional Requirements....6

2.1.3. Non-Functional Requirements....7

2.1.4. Use Case Diagram....8

2.1.5. Use Case Specifications....8

2.2. System Design....11

2.2.1. Overall System Architecture....11

2.2.2. Component Diagram....13

2.2.3. Entity–Relationship Diagram....13

3. IMPLEMENTATION....14

3.1. User Interface Implementation....14

3.1.1. Login/Register/Logout....14

3.1.2. Homepage and Product Listing....15

3.1.3. Product Detail Page....15

3.1.4. Shopping Cart Page....16

3.1.5. English/Vietnamese UI....17

3.1.6. Shop-lists....18

3.1.7. Account Page/Register Shop....18

3.1.8. Checkout Page....19

3.1.9. Invoice History Page....21

3.1.10. Shop Management Page....22

3.1.11. admin dashboard....25

3.2. Functional Implementation....28

3.2.1. Product Data Handling....28

3.2.2. Cart Logic and State Update..... 28
3.2.3. Invoice Generation Logic..... 29
3.2.4. Data Formatting and Time Handling..... 29
3.3. Error Handling and Edge Cases..... 30
3.3.1. Empty Cart Handling..... 30
3.3.2. Error Page..... 31
3.3.3. Invalid Product Data..... 32
4. DEPLOYMENT AND SYSTEM CONFIGURATION..... 32
4.1. Deployment Architecture Overview..... 32
4.2. Database Deployment with Supabase..... 33
4.3. Application Deployment using Vercel..... 34
4.4. Custom Domain Configuration with Cloudflare..... 34
4.5. Deployment Limitations and Considerations..... 35
5. DISCUSSION AND CONCLUSION..... 35
5.1. Achievements..... 35
5.2. Lessons Learned..... 36
5.3. Future Improvements..... 36
6. REFERENCES..... 37

1. INTRODUCTION

1.1. About the Project

In recent years, web-based applications have become an essential component of modern business operations, especially in the field of e-commerce. With the rapid growth of online shopping platforms, users increasingly expect fast, intuitive, and reliable web applications that can assist them in browsing products, managing orders, and completing transactions efficiently. As a result, the development of well-structured and user-friendly web applications has become a critical skill for software engineering students.

This project, named Shople, is developed as a final project for the Web Application Development course. Shople is a web-based shopping application that allows users to browse products, manage a shopping cart, perform checkout operations, and store invoices for later reference. The system demonstrates the integration of frontend technologies with a cloud-based backend service, providing a practical example of a modern web application architecture.

1.2. Project Objectives

The scope of this project includes the development of both frontend and backend-related components through cloud services. The application supports essential e-commerce functionalities such as product listing, shopping cart management, checkout processing, and invoice storage.

For data persistence, the project uses a PostgreSQL database hosted on Supabase, which serves as the backend database solution. This allows the system to store and retrieve product information and transaction data reliably without requiring manual database server management.

The application is deployed using Vercel, which is directly connected to the project's GitHub repository for continuous deployment. Additionally, a custom domain name, hathanhhome.io.vn, is configured for the deployed application through Cloudflare, providing domain management and DNS services.

Advanced features such as online payment gateway integration and complex role-based access control are considered outside the scope of this project but can be added in future enhancements.

1.3. Development Environment and Tools

Shople is developed and deployed using modern web development technologies and cloud platforms. The main tools and technologies used in this project include:

- **Next.js** for application structure, routing, and server-side rendering support.
- **React** for building reusable and interactive user interface components.
- **TypeScript** for type safety and improved code maintainability.
- **PostgreSQL (Supabase)** as the cloud-hosted relational database system.
- **Supabase** for backend services and database management.
- **GitHub** for source code management and version control.
- **Vercel** for application deployment and continuous integration.
- **Cloudflare** for custom domain configuration and DNS management.
- **HTML and CSS** for page structure and styling.
- **Visual Studio Code** as the primary development environment.

These technologies collectively enable the development of a scalable, maintainable, and production-ready web application.

2. REQUIREMENT ANALYSIS AND DESIGN

2.1. Requirement Analysis

2.1.1. Overall System Description

The system is a multi-role e-commerce web application that enables product discovery, authenticated cart management, a streamlined checkout flow that persists invoices, and a shop verification workflow governed by administrative oversight. Authentication relies on JWT tokens stored in HTTP-only cookies, with role-based authorization distinguishing guests, customers, shop owners,

and administrators. Core business data—users, shops, products, carts, invoices, verification requests, notifications, and supporting product images—is stored relationally in Supabase with referential and validation constraints. Customers can browse and search products, manage carts, and execute checkout, which generates an order reference for presentation and persists an invoice while clearing the cart upon success. Shop owners register via a dedicated flow, manage products for their own shops, and submit verification requests; administrators review and approve/reject these requests (verifying shops), can create shops/products, and may update order statuses irrespective of ownership. The platform enforces basic rate limiting on sensitive endpoints, uses hashed passwords for credential storage, and applies short-lived caching for product listings to balance responsiveness and load.

2.1.2. Functional Requirements

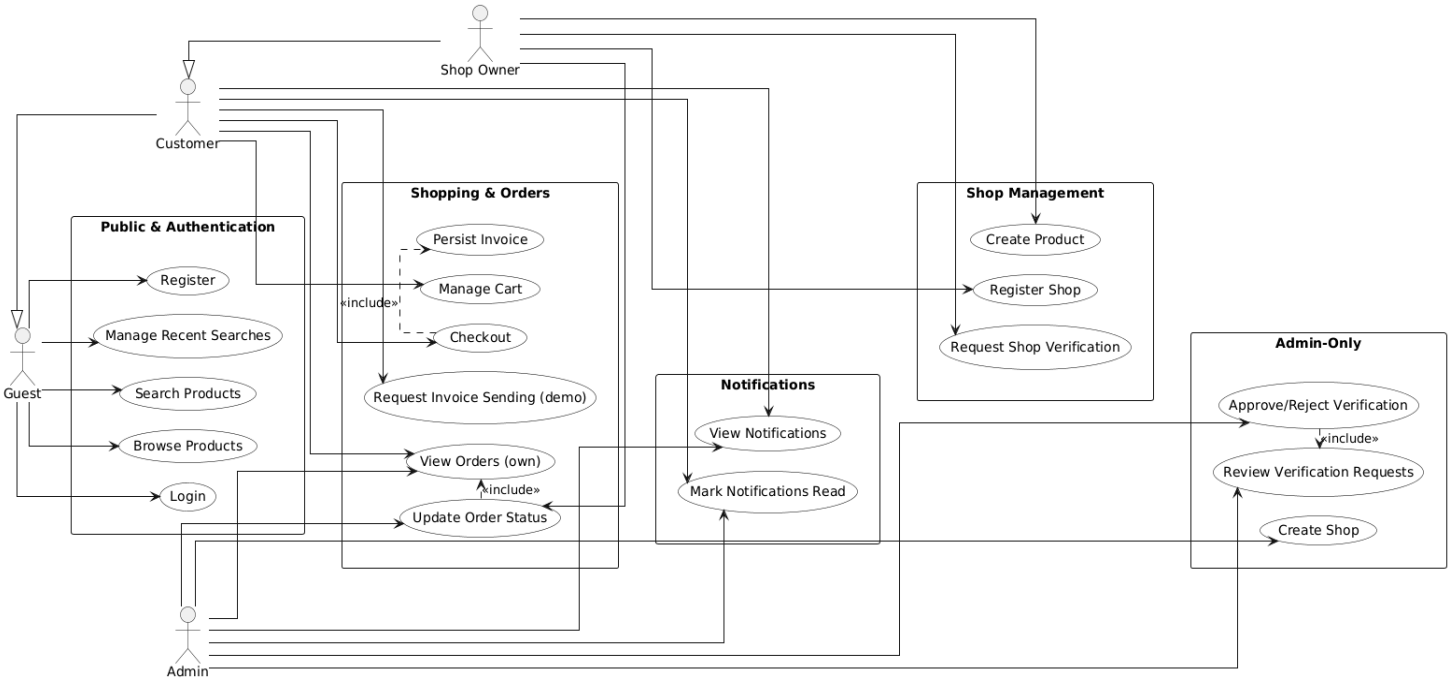
- **Authentication & Session:** Register/login with validation and bcrypt hashing; issue JWT in HTTP-only cookies; rate-limit high-risk auth requests.
- **Product Discovery:** List products with pagination; search by title keyword; optionally return total count; short-lived caching for listings.
- **Cart Management (Authenticated):** Auto-create cart if missing; add items (increment if exists); update quantity; remove items; rate-limit cart mutations.
- **Checkout & Invoice (Implemented Outcome):** Compute totals from cart; generate order reference; upsert invoice (order reference as key) with items and contact data; clear cart only after successful invoice persistence; support “send invoice” request (demo, no real email).
- **Shop Registration & Verification:** Shop-owner registration creates SHOP user and unverified shop; shop owner can submit verification requests; admin can review and approve/reject, marking shop verified on approval.
- **Product Management:** Shop owners can create products only for their own shops; admins can create products for any shop; support multiple images per product.

- **Orders (Partial):** Authenticated users can read their own orders (if present); admins can update any order status; shop owners can update order status for orders of their own shop.
- **Notifications:** Retrieve recent notifications; retrieve unread count; mark all as read; system issues notifications on verification decisions.
- **Recent Searches:** Store a small, per-user/guest list of recent search queries in cookies.

2.1.3. Non-Functional Requirements

- **Security:** Bcrypt password hashing; JWT in HTTP-only cookies; role/ownership authorization on protected APIs; rate limiting on authentication and cart mutations.
- **Data Integrity:** Foreign keys and unique/check constraints for roles, statuses, prices, and quantities; automatic updated-at triggers; idempotent invoice upsert by order reference.
- **Performance:** Paginated product queries with bounded page size; short-lived caching headers for product listings; indexed columns for common filters (users, shops, products, carts, orders, requests, notifications).
- **Reliability & Fault Handling:** Structured error responses; cart clearing only after successful invoice persistence; graceful fallbacks when auxiliary writes (e.g., cart creation, notifications) fail.
- **Usability:** Clear success/error messaging in critical flows (auth, cart, checkout, verification); bilingual UI strings (VI/EN) in key pages.
- **Privacy & Session Safety:** HTTP-only, same-site cookies; secure flag in production; minimal exposure of sensitive fields in responses.

2.1.4. Use Case Diagram



2.1.5. Use Case Specifications

UC1: Register (Customer)

- **Preconditions:** Email unused; guest state.
- **Main flow:** Submit name/email/password → validate format/strength → hash password → create USER and cart → issue JWT cookie → redirect.
- **Exceptions:** Invalid input; email exists; rate limit reached; persistence error.

UC2: Login

- **Preconditions:** Account exists; guest state.
- **Main flow:** Submit email/password → rate-limit check → verify credentials → issue JWT cookie → redirect.
- **Exceptions:** Invalid credentials; rate limit; system error.

UC3: Browse/Search Products

- **Preconditions:** None (public).
- **Main flow:** Request list with optional keyword + pagination → return items (and total if requested) with short-lived caching.
- **Exceptions:** Data access error.

UC4: Manage Cart (Add/Update/Delete)

- **Preconditions:** Authenticated user.
- **Main flow:** Auto-create cart if missing → add (increment), update quantity, or remove item with validation and rate limiting.
- **Exceptions:** Unauthenticated; invalid product/quantity; rate limit; data error.

UC5: Checkout & Persist Invoice

- **Preconditions:** Authenticated; cart has items.
- **Main flow:** Compute totals and order reference → upsert invoice (items + contact/shipping) → clear cart on success → show success view.
- **Exceptions:** Empty cart; invoice persistence error (cart not cleared).

UC6: Request Invoice Sending (demo)

- **Preconditions:** Authenticated; invoice reference available.
- **Main flow:** Submit email/order data → acknowledge request (demo, no real email).
- **Exceptions:** Missing required fields; system error.

UC7: Register Shop (Merchant Signup)

- **Preconditions:** Email unused.
- **Main flow:** Submit shop name/email/password → validate → create SHOP user and unverified shop → auto-create verification request (best-effort) → issue JWT cookie → redirect.
- **Exceptions:** Invalid input; email exists; persistence error.

UC8: Request Shop Verification

- **Preconditions:** Authenticated SHOP; owns a shop.
- **Main flow:** Submit verification request → save as PENDING.
- **Exceptions:** No owned shop; persistence error.

UC9: Approve/Reject Verification (Admin)

- **Preconditions:** Authenticated ADMIN; request is PENDING.

- **Main flow (Approve):** Mark shop verified → update request to APPROVED → notify requester.
- **Main flow (Reject):** Update request to REJECTED → notify requester.
- **Exceptions:** Invalid request state; shop/owner not found; persistence error.

UC10: Create Product

- **Preconditions:** Authenticated; ADMIN or SHOP owner of target shop.
- **Main flow:** Submit title/price/[description]/[images] → authorize ownership/role → create product (and images if provided).
- **Exceptions:** Unauthorized/forbidden; invalid payload; persistence error.

UC11: View Orders

- **Preconditions:** Authenticated.
- **Main flow:** USER views own orders; ADMIN views any; SHOP can view orders of their shop (if present).
- **Exceptions:** Unauthorized/forbidden; data error.

UC12: Update Order Status

- **Preconditions:** Authenticated; ADMIN or SHOP owner of the order's shop.
- **Main flow:** Submit new status → authorize → update status.
- **Exceptions:** Forbidden by role/ownership; invalid status; data error.

UC13: Notifications (View/Mark Read)

- **Preconditions:** Authenticated.
- **Main flow:** Fetch recent notifications; fetch unread count; mark all read.
- **Exceptions:** Unauthenticated (empty/unauthorized response); data error.

UC14: Recent Searches

- **Preconditions:** None.
- **Main flow:** Store/retrieve up to a small list of recent queries in cookies (separate for guest vs authenticated user).
- **Exceptions:** Bad request payload.

2.2. System Design

2.2.1. Overall System Architecture

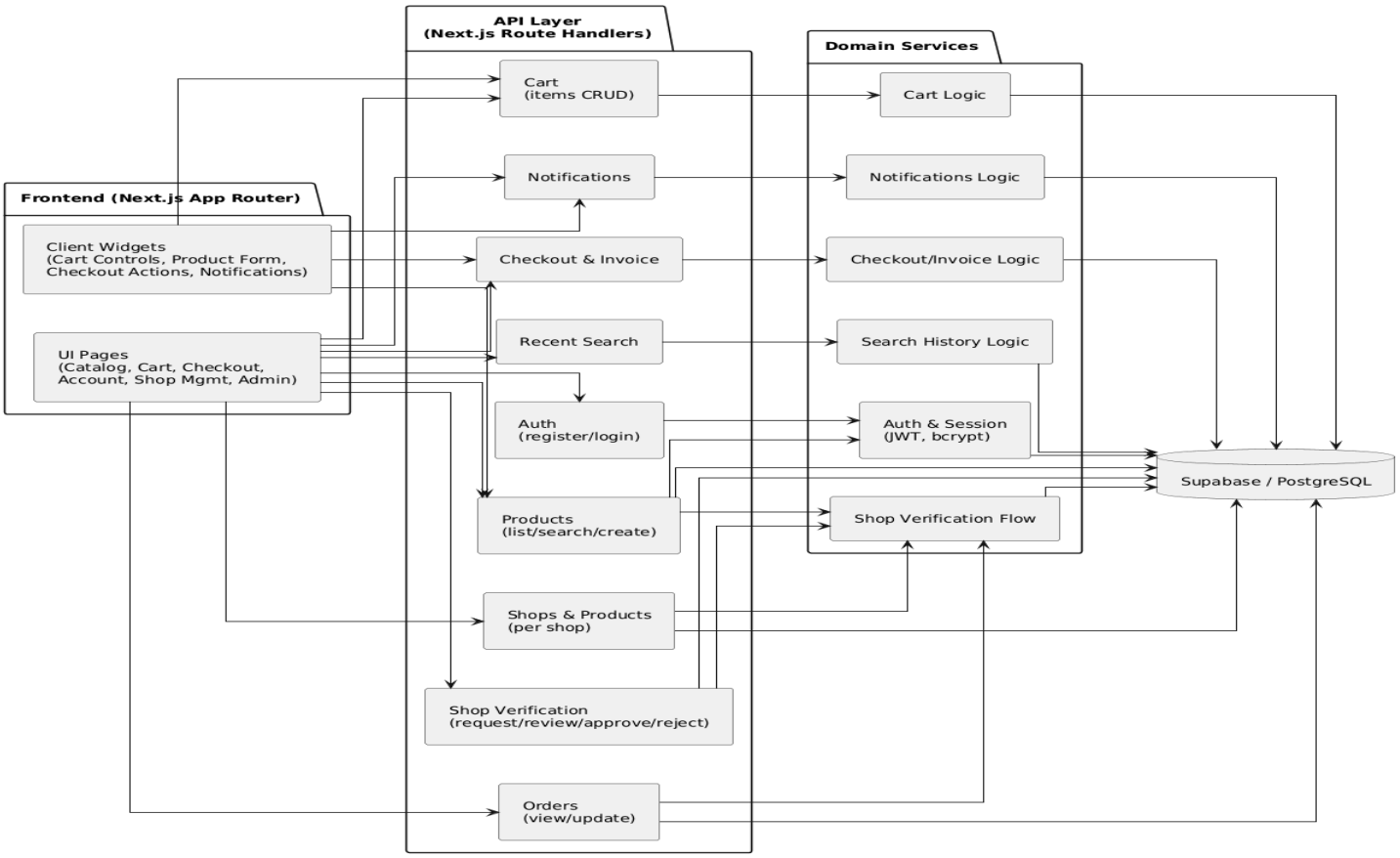
The application follows a modular, service-backed web architecture built on Next.js (App Router) with server-side rendered pages and co-located API routes. It uses Supabase as the backend data layer for relational storage, enforcing referential integrity and role/status constraints. Authentication is handled via JWTs stored in HTTP-only cookies; role-based authorization is enforced in API handlers for USER, SHOP, and ADMIN scopes.

