

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐH NGOẠI NGỮ TIN HỌC TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**🙢 🙠**

**BÁO CÁO MÔN**

**MẪU THIẾT KẾ PHẦN MỀM**

**ĐỀ TÀI:**

**Website thời trang The20s**

**Giảng viên hướng dẫn:** Lương Văn Minh

**Sinh viên thực hiện đề tài:**

* Nguyễn Anh Khoa – 18DH110239
* Cao Hữu Tài – 18DH110218

**BẢNG CHỮ KÝ**

**Tác giả:**

|  |  |
| --- | --- |
| Tên: Cao Hữu Tài  Vị trí: Thành viên | Chữ ký:  Ngày: |
| Tên: Nguyễn Anh Khoa  Vị trí: Thành viên | Chữ ký:  Ngày: |

**Người điều chỉnh:**

|  |  |
| --- | --- |
| Tên:  Vị trí: | Chữ ký:  Ngày: |

**Người duyệt:**

|  |  |
| --- | --- |
| Tên:  Vị trí: | Chữ ký:  Ngày: |

**MỤC LỤC**

[CHƯƠNG I: GIỚI THIỆU 1](#_Toc92630949)

[CHƯƠNG II: PHÂN TÍCH YÊU CẦU 4](#_Toc92630950)

[1. Các loại lược đồ 4](#_Toc92630951)

[1.a. Use Cases Diagram 4](#_Toc92630952)

[1.b. Entity Relationship Diagram (ERD) 5](#_Toc92630953)

[1.c. Data Flow Diagram (DFD) 6](#_Toc92630954)

[2. Mô tả chi tiết các phân hệ của ứng dụng 7](#_Toc92630955)

[CHƯƠNG III: CÔNG NGHỆ SỬ DỤNG 9](#_Toc92630956)

[1. Design Pattern 9](#_Toc92630957)

[2. Các mẫu Pattern được sử dụng 9](#_Toc92630958)

[2.a. MVC Pattern: 9](#_Toc92630959)

[2.b. Singleton Pattern 10](#_Toc92630960)

[2.c. Template Method Pattern 11](#_Toc92630961)

[2.d. Proxy Pattern 13](#_Toc92630962)

[2.e. Prototype Pattern 14](#_Toc92630963)

[2.f. Iterator Pattern 15](#_Toc92630964)

[CHƯƠNG IV: XÂY DỰNG VÀ TRIỂN KHAI 18](#_Toc92630965)

[1. Mockup 18](#_Toc92630966)

[2. Giao diện của ứng dụng 21](#_Toc92630967)

[3. Thanh toán trực tiếp 24](#_Toc92630968)

[CHƯƠNG V: TỔNG KẾT 31](#_Toc92630969)

[1. Những kết quả đạt được 31](#_Toc92630970)

[2. Kết luận 31](#_Toc92630971)

[CHƯƠNG V: TÀI LIỆU THAM KHẢO 32](#_Toc92630972)

1. GIỚI THIỆU

**Mô tả đề tài**

Hiện nay việc kinh doanh – mua bán là nhu cầu không thể thiếu đối với mỗi chúng ta. Trong cuộc sống cạnh tranh ngày nay thì việc giới thiệu sản phẩm kinh doanh đến từng cá nhân với chi phí thấp, hiệu quả cao là một vấn đề luôn được quan tâm của người kinh doanh cùng với nhu cầu mua sắm với những loại sản phẩm đa dạng và hợp túi tiền của người tiêu dùng vì vậy thương mại điện tử đã được ra đời và càng ngày càng phát triển trên toàn thế giới.

Nắm bắt được xu hướng hiện nay nhóm chúng em đã tiến hành thiết kế một trang website thương mại điện tử “Thời Trang Bán Quần Áo” để có thể giúp cho cửa hàng phát triển nhanh hơn, có thể quản lý cửa hàng một cách tốt hơn. Việc xây dựng một trang website thương mại điện tử này đã mang lại cho cửa hàng nhiều lợi ích như giảm thiểu được chi phí bán hàng tiếp thị và giao dịch. Bằng phương tiện điện tử này có thể giúp cho website giao dịch được với rất nhiều khách hàng, Catalogue điện tử (e-Catalogue) trên Web cũng phong phú hơn và thường xuyên cập nhật so với Catalogue in ấn. Và nhóm chúng em đã tích hợp vào Website các phương thức thanh toán Online mới, hiện đại hơn giúp cho khách hàng giảm đáng kể về thời gian và chi phí giao dịch. Thời gian giao dịch qua Internet chỉ bằng 7% thời gian giao dịch qua Fax và chi phí thanh toán điện tử qua Internet cũng chỉ bằng 10% đến 20% chi phí thanh toán theo lối truyền thống. Những việc tiếp cận trực tuyến hầu hết sẽ mang lại lợi nhuận và lợi ích cho Website.

Mô hình kinh doanh của Website nhóm chúng em hướng tới đó chính là B2C (Business-to-Consumer). Nhóm sinh viên chúng em sử dụng công nghệ phát triển web ASP.NET MVC5 để thiết kế xây dụng Website là “Thời trang bán quần áo”. Nhóm đối tượng khách hàng là dành cho cả phái nam và phái nữ đặc biệt chuyên về thời trang trẻ trung từ 15-30 tuổi. Bên cạnh đó còn cả khách hàng là các người lớn tuổi có thể ghé Website mua sản phẩm của shop để làm quà tặng con trai, con gái, cháu của mình, …

Mục tiêu của nhóm chúng em không chỉ là hoàn thành bài tập nhóm đã được giao mà còn lớn hơn thế, mong muốn tương lai sẽ có thể thực thi ý tưởng này, sử dụng những gì được dạy về thương mại điện tử áp dụng vào Website đã tạo như một công cụ để quảng bá hình ảnh, giới thiệu sản phẩm và hỗ trợ kinh doanh. Ý tưởng của chúng em không phải là một ý tưởng mới, nhưng nó đã giải quyết được những mục tiêu về bán hàng, quảng bá thương hiệu, giới thiệu sản phẩm đến khách hàng, giúp cho việc mua sắm sản phẩm của khách hàng và thanh toán điện tử được thuận tiện và dễ dàng hơn. Chỉ với một vài cú Click chuột, khách hàng có thể tuỳ ý mua sắm các sản phẩm mà mình yêu thích, với các thông tin chi tiết về sản phẩm như giá cả, xuất sứ, size, màu, số lượng, … Đối với kinh doanh, sự tiện lợi và hài lòng của khách hàng được đặt lên trên hết, vì vậy trang web luôn có sự hỗ trợ trực tuyến online 24/7, khách hàng có thể hỏi về các thông tin sản phẩm, giảm giá, hay nhân viên có thể tư vấn cho khách hàng về những sản phẩm có thể đi kèm, các xu hướng thời trang mới cập nhật.

Một số trang chính của Website:

* Trang chủ: Đây là trang giới thiệu về các sản phẩm nổi bật của cửa hàng. Tổng hợp những thông tin mới nhất của cửa hàng, các thông tin khuyến mãi, bộ sưu tập mới.
* Trang giới thiệu: Trang này giới thiệu về shop của nhóm.
* Trang sản phẩm: Đây là trang khách hàng xem hoặc tham khảo các mặt hàng, sản phẩm đang có mặt tại shop để lựa chọn.
* Trang mua hàng: Đây la trang để khách hàng có thể chọn các loại thanh toán mong muốn và mua hàng.
* Trang liên hệ: đây là trang mà khách hàng có thể biết thông tin về shop để có thể liên hệ với cửa hàng.

Đối với khách hàng:

Được đa dạng hóa nhu cầu mua sản phẩm, có nhiều sự lựa chọn hơn. Không phải lo về tình trạng sản phẩm bị biến động giá quá nhiều vì nguồn cung sản phẩm là 1 nguồn cung khổng lồ và được phân loại theo từng nhu cầu nên khách hàng không phải lo về việc sản phẩm mình cần bị thiếu hụt hoặc bị biến động giá quá nhiều.

Với cùng một sản phẩm được đa dạng hóa về chất liệu và cách thiết kế làm cho sản phẩm trở nên tối ưu cũng như bền bỉ hơn giúp cho bộ máy tổng thể vận hành tốt hơn. Vì vậy mà một loại sản phẩm có nhiều giá cả khách nhau từ thấp đến cao cho khách hàng có nhiều sự lựa chọn hơn trong việc chọn sản phẩm phù hợp theo điều kiện kinh tế của khách hàng.

Khách hàng có thể lựa chọn nhiều hình thức khác nhau khi đặt hàng trên ứng dụng như thanh toán bằng tiền mặt khi nhận hàng hoặc thanh toán bằng thẻ ngân hàng hoặc các ví điện tử thông dụng (Airpay, Momo, ...)

Khách hàng cũng được hỗ trợ tư vấn trực tuyến trên ứng dụng cũng như là phản hồi ý kiến về ứng dụng cũng như đánh giá chất lượng sản phẩm hoặc đổi trả hàng trong vòng 72 giờ nếu sản phẩm bị lỗi từ nhà sản xuất. Khách hàng cũng có thể sử dụng dịch vụ giao hàng của shop để tiến hành đổi trả hàng nhưng thời gian có thể sẽ lâu hơn khi khách hàng trực tiếp đến shop để đổi trả hàng.

Khách hàng còn nhận được những ưu đãi hấp dẫn như sale trong một số ngày đặc biệt trong tháng (những ngày và tháng giống nhau. Vd: 1/1,2/2…) khi đặt hàng trên ứng dụng. Đồng thời căn cứ vào số lượng đơn hàng mà khách hàng đã có, shop sẻ ưu đãi cho khách hàng một vài đơn hàng được miễn phí vẫn chuyển nếu đơn hàng đạt giá trị quy định (số lượng đơn hàng đã đặt của khách hàng càng nhiều, đơn miễn phí vận chuyển có giá trị càng lớn).

1. PHÂN TÍCH YÊU CẦU
   1. Các loại lược đồ
      1. Use Cases Diagram

Diagram

Description automatically generated

*Hình 01: Diagram 01 – User role*

Diagram

Description automatically generated

*Hình 02: Diagram 02 – Khách hàng role*

Diagram

Description automatically generated

*Hình 03: Diagram 03 – Admin role*

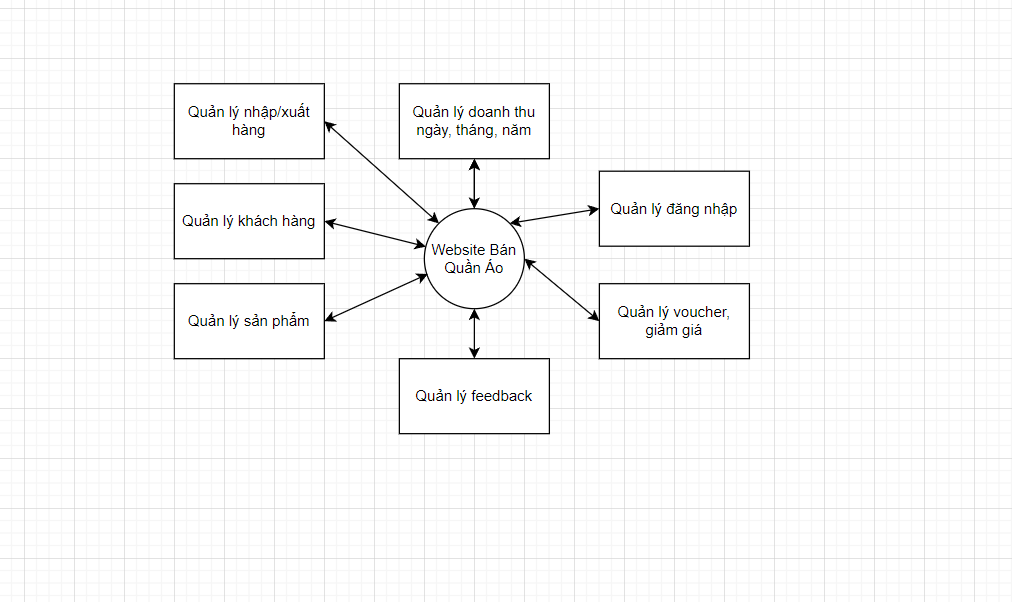
* + 1. Entity Relationship Diagram (ERD)

