

ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

СОВРЕМЕННЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ

*Методические рекомендации к выполнению лабораторных работ
для студентов дневной и заочной форм получения образования специальности
1-53 01 02 «Автоматизированные системы обработки информации»*

2 семестра

УДК 681.3

ББК 32.973
К68

Рекомендовано к изданию
Учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «7» марта 2022 г., протокол № 8.

Составитель: ст. преп. Выговская Н.В.

Рецензент

Приведены методические рекомендации для самостоятельной работы по дисциплине «Современные системы программирования». Методические рекомендации также содержат краткие теоретические сведения, необходимые для выполнения работ и контрольные вопросы для защиты лабораторных работ.

Учебно-методическое издание

СОВРЕМЕННЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ

Ответственный за выпуск
Технический редактор
Компьютерная верстка

А.И.Якимов

Подписано в печать . Формат 60x84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Ус. печ. л. . Уч.-изд.л. . Тираж экз. Заказ №

Издатель и полиграфическое исполнение:
Государственное учреждение высшего профессионального образования
«Белорусско-Российский университет»
Свидетельство о государственной регистрации издателя,
Изготовителя, распространителя печатных изданий
№1/156 от 24.01.2014.
Пр. Мира, 43, 212000, Могилев

Содержание

Лабораторная работа №1	5
«Введение в XAML и WPF».....	5
Лабораторная работа №2	10
«Диспетчеры компоновки»	10
Лабораторная работа №3	18
Основные элементы управления WPF	18
Лабораторная работа №4	27
«Привязка данных в WPF»	27
Лабораторная работа №5	33
«Использование стилей в WPF-приложениях».....	33
Лабораторная работа №6	38
«Триггеры в WPF-приложениях».....	38
Лабораторная работа №7	43
«Анимация в WPF-приложениях»	43
Лабораторная работа №8	48
«Трансформация в WPF-приложениях»	48
Лабораторная работа №9	56
«Использование кистей в WPF-приложениях»	56
Лабораторная работа №10	69
«Использование фигур в WPF-приложениях»	69
Лабораторная работа №11-12.....	80
«Разработка ASP.Net-приложения с базой данных: модель, контроллер и представления»	80
Тестирование MVC-приложения.....	125
Лабораторная работа №13	126
Функциональное программирование.	126
Лабораторная работа №14	138
Разработка приложений методом TDD	138
Лабораторная работа № 15	150
Анализ предметной области. Разработка диаграммы вариантов использования	150
Лабораторная работа №16	167
Разработка приложения ASP.Net Core MVC	167
Лабораторная работа №17	193
Разработка Тест-плана	193

Лабораторная работа №18	20207
Разработка мобильных приложений для ОС Андроид.....	20207

Введение

Цель методических рекомендаций к самостоятельной работе по дисциплине «Современные системы программирования» заключается в овладении студентами практических навыков разработки приложений в среде Microsoft Visual Studio на платформе WPF.

Цель изучения дисциплины - подготовка студентов к использованию современных технологий и программных средств как профессионального инструмента для решения научных и практических задач.

Дисциплина «Современные системы программирования» является неотъемлемой частью знаний инженера-программиста и связана с разработкой современных программных систем и комплексов.

Полученные при изучении дисциплины знания и навыки могут быть востребованы в курсовом и дипломном проектировании и станут инструментом для реализации собственных проектов.

Общие требования ко всем лабораторным работам:

Отчёт включает в себя:

- титульный лист
- цель работы
- коды программ
- скриншот с выполнением программы с подписями рисунков
- вывод по лабораторной работе (чему научились)

Лабораторная работа №1

«Введение в XAML и WPF»

WPF и XAML

Графическая система Windows Presentation Foundation предназначена для создания пользовательских интерфейсов, 2D и 3D графики. Преимущество WPF заключается в том, что 2D графика строится в векторном виде, а это значит, что интерфейсы будут максимально независимы от разрешения экрана и размера окна.

Они будут легко масштабироваться без потери качества и быстро работать благодаря максимальному использованию возможностей современных графических ускорителей. WPF объединяет документы, формы и мультимедийное содержание в пакет, состоящий из языка разметки и процедурного языка программирования.

Для создания и инициализации объектов в WPF используется язык разметки XAML - Extensible

Application Markup Language (расширяемый язык разметки приложений). XAML использует основные четыре категории элементов:

- панели размещения;
- элементы управления;
- элементы, связанные с документом;
- графические фигуры.

XAML является диалектом языка XML. Файл XAML содержит ровно одну корневую вершину и является деревом отображения. На вершине иерархии находится один из контейнерных объектов. Внутри этих объектов располагаются элементы управления и другие контейнеры. В XAML названия элементов чувствительны к регистру и совпадают с именами классов, доступных в кодовой части WPF.

Цель работы: научиться создавать проекты в WPF.

Задание 1:

В среде Microsoft Visual C# 2010 Express создайте проект «Приложение WPF». Среда разработки создаст заготовку, показанную на рисунке:



Рисунок 1 – Окно проекта WPF с кодом Xaml.

Среда разработки предоставляет возможность графического и дескрипторного способов разработки пользовательского интерфейса, которые являются равнозначными. Дескрипторный файл MainWindow.xaml и кодовый файл MainWindow.xaml.cs дополняют друг друга при описании одного и того же содержимого - класса MainWindow в пространстве имен WpfApplication1, совпадающим с названием проекта.

Платформа WPF проектировалась в рамках концепции отделения дизайнерской части интерфейса от кодовой части программирования функциональности. Дизайнерская часть проектируется декларативно, чаще всего - с помощью графического дизайнера формы, который в автоматическом режиме генерирует соответствующий синтаксически правильный дескрипторный код на языке XAML.

В заготовке дескрипторного XAML-кода видно, что корнем приложения является контейнер <Window>, в который в дальнейшем будут включены дочерние элементы. Все элементы WPF существуют в двух вариантах: дескрипторном и объектном. Объектное описание размещается в пространствах имен, подключаемых в кодовую часть проекта с помощью инструкции `using` для видимости компилятором.

Дескрипторное описание находится в двух пространствах имен: обычном и расширенном. Эти пространства имен подключаются как значения атрибутов `xmlns` и `xmlns:x` в корневом дескрипторе проекта

```

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

```

Используемые URL-адреса не указывают на какой-либо документ или содержимое на веб-сервере, а используются лишь для определения уникальных пространств имен.

Разместите в коде XAML в содержимом элемента Grid следующий код:

```
<Button x:Name="Btn1"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Width="150"
    Height="30"
    FontSize="17"
    Content="Обычная кнопка"
    Background="#f0f0f0"
    BorderBrush="#303030" />
```

Запустите приложение и проверьте его поведение при изменении размеров окна.

В приведенном выше примере для элемента Button было задано простое значение атрибута

Background. Для этого был использован синтаксис

<ЭЛЕМЕНТ АТРИБУТ=”ЗНАЧЕНИЕ” />

Для задания значения атрибуты может быть использован другой синтаксис:

```
<ЭЛЕМЕНТ>
<ЭЛЕМЕНТ.АТРИБУТ>
    ЗНАЧЕНИЕ_АТРИБУТА
</ЭЛЕМЕНТ. АТРИБУТ>
</ЭЛЕМЕНТ>
```

Например, для задания того же значения атрибута Background можно записать:

<Button>

```
<Button.Background>
    #f0f0f0
</Button.Background>
</Button>
```

Данный синтаксис используется для задания сложных значений атрибутов в виде дерева элементов.

Пример задания для фона кнопки линейной градиентной заливки:

```
<Button.Background>
    <LinearGradientBrush>
        <LinearGradientBrush.GradientStops>
            <GradientStop Color="LightBlue" Offset="0" />
            <GradientStop Color="DarkBlue" Offset="1" />
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
</Button.Background>
```

Данное дерево элементов задает градиентную заливку с использованием двух цветов: LightBlue и DarkBlue. В атрибуте Offset указывается относительное значение от 0 до 1, соответствующее положению цвета на отрезке от начальной точки до конечной.

Внешний вид данного приложения:

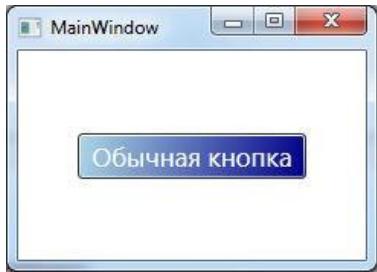


Рисунок 2 – Внешний вид кнопки

Объект Application

Любое приложение использует класс Application, который организует его подключение к модели событий операционной системы с помощью метода Run(). Объект Application отвечает за управление временем жизни приложения, отслеживает видимые окна, освобождает ресурсы и контролирует глобальное состояние приложения. Метод Run() запускает диспетчер среды исполнения, который начинает посыпать события и сообщения компонентам приложения.

В каждый момент времени может быть активен только один объект Application и он будет работать до тех пор, пока приложение не завершится. К исполняемому объекту Application можно получить доступ из любого места приложения через статическое свойство Application.Current. Одна из основных задач объекта Application состоит в том, чтобы контролировать время жизни процесса. Конструирование объекта

Application знаменует начало жизни приложения, а возврат из его метода Run() - завершение приложения.

Время жизни приложения WPF и объекта Application состоит из следующих этапов:

1. Конструируется объект Application
2. Вызывается его метод Run()
3. Выполняется событие Application.Startup
4. Пользовательский код конструирует один или несколько объектов Window (или Page) и приложение выполняет работу
5. Вызывается метод Application.Shutdown()
6. Вызывается метод Application.Exit()

При создании проекта среда разработки поместила в проект два файла, связанные с объектом Application: App.xaml и App.xaml.cs. В этих файлах нет кода, создающего объекты Application и Windows и вызывающего метод Run() – это происходит неявно. В файле App.xaml для элемента Window задается атрибут StartupUri, в котором определяется имя XAML-файла с окном, которое открывается при запуске приложения.

Задание 2:

В XAML-коде для элемента Windows определите линейную градиентную заливку фона в соответствии с рисунком:



Рисунок 3- Градиентная заливка фона

Используемые цвета: DarkBlue и LightBlue.

Необходимо указать четыре промежуточные точки со смещениями 0, 0.2, 0.8 и 1.

Для задания вертикальной заливки необходимо определить атрибуты StartPoint и EndPoint для элемента LinearGradientBrush. Значения этих атрибутов указываются в формате “X,Y”, где X – относительное значение (от 0 до 1) абсциссы точки, Y – относительное значение (от 0 до 1) ординаты точки. Начало координат находится в левом верхнем углу окна. По умолчанию значения атрибутов StartPoint и EndPoint следующие: StartPoint=”0,0” EndPoint=”1,1”.

Контрольные вопросы:

1. Для чего предназначена платформа Windows Presentation Foundation?
2. Что такое XAML?
3. Какие основные четыре категории элементов использует XAML?
4. Перечислите этапы времени жизни приложения WPF и объекта Application?
5. Назовите типы свойств объектов в языке XAML?

Лабораторная работа №2

«Диспетчеры компоновки»

Цель работы: ознакомиться с различными диспетчерами компоновки.

Окно WPF-приложения обычно представлено корневым элементом Window. Дочерним элементом корневого элемента является диспетчер компоновки, который в свою очередь содержит любое количество элементов (в том числе, вложенных диспетчеров компоновки), определяющих пользовательский интерфейс. Диспетчер компоновки является объектом класса, унаследованного от абстрактного класса System.Windows.Controls.Panel.

Основные панели (диспетчеры компоновки, контейнерные элементы управления) WPF:

Элементы остаются в точности там, где были размещены во время проектирования

Привязывает содержимое к определенной стороне панели (Top (верхняя), Bottom (нижняя), Left (левая) или Right (правая))

Располагает содержимое внутри серии ячеек, расположенных в табличной сетке

Выводит содержимое по вертикали или горизонтали, в зависимости от значения свойства Orientation

Позиционирует содержимое слева направо, перенося на следующую строку по достижении границы панели. Последовательность размещения происходит сначала сверху вниз или сначала слева направо, в зависимости от значения свойства Orientation

Диспетчер компоновки Canvas

Панель Canvas поддерживает абсолютное позиционирование содержимого пользовательского интерфейса. Если пользователь изменяет размер окна, делая его меньше, чем компоновка, обслуживаемая панелью Canvas, ее внутреннее содержимое становится невидимым до тех пор, пока контейнер вновь не увеличится до размера, равного или больше начального размера области Canvas.

Панель Canvas обладает следующим недостатком: элементы внутри Canvas не изменяются динамически при применении стилей или шаблонов.

Рассмотрим панель Canvas со следующим содержимым:

<Canvas>

```
<Label Canvas.Left="10" Canvas.Top="10" Content="Регистрация пользователя" />
<Label Canvas.Left="10" Canvas.Top="40" Content="ФИО" />
<TextBox Canvas.Left="70" Canvas.Top="40" Width="200" />
<Label Canvas.Left="10" Canvas.Top="70" Content="Email" />
<TextBox Canvas.Left="70" Canvas.Top="70" Width="200" />
<Button Canvas.Left="50" Canvas.Top="100" Content="Зарегистрироваться" /></Canvas>
```

У элементов управления Label, TextBox, Button отсутствуют атрибуты Left и Top, поэтому для определения положения элементов на панели используется синтаксис присоединяемых свойств.

Присоединяемые свойства XAML (attached properties)

В XAML поддерживается специальный синтаксис, используемый для определения значения присоединяемого свойства. Присоединяемые свойства позволяют дочернему элементу устанавливать значение какого-то свойства, которое в действительности определено в родительском элементе. Общий шаблон:

<РодительскийЭлемент>

```
<ДочернийЭлемент РодительскийЭлемент.СвойствоРодительскогоЭлемента =
"Значение">
```

</РодительскийЭлемент>

С помощью присоединяемых свойств можно определить значения лишь ограниченного набора свойств родительских элементов, которые определены специальным образом в классе родительского элемента.

Дополнительное задание: определите, каким образом присоединяемые свойства объявляются в классе родительского элемента.

Пример работы приложения:



Рисунки 4-5 – пример работы приложения

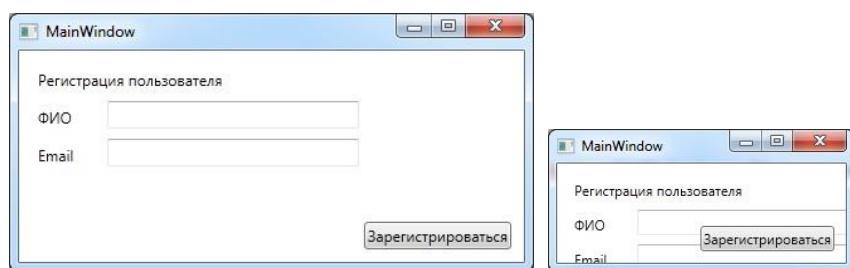
Для дочернего элемента необходимо указать привязку по вертикали (Canvas.Top или Canvas.Bottom) и привязку по горизонтали (Canvas.Left или Canvas.Right). Также можно (не обязательно) задать ширину и высоту элемента с помощью атрибутов Width и Height.

Таким образом, положение дочернего элемента управления можно задать относительно любого угла окна. Например, в следующем примере положение кнопки задано относительно нижнего правого угла окна:

<Canvas>

```
<Button Canvas.Right="10" Canvas.Bottom="10" Content="Зарегистрироваться" />
```

</Canvas>



Рисунки 6-7 – пример работы приложения

Порядок объявления дочерних элементов управления определяет порядок их вывода на экран. В приведенном выше примере кнопка выводится перед текстовыми полями, т.к. она была объявлена в файле XAML последней.

Диспетчер компоновки WrapPanel

Панель WrapPanel выводит дочерние элементы последовательно слева направо (либо сверху вниз, если для атрибута Orientation установлено значение “Vertical”) и при достижении границы окна переходит на новую строку (столбец). При изменении размеров окна панель перераспределяет компоненты таким образом, чтобы они находились в окне.

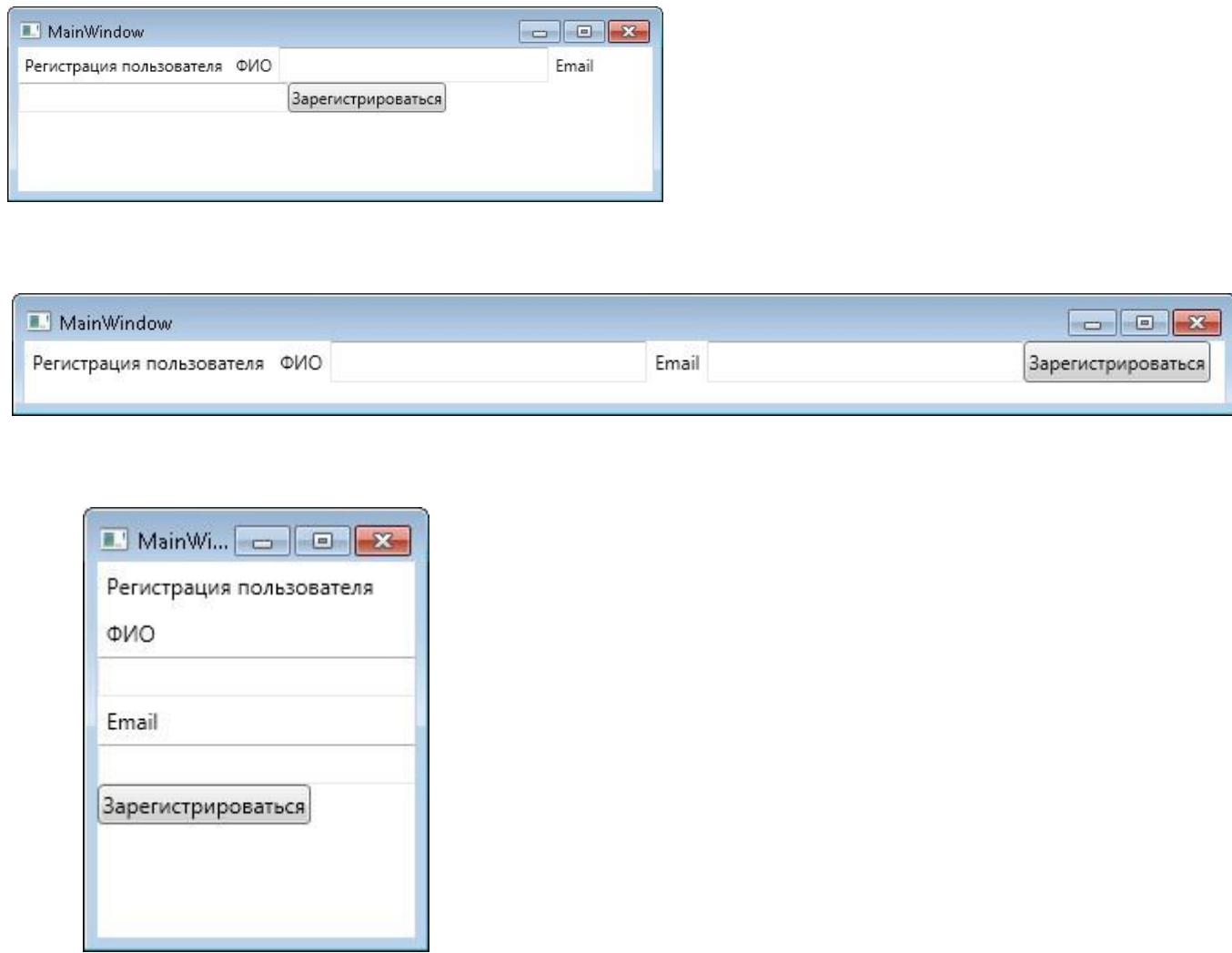
Рассмотрим панель WrapPanel со следующим содержимым:

<WrapPanel>

```
<Label Content="Регистрация пользователя" />
<Label Content="ФИО" />
<TextBox Width="200" />
<Label Content="Email" />
<TextBox Width="200" />
<Button Content="Зарегистрироваться" />
```

</WrapPanel>

Пример работы приложения:



Рисунки 8-10 – пример работы приложения

Диспетчер компоновки StackPanel

Панель StackPanel располагает содержащиеся в нем элементы управления либо в вертикальном столбце (по умолчанию), либо в горизонтальной строке (если в атрибут Orientation записано значение “Vertical”). Если в панель StackPanel добавлено больше элементов управления, чем может быть отображено по ширине/высоте StackPanel, лишние элементы обрезаются и не отображаются.

При выводе элементов сверху вниз элементы по умолчанию растягиваются по горизонтали. Это поведение можно изменить с помощью свойств HorizontalAlignment и VerticalAlignment. Рассмотрим панель StackPanel со следующим содержимым:

```
<StackPanel HorizontalAlignment="Center">
    <Label Content="Регистрация пользователя" />
    <Label Content="ФИО" />
    <TextBox Width="200" />
    <Label Content="Email" />
    <TextBox Width="200" />
    <Button Content="Зарегистрироваться" />
</StackPanel>
```

Пример работы приложения:

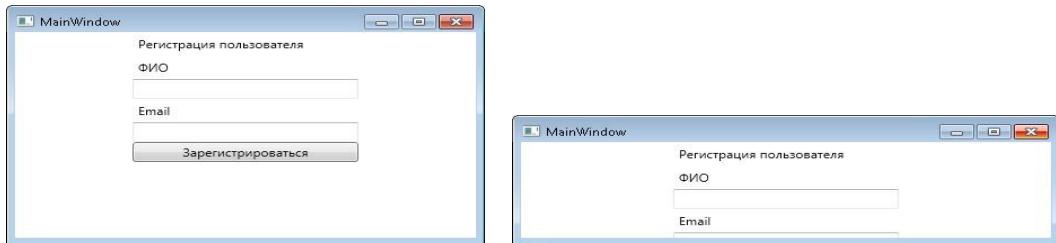


Рисунок 11 – пример работы приложения

Диспетчер компоновки DockPanel

Панель DockPanel пристыковывает дочерние элементы к различным сторонам панели: Top, Bottom, Left, Right. Атрибут LastChildFill по умолчанию имеет значение True, что означает, что последний дочерний элемент управления будет занимать всё оставшееся пространство панели.

Рассмотрим панель DockPanel со следующим содержимым:

```
<DockPanel LastChildFill="False">
    <Label DockPanel.Dock="Top" Content="Регистрация пользователя" />
    <Label DockPanel.Dock="Left" Content="ФИО" />
    <TextBox DockPanel.Dock="Left" Width="200" />
    <Label DockPanel.Dock="Right" Content="Email" />
    <TextBox DockPanel.Dock="Right" Width="200" />
    <Button DockPanel.Dock="Bottom" Content="Зарегистрироваться" />
</DockPanel>
```

Пример работы приложения:

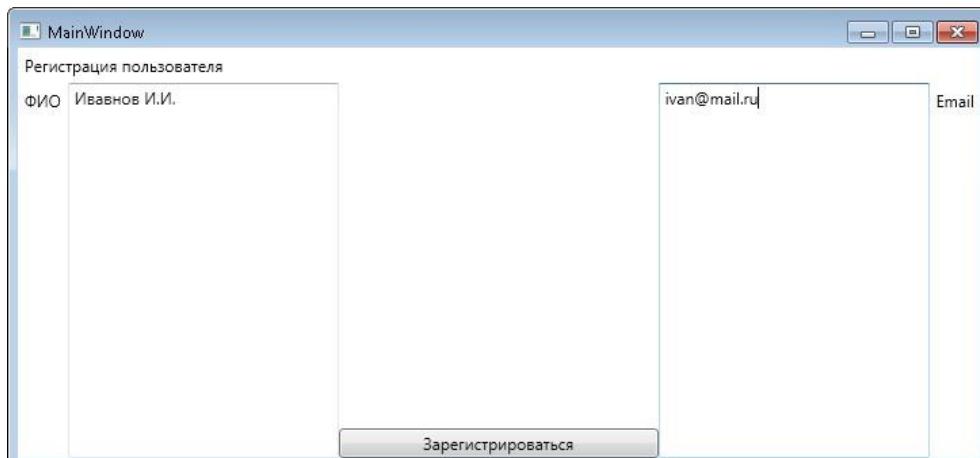


Рисунок 12 – пример работы приложения

Диспетчер компоновки Grid

Подобно HTML-таблице, панель Grid может состоять из набора ячеек, каждая из которых имеет свое содержимое. При определении панели Grid выполняются следующие шаги:

1. Определение и конфигурирование каждого столбца.
2. Определение и конфигурирование каждой строки.
3. Назначение содержимого каждой ячейке сетки с использованием синтаксиса присоединяемых свойств.

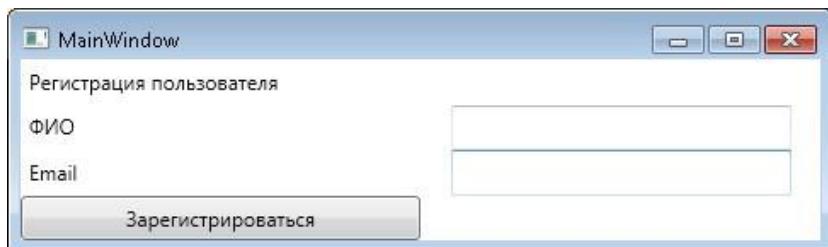
Если не определить никаких строк и столбцов, то по умолчанию панель Grid будет состоять из одной ячейки, занимающей всю поверхность окна. Кроме того, если не указать ячейку для дочернего элемента, то он разместится в столбце 0 и строке 0.

Определение столбцов и строк выполняются за счет использования элементов <Grid.ColumnDefinitions> и <Grid.RowDefinitions>, которые содержат коллекции элементов <ColumnDefinition> и <RowDefinition>, соответственно.

Каждый дочерний элемент прикрепляется к ячейке сетки, используя присоединяемые свойства Grid.Row и Grid.Column. Левая верхняя ячейка определяется с помощью Grid.Column="0" и Grid.Row="0". Рассмотрим панель Grid со следующим содержимым:

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <Label Grid.Row="0" Grid.Column="0" Content="Регистрация пользователя" />
  <Label Grid.Row="1" Grid.Column="0" Content="ФИО" />
  <TextBox Grid.Row="1" Grid.Column="1" Width="200" />
  <Label Grid.Row="2" Grid.Column="0" Content="Email" />
  <TextBox Grid.Row="2" Grid.Column="1" Width="200" />
  <Button Grid.Row="3" Grid.Column="0" Content="Зарегистрироваться" />
</Grid>
```

Пример работы приложения:



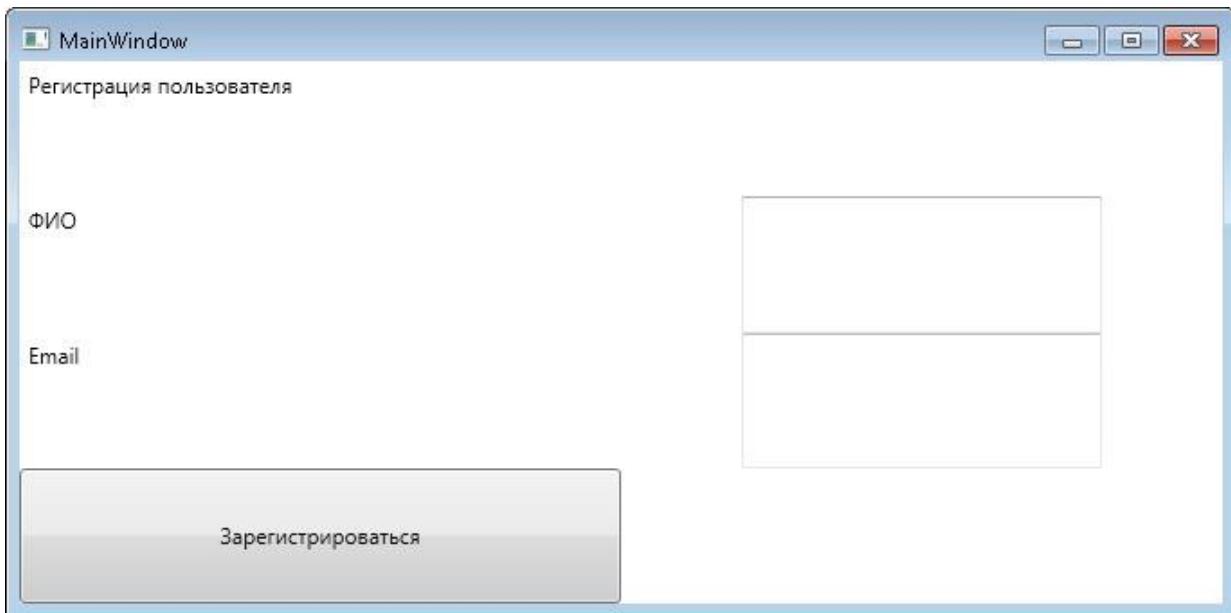


Рисунок 13 – пример работы приложения

Объединение ячеек осуществляется с помощью присоединяемых свойств Grid.ColumnSpan и Grid.RowSpan аналогично объединению ячеек в HTML-таблицах. Рассмотрим панель Grid со следующим содержимым:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Label Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2" HorizontalAlignment="Center"
Content="Регистрация пользователя" />
    <Label Grid.Row="1" Grid.Column="0" Content="ФИО" />
    <TextBox Grid.Row="1" Grid.Column="1" Width="200" />
    <Label Grid.Row="2" Grid.Column="0" Content="Email" />
    <TextBox Grid.Row="2" Grid.Column="1" Width="200" />
    <Button Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="2" HorizontalAlignment="Center"
Content="Зарегистрироваться" />
</Grid>
```

Пример работы приложения:

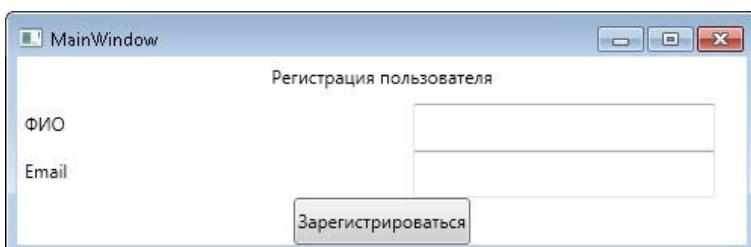


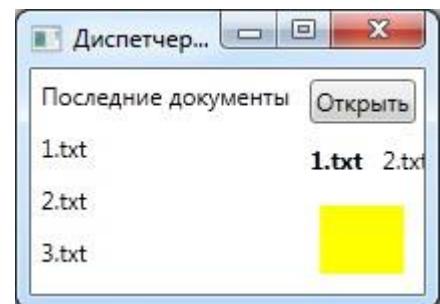
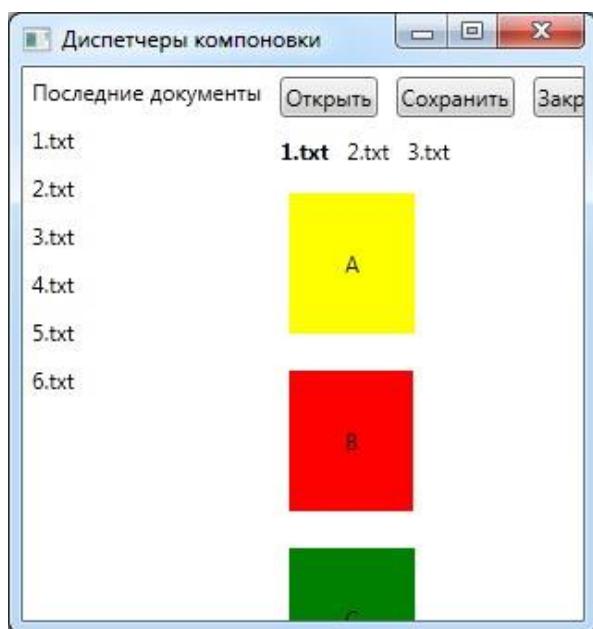
Рисунок 14 – пример работы приложения

При определении ряда можно задать его высоту с помощью атрибута Height, а при определении столбца можно задать его ширину с помощью атрибута Width. Значение этих атрибутов может быть следующим:

- "Auto" - высота строчки (или ширина колонки) определяется содержимым;
- "Число" - высота строчки (или ширина колонки) равна указанному числу точек;
- "*" - высота строчки (или ширина колонки) занимает всё свободное пространство. Если строчек (колонок) с таким значением атрибута несколько, то свободное пространство перераспределяется между ними.

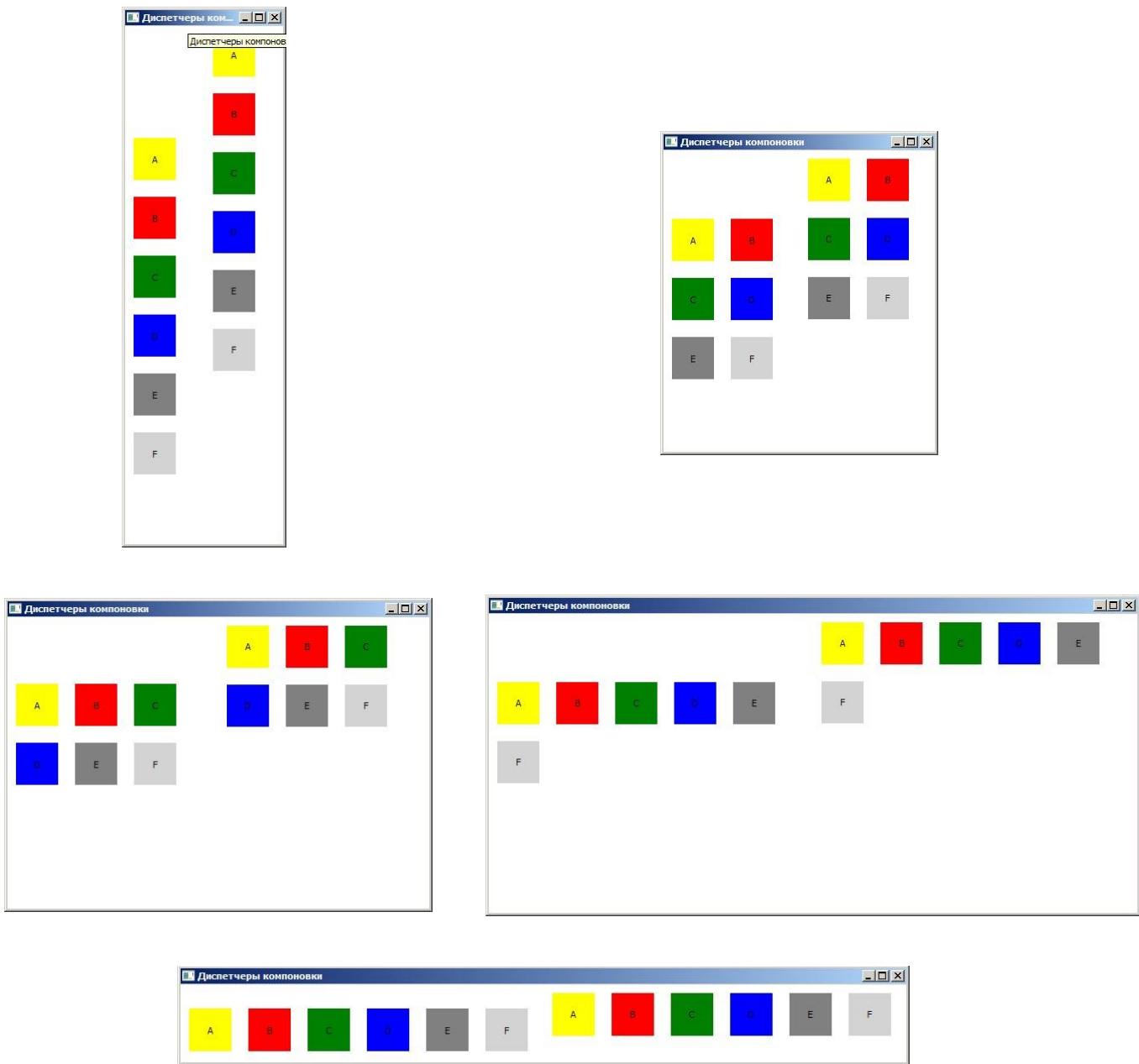
Задание

Разработать приложение WPF со следующим графическим интерфейсом:



Дополнительное задание

Разработать приложение WPF со следующим графическим интерфейсом:



Контрольные вопросы:

1. Какие существуют диспетчеры компоновки?
2. Для чего используются присоединяемые свойства?
3. Чем StackPanel отличается от WrapPanel и что между ними общего?
4. Какие возможности предоставляет диспетчер компоновки Canvas?

Лабораторная работа №3

Основные элементы управления WPF

Цель работы: изучить основные элементы управления WPF

Элемент управления Button (кнопка)

Представляет собой обычную кнопку.

Код XAML	Результат
<Button Click="Button_Click">Отправить запрос</Button>	
Код C#	
<pre>private void Button_Click(object sender, RoutedEventArgs e) { MessageBox.Show("Кнопка нажата"); }</pre>	

Рисунок 15 – элемент управления кнопка

Отличительные особенности:

- Событие **Click** – нажатие на кнопку. В атрибуте Click указывается название функции-обработчика этого события.
- Свойство **IsCancel**. Возможные значения: True, False. Если записано True, то кнопка будет срабатывать при нажатии на кнопку Esc в данном окне, т.е. когда пользователь хочет закрыть окно без выполнения каких-либо действий.
- Свойство **IsDefault**. Возможные значения: True, False. Если записано True, то кнопка будет срабатывать при нажатии на кнопку Enter в данном окне, но только если не выделена какая-либо другая кнопка. В отличие от приложения Windows Forms, в WPF-приложении при открытии окна не происходит автоматического выделения какого-либо элемента. Чтобы выделить первый элемент в окне, необходимо нажать кнопку Tab. Кнопка со свойством IsDefault="True" подсвечивается в окне, как будто она получила фокус. Но на самом деле кнопка не получает фокус, т.к. нажатие на клавишу «Пробел» не приводит к нажатию кнопки, а нажатие клавиши Tab приводит к выделению первого элемента на странице, а не элемента, следующего за кнопкой.

Код XAML	Результат
----------	-----------

странице, а не элемента, следующего за кнопкой.

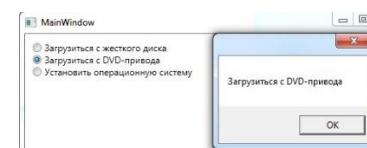
Элемент управления **ToggleButton** (переключаемая кнопка)

Представляет собой кнопку, которая может находиться в двух состояниях: нажатом и отжатом.

Отличительные особенности:

```
<RadioButton GroupName="Boot"
x:Name="RadioButton_Boot1">Загрузиться с жесткого
диска</RadioButton>
<RadioButton GroupName="Boot"
x:Name="RadioButton_Boot2">Загрузиться с
DVDпривода</RadioButton>
<RadioButton GroupName="Boot"
x:Name="RadioButton_Boot3">Установить операционную
систему</RadioButton>
```

Загрузиться с жесткого диска
 Загрузиться с DVD-привода
 Установить операционную систему



Код C#

```
...
if (RadioButton_Boot1.IsChecked == true)
    MessageBox.Show(RadioButton_Boot1.Content.ToString());
else
    if (RadioButton_Boot2.IsChecked == true)
        MessageBox.Show(RadioButton_Boot2.Content.ToString());
    else
        if (RadioButton_Boot3.IsChecked == true)
            MessageBox.Show(RadioButton_Boot3.Content.ToString());
...
}
```

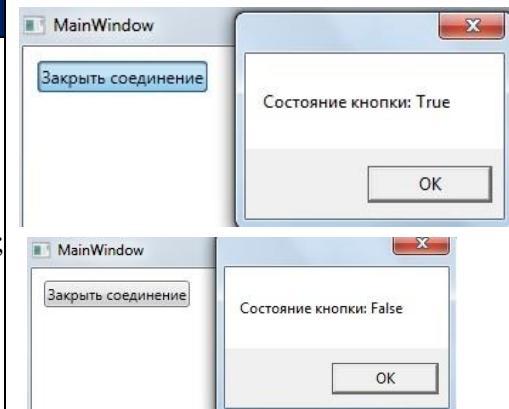
Код XAML

```
<ToggleButton Click="ToggleButton_Click">Закрыть
соединение</ToggleButton>
```

Код C#

```
private void ToggleButton_Click(object sender,
RoutedEventArgs e)
{
    MessageBox.Show("Состояние кнопки: " + (sender as
System.Windows.Controls.Primitives.ToggleButton).IsChecked);
}
```

Результат



- Событие **Click** – нажатие или отжатие кнопки. В атрибуте Click указывается название функции-обработчика этого события.

Событие **Checked** – нажатие кнопки. В атрибуте Checked указывается название функции-обработчика этого события.

- Событие **Unchecked** – отжатие кнопки. В атрибуте Unchecked указывается название функции-обработчика этого события.

- Свойство **IsChecked** – состояние кнопки. True – кнопка нажата, False – кнопка отжата.

<pre><CheckBox x:Name="CheckBox_CloseAfterComplete">Закрыть окно по завершении</CheckBox></pre> <p>Код C#</p> <pre>... if (CheckBox_CloseAfterComplete.IsChecked == true) MessageBox.Show("IsChecked == true"); ...</pre>	
<p>Рисунки 16-19 – использование CheckBox,ToggleButton</p>	

Класс CheckBox является наследником от класса ToggleButton и наследует его свойства и события.

Для обращения к элементу управления из кода программы необходимо в XAML-коде задать для него имя в атрибуте Name с префиксом x:, как это показано в примере выше. Префикс 'x:' означает пространство имен XAML, а не пространство имен WPF.

Элемент управления RadioButton (зависимый переключатель)

Рисунки 20-22 – использование RadioButton

Класс RadioButton является наследником от класса ToggleButton и наследует его свойства и события.

Отличительные особенности:

- Свойство **GroupName** – название группы зависимых переключателей. В одном окне может быть несколько групп зависимых переключателей с разными названиями групп.

Элемент управления ComboBox (выпадающий список)

Элемент ComboBox представляет собою выпадающий список, элементы которого определены с помощью элементов ComboBoxItem:

```
<ComboBox SelectedIndex="1">  
  <ComboBoxItem Content="Red" />  
  <ComboBoxItem Content="Green" />  
  <ComboBoxItem Content="Blue" />  
</ComboBox>
```

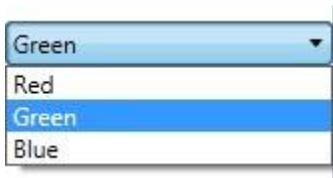


Рисунок 23 – использование ComboBox

В качестве содержимого элементов выпадающего списка можно задавать не только текст, но и другие элементы, например эллипс или прямоугольник.

Элемент управления ListBox (список)

Элемент ListBox представляет собою список, элементы которого определены с помощью элементов ListBoxItem:

```
<ListBox SelectedIndex="1">
  <ListBoxItem Content="Red" />
  <ListBoxItem Content="Green" />
  <ListBoxItem Content="Blue" />
</ListBox>
```



Рисунок 24 – использование ListBox

В качестве содержимого элементов списка можно задавать не только текст, но и другие элементы.

После заполнения элемента управления ComboBox (или ListBox) есть три способа определен выбранного в них элемента. Во-первых, если необходимо найти числовой индекс выбранного элемента, необходимо использовать свойство SelectedIndex (отсчет начинается с 0; -1 означает отсутствие выбора). Во вторых, если требуется получить объект, выбранный внутри списка, то используется свойство SelectedItem. В-третьих, SelectedValue позволяет получить значение выбранного объекта.

Элемент управления Slider

Элемент Slider представляет собою ползунок с минимальным значением Minimum, максимальным значением Maximum и текущим значением Value.

```
<Slider Height="25" Width="100" Minimum="1" Maximum="100"
Value="20" />
```



Рисунок 25 – использование Slider

Меню

Меню в WPF представлено классом `Menu`, который может включать в себя набор объектов `MenuItem`. Каждый объект `MenuItem` в свою очередь может включать в себя другие объекты `MenuItem` и объекты `Separator` (разделитель).

Пример элемента `Menu`:

```
<Menu Background="White" BorderBrush="Navy" BorderThickness="1">
<MenuItem Header="_Файл">
    <MenuItem Header="_Открыть" />
    <MenuItem Header="_Сохранить" />
    <Separator />
    <MenuItem Header="_Закрыть" />
</MenuItem>
    <MenuItem Header="_О программе" />
</Menu>
```

Знак подчеркивания в названиях пунктов меню указывает «горячие» клавиши для доступа к этим пунктам меню.

Пример работы приложения:

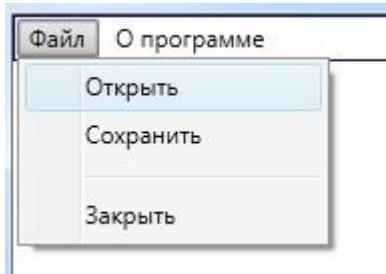


Рисунок 26 – использование `Menu`

Элемент `MenuItem` может содержать и другие элементы управления, например зависимые (`RadioButton`) и независимые (`CheckBox`) переключатели:

```
<CheckBox Content="Предупреждать о несохраненных данных при закрытии" />
<RadioButton GroupName="codepage" Content="Windows-1251" />
<RadioButton GroupName="codepage" Content="Koi8-r" />
<RadioButton GroupName="codepage" Content="UTF-8" />
```

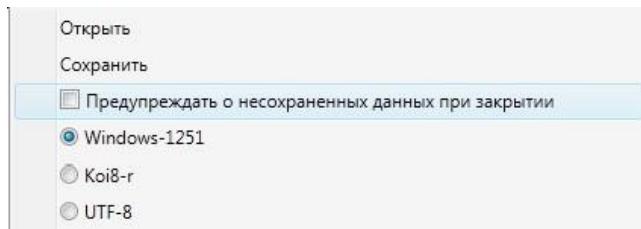


Рисунок 27 – использование `MenuItem`

Панель инструментов

Панель инструментов в WPF представлена классом ToolBar, который в качестве содержимого может включать в себя коллекцию любых других элементов. Панели инструментов обычно используются как альтернативный способ активизации пунктов меню.

Пример элемента ToolBar:

```
<ToolBar>
    <Button>
        <Image Source="open.png"></Image>
    </Button>
    <Separator/>
    <Button>
        <Image Source="http://www.readyicons.com/IconSets/Sky_Light_%28Basic%29/48x48-save.png"></Image>
    </Button>
</ToolBar>
```

Пример работы приложения:



Рисунок 28 – использование ToolBar

Кнопки содержат элементы Image. Первый элемент Image получает данные из файла open.png, включенного в проект. Второй элемент Image получает данные с веб-сайта по протоколу HTTP. Другие изображения можно выбрать, открыв в браузере адрес http://www.readyicons.com/IconSets/Sky_Light_%28Basic%29/

Для создания нескольких панелей инструментов элементы ToolBar необходимо поместить в элемент ToolBarTray.

Строка состояния

Строка состояния в WPF представлена классом StatusBar, который в качестве содержимого может включать в себя коллекцию любых других элементов, в том числе StatusBarItem.

Пример элемента StatusBar:

```
<StatusBar DockPanel.Dock="Bottom">
    <TextBlock Text="Сохранение документа ... " />
    <StatusBarItem HorizontalAlignment="Right" >
```

```
<TextBlock Text="Подключение к БД: ОК" />
</StatusBarItem>
</StatusBar>
```

Пример работы приложения:



Рисунок 29 – использование StatusBar

Элемент TextBlock может применяться для отображения текста с добавлением форматирования: полужирный текст, подчеркнутый текст, разрывы строк и т.д.

Элемент управления InkCanvas

Элемент управления InkCanvas позволяет рисовать и редактировать линии с помощью мыши или пера. Размеры элемента управления можно задать с помощью свойств Width и Height. Свойства пера (цвет, ширину и высоту) можно настроить с помощью свойства DefaultDrawingAttributes:

```
<InkCanvas>
<InkCanvas.DefaultDrawingAttributes>
  <DrawingAttributes Color="Red" Height="10" Width="1"/>
</InkCanvas.DefaultDrawingAttributes>
</InkCanvas>
```

Результат выполнения данного участка программы:

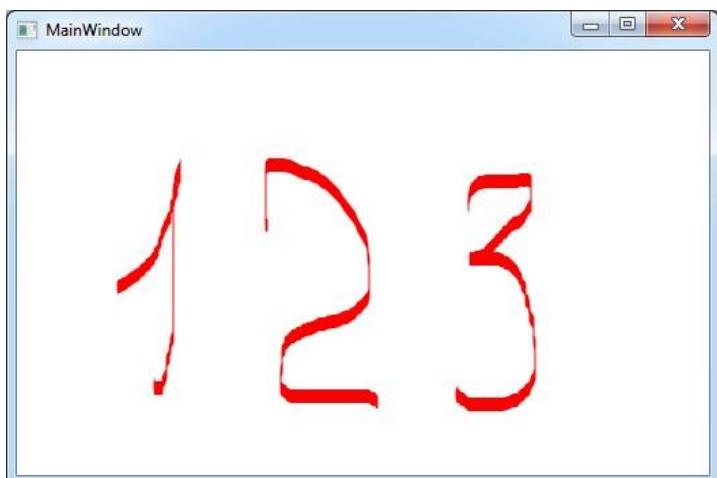


Рисунок 30 – использование InkCanvas

Свойство EditingMode позволяет настроить режим редактирования: рисование (Ink), выбор и редактирование фигур (Select), удаление по точкам (EraseByPoint) и удаление фигур (EraseByStroke).

Обработчики событий

Для добавления обработчика для какого-либо события объекта необходимо в открывающем теге элемента написать имя события и через знак «==» имя функции-обработчика, либо выбрать команду «Новый обработчик события»:



При выборе команды «Новый обработчик события» в CS-файле, относящемся к XAML-файлу, будет добавлена соответствующая функция:

```
private void MenuItem_Click(object sender, RoutedEventArgs e)
{
}
```

В обработчике можно обратиться по имени к любому объекту, для которого в XAML-файле было определено имя с помощью атрибута Name или x:Name:

```
<MenuItem Name="mi_open" Header="_Открыть" Click="MenuItem_Click" />
```

```
private void MenuItem_Click(object sender, RoutedEventArgs e)
{
    mi_open.Background = Brushes.LightGreen;
}
```

С помощью объекта sender, переданного в качестве параметра, можно получить доступ к элементу управления, для которого возникло обрабатываемое событие, даже в случае, если для него не задано имя:

```
private void CheckBox_Click(object sender, RoutedEventArgs e)
{
    ((FrameworkElement)sender).Visibility = System.Windows.Visibility.Hidden;      }
private void CheckBox_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show(((CheckBox)sender).IsChecked.ToString());
}
```

В первом примере объект sender был приведен к базовому классу FrameworkElement для доступа к базовым свойствам, присущим всем элементам управления. Во втором случае объект sender был приведен к классу CheckBox для доступа к специфическим свойствам данного элемента управления.

Если для нескольких элементов управления определен один обработчик какого-либо события, то для определения выбранного элемента управления в коде обработчика можно использовать свойство Tag, доступное для всех элементов управления:

```
private void MenuItem_Click(object sender, RoutedEventArgs e)
{
    if (((FrameworkElement)sender).Tag.ToString() == "open") MessageBox.Show("Выбрана команда
'Открыть'); else
    if (((FrameworkElement)sender).Tag.ToString() == "save") MessageBox.Show("Выбрана команда 'Сохранить');");
}
```

Наиболее часто используемые события:

Click	Происходит при нажатии на элемент управления
MouseMove	Происходит, когда указатель мыши совершает движение по этому элементу
MouseEnter	Происходит, когда указатель мыши входит в границы данного элемента
MouseLeave	Происходит, когда указатель мыши покидает границы данного элемента
MouseDown	Происходит при нажатии кнопки мыши, если указатель мыши находится на элементе
MouseUp	Происходит, когда кнопка мыши отпускается на элементе
MouseWheel	Происходит при прокрутке пользователем колесика мыши, если указатель мыши находится на элементе
KeyDown	Происходит при нажатии клавиши, если элемент имеет фокус
KeyUp	Происходит при отжатии клавиши, если элемент имеет фокус

Задание 1

Разработать WPF-приложение с меню, панелью инструментов и строкой состояния. С помощью пунктов меню пользователь может изменять цвет фона окна, получить информацию о разработчике, а также закрыть окно. Кнопки панели инструментов дублируют команды меню. При наведении на пункты меню или кнопки панели инструментов в строке состояния отображается информация об этих элементах управлении

Задание 2

Разработать WPF-приложение «Графический редактор» с выпадающим списком для выбора цвета кисти, ползунком для выбора размеров кисти и зависимыми переключателями для выбора режима работы: «рисование», «редактирование», «удаление».

Дополнительное задание

Организовать сохранение рисунка Задания 2 в файл.

Лабораторная работа №4

«Привязка данных в WPF»

Цель работы: научиться использовать механизм привязки данных в WPF.

Привязка данных (data binding) в графической системе WPF представляет собою отношение, которое сообщает WPF о необходимости извлечения данных из свойства исходного объекта (Source) и

использования её для задания значения некоторого свойства целевого объекта (Target) (и, в некоторых случаях, наоборот).

Объектом-источником может быть как элемент WPF, так и объект ADO.NET или пользовательский объект, хранящий данные. В данной лабораторной работе рассматривается связывание элементов управления WPF.

Рассмотрим пример приложения из двух элементов управления: ползунка (Slider) и текстового блока (TextBlock). При изменении положения ползунка размер шрифта текстового блока должен меняться. Такое поведение можно реализовать за счет обработки события изменения положения ползунка ValueChanged:

Пример 1 Код

XAML

```
<Slider Minimum="8" Maximum="30" ValueChanged="Slider_ValueChanged"></Slider>
<TextBlock x:Name="Message" FontSize="20">
    Пример WPF-приложения для демонстрации привязки данных
</TextBlock>
```

Код C#

```
private void Slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (Message != null)
        Message.FontSize = ((Slider)sender).Value;
}
```

Результат

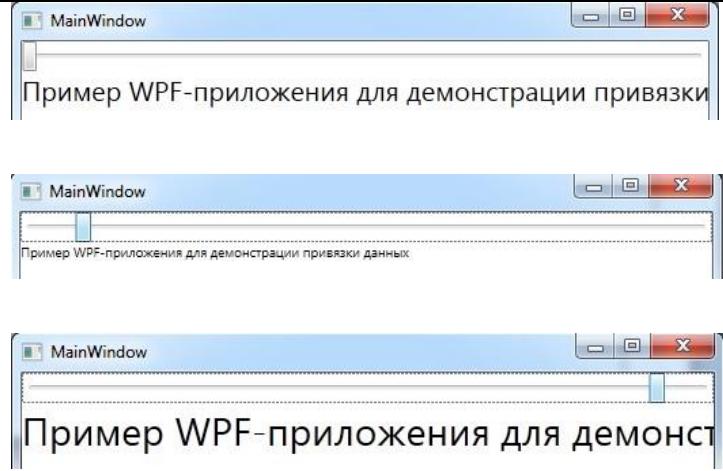


Рисунок 31 – демонстрация привязки данных

Как видно из исходного кода, возникает необходимость проверки существования объекта Message, т.к. первый вызов обработчика Slider_ValueChanged происходит в момент обработки элемента Slider XAMLфайла,

когда элемент `TextBlock` еще не обработан и, соответственно, объект `Message` еще не создан. Второй проблемой является несоответствие начального значения ползунка и начального размера шрифта.

Для решения поставленной задачи с помощью привязки данных, необходимо указать в качестве значения свойства `FontSize` текстового блока следующее выражение привязки:

```
{Binding ElementName=SliderFontSize, Path=Value}
```

Выражение привязки данных задается в виде расширения разметки XAML в фигурных скобках.

Составляющие выражения привязки:

Binding – означает, что будет создан объект класса `System.Windows.Data.Binding`

- имя исходного объекта,

Пример 2 Код

XAML

```
<Slider Minimum="8" Maximum="30" x:Name="SliderFontSize"></Slider>
<TextBlock x:Name="Message" FontSize="{Binding ElementName=SliderFontSize, Path=Value}">
    Пример WPF-приложения для демонстрации привязки данных
</TextBlock>
```

Результат

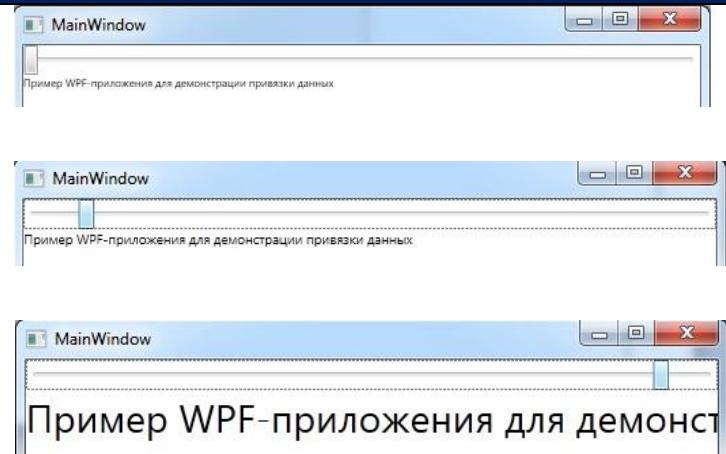


Рисунок 32 – использование привязки

Path – имя свойства (или путь до свойства) исходного объекта. Пример пути до свойства: `Background.Opacity`

В данном примере отсутствуют проблемы, обнаруженные в предыдущем примере. Начальные значения связанных свойств будут согласованы даже в том случае, если элемент `TextBlock` будет предшествовать элементу `Slider`.

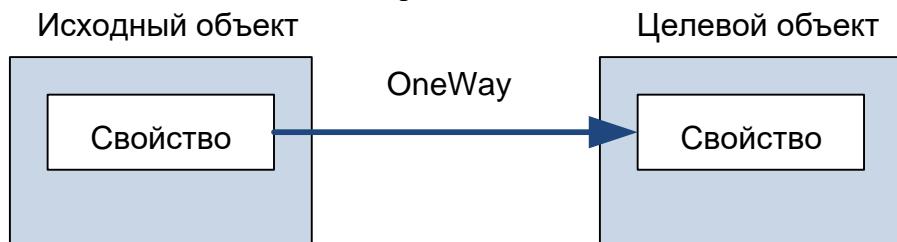
Задание 1

Проверьте реакцию среды разработки на неверные значения параметров ElementName и Path.
Проанализируйте сообщения, которые выводятся в окне вывода (Вид → Вывод) при построении и при запуске приложения.

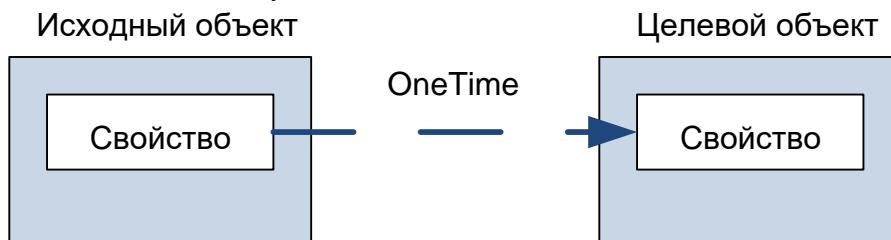
Режимы привязки

В выражении привязки с помощью параметра Mode можно задать одно из следующих пяти значений режима привязки:

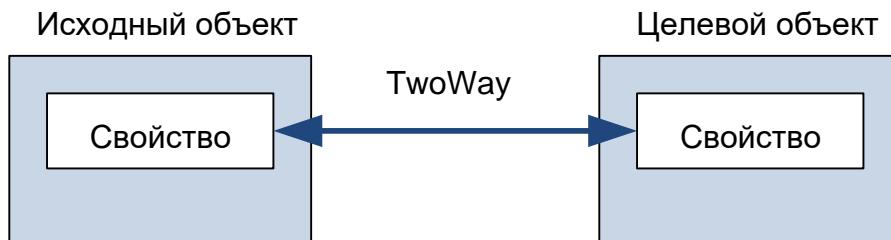
- 1) OneWay – целевое свойство обновляется при изменении исходного свойства.



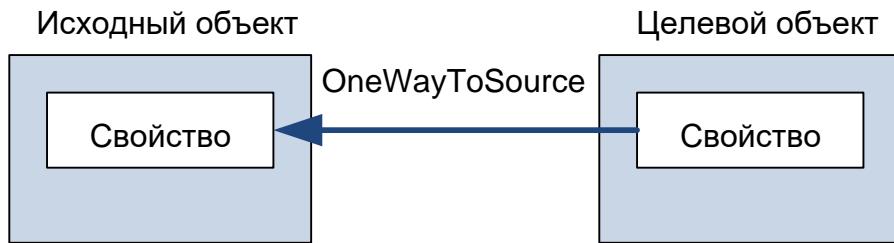
- 1) OneTime – первоначально значение исходного свойства копируется в целевое свойство, но дальнейшие изменения исходного свойства не учитываются.



- 2) TwoWay - целевое свойство обновляется при изменении исходного свойства, исходное свойство обновляется при изменении целевое свойства.



- 3) OneWayToSource – исходное свойство обновляется при изменении целевое свойства.



