АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАСЗКИЙ СОЦИАЛЬНЫЙ ИНСТИТУТ»

Утверждаю
Декан факультета
_____Ж.В. Игнатенко
«15» сентября 2025 г.

Методические указания
к семинарам и по выполнению самостоятельной работы
ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА

Специальность: 09.02.11 Разработка и управление программным обеспечением

Квалификация: программист

Направленность: разработка приложений для мобильных платформ

Форма обучения: очная

Разработана
Канд. пед. наук, доцент
_____ Горбатовская Н.Н.

Согласована
зав. выпускающей кафедры
_____Д.Г. Ловянников

Рекомендована
на заседании кафедры ПИМ
от «15» сентября 2025г.
протокол № 2
Зав. кафедрой _____ Д.Г. Ловянников

Одобрена
на заседании учебно-методической комиссии факультета
от «15» сентября 2025 г.
протокол № 2
ПредседательУМК_____Ж.В. Игнатенко

Ставрополь, 2025 г.

СОДЕРЖАНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Методические указания обучающимся по дисциплине «Программные решения для бизнеса» предназначены для методического обеспечения практической и самостоятельной работы студента.

Основное назначение – помочь студенту самостоятельно, без помощи преподавателя, углубить теоретические знания и практические навыки по дисциплине.

Преподаватель оказывает методическую помощь, осуществляет контроль за качеством подготовки и проведения занятий.

1. Методические указания обучающимся при подготовке к практическим занятиям

Практическое занятие – форма организации обучения, при которой преподаватель организует детальное рассмотрение студентами отдельных теоретических положений дисциплины и формирует умение и навыки их практического применения в индивидуальном исполнении в соответствии со сформулированными задачами.

При подготовке к практическим (семинарским) занятиям можно выделить 2 этапа:

1-й - организационный,

2-й - закрепление и углубление теоретических знаний.

На первом этапе обучающийся планирует свою самостоятельную работу, которая включает:

-уяснение задания на самостоятельную работу;

-подбор рекомендованной литературы;

-составление плана работы, в котором определяются основные пункты предстоящей подготовки.

Составление плана дисциплинирует и повышает организованность в работе.

Второй этап включает непосредственную подготовку обучающегося к занятию. Начинать надо с изучения рекомендованной литературы. Необходимо помнить, что на лекции обычно рассматривается не весь материал, а только его часть. Остальная его часть восполняется в процессе самостоятельной работы. В связи с этим работа с рекомендованной литературой обязательна. Особое внимание при этом необходимо обратить на содержание основных положений и выводов, объяснение явлений и фактов, уяснение практического приложения рассматриваемых теоретических вопросов.

В процессе этой работы обучающийся должен стремиться понять и запомнить основные положения рассматриваемого материала, примеры, поясняющие его, а также разобраться в иллюстративном материале.

Заканчивать подготовку следует составлением плана (перечня основных пунктов) по изучаемому материалу (вопросу). Такой план позволяет составить концентрированное, сжатое представление по изучаемым вопросам. В процессе подготовки к практическому (семинарскому) занятию рекомендуется взаимное обсуждение материала, во время которого закрепляются знания, а также приобретается практика в изложении и разъяснении полученных знаний, развивается речь.

При необходимости следует обращаться за консультацией к преподавателю. Идя на консультацию, необходимо хорошо продумать вопросы, которые требуют разъяснения. В начале практического (семинарского) занятия, обучающиеся под руководством преподавателя, более глубоко осмысливают теоретические положения по теме занятия, раскрывают и объясняют основные положения и практическое применение. В процессе творческого обсуждения и дискуссии вырабатываются умения и навыки использовать приобретенные знания для решения практических задач.

Для того чтобы практические занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по лекционному материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. После усвоения лекционного материала, он будет закрепляться на практических занятиях путем решения проблемных и прикладных задач. При этих условиях обучающийся не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул для активной проработки лекции.

Решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, с выводом и ответами на контрольные вопросы.

При подготовке к практическим занятиям следует использовать основную и дополнительную литературу из рабочей программы дисциплины, а также руководствоваться приведенными указаниями и рекомендациями.

Обучающемуся рекомендуется следующая схема подготовки к практическому занятию:

1) проработать конспект лекций;
2) изучить основную и дополнительную литературу;
3) выделить проблемные области;
4) ответить на контрольные вопросы;
5) проработать тестовые задания и задачи (если таковые имеются);
6) при затруднениях сформулировать вопросы к преподавателю.

Методические указания к практическим работам в приложении А.

2. Методические указания обучающимся по выполнению внеаудиторной (самостоятельной) работы

Внеаудиторная самостоятельная работа – это планируемая учебная, учебно-исследовательская работа студентов, выполняемая во внеаудиторное время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Внеаудиторная самостоятельная работа производится с целью:

- систематизации и закрепления полученных теоретических знаний и практических умений студентов;
- углубления и расширения теоретических знаний;
- формирования дополнительных практических профессиональных навыков;
- развития познавательной способности и активности студентов, творческой инициативы, самостоятельности, ответственности, организованности;
- формирования самостоятельного мышления, способностей к самообразованию, самосовершенствованию и самореализации.

Критериями оценки результатов внеаудиторной самостоятельной работы студента являются:

- уровень освоения учебного материала;
- умение использовать теоретические знания и умения при выполнении практических задач;
- уровень сформированности компетенций, предусмотренных основной образовательной программой.

В ходе дисциплины «Программные решения для бизнеса» предлагаются следующие формы и виды самостоятельной работы студентов:

1. Чтение основной и дополнительной литературы. Самостоятельное изучение материала по литературным источникам.

2. Подготовка к лекциям.

3. Поиск необходимой информации в сети Интернет.

4. Конспектирование источников.

5. Подготовка реферата, доклада, презентации. Подготовка к защите и защита выполненной самостоятельной работы.

Примерные темы для самостоятельной работы (доклад, презентация или индивидуальное задание):

- Современные информационные технологии.
- Отечественное программное обеспечение.

2.1. Указания по подготовке к лекциям

Подготовка к лекциям предполагает изучение рабочей программы дисциплины, установление связи с ранее полученными знаниями, выделение наиболее значимых и актуальных проблем, на изучение которых следует обратить особое внимание.

Самостоятельная работа начинается до прихода обучающегося на лекцию. Обучающимся необходимо использовать «систему опережающего чтения», то есть предварительно прочитывать лекционный материал, содержащийся в учебниках и учебных пособиях, закладывая базу для более глубокого восприятия лекции.

Кроме того, самостоятельная подготовка обучающегося к лекции должна состоять в перечитывании конспекта предыдущей лекции. Это помогает лучше понять материал новой лекции, опираясь на предшествующие знания.

Чтобы понимать излагаемый лектором материал, обучающийся должен знать пройденные ранее темы и полученные практические знания, понимать все особенности изученных ранее тем. Этими свойствами и особенностями определяется и постановка новых задач на последующих лекциях, и характер решения этих задач. От них зависят характеристики других, более сложных объектов, подлежащих изучению на последующих лекциях.

Главное в период подготовки к лекционным занятиям - научиться методам самостоятельного умственного труда, сознательно развивать свои творческие способности и овладевать навыками творческой работы.

2.2. Указания по конспектированию источников

Конспект – это краткое последовательное изложение содержания статьи, книги, лекции. Его основу составляют план тезисы, выписки, цитаты. Конспект, в отличие от тезисов воспроизводят не только мысли оригинала, но и связь между ними. В конспекте отражается не только то, о чем говорится в работе, но и что утверждается, и как доказывается.

В отличие от тезисов и выписок, конспекты при обязательной краткости содержат не только основные положения и выводы, но и факты, и доказательства, и примеры, и иллюстрации.

Типы конспектов:

1. Плановый.

2. Текстуальный.

3. Свободный.

4. Тематический.

Краткая характеристика типов конспектов:

1. Плановый конспект: являясь сжатым, в форме плана, пересказом прочитанного, этот конспект – один из наиболее ценных, помогает лучше усвоить материал еще в процессе его изучения. Он учит последовательно и четко излагать свои мысли, работать над книгой, обобщая содержание ее в формулировках плана. Такой конспект краток, прост и ясен по своей форме. Это делает его незаменимым пособием при быстрой подготовке доклада, выступления. Недостаток: по прошествии времени с момента написания трудно восстановить в памяти содержание источника.

2. Текстуальный конспект– это конспект, созданный в основном из отрывков подлинника – цитат. Это прекрасный источник дословных высказываний автора и приводимых им фактов. Текстуальный конспект используется длительное время. Недостаток: не активизирует резко внимание и память.

3. Свободный конспект – представляет собой сочетание выписок, цитат, иногда тезисов, часть его текста может быть снабжена планом. Это наиболее полноценный вид конспекта.

4. Тематический конспект – дает более или менее исчерпывающий ответ на поставленный вопрос темы. Составление тематического конспекта учит работать над темой, всесторонне обдумывая ее, анализируя различные точки зрения на один и тот же вопрос. Таким образом, этот конспект облегчает работу над темой при условии использования нескольких источников.

5. Конспект-схема. Удобно пользоваться схематичной записью прочитанного. Составление конспектов-схем служит не только для запоминания материала. Такая работа становится средством развития способности выделять самое главное, существенное в учебном материале, классифицировать информацию.

Алгоритм составления конспекта:

- Определите цель составления конспекта.
- Читая изучаемый материал, подразделяйте его на основные смысловые части, выделяйте главные мысли, выводы.
- Если составляется план-конспект, сформулируйте его пункты и определите, что именно следует включить в план-конспект для раскрытия каждого из них.
- Наиболее существенные положения изучаемого материала (тезисы) последовательно и кратко излагайте своими словами или приводите в виде цитат.
- В конспект включаются не только основные положения, но и обосновывающие их выводы, конкретные факты и примеры (без подробного описания).
- Составляя конспект, можно отдельные слова и целые предложения писать сокращенно, выписывать только ключевые слова, вместо цитирования делать лишь ссылки на страницы конспектируемой работы, применять условные обозначения.
- Чтобы форма конспекта как можно более наглядно отражала его содержание, располагайте абзацы "ступеньками" подобно пунктам и подпунктам плана, применяйте разнообразные способы подчеркивания, используйте карандаши и ручки разного цвета.
- Используйте реферативный способ изложения (например: "Автор считает...", "раскрывает...").
- Собственные комментарии, вопросы, раздумья располагайте на полях.

Правила конспектирования.

Для грамотного написания конспекта необходимо:

1. Записать название конспектируемого произведения (или его части) и его выходные данные.

2. Осмыслить основное содержание текста, дважды прочитав его.

3. Составить план - основу конспекта.

4. Конспектируя, оставить место (широкие поля) для дополнений, заметок, записи незнакомых терминов и имен, требующих разъяснений.

5. Помнить, что в конспекте отдельные фразы и даже отдельные слова имеют более важное значение, чем в подробном изложении.

6. Запись вести своими словами, это способствует лучшему осмыслению текста.

7. Применять определенную систему подчеркивания, сокращений, условных обозначений.

8. Соблюдать правила цитирования - цитату заключать в кавычки, давать ссылку на источник с указанием страницы.

9. Научитесь пользоваться цветом для выделения тех или иных информативных узлов в тексте. У каждого цвета должно быть строго однозначное, заранее предусмотренное назначение. Например, если вы пользуетесь синими чернилами для записи конспекта, то: красным цветом - подчеркивайте названия тем, пишите наиболее важные формулы; черным - подчеркивайте заголовки подтем, параграфов, и т.д.; зеленым - делайте выписки цитат, нумеруйте формулы и т.д. Для выделения большой части текста используется отчеркивание.

10. Учитесь классифицировать знания, т.е. распределять их по группам, параграфам, главам и т.д. Для распределения можно пользоваться буквенными обозначениями, русскими или латинскими, а также цифрами, а можно их совмещать.

2.3. Указания по изучению рекомендованной литературы

Этот вид работы является одним из основных в самостоятельной работе и требует систематических усилий и организованности обучающегося на протяжении всего обучения.

Изучение литературы нужно начинать с предварительного общего ознакомления с работой (монография, учебник, учебное пособие и т.п.). Затем следует ознакомиться с оглавлением и структурой работы, что поможет оценить общий замысел автора, избранную им последовательность анализа тех или иных вопросов. Как правило, в каждой научной работе имеются предисловие или введение которые следует изучить в первую очередь. Написанные автором или рецензентом, они, как правило, дают представление о цели, источниках и литературе, использованной автором, его методологических подходах, исследовательских методах и т.д.

Не менее важно ознакомиться с научным аппаратом автора: просмотреть ссылки на источники, примечания, приложения.

Следующий этап - внимательное чтение работы с начала до конца, при большом объеме - по частям или разделам. Читать следует, тщательно обдумывая содержание, не пропуская кажущиеся неинтересными или сложными фрагменты текста, добиваясь понимания прочитываемого материала. Обычно главная мысль обосновывается рядом доказательств, приводящих к определенным выводам, усвоить которые можно только при ознакомлении со всей его аргументацией, методикой и рассуждениями.

При этом нужно обязательно выделять из прочитанного самое важное и существенное.

В случае необходимости, можно оформлять записи изучаемого текста в виде плана, выписок и цитат, тезисов и конспекта.

3. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

3.1. Основная литература

1. Илюшечкин, В. М. Основы использования и проектирования баз данных : учебник для среднего профессионального образования / В. М. Илюшечкин. — испр. и доп. — Москва : Издательство Юрайт, 2023. — 213 с. — (Профессиональное образование). — ISBN 978-5-534-01283-5. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/513827

2. Советов, Б. Я. Базы данных : учебник для среднего профессионального образования / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2023. — 420 с. — (Профессиональное образование). — ISBN 978-5-534-09324-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/514585

3.2. Дополнительная литература

1. Гордеев, С. И. Организация баз данных в 2 ч. Часть 1 : учебник для среднего профессионального образования / С. И. Гордеев, В. Н. Волошина. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2023. — 310 с. — (Профессиональное образование). — ISBN 978-5-534-11626-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/518510

2. Нестеров, С. А. Базы данных : учебник и практикум для среднего профессионального образования / С. А. Нестеров. — Москва : Издательство Юрайт, 2023. — 230 с. — (Профессиональное образование). — ISBN 978-5-534-11629-8. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/518507

3. Стружкин, Н. П. Базы данных: проектирование : учебник для среднего профессионального образования / Н. П. Стружкин, В. В. Годин. — Москва : Издательство Юрайт, 2023. — 477 с. — (Профессиональное образование). — ISBN 978-5-534-11635-9. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/518499

Периодические издания:

1. Прикладная информатика [Электронный ресурс]. – Режим доступа: https://www.iprbookshop.ru/11770.html - Цифровой образовательный ресурс IPR SMART

3.3. Программное обеспечение

Microsoft Windows;
Microsoft Office Professional Plus 2019 или Яндекс 360
Microsoft Visio;
Microsoft Visual Studio (WPF, ADO.NET Entity Framework, Xamarin);
Microsoft SQL Server Express Edition;
Microsoft SQL Server Management Studio;
MySQL.

3.4. Базы данных, информационно-справочные и поисковые системы, Интернет-ресурсы

Базы данных (профессиональные базы данных)

База данных IT специалиста– [Электронный ресурс]– Режим доступа: http://info-comp.ru/

Информационно-справочные системы

Информационно-справочная система для программистов – [Электронный ресурс]– Режим доступа: http://life-prog.ru

Поисковые системы

Поисковая система Google https://www.google.ru
Поисковая система Yandex https://www.yandex.ru
Поисковая система Rambler http://www.rambler.ru
Поисковая система Yahoo https://www.yahoo.com/

Электронные образовательные ресурсы

– Бесплатная электронная библиотека онлайн «Единое окно доступа к образовательным ресурсам» – [Электронный ресурс]– Режим доступа: http://www.window.edu.ru

– Единая коллекция цифровых образовательных ресурсов – [Электронный ресурс]– Режим доступа: http://school-collection.edu.ru/
– Корпорация Майкрософт в сфере образования – [Электронный ресурс]– Режим доступа: https://www.microsoft.com/ru-ru/education/default.aspx
– Научная электронная библиотека – [Электронный ресурс]– Режим доступа: http://www.elibrary.ru/
– Научная электронная библиотека «Киберленинка» – [Электронный ресурс]– Режим доступа: http://cyberleninka.ru/
– Национальный открытый университет Интуит – интернет университет информационных технологий – [Электронный ресурс]– Режим доступа: http://www.intuit.ru/
– Образовательный портал GeekBrains с доступом к онлайн-обучению (Learning Management System) и к курсам по четырем направлениям: программирование, дизайн, управление и маркетинг – [Электронный ресурс]– Режим доступа: https://university.geekbrains.ru/
– Образовательная платформа ЮРАЙТ – [Электронный ресурс]– Режим доступа: https://urait.ru/
– Электронно-библиотечная система ZNANIUM – [Электронный ресурс]– Режим доступа: https://znanium.com

4. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ РАБОТАМ

Практическая работа №1. Анализ предметной области и проектирование разработки. Проектирование Use case диаграммы, определение функциональных возможностей

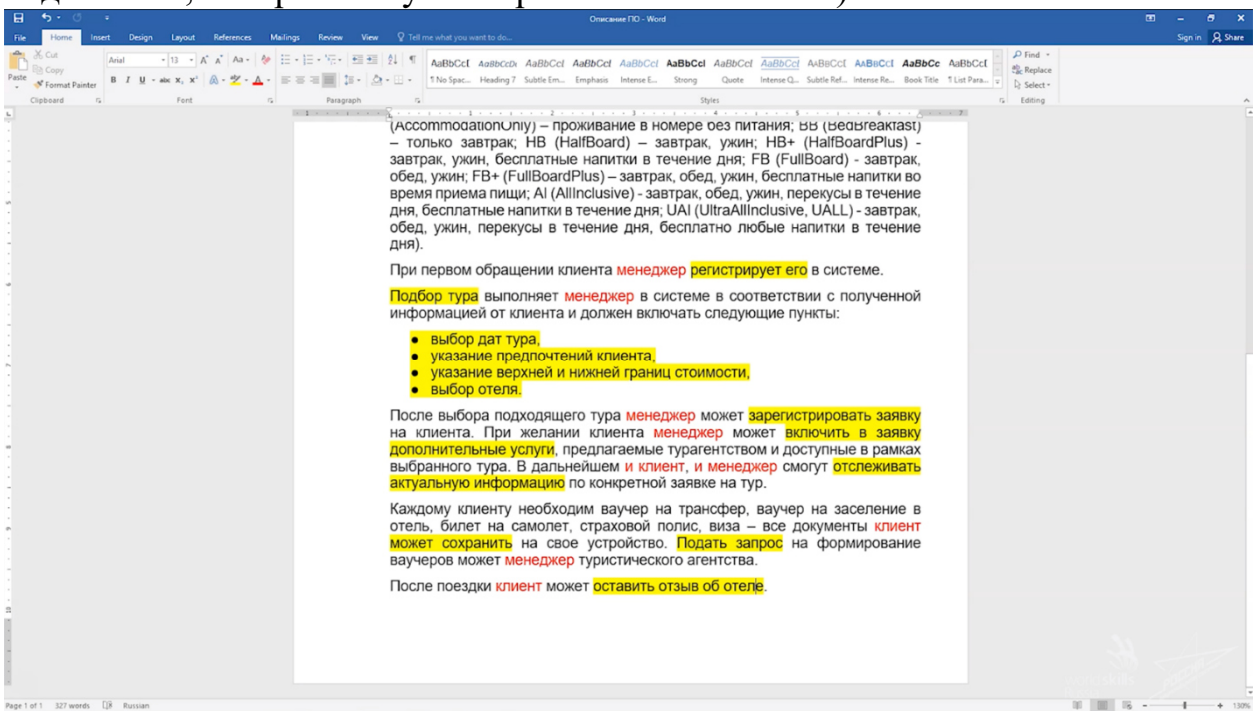
Краткие теоретические сведения

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)


(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, страховой полис, виза – все документы клиент может сохранить на свое устройство. Подать запрос на формирование ваучеров может менеджер туристического агентства.

После поездки клиент может оставить отзыв об отеле.

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

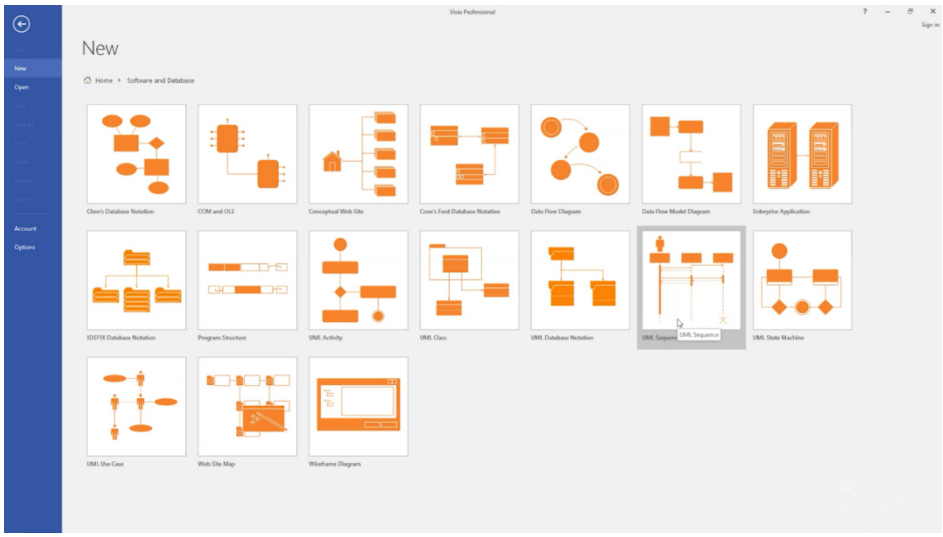
Ревью возможностей MS Visio для создания диаграмм

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

- текстовые описания
- диаграммы
- 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:

- **Диаграмма вариантов использования (*use case diagram*)**
- Диаграмма классов (*class diagram*)
- Диаграмма состояний (*statechart diagram*)
- Диаграмма последовательности (*sequence diagram*)



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

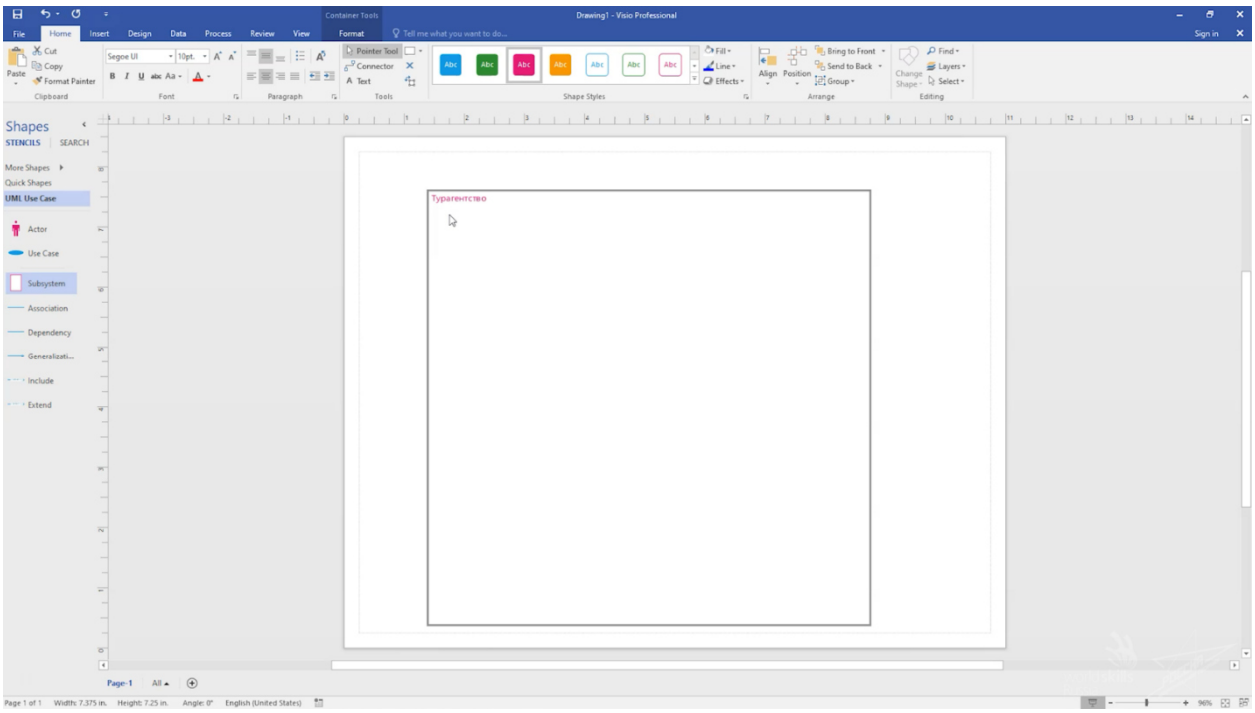
Создание диаграммы для турагенства

1. Определение рамок системы согласно заданию

Для этого используем элемент subsystem, там будут располагаться прецеденты (функционал, реализуемый системой)

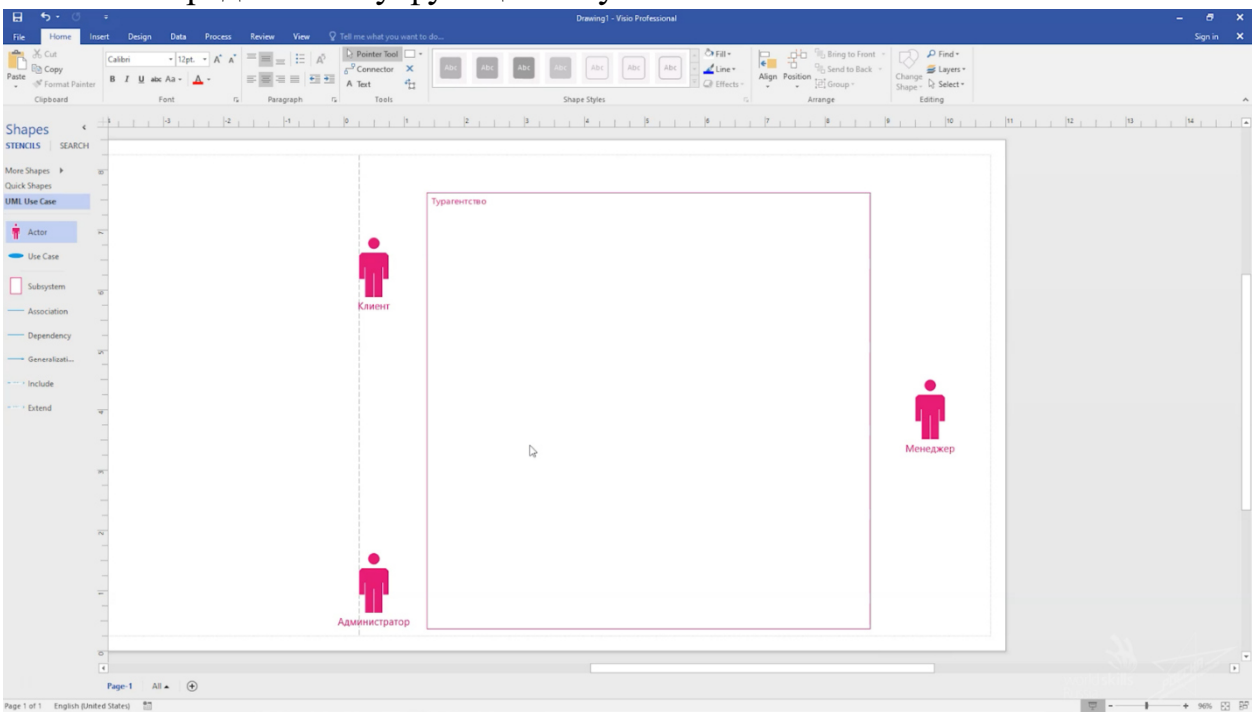
Остановимся на диаграмме вариантов использования. Она достаточно проста, это позволяет использовать ее для согласования технического задания с заказчиком

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

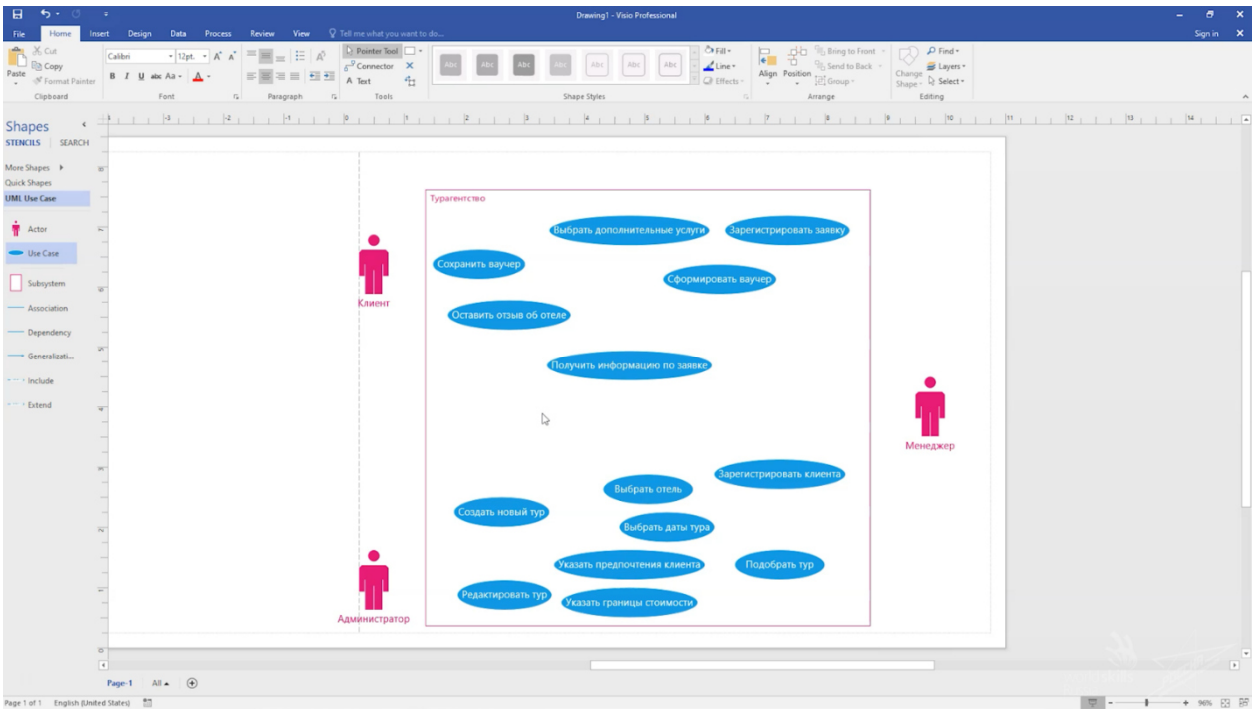


2. Определение основных групп пользователей (ролей) и размещение на диаграмме

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:

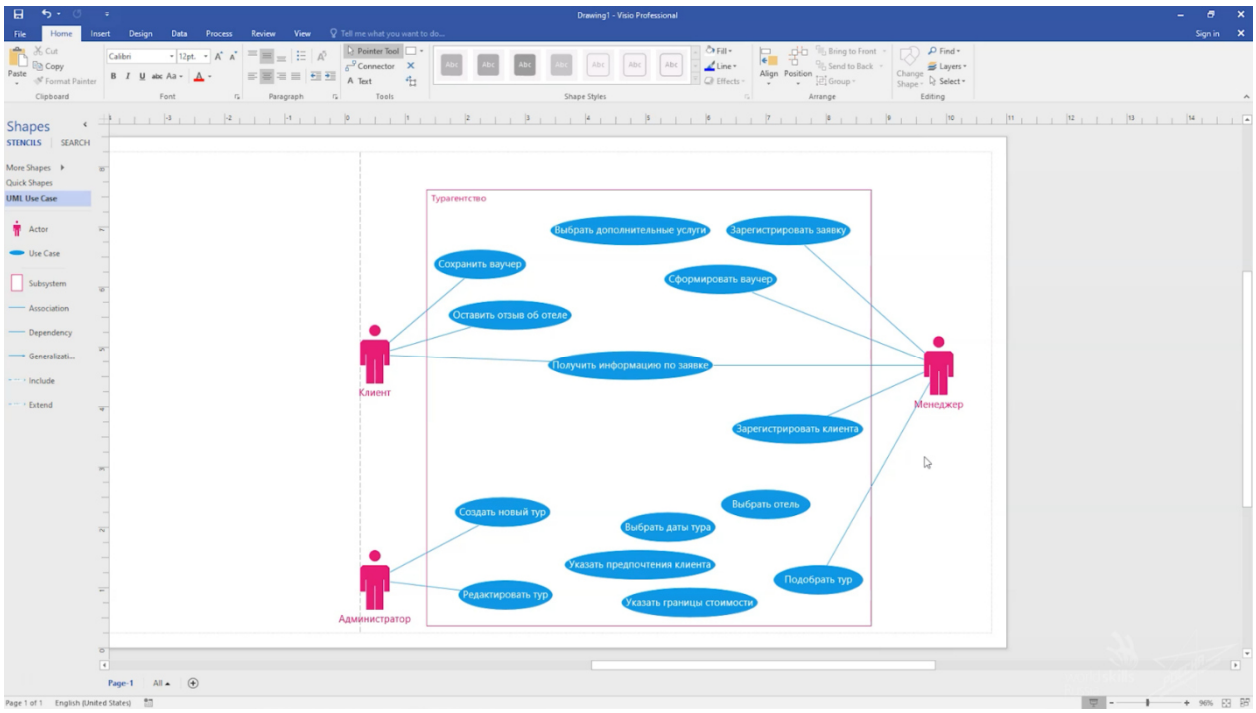
- сохранить ваучер на устройство
- оставить отзыв об отеле

Горячие клавиши (Hot Keys) для переключения инструментов

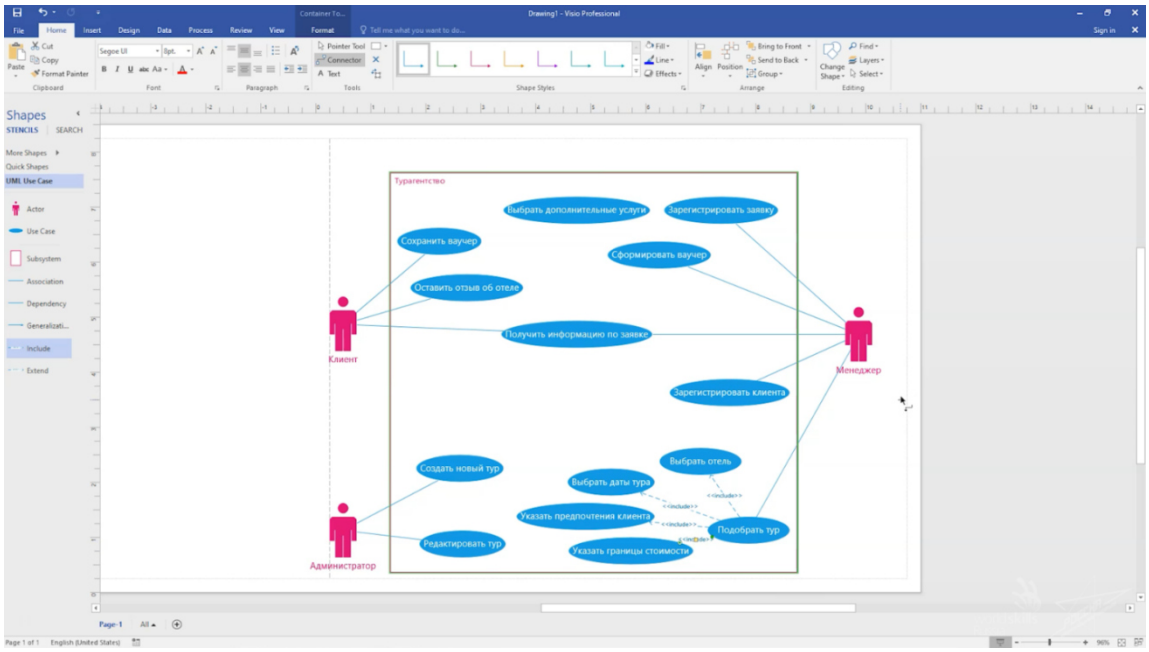
- Ctrl + 1 — выделить элемент
- Ctrl + 2 — добавить комментарий
- Ctrl + 3 — добавить связь между актером и прецедентом

Разграничение прецедентов между актерами и размещение отношений

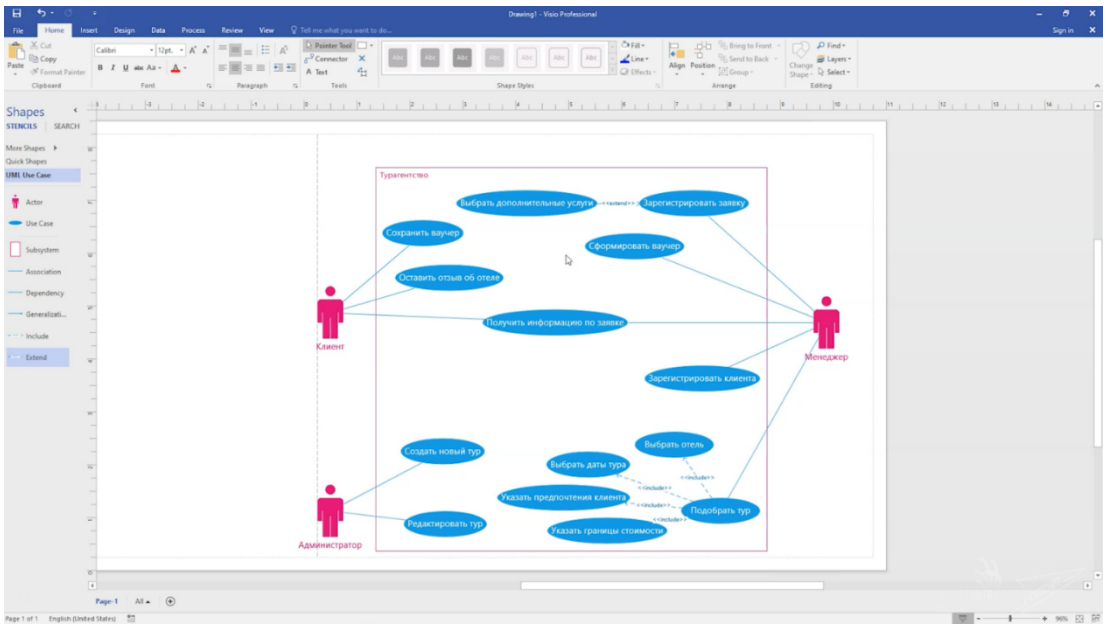
Отношение ассоциации — отражает возможность использования актером прецедента



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



Формат сохранения диаграммы

На этом проектирование диаграммы завершено, и мы можем перейти к ее сохранению. Созданная диаграмма по умолчанию хранится в формате .vsdx, но для гарантированного запуска файла на других устройствах рекомендуется сохранять диаграмму еще и в .pdf формате

Практическая работа №2. Проектирование диаграммы UML (Entity–relationship)

Краткие теоретические сведения

В основе ER-диаграмм лежит принцип «рисунок нагляднее текста»

ER-диаграмма графически представляет сущности (entities) 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:

Добавление сущностей

1. Добавление связей и их настройка
2. Добавление атрибутов

Важно

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

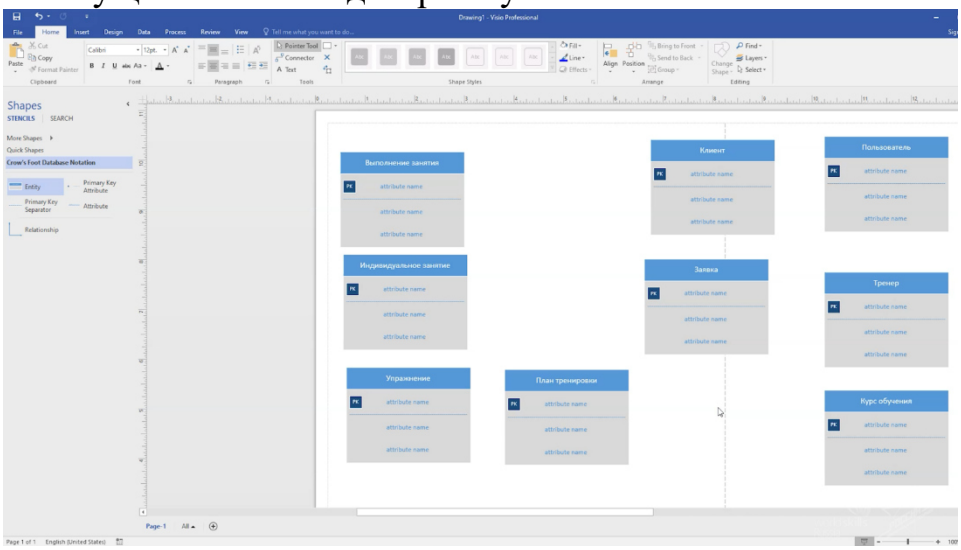
Добавление сущностей

1. 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»

2. Добавляем сущности на ER-диаграмму

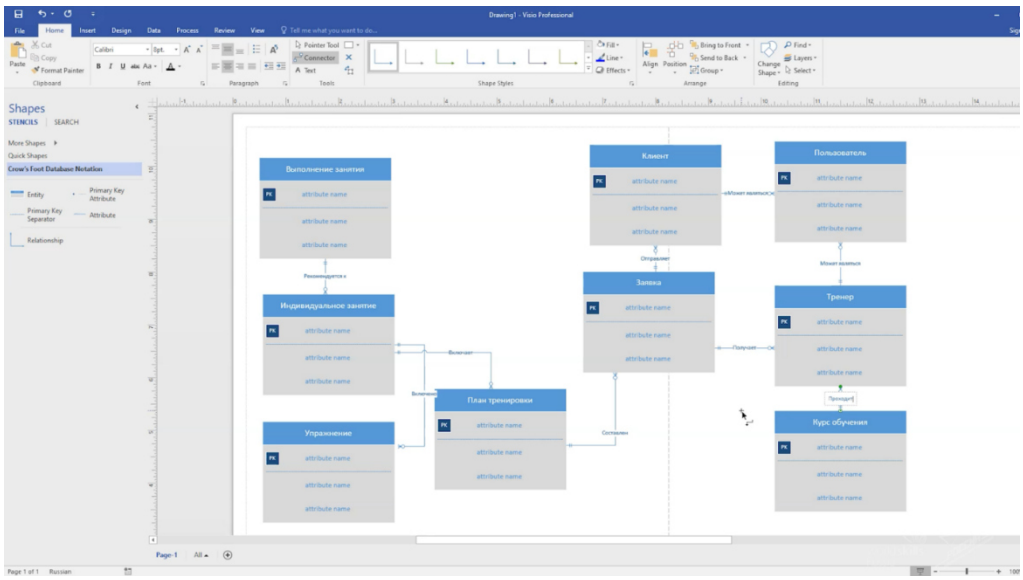


Важно

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

Добавление связей и их настройка

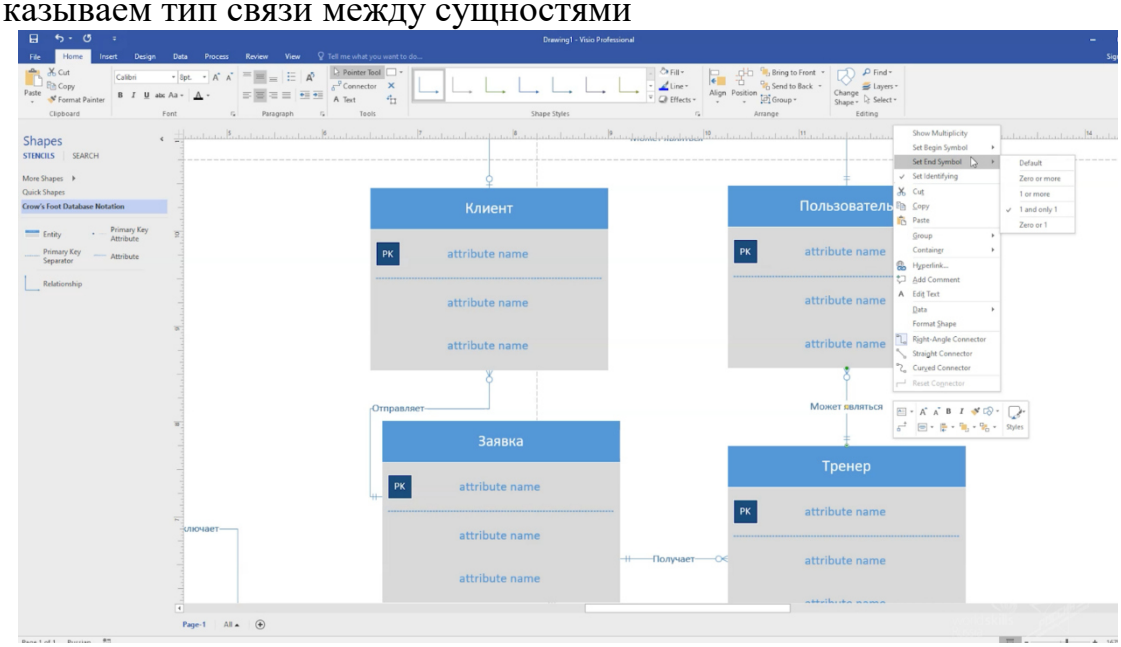
1. Изображаем связи на ER-диаграмме



Важно

Связь (relationship) — ассоциация между сущностями. Для облегчения понимания диаграммы следует добавлять названия связей. Пример связи — «тренер получает заявку»

2. Указываем тип связи между сущностями

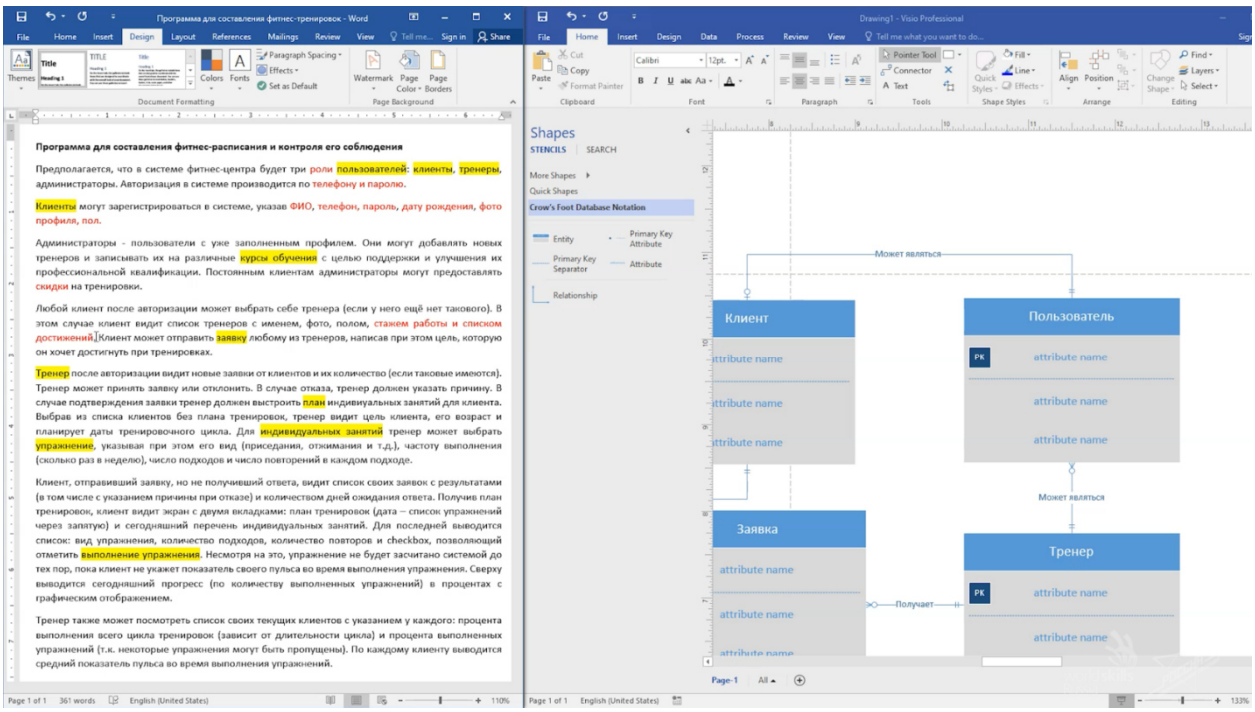


При определении типа следует учитывать модальность связи: «может» или «должен». Модальность «может» означает, что экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может быть и не связан ни с одним экземпляром другой сущности. Модальность «должен» подразумевает 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)
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

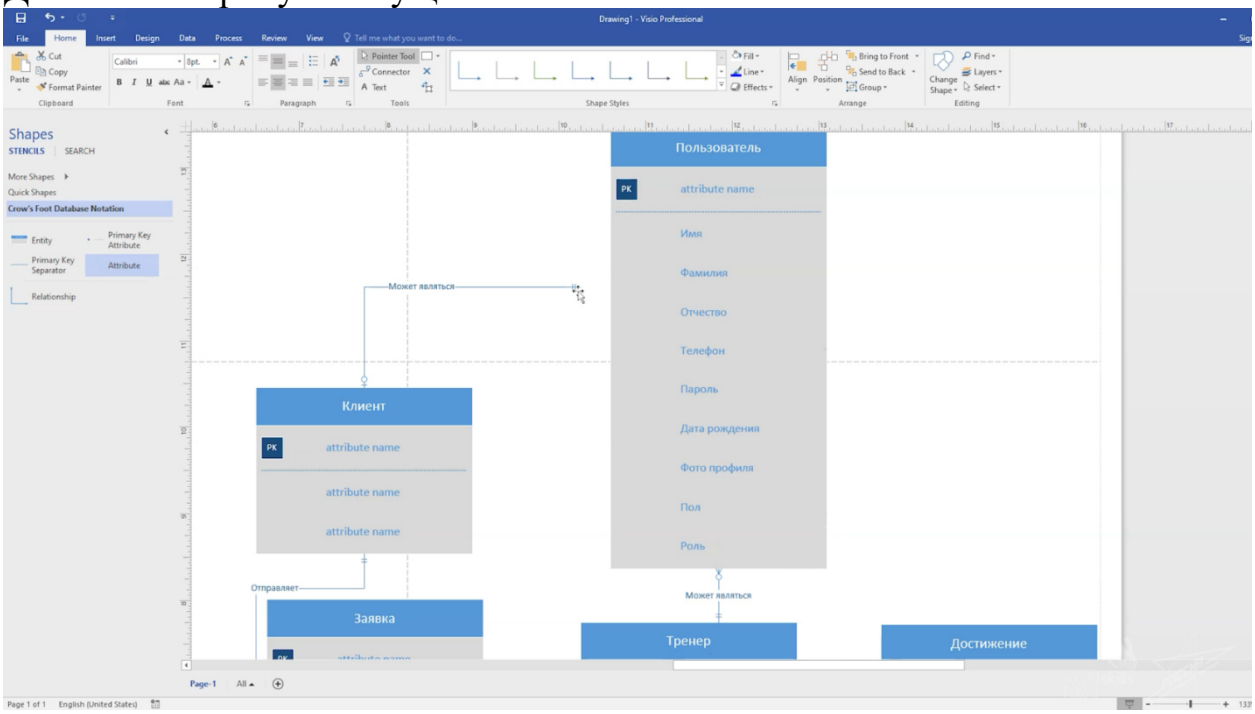
1. Выделяем атрибуты в описании заказчика



Важно

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

2. Добавляем атрибуты к сущностям

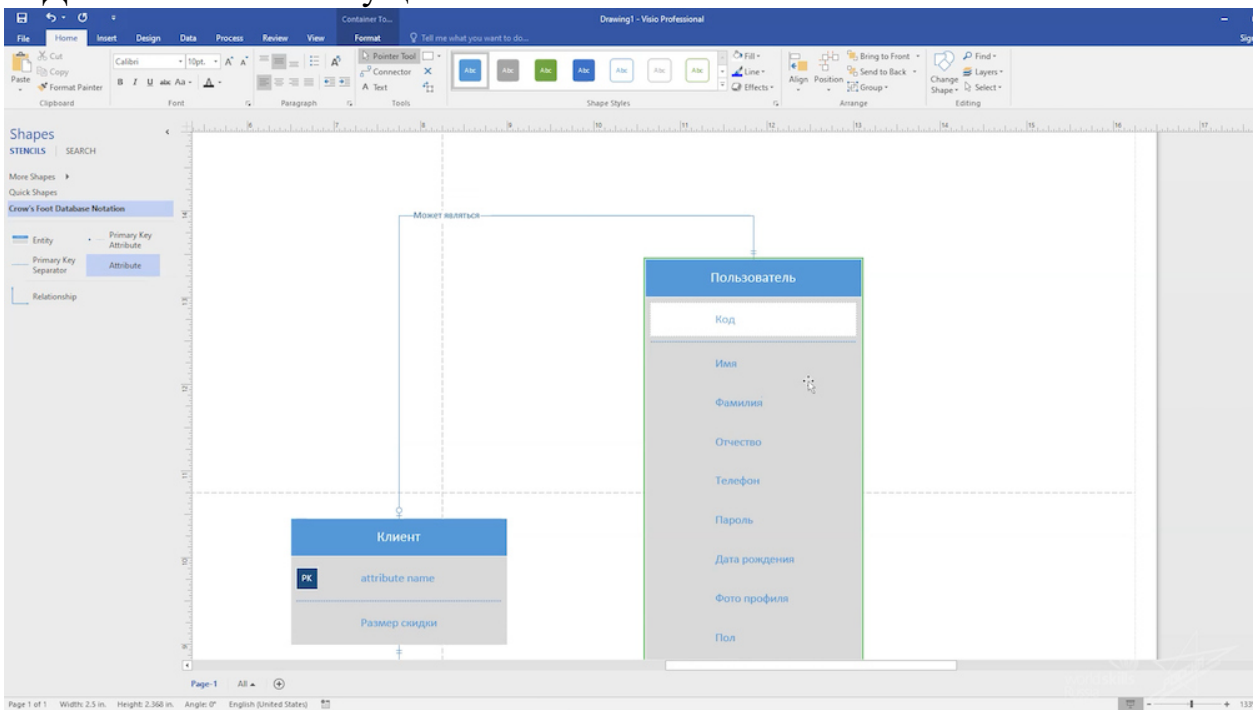


Важно

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

3. Добавляем ключ к сущностям



Важно

Ключ — это один или несколько атрибутов, уникально определяющих сущность. В данном примере в качестве ключа используется атрибут «код»
Интерактивное задание: https://nationalteam.worldskills.ru/skills/proektirovanie-er-diagrammy/

Практическая работа № 3. Проектирование диаграммы классов UML (Class Diagram)

Краткие теоретические сведения

Унифицированный язык моделирования (UML)

UML — унифицированный язык графических нотаций, в основе которого лежит единая метамодель

UML используется для описания и проектирования программных систем, особенно построенных с использованием объектно-ориентированных (ОО) технологий

UML как средство проектирования нацелен на полноту. Используя UML, дизайнер может строить детальные модели для программиста, который далее выполняет кодирование без глубокого погружения в детали

Диаграмма классов

Диаграмма классов описывает типы объектов системы и различного рода статические отношения, которые существуют между ними. На диаграммах классов отображаются также свойства классов, операции

классов и ограничения, которые накладываются на связи между объектами

Важно

Классы могут представлять сущности предметной области (на этапе анализа) или элементы программной системы (на этапе проектирования и реализации). В данном занятии рассматривается первый случай

Пример построения диаграммы классов

В данном занятии демонстрируется построение диаграммы классов для программной системы фитнес-центра. Основные шаги построения диаграммы классов:

Добавление классов

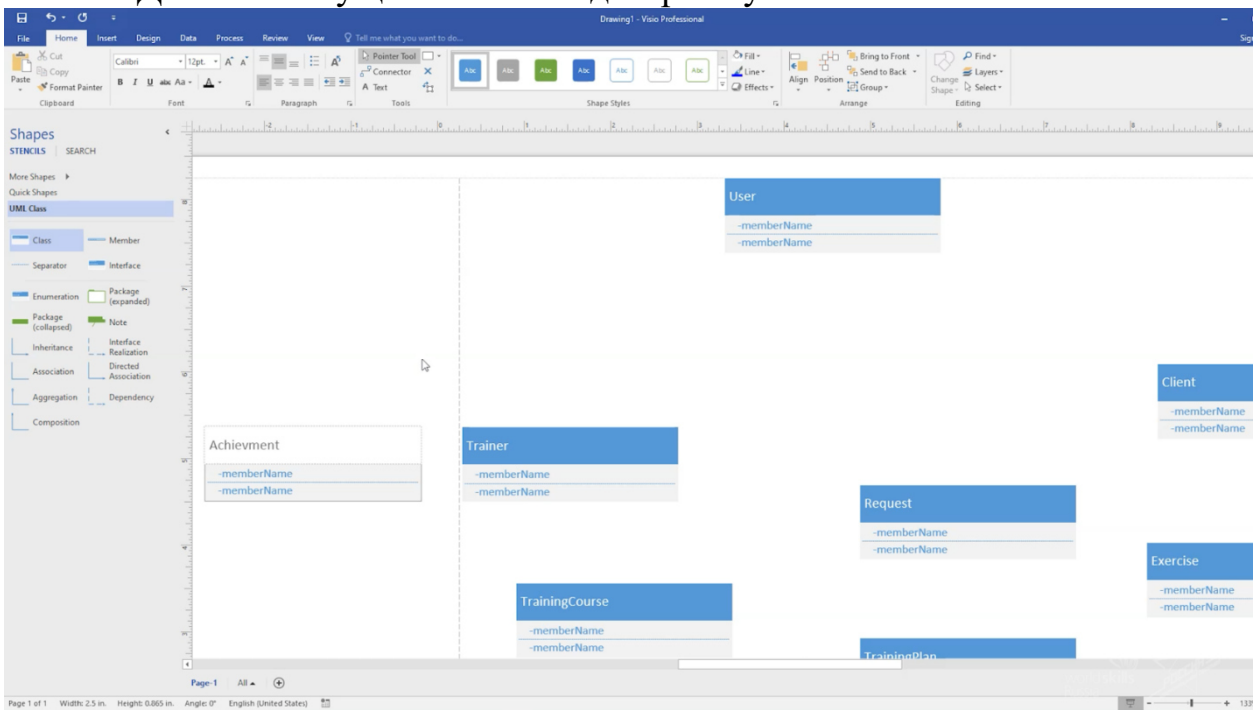
1. Добавление связей и их настройка
2. Добавление атрибутов и операций

Важно

Диаграмма классов составляется в Microsoft Visio на основе описания заказчика. Используется тип диаграммы UML Class из раздела Software and Database

Добавление классов

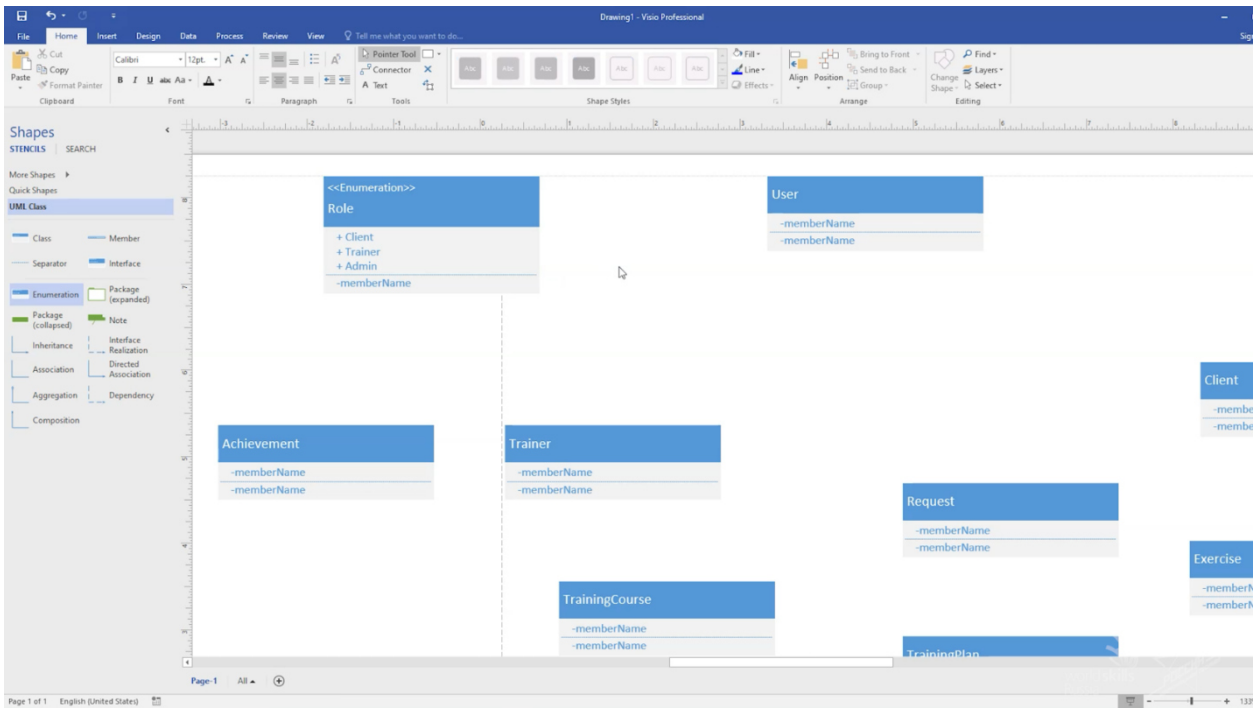
1. Добавляем сущности на ER-диаграмму



Важно

На диаграмме класс изображается в виде прямоугольника, разделенного на три части: имя класса (на английском языке), его атрибуты и его операции. В качестве классов выступают сущности, использованные при построении диаграммы сущность-связь

2. Добавляем другие объекты

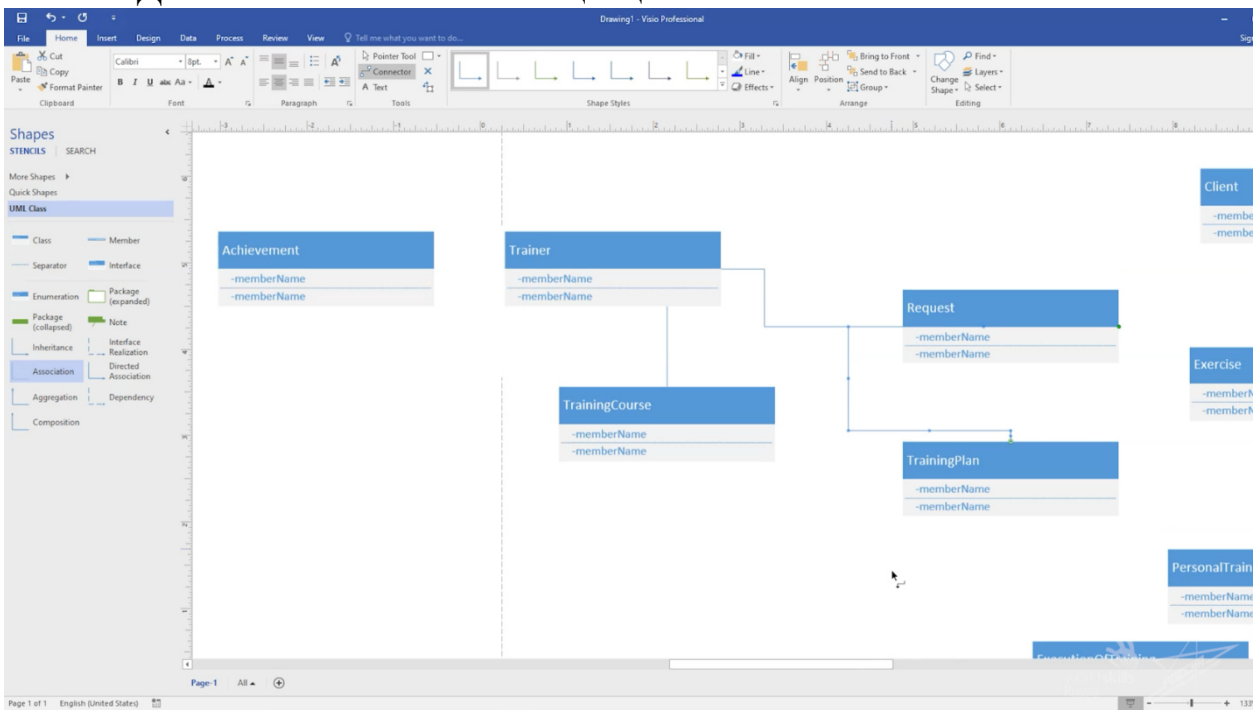


Важно

В данном примере добавляется объект «role» типа «перечисление» (англ. enumeration), представляющий собой набор логически связанных и заранее присвоенных значений (Client, Trainer, Admin)

Добавление связей и их настройка

1. Добавляем связи типа ассоциация

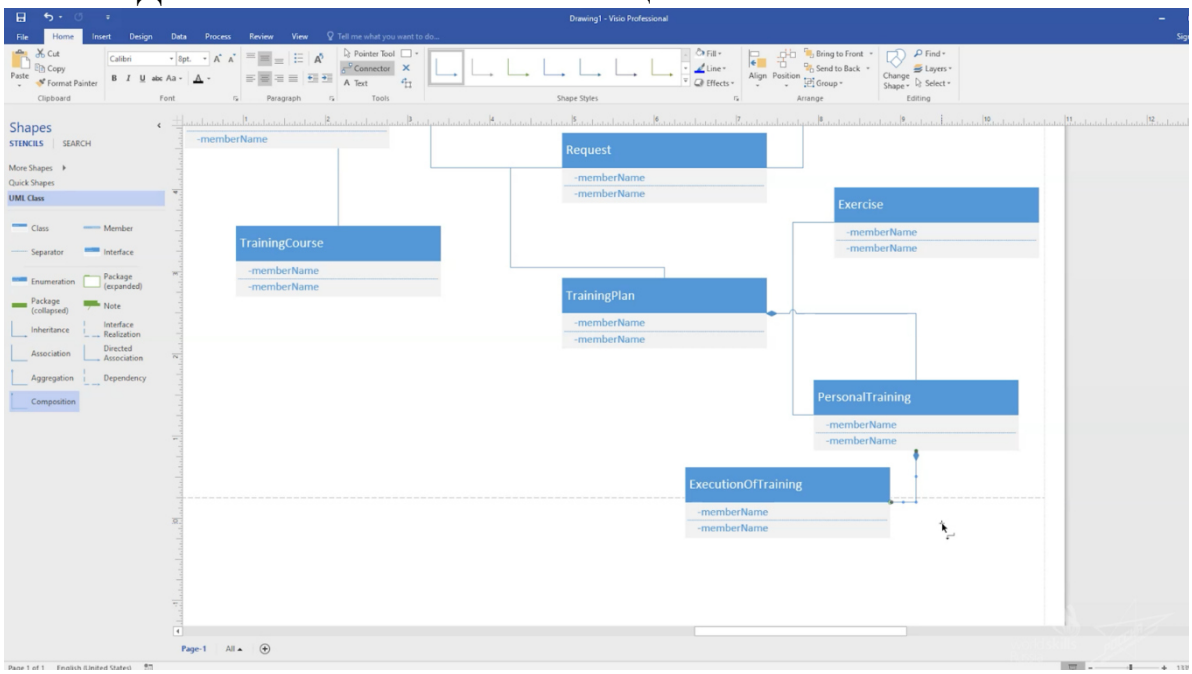


Важно

Ассоциация является одним из двух основных типов связи на диаграмме классов, показывающим, что можно перемещаться между объектами двух связанных классов. Пример ассоциации — «Trainer-

TrainingCourse»

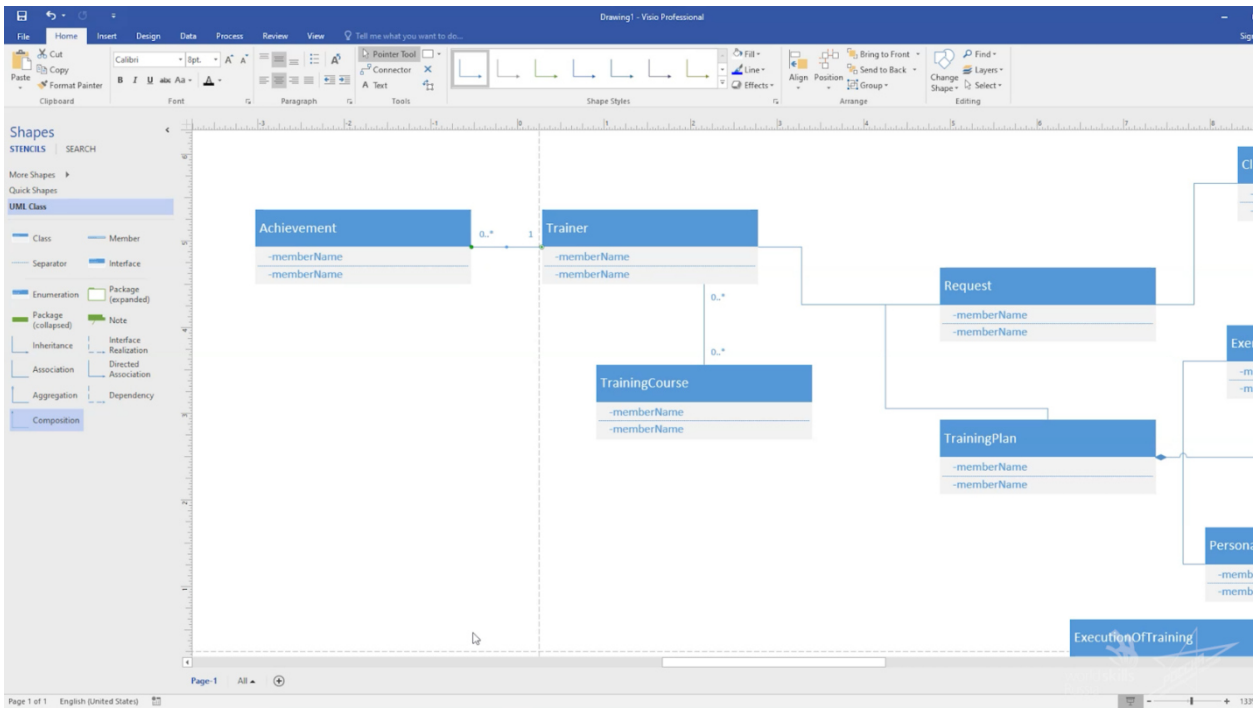
2. Добавляем связи типа композиция



Важно

Композиция — частный случай ассоциации, представляющий собой отношение типа «часть-целое». Композиция имеет четко выраженные отношения владения, а также характеризуется совпадением времени жизни части и целого. Композиция имеет жесткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то все его содержимое будет также уничтожено. Пример композиции — связь «TrainingPlan-PersonalTraining»

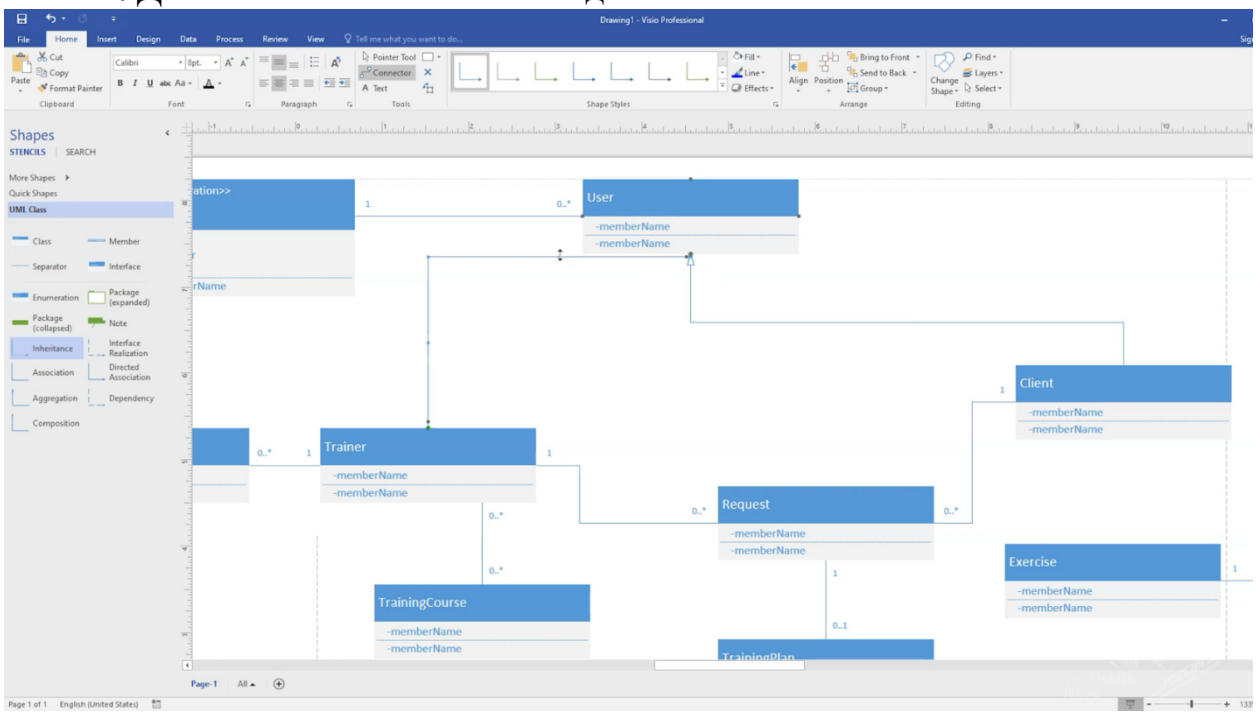
3. Проставляем кратность связей



Важно

Кратность связи или множественность ассоциации — диапазон целых чисел, указывающий возможное количество связанных объектов. Кратность задается путем указания минимального и максимального количества объектов, разделенных двумя точками. Варианты кратности связи: 1 (единица), 0.1 (ноль или один), 0.* (любое значение) и 1.* (один или несколько)

4. Добавляем связи типа наследование

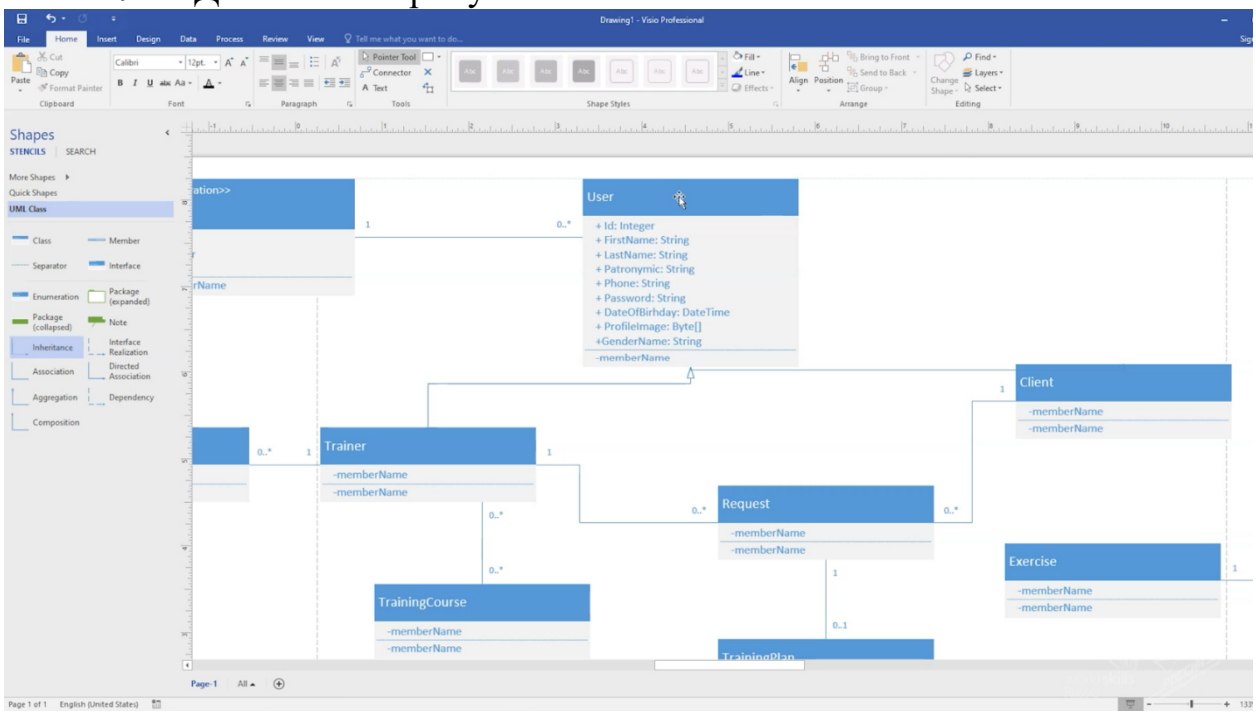


Важно

Наследование (inheritance) — отношение типа «общее-частное», при котором один класс обладает поведением и структурой ряда других классов. Пример наследования — связь «Trainer-User»

Добавление атрибутов и операций

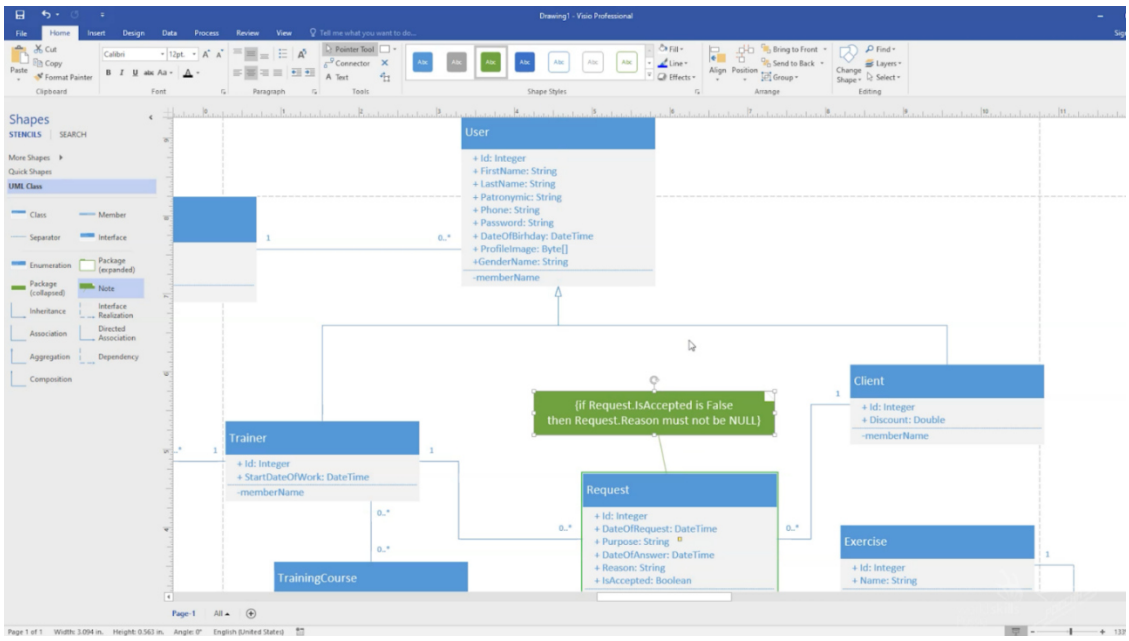
1. Добавляем атрибуты



Важно

Атрибут описывает свойство класса в виде строки текста, имеющей в общем случае следующую структуру: «visibility name: type = defaultValue», где visibility определяет область видимости (public, private или protected), name есть имя атрибута, type определяет тип атрибута, a defaultValue — его значение по умолчанию (для числовых или булевых переменных)

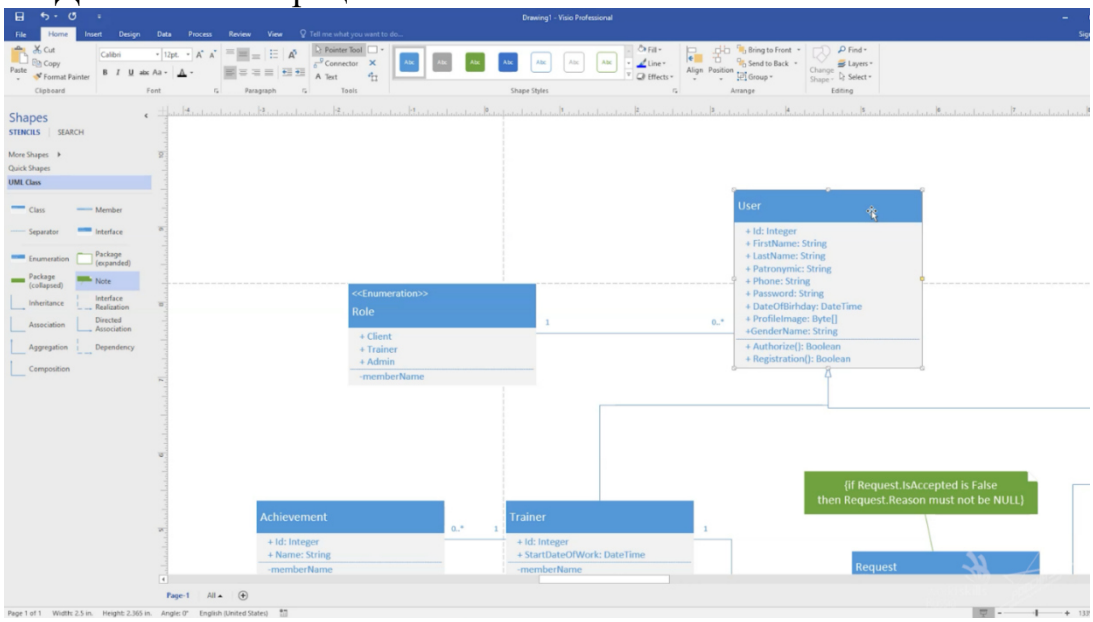
2. Добавляем заметки



Важно

В заметках можно указывать ограничения. В данном примере добавляется следующее ограничение: если заявка не принята, то должна быть указана причина

3. Добавляем операции



Важно

Операции — действия, реализуемые некоторым классом, т. е. по сути методы класса. Общая форма записи операции: visibility name (parameter-list): return-type-expression, т. е. область видимости, имя операции, список параметров, тип возвращаемого значения

Интерактивное задание:

https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-klassov-uml-class-diagram/

Практическая работа №4. Проектирование диаграммы деятельности UML (Activity Diagram)

Краткие теоретические сведения

Диаграмма деятельности — технология, позволяющая описывать логику процедур, бизнес-процессы и потоки работ

Основным отличием диаграмм деятельности от блок-схем является активная поддержка параллельных процессов, что объясняет применение диаграммы деятельности для моделирования потоков работ

Пример построения диаграммы деятельности

В данном занятии демонстрируется построение диаграммы выполнения фитнес-упражнений в течение дня. Основные шаги построения диаграммы деятельности:

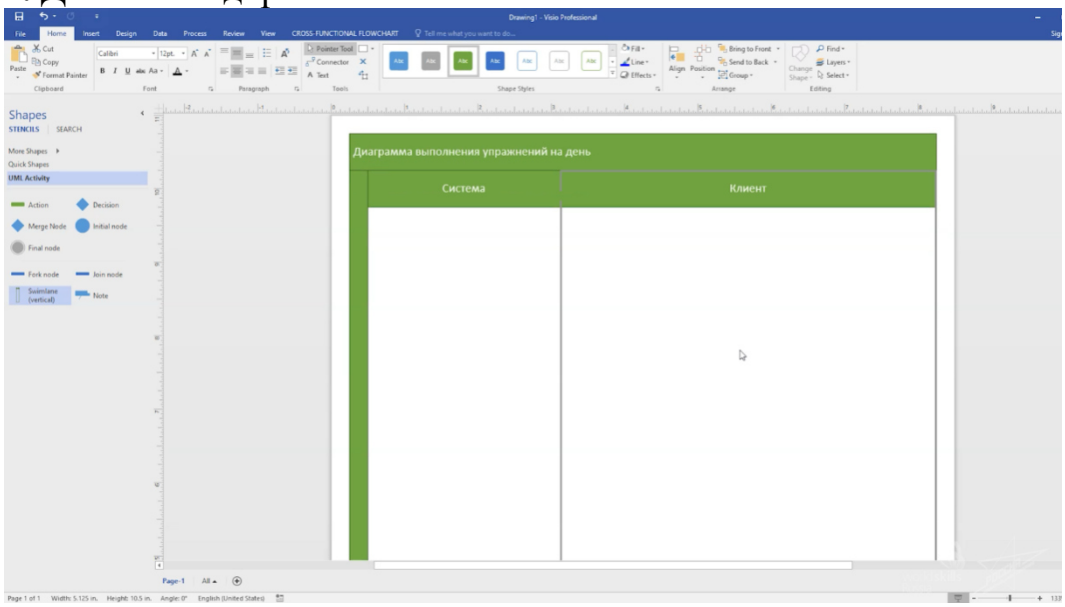
1. Добавление основных элементов
2. Работа с операциями

Важно

Для построения диаграммы деятельности используется шаблон UML Activity из раздела Software and Database программы Visio

Добавление основных элементов

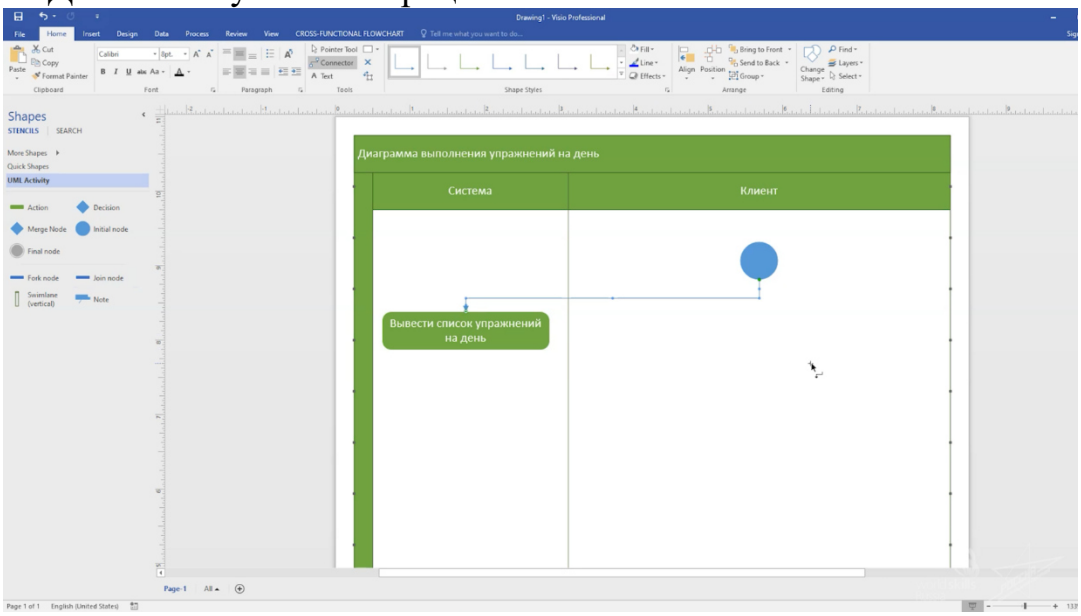
1. Добавляем дорожки



Важно

Дорожка (англ. swimlane) — часть области диаграммы деятельности для отображения деятельностей, за которые отвечает конкретный объект (например, пользователь или организационное подразделение). Диаграмма-пример содержит две дорожки: «Система» и «Клиент»

2. Добавляем узлы и операции

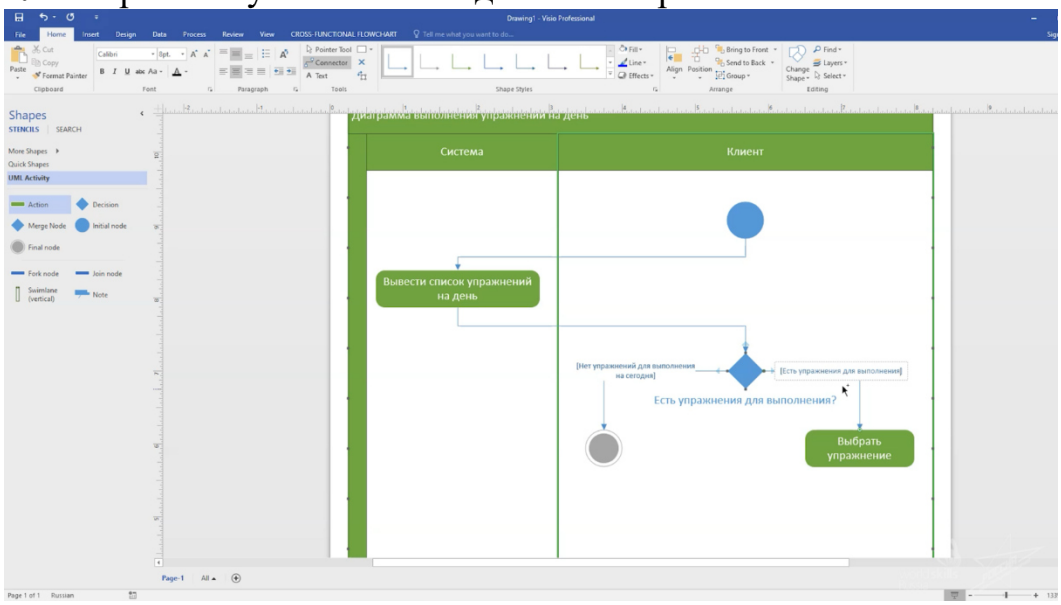


Важно

Диаграмма деятельности должна начинаться с **начального** узла (англ. activity initial node), в котором начинаются поток или потоки при вызове деятельности извне, и завершаться **конечным** узлом (англ. activity final node), который останавливает все потоки данной деятельности. В примере начальный узел изображается голубым кругом, а конечный узел — серым. Операции (англ. operations) являются ключевыми элементами диаграммы деятельности и отображаются зелеными овалами

Работа с операциями

1. Отображаем условное поведение типа решение

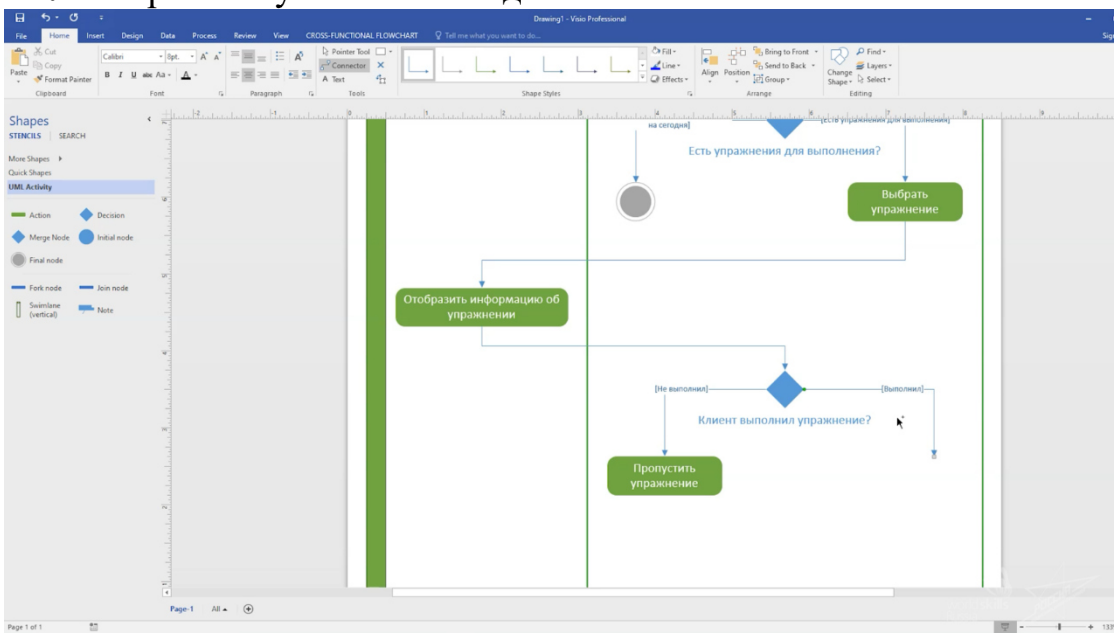


Важно

Решение (англ. decision) имеет один входящий и несколько защищенных выходящих потоков. Защитой является условное выражение, помещенное в скобки. Так как при достижении решения выбирается только один из выходных потоков, защиты должны быть взаимоисключающими.