Key layers and components

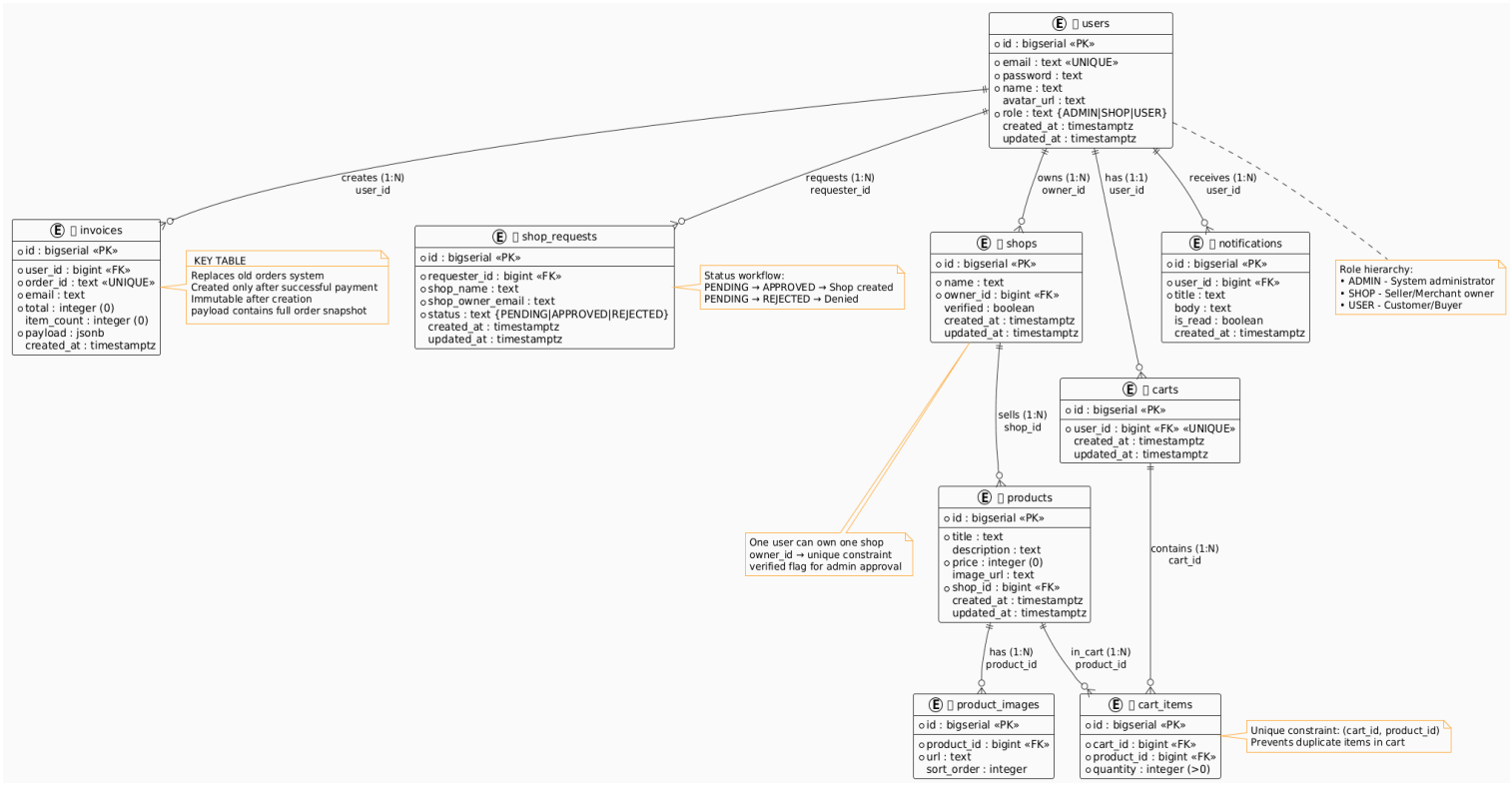
- **Presentation layer (Next.js App Router):** Server-rendered and client-enhanced pages for browsing, cart, checkout, account, shop management, and admin console. Client components handle interactive tasks (e.g., product creation form, cart updates, checkout actions).
- **API layer (Next.js route handlers):** Co-located API endpoints for auth, products, shops, cart, checkout/invoice, orders, notifications, search history, and shop verification workflows. These endpoints perform validation, authorization, rate limiting on sensitive paths, and data access via Supabase.
- **Domain/services:**
    - **Auth & Session:** JWT issuance/verification, cookie configuration, password hashing.
    - **Catalog:** Product listing/search with pagination and short-lived caching; product creation (admin or shop owner with ownership check).
    - **Cart:** User-scoped cart creation and item mutations with rate limiting.
    - **Checkout/Invoice:** Computes totals from cart, generates an order reference, persists invoices idempotently, and clears the cart on success.
    - **Shop & Verification:** Shop owner registration creates an unverified shop; verification requests flow to admin for approve/reject, with notifications.

- **Orders:** Read-only for users' own orders; status updates permitted to admins or shop owners for their shop's orders.
- **Notifications:** Fetch list/count and mark-all-read; system emits notifications on verification decisions.
- **Recent Search:** Cookie-based, per-user/guest storage of recent queries.
- **Data layer (Supabase/PostgreSQL):** Tables for users, shops, products, product_images, carts/cart_items, orders/order_items, payments (placeholder), invoices, shop_requests, and notifications. Constraints (FK, UNIQUE, CHECK) and triggers maintain integrity and timestamps; indexes support common filters.
- **Security & ops concerns:** HTTP-only, same-site cookies; secure flag in production; bcrypt hashing; rate limiting on auth/cart endpoints; ownership and role checks on protected operations; short cache for public product lists.

2.2.2. Component Diagram



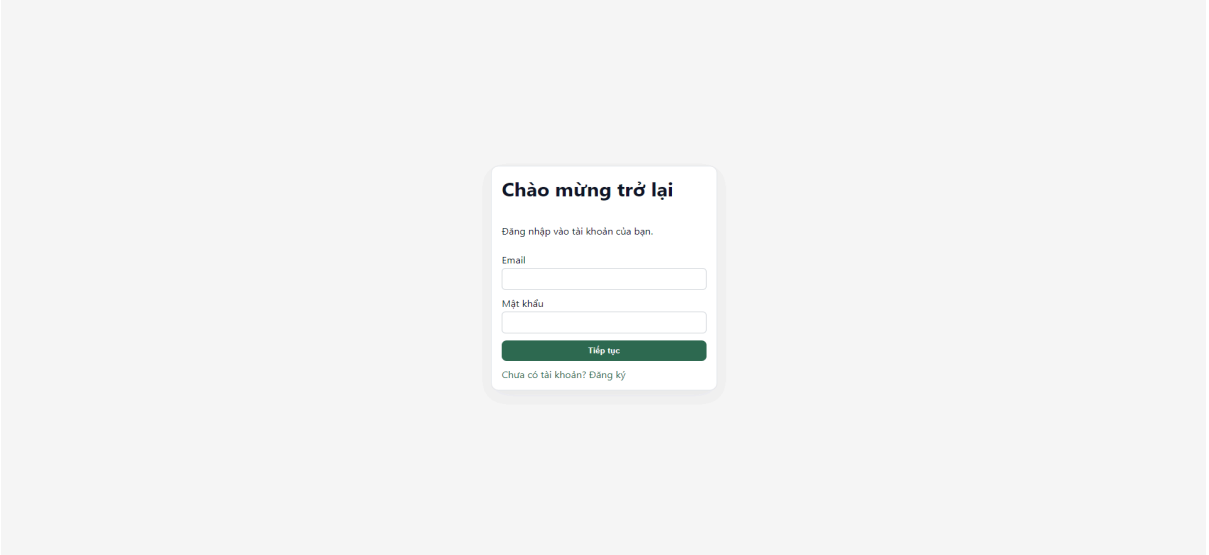
2.2.3. Entity–Relationship Diagram

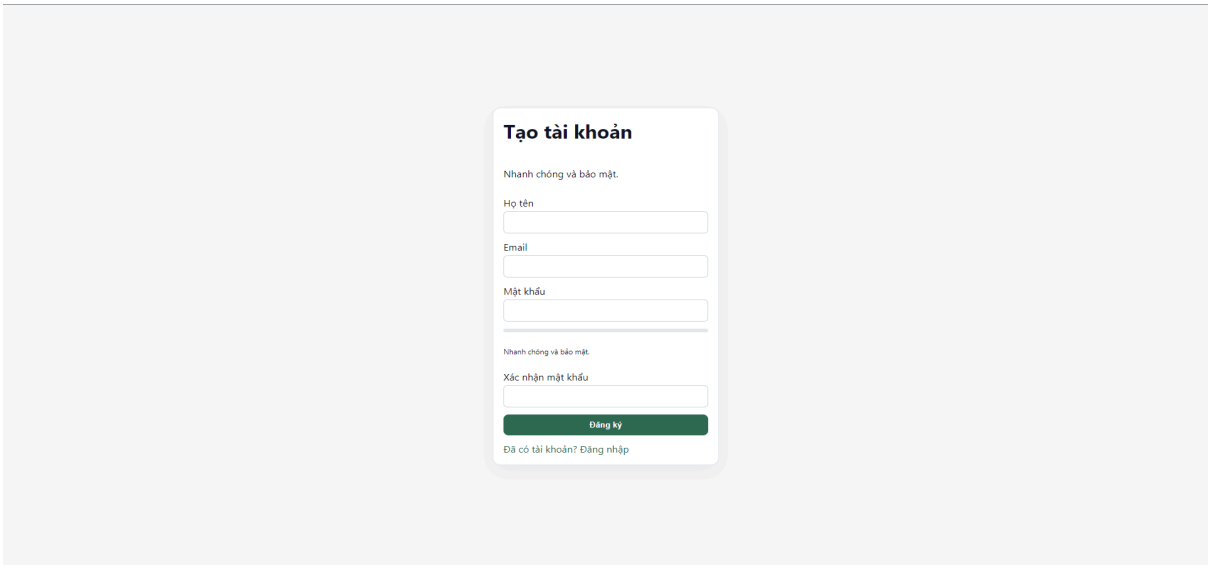


3. IMPLEMENTATION

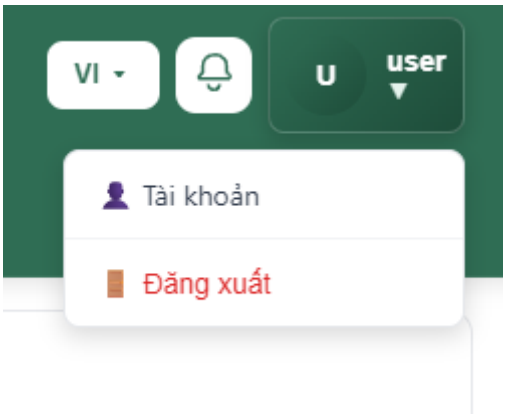
3.1. User Interface Implementation

3.1.1. Login/Register/Logout

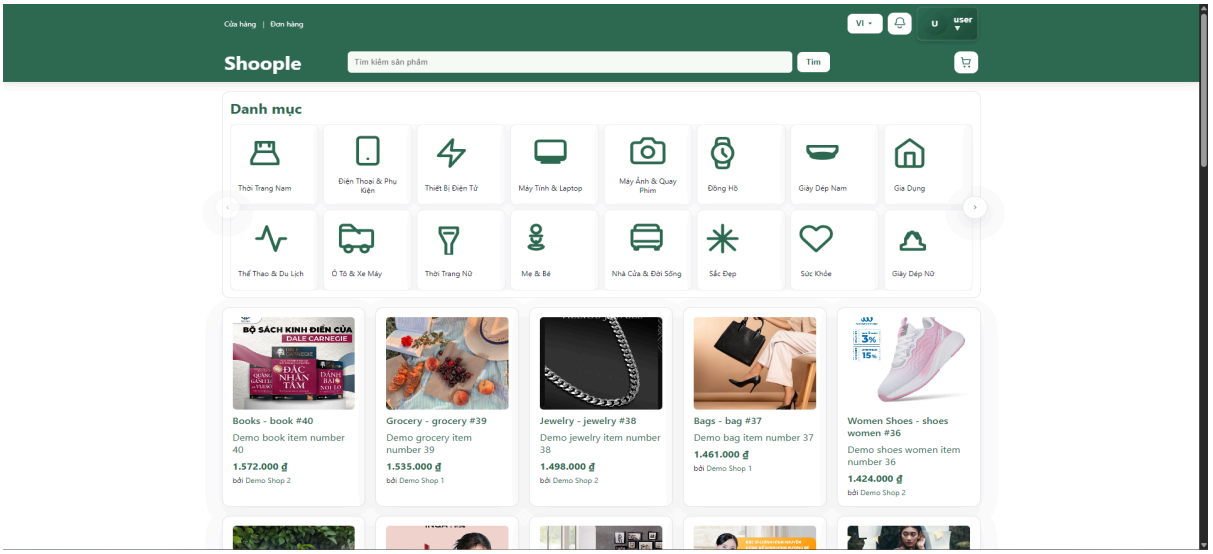


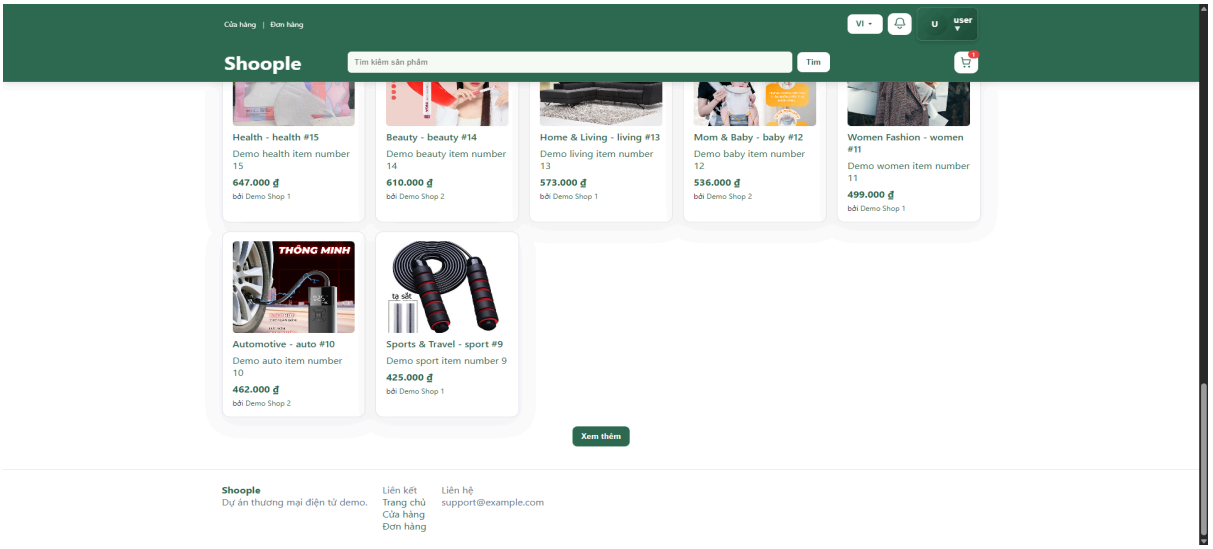


The logout option is located within the dropdown menu of the user avatar.