Diagram

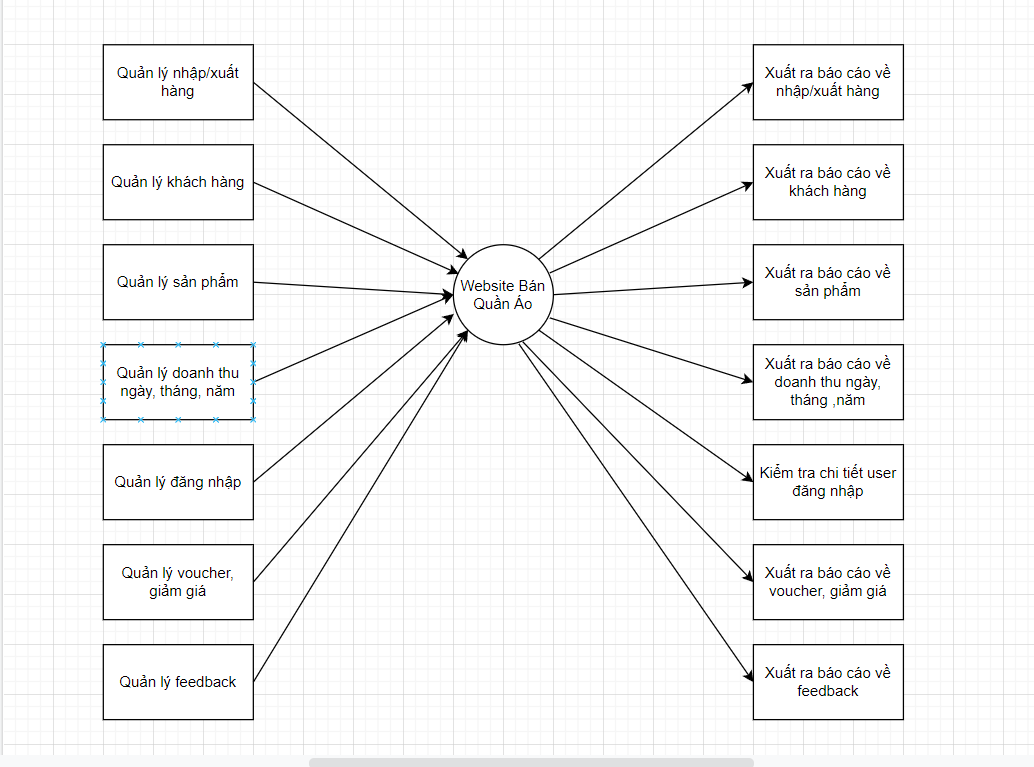
Description automatically generated

*Hình 04: ERD*

* + 1. Data Flow Diagram (DFD)



*Hình 05: DFD lv0*



*Hình 06: DFD lv2*

* 1. Mô tả chi tiết các phân hệ của ứng dụng

***Trang chủ***

Trang chủ được bố trí hình ảnh sản phẩm chất lượng hấp dẫn, cuốn hút người xem, dẫn dắt khách hàng đi từ click trang này đến trang khác. Sắp xếp sản phẩm trong chuyên mục theo nhiều loại như sản phẩm mới, sản phẩm nổi bật, sản phẩm được nhiều người mua, sản phẩm giảm giá, … Hiển thị thêm các thông tin khuyến mãi và giá cả.

***Trang About Us (Giới thiệu)***

Trang About us là nơi chứa đựng câu chuyện về doanh nghiệp, nơi doanh nghiệp tự hào về những thành tựu trong kinh doanh, trả lời câu hỏi thường gặp và cho khách hàng biết vì sao họ nên mua sản phẩm của bạn. Trang abous us sẽ giới thiệu thông tin dễ dàng tới khách hàng hơn biết nhiều hơn về Website từ đó sẽ có được niềm tin của khách hàng, tạo hứng thú và dẫn dắt khách hàng.

***Trang sản phẩm***

Trang sản phẩm là danh sách toàn bộ sản phẩm của shop đăng tải trên website. Tất cả sản phẩm có cùng đặc tính với nhau được phân loại, sắp xếp, tập hợp thành từng nhóm riêng biệt để khách hàng có thể dễ dàng tìm kiếm đến loại sản phẩm mà mình mong muốn mua.

***Trang Contact (Liên hệ)***

Trang liên hệ là một tính năng gần như bắt buộc phải có đối với mỗi trang Web. Nó là trang mà chủ doanh nghiệp có thể để lại địa chỉ Email và các thông tin liên hệ của Shop để khách hàng truy cập sẽ dễ dàng liên hệ hơn. Trang Contact không chỉ cung cấp một cách thuận tiện và chuyên nghiệp hơn cho khách hàng để liên lạc với bạn, nhưng nó cũng giúp bảo vệ thông tin của bạn hơn.

***Trang giỏ hàng***

Giỏ hàng là một trong những tính năng quan trọng và không thể thiếu của bất kỳ thiết kế website bán hàng trực tuyến chuyên nghiệp nào hiện nay. Tính năng này sẽ giúp khách hàng dễ dành nắm bắt được những sản phẩm mình đã chọn là gì, số lượng là bao nhiêu. Họ cũng có thể tùy chỉnh thay đổi số lượng các mặt hàng đã mua, thêm hoặc xóa bớt tùy theo nhu cầu của mình.

***Trang Account***

Là 1 tính năng không thể thiếu của các Website thương mại điện tử. Khách hàng cần có một account để có thể dùng nó đăng nhập vào website và bắt đầu mua sắm dưới thông tin cá nhân chính xác do khách hàng cung cấp. Một website thương mại điện tử cũng cho thấy sự hiệu quả bằng cách dựa trên lượng khách hàng đăng ký account.

***Trang Checkout***

Trang checkout trong website bán hàng thông báo tới người mua hàng những sản phẩm đã có trong giỏ hàng của mình. Đồng thời, Check out yêu cầu bạn phải cung cấp họ tên, địa chỉ, nói chung là thông tin người nhận để có thể chuyển hàng. Trong các website thương mại điện tử, Check out cũng yêu cầu cung cấp thông tin về các phương thức thanh toán mà bạn lựa chọn như thẻ visa, thẻ ngân hàng giúp bạn nhanh chóng thực hiện giao dịch dễ dàng.

1. CÔNG NGHỆ SỬ DỤNG
   1. Design Pattern

Design Pattern (hay còn gọi là mẫu thiết kế) là một giải pháp chung để giải quyết các vấn đề phổ biến khi thiết kế phần mềm trong lập trình hướng đối tượng OOP.

Với sự đúc kết và công nhận từ nhiều nhà nghiên cứu, Design Pattern là mẫu chuẩn tối ưu nhất, có thể áp dụng để giải quyết không chỉ một vấn đề mà nhiều vấn đề có tính chất tương tự nhau, lặp đi lặp lại nhiều lần trong lập trình.

Design Pattern không dành riêng cho một ngôn ngữ lập trình cụ thể nào. Nó có thể được áp dụng trong hầu hết các ngôn ngữ lập trình OOP như: [PHP](https://itviec.com/it-jobs/php?utm_source=blogpost&utm_medium=center_post&utm_campaign=linktosite), [C#](https://itviec.com/it-jobs/csharp?utm_source=blogpost&utm_medium=center_post&utm_campaign=linktosite), [Java](https://itviec.com/it-jobs/java?utm_source=blogpost&utm_medium=center_post&utm_campaign=linktosite), [Python](https://itviec.com/it-jobs/python?utm_source=blogpost&utm_medium=center_post&utm_campaign=linktosite) và nhiều ngôn ngữ khác.

* 1. Các mẫu Pattern được sử dụng
     1. MVC Pattern:

Diagram

Description automatically generated**MVC** (MVC Design Pattern) là viết tắt của **Model - View - Controller**. Đó là một **mẫu kiến ​​trúc, mô hình lập trình**phổ biến được sử dụng để tạo cấu trúc cho nhiều trang web, ứng dụng tiên tiến.

Đơn giản hơn là mô hình này được chia thành 3 phần trong source code. Và mỗi phần đảm nhận vai trò và nhiệm vụ riêng biệt nhau và độc lập.

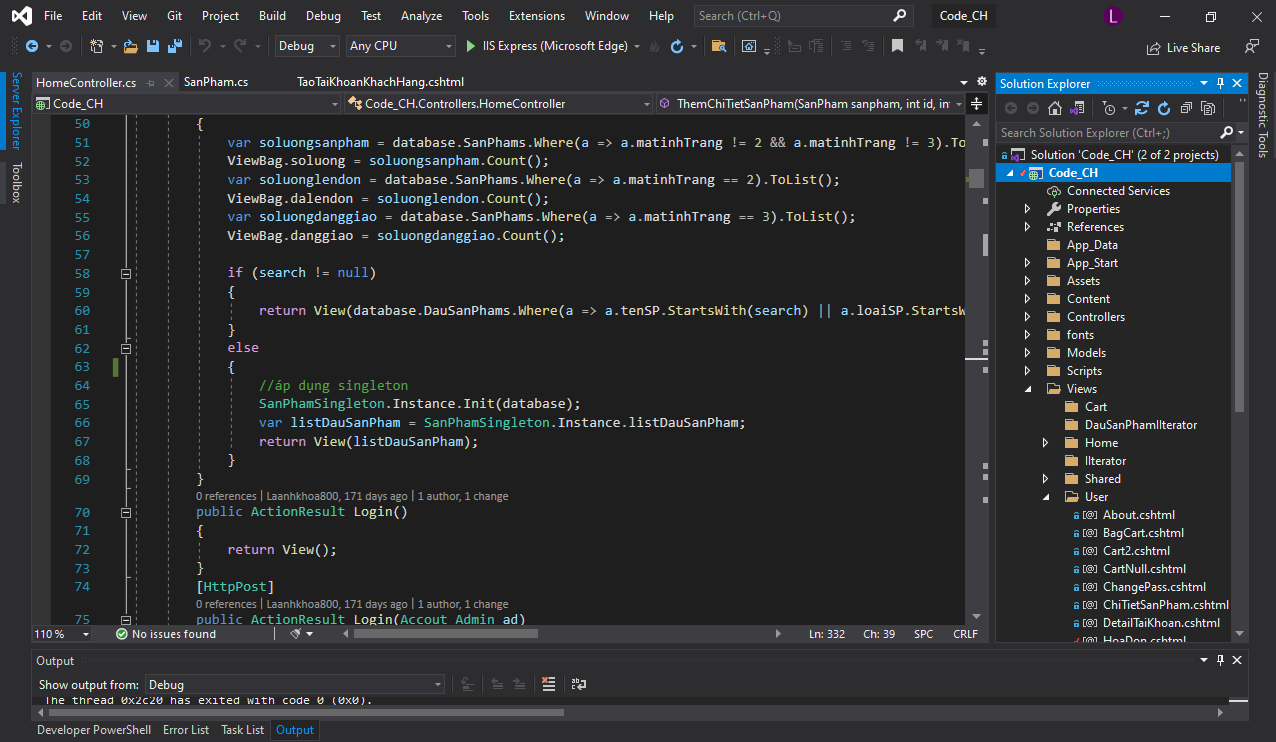
Mô hình MVC (MVC pattern) thường được dùng để phát triển giao diện người dùng. Nó cung cấp các thành phần cơ bản để thiết kế một chương trình cho máy tính hoặc điện thoại di động, cũng như là các ứng dụng web.

