

# Entity Framework, LINQ2Entities

## Tomasz Zawadzki

Pierwsza część sprawozdania zawiera rozwiązania zadań z punktu instrukcji **V. Method syntax vs query syntax**, którego nie zdążyłem zrealizować podczas laboratorium.

Druga część sprawozdania opisuje aplikację przygotowaną jako zadanie domowe. Zawiera ona fragmenty kodu źródłowego, zrzuty ekranu oraz krótki opis wykorzystanych mechanizmów.

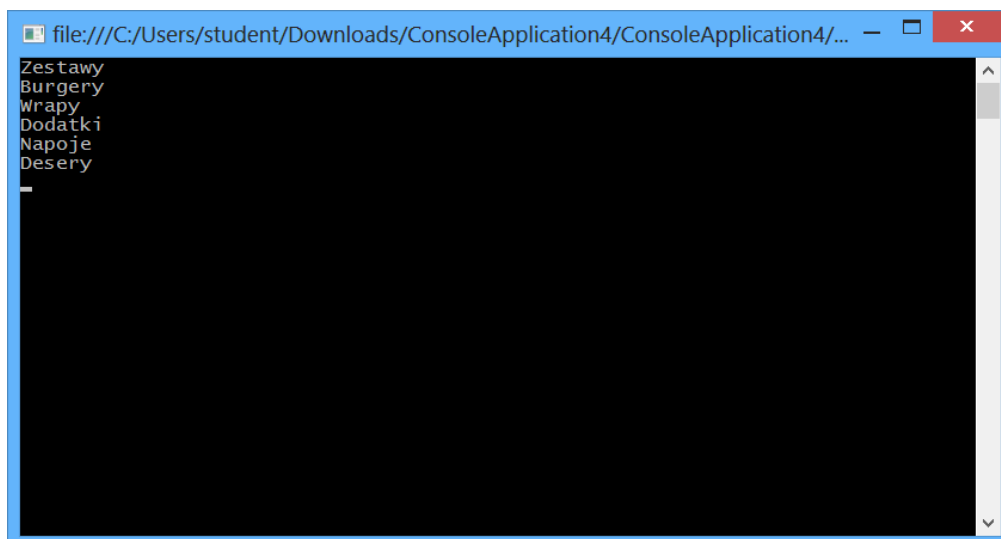
## V. Method syntax vs query syntax

### Methoda pobierająca i wypisująca nazwy kategorii

```
private static void ShowCategories(ProdContext db)
{
    IQueryable<String> categoriesNamesQuery = db.Categories.Select(c => c.Name);
    foreach (var name in categoriesNamesQuery)
    {
        Console.WriteLine(name);
    }
}
```

Wynik uruchomienia poniższego programu:

```
using (var db = new ProdContext())
{
    ShowCategories(db);
    Console.ReadKey();
}
```



Po osiągnięciu breakpointa przy definicji pierwszego zapytania program został wykonany krokowo. SQL Profiler wyświetlił następującą sekwencję zapytań:

SQL Server Profiler - [Untitled - 3 ((localdb)\v11.0)]										
EventClass	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration	ClientProcessID	SPID
Trace Start										
Audit Login	-- network protocol: Named Pipes s...	EntityFrame...	student	asw\st...					3892	52
Audit Login	-- network protocol: Named Pipes s...	EntityFrame...	student	asw\st...					3892	53
SQL:BatchStarting	SELECT Count(*) FROM sys.databases ...	EntityFrame...	student	asw\st...					3892	53
SQL:BatchCompleted	SELECT Count(*) FROM sys.databases ...	EntityFrame...	student	asw\st...	0	4	0	0	3892	53
Audit Logout		EntityFrame...	student	asw\st...	0	0	0	250	3892	52
RPC:Completed	exec sp_reset_connection	EntityFrame...	student	asw\st...	0	0	0	0	3892	52
Audit Login	-- network protocol: Named Pipes s...	EntityFrame...	student	asw\st...					3892	52
SQL:BatchStarting	SELECT TABLE_SCHEMA SchemaName, TAB...	EntityFrame...	student	asw\st...					3892	52
SQL:BatchCompleted	SELECT TABLE_SCHEMA SchemaName, TAB...	EntityFrame...	student	asw\st...	31	341	0	31	3892	52
Audit Logout		EntityFrame...	student	asw\st...	0	4	0	113	3892	53
RPC:Completed	exec sp_reset_connection	EntityFrame...	student	asw\st...	0	0	0	0	3892	53
Audit Login	-- network protocol: Named Pipes s...	EntityFrame...	student	asw\st...					3892	53
SQL:BatchStarting	SELECT Count(*) FROM sys.databases ...	EntityFrame...	student	asw\st...					3892	53
SQL:BatchCompleted	SELECT Count(*) FROM sys.databases ...	EntityFrame...	student	asw\st...	0	4	0	0	3892	53
Audit Logout		EntityFrame...	student	asw\st...	31	341	0	156	3892	52
RPC:Completed	exec sp_reset_connection	EntityFrame...	student	asw\st...	0	0	0	0	3892	52
Audit Login	-- network protocol: Named Pipes s...	EntityFrame...	student	asw\st...					3892	52
SQL:BatchStarting	SELECT [GroupBy1].[A1] AS [C1] F...	EntityFrame...	student	asw\st...					3892	52
SQL:BatchCompleted	SELECT [GroupBy1].[A1] AS [C1] F...	EntityFrame...	student	asw\st...	0	56	0	0	3892	52
Audit Logout		EntityFrame...	student	asw\st...	0	397	0	16	3892	52
RPC:Completed	exec sp_reset_connection	EntityFrame...	student	asw\st...	0	0	0	0	3892	52
Audit Login	-- network protocol: Named Pipes s...	EntityFrame...	student	asw\st...					3892	52
SQL:BatchStarting	SELECT TOP (1) [Project1].[C1] AS...	EntityFrame...	student	asw\st...					3892	52
SQL:BatchCompleted	SELECT TOP (1) [Project1].[C1] AS...	EntityFrame...	student	asw\st...	0	2	0	0	3892	52
Audit Logout		EntityFrame...	student	asw\st...	0	399	0	120	3892	52
RPC:Completed	exec sp_reset_connection	EntityFrame...	student	asw\st...	0	0	0	0	3892	52
Audit Login	-- network protocol: Named Pipes s...	EntityFrame...	student	asw\st...					3892	52
SQL:BatchStarting	SELECT [Extent1].[CategoryID] AS ...	EntityFrame...	student	asw\st...					3892	52
SQL:BatchCompleted	SELECT [Extent1].[CategoryID] AS ...	EntityFrame...	student	asw\st...	0	59	0	0	3892	52

```

SELECT
[Extent1].[CategoryID] AS [CategoryID],
[Extent1].[Name] AS [Name],
[Extent1].[Description] AS [Description]
FROM [dbo].[Categories] AS [Extent1]

```

Trace is running. Ln 30, Col 2 Rows: 30 Connections: 1

# Metody pobierające i wyświetlające wszystkie kategorie i produkty

## Joiny, query syntax

```
static void ShowCategoriesProducts_Join_QuerySyntax(ProdContext db)
{
    var query = from category in db.Categories
                join product in db.Products
                on category.CategoryID equals product.CategoryID
                select new
                {
                    Category = category,
                    Product = product
                };

    foreach (var row in query) {
        Console.WriteLine("{0} ({1})", row.Product.Name, row.Category.Name);
    }
}
```

## Joiny, method syntax

```
static void ShowCategoriesProducts_Join_MethodSyntax(ProdContext db)
{
    var query = db.Categories.Join(
        db.Products,
        product => product.CategoryID,
        category => category.CategoryID,
        (category, product) => new
        {
            Category = category,
            Product = product
        });

    foreach (var row in query)
    {
        Console.WriteLine("{0} ({1})", row.Product.Name, row.Category.Name);
    }
}
```

## Navigation property, lazy loading, query syntax

```
static void ShowCategoriesProducts_NavigationProperty_LazyLoading_QuerySyntax(ProdContext db)
{
    var categoriesQuery = from category in db.Categories select category;
    foreach (var category in categoriesQuery)
    {
        foreach (var product in category.Products)
        {
            Console.WriteLine("{0} ({1})", product.Name, category.Name);
        }
    }
}
```

## Navigation property, lazy loading, method syntax

```
static void ShowCategoriesProducts_NavigationProperty_LazyLoading_MethodSyntax(ProdContext db)
{
    foreach (var category in db.Categories)
    {
        foreach (var product in category.Products)
        {
            Console.WriteLine("{0} ({1})", product.Name, category.Name);
        }
    }
}
```

