Sklep internetowy z częściami komputerowymi PCParts

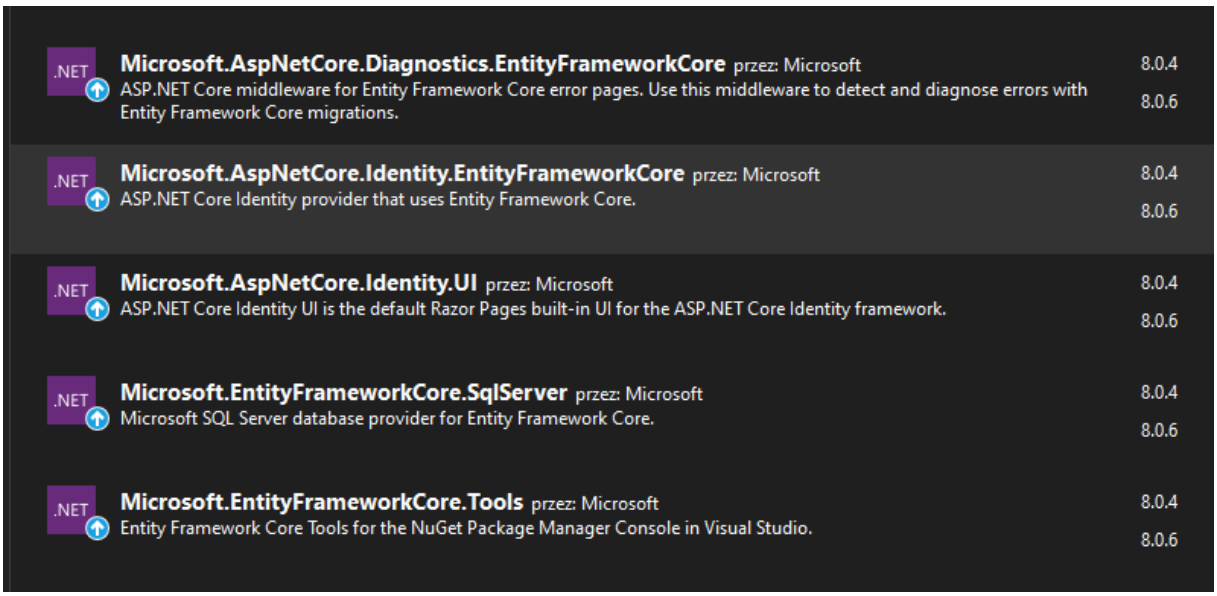
Autorzy:

Marcin Niedzielski

Andrzej Ratajczak

1. Wykorzystane technologie

Projekt został wykonany w ASP.NET 8.0 we wzorcu MVC z wykorzystaniem Entity Framework (Code First) oraz Identity ASP.NET Core



Lista zainstalowanych pakietów

Jako bazę danych wykorzystano domyśle rozwiązanie – lokalna baza danych SQL Server Express. Jej podgląd jest dostępny z poziomu Visual Studio za pomocą SQL Server Object Explorer.

Nazwa bazy danych w projekcie to" aspnet-PZ_Projekt-95eb8fde-4fe2-4beb-a143-7b2efe9ef99c". Jeśli na komputerze znajduję się już baza o takiej nazwie należy ją usunąć przed pierwszym uruchomieniem projektu.

2. Skrócony opis działania

Nasz projekt to prosty sklep internetowy sprzedający części komputerowe. Wyróżniamy w nim 3 poziomy dostępu:

- Użytkownik niezalogowany – może przeglądać podstawowe strony takie jak strona główna, lista produktów i kontakt
- Użytkownik zalogowany – po zalogowaniu użytkownik może dodawać produkty do koszyka oraz realizować i sprawdzać swoje zamówienie
- Administrator – ma dostęp do panelu administratora, który pozwala na zarządzanie użytkownikami, zamówieniami oraz produktami na stronie.

Produkty na stronie są dodawane przez Administratora, koszyk oraz zamówienia są przypisane do danego użytkownika i zapisywane w bazie danych. Dokładniejszy opis wszystkich funkcjonalności znajduje się w kolejnych punktach dokumentacji.

3. **Uruchomienie**
    - Sklonuj repozytorium GIT
    - Upewnij się, że na komputerze nie znajduje się baza danych o takiej samej nazwie jak w projekcie (" aspnet-PZ_Projekt-95eb8fde-4fe2-4beb-a143-7b2efe9ef99c")
    - W konsoli menedżera pakietów wpisz „update-database" (budowanie może zająć dłuższy czas ze względu na ładowanie do bazy danych początkowych produktów oraz użytkowników)
    - Po zakończeniu budowania bazy możesz skompilować i uruchomić projekt

    Jeśli podczas budowania bazy („update-database") pojawił się błąd możesz najpierw dodać migrację komendą „add-migration nazwa_migracji" a następnie wrócić do punktu c.

    Początkowi użytkownicy:

    - admin@admin.com – Administrator, hasło "Wsbtest:123"
    - a@a.com – użytkownik, hasło "Wsbtest:123"
4. **Struktura**
    - Connected Services
    - Properties
    - wwwroot
        - css
        - js
        - lib
        - uploads – folder z przesłanymi zdjęciami produktów
        - Cart.png – ikona koszyka
        - faviconPCP.ico – favicon strony
    - Zależności
    - Areas
    - Controllers
        - AdminController.cs
        - CartController.cs
        - HomeControler.cs
        - ItemController.cs
        - OrderController.cs
    - Data
        - Migrations
        - ApplicationDbContext.cs
    - Models
        - Cart.cs
        - CartItem.cs

        - ErrorViewModel.cs
        - Item.cs
        - Order.cs
        - OrderItem.cs
    - Views
        - Admin
            - AdminPanel.cshtml
            - ManageRoles.cshtml
        - Cart
            - Index.cshtml
        - Home
            - Contact.cshtml
            - Index.cshtml
            - Product.cshtml
        - Item
            - Create.cshtml
            - Delete.cshtml
            - Details.cshtml
            - Edit.cshtml
            - Index.cshtml
        - Order
            - AllOrders.cshtml
            - Checkout.cshtml
            - ConfirmRemoveOrder.cshtml
            - MyOrders.cshtml
            - OrderConfirmation.cshtml
        - Shared
            - _Layout.cshtml
            - _LoginPartial.cshtml
            - _ValidationScriptsPartial.cshtml
            - Error.cshtml
        - _ViewImports.cshtml
        - _ViewStart.cshtml
    - Appsettings.json
    - Program.cs

