**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC PHENIKAA**

**A logo for a university

Description automatically generated**

**BÁO CÁO ỨNG DỤNG PHÂN TÁN**

**Nhóm 11   
 DỰ ÁN QUẢN LÍ TIỆM TẠP HÓA**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSV** | **Email** |
| 1 | Nguyễn Thành Nhân | 23010011 | [23010011@st.phenikaa-uni.edu.vn](mailto:23010011@st.phenikaa-uni.edu.vn) |
| 2 | Hoàng Thị Hoa Mai | 22010320 | [22010320@st.phenikaa-uni.edu.vn](mailto:22010320@st.phenikaa-uni.edu.vn) |
| GVHD | ThS. Phạm Kim Thành | |  |

***Hà nội, tháng 05 năm 2025***

**MỤC LỤC**

[**I. Tổng Quan Dự Án** 1](#_Toc199382734)

[**II. Thiết Kế Kiến Trúc** 1](#_Toc199382735)

[**1. Mô Hình Kiến Trúc** 1](#_Toc199382736)

[**2. Công Nghệ Sử Dụng** 1](#_Toc199382737)

[**III. Phân Tích Chi Tiết Các Thành Phần** 2](#_Toc199382738)

[**1. Giao Diện Người Dùng (Frontend)** 2](#_Toc199382739)

[**1.1. Trang Quản Trị (Admin UI) - admin.html** 2](#_Toc199382740)

[**1.2. Trang Quản Lý Chi Nhánh (Branch Manager UI) - managerbranch.html** 3](#_Toc199382741)

[**1.3. Trang Bán Hàng Tại Chi Nhánh (Branch POS UI) - branch.html** 3](#_Toc199382742)

[**2. Dịch Vụ Backend (Microservices)** 4](#_Toc199382743)

[**2.1. Dịch Vụ Quản Lý Chi Nhánh (Branch Service)** 4](#_Toc199382744)

[**2.2. Dịch Vụ Quản Lý Kho Hàng (Warehouse Service)** 5](#_Toc199382745)

[**3. Cơ Sở Dữ Liệu** 6](#_Toc199382746)

[**3.1. PostgreSQL (postgres-db)** 6](#_Toc199382747)

[**3.2. Elasticsearch (es01, es02, es03)** 6](#_Toc199382748)

[**4. Triển Khai (Deployment) - docker-compose.yml** 7](#_Toc199382749)

[**IV. Các Sơ Đồ Chính Của Hệ Thống** 7](#_Toc199382750)

[**1. Sơ đồ Kiến trúc Hệ thống** 7](#_Toc199382751)

[**2. Sơ đồ Triển khai** 8](#_Toc199382752)

[**3. Sơ đồ Tuần tự: Use Case "Nhân viên Bán Sản Phẩm"** 8](#_Toc199382753)

[**3.1. Các yêu cầu chức năng** 8](#_Toc199382754)

[**3.2. Sơ đồ Use-case:** 9](#_Toc199382755)

[**3.3. Đặc tả các Use-case** 9](#_Toc199382756)

[**3.3.1. Quản lý sản phẩm** 9](#_Toc199382757)

[**3.3.2. Quản lý chi nhánh** 16](#_Toc199382758)

[**V. Phân Tích Vấn Đề và Các Rủi Ro Tiềm Ẩn** 21](#_Toc199382759)

[**1. Vấn Đề Hiện Tại** 21](#_Toc199382760)

[**2. Rủi Ro Tiềm Ẩn** 23](#_Toc199382761)

[**VI. Đề Xuất Giải Pháp và Hướng Phát Triển** 23](#_Toc199382762)

[**1. Giải Quyết Các Vấn Đề Hiện Tại** 23](#_Toc199382763)

[**2. Cải Thiện và Nâng Cấp** 24](#_Toc199382764)

[**3. Hướng Phát Triển Tương Lai** 25](#_Toc199382765)

[**VII. Đánh Giá Theo Tiêu Chí Kỹ Thuật và Đề Xuất** 25](#_Toc199382766)

[**1. Tiêu Chí Bắt Buộc** 25](#_Toc199382767)

[**1.1. Fault Tolerance (Khả năng chịu lỗi):** 25](#_Toc199382768)

[**1.2. Distributed Communication (Giao tiếp Phân tán):** 26](#_Toc199382769)

[**1.3. Sharding hoặc Replication (Phân mảnh hoặc Sao chép Dữ liệu):** 27](#_Toc199382770)

[**1.4. Simple Monitoring / Logging (Giám sát / Ghi nhật ký đơn giản):** 27](#_Toc199382771)

[**1.5. Basic Stress Test (Kiểm tra tải cơ bản):** 29](#_Toc199382772)

[**2. Tiêu Chí Tùy Chọn (Chọn 1–2)** 29](#_Toc199382773)

[**1. Deployment Automation (Tự động hóa Triển khai):** 29](#_Toc199382774)

[**2. Security Features (Tính năng Bảo mật):** 30](#_Toc199382775)

[**VIII. Kết Luận** 30](#_Toc199382776)

# **I. Tổng Quan Dự Án**

Dự án "Quản Lý Tiệm Tạp Hóa" được xây dựng nhằm mục đích cung cấp một hệ thống toàn diện cho việc quản lý hoạt động của một chuỗi cửa hàng tạp hóa. Hệ thống bao gồm các chức năng quản lý chi nhánh, quản lý sản phẩm, quản lý tồn kho tại từng chi nhánh, và giao diện bán hàng (POS) cho nhân viên tại điểm bán.Trong dự án có tích hợp phần tán gồm 2 kiểu phân tán theo kiểu kiến trúc và kiểu dữ liệu.

* Kiến trúc thực hiện build với mỗi container là một dịch vụ riêng biệt được mô phỏng hoá trên docker.
* Và sự phân tán thứ hai là sự phân tán dữ liệu trong các cụm elacticsearch các node được thiết lập để có thể phân mảnh và sao chép dữ liệu.

# **II. Thiết Kế Kiến Trúc**

## **1. Mô Hình Kiến Trúc**

Hệ thống được thiết kế theo kiến trúc microservices, bao gồm các dịch vụ độc lập giao tiếp với nhau qua API:

* **Giao Diện Người Dùng (Frontend):** Ba ứng dụng frontend riêng biệt được xây dựng bằng HTML, CSS và JavaScript thuần túy:
  + Admin UI: Cho quản trị viên hệ thống.
  + Branch Manager UI: Cho người quản lý từng chi nhánh.
  + Branch POS UI: Giao diện bán hàng cho nhân viên tại chi nhánh.
* **Dịch Vụ Backend (Backend Services):**
  + branch-service: Quản lý thông tin chi nhánh.
  + warehouse-service: Quản lý thông tin sản phẩm và tồn kho.
* **Cơ Sở Dữ Liệu (Databases):**
  + PostgreSQL: Lưu trữ dữ liệu có cấu trúc cho branch-service.
  + Elasticsearch: Lưu trữ và tìm kiếm dữ liệu sản phẩm cho warehouse-service.
* **Triển Khai (Deployment):**
  + Tất cả các thành phần được đóng gói và triển khai dưới dạng container Docker, được quản lý bởi docker-compose.
  + Một mạng ảo (tiemtaphoa-network) được tạo để các container có thể giao tiếp với nhau.

## **2. Công Nghệ Sử Dụng**

* **Backend:**
  + Java 17
  + Spring Boot 3.4.5 (bao gồm Spring Web, Spring Data JPA, Spring Data Elasticsearch)
  + Lombok
* **Frontend:**
  + HTML
  + CSS
  + JavaScript (Fetch API)
* **Cơ Sở Dữ Liệu:**
  + PostgreSQL 15-alpine
  + Elasticsearch 8.18.1 (cụm 3 node: es01, es02, es03)
* **Triển Khai & Môi Trường:**
  + Docker & Docker Compose
* **Build Tool (Backend):**
  + Apache Maven

# **III. Phân Tích Chi Tiết Các Thành Phần**

## **1. Giao Diện Người Dùng (Frontend)**

Cả ba giao diện người dùng đều được xây dựng dưới dạng các trang HTML tĩnh, sử dụng JavaScript để tương tác với các API backend.

