cl0thed

vintage store

Dokumentacja techniczna projektu informatycznego

Nazwa projektu: cl0thed

Autor projektu: Olivier Łąg

Kontakt: pzx109486@student.poznan.merito.pl

Poznań, 28.12.2024

Spis treści

1. Główne funkcjonalności strony

Projekt „*cl0thed*” to nowoczesny sklep internetowy stworzony z myślą o miłośnikach odzieży second hand, który łączy funkcjonalność z minimalistycznym, ale efektownym designem. Wykorzystując najnowsze technologie, takie jak .NET 8, aplikacja zapewnia użytkownikom płynne i intuicyjne doświadczenie zakupowe. Została zaprojektowana z myślą o prostocie, elegancji oraz maksymalnej wygodzie użytkowania, co czyni ją idealnym miejscem dla osób poszukujących unikalnych, wysokiej jakości produktów w atrakcyjnych cenach.

Strona „cl0thed” wyróżnia się ciemnym, ale nowoczesnym stylem graficznym, który zapewnia nie tylko estetykę, ale także komfort w użytkowaniu, szczególnie podczas długich sesji zakupowych. Każdy element został starannie zaprojektowany, by stworzyć przyjemną i bezproblemową nawigację. Kluczowym celem było zapewnienie użytkownikom łatwego dostępu do produktów, dzięki czemu mogą oni w prosty sposób znaleźć interesującą ich odzież.

Sklep oferuje pełną interaktywność, pozwalając użytkownikom na łatwe przeglądanie oferty oraz korzystanie z efektów wizualnych, które zwiększają atrakcyjność strony. Intuicyjny układ strony i szybki dostęp do najważniejszych funkcji sprawiają, że zakupy na „cl0thed” to czysta przyjemność. Wszystko to składa się na projekt, który nie tylko spełnia potrzeby użytkowników, ale również zaskakuje nowoczesnym podejściem do e-commerce, przy zachowaniu najwyższych standardów estetycznych i funkcjonalnych.

- *Logowanie i konta użytkowników*

Aby uzyskać pełny dostęp do funkcji sklepu, każdy użytkownik musi zalogować się lub zarejestrować. Po zalogowaniu, użytkownik ma dostęp wyłącznie do swoich danych i produktów, co zapewnia pełną prywatność i bezpieczeństwo informacji.

- *Zakupy*

Po zalogowaniu użytkownik może dodać wybrane produkty do koszyka, a następnie przejść do finalizacji zakupu. Proces zakupu został zaprojektowany z myślą o prostocie i wygodzie, umożliwiając użytkownikowi szybkie sfinalizowanie transakcji. Skupiono się na minimalizacji zbędnych kroków, by zakupy były jak najbardziej efektywne i komfortowe. Projekt z wiadomych przyczyn omija krok z wykonywaniem płatności.

- *Powiadomienia e-mail*

Po każdym zrealizowanym zakupie, użytkownik otrzymuje wiadomość e-mail z potwierdzeniem transakcji. W mailu znajdują się szczegóły zamówienia, w tym lista zakupionych produktów. Dzięki temu użytkownicy mają pełną kontrolę nad swoimi zamówieniami.

2. Opis techniczny projektu

Projekt „cl0thed” został zbudowany w technologii ASP.NET Core 8, co zapewnia wysoką wydajność oraz elastyczność aplikacji. Język programowania C# jest użyty do implementacji logiki backendowej, a dla warstwy frontendowej zastosowano HTML, CSS i JavaScript, co pozwala na stworzenie interaktywnego i responsywnego interfejsu użytkownika. Dzięki tym technologiom, aplikacja działa płynnie i jest łatwa do dalszego rozwijania.

Framework: ASP.NET Core 8

Frontend: HTML, CSS, JavaScript

Backend: C# .NET 8

- *Architektura aplikacji*

Projekt oparty jest na wzorcu MVC (Model-View-Controller), co pozwala na czystą separację odpowiedzialności w aplikacji. Wzorzec ten jest szeroko stosowany w aplikacjach webowych, ponieważ umożliwia łatwą organizację kodu i jego rozbudowę w przyszłości.

Model: Odpowiada za dane aplikacji. Przykłady modeli to: użytkownicy, produkty, zamówienia. Każdy model reprezentuje strukturę danych i zawiera logikę biznesową związaną z jego przetwarzaniem.

Widok: To warstwa odpowiedzialna za interfejs użytkownika. Widoki są tworzone za pomocą HTML i CSS, co umożliwia wyświetlanie danych w sposób przystępny dla użytkownika.

Kontroler: Zajmuje się logiką biznesową aplikacji. Odpowiada za komunikację pomiędzy modelem a widokiem, wykonując odpowiednie akcje w odpowiedzi na żądania użytkownika.

- *Główne podstrony aplikacji*

Strona główna: Na stronie głównej wyświetlane są wybrane produkty, które przyciągają uwagę użytkownika. Strona cechuje się minimalistycznym układem z logo na środku oraz efektami wizualnymi, które wprowadzają dynamikę. Po wejściu w dany produkt można go dodać swobodnie do koszyka.

Koszyk: Podstrona, która umożliwia użytkownikowi przegląd produktów wybranych do zakupu. Koszyk automatycznie sumuje całkowitą kwotę zakupu, umożliwiając edycję pozycji (np. usunięcie produktu).

Panel użytkownika: Panel, w którym użytkownik ma dostęp do zmiany emaila, hasła, dodania numeru telefonu, historii zamówień oraz podejrzenia szczegółów danego zamówienia.

- *Bezpieczeństwo*

Bezpieczeństwo danych użytkowników jest kluczowe w projekcie „cl0thed”. Wszystkie dane osobowe, w tym hasła, są odpowiednio haszowane, co zapobiega ich ujawnieniu w przypadku naruszenia bezpieczeństwa. System logowania i rejestracji oparty jest na mechanizmie **ASP.NET Identity**, który zapewnia łatwą integrację z bazą danych i odpowiednią kontrolę dostępu do aplikacji.

- *Interaktywność*

Aplikacja oferuje szereg dynamicznych interakcji, które poprawiają doświadczenie użytkownika. Projekt cechuje się również minimalistycznym designem, który pozwala użytkownikowi skupić się na produkcie, eliminując zbędne elementy i tworząc przyjemne doświadczenie zakupowe.

3. Przykładowy proces zakupowy

Użytkownik najpierw zakłada konto, podając wymagane dane (adres e-mail, hasło). Logowanie jest wymagane, aby uzyskać dostęp do pełnej funkcjonalności strony. Użytkownik na etapie rejestracji jest poproszony o potwierdzenie maila. Bez potwierdzenia maila użytkownik nie zostanie zarejestrowany.

- *Przegląda produkty na stronie głównej lub w danych kategoriach.*

Po zalogowaniu użytkownik może przeglądać produkty dostępne na stronie głównej. Produkty na stronie głównej są poglądowe, a szczegóły można zobaczyć po kliknięciu na dany produkt.

- *Dodaje wybrane produkty do koszyka.*

Użytkownicy mogą dodawać produkty do koszyka. Koszyk automatycznie sumuje wartość zakupów i umożliwia edycję, np. usunięcie produktu.

- *Przechodzi do koszyka, podaje dane do wysyłki i finalizuje zakup.*

Po dokonaniu wyboru produktów użytkownik przechodzi do koszyka, wpisuje dane do wysyłki (imię, nazwisko, adres e-mail, adres wysyłki) i finalizuje zakup. Z wiadomych względów aplikacja omija krok samej płatności.

- *System automatycznie wysyła e-mail z potwierdzeniem zamówienia.*

Po zakończeniu transakcji, system automatycznie wysyła użytkownikowi e-mail z potwierdzeniem zakupu. W wiadomości znajdują się szczegóły zamówienia: lista zamówionych produktów oraz dodatkowe informacje dotyczące transakcji.

4. Opis struktury projektu

Struktura projektu „cl0thed” została zaprojektowana w sposób przejrzysty i modularny, zgodnie z wzorcem MVC (Model-View-Controller). Taki podział logiczny ułatwia rozwój, testowanie oraz konserwację aplikacji.

- *Controllers*

Kontrolery są odpowiedzialne za logikę aplikacji i zarządzanie żądaniami użytkownika. W projekcie wykorzystano następujące kontrolery:

HomeController - obsługuje stronę główną aplikacji oraz inne ogólne widoki.

ProductsController - zarządza wyświetlaniem produktów oraz szczegółami każdego produktu.

CartsController - odpowiada za operacje związane z koszykiem użytkownika, takie jak dodawanie, usuwanie produktów oraz realizacja zamówienia.

- *Models*

Modele odzwierciedlają dane i logikę biznesową aplikacji, a także odpowiadają za komunikację z bazą danych. Każdy model w projekcie reprezentuje tabelę w bazie danych.

Cart - reprezentuje koszyk użytkownika, przechowując powiązane elementy.

CartItem - reprezentuje pojedynczy element w koszyku, zawiera informacje o produkcie oraz ilości.

Order - przechowuje informacje o zamówieniu, takie jak numer zamówienia, dane użytkownika oraz lista zamówionych produktów.

OrderItem - reprezentuje pojedynczy produkt w zamówieniu.

Product - reprezentuje pojedynczy produkt w sklepie, przechowuje informacje takie jak nazwa, cena, opis i adres URL obrazu.

- Views

Widoki są odpowiedzialne za prezentację danych użytkownikowi i odpowiadają za warstwę wizualną aplikacji. Każdy widok jest związany z odpowiednim kontrolerem.

Home/Index - Widok pełni funkcję strony głównej aplikacji i ma minimalistyczny, estetyczny design z głównym naciskiem na wizualizację produktów.

Products/Details - Ten widok wyświetla szczegóły produktu, w tym jego nazwę, opis, cenę oraz zdjęcie. Umożliwia również dodanie produktu do koszyka oraz powrót do listy produktów.

Products/Index: Widok ten prezentuje listę wszystkich produktów w formie kart z miniaturkami, nazwą, opisem i ceną, z możliwością edycji lub usunięcia produktu przez administratora.

Products/Create - Widok ten umożliwia użytkownikowi tworzenie nowego produktu, wprowadzając nazwę, opis, cenę oraz obrazek. Po zapisaniu formularza nowy produkt jest dodawany do bazy danych.

Products/Edit: Widok ten pozwala na edycję istniejącego produktu, umożliwiając zmianę jego nazwy, opisu, ceny oraz obrazu. Po zapisaniu formularza zmiany są zapisywane w bazie danych.

Products/Delete - Wyświetla formularz, który pyta użytkownika, czy na pewno chce usunąć dany produkt.

Carts/OrderConfirmation wyświetla szczegóły zamówienia po jego złożeniu, w tym numer zamówienia, datę oraz podsumowanie produktów z zamówienia. Jeżeli użytkownik nie jest właścicielem zamówienia, otrzyma komunikat o braku dostępu do tych informacji.

Products/ViewCart wyświetla koszyk zakupowy użytkownika, pokazując produkty dodane do koszyka wraz z ich ilościami, cenami oraz łączną ceną zamówienia.

- Dodatkowe elementy

Projekt zawiera również dodatkowe funkcjonalności oraz narzędzia wspierające jego działanie:

Shared/Layout - Widok jest szablonem głównym dla całej aplikacji, zapewniającym wspólny układ i styl dla wszystkich podstron.

Baza danych - lokalna baza danych zarządzana przez Entity Framework, przechowująca informacje o produktach, koszykach i zamówieniach.

Style - projekt graficzny aplikacji oparty na ciemnym motywie z neonowymi akcentami, wykorzystujący CSS lub SCSS.

Email Confirmation - moduł wysyłania e-maili, zapewniający potwierdzenie zamówienia użytkownikowi.

Java Script - obsługuje dwie główne funkcjonalności: przełączanie widoczności nakładki z opcjami oraz dynamiczną zmianę obrazu tła na stronie.

