МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

ИНТЕГРАЦИЯ ПРОГРАММНЫХ МОДУЛЕЙ И КОМПОНЕНТ

Методические рекомендации к лабораторным работам для студентов направления подготовки 01.03.04 «Прикладная математика» дневной формы обучения

Могилев 2023

УДК 004.428.8
ББК 32.973-018.2
И73

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «17» октября 2023 г., протокол № 3

Составитель ст. преподаватель Н. В. Выговская

Рецензент канд. техн. наук, доц. С. К. Крутолевич

Методические рекомендации предназначены для студентов направления подготовки 01.03.04 «Прикладная математика» дневной формы обучения по дисциплине «Интеграция программных модулей и компонент».

Учебное издание

ИНТЕГРАЦИЯ ПРОГРАММНЫХ МОДУЛЕЙ И КОМПОНЕНТ

Ответственный за выпуск	А. И. Якимов
Корректор	И. В. Голубцова
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

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, г. 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 «Интеграция программных модулей и компонент» заключается в закреплении студентами практических навыков создания и сопровождения программных систем современных ЭВМ на платформах .NET с компонентным подходом на языке C#.

Дисциплина «Интеграция программных модулей и компонент» 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 и 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 Лабораторная работа № 1. Разработка программ методом TDD

Цель работы: изучить методики гибкой (agile) разработки программного обеспечения и управления проектами на примере программирования на основе тестирования.

Теоретические сведения

Разработка через тестирование (англ. *test-drivendevelopment*) – техника программирования, при которой модульные тесты для программы или ее фрагмента пишутся до самой программы (англ. *test-firstdevelopment*) и, по существу, управляют ее разработкой. Является одной из основных практик экстремального программирования.

Разработка в стиле TDD состоит из коротких циклов (длительностью от двух минут в зависимости от опытности и стиля работы программиста). Каждый цикл включает следующие шаги.

1 Из репозитория извлекается программная система, находящаяся в согласованном состоянии, когда весь набор модульных тестов выполняется успешно.

2 Добавляется новый тест. Он может состоять в проверке, реализует ли система некоторое новое поведение или содержит ли некоторую ошибку, о которой недавно стало известно.

3 Успешно выполняется весь набор тестов, кроме нового теста, который выполняется неуспешно. Этот шаг необходим для проверки самого теста – включен ли он в общую систему тестирования и правильно ли отражает новое требование к системе, которому она, естественно, еще не удовлетворяет.

4 Программа изменяется с тем, чтобы как можно скорее выполнялись все тесты. Нужно добавить самое простое решение, удовлетворяющее новому тесту, и одновременно с этим не испортить существующие тесты. Большая часть нежелательных побочных и отдаленных эффектов от вносимых в программу изменений отслеживается именно на этом этапе с помощью достаточно полного набора тестов.

5 Весь набор тестов выполняется успешно.

6 Теперь, когда требуемая в этом цикле функциональность достигнута самым простым способом, программа перестраивается (рефакторинг) для улучшения структуры и устранения избыточного, дублированного кода.

7 Весь набор тестов выполняется успешно.

8 Комплект изменений, сделанных в этом цикле в тестах и программе заносится в репозиторий (операция commit), после чего программа снова находится в согласованном состоянии и содержит четко осязаемое улучшение по сравнению с предыдущим состоянием.

Данный цикл упрощенно описывается Кентом Беком в своей книге как «красный – зеленый – рефакторинг». Красный и зеленый – это цвета полоски

в среде тестирования JUnit, которая показывает, все тесты сработали или нет. При этом на первом («красном») этапе необходимо добиться того, чтобы программа просто компилировалась, без срабатывания добавленного теста.

Задание

Применяя методику разработки на основе тестирования (testdrivendesign, сначала тесты), разработать библиотеку классов для вариантов заданий, приведенных в таблице 1.1.

Разработка должна выполняться последовательно, в строгом соответствии с методикой. Для написания и выполнения тестов использовать ПО Unit (xUnit) для выбранного языка программирования.

Таблица 1.1 – Варианты заданий

Вариант	Задание
1	Дан текст. Сделать заглавной каждую букву каждого слова, начинающегося с заглавной буквы
2	Дан текст. В каждом слове текста заменить заданную литеру заданной литерой (сочетанием литер). Пример: Заменяемая литера: “б”, заменяющее сочетание литер: “ку”, слово: “а***б***ракада***б***ра”, результат: “а***ку***ракада***ку***ра”
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 заданного 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: ‘((aaabbccccddd)(eeefggghhkl)) 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

Контрольные вопросы

1 Что такое разработка через тестирование?
2 Расшифруйте термин TDD.
3 Опишите кратко этапы цикла разработки TDD.

