

*Приложение 4.2 к ОПОП по специальности
09.02.07 Информационные системы и программирование*
Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет
им. Г. И. Носова»
Многопрофильный колледж

**Методические указания
по подготовке к сдаче
демонстрационного экзамена
для обучающихся
специальности 09.02.07 Информационные системы и
программирование**

Квалификация выпускника: программист

Магнитогорск, 2025

ОДОБРЕНО

Предметно-цикловой комиссией Методической комиссией
«Информатика и вычислительная МпК
техника»

Председатель Т.Б.Ремез Протокол № 3 от «19» февраля
Протокол № 5 от «22» января 2025г 2025г

Составители:

Разработчик:

преподаватель ФГБОУ ВО «МГТУ им. Г.И. Носова» Многопрофильный
колледж

В.Д.Тутарова

Методические указания разработаны на основе ФГОС СПО по специальности 09.02.07 Информационные системы и программирование, утвержденного приказом Министерства образования и науки Российской Федерации от 09.12.2016 г. № 1547, оценочных материалов для проведения демонстрационного экзамена КОД 09.02.07-2-2025: Программист.

Методические указания содержат общие положения по проведению демонстрационного экзамена, в полном объеме изложены рекомендации по выполнению заданий демонстрационного экзамена.

СОДЕРЖАНИЕ

1 ОБЩИЕ ПОЛОЖЕНИЯ	4
2 МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПОДГОТОВКЕ К ДЕМОНСТРАЦИОННОМУ ЭКЗАМЕНУ	16
3 ИНФОРМАЦИОННО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ	17

1 ОБЩИЕ ПОЛОЖЕНИЯ

Демонстрационный экзамен направлен на определение уровня освоения выпускником материала, предусмотренного образовательной программой, и степени сформированности профессиональных умений и навыков путем проведения независимой экспертной оценки выполненных выпускником практических заданий в условиях реальных или смоделированных производственных процессов.

Демонстрационный экзамен направлен на контроль освоения следующих основных видов деятельности и соответствующих им общих и профессиональных компетенций:

Вид деятельности (вид профессиональной деятельности)	Перечень оцениваемых ОК, ПК	Перечень оцениваемых умений, навыков (практического опыта)
<i>ИНВАРИАНТНАЯ ЧАСТЬ КОД 09.02.07-2-2025</i>		
Разработка, администрирование и защита баз данных	ПК 11.3: Разрабатывать объекты базы данных в соответствии с результатами анализа предметной области	Умение: создавать объекты баз данных в современных СУБД
		Практический опыт: использовать средства заполнения базы данных
	ПК 11.4: Реализовывать базу данных в конкретной системе управления базами данных	Практический опыт: работать с объектами базы данных в конкретной системе управления базами данных
Разработка модулей программного обеспечения для компьютерных систем	ПК 1.1: Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием	Практический опыт: разрабатывать алгоритм решения поставленной задачи и реализовывать его средствами автоматизированного проектирования
	ПК 1.2: Разрабатывать программные модули	Практический опыт: разрабатывать код

	в соответствии с техническим заданием	программного продукта на основе готовой спецификации на уровне модуля
	ПК 1.3: Выполнять отладку программных модулей с использованием специализированных программных средств	Умение: выполнять отладку и тестирование программы на уровне модуля
Сопровождение и обслуживание программного обеспечения компьютерных систем	ПК 4.3: Выполнять работы по модификации отдельных компонент программного обеспечения в соответствии с потребностями заказчика	Умение: разрабатывать и настраивать программные модули программного продукта
		Практический опыт: модифицировать отдельные компоненты программного обеспечения в соответствии с потребностями заказчика
	ОК 02: Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности	Умение: выделять наиболее значимое в перечне информации
Осуществление интеграции программных модулей	ПК 2.2: Выполнять интеграцию модулей в программное обеспечение	Умение: использовать выбранную систему контроля версий
		Практический опыт: интегрировать модули в программное обеспечение
	ОК 01: Выбирать	Умение: распознавать

	способы решения задач профессиональной деятельности применительно к различным контекстам	задачу и/или проблему в профессиональном и/или социальном контексте
--	--	---

Для проведения демонстрационного экзамена составляется расписание экзамена и консультаций.

Демонстрационный экзамен по специальности 09.02.07 Информационные системы и программирование проводится на профильном уровне.

Демонстрационный экзамен профильного уровня проводится по решению образовательной организации на основании заявлений выпускников на основе требований к результатам освоения образовательных программ среднего профессионального образования, установленных в соответствии с ФГОС СПО.

Комплект оценочной документации включает комплекс требований для проведения демонстрационного экзамена, перечень оборудования и оснащения, расходных материалов, средств обучения и воспитания, план застройки площадки демонстрационного экзамена, требования к составу экспертных групп, инструкции по технике безопасности, а также образцы заданий.

Задание демонстрационного экзамена включает комплексную практическую задачу, моделирующую профессиональную деятельность и выполняемую в режиме реального времени.

5.2 Типовое задание для демонстрационного экзамена профильного уровня

5.2.1 Структура и содержание типового задания

Демонстрационный экзамен профильного уровня проводится с использованием единых оценочных материалов, включающих в себя конкретные комплекты оценочной документации (КОД), варианты заданий и критерии оценивания, разрабатываемых оператором. Комплект оценочной документации приведен в <https://bom.firpo.ru/file/public/77752/%D0%9A%D0%9E%D0%94%2009.02.07-2-2025%20%D0%A2%D0%BE%D0%BC%201.pdf>

Задание состоит из 4 модулей:

Модуль 1. Разработка, администрирование и защита баз данных
Задание:

Компания занимается производством и реализует свою продукцию через партнеров, которые доставляют продукцию компании до конечных потребителей. Для эффективного взаимодействия с партнерами и контроля их работы требуется система, позволяющая обрабатывать всю информацию в цифровом формате.

Разработать подсистему для работы с партнерами компании, обеспечивающую следующий функционал:

- просмотр списка партнеров;
- добавление/редактирование данных о партнере;
- просмотр истории реализации продукции партнером.

На основе описания предметной области необходимо создать базу данных в выбранной СУБД для разрабатываемой системы. Обязательна 3 нормальная форма с обеспечением ссылочной целостности. При разработке базы данных обратить внимание на согласованную схему именования, создать необходимые первичные и внешние ключи.

На данном этапе нет необходимости воспроизводить все сущности предметной области, достаточно создать таблицы, поля с подходящими типами данных и связи, непосредственно относящиеся к разрабатываемой подсистеме и ее функционалу.

Получить ER-диаграмму средствами СУБД: ER-диаграмма должна быть представлена в формате PDF и содержать таблицы, связи между ними, атрибуты и ключи (типами данных на данном этапе можно пренебречь).

Заказчик системы предоставил файлы с данными (с пометкой import в ресурсах) для переноса в новую систему. Необходимо подготовить данные файлов для импорта и загрузить в разработанную базу данных.

Сохранить полученные результаты: создать скрипт БД.

Необходимые приложения: Приложение 1: Описание предметной области.

Модуль 2. Разработка модулей программного обеспечения для компьютерных систем

Задание:

Сформировать алгоритм подсистемы для работы с партнерами.

Разработать алгоритм функции расчета индивидуальной скидки для партнера.

Алгоритмы реализовать в виде кода программного продукта средствами любой среды разработки и языка программирования из доступных.

Компоненты системы должны иметь единый согласованный внешний вид, соответствующий руководству по стилю, представленному в Приложении 2. Заголовок окна (страницы) должен соответствовать назначению. Следует установить иконку приложения, если это реализуемо в рамках платформы, и логотип компании на главной форме, из ресурсов.

Оформление кода: идентификаторы должны отражать их назначение и соответствовать соглашению об именовании и стилю CamelCase (для C# и Java), snake_case (для Python) и <https://its.1c.ru/db/v8std#browse:13:-1:31> (для 1C). Допустимо использование не более одной команды в строке.

Разработать программный модуль для учета партнеров. Необходимо реализовать вывод списка партнеров, информация о которых хранятся в базе данных, согласно предоставленному макету:

Тип Наименование партнера	10%
Директор	
+7 223 322 22 32	
Рейтинг: 10	

Тип Наименование партнера	10%
Директор	
+7 223 322 22 32	
Рейтинг: 10	

Тип Наименование партнера	10%
Директор	
+7 223 322 22 32	
Рейтинг: 10	

Величина скидки для партнера рассчитывается на основании продажи продукции за весь период работы. Скидка зависит от общего количества реализованной партнером продукции и составляет: до 10000 – 0%, от 10000 – до 50000 – 5%, от 50000 – до 300000 – 10%, более 300000

– 15%.

Созданную базу данных подключить к приложению работы с партнерами, реализующему необходимый функционал. Список партнеров на главной форме должен отображать информацию из базы данных.

Выполнить отладку и тестирование модуля для проверки функциональности: приложение должно корректно работать и не должно происходить аварийного завершения работы.

Необходимые приложения: Приложение 1: Описание предметной области Приложение 2: Руководство по стилю.

Модуль 3. Сопровождение и обслуживание программного обеспечения компьютерных систем;

Задание:

Разработать интерфейс программного модуля для работы с партнерами.

Реализовать последовательный пользовательский интерфейс, позволяющий перемещаться между существующими окнами (страницами) в приложении (в том числе обратно, например, с помощью кнопки «Назад»). Обеспечить соответствующий заголовок на каждом окне (странице) приложения.

Реализовать обработку исключительных ситуаций в приложении. Необходимо уведомлять пользователя о совершаемых им ошибках или о запрещенных в рамках задания действиях, предупреждать о неотвратимых операциях. Окна сообщений соответствующих типов (например, ошибка, предупреждение, информация) должны отображаться с соответствующим заголовком и пиктограммой. Текст сообщения должен быть полезным и информативным, содержать полную информацию о совершенных ошибках пользователя и порядок действий для их исправления. Также можно использовать визуальные подсказки для пользователя при вводе данных.

Необходимо использовать комментарии для пояснения неочевидных фрагментов кода. Комментарии должны присутствовать только в местах, которые требуют дополнительного пояснения.

Реализовать функции добавления/редактирования данных партнера в новом окне (странице) – форме для добавления/редактирования партнера. Переходы на эту форму должны быть реализованы из главной

формы списка партнеров: для редактирования – при нажатии на конкретный элемент, для добавления – при нажатии кнопки.

На форме для добавления/редактирования партнера должны быть предусмотрены следующие поля: наименование, тип партнера (выпадающий список), рейтинг, адрес, ФИО директора, телефон и email компании.

Рейтинг партнера должен быть целыми неотрицательным числом.

При открытии формы для редактирования все поля выбранного объекта должны быть подгружены в соответствующие поля из базы данных, а таблица заполнена актуальными значениями.

После добавления/редактирования партнера данные в окне списка партнеров должны быть обновлены.

Необходимые приложения: Приложение 1: Описание предметной области Приложение 2: Руководство по стилю.

Модуль 4. Осуществление интеграции программных модулей

Задание:

Добавить функционал в систему для работы с партнерами компании согласно требованиям заказчика. Необходимо осуществить вывод истории реализации продукции партнером с указанием наименования продукции, количества и даты продажи.

Выполнить интеграцию модуля вывода истории реализации продукции партнером: обеспечить соответствие стилю приложения, единый для системы согласованный внешний вид. Реализовать переход на данное окно (страницу), получить историю реализации продукции для конкретного партнера.

С целью обеспечить одинаковый расчет количества материала, требуемого для производства продукции, необходимо разработать метод.

Метод должен принимать идентификатор типа продукции, идентификатор типа материала, количество получаемой продукции – целые числа, параметры продукции (два параметра) – вещественные, положительные числа, а возвращать целое число – количество необходимого материала с учетом возможного брака материала.

Метод должен рассчитывать целое количество материала, необходимого для производства указанного количества продукции, учитывая возможный брак материала. Количество необходимого материала на одну единицу продукции рассчитывается как произведение

параметров продукции, умноженное на коэффициент типа продукции. Кроме того, нужно учитывать процент брака материала в зависимости от его типа: с учетом возможного брака материала необходимое количество материала должно быть увеличено. Коэффициент типа продукции и процент брака – вещественные числа.

Если в качестве параметров метода будут указаны несуществующие типы продукции, материалов или другие неподходящие данные, то метод должен вернуть -1.

Необходимо загрузить исходный код метода расчета материала в отдельный репозиторий с названием, совпадающим с названием проекта.

Все практические результаты должны быть переданы путем загрузки файлов на предоставленный репозиторий системы контроля версий.

Практические результаты:

- исходный код приложения (структура с файлами, не архив);
- исполняемые файлы;
- файл скрипта базы данных;
- прочие графические/текстовые файлы.

Результаты работ загружать в рамках выполнения задания модуля.

Необходимые приложения: Приложение 1: Описание предметной области Приложение 2: Руководство по стилю.

5.2.2 Оснащение рабочего места для проведения демонстрационного экзамена по типовому заданию

Материально-техническая база соответствует инфраструктурному листу КОД 09.02.07-2-2025: Программист.

5.3 Критерии оценки выполнения задания демонстрационного экзамена

Процедура оценивания результатов выполнения заданий демонстрационного экзамена осуществляется членами экспертной группы по 100-балльной системе в соответствии с требованиями комплекта оценочной документации.

Распределение баллов по критериям оценивания демонстрационного экзамена профильного уровня представлена в таблице.

№ п/п	Модуль задания (вид деятельности, вид профессиональной деятельности)	Критерий оценивания
1	Разработка, администрирование и защита баз данных	Разработка объектов базы данных в соответствии с результатами анализа предметной области
		Реализация базы данных в конкретной системе управления базами данных
2	Разработка модулей программного обеспечения для компьютерных систем	Формирование алгоритмов разработки программного обеспечения модулей в соответствии с техническим заданием
		Разработка программных модулей в соответствии с техническим заданием
		Выполнение отладки программных модулей с использованием специализированных программных средств
3	Сопровождение и обслуживание программного обеспечения компьютерных систем	Выполнение работ по модификации отдельных компонент программного обеспечения в соответствии с потребностями заказчика
		Использование современных средств поиска и интерпретации информации и информационных технологий для выполнения задач профессиональной деятельности
4	Осуществление интеграции программных модулей	Выполнение интеграции модулей в программное обеспечение
		Выбор способов решения задач профессиональной деятельности применительно к различным контекстам
ИТОГО		

Необходимо осуществить перевод количества баллов в оценки «отлично», «хорошо», «удовлетворительно», «неудовлетворительно». Перевод полученного количества баллов в оценки осуществляется государственной экзаменационной комиссией с обязательным присутствием главного эксперта.

Перевод баллов в оценку может быть осуществлен на основе таблицы (профессиональный уровень):

Оценка ГИА	«2»	«3»	«4»
Отношение полученного количества баллов к	0,00 - 19,99%	20,00 – 39,99%	40,00 – 69,99%

максимально возможному процентах)	(в			
---	----	--	--	--

Баллы выставляются в протоколе проведения демонстрационного экзамена, который подписывается каждым членом экспертной группы и утверждается главным экспертом после завершения экзамена для экзаменационной группы.

При выставлении баллов присутствует член ГЭК, не входящий в экспертную группу, присутствие других лиц запрещено.

Подписанный членами экспертной группы и утвержденный главным экспертом протокол проведения демонстрационного экзамена далее передается в ГЭК для выставления оценок по итогам ГИА.

Оригинал протокола проведения демонстрационного экзамена передается на хранение в образовательную организацию в составе архивных документов.

Статус победителя, призера чемпионатов профессионального мастерства, проведенных Агентством (Союзом «Агентство развития профессиональных сообществ и рабочих кадров «Молодые профессионалы (Ворлдскиллс Россия)»)» либо международной организацией «WorldSkills International», в том числе «WorldSkills Europe» и «WorldSkills Asia», и участника национальной сборной России по профессиональному мастерству по стандартам «Ворлдскиллс» выпускника по профилю осваиваемой образовательной программы среднего профессионального образования засчитывается в качестве оценки «отлично» по демонстрационному экзамену в рамках проведения ГИА по данной образовательной программе среднего профессионального образования.

2 МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПОДГОТОВКЕ К ДЕМОНСТРАЦИОННОМУ ЭКЗАМЕНУ

Модуль 1 Разработка модулей программного обеспечения для компьютерных систем

ПРОЕКТИРОВАНИЕ USE CASE ДИАГРАММЫ. ОПРЕДЕЛЕНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ СИСТЕМЫ

Глоссарий

Для успешного освоения материала рекомендуем вам изучить следующие понятия:

Use Case Diagram

Диаграмма вариантов использования – диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Предметная область – часть реального мира, рассматриваемая в пределах данного контекста.

UML – Unified Modeling Language (унифицированный язык моделирования). Язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

ТЗ – техническое задание. Документ, содержащий требования заказчика к объекту закупки, определяющие условия и порядок ее проведения для обеспечения государственных или муниципальных нужд, в соответствии с которым осуществляются поставка товара, выполнение работ, оказание услуг и их приемка.

Actor – актер (Use Case). Роль объекта вне системы, который прямо взаимодействует с ее частью — конкретным элементом.

Use Case – вариант использования (прецедент). Описание поведения системы, когда она взаимодействует с кем-то (или чем-то) из внешней среды.

Hot Keys – горячие клавиши. Комбинация клавиш на клавиатуре, нажатие на которые позволяет выполнять различные действия в операционной системе и программах, не прибегая к использованию мыши и не вызывая меню действий.

Конспект

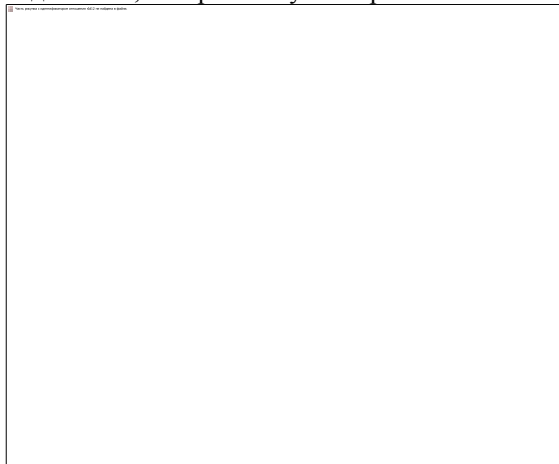
Анализ предметной области и проектирование

Анализ предметной области и проектирование являются первыми этапами в жизненном цикле создания программного решения. Одним из

результатов этого этапа является диаграмма вариантов использования (Use Case), описывающая основные группы пользователей системы и варианты ее использования.

Предметная область — это часть реального мира, данные и особенности которой будут отражены в разрабатываемом программном решении. Например, в качестве предметной области можно выбрать бухгалтерию какого-либо предприятия, отдел кадров, банк, магазин и т. д. Предметная область бесконечна и содержит как важные понятия и данные, так и малозначащие или вообще ничего не значащие данные. Так, если в качестве предметной области выбрать учет товаров на складе, то понятия «накладная» и «счет-фактура» являются важными, а то, что сотрудница, принимающая накладные, имеет двоих детей — это для учета товаров неважно. Однако с точки зрения отдела кадров данные о наличии детей являются важными. Таким образом, значимость данных зависит от выбора предметной области.

В рамках курса для демонстрации основных модулей было выбрано туристическое агентство. Давайте проанализируем вводное описание и определим данные, которые действительно необходимы для нашей системы. Перед вами описание предметной области (важные данные мы будем отмечать маркерами: красным — роль пользователя, желтым — важные действия, которые могут совершать пользователи).



Итак, мы выделили:

Администратор — создание новых туров и редактирование существующих.

Менеджер — регистрация клиента в системе.

Менеджер — подбор тура для клиента + 4 дополнительных

действия.

Менеджер — регистрация заявки на клиента + включение в заявку дополнительных услуг.

Клиент и менеджер — отслеживание актуальной информации по заявке.

Клиент — сохранение ваучеров на свое устройство.

Менеджер — подача запроса на формирование ваучеров.

Клиент — возможность оставить отзыв об отеле.

Ревью возможностей MS Visio для создания диаграмм

После определения требований переходим к этапу проектирования. В ходе проектирования архитектором создается проектная документация, включающая:

1. текстовые описания;
2. диаграммы;
3. модели будущей программы.

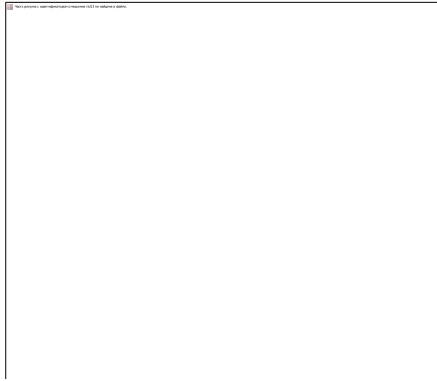
Для этого используется графический язык для визуализации, описания параметров, конструирования и документирования различных систем UML. Для визуализации модели существуют различные типы диаграмм:

1. Диаграмма вариантов использования (use case diagram);
2. Диаграмма классов (class diagram);
3. Диаграмма состояний (statechart diagram);
4. Диаграмма последовательности (sequence diagram).

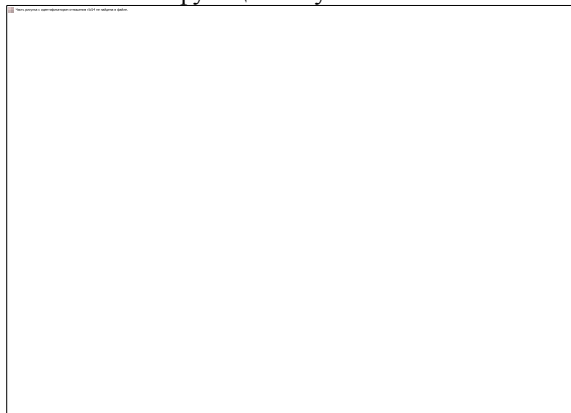
Остановимся на диаграмме вариантов использования. Она достаточно проста, это позволяет использовать ее для согласования технического задания с заказчиком

Создание диаграммы для турагенства

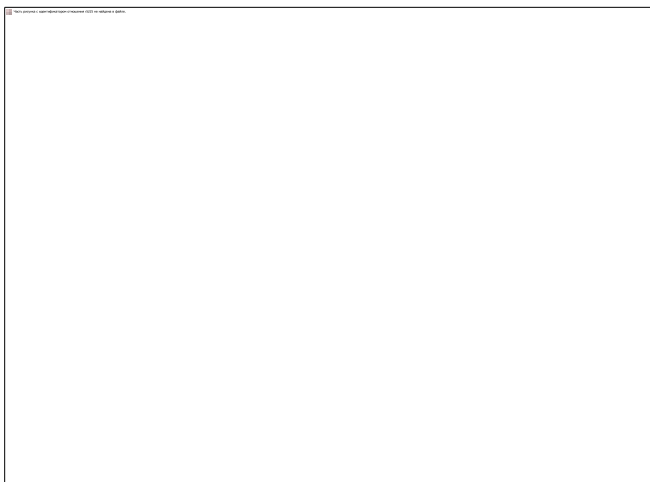
1. Определение рамок системы согласно заданию. Для этого используем элемент subsystem, там будут располагаться прецеденты (функционал, реализуемый системой).



2. Определение основных групп пользователей (ролей) и размещение на диаграмме. Это те, кто будет использовать систему, и в нашем случае, как следует из тех. задания, – это клиент, менеджер и администратор. После размещения будет наглядно видно, что разные группы пользователей имеют доступ только к определённому функционалу.



3. Определение вариантов использования (прецедентов), размещение их на диаграмме.



А) Для администратора:

- создать новый тур;
- редактировать существующий тур.

Б) Для менеджера:

- зарегистрировать клиента;
- подобрать тур: выбрать даты тура, указать предпочтения клиента, указать границы стоимости, выбрать отель;
- зарегистрировать заявку: выбрать дополнительные услуги;
- сформировать ваучер;

В) Для клиента и менеджера:

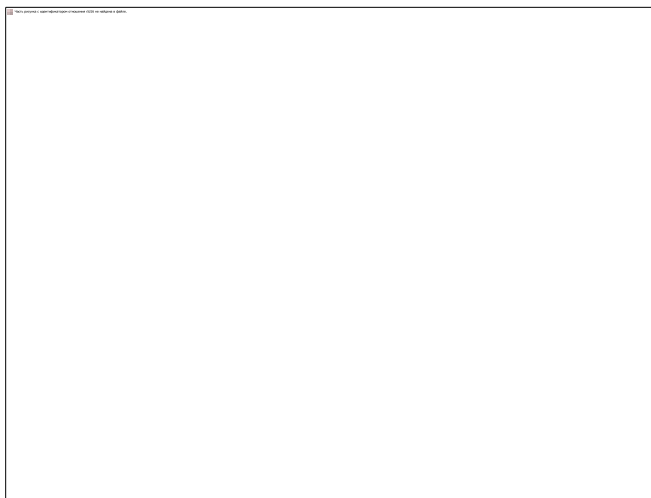
- получить информацию по заявке;

Г) Для клиента:

- сохранить ваучер на устройство;
- оставить отзыв об отеле;

Разграничение прецедентов между актерами и размещение отношений

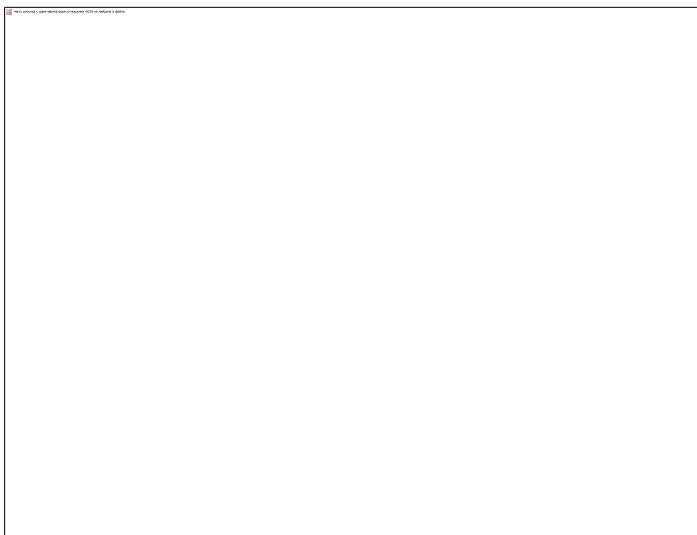
Отношение ассоциации – отражает возможность использования актером прецедента.



Отношение включения – поведение одного прецедента включается в другой в качестве составного, причем дополняемый вариант использования не сможет выполняться без основного.



Отношение расширения – отражает возможное присоединение одного использования к другому, при этом расширяющий вариант использования выполняется лишь при определенных условиях и не является обязательным для выполнения основного прецедента.

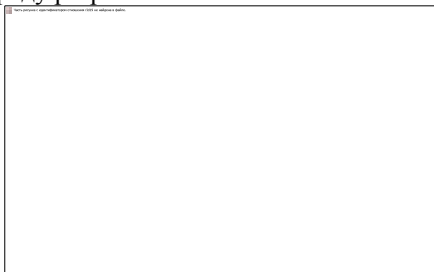


СОЗДАНИЕ КАРКАСА ПРИЛОЖЕНИЯ. СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ СТИЛЕЙ

Использование Windows Presentation Foundation (WPF) для создания интерактивных настольных приложений

Создание нового проекта

1. Запустить среду разработки Visual Studio



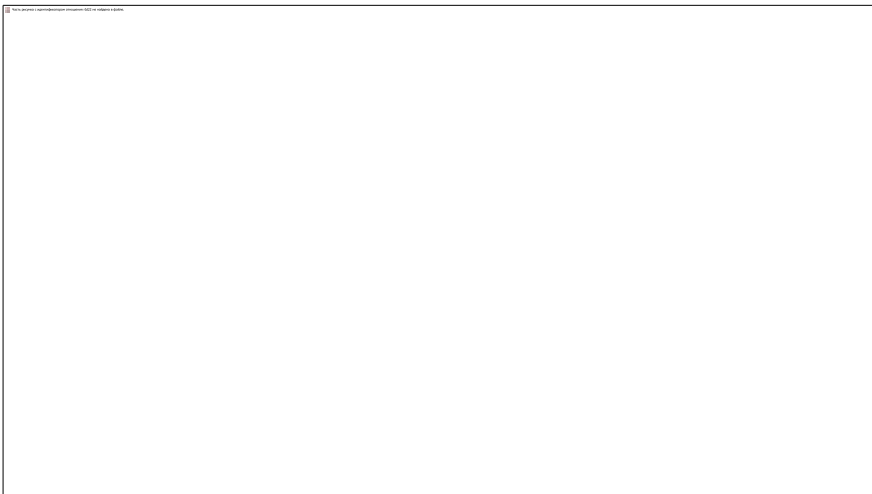
2. Выбрать «Create new project»

Ссылка на файл: [https://www.microsoft.com/ru-ru/evalcenter/evaluate-office-365](#)

3. Выбрать «WPF App»

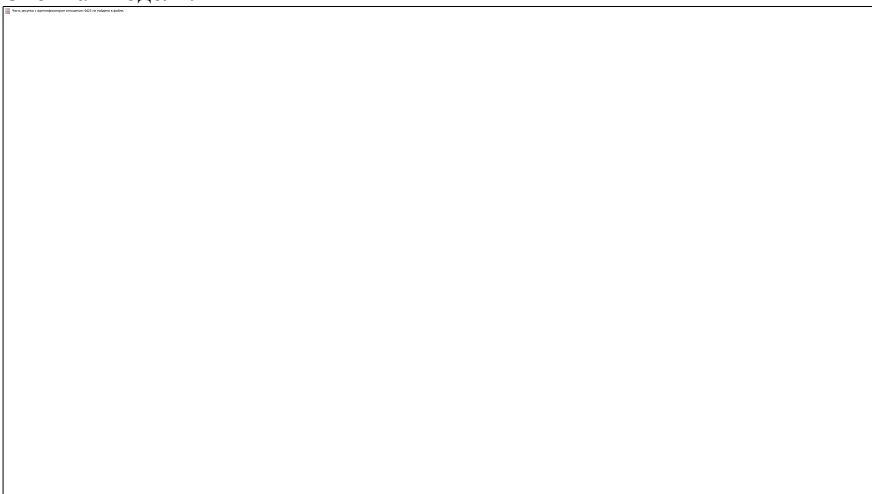
Ссылка на файл: [https://www.microsoft.com/ru-ru/evalcenter/evaluate-office-365](#)

4. Заполнить графу «Project name»

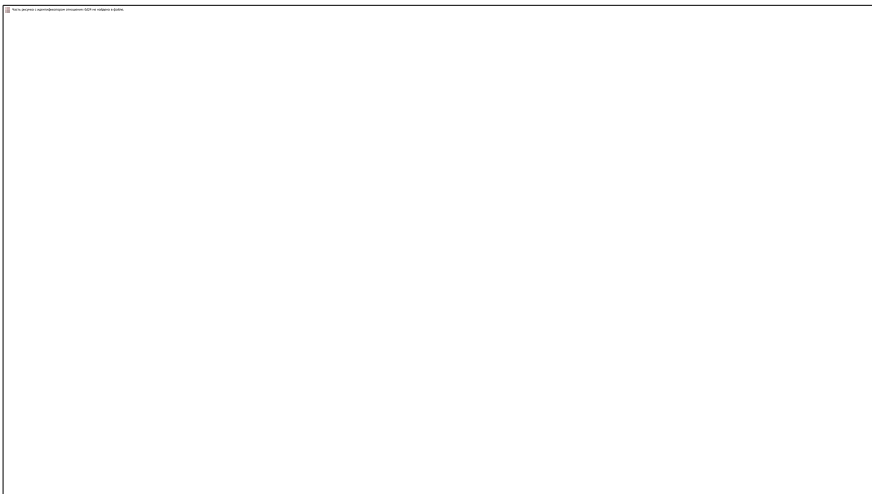


При разработке интерфейсов разработчик может использовать две модели: оконную или страничную (в настоящее время используют чаще).

Оконная модель:

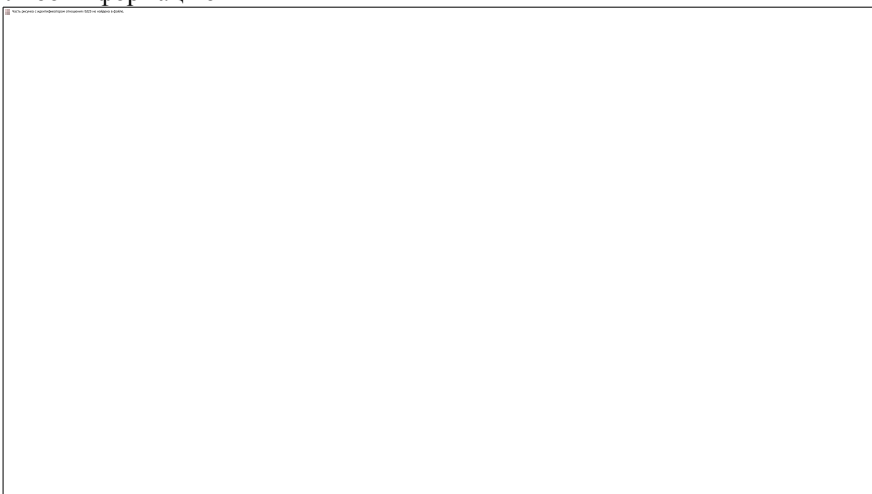


Страничная модель

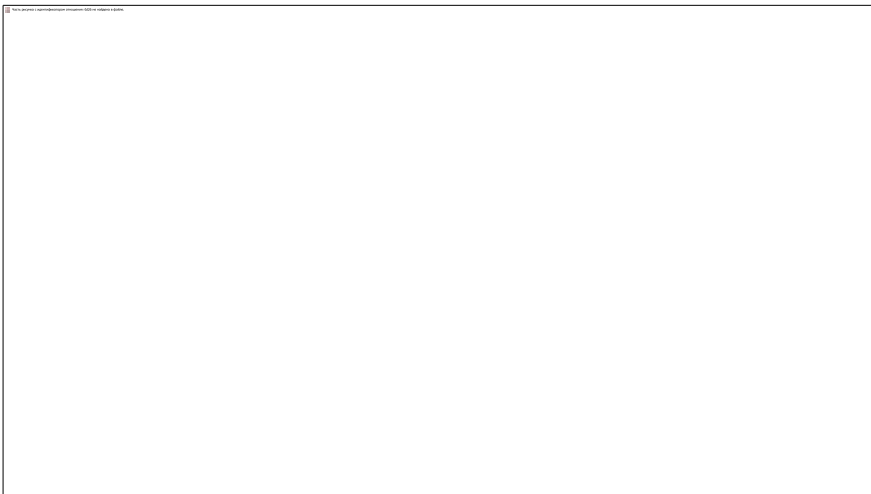


Элементы управления в приложении

Интерфейс состоит из элементов управления, которые непосредственно взаимодействуют с пользователем или отображают какую-либо информацию



Доступные элементы управления находятся на панели Toolbox



Использование Windows Presentation Foundation (WPF) для создания интерактивных настольных приложений

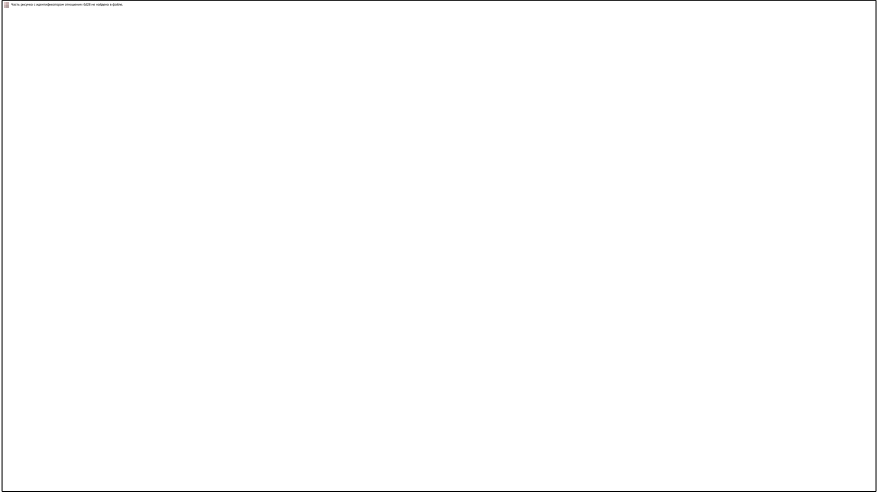
Контейнеры компоновки

1. Grid

Наиболее мощный и часто используемый контейнер, похожий на таблицу. Он содержит столбцы и строки, количество которых можно задать. Для определения строк используется свойство `RowDefinition`,



а для столбцов — `ColumnDefinition`



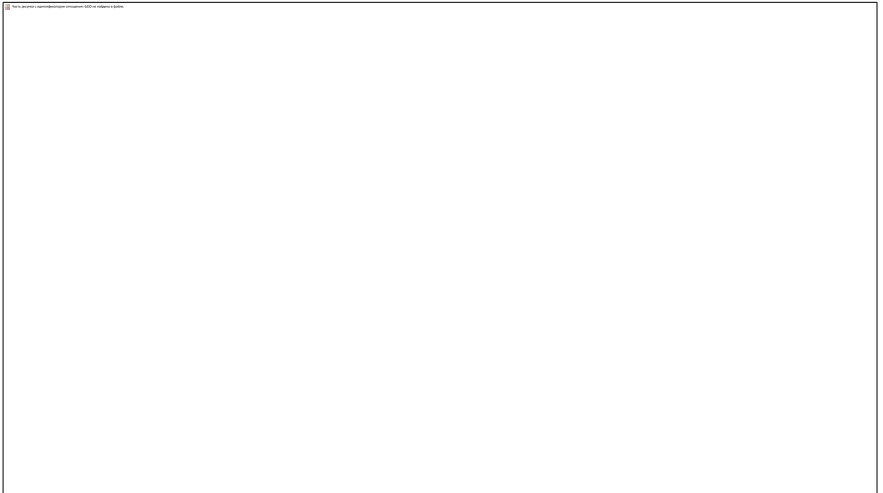
Важно

Для привязки элемента управления к конкретной ячейке необходимо использовать свойства `Grid.Column` и `Grid.Row`, причем нумерация строк и столбцов начинается с нуля

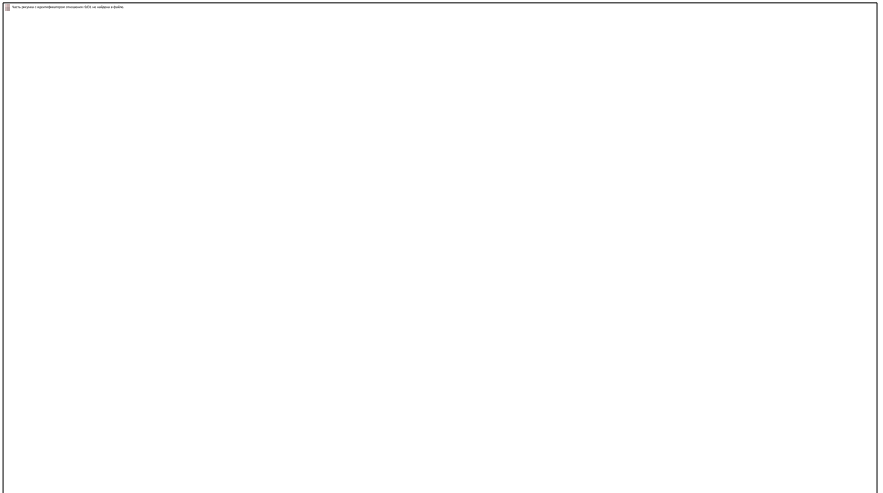


2. StackPanel

Позволяет размещать элементы управления поочередно друг за другом

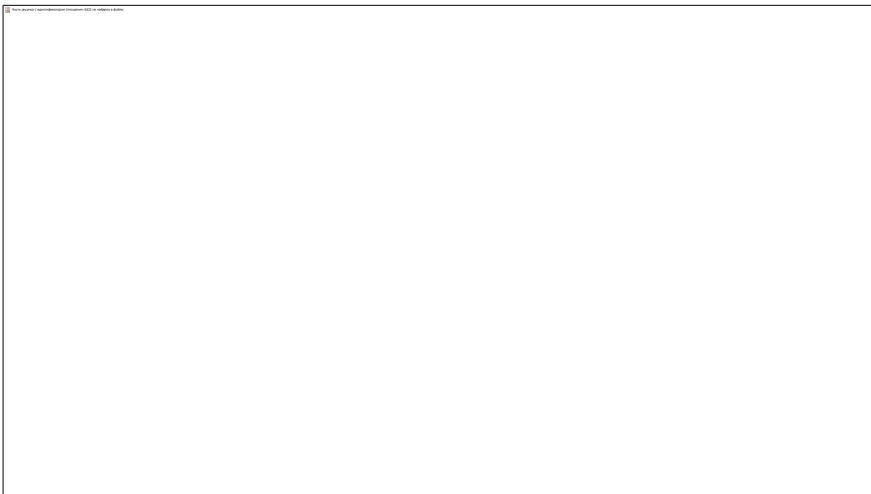


Также существует возможность выбора ориентации размещения с помощью свойства `Orientation`.