Пример защиты — «[Есть упражнения для выполнения]»

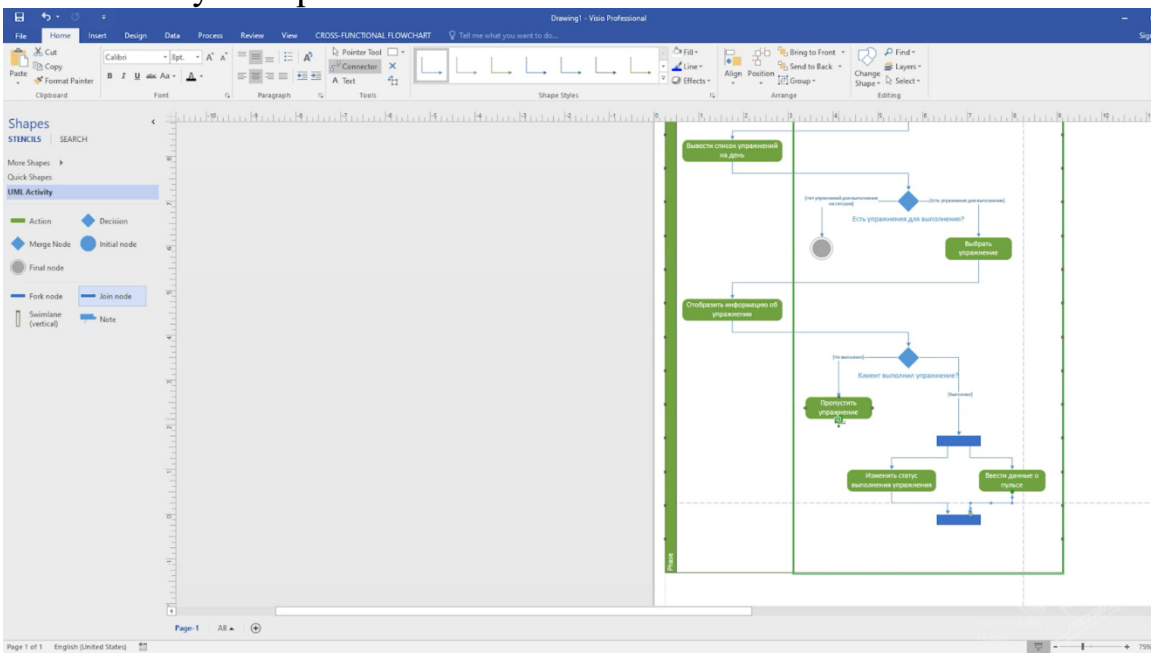
2. Отображаем условное поведение типа слияние



Важно

Слияние (англ. merge) означает завершение условного поведения, которое было начато решением

3. Реализуем параллельные потоки



Важно

Для реализации параллельных потоков используются точки разделения и точки слияния. Из точки разделения (англ. fork node) выходят два и более потока, выполняющихся паралелльно. Точка слияния (англ. join node) синхронизирует потоки, т. к. каждый из них ждет достижения этой точки

остальными потоками, после чего продолжается последовательное исполнение

Интерактивное задание:
https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-deyatelnosti-uml-activity-diagram/

Практическая работа №5. Проектирование диаграммы последовательности UML (Sequence Diagram)

Краткие теоретические сведения

Диаграмма последовательности позволяет изобразить поведение нескольких объектов в рамках одного прецедента

Диаграмма последовательности удобна для представления взаимодействия объектов, но не для точного определения их поведения

Диаграмма показывает экземпляры объектов и сообщения, которыми обмениваются экземпляры в рамках одного прецедента

Пример построения диаграммы последовательности

В данном занятии демонстрируется построение диаграммы последовательности ответа фитнес-тренера на заявку клиента. Основные шаги построения диаграммы последовательности:

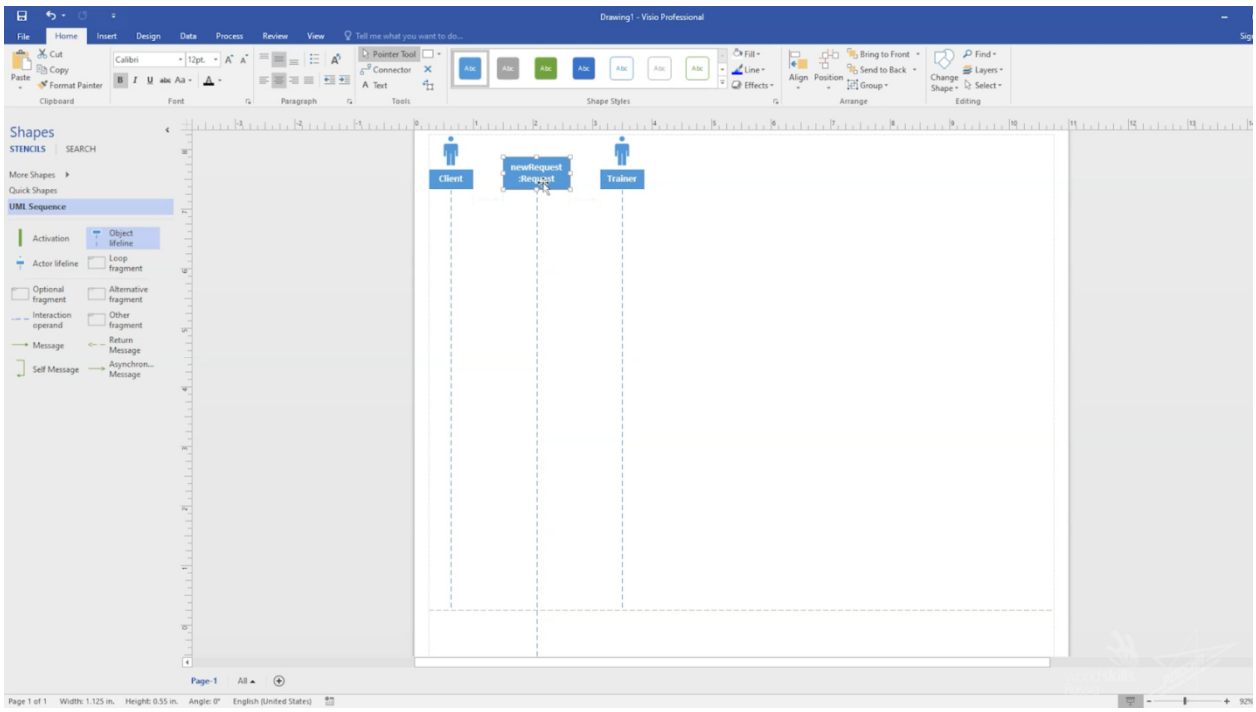
1. добавление основных элементов
2. работа с сообщениями

Важно

Для построения диаграммы последовательности используется шаблон UML Sequence из раздела Software and Database программы Visio

Добавление основных элементов

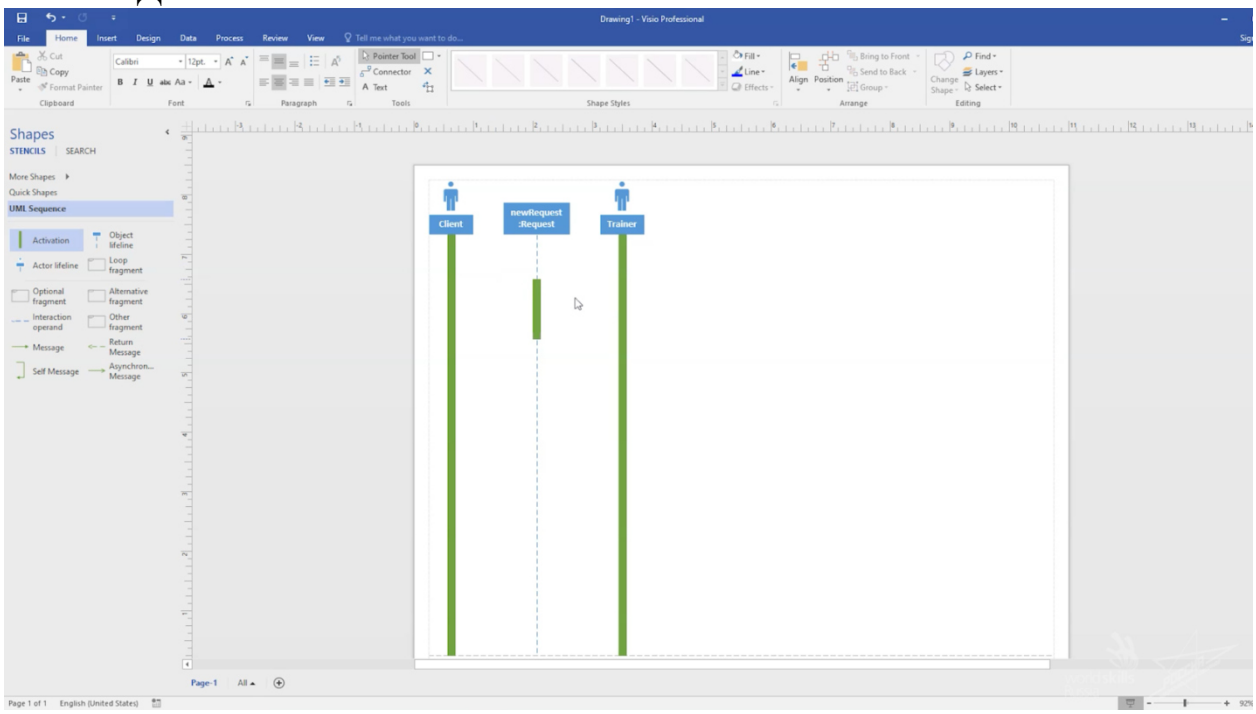
1. Добавляем объекты



Важно

Объекты обычно подписываются в формате «объект:класс» и изображаются как в виде обычных прямоугольников, так и с использованием дополнительных обозначений. В представленном примере объектами являются запрос, обозначенный прямоугольником, а также тренер и клиент, обозначенные элементом «Актер»

2. Добавляем полосы активности на линии жизни



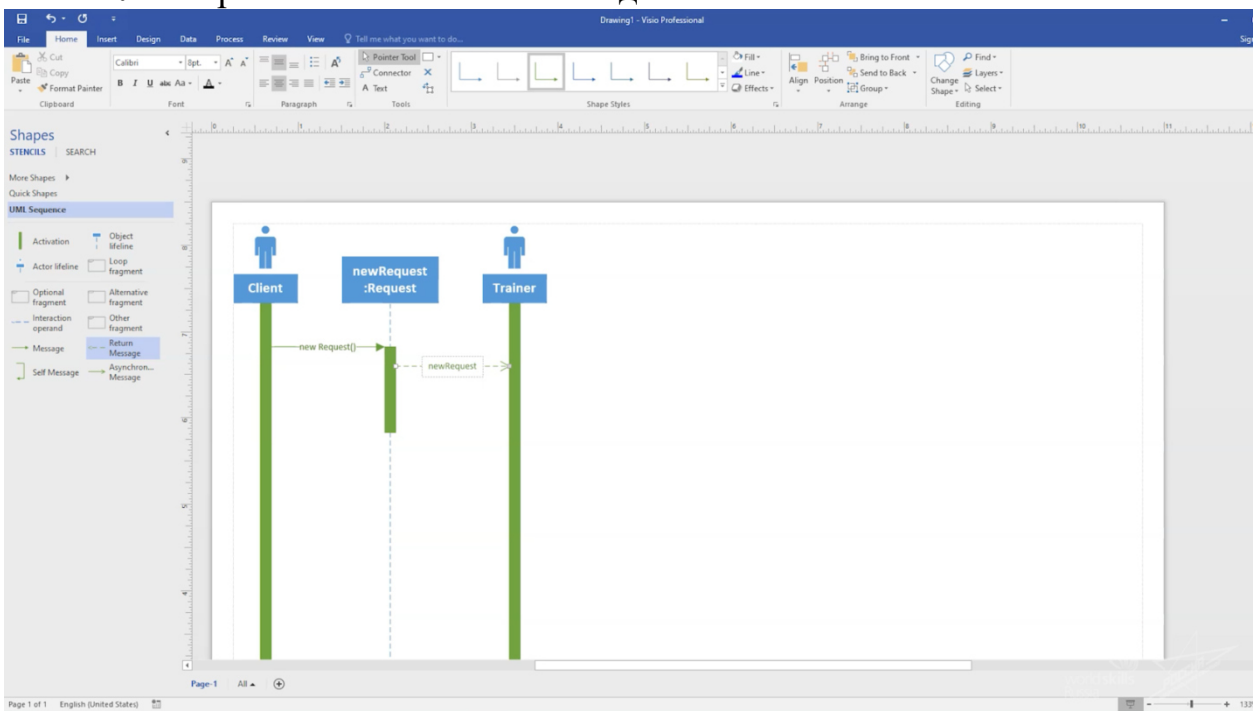
Важно

Линия жизни (англ. lifeline) идет вертикально вниз от каждого объекта и упорядочивает сообщения на странице таким образом, чтобы они читались сверху вниз. Каждая линия жизни имеет полосу активности (зеленые

вертикальные прямоугольники), показывающую интервал активности участника при взаимодействии

Работа с сообщениями

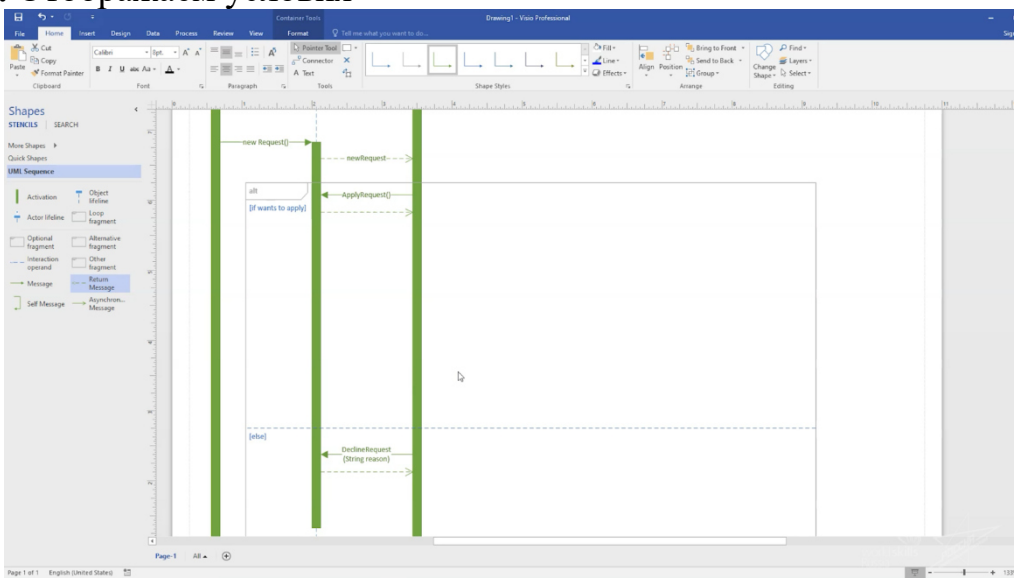
1. Отображаем основные взаимодействия



Важно

Сообщения показывают взаимодействие между объектами в виде горизонтальной стрелки, концы которой лежат на линиях жизни. Направление стрелки указывает на адресата, а положение на линии жизни упорядочивает сообщения по времени. При создании нового объекта и применении конструктора можно не указывать имя сообщения, указав ключевое слово «new»

2. Отображаем условия

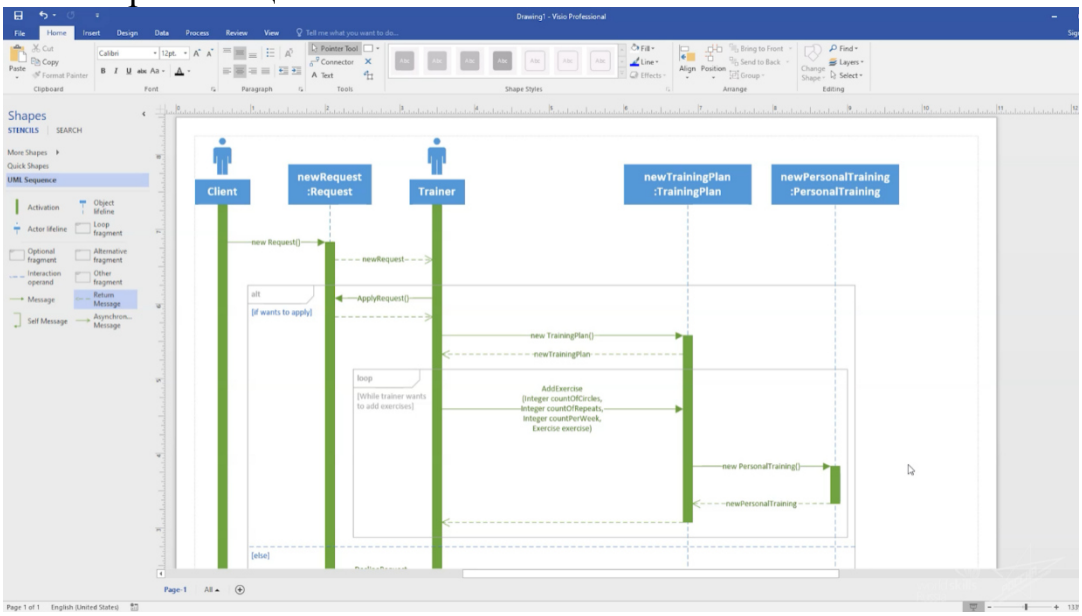


Важно

Условия, как и циклы, изображаются с помощью фреймов

взаимодействий (англ. interaction frames), позволяющих разметить диаграмму взаимодействия. Каждый фрейм представляет собой разделенную на несколько фрагментов область диаграммы, причем каждый фрейм имеет оператор, а каждый фрагмент может иметь защиту. В данном примере для условной логики используется оператор alt и будет выполнено условие, защита которого имеет истинное значение (т.е. либо принятие заявки «ApplyRequest ()», либо отказ от заявки с указанием причины «DeclineRequest (String reason)"

3. Отображаем циклы



Важно

Для отображения цикла применяется оператор loop с единственным фрагментом, причем тело итерации помещается в защиту. В данном случае для добавления тренером упражнений в план занятий используется метод «AddExercise». Данный метод создает объект «newPersonalTraining», который далее возвращается тренеру

Интерактивное задание:
https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-posledovatelnosti-uml-sequence-diagram/

Практическая работа №6. Проектирование диаграммы состояний UML (Statechart Diagram)

Краткие теоретические сведения

Диаграмма состояний позволяет описывать поведение системы

В объектно-ориентированном подходе разрабатывается диаграмма состояний единственного класса, демонстрирующая поведение одного объекта в течение его жизни

Состояние на диаграмме является более абстрактным понятием, чем состояние объекта (последнее есть комбинация всех данных из полей объекта)

Диаграмма позволяет проектировать различные способы реакции на события

Пример построения диаграммы состояний

В данном занятии демонстрируется построение диаграммы состояний заявки клиента. Основные шаги построения диаграммы состояний:

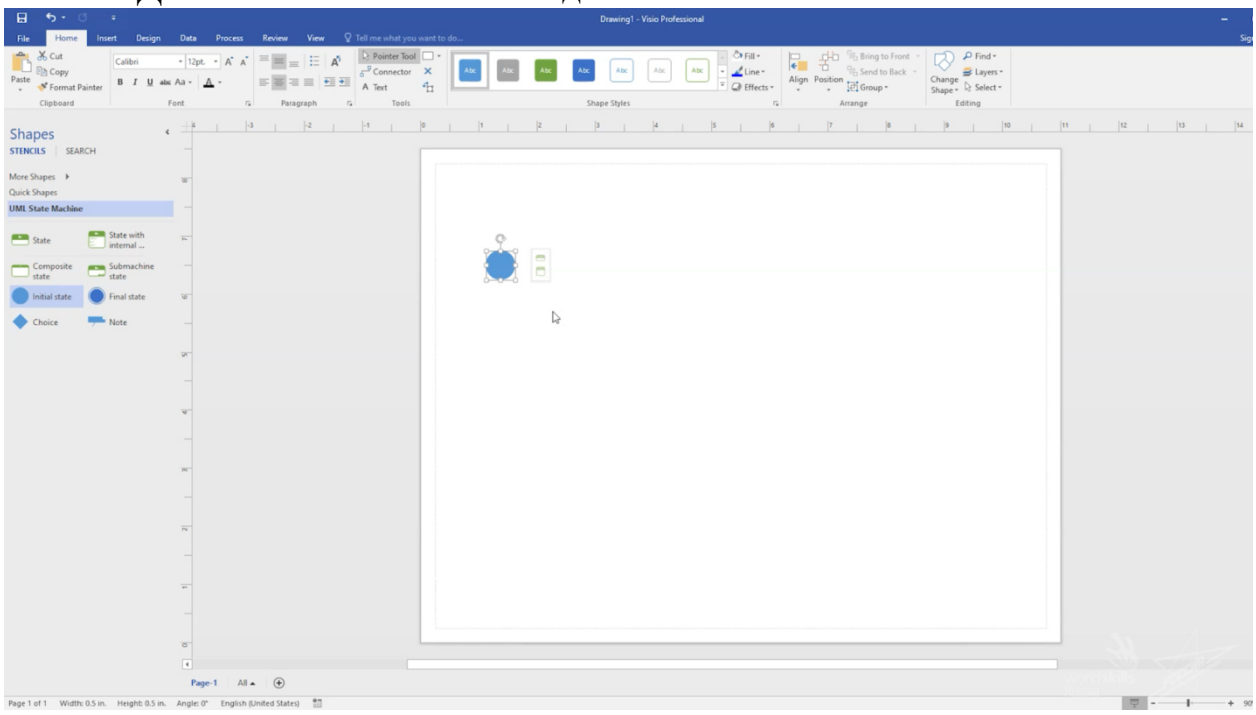
1. добавление состояний
2. указание переходов
3. добавление внутренних активностей
4. указание подсостояний и суперсостояний

Важно

Для построения диаграммы последовательности используется шаблон UML State Machine из раздела Software and Database программы Visio

Добавление состояний

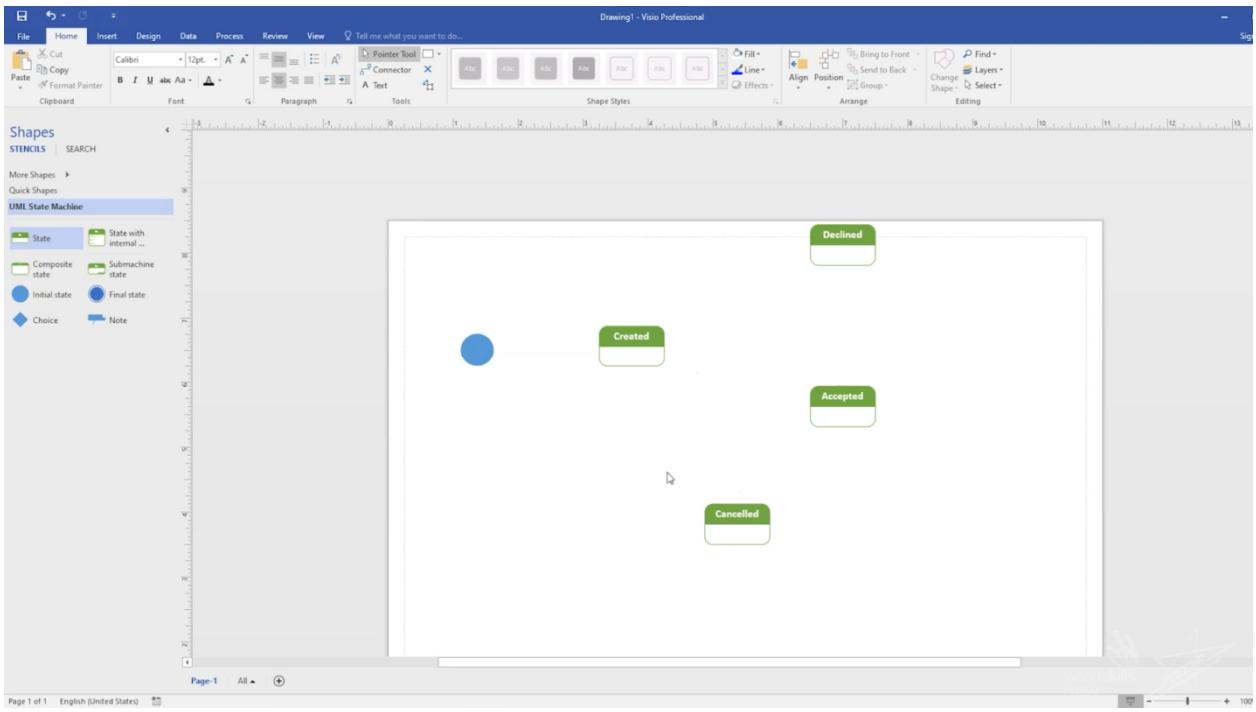
1. Добавляем начальное псевдосостояние



Важно

Начальное псевдосостояние (initial pseudostate) не является состоянием, но имеет стрелку, указывающую на начальное состояние

2. Добавляем возможные состояния

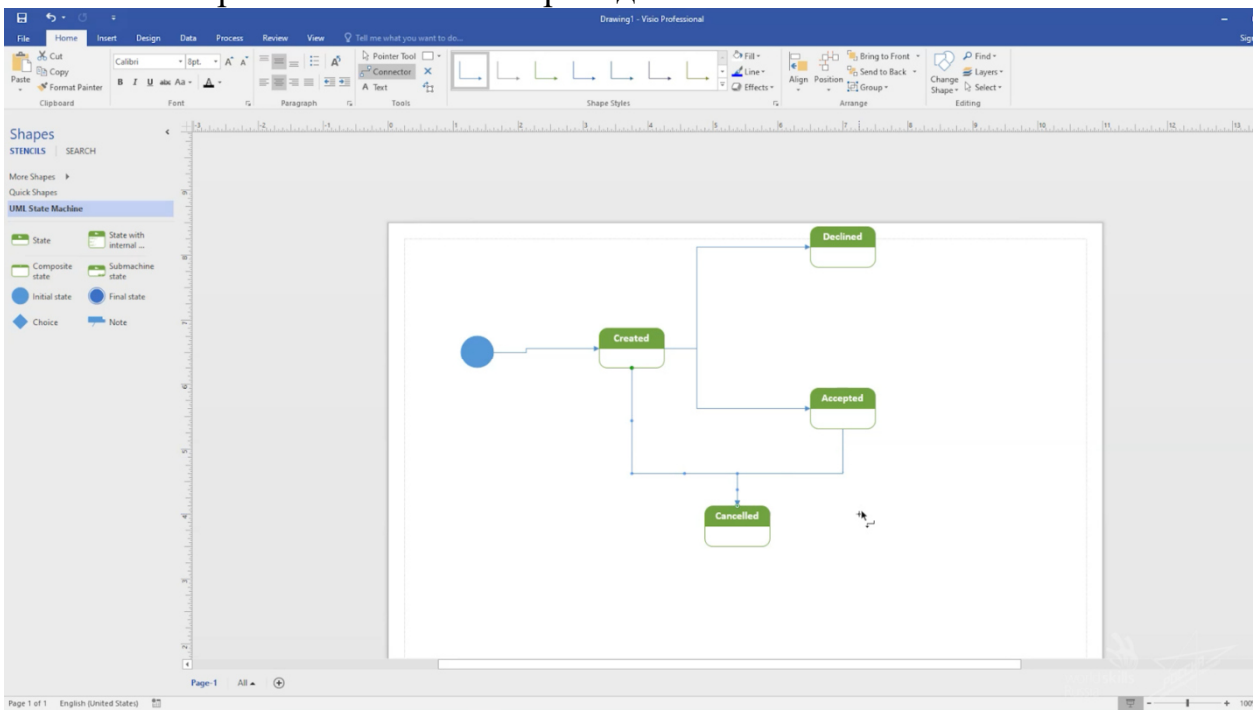


Важно

На диаграмме отображаются состояния, в которых объект может находиться продолжительное время. Состояние может быть прервано вследствие наступления определенного события. Пример состояния заявки: «created»

Указание переходов

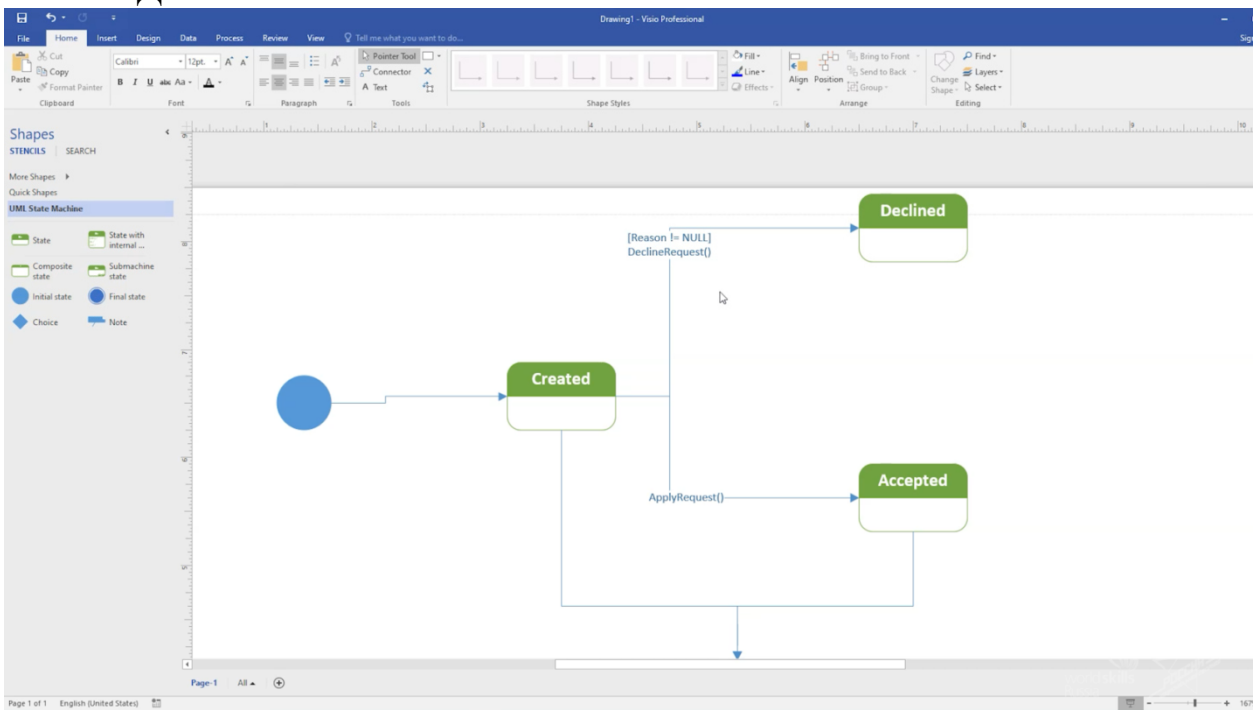
1. Отображаем основные переходы



Важно

Переход (transition) означает перемещение из одного состояния в другое и изображается в виде линий, связывающих состояния

2. Добавляем метки



Каждый переход имеет метку, состоящую из следующих необязательных частей:

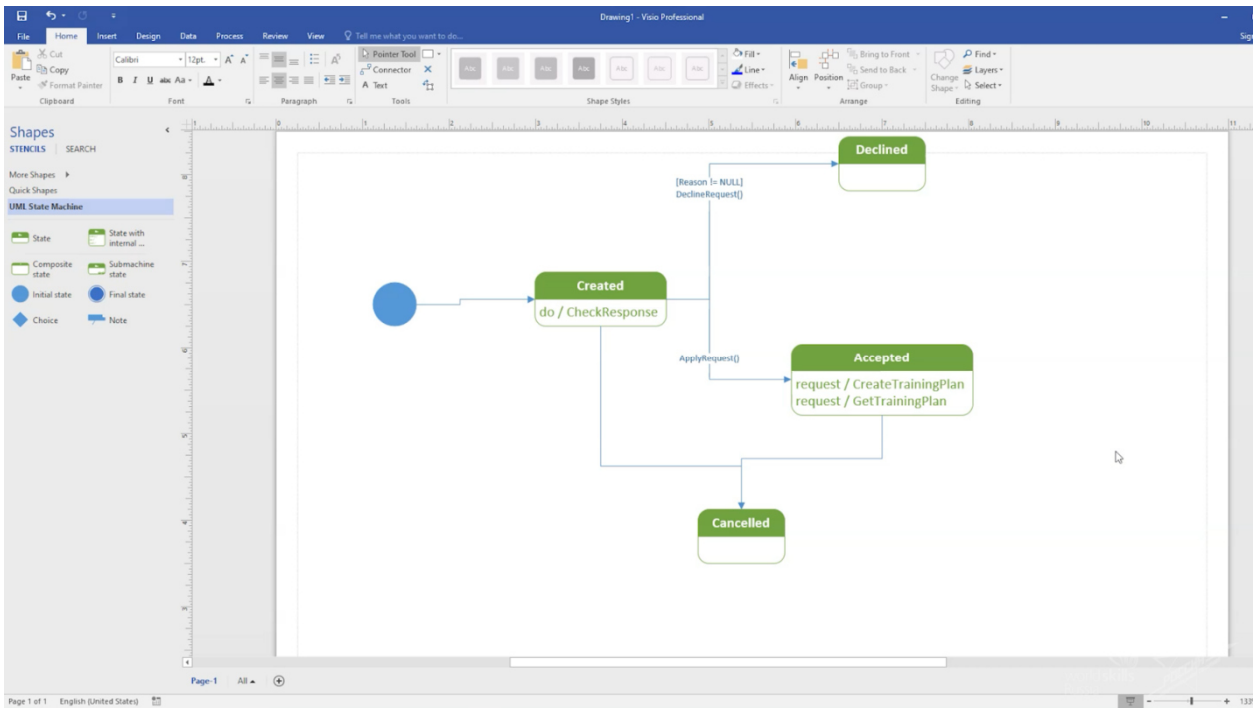
Триггер-идентификатор — единственное событие, способное вызвать изменение состояния. Пропуск этой части означает, что переход происходит немедленно

Защита — логическое условие, выполнение которого обязательно для осуществления перехода. Пропуск защиты означает, что в ответ на инициирующее событие переход всегда осуществляется

Активность — поведение системы во время перехода. Пропуск активности означает, что в процессе перехода ничего не происходит

На представленной диаграмме методы «ApplyRequest()» и «DeclineRequest()» служат примерами событий, вызывающих переходы между состояниями. Примером защиты является выражение «[Reason != null]», означающее, что для отказа заявки всегда следует указывать причину

Добавление внутренних активностей

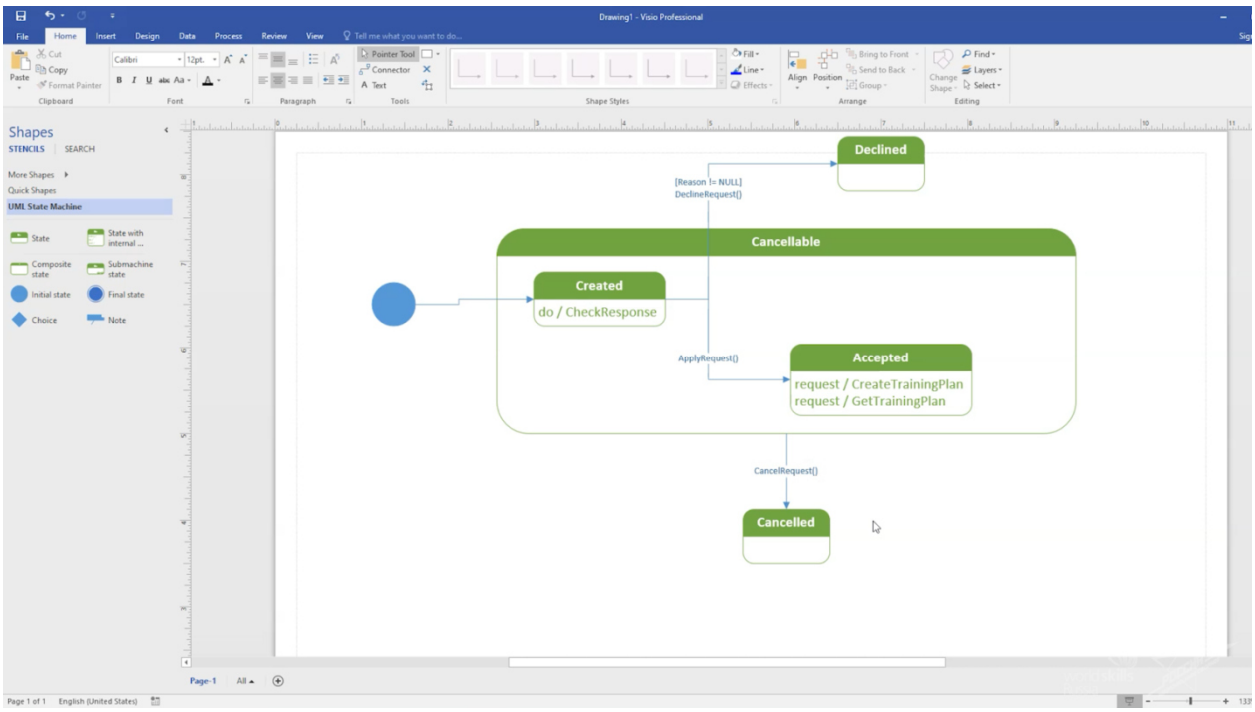


Внутренние активности (internal activities) используются для описания действий объекта, совершаемых без перехода. Список основных действий включает следующие значения:

- **входное действие (entry)** — действие, которое выполняется в момент входа в данное состояние
- **выходное действие (exit)** — действие, которое выполняется в момент выхода из данного состояния
- **выполняющая деятельность (do)** — действие, которое выполняется в течение всего времени нахождения объекта в данном состоянии. Разница между обычными и выполняющими деятельностями / активностями состоит в том, что первые происходят мгновенно и не могут быть прерваны обычными событиями

В данном примере с помощью внутренней активности «do/checkResponse» заявка может проверять, пришел ли ответ от тренера

Работа с общими переходами и внутренними активностями



В случае наличия у нескольких состояний общих переходов и внутренних активностей, само состояние можно превратить в подсостояние (substates), а их общее поведение перенести в суперсостояние (superstate). В данном примере заявка может быть отклонена как из состояния «Created», так и из состояния «Accepted». Эти состояния могут быть объединены в суперсостояние «Cancelable»

Интерактивное задание:
https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-sostoyaniy-uml-statechart-diagram/

Практическая работа №7. Создание баз данных My/MS-SQL

Краткие теоретические сведения

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

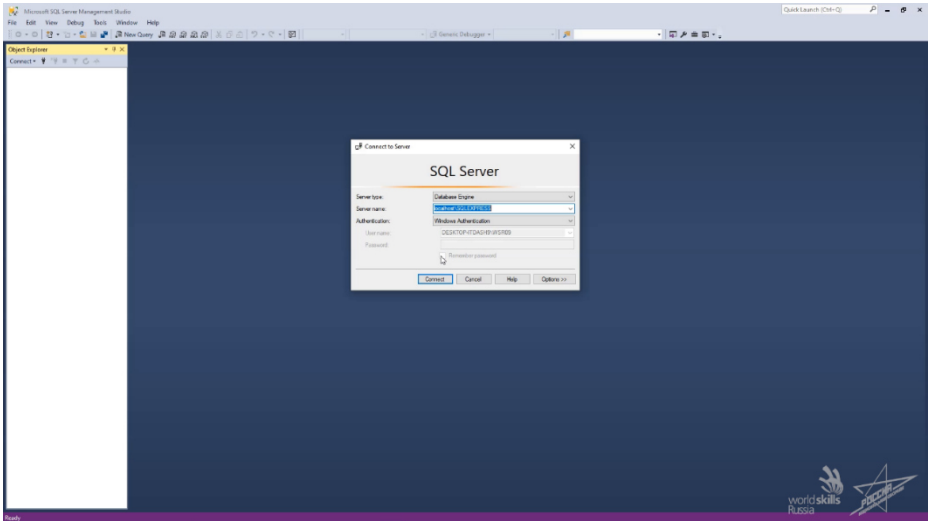
MS SQL Management Studio

ПО для управления базами данных. Основные элементы интерфейса. Создание новой базы данных

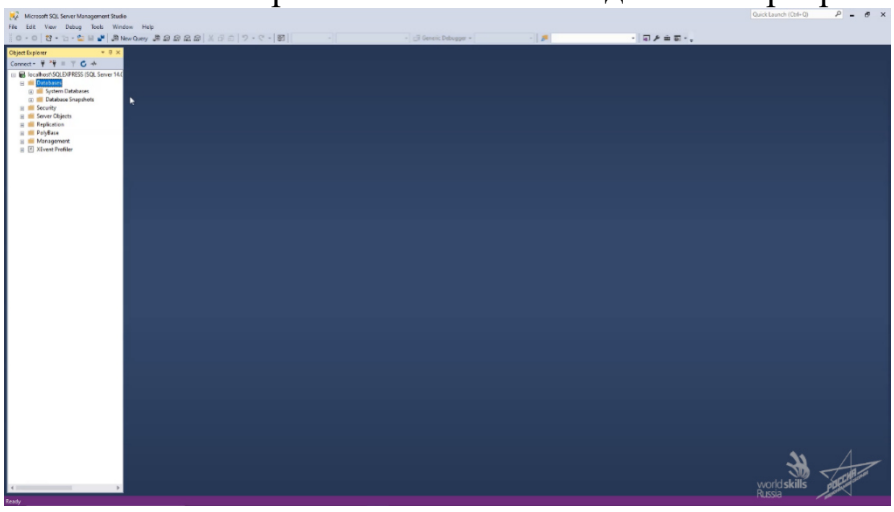
1. Запускаем MS SQL Management Studio

А)Подключаемся к серверу:

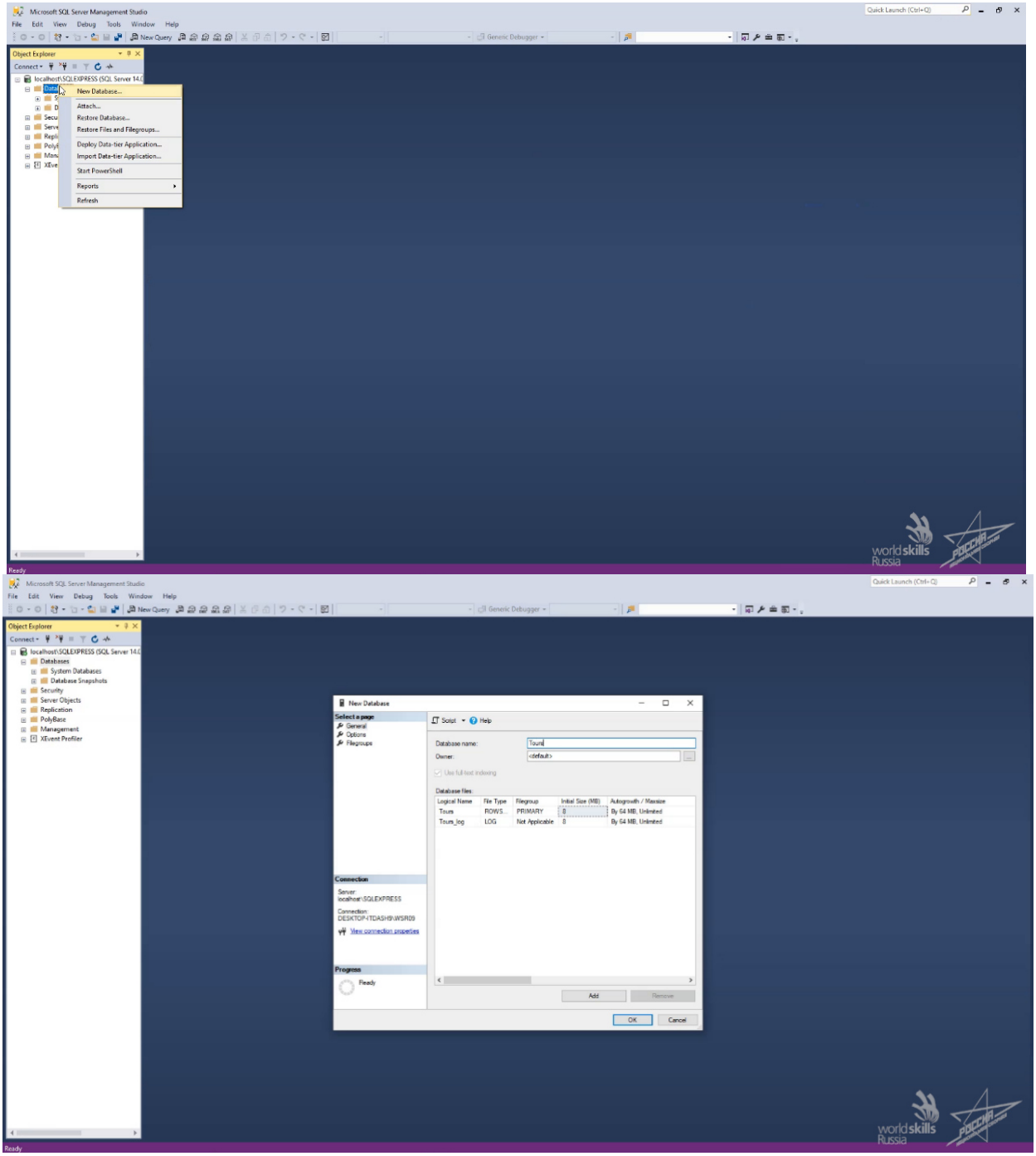
Server Name: localhost\SQLEXPRESS
Authentification: Windows Authentification



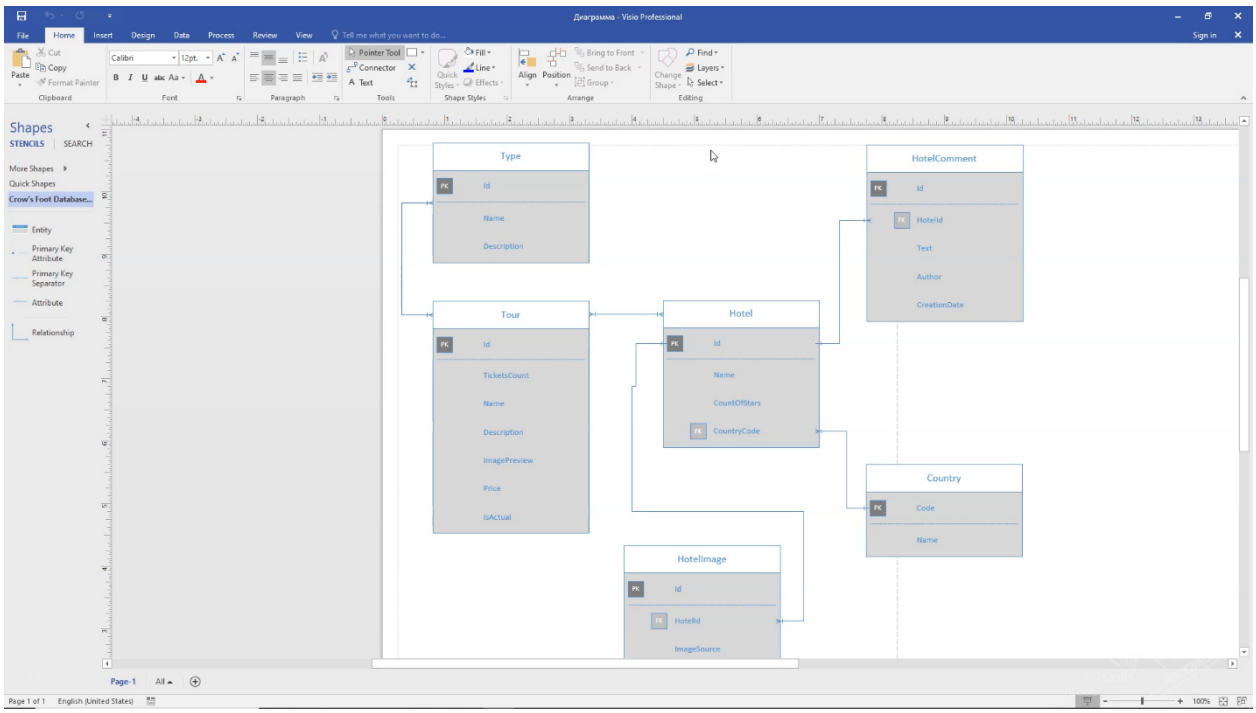
Б) Жмем Connect. Открывается список баз данных сервера



2. Создаем новую базу данных

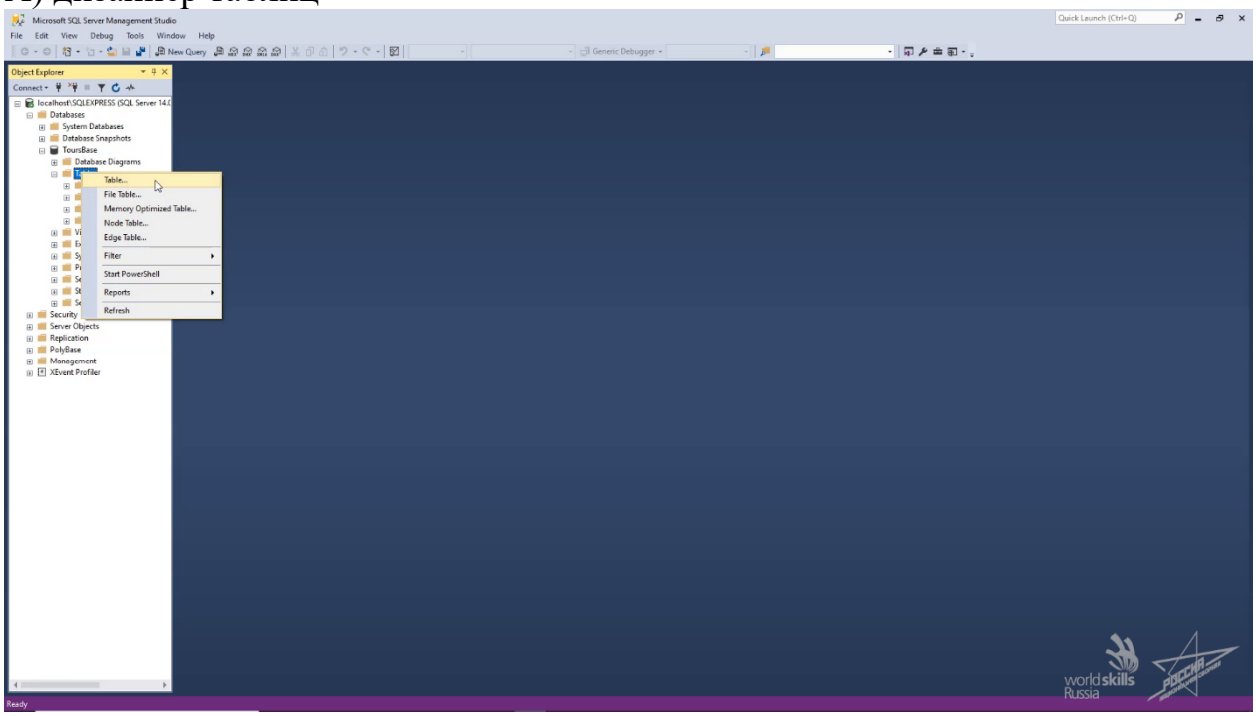


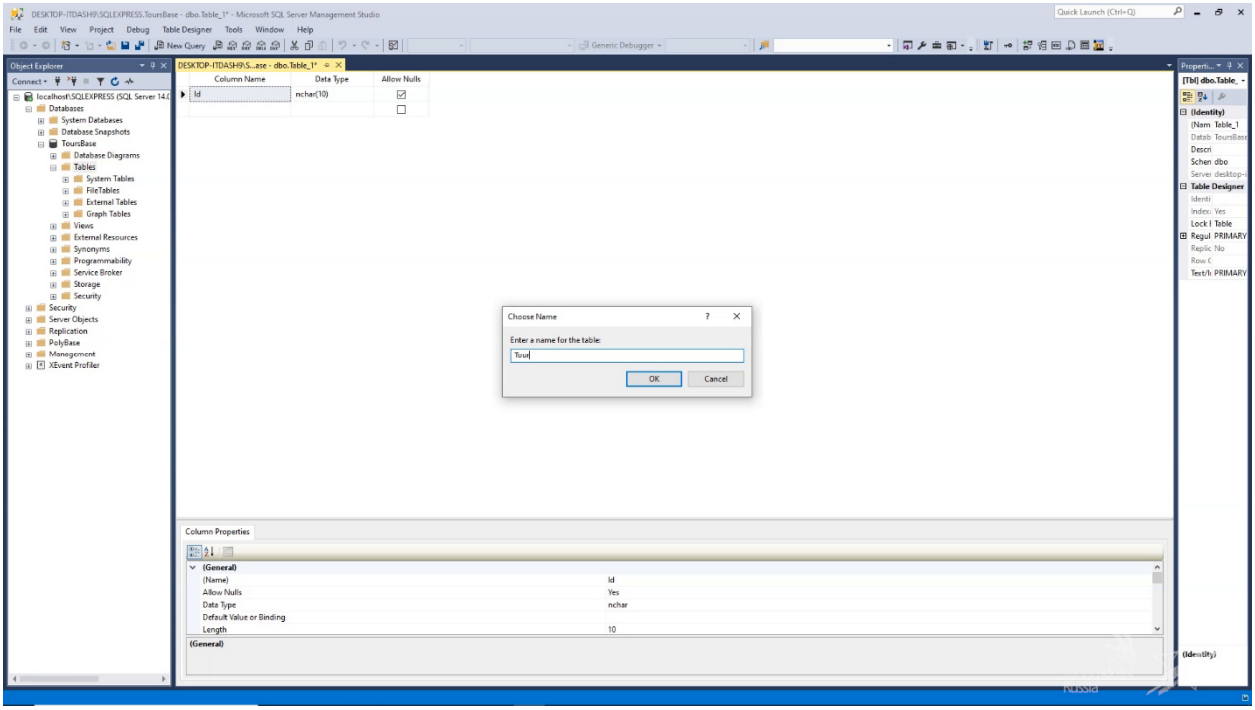
3. Определяем основные сущности и создаем таблицы. На основании предоставленной диаграммы ресурсов добавим таблицы в базу данных, разделив информацию на 2 блока: туры (туры, типы) и отели (отели, отзывы, изображения, страны)



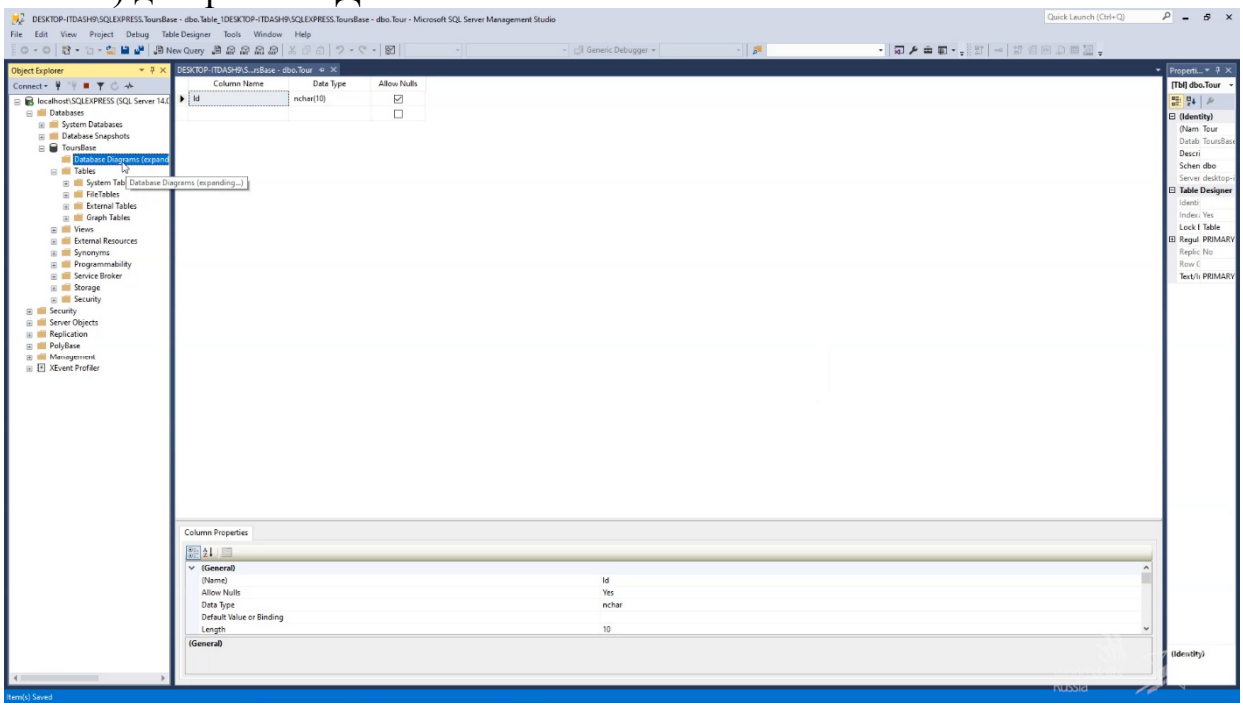
Создаем таблицы.
Существует несколько способов:

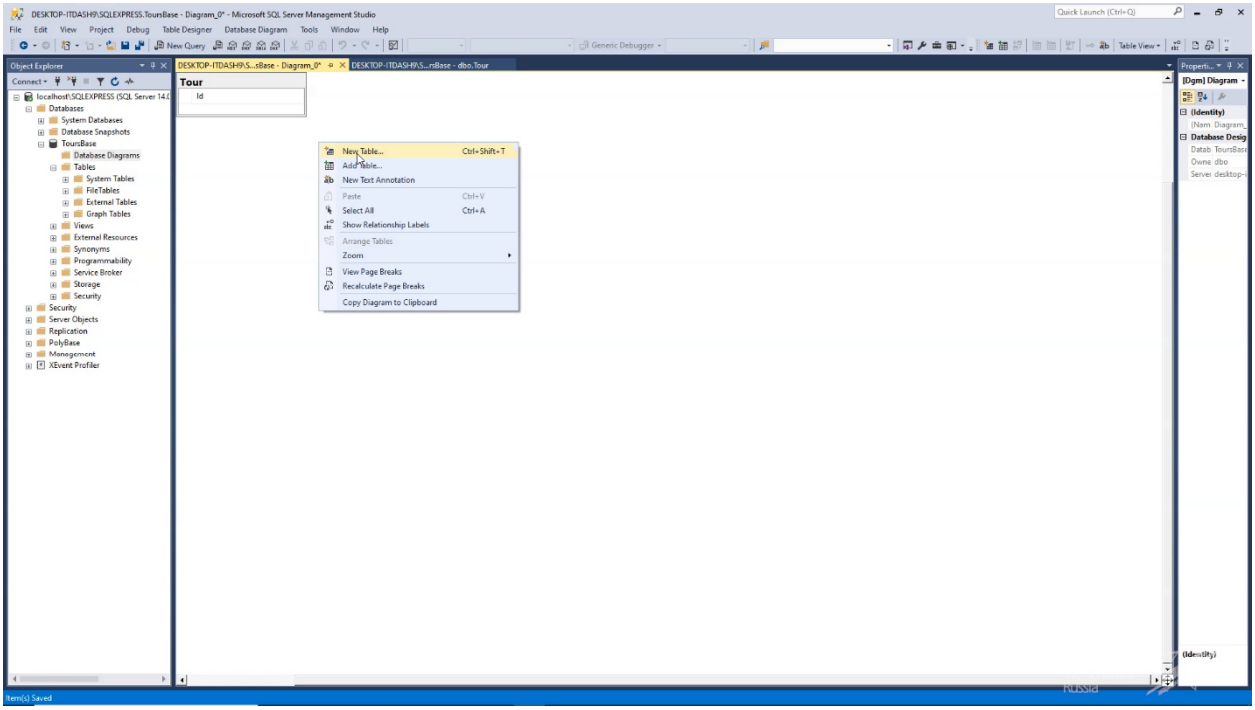
А) дизайнер таблиц



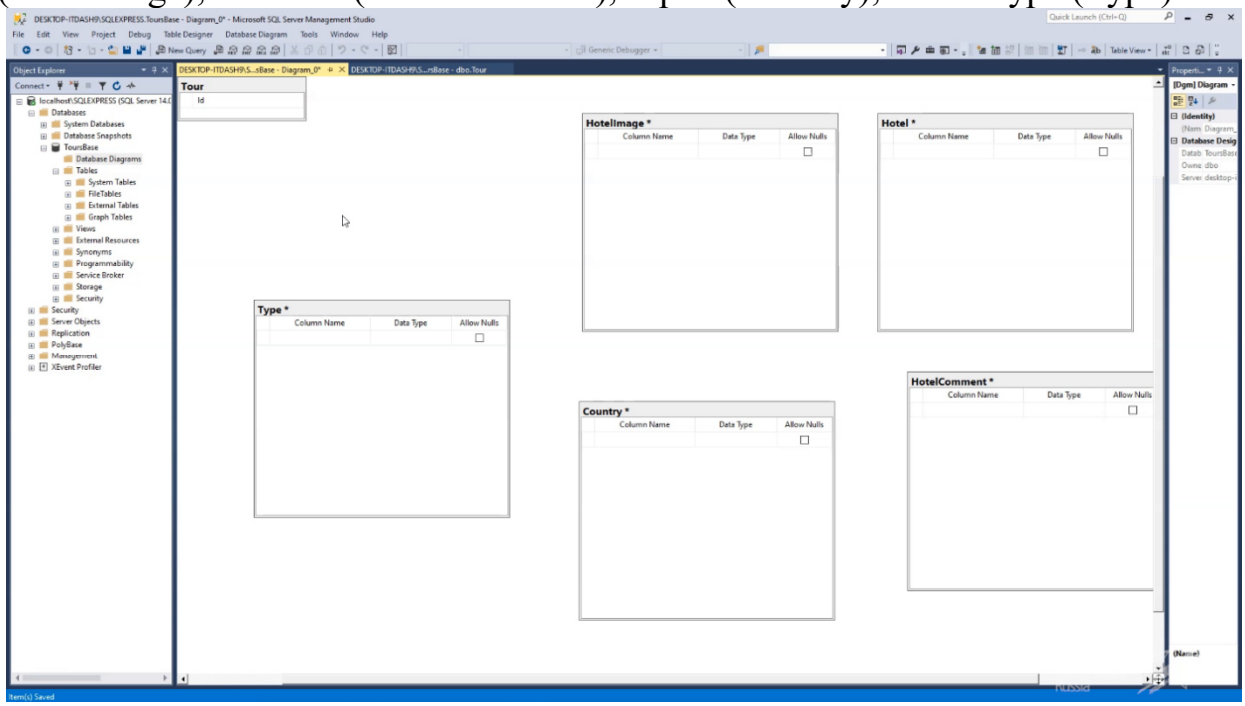


Б) диаграмма БД

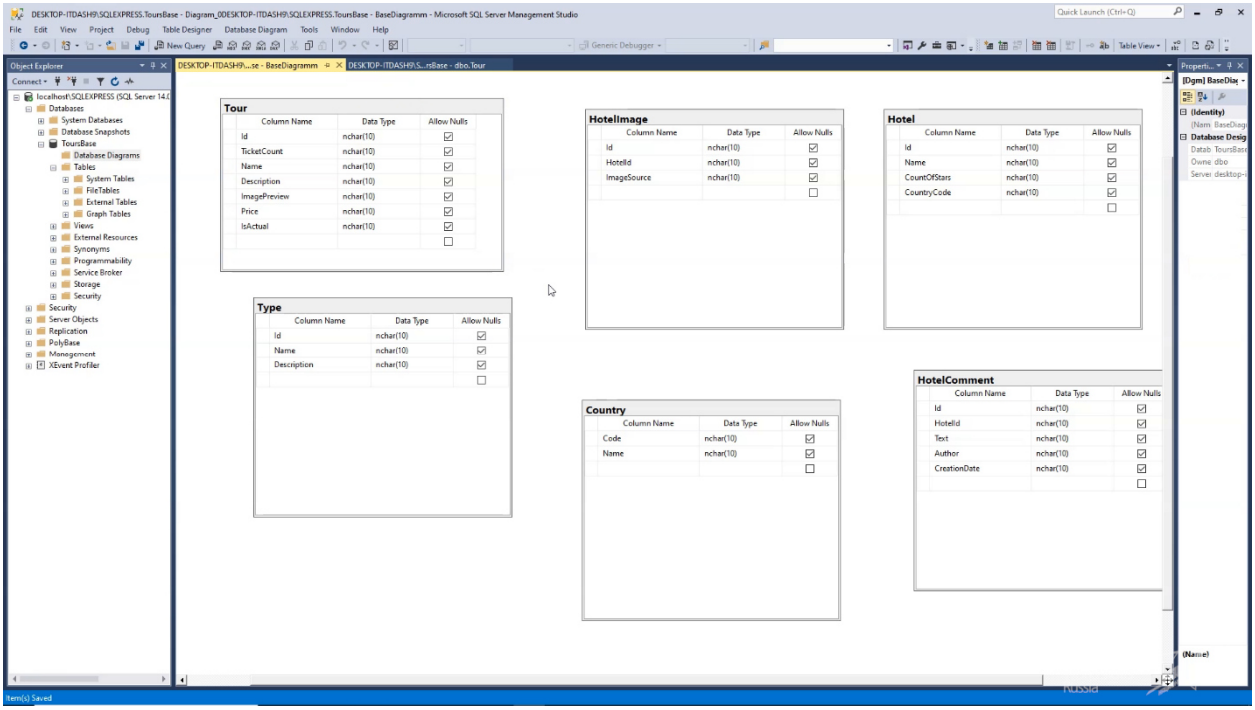




Создаем таблицы туров (Tour), отелей (Hotel), изображений (Hotelimage), отзывов (HotelComment), стран (Country), типов тура (Type)



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

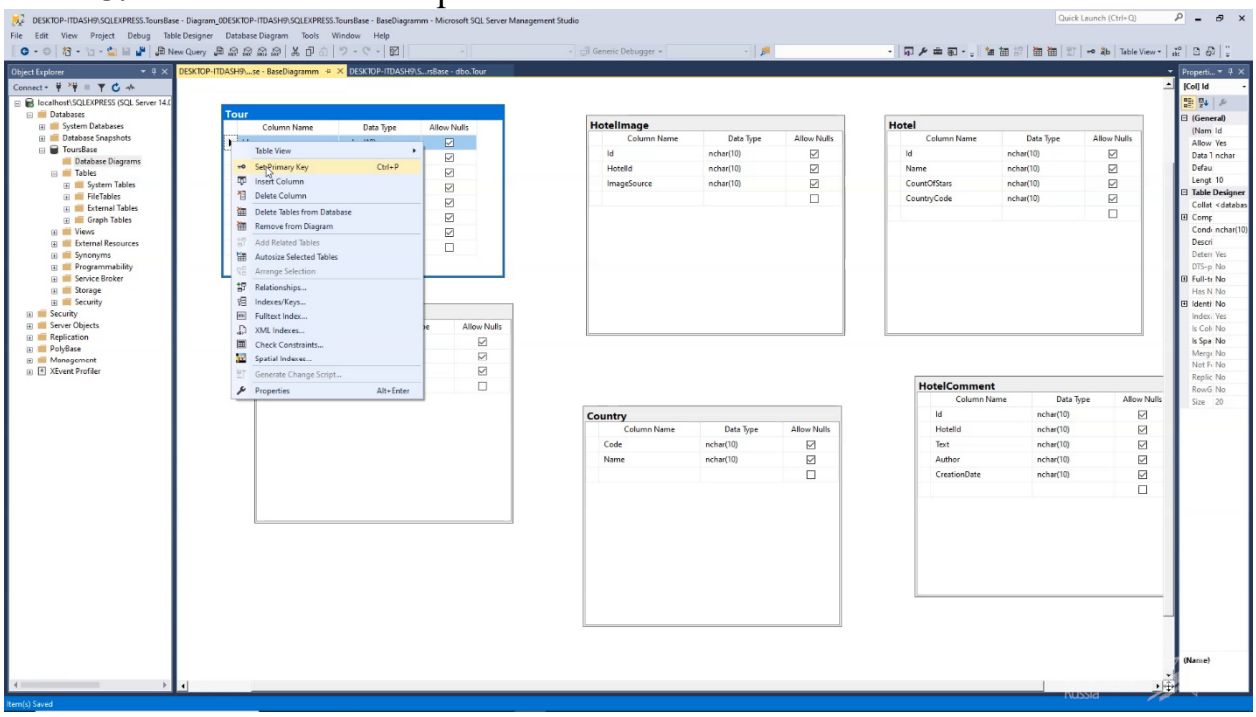
- Code

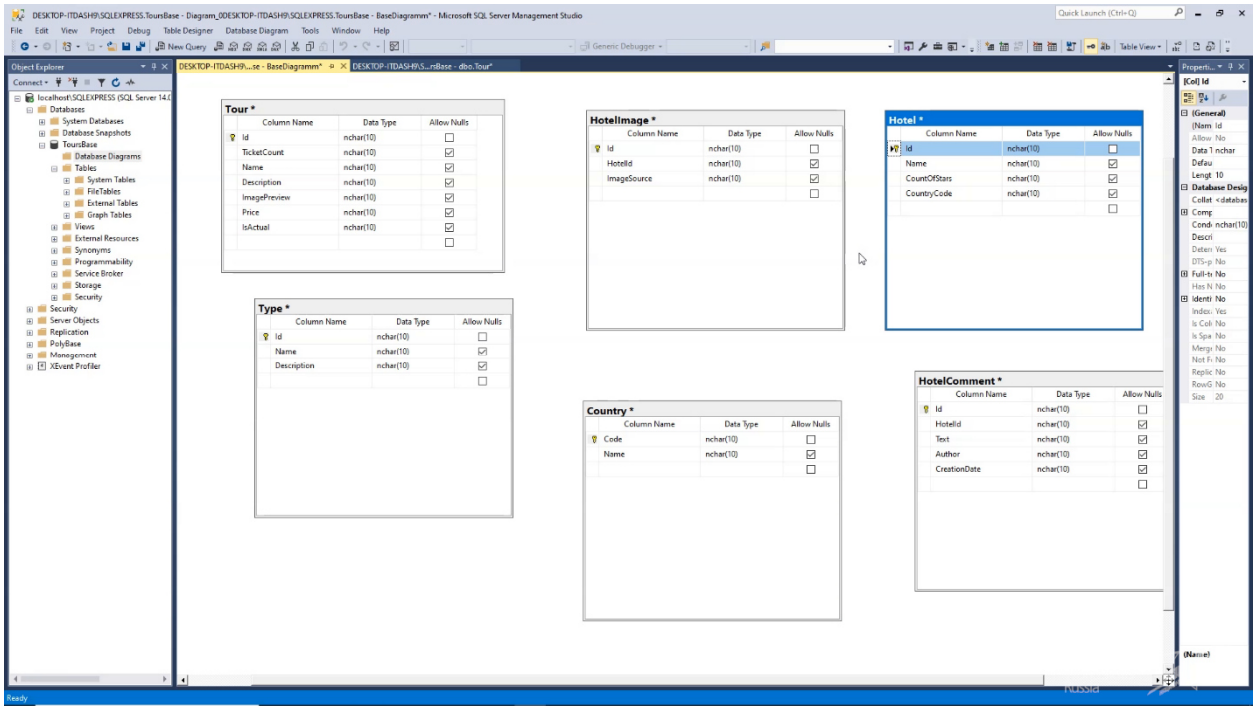
- Name

Е)Таблица Type

- Name
- Description

5. Расставляем первичные ключи





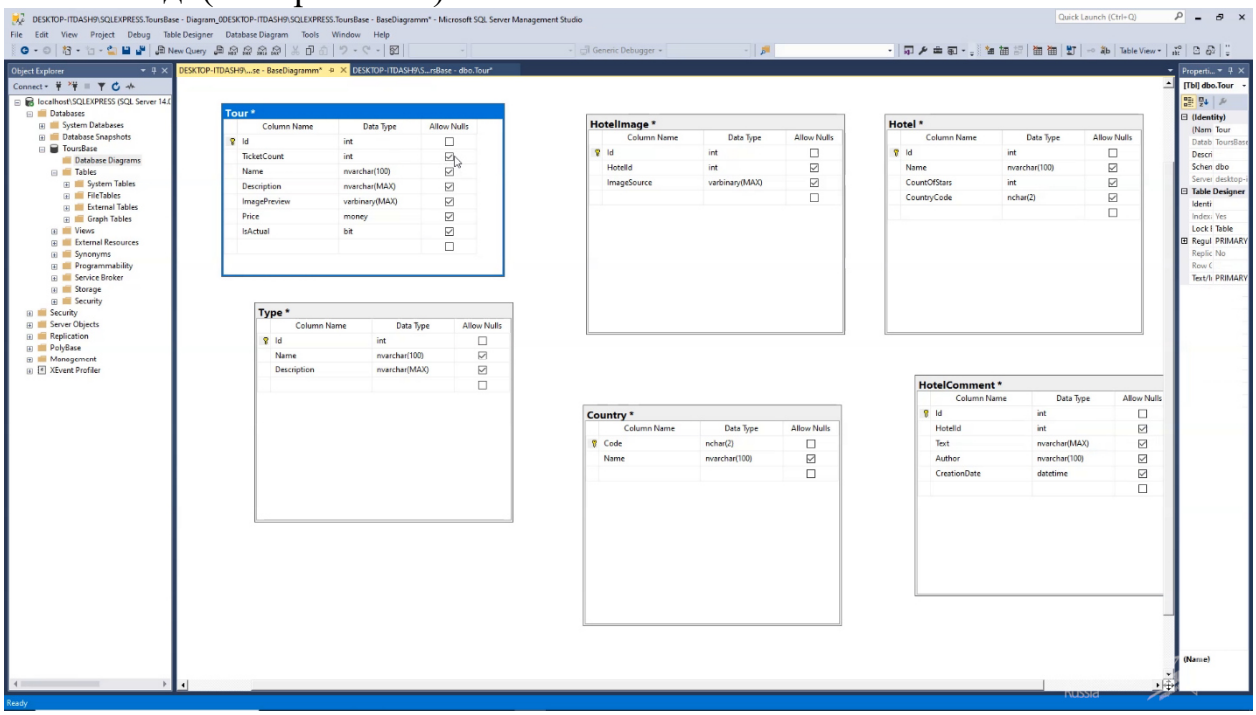
Важно

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

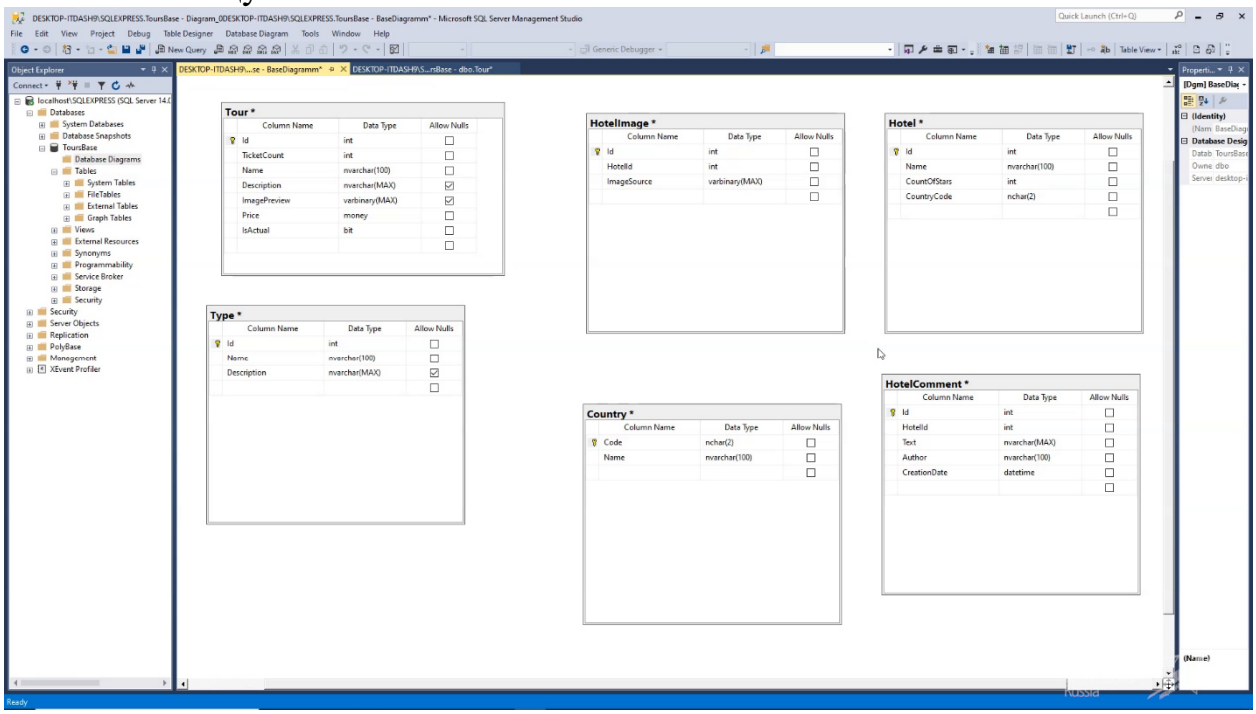
6. Устанавливаем типы данных

- 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 необязательным при заполнении в таблицу

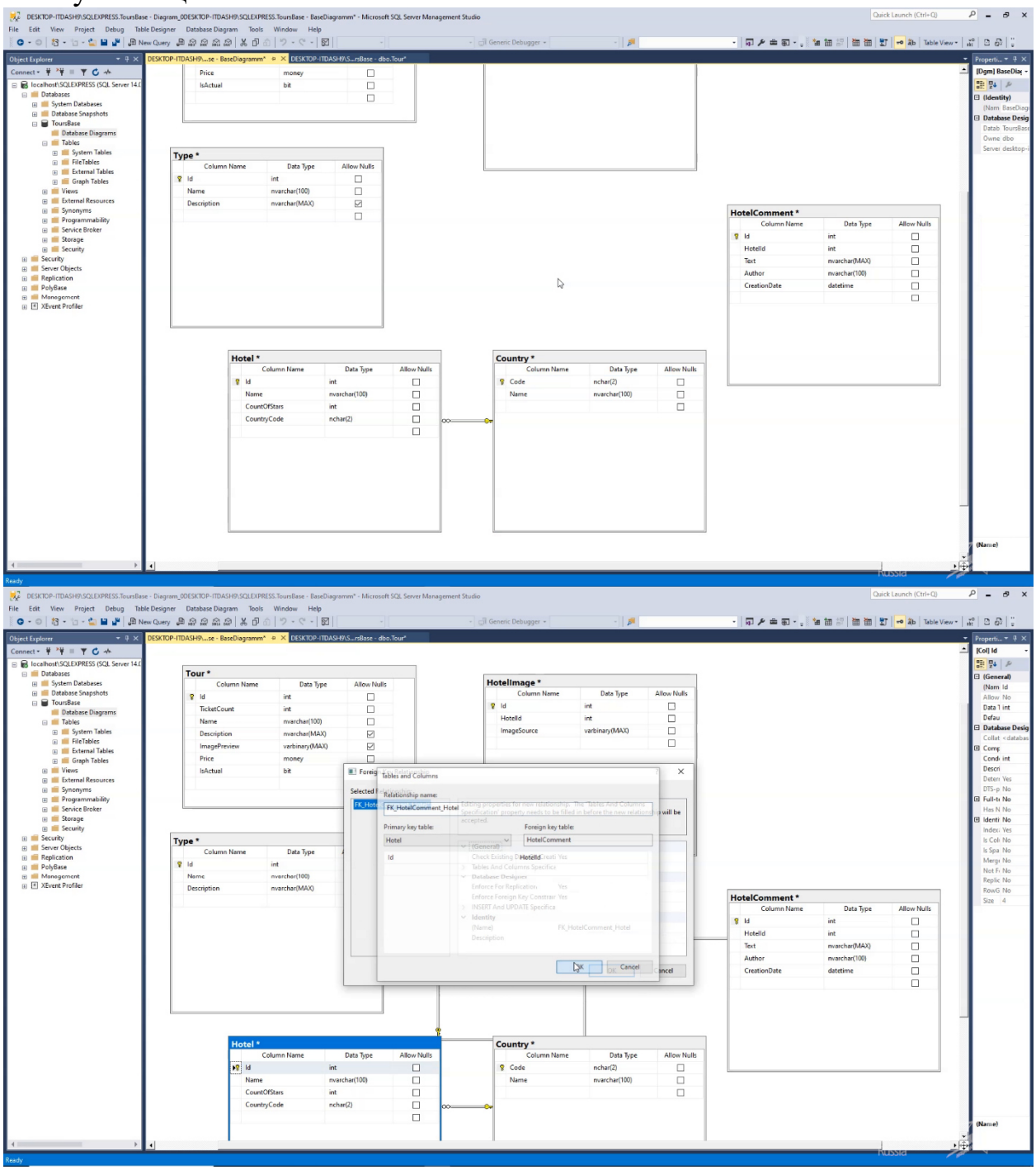


8. Устанавливаем связи между таблицами

А) один-ко-многим

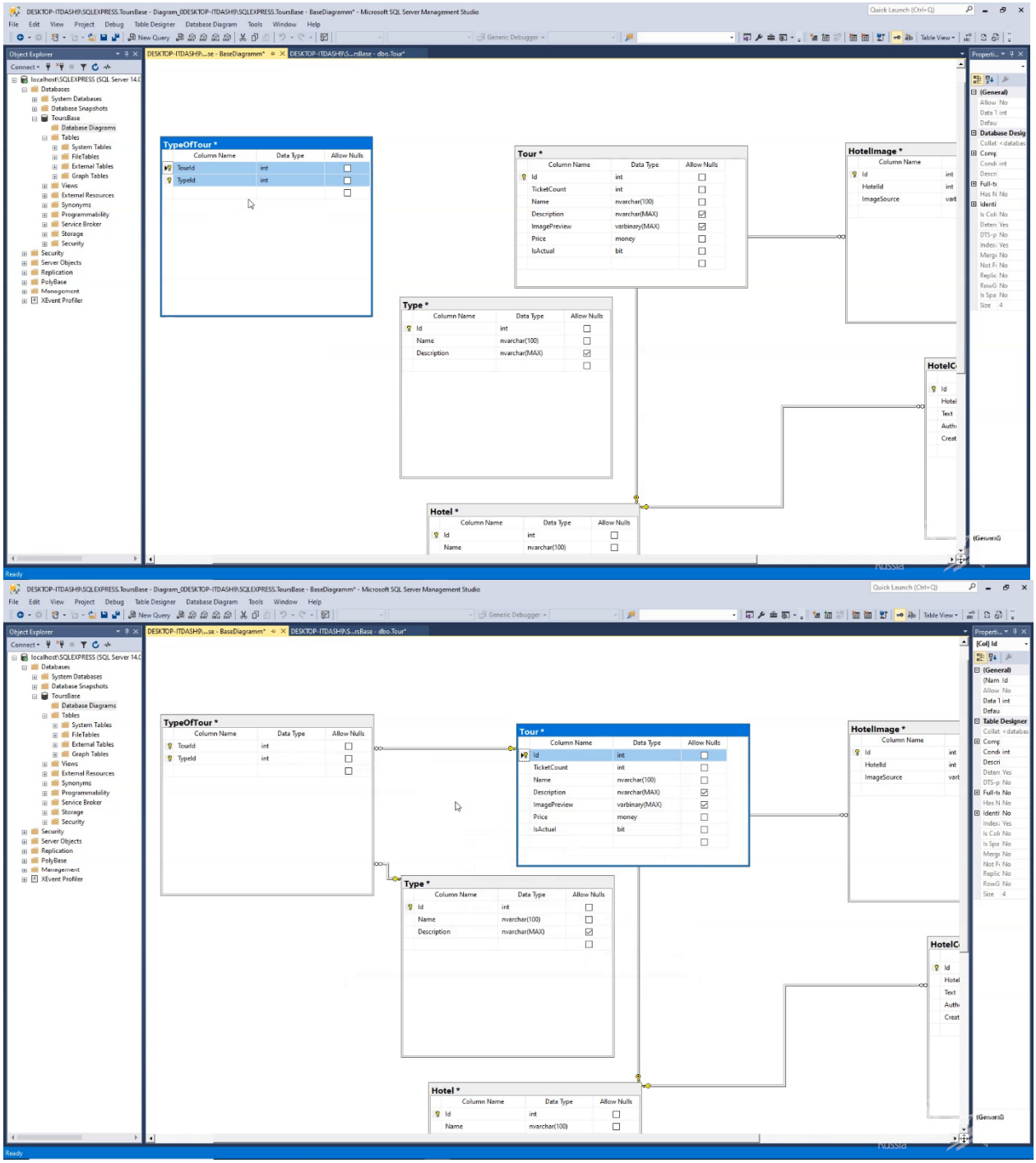
Чтобы связать таблицу стран и отелей, в таблицу Hotel необходимо добавить специальное поле — внешний ключ (в нашем случае это CountryCode), который по типу совпадает с тем, что является первичным ключом в таблице Country.

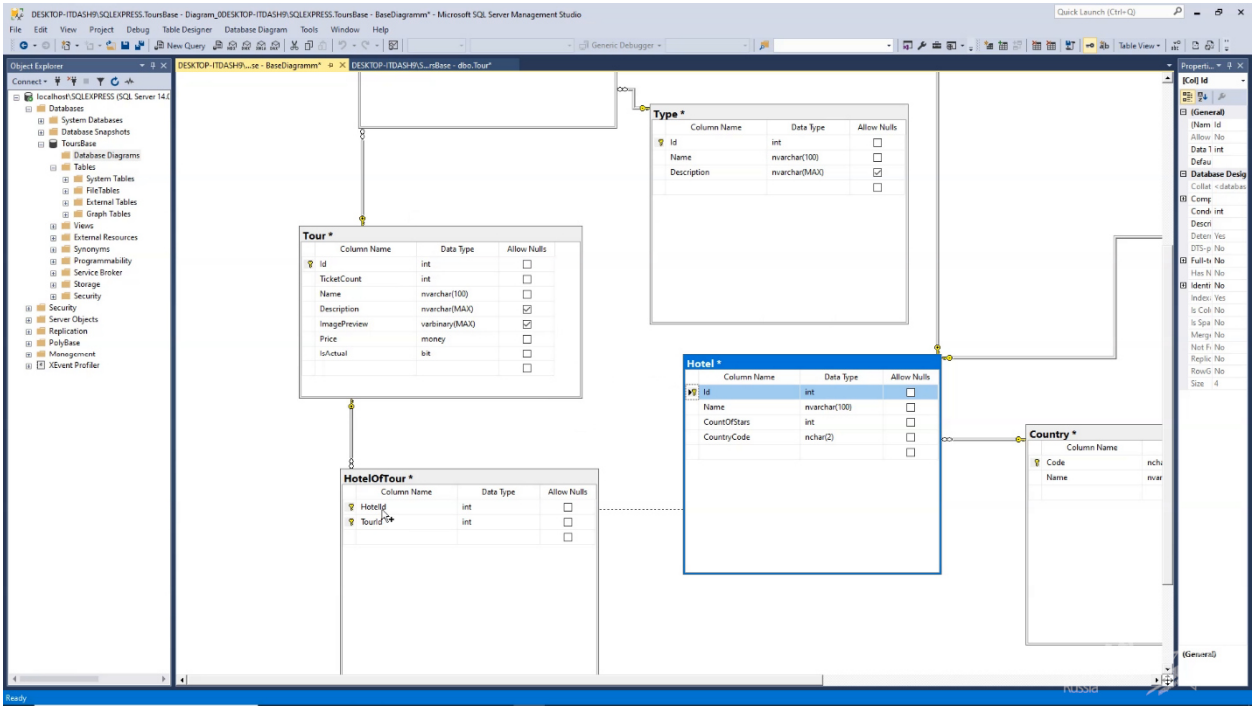
Далее от первичного ключа таблицы Country ведется связь к внешнему ключу таблицы



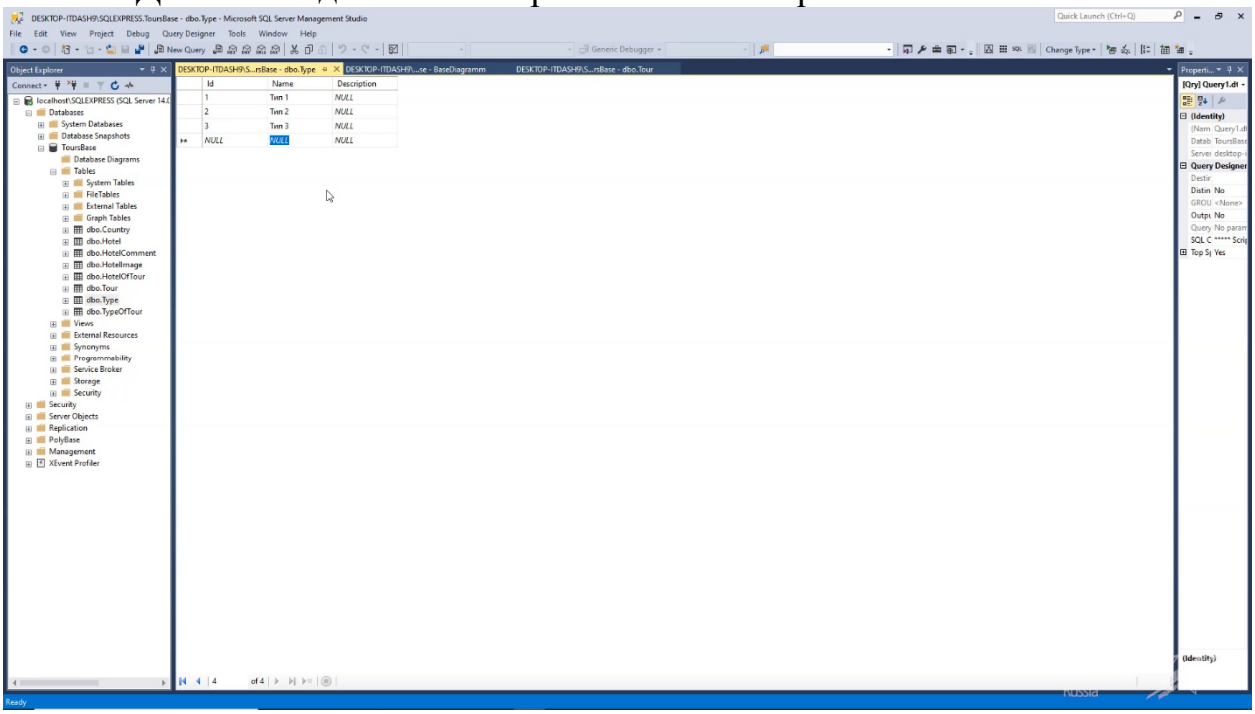
Б) многие-ко-многим

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 TypeOfTour) и создадим поля — первичные ключи из других таблиц. В данной таблице оба поля будут являться ключевыми

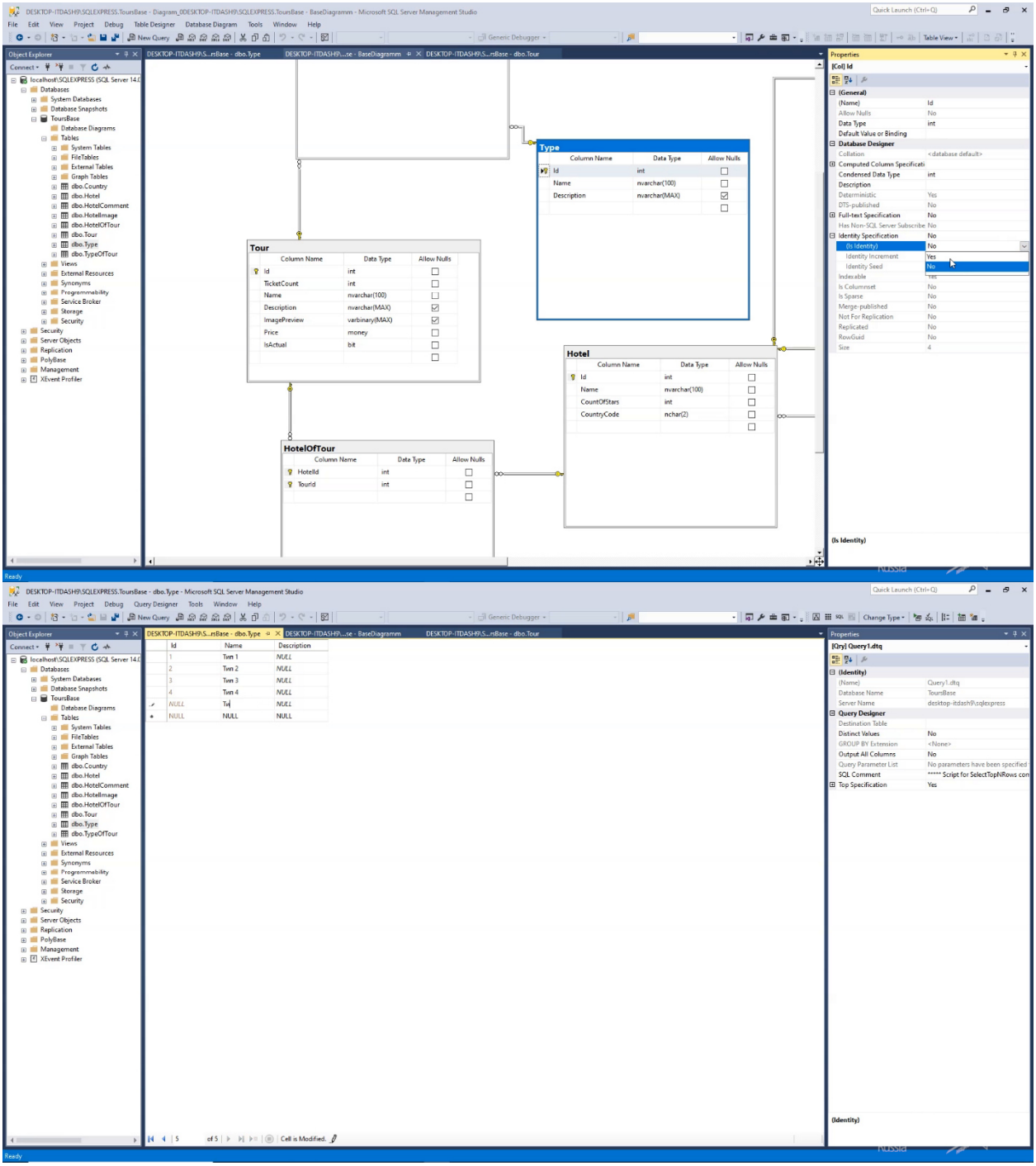

TypeOfTour *
Column Name
Data Type
Allow Nulls
TourId
int
TypeId
int
Tour *
Id
int
TicketCount
int
Name
nvarchar(100)
Description
nvarchar(MAX)
ImagePreview
varbinary(MAX)
Price
money
IsActual
bit
HotelImage *
HotelId
ImageSource
Type *
Name
Hotel *



9. Добавляем данные. Настраиваем автоинкременты

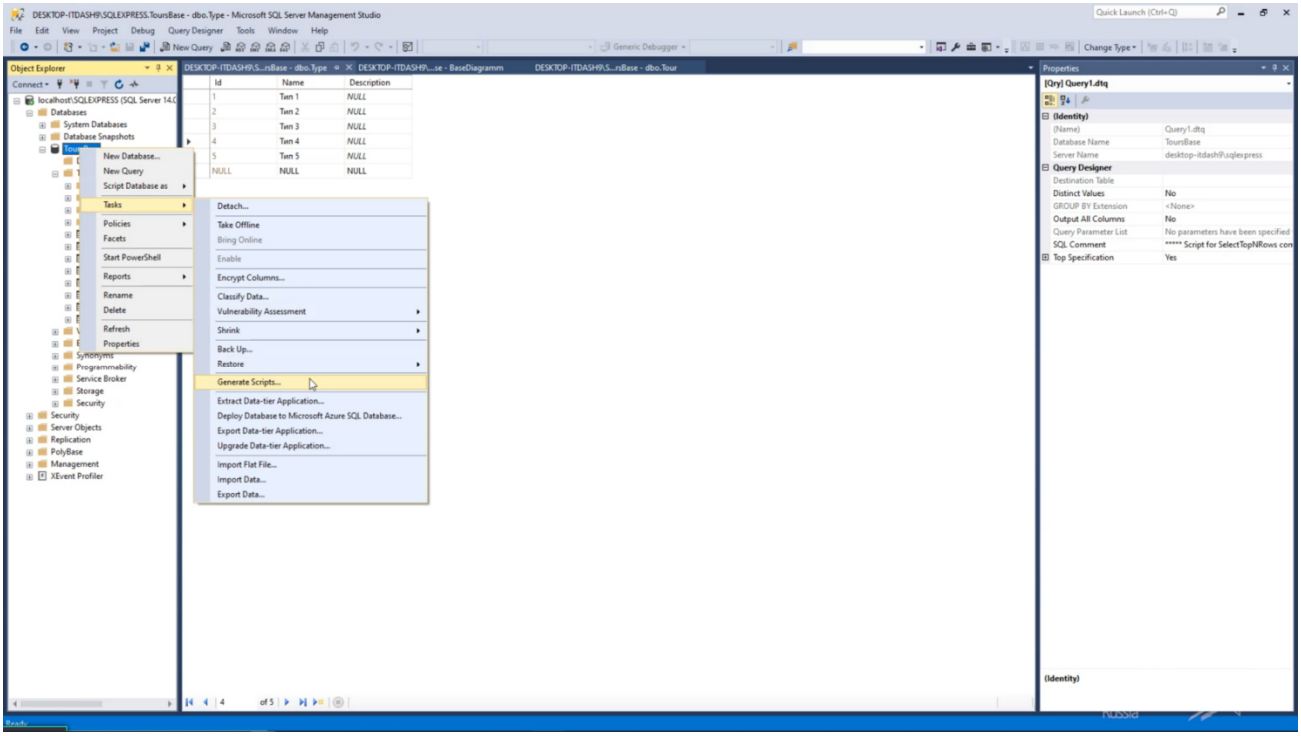


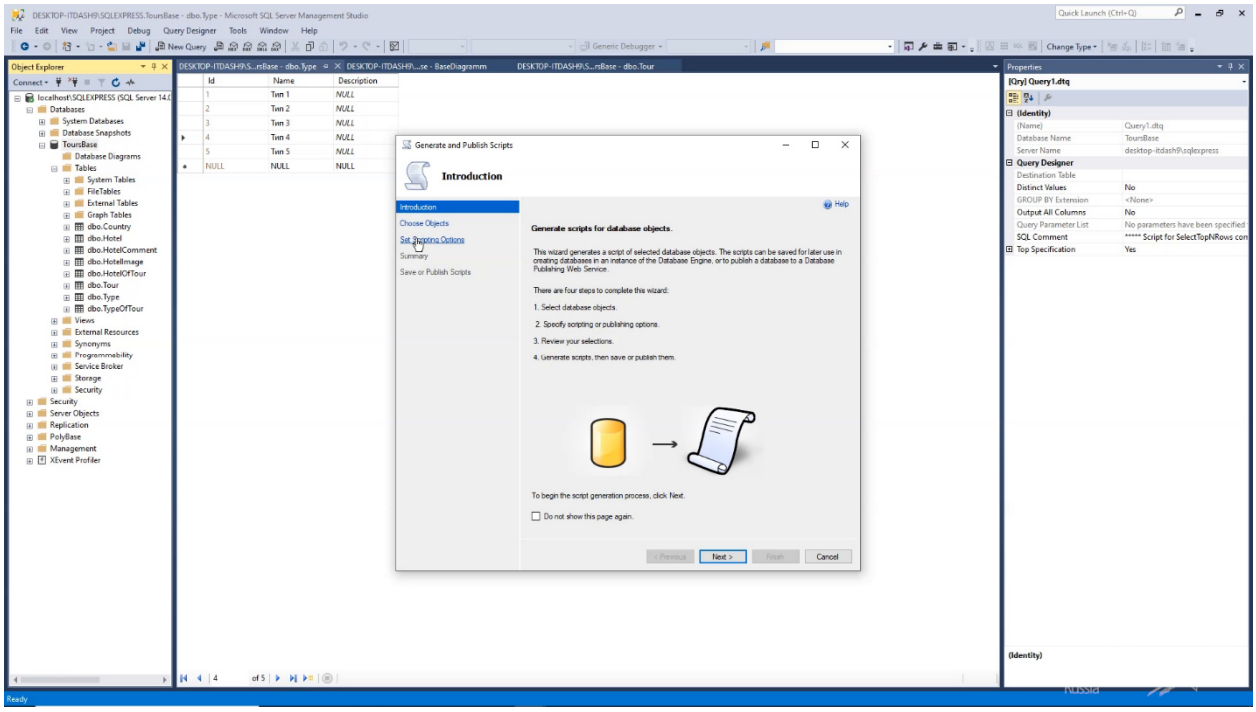
Для того, чтобы не вести учет количества порядковых номеров, в настройках поля можно установить автоматическое определение значения поля при добавлении записи. Например, если в таблице есть 3 записи с номерами от 1 до 3, то следующая запись автоматически будет иметь номер 4

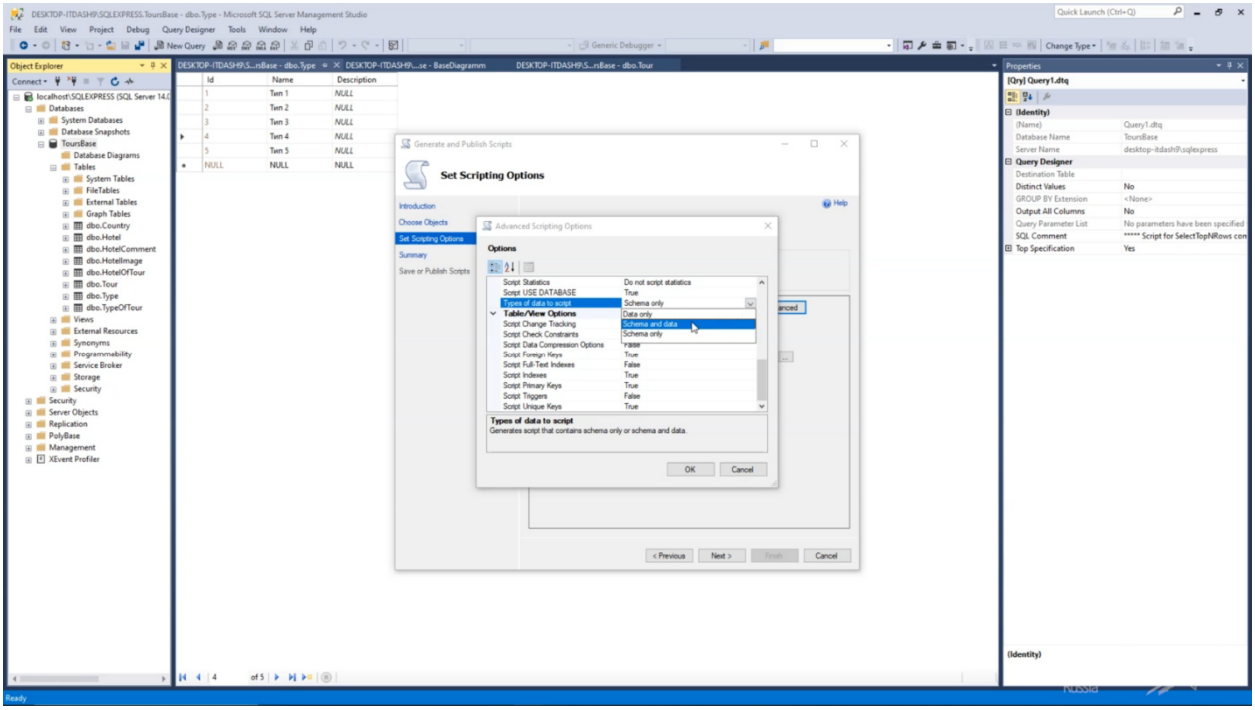


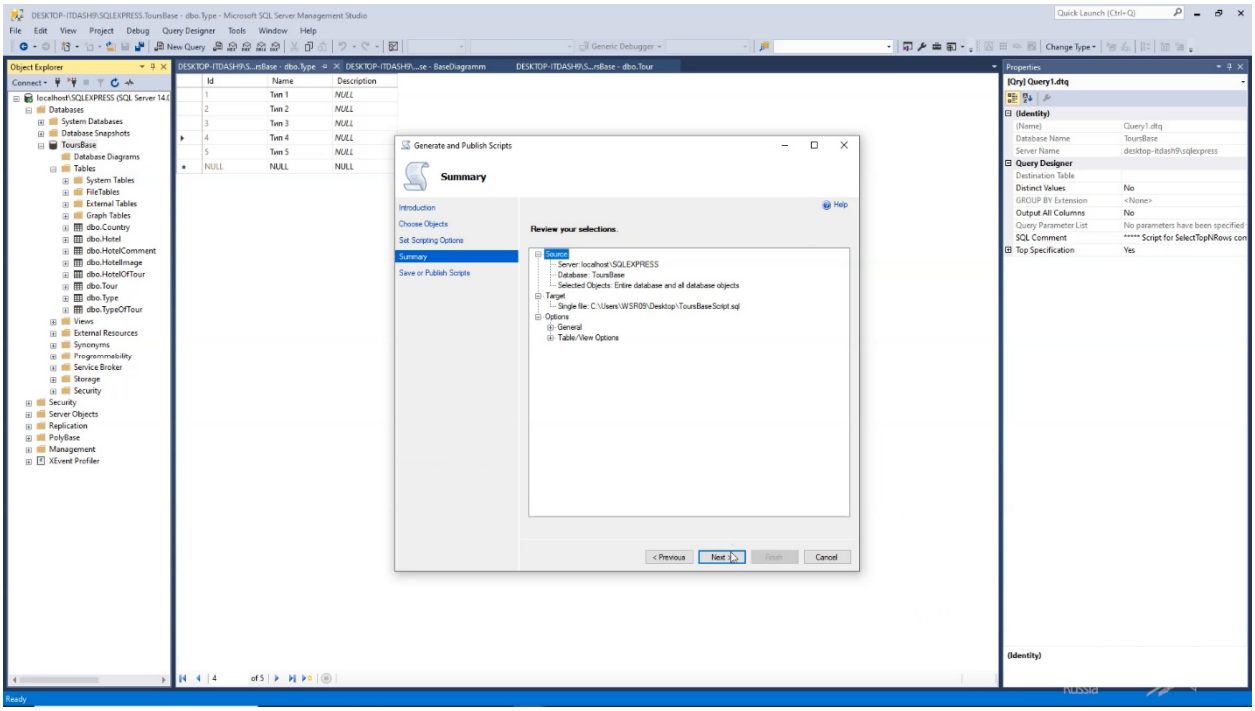
10. Сохранение БД и создание скрипта

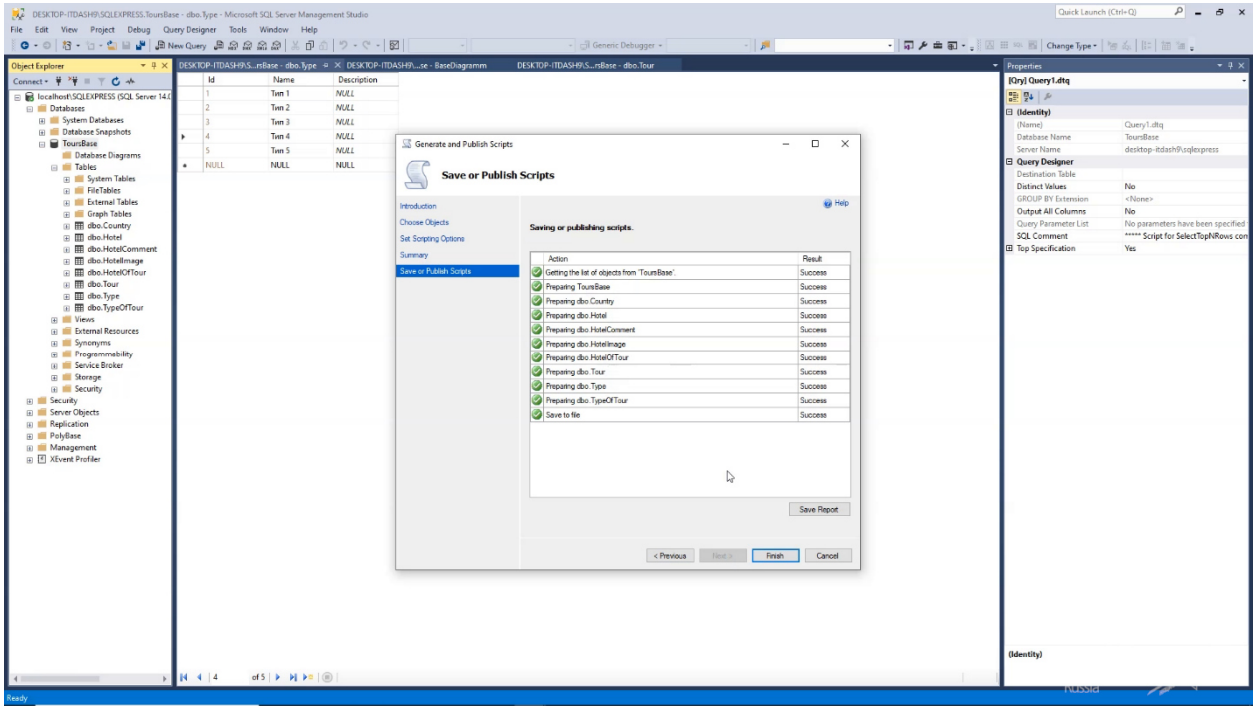
Все созданные базы данных хранятся на сервере. Чтобы перенести базу данных на другой сервер, необходимо правильно ее сохранить. Один из методов переноса - создание скрипта базы данных.