5. **Modele**
    - Cart.cs - Model koszyka, przechowuje informacje o przedmiotach w koszyku użytkownika

```
// Identyfikator koszyka
public int Id { get; set; }

// Identyfikator użytkownika, do którego przypisany jest koszyk
public string UserId { get; set; }

// Lista przedmiotów w koszyku
public List<CartItem> CartItems { get; set; }
```

    - CartItem.cs - Model elementu koszyka, pomocniczy do modelu Cart.cs

```
// Identyfikator elementu koszyka
 public int Id { get; set; }

 // Identyfikator przedmiotu
 public int ItemId { get; set; }

 // Referencja do przedmiotu
 public Item Item { get; set; }

 // Ilość danego przedmiotu w koszyku
 [Required]
 public int Quantity { get; set; }

 // Identyfikator użytkownika, do którego przypisany jest koszyk
 [Required]
 public string UserId { get; set; }
```

- ErrorViewModel.cs – model błędu widoku

```
public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
```

- Item.cs - Model produktu, przechowuje właściwości produktu

```
// Identyfikator przedmiotu
public int Id { get; set; }

// Nazwa przedmiotu, wymagana, maksymalna długość 70 znaków
[Required(ErrorMessage = "Pole Nazwa nie może być puste")]
[StringLength(70, MinimumLength = 1)]
public string Name { get; set; }

// Opis przedmiotu, wymagany, maksymalna długość 1000 znaków
[Required(ErrorMessage = "Pole Opis nie może być puste")]
[StringLength(1000)]
public string Description { get; set; }

// Cena przedmiotu, wymagana, zakres od 1 do 9999
[Required(ErrorMessage = "Pole Cena nie może być puste")]
[Range(1, 9999, ErrorMessage = "Cena musi być w zakresie od {1} do
{2}.")]
public decimal Price { get; set; }

// Kategoria przedmiotu
public string? Category { get; set; }

// Ścieżka do zapisanego obrazu na serwerze
public string? ImageUrl { get; set; }

// Wymagane zdjęcie (przesyłane przez formularz)
[Required(ErrorMessage = "Wymagane zdjęcie")]
[NotMapped]
[AllowedExtensions(new string[] { ".jpg", ".jpeg", ".png" })]
public IFormFile ImageFile { get; set; }
```

- Order.cs - Model zamówienia, przechowuje zamówienie złożone przez użytkownika, jaki użytkownik je złożył oraz status i datę zamówienia

```
// Identyfikator zamówienia
public int Id { get; set; }
```

```
// Identyfikator użytkownika, który złożył zamówienie
[Required]
public string UserId { get; set; }

// Data złożenia zamówienia
[Required]
public DateTime OrderDate { get; set; }

// Metoda dostawy zamówienia, wymagana
[Required(ErrorMessage = "Metoda dostawy jest wymagana.")]
public string DeliveryMethod { get; set; }

// Adres dostawy zamówienia, wymagany, maksymalna długość 100
znaków
[Required(ErrorMessage = "Adres dostawy jest wymagany.")]
[StringLength(100)]
public string Address { get; set; }

// Status zamówienia (np. "Nowe", "W trakcie", "Gotowe do
odbioru", itp.)
public string? Status { get; set; }

// Lista pozycji zamówienia
public List<OrderItem> OrderItems { get; set; }
```

- OrderItem.cs - Model pozycji zamówienia, pomocniczy do Order.cs

```
// Identyfikator pozycji zamówienia
public int Id { get; set; }

// Identyfikator zamówienia, do którego należy ta pozycja
[Required]
public int OrderId { get; set; }

// Zamówienie, do którego należy ta pozycja
public Order Order { get; set; }

// Identyfikator przedmiotu, który został zamówiony
[Required]
public int ItemId { get; set; }

// Przedmiot, który został zamówiony
public Item Item { get; set; }

// Ilość zamówionego przedmiotu
[Required]
public int Quantity { get; set; }
```

6. Kontrolery

- AdminController.cs – kontroler odpowiedzialny za panel administratora i zarządzanie rolami użytkowników, dostęp do niego wymaga uprawnień administratora.
  Metody:

```
// Pobieranie listy wszytskich użytkowników bez aktualnie
zalogowanego admina (żeby zapobiec usunięciu jedynego admina)
// GET: /Admin/ManageRoles
public async Task<IActionResult> ManageRoles()
```

```
{
  var currentUser = await _userManager.GetUserAsync(User);
  var users = await _userManager.Users
  .Where(u => u.Id != currentUser.Id)
  .ToListAsync();
  return View(users);
 }
```

```
// Dodawanie uprawnień Administratora
// POST: /Admin/AddToAdminRole
[HttpPost]
public async Task<IActionResult> AddToAdminRole(string userId)
{
    var user = await _userManager.FindByIdAsync(userId);
    if (user != null)
    {
        await _userManager.AddToRoleAsync(user, "Administrator");
        return RedirectToAction("ManageRoles");
    }
    return NotFound();
}
```

```
// Usuwanie uprawnień Administratora
// POST: /Admin/RemoveFromAdminRole
[HttpPost]
public async Task<IActionResult> RemoveFromAdminRole(string
userId)
{
    var user = await _userManager.FindByIdAsync(userId);
    if (user != null)
    {
        await _userManager.RemoveFromRoleAsync(user,
"Administrator");
        return RedirectToAction("ManageRoles");
    }
    return NotFound();
}
```

- CartController.cs – kontroler odpowiedzialny za koszyk - dodawanie do koszyka, wyświetlanie koszyka. Dostęp do niego jest możliwy tylko dla zalogowanych użytkowników.

```
// Wyświetlanie koszyka
public async Task<IActionResult> Index()
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    var cart = await _context.Cart
        .Include(c => c.CartItems)
        .ThenInclude(ci => ci.Item)
        .FirstOrDefaultAsync(c => c.UserId == userId);

    return View(cart?.CartItems ?? new List<CartItem>());
}
```

```
// Dodawanie produktu do koszyka
public async Task<IActionResult> AddToCart(int id)
{
    var item = await _context.Item.FindAsync(id);
    if (item == null)
    {
        return NotFound();
    }

    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
```

```
    var cart = await _context.Cart
        .Include(c => c.CartItems)
        .FirstOrDefaultAsync(c => c.UserId == userId);

    if (cart == null)
    {
        cart = new Cart { UserId = userId, CartItems = new
List<CartItem>() };
        _context.Cart.Add(cart);
    }

    var cartItem = cart.CartItems.FirstOrDefault(ci => ci.ItemId
== id);
    if (cartItem != null)
    {
        cartItem.Quantity++;
    }
    else
    {
        cart.CartItems.Add(new CartItem { Item = item, ItemId =
id, UserId = userId, Quantity = 1 });
    }

    await _context.SaveChangesAsync();

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

