МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

ТЕХНОЛОГИИ КОМПОНЕНТНОГО ПРОГРАММИРОВАНИЯ

Методические рекомендации к лабораторным работам для студентов специальности 1-40 80 02 «Системный анализ, управление и обработка информации» очной и заочной форм обучения

Могилев 2020

УДК 004.4
ББК 32.97
Т38

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «02» сентября 2020 г., протокол № 1

Составитель ст. преподаватель Н. В. Выговская

Рецензент И. В. Лесковец

Методические рекомендации предназначены для студентов специальности 1-40 80 02 «Системный анализ, управление и обработка информации» дневной и заочной форм обучения по дисциплине «Технологии компонентного программирования».

Учебно-методическое издание

ТЕХНОЛОГИИ КОМПОНЕНТНОГО ПРОГРАММИРОВАНИЯ

Ответственный за выпуск	А. И. Якимов
Корректор	Е. А. Галковская
Компьютерная верстка	Е. В. Ковалевская

Подписано в печать 9.11.2020 . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс. Печать трафаретная. Усл. печ. л. 2,79 . Уч.-изд. л. 2,88 . Тираж 16 экз. Заказ № 606.

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, Могилев.

Содержание

Введение

Цель изучения дисциплины – приобретение навыков в использовании современных средств разработки программ на профессиональном уровне и создании приложений на основе современных платформ .NET.

Цель лабораторных занятий по дисциплине «Технологии компонентного программирования» заключается в закреплении студентами практических навыков создания и сопровождения программных систем современных ЭВМ на платформах .NET с компонентным подходом на языке C#.

Дисциплина «Технологии компонентного программирования» является неотъемлемой частью современных знаний и связана с такими дисциплинами, как «Базы данных» и «Объектно-ориентированное программирование».

Выполнение заданий позволит студентам выработать практические навыки разработки больших приложений на языке C# с использованием баз данных, географических карт, с красивыми и удобными интерфейсами. Особенностью разработки приведенного приложения MVC является использование модульных тестов. Полученные при изучении дисциплины знания и навыки будут востребованы при практической разработке серверных и клиентских приложений, работающих в сети Интернет и автономно.

В процессе выполнения лабораторных работ студенты ознакомятся с теоретическим материалом, методами решения задач, выполнят индивидуальное задание и оформят отчет.

Отчет должен содержать название и цель лабораторной работы, структуру и листинг электронных документов, анализ полученных результатов и выводы.

1 Технология разработки корпоративного приложения

1.1 WPF. Разработка корпоративного приложения. Элементы интерфейса

Цель работы: изучить методику разработки интерфейса корпоративного приложения.

Для разработки приложения необходимо создать WPF-проект.

В XAML-документе имеется один элемент верхнего уровня <Window>. Дескриптор </Window> завершает весь документ. В XAML-документе приведено имя класса MainWindow:

```
x:Class="WpfApplProject.MainWindow
```

Два пространства имен:

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

И три свойства:

```
Title="MainWindow" Height="350" Width="525"
```

Каждый атрибут соответствует определенному свойству класса Window. Приведенные атрибуты предписывают WPF создать окно с надписью MainWindow и размером 350 × 525 единиц. При компиляции и запуске проекта приложения на дисплей выводится окно, приведенное на рисунке 1.

Рисунок 1 – MainWindow проекта

Когда выполняется компиляция приложения, XAML-файл, который определяет пользовательский интерфейс (MainWindow.xaml), транслируется в объявление типа CLR, которое объединяется с логикой приложения из файла класса отдельного кода (MainWindow.xaml.cs).

Метод InitializeComponent() генерируется во время компиляции приложения и в исходном коде не присутствует.

Для программного управления элементами управления, описанными в XAML-документе, необходимо задать XAML атрибут Name. Так для задания имени элементу Grid необходимо записать следующую разметку:

```
<Grid Name="grid">

</Grid>
```

Страницы приложения можно размещать внутри окон и внутри других страниц. В WPF при создании страничных приложений контейнером наивысшего уровня могут быть следующие объекты:

– NavigationWindow, который представляет собой несколько видоизмененную версию класса Window;

– Frame, находящийся внутри другого окна или другой страницы;

– Frame, обслуживаемый непосредственно в Internet explorer.

Для вставки страницы внутрь окна будем использовать класс Frame (рисунок 2).

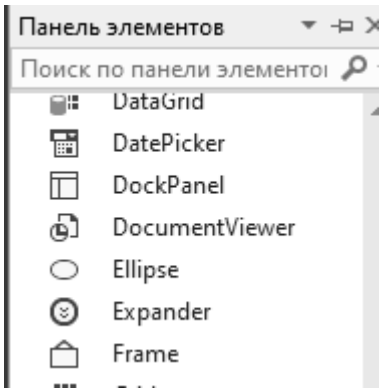

Рисунок 2 – Выбор класса Frame в панели инструментов

В XAML-документ проекта будет добавлена следующая строка:

```
<Frame Height="100" HorizontalAlignment="Left"
Name="frame1" VerticalAlignment="Top" Width="200" />
```

С учетом того, что создается страничное приложение, размеры фрейма не нужно фиксировать, поэтому изменим описание свойств фрейма:

<Frame Name="frame1" Margin="3" />

В результате фрейм заполнит всё окно (рисунок 3).

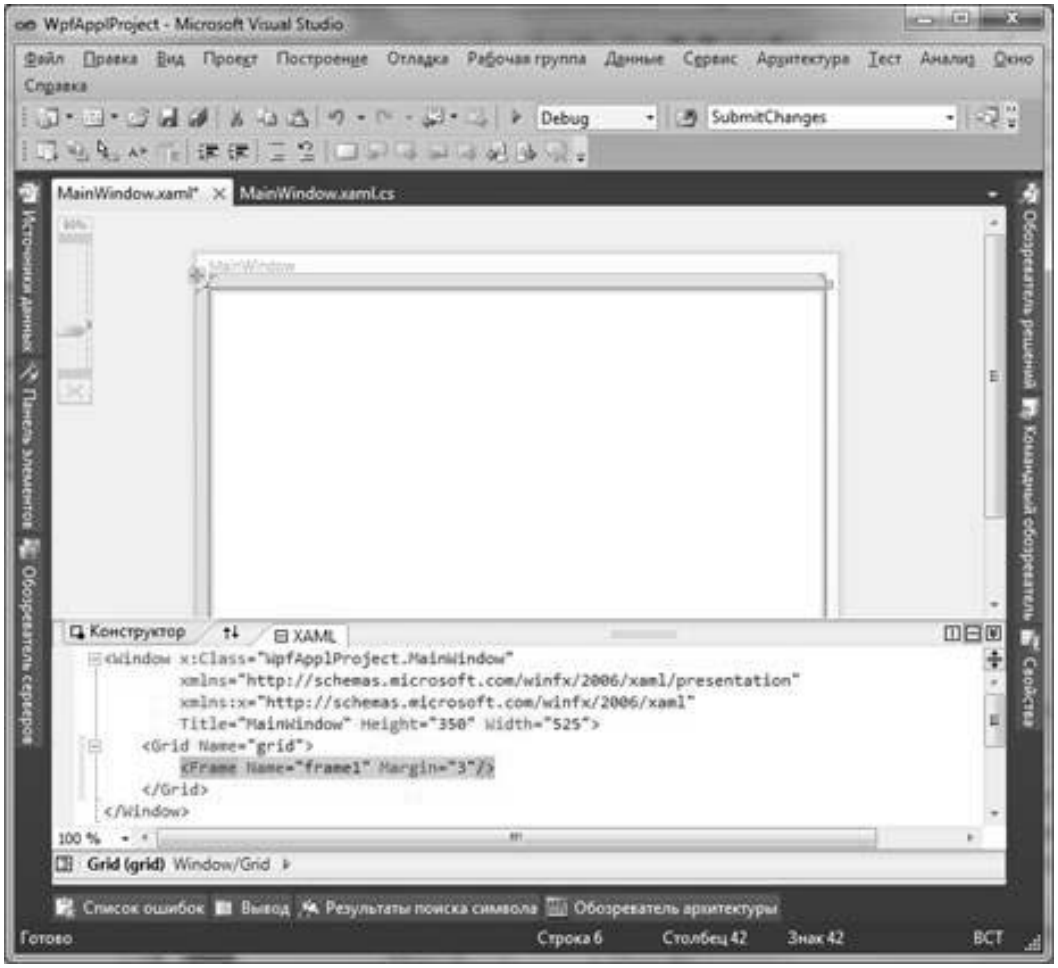

Рисунок 3 – Фрейм, заполняющий всё окно

Созданный фрейм необходим для размещения в нем страницы WPF-приложения. Класс Page допускает наличие только одного вложенного элемента. Он не является элементом управления содержимым и наследуется от класса FrameworkElement. Класс Page имеет небольшой набор дополнительных свойств, которые позволяют настраивать его внешний вид, взаимодействовать с контейнером и использовать навигацию. Для перехода на другую страницу необходимо использовать навигацию.

Добавьте в проект начальную страницу. Для этого в Обозревателе решений щелкните правой кнопкой мыши на проекте WpfApplProject. В контекстном меню выберите пункт Добавить, а в раскрывающимся меню – пункт Страница (рисунок 4).

В окне Добавления нового элемента необходимо выбрать шаблон «Страница (WPF)» (1) и задать имя страницы PageMain .

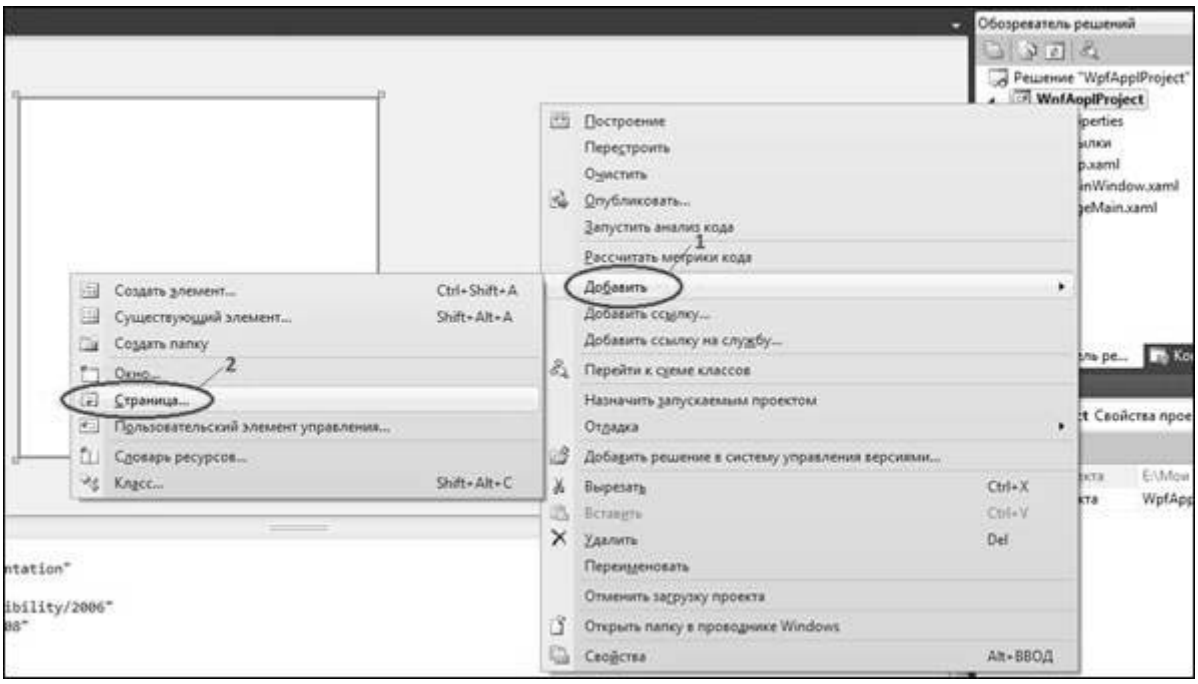

Рисунок 4 – Меню создания страницы

В дизайнере проекта сгенерируется страница PageMain.xaml (рисунок 5).