## Navigation property, eager loading, query syntax

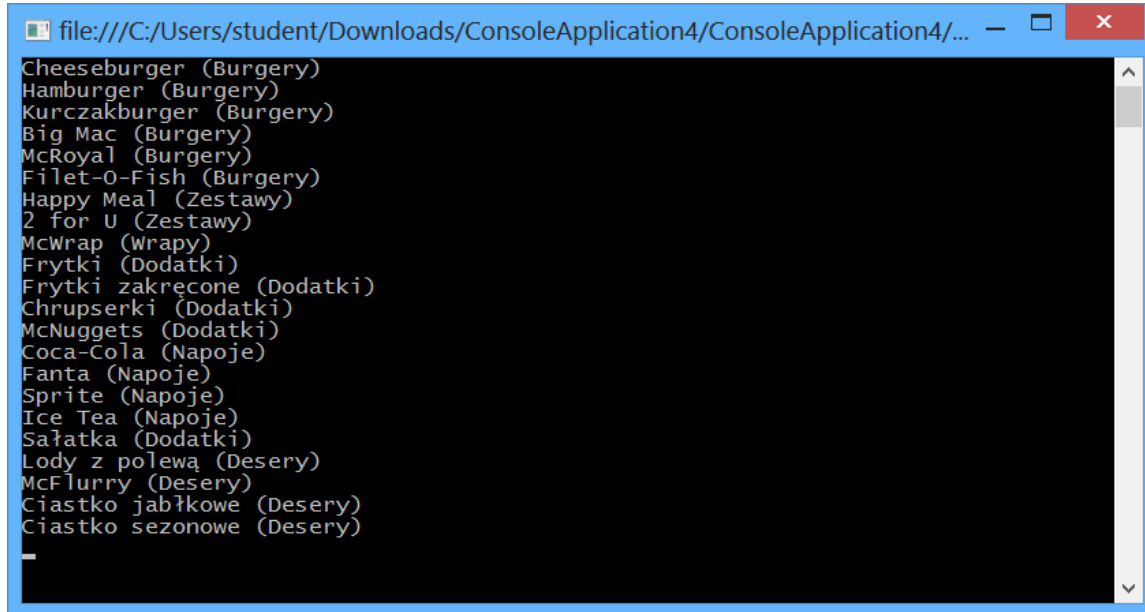
```
static void ShowCategoriesProducts_NavigationProperty_EagerLoading_QuerySyntax(ProdContext db)
{
    var categoriesQuery = from category in db.Categories
                          select new
                          {
                              Category = category,
                              Products = category.Products
                          };
    foreach (var category in categoriesQuery)
    {
        foreach (var product in category.Products)
        {
            Console.WriteLine("{0} ({1})", product.Name, category.Category.Name);
        }
    }
}
```

## Navigation property, eager loading, method syntax

```
static void ShowCategoriesProducts_NavigationProperty_EagerLoading_MethodSyntax(ProdContext db)
{
    var categoriesQuery = db.Categories.Include(c => c.Products);
    foreach (var category in categoriesQuery)
    {
        foreach (var product in category.Products)
        {
            Console.WriteLine("{0} ({1})", product.Name, category.Name);
        }
    }
}
```

Wynik uruchomienia poniższego programu:

```
using (var db = new ProdContext())
{
    ShowCategoriesProducts_QuerySyntax_Join(db);
    Console.ReadKey();
}
```



The screenshot shows a Windows console window titled "file:///C:/Users/student/Downloads/ConsoleApplication4/ConsoleApplication4/...". The console output lists 25 menu items, each followed by its category in parentheses. The items are: Cheeseburger (Burgery), Hamburger (Burgery), Kurczakburger (Burgery), Big Mac (Burgery), McRoyal (Burgery), Filet-O-Fish (Burgery), Happy Meal (Zestawy), 2 for U (Zestawy), McWrap (Wrapy), Frytki (Dodatki), Frytki zakrecone (Dodatki), Chrupserki (Dodatki), McNuggets (Dodatki), Coca-Cola (Napoje), Fanta (Napoje), Sprite (Napoje), Ice Tea (Napoje), Sałatka (Dodatki), Lody z polewą (Desery), McFlurry (Desery), Ciastko jabłkowe (Desery), and Ciastko sezonowe (Desery). The list is displayed on a black background with white text.

```
file:///C:/Users/student/Downloads/ConsoleApplication4/ConsoleApplication4/...
Cheeseburger (Burgery)
Hamburger (Burgery)
Kurczakburger (Burgery)
Big Mac (Burgery)
McRoyal (Burgery)
Filet-O-Fish (Burgery)
Happy Meal (Zestawy)
2 for U (Zestawy)
McWrap (Wrapy)
Frytki (Dodatki)
Frytki zakrecone (Dodatki)
Chrupserki (Dodatki)
McNuggets (Dodatki)
Coca-Cola (Napoje)
Fanta (Napoje)
Sprite (Napoje)
Ice Tea (Napoje)
Sałatka (Dodatki)
Lody z polewą (Desery)
McFlurry (Desery)
Ciastko jabłkowe (Desery)
Ciastko sezonowe (Desery)
```

