

Лабораторная работа №12

Разработка клиентского приложения

Цель работы: получение навыков написания клиентского приложения на языке C#.

Теоретическая часть

Заготовку приложения можно скачать из репозитория [SupplyApp](#). Обратите внимание, что у данного проекта версия .NET Framework 4.5.

Шаг 1

На сервере необходимо развернуть БД Supply, скрипт лежит в репозитории (файл SupplyQuery.sql).

Шаг 2

Создайте Windows Forms приложение с именем SupplyApp. Задайте следующие свойства формы:

- **Name** : SupplyForm
- **Text** : Поставки
- **BackColor** : Window
- **StartPosition** : CenterScreen

Шаг 3

Добавьте на форму элемент **TabControl**. Задайте свойства:

- **Name** : tabControlSupply
- Вкладки:
 - **Name** : tabPageItem, **Text** : Товары
 - **Name** : tabPageSupplier, **Text** : Поставщики

- Name : tabPageSupply, Text : Поставки

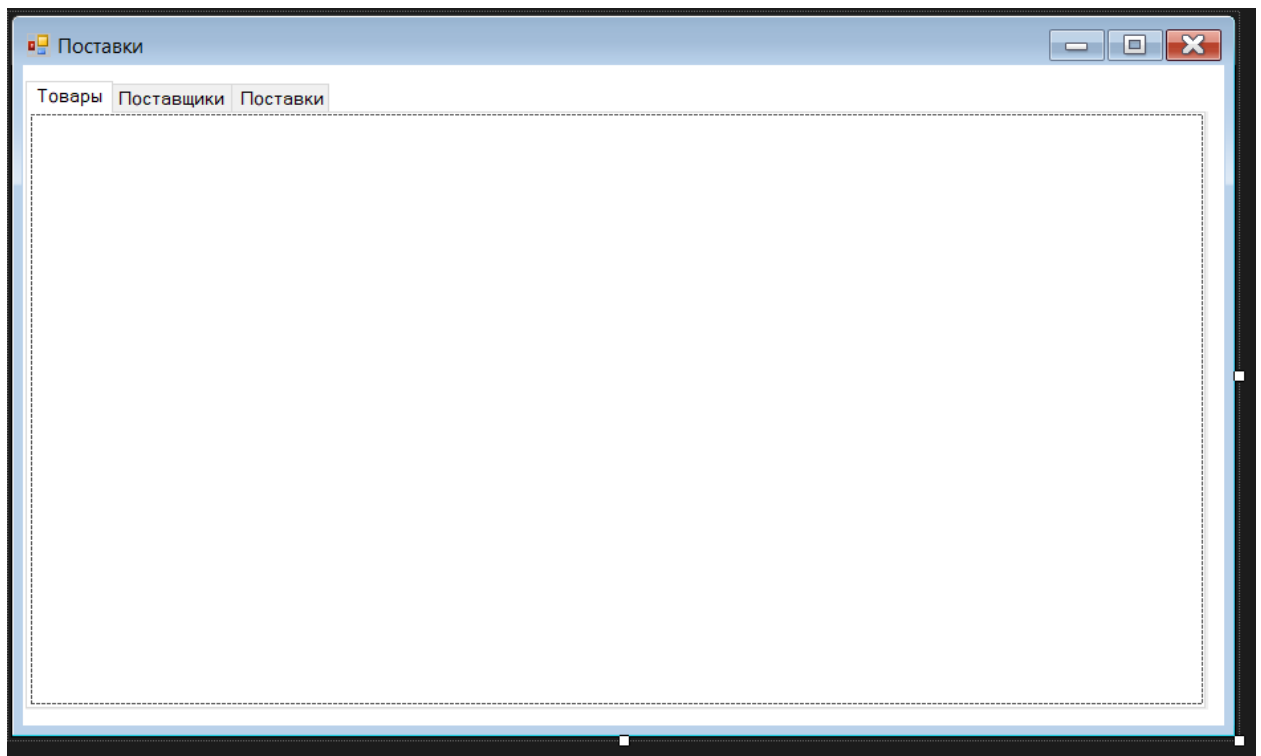


Рисунок 1 – Форма SupplyForm

Шаг 4

Добавьте на вкладку Товары элемент **DataGridView**. Он нужен для отображения данных в виде таблицы. Задайте свойства:

- Name : itemGrid
- BackgroundColor : Window
- BorderStyle : None
- ColumnHeadersBorderStyle : Single
- ColumnHeadersCellStyle : как на рисунке 2
- ColumnHeaderHeightSizeMode : AutoSize
- DefaultCellStyle : как на рисунке 2
- GridColor : ActiveCaption

- RowHeadersVisible : False
- RowHeadersWidthSizeMode : DisableResizing
- RowTemplate.Height : 24
- ScrollBars : Vertical

▼ Appearance	
BackColor	<input type="checkbox"/> Window
> Font	Microsoft Sans Serif; 7,8pt
ForeColor	<input checked="" type="checkbox"/> WindowText
SelectionBackColor	<input checked="" type="checkbox"/> GradientActiveCaption
SelectionForeColor	<input type="checkbox"/> HighlightText
▼ Behavior	
Format	
▼ Data	
NullValue	
▼ Layout	
Alignment	MiddleCenter ▼
> Padding	0; 0; 0; 0
WrapMode	True

Рисунок 2 – Стили ячеек таблицы

Шаг 5

Добавьте элемент `ContextMenuStrip`, задайте контролу имя `itemContextMenu`. Задайте его в качестве `ContextMenuStrip` для `tabPageItem` и `itemGrid`. Добавьте в контекстное меню кнопки как на рисунке 3, задав им соответствующие назначению имена (рисунок 4).

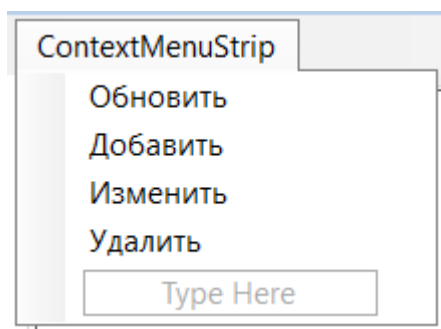


Рисунок 3 – Кнопки меню

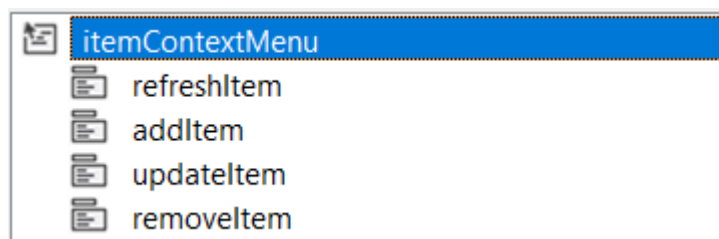


Рисунок 4 – Имена кнопок

Шаг 6

Теперь к проекту необходимо подключить Entity Framework. Для этого нужно открыть диспетчер пакетов NuGet (Проект – Управление пакетами NuGet), ввести в поиске Entity Framework и установить. Если возникнут ошибки – попробуйте изменить версию пакета.

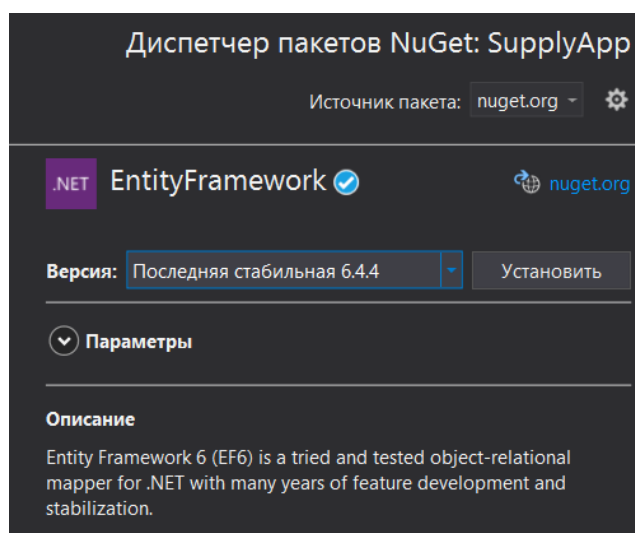


Рисунок 5 – Подключение EF

Теперь добавим новый элемент в проект (правой кнопкой по проекту), выбираем тип и задаем название **SupplyModel**.