```
// Usuwanie z koszyka
public async Task<IActionResult> RemoveFromCart(int id)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    var cart = await _context.Cart
        .Include(c => c.CartItems)
        .FirstOrDefaultAsync(c => c.UserId == userId);

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

    var cartItem = cart.CartItems.FirstOrDefault(ci => ci.ItemId
== id);
    if (cartItem != null)
    {
        if (cartItem.Quantity > 1)
        {
            cartItem.Quantity--;
        }
        else
        {
            cart.CartItems.Remove(cartItem);
        }

        await _context.SaveChangesAsync();
    }

    return RedirectToAction("Index", "Cart");
}
```

- HomeControler.cs – kontroler odpowiedzialny za stronę główną, listę produktów oraz stronę z danymi kontaktowymi. Nie wymaga autoryzacji.

```
// Strona główna
public IActionResult Index()
{
    // Pobieranie z bazy 6 najnowszych produktów (największe ID)
    var latestItems = _context.Item
        .OrderByDescending(i => i.Id)
        .Take(6)
        .ToList();

    return View(latestItems);
}
```

```
// Strona z produktami
public IActionResult Products(string sortOrder, string
categoryFilter, string searchString)
{
    // Ustawienie parametrów sortowania i filtrów widoku
    ViewData["NameSortParam"] = string.IsNullOrEmpty(sortOrder) ?
"name_desc" : "";
    ViewData["PriceSortParam"] = sortOrder == "price_asc" ?
"price_desc" : "price_asc";
    ViewBag.SortOrder = sortOrder;
    ViewBag.CategoryFilter = categoryFilter;
    ViewBag.SearchString = searchString;

    // Zapytanie do bazy danych
    IQueryable<Item> itemsQuery = _context.Item.AsQueryable();

    // Zastosuj filtr kategorii, jeśli jest podany
    if (!string.IsNullOrEmpty(categoryFilter))
    {
        itemsQuery = itemsQuery.Where(item => item.Category ==
categoryFilter);
    }
    // Zastosuj wyszukiwanie, jeśli jest podane
    if (!string.IsNullOrEmpty(searchString))
    {
        itemsQuery = itemsQuery.Where(item =>
item.Name.Contains(searchString));
    }
    // Zastosuj sortowanie
    switch (sortOrder)
    {
        case "name_desc":
            itemsQuery = itemsQuery.OrderByDescending(i =>
i.Name);
            break;
        case "price_asc":
            itemsQuery = itemsQuery.OrderBy(i => i.Price);
            break;
        case "price_desc":
            itemsQuery = itemsQuery.OrderByDescending(i =>
i.Price);
            break;
        default:
            itemsQuery = itemsQuery.OrderBy(i => i.Name);
            break;
    }

    var items = itemsQuery.ToList();
    return View(items);
}
```

```
// Strona z danymi kontaktowymi
public IActionResult Contact()
```

```
{
    return View();
}
```

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

- ItemController.cs – Kontroler odpowiedzialny za produkty na stronie (operacje CRUD, lista produktów). Dostęp różni się w zależności od metody.

```
// autoryzacja użytkownika – wymagana rola "Administrator"
// Widok wszystkich produktów
[Authorize(Roles = "Administrator")]
// GET: Item/Index
public async Task<IActionResult> Index()
{
    var items = await _context.Item.ToListAsync();
    return View(items);
}
```

```
// Szczegóły danego produktu
// GET: Item/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

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

    return View(item);
}
```

```
// autoryzacja użytkownika – wymagana rola "Administrator"
// Wyświetlanie dodawania produktów
[Authorize(Roles = "Administrator")]
// GET: Item/Create
public IActionResult Create()
{
    return View();
}
```

```
// autoryzacja użytkownika – wymagana rola "Administrator"
// Akcja dodająca produkt
[Authorize(Roles = "Administrator")]
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Item item)
{
    if (ModelState.IsValid)
    {
        if (item.ImageFile != null && item.ImageFile.Length > 0)
        {
```

```
            var fileName = Guid.NewGuid().ToString() +
Path.GetExtension(item.ImageFile.FileName);
            var filePath =
Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
"uploads", fileName);
            using (var stream = new FileStream(filePath,
FileMode.Create))
            {
                await item.ImageFile.CopyToAsync(stream);
            }
            item.ImageUrl = "/uploads/" + fileName;
        }
        _context.Add(item);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(item);
}
```

```
// autoryzacja użytkownika - wymagana rola "Administrator"
// Wyświetlanie edycji produktu
[Authorize(Roles = "Administrator")]
// GET: Item/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var item = await _context.Item.FindAsync(id);
    if (item == null)
    {
        return NotFound();
    }
    return View(item);
}
```

```
// autoryzacja użytkownika - wymagana rola "Administrator"
 // Akcja edytująca produkt
 [Authorize(Roles = "Administrator")]
 [HttpPost]
 [ValidateAntiForgeryToken]
 public async Task<IActionResult> Edit(int id, Item item)
 {
     if (id != item.Id)
     {
         return NotFound();
     }

     if (ModelState.IsValid)
     {
         try
         {
             // Unikanie śledzenia istniejącej encji
             _context.Entry(item).State = EntityState.Modified;

             // Sprawdzanie, czy nowy obraz został przesłany
             if (item.ImageFile == null)
             {
                 // Jeśli nie przesłano nowego obrazka, zachowaj
istniejącą wartość ImageUrl
                 var existingItem = await
_context.Item.FindAsync(id);
                 item.ImageUrl = existingItem.ImageUrl;
```

```
            }
            else
            {
                // Jeśli przesłano nowy obrazek, zapisz go i
zaktualizuj ImageUrl
                var fileName = Guid.NewGuid().ToString() +
Path.GetExtension(item.ImageFile.FileName);
                var filePath =
Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
"uploads", fileName);
                using (var stream = new FileStream(filePath,
FileMode.Create))
                {
                    await item.ImageFile.CopyToAsync(stream);
                }
                item.ImageUrl = "/uploads/" + fileName;
            }

            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!ItemExists(item.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(item);
}
```

