**TRƯỜNG ĐẠI HỌC ĐÀ LẠT**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO MÔN MẪU THIẾT KẾ**

**XÂY DỰNG WEBSITE BÁN QUẦN ÁO**

Giảng viên hướng dẫn: GV.Đoàn Minh Khuê

Sinh viên thực hiện: 2115255 - Chế Công Quang

Mục Lục

[**1. Giới thiệu 3**](#_257chmd833su)

[**1.1. Mục tiêu đề tài 3**](#_m6sr4ovvucg0)

[**1.2. Lý do chọn đề tài 3**](#_272grsyq9kp4)

[**2. Phân tích yêu cầu 3**](#_ket5mulw975n)

[**2.1. Yêu cầu chức năng 3**](#_3ruc82uryqnh)

[**2.2. Yêu cầu phi chức năng 4**](#_m6tbm6zeicfb)

[**3. Thiết kê hệ thống 4**](#_wpu9xnso4oi2)

[**4. Triển khai hệ thống 4**](#_me0ryyik1ea4)

[**4.1. Công nghệ sử dụng 4**](#_q4nsnklvjya)

[**4.2.Mô tả chi tiết các module 5**](#_w9a9oq41wkcc)

[**4.2.1.Module người dùng 5**](#_h1f447dr1bpt)

[**4.2.2.Module sản phẩm 5**](#_6xlkbze21146)

[**4.2.3.Module giỏ hàng 5**](#_bh0mclpxh6q6)

[**4.3. Một số pattern được sử dụng 5**](#_5yp5cpgrka2k)

[**4.4. Triển khai hệ thống 6**](#_f7wszrmu2n6)

[**4.4.1.Thiết kế giao diện 6**](#_8cm3ejl1s7lu)

[**4.4.2.Triển khai mã nguồn 10**](#_ybez9t3we1cm)

[**5. Kết quả và đánh giá 25**](#_jwhm45xx45bk)

[**5.1. Kết quả đạt được 25**](#_xex9oysuuhr6)

[**5.2. Đánh giá 26**](#_1o5j5evwkbu)

[**6. Kết luận 26**](#_xg581w9tvxza)

1. **Giới thiệu**

## 1.1. Mục tiêu đề tài

Mục tiêu của đề tài là xây dựng một website bán quần áo trực tuyến sử dụng mẫu thiết kế Singleton và Factory nhằm quản lý tốt hơn các đối tượng trong hệ thống và đảm bảo tính nhất quán của dữ liệu.

## 1.2. Lý do chọn đề tài

Việc áp dụng các mẫu thiết kế trong phát triển phần mềm giúp cải thiện cấu trúc mã nguồn, tăng tính bảo trì và mở rộng của hệ thống. Trong dự án này, chúng tôi chọn Singleton để quản lý các tài nguyên dùng chung và Factory để tạo ra các đối tượng một cách linh hoạt

1. **Phân tích yêu cầu**
   1. **Yêu cầu chức năng**

**Đăng ký/Đăng nhập người dùng:**

* Người dùng cần có khả năng tạo tài khoản mới thông qua việc cung cấp thông tin cá nhân như tên, địa chỉ email, mật khẩu, và các thông tin khác.
* Chức năng đăng nhập yêu cầu người dùng nhập đúng thông tin đăng nhập đã được đăng ký (email và mật khẩu).
* Hệ thống phải hỗ trợ tính năng "Quên mật khẩu" để người dùng có thể khôi phục mật khẩu qua email.
* Người dùng sau khi đăng nhập thành công sẽ có phiên làm việc bảo mật và có thể quản lý thông tin cá nhân, xem lịch sử mua hàng, và theo dõi trạng thái đơn hàng.

**Xem danh sách sản phẩm:**

* Người dùng có thể xem danh sách các sản phẩm quần áo hiện có trong shop.
* Danh sách sản phẩm có thể được phân loại theo nhiều tiêu chí như loại sản phẩm (áo, quần, phụ kiện), kích thước, màu sắc, và giá cả.
* Sản phẩm cần hiển thị hình ảnh, tên, mô tả ngắn gọn, giá và các tùy chọn khác (kích thước, màu sắc).

**Tìm kiếm sản phẩm:**

* Hệ thống cần cung cấp thanh công cụ tìm kiếm để người dùng có thể tìm kiếm sản phẩm bằng từ khóa (tên sản phẩm, loại sản phẩm).
* Tìm kiếm cần trả về kết quả chính xác và có tính năng gợi ý khi người dùng nhập từ khóa.
* Kết quả tìm kiếm nên được sắp xếp theo mức độ liên quan hoặc các tiêu chí khác như giá tăng dần/giảm dần, sản phẩm mới nhất.

**Thêm sản phẩm vào giỏ hàng:**

* Người dùng có thể thêm sản phẩm vào giỏ hàng từ trang danh sách sản phẩm hoặc trang chi tiết sản phẩm.
* Giỏ hàng cần hiển thị danh sách các sản phẩm đã chọn, số lượng, và tổng giá trị đơn hàng.
* Người dùng có thể chỉnh sửa số lượng sản phẩm, xóa sản phẩm khỏi giỏ hàng và tiếp tục mua sắm.

**Thanh toán:**

* Hệ thống cần hỗ trợ quy trình thanh toán an toàn và thuận tiện.
* Người dùng cần cung cấp thông tin giao hàng và lựa chọn phương thức thanh toán (thẻ tín dụng, ví điện tử, chuyển khoản).
* Hệ thống cần gửi xác nhận đơn hàng qua email sau khi hoàn tất thanh toán, và cập nhật trạng thái đơn hàng trong tài khoản người dùng.
  1. **Yêu cầu phi chức năng**

**Hiệu suất:**

* Hệ thống cần được tối ưu hóa để đảm bảo tốc độ truy cập nhanh chóng, ngay cả khi có số lượng lớn người dùng truy cập đồng thời.
* Các trang web cần tải nhanh chóng và phản hồi ngay lập tức với các thao tác của người dùng.
* Cơ sở dữ liệu cần được thiết kế và tối ưu hóa để xử lý hiệu quả các truy vấn và giao dịch.