- 4) Default – значение по умолчанию. Если целевое свойство устанавливается пользователем (например, `TextBox.Text`, `Slider.Value`, `CheckBox.IsChecked`, ...), то это `TwoWay`, в остальных случаях – это `OneWay`.

Пример выражения привязки с параметром Mode: `{Binding ElementName=slider1, Path=Value, Mode=OneTime}`

Задание 2

Запустите приложение со следующим XAML-кодом:

```
<TextBox x:Name="t1" />
<TextBox x:Name="t2" Text="{ Binding ElementName=t1, Path=Text }" />
<Slider x:Name="slider1" />
<Slider x:Name="slider2" Value="{ Binding ElementName=slider1, Path=Value }" />
```

Определите различие в поведении полей t1 и t2 и модифицируйте код, чтобы устранить это различие.

Задание 3

Дополните пример №2 текстовым полем ввода `TextBox`, в котором пользователь может ввести размер шрифта, и задайте выражения привязки таким образом, чтобы значение ползунка, текст текстового поля и размер шрифта текстового блока соответствовали друг другу.

Задание 4

Модифицируйте приложения, разработанные в предыдущей лабораторной работе: удалите как можно больше обработчиков событий и реализуйте ту же функциональность приложения с помощью привязки данных.

Подсказки:

Свойство EditingMode (тип данных InkCanvasEditingMode) элемента управления InkCanvas нельзя напрямую связать с текстовым свойством выпадающего списка ComboBox или списка ListBox, т.к. в этом случае будет несовпадение типов. Для привязки данных необходимо, чтобы тип элементов списка совпадал с типом свойства EditingMode. Для этой цели необходимо добавить в ресурсы окна приложения (элемент Windows.Resources) массив (элемент x:Array) элементов типа InkCanvasEditingMode (атрибут x>Type), данному ресурсу необходимо задать ключ (атрибут x:Key), который необходимо указать в свойстве ItemSource списка ListBox или выпадающего списка ComboBox. В этом случае можно будет осуществить привязку данных между свойством EditingMode и выделенным элементом списка:

```
<Window x:Class="WpfApplication1.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
<Window.Resources>
    <x:Array x:Key="MyEditingModes" x>Type="{x:Type InkCanvasEditingMode}">
        <x:Static Member="InkCanvasEditingMode.Ink"/>
        <x:Static Member="InkCanvasEditingMode.Select"/>
        <x:Static Member="InkCanvasEditingMode.EraseByPoint"/>
        <x:Static Member="InkCanvasEditingMode.EraseByStroke"/>
    </x:Array>
</Window.Resources>
<StackPanel>
    <InkCanvas EditingMode="{Binding ElementName=lbEditingModes, Path=SelectedValue}" />
    <ListBox x:Name="lbEditingModes" ItemsSource="{StaticResource MyEditingModes}" />
</StackPanel>
</Window>
```

Аналогичным образом можно задать привязку данных между свойством DefaultDrawingAttributes и выделенным элементом списка (в данном случае массив x:Array будет содержать элементы типа DrawingAttributes):

```
<Window x:Class="WpfApplication1.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
<Window.Resources>
    <x:Array x:Key="MyDrawingAttributes" x>Type="{x:Type DrawingAttributes}">
        <DrawingAttributes Color="Red" Width="3" Height="3"/>
        <DrawingAttributes Color="Green" Width="10" Height="10"/>
        <DrawingAttributes Color="Blue" Width="15" Height="15"/>
    </x:Array>
</Window.Resources>
<StackPanel>
    <InkCanvas DefaultDrawingAttributes="{Binding ElementName=lbColors, Path=SelectedValue}" />
    <ListBox x:Name="lbColors" ItemsSource="{StaticResource MyDrawingAttributes}" />
```

Лабораторная работа №5

«Использование стилей в WPF-приложениях»

Цель работы: научиться использовать стили в WPF-приложениях

Стили WPF позволяют определять внешний вид и поведение для группы элементов управления.
Рассмотрим пример WPF-приложения:

Пример 1 Код

XAML

```
<StackPanel Orientation="Horizontal" VerticalAlignment="Top">
    <Button Background="DarkBlue" Foreground="White" FontFamily="Verdana" Padding="5"
Margin="5">Открыть</Button>
    <Button Background="DarkBlue" Foreground="White" FontFamily="Verdana" Padding="5"
Margin="5">Обработать</Button>
    <Button Background="DarkBlue" Foreground="White" FontFamily="Verdana" Padding="5"
Margin="5">Сохранить</Button>
    <Button Padding="5" Margin="5">Закрыть</Button>
</StackPanel>
```

Результат

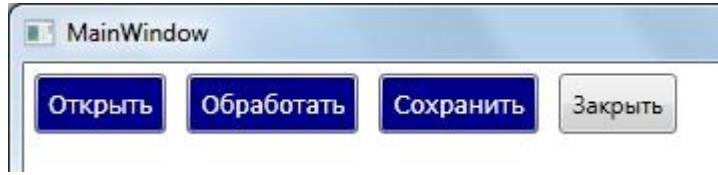


Рисунок 33 – пример использования стилей

Для трех элементов управления повторяются одни и те же атрибуты с одинаковыми значениями. Как при программировании наличие повторяющегося кода является плохим стилем, так и при разработке WPF-интерфейса повторение участков XAML-кода не приветствуется. В данном случае правильным решением является определение внешнего вида кнопки с помощью стиля и задание этого стиля для трех кнопок. По функциональности стили похожи на каскадные таблицы стилей CSS для HTML-файлов. Обычно стили, как и другие ресурсы приложения, определяются в ресурсах окна:

```
<Window.Resources>
```

```
...
```

```
<Style>
```

```
</Style>
```

```
...
```

</Window.Resources>

Свойство Resources объявлено для класса FrameworkElement, поэтому ресурсы можно объявить практически для любого элемента управления:

<StackPanel.Resources>

```
...
<Style>
```

```
</Style>
...
</StackPanel.Resources>
```

<Button.Resources>

```
...
<Style>
```

```
</Style>
...
</Button.Resources>
```

Область действия стиля, объявленного в ресурсах какого-либо элемента управления, распространяется только на этот элемент управления и его дочерние элементы управления. Следует учесть, что статический ресурс должен быть определен в коде разметки **перед** ссылкой на него.

Стиль определяется с помощью элемента Style и может использоваться для определения:

- значений атрибутов;
- обработчиков событий;
- триггеров**, меняющих атрибуты элемента управления при возникновении каких-либо событий или при изменении каких-либо свойств;
- шаблонов**, переопределяющих внешний вид элементов управления.

Ключевые свойства, определенные в классе Style:

- TargetType – тип элемента, для которого определяется данный стиль;
- BasedOn – родительский стиль (позволяет задавать иерархические стили);
- Setters – коллекция объектов Setter или EventSetter, которые предназначены для установления значений свойств и обработчиков событий;
- Triggers – коллекция триггеров;
- Resources – коллекция ресурсов, которые необходимо использовать с ресурсами.

Объект Setter определяет значение одного свойства элемента управления:

```
<Setter Property="НАЗВАНИЕ_СВОЙСТВА" Value="ЗНАЧЕНИЕ" />
```

Рассмотрим модифицированную версию примера 1 с использованием стилей:

Пример 2 Код

XAML

```
<Window.Resources>
<Style TargetType="Button">
    <Style.Setters>
        <Setter Property="Background" Value="DarkBlue" />
        <Setter Property="Foreground" Value="White" />
        <Setter Property="FontFamily" Value="Verdana" />
        <Setter Property="Padding" Value="5" />
        <Setter Property="Margin" Value="5" />
    </Style.Setters>
</Style>
</Window.Resources>
<StackPanel Orientation="Horizontal" VerticalAlignment="Top">
    <Button>Открыть</Button>
    <Button>Обработать</Button>
    <Button>Сохранить</Button>
    <Button Padding="5" Margin="5">Закрыть</Button>
</StackPanel>
```

Результат

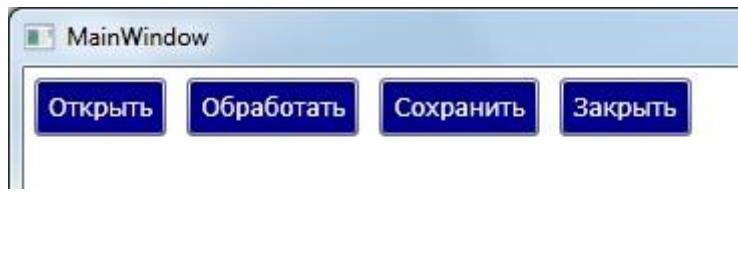


Рисунок 34 – пример использования стилей

Допускается не указывать элемент Style.Setters:

```
<Style TargetType="Button">
    <Setter Property="Background" Value="DarkBlue" />
    <Setter Property="Foreground" Value="White" />
    <Setter Property="FontFamily" Value="Verdana" />
    <Setter Property="Padding" Value="5" />
```

```
<Setter Property="Margin" Value="5" />
</Style>
```

В данном примере стиль был применен ко всем кнопкам окна (тип элементов управления определен в атрибуте TargetType). Если для элемента Style определить атрибут x:Key с именем стиля, то данный стиль будет определен только к тем кнопкам, для которых указано имя стиля в атрибуте Style с помощью расширения разметки StaticResource:

```
Style="{StaticResource ResourceKey=DocButton}"
```

Пример стиля, который применяется к трем кнопкам и не применяется к кнопке «Закрыть».

Пример 3 Код

XAML

```
<Window.Resources>
  <Style TargetType="Button" x:Key="DocButton">
    <Setter Property="Background" Value="DarkBlue" />
    <Setter Property="Foreground" Value="White" />
    <Setter Property="FontFamily" Value="Verdana" />
    <Setter Property="Padding" Value="5" />
    <Setter Property="Margin" Value="5" />
  </Style>
</Window.Resources>
<StackPanel Orientation="Horizontal" VerticalAlignment="Top">
  <Button Style="{ StaticResource ResourceKey=DocButton }">Открыть</Button>
  <Button Style="{ StaticResource ResourceKey=DocButton }">Обработать</Button>
  <Button Style="{ StaticResource ResourceKey=DocButton }">Сохранить</Button>
  <Button Padding="5" Margin="5">Закрыть</Button>
</StackPanel>
```

Результат

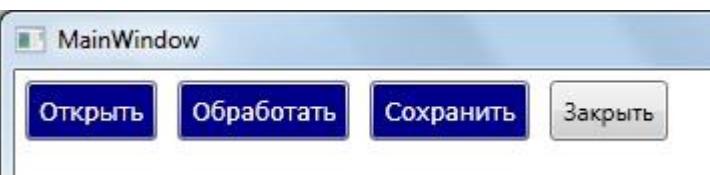


Рисунок 35 – пример использования стилей

Допускается вместо `{StaticResource ResourceKey=DocButton}` указывать `{StaticResource DocButton}`, т.к. `ResourceKey` является единственным параметром для расширения разметки `StaticResource`.

Задание 1

Проверьте, какое значение имеет больший приоритет: значение свойства, указанное в стиле, или значение атрибута элемента.

Свойства `BasedOn` класса `Style` позволяет определять иерархические стили. В этом свойстве с помощью расширения разметки `StaticResource` указывается родительский стиль. Дочерний стиль наследует все свойства родительского стиля, которые он может дополнить или переопределить. Пример определения дочернего стиля `ActiveDocButton` на основе родительского стиля `DocButton`.

Объект `EventSetter` определяет имя функции-обработчика для события:

`<EventSetter Event="НАЗВАНИЕ_СОБЫТИЯ" Handler="ИМЯ_ФУНКЦИИ" />`

Пример задания одного обработчика для всех кнопок окна:

Пример 5 Код

XAML

```
<Window.Resources>
  <Style TargetType="Button">
    <Style.Setters>
      <Setter Property="Margin" Value="5" />
      <EventSetter Event="Click" Handler="Button_Click" />
    </Style.Setters>
  </Style>
</Window.Resources>
<StackPanel Orientation="Horizontal" VerticalAlignment="Top">
  <Button>Открыть</Button>
  <Button>Обработать</Button>
  <Button>Сохранить</Button>
</StackPanel>
```

Код C#

```
private void Button_Click(object sender, RoutedEventArgs e) {
  MessageBox.Show("Button is clicked");
}
```

Результат

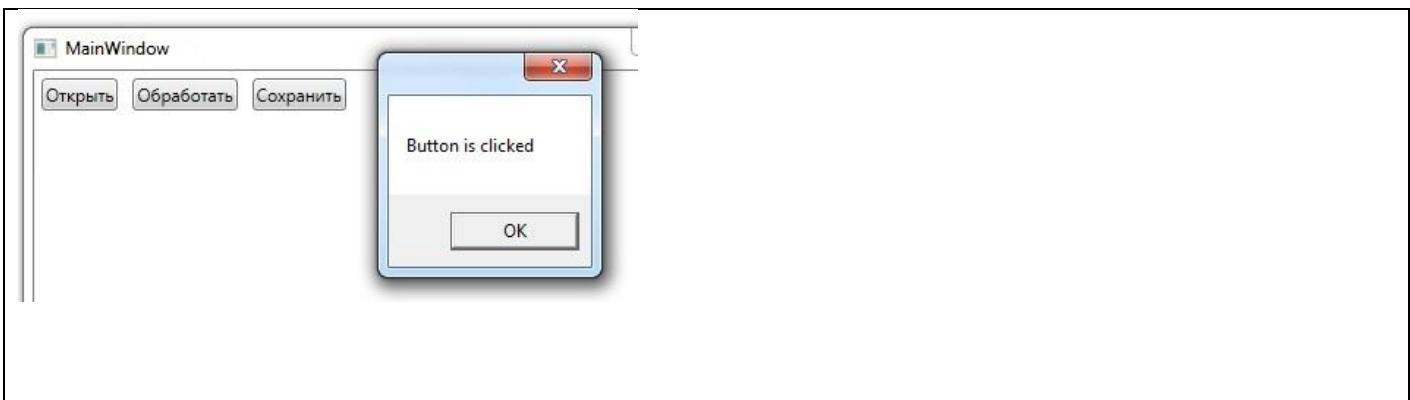


Рисунок 36 – использование стилей

Задание 2

Модифицируйте WPF-приложение, разработанное в 3-й лабораторной работе: используйте стили для однотипных элементов управления.

Задание 3

Разработайте приложение MultiEdit для одновременной работы с несколькими текстами. Окно должно быть разделено на две части с одинаковыми градиентами. В каждой части окна должно быть несколько многострочных текстовых полей: одно из них большого размера с крупным шрифтом, а остальные маленькоего размера с мелким шрифтом. То текстовое окно, в котором пользователь набирает текст, должно быть большим, остальные текстовые поля должны быть маленькими. Внешний вид однотипных элементов управления должен определяться с помощью стилей.

Изменить стиль элемента управления в коде можно следующим образом:

```
(sender as FrameworkElement).Style = (Style)Resources["ИМЯ_СТИЛЯ"];
```

Контрольные вопросы:

- 1) Для чего используются стили?
- 2) С помощью какого элемента определяется стиль?
- 3) Свойства класса Style?
- 4) Объект Setter определяет?

Лабораторная работа №6

«Триггеры в WPF-приложениях»

Цель работы: научиться использовать триггеры в WPF приложениях

В предыдущей лабораторной работе было указано, что в стиле могут быть определены триггеры, предназначенные для изменения значений атрибутов элементов управления при возникновении какихлибо событий или при изменении каких-либо свойств. Триггеры определяются в коллекции

Triggers стиля:

```

<Style ...>
<Style.Triggers>
    <!-- Триггеры -->
</Style.Triggers>
</Style>

```

Существуют следующие виды триггеров:

- 1) **Простой триггер** (Trigger) срабатывает в случае, когда заданное свойство текущего элемента управления принимает заданное значение. При срабатывании изменяет значения свойств элемента управления или применяет анимацию с использований свойств текущего элемента управления.
- 2) **Триггер привязки** (DataTrigger) отличается от простого триггера тем, что проверяет связанное свойство другого элемента управления.
- 3) **Триггер события** (EventTrigger) срабатывает при возникновении в текущем элементе управления заданного события. При срабатывании применяет анимацию с использований свойств текущего элемента управления.
- 4) **Множественный триггер** (MultiTrigger) отличается наличием нескольких условий. Триггер срабатывает только в том случае, когда все условия выполняются. При срабатывании изменяет значения свойств элемента управления или применяет анимацию с использований свойств текущего элемента управления.
- 5) **Множественный триггер привязки** (MultiDataTrigger) отличается от множественного триггера тем, что позволяет проверять связанные свойства других элементов управления.

Простой триггер (Trigger)

Формат простого триггера:

```

<Trigger Property="СВОЙСТВО" Value="ЗНАЧЕНИЕ">
    <Trigger.Setters>
        <!-- Коллекция элементов Setter -->
    </Trigger.Setters>
</Trigger>

```

Свойство Setters можно не указывать:

```

<Trigger Property="СВОЙСТВО" Value="ЗНАЧЕНИЕ">
    <!-- Коллекция элементов Setter -->
</Trigger>

```

Триггер срабатывает, когда свойство, указанное в атрибуте Property, принимает значение, указанное в атрибуте Value. В триггере нельзя задать сложные зависимости (больше, меньше, вхождения в диапазон и т.п.). При срабатывании к текущему элементу управления применяются элементы Setter, изменяющие его свойства. Как только действие триггера прекратится (свойство, указанное в атрибуте Property, принимает значение отличное от указанного в атрибуте Value), измененные свойства возвращаются к своим первоначальным значениям.

Примеры часто используемых свойств и их значений:

Свойство	Значения	Описание
IsVisible	True, False	Является ли элемент управления видимым
.IsEnabled	True, False	Является ли элемент управления доступным
IsFocused	True, False	Имеет ли элемент логический фокус
IsMouseOver	True, False	Находится ли курсор над данным элементом
IsPressed	True, False	Активизирован ли данный элемент управления

Пример WPF-приложения, в котором при наведении курсора на кнопку размер шрифта этой кнопки увеличивается.

Пример 1 Код

XAML

```

<Window.Resources>
    <Style TargetType="Button">
        <Style.Triggers>
            <Trigger Property="IsMouseOver" Value="True">
                <Trigger.Setters>
                    <Setter Property="FontSize" Value="20" />
                </Trigger.Setters>
            </Trigger>
        </Style.Triggers>
    </Style>
</Window.Resources>
<StackPanel Orientation="Horizontal" VerticalAlignment="Top">
    <Button>Открыть</Button>
    <Button>Обработать</Button>
    <Button>Сохранить</Button>
    <Button>Закрыть</Button>
</StackPanel>

```

Результат

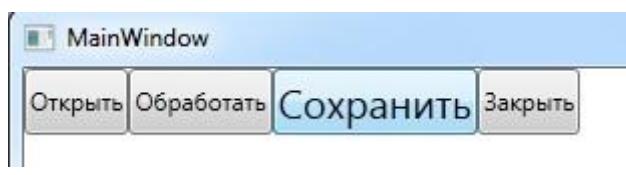


Рисунок 38 – пример использования триггеров

Задание 1

Рассмотрите случай, когда для одного и того же элемента управления срабатывают сразу несколько триггеров, устанавливающих для одного и того же свойства различные значения, и определите правило, по которому определяется приоритет применения элементов Setter этих триггеров.

Триггер привязки (DataTrigger)

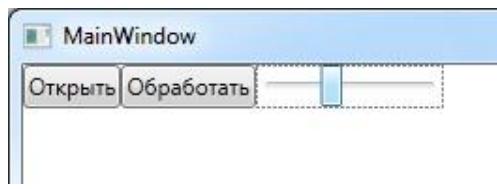
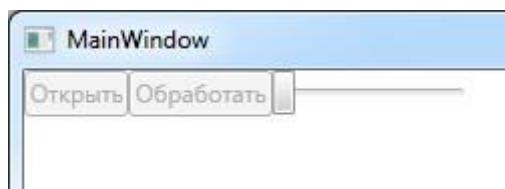
Формат простого триггера:

Пример 2 Код

XAML

```
<Window.Resources>
  <Style TargetType="Button">
    <Style.Triggers>
      <DataTrigger Binding="{ Binding ElementName=slider1, Path=Value }" Value="0">
        <Setter Property="IsEnabled" Value="False"></Setter>
      </DataTrigger>
    </Style.Triggers>
  </Style>
</Window.Resources>  <StackPanel Orientation="Horizontal" VerticalAlignment="Top">
  <Button>Открыть</Button>
  <Button>Обработать</Button>
  <Slider Height="23" x:Name="slider1" Width="100" />
</StackPanel>
```

Результат



```
<DataTrigger Binding="{ Binding ElementName=ИМЯ_СВЯЗАННОГО_ОБЪЕКТА, Path=СВОЙСТВО }"
Value="ЗНАЧЕНИЕ">
  <DataTrigger.Setters>
    <!-- Коллекция элементов Setter -->
```

```
</DataTrigger.Setters>
</DataTrigger>
```

Свойство Setters можно не указывать:

```
<DataTrigger Binding="{Binding ElementName=ИМЯ_СВЯЗАННОГО_ОБЪЕКТА, Path=СВОЙСТВО}"
Value="ЗНАЧЕНИЕ">
<!-- Коллекция элементов Setter -->
</DataTrigger>
```

Триггер срабатывает, когда свойство связанного объекта, указанное в атрибуте Binding, принимает значение, указанное в атрибуте Value. Как только действие триггера прекратится, измененные свойства возвращаются к своим первоначальным значениям.

Пример WPF-приложения, в котором сдвиге ползунка до минимального значения кнопки становятся неактивными:

Множественный триггер (MultiTrigger)

Формат множественного триггера:

```
<MultiTrigger>
  <MultiTrigger.Conditions>          <Condition
.../>
...
  <Condition .../>
</MultiTrigger.Conditions>
  <MultiTrigger.Setters>
    <!-- Коллекция элементов Setter -->
  </MultiTrigger.Setters>
</MultiTrigger>
```

Свойство Setters можно не указывать:

```
<MultiTrigger>
  <MultiTrigger.Conditions>          <Condition
.../>
...
  <Condition .../>
</MultiTrigger.Conditions>
  <!-- Коллекция элементов Setter -->
</MultiTrigger>
```

Триггер срабатывает, когда все свойства, указанные в элементах Condition, принимают соответствующие значения. Как только действие триггера прекратится, измененные свойства возвращаются к своим первоначальным значениям.

Элемент условия Condition может проверять значение свойства текущего элемента управления:

```
<Condition Property="СВОЙСТВО" Value="ЗНАЧЕНИЕ" />
```

Множественный триггер привязки (MultiDataTrigger)

Множественный триггер привязки отличается от триггера привязки тем, что элемент условия Condition может проверять как значение свойства текущего элемента управления:

<Condition Property="СВОЙСТВО" Value="ЗНАЧЕНИЕ" /> так и

значение связанного свойства другого элемента управления:

<Condition Binding="{Binding ElementName=ИМЯ_СВЯЗАННОГО_ОБЪЕКТА, Path=ИМЯ_СВОЙСТВА}" Value="ЗНАЧЕНИЕ"/>

Задание 2

Разработайте WPF-приложение с двумя многострочными текстовыми полями, кнопками «Открыть», «Очистить», «Закрыть» и выпадающим списком для задания внешнего вида текстовых полей. Задайте для текстовых полей одинаковый градиентный фон. Кнопка «Закрыть» должна быть доступна только в том случае, если в обоих текстовых полях нет текста. Задайте для кнопок различный внешний вид при наведении курсора и при нажатии на них. Внешний вид текстовых полей (тип шрифта, размер шрифта, цвет шрифта) должен меняться в зависимости от значения, выбранного в выпадающем списке.

Дополнительное задание: Реализовать задание 2 с различными типами триггеров.

Контрольные вопросы:

- 1) Что такое триггер?
- 2) Для чего используются триггеры в WPF-приложениях?
- 3) Какие существуют типы триггеров?
- 4) Отличие множественного триггера привязки от триггера привязки?

Лабораторная работа №7

«Анимация в WPF-приложениях»

Цель работы: научиться использовать анимацию в WPF приложениях

Анимация в WPF-приложениях обеспечивается постепенным изменением свойств элементов.

Типы анимации

Все классы анимации объявлены в пространстве имен System.Windows.Media.Animation. Имена классов анимации начинаются с имени типа свойства, для которого предназначена данная анимация.

Примеры:

Имя класса анимации	Описание
ByteAnimation...	Анимация, предназначенная для изменения свойства типа Byte
Int32Animation...	Анимация, предназначенная для изменения свойства типа Int32

DoubleAnimation...	Анимация, предназначенная для изменения свойства типа Double
BooleanAnimation...	Анимация, предназначенная для изменения свойства типа Boolean
ColorAnimation...	Анимация, предназначенная для изменения свойства типа Color
SizeAnimation...	Анимация, предназначенная для изменения свойства типа Size
StringAnimation...	Анимация, предназначенная для изменения свойства типа String
ThicknessAnimation...	Анимация, предназначенная для изменения свойства типа Thickness (например, для свойств Margin, Padding, BorderThickness); примеры значений: «10» - задание значения для всех сторон; «5,10,5,20» - задание значений для разных сторон: 5 для левой стороны, 10 – для верхней, 5 – для правой, 20 – для нижней
VectorAnimation...	Анимация, предназначенная для изменения свойства типа Vector

Набор классов **<Тип>Animation** предназначен для линейного изменения значения свойства от начального (или текущего) до конечного за указанный промежуток времени. Конечное значение свойства может быть задано явно, либо как приращение к начальному (или текущему) значению свойства.

Набор классов **<Тип>AnimationUsingKeyFrames** предназначен для анимации с использованием ключевых кадров. Разработчик задает набор значений для изменяемого свойства, временные характеристики и способ перехода между этими значениями.

Набор классов **<Тип>AnimationUsingPath** предназначен для изменения значения свойства в соответствии с геометрическим путем.

Класс <Тип>Animation

Класс **<Тип>Animation** включает следующие основные свойства:

- **From** – начальное значение анимируемого свойства. Если свойство не задано, то в качестве начального значения используется текущее значение свойства.
- **To** – конечное значение анимируемого свойства. Может быть не определено, если задано свойство **By**.
- **By** – смещение, прибавляемое к начальному значению свойства для получения конечного значения свойства.

После завершения действия анимации измененные свойства **не** возвращаются к первоначальным значениям. Вернуть первоначальное значение свойства можно с помощью дополнительного объекта анимации, для которого не заданы свойства **To** и **By**. Т.е. если не заданы свойства **To** и **By**, то целевым значением свойства является начальное значение, заданное до начала серии анимаций.

Примеры:

<DoubleAnimation To="10" /> – изменение свойства от текущего значения до значения 10;
<DoubleAnimation From="1" By="4" /> – изменение свойства от значения 1 до значения 5;
<DoubleAnimation By="4" /> – увеличение значения свойства на 4 от текущего значения (если вызвать анимацию несколько раз, то каждый раз значение свойства будет увеличиваться);
<DoubleAnimation /> – возвращение к первоначальному значению свойства, заданному до начала действия серии анимаций;

- **Duration** – временной интервал, за который осуществляется анимация в формате «часы:минуты:секунды».
 - Пример:
<DoubleAnimation Duration="0:1:5.5" ... > – анимация осуществляется за 1 минуту и 5.5 секунд;
- **BeginTime** – временной интервал задержки перед началом анимации в формате «часы:минуты:секунды».
- **AutoReverse** – если true, то по завершении анимации начнется обратная анимация к первоначальному состоянию; значение по умолчанию – false.
- **RepeatBehavior** – способ повторения анимации; если указано значение в формате «ЧИСЛОx» (например, «5x»), то анимация будет повторена указанное число раз; если указано значение «Forever» то анимация будет повторяться неограниченное количество раз.

Класс Storyboard – раскадровка

Элементы анимации объединяются в родительском элементе Storyboard (раскадровка), которому может быть присвоено некоторое имя:

```

<Storyboard x:Name="ButtonAnimation">
  <DoubleAnimation ... />
<ColorAnimation ... />
  ...
</Storyboard>

```

Анимируемое свойство определяется в присоединяемых свойствах Storyboard.TargetName и Storyboard.TargetProperty:

```

<Storyboard>
  <DoubleAnimation Storyboard.TargetName="ИМЯ_ЭЛЕМЕНТА"
  Storyboard.TargetProperty="ИМЯ_СВОЙСТВА" ... />
  <ColorAnimation Storyboard.TargetName="ИМЯ_ЭЛЕМЕНТА"
  Storyboard.TargetProperty="ИМЯ_СВОЙСТВА" ... />
  ...
</Storyboard>

```

Присоединяемые свойства пишутся в скобках:

```
Storyboard.TargetProperty="(Canvas.Left)"
Storyboard.TargetProperty="Background.(SolidColorBrush.Color)"
Storyboard.TargetProperty="Background.(RadialGradientBrush.GradientStops)[0].Color"
Storyboard.TargetProperty="Background.(LinearGradientBrush.GradientStops)[1].Offset"
```

Для всей раскадровки можно задать свойства AutoReverse и RepeatBehavior.

Запуск раскадровки осуществляется с помощью элемента BeginStoryboard, который соответствует не объекту, а действию. Раскадровка объявляется либо в самом элементе BeginStoryboard:

```
<BeginStoryboard>
  <S> ...
  </Storyboard>
</BeginStoryboard>
```

либо имя раскадровки указывается в атрибуте Storyboard элемента BeginStoryboard (для этого раскадровка должна быть определена ранее как ресурс с именем):

```
<BeginStoryboard Storyboard="{StaticResource ButtonStoryboard}" />
```

Анимацию можно не только запустить, но и остановить, приостановить и возобновить с помощью следующих элементов:

```
<StopStoryboard BeginStoryboardName="{StaticResource PanelStoryboard}" />
<PauseStoryboard BeginStoryboardName="{StaticResource PanelStoryboard}" />
<ResumeStoryboard BeginStoryboardName="{StaticResource PanelStoryboard}" />
```

Действия с раскадровками можно выполнять:

1) в триггере события EventTrigger при возникновении какого-либо события:

```
<EventTrigger RoutedEvent="СОБЫТИЕ">
```

Элементы BeginStoryboard, StopStoryboard, PauseStoryboard, ResumeStoryboard

```
</EventTrigger>
```

2) в других типах триггеров (Trigger, MultiTrigger, DataTrigger, MultiDataTrigger) в коллекциях EnterActions (при срабатывании триггера) и ExitActions (при прекращении действия триггера):

```
<MultiDataTrigger ...> ...
```

```
<MultiDataTrigger.EnterActions>
```

Элементы BeginStoryboard, StopStoryboard, PauseStoryboard, ResumeStoryboard, запускаемые при срабатывании триггера

```
</MultiDataTrigger.EnterActions>  
<MultiDataTrigger.ExitActions >
```

Элементы BeginStoryboard, StopStoryboard, PauseStoryboard, ResumeStoryboard, запускаемые при прекращении действия триггера

```
</MultiDataTrigger.ExitActions>  
</MultiDataTrigger>
```

Задание 1

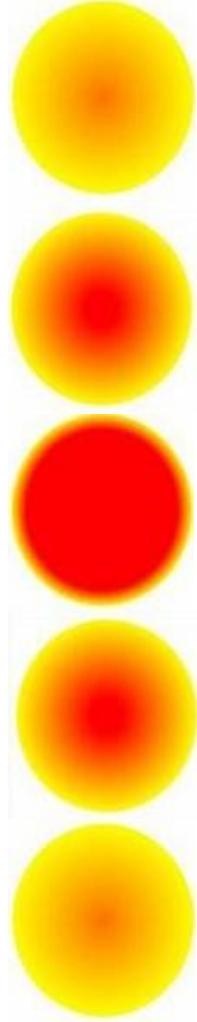
Добавьте в WPF-приложение MultiEdit, разработанное в лабораторной работе №5, эффекты анимации.

Задание 2

Разработайте WPF-приложение «Убегающая кнопка»: при наведении курсора мыши на кнопку она смещается на некоторое расстояние от курсора. Событие наведения курсора мыши – MouseEnter.

Задание 3

Разработайте WPF-приложение «Пульсар», изображающее круг, плавно меняющий свое следующей схеме:



и т.д.

Используйте элемент «Эллипс» с радиальной градиентной заливкой RadialGradientBrush для свойства Fill(заливка)

Контрольные вопросы:

- 1) Чем обеспечивается анимация в WPF-приложениях?
- 2) Какие существуют типы анимации?
- 3) Свойства типа StoryBoard?
- 4) Свойства типа Animation?

Лабораторная работа №8

«Трансформация в WPF-приложениях»

Цель работы : научиться использовать трансформацию в WPF-приложениях

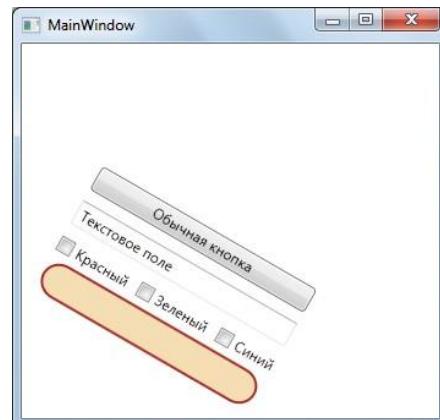
Любая часть пользовательского интерфейса WPF-приложения может быть подвергнута трансформации с помощью свойства RenderTransform. Существуют следующие виды трансформации:

- 1) **Вращение** (элемент RotateTransform) – поворачивает координатную систему на угол Angle, заданный в градусах, относительно центра трансформации. Центр трансформации может быть задан с помощью атрибутов CenterX и CenterY (0,0 – это левый верхний угол элемента управления). Положительное значение угла соответствует вращению по часовой стрелке.

Пример 1 Код XAML

```
<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
    <Button>Обычная кнопка</Button>
    <TextBox>Текстовое поле</TextBox>
    <StackPanel Orientation="Horizontal">
        <CheckBox>Красный</CheckBox>
        <CheckBox>Зеленый</CheckBox>
        <CheckBox>Синий</CheckBox>
    </StackPanel>
    <Rectangle Width="200" Height="30" Fill="Wheat" Stroke="Brown" StrokeThickness="2"
RadiusX="15" RadiusY="15" />
    <StackPanel.RenderTransform>
        <RotateTransform Angle="30" CenterX="0" CenterY="0" />
    </StackPanel.RenderTransform>
</StackPanel>
```

Результат



Без трансформации:

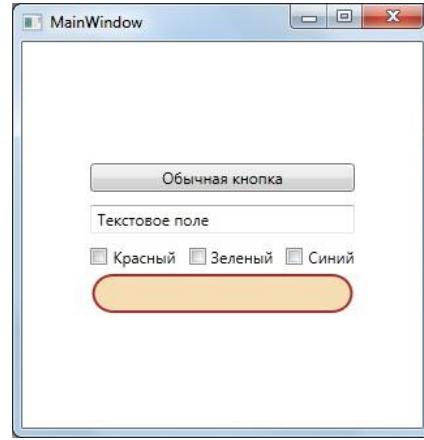


Рисунок 40 – пример трансформации вращения

- 2) **Масштабирование** (элемент ScaleTransform) – масштабирует координатную систему в большую или в меньшую сторону относительно центра трансформации. Центр трансформации может быть задан с помощью атрибутов CenterX и CenterY (0,0 – это левый верхний угол элемента управления). Коэффициент масштабирования задается в относительных единицах в атрибутах ScaleX и ScaleY для разных координатных осей. ScaleX="2" означает, что по оси абсцисс элемент управления будет растянут в два раза; ScaleX="0.5" означает, что по оси ординат элемент управления будет сжат в два раза.

Пример 2 Код XAML

```
<StackPanel.RenderTransform>
  <ScaleTransform ScaleX="2" ScaleY="0.8" CenterX="100" CenterY="100" />
</StackPanel.RenderTransform>
```

Результат

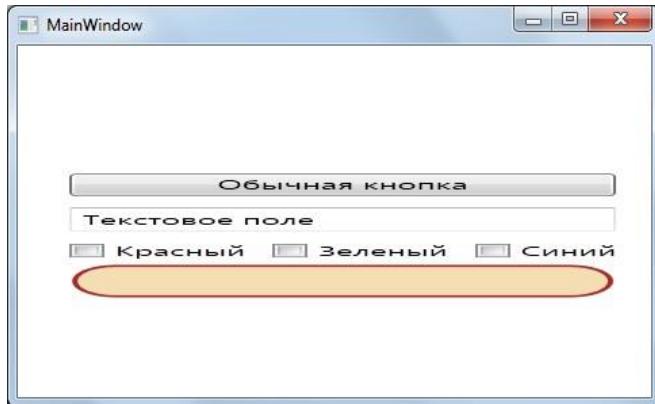


Рисунок 41 – пример трансформации масштабирования

- 3) **Сдвиг** (элемент SkewTransform) – деформирует координатную систему, наклоняя ее относительно центра трансформации на AngleX градусов по оси абсцисс и на AngleY градусов по оси ординат. Центр трансформации может быть задан с помощью атрибутов CenterX и CenterY (0,0 – это левый верхний угол элемента управления).

Пример 3 Код XAML

```
<StackPanel.RenderTransform>
  <SkewTransform AngleX="20" AngleY="20" CenterX="100" CenterY="100" />
</StackPanel.RenderTransform>
```

Результат

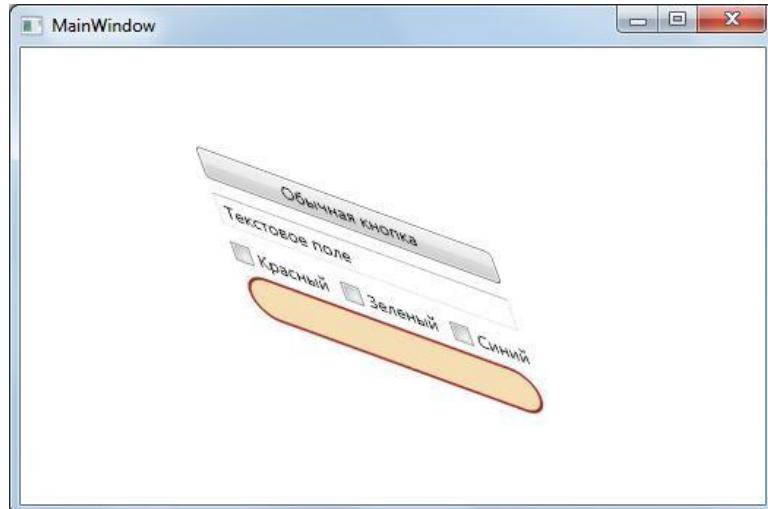


Рисунок 42 – пример трансформации сдвига

- 4) **Смещение** (элемент TranslateTransform) – смещает координатную систему на величину X по оси абсцисс и на величину Y по оси ординат.

Пример 4 Код XAML

```
<StackPanel.RenderTransform>
  <TranslateTransform X="100" Y="100" />
</StackPanel.RenderTransform>
```

Результат

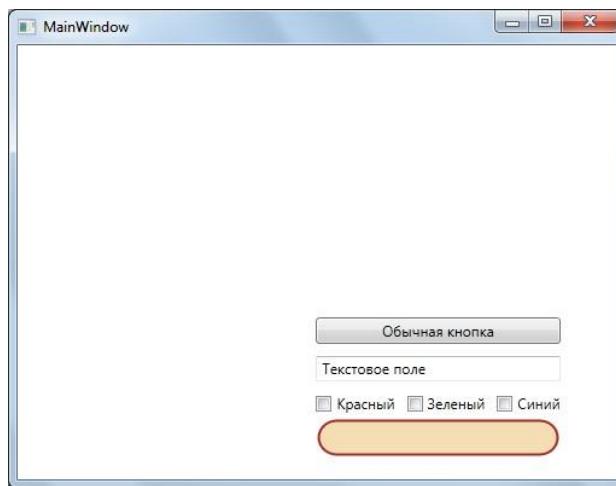


Рисунок 43 – пример трансформации смещения

- 5) **Матричное преобразование** (элемент MatrixTransform) – модифицирует координатную систему, используя матричное умножение с указанной матрицей.

Пример 5

Код XAML

```
<StackPanel.RenderTransform>
  <TransformGroup>
    <RotateTransform Angle="30" />
    <ScaleTransform ScaleX="2" ScaleY="0.8"/>
    <TranslateTransform X="-200" Y="-100" />
  </TransformGroup>
</StackPanel.RenderTransform>
```

Результат

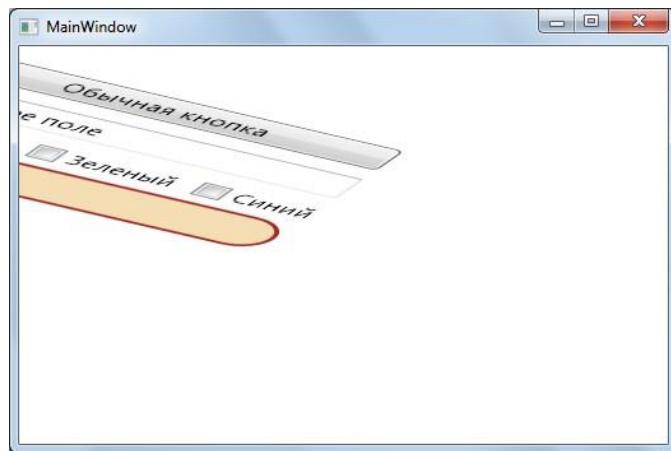


Рисунок 44 – пример трансформации матричного преобразования

Использовать атрибуты `CenterX` и `CenterY` для задания центральной точки трансформации имеет смысл в том случае, когда точно известны размеры элемента управления, подвергаемого трансформации. В том случае, когда размеры неизвестны или изменяются в процессе работы, имеет смысл задавать атрибут `RenderTransformOrigin` элемента управления (не трансформации), подвергаемого трансформации, в следующем формате

`RenderTransformOrigin="X,Y"`, где

X – относительная координата X (0 – начало элемента управления; 1 – конец элемента управления); Y – относительная координата Y (0 – начало элемента управления; 1 – конец элемента управления).

Например, `RenderTransformOrigin="0.5,0.5"` смещает центр трансформации в центр элемента управления.

Анимированные трансформации

Свойства трансформации могут быть анимированы с помощью объектов анимации. Для этого необходимо добавить в код XAML элементы трансформации (можно без атрибутов), а в элементах анимации изменять следующие свойства трансформации:

- если не используется комбинированная трансформация `TransformGroup`:

`RenderTransform.Angle`

`RenderTransform.AngleX`

`RenderTransform.AngleY`

`RenderTransform.CenterX`

`RenderTransform.CenterY`

`RenderTransform.X`

`RenderTransform.Y`

`RenderTransform.ScaleX`

`RenderTransform.ScaleY`

- если используется комбинированная трансформация `TransformGroup`:

`RenderTransform.Children[ИНДЕКС_ТРАНСФОРМАЦИИ_В_ГРУППЕ].Angle`

`RenderTransform.Children[ИНДЕКС_ТРАНСФОРМАЦИИ_В_ГРУППЕ].AngleX`

`RenderTransform.Children[ИНДЕКС_ТРАНСФОРМАЦИИ_В_ГРУППЕ].AngleY`

`RenderTransform.Children[ИНДЕКС_ТРАНСФОРМАЦИИ_В_ГРУППЕ].CenterX`

`RenderTransform.Children[ИНДЕКС_ТРАНСФОРМАЦИИ_В_ГРУППЕ].CenterY`

`RenderTransform.Children[ИНДЕКС_ТРАНСФОРМАЦИИ_В_ГРУППЕ].X`

`RenderTransform.Children[ИНДЕКС_ТРАНСФОРМАЦИИ_В_ГРУППЕ].Y`

`RenderTransform.Children[ИНДЕКС_ТРАНСФОРМАЦИИ_В_ГРУППЕ].ScaleX`

`RenderTransform.Children[ИНДЕКС_ТРАНСФОРМАЦИИ_В_ГРУППЕ].ScaleY`

Пример 6 Код XAML

```
<Button RenderTransformOrigin="0.5,0.5">
    Обычная кнопка
    <Button.RenderTransform>
        <RotateTransform />
    </Button.RenderTransform>
    <Button.Triggers>
        <EventTrigger RoutedEvent="MouseEnter">
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Angle"
From="0" To="360" Duration="0:0:1" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Button.Triggers>
</Button>
```

Результат

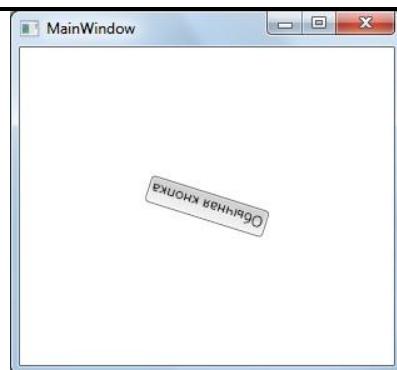


Рисунок 45 – пример анимированной трансформации

Задание 1

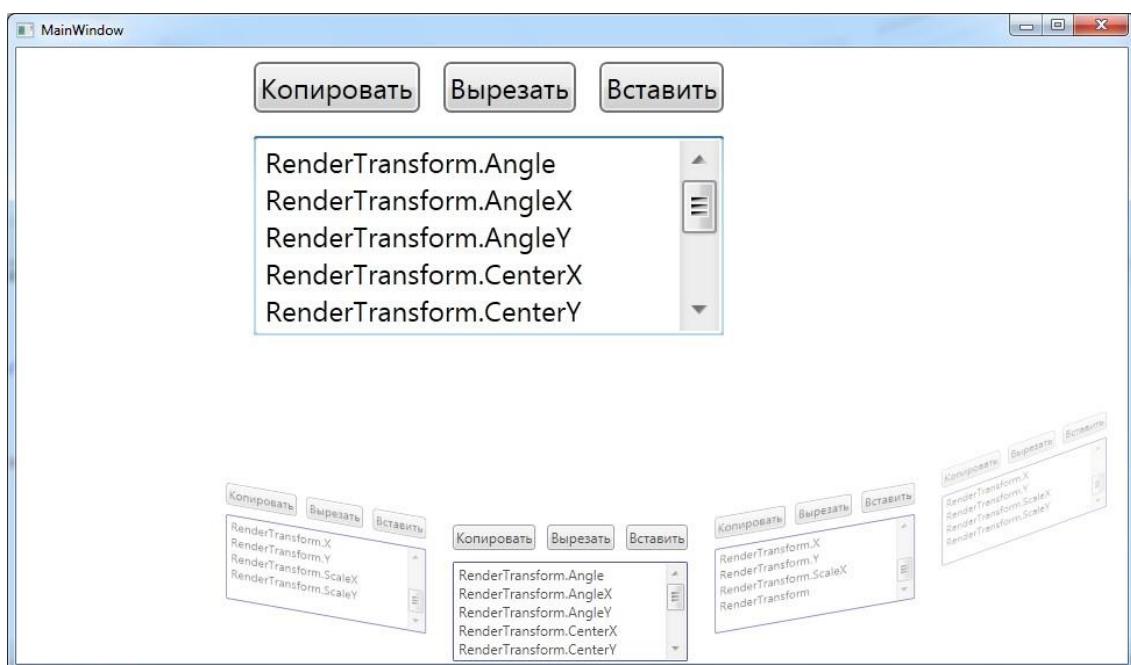
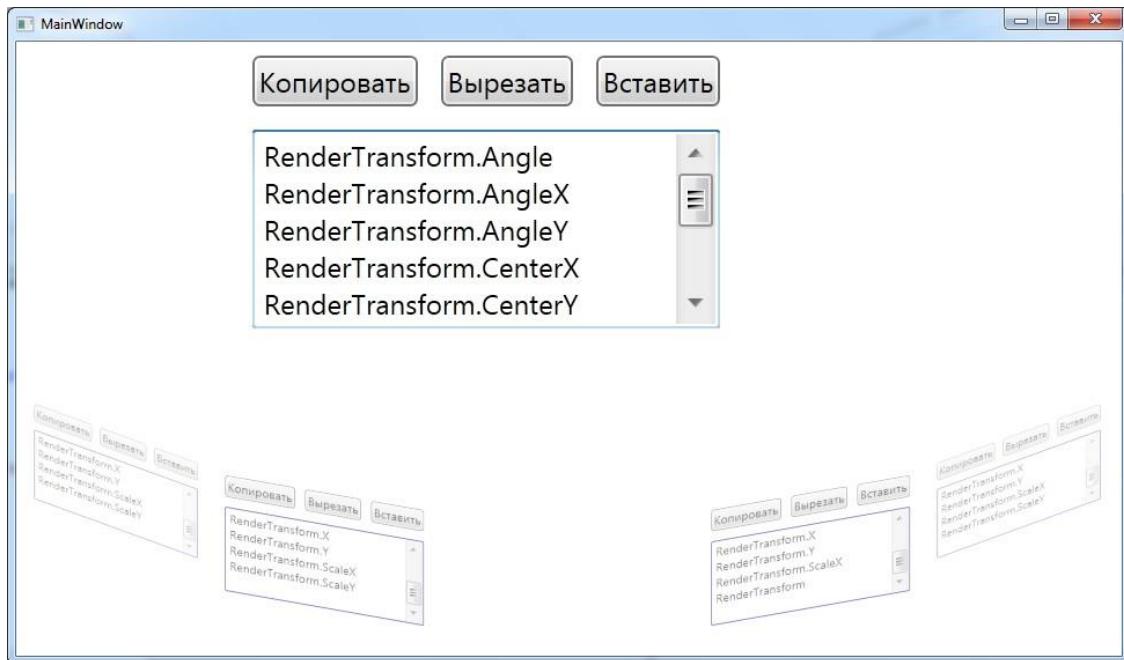
Для любого WPF-приложения с большим количеством элементов управления, разработанного в одной из предыдущих лабораторных работ, реализуйте масштабирование всего пользовательского интерфейса с помощью ползунка Slider.

Задание 2

Разработайте WPF-приложение «Текстовый редактор» в соответствии с изображениями пользовательского интерфейса, приведенными ниже. Блоки с многострочными текстовыми полями выводятся в нижней части интерфейса «полукругом». При выделении какого-либо блока (событие GotFocus) он плавно перемещается в центральную часть. При потере фокуса (событие LostFocus) блок возвращается в свое исходное состояние. Приложение не должно содержать код на языке C#.

Подсказки:

- используйте диспетчер компоновки Canvas;
- если для анимации не заданы атрибуты To и By, то целевым значением является исходное значение свойства, действующее до начала серии анимаций.



Дополнительное задание: для задания №2 применить различные типы трансформации

Контрольные вопросы:

- 1) Что позволяет трансформация в WPF приложениях?
- 2) С помощью какого свойства обеспечивается трансформация в WPF-приложениях
- 3) Какие существуют виды трансформации в WPF?
- 4) Что такое анимированные трансформации?

Лабораторная работа №9

«Использование кистей в WPF-приложениях»

Цель работы: научиться использовать кисти в WPF-приложениях

Элементы управления содержимым

Элемент управления ScrollViewer

Данный элемент управления используется в случае, когда необходимо в ограниченной области окна вывести большой объем содержимого. Может содержать только один дочерний элемент (обычно – диспетчер компоновки).

Основные свойства:

Width – ширина;

Height – высота;

HorizontalScrollBarVisibility – отображение вертикальных полос прокрутки: Visible – отображаются, Auto – отображаются при необходимости, Disabled – не отображаются и прокрутка в данном направлении невозможна, Hidden – не отображаются, но прокрутка возможна (например, с помощью клавиш управления курсором);

VerticalScrollBarVisibility – отображение горизонтальных полос прокрутки;

Пример 1 Код XAML

```
<ScrollViewer Width="200" Height="90">
    <StackPanel Orientation="Vertical">
        <Button Background="Red">Красный</Button>
        <Button Background="Green">Зеленый</Button>
    <Button Background="Blue">Синий</Button>
        <Button Background="Yellow">Желтый</Button>
        <Button Background="Brown">Коричневый</Button>
    </StackPanel>
</ScrollViewer>
```

Результат

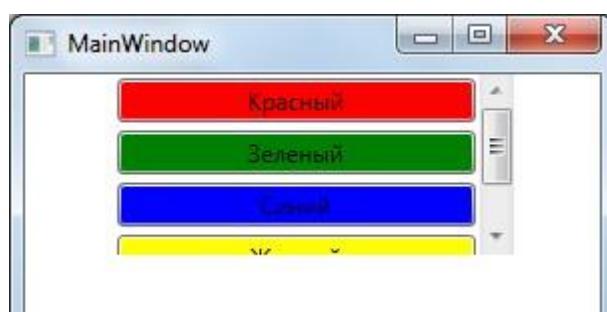


Рисунок 46 – пример работы ScrollViewer

Элемент управления Border

Данный элемент управления используется для вывода закругленной границы вокруг других элементов управления. Может содержать только один дочерний элемент (обычно – диспетчер компоновки).

Основные свойства:

BorderBrush – кисть для рисования границы (в простейшем случае – цвет сплошной линии);

BorderThickness – толщина границы;

CornerRadius – радиус закругления границы; если указано одно число, то оно используется для всех углов границы, либо можно задать значение в формате «A,B,C,D», где A – радиус закругления верхнего левого угла; B – верхнего правого, C – нижнего правого, D – нижнего левого.

Пример 2 Код XAML

```
<Border BorderBrush="Navy" BorderThickness="4" CornerRadius="20" Padding="10"
Margin="20">
<StackPanel Orientation="Vertical" Background="White">
<TextBlock>Красный</TextBlock>
<TextBlock>Зеленый</TextBlock>
<TextBlock>Синий</TextBlock>
<TextBlock>Желтый</TextBlock>
<TextBlock>Коричневый</TextBlock>
</StackPanel>
</Border>
```

Результат

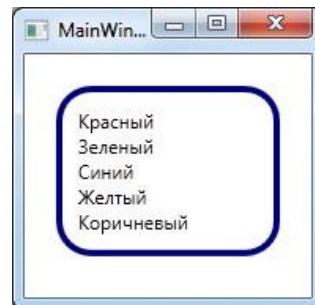


Рисунок 47 – пример работы border

Элемент управления **GroupBox**

Данный элемент управления используется для группировки взаимосвязанных элементов управления. Может содержать только один дочерний элемент (обычно – диспетчер компоновки).

Основные свойства:

Header – заголовок элемента управления. Может быть текстовой строкой, либо любым другим элементом управления.

Пример 3 Код XAML

```
<GroupBox Margin="10" Header="Цвета">
    <StackPanel Orientation="Vertical">
        <Button Background="Red">Красный</Button>
        <Button Background="Green">Зеленый</Button>
        <Button Background="Blue">Синий</Button>
        <Button Background="Yellow">Желтый</Button>
        <Button Background="Brown">Коричневый</Button>
    </StackPanel>
</GroupBox>
```

Результат

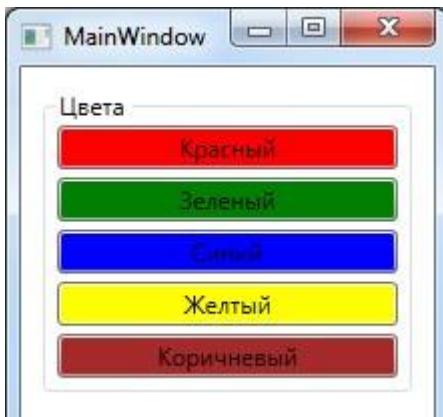


Рисунок 48 – пример работы GroupBox

Элемент управления Expander

Данный элемент управления содержит область, которую пользователь может раскрыть или скрыть, кликнув на заголовок. Может содержать только один дочерний элемент (обычно – диспетчер компоновки).

Основные свойства:

Header – заголовок элемента управления. Может быть текстовой строкой, либо любым другим элементом управления;

IsExpanded – состояние элемента управления: True – раскрыт, False – закрыт;

ExpandDirection – направление раскрытия: Down (значение по умолчанию), Up, Left, Right.

Пример 4 Код XAML

```
<Expander Header="Цвета">
    <StackPanel Orientation="Vertical">
        <Button Background="Red">Красный</Button>
        <Button Background="Green">Зеленый</Button>
        <Button Background="Blue">Синий</Button>
        <Button Background="Yellow">Желтый</Button>
        <Button Background="Brown">Коричневый</Button>
    </StackPanel>
</Expander>
```

Результат

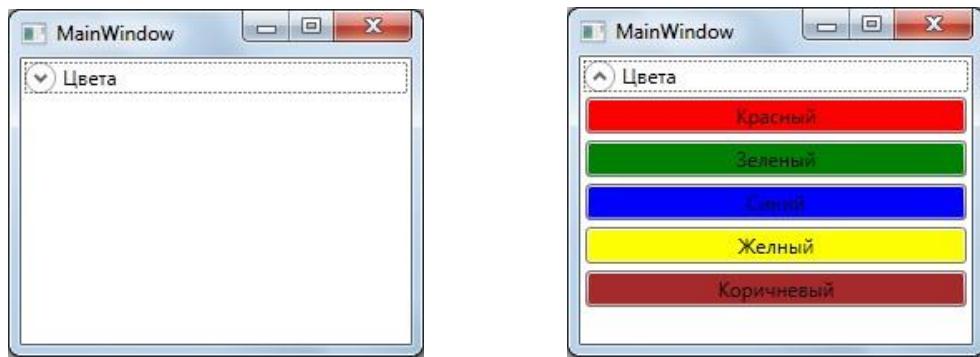


Рисунок 49 – пример работы Expander

Элемент управления TabItem

Данный элемент управления представляет собой страницу-вкладку с заголовком в элементе управления вкладками TabControl. Переключение между вкладками происходит при нажатии на заголовок вкладки. TabItem может содержать только один дочерний элемент (обычно – диспетчер компоновки).

Основные свойства:

Header – заголовок элемента управления. Может быть текстовой строчкой, либо любым другим элементом управления;

IsSelected – состояние вкладки: True – раскрыта, False – скрыта.

Пример 5 Код XAML

```
<TabControl>
    <TabItem Header="Цвет">
        <StackPanel Orientation="Vertical">
            <Button Background="Red">Красный</Button>
            <Button Background="Green">Зеленый</Button>
            <Button Background="Blue">Синий</Button>
            <Button Background="Yellow">Желтый</Button>
            <Button Background="Brown">Коричневый</Button>
        </StackPanel>
    </TabItem>
    <TabItem Header="Шрифт">
    </TabItem>
    <TabItem Header="Фон">
    </TabItem>
</TabControl>
```

Результат

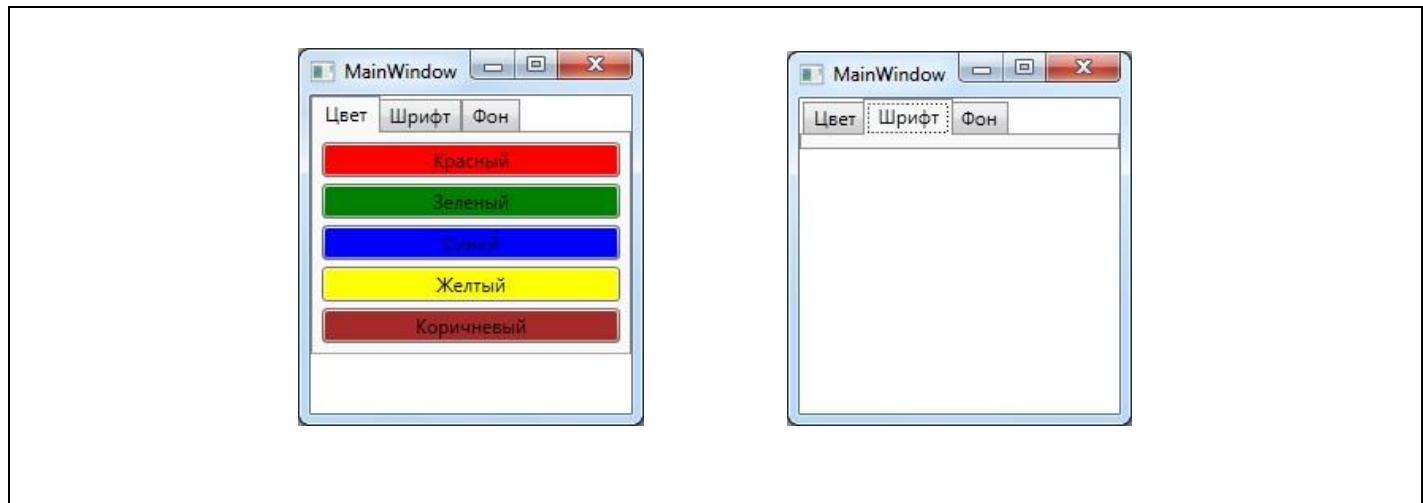


Рисунок 50 – пример работы TableItem

Кисти

Кисти в WPF-приложениях используются для рисования линий и для заполнения областей: фонов, передних планов, границ элементов, областей прозрачности.

Кисть SolidColorBrush

Используется для рисования линий и заполнения области одним сплошным цветом. Данный тип кисти используется в случаях, когда для свойств Foreground, Background, Fill, Stroke заданы текстовые значения в виде названия цвета.