# Metoda wyświetlająca wszystkie kategorie wraz z liczbą produktów

## Query syntax

```
static void ShowCategoriesWithProductsCount_QuerySyntax(ProdContext db)
{
    var query = from category in db.Categories
                join product in db.Products
                on category.CategoryID equals product.CategoryID
                into productsGroup
                select new
                {
                    Category = category,
                    ProductsCount = productsGroup.Count()
                };

    foreach (var row in query)
    {
        Console.WriteLine("{0} ({1})", row.Category.Name, row.ProductsCount);
    }
}
```

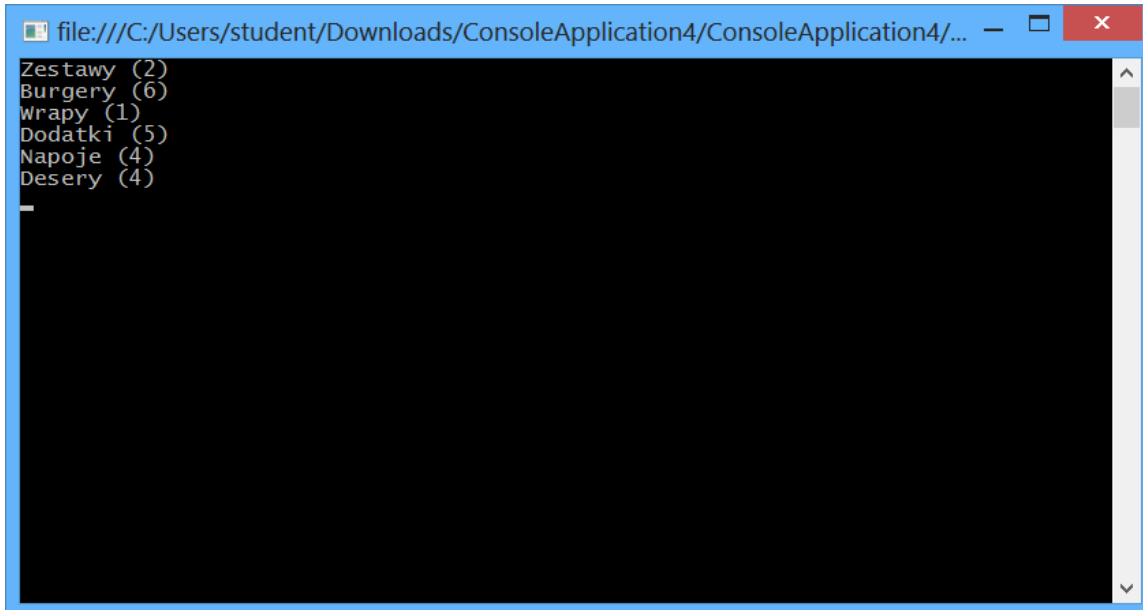
## Method syntax

```
static void ShowCategoriesWithProductsCount_QuerySyntax(ProdContext db)
{
    var query = from category in db.Categories
                join product in db.Products
                on category.CategoryID equals product.CategoryID
                into productsGroup
                select new
                {
                    Category = category,
                    ProductsCount = productsGroup.Count()
                };

    foreach (var row in query)
    {
        Console.WriteLine("{0} ({1})", row.Category.Name, row.ProductsCount);
    }
}
```

Wynik uruchomienia poniższego programu:

```
using (var db = new ProdContext())
{
    ShowCategoriesWithProductsCount_MethodSyntax(db);
    // ShowCategoriesWithProductsCount_QuerySyntax(db);
    Console.ReadKey();
}
```