**Bảo mật:**

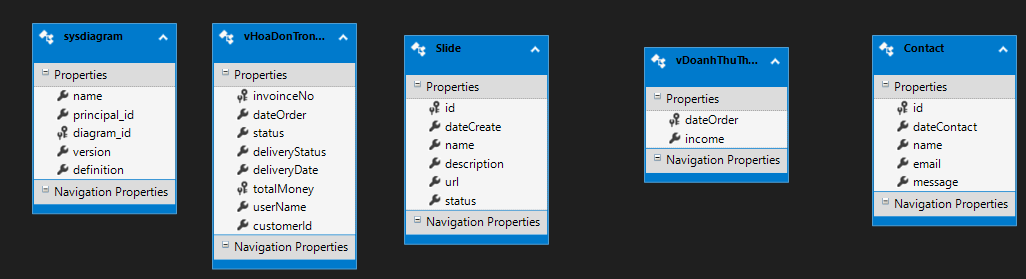
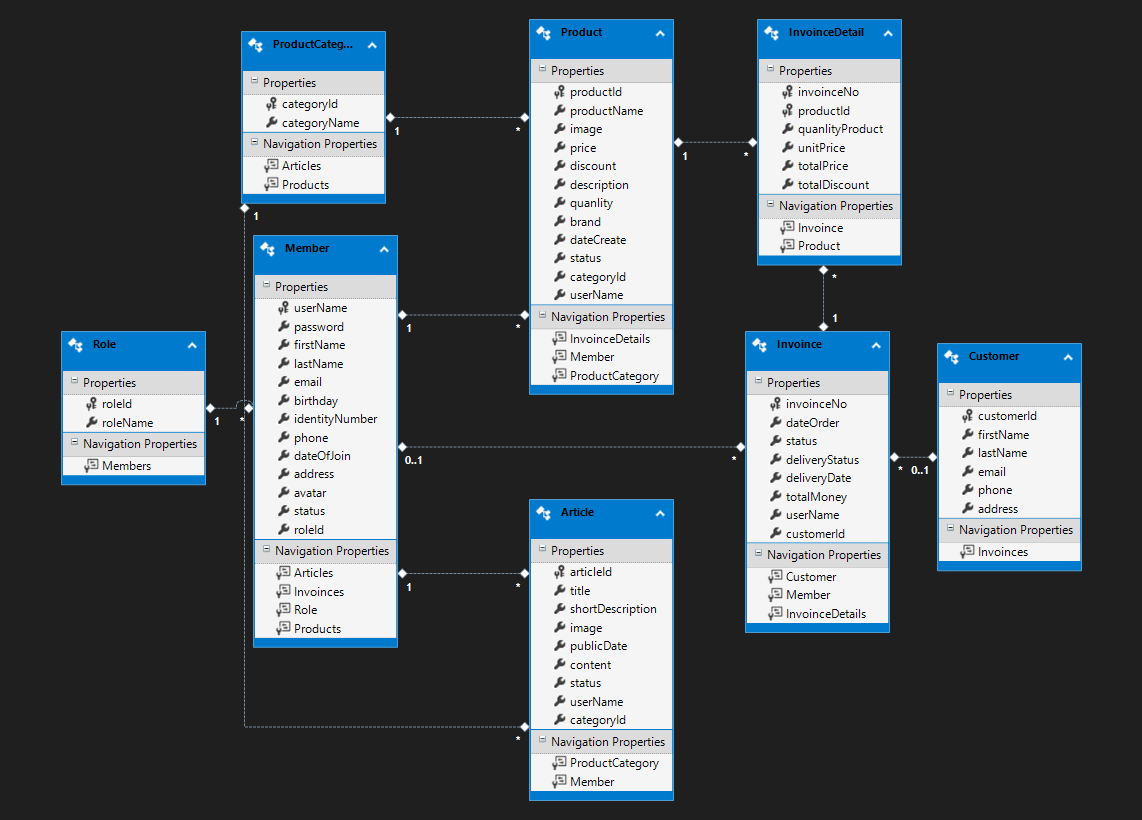
* Thông tin cá nhân của người dùng cần được mã hóa và bảo vệ chống lại các tấn công từ bên ngoài.
* Hệ thống phải tuân thủ các tiêu chuẩn bảo mật như HTTPS, mã hóa mật khẩu và thông tin nhạy cảm.
* Cần có cơ chế phát hiện và ngăn chặn các hành vi xâm nhập trái phép, tấn công DDoS, và các lỗ hổng bảo mật khác.

**Tính nhất quán:**

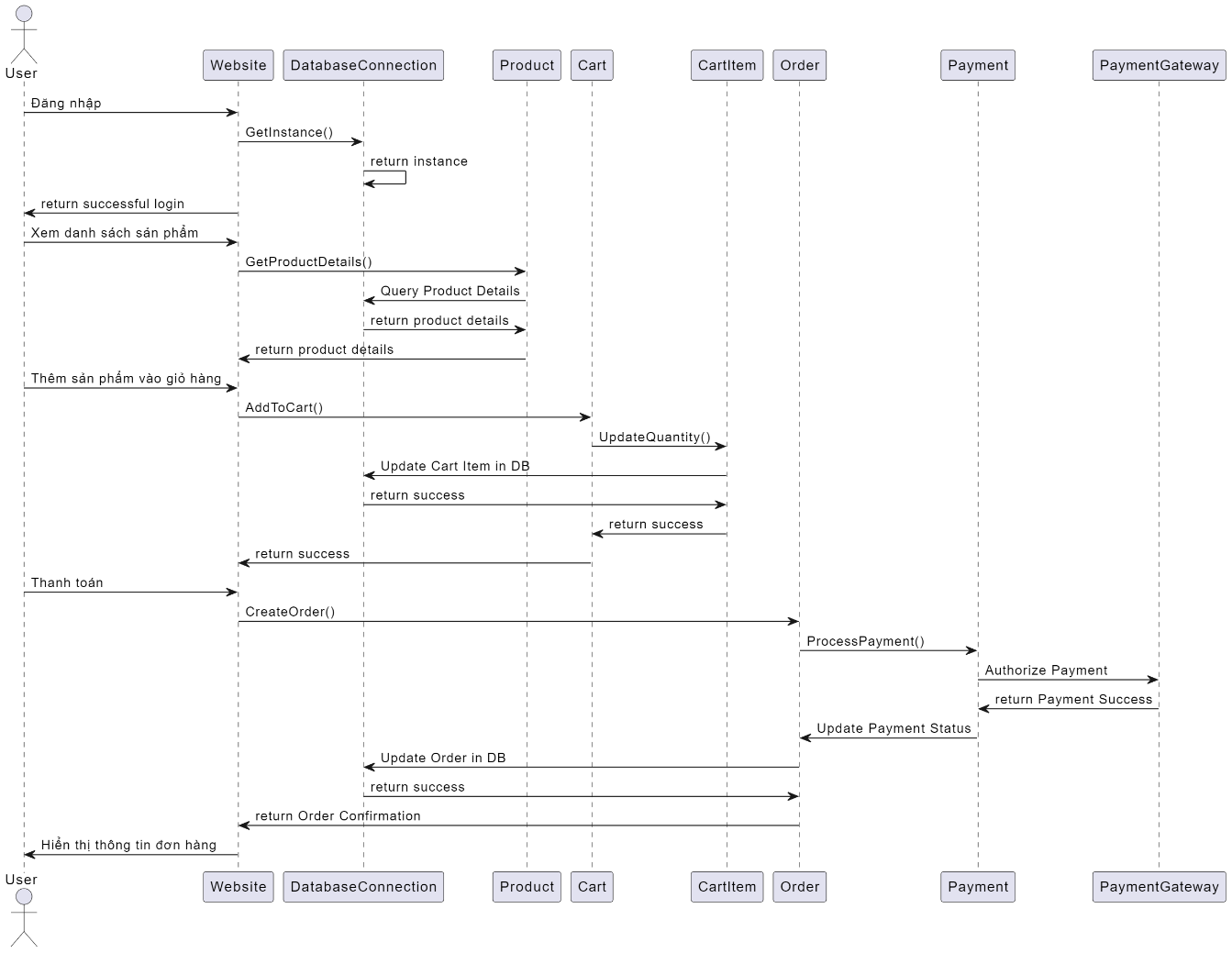
* Dữ liệu hiển thị trên hệ thống cần được cập nhật và phản ánh chính xác trạng thái thực tế của kho hàng và các giao dịch.
* Khi có sự thay đổi dữ liệu (ví dụ: thêm sản phẩm mới, cập nhật thông tin sản phẩm), hệ thống cần đảm bảo rằng tất cả người dùng đều nhìn thấy thông tin mới nhất.
* Các giao dịch cần được xử lý một cách nhất quán, đảm bảo không xảy ra tình trạng lỗi hoặc mất dữ liệu.