5. Kontrolery

HomeController

Jest jednym z podstawowych kontrolerów w aplikacji ASP.NET MVC. Jego głównym zadaniem jest obsługa żądań związanych ze stroną główną aplikacji, stroną prywatności oraz wyświetlaniem błędów.

Wygląd kontrolera:

```
using System.Diagnostics;
using clothed.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace clothed.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
        }
    }
}
```

Opis kontrolera oraz jego metody:

Zależności:
- ILogger<HomeController>: Interfejs umożliwiający logowanie informacji, błędów oraz ostrzeżeń, co jest przydatne w diagnostyce i monitorowaniu działania aplikacji.

- Microsoft.AspNetCore.Mvc: Udostępnia funkcjonalności MVC, w tym klasy bazowe jak Controller.

Metody:

- Index(): Ta metoda obsługuje żądanie HTTP GET dla strony głównej aplikacji.

ProductsController

Odpowiada za obsługę operacji związanych z zarządzaniem produktami w sklepie internetowym. Umożliwia użytkownikom przeglądanie, tworzenie, edytowanie i usuwanie produktów.

Wygląd kontrolera:

```
using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using clothed.Data;
using clothed.Models;
using Microsoft.AspNetCore.Authorization;

namespace clothed.Controllers
{
    public class ProductsController : Controller
    {
        private readonly ApplicationDbContext _context;

        public ProductsController(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<IActionResult> Index()
        {
            return View(await _context.Products.ToListAsync());
        }

        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var product = await _context.Products.FirstOrDefaultAsync(m => m.Id
== id);
            if (product == null)
            {
                return NotFound();
            }

            return View(product);
        }

        public IActionResult Create()
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
            }

            return View();
        }
```

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("Id,Name,Description,Price")] Product product, IFormFile ImageFile)
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
            }

            if (ModelState.IsValid)
            {
                if (ImageFile != null && ImageFile.Length > 0)
                {
                    var filePath = Path.Combine(Directory.GetCurrentDirectory(),
"wwwroot/images", ImageFile.FileName);
                    using (var stream = new FileStream(filePath,
FileMode.Create))
                    {
                        await ImageFile.CopyToAsync(stream);
                    }

                    product.ImageUrl = $"/images/{ImageFile.FileName}";
                }

                _context.Add(product);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }

            return View(product);
        }

        public async Task<IActionResult> Edit(int? id)
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
            }

            if (id == null)
            {
                return NotFound();
            }

            var product = await _context.Products.FindAsync(id);
            if (product == null)
            {
                return NotFound();
            }

            return View(product);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("Id,Name,Description,Price,ImageUrl")] Product product, IFormFile?
ImageFile)
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
```

```
            }

            if (id != product.Id)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    var existingProduct = await _context.Products.FindAsync(id);
                    if (existingProduct == null)
                    {
                        return NotFound();
                    }

                    existingProduct.Name = product.Name;
                    existingProduct.Description = product.Description;
                    existingProduct.Price = product.Price;

                    if (ImageFile != null && ImageFile.Length > 0)
                    {
                        var fileName = Path.GetFileName(ImageFile.FileName);
                        var filePath = Path.Combine("wwwroot/images", fileName);

                        using (var stream = new FileStream(filePath,
FileMode.Create))
                        {
                            await ImageFile.CopyToAsync(stream);
                        }

                        existingProduct.ImageUrl = "/images/" + fileName;
                    }

                    _context.Update(existingProduct);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!ProductExists(product.Id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }

                return RedirectToAction(nameof(Index));
            }

            return View(product);
        }

        public async Task<IActionResult> Delete(int? id)
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
            }
```

```
            if (id == null)
            {
                return NotFound();
            }

            var product = await _context.Products.FirstOrDefaultAsync(m => m.Id
== id);
            if (product == null)
            {
                return NotFound();
            }

            return View(product);
        }

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
            }

            var product = await _context.Products.FindAsync(id);
            if (product != null)
            {
                if (!string.IsNullOrEmpty(product.ImageUrl))
                {
                    var imagePath = Path.Combine("wwwroot",
product.ImageUrl.TrimStart('/'));
                    if (System.IO.File.Exists(imagePath))
                    {
                        System.IO.File.Delete(imagePath);
                    }
                }

                _context.Products.Remove(product);
                await _context.SaveChangesAsync();
            }

            return RedirectToAction(nameof(Index));
        }

        private bool ProductExists(int id)
        {
            return _context.Products.Any(e => e.Id == id);
        }
    }
}
```

- *Opis kontrolera oraz jego metody*

Metoda Index:

Pobiera wszystkie produkty z bazy danych (asynchronicznie) i przekazuje je do widoku Index.

```
        public async Task<IActionResult> Index()
        {
            return View(await _context.Products.ToListAsync());
        }
```

Metoda Details:

Jeśli id jest prawidłowe, pobiera produkt z bazy danych i wyświetla jego szczegóły. Jeśli produkt nie istnieje, zwraca stronę "NotFound".

```
public async Task<IActionResult> Details(int? id)
 {
     if (id == null)
     {
         return NotFound();
     }

     var product = await _context.Products.FirstOrDefaultAsync(m => m.Id == id);
     if (product == null)
     {
         return NotFound();
     }

     return View(product);
 }
```

Metoda Create (GET)

Sprawdza, czy użytkownik jest administratorem (na podstawie jego adresu e-mail). Jeśli nie, przekierowuje do strony głównej. Jeśli tak, renderuje widok formularza.

```
        public IActionResult Create()
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
            }

            return View();
        }
```

Metoda Create (POST)

Sprawdza, czy użytkownik jest administratorem. Następnie, jeżeli model jest poprawny, zapisuje dane produktu w bazie danych oraz przesłany plik obrazu na dysk. Po zapisaniu przekierowuje do listy produktów.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Name,Description,Price")]
Product product, IFormFile ImageFile)
{
    if (User.Identity.Name != "admin@admin.pl")
    {
        return RedirectToAction(nameof(Index));
    }

    if (ModelState.IsValid)
    {
        if (ImageFile != null && ImageFile.Length > 0)
        {
            var filePath = Path.Combine(Directory.GetCurrentDirectory(),
"wwwroot/images", ImageFile.FileName);
            using (var stream = new FileStream(filePath, FileMode.Create))
            {
                await ImageFile.CopyToAsync(stream);
            }

            product.ImageUrl = $"/images/{ImageFile.FileName}";
        }

        _context.Add(product);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    return View(product);
}
```

Metoda Edit (GET)

Jeśli użytkownik jest administratorem, sprawdza, czy id jest prawidłowe. Jeśli tak, pobiera produkt z bazy danych i renderuje formularz edycji.

```
        public async Task<IActionResult> Edit(int? id)
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
            }

            if (id == null)
            {
                return NotFound();
            }

            var product = await _context.Products.FindAsync(id);
            if (product == null)
            {
                return NotFound();
            }

            return View(product);
```

Metoda Edit (POST)

Sprawdza, czy użytkownik jest administratorem i czy id jest poprawne. Następnie aktualizuje produkt w bazie danych. Jeśli przesłano nowy obraz, zapisuje go w odpowiednim katalogu.

```
[HttpPost]
 [ValidateAntiForgeryToken]
 public async Task<IActionResult> Edit(int id,
[Bind("Id,Name,Description,Price,ImageUrl")] Product product, IFormFile?
ImageFile)
 {
     if (User.Identity.Name != "admin@admin.pl")
     {
         return RedirectToAction(nameof(Index));
     }

     if (id != product.Id)
     {
         return NotFound();
     }

     if (ModelState.IsValid)
     {
         try
         {
             var existingProduct = await _context.Products.FindAsync(id);
             if (existingProduct == null)
             {
                 return NotFound();
             }

             existingProduct.Name = product.Name;
             existingProduct.Description = product.Description;
             existingProduct.Price = product.Price;

             if (ImageFile != null && ImageFile.Length > 0)
             {
                 var fileName = Path.GetFileName(ImageFile.FileName);
                 var filePath = Path.Combine("wwwroot/images", fileName);

                 using (var stream = new FileStream(filePath, FileMode.Create))
                 {
                     await ImageFile.CopyToAsync(stream);
                 }

                 existingProduct.ImageUrl = "/images/" + fileName;
             }

             _context.Update(existingProduct);
             await _context.SaveChangesAsync();
         }
         catch (DbUpdateConcurrencyException)
         {
             if (!ProductExists(product.Id))
             {
                 return NotFound();
             }
             else
             {
                 throw;
             }
         }
```

```
            return RedirectToAction(nameof(Index));
        }

        return View(product);
    }
```

Metoda Delete (GET)

Jeśli użytkownik jest administratorem, sprawdza, czy id jest poprawne i jeśli tak, renderuje formularz usunięcia produktu.

```
        public async Task<IActionResult> Delete(int? id)
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
            }

            if (id == null)
            {
                return NotFound();
            }

            var product = await _context.Products.FirstOrDefaultAsync(m => m.Id
== id);
            if (product == null)
            {
                return NotFound();
            }

            return View(product);
        }
```

Metoda Delete (POST)

Sprawdza, czy użytkownik jest administratorem. Jeśli tak, usuwa produkt z bazy danych oraz jego powiązany obraz, jeśli istnieje.

```
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (User.Identity.Name != "admin@admin.pl")
            {
                return RedirectToAction(nameof(Index));
            }

            var product = await _context.Products.FindAsync(id);
            if (product != null)
            {
                if (!string.IsNullOrEmpty(product.ImageUrl))
                {
                    var imagePath = Path.Combine("wwwroot",
product.ImageUrl.TrimStart('/'));
                    if (System.IO.File.Exists(imagePath))
                    {
                        System.IO.File.Delete(imagePath);
                    }
                }

                _context.Products.Remove(product);
                await _context.SaveChangesAsync();
            }

            return RedirectToAction(nameof(Index));
        }
```

Metoda pomocnicza ProductExists

Wykonuje zapytanie do bazy danych, aby sprawdzić, czy produkt o podanym id istnieje.

```
        private bool ProductExists(int id)
        {
            return _context.Products.Any(e => e.Id == id);
        }
```

CartsController

Sprawdza, czy użytkownik jest zalogowany. Jeśli nie, przekierowuje go do strony logowania. Jeśli użytkownik jest zalogowany, pobiera produkt z bazy danych. Sprawdza, czy koszyk użytkownika już istnieje w bazie. Jeśli nie, tworzy nowy koszyk. Jeśli produkt już znajduje się w koszyku, zwiększa jego ilość, w przeciwnym razie dodaje go jako nowy element. Zapisuje zmiany w bazie danych.