Пример 6 Код XAML	Результат
<pre><TextBlock> <TextBlock.Background> <SolidColorBrush Color="DarkGreen" /> </TextBlock.Background> <TextBlock.Foreground> <SolidColorBrush Color="White" /> </TextBlock.Foreground> Кисть SolidColorBrush </TextBlock></pre>	
<pre><TextBlock Background="DarkGreen" Foreground="White"> Кисть SolidColorBrush </TextBlock></pre>	

Рисунок 51 – пример работы кисти SolidColorBrush

Кисть RadialGradientBrush

Используется для радиального градиентного заполнения области. Основные свойства:

GradientStops – коллекция объектов GradientStop, определяющих промежуточные точки градиента;

GradientOrigin – относительные координаты центра радиального градиента в формате «X,Y».

Значение по умолчанию «0.5,0.5».

Пример 7 Код

XAML

```
<TextBlock Foreground="White" Height="150">
    <TextBlock.Background>
        <RadialGradientBrush>
            <GradientStop Color="DarkGreen" Offset="0" />
            <GradientStop Color="DarkBlue" Offset="0.5" />
            <GradientStop Color="DarkRed" Offset="1" />
        </RadialGradientBrush>
    </TextBlock.Background>
    Кисть RadialGradientBrush
</TextBlock>
```

Результат

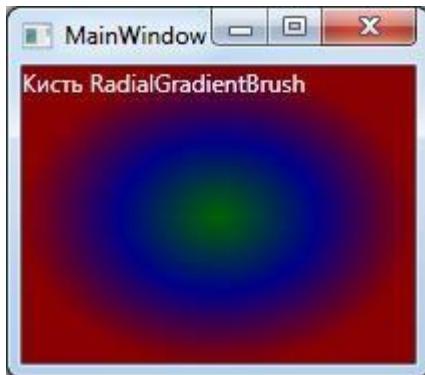


Рисунок 52 – пример работы кисти RadialGradientBrush

Пример 8 Код

XAML

```
<Ellipse Margin="10" Width="200" Height="200">
    <Ellipse.Fill>
        <RadialGradientBrush GradientOrigin="0.3,0.3">
            <GradientStop Color="White" Offset="0"/></GradientStop>
            <GradientStop Color="Blue" Offset="1"/></GradientStop>
        </RadialGradientBrush>
    </Ellipse.Fill>
</Ellipse>
```

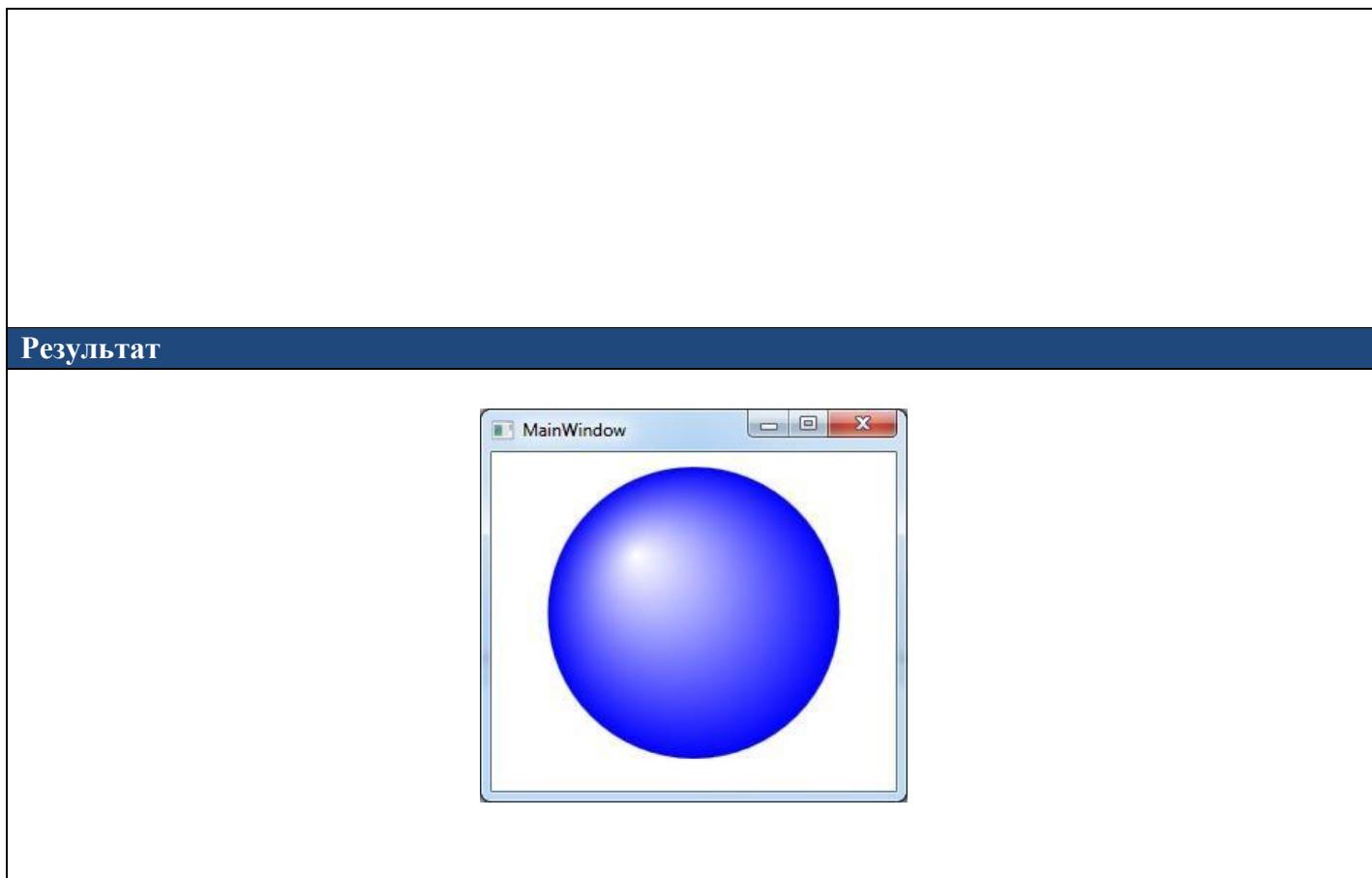


Рисунок 53 – пример работы кисти RadialGradientBrush

Кисть LinearGradientBrush

Используется для линейного градиентного заполнения области. Основные свойства:

GradientStops – коллекция объектов GradientStop, определяющих промежуточные точки градиента;

StartPoint – относительные координаты начала линейного градиента в формате «X,Y». Значение по умолчанию «0,0»;

EndPoint – относительные координаты конца линейного градиента в формате «X,Y». Значение по умолчанию «1,1».

Пример 9 Код XAML

```
<TextBlock Foreground="White" Height="150">
    <TextBlock.Background>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
            <GradientStop Color="DarkGreen" Offset="0" />
            <GradientStop Color="DarkBlue" Offset="0.5" />
            <GradientStop Color="DarkRed" Offset="1" />
        </LinearGradientBrush>
    </TextBlock.Background>
</TextBlock>
```

Кисть LinearGradientBrush

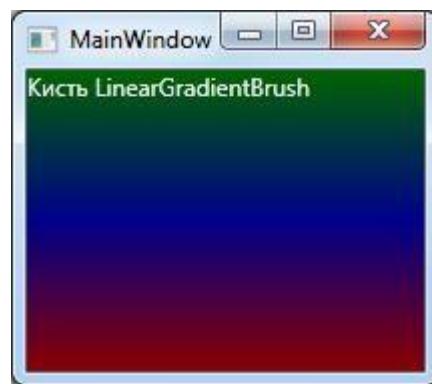
Результат

Рисунок 54 – пример работы кисти LinearGradientBrush

Пример 10 Код XAML

```
<TextBlock FontSize="100">
    <TextBlock.Foreground>
        <LinearGradientBrush>
            <GradientStop Color="Yellow" Offset="0.0" />
            <GradientStop Color="Red" Offset="0.2" />
            <GradientStop Color="Blue" Offset="0.7" />
            <GradientStop Color="LightGreen" Offset="1.0" />
        </LinearGradientBrush>
    </TextBlock.Foreground>
    Градиент
</TextBlock>
```

Результат

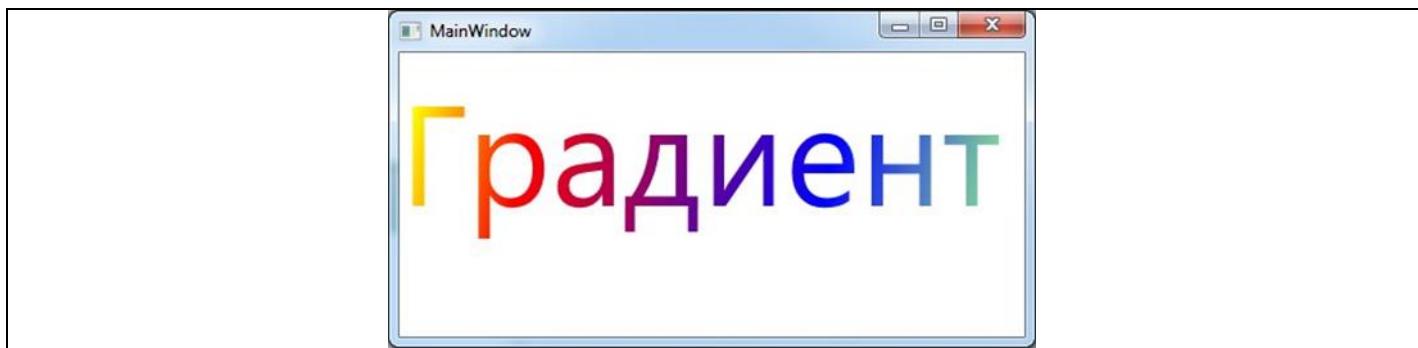


Рисунок 55 – пример работы кисти LinearGradientBrush

Кисть ImageBrush

Используется для заполнения области графическим изображением, которое может растигиваться, масштабироваться или многократно повторяться. Основные свойства:

ImageSource – определяет изображение (ресурс, внешний файл или URL-адрес).

Пример 11 Код XAML

```
<TextBlock FontWeight="ExtraBold" Height="150">
    <TextBlock.Background>
        <ImageBrush Opacity="0.3" ImageSource="http://aics.ru/img/logo.gif" />
    </TextBlock.Background>
    Кисть ImageBrush
</TextBlock>
```

Результат



Рисунок 56 – пример работы кисти ImageBrush

Кисть VisualBrush

Используется для заполнения области на основе визуального содержимого какого-либо элемента. Кисть может быть использована для уменьшенного изображения какой-либо области окна (в том числе невидимой в данный момент), а также для создания эффекта отражения. Заполненная область автоматически перерисовывается при каждом изменении внешнего вида исходного элемента. Основные свойства:

Visual – ссылка на элемент, визуальное изображение которого используется. Значение свойства указывается в виде расширения разметки Binding, например: Visual="{Binding ElementName=ButtonRun}"

В примере 12 кисть Visual используется для вывода уменьшенного изображения вкладки в заголовок этой вкладки.

Пример 12 Код

XAML

```
<TabControl>
    <TabItem>
        <TabItem.Header>
            <Rectangle Width="50" Height="50">
                <Rectangle.Fill>
                    <VisualBrush Visual="{Binding ElementName=colors}" />
                </Rectangle.Fill>
            </Rectangle>
        </TabItem.Header>

        <StackPanel Orientation="Vertical" x:Name="colors">
            <Button Background="Red">Красный</Button>
            <Button Background="Green">Зеленый</Button>
            <Button Background="Blue">Синий</Button>
            <Button Background="Yellow">Желтый</Button>
            <Button Background="Brown">Коричневый</Button>
        </StackPanel>
    </TabItem>
    <TabItem Header="Шрифт">
    </TabItem>
    <TabItem Header="Фон">
    </TabItem>
</TabControl>
```

Результат



Рисунок 57 – пример работы кисти VisualBrush

В примере 13 кисть Visual используется для создания эффекта отражения текстового поля. При изменении текста в текстовом поле отражение автоматически изменяется.

Пример 13 Код XAML

```
<StackPanel>
    <TextBox Name="txt" FontSize="20">Пример отражения текстового поля</TextBox>
    <Rectangle Height="20" Opacity="0.2" RenderTransformOrigin="0,0.5">
        <Rectangle.Fill>
            <VisualBrush Visual="{ Binding ElementName=txt}" />
        </Rectangle.Fill>
        <Rectangle.RenderTransform>
            <ScaleTransform ScaleY="-1"></ScaleTransform>
        </Rectangle.RenderTransform>
    </Rectangle>
</StackPanel>
```

Результат

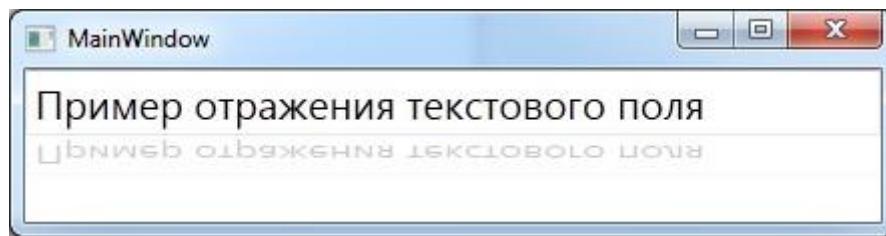


Рисунок 58 – пример работы кисти Visual

Задание 1

На основе примера 8 разработайте WPF-приложение с анимацией источника света, светового пятна от него на шаре и отражения шара. Для анимации начала радиального градиента используйте элемент PointAnimation, свойства From и To которого задаются в формате "X,Y" (пример: To="0,1")

Дополнительное задание: для задания 1 примените элемент Expander.

Контрольные вопросы:

- 1) Какие существуют элементы управления содержимым WPF?
- 2) Для чего используется кисть VisualBrush?
- 3) В чем различие SolidColorBrush и VisualBrush?
- 4) Для чего используется ImageBrush?

Лабораторная работа №10

«Использование фигур в WPF-приложениях»

Цель работы: научиться использовать фигуры в WPF-приложениях

Фигуры в WPF-приложениях используются для рисования двухмерных графических примитивов: линий (Line, Polyline), эллипсов (Ellipse), прямоугольников (Rectangle) и многоугольников (Polygon). Классы фигур являются наследниками абстрактного класса System.Windows.Shapes.Shape, который является наследником класса FrameworkElement. Таким образом, фигуры обрабатываются как обычные элементы пользовательского интерфейса, поддерживают те же основные события, что и другие элементы.

В классе Shape определены следующие базовые свойства фигур:

Stroke – кисть для рисования границы фигуры;

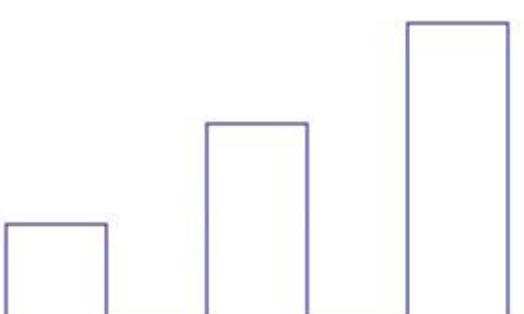
<pre>Stroke="DarkBlue"</pre>	
<pre><Polyline.Stroke> <LinearGradientBrush> <GradientStop Color="Yellow" Offset="0.0" /> <GradientStop Color="Red" Offset="0.2" /> <GradientStop Color="Blue" Offset="0.7" /> <GradientStop Color="LightGreen" Offset="1.0" /> </LinearGradientBrush> </Polyline.Stroke></pre>	

Рисунок 59 – пример использования свойства Stroke

StrokeThickness – толщина границы фигуры;

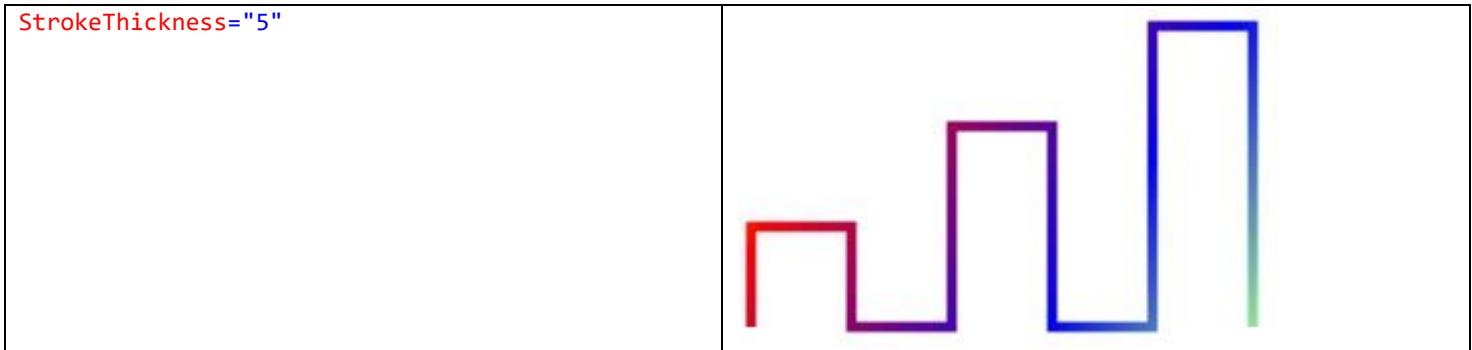


Рисунок 60 – пример использования свойства StrokeThickness

Fill – кисть для рисования поверхности фигуры;

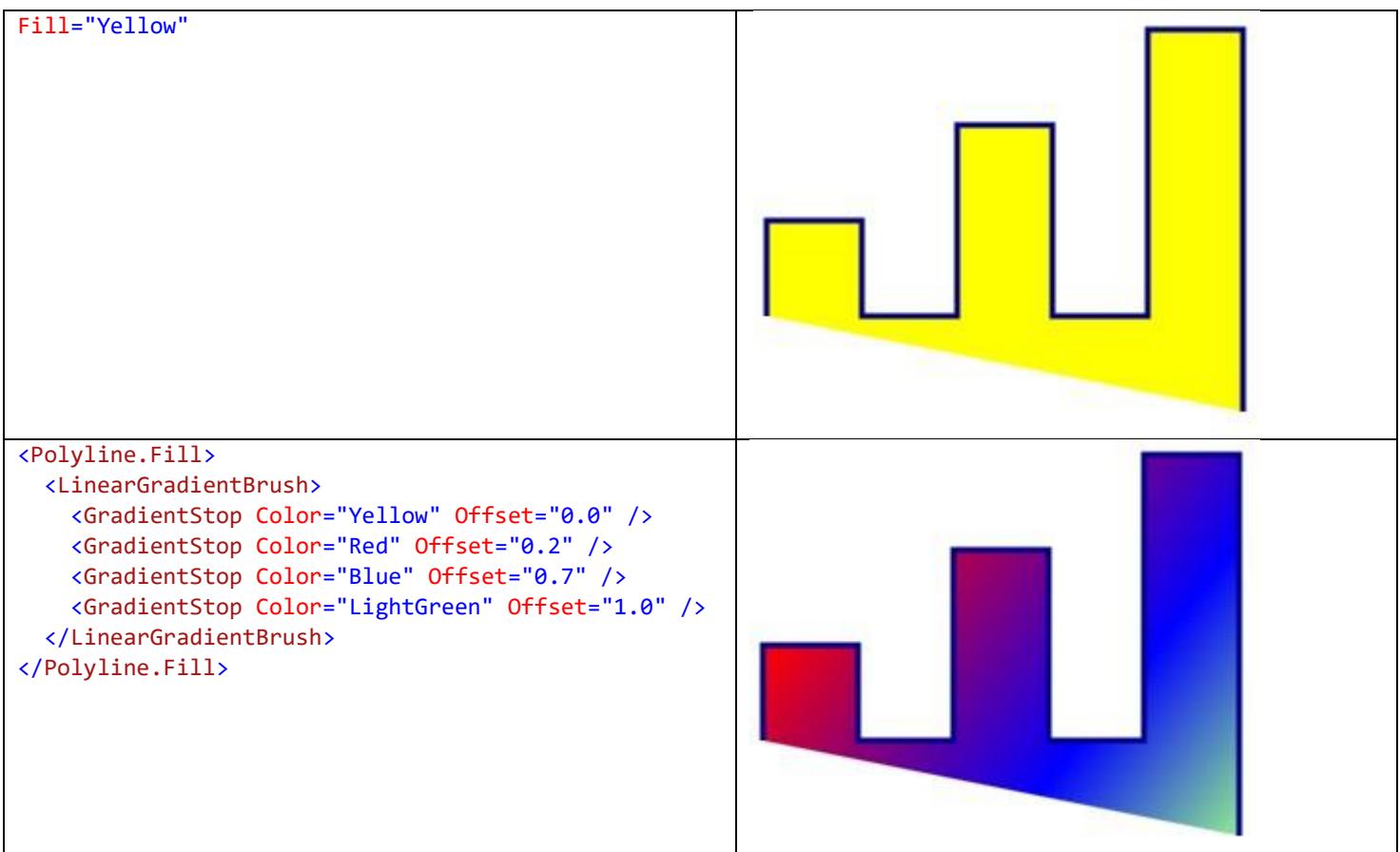


Рисунок 61 – пример использования свойства Fill

StrokeStartLineCap, **StrokeEndLineCap** – контуры краев в начале и в конце линии для незамкнутых фигур; возможные значения:

Flat (значение по умолчанию) – четкая граница;



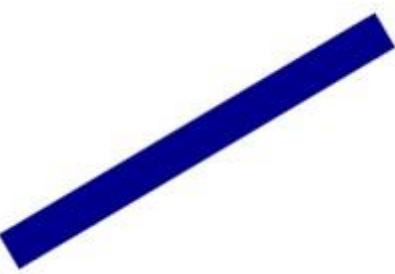
Round – скругленная граница;



Triangle – треугольная граница;



Square – четкая граница (отличается от Flat тем, что линия увеличивается на половину толщины);



StrokeLineJoin – контур углов фигуры; возможные значения:

Miter (значение по умолчанию) – четкие грани углов;



Bevel – угол обрезается в точке сопряжения;

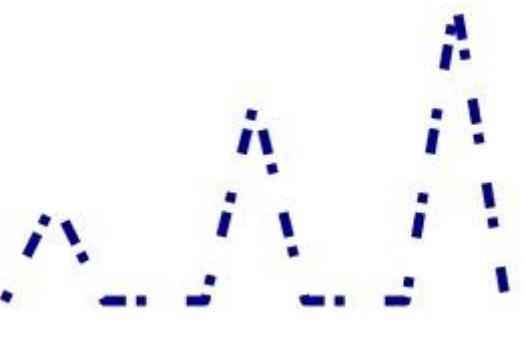


Round – скругление углов;

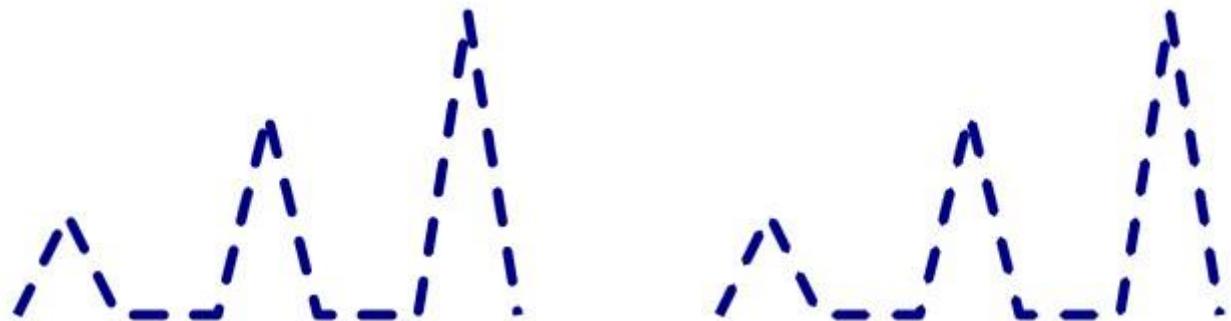


StrokeDashArray – шаблон для определения вида пунктирной линии границы фигуры; определяется в виде набора чисел, разделенных пробелом или запятой. Каждое значение, умноженное на ширину линии, определяет длину либо сплошного сегмента, либо промежутка между сегментами. Если количество чисел нечетное, то каждое значение используется попеременно то для сплошного сегмента, то для промежутка. Если количество чисел в шаблоне четное, то каждое число, стоящее на нечетной позиции, определяет длину сплошного сегмента, а каждое число, стоящее на четной позиции, определяет длину промежутка между сегментами.

<code>StrokeDashArray="1"</code>	
<code>StrokeDashArray="3"</code>	

<code>StrokeDashArray="5,1"</code>	
<code>StrokeDashArray="1,4,2.5,1"</code>	

StrokeDashCap – контуры краев пунктирной линии границы фигуры; возможные значения: Flat, Round, Triangle, Square (такие же, как и у свойств `StrokeStartLineCap` и `StrokeEndLineCap`).



Линия Line

Фигура `Line` представляет отрезок прямой, соединяющий две точки. Начальная и конечная точки задаются свойствами `X1` и `Y1` (для первой точки) и `X2` и `Y2` (для второй точки). Координаты отсчитываются относительно верхнего левого угла элемента `Line`. Свойство `Fill` для линии не используется. Ширина и высота элемента `Line` определяется автоматически в соответствии со значениями свойств `X1`, `Y1`, `X2`, `Y2`.

<pre><Canvas> <Line X1="0" Y1="100" X2="100" Y2="0" Stroke="Brown" /> </Canvas></pre>	
<pre><Canvas> <Line Canvas.Top="20" Canvas.Left="20" X1="0" Y1="100" X2="100" Y2="0" Stroke="Brown" /> </Canvas></pre>	

Рисунок 62 – пример использования свойства Линий

Ломаная линия Polyline

Фигура Polyline представляет последовательность связанных отрезков. В свойстве Points указываются координаты вершин, разделенные пробелом или запятой.

<pre><Polyline Stroke="Brown" Fill="Yellow" Points="10,200 10,150 60,150 60,200 110,200 110,100 160,100 160,200 210,200 210,50 260,50, 260,250" /></pre>	
--	--

Рисунок 63 – пример использования свойства Ломаной линии

Многоугольник Polygon

Фигура Polygon представляет многоугольник, который отличается от ломаной линии Polyline только тем, что начальная и конечная точки ломаной линии соединены отрезком.

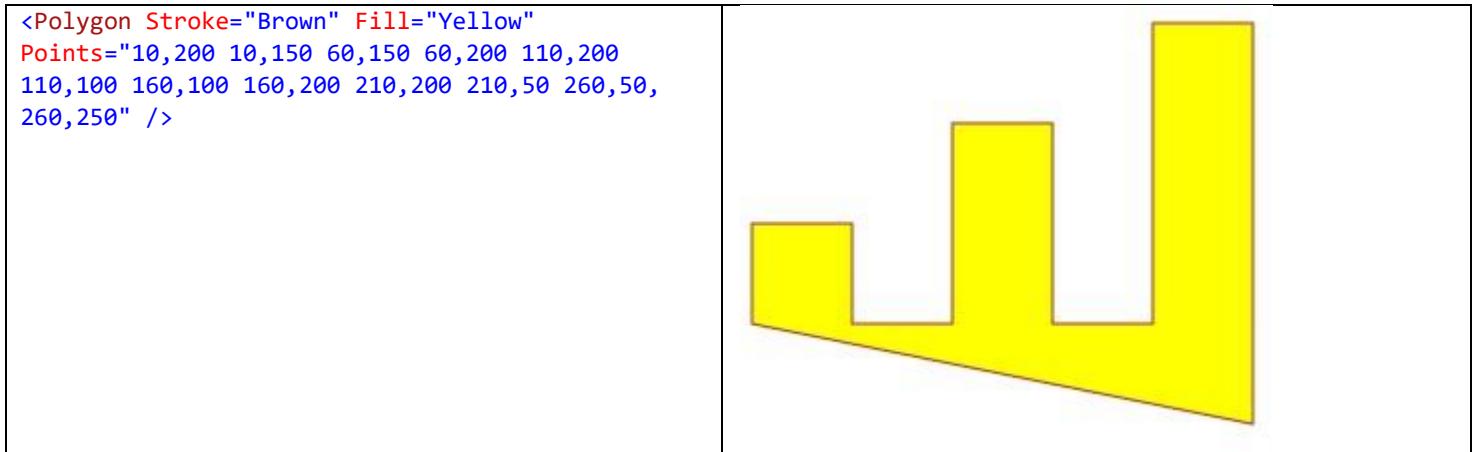


Рисунок 64 – пример использования многоугольника

Элемент Viewbox

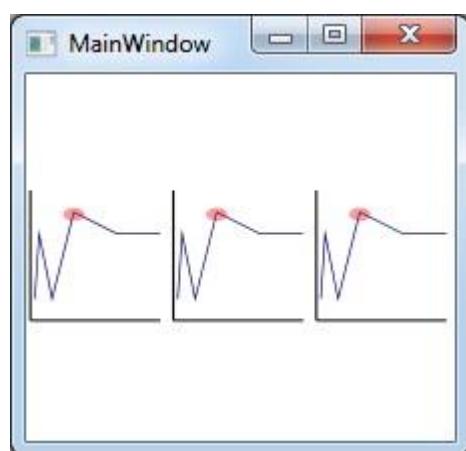
Наиболее распространенным диспетчером компоновки для фигур является Canvas, так как он позволяет совмещать и размещать их произвольным образом. В свою очередь, Canvas помещают в элемент Viewbox, который масштабирует дочерний элемент к своим размерам.

В приведенном ниже примере в таблицу из трех ячеек помещены элементы Viewbox, содержащие диспетчеры компоновки Canvas с фигурами. При изменении размеров окна фигуры масштабируются.

