Moskała Przemysław

EntityFramework, LINQ2Entites

Laboratorium 2 – sprawozdanie

1.Model bazy danych

Przy użyciu Code First została wygenerowana baza danych.

Kategorie:

```
class Category
{
    private readonly ObservableListSource<Product> _products =
                    new ObservableListSource<Product>();
    public int CategoryId { get; set; }
    public string Name { get; set; }
    public virtual ObservableListSource<Product> Products { get { return _products; } }
}
```

Produkty:

```
class Product
{
    [Column(TypeName="Money")]
    public int ProductId { get; set; }
    public string Name { get; set; }
    public int UnitsInStock { get; set; }
    public decimal UnitPrice { get; set; }
    public int CategoryId { get; set; }
}
```

Podmioty:

```
class Customer
{
    private readonly ObservableListSource<Order> _orders =
                    new ObservableListSource<Order>();
    [Key]
    public string CompanyName { get; set; }
    public string Descritpion { get; set; }
    public string EMail { get; set; }
    public string Adress { get; set; }
    public virtual ObservableListSource<Order> Orders { get { return _orders; } }
}
```

Zamówienia:

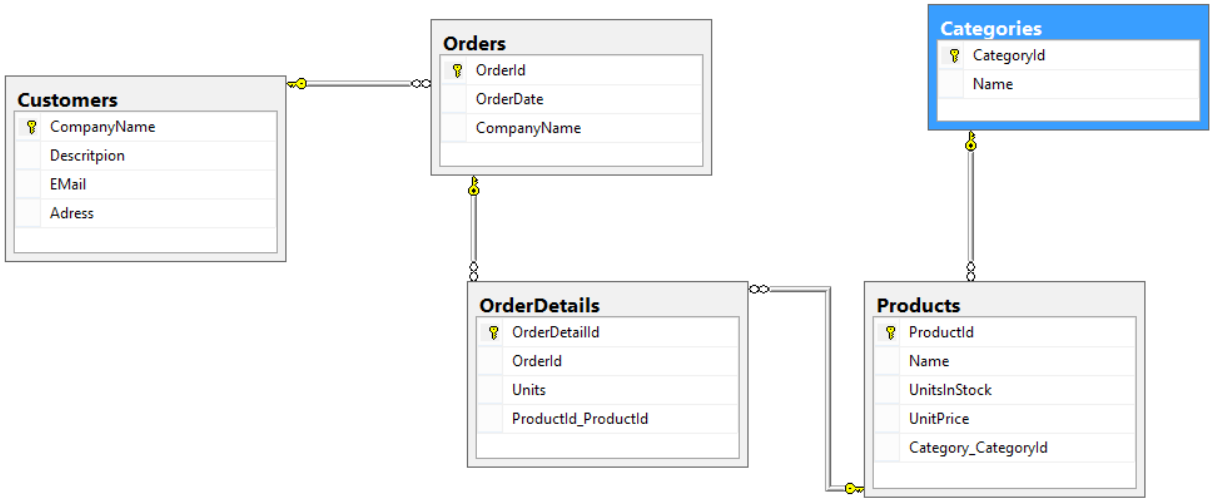
```
class Order

{
    private ObservableListSource<OrderDetails> _orderDetails =
                    new ObservableListSource<OrderDetails>();
    public int OrderId { get; set; }
    public DateTime OrderDate { get; set; }
    public string CompanyName { get; set; }
    public virtual ObservableListSource<OrderDetails> OrdersDetails { get { return _orderDetails;
} }
}

class OrderDetails
```

```
{
    public int OrderId { get; set; }
    [Key]
    public int OrderDetailId { get; set; }
    public Product ProductId { get; set; }
    public int Units { get; set; }
}
```

Dzięki navigation properties w bazie danych zostały utworzone nienullowane klucze obce.