```
      // autoryzacja użytkownika - wymagana rola "Administrator"
// Wyświetlanie usuwania produktu
[Authorize(Roles = "Administrator")]
// GET: Item/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

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

    return View(item);
}
```

```
// autoryzacja użytkownika - wymagana rola "Administrator"
// Usuwanie produktu
[Authorize(Roles = "Administrator")]
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var item = await _context.Item.FindAsync(id);
```

```
    if (item != null)
    {
        _context.Item.Remove(item);
        await _context.SaveChangesAsync();
    }
    return RedirectToAction(nameof(Index));
}
```

```
// autoryzacja użytkownika – wymagana rola "Administrator"
// Sprawdzanie czy przedmiot istnieje
[Authorize(Roles = "Administrator")]
private bool ItemExists(int id)
{
    return _context.Item.Any(e => e.Id == id);
}
```

```
// Walidacja dla rozszerzeń plików (do przesyłania obrazów)
 public class AllowedExtensionsAttribute : ValidationAttribute
 {
     private readonly string[] _extensions;

     public AllowedExtensionsAttribute(string[] extensions)
     {
         _extensions = extensions;
     }

     protected override ValidationResult IsValid(object value,
ValidationContext validationContext)
     {
         if (value == null)
             return ValidationResult.Success;

         var file = value as IFormFile;
         var extension = Path.GetExtension(file.FileName);

         if (file == null || Array.IndexOf(_extensions,
extension.ToLower()) == -1)
         {
             return new ValidationResult(GetErrorMessage());
         }

         return ValidationResult.Success;
     }
     private string GetErrorMessage()
     {
         return $"Dozwolone rozszerzenia plików: {string.Join(",
", _extensions)}";
     }
 }
```

- OrderController.cs - Kontroler obsługuje widoki w folderze Oder (wyświetlanie i usuwanie zamówień, zmiana statusu). Dostęp tylko dla zalogowanych użytkowników.

```
       // Wyświetlanie podsumowania zamówienia
 public async Task<IActionResult> Checkout()
 {
     var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
     var cart = await _context.Cart
         .Include(c => c.CartItems)
         .ThenInclude(ci => ci.Item)
         .FirstOrDefaultAsync(c => c.UserId == userId);

     if (cart == null || !cart.CartItems.Any())
```

```
     {
         return RedirectToAction("Index", "Cart");
     }

     var order = new Order
     {
         UserId = userId,
         OrderDate = DateTime.Now,
         DeliveryMethod = "Pickup", // Domyślna metoda dostawy
         Address = "", // Puste pole adresu, do uzupełnienia przez
użytkownika
         Status = "Nowe", // Domyślny status
         OrderItems = cart.CartItems != null ?
cart.CartItems.Select(ci => new OrderItem
         {
             ItemId = ci.ItemId,
             Item = ci.Item,
             Quantity = ci.Quantity
         }).ToList() : new List<OrderItem>() // Zabezpieczenie
przed null exception
     };

     return View(order);
 }
```

```
// Składanie zamówienia
 [HttpPost]
 public async Task<IActionResult> Checkout(Order order, string
deliveryMethod, string address)
 {
     var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

     var cart = await _context.Cart
         .Include(c => c.CartItems)
         .FirstOrDefaultAsync(c => c.UserId == userId);

     if (cart == null || !cart.CartItems.Any())
     {
         return RedirectToAction("Index", "Cart");
     }

     if (string.IsNullOrEmpty(deliveryMethod) ||
string.IsNullOrEmpty(address))
     {
         ModelState.AddModelError("", "Delivery method and address
are required.");

         // Przekazujemy model Order ponownie do widoku
         return View(order);
     }

     // Jeśli istnieje ID zamówienia, pobierz istniejący obiekt
Order z bazy danych
     if (order.Id != 0)
     {
         var existingOrder = await _context.Order
             .Include(o => o.OrderItems)
             .FirstOrDefaultAsync(o => o.Id == order.Id);

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

```
            // Zaktualizuj istniejący obiekt Order
            existingOrder.DeliveryMethod = deliveryMethod;
            existingOrder.Address = address;

            _context.Order.Update(existingOrder);
        }
        else
        {
            // Utwórz nowy obiekt Order
            order.UserId = userId;
            order.OrderDate = DateTime.Now;
            order.DeliveryMethod = deliveryMethod;
            order.Address = address;
            order.OrderItems = cart.CartItems.Select(ci => new
OrderItem
            {
                ItemId = ci.ItemId,
                Quantity = ci.Quantity
            }).ToList();

            _context.Order.Add(order);
        }

        // Usuń koszyk i zapisz zmiany w bazie danych
        _context.CartItem.RemoveRange(cart.CartItems);
        _context.Cart.Remove(cart);
        await _context.SaveChangesAsync();

        return RedirectToAction("OrderConfirmation", new { orderId =
order.Id });
    }
```

```
// Zmiana statusu zamówienia - tylko dla Administratora
 [HttpPost]
 [Authorize(Roles = "Administrator")]
 public async Task<IActionResult> ChangeOrderStatus(int id, string
status)
 {
     var order = await _context.Order.FindAsync(id);
     if (order == null)
     {
         return NotFound();
     }


     if (!IsValidOrderStatus(status))
     {
         ModelState.AddModelError("", "Invalid order status.");
         return RedirectToAction(nameof(AllOrders));
     }


     order.Status = status;

     await _context.SaveChangesAsync();

     // Przekierowanie użytkownika na tę samą stronę
     return RedirectToAction(nameof(AllOrders));
 }
```

```
private bool IsValidOrderStatus(string status)
{
    // Lista prawidłowych statusów zamówienia
```

```
    var validStatuses = new List<string> { "Nowe", "W trakcie",
"Gotowe do odbioru", "Gotowe do wysyłki", "Wysłane", "Zakończone"
};
    return validStatuses.Contains(status,
StringComparer.OrdinalIgnoreCase);
}
```

```
//Wyświetlanie potwierdzenia zamówienia
public async Task<IActionResult> OrderConfirmation(int orderId)
{
    var order = await _context.Order
        .Include(o => o.OrderItems)
        .ThenInclude(oi => oi.Item)
        .FirstOrDefaultAsync(o => o.Id == orderId);

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

    return View(order);
}
```

```
      //Wyświetlanie listy zamówień zalogowanego użytkownika
public async Task<IActionResult> MyOrders()
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    var orders = await _context.Order
        .Where(o => o.UserId == userId)
        .Include(o => o.OrderItems)
        .ThenInclude(oi => oi.Item)
        .ToListAsync();

    return View(orders);
}
```

```
//Wyświetlanie listy wszystkich zamówień
[Authorize(Roles = "Administrator")]
public async Task<IActionResult> AllOrders()
{
    var orders = await _context.Order
        .Include(o => o.OrderItems)
        .ThenInclude(oi => oi.Item)
        .ToListAsync();

    // Pobierz słownik par Id użytkownika - Nazwa użytkownika
    var userIdToNameMap = await GetUserIdToNameMap();

    // Przekazuj słownik do widoku wraz z zamówieniami
    ViewData["UserIdToNameMap"] = userIdToNameMap;

    return View(orders);
}
```

```
// Metoda pobierająca mapowanie ID użytkownika na nazwę
użytkownika ( wyświetlanie nazwy użytkownika zamiast ID)
private async Task<Dictionary<string, string>>
GetUserIdToNameMap()
{
    // Pobierz wszystkich użytkowników z bazy danych
    var users = await _userManager.Users.ToListAsync();

    // Utwórz słownik Id użytkownika - Nazwa użytkownika
    var userIdToNameMap = users.ToDictionary(u => u.Id, u =>
u.UserName);

    return userIdToNameMap;
```

```
}
```