1. **Thiết kê hệ thống**

## 3.1.Sơ đồ Class Diagram



## 3.2.Sơ Đồ Sequence Diagram



1. **Triển khai hệ thống**

## 4.1. Công nghệ sử dụng

Visual Studio ASP.NET web Application(.NET Framework)

## 4.2.Mô tả chi tiết các module

### 4.2.1.Module người dùng

* Quản lý thông tin người dùng: Bao gồm các chức năng như đăng ký tài khoản mới, đăng nhập, cập nhật thông tin cá nhân, và quản lý mật khẩu.
* Đăng ký: Người dùng cung cấp thông tin cá nhân như tên, email, mật khẩu. Hệ thống kiểm tra tính hợp lệ và lưu trữ thông tin người dùng vào cơ sở dữ liệu.
* Đăng nhập: Người dùng nhập email và mật khẩu. Hệ thống xác thực thông tin và tạo phiên làm việc cho người dùng.
* Quên mật khẩu: Gửi email khôi phục mật khẩu cho người dùng khi họ yêu cầu.
* Quản lý thông tin cá nhân: Người dùng có thể cập nhật thông tin cá nhân, xem lịch sử mua hàng và trạng thái đơn hàng.

### 4.2.2.Module sản phẩm

* Quản lý thông tin sản phẩm: Bao gồm việc thêm mới, cập nhật, xóa và hiển thị thông tin sản phẩm.
* Thêm sản phẩm: Quản trị viên có thể thêm mới sản phẩm bằng cách cung cấp thông tin chi tiết như tên sản phẩm, mô tả, giá, hình ảnh, kích thước và màu sắc.
* Cập nhật sản phẩm: Quản trị viên có thể cập nhật thông tin của sản phẩm hiện có.
* Xóa sản phẩm: Quản trị viên có thể xóa sản phẩm không còn kinh doanh.
* Hiển thị sản phẩm: Hệ thống hiển thị danh sách sản phẩm theo nhiều tiêu chí khác nhau như loại sản phẩm, giá, kích thước, và màu sắc. Người dùng có thể xem chi tiết từng sản phẩm.

### 4.2.3.Module giỏ hàng

* Quản lý giỏ hàng: Cho phép người dùng thêm sản phẩm vào giỏ hàng, xem và chỉnh sửa giỏ hàng.
* Thêm vào giỏ hàng: Người dùng có thể thêm sản phẩm vào giỏ hàng từ trang danh sách sản phẩm hoặc trang chi tiết sản phẩm.
* Xem giỏ hàng: Hiển thị danh sách sản phẩm trong giỏ hàng, bao gồm tên sản phẩm, số lượng, và giá.
* Chỉnh sửa giỏ hàng: Người dùng có thể thay đổi số lượng sản phẩm hoặc xóa sản phẩm khỏi giỏ hàng.

## 4.3. Một số pattern được sử dụng

**Singleton Pattern:**

Được sử dụng để đảm bảo rằng một lớp chỉ có hoặc một thể hiện duy nhất và cung cấp môt cách tiếp cận toàn cục để lấy thể hiện nó.

Trong đoạn mã, Singleton Pattern đảm bảo chỉ có một thể hiện của ‘menfashionEntities’, giúp quản lý tài nguyên hiệu quả.

**Factory Pattern:**

Được sử dụng để tạo ra các đối tượng mà không cần phải biết rõ về cách chúng được tạo. Điều này giúp tăng tính linh hoạt và giảm sự phụ thuộc giữa các thành phần của hệ thống.

Factory Pattern được thể hiện thông qua các phương thức ‘CreatePayment’, ‘ExecutePayment’, các đối tượng này được tạo theo kiểu factory.

**Repository Pattern:**

Được sử dụng trong các ứng dụng làm việc với cơ sở dữ liệu. Giúp tạo ra một lớp trung gian giữa ứng dụng và cơ sở dữ liệu để tách biệt logic truy cập dữ liệu.

Trong đoạn mã, Repository Pattern sử dụng ‘menfashionEntities’ để quản lý dữ liệu, các thao tác thêm, cập nhật và thay đổi dữ liệu.

**Template Method Pattern:**

Địng nghĩa một bản khuôn (temple) cho một thuật toán và để các bước cụ thể của thuật toán được triển khai bởi các lớp con.

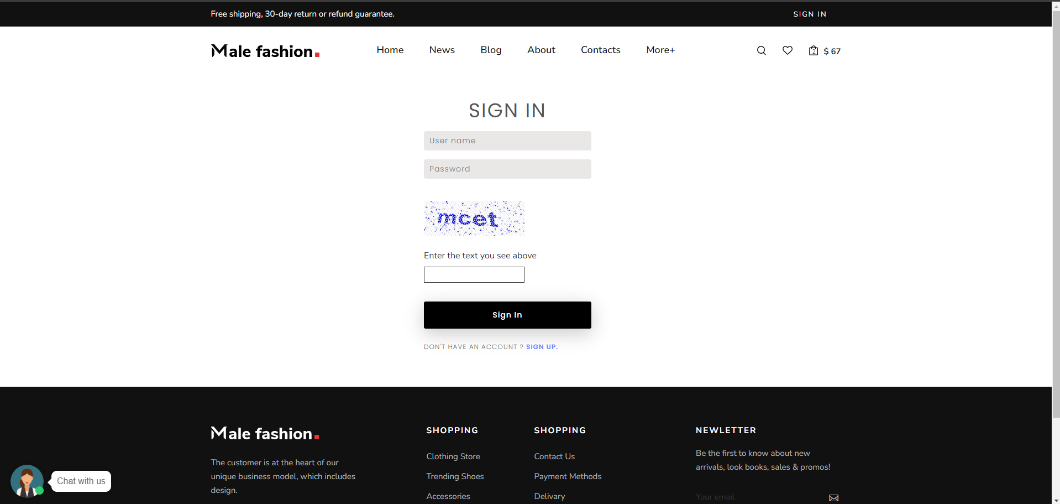
Trong đoạn mã, Template Method Pattern được sử dụng trong phương thức ‘EditProfile’, đảm bảo tính nhất quán trong quá trình chỉnh sửa hồ sơ người dùng, cho phép các bước cụ thể của thuật toán được tùy chỉnh mà không làm thay đổi cấu trúc tổng thể.