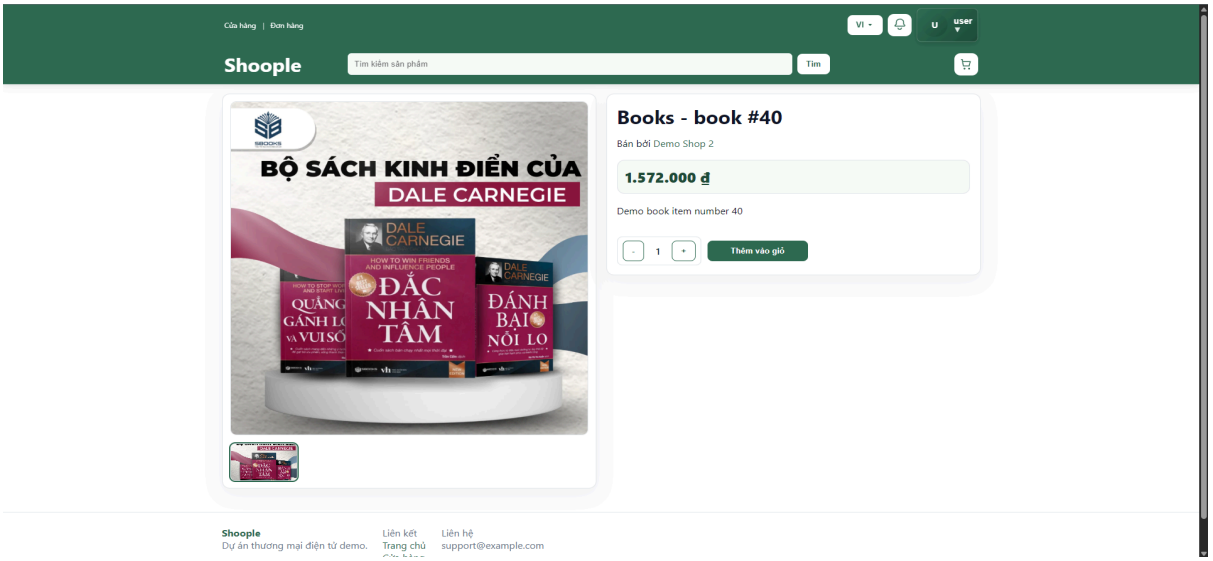


3.1.2. Homepage and Product Listing

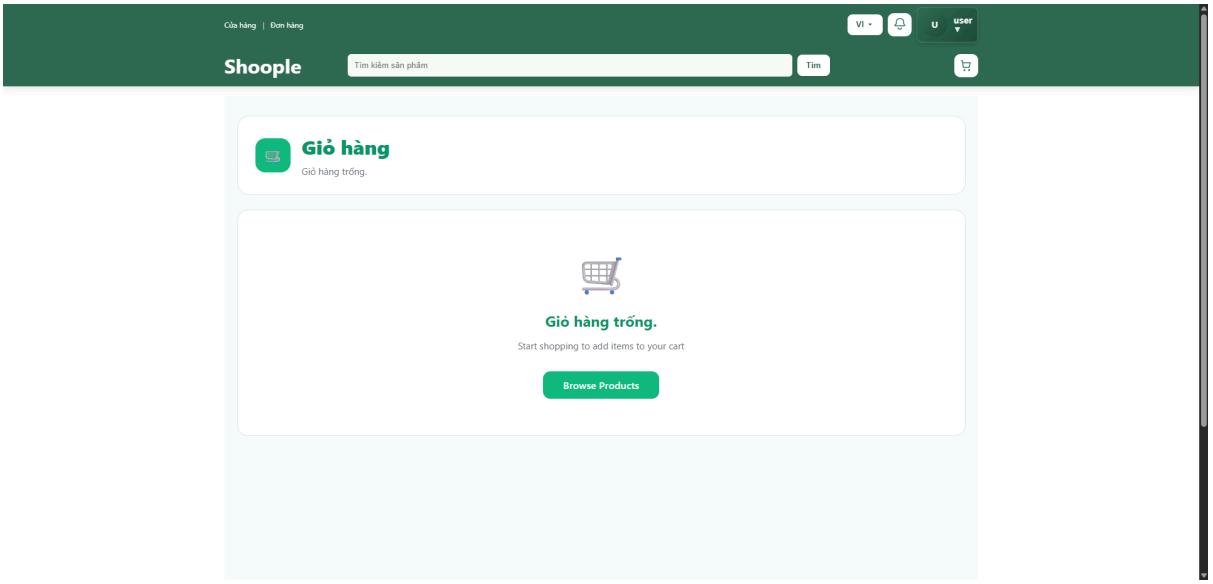


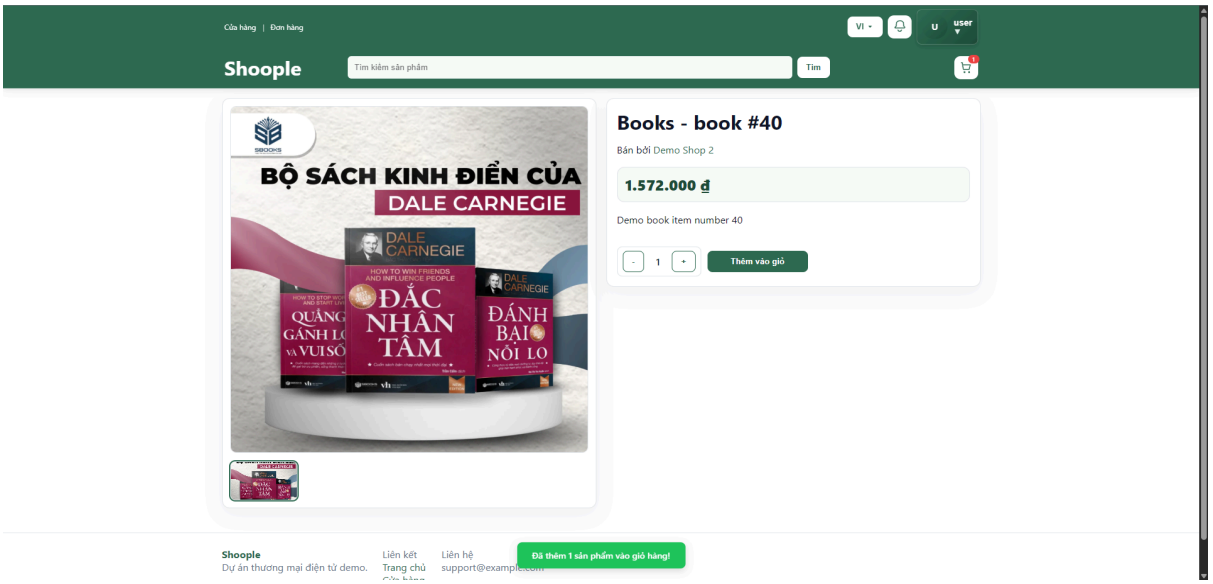


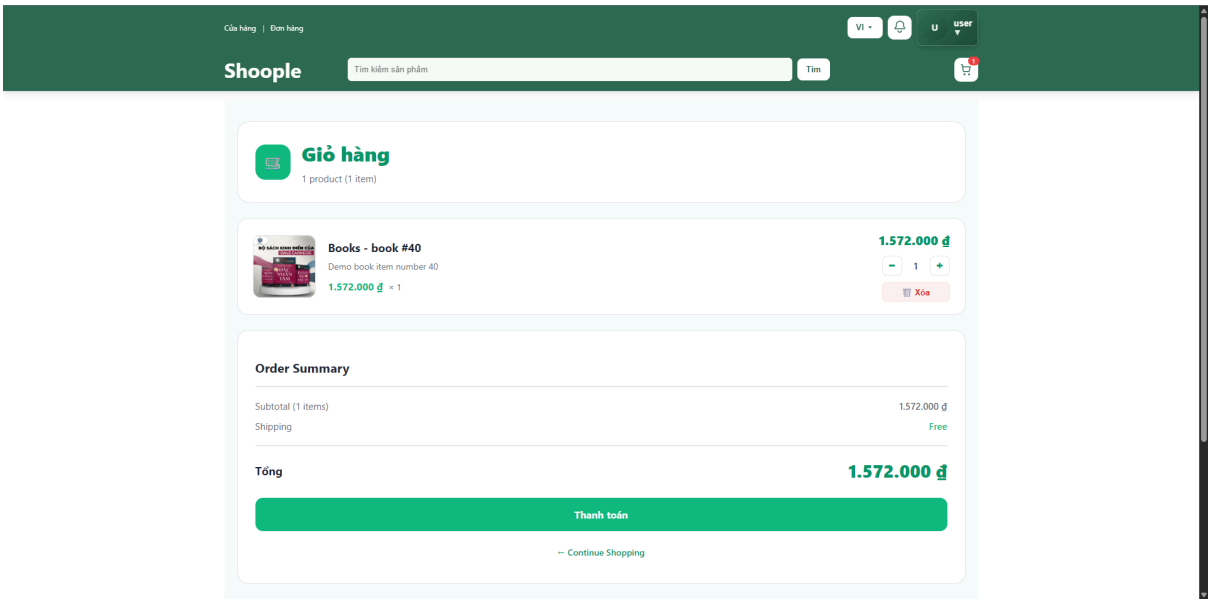
3.1.3. Product Detail Page



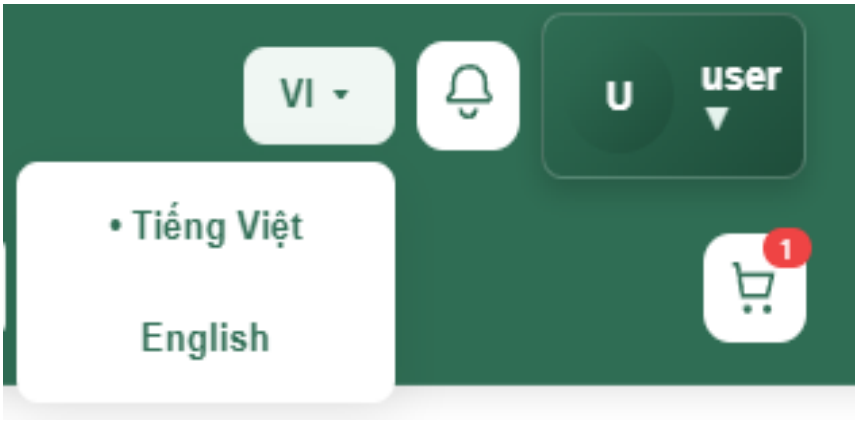
3.1.4. *Shopping Cart Page*

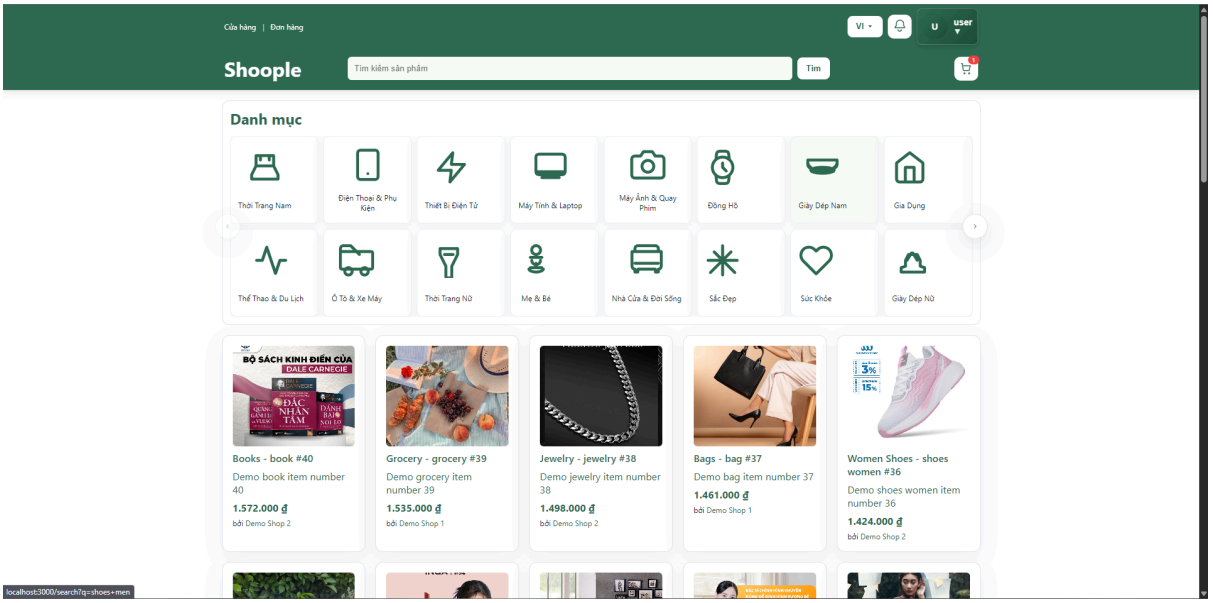


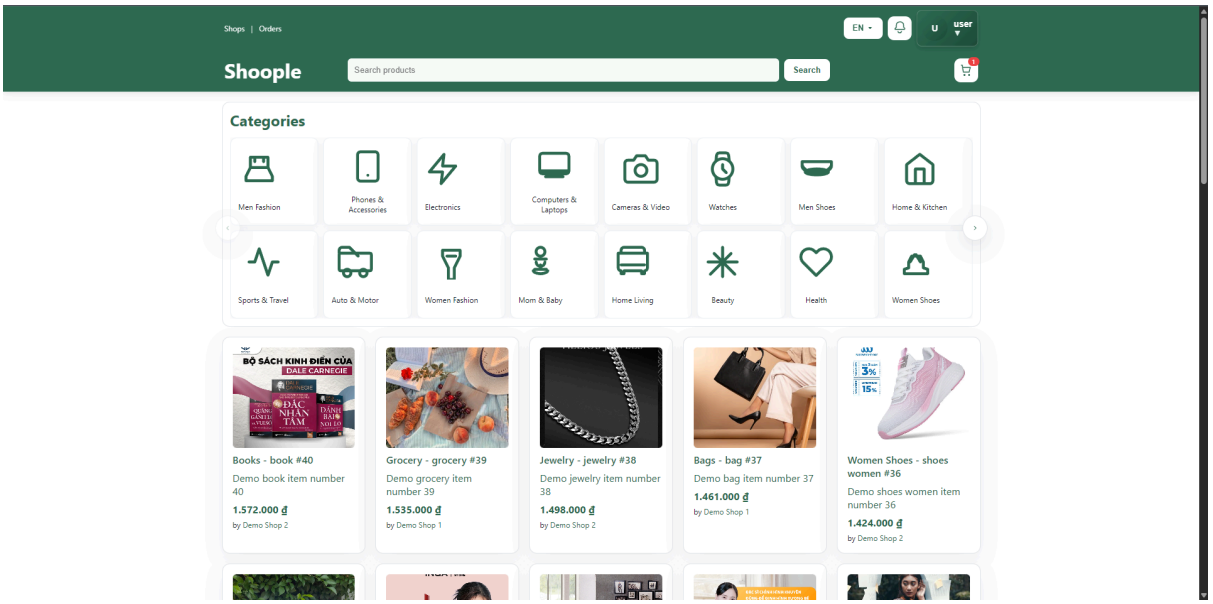




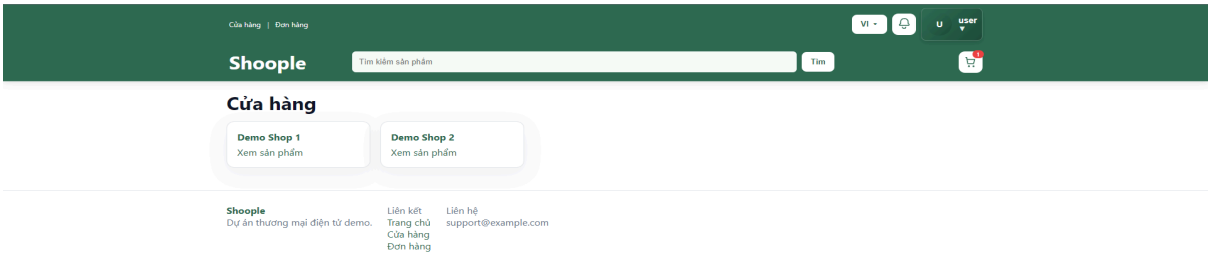
3.1.5. English/Vietnamese UI

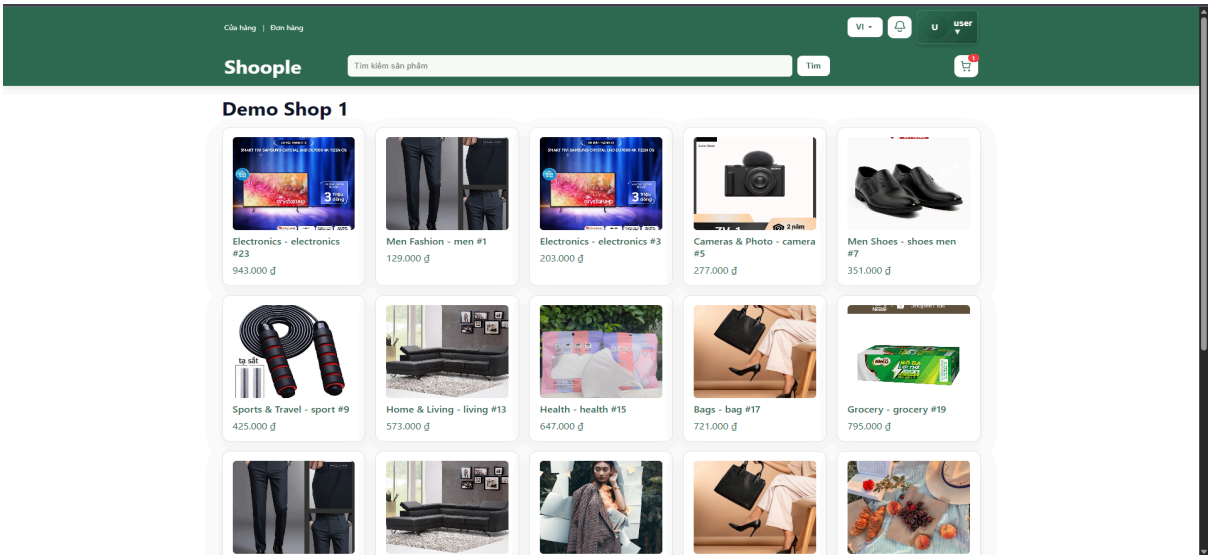




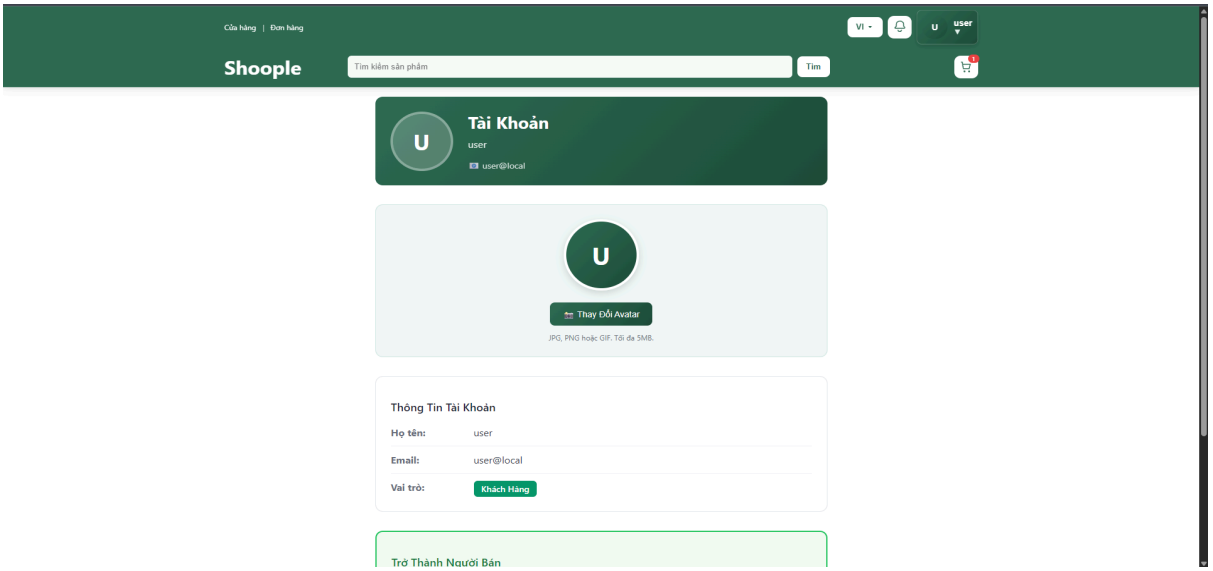


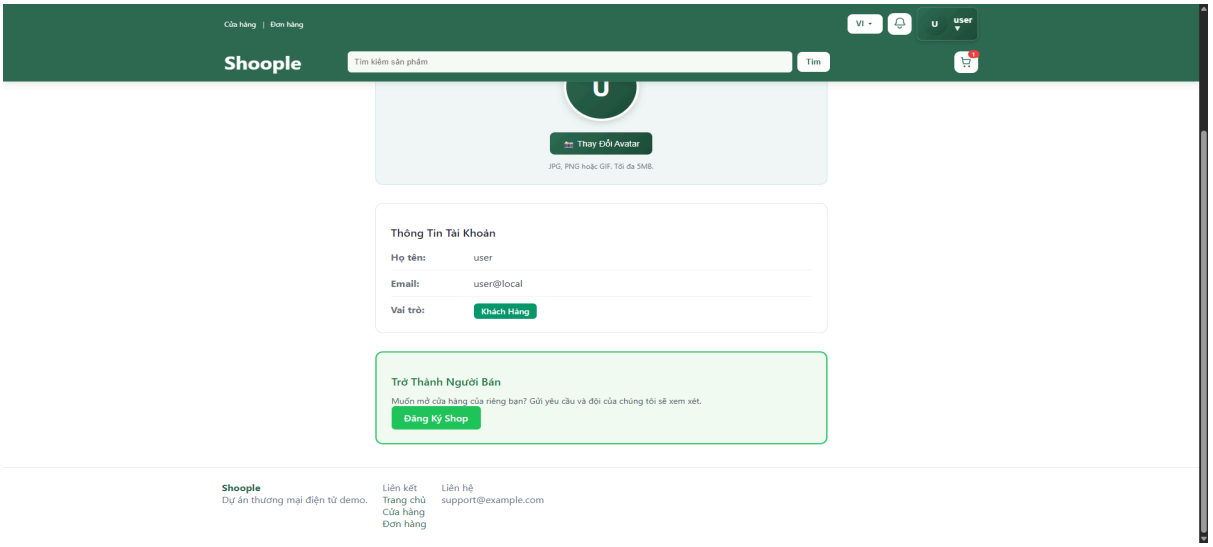
3.1.6. *Shop-lists*

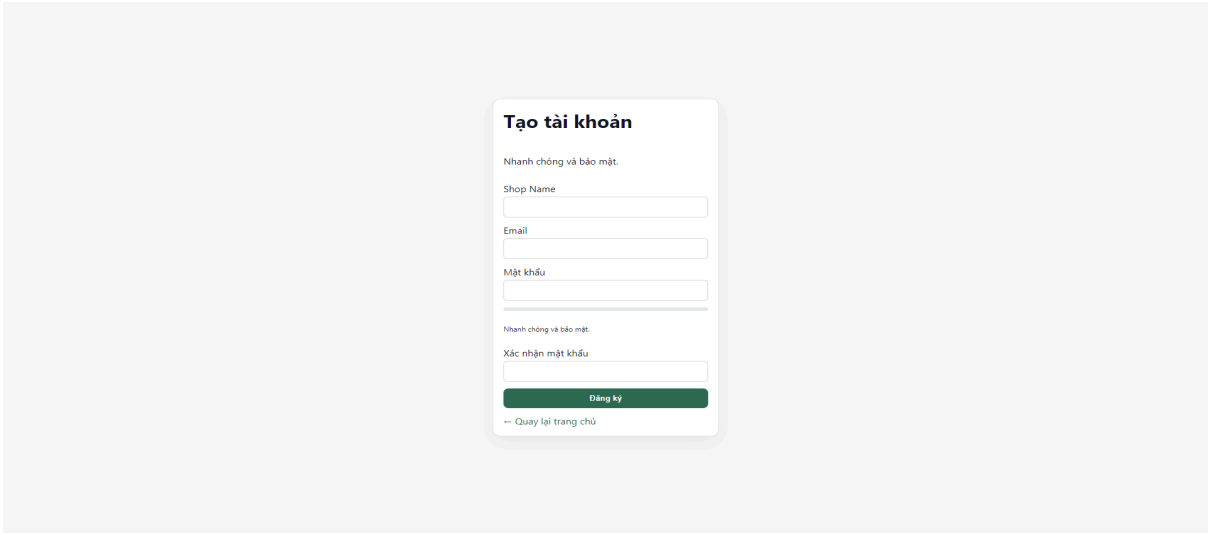




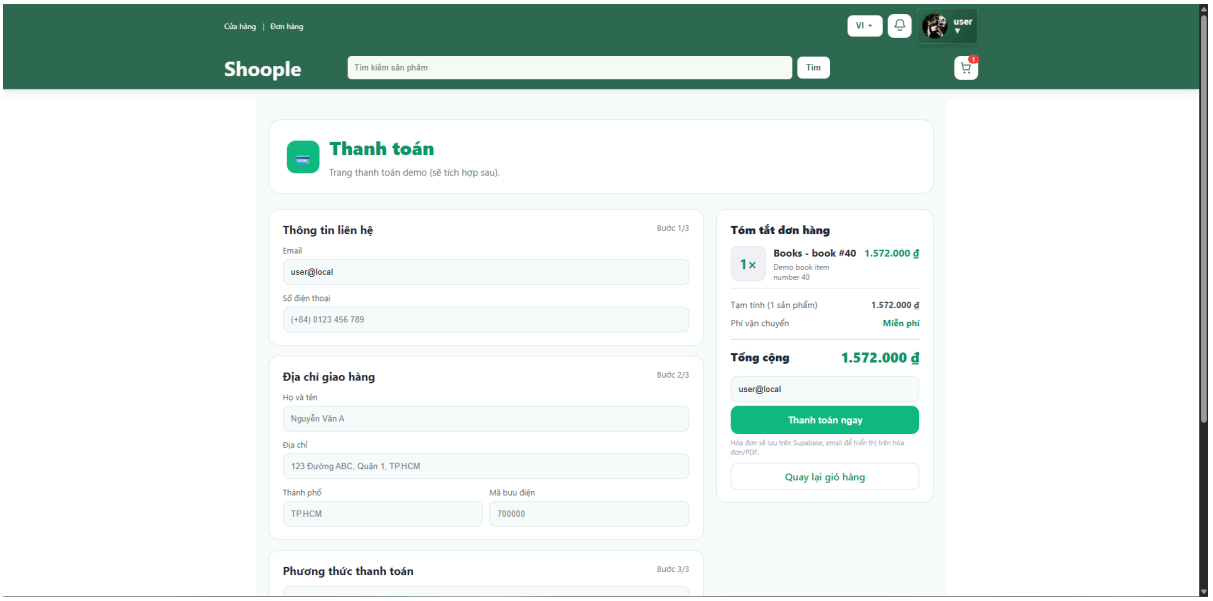
3.1.7. *Account Page/Register Shop*

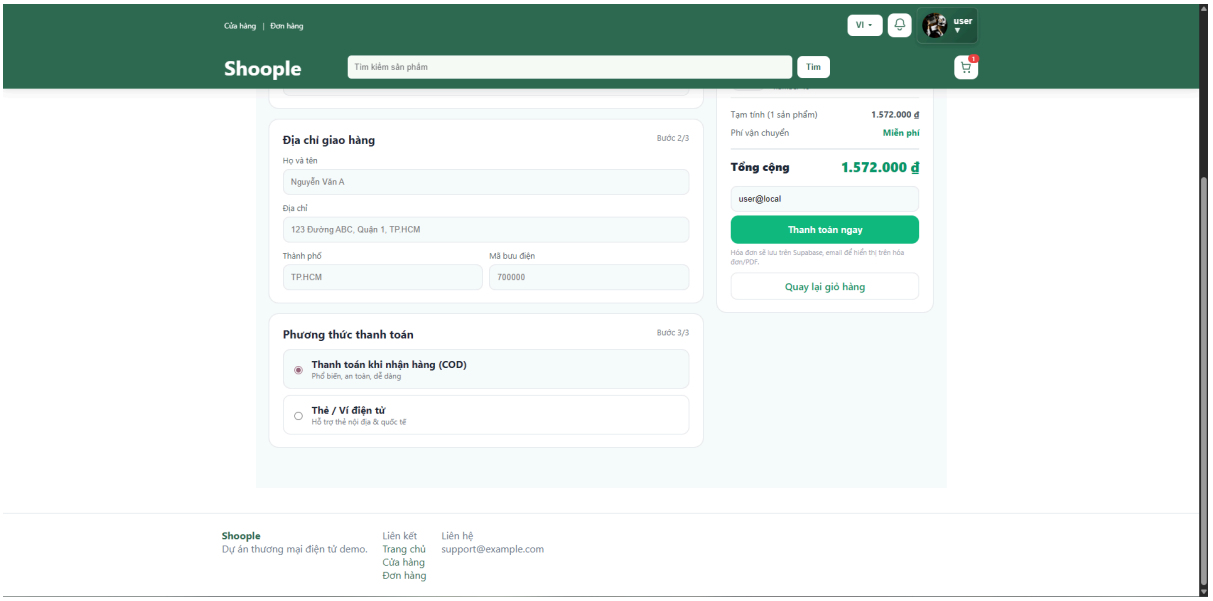


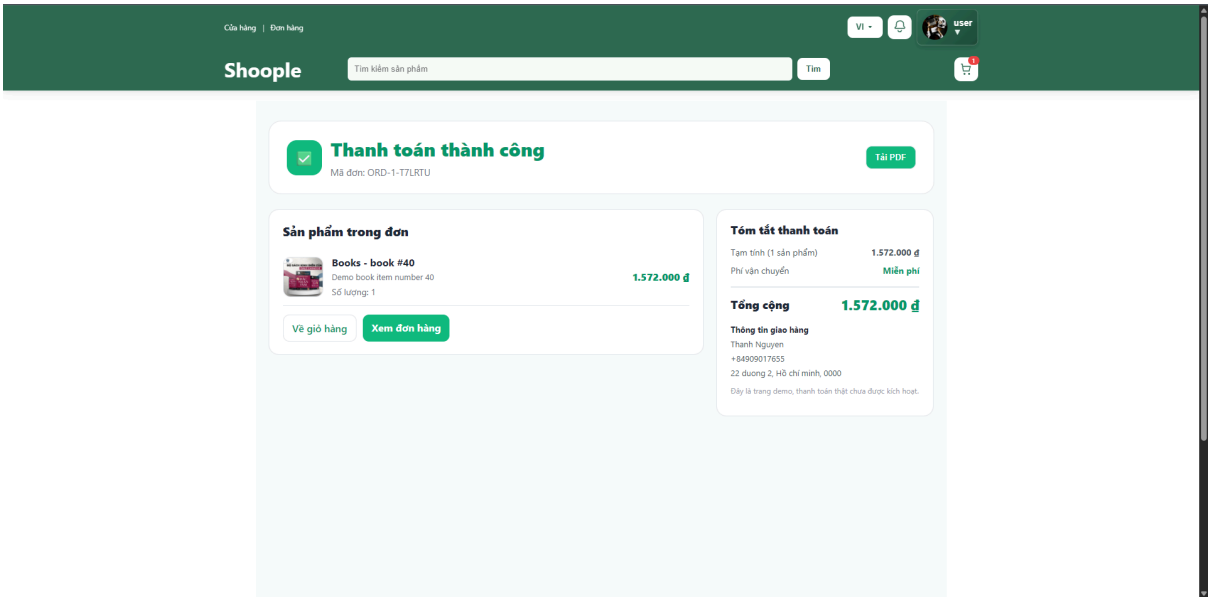


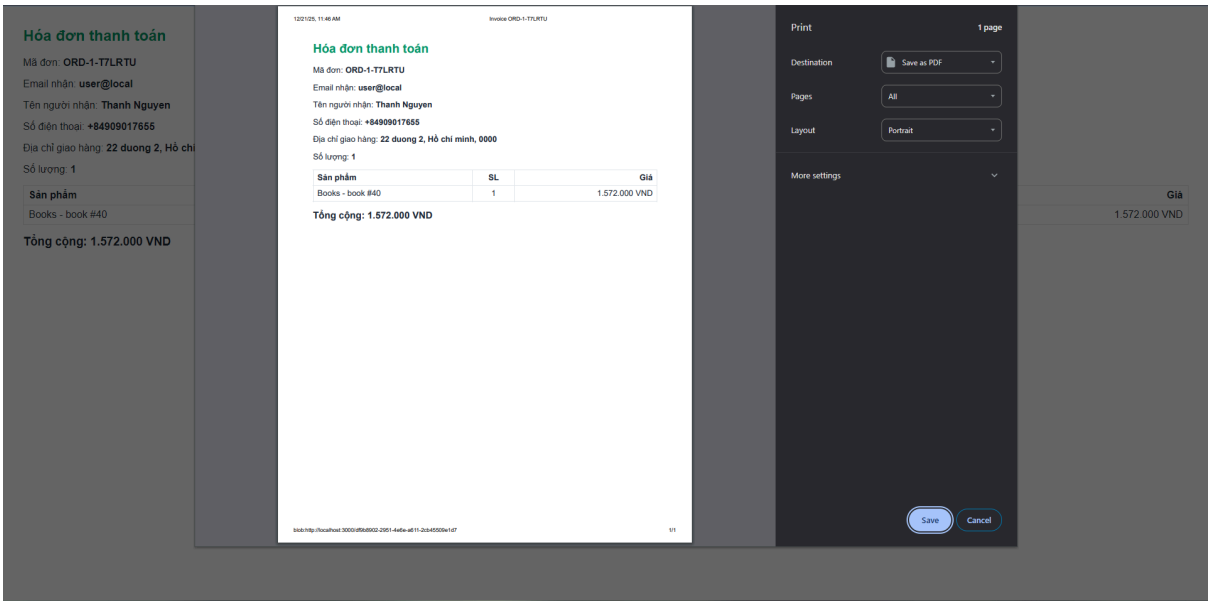


3.1.8. *Checkout Page*