Код XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
<ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Viewbox Grid.Row="0" Grid.Column="0">
        <Canvas Margin="10" Width="310" Height="310">
            <Line Stroke="Black" StrokeThickness="5" X1="0" Y1="0" X2="0" Y2="300" />
            <Line Stroke="Black" StrokeThickness="5" X1="0" Y1="300" X2="300" Y2="300" />
            <Polyline Stroke="DarkBlue" StrokeThickness="3" Points="10,250 20,100 50,250 100,50
200,100 300,100" />
            <Ellipse Width="50" Height="30" Fill="Red" Canvas.Top="40" Canvas.Left="75" Opacity="0.4" />
        </Canvas>
    </Viewbox>
    <Viewbox Grid.Row="0" Grid.Column="1">
    ...
    </Viewbox>
    <Viewbox Grid.Row="0" Grid.Column="2">
        ...
    </Viewbox>
</Grid>
```

Результат



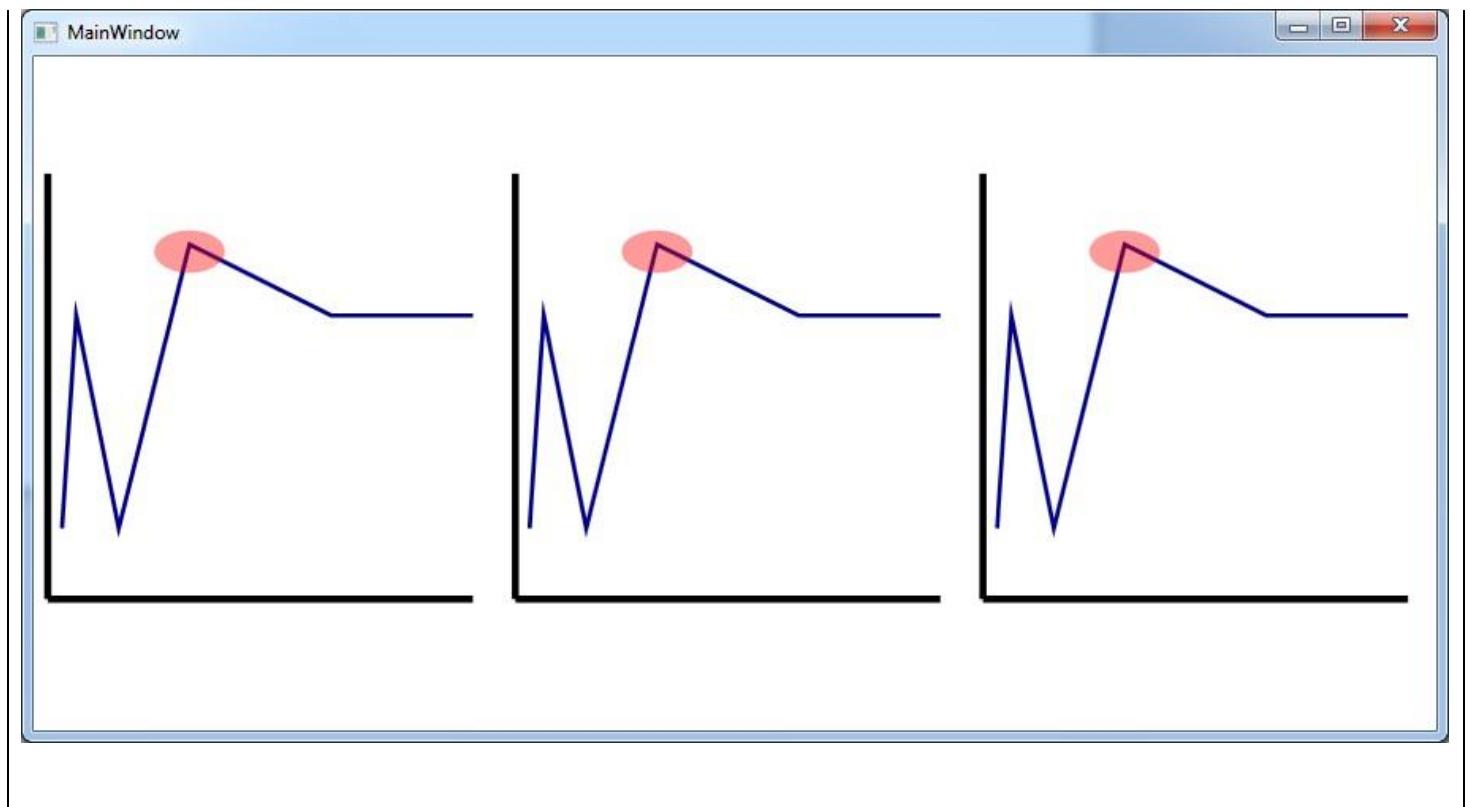


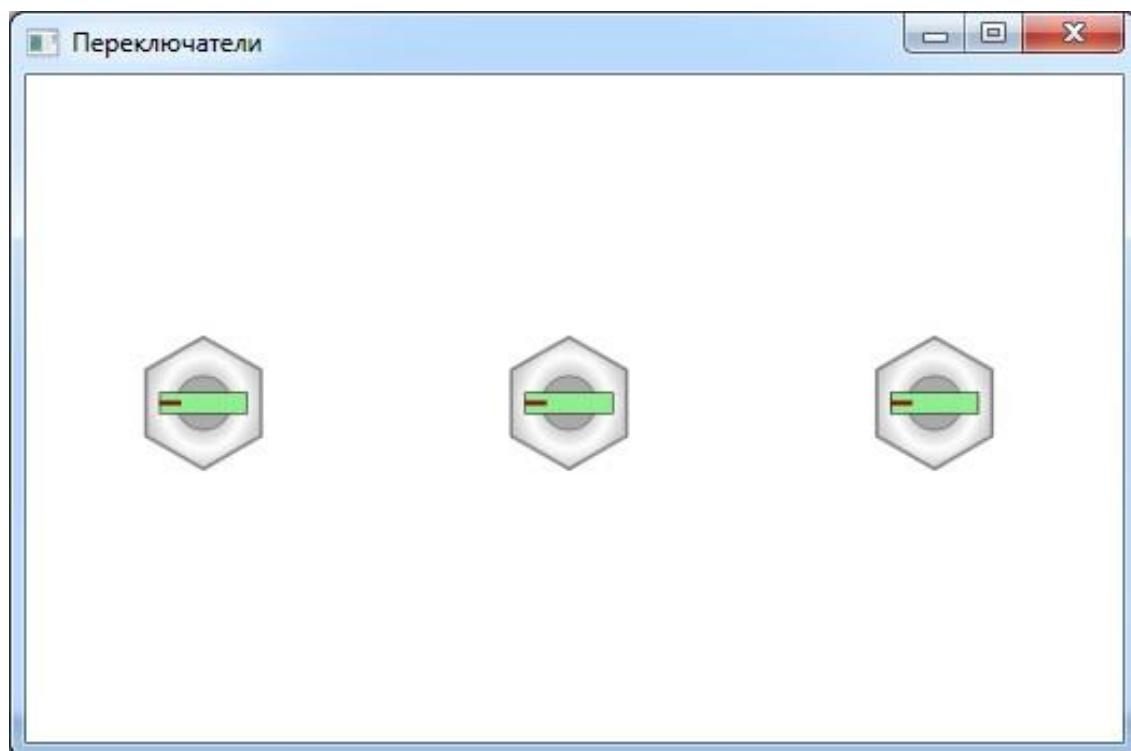
Рисунок 65 – пример использования свойства ViewBox

Задание 1

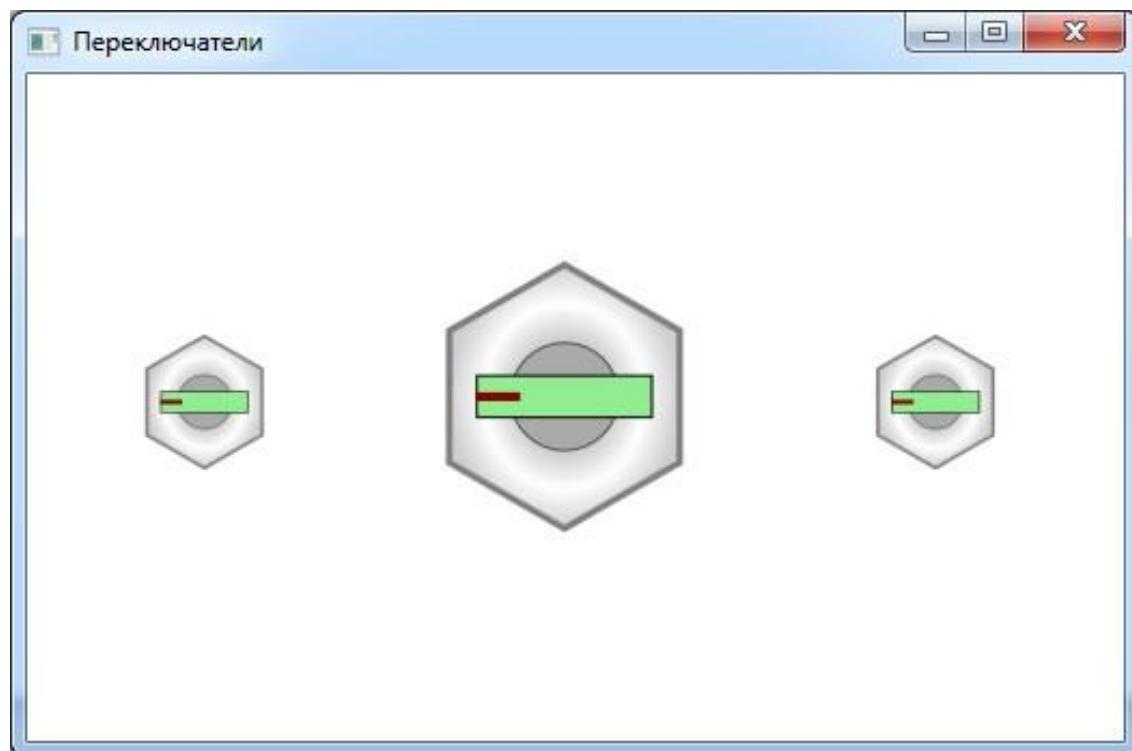
Определите, как изменится поведение программы, если из нее удалить элементы Viewbox и в ячейках таблицы вместо Viewbox разместить сразу Canvas.

Задание 2

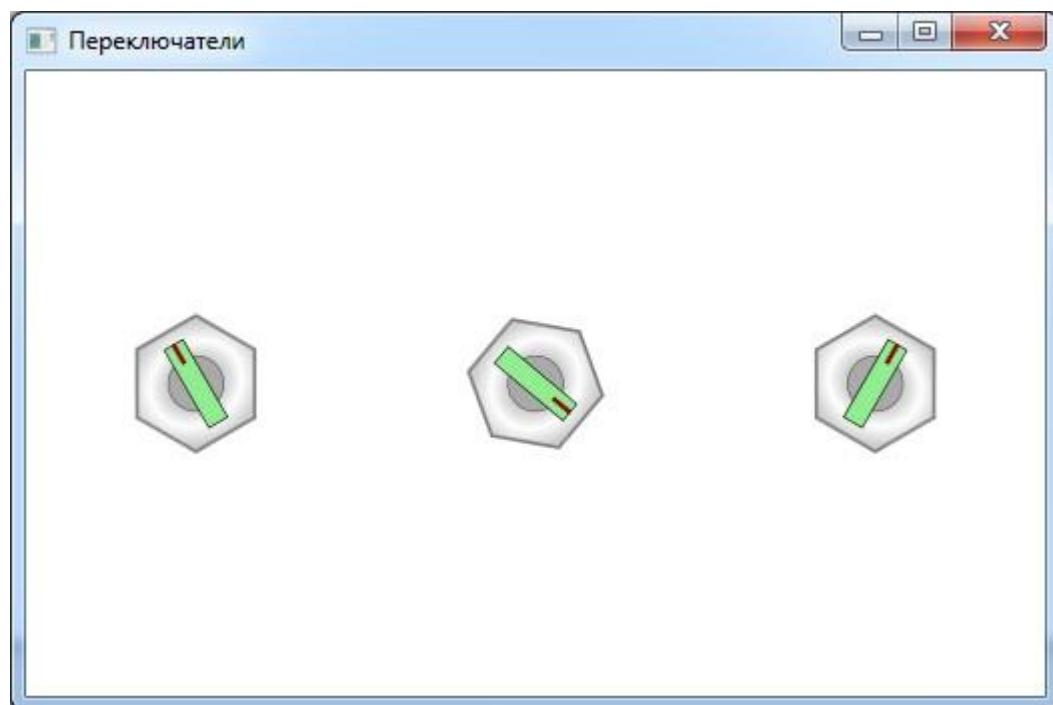
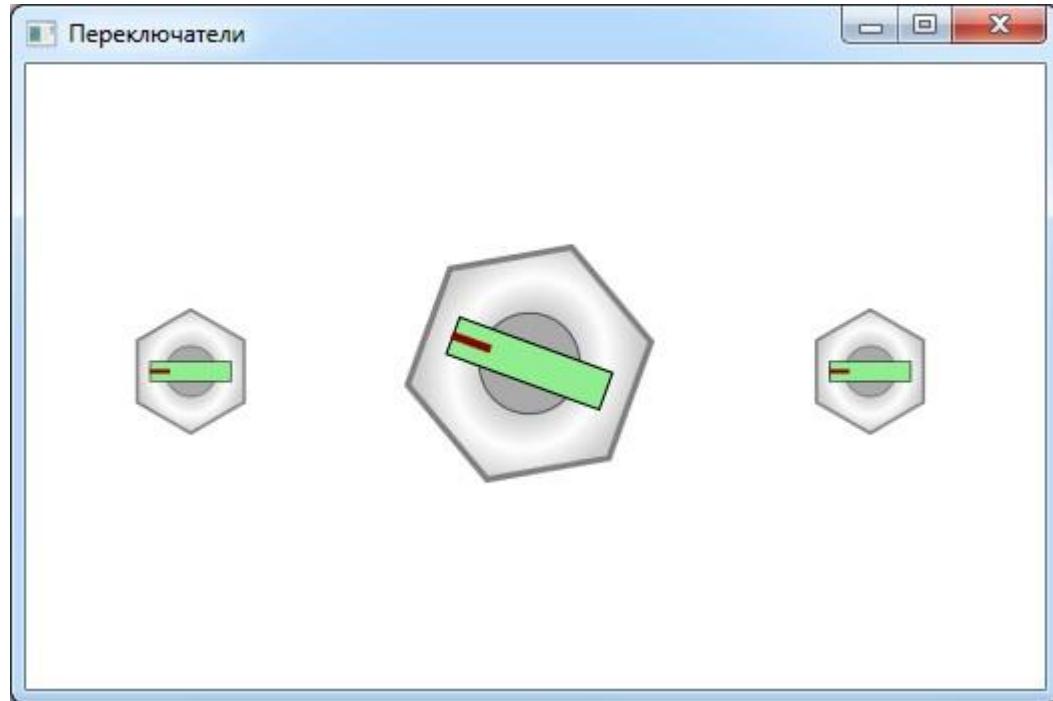
Разработайте WPF-приложение «Переключатели». На экране изображены три одинаковых переключателя, которые при изменении размеров окна пропорционально увеличиваются или уменьшаются :



При наведении курсора на переключатель он увеличивается :



При каждом клике по переключателю он **плавно** поворачивается на 20 градусов по часовой стрелке:



Дополнительное задание: с помощью фигур сделать своё имя.

Контрольные вопросы:

- 1) Какие геометрические примитивы существуют в WPF?
- 2) Для чего нужен элемент ViewBox?
- 3) Какие базовые свойства фигур определены в классе Shape?
- 4) В чём отличие между Line и Polyline ?

Лабораторная работа №11-12

«Разработка ASP.Net-приложения с базой данных: модель, контроллер и представления»

Теоретические сведения.

Общие сведения о ASP.NET MVC

Схема архитектуры **Model-View-Controller (MVC)** разделяет приложение на три основных компонента: **модель**, **представление** и **контроллер**. Платформа ASP.NET MVC представляет собой альтернативу схеме веб-форм ASP.NET при создании веб-приложений. Платформа ASP.NET MVC является легковесной платформой отображения с широкими возможностями тестирования и, подобно приложениям на основе веб-форм, интегрирована с существующими функциями ASP.NET, например с главными страницами и проверкой подлинности на основе членства. Платформа MVC определяется в сборке **System.Web.Mvc**.

Поддержка разработки через тестирование

В дополнение к упрощению сложных структур схема MVC также облегчает тестирование приложений по сравнению с веб-приложениями ASP.NET на основе веб-форм. Например, в веб-приложении ASP.NET на основе веб-форм один класс используется для отображения вывода и для ответа на ввод пользователя. Создание автоматических тестов для приложений ASP.NET на основе веб-форм может представлять сложности, так как для тестирования отдельной страницы следует создать экземпляр класса страницы, всех дочерних элементов управления и других зависимых классов приложения. Большое число экземпляров классов, необходимое для запуска страницы, усложняет создание тестов для отдельных частей приложения. Из-за этого тестирование приложений ASP.NET на основе вебформ может быть сложнее тестирования приложения MVC. Более того,

для тестирования приложения ASP.NET необходим веб-сервер. Платформа MVC разделяет компоненты и активно использует интерфейсы, что позволяет тестировать отдельные элементы вне остальной структуры.

Причины для создания приложения MVC

Следует внимательно продумать вопрос о создании веб-приложения на основе платформы ASP.NET MVC или на основе модели веб-форм ASP.NET. Платформа MVC не заменяет собой модель веб-форм. Обе модели можно использовать для веб-приложений (при наличии существующих приложений на основе веб-форм они будут продолжать работу в нормальном режиме).

Перед использованием платформы MVC или модели веб-форм для определенного веб-сайта следует взвесить все преимущества каждого из подходов.

Платформа ASP.NET MVC имеет следующие преимущества:

- Она облегчает управление сложными структурами путем разделения приложения на модель, представление и контроллер.
- Она не использует состояние просмотра и серверные формы. Это делает платформу MVC идеальной для разработчиков, которым необходим полный контроль над поведением приложения.
- Она использует схему основного контроллера, при которой запросы веб-приложения обрабатываются через один контроллер. Это позволяет создавать приложения, поддерживающие расширенную инфраструктуру маршрутизации.
- Она обеспечивает расширенную поддержку разработки на основе тестирования.
- Она хорошо подходит для веб-приложений, поддерживаемых крупными коллективами разработчиков, а также веб-разработчикам, которым необходим высокий уровень контроля над поведением приложения.

Возможности платформы ASP.NET MVC

Платформа ASP.NET MVC предоставляет следующие возможности.

- **Разделение задач приложения (логика ввода, бизнес-логика и логика пользовательского интерфейса), широкие возможности тестирования и разработки на основе тестирования.** Все основные контракты платформы MVC основаны на интерфейсе и подлежат

тестированию с помощью макетов объекта, которые имитируют поведение реальных объектов приложения. Приложение можно подвергать модульному тестированию без запуска контроллеров в процессе ASP.NET, что ускоряет тестирование и делает его более гибким. Для тестирования возможно использование любой платформы модульного тестирования, совместимой с .NET Framework.

- **Расширяемая и дополняемая платформа.** Компоненты платформы ASP.NET MVC можно легко заменить или настроить. Разработчик может подключать собственный механизм представлений, политику маршрутизации URL-адресов, сериализацию параметров методов действий и другие компоненты. Платформа ASP.NET MVC также поддерживает использование моделей контейнера внедрения зависимости (DI) и инверсии элемента управления (IOC). Модель внедрения зависимости позволяет внедрять объекты в класс, а не ожидать создания объекта самим классом. Модель инверсии элемента управления указывает на то, что если один объект требует другой объект, то первые объекты должны получить второй объект из внешнего источника (например, из файла конфигурации).

Это облегчает тестирование.

- **Расширенная поддержка маршрутизации ASP.NET.** Этот мощный компонент сопоставления URL-адресов позволяет создавать приложения с понятными URL-адресами, которые можно использовать в поиске. URL-адреса не должны содержать расширения имен файлов и предназначены для поддержки шаблонов именования URL-адресов, обеспечивающих адресацию, оптимизированную для поисковых систем (SEO) и для передачи репрезентативного состояния (REST).

- **Поддержка использования разметки в существующих файлах страниц ASP.NET (ASPX), элементов управления (ASCX) и главных страниц (MASTER) как шаблонов представлений.** Вместе с платформой ASP.NET MVC можно использовать существующие функции ASP.NET, например вложенные главные страницы, встроенные выражения (`<% = %>`), декларативные серверные элементы управления, шаблоны, привязку данных, локализацию и т. д.

- **Поддержка существующих функций ASP.NET.** ASP.NET MVC позволяет использовать такие функции, как проверка подлинности с помощью форм и Windows, проверка подлинности по URL-адресу, членство и роли, кэширование вывода и данных, управление состоянием сеанса и профиля, наблюдение за работоспособностью, система конфигурации и архитектура поставщика.

Шаблон проекта MVC

В состав платформы ASP.NET MVC входит шаблон проекта Visual Studio, который позволяет создавать веб-приложения, структура которых соответствует шаблону MVC. Этот шаблон создает новое веб-приложение MVC, конфигурация которого предусматривает все необходимые папки, шаблоны элементов и записи файла конфигурации.

При создании нового веб-приложения MVC Visual Studio предоставляет возможность создания двух проектов одновременно. Первый проект является веб-проектом, в котором реализуется приложение. Второй проект представляет собой проект модульного теста, в котором возможно создание модульных тестов для компонентов MVC первого проекта.

Для тестирования приложений ASP.NET MVC можно использовать любую платформу модульного тестирования, совместимую с платформой .NET Framework. Visual Studio Professional Edition поддерживает тестирование проектов в MSTest.

2. Выполнение работы.

1. Выполнить пример 1.

2. Разработать MVC – приложение для своего варианта (по последней цифре зачетки).

***3. Выполнить пример 2 - MVC-приложение с картой методом TDD.**

Задания для самостоятельного выполнения по вариантам.

Вариант 1

Разработать MVC-приложение со всеми функциями (CRUD-create, read, update, delete) для работы с базой данных о студентах, для их распределения по местам практики: фамилия, год рождения, пол, группа, факультет, средний балл, место работы, город.

Вариант 2

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных об автомобилях: номер, год выпуска, марка, цвет, состояние, фамилия владельца, адрес.

Вариант 3

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных о квартирах, предназначенных для продажи: район, этаж, площадь, количество комнат, сведения о владельце, цена.

Вариант 4

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных о книгах, купленных библиотекой: название, автор, год издания, адрес автора, адрес издательства, цена, книготорговая фирма.

Вариант 5

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных о сотрудниках, имеющих компьютер: фамилия, номер комнаты, название отдела, данные о компьютерах, дата приобретения ПК.

Вариант 6

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных о заказах, полученных сотрудниками фирмы: фамилия, сумма заказа, наименование товара, название фирмы - клиента, фамилия заказчика.

Вариант 7

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных об оценках, полученных студентами на экзаменах: фамилия, группа, предмет, номер билета, оценка, преподаватель.

Вариант 8

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных об авторах web-сайта и их статьях: имя, адрес, учетная запись, пароль, тема, заголовок, дата публикации.

Вариант 9

Спроектировать структуру базы данных о списке рассылки и подписчиках: тема и содержание письма, дата отправки, имена и адреса подписчиков, их учетные записи и пароли.

Вариант 10

В базе данных содержится информация о туристических поездках: страна, город, изображение городской достопримечательности, количество дней, дата поездки, класс отеля, цена.

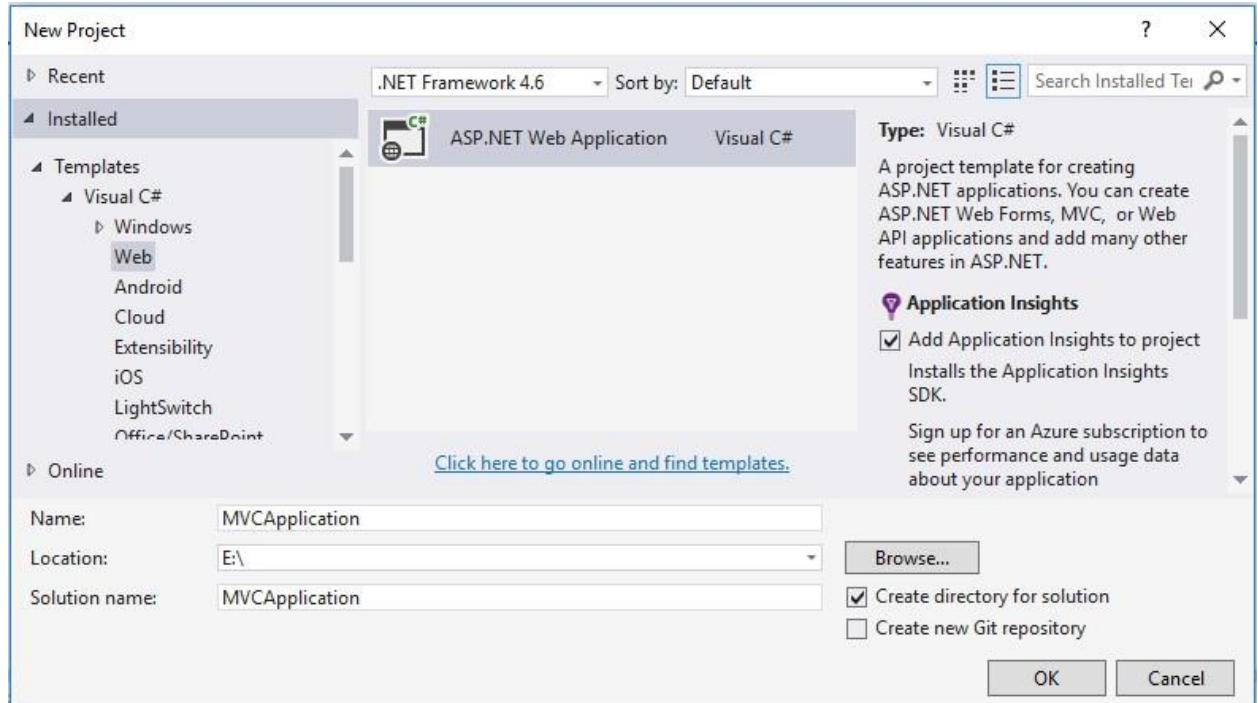
Номер варианта – по номеру по последней цифре шифра зачетки.

Все примеры выполнены с учетом использования ASP.NET MVC 5, и .Net Framework 4.6 (4.7)

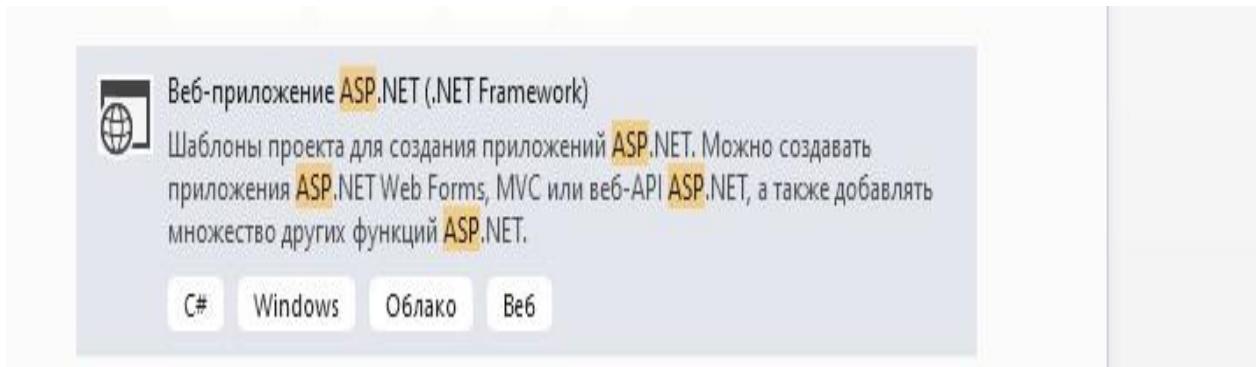
Пример 1. Создание простого MVC 5 приложения ASP.NET в Visual Studio 2015-2017 с созданием базы данных SQL-Server через ADO.NET Entity с подходом DataBase First.

Создание нового проекта MVC.

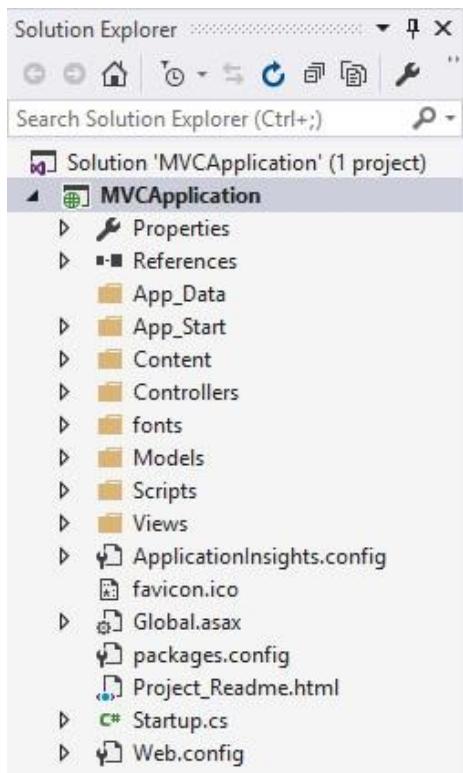
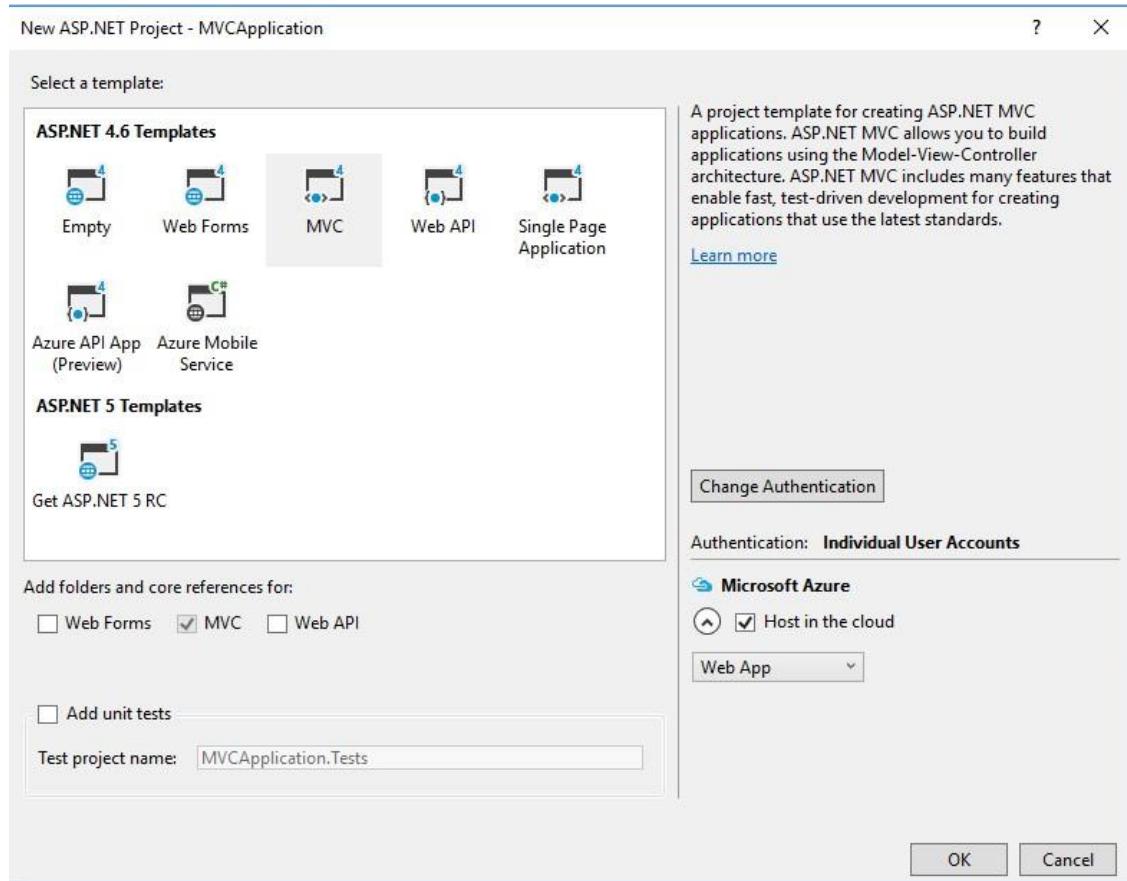
В меню **Файл** выберите **Создать -> Проект**. Откроется диалоговое окно **Создать проект**. Выбираем вкладку **Web** в левой части окна, и выбираем проект **Asp.NET веб приложение (ASP.NET Web Application)**. Вводим имя проекта и выбираем расположение. Нажимаем кнопку **OK**.



Примечание для Visual Studio 2019:



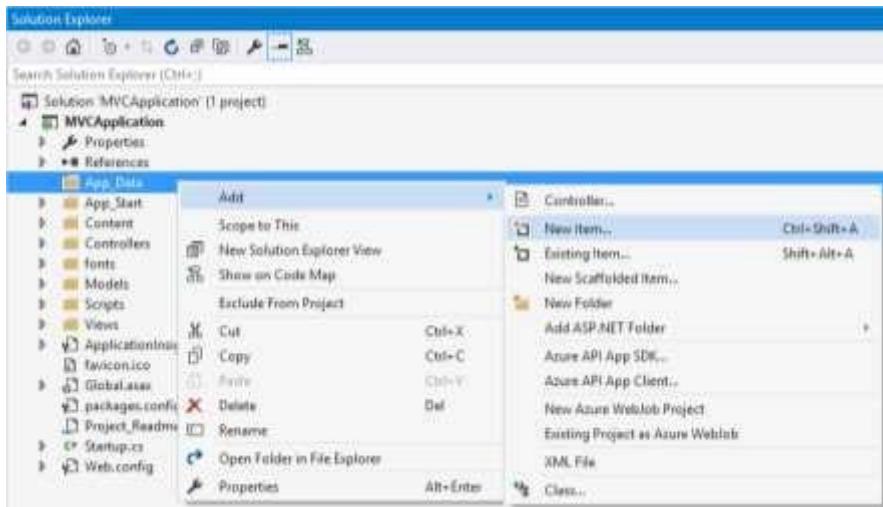
Откроется диалоговое окно выбора типа проекта, необходимо выбрать MVC и настроить параметры аутентификации пользователей, и другие предлагаемые настройки и нажать OK.



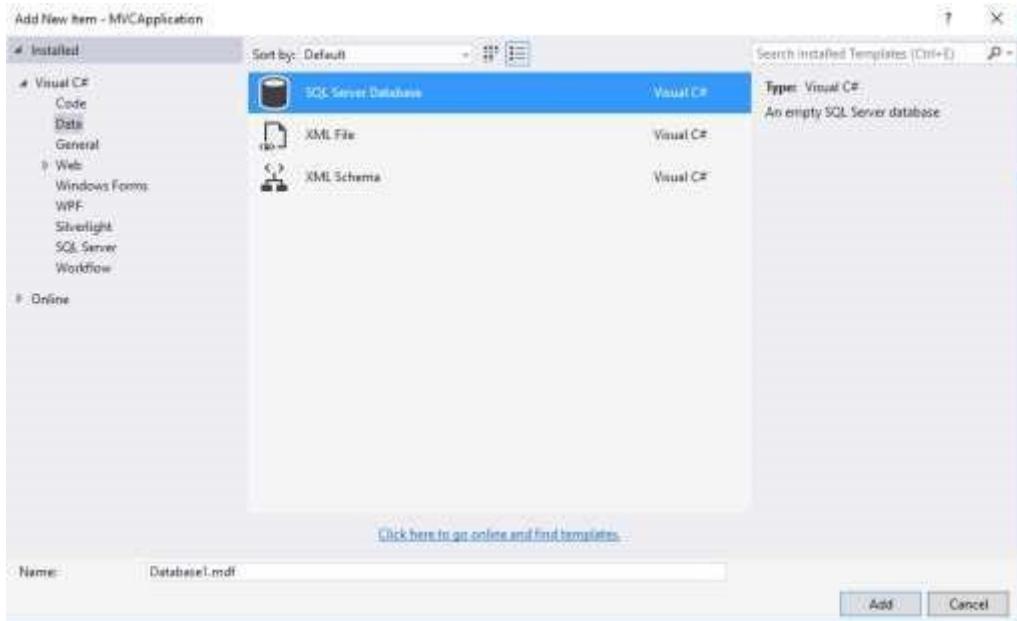
Добавление нового источника данных в локальное приложение.

Здесь будет рассмотрен подход к созданию приложения MVC5 database-first.

Добавляем New Item.

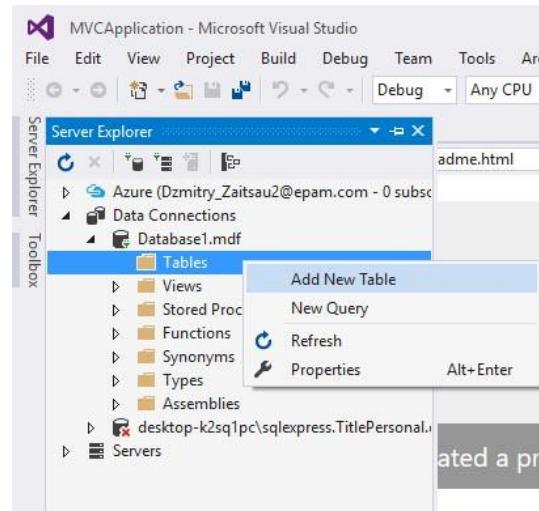


Добавляем локальную базу данных.



Открываем базу для создания таблиц.

Добавляем новую таблицу в базу.



Добавим поле **Id**. В свойствах поля в **Identity Specification** мы ставим параметр **Is Identity** в **Yes**. Таким образом мы получим поле **id** с автоинкрементом с шагом 1. В параметре **Identity increment** можно выставить любой другой шаг. Шаг должен быть целым числом больше 0.

Name	Data Type	Allow Nulls	Default
Id	int	<input checked="" type="checkbox"/>	

Properties

- Id Column**
- Table Designer**
 - Computed Column Specifications:**
 - Full Text Specification:**
 - Identity Specification:**
 - (Is Identity):** True
 - Identity Increment:** 1
 - Identity Seed:** 1

Добавим еще несколько полей **имя (name)** и **Должность (post)**.

Name	Data Type	Allow Nulls	Default
Id	int	<input checked="" type="checkbox"/>	
name	nvarchar(50)	<input type="checkbox"/>	
post	nvarchar(50)	<input type="checkbox"/>	

Устанавливаем Primary key полю **id**.

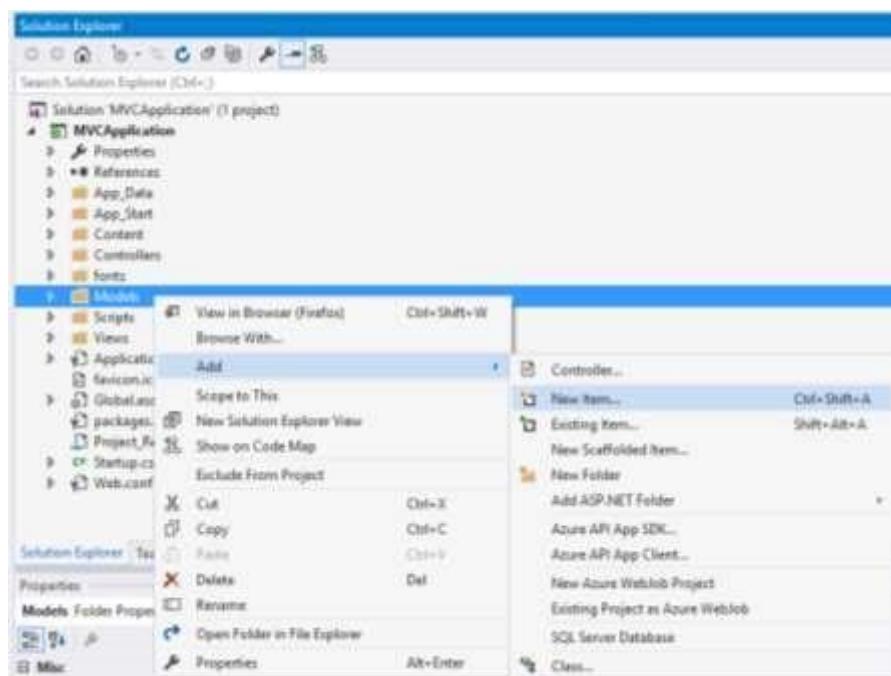
Name	Data Type	Allow Nulls	Default
id	int	<input checked="" type="checkbox"/>	
name	nvarchar(50)	<input type="checkbox"/>	
post	nvarchar(50)	<input type="checkbox"/>	

Keys (1)
PK_Table (Primary Key, Clustered: Id)
Check Constraints (0)
Indexes (0)
Foreign Keys (0)
Triggers (0)

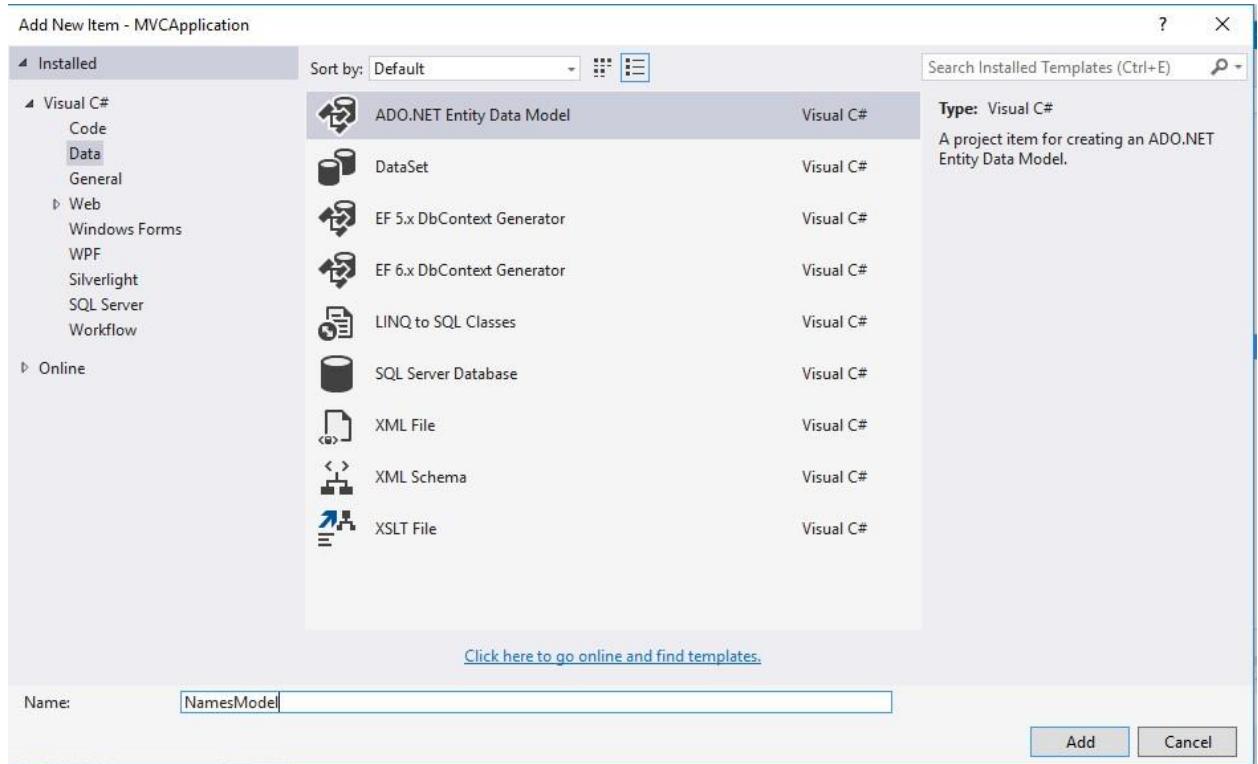
Добавление новой модели к приложению.



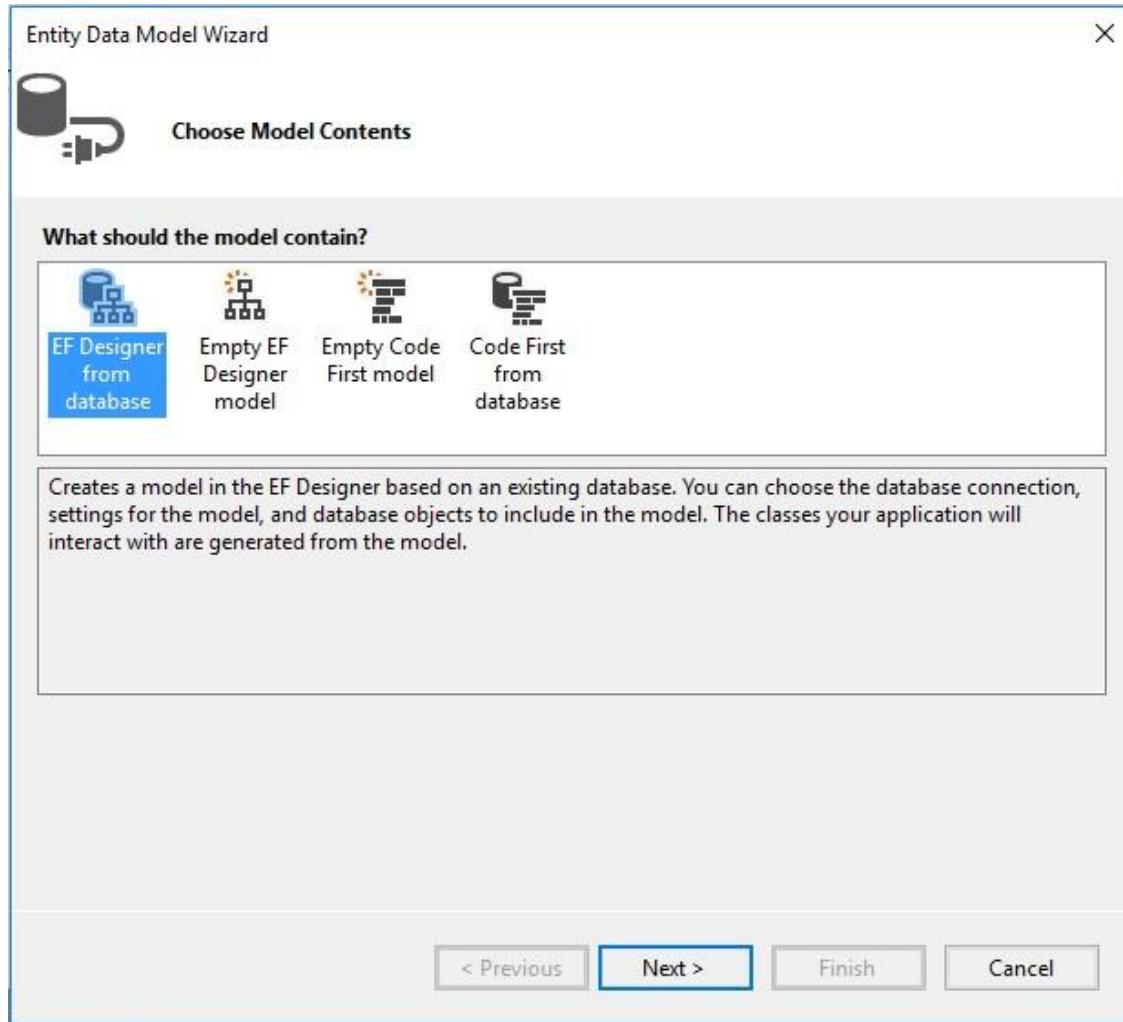
Нажимаем кнопку **Update** и после обновления БД переходим в **Solution Explorer**, чтобы добавить модель.



Добавим **ADO.NET Entity Data Model**. Назовем её **NamesModel**.

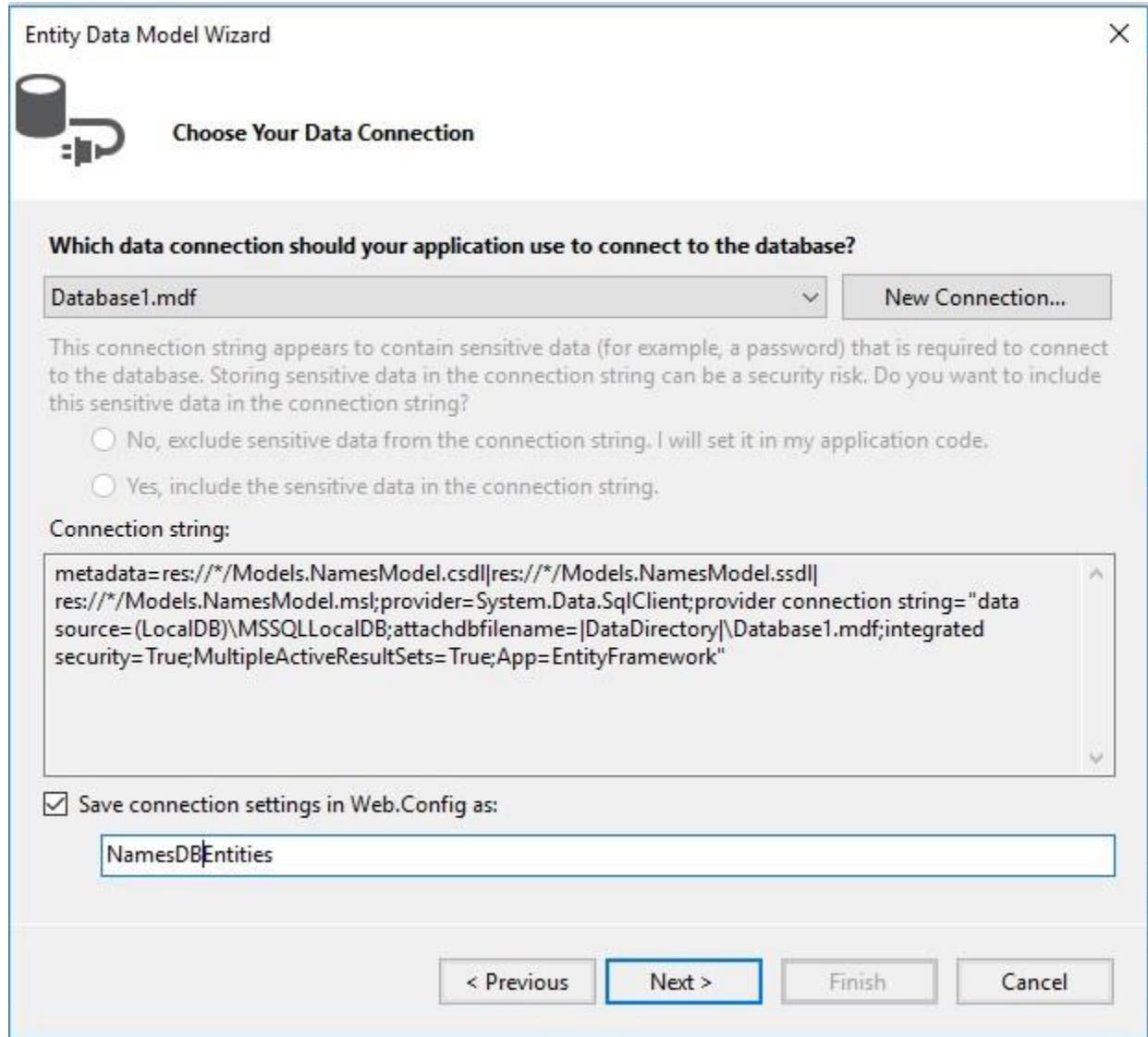


Далее действия вполне просты. Необходимо сгенерировать модель из базы данных.

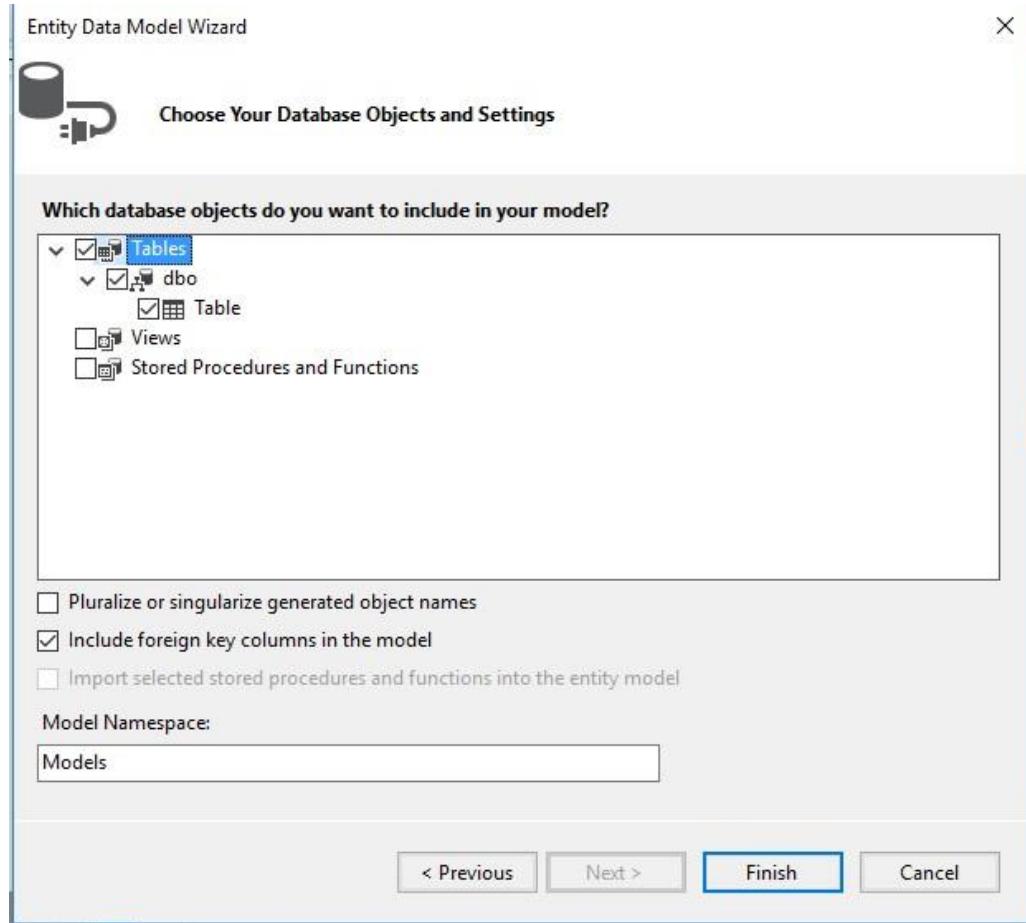


Далее выбираем подключение. На этом этапе Visual Studio предлагает нам подключиться к созданной локальной базе. Так же можно создать новое подключение, например к MS SQL Server, используя стандартные настройки ADO.NET. Так же, если установлены необходимые драйверы, можно подключиться к любому серверу баз данных, к которым возможен доступ.

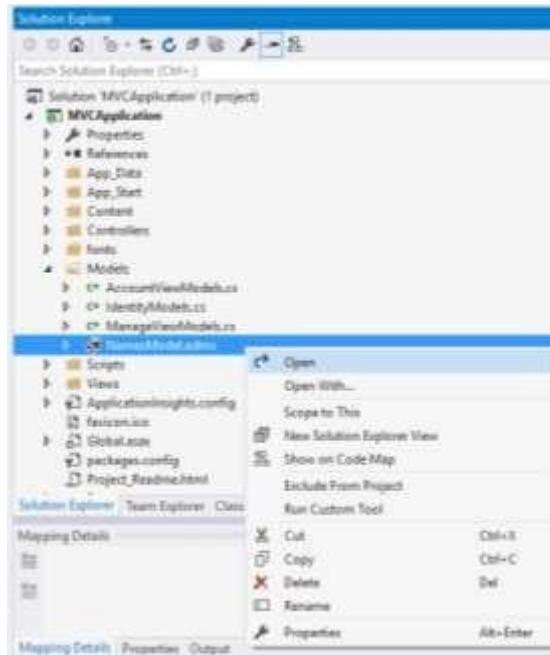
Для своего приложения мы выбираем локальную базу данных DataBase1.mdf. Далее назовем нашу сущность **NamesDBEntities**. Это будет имя контекста данных (укажите потом при создании представлений).



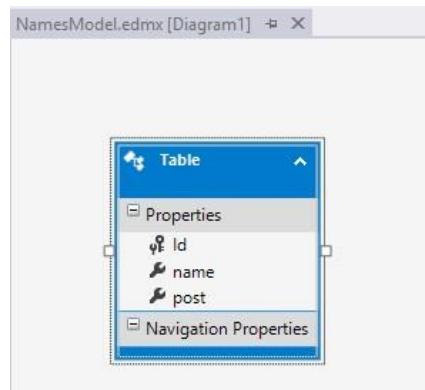
Добавляем, собственно, саму модель из таблица. В нашем случае это таблица Table1. Обратим внимание на **Namespace (Пространство имен)**. Оно может быть произвольным, но лучше использовать существующее пространство имен Models.



Откроем модель для просмотра.



Представление модели выглядит следующим образом.

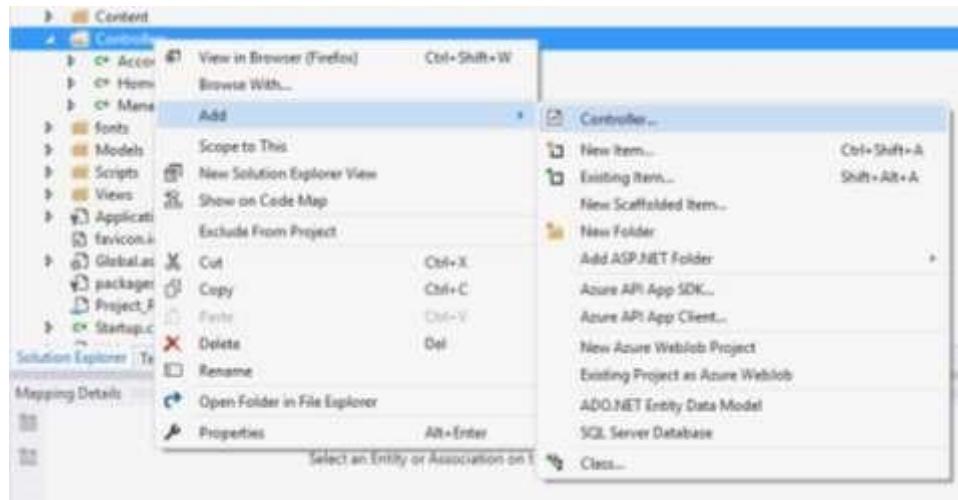


На этом добавление модели к MVC приложению закончено. Теперь создадим контроллер для работы с нашей моделью. Пусть контроллер называется NameController.

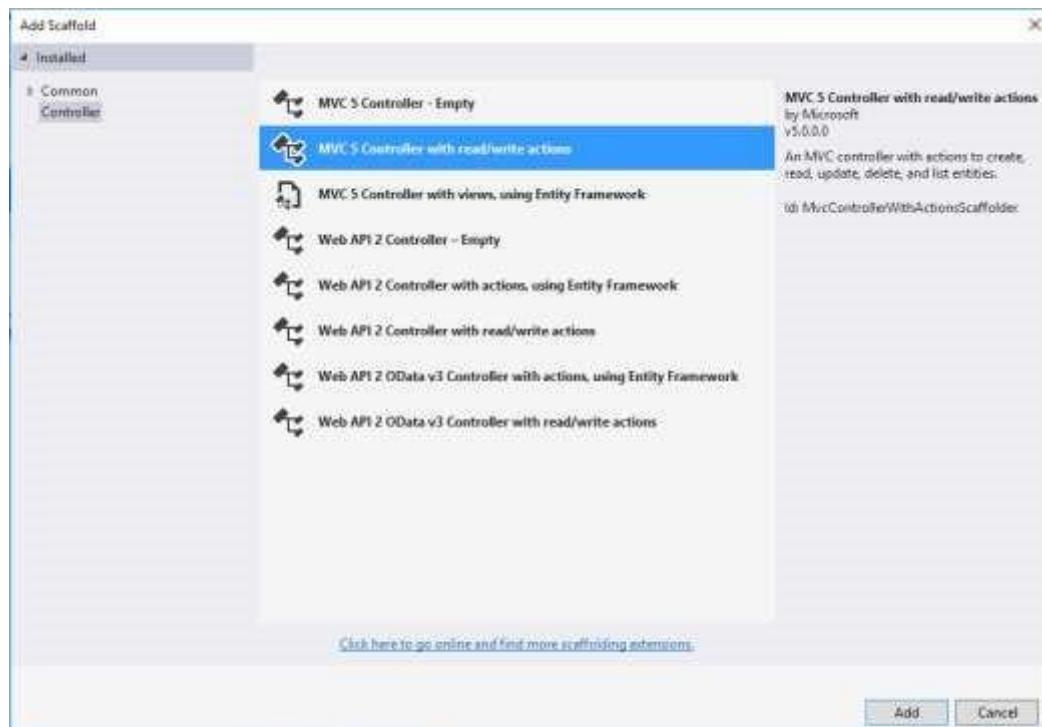
Обратите внимание, что имя таблицы – Table!

Добавление контроллера.

В **Solution Explorer** выбираем папку контроллеров.



Среда разработки предложит нам выбрать варианта создания контроллера. Необходимо создать в контроллере методы для работы с моделью.

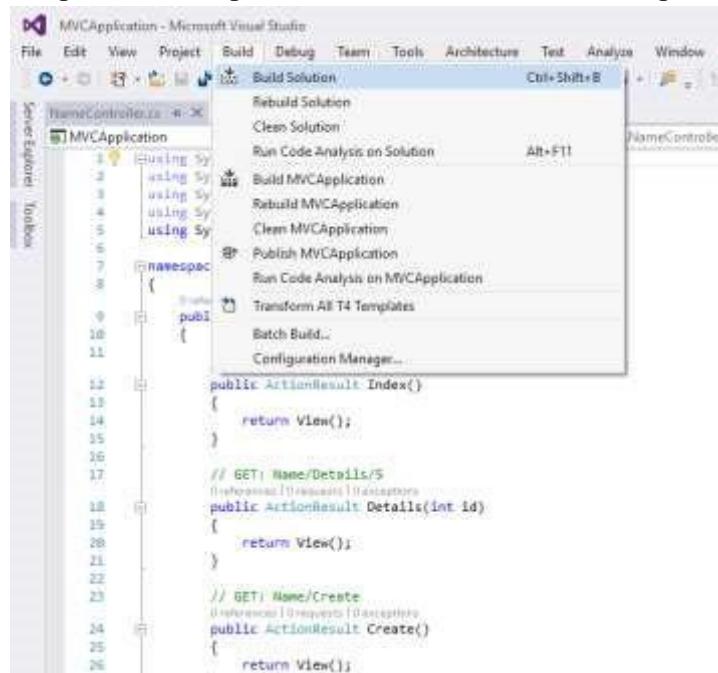


Далее необходимо ввести имя контроллера.

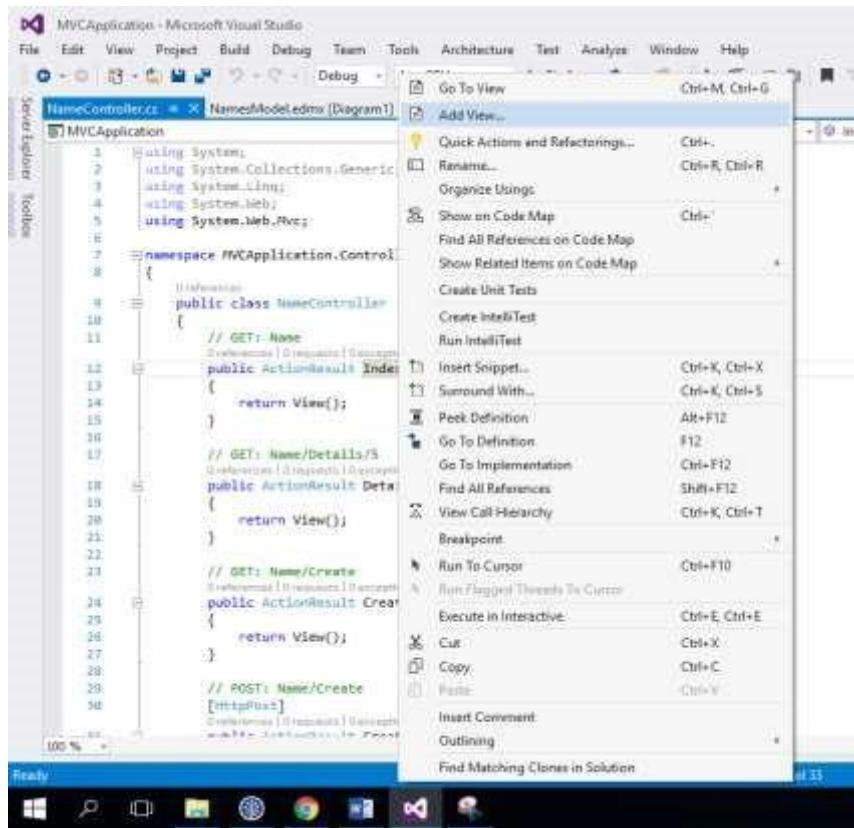


Добавление представления.

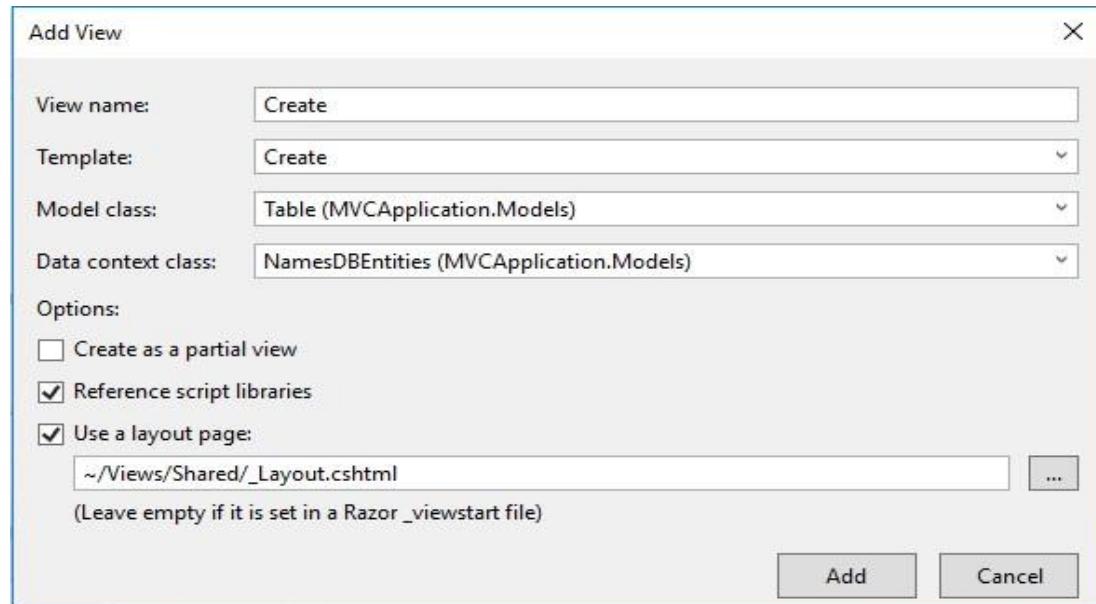
Теперь необходимо создать представления. Но прежде мы должны собрать наше решение (Выполнить Build Solution), иначе добавленная нами модель не будет привязана к пространству имен Models, и мы не сможем сгенерировать правильные представления для методов контроллера.



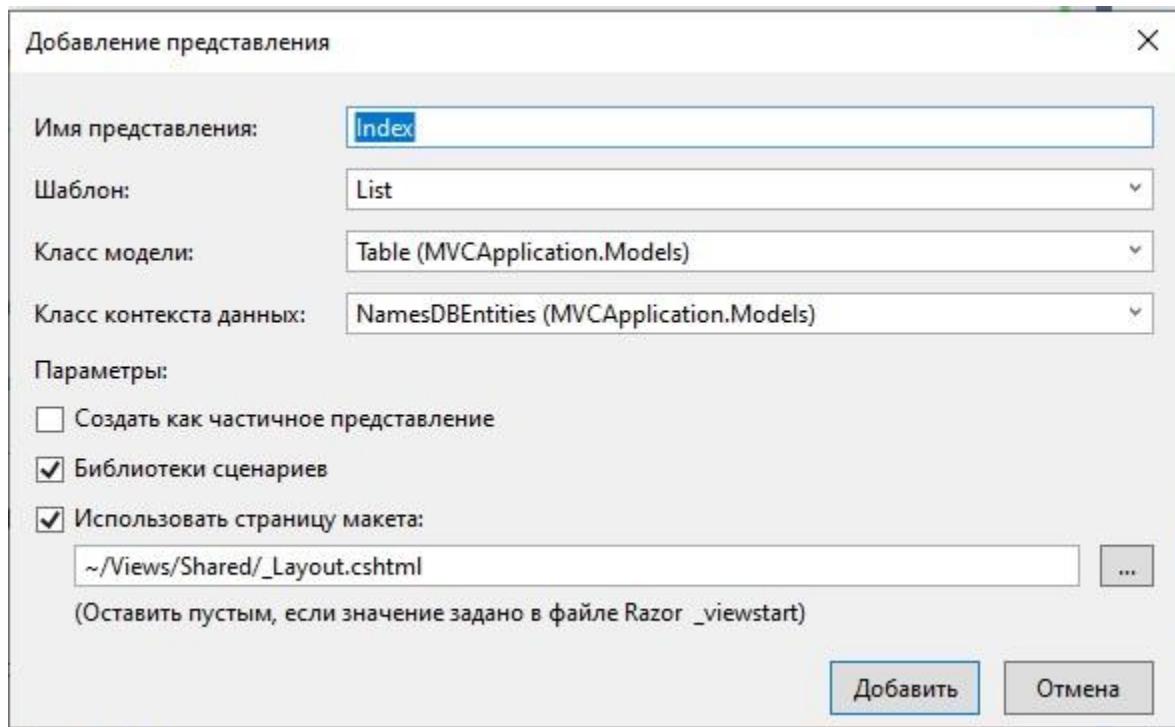
Переходим к созданию представлений. Ставим курсор на имя метода и нажимаем правой кнопкой мыши.



Выбираем из пространства имен Models наш класс Table1 и задаем шаблон (контент) для представления. Например для метода Create целесообразно задать шаблон представления Create.



На Index нужно list.



~/Views/Shared/_Layout.cshtml

Аналогичные операции мы должны проделать для остальных методов.

За исключением тех, которые принимают данные.

Так, например, метод Create имеет два вида (Две перегрузки). Первая перегрузка метода просто вызывает представление для ввода данных. Вторая сохраняет введенные данные в базу. Метод, который принимает данные помечен [HttpPost]. Следовательно, генерация представления для него в данном случае не важна.

После того как мы сгенерировали представление для первого метода Create, необходимо описать второй метод, который обработает введенные данные.

Для начала подключим пространство имен модели

Для этого в разделе using добавим

```
using MVCApplication.Models;
```

Нужно упомянуть это (для работы
HttpStatusCode) using System.Net;

Затем в главном классе `public class NameController : Controller` создадим объект модели Table1. Назовем его db:

```
private NamesDBEntities db = new NamesDBEntities();
```

Должно получиться так:

```
using
System.Linq;
using
System.Web.
Mvc; using
MVCApplication.
Models;
using
System.Net;

namespace MVCApplication.Controllers
{
    public class NameController : Controller
    {
        private NamesDBEntities db = new
NamesDBEntities();
```

...

Мы подготовили наш класс к работе с моделью, можно вносить изменения в существующие методы. Найдем кусок кода:

```
[HttpPost]
0 references | 0 requests | 0 exceptions
public ActionResult Create(FormCollection collection)
{
    try
    {
        // TODO: Add insert logic here

        return RedirectToAction("Index");
    }
    catch
    {
        return View();
    }
}
```

Изменим его:

```
// POST: Name/Create
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "Id,name,post")] Table table)
{
    if (ModelState.IsValid)
    {
        db.Table.Add(table);
    }
    db.SaveChanges();
    return RedirectToAction("Index");
}
return View(table);

}
```

Данные из формы поступают в класс в виде объекта нашей модели, в данном случае это модель Table1, объект NameToSave.

```
db.Table.Add(table); // добавляем объект в модель;
```

```
db.SaveChanges(); // сохраняем изменения в модели;
```

Создадим для метода index представление с шаблоном list;
Изменим метод index:

```
// GET: Name
public ActionResult Index()
{
    return View(db.Table.ToList());
}
```

Перейдем к методу Details: создадим для него представление с шаблоном details; исправим метод:

```
// GET: Name/Details/5
public ActionResult
Details(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Table table =
db.Table.Find(id);
    if (table
== null)
    {
        return HttpNotFound();
    }
    return View(table);
}
```

Перейдем к методу Edit: создадим для него представление с шаблоном Edit; исправим метод.

```
public ActionResult Edit(int id)
{
    return View(db.Table.FirstOrDefault(P => P.Id == id));
}
```

Теперь отредактируем метод Edit, который принимает данные:

```
[HttpPost]
public ActionResult Edit(Table table)
```

```

{
    if (ModelState.IsValid)
    {
        db.Entry(table).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(table);
}

```

Остался последний метод, Delete. Для первого метода создаем представление с шаблоном Delete и исправим сам метод.

```

public ActionResult Delete(int id)
{
    return View(db.Table.Single(P => P.Id == id));
}

```

И отредактируем метод, который собственно удаляет запись:

```

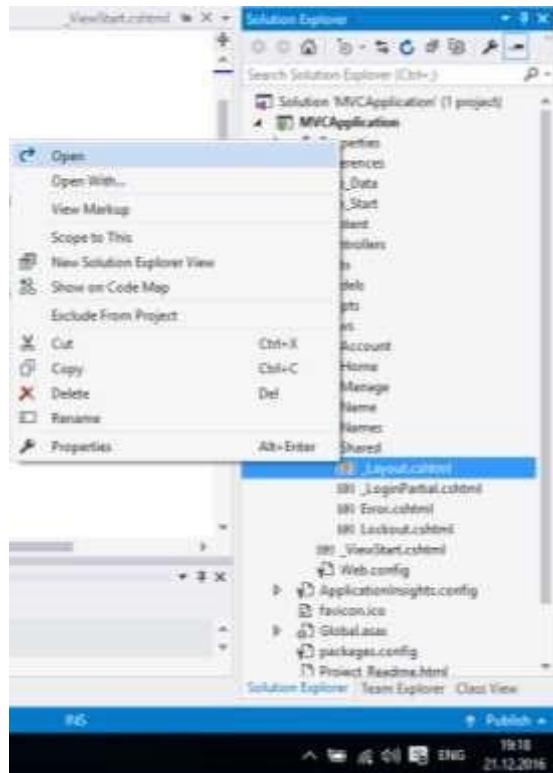
public ActionResult Delete(int id, Table name)

{
try
{
    db.Table.Remove(db.Table.Single(P => P.Id == id));
    db.SaveChanges();
    return RedirectToAction("Index");

}
catch
{
    return View();
}
}

```

Теперь добавим ссылку на наш контроллер на метод index. Для этого отредактируем главную страницу _Layout.cshtml:



Найдем код:

```
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li>@Html.ActionLink("Home", "Index", "Home")</li>
        <li>@Html.ActionLink("About", "About", "Home")</li>
        <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
    </ul>
    @Html.Partial("_LoginPartial")
</div>
```

добавим:

```
<li>@Html.ActionLink("My Name Controller", "Index", "Name")</li>
```

Где My name Controller – то, что отобразится в ссылке, Index – метод, Name – контроллер.

Делаем Build Solution и запускаем наше приложение. В результате, оно должно выглядеть так:

Создание новой записи:

Результат создания записи:

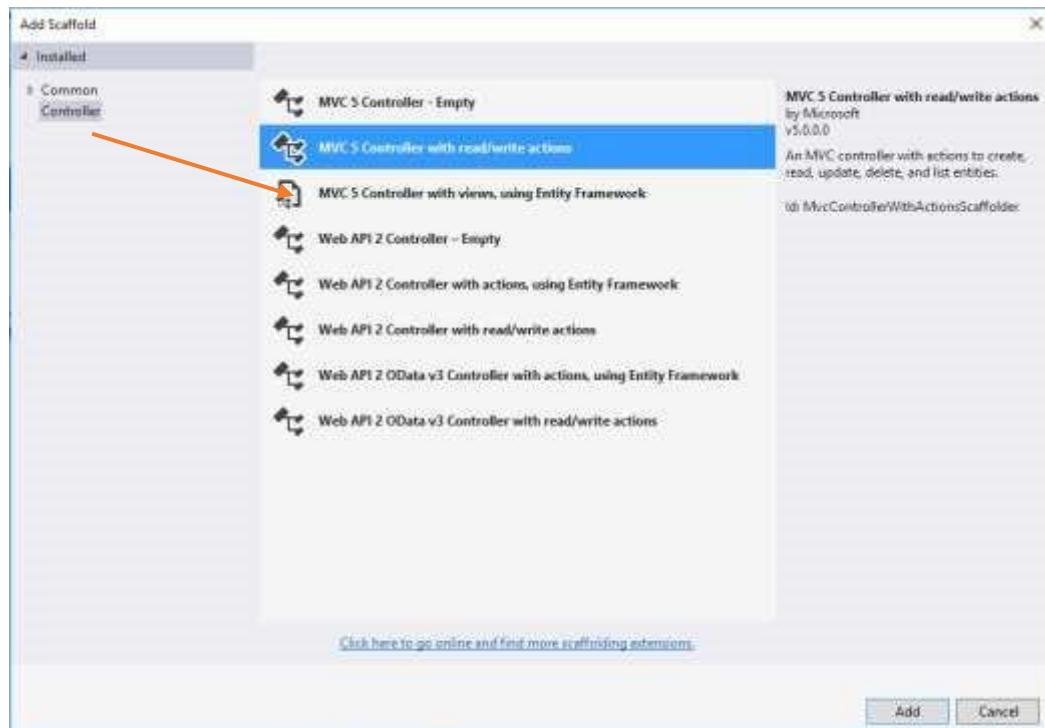
Редактирование записи:

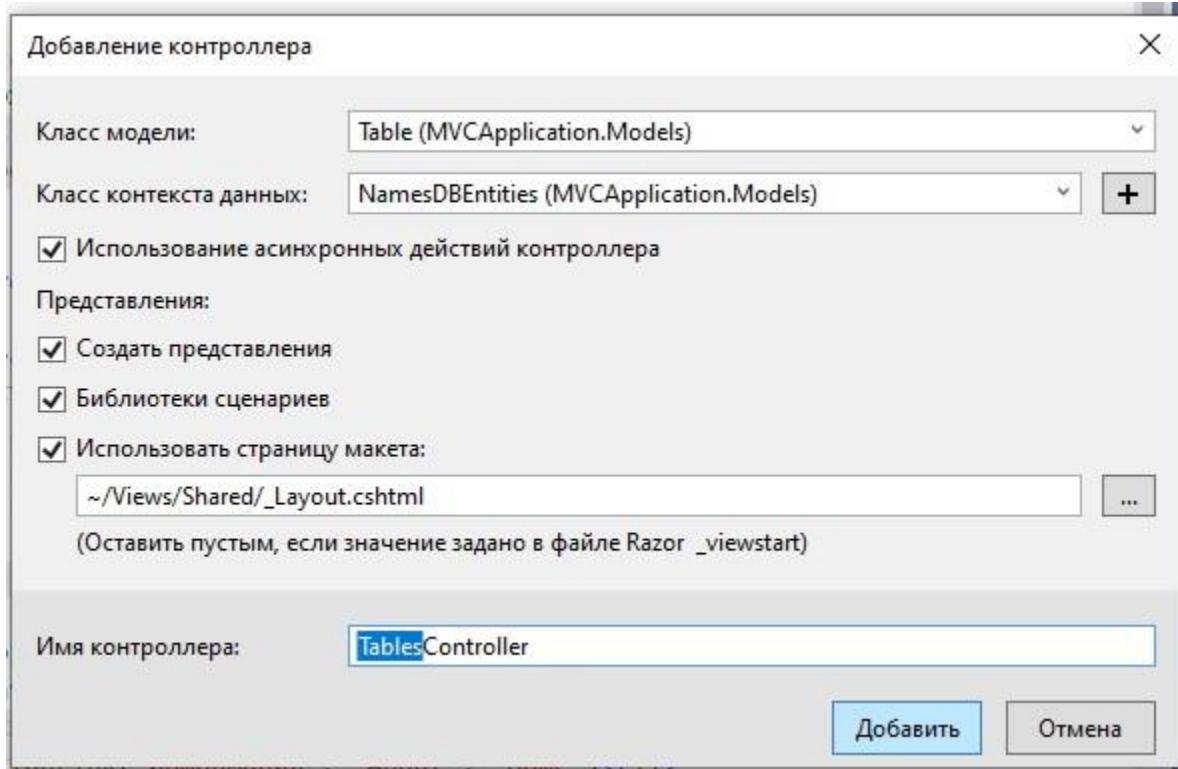


Альтернативный подход к созданию контроллера.

Вы научились создавать контроллеры и представления используя общие шаблонные заготовки, где значительную часть кода пришлось вводить вручную. Но Visual Studio предлагает и более автоматизированный подход к созданию контроллера и типизированных представлений CRUD (create, remove, update, delete).

Создадим еще один контроллер TablesController, выбрав из контекстного меню при создании контроллера вариант не 2-й, как раньше, а 3-й : MVC 5 Controller with views using Entity Framework.





Не забудьте указать класс модели и класс контекста данных, как на рисунке выше, а также включить все опции. Имя контроллера – TablesController. Нажмите кнопку Добавить и получите полностью готовый контроллер с представлениями для просмотра, добавления, удаления и редактирования данных.

Теперь осталось добавить пункт меню Entity controller на первую страницу сайта (чтобы проверить работу нового контроллера). Для этого отредактируем главную страницу _Layout.cshtml:

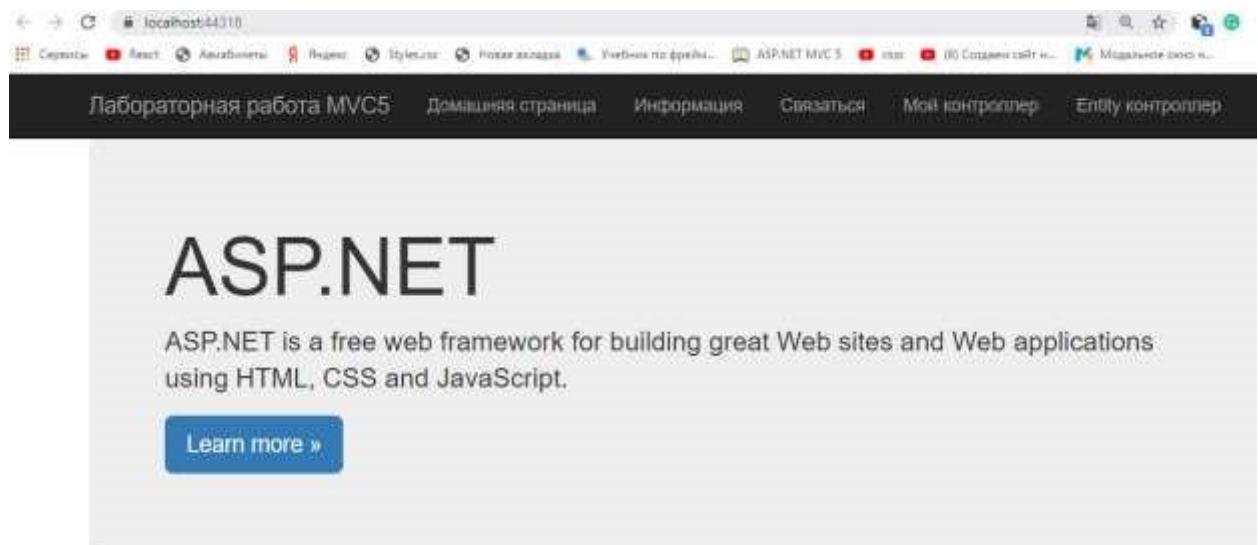
```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>@ViewBag.Title - приложение ASP.NET</title>
@Styles.Render("~/Content/css")
@Scripts.Render("~/bundles/modernizr")
</head>
<body>
<div class="navbar navbar-inverse navbar-fixed-top">
```

```

<div class="container">
    <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="collapse"
        data-target=".navbar-collapse">
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
        </button>
        @Html.ActionLink("Лабораторная работа MVC5", "Index", "Home", new {
            area
            = "" }, new { @class = "navbar-brand" })
    </div>
    <div class="navbar-collapse collapse">
        <ul class="nav navbar-nav">
            <li>@Html.ActionLink("Домашняя страница", "Index", "Home")</li>
            <li>@Html.ActionLink("Информация", "About", "Home")</li>
            <li>@Html.ActionLink("Связаться", "Contact", "Home")</li>
            <li>@Html.ActionLink("Мой контроллер", "Index", "Name")</li>
            <li>@Html.ActionLink("Entity контроллер", "Index", "Tables")</li>
        </ul>
    </div>
</div>

```

Запустим приложение и увидим

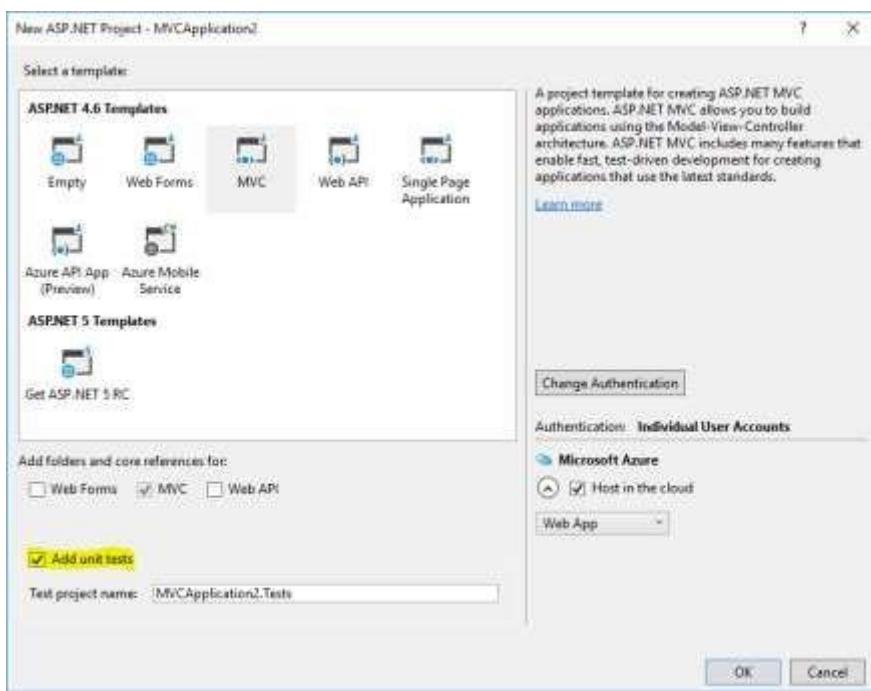


При выборе пункта меню Entity контроллер, он будет выполнять все функции, как и Мой контроллер, т.е. NameController, созданный ранее.

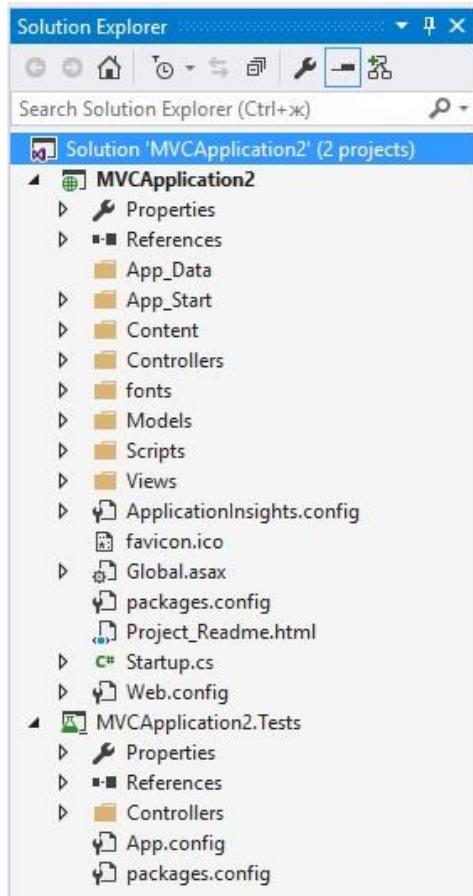
Пример 2. Создание простого проекта MVC с помощью модульных тестов в Visual Studio

Создание нового проекта MVC.

1. В меню **Файл** выберите **Создать -> Проект**. Откроется диалоговое окно **Создать проект**. (Аналогично первому примеру). Необходимо отметить галочку создания тестов.



При создании проекта веб-приложения ASP.NET MVC компоненты MVC разделяются по папкам проекта, которые указаны на следующем рисунке:



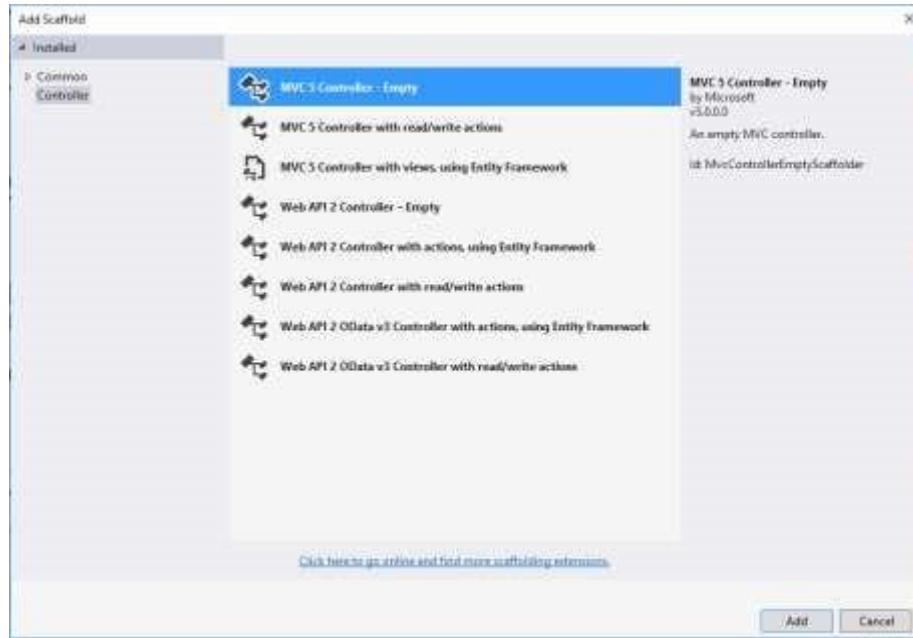
Проект модульных тестов также готов к компиляции и запуску. В него будет добавлен контроллер с методом действия и представление, а для метода действия также будет добавлен модульный тест.

Добавление контроллера в проект MVC

1. В **обозревателе решений** щелкните правой кнопкой мыши папку **Контроллеры**, а затем последовательно выберите пункты **Добавить** и **Контроллер**.

Появится диалоговое окно **Добавить контроллер**.

2. Необходимо выбрать пустой контроллер **MVC 5 Controller - Empty**



3. В поле **Имя** введите **MapsController**.

В платформе MVC ASP.NET имена контроллеров должны кончаться на «Controller», например **HomeController**, **GameController** или **MapsController**.

4. Нажмите кнопку **Добавить**.

Visual Studio добавляет в проект класс **MapsController** и открывает его в редакторе.

Создание заглушки метода действия

Для применения методологии TDD в этом проекте следует создать модульный тест для метода действия, прежде чем создавать сам метод. Если модульный тест должен компилироваться, то для запланированного метода действия необходима заглушка, которая здесь называется **ViewMaps**.

Добавление заглушки метода действия

1. Откройте класс **MapsController** или переключитесь на него в редакторе.

2. Замените метод действия **Index** на следующий код, чтобы создать заглушку метода действия **ViewMaps**.

VB