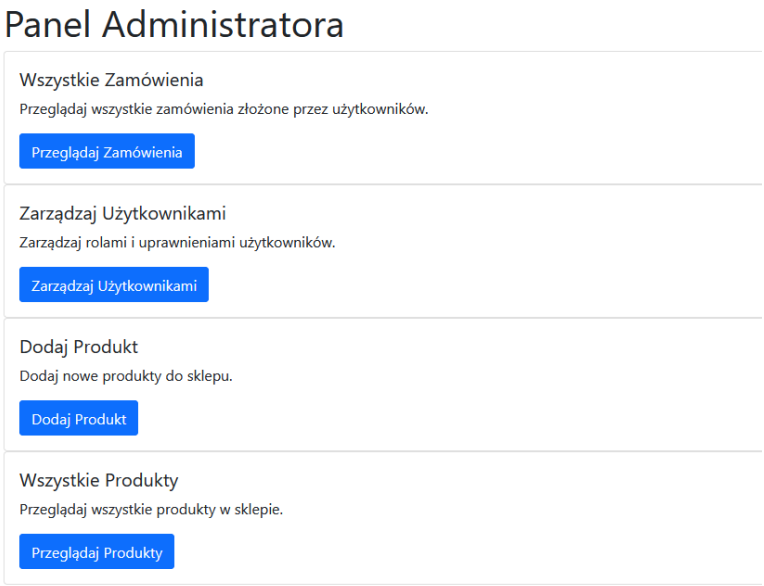
```
// Usuwanie zamówienia
 [HttpPost]
 [Authorize(Roles = "Administrator")]
 public async Task<IActionResult> RemoveOrder(int id)
 {
     var order = await _context.Order.FindAsync(id);
     if (order == null)
     {
         return NotFound();
     }

     return View("ConfirmRemoveOrder", order);
 }
```

```
// Potwierdzenie usuwania zamówienia
 [HttpPost]
 [Authorize(Roles = "Administrator")]
 public async Task<IActionResult> ConfirmRemoveOrder(int id)
 {
     var order = await _context.Order.FindAsync(id);
     if (order == null)
     {
         return NotFound();
     }

     _context.Order.Remove(order);
     await _context.SaveChangesAsync();

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

