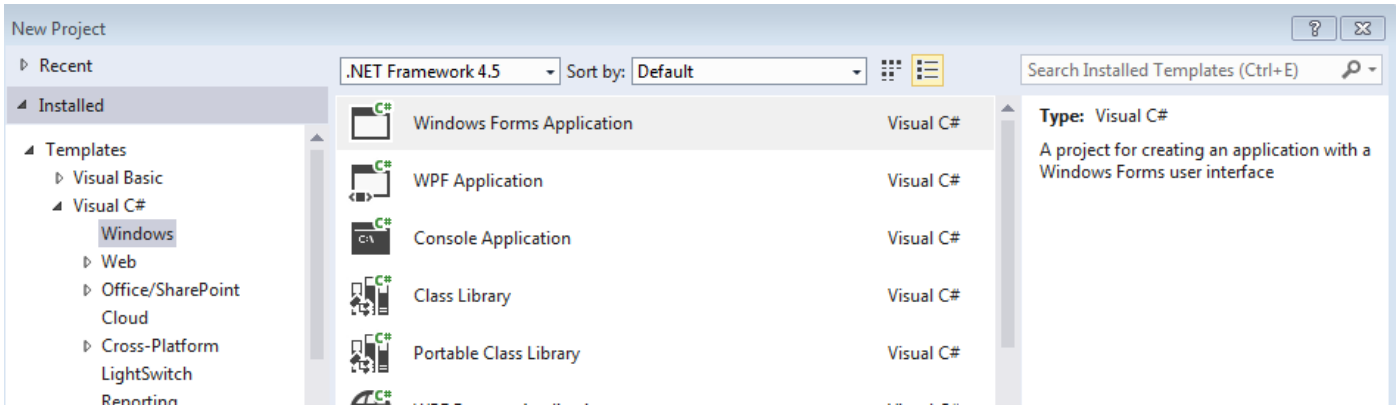
Упражнения: Winforms като потребителски интерфейс

Създаване на просто приложение

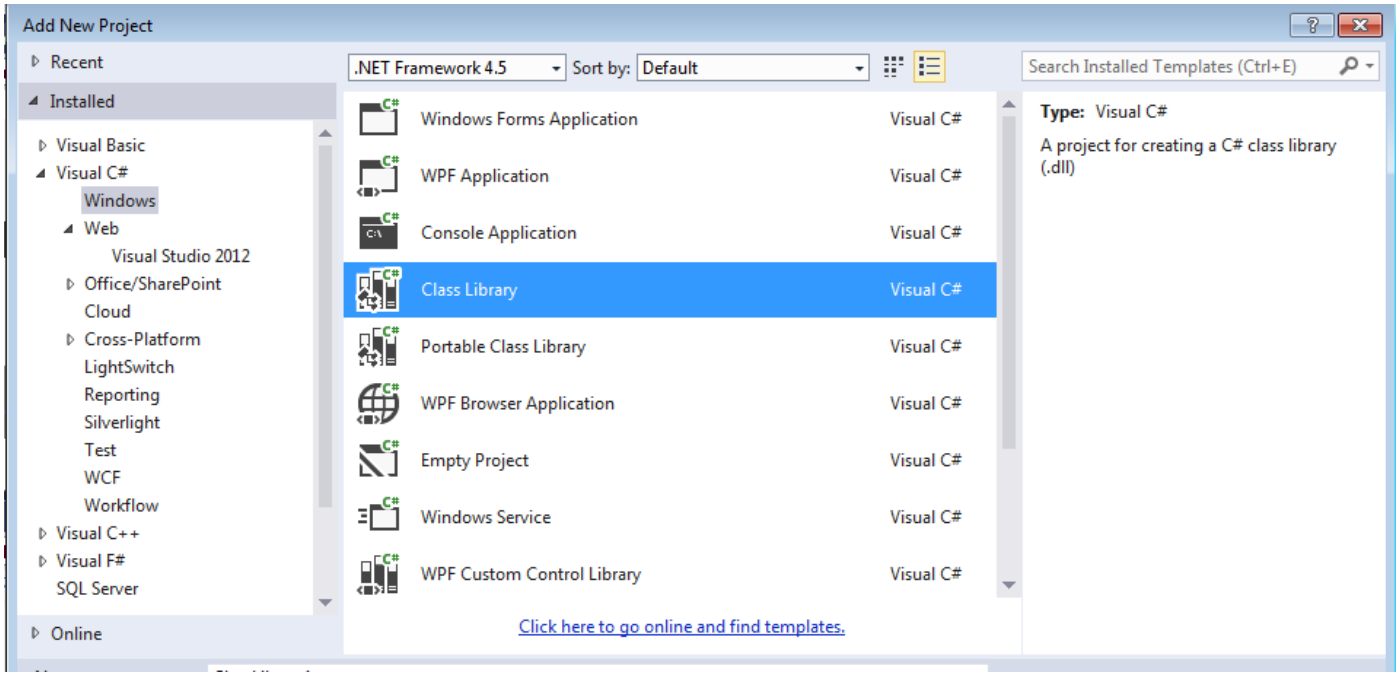
В рамките на това упражнение ще направим трислойно приложение, като слоевете за данни и услуги ще са аналогични с досега направените, но за презентационен слой ще използваме Winforms.