## 4.4. Triển khai hệ thống

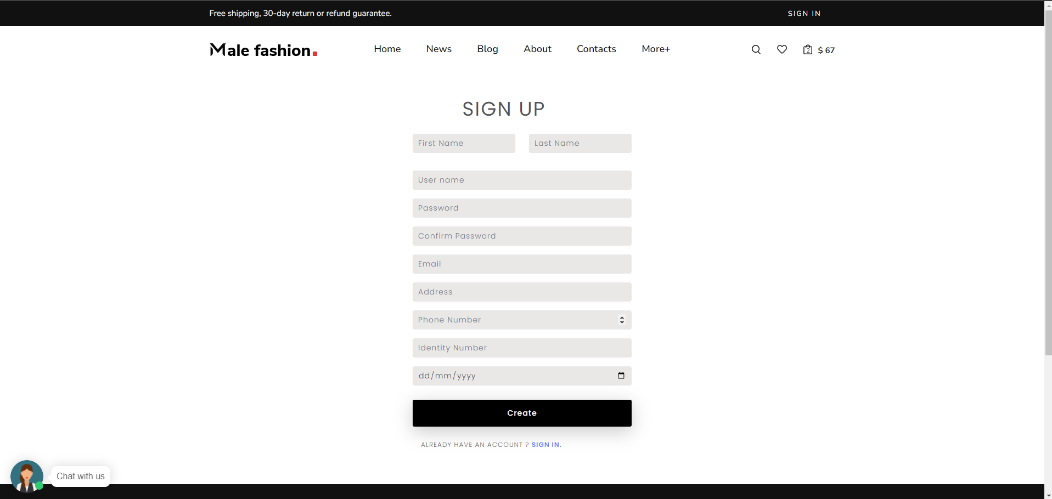
### 4.4.1.Thiết kế giao diện

**Quản lý người dùng**:

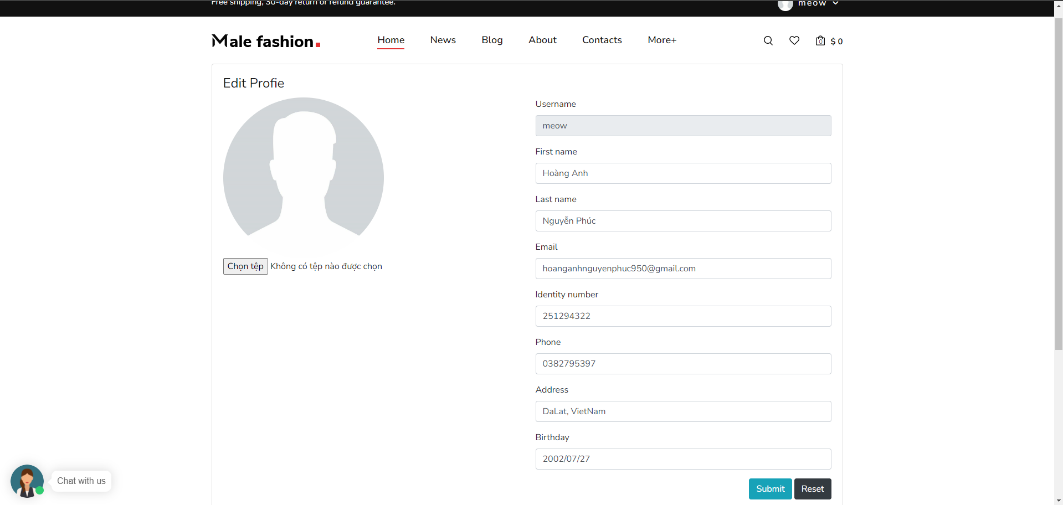
Người dùng có thể đăng nhập vào hệ thống nếu đã có đăng ký tài khoản từ trước.



Nếu chưa có tài khoản, người dùng có thể đăng ký một tài khoản mới. Hệ thống sẽ yêu cầu nhập đầy đủ thông tin cần thiết để lưu vào hệ thống.

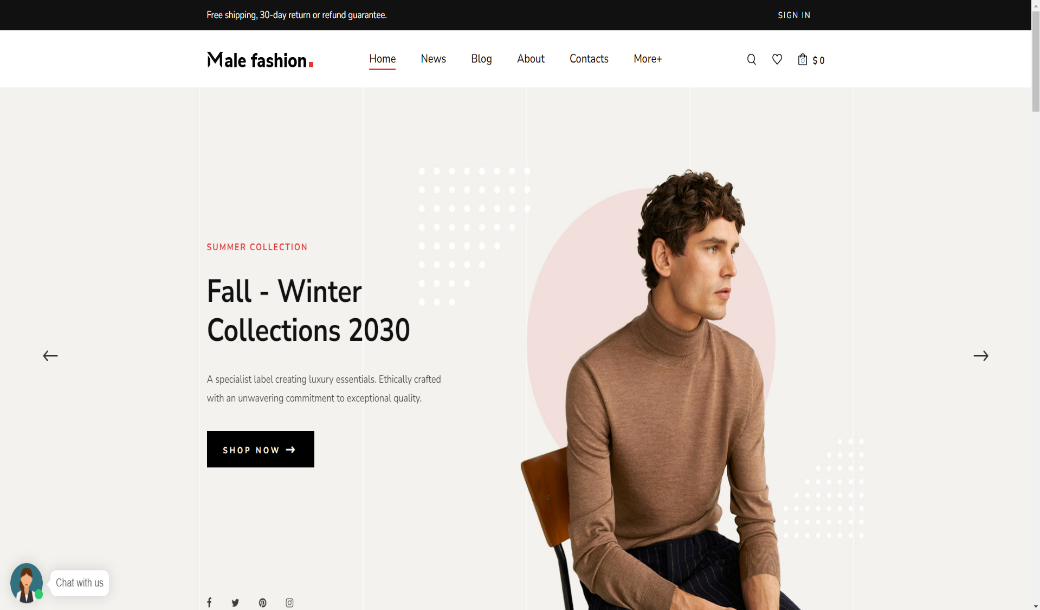


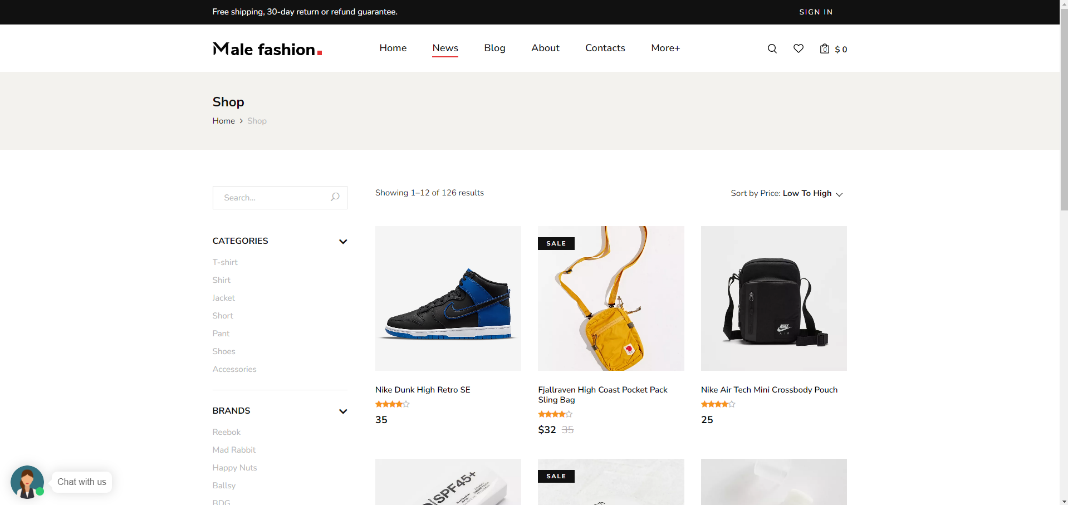
Người dùng có thể thay đổi thông tin cá nhân nếu cần thiết

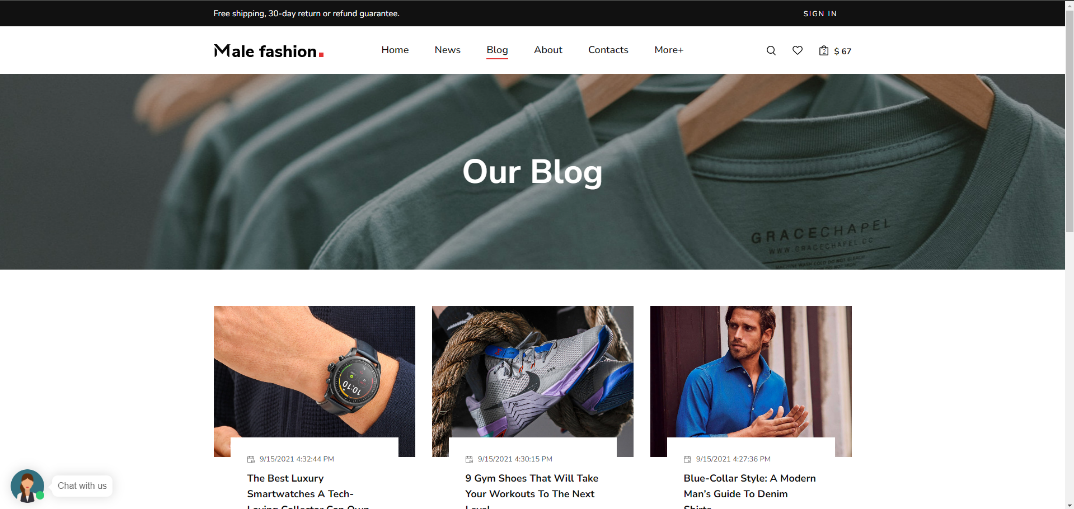


**Quản lý sản phẩm:**

Toàn bộ sản phẩm của shop được hiển thị để người dùng dễ dàng tìm kiếm sản phẩm mà mình muốn.