Ilustracja 1: Schemat bazy danych aplikacji

2. Opis aplikacji

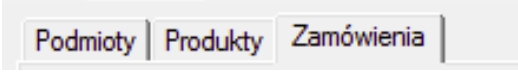
Aplikacja Produkty pozwala:

- przeglądać/dodać/edytować dane klientów
- przeglądać/dodać/edytować dane produktu
- przeglądać/dodać/edytować zamówienia
- generować faktury dla zamówień

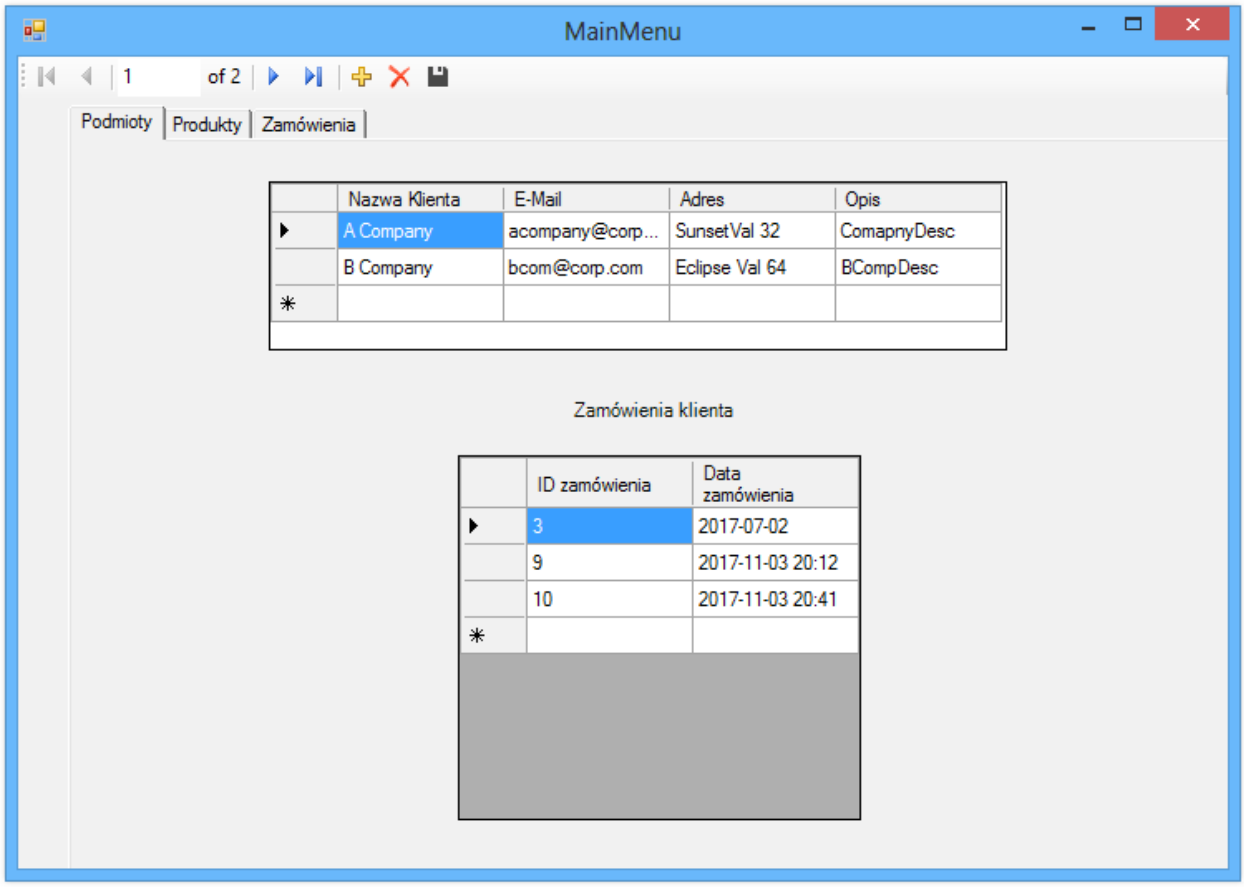
Aplikacja została zrealizowana jako WindowsForm Project. Każda kontroler zbierający dane z formatki wykonuje metody odpowiednich serwisów, odpowiedzialnych za dostarczenie logiki biznesowej dla wyróżnionych obszarów.

- OrderService – obszar zamówień
- CustomerService – obszar podmiotów
- ProductService – obszar produktu

Każdy obszar posiada swoją zakładkę w formatce.