1. Структура на проекта

Започнете със създаване на **Windows Forms Application** проект **ProductApp**.



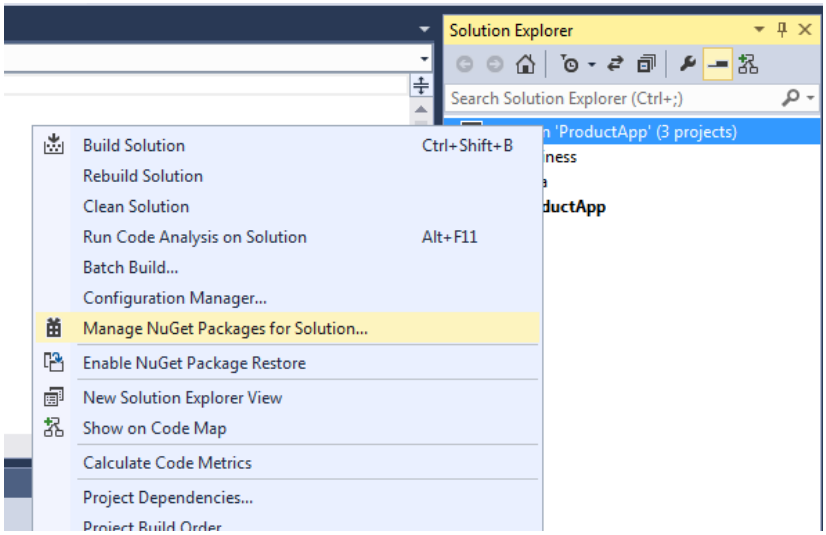
Сега създайте нов проект, като тук изберете **Class Library.** Това ще бъде проекта за слоя данни **Data.**



По сходен начин ще създадем и проект **Business.** Използваме **Class Library**, тъй като във всеки от тези проекти създаваме класове, които да могат да бъдат достъпвани от други проекти.

2. Добавяне на EntityFramework и референции

След като сме създали всички проекти е време да добавим **EntityFramework.** Цъкнете с десен бутон върху solution-а и изберете **Manage NuGet Packages for Solution** и инсталирайте **EntityFramework**. Изберете инсталация за **всичките** проекти в solution-а.