2 Лабораторная работа № 2. WPF. Разработка корпоративного приложения. Элементы интерфейса

Цель работы: ознакомиться с WPF-приложениями и структурой этих приложений.

Теоретические сведения

Для разработки приложения необходимо создать WPF-проект, инструментальная система сгенерирует следующий XAML-документ:

```
<Window x:Class="WpfApplProject.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApplProject"
mc:Ignorable="d"
Title="ИнформационнаясистемаВУ" Height="450" Width="825">
        <Grid>
        <Frame x:Name="frame" Content="Frame" Margin="3"
Source="/WpfApplProject;component/PageMain.xaml"
NavigationUIVisibility="Hidden">
        <Frame.Background>
        <LinearGradientBrush>
        <LinearGradientBrush.GradientStops>
        <GradientStop Offset="0.00" Color="#FFFF8686"/>
        <GradientStop Offset="1.00" Color="#FFA5B1FF"/>
        </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>

        </Frame.Background>
        </Frame>
        </Grid>
</Window>
```

В данном 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.

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, нужно задать XAML атрибут Name. Так, для задания имени элементу Grid необходимо записать следующую разметку:

<GridName="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 видоизменную версию класса Window;

– Frame, находящийся внутри другого окна или другой страницы;

– Frame, обслуживаемый непосредственно в Internetexplorer.

Задание

Создать интерфейс WPF-приложения с навигацией по страницам для работы с базой данных. Разрабатываемое приложение должно обеспечивать хранение и обработку указанных данных по варианту. Функции приложения:

– просмотр данных;
– ввод данных;
– редактирование данных;
– удаление данных;
– поиск данных по клиенту.

Вариант 1

Информационная подсистема ведения счетов клиентов.

БД включает следующие таблицы: Account, Aggreement, Bank, TypeAccount.

Назначение подсистемы ведения счетов клиентов – поддержание в актуальном состоянии инвестиционных счетов клиентов.

Вариант 2

Информационная подсистема ведения адресов клиентов.

БД включает следующие таблицы: City, Region, Address, Country.

Назначение подсистемы ведения адресов клиентов поддержание в актуальном состоянии адресов клиентов.

Вариант 3

Информационная подсистема ведения договоров клиентов.

БД включает следующие таблицы: Person, Aggreement, StatusAggrement, TypeAggrement.

Назначение подсистемы ведения договоров клиентов – поддержание в актуальном состоянии договоров клиентов.

Вариант 4

Информационная подсистема ведения клиентов – физических лиц.

БД включает следующие таблицы: Person, Citizen, Document.

Назначение подсистемы ведения клиентов поддержание в актуальном состоянии информации по клиентам – физическим лицам.

Контрольные вопросы

1 Какой элемент верхнего уровня есть в XAML-документе, который генерирует система при создании WPF-приложения?

2 Как задать имя элементу в XAML-документе?

3 В каких контейнерах можно размещать страницы WPF?

4 Из каких объектов можно сформировать меню приложения?

5 Какие элементы контроля используются для перехода между страницами приложения?

3 Лабораторная работа № 3. WPF. Разработка корпоративного приложения. Разработка БД

Цель работы: изучить методику разработки модели данных корпоративного приложения.

Теоретические сведения

Для создания EDM-модели данных необходимо добавить в проект новый элемент – модель ADO.NET EDM (рисунок 3. 1), присвоив файлу модели имя в соответствии с заданием, в рассматриваемом примере модель имеет имя TitleEmployee.edmx.