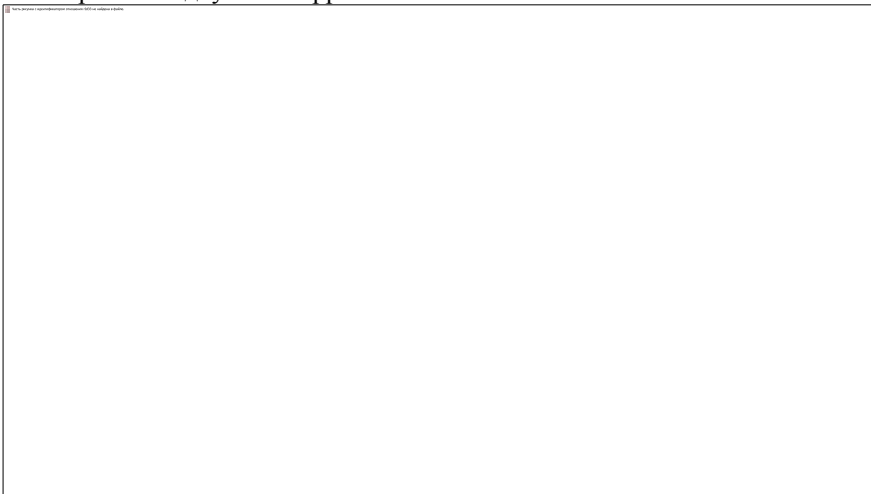
Стилизация приложения

Перед стилизацией разметим окно приложения с помощью контейнера `Grid`

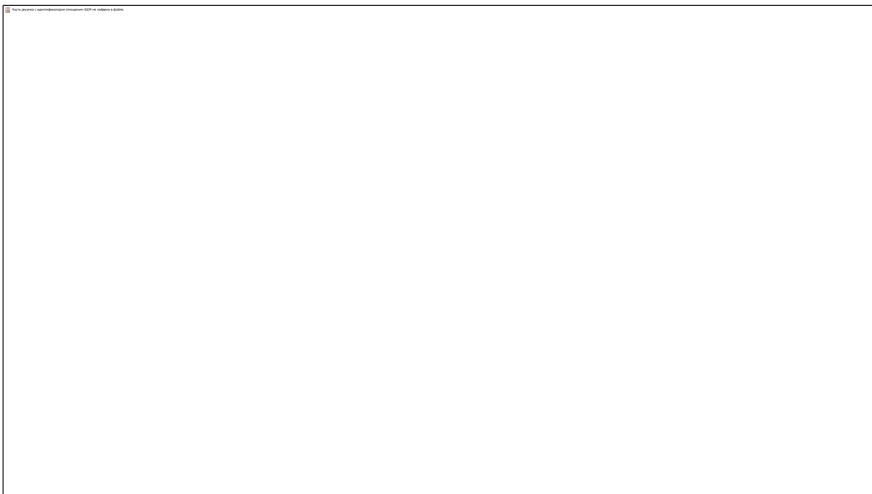


Добавим логотип приложения в Resources

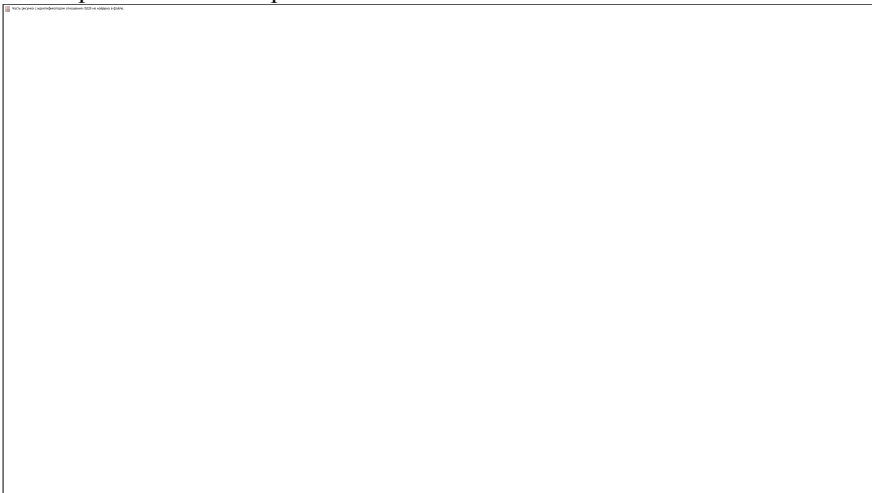
1. Выбрать вкладку ToursApp



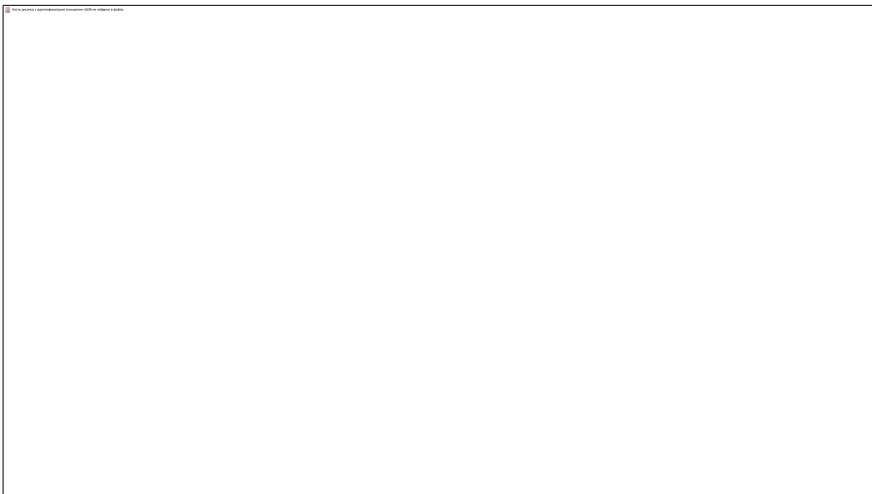
2. Открыть вкладку Resources



3. Выбрать список изображений



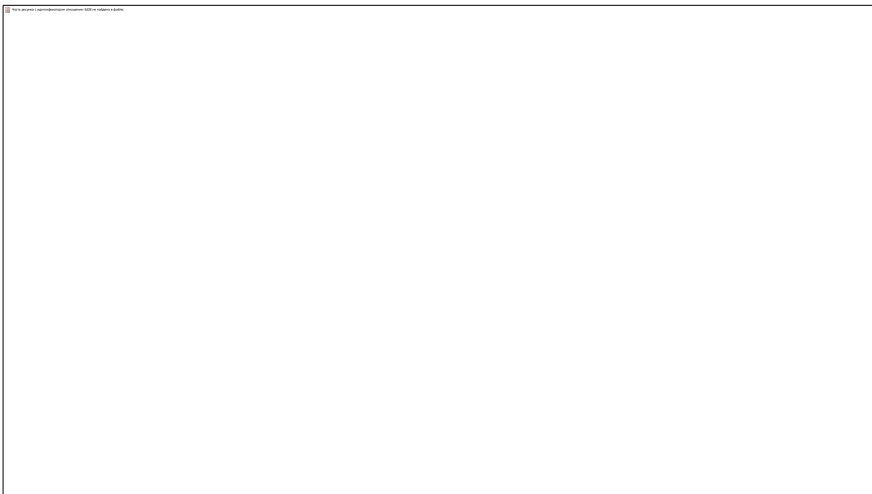
4. Перетащить логотип в открывшееся окно



5. Установить значение поля Build Action – Resource



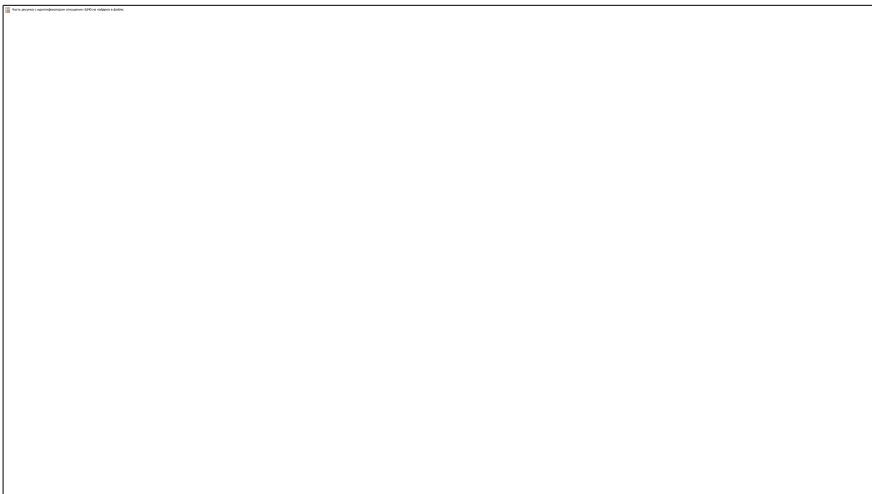
Для отображения логотипа используем элемент Image



Установим заголовок приложения

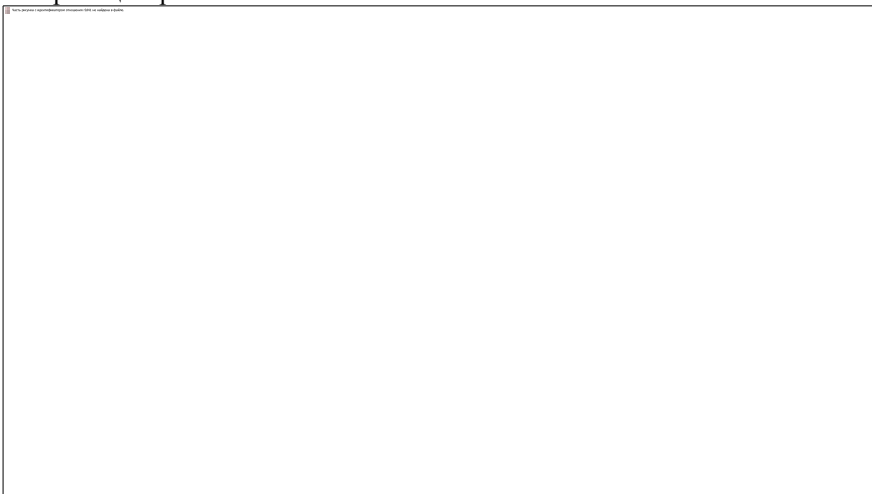


Установим цвета для верхней и нижней частей



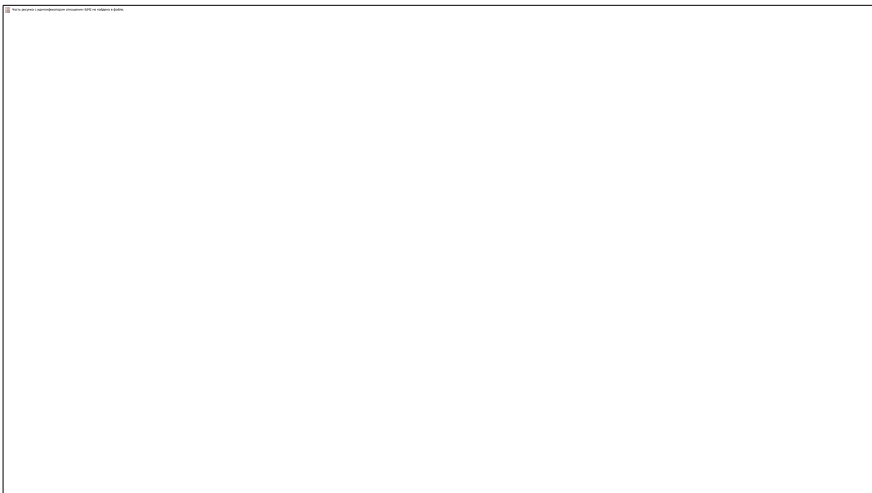
Создание навигации

Для оконной навигации основным элементом является страница — Page, которая должна находиться в каком-либо контейнере. Для этого в основном окне приложения разместим элемент Frame, где будут собраны страницы приложения

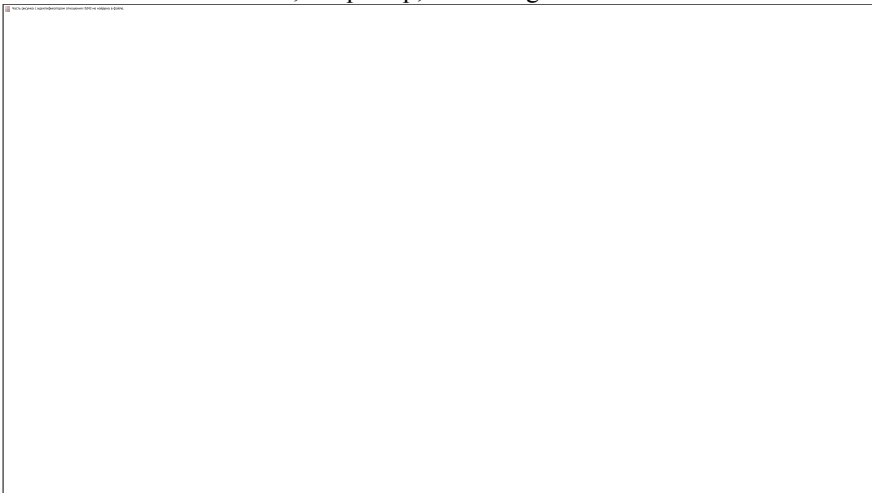


Далее создадим страницу, которая будет отображаться при первом запуске приложения, и вторую страницу для тестирования навигации между ними

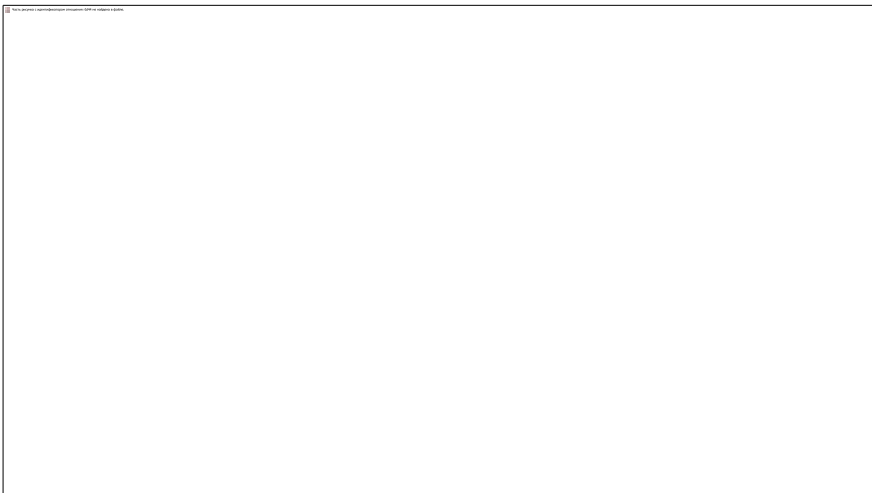
1. Правой кнопкой жмем на название проекта — Add — Page



2. Указываем ее название, например, HotelsPage



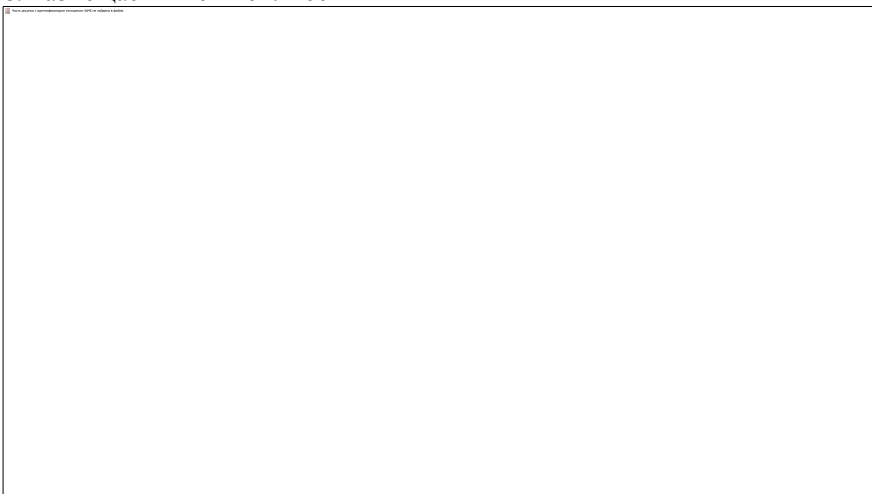
3. Размещаем на странице одну кнопку



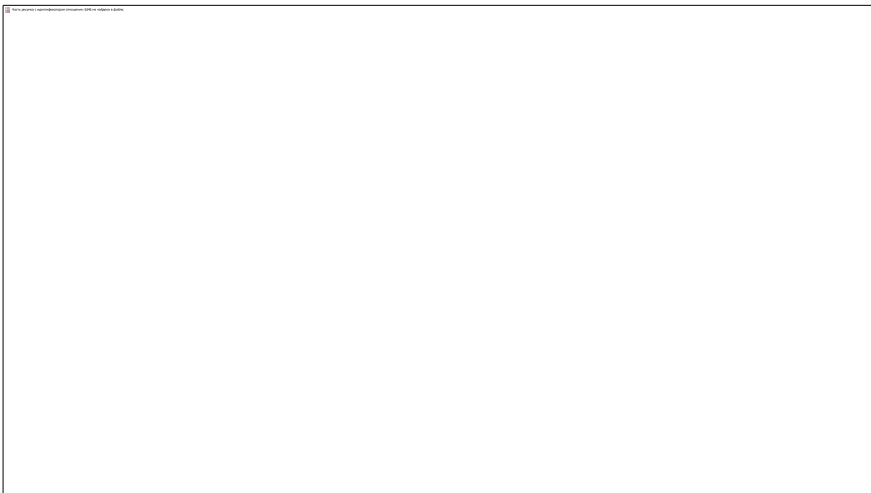
4. Создаем еще одну страницу, повторив пункт 1

5. Даем ей название AddEditPage

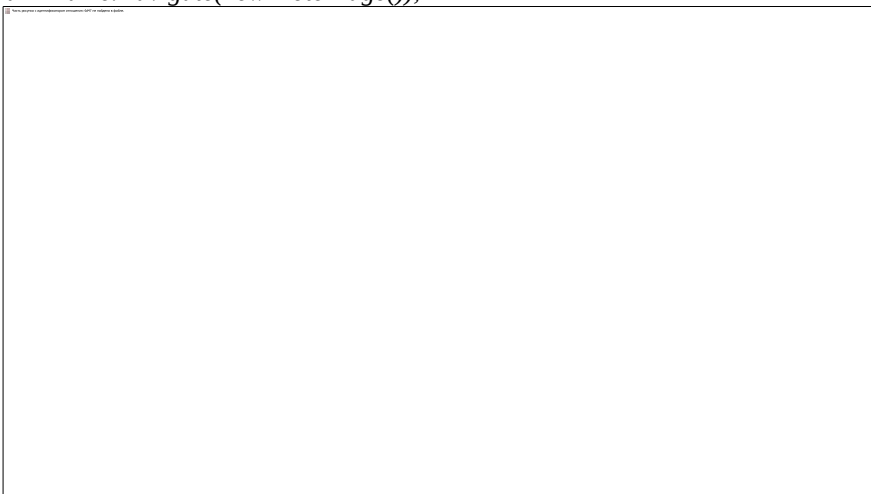
6. Размещаем в ней TextBlock



Для того, чтобы использовать созданные элементы в приложении, необходимо указывать для них имена. Имя — это тоже одно из свойств, по которому можно обращаться к тем или иным элементам в коде. Укажем имя фрейма:



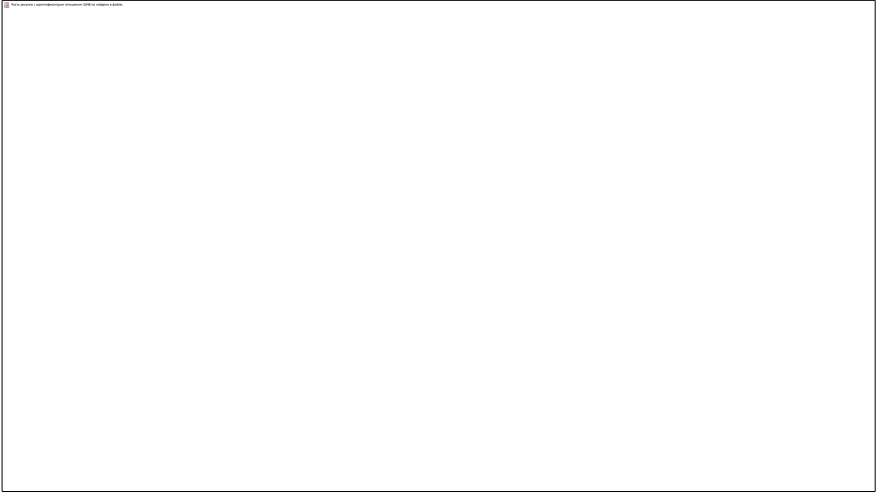
Для отображения первой страницы необходимо прописать следующий код (для перехода в нужное окно нажмите F7): `MainFrame.Navigate(new HotelPage());`



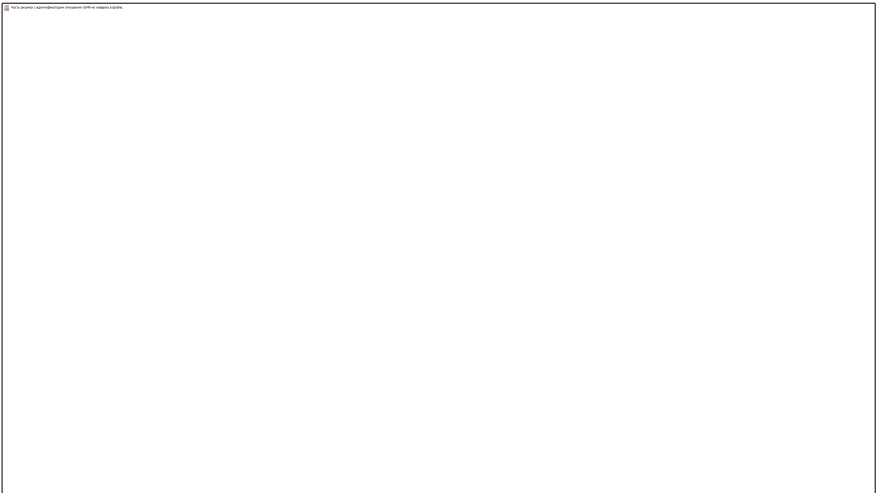
Свойства элементов управления

Например: Background – меняет фоновый цвет элемента

Все свойства элемента можно посмотреть в окне Properties. Там же можно найти все возможные события, доступные для выбранного элемента управления.



Для взаимодействия с кнопкой будем использовать событие Click(). С помощью него мы сможем перейти с первой страницы на вторую. Для создания события пропишем соответствующее значение в верстке:

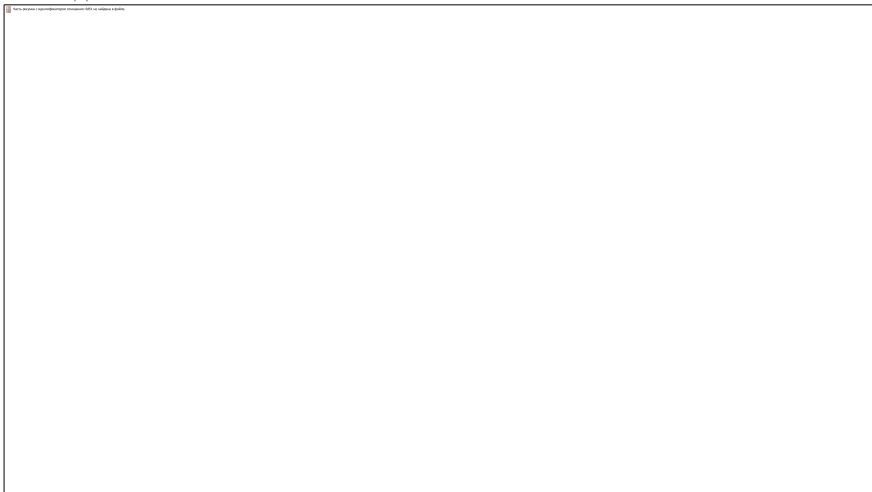


Далее нажмем F12 и попадем в окно обработки нажатия на кнопку. Чтобы добраться до MainFrame для навигации, необходимо создать новый класс:

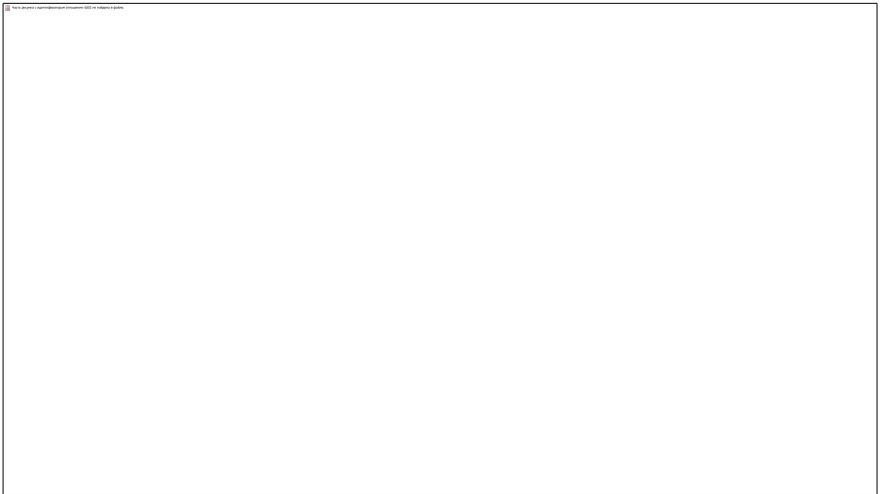
1. Правой кнопкой на название проекта — Add — Class



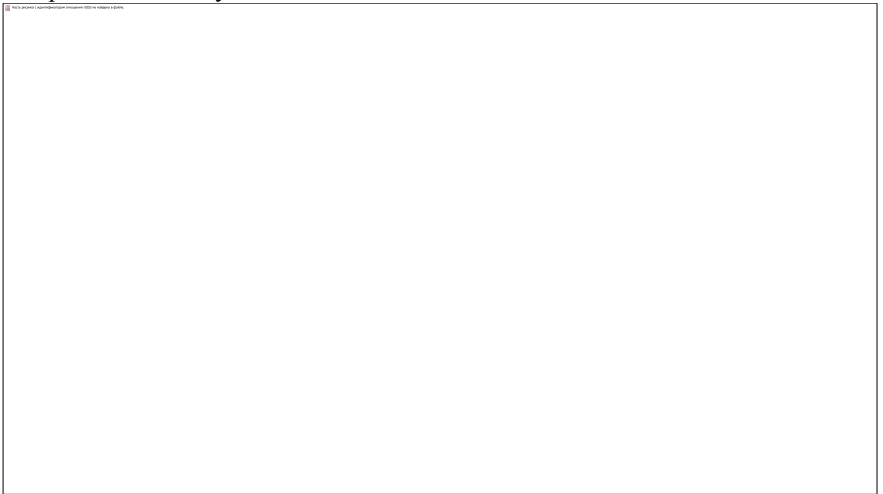
2. Вводим название класса



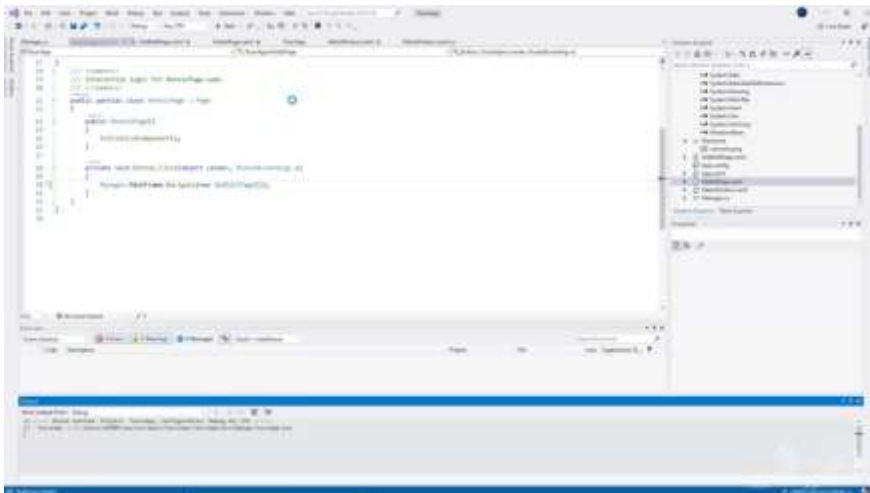
3. Создаем статическое свойство MainForm



4. Присваиваем ему значение в MainWindow



Теперь можем создать навигацию на главной странице

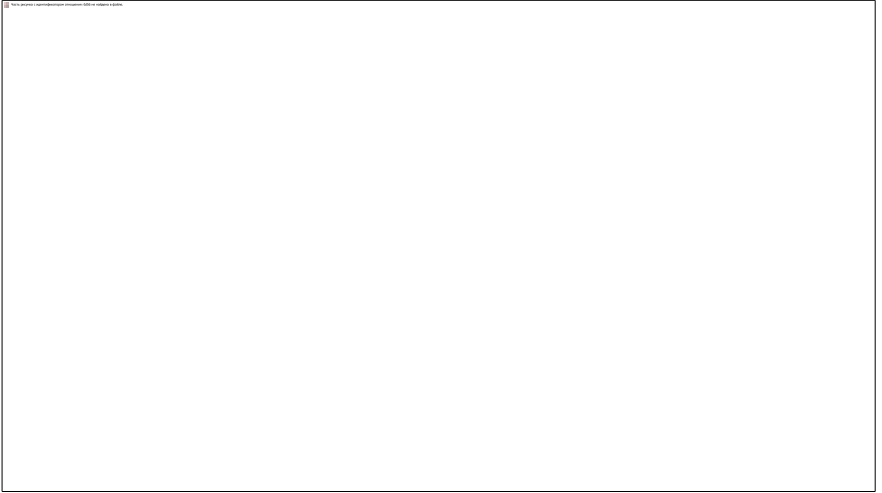


Добавим кнопку «Назад» на главном окне приложения. Установим для нее следующие свойства:

- Name
- Width, Height
- HorizontalAlignment, Margin



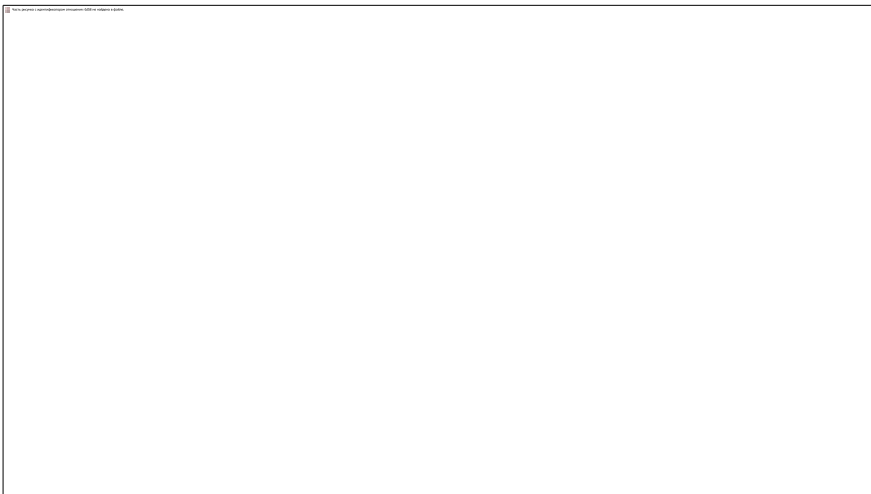
Далее нажимаем F12 и попадаем в окно обработки нажатия на кнопку. Используем следующую логику: обращаемся к менеджеру (Manager), к фрейму (MainFrame) и вызываем метод GoBack



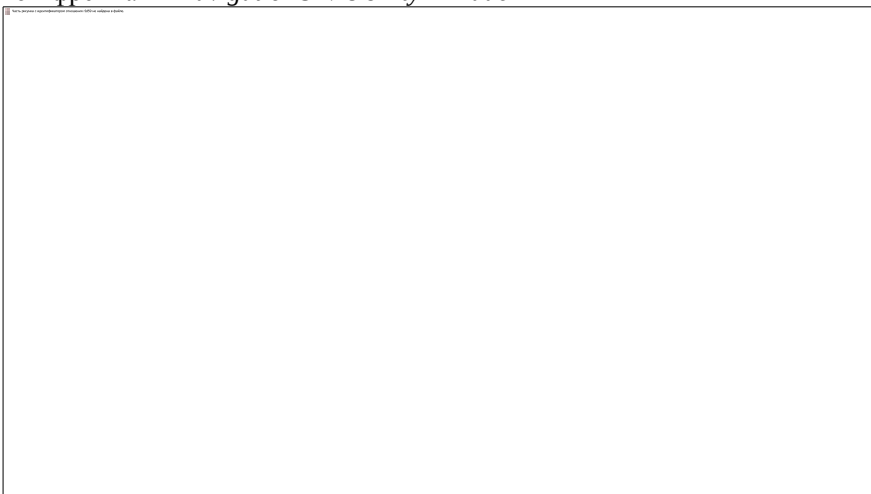
Но она не нужна на главном экране. Чтобы скрыть ее, воспользуемся событием `ContentRendered`



В коде пропишем:



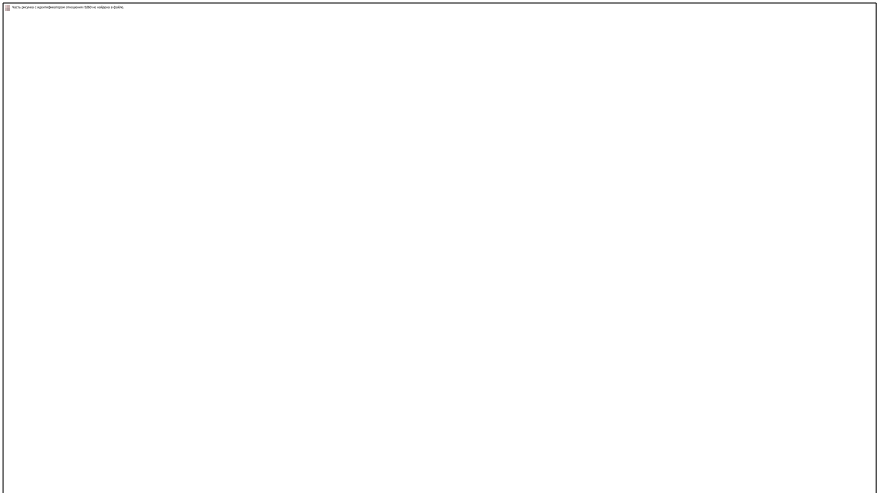
А чтобы скрыть стандартное навигационное меню, воспользуемся свойством фрейма — *NavigationUIVisibility='Hidden'*



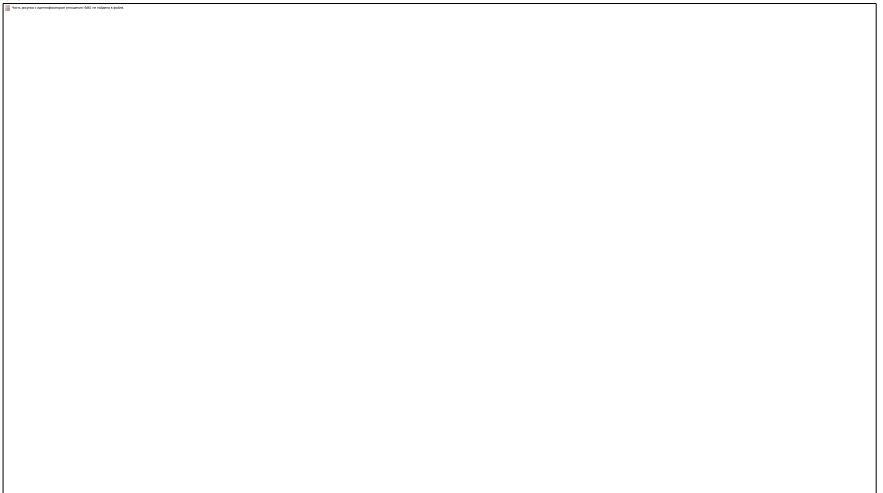
Глобальные стили приложения

Для большинства созданных элементов мы использовали похожий набор свойств: ширина, высота, размер шрифта, отступы и др. Чтобы применять определенные наборы свойств для элементов, WPF предлагает использование глобальных стилей в проекте. Чтобы их создавать в проекте, есть файл *App.xaml*. Используем тег *Style* и свойство *TargetType*, чтобы указать, для каких элементов предназначен данный

стиль.



Внутри тега `Style` используем тег `Setter`. В нем установим необходимые свойства:



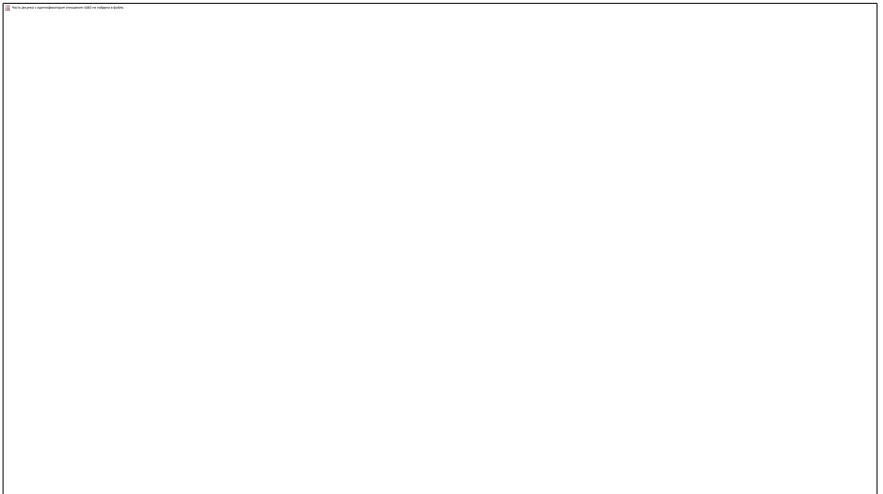
И теперь удалим написанные ранее свойства элементов, для которых есть стиль

РАБОТА С БАЗОЙ ДАННЫХ В ПРИЛОЖЕНИИ: ЧТЕНИЕ, ДОБАВЛЕНИЕ, РЕДАКТИРОВАНИЕ, УДАЛЕНИЕ ДАННЫХ)

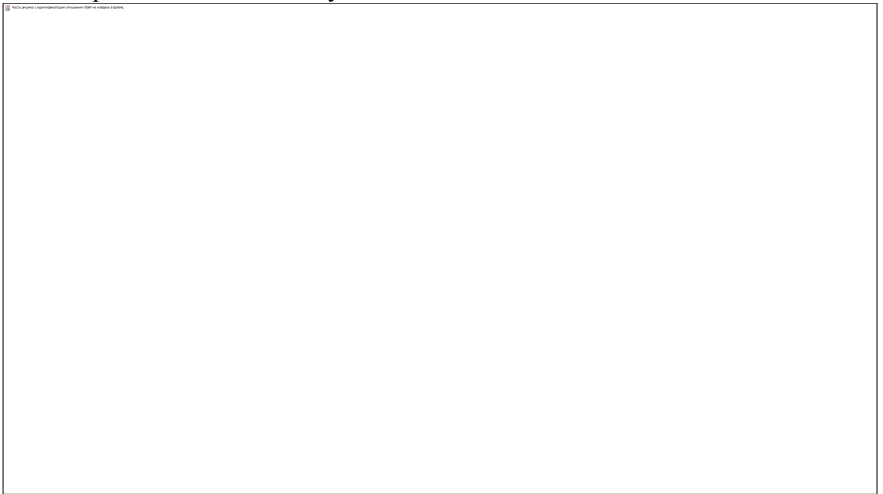
Тему работы с базой данных в приложении будем рассматривать на примере отображения, добавления, редактирования и удаления данных. Для отображения определенных данных система обращается к базе данных, получает ответ, а затем преобразует его в удобный для пользователя вид, который настраивает разработчик. Для работы с базой данных в приложении мы будем использовать Entity Framework, который позволяет работать с базой данных через объектно-ориентированный подход. Он предоставляет ряд существенных преимуществ: вам не нужно беспокоиться о коде доступа к данным, а также знать детали работы СУБД SQL Server и синтаксиса языка структурированных запросов SQL. Вместо этого вы работаете с таблицами базы данных как с классами C#, а с полями этих таблиц, как со свойствами классов, используя вместо SQL запросов более удобный подход – LINQ. Entity Framework берет на себя обязанности по преобразованию кода C# в SQL инструкции

Использование Windows Presentation Foundation (WPF) для создания интерактивных настольных приложений

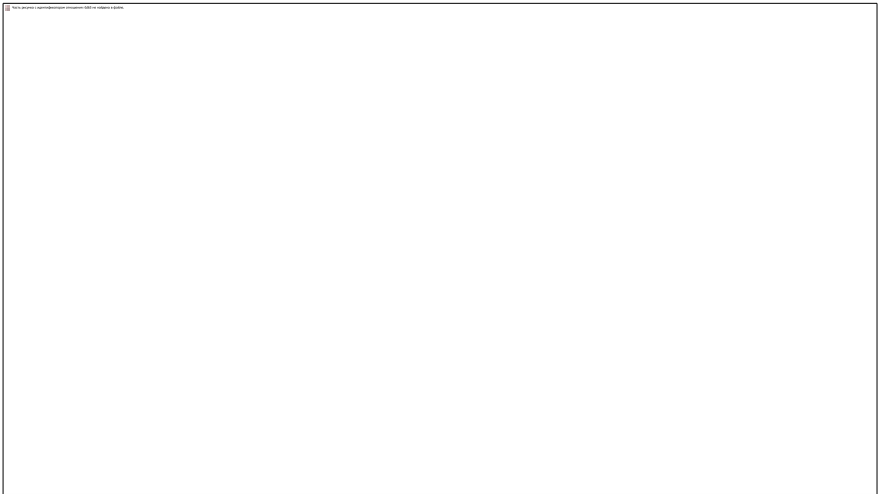
1. New Item...



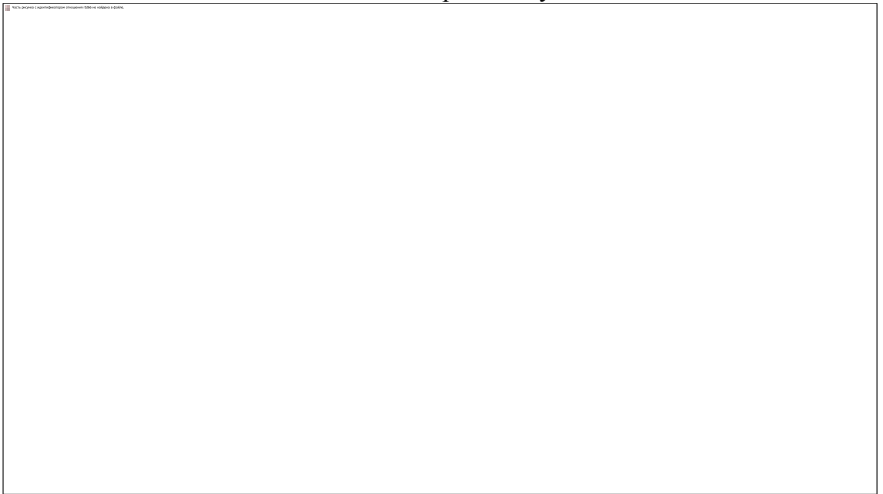
2. Выбираем ADO.NET Entity Data Model и даем ей название



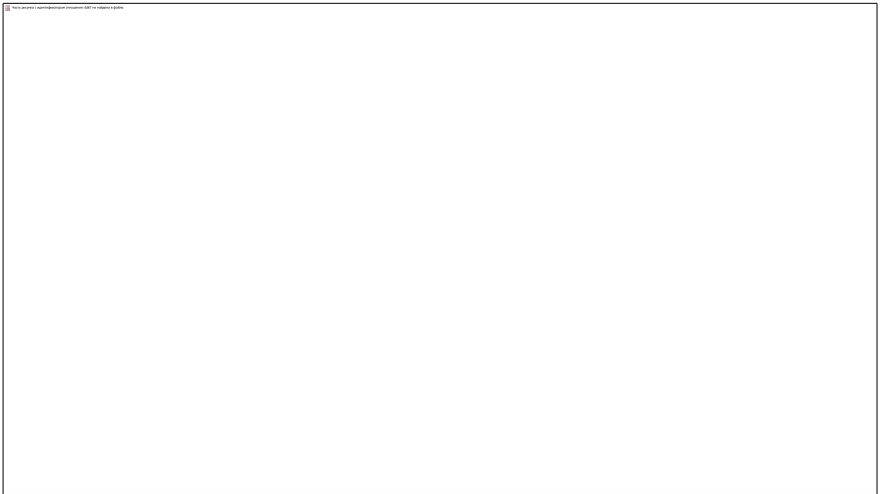
3. Выбираем EF Designer from database



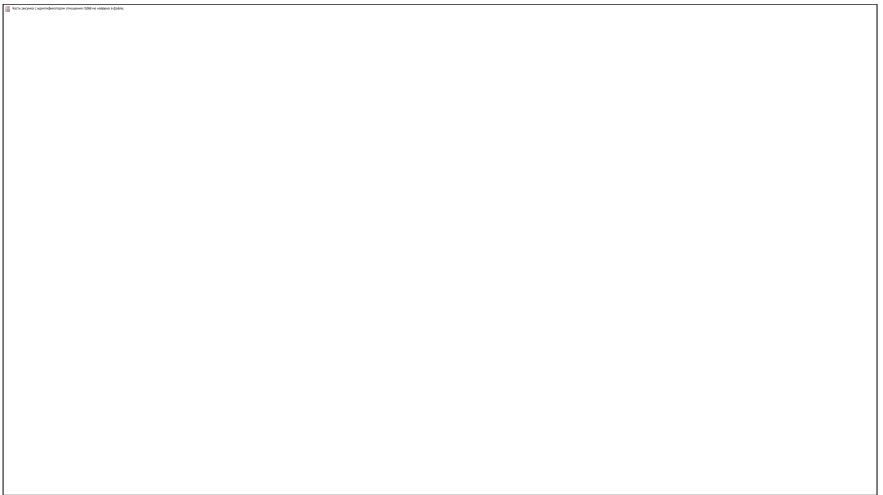
4. Создаем новое подключение и выбираем базу данных



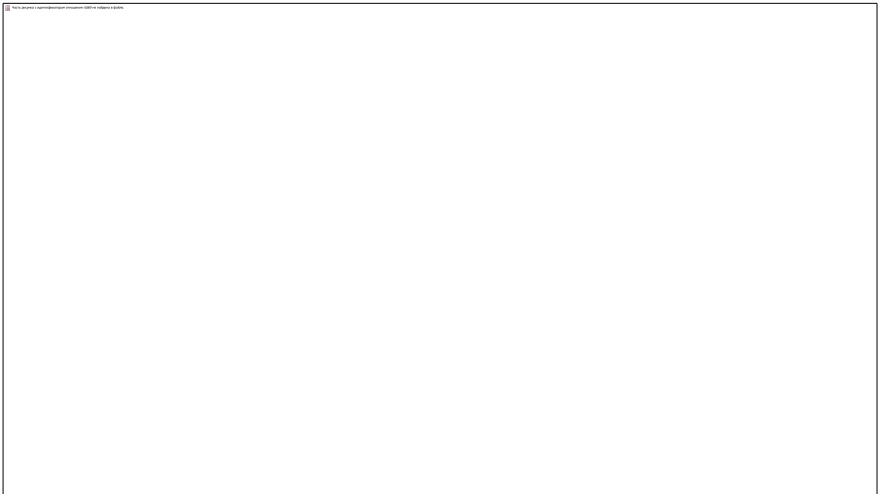
5. Нажимаем Next



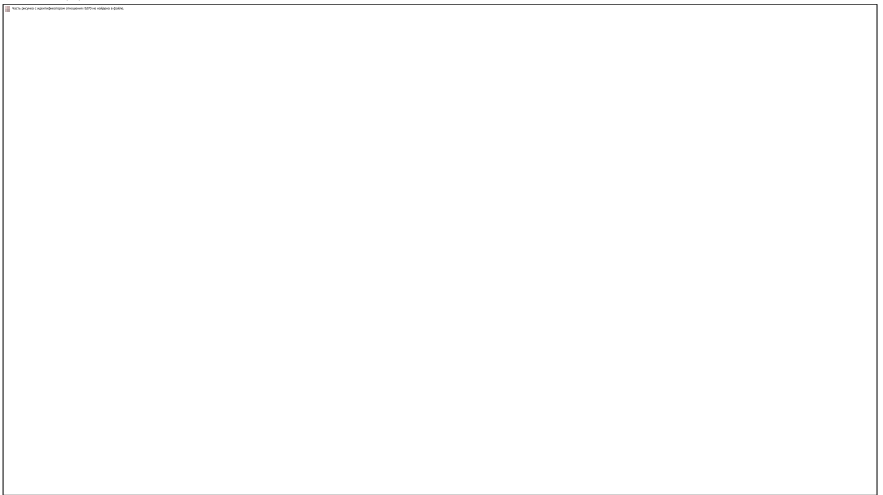
6. Нажимаем Next



7. Выбираем все таблицы из списка

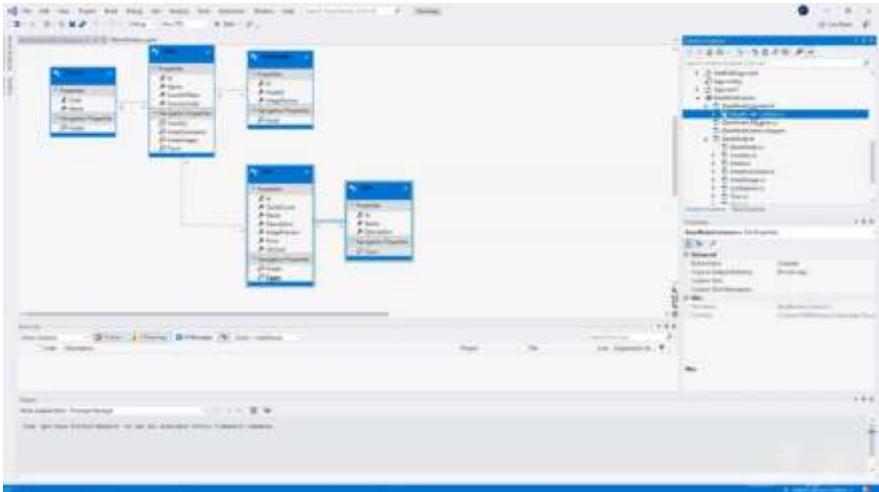


8. Модель готова

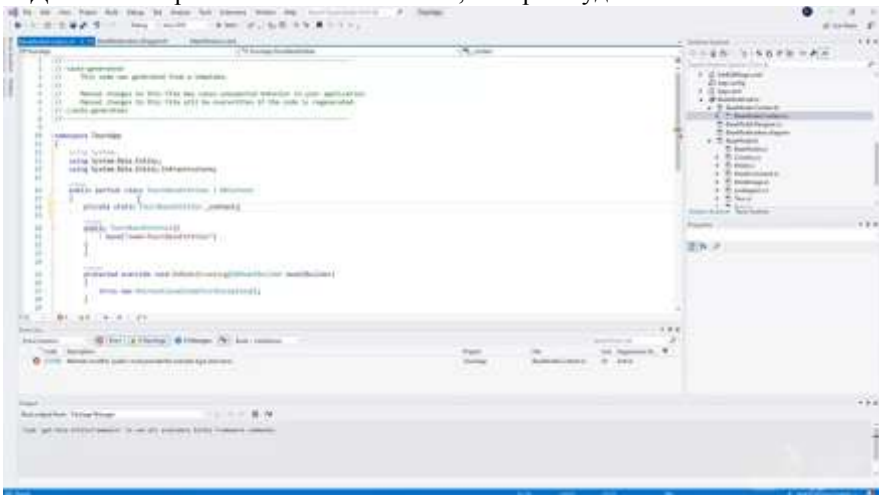


Обращение к модели данных. Паттерн SingleTone. Чтение, добавление, редактирование, удаление данных