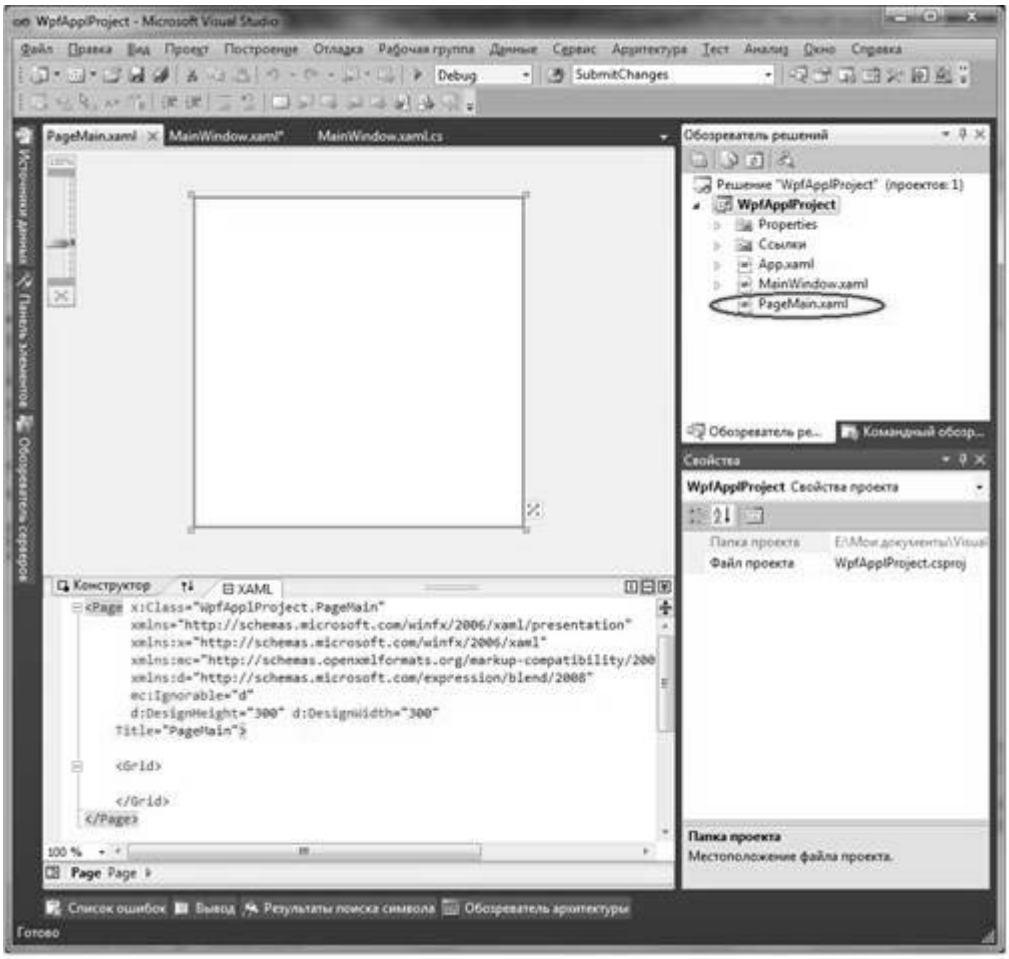

Рисунок 5 – Дизайнер страницы PageMain.xaml

В сгенерированной странице в качестве контейнера верхнего уровня используется Grid. Замените Grid на контейнер StackPanel.

Главная страница приложения в дизайнере представлена на рисунке 6.

Рисунок 6 – Главная страница WPF-приложения в дизайнере

Измените свойство Title окна класса Window в соответствии с вариантом лабораторной работы (рисунок 7).

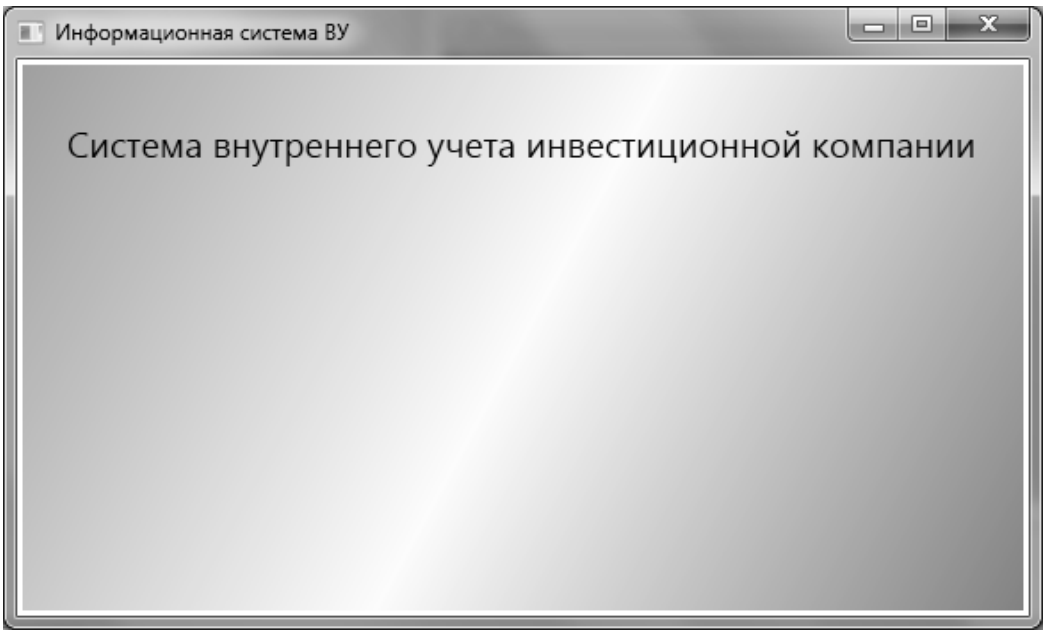

Рисунок 7 – Главная страница WPF-приложения

Основная страница должна обеспечивать переход на другие страницы, обеспечивающие интерфейс для отдельных функций и выход из системы. Для перехода на другие страницы используйте гиперссылки. Гиперссылки позволяют пользователю перемещаться с одной страницы на другую. Элемент

гиперссылки, соответствующий объекту класса Hyperlink, определяется следующей строкой:

<Hyperlink >Текст Гиперссылки</Hyperlink>

Класс Hyperlink имеет свойство NavigateUri. Данное свойство определяет на какую страницу будет переходить приложение при щелчке на соответствующей гиперссылке. Например, NavigateUri = Page2.xaml?

В WPF гиперссылки являются не отдельными, а внутристроковыми потоковыми элементами, которые должны размещаться внутри другого поддерживающего их элемента. Это можно сделать, например, в элементе TextBlock, который для гиперссылки является контейнером.

На первой странице создайте гиперссылки для перехода на страницы приложения в соответствии с заданием (рисунок 8).

Рекомендация: используйте StackPanel для размещения в нём Textblock.

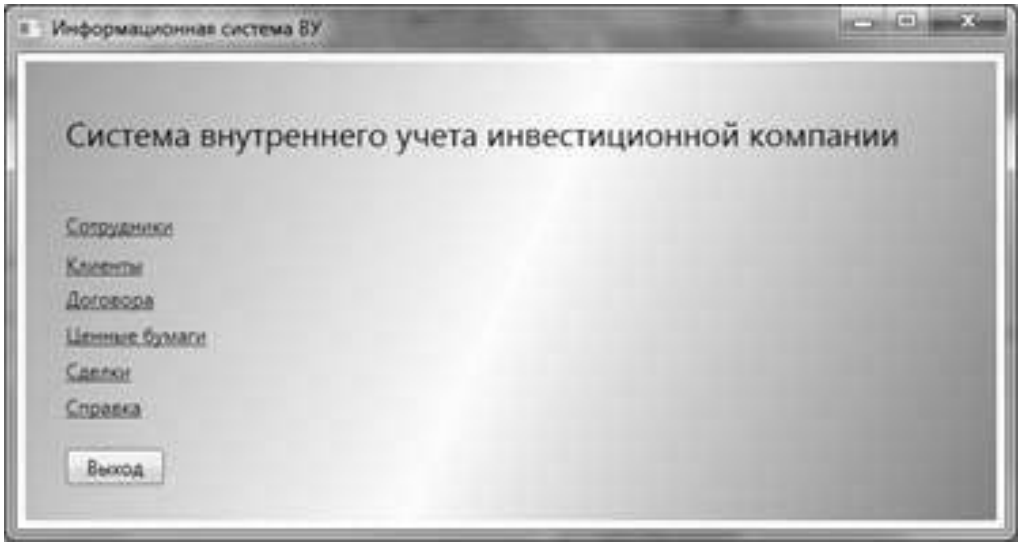

Рисунок 8 – Начальная страница WPF-приложения с гиперссылками

Создание основного меню. Создайте основное меню с помощью класса Menu, который представляет Windows элементы управления меню, позволяющие иерархически организовать элементы, связанные с командами и обработчиками событий. Меню формируют из объектов MenuItem (имя пункта меню) и Separator (разделитель). Класс MenuItem имеет свойство Header, которое определяет текст элемента меню. Данный класс может хранить коллекцию объектов MenuItem, которая соответствует подпунктам меню. Класс Separator отображает горизонтальную линию, разделяющую пункты меню.

Добавьте в StackPanel страницы приложения XAML-описание меню, которое на верхнем уровне будет содержать, например, два пункта – Действие и Отчет. Пункт Действие включает подпункты: Отменить, Создать, Редактировать, Сохранить, Найти и Удалить. Между пунктами Отменить, Создать и пунктами Найти, Удалить добавьте разделительные линии (рисунок 9).

```
        <Menu FontSize="14" FontFamily="Times New Roman">
          <MenuItem Header="Действие" >
          <MenuItem Header="Отменить"></MenuItem>
          <Separator></Separator>
          <MenuItem Header="Создать"></MenuItem>
          <MenuItem Header="Редактировать"></MenuItem>
          <MenuItem Header="Сохранить"></MenuItem>
          <Separator></Separator>
          <MenuItem Header="Удалить"></MenuItem>
          </MenuItem>
          <MenuItem Header="Отчет"></MenuItem>
</Menu>
```

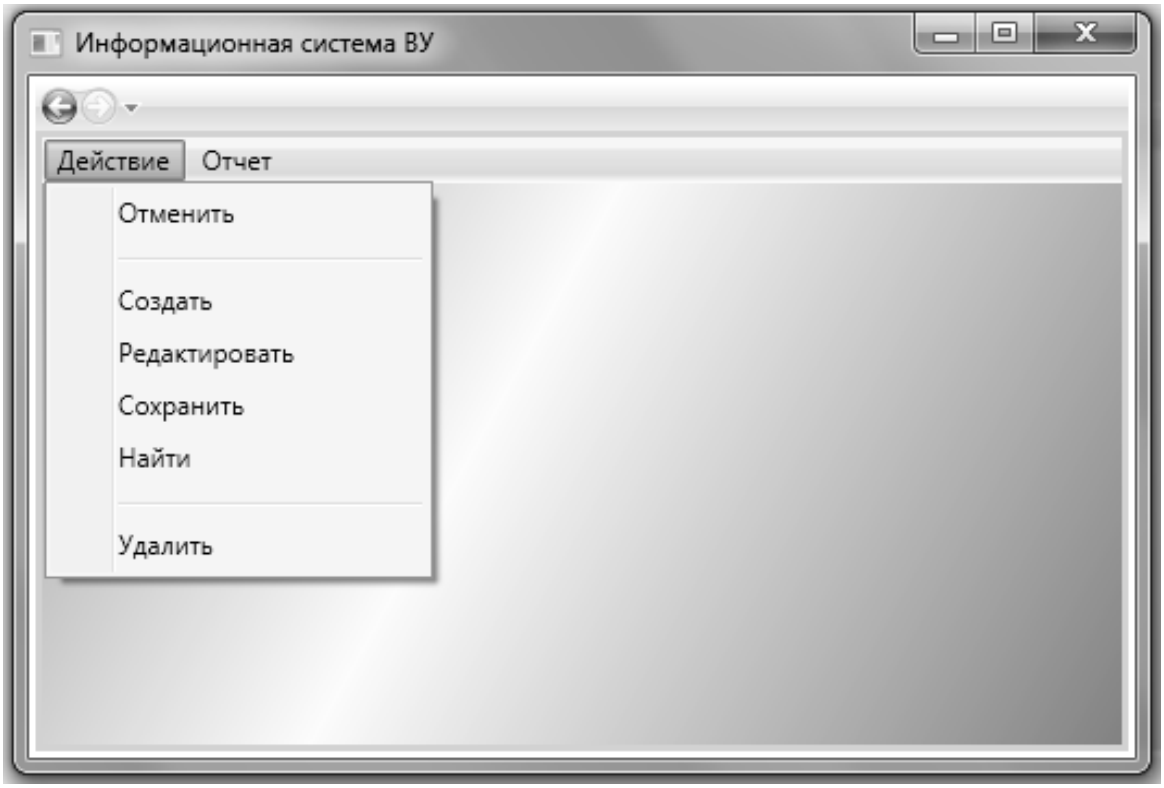

Рисунок 9 – Главное меню страницы PageEmployee

Создание панели инструментов. Панель инструментов представляет специализированный контейнер для хранения коллекции элементов, обычно кнопок. Расположите в панели инструментов кнопки, функциональное назначение которых будет соответствовать подпунктам меню Действие, т. е. Отменить, Создать, Редактировать, Сохранить, Найти и Удалить.

На лицевой стороне кнопок поместите графическое изображение соответствующего действия. Для этого добавьте в файл проекта папку Images и в неё включите графические объекты, которые можно найти в библиотеке графических объектов.

После добавления графических файлов в проект они будут отображены в обозревателе решений (рисунок 10).