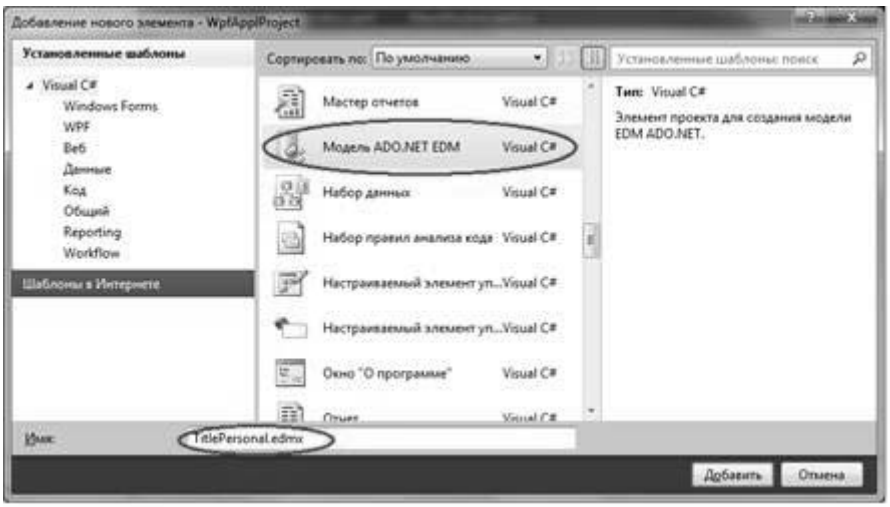

Рисунок 3.1 – Добавление в проект модели EDM

В мастере создания EDM-модели выберем опцию «Создать из базы данных» (рисунок 3. 2). Для создания соединения с базой данных в окне «Выбор подключения к данным» укажем соединение с базой данных (имя_сервера.имя_базы данных). Зададим имя модели данных – TitlePersonEntities (рисунок 3. 3).

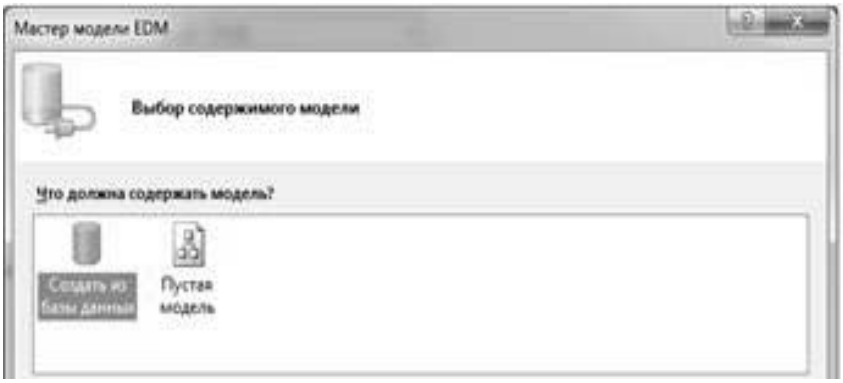

Рисунок 3.2 – Создание модели EDM из базы данных

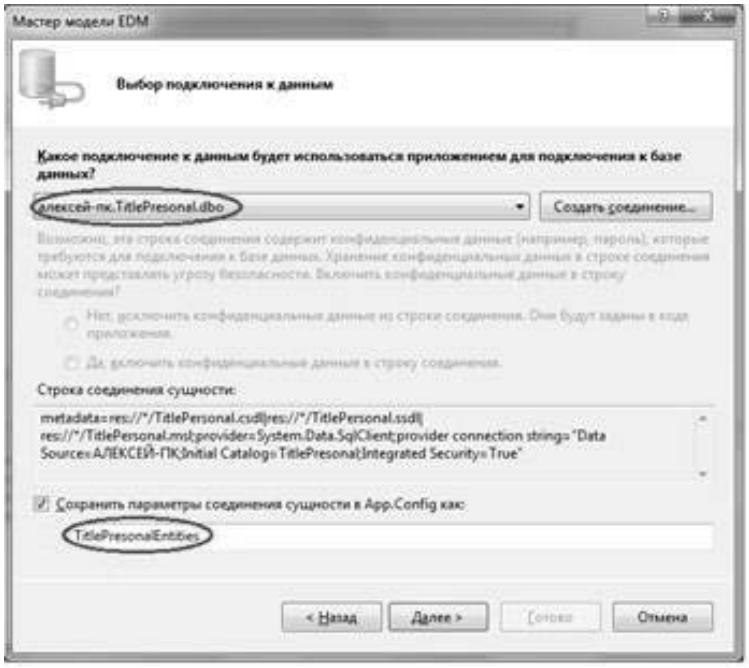

Рисунок 3.3 – Создание соединения с базой данных

В окне «Выбор объектов базы данных» отметим необходимые для приложения таблицы (в данном случае это таблицы Employee и Title) и флаг формирования объектов в единственном или множественном числе (рисунок 3.4).