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

* Model: Một model là dữ liệu được sử dụng bởi chương trình. Đây có thể là cơ sở dữ liệu, file hay một đối tượng đơn giản. Chẳng hạn như biểu tượng hay là một nhân vật trong game.
* View: View là phương tiện hiển thị các đối tượng trong một ứng dụng. Chẳng hạn như hiển thị một cửa sổ, nút hay văn bản trong một cửa sổ khác. Nó bao gồm bất cứ thứ gì mà người dùng có thể nhìn thấy được.
* Controller: Một controller bao gồm cả Model lẫn View. Nó nhận input và thực hiện các update tương ứng.

Các mẫu thiết kế MVC được sử dụng phổ biến bởi các ứng dụng web và để thiết kế giao diện. Một số ngôn ngữ lập trình phổ biến như C #, Python, PHP và Java hoạt động trên lý thuyết MVC. Các mẫu thiết kế giúp quản lý mã tốt hơn. Việc tách các thành phần của MVC giúp phát triển các mã có thể tái sử dụng dễ học hơn.

Code áp dụng mẫu MVC:

Sử dụng theo mô hình MVC

* + 1. Singleton Pattern

Trong design pattern, mẫu thiết kế **Singleton Pattern** được dùng để đảm bảo chỉ có duy nhất một instance trong một class, và class đó sẽ cung cấp phương thức toàn cục để truy cập đến thực thể đó. Như vậy, Singleton Pattern được dùng khi bạn tạo ra một class mà bạn chỉ muốn chỉ có duy nhất một thực thể là instance của class đó và bạn có thể truy cập đến nó ở bất kỳ nơi đâu khi bạn muốn.

Diagram

Description automatically generated

Để biến một class thành Singleton, cần đảm bảo rằng:

* Định nghĩa một attribute là private static và đó là thể hiện duy nhất của class này
* Định nghĩa public static getInstance() dùng để khởi tạo đối tượng (hàm accessor)
* Thực hiện lazy-init trong hàm accessor (chỉ khi gọi mới khởi tạo thể hiện)
* Constructor (hoặc các constructor) là private hay protected, vì bạn không muốn client tạo nhiều thể hiện
* Client chỉ có thể gọi hàm accessor khi muốn có thể hiện của class

Một số trường hợp sử dụng của Singleton Pattern thường gặp là:

* Vì class dùng Singleton chỉ tồn tại 1 Instance nên nó thường được dùngcho các trường hợp giải quyết những bài toán cần truy cập vào các ứng dụng như: Shared resource, Logger, Configuration, Thread pool,..
* Sử dụng trong một số class của core java như: java.lang.Runtime, java.awt.Desktop.

**Code áp dụng mẫu Singleton**

* Tạo class SanPhamSingleton có 1 instance duy nhất dung hàm Init để khởi tạo danh sách các sản phẩm “Đầu Sản Phẩm” từ database

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using Code\_CH.Models;

using System.Data;

namespace Code\_CH.Models

{

public sealed class SanPhamSingleton

{

// Tạo một đối tượng cho singleton

public static SanPhamSingleton Instance { get; } = new SanPhamSingleton();

public List<DauSanPham> listDauSanPham { get; } = new List<DauSanPham>();

// Xét phương thức khởi tạo ở chế độ riêng tư

// Khơi tạo

private SanPhamSingleton() { }

public void Init(CuaHangEntities database)

{

if (listDauSanPham.Count == 0)

{

var DauSanPham = (from s in database.DauSanPhams select s).ToList();

foreach (var item in DauSanPham)

{

listDauSanPham.Add(item);

}

}

}

}

}

* Thực hiện khai báo và triển khai truy cập vào Instance của HomeController

Và UserController để hiển thị Danh sách sản phẩm

+ HomeController

public ActionResult DauSanPham(string search)

{

var soluongsanpham = database.SanPhams.Where(a => a.matinhTrang != 2 && a.matinhTrang != 3).ToList();

ViewBag.soluong = soluongsanpham.Count();

var soluonglendon = database.SanPhams.Where(a => a.matinhTrang == 2).ToList();

ViewBag.dalendon = soluonglendon.Count();

var soluongdanggiao = database.SanPhams.Where(a => a.matinhTrang == 3).ToList();

ViewBag.danggiao = soluongdanggiao.Count();

if (search != null)

{

return View(database.DauSanPhams.Where(a => a.tenSP.StartsWith(search) || a.loaiSP.StartsWith(search)).ToList());

}

else

{

//áp dụng singleton

SanPhamSingleton.Instance.Init(database);

var listDauSanPham = SanPhamSingleton.Instance.listDauSanPham;

return View(listDauSanPham);

}

}

+UserController

public ActionResult Index()

{

ViewBag.hethang = "sản phẩm đã hết!";

var animation = database.Animations.Where(a => a.idAnimation == 1).FirstOrDefault();

ViewBag.ani1 = animation.Img\_animation\_1;

ViewBag.ani2 = animation.Img\_animation\_2;

ViewBag.ani3 = animation.Img\_animation\_3;

ViewBag.ani4 = animation.Img\_animation\_4;

/\* return View(database.DauSanPhams.ToList());\*/

//áp dụng singleton

// Lỗi thời gian biên dịch: Hàm tạo SingleObject () không hiển thị

// Đối tượng SanPhamSingleton = new SanPhamSingleton();

// Lấy đối tượng duy nhất có sẵn

SanPhamSingleton.Instance.Init(database);

var listDauSanPham = SanPhamSingleton.Instance.listDauSanPham;

return View(listDauSanPham);

}

* + 1. Template Method Pattern

**Template Method Pattern** là một trong những Pattern thuộc nhóm hành vi (Behavior Pattern). Pattern này nói rằng “Định nghĩa một bộ khung của một thuật toán trong một chức năng, chuyển giao việc thực hiện nó cho các lớp con. Mẫu Template Method cho phép lớp con định nghĩa lại cách thực hiện của một thuật toán, mà không phải thay đổi cấu trúc thuật toán”.

Diagram

Description automatically generated

Template Method Pattern được sử dụng khá nhiều trong mô hình Abstract – Concrete Class. Khi chúng ta muốn các Concrete class tự thực thi xử lí theo cách của nó, nhưng đồng thời vẫn đảm bảo tuận theo những ràng buộc nhất định từ Abstract class. Ví dụ như ràng buộc về thứ tự các bước thực hiện, hay ràng buộc về dữ liệu đầu vào, đầu ra, …

Trong Template method pattern, Abstract class định nghĩa ra một template method để thực hiện một chức năng nào đó. Template method này sẽ gọi đến các method khác bên trong Abstract class để tạo dựng nên bộ khung. Nhưng có thể các method đó sẽ không được thực thi bên trong Abstract class, mà sẽ được override và thực thi lại bên trong các Concrete class.

Các thành phần tham gia Template Method Pattern:

* **AbstractClass**:

Định nghĩa các phương thức trừu tượng cho từng bước có thể được điều chỉnh bởi các lớp con.

Cài đặt một phương thức duy nhất điều khiển thuật toán và gọi các bước riêng lẻ đã được cài đặt ở các lớp con.

* **ConcreteClass**:

Là một thuật toán cụ thể, cài đặt các phương thức của AbstractClass. Các thuật toán này ghi đè lên các phương thức trừu tượng để cung cấp các triển khai thực sự. Nó không thể ghi đè phương thức duy nhất đã được cài đặt ở AbstractClass (templateMethod).

Sử dụng Template Method Pattern khi nào:

* Khi có một thuật toán với nhiều bước và mong muốn cho phép tùy chỉnh chúng trong lớp con.
* Mong muốn chỉ có một triển khai phương thức trừu tượng duy nhất của một thuật toán.
* Mong muốn hành vi chung giữa các lớp con nên được đặt ở một lớp chung.
* Các lớp cha có thể gọi các hành vi trong các lớp con của chúng một cách thống nhất (step by step).

**Code áp dụng mẫu Template Methol**

* Khởi tạo các hàm ở UserController, CheckTaiKhoan và IdentityTaiKhoan để kiểm tra Session của tài khoản đăng nhập và một Hàm GetInfoTaiKhoan để chứa các dòng công việc chỉ cần thực hiện một lần trong thuật toán của các Abstract class (lớp từu tượng) và các biến thể cần thiết để thực hiện ở lớp con

//Check Session

public bool CheckTaiKhoan()

{

if (Session["User"] != null)

return true;

return false;

}

//ID User

public TaiKhoanKhachHang IdentityTaiKhoan()

{

string email = Session["Email"].ToString();

TaiKhoanKhachHang tk = database.TaiKhoanKhachHangs.Where(a => a.emailKH == email).SingleOrDefault();

return tk;

}

//Get Info Tài khoản

public void GetInfoTaiKhoan(TaiKhoanKhachHang tk)

{

ViewBag.email = tk.emailKH;

ViewBag.hotenKH = tk.tenKH;

ViewBag.ctTinhTrang = tk.TinhTrangTaiKhoan.tenTinhTrang;

ViewBag.sodienthoai = tk.sodienthoaiKH;

ViewBag.diachi = tk.diachiKH;

ViewBag.diachitinh = tk.tinhKH;

ViewBag.diachihuyen = tk.huyenKH;

}

//Get Tài khoản khách hàng - TemplateMethod pattern

public void GetTaiKhoan()

{

if (CheckTaiKhoan() == true)

{

GetInfoTaiKhoan(IdentityTaiKhoan());

}

}

* Khai báo và thực thi để hiển thị thông tin user khi đăng nhập tài khoản ở 2 page

+ Thông tin Tài khoản

/\*Hiển thị thông tin user\*/

// Áp dụng template methol hiển thị Tài khoản khách hàng

public ActionResult DetailTaiKhoan()

{

if (Session["User"] != null)

{

GetTaiKhoan();

return View();

}

else

{

return RedirectToAction("Login", "MainUser");

}

}

+ Thông tin Khách hàng khi đặt hàng

[HttpGet]

public ActionResult HoaDon( TaiKhoanKhachHang taikhoan, FormCollection frm, string thanhphoKH)

{

Cart ca = Session["Cart"] as Cart;

string email = Session["Email"].ToString();

taikhoan = database.TaiKhoanKhachHangs.Where(a => a.emailKH == email).SingleOrDefault();

if (Session["User"] != null)

{

GetTaiKhoan();

string tinhship = thanhphoKH;

if (taikhoan.tinhKH.Contains("Thành phố Hồ Chí Minh"))

{

ViewBag.phiship = "25000";

double tong = 25000 + ca.Total\_shopping();

ViewBag.tong = tong;

}

else

{

ViewBag.phiship = "35000";

double tong = 35000 + ca.Total\_shopping();

ViewBag.tong = tong;

} }

if (Session["Cart"] == null)

{

return RedirectToAction("CartNull", "User");

}

return View(ca);

}

* + 1. Proxy Pattern

**Proxy Pattern** là một trong những Pattern thuộc nhóm cấu trúc (Structural Pattern).

Proxy có nghĩa là “ủy quyền” hay “đại diện”. Mục đích xây dựng Proxy pattern cũng chính vì muốn tạo ra một đối tượng sẽ ủy quyền, thay thế cho một đối tượng khác.

Proxy Pattern là mẫu thiết kế mà ở đó tất cả các truy cập trực tiếp đến một đối tượng nào đó sẽ được chuyển hướng vào một đối tượng trung gian (Proxy Class). Mẫu Proxy (người đại diện) đại diện cho một đối tượng khác thực thi các phương thức, phương thức đó có thể được định nghĩa lại cho phù hợp với múc đích sử dụng.