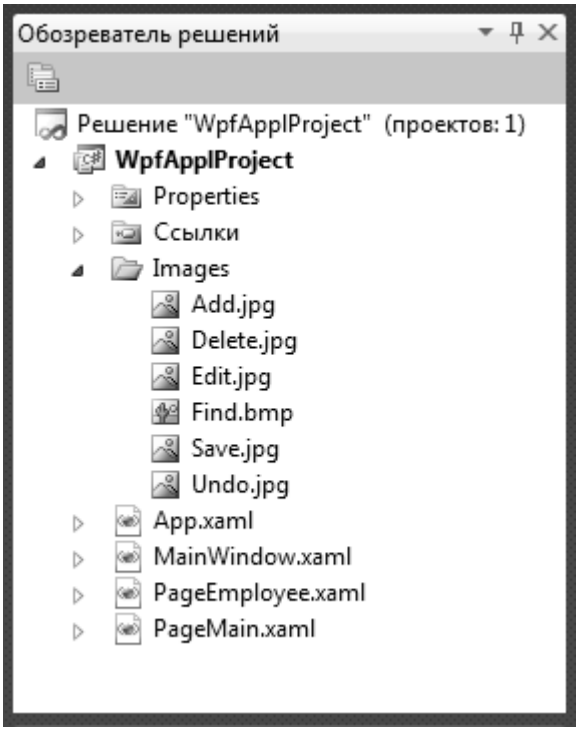

Рисунок 10 – Включение в проект папки Images с файлами рисунков

Для каждой кнопки задайте свойство Name – имя объекта в программе и свойство ToolTip с текстом всплывающей подсказки при наведении указателя мыши на кнопку (рисунок 11).

Свойство Margin определяет внешние отступы для кнопки. Задание графического объекта для кнопки осуществляется определением для объекта Image источника Source, который должен соответствовать полному пути к графическому файлу.

```
<ToolBar Name="ToolBar1" Margin="3">
        <Button Name="btnUndo" ToolTip="Отменить редактирование/создание"
Margin="5,2,5,2" Width="30" Height="30">
                <Button.OpacityMask>
                        <ImageBrush Stretch="Uniform" ImageSource="icons/undo.png"/>
                </Button.OpacityMask>
                <Button.Background>
                        <ImageBrush Stretch="Uniform" ImageSource="icons/undo.png"/>
                </Button.Background>
…
        </Button>
</ToolBar>
```

Проектирование интерфейсных элементов. При проектировании интерфейсных элементов для страницы приложения используются элементы контроля ListBox, ListView, TextBox, TextBlock, ComboBox, DataGrid и других.

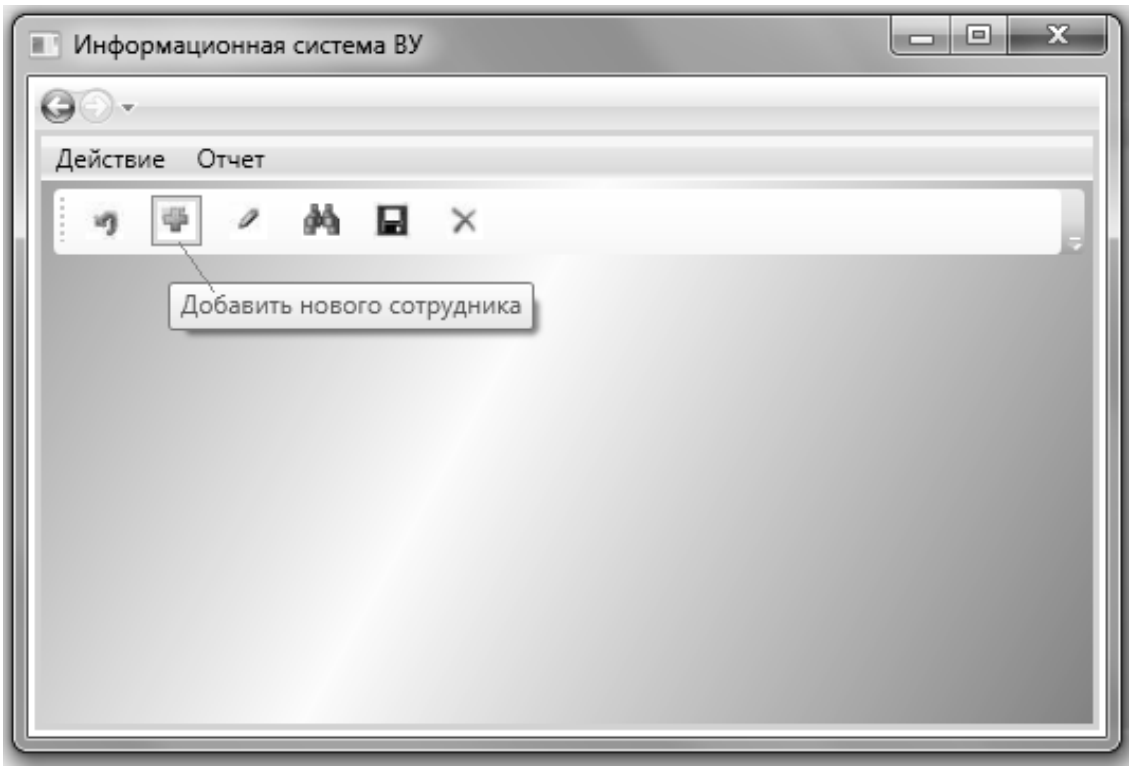

Рисунок 11 – Страница PageEmployee с панелью инструментов

Класс DataGrid представляет элемент управления, отображающий данные в настраиваемой сетке строк и столбцов. По умолчанию DataGrid (рисунки 12, 13) автоматически создает столбцы на основе источника данных. При этом генерируются следующие типы столбцов:

– DataGridTextColomn – для отображения в ячейках столбцов текстового содержимого;
– DataGridCheckBoxColomn – для отображения в ячейках столбцов логических данных;
– DataGridComboBoxColomn – для отображения в ячейках столбцов данных, когда имеется набор элементов для выбора;
– DataGridHyperlinkColomn – для отображения в ячейках столбцов элементов Uri.

Если разработчика не устраивает автоматическая генерация столбцов DataGrid, можно её отключить. Для этого свойству AutoGenerateColumns необходимо присвоить значение false. Далее можно создать собственный набор столбцов (Columns), используя существующие типы столбцов или создать новый тип столбца с помощью шаблона DataGridTemplateColumn.

DataGrid поддерживает множество сценариев настройки отображения данных (таблица 1).

Таблица 1 – Способы настройки отображения данных

Способ	Подход
Переменные цвета фона	Задайте для свойства *AlternationIndex* значение 2 или больше, а затем назначьте объект *Brush* свойствам *RowBackground* и *AlternatingRowBackground*.
Определение поведения при выборе ячейки и строки	Установите свойства *SelectionMode* и *SelectionUnit*.
Настройка внешнего вида заголовков, ячеек и строк	Примените новый *Style* к свойствам *ColumnHeaderStyle, RowHeaderStyle, CellStyle* или *RowStyle*.
Доступ к выбранным элементам	Проверьте свойство *SelectedCells*, чтобы получить выделенные ячейки, и свойство *SelectedItems*, чтобы получить выбранные строки.
Настройка взаимодействия с пользователем	Установите свойства *CanUserAddRows, CanUserDeleteRows, CanUserReorderColumns, CanUserResizeColumns, CanUserResizeRows* и *CanUserSortColumns*.
Отмена или изменение автоматически созданных столбцов	Обработать событие *AutoGeneratingColumn*.
Заморозка столбца	Задайте для свойства *FrozenColumnCount* значение 1 и переместите столбец в крайнюю левую позицию, задав для свойства *DisplayIndex* значение 0.
В качестве источника данных используются данные XML	Привяжите *ItemsSource* в *DataGrid* к запросу *XPath*, представляющему коллекцию элементов. Создайте каждый столбец в *DataGrid*. Свяжите каждый столбец, указав *XPath* для привязки к запросу, который получает свойство из источника элемента.

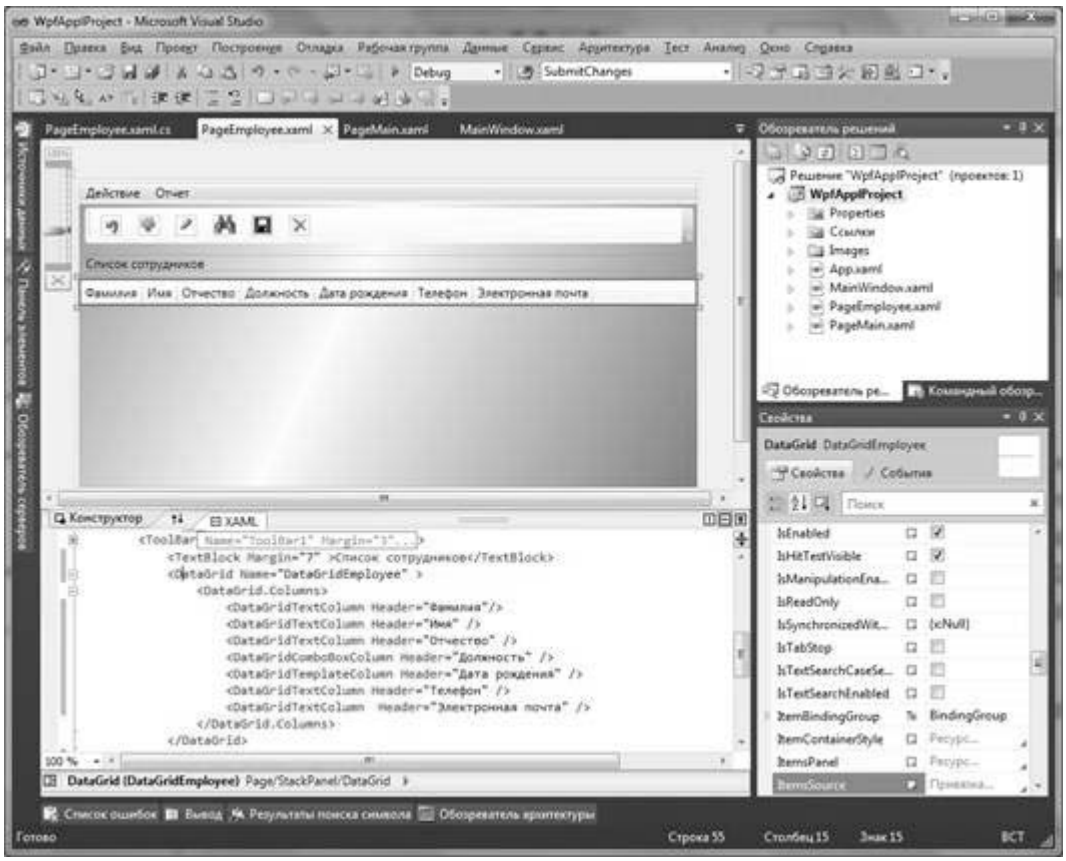

Рисунок 12 – Страница PageEmployee дизайнера с элементом DataGrid

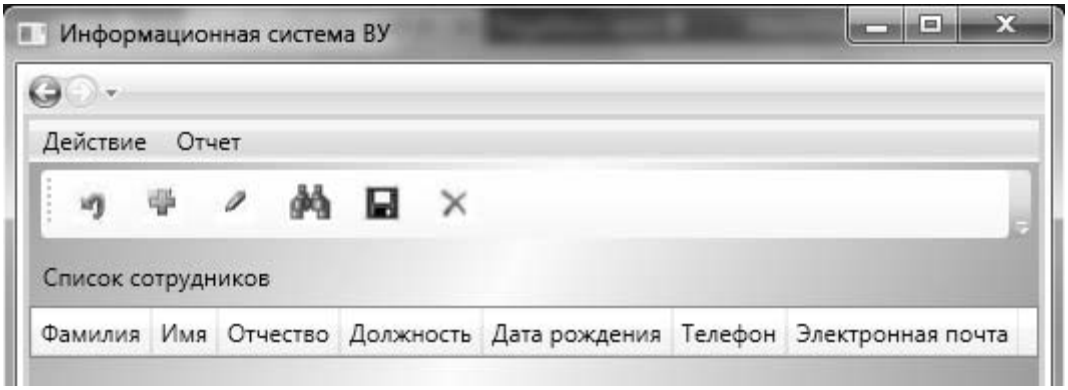

Рисунок 13 – Страница PageEmployee с элементом DataGrid

Разработка бизнес-логики приложения

Для реализации функциональности приложения необходимо для страницы приложения установить механизм запуска задач при выборе соответствующих пунктов меню и нажатии на кнопки панели инструментов. Технология WPF предлагает модель команд для выполнения такой привязки.

Модель команд обеспечивает делегирование событий определенным командам и управление доступностью элементов управления в зависимости от состояния соответствующей команды. В WPF команда представляет собой задачу приложения и механизм слежения за тем, когда она может быть

выполнена. В то же время сама команда не содержит конкретного кода выполнения задачи. Одна и та же команда может быть привязана к одному или нескольким интерфейсным элементам приложения. Инициируют команду источники, которые могут быть различными элементами управления, например, пункты меню MenuItem или кнопки Button. Целевым объектом команды является элемент, для которого предназначена эта команда.

Классы, реализующие команды, должны поддерживать интерфейс ICommand. В этом интерфейсе определены два метода Execute, CanExecuteи событие CanExecuteChanged.

```
public interface ICommand1
{
        void Execute(object par);
        bool CanExecute(object par);
        event EventHandler CanExecuteChanged;
}
```

Метод Execute может содержать код, реализующий бизнес-логику приложения. Метод CanExecute возвращает информацию о состоянии команды. Событие CanExecuteChanged вызывается при изменении состояния команды.