DESKTOP-ITDASH9\SQLEXPRESS.ToursBase - dbo.Type - Microsoft SQL Server Management Studio
Object Explorer
Connect
localhost\SQLEXPRESS (SQL Server 14.0
Databases
System Databases
Database Snapshots
New Database...
New Query
Script Database as
Tasks
Policies
Facets
Start PowerShell
Reports
Rename
Delete
Refresh
Properties
Detach...
Take Offline
Bring Online
Enable
Encrypt Columns...
Classify Data...
Vulnerability Assessment
Shrink
Back Up...
Restore
Generate Scripts...
Extract Data-tier Application...
Deploy Database to Microsoft Azure SQL Database...
Export Data-tier Application...
Upgrade Data-tier Application...
Import Flat File...
Import Data...
Export Data...
Id
Name
Description
Properties
[Qry] Query1.dtq
(Identity)
Query Designer
Ready


Generate and Publish Scripts
Introduction
Introduction
Choose Objects
Set Scripting Options
Summary
Save or Publish Scripts
Help
Generate scripts for database objects.
This wizard generates a script of selected database objects. The scripts can be saved for later use in creating databases in an instance of the Database Engine, or to publish a database to a Database Publishing Web Service.
There are four steps to complete this wizard:
1. Select database objects.
2. Specify scripting or publishing options.
3. Review your selections.
4. Generate scripts, then save or publish them.
To begin the script generation process, click Next.
Do not show this page again.
< Previous
Next >
Finish
Cancel

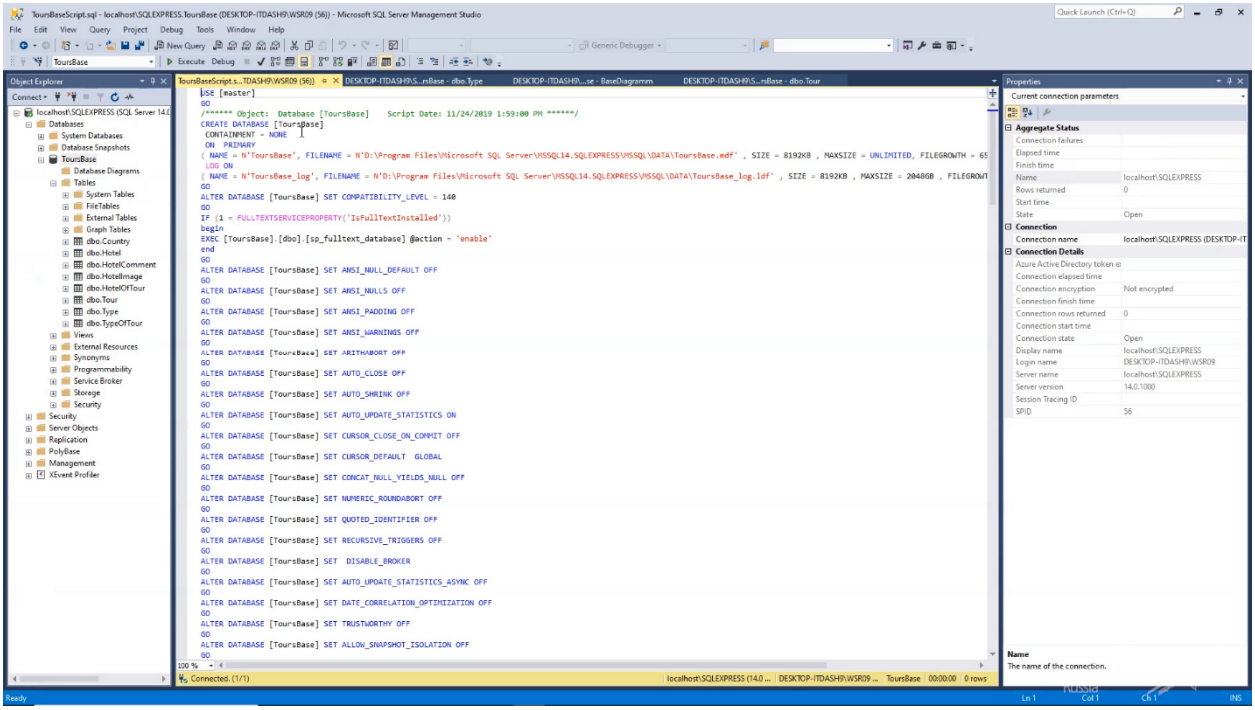

Set Scripting Options
Advanced Scripting Options
Options
Types of data to script
Generates script that contains schema only or schema and data.
OK
Cancel
< Previous
Next >
Finish
Cancel


Summary
Review your selections.
Source
Server: localhost\SQLEXPRESS
Database: ToursBase
Selected Objects: Entire database and all database objects
Target
Single file: C:\Users\WSR09\Desktop\ToursBaseScript.sql
Options
General
Table/View Options
< Previous
Next >
Cancel



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



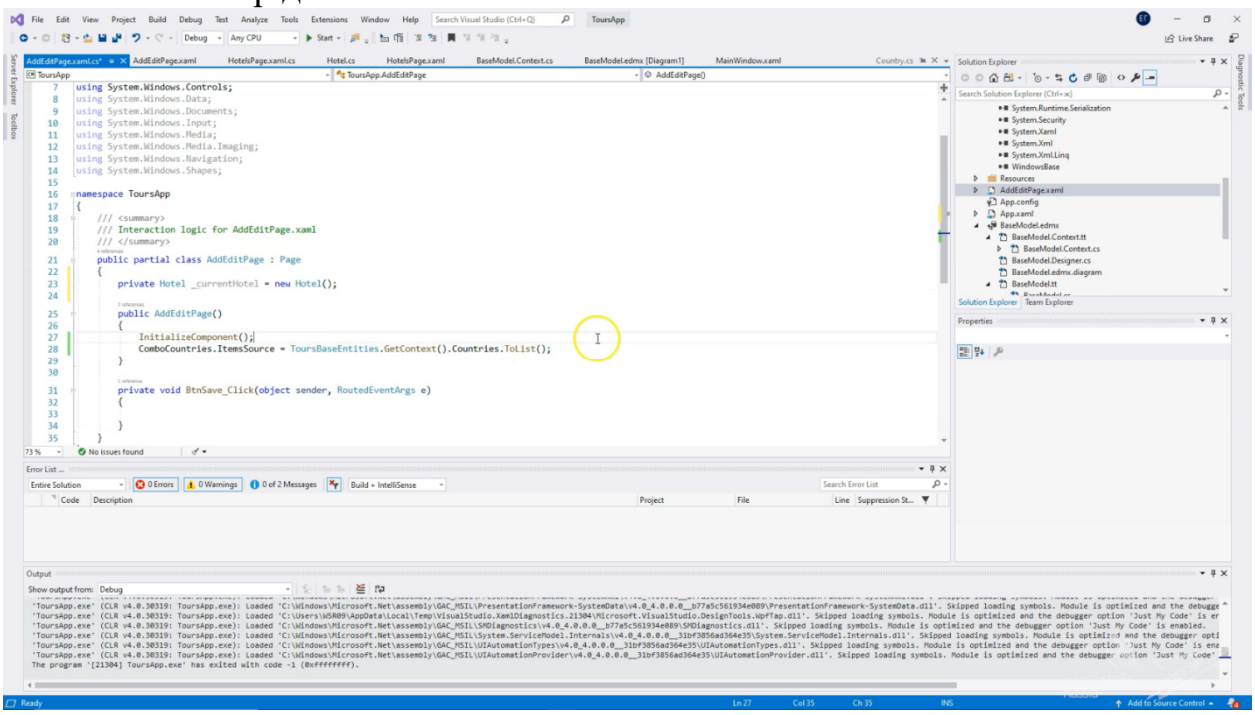
Интерактивное задание
https://nationalteam.worldskills.ru/skills/sozdanie-bazy-dannykh/

Практическая работа №8. Работа с базой данных в графических инструментах: добавление, редактирование, чтение, обновление и удаление данных

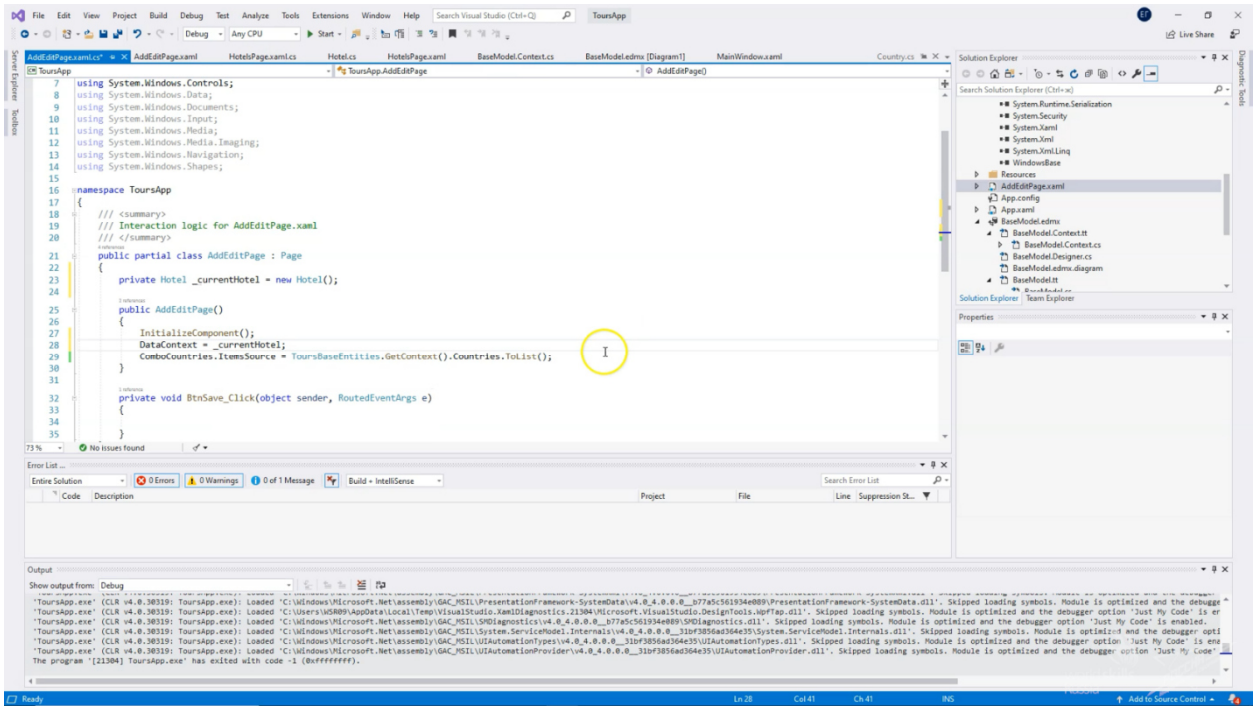
Краткие теоретические сведения

Реализация функции добавления

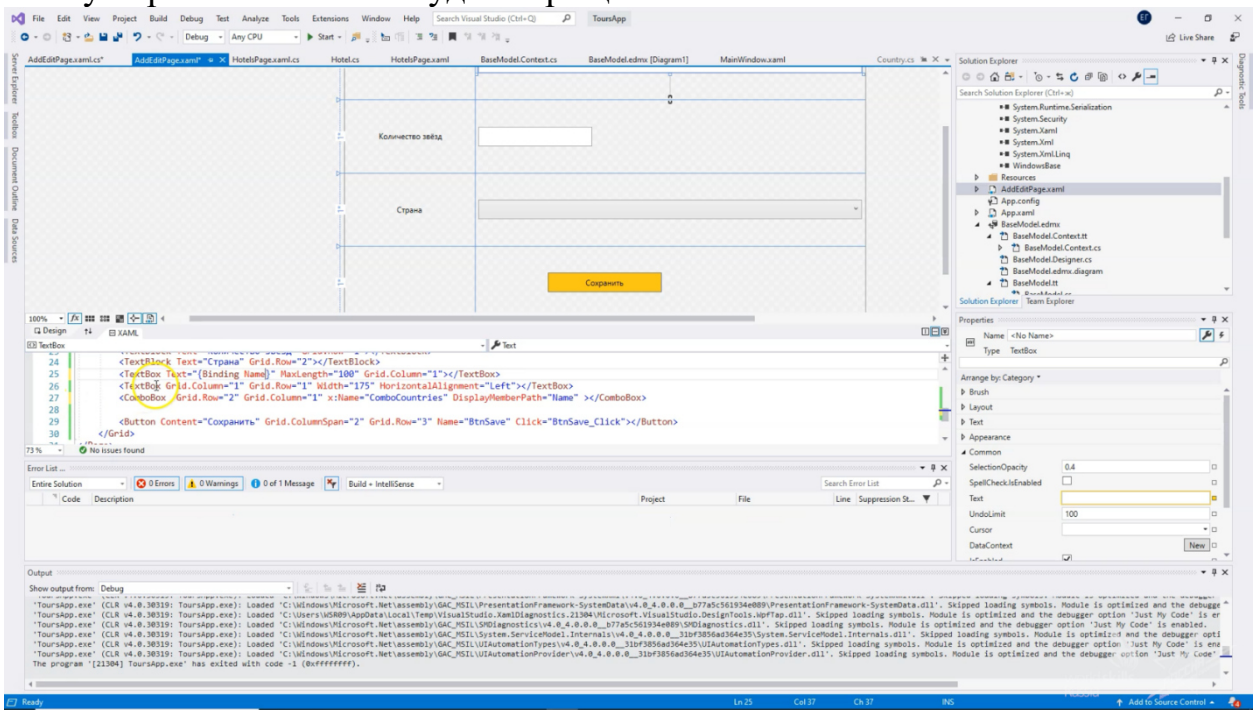
1. На странице AddEditPage добавим новое поле, которое будет хранить в себе экземпляр добавляемого отеля



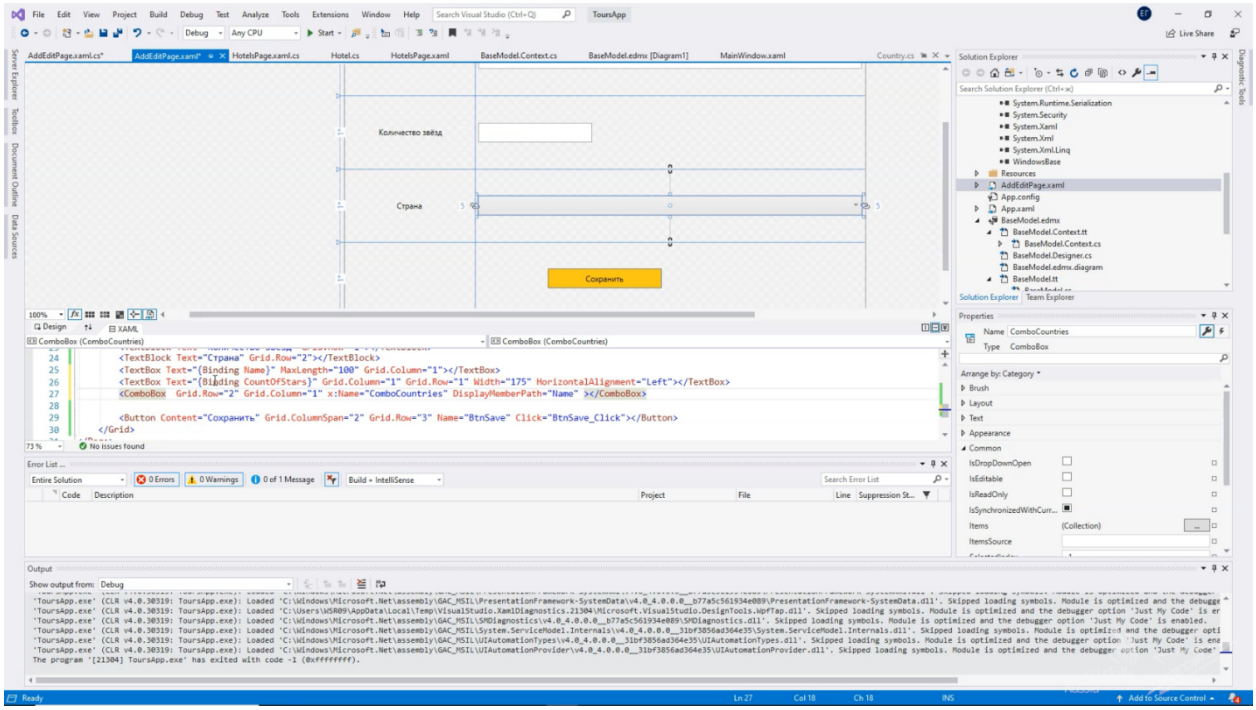
2. При инициализации установим DataContext страницы — этот созданный объект



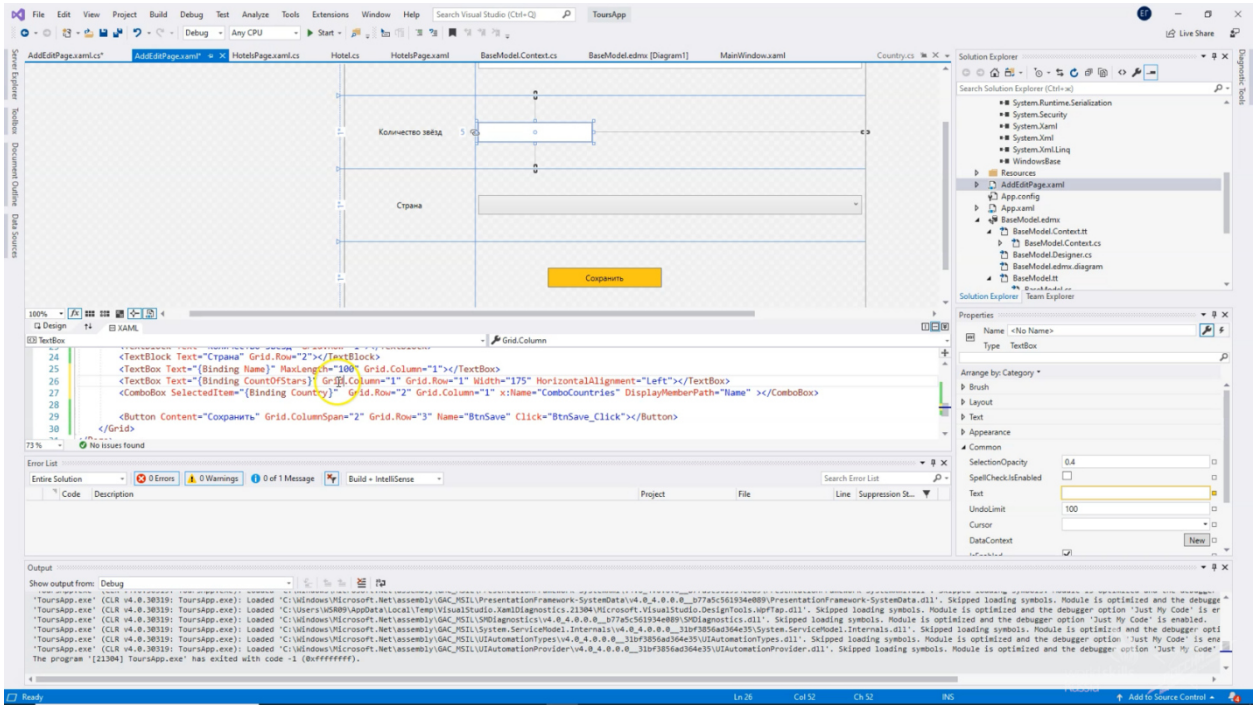
3. Затем, используя привязку данных, укажем, какому свойству обращаться к каждому элементу при загрузке данных. Например, свойство Text у первого TextBox'a будет обращаться к названию отеля



4. Второй элемент будет обращаться к количеству звезд

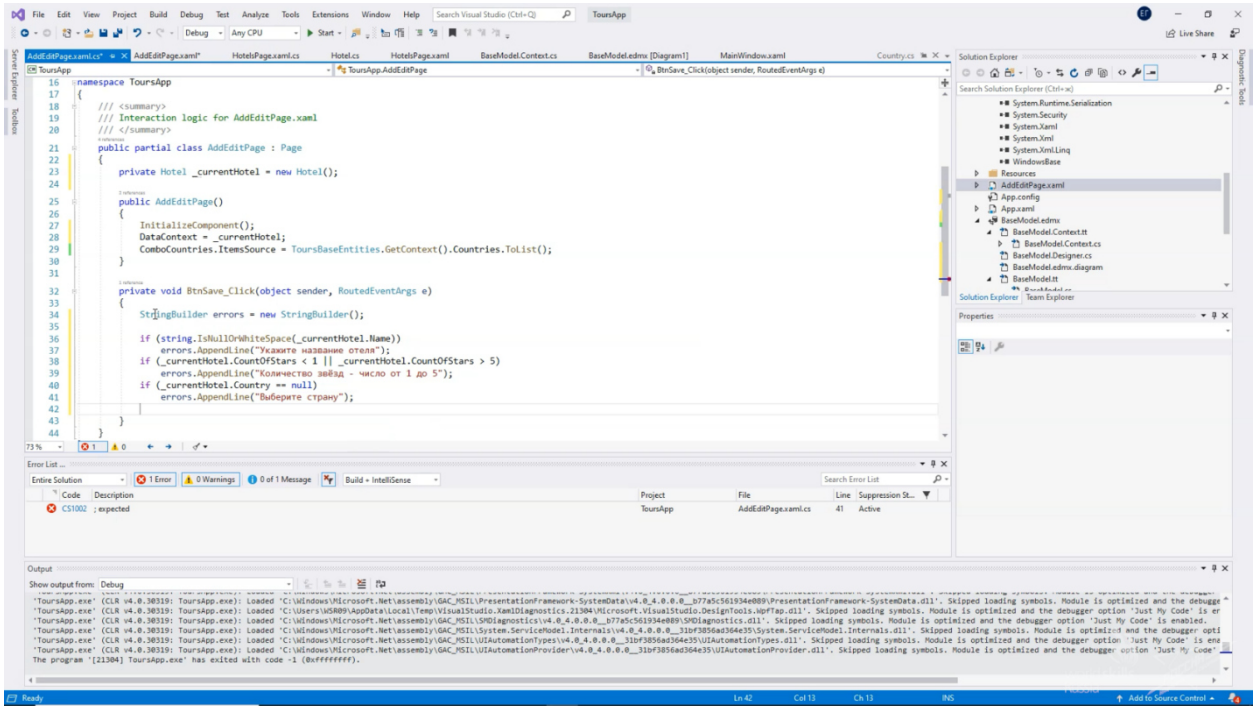


5. И комбо-бокс будет обращаться к стране, которую мы выбрали для отеля

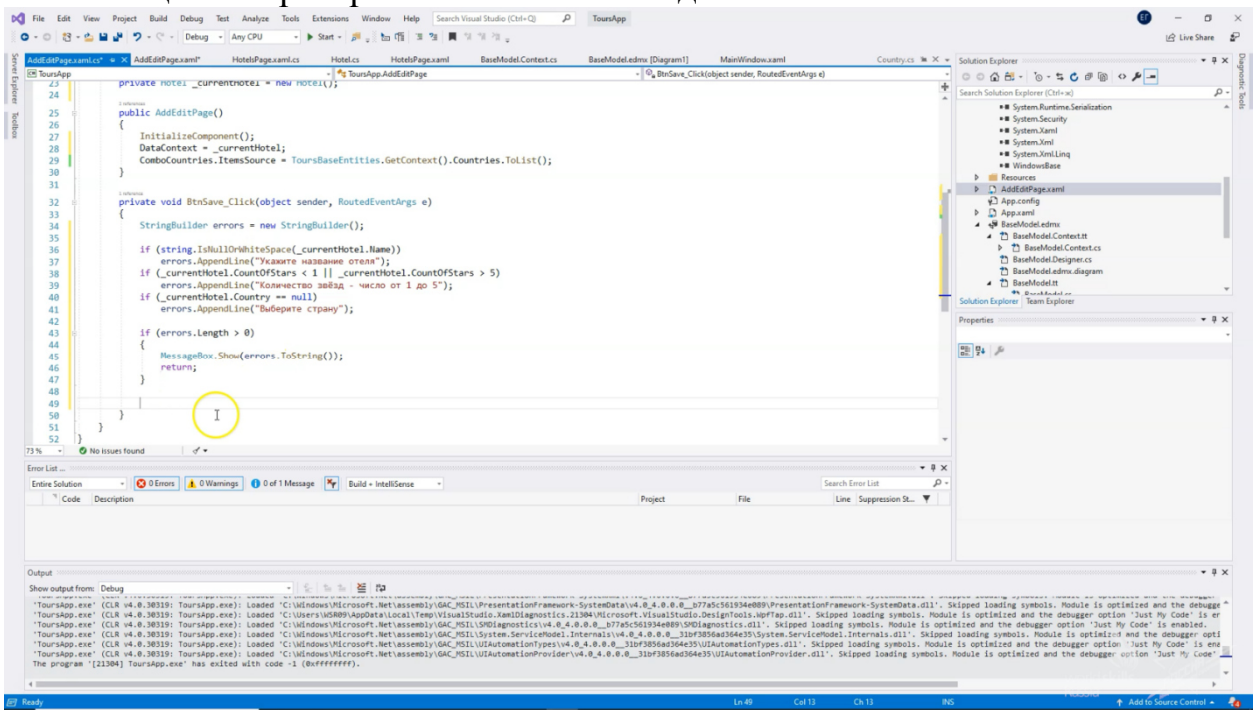


6. Далее обработаем нажатие на кнопку Сохранение и в коде пропишем логику обращения к модели данных и добавления нового экземпляра отеля

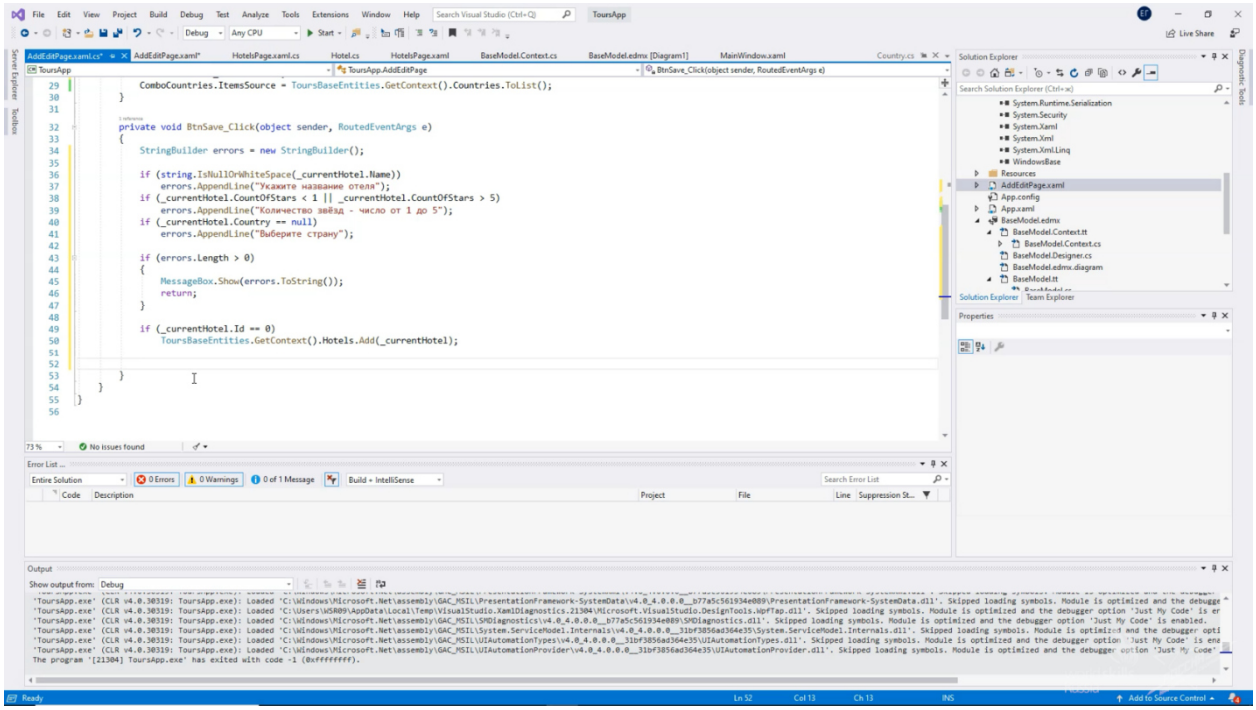
а) Прежде чем сохранять данные, сделаем проверки на количество символов, заполняемость объектов, звездность (т. к. количество звезд должно быть от одного до пяти) и выбор страны



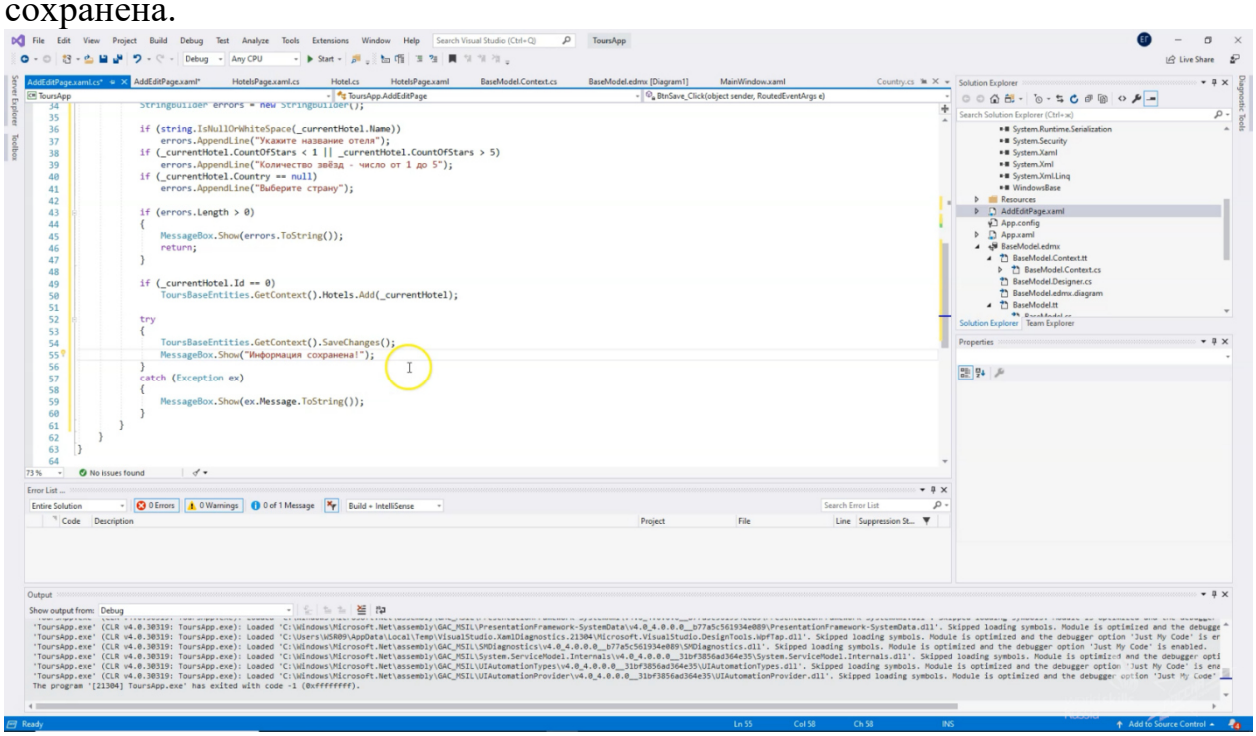
b) После прохождения проверки нужно узнать, возникли ли ошибки, обратившись к переменной errors. Если в переменной что-то есть, то необходимо вывести сообщение об ошибке (то, что накопилось во время проверки). Соответственно, дальнейшее выполнение функции не нужно, и с помощью оператора return — мы выходим



c) 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

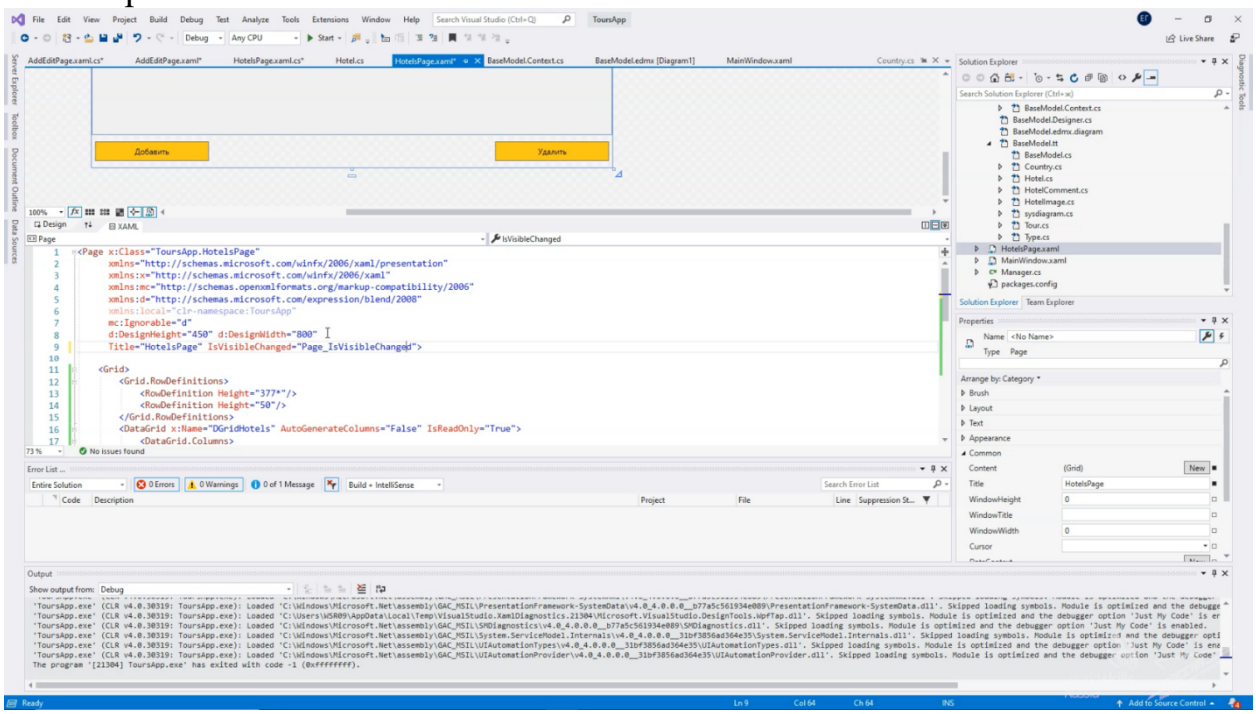


d) Далее напишем код для сохранения изменений, используя метод SaveChanges. Этот метод является коварным, его необходимо поместить в блок try-catch, чтобы он отработал корректно, и в случае возникновения какой-либо непредвиденной ошибки, приложение не «упало», а корректно работало. Мы выведем сообщение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

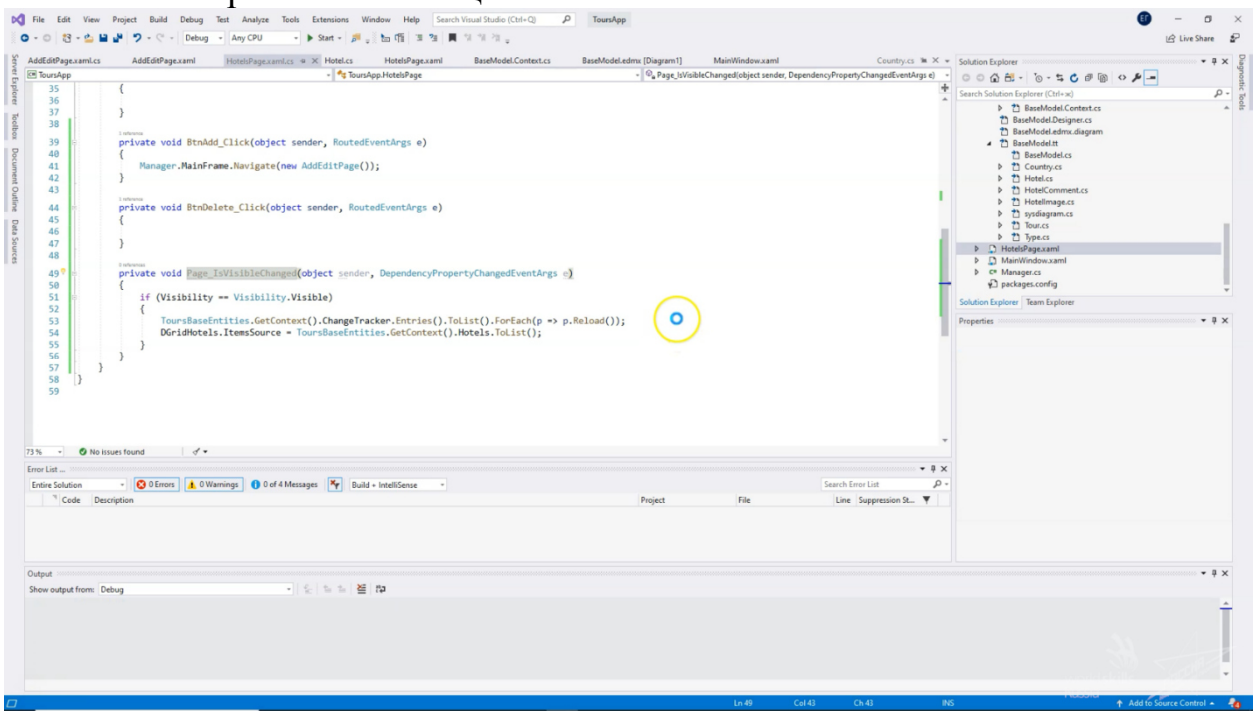


7. Теперь можно вернуться назад. При возврате на страницу со списком отелей, нам необходимо выводить актуальную информацию, обновляя список в таблице. Для этого мы будем использовать событие у страницы

IsVisibleChange. Оно срабатывает каждый раз, когда страница отображается, либо скрывается



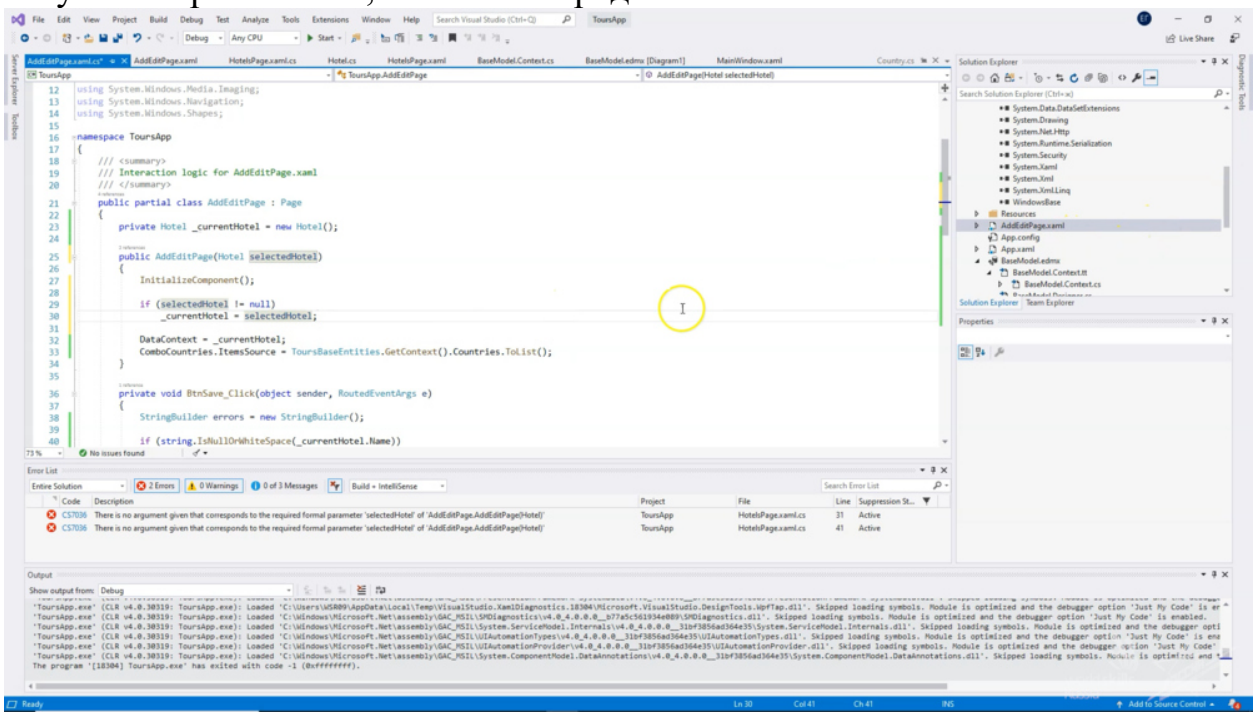
С помощью F12 переходим в код. Если видимость страницы isVisible, мы будем обращаться к контексту с помощью свойства ChangeTracker ко всем сущностям, которые есть. И для каждой из них будем выполнять метод перезагрузки и вывода актуальных данных. После этого таблицу DGridHotels присвоим таблице «список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



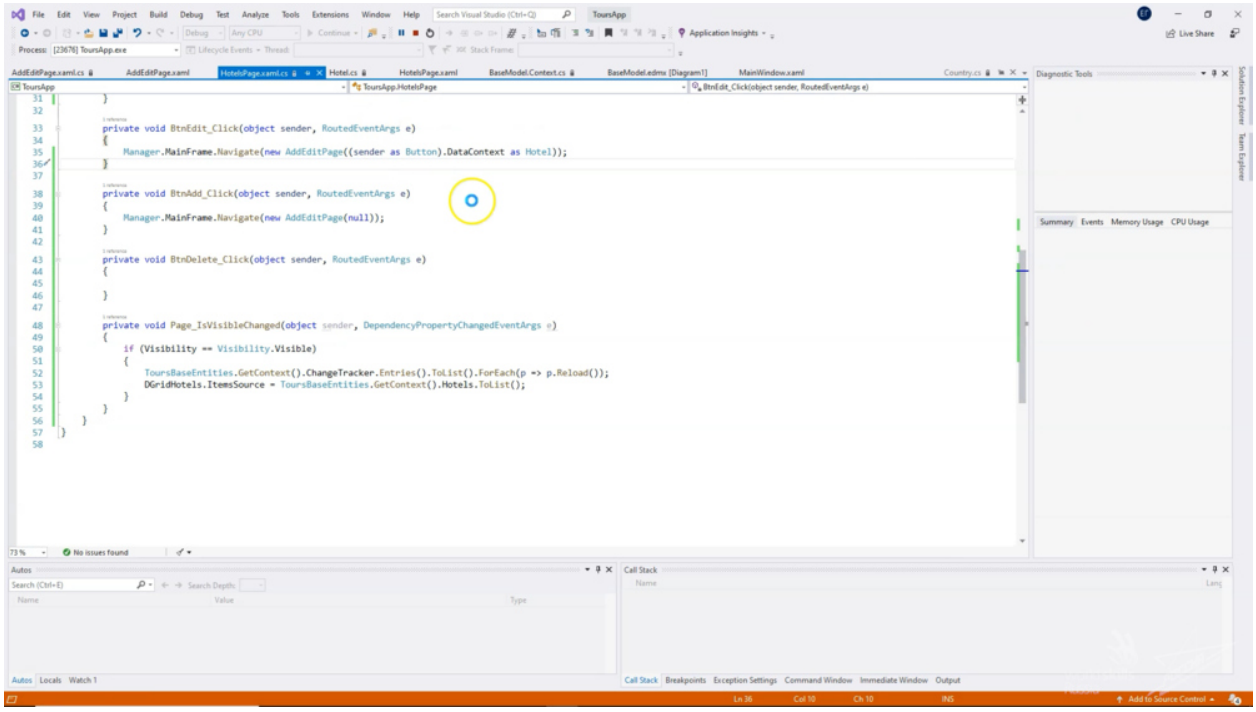
8. Запускаем программу и проверяем функцию добавления данных

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.

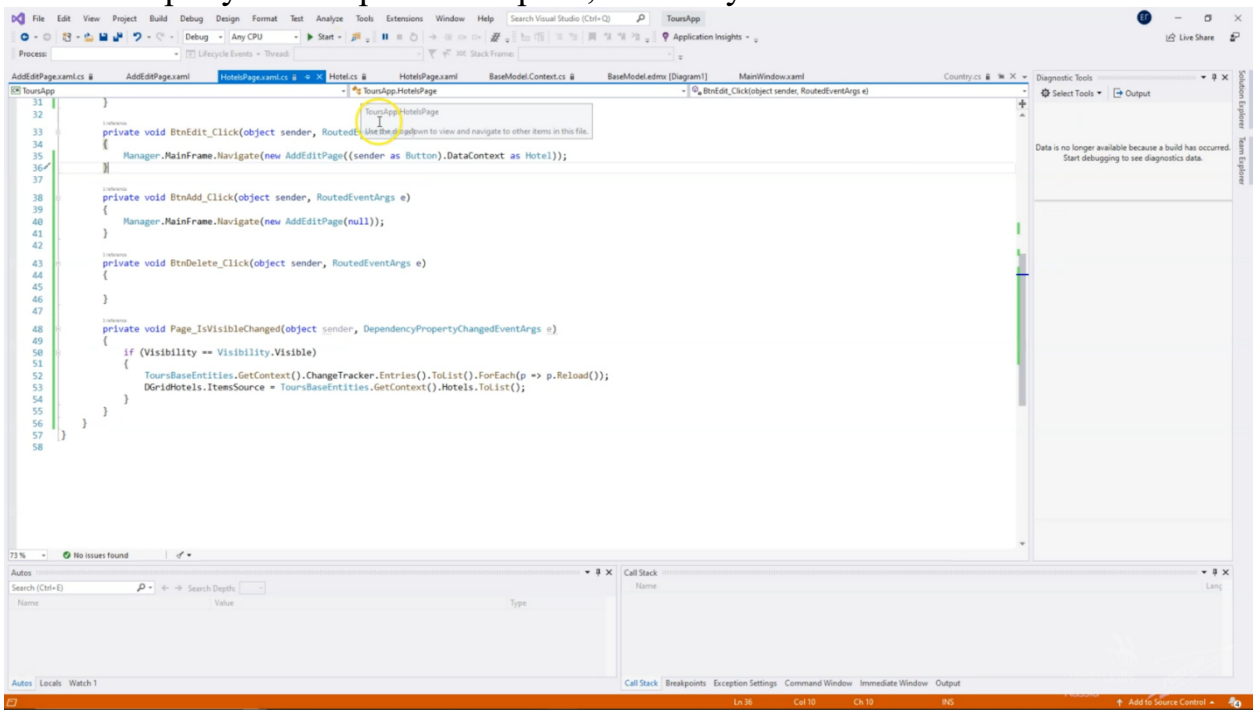
В первую очередь добавим параметр нашей странице AddEditPage. В нее мы будем передавать экземпляр выбранного отеля и, в случае если он не пустой, присваивать нашему полю CurrentHotel. Мы не можем сейчас запустить приложение, т. к. возник ряд ошибок



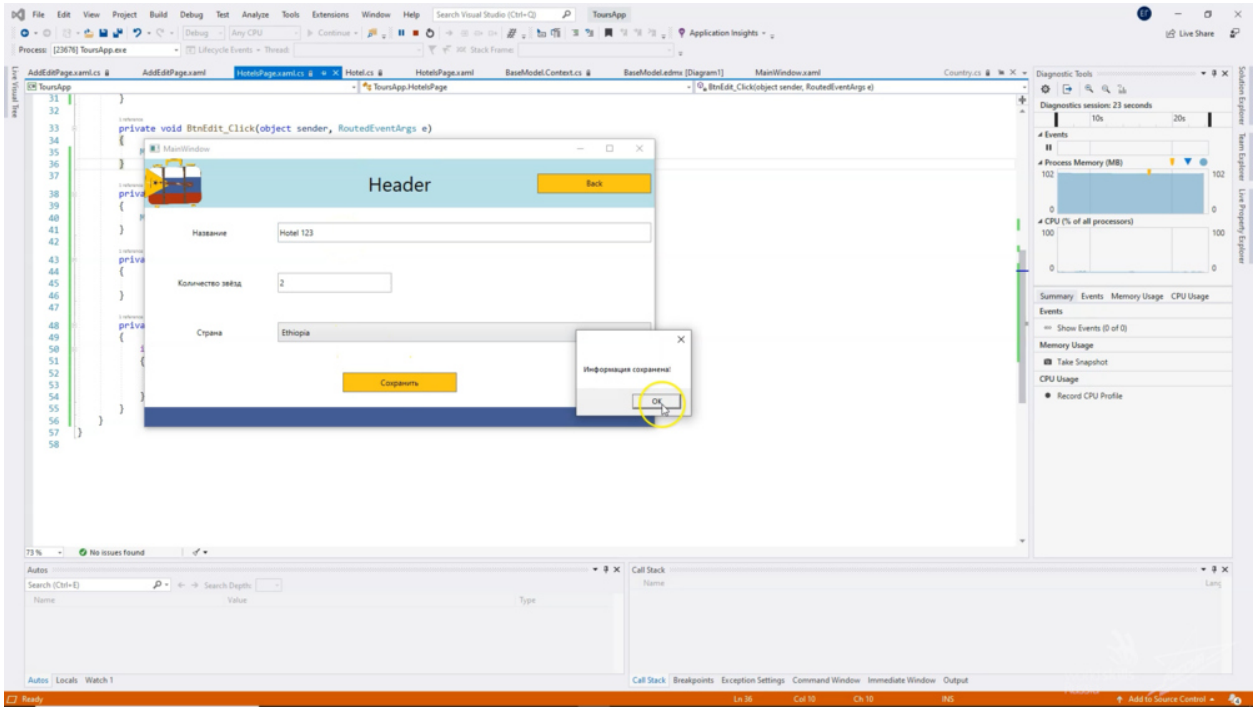
Вызов страницы AddEditPage теперь требует какого-то аргумента. В случае, если мы будем делать добавление, мы просто пропишем null (отправим пустой экземпляр). При этом для редактирования BtnEdit мы уже будем передавать экземпляр, прописав для этого код. Вместо null будем обращаться к кнопке, на которую нажали, получать ее контекст и знать, что это — отель



Попробуем теперь посмотреть, что получилось



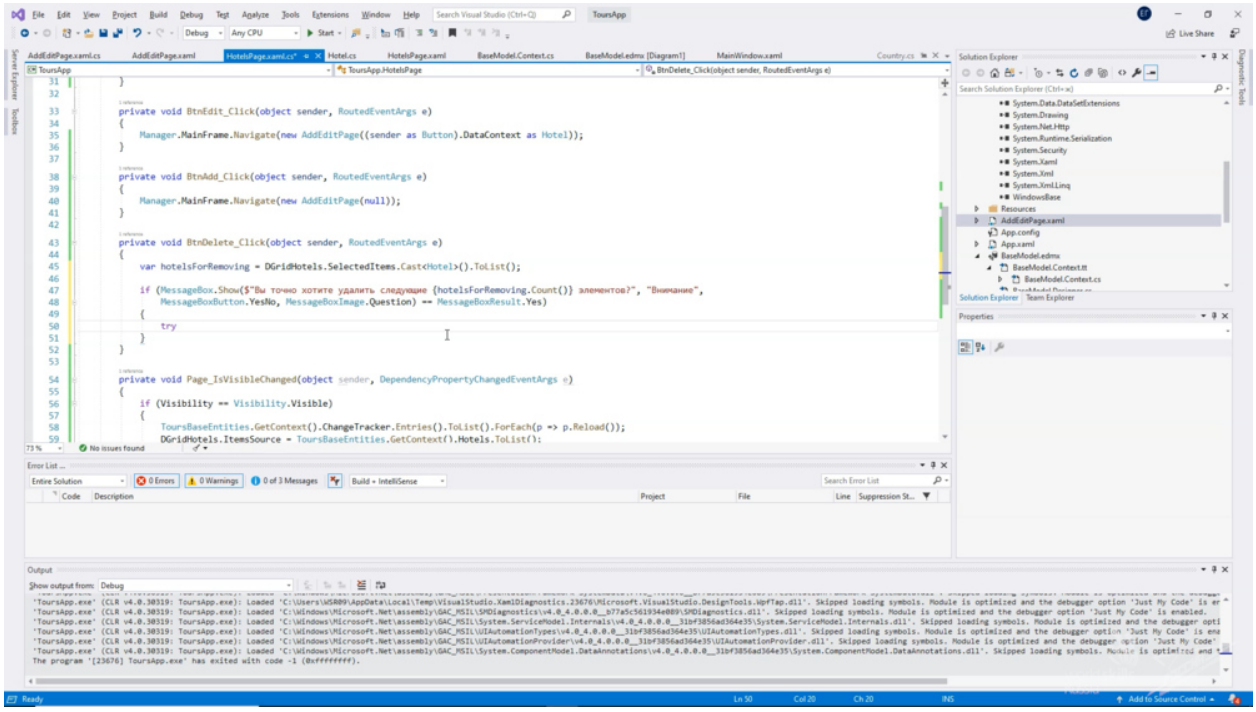
Те данные, которые мы хотим отредактировать, автоматически привязались к этим элементам управления и отображаются корректно. В случае, если мы отредактируем какое-то поле — нажимаем на кнопку Сохранить. Информация будет обновлена



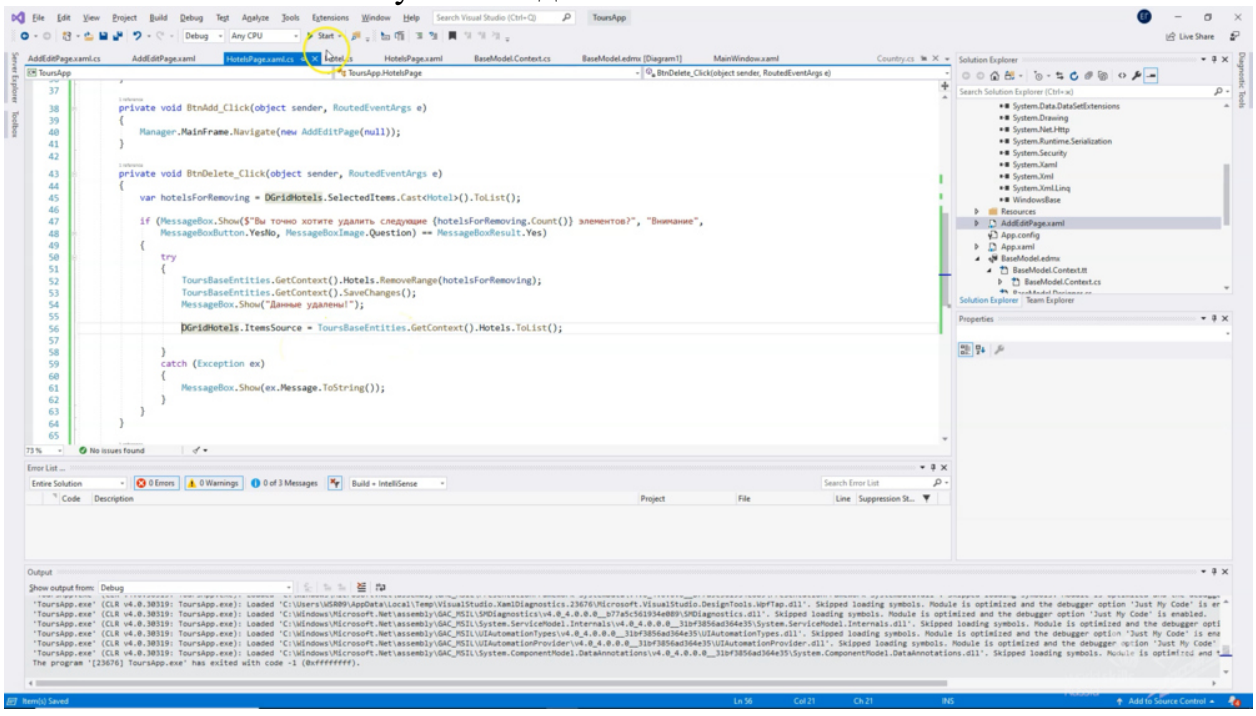
Реализация функции удаления

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.

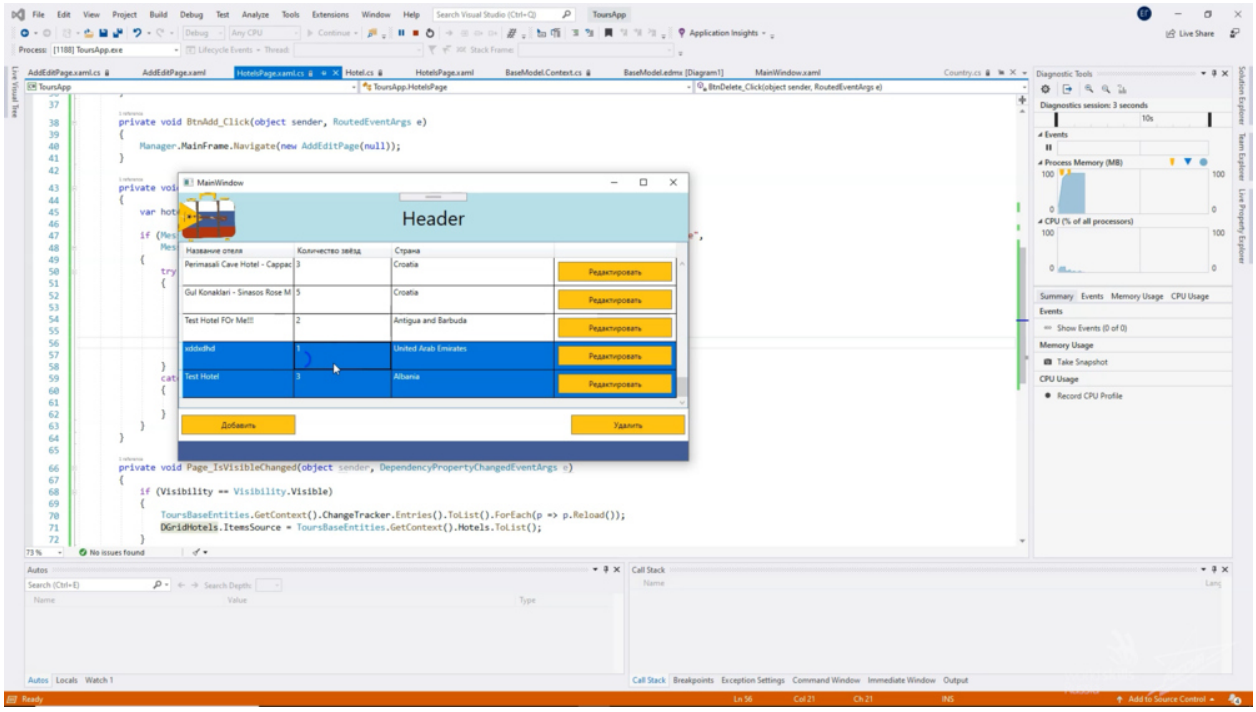
Итак, получаем список отелей для удаления, обратившись к таблице с отелями. Выбираем все элементы, которые мы выделили, преобразуем их в список отелей. И затем, в сообщении, будем спрашивать пользователя: «Вы точно хотите удалить следующие hotelsForRemoving.Count() элементов?». Укажем здесь заголовок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



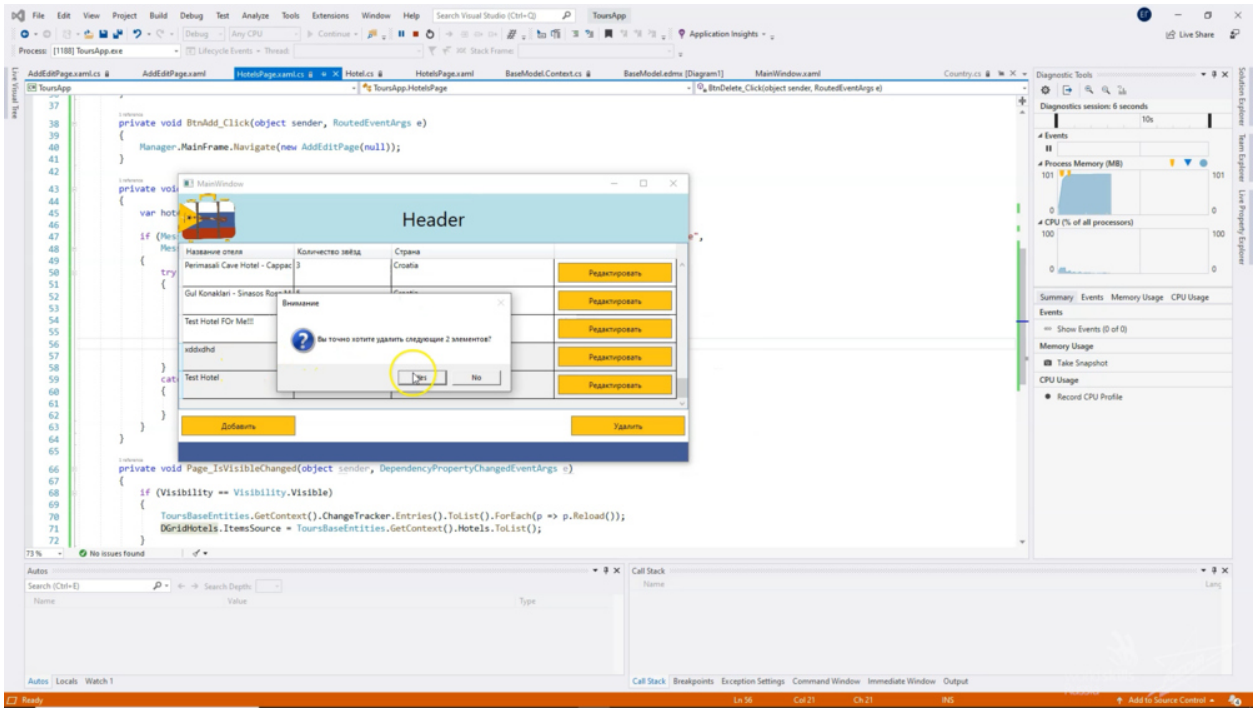
Если результатом диалога от пользователя было нажатие на кнопку «yes», то мы будем выполнять удаление. Для этого обратимся к модели данных, используя блок TryCatch. Получив контекст, попробуем с помощью метода RemoveRange удалить все полученные выделенные отели. В случае, если все будет хорошо, отобразим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

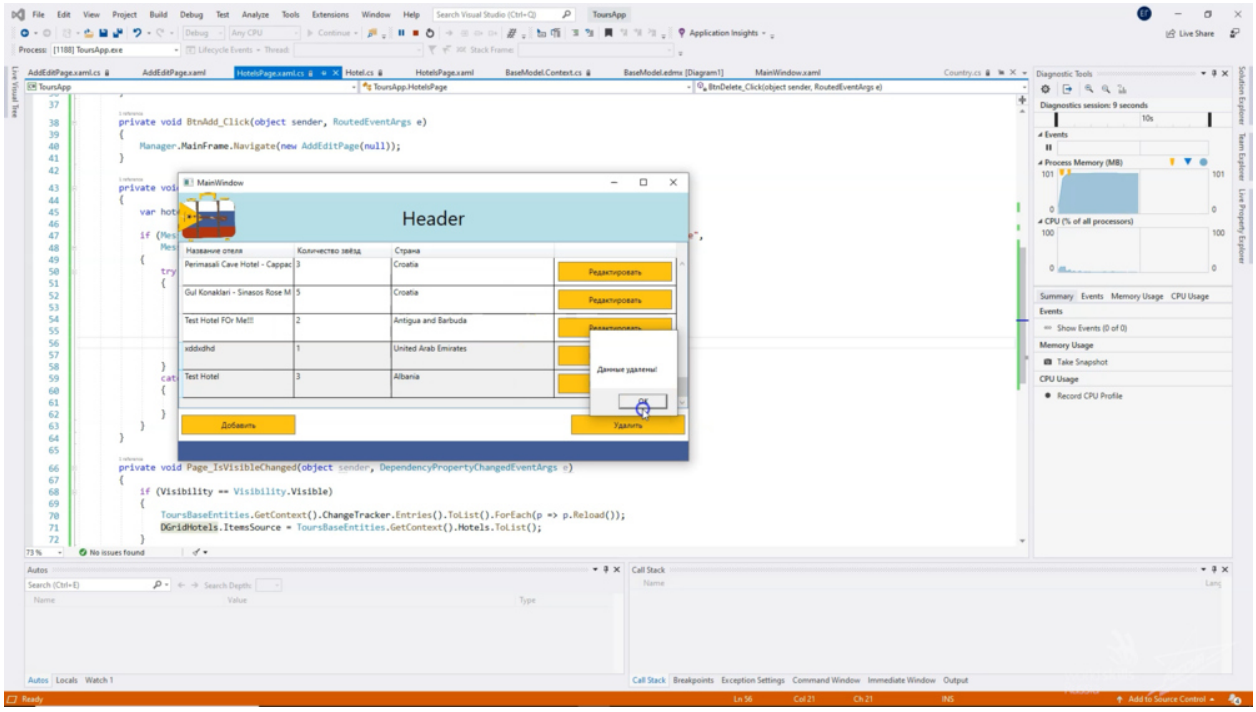


Сохраним и проверим работу приложения. Выделим два отеля

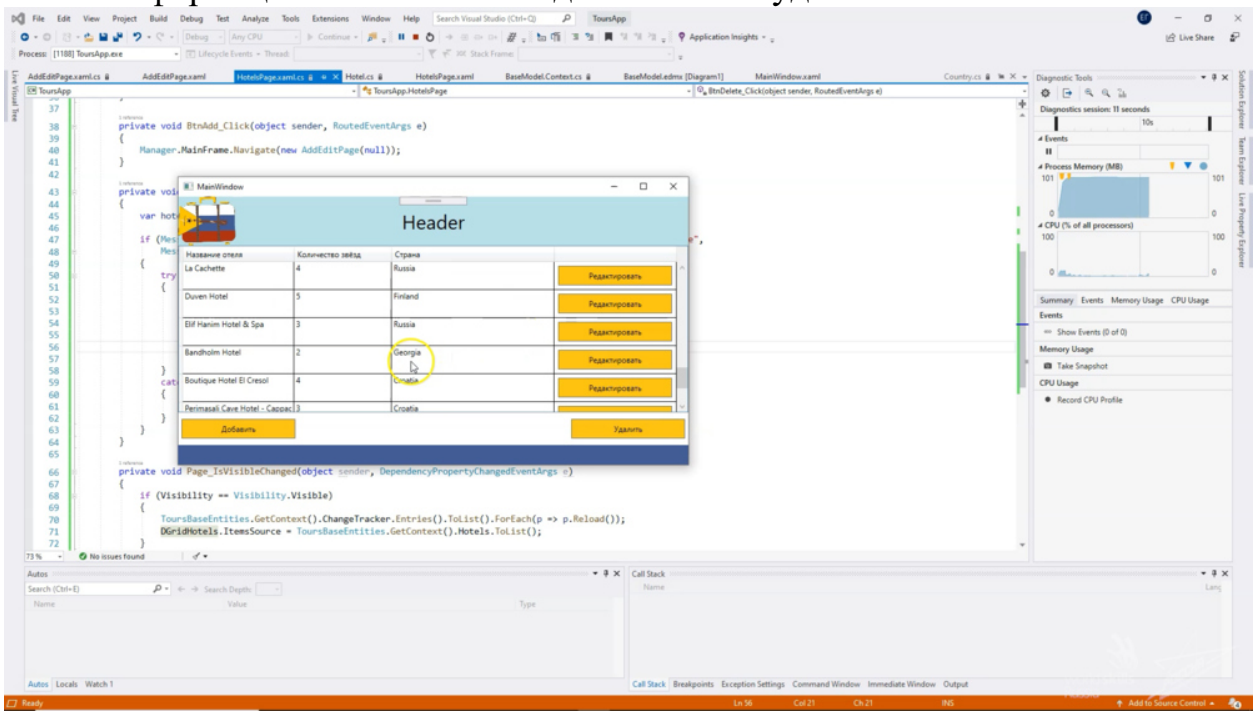


Нажмем удалить — точно хотим удалить следующие 2 элемента





Информация обновилась — данные были удалены



Интерактивное задание https://nationalteam.worldskills.ru/skills/rabota-s-bazoy-dannykh-v-prilozhenii-chtenie-dobavlenie-redaktirovanie-udalenie-dannykh-chast-2/

Практическая работа №9. Работа с неструктурированными данными, обработка и импорт «сырых» данных в базу данных

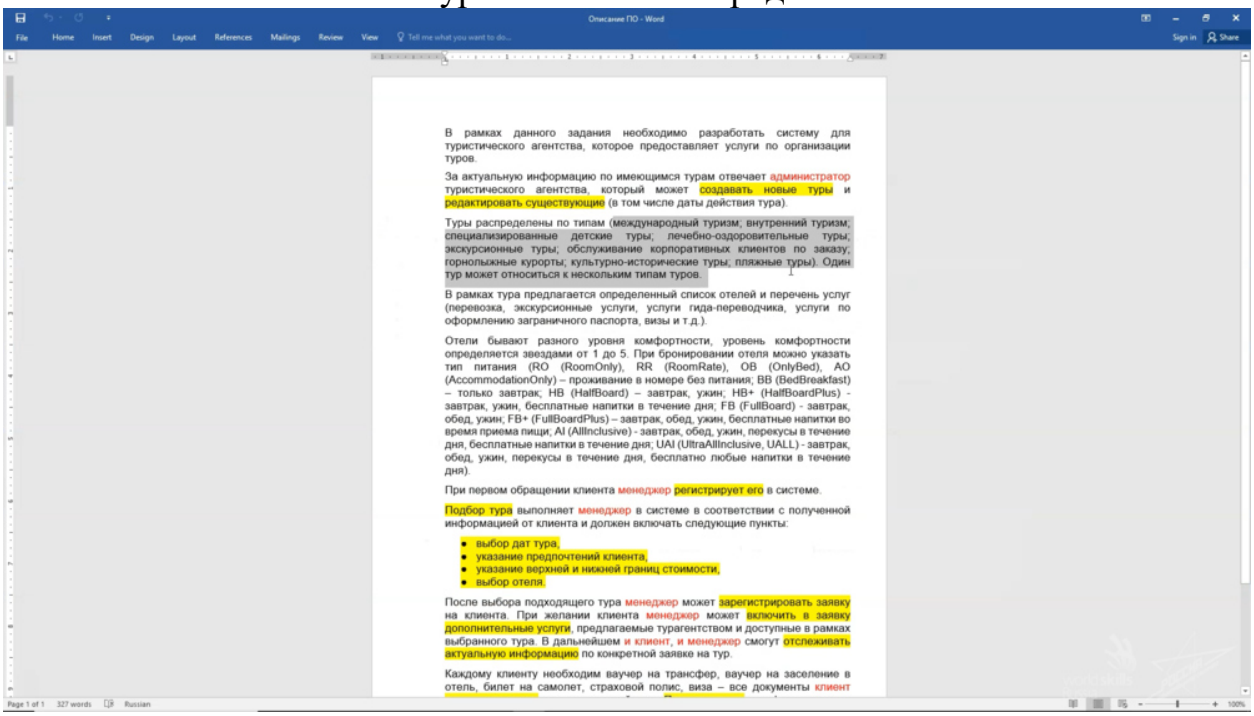
Краткие теоретические сведения

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.

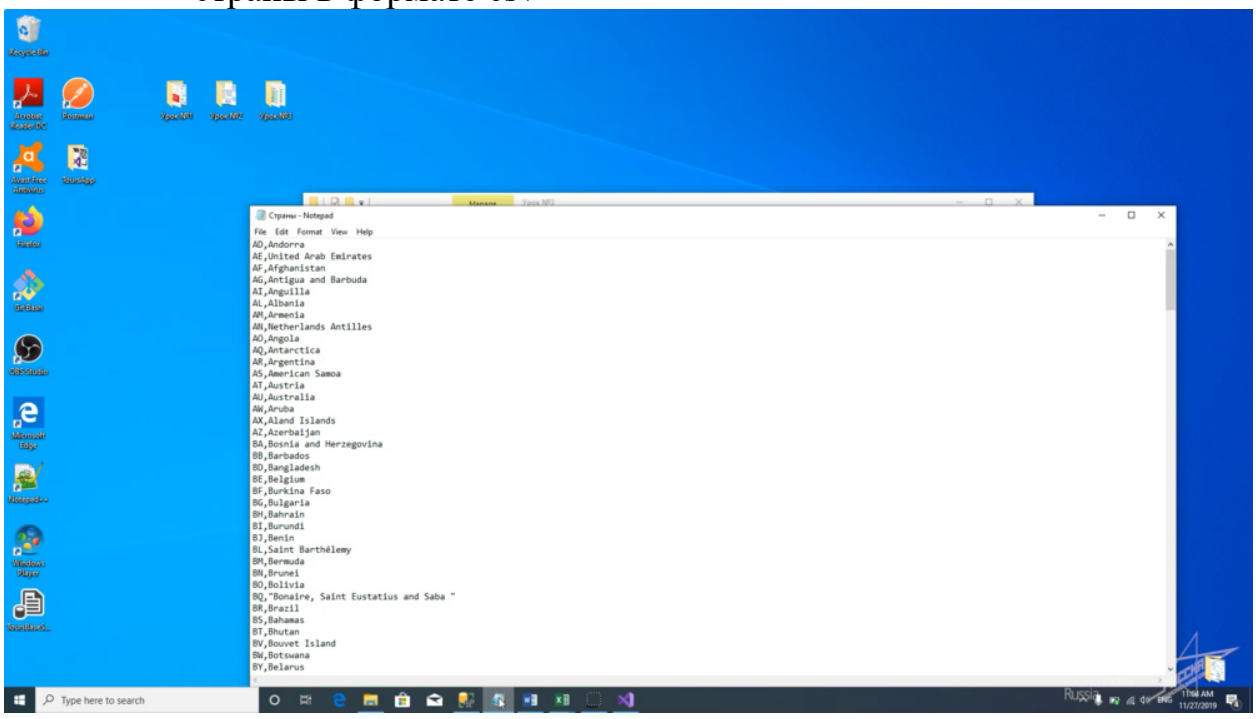
Подбор тура выполняет менеджер в системе в соответствии с полученной информацией от клиента и должен включать следующие пункты:

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

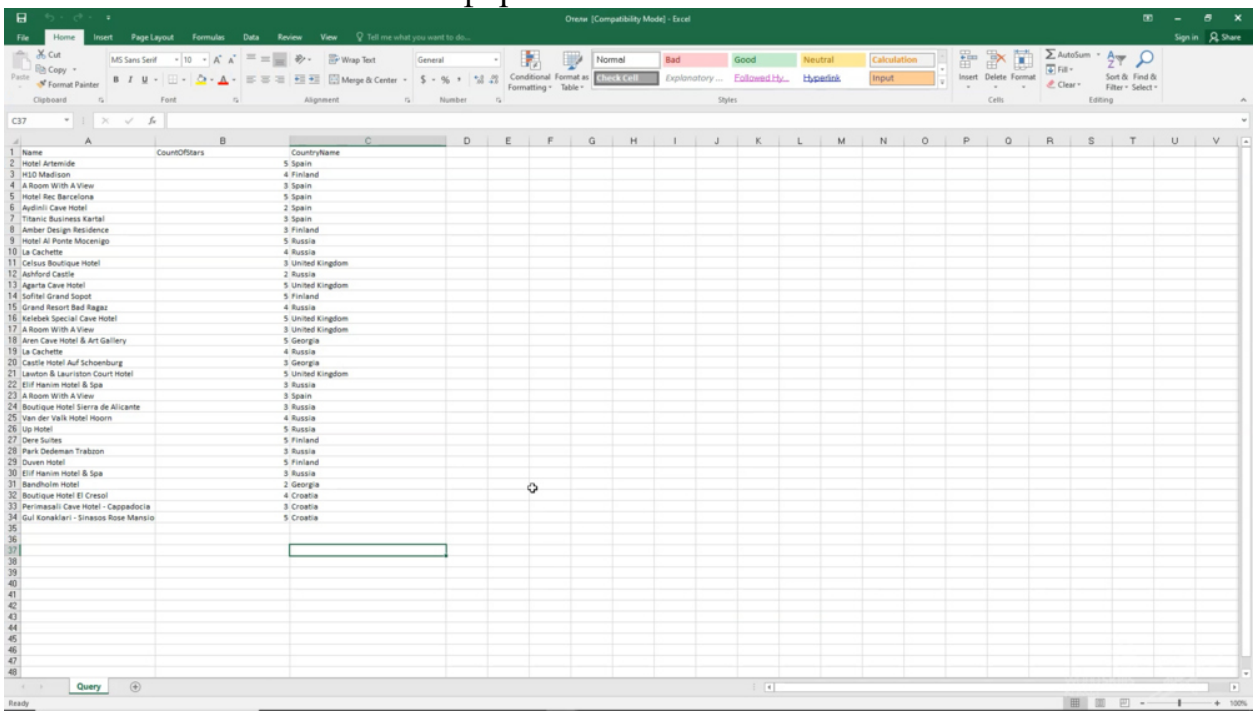
После выбора подходящего тура менеджер может зарегистрировать заявку на клиента. При желании клиента менеджер может включить в заявку дополнительные услуги, предлагаемые турагентством и доступные в рамках выбранного тура. В дальнейшем и клиент, и менеджер смогут отслеживать актуальную информацию по конкретной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

Каждому клиенту необходим ваучер на трансфер, ваучер на заселение в отель, билет на самолет, страховой полис, виза – все документы клиент

- страны в формате csv

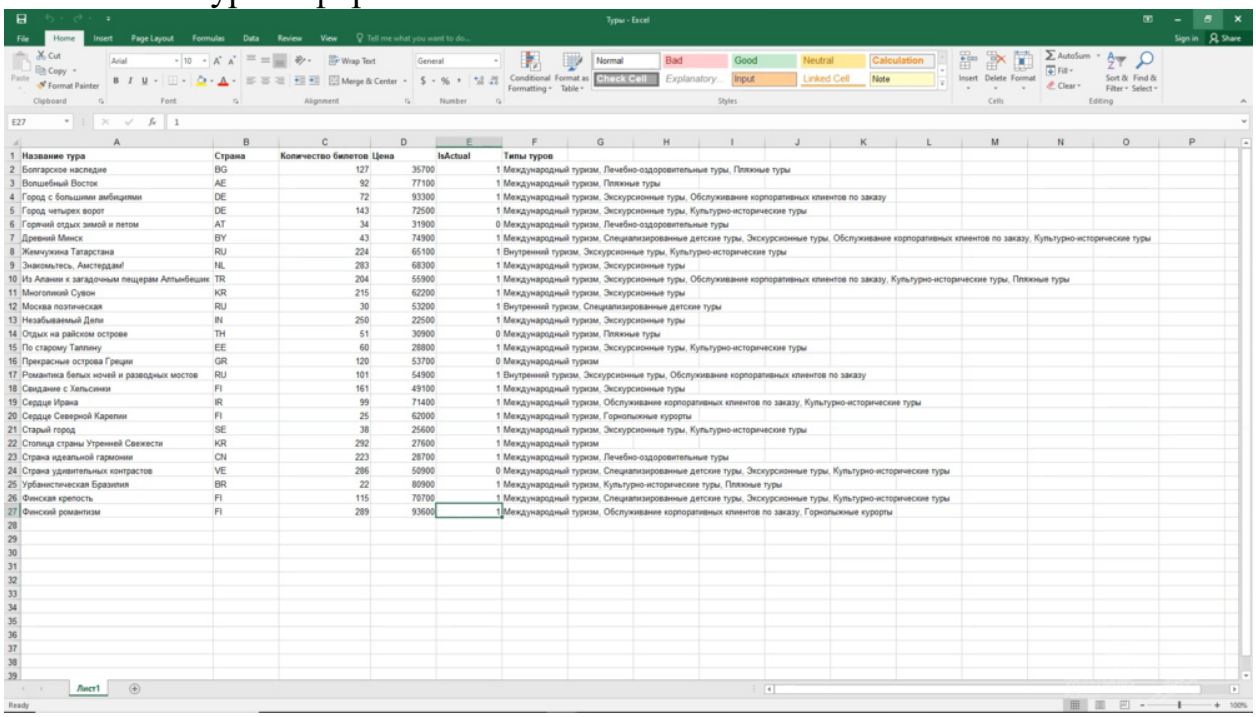


- список отелей в формате xls



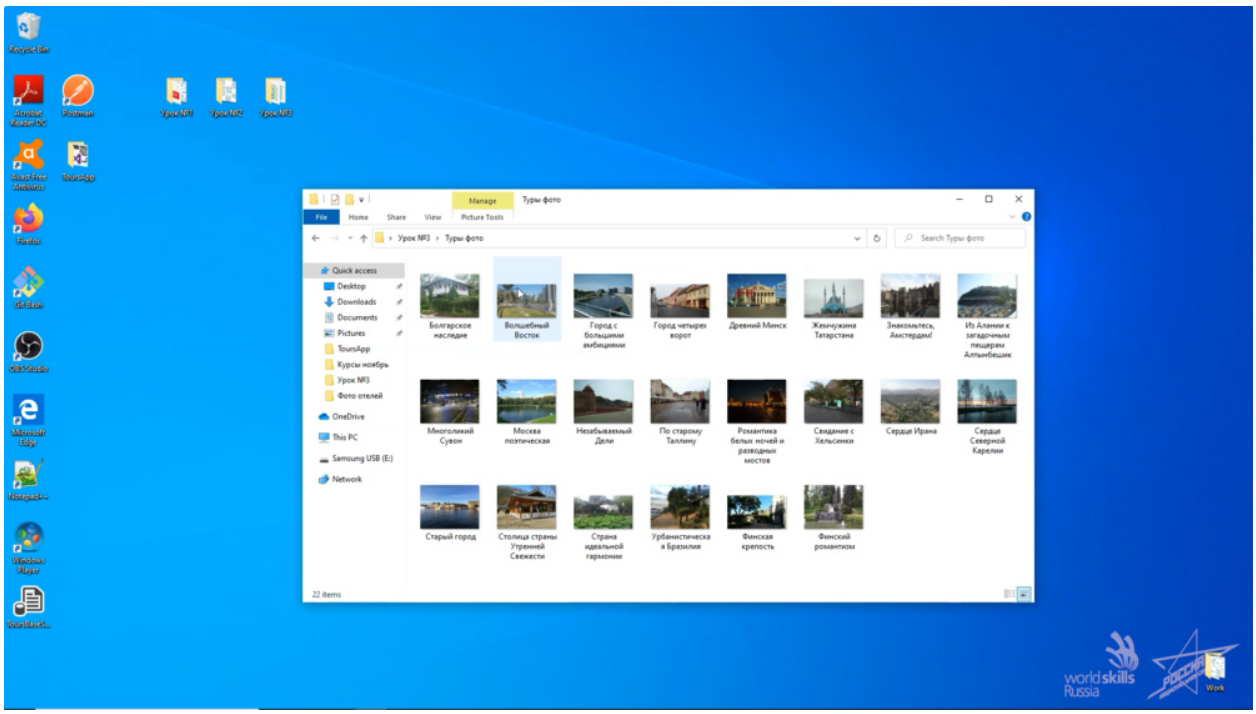
Name	CountOfStars	CountryName
Hotel Artemide	5	Spain
H10 Madison	4	Finland
A Room With A View	3	Spain
Hotel Rec Barcelona	5	Spain
Aydinli Cave Hotel	2	Spain
Titanic Business Kartal	3	Spain
Amber Design Residence	3	Finland
Hotel Al Ponte Mocenigo	5	Russia
La Cachette	4	Russia
Celsus Boutique Hotel	3	United Kingdom
Ashford Castle	2	Russia
Agarta Cave Hotel	5	United Kingdom
Sofitel Grand Sopot	5	Finland
Grand Resort Bad Ragaz	4	Russia
Kelebek Special Cave Hotel	5	United Kingdom
A Room With A View	3	United Kingdom
Aren Cave Hotel & Art Gallery	5	Georgia
La Cachette	4	Russia
Castle Hotel Auf Schoenburg	3	Georgia
Lawton & Lauriston Court Hotel	5	United Kingdom
Elif Hanim Hotel & Spa	3	Russia
A Room With A View	3	Spain
Boutique Hotel Sierra de Alicante	3	Russia
Van der Valk Hotel Hoorn	4	Russia
Up Hotel	5	Russia
Dere Suites	5	Finland
Park Dedeman Trabzon	3	Russia
Duven Hotel	5	Finland
Elif Hanim Hotel & Spa	3	Russia
Bandholm Hotel	2	Georgia
Boutique Hotel El Cresol	4	Croatia
Perimasali Cave Hotel - Cappadocia	3	Croatia
Gul Konaklari - Sinasos Rose Mansio	5	Croatia

- туры в формате xlsx



Название тура	Страна	Количество билетов	Цена	IsActual	Типы туров
Болгарское наследие	BG	127	35700	1	Международный туризм, Лечебно-оздоровительные туры, Пляжные туры
Волшебный Восток	AE	92	77100	1	Международный туризм, Пляжные туры
Город с большими амбициями	DE	72	93300	1	Международный туризм, Экскурсионные туры, Обслуживание корпоративных клиентов по заказу
Город четырех ворот	DE	143	72500	1	Международный туризм, Экскурсионные туры, Культурно-исторические туры
Горячий отдых зимой и летом	AT	34	31900	0	Международный туризм, Лечебно-оздоровительные туры
Древний Минск	BY	43	74900	1	Международный туризм, Специализированные детские туры, Экскурсионные туры, Обслуживание корпоративных клиентов по заказу, Культурно-исторические туры
Жемчужина Татарстана	RU	224	65100	1	Внутренний туризм, Экскурсионные туры, Культурно-исторические туры
Знакомьтесь, Амстердам!	NL	283	68300	1	Международный туризм, Экскурсионные туры
Из Алании к загадочным пещерам Аппынбещих	TR	204	55900	1	Международный туризм, Экскурсионные туры, Обслуживание корпоративных клиентов по заказу, Культурно-исторические туры, Пляжные туры
Многолики Сувон	KR	215	62200	1	Международный туризм, Экскурсионные туры
Москва поэтическая	RU	30	53200	1	Внутренний туризм, Специализированные детские туры
Незабываемый Дели	IN	250	22500	1	Международный туризм, Экскурсионные туры
Отдых на райском острове	TH	51	30900	0	Международный туризм, Пляжные туры
По старому Таллину	EE	60	28800	1	Международный туризм, Экскурсионные туры, Культурно-исторические туры
Прекрасные острова Греции	GR	120	53700	0	Международный туризм
Романтика белых ночей и разводных мостов	RU	101	54900	1	Внутренний туризм, Экскурсионные туры, Обслуживание корпоративных клиентов по заказу
Свидание с Хельсинки	FI	161	49100	1	Международный туризм, Экскурсионные туры
Сердце Ирана	IR	99	71400	1	Международный туризм, Обслуживание корпоративных клиентов по заказу, Культурно-исторические туры
Сердце Северной Карелии	FI	25	62000	1	Международный туризм, Горнолыжные курорты
Старый город	SE	38	25600	1	Международный туризм, Экскурсионные туры, Культурно-исторические туры
Столица страны Утренней Свежести	KR	292	27600	1	Международный туризм
Страна идеальной гармонии	CN	223	28700	1	Международный туризм, Лечебно-оздоровительные туры
Страна удивительных контрастов	VE	286	50900	0	Международный туризм, Специализированные детские туры, Экскурсионные туры, Культурно-исторические туры
Урбанистическая Бразилия	BR	22	80900	1	Международный туризм, Культурно-исторические туры, Пляжные туры
Финская крепость	FI	115	70700	1	Международный туризм, Специализированные детские туры, Экскурсионные туры, Культурно-исторические туры
Финский романтизм	FI	289	93600	1	Международный туризм, Обслуживание корпоративных клиентов по заказу, Горнолыжные курорты

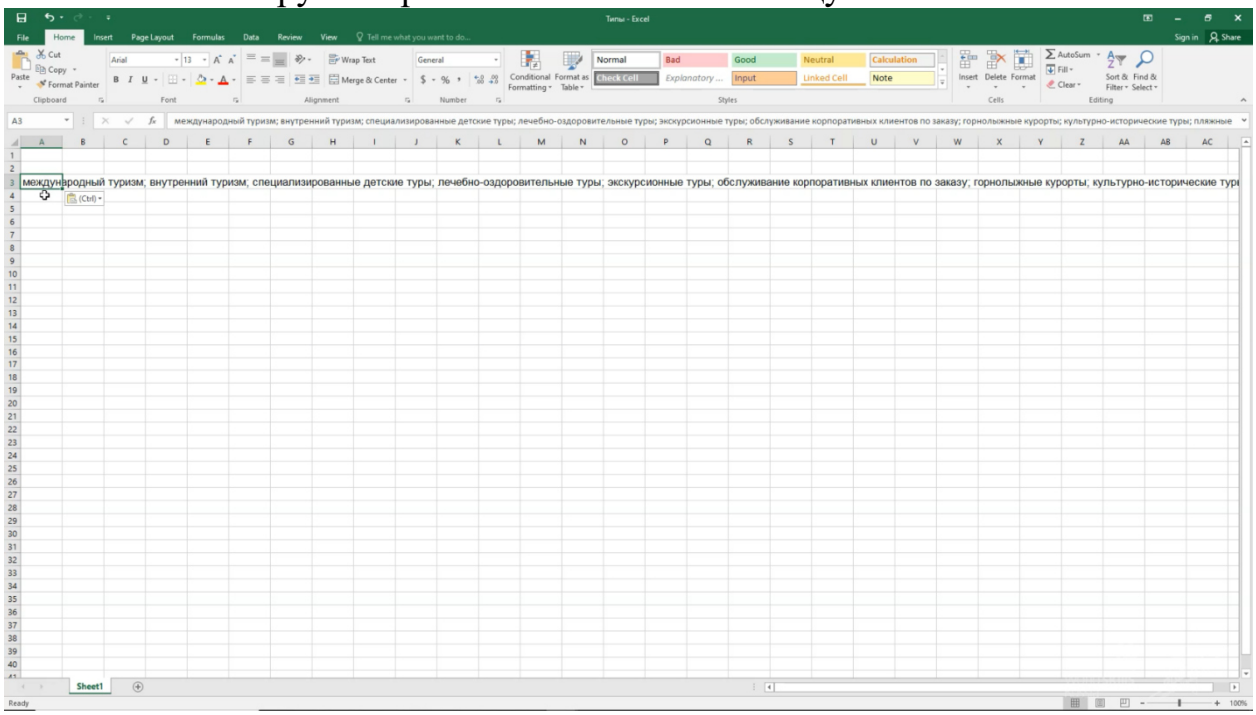
- список фотографий туров в виде изображений



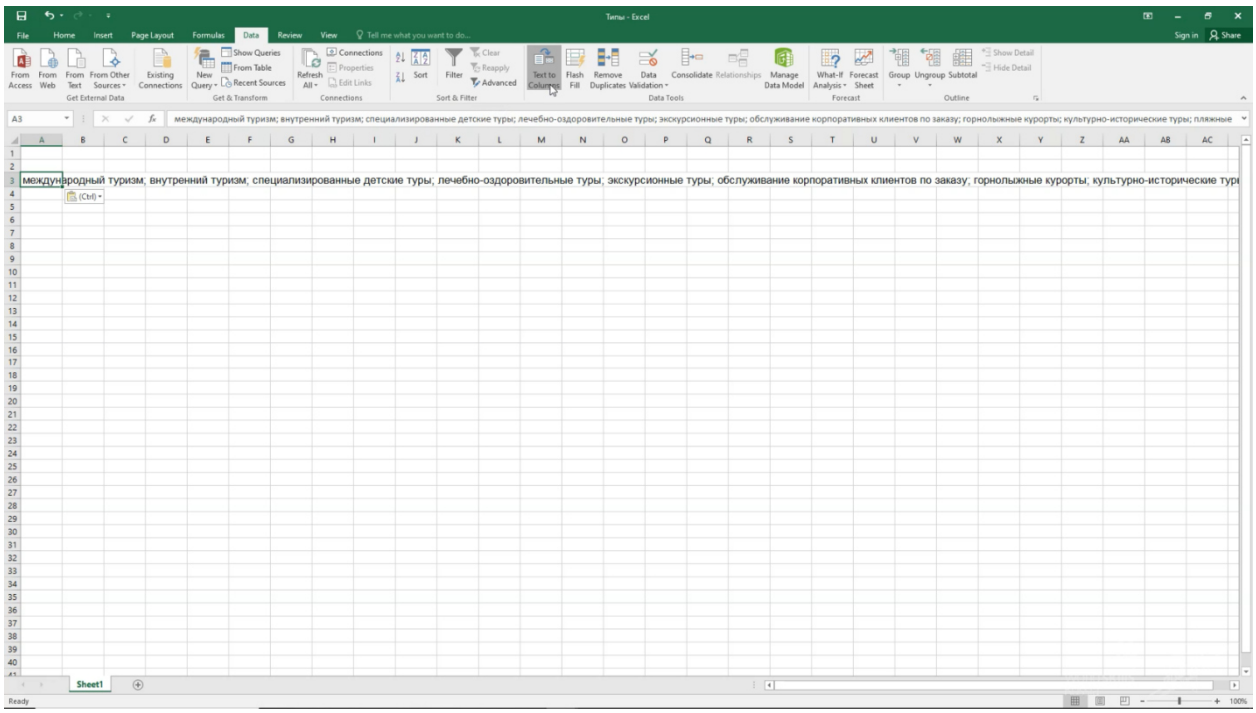
В этом уроке мы протестируем несколько разных способов импорта данных, предварительно почистив базу данных

Работа с текстовыми данными

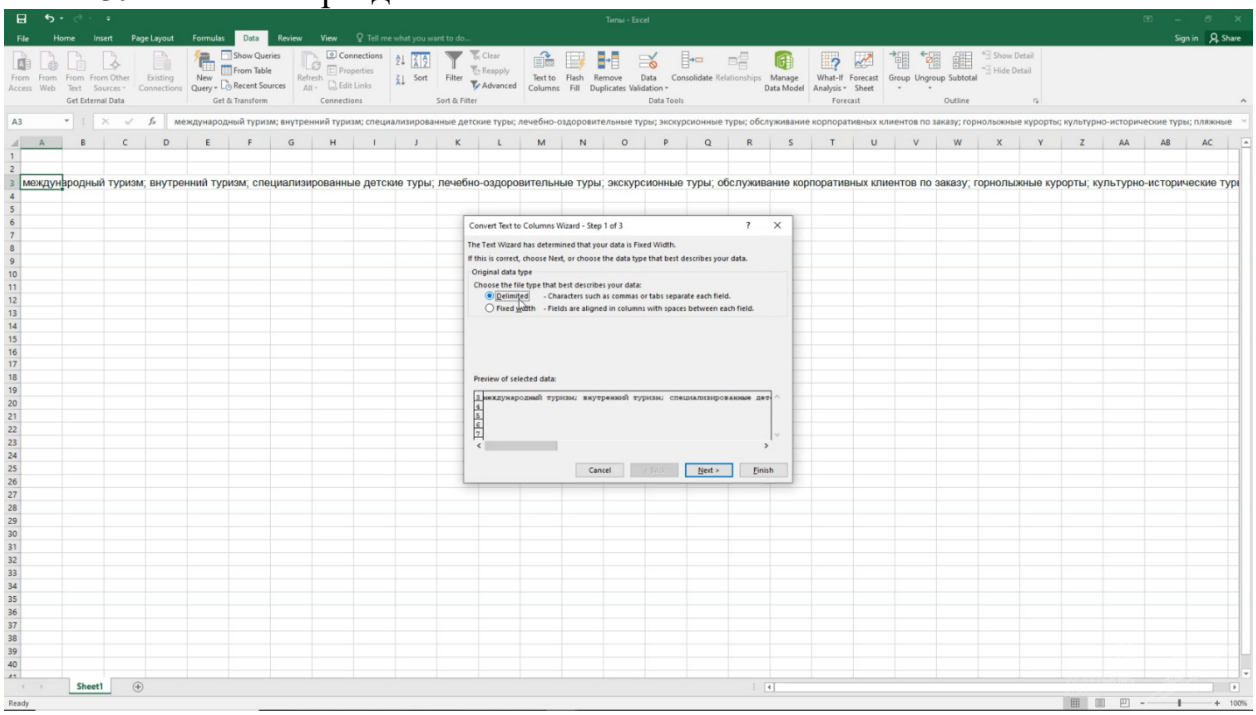
1. Копируем перечисление в Excel таблицу