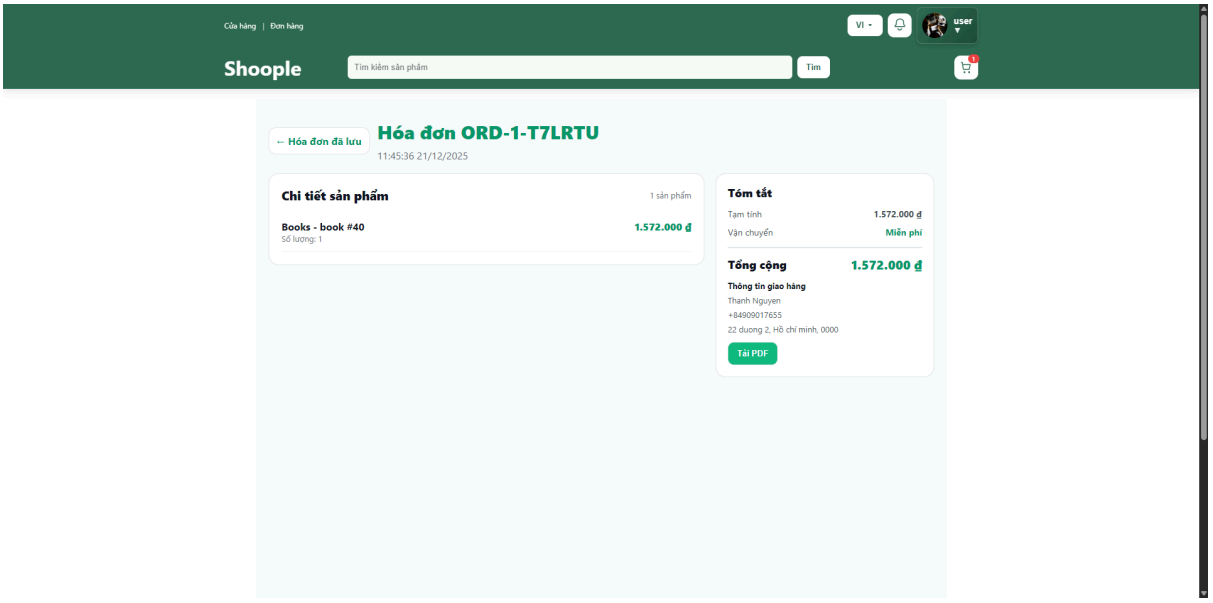


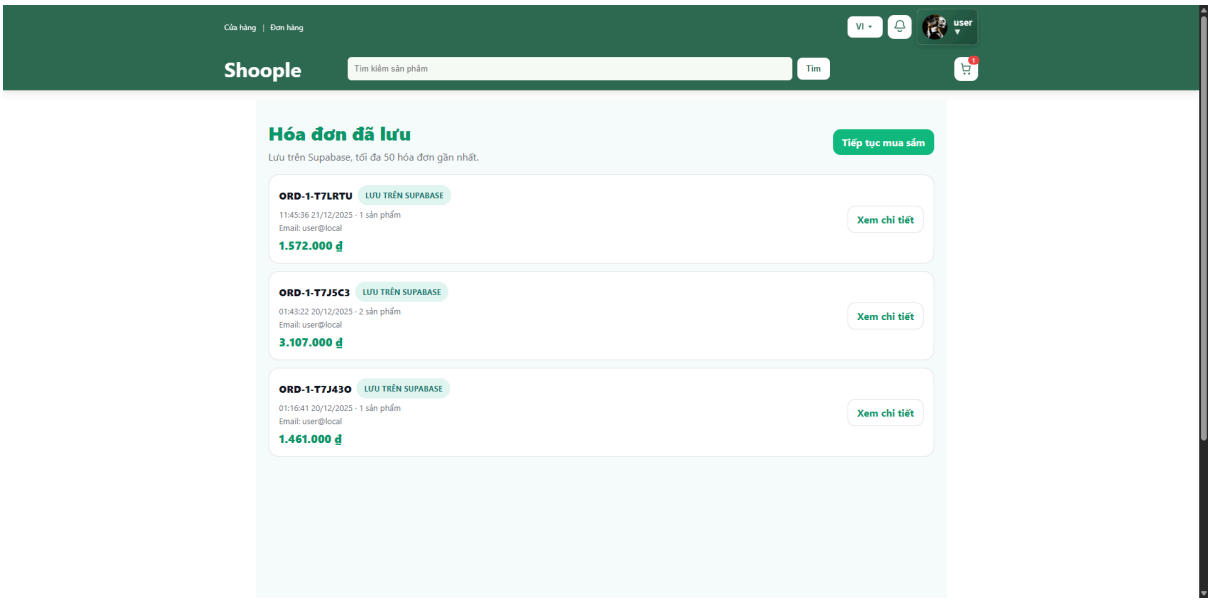

Cửa hàng | Đơn hàng
VI
user
Shoople
Tìm kiếm sản phẩm
Tìm
Địa chỉ giao hàng
Bước 2/3
Họ và tên
Nguyễn Văn A
Địa chỉ
123 Đường ABC, Quận 1, TP.HCM
Thành phố
TP.HCM
Mã bưu điện
700000
Phương thức thanh toán
Bước 3/3
Thanh toán khi nhận hàng (COD)
Phổ biến, an toàn, dễ dàng
Thẻ / Ví điện tử
Hỗ trợ thẻ nội địa & quốc tế
Tạm tính (1 sản phẩm)
1.572.000 ₫
Phí vận chuyển
Miễn phí
Tổng cộng
1.572.000 ₫
user@local
Thanh toán ngay
Hóa đơn sẽ lưu trên Supabase, email để hiển thị trên hóa đơn/PDF.
Quay lại giỏ hàng
Shoople
Dự án thương mại điện tử demo.
Liên kết
Trang chủ
Cửa hàng
Đơn hàng
Liên hệ
support@example.com


Cửa hàng | Đơn hàng
VI
user
Shoople
Tìm kiếm sản phẩm
Tìm
Thanh toán thành công
Mã đơn: ORD-1-T7LRTU
Tải PDF
Sản phẩm trong đơn
Books - book #40
Demo book item number 40
Số lượng: 1
1.572.000 ₫
Về giỏ hàng
Xem đơn hàng
Tóm tắt thanh toán