В WPF имеется библиотека базовых команд. Команды доступны через статические свойства следующих статических классов:

– ApplicationCommands;
– NavigationCommands;
– EditingCommands;
– MediaCommands.

Для создания пользовательских команд целесообразно использовать класс RoutedCommand, который имеет реализацию интерфейса ICommand.

Для разработки пользовательских команд добавьте в проект папку Commands и в ней создайте класс DataCommands.

```
public static class DataCommands
{
    public static RoutedCommand Delete { get; set; }
    public static RoutedCommand Edit { get; set; }
    public static RoutedCommand Add { get; set; }
    public static RoutedCommand Undo { get; set; }
    static DataCommands()
    {
            InputGestureCollection inputs = new InputGestureCollection();
            inputs.Add(new KeyGesture(Key.E, ModifierKeys.Control, "Ctrl+E"));
            Edit = new RoutedCommand("Edit", typeof(DataCommands), inputs);
            inputs = new InputGestureCollection();
            inputs.Add(new KeyGesture(Key.D, ModifierKeys.Control, "Ctrl+D"));
            Delete = new RoutedCommand("Delete", typeof(DataCommands), inputs);
            inputs = new InputGestureCollection();
            inputs.Add(new KeyGesture(Key.N, ModifierKeys.Control, "Ctrl+N"));
```

```
            Add = new RoutedCommand("Add", typeof(DataCommands), inputs);
            inputs = new InputGestureCollection();
            inputs.Add(new KeyGesture(Key.Z, ModifierKeys.Control, "Ctrl+Z"));
            Undo = new RoutedCommand("Undo", typeof(DataCommands), inputs);
    }
}
```

В классе DataCommands объявлены два свойства Delete и Edit типа RoutedCommand. Класс RoutedCommand определяет команду, реализующую ICommand. В конструкторе данного класса определяется объект inputs типа InputGestureCollection. Класс InputGestureCollection представляет упорядоченную коллекцию объектов InputGesture, которые позволяют с помощью класса KeyGesture задать комбинацию клавиш для вызова команды.

Для использования страницей приложения пользовательских команд в XAML-документе необходимо добавить пространство имен, где расположен класс DataCommands. Данному пространству имен присвоим ссылку command.

```
xmlns:command="clr-namespace:WpfApplProject.Commands"
```

Теперь для страницы приложения сформируйте коллекцию объектов CommandBinding, которая осуществляет привязку команд для данного элемента и объявляет связь между командой, ее событиями и обработчиками.

```
<Page.CommandBindings>
        <CommandBinding                                                Command="Undo"
  Executed="UndoCommandBinding_Executed"
  CanExecute="UndoCommandBinding_CanExecute" />
        <CommandBinding      Command="{x:Static      command:DataCommands.Edit}"
  CanExecute="EditCommandBinding_CanExecute"
        Executed="DeleteCommandBinding_Executed" />
</Page.CommandBindings>
```

Для класса CommandBinding свойство Command определяет ссылку на соответствующую команду, а свойства Executed и CanExecute задают обработчики событий при выполнении команды.

На странице приложения используются следующие команды: Отменить, Создать, Редактировать, Поиск, Сохранить и Удалить. Команды могут быть доступны или недоступны пользователю при работе приложения. Это проверяет метод CanExecute при генерации события CanExecuteChanged, которое вызывается при изменении состояния команды. Доступность команд определяется состоянием, в котором находится приложение. В то же время выполнение какой-либо команды переводит, как правило, приложение в какое-либо другое состояние. Для проектируемого приложения можно определить следующие состояния:

– первоначальная загрузка страницы;
– просмотр данных;
– редактирование данных;
– создание новой записи.

Таблица 2 – Доступность команд в различных состояниях приложения

Состояние	Доступность команд					
	Сохранить	Отменить	Создать	Поиск	Редактировать	Удалить
1	false	false	true	true	true	true
2	false	false	true	true	true	true
3	true	true	false	false	false	false
4	true	true	false	false	false	false

В код программы класса страницы приложения введите логическое поле isDirty для управления доступностью команд.

```
private bool isDirty = true;
```

В код класса добавьте обработчики (реализация метода Executed), определяющие бизнес-логику приложения. На данном этапе проектирования системы обработчики будут содержать только вывод сообщений о вызове команды и изменение поля isDirty. Код обработчика команды Удалить приведен ниже.

```
private void UndoCommandBinding_Executed(object sender, ExecutedRoutedEventArgs e)
{
    Context = new TitleEmployeeEntities();
    Context.Employe.Load();
    DataGridEmployee.ItemsSource = Context.Employe.Local;
    DataGridEmployee.IsReadOnly = true;
}
```

В дальнейшем в обработчики необходима будет добавить код для обеспечения требуемой функциональности.

В коде класса необходимо добавить обработчики (реализация метода CanExecute), которые управляют доступностью команд.

```
private void EditCommandBinding_CanExecute(object sender, CanExecuteRoutedEventArgs e)
{
	e.CanExecute = true;
	}
	private void SaveCommandBinding_CanExecute(object sender,
CanExecuteRoutedEventArgs e)
	{
	e.CanExecute = Context.ChangeTracker.HasChanges();
	}
```

Необходимо модифицировать XAML-документ в части задания свойства Command при описании пунктов меню и панели инструментов для привязки команд (рисунок 14).

При описании меню (Menu) XAML-документ модифицирован следующим образом:

```
	<Menu FontSize="14" FontFamily="Times New Roman">
	<MenuItem Header="Действие" ></MenuItem>
	<MenuItem Header="Отменить" Command="Undo" ></MenuItem>
		<MenuItem Header="Редактировать" Command="{x:Static
	  command:DataCommands.Edit}"></MenuItem>
	…
</Menu>
```

Соответствующие изменения XAML-документа необходимо провести и для панели инструментов ToolBar.

```
	<ToolBar Name="ToolBar1" Margin="3">
	<Button Name="btnUndo" ToolTip="Отменить редактирование/создание"
Command="Undo" Margin="5,2,5,2" Width="30" Height="30">
		<Button.OpacityMask>
			<ImageBrush Stretch="Uniform" ImageSource="icons/undo.png"/>
		</Button.OpacityMask>
		<Button.Background>
			<ImageBrush Stretch="Uniform" ImageSource="icons/undo.png"/>
		</Button.Background>
	</Button>
	...
	</ToolBar>
```

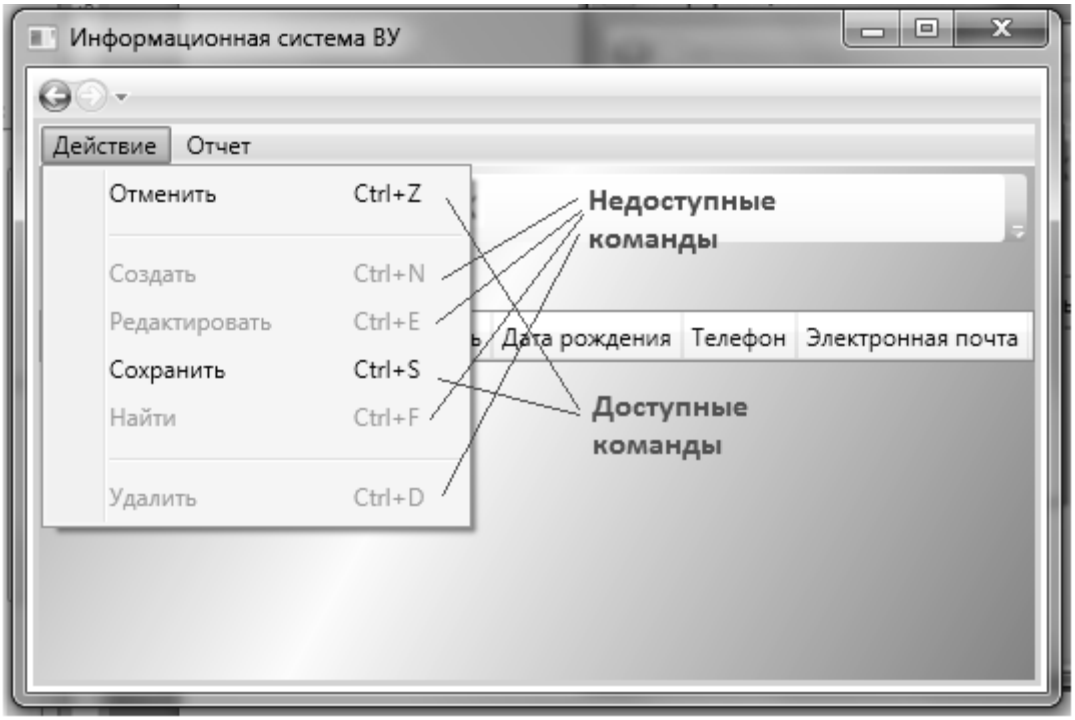

Рисунок 14 – Привязка команд

1.2 WPF. Разработка корпоративного приложения. Разработка БД

Цель работы: изучить методику разработки модели данных корпоративного приложения.

Для создания EDM-модели данных необходимо добавить в проект новый элемент – модель ADO.NET EDM (рисунок 15), присвоив файлу модели имя в соответствии с заданием, в рассматриваемом примере модель имеет имя TitleEmployee.edmx.

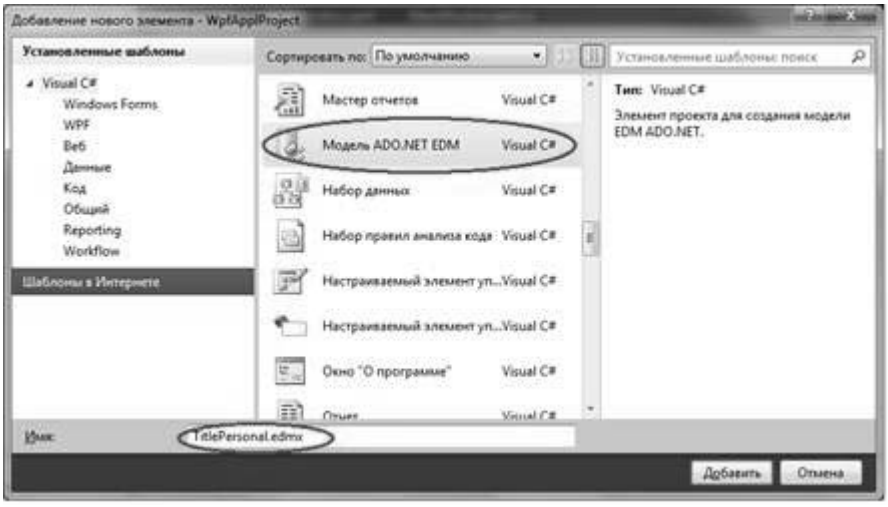

Рисунок 15 – Добавление в проект модели EDM

В мастере создания EDM-модели выберем опцию «Создать из базы данных» (рисунок 16). Для создания соединения с базой данных в окне «Выбор подключения к данным» укажем соединение с базой данных (имя_сервера.имя_базы данных, в рассматриваемом примере – алексей-пк.TitlePerson) и зададим имя модели данных - TitlePersonEntities (рисунок 17).

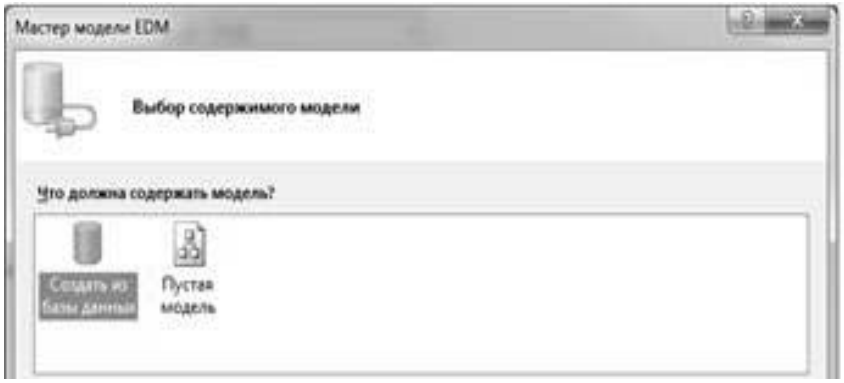

Рисунок 16 – Создание модели EDM из базы данных

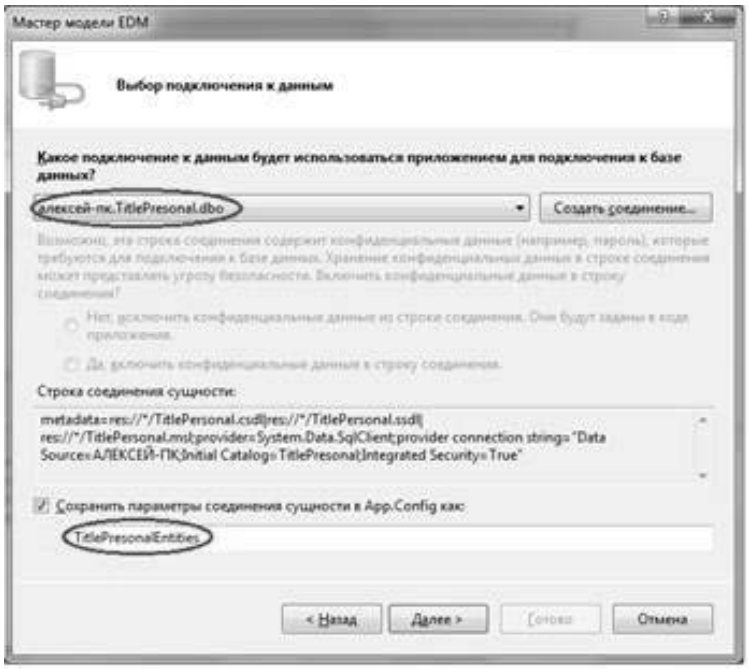

Рисунок 17 – Создание соединения с базой данных

В окне «Выбор объектов базы данных» отметим необходимые для приложения таблицы (в нашем случае это таблицы Employee и Title) и флаг формирования объектов в единственном или множественном числе (рисунок 18).