### **1.1. Trang Quản Trị (Admin UI) - admin.html**

* **Chức năng:**
  + Thêm sản phẩm mới: Nhập Mã SP, Tên SP, Mô tả, Giá Gốc.
  + Quản lý tồn kho ban đầu cho sản phẩm tại các chi nhánh: Cho phép thêm thông tin tồn kho (Mã Chi Nhánh, Số Lượng, Giá Bán) cho nhiều chi nhánh khi tạo hoặc cập nhật sản phẩm.
  + Xem danh sách tất cả sản phẩm: Hiển thị Mã SP, Tên SP, Giá Gốc, và chi tiết tồn kho tại từng chi nhánh (Mã CN, Số Lượng, Giá Bán).
  + Sửa thông tin sản phẩm: Chỉnh sửa các thông tin cơ bản và thông tin tồn kho tại các chi nhánh.
  + Xóa sản phẩm.
* **Tương tác API:** Gọi đến các endpoints của warehouse-service (mặc định tại http://localhost:8080/api/warehouse/products).
  + Đoạn code JavaScript gọi API thêm sản phẩm mới:
* // là đường dẫn api trả về sản phẩm
* const WAREHOUSE\_API\_BASE = 'http://localhost:8080/api/warehouse/products';
* // lắng nghe sự kiên thêm sản phẩm từ nút
* document.getElementById('addProductForm').addEventListener('submit', async function(event) {
* event.preventDefault();
* //Ngăn chặn hành vi tải lại trang
* const responseMessageDiv = document.getElementById('responseMessage');
* // lấy thể div để hiện thông tin phản hồi
* setResponseMessage(responseMessageDiv, '', '');
* // xoá thông tin cũ nếu có
* // thu thập dữ liệu từ các trường input trong form
* const product = {
* productId: document.getElementById('productId').value,
* name: document.getElementById('name').value,
* description: document.getElementById('description').value,
* basePrice: parseFloat(document.getElementById('basePrice').value),
* category: document.getElementById('category').value,
* branchesInventory: getBranchInventoriesFromFields('addBranchesContainer', 'add\_')
* // gọi hàm để lấy thông tin tồn kho tại chi nhánh
* };
* //in ra log dữ liệu gửi xem có lỗi không
* console.log('Sending product data:', JSON.stringify(product));
* // thực hiện gọi api bằng fetch
* try {
* const response = await fetch(WAREHOUSE\_API\_BASE, {
* method: 'POST',// phương thức http
* headers: { 'Content-Type': 'application/json' },// định dạng dữ liệu kiểu Json
* body: JSON.stringify(product)// chuyển đổi product thành chuỗi dữ liệu đẻ gửi qua
* });
* // các xử lý phản hồi từ api
* if (response.ok) {
* // in ra các thông báo thành công
* const createdProduct = await response.json();
* setResponseMessage(responseMessageDiv, 'Sản phẩm đã được thêm thành công! ID nội bộ: ' + createdProduct.id, 'success');
* // xoá các giá trị đã thông báo từ form
* document.getElementById('addProductForm').reset();
* document.getElementById('addBranchesContainer').innerHTML = createBranchInventoryFieldsHTML(0, 'add\_'); *// Reset về 1 mục*
* fetchAllProductsAdmin(); *// Tải lại danh sách sản phẩm*
* } else {// xử lý nếu lỗi
* const errorText = await response.text();
* setResponseMessage(responseMessageDiv, `Lỗi khi thêm sản phẩm: ${response.status} - ${errorText || response.statusText}`, 'error');
* console.error('Error data:', errorText);
* }
* } catch (error) {
* setResponseMessage(responseMessageDiv, 'Lỗi kết nối hoặc lỗi JavaScript: ' + error.message, 'error');
* console.error('Fetch error:', error);
* }
* });
* Và bao gồm các hàm khác hỗ trợ các công việc in lỗi lấy thông tin sản phẩm …
  + Hàm phụ trợ để hiển thị thông báo (được giả định là đã định nghĩa ở nơi khác trong file)
    - Function setResponseMessage(element, text, type) { ... }
  + Hàm phụ trợ để lấy thông tin tồn kho từ các trường động (được giả định là đã định nghĩa)
    - Function getBranchInventoriesFromFields(containerId, prefix) { ... }
  + Hàm phụ trợ để tạo các trường input cho thông tin tồn kho (được giả định là đã định nghĩa)
    - function createBranchInventoryFieldsHTML(index, prefix, branch = {branchId: '', quantity: '', sellingPrice: ''}) { ... }
  + Hàm phụ trợ để tải lại danh sách sản phẩm (được giả định là đã định nghĩa)
    - async function fetchAllProductsAdmin() { ... }

### **1.2. Trang Quản Lý Chi Nhánh (Branch Manager UI) - managerbranch.html**

* **Chức năng:**
  + Thêm chi nhánh mới: Nhập Mã Chi Nhánh, Tên Chi Nhánh, Địa Chỉ, Thành Phố, Số Điện Thoại, Trạng Thái Hoạt Động.
  + Xem danh sách chi nhánh: Hiển thị ID CSDL, Mã CN, Tên CN, Địa Chỉ, Thành Phố, SĐT, Trạng Thái.
  + Sửa thông tin chi nhánh.
  + Xóa chi nhánh.
* **Tương tác API:** Gọi đến các endpoints của branch-service (mặc định tại http://localhost:8081/api/branches).
  + Đoạn code JavaScript gọi API thêm/cập nhật chi nhánh:
* const apiUrl = 'http://localhost:8081/api/branches';
* // các bước lấy tham số để nhằm hiện thông tin hoặc báo lỗi
* const branchForm = document.getElementById('branchForm');
* const formTitle = document.getElementById('formTitle');
* const submitButton = document.getElementById('submitButton');
* const branchIdField = document.getElementById('branchId');
* const formResponseMessage = document.getElementById('formResponseMessage');
* const responseMessageDiv = document.getElementById('responseMessage');
* *// Xử lý submit form (Thêm mới hoặc Cập nhật)*
* branchForm.addEventListener('submit', async function (event) {
* // ngăn chặn reload lại trang
* event.preventDefault();
* // xoá các thông tin khi vừa ấn nút
* formResponseMessage.textContent = '';
* formResponseMessage.className = '';
* // thu thập dữ liệu từ các trường input
* const branchData = {
* branchCode: document.getElementById('branchCode').value,
* name: document.getElementById('name').value,
* address: document.getElementById('address').value,
* city: document.getElementById('city').value,
* phoneNumber: document.getElementById('phoneNumber').value,
* active: document.getElementById('active').value === 'true'
* };
* let url = apiUrl;
* let method = 'POST';
* const id = branchIdField.value;
* if (id) { *// Nếu có ID -> Cập nhật*
* url = `${apiUrl}/${id}`;
* method = 'PUT';
* branchData.id = parseInt(id); *// Gửi kèm id nếu API yêu cầu (thường không cần cho body)*
* }
* // lại thực hiện gọi api bằng fetch
* try {
* const response = await fetch(url, {
* method: method,
* headers: {
* 'Content-Type': 'application/json'
* },
* body: JSON.stringify(branchData)
* });
* if (response.ok) {
* const resultText = id ? 'cập nhật' : 'thêm mới';
* showMessage(formResponseMessage, `Chi nhánh đã được ${resultText} thành công!`, true);
* document.getElementById('clearFormButton').click(); *// Xóa form*
* fetchBranches(); *// Tải lại danh sách*
* } else {
* const errorResult = await response.text();
* showMessage(formResponseMessage, `Lỗi: ${response.status} - ${errorResult || response.statusText}`, false);
* }
* } catch (error) {
* showMessage(formResponseMessage, `Lỗi kết nối: ${error.message}`, false);
* }
* });
* Và các hàm khác hiện lỗi và hỗ trợ tải lại danh sách chi nhánh như:
  + Hàm phụ trợ để hiển thị thông báo (được giả định là đã định nghĩa ở nơi khác trong file)
    - function showMessage(element, message, isSuccess) { ... }
  + Hàm phụ trợ để tải lại danh sách chi nhánh (được giả định là đã định nghĩa)
    - async function fetchBranches() { ... }
* **Người dùng:** Người quản lý chi nhánh.