Wygląd kontrolera:

```
using System;
using Microsoft.AspNetCore.Mvc;
using clothed.Data;
using clothed.Models;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using System.ComponentModel.DataAnnotations;
using System.Net.Mail;
using System.Net;
using System.Text;
using Microsoft.AspNetCore.Authorization;

namespace clothed.Controllers
{
    public class CartsController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly UserManager<IdentityUser> _userManager;
        private readonly IEmailSender _emailSender;

        public CartsController(ApplicationDbContext context,
UserManager<IdentityUser> userManager, IEmailSender emailSender)
        {
            _context = context;
            _userManager = userManager;
            _emailSender = emailSender;
        }

        [HttpPost]
        public async Task<IActionResult> AddToCart(int productId)
        {
            if (!User.Identity.IsAuthenticated)
            {
                return
Redirect($"/Identity/Account/Login?ReturnUrl={Url.Action("Index", "Home")}");
            }

            var userEmail = User.Identity.Name;
            var product = await _context.Products.FindAsync(productId);
            if (product == null)
            {
                return NotFound();
            }

            var cart = await _context.Carts
                .FirstOrDefaultAsync(c => c.UserEmail == userEmail);

            if (cart == null)
            {
```

```
                cart = new Cart
                {
                    UserEmail = userEmail,
                    CreatedAt = DateTime.Now,
                    Items = new List<CartItem>()
                };
                _context.Carts.Add(cart);
                await _context.SaveChangesAsync();
            }

            var existingItem = cart.Items.FirstOrDefault(i => i.ProductId ==
productId);
            if (existingItem != null)
            {
                existingItem.Quantity++;
            }
            else
            {
                cart.Items.Add(new CartItem
                {
                    ProductId = productId,
                    Quantity = 1
                });
            }

            await _context.SaveChangesAsync();
            return RedirectToAction("Index", "Products");
        }

        public async Task<IActionResult> ViewCart()
        {
            var userEmail = User.Identity.Name;
            var cart = await _context.Carts
                .Where(c => c.UserEmail == userEmail)
                .Include(c => c.Items)
                .ThenInclude(i => i.Product)
                .FirstOrDefaultAsync();

            if (cart == null)
            {
                return View(new Cart());
            }

            return View(cart);
        }

        public async Task<IActionResult> RemoveFromCart(int cartItemId)
        {
            var cartItem = await _context.CartItems.FindAsync(cartItemId);
            if (cartItem != null)
            {
                _context.CartItems.Remove(cartItem);
                await _context.SaveChangesAsync();
            }

            return RedirectToAction("ViewCart");
        }

        [HttpPost]
        [Authorize]
        public async Task<IActionResult> Checkout()
        {
            var userEmail = User.Identity.Name;
```

```
            var cart = await _context.Carts
                .Where(c => c.UserEmail == userEmail)
                .Include(c => c.Items)
                .ThenInclude(i => i.Product)
                .FirstOrDefaultAsync();

            if (cart == null || !cart.Items.Any())
            {
                return RedirectToAction("ViewCart");
            }

            var random = new Random();
            var orderNumber = random.Next(1000000000, 2000000000).ToString();

            var order = new Order
            {
                OrderNumber = orderNumber,
                UserEmail = userEmail,
                CreatedAt = DateTime.Now,
                Items = cart.Items.Select(i => new OrderItem
                {
                    ProductId = i.ProductId,
                    Quantity = i.Quantity
                }).ToList()
            };

            _context.Orders.Add(order);

            _context.Carts.Remove(cart);

            await _context.SaveChangesAsync();

            var emailSubject = "CLOTHED | Your Order Confirmation";
            var orderDetails = string.Join("<br>", cart.Items.Select(item =>
                $"{item.Product.Name} (Quantity: {item.Quantity})"));

            var emailBody = $@"
        <html>
        <body style='font-family: Arial, sans-serif; background-color: #000000;
color: #fff; padding: 20px;'>
            <div style='max-width: 600px; margin: 0 auto; padding: 20px;
background-color: rgba(0, 0, 0, 0.8); border-radius: 8px; text-align: center;'>
                <h2 style='color: #00ffff;'>Thank you for your order!</h2>
                <p>Your order number is <strong>{orderNumber}</strong>.</p>
                <p>Below are the items you ordered:</p>
                <div style='text-align: left;'>
                    <p>{orderDetails}</p>
                </div>
                <p>Your items will be shipped soon. Thank you for shopping with
CLOTHED.</p>
                <p style='font-size: 14px; color: #bbb;'>Best regards,<br>The
CLOTHED Team</p>
            </div>
        </body>
        </html>";

            SendEmail(userEmail, emailSubject, emailBody);

            return RedirectToAction("OrderConfirmation", new { orderNumber });
        }

        private bool SendEmail(string email, string subject, string confirmLink)
```

```
        {
            try
            {
                MailMessage message = new MailMessage();
                SmtpClient smtpClient = new SmtpClient();
                message.From = new MailAddress("noreply@clothed.com");
                message.To.Add(email);
                message.Subject = subject;
                message.IsBodyHtml = true;
                message.Body = confirmLink;

                smtpClient.Port = 587;
                smtpClient.Host = "smtp.gmail.com";

                smtpClient.EnableSsl = true;
                smtpClient.UseDefaultCredentials = false;
                smtpClient.Credentials = new
NetworkCredential("clothed.666@gmail.com", "bjwcaamwvinfifog");
                smtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;
                smtpClient.Send(message);
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }
        [Authorize]
        public async Task<IActionResult> OrderConfirmation(string orderNumber)
        {
            var order = await _context.Orders
                .Where(o => o.OrderNumber == orderNumber)
                .Include(o => o.Items)
                .ThenInclude(i => i.Product)
                .FirstOrDefaultAsync();

            if (order == null)
            {
                return NotFound();
            }

            return View(order);
        }
        [Authorize]
        public async Task<IActionResult> OrderDetails(int orderId)
        {
            var order = await _context.Orders
                .Where(o => o.Id == orderId)
                .Include(o => o.Items)
                .ThenInclude(i => i.Product)
                .FirstOrDefaultAsync();

            if (order == null)
            {
                return NotFound();
            }

            return View("~/Areas/Identity/Pages/Account/OrderDetails.cshtml",
order);
        }
    }
}
```

- *Opis kontrolera oraz jego metody*

Metoda AddToCart

Sprawdza, czy użytkownik jest zalogowany. Jeśli tak, sprawdza, czy koszyk istnieje; jeśli nie, tworzy nowy. Dodaje produkt do koszyka lub zwiększa jego ilość.

```
[HttpPost]
 public async Task<IActionResult> AddToCart(int productId)
 {
     if (!User.Identity.IsAuthenticated)
     {
         return Redirect($"/Identity/Account/Login?ReturnUrl={Url.Action("Index",
"Home")}");
     }

     var userEmail = User.Identity.Name;
     var product = await _context.Products.FindAsync(productId);
     if (product == null)
     {
         return NotFound();
     }

     var cart = await _context.Carts
         .FirstOrDefaultAsync(c => c.UserEmail == userEmail);

     if (cart == null)
     {
         cart = new Cart
         {
             UserEmail = userEmail,
             CreatedAt = DateTime.Now,
             Items = new List<CartItem>()
         };
         _context.Carts.Add(cart);
         await _context.SaveChangesAsync();
     }

     var existingItem = cart.Items.FirstOrDefault(i => i.ProductId == productId);
     if (existingItem != null)
     {
         existingItem.Quantity++;
     }
     else
     {
         cart.Items.Add(new CartItem
         {
             ProductId = productId,
             Quantity = 1
         });
     }

     await _context.SaveChangesAsync();
     return RedirectToAction("Index", "Products");
 }
```

Metoda ViewCart

Pobiera koszyk użytkownika na podstawie adresu e-mail i wczytuje produkty w koszyku.

```
public async Task<IActionResult> ViewCart()
{
    var userEmail = User.Identity.Name;
    var cart = await _context.Carts
        .Where(c => c.UserEmail == userEmail)
        .Include(c => c.Items)
        .ThenInclude(i => i.Product)
        .FirstOrDefaultAsync();

    if (cart == null)
    {
        return View(new Cart());
    }

    return View(cart);
}
```

Metoda RemoveFromCart

Pobiera i usuwa element koszyka na podstawie identyfikatora, a następnie zapisuje zmiany.

```
public async Task<IActionResult> RemoveFromCart(int cartItemId)
{
    var cartItem = await _context.CartItems.FindAsync(cartItemId);
    if (cartItem != null)
    {
        _context.CartItems.Remove(cartItem);
        await _context.SaveChangesAsync();
    }

    return RedirectToAction("ViewCart");
}
```

Metoda Checkout

Sprawdza, czy koszyk użytkownika nie jest pusty, generuje numer zamówienia, zapisuje zamówienie i wysyła potwierdzenie e-mailem.

```
[HttpPost]
[Authorize]
public async Task<IActionResult> Checkout()
{
    var userEmail = User.Identity.Name;

    var cart = await _context.Carts
        .Where(c => c.UserEmail == userEmail)
        .Include(c => c.Items)
        .ThenInclude(i => i.Product)
        .FirstOrDefaultAsync();

    if (cart == null || !cart.Items.Any())
    {
```

```
            return RedirectToAction("ViewCart");
        }

        var random = new Random();
        var orderNumber = random.Next(1000000000, 2000000000).ToString();

        var order = new Order
        {
            OrderNumber = orderNumber,
            UserEmail = userEmail,
            CreatedAt = DateTime.Now,
            Items = cart.Items.Select(i => new OrderItem
            {
                ProductId = i.ProductId,
                Quantity = i.Quantity
            }).ToList()
        };

        _context.Orders.Add(order);

        _context.Carts.Remove(cart);

        await _context.SaveChangesAsync();

        var emailSubject = "CLOTHED | Your Order Confirmation";
        var orderDetails = string.Join("<br>", cart.Items.Select(item =>
            $"{item.Product.Name} (Quantity: {item.Quantity})"));

        var emailBody = $@"
    <html>
    <body style='font-family: Arial, sans-serif; background-color: #000000;
color: #fff; padding: 20px;'>
        <div style='max-width: 600px; margin: 0 auto; padding: 20px;
background-color: rgba(0, 0, 0, 0.8); border-radius: 8px; text-align: center;'>
            <h2 style='color: #00ffff;'>Thank you for your order!</h2>
            <p>Your order number is <strong>{orderNumber}</strong>.</p>
            <p>Below are the items you ordered:</p>
            <div style='text-align: left;'>
                <p>{orderDetails}</p>
            </div>
            <p>Your items will be shipped soon. Thank you for shopping with
CLOTHED.</p>
            <p style='font-size: 14px; color: #bbb;'>Best regards,<br>The
CLOTHED Team</p>
        </div>
    </body>
    </html>";

        SendEmail(userEmail, emailSubject, emailBody);

        return RedirectToAction("OrderConfirmation", new { orderNumber });
    }
```

Metoda SendEmail

Tworzy wiadomość e-mail, ustawia parametry SMTP i wysyła wiadomość na podany adres e-mail.

```
  private bool SendEmail(string email, string subject, string confirmLink)
  {
      try
      {
          MailMessage message = new MailMessage();
          SmtpClient smtpClient = new SmtpClient();
          message.From = new MailAddress("noreply@clothed.com");
          message.To.Add(email);
          message.Subject = subject;
          message.IsBodyHtml = true;
          message.Body = confirmLink;

          smtpClient.Port = 587;
          smtpClient.Host = "smtp.gmail.com";

          smtpClient.EnableSsl = true;
          smtpClient.UseDefaultCredentials = false;
          smtpClient.Credentials = new NetworkCredential("clothed.666@gmail.com",
"bjwcaamwvinfifog");
          smtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;
          smtpClient.Send(message);
          return true;
      }
      catch (Exception)
      {
          return false;
      }
  }
```

Metoda OrderConfirmation

Pobiera zamówienie na podstawie numeru zamówienia i wyświetla je w widoku.

```
        [Authorize]
        public async Task<IActionResult> OrderConfirmation(string orderNumber)
        {
            var order = await _context.Orders
                .Where(o => o.OrderNumber == orderNumber)
                .Include(o => o.Items)
                .ThenInclude(i => i.Product)
                .FirstOrDefaultAsync();

            if (order == null)
            {
                return NotFound();
            }

            return View(order);
        }
```

Metoda OrderDetails

Pobiera zamówienie na podstawie identyfikatora zamówienia i wyświetla je w odpowiednim widoku.

```
        [Authorize]
        public async Task<IActionResult> OrderDetails(int orderId)
        {
            var order = await _context.Orders
                .Where(o => o.Id == orderId)
                .Include(o => o.Items)
                .ThenInclude(i => i.Product)
                .FirstOrDefaultAsync();

            if (order == null)
            {
                return NotFound();
            }

            return View("~/Areas/Identity/Pages/Account/OrderDetails.cshtml",
order);
        }
```