3. Obszar podmiotów



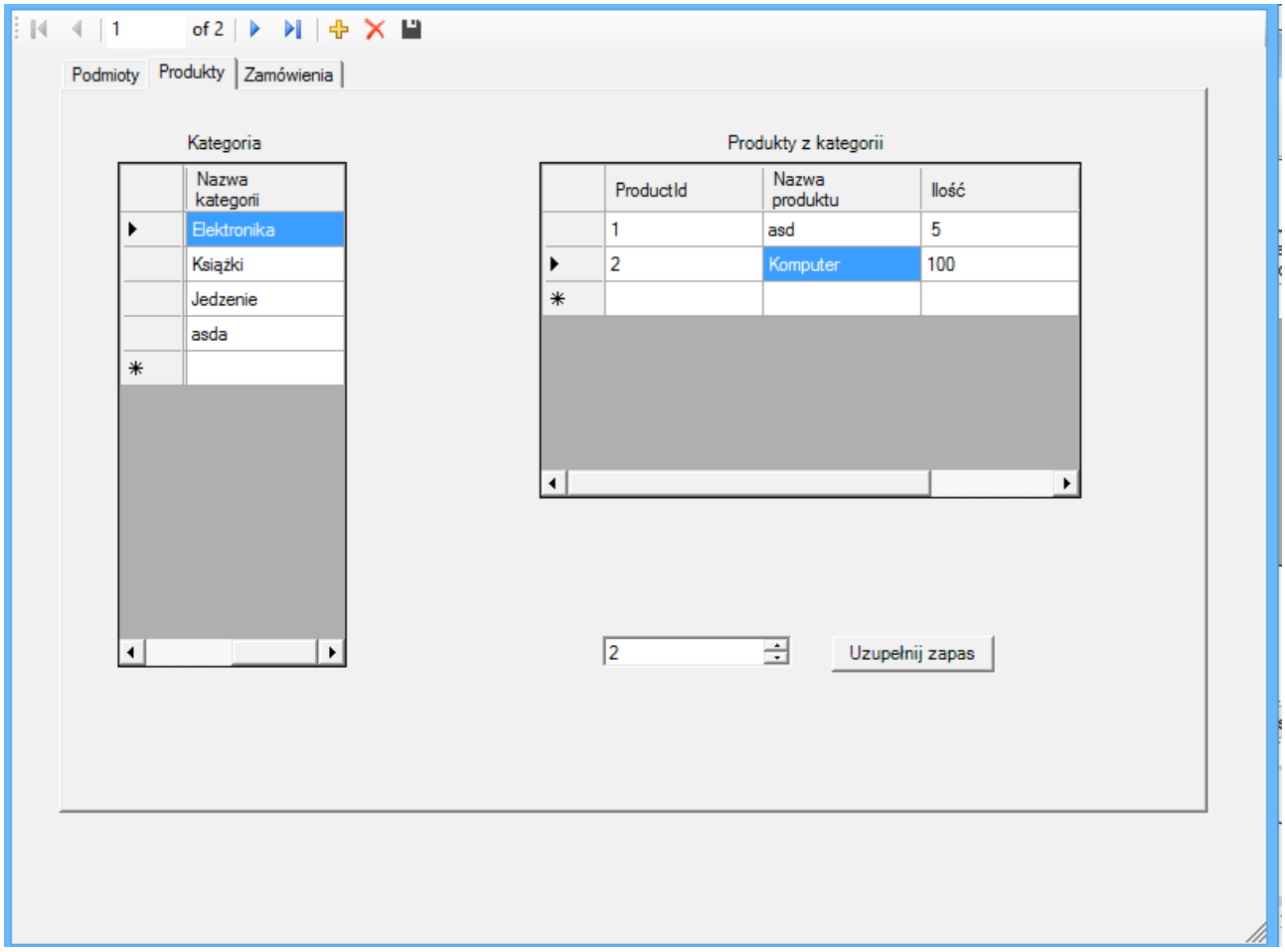
Zakładka1 - obszar podmiotów

Możemy w nim dodać lub edytować podmiot.