```
// Sprawdzanie czy zamówienie istnieje
private bool OrderExists(int id)
{
    return _context.Order.Any(e => e.Id == id);
}
```

7. Widoki

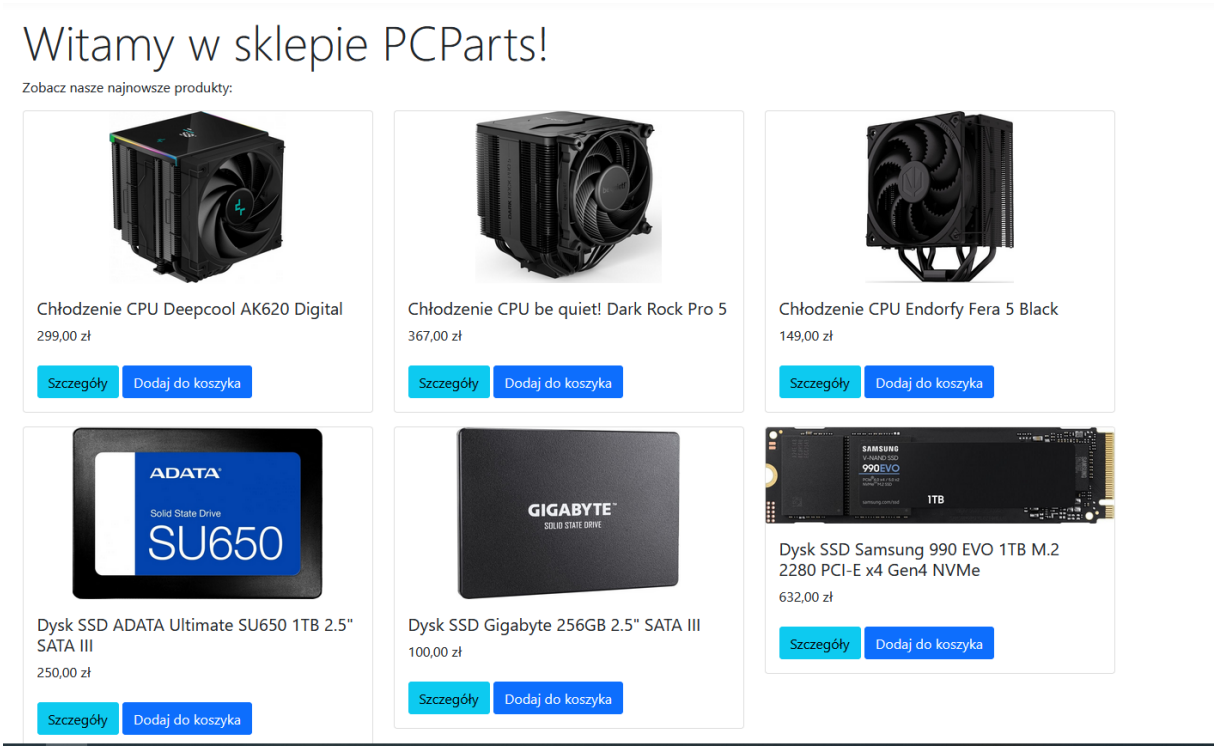
- Admin – kontroler obsługujący „AdminController.cs”
    - AdminPanel.cshtml – pozwala przejść do poszczególnych funkcjonalności dostępnych dla Administratora



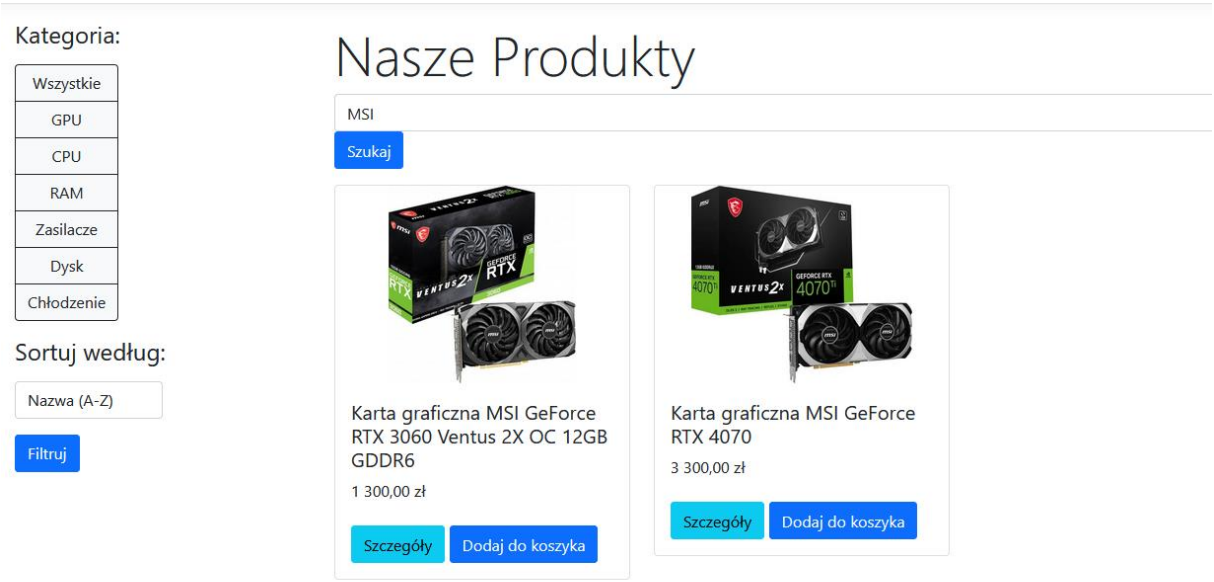
- ManageRoles.cshtml – pozwala przypisać uprawnienia wszystkim użytkownikom z pominięciem aktualnie zalogowanego administartora

- Cart – kontroler obsługujący „CartController.cs”
  - Index.cshtml – wyświetla zawartość koszyka aktualnie zalogowanego użytkownika

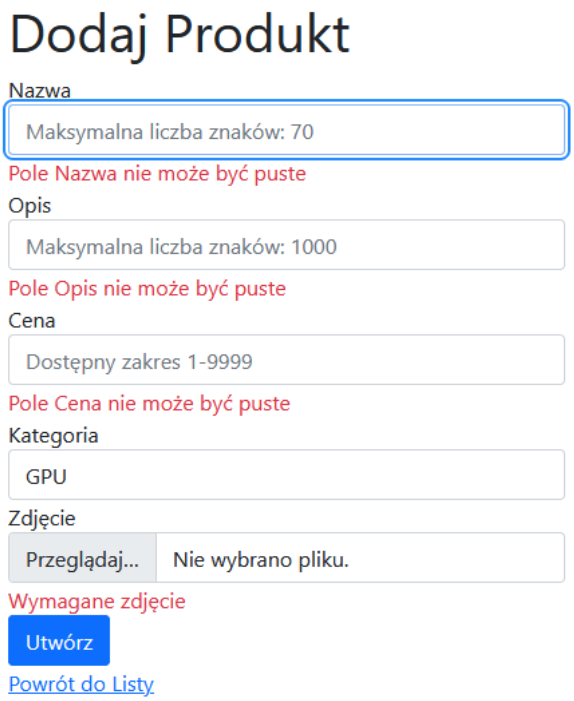
- Home – kontroler obsługujący „HomeController.cs”
  - Contact.cshtml – strona z danymi kontaktowymi
  - Index.cshtml – strona główna z nowościami, wyświetla 6 najnowszych (o najwyższym ID) produktów



- Product.cshtml – lista wszystkich produktów w sklepie, posiada wyszukiwarkę tekstową oraz system filtrów które współdziałają ze sobą. Czyli możemy wyszukać wszystkie produkty w kategorii GPU z frazą „MSI” i ustawić je względem ceny.

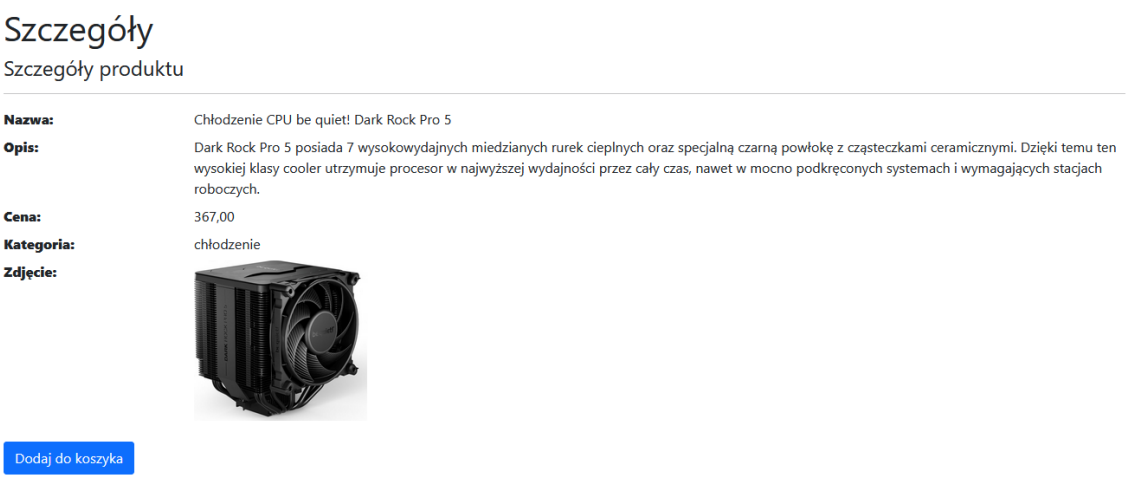


- Item – kontroler obsługujący „ItemController.cs”
  - Create.cshtml – dodawanie produktów do systemu, dostępne tylko dla Administratora. Pola są walidowane i wyświetlane są informację o ograniczeniach pól. Dodatkowo wgrywanie zdjęcia jest zabezpieczone przed wgraniem innych plików niż graficzne.

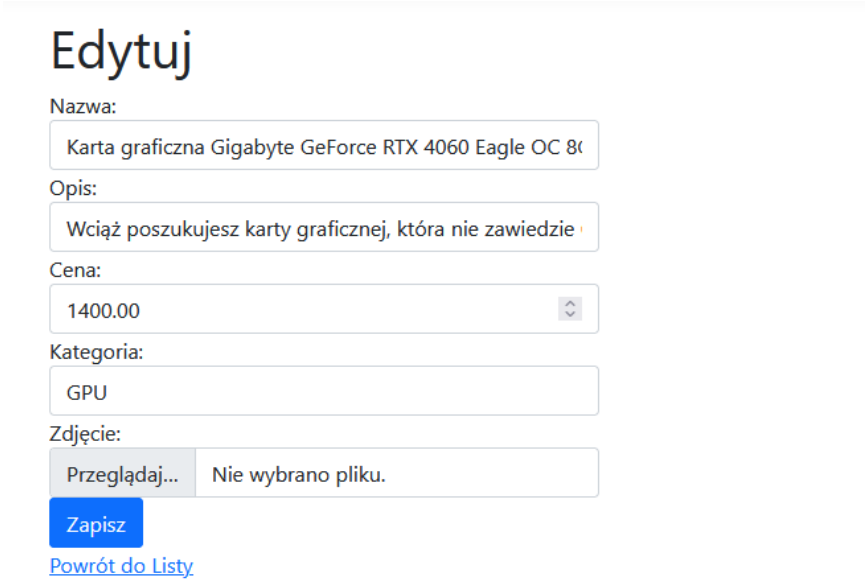


- Delete.cshtml – usuwanie produktu z systemu, dostępne tylko dla Administratora

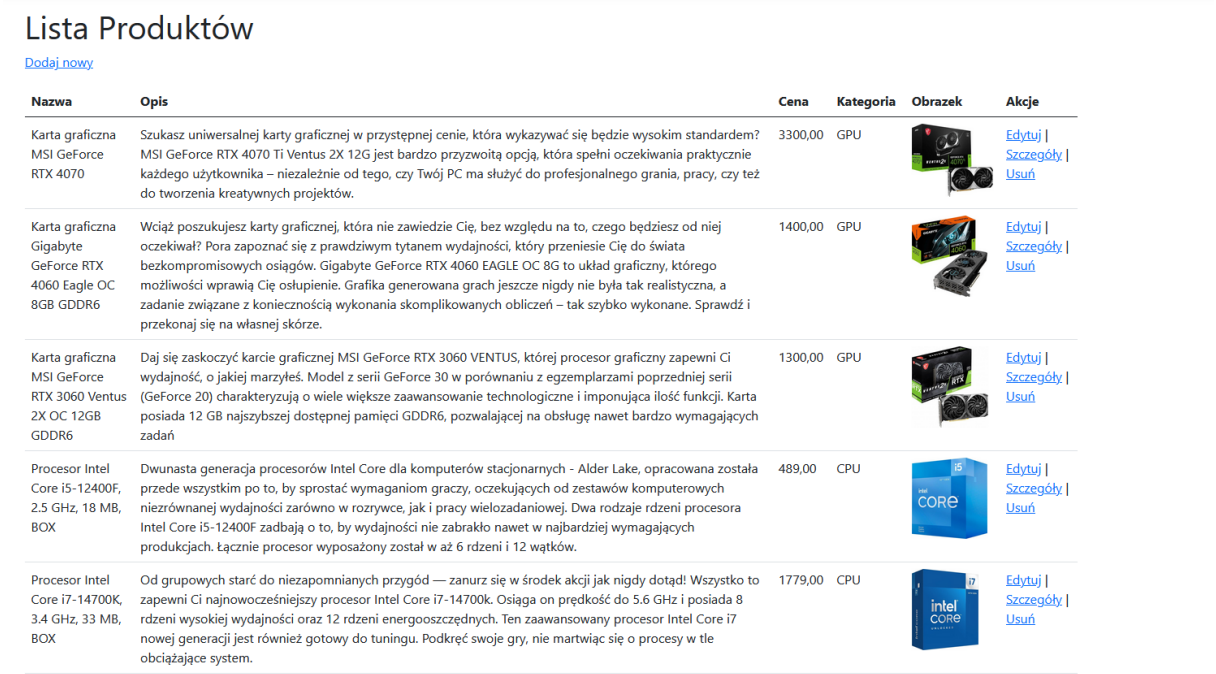
- Details.cshtml – wyświetlanie szczegółów produktu, można z tego widoku dodać produkt do koszyka. Nie wymaga uwierzytelniania



- Edit.cshtml – edycja produktu w systemie, analogicznie jak tworzenie nowego produktu, dostępne tylko dla administratora



- Index.cshtml – lista wszystkich produktów z możliwością edycji, usunięcia itp. Dostępne tylko dla Administratora

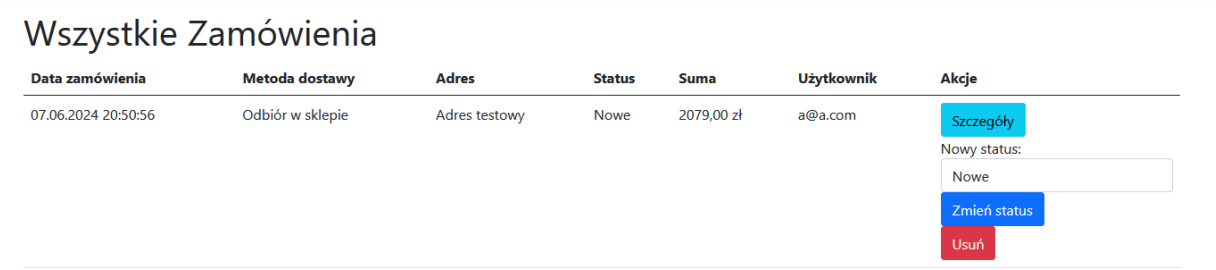


Lista Produktów