6. Modele

Model Cart

Model reprezentujący koszyk użytkownika. Zawiera informacje o użytkowniku, dacie utworzenia koszyka oraz kolekcji przedmiotów (produkty) znajdujących się w koszyku. Przechowuje dane użytkownika, adres e-mail, datę utworzenia koszyka oraz listę przedmiotów w koszyku, które są powiązane z koszykiem.

```
namespace clothed.Models
{
    public class Cart
    {
        public int Id { get; set; }
        public string UserEmail { get; set; }
        public DateTime CreatedAt { get; set; }

        public ICollection<CartItem> Items { get; set; } = new List<CartItem>();
    }
}
```

Model CartItem

Model reprezentujący przedmiot w koszyku. Zawiera dane dotyczące produktu, jego ilości oraz powiązanie z koszykiem. Przechowuje identyfikator przedmiotu, identyfikator produktu, powiązanie z produktem (dzięki obiektowi Product), ilość produktu w koszyku oraz powiązanie z koszykiem (CartId i Cart).

```
namespace clothed.Models
{
    public class CartItem
    {
        public int Id { get; set; }
        public int ProductId { get; set; }
        public Product Product { get; set; }
        public int Quantity { get; set; }
        public int CartId { get; set; }
        public Cart Cart { get; set; }
    }
}
```

Model Order

Model reprezentujący zamówienie. Zawiera dane związane z zamówieniem, takie jak numer zamówienia, e-mail użytkownika, data utworzenia oraz lista przedmiotów w zamówieniu. Przechowuje identyfikator zamówienia, numer zamówienia, e-mail użytkownika składającego zamówienie, datę utworzenia zamówienia oraz listę przedmiotów zamówienia (powiązaną z modelem OrderItem).

```
namespace clothed.Models
{
    public class Order
    {
        public int Id { get; set; }
        public string OrderNumber { get; set; }
        public string UserEmail { get; set; }
        public DateTime CreatedAt { get; set; }
        public ICollection<OrderItem> Items { get; set; } = new
List<OrderItem>();

    }
}
```

Model OrderItem

Model reprezentujący pojedynczy przedmiot w zamówieniu. Zawiera dane związane z produktem zamówionym przez użytkownika, takie jak identyfikator produktu, jego ilość, oraz powiązanie z zamówieniem. Model przechowuje identyfikatory zamówionego przedmiotu i produktu, ilość tego produktu w zamówieniu oraz powiązanie z zamówieniem (model Order). Przechowuje również szczegóły dotyczące samego produktu.

```
namespace clothed.Models
{
    public class OrderItem
    {
        public int Id { get; set; }
        public int ProductId { get; set; }
        public Product Product { get; set; }
        public int Quantity { get; set; }
        public int OrderId { get; set; }
        public Order Order { get; set; }
    }
}
```

Model Product

Model reprezentujący produkt w sklepie. Zawiera podstawowe informacje o produkcie, takie jak nazwa, opis, cena oraz opcjonalny adres URL do obrazu produktu. Model przechowuje szczegóły dotyczące produktów oferowanych w sklepie. Jest używany do przechowywania danych, które będą wyświetlane na stronie produktu w sklepie internetowym, w tym informacji o nazwie produktu, jego opisie, cenie oraz ścieżce do obrazu, jeśli taki został dodany.

```
namespace clothed.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
        public string? ImageUrl { get; set; }
    }
}
```

7. Widoki

Home / Index

Widok jest odpowiedzialny za wyświetlenie głównego obrazka oraz umożliwienie użytkownikowi przejścia do strony produktów. Widok zawiera także skrypt do zmiany obrazka głównego przy użyciu tablicy obrazów. Główny obrazek jest początkowo ustawiony na "/images/RL1.png", ale może się zmieniać na inne obrazy zawarte w tablicy images. Wykorzystany jest również mechanizm TempData do przechowywania wiadomości związanych z wylogowaniem użytkownika.

```
@{
    var logoutMessage = TempData["LogoutMessage"];
}

<div style="text-align: center; margin-top: 100px;">
    <h1></h1>
</div>

<div class="main-container">
    <div class="main-image-wrapper">
        <a asp-controller="Products" asp-action="Index">
            <img src="/images/RL1.png" alt="Main Product" class="main-image"
id="displayedImage">
        </a>
    </div>
</div>


<script>
    const images = [
        "/images/RL1.png",
        "/images/adidas1.png",
        "/images/bones1.png",
        "/images/hoodie1.png",
```

```
        "/images/pegador1.png",
        "/images/WEEK1.png",
    ];

    let currentImageIndex = 0;

    function updateImage() {
        const imageElement = document.getElementById("displayedImage");
        imageElement.src = images[currentImageIndex];
    }
</script>
```

Shared / Layout

Widok zawiera szablon strony głównej, który jest odpowiedzialny za wyświetlenie nagłówka, głównej treści strony, opcji w menu oraz stopki. Jest to struktura wykorzystywana w aplikacjach internetowych opartych na technologii ASP.NET, która łączy HTML, CSS i JavaScript z dynamicznymi danymi serwera. Widok używa mechanizmów ASP.NET do dynamicznego generowania treści, takich jak użytkownik zalogowany/niezalogowany, oraz wyświetlanie zależnych opcji. Wykorzystuje Bootstrap do stylizacji i responsywności.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - My ASP.NET Application</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/styles.css" />
    @RenderSection("Styles", false)
</head>
<body>

    <header>
        <div class="header-container">
            <a asp-area="" asp-controller="Home" asp-action="Index" class="brand-
container">
                <img src="~/images/logo.png" alt="clothed logo" class="brand-
logo">
            </a>
        </div>
    </header>

    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <div class="options-overlay" id="optionsOverlay">
        <div class="options-container">
            <button class="option">
                <a href="~/Products">SHOP ALL</a>
            </button>

            @if (User.Identity.IsAuthenticated)
            {
```

```
                <button class="option">
                    <a asp-area="Identity" asp-page="/Account/Manage/Index">MY
ACCOUNT</a>
                </button>


                    <form asp-area="Identity" asp-page="/Account/Logout" asp-
route-returnUrl="@Url.Page("/", new { area = "" })" method="post">
                        <button class="option">LOGOUT</button>
                    </form>
                <button class="option">
                    <a href="~/Carts/ViewCart">VIEW CART</a>
                </button>
            }
            else
            {
                <button class="option">
                    <a asp-area="Identity" asp-page="/Account/Login">LOGIN</a>
                </button>
                <button class="option">
                    <a asp-area="Identity" asp-
page="/Account/Register">REGISTER</a>
                </button>
            }

            @if (User.Identity.Name == "admin@admin.pl")
            {
                <button class="option">
                    <a href="~/Products/Create">CREATE NEW ITEM</a>
                </button>
            }
        </div>
    </div>

    <footer>
        <div class="bottom-button" onclick="toggleOptions()">
            <img src="/images/vinyl.gif" alt="Vinyl" />
        </div>
    </footer>




    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js
"></script>
    <script src="~/js/site.js"></script>
</body>
</html>
```

Products / Index

Widok ten służy do wyświetlania listy produktów w formie kart, z możliwością przejścia do szczegółów produktu, edycji i usuwania (dla administratora). Każdy produkt ma obraz, nazwę, opis oraz cenę, a dla użytkowników z uprawnieniami administratora dodatkowe przyciski do edycji i usuwania. Widok wykorzystuje pętlę foreach do iteracji po kolekcji produktów przekazanej z kontrolera. Wyświetla dane produktów dynamicznie w odpowiednich sekcjach HTML.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Product List</title>

</head>
<body style="background-color: #000; color: #fff;">
    <div class="container">
        <div class="row">
            @foreach (var item in Model)
            {
                <div class="col-md-4 mb-4 d-flex justify-content-center">
                    <div class="card w-100">
                        <a href="@Url.Action("Details", "Products", new { id =
item.Id })">
                            <div class="image-container">
                                <img src="@item.ImageUrl" class="card-img-top"
alt="@item.Name">
                            </div>
                        </a>
                        <div class="card-body text-center">
                            <h5 class="card-title">@item.Name</h5>
                            <p class="card-text">@item.Description</p>
                            <p class="card-price">@string.Format("{0:C}",
item.Price)</p>
                            <div class="d-flex justify-content-center">
                                @if (User.Identity.Name == "admin@admin.pl")
                                {
                                    <a href="@Url.Action("Edit", "Products", new
{ id = item.Id })" class="btn btn-outline-warning btn-sm ml-2">Edit</a>
                                    <a href="@Url.Action("Delete", "Products",
new { id = item.Id })" class="btn btn-outline-danger btn-sm ml-2">Delete</a>
                                }
                            </div>
                        </div>
                    </div>
                </div>
            }
        </div>
    </div>

</body>
</html>
```

Products / Details

Widok ten jest przeznaczony do wyświetlania szczegółowych informacji o produkcie, w tym jego nazwy, obrazu, opisu, ceny oraz przycisku umożliwiającego dodanie produktu do koszyka. Dodatkowo widok zawiera link powrotu do listy produktów.

```
@model clothed.Models.Product

@{
    ViewData["Title"] = "Product Details";
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewData["Title"]</title>

</head>
<body>
    <div class="container">
        <div class="product-header">
            <h1>@Model.Name</h1>
            <hr />
        </div>

        <div class="product-details">
            <div class="product-image-container">
                <img src="@Model.ImageUrl" alt="@Model.Name">
            </div>

            <div class="product-info">
                <h2>Description</h2>
                <p class="description">@Model.Description</p>
                <p class="price">@string.Format("{0:C}", Model.Price)</p>

                <form asp-controller="Carts" asp-action="AddToCart"
method="post">
                    @Html.AntiForgeryToken()
                    <input type="hidden" name="productId" value="@Model.Id" />
                    <button type="submit" class="btn-add-to-cart">Add to
Cart</button>
                </form>
            </div>
        </div>

        <div class="back-link">
            <a href="@Url.Action("Index", "Products")">Back to List</a>
        </div>
    </div>
</body>
</html>
```

Products / Edit

Formularz działa w kontekście edycji produktu. Zawiera opcję zmiany nazwy, opisu, ceny oraz obrazu. Zmiany są zapisywane po kliknięciu przycisku „Save” przez metodę POST.

```
@model clothed.Models.Product

@{
    ViewData["Title"] = "Edit";
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewData["Title"]</title>

</head>
<body>

    <div class="container">

        <form asp-action="Edit" method="post" enctype="multipart/form-data">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>

            <h1>Edit Product</h1>

            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>

            <div class="form-group">
                <label asp-for="Description" class="control-label"></label>
                <textarea asp-for="Description" class="form-control"></textarea>
                <span asp-validation-for="Description" class="text-
danger"></span>
            </div>

            <div class="form-group">
                <label asp-for="Price" class="control-label"></label>
                <input asp-for="Price" class="form-control" />
                <span asp-validation-for="Price" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label for="ImageFile" class="control-label">Product
Image</label>

                @if (!string.IsNullOrEmpty(Model.ImageUrl))
                {
                    <div>
                        <p>Current Image:</p>
                        <img src="@Model.ImageUrl" alt="Current Product Image"
style="max-width: 20%; height: auto; border-radius: 6px;" />
                    </div>
                }

                <input type="file" name="ImageFile" class="form-control" />
                <span asp-validation-for="ImageUrl" class="text-danger"></span>
            </div>
```

```
            <div class="form-group">
                <input type="submit" value="Save" />
            </div>
        </form>

        <div class="back-btn-container">
            <a class="back-btn" asp-action="Index">Back to List</a>
        </div>
    </div>

</body>
</html>
```