Po kliknięciu na wiesz w górnej tabeli do tabeli dolnej zostają załadowane zamówienia wybranego klienta. Kolumna z zamówieniami spełnia założenia Lazy Fetching – dane zamówień dla wybranego podmiotu zostaną pobrane z bazy danych tylko wtedy, kiedy użytkownik kliknie na konkretny rekord.

Użytkownik może dodawać wiersze do górnej tabeli, a następnie za pomocą ikonki zapisu wywołać metodę *SaveChanges()* na kontekście.

4.Obszar produktów



Zakładka 2 - obszar produktów

W tej zakładce użytkownik może

- Przeglądać kategorię i produkty przypisane do tej kategorii
- Dodać produkt do kategorii
- Edytować nazwę produktu
- Uzupełnić zapas produktu

Do interakcji z bazą danych została zbudowana klasa *ProductService*, która udostępnia biznesowe operację nad zdefiniowanym wcześniej modelem. Zostały w niej użyte mechanizmy L2E MethodSyntax i QuerySyntax.

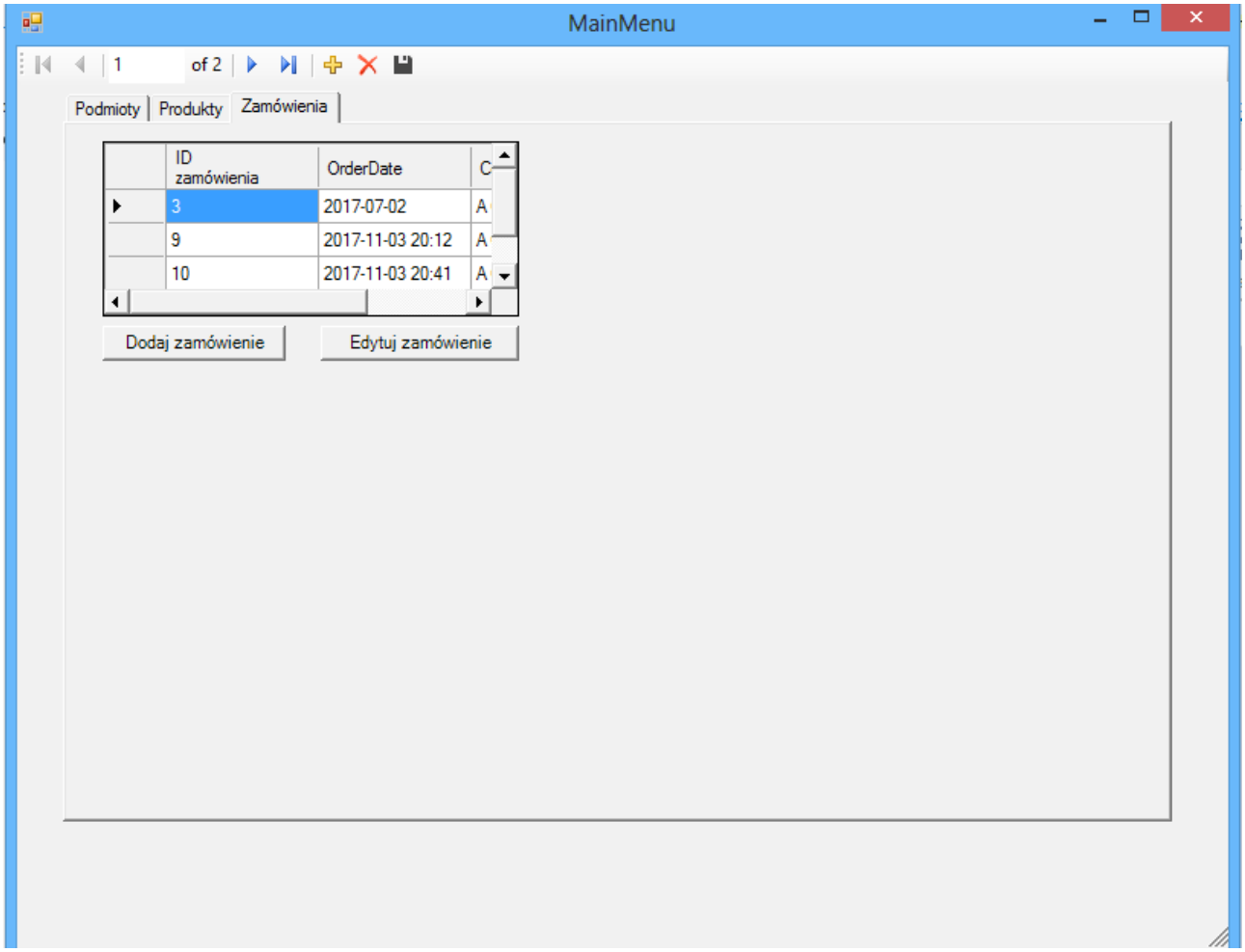
```
class ProductService
{
    ProductContext context;
    public static Product findProductById(ProductContext context, int productId)
    {
        return context.Products.Where(p => p.ProductId == productId).First();
    }

    public static List<Product> findProductsByCategoryId(ProductContext context, int categoryId)
    {
        return (from p in context.Products select p).ToList();
    }
    public static Object getCategoriesWithProducts(ProductContext context)
    {
```

```
        return context.Categories.Join(context.Products,
                                c => c.CategoryId,
                                p => p.CategoryId,
                                (c, p) => new { Name = c.Name, Products = p }).Select(x =>
x).ToList();
    }
    public static Object getCategoriesWithProductsQuerySyntax(ProductContext context)
    {
        return (from c in context.Categories
                join p in context.Products
                on c.CategoryId equals p.CategoryId
                select c).ToList();
    }
    public static int supplyNewProducts(ProductContext context, int productId, int units)
    {
       var product = context.Products.
                        Where(p => p.ProductId == productId).First();
       product.UnitsInStock += units;
       context.SaveChanges();
       return product.UnitsInStock;

    }
}
```