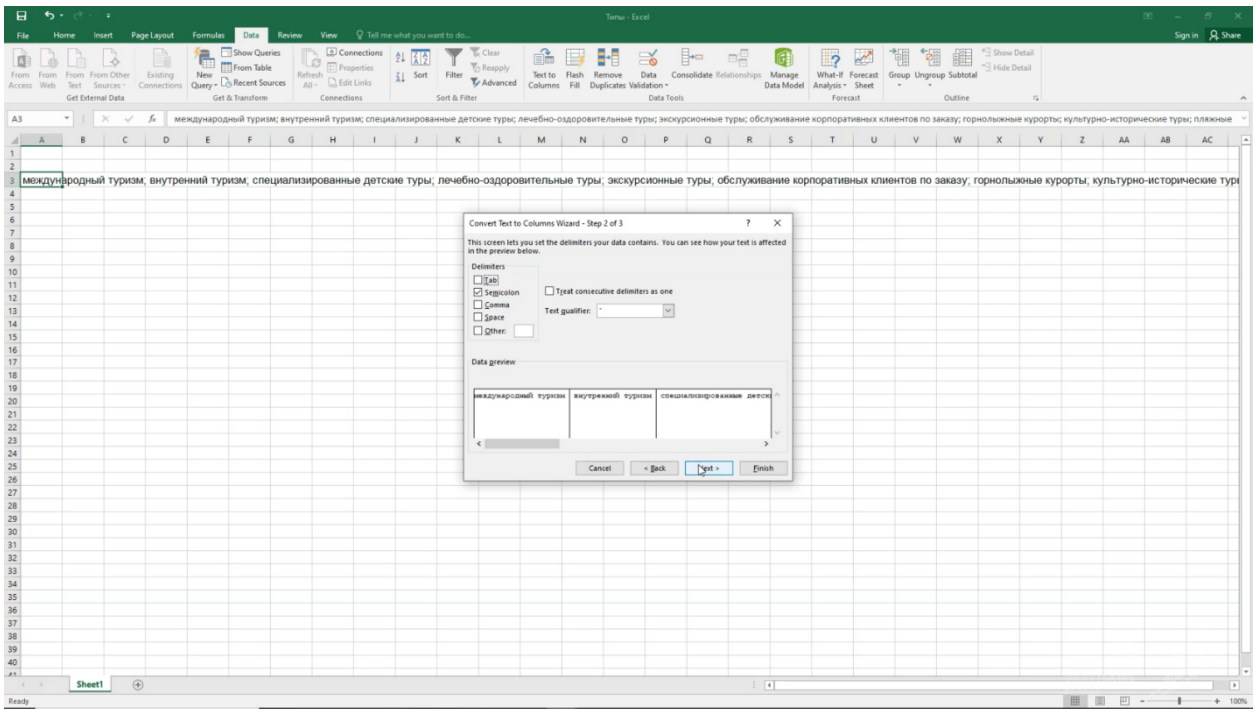


2. Разбиваем текст по столбцам с помощью функции Text to Columns на вкладке Data

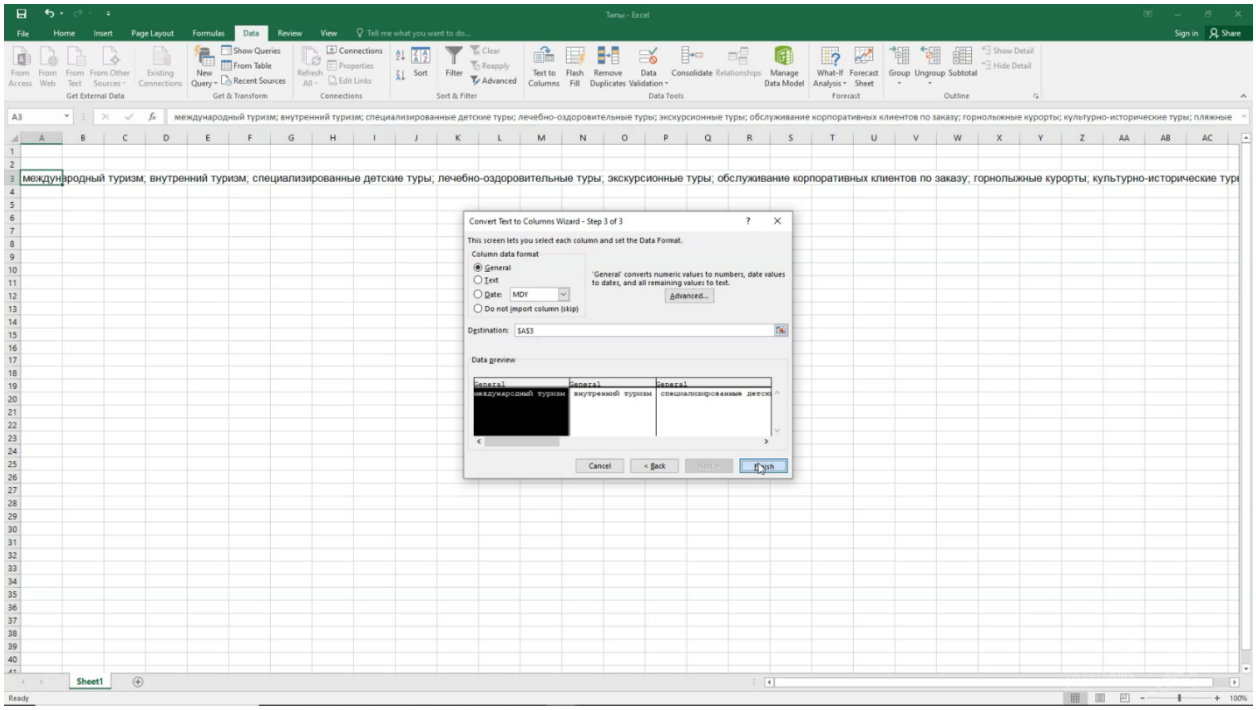


3. Указываем разделитель

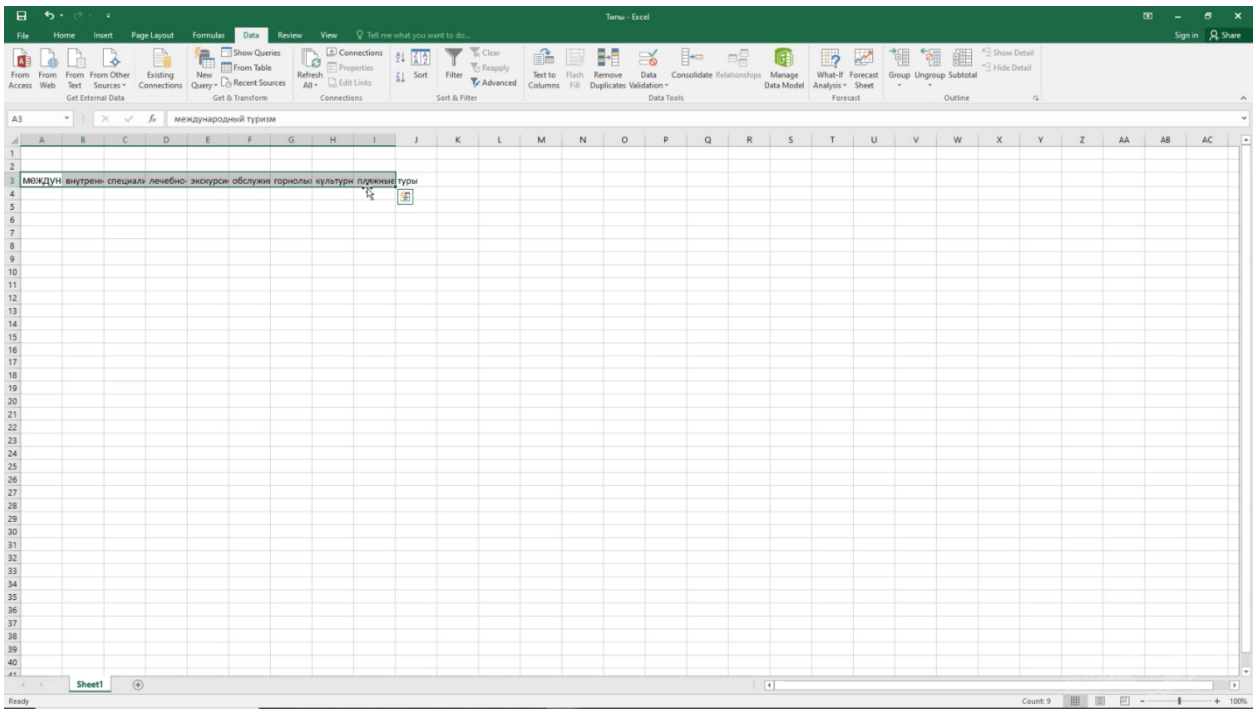




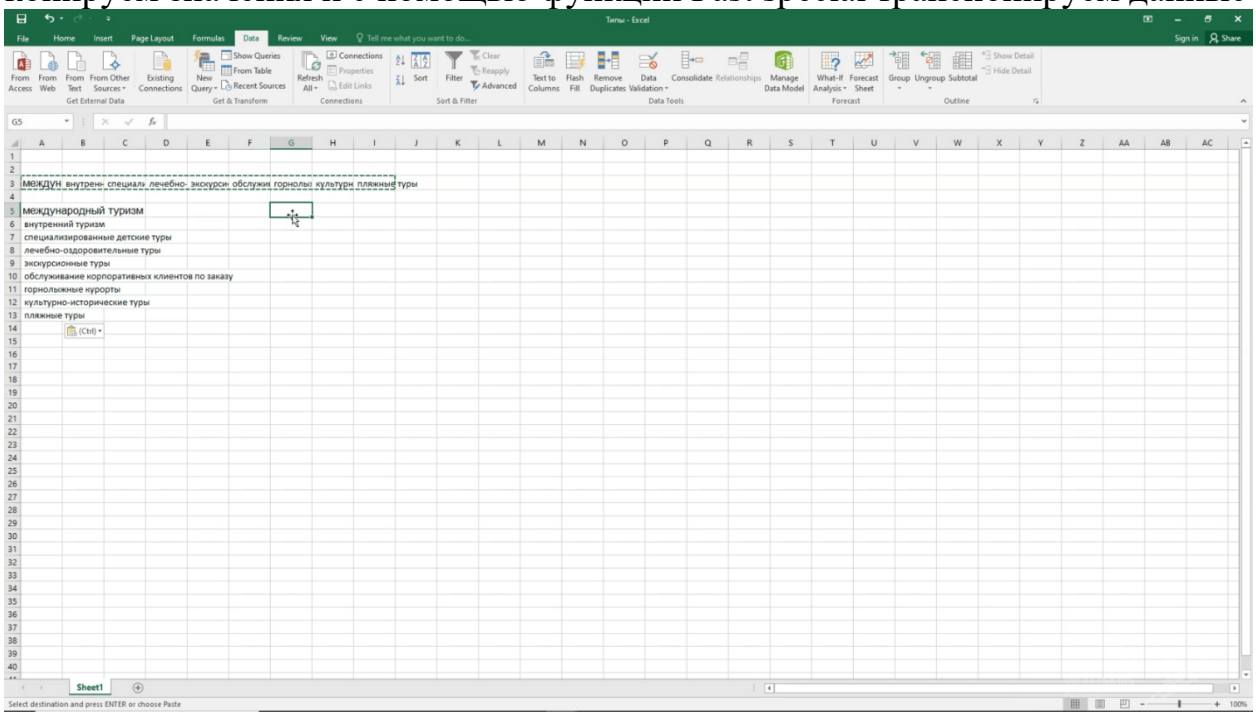
4. Finish



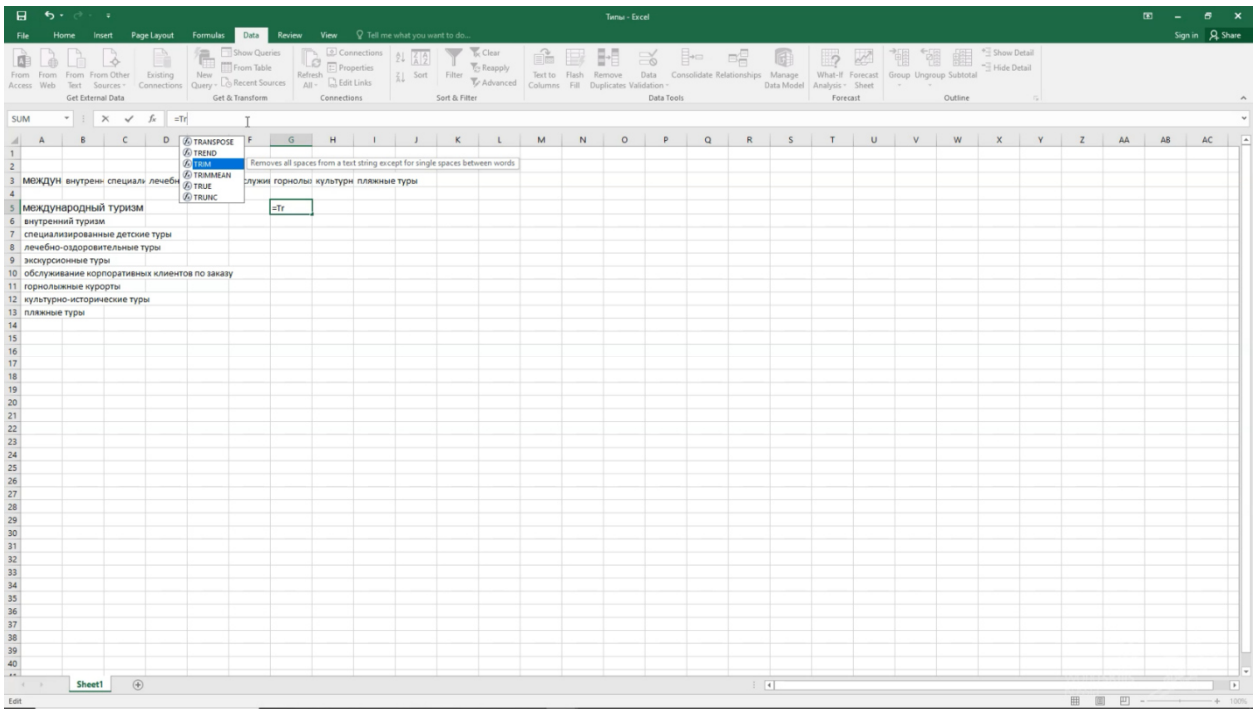
5. Получаем каждую запись в отдельной ячейке



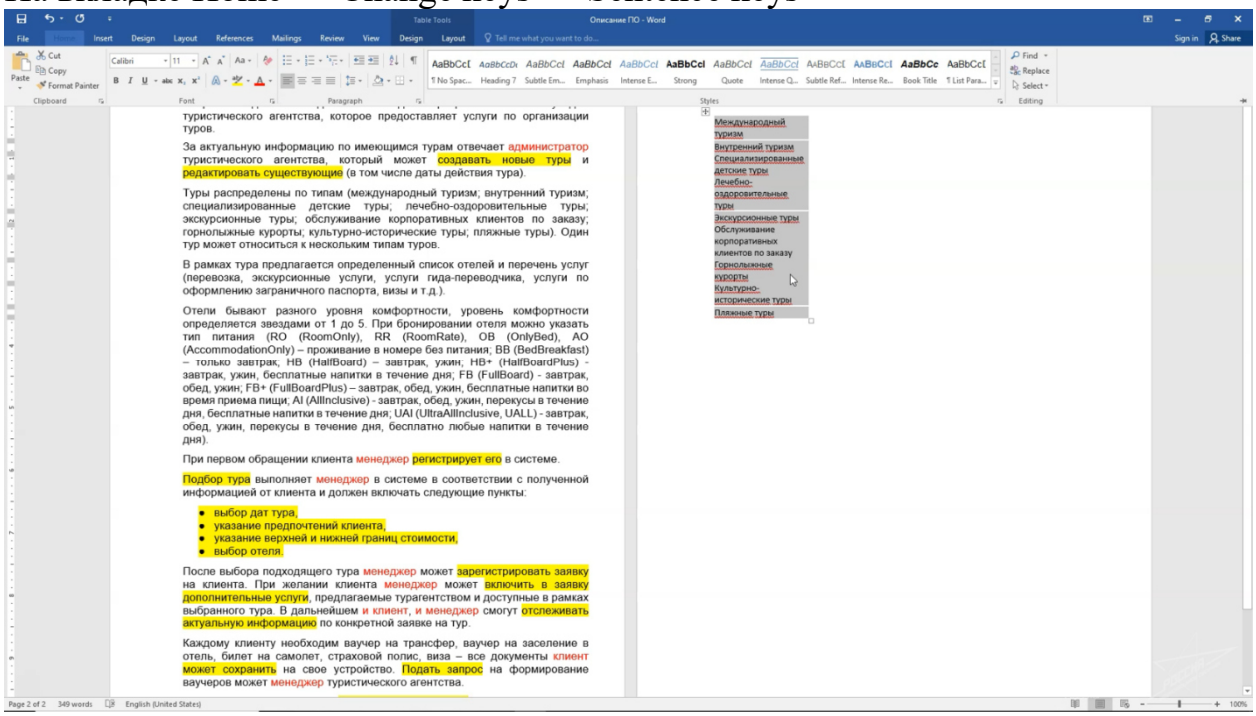
6. Теперь каждый тип списка изменяем на вертикальный. Для этого копируем значения и с помощью функции Past special транспонируем данные



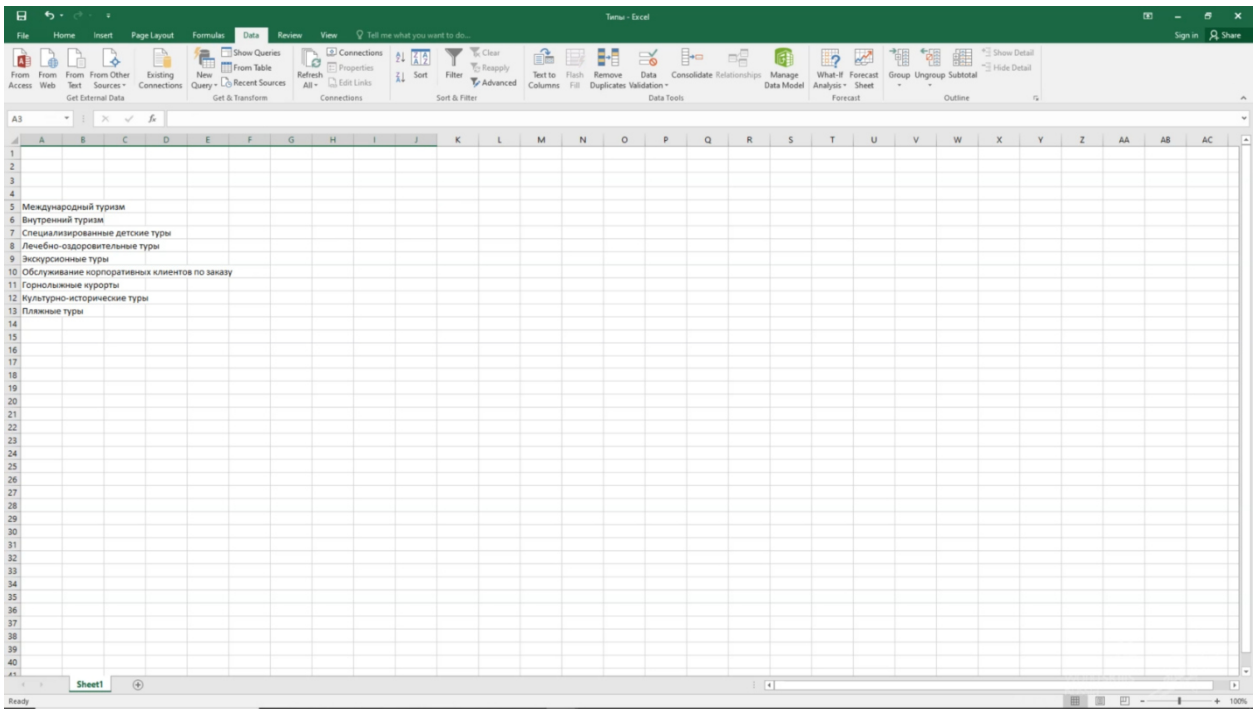
7. Далее было бы неплохо эти данные отформатировать. Например, избавиться от пробелов с помощью функции Trim()



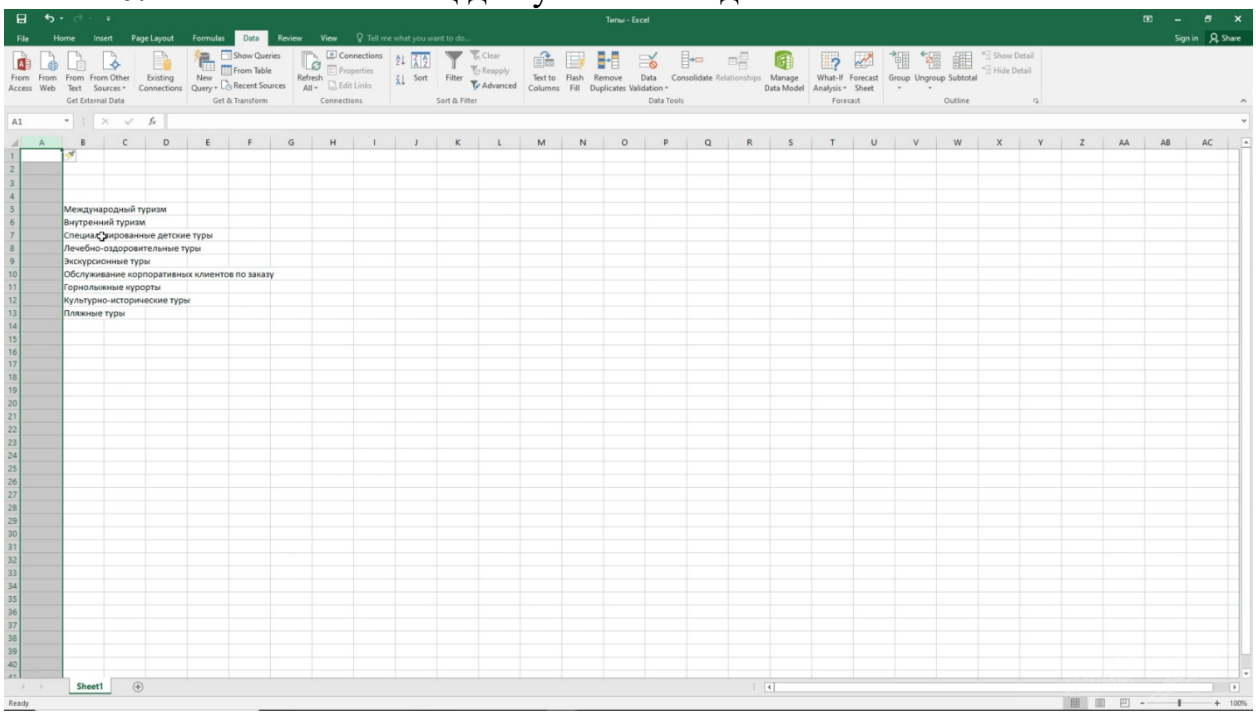
8. Отформатируем текст, чтобы название каждого типа начиналось с заглавной буквы. Для этого воспользуемся встроенными функциями Word. На вкладке Home — Change keys — Sentence keys



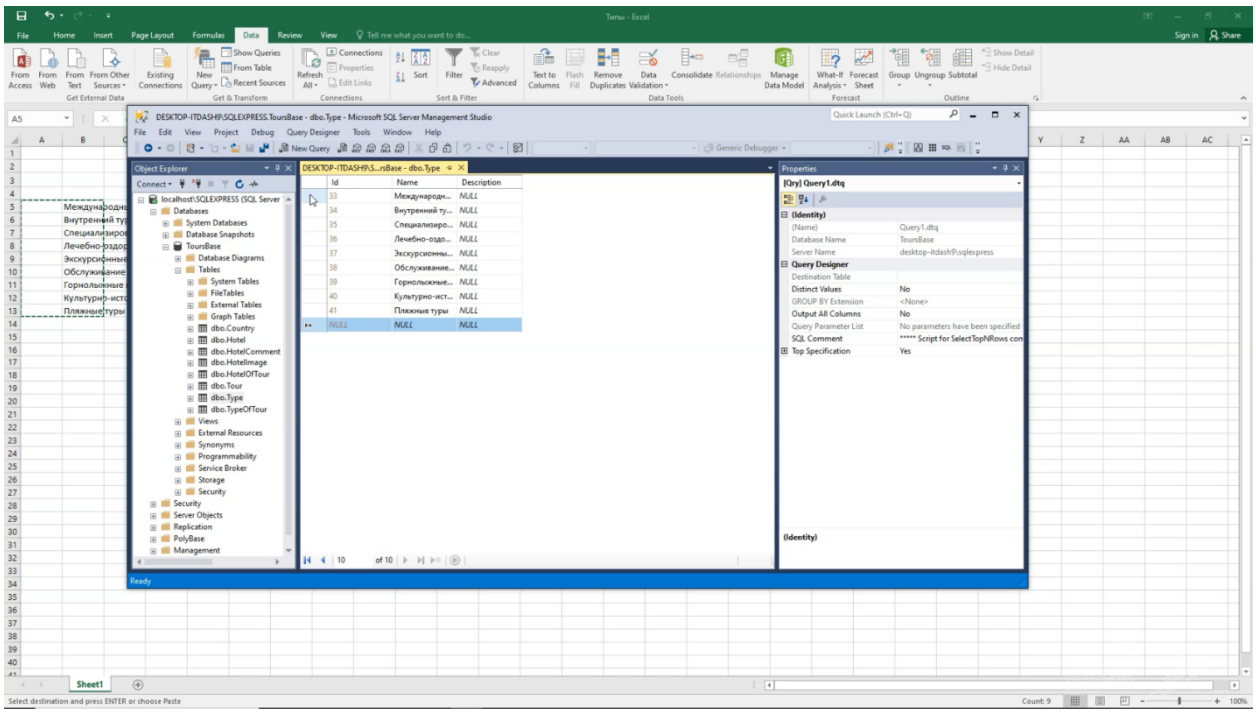
9. Выделяем сущности в описании заказчика



10. Вставляем столбец для указания кодов



11. Копируем нашу базу данных в таблицу Type

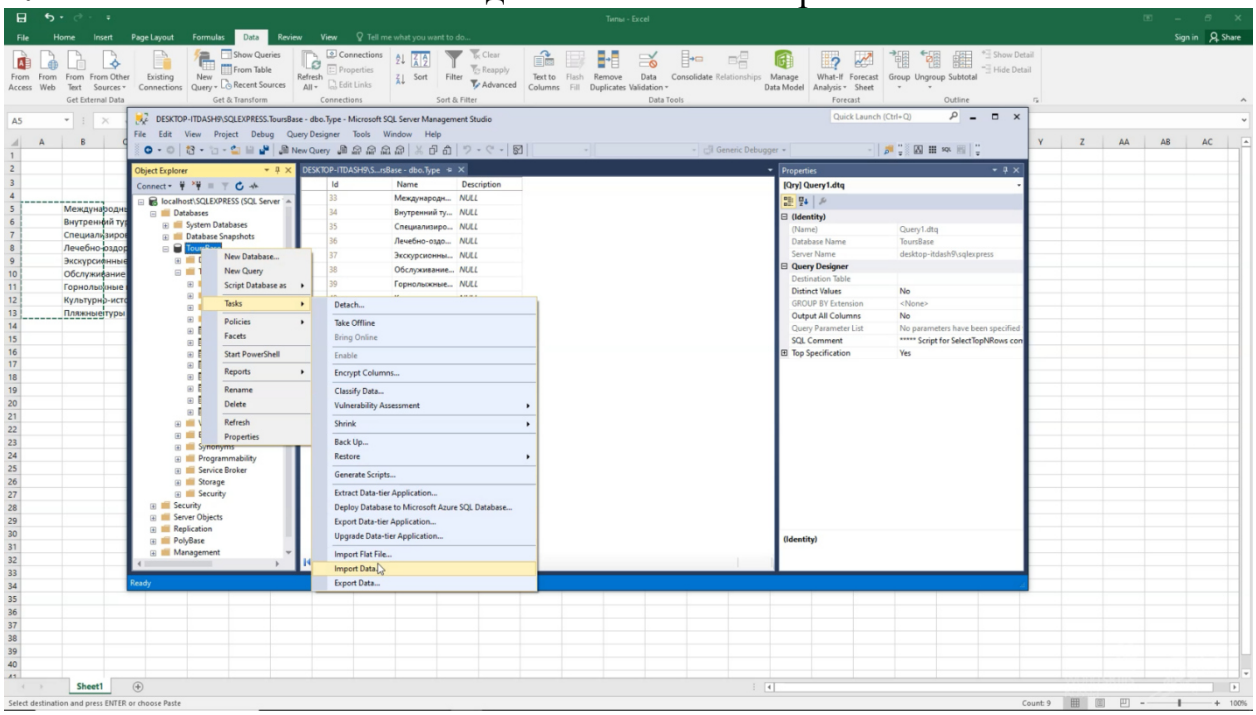


Работа с текстовыми данными

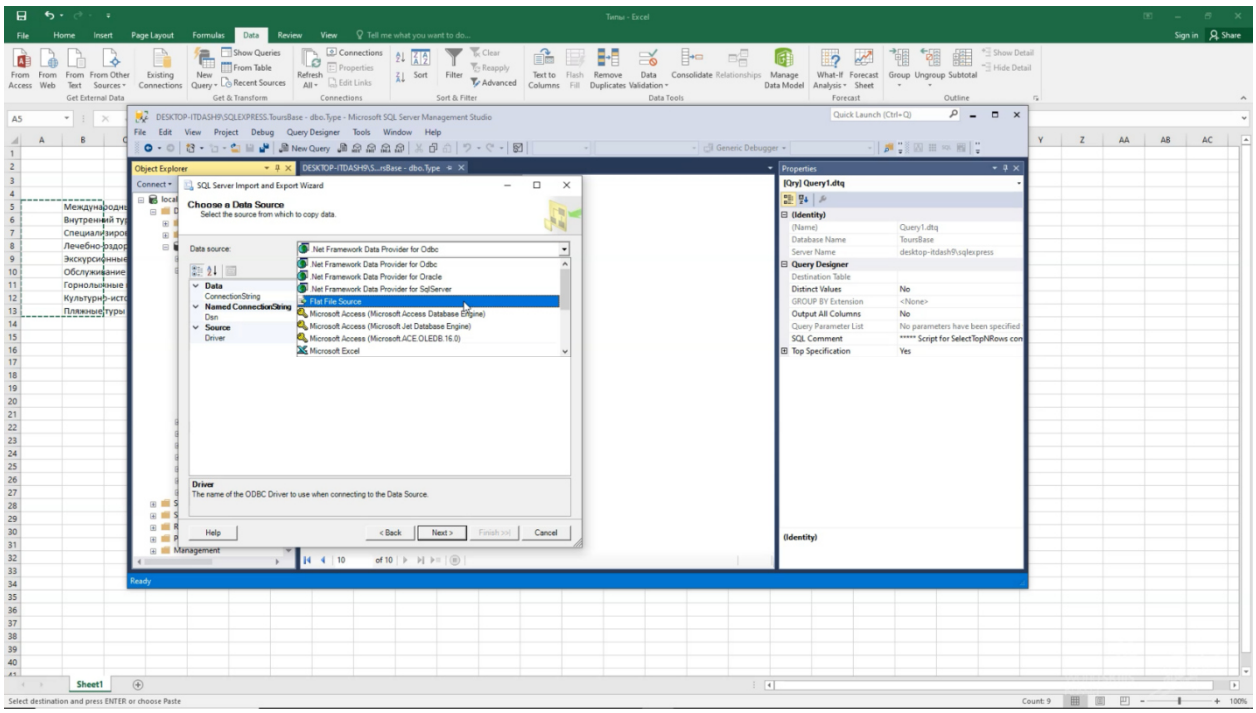
Импорт списка стран

Переходим к списку стран и воспользуемся мастером импорта и экспорта.

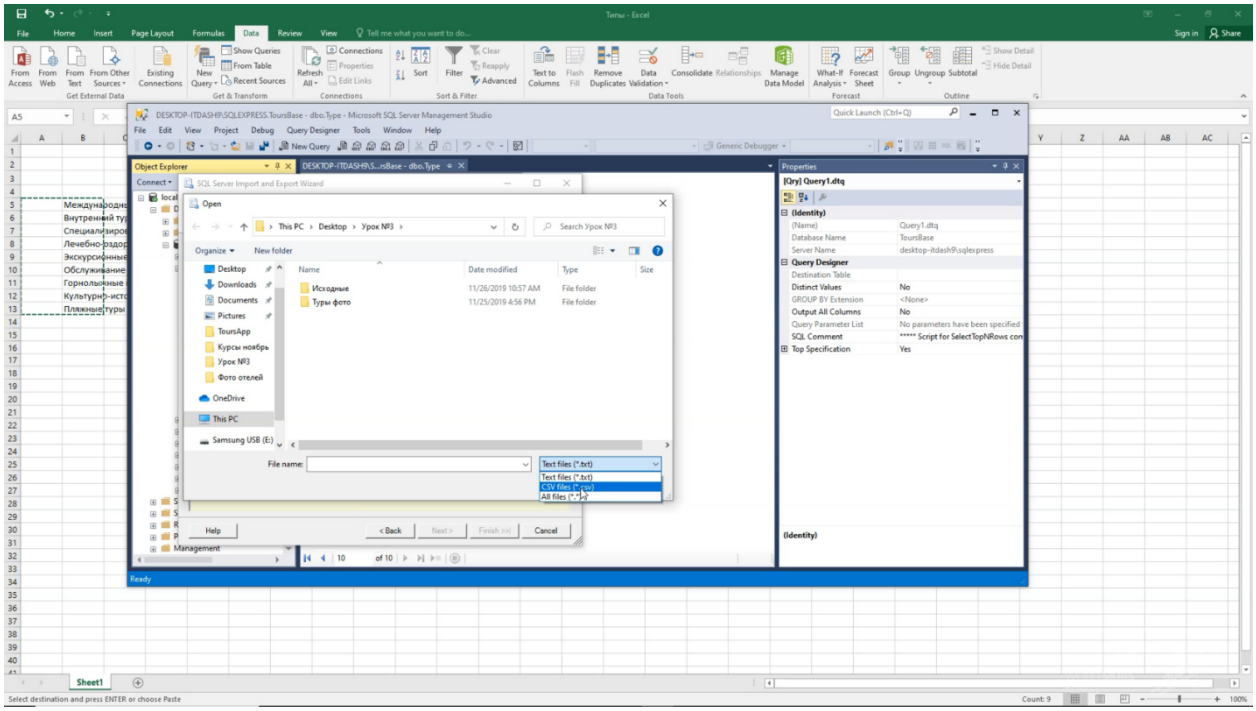
1. Кликаем по названию базы данных Tasks — Import data



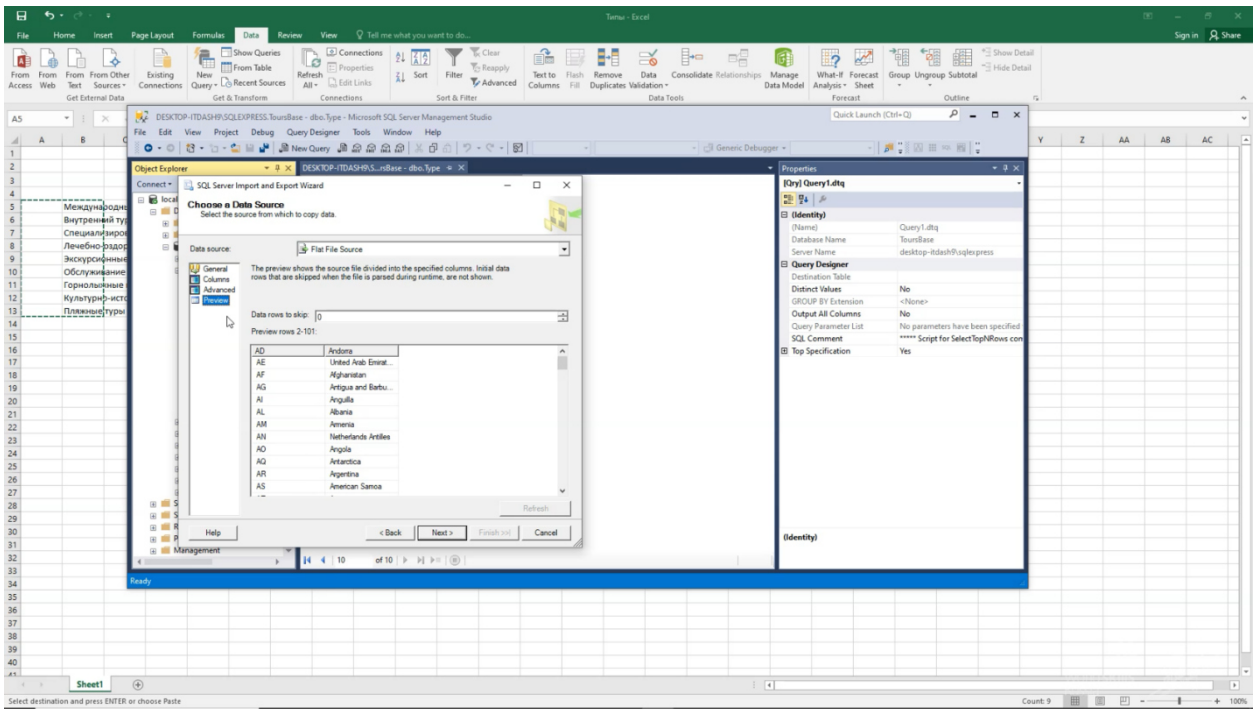
2. В качестве источника данных выбираем неструктурированный файл



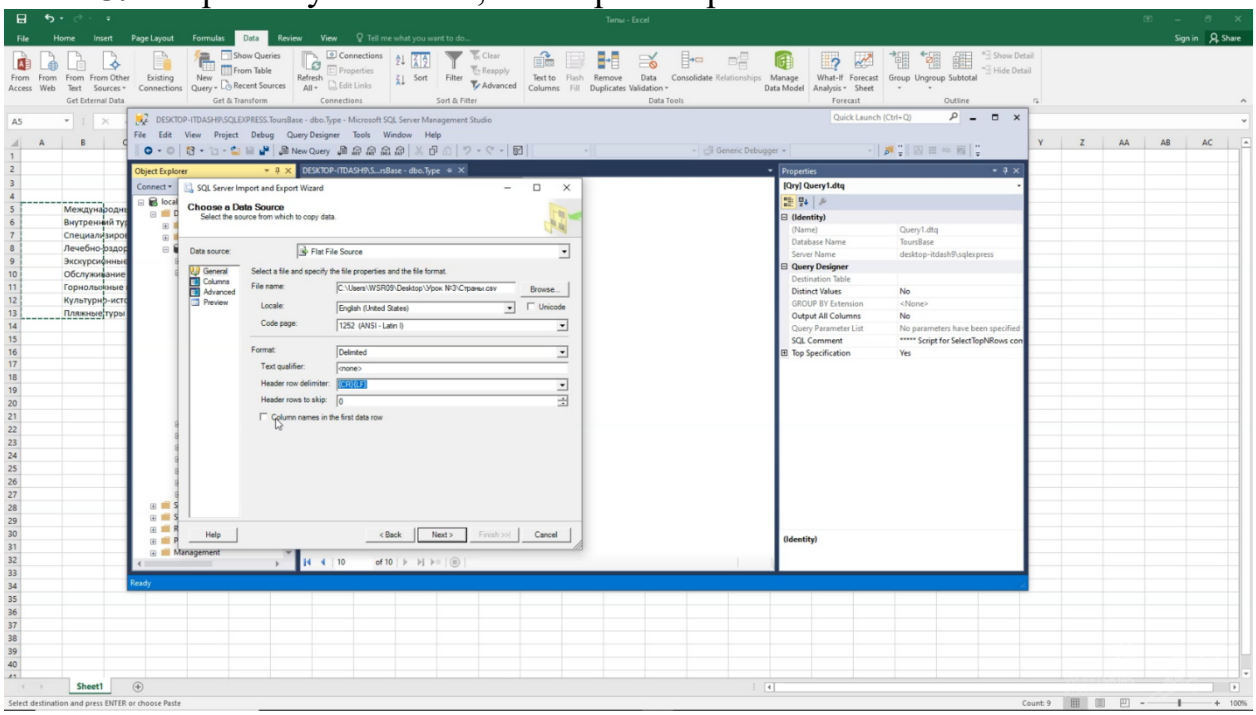
3. Тип csv files



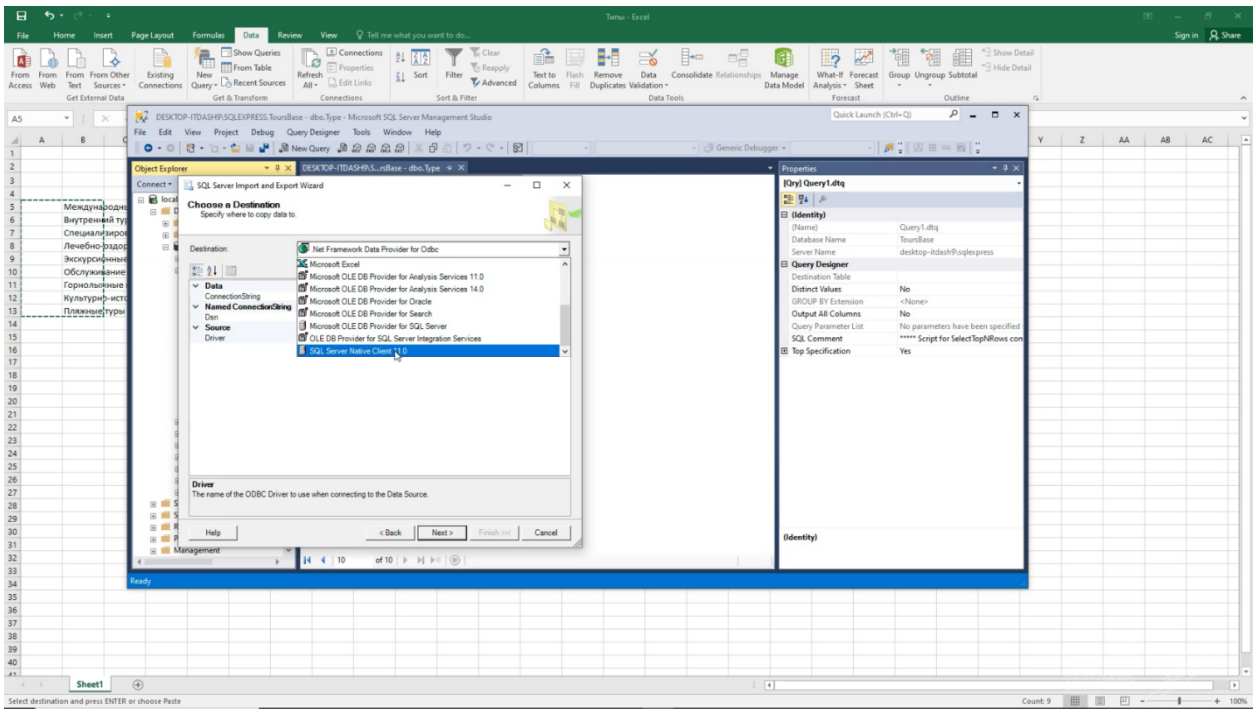
4. На Preview мы можем увидеть, как они будут представлены



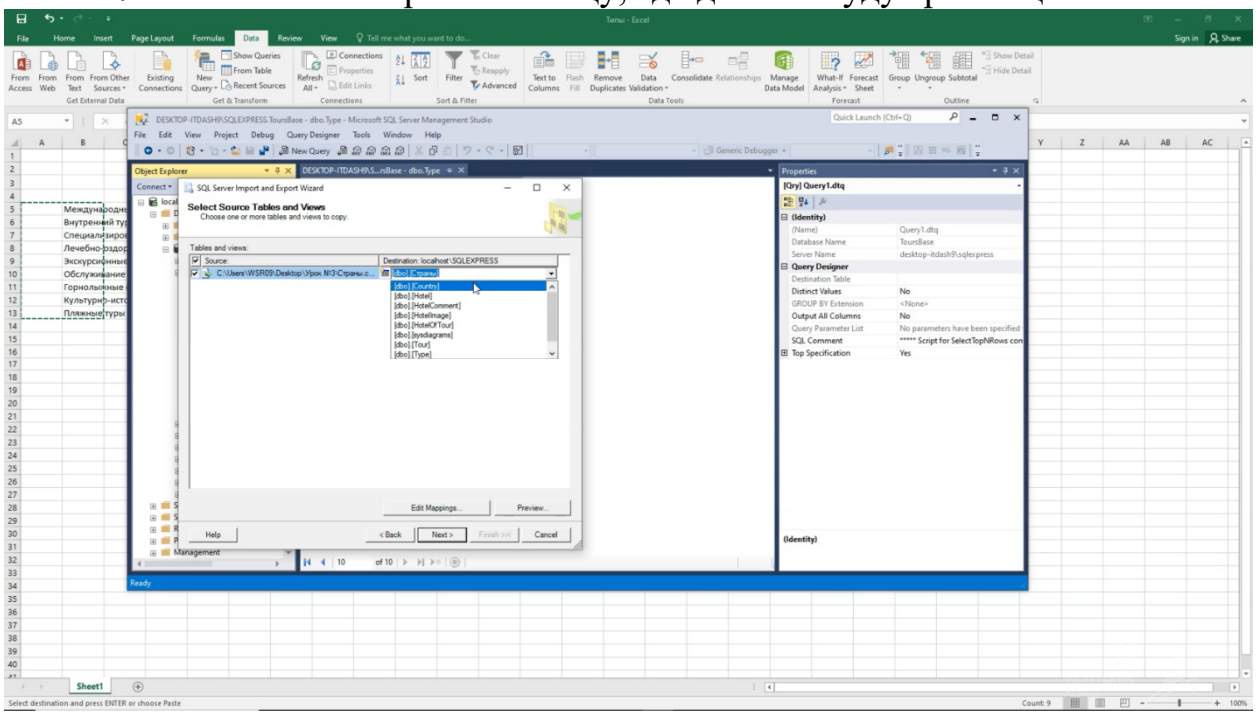
5. Убираем пункт о том, что первая строка — это заголовок



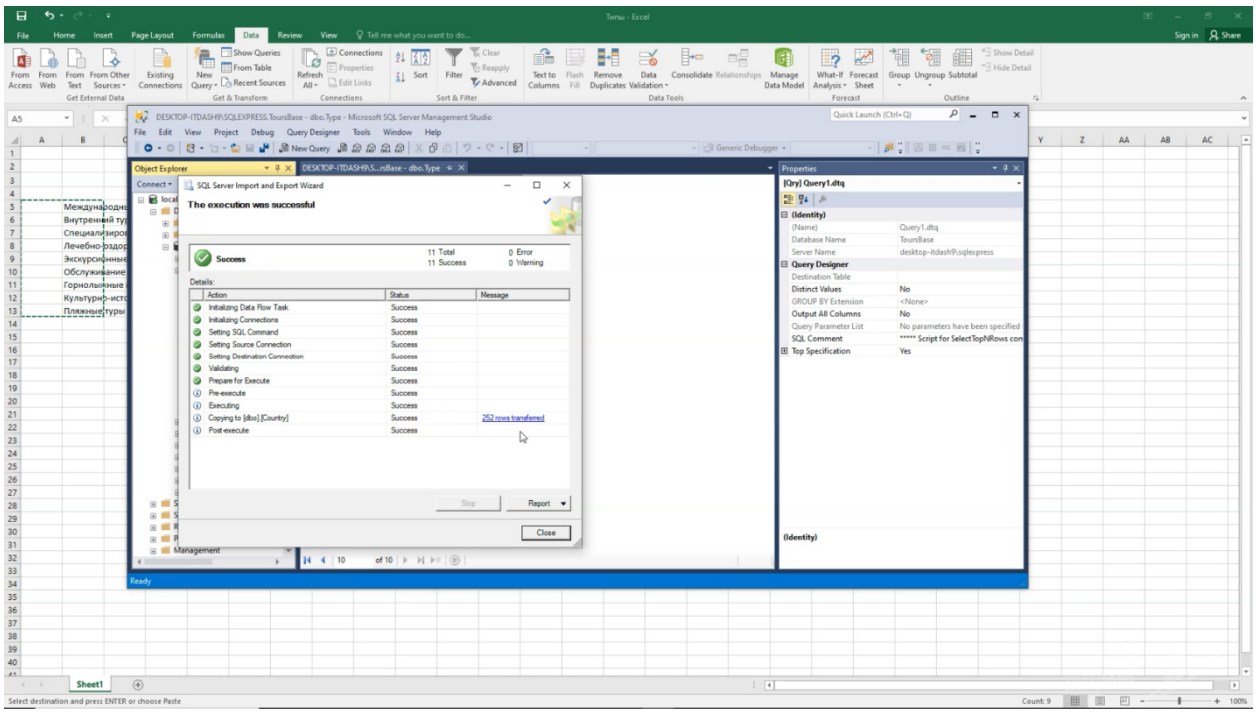
6. Далее выбираем, куда мы хотим импортировать данные — нашу базу



7. После чего выбираем таблицу, где данные будут размещены

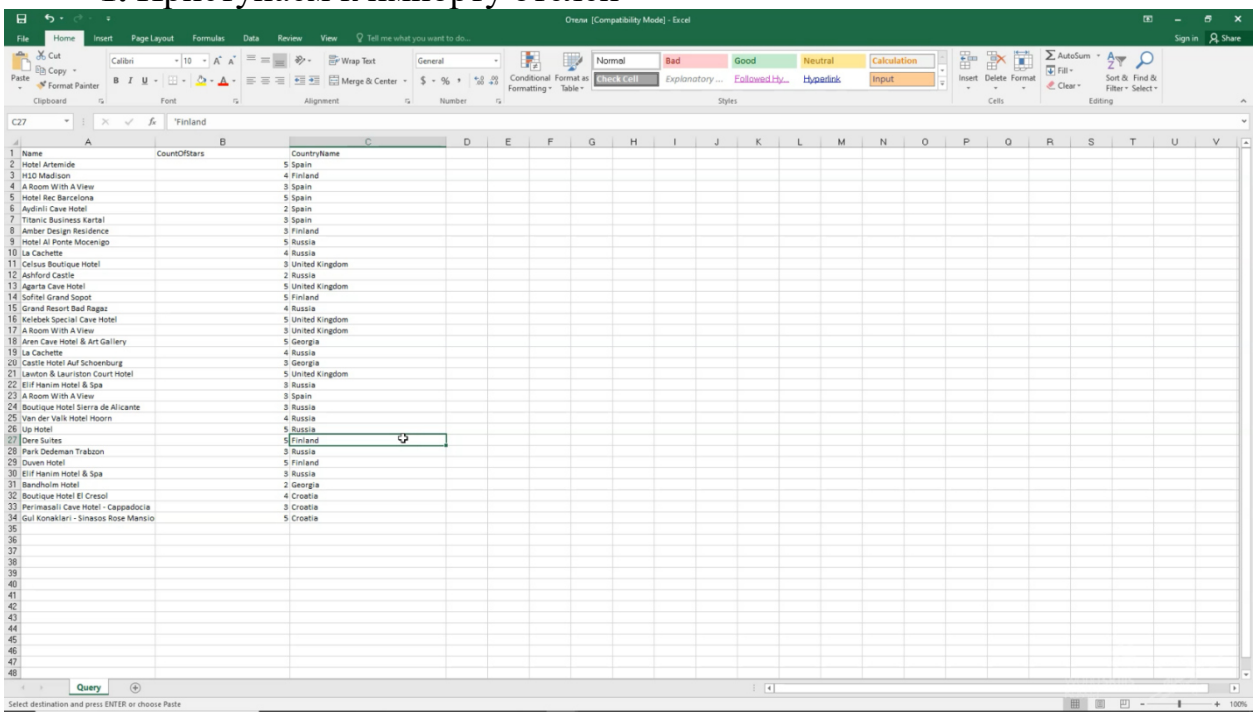


8. Далее — Далее — Финиш. 252 строки было импортировано. Можем их увидеть в таблице

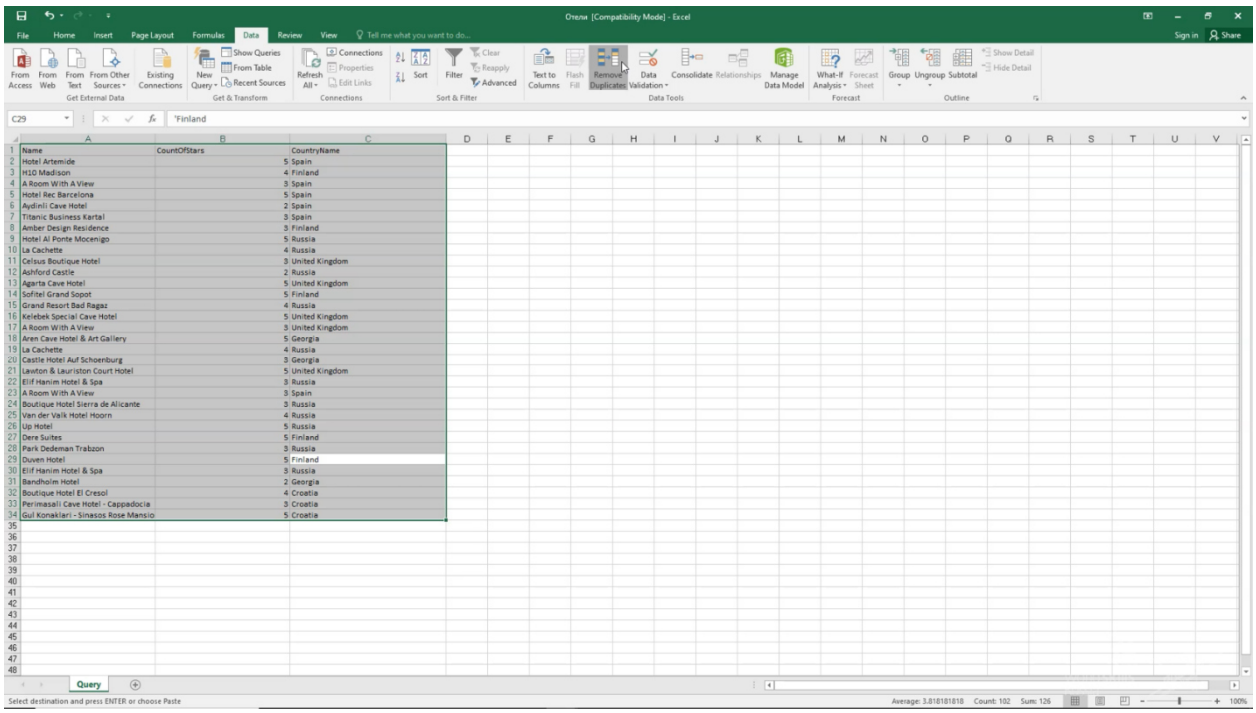


Импорт списка отелей

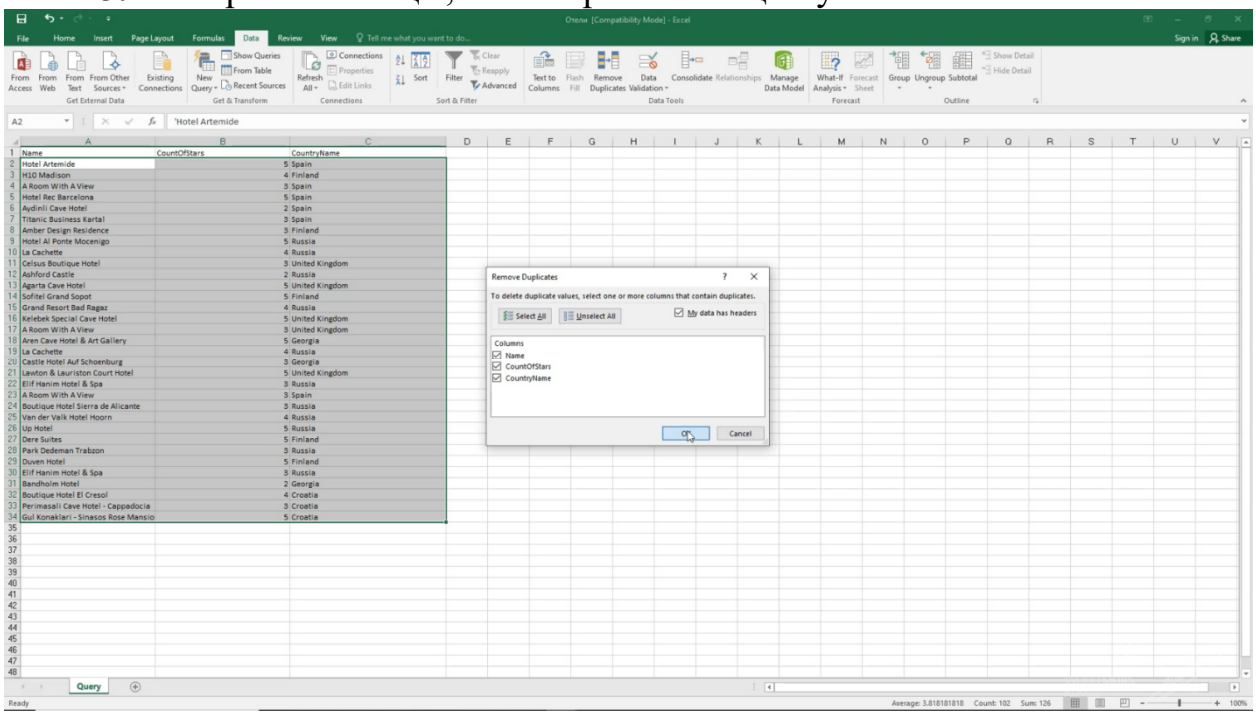
1. Приступаем к импорту отелей



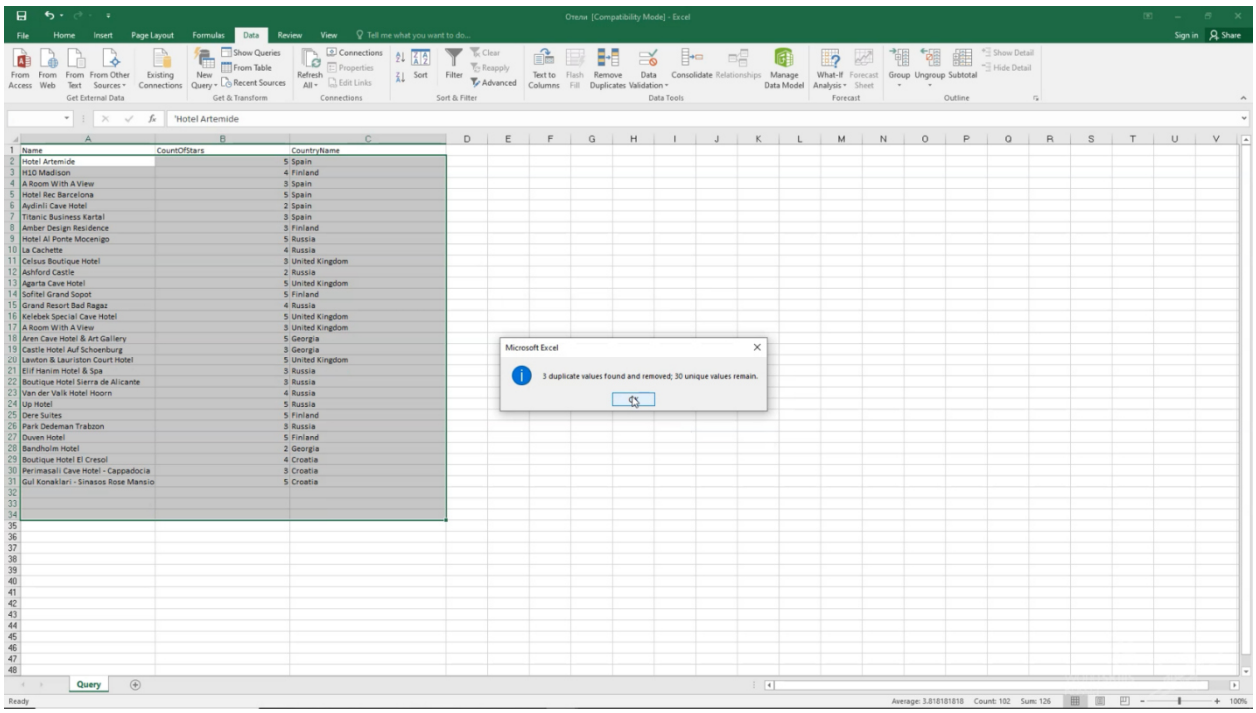
2. В таблицах могут встречаться дубликаты, поэтому желательно проверять этот момент и удалять их. Выделяем все данные — Data — Remove duplicates



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

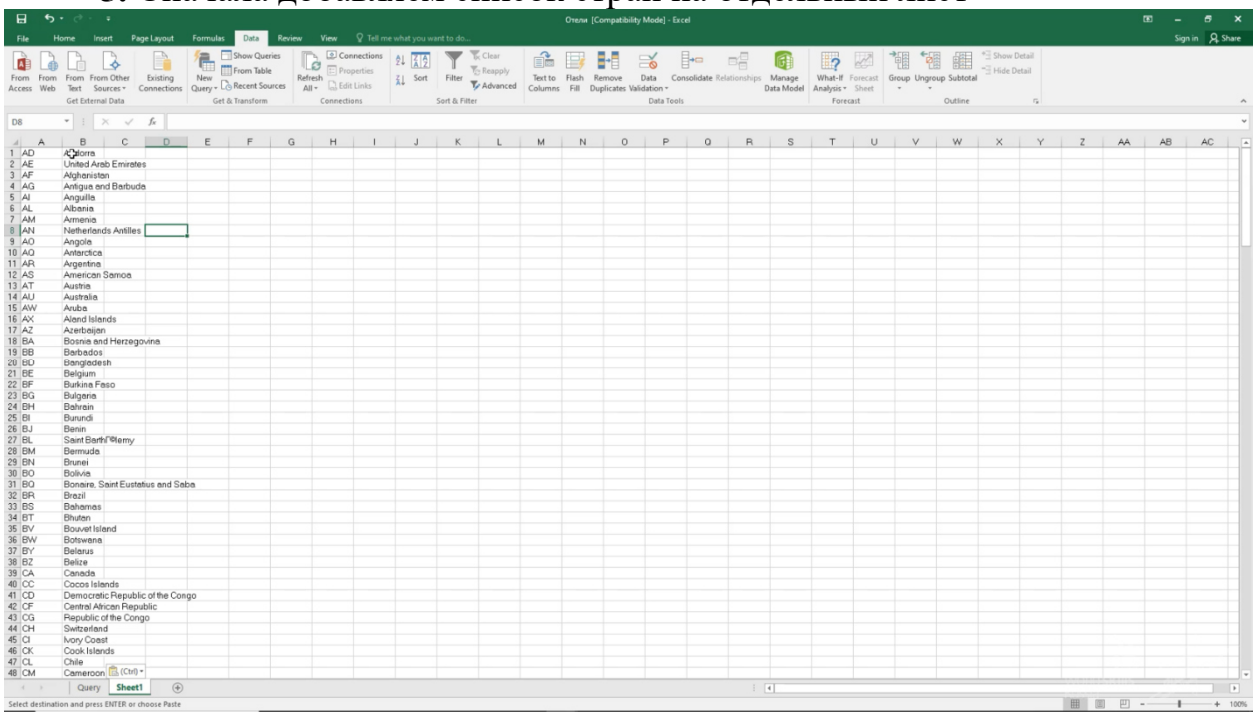


4. Три дубликата было удалено

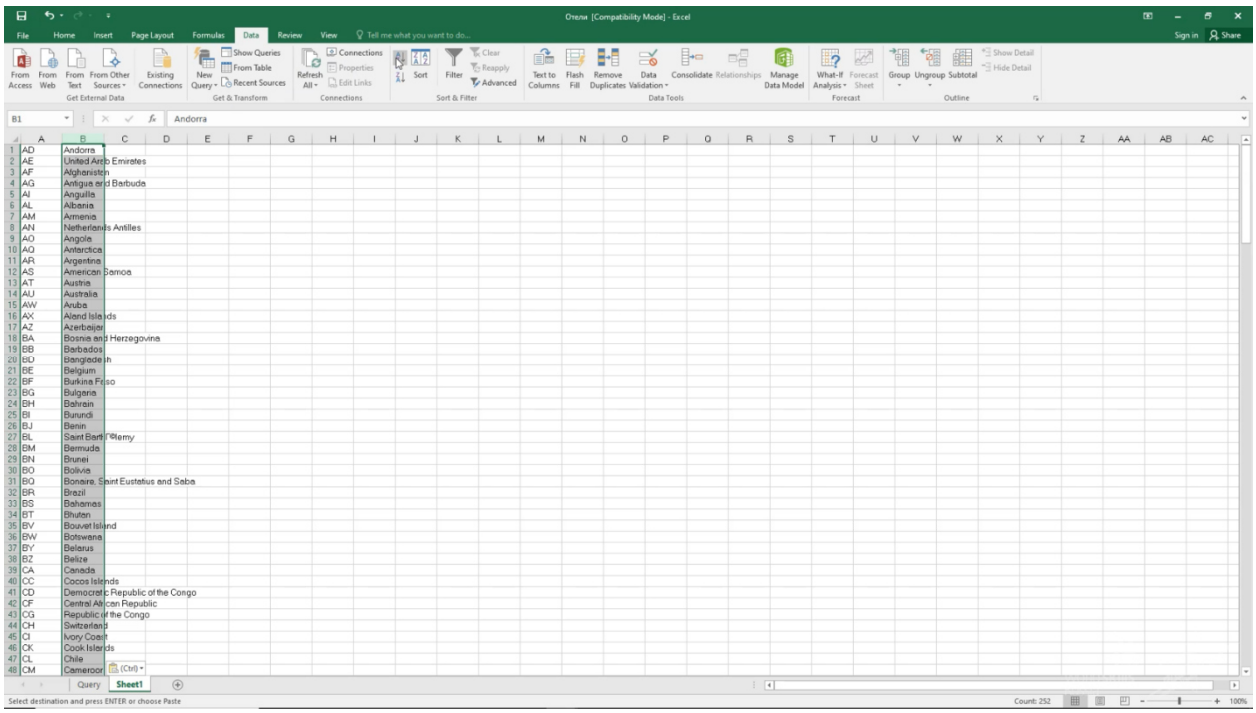


Для импорта в базу данных нам нужен столбец с кодами стран вместо названий. Для замены мы будем использовать функцию LOOKUP

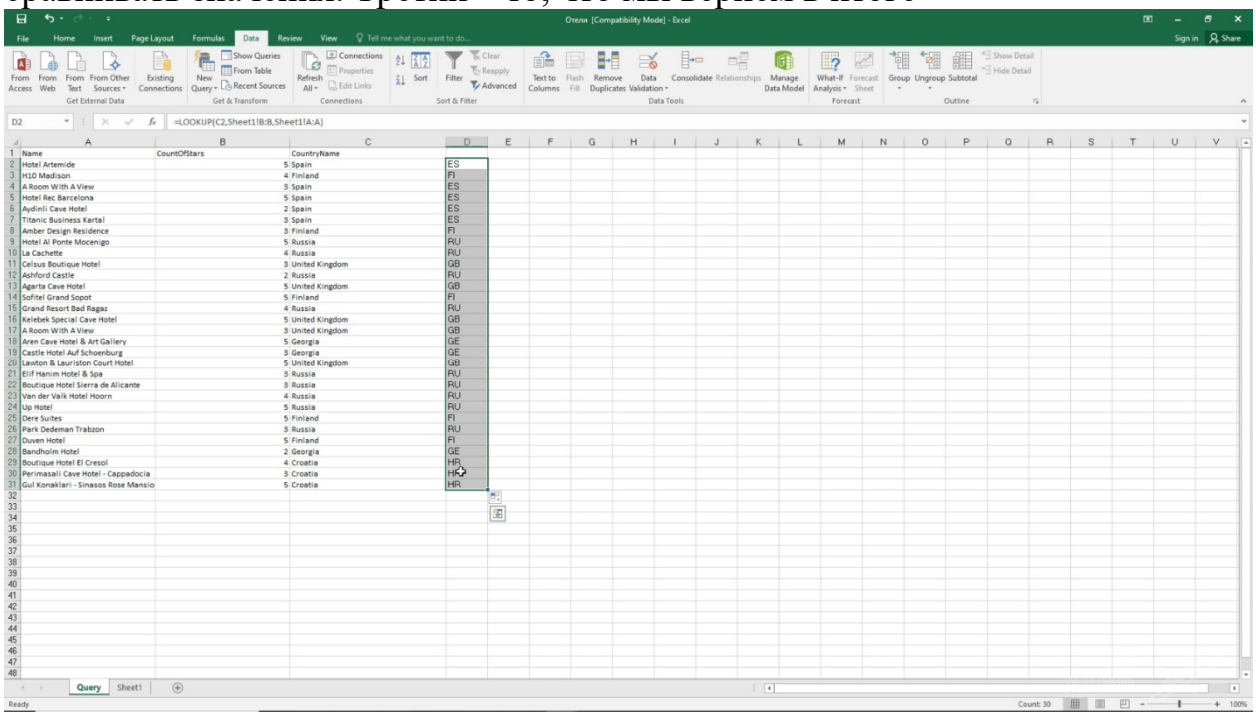
5. Сначала добавляем список стран на отдельный лист



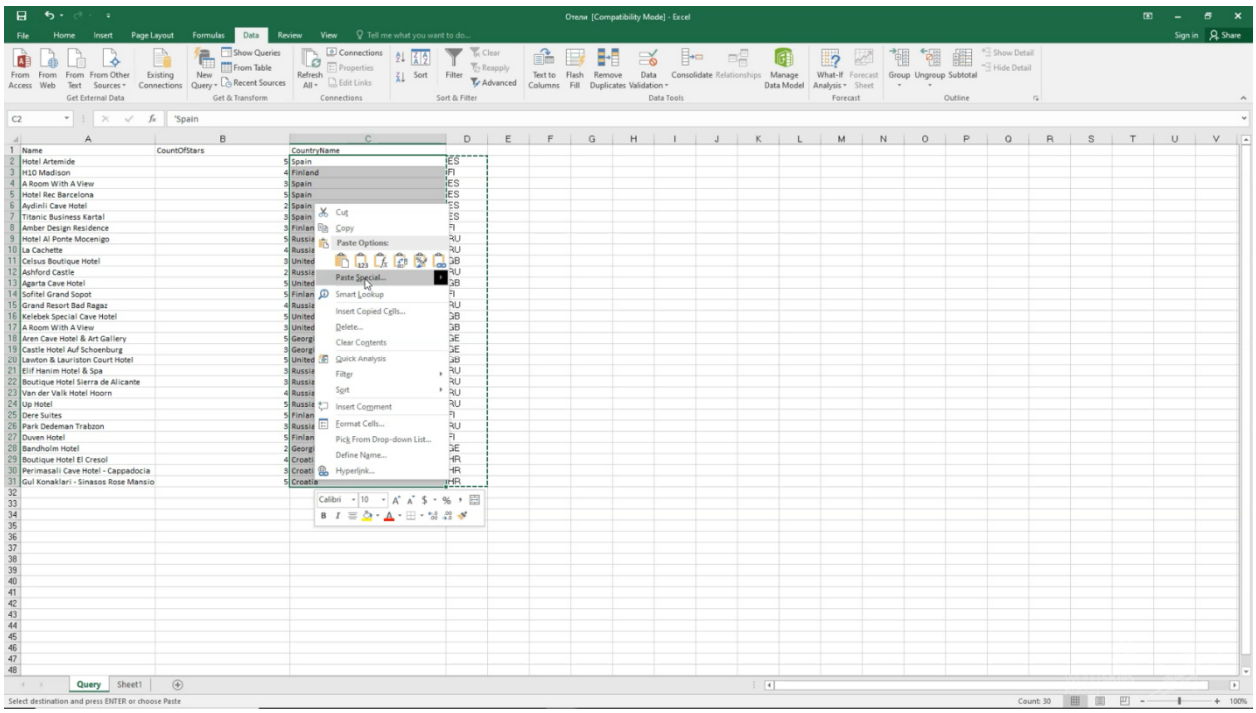
6. Для использования функции LOOKUP лучше, если данные отсортированы в алфавитном порядке



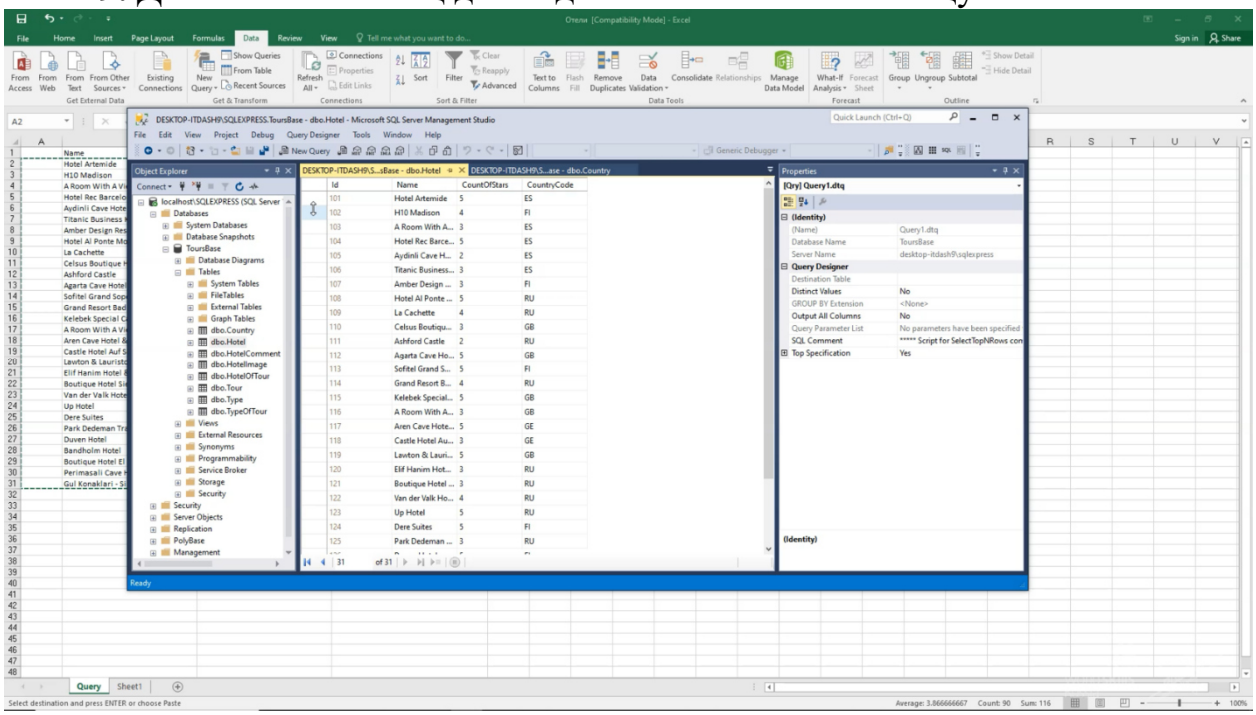
7. Воспользуемся функцией. Первый аргумент — это значение, по которому мы происходит поиск. Второй — это столбец, где мы будем сравнивать значения. Третий – то, что мы вернем в итоге



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



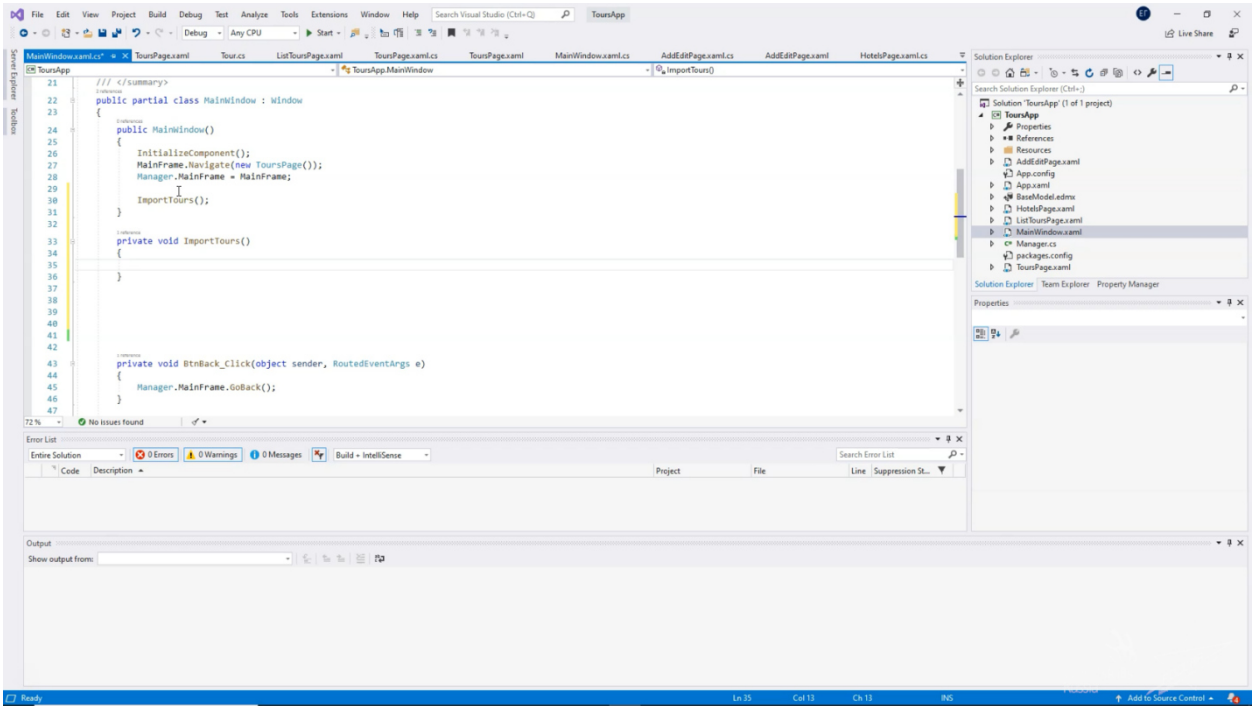
9. Добавляем столбец для кодов и вставляем в таблицу Hotel



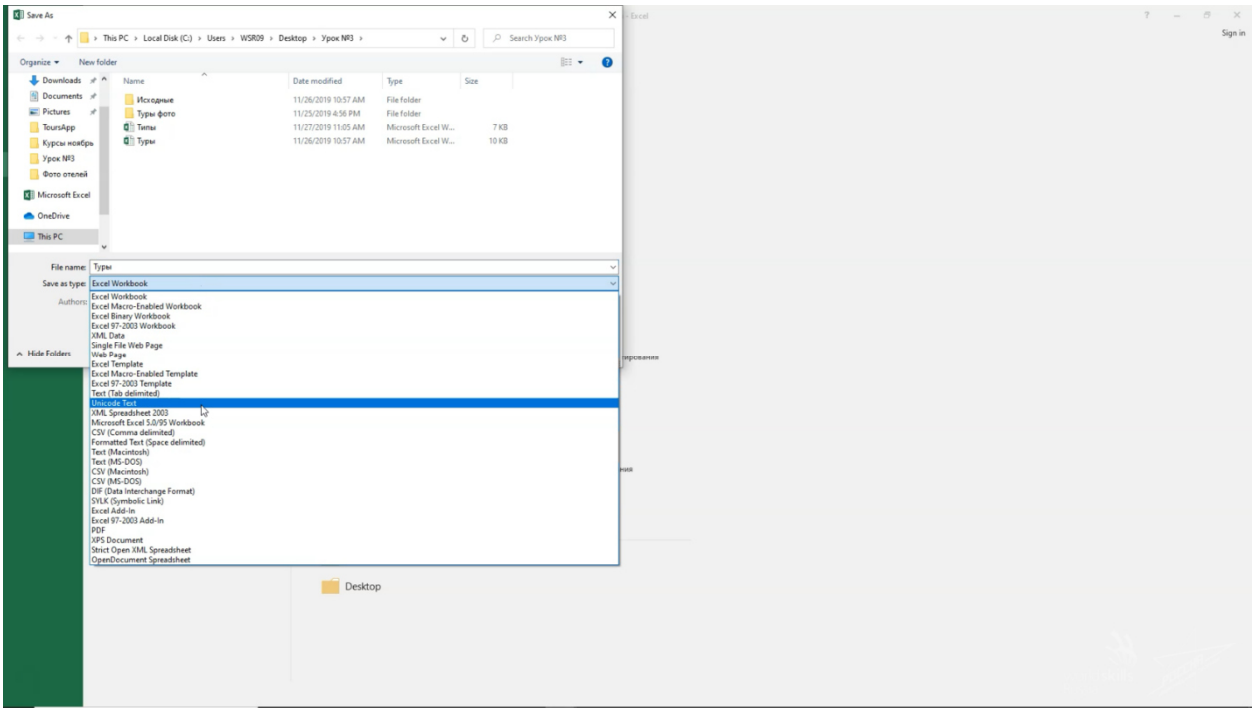
Импорт таблицы туров

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

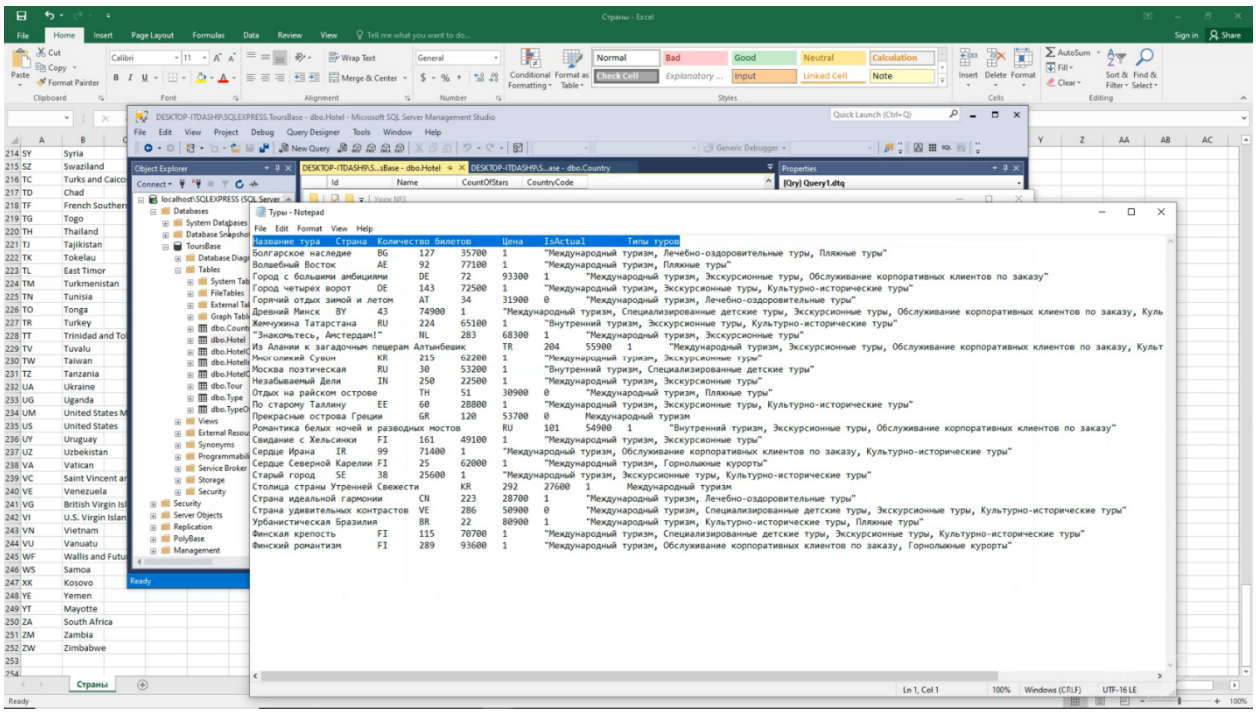
1. Воспользуемся существующим проектом, и в Mail windows создаем метод для импорта туров. Можем сразу вызвать его при запуске



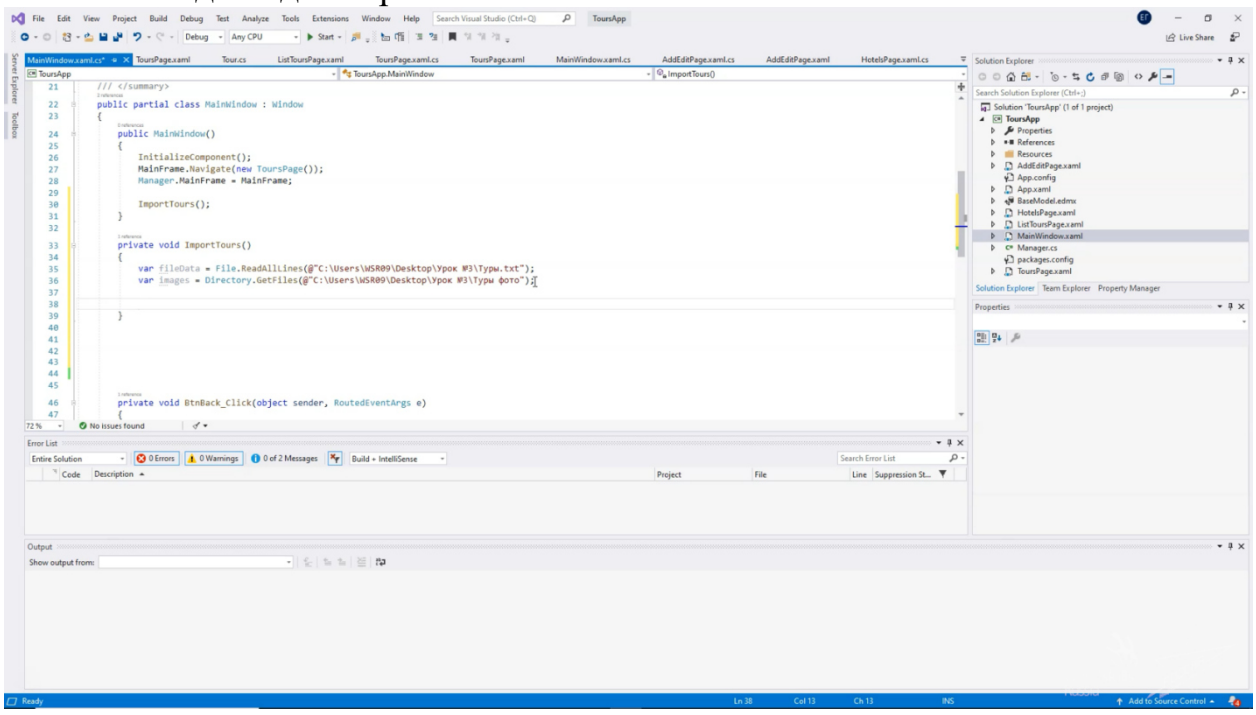
2. Прежде чем работать в коде с файлом туров, сохраняем его как unicode text и удаляем в итоговом файле первую строку с заголовком. File — Save as — Unicode text



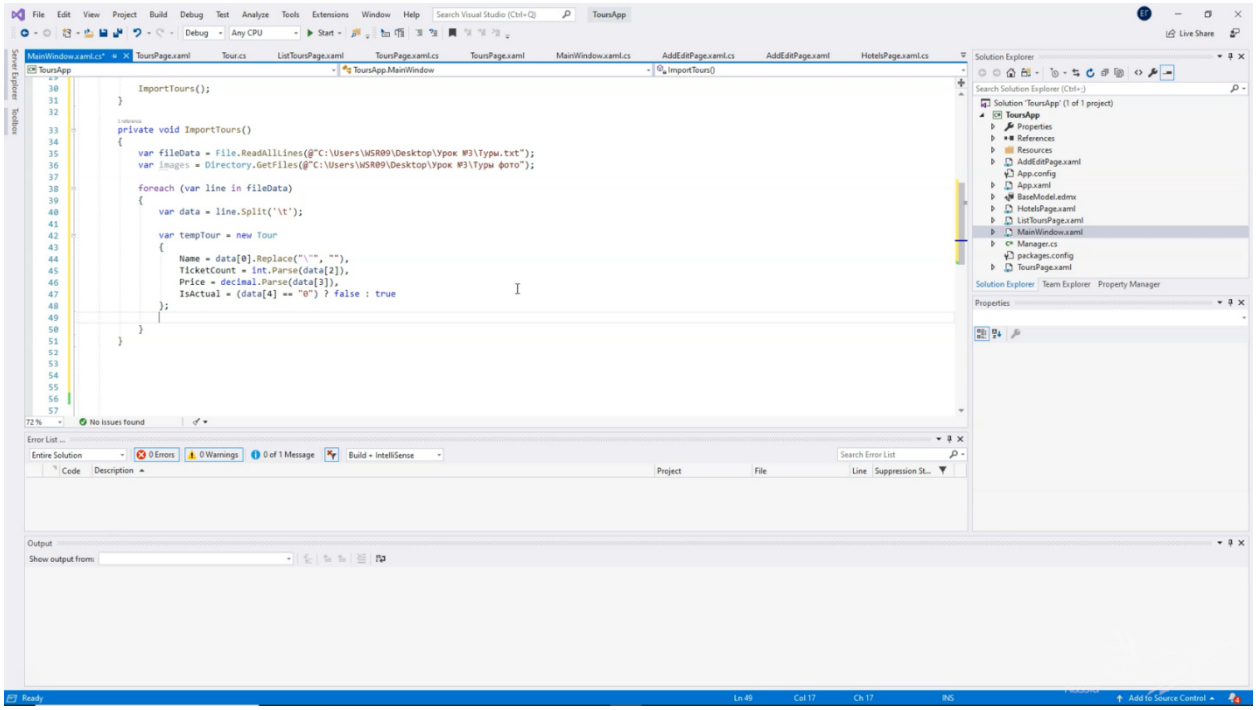
3. Удаляем строку с заголовками



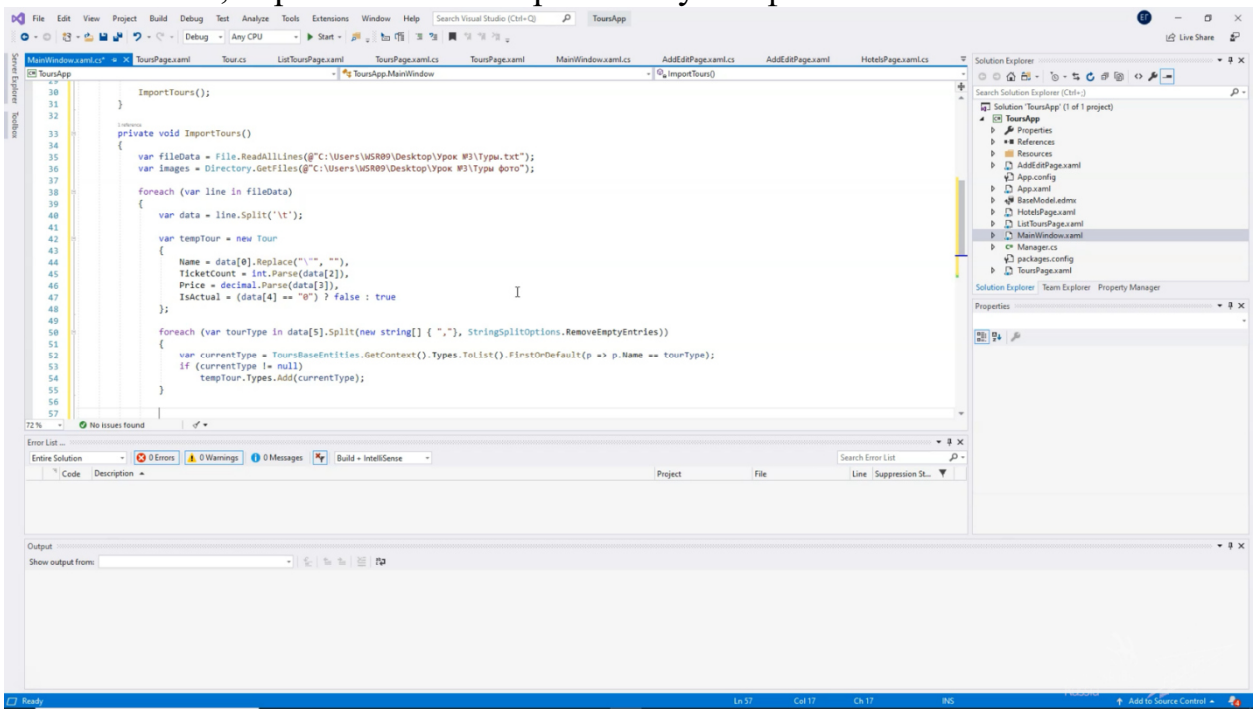
4. Создаем две переменные



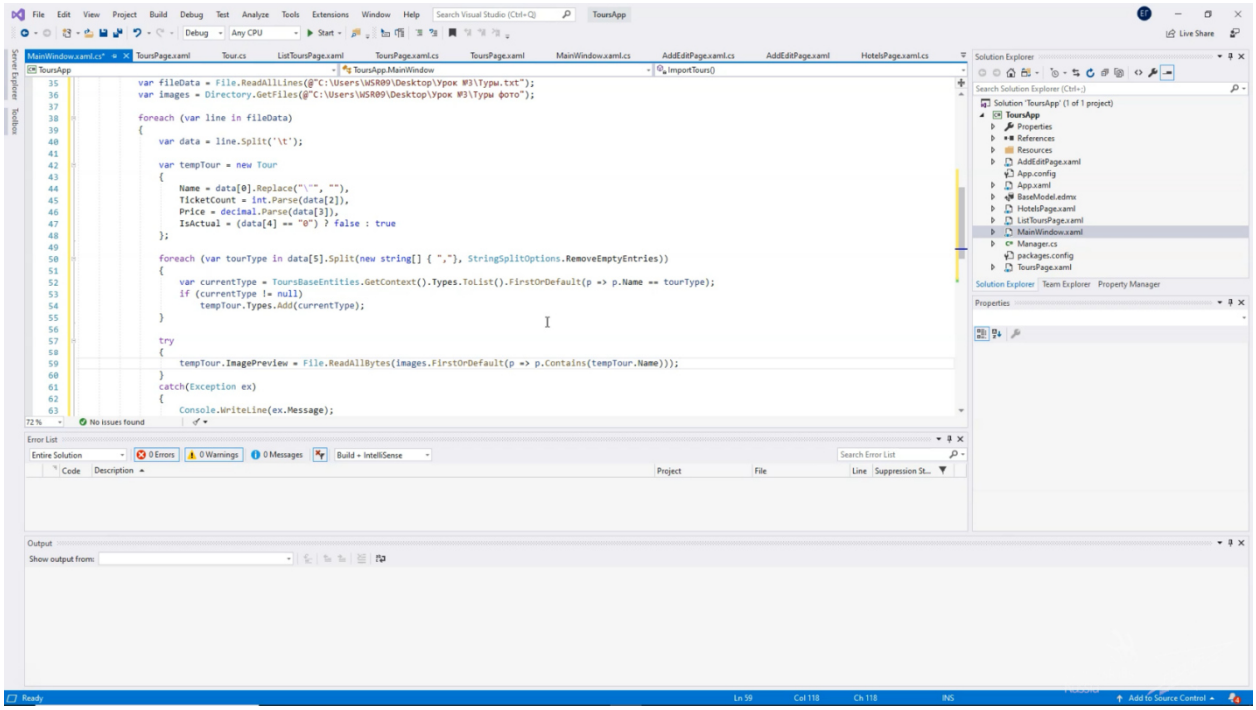
5. Пробежимся по строкам в файле, разделив данные с помощью табов, и создаем экземпляр класса, заполнив свойства соответствующими значениями, и не забывая про типы данных



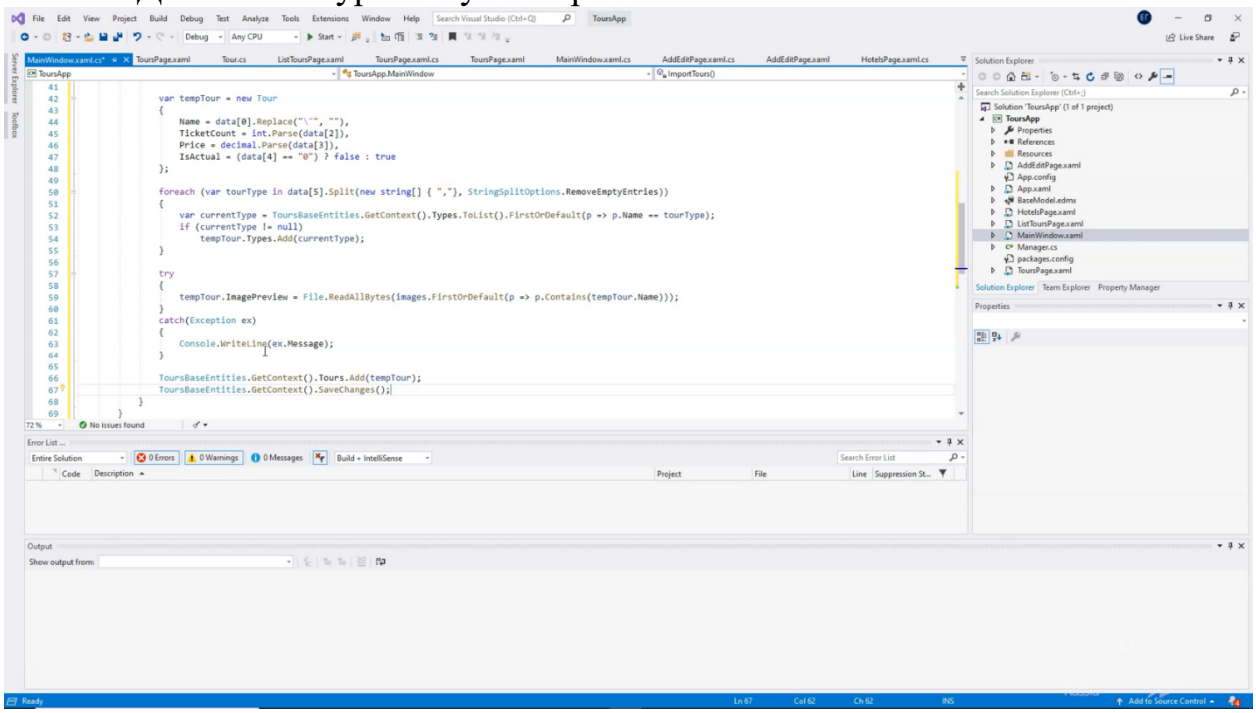
6. Заполняем коллекцию типов тура, выполнив поиск типов по названиям, перечисленным через запятую в файле



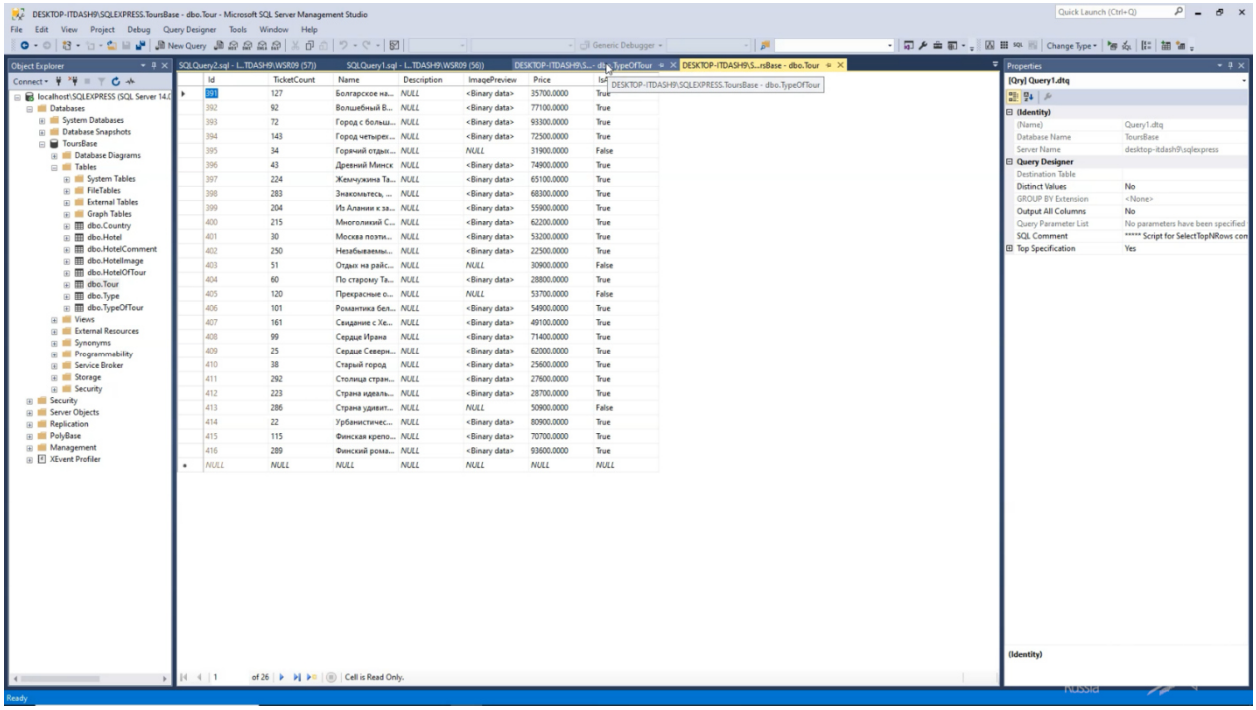
7. Записываем изображение в базу данных с помощью метода ReadAllBytes()