Products / Create

Formularz działa w kontekście tworzenia nowego produktu. Umożliwia użytkownikowi wprowadzenie danych produktu i zapisanie ich po kliknięciu „Create”. Nowy produkt jest przesyłany do kontrolera za pomocą metody POST.

```
@model clothed.Models.Product

@{
    ViewData["Title"] = "Create";
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewData["Title"]</title>
</head>
<body>

    <div class="container">

        <form asp-action="Create" method="post" enctype="multipart/form-data">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>

            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>

            <div class="form-group">
                <label asp-for="Description" class="control-label"></label>
                <input asp-for="Description" class="form-control" />
                <span asp-validation-for="Description" class="text-
danger"></span>
            </div>

            <div class="form-group">
                <label asp-for="Price" class="control-label"></label>
                <input asp-for="Price" class="form-control" />
                <span asp-validation-for="Price" class="text-danger"></span>
            </div>

            <div class="form-group">
                <label for="ImageFile" class="control-label">Product
Image</label>
                <input type="file" name="ImageFile" class="form-control" />
```

```
                <span asp-validation-for="ImageUrl" class="text-danger"></span>
            </div>

            <div class="form-group">
                <input type="submit" value="Create" />
            </div>
        </form>

        <div class="back-btn-container">
            <a class="back-btn" asp-action="Index">Back to List</a>
        </div>
    </div>

    @section Scripts {
        @{
            await Html.RenderPartialAsync("_ValidationScriptsPartial");
        }
    }

</body>
</html>
```

Products / Delete

Formularz działa na zasadzie potwierdzenia usunięcia wybranego produktu z bazy danych. Po kliknięciu „Delete” produkt zostanie usunięty, jeśli metoda w kontrolerze obsługująca tę akcję przeprowadzi odpowiednią operację.

```
@model clothed.Models.Product

@{
    ViewData["Title"] = "Delete";
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewData["Title"]</title>

</head>
<body>

    <div class="container">
        <h3>Are you sure you want to delete this?</h3>

            <form asp-action="Delete">
                <input type="hidden" asp-for="Id" />
                <div style="text-align: center;">
                    <input type="submit" value="Delete" class="btn-danger" />
                </div>
            </form>

            <div class="back-btn-container">
                <a class="back-btn" asp-action="Index">Back to List</a>
            </div>
    </div>

</body>
</html>
```

Carts / ViewCart

Formularz adresowy wysyła dane do kontrolera, który zajmuje się realizacją zamówienia. W sekcji podsumowania zamówienia użytkownik widzi swoje produkty, może usunąć je z koszyka oraz poznać łączny koszt.

```
@model clothed.Models.Cart

@{
    ViewData["Title"] = "Shopping Cart";
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewData["Title"]</title>

</head>
<body>

    <div class="container">
        <div class="shipping-address">
            <h3>Shipping Address</h3>
            <form method="post" action="@Url.Action("Checkout", "Carts")"
id="shippingForm">
                <input type="text" name="FirstName" placeholder="First Name"
required />
                <input type="text" name="LastName" placeholder="Last Name"
required />
                <input type="text" name="Address" placeholder="Street Address"
required />
                <input type="text" name="AptSuite" placeholder="Apt/Suite" />
                <input type="text" name="City" placeholder="City" required />
                <input type="text" name="ZipCode" placeholder="ZIP Code" required
/>
                <input type="text" name="Country" placeholder="Country" required
/>
                <button type="submit" class="btn-checkout" id="checkoutBtn"
style="display: none;">Proceed to Checkout</button>
            </form>
        </div>

        <div class="order-summary">
            <h3>Order Summary</h3>
            @if (Model?.Items != null && Model.Items.Any())
            {
                foreach (var item in Model.Items)
                {
                    <div class="cart-item">
                        <img src="@item.Product.ImageUrl"
alt="@item.Product.Name" />
                        <div class="cart-item-details">
                            <h4>@item.Product.Name</h4>
                            <p class="price">@string.Format("{0:C}",
item.Product.Price)</p>
                            <p class="quantity">Quantity: @item.Quantity</p>
                        </div>
```

```
                        <form method="post" action="@Url.Action("RemoveFromCart",
"Carts")">
                            <input type="hidden" name="cartItemId"
value="@item.Id" />
                            <button type="submit" class="remove-btn">X</button>
                        </form>
                    </div>
                }
            }
            else
            {
                <p>Your cart is empty.</p>
            }

            <div class="total">
                <p>Total Price:</p>
                <p class="total-price">
                    @string.Format("{0:C}", Model.Items.Sum(i => i.Product.Price
* i.Quantity))
                </p>
            </div>
        </div>
    </div>

    <script>
        document.getElementById('shippingForm').addEventListener('input',
function() {
            var form = this;
            var checkoutBtn = document.getElementById('checkoutBtn');
            var isValid = form.checkValidity();

            if (isValid) {
                checkoutBtn.style.display = 'block';
            } else {
                checkoutBtn.style.display = 'none';
            }
        });
    </script>


</body>
</html>
```

Products / OrderConfirmation

Ten widok służy do wyświetlania szczegółów zamówienia użytkownika, w tym numeru zamówienia, daty jego złożenia oraz szczegółowego podsumowania produktów. Zawiera również mechanizm kontroli dostępu, który zapewnia, że tylko użytkownik, który złożył dane zamówienie, ma dostęp do jego szczegółów.

```
@model clothed.Models.Order

@{
    ViewData["Title"] = "Order Confirmation";
}
@{
    var userEmail = User.Identity.Name;
}

<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewData["Title"]</title>
</head>
<body>
    @if (Model.UserEmail != userEmail)
    {
        <div class="access-denied" style="margin-top: 400px; padding: 100px;">
            <h2>You do not have permission to view this order.</h2>
        </div>
    }
    else
    {
        <div class="confirmation-container">
            <h2 class="confirmation-header">Order Confirmation</h2>
            <div class="confirmation-details">
                <p>Thank you for your order!</p>
                <p><strong>Order Number:</strong> @Model.OrderNumber</p>
                <p><strong>Order Date:</strong>
@Model.CreatedAt.ToString("g")</p>
            </div>

            <div class="order-summary">
                <h3>Order Summary</h3>
                @foreach (var item in Model.Items)
                {
                    <div class="order-item">
                        <span><strong>@item.Product.Name</strong> (@item.Quantity
x @string.Format("{0:C}", item.Product.Price))</span>
                        <span>@string.Format("{0:C}", item.Quantity *
item.Product.Price)</span>
                    </div>
                }
            </div>

            <div class="order-total">
                <p>Total Price: <span>@string.Format("{0:C}", Model.Items.Sum(i
=> i.Product.Price * i.Quantity))</span></p>
            </div>
        </div>
    }
</body>
</html>
```

8. Elementy Identity

W projekcie ASP.NET MVC elementy te odpowiadają za zarządzanie kontami użytkowników, w tym rejestrację, logowanie, odzyskiwanie hasła oraz inne operacje związane z tożsamością użytkownika.

Logowanie

Widok logowania zawiera formularz umożliwiający użytkownikowi wprowadzenie adresu e-mail oraz hasła w celu zalogowania się do aplikacji. Formularz jest stylizowany na ciemny motyw z akcentami neonowymi, a także zawiera linki do odzyskiwania hasła i rejestracji dla nowych użytkowników.

```
@page
@model LoginModel

@{
    ViewData["Title"] = "Log in";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="d-flex justify-content-center align-items-center" style="height:
100vh; margin-top: -100px;">
    <div class="text-center" style="width: 100%; max-width: 400px;">
        <form id="account" method="post" style="background-color: #121212;
padding: 30px; border-radius: 10px;">
            <div asp-validation-summary="ModelOnly" class="text-danger"
role="alert"></div>
            <div class="form-group mb-3">
                <label asp-for="Input.Email" style="color:
#00ffff;">Email</label>
                <input asp-for="Input.Email" class="form-control"
autocomplete="username" placeholder="Email" style="background-color: #333; color:
#fff; border: 1px solid #444;" />
                <span asp-validation-for="Input.Email" class="text-
danger"></span>
            </div>
            <div class="form-group mb-3">
                <label asp-for="Input.Password" style="color:
#00ffff;">Password</label>
                <input asp-for="Input.Password" class="form-control"
type="password" autocomplete="current-password" placeholder="Password"
style="background-color: #333; color: #fff; border: 1px solid #444;" />
                <span asp-validation-for="Input.Password" class="text-
danger"></span>
            </div>
            <button type="submit" class="btn btn-primary w-100"
style="background-color: #555; border: none;">Log in</button>
            <div class="mt-3">
                <a asp-page="./ForgotPassword" style="color: #00ffff; text-
decoration: none;">Forgot your password?</a>
            </div>
            <div class="mt-3">
                <a asp-page="./Register" style="color: #00ffff; text-decoration:
none;">Don't have an account?</a>
            </div>
        </form>
    </div>
</div>
```

Plik Login.cshtml.cs zawiera logikę obsługującą proces logowania użytkownika w aplikacji. Implementuje on metody obsługi GET i POST, umożliwiając logowanie przy użyciu standardowego formularza, weryfikację danych użytkownika, a także obsługę błędów i przekierowań w przypadku nieudanych prób logowania.

```
// Licensed to the .NET Foundation under one or more agreements.
// The .NET Foundation licenses this file to you under the MIT license.
#nullable disable

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;

namespace clothed.Areas.Identity.Pages.Account
{
    public class LoginModel : PageModel
    {
        private readonly SignInManager<IdentityUser> _signInManager;
        private readonly ILogger<LoginModel> _logger;

        public LoginModel(SignInManager<IdentityUser> signInManager,
ILogger<LoginModel> logger)
        {
            _signInManager = signInManager;
            _logger = logger;
        }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        [BindProperty]
        public InputModel Input { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        public IList<AuthenticationScheme> ExternalLogins { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        public string ReturnUrl { get; set; }

        /// <summary>
```

```
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        [TempData]
        public string ErrorMessage { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        public class InputModel
        {
            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [Required]
            [EmailAddress]
            public string Email { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [Required]
            [DataType(DataType.Password)]
            public string Password { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [Display(Name = "Remember me?")]
            public bool RememberMe { get; set; }
        }

        public async Task OnGetAsync(string returnUrl = null)
        {
            if (!string.IsNullOrEmpty(ErrorMessage))
            {
                ModelState.AddModelError(string.Empty, ErrorMessage);
            }

            returnUrl ??= Url.Content("~/");

            // Clear the existing external cookie to ensure a clean login process
            await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

            ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

            ReturnUrl = returnUrl;
        }
```

```
        public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl ??= Url.Content("~/");

            ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

            if (ModelState.IsValid)
            {
                // This doesn't count login failures towards account lockout
                // To enable password failures to trigger account lockout, set
lockoutOnFailure: true
                var result = await
_signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe,
lockoutOnFailure: false);
                if (result.Succeeded)
                {
                    _logger.LogInformation("User logged in.");
                    return LocalRedirect(returnUrl);
                }
                if (result.RequiresTwoFactor)
                {
                    return RedirectToPage("./LoginWith2fa", new { ReturnUrl =
returnUrl, RememberMe = Input.RememberMe });
                }
                if (result.IsLockedOut)
                {
                    _logger.LogWarning("User account locked out.");
                    return RedirectToPage("./Lockout");
                }
                else
                {
                    ModelState.AddModelError(string.Empty, "Invalid login
attempt.");
                    return Page();
                }
            }

            // If we got this far, something failed, redisplay form
            return Page();
        }
    }
}
```