5. Obszar zamówień



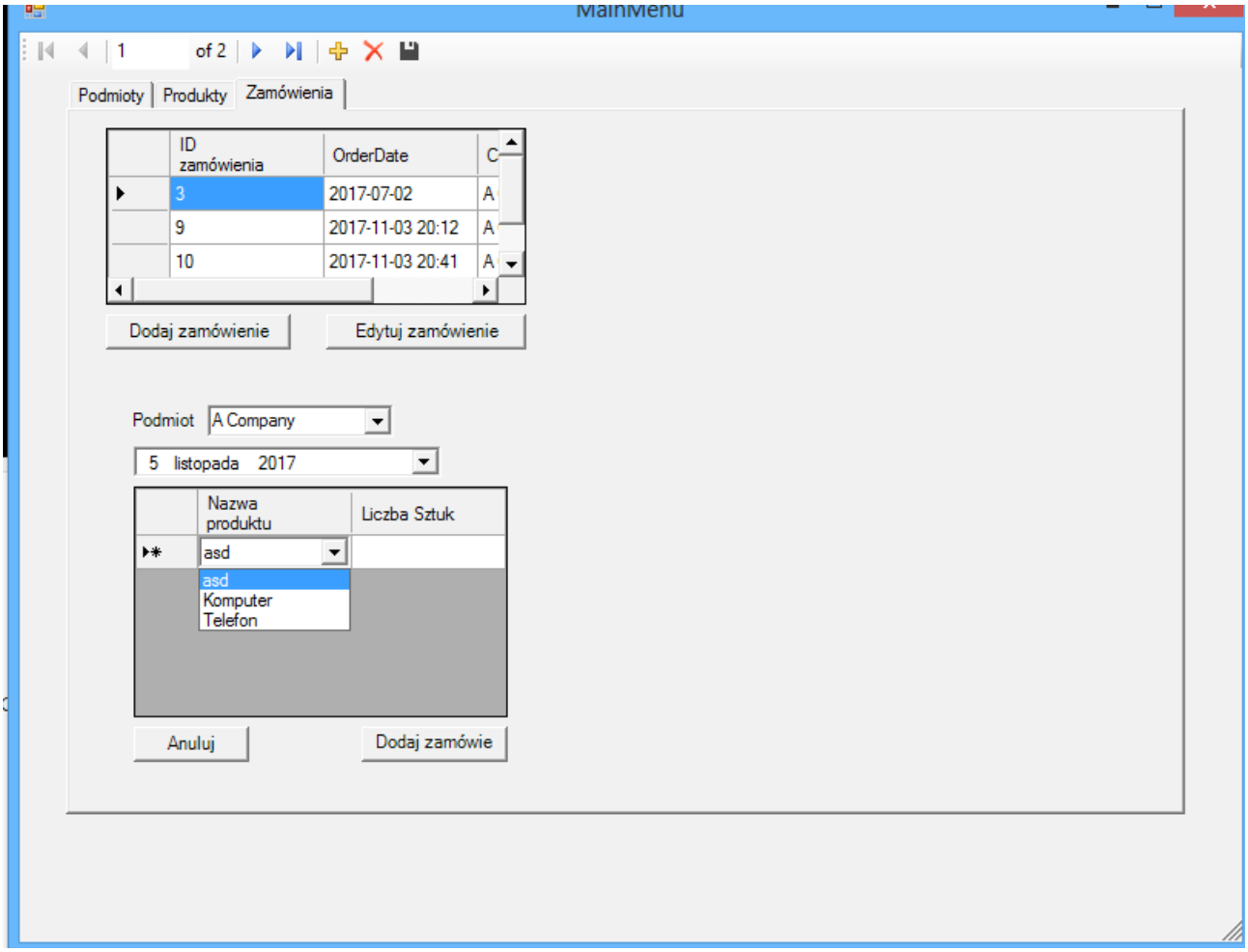
Zakładka 3 - obszar zamówień

W tej zakładce udostępnione zostały funkcjonalności

- przeglądania zamówień
- dodawania zamówień

- edytowania zamówień