```
Function ViewMaps()
    ' Add action logic here
    Throw New NotImplementedException()
End Function C#
```

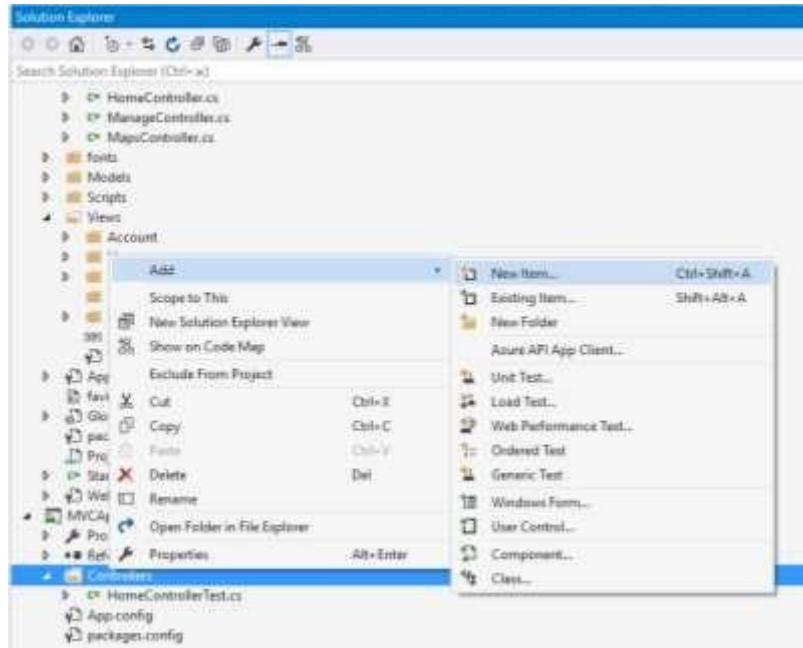
```
public ActionResult ViewMaps()
{
    // Add action logic here
    throw new NotImplementedException(); }
```

Добавление модульных тестов для методов действий

Далее следует добавить в тестовый проект тестовый класс контроллера. В классе будет добавлен модульный тест для метода действия **ViewMaps**. Модульный тест завершится ошибкой, так как заглушка метода действия **ViewMaps** создает исключение. Когда далее метод действия будет готов, тест будет проходить без ошибок.

Добавление модульных тестов для методов действий

1. В проекте тестов щелкните правой кнопкой мыши папку **Контроллеры**, а затем последовательно выберите пункты **Добавить** и **Новый элемент**.



2. В текстовом поле **Имя** введите **MapsControllerTest**.

3. Нажмите кнопку **Добавить**.

Visual Studio добавит класс **MapsControllerTest** в тестовый проект.

4. Откройте класс **MapsControllerTest** и введите следующий код:

C#

```
using System;
using
System.Collections.Gene
ric; using System.Linq;
using System.Text;
using
Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Web.Mvc; using
MVCApplication2;
using MVCApplication2.Controllers;

namespace MVCApplication2.Tests.Controllers
{
    [TestClass]
    public class MapsControllerTest
    {
```

```

[TestMethod]
public void
ViewMaps()
{
    // Arrange
    MapsController controller = new MapsController();

    // Act
    ViewResult result = controller.ViewMaps() as ViewResult;

    // Assert
    Assert.IsNotNull(result);
}
}
}

```

VB

```

Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.Web.Mvc
Imports Microsoft.VisualStudio.TestTools.UnitTesting
Imports MVCApplication2

<TestClass()> _
Public Class MapsControllerTest
    Private testContextInstance As TestContext
    ""<summary>
    ""Gets or sets the test context which provides
    ""information about and functionality for the current test run.
    ""</summary>
    Public Property TestContext() As TestContext
        Get
            Return testContextInstance
        End Get
        Set(ByVal value As TestContext)
            testContextInstance = value
        End Set
    End Property
End Class

```

[End Property](#)

```
<TestMethod()> _
Public Sub ViewMaps()
    ' Arrange
    Dim controller As MapsController = New MapsController()

    ' Act
    Dim result As ViewResult = controller.ViewMaps()

    ' Assert
    Assert.IsNotNull(result)
End Sub
```

[End Class](#)

Этот код определяет модульные тесты для метода действия, который будет доработан позднее.

5. Выберите в **обозревателе решений** тестовый проект, а затем нажмите сочетание клавиш **CTRL+F5**, чтобы запустить модульное тестирование.

Тест завершится ошибкой, так как метод действия **ViewMaps** пока что создает исключение.

Добавление кода метода действия

Теперь добавим в класс **MapsController** код для метода действия **ViewMaps**, отображающий представление **Maps**.

Добавление кода метода действия

Откройте класс **MapsController** и замените заглушки метода действия **ViewMaps** на следующий код, отображающий представление **Maps**:

VB

```
Function ViewMaps() As ActionResult
```

Retu

rn

```

View
()
End
Func
tion
C#
public ActionResult ViewMaps()
{
    return View();
}

```

Сохраните и закройте файл.

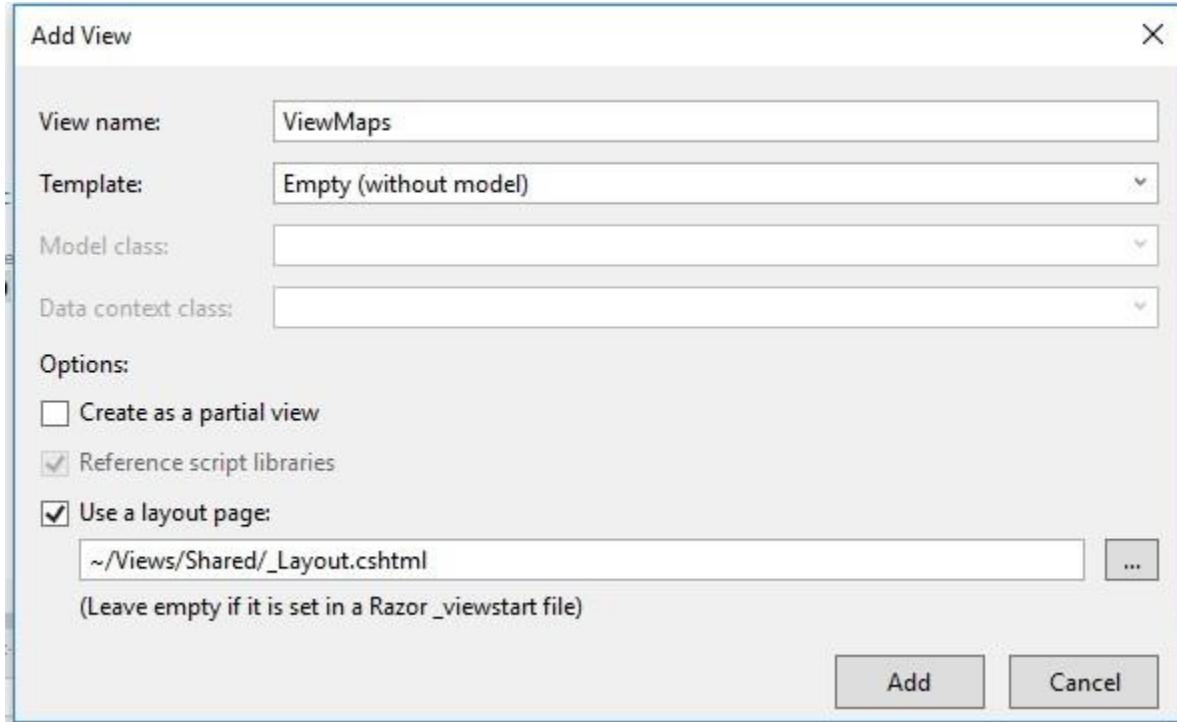
Добавление представления

Далее добавим представление **Maps**. Чтобы поддерживать порядок представлений, сначала добавим папку **Maps** в папке **Views**.

Добавление представления страничного содержимого в проект MVC

Откройте класс **MapsController**, щелкните правой кнопкой мыши внутри метода действия **ViewMaps** и выберите пункт **Добавить представление**.

Отобразится диалоговое окно **Добавление представления**.



В поле **Имя представления** введите **ViewMaps**.

Шаблон - Empty

Установите флажок **Выбрать главную страницу** и укажите главную страницу **~/Views/Shared/_Layout.cshtml**.

Нажмите кнопку **Добавить**.

Новое представление будет добавлено в проект в папке **Maps**.

Добавление содержимого в представление

Далее следует добавить содержимое к новому представлению.

Добавление содержимого в представление

Откройте файл **ViewMaps.aspx** и добавьте следующее содержимое внутри элемента **Content**:

```
<h2>My City Maps</h2>
```

Select map:

```
<select onclick="GetMap(value);">
    <option value="Minsk">Минск</option>
    <option value="Mogilev">Могилев</option>
```

```

</select><br />
<br />

<iframe id="frame1"
src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d15047
5.3322718522!2d27.
451567660957238!3d53.89305669809051!2m3!1f0!2f0!3f0!3m2!1i1024!2i
768!4f13.1!3m3!1m
2!1s0x46dbcf35b1e6ad3%3A0xb61b853ddb570d9!2z0JzQuNC90YHQug!
5e0!3m2!1sru!2sby!4v15
76007412668!5m2!1sru!2sby" width="400" height="300" frameborder="0"
style="border:0;" allowfullscreen=""></iframe>

<script charset="UTF-8" type="text/javascript"

src="http://dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6.2&mkt=e n-
us">
</script>
<script
type="text/javasc
ript">  var map
= null;  var
mapID = "";

function GetMap(mapID) {
    var a =
document.getElementById("frame1");
switch (mapID) {      case 'Minsk':
        a.src =
"https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d150475.3322718522!2d
27.45156766095
7238!3d53.89305669809051!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x
46dbcf35b1e
6ad3%3A0xb61b853ddb570d9!2z0JzQuNC90YHQug!5e0!3m2!1sru!2sby!4v157600741
2668!5m2!1sru!2sby
";
break;
case
'Mogilev':
a.src =
"https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d150517.9088211986!2d
30.21822307530

```

```

9055!3d53.88123104449437!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x
46d0521c5284
4571%3A0xcf85d14239bb73b6!2z0JzQvtCz0LjQu9GR0LI!5e0!3m2!1sru!2sby!4v1576
007883514!5m2!1sr u!2sby";
    break;
}
</script>

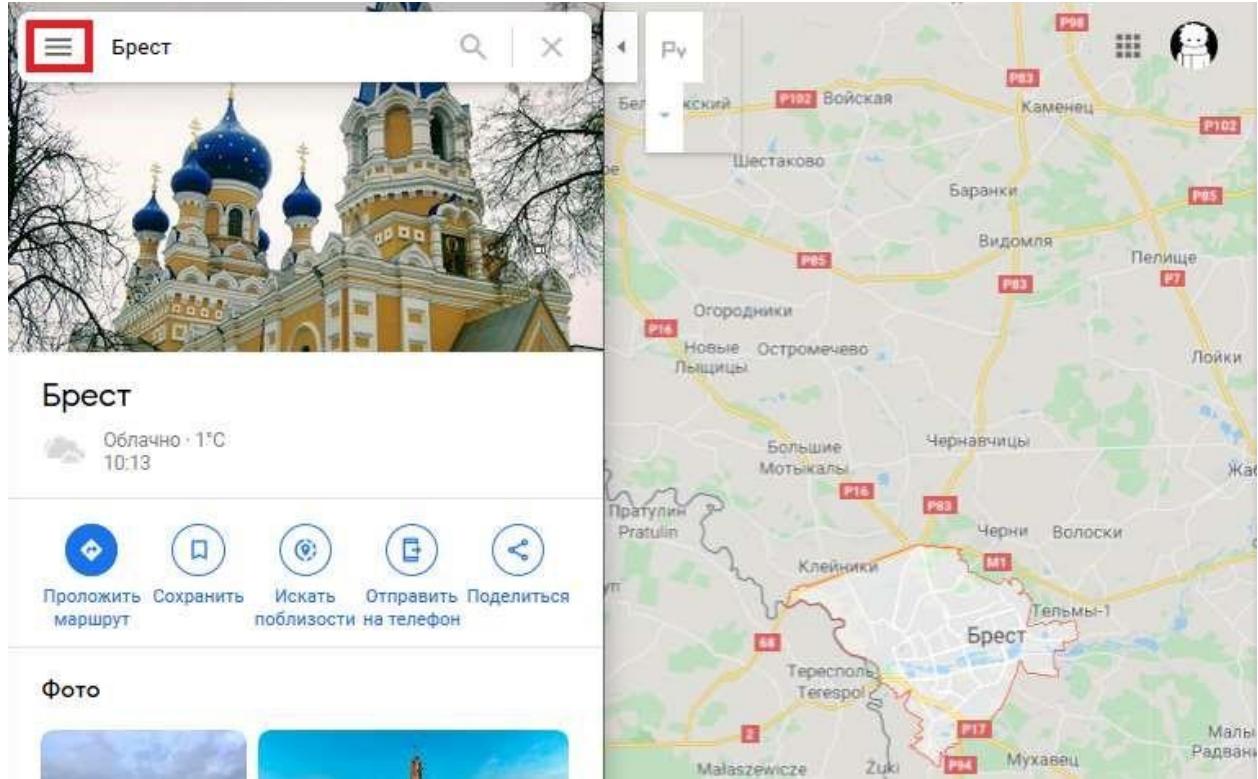
```

Эта разметка определяет раскрывающийся список выбора карты и код на JavaScript, реализующий извлечение выбранной карты через веб-службу Google Maps. Сохраните и закройте файл.

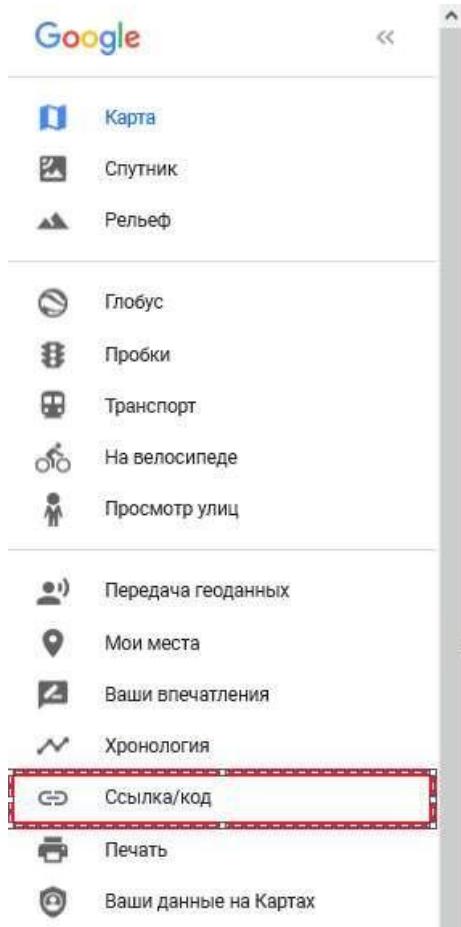
Получение iframe с Google Maps

Для того, чтобы добавить iframe с google картой, необходимо зайти на <https://www.google.com/maps>

В поиске находим необходимый город и нажимаем на кнопку, выделенную **красным**:



Находим пункт «Ссылка/код» и кликаем на него



Выбираем пункт встраивание карт. Далее можно изменить размер отображаемого фрейма (1) и отредактировать размер отображаемой области (2):

Поделиться

X

Отправка ссылки

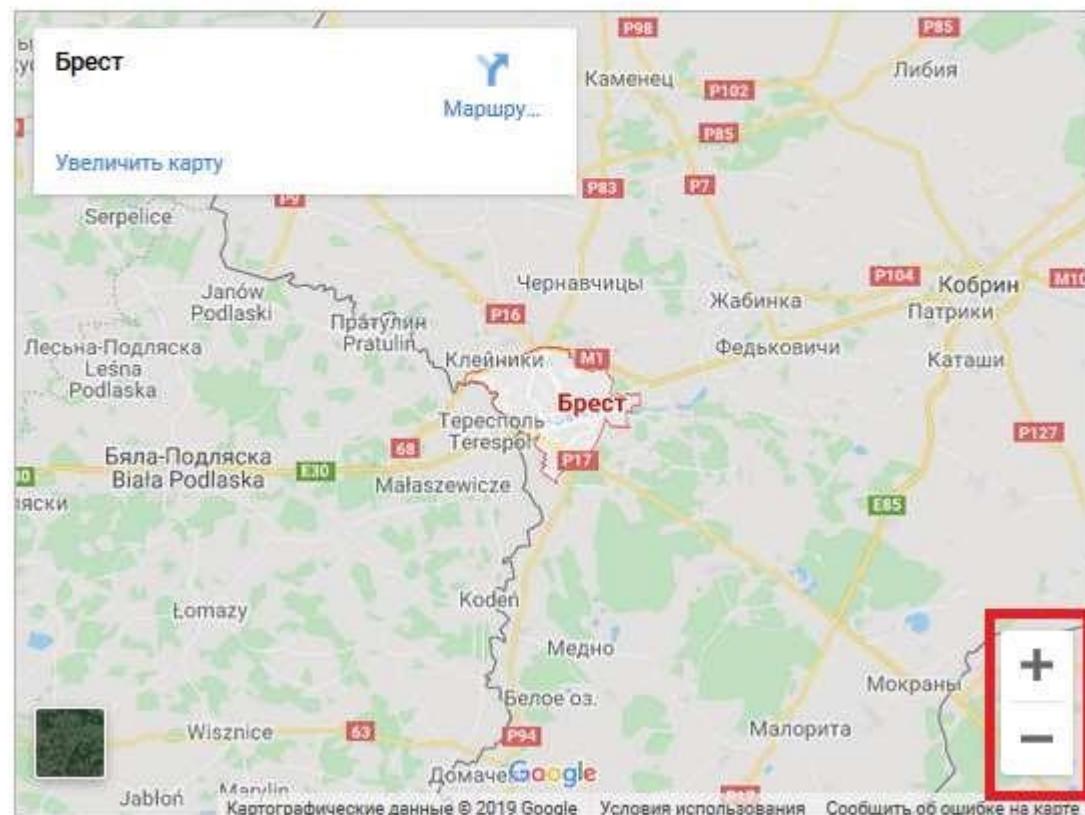
Встраивание карт

1

Средняя -

<iframe src="https://www.google.com/maps/embed?pb=!1m

КОПИРОВАТЬ HTML



Размещая карту на сайте, вы принимаете Условия использования.

Нажимаем

копировать

HTML

Поделиться

X

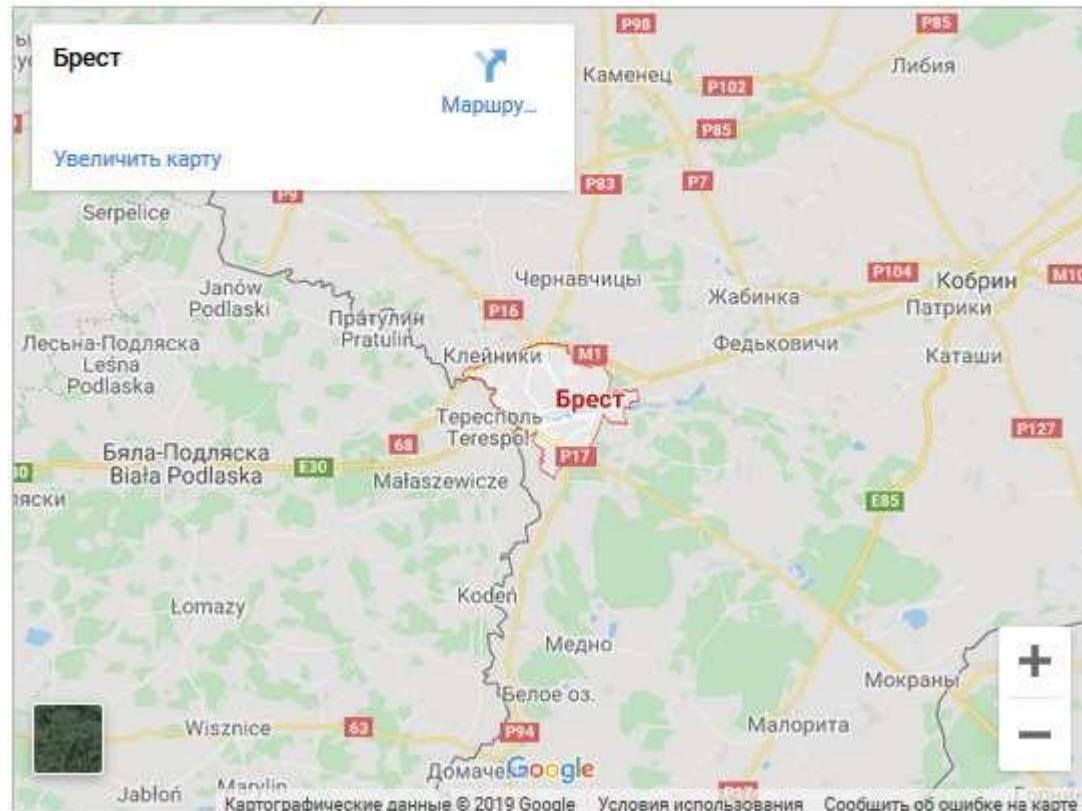
Отправка ссылки

Встраивание карт

Средняя

<iframe src="https://www.google.com/maps/embed?pb=!1m

КОПИРОВАТЬ HTML



Размещая карту на сайте, вы принимаете Условия использования.

И достаем значение атрибута «src» любыми возможными методами.

Изменим функцию GetMap следующим образом:

```
function GetMap(mapID) {
    var a =
        document.getElementById("frame1");
    switch (mapID) {           case 'Minsk':
        a.src =
            "https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d150475.3322718522!2d
27.45156766095
            7238!3d53.89305669809051!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x
46dbcf35b1e
```

```

6ad3%3A0xb61b853ddb570d9!2z0JzQuNC90YHQu!5e0!3m2!1sru!2sby!4v157600741
2668!5m2!1sru!2sby
";
break;
case
'Mogilev':
a.src =
"https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d150517.9088211986!2d
30.21822307530
9055!3d53.88123104449437!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x
46d0521c5284
4571%3A0xcf85d14239bb73b6!2z0JzQvtCz0LjQu9GR0LI!5e0!3m2!1sru!2sby!4v1576
007883514!5m2!1sr u!2sby";
break;
case 'Brest':
a.src =
"https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d78449.31346339424!2d
23.63284832996
4006!3d52.088084177221866!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0
x47210c02236
30975%3A0x4d319ea41f64ae99!2z0JHRgNC10YHRgg!5e0!3m2!1sru!2sby!4v1576008
086154!5m2!1sru!2s by" ;
break;
}
}

```

Теперь при выборе Бреста в списке отобразится нужная нам карта:

My City Maps

Select map: Брест ▾



© 2019 - мое приложение ASP.NET

По данному примеру добавьте карты следующих городов: Брест, Витебск, Гродно, Гомель.

Добавление вкладки в меню главной страницы

Теперь добавим в меню главной страницы элемент, вызывающий метод действия `ViewMaps`.

Добавление вкладки в меню главной страницы

В папке `Shared` откройте файл `_Layout.cshtml` и найдите элемент неупорядоченного списка (`ul`) в элементе `div`.

```

<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li>@Html.ActionLink("Home", "Index", "Home")</li>
        <li>@Html.ActionLink("About", "About", "Home")</li>
        <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
    </ul>
    @Html.Partial("_LoginPartial")
</div>

```

Добавьте в список следующий код между вкладками Index и About :

```
<li>@Html.ActionLink("My City Maps", "ViewMaps", "Maps")</li>
```

ActionLink – это вспомогательный метод, создающий ссылку на метод действия. Он принимает следующие параметры: текст ссылки, имя метода действия и имя контроллера.

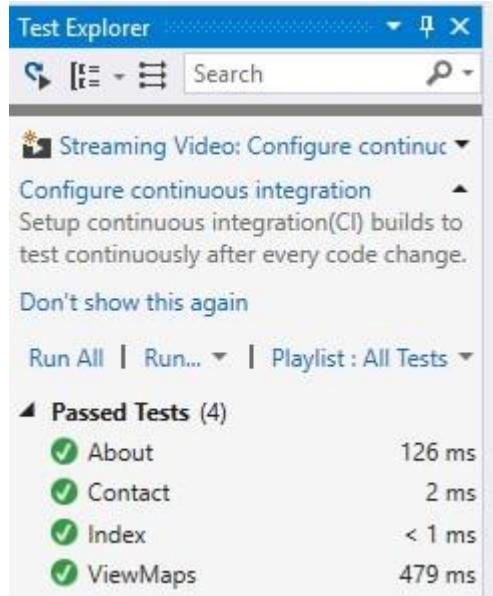
Сохраните и закройте файл.

Тестирование MVC-приложения

Теперь приложение можно протестировать.

Тестирование MVC-приложения

В меню Тест выберите команду Выполнить, а затем выберите Все тесты в решении.



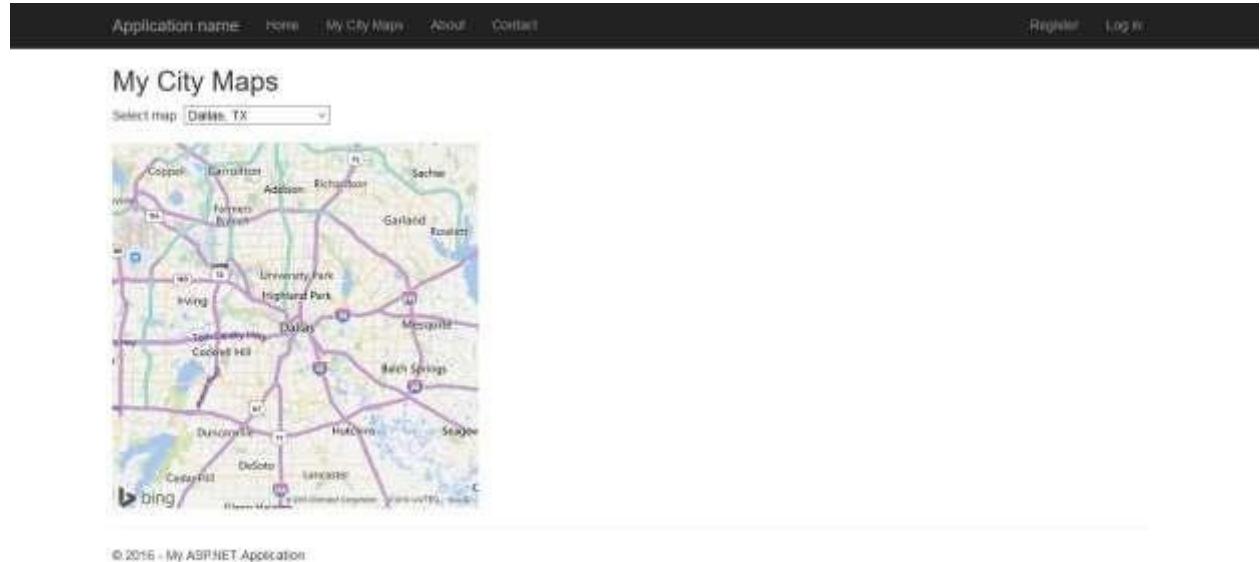
Результаты будут показаны в окне Результаты теста. Результат на этот раз положительный.

Выберите в обозревателе решений проект, созданный в данном руководстве, а затем нажмите сочетание клавиш **CTRL+F5**, чтобы запустить приложение.

Отобразится страница Index.aspx, на которой будут представлены вкладки, определенные на главной странице.

Щелкните вкладку **My City Maps**.

Отобразится страница **My City Maps**. Выберите любую карту, чтобы отобразить ее.



- 1) Как создать приложение Asp Net MVC Framework?
- 2) Что позволяет Asp Net MVC Framework?
- 3) Какой фреймворк применяется для работой с базой данных в Asp Net MVC Framework?
- 4) Опишите компоненты MVC(model view controller)?

Лабораторная работа №13

Функциональное программирование.

Цель работы: Приобрести навыки работы с интерпретатором языка Haskell. Получить представление об основных типах языка Haskell. Научиться определять простейшие функции.

1. Основы работы с интерпретатором Hugs

Для выполнения лабораторных работ будет использоваться интерпретатор языка Haskell. Существует несколько реализаций интерпретатора; в настоящем курсе будет использоваться интерпретатор Hugs (это название является аббревиатурой слов Haskell users' Gofer system. Gofer-название языка программирования, который был одним из предшественников Haskell).

После запуска интерпретатора Hugs (Hugs (Haskell98 mode)) на экране появляется диалоговое окно среды разработчика, автоматически загружается специальный файл предопределений типов и определений стандартных функций на языке Haskell (Prelude.hs), и выводится стандартное приглашение к работе. Это приглашение имеет вид:

Prelude>

Вообще, перед символом > выводится имя последнего загруженного модуля. Если приглашение интерпретатора изменено на **Main>**, значит загружен какой-то модуль по умолчанию. Дело в том, что если не указано имя модуля, считается, что оно равно Main. Перед началом работы следует выгрузить лишние модули. Для этого зайдите в меню **File -> Module Manager** и удалите файл с именем Hugs.hs. Если приглашение интерпретатора изменилось на **Prelude>**, значит вы все сделали верно.

После вывода приглашения можно вводить выражения языка Haskell, либо команды интерпретатора. Команды интерпретатора отличаются от выражений языка Haskell тем, что начинаются с символа двоеточия (:). Примером команды интерпретатора является команда **:quit**, по которой происходит завершение работы интерпретатора. Команды интерпретатора можно сокращать до одной буквы; таким образом, команды **:quit** и **:q** эквивалентны. Команда **:set** используется для того, чтобы установить различные опции интерпретатора. Команда **:?** выводит список доступных команд интерпретатора.

1. Типы

Программы на языке Haskell представляют собой выражения, вычисление которых приводят к значениям. Каждое значение имеет тип. Интуитивно тип можно понимать просто как множество допустимых значений выражения. Для того, чтобы узнать тип некоторого выражения, можно использовать команду интерпретатора **:type** (или **:t**). Кроме того, можно выполнить команду **:set +t**, для того, чтобы интерпретатор автоматически печатал тип каждого вычисленного результата.

Текст программы:

Prelude>:set +t

Prelude>1

1 :: Integer

Задание 1. По приведенному примеру проверьте, каких типов будут следующие данные:

- 50000
- 5.555
- ‘5’
- “5555”
- True

2. Арифметика

Интерпретатор Hugs можно использовать для вычисления арифметических выражений. При этом можно использовать операторы +, -, *, / (сложение, вычитание, умножение и деление) с обычными правилами приоритета. Кроме того, можно использовать оператор ^ (возвведение в степень). Таким образом, сеанс работы может выглядеть следующим образом:

Текст программы:

```
Prelude>2*2
4 :: Integer
```

Кроме того, можно использовать стандартные математические функции sqrt (квадратный корень), sin, cos, exp и т.д. В отличие от многих других языков программирования, в Haskell при вызове функции не обязательно помещать аргумент в скобки. Таким образом, можно просто писать sqrt 2, а не sqrt(2). Пример:

Текст программы:

```
Prelude>sqrt 2
1.4142135623731 :: Double
```

Задание 2. Выполните арифметические действия:

- $5^2 + 2 * 4 - 3$
- $\sqrt{9} - 1$
- $\sqrt{9 - 1}$
- 5^{5000}
- $2 * \cos \pi$

Операторы, определённые в стандартном модуле Prelude. Значения

ассоциативности и приоритета для этих операторов также указаны в таблице.

Символ	Значение	Тип	Ас.	Пр.
!!	Индекс	[a] -> Int -> a	Л	9
.	Композиция	(a -> b) -> (c -> a) -> c -> b	П	9
^	Экспонента	(Integral b, Num a) => a -> b -> a	П	8
^^	Экспонента	(Fractional a, Integral b) => a -> b -> a	П	8
**	Экспонента	Floating a => a -> a -> a	П	8
*	Умножение	Num a => a -> a -> a	Л	7
/	Деление	Rational a => a -> a -> a	Л	7
quot	Целое деление	Num a => a -> a -> a	Л	7
rem	Остаток	Num a => a -> a -> a	Л	7
div	Целое деление	Num a => a -> a -> a	Л	7
mod	Остаток	Num a => a -> a -> a	Л	7
:%	Дробь	Integral a => a -> a -> Ratio a	Л	7
%	Сокращение	Integral a => a -> a -> Ratio a	Л	7
+	Сложение	Num a => a -> a -> a	Л	6
-	Вычитание	Num a => a -> a -> a	Л	6
:	Создание списка	a -> [a] -> [a]	П	5
++	Конкатенация	[a] -> [a] -> [a]	П	5
/=	Неравенство	Eq a => a -> a -> Bool		4
==	Равенство	Eq a => a -> a -> Bool		4
<	Меньше	Ord a => a -> a -> Bool		4
<=	Меньше или равно	Ord a => a -> a -> Bool		4
>	Больше	Ord a => a -> a -> Bool		4
>=	Больше или равно	Ord a => a -> a -> Bool		4
elem	Существование	Eq a => a -> [a] -> Bool		4
notElem	Несуществование	Eq a => a -> [a] -> Bool		4
&&	Логическое И	Bool -> Bool -> Bool	П	3
	Логическое ИЛИ	Bool -> Bool -> Bool	П	2
>>	Связывание	m a -> m b -> m b	Л	1
>>=	Связывание	m a -> (a -> m b) -> m b	Л	1
=<<	Связывание	Monad m => (a -> m b) -> m a -> m b	П	1
\$	Стр. композиция	(a -> b) -> a -> b	П	0
\$!	Строгость	(a -> b) -> a -> b	П	0
seq	Строгость	a -> b -> b	П	0

3. Кортежи

В языке Haskell пару можно задать, перечислив компоненты через запятую и взяв их в скобки: (5,3). Компоненты пары не обязательно должны принадлежать одному типу: можно составить пару, первым элементом которой будет строка, а вторым- число и т.д. Например, пара (5,3) имеет тип (Integer, Integer); пара (1, 'a') принадлежит типу(Integer, Char). Можно привести и более сложный пример: пара

((1,'a'),1.2) принадлежит типу ((Integer,Char),Double). Проверьте это с помощью интерпретатора.

Кроме пар, аналогичным образом можно определять тройки, четверки и т.д. Их типы записываются аналогичным образом.

Для работы с парами в языке Haskell существуют стандартные функции fst и snd, возвращающие, соответственно, первый и второй элементы пары. Эти функции определены только для пары и не работают для других кортежей.

Задание 3. Зададим кортеж вида: (5, (“hello”, “world”), ’a’)

Извлеките из него:

- число 5;
- слово «hello»;
- символ ‘a’.

4. Списки

В отличие от кортежей, список может хранить произвольное количество элементов. Чтобы задать список в Haskell, необходимо в квадратных скобках перечислить его элементы через запятую. Все эти элементы должны принадлежать одному и тому же типу. В списке может не быть ни одного элемента. Пустой список обозначается как [].

Элементами списка могут быть любые значения- числа, символы, кортежи, другие списки и т.д.

Оператор : (двоеточие) используется для добавления элемента в начало списка. Его левым аргументом должен быть элемент, а правым- список:

Текст программы:

```
Prelude>1:[2,3]
[1,2,3] :: [Integer]
```

С помощью оператора (:) и пустого списка можно построить любой список:

Текст программы:

```
Prelude>1:2:3: []
[1,2,3] :: Integer
```

Для работы со списками в языке Haskell существует большое количество функций. Вданной лабораторной работе рассмотрим только некоторые из них.

Функция head возвращает первый элемент списка.

Функция last возвращает последний элемент списка.

Функция tail возвращает список без первого элемента

Функция `init` возвращает список без последнего элемента

Функция `null` проверяет список на пустоту. Если в качестве аргумента этой операции будет задан пустой список, то функция выдаст значение `True`, в противном случае `False`.

Функция `length` возвращает длину списка.

Функция `elem` проверяет наличие элемента в списке.

Функция `take` возвращает список, состоящий из `n` первых элементов исходного списка.

Функция `zip` возвращает список, состоящий из пар объединенных исходных списков.

Функция `!!` возвращает элемент, номер которого задан (начиная с 0).

Функции `head` и `tail` определены для непустых списков. При попытке применить их к пустому списку интерпретатор сообщает об ошибке.

Примеры работы с указанными функциями:

Текст программы:

```
Prelude>head [1,2,3]
1 :: Integer
Prelude>tail [1,2,3]
[2,3] :: [Integer]
Prelude>tail [1]
[] :: Integer
Prelude>length [1,2,3]
3 :: Int
Prelude>elem 2 [1,2,3]
True :: Bool
Prelude> take 2 [1,2,3]
[1,2] :: [Integer]
Prelude> zip ["20","30"] [1,2,3]
[("20",1),("30",2)] :: [([Char],Integer)]
Prelude> [1,2,3,4,5] !! 3
4 :: Integer
```

Заметьте, что результат функции `length` принадлежит типу `Int`, а не типу `Integer`.

Для соединения (конкатенации) списков в Haskell определен оператор `++`.

Задание 4. Приведите пример нетривиальных выражений, принадлежащих следующему типу:

- `((Char,Integer), String, [Double])`
- `[(Double,Bool),(String,Integer))]`
- `([Integer],[Double],[(Bool,Char)])`
- `[[[(Integer,Bool)]]]`
- `(([Double],[Bool]),[Integer])`

Требование нетривиальности в данном случае означает, что встречающиеся в выражениях списки должны содержать больше одного элемента.

5. Строки

Строковые значения в языке Haskell, как и в Си, задаются в двойных кавычках. Они принадлежат типу String.

Текст программы:

```
Prelude>"hello"
"hello" :: String
```

В действительности строки являются списками символов; таким образом, выражения "hello", ['h','e','l','l','o'] и 'h':'e':'l':'l':'o':[] означают одно и то же, а тип String является синонимом для [Char]. Все функции для работы со списками можно использовать при работе со строками:

Для преобразования числовых значений в строки и наоборот существуют функции read и show:

Текст программы:

```
Prelude>show 1
"1" :: [Char]
Prelude>"Formula " ++ show 1
"Formula 1" :: [Char]
Prelude>1 + read "12"
13 :: Integer
```

Если функция read не сможет преобразовать строку в число, она сообщит об ошибке.

6. Функции

До сих пор мы использовали встроенные функции языка Haskell. Теперь пришла пора научиться определять собственные функции. Для этого нам необходимо изучить еще несколько команд интерпретатора (напомним, что эти команды могут быть сокращены до одной буквы):

Команда :load позволяет загрузить в интерпретатор программу на языке Haskell, содержащуюся в указанном файле.

Команда :edit запускает процесс редактирования последнего загруженного файла.

Команда :reload перечитывает последний загруженный файл.

Определения пользовательских функций должны находиться в файле, который нужно загрузить в интерпретатор Hugs с помощью команды :load. Для редактирования загруженной программы можно использовать команду :edit. Она

запускает внешний редактор (по умолчанию это Notepad) для редактирования файла. После завершения сеанса редактирования редактор необходимо закрыть; при этом интерпретатор Hugs перечитает содержимое изменившегося файла. Однако файл можно редактировать и непосредственно из оболочки Windows. В этом случае, для того чтобы интерпретатор смог перечитать файл, необходимо явно вызывать команду :reload.

Рассмотрим пример. Создайте в каком-либо каталоге файл lab1.hs.

Пусть полный путь к этому файлу c:\labs\lab1.hs. В интерпретаторе Hugs выполните следующие команды:

```
Prelude>:load "c:\\labs\\lab1.hs"
```

Если загрузка проведена успешно, приглашение интерпретатора меняется на Main>.

Обратите внимание, что при записи имени файла в аргументе команды :load символы \ дублируются. Также, как и в языке Си, в Haskell символ \ служит индикатором начала служебного символа ('\' и т.п.) Для того, чтобы ввести непосредственно символ \, необходимо, как и в Си, экранировать его еще одним символом \.

Создайте, в соответствие с процессом, описанным выше, какой-либо файл и запишите в него следующий текст:

Текст программы:

```
square :: Integer -> Integer
square x = x * x
```

Загрузите созданный файл в интерпретатор и выполните следующие команды:

Текст программы:

```
Main>square 2
4 :: Integer
```

7. Условные выражения

В определении функции в языке Haskell можно использовать условные выражения.

Запишем функцию signum, вычисляющую знак переданного ей аргумента:

Текст программы:

```
signum :: Integer -> Integer
signum x = if x > 0 then 1
           else if x < 0 then -1
           else 0
```

else 0

Условное выражение записывается в виде: if условие then выражение else выражение. Выражения в then-части и в else-части условного оператора должны принадлежать одному типу.

8. Функции многих переменных и порядок определения функций

До сих пор мы определяли функции, принимающие один аргумент. Разумеется, в языке Haskell можно определять функции, принимающие произвольное количество аргументов. Определение функции add, принимающей два целых числа и возвращающей их сумму, выглядит следующим образом:

Текст программы:

```
add :: Integer -> Integer -> Integer
add x y = x + y
```

Задание 5. Напишите функцию, вычисляющую сумму квадратов двух чисел и вызовите ее с числами, например 2 и 3:

Текст программы:

```
f :: Int -> Int -> Int
f x y = x*x + y*y
```

Main> f 2 3

А теперь попробуйте такой вызов:

Main> f 2.3 4.2

Вы получите ошибку. Проблема заключается в том, что 4.2 это не Int.

Одним из вариантов решения будет не указывать явно тип функции f. Тогда Haskell выведет для нас наиболее общий подходящий тип:

Текст программы:

```
f x y = x*x + y*y
```

Main> f 2.3 4.2

В таком случае нам не надо создавать новую функцию для каждого типа данных.

Задание 6. Напишите функции, решающие следующие задачи:

1. Дано 2 числа. Проверить заданные числа на четность/нечетность.
2. По введенному номеру недели вывести наименование этого дня. Предусмотреть сообщение в случае некорректного ввода данных.
3. Функция возвращает произведение всех положительных чисел в списке.
4. Дано 3 числа. Упорядочьте их по возрастанию.
5. Дано 3 числа. Упорядочьте их по убыванию.
6. Даны два списка целых чисел. Объедините их попарно в один список.
7. Дан список целых чисел. Добавьте элемент в конец списка.
8. Реализуйте факториал числа.
9. Реализуйте число Фибоначчи.
10. Дан список чисел. Найдите количество чисел, которые больше 10.
11. Даны два списка целых чисел. Объедините их попарно и посчитайте сумму второй пары.
12. Дан список чисел. Найдите в нем сумму положительных чисел.
13. Дан список чисел. Найдите его длину.
14. Дан список чисел. Найти их произведение.
15. Дан список чисел. Найти в нем количество нулей.
16. По введенному номеру месяца вывести его наименование. Предусмотреть сообщение в случае некорректного ввода данных.
17. Дано 2 числа. Найти их НОД.

Лабораторная работа 13(2 часть)

Согласно своему варианту решите следующие задачи:

1. Конструирование конечных списков (N — количество элементов в списке).

1. Список натуральных чисел. $N = 20$.
2. Список нечётных натуральных чисел. $N = 20$.
3. Список чётных натуральных чисел. $N = 20$.
4. Список квадратов натуральных чисел. $N = 30$.
5. Список кубов натуральных чисел. $N = 30$.
6. Список факториалов. $N = 50$.
7. Список степеней двойки. $N = 25$.
8. Список степеней тройки. $N = 25$.
9. Список степеней пятерки. $N = 15$.
10. Список пятых степеней натуральных чисел. $N = 20$.
11. Список чисел Фибоначи. $N = 20$
12. Список степеней семерки. $N = 20$

2. Конструирование бесконечных списков.

1. Список факториалов.
2. Список чисел Фибоначи
3. Список квадратов натуральных чисел.
4. Список кубов натуральных чисел.
5. Список пятых степеней натуральных чисел.
6. Список степеней пятёрки.
7. Список степеней семёрки.
8. Список степеней двойки.
9. Список четных чисел
10. Список нечетных чисел
11. Список степеней тройки.
12. Список натуральных чисел

3. Функции работы со строками.

1. $GetN(L,n)$ — функция вычленения N -го элемента из заданного списка.
2. $ListSumm(L_1,L_2)$ — функция сложения элементов двух списков. Возвращает список, составленный из сумм элементов списков-параметров. Учесть, что переданные списки могут быть разной длины.

3. $OddEven(L)$ — функция перестановки местами соседних чётных и нечётных элементов в заданном списке.
4. $Reverse(L)$ — функция, обращающая список (первый элемент списка становится последним, второй — предпоследним, и так далее до последнего элемента).
5. $Map(F,L)$ — функция применения другой переданной в качестве параметра функции ко всем элементам заданного списка.
6. $ReverseAll(S)$ — функция, получающая на вход списочную структуру и обращающая все её элементы, а также её саму.
7. $Position(L,A)$ — функция, возвращающая номер первого вхождения заданного атома в список.
8. $Set(L)$ — функция, возвращающая список, содержащий единичные вхождения атомов заданного списка.
9. $Frequency(L)$ — функция, возвращающая список пар (символ, частота). Каждая пара определяет атом из заданного списка и частоту его вхождения в этот список.
10. $Insert(L,A,n)$ — функция включения в список заданного атома на определённую позицию.

4. Напишите функции, решающие следующие задачи:

1. Три точки А, В, С лежат на одной прямой. Заданы длины АВ, ВС, АС. Может ли точка А лежать между точками В и С.
2. Чему равен угол, если два смежных с ним угла составляют в сумме заданное число градусов.
3. Периметр равнобедренного треугольника равен Р метрам, а боковая сторона L. Найти основание треугольника.
4. Найти углы треугольника, если они пропорциональны заданным числам А, В, С.
5. В равнобедренном треугольнике боковая сторона А, а основание В. Найти высоту, опущенную на основание.
6. Даны координаты центров и радиусы двух окружностей на плоскости. Может ли вторая окружность целиком содержаться внутри первой? Если нет, то сколько точек пересечения имеют окружности?
7. Можно ли из отрезков с заданными длинами А, В, С построить прямоугольный треугольник?
8. Можно ли из круглого листа железа диаметром D метров вырезать квадрат со стороной А метров?
9. Стороны треугольника А, В, С. Найти высоту, опущенную на сторону С.
10. Высота равнобедренного треугольника Н метров, основание L. Найти углы треугольника и длину боковой стороны.
11. Найти точку, равноудаленную от осей координат и от точки с заданными координатами (x,y).

12. Даны четыре точки A, B, C, D на плоскости. Является ли четырехугольник ABCD параллелограммом?

13. Составить уравнение прямой, проходящей через 2 заданные точки.

14. Даны четыре точки A, B, C, D на плоскости. Является ли четырехугольник ABCD квадратом?

Контрольные вопросы:

1. В чем отличие команд интерпретатора от выражений языка Haskell?
2. Основные типы языка Haskell.
3. Функции для работы с кортежами.
4. Функции для работы со списками.
5. Условные выражения в языке Haskell.
6. Определение функций в языке Haskell.

Лабораторная работа №14

Разработка приложений методом TDD



DEV305ILL

Test Driven Development в Microsoft Visual Studio 2019

Введение

Test Driven Development (TDD), или Test Driven Design - методология разработки программного обеспечения, при которой разработчик сначала

пишет модульный тест, а затем добавляет программный код, функциональных возможностей которого достаточно для прохождения данного теста. Модульный тест можно рассматривать как небольшую спецификацию поведения системы, написание которого позволяет разработчику сосредоточиться на добавление кода, достаточного только для прохождения данного теста, тем самым обеспечивается плотность, легкость системы и добавляемые функциональные возможности полностью соответствуют требованиям документации.

При разработки программного обеспечения, следуя методологии TDD, принято придерживаться последовательности действий, которую сокращенно можно описать в виде 3 слов "Красный, Зеленый, Рефакторинг". Красный символизирует первоначальный этап цикла разработки, когда есть только тело теста, но нет соответствующих реализаций в тестируемом классе, как следствие тест проваливается. Зеленый – этап, когда в тестируемый класс добавлен функционал, достаточный для прохождения данного теста. Рефакторинг – этап

Примечание: В рамках данной лабораторной работы не ставится задачи научить вас работать по методологии TDD, вместо этого описывается ее поддержка в среде разработки Microsoft Visual Studio 2019. В качестве отправной точки для получения информации о данной методологии разработки и ее преимуществах вы можете использовать книгу:

Test Driven Development в Microsoft .NET Джеймсом Ньюкирк и Алексеем Voronstov, ISBN 0735619484

модификация кода с целью его оптимизации, повышения гибкости. Во время разработки данная последовательность действий многократно повторяется.

Одним из важнейших аспектов TDD является темп разработки. Для продуктивного темпа разработки необходимо не только на достаточном уровне понимать данную методологию, но и уметь использовать инструменты, которые помогают разработчику работать быстрее и эффективнее.

В Visual Studio 2019 внесен ряд усовершенствований, избавляющих разработчика от выполнения рутинных повторяющихся действий и дающих возможность больше сосредоточиться на решение задачи – написание высококачественного кода. В последующих упражнениях вы познакомитесь с новыми возможностями, которыми TDD разработчик может использовать для повышения своей эффективности. VS 2019

способствует повышению вашей эффективности, сокращая количество нажатий клавиш для выполнения часто используемых действий, ускоряет навигацию по вашему решению и позволяет использовать в качестве тестового фреймворка не только MSTest, но и другие.

Демонстрация новых возможностей будет проводиться в рамках разработки калькулятора (**SimpleStack** класс) в *code first* стиле.

Примечание: В информатике, стек - способ организации хранения данных, основанный на принципе последним пришел, первым ушел (LIFO).

Если вы не знакомы с данной структурой, то вам достаточно прочитать любую иззнакомительных статей, например на Wikipedia.

Цели

Изучить unit-тестирование и методологию TDD на практике.

3. Инструменты и платформа

В качестве среды разработки будем использовать Microsoft Visual Studio 2019, а в качестве платформы .NET Framework

4. Упражнение 1: Красный, Зеленый...

В рамках данного упражнения вы познакомитесь с возможностями Visual Studio 2019, которые способствуют разрабатывать по методологии TDD.

Задание 1 – Создание нового проекта

Создайте C# проект, а также добавьте тестовый проект, который будет использоваться для изучения новых возможностей Visual Studio 2019 в контексте методологии TDD.

1. В открывшейся Visual Studio на стартовой странице нажмите кнопку **Создание проекта**.

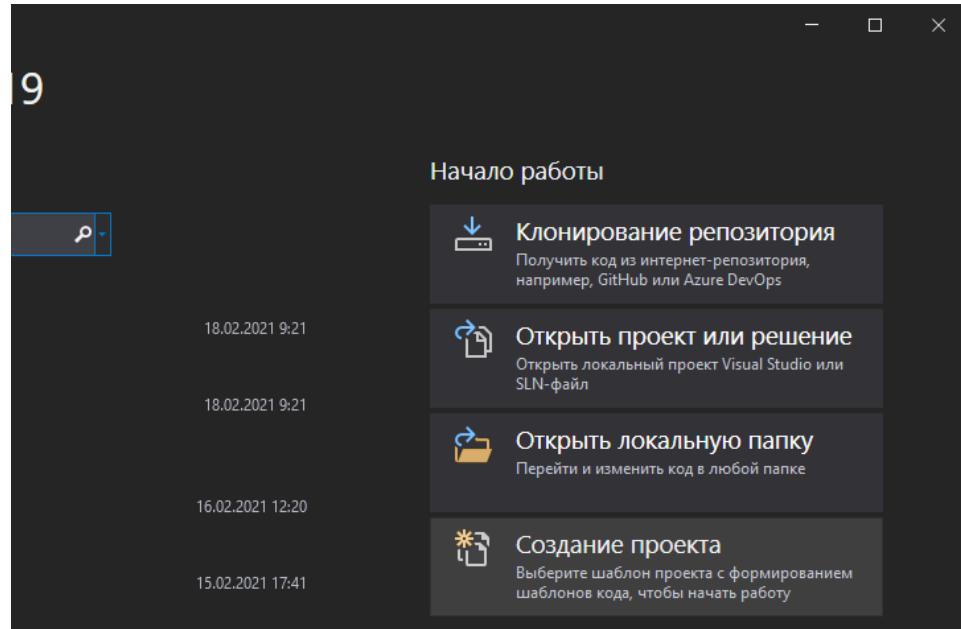


Рисунок 1 Создание нового проекта.

2. В диалоговом окне **Создание проекта** выберите тип проекта **Проект модульного теста(.NET Framework)**.

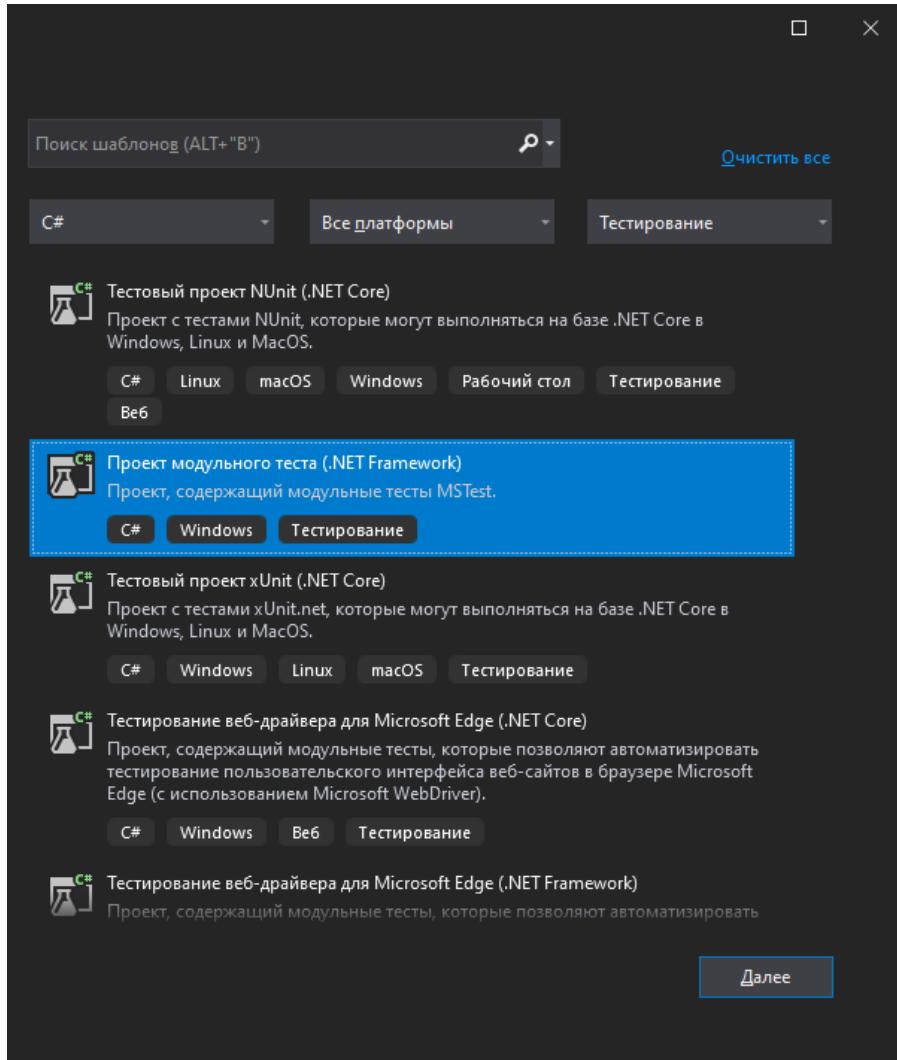


Рисунок 2 Тип проекта «Проект модульного теста».

3. В качестве имени проекта задайте **Calculator**, затем нажмите кнопку **OK**.

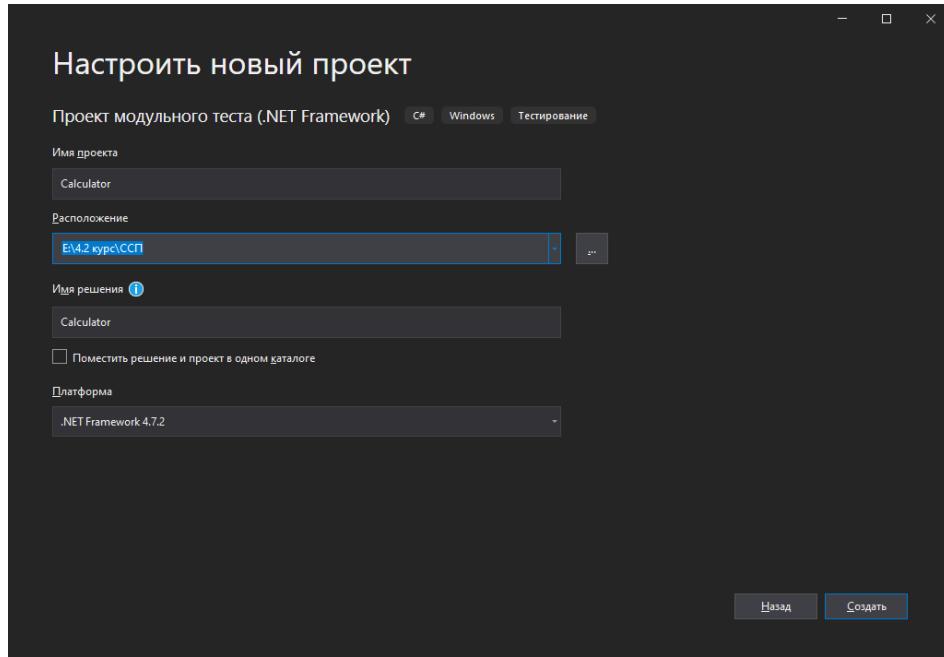


Рисунок 3 Название проекта «Calculator»

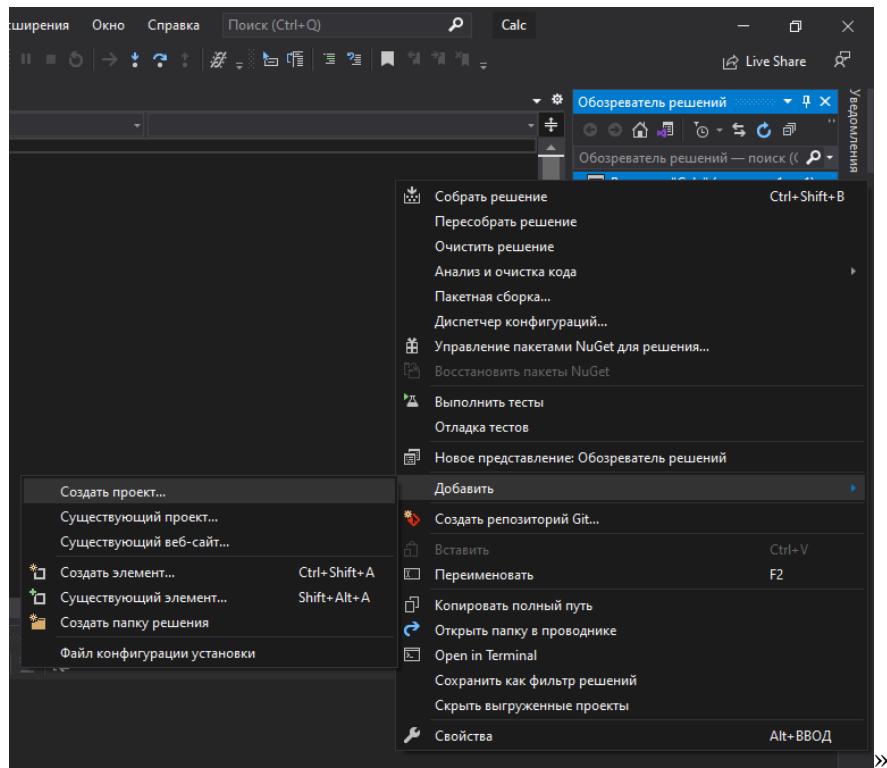


Рисунок 4 Создание нового проекта в решении.

4. В Solution Explorer, правой кнопкой мыши кликните на решение **Calculator** и в контекстном меню выберите **Add | New Project...**

5. После чего в диалоговом окне выберите тип проекта **Библиотека классов**. Убедитесь, что выбран **.NetFramework**.

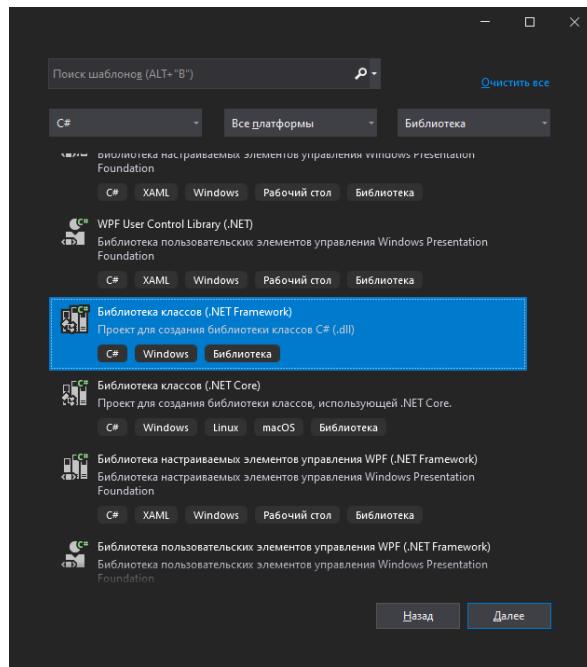


Рисунок 5 Тип проекта «Библиотека классов».

6. В качестве имени проекта задайте **ClassLibrary1** и нажмите на кнопку **OK**.

7. Теперь добавим ссылку на проект **ClassLibrary1** в проект **Calculator**.

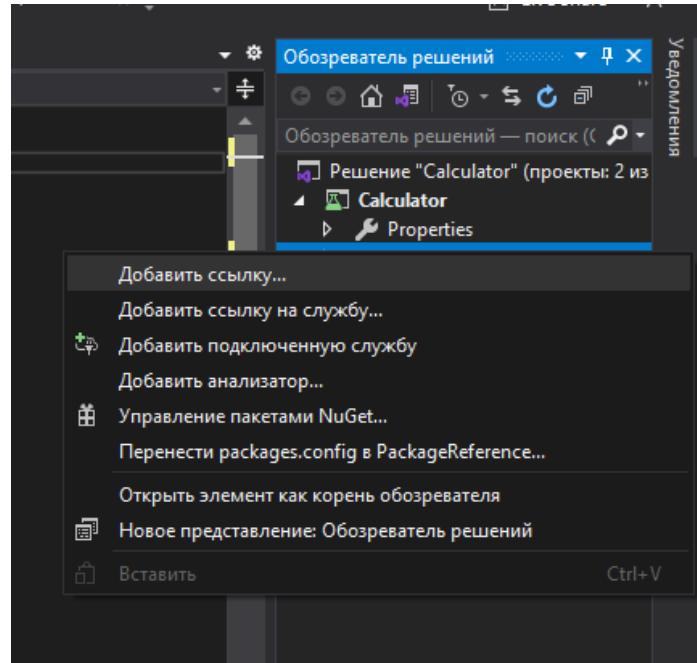


Рисунок 6 Добавление ссылки.

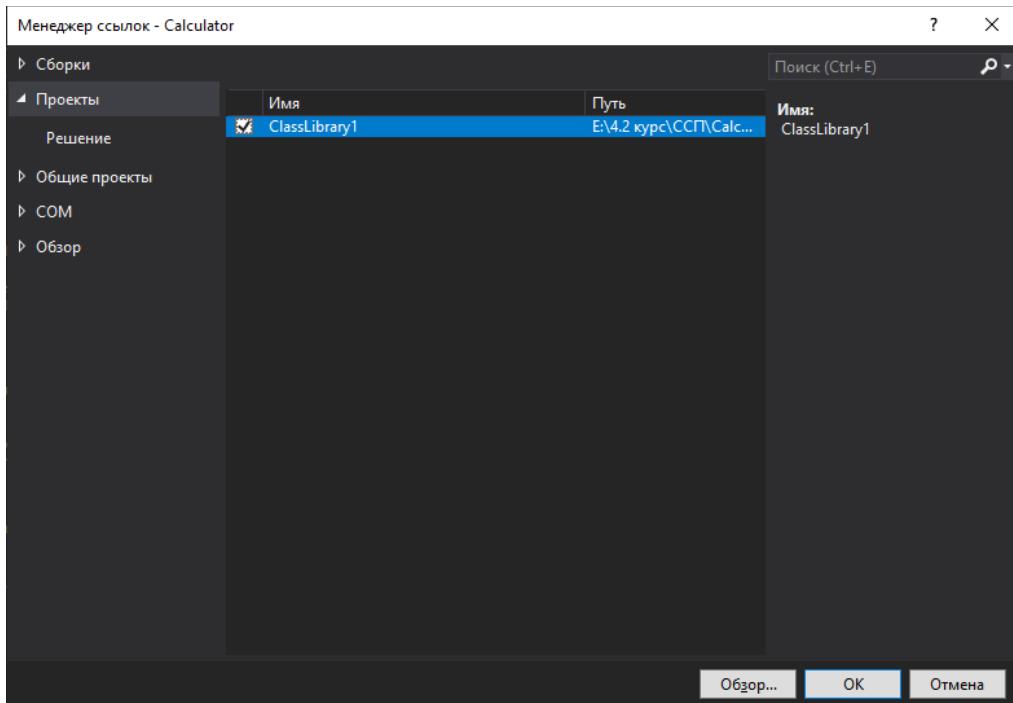


Рисунок 7 Добавление ссылки.

8. После добавления ссылки в проект **Calculator**, нужно добавить `using ClassLibrary1` в начало проекта.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
```

```
using System;
```

```
using ClassLibrary1;
```