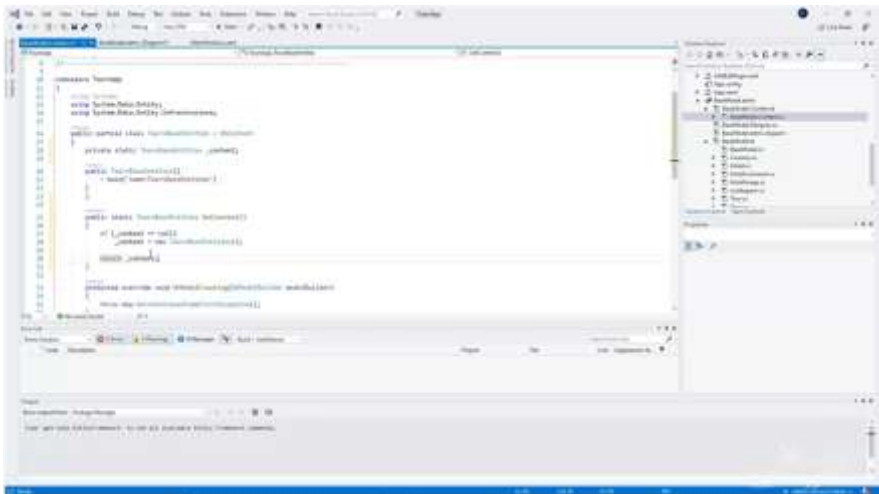
1. Открываем файл BaseModel.Context.cs



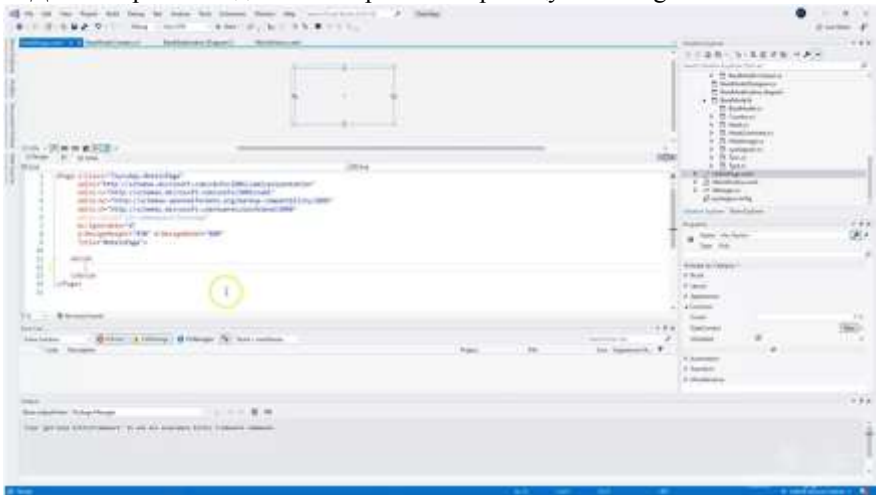
2. Добавляем приватное статическое поле, которое будет контекстом



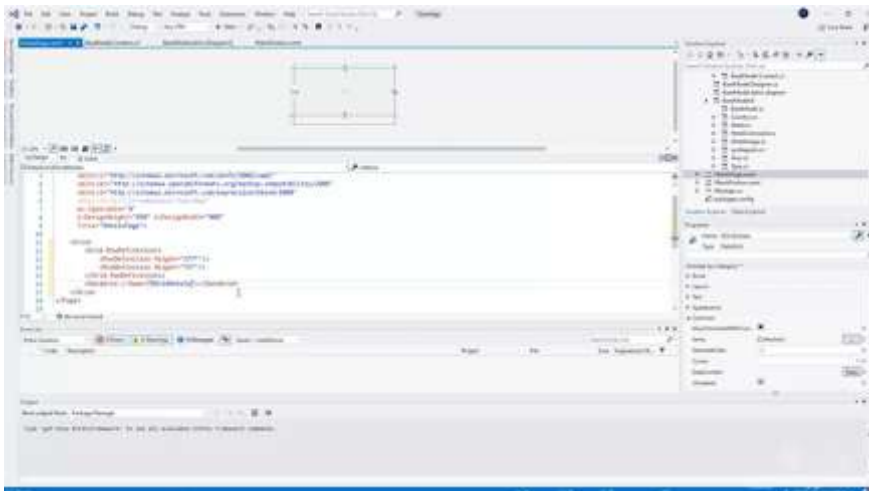
3. Добавляем метод получения экземпляра этого контекста



4. Для отображения данных открываем страницу HotelPage.xaml



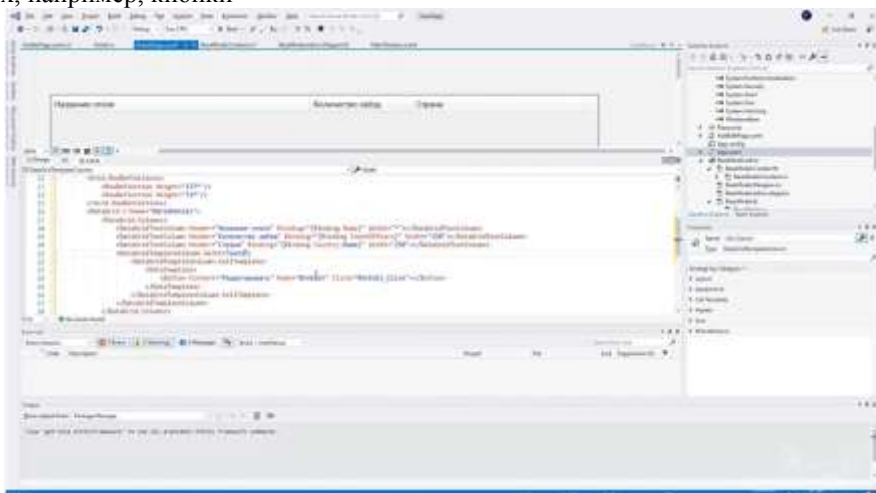
5. Размечаем Grid на две части



6. Список данных выводится по столбцам, которые прописывает разработчик. Устанавливаем их с помощью свойства `DataGrid.Columns`, которое и описывает набор столбцов.

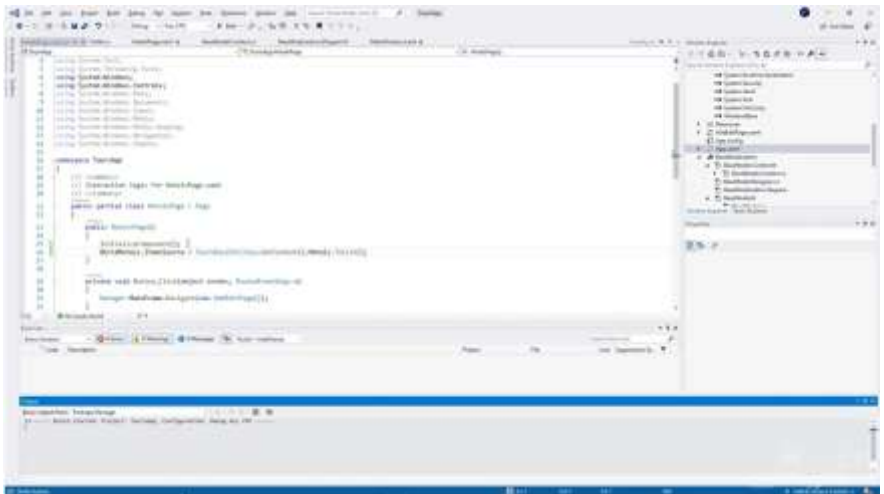
`DataGridTextColumn` для текстовых столбцов

`DataGridTemplateColumn` для более сложного представления данных, например, кнопки

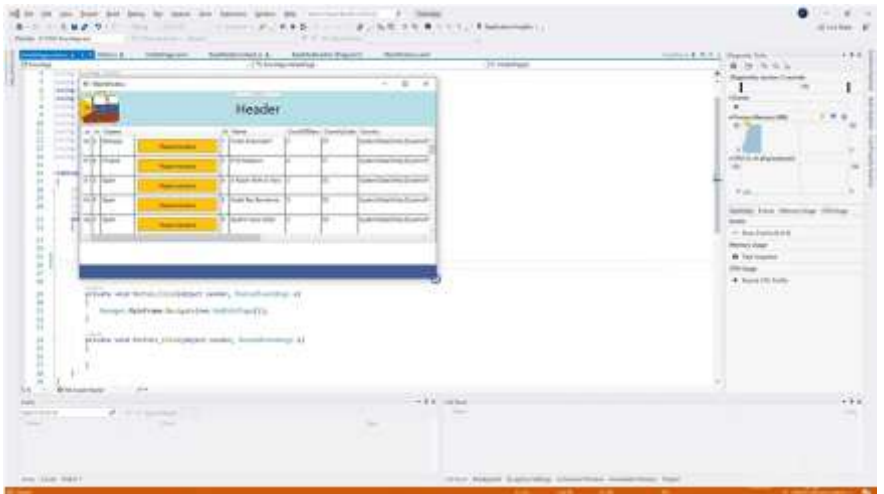


7. Далее загрузим список отелей в код в таблицу:

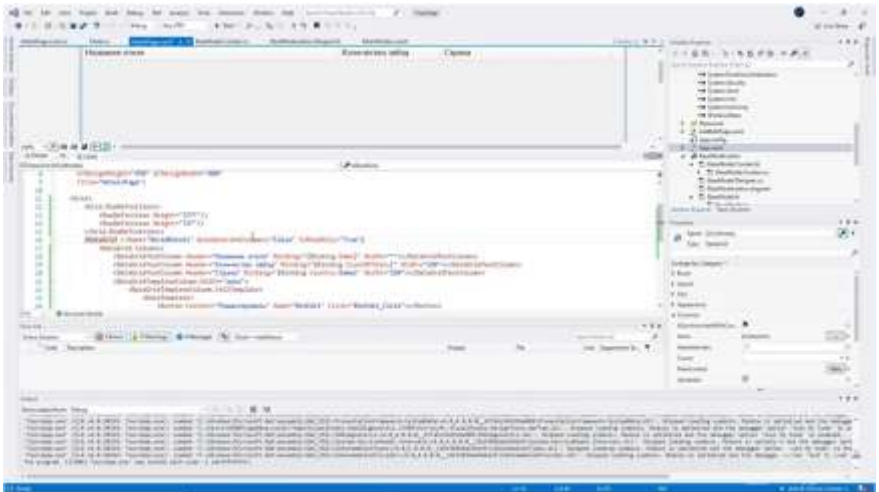
- Нажимаем F7
- Обращаемся к контексту модели



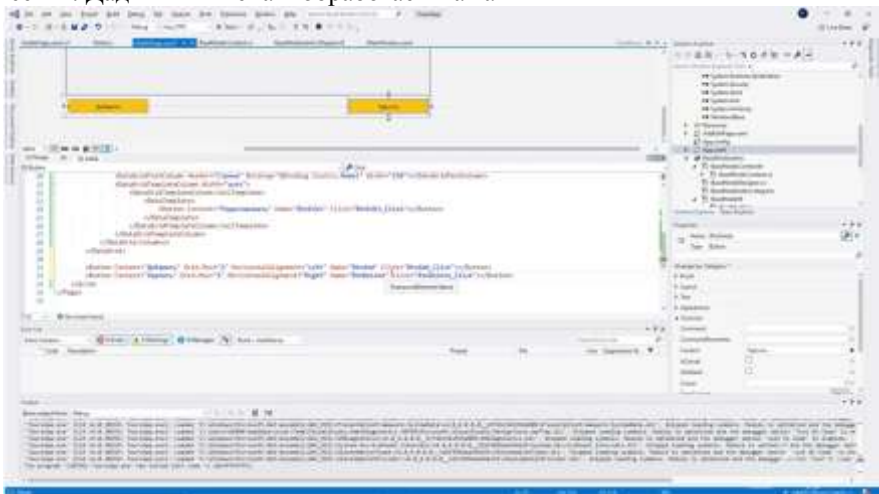
8. Пробуем запустить программу и видим набор данных уже в приложении



9. Для отключения загрузки всех свойств объектов, необходимо прописать `AutoGenerateColumns="False"` и `IsReadOnly="True"`



10. Также добавим кнопки для добавления и удаления во второй строке сетки. Дадим им имена и обработаем нажатия

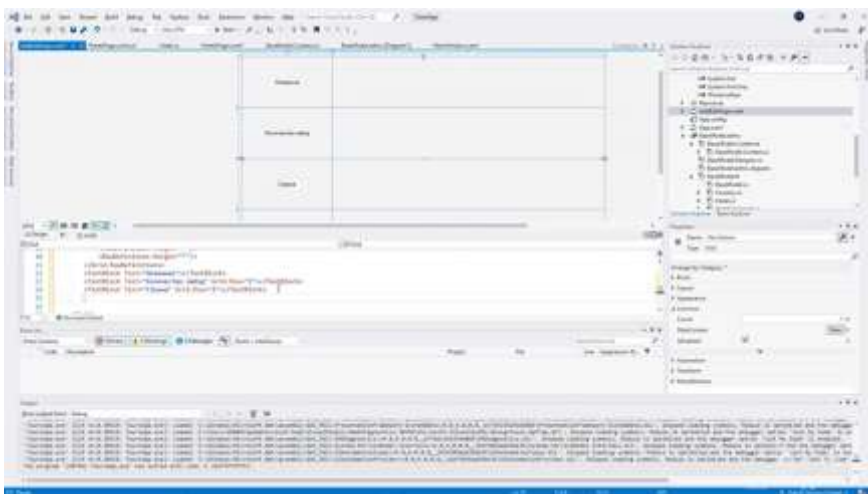


11. Информация об отелях может меняться, поэтому важно реализовать функции добавления, редактирования и удаления. Для этого используем уже созданную нами вторую страницу, добавив необходимые элементы управления

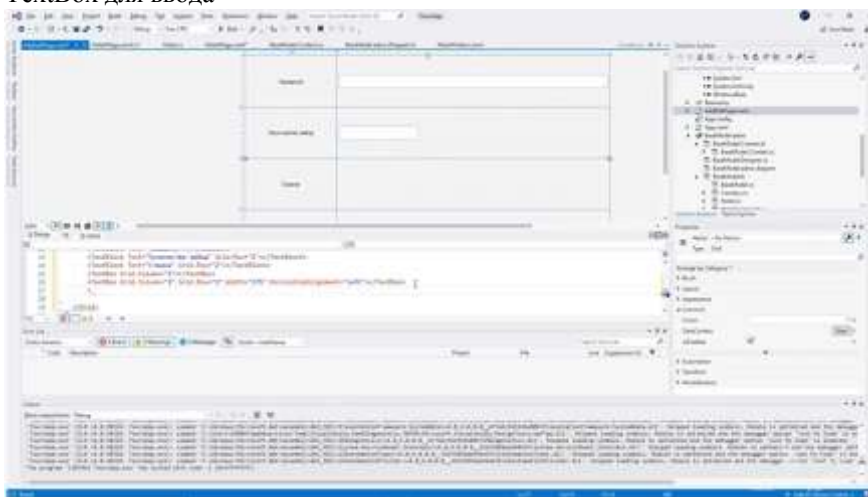
а) Сверстаем сетку, состоящую из трех строк и двух столбцов

б) Разместим элементы для ввода данных:

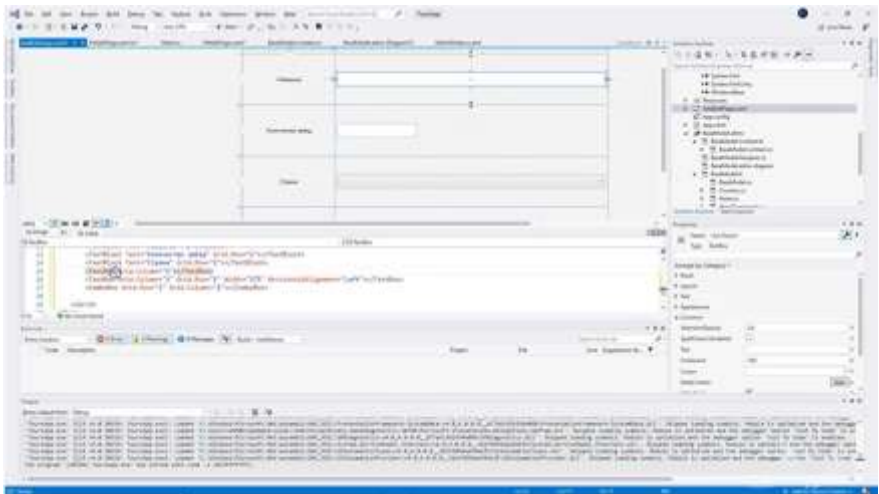
Текстовые блоки для отображения подсказок, что именно вводить



TextBox для ввода

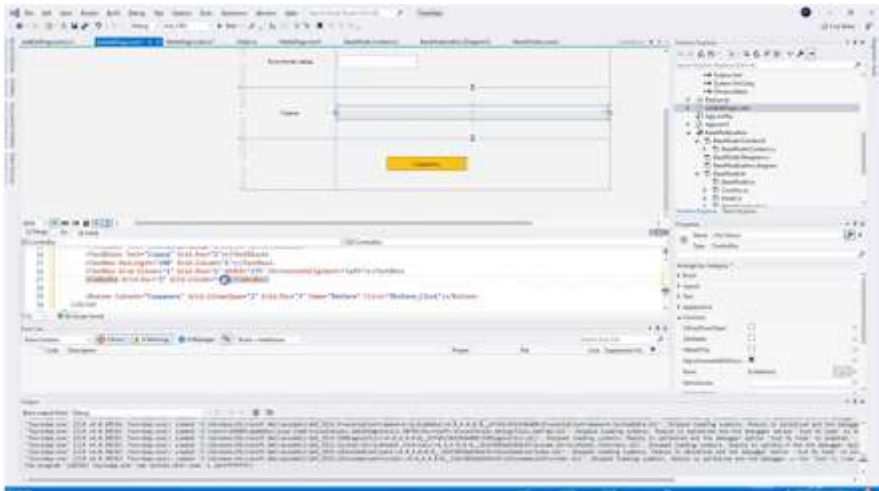


ComboBox для выпадающего списка стран

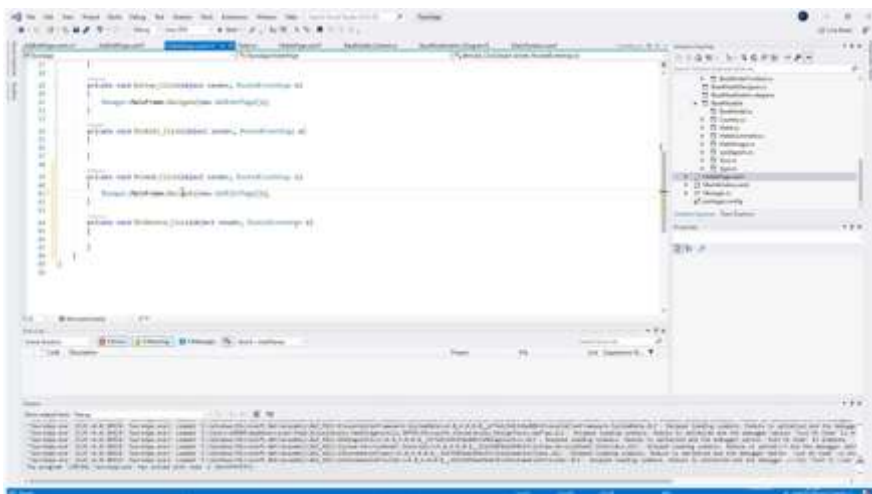


с) Устанавливаем максимальное число символов для текст-боксов равных максимальному числу символов в базе данных

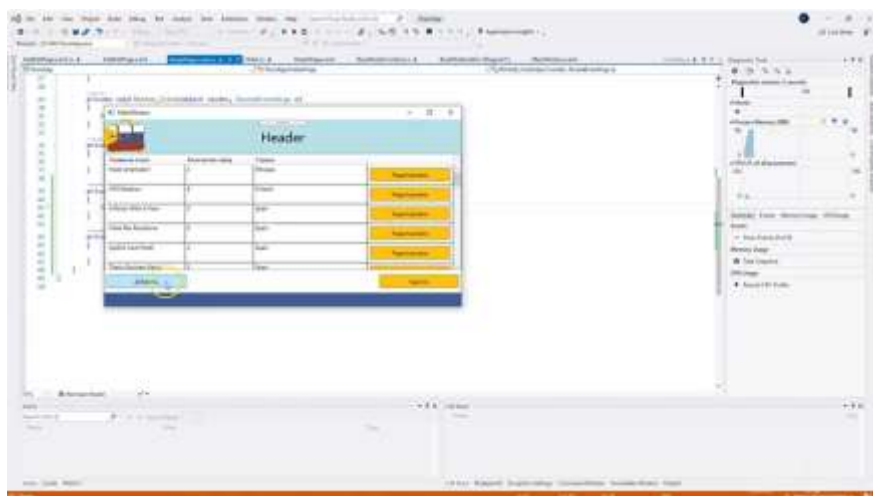
д) Добавим кнопку для сохранения изменений, дадим ей имя и обработаем нажатие

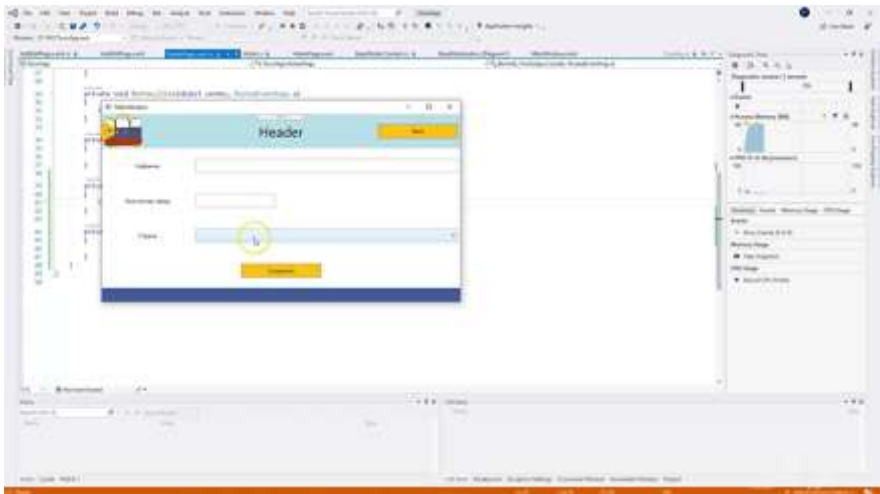


е) Доработаем комбо-бокс для выпадающего списка: дадим ему имя, укажем отображаемое свойство



13. Проверяем работоспособность

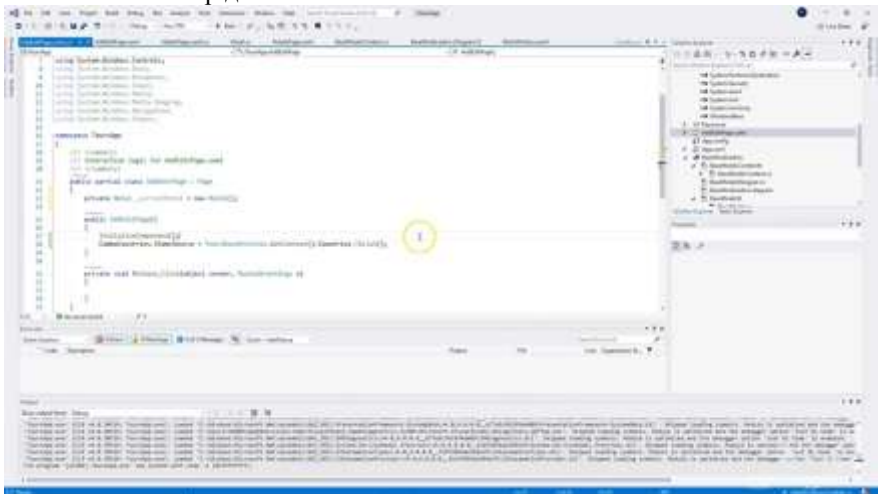




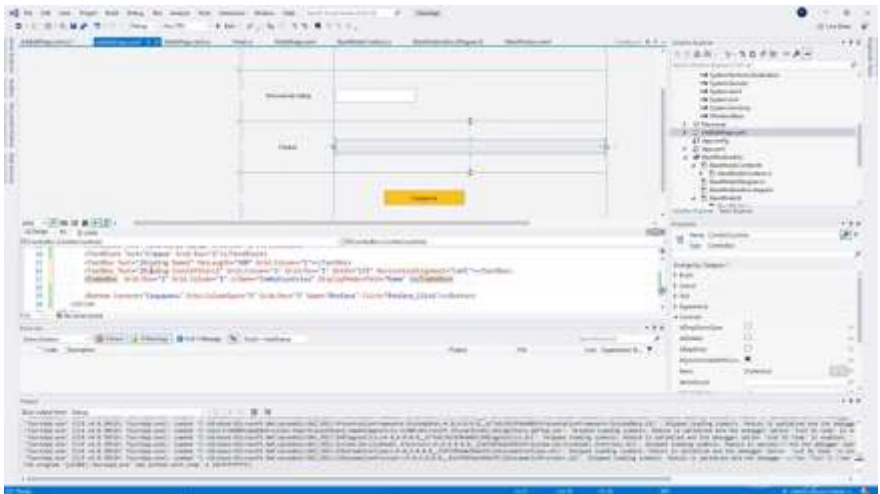
Реализуем функции добавления, редактирования и удаления данных

Реализация функции добавления

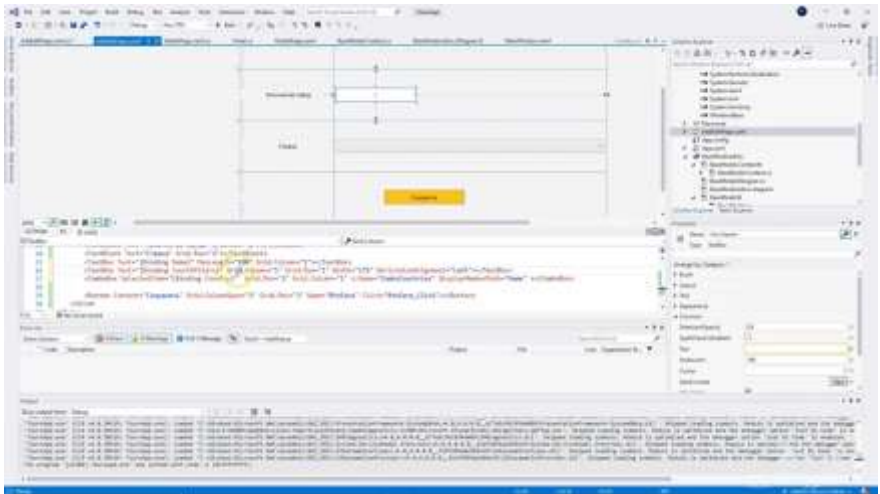
1. На странице AddEditPage добавим новое поле, которое будет хранить в себе экземпляр добавляемого отеля



2. При инициализации установим DataContext страницы – этот созданный объект

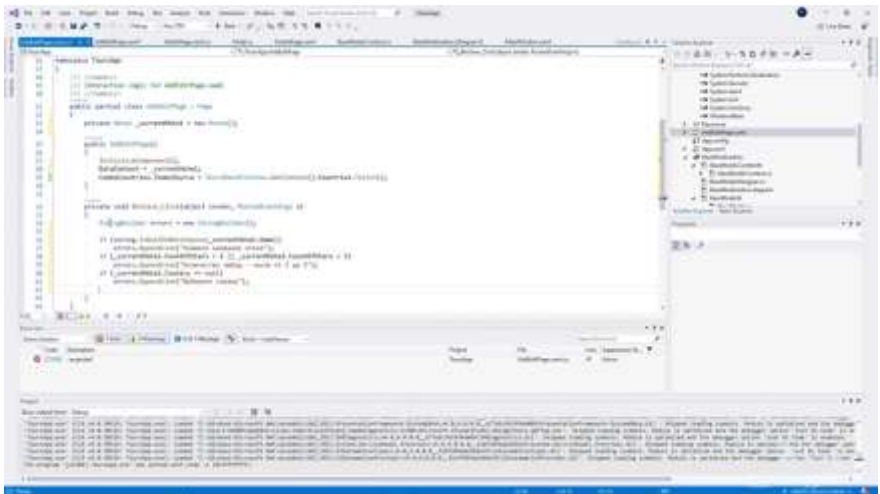


5. И комбо-бокс будет обращаться к стране, которую мы выбрали для отеля

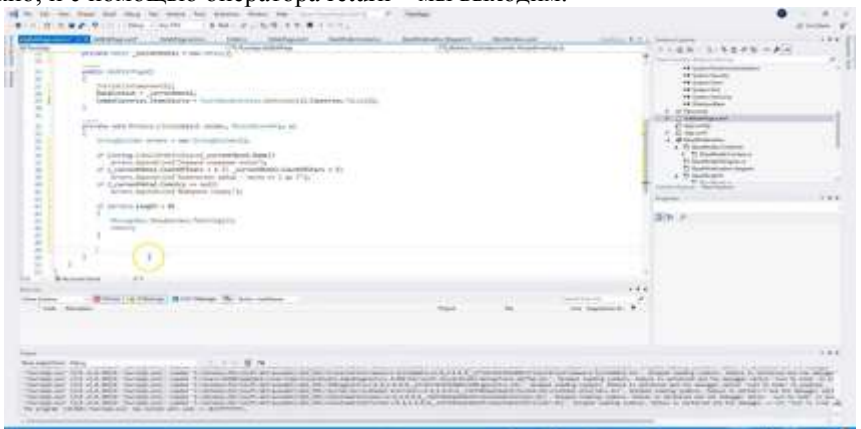


6. Далее обрабатываем нажатие на кнопку Сохранение и в коде пропишем логику обращения к модели данных и добавления нового экземпляра отеля.

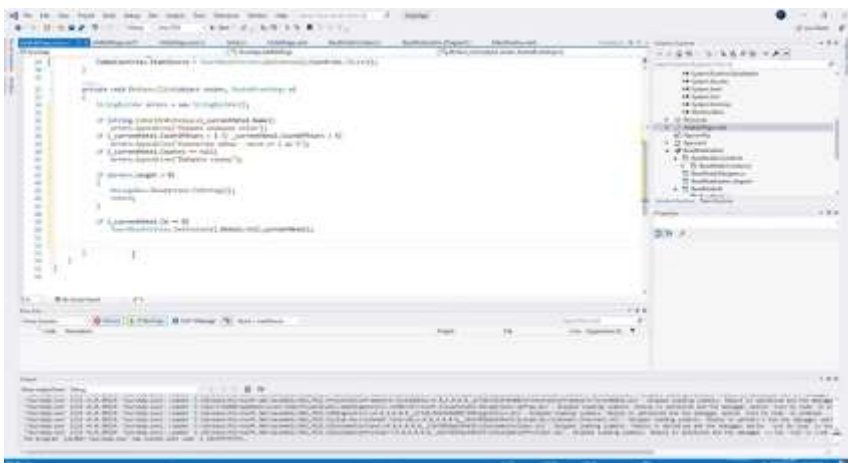
а) Прежде чем сохранять данные, сделаем проверки на количество символов, заполняемость объектов, звездность (т. к. количество звезд должно быть от одного до пяти) и выбор страны



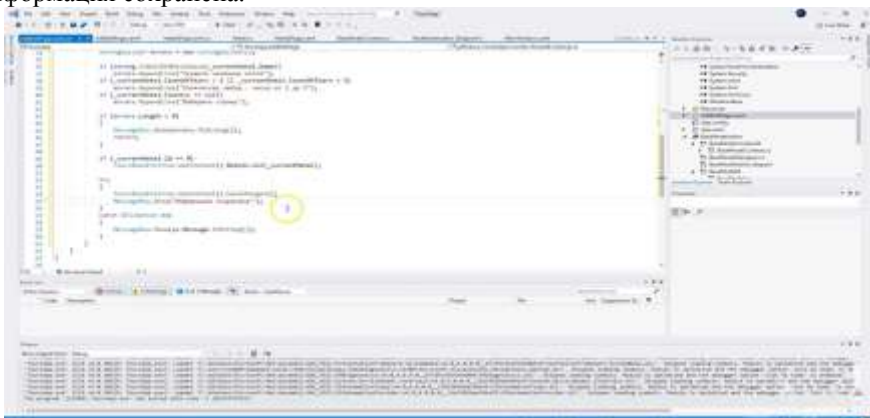
б) После прохождения проверки нужно узнать, возникли ли ошибки, обратившись к переменной `errors`. Если в переменной что-то есть, то необходимо вывести сообщение об ошибке (то, что накопилось во время проверки). Соответственно, дальнейшее выполнение функции не нужно, и с помощью оператора `return` – мы выходим.



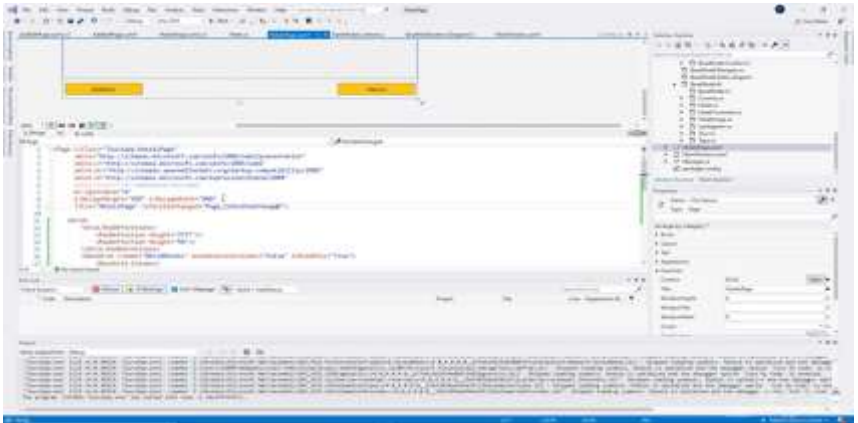
с) Если же все хорошо, и у нас происходит операция добавления (т.е. еще не присвоен код нового отеля), то мы будем пытаться добавить модель или экземпляр созданного отеля. Получив его контекст и обратившись к таблице отелей, с помощью метода `Add` мы добавляем созданный экземпляр.



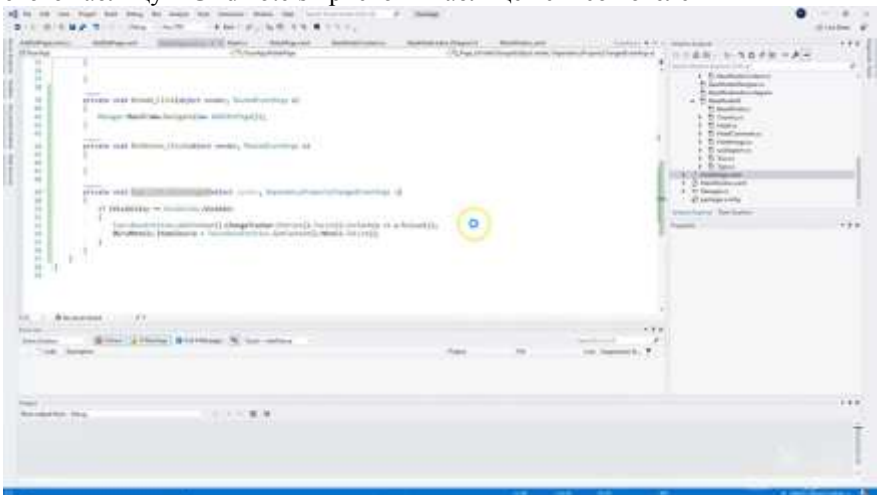
д) Далее напишем код для сохранения изменений, используя метод SaveChanges. Этот метод является коварным, его необходимо поместить в блок try-catch, чтобы он отработал корректно, и в случае возникновения какой-либо непредвиденной ошибки, приложение не «упало», а корректно работало. Мы выведем сообщение об ошибке, если она появилась. В случае успешного сохранения выведем сообщение о том, что информация сохранена.



7. Теперь можно вернуться назад. При возврате на страницу со списком отелей, нам необходимо выводить актуальную информацию, обновляя список в таблице. Для этого мы будем использовать событие у страницы isVisibleChange. Оно срабатывает каждый раз, когда страница отображается, либо скрывается



С помощью F12 переходим в код. Если видимость страницы isVisible, мы будем обращаться к контексту с помощью свойства ChangeTracker ко всем сущностям, которые есть. И для каждой из них будем выполнять метод перезагрузки и вывода актуальных данных. После этого таблицу DGridHotels присвоим таблице «список отелей»

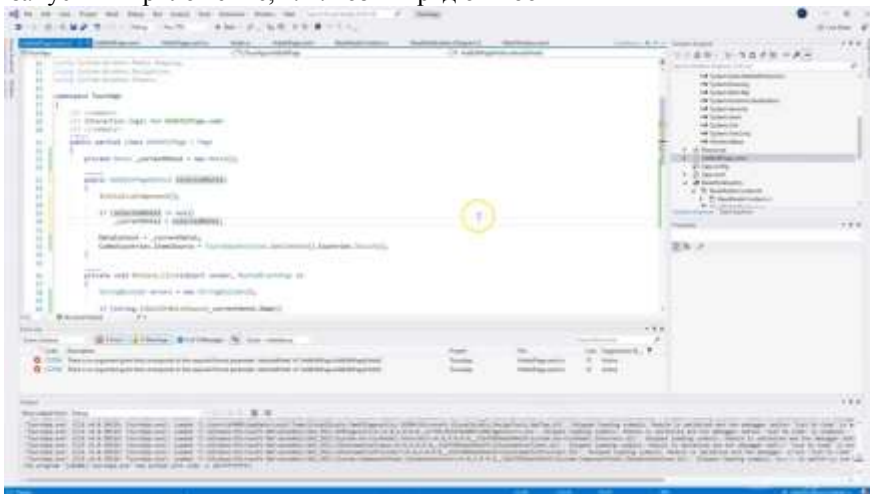


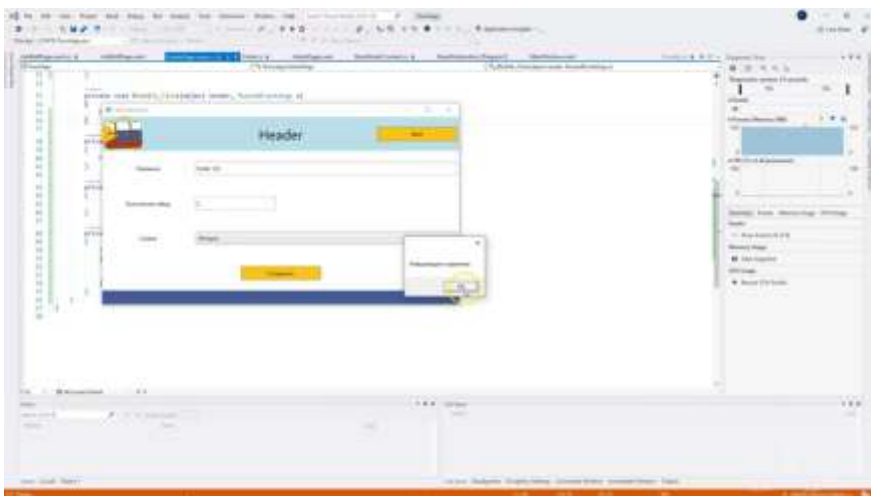
8. Запускаем программу и проверяем функцию добавления данных Реализация функции редактирования

Для функции редактирования данных целесообразно использовать ту же страницу, что мы делали для добавления. Каким образом это будет происходить? В случае, если пользователь намерен изменить информацию об объекте, система будет отображать страницу добавления с информацией о редактируемом объекте. Измененная информация будет

фиксироваться в базе данных и отображаться в списке, как было при добавлении.

В первую очередь добавим параметр нашей странице AddEditPage. В нее мы будем передавать экземпляр выбранного отеля и, в случае если он не пустой, присваивать нашему полю CurrentHotel. Мы не можем сейчас запустить приложение, т. к. возник ряд ошибок

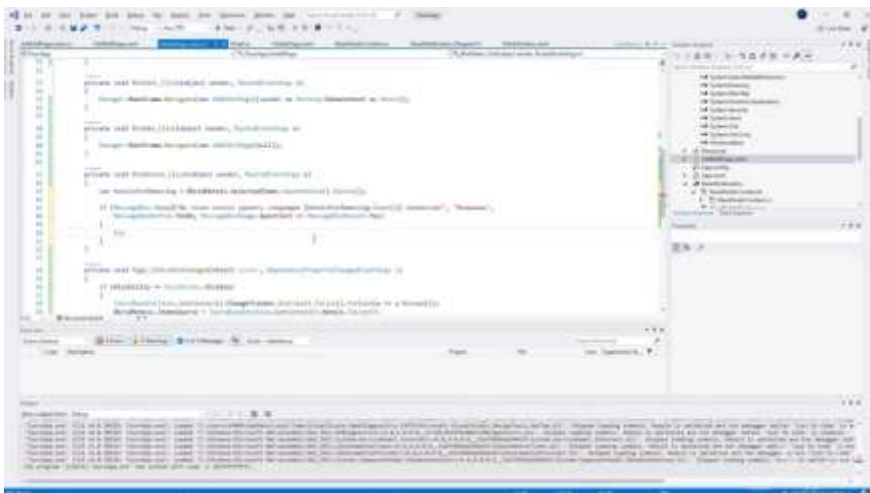




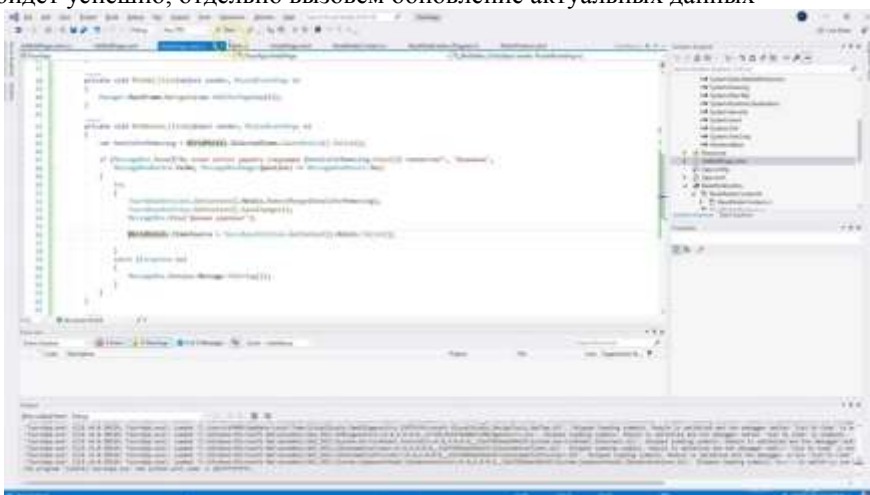
Реализация функции удаления

Для реализации функции удаления отдельное окно не потребуется. Это значительно сократит время на разработку, но удаление требует к себе особого внимания. Любые действия, безвозвратно изменяющие данные в базе данных, должны запрашивать подтверждение пользователя. Именно поэтому, в первую очередь, для нажатия на кнопку удаления мы реализуем сообщение с вопросом: действительно ли пользователь хочет это сделать.

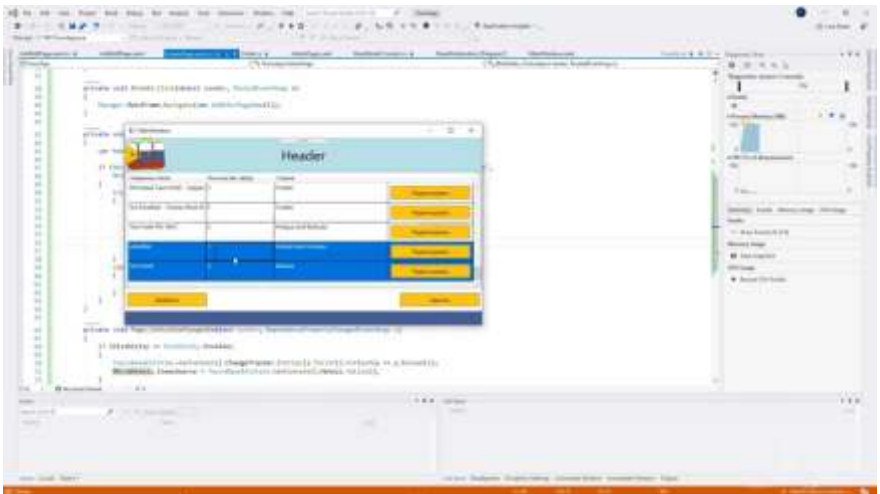
Итак, получаем список отелей для удаления, обратившись к таблице с отелями. Выбираем все элементы, которые мы выделили, преобразуем их в список отелей. И затем, в сообщении, будем спрашивать пользователя: «Вы точно хотите удалить следующие `hotelsForRemoving.Count()` элементов?». Укажем здесь заголовок сообщения – «Внимание», затем укажем, какие кнопки доступны при диалоге с пользователем: «Да» или «Нет», и выберем изображение – «Question».