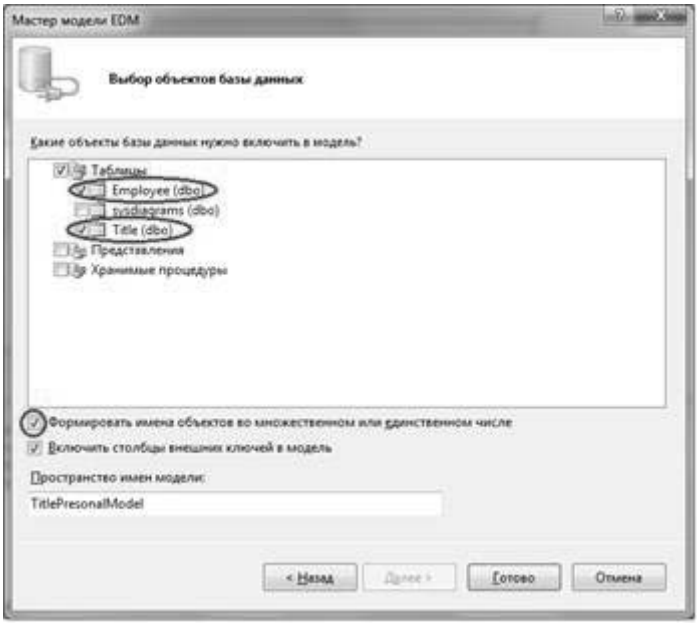

Рисунок 3.4 – Выбор таблиц базы данных

При завершении работы мастера создания EDM-модели в проект будет добавлен файл (в рассматриваемом примере TitleEmployee.edmx – рисунок 3.5), ссылки на необходимые библиотеки и конфигурационный файл.