A screenshot of a Windows console application window. The title bar shows the file path: file:///C:/Users/student/Downloads/ConsoleApplication4/ConsoleApplication4/... The console output displays a list of categories and their product counts: Zestawy (2), Burgery (6), Wrapy (1), Dodatki (5), Napoje (4), and Desery (4). A cursor is visible on the line following the last output.

```
file:///C:/Users/student/Downloads/ConsoleApplication4/ConsoleApplication4/...
Zestawy (2)
Burgery (6)
Wrapy (1)
Dodatki (5)
Napoje (4)
Desery (4)
_
```

## VI. Zadanie domowe

Zmiany w modelu były dokonywane przyrostowo, z wykorzystaniem komend `Add-Migration` oraz `Update-Database`.

Klasa `Product` reprezentująca produkty uległa istotnym zmianom. Zostały dodane pola zawierające informację o kolorze i kolejności przycisków służących do wybierania produktów.

```
public class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; }
    public int UnitsInStock { get; set; }
    public int CategoryID { get; set; }

    [Column(TypeName = "money")]
    public decimal Unitprice { get; set; }

    [MaxLength(32)]
    public string ButtonColor { get; set; }

    public int ButtonOrder { get; set; }

    public Category Category { get; set; }
}
```

Klasa `Order` reprezentująca zamówienia została wzbogacona polami zawierającymi rodzaj zamówienia oraz aktualny status. Dodano metodę `GetOrderNumber` zwracającą trzycyfrowy numer zamówienia, który powinien zostać wyświetlony na ekranie nad stanowiskiem do odbioru zamówienia.

```
public class Order
{
    public int OrderID { get; set; }
    public virtual List<OrderDetail> Details { get; set; }
    public OrderType Type { get; set; }
    public OrderStatus Status { get; set; }

    public string GetOrderNumber()
    {
        return string.Format("{0,3:D3}", this.OrderID % 100);
    }
}
```



Rodzaj zamówienia (na miejscu, na wynos, McDrive) oraz aktualny status (składane, złożone, opłacone, odebrane, anulowane) jest reprezentowany przez enumeracje `OrderType` oraz `OrderStatus`.

```
public enum OrderType
{
    EatHere,
    TakeAway,
    DriveThru,
}

public enum OrderStatus
{
    New,
    Cancelled,
    Submitted,
    Paid,
    Collected
}
```

Przy podejściu Code First, Entity Framework w wersji 6 mapuje enumeratory na `int`, co można zauważyć po otwarciu automatycznie wygenerowanego pliku migracji `AddOrderStatusAndType`:

```
public partial class AddOrderStatusAndType : DbMigration
{
    public override void Up()
    {
        AddColumn("dbo.Orders", "Type", c => c.Int(nullable: false));
        AddColumn("dbo.Orders", "Status", c => c.Int(nullable: false));
    }

    public override void Down()
    {
        // ...
    }
}
```

Zamówiony produkt wraz z liczbą sztuk jest reprezentowany tradycyjnie jako obiekt klasy `OrderDetail`, która zawiera navigation properties `Order` oraz `Product` pozwalające odwołać się odpowiednio do zawierającego daną pozycję zamówienia oraz zamawianego produktu.

```
public class OrderDetail
{
    [Key, Column(Order = 1)]
    public int OrderID { get; set; }
    [Key, Column(Order = 2)]
    public int ProductID { get; set; }

    [Column(TypeName = "money")]
    public decimal Unitprice { get; set; }
    // product price may change over time
}
```

```

public int NumberOfUnits { get; set; }

public Order Order { get; set; }
public Product Product { get; set; }

public override string ToString()
{
    return this.Product.Name + " x" + this.NumberOfUnits;
}
}

```

W definicji tego obiektu chciałem wykorzystać fakt, że Entity Framework wspiera klucze złożone, które należy zdefiniować wykorzystując data annotations `Key` oraz `Column` z parametrem `Order` (kolejność).