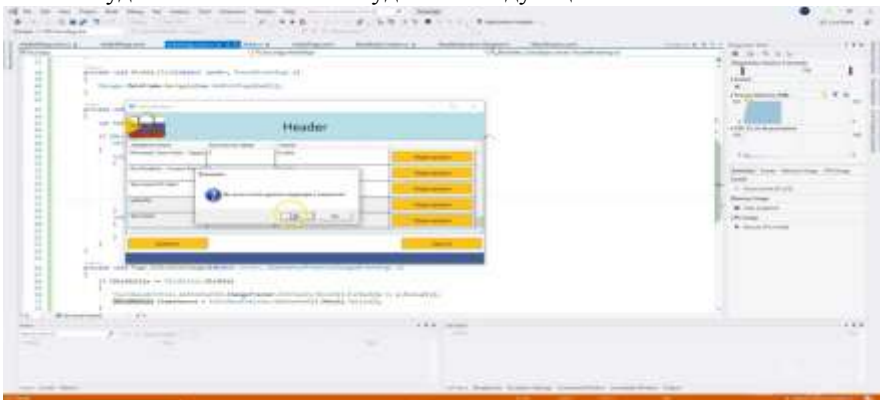
Если результатом диалога от пользователя было нажатие на кнопку «yes», то мы будем выполнять удаление. Для этого обратимся к модели данных, используя блок TryCatch. Получив контекст, попробуем с помощью метода RemoveRange удалить все полученные выделенные отели. В случае, если все будет хорошо, отобразим сообщение. Иначе, выводим сообщение об ошибке. Также, в случае, если удаление произойдет успешно, отдельно вызовем обновление актуальных данных

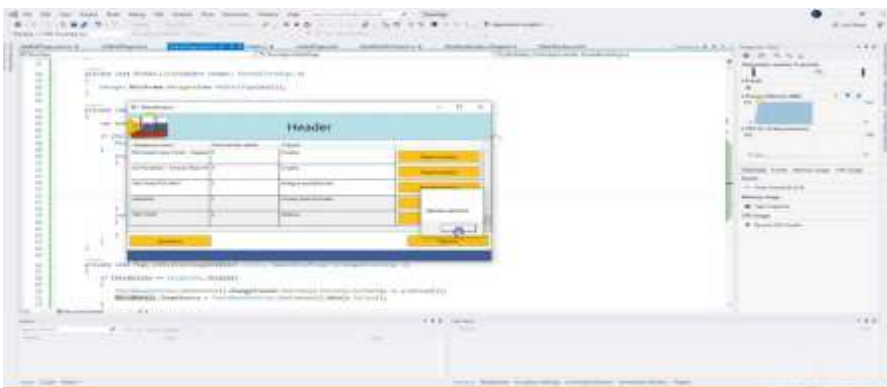


Сохраним и проверим работу приложения. Выделим два отеля.

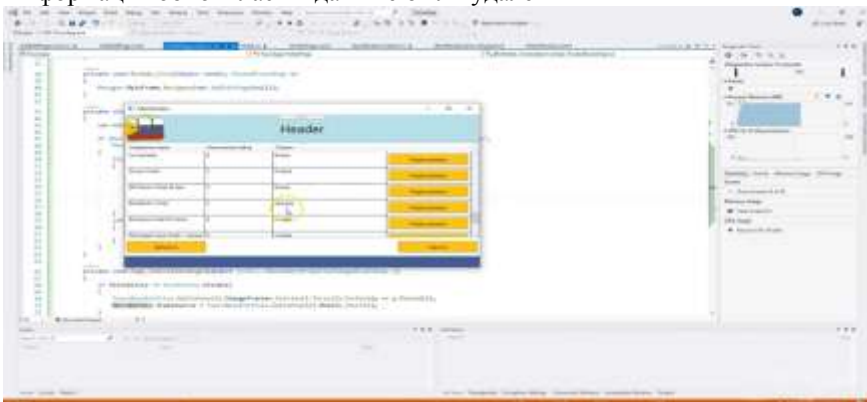


Нажмем удалить – точно хотим удалить следующие 2 элемента





Информация обновилась – данные были удалены



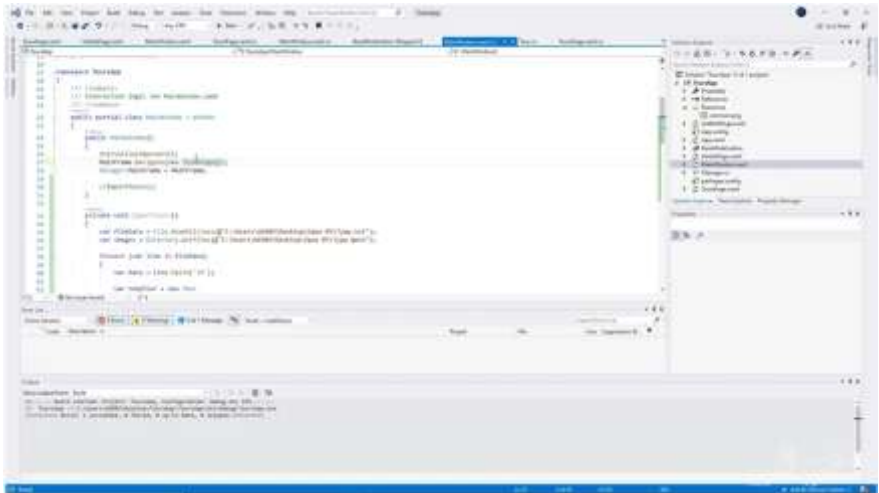
СОЗДАНИЕ СПИСКОВ (LISTVIEW). ПОИСК И ФИЛЬТРАЦИЯ ДАННЫХ

Продолжим разработку настольного приложения и поговорим об альтернативном DataGrid'у элементе, который может отображать информацию из базы данных – ListView. Как правило, он представляет собой стандартный список. Однако при желании вы можете сделать сложную компоновку объекта, которую не получилось бы реализовать с помощью DataGrid. Также можно вывести элементы не только построчно, но и, например, плитками, реализовать поиск и фильтрацию информации.

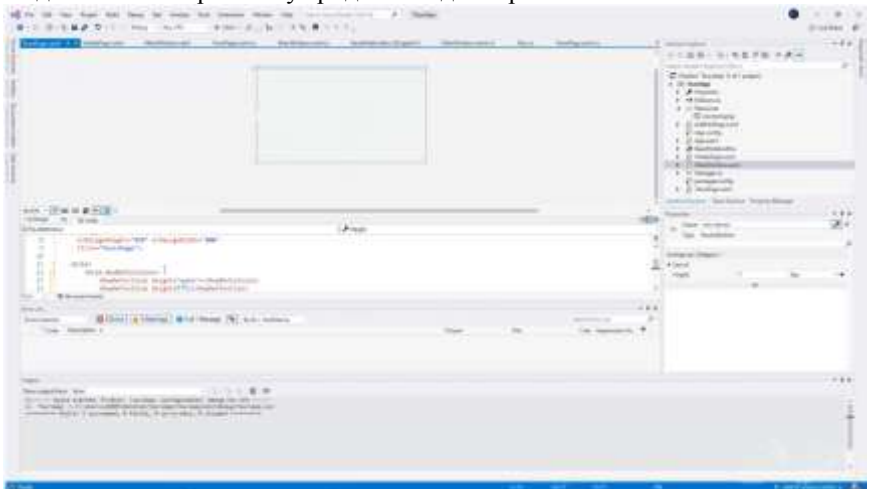
Вывод информации о турах с ListView

Для вывода информации о турах добавляем новую страницу с ListView. Переходим на нее сразу после инициализации компонентов

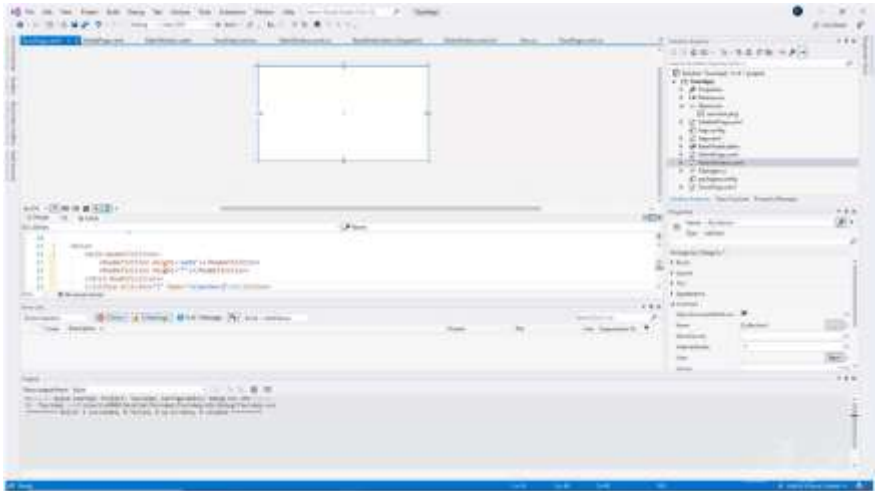
MainWindow



Подготавливаем разметку грида – это две строки

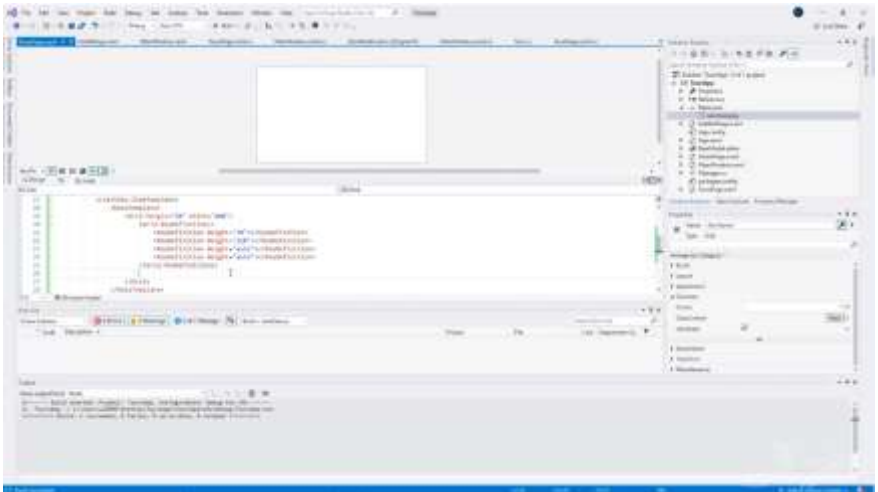


Добавляем элемент ListView, размечаем в гриде и задаем имя

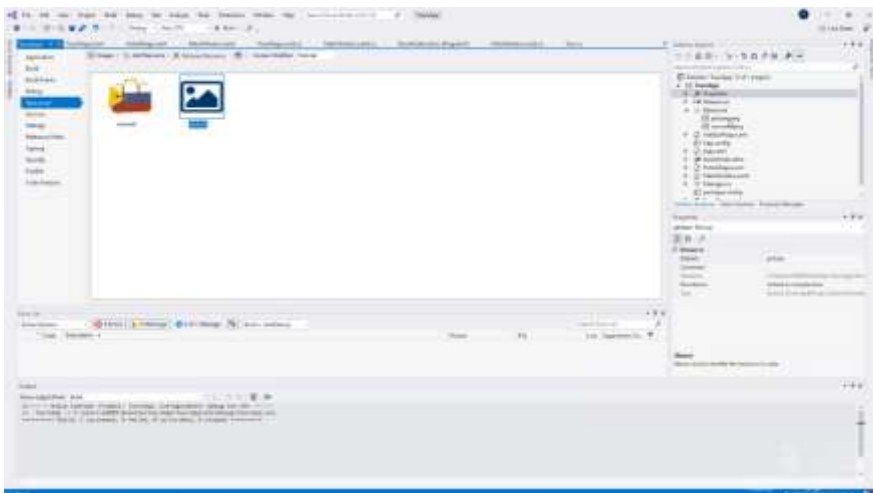


Для вывода информации о туре создается шаблон элемента в списке – то представление, которое должно отображаться для каждого элемента

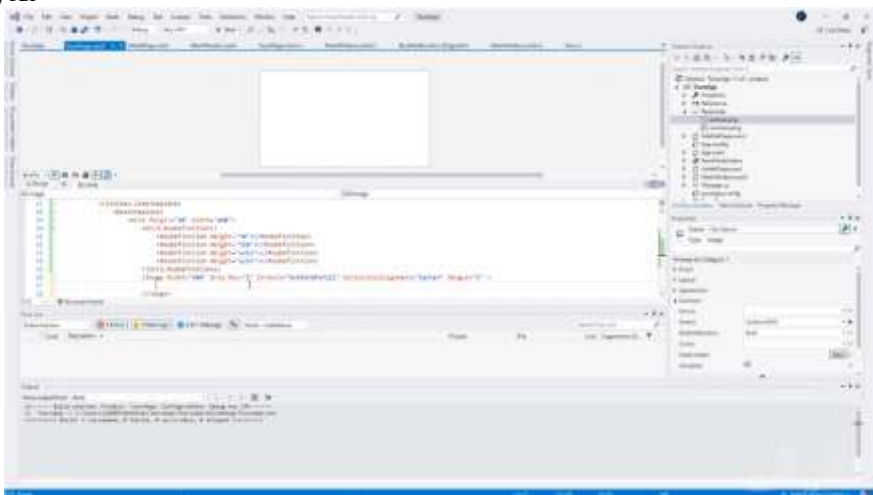
Создаем шаблон и размечаем его грид, указав при этом размер строк



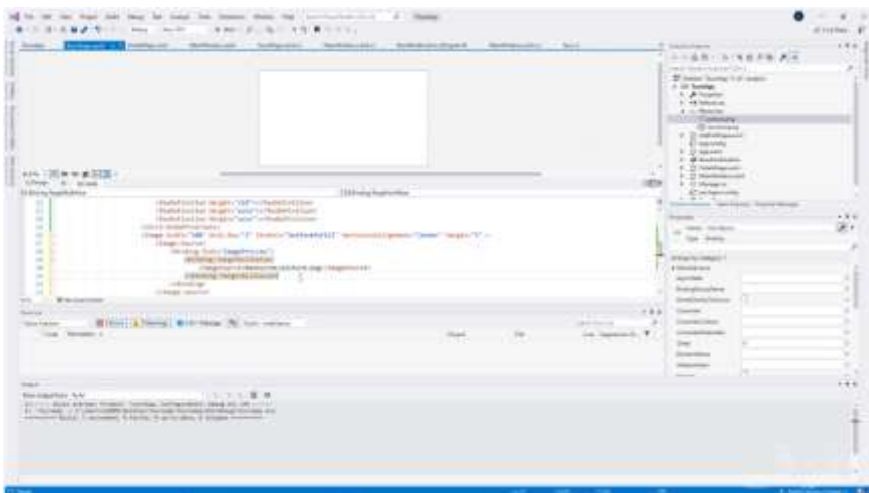
Добавляем в ресурсы изображение, которое будем выводить при отсутствии картинки у тура



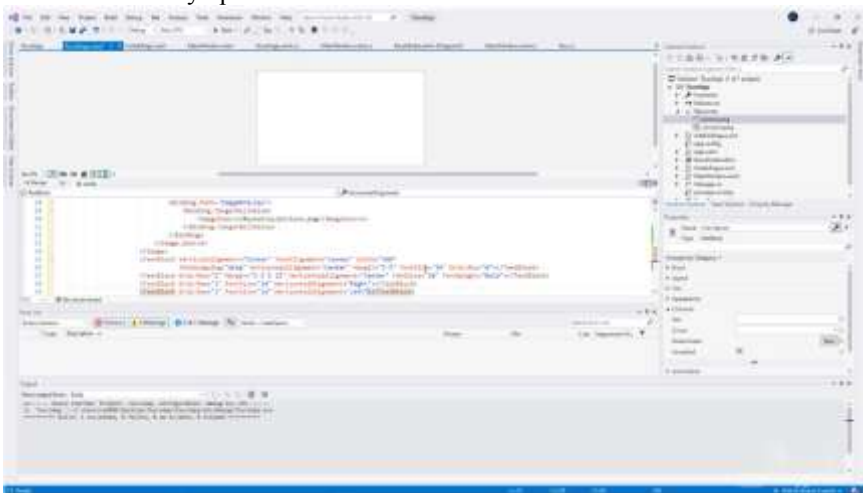
Добавляем изображение тура в верстку, установив необходимые атрибуты



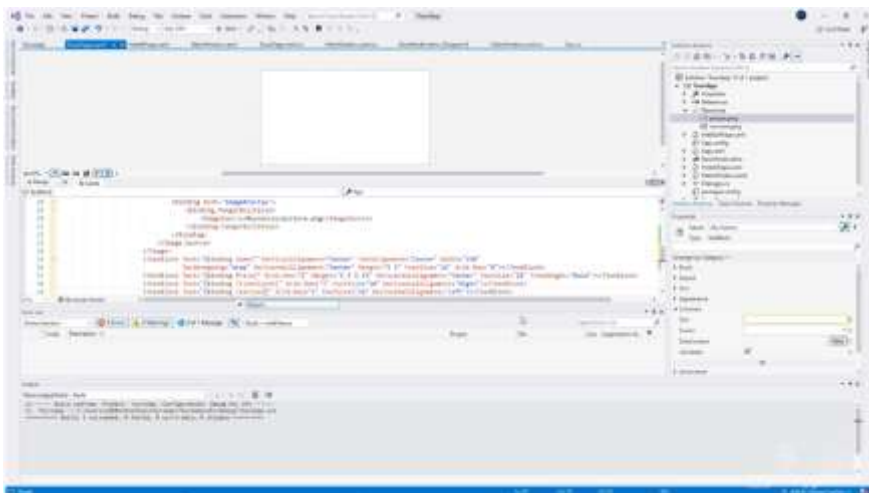
Теперь задаем объекты для привязки, а также изображение, если изображения тура не будет



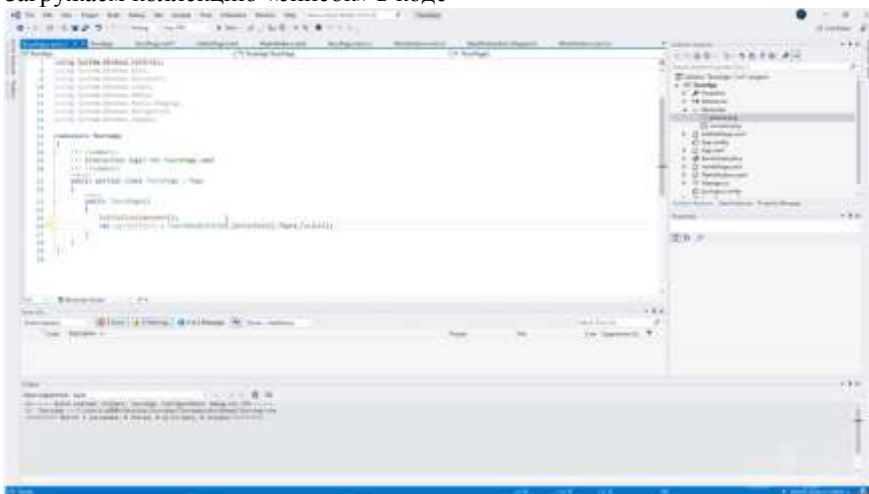
Добавляем текстовые поля для наименования, указав: перенос текста, выравнивание, большой шрифт; для стоимости, указав: выравнивание, также большой шрифт, но делаем текст жирным; количество билетов – выравниваем по правому краю; и актуальности – выравниваем по левому краю



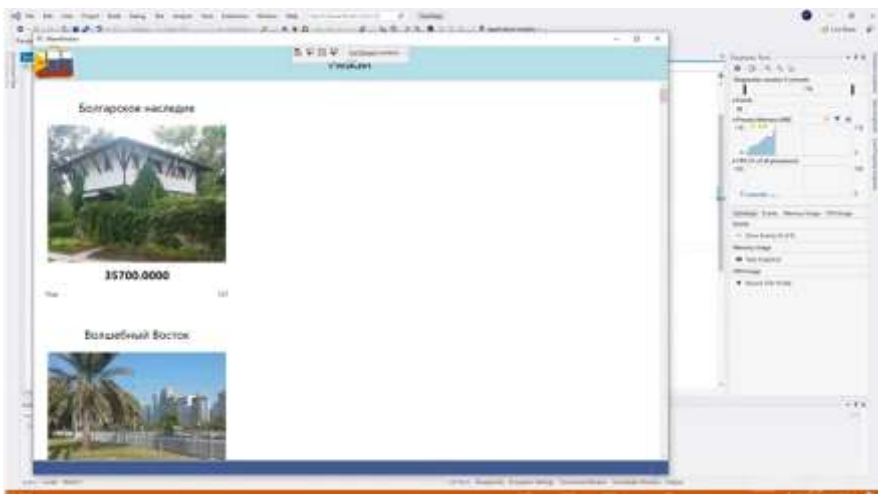
Добавляем привязки для наименования, стоимости, количества билетов и актуальности.



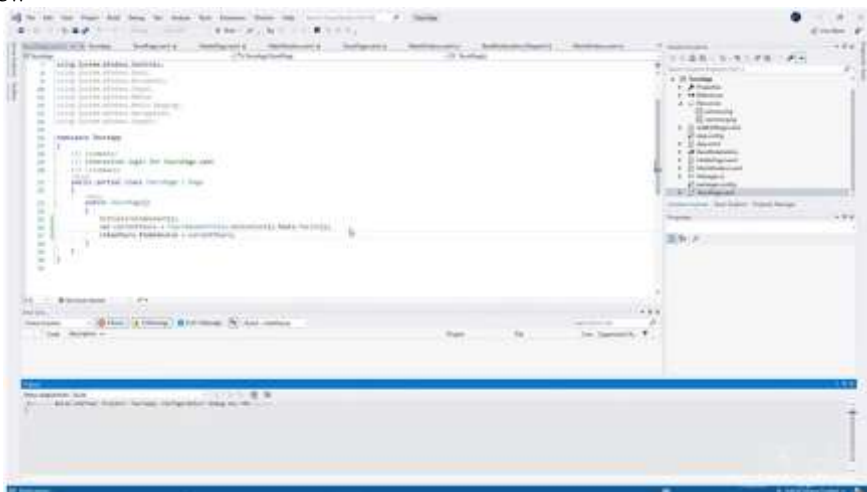
Загружаем коллекцию «список» в коде



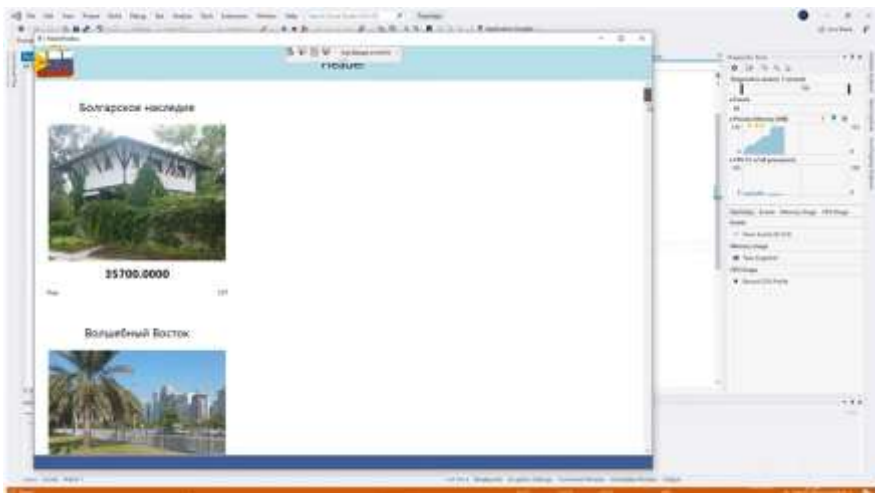
Далее – Далее – Финиш. 252 строки было импортировано. Можем их увидеть в таблице



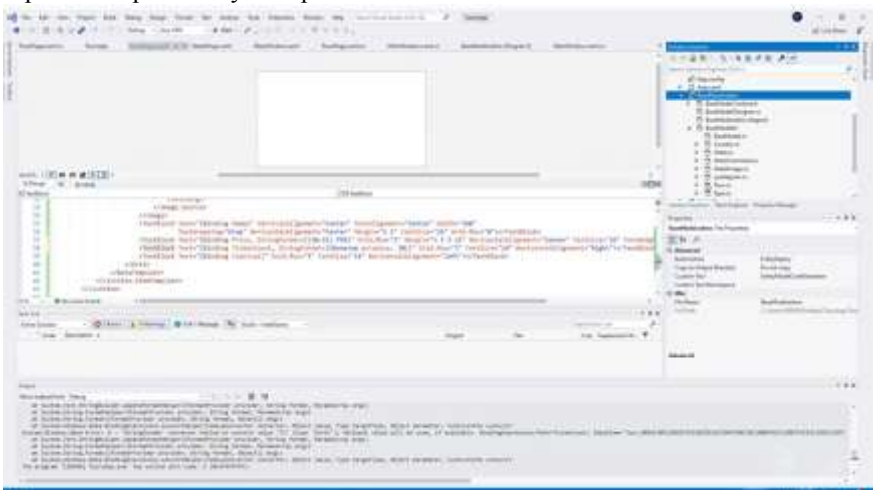
В чем у нас ошибка? Мы не загрузили туры в список – наш ListView



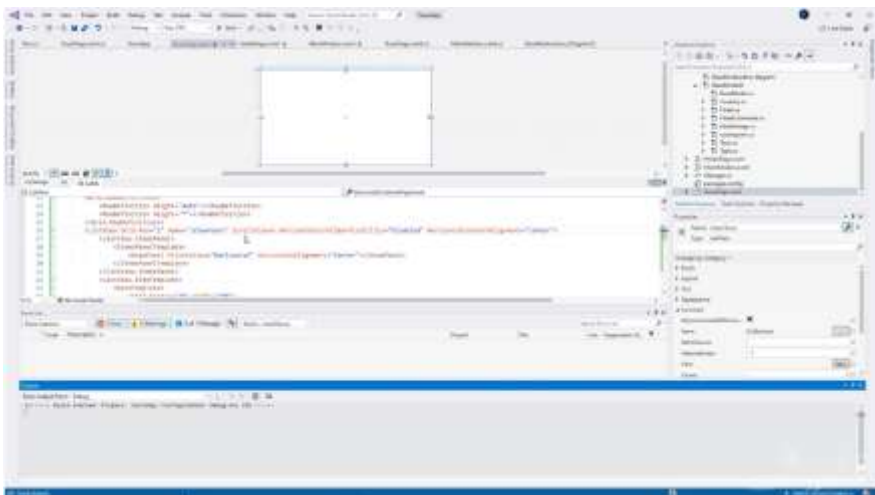
Запускаем приложение еще раз



Работаем над форматом отображения стоимости и количества билетов. Указываем им `StringFormat` формат для указания цены (два знака после запятой) и примечание к количеству билетов. Делать это можно прямо во время запуска приложения



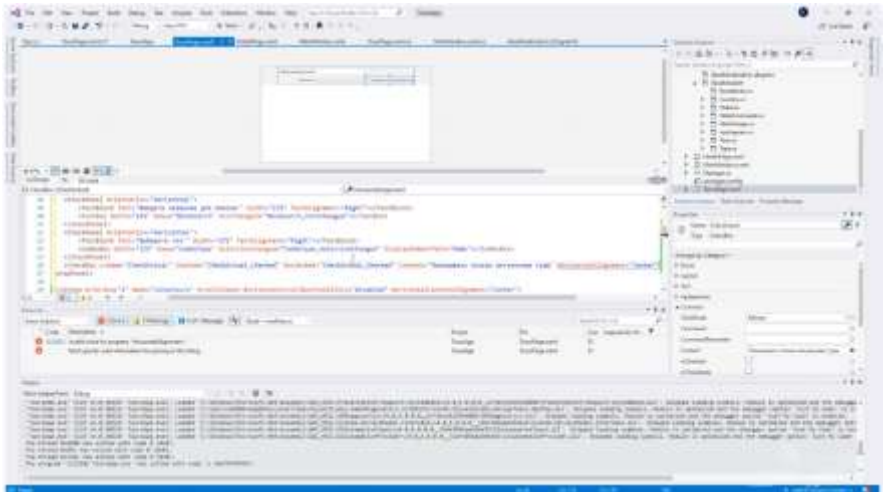
Для отображения актуальности делаем дополнительное свойство в классе `Тур` – назовем его `ActualText`



Реализация поиска и фильтрации информации

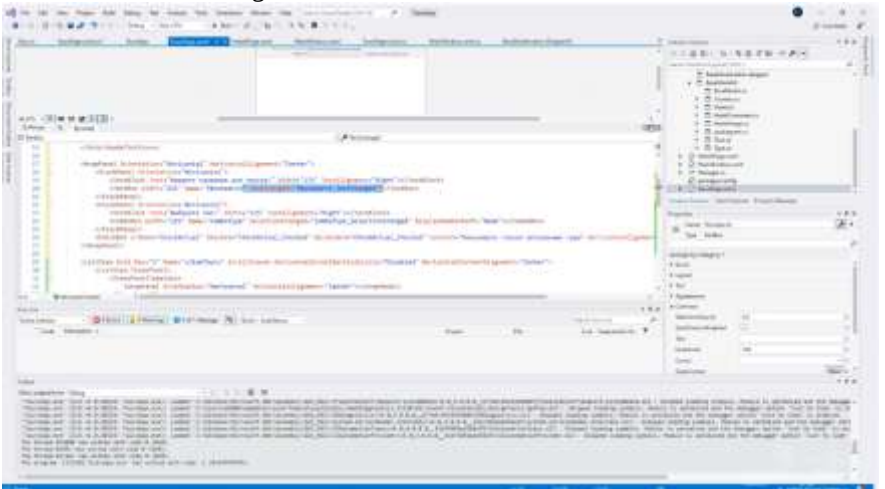
Как говорилось ранее, для работы с большими объемами информации полезно реализовать поиск и фильтрацию. Как это работает? Пользователь вводит в специальные элементы управления данные для поиска или выбирает категории из списка для фильтрации. Затем в коде разработчик приводит коллекцию данных к виду, который соответствует поиску, и загружает результаты в ListView. Давайте сделаем это.

Для начала необходимо подготовить внешний вид страницы. Добавляем элементы для поиска и фильтрации. Для поиска это будет TextBox, для фильтрации – ComboBox, который представляет собой выпадающий список объектов. Обязательно даем подсказки, чтобы пользователь знал, что именно вводить. И устанавливаем атрибуты, например, на выравнивание и размеры. Также указываем, какое поле отображать для ComboBox, и добавляем Checkbox для отображения только актуальных туров. Делать это мы будем также в RowPanel, чтобы переносить элементы при изменениях размера экрана

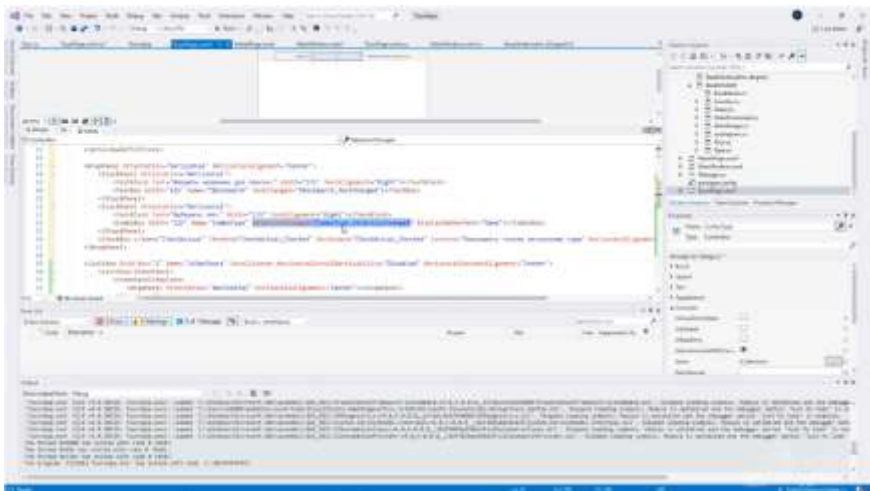


В плане удобства использования наиболее выигрышно выглядит механизм, когда результаты поиска выводятся сразу по мере ввода ключевого слова или выбора значения в выпадающем списке. Поэтому мы обработаем соответствующие события на каждый элемент.

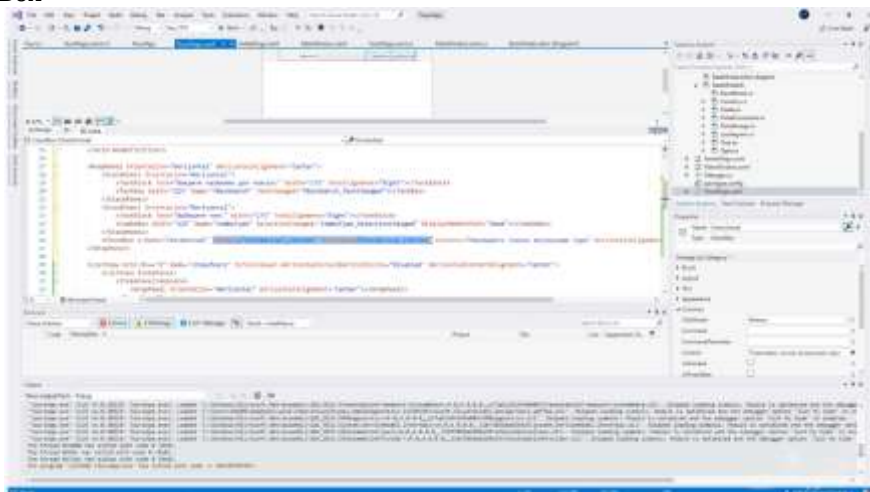
Во-первых, это TextChanged на изменение текста для поиска. TextBox – Search – TextChanged.



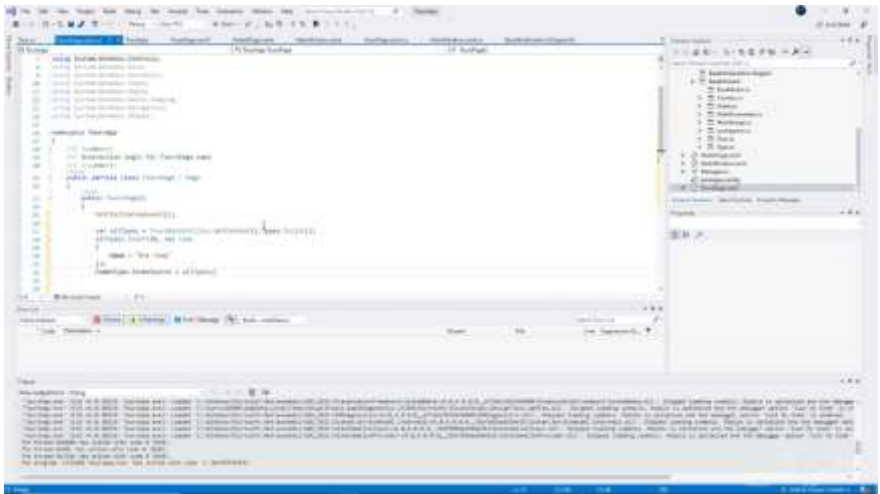
Затем при изменении выбора в выпадающем списке – Selection Changed



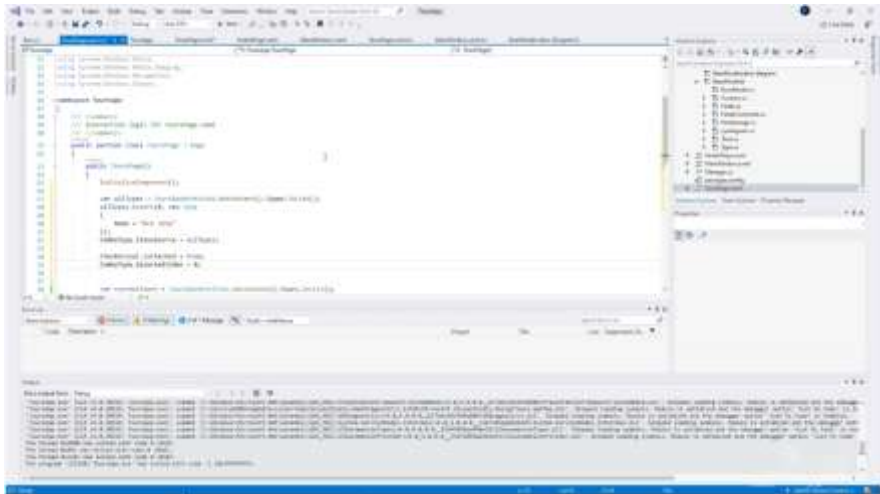
После этого мы добавляем обработку нажатия и снятия флажка у CheckBox



Загружаем данные в ComboBox, добавив элемент «все типы». Делается это в коде

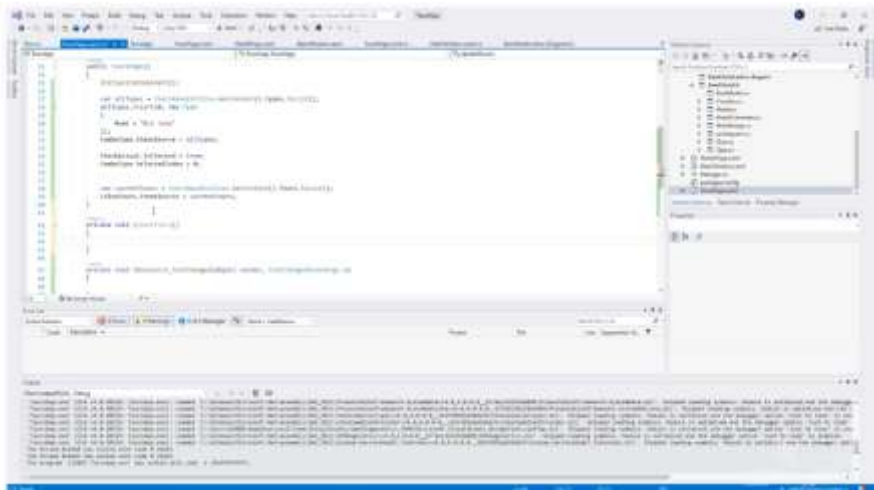


Устанавливаем стартовые значения для элементов управления: CheckBox и ComboBox.

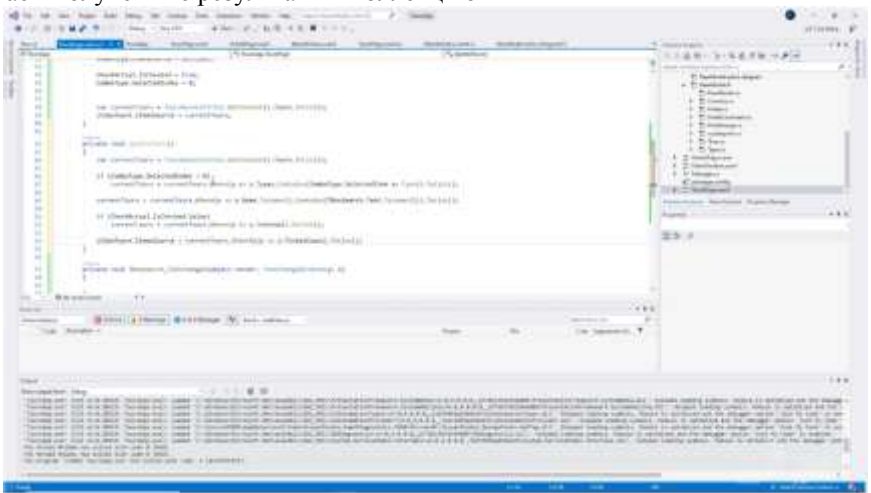


Обработываем методы поиска и фильтрации так, чтобы они работали вместе. Для этого в одном методе выполним фильтрацию коллекции, поиск по ней и сортировку, а затем вызываем этот метод из обработчиков событий всех наших элементов управления

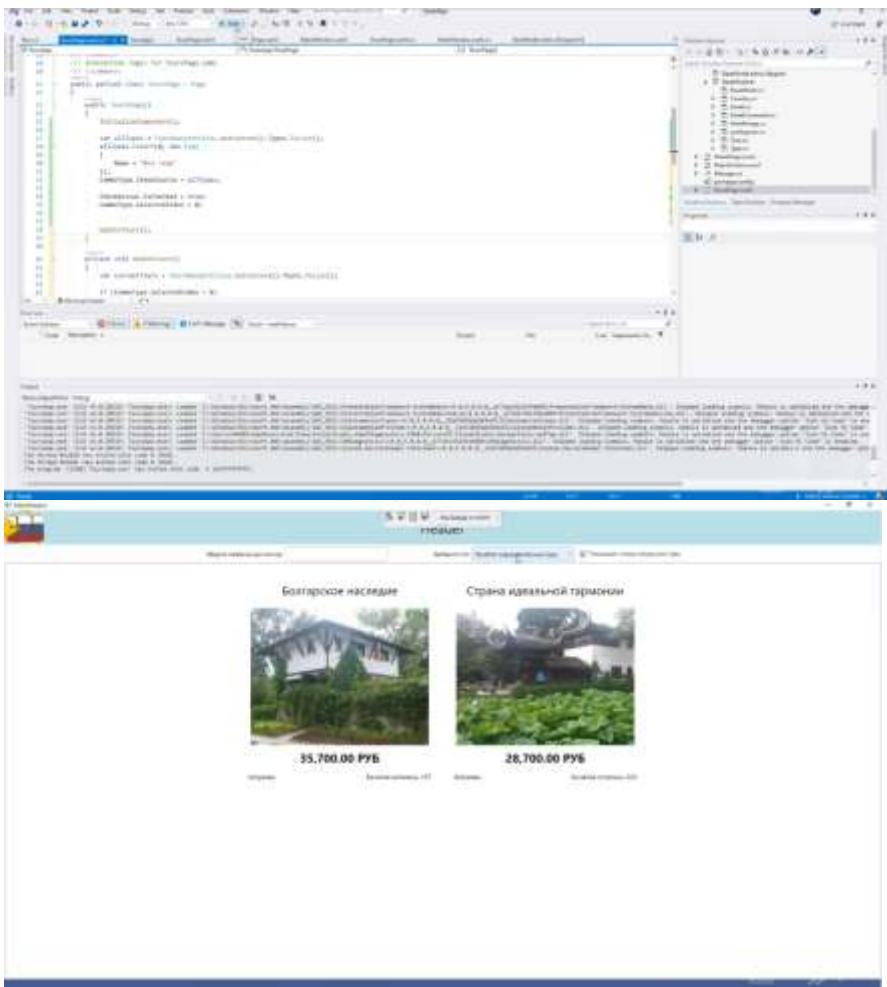
Для начала создаем метод UpdateTours()



Выполняем фильтрацию, поиск и сортировку. После этого загружаем полученные результаты в коллекцию



Вызываем этот метод в каждом обработчике элементов управления и при запуске страницы



Графическое представление

Демонстрация работы с графиками в Windows Forms

На данном занятии будет разработано простое приложение Windows Forms для визуализации расходов пользователей. Пользователи распределяют затраты по разным категориям и хранят их в общей базе данных. Итогом работы приложения будет служить работающая форма, в которой для каждого пользователя можно построить диаграммы

различных типов для визуализации их расходов по категориям. Основные шаги построения приложения:

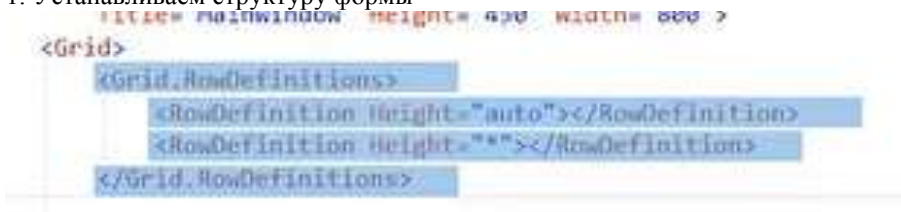
1. Разработка интерфейса приложения
2. Настройка взаимодействия с базой данных
3. Реализация обработки данных

Важно

В рамках примера используется готовая база данных с информацией о пользователях, их платежах и категориях расходов

Разработка интерфейса приложения

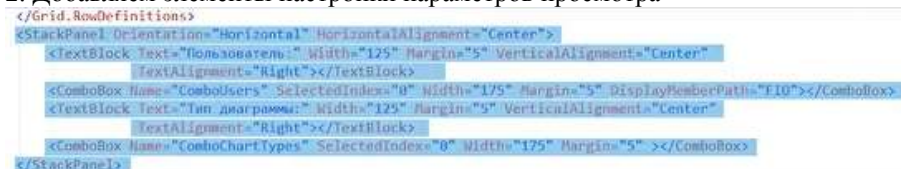
1. Устанавливаем структуру формы



Важно

Интерфейс приложения будет состоять из двух основных частей: области построения и области настройки параметров просмотра

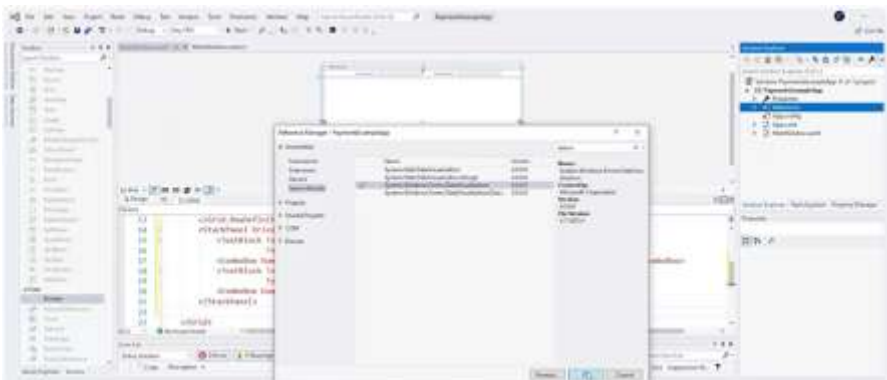
2. Добавляем элементы настройки параметров просмотра



Важно

Элементами настройки параметров просмотра будут являться выпадающие списки, позволяющие выбрать пользователя и тип диаграммы

3. Подключаем библиотеки для просмотра диаграмм



Важно

Диаграмма будет визуализироваться с помощью элемента Chart из WindowsForms. Воспользоваться данным элементом можно после подключения библиотеки System.Windows.Forms.DataVisualization, расположенного во вкладке Assemblies (сборки)

4. Добавляем элементы управления диаграммой

```
</Grid.RowDefinitions>
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
  <TextBlock Text="Пользователи:" Width="125" Margin="5" VerticalAlignment="Center"
    TextAlignment="Right"></TextBlock>
  <ComboBox Name="ComboUsers" SelectedIndex="0" Width="175" Margin="5" DisplayMemberPath="FIO"></ComboBox>
  <TextBlock Text="Тип диаграммы:" Width="125" Margin="5" VerticalAlignment="Center"
    TextAlignment="Right"></TextBlock>
  <ComboBox Name="ComboChartTypes" SelectedIndex="0" Width="175" Margin="5" ></ComboBox>
</StackPanel>
</WindowsFormsHost />
```

Важно

Диаграмма будет располагаться внутри элемента WindowsFormsHost. Данный элемент добавляется из окна ToolBox простым перетаскиванием

5. Добавляем пространство имен для работы с элементом Chart

```
x:Class="PaymentsExampleApp.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:clr-namespace="System.Windows.Forms.DataVisualization"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800"
>
<Grid.RowDefinitions>
```

6. Добавляем дополнительные параметры просмотра

```

</StackPanel>
<WindowsFormsHost Grid.Row="1" Margin="5">
  <charts:Chart x:Name="ChartPayments">
    <charts:Chart.Legends>
      <charts:Legend>
        </charts:Legend>
      </charts:Chart.Legends>
    </charts:Chart>
  </WindowsFormsHost>

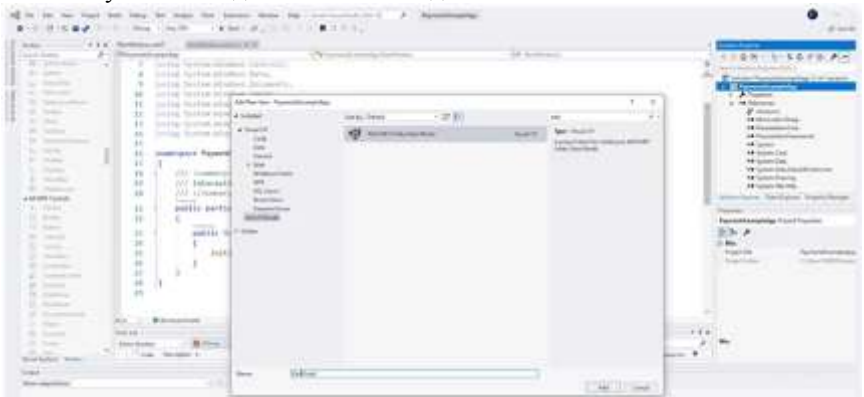
```

Важно

Дополнительными параметрами являются имя диаграммы, а также легенда

Настройка взаимодействия с базой данных

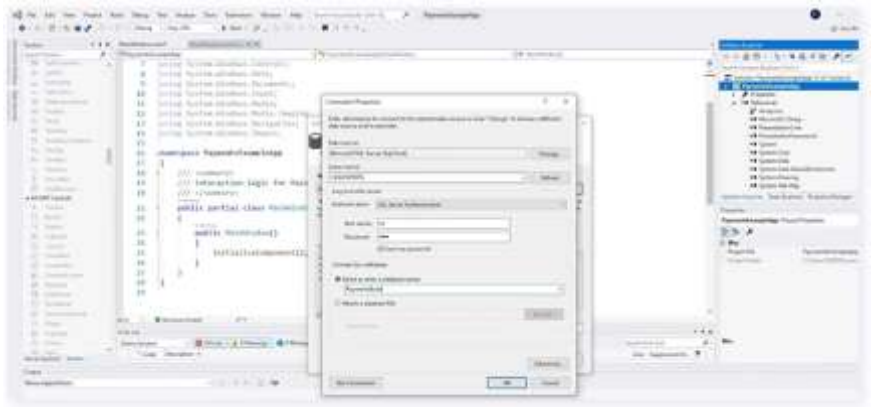
1. Реализуем взаимодействие с базой данных



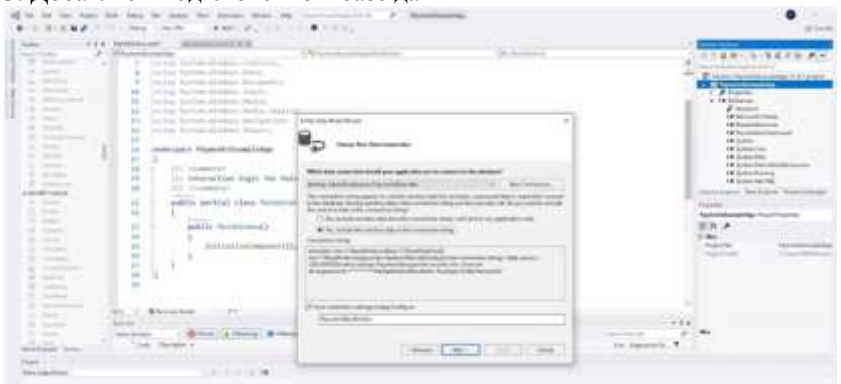
Важно

Взаимодействие реализуется путем добавления элемента «ADO.NET Entity Data Model».