Rejestracja

Strona zawiera formularz rejestracji użytkownika, umożliwiający wprowadzenie adresu e-mail, hasła oraz potwierdzenia hasła. Formularz obsługuje walidację danych wejściowych, a po udanym zarejestrowaniu użytkownik jest przekierowywany na stronę logowania lub do wcześniej określonego adresu URL.

```
@page
@model RegisterModel
@{
    ViewData["Title"] = "Register";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="d-flex justify-content-center align-items-center" style="height:
100vh; margin-top: -100px;">
    <div class="text-center" style="width: 100%; max-width: 400px;">
        <form id="registerForm" asp-route-returnUrl="@Model.ReturnUrl"
method="post" style="background-color: #121212; padding: 30px; border-radius:
10px;">
            <div asp-validation-summary="ModelOnly" class="text-danger"
role="alert"></div>
            <div class="form-group mb-3">
                <label asp-for="Input.Email" style="color:
#00ffff;">Email</label>
                <input asp-for="Input.Email" class="form-control"
autocomplete="username" placeholder="Email" style="background-color: #333; color:
#fff; border: 1px solid #444;" />
                <span asp-validation-for="Input.Email" class="text-
danger"></span>
            </div>
            <div class="form-group mb-3">
                <label asp-for="Input.Password" style="color:
#00ffff;">Password</label>
                <input asp-for="Input.Password" class="form-control"
type="password" autocomplete="new-password" placeholder="Password"
style="background-color: #333; color: #fff; border: 1px solid #444;" />
                <span asp-validation-for="Input.Password" class="text-
danger"></span>
            </div>
            <div class="form-group mb-3">
                <label asp-for="Input.ConfirmPassword" style="color:
#00ffff;">Confirm Password</label>
                <input asp-for="Input.ConfirmPassword" class="form-control"
type="password" autocomplete="new-password" placeholder="Confirm Password"
style="background-color: #333; color: #fff; border: 1px solid #444;" />
                <span asp-validation-for="Input.ConfirmPassword" class="text-
danger"></span>
            </div>
            <button id="registerSubmit" type="submit" class="btn btn-primary w-
100" style="background-color: #555; border: none;">Register</button>
            <div class="mt-3">
                <a asp-page="./Login" style="color: #00ffff; text-decoration:
none;">Already have an account? Log in here</a>
            </div>
        </form>
    </div>
</div>
```

Plik Register.cshtml.cs zawiera logikę obsługi rejestracji użytkownika. Model jest odpowiedzialny za tworzenie nowego użytkownika, weryfikację danych, wysyłanie e-maila potwierdzającego rejestrację oraz logowanie użytkownika, jeśli konto wymaga natychmiastowej aktywacji. Ponadto, jeśli użytkownik nie potwierdzi adresu e-mail, będzie wymagane, aby kliknął link aktywacyjny wysłany na jego e-mail.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Net.Mail;
using System.Net;
using System.Text;
using System.Text.Encodings.Web;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.WebUtilities;
using Microsoft.Extensions.Logging;
using System.Drawing;
using System.Web;
using System.IO;
using System.Net.Mime;

namespace clothed.Areas.Identity.Pages.Account
{
    public class RegisterModel : PageModel
    {
        private readonly SignInManager<IdentityUser> _signInManager;
        private readonly UserManager<IdentityUser> _userManager;
        private readonly IUserStore<IdentityUser> _userStore;
        private readonly IUserEmailStore<IdentityUser> _emailStore;
        private readonly ILogger<RegisterModel> _logger;
        private readonly IEmailSender _emailSender;

        public RegisterModel(
            UserManager<IdentityUser> userManager,
            IUserStore<IdentityUser> userStore,
            SignInManager<IdentityUser> signInManager,
            ILogger<RegisterModel> logger,
            IEmailSender emailSender)
        {
            _userManager = userManager;
            _userStore = userStore;
            _emailStore = GetEmailStore();
            _signInManager = signInManager;
            _logger = logger;
            _emailSender = emailSender;
        }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
```

```
        [BindProperty]
        public InputModel Input { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        public string ReturnUrl { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        public IList<AuthenticationScheme> ExternalLogins { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        public class InputModel
        {
            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [Required]
            [EmailAddress]
            [Display(Name = "Email")]
            public string Email { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [Required]
            [StringLength(100, ErrorMessage = "The {0} must be at least {2} and
at max {1} characters long.", MinimumLength = 6)]
            [DataType(DataType.Password)]
            [Display(Name = "Password")]
            public string Password { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [DataType(DataType.Password)]
            [Display(Name = "Confirm password")]
            [Compare("Password", ErrorMessage = "The password and confirmation
password do not match.")]
            public string ConfirmPassword { get; set; }
        }
```

```
        public async Task OnGetAsync(string returnUrl = null)
        {
            ReturnUrl = returnUrl;
            ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
        }

        public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl ??= Url.Content("~/");
            ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            if (ModelState.IsValid)
            {
                var user = CreateUser();

                await _userStore.SetUserNameAsync(user, Input.Email,
CancellationToken.None);
                await _emailStore.SetEmailAsync(user, Input.Email,
CancellationToken.None);
                var result = await _userManager.CreateAsync(user,
Input.Password);

                if (result.Succeeded)
                {
                    _logger.LogInformation("User created a new account with
password.");

                    var userId = await _userManager.GetUserIdAsync(user);
                    var code = await
_userManager.GenerateEmailConfirmationTokenAsync(user);
                    code =
WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
                    var callbackUrl = Url.Page(
                        "/Account/ConfirmEmail",
                        pageHandler: null,
                        values: new { area = "Identity", userId = userId, code =
code, returnUrl = returnUrl },
                        protocol: Request.Scheme);

                    SendEmail(Input.Email, "CLOTHED | EMAIL VERIFICATION",
                        $@"<html>
        <body style='font-family: Arial, sans-serif; background-color: #000000;
color: #fff; padding: 20px;'>
            <div style='max-width: 600px; margin: 0 auto; padding: 20px;
background-color: rgba(0, 0, 0, 0.8); border-radius: 8px; text-align: center;'>
                <h2 style='color: #00ffff;'>Welcome to CLOTHED!</h2>
                <p style='font-size: 16px; color: #ff69b4;'>Thank you for
registering with us. To complete your registration, please confirm your email
address by clicking the button below:</p>
                <a href='{HtmlEncoder.Default.Encode(callbackUrl)}' style='
                    display: inline-block;
                    background-color: #ff69b4;
                    color: white;
                    text-decoration: none;
                    font-weight: bold;
                    padding: 12px 25px;
                    border-radius: 5px;
                    margin-top: 20px;
                    text-align: center;
                    font-size: 16px;
                    transition: background-color 0.3s ease;
                '>
```

```
                    Confirm Email Address
                </a>
                <p style='font-size: 14px; color: #bbb; margin-top: 20px;'>If you
did not create an account, no further action is required.</p>
                <p style='font-size: 14px; color: #bbb;'>Best regards,<br>The
CLOTHED Team</p>
            </div>
        </body>
    </html>");


                    if (_userManager.Options.SignIn.RequireConfirmedAccount)
                    {
                        return RedirectToPage("RegisterConfirmation", new { email
= Input.Email, returnUrl = returnUrl });
                    }
                    else
                    {
                        await _signInManager.SignInAsync(user, isPersistent:
false);
                        return LocalRedirect(returnUrl);
                    }
                }
                foreach (var error in result.Errors)
                {
                    ModelState.AddModelError(string.Empty, error.Description);
                }
            }

            // If we got this far, something failed, redisplay form
            return Page();
        }

        private bool SendEmail(string email, string subject, string confirmLink)
        {
            try
            {
                MailMessage message = new MailMessage();
                SmtpClient smtpClient = new SmtpClient();
                message.From = new MailAddress("noreply@clothed.com");
                message.To.Add(email);
                message.Subject = subject;
                message.IsBodyHtml = true;
                message.Body = confirmLink;

                smtpClient.Port = 587;
                smtpClient.Host = "smtp.gmail.com";


                smtpClient.EnableSsl = true;
                smtpClient.UseDefaultCredentials = false;
                smtpClient.Credentials = new
NetworkCredential("clothed.666@gmail.com", "bjwcaamwvinfifog");
                smtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;
                smtpClient.Send(message);
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }
```

```
        private IdentityUser CreateUser()
        {
            try
            {
                return Activator.CreateInstance<IdentityUser>();
            }
            catch
            {
                throw new InvalidOperationException($"Can't create an instance of
'{nameof(IdentityUser)}'. " +
                    $"Ensure that '{nameof(IdentityUser)}' is not an abstract
class and has a parameterless constructor, or alternatively " +
                    $"override the register page in
/Areas/Identity/Pages/Account/Register.cshtml");
            }
        }

        private IUserEmailStore<IdentityUser> GetEmailStore()
        {
            if (!_userManager.SupportsUserEmail)
            {
                throw new NotSupportedException("The default UI requires a user
store with email support.");
            }
            return (IUserEmailStore<IdentityUser>)_userStore;
        }
    }
}
```

Forgot Password

Widok ForgotPassword.cshtml umożliwia użytkownikowi zresetowanie hasła, podając adres e-mail. Formularz zawiera następujące elementy:

Email Input: Pole do wprowadzenia adresu e-mail, który zostanie użyty do zresetowania hasła. Submit Button: Przycisk do wysłania żądania resetowania hasła. Link to Login: Link umożliwiający powrót do strony logowania, jeśli użytkownik pamięta swoje hasło.

```
@page
@model ForgotPasswordModel
@{
    ViewData["Title"] = "Forgot your password?";
}

<hr />
<div class="d-flex justify-content-center align-items-center" style="height:
100vh; margin-top: -130px;">
    <div class="text-center" style="width: 100%; max-width: 400px;">
        <form id="forgot-password" method="post" style="background-color:
#121212; padding: 30px; border-radius: 10px;">
            <div asp-validation-summary="ModelOnly" class="text-danger"
role="alert"></div>
            <div class="form-group mb-3">
                <label asp-for="Input.Email" style="color: #00ffff;">Enter your
email address</label>
```

```
                <input asp-for="Input.Email" class="form-control"
autocomplete="username" placeholder="Email" style="background-color: #333; color:
#fff; border: 1px solid #444;" />
                <span asp-validation-for="Input.Email" class="text-
danger"></span>
            </div>
            <button type="submit" class="btn btn-primary w-100"
style="background-color: #555; border: none;">Reset Password</button>
            <div class="mt-3">
                <a asp-page="./Login" style="color: #00ffff; text-decoration:
none;">Remember your password? Log in</a>
            </div>
        </form>
    </div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

Plik ForgotPassword.cshtml.cs odpowiada za zresetowanie hasła użytkownika w przypadku., gdy ten zapomniał swojego aktualnego. InputModel Zawiera pole Email, które jest wymagane do zresetowania hasła.

OnPostAsync: Główna metoda, która:

Sprawdza poprawność formularza. Szuka użytkownika w bazie danych na podstawie podanego adresu e-mail. Jeśli użytkownik istnieje i jego e-mail jest potwierdzony, generuje token do resetowania hasła. Tworzy link do resetowania hasła i wysyła go na podany e-mail. Przekierowuje użytkownika na stronę potwierdzenia, gdy operacja przebiegnie pomyślnie.