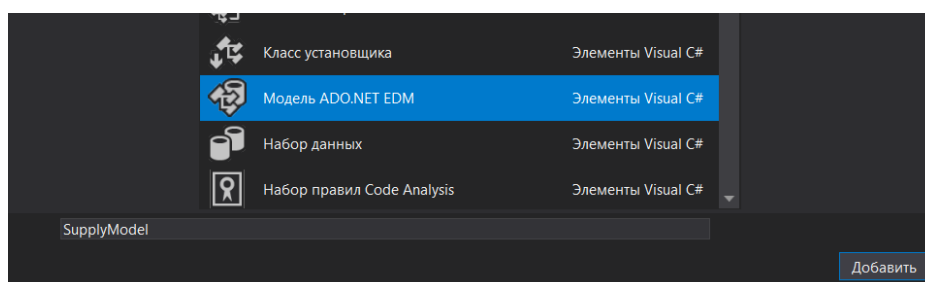


Рисунок 6 – Добавление модели

Далее выбираем «Code first из базы данных»

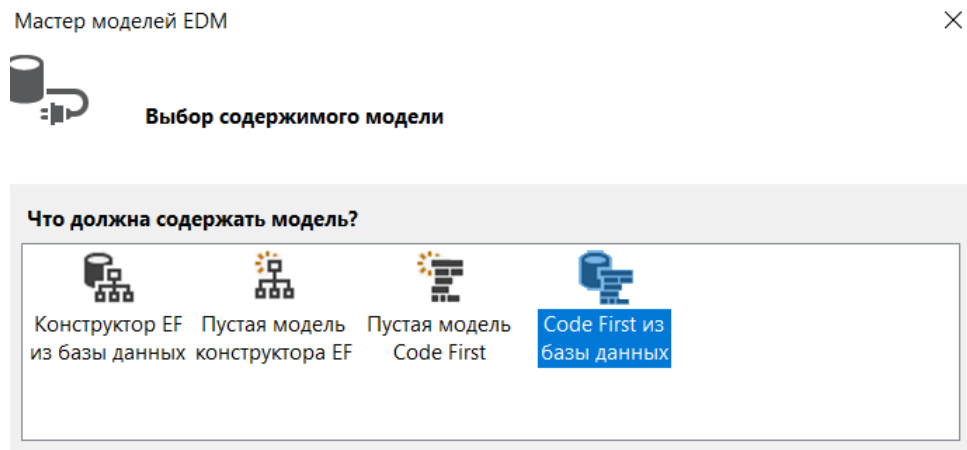


Рисунок 7 – Выбор содержимого модели

Выбираем БД Supply, задаем название для параметра в конфиге.

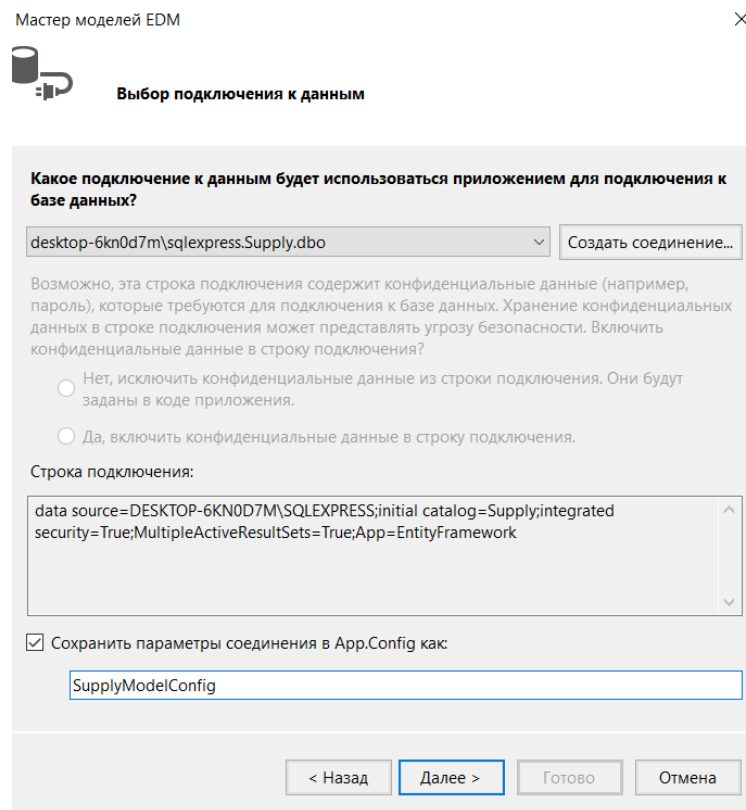


Рисунок 8 – Создание подключения

Выбираем таблицы, по которым фреймворк будет создавать классы.