Một số kiểu proxy thường được sử dụng:

* Một remote proxy từ xa cung cấp một biểu diễn cục bộ cho một đối tượng trong một không gian địa chỉ khác, có thể sử dụng khi bạn cần một thêm chiếu định vị cho một đối tượng trong không gian địa chỉ (JVM)
* Một virtual proxy ảo là nó tạo ra một đối tượng có chi phí cao khi có yêu cầu theo yêu cầu –> lưu giữ các thông tin thêm vào về một dịch vụ thực vì vậy chúng có thể hoãn lại sự truy xuất vào dịch vụ này
* Một protection proxy bảo vệ điều khiển việc truy nhập đối tượng gốc. Các protection proxy rất hữu ích khi các đối tượng có các quyền truy nhập khác nhau –> xác thực quyền truy xuất vào một đối tượng thực
* Một smart reference proxy là sự thay thế cho một con trỏ rỗng cho phép thực hiện các chức năng thêm vào khi một đối tượng được truy nhập.

Diagram

Description automatically generated

Trong đó:

* Subject : Đối tượng này xác định giao diện chung cho RealSubject và Proxy để Proxy có thể được sử dụng bất cứ nơi nào mà RealSubject mong đợi.
* Proxy : Nó duy trì một tham chiếu đến RealSubject để Proxy có thể truy cập nó. Nó cũng thực hiện các giao diện tương tự như RealSubject để Proxy có thể được sử dụng thay cho RealSubject. Proxy cũng điều khiển truy cập vào RealSubject và có thể tạo hoặc xóa đối tượng này.
* RealSubject : Đây là đối tượng chính mà proxy đại diện.

**Code áp dụng mẫu Proxy**

* Khởi tạo Interface ProxySanPham trong model SanPham.cs và tạo class Proxy kế thừa Interface ProxySanPham và viết hàm addSanPhamDB để kiểm tra loại sản phẩm được tạo có đúng yêu cầu theo mẫu Virtual proxy
* namespace Code\_CH.Models
* {
* using System;
* using System.Collections.Generic;
* interface ProxySanPham
* {
* void addSanPhamDB(CuaHangEntities database);
* }
* public partial class SanPham : ProxySanPham
* {
* [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
* public SanPham()
* {
* this.ChiTietHoaDons = new HashSet<ChiTietHoaDon>();
* }
* public int maSP { get; set; }
* public Nullable<int> maDSP { get; set; }
* public Nullable<int> sokiemsoat { get; set; }
* public string size { get; set; }
* public Nullable<int> matinhTrang { get; set; }
* [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
* public virtual ICollection<ChiTietHoaDon> ChiTietHoaDons { get; set; }
* public virtual DauSanPham DauSanPham { get; set; }
* public virtual TinhTrangSanPham TinhTrangSanPham { get; set; }
* public void addSanPhamDB(CuaHangEntities database)
* {
* database.SanPhams.Add(this);
* database.SaveChanges();
* }
* }
* public class Proxy : ProxySanPham
* {
* private SanPham \_sanpham;
* public Proxy(SanPham sp)
* {
* \_sanpham = sp;
* }
* public void addSanPhamDB(CuaHangEntities database)
* {
* string notAllowSizeS = "S";
* string notAllowSizeM = "M";
* string notAllowSizeL = "L";
* string notAllowSizeXL = "XL";
* var checkSizeS = \_sanpham.size.Contains(notAllowSizeS);
* var checkSizeM = \_sanpham.size.Contains(notAllowSizeM);
* var checkSizeL = \_sanpham.size.Contains(notAllowSizeL);
* var checkSizeXL = \_sanpham.size.Contains(notAllowSizeXL);
* if (checkSizeS == true || checkSizeM == true || checkSizeL == true || checkSizeXL == true)
* {
* \_sanpham.addSanPhamDB(database);
* }
* }
* }
* }
* Thực thi mẫu pro trên ThêmChiTietSanPham của HomeController
* public ActionResult ThemChiTietSanPham(int id)
* {
* var flag = database.DauSanPhams.Where(a => a.maDSP == id).SingleOrDefault();
* SanPham dsp = new SanPham();
* return View(dsp);
* }
* [HttpPost]
* public ActionResult ThemChiTietSanPham(SanPham sanpham, int id, int sks, string siz)
* {
* var DauSanPham = database.DauSanPhams.Where(a => a.maDSP == id).SingleOrDefault();
* var checkSoLuong = database.SanPhams.Where(a => a.maDSP == id).Count();
* int soluongSP = sks;
* if (checkSoLuong == 0)
* {
* for (int i = 1; i <= soluongSP; i++)
* {
* var a = new SanPham();
* a.maSP = id;
* a.maDSP = DauSanPham.maDSP;
* a.sokiemsoat = i;
* a.size = siz;
* a.matinhTrang = 1;
* /\* database.SanPhams.Add(a);\*/
* // design pattern Proxy
* Proxy proxy = new Proxy(a);
* proxy.addSanPhamDB(database);
* }
* /\*----- Cap nhat so luong -----\*/
* DauSanPham.soluongSP += soluongSP;
* ViewBag.ThemSach\_message\_suss = "Thêm thành công!";
* return RedirectToAction("DauSanPham", "Home");
* }
* else
* {
* for (int i = checkSoLuong + 1; i <= checkSoLuong + soluongSP; i++)
* {
* var a = new SanPham();
* a.maDSP = id;
* a.sokiemsoat = i;
* a.size = siz;
* a.matinhTrang = 1;
* /\* database.SanPhams.Add(a);\*/
* Proxy proxy = new Proxy(a);
* proxy.addSanPhamDB(database);
* }
* /\*----- Cap nhat so luong -----\*/
* DauSanPham.soluongSP += soluongSP;
* database.SaveChanges();
* ViewBag.ThemSach\_message\_suss = "Thêm thành công!";
* return RedirectToAction("DauSanPham", "Home");
* }
* }
  + 1. Prototype Pattern

**Prototype pattern** là một trong những **Creational pattern**. Nó có nhiệm vụ khởi tạo một đối tượng bằng cách **clone** một đối tượng đã tồn tại thay vì khởi tạo với từ khoá **new**. Đối tượng mới là một bản sao có thể giống 100% với đối tượng gốc, chúng ta có thể thay đổi dữ liệu của nó mà không ảnh hưởng đến đối tượng gốc.

Prototype Pattern được dùng khi việc tạo một object tốn nhiều chi phí và thời gian trong khi bạn đã có một object tương tự tồn tại.

Diagram

Description automatically generatedTrong Java cung cấp mẫu prototype pattern này bằng việc implement interface [**Cloneable**](https://gpcoder.com/2361-object-cloning-trong-java/) và sử dụng method **clone()** để tạo object có đầy đủ thuộc tính của đối tượng ban đầu. Các bạn có thể xem lại bài viết về [Object cloning trong java](https://gpcoder.com/2361-object-cloning-trong-java/) để hiểu thêm về interface Clonable trước khi tiếp tục xem phần tiếp theo của bài viết này.

Một Prototype Pattern gồm các thành phần cơ bản sau:

* **Prototype**: khai báo một class, interface hoặc abtract class cho việc clone chính nó.
* **ConcretePrototype** class: các lớp này thực thi interface (hoặc kế thừa từ lớp abstract) được cung cấp bởi Prototype để copy (nhân bản) chính bản thân nó. Các lớp này chính là thể hiện cụ thể phương thức clone(). Lớp này có thể không cần thiết nếu: Prototype là một class và nó đã implement việc clone chính nó.
* **Client** class: tạo mới object bằng cách gọi Prototype thực hiện clone chính nó.

**Code áp dụng mẫu prototype**

* Tạo interface DauSanPhamPrototype để bở tạo một clone Object để nhân bản Object
* //------------------------------------------------------------------------------
* // <auto-generated>
* // This code was generated from a template.
* //
* // Manual changes to this file may cause unexpected behavior in your application.
* // Manual changes to this file will be overwritten if the code is regenerated.
* // </auto-generated>
* //------------------------------------------------------------------------------
* namespace Code\_CH.Models
* {
* using System;
* using System.Collections.Generic;
* using System.ComponentModel.DataAnnotations.Schema;
* using System.Web;
* public interface DauSanPhamPrototype
* {
* DauSanPhamPrototype Clone();
* }
* public class DauSanPham : DauSanPhamPrototype
* {
* [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
* public DauSanPham()
* {
* this.SanPhams = new HashSet<SanPham>();
* }
* public int maDSP { get; set; }
* public string tenSP { get; set; }
* public string loaiSP { get; set; }
* public string hinhanhSP { get; set; }
* public Nullable<int> soluongSP { get; set; }
* public Nullable<double> giaSP { get; set; }
* public string hinhanhphu1 { get; set; }
* public string hinhanhphu2 { get; set; }
* public string hinhanhphu3 { get; set; }
* public string hinhanhphu4 { get; set; }
* public string hinhanhphu5 { get; set; }
* public string ghichu { get; set; }
* [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
* [NotMapped]
* public HttpPostedFileBase imageUploader { get; set; }
* [NotMapped]
* public HttpPostedFileBase imageUploader2 { get; set; }
* [NotMapped]
* public HttpPostedFileBase imageUploader3 { get; set; }
* [NotMapped]
* public HttpPostedFileBase imageUploader4 { get; set; }
* [NotMapped]
* public HttpPostedFileBase imageUploader5 { get; set; }
* [NotMapped]
* public HttpPostedFileBase imageUploader6 { get; set; }
* public virtual ICollection<SanPham> SanPhams { get; set; }
* public DauSanPhamPrototype Clone()
* {
* DauSanPham dausanpham = new DauSanPham();
* dausanpham.maDSP = maDSP;
* dausanpham.tenSP = tenSP;
* dausanpham.giaSP = giaSP;
* dausanpham.soluongSP = soluongSP;
* dausanpham.hinhanhphu1 = hinhanhphu1;
* dausanpham.hinhanhphu2 = hinhanhphu2;
* dausanpham.hinhanhphu3 = hinhanhphu3;
* dausanpham.hinhanhphu4 = hinhanhphu4;
* dausanpham.hinhanhphu5 = hinhanhphu5;
* dausanpham.hinhanhSP = hinhanhSP;
* return dausanpham;
* }
* }
* }
* Áp dụng mẫu Prototype để nhân bản sản phẩm đã có mà không trùng ID
* [HttpGet]
* [validateAntiForgrytoken]
* public ActionResult DPrototype(int id)
* {
* DauSanPham dausanpham = database.DauSanPhams.Find(id);
* var cloneDauSanPham = dausanpham.Clone();
* database.DauSanPhams.Add((DauSanPham)cloneDauSanPham);
* database.SaveChanges();
* return RedirectToAction("DauSanPham", "Home");
* }
  + 1. Iterator Pattern

**Iterator Pattern** là một trong những Pattern thuộc nhóm hành vi (Behavior Pattern). Nó được sử dụng để “Cung cấp một cách thức truy cập tuần tự tới các phần tử của một đối tượng tổng hợp, mà không cần phải tạo dựng riêng các phương pháp truy cập cho đối tượng tổng hợp này”.

Nói cách khác, một Iterator được thiết kế cho phép xử lý nhiều loại tập hợp khác nhau bằng cách truy cập những phần tử của tập hợp với cùng một phương pháp, cùng một cách thức định sẵn, mà không cần phải hiểu rõ về những chi tiết bên trong của những tập hợp này.

Chúng ta có thể thấy Interator Pattern được áp dụng trong java với Interface iterator trong gói **java.util.Iterator**. Interface này định nghĩa các phương thức sau:

* Hàm next() : trả về phần tử kế tiếp trong tập hợp
* Hàm hasNext() : trả về giá trị True nếu vẫn còn phần tử trong tập hợp và trả về false trong trường hợp ngược lại.

Đó là cách Iterator làm việc. Nó cung cấp một giao diện đơn giản, nhất quán để làm việc với các tập hợp khác nhau.