- filtrowania zamówień po nazwie podmiotu lub dacie
- generowania faktury

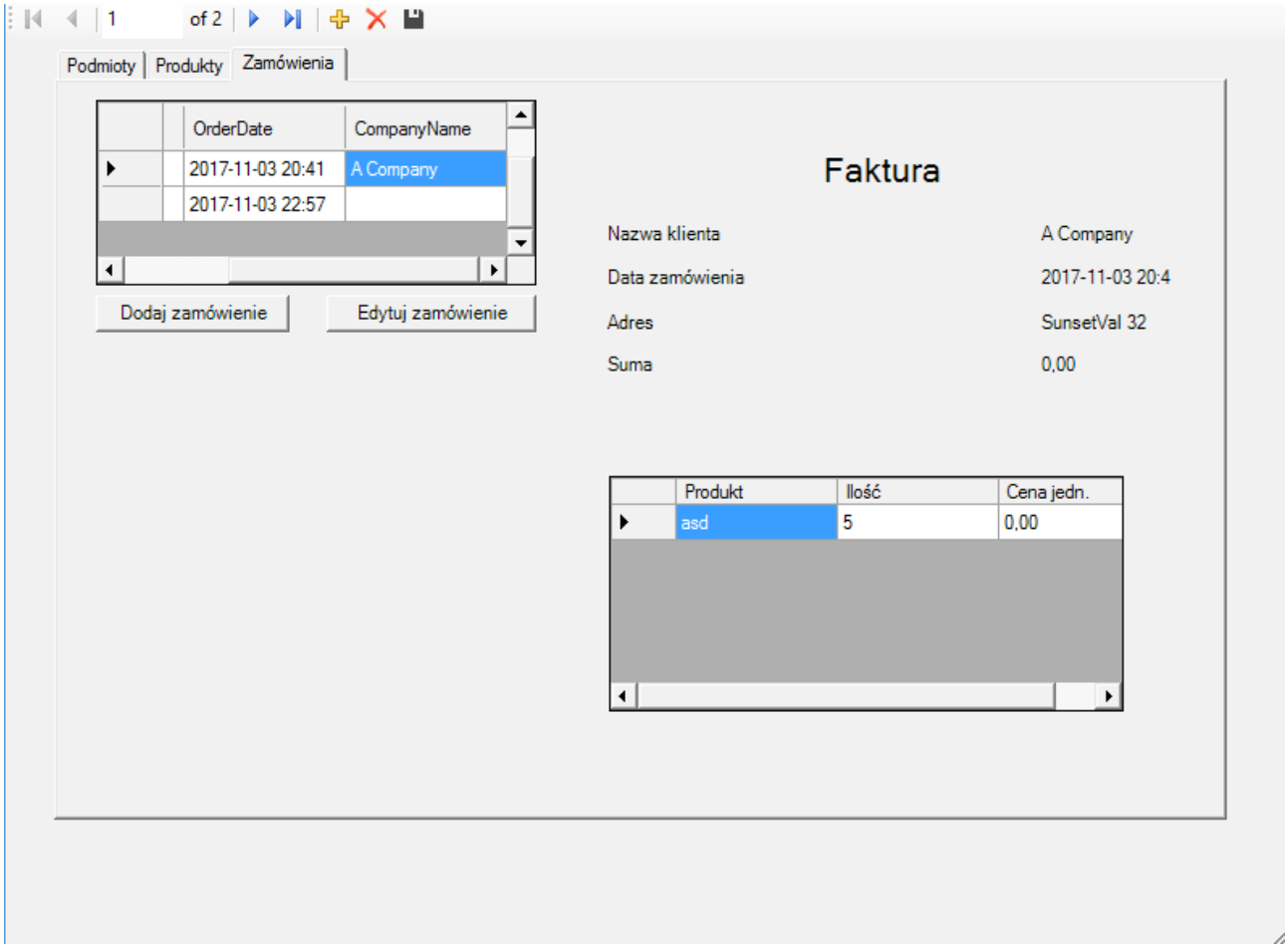
Po kliknięciu *Dodaj zamówienie* wyświetla się formatka dodania zamówienia:



Zakładka 3 – obszar zamówień – dodawanie zamówienia

W tej formatce użytkownik wybiera z comboboxa nazwę podmiotu i nazwe produktów, a następnie definiuje ilość zamawianych sztuk. Kliknięcie *Dodaj zamówienie* powoduje uruchomienie metody walidującej zamówienie i ewentualne dodanie zamówienia bądź wyświetlenie komunikatu z błędem(np z powodu braku towaru).

Użytkownikowi została udostępniona funkcjonalność generowania faktury dla wybranego zamówienia z tabelki górnej.



Zakładka 3 - generowanie faktury

Logika biznesowa zakładki 3 jest oparta o klasę *OrderService*

```
class OrderService
{
    public static List<Order> findOrderByCustomerId(string companyName, ProductContext context)
    {
        return context.Orders.Where(o => o.CompanyName == companyName).Select( o => o).ToList();
    }
    public static List<OrderDetails> findOrderDetailsByOrderId(int orderId, ProductContext context)
    {
        return (from od in context.OrderDetails where od.OrderId == orderId select od).ToList();
    }
    public static List<OrderBillDto> findOrderBillDtosByOrder(Order order)
    {
        var a = new List<OrderBillDto>();
        foreach (var orderDetail in order.OrdersDetails)
        {
            a.Add(new OrderBillDto { ProductName = orderDetail.ProductId.Name, Unit =
orderDetail.Units, UnitPrice = orderDetail.ProductId.UnitPrice });
        }
        return a;
    }
    public static void addOrder(ProductContext context, Customer customter, DateTime orderDate,
OrderDetails orderDetails)
    {
        context.Orders.Add(new Order() {CompanyName = customter.CompanyName});
        context.SaveChanges();
    }
    internal static decimal findSumForOrderBill(List<OrderBillDto> orBill)
    {
        decimal sum = 0;
        foreach (var bille in orBill)
```

```
        {
            sum += bille.Unit * bille.UnitPrice;
        }
        return sum;
    }
    internal static List<string> validOrder(Order order, ProductContext context)
    {
        List<string> errors = new List<string>();
        foreach (OrderDetails od in order.OrdersDetails)
        {
            var left = context.Products.Where(p => p.ProductId == od.ProductId.ProductId).Select(p
=> p).First();
            if (left.UnitsInStock < od.Units)
            errors.Add(string.Format("{0} zostało sztuk : {1}", left.Name, left.UnitsInStock));
        }
        return errors;
    }
}
```

Kod źródłowy projektu

agh-db-ef