2. Настраиваем свойства подключения базы данных



3. Добавляем подключение к базе данных



Реализация обработки данных

1. Создаем область построения графиков

```
MainWindow.xaml | MainWindow.xaml.cs | PaymentsExampleApp-MainWindow
PaymentsExampleApp
17 namespace PaymentsExampleApp
18 {
19     /// <summary>
20     /// Interaction logic for MainWindow.xaml
21     /// </summary>
22     <!--
23     </-->
24     public partial class MainWindow : Window
25     {
26         private PaymentsBaseEntities _context = new PaymentsBaseEntities();
27
28         public MainWindow()
29         {
30             InitializeComponent();
31             ChartPayments.ChartAreas.Add(new ChartArea("Main"));
32         }
33     }
34 }
```

Важно

Сперва создается поле для контекста EntityFramework с инициализацией. Затем создается область построения диаграммы ChartArea и добавляется в соответствующую коллекцию в конструкторе MainWindow

2. Добавляем наборы данных

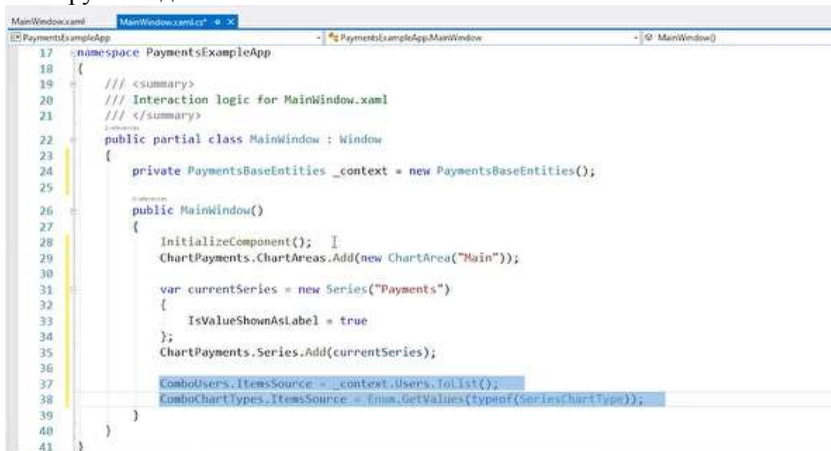
```
MainWindow.xaml | MainWindow.xaml.cs | PaymentsExampleApp-MainWindow | MainWindow()
PaymentsExampleApp
17 namespace PaymentsExampleApp
18 {
19     /// <summary>
20     /// Interaction logic for MainWindow.xaml
21     /// </summary>
22     <!--
23     </-->
24     public partial class MainWindow : Window
25     {
26         private PaymentsBaseEntities _context = new PaymentsBaseEntities();
27
28         public MainWindow()
29         {
30             InitializeComponent();
31             ChartPayments.ChartAreas.Add(new ChartArea("Main"));
32
33             var currentSeries = new Series("Payments");
34             {
35                 IsValueShownAsLabel = true;
36             }
37             ChartPayments.Series.Add(currentSeries);
38         }
39     }
40 }
```

Важно

Для каждого набора данных (например, данные линии на графике) необходимо добавлять в коллекцию Series. В данном случае есть одна

серия данных для отображения сумм платежей по категориям. Объект Series создается с указанием названия и необходимости отображения на диаграмме

3. Загружаем данные из базы



```
17 namespace PaymentsExampleApp
18 {
19     /// <summary>
20     /// Interaction logic for MainWindow.xaml
21     /// </summary>
22     public partial class MainWindow : Window
23     {
24         private PaymentsBaseEntities _context = new PaymentsBaseEntities();
25
26         public MainWindow()
27         {
28             InitializeComponent();
29             ChartPayments.ChartAreas.Add(new ChartArea("Main"));
30
31             var currentSeries = new Series("Payments")
32             {
33                 IsValueShownAsLabel = true
34             };
35             ChartPayments.Series.Add(currentSeries);
36
37             ComboUsers.ItemsSource = _context.Users.ToList();
38             ComboChartTypes.ItemsSource = Enum.GetValues(typeof(SeriesChartType));
39         }
40     }
41 }
```

Важно

Данные о пользователях загружаются в выпадающий список. Также загружаются типы диаграммы из перечисления SeriesChartType

4. Реализуем адаптивное изменение интерфейса



```
<TextBlock Text="Пользователи:" Width="125" Margin="5" VerticalAlignment="Center"
TextAlignment="Right"></TextBlock>
<ComboBox Name="ComboUsers" SelectionChanged="UpdateChart" SelectedIndex="0" Width="175" Margin="5" DisplayMemberBinding="{Binding Name}"></ComboBox>
<TextBlock Text="Тип диаграммы:" Width="125" Margin="5" VerticalAlignment="Center"
TextAlignment="Right"></TextBlock>
<ComboBox Name="ComboChartTypes" SelectionChanged="UpdateChart" SelectedIndex="0" Width="175" Margin="5"></ComboBox>
</StackPanel>
<WindowsFormsHost Grid.Row="1" Margin="5">
<charts:Chart x:Name="ChartPayments">
<charts:Chart.Legends>
<charts:Legend>
```

Важно

При выборе другого значения в ComboBox будет вызываться метод UpdateChart()

5. Получаем выбранные значения в выпадающих списках

```

        ComboUsers.ItemsSource = _context.Users.ToList();
        ComboChartTypes.ItemsSource = Enum.GetValues(typeof(SeriesChartType));
    }

    private void UpdateChart(object sender, SelectionChangedEventArgs e)
    {
        if (ComboUsers.SelectedItem is User currentUser &&
            ComboChartTypes.SelectedItem is SeriesChartType currentType)
        {
        }
    }
}

```

Важно

Значения получаются как currentUser и currentType

6. Обрабатываем данные диаграммы

```

private void UpdateChart(object sender, SelectionChangedEventArgs e)
{
    if (ComboUsers.SelectedItem is User currentUser &&
        ComboChartTypes.SelectedItem is SeriesChartType currentType)
    {
        Series currentSeries = ChartPayments.Series.FirstOrDefault();
        currentSeries.ChartType = currentType;
        currentSeries.Points.Clear();
    }
}

```

Важно

Приведенный код описывает получение серии данных диаграммы из соответствующей коллекции Series, установку типа диаграммы и очистку предыдущих данных

7. Обрабатываем список категорий

```

if (ComboUsers.SelectedItem is User currentUser &&
    ComboChartTypes.SelectedItem is SeriesChartType currentType)
{
    Series currentSeries = ChartPayments.Series.FirstOrDefault();
    currentSeries.ChartType = currentType;
    currentSeries.Points.Clear();

    var categoriesList = _context.Categories.ToList();
    foreach (var category in categoriesList)
    {
        currentSeries.Points.AddXY(category.Name,
            _context.Payments.ToList().Where(p => p.User == currentUser
                && p.Category == category).Sum(p => p.Price * p.Num));
    }
}

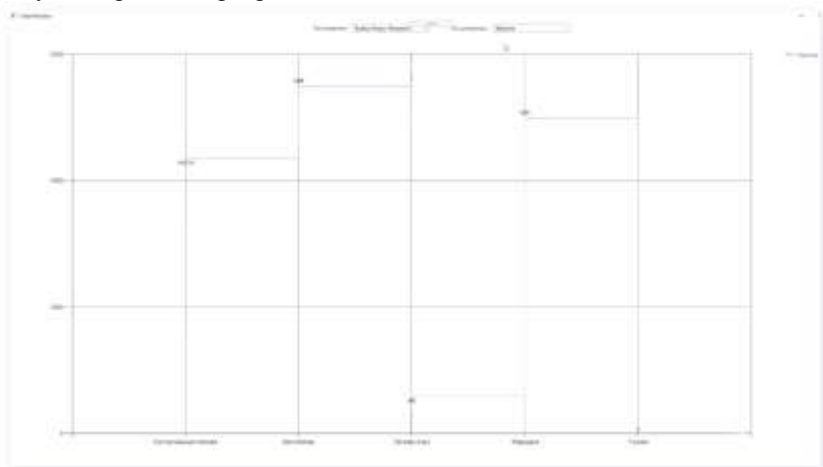
```

Важно

Список категорий получается из базы данных. Далее, в цикле foreach для каждой категории значение точки диаграммы добавляется в

Points. Координата X будет названием категории, а координата Y будет суммой платежа для выбранного пользователя в текущей категории

Результат работы программы



Формирование отчетов в Excel

Демонстрация работы с таблицами Excel в WPF

На данном занятии будет реализована возможность экспорта данных из приложения для визуализации расходов пользователей в таблицу Excel. Расходы каждого пользователя будут экспортироваться в отдельный лист, названием которого будет ФИО пользователя. Расходы будут распределены по категориям, причем по каждой категории будут указываться общие затраты. Основные шаги построения приложения:

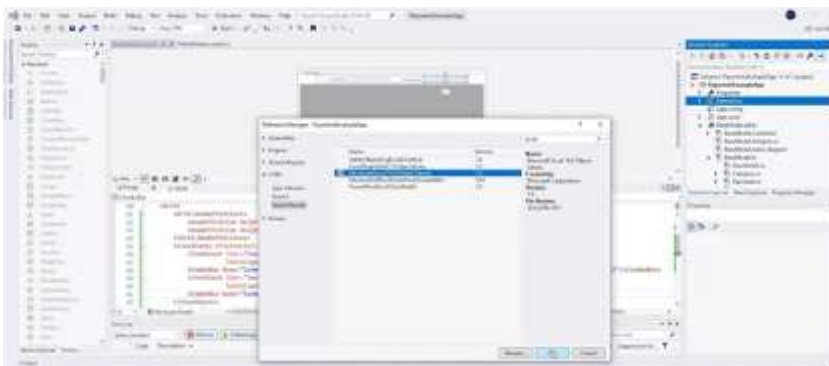
Предварительные шаги

Реализация экспорта

Проверка корректной работы приложения

Предварительные шаги

1. Подключаем библиотеку для работы с Excel



Важно

Для экспорта данных в Excel используется библиотека Interop.Excel (Object library), расположенная во вкладке COM

2. Добавляем кнопку экспорта



Важно

Экспорт данных в Excel будет осуществляться с помощью кнопки «Экспорт в Excel»

3. Подключаем пространство имен для работы с Excel

```

7 using System.Windows.Controls;
8 using System.Windows.Data;
9 using System.Windows.Documents;
10 using System.Windows.Forms.DataVisualization.Charting;
11 using System.Windows.Input;
12 using System.Windows.Media;
13 using System.Windows.Media.Imaging;
14 using System.Windows.Navigation;
15 using System.Windows.Shapes;
16 using Excel = Microsoft.Office.Interop.Excel;
17
18 namespace PaymentsExampleApp
19 {
20     /// <summary>
21     /// Interaction logic for MainWindow.xaml
22     /// </summary>
23     [assembly: ThemeResource("Resources/Theme.xaml")]
24     public partial class MainWindow : Window
25     {
26         private PaymentsBaseEntities _context = new PaymentsBaseEntities();
27
28         public MainWindow()
29         {
30             InitializeComponent();
31             ChartPayments.ChartAreas.Add(new ChartArea("Main"));
32
33             var currentSeries = new Series("Payments")
34             {
35                 IsValueShownAsLabel = true
36             };
37             ChartPayments.Series.Add(currentSeries);
38         }
39     }
40 }

```

Важно

Требуемое пространство имен подключается с помощью директивы using

Реализация экспорта

1. Получаем список пользователей

```

private void ComboBox_SelectedIndexChanged(object sender, RoutedEventArgs e)
{
    ComboBoxTypes.SelectedItem is SeriesChartType currentType

    Series currentSeries = ChartPayments.Series.FirstOrDefault();
    currentSeries.ChartType = currentType;
    currentSeries.Points.Clear();

    var categoriesList = _context.Categories.ToList();
    foreach (var category in categoriesList)
    {
        currentSeries.Points.AddXY(category.Name,
            _context.Payments.ToList().Where(p => p.User == currentUser
            && p.Category == category).Sum(p => p.Price * p.Num));
    }
}

private void BtnExportToExcel_Click(object sender, RoutedEventArgs e)
{
    var allUsers = _context.Users.ToList().OrderBy(p => p.FIO).ToList();
}

```

Важно

Список пользователей выгружается из базы данных, причем сразу производится сортировка по ФИО

2. Создаем новую книгу Excel

```

private void BtnExportToExcel_Click(object sender, RoutedEventArgs e)
{
    var allUsers = _context.Users.ToList().OrderBy(p => p.FIO).ToList();

    var application = new Excel.Application();
    application.SheetsInNewWorkbook = allUsers.Count();

    Excel.Workbook workbook = application.Workbooks.Add(Type.Missing);
}

```

Важно

Объявляем переменную с приложением Excel, указывая количество листов (sheets) равным количеству пользователей в БД. Также добавляем рабочую книгу (workbook)

3. Называем листы

```

Excel.Workbook workbook = application.Workbooks.Add(Type.Missing);

int startRowIndex = 1;

for (int i = 0; i < allUsers.Count(); i++)
{
    Excel.Worksheet worksheet = application.Worksheets.Item[i + 1];
    worksheet.Name = allUsers[i].FIO;
}

```

Важно

В цикле по списку пользователей выбирается текущий лист. Текущему листу присваивается ФИО текущего пользователя. Следует обратить внимание, что строки в Excel начинаются с 1, потому счетчик строк startRowIndex=1

4. Добавляем название колонок

```

worksheet.Cells[1][startRowIndex] = "Дата платежа";
worksheet.Cells[2][startRowIndex] = "Назначение";
worksheet.Cells[3][startRowIndex] = "Сумма";
worksheet.Cells[4][startRowIndex] = "Количество";
worksheet.Cells[5][startRowIndex] = "Сумма";

startRowIndex++;

```

Важно

Название столбцов добавляется в верхнюю строчку листа, после чего увеличивается значение счетчика startRowIndex. Следует обратить внимание, что при обращении к ячейке сначала указывается номер ее столбца, а затем – номер строки

5. Группируем платежи по категориям

```
var userCategories = AllUsers[1].Payments.OrderBy(p => p.Date).GroupBy(p => p.Category).OrderBy(g => g.Key.Name);
```

Важно

Платежи текущего пользователя группируются с помощью GroupBy и сортируются по дате и названию категории

6. Настраиваем отображение названий категорий

```
foreach (var groupCategory in userCategories)
{
    Excel.Range headerRange = worksheet.Range[worksheet.Cells[1][startRowIndex], worksheet.Cells[5][startRowIndex]];
    headerRange.Merge();
    headerRange.Value = groupCategory.Key.Name;
    headerRange.HorizontalAlignment = Excel.XlHAlign.XlHAlignCenter;
    headerRange.Font.Italic = true;

    startRowIndex++;
}
```

Важно

Название каждой категории помещается в объединенную ячейку, выравнивается по центру и отображается курсивом. Далее идет переход к следующей строке

7. Добавляем информацию о платежах

```
foreach (var payment in groupCategory)
{
    worksheet.Cells[1][startRowIndex] = payment.Date.ToString("dd.MM.yyyy HH:mm");
    worksheet.Cells[2][startRowIndex] = payment.Name;
    worksheet.Cells[3][startRowIndex] = payment.Price;
    worksheet.Cells[4][startRowIndex] = payment.Num;
}
```

Важно

В цикле по платежам для каждой категории заносим в ячейки таблицы текущей строки дату платежа (в заданном формате), наименование платежа, цену и количество платежей данного типа

8. Рассчитываем величину платежа каждой категории

```
foreach (var payment in groupCategory)
{
    worksheet.Cells[1][startRowIndex] = payment.Date.ToString("dd.MM.yyyy HH:mm");
    worksheet.Cells[2][startRowIndex] = payment.Name;
    worksheet.Cells[3][startRowIndex] = payment.Price;
    worksheet.Cells[4][startRowIndex] = payment.Num;

    worksheet.Cells[5][startRowIndex].Formula = "=" & (startRowIndex + 1) * (startRowIndex + 2);

    worksheet.Cells[3][startRowIndex].NumberFormat =
        worksheet.Cells[3][startRowIndex].NumberFormat + "x,###.00";

    startRowIndex++;
}
```

Важно

Чтобы Excel автоматически пересчитывал сумму платежа при изменении количества или цены платежа, следует рассчитывать сумму не

в коде, а прямо в ячейке Excel, добавляя туда формулу для расчета. Также для денежных значений можно установить числовой формат

9. Добавляем название к ячейкам для хранения общих затрат

```
foreach (var payment in groupCategory)
{
    worksheet.Cells[1][startRowIndex] = payment.Date.ToString("dd.MM.yyyy HH:mm");
    worksheet.Cells[2][startRowIndex] = payment.Name;
    worksheet.Cells[3][startRowIndex] = payment.Price;
    worksheet.Cells[4][startRowIndex] = payment.Num;

    worksheet.Cells[5][startRowIndex].Formula = "$=C{startRowIndex}*D{startRowIndex}";

    worksheet.Cells[3][startRowIndex].NumberFormat =
        worksheet.Cells[3][startRowIndex].NumberFormat = "#,###.00";

    startRowIndex++;
}

Excel.Range sumRange = worksheet.Range(worksheet.Cells[4][startRowIndex], worksheet.Cells[4][startRowIndex]);
sumRange.Merge();
sumRange.Value = "ИТОГО:";
sumRange.HorizontalAlignment = Excel.XlHAlign.XlHAlignRight;
```

Важно

Название ячейки Итого помещается в объединенную ячейку (1–4 столбец)

10. Рассчитываем величину общих затрат

```
worksheet.Cells[1][startRowIndex] = payment.Date.ToString("dd.MM.yyyy HH:mm");
worksheet.Cells[2][startRowIndex] = payment.Name;
worksheet.Cells[3][startRowIndex] = payment.Price;
worksheet.Cells[4][startRowIndex] = payment.Num;

worksheet.Cells[5][startRowIndex].Formula = "$=C{startRowIndex}*D{startRowIndex}";

worksheet.Cells[3][startRowIndex].NumberFormat = "#,###.00";
worksheet.Cells[3][startRowIndex].NumberFormat = "#,###.00";

startRowIndex++;

Excel.Range sumRange = worksheet.Range(worksheet.Cells[1][startRowIndex], worksheet.Cells[4][startRowIndex]);
sumRange.Merge();
sumRange.Value = "ИТОГО:";
sumRange.HorizontalAlignment = Excel.XlHAlign.XlHAlignRight;

worksheet.Cells[5][startRowIndex].Formula = "$=SUM(E{startRowIndex : groupCategory.Count()}: " +
    "$" + E{startRowIndex - 1});

sumRange.Font.Bold = worksheet.Cells[5][startRowIndex].Font.Bold = true;
worksheet.Cells[5][startRowIndex].NumberFormat = "#,###.00";
```

Важно

Для расчета начального значения диапазона ячеек, учитываемого при расчете, из номера текущей строки вычитается общее количество платежей в рамках категории. Далее величина итоговой суммы выделяется жирным шрифтом и к ней применяется денежный формат

11. Завершаем оформление таблицы и реализацию приложения

```

Excel.Range sumRange = worksheet.Range[worksheet.Cells[1][startRowIndex], worksheet.Cells[4][startRowIndex]];
sumRange.Merge();
sumRange.Value = "ИТОГО:";
sumRange.HorizontalAlignment = Excel.XlHAlign.XlHAlignRight;

worksheet.Cells[5][startRowIndex].Formula = "$=SUM(E{startRowIndex - groupCategory.Count()}: " +
    $"E{startRowIndex - 1})";

sumRange.Font.Bold = worksheet.Cells[5][startRowIndex].Font.Bold = true;
worksheet.Cells[5][startRowIndex].NumberFormat = "#,###.00";

startRowIndex++;

Excel.Range rangeBorders = worksheet.Range[worksheet.Cells[1][1], worksheet.Cells[5][startRowIndex - 1]];
rangeBorders.Borders[Excel.XlBordersIndex.xlEdgeBottom].LineStyle =
rangeBorders.Borders[Excel.XlBordersIndex.xlEdgeLeft].LineStyle =
rangeBorders.Borders[Excel.XlBordersIndex.xlEdgeRight].LineStyle =
rangeBorders.Borders[Excel.XlBordersIndex.xlEdgeTop].LineStyle =
rangeBorders.Borders[Excel.XlBordersIndex.xlInsideHorizontal].LineStyle =
rangeBorders.Borders[Excel.XlBordersIndex.xlInsideVertical].LineStyle = Excel.XlLineStyle.xlContinuous;

worksheet.Columns.AutoFit();
}
application.Visible = true;
}

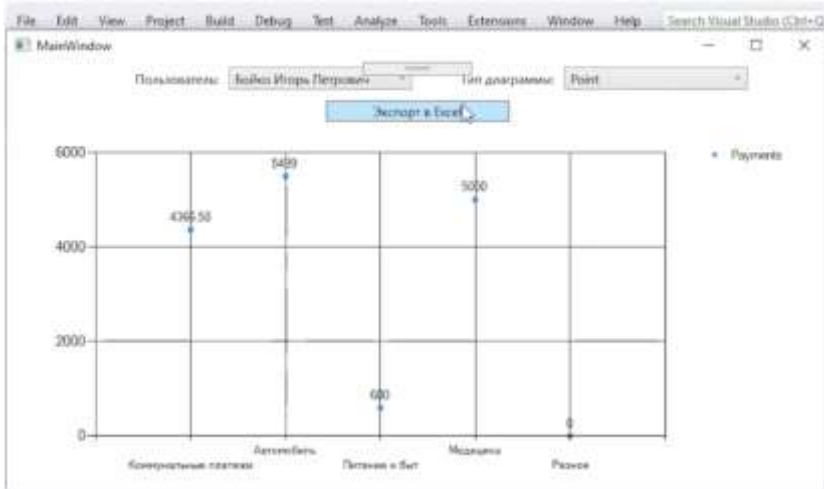
```

Важно

Оформление включает в себя: добавление границ (внешних и внутренних), установку автоширины всех столбцов листа. Последняя строчка разрешает отобразить таблицу по завершении экспорта

Проверка корректной работы приложения

1. Делаем экспорт данных



Важно

Экспорт данных производится по нажатию на кнопку «Экспорт в Excel»

2. Проводим анализ корректности работы

Дата платежа	Название	Стоимость	Количество	Сумма
Автомобиль				
01.01.2015 00:00	Взнос за гараж	5,000.00	1	5000
Итого:				5,000.00

Важно

На всех листах, кроме первого, видна ошибка с индексацией: значения вставлены не в те строки

3. Устраняем ошибки

```

var allUsers = _context.Users.ToList().OrderBy(p => p.FIO).ToList();

var application = new Excel.Application();
application.SheetsInNewWorkbook = allUsers.Count();

Excel.Workbook workbook = application.Workbooks.Add(Type.Missing);

for (int i = 0; i < allUsers.Count(); i++)
{
    int startRowIndex = 1;

    Excel.Worksheet worksheet = application.Worksheets.Item[i + 1];
    worksheet.Name = allUsers[i].FIO;

    worksheet.Cells[1][startRowIndex] = "Дата платежа";
    worksheet.Cells[2][startRowIndex] = "Название";
    worksheet.Cells[3][startRowIndex] = "Стоимость";
    worksheet.Cells[4][startRowIndex] = "Количество";
    worksheet.Cells[5][startRowIndex] = "Сумма";

    startRowIndex++;

    var usersCategories = allUsers[i].Payments.OrderBy(p => p.Date).GroupBy(p => p.Category);
    foreach (var groupCategory in usersCategories)

```

Важно

Ошибка состояла в том, что счетчик текущей строки был задан перед циклом по всем пользователям. Для устранения ошибки достаточно перенести определение счетчика внутрь этого цикла

4. Проверяем результат

	А	В	С	Д	Е
1	Дата платежа	Название	Стоимость	Количество	Сумма
2	Автомобиль				
3	01.03.2015 00:00	Взнос за гараж	5,000.00	1	5000
4	ИТОГО:				5,000.00
5					
6					

Формирование отчетов в Word

Демонстрация работы с документами Word в WPF

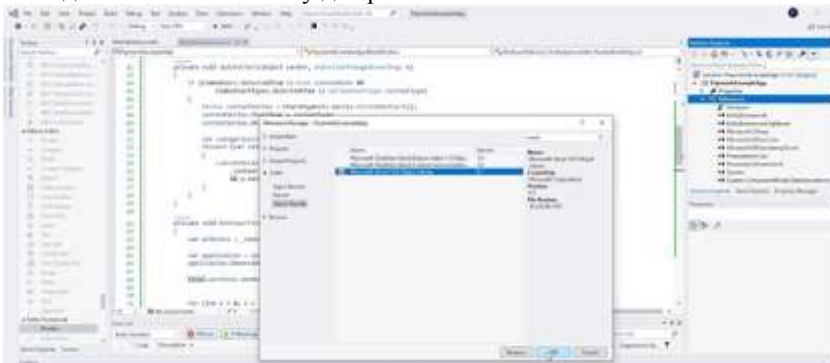
На данном занятии будет реализована возможность экспорта данных из приложения для визуализации расходов пользователей в документ Word. Расходы каждого пользователя будут экспортироваться на отдельную страницу, названием которой будет ФИО пользователя. Расходы будут просуммированы по категориям и представлены в виде

таблицы. Под таблицей будет размещена информация о максимальном и минимальном платежах данного пользователя. Основные шаги построения приложения:

1. Подготовительный этап
2. Реализация экспорта в документ Word
3. Завершение оформления документа Word

Подготовительный этап

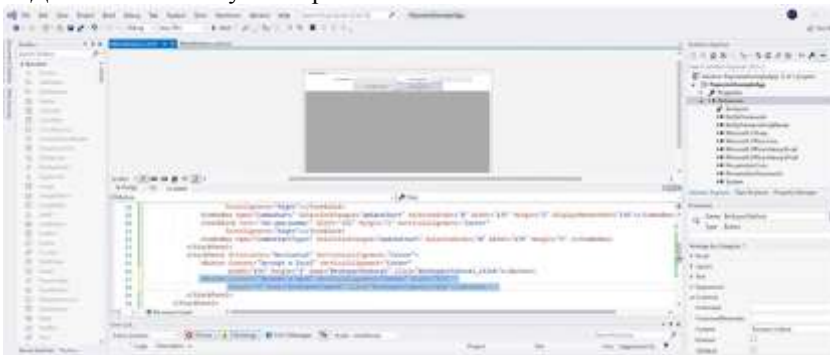
1. Подключаем библиотеку для работы с Word



Важно

Для экспорта данных в Word используется библиотека InteropWord (Object Library), расположенная во вкладке COM

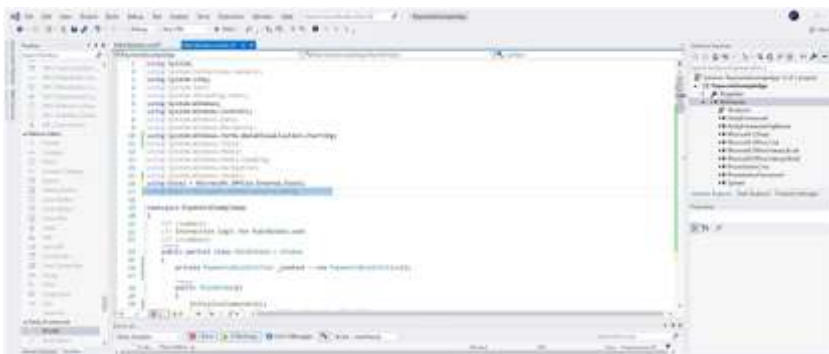
2. Добавляем кнопку экспорта



Важно

Экспорт данных в Word будет осуществляться с помощью кнопки «Экспорт в Word»

3. Подключаем пространство имен для работы с Word

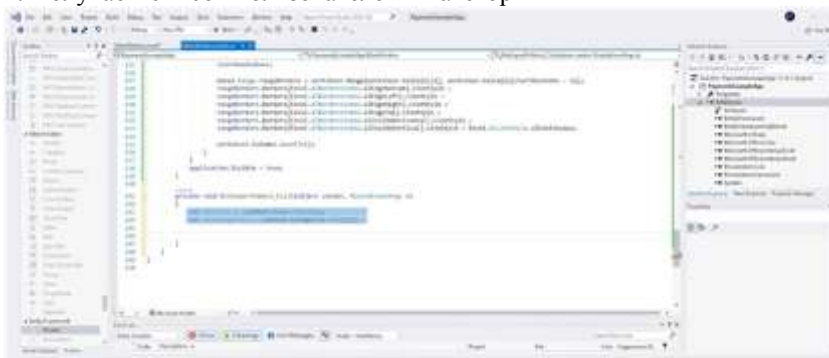


Важно

Требуемое пространство имен подключается с помощью директивы using

Реализация экспорта в документ Word

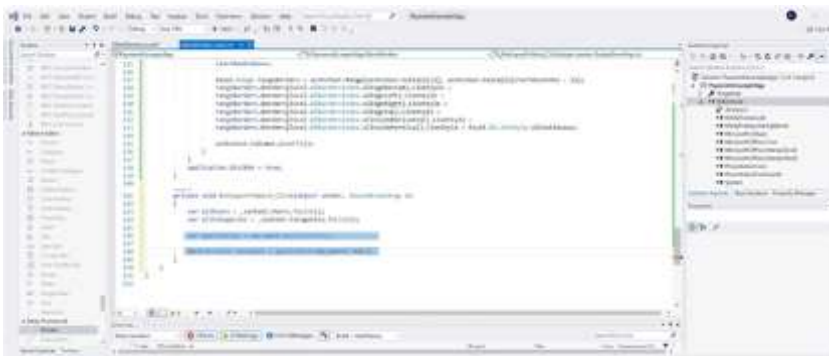
1. Получаем список пользователей и категорий



Важно

Список пользователей и категорий выгружается из базы данных

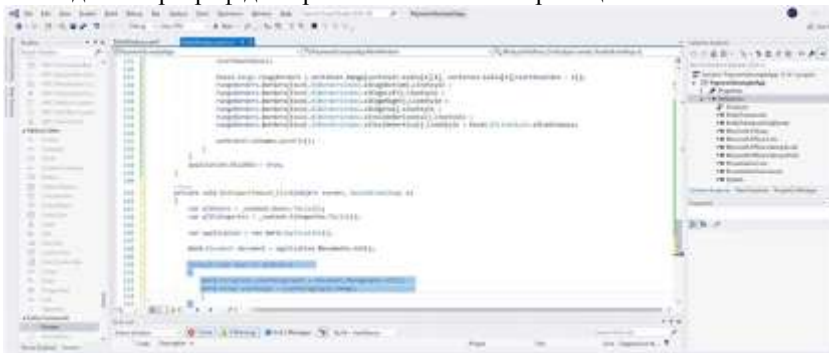
2. Создаем новый документ Word



Важно

После создания экземпляра Word в приложение добавляется новый документ, с которым далее происходит работа

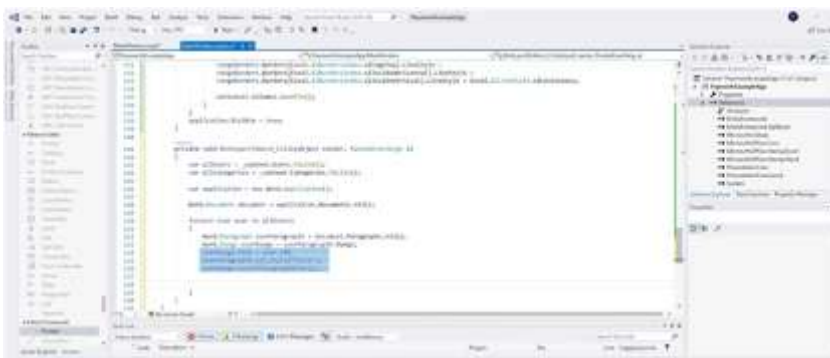
3. Создаем параграф для хранения названий страниц



Важно

Основной структурной единицей текста является параграф, представленный объектом Paragraph. Все абзацы объединяются в коллекцию Paragraphs, причем новые параграфы добавляются с помощью метода Add. Доступ к тексту предоставляет объект Range, являющийся свойством Paragraph, а текстовое содержание абзаца доступно через Range.Text. В данном случае для хранения ФИО каждого пользователя создается новый параграф

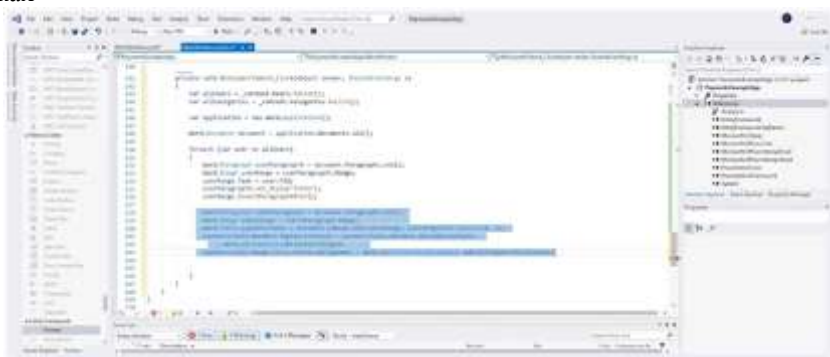
4. Добавляем названия страниц



Важно

В качестве названия выбирается имя пользователя, к которому применяется стиль «Title», после чего добавляется новый параграф для таблицы с платежами

5. Добавляем и форматируем таблицу для хранения информации о платежах



Важно

После создания параграфа для таблицы и получения его Range, добавляется таблица с указанием числа строк (по количеству категорий + 1) и столбцов. Последние две строчки касаются указания границ (внутренних и внешних) и выравнивания ячеек (по центру и по вертикали)

6. Добавляем названия колонок и их форматирование



Важно

Таблица состоит из трех колонок с названиями «Иконка», «Категория» и «Сумма расходов». Названия колонок выделяются жирным шрифтом и выравниваются по центру

7. Заполняем первую колонку таблицы



Важно

Положение ячейки заносится в переменную cellRange. Метод AddPicture() класса InlineShape позволяет добавить изображение в ячейку. Иконки категорий размещаются в новой папке Assets, основные шаги создания которой изображены на скриншоте

8. Форматируем первую колонку таблицы



Важно

Для первой колонки устанавливаются длина, ширина, а также горизонтальное выравнивание по центру

9. Заполняем вторую и третью колонки



Важно

Сумма платежей приводится к нужному формату с указанием единиц измерения (руб.) непосредственно в коде

Завершение оформления документа Word

1. Добавляем максимальную величину платежа



Важно

Для поиска максимального платежа сначала платежи сортируются по стоимости. В случае, если такой платеж найден, добавляется новый параграф. Получается диапазон и выводится текст с информацией о наименовании платежа, его стоимости и дате совершения. В заключение устанавливается стиль и цвет текста (красный)

2. Добавляем минимальную величину платежа



Важно

Аналогично среди всех платежей данного пользователя определяется наименьший платеж и отображается шрифтом зеленого цвета

3. Делаем заключительные шаги

предметной области, свойства (attributes) сущностей и связи (relationship) между ними

ER-диаграммы делятся на концептуальные и физические. В отличие от физических, в концептуальных ER-диаграммах не учитываются особенности конкретной базы данных. Впоследствии сущности концептуальных ER-диаграмм становятся таблицами, атрибуты – колонками, а связи реализуются путем миграции ключевых атрибутов родительских сущностей и создания внешних ключей

Пример построения ER-диаграммы

Предметная область – фитнес-индустрия. Цель заказчика – разработка платформы для удаленных тренировок. Основные шаги построения ER-диаграммы:

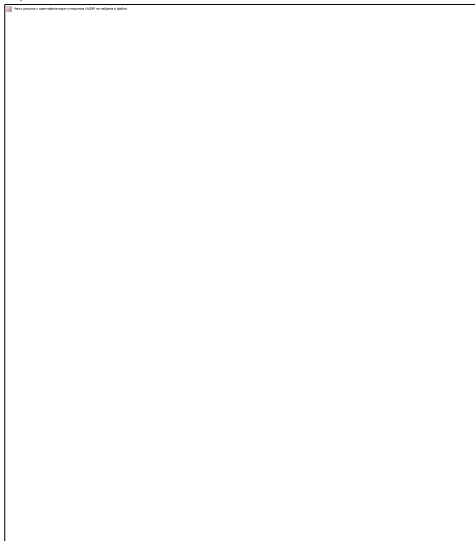
1. Добавление сущностей
2. Добавление связей и их настройка
3. Добавление атрибутов

Важно

В данном занятии ER-диаграмма составляется в Microsoft Visio на основе описания заказчика. Используется тип диаграммы Crow's Food database notation

Добавление сущностей

Выделяем сущности в описании заказчика

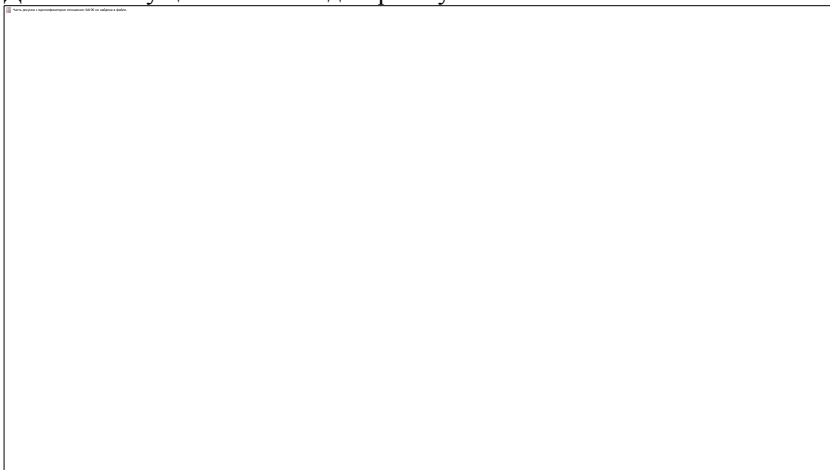


Важно

Сущность (entity) – класс реальных или виртуальных однотипных объектов, информацию о которых необходимо хранить в базе данных.

Пример сущности – «тренер»

Добавляем сущности на ER-диаграмму

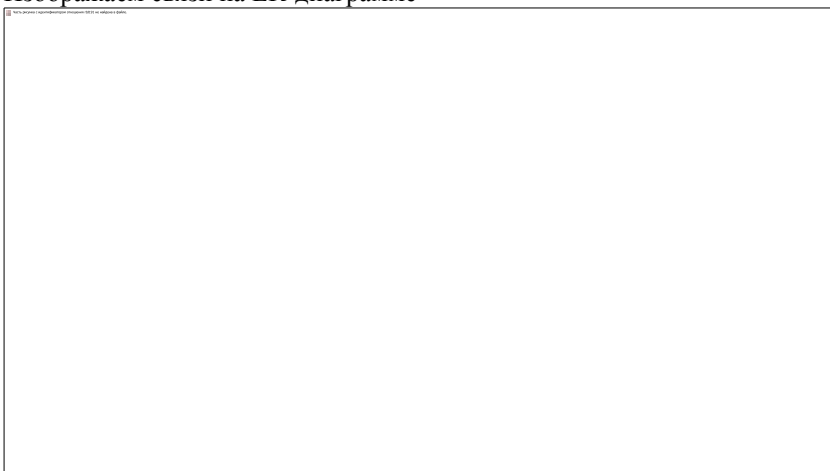


Важно

На ER-диаграмме сущность изображается в виде прямоугольника, внутри которого содержится имя сущности в форме существительного в единственном числе

Добавление связей и их настройка

Изображаем связи на ER-диаграмме



Важно

Связь (relationship) – ассоциация между сущностями. Для облегчения понимания диаграммы следует добавлять названия связей.