Tạm tính (1 sản phẩm)
1.572.000 ₫
Phí vận chuyển
Miễn phí
Tổng cộng
1.572.000 ₫
Thông tin giao hàng
Thanh Nguyen
+84909017655
22 duong 2, Hồ chí minh, 0000
Đây là trang demo, thanh toán thật chưa được kích hoạt.


Hóa đơn thanh toán
Mã đơn: ORD-1-T7LRTU
Email nhận: user@local
Tên người nhận: Thanh Nguyen
Số điện thoại: +84909017655
Địa chỉ giao hàng: 22 duong 2, Hồ chí
Số lượng: 1
Sản phẩm
Books - book #40
Tổng cộng: 1.572.000 VND
12/21/25, 11:46 AM
Invoice ORD-1-T7LRTU
Hóa đơn thanh toán
Mã đơn: ORD-1-T7LRTU
Email nhận: user@local
Tên người nhận: Thanh Nguyen
Số điện thoại: +84909017655
Địa chỉ giao hàng: 22 duong 2, Hồ chí minh, 0000
Số lượng: 1
Sản phẩm
SL
Giá
Books - book #40
1
1.572.000 VND
Tổng cộng: 1.572.000 VND
blob:http://localhost:3000/d5bd602-2051-4e6e-a611-2cb455509e1d7
1/1
Print
1 page
Destination
Save as PDF
Pages
All
Layout
Portrait
More settings
Giá
1.572.000 VND
Save
Cancel