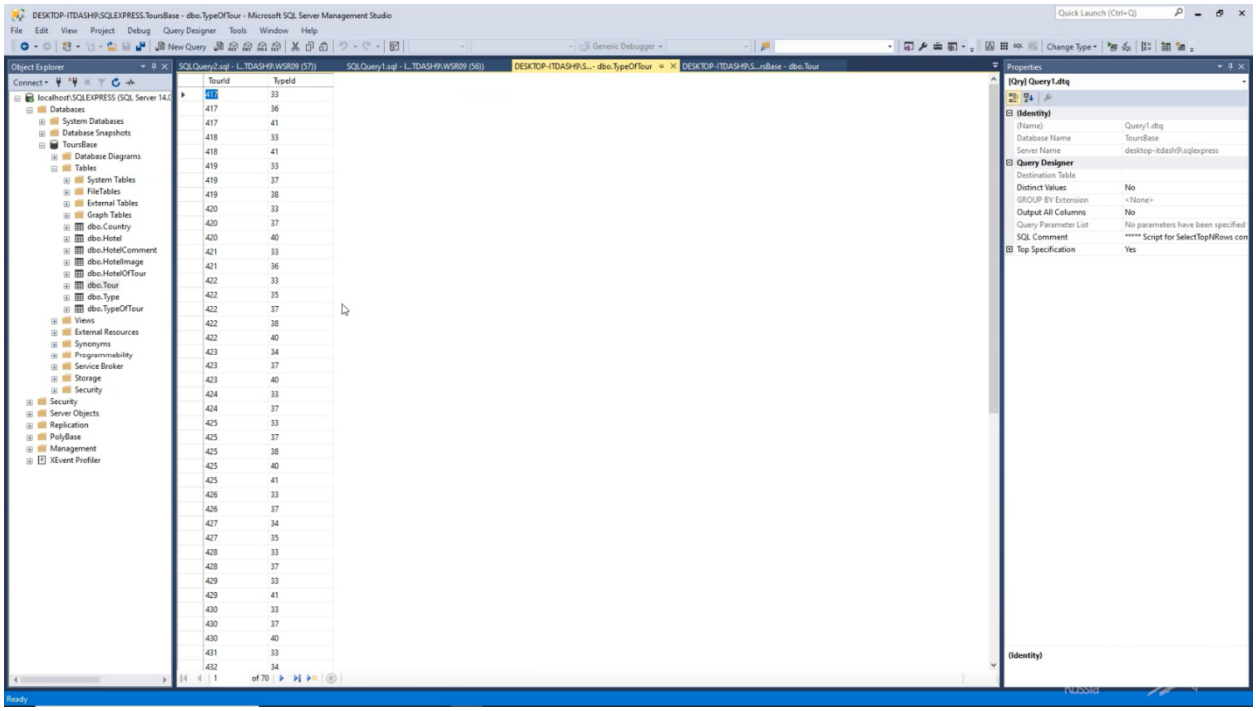


8. Добавляем тур в базу и сохраняем



9. Вызываем метод в конструкторе MainWindow и проверяем результат в базе данных. Список туров и типы импортированы



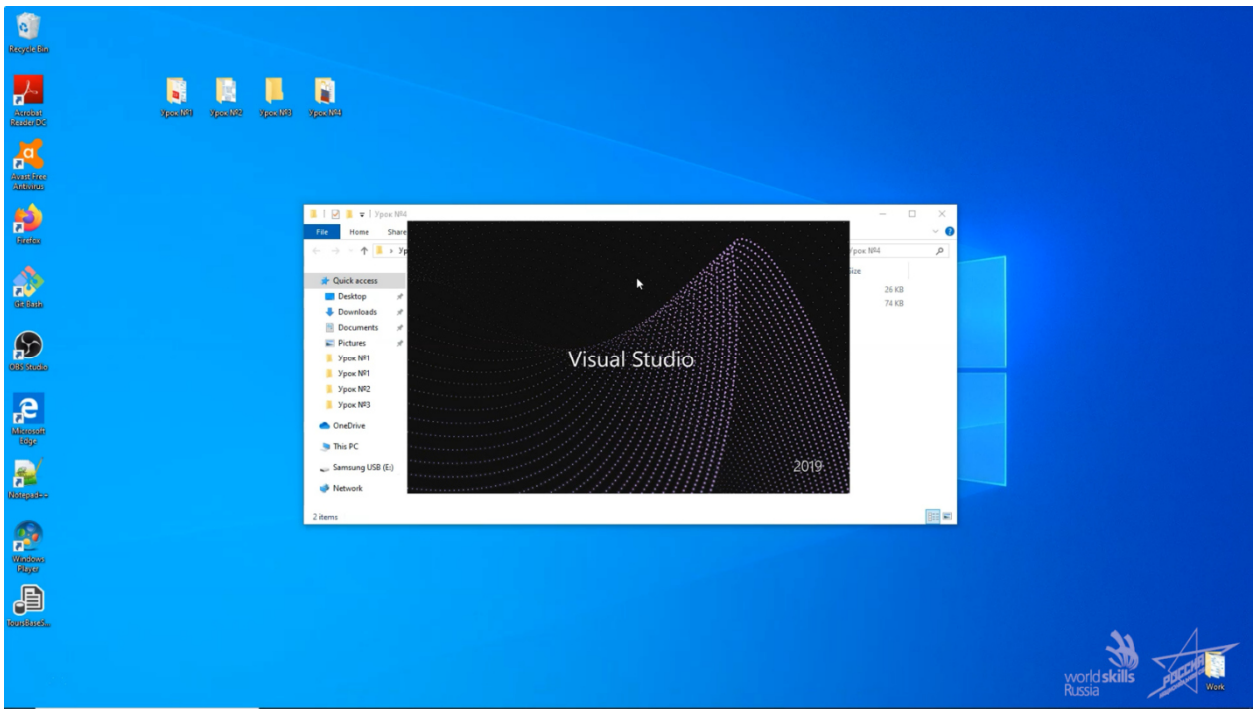


Интерактивное задание https://nationalteam.worldskills.ru/skills/rabota-s-nestrukturirovannymi-dannymi-obrabotka-i-import-v-bazu-dannykh/

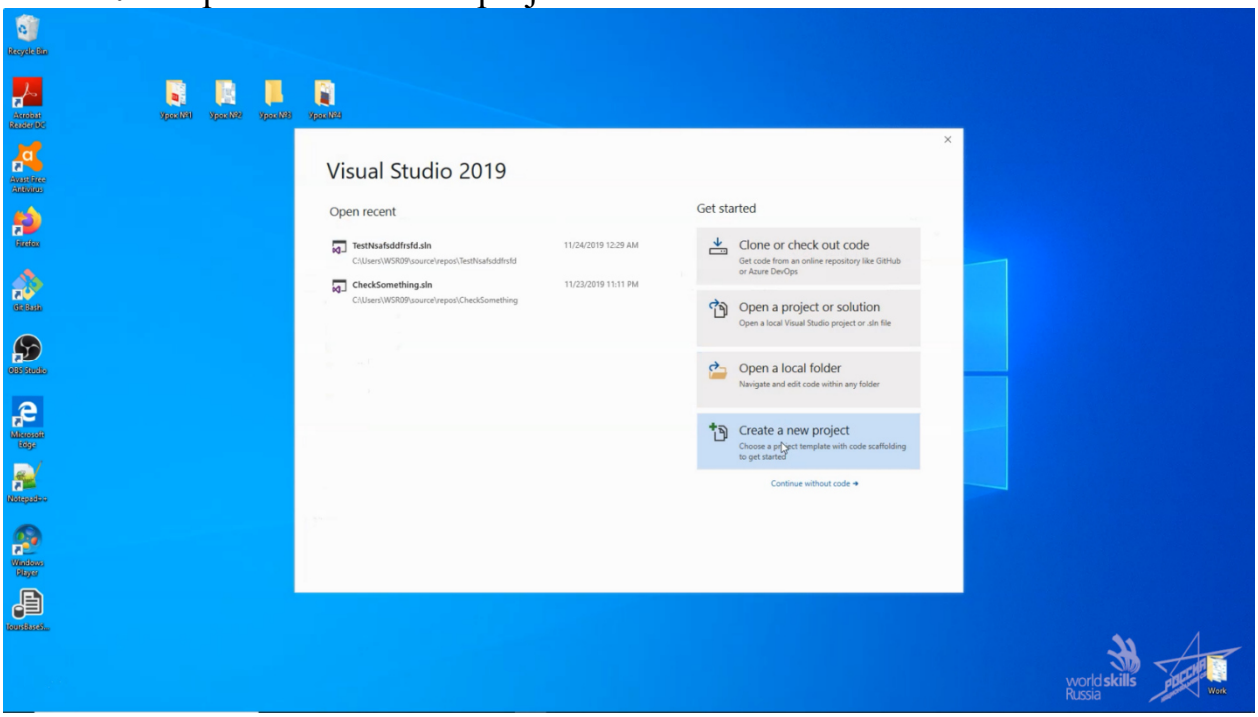
Практическая работа №10. Создание каркаса приложения с использованием WPF. Создание и использование стилей

Создание нового проекта

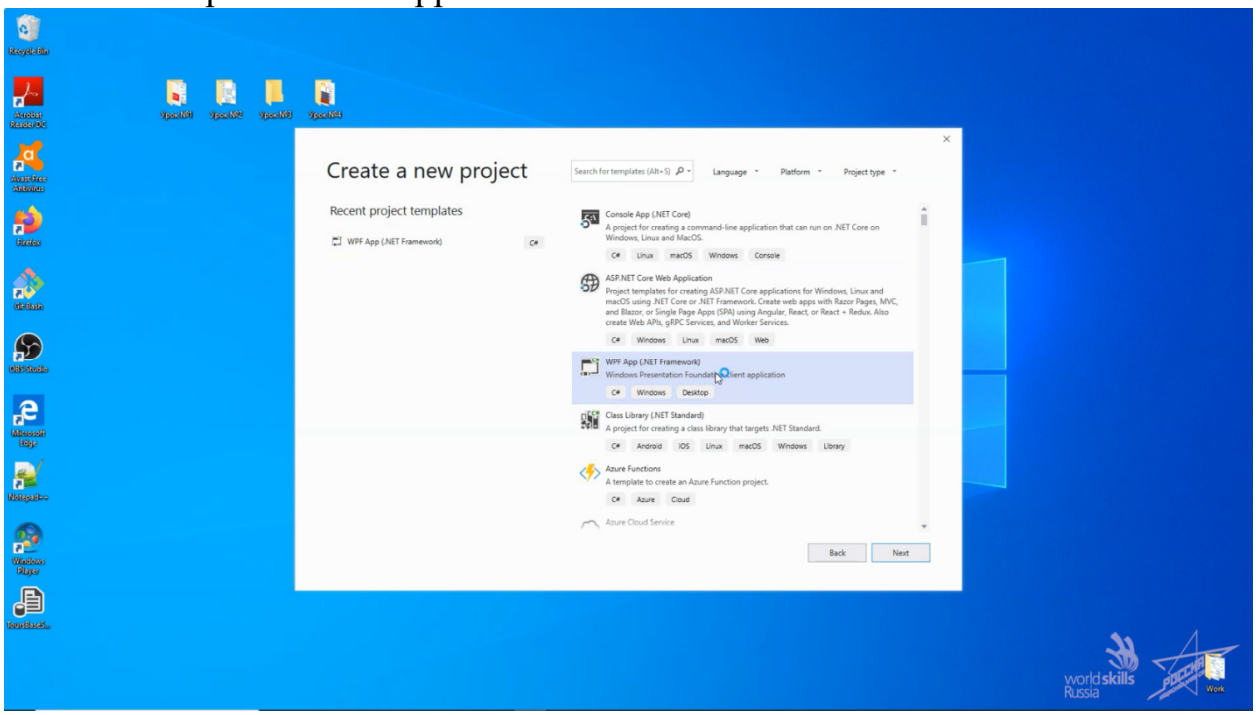
1. Запустить среду разработки Visual Studio



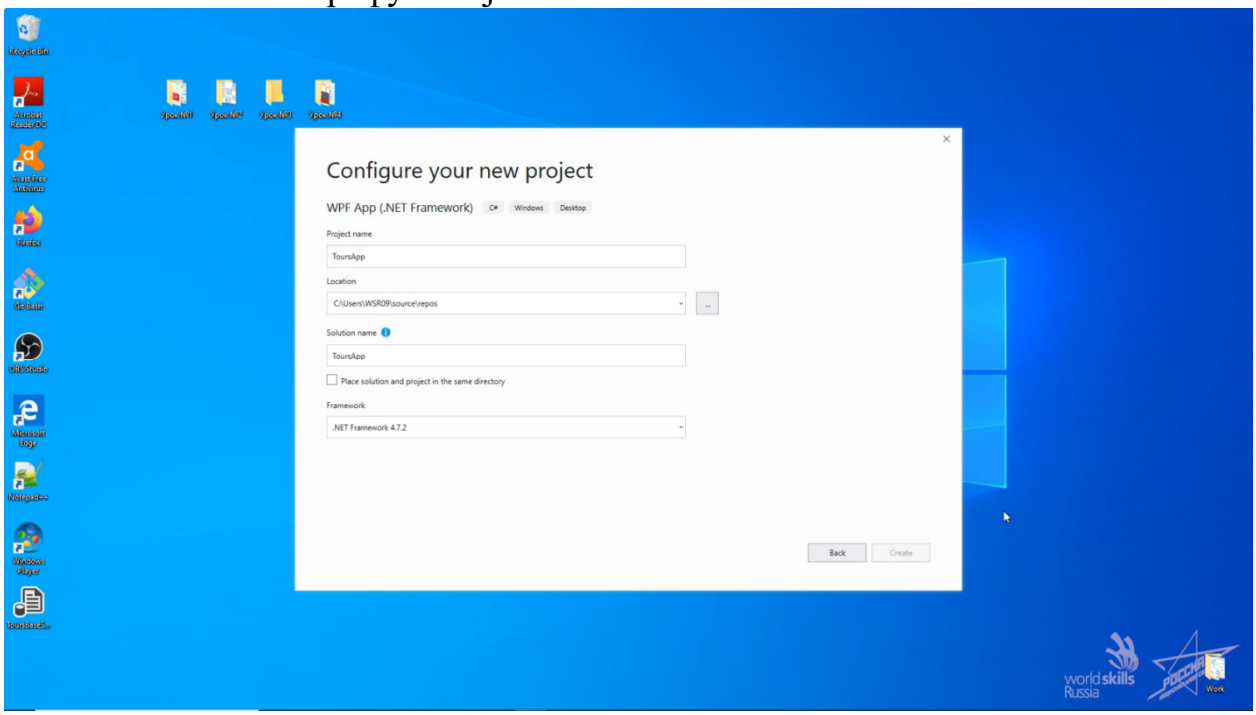
2. Выбрать «Create new project»



3. Выбрать «WPF App»

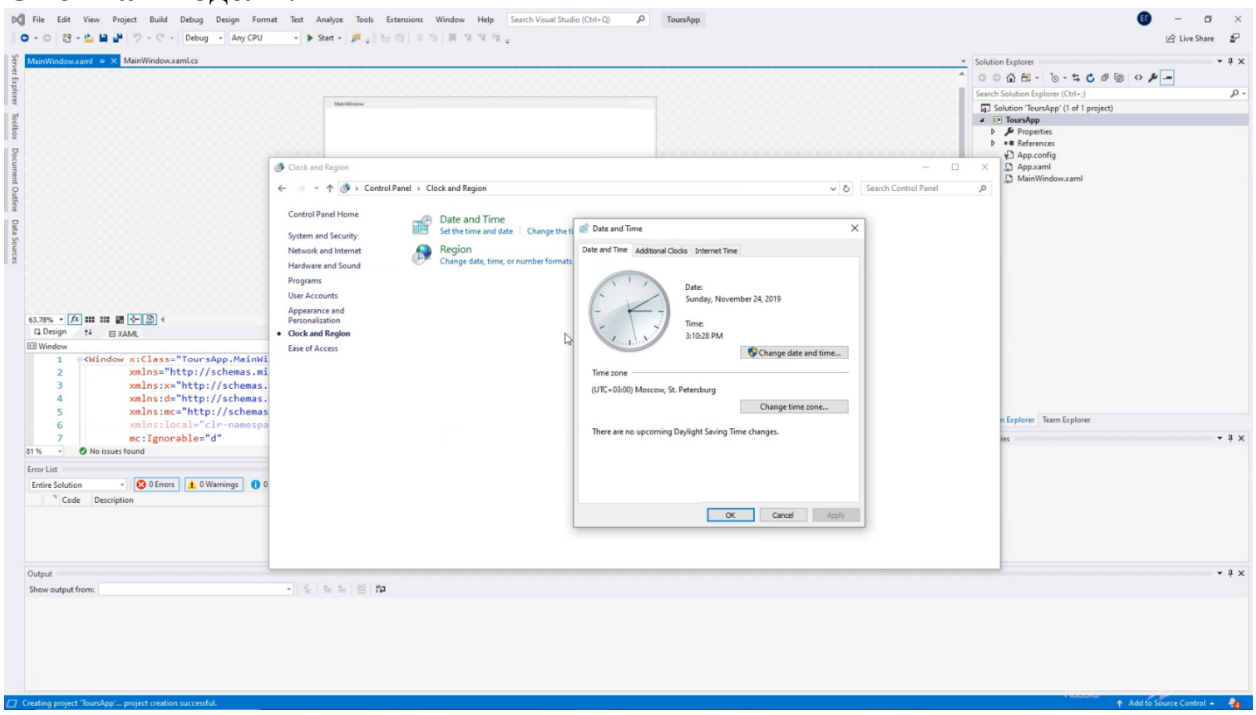


4. Заполнить графу «Project name»

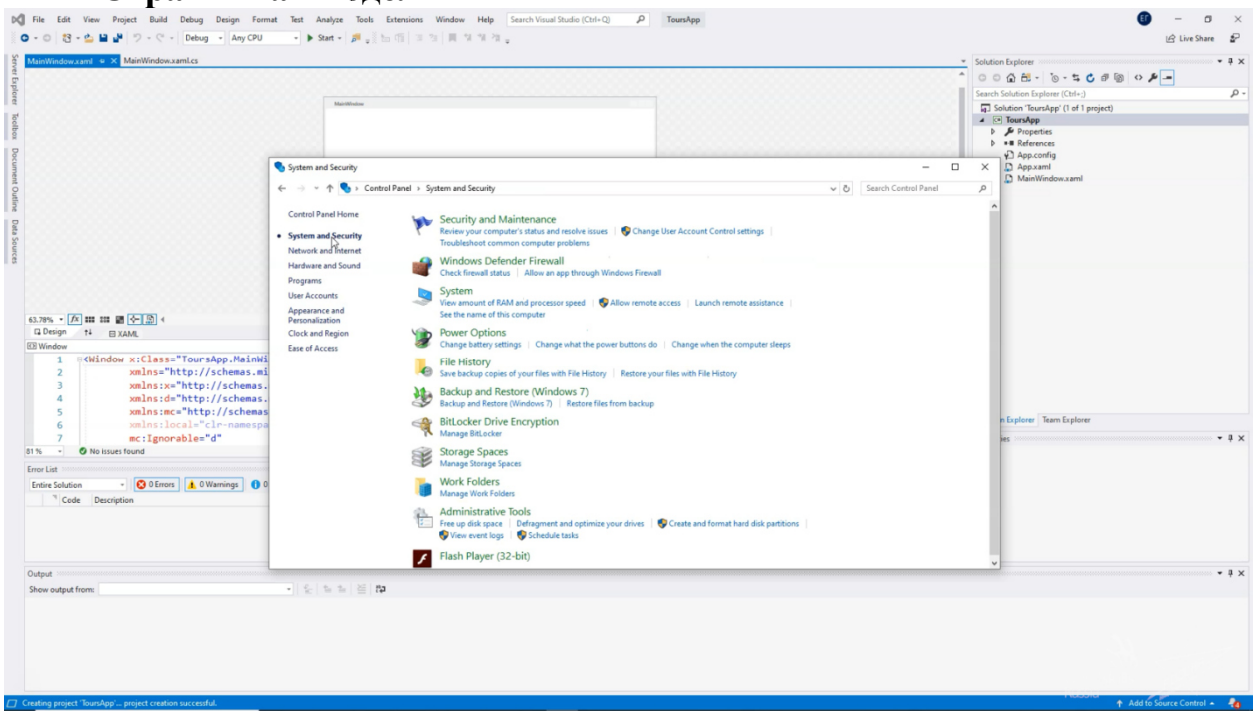


При разработке интерфейсов разработчик может использовать две модели: **оконную** или **страничную** (в настоящее время используют чаще)

Оконная модель:

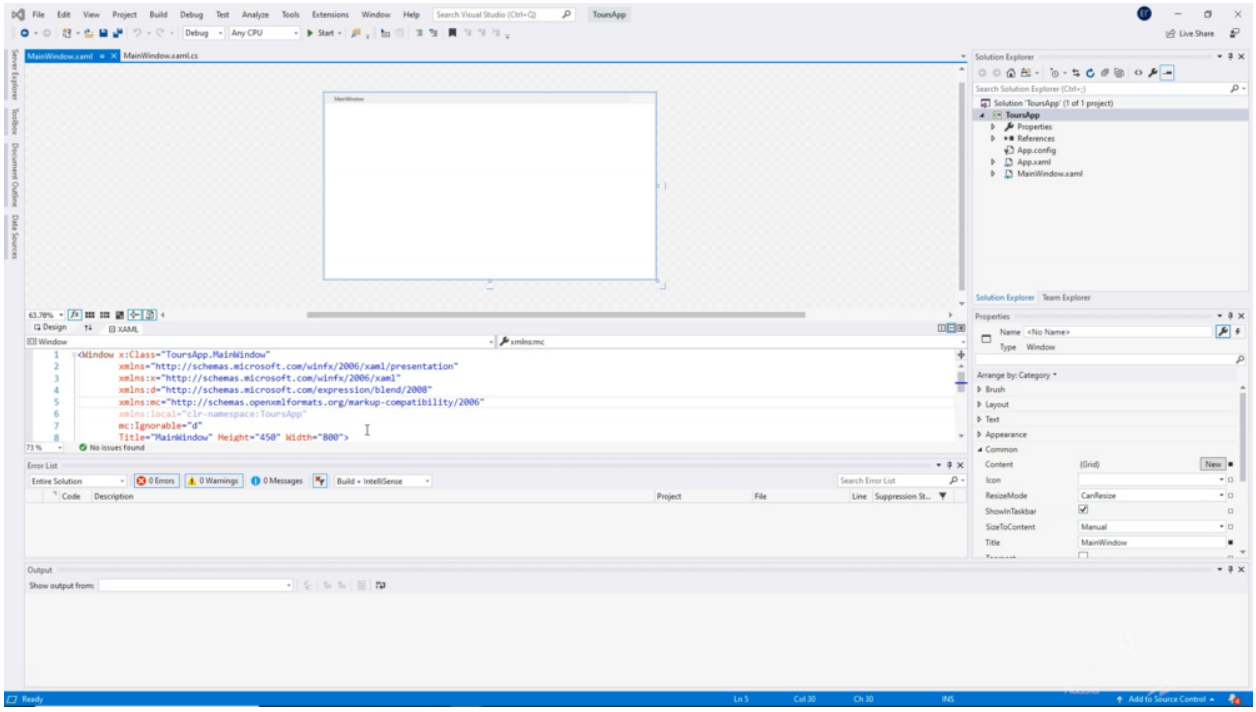


Страничная модель

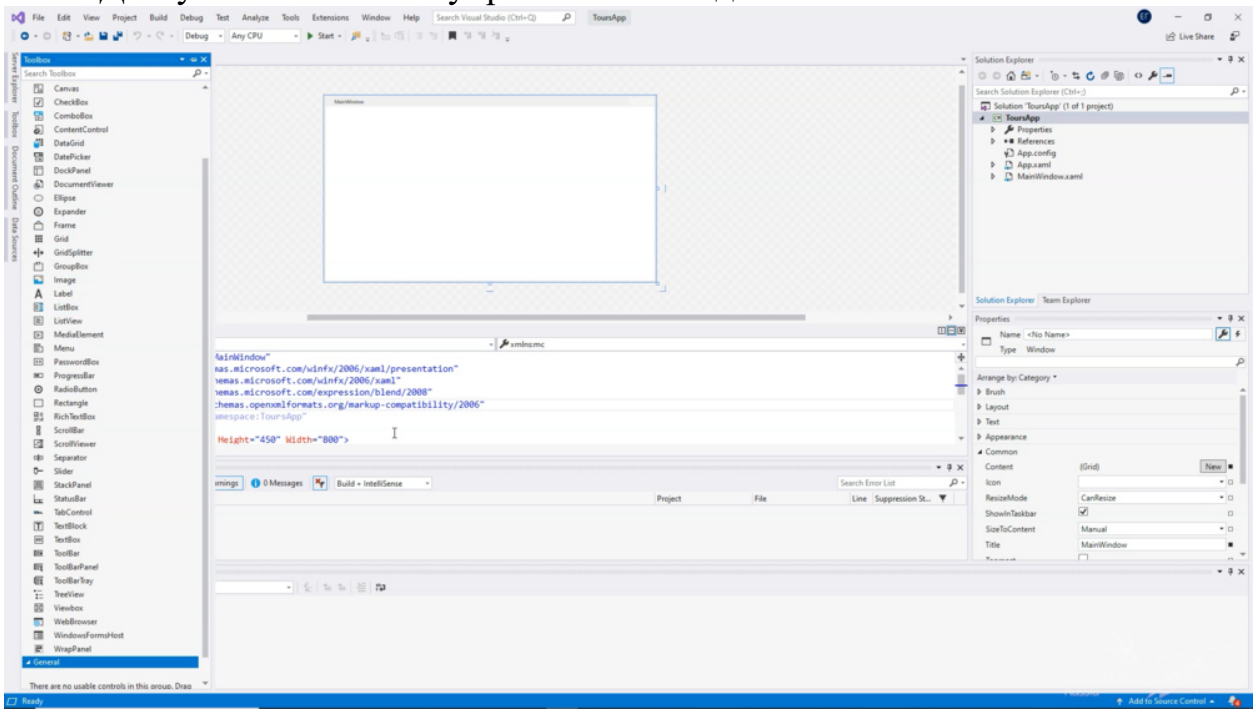


Элементы управления в приложении

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



Доступные элементы управления находятся на панели Toolbox



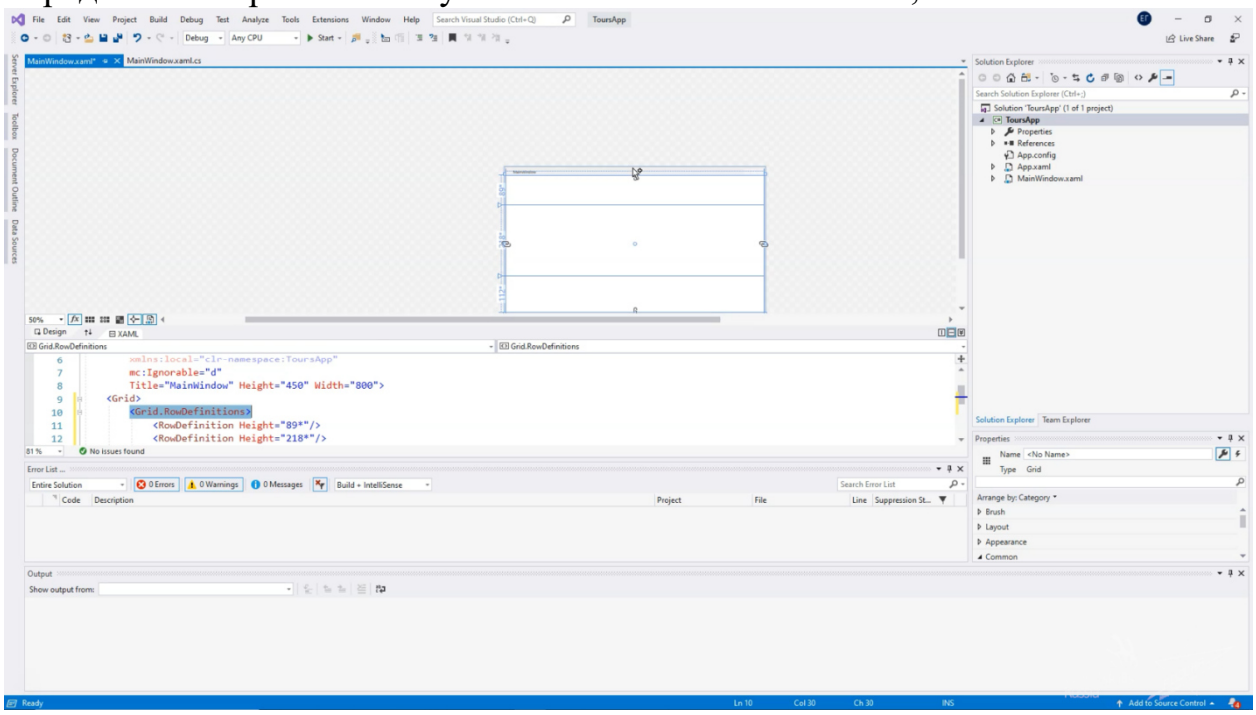
Использование Windows Presentation Foundation (WPF) для создания интерактивных настольных приложений

Контейнеры компановки

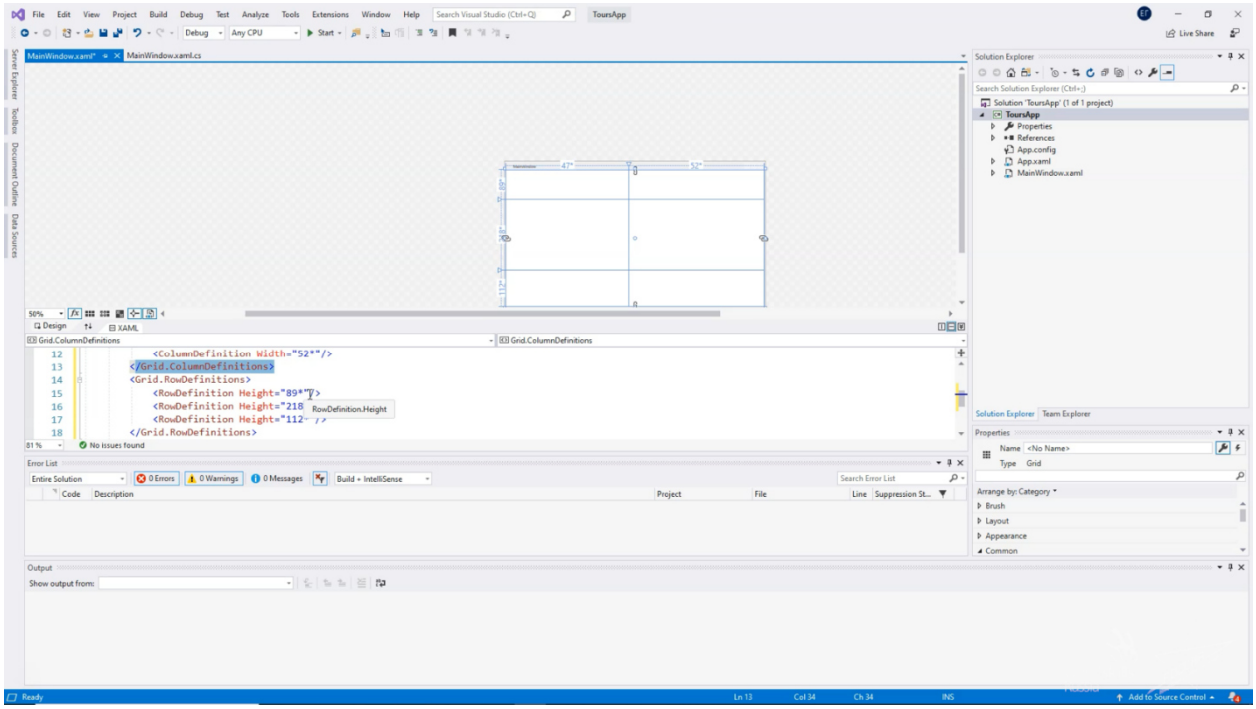
1.Grid

Наиболее мощный и часто используемый контейнер, похожий на таблицу. Он содержит столбцы и строки, количество которых можно задать. Для

определения строк используется свойство RowDefinition,

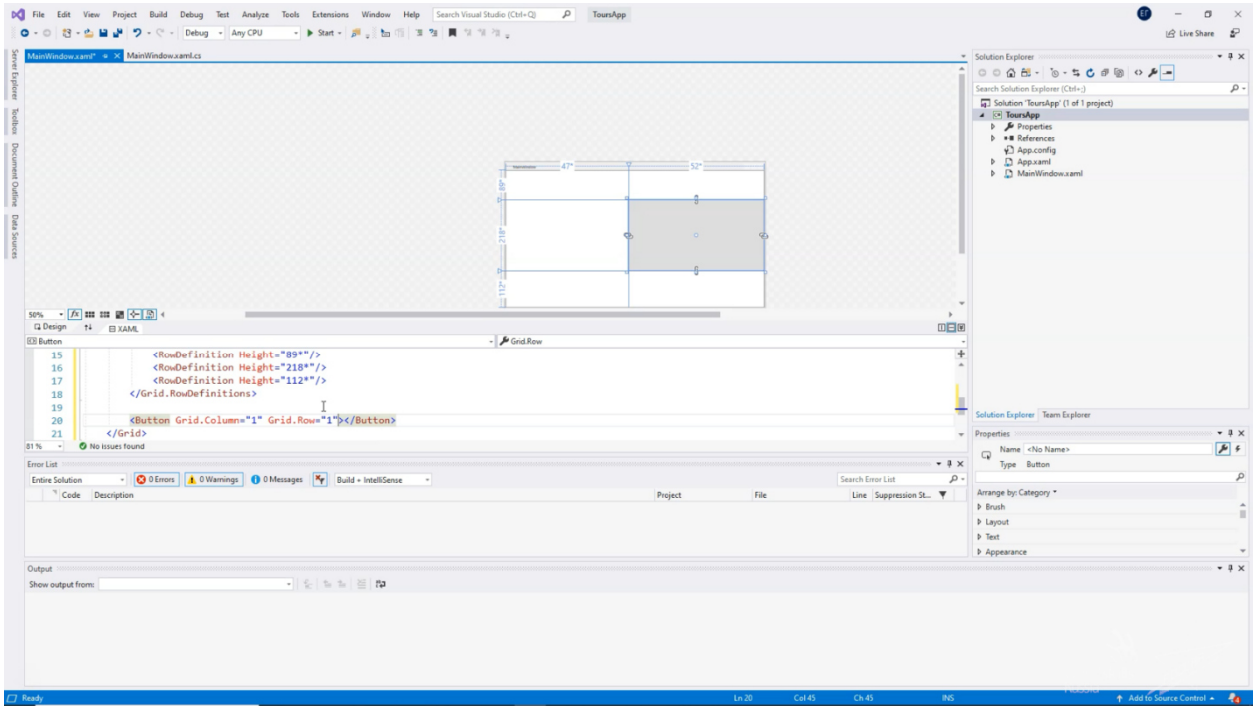


а для столбцов — ColumnDefinition



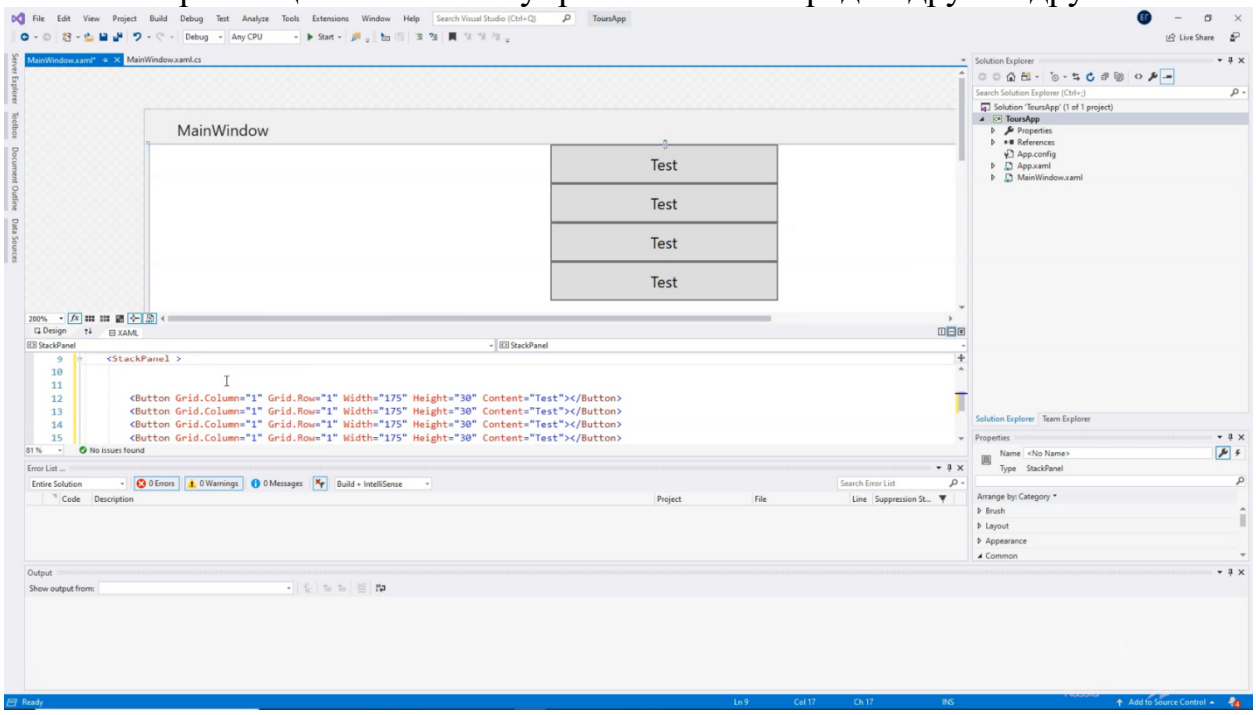
Важно

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

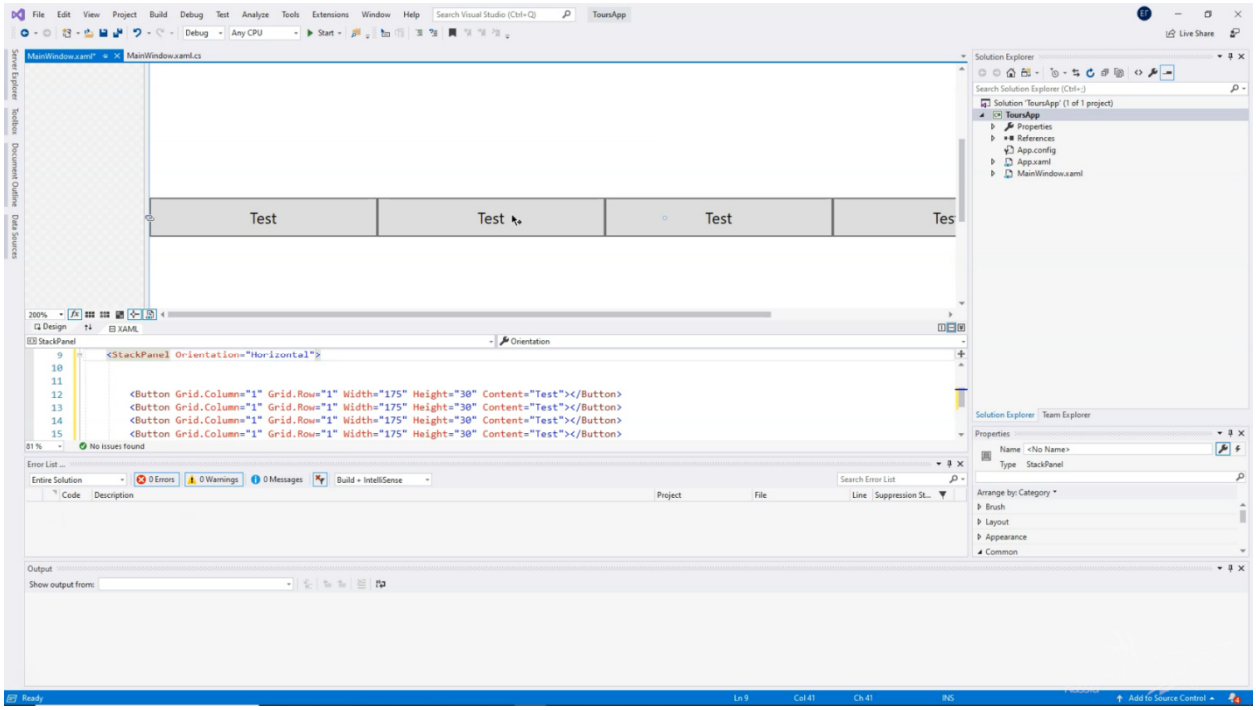


2.StackPanel

Позволяет размещать элементы управления поочередно друг за другом

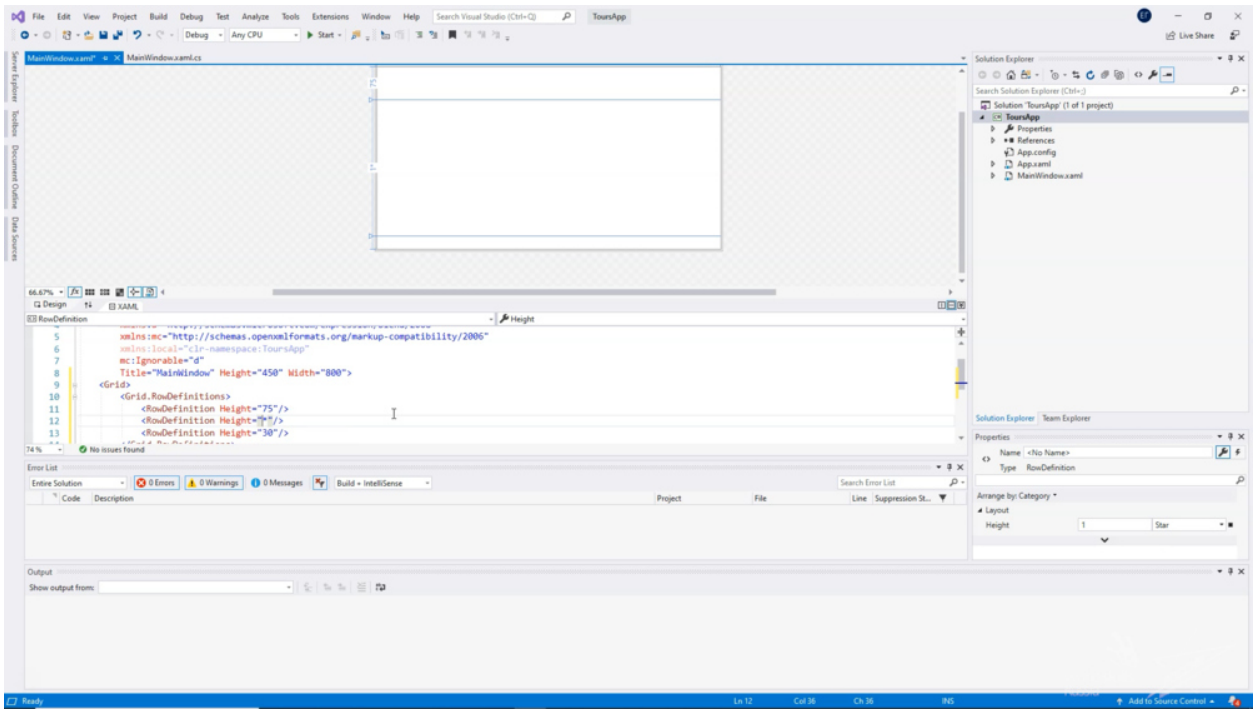


Также существует возможность выбора ориентации размещения с помощью свойства Orientation



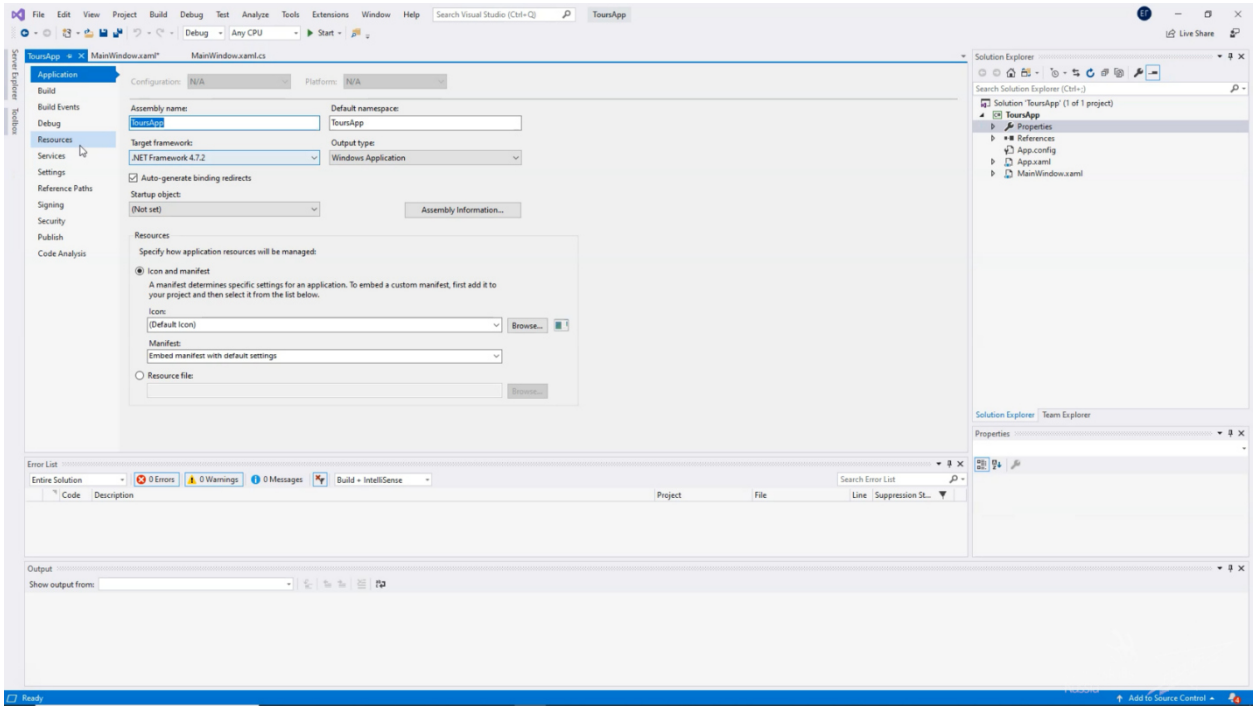
Стилизация приложения

Перед стилизацией разметим окно приложения с помощью контейнера Grid

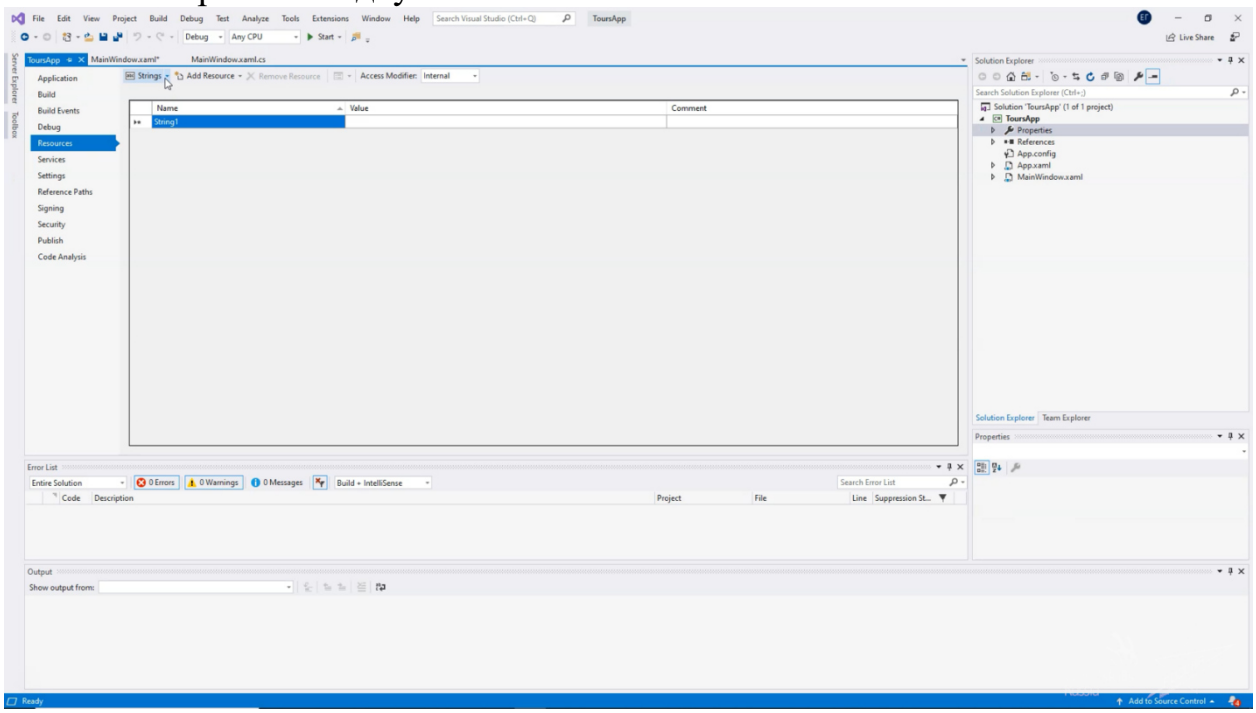


Добавим логотип приложения в Resourses

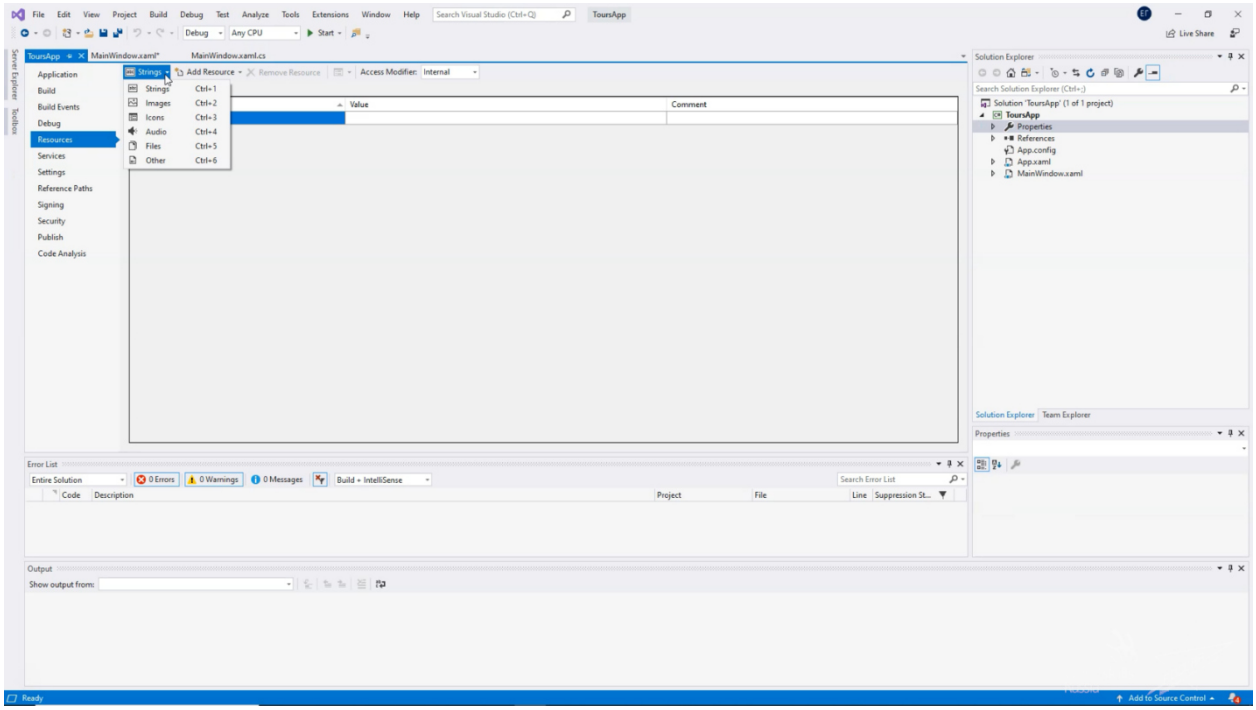
1. Выбрать вкладку ToursApp



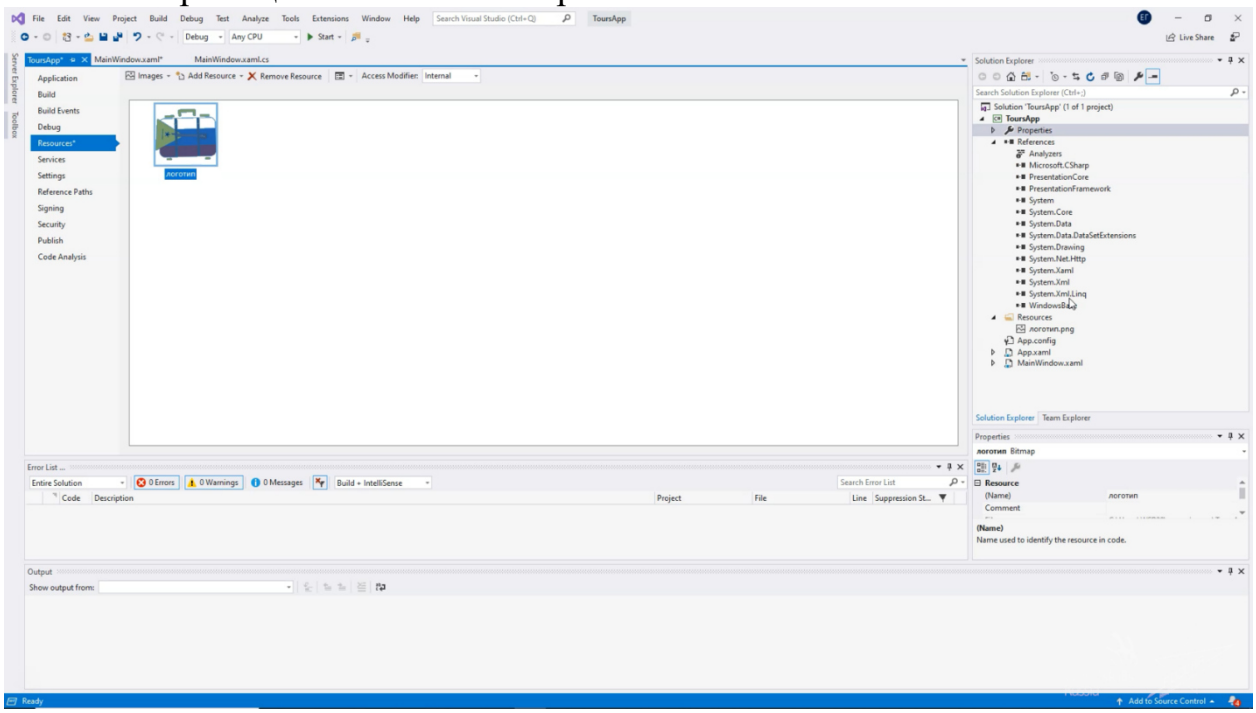
2. Открыть вкладку Resources



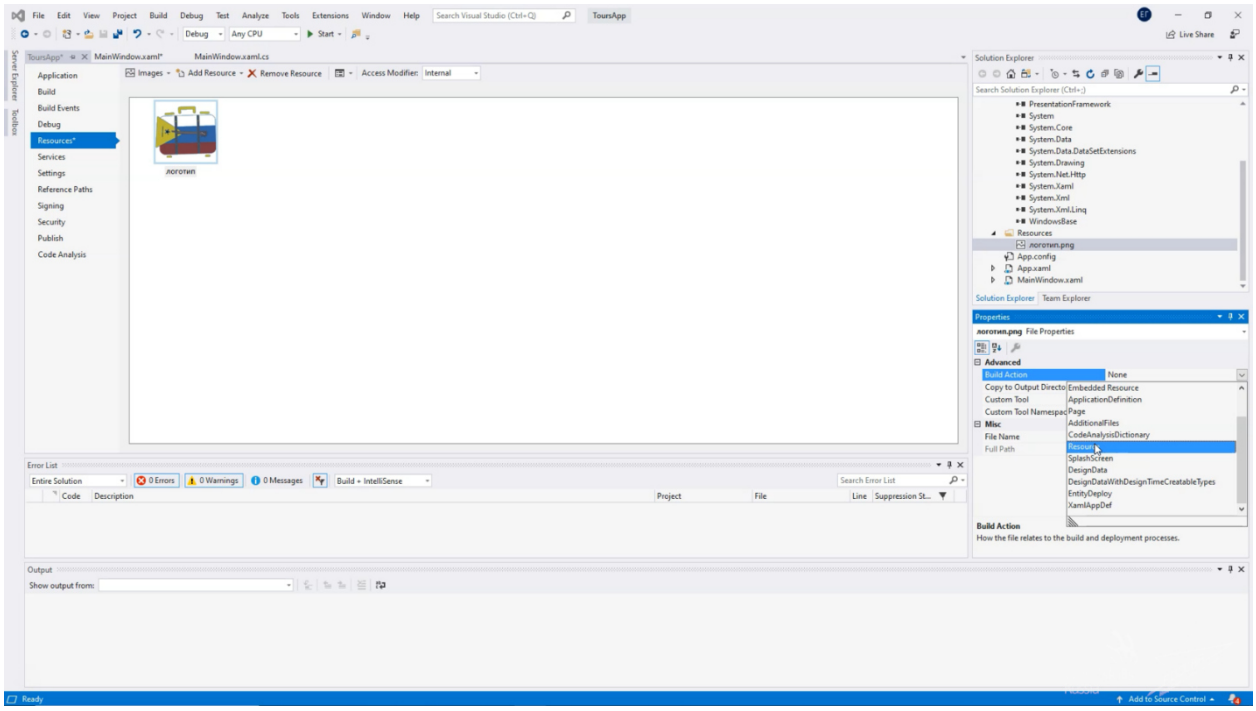
3. Выбрать список изображений



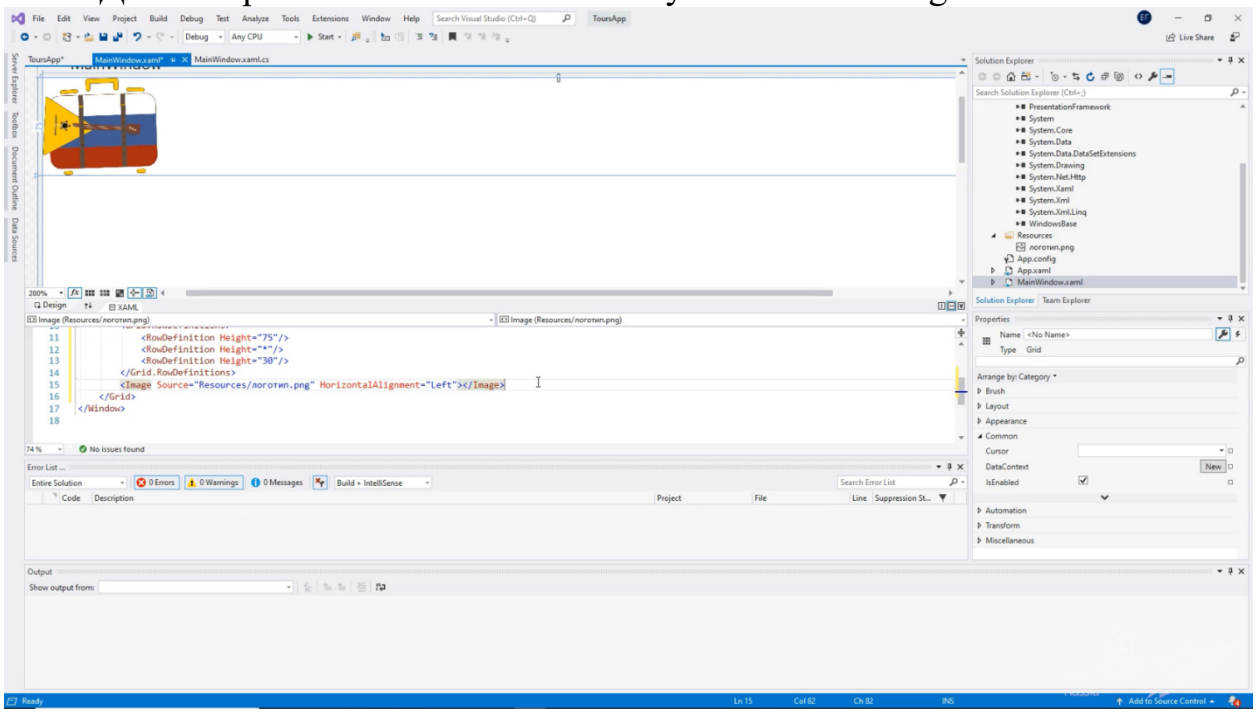
4. Перетащить логотип в открывшееся окно



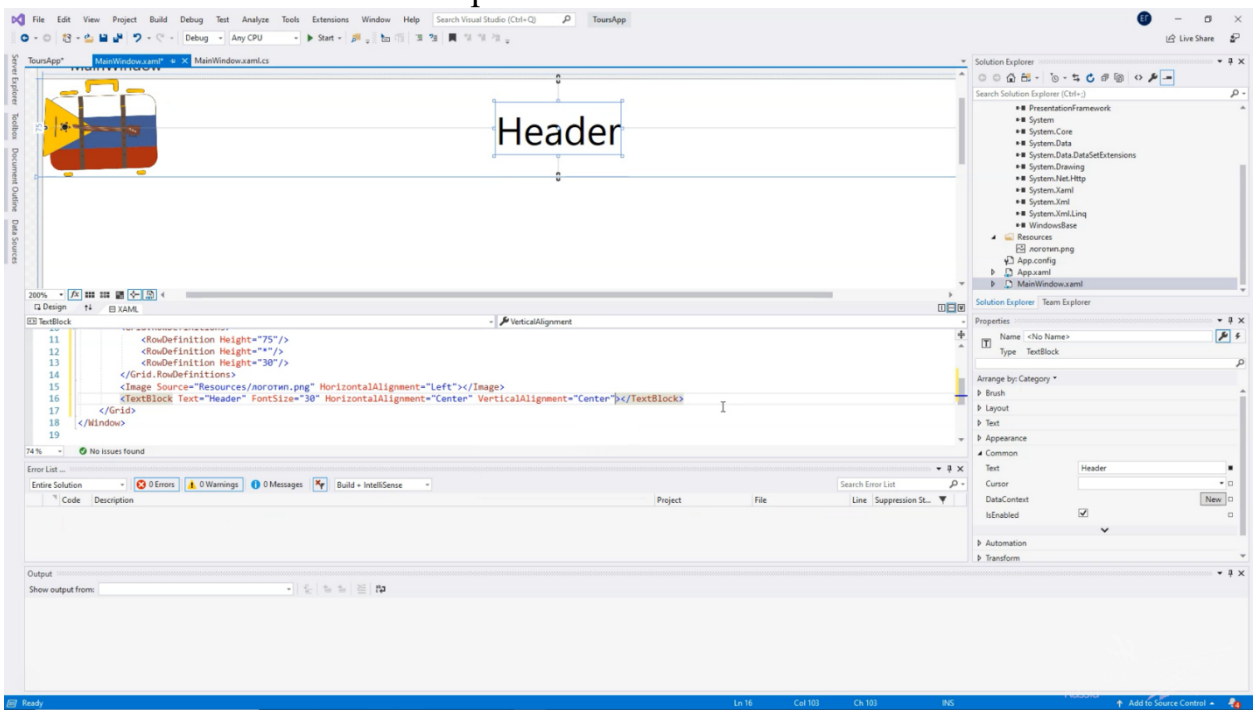
5. Установить значение поля Build Action — Resource



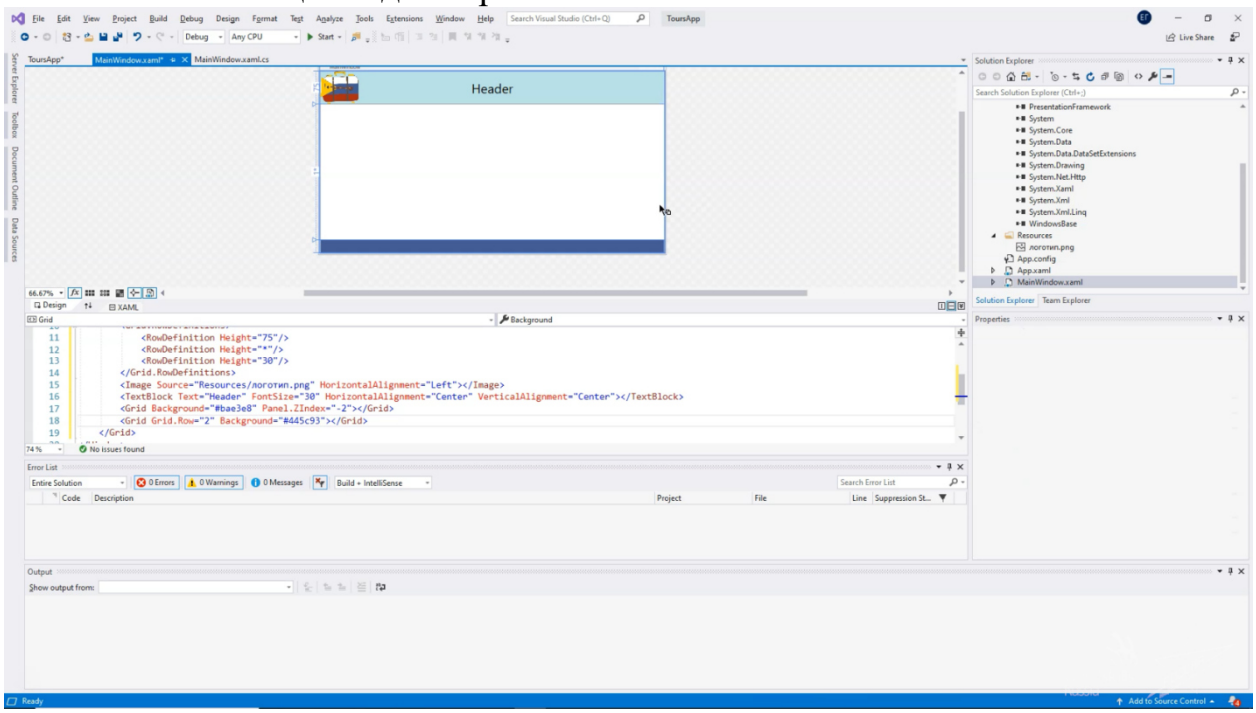
Для отображения логотипа используем элемент Image



Установим заголовок приложения

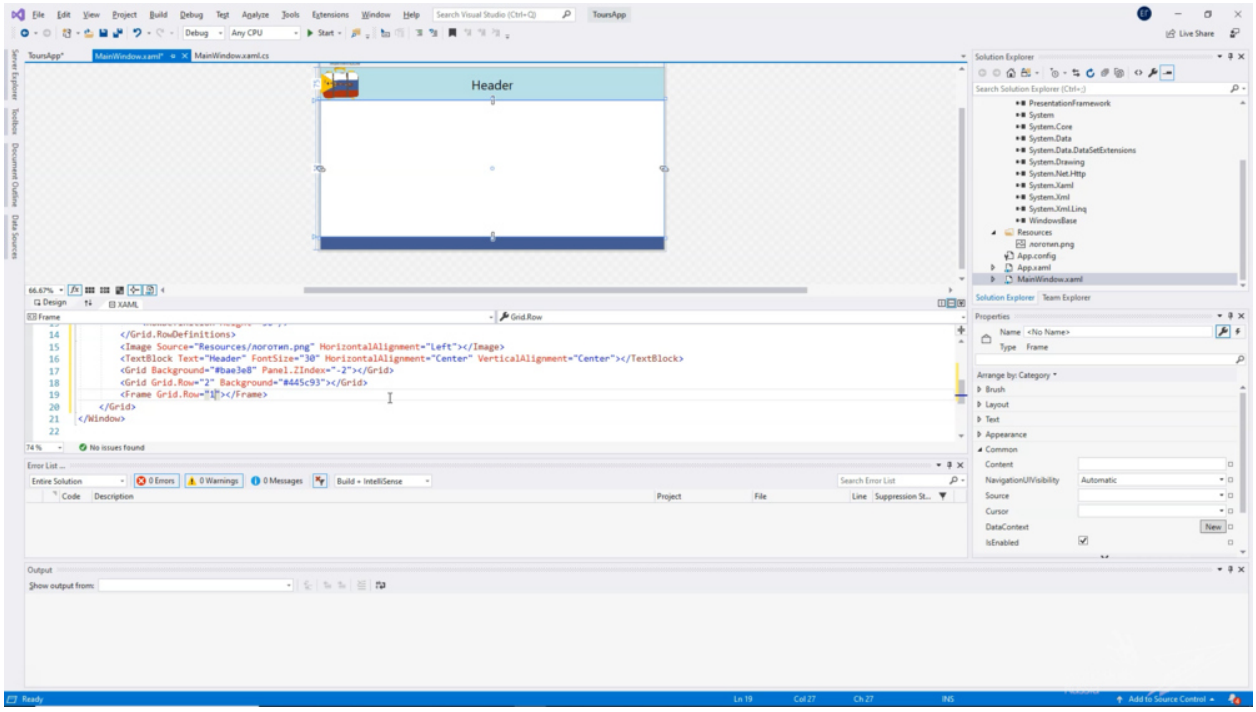


Установим цвета для верхней и нижней частей



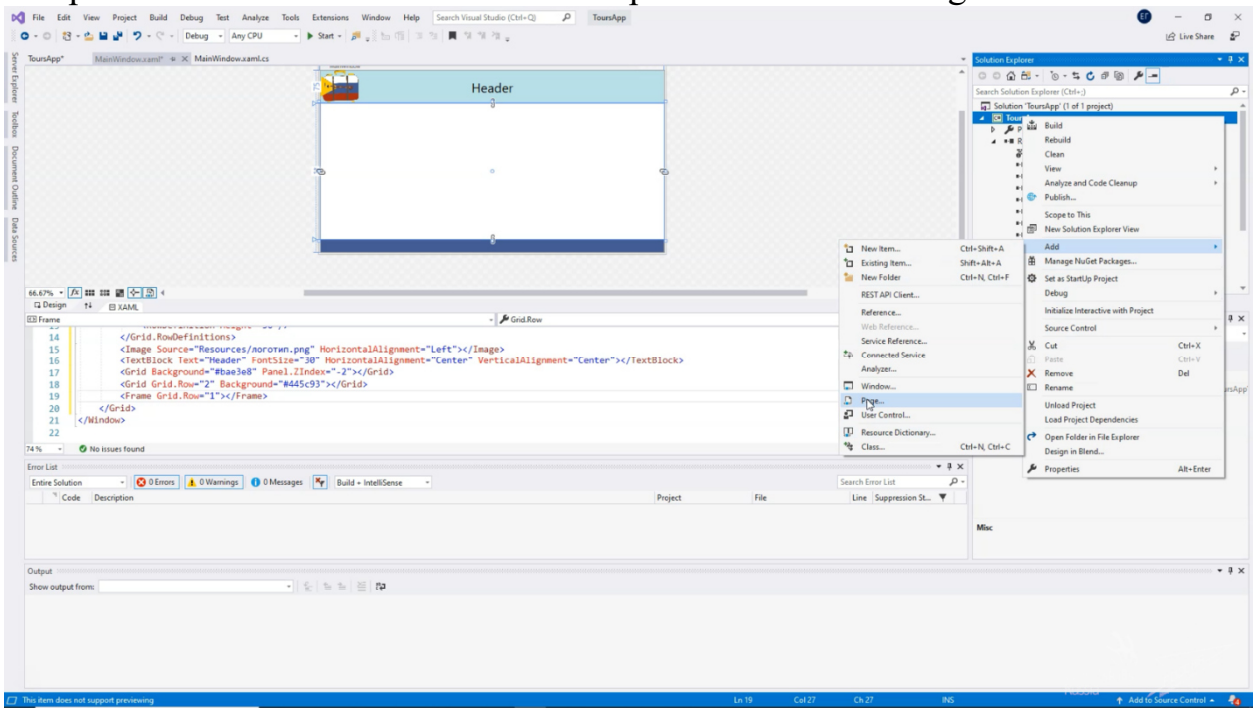
Создание навигации

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

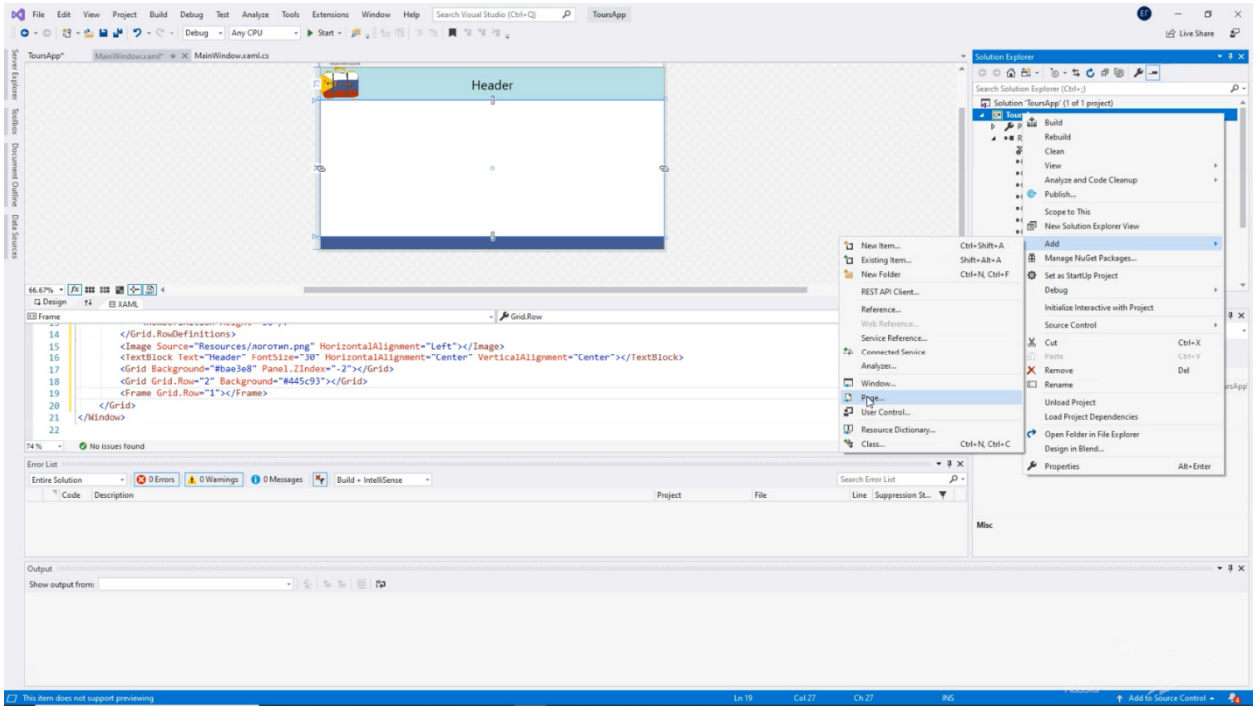


Далее создадим страницу, которая будет отображаться при первом запуске приложения, и вторую страницу для тестирования навигации между ними

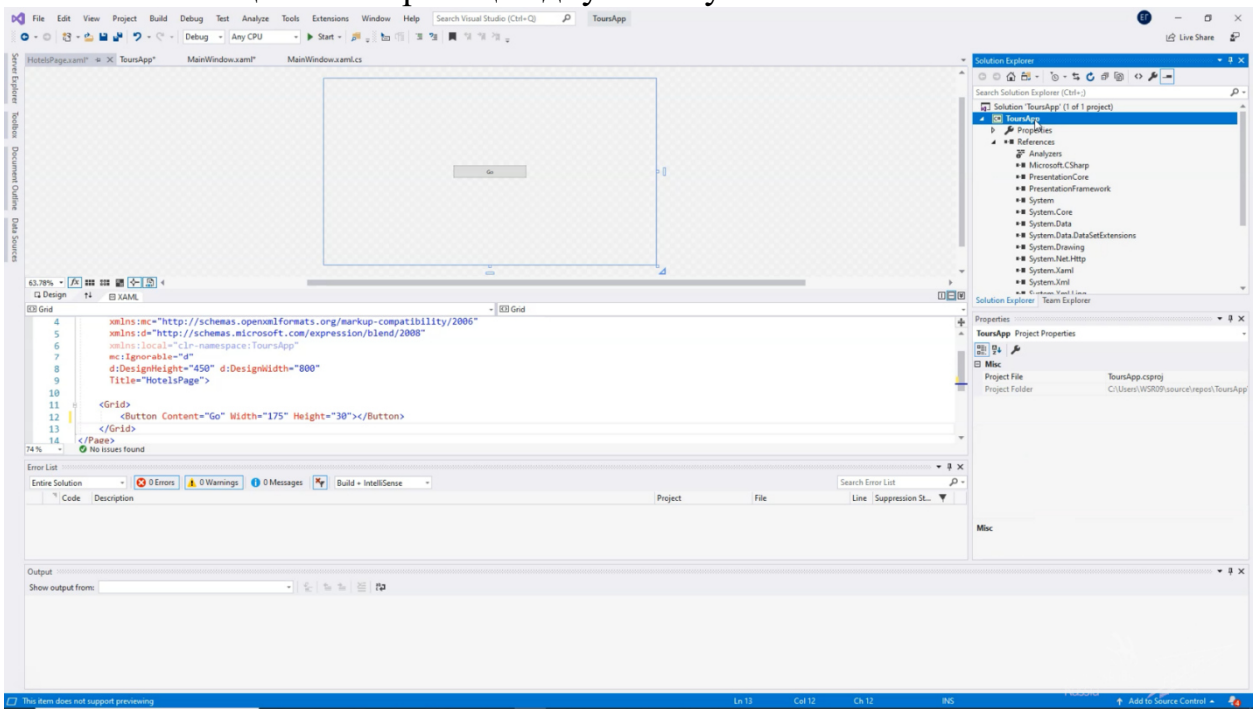
1. Правой кнопкой жмем на название проекта — Add — Page



2. Указываем ее название, например, HotelsPage



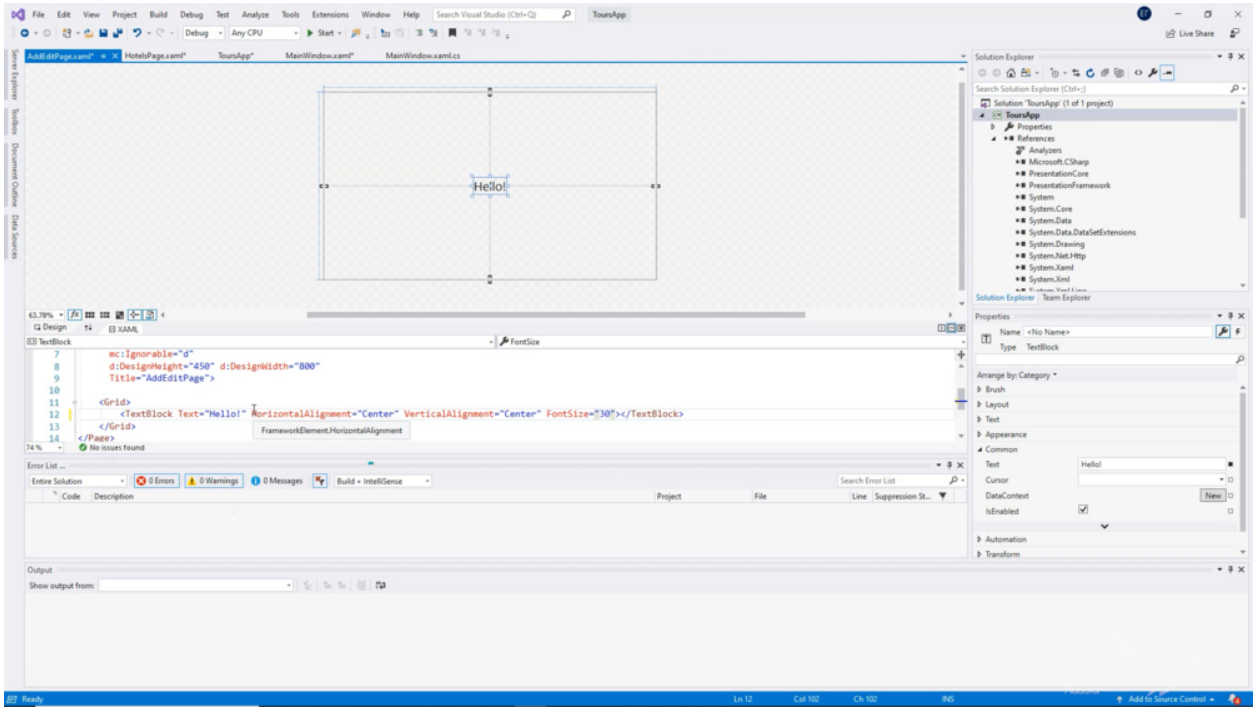
3. Размещаем на странице одну кнопку



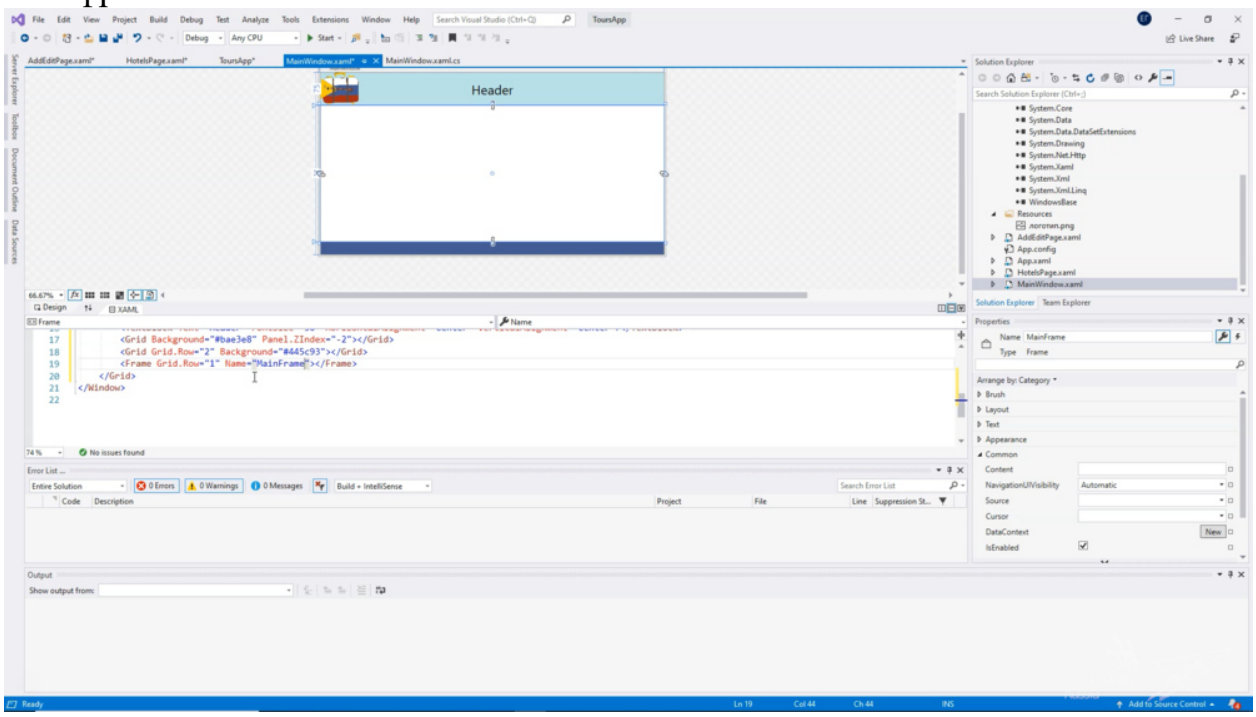
4. Создаем еще одну страницу, повторив пункт 1

5. Даем ей название AddEditPage

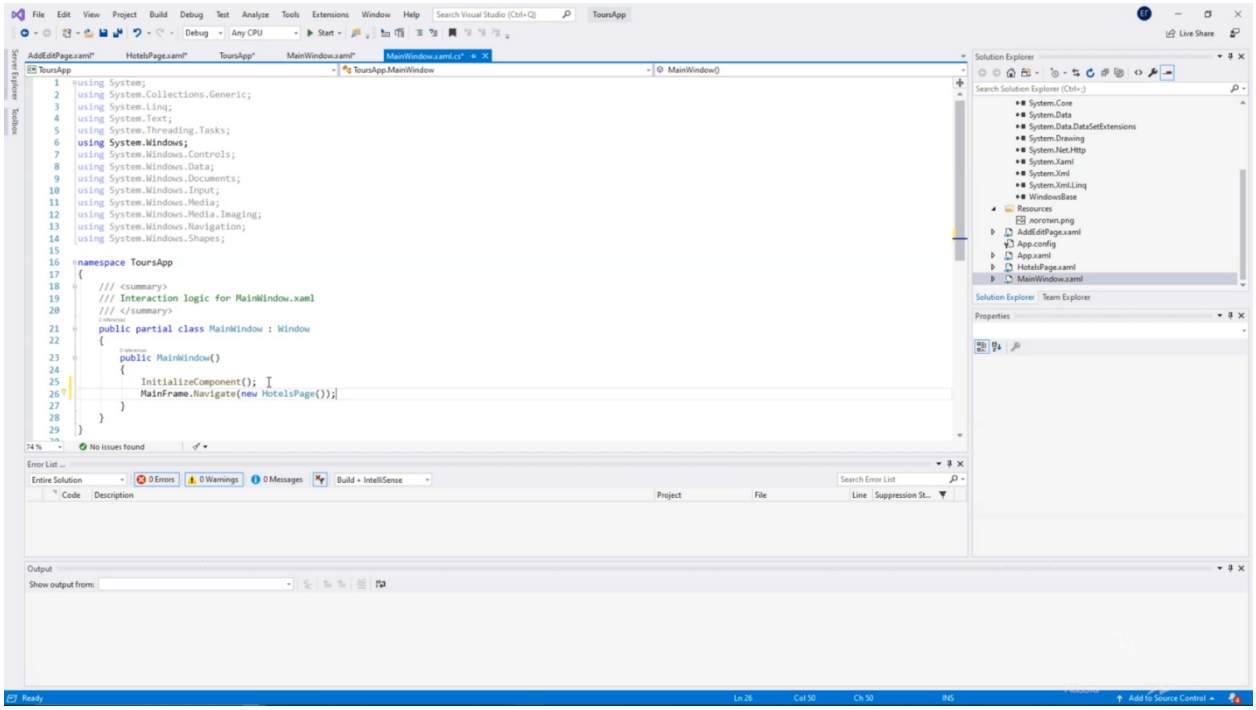
6. Размещаем в ней TextBlock



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());'*

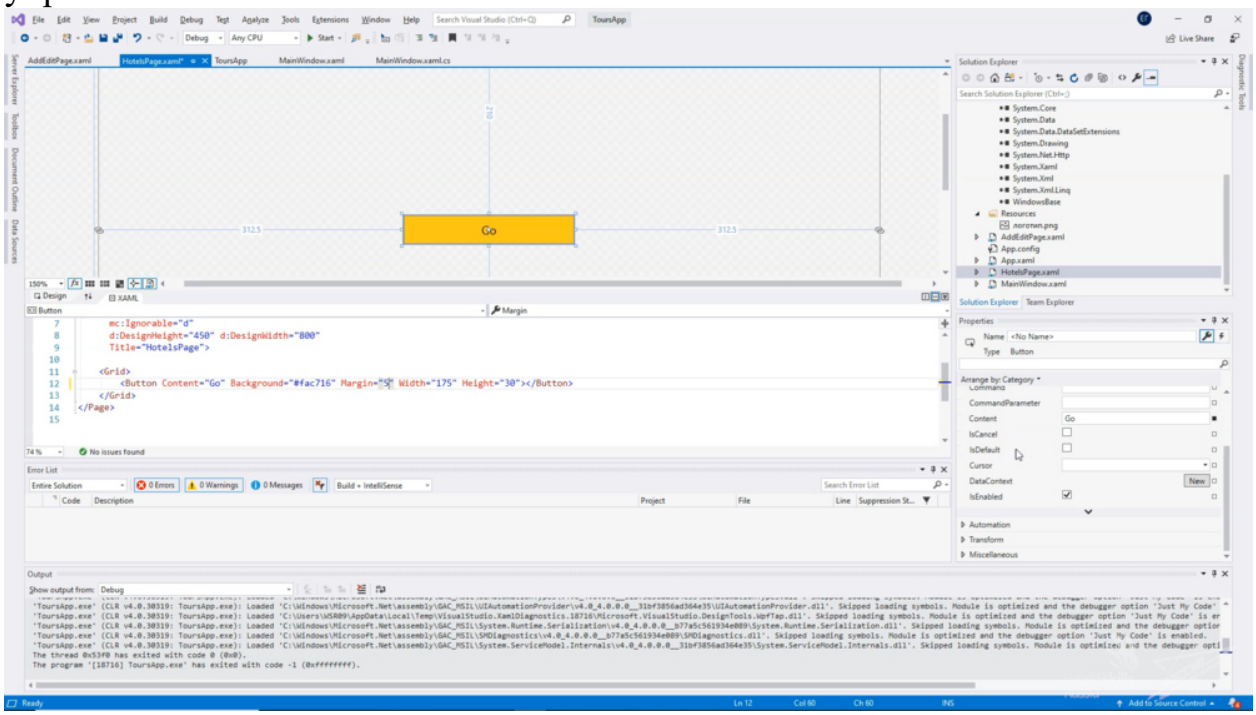


Свойства элементов управления

Например:

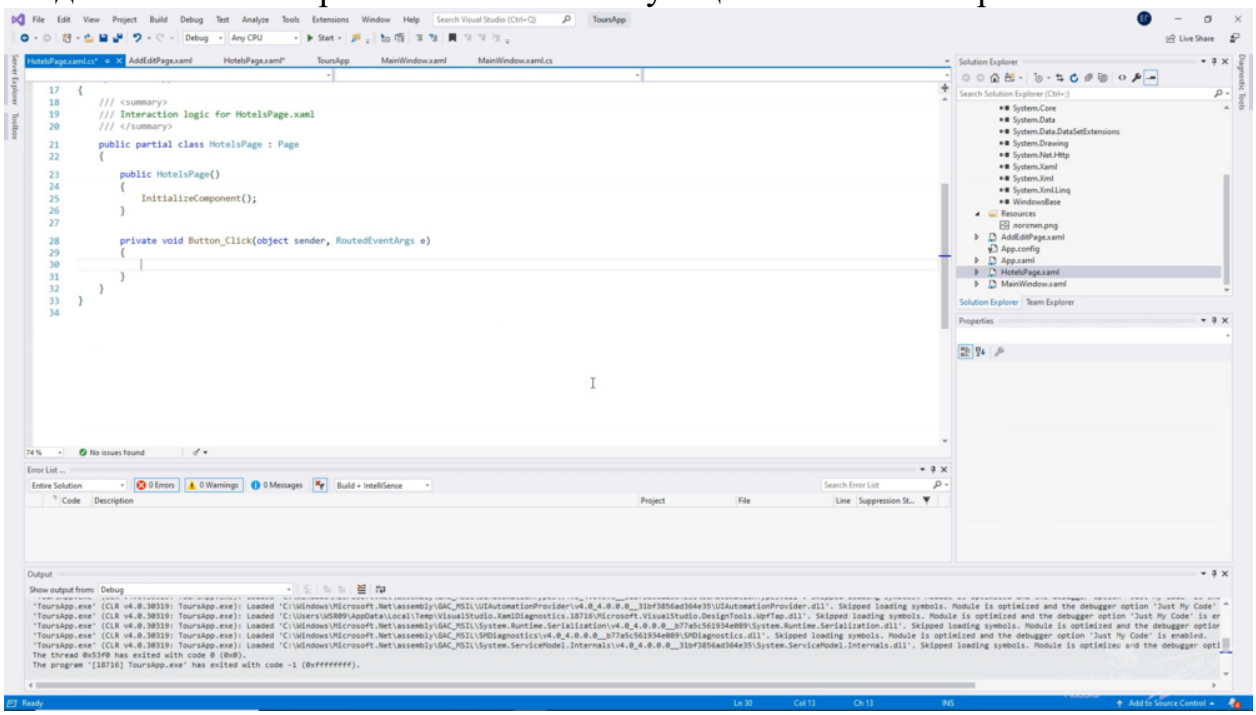
- Background — меняет фоновый цвет элемента

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



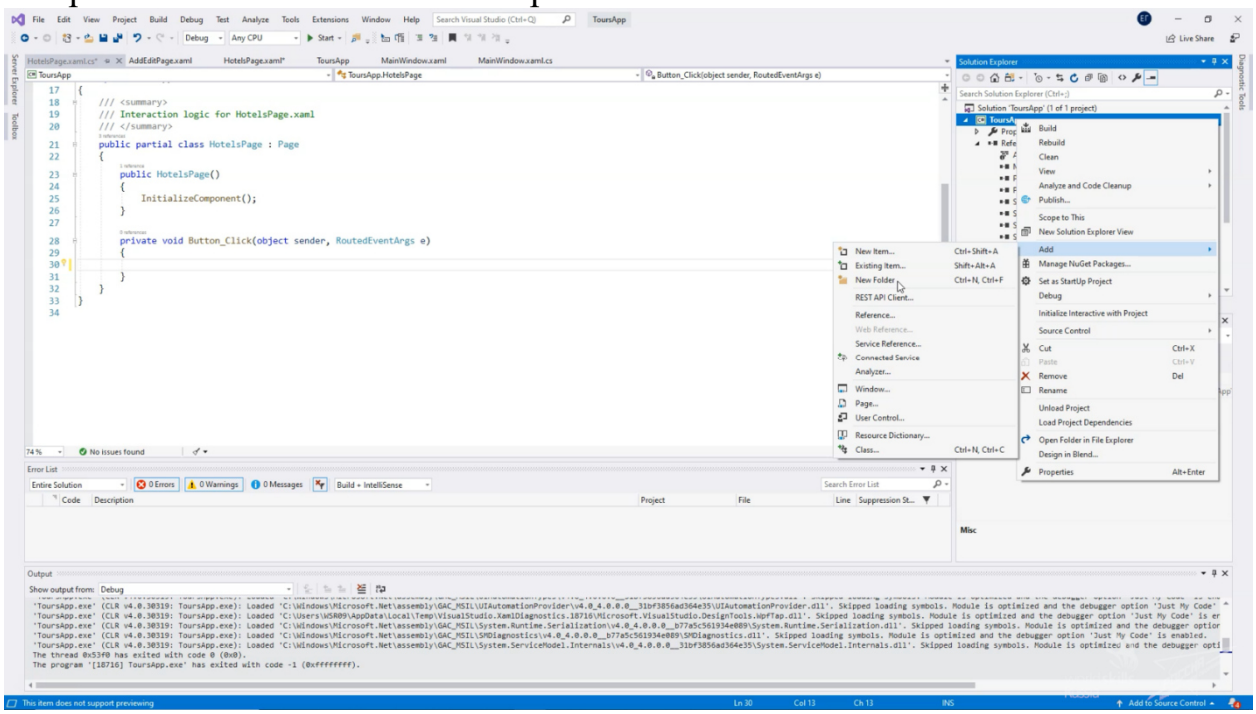
Для взаимодействия с кнопкой будем использовать событие Click(). С помощью него мы сможем перейти с первой страницы на вторую. Для

создания события пропишем соответствующее значение в верстке:

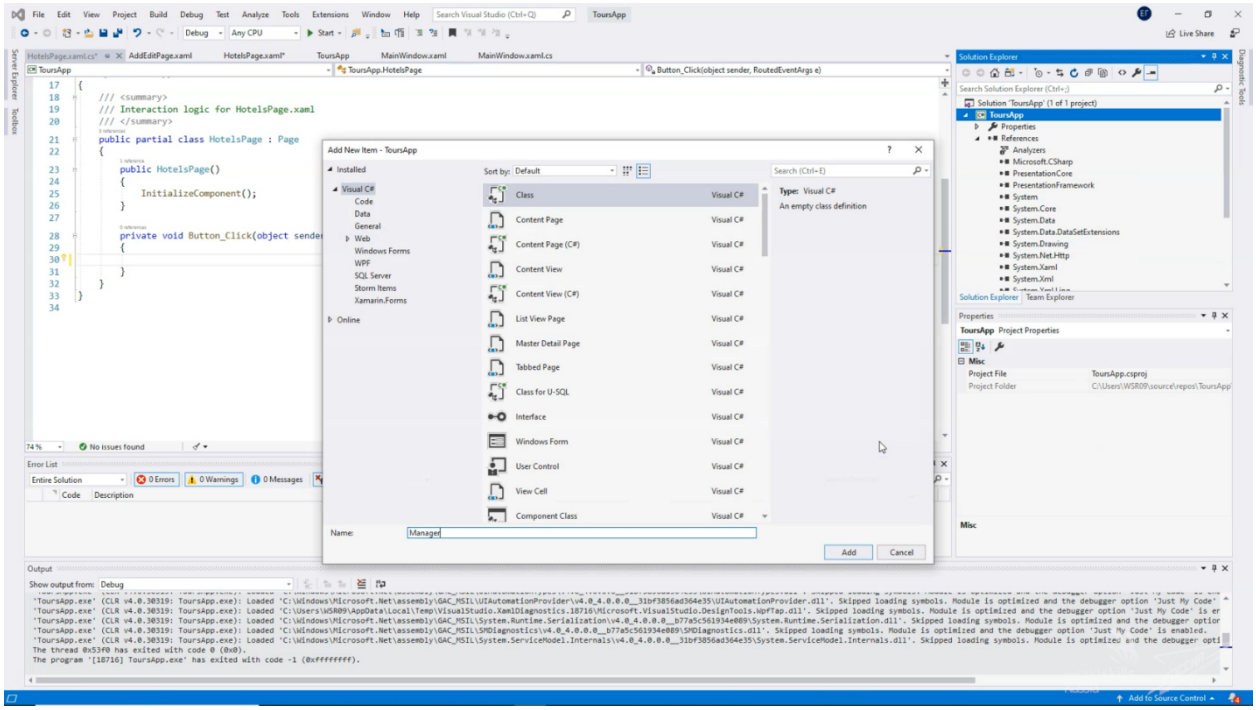


Далее нажмем F12 и попадем в окно обработки нажатия на кнопку. Чтобы добраться до MainFrame для навигации, необходимо создать новый класс:

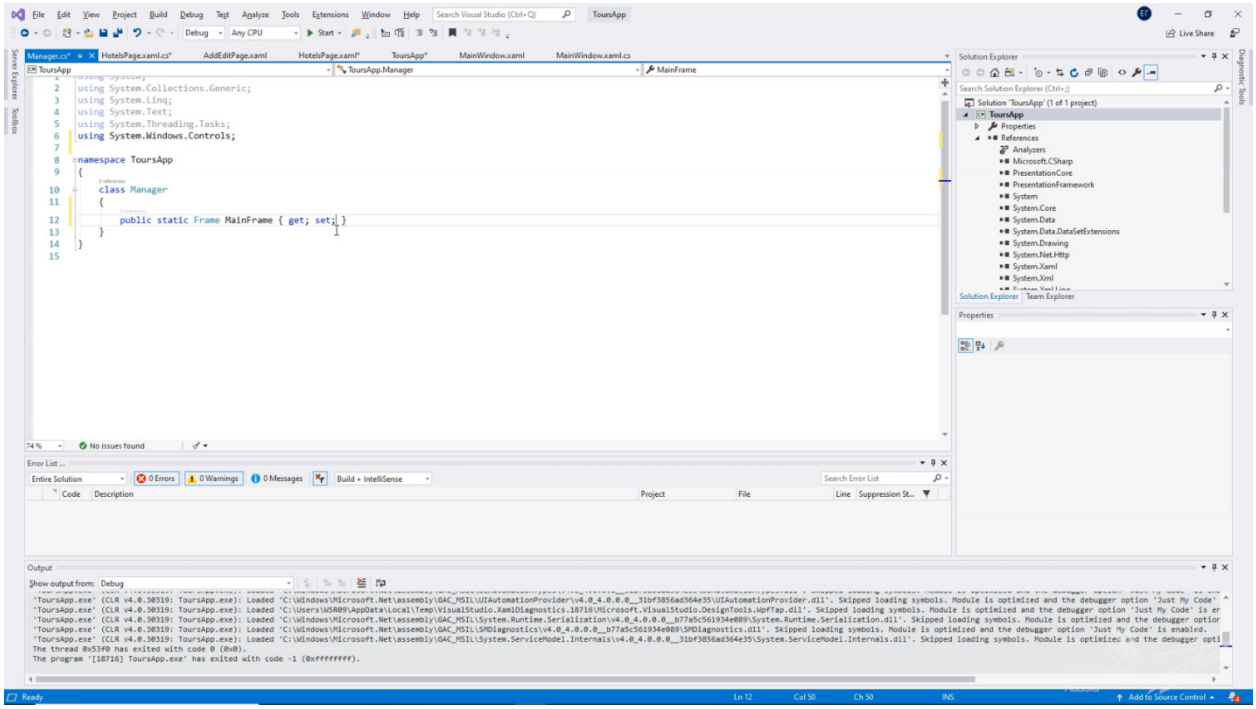
1. Правой кнопкой на название проекта — Add — Class