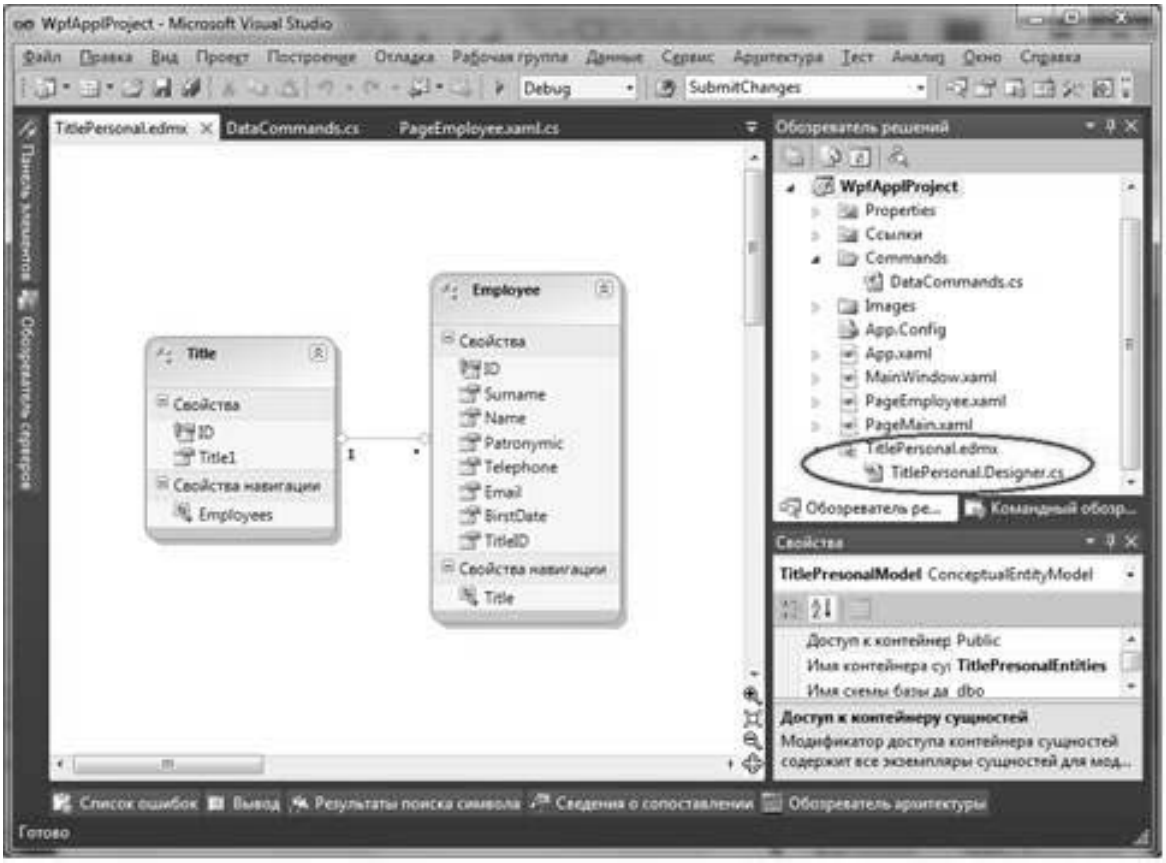

Рисунок 3.5 – Выбор таблиц базы данных

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>
```

Задание

Создать интерфейс WPF-приложения с навигацией по страницам для работы с базой данных. Разрабатываемое приложение должно обеспечивать хранение и обработку указанных данных по варианту. Функции приложения:

– просмотр данных;
– ввод данных;
– редактирование данных;
– удаление данных;
– поиск данных по клиенту.

Вариант 1

Назначение атрибутов таблицы Счет – Account:
– ID – суррогатный ключ;
– TypeID – внешний ключ для связи с таблицей TypeAccount;
– BankID – внешний ключ для связи с таблицей Bank;
– AgreementID – внешний ключ для связи с таблицей Agreement;
– Account – номер инвестиционного счета.
Назначение атрибутов таблицы Тип счета – TypeAccount:
– ID – суррогатный ключ;

– TypeAccount – тип счета.

Назначение атрибутов таблицы Банк – Bank:

– ID – суррогатный ключ;
– NameFull – полное наименование банка;
– NameShort – краткое наименование банка;
– Inn – ИНН банка;
– Bik – БИК банка;
– CorAccount – номер корсчета;
– Account – номер счета;
– City – город.

Назначение атрибутов таблицы Договор – Agreement:

– ID – суррогатный ключ;
– PersonID – внешний ключ для связи с таблицей Person;
– TypeID – внешний ключ для связи с таблицей Type;
– StatusID – внешний ключ для связи с таблицей Status;
– Number – номер договора;
– DataOpen – дата заключения договора;
– DataClouse – дата закрытия договора;
– Note – пояснения.

Вариант 2

Назначение атрибутов таблицы Адрес – Address:

– ID – суррогатный ключ;
– IndexAddress – адресный индекс;
– PersonID – внешний ключ для связи с таблицей Person;
– CountryID – внешний ключ для связи с таблицей Country;
– RegionID – внешний ключ для связи с таблицей Region;
– CityID – внешний ключ для связи с таблицей City;
– Street – наименование улицы;
– Bulding – номер строения, дома;
– Office – номер офиса.

Назначение атрибутов таблицы Город – City:

– ID – суррогатный ключ;
– RegionID – внешний ключ для связи с таблицей Region;
– CountryID – внешний ключ для связи с таблицей Country;
– City – город.

Назначение атрибутов таблицы Регион – Region:

– ID – суррогатный ключ;
– CountryID – внешний ключ для связи с таблицей Country;
– Region – регион.
– Назначение атрибутов таблицы Страна – Country:
– ID – суррогатный ключ;
– CountryFull – полное наименование страны;
– CountryShort – краткое наименование страны.

Вариант 3

Назначение атрибутов таблицы Договор – Agreement:
– ID – суррогатный ключ;
– PersonID – внешний ключ для связи с таблицей Person;
– TypeID – внешний ключ для связи с таблицей Type;
– StatusID – внешний ключ для связи с таблицей Status;
– Number – номер договора;
– DataOpen – дата заключения договора;
– DataClouse – дата закрытия договора;
– Note – пояснения.
Назначение атрибутов таблицы Статус договор – StatusAggrement:
– ID – суррогатный ключ;
– Status – статус договора.
Назначение атрибутов таблицы Клиент – Person:
– ID – суррогатный ключ;
– OrgLicenseID – внешний ключ для связи с таблицей OrgLicense;
– VerietyID – внешний ключ для связи с таблицей Veriety;
– StatusID – внешний ключ для связи с таблицей Status;
– Inn – ИНН клиента;
–Type – тип клиента;
– Shifer – шифр клиента;
– Data – дата регистрации клиента.

Вариант 4

Назначение атрибутов таблицы Клиент – Person:
– ID – суррогатный ключ;
– OrgLicenseID – внешний ключ для связи с таблицей OrgLicense;
– VerietyID – внешний ключ для связи с таблицей Veriety;
– StatusID – внешний ключ для связи с таблицей Status;
– Inn – ИНН клиента;
– Type – тип клиента;
– Shifer – шифр клиента;
– Data – дата регистрации клиента.
Назначение атрибутов таблицы Физическое лицо – Citizen:
– ID – суррогатный ключ;
– DocumentID – внешний ключ для связи с таблицей Document;
– SurName – фамилия клиента;
– Name – имя клиента;
– Patronic – отчество клиента;
– Number – номер документа, удостоверяющего личность;
– Seriy – серия документа, удостоверяющего личность;
– Organ – орган, выдавший документ, удостоверяющий личность;
– Data – дата выдачи документа, удостоверяющего личность.

Назначение атрибутов таблицы Документ – Document:
– ID – суррогатный ключ;
– Document – наименование документа, удостоверяющего личность.

Контрольные вопросы

1 Из каких объектов можно сформировать меню приложения?
2 Какие типы столбцов автоматически генерируются для элемента управления DataGrid?
3 Поясните назначение модели команд WPF.
4 Какие в WPF имеются библиотеки базовых команд?
5 Как можно управлять доступностью команд в приложении WPF?

4 Лабораторная работа № 4. WPF. Разработка корпоративного приложения. Подключение к БД

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

Теоретические сведения

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, буде 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 4.1).

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();
}
```