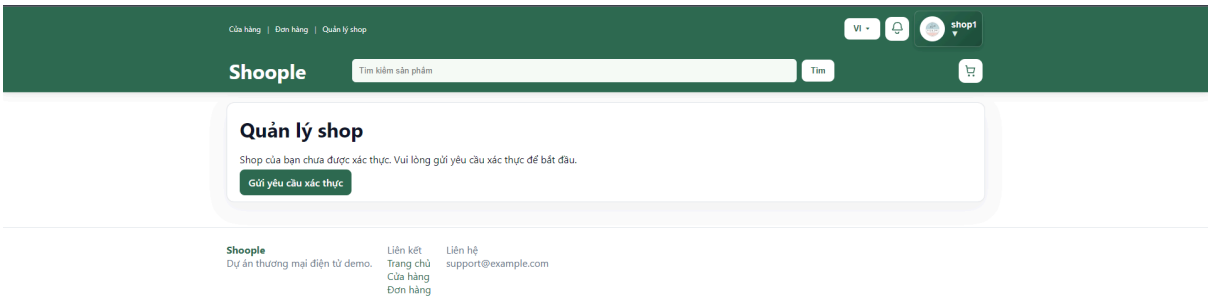
3.1.9. *Invoice History Page*

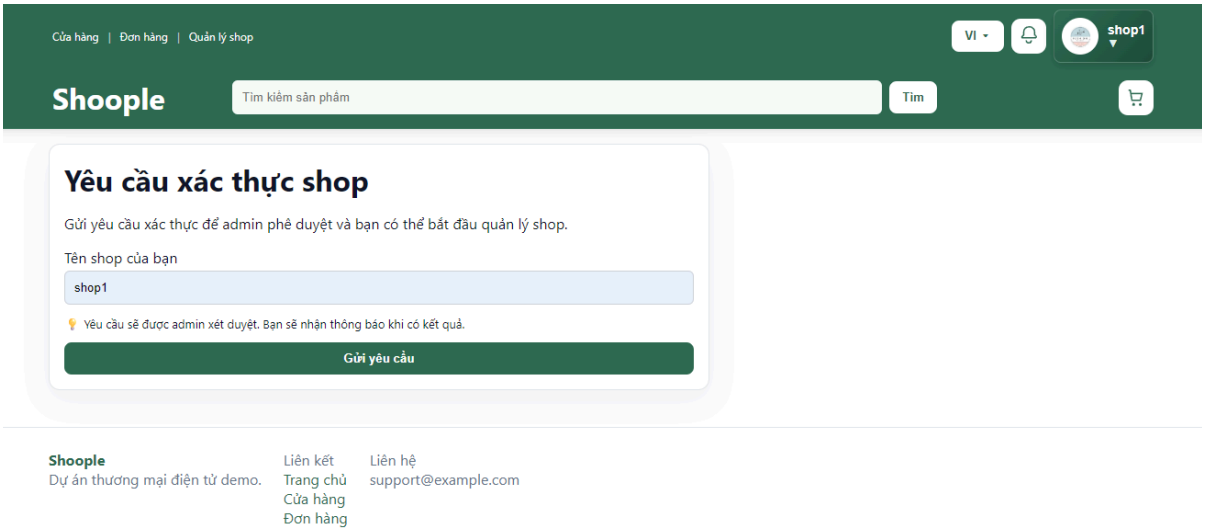




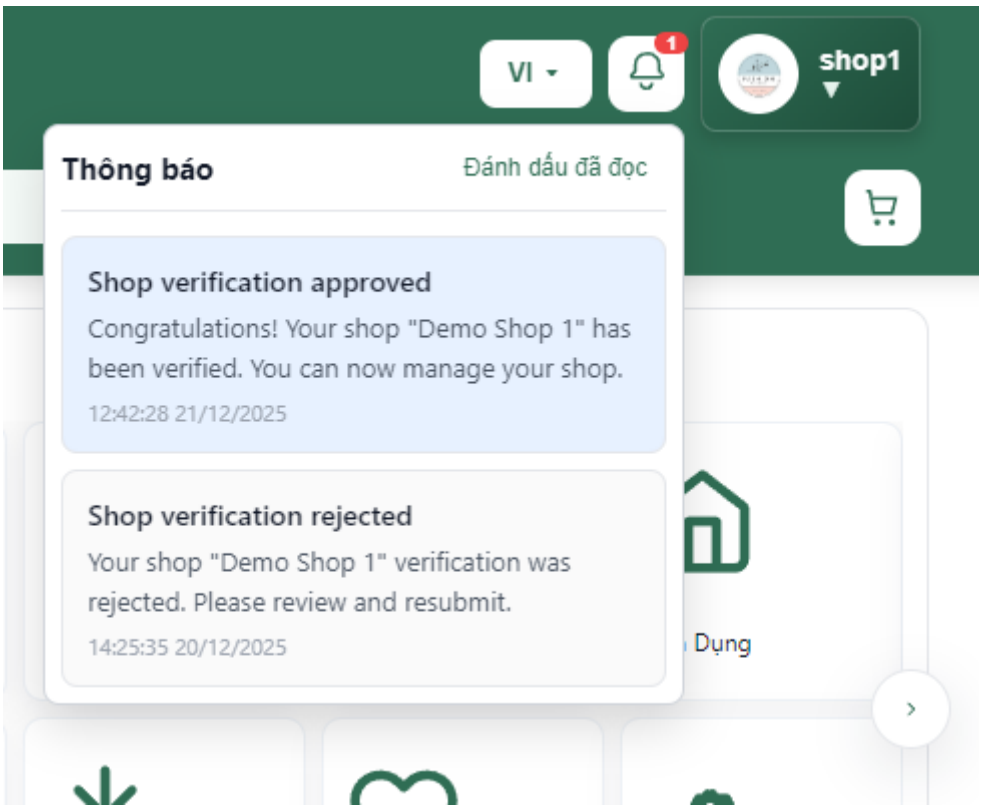
3.1.10. Shop Management Page

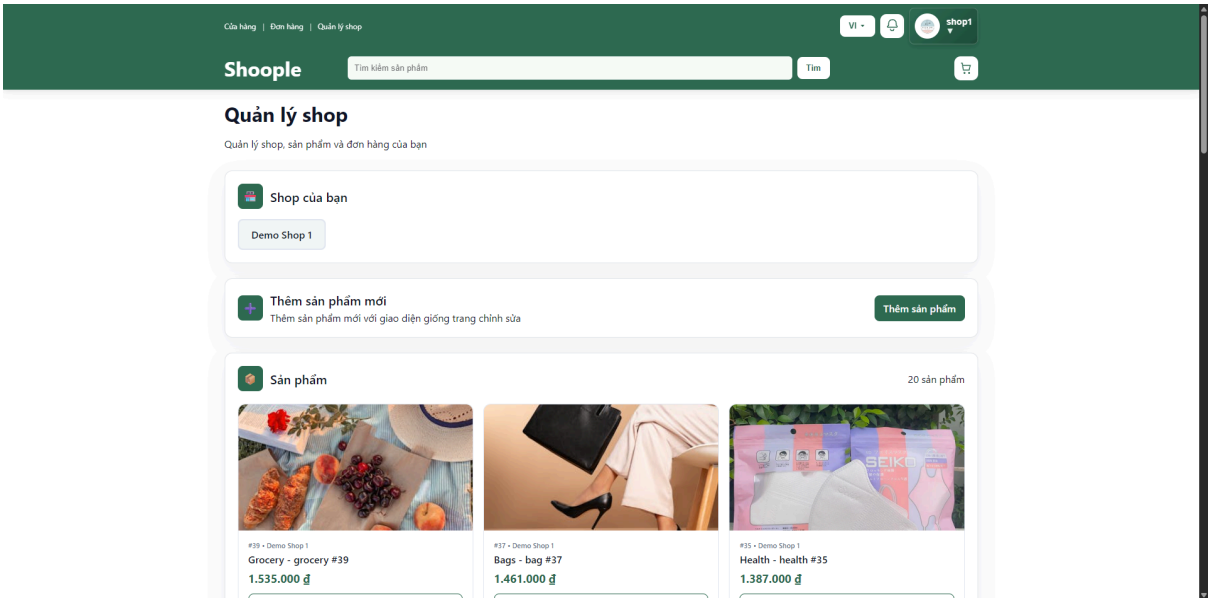
When a new shop account is created, a verification request is automatically sent to the administrator. If the request has not yet been approved or rejected, a notification is displayed to inform the user, and the user is allowed to resubmit the verification request.

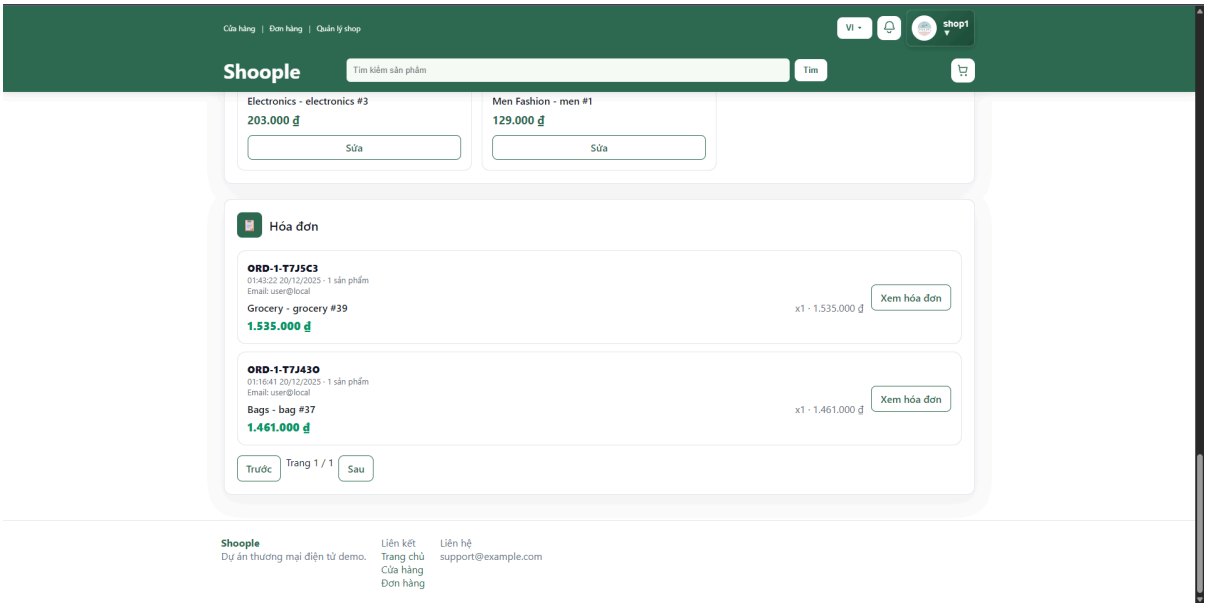


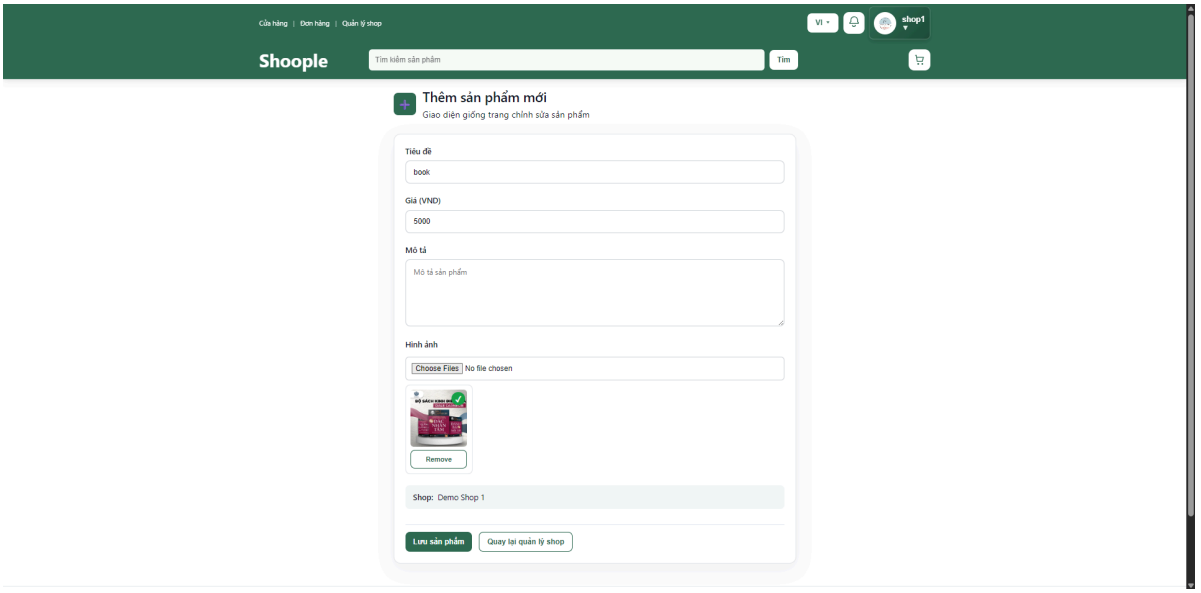


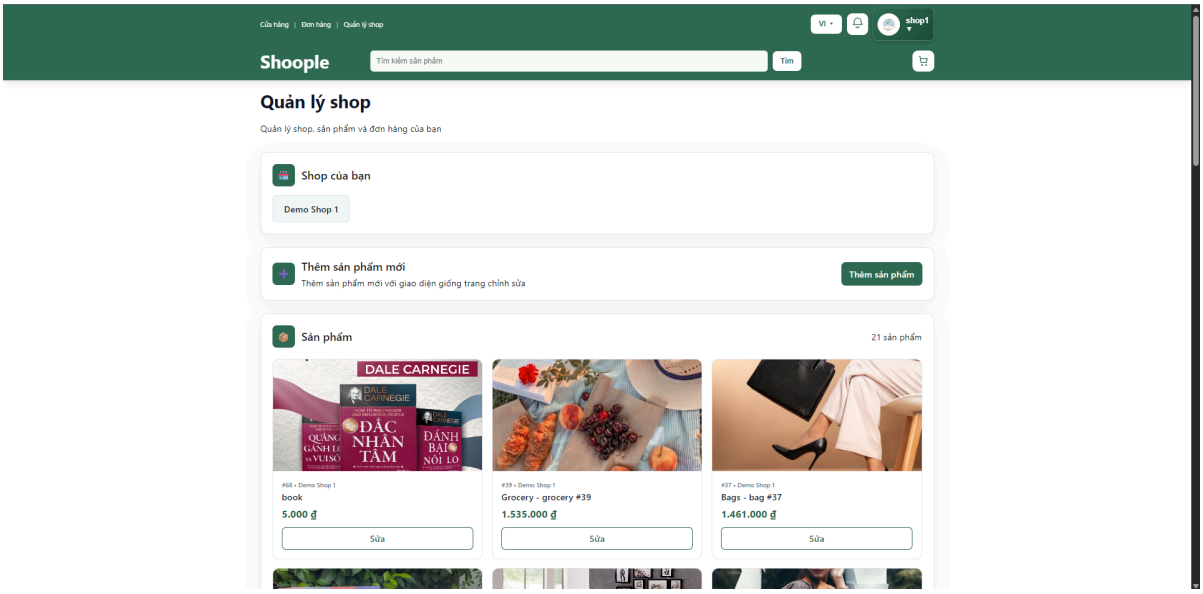
Once the shop account has been successfully verified, the user is granted access to the shop management page.

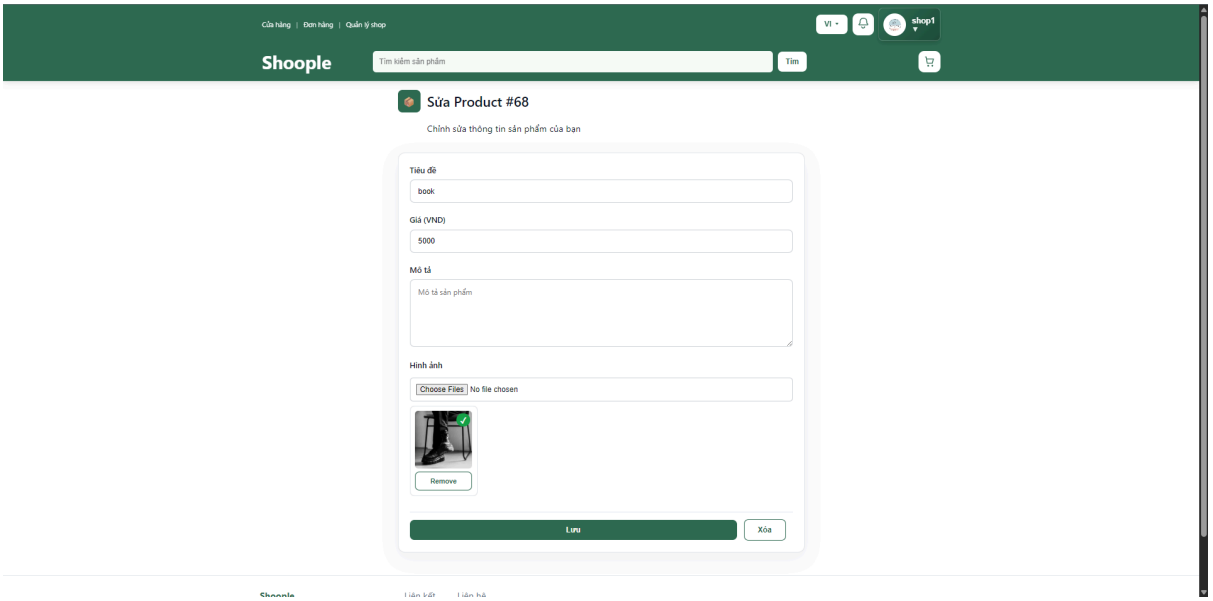




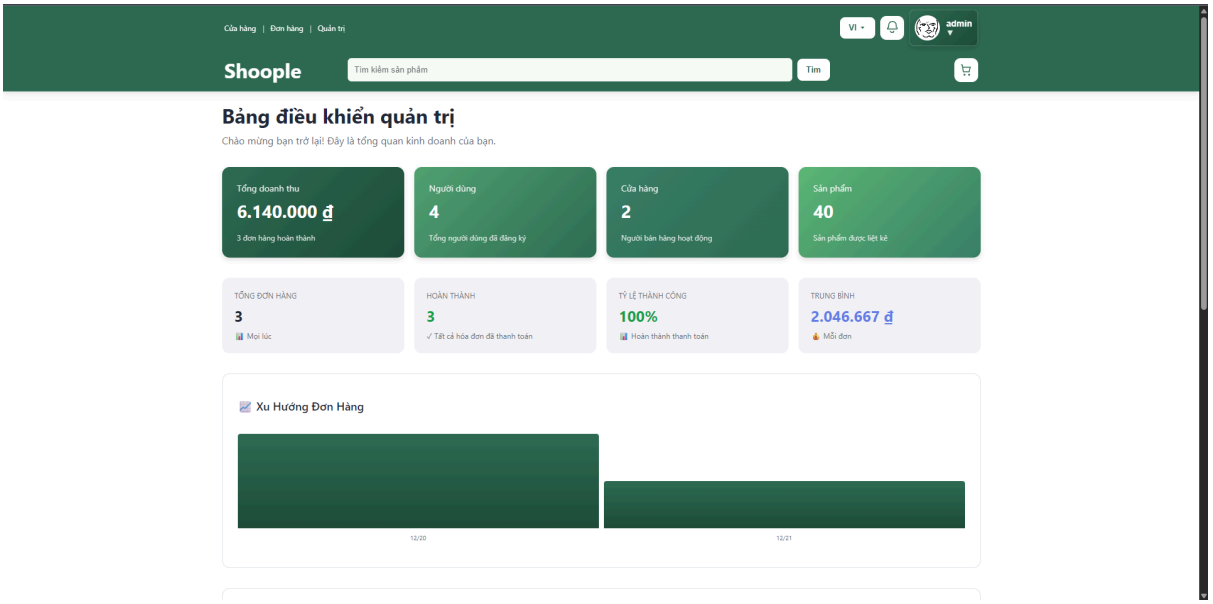

Cửa hàng | Đơn hàng | Quản lý shop
VI
shop1
Shoople
Tìm kiếm sản phẩm
Tìm
Electronics - electronics #3
203.000 ₫
Sửa
Men Fashion - men #1
129.000 ₫
Sửa
Hóa đơn
ORD-1-T7J5C3
01:43:22 20/12/2025 - 1 sản phẩm
Email: user@local
Grocery - grocery #39
1.535.000 ₫
x1 · 1.535.000 ₫
Xem hóa đơn
ORD-1-T7J43O
01:16:41 20/12/2025 - 1 sản phẩm
Email: user@local
Bags - bag #37
1.461.000 ₫
x1 · 1.461.000 ₫
Xem hóa đơn
Trước
Trang 1 / 1
Sau
Shoople
Dự án thương mại điện tử demo.
Liên kết
Trang chủ
Cửa hàng
Đơn hàng
Liên hệ
support@example.com