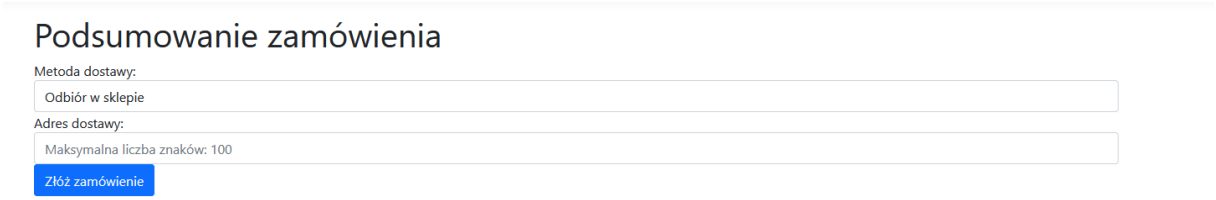
Dodaj nowy

Nazwa	Opis	Cena	Kategoria	Obrazek	Akcje
Karta graficzna MSI GeForce RTX 4070	Szukasz uniwersalnej karty graficznej w przystępnej cenie, która wykazywać się będzie wysokim standardem? MSI GeForce RTX 4070 Ti Ventus 2X 12G jest bardzo przyzwoitą opcją, która spełni oczekiwania praktycznie każdego użytkownika – niezależnie od tego, czy Twój PC ma służyć do profesjonalnego grania, pracy, czy też do tworzenia kreatywnych projektów.	3300,00	GPU		Edytuj | Szczegóły | Usuń
Karta graficzna Gigabyte GeForce RTX 4060 Eagle OC 8GB GDDR6	Wciąż poszukujesz karty graficznej, która nie zawiedzie Cię, bez względu na to, czego będziesz od niej oczekiwał? Pora zapoznać się z prawdziwym tytanem wydajności, który przeniesie Cię do świata bezkompromisowych osiągów. Gigabyte GeForce RTX 4060 EAGLE OC 8G to układ graficzny, którego możliwości wprawią Cię osłupienie. Grafika generowana grach jeszcze nigdy nie była tak realistyczna, a zadanie związane z koniecznością wykonania skomplikowanych obliczeń – tak szybko wykonane. Sprawdź i przekonaj się na własnej skórze.	1400,00	GPU		Edytuj | Szczegóły | Usuń
Karta graficzna MSI GeForce RTX 3060 Ventus 2X OC 12GB GDDR6	Daj się zaskoczyć karcie graficznej MSI GeForce RTX 3060 VENTUS, której procesor graficzny zapewni Ci wydajność, o jakiej marzyłeś. Model z serii GeForce 30 w porównaniu z egzemplarzami poprzedniej serii (GeForce 20) charakteryzują o wiele większe zaawansowanie technologiczne i imponująca ilość funkcji. Karta posiada 12 GB najszybszej dostępnej pamięci GDDR6, pozwalającej na obsługę nawet bardzo wymagających zadań	1300,00	GPU		Edytuj | Szczegóły | Usuń
Procesor Intel Core i5-12400F, 2.5 GHz, 18 MB, BOX	Dwunasta generacja procesorów Intel Core dla komputerów stacjonarnych - Alder Lake, opracowana została przede wszystkim po to, by sprostać wymaganiom graczy, oczekujących od zestawów komputerowych niezrównanej wydajności zarówno w rozrywce, jak i pracy wielozadaniowej. Dwa rodzaje rdzeni procesora Intel Core i5-12400F zadbają o to, by wydajności nie zabrakło nawet w najbardziej wymagających produkcjach. Łącznie procesor wyposażony został w aż 6 rdzeni i 12 wątków.	489,00	CPU		Edytuj | Szczegóły | Usuń