9. Заполним ClassLibrary1 следующими методами:

```
public int Add(int v1, int v2)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

```
public int Substract(int v1, int v2)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

```
public int Multiply(int v1, int v2)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

10. Добавим в проект **Calculator** следующие тесты:

```
[TestMethod]
```

```
public void TestMethodAdd()
```

```
{
```

```
    Calculate obj = new Calculate();
```

```
    int result = obj.Add(10, 10);
```

```
    Assert.AreEqual(20, result);
```

```
}
```

```
[TestMethod]  
public void TestMethodSubstr()  
{  
    Calculate obj = new Calculate();  
    int result = obj.Substract(10, 10);  
    Assert.AreEqual(0, result);  
}  
  
[TestMethod]  
public void TestMethodMultiply()  
{  
    Calculate obj = new Calculate();  
    int result = obj.Multiply(10, 10);  
    Assert.AreEqual(100, result);  
}
```

11. Запустите тесты, убедитесь, что они провалились.

12. Изменим код в проекте ClassLibrary1 на следующий:

```
public int Add(int num1, int num2)  
{  
    return num1 + num2;  
}  
  
public int Substract(int num1, int num2)  
{  
    return num1 - num2;  
}  
  
public int Multiply(int num1, int num2)  
{
```

```

        return num1 * num2;
    }
}

```

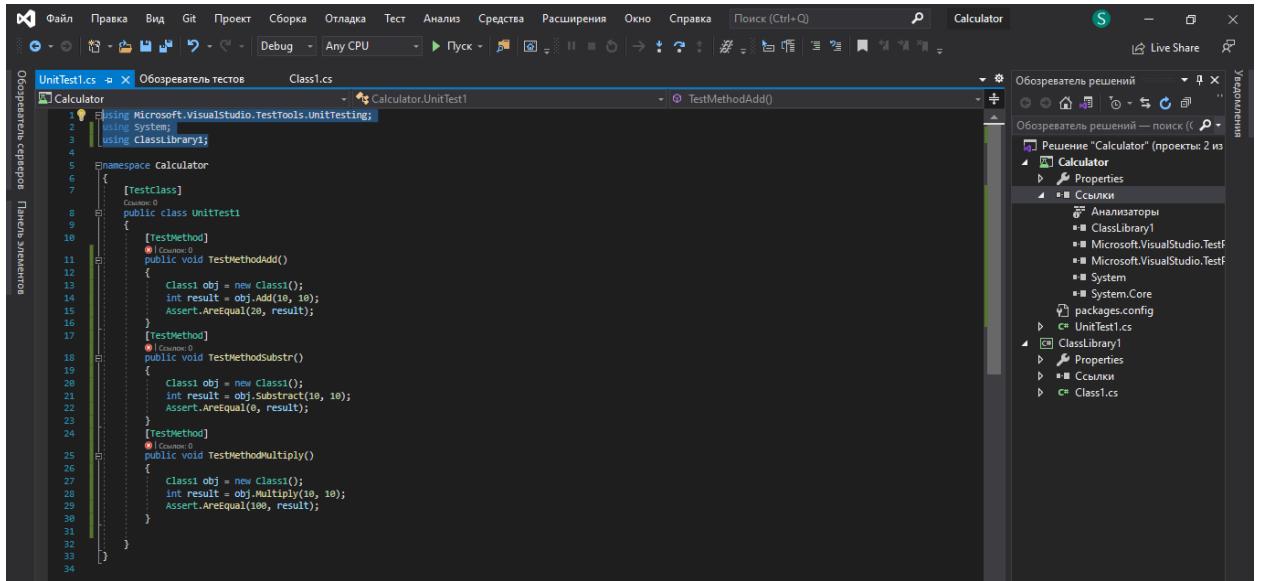


Рисунок 8 Тесты в проекте «Calculator»

11. Снова запустите тесты, убедитесь, что они прошли.

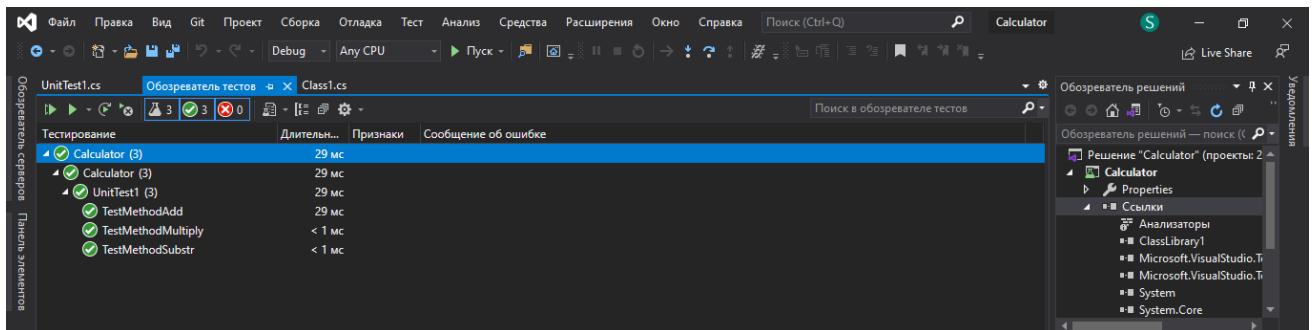


Рисунок 9 Тесты прошли

Вариант	Задание
1	Дан текст. Сделать заглавной каждую букву каждого слова, начинаящегося с заглавной буквы.
2	Дан текст. В каждом слове текста заменить заданную литеру за-данной литературой (сочетанием литер). Пример : Заменяемая литера : “б”, заменяющее сочетание литер : “ку”, слово : “абракадабра”, результат : “акуракадакура”.
3	В каждом слове удалить литеру, стоящую между двумя заданными.
4	Сформировать список, информирующий о вхождении заданной литеры в текст в виде ($<0\ 1\ 5\ 2\ 0>$) ($<3\ 0\ 1\ 5\ 2\ 0\ 1\ 0>$...). Цифры указывают количество вхождений литеры в каждое слово предложения.
5	Дан текст. Заменить в каждом предложении все вхождения за-данного слова на заданное новое слово.
6	Дан текст. Удалить из каждого слова в каждом предложении все повторяющиеся литеры.
7	Дан текст. В каждом слове каждого предложения для повторяющихся литер произвести следующую замену : повторные вхождения литер удалить, к первому вхождению литеры приписать число вхождений литеры в слово. Пример : ‘((aaabb cccddd)(eeefggg hhkl)) преобразуется в ‘((a3b2 c4d3)(e3fg3 h2kl))
8	Дан текст. В каждом слове вставить после заданного 3-буквенного сочетания заданное 2-буквенное.
9	Дан текст. Вставить заданное новое слово после каждого вхождения другого заданного слова.
10	Дан текст. Записать каждое предложение текста в порядке возрастания количества гласных букв в слове.
11	Дан текст. Переписать каждое предложение, расположив слова в обратном алфавитном порядке.
12	Написать программу, которая в каждом слове исходного текста меняет местами первую и последнюю буквы.

Разработать программный код на языке C# с методом для решения задачи по своему варианту. Разработать UNIT-test для проверки работы метода.

Контрольные вопросы:

- 1) Что такое Test Driven Development?
- 2) Что позволяет TDD?
- 3) Что такое стек?
- 4) Каково значение последовательности действий, которую сокращенно можно описать в виде 3 слов "Красный, Зеленый, Рефакторинг?"

Лабораторная работа № 15

Анализ предметной области. Разработка диаграммы вариантов использования

1. Цель работы:

разработка интерфейса информационной системы.

2. Порядок выполнения работы:

1. Выполнить сбор сведений по проектируемой системе и анализ предметной области.
2. Разработать функциональные и нефункциональные требования к системе.
3. Оформить документ описания системы по шаблону ПРИЛОЖЕНИЯ 1 (Шаблон Вигерса).
4. Изучить основные теоретические положения по разработке функциональной спецификации.
5. Спроектировать интерфейс информационной системы в соответствии с функциональными требованиями, разработанными в п. 2.
6. Сделать электронный отчет по образцу, приведенному в конце методички, а также внести в него цель работы.
7. Защитить отчет у преподавателя.

3. Основные теоретические положения:

Функциональная спецификация - формальное описание, которое объясняет, что и как будет делать программа. Она достаточно детально показывает строение всех модулей и их взаимодействие с учетом проектных ограничений. Спецификация невозможна без четкого описания структур данных программы.

Функциональная спецификация разрабатывается под руководством и при непосредственном участии архитектора (бизнес – аналитика) проекта.

Функциональная спецификация может сильно изменяться от проекта к проекту. В крупных комплексных проектах спецификации имеют несколько уровней детализации. Первоисточником для разработки функциональных спецификаций является **Техническое задание (Customer Requirements Specification)**. В этом документе описываются требования к программному продукту.

На *верхнем уровне* специфирования программного продукта разрабатывается **Внешняя спецификация** для заказчика (*Software Specification Document*). Это документ, объясняющий в бизнес–терминах, что должна делать

система. Документ разрабатывается с ориентацией на пользователя и, соответственно, должен отражать все его интересы. Документ не должен быть перегружен техническими подробностями. Пользователю интересно, какие меню, экраны и отчеты будут представлены в программе, и как программа будет осуществлять переходы из одной точки в другую. В зависимости от уровня подготовленности заказчика спецификация может иметь различную степень детализации.

На втором уровне может быть разработан концептуальный документ, описывающий *Архитектуру системы (Software Architecture Document, height level design)*. Эта спецификация показывает функционирование всей системы в целом, не детализируя устройство отдельных модулей. Она представляет структуру объектов и их зависимости. Спецификация дает профессиональному быстрое понимание организации системы и функционирования компонент, как части общей системы. Инструментами для описания архитектуры являются такие средства: UML, DFD, ERD и т.п.

На заключительном этапе разрабатывается *Техническая спецификация (Detail Design Document, low level design)*. Этот документ является завершающим в цепочке спецификаций и позволяет полностью сосредоточиться на стадии кодирования системы. Спецификация описывает низкоуровневую организацию продукта. Здесь для каждого модуля разработчик должен определить все требования, включая передаваемые параметры, глобальные структуры и переменные, вызываемые подпрограммы и т.д. Эта информация важна для кодировщиков, которые параллельно реализуют различные модули, взаимодействующие друг с другом. Хорошо выполненная техническая спецификация снимает все проблемы, возникающие при объединении отдельных модулей в единое целое. Вторая часть низкоуровневого проектирования - создание псевдокода - является трудной, но очень важной частью процесса. Программисты не любят писать псевдокод, поскольку им кажется, что непосредственное кодирование осуществить быстрее. Часто это действительно так. Для маленьких процедур и модулей код с размером в несколько строк написать проще, при этом не надо тратить время на представление одних и тех же действий дважды. Однако для крупных проектов с большой командой усилия, затраченные на эту работу, с лихвой окупятся на этапе сопровождения продукта, делая его более понятным, а, следовательно, модифицируемым с меньшими затратами.

Задача разработки спецификаций проекта считается выполненной, если пользователь четко сформировал свои ожидания к результирующему продукту, а программист способен однозначно реализовать эти ожидания в продукте без любых других знаний о проекте, руководствуясь только спецификациями.

Рекомендации по оформлению ИНТЕРФЕЙСА

Ориентация на пользователя.

При проектировании интерфейса необходимо мыслить как пользователь, а не как программист. Это не так просто, потому что знание внутреннего устройства программы из головы не выкинешь. Но обязательно нужно попробовать поставить себя на место пользователя, сделать набросок интерфейса, который мог бы пользователя удовлетворить, и только тогда браться за реализацию.

Кнопки

Кнопки бывают нескольких видов:

- кнопки прямого действия (командные кнопки), запускают какое-то действие;
- кнопки доступа к меню, название говорит само за себя;
- чекбоксы и радиокнопки, позволяют пользователю делать выбор из множества вариантов.

Другие возможные нестандартные решения так или иначе можно отнести к одному из этих трех основных видов кнопок (к примеру, [umbrUI — CSS3 range slider, checkbox + radio button](#) — нестандартная реализация стандартных кнопок).

Из неформального определения командной кнопки следует, что лишь по нажатию оной может быть инициировано действие, и ни в коем случае не по нажатию чекбоксов или радиокнопок.

Размер:

- На размер кнопки распространяется закон Фиттса — чем больше кнопка, тем легче в нее попасть курсором. Из этого следует, что кнопки надо делать большими. Однако это не всегда правильно, в некоторых случаях оправдано использование маленьких кнопок.
- Помимо этого, размеры нескольких кнопок, расположенных в одном окне, должны быть равными или хотя бы пропорциональными.

Положение:

- Нежелательно размещать кнопки впритык, лучше оставлять между ними пустой промежуток, потому как одно дело, когда пользователь промахивается мимо кнопки, и совсем другое — если, промахнувшись, он еще и ошибочно нажимает на другую кнопку.
- Совершенно неприемлемо удалять из видимости те кнопки, на которые нажать нельзя, взамен этого их необходимо делать неактивными, иначе пользователь будет теряться («клянусь, эта кнопка только что была здесь!»).
- В группе не должно быть менее двух радиокнопок (это очевидно, однако, бывают случаи...). Помимо этого, как минимум одна радиокнопка должна быть отмечена по умолчанию, об этом частенько забывают.
- Желателен вертикальный порядок чекбоксов и радиокнопок, приемлемо и расположение в две-три колонки, но никак не в разноброс. Пользователь должен сразу видеть, какие кнопки к какой группе относятся.

Текст:

- Надпись на кнопке по возможности должна быть в виде глагола в форме инфинитива (Прийти, Увидеть, Нажать) и как можно более точно отражать суть.
- Более внимательно нужно отнестись к использованию к кнопки «OK» и по возможности заменять «OK» на соответствующий глагол.
- Желательно совсем отказаться от использования кнопки «Применить», особенно в наборе OK-Применить-Отмена. Такое сочетание кнопок порождает некоторые неопределенности у пользователя, не будем обсуждать какие.
- На кнопке доступа к меню должна быть какая-нибудь индикация о том, что она именно отображает меню (самый лучший вариант — стрелочка вниз/вбок).
- Подписи к кнопкам не должны содержать отрицание во избежание возможных заблуждений пользователя.
- Пользователь скажет вам отдельное спасибо, если подписи к чекбоксам и радиокнопкам так же будут реагировать на нажатие.

Списки

- Как и в случае с чекбоксами и радиокнопками, списки ни в коем случае не должны инициировать какое-либо действие, чем обычно грешат сайты с

выбором города, языка и прочего, которые тут же перекидывают на другую страницу.

- Если в списке одиночного выбора присутствует «пустой» элемент, то это не должна быть пустая строка, следует применять мета-элемент «ничего».
- В списке множественного выбора желательно присутствие мета-элемента «все значения».
- По возможности элементы списка должны быть каким-либо образом отсортированы, если не по типу, то хотя бы по алфавиту.

Размер:

- Ширина списка должна быть достаточна как минимум для того, чтобы пользователь мог различить его элементы.
- Высота списка не должна превышать 7-8 строк. Такое количество элементов легче запоминается.

Поля ввода

Размер:

- Ширина поля должна соответствовать объему вводимого текста.
- Так же ширина поля не должна быть больше максимальной длины строки. Если пользователь ввел в поле максимальное количество символов, и наблюдает пустое пространство, то он может подумать, что где-то ошибся, не надо вводить пользователя в заблуждение.

Текст:

Расположение подписи к полю ввода — больная тема. Но вы не ошибетесь, если расположите ее сверху или слева от поля. Другие ухищрения редко могут дать прибавку к юзабилити и привлекательности интерфейса.

Меню

Текст:

- Название меню должно быть самым эффективным из всех возможных. Идеальный вариант — название в одно слово.

- Если элемент меню вызывает диалоговое окно, то к названию необходимо присоединить троеточие "...".
- Переключающиеся элементы лучше всего отмечать галочкой, нежели менять надпись элемента. Меню не должно меняться с течением времени.

Пиктограммы:

Самой распространенной ошибкой является наделение всех до одного элементов меню пиктограммами. Снабжать пиктограммами следует наиболее важные элементы меню, и то в пределах разумного. Вообще, лучше чтобы количество элементов с пиктограммой не превышало половину числа всех элементов. Чаще всего пользователь ориентируется в меню именно по пиктограммам («нужный мне элемент находится под синенькой пиктограммой, а другой — между теми двумя зелеными»). И если пиктограмм будет в избытке, то пользователь не будет смотреть на них в принципе, так как они потеряют свойства ориентирования, и придется читать надписи.

Группировка:

- Всегда группируйте элементы меню, не стоит бояться чрезмерного использования разделителей. Вы только поможете пользователю быстрее ориентироваться в вашем меню.
- Группировать элементы следует максимально логично, причем исходя из логики пользователя, а не программиста.
- Взаимоисключающие элементы желательно помещать в отдельный уровень иерархии.

Контекстное меню:

- Контекстное меню не должно быть единственным способом вызова функций.
- Меню желательно делать не особо длинным, 7-8 элементов.
- В контекстном меню особенно важен порядок — первыми должны быть более релевантные элементы.

Прочее

- Горизонтальные полосы прокрутки — это очень плохо. Пользователь их не любит.
- В строке заголовка окна первым должно идти имя документа, а лишь затем — приложения. Ярким примером служит Microsoft Word 2003: «Microsoft

Word — Document1.doc», и при большом количестве открытых окон в панели задач не будут различимы наименования документов.

- Строку статуса необходимо использовать либо как индикатор состояния системы, либо как панель инструментов для опытных пользователей. Худшим вариантом является использование строки статуса в качестве подсказок при наведении на тот или иной элемент окна.
- Панель инструментов нежелательно делать единственным способом вызова функций.

Отклик системы:

- Если системой совершается длительная операция, курсор необходимо сменить на «курсор с часиками». В любом случае, пользователя необходимо уведомить, что система что-то делает, а не пропадает.
- При более длительных операциях следует отвлечь внимание пользователя. К примеру, полосой прогресса, удивительно, но она ускоряет время. Так же можно использовать какой-либо звук (например, в пасьянсах при раздаче слышен шелест карт).
- Когда система завершила длительную операцию, об этом нужно уведомить пользователя, например «пропищать».

Интуитивный интерфейс — это знакомый интерфейс. Единообразие приложений очень важно. Если пользователь, к примеру, привык сохранять документ на сочетание Ctrl+S, то не надо в своем приложении учить его новым сочетаниям клавиш.

Образец проектирования интерфейса информационной системы:

Структура формы документа «Журнал успеваемости»					
Журнал успеваемости					
Группа:	Выпадающий список «Выберите группу:»				
Студент:	Выпадающий список «Выберите студента:»				
Семестр:	Выпадающий список «Выберите семестр:»				
Семестр	Дисциплина	Форма контроля	Оценка	Преподаватель	Дата

Кнопка «Экспорт в Excel»

ПРИЛОЖЕНИЕ 1**Проектирование информационной системы (рекомендуемые этапы)**

1.....	Бизнес-тре
1.1. Исходные данные	158
1.2. Возможности бизнеса	158
1.3. Бизнес-цели и критерии успеха	158
1.3.1. Финансовые бизнес-цели	158
1.3.2. Нефинансовые бизнес-цели	159
1.4. Потребности клиентов или рынка	160
1.5. Бизнес-риски.....	160
2.....	Образ
2.1. Положение об образе проекта.....	160
2.2. Основные функции	161
2.3. Предположения и зависимости.....	161
3.....	Масштабы и ограничени
3.1. Объем первоначальной версии	162
3.2. Объем последующих версий	163
4.....	Бизнес
4.1. Профили заинтересованных лиц	163
4.2. Приоритеты проекта	164
4.3. Операционная среда.....	165
5.....	Контекстная д

Бизнес-требования

Бизнес-требования описывают основные преимущества, которые новая система даст ее заказчикам, покупателям и пользователям. Для различных типов продуктов — информационных систем, коммерческих пакетов ПО и систем контроля, работающих в режиме реального времени, — выделяются различные преимущества.

Исходные данные

Суммирует обоснование и содержание нового продукта. Здесь помещают общее описание предыстории или ситуации, в результате которых было принято решение о создании продукта.

Возможности бизнеса

Для коммерческого продукта описывают существующие рыночные возможности и рынок, на котором продукту придется конкурировать с другими продуктами. Для корпоративной информационной системы описывают бизнес-проблему, которая разрешается посредством этого продукта, или бизнес-процессы, для улучшения которых требуется продукт, а также среду, в которой система будет использоваться. Кроме того, приведите здесь сравнительную оценку существующих продуктов и возможных решений, указав, в чем заключается привлекательность продукта и его преимущества. Опишите проблемы, которые не удается разрешить без продукта. Покажите, насколько он соответствует тенденциям рынка, развитию технологий или корпоративной стратегии. Кратко опишите другие технологии, процессы или ресурсы необходимые для удовлетворения клиента.

Бизнес-цели и критерии успеха

Суммирует важные преимущества бизнеса, предоставляемые продуктом, в количественном и измеряемом виде. Определите, как заинтересованные лица будут определять и измерять успех проекта. Установите факторы, которые максимально влияют на успех проекта. Из них выделите те, которые организация может контролировать, и те, которые находятся вне сферы ее влияния. Определите меру для оценки того, были ли достигнуты бизнес-цели.

Примеры финансовых и нефинансовых бизнес-целей:

Финансовые бизнес-цели

- Освоить $X\%$ рынка за Y месяцев
- Увеличить сектор рынка в стране X на $Y\%$ за Z месяцев
- Достигнуть объема продаж X единиц или дохода, равного $\$Y$, за Z месяцев
- Получить $X\%$ прибыли или дохода по инвестициям в течение Y месяцев
- Достигнуть положительного баланса по этому продукту в течение Y месяцев
- Сэкономить $\$X$ в год, которые в настоящий момент расходуются на обслуживание системы
- Уменьшить затраты на поддержку на $X\%$ за Z месяцев
- Получить не более X звонков в службу обслуживания по каждой единице товара и Y звонков по гарантии каждой единицы товара в течение Z месяцев после выпуска товара
- Увеличить валовую прибыль для существующего бизнеса с X до $Y\%$

Нефинансовые бизнес-цели

- Достигнуть показателя удовлетворения покупателей, равного, по крайней мере, X , в течение Y месяцев со времени выпуска товара
- Увеличить производительность обработки транзакций на $X\%$ и снизить уровень ошибок данных до величины не более $Y\%$
- Достигнуть определенного времени для достижения доминирующего положения на рынке
- Разработать надежную платформу для семи связанных продуктов
- Разработать специальную базовую технологическую основу для организации
- Получить X положительных отзывов в отраслевых журналах к определенной дате
- Добиться признания продукта лучшим по надежности в опубликованных обзорах продуктов к определенной дате

- Соответствовать определенным федеральным и государственным постановлениям
- Уменьшить время оборота до X часов на Y% звонков покупателей в службу поддержки

Потребности клиентов или рынка

Опишите потребности типичных покупателей или целевых сегментов рынка, включая потребности, которые не удовлетворяют настоящие продукты или информационные системы. Представьте проблемы, с которыми в настоящее время сталкиваются клиенты и которые решит новая система, и предоставьте примеры того, как покупатели будут использовать этот продукт. Определите на высоком уровне все известные важные требования к интерфейсу или производительности, но не касайтесь деталей дизайна или реализации.

Бизнес-риски

В этой части обобщите важнейшие бизнес-риски, связанные с разработкой — или не с разработкой — этого продукта. В категории рисков входят рыночная конкуренция, временные факторы, приемлемость для пользователей, проблемы, связанные с реализацией, и возможные негативные факторы, влияющие на бизнес. Оцените возможные потери от каждого фактора риска, вероятность его возникновения и вашу способность контролировать его. Определите все возможные действия по смягчению ситуации. Если вы уже подготовили эту информацию для анализа бизнес-задач или похожего документа, ссылайтесь на этот источник, а не копируйте эту информацию здесь.

Образ решения

В этом разделе документа определяется стратегический образ системы, позволяющей выполнять бизнес-задачи. Этот образ обеспечивает основу для принятия решений в течение жизненного цикла продукта. В него не надо включать детали функциональных требований или информацию, связанную с планированием проекта.

Положение об образе проекта

Составьте сжатое положение об образе проекта, обобщающее долгосрочные цели и назначение нового продукта. В этом документе следует отразить сбалансированный образ, удовлетворяющий различные заинтересованные лица. Он может быть несколько идеалистичным, но должен быть основан на существующих или предполагаемых рыночных факторах, архитектуре

предприятия, стратегическом направлении развития корпорации или ограничениях ресурсов. Ниже показан шаблон, состоящий из ключевых слов, который прекрасно подходят для документа об образе продукта:

для [целевая аудитория покупателей];

который [положение о потребностях или возможностях];

эта (этот) [имя продукта]

является [категория продукта];

который(ая) [ключевое преимущество, основная причина для покупки или использования];

в отличие от [основной конкурирующий продукт, текущая система или текущий бизнес-процесс];

наш продукт [положение об основном отличии и преимуществе нового продукта].

Вот как может выглядеть положение об образе (ключевые слова выделены):

Для сетевых инженеров, которым необходимо наблюдать и управлять состоянием сетевого оборудования, данная Network Visualization System является информационной системой, которая обеспечит единый интерфейс предоставления информации о состоянии сетевого оборудования.

В отличие от действующих сейчас систем наблюдения, которые предназначены для мониторинга только однотипного оборудования, наш продукт будет предоставлять информацию об оборудовании разных типов, работающих в единой сети. Кроме этого наш продукт будет соответствовать стандартам NGOSS, что обеспечит возможность оптимизации бизнес-процессов телеком-оператора.

Основные функции

Назовите или пронумеруйте каждую основную функцию нового продукта или возможность, предоставляемую пользователям, уникальным последовательным способом, подчеркивая те из них, которые отличают его от предыдущих или конкурирующих продуктов. Дав каждой функции уникальное имя, вы сможете отследить каждую функцию до отдельных требований пользователей, функциональных требований и других элементов систем.

Предположения и зависимости

Опишите все предположения, сделанные заинтересованными лицами, когда они обдумывали проект и создавали данный документ об образе и границах. Часто предположения одних лиц не разделяют другие стороны. Если вы запишите их и просмотрите позже, то получите возможность обговорить основные положения проекта. Так вы избежите путаницы и ухудшения ситуации в будущем. Также опишите важнейшие зависимости проекта от внешних факторов — изменения индустриальных стандартов или правительственные положения, других проектов, поставщиков со стороны или партнеров по разработке.

Масштабы и ограничения проекта

Вам необходимо указать, что может делать система, а что не может.

Границы проекта определяют концепцию и круг действия предложенного решения. В ограничениях указываются определенные возможности, которые не будут включены в продукт. Рамки и ограничения помогают установить реалистичные ожидания заинтересованных лиц.

Иногда клиенты запрашивают функции, слишком дорогостоящие или выходящие за предполагаемые границы продукта. Требования, выходящие за границы продукта, следует отклонять, если только они не настолько ценные, чтобы специально под них расширить проект, естественно, соответствующим образом изменив в бюджет, график и кадровый состав. Документируйте отклоненные требования и причины отказа от них, поскольку они имеют свойство появляться снова.

Объем первоначальной версии

Обобщает основные запланированные функции, включенные в первоначальную версию продукта. Опишите характеристики качества, которые позволяют продукту предоставлять предполагаемые выгоды различным классам пользователей. Если ваша задача — сосредоточиться на разработке и уложиться в график, вам следует избегать искушения включить в версию 1.0 каждую функцию, которая когда-нибудь в будущем может понадобиться какому-то потенциальному покупателю.

Увеличение сроков и сдвиг графика — типичный исход такого коварного расположения объема. Сосредоточьтесь на наиболее ценных функциях, имеющих максимально приемлемую стоимость, годных для самой широкой целевой аудитории, которые удастся создать как можно раньше.

Версия 1 не обязательно должна быть быстрой, красиво оформленной или легкой в использовании, но она должна быть надежной; это основа работы команды. Первая версия системы выполняет лишь базовые задачи. В будущие выпуски будут включены дополнительные функции, возможности и средства, обеспечивающие легкость и простоту использования,

Объем последующих версий

Если вы представляете поэтапную эволюцию продукта, укажите, какие функции будут отложены и желательные сроки последующих выпусков. В последующих версиях вы сможете реализовать дополнительные варианты использования и функции и расширить возможности первоначальных вариантов использования и функций. Вы также сможете повысить производительность, надежность и другие характеристики качества по мере совершенствования продукта. Чем дальше вы заглядываете, тем более расплывчатыми будут границы проекта. Вам наверняка придется передвинуть функциональность с одного запланированного выпуска до другого и, возможно, добавлять незапланированные функции. Короткие циклы выпуска часто удобны для сбора отзывов клиентов.

Бизнес-контекст

В этом разделе обобщаются некоторые бизнес-проблемы проекта, включая профили основных категорий заинтересованных лиц и приоритеты управления.

Профили заинтересованных лиц

Заинтересованными в проекте лицами (stakeholders) называются отдельные лица, группы или организации, которые активно вовлечены в проект, на которых влияет результат проекта и которые сами могут влиять на этот результат.

Профили заинтересованных лиц описывают различные категории клиентов и других ключевых лиц, заинтересованных в этом проекте. Вам не нужно описывать каждую группу заинтересованных лиц, например юристов, которые проверяют соответствие надлежащим законам. Сферой вашего интереса должны стать различные группы клиентов, целевые рыночные сегменты и различные классы пользователей, входящих в эти сегменты. В профиль каждого заинтересованного в проекте лица включается следующая информация:

- основная ценность или преимущество, которое продукт принесет заинтересованным лицам и то, как продукт удовлетворит покупателей. Ценность для заинтересованных лиц представляют:
 - улучшенная производительность;

- *меньшее количество переделок;*
- *снижение себестоимости;*
- *ускорение бизнес-процессов;*
- *автоматизация задач, ранее выполнявшихся вручную;*
- *возможность выполнять совершенно новые задачи;*
- *соответствие соответствующим стандартам и правилам;*
- *лучшая, по сравнению с текущими продуктами, легкость и простота использования;*
- *их вероятное отношение к продукту;*
- *наиболее интересные функции и характеристики;*
- *все известные ограничения, которые должны быть соблюдены.*

Приоритеты проекта

Чтобы принимать эффективные решения, заинтересованные лица должны договориться о приоритетах проекта. Один из подходов к этому заключается в рассмотрении пяти измеряемых параметров проекта: функции (или объем), качество, график, затраты и кадры. В любом проекте каждый из этих параметров относится к одной из трех категорий:

- *ограничение — лимитирующий фактор, в рамках которого должен оперировать менеджер проекта;*
- *ключевой фактор — важный фактор успеха, ограниченно гибкий при изменениях;*
- *степень свободы — фактор, который менеджер проекта может до определенной степени изменять и балансировать относительно других параметров.*

Приоритеты проекта могут составляться аналитиком совместно с менеджером проекта. Эти параметры помогут менеджеру проекта в управлении в нестандартных ситуациях.

Задача менеджера проекта и аналитика — настроить те факторы, которые представляют собой степени свободы для достижения ключевых факторов

успеха проекта в рамках, налагаемых ограничениями. Не все факторы могут быть ключевыми, как и не все — ограничениями. Менеджеру проекта необходима определенная степень свободы для того, чтобы он мог реагировать должным образом на изменение требований к проекту или внешних обстоятельств. Представьте себе, что отдел маркетинга неожиданно требует создать продукт на месяц раньше срока.

- *Какова будет ваша реакция?*
- *Вы отложите реализацию определенных требований до более поздней версии?*
- *Сократите запланированный цикл тестирования системы?*
- *Оплатите сверхурочную работу вашим специалистам или пригласите специалистов по контракту для ускорения разработки?*
- *Привлечете ресурсы других проектов для разрешения ситуации?*

Именно от приоритетов проекта зависят ваши действия в подобных ситуациях.

Операционная среда

Опишите среду, в которой будет использоваться система, и определите важнейшие требования к доступности, надежности, производительности и целостности. Эта информация существенно влияет на определение архитектуры системы, что является первым — и часто самым важным — этапом дизайна. Архитектура системы, предназначенной для поддержки пользователей, которые находятся далеко друг от друга и которым необходим круглосуточный доступ, сильно отличается от той, что предназначена для доступа пользователей, находящихся рядом, только в рабочие часы. На нефункциональные требования, такие как отказоустойчивость и способность обслуживать систему во время ее работы, требуется значительное количество средств, отпущеных на дизайн и реализацию. Чтобы прояснить ситуацию, задайте заинтересованным лицам уточняющие вопросы. Например:

- *Пользователи расположены далеко (географически) или близко друг от друга? В скольких часовых поясах работают ваши пользователи?*
- *Когда пользователям, находящимся в различных географических местоположениях, требуется доступ к системе?*

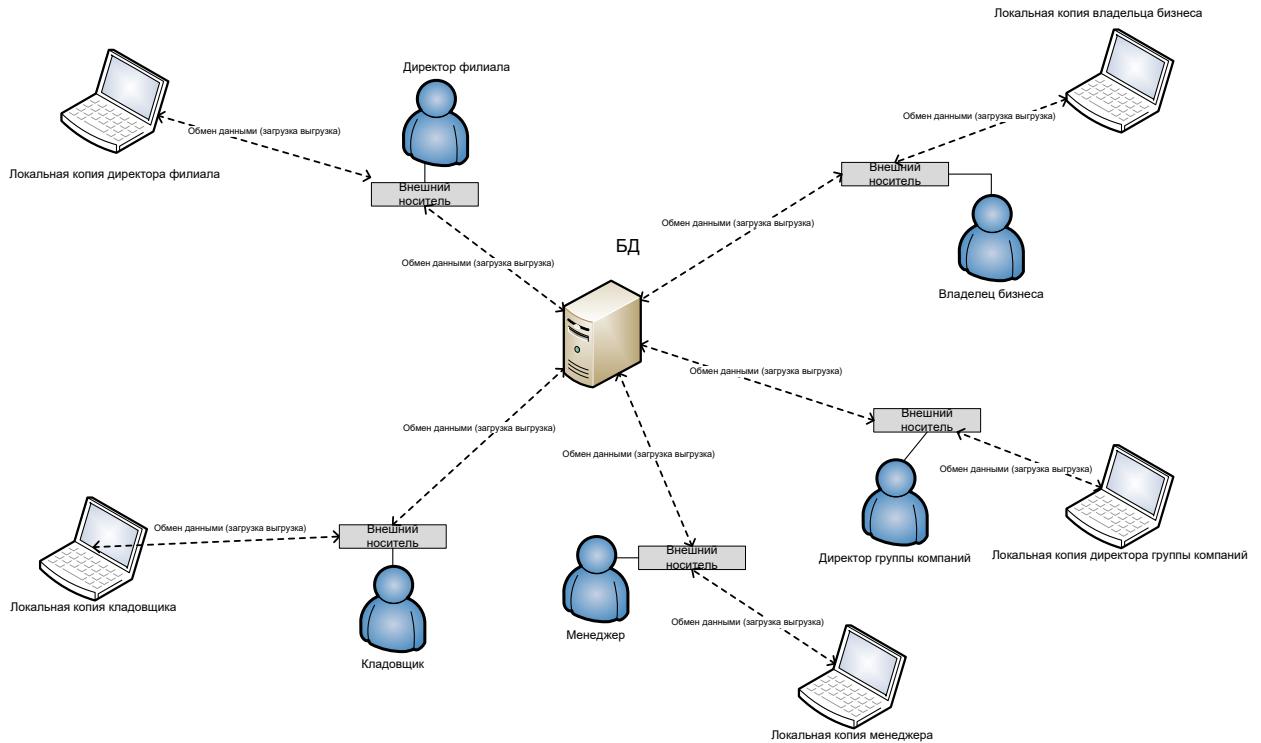
- Где данные генерируются и используются? Насколько далеко друг от друга расположены эти местоположения? Нужно ли объединять данные из разных местоположений?
- Известно ли максимальное время отклика для получения доступа к данным, которые могут храниться удаленно?
- Готовы ли пользователи смириться с прерыванием работы службы или непрерывный доступ к системе крайне важен для работы их компании?
- Какие элементы управления безопасностью и требования к защите данных необходимы?

Контекстная диаграмма

Уточнение рамок определяет границу и связи системы, которую мы разрабатываем, со всем остальным миром. Контекстная диаграмма (context diagram) графически иллюстрирует эту границу. Она определяет оконечные элементы (terminators), расположенные вне системы, которые определенным образом взаимодействуют с ней, а также данные, элементы управления и материальные потоки, протекающие между оконечными элементами и системой.

Контекстная диаграмма может быть представлена в достаточно произвольном виде. Важно, чтобы она была понятна как заказчикам, так и разработчикам.

Пример (для одного небольшого проекта):



Контрольные вопросы:

- 1) Что описывает диаграмма вариантов использования?
- 2) Какие элементы присутствуют на диаграмме вариантов использования?
- 3) Может ли быть несколько диаграмм вариантов использования?
- 4) Что такое функциональная спецификация?

Лабораторная работа №16

Разработка приложения ASP.Net Core MVC

Цель работы : изучить работу фреймворка ASP.NET Core MVC , создание компонент модели контроллера и представления для приложения с базой данных. Дополнительно ознакомиться с концепцией развёртывания приложения в Docker.

1. Теоретические сведения.

Фреймворк ASP.NET Core MVC

Фреймворк ASP.NET Core MVC является частью платформы ASP.NET Core, его отличительная особенность - применение паттерна MVC. Преимуществом использования фреймворка ASP.NET Core MVC по сравнению с "чистым" ASP.NET Core является то, что он упрощает в ряде ситуаций и сценариев организацию и создание приложений, особенно это относится к большим приложениям.

Стоит отметить, сам паттерн MVC не является исключительной особенностью ASP.NET Core MVC, данный паттерн появился еще в конце 1970-х годов в компании Xerox как способ организации компонентов в графическом приложение на языке Smalltalk и в настоящее время применяется во многих платформах и на различных языках программирования. Особенno популярен паттерн MVC в веб-приложениях.

Концепция паттерна MVC предполагает разделение приложения на три компонента:

- **Модель (model):** описывает используемые в приложении данные, а также логику, которая связана непосредственно с данными, например, логику валидации данных. Как правило, объекты моделей хранятся в базе данных.

В MVC модели представлены двумя основными типами: модели представлений, которые используются представлениями для отображения и передачи данных, и модели домена, которые описывают логику управления данными.

Модель может содержать данные, хранить логику управления этими данными. В то же время модель не должна содержать логику взаимодействия с пользователем и не должна определять механизм обработки запроса. Кроме того, модель не должна содержать логику отображения данных в представлении.

- **Представление (view):** отвечают за визуальную часть или пользовательский интерфейс, нередко html-страница, через который пользователь взаимодействует с приложением. Также представление может содержать логику, связанную с отображением данных. В то же время представление не должно содержать логику обработки запроса пользователя или управления данными.
- **Контроллер (controller):** представляет центральный компонент MVC, который обеспечивает связь между пользователем и приложением, представлением и хранилищем данных. Он содержит логику обработки запроса пользователя. Контроллер получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки

отправляет пользователю определенный вывод, например, в виде представления, наполненного данными моделей.

Отношения между компонентами паттерна можно описать следующей схемой:



Рисунок 82 - Отношения между компонентами паттерна MVC

В этой схеме модель является независимым компонентом - любые изменения контроллера или представления никак не влияют на модель. Контроллер и представление являются относительно независимыми компонентами. Так, из представления можно обращаться к определенному контроллеру, а из контроллера генерировать представления, но при этом нередко их можно изменять независимо друг от друга.

Такое разграничение компонентов приложения позволяет реализовать концепцию разделение ответственности, при которой каждый компонент отвечает за свою строго очерченную сферу. В связи с чем легче построить работу над отдельными компонентами. И благодаря этому приложение легче разрабатывать, поддерживать и тестировать отдельные компоненты. Допустим, если нам важна визуальная часть или фронтэнд, то мы можем тестировать представление независимо от контроллера. Либо мы можем сосредоточиться на бэкэнде и тестировать контроллер.

Docker и docker-compose

Докер — это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрого развертывания ваших приложений. С помощью docker вы можете отделить ваше приложение от вашей инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Docker помогает быстрее развертывать приложения и уменьшить время между написанием кода и запуска кода. Docker делает это с помощью легковесной платформы контейнерной виртуализации, используя процессы и утилиты, которые помогают управлять и выкладывать ваши приложения.

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно.

2. Выполнение работы.

1. Выполнить пример 1.

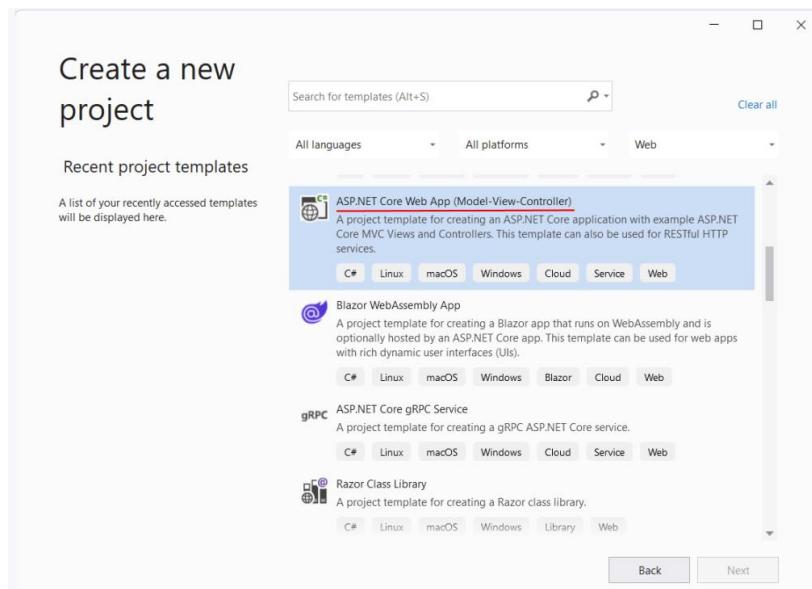
Прежде чем приступить к выполнению примеров, убедитесь, что ваша IDE использует .NET 6.0.

Пример 1

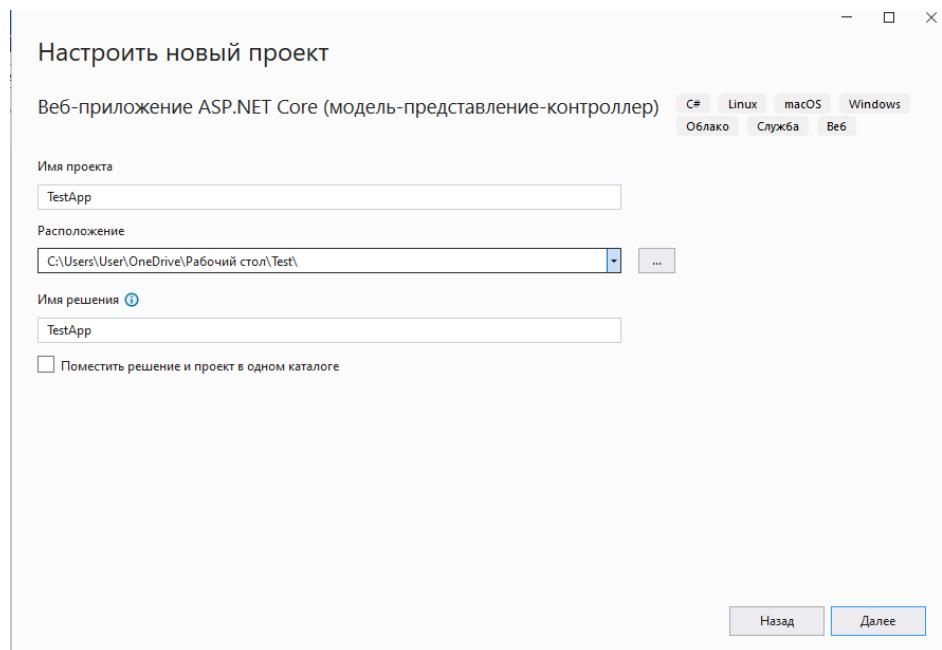
Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных о книгах, купленных библиотекой: название, автор, год издания, адрес автора, адрес издательства, цена, книготорговая фирма.

Создание проекта ASP.NET Core MVC

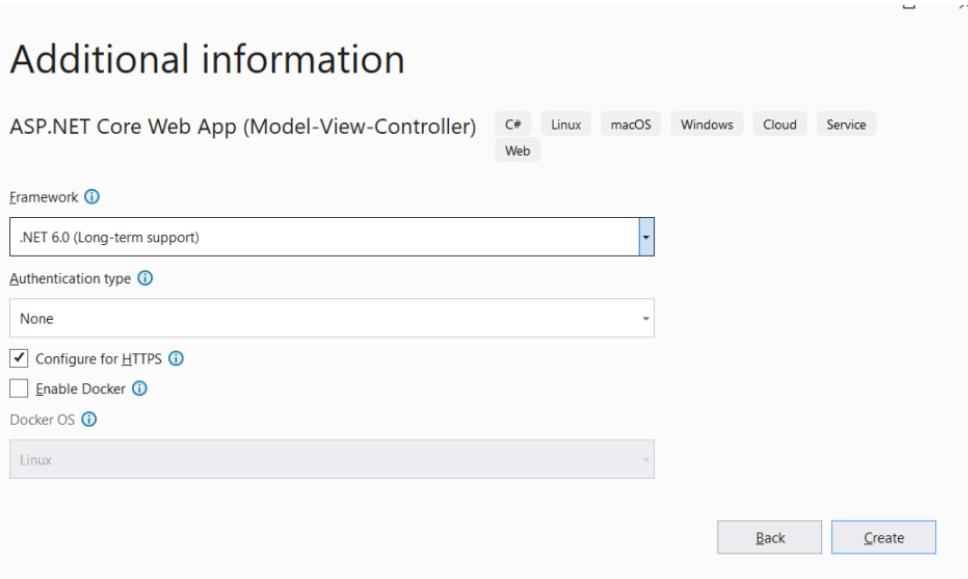
Для создания проекта на ASP.NET Core MVC мы можем выбрать любой тип проекта на ASP.NET Core и в нем уже добавлять необходимые компоненты. Однако для упрощения Visual Studio уже по умолчанию предоставляет для этого шаблон ASP.NET Core Web App (Model-View-Controller):



Выберем данный шаблон для создания проекта. Дальше нам откроется окно для установки имени проекта. Допустим, проект будет называться HelloMvcApp:

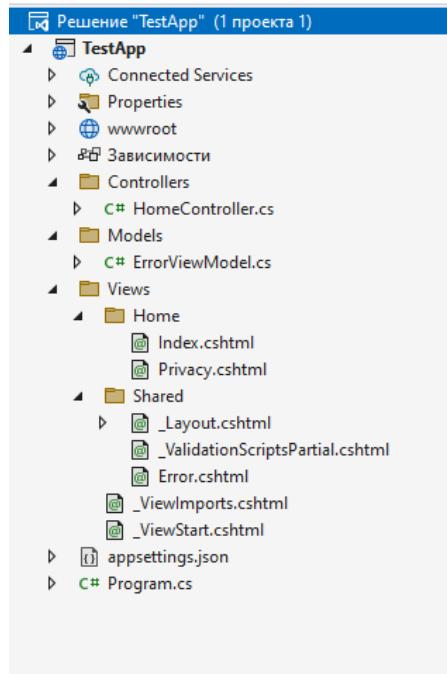


Далее нам надо будет настроить стандартные настройки для ASP.NET Core:



Оставим все настройки по умолчанию и нажмем на OK. И Visual Studio создаст новый проект MVC.

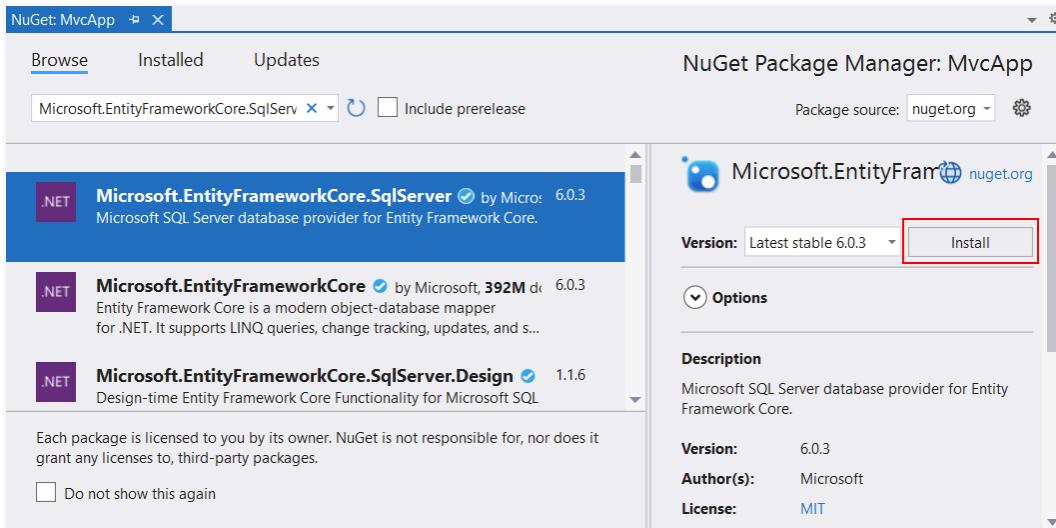
Структура создаваемого проекта будет отличаться от структуры проекта по типу Empty. В частности, мы увидим ряд новых папок и файлов:



Работа с данными в Entity Framework

Данные моделей, как правило, хранятся в базе данных. Для работы с базой данных очень удобно пользоваться фреймворком Entity Framework, который позволяет абстрагироваться от написания sql-запросов, от строения базы данных и полностью сосредоточиться на логике приложения.

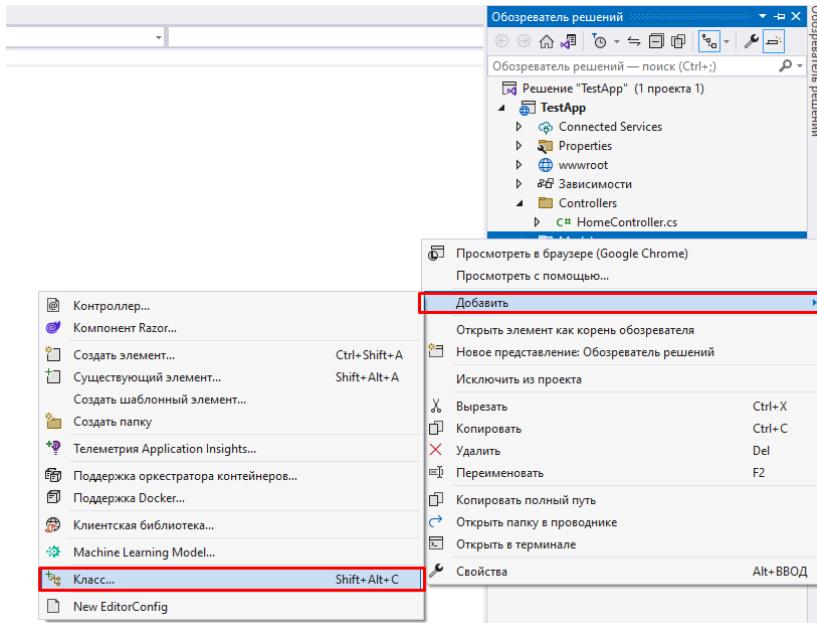
Entity Framework представляет прекрасное ORM-решение, которое позволяет автоматически связать обычные классы языка C# с таблицами в базе данных. Entity Framework Core поддерживает различные СУБД, но в данном случае мы будем работать с базами данных MS SQL Server.



Помимо этого, установите **Microsoft.EntityFrameworkCore.Design**, который позволяет IDE успешно реализовать интерфейсы поставщиков баз данных.

Entity Framework поддерживает подход "Code first", который предполагает сохранение или извлечение информации из БД на SQL Server без создания схемы базы данных или использования дизайнера в Visual Studio. Наоборот, мы создаем обычные классы, а Entity Framework уже сам определяет, как и где сохранять объекты этих классов.

Далее в папке Models (ПКМ по папке => Добавить => Класс) создадим класс необходимой сущности:



```

namespace TestApp.Models
{
    public class Book
    {
        public string Title { get; set; } = null!;
        public string Author { get; set; } = null!;
        public int YearOfPublishing { get; set; }
        public string AuthorsAddress { get; set; } = null!;
        public string PublishersAddress { get; set; } = null!;
        public decimal Price { get; set; }
        public string BooksellingCompany { get; set; } = null!;
    }
}

```

Эта модель представляет те объекты, которые будут храниться в базе данных.

Чтобы взаимодействовать с базой данных через Entity Framework нам нужен контекст данных - класс, унаследованный от класса Microsoft.EntityFrameworkCore.DbContext. Создадим класс контекста данных(BookDbContext)

```

using Microsoft.EntityFrameworkCore;
using TestApp.Models;

namespace TestApp.DbContexts
{
    public class BookDbContext : DbContext
    {
        public DbSet<Book> Books { get; set; } = null!;
        public BookDbContext(DbContextOptions<BookDbContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }
    }
}

```

Через параметр options в конструктор контекста данных будут передаваться настройки контекста.

В конструкторе с помощью вызова `Database.EnsureCreated()` по определению моделей будет создаваться база данных (если она отсутствует).

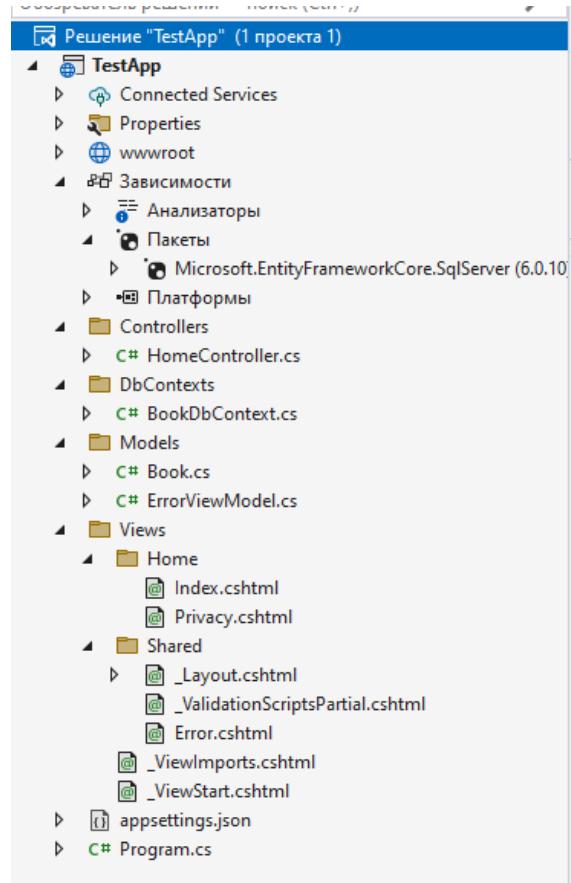
Чтобы подключаться к базе данных, нам надо задать параметры подключения. Для этого изменим файл `appsettings.json`, добавив в него определение строки подключения:

```
},
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=booksdb;Trusted_Connection=True;"
},
```

И последним шагом в настройке проекта является изменение файла `Program.cs` а именно регистрация контекста:

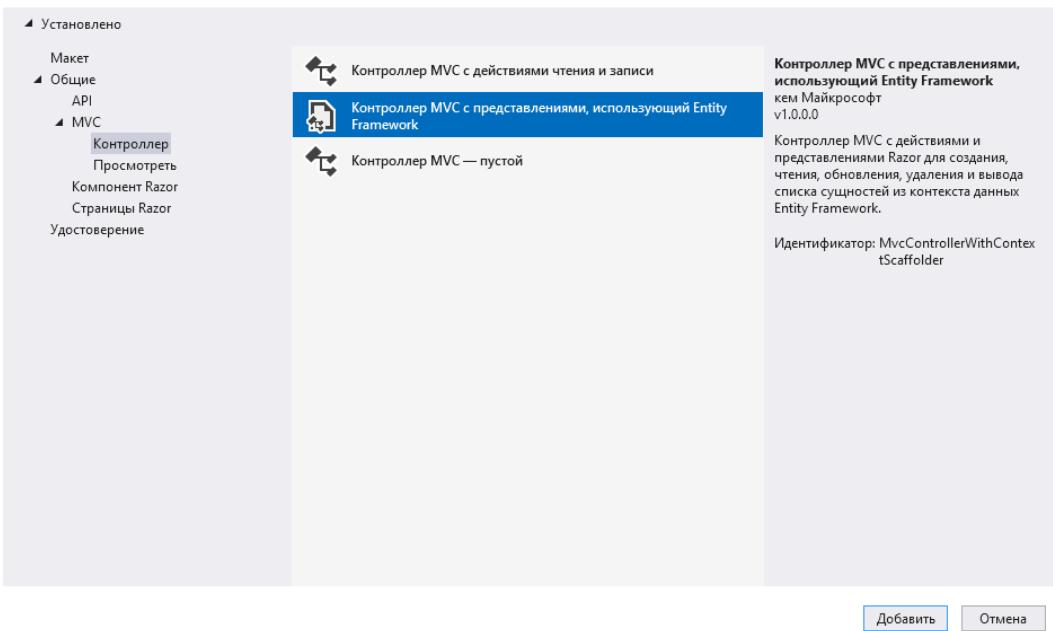
```
builder.Services.AddDbContext<BookDbContext>(options =>
{
  options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));
});
```

Структура проекта на данный момент:

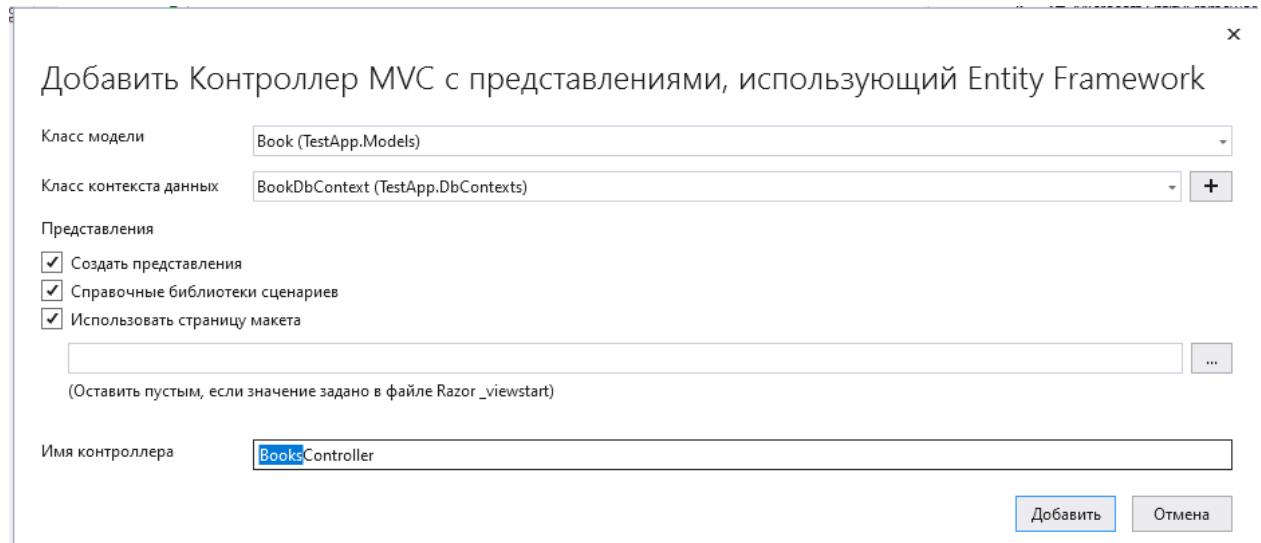


Теперь можно приступить к генерации контроллеров и представлений для выполнения CRUD-операций. Для этого нажимаем ПКМ по папке Controllers, выбираем Добавить => Контроллер. После чего открывается следующее окно:

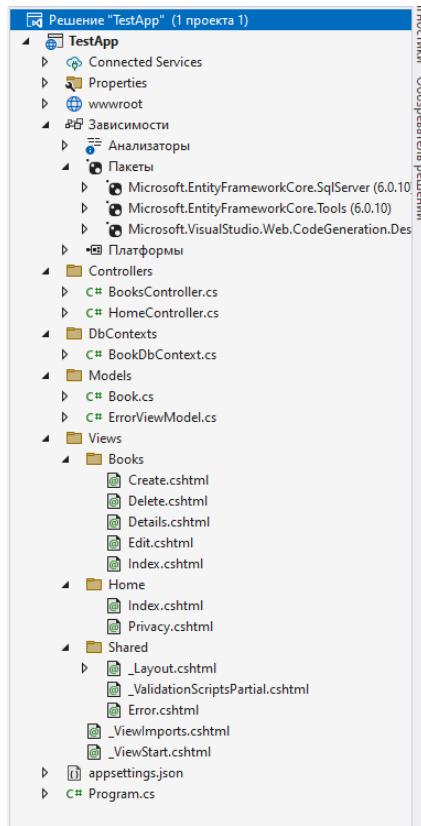
Добавить новый шаблонный элемент



Выбираем «Контроллер MVC с представлениями, использующий Entity Framework» и нажимаем «Добавить». В появившемся окне выбираем класс модели (соответствующий определенной таблице в БД), для которой мы хотим создать контроллер CRUD-операций, выбираем файл контекста, проверяем, чтобы все галочки были установлены.



После нажатия на кнопку Добавить автоматически сгенерируется контроллер и все базовые представления, структура проекта в конечном итоге:



Теперь осталось только добавить либо кнопку для получения представления Book.Index.cshtml, либо простую переадресацию в методе HomeController:

```
Ссылок: 1
public class HomeController : Controller
{
    Ссылок: 0
    public HomeController(BookDbContext context)
    {
    }

    Ссылок: 0
    public IActionResult Index()
    {
        return Redirect("~/Books/Index");
    }
}
```

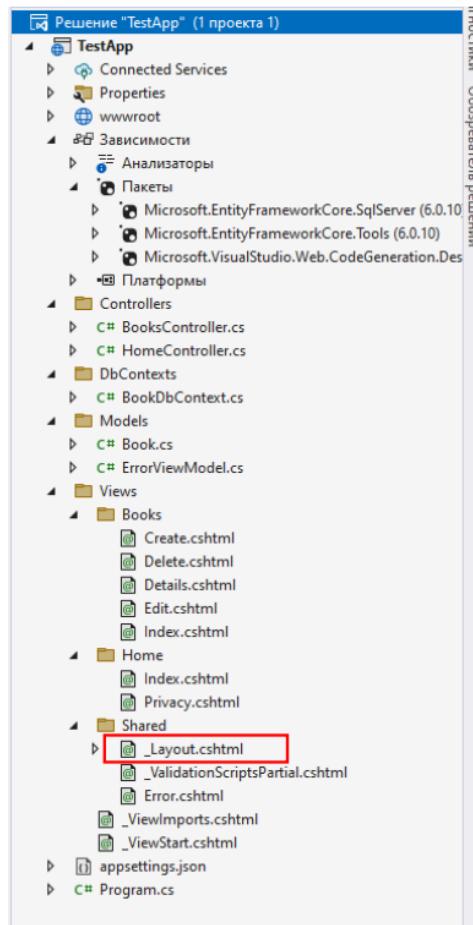
Теперь можно запустить и проверить работу приложения:

Index

[Create New](#)

Title	Author	YearOfPublising	AuthorsAddress	PublishersAddress	Price	BooksellingCompany	
sfdgfg	asdsf	2023	asdfg	asdf	22.00	sdf	Edit Details Delete

Для добавления ссылок в панель навигации, необходимо немного изменить файл _Layout.cshtml который можно найти по следующему пути
“Views/Shared/_Layout.cshtml”



После открытия файла, ищем в разметке следующий элемент неупорядоченный список.