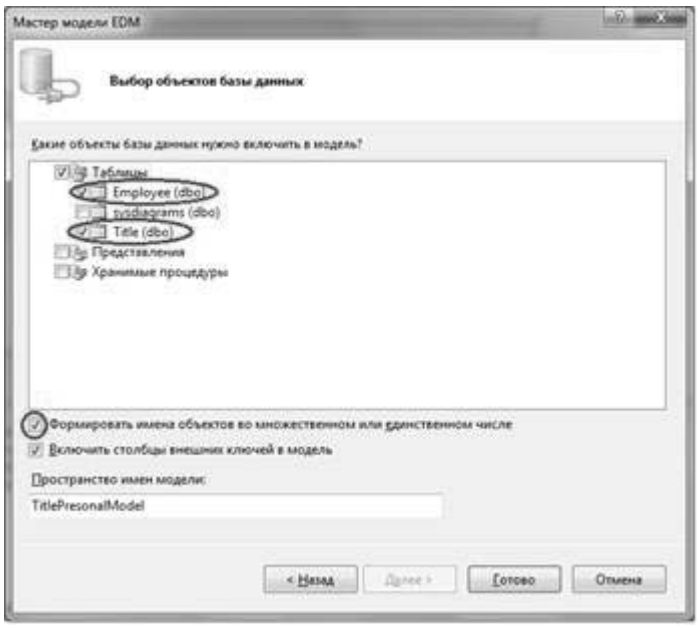

Рисунок 18 – Выбор таблиц базы данных

При завершении работы мастера создания EDM-модели в проект будет добавлен файл (в рассматриваемом примере TitleEmployee.edmx – рисунок 19), ссыки на необходимые библиотеки и конфигурационный файл.

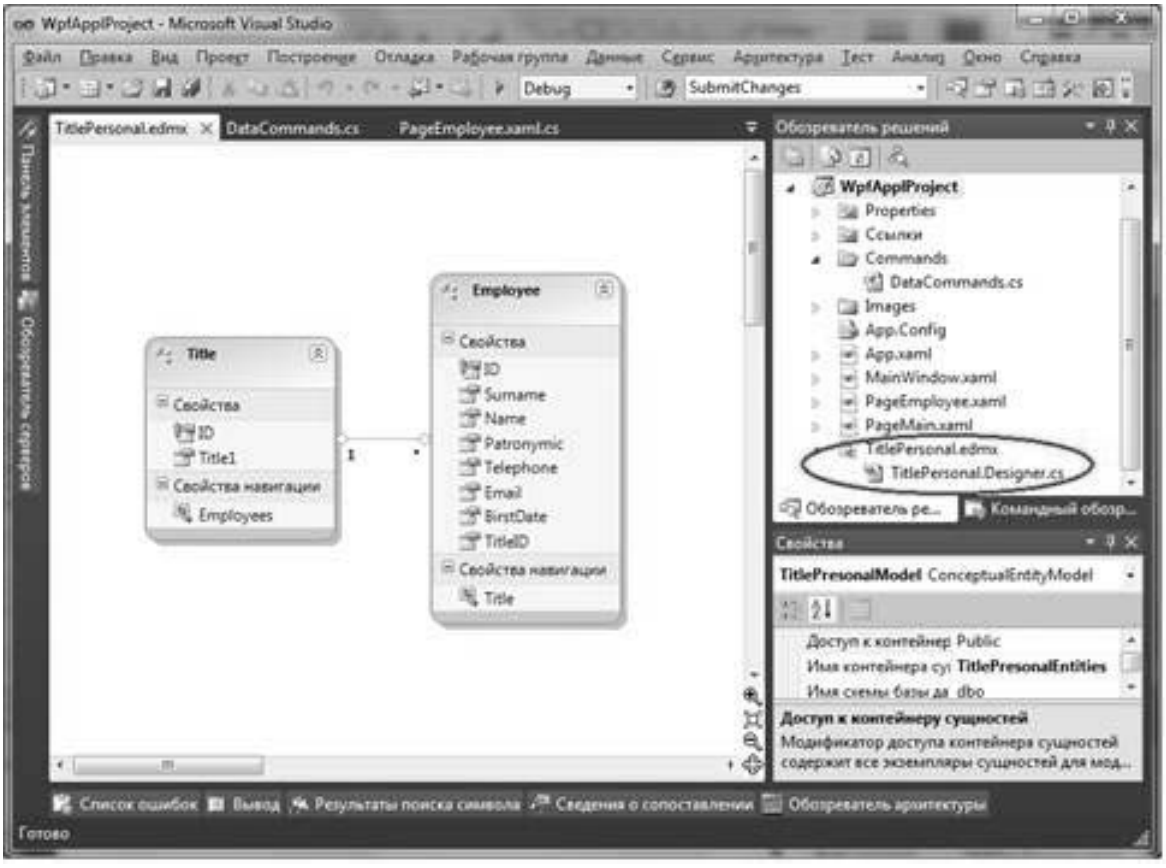

Рисунок 19 – Выбор таблиц базы данных

Автоматически сгенерированный класс TitlePresonalEntities, который наследуется от класса ObjectContext, представляет сущности базы данных TitlePerson, содержит свойства, моделирующие таблицы Employee и Title, связи между таблицами.

При создании модели данных в проекте автоматически был сгенерирован конфигурационный файл App.Config, который содержит строку соединения с базой данных.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<configSections>
        <!-- For more information on Entity Framework configuration, visit
    http://go.microsoft.com/fwlink/?LinkID=237468 -->
        <section name="entityFramework"
    type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
    EntityFramework, Version=6.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" requirePermission="false"/>
</configSections>
<startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"/>
</startup>
<connectionStrings>
        <add name="TitleEmployeeEntities"
    connectionString="metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.
    msl; provider=System.Data.SqlClient;provider connection string=&amp;quot;data
    source=(LocalDb)\MSSQLLocalDB; initial catalog=TitleEmployee;integrated
    security=True;MultipleActiveResultSets=True; App=EntityFramework&amp;quot;"
    providerName="System.Data.EntityClient"/>
</connectionStrings>
<entityFramework>
        <defaultConnectionFactory
    type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory,
    EntityFramework">
        <parameters>
            <parameter value="mssqllocaldb"/>
        </parameters>
    </defaultConnectionFactory>
    <providers>
            <provider invariantName="System.Data.SqlClient"
          type="System.Data.Entity.SqlServer.SqlProviderServices,
          EntityFramework.SqlServer"/>
    </providers>
</entityFramework>
</configuration>
```

1.3 WPF. Разработка корпоративного приложения. Подключение к БД

Цель работы: изучить методику разработки привязки данных при создании корпоративного приложения.

Для взаимодействия приложения с базой данных необходимо в коде класса страницы объявить статическое свойство контекста данных сформированной EDM-модели. Это свойство целесообразно объявлять статическим. Например:

```
public static TitlePresonalEntities DataEntitiesEmployee {get; set;}
```

Также необходимо объявить обобщенную коллекцию типа ObservableCollection<Employee> для работы приложения с коллекцией объектов базы данных. Этот тип представляет коллекцию динамических данных, обеспечивающих выдачу уведомления при получении и удалении элементов или при обновлении всего списка. Тип ObservableCollection<T> находится в пространстве имен System.Collections.ObjectModel, ссылку на которое нужно добавить в объявлении класса приложения.

```
using System.Collections.ObjectModel;
```

Экземпляры свойств контекста данных и коллекции необходимо создать в конструкторе класса страницы. Например:

```
public partial class PageEmployee : Page
    {
    public static TitleEmployeeEntities Context = new TitleEmployeeEntities();
    public PageEmployee()
    {
            InitializeComponent();
    }
    …
    }
```

Формирование данных для приложения, которые должны предоставляться из базы данных, будет проводиться при загрузке страницы приложения. Для этого в XAML-документ Page добавьте свойство Loaded.

```
Loaded="Page_Loaded"
```

В код класса страницы приложения включаем обработчик Page_Loaded.

```
private void filter()
    {
    var query = from e in Context.Employe
            where e.Surname.Contains(TextBoxSurname.Text)
                    select e;
```

```
        if(ComboBoxTitle.SelectedIndex > -1)
        {
                query = from e in query
                        where e.ID_tittle == ((Title)ComboBoxTitle.SelectedValue).ID
                                select e;
        }
        DataGridEmployee.ItemsSource = query.ToList();
}
```

Поле employees имеет тип ObjectQuery<Employee>. Класс ObjectQuery<T> представляет запрос, возвращающий коллекцию типизированных сущностей с любым количеством элементов. Запрос сформируем с помощью технологии LINQ.

```
var query = from e in Context.Employe
        where e.Surname.Contains(TextBoxSurname.Text)
                select e;
```

Результаты запроса целесообразно отсортировать, например, по фамилии сотрудника (orderby employee.Surname). Далее формируем коллекцию объектов базы данных и источник данных для сетки DataGrid.

```
foreach (Employee emp in queryEmployee)
{
         ListEmployee.Add(emp);
}
DataGridEmployee.ItemsSource = ListEmployee;
```

В результате проектирования в приложении сформирована коллекция с данными из таблицы базы данных.

Необходимо настроить сетку DataGrid для корректного отображения данных. Модифицируйте общее описание DataGrid.

1 Определите привязку для источника данных.

```
ItemsSource="{Binding}"
```

2 Отмените автоматическую генерацию столбцов.

```
AutoGenerateColumns="False"
```

3 Установите привязку к левому краю страницы.

```
HorizontalAlignment="Left"
```

4 Определите максимальные размеры сетки.

```
MaxWidth="1000" MaxHeight="295"
```

5 Установите основной и альтернативный цвета заливки сетки.

```
RowBackground="#FFE6D3EF"
AlternatingRowBackground="#FC96CFD4"
```

6 Определите цвет заливки и толщину линии для рамки сетки.

```
BorderBrush="#FF1F33EB"
BorderThickness="3"
```

7 Определите высоту строк сетки.

```
RowHeight="25"
```

8 Переопределите форму курсора при наведении указателя мыши на таблицу DataGridEmployee.

Модифицированное XAML-описание сетки DataGrid:

```
<DataGrid Name="DataGridEmployee"
AutoGenerateColumns="False"
 HorizontalAlignment="Left"
RowBackground="#FFE6D3EF"
AlternatingRowBackground="#FC96CFD4"
BorderBrush="#FF1F33EB"
BorderThickness="3"
RowHeight="25"
Cursor="Hand"
 IsReadOnly="True"
Width="Auto">
```

Привязка текстового поля. Для каждого текстового столбца задайте свойство привязки Binding. В расширении разметки привязки данного свойства определите свойство класса источника данных, к которому привязывается колонка. Далее укажите режим двусторонней привязки (Mode=TwoWay), который определяет синхронное обновление как источника, так и приемника привязки. Способ обновления источника привязки задается свойством UpdateSourceTrigger, которое принимает значение PropertyChanged, что соответствует немедленному обновлению источника привязки каждый раз при изменении свойства цели привязки. Например:

```
<DataGridTextColumn Header="Фамилия" Binding="{Binding Surname, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
```

Привязка выпадающего списка. Привязка данных к колонке типа DataGridComboBoxColumn требует определенной подготовительной работы. Если в таблице модели данных хранится не текстовое значение поля, а внешний ключ для другой таблицы, где находятся данные, то в EDM-модели можно получить значение атрибута из связанных таблиц. Для этого используется атрибут связи в таблице EDM-модели.

Например, для обеспечения возможности работы с коллекцией таблицы Title в приложении добавьте в проект папку Model и в ней создайте класс ListTitle.

```
public class ListTitle : ObservableCollection<Title>
{
	public ListTitle()
	{
		TitleEmployeeEntities context = new TitleEmployeeEntities();
		context.Title.Load();
		foreach (Title titl in context.Title.Local)
		{
			Add(titl);
		}
	}
}
```

Класс ListTitle наследуется от класса обобщенной коллекции ObservableCollection<T> и его назначение создавать коллекцию объектов Title. Поле titles является запросом типа ObjectQuery<Title>. Данному полю присваивается свойство Titles контекста данных DataEntitiesEmployee, который определен в классе приложения.

Запрос LINQ получает данные из базы данных в поле queryTitle и затем в цикле foreach формируется коллекция класса ListTitle.

Для использования класса ListTitle в XAML-документе класса приложения необходимо объявить данный класс как ресурс. При этом нужно подключить пространство имен WpfApplProject.Model.