а) *б*) *в*)

Рисунок 4.1 – Отображение даты в сетке

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", essageBoxButton.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>
```

Задание

К вариантам заданий из лабораторной работы № 2 разработать бизнес-логику для добавления, редактирования и удаления данных.

Контрольные вопросы

1

2 Поясните назначение свойства Content класса Page.

3 В каких контейнерах можно размещать страницы WPF?

4 Какие элементы контроля используются для перехода между страницами приложения?

5 Лабораторная работа № 5. WPF. Разработка корпоративного приложения. Поиск и фильтрация

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

Теоретические сведения

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 5.1.

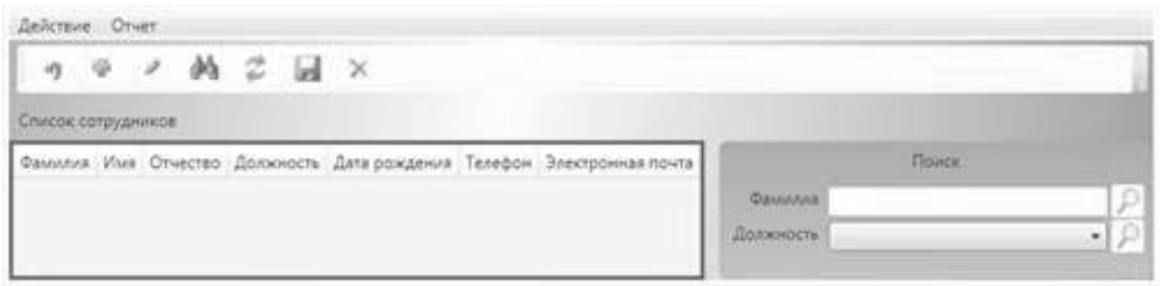

Рисунок 5.1 – Дизайн страницы с элементами контроля для поиска

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 5.2.

Рисунок 5.2 – 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 5.3).

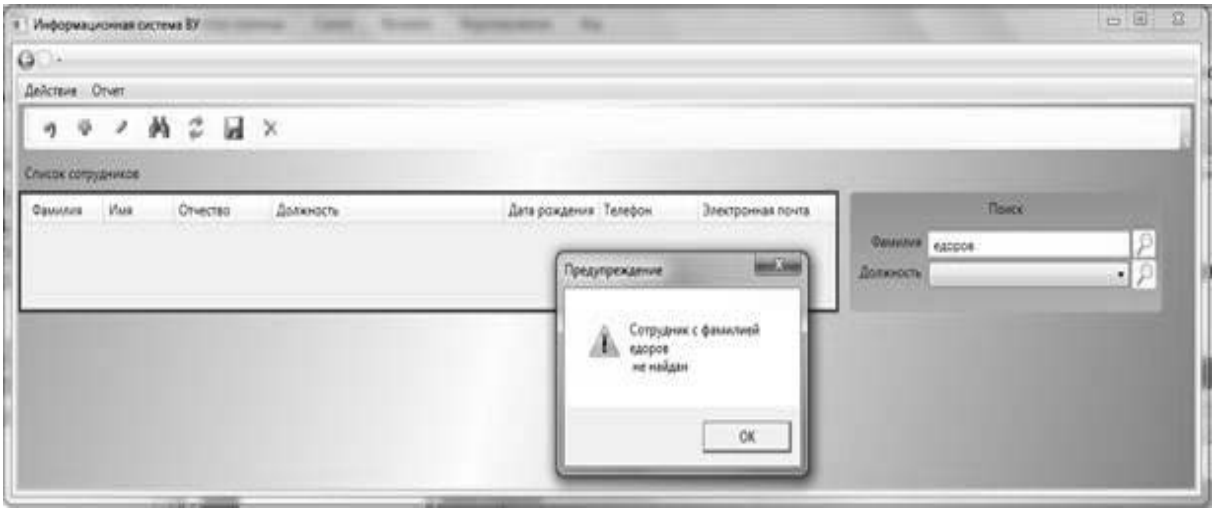

Рисунок 5.3 – Неудачный поиск по фамилии

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.

Задание

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

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?

6 Лабораторная работа № 6. Разработка приложений ASP.NET по шаблону MVC: модель

Цель работы: изучить методику разработки модели MVC-приложения.

Теоретические сведения

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 6.1).

Рисунок 6.1 – 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);
        }
    }
}
```

Контрольные вопросы

1 Назначение представлений в проекте MVC.
2 Как добавить в проект MVC модульные тесты?
3 Как выполнить встраивание карт в представлении?