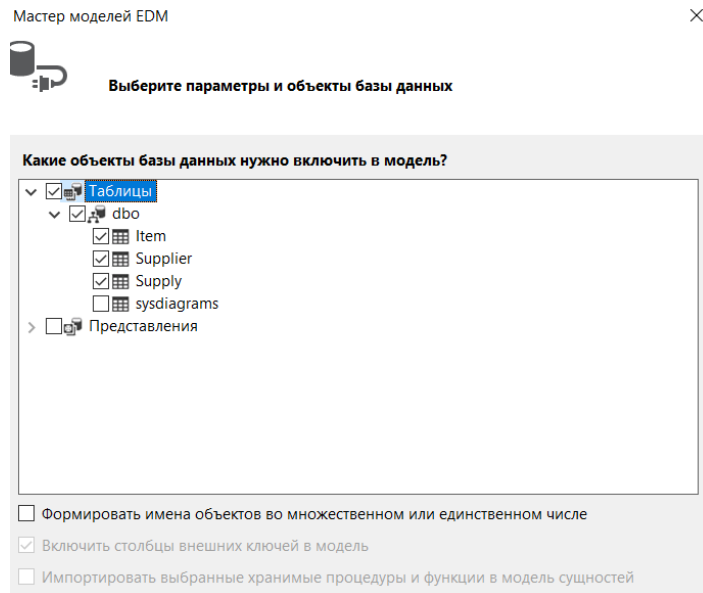


Рисунок 9 – Выбор объектов БД

Появились 3 класса, можно создать для них отдельную папку

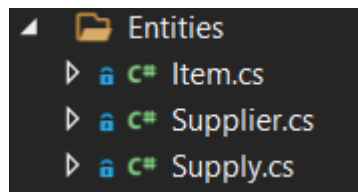


Рисунок 10 – Сгенерированные классы

Шаг 7

Теперь заполним информацию в таблице **itemGrid** данными из БД. Для этого используется следующие методы:

```
public IEnumerable<Item> GetItems()
{
    using (var db = new SupplyModel())
    {
        return db.Item.ToList();
    }
}

private void SetItemGrid()
{
    if(itemGrid.ColumnCount < 4)
    {
        itemGrid.Columns.Add("colId", "Артикул");
        itemGrid.Columns.Add("colName", "Наименование");
        itemGrid.Columns.Add("colManufacturer", "Производитель");
        itemGrid.Columns.Add("colPrice", "Цена");
    }
}
```

```

itemGrid.Rows.Clear();

foreach (var item in GetItems())
{
    itemGrid.Rows.Add(item.ID, item.Name, item.Manufacturer, item.Price);
}
}

```

В методе **GetItems** открывается подключение к БД и выполняется выборка из таблицы **Item**, результат возвращается в виде списка.

В методе **SetItemGrid** в таблицу добавляются столбцы (изначально мы их не добавляли), затем таблица очищается от существующей информации. Далее в цикле построчно заносятся данные в таблицу.

Можно попробовать и другой способ. Для многих элементов управления можно указать коллекцию в качестве **DataSource**. В случае с таблицами столбцы и строки будут автоматически добавляться. Иногда требуется вручную изменить заголовки столбцов.

Если в качестве источника данных указать **db.Item.ToList()**, то вместе с информацией о каждом товаре подтянется поле **Supply** (связь по внешнему ключу). Можно вручную скрыть этот столбец.