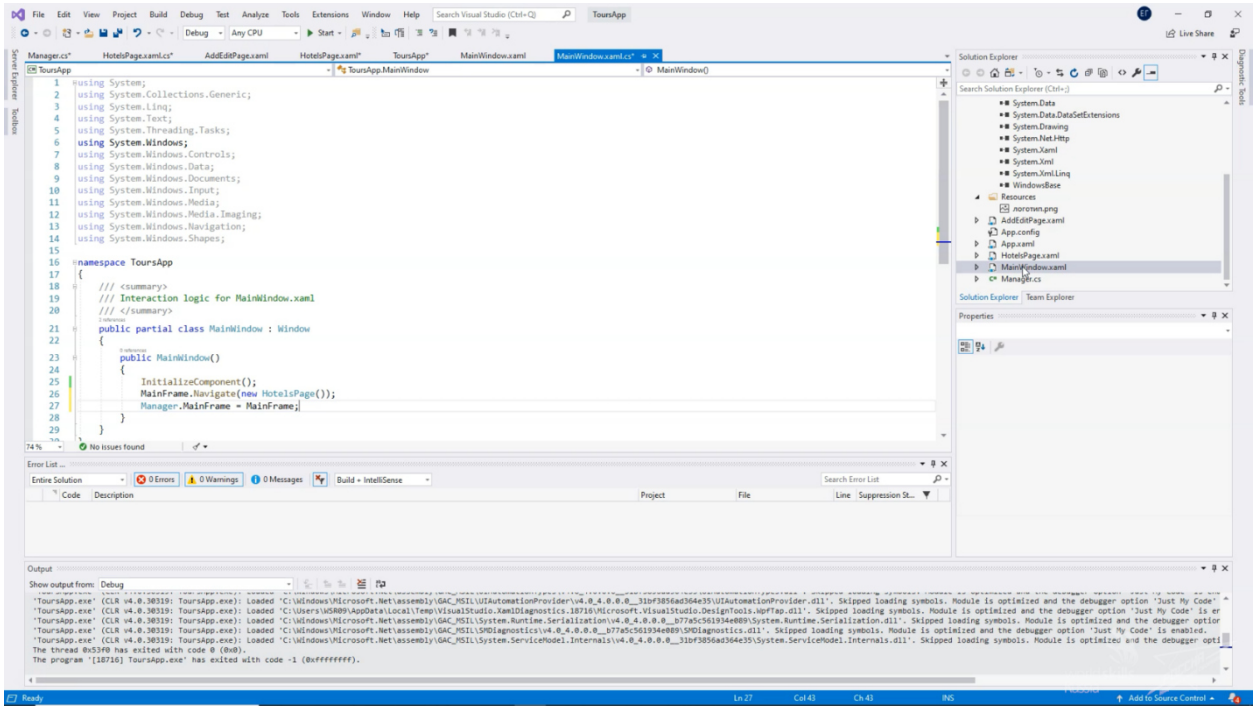
2. Вводим название класса



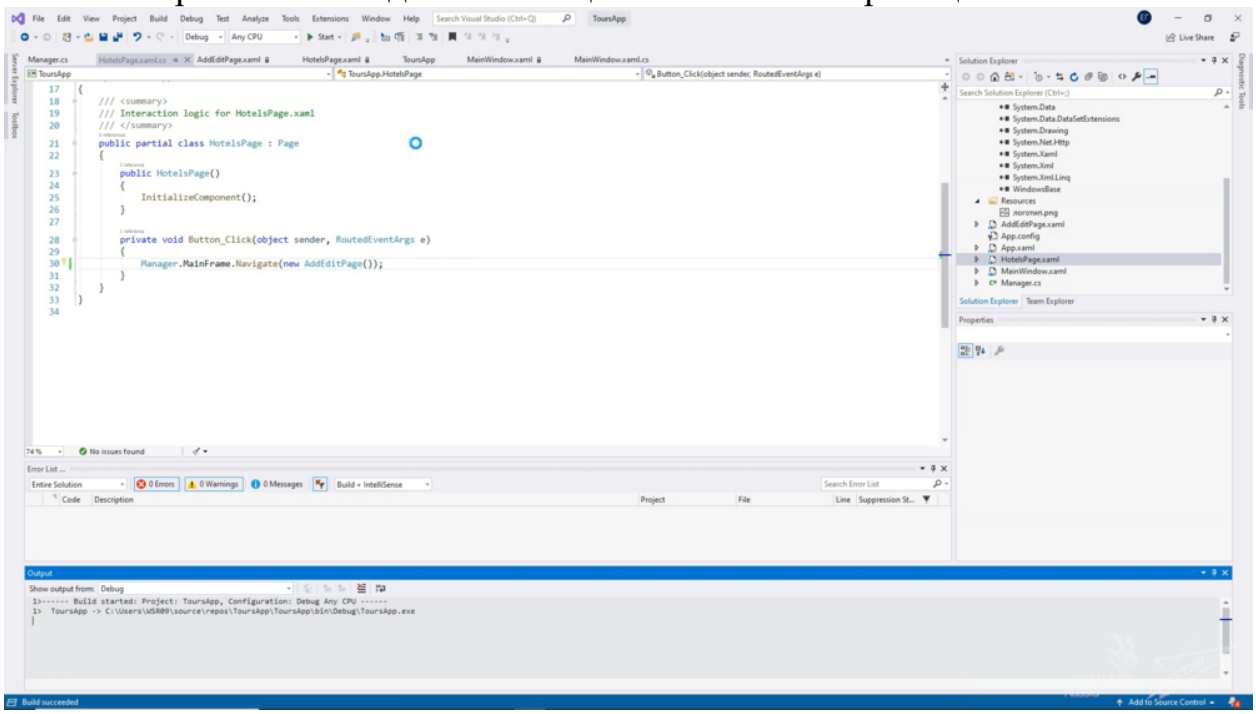
3. Создаем статичное свойство MainFrame



4. Присваиваем ему значение в MainWindow

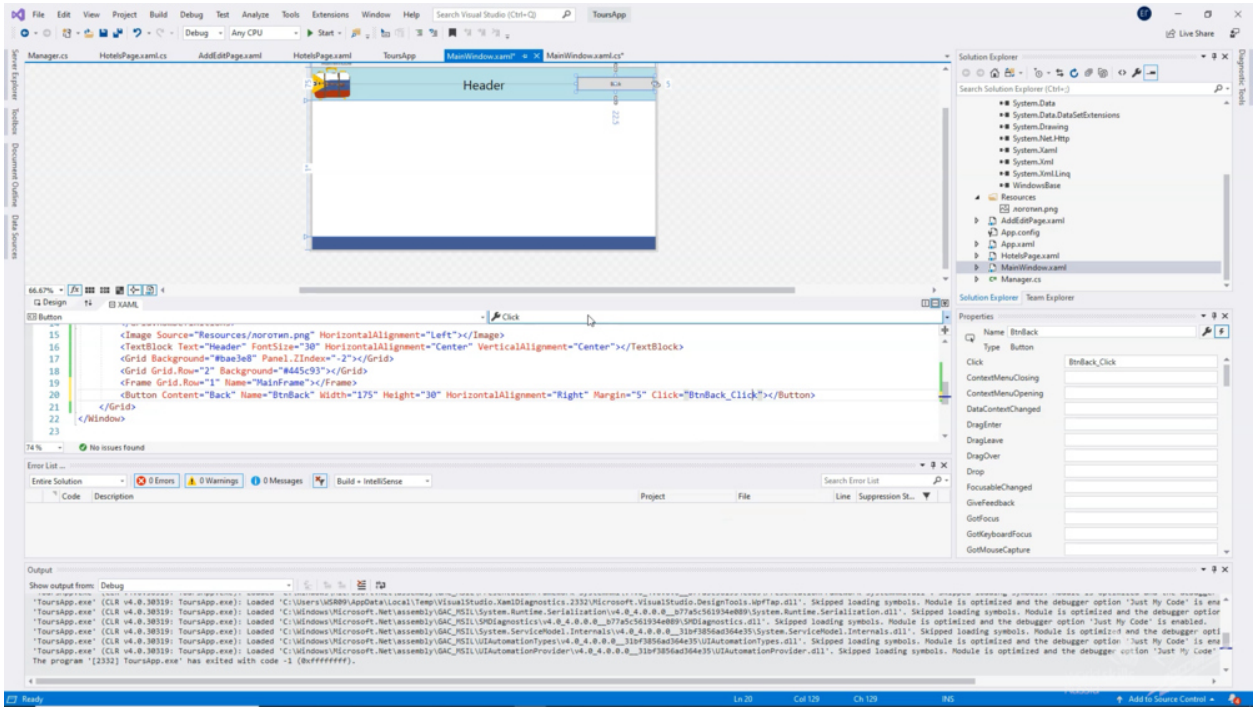


Теперь можем создать навигацию на главной странице

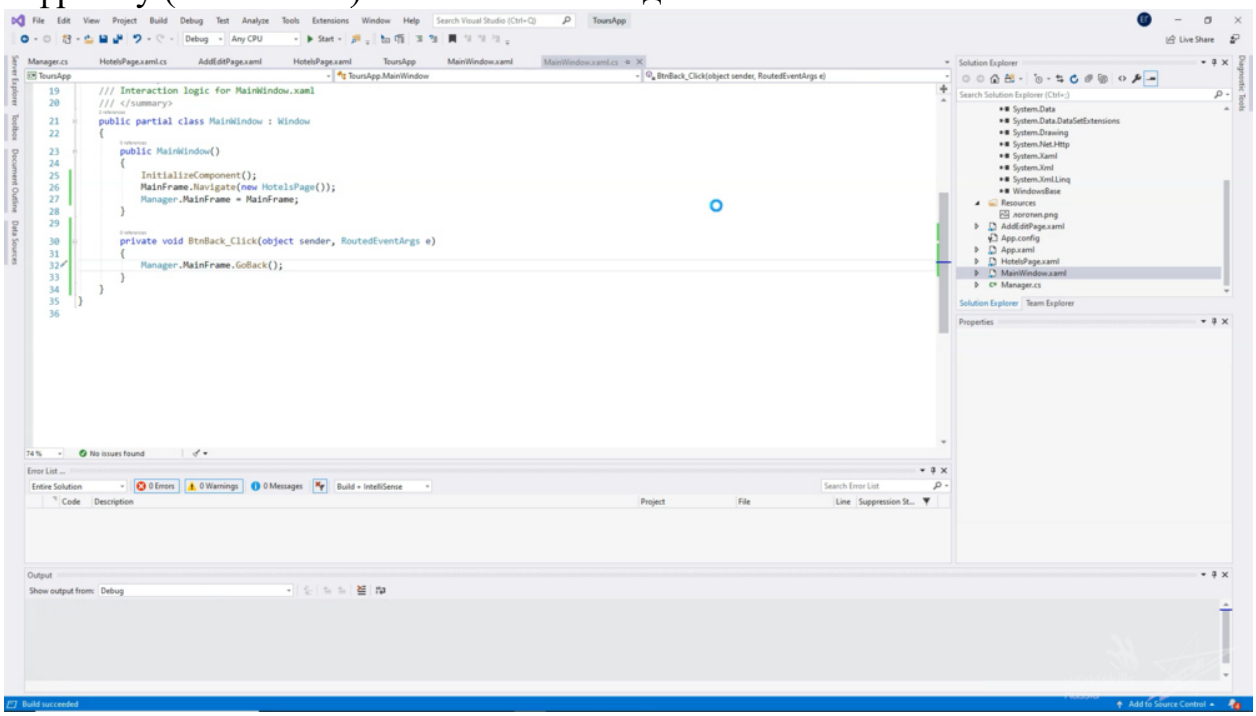


Добавим кнопку «Назад» на главном окне приложения. Установим для нее следующие свойства:

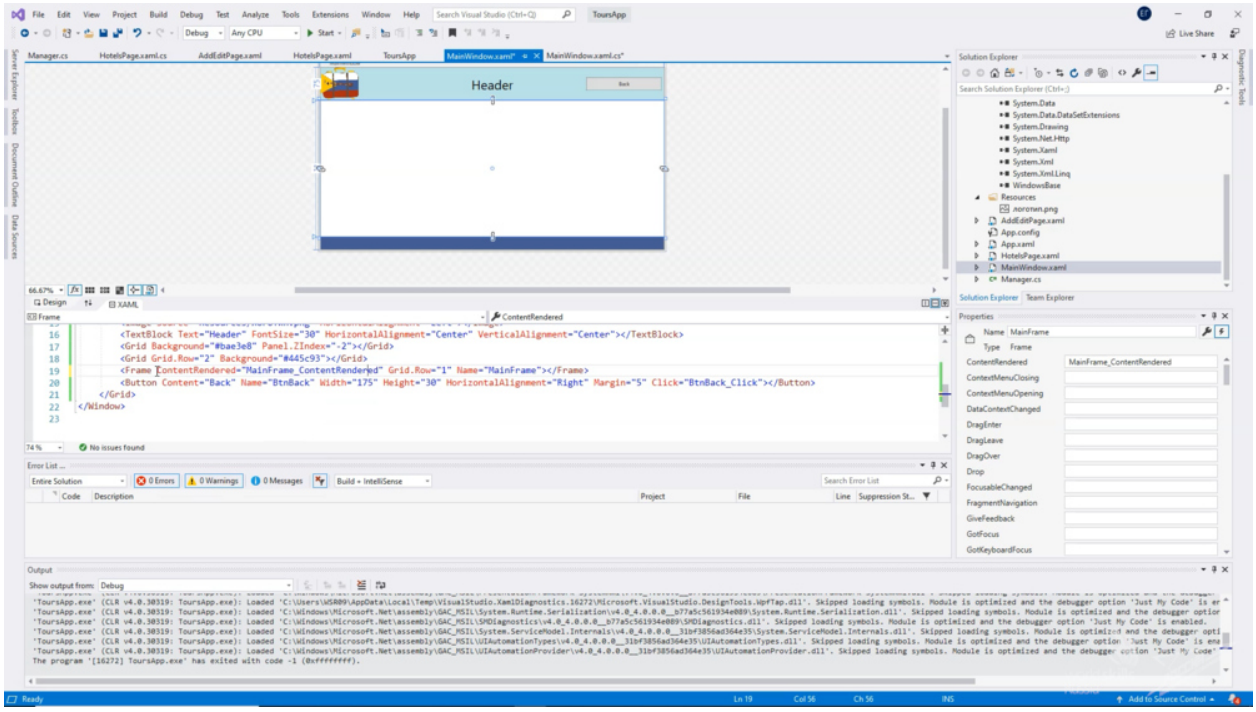
1. Name
2. Width, Height
3. HorizontalAlignment, Margin



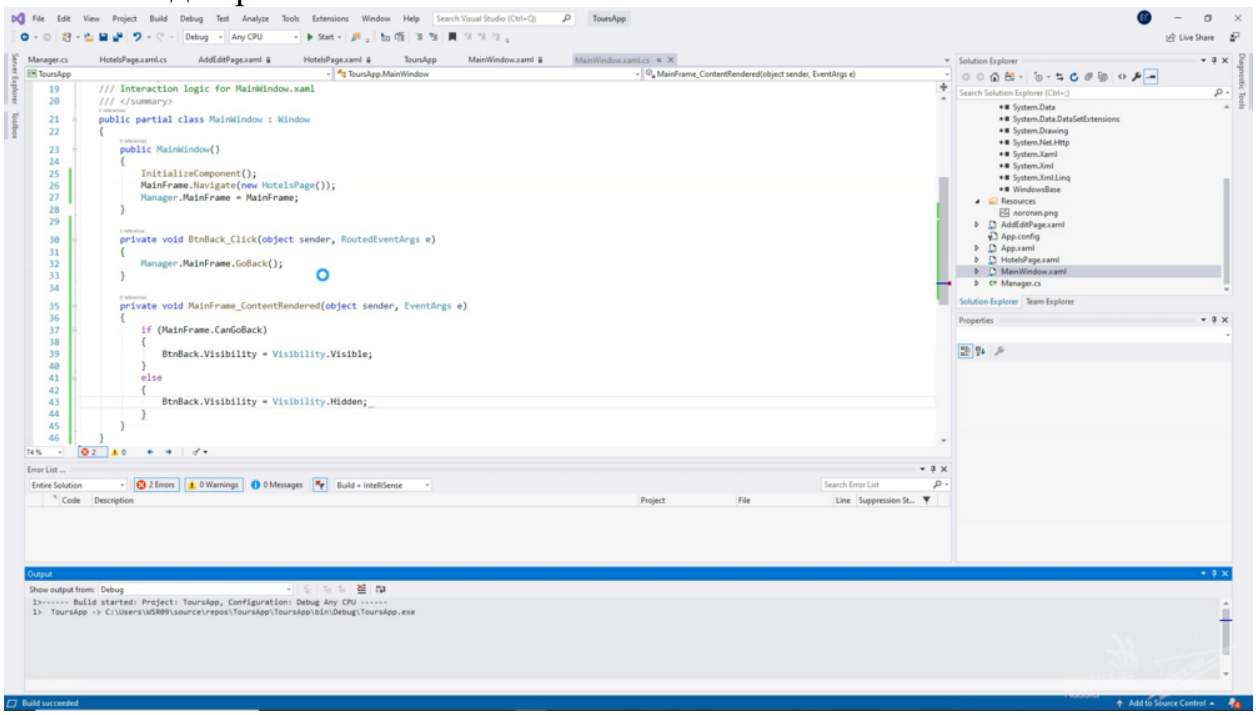
Далее нажимаем F12 и попадаем в окно обработки нажатия на кнопку. Используем следующую логику: обращаемся к менеджеру (Manager), к фрейму (MainFrame) и вызываем метод GoBack



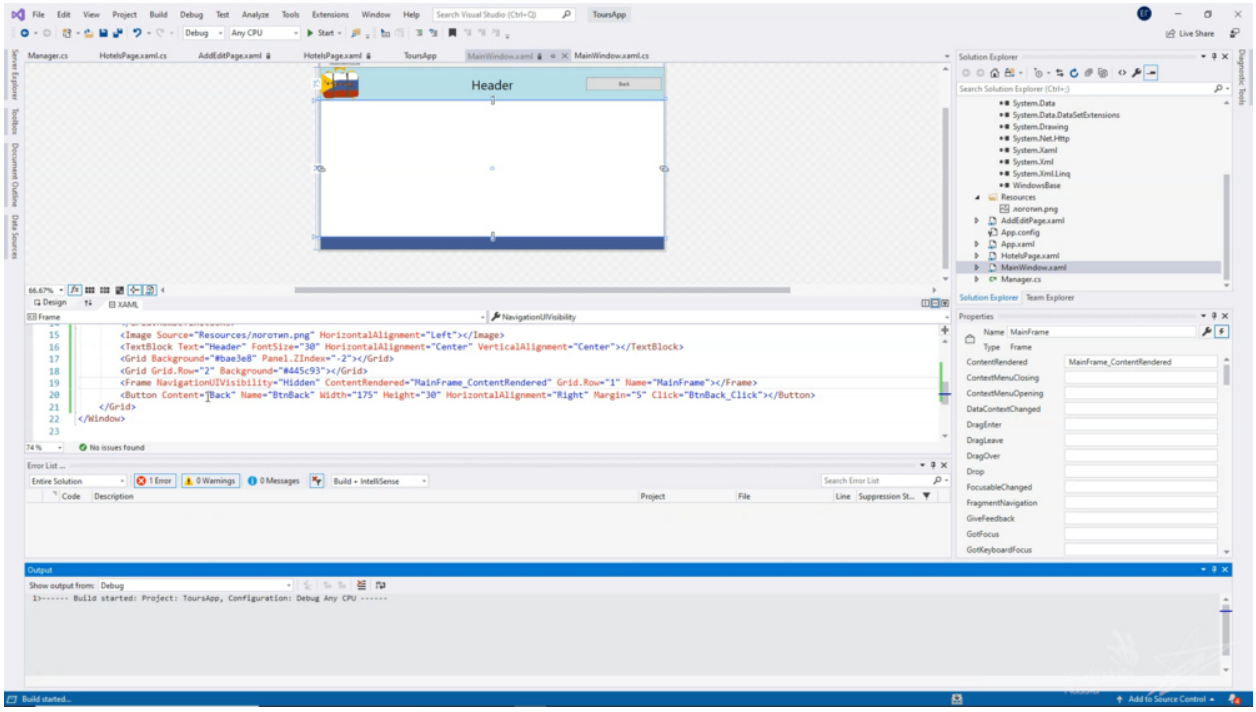
Но она не нужна на главном экране. Чтобы скрыть ее, воспользуемся событием ContentRendered



В коде пропишем:

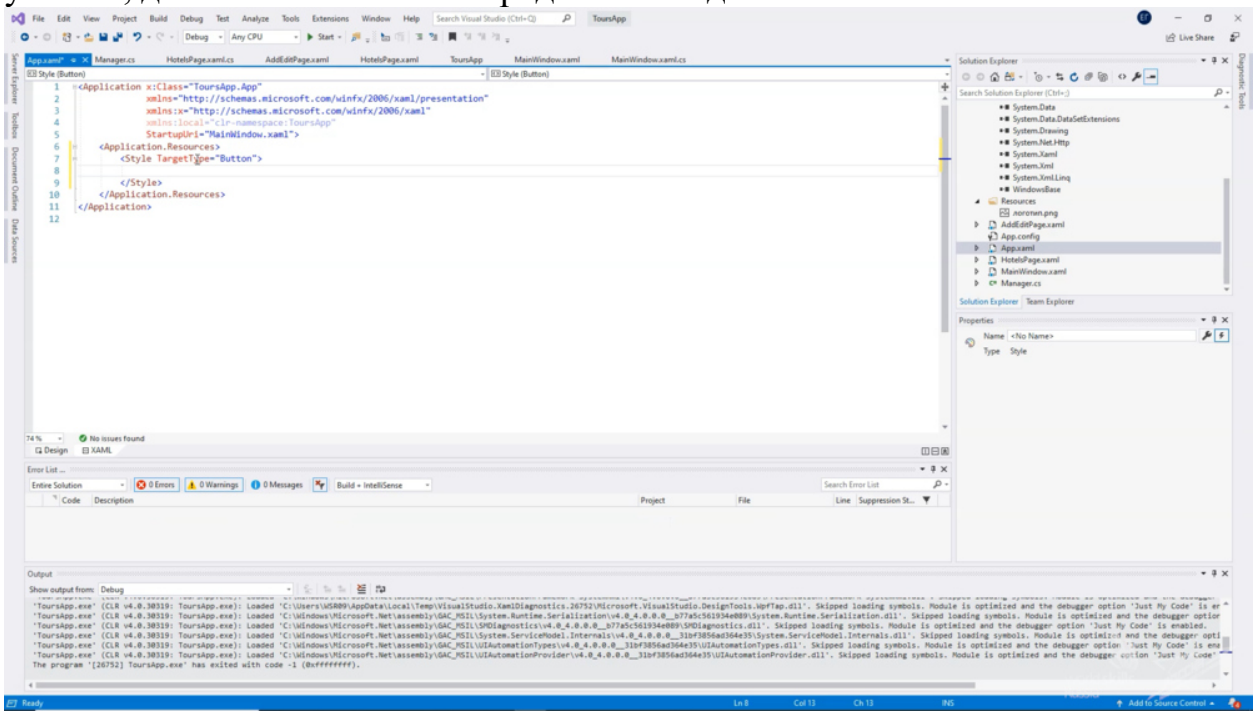


А чтобы скрыть стандартное навигационное меню, воспользуемся свойством фрейма — *NavigationUIVisibility='Hidden'*

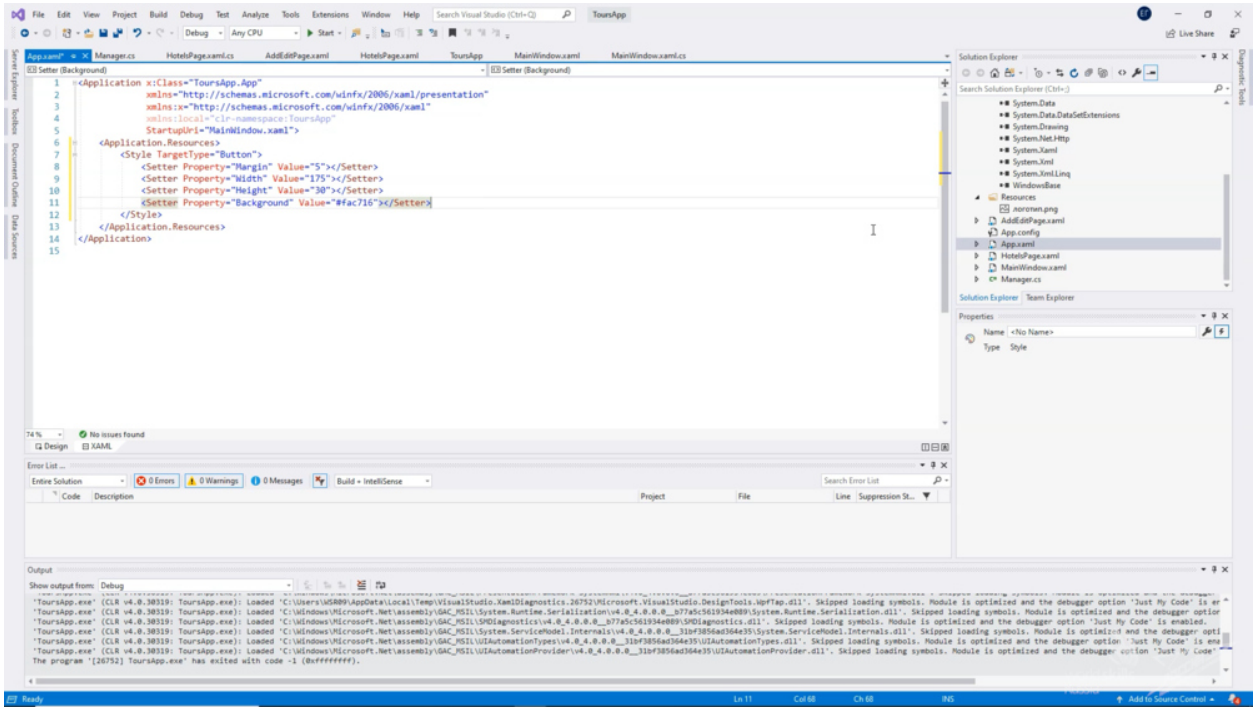


Глобальные стили приложения

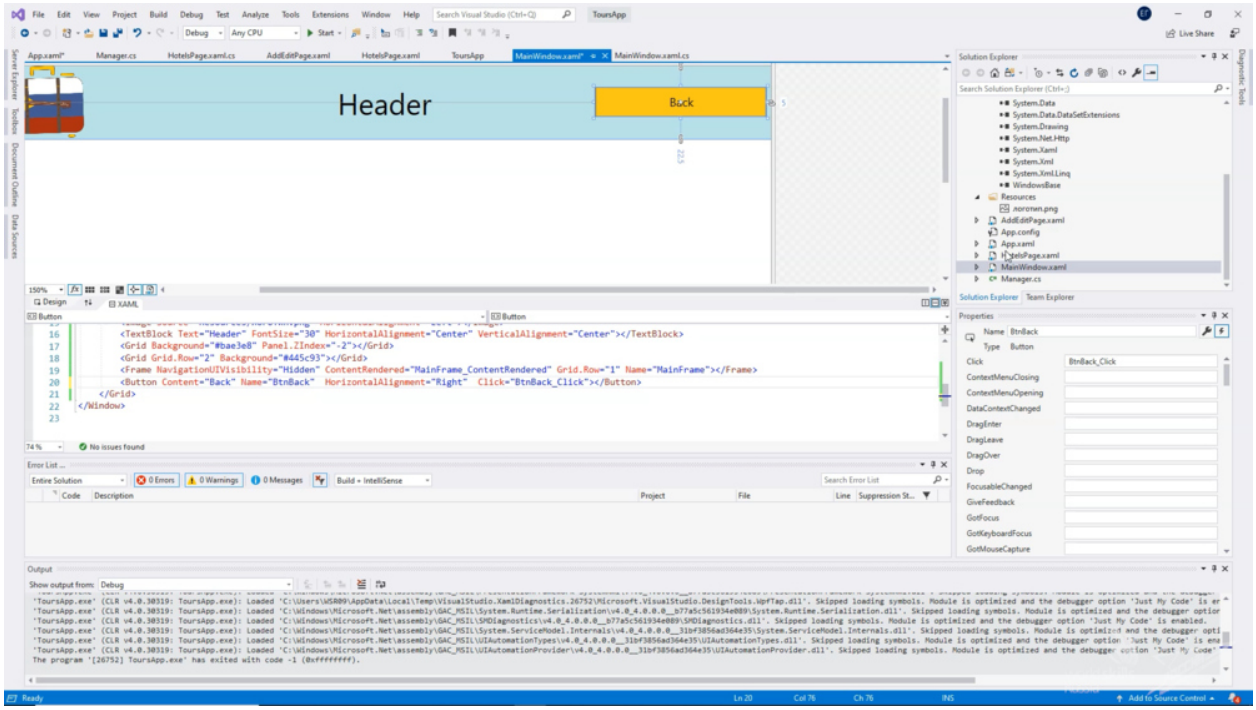
Для большинства созданных элементов мы использовали похожий набор свойств: ширина, высота, размер шрифта, отступы и др. Чтобы применять определенные наборы свойств для элементов, WPF предлагает использование глобальных стилей в проекте. Чтобы их создавать в проекте, есть файл App.xaml. Используем тег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



Интерактивное задание https://nationalteam.worldskills.ru/skills/sozdanie-karkasa-prilozheniya-sozdanie-i-ispolzovanie-stiley/

Практическая работа № 11. Создание списков (List View). Поиск и фильтрация данных

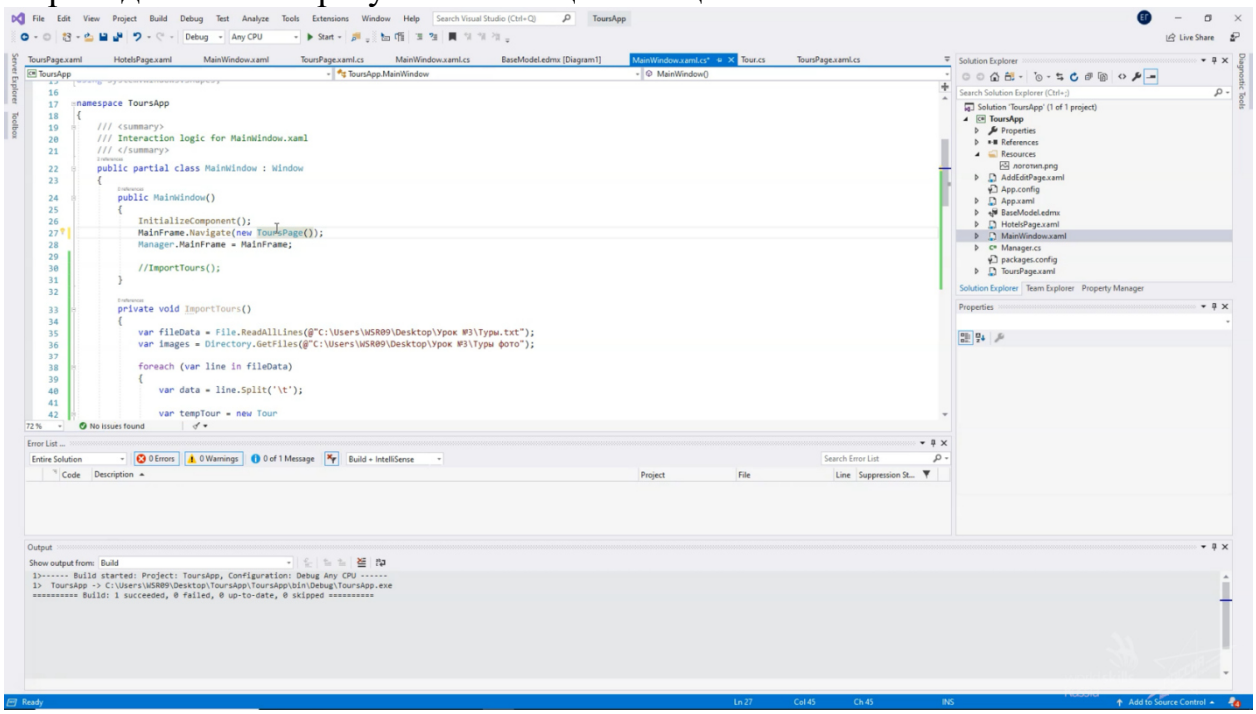
Краткие теоретические сведения

Продолжим разработку настольного приложения и поговорим

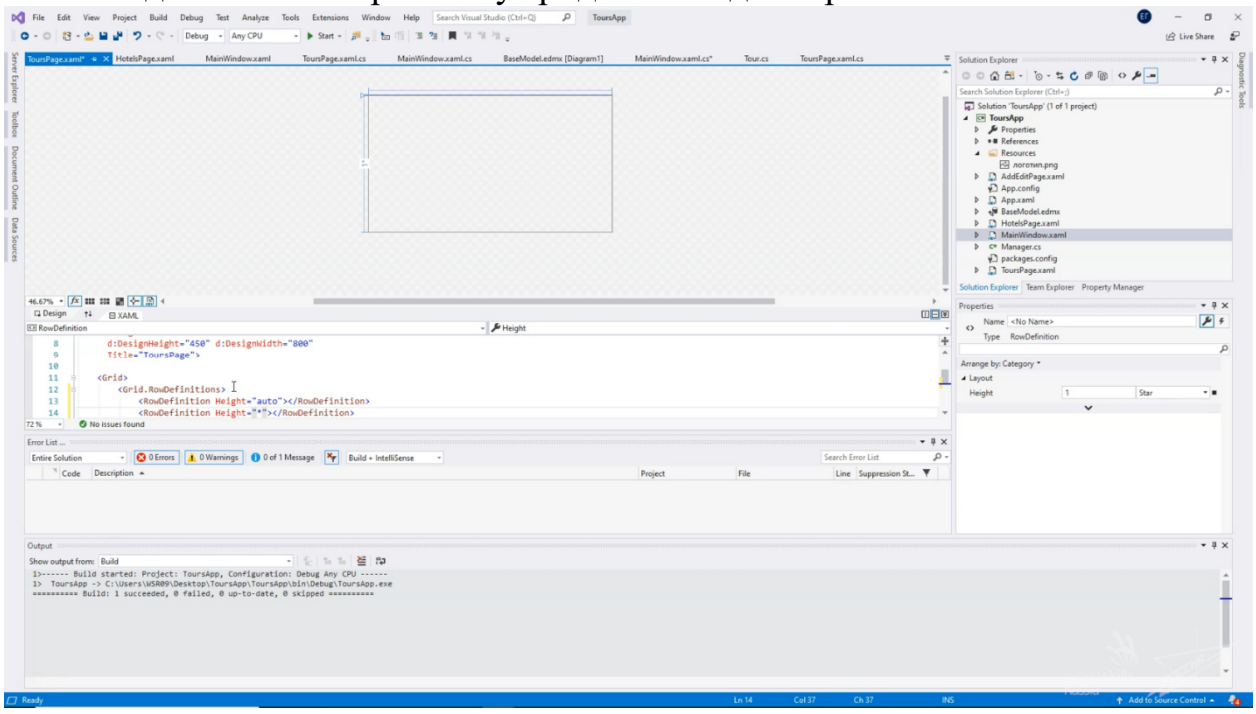
об альтернативном DataGrid'у элементе, который может отображать информацию из базы данных — ListView. Как правило, он представляет собой стандартный список. Однако при желании вы можете сделать сложную компоновку объекта, которую не получилось бы реализовать с помощью DataGrid. Также можно вывести элементы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

Вывод информации о турах с ListView

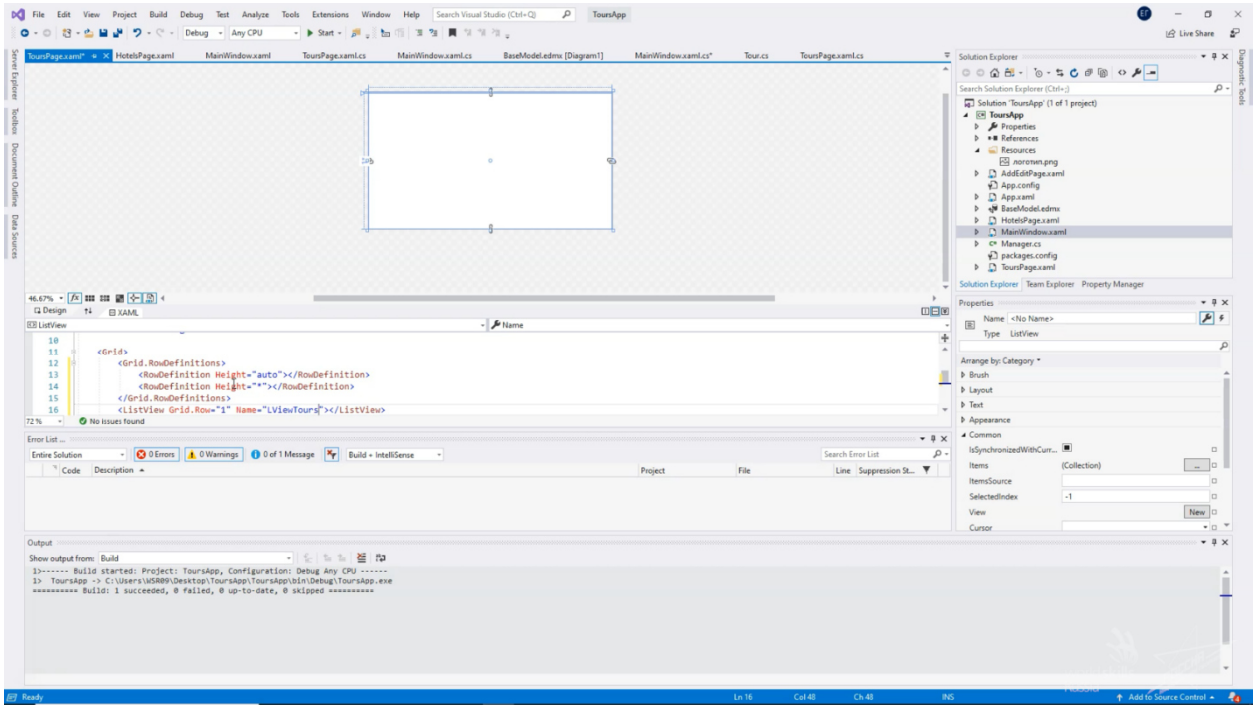
Для вывода информации о турах добавляем новую страницу с ListView Переходим на нее сразу после инициализации компонентов MainWindow



Подготавливаем разметку грида — это две строки

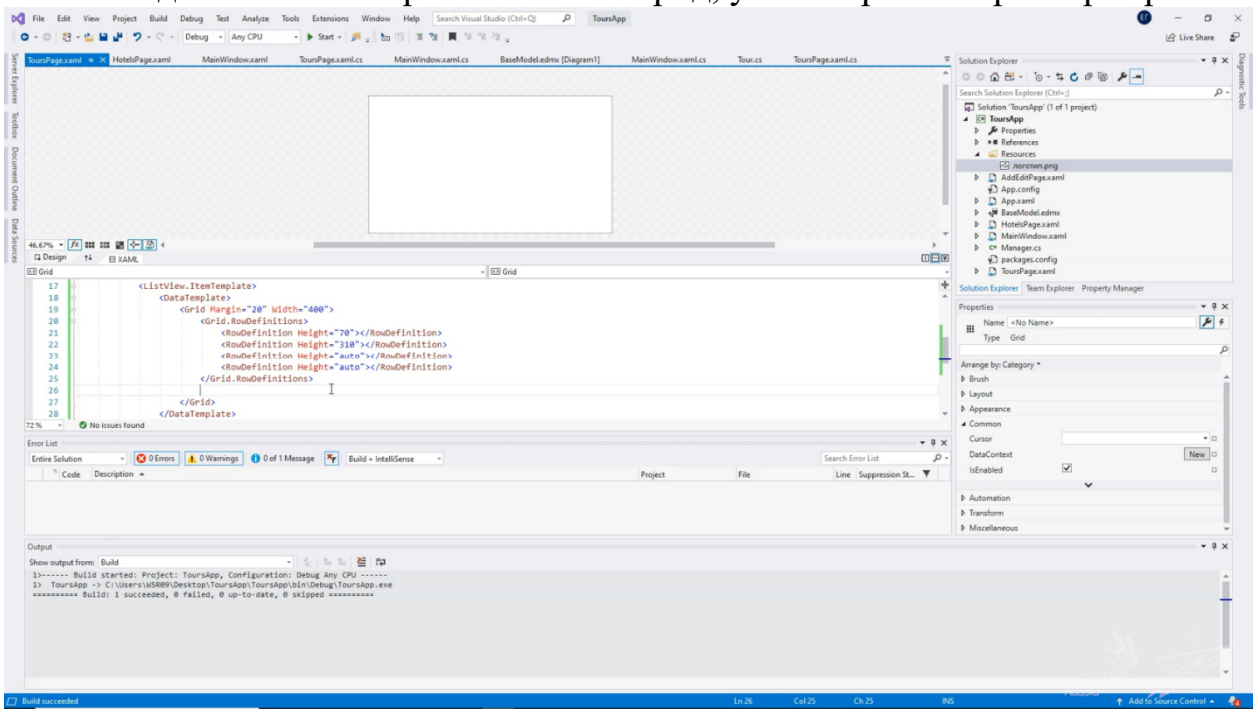


Добавляем элемент ListView, размечаем в гриде и задаем имя

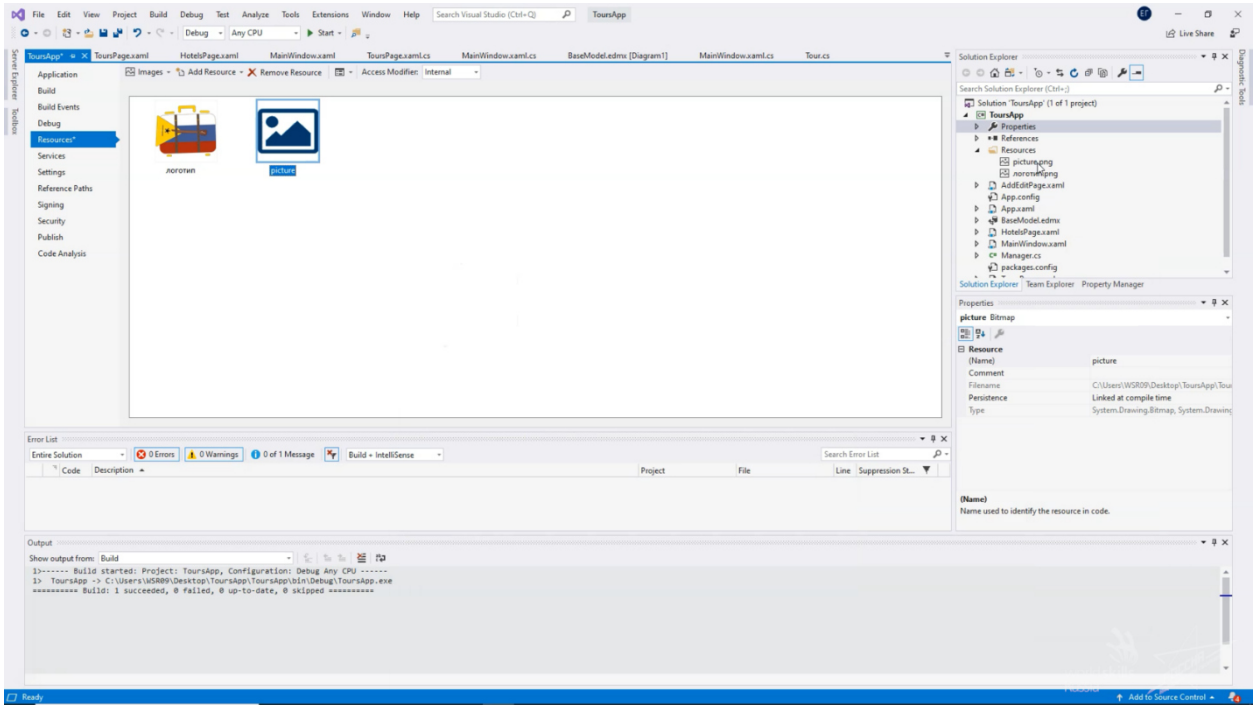


Для вывода информации о туре создается шаблон элемента в списке — то представление, которое должно отображаться для каждого элемента

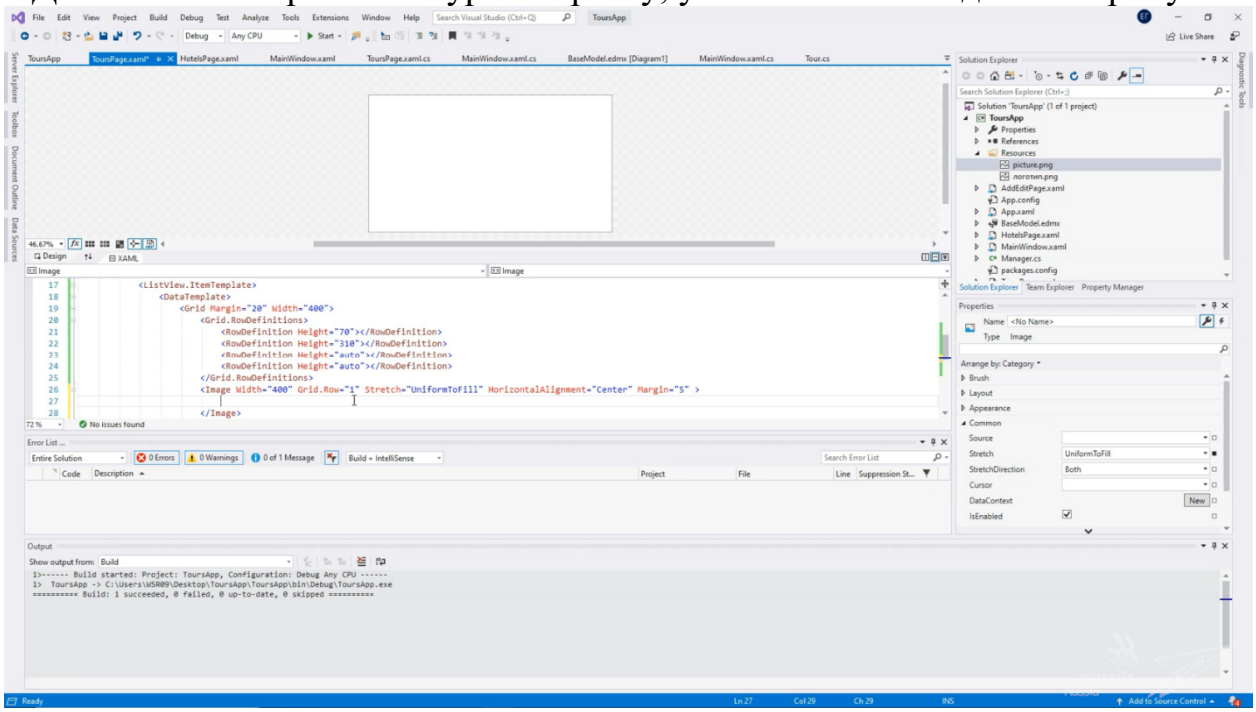
Создаем шаблон и размечаем его грид, указав при этом размер строк



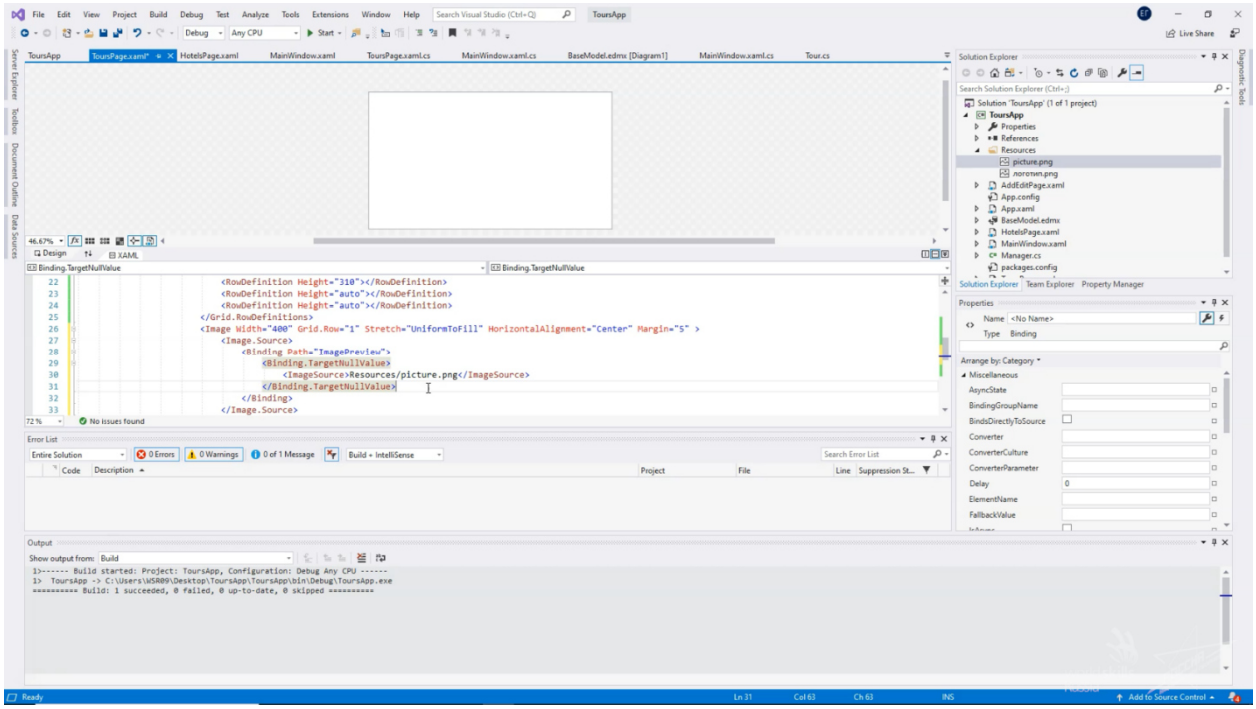
Добавляем в ресурсы изображение, которое будем выводить при отсутствии картинки у тура



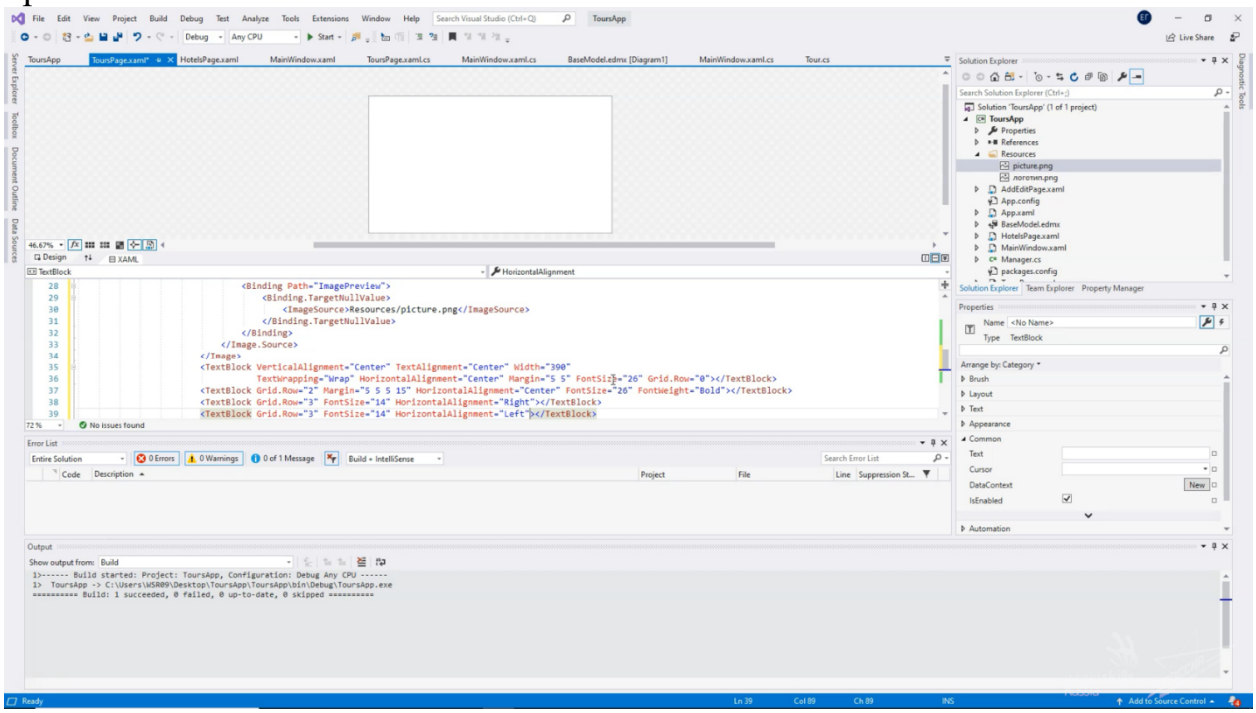
Добавляем изображение тура в верстку, установив необходимые атрибуты



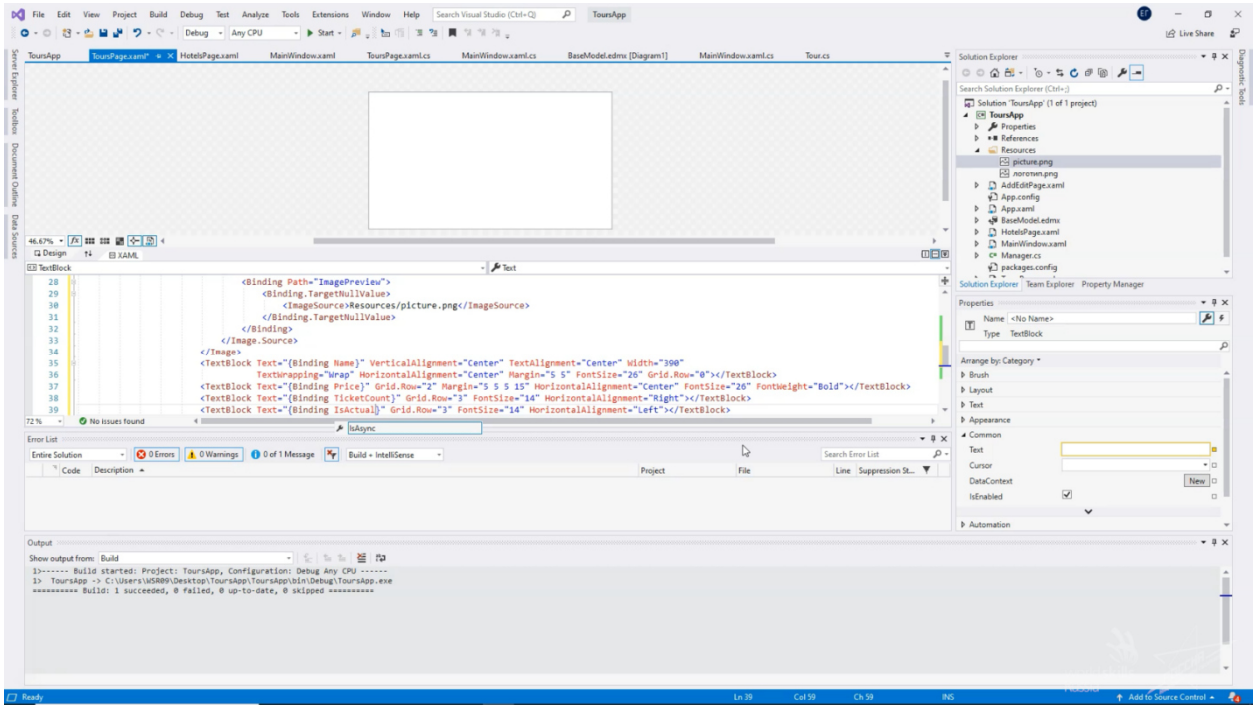
Теперь задаем объекты для привязки, а также изображение, если изображения тура не будет



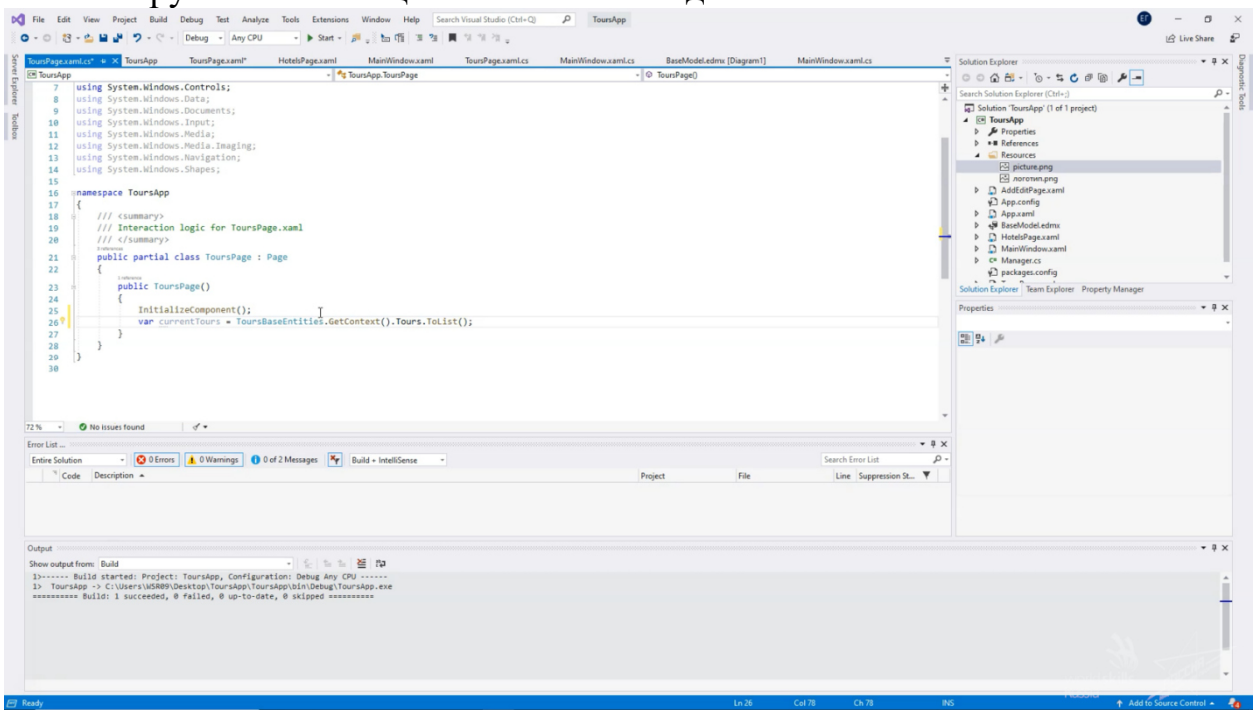
Добавляем текстовые поля для наименования, указав: перенос текста, выравнивание, большой шрифт; для стоимости, указав: выравнивание, также большой шрифт, но делаем текст жирным; количество билетов — выравниваем по правому краю; и актуальности — выравниваем по левому краю



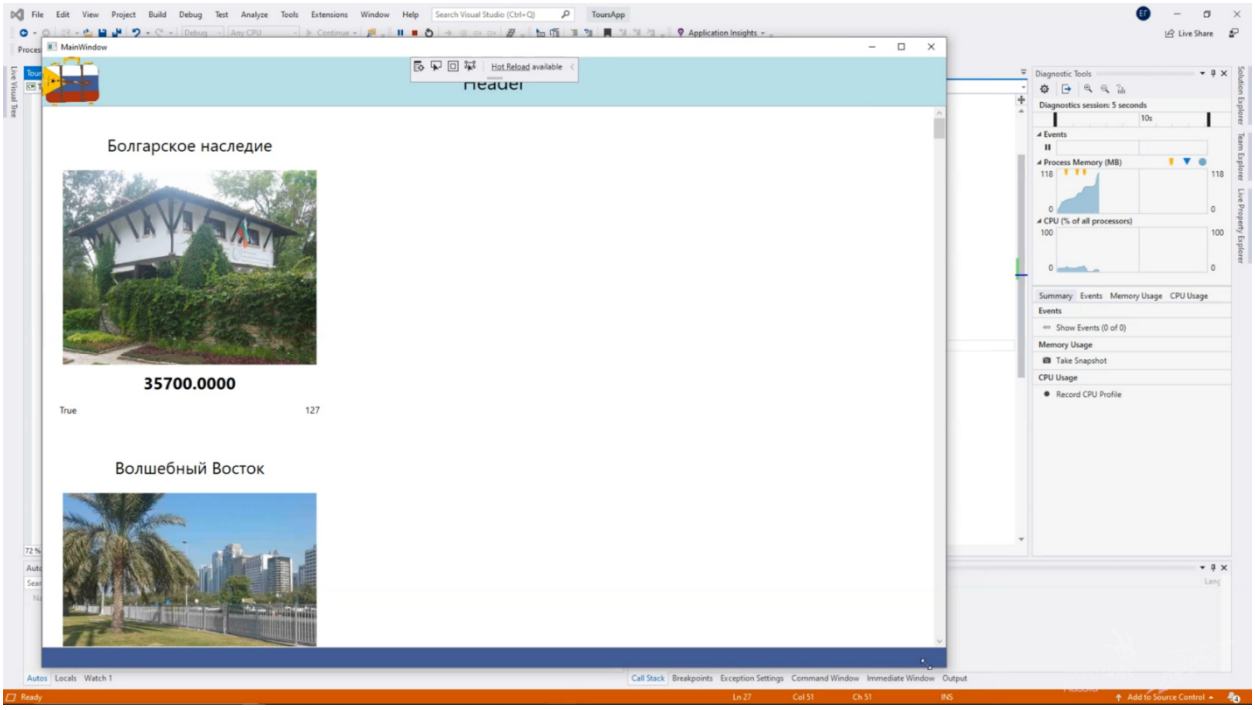
Добавляем привязки для наименования, стоимости, количества билетов и актуальности



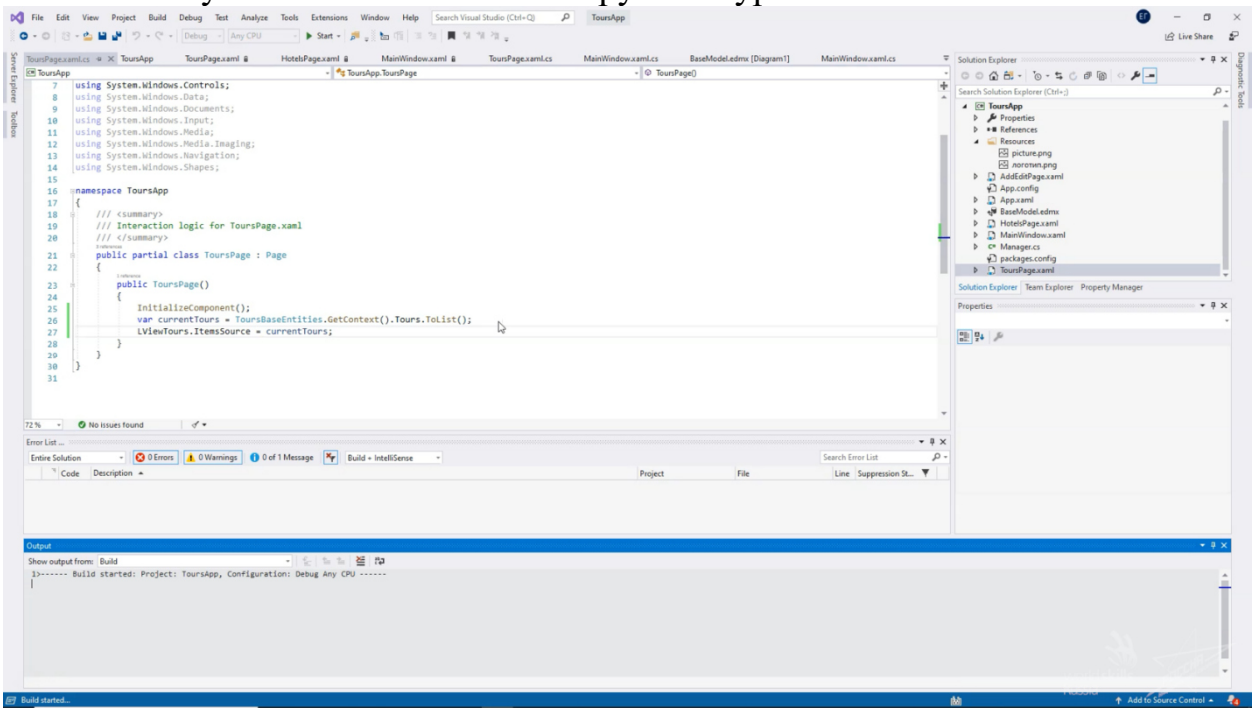
Загружаем коллекцию «список» в коде



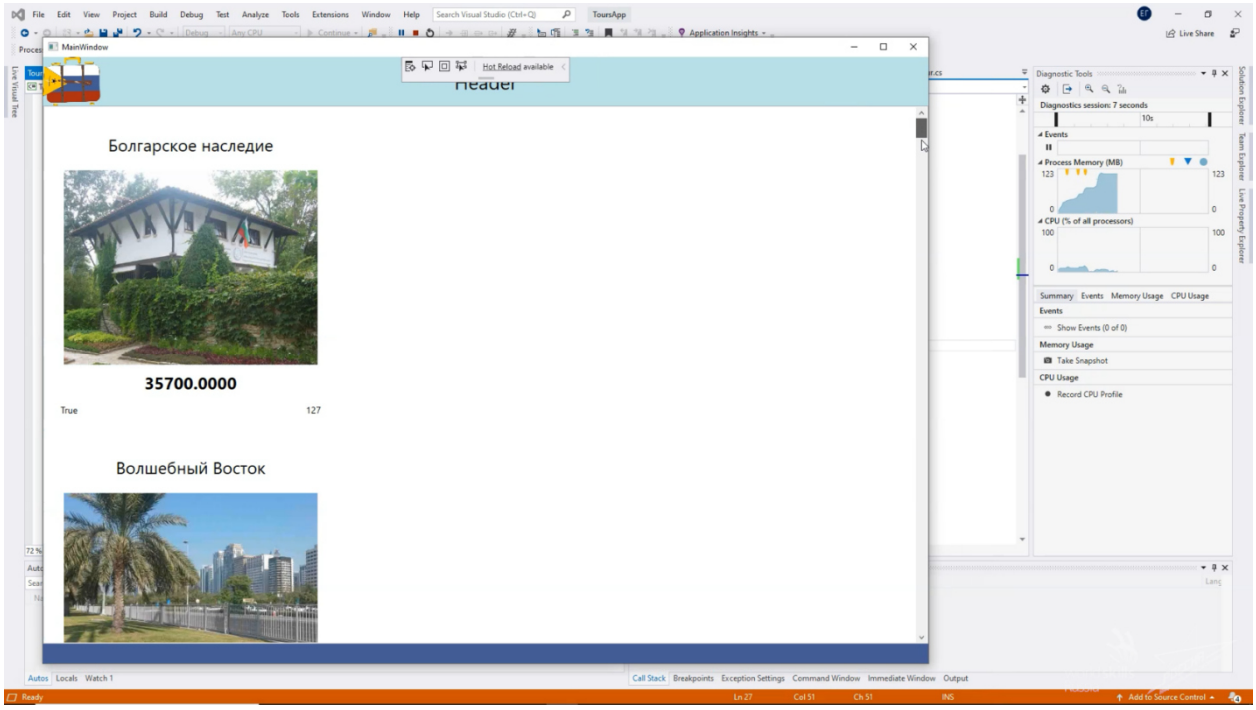
Далее — Далее — Финиш. 252 строки было импортировано. Можем их увидеть в таблице



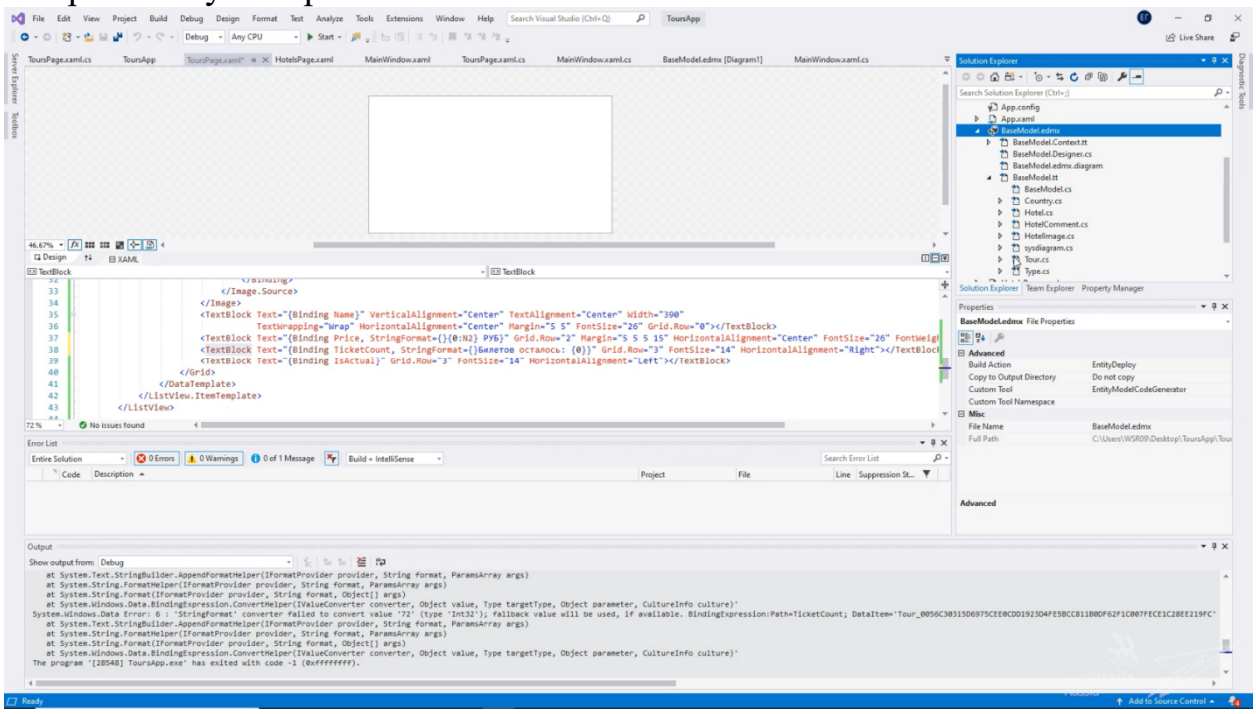
В чем у нас ошибка? Мы не загрузили туры в список — наш ListView



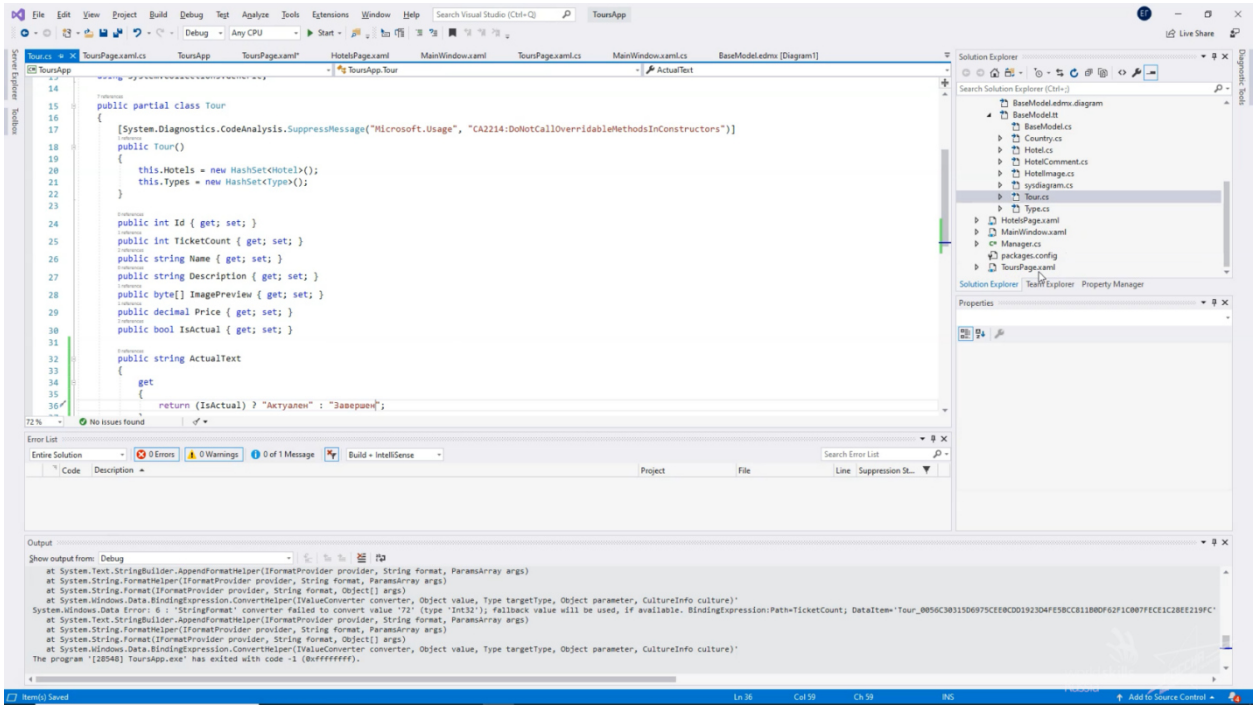
Запускаем приложение еще раз



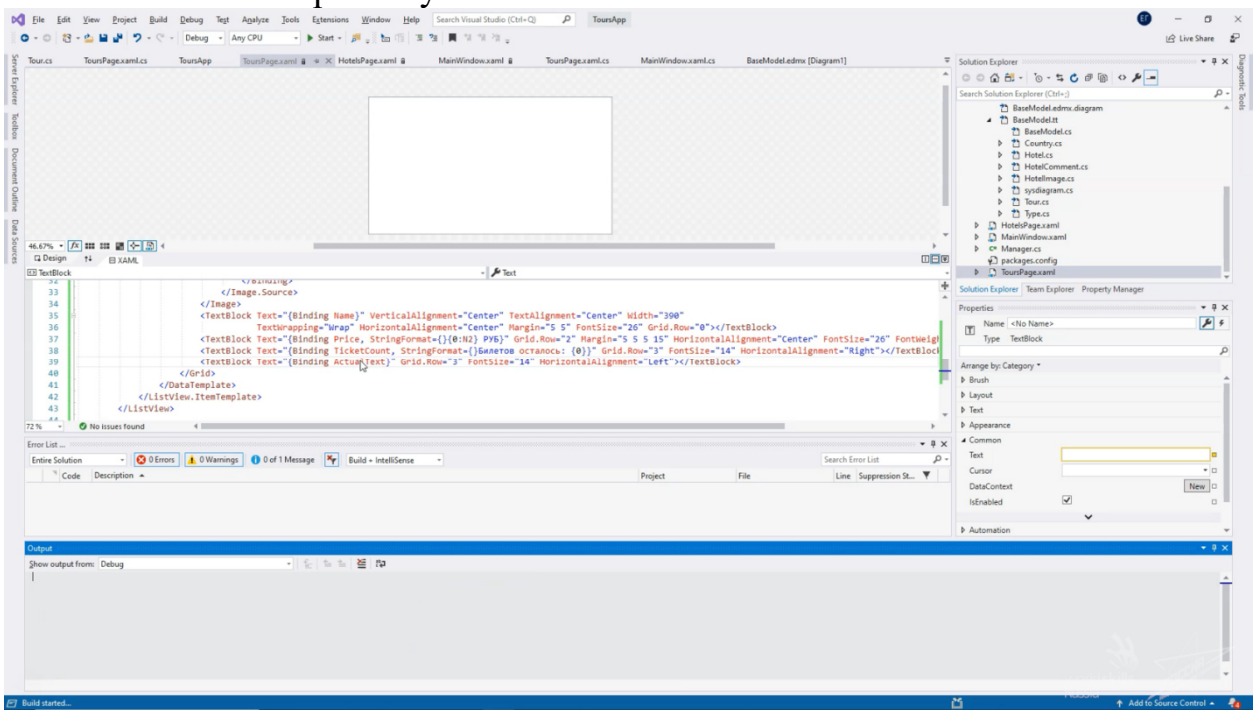
Работаем над форматом отображения стоимости и количества билетов. Указываем им StringFormat формат для указания цены (два знака после запятой) и примечание к количеству билетов. Делать это можно прямо во время запуска приложения



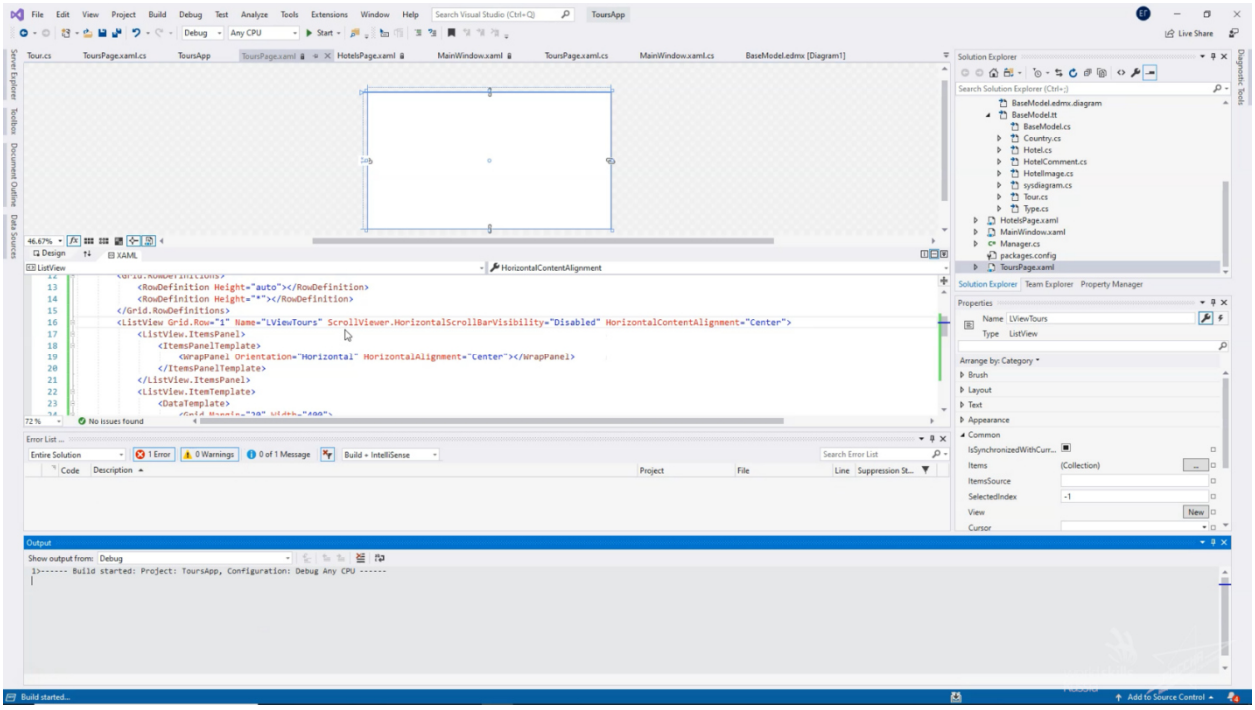
Для отображения актуальности делаем дополнительное свойство в классе Тур — назовем его ActualText



Выполняем привязку



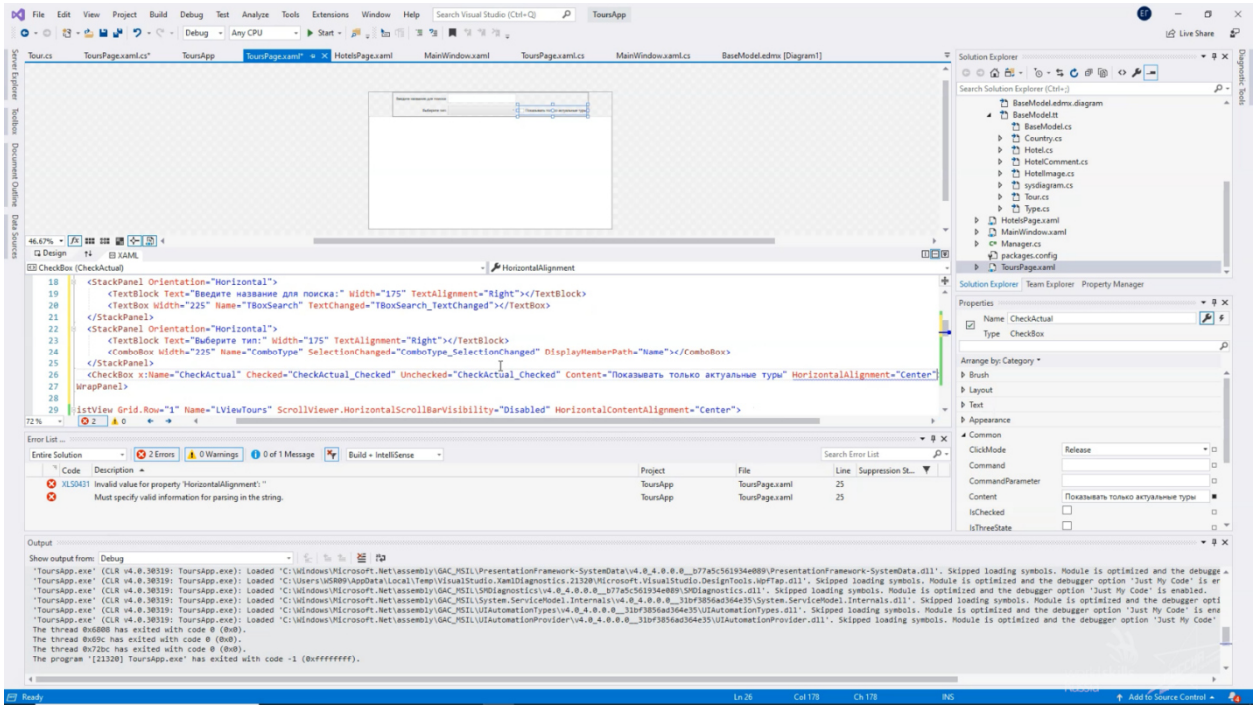
Теперь изменяем представление ListView на плитку. У нас в качестве ItemsPanel будет находиться RowPanel, который позволяет нам переносить элементы в виде плиток. И не забываем убрать горизонтальную прокрутку в ListView



Реализация поиска и фильтрации информации

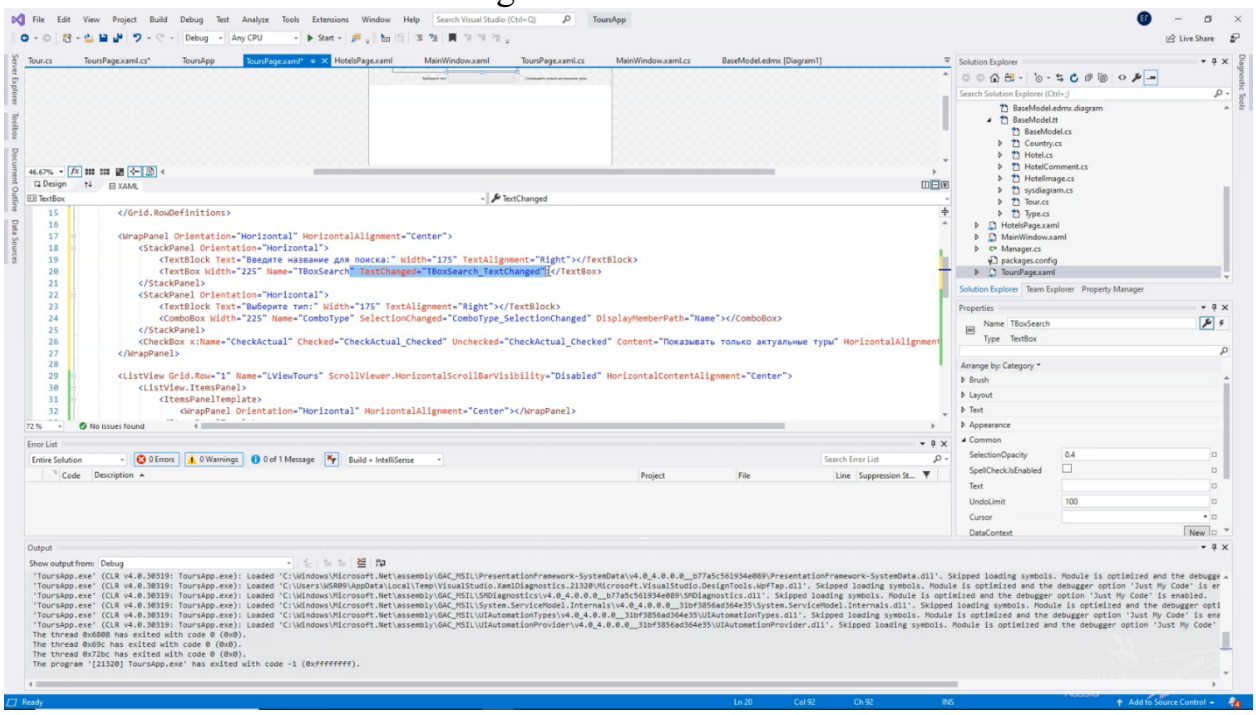
Как говорилось ранее, для работы с большими объемами информации полезно реализовать поиск и фильтрацию. Как это работает? Пользователь вводит в специальные элементы управления данные для поиска или выбирает категории из списка для фильтрации. Затем в коде разработчик приводит коллекцию данных к виду,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

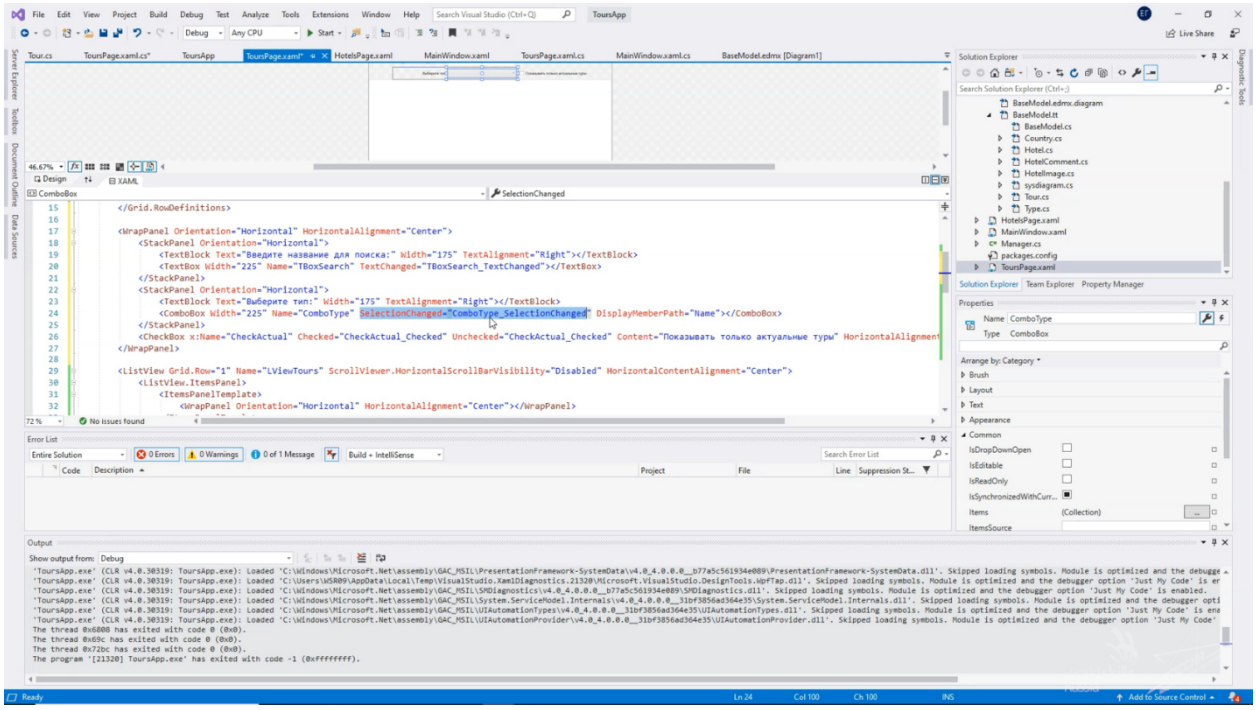


В плане удобства использования наиболее выигрышно выглядит механизм, когда результаты поиска выводятся сразу по мере ввода ключевого слова или выбора значения в выпадающем списке. Поэтому мы обработаем соответствующие события на каждый элемент

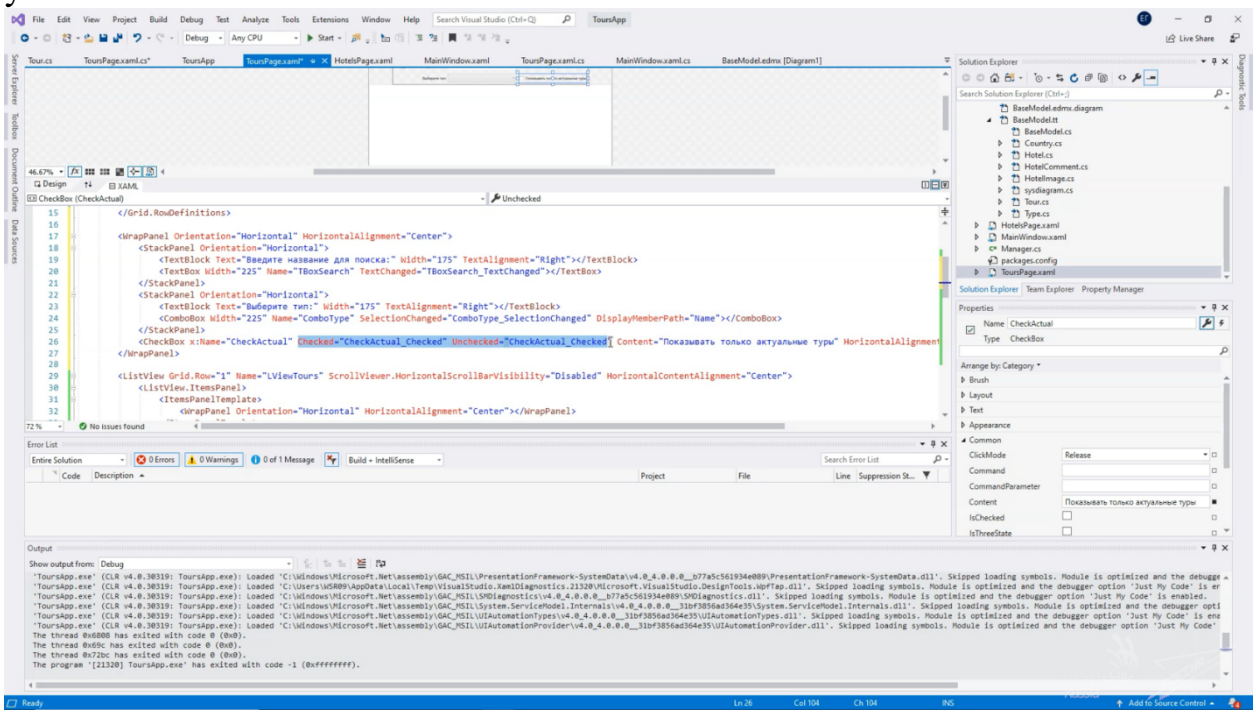
Во-первых, это TextChanged на изменение текста для поиска. TextBox — Search — TextChanged



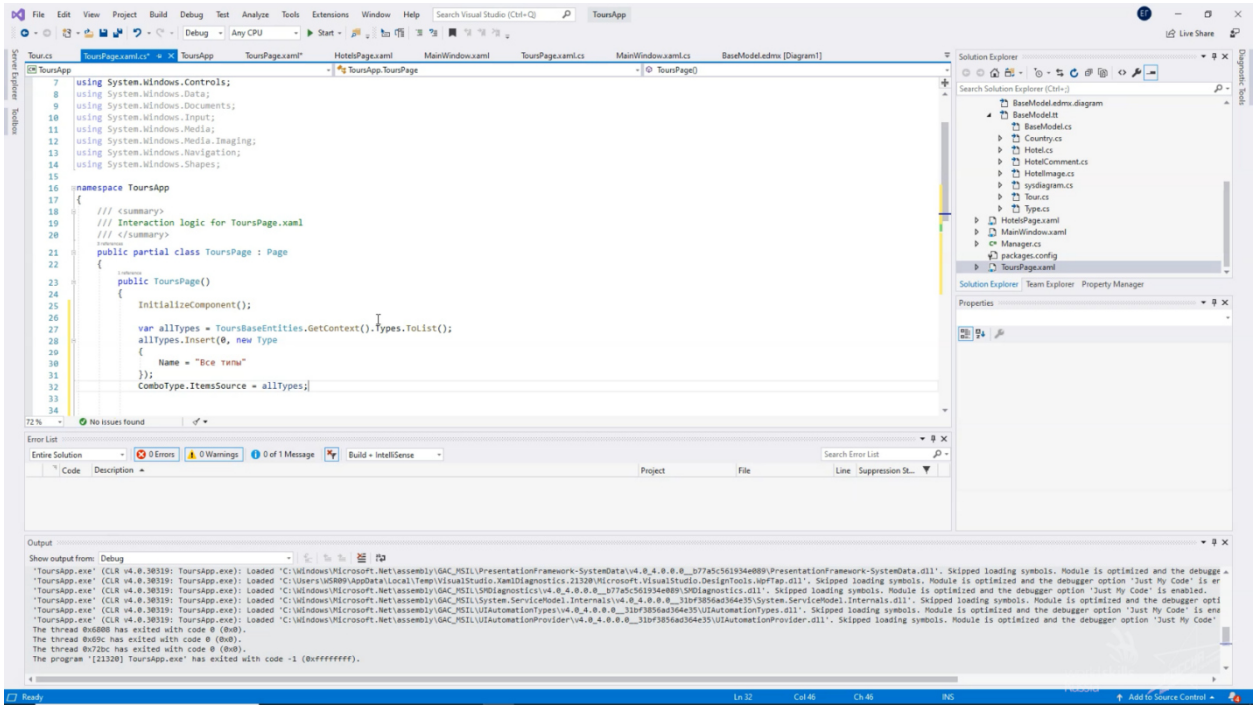
Затем при изменении выбора в выпадающем списке — Selection Changed



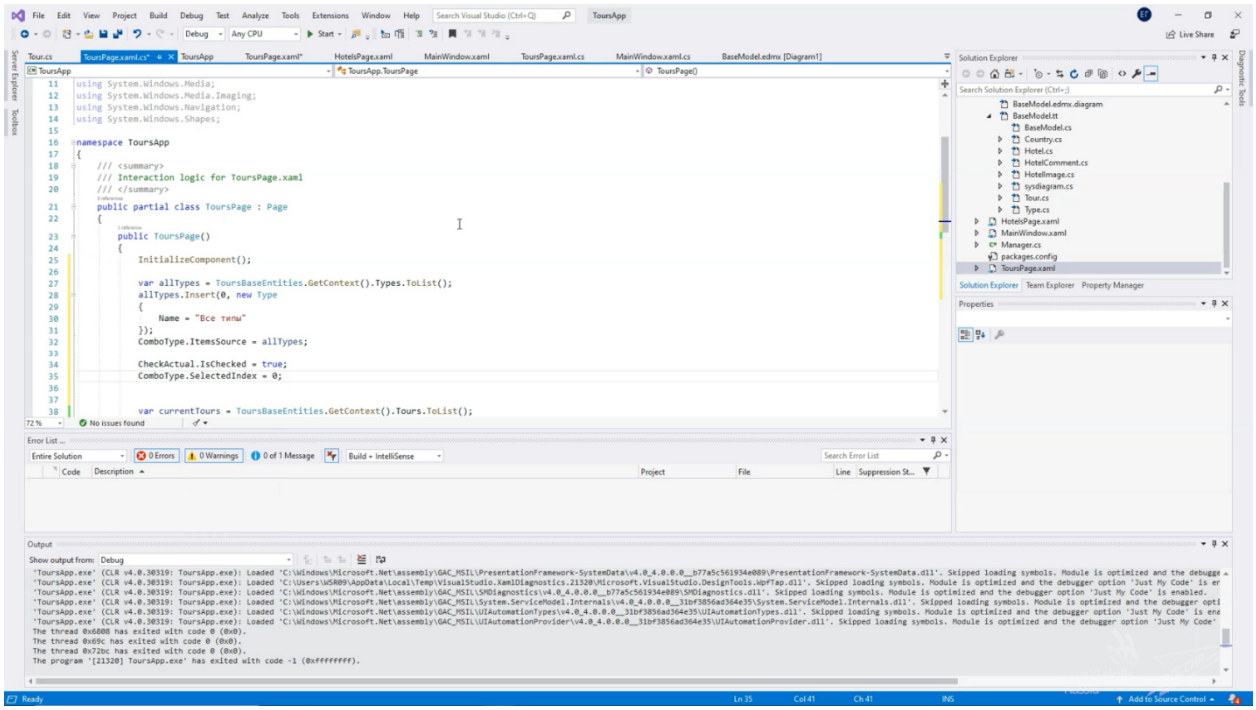
После этого мы добавляем обработку нажатия и снятия флажка у CheckBox



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

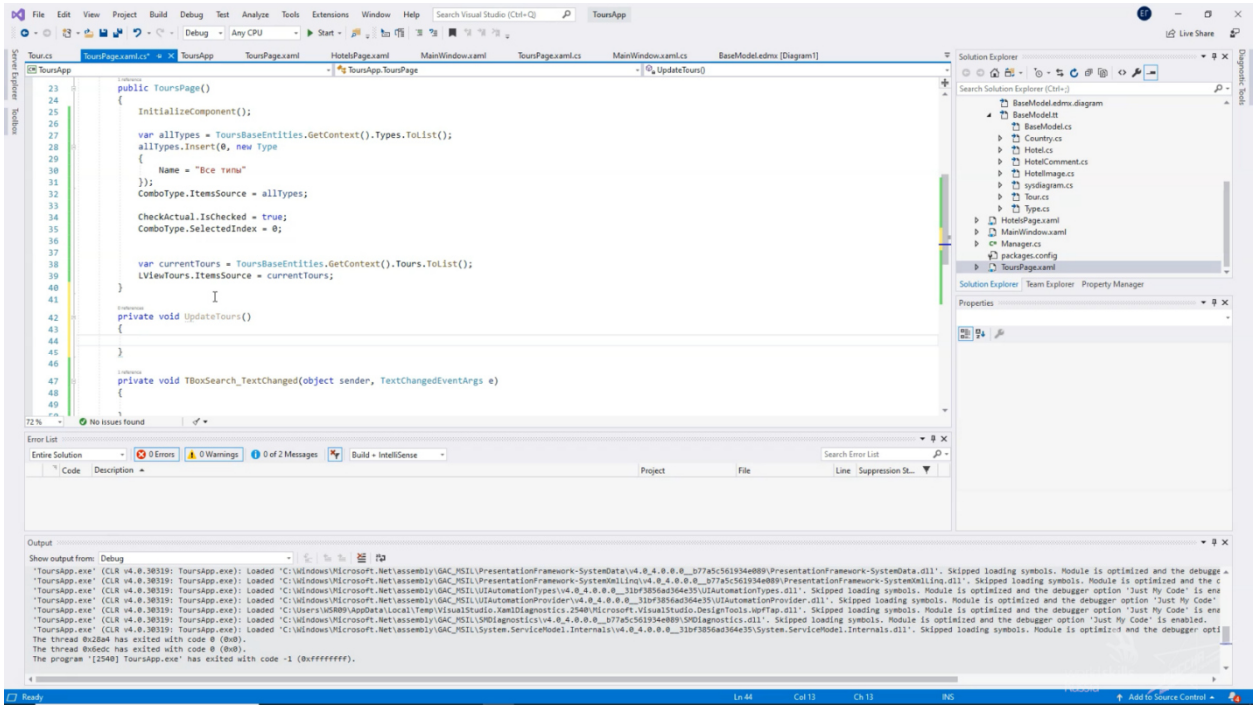


Устанавливаем стартовые значения для элементов управления: CheckBox и ComboBox

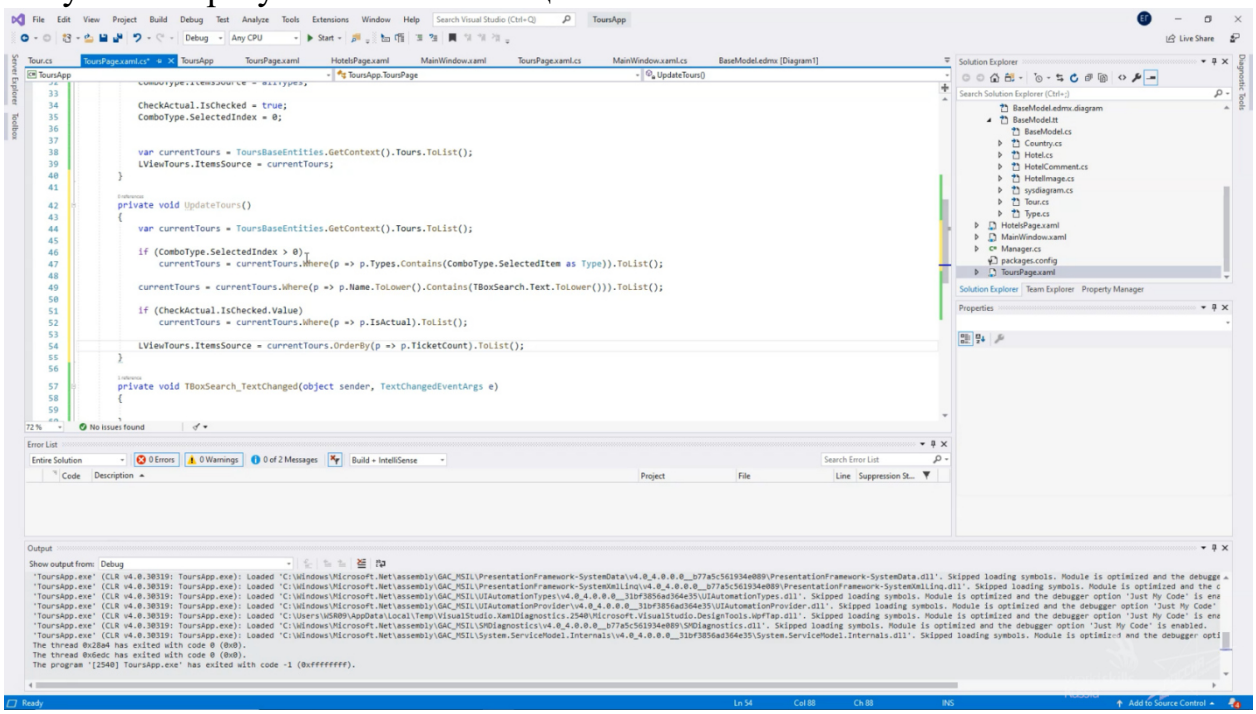


Обрабатываем методы поиска и фильтрации так, чтобы они работали вместе. Для этого в одном методе выполним фильтрацию коллекции, поиск по ней и сортировку, а затем вызываем этот метод из обработчиков событий всех наших элементов управления

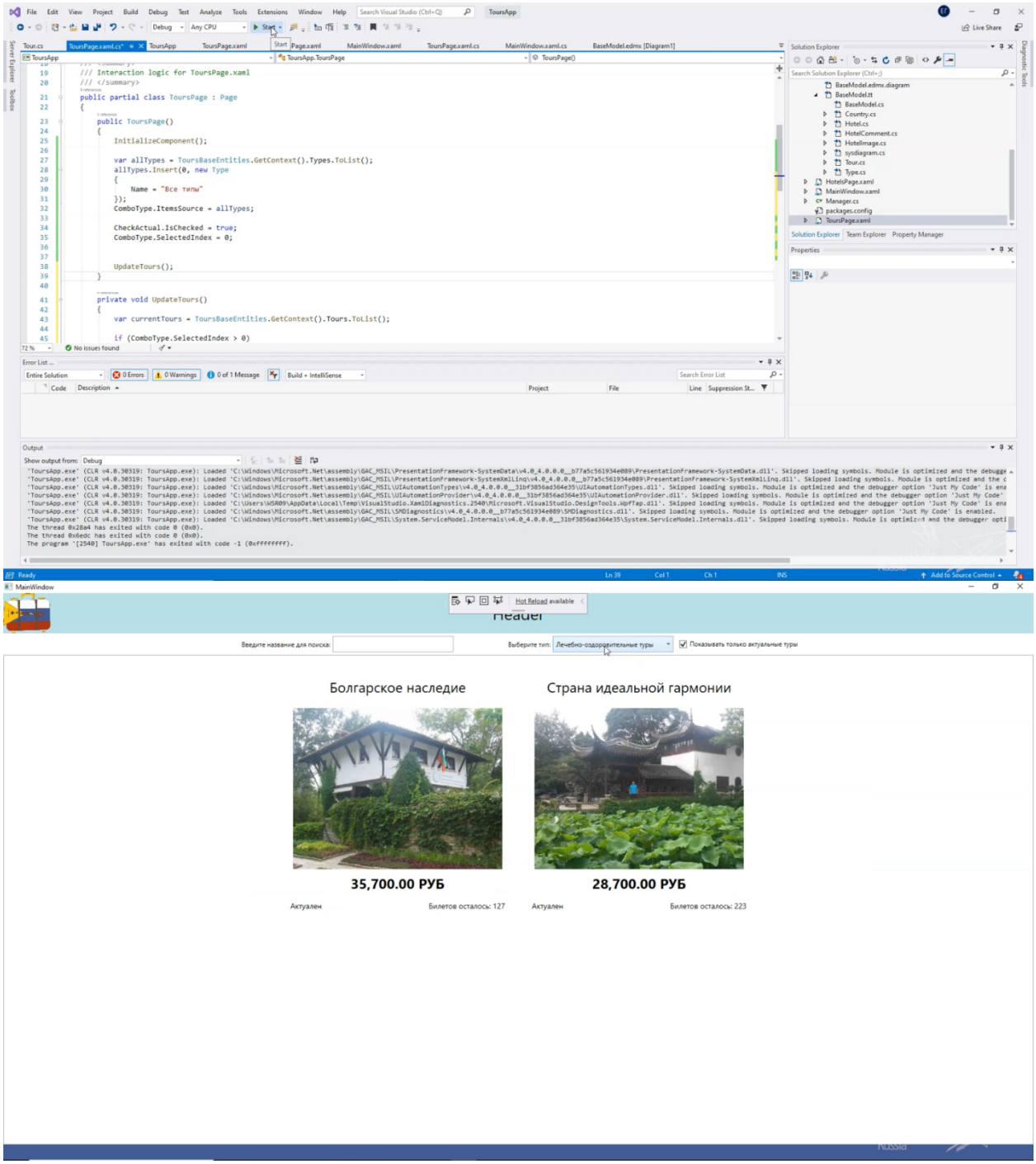
Для начала создаем метод UpdateTours()



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



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



Интерактивное задание https://nationalteam.worldskills.ru/skills/sozdanie-spiskov-listview-poisk-i-filtratsiya-dannykh/

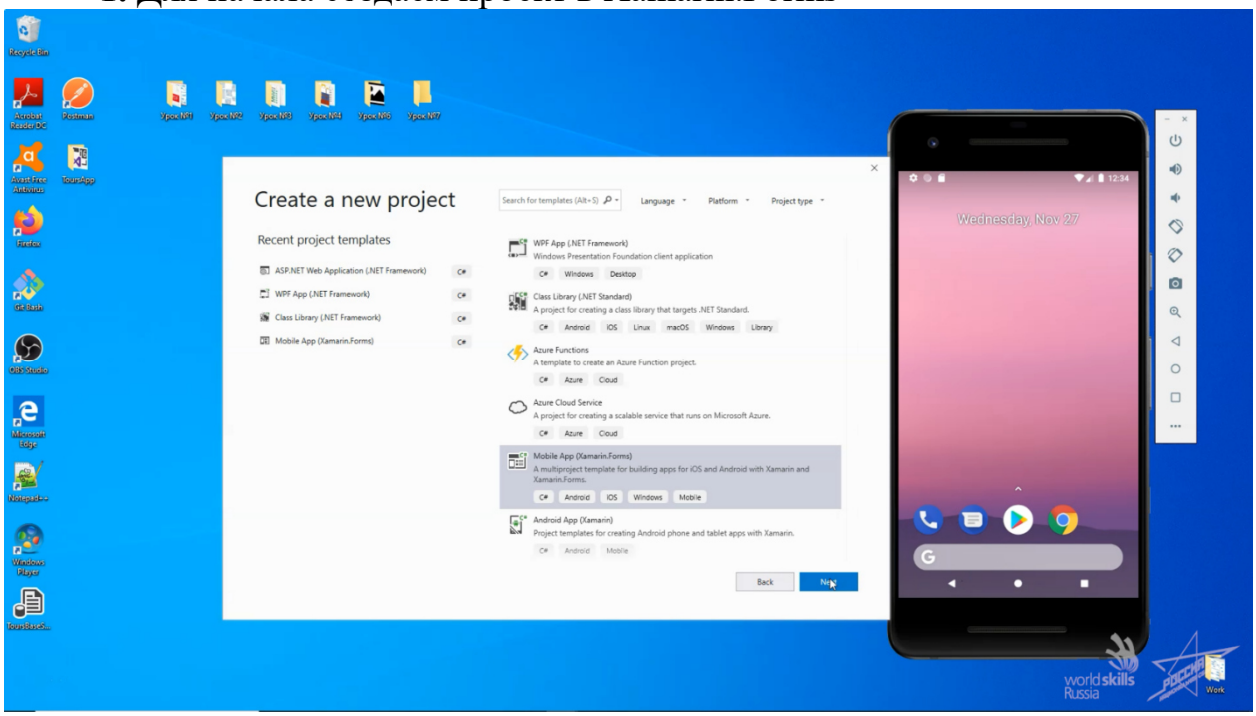
Практическая работа № 12. Кроссплатформенная мобильная разработка Xamarin Forms

Краткие теоретические сведения

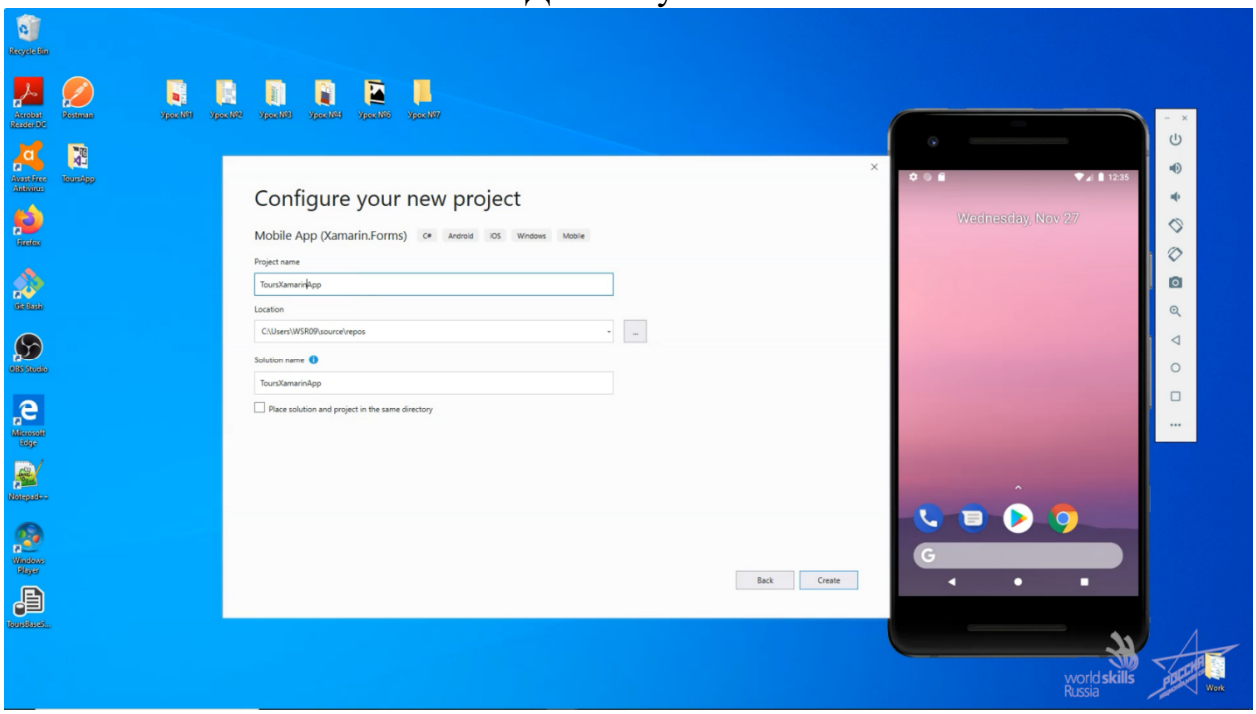
Продолжим тему мобильной разработки и поговорим о кроссплатформенной разработке на Xamarin.Forms. Эта платформа позволяет создавать одну единственную логику приложения с применением

C# сразу для всех трех платформ: Android, iOS и UWP. Из преимуществ можно назвать: единый код для всех платформ, прямой доступ к нативным API каждой платформы, возможность использования платформы .net и языка программирования C#, который является достаточно производительным, и в то же время простым для освоения и понимания.