```

<header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">ASP</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                <ul class="navbar-nav flex-grow-1">
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

```

В нем можно скопировать любой элемент списка и вставить в конец списка, например, первый.

```

<ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
    </li>
</ul>

```

Теперь необходимо исправить атрибуты и содержимое элемента на необходимые. В атрибуте `asp-controller` необходимо указать контроллер, действия которого необходимо вызывать. Далее в атрибуте `asp-action` необходимо указать название действие которые должно вызываться.

Просмотрим раннее созданный контроллер `BooksController`, в нем есть метод `Create`, который нам нужно вызвать.

```

public class BooksController : Controller
{
    private readonly BookDbContext _context;

    public BooksController(BookDbContext context)
    {
        _context = context;
    }

    // GET: Books/Create
    public IActionResult Create()
    {
        return View();
    }
}

```

Заменим атрибуты скопированного элемента списка

```
<ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Books" asp-action="Create">Create</a>
    </li>
```

Запустим приложение

Как видно в навигационную панель добавился новый пункт для создания нового объекта база данных. Кликнем для проверки.

Create

Book

Title

Author

YearOfPublishing

AuthorsAddress

PublishersAddress

[Create](#)

[Back to List](#)

Открылось представление для добавления.

Часть 2. Разработка ASP.NET Core MVC приложения с использованием Docker

Архитектура Docker

Docker использует архитектуру клиент-сервер. Docker клиент общается с демоном Docker, который берет на себя тяжесть создания, запуска, распределения ваших контейнеров. Оба, клиент и сервер могут работать на одной системе, вы можете подключить клиент к удаленному демону docker. Клиент и сервер общаются через сокет или через RESTful API.

Docker-демон

Демон запускается на хост-машине. Пользователь не взаимодействует с сервером на прямую, а использует для этого клиент.

Docker-клиент

Docker-клиент, программа docker — главный интерфейс к Docker. Она получает команды от пользователя и взаимодействует с docker-демоном.

Внутри docker-a

Чтобы понимать, из чего состоит docker, нужно знать о трех компонентах:

- образы (images)
- реестр (registries)
- контейнеры

Образы

Docker-образ — это read-only шаблон. Например, образ может содержать операционку Ubuntu с Apache и приложением на ней. Образы используются для создания контейнеров. Docker позволяет легко создавать новые образы, обновлять существующие, или скачивать образы, созданные другими людьми. Образы — это компонента сборки docker-a.

Реестр

Docker-реестр хранит образы. Есть публичные и приватные реестры, из которых можно скачать либо загрузить образы. Публичный Docker-реестр — это Docker Hub. Там хранится огромная коллекция образов. Реестры — это компонента распространения.

Контейнеры

Контейнеры похожи на директории. В контейнерах содержится все, что нужно для работы приложения. Каждый контейнер создается из образа. Контейнеры могут быть созданы, запущены, остановлены, перенесены или удалены. Каждый контейнер изолирован и является безопасной платформой для приложения. Контейнеры — это компонента работы.

Скачать Docker: <https://docs.docker.com/desktop/install/windows-install/>

Docker-compose

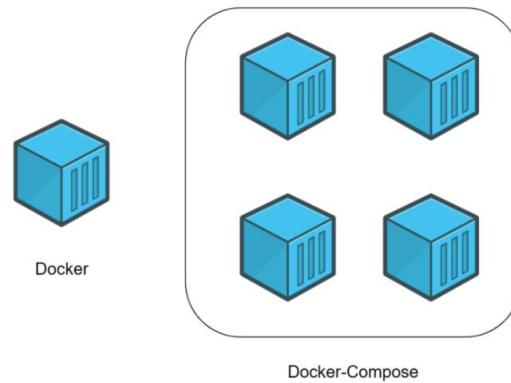
Docker Compose — это инструментальное средство, входящее в состав Docker. Оно предназначено для решения задач, связанных с развёртыванием проектов.

Технология Docker Compose, если описывать её упрощённо, позволяет, с помощью одной команды, запускать множество сервисов.

Разница между Docker и Docker Compose

Docker применяется для управления отдельными контейнерами (сервисами), из которых состоит приложение.

Docker Compose используется для одновременного управления несколькими контейнерами, входящими в состав приложения. Этот инструмент предлагает те же возможности, что и Docker, но позволяет работать с более сложными приложениями.



Вся работа с docker-compose реализована через файл docker-compose.yml. Пример для ознакомления:

```

# Файл docker-compose должен начинаться с тега версии.
# Мы используем "3" так как это - самая свежая версия на момент написания этого
кода.

version: "3"

# Следует учитывать, что docker-composes работает с сервисами.
# 1 сервис = 1 контейнер.
# Сервисом может быть клиент, сервер, сервер баз данных...
# Раздел, в котором будут описаны сервисы, начинается с 'services'.

services:

# Первый сервис (контейнер): сервер.
# Назвать его можно так, как нужно разработчику.
# Понятное название сервиса помогает определить его роль.
# Здесь мы, для именования соответствующего сервиса, используем ключевое
# слово 'server'.

server:

# Ключевое слово "build" позволяет задать
# путь к файлу Dockerfile, который нужно использовать для создания образа,
# который позволит запустить сервис.
# Здесь 'server/' соответствует пути к папке сервера,
# которая содержит соответствующий Dockerfile.

build: server/

# Команда, которую нужно запустить после создания образа.
# Следующая команда означает запуск "python ./server.py".

command: python ./server.py

# Вспомните о том, что в качестве порта в 'server/server.py' указан порт 1234.
# Если мы хотим обратиться к серверу с нашего компьютера (находясь за
пределами контейнера),
# мы должны организовать перенаправление этого порта на порт компьютера.
# Сделать это нам поможет ключевое слово 'ports'.
# При его использовании применяется следующая конструкция: [порт
компьютера]:[порт контейнера]
# В нашем случае нужно использовать порт компьютера 1234 и организовать его
связь с портом
# 1234 контейнера (так как именно на этот порт сервер

```

```
# ожидает поступления запросов).

ports:
- 1234:1234

# Второй сервис (контейнер): клиент.
# Этот сервис назван 'client'.

client:
# Здесь 'client/' соответствует пути к папке, которая содержит
# файл Dockerfile для клиентской части системы.

build: client/

# Команда, которую нужно запустить после создания образа.
# Следующая команда означает запуск "python ./client.py".

command: python ./client.py

# Ключевое слово 'network_mode' используется для описания типа сети.
# Тут мы указываем то, что контейнер может обращаться к 'localhost'
компьютера.

network_mode: host

# Ключевое слово 'depends_on' позволяет указывать, должен ли сервис,
# прежде чем запуститься, ждать, когда будут готовы к работе другие сервисы.
# Нам нужно, чтобы сервис 'client' дождался бы готовности к работе сервиса
'server'.

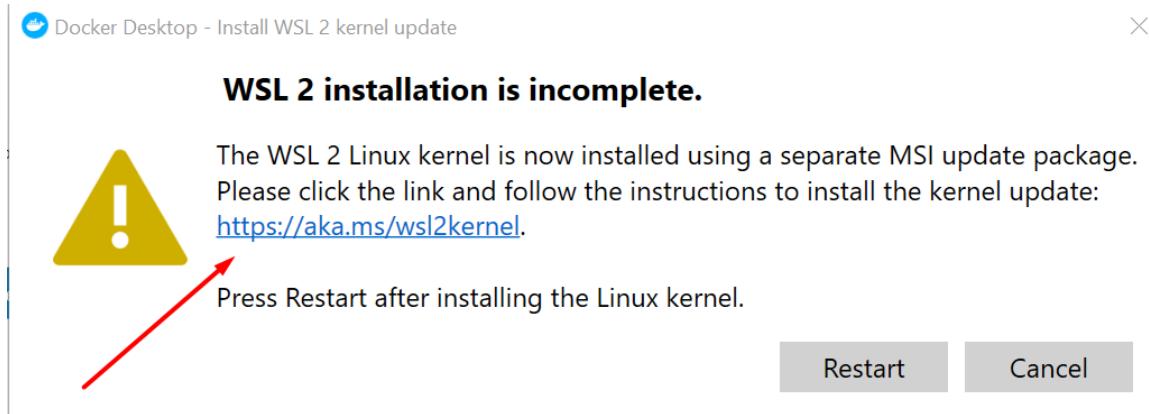
depends_on:
- server
```

Пример выполнения дополнительного задания(продолжение Примера 1)

Примечание: для дальнейшей работы необходима установка Docker-desktop.

Скачать Docker: <https://docs.docker.com/desktop/install/windows-install/>

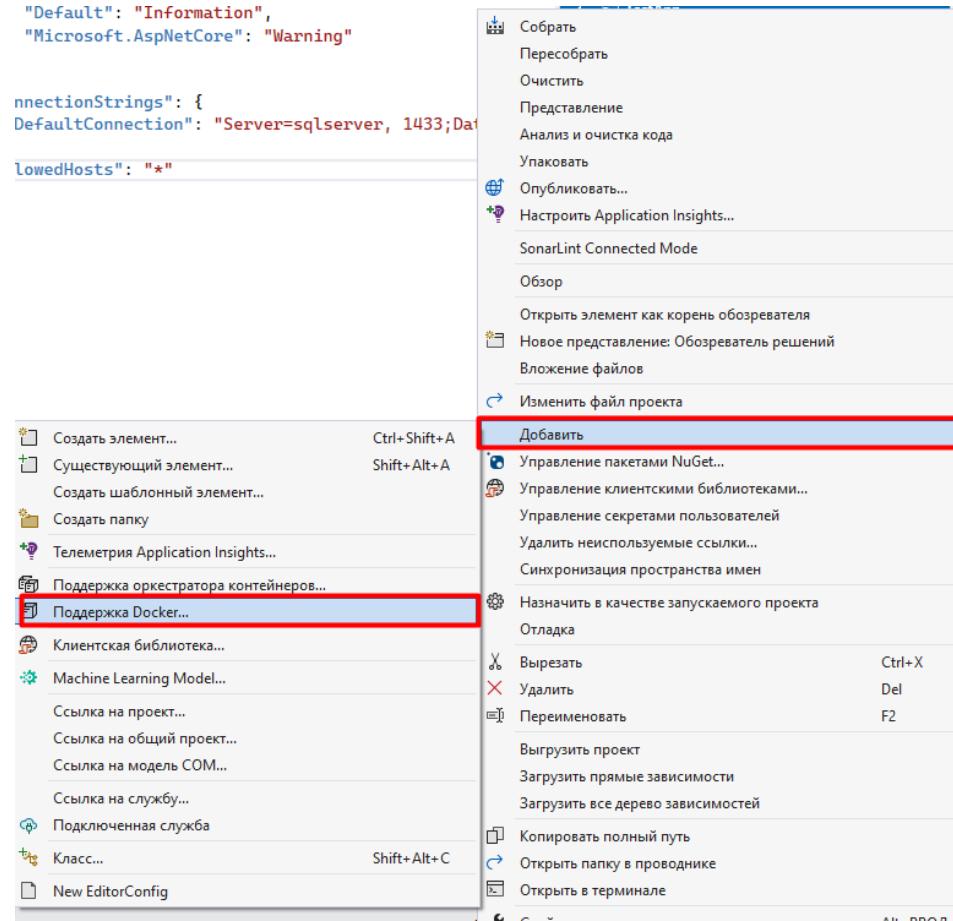
Для работы с Linux контейнерами необходима установка WSL (Windows Subsystem for Linux). После установки Docker Desktop, если не установлена WSL появится следующие окно:



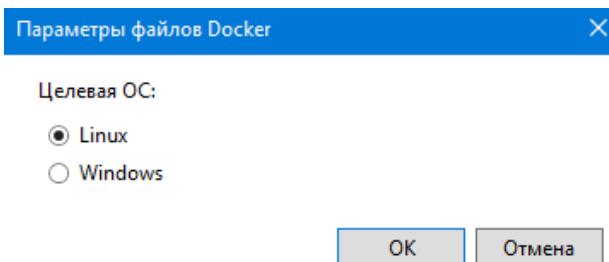
Необходимо перейти по указанной ссылке для установки WSL после чего перезапустить Docker.

Теперь развернем приложение и базу данных в контейнерах Docker. Для этого воспользуемся технологией docker-compose:

Сначала сгенерируем dockerfile для нашего приложения, для этого кликаем ПКМ по имени проекта в обозревателе решений => Добавить => Поддержка Docker:



Далее в открывшемся окне выбираем Linux:



После этого у нас автоматически сгенерировался следующий dockerfile:

```

Microsoft.Vis...compose.targets      Program.cs      appsettings.json      BookDbContext.cs      Dockerfile      Book.c
1  #See https://aka.ms/containerfastmode to understand how Visual Studio uses this
2
3  FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
4  WORKDIR /app
5  EXPOSE 80
6  EXPOSE 443
7
8  FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
9  WORKDIR /src
10 COPY ["TestApp/TestApp.csproj", "TestApp/"]
11 RUN dotnet restore "TestApp/TestApp.csproj"
12 COPY ..
13 WORKDIR "/src/TestApp"
14 RUN dotnet build "TestApp.csproj" -c Release -o /app/build
15
16 FROM build AS publish
17 RUN dotnet publish "TestApp.csproj" -c Release -o /app/publish /p:UseAppHost=fal
18
19 FROM base AS final
20 WORKDIR /app
21 COPY --from=publish /app/publish .
22 ENTRYPOINT ["dotnet", "TestApp.dll"]

```

В файлах Dockerfile содержатся инструкции по созданию образа. С них, набранных заглавными буквами, начинаются строки этого файла.

FROM — задаёт базовый (родительский) образ.

WORKDIR — задаёт рабочую директорию для следующей инструкции.

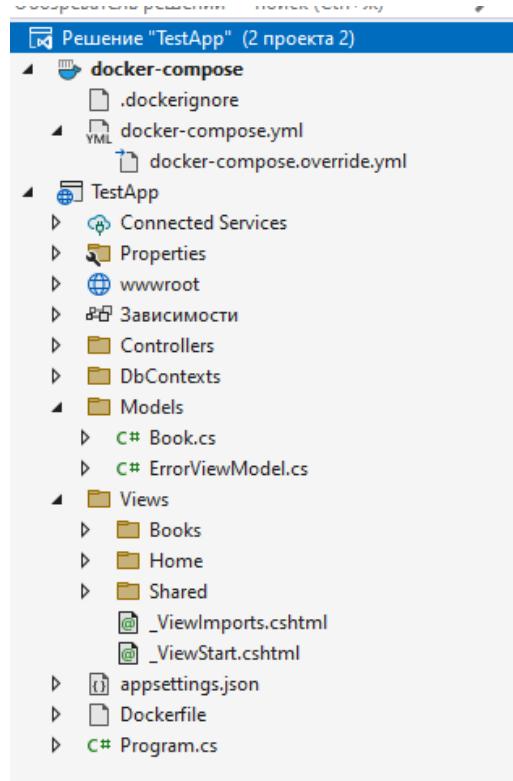
EXPOSE — указывает на необходимость открыть порт.

COPY — копирует в контейнер файлы и папки.

RUN — выполняет команду и создаёт слой образа. Используется для установки в контейнер пакетов.

ENTRYPOINT — предоставляет команду с аргументами для вызова во время выполнения контейнера. Аргументы не переопределяются.

Теперь создадим docker-compose, для этого кликаем ПКМ по имени проекта в обозревателе решений => Добавить => Поддержка оркестратора контейнеров и также выбираем Linux. При удачном выполнении структура решения выглядит след. образом:



Как видно из скрина, у нас есть два файла docker-compose. Docker Compose по умолчанию читает два файла: docker-compose.yml и docker-compose.override.yml. В файле docker-compose-override.yml можно хранить переопределения для существующих сервисов или определять новые.

Но на данный момент в docker-compose существует только контейнер с нашим приложением, добавим в него контейнер с MS SQL:

docker-compose.yml:

```
sqlserver:
  container_name: sqlserver
  image: "mcr.microsoft.com/mssql/server:2019-latest"
  hostname: sqlserver
```

Файл должен содержать следующие строки:

```

version: '3.4'

services:
  testapp:
    image: ${DOCKER_REGISTRY}-testapp
    build:
      context: .
      dockerfile: TestApp/Dockerfile
  sqlserver:
    container_name: sqlserver
    image: "mcr.microsoft.com/mssql/server:2019-latest"
    hostname: sqlserver

```

docker-compose.override.yml:

```

sqlserver:
  ports:
    - 1433:1433
  environment:
    - ACCEPT_EULA=Y
    - SA_PASSWORD=1234@qwerty

```

Файл должен содержать следующие строки:

```

version: '3.4'

services:
  testapp:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
    ports:
      - "80"
      - "443"
    volumes:
      - ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
  sqlserver:
    ports:
      - 1433:1433
    environment:
      - ACCEPT_EULA=Y
      - SA_PASSWORD=1234@qwerty

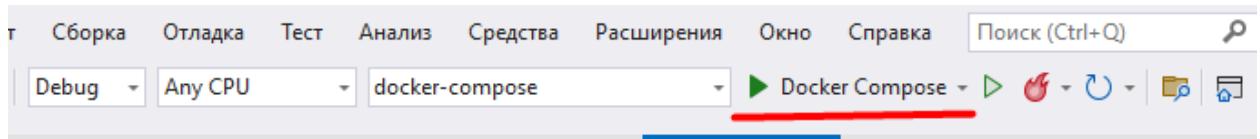
```

Ну и последний шаг - изменение строки подключения базы данных. Так как и приложение и база данных теперь находятся в докере адрес базы данных нужно изменить:

```
'  
"ConnectionStrings": {  
    "DefaultConnection": "Server=sqlserver, 1433;Database=booksDb;User Id=SA;Password=1234@qwerty;Encrypt=false;"  
},
```

Прошу заметить, что имя сервера и пароль должны в точности соответствовать тому, что прописано в docker-compose.

Теперь мы можем запустить весь docker-compose прямо из Visual Studio:



И в Docker-desktop отобразятся контейнеры:

	NAME	IMAGE	STATUS	PORT(S)	STARTED	
□	dockercompose96305603 ²	-	Running (2/2)	-	3 minutes ago	Open ▶ II ⏪ ⏴ ⏵
□	sqlserver eedd39684bcc	mcr.microsoft.com/mssql/	Running	1433	3 minutes ago	Open ▶ II ⏪ ⏴ ⏵
□	TestApp c263736699b1	testapp	Running	55092...	3 minutes ago	Open ▶ II ⏪ ⏴ ⏵

2. Разработать приложение ASP.NET 6 Core MVC согласно варианту (без использования Docker).

Дополнительное задание: развернуть приложение в докере согласно Примеру 1.

Вариант 1

Разработать MVC-приложение со всеми функциями (CRUD-create, read, update, delete) для работы с базой данных о студентах, для их распределения по местам практики: фамилия, год рождения, пол, группа, факультет, средний балл, место работы, город.

Вариант 2

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных об автомобилях: номер, год выпуска, марка, цвет, состояние, фамилия владельца, адрес.

Вариант 3

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных о квартирах, предназначенных для продажи: район, этаж, площадь, количество комнат, сведения о владельце, цена.

Вариант 4

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных о сотрудниках, имеющих компьютер: фамилия, номер комнаты, название отдела, данные о компьютерах, дата приобретения ПК.

Вариант 5

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных о заказах, полученных сотрудниками фирмы: фамилия, сумма заказа, наименование товара, название фирмы - клиента, фамилия заказчика.

Вариант 6

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных об оценках, полученных студентами на экзаменах: фамилия, группа, предмет, номер билета, оценка, преподаватель.

Вариант 7

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных об авторах web-сайта и их статьях: имя, адрес, учетная запись, пароль, тема, заголовок, дата публикации.

Вариант 8

Спроектировать структуру базы данных о списке рассылки и подписчиках: тема и содержание письма, дата отправки, имена и адреса подписчиков, их учетные записи и пароли.

Вариант 9

В базе данных содержится информация о туристических поездках: страна, город, изображение городской достопримечательности, количество дней, дата поездки, класс отеля, цена.

Вариант 10

Разработать MVC-приложение со всеми функциями (CRUD) для работы с базой данных о компьютерах, предназначенных для продажи: производитель, модель, цена, гарантийный срок, модель процессора, модель видеокарты.

Контрольные вопросы:

- 1)Как создать приложение Asp Net Core MVC?
- 2)Что позволяет Asp Net Core MVC?
- 3)Какой фреймворк применяется для работой с базой данных в Asp Net Core MVC?
- 4)Опишите компоненты MVC(model view controller)?

Лабораторная работа №17

Разработка Тест-плана

Тест план (Test Plan) – это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Рекомендации по написанию Тест Плана.

Каждая методология или процесс пытаются навязать нам свои форматы оформления планов тестирования. Рассмотрим, как пример, шаблоны тест планов от [RUP \(Rational Unified Process\)](#) и стандарт [IEEE 829](#).

Присмотревшись внимательнее, становится ясно, что оба документа описывают одно и то же, но в разной форме. В случае, если вам не подходит стандартный шаблон или вы решили придумать свой собственный, более подходящий для вас формат документа, то хороший тест план должен как минимум описывать следующее:

1. Что надо тестировать?

Описание объекта тестирования: системы, приложения, оборудование.

2. Что будете тестировать?

Список функций и описание тестируемой системы и ее компонентов в отдельности.

3. Как будете тестировать?

Стратегия тестирования, а именно: виды тестирования и их применение по отношению к объекту тестирования.

Все виды тестирования программного обеспечения, в зависимости от преследуемых целей, можно условно разделить на **функциональные, нефункциональные, связанные с изменениями**.

Функциональные виды тестирования.

Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех **уровнях тестирования: компонентном или модульном (Component/Unit testing), интеграционном (Integration testing), системном (System testing) и приемочном (Acceptance testing)**. Функциональные виды тестирования рассматривают внешнее поведение системы. Далее перечислены одни из самых распространенных видов функциональных тестов:

- **Функциональное тестирование (Functional testing)**
- **Тестирование безопасности (Security and Access Control Testing)**
- **Тестирование взаимодействия (Interoperability Testing)**

Нефункциональные виды тестирования.

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом, это тестирование того, «Как» система работает. Далее перечислены основные виды нефункциональных тестов:

- **Все виды тестирования производительности**
- **Тестирование установки**
- **Тестирование удобства пользования**
- **Тестирование на отказ и восстановление**
- **Конфигурационное тестирование**

Связанные с изменениями виды тестирования.

После проведения необходимых изменений, таких как бага/дефекта, программное обеспечение должно быть перетестировано для подтверждения того факта, что проблема была действительно решена. Ниже перечислены виды тестирования, которые необходимо проводить после установки программного обеспечения, для подтверждения работоспособности приложения или правильности осуществленного исправления дефекта:

- **Дымовое тестирование**
- **Регрессионное тестирование**
- **Тестирование сборки**
- **Санитарное тестирование или проверка согласованности/исправности (Sanity Testing)**

4. Когда будете тестировать?

Последовательность проведения работ: подготовка (Test Preparation), тестирование (Testing), анализ результатов (Test Result Analisys) в разрезе запланированных фаз разработки.

5. Критерии начала тестирования:

- Готовность тестовой платформы
- Законченность разработки требуемого функционала
- Наличие всей необходимой документации
- ...

6. Критерии окончания тестирования:

- Результаты тестирования удовлетворяют критериям качества продукта:
 - Требования к количеству открытых багов выполнены
 - Выдержка определенного периода без изменения исходного кода приложения Code Freeze (CF)
 - Выдержка определенного периода без открытия новых багов Zero Bug Bounce (ZBB)
 - ...

Ответив в своем тест плане на вышеперечисленные вопросы, можно считать, что у вас на руках уже есть хороший черновик документа по планированию тестирования. Далее, чтобы документ приобрел более менее серьезный вид, нужно дополнить его следующими пунктами:

- Окружение тестируемой системы (описание программно-аппаратных средств)
- Необходимое для тестирования оборудование и программные средства (тестовый стенд и его конфигурация, программы для автоматизированного тестирования и т.д.)
- Риски и пути их разрешения

Виды тест планов

Чаще всего на практике приходится сталкиваться со следующими видами тест планов:

1. Мастер Тест План (Master Plan or Master Test Plan)
2. Тест План (Test Plan), назовем его детальный тест план
3. План приемочных испытаний (Product Acceptance Plan) – документ, описывающий набор действий, связанных с приемочным тестированием (стратегия, дата проведения, ответственные работники и т.д.)

Явное отличие Мастер Тест Плана от просто Тест Плана в том, что мастер тест план является более статичным в силу того, что содержит в себе высокоуровневую (High Level) информацию, которая не подвержена частому изменению в процессе тестирования и пересмотра требований. Сам же детальный тест план, который содержит более конкретную информацию по стратегии, видам тестирования, расписанию выполнения работ, является «живым» документом, который постоянно претерпевает изменения, отражающие реальное положение дел на проекте.

В повседневной жизни на проекте может быть один Мастер Тест План и несколько детальных тест планов, описывающих отдельные модули одного приложения.

Шаблон для оформления тест плана в формате MS Word

<i>Project Name</i>		<i>Prepared By</i>	
Validated By		Dated	
Plan Version			

Test Plan Contents

1. Introduction

1.1 Purpose

1.2 Background

1.3 Scope

1.4 Project Identification

2. Test Approach

2.1 Testing Stages

2.2 Testing Types

3. Test Deliverables

4. Environmental Needs

4.1 Hardware

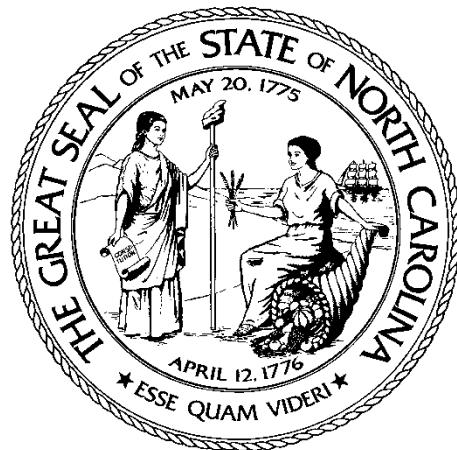
4.2 Software**4.3 Tools****5. Staffing****5.1 Responsibilities****5.2 Training****6. Dependencies/Risks****7. Schedule and Milestones****8. Approvals****Пример 2 тест плана****Software Test Plan (STP) Template**

Items that are intended to stay in as part of your document are in **bold**; explanatory comments are in *italic* text. Plain text is used where you might insert wording about your project.

This document is an annotated outline for a Software Test Plan, adapted from the IEEE Standard for Software Test Documentation (Std 829-1998).

Tailor as appropriate. Where you decide to omit a section, you might keep the header, but insert a comment saying why you omit the element.

Agency Name



Project Name

Software Quality Assurance Plan

Version: (n)

Date: (mm/dd/yyyy)

Document History and Distribution

Revision History

Distribution

Introduction

(NOTE 1: THE SOFTWARE TEST PLAN GUIDELINES WERE DERIVED AND DEVELOPED FROM IEEE STANDARD FOR SOFTWARE TEST DOCUMENTATION (829-1998)).

(Note 2: The ordering of Software Test Plan (STP) elements is not meant to imply that the sections or subsections must be developed or presented in that order. The order of presentation is intended for ease of use, not as a guide to preparing the various elements of the Software Test Plan. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content.)

The Introduction section of the Software Test Plan (STP) provides an overview of the project and the product test strategy, a list of testing deliverables, the plan for development and evolution of the STP, reference material, and agency definitions and acronyms used in the STP.

The Software Test Plan (STP) is designed to prescribe the scope, approach, resources, and schedule of all testing activities. The plan must identify the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.

1.1 Objectives

(Describe, at a high level, the scope, approach, resources, and schedule of the testing activities. Provide a concise summary of the test plan objectives, the products to be delivered, major work activities, major work products, major milestones, required resources, and master high-level schedules, budget, and effort requirements.)

1.2 Testing Strategy

Testing is the process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item. (This may appear as a specific document (such as a Test Specification), or it may be part of the organization's standard test approach. For each level of testing, there should be a test plan and an appropriate set of deliverables. The test strategy should be clearly defined and the Software Test Plan acts as the high-level test plan. Specific testing activities will have their own test plan. Refer to section 5 of this document for a detailed list of specific test plans.)

Specific test plan components include:

-
- Purpose for this level of test,
 - Items to be tested,
 - Features to be tested,
 - Features not to be tested,
 - Management and technical approach,
 - Pass / Fail criteria,
 - Individual roles and responsibilities,
 - Milestones,
 - Schedules, and
 - Risk assumptions and constraints.

1.3 Scope

(Specify the plans for producing both scheduled and unscheduled updates to the Software Test Plan (change management). Methods for distribution of updates shall be specified along with version control and configuration management requirements must be defined.)

Testing will be performed at several points in the life cycle as the product is constructed. Testing is a very 'dependent' activity. As a result, test planning is a continuing activity performed throughout the system development life cycle. Test plans must be developed for each level of product testing.

1.4 Reference Material

(Provide a complete list of all documents and other sources referenced in the Software Test Plan. Reference to the following documents (when they exist) is required for the high-level test plan:

- Project authorization,
- Project plan,
- Quality assurance plan,
- Configuration management plan,
- Organization policies and procedures, and
- Relevant standards.)

1.5 Definitions and Acronyms

(Specify definitions of all terms and agency acronyms required to properly interpret the Software Test Plan. Reference may be made to the Glossary of Terms on the IRMC web page.)

Test Items

(Specify the test items included in the plan. Supply references to the following item documentation:

- Requirements specification,
- Design specification,
- Users guide,
- Operations guide,
- Installation guide,
- Features (availability, response time),
- Defect removal procedures, and
- Verification and validation plans.)

2.1 Program Modules

(Outline testing to be performed by the developer for each module being built.)

2.2 Job Control Procedures

(Describe testing to be performed on job control language (JCL), production scheduling and control, calls, and job sequencing.)

2.3 User Procedures

(Describe the testing to be performed on all user documentation to ensure that it is correct, complete, and comprehensive.)

2.4 Operator Procedures

(Describe the testing procedures to ensure that the application can be run and supported in a production environment (include Help Desk procedures)).

3. FEATURES TO BE TESTED

(Identify all software features and combinations of software features to be tested. Identify the test design specifications associated with each feature and each combination of features.)

4. Features Not To Be Tested

(Identify all features and specific combinations of features that will not be tested along with the reasons.)

5. Approach

(Describe the overall approaches to testing. The approach should be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to do each task. Identify the types of testing to be performed along with the methods and criteria to be used in performing test activities. Describe the specific methods and procedures for each type of testing. Define the detailed criteria for evaluating the test results.)

(For each level of testing there should be a test plan and the appropriate set of deliverables. Identify the inputs required for each type of test. Specify the source of the input. Also, identify the outputs from each type of testing and specify the purpose and format for each test output. Specify the minimum degree of comprehensiveness desired. Identify the techniques that will be used to judge the comprehensiveness of the testing effort. Specify any additional completion criteria (e.g., error frequency). The techniques to be used to trace requirements should also be specified.)

5.1 Component Testing

(Testing conducted to verify the implementation of the design for one software element (e.g., unit, module) or a collection of software elements. Sometimes called unit testing. The purpose of component testing is to ensure that the program logic is complete and correct and ensuring that the component works as designed.)

5.2 Integration Testing

(Testing conducted in which software elements, hardware elements, or both are combined and tested until the entire system has been integrated. The purpose of integration testing is to ensure that design objectives are met and ensures that the software, as a complete entity, complies with operational requirements. Integration testing is also called System Testing.)

5.3 Conversion Testing

(Testing to ensure that all data elements and historical data is converted from an old system format to the new system format.)

5.4 Job Stream Testing

(Testing to ensure that the application operates in the production environment.)

5.5 Interface Testing

(Testing done to ensure that the application operates efficiently and effectively outside the application boundary with all interface systems.)

5.6 Security Testing

(Testing done to ensure that the application systems control and auditability features of the application are functional.)

5.7 Recovery Testing

(Testing done to ensure that application restart and backup and recovery facilities operate as designed.)

5.8 Performance Testing

(Testing done to ensure that the application performs to customer expectations (response time, availability, portability, and scalability)).

5.9 Regression Testing

(Testing done to ensure that applied changes to the application have not adversely affected previously tested functionality.)

5.10 Acceptance Testing

(Testing conducted to determine whether or not a system satisfies the acceptance criteria and to enable the customer to determine whether or not to accept the system. Acceptance testing ensures that customer requirements' objectives are met and that all components are correctly included in a customer package.)

5.11 Beta Testing

(Testing, done by the customer, using a pre-release version of the product to verify and validate that the system meets business functional requirements. The purpose of beta testing is to detect application faults, failures, and defects.)

6. Pass / Fail Criteria

(Specify the criteria to be used to determine whether each item has passed or failed testing.)

6.1 Suspension Criteria

(Specify the criteria used to suspend all or a portion of the testing activity on test items associated with the plan.)

6.2 Resumption Criteria

(Specify the conditions that need to be met to resume testing activities after suspension. Specify the test items that must be repeated when testing is resumed.)

6.3 Approval Criteria

(Specify the conditions that need to be met to approve test results. Define the formal testing approval process.)

7. Testing Process

(Identify the methods and criteria used in performing test activities. Define the specific methods and procedures for each type of test. Define the detailed criteria for evaluating test results.)

7.1 Test Deliverables

(Identify the deliverable documents from the test process. Test input and output data should be identified as deliverables. Testing report logs, test incident reports, test summary reports, and metrics' reports must be considered testing deliverables.)

7.2 Testing Tasks

(Identify the set of tasks necessary to prepare for and perform testing activities. Identify all intertask dependencies and any specific skills required.)

7.3 Responsibilities

(Identify the groups responsible for managing, designing, preparing, executing, witnessing, checking, and resolving test activities. These groups may include the developers, testers, operations staff, technical support staff, data administration staff, and the user staff.)

7.4 Resources

(Identify the resources allocated for the performance of testing tasks. Identify the organizational elements or individuals responsible for performing testing

(activities. Assign specific responsibilities. Specify resources by category. If automated tools are to be used in testing, specify the source of the tools, availability, and the usage requirements.)

7.5 Schedule

(Identify the high level schedule for each testing task. Establish specific milestones for initiating and completing each type of test activity, for the development of a comprehensive plan, for the receipt of each test input, and for the delivery of test output. Estimate the time required to do each test activity.)
(When planning and scheduling testing activities, it must be recognized that the testing process is iterative based on the testing task dependencies.)

8. Environmental Requirements

(Specify both the necessary and desired properties of the test environment including the physical characteristics, communications, mode of usage, and testing supplies. Also provide the levels of security required to perform test activities. Identify special test tools needed and other testing needs (space, machine time, and stationary supplies. Identify the source of all needs that is not currently available to the test group.)

8.1 Hardware

(Identify the computer hardware and network requirements needed to complete test activities.)

8.2 Software

(Identify the software requirements needed to complete testing activities.)

8.3 Security

(Identify the testing environment security and asset protection requirements.)

8.4 Tools

(Identify the special software tools, techniques, and methodologies employed in the testing efforts. The purpose and use of each tool shall be described. Plans for the acquisition, training, support, and qualification for each tool or technique.)

8.5 Publications

(Identify the documents and publications that are required to support testing activities.)

8.6 Risks and Assumptions

(Identify significant constraints on testing such as test item availability, test resource availability, and time constraints. Identify the risks and assumptions associated with testing tasks including schedule, resources, approach and documentation. Specify a contingency plan for each risk factor.)

9. Change Management Procedures

(Identify the software test plan change management process. Define the change initiation, change review, and change authorization process.)

10. Plan Approvals

(Identify the plan approvers. List the name, signature and date of plan approval.)

Задание: составить тест-план по своей предметной области для тестирования приложения, предметную область согласовать с преподавателем.

Контрольные вопросы:

- 1) Что такое тест-план?
- 2) Для чего нужен тест-план?
- 3) Структура тест-плана?
- 4) Виды функционального тестирования?

Лабораторная работа №18

Разработка мобильных приложений для ОС Андроид

Цель работы: Познакомиться с android. Установить Android Studio и Android SDK. Создать первое приложение

Общие краткие теоретические сведения.

Бурное развитие информационных технологий в последнее время привело к тому, что появилось много новых устройств и технологий, таких, как планшеты, смартфоны, нетбуки, другие гаджеты. Они все более прочно

входят в нашу жизнь и становятся привычным делом. Лидирующей платформой среди подобных гаджетов на сегодняшний день является ОС Андроид.

Android используется на самых разных устройствах. Это и смартфоны, и планшеты, и телевизоры, и смарт-часы, и ряд других гаджетов. По разным подсчетам за 2020 год этой операционной системой пользуются около 85% владельцев смартфонов, а общее количество пользователей смартфонов на ОС Android оценивается в более чем 2,5 млрд. человек по всему миру.

ОС Андроид была создана разработчиком Энди Рубином (Andy Rubin) в качестве операционной системы для мобильных телефонов и поначалу развивалась в рамках компании Android Inc. Но в 2005 году Google покупает Android Inc. и начинает развивать операционную систему с новой силой. Android постоянно эволюционирует, и вместе с операционной системой эволюционируют средства и инструменты для разработки. На данный момент (ноябрь 2020 года) последней версией является Android 11.0, которая вышла в сентябре 2020 года

Что нужно для разработки?

Стоит отметить, что разрабатывать приложения под Android можно с помощью различных фреймворков и языков программирования. Так как языков программирования могут применяться Java, Kotlin, Dart (фреймворк Flutter), C++, Python, C# (платформа Xamarin) и т.д. В данном руководстве мы будем использовать именно язык Java, как наиболее распространенный и используемый. Поэтому прежде чем приступить к освоению программирования под Android по данному руководству, необходимо освоить хотя бы базовые моменты языка Java.

Установка средств разработки

Существуют разные среды разработки для Android. Рекомендуемой средой разработки является Android Studio, которая создана специально для разработки под ОС Android. Поэтому мы ее и будем использовать. Загрузить файл установщика можно с официального сайта:

<https://developer.android.com/studio>

Важно! Установка Android Studio. Заметка

Для корректной установки Android Studio необходимо, чтобы название учётной записи пользователя было на английском языке. В противном

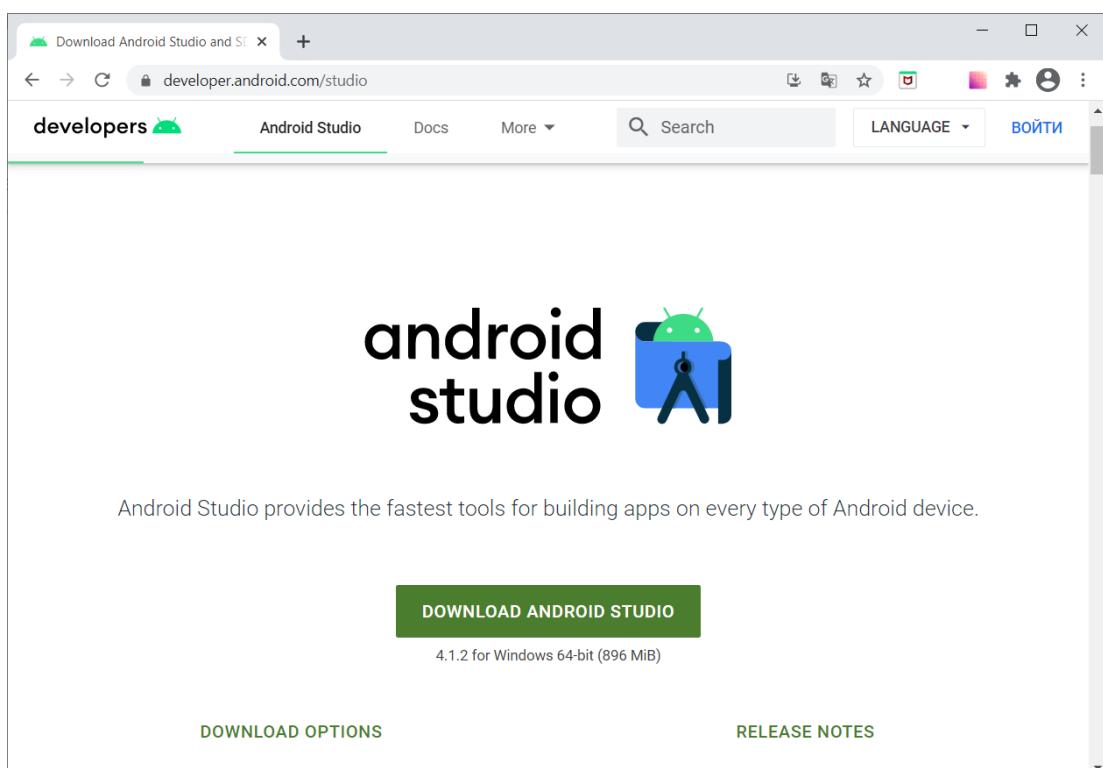
случае Android Studio может не работать!

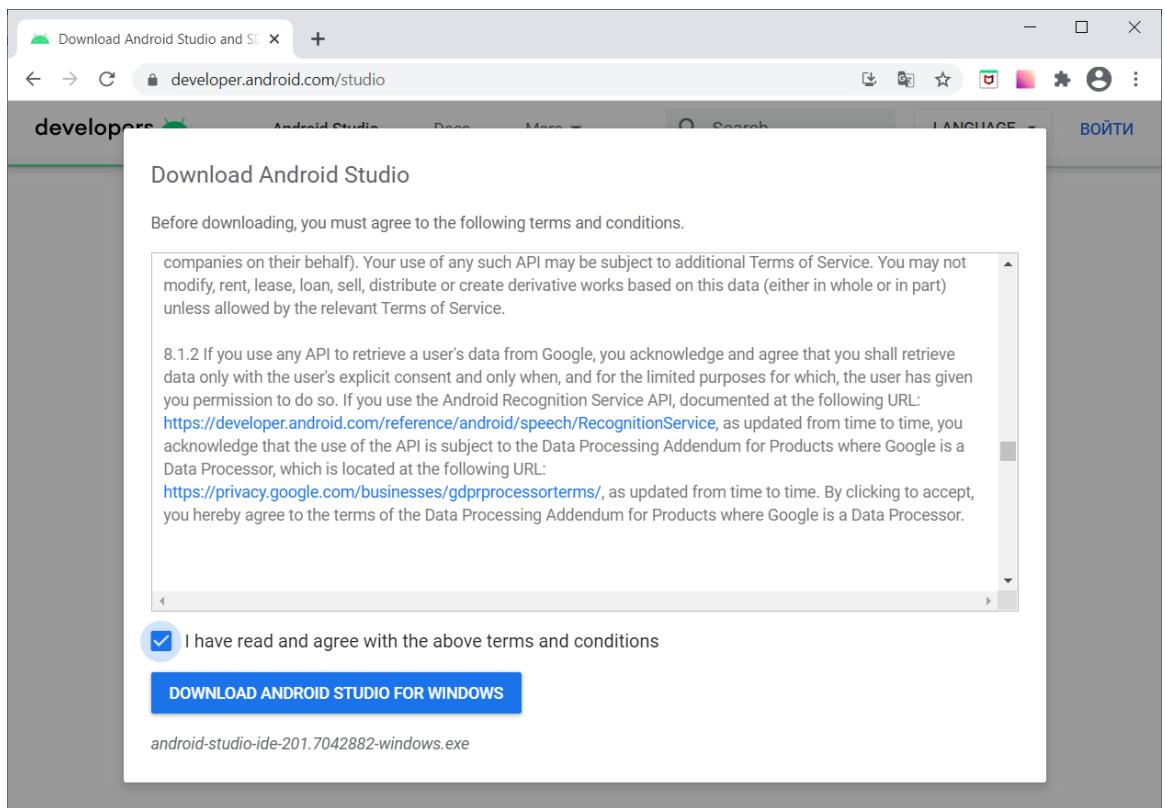
Поэтому, если вы собираетесь устанавливать Android Studio на компьютер с названием учётной записи пользователя на кириллице, прежде всего вам нужно переименовать её, либо создать новую на английском языке. Можете погуглить "учетная запись windows как переименовать". Вот пара ссылок по этому запросу

<https://answers.microsoft.com/ru-ru/windows/forum/all/%D0%BA%D0%B0%D0%BA/fc7f2cb9-6d5c-488a-a782-1f7b0ce05c74>

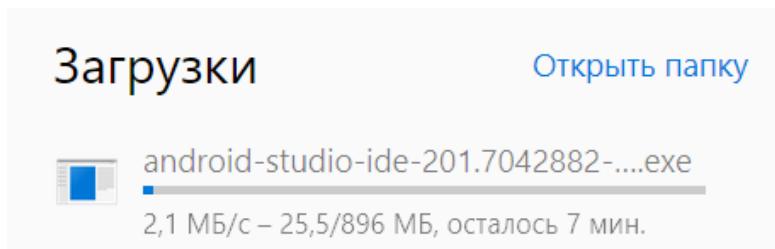
https://answers.microsoft.com/ru-ru/windows/forum/windows_7-performance/%D0%BA%D0%B0%D0%BA/0ed802de-d597-44c6-8dcf-d6002ffce86d

Официальный сайт:

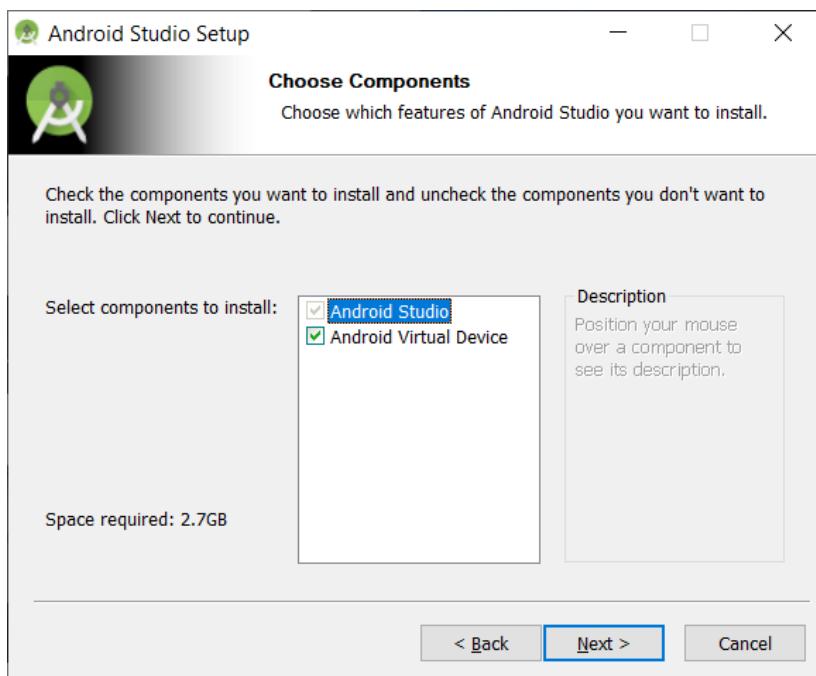




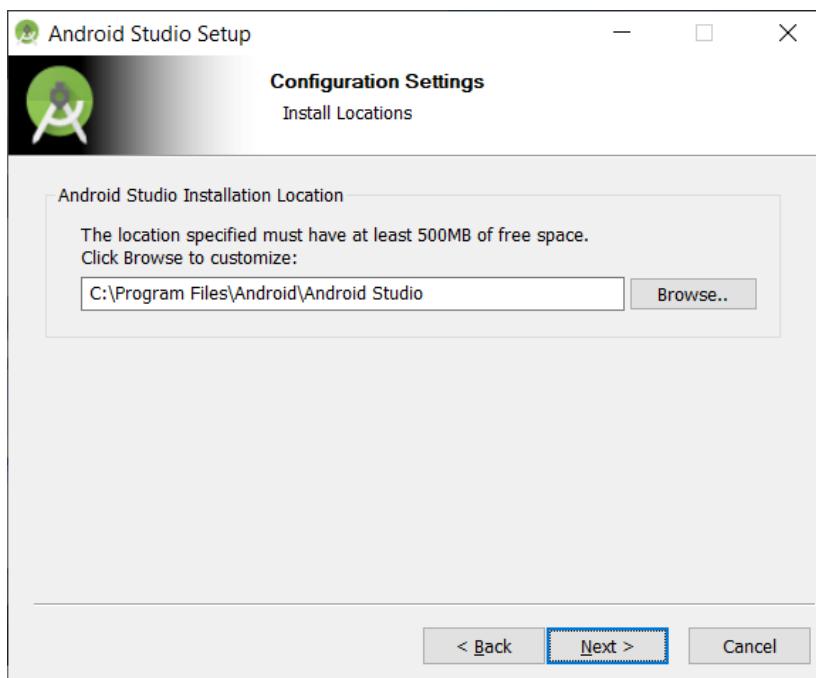
Загрузка может занять какое-то время:

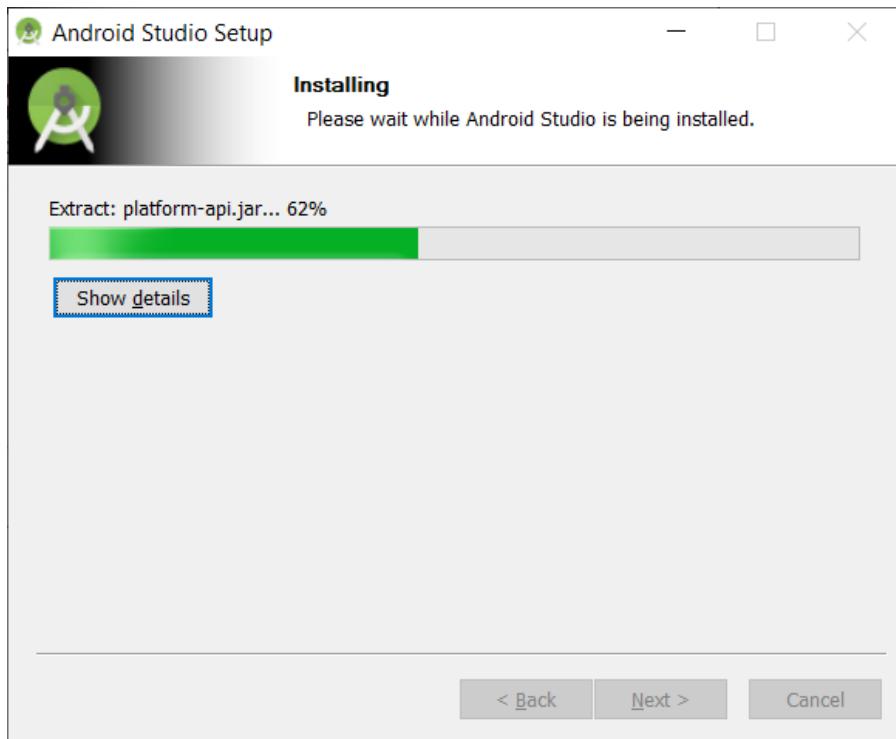


Оставляем как есть, жмем Next

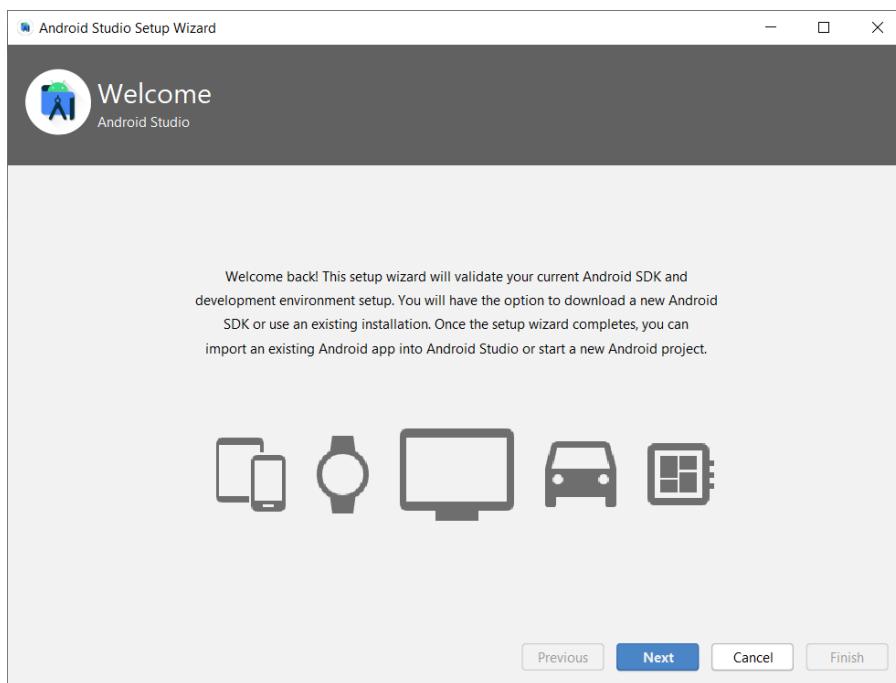


Выбираем расположение куда хотим установить, жмем Next

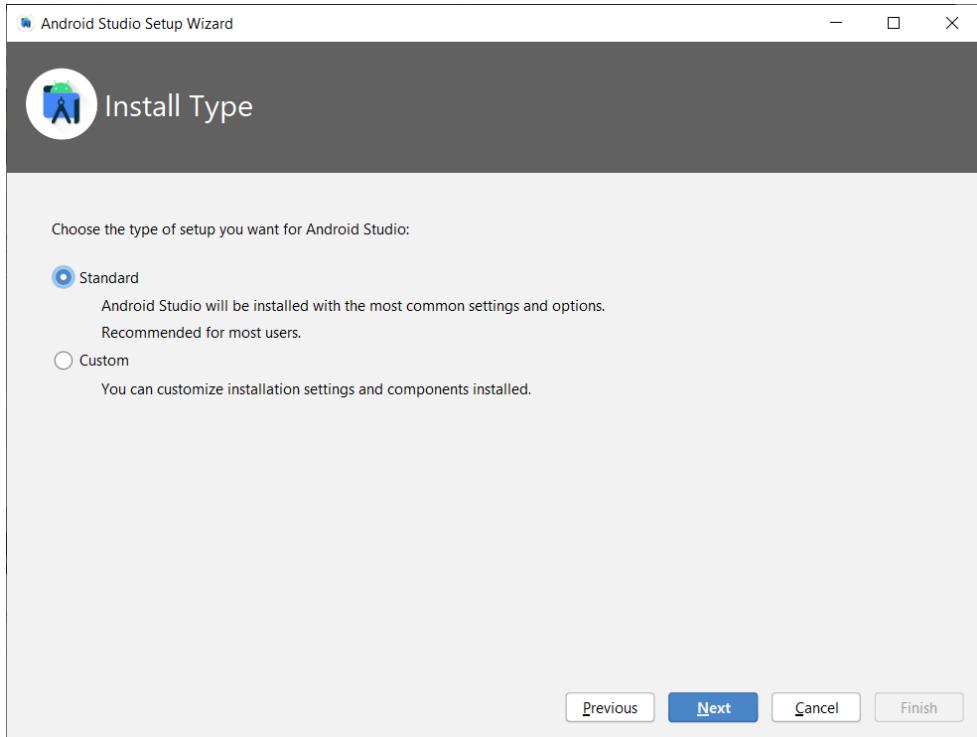




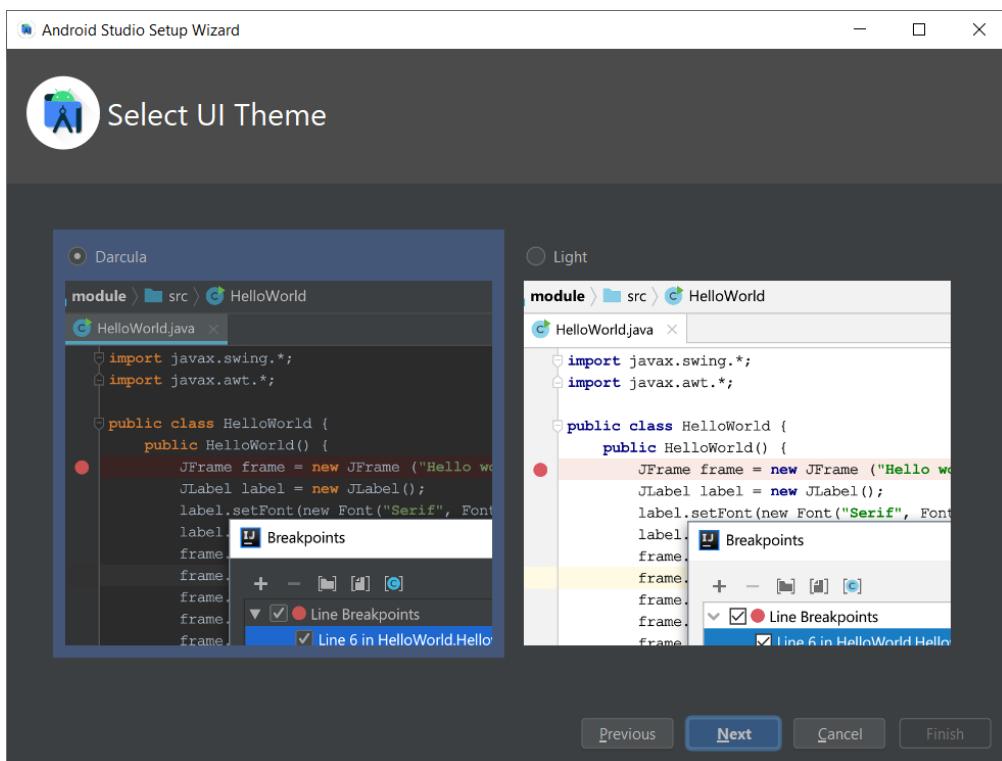
Откроется такое окно, жмем здесь снова Next.



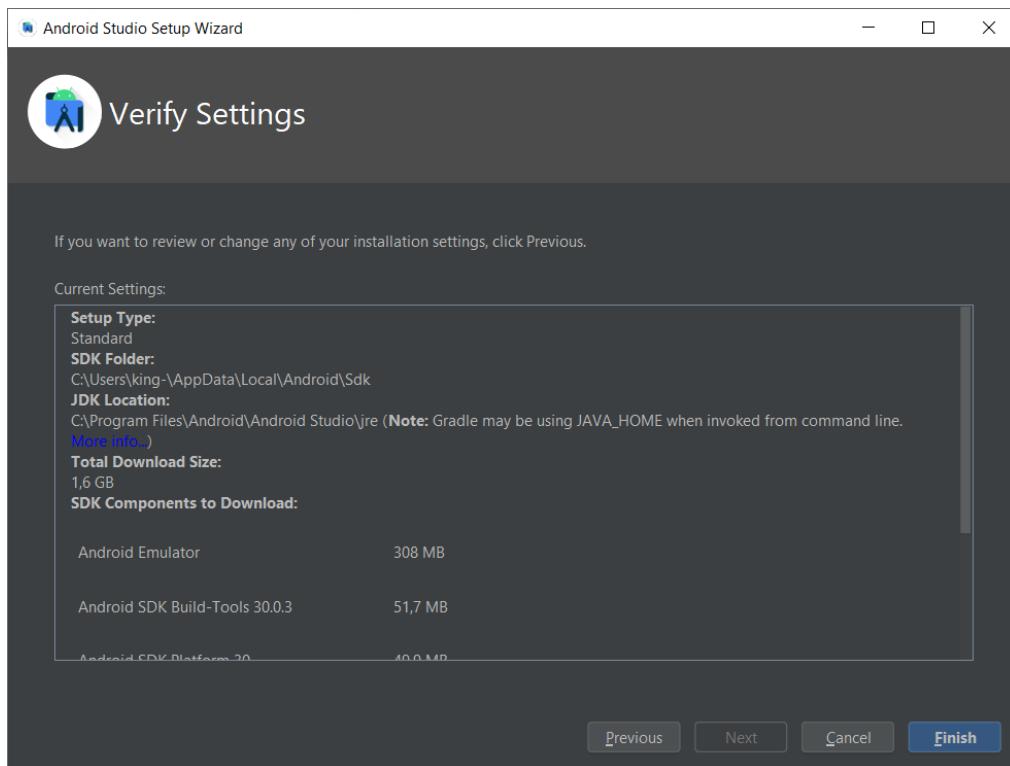
Выбираем Standard и жмем Next



Выбираем тему на свое усмотрение, жмем Next



Визард сообщает нам, что ему необходимо загрузить несколько компонентов для завершения установки, жмем Finish



Дальше пойдет процесс загрузки

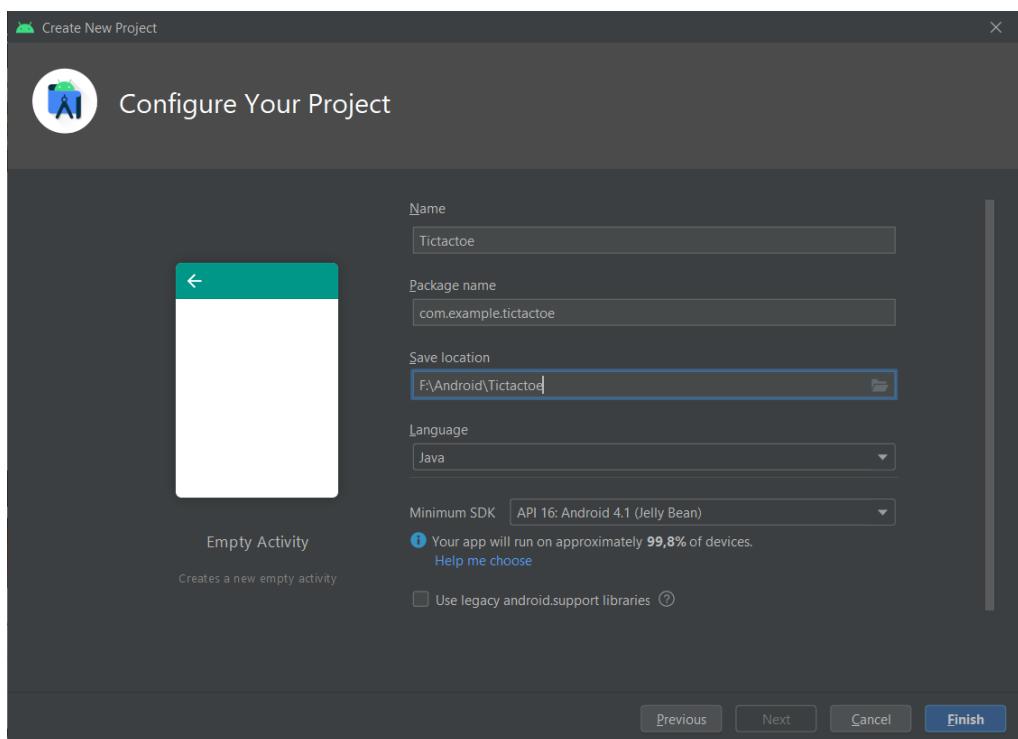
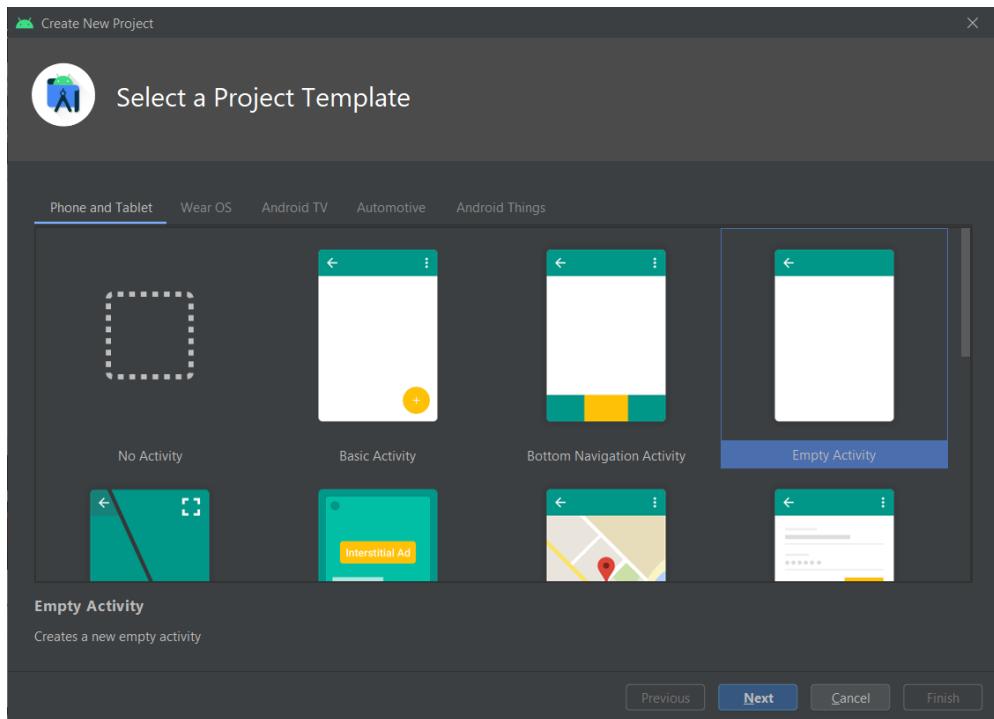
Создание первого приложения

Пример выполнения задания

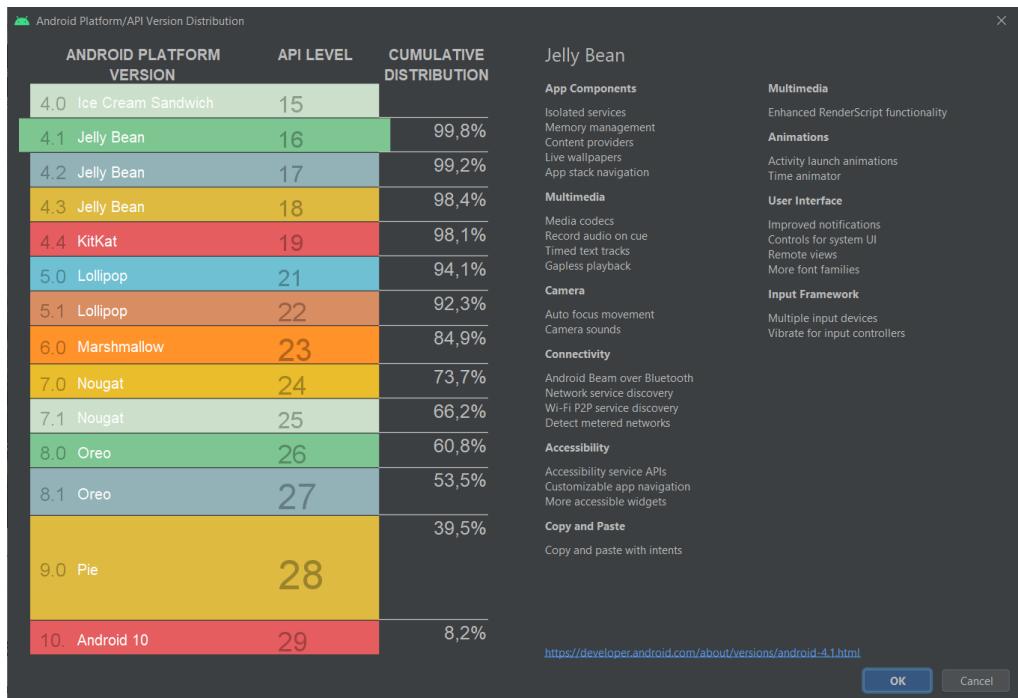
Задание: создать приложение игры в крестики-нолики.

Мы затронем весь цикл разработки приложения. Вместе мы напишем простенькую игру “Крестики-Нолики” с одним экраном.





Параметр Minimum SDK это минимальная операционная система, которая будет поддерживаться вашим приложением, чем версия ниже, тем больше телефонов смогут его поддерживать



Вам сразу откроется проект, в котором будут активны две вкладки `MainActivity.java`, где описывается логика вашего приложения и `activity_main.xml`, где создается интерфейс вашего приложения

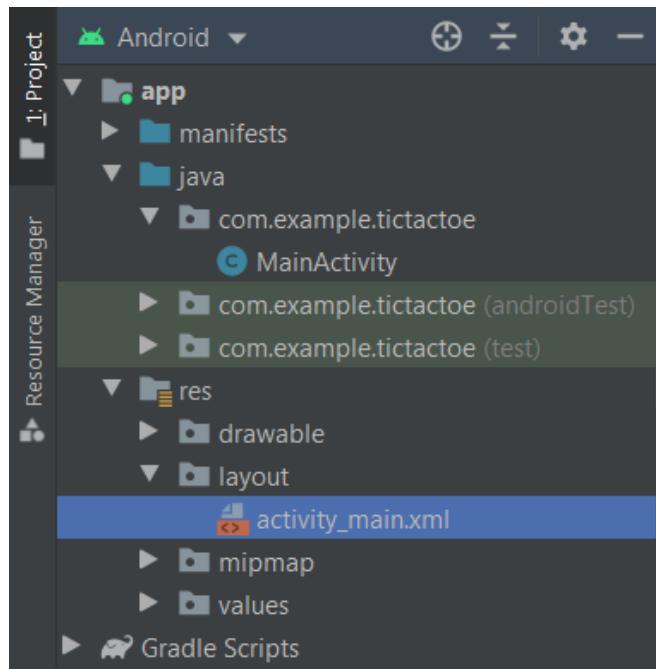
The screenshot shows the Android Studio interface with the project 'Tictactoe' open. The left sidebar displays the project structure with 'app' selected. The main editor window shows the 'activity_main.xml' file and the 'MainActivity.java' file. The Java code defines a MainActivity that extends AppCompatActivity and overrides the onCreate method to set the content view to R.layout.activity_main.