### **1.3. Trang Bán Hàng Tại Chi Nhánh (Branch POS UI) - branch.html**

* **Chức năng:**
  + Hiển thị thông tin chi nhánh hiện tại (Mã CN, Tên, Địa chỉ, Trạng thái) dựa trên mã chi nhánh lấy từ URL (/pos/{branchCode}).
  + Kiểm tra trạng thái hoạt động của chi nhánh trước khi cho phép thao tác.
  + Hiển thị danh sách sản phẩm có bán tại chi nhánh: Mã SP, Tên SP, Giá Bán, Tồn Kho.
  + Tìm kiếm/lọc sản phẩm theo Mã SP hoặc Tên SP.
  + Thực hiện thao tác "Bán": Nhập số lượng cần bán, hệ thống sẽ gọi API để giảm tồn kho.
* **Tương tác API:**
  + Gọi branch-service để lấy thông tin và trạng thái chi nhánh.
  + Gọi warehouse-service để lấy danh sách sản phẩm của chi nhánh (/api/warehouse/products/branch/{branchId}) và để thực hiện giảm tồn kho (/api/warehouse/products/inventory/decrement).

1. Lấy danh sách sản phẩm cho chi nhánh (fetchProductsForBranch):

const warehouseApiBase = 'http://localhost:8080/api/warehouse/products';

*// Tải danh sách sản phẩm cho chi nhánh*

async function fetchProductsForBranch(branchCode) {

// hiện thi thông báo tìm thấy chi nhánh và đang tải

showStatusMessage('Đang tải sản phẩm của chi nhánh...', 'info');

try {

//gọi api lấy sản phẩm chi nhánh hiện tại

const response = await fetch(`${warehouseApiBase}/branch/${branchCode}`);

// xử lý phản hổi từ api

if (!response.ok) {

if (response.status === 204 || response.status === 404) {

branchProducts = [];

renderProductsTable();

showStatusMessage(`Chi nhánh ${branchCode} không có sản phẩm nào.`, 'info');

return;

}

throw new Error(`Lỗi HTTP khi tải sản phẩm: ${response.status}`);

}

branchProducts = await response.json();

renderProductsTable();

showStatusMessage('Tải sản phẩm thành công!', 'success');

} catch (error) {

showStatusMessage(`Lỗi khi tải sản phẩm: ${error.message}`, 'error');

console.error("Error fetching products for branch:", error);

branchProducts = [];

renderProductsTable();

}

}

* Và các hàm phụ tợ hiện thông báo như :
* Hàm phụ trợ hiển thị thông báo (giả định đã định nghĩa)
  + function showStatusMessage(text, type = 'info') { ... }
* Hàm phụ trợ vẽ lại bảng sản phẩm (giả định đã định nghĩa)
  + Function renderProductsTable(productsToRender = branchProducts) { ... }

1. Bán sản phẩm - Giảm tồn kho (sellProduct):

*// "Bán" sản phẩm (giảm tồn kho)*

async function sellProduct(productId, branchId, quantityToDecrement) { *// quantityToDecrement giờ sẽ là số lượng người dùng nhập*

*// Xác nhận người dùng*

if (!confirm(`Xác nhận bán ${quantityToDecrement} sản phẩm "${productId}" tại chi nhánh ${branchId}?`)) {

return; // nếu không đồng ý thì ,thì không làm gì cả

}

showStatusMessage('Đang xử lý yêu cầu giảm tồn kho...', 'info');

try {

// gọi api để giảm tồn kho

const response = await fetch(`${warehouseApiBase}/inventory/decrement`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ productId, branchId, quantityToDecrement })

});

if (response.ok) {

showStatusMessage(`Đã bán ${quantityToDecrement} sản phẩm "${productId}". Tồn kho đã được cập nhật.`, 'success');

*// Tải lại danh sách sản phẩm để cập nhật số lượng*

fetchProductsForBranch(currentBranchCode);

} else {

const errorData = await response.json(); *// Cố gắng parse lỗi JSON từ backend*

let errorMessage = errorData.message || JSON.stringify(errorData); *// Lấy message nếu có, hoặc toàn bộ lỗi*

if (response.status === 400 && errorData.message && errorData.message.startsWith("Not enough stock")) {

showStatusMessage(`Lỗi khi bán sản phẩm: ${errorData.message}`, 'error');

} else {

showStatusMessage(`Lỗi khi bán sản phẩm: ${response.status} - ${errorMessage}`, 'error');

}

console.error("Error selling product:", errorData);

}

} catch (error) {

showStatusMessage(`Lỗi kết nối khi bán sản phẩm: ${error.message}`, 'error');

console.error("Error selling product:", error);

}

}

* **Người dùng:** Nhân viên bán hàng tại chi nhánh.

## **2. Dịch Vụ Backend (Microservices)**

### **2.1. Dịch Vụ Quản Lý Chi Nhánh (Branch Service)**

* **Mô tả:** Chịu trách nhiệm quản lý thông tin về các chi nhánh của cửa hàng.
* **Công nghệ:** Spring Boot, Spring Data JPA, PostgreSQL.
* **Cấu hình:** Chạy trên cổng 8081. Kết nối tới PostgreSQL tại postgres-db:5432/branchdb.
* **Model (Branch.java):**
  + id: Long (PK, tự tăng)
  + branchCode: String (Unique, Not null)
  + name: String (Not null)
  + address: String
  + city: String
  + phoneNumber: String
  + active: boolean (Not null, default true)
* **Repository (BranchRepository.java):** Cung cấp các phương thức CRUD cơ bản và các truy vấn tùy chỉnh như findByBranchCode, existsByBranchCode, findByBranchCodeAndActiveTrue, findAllByActiveTrue.
* **Service (BranchService.java):** Xử lý logic nghiệp vụ, ví dụ kiểm tra branchCode đã tồn tại trước khi tạo mới hoặc cập nhật.

package com.tiemtaphoa.branchservice.service;

import com.tiemtaphoa.branchservice.model.Branch;

import com.tiemtaphoa.branchservice.repository.BranchRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@Service

public class BranchService {

private final BranchRepository branchRepository;

// khai báo đối tượng log

private static final Logger logger = LoggerFactory.getLogger(BranchService.class);

@Autowired

public BranchService(BranchRepository branchRepository) {

this.branchRepository = branchRepository;

}

*// kiểm tra chi nhánh đã tồn tại hay chưa*

public Branch createBranch(Branch branch) {

if (branchRepository.existsByBranchCode(branch.getBranchCode())) {

throw new IllegalArgumentException("Mã chi nhánh " + branch.getBranchCode() + " đã tồn tại.");

}

return branchRepository.save(branch);

}

*// trả về địa chỉ chi nhánh giựa theo id*

public Optional<Branch> getBranchById(Long id) {

logger.info("SERVICE: Tìm kiếm chi nhánh với ID: {}", id);

Optional<Branch> branch = branchRepository.findById(id);

if (branch.isPresent()) {

logger.info("SERVICE: Tìm thấy chi nhánh: {} với ID: {}", branch.get().getName(), id);

} else {

logger.warn("SERVICE: Không tìm thấy chi nhánh với ID: {}", id);

}

return branch;

}

Và các phương thức hỗ trợ lấy thông tin khác hỗ trợ lấy truy xuất sản phẩm của chi nhánh hoặc kiểm tra chi nhánh

* **Controller (BranchController.java):** Cung cấp các REST APIs cho việc quản lý chi nhánh qua đường dẫn /api/branches. Cho phép CORS từ mọi nguồn (\*).
  + POST /: Tạo chi nhánh mới.
  + GET /: Lấy tất cả chi nhánh (có thể lọc theo trạng thái active).
  + GET /{id}: Lấy chi nhánh theo ID CSDL.
  + GET /by-code/{branchCode}: Lấy chi nhánh theo mã chi nhánh (có thể lọc chỉ lấy chi nhánh active).
  + PUT /{id}: Cập nhật thông tin chi nhánh.
  + DELETE /{id}: Xóa chi nhánh.

package com.tiemtaphoa.branchservice.controller;

import com.tiemtaphoa.branchservice.model.Branch;

import com.tiemtaphoa.branchservice.service.BranchService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.util.List;

import java.util.Optional;

@RestController *// @Controller and @ResponseBody*

@RequestMapping("/api/branches") *// tất cả request đến /api/branches/\*\* đều được xử lý trong controller này*

@CrossOrigin(origins = "\*") *//Cho phép chấp nhận các request từ domain khác*

public class BranchController {

private static final Logger logger = LoggerFactory.getLogger(BranchController.class);

private final BranchService branchService;

@Autowired

public BranchController(BranchService branchService) {

this.branchService = branchService;

}

@PostMapping

public ResponseEntity<?> createBranch(@RequestBody Branch branch) {

logger.info("Nhận yêu cầu tạo chi nhánh với mã: {} ", branch.getBranchCode());

try {

Branch createdBranch = branchService.createBranch(branch);

logger.info("Chi nhánh {} đã được tạo thành công với ID: {}", createdBranch.getBranchCode(), createdBranch.getId());

return new ResponseEntity<>(createdBranch, HttpStatus.CREATED);

} catch (IllegalArgumentException e) {

logger.error("Lỗi khi tạo chi nhánh với mã: {}. Lỗi: {}", branch.getBranchCode(), e.getMessage(), e);

return ResponseEntity.badRequest().body(e.getMessage());

}

}

Và các post khác hỗ trợ các phương thức huỷ ,tạo, sửa…

}

- POST /api/branches nhận dữ liệu chi nhánh mới từ client dưới dạng JSON, ủy thác cho BranchService để xử lý logic và lưu trữ. Nếu thành công, nó trả về thông tin chi nhánh vừa tạo cùng status 201 Created. Nếu có lỗi nghiệp vụ (như mã chi nhánh trùng), nó trả về status 400 Bad Request kèm thông điệp lỗi.