Jak później się okazało, wyświetlanie listy zamawianych produktów w kolejności ich dodania do koszyka jest niemożliwe bez dodania kolejnej kolumny. Zrezygnowałem więc z tego rozwiązania i utworzyłem nowy automatycznie inkrementowany klucz główny `OrderDetailID`, wykorzystywany do sortowania listy zamówionych produktów.

```

public class OrderDetail
{
    public int OrderDetailID { get; set; }
    public int OrderID { get; set; }
    public int ProductID { get; set; }
    // ...
}

```

Po wprowadzonych modyfikacjach klasa `ProdContext` jest zdefiniowana następująco:

```

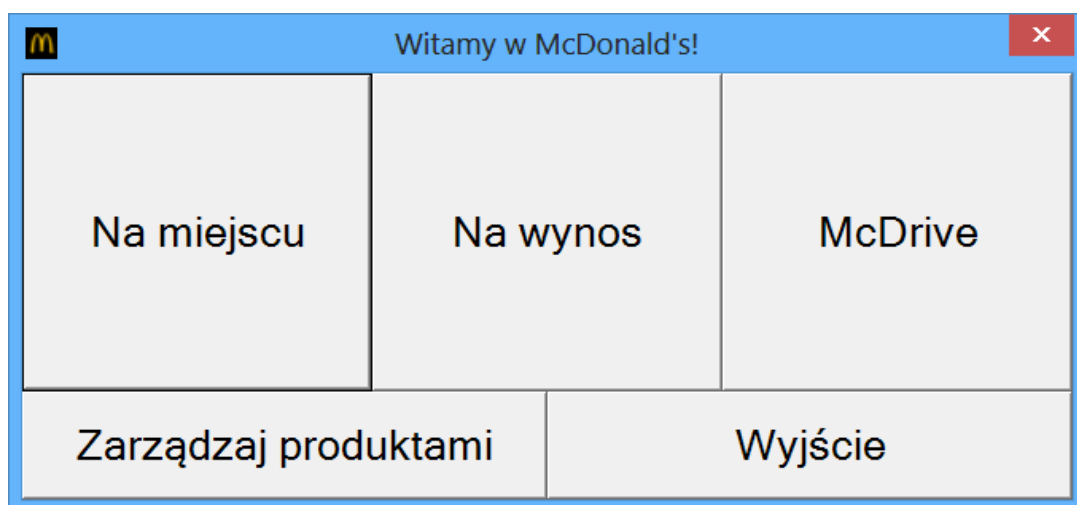
public class ProdContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<OrderDetail> OrderDetails { get; set; }
    public DbSet<Customer> Customers { get; set; }
}

```

W trakcie realizowania zadania środowisko programistyczne odmówiło współpracy, co okazywało kompletnym zawieszaniem się już na etapie ładowania projektu. Zaproponowane przez jednego z użytkowników platformy Stack Overflow odinstalowanie menedżera pakietów NuGet okazało się celnym trafem, ale ponowna instalacja przywracała opisany problem. Finalnym rozwiązaniem było przekopiowanie nowych fragmentów kodu źródłowego do wersji zapasowej projektu po ukończeniu instrukcji do laboratorium.

Entity Framework dość mocno buntował się również wtedy, gdy zdarzyło mi się namieszać przy modyfikacji modelu i migracjach. Tutaj rozwiązaniem okazało się wycofywanie migracji przy użyciu komendy `Update-Database` wraz z argumentem `-TargetMigration`.

Aplikacja przeznaczona jest dla restauracji typu fast-food. Została zaprojektowana dla stosowanych obecnie kilkunastocalowych ekranów dotykowych. Po uruchomieniu aplikacji wyświetlony zostaje ekran powitalny.



```
private void NewOrder(OrderType type)
{
    var order = new Order { Type = type };
    db.Orders.Add(order);
    db.SaveChanges();
    var form = new OrderForm(this.db, order);
    form.ShowDialog();
}

private void NewEatHereOrderButton_Click(object sender, EventArgs e)
{
    this.NewOrder(OrderType.EatHere);
}

private void NewTakeAwayOrderButton_Click(object sender, EventArgs e)
{
    this.NewOrder(OrderType.TakeAway);
}

private void NewDriveThruOrder_Click(object sender, EventArgs e)
{
    this.NewOrder(OrderType.DriveThru);
}
```



Po kliknięciu przycisku *Na miejscu*, *Na wynos* albo *McDrive*, wyświetlone zostaje okno składania zamówienia. Aplikacja wspiera zarówno nazwy kolorów stosowane w językach HTML oraz CSS (np. `Gold` ), a także zapis heksadecymalny ze znakiem `#` na początku (np. `#123456` ).