```
package com.example.tictactoe;
import ...
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

The screenshot shows the Android Studio interface with the project 'Tictactoe' open. The left sidebar displays the project structure with 'app' selected. The main editor window shows the 'activity_main.xml' file in the XML layout editor. The XML code defines a ConstraintLayout containing a single TextView with the text "Hello World!". The right side of the screen shows the visual representation of the layout, featuring a white background with a blue rectangular placeholder for the text view.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Если вы случайно закроете одно из них, то вот где они располагаются в вашем проекте, кликните двойным щелчком мыши, чтобы открыть



Чтобы запустить ваше приложение есть два способа:

- 1) С помощью эмулятора в Android Studio
- 2) Сразу на вашем мобильном устройстве

Рассмотрим оба варианта, но я вам больше рекомендую использовать второй вариант, так как это намного быстрее и не нужно выделять память на компьютере (около 10 гб) на каждое виртуальное устройство.

Замечание:

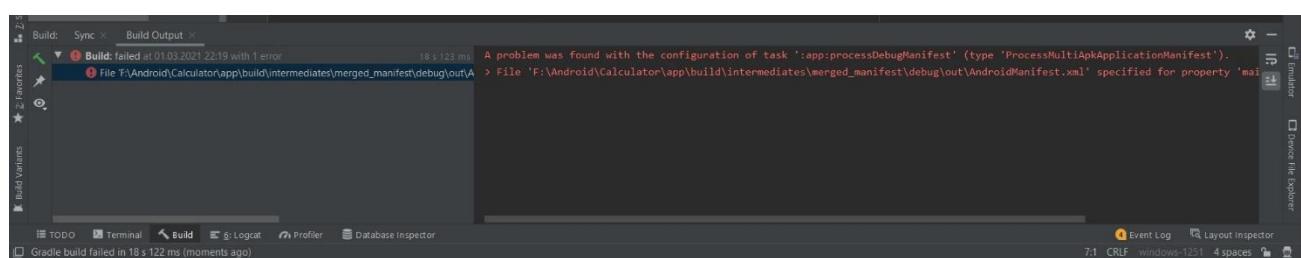
Если во время запуска приложения выскоцила такая ошибка:

a problem was found with the configuration of task app:processDebugManifest type ProcessMultiapkApplicationManifest

specified for property 'mainMergedManifest' does not exist.

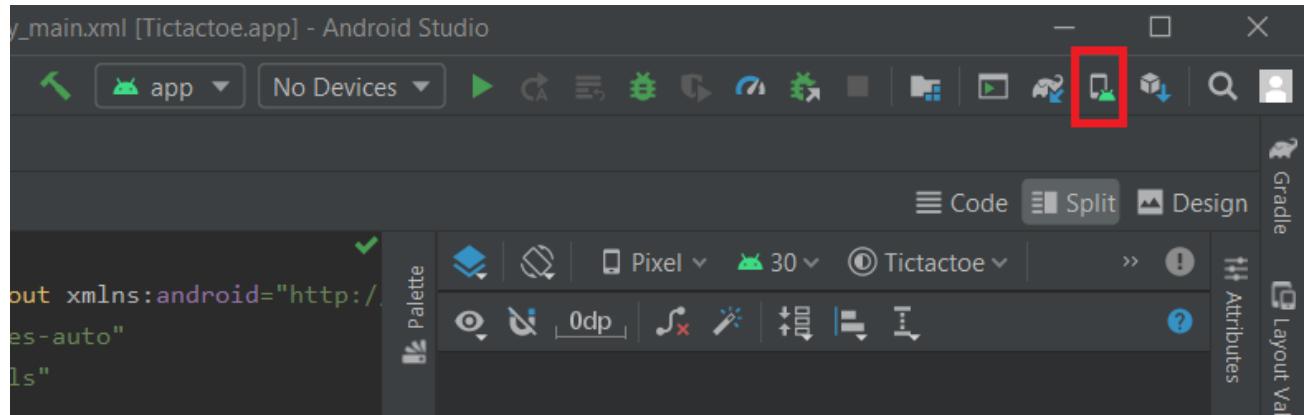
Рекомендую посмотреть видео с решением этой проблемы по ссылке:

<https://www.youtube.com/watch?v=U420dx6C60I>

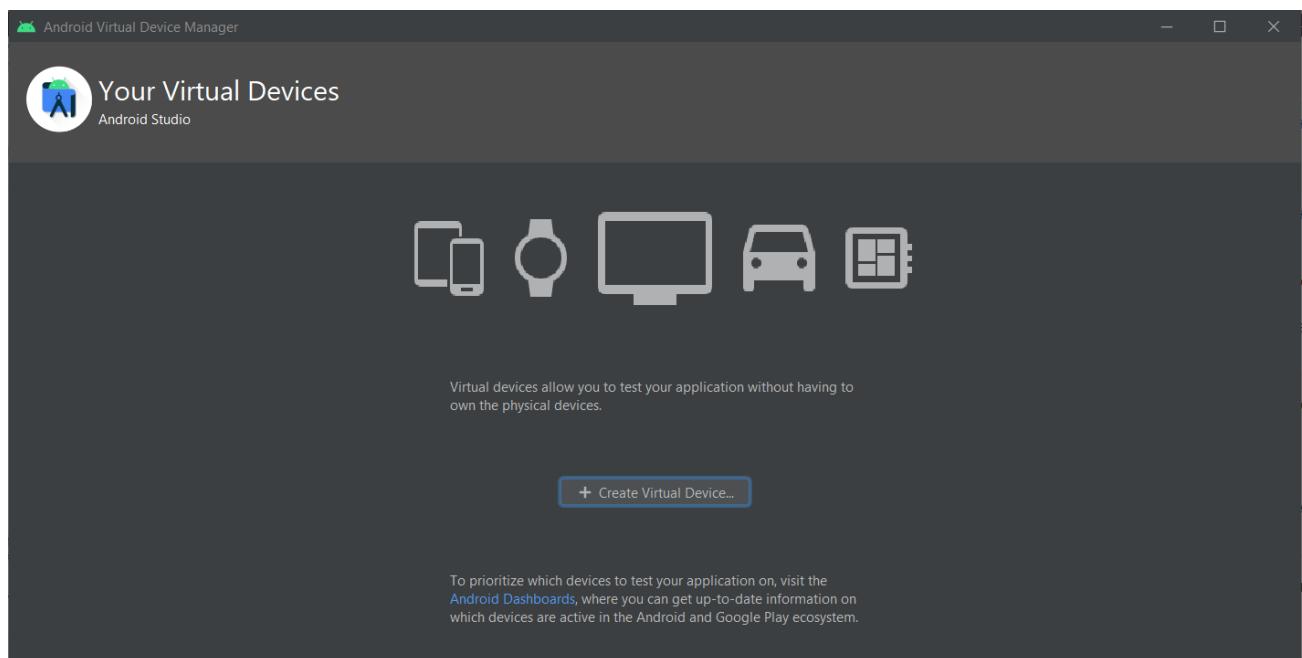


1) Способ запуска через эмулятор

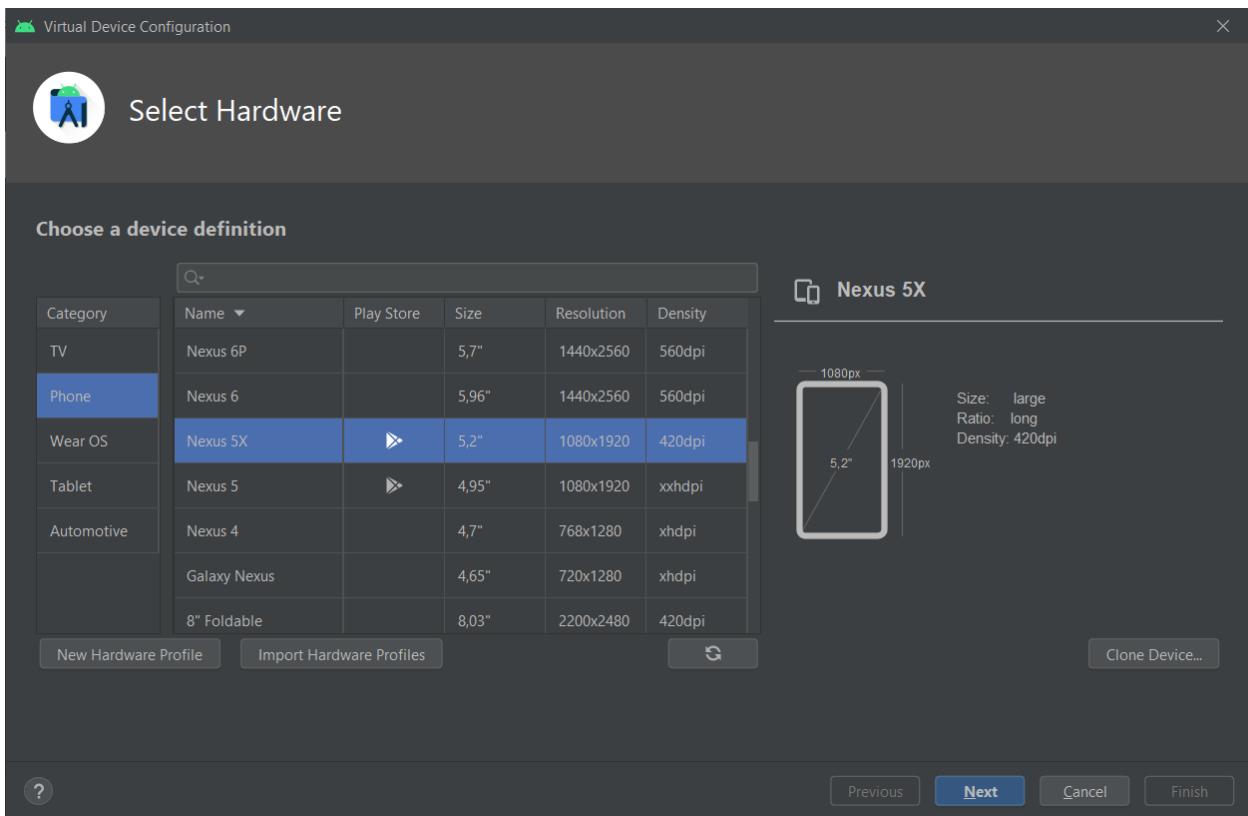
В верхнем правом углу нажимаем на значок, показанный ниже на скриншоте



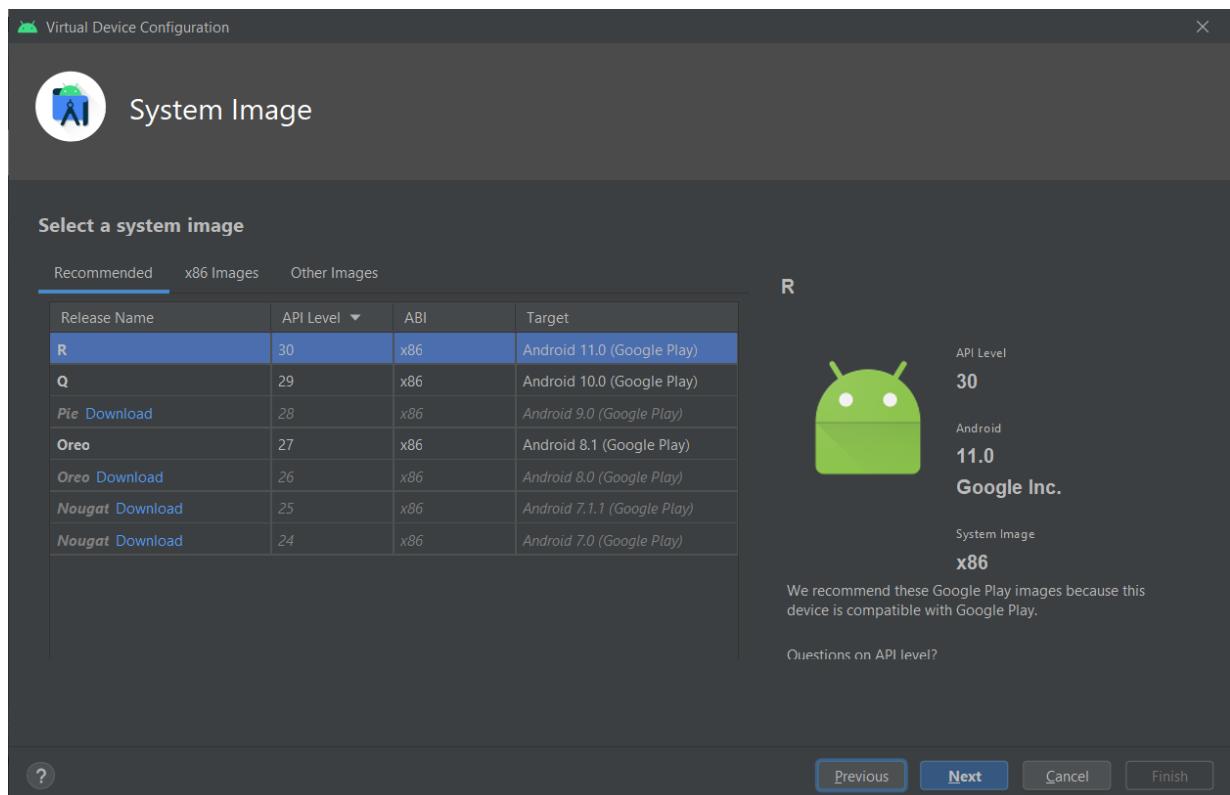
Откроется окно. Нажимаем здесь на Create Virtual Device...



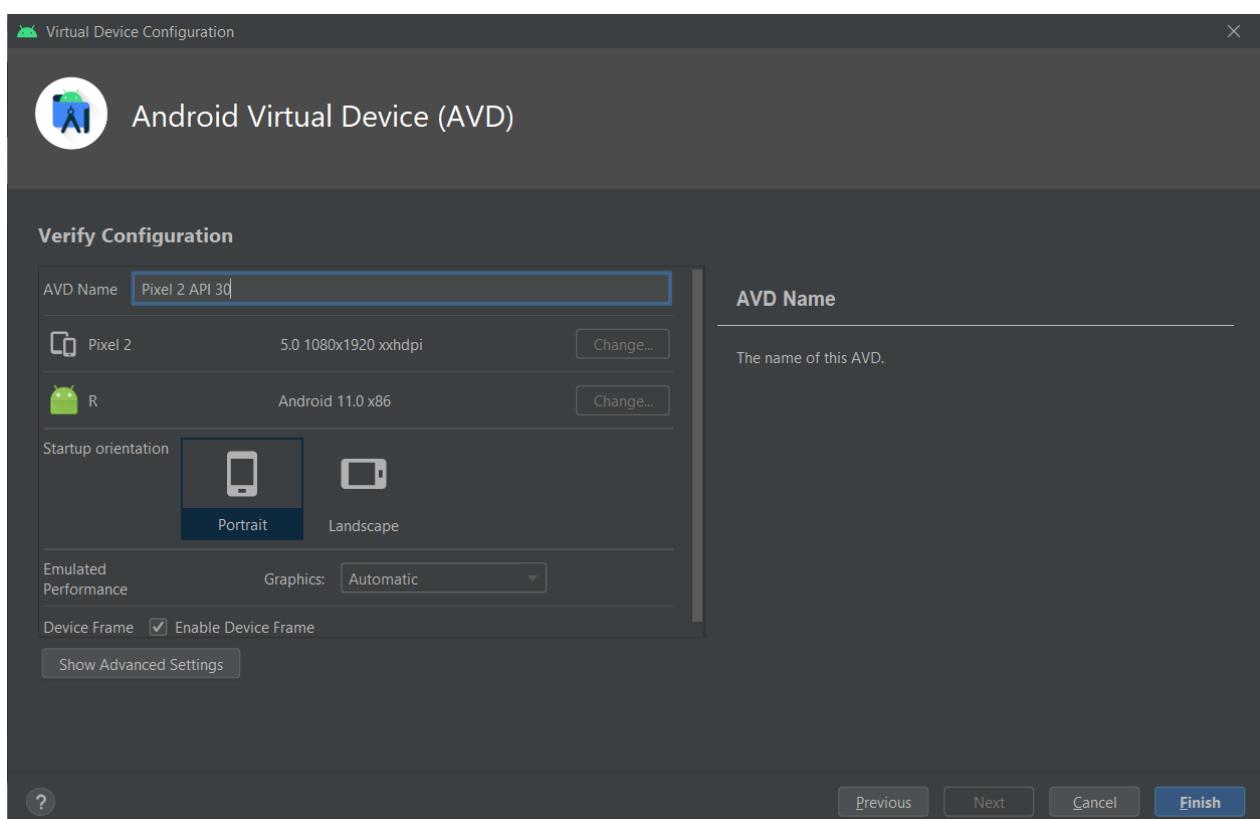
Выбираем любой смартфон, жмем Next



Здесь необходимо скачать версию операционной системы для вашего эмулятора, после скачивания выбираем ее и жмем Next

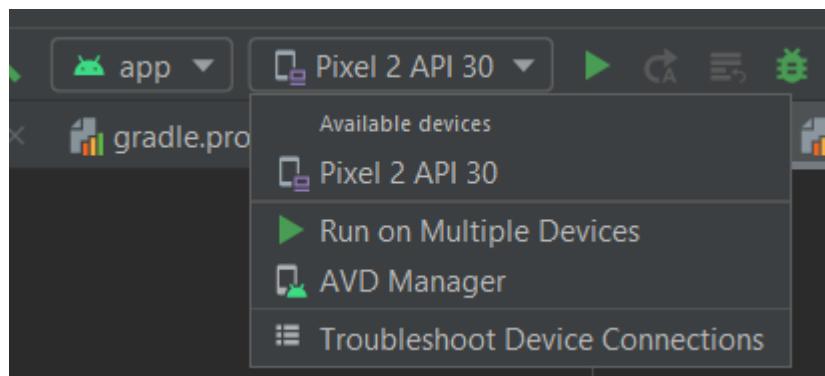


Оставляем без изменений и жмем Finish



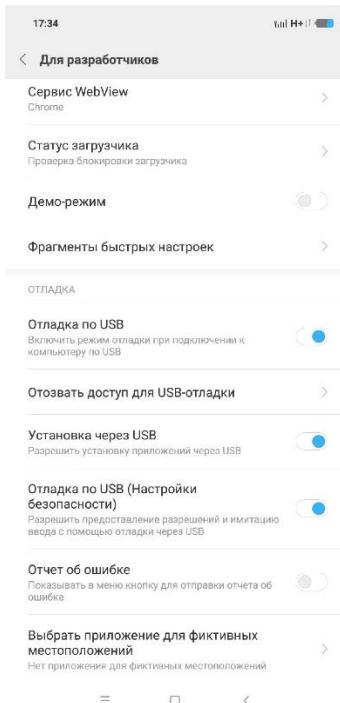
Теперь здесь отобразится ваше устройство, так же если нажать на кнопку AVD Manager, то можно увидеть более подробную информацию и для запуска приложения

нажимаем на зеленый треугольник



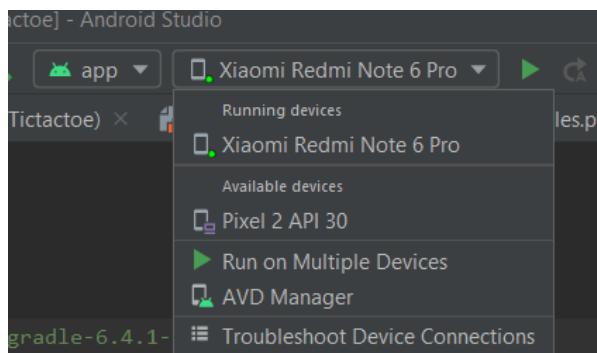
2) Запуск на реальном устройстве

- 1) Подключите ваш телефон через usb кабель к компьютеру
- 2) Выполните следующие шаги, чтобы включить отладку по USB в окне параметров разработчика.
 - a) Откройте настройки
 - b) Если ваш телефон использует версию Android 8.0 и выше выберите Система (System). В противном случае переходите к следующему шагу
 - c) В настройках найдите пункт “О телефоне” (About phone).
 - d) Найдите Номер сборки (Build number) и щелкните 7 раз для получения статуса разработчика
 - e) Теперь вернитесь назад и найдите “Для разработчиков” или “Параметры разработчика” (Developer options)
 - f) Здесь поставьте флажки на “Откладка по USB” и “Установка через USB” и “Откладка по USB (настройка безопасности)” (На последнее, если есть)



У вас что-то может отличаться в этом случае посмотрите в интернете, где что находится на вашей модели телефона.

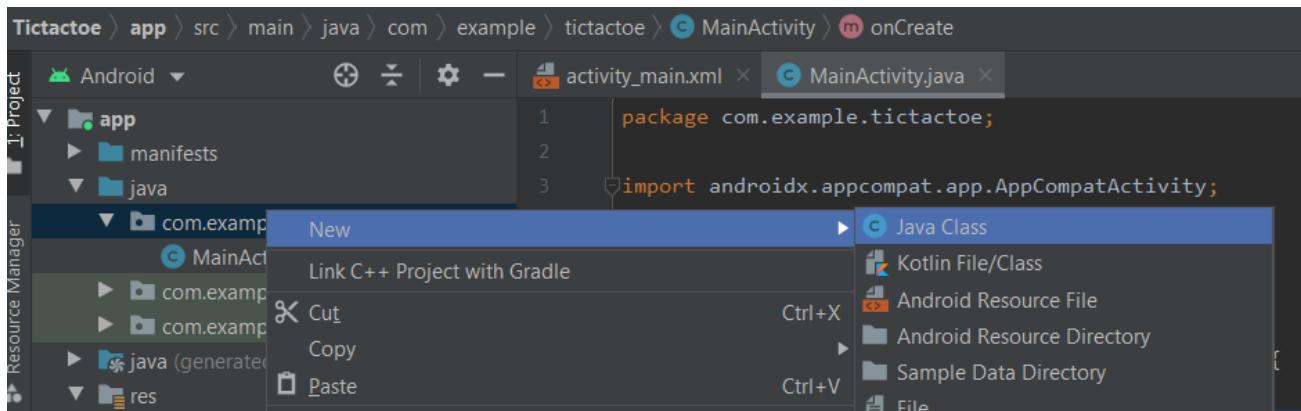
Если все сделали правильно, то теперь при подключении вашего телефона через USB к компьютеру он будет сразу обнаруживаться Android Studio



Для запуска также нажимаем на зеленый треугольник, после создания сборки на ваш телефон придет сообщение с установить или отказаться

Теперь перейдем к написанию кода

Для добавление новых классов нажимаем на необходимую вкладку правой кнопкой мыши и делаем как на скриншоте



Для нашего приложения идеально подойдёт макет **TableLayout** с идентификатором **android:id="@+id/tableLayout"**.

Заходим в **activity_main.xml** и заменяем весь код на следующий

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/tableLayout"
    android:gravity="center">
</TableLayout>
```

Теперь заходим в **MainActivity.java** и заменяем весь код на следующий

```
package com.example.tictactoe;

import android.os.Bundle;
import android.widget.TableLayout;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private TableLayout tableLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        tableLayout = findViewById(R.id.tableLayout);
        buildGameField(); // создание игрового поля
    }
}
```

Теперь необходимо реализовать метод **buildGameField()**. Для этого требуется создать поле в виде матрицы. Этим будет заниматься класс **Game**. Сначала нужно

создать класс **Player**, объекты которого будут заполнять ячейки игрового поля и класс **Square** для самих ячеек.

Создаем класс **Player.java**

```
package com.example.tictactoe;

public class Player
{
    private String name;

    public Player(String name)
    {
        this.name = name;
    }

    public CharSequence getName()
    {
        return name;
    }
}
```

Square.java

```
package com.example.tictactoe;

public class Square
{
    private Player player = null;

    public void fill(Player player)
    {
        this.player = player;
    }

    public boolean isFilled()
    {
        return player != null;
    }

    public Player getPlayer()
    {
        return player;
    }
}
```

Game.java

```
package com.example.tictactoe;

public class Game
{
    private Square[][] field;
    private int squareCount;
```

```
public Game()
{
    field = new Square[3][3];
    squareCount = 0;
    // заполнение поля
    for (int i = 0, l = field.length; i < l; i++)
    {
        for (int j = 0, l2 = field[i].length; j < l2; j++)
        {
            field[i][j] = new Square();
            squareCount++;
        }
    }
}

public Square[][] getField()
{
    return field;
}
}
```

Метод **buildGameField()** динамически добавляет строки и колонки в таблицу (игровое поле), его необходимо добавить в **MainActivity.Java**:

```
private Button[][] buttons = new Button[3][3];

private void buildGameField()
{
    Square[][] field = game.getField();
    for (int i = 0, lenI = field.length; i < lenI; i++ ) {
        TableRow row = new TableRow(this); // создание строки таблицы
        for (int j = 0, lenJ = field[i].length; j < lenJ; j++)
        {
            Button button = new Button(this);
            buttons[i][j] = button;
            button.setOnClickListener(new Listener(i, j)); // установка слушателя,
реагирующего на клик по кнопке
            row.addView(button, new
TableRow.LayoutParams(TableLayout.LayoutParams.WRAP_CONTENT,
                    TableRow.LayoutParams.WRAP_CONTENT)); // добавление кнопки в
строку таблицы
            button.setWidth(350);
            button.setHeight(350);
        }
        tableLayout.addView(row,
                new TableLayout.LayoutParams(TableLayout.LayoutParams.WRAP_CONTENT,
                    TableLayout.LayoutParams.WRAP_CONTENT)); // добавление строки в
таблицу
    }
}
```

И после строк:

```
import android.os.Bundle;
import android.view.View;
```

добавьте:

```
import android.widget.Button;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.Toast;
```

В коде создаётся объект, реализующий интерфейс **View.OnClickListener**. Создадим вложенный класс Listener. Он будет виден только из активности.

```
public class Listener implements View.OnClickListener
{
    private int x = 0;
    private int y = 0;

    Listener(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public void onClick(View view)
    {
        Button button = (Button) view;
    }
}
```

Осталось реализовать логику игры. Возвращаемся к **Game.java**:

```
package com.example.tictactoe;

public class Game {
    // игроки
    private Player[] players;

    // поле
    private Square[][] field;

    // начата ли игра?
    private boolean started;

    // текущий игрок
    private Player activePlayer;

    // Считает количество заполненных ячеек
    private int filled;

    // Всего ячеек
    private int squareCount;

    public Game() {
```

```
    field = new Square[3][3];
    squareCount = 0;
    // заполнение поля
    for (int i = 0, l = field.length; i < l; i++) {
        for (int j = 0, l2 = field[i].length; j < l2; j++) {
            field[i][j] = new Square();
            squareCount++;
        }
    }
    players = new Player[2];
    started = false;
    activePlayer = null;
    filled = 0;
}

public void start() {
    resetPlayers();
    started = true;
}

private void resetPlayers() {
    players[0] = new Player("X");
    players[1] = new Player("O");
    setCurrentActivePlayer(players[0]);
}

private void setCurrentActivePlayer(Player player) {
    activePlayer = player;
}

public Square[][] getField() {
    return field;
}

public boolean makeTurn(int x, int y) {
    if (field[x][y].isFilled()) {
        return false;
    }
    field[x][y].fill(getCurrentActivePlayer());
    filled++;
    switchPlayers();
    return true;
}

private void switchPlayers() {
    activePlayer = (activePlayer == players[0]) ? players[1] : players[0];
}

public Player getCurrentActivePlayer() {
    return activePlayer;
}

public boolean isFieldFilled() {
    return squareCount == filled;
}

public void reset() {
    resetField();
    resetPlayers();
}
```

```
    }

    private void resetField() {
        for (int i = 0, l = field.length; i < l; i++) {
            for (int j = 0, l2 = field[i].length; j < l2; j++) {
                field[i][j].fill(null);
            }
        }
        filled = 0;
    }
}
```

Определение победителя

Очевидно, что в крестики-нолики выигрывает тот, кто построит X или O в линию длиной, равной длине поля по вертикали, по горизонтали или по диагонали. Первая мысль, которая приходит в голову — это написать методы для каждого случая. Думаю, в этом случае хорошо подойдёт паттерн Chain of Responsibility. Определим интерфейс:

```
package com.example.tictactoe;

public interface WinnerCheckerInterface
{
    public Player checkWinner();
}
```

Так как **Game** наделён обязанностью выявлять победителя, он реализует этот интерфейс. Настало время создать виртуальных «лайнсменов», каждый из которых будет проверять свою сторону. Все они реализует интерфейс **WinnerCheckerInterface**.

WinnerCheckerHorizontal.java

```
package com.example.tictactoe;

public class WinnerCheckerHorizontal implements WinnerCheckerInterface {
    private Game game;

    public WinnerCheckerHorizontal(Game game) {
        this.game = game;
    }

    public Player checkWinner() {
        Square[][] field = game.getField();
        Player currPlayer;
        Player lastPlayer = null;
        for (int i = 0, len = field.length; i < len; i++) {
            lastPlayer = null;
            int successCounter = 1;
            for (int j = 0, len2 = field[i].length; j < len2; j++) {
                currPlayer = field[i][j].getPlayer();
                if (currPlayer == lastPlayer && (currPlayer != null && lastPlayer !=
```

```
        null)) {
            successCounter++;
            if (successCounter == len2) {
                return currPlayer;
            }
        }
        lastPlayer = currPlayer;
    }
}
return null;
}
```

WinnerCheckerVertical.java

```
package com.example.tictactoe;

public class WinnerCheckerVertical implements WinnerCheckerInterface {
    private Game game;

    public WinnerCheckerVertical(Game game) {
        this.game = game;
    }

    public Player checkWinner() {
        Square[][] field = game.getField();
        Player currPlayer;
        Player lastPlayer = null;
        for (int i = 0, len = field.length; i < len; i++) {
            lastPlayer = null;
            int successCounter = 1;
            for (int j = 0, len2 = field[i].length; j < len2; j++) {
                currPlayer = field[j][i].getPlayer();
                if (currPlayer == lastPlayer && (currPlayer != null && lastPlayer != null)) {
                    successCounter++;
                    if (successCounter == len2) {
                        return currPlayer;
                    }
                }
            }
            lastPlayer = currPlayer;
        }
    }
}
```

WinnerCheckerDiagonalLeft.java

```
package com.example.tictactoe;

public class WinnerCheckerDiagonalLeft implements WinnerCheckerInterface {
    private Game game;

    public WinnerCheckerDiagonalLeft(Game game) {
```

```
        this.game = game;
    }

    public Player checkWinner() {
        Square[][] field = game.getField();
        Player currPlayer;
        Player lastPlayer = null;
        int successCounter = 1;
        for (int i = 0, len = field.length; i < len; i++) {
            currPlayer = field[i][i].getPlayer();
            if (currPlayer != null) {
                if (lastPlayer == currPlayer) {
                    successCounter++;
                    if (successCounter == len) {
                        return currPlayer;
                    }
                }
            }
            lastPlayer = currPlayer;
        }
        return null;
    }
}
```

WinnerCheckerDiagonalRight.java

```
package com.example.tictactoe;

public class WinnerCheckerDiagonalRight implements WinnerCheckerInterface {
    private Game game;

    public WinnerCheckerDiagonalRight(Game game) {
        this.game = game;
    }

    public Player checkWinner() {
        Square[][] field = game.getField();
        Player currPlayer;
        Player lastPlayer = null;
        int successCounter = 1;
        for (int i = 0, len = field.length; i < len; i++) {
            currPlayer = field[i][len - (i + 1)].getPlayer();
            if (currPlayer != null) {
                if (lastPlayer == currPlayer) {
                    successCounter++;
                    if (successCounter == len) {
                        return currPlayer;
                    }
                }
            }
            lastPlayer = currPlayer;
        }
        return null;
    }
}
```

Проинициализируем их в конструкторе **Game**:

```
// "Судьи". После каждого хода они будут проверять,  
// нет ли победителя  
private WinnerCheckerInterface[] winnerCheckers;  
  
// Инициализация "судей"  
winnerCheckers = new WinnerCheckerInterface[4];  
winnerCheckers[0] = new WinnerCheckerHorizontal(this);  
winnerCheckers[1] = new WinnerCheckerVertical(this);  
winnerCheckers[2] = new WinnerCheckerDiagonalLeft(this);  
winnerCheckers[3] = new WinnerCheckerDiagonalRight(this);
```

Реализация **checkWinner()**:

```
public Player checkWinner() {  
    for (WinnerCheckerInterface winChecker : winnerCheckers) {  
        Player winner = winChecker.checkWinner();  
        if (winner != null) {  
            return winner;  
        }  
    }  
    return null;  
}
```

Возвращаемся в класс активности. Победителя проверяем после каждого хода.
Добавим код в метод **onClick()** класса **Listener**

```
public void onClick(View view)  
{  
    Button button = (Button) view;  
    Game g = game;  
    Player player = g.getCurrentActivePlayer();  
    if (makeTurn(x, y))  
    {  
        button.setText(player.getName());  
    }  
    Player winner = g.checkWinner();  
    if (winner != null)  
    {  
        gameOver(winner);  
    }  
    if (g.isFieldFilled())  
    { // в случае, если поле заполнено  
        gameOver();  
    }  
}
```

Метод **gameOver()** реализован в 2-х вариантах.

Метод **makeTurn()** проверяет очерёдность хода игрока, а

метод **refresh()** обновляет состояние поля:

Весь код **MainActivity.java**.

```
package com.example.tictactoe;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private TableLayout tableLayout;

    private Button[][] buttons = new Button[3][3];

    private Game game;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        tableLayout = findViewById(R.id.tableLayout);

        game = new Game();
        buildGameField(); // создание игрового поля
        game.start(); // будет реализован позже
    }

    private void buildGameField() {
        Square[][] field = game.getField();
        for (int i = 0, lenI = field.length; i < lenI; i++) {
            TableRow row = new TableRow(this); // создание строки таблицы
            for (int j = 0, lenJ = field[i].length; j < lenJ; j++) {
                Button button = new Button(this);
                buttons[i][j] = button;
                button.setOnClickListener(new Listener(i, j)); // установка слушателя,
реагирующего на клик по кнопке
                row.addView(button, new
                    TableRow.LayoutParams(TableRow.LayoutParams.WRAP_CONTENT,
                                         TableRow.LayoutParams.WRAP_CONTENT)); // добавление кнопки в
строку таблицы
                button.setWidth(350);
                button.setHeight(350);
            }
            tableLayout.addView(row,
                               new
                    TableLayout.LayoutParams(TableLayout.LayoutParams.WRAP_CONTENT,
                                         TableLayout.LayoutParams.WRAP_CONTENT)); // добавление
строки в таблицу
    }
}
```

```
        }
    }

public class Listener implements View.OnClickListener {
    private int x;
    private int y;

    Listener(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void onClick(View view) {
        Button button = (Button) view;
        Game g = game;
        Player player = g.getCurrentActivePlayer();
        if (makeTurn(x, y)) {
            button.setText(player.getName());
        }
        Player winner = g.checkWinner();
        if (winner != null) {
            gameOver(winner);
        }
        if (g.isFieldFilled()) { // в случае, если поле заполнено
            gameOver();
        }
    }
}

private void gameOver(Player player) {
    CharSequence text = "Player \"" + player.getName() + "\" won!";
    Toast.makeText(this, text, Toast.LENGTH_SHORT).show();
    game.reset();
    refresh();
}

private void gameOver() {
    CharSequence text = "Draw";
    Toast.makeText(this, text, Toast.LENGTH_SHORT).show();
    game.reset();
    refresh();
}

private boolean makeTurn(int x, int y) {
    return game.makeTurn(x, y);
}

private void refresh() {
    Square[][] field = game.getField();

    for (int i = 0, len = field.length; i < len; i++) {
        for (int j = 0, len2 = field[i].length; j < len2; j++) {
            if (field[i][j].getPlayer() == null) {
                buttons[i][j].setText("");
            } else {
                buttons[i][j].setText(field[i][j].getPlayer().getName());
            }
        }
    }
}
```

```
    }  
}
```

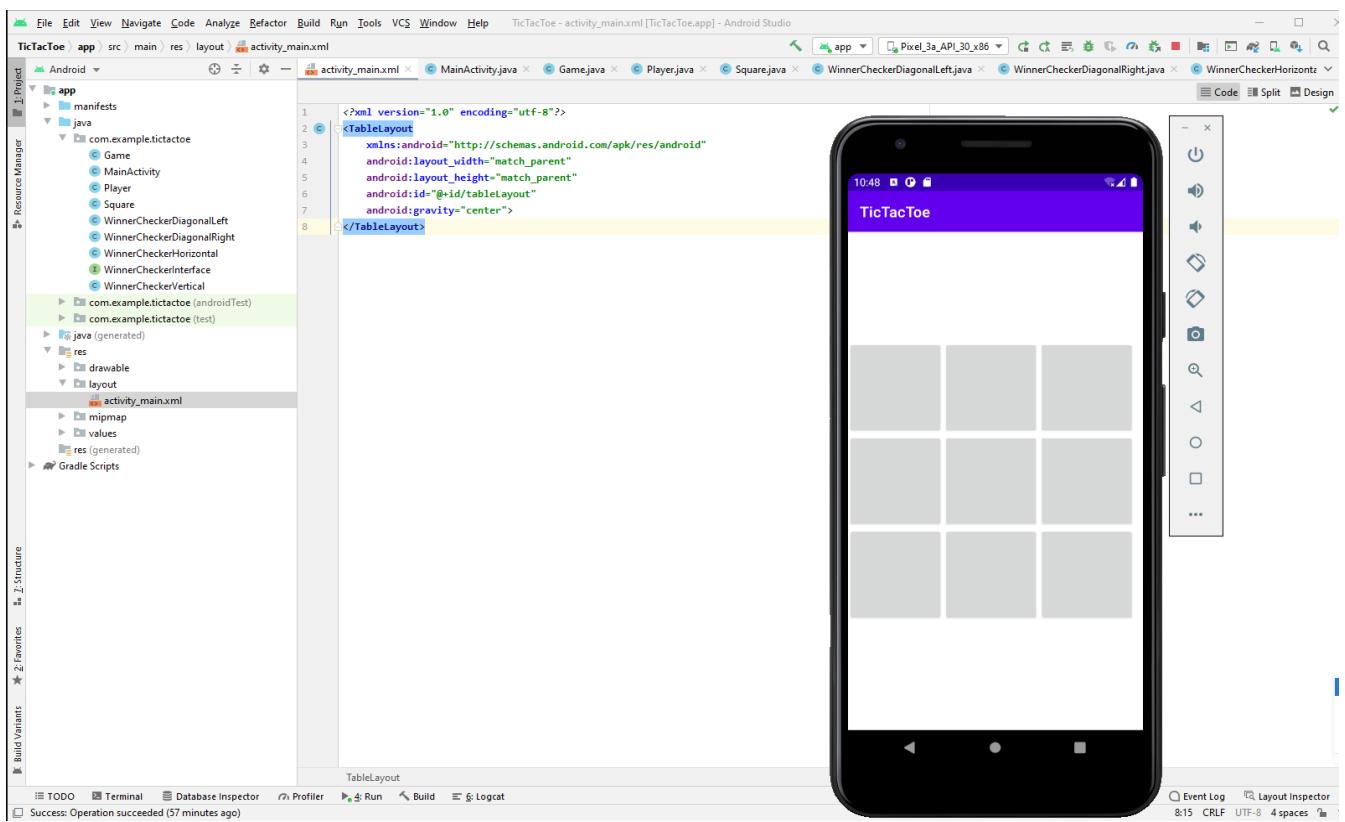


Рисунок 1 – Готовое приложение игры «Крестики-нолики»

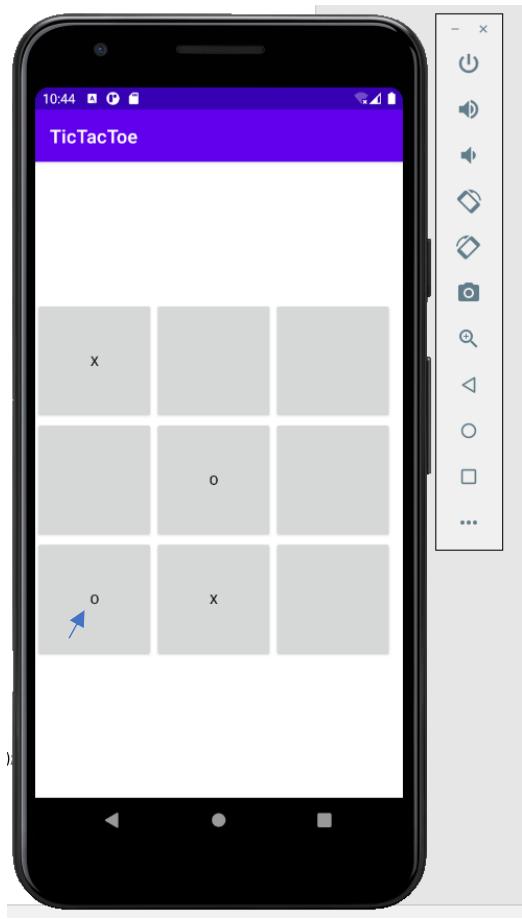


Рисунок 2 – Тестирование игры «Крестики-нолики» в режиме ввода “нолика”

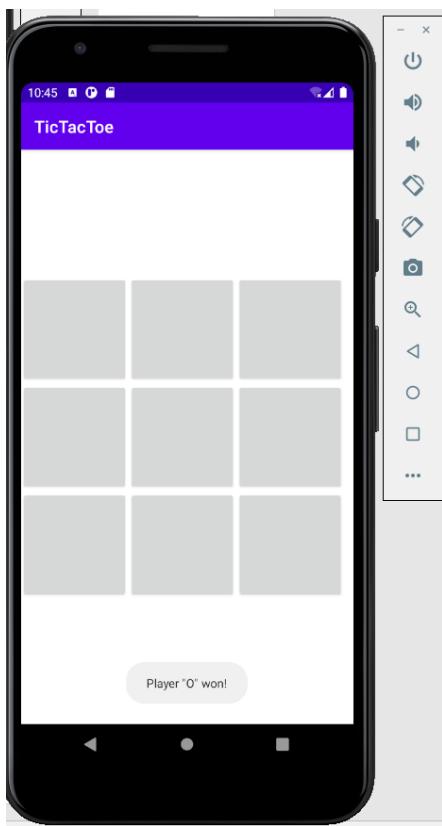


Рисунок 3 – Тестирование игры «Крестики-нолики» в случае победы “нолика”

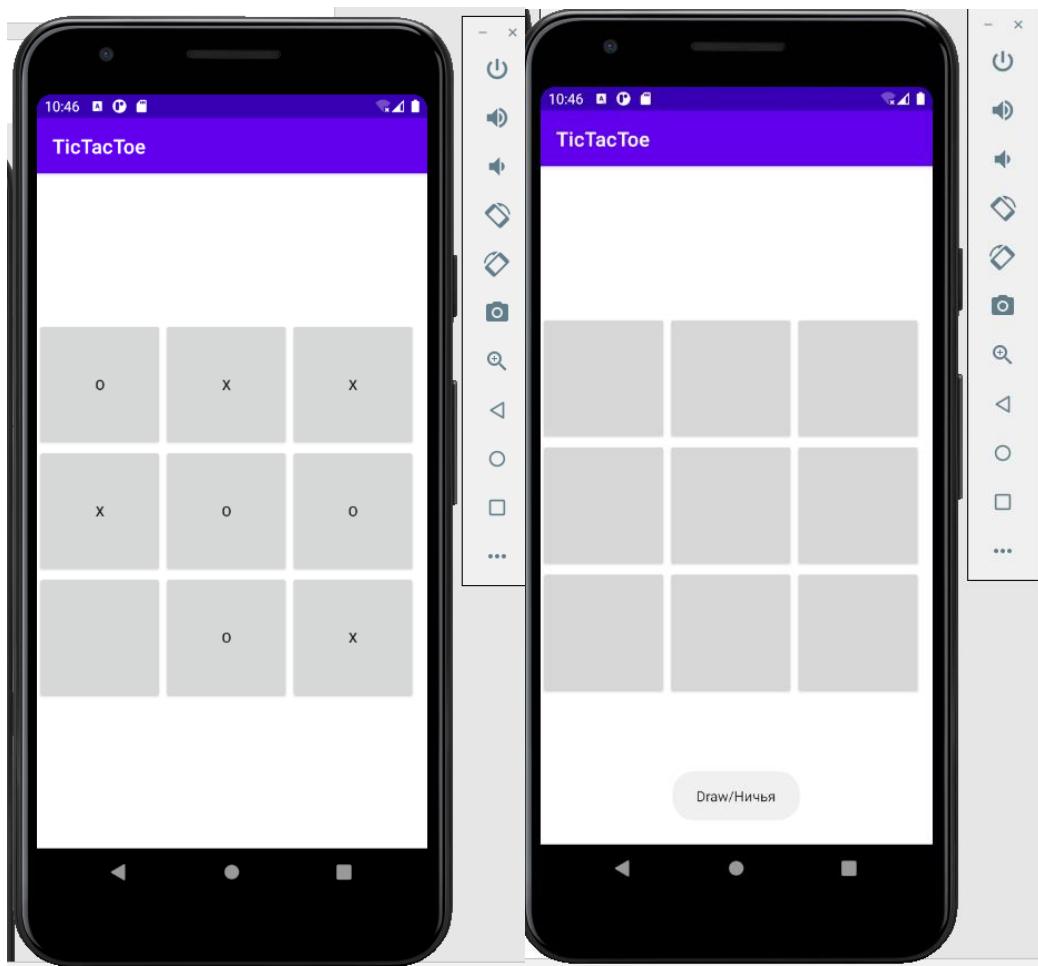


Рисунок 4 – Тестирование игры «Крестики-нолики» в случае “ничья”

Задание : Изучить программу игры крестики-нолики и увеличить размер поля игры до 4*4.

Контрольные вопросы:

- 1) Какие среды разработки существуют для создания приложений под ОС Android?
- 2) Способы запуска разработанных приложений?
- 3) Способы тестирования разработанных приложений?
- 4) Какие еще существуют мобильные ОС?