Пример связи – «тренер получает заявку»

Указываем тип связи между сущностями

При определении типа следует учитывать модальность связи: «может» или «должен». Модальность «может» означает, что экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может быть и не связан ни с одним экземпляром другой сущности. Модальность «должен» подразумевает связь не менее чем с одним экземпляром другой сущности. Примеры возможных типов связей представлены в таблице

Название типа	Пример	Комментарий
Один-к-одному	План тренировки должен быть составлен по одной заявке / По заявке может быть составлен один план тренировки	Данный тип следует использовать исключительно для связывания различных сущностей (разные сущности должны иметь разные атрибуты)

Название типа	Пример	Комментарий
Один-ко-многим	План тренировки может включать много индивидуальных занятий / Индивидуальное занятие должно относиться к одному плану тренировки	Наиболее часто используемый тип связи

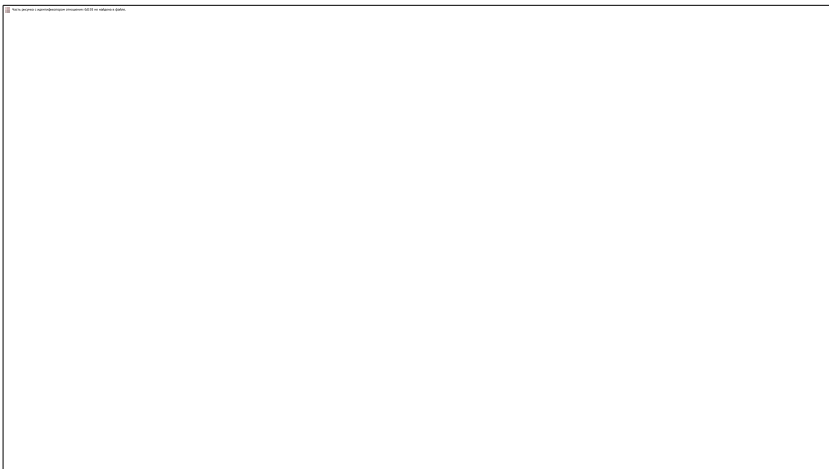
Многие-ко-многим

Тренер может пройти несколько курсов обучения / Курс обучения может быть пройден многими тренерами

Используется исключительно в качестве временного типа. При дальнейшей разработке данная связь заменяется на две связи типа «один-ко-многим» путем добавления промежуточной сущности

Добавление атрибутов

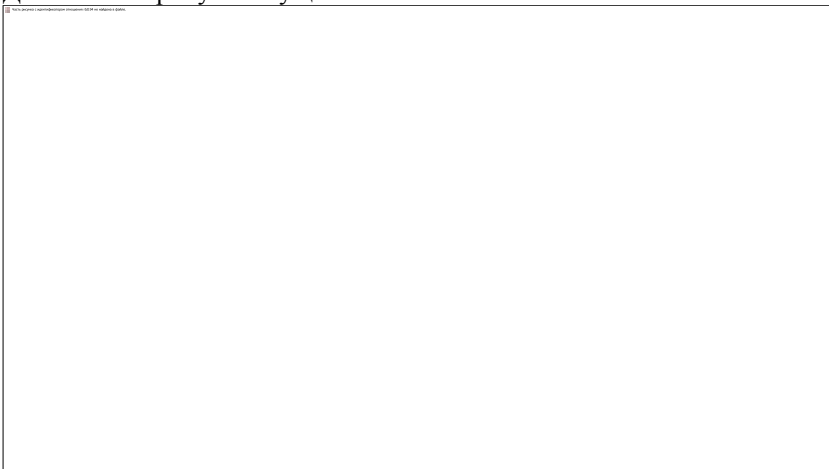
Выделяем атрибуты в описании заказчика



Важно

Атрибуты предназначены для описания сущности. В приведенном примере они выделены красным цветом шрифта

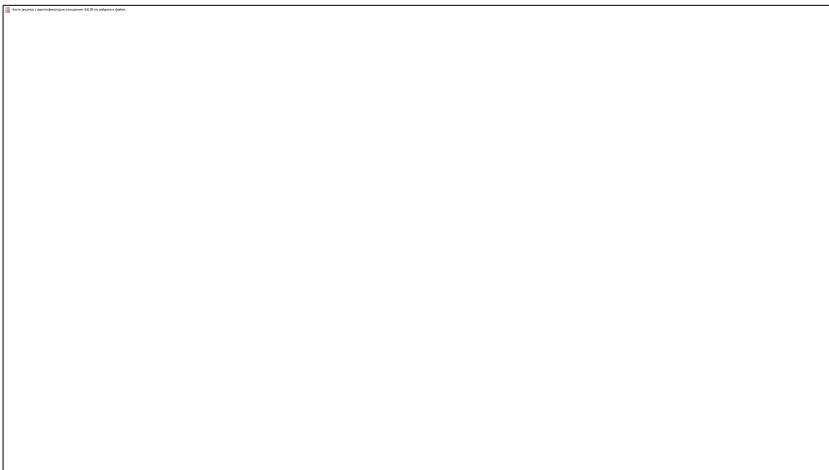
Добавляем атрибуты к сущностям



Важно

Следует учитывать, что не все атрибуты могут быть указаны явно в техническом задании. Например, для определения стажа тренера удобно хранить в базе данных дату его трудоустройства

Добавляем ключ к сущностям



Важно

Ключ – это один или несколько атрибутов, уникально определяющих сущность. В данном примере в качестве ключа используется атрибут «код».

СОЗДАНИЕ БАЗЫ ДАННЫХ

Базы данных и правила их создания

Разрабатываемые нами программные решения предполагают работу с большим объемом информации, которую очень важно хранить в едином по структуре и стилистике виде. Эта информация хранится в базе данных и может постоянно пополняться. От того, как часто это делается, зависит ее актуальность.

Базы данных, как способ хранения больших объемов информации и эффективного манипулирования ею, используются практически во всех областях человеческой деятельности. В них хранят документы, изображения, сведения об объектах недвижимости, физических и юридических лицах и прочие данные, с которыми необходимо работать в рамках предметной области. При этом, вся информация не хранится в каком-то обобщенном виде, а разбивается на таблицы, каждая из которых отвечает за определенный объект предметной области. Чем больше данные обособляются в таблицы, тем выше вероятность избежать дублирования информации и захламления базы данных, а также сокращает время и ресурсы на поиск необходимых данных

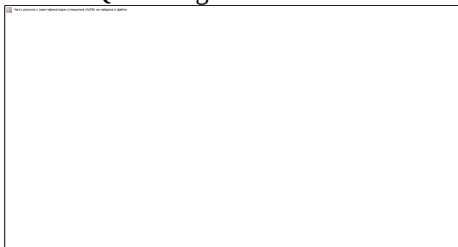
MS SQL Management Studio

ПО для управления базами данных. Основные элементы

интерфейса.

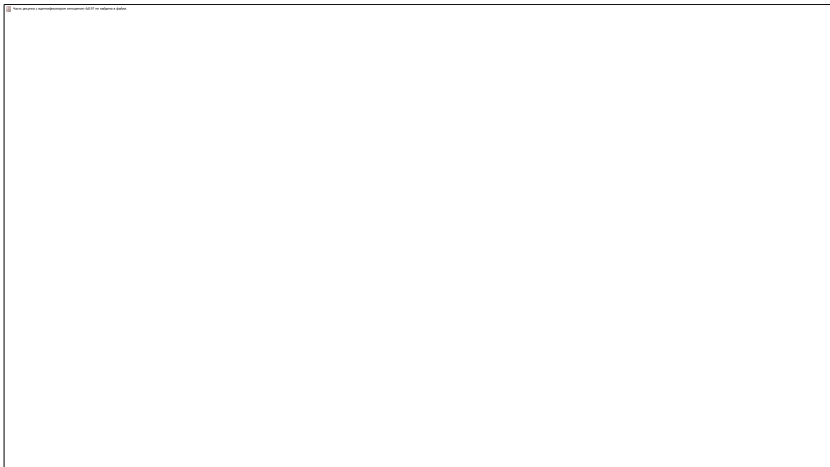
Создание новой базы данных

1. Запускаем MS SQL Management Studio.

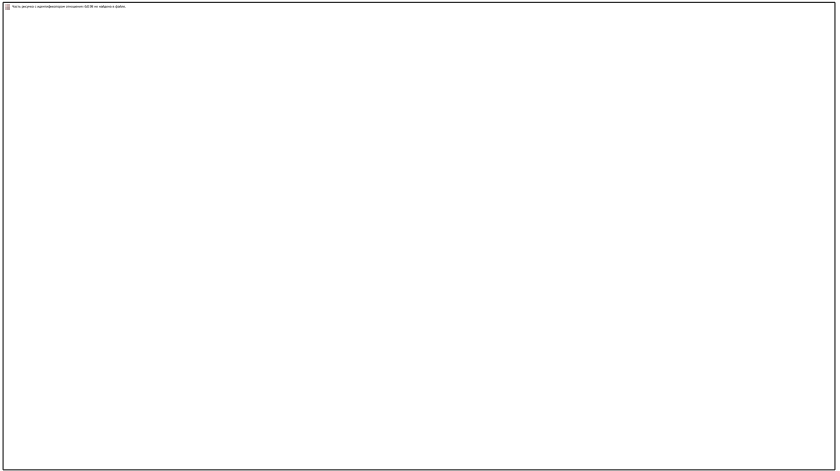


А) Подключаемся к серверу:

Server Name: localhost\SQLEXPRESS Authentication: Windows Authentication.

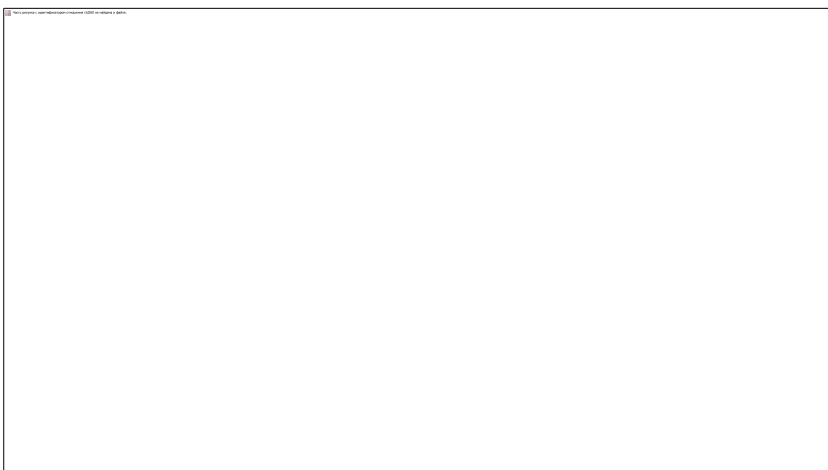


Б) Жмем Connect. Открывается список баз данных сервера



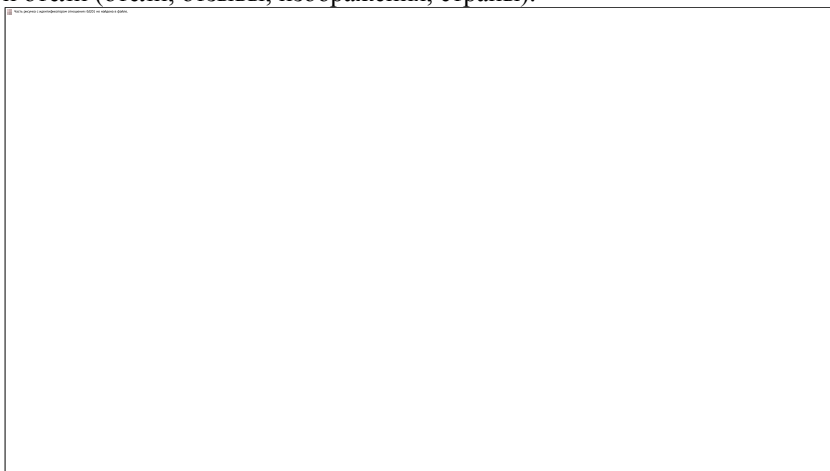
2. Создаем новую базу данных





3. Определяем основные сущности и создаем таблицы.

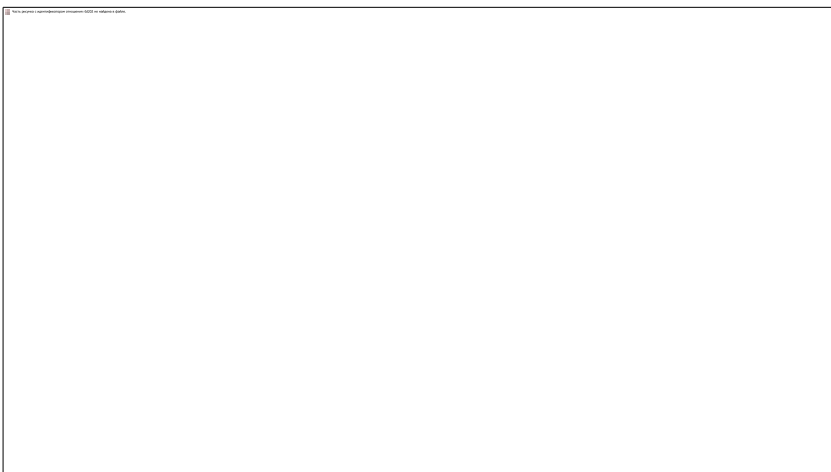
На основании предоставленной диаграммы ресурсов добавим таблицы в базу данных, разделив информацию на 2 блока: туры (туры, типы) и отели (отели, отзывы, изображения, страны).



Создаем таблицы.

Существует несколько способов:

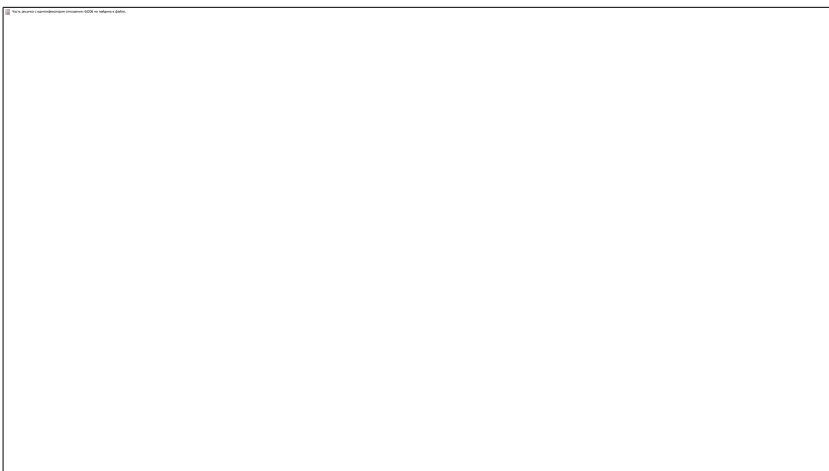
А) дизайнер таблиц



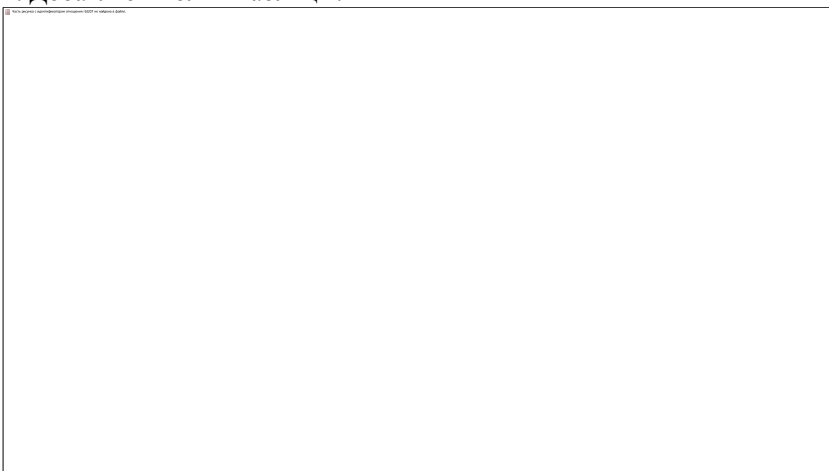
Б) диаграмма БД



Создаем таблицы туров (Tour), отелей (Hotel), изображений (Hotelimage), отзывов (HotelComment), стран (Country), типов тура (Type).



4. Добавляем поля в таблицы:



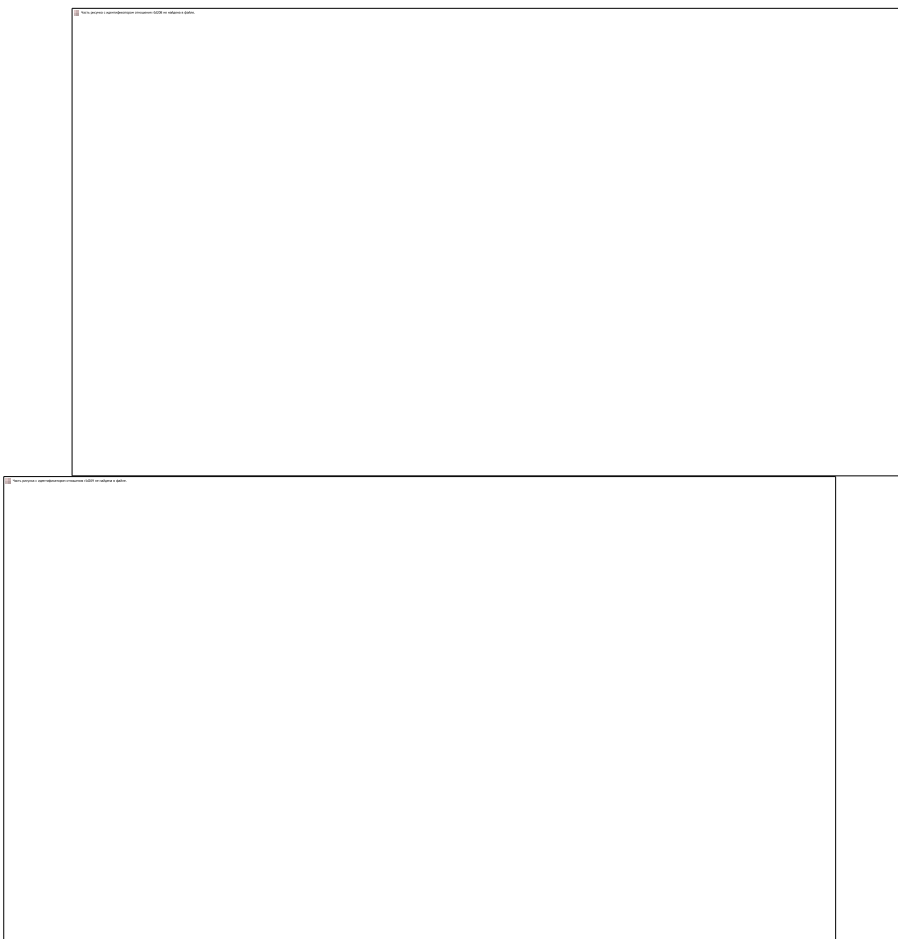
А) Таблица Tour

- Код тура (id)
- Количество билетов (TicketCount)
- Название (Name)
- Описание (Description)
- Изображение (ImagePreview)
- Стоимость (Price)
- Актуальность (isActual)

Б) Таблица Hotel

- id

- Name
 - CountOfStars
 - CountryCode
 - В) Таблица HotelImage
 - id
 - Hotelid
 - ImageSource
 - Г) Таблица HotelComment
 - id
 - Hotelid
 - Text
 - Author
 - CreationDate
 - Д) Таблица Country
 - Code
 - Name
 - Е) Таблица Type
 - Name
 - Description
5. Расставляем первичные ключи

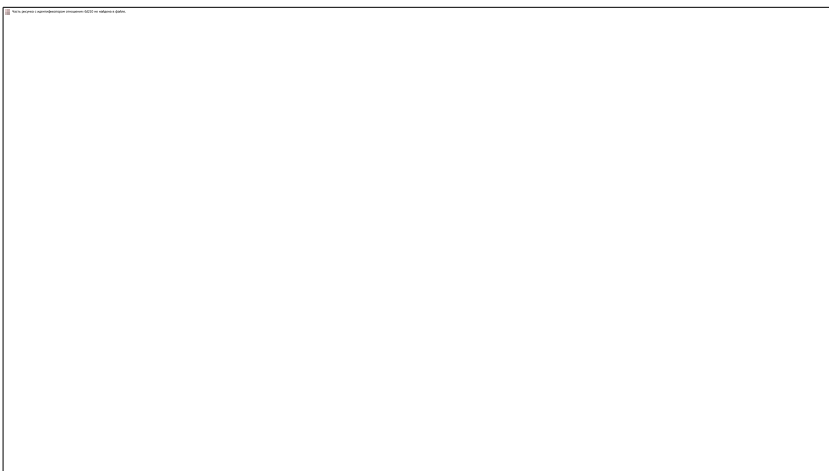


Важно

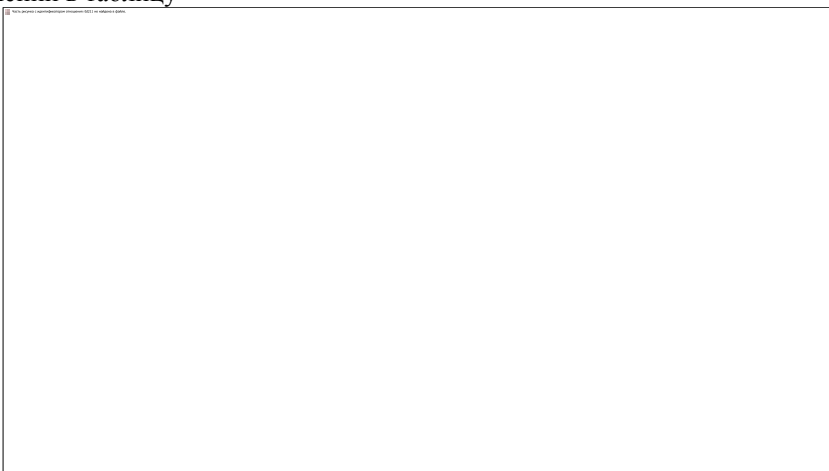
Первичный ключ – поле, которое уникально характеризует запись (строку) в таблице

6. Устанавливаем типы данных

- название отеля – текстовый тип данных
- количество звезд – числовой
- дата создания отзыва – тип date
- и т. д. (на скриншоте)



7. Устанавливаем обязательность или необязательность поля. В третьем столбце есть маркер, отвечающий за обязательность поля. В случае, если мы отметим его галочкой, поле будет обязательным при заполнении в таблицу



8. Устанавливаем связи между таблицами

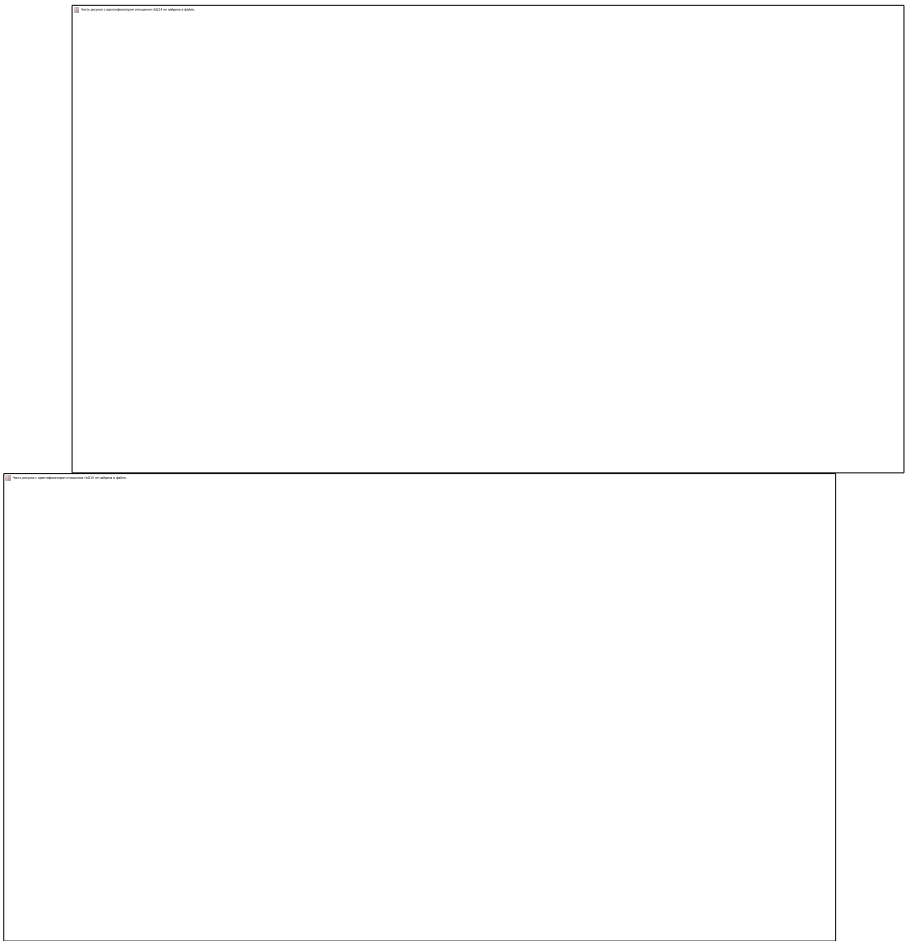
А) один-ко-многим

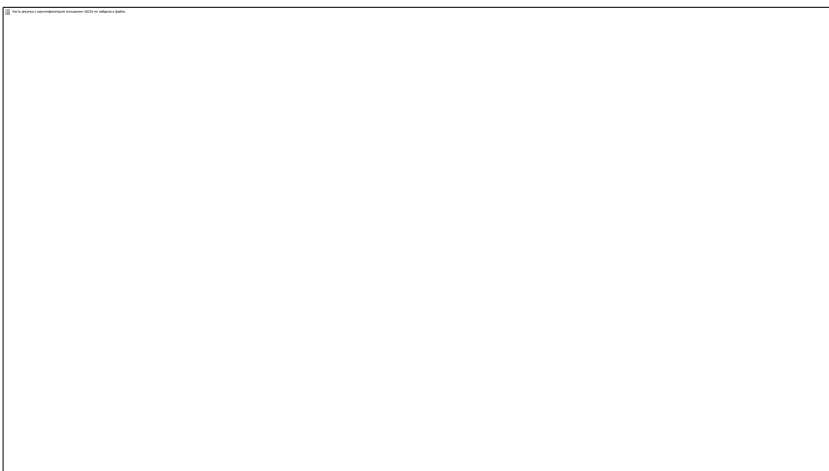
Чтобы связать таблицу стран и отелей, в таблицу Hotel необходимо добавить специальное поле – внешний ключ (в нашем случае это CountryCode), который по типу совпадает с тем, что является первичным ключом в таблице Country. Далее от первичного ключа таблицы Country ведется связь к внешнему ключу таблицы.



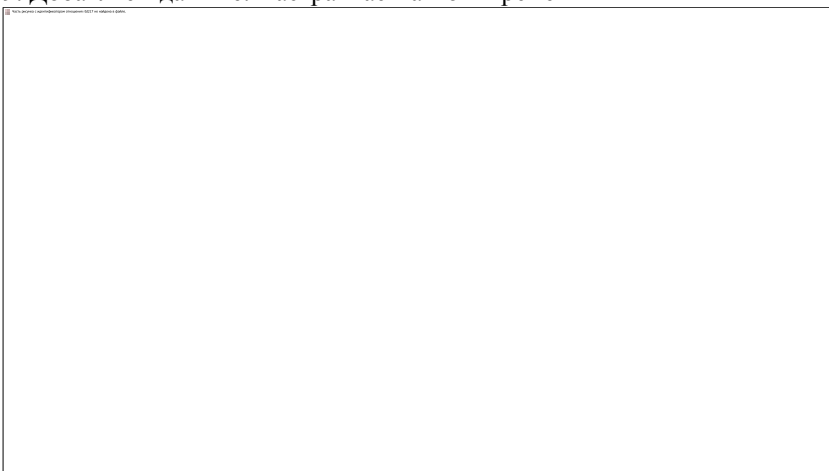
Б) многие-ко-многим

На диаграмме ресурсов между таблицами туров и типов была связь «многие-ко-многим», которую нам необходимо реализовать в базе данных. Для этого нужно создать еще одну таблицу (назовем ее `TourOfTour`) и создадим поля – первичные ключи из других таблиц. В данной таблице оба поля будут являться ключевыми





9. Добавляем данные. Настраиваем автоинкременты



Для того, чтобы не вести учет количества порядковых номеров, в настройках поля можно установить автоматическое определение значения поля при добавлении записи. Например, если в таблице есть 3 записи с номерами от 1 до 3, то следующая запись автоматически будет иметь номер 4.

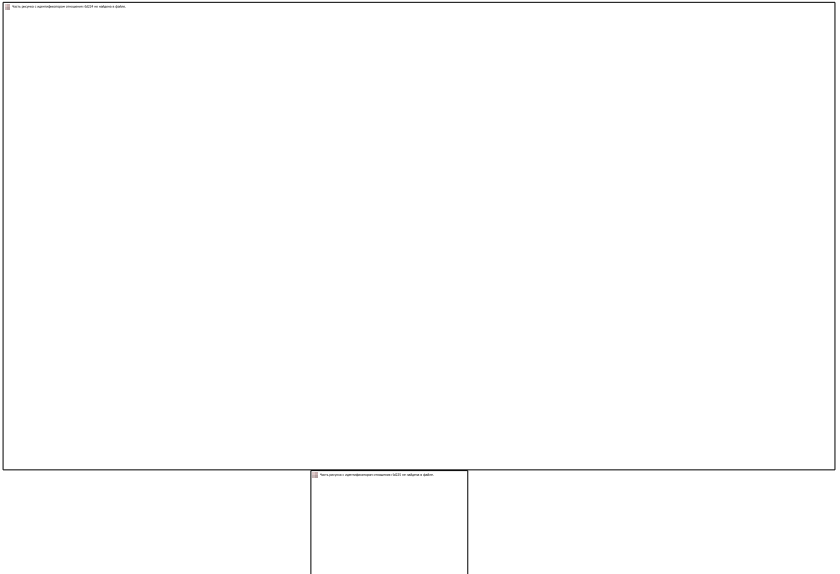


10. Сохранение БД и создание скрипта.

Все созданные базы данных хранятся на сервере. Чтобы перенести базу данных на другой сервер, необходимо правильно ее сохранить. Один из методов переноса - создание скрипта базы данных.

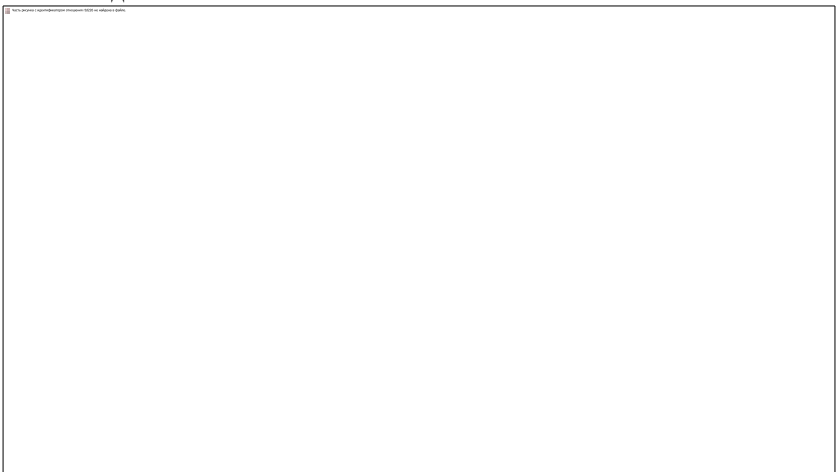
1. Selecione a opção correta para cada item.
2. Selecione a opção correta para cada item.

1. Selecione a opção correta para cada item.	
2. Selecione a opção correta para cada item.	



В данном случае выполнение скрипта приведет к восстановлению структуры таблиц и переносу записей из вашей базы данных.

Так что его можно использовать не только для переноса базы на другой сервер, но и для хранения резервных копий предыдущих состояний базы данных.



РАБОТА С НЕСТРУКТУРИРОВАННЫМИ ДАННЫМИ: ОБРАБОТКА И ИМПОРТ В БАЗУ ДАННЫХ

Часто информация хранится в неструктурированном виде, т. е. не имеет заранее определенной структуры данных, либо не организована в установленном порядке. Такие данные, как правило, представлены в форме текста или больших неупорядоченных таблиц. Это приводит к трудностям анализа, особенно в случае использования традиционных программ, предназначенных для работы со структурированными данными.

Помимо ручного добавления данных в базу может возникнуть необходимость обработки данных и импорта. Для импорта могут быть представлены файлы и данные в разном формате. Например, сейчас у нас есть:

– список типов туров в описании предметной области

В рамках данного задания необходимо разработать систему для туристического агентства, которое предоставляет услуги по организации туров.

За актуальную информацию по имеющимся турам отвечает администратор туристического агентства, который может создавать новые туры и редактировать существующие (в том числе даты действия тура).

Туры распределены по типам (международный туризм, внутренний туризм, специализированные детские туры, лечебно-оздоровительные туры, экскурсионные туры, обслуживание корпоративных клиентов по заказу, горнолыжные курорты, культурно-исторические туры, пляжные туры). Один тур может относиться к нескольким типам туров.

В рамках тура предлагается определенный список отелей и перечень услуг (перевозка, экскурсионные услуги, услуги гида-переводчика, услуги по оформлению заграничного паспорта, визы и т.д.).

Отели бывают разного уровня комфортности, уровень комфортности определяется звездами от 1 до 5. При бронировании отеля можно указать тип питания (RO (RoomOnly), RR (RoomRate), OB (OnlyBed), AO (AccommodationOnly) – проживание в номере без питания; BB (BedBreakfast) – только завтрак; HB (HalfBoard) – завтрак, ужин; HB+ (HalfBoardPlus) – завтрак, ужин, бесплатные напитки в течение дня; FB (FullBoard) – завтрак, обед, ужин; FB+ (FullBoardPlus) – завтрак, обед, ужин, бесплатные напитки во время приема пищи; AI (AllInclusive) – завтрак, обед, ужин, перекусы в течение дня, бесплатные напитки в течение дня; UAI (UltraAllInclusive, UALL) – завтрак, обед, ужин, перекусы в течение дня, бесплатно любые напитки в течение дня).

При первом обращении клиента менеджер регистрирует его в системе.

Подбор тура выполняет менеджер в системе в соответствии с полученной информацией от клиента и должен включать следующие пункты:

- выбор дат тура,
- указание предпочтений клиента,
- указание верхней и нижней границ стоимости,
- выбор отеля.

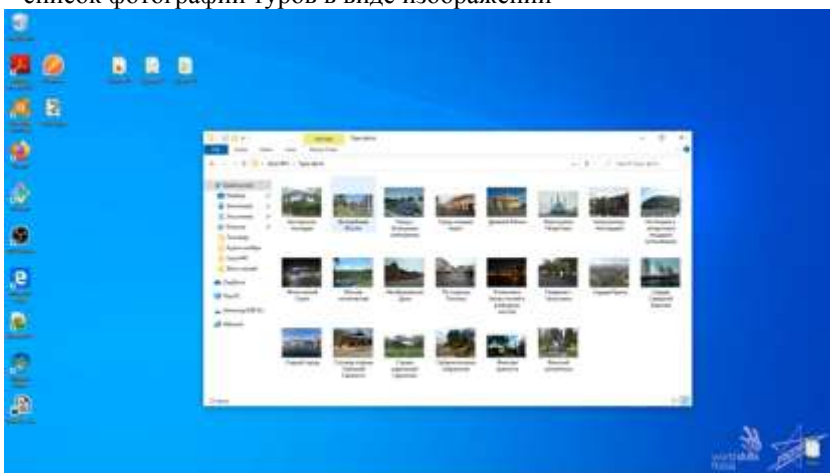
После выбора подходящего тура менеджер может зарегистрировать заявку на клиента. При желании клиента менеджер может включить в заявку дополнительные услуги, предлагаемые турагентством и доступные в рамках выбранного тура. В дальнейшем и клиент, и менеджер смогут отслеживать актуальную информацию по конкретной заявке на тур.

Каждому клиенту необходим ваучер на трансфер, ваучер на заселение в отель. Билет на самолет, страховка, виза – все это входит в стоимость тура.

– страны в формате csv

Наименование тура	Стоимость	Описание тура
1. Экскурсия в Петербург	100	1. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
2. Экскурсия в Петербург	100	2. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
3. Экскурсия в Петербург	100	3. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
4. Экскурсия в Петербург	100	4. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
5. Экскурсия в Петербург	100	5. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
6. Экскурсия в Петербург	100	6. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
7. Экскурсия в Петербург	100	7. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
8. Экскурсия в Петербург	100	8. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
9. Экскурсия в Петербург	100	9. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
10. Экскурсия в Петербург	100	10. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
11. Экскурсия в Петербург	100	11. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
12. Экскурсия в Петербург	100	12. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
13. Экскурсия в Петербург	100	13. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
14. Экскурсия в Петербург	100	14. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
15. Экскурсия в Петербург	100	15. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
16. Экскурсия в Петербург	100	16. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
17. Экскурсия в Петербург	100	17. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
18. Экскурсия в Петербург	100	18. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
19. Экскурсия в Петербург	100	19. Экскурсия в Петербург. Петербургский тур. Петербургский тур.
20. Экскурсия в Петербург	100	20. Экскурсия в Петербург. Петербургский тур. Петербургский тур.

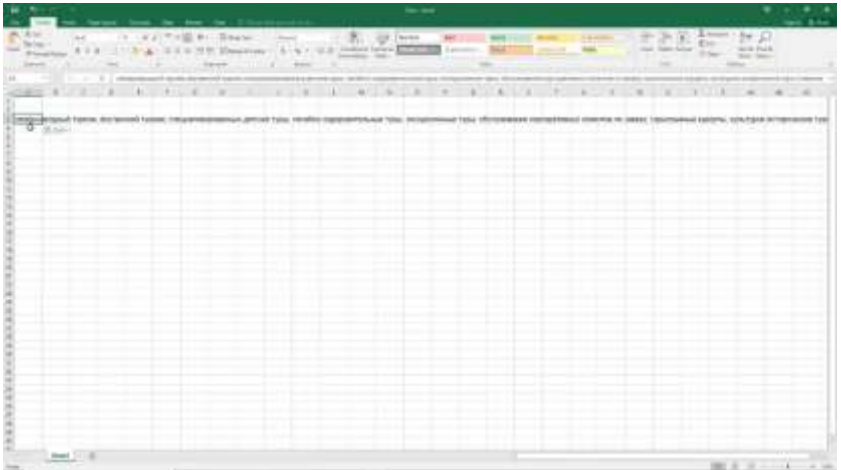
– список фотографий туров в виде изображений



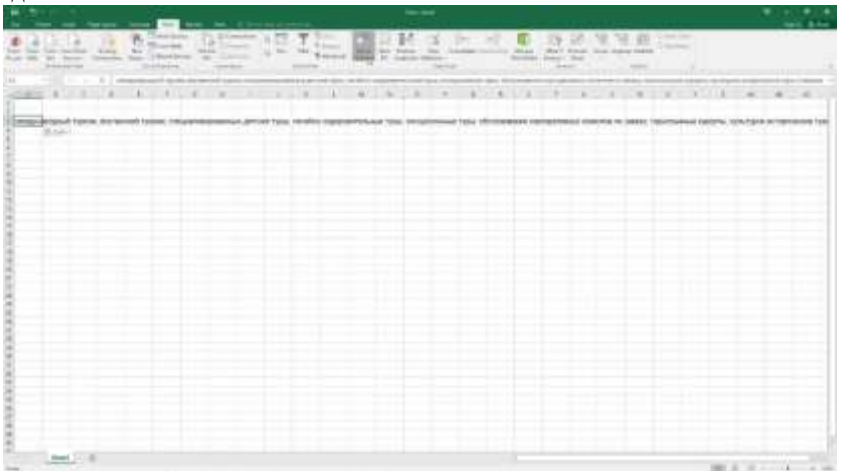
В этом уроке мы протестируем несколько разных способов импорта данных, предварительно почистив базу данных

Работа с текстовыми данными

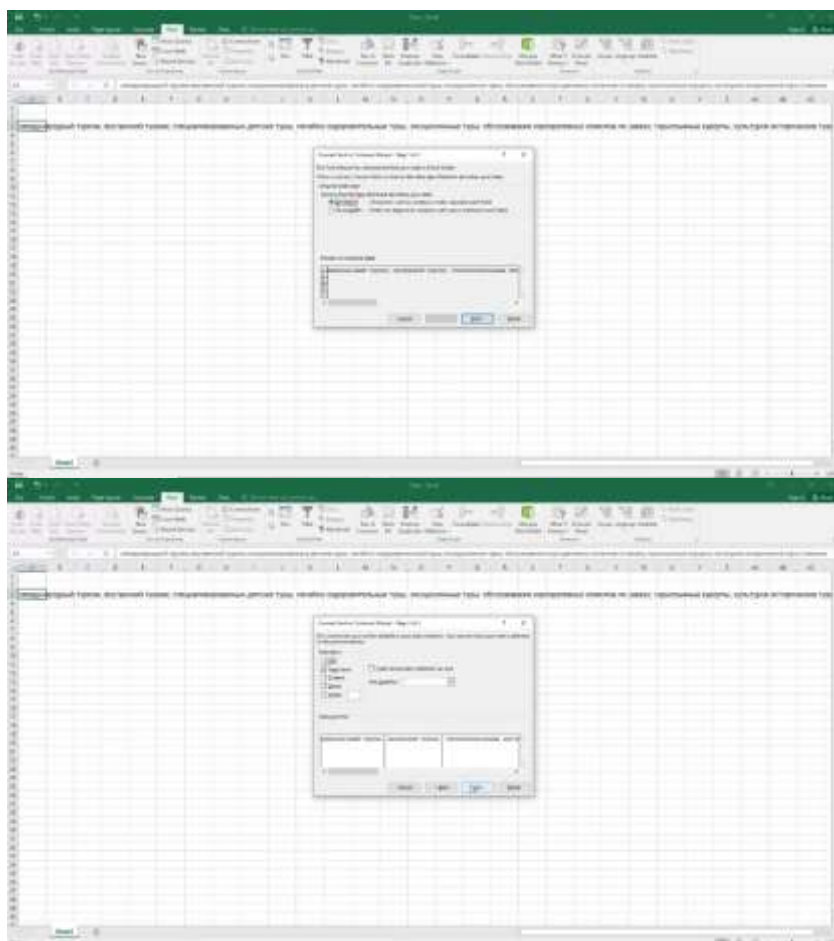
Копируем перечисление в Excel таблицу

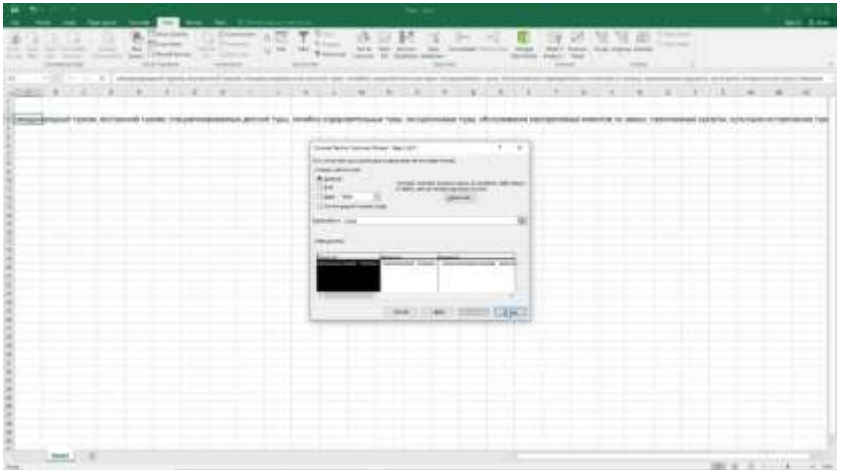


Разбиваем текст по столбцам с помощью функции Text to Columns на вкладке Data

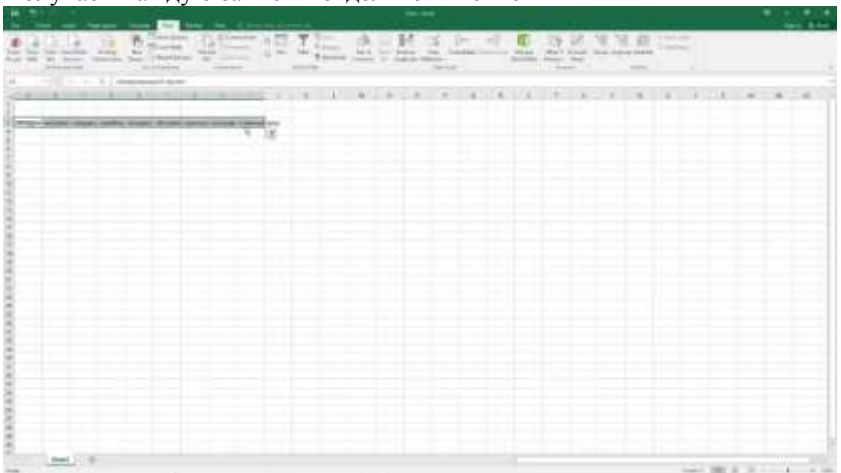


3. Указываем разделитель

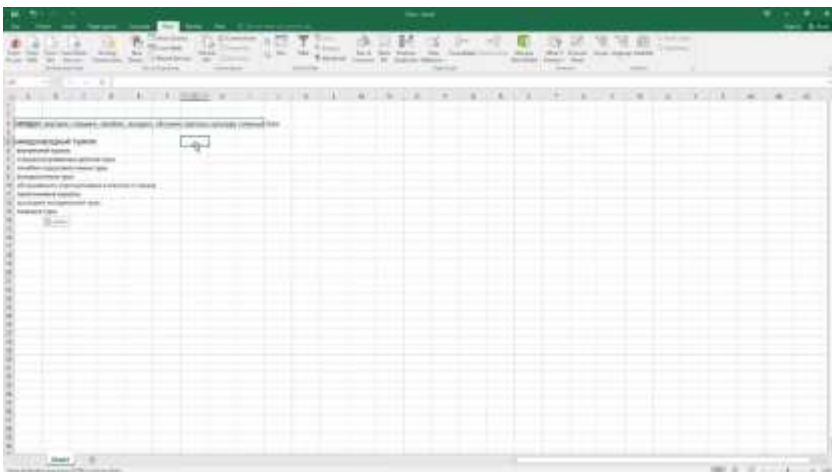




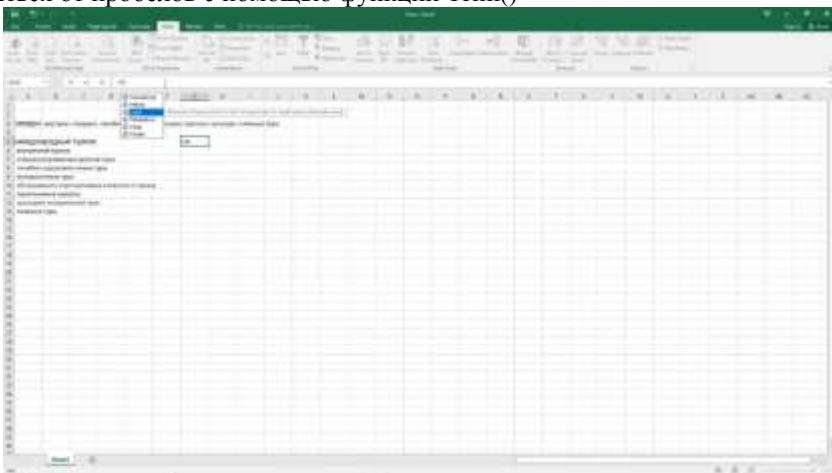
Получаем каждую запись в отдельной ячейке



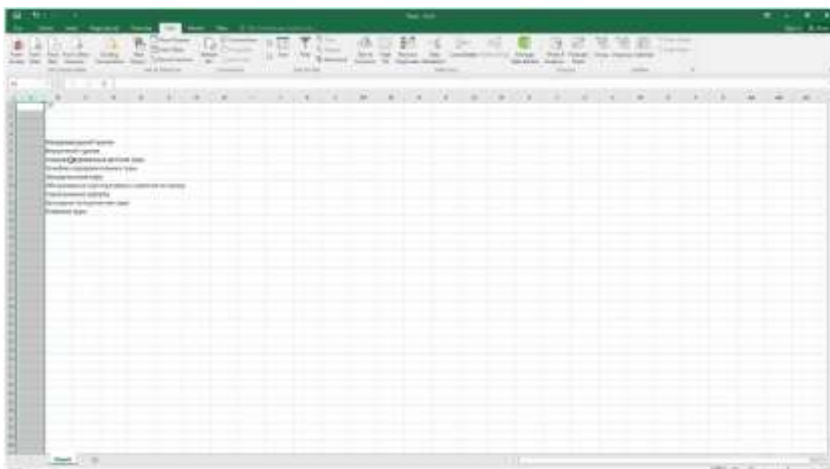
Теперь каждый тип списка изменяем на вертикальный. Для этого копируем значения и с помощью функции Past special транспонируем данные



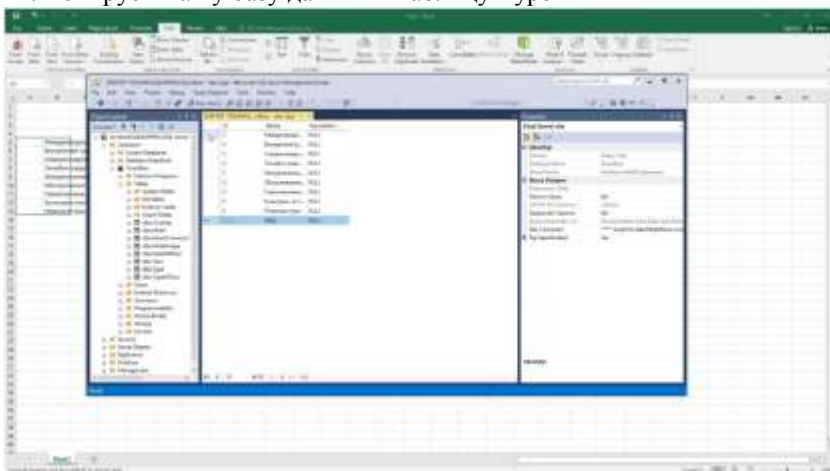
Далее было бы неплохо эти данные отформатировать. Например, избавиться от пробелов с помощью функции Trim()



Отформатируем текст, чтобы название каждого типа начиналось с заглавной буквы. Для этого воспользуемся встроенными функциями Word. На вкладке Home Change keys Sentence keys



11. Копируем нашу базу данных в таблицу Туре

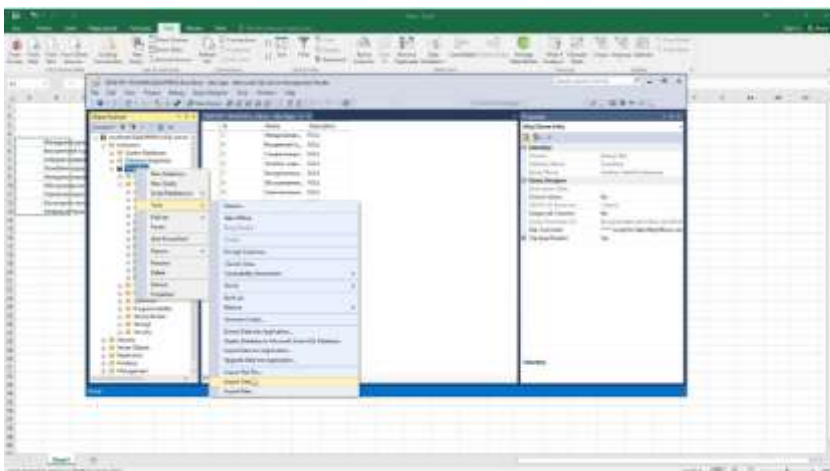


Работа с текстовыми данными

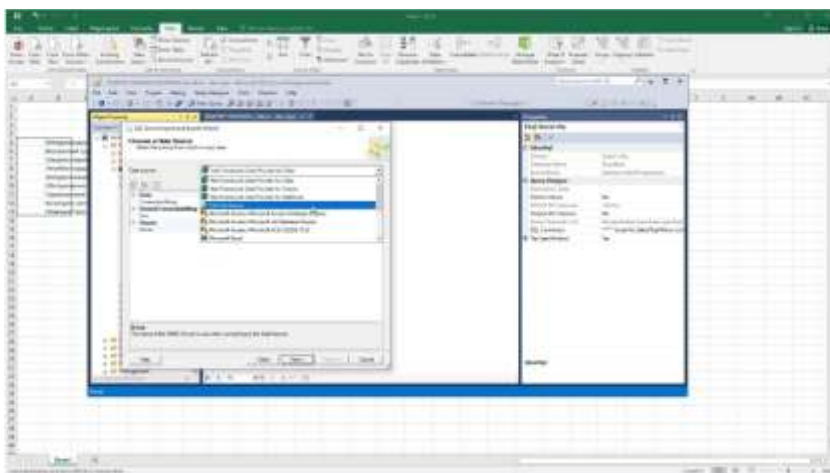
Импорт списка стран

Переходим к списку стран и воспользуемся мастером импорта и экспорта.