Diagram

Description automatically generated

Các thành phần tham gia mẫu Iterator:

* **Aggregate**: là một interface định nghĩa định nghĩa các phương thức để tạo Iterator object.
* **ConcreteAggregate**: cài đặt các phương thức của Aggregate, nó cài đặt interface tạo Iterator để trả về một thể hiện của ConcreteIterator thích hợp.
* **Iterator**: là một interface hay abstract class, định nghĩa các phương thức để truy cập và duyệt qua các phần tử.
* **ConcreteIterator**: cài đặt các phương thức của Iterator, giữ index khi duyệt qua các phần tử.
* **Client**: đối tượng sử dụng Iterator Pattern, nó yêu cầu một iterator từ một đối tượng collection để duyệt qua các phần tử mà nó giữ. Các phương thức của iterator được sử dụng để truy xuất các phần tử từ collection theo một trình tự thích hợp.

Code áp dụng mẫu Iterator

* Tạo class IIteratorTaiKhoan thực hiện hành vi để xử lý tập hợp các Tài khoản khách hàng.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Code\_CH.Models;

namespace Code\_CH.Controllers

{

public interface IIteratorTaiKhoan

{

// Hóa đơn

TaiKhoanKhachHang First();

TaiKhoanKhachHang Next();

bool IsDone { get; }

TaiKhoanKhachHang CurrentItem { get; }

}

public class TaiKhoansIIterator : IIteratorTaiKhoan

{

public List<TaiKhoanKhachHang> \_listTaiKhoan { get; } = new List<TaiKhoanKhachHang>();

int current = 0;

int step = 1;

public TaiKhoansIIterator(List<TaiKhoanKhachHang> listTaiKhoan)

{

\_listTaiKhoan = listTaiKhoan;

}

public bool IsDone

{

get { return current >= \_listTaiKhoan.Count; }

}

public TaiKhoanKhachHang CurrentItem => \_listTaiKhoan[current];

TaiKhoanKhachHang IIteratorTaiKhoan.CurrentItem => throw new NotImplementedException();

public TaiKhoanKhachHang First()

{

current = 0;

return \_listTaiKhoan[current];

}

public TaiKhoanKhachHang Next()

{

current += step;

if (!IsDone)

return \_listTaiKhoan[current];

else

return null;

}

TaiKhoanKhachHang IIteratorTaiKhoan.First()

{

current = 0;

return \_listTaiKhoan[current];

}

TaiKhoanKhachHang IIteratorTaiKhoan.Next()

{

current += step;

if (!IsDone)

return \_listTaiKhoan[current];

else

return null;

}

}

}

-Khai báo và thực thi ở DS\_TaiKhoan HomeCOntroller thay đổi mã tình trạng tài khoản khách hàng từ block thành active

public ActionResult DS\_TaiKhoan(string search)

{

var tkkh = database.TaiKhoanKhachHangs.Where(a => a.maKH != 0).ToList();

ViewBag.soluongtaikhoan = tkkh.Count();

if(search != null)

{

return View(database.TaiKhoanKhachHangs.Where(a => a.tenKH.StartsWith(search) || a.emailKH.StartsWith(search) || a.sodienthoaiKH.StartsWith(search)).ToList());

}

else

{

IIteratorTaiKhoan iterator = new TaiKhoansIIterator(tkkh);

var taikhoan = iterator.First();

while (!iterator.IsDone)

{

if (taikhoan.maTT != 2)

{

taikhoan.TinhTrangTaiKhoan.tenTinhTrang = "block";

database.SaveChanges();

}

taikhoan = iterator.Next();

}

return View(tkkh);