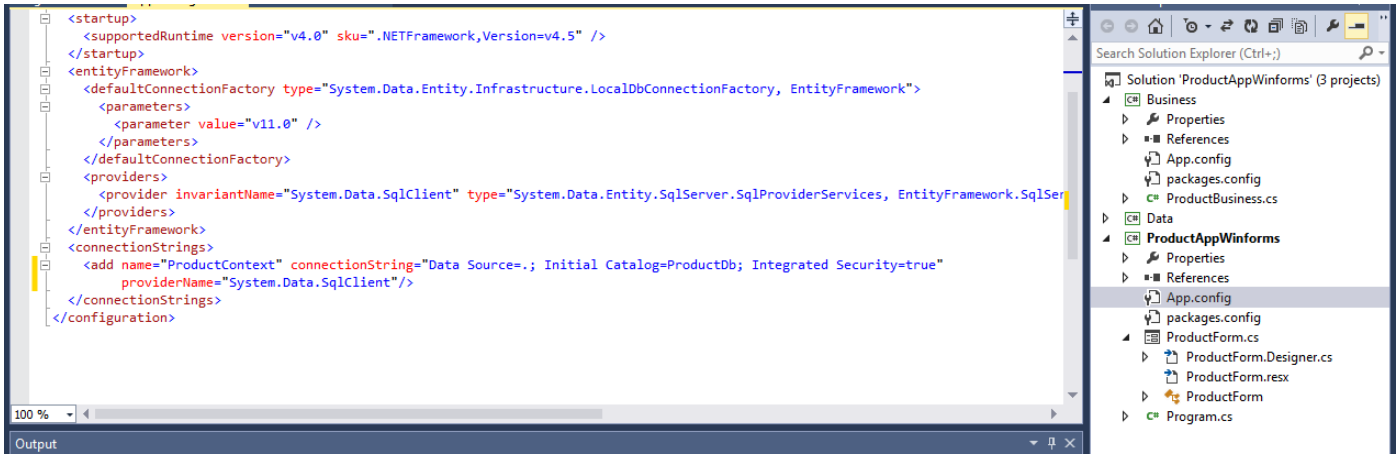


Освен това добавете следните референции (десен бутон върху проекта -> Add -> Reference) за всеки от проектите:

- На проект **Business** добавете референция към **Data**
- На проект **ProductApp** /Winforms проекта/ добавете референция към **Business** и **Data**

3. База данни

Тук ще използваме вече съществуващата от предните проекти **ProductDb**. Отново единственият ни ангажимент е да добавим низа за връзка. В **App.config** на проект **ProductApp** добавете **<connectionStrings>** тага и в него добавете **<add>** таг с низа за връзката преди **</configuration>**:



4. Слой за данни

Слоят за данни тук е напълно аналогичен с този в предното упражнение.
Тук той се състои от папка с **модел** и **контекст**.

Моделът тук е клас **Product.cs,** находящ се в **Model** папката, като в него описваме единствено свойствата му, които всъщност съответстват на колоните от таблицата.

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public int Stock { get; set; }
}
```

В самата папка **Data** се намира и **ProductContext** класа. Той трябва да наследява **DbContext**. Тук ще се наложи да добавите и **using** директива, понеже класа **DbContext** е част от **EntityFramework**.
Самият клас ще съдържа конструктор, в който ще има обръщение към конструктора на базовия клас, където за параметър се подава името (**name** атрибутът) от низа за връзка, който добавихме в **App.config** по-рано.

Освен това в конструктора ще имаме и свойство, което ще е от **DbSet<Product>.** Кодът е както следва:

```
public class ProductContext : DbContext
{
    public ProductContext()
        : base("name=ProductContext")
    {