### **2.2. Dịch Vụ Quản Lý Kho Hàng (Warehouse Service)**

* **Mô tả:** Chịu trách nhiệm quản lý thông tin sản phẩm và tồn kho của từng sản phẩm tại các chi nhánh.
* **Công nghệ:** Spring Boot, Spring Data Elasticsearch.
* **Cấu hình:** Chạy trên cổng 8080. Kết nối tới cụm Elasticsearch (es01, es02, es03) tại cổng 9200. (Cấu hình trong docker-compose.yml sẽ ghi đè application.properties về URI của Elasticsearch).
* **Model (Product.java, BranchInventory.java):**
  + Product (Document trong index warehouse\_products):
    - id: String (ID document của Elasticsearch, tự sinh UUID nếu không có)
    - productId: String (Mã sản phẩm tùy chỉnh, ví dụ: SP001)
    - name: String (Tên sản phẩm)
    - description: String (Mô tả)
    - basePrice: double (Giá gốc)
    - category: String (Danh mục)
    - branchesInventory: List<BranchInventory> (Nested document, lưu thông tin tồn kho tại từng chi nhánh)
  + BranchInventory (POJO):
    - branchId: String (Mã chi nhánh)
    - quantity: int (Số lượng tồn)
    - sellingPrice: double (Giá bán tại chi nhánh)
* **Repository (ProductRepository.java):** Cung cấp các phương thức CRUD cho document Product và các truy vấn tùy chỉnh như findByProductId và findByBranchIdInInventory (sử dụng @Query với nested query).

package com.tiemtaphoa.warehouseservice.repository;

import com.tiemtaphoa.warehouseservice.model.Product;

import org.springframework.data.elasticsearch.annotations.Query;

import org.springframework.data.elasticsearch.repository.ElasticsearchRepository;

import java.util.List;

import java.util.Optional;

public interface ProductRepository extends ElasticsearchRepository<Product, String> {

*// ElasticsearchRepository<Product, String> nghĩa là repository này làm việc với*

*// document Product và kiểu dữ liệu của ID là String.*

Optional<Product> findByProductId(String productId);

@Query("{\"nested\": {\"path\": \"branchesInventory\", \"query\": {\"term\": {\"branchesInventory.branchId.keyword\": \"?0\"}}}}")

List<Product> findByBranchIdInInventory(String branchId);

}

* Trong ProductRepository có hai phường thức chính để tìm kiếm sản phẩm :
  + findByProductId(String productId)
    - **Mục đích đơn giản:** Tìm một sản phẩm dựa trên **mã sản phẩm cụ thể** .
    - **Cách hoạt động :** Spring Data Elasticsearch sẽ tự động tạo truy vấn để tìm chính xác sản phẩm có mã đó.
  + findByBranchIdInInventory(String branchId) (với @Query tùy chỉnh):
    - **Mục đích đơn giản:** Tìm tất cả các sản phẩm hiện **đang có bán tại một chi nhánh cụ thể**.
    - Cách hoạt động :
      * Chúng ta sử dụng một truy vấn Elasticsearch đặc biệt gọi là **'nested query'**.
      * Truy vấn này sẽ 'nhìn sâu' vào bên trong thông tin tồn kho của mỗi sản phẩm.
      * Nó tìm những sản phẩm nào có mục tồn kho khớp với **mã chi nhánh** chúng ta đưa vào.
      * Trong đó tuỳ chỉnh truy vấn sâu điều hướng tìm kiếm @Query khá phức tạp có thêm tìm các doc hỗ trợ thư viện elaticsearch để tìm hiểu thêm.
* **Service (ProductService.java):** Xử lý logic nghiệp vụ:
  + Tạo/cập nhật/xóa sản phẩm.
  + Truy vấn sản phẩm theo ID, mã sản phẩm, hoặc theo chi nhánh.
  + decrementInventory: Giảm số lượng tồn kho của sản phẩm tại một chi nhánh cụ thể, kiểm tra số lượng đủ trước khi thực hiện.

public static class DecrementInventoryRequest {

private String productId; *// productId của sản phẩm*

private String branchId; *// branchId của chi nhánh*

private int quantityToDecrement;

*// Getters and Setters*

public String getProductId() { return productId; }

public void setProductId(String productId) { this.productId = productId; }

public String getBranchId() { return branchId; }

public void setBranchId(String branchId) { this.branchId = branchId; }

public int getQuantityToDecrement() { return quantityToDecrement; }

public void setQuantityToDecrement(int quantityToDecrement) { this.quantityToDecrement = quantityToDecrement; }

}

//các phương thức hỗ trợ khác

@Transactional

public Product decrementInventory(String productId, String branchId, int quantityToDecrement) {

logger.info("Đang giảm tồn kho cho sản phẩm ID: {}, chi nhánh ID: {}, số lượng: {}", productId, branchId, quantityToDecrement);

Product product = productRepository.findByProductId(productId)

.orElseThrow(() -> {

logger.error("Không tìm thấy sản phẩm với productId: {}", productId);

return new IllegalArgumentException("Không tìm thấy sản phẩm với id: " + productId);

});

logger.info("Tìm sản phẩm: {}. Xem sản phẩm của chi nhánh: {}", product.getProductId(), branchId);

if (product.getBranchesInventory() != null) {

product.getBranchesInventory().forEach(inv -> logger.info("Tồn kho hiện có của sản phẩm {}: chi nhánh ID={}, số lượng={}", product.getProductId(), inv.getBranchId(), inv.getQuantity()));

} else {

logger.warn("Sản phẩm {} có danh sách tồn kho chi nhánh là null.", product.getProductId());

}

BranchInventory inventoryToUpdate = product.getBranchesInventory().stream()

.filter(inv -> {

boolean match = branchId.equals(inv.getBranchId()); *// So sánh branchId nhận từ request với branchId trong inventory*

if (!match) {

logger.info("Tồn kho của chi nhánh ID '{}' trong sản phẩm {} không khớp với chi nhánh ID được yêu cầu '{}'", inv.getBranchId(), productId, branchId);

}

return match;

})

.findFirst()

.orElseThrow(() -> {

logger.error("Không tìm thấy chi nhánh: {} và id sản phẩm: {}", branchId, productId);

return new IllegalArgumentException("không tìm thấy chi nhánh: " + branchId + " và id sản phẩm: " + productId);

});

logger.info("Đã tìm và cập nhật sản phẩm {} tại chi nhánh {}: số lượng={}", productId, branchId, inventoryToUpdate.getQuantity());

if (inventoryToUpdate.getQuantity() < quantityToDecrement) {

String errorMessage = "Không đủ hàng cho sản phẩm " + productId + " tại chi nhánh " + branchId + ". Hiện có: " + inventoryToUpdate.getQuantity() + ", Yêu cầu: " + quantityToDecrement;

logger.error(errorMessage);

throw new IllegalArgumentException(errorMessage);

}

inventoryToUpdate.setQuantity(inventoryToUpdate.getQuantity() - quantityToDecrement);

logger.info("Sản phẩm {} tại chi nhánh {} có số lượng mới sau khi giảm: {}", productId, branchId, inventoryToUpdate.getQuantity());

return productRepository.save(product);