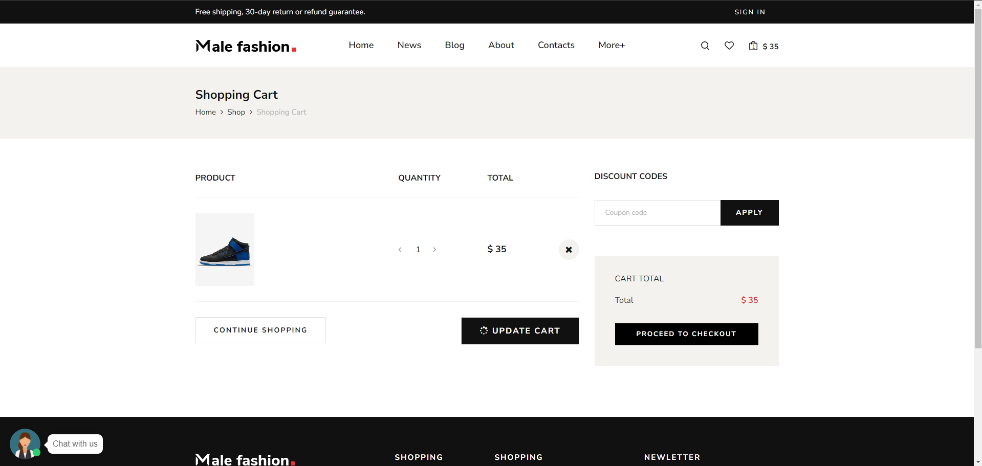






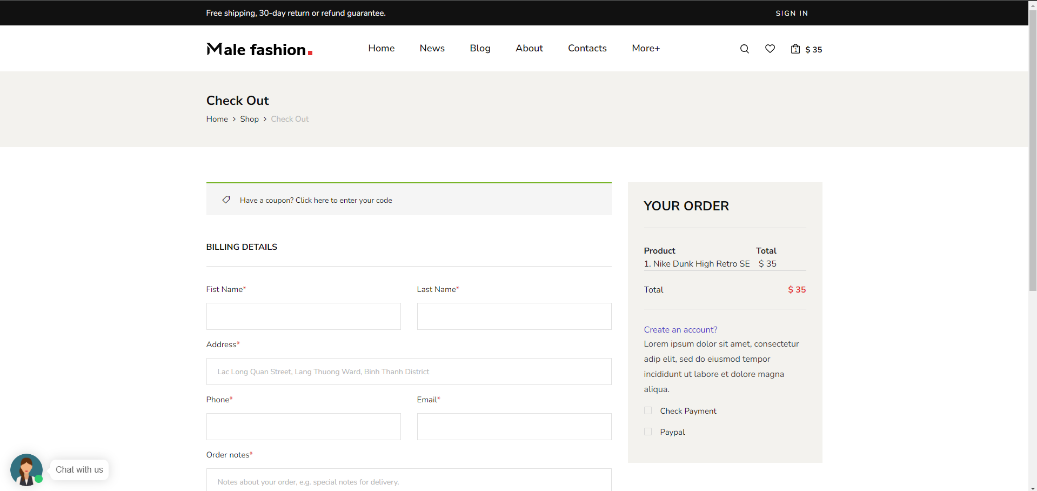
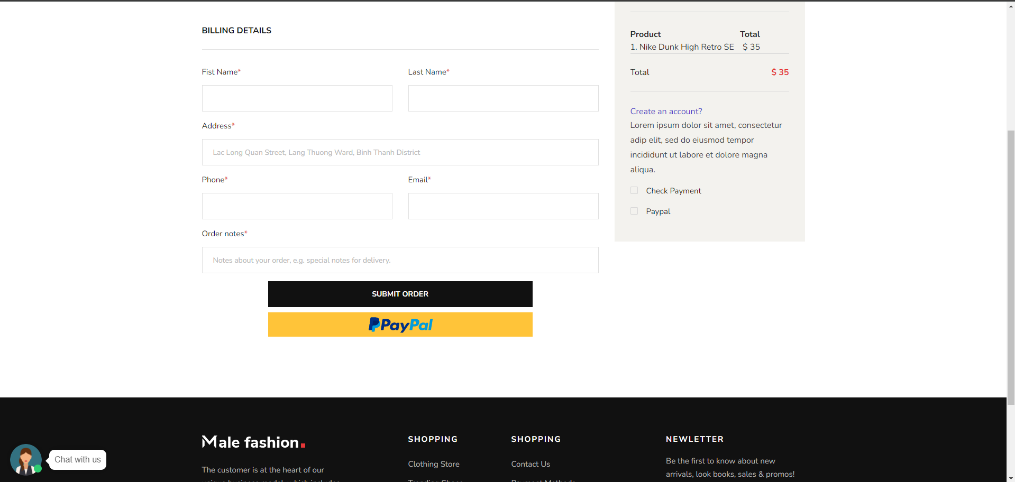
**Quản lý giỏ hàng:**

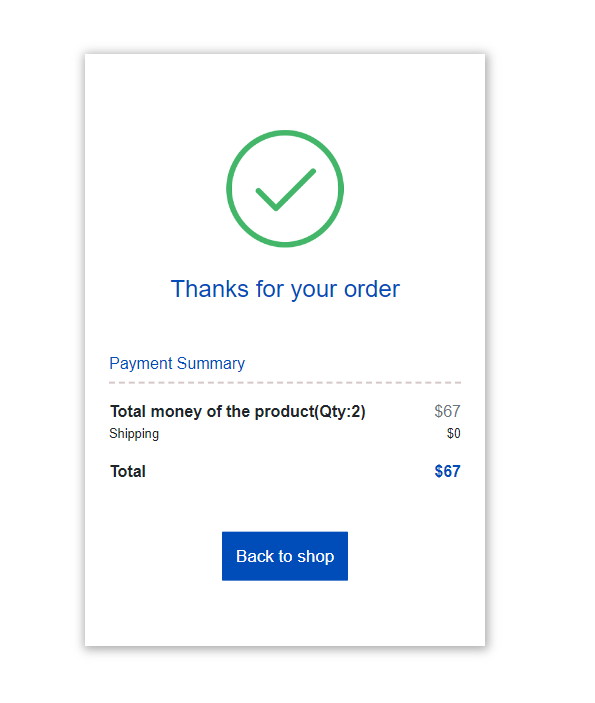
Người dùng có thể xem những sản phầm mà mình đã thêm vào giỏ hàng.

Người dùng có thể thêm, sửa hoặc xóa sản phầm có trong giỏ hàng

**Quản lý thanh toán**:

Người dùng khi muốn thanh toán sẽ được chuyển qua trang thanh toán. Ở đây người dùng cần nhập đầy đủ thông tin để có thể thanh toán và mua hàng ở trong giỏ hàng.



Sau khi thanh toán thành công, một bảng thông báo hiện ra xác nhận rằng việc thanh hoán đã hoàn tất.