    }
    public DbSet<Product> Products { get; set; }
}
```

С това сме готови с нашия слой за данни.

5. Бизнес слой

Слоят за бизнес логиката тук е напълно аналогичен с този в предното упражнение.

Тук ще имаме поле от тип **ProductContext**, което ще използваме в методите.

Методите като логика работят по абсолютно сходен начин с предното упражнение. Класът изглежда по следния начин:

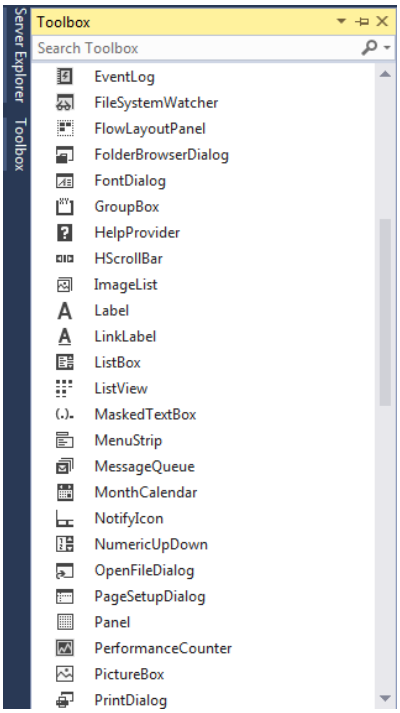
```
class ProductBusiness
{
    private ProductContext productContext;

    public List<Product> GetAll()...

    public Product Get(int id)...

    public void Add(Product product)...

    public void Update(Product product)...

    public void Delete(int id)...
}
```

А методите са съответно:

```
public List<Product> GetAll()
{
    using (productContext = new ProductContext())
    {
        return productContext.Products.ToList();
    }
}

public Product Get(int id)
{
    using (productContext = new ProductContext())
    {
        return productContext.Products.Find(id);

    }
}

public void Add(Product product)
{
    using (productContext = new ProductContext())
    {
        productContext.Products.Add(product);
        productContext.SaveChanges();

    }
}

public void Update(Product product)
{
    using (productContext = new ProductContext())
    {

        var item = productContext.Products.Find(product.Id);
        if (item != null)
        {
            productContext.Entry(item).CurrentValues.SetValues(product);
            productContext.SaveChanges();
        }