}

* trong đó decrementInventory thực hiện một chuỗi các bước: tìm sản phẩm, tìm thông tin tồn kho của sản phẩm đó tại chi nhánh cụ thể, kiểm tra xem có đủ hàng để bán không, và cuối cùng là cập nhật số lượng tồn kho nếu tất cả các điều kiện đều thỏa mãn. Nó sử dụng exceptions để xử lý các trường hợp lỗi nghiệp vụ và ghi log để theo dõi.
* **Controller (WarehouseController.java):** Cung cấp REST APIs qua /api/warehouse/products. Cho phép CORS từ các frontend UI cụ thể (ports 8000, 8002, 8003).
  + POST /: Thêm sản phẩm mới.
  + GET /by-product-id/{productId}: Lấy sản phẩm theo mã sản phẩm.
  + GET /{internalId}: Lấy sản phẩm theo ID document Elasticsearch.
  + GET /: Lấy tất cả sản phẩm.
  + PUT /{internalId}: Cập nhật sản phẩm.
  + DELETE /{internalId}: Xóa sản phẩm.
  + GET /branch/{branchId}: Lấy danh sách sản phẩm và thông tin tồn kho (số lượng, giá bán) của một chi nhánh cụ thể.
  + POST /inventory/decrement: Giảm số lượng tồn kho của một sản phẩm tại một chi nhánh.

package com.tiemtaphoa.warehouseservice.controller;

import com.tiemtaphoa.warehouseservice.model.Product;

import com.tiemtaphoa.warehouseservice.service.ProductService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*; *// Đảm bảo import này*

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@RestController

@RequestMapping("/api/warehouse/products")

@CrossOrigin(

origins = {"http://localhost:8000", "http://localhost:8002", "http://localhost:8003"},

methods = {RequestMethod.GET, RequestMethod.POST, RequestMethod.PUT, RequestMethod.DELETE, RequestMethod.OPTIONS}, *// Cho phép các phương thức này*

allowedHeaders = "\*" *// Cho phép tất cả các header (hoặc chỉ định cụ thể nếu muốn chặt chẽ hơn)*

)

public class WarehouseController {

private final ProductService productService;

private static final Logger logger = LoggerFactory.getLogger(WarehouseController.class);

@Autowired

public WarehouseController(ProductService productService) {

this.productService = productService;

}

// các phương thức lấy sản phẩm ,xoá sản phẩm …

@GetMapping("/branch/{branchId}")

public ResponseEntity<List<Product>> getProductsByBranchId(@PathVariable String branchId) {

List<Product> products = productService.getProductsByBranch(branchId); *// Bạn sẽ cần tạo phương thức này trong ProductService*

if (products.isEmpty()) {

*// Quyết định trả về danh sách rỗng (200 OK) hay 204 No Content*

*// Trả về danh sách rỗng thường tốt hơn cho client dễ xử lý*

return ResponseEntity.ok(new ArrayList<>());

}

return ResponseEntity.ok(products);

}

}

- GET /api/warehouse/products/branch/{branchId} nhận một mã chi nhánh từ URL, sử dụng ProductService để lấy tất cả các sản phẩm liên quan đến chi nhánh đó (chỉ với thông tin tồn kho của chi nhánh đó), và trả về danh sách này dưới dạng JSON cho client. Nếu không có sản phẩm nào, nó vẫn trả về một danh sách rỗng với status 200 OK.

## **3. Cơ Sở Dữ Liệu**

### **3.1. PostgreSQL (postgres-db)**

* **Mục đích:** Lưu trữ dữ liệu quan hệ cho branch-service.
* **Dữ liệu:** Thông tin chi tiết về các chi nhánh (mã, tên, địa chỉ, v.v.).
* **Cấu hình:** POSTGRES\_USER: adminuser, POSTGRES\_PASSWORD: adminpass, POSTGRES\_DB: branchdb. Dữ liệu được lưu trữ bền vững thông qua volume postgres\_data\_compose.

### **3.2. Elasticsearch (es01, es02, es03)**

* **Mục đích:** Lưu trữ, tìm kiếm và phân tích dữ liệu sản phẩm cho warehouse-service. Phù hợp cho việc tìm kiếm nhanh và các truy vấn phức tạp trên dữ liệu sản phẩm và tồn kho.
* **Dữ liệu:** Thông tin chi tiết về sản phẩm, bao gồm cả thông tin tồn kho lồng nhau (nested) tại các chi nhánh.
* **Cấu hình:** Một cụm Elasticsearch gồm 3 node được thiết lập để đảm bảo tính sẵn sàng cao và khả năng chịu lỗi. Xpack security và SSL được tắt (xpack.security.enabled=false). Dữ liệu được lưu trữ bền vững qua các volume esdata01, esdata02, esdata03. bootstrap.memory\_lock=true được bật để tăng hiệu năng.

## **4. Triển Khai (Deployment) - docker-compose.yml**

* Hệ thống sử dụng Docker Compose để định nghĩa và chạy ứng dụng đa container.
* **Services:**
  + postgres-db: Container cho cơ sở dữ liệu PostgreSQL.
  + es01, es02, es03: Ba container cho cụm Elasticsearch.
  + branch-service: Container cho dịch vụ quản lý chi nhánh, build từ Dockerfile trong ./branch-service.
  + warehouse-service: Container cho dịch vụ quản lý kho, build từ Dockerfile trong ./warehouse-service.
  + admin-ui: Container cho giao diện quản trị, build từ Dockerfile trong ./font\_admin.
  + branch-manager-ui: Container cho giao diện quản lý chi nhánh, build từ Dockerfile trong ./font\_manager\_branch.
  + branch-pos-ui: Container cho giao diện POS, build từ Dockerfile trong ./font\_branch.
* **Networking:** Tất cả các service được kết nối vào một bridge network tên là tiemtaphoa-network, cho phép giao tiếp nội bộ giữa các container bằng tên service.
* **Healthchecks:** Được định nghĩa cho postgres-db, các node Elasticsearch, và branch-service để đảm bảo các service phụ thuộc khởi động theo đúng thứ tự và chỉ khi service nó phụ thuộc đã sẵn sàng.
* **Volumes:** Sử dụng để lưu trữ dữ liệu của PostgreSQL và Elasticsearch một cách bền vững.
* **Restart Policy:** Hầu hết các services được cấu hình với restart: unless-stopped, giúp tự động khởi động lại nếu có lỗi, trừ khi bị dừng thủ công.

# **IV. Các Sơ Đồ Chính Của Hệ Thống**

Phần này trình bày các sơ đồ trực quan hóa cấu trúc và luồng hoạt động của hệ thống.