```
xmlns:e ="clr-namespace:WpfApplProject.Model"
```

Затем определите ресурс страницы с ключом listTitle.

```
<Page.Resources>
	<e:ListTitle x:Key="ListTitles"/>
</Page.Resources>
```

Далее модифицируйте XAML-описание для типа DataGridComboBoxColumn.

```
<DataGridComboBoxColumn Header="Должность"
ItemsSource="{Binding Source={StaticResource ListTitles}}"
```

```
DisplayMemberPath="title1"
SelectedValueBinding="{Binding Path=ID_tittle, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedValuePath="ID" />
```

Источник выпадающего списка задается как статический ресурс {StaticResource listTitle}. Выводимое в ячейки колонки поле должно соответствовать полю Title1 таблицы Title EDM-модели (DisplayMemberPath="Title1"). Выбираемый в списке параметр (SelectedValueBinding) должен быть привязан к полю TitleID таблицы Employee.

```
SelectedValueBinding="{Binding Path=TitleID, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
```

Выбор в свойства SelectedValueBinding производится по пути, определенному свойством SelectedValuePath (SelectedValuePath="ID").

Привязка даты. При привязке даты ставится задача для невыделенной ячейки представлять дату в виде цифр дня, месяца и года, разделенных точками, а для выделенной ячейки – использовать класс для выбора даты.

Решение данной задачи осуществлено с помощью разработки двух шаблонов данных DataTemplate.

В первом шаблоне, для которого задан ключ DateTemplate, используется элемент управления TextBlock, свойство Text которого привязывается к атрибуту BirstDate таблицы Employee. Данные в привязке форматируются свойством StringFormat и строковые данные выравниваются по центру. Этот шаблон используется для визуализации данных в режиме их просмотра.

```
<DataTemplate x:Key="DateTemplate" >
    <TextBlock Text="{Binding BirthData, StringFormat={}{0:dd\.}{0:MM\.}{0:yyyy}}"
    VerticalAlignment="Center" HorizontalAlignment="Center" />
</DataTemplate>
```

Второй шаблон с ключом EditingDateTemplate использует класс DatePicker. Этот шаблон используется в режиме редактирования данных.

```
<DataTemplate x:Key="EditingDateTemplate">
    <DatePicker SelectedDate="{Binding BirthData, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"/>
</DataTemplate>
```

Разработанные шаблоны необходимо добавить к ресурсам в XAML-документ страницы PageEmployee.

Для настройки колонки “Дата рождения” модифицируем её XAML-описание:

```
<DataGridTemplateColumn Header="Дата рождения"
```

```
CellTemplate="{StaticResource DateTemplate}"
CellEditingTemplate="{StaticResource EditingDateTemplate}" />
```

Невыделенную ячейку колонки (CellTemplate) свяжите с ресурсом DateTemplate, а редактируемую ячейку (CellEditingTemplate) – с ресурсом EditingDateTemplate.

Ячейка столбца для отображения даты типа DataGridTemplateColumn имеет три представления (рисунок 20). Если ячейка не выделена, то представление обычное текстовое (рисунок 20, *а*). При выделении ячейки прорисовывается свернутое содержание класса DatePicker (рисунок 20, *б*). При щелчке на ячейке появляется календарь (рисунок 20, *в*).

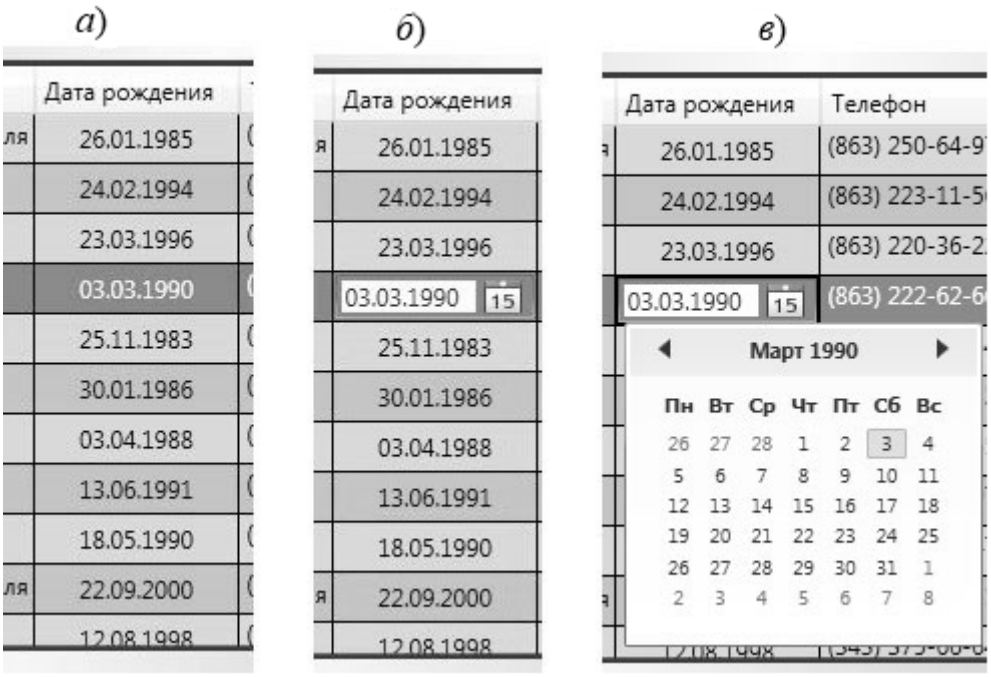

Рисунок 20 – Отображение даты в сетке

Разработка бизнес-логики приложения

Для редактирования данных в приложении модифицируйте метод EditCommandBinding_Executed.

```
private void EditCommandBinding_Executed(object sender, ExecutedRoutedEventArgs e)
{
    DataGridEmployee.IsReadOnly = false;
    DataGridEmployee.BeginEdit();
}
```

Свойству IsReadOnly сетки DataGridEmployee присвойте значение false, что обеспечит возможность редактирования строк и ячеек сетки. Далее

используйте метод BeginEdit для начала редактирования ячейки, на которую наведен указатель мыши.

Результаты редактирования необходимо сохранить в базе данных. Для этого необходимо вызвать метод SaveCommandBinding_Executed.

```
private void SaveCommandBinding_Executed(object sender, ExecutedRoutedEventArgs e)
{
    Context.SaveChanges();
    DataGridEmployee.IsReadOnly = true;
}
```

В методе SaveCommandBinding_Executed вызывается метод SaveChanges класса dataEntitiesEmployee. Метод SaveChanges сохраняет все обновления в источнике данных.

Для создания новых данных модифицируйте метод NewCommandBinding_Executed. Например:

```
private void AddCommandBinding_Executed(object sender, ExecutedRoutedEventArgs e)
{
    Employe employee = new Employe
    {
        BirthData = Convert.ToDateTime("2000-01-01"),
        Name = "unknown",
        Surname = "unknown",
        Email = "unknown",
        Patronymic = "unknown",
        Telephone = "unknown"
    };
    try
    {
        Context.Employe.Add(employee);
    }
    catch (Exception ex)
        {
        throw new ApplicationException("Ошибка добавления данных" + ex.ToString());
    }
}
```

В методе NewCommandBinding_Executed объект employee класса Employee создается с помощью метода CreateEmployee. В блоке try ... catch созданный объект employee добавляется в контекст данных методом AddObject сущности Employees EDM-модели DataEntitiesEmployee, а также в коллекцию ListEmployee.

При инициализации команды создания данных в сетке формируется первоначальная строка с данными «по умолчанию». В сформированную строку необходимо ввести реальные данные и сохранить их, вызвав команду «Сохранить».

Для удаления данных по сотруднику используем метод DeleteCommandBinding_Executed.

```
private void DeleteCommandBinding_Executed(object sender, ExecutedRoutedEventArgs e)
{
	Employe emp = DataGridEmployee.SelectedItem as Employe;
	if (emp != null)
	{
		MessageBoxResult result = MessageBox.Show("Вы уверены, что хотите удалить
	 запись?", "Предупреждение", MessageBoxButton.OKCancel);
		 if (result == MessageBoxResult.OK)
		 {
			Context.Employe.Remove(emp);
		 }
	}
	else
	{
		MessageBox.Show("Выберите строку для удаления");
	}
}
```

В методе определяется объект emp класса Employee, который выделен в сетке DataGridEmployee.

Если объект emp найден, то с помощью диалогового окна класса MessageBoxResult выводится сообщение с данными, которые предполагается удалить.

Объект emp удаляется из коллекции ListEmployee.

```
Context.Employe.Remove(emp);
```

Метод UndoCommandBinding_Executed реализует отмену редактирования последних элементов сетки DataGridEmployee и переводит её в режим просмотра.

Создайте метод RewriteEmployee, который будет обновлять контекст данных и коллекцию ListEmployee.

```
private void RewriteEmployee()
{
	DataEntitiesEmployee = new TitlePresonalEntities();
	ListEmployee.Clear();
	GetEmployees();
}
```

С учетом проведенных модификаций кода метод UndoCommandBinding_Executed будет иметь следующий вид.

```
private void UndoCommandBinding_Executed(object sender, ExecutedRoutedEventArgs e)
{
	Context = new TitleEmployeeEntities();
```

```
        Context.Employe.Load();
        DataGridEmployee.ItemsSource = Context.Employe.Local;
        DataGridEmployee.IsReadOnly = true;
}
```

При запуске команды Отмена будет отменено редактирование в текущей ячейке сетки, а если редактировалось несколько ячеек, то при повторном запуске команды будет обновлена вся строка.

Валидация данных

Разработку пользовательского правила валидации данных рассмотрим на примере проверки шаблона при вводе свойства Email, т. е. адреса электронной почты. В строке ввода должны присутствовать символы @ и точка.

Спроектируйте правило проверки для привязки, которое будет использоваться со столбцом «Электронная почта». Для этого необходимо создать класс правила проверки EmailRule, который должен наследоваться от базового класса ValidationRule. Класс EmailRule поместите во вновь созданную папку ValidationRules проекта.

```
public class EmailRule : ValidationRule
{
        public override ValidationResult Validate(object value, System.Globalization.CultureInfo
cultureInfo)
        {
                string email = string.Empty;
                if (value != null)
                {
                        email = value.ToString();
                }
                else
                        return new ValidationResult(false, " Адрес электронной почты не задан! ");
                if (email.Contains("@") && email.Contains("."))
                {
                        return new ValidationResult(true, null);
                }
                else
                {
                        return new ValidationResult(false,"Адрес электронной почты должен
                        содержать символы @ и(.) точки \n Шаблон адреса:
                        adres@mymail.com");
                }
        }
}
```

В классе правил проверки необходимо переопределить метод Validate, который возвращает результат проверки – экземпляр классаValidationResult. Если проверка проходит успешно, тогда возвращаемым значением является объект класса ValidationResult, который создается с параметрами true и null.

```
return new ValidationResult(true, null);
```

Если проверка приводит к выявлению несоответствия, то класс ValidationResult создается с параметрами false и строка сообщения об ошибке.

```
return new ValidationResult(false, "Адрес электронной почты должен содержать символы @ и (.) точки \n Шаблон адреса:   adres@mymail.com");
```

Для использования объекта EmailRule в XAML-документе страницы приложения добавьте в её описание пространство именWpfApplProject.ValidationRules.

```
xmlns:rule="clr-namespace:WpfApplProject.ValidationRules"
```

Внесите изменения в XAML-описание колонки Электронная почта.

```
<DataGridTextColumn Header="Электронная почта" EditingElementStyle="{StaticResource errorStyle}">
    <DataGridTextColumn.Binding >
        <Binding Path="Email" Mode="TwoWay"
        UpdateSourceTrigger="PropertyChanged" ValidatesOnExceptions ="True" >
            <Binding.ValidationRules>
                <rule:EmailRule/>
            </Binding.ValidationRules>
        </Binding>
    </DataGridTextColumn.Binding>
</DataGridTextColumn>
```

Для привязки Binding свойству ValidatesOnExceptions задайте значение true. Задание данного свойства обеспечивает использование элемента ExceptionValidationRule. ExceptionValidationRule предоставляет встроенное правило проверки, проверяющее возникновение исключений при обновлении свойства источника. В случае возникновения ошибки механизм привязки создает ValidationError с исключением и добавляет его в коллекцию Validation.Errors привязанного элемента. Отсутствие ошибок сбрасывает это состояние обратной связи проверки, если другое правило не вызовет событие проверки.

Правило проверки для свойства ValidationRules класса Binding.