SendEmail: Funkcja wysyłająca e-maila do użytkownika z linkiem do resetowania hasła. Używa konta Gmail do wysyłania wiadomości.

```
// Licensed to the .NET Foundation under one or more agreements.
// The .NET Foundation licenses this file to you under the MIT license.
#nullable disable

using System;
using System.ComponentModel.DataAnnotations;
using System.Net.Mail;
using System.Net;
using System.Text;
using System.Text.Encodings.Web;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.WebUtilities;

namespace clothed.Areas.Identity.Pages.Account
{
    public class ForgotPasswordModel : PageModel
    {
        private readonly UserManager<IdentityUser> _userManager;
        private readonly IEmailSender _emailSender;
```

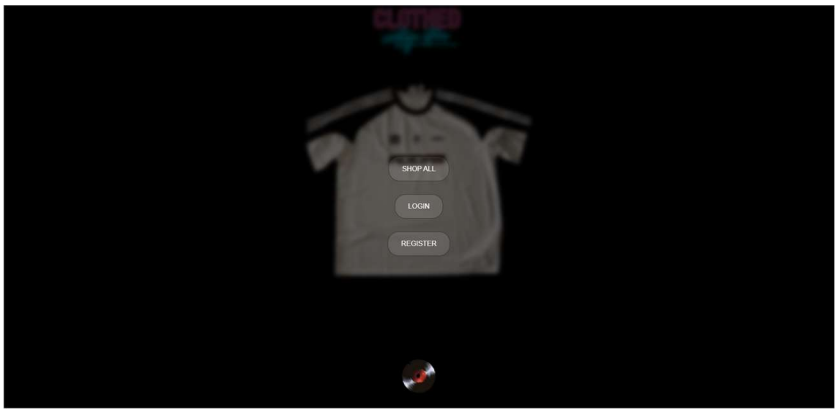
```
        public ForgotPasswordModel(UserManager<IdentityUser> userManager,
IEmailSender emailSender)
        {
            _userManager = userManager;
            _emailSender = emailSender;
        }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        [BindProperty]
        public InputModel Input { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in
future releases.
        /// </summary>
        public class InputModel
        {
            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [Required]
            [EmailAddress]
            public string Email { get; set; }
        }

        public async Task<IActionResult> OnPostAsync()
        {
            if (ModelState.IsValid)
            {
                var user = await _userManager.FindByEmailAsync(Input.Email);
                if (user == null || !(await
_userManager.IsEmailConfirmedAsync(user)))
                {
                    // Don't reveal that the user does not exist or is not
confirmed
                    return RedirectToPage("./ForgotPasswordConfirmation");
                }

                // For more information on how to enable account confirmation and
password reset please
                // visit https://go.microsoft.com/fwlink/?LinkID=532713
                var code = await
_userManager.GeneratePasswordResetTokenAsync(user);
                code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
                var callbackUrl = Url.Page(
                    "/Account/ResetPassword",
                    pageHandler: null,
                    values: new { area = "Identity", code },
                    protocol: Request.Scheme);

                SendEmail(
                    Input.Email, "CLOTHED | RESET YOUR PASSWORD",
                        $@"<html>
```

```
        <body style='font-family: Arial, sans-serif; background-color: #000000;
color: #fff; padding: 20px;'>
            <div style='max-width: 600px; margin: 0 auto; padding: 20px;
background-color: rgba(0, 0, 0, 0.8); border-radius: 8px; text-align: center;'>
                <h2 style='color: #00ffff;'>Please reset your password</h2>
                <a href='{HtmlEncoder.Default.Encode(callbackUrl)}' style='
                    display: inline-block;
                    background-color: #ff69b4;
                    color: white;
                    text-decoration: none;
                    font-weight: bold;
                    padding: 12px 25px;
                    border-radius: 5px;
                    margin-top: 20px;
                    text-align: center;
                    font-size: 16px;
                    transition: background-color 0.3s ease;
                '>
                    Reset your password
                </a>
                <p style='font-size: 14px; color: #bbb;'>Best regards,<br>The
CLOTHED Team</p>
            </div>
        </body>
    </html>");

                return RedirectToPage("./ForgotPasswordConfirmation");
            }

            return Page();
        }
        private bool SendEmail(string email, string subject, string confirmLink)
        {
            try
            {
                MailMessage message = new MailMessage();
                SmtpClient smtpClient = new SmtpClient();
                message.From = new MailAddress("noreply@clothed.com");
                message.To.Add(email);
                message.Subject = subject;
                message.IsBodyHtml = true;
                message.Body = confirmLink;

                smtpClient.Port = 587;
                smtpClient.Host = "smtp.gmail.com";


                smtpClient.EnableSsl = true;
                smtpClient.UseDefaultCredentials = false;
                smtpClient.Credentials = new
NetworkCredential("clothed.666@gmail.com", "bjwcaamwvinfifog");
                smtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;
                smtpClient.Send(message);
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }

    }
}
```

9. Działanie aplikacji krok po kroku

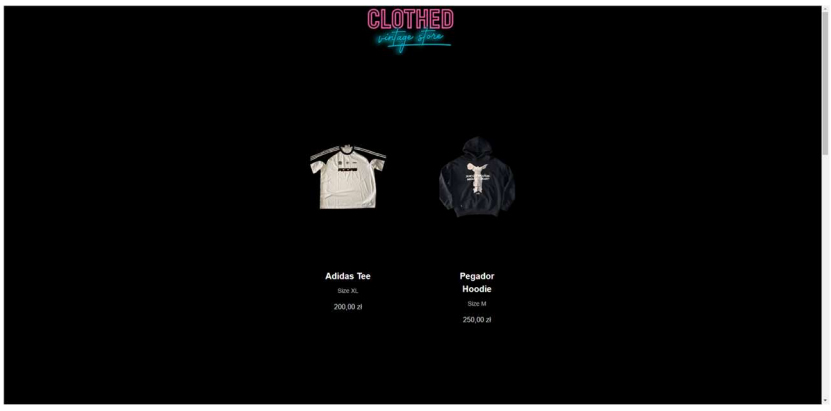
Korzystanie z aplikacji

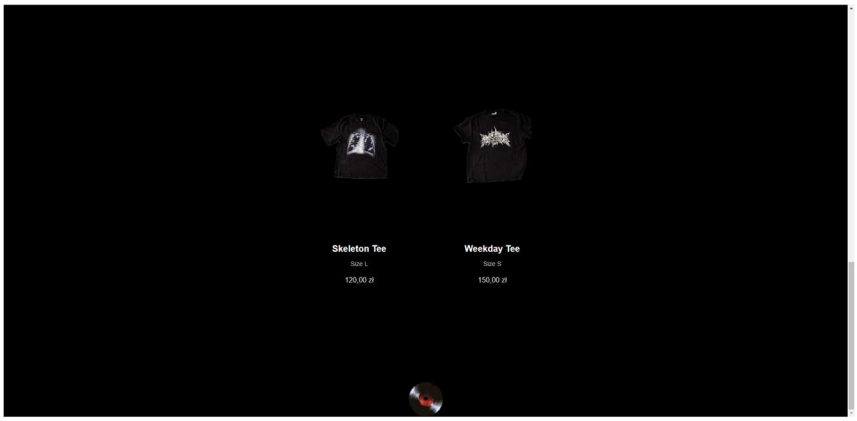
Użytkownik po wejściu na stronę główną aplikacji zostanie przywitany przez logo strony umieszczone w headerze oraz kręcącą się płytę winylową w footerze. Dodatkowo widzimy zdjęcie produktu na środku strony. Za pomocą scrolla możemy zmienić zdjęcie w przód oraz tył.



Jako gość przycisk w stopce pokazuje nam tylko 3 opcje, z których możemy jak najbardziej skorzystać. Przycisk Shop All przenosi nas na strone główną z dostępnymi produktami. Dwa pozostałe przyciski odpowiadają funkcją dostępnymi w nazwie.

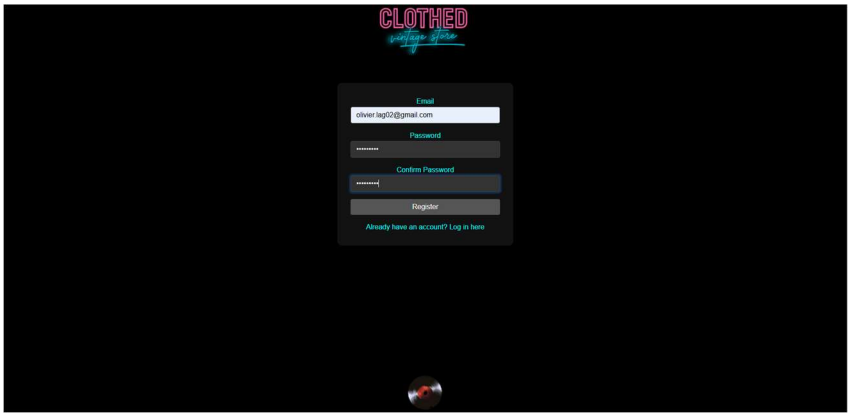
Dodatkowo jako gość możemy nacisnąć na zdjęcie na stronie głównej, co również przeniesie nas na stronę z dostępnymi produktami.





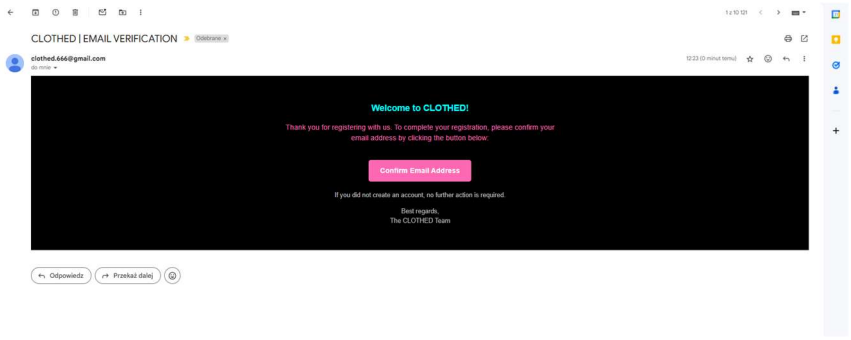
Jako gość możemy również wybrać swój produkt i wejść w jego detale. Cała strona jest również responsywna i interaktywna.

W momencie, w którym niezalogowany użytkownik wciśnie przycisk Add To Cart strona przeniesie nas na strone logowania. Jest to zabezpieczenie, żeby każdy koszyk miał przypisanego użytkownika w bazie danych.



Jako nowy użytkownik nie mamy konta wiec możemy skorzystać z dostępnej opcji rejestracji nowego użytkownika. Wystarczy, że w odpowiednim polu wpiszemy swojego maila (radziłbym prawdziwego), hasło i potwierdzenie hasła. Po wpisaniu tych informacji strona zakomunikuje nam wysłanie maila potwierdzającego konto.

W wiadomości mailowej należy zatwierdzić konto klikając w link. (musi to być z tego samego urządzenia).

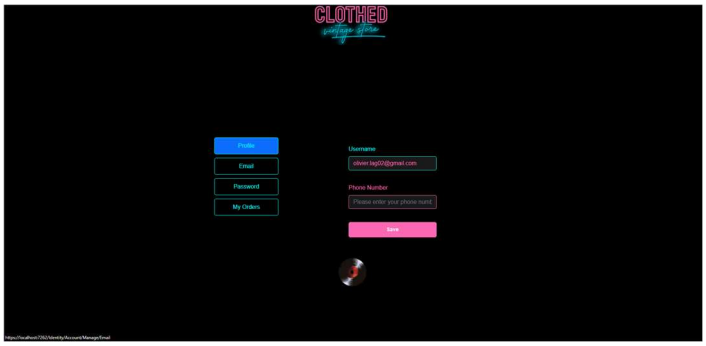


Po zatwierdzeniu swojego konta strona przeniesie nas na potwierdzenie konta. Musimy się zalogować, aby uzyskać pełny dostęp do swojego konta.

W momencie, w którym jesteśmy już zalogowani, po kliknięciu przycisku w stopce pojawią się nam kolejne opcje.

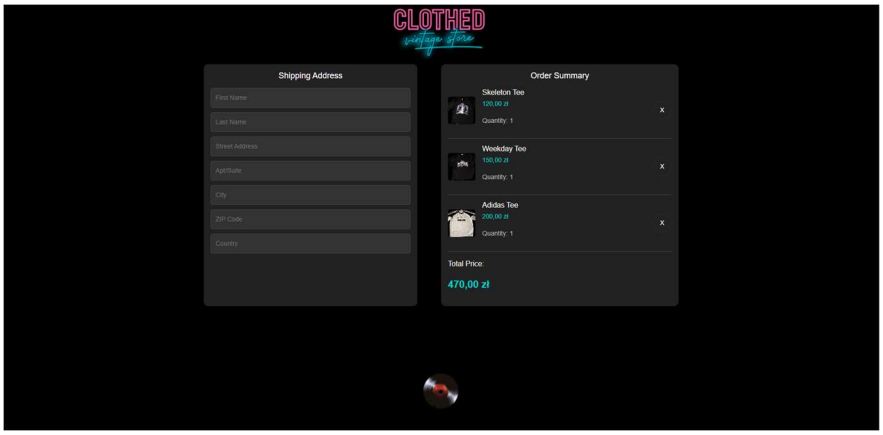
LogOut – co sprawi, że strona przeniesie nas na stronę główną i znów będziemy traktowani jako gość

My Account – co sprawi, że aplikacja przeniesie nas na strone, gdzie możemy zmodyfikować swoje konto lub dodać do niego jakieś informację.