### 4.4.2.Triển khai mã nguồn

**Quản lý người dùng:**

Đăng nhập:

public ActionResult Login(FormCollection collection)

{

var tk = collection["username"];

var mk = collection["password"];

mk = Encryptor.MD5Hash(mk);

var check = db.Members.SingleOrDefault(model => model.userName == tk && model.password == mk);

if (ModelState.IsValid)

{

if (check == null)

{

ModelState.AddModelError("", "There was a problem logging in. Check your username and password or create an account.");

}

else

{

if (!this.IsCaptchaValid(""))

{

ViewBag.captcha = "Captcha is not valid";

}

else

{

Session["info"] = check;

return RedirectToAction("Index", "Home");

}

}

}

return View();

}

Trong phương thức Login, dữ liệu đăng nhập được lấy từ FormCollection (collection["username"], collection["password"]) và sau đó được sử dụng để kiểm tra đăng nhập.

Sử dụng Repository Pattern để truy xuất và lưu trữ dữ liệu thông qua đối tượng ‘db’

Nếu tài khoản và mật khẩu trùng khớp thì người dùng sẽ đăng nhập được vào hệ thống

Đăng ký:

public ActionResult Register(Member member)

{

try

{

if (ModelState.IsValid)

{

var check = db.Members.Where(model => model.userName == member.userName).FirstOrDefault();

if (check != null)

{

// check username constained in database

ModelState.AddModelError("", "There was a problem creating your account. Your username already exists.");

return View(member);

}

else

{

member.password = Encryptor.MD5Hash(member.password);

member.dateOfJoin = DateTime.Now;

member.roleId = 3;

member.avatar = "~/Content/img/avatar/avatar.jpg";

member.status = true;

db.Members.Add(member);

var result = db.SaveChanges();

if (result > 0)

{

TempData["msgSuccess"] = "Successfully create account!";

return RedirectToAction("Login");

}

}

}

return View(member);

}

catch(Exception ex)

{

TempData["msgFailed"] = "Failed create account! " +ex.Message;

return RedirectToAction("Login");

}

}

Sử dụng FormColection, dữ liệu khi đăng ký được ràng buộc trực tiếp với đối tượng ‘Member’, cho phép kiểm tra hợp lệ và lưu trữ dữ liệu đơn giản hơn.

Sử dụng Repository Pattern để truy xuất và lưu trữ dữ liệu thông qua đối tượng ‘db’

Người dùng sau khi đăng kí sẽ hiện ra một bảng thông báo xác nhận người dùng đã đăng ký thành công.

Chỉnh sửa thông tin:

public ActionResult EditProfie(Member member, HttpPostedFileBase uploadFile)

{

try

{

if (ModelState.IsValid)

{

if (uploadFile != null)

{

var fileName = Path.GetFileName(uploadFile.FileName);

var path = Path.Combine(Server.MapPath("~/Content/img/avatar"), fileName);

member.avatar = "~/Content/img/avatar/" + fileName;

db.Entry(member).State = EntityState.Modified;

string oldImgPath = Request.MapPath(Session["imgPath"].ToString()); // Lấy đường dẫn ảnh (absolute path)

var avatarName = Session["imgPath"].ToString(); // Lấy đường dẫn ảnh (relative path)

var checkAvatart = db.Members.Where(model => model.avatar == avatarName).ToList(); // Kiểm tra ảnh có trùng với avatar của member nào không

if (db.SaveChanges() > 0)

{

uploadFile.SaveAs(path);

if (System.IO.File.Exists(oldImgPath) && checkAvatart.Count < 2) // Nếu tồn tại hình trong folder và không member nào có hình này thì xóa ra khỏi folder

{

System.IO.File.Delete(oldImgPath);

}

var info = db.Members.Where(model => model.userName == member.userName).SingleOrDefault();// Lấy thông tin mới cập nhập lưu vào session

Session["info"] = info;

return RedirectToAction("Index");

}

}

else

{

member.avatar = Session["imgPath"].ToString();

db.Entry(member).State = EntityState.Modified;

if (db.SaveChanges() > 0)

{

var info = db.Members.Where(model => model.userName == member.userName).SingleOrDefault();// Lấy thông tin mới cập nhập lưu vào session

Session["info"] = info;

return RedirectToAction("Index");

}

}

}

ViewBag.roleId = new SelectList(db.Roles, "roleId", "roleName", member.roleId);

return View(member);

}

catch (Exception ex)

{

TempData["msgEditProfieFailed"] = "Edit failed! " + ex.Message;

return RedirectToAction("Index");

}

}

Trước tiên, kiểm tra xem người dùng đã tải lên một hình ảnh mới hay không thông qua tham số uploadFile. Nếu có hình ảnh mới, thì tiến hành lưu hình ảnh vào thư mục ~/Content/img/avatar trên server.

Sau đó, cập nhật đường dẫn của hình ảnh vào thuộc tính avatar của đối tượng member.

Thiết lập trạng thái của đối tượng member trong context của Entity Framework là Modified để thông báo rằng đối tượng này đã được thay đổi.

Tiếp theo, kiểm tra xem người dùng đã tải lên hình ảnh mới hay không. Nếu có, thì lưu thông tin mới vào cơ sở dữ liệu.

Nếu không có hình ảnh mới, thì cũng cập nhật thông tin mới vào cơ sở dữ liệu.

Sau khi lưu thông tin thành công, cập nhật thông tin mới của người dùng vào Session và chuyển hướng người dùng đến trang chủ.

Sử dụng Repository Pattern để truy xuất và lưu trữ dữ liệu thông qua đối tượng ‘db’

**Quản lý sản phầm**

public class ShopController : Controller

{

menfashionEntities db = new menfashionEntities();

// GET: Shop

public ActionResult ProductList()

{

return View();

}

public PartialViewResult ListItem(string brand,int? categories,int? page, string searching) // Show product

{

var pageNumber = page ?? 1;

var pageSize = 9;

if(searching != null)

{

ViewBag.categories = categories;

var list = db.Products.Where(model => model.productName.Contains(searching) || searching == null && model.status == true).OrderByDescending(model => model.dateCreate).ToPagedList(pageNumber, pageSize);

return PartialView(list);

}

else

{

if (brand != null && categories == null)

{

ViewBag.categories = categories;

var list = db.Products.OrderByDescending(model => model.dateCreate).Where(model => model.brand == brand && model.status == true).ToPagedList(pageNumber, pageSize);

return PartialView(list);

}

else if (brand == null && categories != null)

{

ViewBag.categories = categories;

var list = db.Products.OrderByDescending(model => model.dateCreate).Where(model => model.categoryId == categories && model.status == true).ToPagedList(pageNumber, pageSize);

return PartialView(list);

}

else

{

var list = db.Products.OrderByDescending(model => model.dateCreate).Where(model => model.status == true).ToPagedList(pageNumber, pageSize);

return PartialView(list);

}

}

}

public PartialViewResult Categories() // List categories

{

var list = db.ProductCategories.ToList();

return PartialView(list);

}

public PartialViewResult Brand() // List brand

{

List<String> brand = new List<string>();

foreach (Product i in this.db.Products)

{

if (!brand.Contains(i.brand.Trim()))

brand.Add(i.brand.Trim());

}

return PartialView(brand);

}

public PartialViewResult RelationProduct(int? category) // List brand

{

var categories = db.Products.Where(model => model.categoryId == category).Take(4).ToList();

return PartialView(categories);

}

public ActionResult ProductDetail(int? id, int? category)

{

if (id == null)

{

return RedirectToAction("Error", "Home");

}

var detail = db.Products.Where(model => model.productId == id).Single();

ViewBag.NewPrice = detail.price - ((detail.price \* detail.discount) / 100);

if (detail == null)

{

return RedirectToAction("Error", "Home");

}

return View(detail);

}

}

Giải thích đoạn code:

ProductList: Phương thức này trả về một view để hiển thị danh sách sản phẩm.