```
<Binding.ValidationRules>
    <rule:EmailRule />
</Binding.ValidationRules>
```

Модель привязки данных WPF позволяет связывать ValidationRules с объектом Binding, используя пользовательские правила. Подсистема привязки проверяет каждое из ValidationRule, связанных с привязкой, каждый раз, когда вводимое значение (значение свойства цель привязки) переносится в свойство источник привязки.

В WPF имеются стандартные стили для отображения ячеек сетки при возникновении ошибки ввода, но они являются недостаточно информативными. Для более эффектного выделения ячейки сетки при ошибке ввода создайте специальный стиль errorStyle.

```
<Style x:Key="errorStyle" TargetType="{x:Type TextBox}">
    <Setter Property="Padding" Value="-2"/>
    <Style.Triggers>
        <Trigger Property="Validation.HasError" Value="True">
            <Setter Property="Background" Value="Red"/>
            <Setter Property="BorderThickness" Value="1" />
            <Setter Property="ToolTip"
            Value="{Binding RelativeSource={RelativeSource Self},
            Path=(Validation.Errors)[0].ErrorContent}"/>
        </Trigger>
    </Style.Triggers>
</Style>
```

В стиле errorStyle триггер срабатывает, когда свойство Validation.HasError принимает значение true. При этом цвет заливки ячейки становится красным и при наведении на ячейку указателя мыши выводится сообщение-подсказка (свойство ToolTip), текст которого определяется в правиле проверки EmailRule при обнаружении ошибки (Validation.Errors)[0] .ErrorContent).

Стандартные средства WPF при обнаружении ошибки ввода в ячейке помечают строку сетки красным восклицательным знаком. Создадим шаблон ControlTemplate для визуального указания ошибки при проверке строки. Данный шаблон задается для свойства RowValidationErrorTemplate сетки DataGrid. Проектируемый шаблон должен обеспечить вывод красного круга с белым восклицательным знаком слева от строки сетки с обнаруженной ошибкой ввода. При наведении указателя мыши на круг должна выводиться подсказка, сформированная в классе правил проверки ввода.

```
<DataGrid.RowValidationErrorTemplate>
    <ControlTemplate>
        <Grid Margin="0,-2,0,-2"
        ToolTip="{Binding RelativeSource={RelativeSource FindAncestor,
        AncestorType={x:Type DataGridRow}},
        Path=(Validation.Errors)[0].ErrorContent}">
            <Ellipse StrokeThickness="0" Fill="Red" Width="{TemplateBinding
            FontSize}" Height="{TemplateBinding FontSize}"/>
            <TextBlock Text="!" FontSize="{TemplateBinding FontSize}"
            FontWeight="Bold"
            Foreground="White" HorizontalAlignment="Center"/>
```

```
            </Grid>
        </ControlTemplate>
</DataGrid.RowValidationErrorTemplate>
```

1.4 WPF. Разработка корпоративного приложения. Поиск и фильтрация

Цель работы: изучить методику разработки поиска и фильтрации в корпоративном приложении.

Функцию поиска данных в приложении необходимо выполнить на основе данных, имеющихся в базе данных варианта лабораторной работы. Параметры поиска необходимо согласовать с преподавателем.

Общий подход к поиску реализации поиска данных, получаемых из базы данных, рассмотрим на примере поиска сотрудника по фамилии и должности. Для реализации функций поиска добавьте на страницу приложения элементы контроля в соответствии с рисунком 21.

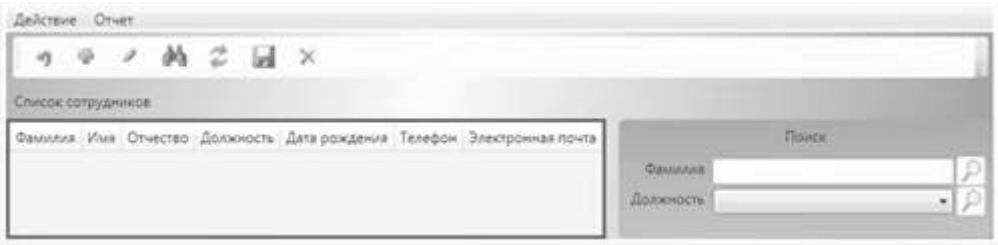

Рисунок 21 – Дизайн страницы с элементами контроля для поиска

XAML-описание элементов контроля, поддерживающих поиск имеет следующий вид.

```
<TextBlock Grid.Column="0" Grid.Row="1" x:Name="TextBlockSurname" Text="Фамилия" />
<TextBlock Grid.Column="0" Grid.Row="2" x:Name="TextBlockTitle" Text="Должность" />
<TextBox Grid.Column="1" Grid.Row="1" x:Name="TextBoxSurname"/>
<ComboBox Grid.Column="1" Grid.Row="2" x:Name="ComboBoxTitle"
ItemsSource="{Binding Source={StaticResource ListTitles}}"
DisplayMemberPath="title1"/>
<Button Width="25" Height="25" Grid.Column="2" Grid.Row="1"
x:Name="ButtonFindSurname" ToolTip="Поиск по фамилии"
Click="ButtonFindSurname_Click">
    <Image Source="icons/Find.ico"/>
</Button>
<Button Width="25" Height="25" Grid.Column="2" Grid.Row="2" x:Name="ButtonFindTitle"
ToolTip="Поиск по должности" Click="ButtonFindTitle_Click">
    <Image Source="icons/Find.ico" />
</Button>
```

Элементы контроля размещаем в рамке BorderFind. В рамке располагаем сетку gridFind с тремя колонками и строками. Первая строка в объединенных колонках содержит текстовый блок find с текстом «Поиск». Во второй и третьей строках первой колонки размещены текстовые блоки TextBlockSurname и TextBlockTitle соответственно. Во второй колонке размещаются элементы контроля для ввода информации: текстовый блок TextBoxSurname (вторая строка) и выпадающий список ComboBoxTitle (третья строка). В третьей колонке размещаются кнопки: для поиска по фамилии ButtonFindSurname (вторая строка) и для поиска по должности ButtonFindTitle (третья строка).

При загрузке страницы PageEmployee рамка BorderFind невидима, т. к. свойству Visibility задано значение «Hidden». При активизации команды Find из меню или панели инструментов запускается обработчик FindCommandBindingExecuted, который делает рамку ButtonFindTitle видимой.

```
private void btnFind_Click(object sender, RoutedEventArgs e)
{
	if(findGrid.Visibility == Visibility.Visible)
	{
		findGrid.Visibility = Visibility.Hidden;
	}
	else
	{
		findGrid.Visibility = Visibility.Visible;
	}
}
```

Если в текстовом блоке TextBoxSurname не введена информация или в выпадающем списке ComboBoxTitle не сделан выбор, то кнопки ButtonFindSurname и ButtonFindTitle будут недоступны.

При вводе информации в текстовый блок TextBoxSurname запускается обработчик TextBoxSurname_TextChanged, который делает доступной кнопку ButtonFindSurname.

```
private void TextBoxSurname_TextChanged(object sender, TextChangedEventArgs e)
{
	ButtonFindSurname.IsEnabled = true;
	ButtonFindTitle.IsEnabled = false;
	ComboBoxTitle.S = -1;
}
```

При нажатии кнопки ButtonFindSurname запускается обработчик ButtonFindSurname_Click.

```
private void ButtonFindSurname_Click(object sender, RoutedEventArgs e)
{
	string surname = TextBoxSurname.Text;
	DataEntitiesEmployee = new TitlePresonalEntities();
	ListEmployee.Clear();
```

```
        ObjectQuery<Employee> employees = DataEntitiesEmployee.Employees;
        var queryEmployee = from employee in employees
                where employee.Surname == surname
                        select employee;
        foreach (Employee emp in queryEmployee)
        {
                ListEmployee.Add(emp);
        }
        if (ListEmployee.Count > 0)
        {
                DataGridEmployee.ItemsSource = ListEmployee;
                ButtonFindSurname.IsEnabled = true;
                ButtonFindTitle.IsEnabled = false;
        }
        else
        MessageBox.Show("Сотрудник с фамилией \n"+surname+"\n не найден",
        "Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning);
}
```

В данном обработчике события Click выполняется LINQ запрос на получение данных из базы TitlePersonal в соответствии с заданной фамилией сотрудника.

```
var queryEmployee = from employee in employees
        where employee.Surname == surname
                select employee;
```

При выполнении запроса по фамилии страница PageEmployee имеет вид, приведенный на рисунке 22.

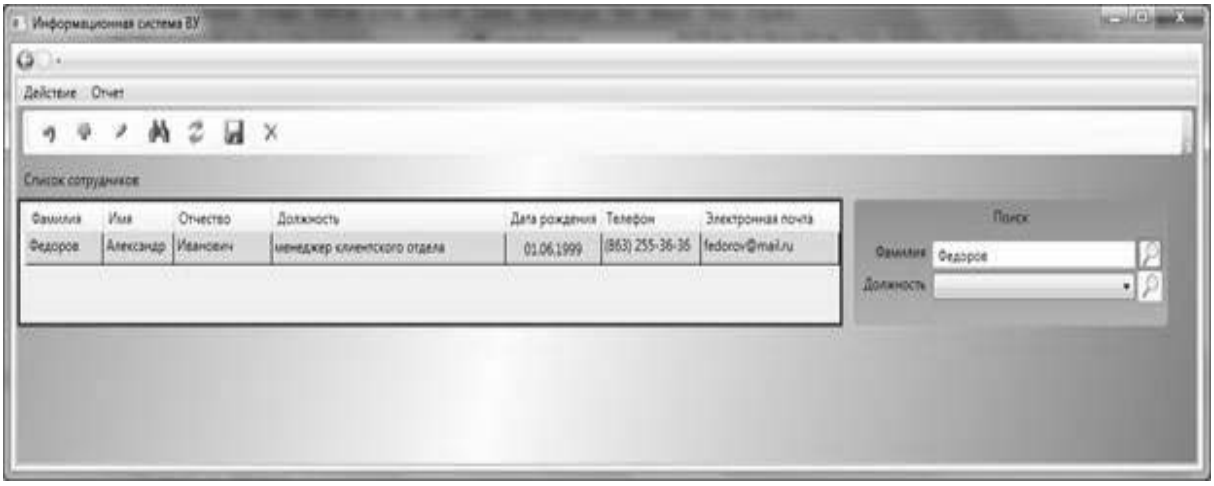

Рисунок 22 – Поиск по фамилии

Найденные данные по сотруднику можно редактировать и удалять.

Если поиск в базе данных не привел к нахождению информации, то выдается информационное сообщение (рисунок 23).

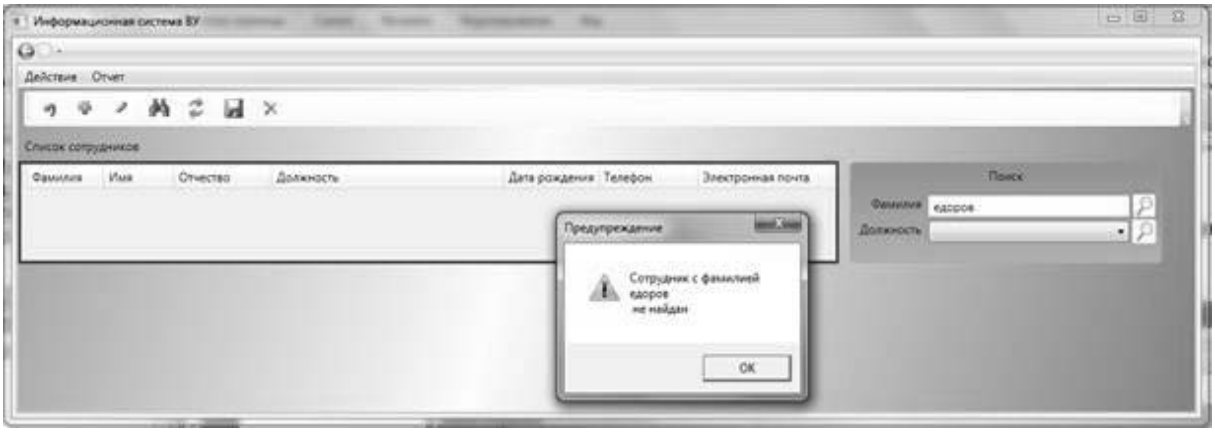

Рисунок 23 – Неудачный поиск по фамилии

Для поиска информации из базы данных по должности сотрудников необходимо сделать выбор из выпадающего списка ComboBoxTitle. В результате выбора срабатывает обработчик ComboBoxTitle_SelectionChanged, который делает доступной кнопку ButtonFindTitle.

```
private void ComboBoxTitle_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    ButtonFindTitle.IsEnabled = true;
    ButtonFindSurname.IsEnabled = false;
    TextBoxSurname.Text = "";
}
```

При нажатии на кнопку ButtonFindTitle срабатывает обработчик ButtonFindTitle_Click.

```
private void ButtonFindTitle_Click(object sender, RoutedEventArgs e)
{
    DataEntitiesEmployee = new TitlePresonalEntities();
    ListEmployee.Clear();
    Title title = ComboBoxTitle.SelectedItem as Title;
    ObjectQuery<Employee> employees = DataEntitiesEmployee.Employees;
    var queryEmployee = from employee in employees
            where employee.TitleID == title.ID
                orderby employee.Surname
                    select employee;
    foreach (Employee emp in queryEmployee)
    {
            ListEmployee.Add(emp);
    }
    DataGridEmployee.ItemsSource = ListEmployee;
}
```

При запуске данного обработчика выполняется LINQ-запрос к базе данных TitlePersonal для выборки данных по сотрудникам, имеющим заданную должность.

Для обновления выводимого списка сотрудников в приложение добавлена функция обновления, которая реализуется командой Refresh. Данная команда добавлена в коллекцию команд страницы PageEmployee.