/\* return View(database.TaiKhoanKhachHangs);\*/

}

}

1. XÂY DỰNG VÀ TRIỂN KHAI
   1. Mockup

Shape, polygon

Description automatically generated

*Hình 07: Trang Home*

A picture containing shape

Description automatically generated

*Hình 08: Trang Product Detail*

Graphical user interface, application

Description automatically generated

*Hình 09: Trang Checkout*

Graphical user interface

Description automatically generated

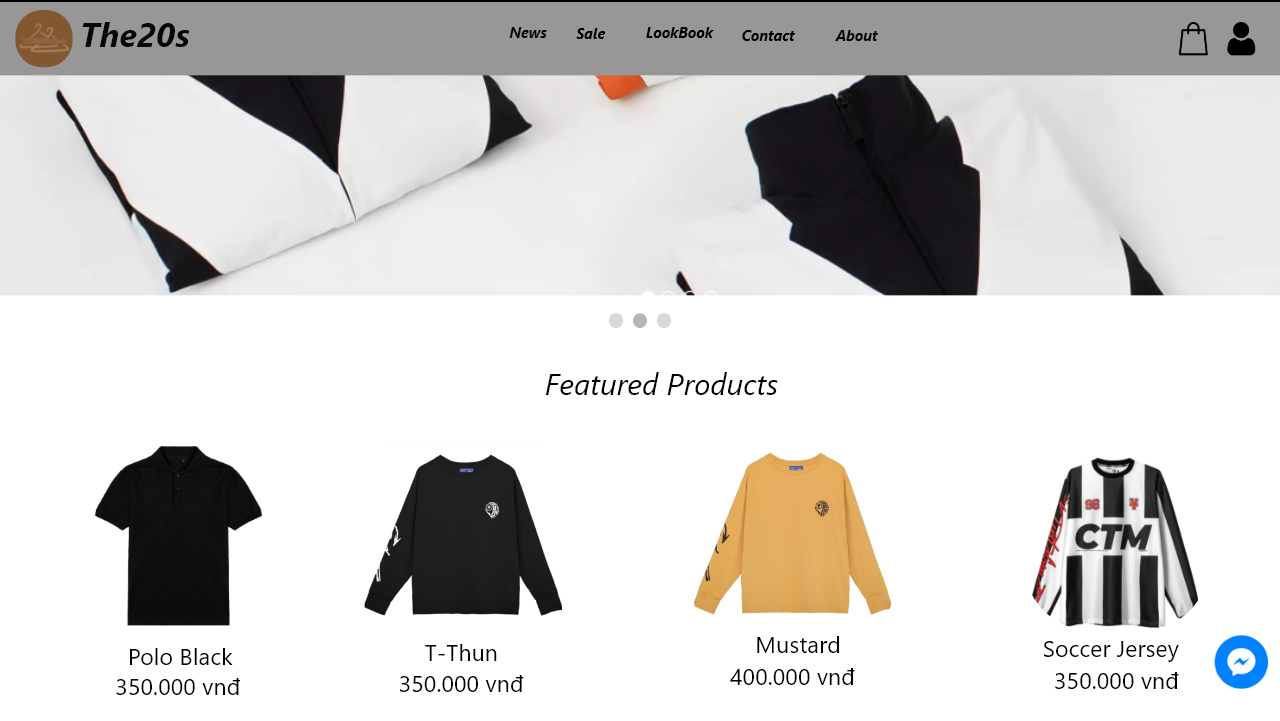
*Hình 10: Trang Payment*

Graphical user interface

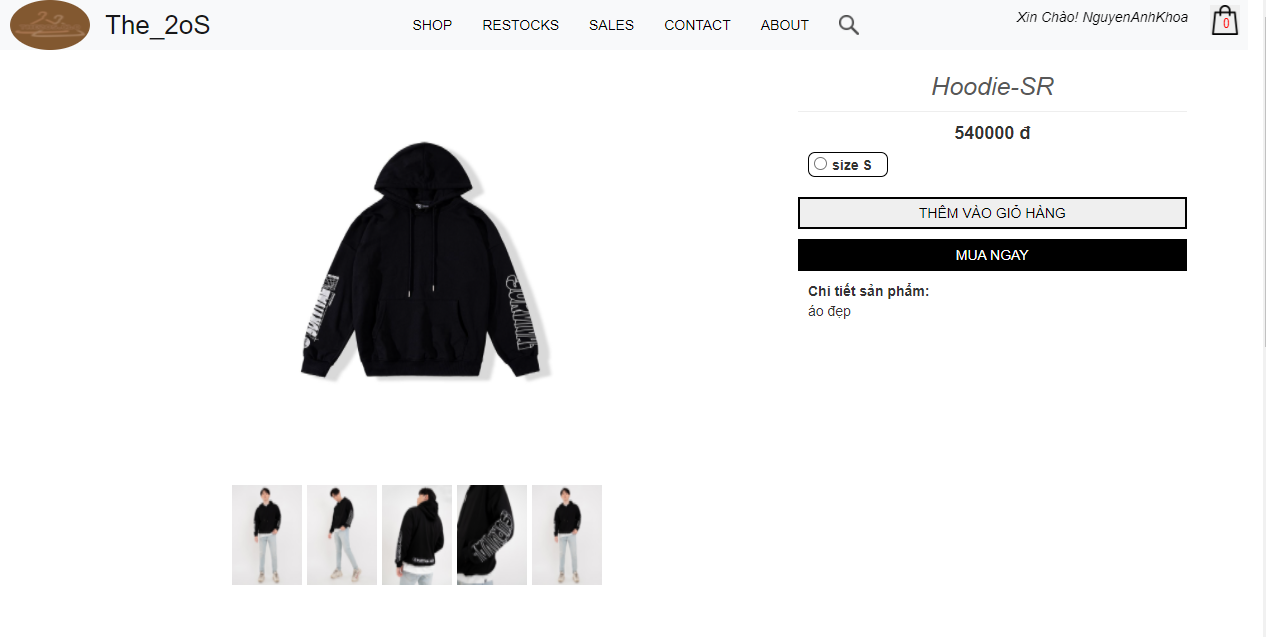
Description automatically generated

*Hình 11: Trang Cart*

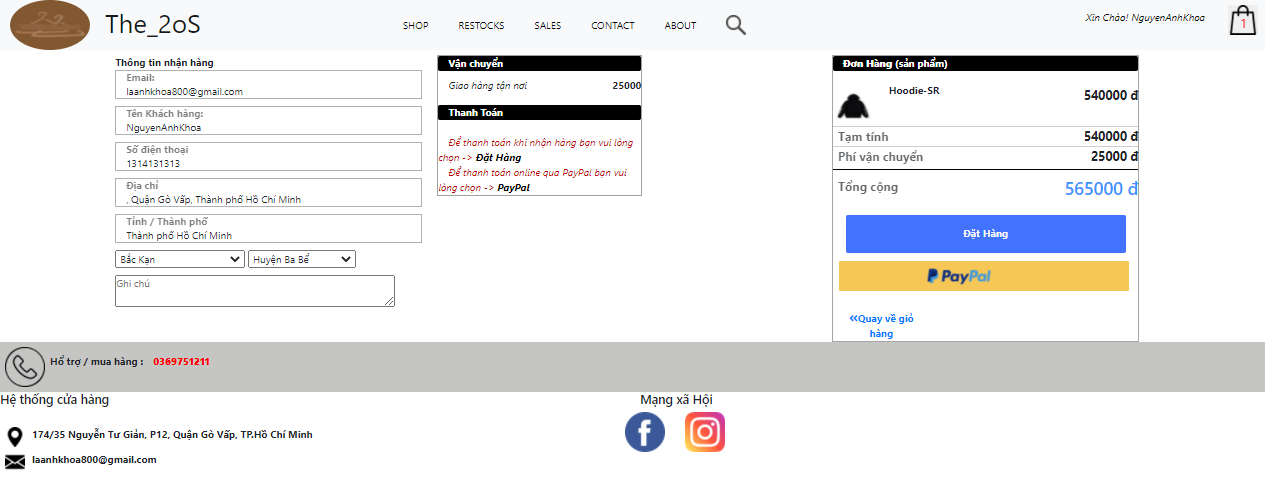
* 1. Giao diện của ứng dụng



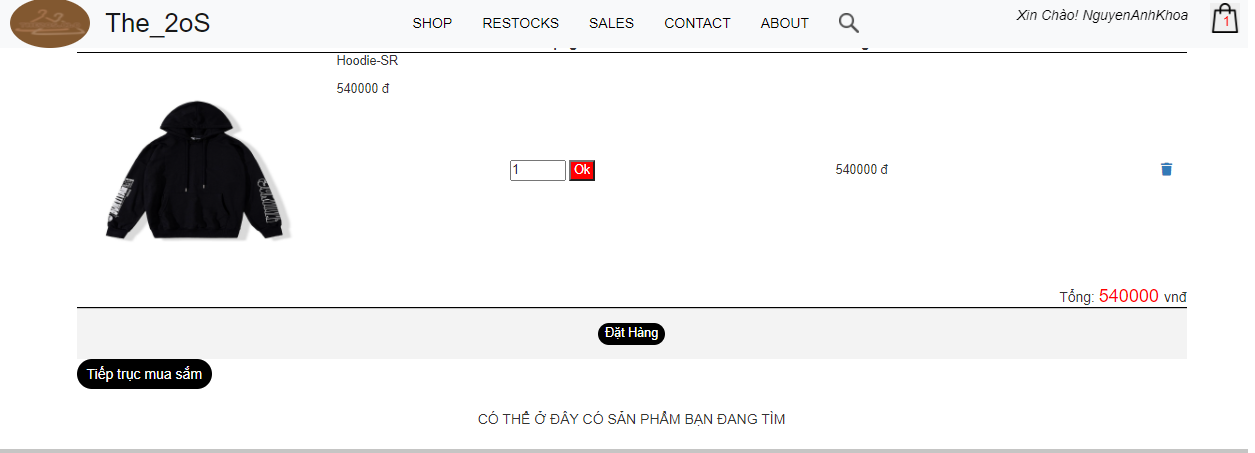
*Hình 12: Trang Home*



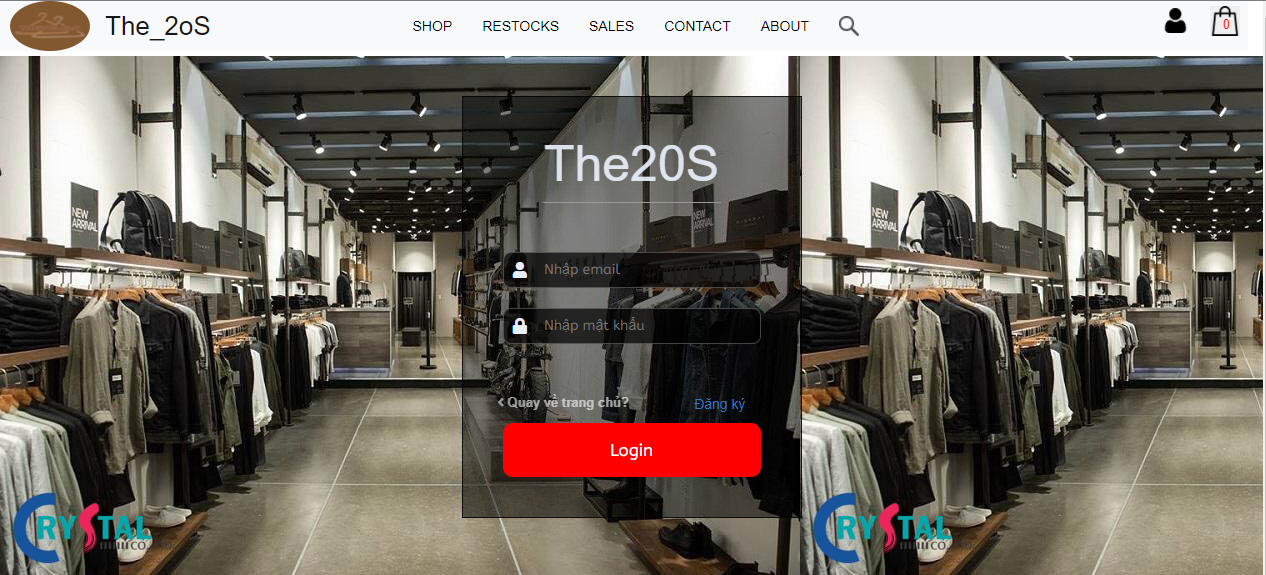
*Hình 13: Product detail*



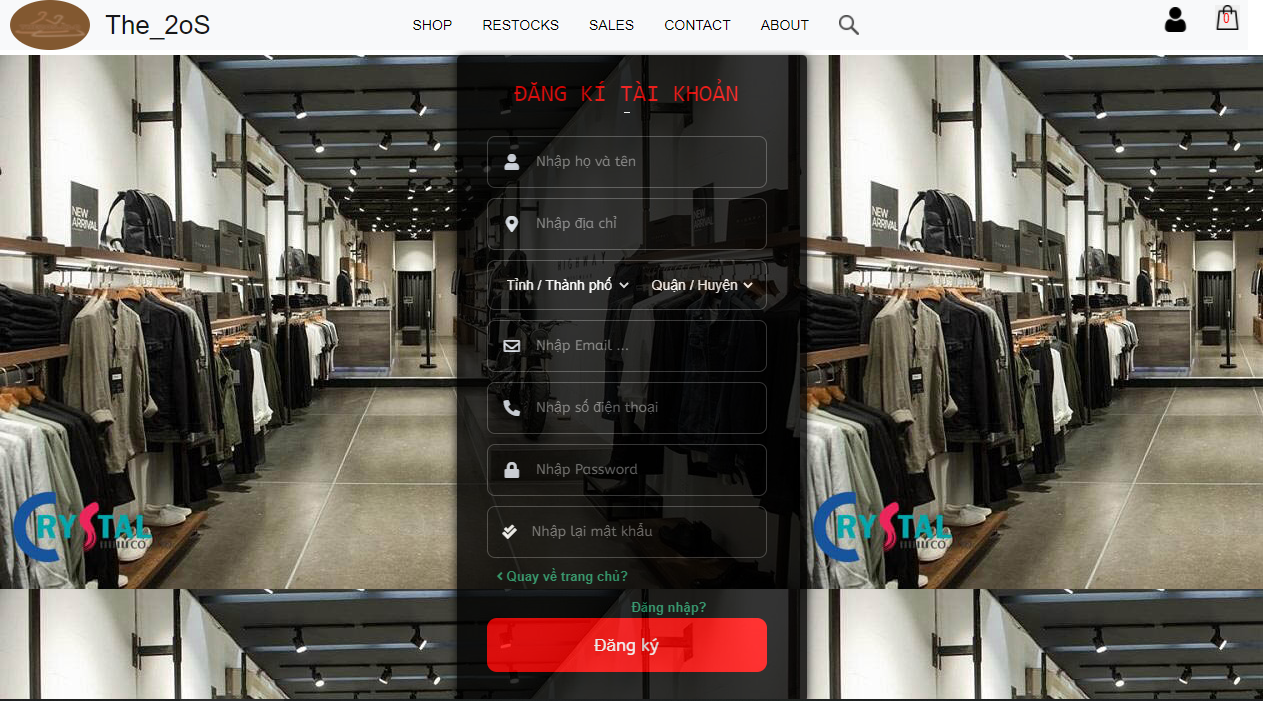
*Hình 14: Trang Checkout*



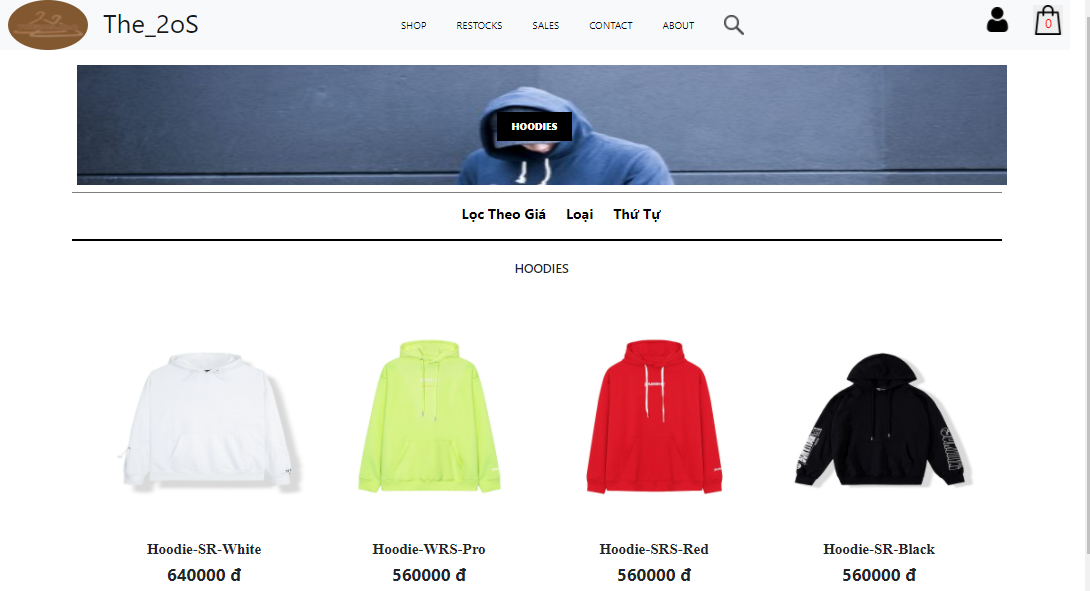