ListItem: Phương thức này được sử dụng để lấy danh sách sản phẩm dựa trên các tham số như thương hiệu (brand), danh mục (categories), trang (page), và từ khóa tìm kiếm (searching). Nó trả về một PartialView chứa danh sách sản phẩm.

Categories: Phương thức này trả về một PartialView chứa danh sách các danh mục sản phẩm.

Brand: Phương thức này trả về một PartialView chứa danh sách các thương hiệu sản phẩm.

RelationProduct: Phương thức này trả về một PartialView chứa danh sách sản phẩm liên quan dựa trên một danh mục cụ thể.

ProductDetail: Phương thức này hiển thị chi tiết của một sản phẩm dựa trên id của sản phẩm và category tương ứng. Nó trả về một view hiển thị chi tiết sản phẩm.

Session Management: Sử dụng Session để lưu thông tin người dùng sau khi họ đăng nhập, giúp duy trì trạng thái đăng nhập của người dùng qua nhiều trang và yêu cầu.

**Quản lý giỏ hàng**

Thêm sản phẩm:

public ActionResult AddToCart(int id, string strURL) // Add item in cart

{

List<Cart> listCart = getCart();

Cart item = listCart.Find(model => model.IdItem == id);

if (item == null)

{

item = new Cart(id);

listCart.Add(item);

return Redirect(strURL);

}

else

{

item.Quantity++;

return Redirect(strURL);

}

}

Hiển thị thông tin giỏ hàng:

private int Quanlity() // Lấy tổng số sản phẩm giỏ hàng hiện tại

{

int amount = 0;

List<Cart> listCart = Session["Cart"] as List<Cart>;

if (listCart != null)

{

amount = listCart.Sum(model => model.Quantity);

}

return amount;

}

private double TotalPrice() // Lấy tổng số tiền sản phẩm

{

double total = 0;

List<Cart> listCart = Session["Cart"] as List<Cart>;

if (listCart != null)

{

total = listCart.Sum(model => model.PriceTotal);

}

return total;

}

public PartialViewResult Navbar() // Hiển thị số lượng sản phẩm và tiền trên navbar

{

ViewBag.quanlityItem = Quanlity();

ViewBag.totalPrice = TotalPrice();

return PartialView();

}

public ActionResult NoICart()

{

return View();

}

[HttpGet]

public ActionResult Cart()

{

List<Cart> listCart = getCart();

Session["Cart"] = listCart;

if (listCart.Count == 0)

{

return RedirectToAction("NoICart", "Cart");

}

ViewBag.quanlityItem = Quanlity();

ViewBag.totalPrice = TotalPrice();

return View(listCart);

}

public ActionResult DeteteCart(int id)

{

List<Cart> listCart = getCart();

Cart item = listCart.SingleOrDefault(model => model.IdItem == id);

if (item != null)

{

listCart.RemoveAll(model => model.IdItem == id);

return RedirectToAction("Cart", "Cart");

}

if (listCart.Count == 0)

{

return RedirectToAction("Index", "Home");

}

return RedirectToAction("Cart", "Cart");

}

public ActionResult UpdateCart(FormCollection form)

{

string[] qualities = form.GetValues("quanlity");

List<Cart> listCart = getCart();

for (int i = 0; i < listCart.Count; i++)

{

listCart[i].Quantity = Convert.ToInt32(qualities[i]);

}

if (Quanlity() == 0)

{

Session["Cart"] = null;

return RedirectToAction("NoICart", "Cart");

}

else

{

return RedirectToAction("Cart", "Cart");

}

}

Phương thức Quanlity và TotalPrice:

Quanlity: Trả về tổng số lượng sản phẩm trong giỏ hàng. Sử dụng LINQ để tính tổng số lượng các mặt hàng trong giỏ hàng.

TotalPrice: Trả về tổng giá tiền của tất cả các mặt hàng trong giỏ hàng. Tương tự như Quanlity, sử dụng LINQ để tính tổng giá tiền của các mặt hàng trong giỏ hàng.

Phương thức Navbar: Trả về một PartialView chứa thông tin về số lượng sản phẩm và tổng giá tiền hiện có trong giỏ hàng. Sử dụng ViewBag để truyền dữ liệu đến view.

Phương thức NoICart: Trả về một view hiển thị thông báo khi giỏ hàng trống.

Phương thức Cart:

GET: Hiển thị danh sách các mặt hàng trong giỏ hàng. Nếu giỏ hàng trống, chuyển hướng người dùng đến trang hiển thị thông báo.

Truyền dữ liệu về số lượng sản phẩm và tổng giá tiền vào ViewBag để hiển thị trên giao diện người dùng.

Phương thức DeteteCart: Xóa một mặt hàng khỏi giỏ hàng dựa trên id của mặt hàng. Nếu mặt hàng không tồn tại trong giỏ hàng hoặc nếu giỏ hàng trống sau khi xóa, chuyển hướng người dùng đến trang chính.

Phương thức UpdateCart: Cập nhật số lượng của các mặt hàng trong giỏ hàng dựa trên dữ liệu được gửi từ form. Nếu giỏ hàng trống sau khi cập nhật, giỏ hàng sẽ được xóa và chuyển hướng đến trang thông báo giỏ hàng trống.

Session Management: Sử dụng Session để lưu trữ thông tin về giỏ hàng và duy trì trạng thái giỏ hàng qua các yêu cầu.

**Quản lý thanh toán**

//Trang checkout khi không đăng nhập tài khoản

[HttpGet]

public ActionResult CheckoutNoAccount()

{

List<Cart> listCart = getCart();

ViewBag.quanlityItem = Quanlity();

ViewBag.totalPrice = TotalPrice();

return View(listCart);

}

[HttpPost]

public ActionResult CheckoutNoAccount(Customer customer)

{

try

{

if (ModelState.IsValid)

{

var check = db.Customers.Where(model => model.lastName == customer.lastName && model.email == customer.email).FirstOrDefault();

// Kiểm tra xem tên khách hàng có email nhập đã tồn tại hay chưa

if (check == null) // Nếu chưa thì lưu lại thông tin vào bảng customer

{

db.Customers.Add(customer);

db.SaveChanges();

Invoince bill = new Invoince();

List<Cart> listCart = getCart();

bill.customerId = customer.customerId;

bill.invoinceNo = CreateKey("HD");

bill.dateOrder = DateTime.Now;

bill.status = true;

bill.deliveryDate = null;

bill.deliveryStatus = false;

int totalmoney = 0;

foreach (var item in listCart)

{

totalmoney += Convert.ToInt32(item.PriceTotal);

}

bill.totalMoney = totalmoney;

db.Invoinces.Add(bill);

db.SaveChanges();

foreach (var item in listCart)

{

InvoinceDetail ctdh = new InvoinceDetail();

ctdh.invoinceNo = bill.invoinceNo;

ctdh.productId = item.IdItem;

ctdh.quanlityProduct = item.Quantity;

ctdh.unitPrice = item.unitPrice;

ctdh.totalPrice = (int?)(long)item.PriceTotal;

ctdh.totalDiscount = item.Discount \* item.Quantity;

db.InvoinceDetails.Add(ctdh);

}

db.SaveChanges();

return RedirectToAction("SubmitBill", "Cart");

}

else

{

Invoince bill = new Invoince();