7 Лабораторная работа № 7. Разработка компонента ASP.NET по шаблону MVC: контроллер

Цель работы: изучить методику разработки контроллера MVC-приложения.

Теоретические сведения

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.

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

Появится диалоговое окно Добавить контроллер.

Необходимо выбрать пустой контролер MVC 5 Controller – Empty.

В поле Имя введите MapsController (рисунок 7.1).

Рисунок 7.1 – Ввод имени контроллера

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

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.

Откройте класс MapsController или переключитесь на него в редакторе.

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.

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

В текстовом поле Имя введите MapsControllerTest.

Нажмите кнопку Добавить.

Visual Studio добавит класс MapsControllerTest в тестовый проект.

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);
        }
    }
}
```

Контрольные вопросы

1 Назначение представлений в проекте MVC.
2 Как добавить в проект MVC модульные тесты?
3 Как выполнить встраивание карт в представлении?

8 Лабораторная работа № 8. Разработка компонента ASP.NET по шаблону MVC: представление

Цель работы: изучить методику разработки представления MVC-приложения.

Теоретические сведения

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!1d150475.33
    22718522!2d27.4515676609572 38!3d53.893056698090 51!2m3!1f0!2f0!3f0!3m
    2!1i1024!2i768!4f13.1!3m3!1m2!1s0x46dbcfd35b1e6ad3%3A0xb61b853ddb57
    0d9!2z0JzQuNC90YHQug!5e0!3m2!1sru!2sby!4v1576007412668!5m2!1sru!2s
    by" 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!1d150475.332271852
2!2d27.4515676609572 38!3d53.893056698090 51!2m3!1f0!2f0!3f0!3m2!1i1024!2i7
68!4f13.1!3m3!1m2!1s0x46dbcfd35b1e6ad3%3A0xb61b853ddb570d9!2z0JzQuNC9
0YHQug!5e0!3m2!1sru!2sby!4v1576007412668!5m2!1sru!2sby";
                break;
             case 'Mogilev':
          a.src =
"https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d150517.908821198
6!2d30.2182230753090 55!3d53.881231044494 37!2m3!1f0!2f0!3f0!3m2!1i1024!2i7
68!4f13.1!3m3!1m2!1s0x46d0521c52844571%3A0xcf85d14239bb73b6!2z0JzQvtCz
0LjQu9GR0LI!5e0!3m2!1sru!2sby!4v1576007883514!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 8.1).

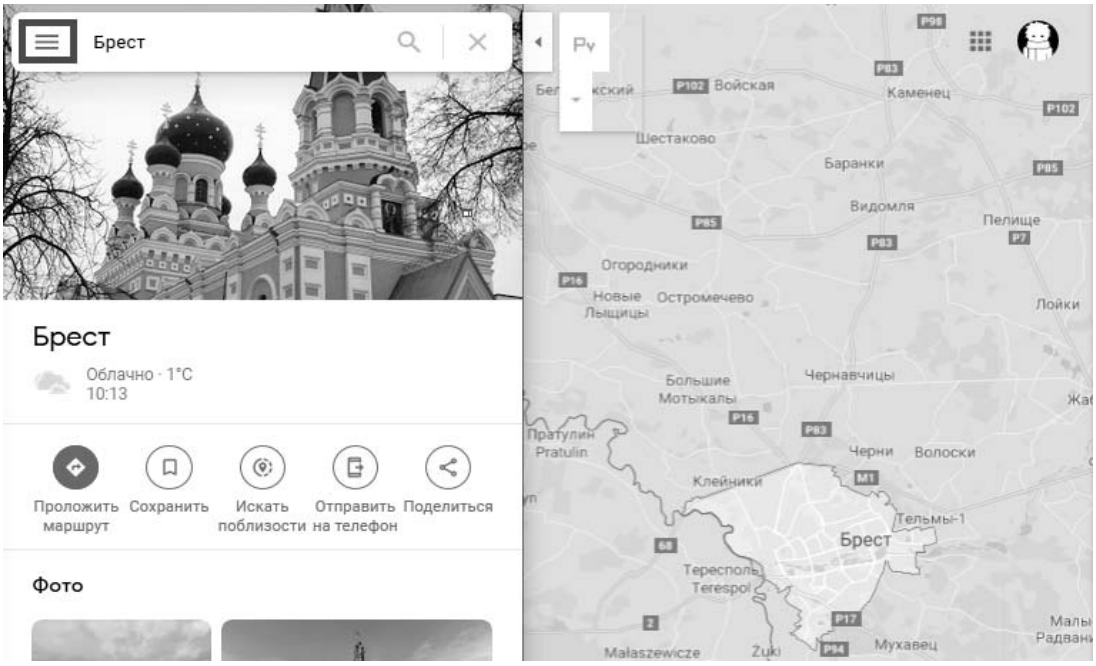

Рисунок 8.1 – Получение iframe с Google Maps Бреста

Находим пункт «Ссылка/код» и кликаем на него (рисунок 8.2).

Выбираем пункт «встраивание карт». Далее можно изменить размер отображаемого фрейма (рисунок 8.3) и отредактировать размер области.

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!1d78449.3134633942
```

```
4!2d23.632848329964006!3d52.088084177221866!2m3!1f0!2f0!3f0!3m2!1i1024!2i
768!4f13.1!3m3!1m2!1s0x47210c0223630975%3A0x4d319ea41f64ae99!2z0JHRgN
C10YHRgg!5e0!3m2!1sru!2sby!4v1576008086154!5m2!1sru!2sby" ;
                break;
            }
    }
```