Procesor Intel Core i7-14700K, 3.4 GHz, 33 MB, BOX	Od grupowych starć do niezapomnianych przygód — zanurz się w środek akcji jak nigdy dotąd! Wszystko to zapewni Ci najnowocześniejszy procesor Intel Core i7-14700k. Osiąga on prędkość do 5.6 GHz i posiada 8 rdzeni wysokiej wydajności oraz 12 rdzeni energooszczędnych. Ten zaawansowany procesor Intel Core i7 nowej generacji jest również gotowy do tuningu. Podkręć swoje gry, nie martwiąc się o procesy w tle obciążające system.	1779,00	CPU		Edytuj | Szczegóły | Usuń

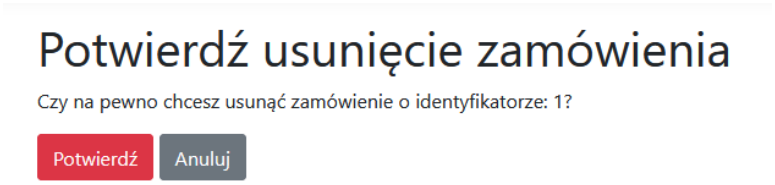
- Order – kontroler obsługujący „OrderController.cs”
  - AllOrders.cshtml – widok dostępny dla Administratora wyświetlający wszystkie zamówienia, umożliwia zmianę statusu zamówienia.



- Checkout.cshtml – widok z podsumowaniem zamówienia, użytkownik musi wybrać metodę dostawy oraz adres. Dostępny tylko dla zalogowanych użytkowników. Kwota zamówienia jest wyświetlana w koszyku.



- ConfirmRemoveOrder.cshtml – widok potwierdzający usunięcie zamówienia



- MyOrders.cshtml – lista zamówień zalogowanego użytkownika

- OrderConfirmation.cshtml – potwierdzenie dokonania zamówienia

8. System użytkowników

Wyróżniamy 3 poziomy dostępu:

- Użytkownik niezalogowany – może przeglądać podstawowe strony takie jak strona główna, lista produktów i kontakt
- Użytkownik zalogowany – po zalogowaniu użytkownik może dodawać produkty do koszyka oraz realizować i sprawdzać swoje zamówienie. Widoki związane z zamówieniami oraz koszyk są personalizowane dla aktualnie zalogowanego użytkownika.
- Administrator – ma dostęp do panelu administratora, który pozwala na zarządzanie użytkownikami, zamówieniami oraz produktami na stronie.

Nadawanie ról użytkownikom jest dostępne w panelu Administratora w widoku ManageRoles.cshtml.

Początkowi użytkownicy:

- admin@admin.com – Administrator, hasło “Wsbtest:123”
- a@a.com – użytkownik, hasło “Wsbtest:123”