Zamówienie

	Liczba	Nazwa	Cena	Suma	
▶	2 szt.	Cheeseburger	4,50 zł	9,00 zł	Złóż zamówienie
	1 szt.	Frytki zakręcone	6,90 zł	6,90 zł	
	1 szt.	Ice Tea	5,90 zł	5,90 zł	
				Σ=21,80 zł	

Anuluj zamówienie

1	2	3	4	5	6	7	8
Cheeseburger (4,50 zł)	Hamburger (4,00 zł)	Kurczakburger (5,00 zł)	Big Mac (9,60 zł)	McRoyal (10,50 zł)	McChicken (8,90 zł)	Filet-O-Fish (8,70 zł)	Happy Meal (10,50 zł)
2 for U (6,00 zł)	McWrap (11,70 zł)	Frytki (5,90 zł)	Frytki zakręcone (6,90 zł)	Chrupserki (7,90 zł)	McNuggets (7,90 zł)	Coca-Cola (5,90 zł)	Fanta (5,90 zł)
Sprite (5,90 zł)	Ice Tea (5,90 zł)	Salatka (5,90 zł)	Lody z polewą (6,20 zł)	Shake (6,40 zł)	McFlurry (7,20 zł)	Ciastko jabłkowe (4,50 zł)	Ciastko sezonowe (4,50 zł)

Oto fragment kodu źródłowego odpowiedzialny za dynamiczne tworzenie przycisków na podstawie zawartości bazy danych:

```
int x = 0, y = 1;
var products = this.db.Products.Include("Category").OrderBy(p => p.ButtonOrder);
foreach (var product in products)
{
    Button button = new Button();
    EventHandler handler = (_, __) =>
    {
        this.AddProduct(product, this.currentNumberOfUnits);
        this.currentNumberOfUnits = 1;
        this.RefreshDetailsList();
    };
    button.Click += handler;
    button.Text = product.Name + "\n(" + Util.FormatPrice(product.Unitprice) + ")";
    button.Font = new Font(button.Font.FontFamily, 12);
    button.BackColor = System.Drawing.ColorTranslator.FromHtml(product.ButtonColor ?? "red");
    button.ForeColor = (button.BackColor.GetBrightness() > 0.5) ? Color.Black : Color.White;
    button.Top = this.DetailsDataGridView.Height + y * this.buttonHeight;
    button.Left = x * this.buttonWidth;
    button.Width = this.buttonWidth;
    button.Height = this.buttonHeight;
    this.Controls.Add(button);

    ++x;
    if (x >= this.numberOfButtonColumns)
    {
        x = 0;
        ++y;
    }
}
```

Kliknięcie przycisku spowoduje dodanie do zamówienia 1 szt. wskazanego produktu. Aby dodać kilka sztuk danego produktu do koszyka, należy:

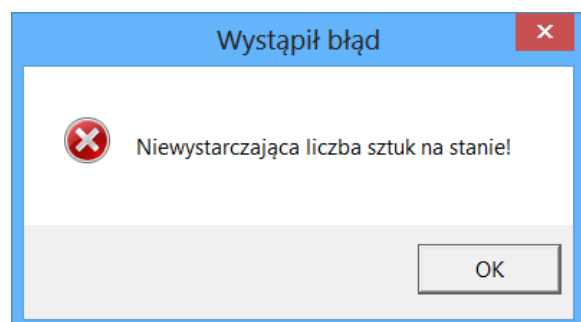
- najpierw wybrać liczbę sztuk, klikając jeden z przycisków 1-8
- potem wybrać produkt, klikając na odpowiedni przycisk.

Aktualna liczba sztuk jest automatycznie zmieniana na 1 szt. po dodaniu produktu do zamówienia. Kolejne dodanie produktu, który został już zamówiony, spowoduje zwiększenie licznika w odpowiednim wierszu listy zamówień (również w tabeli `OrderDetails` ).

```
private void AddProduct(Product product, int units) {
    if (product.UnitsInStock < units)
    {
        MessageBox.Show(
            "Niewystarczająca liczba sztuk na stanie!",
            "Wystąpił błąd", MessageBoxButtons.OK, MessageBoxIcon.Error
        );
        return;
    }
    product.UnitsInStock -= units;

    try
    {
        var detail = this.db.OrderDetails.Where(
            d => d.OrderID == this.Order.OrderID &&
            d.ProductID == product.ProductID
        ).First();
        detail.NumberOfUnits += units;
    }
    catch (InvalidOperationException)
    {
        var newDetail = new OrderDetail
        {
            OrderID = this.Order.OrderID,
            ProductID = product.ProductID,
            Unitprice = product.Unitprice,
            NumberOfUnits = units
        };
        db.OrderDetails.Add(newDetail);
    }
    finally
    {
        this.db.SaveChanges();
    }
}
```

Próba zamówienia większej liczby produktu niż dostępna (pole `UnitsInStock` ) zakończy się niepowodzeniem. Użytkownikowi zostanie wyświetlona informacja o wystąpieniu błędu.