    }
}
```

```
public void Delete(int id)
{
    using (productContext = new ProductContext())
    {
        var product = productContext.Products.Find(id);
        if (product != null)
        {
            productContext.Products.Remove(product);
            productContext.SaveChanges();
        }
    }
}
```

6. Презентационен слой

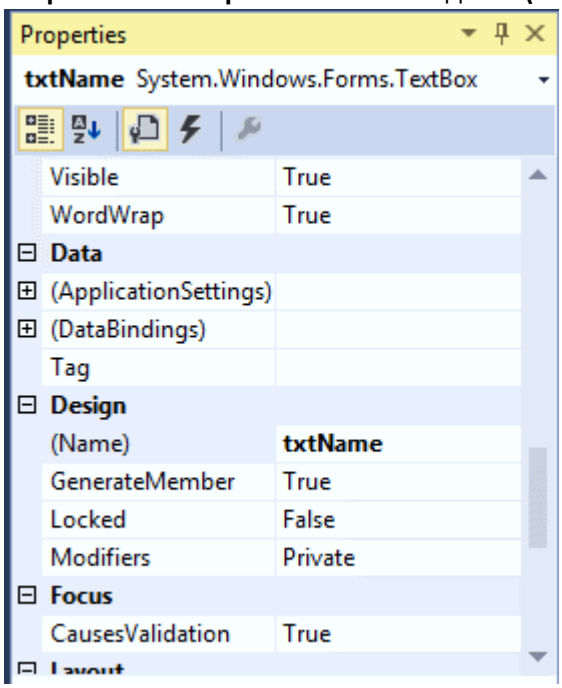


Winforms работи чрез компоненти в графичния интерфейс. За целите на нашите CRUD операции ние ще имаме нужда от няколко етикета (Label), текстови кутии (TextBox), бутони (Button) и един DataGridView.

Добавянето на компоненти се случва чрез **Toolbox** лентата (**View->Toolbox**, ако не я виждате).

Сега нека да добавим 3 **TextBox**-а. Това става чрез завличането на **TextBox** компонента върху самата форма.

След поставянето на първия компонент е добре да зададем някои свойства. Цъкнете с десен бутон върху новодобаваевния компонент и изберете **Properties**. От **Properties** лентата задайте **(Name)** свойството на **txtName**:

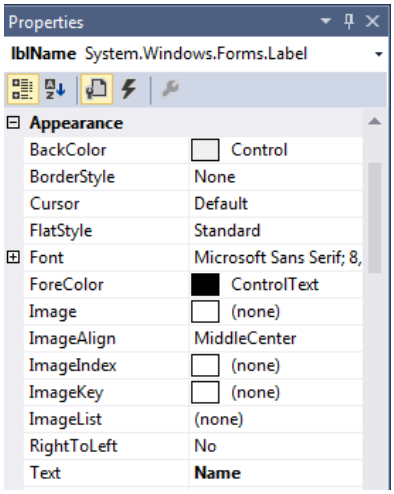


Това свойство е от изключителна важност, тъй като това ще е името на променливата, чрез която ще достъпваме в кода този **TextBox**. Задавайте имена според конвенциите, така както бихте кръстили и променливата си.

По аналогичен начин създайте и именувайте и другите 2 текстови кутийки: **txtPrice** и **txtStock.**

Ако сега дадете вашето приложение на човек, който го вижда за първи път, той ще има един доста основен проблем – няма да знае коя кутийка за какво е. Именно затова съществуват етикетите (Label) – това е текст, който се поставя като пояснителен надпис. Нашето приложение се нуждае от три такива надписа – за

името, цената и количеството. Нека да добавим първият етикет. Задайте неговото име на **lblName**. След тази стъпка може би сте изненадани, че етикетчето все още изписва **Label1**? Това е така понеже има отделно свойство **Text**, което контролира какво вижда потребителя. Редактирайте това свойство, така че на етикета да пише "**Name**":



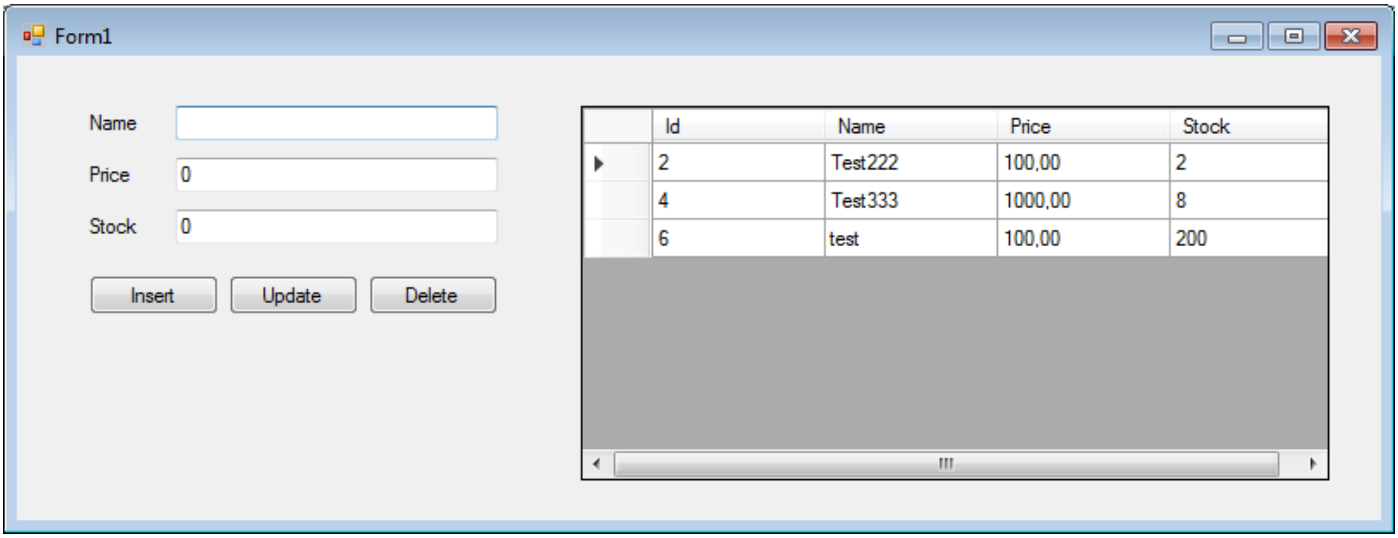
По подобен начин задайте и етикетчета **lblPrice**, с текст „**Price**", **lblStock** с текст „**Stock**".

Сега е ред на бутоните. Трябва да си направим няколко бутона:

- Insert, с име **btnInsert** – при натискането на този бутон информацията от текстовите кутийки ще се взема и изпраща към бизнес логиката, която от своя страна пък ще я изпрати към слоя за данни, което ще доведе до запазването на въведената информация в базата данни.
- Update, с име **btnUpdate –** при натискането на този бутон ще се проверява дали и кой ред от табличката с данни е избран. Ако е избран ред, то информацията от него ще се прехвърли в кутийките, а те ще бъдат готови за редакция. В този момент на мястото на бутон Update, трябва да се появи бутон Save, който да съхрани информацията при натискането си.
- Save, с име **btnSave,** при създаването на този бутон задайте свойството му **Visible** на **false –** при натискането на този бутон ще се запазва информацията от текстовите кутийки и ще се подава към бизнес логиката, а тя от своя страна ще води до актуализирането на обекта в базата данни. Когато това приключи бутонът Save трябва да изчезне, а на негово място отново да се появи Update.
- Delete, с име btnDelete – при натискането на този бутон ще се проверява дали и кой ред от табличката с данни е избран. Ако е избран ред, то това е продуктът, който изтриваме.

И накрая, но не по значение трябва да добавим компонент **DataGridView.** При добавянето на този компонент ще бъдете попитани за източник на данни – на този етап не посочвайте нищо и просто натиснете **Esc**, за да изчезне диалога.

Един примерен начин, по който да изглежда нашето приложение е:



Сега нека да преминем към писането на код.

Цъкнете два пъти върху прозореца на формата. Visual Studio ще ви покаже редактор за код, а стова и генериран метод вътре в него:

```
public partial class ProductForm : Form
{
    public ProductForm()
    {
        InitializeComponent();
    }

    private void ProductForm_Load(object sender, EventArgs e)
    {

    }
```

В рамките на този метод **Form1_Load(…)** ще извикаме два метода, който ще напишем допълнително – **UpdateGrid()** и **ClearTextBoxes().** Тези методи ще бъдат използвани и от други методи малко по-късно.

Метод **UpdateGrid()** има за цел да обновява данните в нашата табличка. Както и да не допуска редакция през самата табличка, задавайки я като **ReadOnly**. Другото важно свойство за тази табличка е начинът на селектиране – ще направим така че да може да се избират само цели редове (по подразбиране може да се избира клетка). Кодът е както следва:

```
private void UpdateGrid()
{
    dataGridView1.DataSource = productBusiness.GetAll();
    dataGridView1.ReadOnly = true;
    dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
}
```

Реализацията на **ClearTextBoxes()** метода е тривиална – кутийката за име трябва да стане празна, а кутийката за цена и количество трябва да получат стойности 0. **Забележете, че боравим със свойството Text. То е от тип string, съответно поради тази причина използваме кавички и за нулите:**

```
private void ClearTextBoxes()
{
    txtName.Text = "";
    txtPrice.Text = "0";
    txtStock.Text = "0";
}
```

В тялото на Form1_Load() метода трябва само да добавим извикване на **UpdateGrid()** и **ClearTextBoxes()**.

Сега нека да реализираме добавянето на продукт. Натиснете два пъти върху Insert бутончето. Visual Studio отново ще генерира за вас код на метод **private void btnInsert_Click(object sender, EventArgs e)**. В този метод трябва да прочетем стойностите от кутийките, да създадем обект от клас Product, на който да зададем за стойности на свойствата стойностите от кутийките, след което да извикаме метода Add на обекта на бизнес логиката. За да може да ползваме този обект, трябва да го създадем като поле на класа на формата. Това се случва по познатия за вас начин:

```
public partial class ProductForm : Form
{
    private ProductBusiness productBusiness = new ProductBusiness();
```

Тялото на метода пък е както следва:

```
private void btnInsert_Click(object sender, EventArgs e)
{
    var name = txtName.Text;
    decimal price = 0;
    decimal.TryParse(txtPrice.Text, out price);
    int stock = 0;
    int.TryParse(txtStock.Text, out stock);

    Product product = new Product();
    product.Name = name;
    product.Price = price;
    product.Stock = stock;

    productBusiness.Add(product);
    UpdateGrid();
    ClearTextBoxes();
}
```

Забележете, че накрая извикваме и UpdateGrid(), както и ClearTextBoxes()

Сега нека да реализираме функционалност и при натискане на Update.

Ще започнем с проверка дали сме избрали поне 1 ред. Ако има такъв, то достъпваме обект от данните от колоните в таблицата. В колоната с индекс 0 е записано Id-то. Това Id е от особена важност за нас, тъй като трябва да го запишем в поле на класа. След това извикваме **UpdateTextBoxes()** метод, който да зададе на кутийките стойностите на продукта със съответното избрано Id. Освен това ще изикваме последователно и методите **ToggleSaveUpdate()**, както и **DisableSelect()**.

Нека да пристъпим към реализацията на тези методи:

- **UpdateTextBoxes(int id)** – този метод ще извлича данните за продукт по неговото **id** чрез извикване на бизнес логиката, след което ще задава **.Text** свойствата на текстовите кутийки. Кодът е както следва:

```
private void UpdateTextboxes(int id)
{
    Product update = productBusiness.Get(id);
    txtName.Text = update.Name;
    txtPrice.Text = update.Price.ToString();
    txtStock.Text = update.Stock.ToString();
}
```

- **ToggleSaveUpdate() –** този метод проверява кой бутон е видим и кой не. Ако бутонът Update е видим – правим го невидим и показваме Save. В противен случай – скриваме Save и показваме Update.

```
private void ToggleSaveUpdate()
{
    if (btnUpdate.Visible)
    {
        btnSave.Visible = true;
        btnUpdate.Visible = false;
    }
    else
    {
        btnSave.Visible = false;
        btnUpdate.Visible = true;
    }
}
```

- **DisableSelect() –** спира възможността за избиране на редове от табличката, за да не бъде избран някой друг ред, докато редактираме.

```
private void DisableSelect()
{
    dataGridView1.Enabled = false;
}
```

Сега нека да реализираме и самият метод, който Visual Studio генерира за нас при натискане на бутон Update:

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    if (dataGridView1.SelectedRows.Count > 0)
    {
        var item = dataGridView1.SelectedRows[0].Cells;
        var id = int.Parse(item[0].Value.ToString());
        editId = id;
        UpdateTextboxes(id);
        ToggleSaveUpdate();
        DisableSelect();
    }
}
```

Не забравяйте да реализирате и полето editId на класа:

```
public partial class ProductForm : Form
{
    private ProductBusiness productBusiness = new ProductBusiness();
    private int editId = 0;
```

След като сме готови с това, нека да реализираме и метода при натискане на бутон Save:

При него ще извикваме **GetEditedProduct()**, който ще връща обект от **Product**, създаден на базата на текстовите кутийки. След това този обект се подава на бизнес логиката и извикваме методите **UpdateGrid()**, **ResetSelect()** и **ToggleSaveUpdate().**

От тези методи все още не сме реализирали **GetEditedProduct().** Неговият код е както следва:

```
private Product GetEditedProduct()
{
    Product product = new Product();
    product.Id = editId;

    var name = txtName.Text;
    decimal price = 0;
    decimal.TryParse(txtPrice.Text, out price);
    int stock = 0;
    int.TryParse(txtStock.Text, out stock);
    product.Name = name;
    product.Price = price;
    product.Stock = stock;

    return product;
}
```