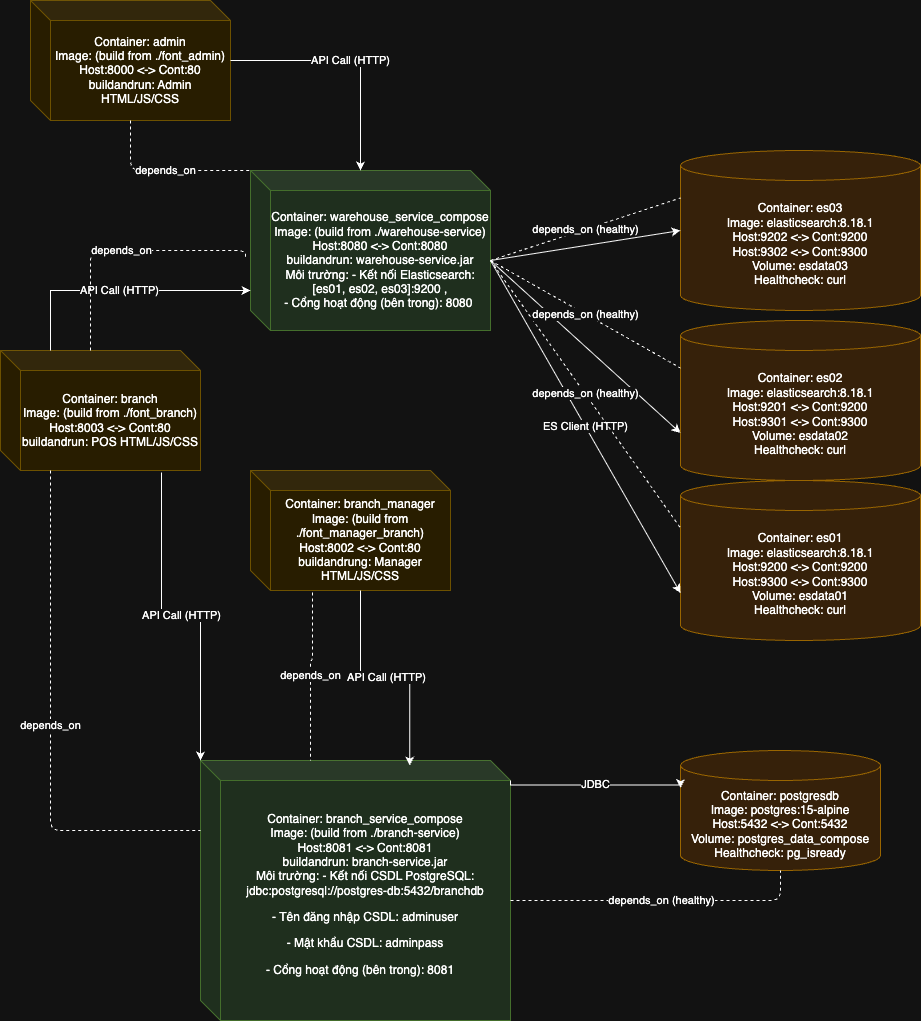
* 1. **Sơ đồ Kiến trúc Hệ thống**

A diagram of a computer

AI-generated content may be incorrect.

## 

## **2. Sơ đồ Triển khai**



**3. Các yêu cầu chức năng**

|  |  |  |
| --- | --- | --- |
| **Nhóm chức năng** | **Chức năng** | **Tác nhân** |
| R1. Quản lý sản phẩm | R1.1: Hệ thống cho phép thêm sản phẩm mới. | Người dùng |
| R1.3: Hệ thống cho phép cập nhật danh sách sản phẩm. |
| R1.4: Hệ thống cho phép sửa danh sách sản phẩm. |
| R1.5: Hệ thống cho phép xóa danh sách sản phẩm. |
| R2. Quản lý chi nhánh | R2.1: Hệ thống cho phép thêm chi nhánh. | Người dùng |
| R2.2: Hệ thống cho phép cập nhật danh sách chi nhánh. |
| R2.3: Hệ thống cho phép sửa danh sách chi nhánh. |
| R2.4: Hệ thống cho phép xóa danh sách chi nhánh. |
| R3. Nhân viên chi nhánh | R3.1:Tìm kiếm sản phẩm và xem tất cả các sản phẩm của chi nhánh đó. | Người dùng |

### **3.2. Sơ đồ Use-case:**

A black background with white text

AI-generated content may be incorrect.

**Use-case chung của hệ thống**

**3.3. Đặc tả các Use-case**

**3.3.1. Quản lý sản phẩm**

**a.Thêm sản phẩm**

|  |  |  |
| --- | --- | --- |
| Mô tả | Chức năng thêm sản phẩm cho phép người dùng them sản phẩm mới vào hệ thống.  Người dùng sẽ cung cấp các thông tin cần thiết như mã sản phẩm, tên sản phẩm, mô tả, giá,… . | |
| Tác nhân | Người dùng. | |
| Luồng sự kiện | Luồng chính | * Người dùng vào trang Thêm Sản Phẩm Mới. * Hệ thống hiển thị form thêm sản phẩm. * Người dùng nhập thông tin và ẩn vào A green rectangle with white text    AI-generated content may be incorrect. sau khi điền xong. * Hệ thống hiển thị thêm tành công. |
| Luồng con | * Nếu người không nhập đủ thông tin cần thiết,hệ thống sẽ hiển thị A white rectangular sign with black text    AI-generated content may be incorrect. |
| Tiền điều kiện | Sản phẩm chưa được lưu trong hệ thống. | |
| Hậu điều kiện | Sản phẩm mới được tạo và có thể theo dõi. | |

|  |
| --- |
| Giao diện thêm sản phẩm |
| *A green text box with black text  AI-generated content may be incorrect.* Giao diện khi sản phẩn mới được lưu vào hệ thống |

**b. Cập nhật danh sách sản phẩm (UC1.2)**

|  |  |  |
| --- | --- | --- |
| Mô tả | | Chức năng cho phép người dùng cập nhật danh sách sản phẩm sau khi thêm. |
| Tác nhân | | Người dùng |
| Luồng sự kiện | Luồng chính | * Người dùng chọn  sau khi đã thêm sản phẩm mới. * Hệ thống cập nhật danh sách toàn bộ sản phẩm đã lưu trước đó. * Người dùng xem danh sách mới cập nhật. |
| Luồng con | * Hệ thống hiển thị tải danh sách sản phẩm thành công. |
| Tiền điều kiện | | Người dùng đã nhập sản phẩm mới. |
| Hậu điều kiện | | Người dùng đã lưu thông tin sản phẩm mới. |

|  |
| --- |
| Giao diện danh sách cập nhật sản phẩm |
| *A screenshot of a computer  AI-generated content may be incorrect.* Giao diện cập nhật thành công |

**c. Xóa danh sách (UC1.3)**

|  |  |  |
| --- | --- | --- |
| Mô tả | | Chức năng cho phép người dùng xóa sản phẩm trong danh sách khi cần. |
| Tác nhân | | Người dùng |
| Luồng sự kiện | Luồng chính | * Người dùng vào giao diện danh sách toàn bộ sản phẩm. * Người dùng chọn  tại sản phẩm muốn xóa. * Hệ thống hiển thị thông báo xác thực và 2 đơn vị lựa chọn là “OK” và “Hủy”. * Người dùng chọn “OK” để xóa. * Hệ thống xóa sản phẩm vừa chọn. |
| Luồng con | Nếu người dùng từ chối chọn “Hủy”, hệ thống hủy yêu cầu và giữ nguyên danh sách. |
| Tiền điều kiện | | Người dùng đã vào giao diện và giao diện hoạt động. |
| Hậu điều kiện | | Danh sách cập nhật lại không hiển thị sản phẩm vừa xóa. |

|  |
| --- |
| Giao diện danh sách toàn bộ sản phẩm |
| Giao diện hệ thống hiển thị thông báo xóa |
| Giao diện xóa thành công |

**d. Sửa danh sách(UC1.4)**

|  |  |  |
| --- | --- | --- |
| Mô tả | | Chức năng sửa danh sách cho phép người dùng sửa thông tin cần sửa với sản phẩm trong danh sách. |
| Tác nhân | | Người dùng |
| Luồng sự kiện | Luồng chính | * Người dùng chọn  với sản phẩm. * Hệ thống hiển thị giao diện sửa thông tin sản phẩm. * Người dùng sửa thông tin cần sửa và chọn sau khi sửa xong. * Hệ thông lưu thông tin và hiển thị thông tin đã sửa. |
| Luồng con | * Người dùng chọn Hủy . * Hệ thống giữ nguyên thông tin. |
| Tiền điều kiện | | Người dùng vào và giao diện sửa hoạt động bình thường. |
| Hậu điều kiện | | Người dùng sửa thành công và hệ thống hiển thị sản phẩm trên giao diện. |

|  |
| --- |
| Giao diện danh sách toàn bộ sản phẩm |
| *A screenshot of a computer  AI-generated content may be incorrect.* Giao diện sửa danh sách |
| Giao diện sau khi sửa |

**3.3.2. Quản lý chi nhánh**

**a. Thêm chi nhánh (UC1.5)**

|  |  |  |
| --- | --- | --- |
| Mô tả | | Chức năng cho phép người dùng xem những chi nhánh mới. |
| Tác nhân | | Người dùng |
| Luồng sự kiện | Luồng chính | * Người dùng vào giao diện Quản lý chi nhánh. * Hệ thống hiển thị giao diện thêm chi nhánh mới. * Người dùng điền thông tin và chọn  sau khi xong. * Hệ thống lưu thông tin. |
| Luồng con | * Người dùng có thể chọn để xóa nếu điền sai. * Hệ thống xóa những thông tin đã điền. |
| Tiền điều kiện | | Người dùng điền chi nhánh mới chưa có. |
| Hậu điều kiện | | Hệ thống hiển thị chi nhánh đã lưu vào danh sách. |

|  |
| --- |
| A screenshot of a computer  AI-generated content may be incorrect. Giao diện thêm chi nhánh |
| *A screenshot of a computer  AI-generated content may be incorrect.* Giao diện điền thông tin |
| Giao diện lưu và hiển thị thông tin |