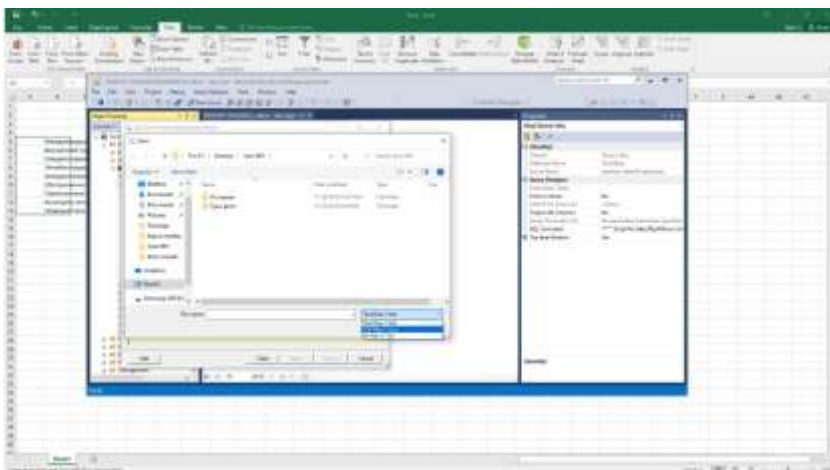
1. Кликаем по названию базы данных Tasks Import data



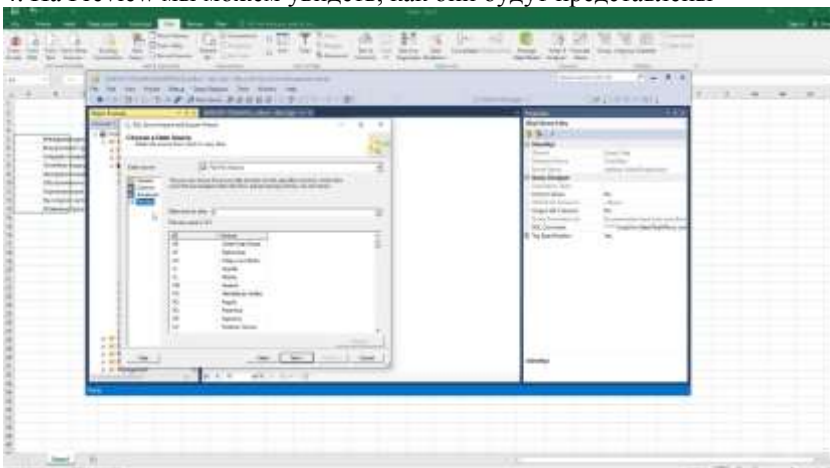
2. В качестве источника данных выбираем неструктурированный файл



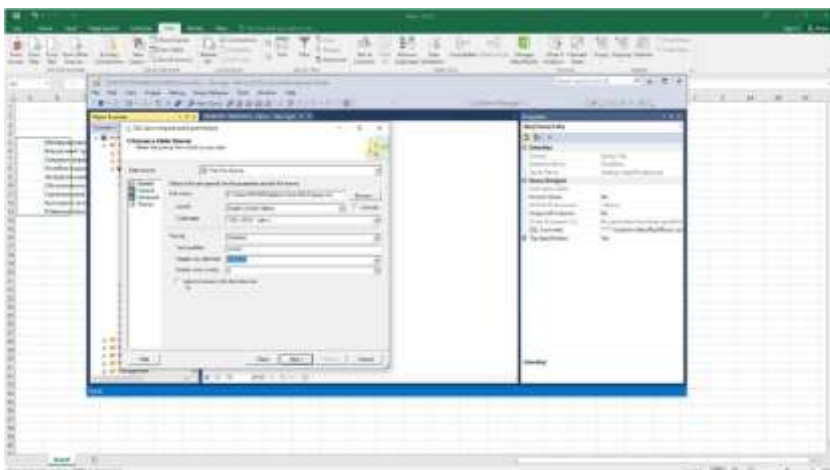
3. Тип csv files



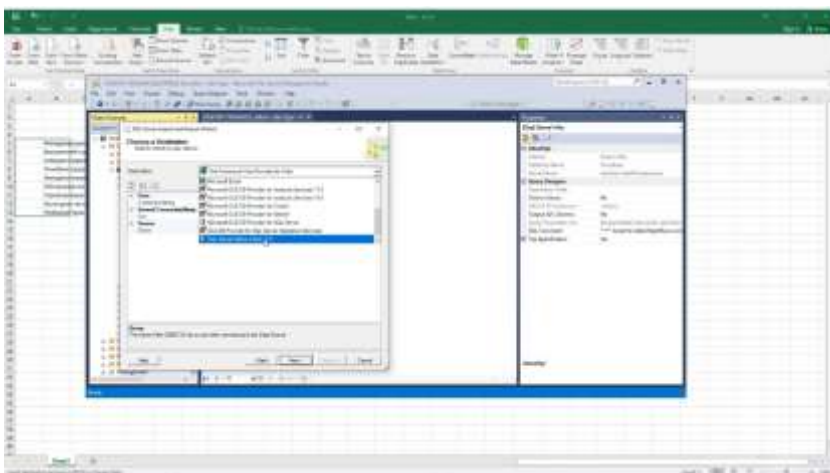
4. На Preview мы можем увидеть, как они будут представлены



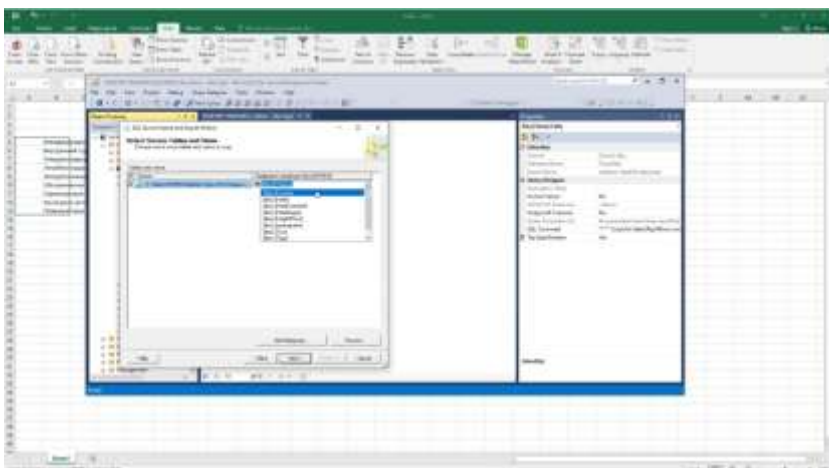
5. Убираем пункт о том, что первая строка - это заголовок



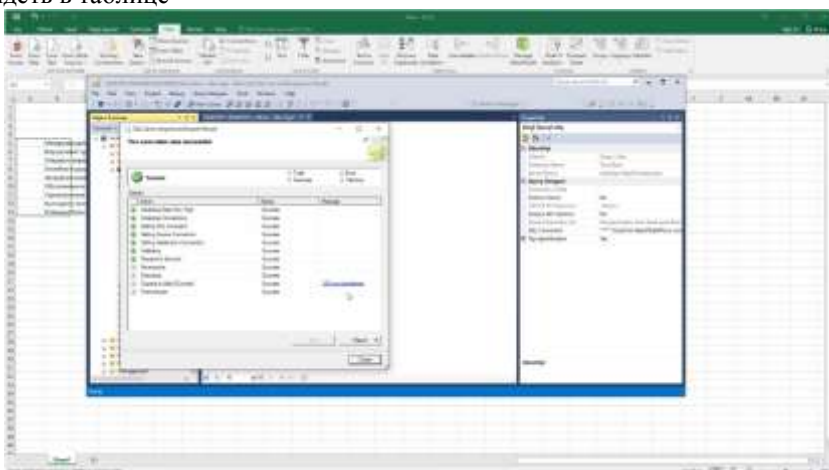
6. Далее выбираем, куда мы хотим импортировать данные нашу базу



7. После чего выбираем таблицу, где данные будут размещены

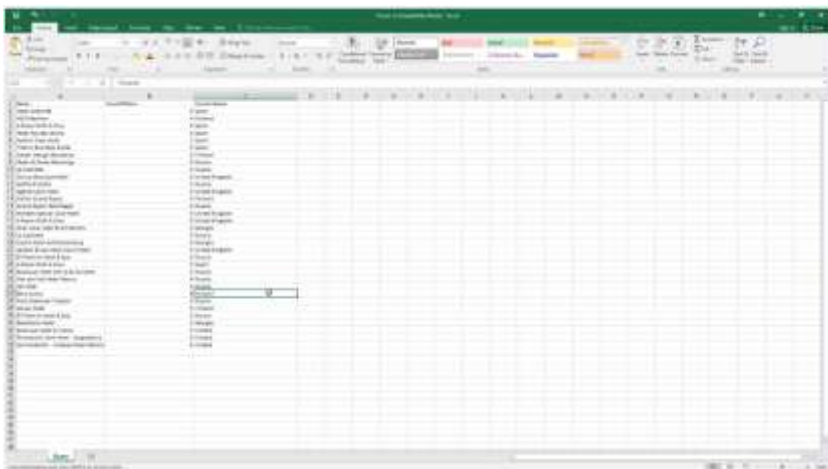


8. Далее Далее Финиш. 252 строки было импортировано. Можем их увидеть в таблице

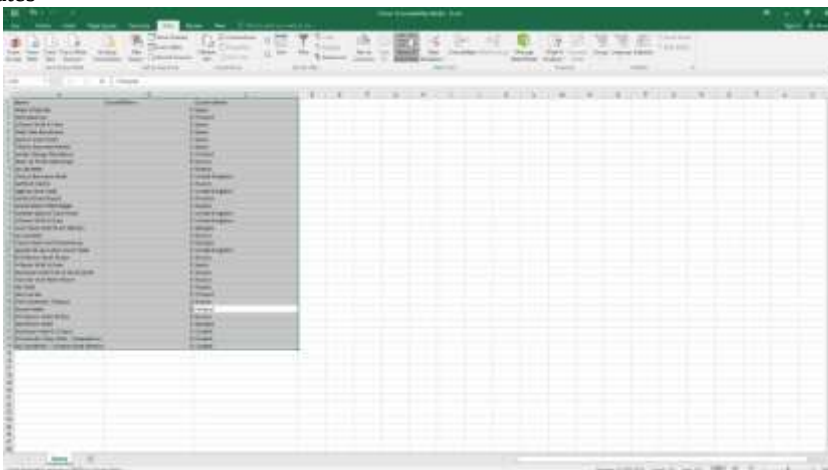


Импорт списка отелей

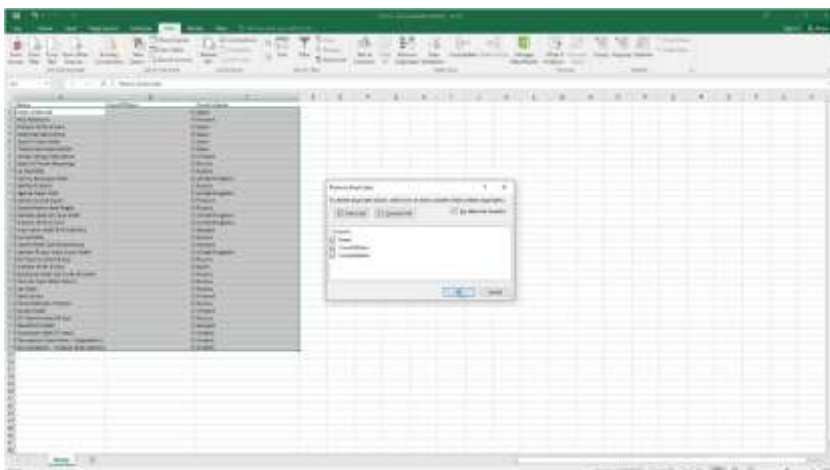
1. Приступаем к импорту отелей



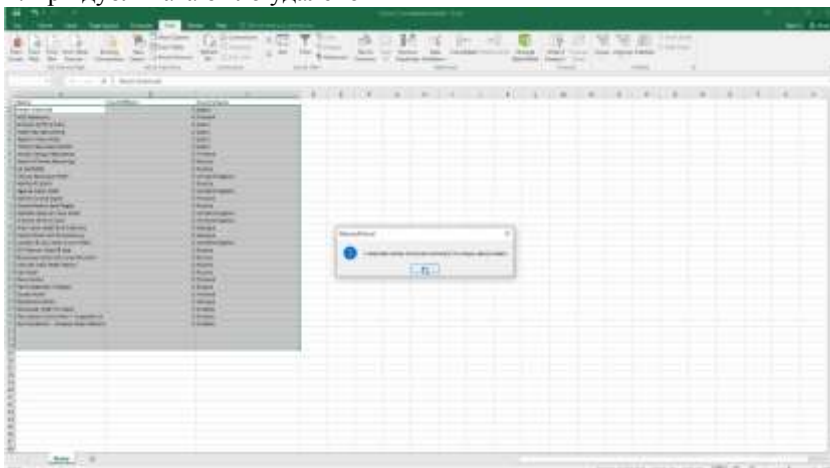
2. В таблицах могут встречаться дубликаты, поэтому желательно проверять этот момент и удалять их. Выделяем все данные Data Remove duplicates



3. Выбираем столбцы, по которым мы ищем уникальные значения.
ОК

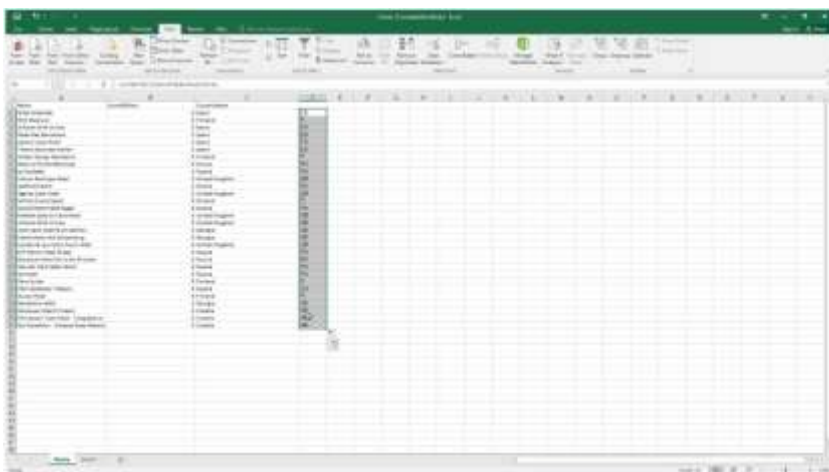


4. Три дубликата было удалено

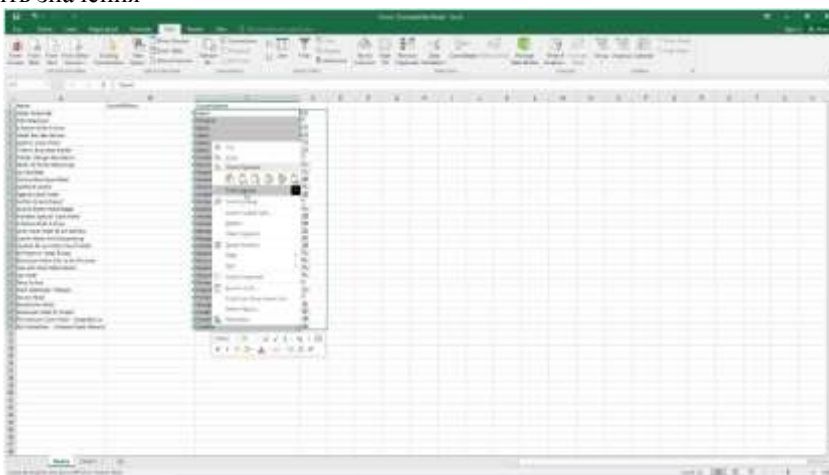


Для импорта в базу данных нам нужен столбец с кодами стран вместо названий. Для замены мы будем использовать функцию LOOKUP

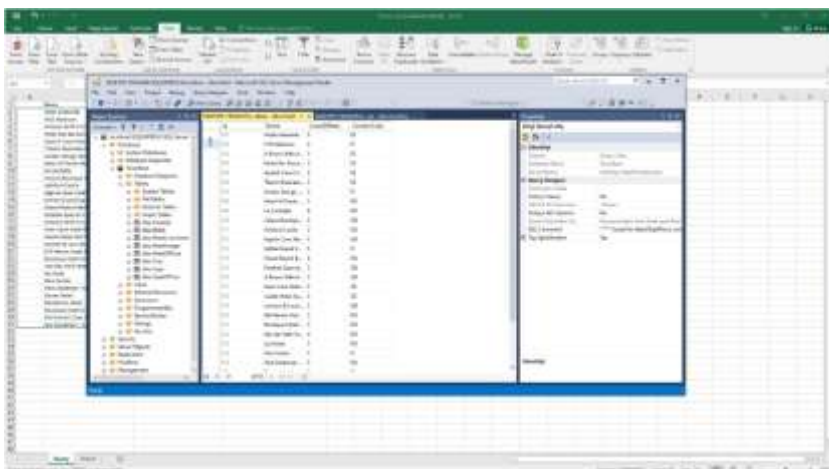
5. Сначала добавляем список стран на отдельный лист



8. Можем скопировать и с помощью специальной функции вставки добавить значения



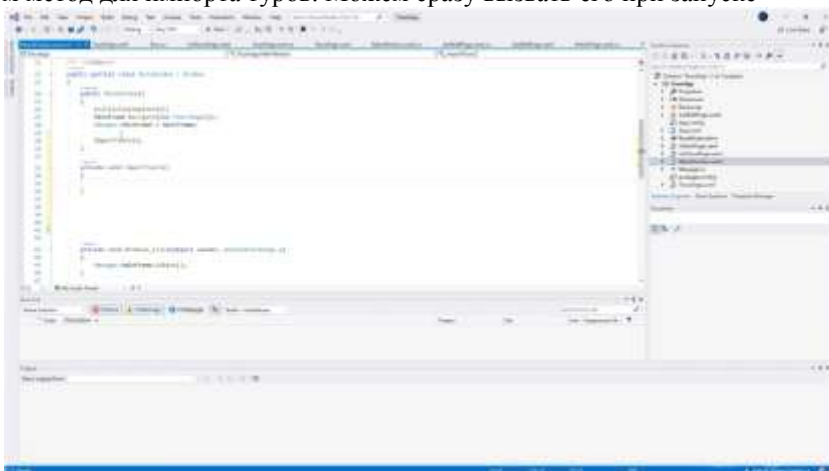
9. Добавляем столбец для кодов и вставляем в таблицу Hotel



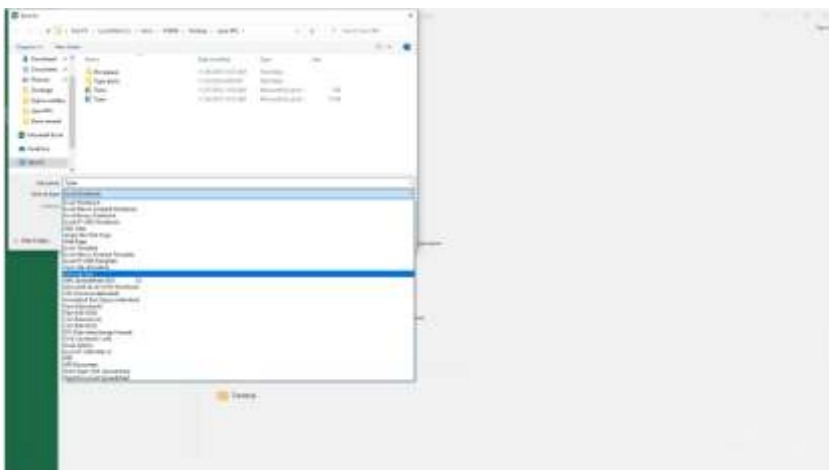
Импорт таблицы туров

Последняя таблица для импорта туры. Так как структура данных сложная: список типов через запятую, связи «многие-ко-многим», папки с картинками, которые также должны храниться в базе данных, предлагаем выполнить импорт с помощью кода C# в Visual Studio

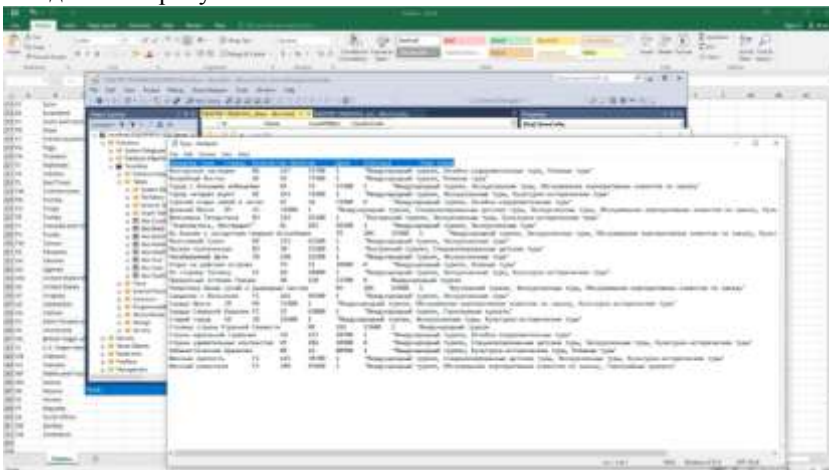
1. Воспользуемся существующим проектом, и в Mail windows создаем метод для импорта туров. Можем сразу вызвать его при запуске



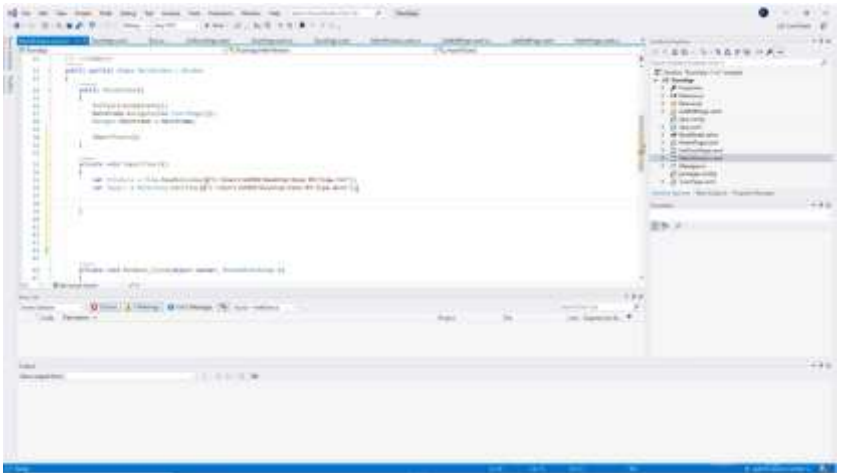
2. Прежде чем работать в коде с файлом туров, сохраняем его как unicode text и удаляем в итоговом файле первую строку с заголовком. File Save as Unicode text



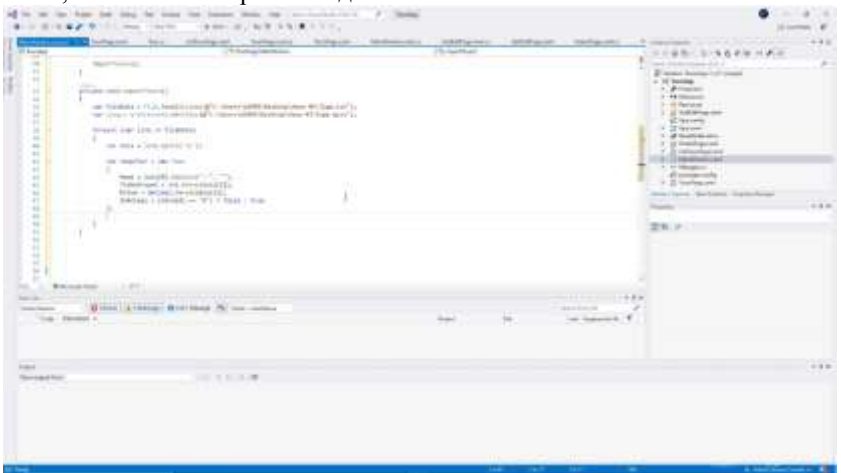
3. Удаляем строку с заголовками



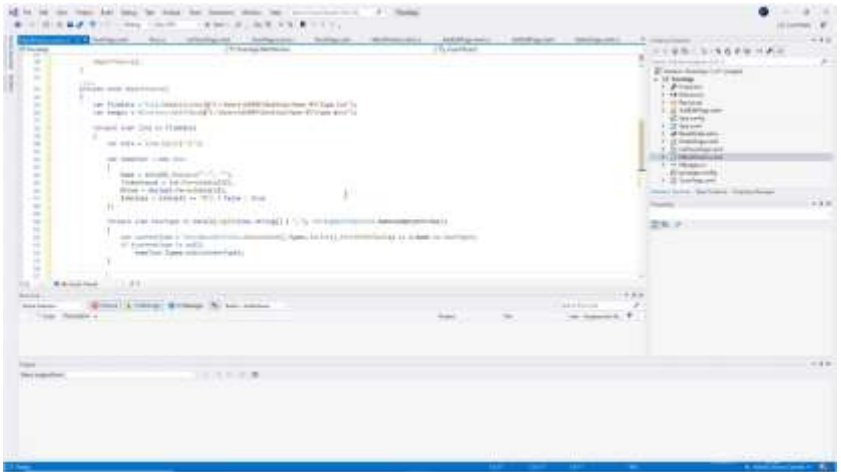
4. Создаем две переменные



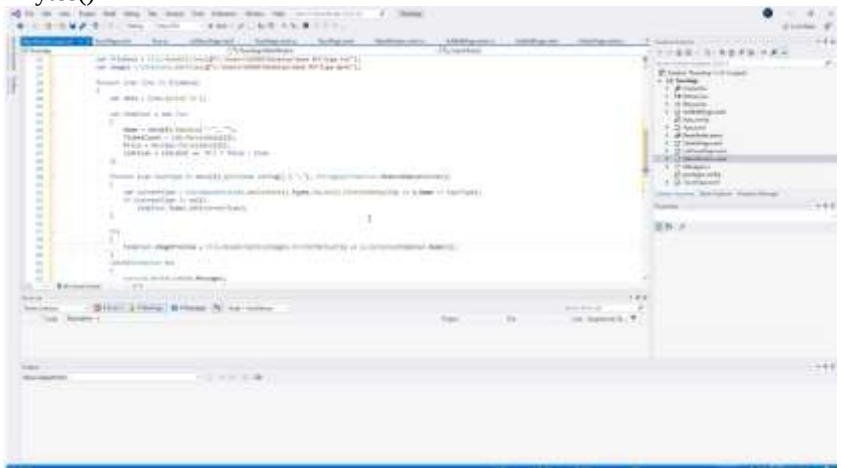
5. Пробежимся по строкам в файле, разделив данные с помощью табов, и создаем экземпляр класса, заполнив свойства соответствующими значениями, и не забывая про типы данных



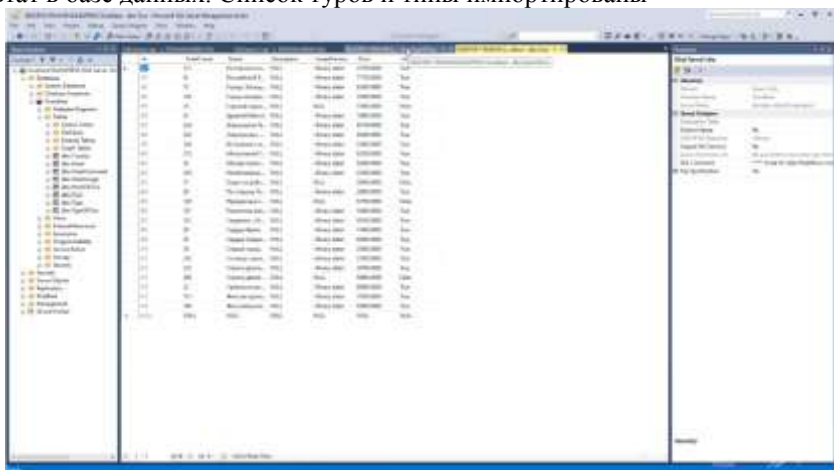
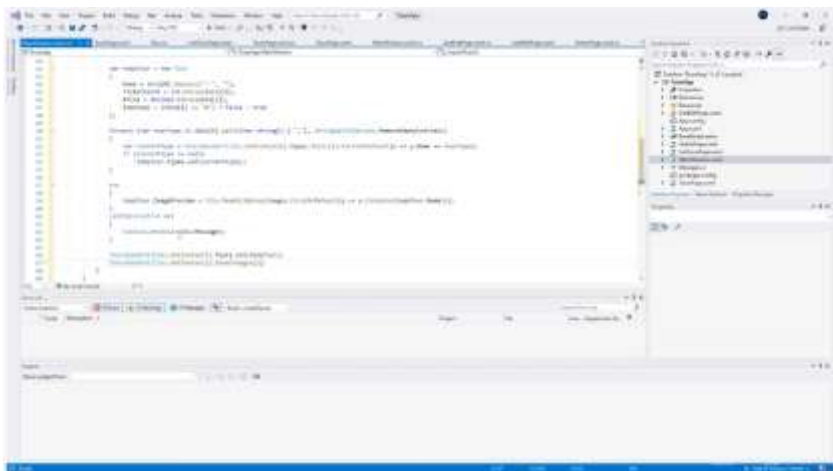
6. Заполняем коллекцию типов тура, выполнив поиск типов по названиям, перечисленным через запятую в файле

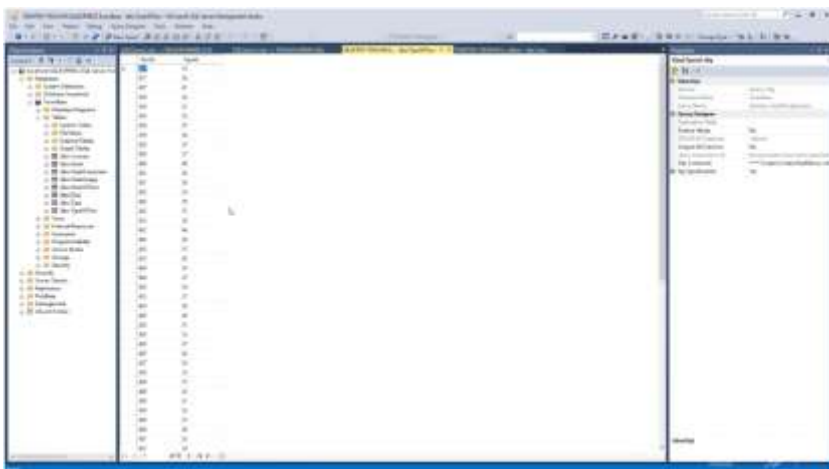


7. Записываем изображение в базу данных с помощью метода ReadAllBytes()



8. Добавляем тур в базу и сохраняем





Модуль 3 Сопровождение и обслуживание программного обеспечения компьютерных систем

Модульное тестирование unit-test

Несовершенное программное решение может оказать колоссальный эффект на генерацию доходов, надежность и репутацию в долгосрочной перспективе. Так, прежде чем доставить ПО клиенту, каждая компания должна гарантировать, что оно работает безупречно и соответствует всем спецификациям и требованиям. Поэтому тестирование уже сейчас становится неотъемлемой и значимой частью жизненного цикла разработки ПО. В этом нам поможет подход TDD (Test Driven Development) – разработка через тестирование. Основные принципы его применения:

- прежде чем писать код реализации некой возможности, пишут тест, который позволяет проверить, работает этот будущий код реализации или нет

- создают реализацию возможностей и добиваются того, чтобы она успешно прошла тестирование

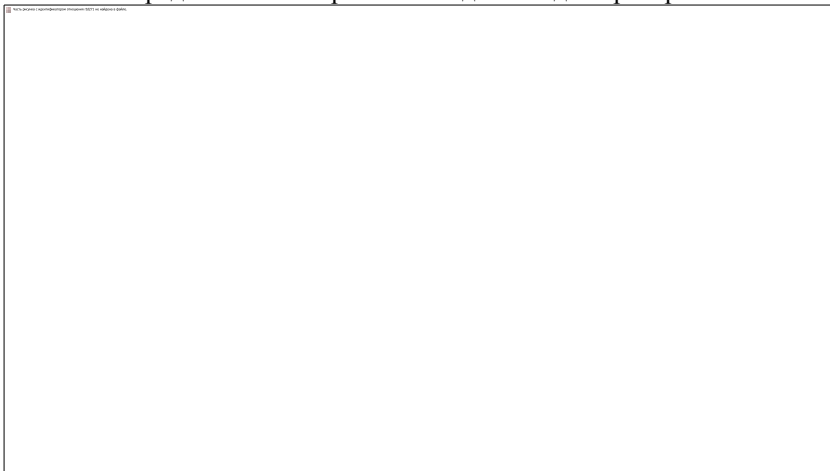
- выполняют, если это нужно, рефакторинг кода (рефакторинг, который при наличии теста способен указать разработчику на правильность или неправильность работы системы, вселяет разработчику уверенность в его действиях)

Наша небольшая простая функция, которую мы будем тестировать, должна всего лишь проверять сложность пароля по

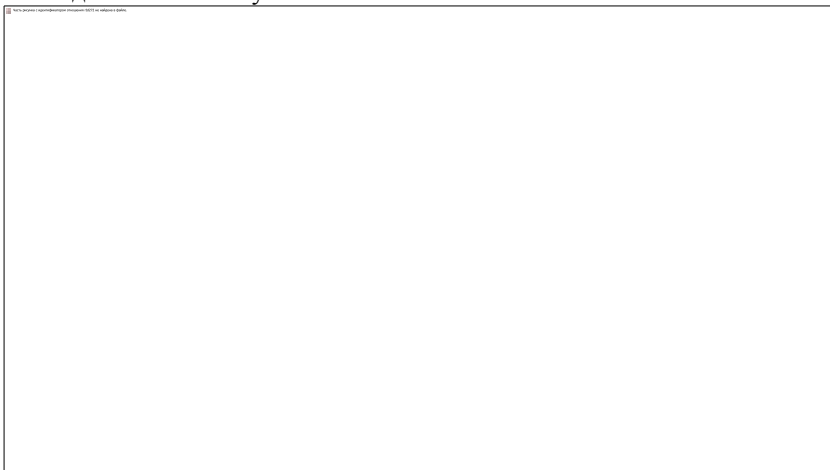
следующим правилам:

- количество символов от 8 до 20
- наличие цифр
- наличие спецсимволов
- наличие прописных и строчных букв

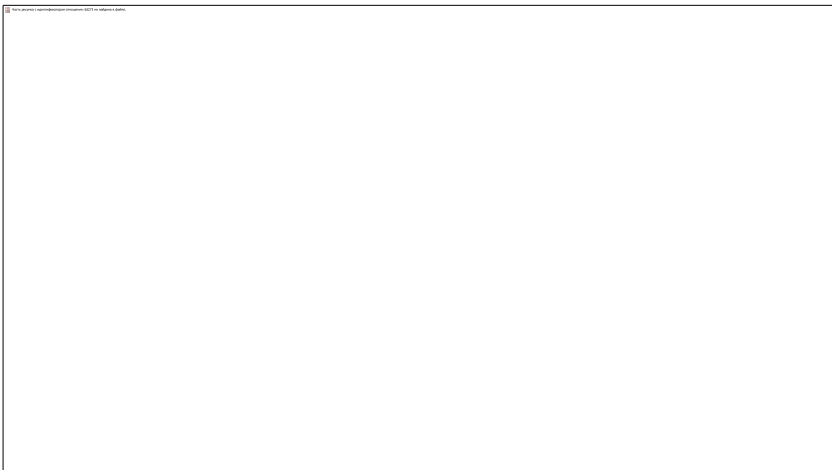
Также нам представлен набор тестовых данных для проверки



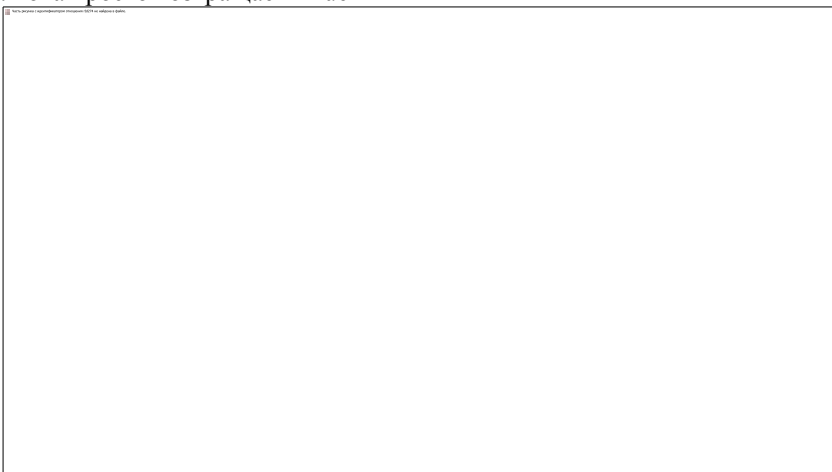
1. Создаем библиотеку .NET Framework



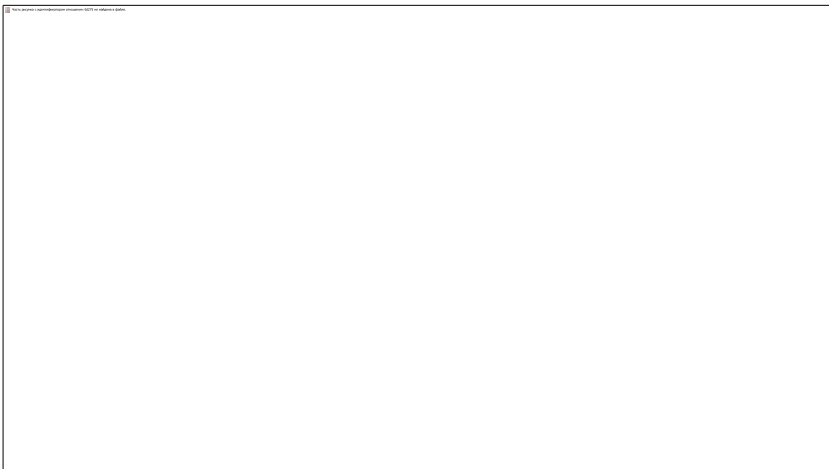
2. Переименовываем стандартный класс в PasswordChecker



3. Затем создаем статичный метод `ValidatePassword`, а в теле метода пока просто возвращаем `True`



4. Затем создаем тестовый проект для этого метода (правой кнопкой мыши – `Create unit-test`), где и будем создавать тесты



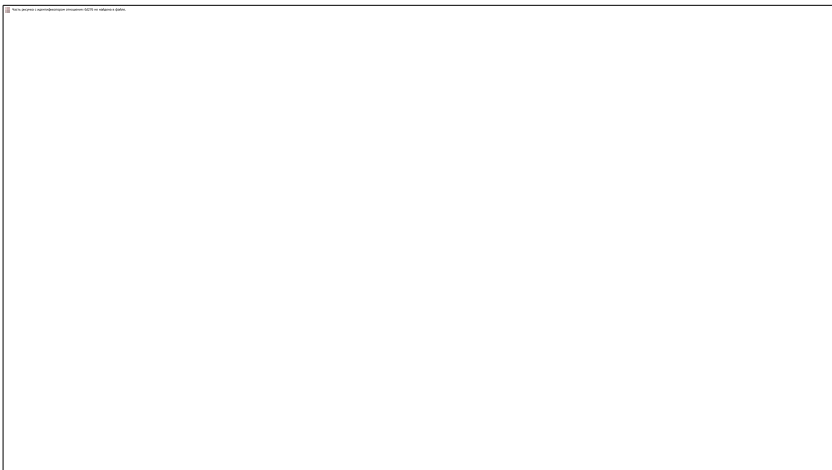
1. Есть очень хороший подход оформления тестов, который формулируется Arrange-Act-Assert. Суть его заключается в том, чтобы в модульном тесте четко определить:

- предусловие (блок Arrange) – устанавливает начальные условия для выполнения теста

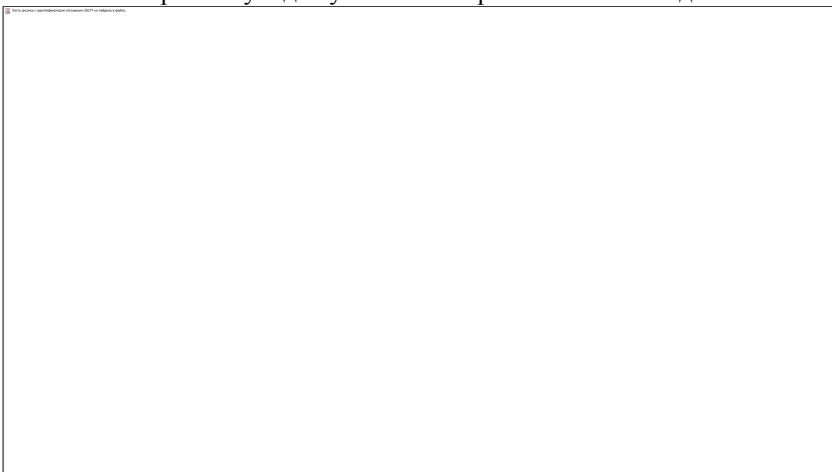
- действие (блок Act) – выполняет сам тест