```
<Page.CommandBindings>
....
<CommandBinding Command="Refresh" Executed="RefreshCommandBinding_Executed" />
...
</Page.CommandBindings>
```

Вызов команды добавлен в меню:

```
<MenuItem Header="Обновить" Command="Refresh"/>
```

И панель инструментов:

```
<Button Name="Refresh" Margin="5,2,5,2"  Command="Refresh" ToolTip="обновить данные по сотрудникам">
    <Image Source="Images/Refresh.jpg" ></Image>
</Button>
```

Основное назначение обработчика команды RefreshCommandBinding_Executed – обновление списка ListEmployee.

```
private void RefreshCommandBinding_Executed(object sender,
ExecutedRoutedEventArgs e)
{
    RewriteEmployee();
    DataGridEmployee.IsReadOnly = false;
    isDirty = false;
    SetUnvisibilityFind();
}
```

Команда «Обновить» обычно используется после поиска данных или ввода данных по новому сотруднику.

После выполнения данного приложения рекомендуется выполнить один из предложенных ниже вариантов самостоятельных заданий для закрепления навыков разработки корпоративных настольных приложений по технологии WPF.

Краткие итоги

В данных работах были рассмотрены вопросы разработки страничного WPF-приложения, организации перехода по страницам, создания системы меню, панели команд, табличного представления информации для просмотра и

редактирования данных, системы универсальных команд многократного использования и подключения базы данных.

Индивидуальное задание.

Студент должен получить индивидуальное задание по разработке корпоративного приложения у преподавателя и выполнить его в соответствии с подразделами 1.1–1.4.

Контрольные вопросы

1 Какой дескриптор верхнего уровня используется в WPF-проектах?
2 Какое назначение метода InitializeComponent() в коде класса?
3 Сколько можно вложить элементов в класс Page?
4 Поясните назначение свойства Content класса Page.
5 В каких контейнерах можно размещать страницы WPF?
6 Какие элементы контроля используются для перехода между страницами приложения?
7 Из каких объектов можно сформировать меню приложения?
8 Какие типы столбцов автоматически генерируются для элемента управления DataGrid?
9 Поясните назначение модели команд WPF.
10 Какие в WPF имеются библиотеки базовых команд?
11 Как можно управлять доступностью команд в приложении WPF?

2 Технология разработки приложений ASP.NET по шаблону MVC

2.1 Разработка приложений ASP.NET по шаблону MVC: модель и контроллер

Цель работы: изучить методику разработки модели и контроллера MVC-приложения.

В меню Файл выберите Создать -> Проект. Откроется диалоговое окно Создать проект. Необходимо отметить галочку создания тестов. При создании проекта веб-приложения ASP.NET MVC компоненты MVC разделяются по папкам проекта.

Проект модульных тестов также готов к компиляции и запуску. В него будет добавлен контроллер с методом действия и представления, 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 контролер **MVC 5 Controller – Empty.**

3 В поле **Имя** введите MapsController (рисунок 24).

Рисунок 24 – Ввод имени контроллера

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.

```
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.

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**.

4 Visual Studio добавит класс MapsControllerTest в тестовый проект.

5 Откройте класс MapsControllerTest и введите следующий код:

```
using System;
using System.Collections.Generic;
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);
         }
     }
}
```

2.2 Разработка компонента ASP.NET по шаблону MVC: представление

Цель работы: изучить методику разработки представления MVC-приложения.

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!1d150475.3322718522!2d
    27.451567660957238!3d53.89305669809051!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3
    m3!1m2!1s0x46dbcfd35b1e6ad3%3A0xb61b853ddb570d9!2z0JzQuNC90YHQug!5e0!3m2!
    1sru!2sby!4v1576007412668!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=en-us">
      </script>
<script type="text/javascript">
  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!2d27.451567
```

```
660957238!3d53.89305669809051!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x46d
bcfd35b1e6ad3%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!2d30.218223
075309055!3d53.88123104449437!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x46d
0521c52844571%3A0xcf85d14239bb73b6!2z0JzQvtCz0LjQu9GR0LI!5e0!3m2!1sru!2sby!4v1576
007883514!5m2!1sru!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 (рисунок 25).

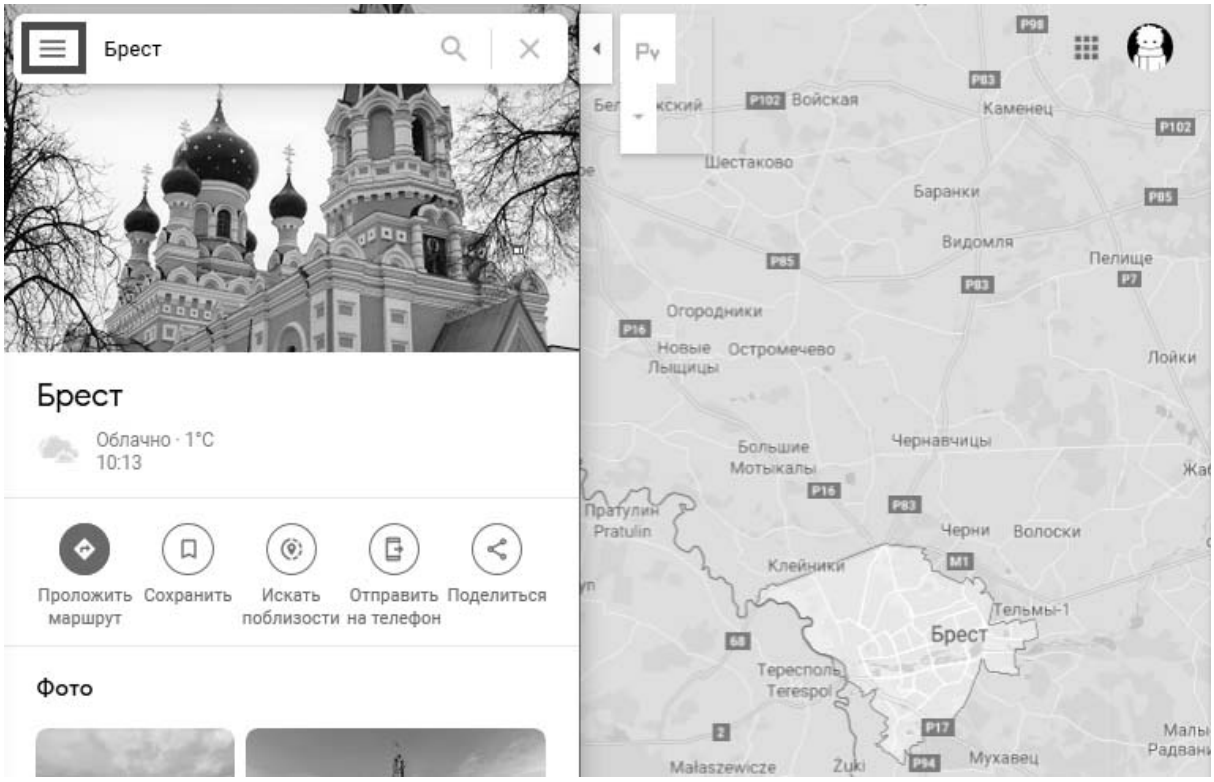

Рисунок 25 – Получение iframe с Google Maps для города Брест

Находим пункт «Ссылка/код» и кликаем на него (рисунок 26).

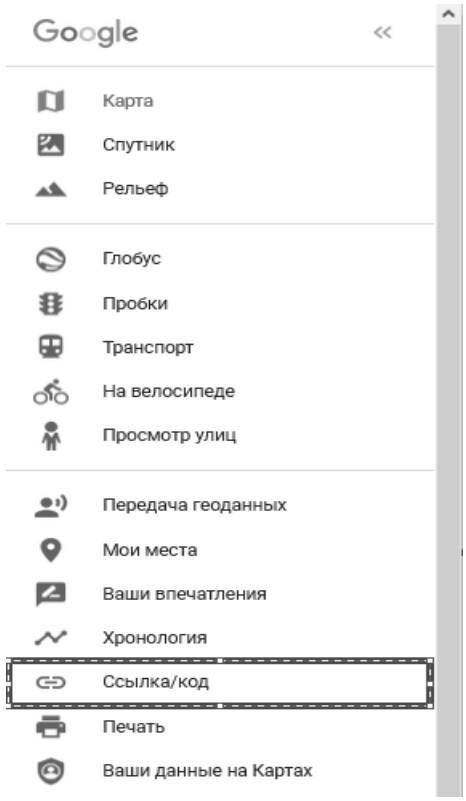

Рисунок 26 – Получение iframe с Google Maps для города Брест

Выбираем пункт «встраивание карт». Далее можно изменить размер отображаемого фрейма (рисунок 27) и отредактировать размер области.

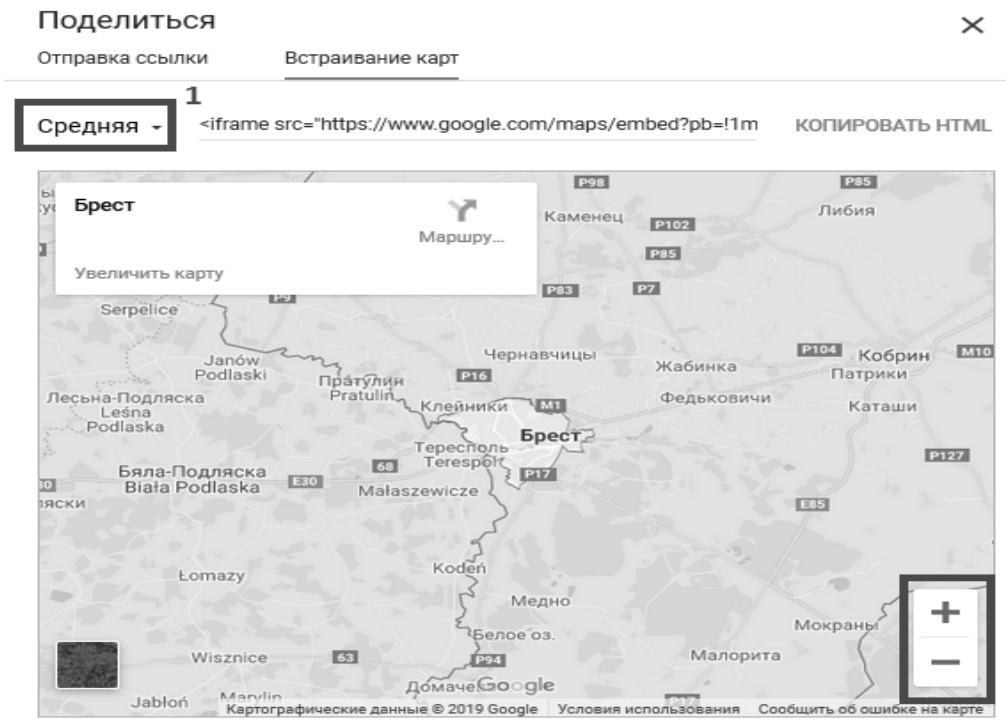

Рисунок 27 – Изменение размеров фрейма

Достаем значение атрибута «src» любыми возможными методами.

Изменим функцию GetMap следующим образом:

```
function GetMap(mapID) {
        var a = document.getElementById("frame1");
        switch (mapID) {
        case 'Brest':
      a.src =
"https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d78449.31346339424!2d23.632848
329964006!3d52.088084177221866!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x47
210c0223630975%3A0x4d319ea41f64ae99!2z0JHRgNC10YHRgg!5e0!3m2!1sru!2sby!4v157600
8086154!5m2!1sru!2sby" ;
                    break;
                }
        }
```

Теперь при выборе Бреста в списке отобразится нужная нам карта.

Индивидуальное задание.

По данному примеру добавьте карты следующих городов: Могилев, Витебск, Гродно, Гомель и сделайте их выбор в представлении MVC. Также выбор городов можно согласовать с преподавателем.

Контрольные вопросы

1 Назначение шаблона MVC. Состав его компонент.
2 Назначение контроллера в проекте MVC.
3 Назначение модели в проекте MVC.
4 Назначение представлений в проекте MVC.
5 Как добавить в проект MVC модульные тесты?
6 Как выполнить встраивание карт в представлении?

Список литературы

1 Джеффри, Рихтер. CLR via C#: программирование на платформе Microsoft Net FRAMEWORK 4.5 на языке C# / Р. Джеффри: Питер, 2019. – 896 с. 4-е изд.

2 Фримен, Адам. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов: Пер. с англ. / А. Фримен. – Москва: И. Д. Вильямс, 2018. – 736 с.: ил.

3 Руководство по WPF [Электронный ресурс] / METANIT.COM: Сайт о программировании – Режим доступа: http:// metanit.com/sharp/wpf/. – Дата доступа: 18.09.2020.