*Hình 16: Trang Cart*



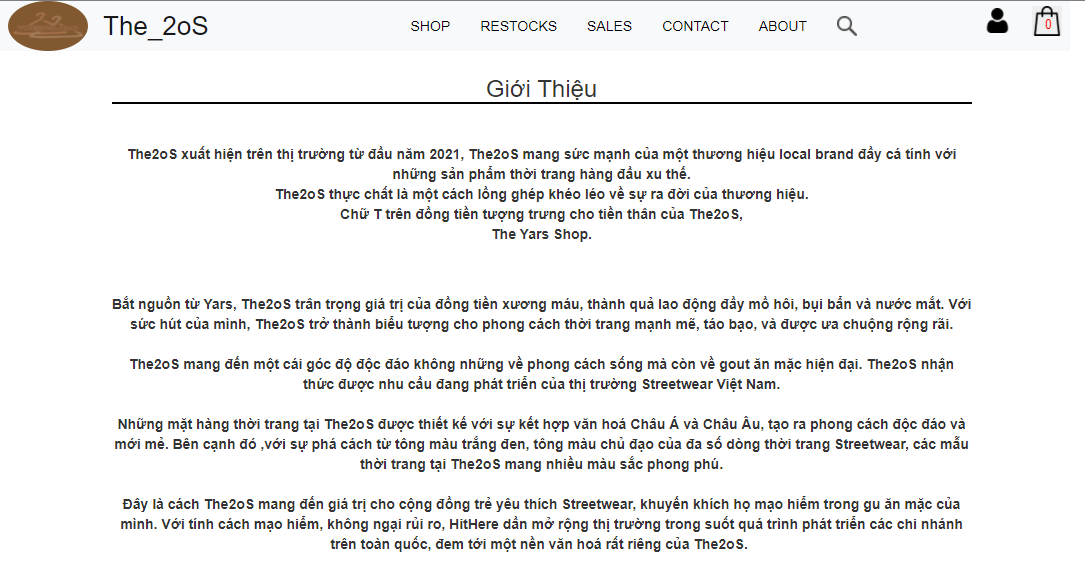
*Hình 17: Login*



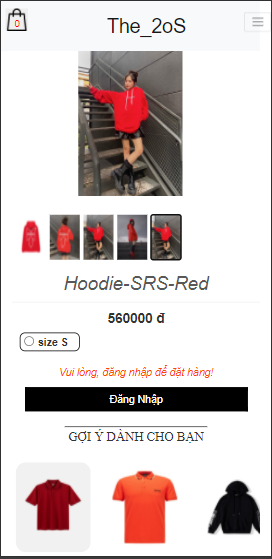
*Hình 18: Register*



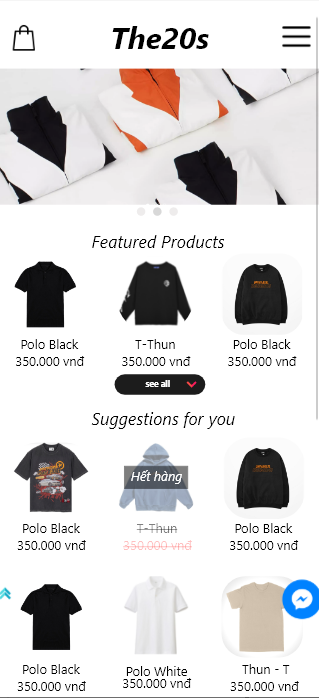
*Hình 19: Sản Phẩm Hoodies*



*Hình 20: About*



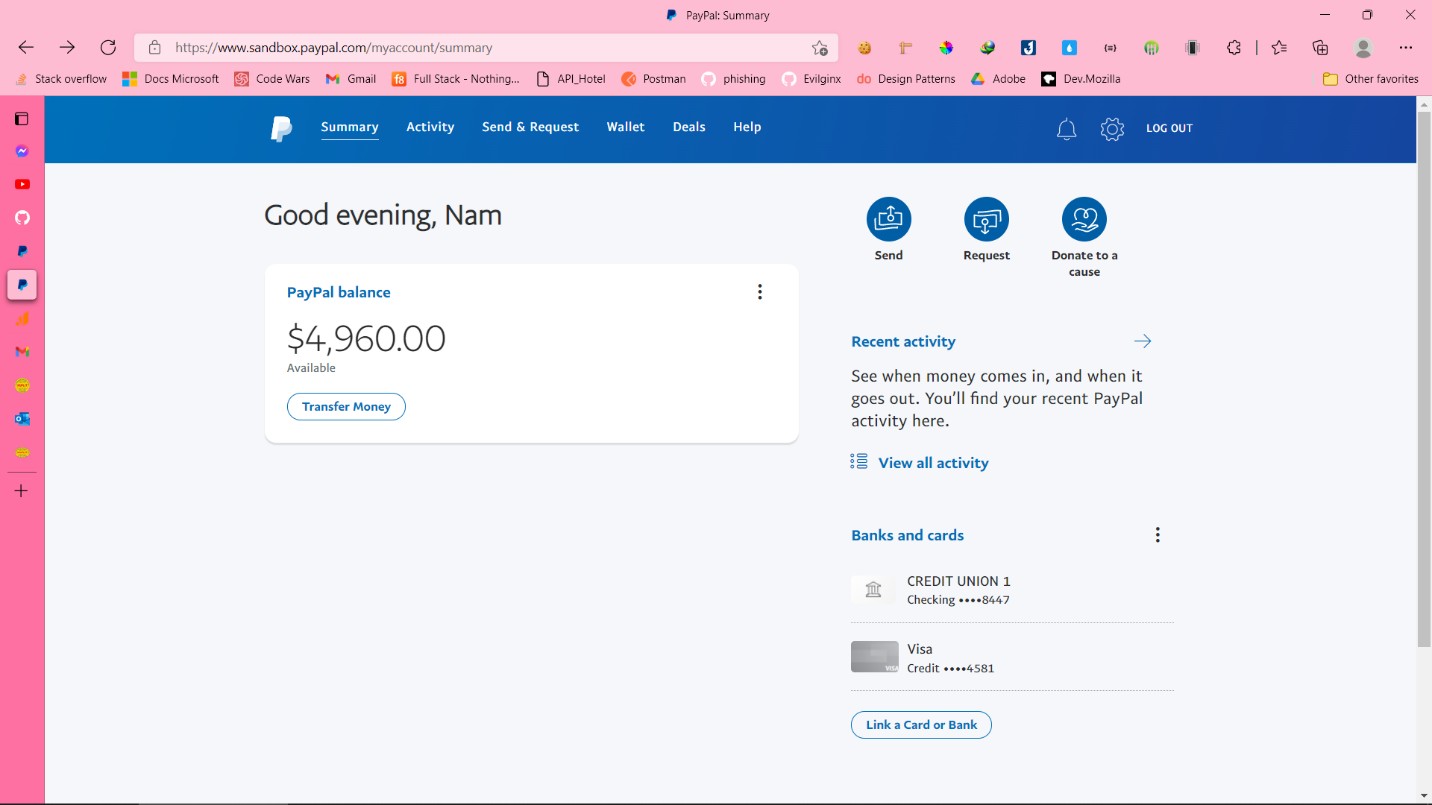
*Hình 21: Giao diện mobile*



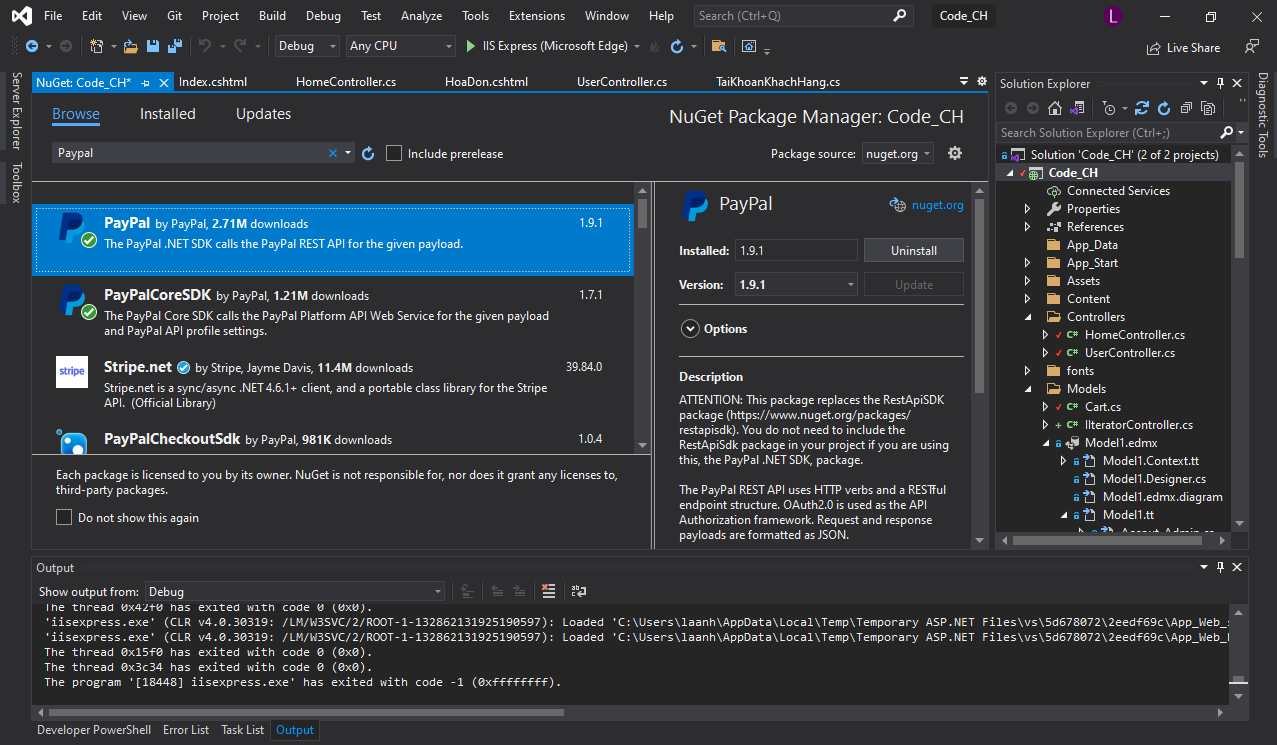
*Hình 22: Giao diện Home mobile*

* 1. Thanh toán trực tiếp

Tích hợp Paypal vào ASP.NET MVC5  
**Bước 1**: Vào trang Sandbox accounts - PayPal Developer tạo account

  
**Bước 2**: Truy cập vào trang https://www.sandbox.paypal.com/ và login vào account mới tạo

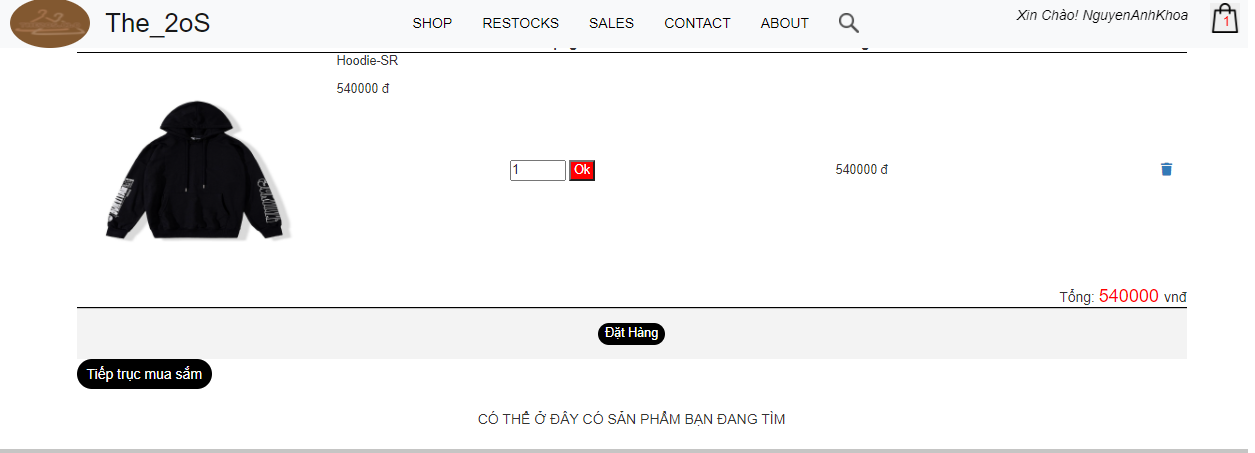
  
**Bước 3**: Tạo Rest API app

**Bước 4**: chạy cmd $ dotnet add package PayPal -v 2.0.0-rc2 hoặc mởManage Nuget package trong Visual  
Studio, tìm và cài đặt Paypal

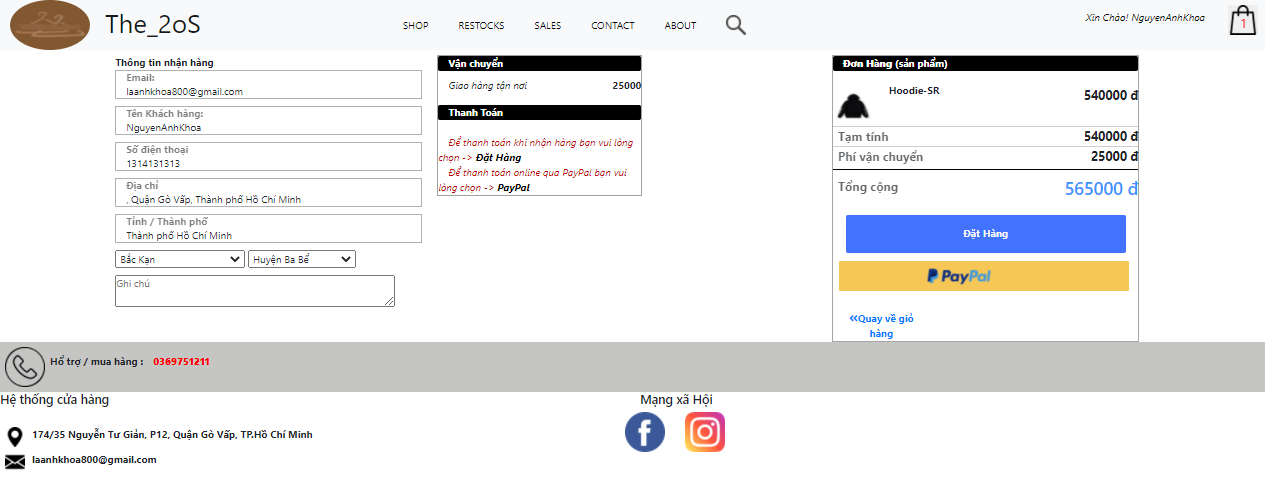
**Bước 5**: Cài đặt thư viện  
using System.Linq;  
using PayPal.Core;  
using PayPal.Payment;