List<Cart> listCart = getCart();

bill.customerId = check.customerId;

bill.invoinceNo = CreateKey("HD");

bill.dateOrder = DateTime.Now;

bill.status = true;

bill.deliveryDate = null;

bill.deliveryStatus = false;

int totalmoney = 0;

foreach (var item in listCart)

{

totalmoney += Convert.ToInt32(item.PriceTotal);

}

bill.totalMoney = totalmoney;

db.Invoinces.Add(bill);

db.SaveChanges();

foreach (var item in listCart)

{

InvoinceDetail ctdh = new InvoinceDetail();

ctdh.invoinceNo = bill.invoinceNo;

ctdh.productId = item.IdItem;

ctdh.quanlityProduct = item.Quantity;

ctdh.unitPrice = item.unitPrice;

ctdh.totalPrice = (int?)(long)item.PriceTotal;

ctdh.totalDiscount = item.Discount \* item.Quantity;

db.InvoinceDetails.Add(ctdh);

}

db.SaveChanges();

return RedirectToAction("SubmitBill", "Cart");

}

}

return View();

}

catch(Exception ex)

{

return RedirectToAction("Error", "Home");

}

}

[HttpGet]

public ActionResult SubmitBill()

{

ViewBag.quanlityItem = Quanlity();

ViewBag.totalPrice = TotalPrice();

return View();

}

[HttpPost]

public ActionResult SubmitBill(FormCollection form)

{

Session.Remove("Cart");

//Session["Cart"] = null;

return RedirectToAction("Index", "Home");

}

CheckoutNoAccount (GET):

Phương thức này được sử dụng để hiển thị trang thanh toán khi người dùng không đăng nhập.

Nó lấy danh sách các mặt hàng trong giỏ hàng, số lượng sản phẩm và tổng giá trị của giỏ hàng và truyền chúng vào ViewBag để hiển thị trên giao diện.

Sau đó, nó trả về một View có chứa thông tin giỏ hàng để người dùng xem và nhập thông tin khách hàng.

CheckoutNoAccount (POST):

Phương thức này được sử dụng để xử lý việc người dùng gửi thông tin thanh toán từ trang CheckoutNoAccount (GET).

Đầu tiên, nó kiểm tra tính hợp lệ của dữ liệu đầu vào bằng cách sử dụng ModelState.IsValid.

Sau đó, nó kiểm tra xem khách hàng có tồn tại trong cơ sở dữ liệu không dựa trên họ và email.

Nếu không tồn tại, nó lưu thông tin khách hàng mới vào cơ sở dữ liệu và tạo hóa đơn mới.

Nếu tồn tại, nó sử dụng thông tin khách hàng đã tồn tại để tạo hóa đơn mới.

Sau khi tạo hóa đơn thành công, nó lưu chi tiết hóa đơn vào cơ sở dữ liệu và chuyển hướng người dùng đến trang SubmitBill để thông báo hoàn tất đơn hàng.

SubmitBill (GET):

Phương thức này được sử dụng để hiển thị trang xác nhận hoàn tất đơn hàng sau khi người dùng đã thanh toán thành công.

Nó lấy số lượng sản phẩm và tổng giá trị của giỏ hàng và truyền chúng vào ViewBag để hiển thị trên giao diện.

SubmitBill (POST):

Phương thức này được sử dụng để xử lý việc người dùng xác nhận và hoàn tất đơn hàng.

Nó xóa giỏ hàng khỏi Session và chuyển hướng người dùng về trang chủ sau khi hoàn tất đơn hàng.

1. **Kết quả và đánh giá**

## 5.1. Kết quả đạt được

Sau khi triển khai và thử nghiệm trên nền tảng Visual Studio với ASP.NET Web Application (.NET Framework), hệ thống website shop bán quần áo đã hoạt động ổn định và đáp ứng đầy đủ các yêu cầu chức năng và phi chức năng đã đề ra. Cụ thể:

* Đăng ký/Đăng nhập người dùng: Hệ thống cho phép người dùng đăng ký và đăng nhập một cách dễ dàng và an toàn. Tính năng khôi phục mật khẩu qua email cũng hoạt động hiệu quả, giúp người dùng khôi phục tài khoản một cách nhanh chóng.
* Xem danh sách sản phẩm: Người dùng có thể duyệt qua danh sách sản phẩm một cách mượt mà, với thông tin sản phẩm hiển thị rõ ràng và hấp dẫn. Các sản phẩm được phân loại chính xác.
* Tìm kiếm sản phẩm: Chức năng tìm kiếm hoạt động chính xác và cung cấp kết quả nhanh chóng. Tính năng gợi ý từ khóa giúp người dùng dễ dàng tìm thấy sản phẩm họ cần.
* Thêm sản phẩm vào giỏ hàng: Quá trình thêm sản phẩm vào giỏ hàng rất đơn giản và trực quan. Người dùng có thể dễ dàng xem và chỉnh sửa giỏ hàng của mình trước khi tiến hành thanh toán.
* Thanh toán: Quy trình thanh toán được thực hiện an toàn và thuận tiện, hỗ trợ nhiều phương thức thanh toán khác nhau. Sau khi thanh toán thành công, người dùng nhận được xác nhận qua email và có thể theo dõi trạng thái đơn hàng trong tài khoản của mình.

## 5.2. Đánh giá

**Ưu điểm:**

* Việc áp dụng mẫu thiết kế Singleton giúp giảm thiểu tài nguyên hệ thống và đảm bảo tính nhất quán của dữ liệu, đặc biệt là trong quản lý kết nối cơ sở dữ liệu và cấu hình hệ thống.
* Mẫu thiết kế Factory giúp hệ thống dễ dàng mở rộng khi cần thêm loại sản phẩm mới mà không cần thay đổi nhiều trong mã nguồn hiện tại, giúp giảm thiểu rủi ro và công sức bảo trì.
* Công nghệ .NET Framework và Visual Studio cung cấp môi trường phát triển mạnh mẽ, hỗ trợ tốt cho quá trình phát triển và gỡ lỗi.
* Hệ thống bảo mật được triển khai tốt, đảm bảo an toàn thông tin người dùng và giao dịch.

**Nhược điểm:**

* Việc sử dụng mẫu thiết kế Singleton có thể gây ra khó khăn trong việc kiểm thử đơn vị (unit testing) do các phụ thuộc cố định (fixed dependencies).
* Việc triển khai và cấu hình ASP.NET có thể phức tạp và đòi hỏi kiến thức chuyên sâu, điều này có thể là thách thức đối với các nhà phát triển mới làm quen với công nghệ này.

1. **Kết luận**

Trong quá trình thực hiện dự án xây dựng website shop bán quần áo trực tuyến bằng Visual Studio với ASP.NET Web Application (.NET Framework), việc áp dụng các mẫu thiết kế Singleton và Factory đã chứng minh được hiệu quả rõ rệt trong việc quản lý các tài nguyên dùng chung và tạo ra các đối tượng linh hoạt. Các mẫu thiết kế này không chỉ giúp tăng tính nhất quán và hiệu quả của hệ thống mà còn tạo điều kiện thuận lợi cho việc mở rộng và bảo trì trong tương lai.

Hệ thống đã hoàn thành các chức năng cốt lõi như đăng ký/đăng nhập người dùng, quản lý sản phẩm, giỏ hàng và thanh toán, đồng thời đáp ứng tốt các yêu cầu phi chức năng về hiệu suất, bảo mật và tính nhất quán.

Việc sử dụng các công nghệ hiện đại như .NET Framework, Visual Studio và SQL Server đã giúp tăng cường hiệu quả phát triển và khả năng mở rộng của hệ thống. Những trải nghiệm và kiến thức thu được từ dự án này sẽ là nền tảng vững chắc cho các dự án phát triển phần mềm trong tương lai.