Другият досега нереализиран метод е **ResetSelect()**, който изчиства самата селекция на таблицата:

```
private void ResetSelect()
{
    dataGridView1.ClearSelection();
    dataGridView1.Enabled = true;
}
```

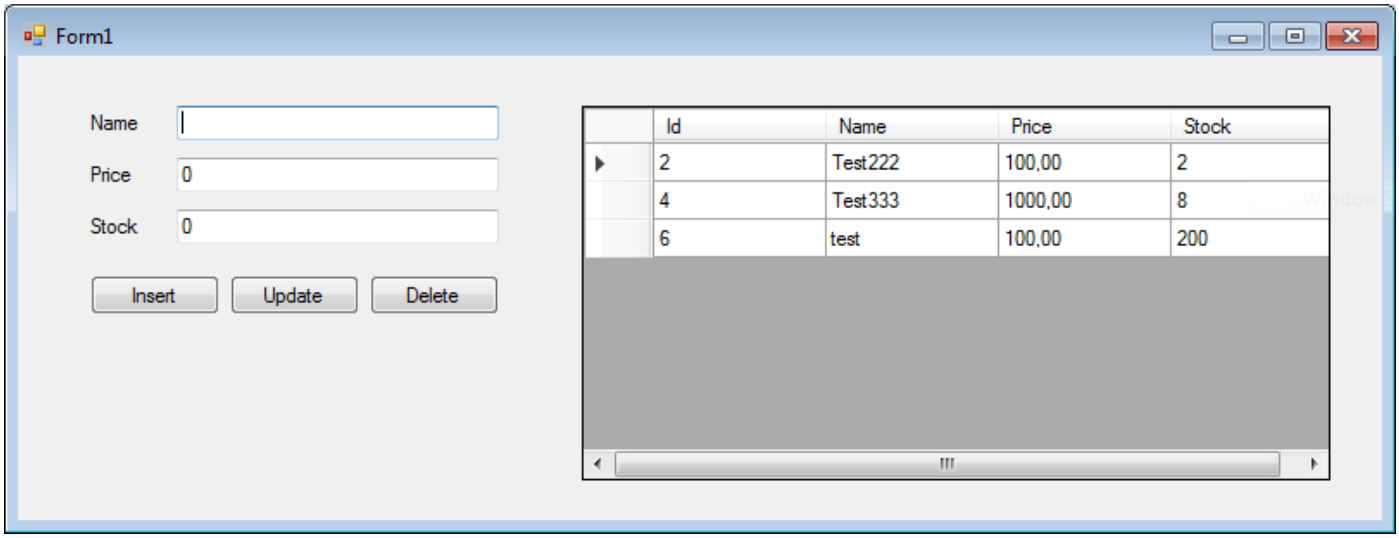
Накрая реализирайте и самият **btnSave_Click()** метод:

```
private void btnSave_Click(object sender, EventArgs e)
{
    Product editedProduct = GetEditedProduct();
    productBusiness.Update(editedProduct);
    UpdateGrid();
    ResetSelect();
    ToggleSaveUpdate();
}
```

И на последно място, но не по значение, нека да реализираме и функционалността на бутончето Delete. Логиката тук е сходна с тази на редактирането, с разликата, че като получим id-то, изпращаме продукта за изтриване чрез бизнес логиката.

```
private void btnDelete_Click(object sender, EventArgs e)
{
    if (dataGridView1.SelectedRows.Count > 0)
    {
        var item = dataGridView1.SelectedRows[0].Cells;
        var id = int.Parse(item[0].Value.ToString());
        productBusiness.Delete(id);
        UpdateGrid();
        ResetSelect();
    }
}
```

С това приключихме реализацията. Крайният резултат би трябвало да изглежда по подобен начин:



Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист".

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).