- постусловие (блок Assert) – верифицирует результат теста, и, в данном случае, оформление – повышает читаемость текста и облегчает его использование в качестве документации к тестируемой функциональности

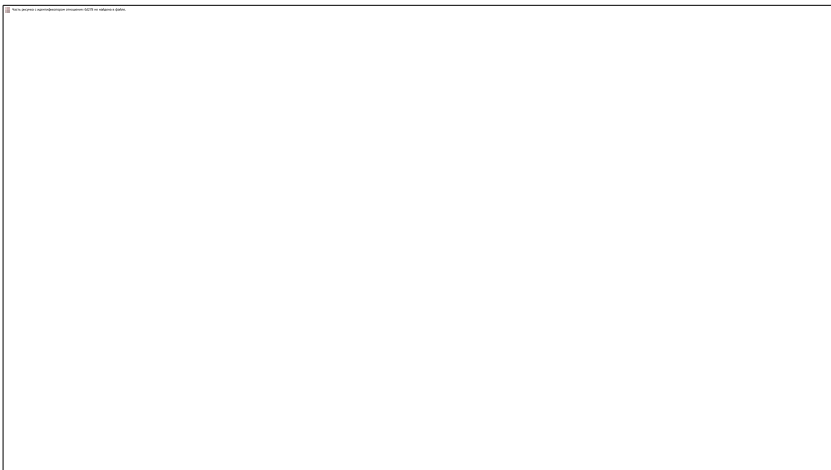
Например, напомним первый тест. Он будет проверять количество символов, где мы разместим блоки Arrange, Act и Assert



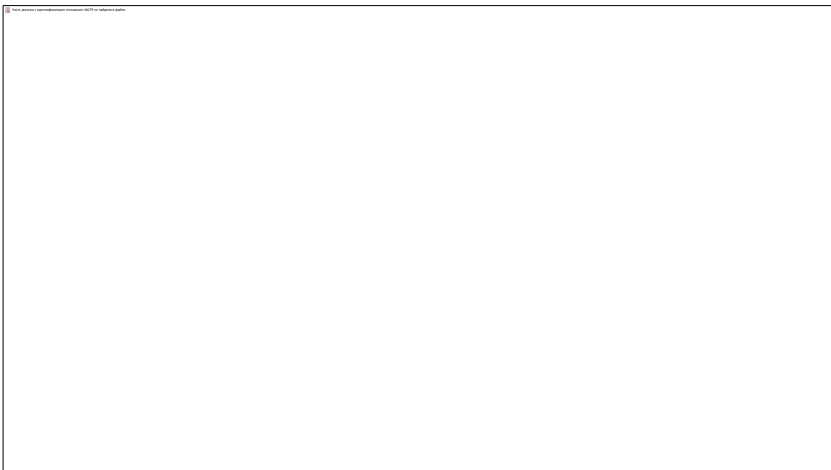
Объявляем переменную для установки пароля из тестовых данных



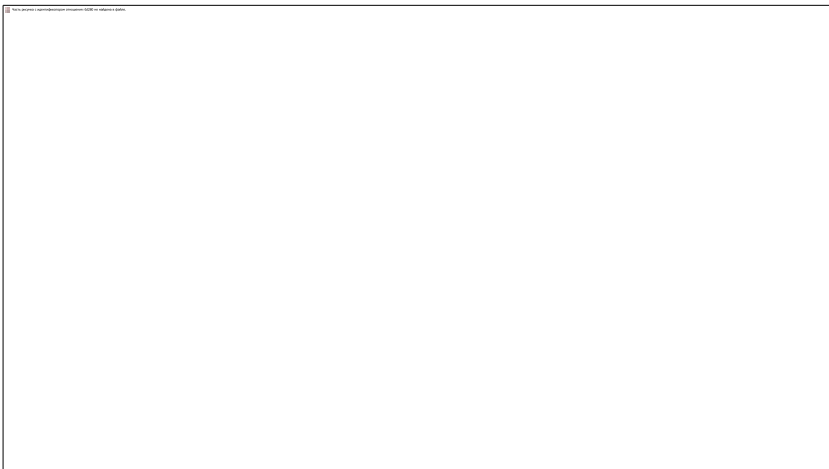
Здесь же объявляем ожидаемое значение в результате выполнения теста



В блоке `Act` создаем переменную, которая вернет актуальный результат при выполнении метода `CheckPassword`. В нашем случае `ValidatePassword`

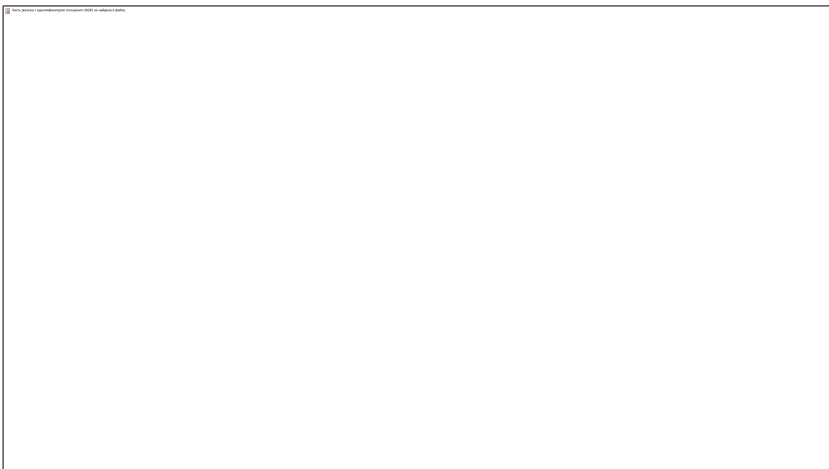


С помощью класса `Assert` сравниваем два значения: ожидаемое и реальное, метод `AreEqual`, и в качестве аргумента – наши данные

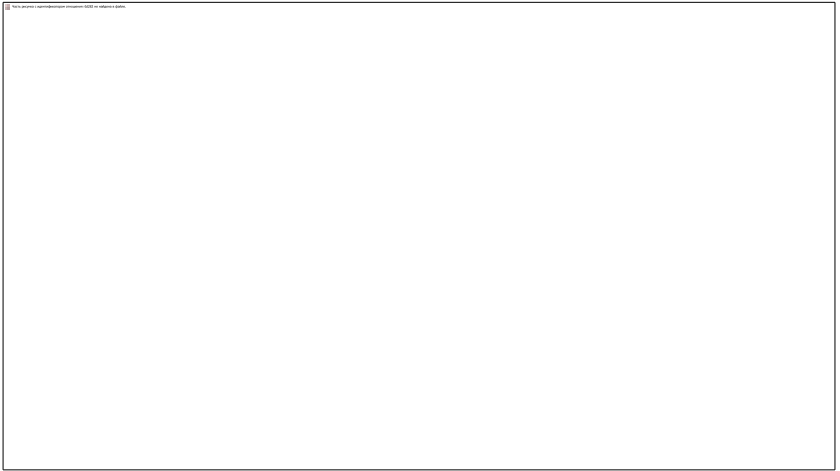


Для проверки результата в классе `Assert`, помимо `AreEqual`, определен ряд методов, среди которых можно выделить, например, следующие: `All`, `Matches`, `Empty`, `IsTrue`, `IsNull`, `Throws` и другие

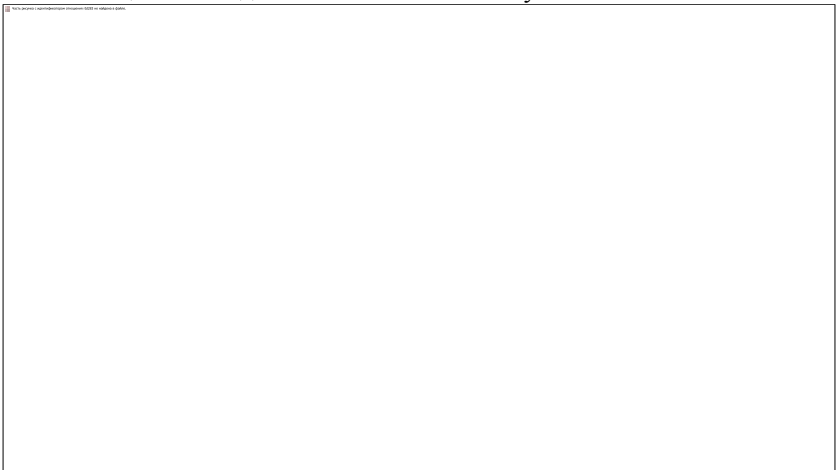
6. Давайте во втором тестовом методе воспользуемся классом `IsFalse` в блоке `Assert`



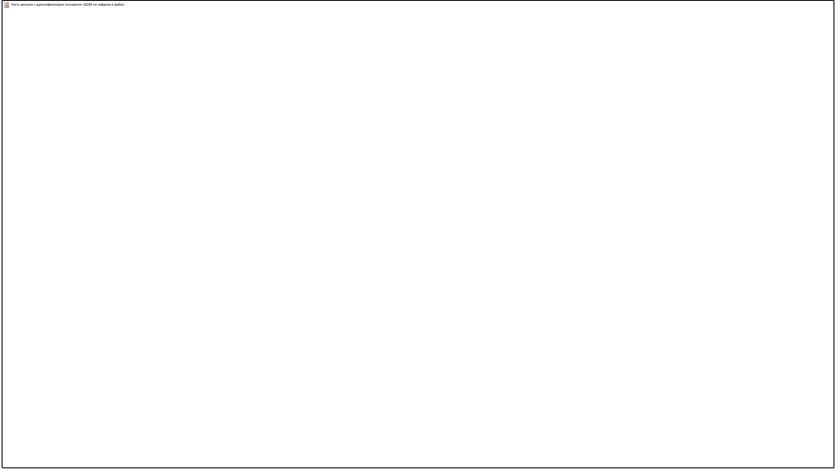
7. Запускаем тесты: на вкладке `Test Windows` можно открыть обозреватель тестов



8. С помощью команды Run All можно их запустить



9. Так как наша функция всегда возвращает True, первый метод проходит, а второй – нет

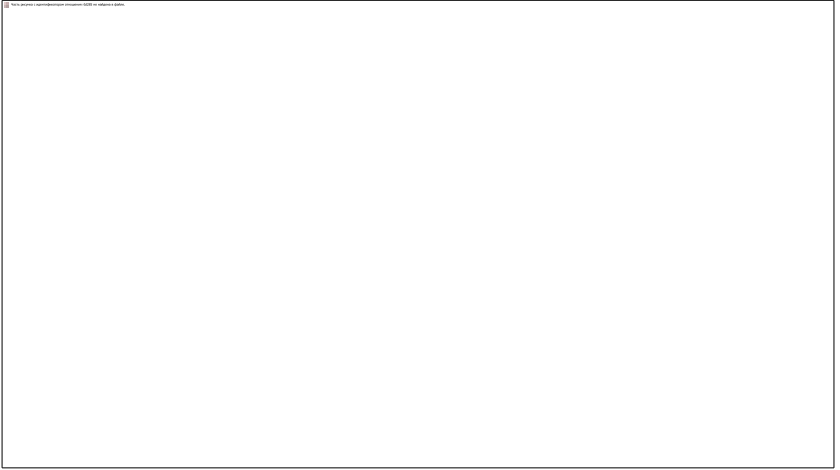


10. Обращаем ваше внимание на наименование тестовых методов. Имя теста должно состоять из трех частей:

- имя тестируемого метода
- сценарий, в котором выполняется тестирование
- ожидаемое поведение при вызове сценария

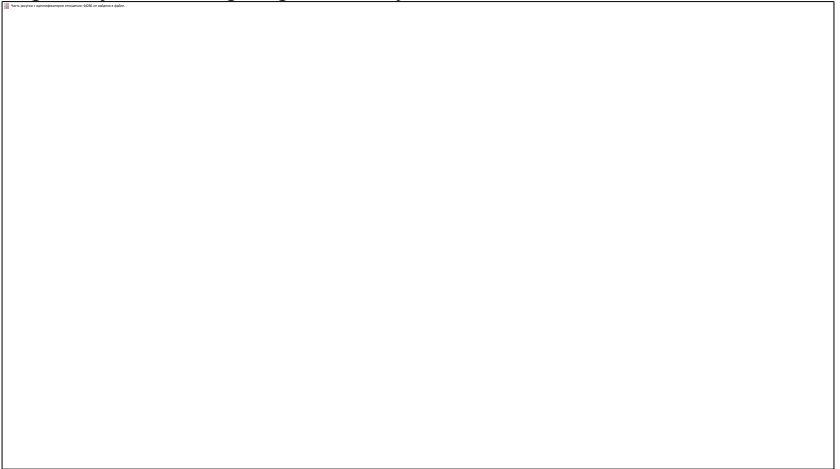
Например, в методе `Check_4SymbolsReturnsFalse` имя тестируемого метода находится в первой части названия – `Check`, затем идет сценарий – то, что у нас используется 4 символа, а затем ожидаемое поведение, у нас вернется `False`.

11. Реализуем все остальные тестовые методы по аналогии, группируя их по требованиям к паролю. Пишем тестовый метод для проверки более чем 20 символов (в данном случае их будет 30)

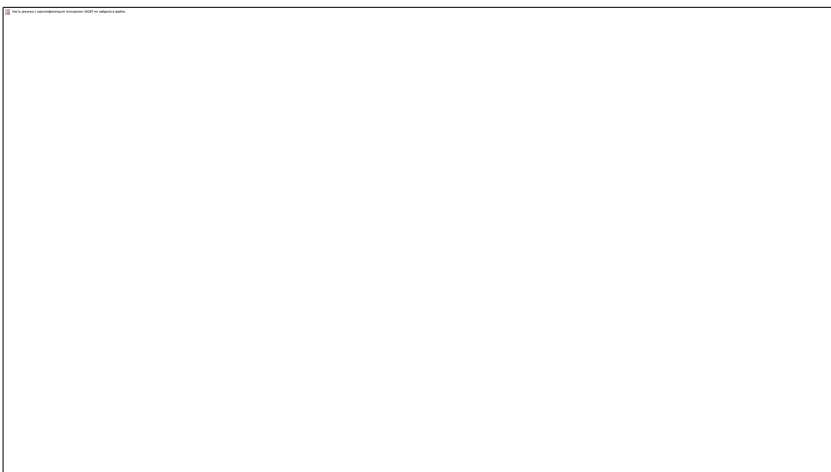


12. Теперь, когда все тесты готовы, самое время приступить к написанию тела метода, проверяющего пароль

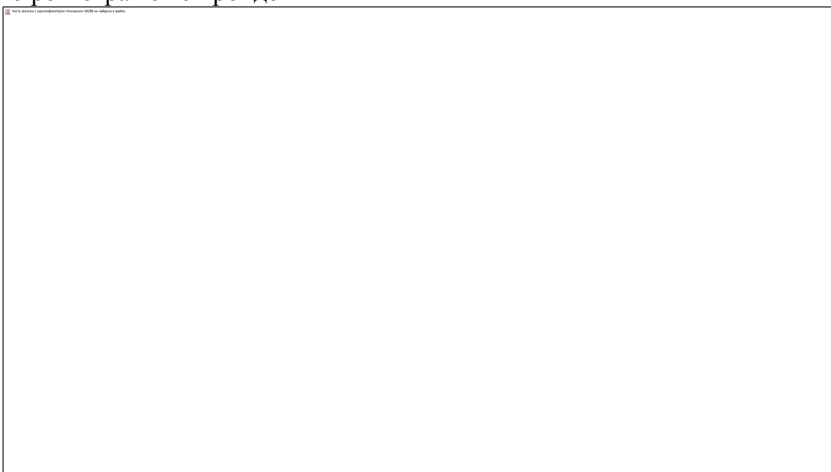
Первым условием проверяем длину



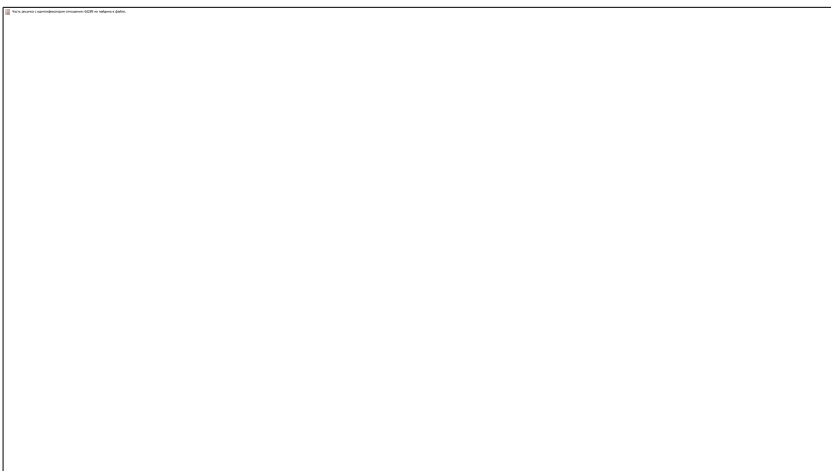
Запускаем тесты. Как видно, тесты на количество символов 34,8 прошли



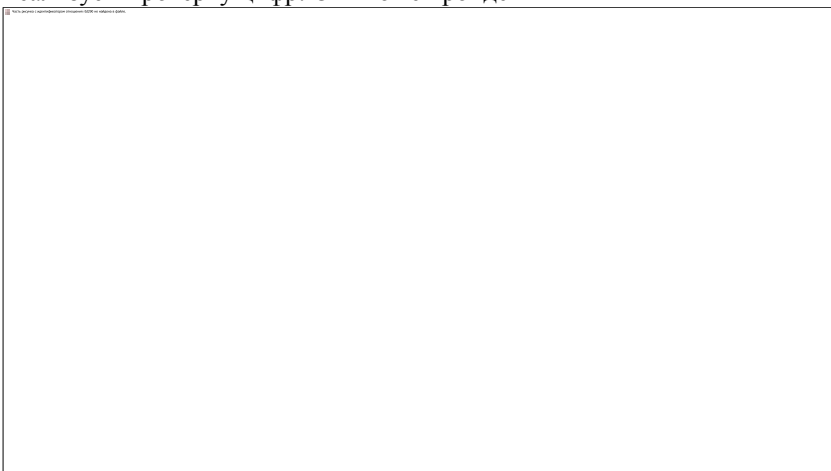
Далее реализуем проверку строчных букв. Два теста на проверку нижнего регистра тоже пройдены



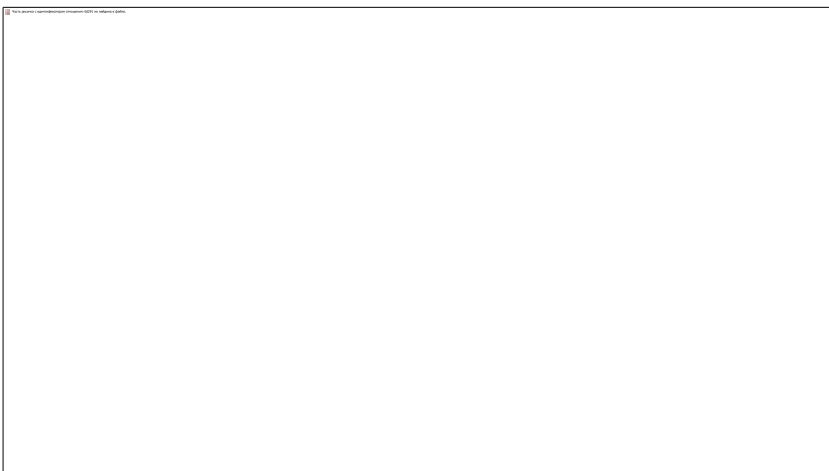
Затем идет проверка заглавных букв. И они тоже оба пройдены



Реализуем проверку цифр. Они тоже пройдены



Реализуем проверку спецсимволов. Метод `Intersect` будет проверять вхождение спецсимволов в пароле



13. Мы обработали все требования и прошли все тесты

Работа с системой контроля версий

Основные возможности системы контроля версий

1. Работа с локальным и удаленным репозиториями
2. Создание и слияние веток
3. Откат к предыдущим версиям

Важно

Локальный репозиторий позволяет управлять версиями на машине разработчика, в то время как удаленный репозиторий позволяет обмениваться данными с другими пользователями

Демонстрация работы с VCS

В данном занятии для управления версиями используется сервис Gogs, построенный на основе Git. Основные возможности VCS демонстрируются на примере написания приложения для вывода названий отелей, полученных с помощью API. В рамках решения задачи в master репозитории разрабатывается метод для вывода списка отелей, а в новой репозитории будет реализована верстка названий. Основные шаги:

- Настройка репозитория в Visual Studio
- Работа в master ветке: разработка функционала
- Работа в новой ветке: изменение верстки, демонстрация отката изменений, слияние веток

Важно

Обычно для решения новой задачи программист заводит новую ветку. После написания кода и его отладки изменения из новой ветки сливаются в master ветку

Настройка репозитория в Visual Studio

Создаем репозиторий

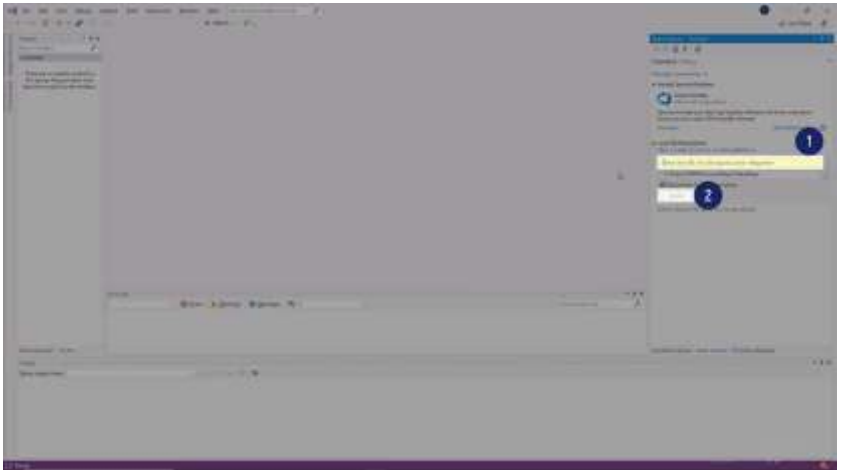
- Открываем шаблон
- Вводим название репозитория
- Выбираем необходимые параметры
- Создаем репозиторий



Клонируем репозиторий в среде разработки

Указываем путь

Клонируем репозиторий



Работа в master ветке

Работаем с кодом

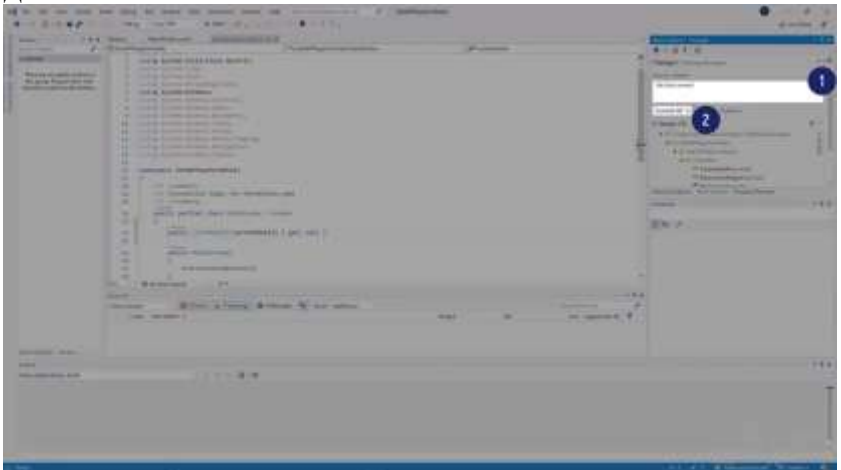
Создаем новый проект в репозитории

Вносим изменения в код

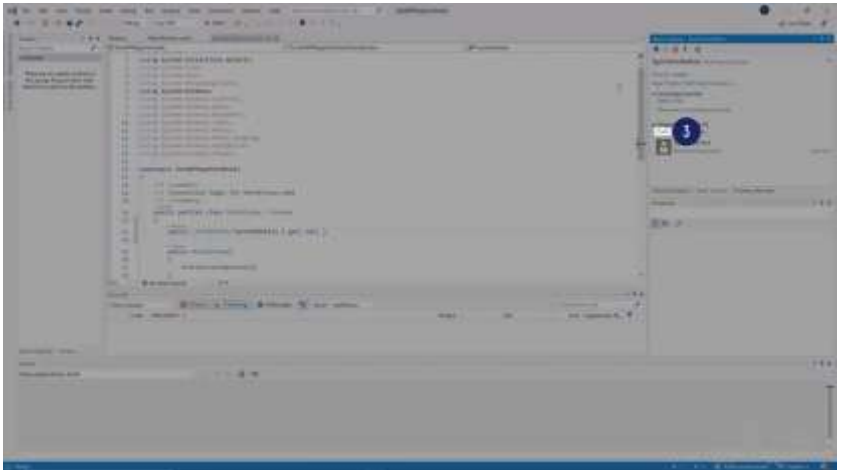
Добавляем изменения в репозиторий

Указываем название commit'a

Делаем commit



4. Делаем push изменений



Важно

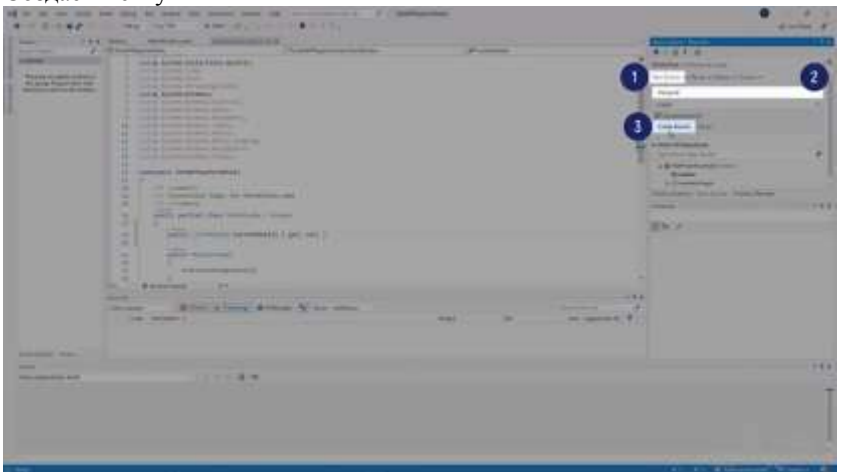
Commit является основным объектом в системе управления версиями и содержит информацию о внесенных изменениях

Создаем новую ветку

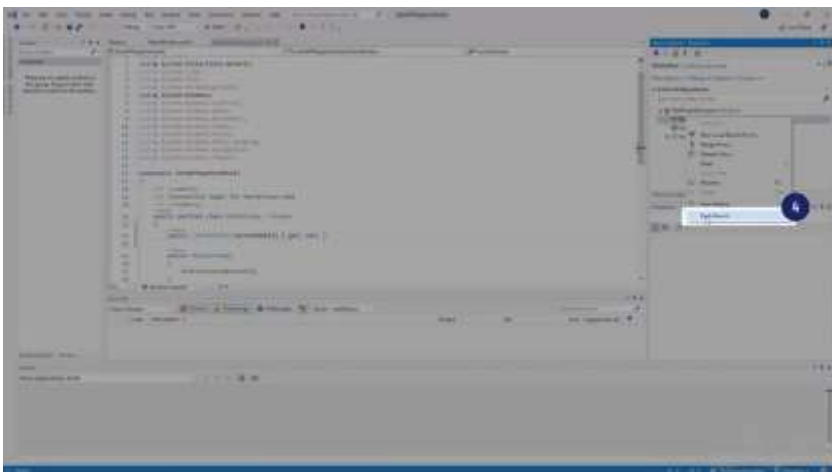
Переходим к созданию ветки

Указываем ее имя

Создаем ветку



4. Делаем push новой ветки



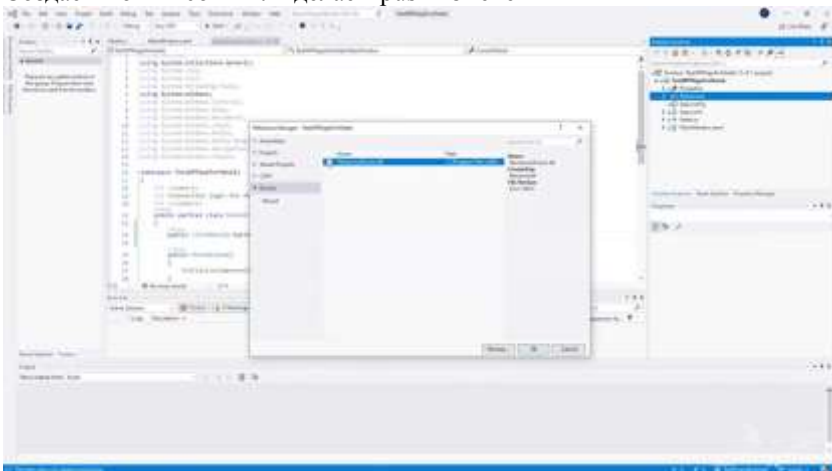
Важно

Ветка в Git является указателем на один из commit'ов, которым чаще всего является последний commit

Изменяем код в master ветке

Добавляем код метода для вывода списка отелей

Создаем новый commit и делаем push изменений



Важно

В новой ветке код данного метода будет отсутствовать, т.к. ветка была создана перед реализацией метода

Работа в новой ветке

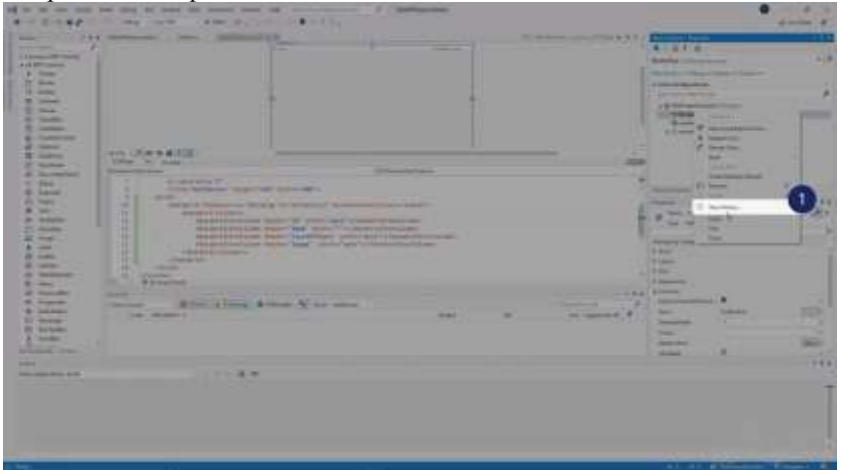
Работаем с кодом

Реализуем верстку названий отелей

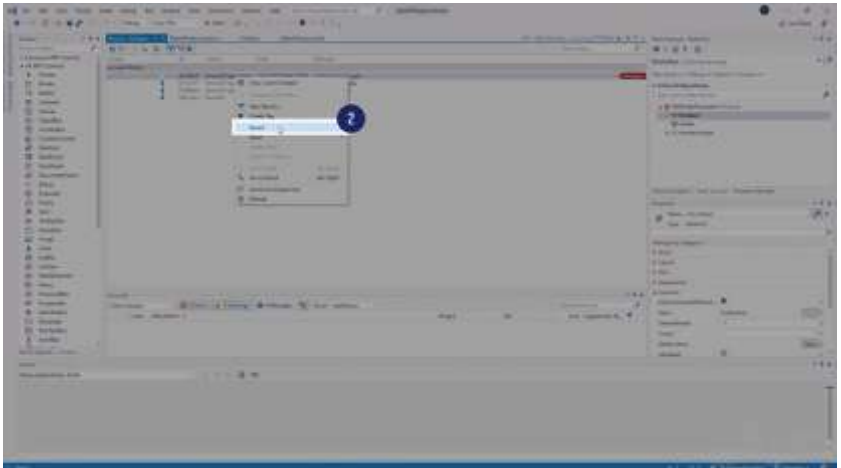
Делаем commit

Как откатить изменения?

Открываем историю commit'ов



2. Откатываем изменения



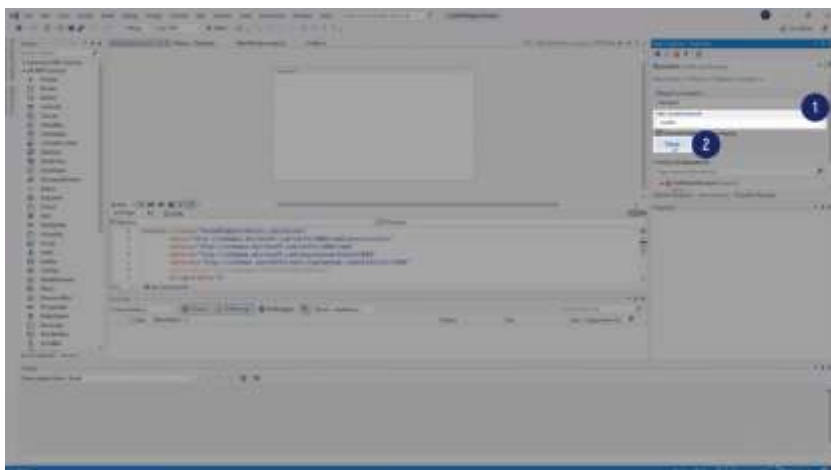
Важно

После отката изменений новая ветка вернется к предыдущему состоянию

Как слить ветки?

Выбираем ветку для слияния

Сливаем ветки



Требования к руководству системного программиста

ГОСТ 19.503-79 Руководство системного программиста.

Требования к содержанию и оформлению

ГОСТ 19.503-79 Руководство системного программиста.

Требования к содержанию и оформлению

УДК 651.7/.78:681.3.06:002:006.354

Группа Т55

ГОСУДАРСТВЕННЫЙ СТАНДАРТ СОЮЗА ССР

Единая система программной документации

*РУКОВОДСТВО СИСТЕМНОГО ПРОГРАММИСТА.
ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И
ОФОРМЛЕНИЮ*

ГОСТ 19.503-
79*

(СТ СЭВ

United system for program documentation.
System programmer's guide. Requirements to contents and
form of presentation

2094-80)

Постановлением Государственного комитета стандартов Совета Министров СССР от 12 января 1979 г. № 74 срок введения установлен

с 01.01. 1980 г.

Настоящий стандарт устанавливает требования к содержанию и оформлению программного документа «Руководство системного программиста», определённого [ГОСТ 19.101-77](#).

Стандарт полностью соответствует СТ СЭВ 2094-80.

1. ОБЩИЕ ПОЛОЖЕНИЯ

1.1. Структуру и оформление документа устанавливают в соответствии с [ГОСТ 19.105-78](#).

Составление информационной части (аннотации и содержания) является обязательным.

1.2. Руководство системного программиста должно содержать следующие разделы:

- общие сведения о программе;
- структура программы;
- настройка программы;
- проверка программы;
- дополнительные возможности;
- сообщения системному программисту.

В зависимости от особенностей документы допускается объединять отдельные разделы или вводить новые.

В обоснованных случаях допускается раздел «Дополнительные возможности» не приводить, а в наименованиях разделов опускать слово «программа» или заменять его на «наименование программы».

(Измененная редакция, Изм. № 1).

2. СОДЕРЖАНИЕ РАЗДЕЛОВ

2.1. В разделе «Общие сведения о программе» должна быть указаны назначение и функции программы и сведения о технических и программных средствах, обеспечивающих выполнение данной программы.

2.2. В разделе «Структура программы» должны быть приведены сведения о структуре программы, ее составных частях, о связях между составными частями и о связях с другими программами.

2.3. В разделе «Настройка программы» должно быть приведено описание действий по настройке программы на условия конкретного применения (настройка на состав технических средств, выбор функций и др.).

При необходимости приводят поясняющие примеры.

2.4. В разделе «Проверка программы» должны быть приведено описание способов проверки, позволяющих дать общее заключение о работоспособности программы (контрольные примеры, методы прогона, результаты).

2.5. В разделе «Дополнительные возможности» должно быть приведено описание дополнительных разделов функциональных возможностей программы и способов их выбора.

2.6. В разделе «Сообщения системному программисту» должны быть указаны тексты сообщений, выдаваемых в ходе выполнения настройки, проверки программы, а также в ходе выполнения программы, описание их содержания и действий, которые необходимо предпринять по этим сообщениям.

2.7. В приложении к руководству системного программиста могут быть приведены дополнительные материалы (примеры, иллюстрации, таблицы, графики и т.п.).

3 ИНФОРМАЦИОННО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Основные источники

1. Дадаян, Э. Г. Конфигурирование и моделирование в системе «1С: Предприятие» [Электронный ресурс] : учебник / Э. Г. Дадаян. — Москва : Вузовский учебник : ИНФРА-М, 2019. — 417 с. + Доп. Материалы. - Режим доступа: <https://new.znanium.com/read?id=327817> – Загл. с экрана.
2. Зыков, С. В. Программирование. Объектно-ориентированный подход [Электронный ресурс] : учебник и практикум для академического бакалавриата / С. В. Зыков. — Москва : Издательство Юрайт, 2019. — 155 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-00850-0. — Режим доступа: <https://urait.ru/bcode/434106> – Загл. с экрана.
3. Кузнецов, А. С. Системное программирование [Электронный ресурс] : учебное пособие / А. С. Кузнецов, И. А. Якимов, П. В. Пересунько. - Красноярск : Сиб. федер. ун-т 2018. - 170с. - ISBN 978-5-7638-3885-5. - Режим доступа: <https://new.znanium.com/read?id=342172> – Загл. с экрана.
4. Соколова, В. В. Разработка мобильных приложений [Электронный ресурс] : учебное пособие для среднего профессионального образования / В. В. Соколова. — Москва : Издательство Юрайт, 2019. — 175 с. — (Профессиональное образование). — ISBN 978-5-534-10680-0. — Режим доступа: <https://urait.ru/bcode/431172> – Загл. с экрана.
5. Гагарина, Л. Г. Технология разработки программного обеспечения [Электронный ресурс] : учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул ; под ред. Л. Г. Гагариной. — Москва : ИД «ФОРУМ» : ИНФРА-М, 2019. — 400 с. — Режим доступа: <https://new.znanium.com/read?id=336552> – Загл. с экрана.
6. Черткова, Е. А. Программная инженерия. Визуальное моделирование программных систем [Электронный ресурс] : учебник для среднего профессионального образования / Е. А. Черткова. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 147 с. — (Профессиональное образование). — ISBN 978-5-534-09823-5. — Режим доступа: <https://urait.ru/bcode/441255> – Загл. с экрана.
7. Федорова, Г. Н. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности [Электронный ресурс] : учеб. пособие / Г.Н. Федорова. — М. :КУРС : ИНФРА-М, 2019. — 336 с. (Среднее Профессиональное Образование). - Режим доступа: <https://new.znanium.com/read?id=330691> – Загл. с экрана.
8. Казарин, О. В. Основы информационной безопасности: надежность и безопасность программного обеспечения [Электронный ресурс] :

учебное пособие для среднего профессионального образования / О. В. Казарин, И. Б. Шубинский. — Москва : Издательство Юрайт, 2019. — 342 с. — (Профессиональное образование). — ISBN 978-5-534-10671-8. — Режим доступа: <https://urait.ru/bcode/431080> — Загл. с экрана.

Дополнительные источники

1. Дадаян, Э. Г. Основы языка программирования 1С 8.3 [Электронный ресурс] : учебное пособие / Э. Г. Дадаян. — Москва : Вузовский учебник: ИНФРА-М, 2018. — 132 с. - Режим доступа: <https://new.znanium.com/read?id=333502> — Загл. с экрана.
2. Соколова, В. В. Вычислительная техника и информационные технологии. Разработка мобильных приложений [Электронный ресурс] : учебное пособие для прикладного бакалавриата / В. В. Соколова. — Москва : Издательство Юрайт, 2019. — 175 с. — (Университеты России). — ISBN 978-5-9916-6525-4. — Режим доступа: <https://urait.ru/bcode/433981> — Загл. с экрана.
3. Хорев, П. Б. Объектно-ориентированное программирование с примерами на C# [Электронный ресурс] : учеб. пособие / П. Б. Хорев. — Москва : ФОРУМ : ИНФРА-М, 2019. — 200 с. — (Высшее образование: Бакалавриат). - Режим доступа: <https://new.znanium.com/read?id=339308> — Загл. с экрана.
4. Черпаков, И. В. Основы программирования [Электронный ресурс] : учебник и практикум для среднего профессионального образования / И. В. Черпаков. — Москва : Издательство Юрайт, 2019. — 219 с. — (Профессиональное образование). — ISBN 978-5-9916-9984-6. — Режим доступа: <https://urait.ru/bcode/436557> — Загл. с экрана.
5. Федорова, Г. Н. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности [Электронный ресурс] : учебное пособие / Г. Н. Федорова. — М. : КУРС : ИНФРА-М, 2019. — 336 с. (Среднее Профессиональное Образование). - Режим доступа: <https://new.znanium.com/read?id=330691> — Загл. с экрана.
6. Хорев, П. Б. Объектно-ориентированное программирование с примерами на C# [Электронный ресурс] : учебное пособие / П. Б. Хорев. — Москва : ФОРУМ : ИНФРА-М, 2019. — 200 с. — Режим доступа: <https://new.znanium.com/read?id=339308> — Загл. с экрана.
7. Баранова, Е. К. Основы информационной безопасности [Электронный ресурс] : учебник / Е. К. Баранова, А. В. Бабаши. - Москва : РИОР: ИНФРА-М, 2019. - 202 с. - (Среднее профессиональное образование). - Режим доступа: <https://new.znanium.com/read?id=339532> — Загл. с экрана.

8. Казарин, О. В. Надежность и безопасность программного обеспечения [Электронный ресурс] : учебное пособие для бакалавриата и магистратуры / О. В. Казарин, И. Б. Шубинский. — Москва : Издательство Юрайт, 2019. — 342 с. — (Бакалавр и магистр. Модуль). — ISBN 978-5-534-05142-1. — Режим доступа: <https://www.urait.ru/bcode/441287> – Загл. с экрана.
9. Гагарина, Л. Г. Разработка и эксплуатация автоматизированных информационных систем [Электронный ресурс] : учеб. пособие / Л.Г. Гагарина. — Москва : ИД «ФОРУМ» : ИНФРА-М, 2019. — 384 с. — (Среднее профессиональное образование). - Режим доступа: <https://new.znanium.com/read?id=333679> – Загл. с экрана.
10. Маркин, А. В. Программирование на SQL в 2 ч. Часть 1 [Электронный ресурс] : учебник и практикум для вузов / А. В. Маркин. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2019. — 403 с. — (Высшее образование). — ISBN 978-5-534-12256-5. — Режим доступа: <https://urait.ru/bcode/447115> – Загл. с экрана.
11. Маркин, А. В. Программирование на SQL в 2 ч. Часть 2 [Электронный ресурс] : учебник и практикум для вузов / А. В. Маркин. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 340 с. — (Высшее образование). — ISBN 978-5-534-12258-9. — Режим доступа: <https://urait.ru/bcode/448191> – Загл. с экрана.
12. Мартишин, С. А. Базы данных. Практическое применение СУБД SQL и NoSQL-типа для проектирования информационных систем [Электронный ресурс] : учеб. пособие / С. А. Мартишин, В. Л. Симонов, М. В. Храпченко. — Москва : ИД «ФОРУМ» : ИНФРА-М, 2019. — 368 с. — (Высшее образование: Бакалавриат). - Режим доступа: <https://new.znanium.com/read?id=333330> – Загл. с экрана.

Интернет-ресурсы

1. Интуит Национальный открытый университет курс Основы разработки приложений для мобильных устройств на платформе Windows Phone
2. Интуит Национальный открытый университет курс Ассемблер в Linux для программистов на C <https://www.intuit.ru/studies/courses/3537/779/info>
3. Интуит Национальный открытый университет курс Основы тестирования программного обеспечения https://www.intuit.ru/studies/professional_retraining/941/courses/48/info
4. Интуит Национальный открытый университет курс Язык UML 2 в анализе и проектировании программных систем и бизнес-

- процессов [Электронный ресурс]. – Режим доступа: <http://www.intuit.ru/studies/courses/480/336/info>, свободный.– Загл. с экрана. Яз. рус.3.3 Учебно-методическое обеспечение самостоятельной работы обучающихся
5. Интуит Национальный открытый университет курс Проектирование информационных систем в Microsoft SQL Server 2008 и Visual Studio 2008 [Электронный ресурс]. – Режим доступа: <http://www.intuit.ru/studies/courses/502/358/info>, свободный.– Загл. с экрана. Яз. рус.
 6. Интуит – национальный открытый университет. [Электронный ресурс]. Администрирование MySQL – Режим доступа: <https://www.intuit.ru/studies/courses/989/165/info>, свободный. – Загл. с экрана. Яз. рус.
 7. Интуит – национальный открытый университет. [Электронный ресурс]. Разработка и защита баз данных в Microsoft SQL Server 2005 – Режим доступа: <https://www.intuit.ru/studies/courses/1141/263/info>, свободный. – Загл. с экрана. Яз. рус.
 8. Практическое владение языком SQL.[Электронный ресурс]. – Режим доступа: <http://www.sql-ex.ru>, свободный.– Загл. с экрана. Яз. рус.
 9. Интуит Национальный открытый университет курс Методы и средства инженерии программного обеспечения [Электронный ресурс]. – Режим доступа: <https://www.intuit.ru/studies/courses/2190/237/info>, свободный.– Загл. с экрана. Яз. рус.
 10. Интуит Национальный открытый университет курс Процессы анализа и управления рисками в области ИТ [Электронный ресурс]. – Режим доступа: <https://www.intuit.ru/studies/courses/3506/748/info>, свободный.– Загл. с экрана. Яз. рус.