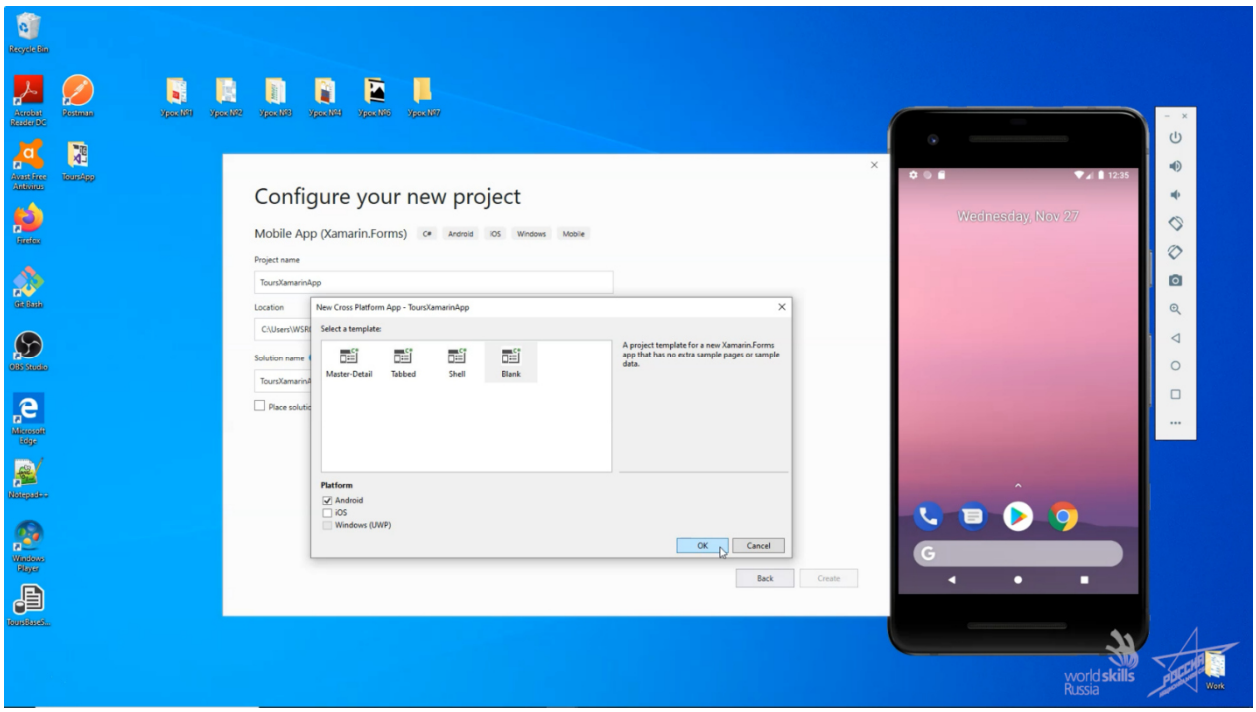
1. Для начала создаем проект в Xamarin.Forms



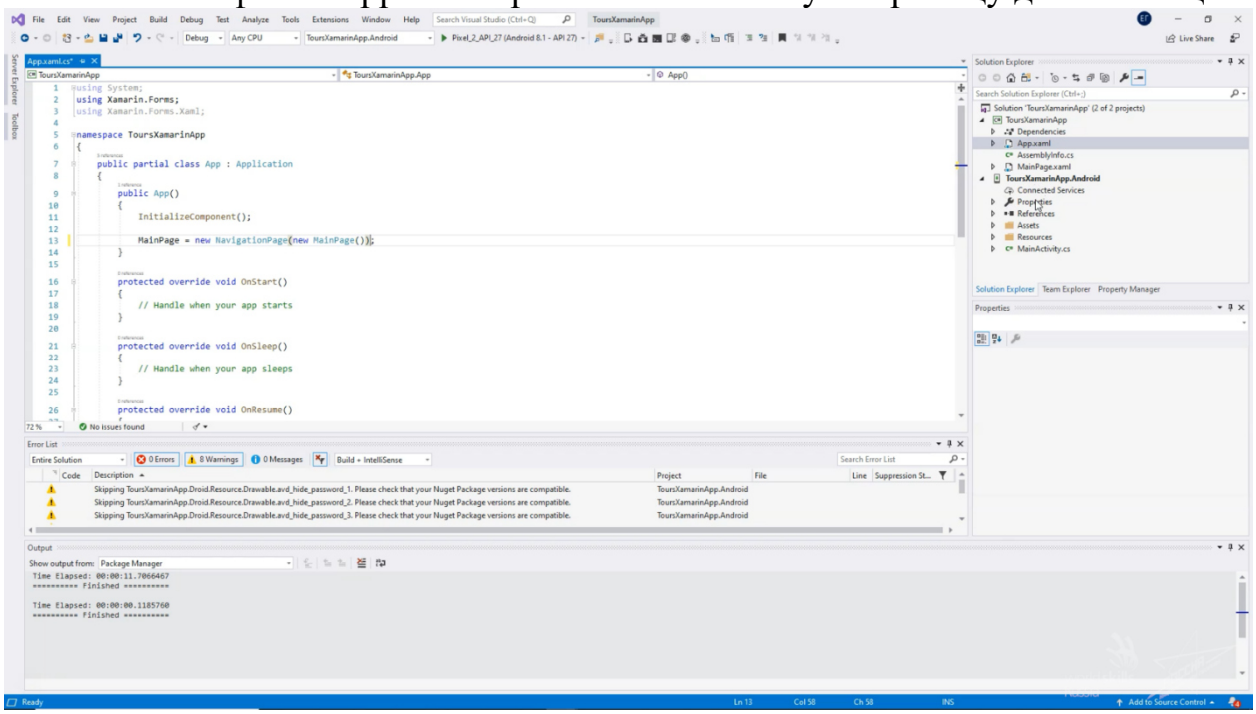
- Даем ему название



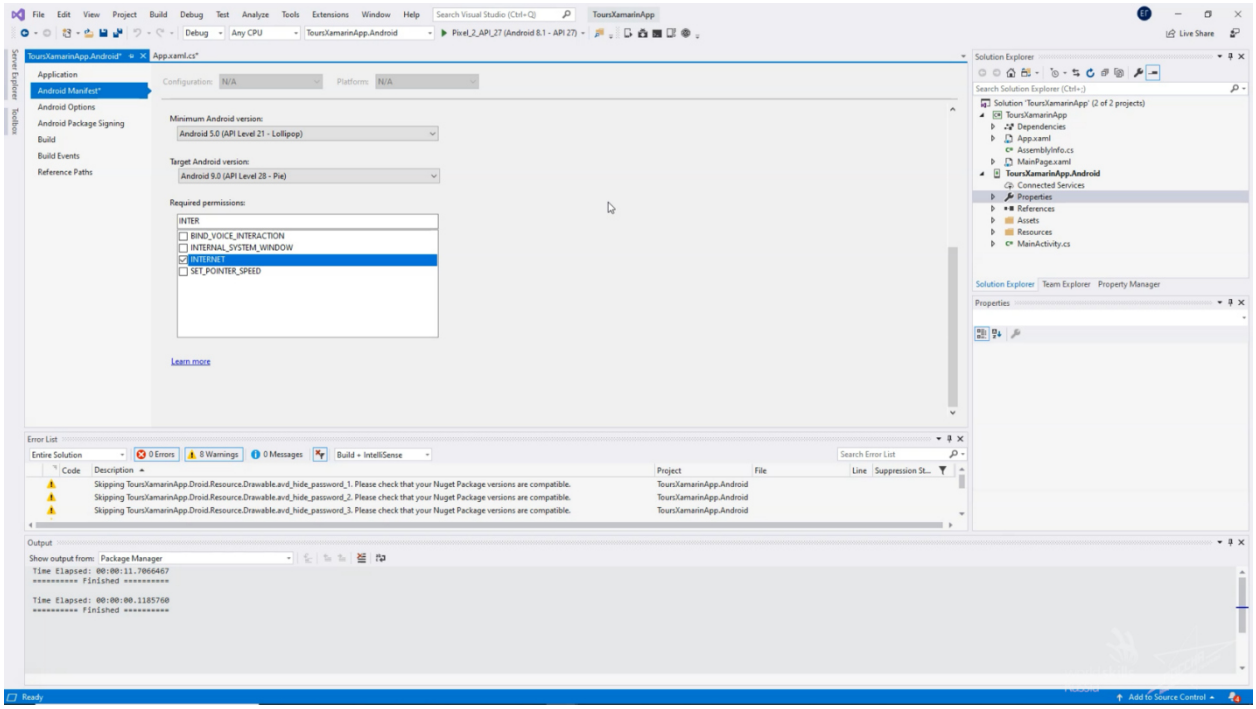
- В Xamarin представлен ряд шаблонов. Можно использовать списки, вкладки… Пока что мы будем просто выбирать Blank — пустую страницу, и убираем платформу iOS — нам сейчас нужен только Android



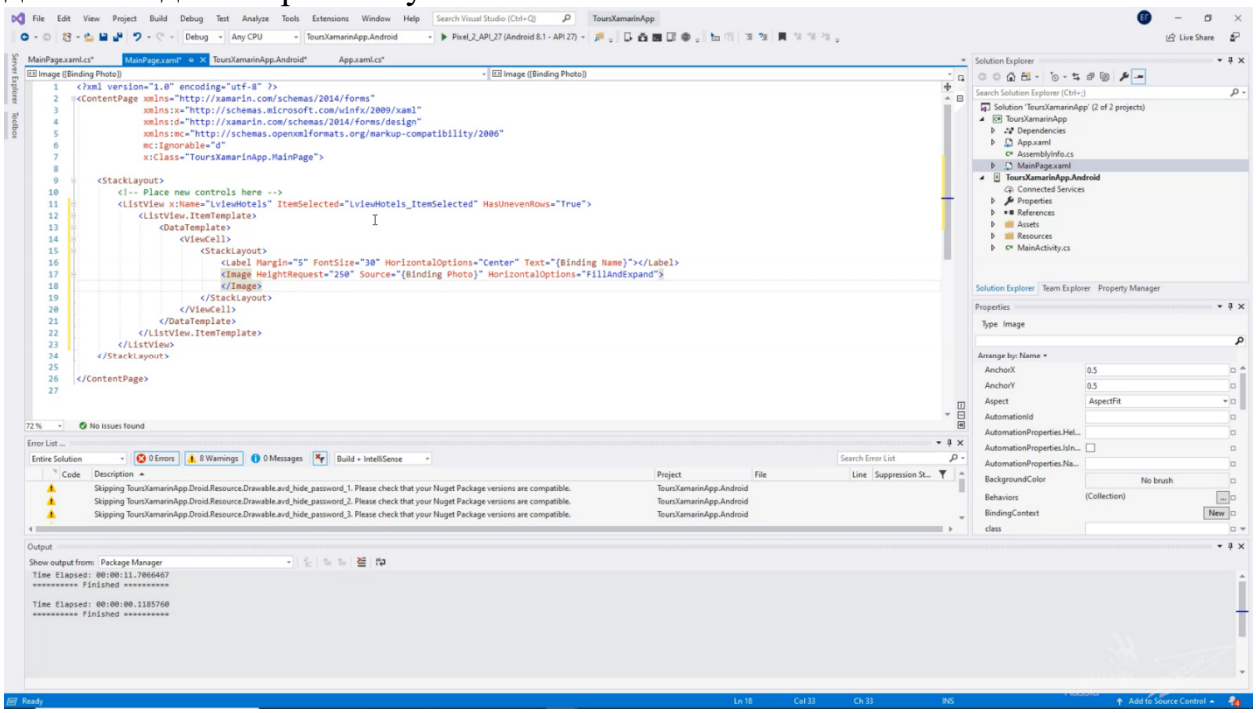
- В файле App.xaml прописываем главную страницу для навигации



- Ставим разрешение на использование интернет-подключения. Делается это в настройках Android Manifest



2. Начинаем с верстки MainPage. Она будет представлять собой простой
список — ListView. Здесь мы обрабатываем события нажатия, привязываем данные и делаем разметку



Важно

В свойство ListView установливаем параметр HasUnivenRows, который позволит иметь динамический размер объекта. Немного упрощаем верстку списка и не отображаем звездность отеля, как было с RatingBar в Android Studio, так как аналогичного стандартного компонента в Xamarin нет, а на его реализацию уйдет значительное количество времени

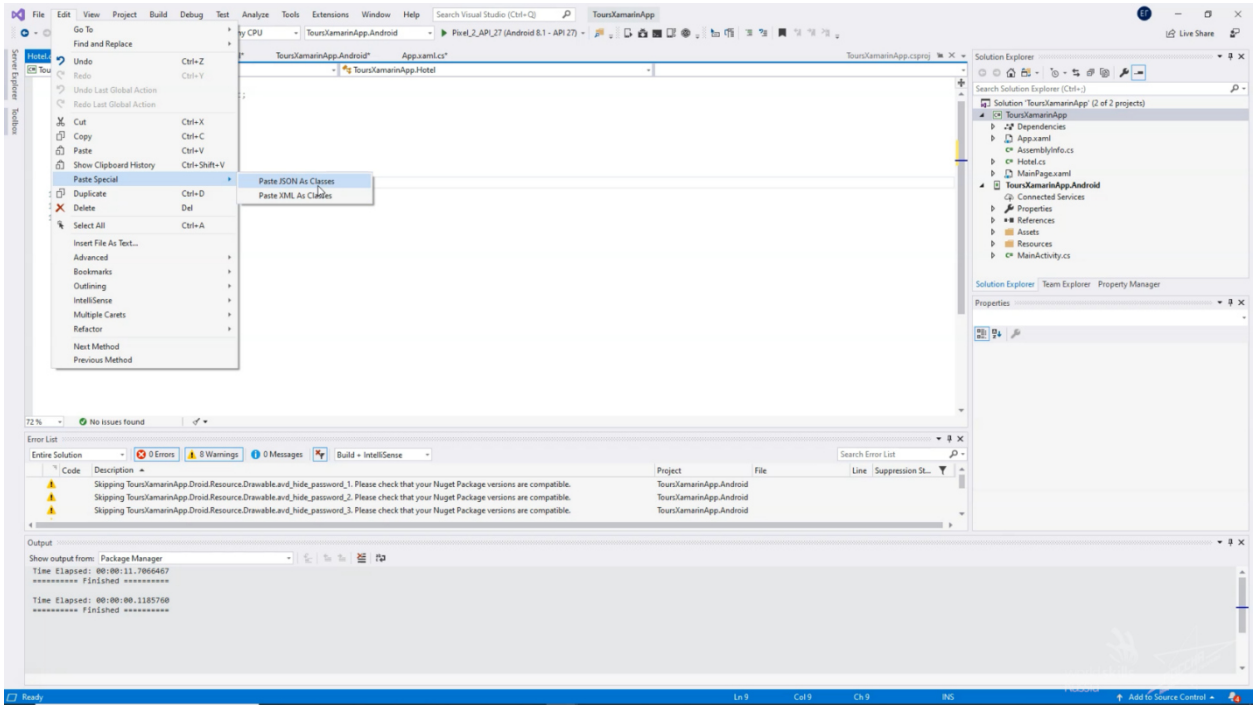
3. Давайте попробуем получить с помощью API список отелей

и работать с ним

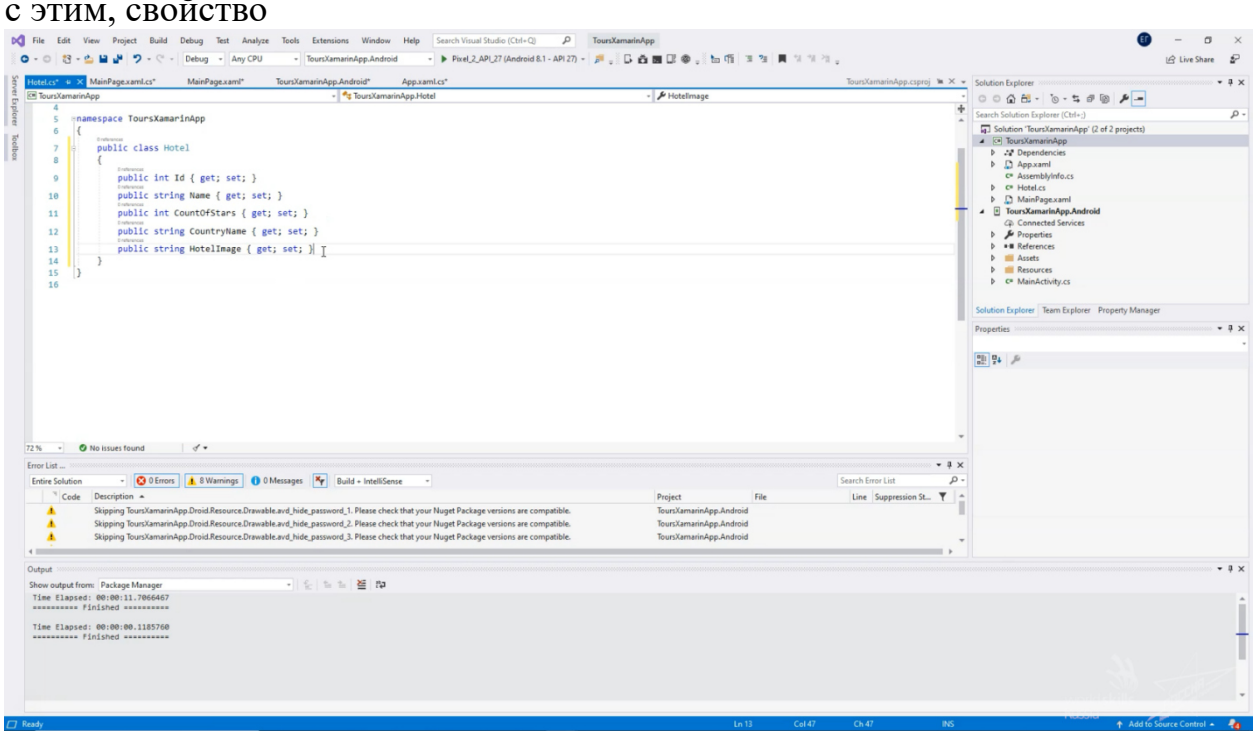
- Сначала тестируем запрос в браузере

- Полученный ответ мы можем скопировать

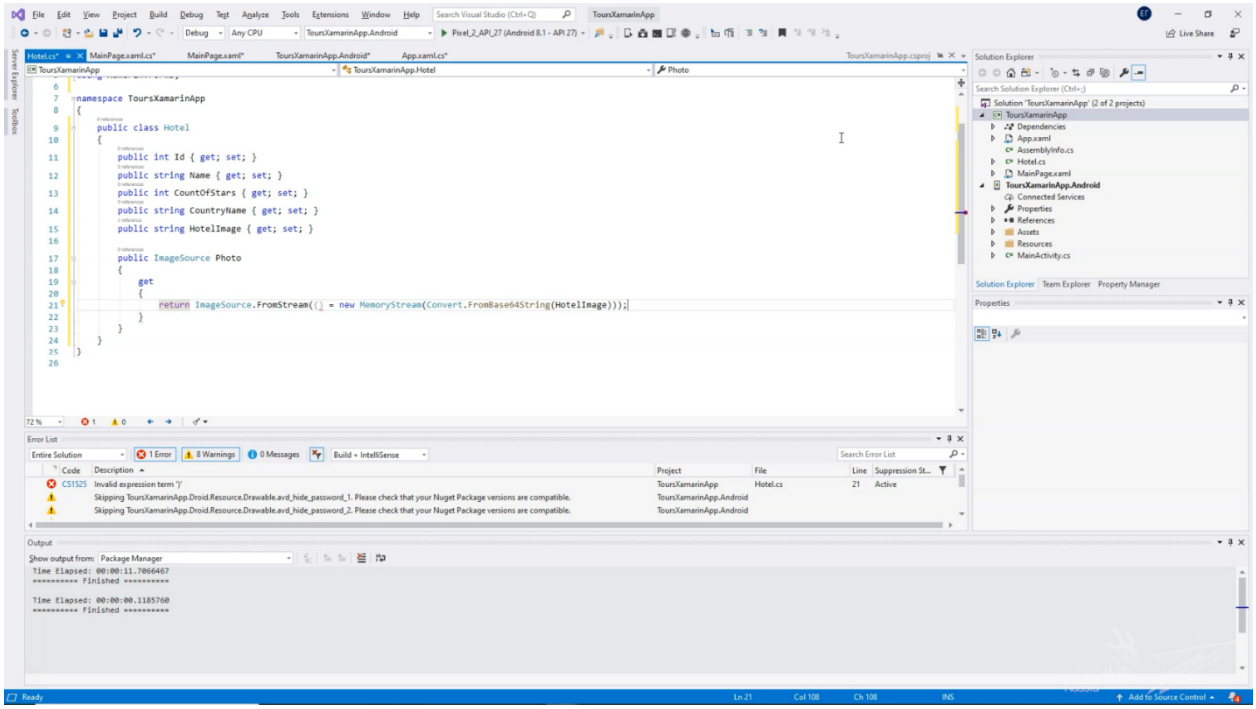
- Создаем новый класс Hotel в нашем проекте. И с помощью специальной вставки в группе Past Special, мы можем вставить скопированный текст в виде JSON-объектов



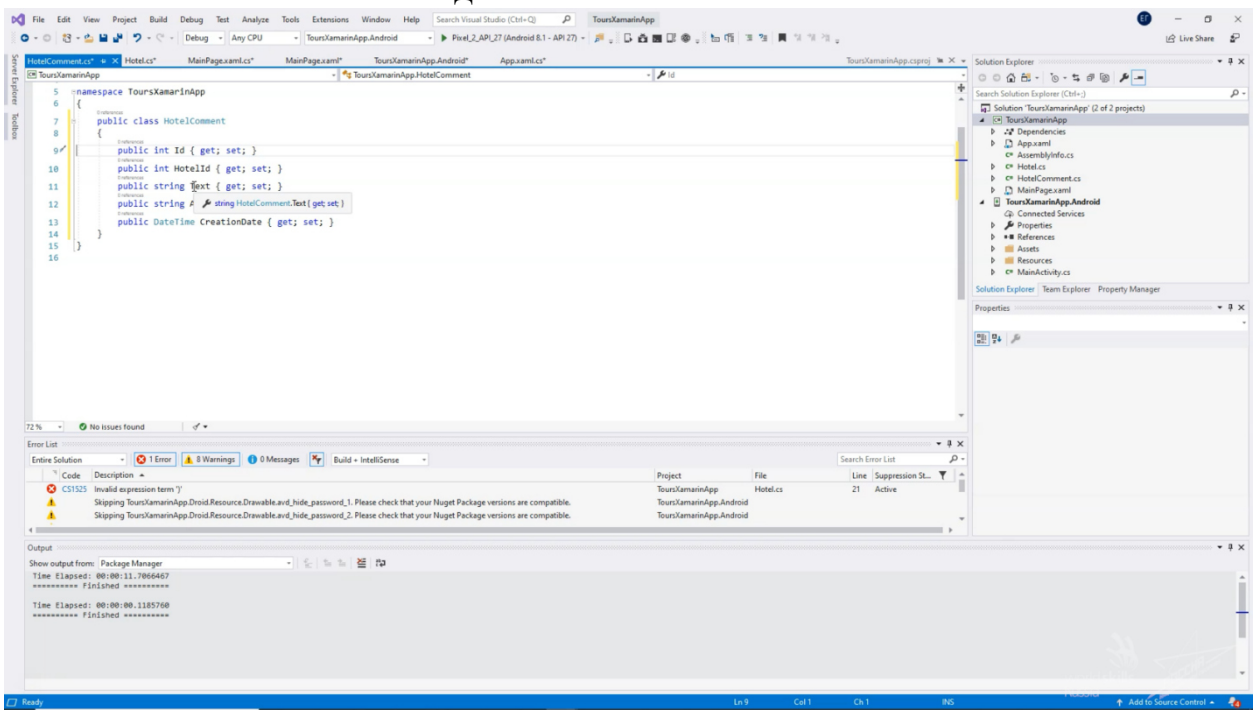
Т. е. мы проанализировали полученный ответ и создаем в соответствии с этим, свойство



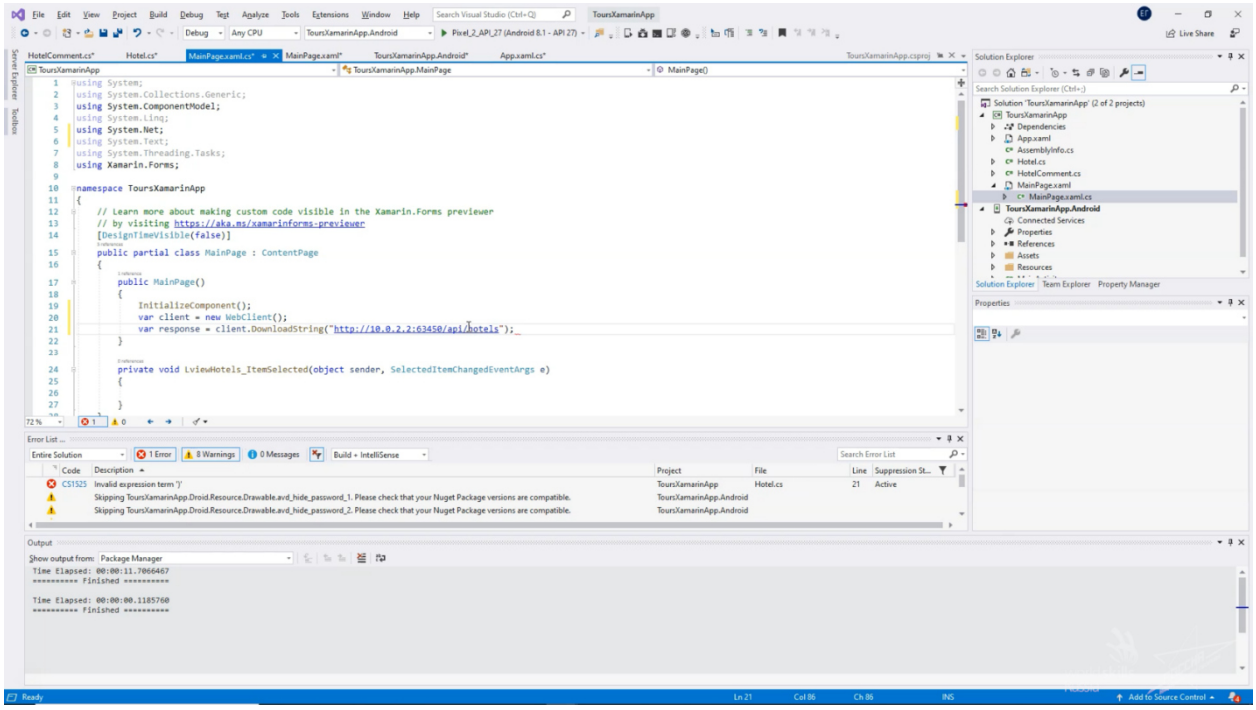
- Также мы можем добавить свое свойство — оно нам пригодится для вывода изображения



- По аналогии создаем класс HotelComment

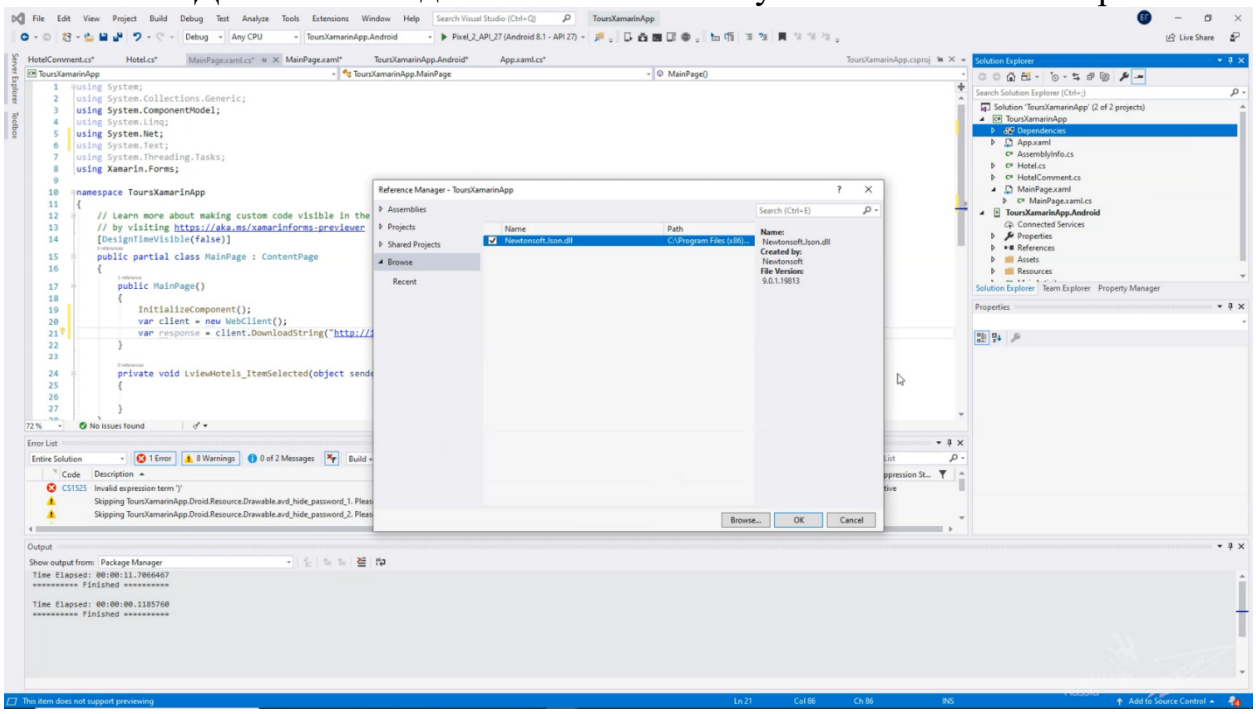


4. На странице MainPage мы получаем список отелей с помощью класса WebClient, где указываем строку подключения, также, как это было в Android 10.0.2.2.

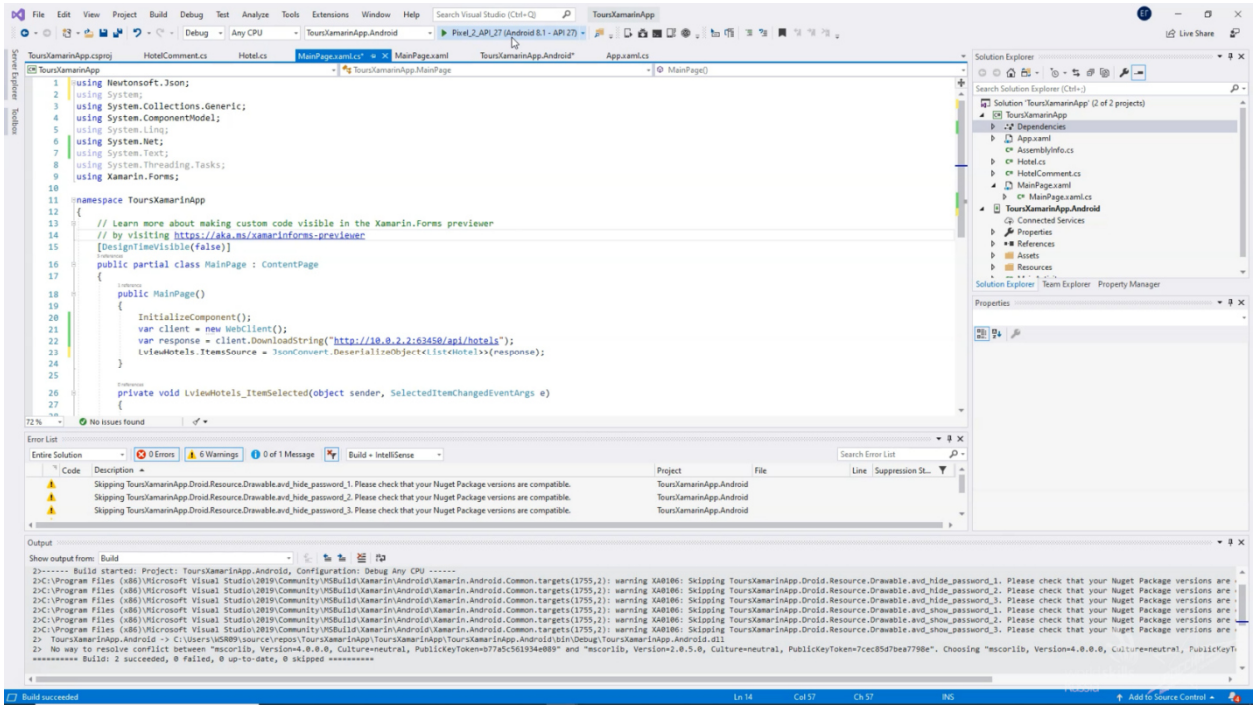


5. Результат возвращается нам в формате json, который мы будем обрабатывать. Удобнее всего использовать библиотеку Newtonsoft. Json, позволяющую конвертировать объекты в формат типа json и обратно. Эти процессы называются сериализацией и десериализацией соответственно.

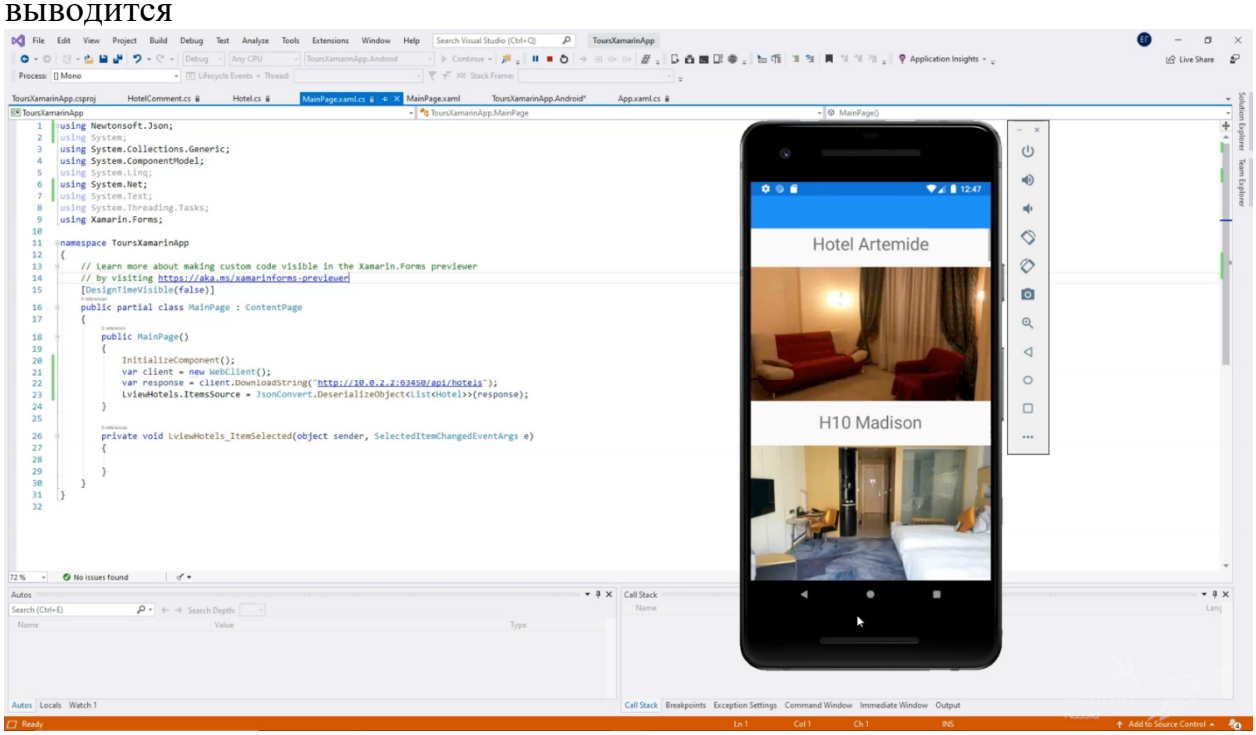
- Для начала подключаем библиотеку Newtonsoft. Json в проект



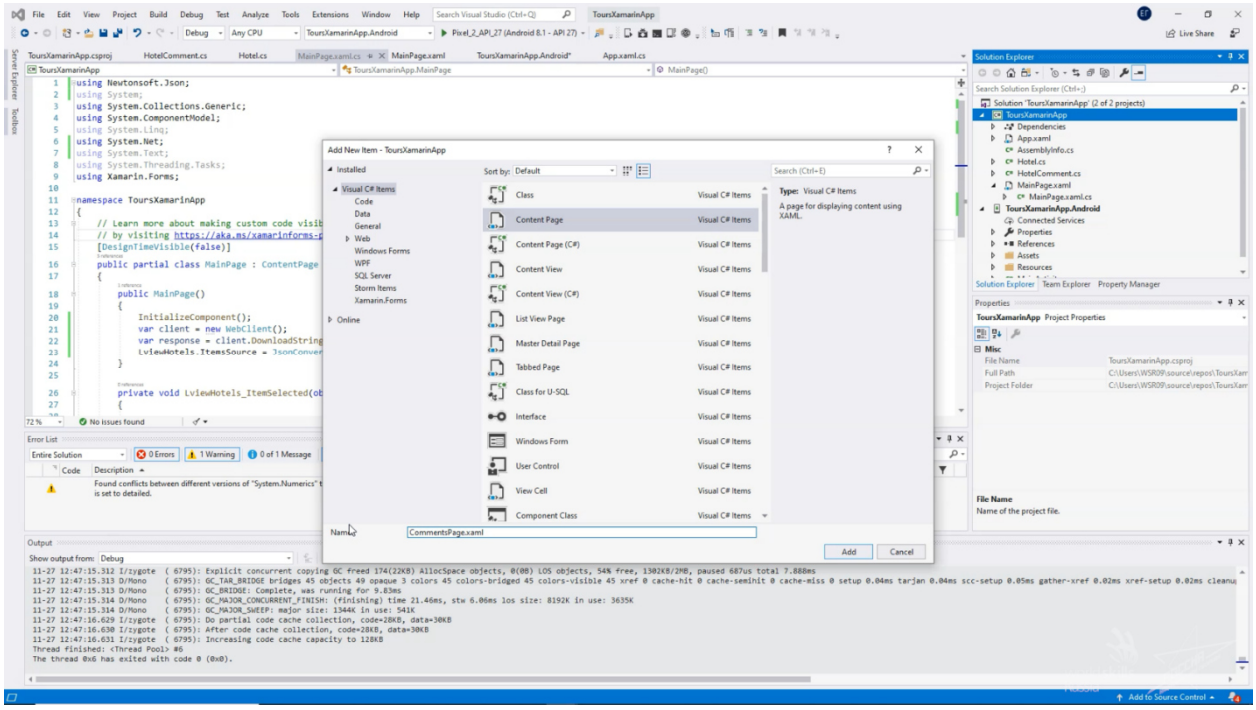
- Десериализуем ответ в список отелей и установливаем в качестве источника данных ListView. Не забудьте пересобрать проект, прежде чем это делать



- Запускаем созданный проект на эмуляторе. Список отелей выводится

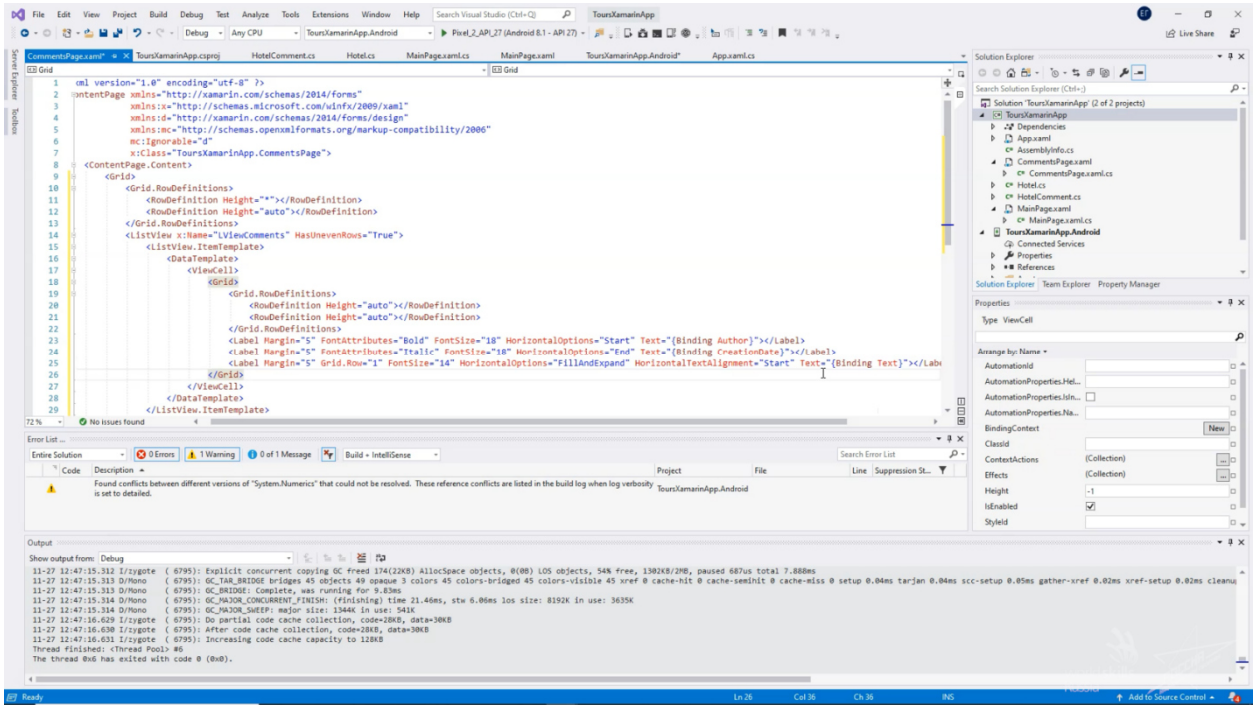


6. Теперь создаем вторую страницу CommentsPage для просмотра комментариев для отеля и добавления новых



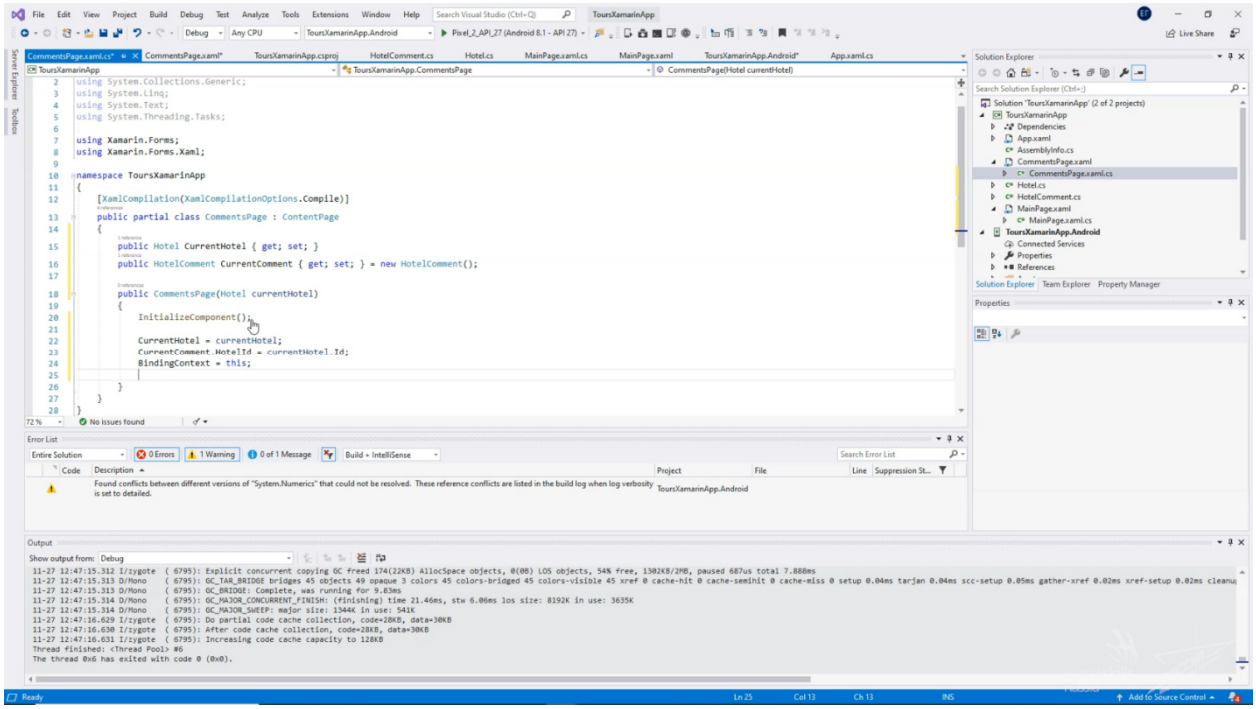
7. Добавляем ListView для вывода комментариев

- указываем динамический размер элементов, привязку данных — как будет отображаться комментарий
- выполняем привязку к указанию автора, содержимого и даты создания

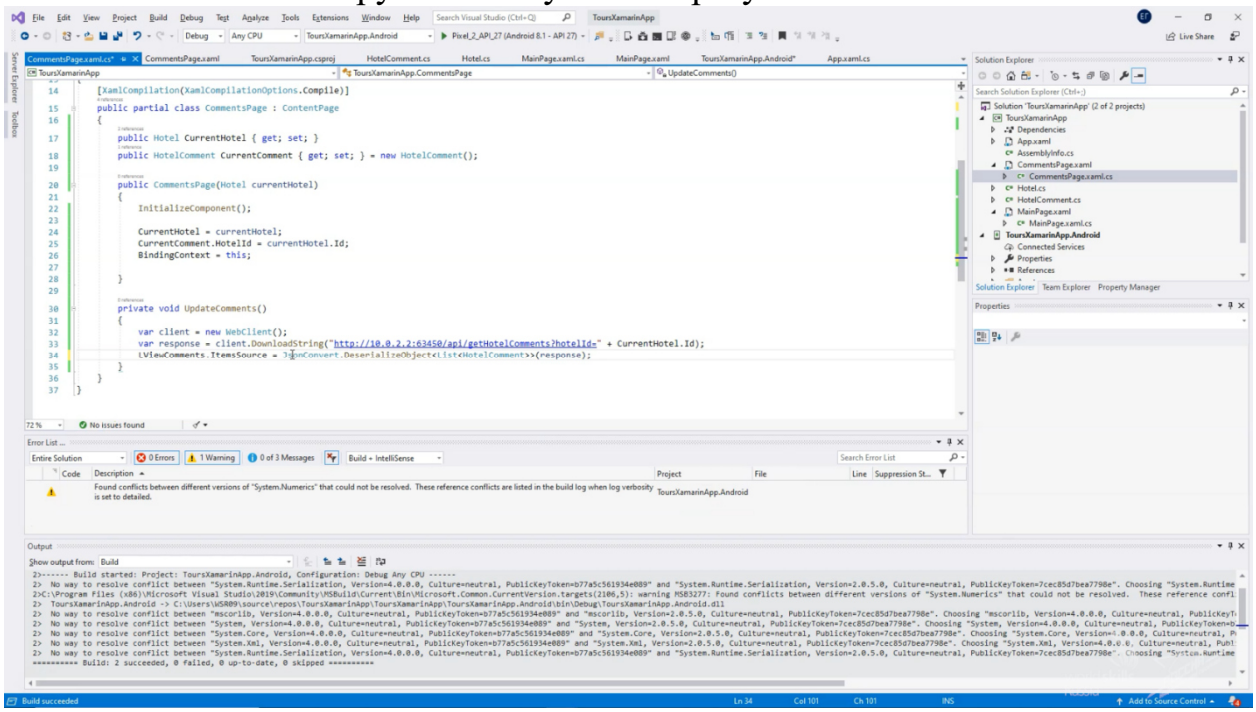


8. В качестве источника данных для привязки мы можем использовать объект любого публичного класса. В том числе и CommentsPage

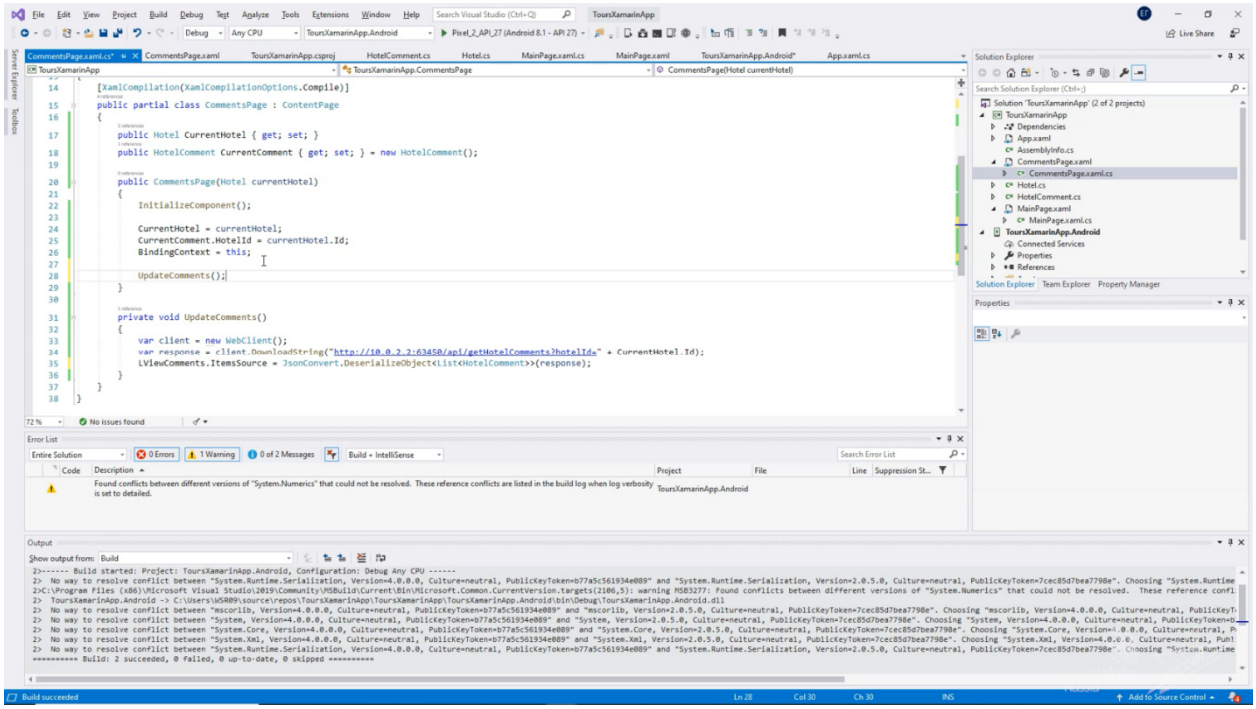
- Поработаем с конструктором новой страницы CommentsPage, передавая в качестве параметра объект выбранного отеля. Создаем дополнительные свойства для биндинга



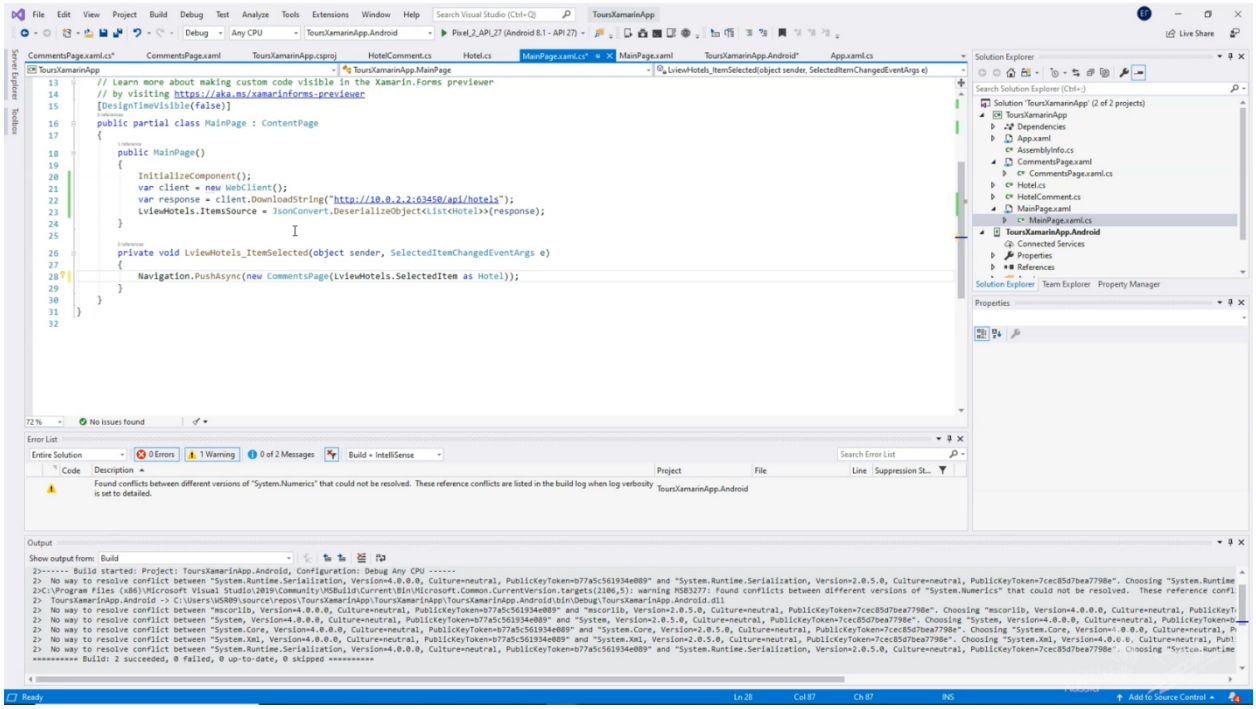
- Создаем метод для обновления списка комментариев, где получаем ответ с помощью запроса GetHotelsComments, и десериализуем ответ. После этого загружаем полученный результат в список



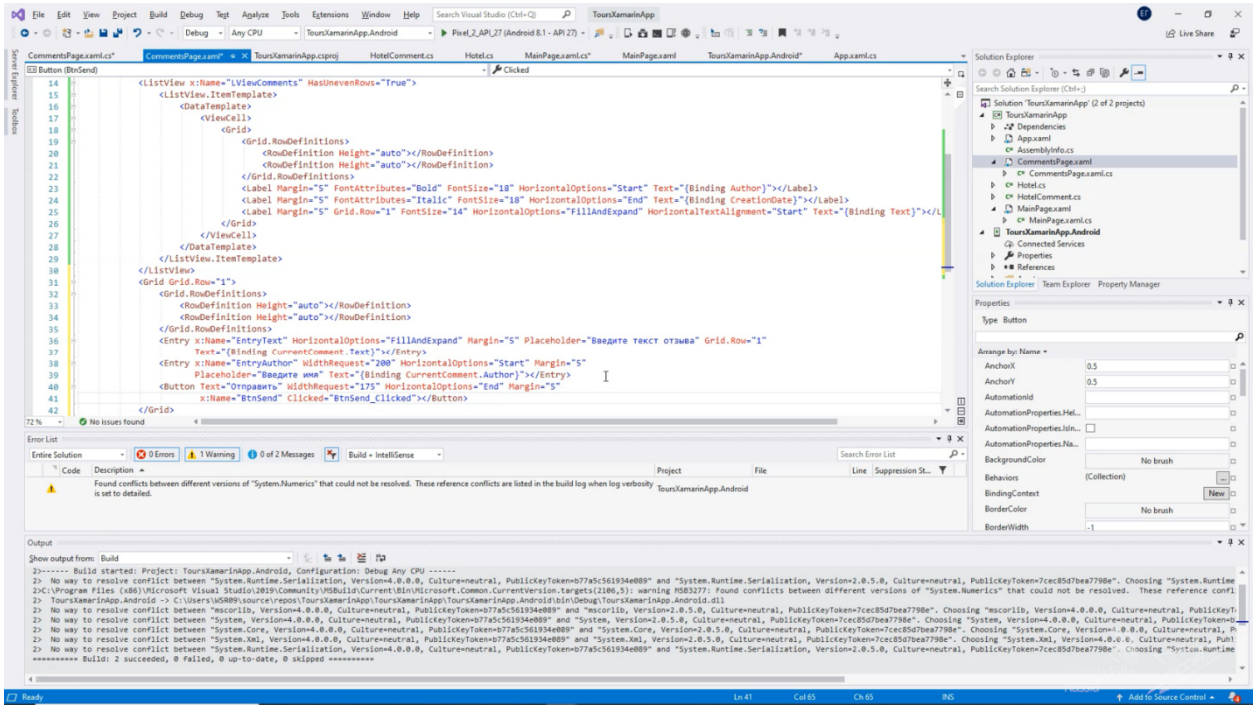
- Вызываем метод updateComments после отображения страницы



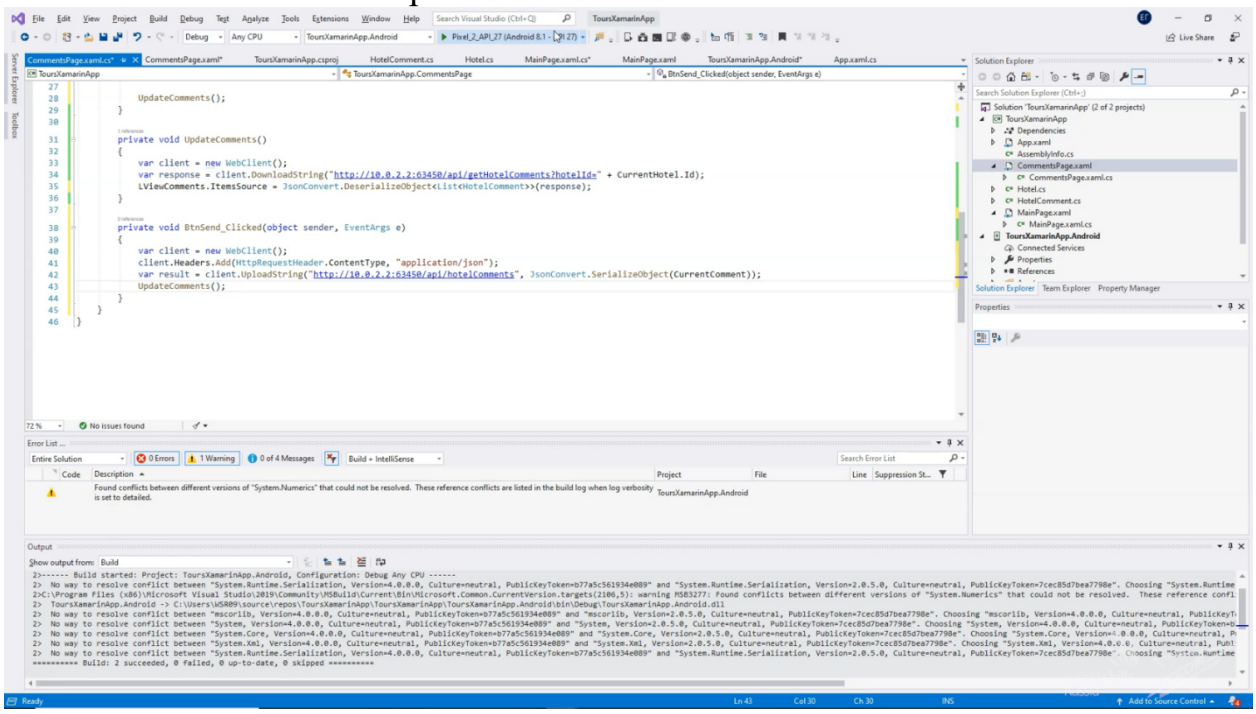
- Теперь создаем переход на эту страницу с главной. С помощью метода PushAsync мы будем передавать экземпляр выбранного отеля в списке



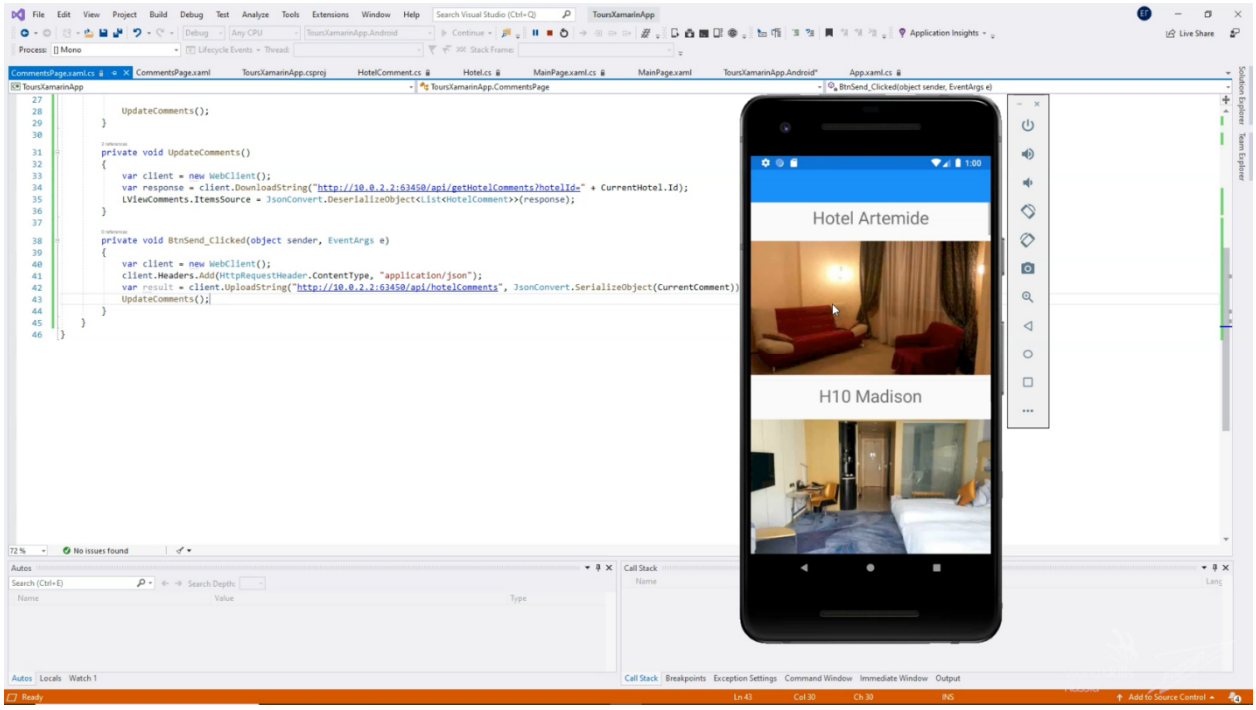
- Делаем остальные элементы управления для добавления комментария. Сразу делаем привязку и обрабатываем клик по кнопке «отправить»

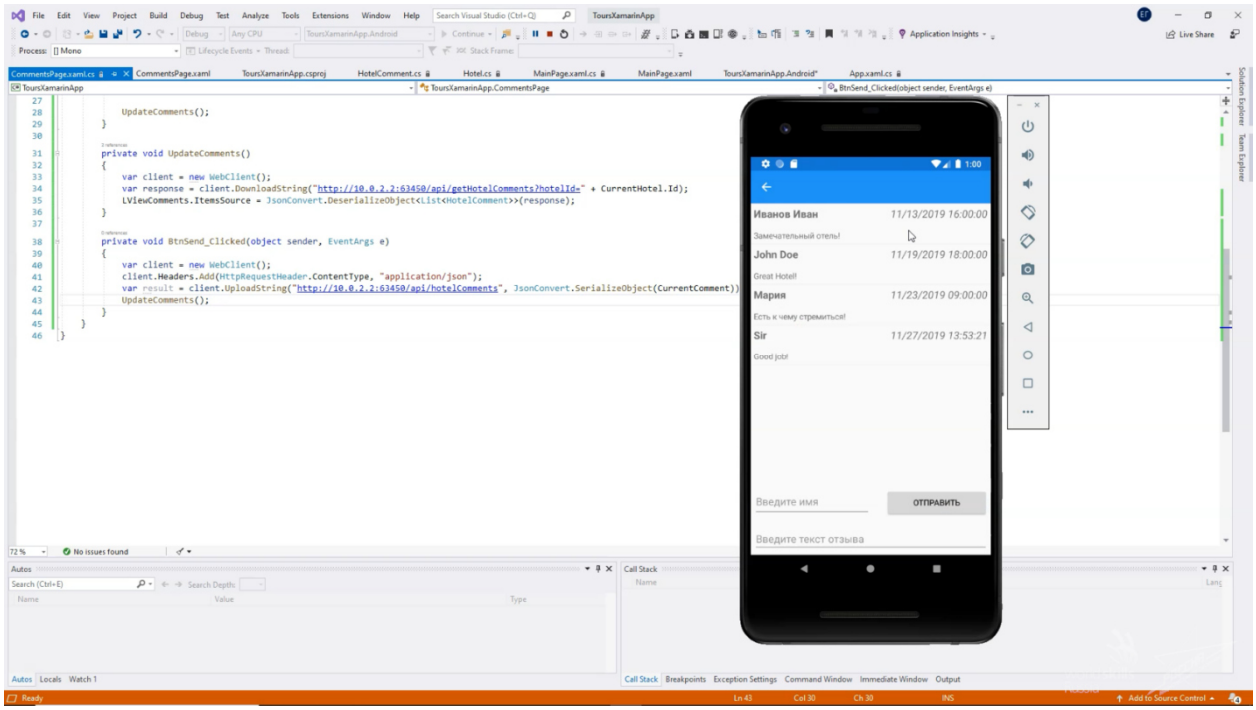


- Используем метод POST для загрузки комментария. В нем мы будем использовать метод UploadString для загрузки данных и передавать туда тот объект, который мы создали. При этом его нужно будет сериализовать. Не забудьте указать ContentType у вашего запроса и после этого обновить комментарии



9. Запускаем и тестируем созданный функционал. Выбрав отель, запускаем комментарии, которые к нему относятся, и можем добавить свой





Практическая работа № 13. Работа с БД My/MS-SQL при помощи ADO.NET Entity Framework

Краткие теоретические сведения

Entity Framework Core (EF Core) представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. EF Core является ORM-инструментом (object-relational mapping - отображения данных на реальные объекты). То есть EF Core позволяет работать базами данных, но представляет собой более

высокий уровень абстракции: EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Entity Framework Core поддерживает множество различных систем баз данных. Таким образом, мы можем через EF Core работать с любой СУБД, если для нее имеется нужный провайдер.

По умолчанию на данный момент Microsoft предоставляет ряд встроенных провайдеров: для работы с MS SQL Server, для SQLite, для PostgreSQL. Также имеются провайдеры от сторонних поставщиков, например, для MySQL.

Также стоит отметить, что EF Core предоставляет универсальный API для работы с данными. И если, к примеру, мы решим сменить целевую СУБД, то основные изменения в проекте будут касаться прежде всего конфигурации и настройки подключения к соответствующим провайдерам. А код, который непосредственно работает с данными, получает данные, добавляет их в БД и т.д., останется прежним.

Entity Framework Core многое унаследовал от своих предшественников, в частности, Entity Framework 6. В тоже время надо понимать, что EF Core - это не новая версия по отношению к EF 6, а совершенно иная технология, хотя в целом принципы работы у них будут совпадать. Поэтому в рамках EF Core используется своя система версий. Текущая версия - 5.0 была выпущена в ноябре 2020 года. И технология продолжает развиваться.

Как технология доступа к данным Entity Framework Core может использоваться на различных платформах стека .NET. Это и стандартные платформы типа Windows Forms, консольные приложения, WPF, UWP и ASP.NET Core. При этом кроссплатформенная природа EF Core позволяет задействовать ее не только на ОС Windows, но и на Linux и Mac OS X.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность определяет набор данных, которые связаны с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их коллекциями.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст. Свойства необязательно представляют простые данные типа int или string, но могут

также представлять и более комплексные типы данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework Core, как технологии ORM, является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем создавать различные запросы на выборку объектов, в том числе связанных различными ассоциативными связями. А Entity Framework при выполнение запроса транслирует выражения LINQ в выражения, понятные для конкретной СУБД (как правило, в выражения SQL).

Основная функциональность Entity Framework Core сосредоточена в следующих пакетах:

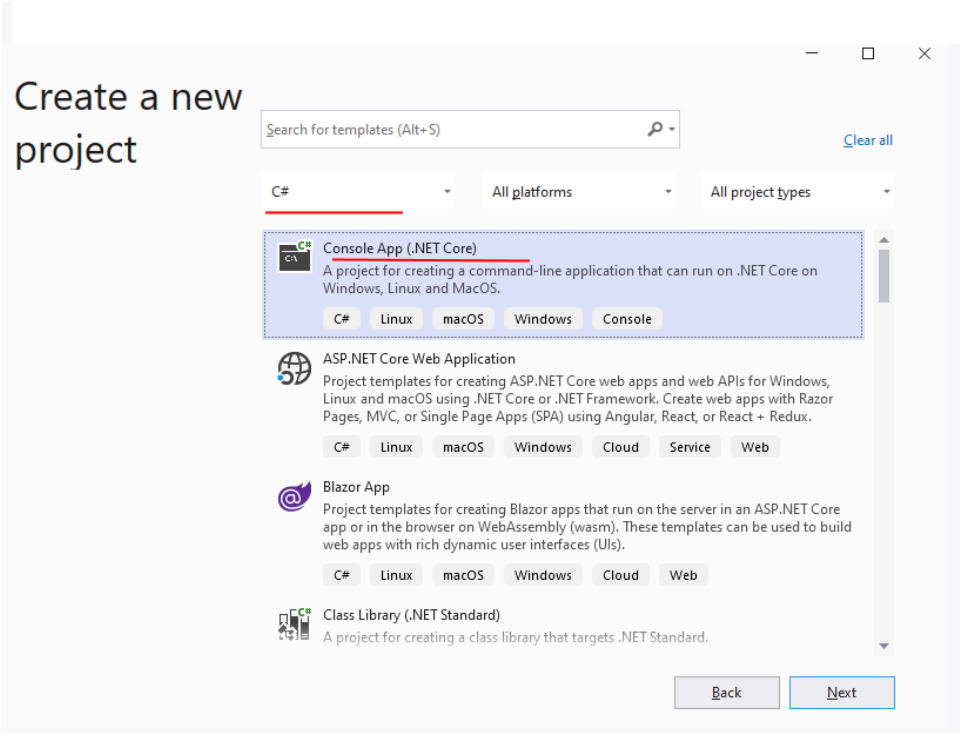
- Microsoft.EntityFrameworkCore:основной пакет EF Core
- Microsoft.EntityFrameworkCore.SqlServer: представляет функциональность провайдера для Microsoft SQL Server и SQL Azure
- Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite: предоставляет поддержку географических типов (spatial types) для SQL Server
- Microsoft.EntityFrameworkCore.Sqlite: представляет функциональность провайдера для SQLite и включает нативные бинарные файлы для движка базы данных
- Microsoft.EntityFrameworkCore.Sqlite.Core: представляет функциональность провайдера для SQLite, но в отличие от предыдущего пакета не содержит нативные бинарные файлы для движка базы данных
- Microsoft.EntityFrameworkCore.Sqlite.NetTopologySuite: предоставляет поддержку географических типов (spatial types) для SQLite
- Microsoft.EntityFrameworkCore.Cosmos: представляет функциональность провайдера для Azure Cosmos DB
- Microsoft.EntityFrameworkCore.InMemory: представляет функциональность провайдера базы данных в памяти
- Microsoft.EntityFrameworkCore.Tools: содержит команды EF Core PowerShell для Visual Studio Package Manager Console; применяется в Visual Studio для миграций и генерации классов по готовой бд
- Microsoft.EntityFrameworkCore.Design: содержит вспомогательные компоненты EF Core, применяемые в процессе разработки

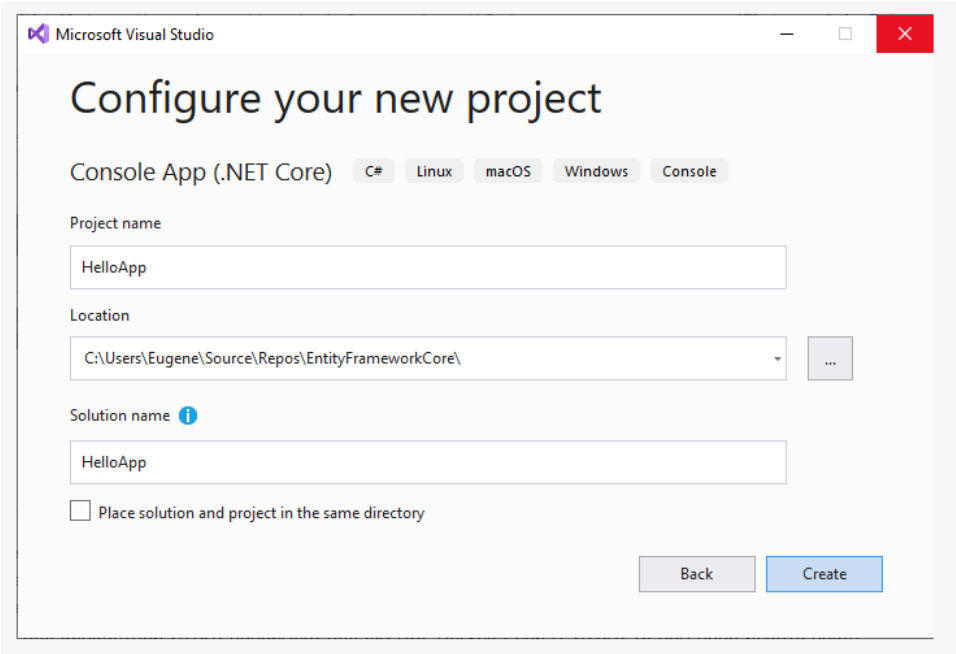
- Microsoft.EntityFrameworkCore.Proxies: хранит функциональность для так называемой "ленивой загрузки" (lazy-loading) и прокси остлеживания изменений
- Microsoft.EntityFrameworkCore.Abstractions: содержит набор абстракций EF Core, которые не зависят от конкретной СУБД
- Microsoft.EntityFrameworkCore.Relational: хранит компоненты EF Core для провайдеров реляционных СУБД
- Microsoft.EntityFrameworkCore.Analyzers: содержит функционал анализаторов C# для EF Core

Первое приложение на EF Core

Платформу Entity Framework Core можно применять в различных технологиях стека .NET. В данном случае мы будем рассматривать базовые моменты платформы на примере консольных приложений, как наиболее простых и не содержащих никакого лишнего кода. Но в последствии также затронем применение EF Core и в других технологиях на конкретных примерах.

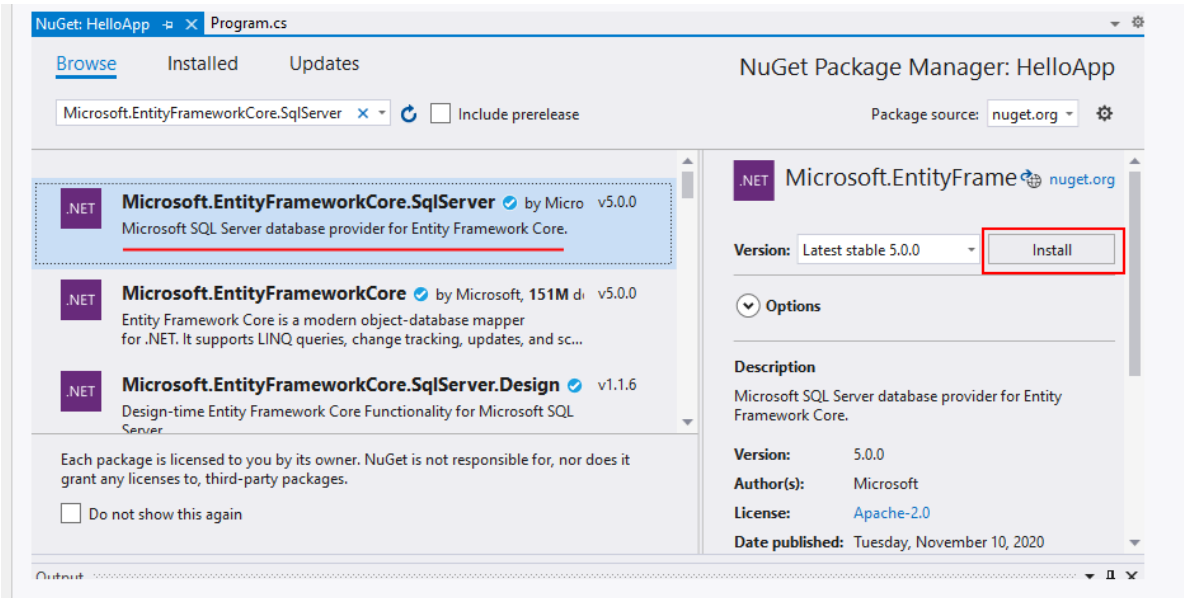
Итак, создадим первое приложение с использованием Entity Framework Core. Для этого создадим в Visual Studio 2019 новый проект по типу **Console App (.NET Core)**, который назовем HelloApp:





Итак, Visual Studio создает проект с базовой примитивной структурой, где можно выделить собственно файл логики приложения - файл *Program.cs*. И чтобы начать работать с EntityFramework Core, нам необходимо вначале добавить в проект пакет EntityFramework Core. Для этого перейдем в проекте к пакетному менеджеру NuGet.

Однако здесь мы ищем не общий пакет для Entity Framework Core, а пакет для конкретной СУБД. Так, в данном случае мы будем использовать MS SQL Server в качестве СУБД, поэтому нам надо искать пакет **Microsoft.EntityFrameworkCore.SqlServer**:



В качестве альтернативы для добавления пакетов можно использовать **Package Manager Console**. Для этого в меню Visual Studio

перейдем к пункту **Tools -> NuGet Package Manager -> Package Manager Console**

В открывшемся внизу в Visual Studio окне Package Manager Console введем команду:

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

После выполнения этой команды выполним вторую команду:

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

Итак, необходимые пакеты добавлены. Теперь мы можем их использовать.