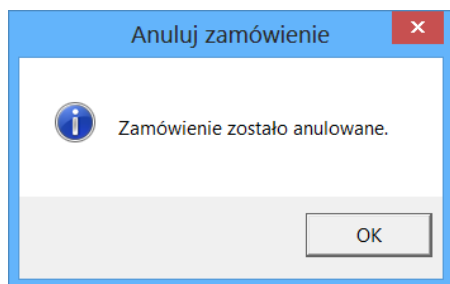
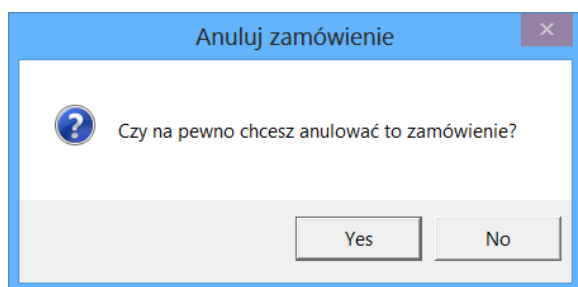


Aby usunąć produkt z listy zamawianych pozycji, należy wskazać odpowiedni wiersz albo komórkę, a następnie kliknąć przycisk *Usuń pozycję*. Usunięte produkty trafiają z powrotem do puli dostępnych produktów (pole `UnitsInStock` ).

```
OrderDetail detail =
    (from d in db.OrderDetails
     where d.OrderID == this.Order.OrderID && d.ProductID == productId
     select d).FirstOrDefault();
if (detail != null)
{
    detail.Product.UnitsInStock += detail.NumberOfUnits;
    this.db.OrderDetails.Remove(detail);
}

this.db.SaveChanges();
this.RefreshDetailsList();
```

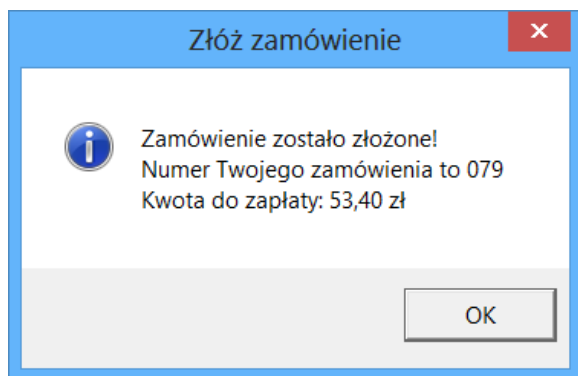
Aktualnie składane zamówienie można jeszcze anulować. Aby nie doszło do przypadkowego anulowania zamówienia, należy potwierdzić chęć wykonania tej operacji. Przy anulowaniu zamówienia wszystkie produkty wracają do puli dostępnych produktów.



```
this.Order.Status = OrderStatus.Cancelled;
foreach (var detail in this.Order.Details)
{
    detail.Product.UnitsInStock += detail.NumberOfUnits;
}
this.db.SaveChanges();
this.Close();
MessageBox.Show(
    "Zamówienie zostało anulowane.",
    "Anuluj zamówienie", MessageBoxButtons.OK, MessageBoxIcon.Information);
```



Po dodaniu wszystkich produktów do koszyka należy złożyć zamówienie, klikając na zielony przycisk. Wówczas zamówienie zmieni stan na przyjęte do realizacji. Za zamówione burgery i frytki trzeba przecież jeszcze słono zapłacić. Użytkownikowi zostanie wyświetlona informacja zawierająca należną kwotę oraz trzycyfrowy numer zamówienia.



```
this.Order.Status = OrderStatus.Submitted;
this.db.SaveChanges();
this.Close();
MessageBox.Show(
    "Zamówienie zostało złożone!\n" +
    "Numer Twojego zamówienia to " + this.Order.GetOrderNumber() + "\n" +
    "Kwota do zapłaty: " + Util.FormatPrice(this.TotalPrice),
    "Złóż zamówienie", MessageBoxButtons.OK, MessageBoxIcon.Information);
```