ID	Name	Manufacturer	Price	Supply
1	Вилка	Арго	20,0000	
2	Вилка	Гурман	25,0000	
3	Ложка	Болео	30,0000	
4	Нож	Гурман	500,0000	
5	Нож	Болео	35,0000	
6	Кастрюля	Гурман	500,0000	

Рисунок 11 – Дополнительный столбец

Можно разработать метод, который будет решать эту проблему.

```

public IEnumerable GetItems()
{
    using (var db = new SupplyModel())
    {
        return db.Item.Select(x =>
            new { x.ID, x.Name, x.Manufacturer, x.Price }).ToList();
    }
}

```

Тогда метод для заполнения таблицы будет выглядеть так:

```
private void SetItemGrid()
{
    itemGrid.Rows.Clear();
    itemGrid.DataSource = GetItems();

    itemGrid.Columns[0].HeaderText = "Артикул";
    itemGrid.Columns[1].HeaderText = "Наименование";
    itemGrid.Columns[2].HeaderText = "Производитель";
    itemGrid.Columns[3].HeaderText = "Стоимость";
}
```

Шаг 8

Пусть данные из БД добавляются как при запуске приложения, так и при нажатии кнопки Обновить.

```
private void SupplyForm_Load(object sender, EventArgs e)
{
    SetItemGrid();
}

// Обновляем данные в таблице при нажатии кнопки Обновить
private void refreshItem_Click(object sender, EventArgs e)
{
    SetItemGrid();
}
```

Шаг 9

Добавим форму **AddItemForm** для добавления нового товара.

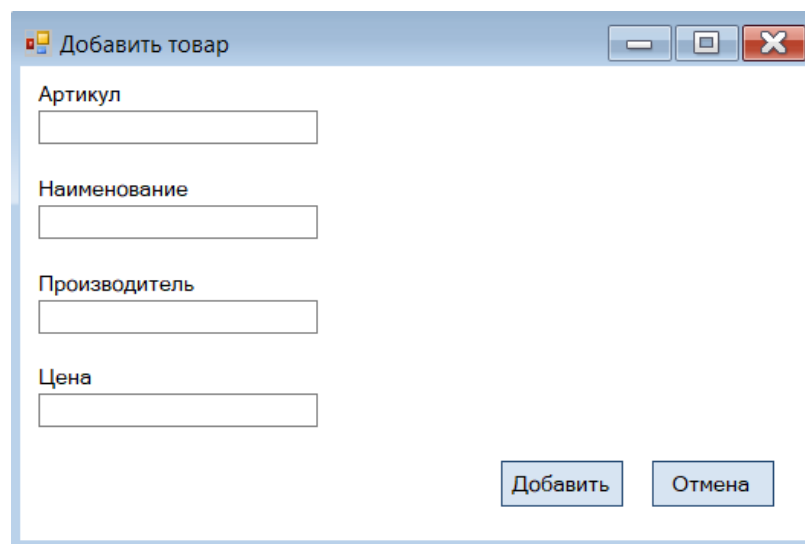


Рисунок 12 – Форма добавления товара

Контролы на форме будут иметь следующие имена:

```
private System.Windows.Forms.Button btnCancel;
private System.Windows.Forms.Button btnAdd;
```



```

private System.Windows.Forms.Label lblPrice;
private System.Windows.Forms.Label lblManufacturer;
private System.Windows.Forms.Label lblName;
private System.Windows.Forms.Label lblId;
private System.Windows.Forms.TextBox txtPrice;
private System.Windows.Forms.TextBox txtManufacturer;
private System.Windows.Forms.TextBox txtName;
private System.Windows.Forms.TextBox txtId;

```

Также добавим на форму элемент **ErrorProvider** с именем **errorProvider**. С его помощью мы запретим пользователю вводить в текстовые поля некорректные данные. Для этого будем работать с событиями **Validating** (действия при вводе данных) и **Validated** (действия, когда данные введены верно) текстовых полей. Для корректной работы **errorProvider** зададим значение свойства формы **AddItemForm.AutoValidate = EnableAllowFocusChange**.

Объявим в классе формы поля:

```

private int _id;
private string _name;
private string _manufacturer;
private decimal _price;

```

Рассмотрим обработчики для работы с элементом **txtId**:

```

private void txtId_Validating(object sender, CancelEventArgs e)
{
    string input = txtId.Text.Trim();
    if (Regex.IsMatch(input, @"(?<=\s|^)\d+(?=\s|$)"))
    {
        errorProvider.SetError(txtId, String.Empty);
        e.Cancel = false;
    }
    else
    {
        errorProvider.SetError(txtId, "Ошибка!");
        e.Cancel = true;
    }
}

```

Здесь **errorProvider** будет выводить сообщение об ошибке, если данные в **txtId** не будут соответствовать регулярному выражению.

```

private void txtId_Validated(object sender, EventArgs e)
{
    _id = Convert.ToInt32(txtId.Text.Trim());
}

```

Заносим валидные данные в соответствующее поле.