**Bước 6**: Cài đặt cấu hình cho phần thanh toán  
var environment = new SandboxEnvironment(ClientAppId, SecretKey);  
var client = new PayPalHttpClient(environment);  
#region Create payment (just example)  
var itemList = new ItemList()  
{  
Items = new List<Item>()  
};  
var total = myCart.Sum(p => p.Total);  
foreach (var item in myCart)  
{  
itemList.Items.Add(new Item()  
{  
Name = item.ProductName,  
Currency = "USD",  
Price = item.UnitPrice.ToString(),  
Quantity = item.Quantity.ToString(),  
Sku = "sku",  
Tax = "0"  
});  
}  
#endregion  
var payment = new Payment()  
{  
Intent = "sale",  
Transactions = new List<Transaction>()  
{  
new Transaction()  
{  
Amount = new Amount()  
{  
Total = total.ToString(),  
Currency = "USD",  
Details = new AmountDetails  
{  
Tax = "0",  
Shipping = "0",  
Subtotal = total.ToString()  
}  
},  
ItemList = itemList,  
Description = "Invoice #001",  
InvoiceNumber = "INV0001"  
}  
},  
RedirectUrls = new RedirectUrls()  
{  
CancelUrl = "/Paypal/CheckoutFail",  
ReturnUrl = "/Paypal/CheckoutSuccess"  
},  
Payer = new Payer()  
{  
PaymentMethod = "paypal"  
}  
};  
• ClientAppId, SecretKey: cấu hình của paypal  
• RedirectUrls: Xác định URL của một số hành động để xử lý kết quả thanh toán.

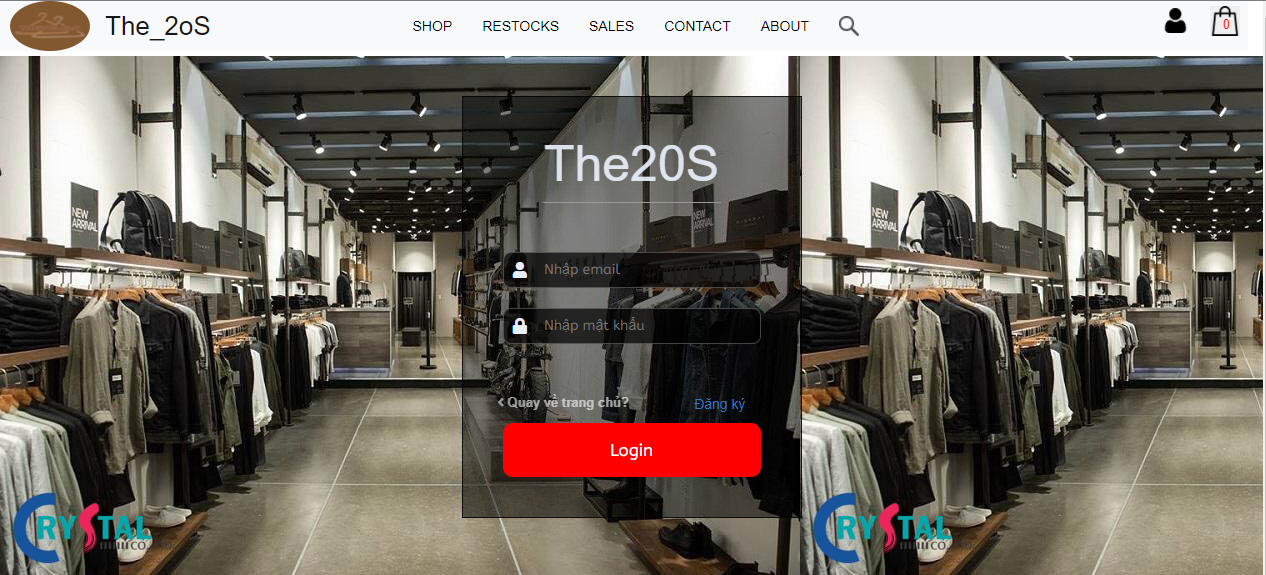
**Bước 7**: Tiến trình thanh toán  
PaymentCreateRequest request = new PaymentCreateRequest();  
request.RequestBody(payment);  
try  
{  
var response = await client.Execute(request);  
var statusCode = response.StatusCode;  
Payment result = response.Result<Payment>();  
var links = result.Links.GetEnumerator();  
string paypalRedirectUrl = null;  
while (links.MoveNext())  
{  
LinkDescriptionObject lnk = links.Current;  
if (lnk.Rel.ToLower().Trim().Equals("approval\_url"))  
{  
//saving the payapalredirect URL to which user will be redirected for payment  
paypalRedirectUrl = lnk.Href;  
}  
}  
return Redirect(paypalRedirectUrl);  
}  
catch (HttpException httpException)  
{  
var statusCode = httpException.StatusCode;  
var debugId = httpException.Headers.GetValues("PayPal-Debug-Id").FirstOrDefault();  
//Process when Checkout with Paypal fails  
return Redirect("/Paypal/CheckoutFail");  
}



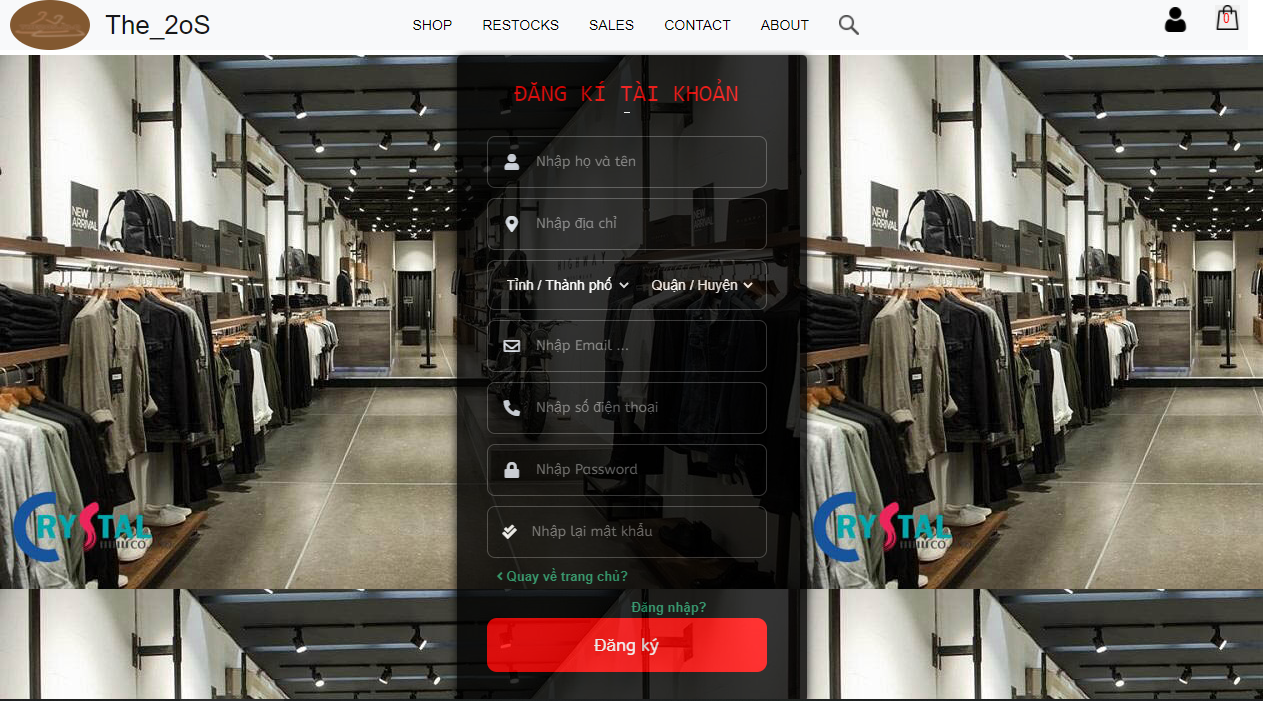
*Hình 23: Cart*



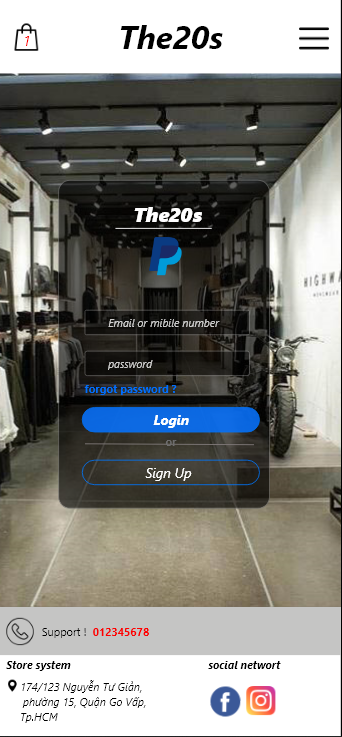
*Hình 24: Checkout*



*Hình 25: Login*



*Hình 26: Register*



*Hình 27: PayPal*



*Hình 28: Receipt*

1. TỔNG KẾT
   1. Những kết quả đạt được

Trong quá trình thực hiện và nghiên cứu về đề tài này nhóm chúng em đã cố gắng hết sức để tìm hiểu xây dựng và cài đặt chương trình tuy nhiên thời gian cho phép có hạn nên có thể chúng em chưa giải quyết được tất cả các vấn đề đặt ra. Em rất mong nhận được sự thông cảm của giảng viên. Em xin chân thành cảm ơn!

Trang web đã đạt được những mục tiêu cơ bản như:

* Hiểu và nắm bắt được các cách thiết kế Web động cũng như cách tổ chức cơ sở dữ liệu.
* Giao diện thân thiện và gần gũi với người dùng.
* Giới thiệu sản phẩm đến với khách hàng.
* Cho phép khách hàng tra cứu thông tin về sản phẩm theo nhu cầu.
* Tiếp nhận đơn hàng, xuất hoá đơn và xác nhận giao hàng nhanh chóng.
* Thanh toán trực tuyến.
  1. Kết luận

Ngày nay việc ứng dụng thương mại điện tử ở Việt Nam chỉ đáp ứng được một số vấn đề cơ bản về mua bán, chưa được linh hoạt. Việc ứng dụng này đòi hỏi phải tốn nhiều thời gian và tiền bạc. Với kiến thức nền tảng đã được học ở trường và bằng sự nỗ lực của nhóm chúng em đã hoàn thành đề tài “Xây dựng Website thời trang”. Mặc dù đã cố gắng và đầu tư rất nhiều nhưng do thời gian cho phép có hạn. Nhóm chúng em rất mong nhận được sự thông cảm và góp ý của giảng viên để đề tài của chúng em được hoàn thiện hơn.

Một lần nữa chúng em xin chân thành cảm ơn thầy Lương Văn Minh

đã tận tình giúp đỡ em trong suốt thời gian thực hiện đề tài này. Em xin chân thành cảm ơn thầy!

1. TÀI LIỆU THAM KHẢO

<https://trustweb.vn/huong-dan-hoc-lap-trinh-asp-net-mvc-5-thiet-ke-website-ban-hang/>

<http://hayvay.blogspot.com/2011/08/aspnet-paypal-viet-chuc-nang-thanh-toan.html>

<https://gpcoder.com/4724-huong-dan-java-design-pattern-iterator/>

<https://gpcoder.com/4413-huong-dan-java-design-pattern-prototype/>

<https://gpcoder.com/5160-huong-dan-java-design-pattern-mvc/>

<https://gpcoder.com/4644-huong-dan-java-design-pattern-proxy/>

https://gpcoder.com/4190-huong-dan-java-design-pattern-singleton/