**b. Cập nhật danh sách chi nhánh (UC1.6)**

|  |  |  |
| --- | --- | --- |
| Mô tả | | Chức năng này cho phép người dùng cập nhật lại chi nhánh khi cần thiết. |
| Tác nhân | | Người dùng |
| Luồng sự kiện | Luồng chính | * Người dùng vào giao diện danh sách chi nhánh. * Hệ thống hiển thị danh sách. * Người dùng nhập chọn  để cập nhật. * Hệ thống cập nhật lại danh sách. |
| Tiền điều kiện | | Người dùng đã nhập thông tin chi nhánh mới. |
| Hậu điều kiện | | Hệ thống cập nhật thành công. Giao diện hoạt động bình thường. |

|  |
| --- |
| *A screenshot of a computer  AI-generated content may be incorrect.* Giao diện danh sách chi nhánh |
| *A screenshot of a computer  AI-generated content may be incorrect.* Giao diện cập nhật danh sách chi nhánh |

**c. Sửa danh sách chi nhánh(UC1.7)**

|  |  |  |
| --- | --- | --- |
| Mô tả | | Chức năng này cho phép sửa thông tin chi nhánh. |
| Tác nhân | | Người dùng |
| Luồng sự kiện | Luồng chính | * Người dùng chọn tại giao diện. * Hệ thống hiện thị giao diện sửa danh sách. * Người dùng sửa thông tin. * Hệ thống cập nhật lại thông tin đã sửa. |
| Luồng con | * Nếu nhập sai thông tin chọn  để xóa. |
| Tiền điều kiện | | Người dùng nhập đúng thông tin chi nhánh mới. |
| Hậu điều kiện | | Nếu thông tin đúng, hệ thống cập nhật thông tin sửa. |

|  |
| --- |
| Giao diện danh sách chi nhánh  A screenshot of a computer  AI-generated content may be incorrect.  Giao diện sửa thông tin chi nhánh |
| *A screenshot of a computer  AI-generated content may be incorrect.* Giao diện sau khi sửa |

**d. Xóa danh sách chi nhánh(UC1.8)**

|  |  |  |
| --- | --- | --- |
| Mô tả | | Chức năng cho phép xóa thông tin chi nhánh. |
| Tác nhân | | Người dùng |
| Luồng sự kiện | Luồng chính | * Người dùng vào danh sách chi nhánh. * Người dùng chọn  để xóa. * Hệ thống hiện thị thông báo và 2 nút chọn.\ * Người dùng chọn A blue rectangle with black text    AI-generated content may be incorrect. để xóa. * Hệ thống sẽ xóa thông tin. |
| Luồng con | * Người dùng chọn  nếu không xóa. * Hệ thông giữ nguyên thông tin. |
| Tiền điều kiện | | Danh sách hiển thị chi nhánh cần xóa. |
| Hậu điều kiện | | Hệ thống hiển thị sau khi xóa. |

A screenshot of a computer

AI-generated content may be incorrect.

Giao diện danh sách chi nhánh

A screen shot of a computer

AI-generated content may be incorrect.

Giao diện thông báo xóa

*A screenshot of a computer

AI-generated content may be incorrect.*

Giao diện đã xóa

# **V. Phân Tích Vấn Đề và Các Rủi Ro Tiềm Ẩn**

## **1. Vấn Đề Hiện Tại**

* **Thiếu xác thực và phân quyền (Authentication & Authorization):**
  + Không có cơ chế đăng nhập hay kiểm tra quyền truy cập cho các API backend. Bất kỳ ai biết địa chỉ API đều có thể thực hiện thao tác.
  + Các frontend UI không có cơ chế đăng nhập, chỉ dựa vào việc truy cập đúng URL.
* **Quản lý lỗi và phản hồi từ API:**
  + Một số hàm JavaScript xử lý lỗi API chỉ hiển thị response.statusText hoặc error.message, có thể không đủ chi tiết cho người dùng hoặc gỡ lỗi. Mặc dù WarehouseController có trả về lỗi dạng JSON cho việc giảm tồn kho, cần đảm bảo tính nhất quán.
* **Giao diện người dùng cơ bản:**
  + Frontend được xây dựng bằng HTML/CSS/JS thuần túy, thiếu các tính năng nâng cao.

## **2. Rủi Ro Tiềm Ẩn**

* **Bảo mật:** Thiếu xác thực/phân quyền là một lỗ hổng lớn, có thể dẫn đến truy cập và thay đổi dữ liệu trái phép.
* **Độ trễ mạng giữa các service:** Trong môi trường microservices, độ trễ mạng có thể ảnh hưởng đến hiệu suất chung. Cần có cơ chế timeout và retry hợp lý.
* **Khó khăn trong gỡ lỗi (Debugging):** Việc gỡ lỗi một yêu cầu đi qua nhiều microservices có thể phức tạp. Cần có logging tập trung và tracing.
* **Phụ thuộc vào phiên bản Elasticsearch:** Cấu hình xpack.security.enabled=false có thể không được khuyến khích cho môi trường production hoặc các phiên bản Elasticsearch mới hơn mà không có cấu hình bổ sung.
* **Tính nhất quán cuối cùng (Eventual Consistency):** Elasticsearch là một hệ thống "eventually consistent". Điều này có nghĩa là dữ liệu mới ghi có thể không ngay lập tức hiển thị trong các truy vấn tìm kiếm. Điều này cần được xem xét, đặc biệt sau các thao tác cập nhật hoặc xóa sản phẩm/tồn kho.

# **VI. Đề Xuất Giải Pháp và Hướng Phát Triển**

## **1. Giải Quyết Các Vấn Đề Hiện Tại**

* **Xác thực và Phân quyền:**
  + **Backend:** Tích hợp Spring Security cho branch-service và warehouse-service. Triển khai phân quyền dựa trên vai trò (ví dụ: ADMIN, BRANCH\_MANAGER, POS\_STAFF).
  + **Frontend:** Xây dựng trang đăng nhập. Lưu token nhận được từ backend và gửi kèm trong header của mỗi yêu cầu API.

## **2. Cải Thiện và Nâng Cấp**

* **Nâng cấp Frontend:**
  + Xem xét việc chuyển đổi các ứng dụng frontend sang một framework JavaScript hiện đại (React, Vue, Angular) để cải thiện trải nghiệm người dùng, khả năng quản lý state, tái sử dụng component và hiệu suất.
* **Logging và Monitoring Tập Trung:**
  + Tích hợp một giải pháp logging tập trung (ví dụ: ELK stack - Elasticsearch, Logstash, Kibana, mà hệ thống đã có sẵn Elasticsearch).
  + Sử dụng các công cụ monitoring (ví dụ: Prometheus, Grafana) và tracing phân tán (ví dụ: Jaeger, Zipkin) để theo dõi sức khỏe và hiệu suất của các microservices.
* **API Gateway:**
  + Triển khai một API Gateway (ví dụ: Spring Cloud Gateway, Kong) làm điểm vào duy nhất cho tất cả các yêu cầu từ frontend. API Gateway có thể xử lý các tác vụ chung như xác thực, rate limiting, routing, và tổng hợp phản hồi.
* **Đảm bảo tính nhất quán dữ liệu giữa các service:**
  + Đối với các thao tác cần sự nhất quán cao hơn giữa branch-service và warehouse-service (ví dụ khi xóa chi nhánh), cân nhắc sử dụng các mẫu thiết kế như Saga pattern để quản lý các giao dịch phân tán.
* **Cơ chế Retry và Circuit Breaker:**
  + Trong giao tiếp giữa các service, triển khai cơ chế retry (thử lại) cho các lỗi tạm thời và Circuit Breaker (ngắt mạch) để tránh làm quá tải một service đang gặp sự cố. Các thư viện như Resilience4j (Java) có thể hữu ích.
* **Tăng cường Health Checks:**
  + Các health check hiện tại trong docker-compose.yml là tốt, nhưng có thể mở rộng để kiểm tra sâu hơn về trạng thái của ứng dụng (ví dụ: kết nối cơ sở dữ liệu, không gian đĩa). Spring Boot Actuator cung cấp endpoint /health có thể tùy chỉnh.
* **Tài liệu API:**
  + Sử dụng Swagger/OpenAPI để tự động tạo tài liệu cho các REST API, giúp việc tích hợp và bảo trì dễ dàng hơn.