Аналогичные обработчики разработайте для остальных текстовых полей самостоятельно.

Теперь добавим обработчики для кнопок Добавить и Отмена.

```
private void btnCancel_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.None;
    this.Close();
}

private void btnAdd_Click(object sender, EventArgs e)
{
    DialogResult = ValidateChildren() ? DialogResult.OK : DialogResult.None;
    if (DialogResult == DialogResult.OK)
    {
        AddItem();
        this.Close();
    }
    else
    {
        MessageBox.Show("Введите корректные данные!", "Ошибка",
        MessageBoxButtons.OK);
    }
}
```

В обработчике кнопки Добавить проверяем, все ли элементы управления прошли валидацию. Если да, то вызываем метод добавления в БД новой записи.

```
private void AddItem()
{
    try
    {
        using (var db = new SupplyModel())
        {
            db.Item.Add(new Item
            {
                ID = _id,
                Name = _name,
                Manufacturer = _manufacturer,
                Price = _price
            });
            db.SaveChanges();
        }
        MessageBox.Show("Данные добавлены!", "Добавлено", MessageBoxButtons.OK);
    }
    catch (Exception)
    {
        MessageBox.Show("Ошибка в данных!", "Ошибка", MessageBoxButtons.OK);
    }
}
```

Как правило, исключение в этом методе возникает при добавлении данных с повторяющимся первичным ключом (хотя могут быть и другие ситуации, например, отсутствие подключения к БД).

Шаг 10

Теперь для **SupplyForm** добавим обработчик кнопки **Добавить** КОНТЕКСТНОГО МЕНЮ.

```
private void addItem_Click(object sender, EventArgs e)
{
    AddItemForm add = new AddItemForm();
    if (add.ShowDialog(this) == DialogResult.OK)
    {
        SetItemGrid();
    }
}
```

Также добавим обработчик для кнопки **Обновить**.

```
private void refreshItem_Click(object sender, EventArgs e)
{
    SetItemGrid();
}
```

И обработчик для кнопки **Удалить**.

```
private void removeItem_Click(object sender, EventArgs e)
{
    DialogResult result = MessageBox.Show("Вы уверены, что хотите удалить этот элемент?",
        "Удаление", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
    if (result == DialogResult.Yes)
    {
        if (itemGrid.SelectedCells.Count > 0)
        {
            var i = itemGrid.SelectedCells[0].OwningRow.Index;
            int itemId = (int)itemGrid[0, i].Value;
            using (var db = new SupplyModel())
            {
                Item item = db.Item.Where(x => x.ID == itemId).First();
                db.Item.Remove(item);
                db.SaveChanges();
            }
        }
        SetItemGrid();
    }
}
```

Практическая часть

Требуется доработать приложение для работы с БД Supply со следующими требованиями:

- на одной форме должны располагаться все три таблицы, каждая таблица в отдельной вкладке;
- для каждой из таблиц необходимо предусмотреть обновление, добавление, изменение и удаление данных через контекстное меню;
- данные из подчиненной таблицы Supply нужно выводить в виде: **дата поставки, наименование поставщика, артикул товара, наименование товара, объем поставки, общая стоимость поставки.**

Чтобы получить данные из связанных таблиц, можно использовать код:

```
public IEnumerable GetSupplies()
{
    using (var db = new SupplyModel())
    {
        return db.Supply.Select(supply => new
        {
            supply.Date,
            Supplier = supply.Supplier.Name,
            supply.ItemID,
            Item = supply.Item.Name,
            supply.Volume,
            Overall = supply.Volume * supply.Item.Price
        }).ToList();
    }
}
```

- при редактировании данных нельзя менять значение первичных ключей;
- при вставке данных в подчиненную таблицу пользователь не должен вручную вводить **ItemID** и **SupplierID**. Нужно предусмотреть выбор нужного поставщика и товара через выпадающие списки с наименованиями. Самый простой способ – передать в **ComboBox**

коллекцию в качестве **DataSource** (можно искать и другие способы);

- для ввода телефона поставщика используйте **MaskedTextBox**.

Содержание отчета

1. Титульный лист
2. Цель работы
3. Задание
4. Код программы
5. Результат выполнения