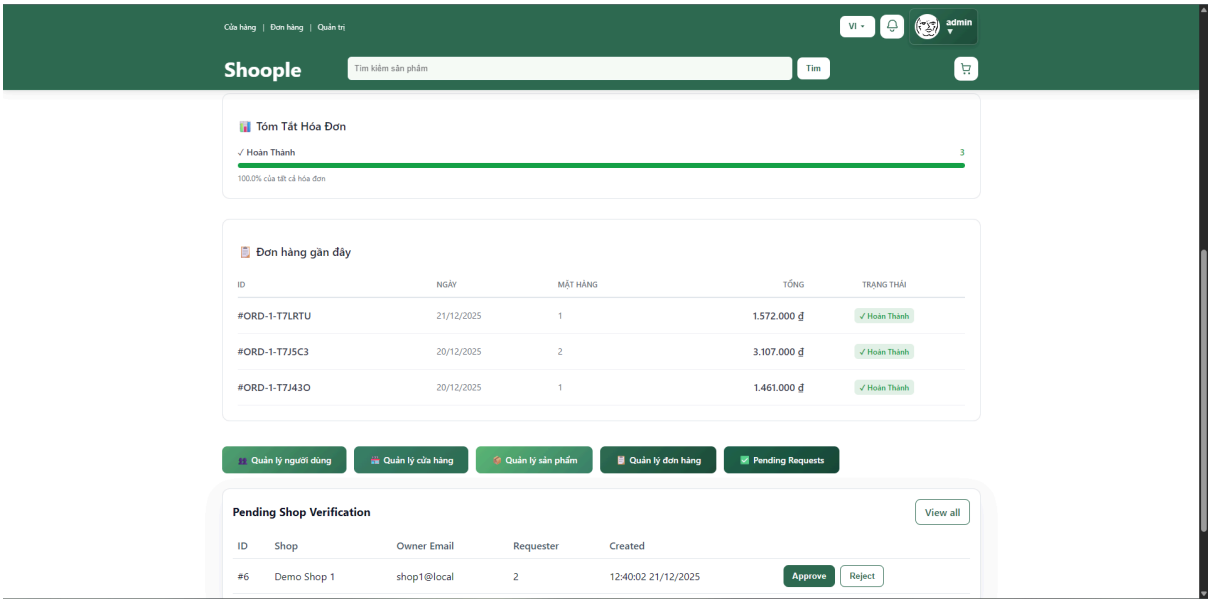

Cửa hàng | Đơn hàng | Quản lý shop
VI
shop1
Shoople
Tìm kiếm sản phẩm
Tìm
Thêm sản phẩm mới
Giao diện giống trang chỉnh sửa sản phẩm
Tiêu đề
book
Giá (VND)
5000
Mô tả
Mô tả sản phẩm
Hình ảnh
Choose Files
No file chosen
Remove
Shop: Demo Shop 1
Lưu sản phẩm
Quay lại quản lý shop

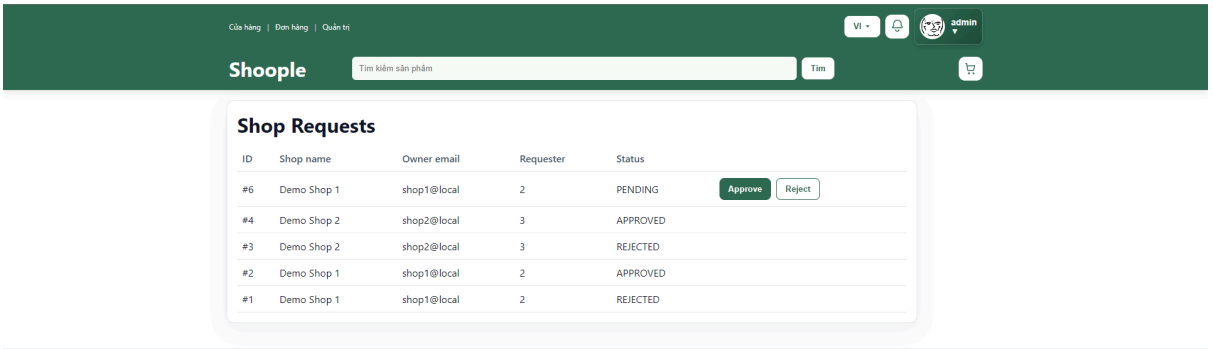

Cửa hàng | Đơn hàng | Quản lý shop
VI
shop1
Shoople
Tìm kiếm sản phẩm
Tìm
Quản lý shop
Quản lý shop, sản phẩm và đơn hàng của bạn
Shop của bạn
Demo Shop 1
Thêm sản phẩm mới
Thêm sản phẩm mới với giao diện giống trang chỉnh sửa
Thêm sản phẩm
Sản phẩm
21 sản phẩm
DALE CARNEGIE
ĐẮC NHÂN TÂM
#68 • Demo Shop 1
book
5.000 ₫
Sửa
#39 • Demo Shop 1
Grocery - grocery #39
1.535.000 ₫
Sửa
#37 • Demo Shop 1
Bags - bag #37
1.461.000 ₫
Sửa



3.1.11. admin dashboard



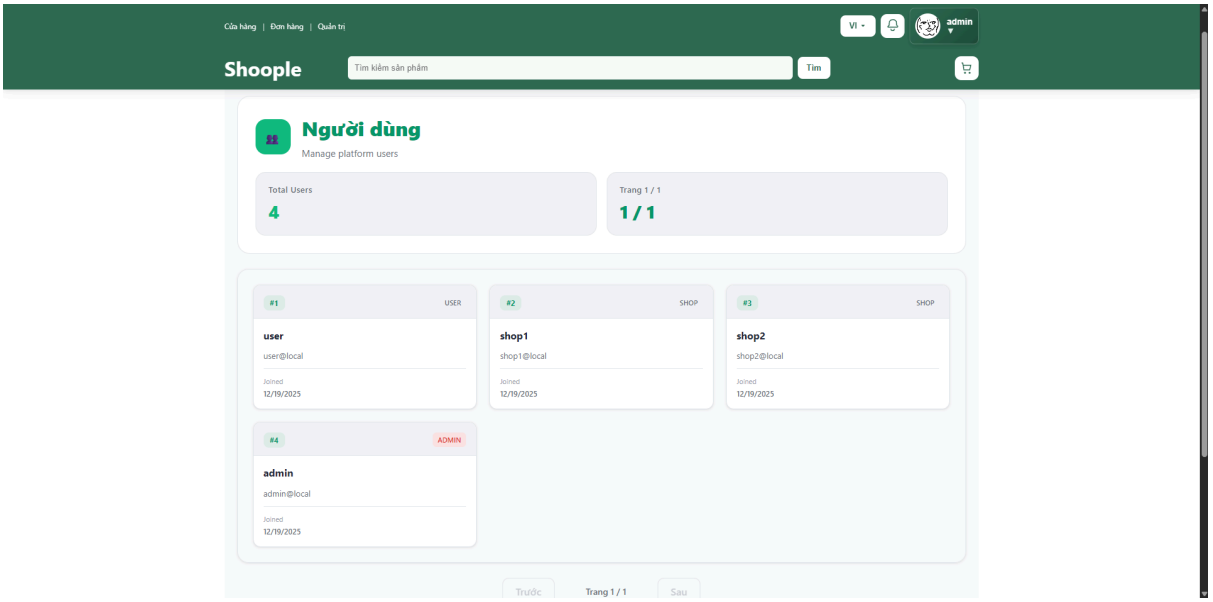


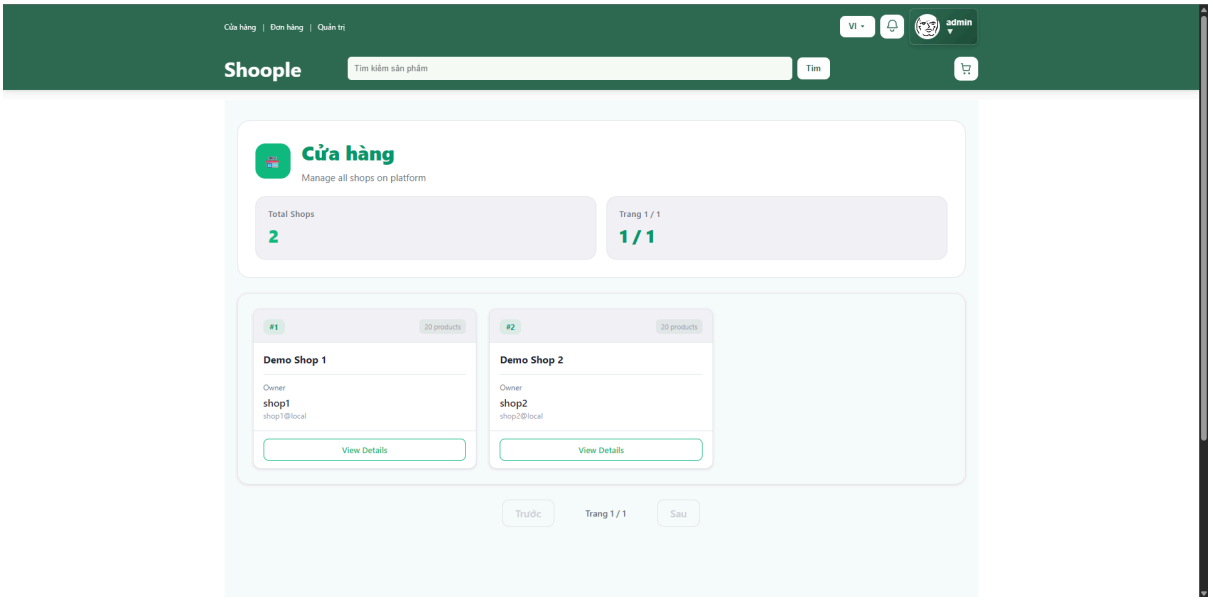


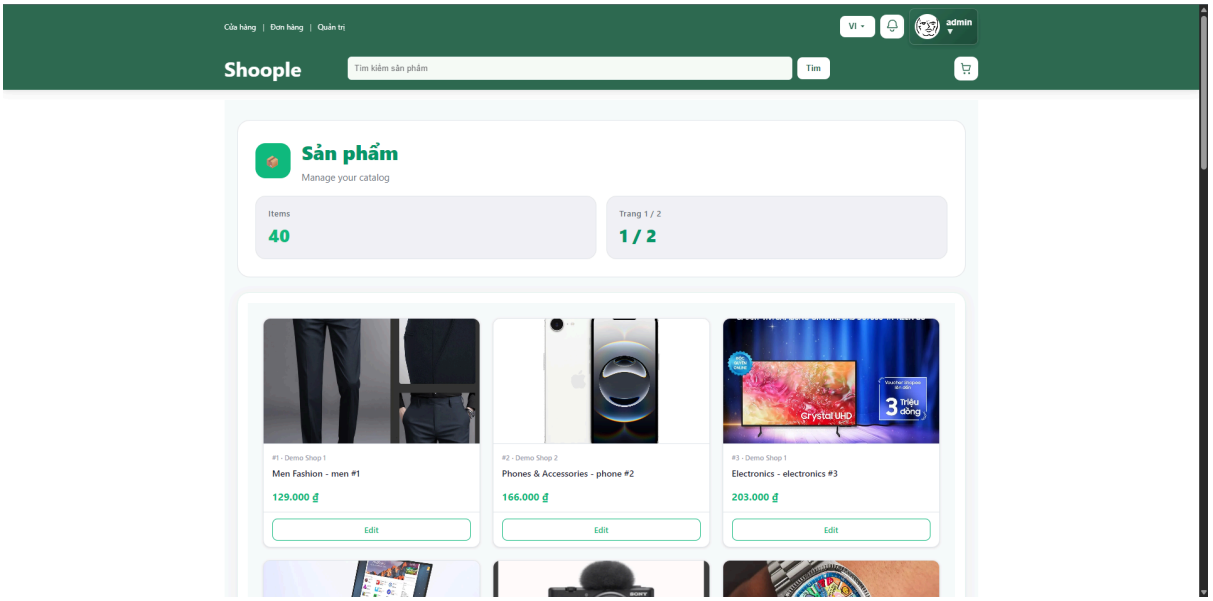
Shoople
Dự án thương mại điện tử demo.

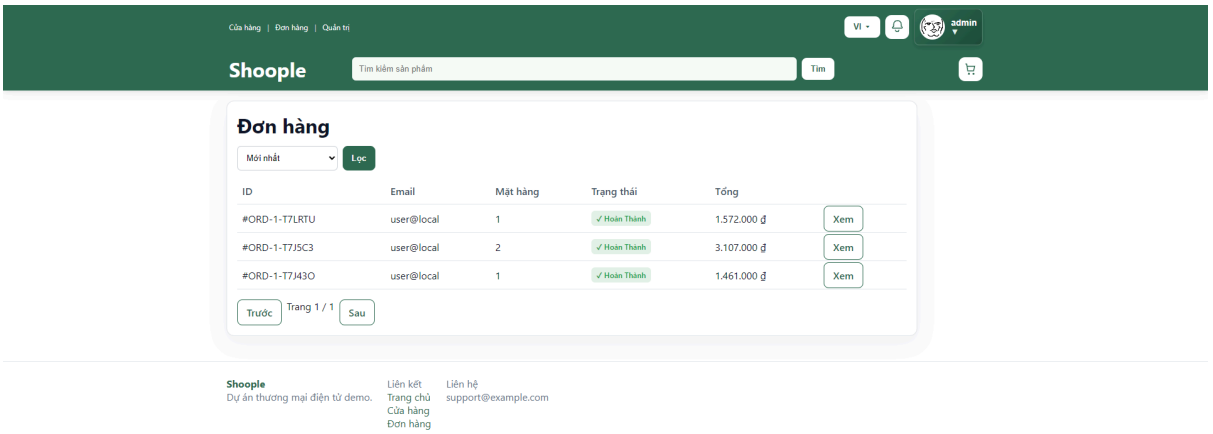
Liên kết
Trang chủ
Cửa hàng
Đơn hàng

Liên hệ
support@example.com




Cửa hàng | Đơn hàng | Quản trị
VI
admin
Shoople
Tìm kiếm sản phẩm
Tìm
Cửa hàng
Manage all shops on platform
Total Shops
2
Trang 1 / 1
1 / 1
#1
20 products
Demo Shop 1
Owner
shop1
shop1@local
View Details
#2
20 products
Demo Shop 2
Owner
shop2
shop2@local
View Details
Trước
Trang 1 / 1
Sau


Cửa hàng | Đơn hàng | Quản trị
VI
admin
Shoople
Tìm kiếm sản phẩm
Tìm
Sản phẩm
Manage your catalog
Items
40
Trang 1 / 2
1 / 2
#1 - Demo Shop 1
Men Fashion - men #1
129.000 ₫
Edit
#2 - Demo Shop 2
Phones & Accessories - phone #2
166.000 ₫
Edit
#3 - Demo Shop 1
Electronics - electronics #3
203.000 ₫
Edit


Cửa hàng | Đơn hàng | Quản trị
VI
admin
Shoople
Tìm kiếm sản phẩm
Tìm
Đơn hàng
Mới nhất
Lọc
ID
Email
Mặt hàng
Trạng thái
Tổng
#ORD-1-T7LRTU
user@local
1
✓ Hoàn Thành
1.572.000 ₫
Xem
#ORD-1-T7J5C3
user@local
2
✓ Hoàn Thành
3.107.000 ₫
Xem
#ORD-1-T7J43O
user@local
1
✓ Hoàn Thành
1.461.000 ₫
Xem
Trước
Trang 1 / 1
Sau

3.2. Functional Implementation

This section describes the core functional logic implemented in the Shople e-commerce platform. The focus is on how data is processed, how user interactions are handled, and how system states are updated during runtime through backend APIs and frontend components.