## **3. Hướng Phát Triển Tương Lai**

* **Quản lý người dùng và vai trò:** Phát triển module quản lý người dùng chi tiết hơn, gán vai trò và quyền hạn linh hoạt.
* **Quản lý đơn hàng và lịch sử giao dịch:** Mở rộng hệ thống để quản lý đơn hàng, lịch sử mua hàng của khách.
* **Báo cáo và thống kê:** Xây dựng các chức năng báo cáo doanh thu, sản phẩm bán chạy, hiệu suất chi nhánh.
* **Thông báo (Notifications):** Hệ thống thông báo cho các sự kiện quan trọng (ví dụ: hàng sắp hết, có sản phẩm mới).
* **Tích hợp thanh toán:** Kết nối với các cổng thanh toán điện tử.
* **Chương trình khách hàng thân thiết:** Quản lý điểm thưởng, ưu đãi cho khách hàng.
* **Kiểm thử (Testing):** Bổ sung unit test, integration test và end-to-end test một cách toàn diện hơn. Các file test hiện tại (BranchServiceApplicationTests.java, WarehouseServiceApplicationTests.java) chỉ là file mặc định.

# **VII. Đánh Giá Theo Tiêu Chí Kỹ Thuật và Đề Xuất**

Phần này đánh giá mức độ đáp ứng của dự án đối với các tiêu chí kỹ thuật được đặt ra, đồng thời đề xuất các giải pháp để hoàn thiện những khía cạnh còn thiếu.

## **1. Tiêu Chí Bắt Buộc**

### **1.1. Fault Tolerance (Khả năng chịu lỗi):**

* + **Hiện trạng:**
    - **Elasticsearch:** Được triển khai dưới dạng cụm 3 node (es01, es02, es03). Điều này cung cấp khả năng chịu lỗi ở mức độ dữ liệu. Nếu một node Elasticsearch gặp sự cố, cụm vẫn có thể hoạt động (tùy thuộc vào cấu hình sharding và replication của index, mặc định Elasticsearch sẽ có replica).
    - **PostgreSQL:** Chỉ có một instance postgres-db. Nếu container này lỗi, branch-service sẽ không thể hoạt động.
    - **Services (warehouse-service, branch-service):** Mỗi service hiện chỉ có một instance. Nếu container của service nào đó lỗi, chức năng tương ứng sẽ ngừng hoạt động. Docker Compose có restart: unless-stopped, giúp tự khởi động lại container nếu nó bị crash, nhưng đây không phải là fault tolerance ở mức độ service vẫn tiếp tục phục vụ yêu cầu khi một instance lỗi.
  + **Thiếu sót:**
    - Chưa có khả năng chịu lỗi thực sự cho PostgreSQL và các application services (nếu một máy chủ/container chết, service sẽ gián đoạn).
  + **Giải pháp:**
    - **PostgreSQL:** Triển khai PostgreSQL với cơ chế replication (ví dụ: streaming replication) để có một standby node sẵn sàng thay thế khi primary node lỗi.

### **1.2. Distributed Communication (Giao tiếp Phân tán):**

* + **Hiện trạng:**
    - Các frontend UIs giao tiếp với backend services qua HTTP/REST API.
    - warehouse-service giao tiếp với Elasticsearch qua client của Elasticsearch (thường dựa trên HTTP).
    - branch-service giao tiếp với PostgreSQL qua JDBC.
    - Hệ thống được thiết kế để các thành phần (container) chạy trên cùng một mạng Docker (tiemtaphoa-network) nhưng có thể được cấu hình để chạy trên nhiều máy nếu hạ tầng Docker Swarm hoặc Kubernetes được sử dụng.

### **1.3. Sharding hoặc Replication (Phân mảnh hoặc Sao chép Dữ liệu):**

* + **Hiện trạng:**
    - **Elasticsearch:** Tự bản thân Elasticsearch khi chạy trong một cụm đã thực hiện cả sharding (phân chia index thành các shard) và replication (tạo bản sao của shard trên các node khác nhau). Đây là một điểm mạnh của dự án. Các truy vấn từ các phương thức của service warehouse (products) sẽ được hưởng lợi từ cơ chế này.
  + **Thiếu sót:**
    - Chưa có replication cho PostgreSQL.
  + **Giải pháp:**
    - **PostgreSQL Replication:** Thiết lập streaming replication cho PostgreSQL. Điều này liên quan đến việc cấu hình một primary node và một hoặc nhiều standby/replica nodes. Dữ liệu sẽ được sao chép từ primary sang các replica.

### **1.4. Simple Monitoring / Logging (Giám sát / Ghi nhật ký đơn giản):**

* + **Hiện trạng:**
    - **Logging:** Các service Spring Boot (warehouse-service, branch-service) sử dụng SLF4J (thông qua Logback mặc định) để ghi log ra console. Khi chạy trong Docker, các log này có thể được xem qua docker logs (container\_name).Hoặc đọc log trược tiếp từ terminal bạn chạy docker-compose up .
    - **Frontend:** Sử dụng console.log trong JavaScript để ghi log phía client.
    - **Elasticsearch & PostgreSQL:** Cũng có cơ chế logging riêng của chúng.
  + **Thiếu sót và Giải pháp chi tiết hơn:**
    - **Logging Tập Trung:**
      * **Hiện trạng:** Chưa có hệ thống logging tập trung. Việc xem log của từng container riêng lẻ sẽ khó khăn.
      * **Giải pháp:**
        1. **Cấu hình output log dạng JSON:** Sửa đổi cấu hình Logback trong các Spring Boot services để output log dưới định dạng JSON. Điều này giúp việc parse và phân tích log dễ dàng hơn.
    - **Monitoring (Giám sát Metrics):**
      * **Hiện trạng:** Chưa có công cụ giám sát trạng thái/hiệu năng chuyên dụng.
      * **Giải pháp (Sử dụng Prometheus và Grafana):**
        1. **Exporter cho các thành phần:**

**Spring Boot Services (warehouse-service, branch-service):**

Thêm thư viện micrometer-registry-prometheus vào pom.xml

Spring Boot Actuator sẽ tự động tạo một điểm cuối /actuator/prometheus. Hệ thống giám sát Prometheus sau đó có thể định kỳ truy cập điểm cuối này để lấy các số liệu đo lường quan trọng về hiệu suất và hoạt động của ứng dụng Spring Boot, giúp bạn theo dõi và đảm bảo ứng dụng chạy ổn định.

**Elasticsearch:**

Triển khai **Elasticsearch Exporter** (ví dụ: prometheus-elasticsearch-exporter của JustWatch). Đây là một service riêng biệt sẽ kết nối tới cụm Elasticsearch của bạn, thu thập các metrics quan trọng và expose chúng qua một HTTP endpoint cho Prometheus scrape.

**PostgreSQL:**

Triển khai **Postgres Exporter** (ví dụ: postgres\_exporter của Prometheus Community). Tương tự, nó sẽ kết nối tới PostgreSQL và expose metrics.

**Node Exporter (Tùy chọn nhưng khuyến nghị):** Triển khai Node Exporter trên các máy chủ Docker host để thu thập metrics về hệ điều hành.

* + - * 1. **Prometheus Server:**
        2. **Grafana:**

### **1.5. Basic Stress Test (Kiểm tra tải cơ bản):**

* + **Thiếu sót:** Chưa thực hiện.
  + **Giải pháp:**
    - Sử dụng các công cụ như Apache JMeter, k6, vegeta, hoặc thậm chí là các script đơn giản (ví dụ: Python với requests) để mô phỏng nhiều người dùng đồng thời gửi yêu cầu đến các API chính (ví dụ: API lấy danh sách sản phẩm, API bán hàng, API thêm sản phẩm).
    - Quan sát hành vi của hệ thống:
      * Thời gian phản hồi của API.
      * Mức sử dụng CPU/RAM của các container.
      * Số lượng lỗi (HTTP 5xx, 4xx).
      * Log của các service để phát hiện bottleneck hoặc lỗi phát sinh dưới tải.
    - Bắt đầu với số lượng request nhỏ và tăng dần để xác định giới hạn của hệ thống.
    - Hoặc sử dụng lệnh spam API để test.Và xem các thông số phản hồi.