Теперь при выборе Бреста в списке отобразится нужная нам карта.

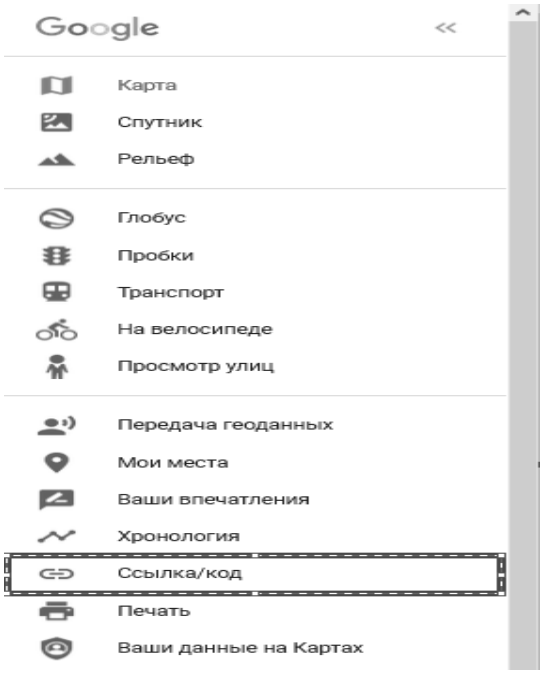

Рисунок 8.2 – Получение iframe с Google Maps для Бреста

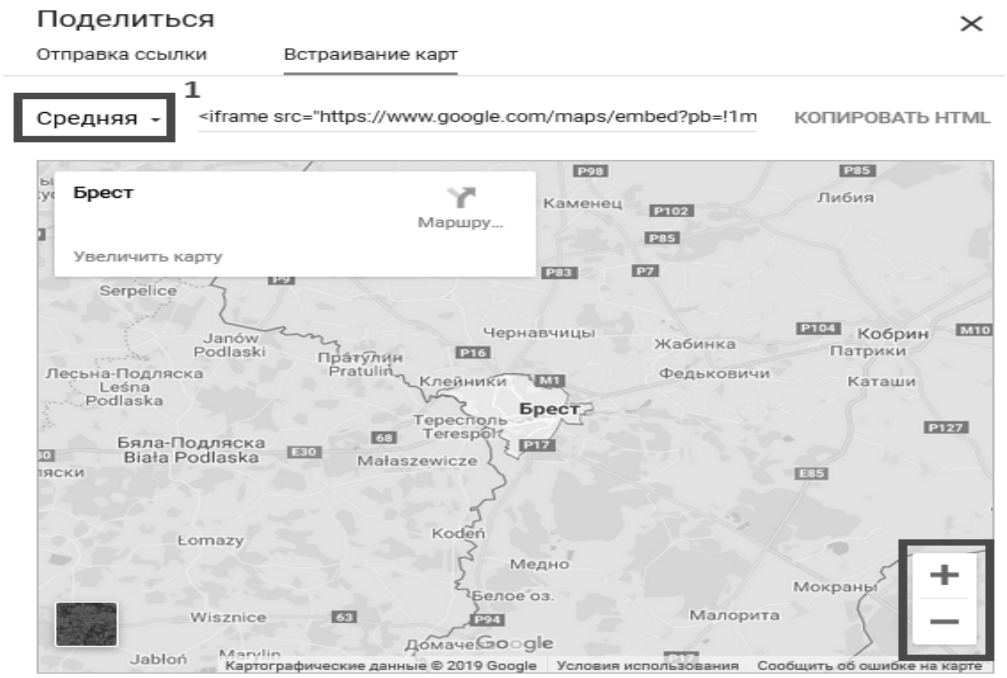

Рисунок 8.3 – Изменение размеров фрейма

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

2 **Фримен, А.** 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.