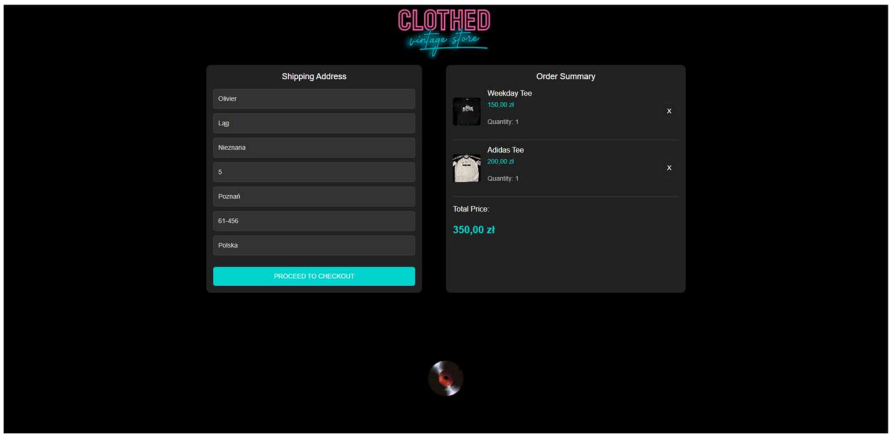


Widzimy, też pole My Orders, niestety po wejściu tam wyświetli się nam komunikat, że nie mamy jeszcze żadnych zamówień.

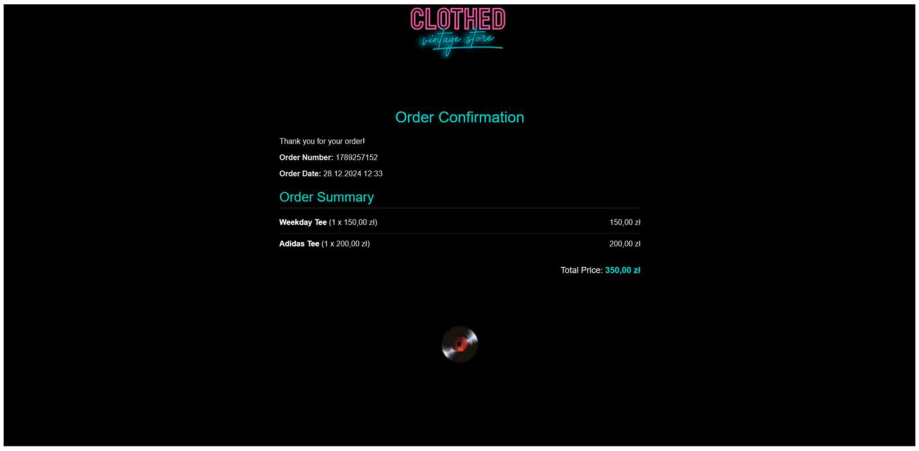
Jest również opcja View Cart – po dodaniu produktu do koszyka strona cofnie nas na stronę główną z produktami. Produkt został dodany do koszyka.



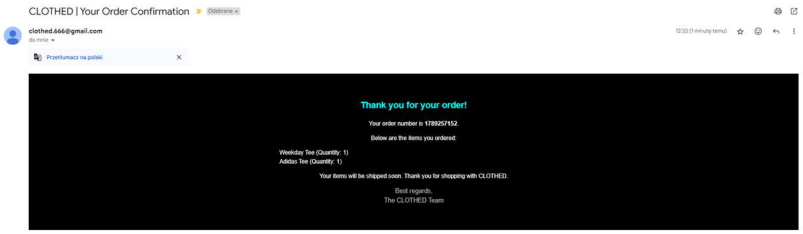
Jak widać wszystkie produkty, które kliknęliśmy zostały dodane do koszyka. Strona również podsumowuje kwotę po dodaniu wszystkich produktów. Przyciskiem „X” możemy usuwać produkty z koszyka. Dodatkowo jak widać aktualnie strona nie ma przycisku świadczącym o płatności. Dopiero w momencie wypisania danych wysyłki strona pozwoli nam przejść do „płatności”



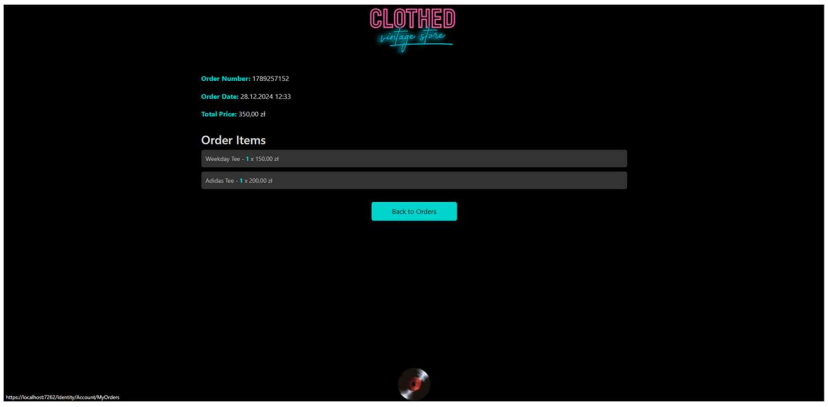
Po kliknięciu odpowiedniego przycisku strona wyświetli nam potwierdzenie zamówienia z losowym numerem zamówienia (aplikacja sama generuje numery).



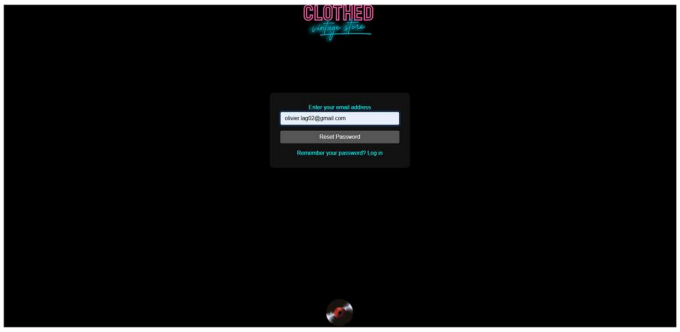
W tym momencie zostało wysłane potwierdzenie zamówienia również na adres mailowy wpisany przy rejestracji użytkownika, gdzie możemy zobaczyć co zamówiliśmy i w jakiej ilości oraz nr. zamówienia.



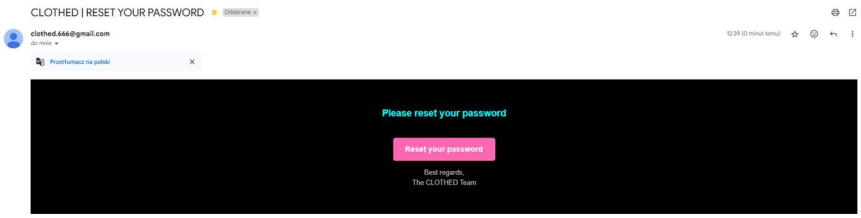
Po złożeniu zamówienia użytkownik może wejść na swoje konto i zobaczyć szczegóły zamówienia.



Dodatkowo również w momencie gdzie użytkownik zapomni swojego hasła ze spokojem może je odzyskać wybierając opcję Forgot Password.

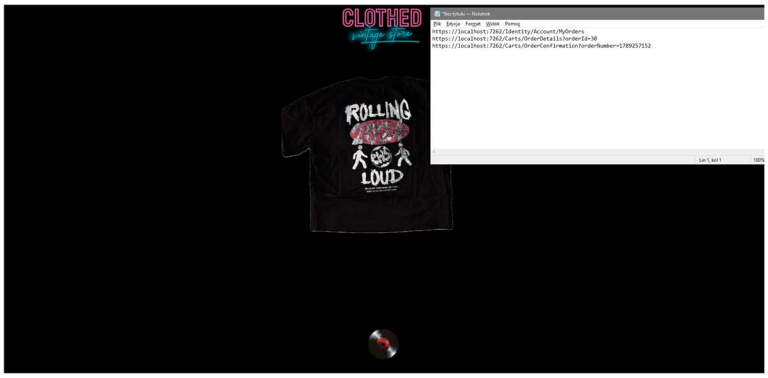


Strona wyślę nam wtedy kolejnego maila, tym razem z linkiem do zmiany hasła.

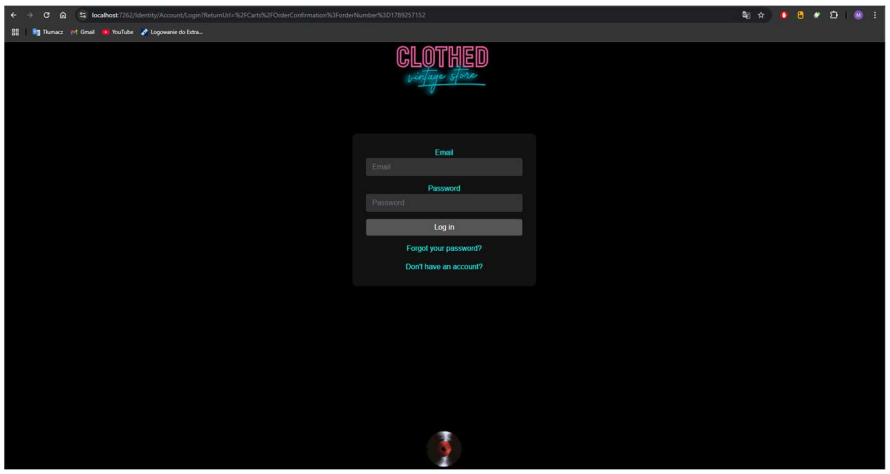


10. Testy

Jako niezalogowany użytkownik sprawdzę teraz, jak strona zachowa się w momencie, gdy posiadam link do informacji, które tylko zalogowany użytkownik powinien widzieć.



Na każdą z tych opcji strona wyświetla nam panel do logowania co powoduje, że nie można podejrzeć innemu użytkownikowi zamówień czy potwierdzeń zamówienia lub też szczegółów.



11. Zakończenie

Podsumowanie i wnioski

Projekt "CLOTHED" został pomyślnie zakończony, a wszystkie kluczowe funkcjonalności zostały wdrożone zgodnie z wymaganiami. Aplikacja umożliwia użytkownikom logowanie, resetowanie hasła, przeglądanie produktów, dodawanie ich do koszyka, finalizowanie zamówień oraz integrację z systemem e-mailowym do wysyłania potwierdzeń. System działa na platformie ASP.NET MVC z użyciem C# oraz bazy danych SQL Server. Dzięki zastosowaniu nowoczesnych technologii udało się osiągnąć założone cele związane z płynnością działania i bezpiecznym przechowywaniem danych.

Wyzwania i ograniczenia

Podczas realizacji projektu napotkano trudności związane z integracją funkcji resetowania hasła z systemem e-mailowym oraz wymaganiami dotyczącymi bezpieczeństwa danych użytkowników. Pomimo tych wyzwań, rozwiązania zostały wdrożone przy użyciu technologii SMTP i odpowiednich zabezpieczeń, co zapewnia bezpieczeństwo przesyłanych informacji.