3.2.1. Product Data Handling

Product data is stored and managed using a PostgreSQL database hosted on Supabase. The database schema is designed in a normalized manner to support both primary product information and multiple product images.

The main product information is stored in a products table, which includes attributes such as product identifier, title, description, price, primary image URL, associated shop identifier, and timestamp fields for creation and updates. To support image galleries, additional images are stored in a separate **product_images** table, linked to the corresponding product through a foreign key relationship.

When users browse or search for products, data is fetched asynchronously through backend API routes. The listing logic supports pagination and keyword-based searching, allowing users to retrieve products efficiently without loading unnecessary data. Product detail pages retrieve both the main product record and its associated image gallery.

To maintain a clean separation of concerns, raw database records are mapped into UI-friendly data structures before being consumed by frontend components. This design improves maintainability and allows future changes to the database schema without significantly impacting the user interface.

3.2.2. Cart Logic and State Update

The shopping cart functionality is implemented using authenticated, server-side persistence rather than client-side browser storage. Each user is associated

with a single cart, which is stored in a dedicated carts table and linked to individual cart items stored in a **cart_items** table.

Cart operations, including adding products, updating quantities, and removing items, are performed through secure backend API endpoints. When a user interacts with the cart, the frontend sends requests to these APIs, and the updated cart state is returned and rendered immediately on the user interface, ensuring responsive feedback.

Cart data is persisted across sessions, allowing users to resume their shopping activity after page refreshes or re-authentication. Quantity updates are handled using idempotent operations to prevent duplication or inconsistent states. At the current stage, stock validation is not enforced due to the absence of inventory-related fields in the product schema; however, the system design allows such constraints to be added in future iterations.

3.2.3. Invoice Generation Logic

Invoice generation occurs after a successful checkout operation. Once the checkout process is completed, the system aggregates all cart items and calculates the total price and item count based on product price and quantity.

An invoice record is then created and stored in the database using an idempotent upsert strategy based on a unique order identifier. Each invoice includes user information, total amount, item count, and a structured payload containing order items and related contact or shipping data.

After the invoice is successfully persisted, the system clears the user's cart to ensure that completed orders are not duplicated. Users can later access their order history, which is retrieved directly from the stored invoice records. A demonstration endpoint for invoice email sending is included as a placeholder, allowing future integration with a real email delivery service without modifying the core invoice logic.

3.2.4. Data Formatting and Time Handling

To ensure consistency and readability, the system applies standardized formatting rules for numerical and temporal data. Monetary values are stored as integers in Vietnamese Dong (VND) and formatted into human-readable currency strings before being displayed on the user interface.

Timestamp data is stored in the database using timezone-aware formats, ensuring consistency across different regions and environments. These timestamps are passed directly to the frontend and can be formatted as needed for display purposes, while the persisted values remain accurate and portable.

This approach separates raw data representation from presentation formatting, improving both data integrity and user experience.

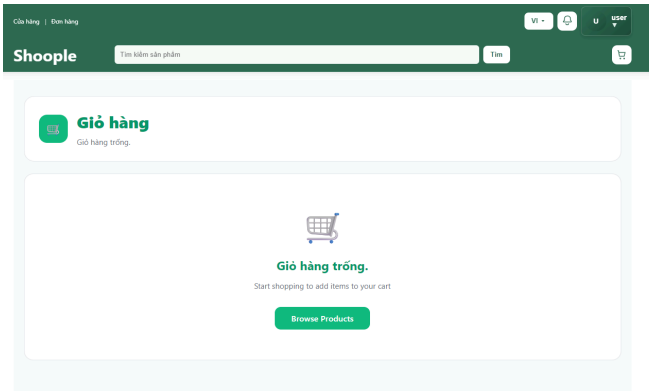
3.3. Error Handling and Edge Cases

This section describes how the system handles common error scenarios and edge cases in order to maintain a stable user experience and prevent unexpected application states. The focus is placed on user interface behavior and current backend limitations based on the existing implementation.

3.3.1. Empty Cart Handling

The application provides explicit user interface handling for empty cart scenarios. When a user accesses the cart or checkout page without any items in the cart, an informative message is displayed to indicate that the cart is empty. In this state, the checkout flow is not presented, and users are encouraged to add products before proceeding.

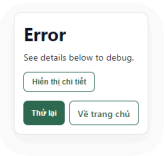
This behavior is implemented at the user interface level on both the cart and checkout pages, ensuring that users do not accidentally attempt to place an order without selected items. At the current stage, the backend invoice endpoint functions as a demonstration and does not validate the cart state in the database to block empty-cart submissions. Such validation is identified as a potential improvement for future iterations.

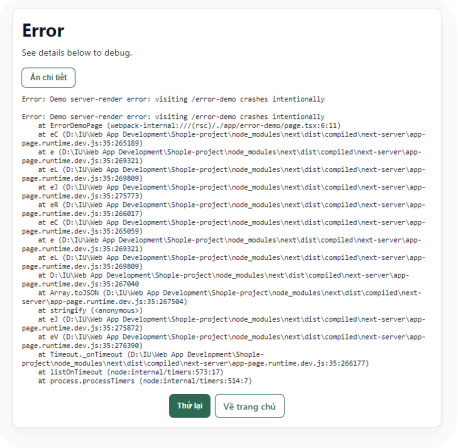


3.3.2. Error Page

The application utilizes a global error handling mechanism provided by the Next.js App Router. When a runtime error or data-fetching failure occurs within a route, the system renders a dedicated error page through an error boundary rather than redirecting the user to a different route.

This approach ensures that errors are presented in a controlled and user-friendly manner while preventing the exposure of sensitive internal details. In production mode, the error page displays a minimal reference message, maintaining application security and improving the overall robustness of the user experience.





3.3.3. Invalid Product Data

The system includes basic validation mechanisms to handle invalid or inconsistent product data. During product creation, backend APIs validate essential fields such as product title, price, and associated shop identifier before persisting data to the database.

For cart-related operations, the current implementation performs basic input validation, such as checking the format and type of product identifiers. The cart APIs rely on database constraints and frontend filtering to handle cases where product data may be missing or invalid. For example, cart items associated with unavailable or null product records are filtered out during rendering on the cart and checkout pages.

More comprehensive product existence checks at the cart API level are not yet implemented and are considered as future enhancements to further strengthen data integrity.

4. DEPLOYMENT AND SYSTEM CONFIGURATION

This section describes the deployment setup and system configuration of the Shople e-commerce platform based on the actual project repository and external service configurations. The focus is placed on components that are verifiable through source code and environment configuration, while clearly distinguishing assumptions that depend on external platform settings.

4.1. Deployment Architecture Overview

The Shople application follows a cloud-based deployment architecture that separates frontend hosting, backend data services, and domain management. The frontend is implemented using Next.js and deployed on Vercel, while backend data persistence is handled by a PostgreSQL database hosted on Supabase.

The application communicates with Supabase through server-side APIs using environment-based credentials. Deployment-related configurations are

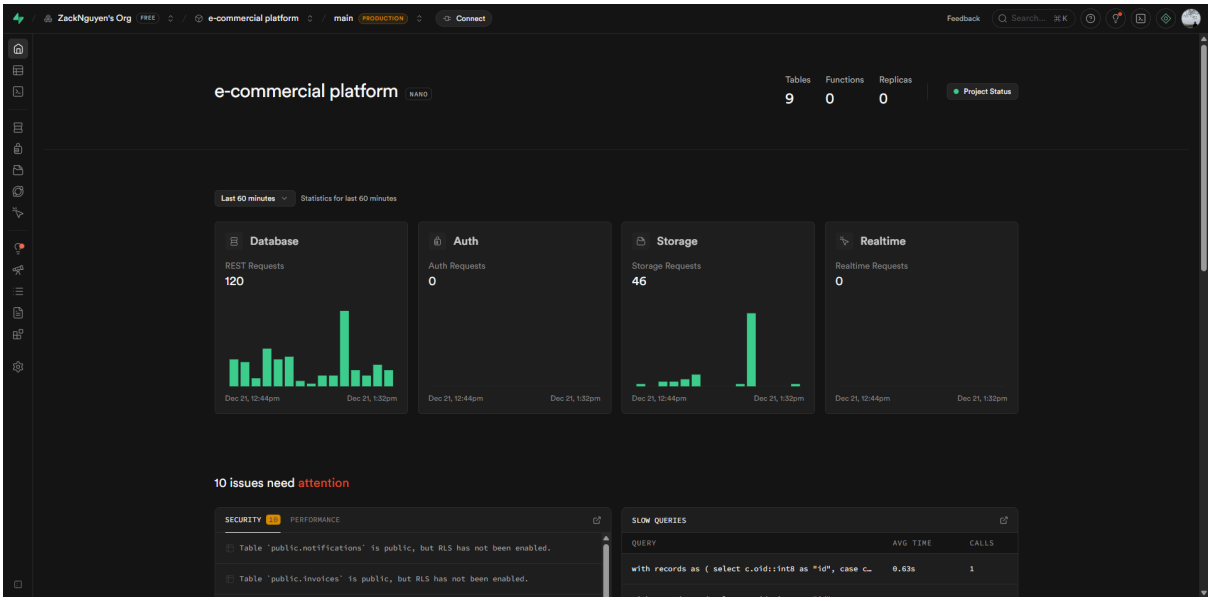
managed through platform dashboards and environment variables rather than hardcoded files, allowing the system to remain flexible across environments.

4.2. Database Deployment with Supabase

The application uses PostgreSQL hosted on Supabase as its primary database system. Database access is performed using the Supabase SDK, with credentials provided through environment variables such as SUPABASE_URL and SUPABASE_SERVICE_ROLE_KEY.

In the current implementation, database access is handled primarily on the server side using the service role key. Row Level Security (RLS) is not enabled in the existing schema, and role-based access control is implemented at the application logic level rather than relying on Supabase's built-in RLS policies.

The database schema includes tables supporting core entities such as users, shops, products, carts, cart items, and invoices. This setup is sufficient for the scope of the project and allows consistent data access while keeping the database configuration simple.

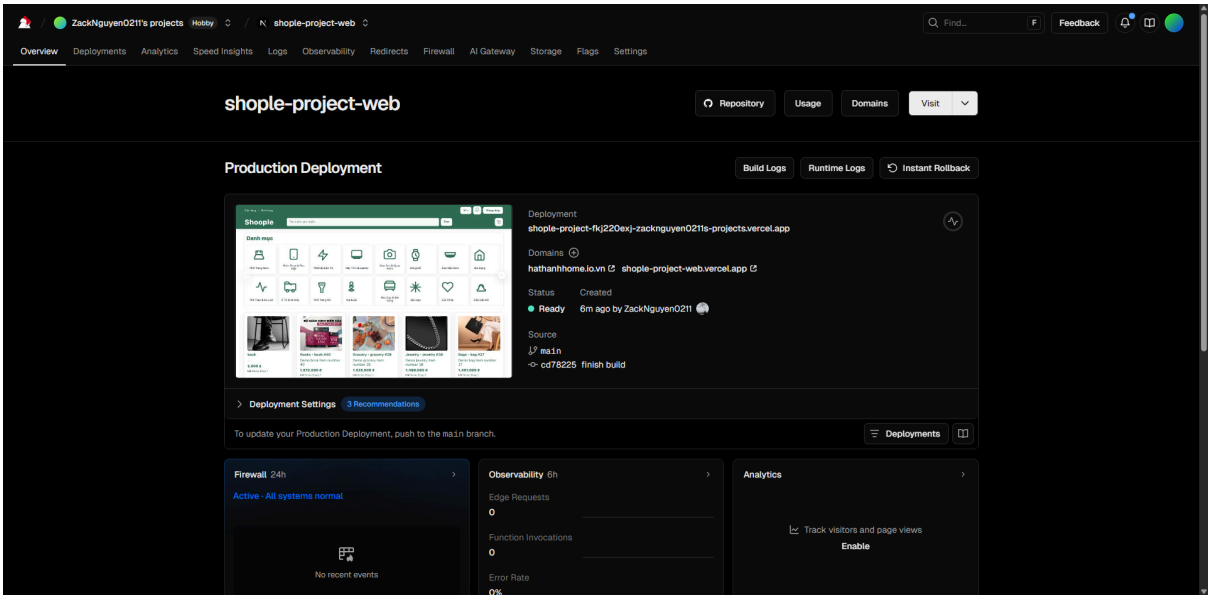


4.3. Application Deployment using Vercel

The frontend application is deployed using Vercel, which provides native support for Next.js applications. The repository contains configuration files and build scripts compatible with Vercel's deployment model.

Environment variables required for backend communication, such as Supabase credentials, are configured through the Vercel platform rather than being stored in the source code. While the project repository itself does not explicitly document the continuous deployment pipeline, the deployment workflow assumes that the application is connected to a GitHub repository via the Vercel dashboard, enabling automated builds upon code updates.

This approach simplifies deployment and allows the application to be updated without manual server management.



4.4. Custom Domain Configuration with Cloudflare

After deployment, the application is assigned a custom domain name, hathanhhome.io.vn, which is managed through Cloudflare. Cloudflare is used primarily for Domain Name System (DNS) configuration, allowing the domain to point to the deployed Vercel application.

This setup enables users to access the system through a personalized and professional domain rather than a default deployment URL. Cloudflare's DNS management provides reliable routing and supports future enhancements such as security and performance optimization.

4.5. Deployment Limitations and Considerations

While the current deployment setup is suitable for an academic project, several limitations are acknowledged. Some deployment-related aspects, such as continuous integration pipelines, monitoring, and automated scaling, are managed externally through platform dashboards and are not explicitly documented in the repository.

Additionally, certain backend features, including invoice email delivery and checkout validation, are implemented as demonstration components rather than fully production-ready services. These limitations align with the scope of the project and highlight potential areas for future improvement.

5. DISCUSSION AND CONCLUSION

This section summarizes the outcomes of the project, reflects on lessons learned during development, and discusses potential improvements based on the current implementation.

5.1. Achievements

The Shople project successfully demonstrates the development of a modern web-based e-commerce application using Next.js for the frontend and PostgreSQL hosted on Supabase for data persistence. The system implements core e-commerce features, including product browsing, shopping cart management, a checkout flow, and invoice persistence.

Product browsing and cart persistence are handled through server-side APIs, ensuring that user cart data is stored reliably across sessions. Invoice persistence is implemented as part of the server-side checkout success flow,

where order data is aggregated and upserted into the database, and cart items are cleared upon successful completion.

The project architecture separates frontend presentation, server-side logic, and database access, reflecting a clear understanding of modern web application structure. In addition, the project is designed to be deployed on cloud platforms such as Vercel, with external services available for custom domain configuration when applied.

5.2. Lessons Learned

One of the key lessons learned from this project is the importance of ensuring that documentation accurately reflects the actual implementation. Describing features strictly based on what is verifiable in code helps avoid ambiguity and strengthens the technical credibility of the report.

The project also highlights the benefits of server-side persistence for managing critical application state, such as shopping carts and invoices, compared to relying on purely client-side storage. Using managed services like Supabase and Vercel simplified infrastructure setup, while emphasizing the need for careful handling of environment variables and access credentials.

Additionally, the development process reinforced the importance of user-friendly error handling and edge-case management, particularly in scenarios such as empty carts, runtime errors, and incomplete data states.

5.3. Future Improvements

Although the current implementation satisfies the scope of the project, several improvements can be considered for future development. Database security can be enhanced by enabling Row Level Security (RLS) and defining more granular access policies once the application logic is further stabilized.

Other potential improvements include introducing inventory and stock tracking, strengthening backend validation during checkout, and replacing the

demonstration invoice email endpoint with a fully functional email delivery service. Monitoring, logging, and extended testing coverage could also be added to improve reliability and maintainability.

These enhancements would move the system closer to a production-ready e-commerce platform while building upon the existing foundation.

6. REFERENCES

- Next.js Documentation. https://nextjs.org/docs
- Supabase Documentation. https://supabase.com/docs
- PostgreSQL Documentation. https://www.postgresql.org/docs/
- Vercel Documentation. https://vercel.com/docs