Далее нам надо определить модель, которая будет описывать данные. Пусть наше приложение будет посвящено работе с пользователями. Поэтому добавим в проект новый класс User:

```
public class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Это обычный класс, который содержит несколько свойств. Каждое свойство будет сопоставляться с отдельным столбцом в таблице из бд.

Надо отметить, что Entity Framework требует определения ключа элемента для создания первичного ключа в таблице в бд. По умолчанию при генерации бд EF в качестве первичных ключей будет рассматривать свойства с именами Id или [Имя_класса]Id (то есть UserId).

Взаимодействие с базой данных в Entity Framework Core происходит посредством специального класса - контекста данных. Поэтому добавим в наш проект новый класс, который назовем ApplicationContext и который будет иметь следующий код:

```
using Microsoft.EntityFrameworkCore;

namespace HelloApp
{
    public class ApplicationContext : DbContext
    {
        public DbSet<User> Users { get; set; }

```

```
9           public ApplicationContext()
10          {
11              Database.EnsureCreated();
12          }
13
14          protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilde
15          {
16              optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=hello
17          }
18      }
19  }
```

Основу функциональности Entity Framework Core для работы с MS SQL Server составляют классы, которые располагаются в пространстве имен **Microsoft.EntityFrameworkCore**. Среди всего набора классов этого пространства имен следует выделить следующие:

- **DbContext**: определяет контекст данных, используемый для взаимодействия с базой данных
- **DbSet/DbSet<TEntity>**: представляет набор объектов, которые хранятся в базе данных
- **DbContextOptionsBuilder**: устанавливает параметры подключения

В любом приложении, работающим с БД через Entity Framework, нам нужен будет контекст (класс производный от DbContext). В данном случае таким контекстом является класс ApplicationContext.

И также в классе определено одно свойство Users, которое будет хранить набор объектов User. В классе контекста данных набор объектов представляет класс DbSet<T>. Через это свойство будет осуществляться связь с таблицей объектов User в бд.

Кроме того, для настройки подключения нам надо переопределить метод **OnConfiguring**. Передаваемый в него параметр класса DbContextOptionsBuilder с помощью метода UseSqlServer позволяет настроить строку подключения для соединения с MS SQL Server. В данном случае мы определяем, что в качестве сервера будет использоваться движок localdb, который предназначен специально для разработки, ("Server=(localdb)\\mssqllocaldb"), а файл базы данных будет называться helloappdb ("Database=helloappdb").

И также стоит отметить, что по умолчанию у нас нет базы данных. Поэтому в конструкторе класса контекста определен вызов метода Database.EnsureCreated(), который при создании контекста

автоматически проверит наличие базы данных и, если она отсуствует, создаст ее.

Теперь определим сам код программы, который будет взаимодействовать с созданной БД. Для этого изменим класс Program следующим образом:

```
using System;
using System.Linq;

namespace HelloApp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            using (ApplicationContext db = new ApplicationContext())
            {
                // создаем два объекта User
                User user1 = new User { Name = "Tom", Age = 33 };
                User user2 = new User { Name = "Alice", Age = 26 };

                // добавляем их в бд
                db.Users.Add(user1);
                db.Users.Add(user2);
                db.SaveChanges();
                Console.WriteLine("Объекты успешно сохранены");

                // получаем объекты из бд и выводим на консоль
                var users = db.Users.ToList();
                Console.WriteLine("Список объектов:");
                foreach (User u in users)
                {
                    Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
                }
            }
            Console.Read();
        }
    }
}
```

Так как класс ApplicationContext через базовый класс DbContext реализует интерфейс IDisposable, то для работы с ApplicationContext с автоматическим закрытием данного объекта мы можем использовать конструкцию using.

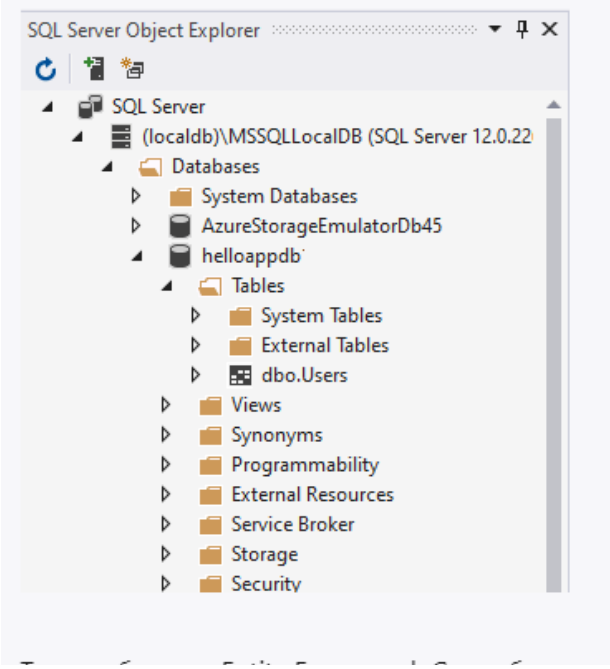
В конструкции using создаются два объекта User и добавляются в базу данных. Для их сохранения нам достаточно использовать метод Add: db.Users.Add(user1)

Чтобы получить список данных из бд, достаточно воспользоваться свойством Users контекста данных: db.Users

В результате после запуска программа выведет на консоль:

```
Объекты успешно сохранены
Список объектов:
1.Tom - 33
2.Alice - 26
```

И после добавления мы можем найти базу данных в окне SQL Server Object Explorer (открыть данное окно можно через меню **View->SQL Server Object Explorer**). Там мы можем раскрыть и посмотреть содержимое ее таблиц:

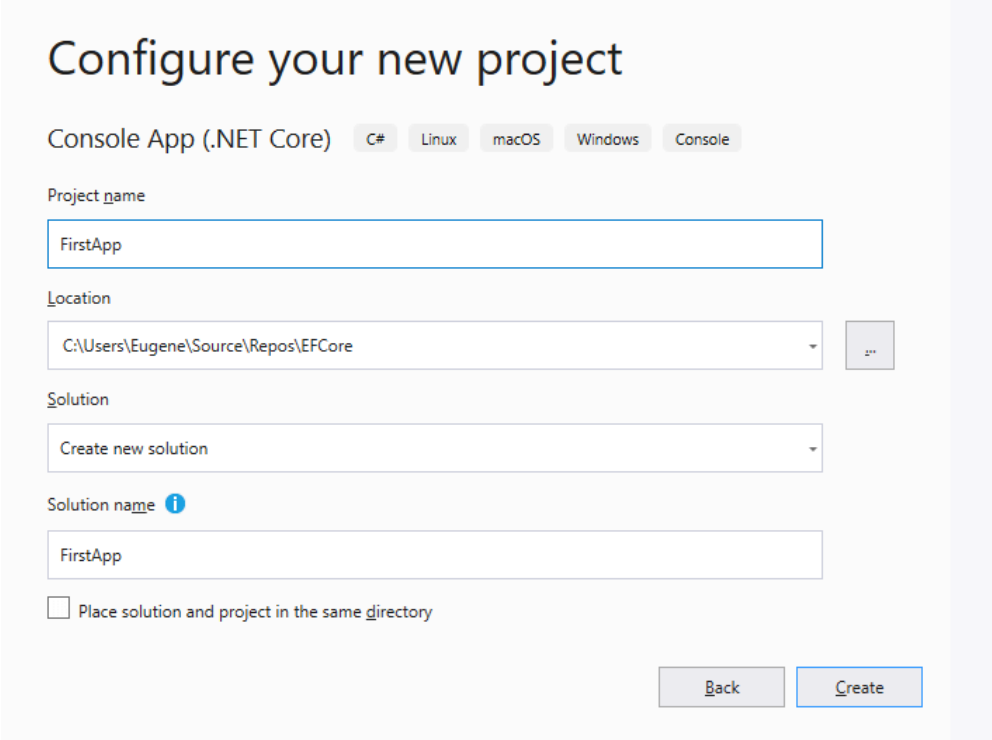


Таким образом, Entity Framework Core обеспечивает простое и удобное управление объектами из базы данных. При том в данном случае нам не надо даже создавать базу данных и определять в ней таблицы. Entity Framework все сделает за нас на основе определения класса контекста данных и классов моделей. И если база данных уже имеется, то EF не будет повторно создавать ее.

Подключение к существующей базе данных

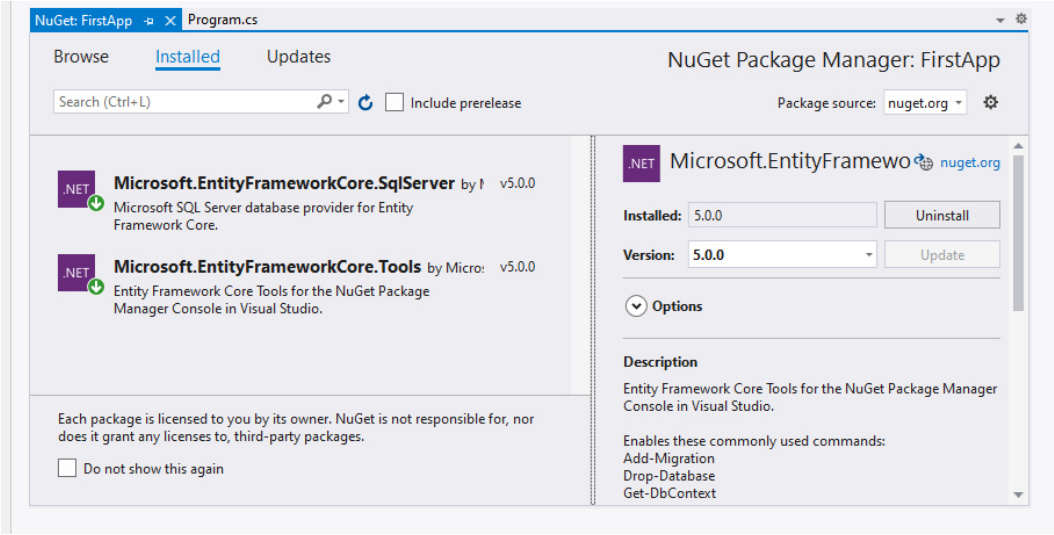
В прошлой теме база данных изначально не существовала и создавалась автоматически в процессе запуска приложения в соответствии с определением классов моделей и класса контекста данных. Но нередко база данных для подключения уже имеется. Рассмотрим, как мы можем подключаться к уже существующей базе данных.

Для подключения к существующей базе данных создадим новый проект. Как и в прошлой теме это будет проект по типу **Console App (.NET Core)**, который назовем FirstApp.

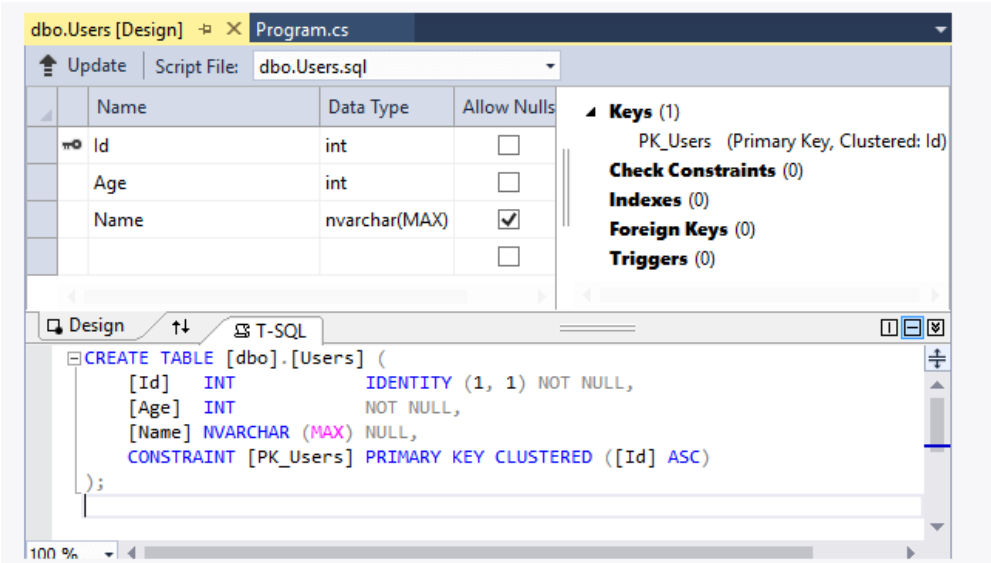


Для работы с существующей БД MS SQL Server нам надо добавить два пакета:

- **Microsoft.EntityFrameworkCore.SqlServer** (представляет функциональность Entity Framework для работы с MS SQL Server)
- **Microsoft.EntityFrameworkCore.Tools** (необходим для создания классов по базе данных, то есть reverse engineering)



К примеру возьмем базу данных, созданную в прошлой теме. Эта база данных называется helloappdb.mdf и имеет одну таблицу Users с тремя столбцами Id, Name и Age.



Чтобы подключаться к базе данных, нам надо будет добавить в проект классы моделей, которые соответствуют определениям таблиц, и класс контекста данных, который соответствует БД. То есть в данном случае мы могли бы вручную добавить в проект классы моделей и класс контекста данных, как в прошлой теме.

Однако добавление классов вручную имеет свои недостатки. Так, если база данных имеет не одну, а множество таблиц, связанных различными ключами, то у нас может возникнуть проблема, как все эти отношения отобразить между класса на C#. Ну и кроме того, это просто долго и может занять некоторое время.

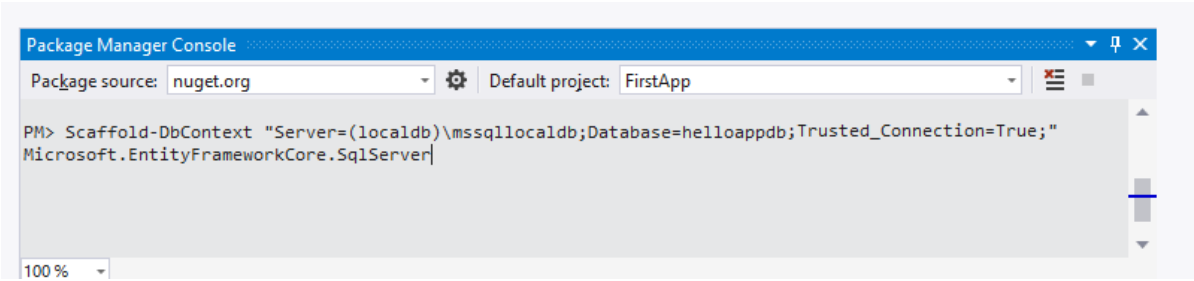
Для решения этих проблем в Entity Framework Core предусмотрена функция **Reverse Engineering**, которая позволяет автоматически создать все

необходимые классы по базе данных. Чтобы воспользоваться этой функцией перейдем в Visual Studio к окну **Package Manager Console**. Его открыть можно, перейдя в меню **Tools** –> **NuGet Package Manager** –> **Package Manager Console**

Далее в Package Manager Console выполним следующую команду:

```
1   Scaffold-DbContext "Server=(localdb)\mssqllocal
    Microsoft.EntityFrameworkCore.SqlServer
```

Здесь в качестве параметра команде Scaffold-DbContext передается строка подключения с указанием сервера и названием базы данных.



После выполнения этой команды в проект будет добавлен класс Users:

```
1   public partial class User
2   {
3       public int Id { get; set; }
4       public string Name { get; set; }
5       public int Age { get; set; }
6   }
```

И также будет добавлен класс контекста данных, который будет называться по имени базы данных плюс суффикс "Context":

```
1   using System;
2   using Microsoft.EntityFrameworkCore;
3   using Microsoft.EntityFrameworkCore.Metadata;
4
5   #nullable disable
6
7   namespace FirstApp
8   {
9       public partial class helloappdbContext : DbContext
10      {
11          public helloappdbContext()
12          {
13          }
```

```
14
15          public helloappdbContext(DbContextOptions<helloappdbContext> options)
16              : base(options)
17          {
18          }
19
20          public virtual DbSet<User> Users { get; set; }
21
22          protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilde
23          {
24              if (!optionsBuilder.IsConfigured)
25              {
26                  optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=he
27              }
28          }
29
30          protected override void OnModelCreating(ModelBuilder modelBuilder)
31          {
32              OnModelCreatingPartial(modelBuilder);
33          }
34
35          partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
36      }
37  }
```

И после генерации данных классов мы сможем работать с базой данных. Для этого изменим код класса Program:

```
1   using System;
2   using System.Linq;
3
4   namespace FirstApp
5   {
6       class Program
7       {
8           static void Main(string[] args)
9           {
10              using (helloappdbContext db = new helloappdbContext())
11              {
12                  // получаем объекты из бд и выводим на консоль
13                  var users = db.Users.ToList();
14                  Console.WriteLine("Список объектов:");
15                  foreach (User u in users)
16                  {
17                      Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
```

```
18                }
19            }
20            Console.ReadKey();
21        }
22    }
23 }
```

Основные операции с данными. CRUD

Большинство операций с данными так или иначе представляют собой CRUD операции (Create, Read, Update, Delete), то есть создание, получение, обновление и удаление. Entity Framework Core позволяет легко выполнять все эти действия.

Для примера создадим проект по типу **Console App (.NET Core)**. И после создания проекта сразу добавим в него функциональность EF Core. Для этого в проект через NuGet пакет **Microsoft.EntityFrameworkCore.SqlServer**.

Затем добавим в проект класс User, объекты которого будут храниться в базе данных:

```
1  public class User
2  {
3      public int Id { get; set; }
4      public string Name { get; set; }
5      public int Age { get; set; }
6  }
```

И добавим класс контекста данных ApplicationContext:

```
1   using Microsoft.EntityFrameworkCore;
2
3   namespace HelloApp
4   {
5       public class ApplicationContext : DbContext
6       {
7           public DbSet<User> Users { get; set; }
8           public ApplicationContext()
```

```
9           {
10              Database.EnsureCreated();
11          }
12
13          protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilde
14          {
15              optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=hel
16          }
17      }
18  }
```

Далее определим в классе Program все базовые операции с данными:

```
1   using System;
2   using System.Linq;
3
4   namespace HelloApp
5   {
6       public class Program
7       {
8           public static void Main(string[] args)
9           {
10              // Добавление
11              using (ApplicationContext db = new ApplicationContext())
12              {
13                  User user1 = new User { Name = "Tom", Age = 33 };
14                  User user2 = new User { Name = "Alice", Age = 26 };
15
16                  // Добавление
17                  db.Users.Add(user1);
18                  db.Users.Add(user2);
```

```
19              db.SaveChanges();
20          }
21
22          // получение
23          using (ApplicationContext db = new ApplicationContext())
24          {
25              // получаем объекты из бд и выводим на консоль
26              var users = db.Users.ToList();
27              Console.WriteLine("Данные после добавления:");
28              foreach (User u in users)
29              {
30                  Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
31              }
32          }
33
34          // Редактирование
35          using (ApplicationContext db = new ApplicationContext())
36          {
37              // получаем первый объект
38              User user = db.Users.FirstOrDefault();
39              if(user!=null)
40              {
41                  user.Name = "Bob";
42                  user.Age = 44;
43                  //обновляем объект
44                  //db.Users.Update(user);
45                  db.SaveChanges();
46              }
47              // выводим данные после обновления
```

```
48              Console.WriteLine("\nДанные после редактирования:");
49              var users = db.Users.ToList();
50              foreach (User u in users)
51              {
52                  Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
53              }
54          }
55
56          // Удаление
57          using (ApplicationContext db = new ApplicationContext())
58          {
59              // получаем первый объект
60              User user = db.Users.FirstOrDefault();
61              if (user != null)
62              {
63                  //удаляем объект
64                  db.Users.Remove(user);
65                  db.SaveChanges();
66              }
67              // выводим данные после обновления
68              Console.WriteLine("\nДанные после удаления:");
69              var users = db.Users.ToList();
70              foreach (User u in users)
71              {
72                  Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
73              }
74          }
75          Console.Read();
76      }
```

```
77        }
78    }
```

И после выполнения мы получим следующий консольный вывод:

```
Данные после добавления
1.Tom - 33
2.Alice - 26

Данные после редактирования
1.Bob - 44
2.Alice - 26

Данные после удаления
2.Alice - 26
```

Добавление

Для добавления объекта используется метод **Add**, определенный у класса DbSet, в который передается добавляемый объект:

```
1    db.Users.Add(user2);
2    db.SaveChanges();
```

Метод Add устанавливает значение Added в качестве состояния нового объекта. Поэтому метод db.SaveChanges() сгенерирует выражение INSERT для вставки модели в таблицу.

Если нам надо добавить сразу несколько объектов, то мы можем воспользоваться методом **AddRange()**:

```
1    User user1 = new User { Name = "Tom", Age = 33 };
2    User user2 = new User { Name = "Alice", Age = 26 };
3    db.Users.AddRange(user1, user2);
```

Удаление

Удаление производится с помощью метода **Remove**:

```
1    db.Users.Remove(user);
2    db.SaveChanges();
```

Данный метод установит статус объекта в Deleted, благодаря чему Entity Framework при выполнении метода db.SaveChanges() сгенерирует SQL-выражение DELETE.

Если необходимо удалить сразу несколько объектов, то можно использовать метод **RemoveRange()**:

```
User user1 = db.Users.FirstOrDefault();
User user2 = db.Users.LastOrDefault();
db.Users.RemoveRange(user1, user2);
```

Редактирование

При изменении объекта Entity Framework сам отслеживает все изменения, и когда вызывается метод **SaveChanges()**, будет сформировано SQL-выражение UPDATE для данного объекта, которое обновит объект в базе данных.

Но надо отметить, что в данном случае действие контекста данных ограничивается пределами конструкции using. Но рассмотрим другой пример. Мы получаем объект в одном месте,а обновляем в другом. Например:

```
User user = null;
using (ApplicationContext db = new ApplicationContext())
{
    // получаем объект
    user = db.Users.FirstOrDefault();
    Console.WriteLine("Данные до редактирования:");
    var users = db.Users.ToList();
    foreach (User u in users)
    {
        Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
    }
}
//.....
```

```
15  // Редактирование
16  using (ApplicationContext db = new ApplicationContext())
17  {
18      // Редактирование
19      if (user != null)
20      {
21          user.Name = "Sam";
22          user.Age = 33;
23      }
24      db.SaveChanges();
25      // выводим данные после обновления
26      Console.WriteLine("\nДанные после редактирования:");
27      var users = db.Users.ToList();
28      foreach (var u in users)
29      {
30          Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
31      }
32  }
```

Несмотря на то, что объект user не равен null, имеется в базе данных, но во втором блоке using обновления соответствующего объекта в БД не произойдет. И в этом случае нам надо использовать метод Update:

```
1   // Редактирование
2   using (ApplicationContext db = new ApplicationContext())
3   {
4       // Редактирование
5       if (user != null)
6       {
7           user.Name = "Sam";
8           user.Age = 33;
```

```
9            db.Users.Update(user);
10       }
11       db.SaveChanges();
12       // выводим данные после обновления
13       Console.WriteLine("\nДанные после редактирования:");
14       var users = db.Users.ToList();
15       foreach (var u in users)
16       {
17           Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
18       }
19   }
```

При необходимости обновить одновременно несколько объектов, применяется метод **UpdateRange()**:

```
1    db.Users.UpdateRange(user1, user2);
```

Практическая работа № 14. Разработка WebAPI

Краткие теоретические сведения

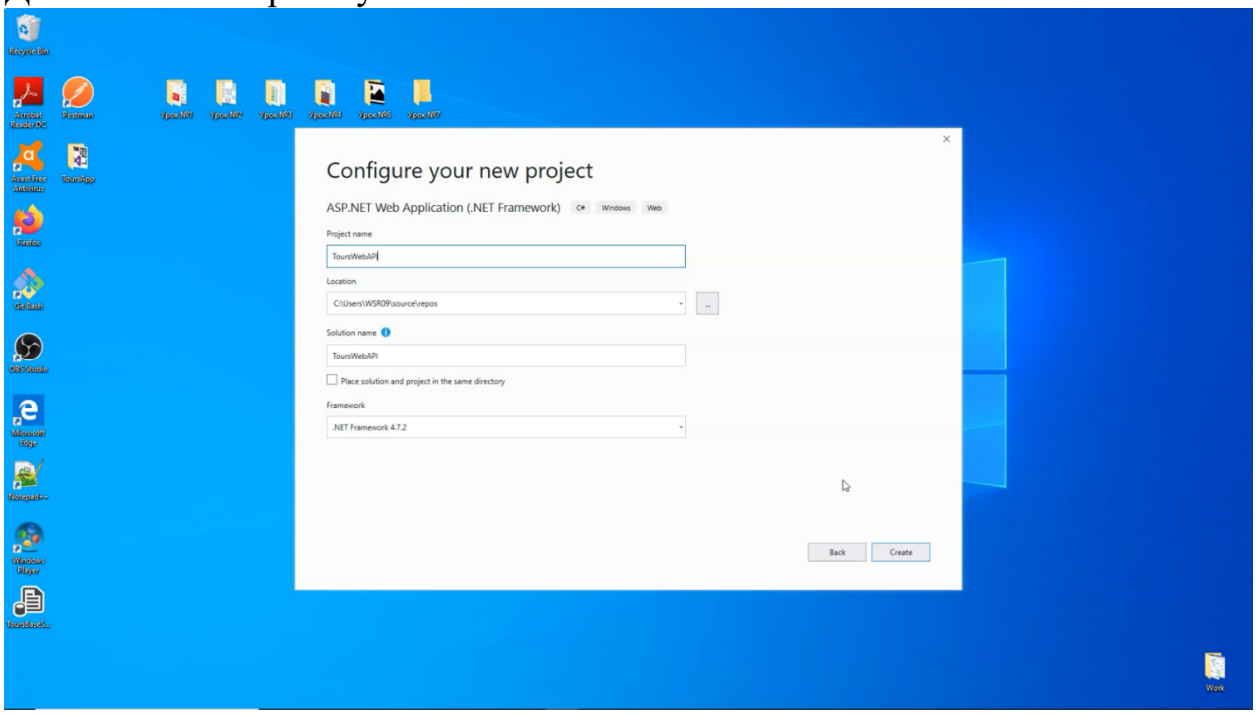
Мы познакомимся с API и попытаемся понять, как же он связывает различные приложения. API — это интерфейс, позволяющий двум независимым компонентам программного обеспечения обмениваться информацией. API играет роль посредника между внутренними и внешними программными функциями, обеспечивая настолько эффективный обмен информацией, что конечный пользователь обычно его просто не замечают.

Например, посредством API может быть передана геолокация из мобильного приложения одного человека на сервер, а затем на мобильное устройство другого. В рамках разработки программного решения можно добиться наибольшей эффективности в том случае, если и настольное, и мобильное приложения будут работать с единой базой данных и получать актуальную информацию в режиме реального времени.

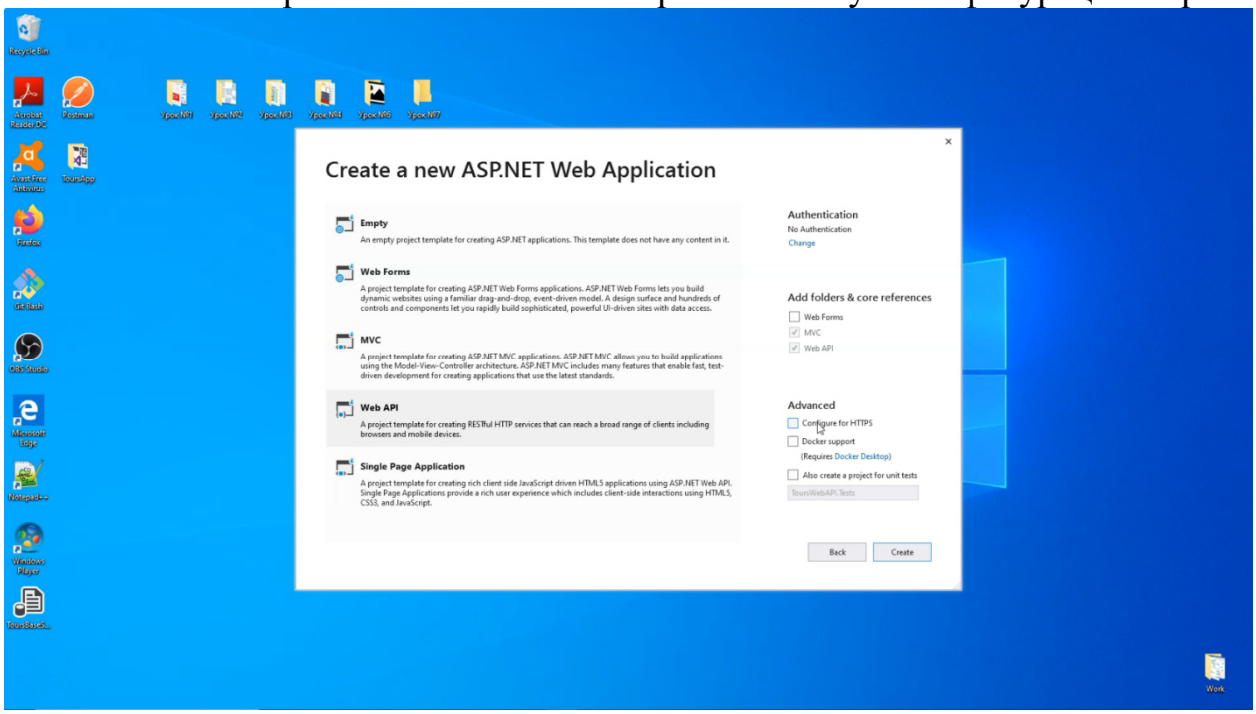
С этой целью мы и создадим свой API, который сам будет работать с базой данных и возвращать ответ в удобном для приложения виде

1. Создаем проект «Web API»

Даем название проекту

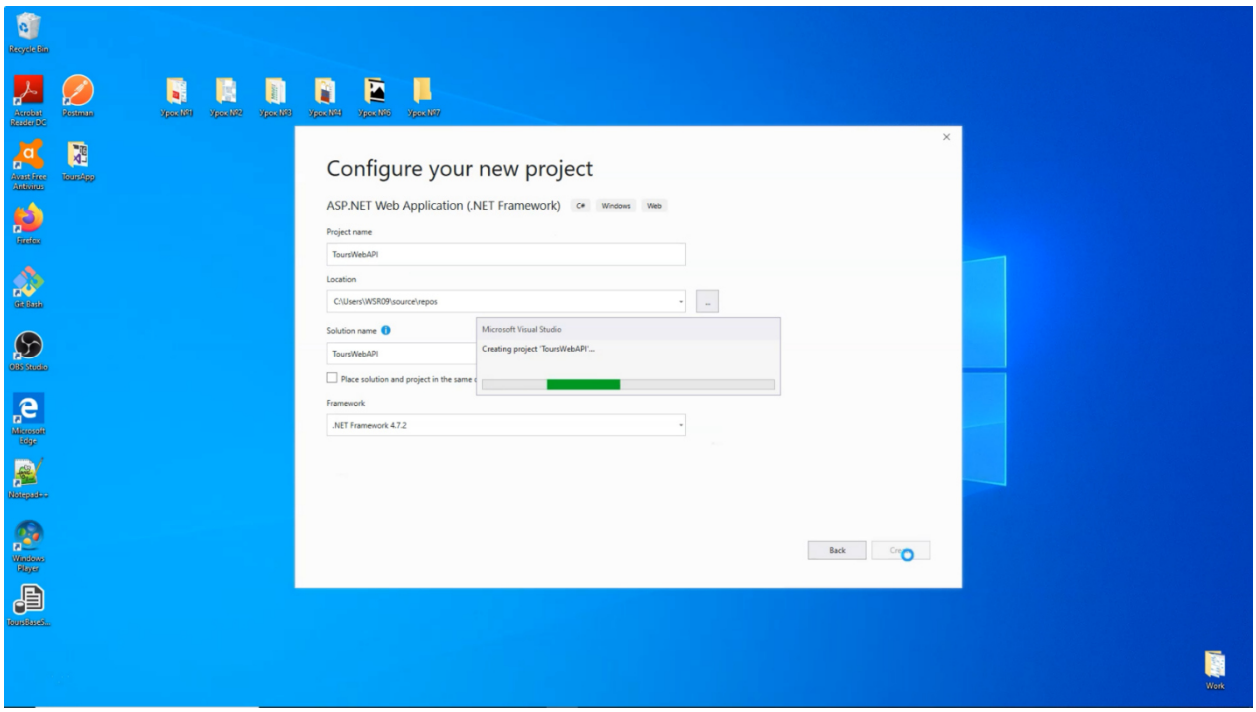


- Выбираем тип Web API. Убираем галочку с конфигурации https



2. На данном этапе нам необходимо реализовать три метода в рамках API:

- для получения списка отелей с изображениями

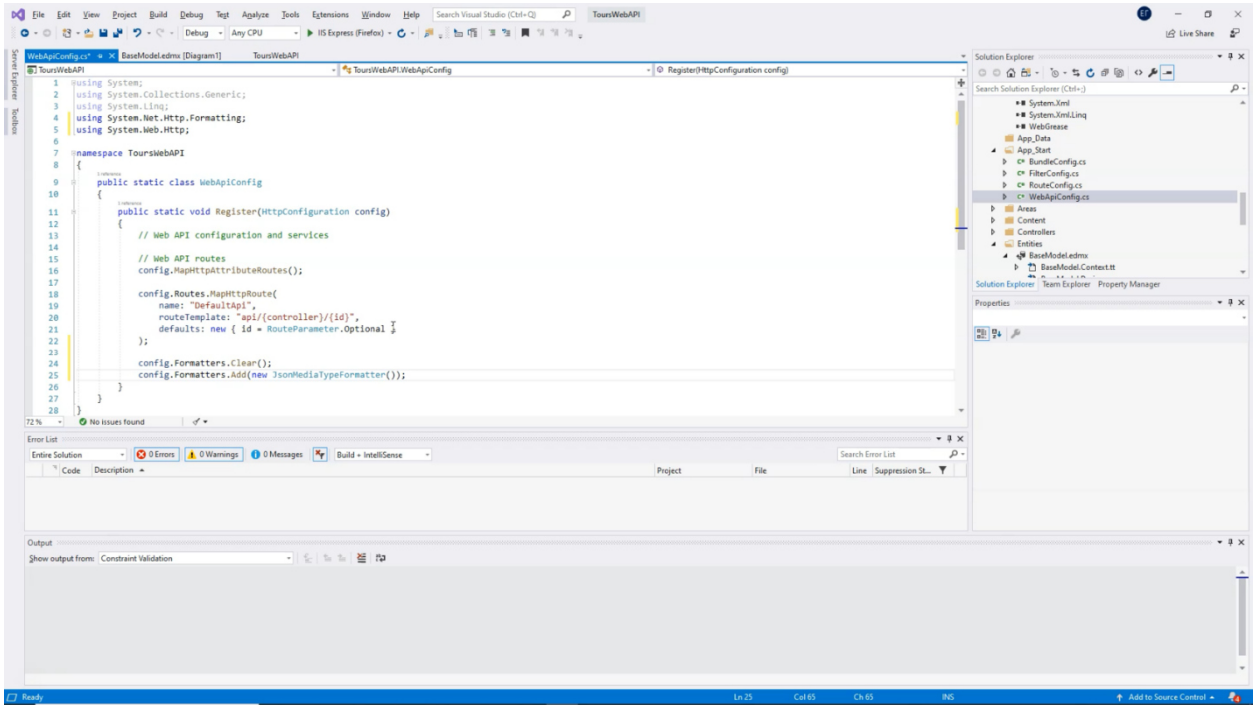


- для получения списка комментариев по конкретному отелю

- для добавления нового отзыва об отеле

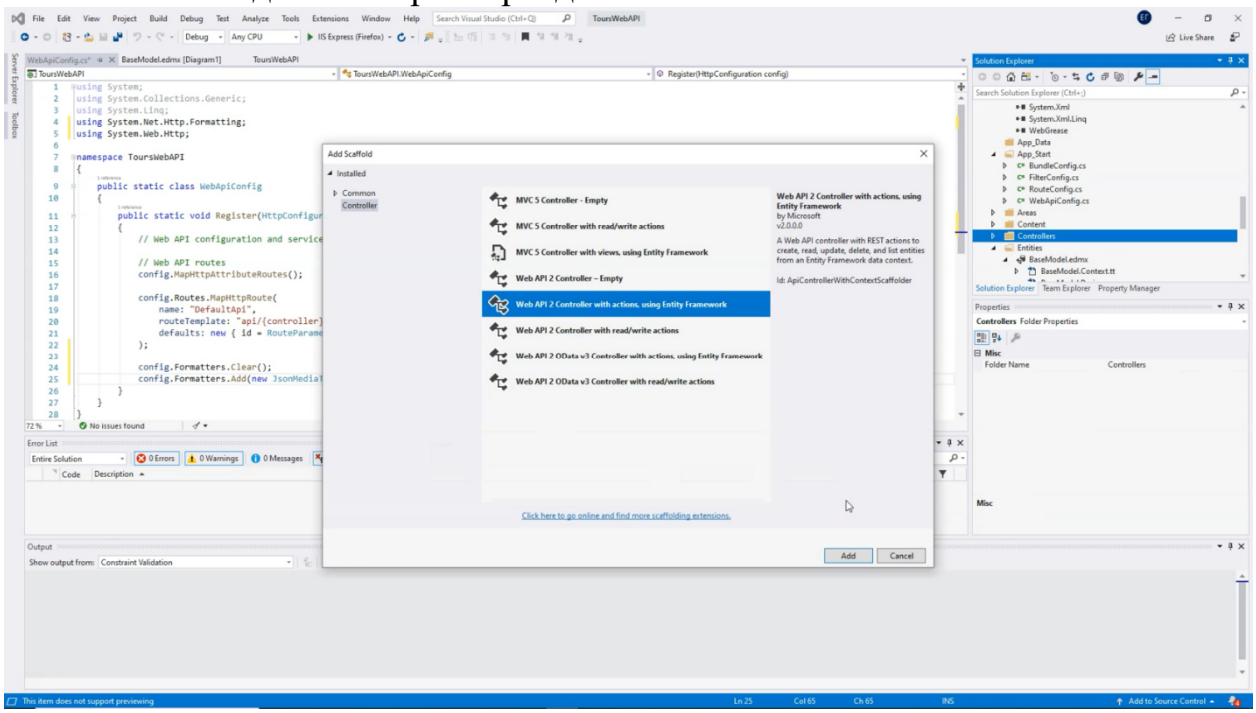
3. В проект добавляем модель Entity Framework. Для работы с данными мы остановимся на формате json, хотя вам может встречаться и xml. За счет своей лаконичности json, по сравнению с xml, представляет собой одну из двух структур в закодированном виде. Это набор из двух пар: ключ — значение и упорядоченный набор значений. Это универсальные структуры данных. Как правило, любой современный язык программирования поддерживает их в той или иной форме

4. Для начала настраиваем возвращаемый формат json в файле WebAPIConfig

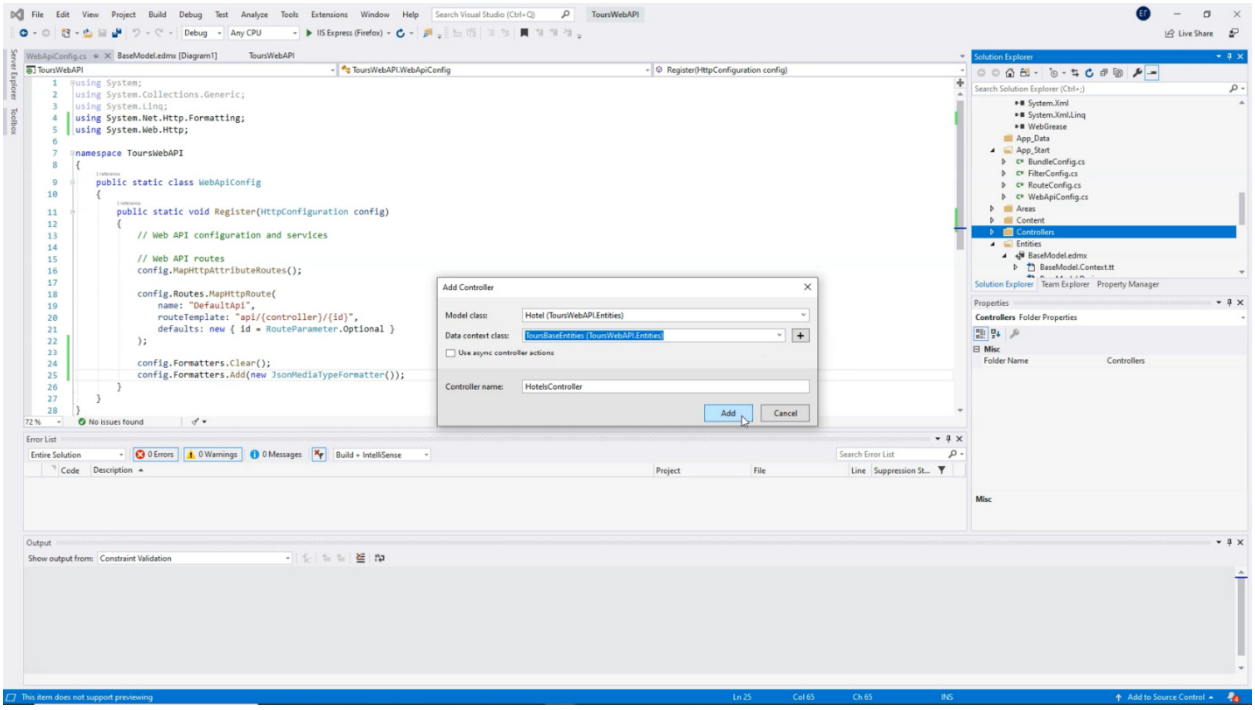


5. На каждую таблицу, с которой мы будем работать создаем отдельный контроллер. Он позволит не перемешивать объекты друг с другом, а писать код только для конкретной таблицы.

- Создаем контроллеры для классов Hotel и HotelComment



- Указываем название модели и контекст приложения



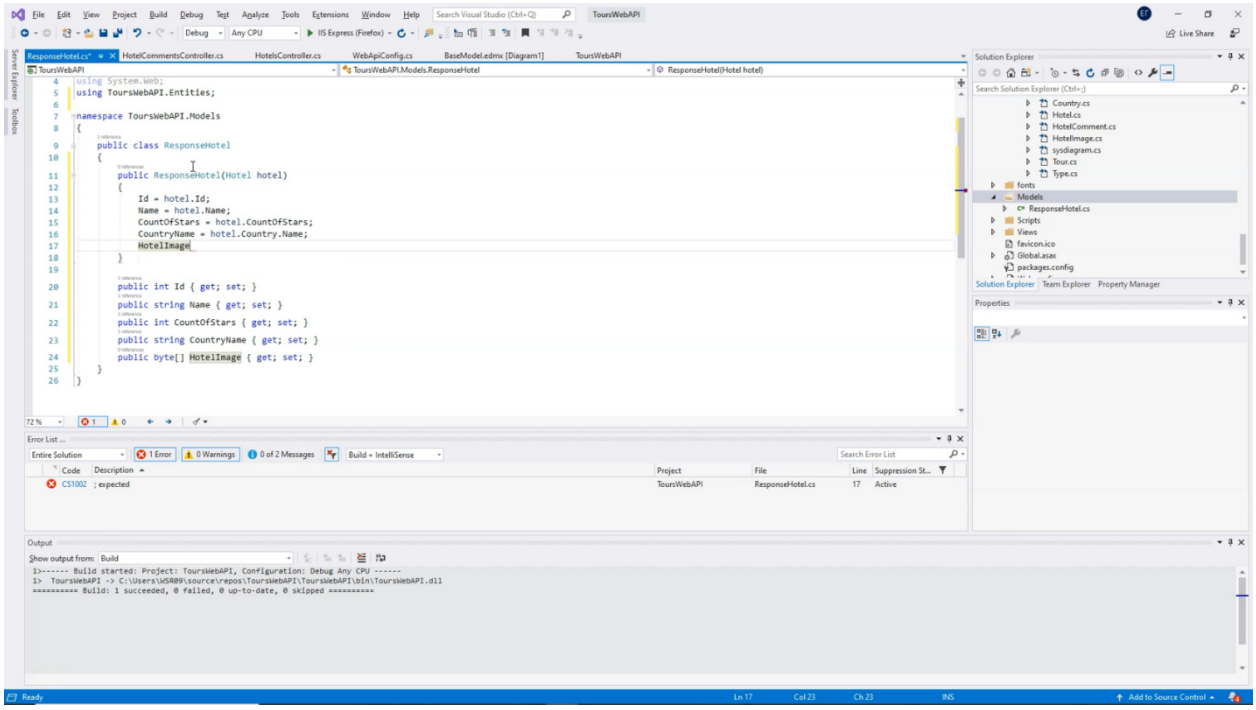
Например, в контроллере отзывов будут находиться все запросы, связанные с получением списка отзывов, с фильтрацией и поиском, добавлением, изменением, удалением и т. д. Для разграничения действий с ресурсами на уровне http методов были придуманы следующие варианты:

1. get — получение ресурса
2. post — создание ресурса
3. put — обновление
4. delete — удаление ресурса

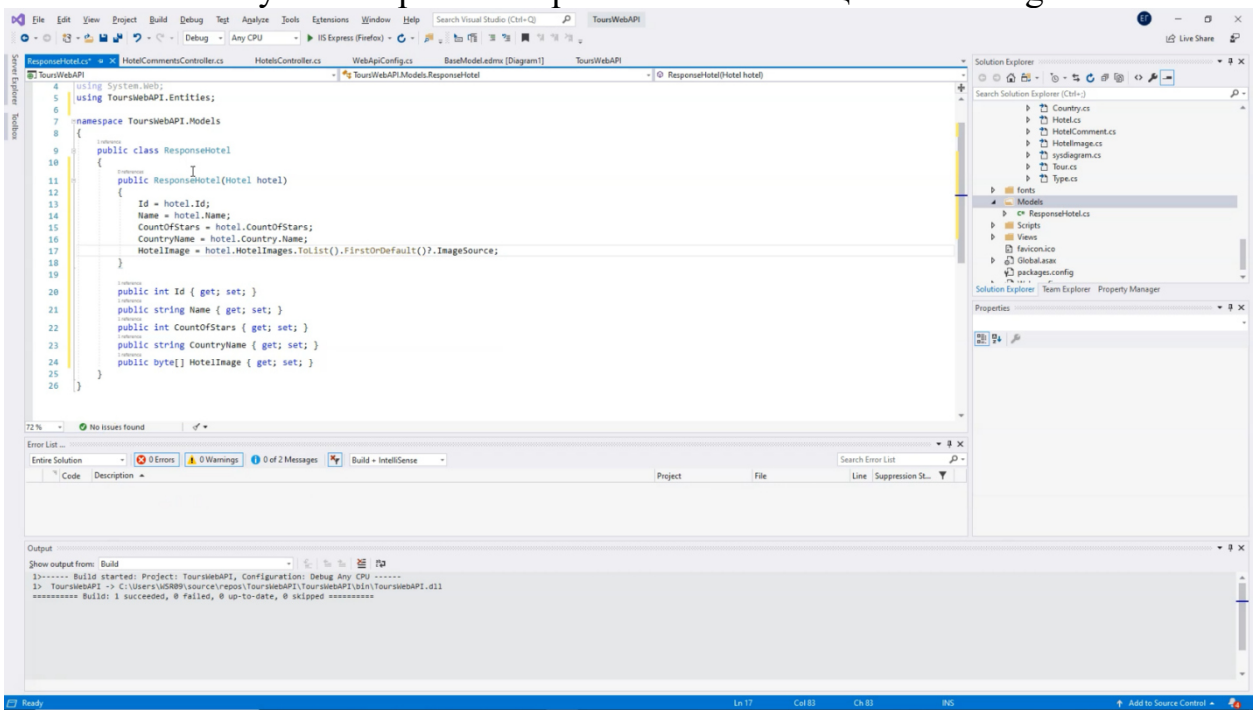
Обратите внимание, спецификация http не обязывает сервер принимать все методы, которых, на самом деле, гораздо больше, чем четыре. Обязателен только get. А также не указывает серверу, что он должен делать при получении запроса с тем или иным методом

6. Создаем Response Model для отелей, чтобы возвращать еще и фотографию

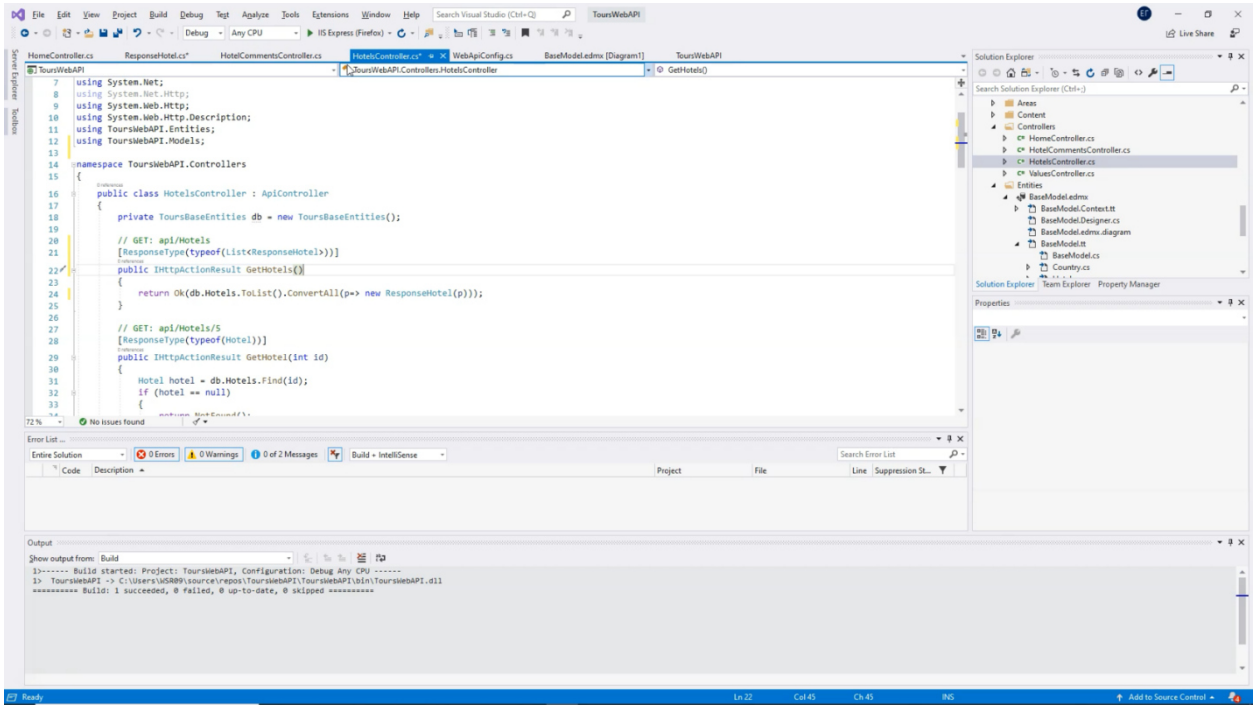
- Сначала перечисляем ряд свойств, которые относятся к этой модели, затем добавляем конструктор



- Получаем первое изображение из таблицы HotelImage

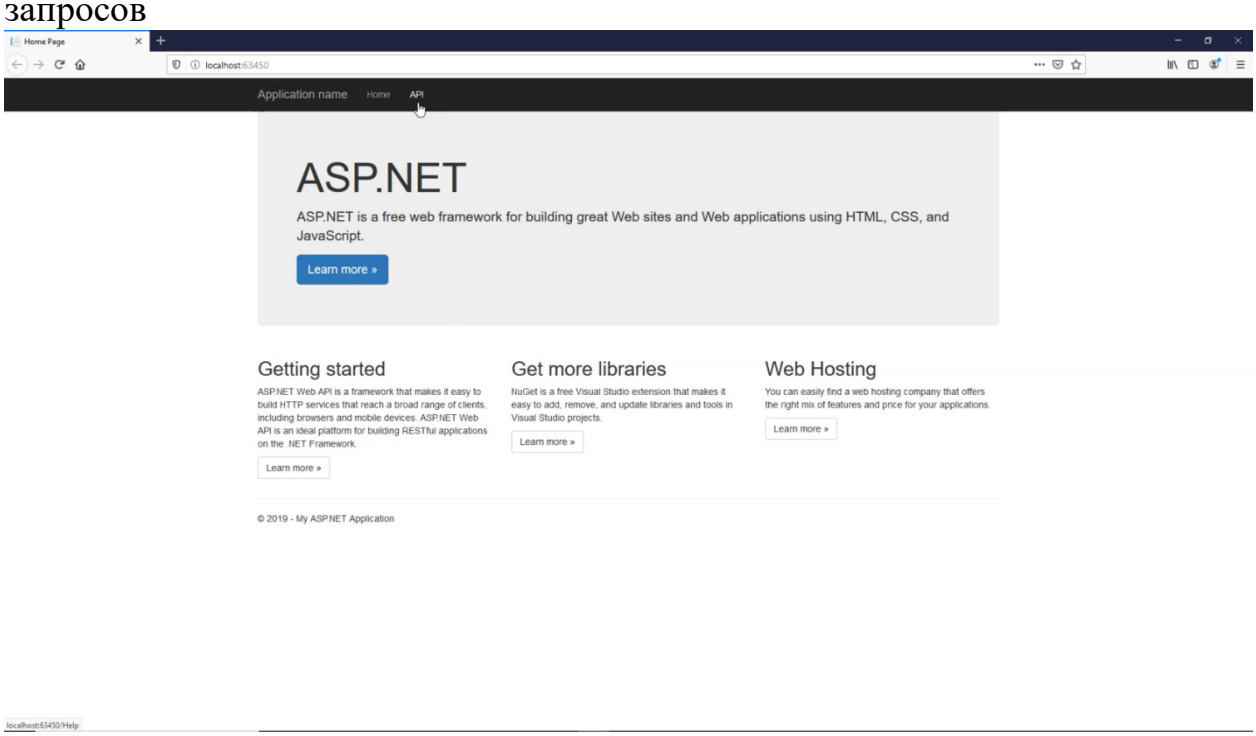


- В контроллере отелей первый метод возвращает все отели. Поработаем над методом GetHotels, а остальные оставляем по умолчанию. В нем мы преобразуем список полученных отелей к типу ResponseHotel. Также мы установим атрибут ResponseType — это тип ответа, который необходим для документации в разделе Информация об ответе

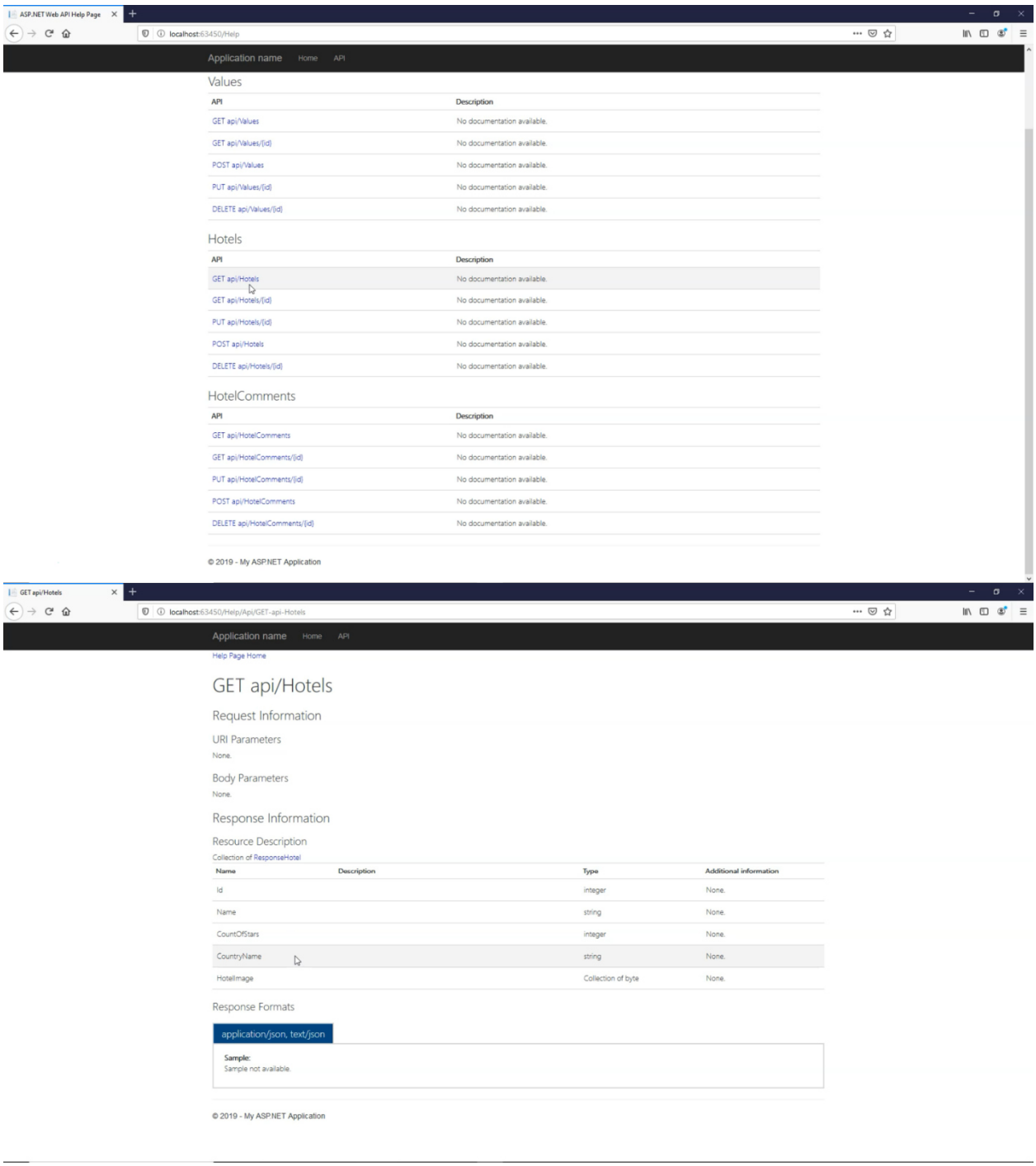


7. Запускаем и тестируем это в браузере

- У нас есть раздел API, который описывает документацию наших запросов



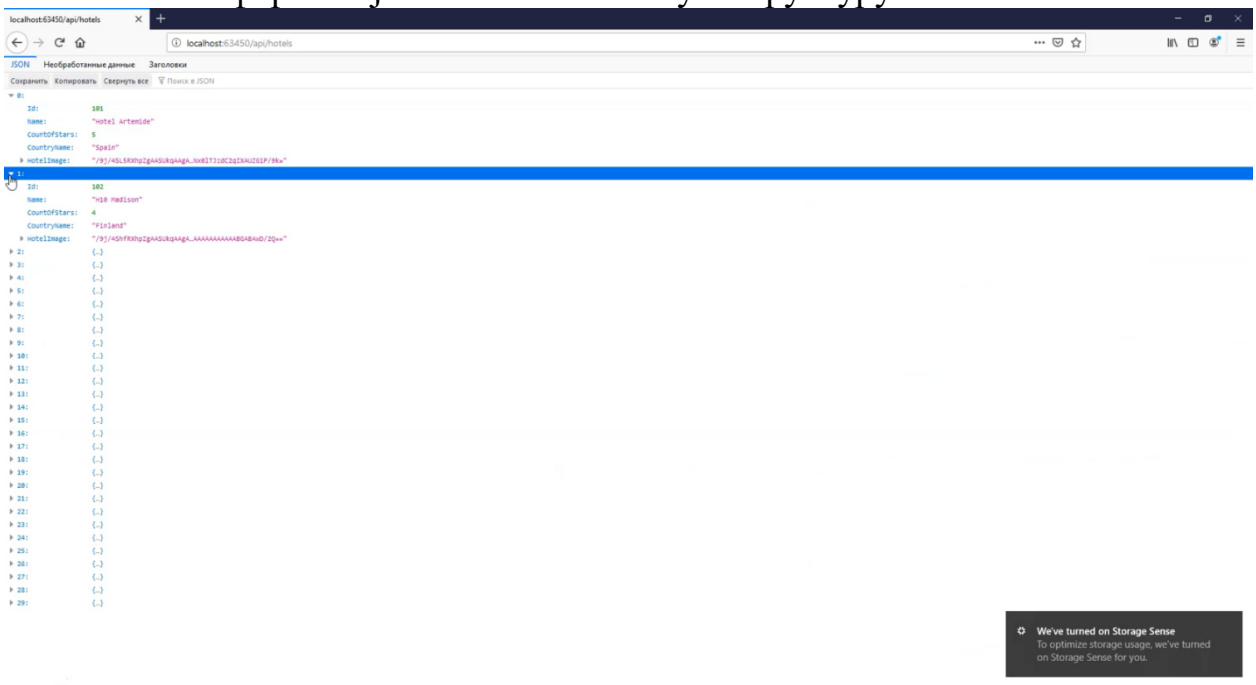
- Здесь у нас есть созданные контроллеры, в том числе Hotels. На методе Hotels типа get у нас есть Response Information, который мы, как раз, указали в качестве атрибута, и список его свойств



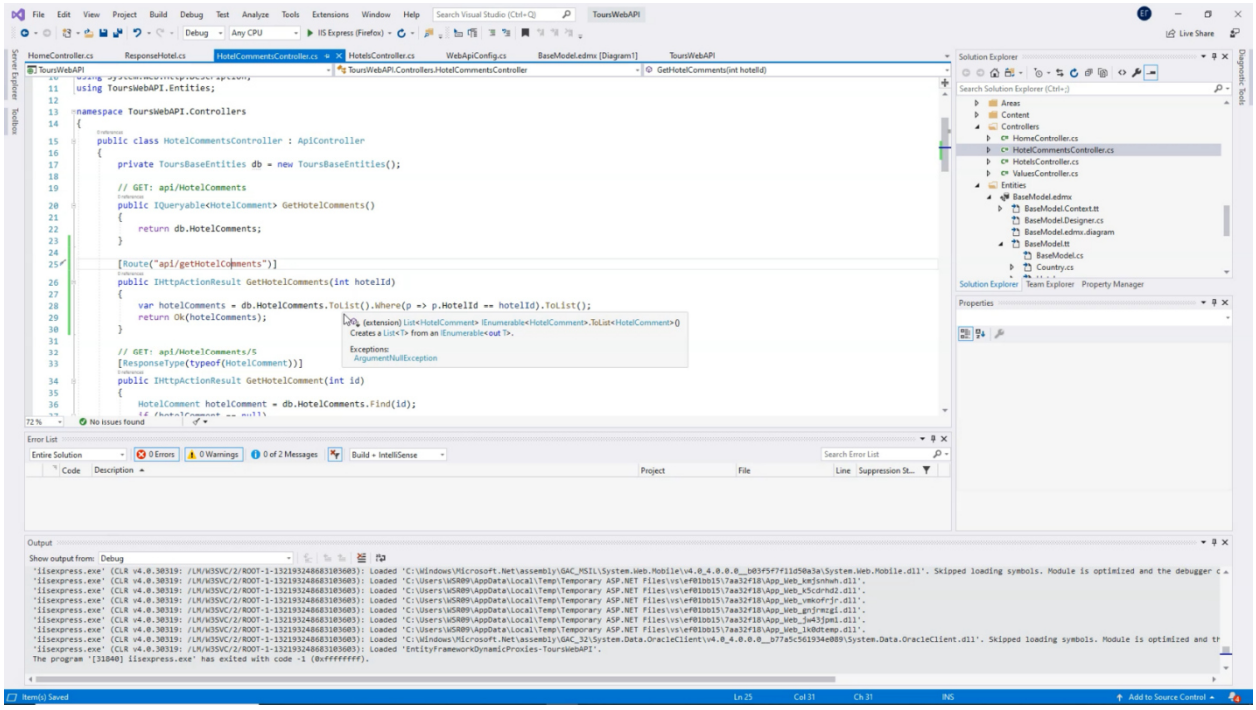
8. Попробуем вызвать запрос на получение списка отелей.

- В необработанном виде они выглядят так

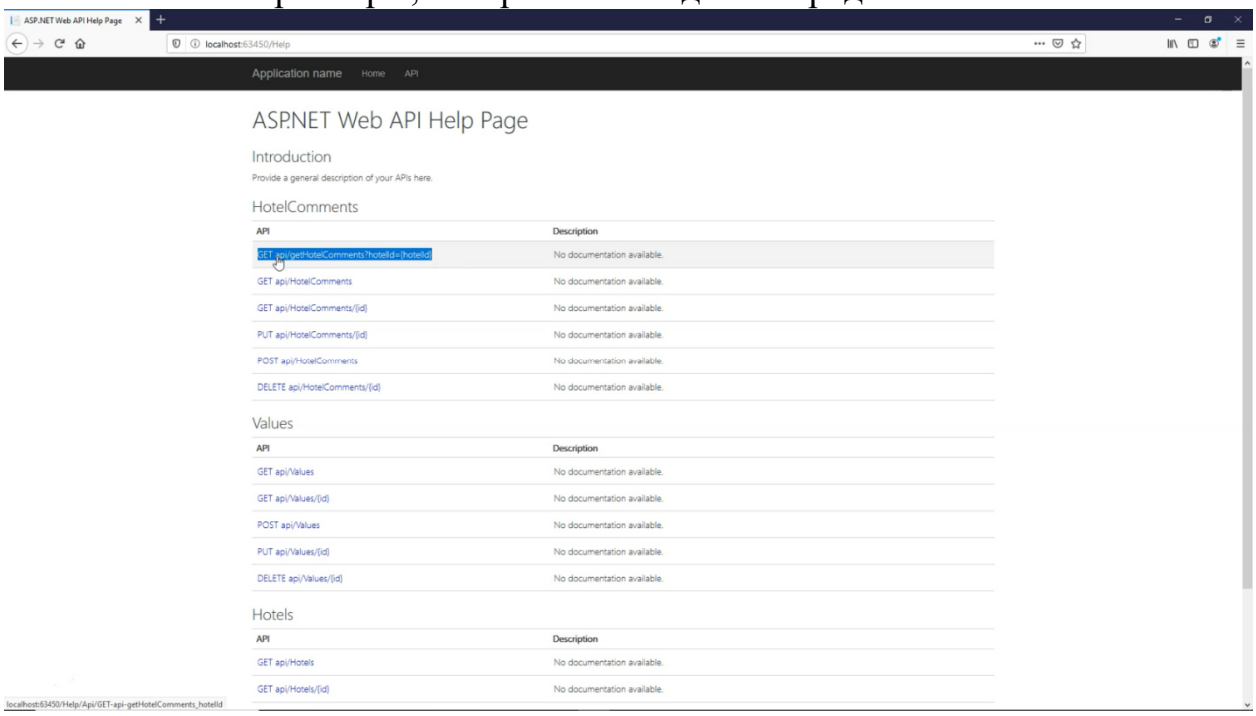
- В формате json они имеют такую структуру

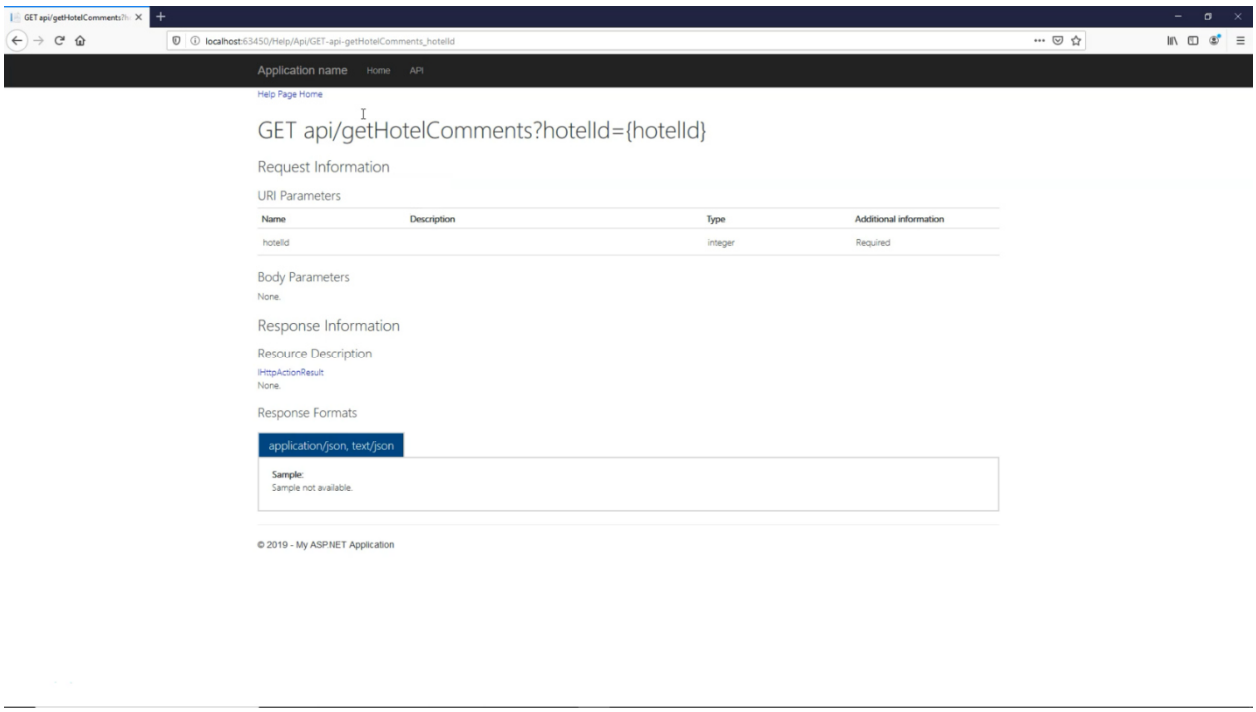


9. Переходим к комментариям. Создавать Response Model в этом случае мы не будем. Подойдет класс от Entity Framework. Добавляем новый метод с параметром и дадим ему название через атрибут Wrote. Если комментариев к отелю не будет, вернется пустой json

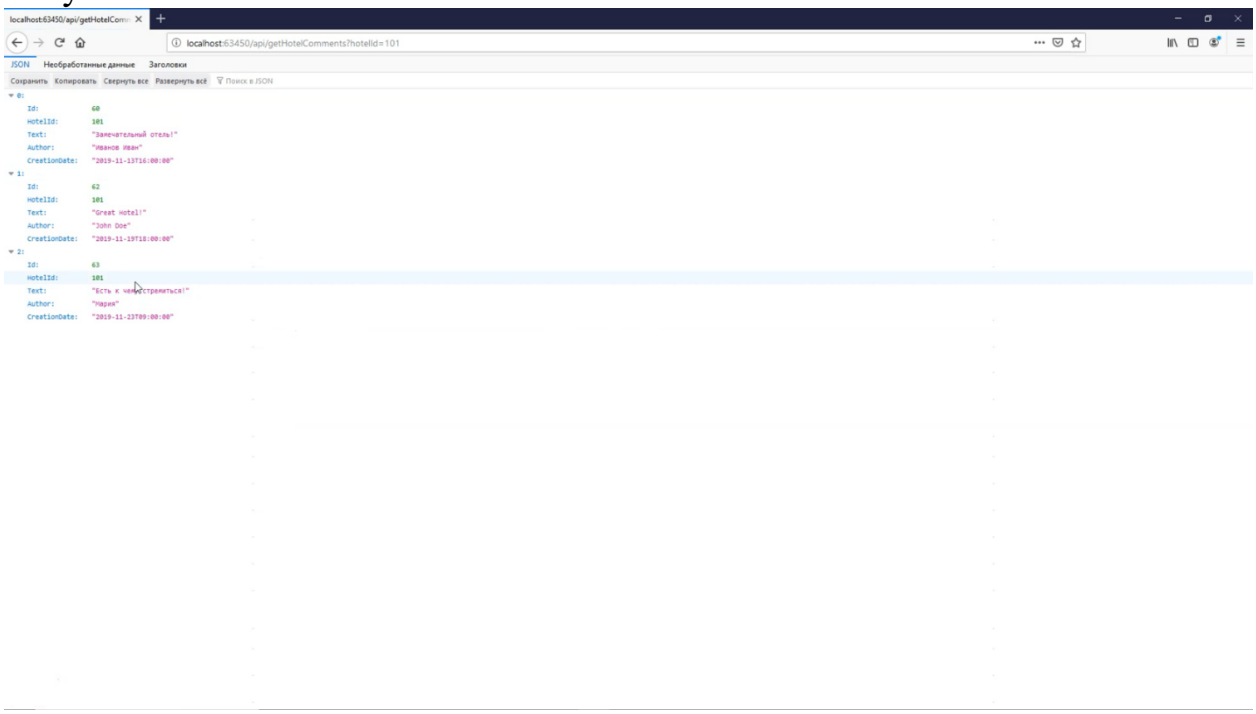


10. Запустим и проверим. При открытии документации в контроле HotelComments, у нас появится новый метод getHotelComments. Он описывает параметры, которые необходимо передать



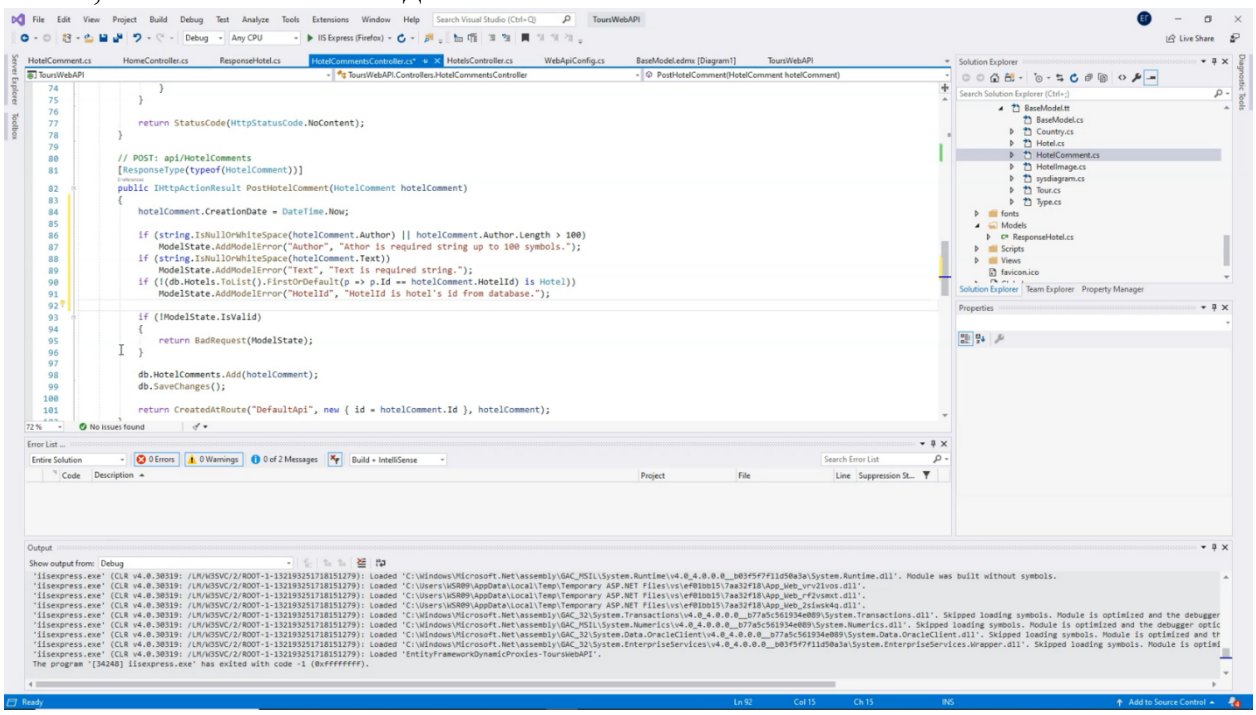


11. Попробуем его вызвать: возьмем код отеля из базы — 101. Данные получены

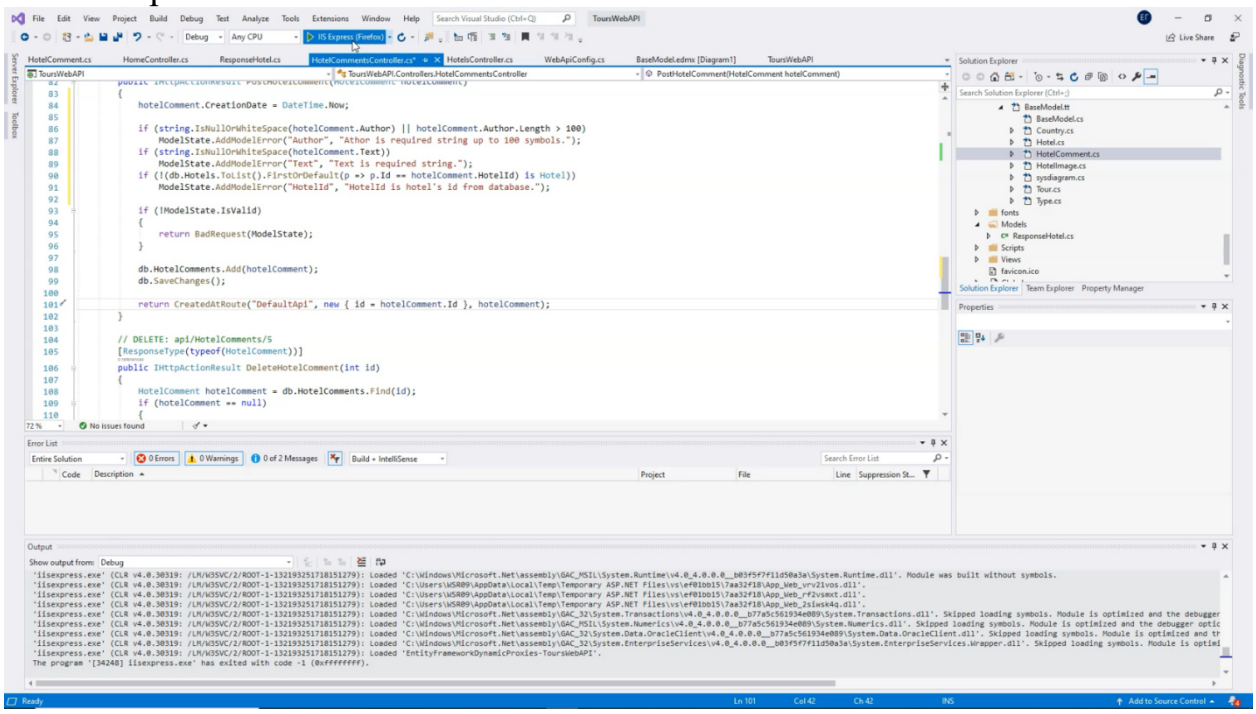


12. Переходим к добавлению данных в контроллере. За это будет отвечать метод POST. Время добавления нового отзыва будем вычислять на стороне сервера, поэтому тот объект, который передают в качестве параметров, мы изменяем. Также мы делаем все проверки полученных данных с помощью свойства ModelState. Обращаем внимание на то, что пользователи могут вводить пустые данные при указании автора и текста. Также у нас есть ограничения в базе данных на размер символов для автора, при этом пользователи должны указывать реально существующее название

отеля, в том числе его код

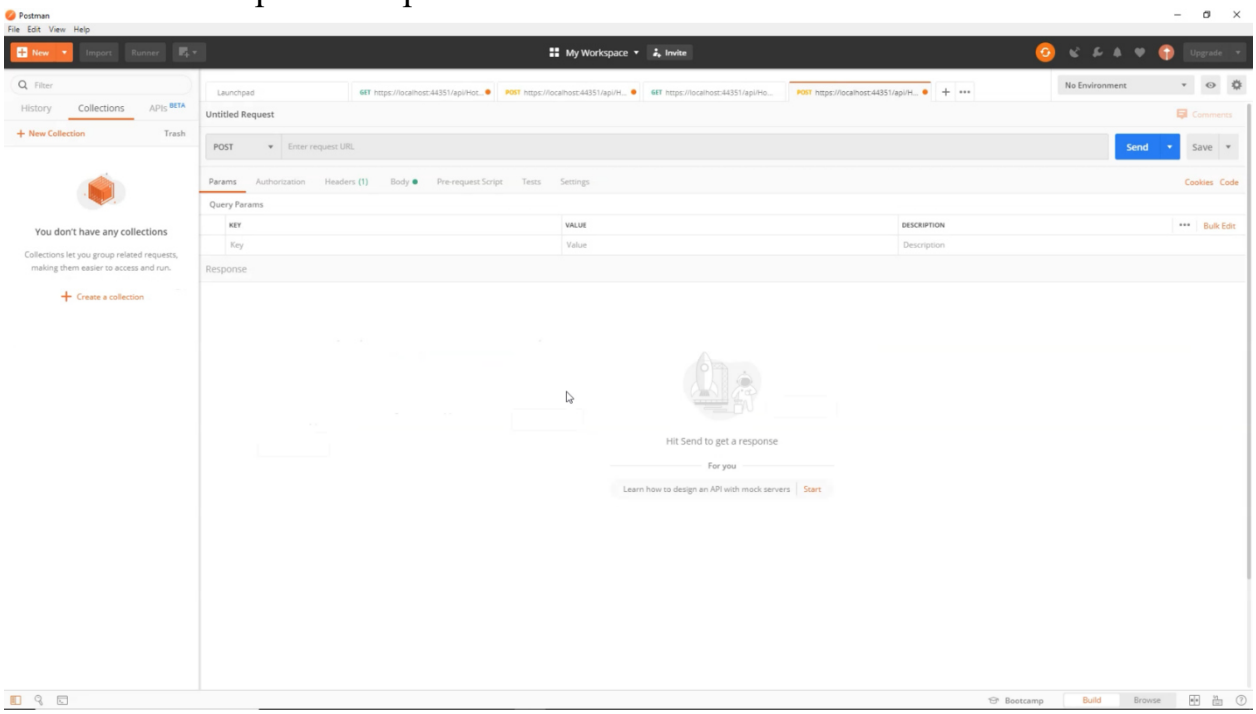


13. В случае, если сущность не пройдет валидацию, мы выдаем Bad Request с сообщениями об ошибках в этой модели. Иначе сохраняем переданную сущность и сохраняем изменения, вернув новый созданный экземпляр

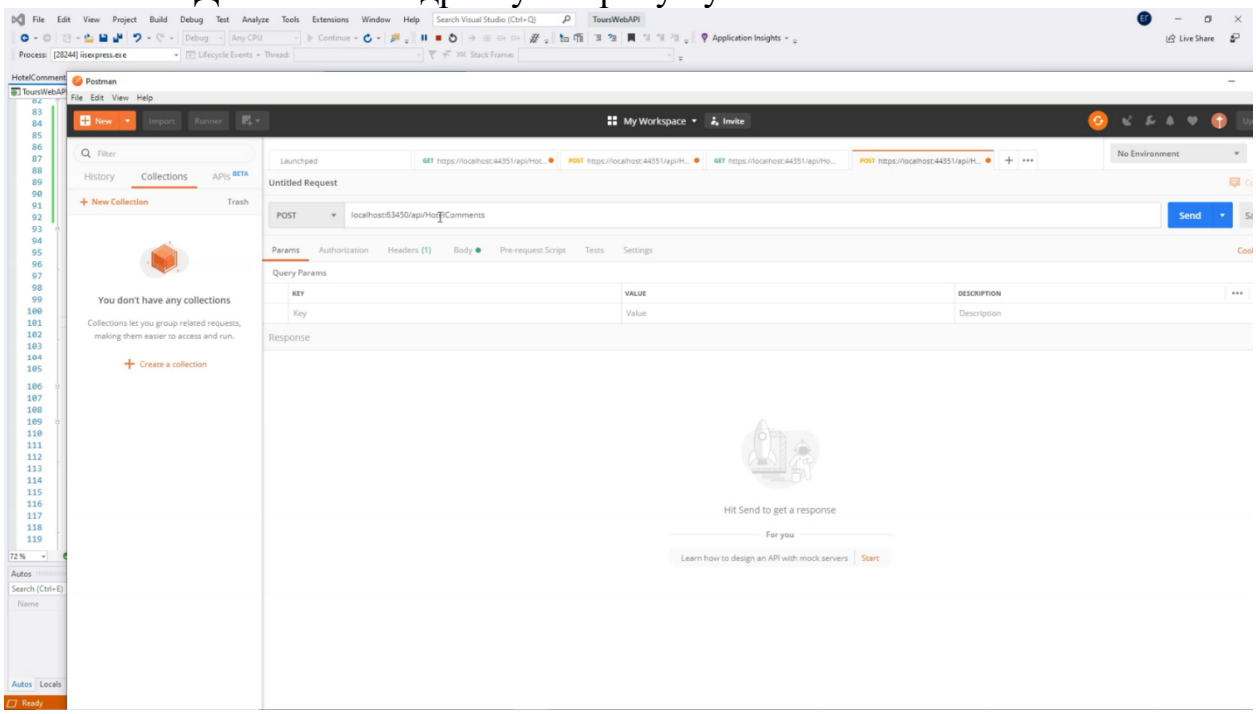


14. Как правило, через браузер можно получить данные только по методу get. Однако, есть приложения или расширения, имитирующие и другие методы. Как, например, приложение Postman, позволяющее отправлять запросы различными методами и получать ответ. Также в качестве альтернативы можно использовать консоль Package Manager в Visual Studio.

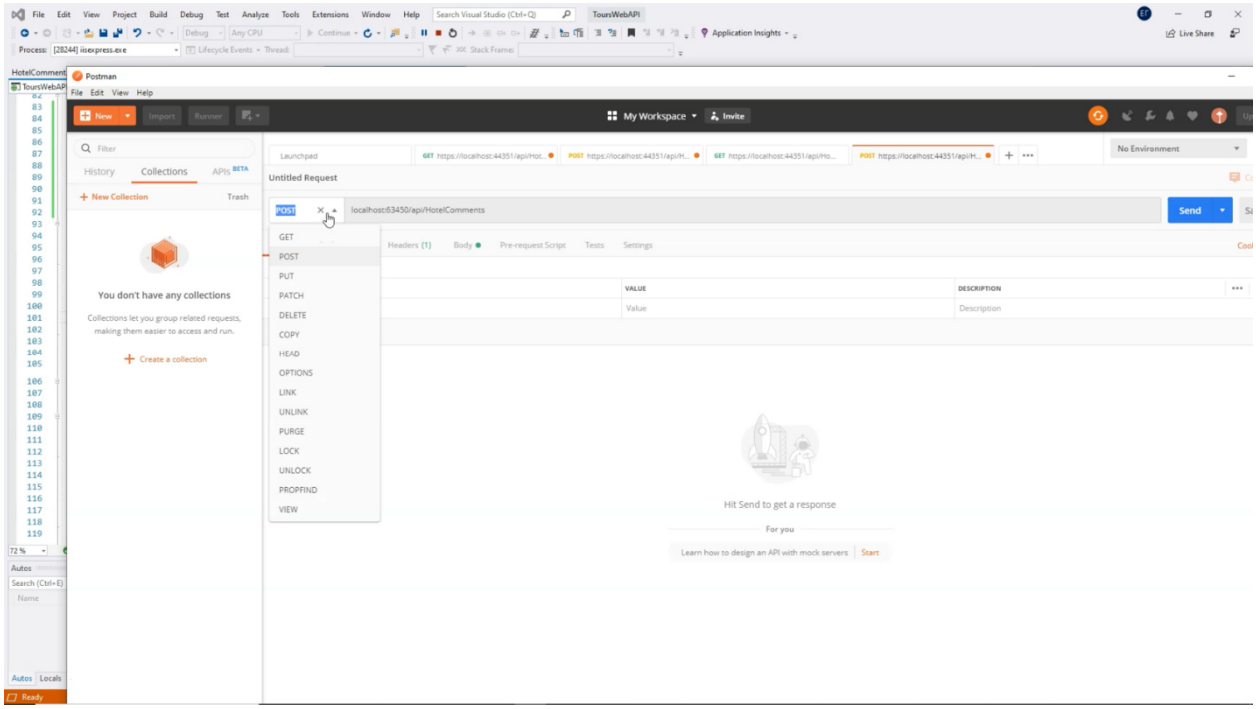
- Открываем приложение Postman



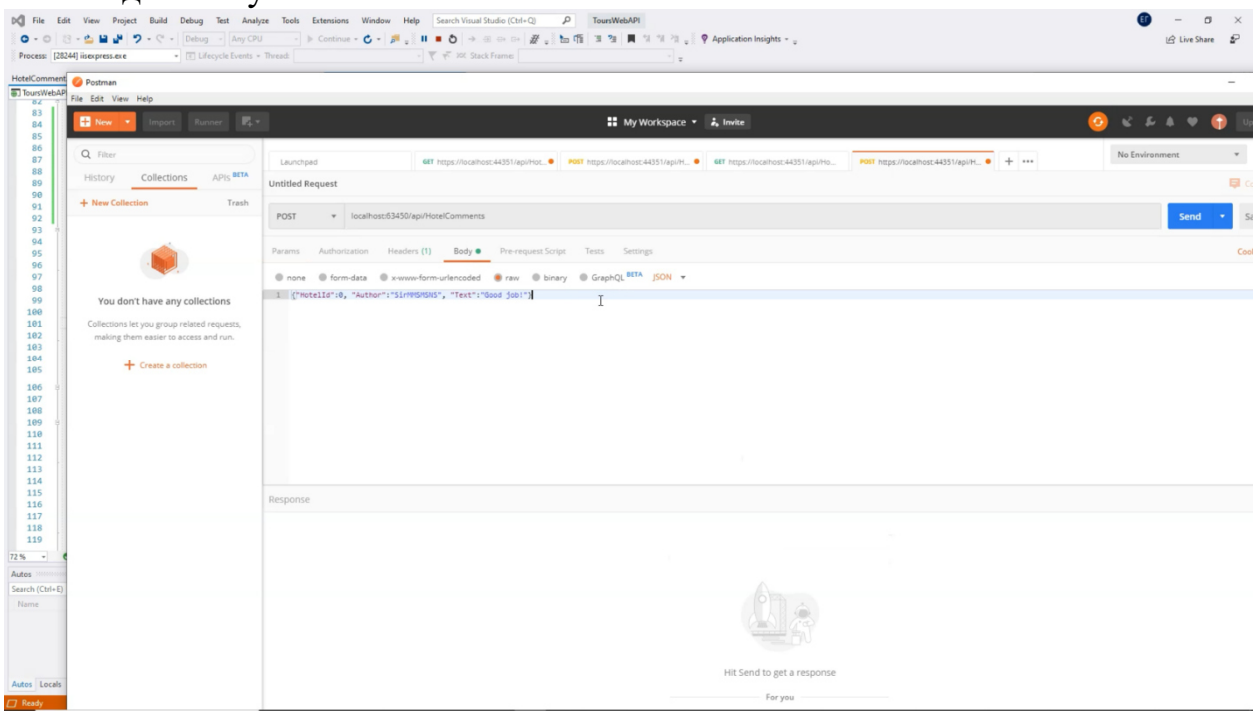
- Добавляем в адресную строку путь



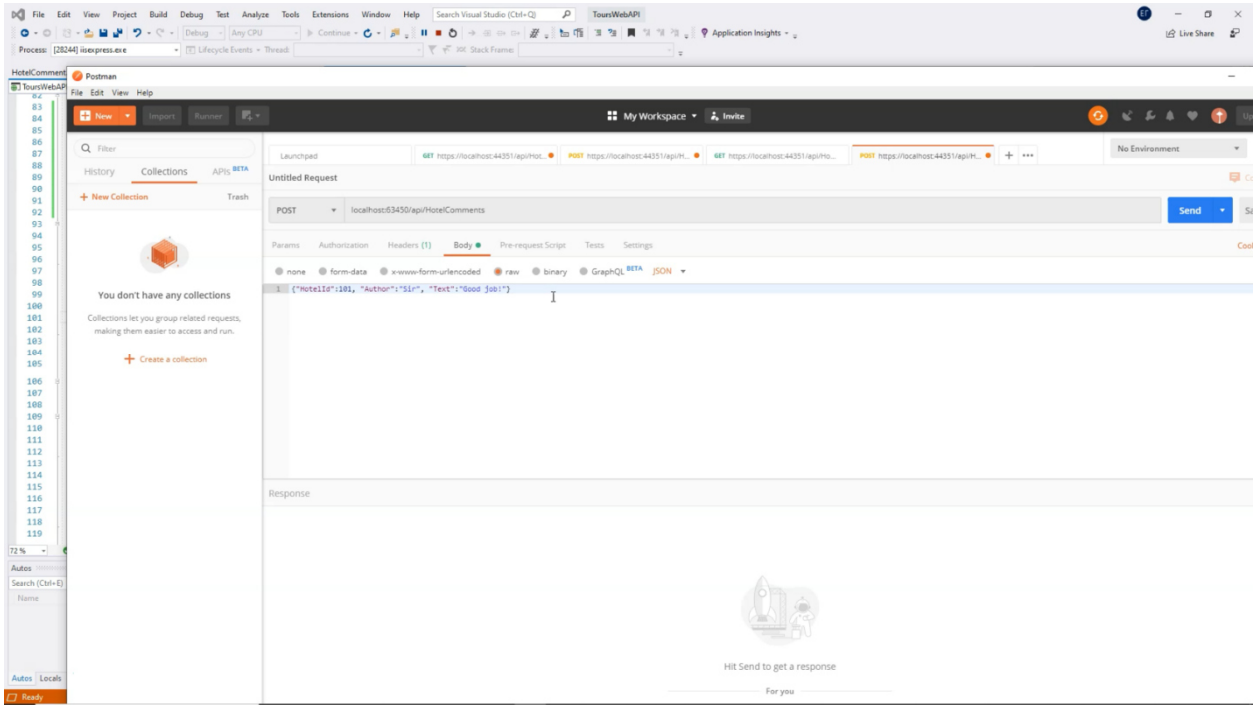
- Выбираем метод POST



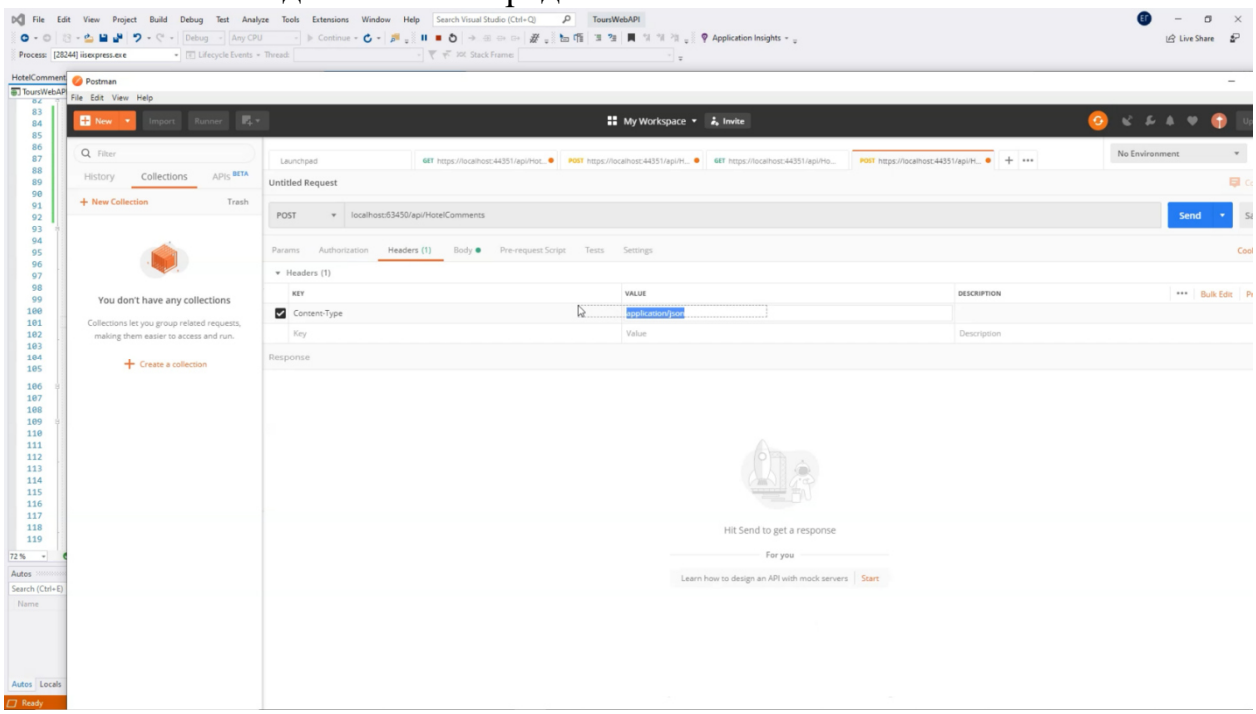
- Сущность должна быть отправлена в формате json в теле запроса на вкладке Body



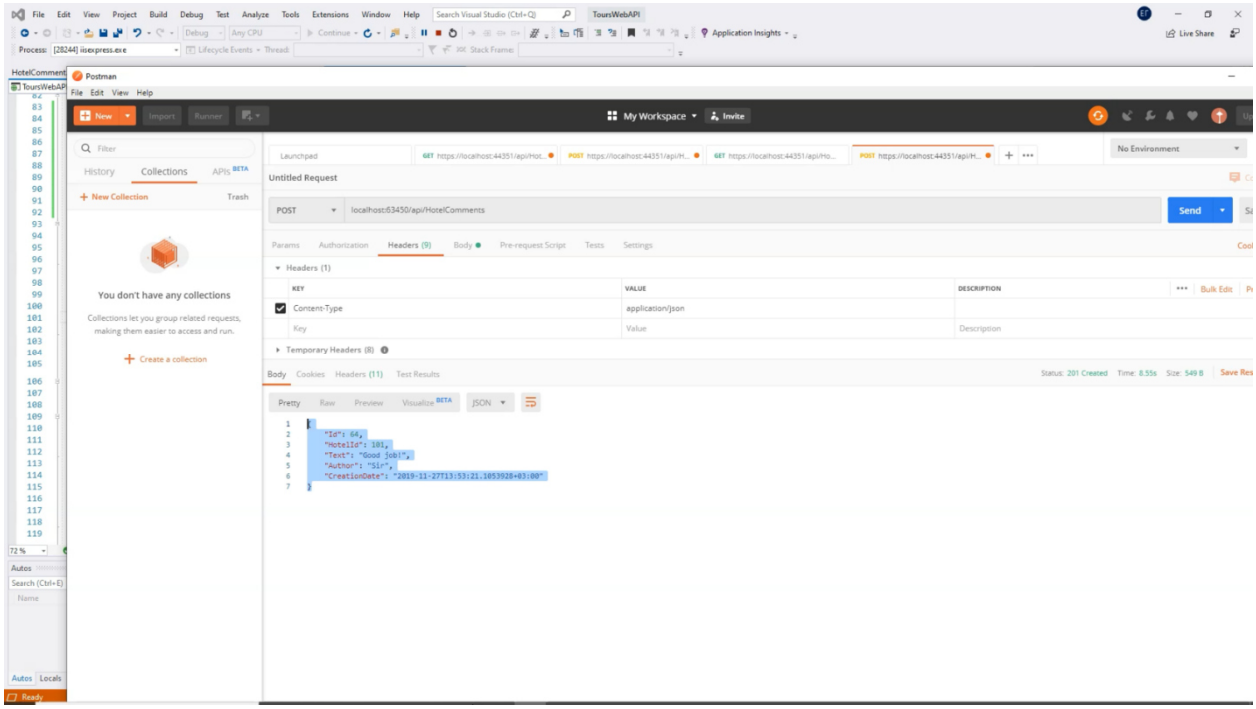
- Указываем код отеля, автора и комментарий



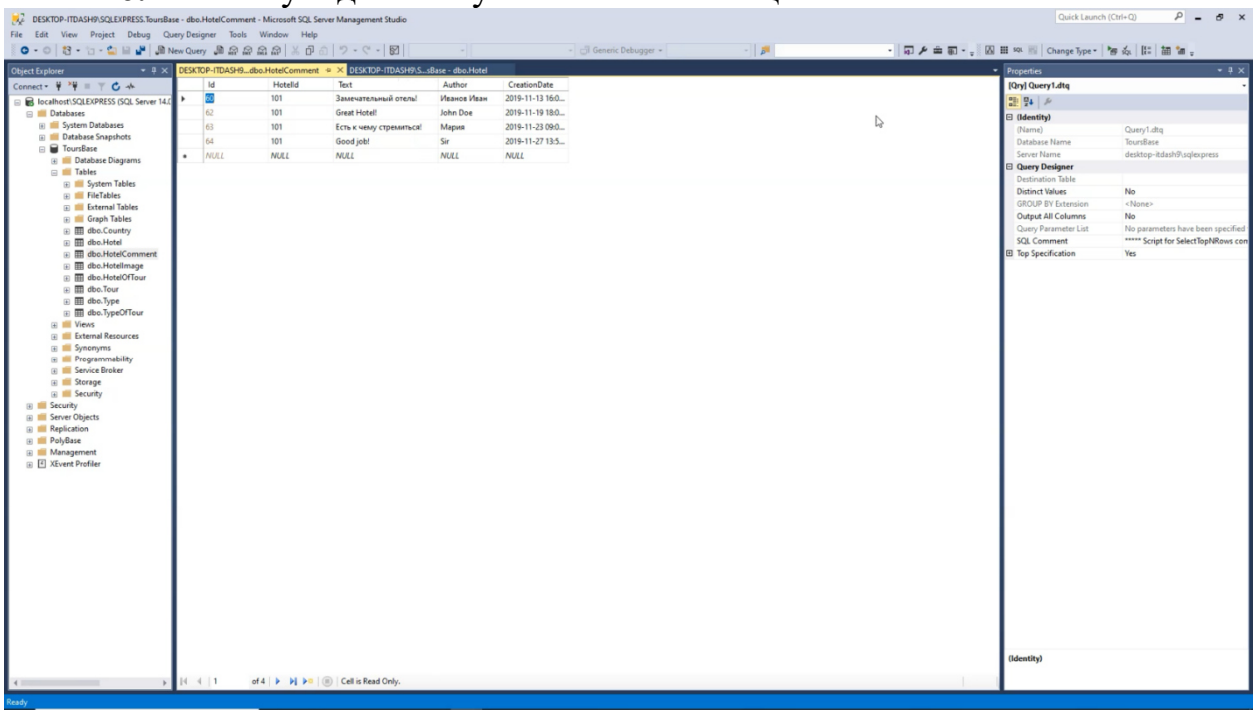
- Не забудьте указать в хэдере Content-Type — application/json. Именно этот тип данных мы передаем



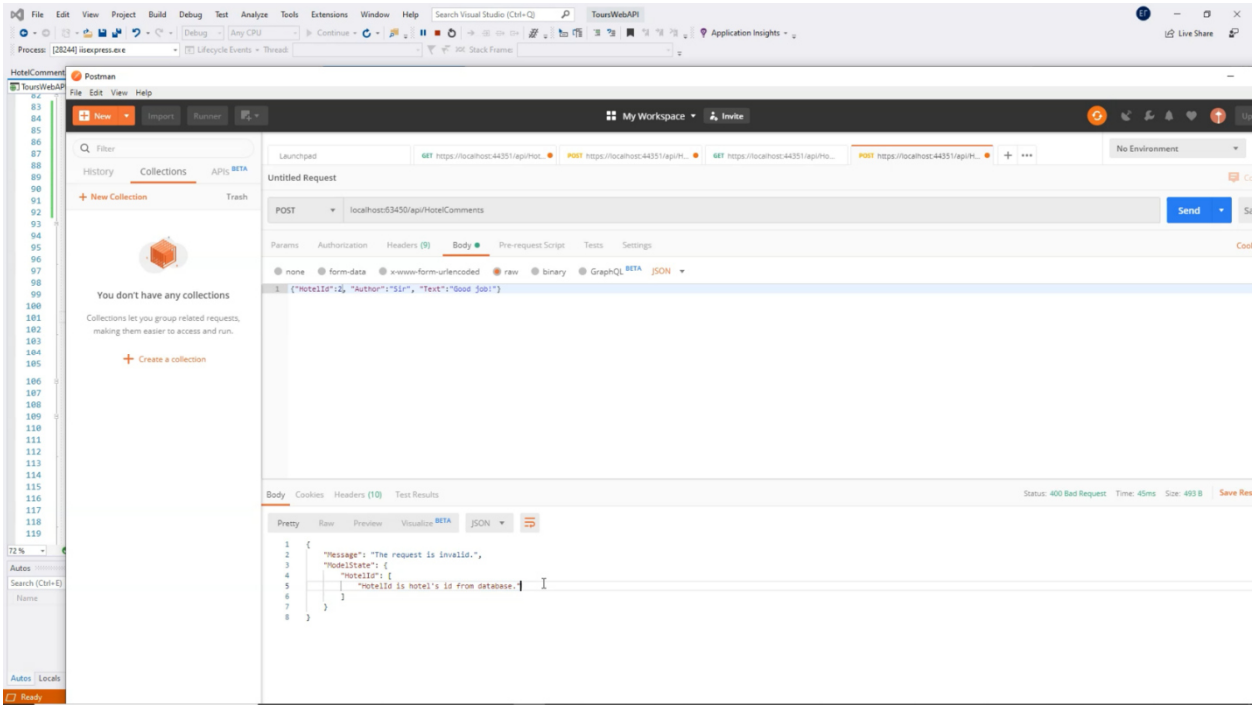
15. Попробуем отправить запрос. Получаем ответ с созданной сущностью



16. Можно увидеть новую запись в таблице HotelComment



17. Кроме того проверяем на наличие ошибок данные, которые мы передали. Если мы указываем несуществующий код отеля, например, 2, тогда то, что мы обработали в API, будет выведено в качестве результата значения ModelState



Интерактивное задание
https://nationalteam.worldskills.ru/skills/razrabotka-api/

Практическая работа №15. Программная работа с файловой системой с помощью пространства имен System.IO

Краткие теоретические сведения

На данном занятии будет разработано простое консольное приложение для структурирования файлов с произвольного носителя. Итогом работы приложения будет служить набор папок, содержащих файлы, отсортированные в соответствие с критериями сортировки, а также файл Info.txt с информацией о количестве отсортированных документов разных типов. Основные шаги построения приложения:

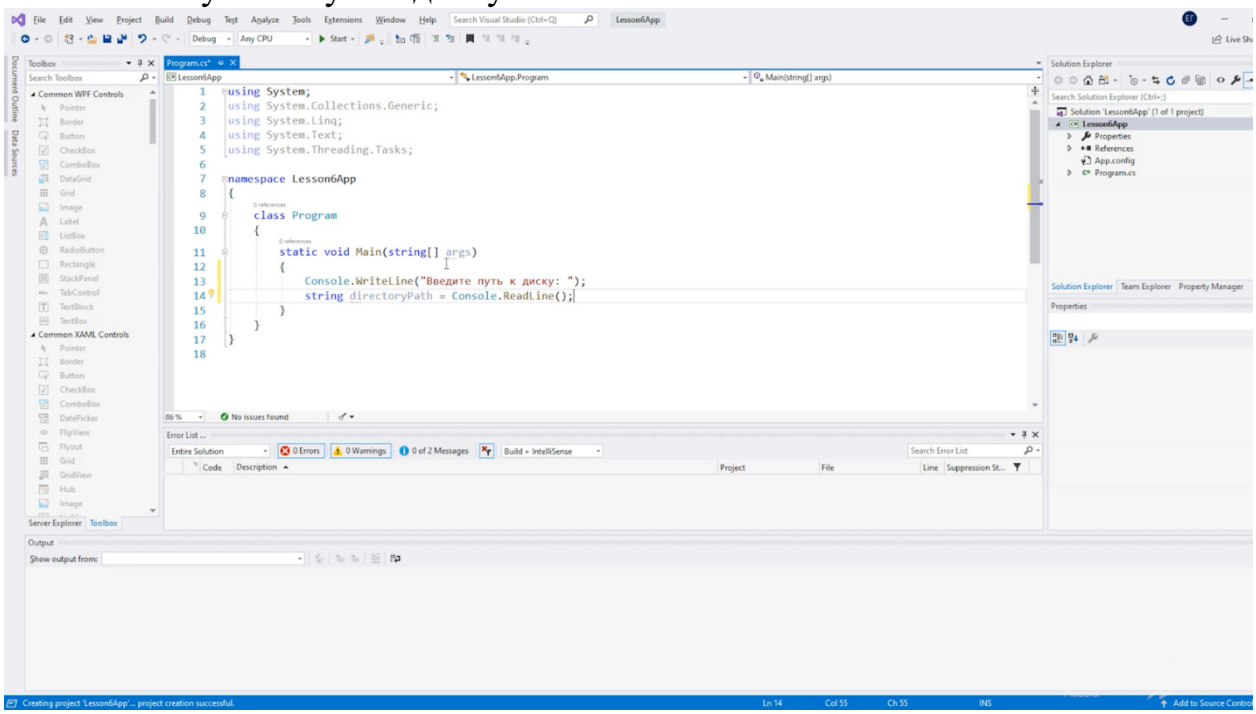
1. Начало работы в Main()
2. Реализация перебора папок
3. Реализация первичной сортировки файлов
4. Реализация перемещения файлов
5. Уточнение критериев сортировки файлов
6. Завершение работы с Main()

Важно

Для работы с каталогами в пространстве имен System. IO предназначены два класса со схожими возможностями: класс Directory и DirectoryInfo. На данном занятии преимущественно используется DirectoryInfo

Начало работы в Main()

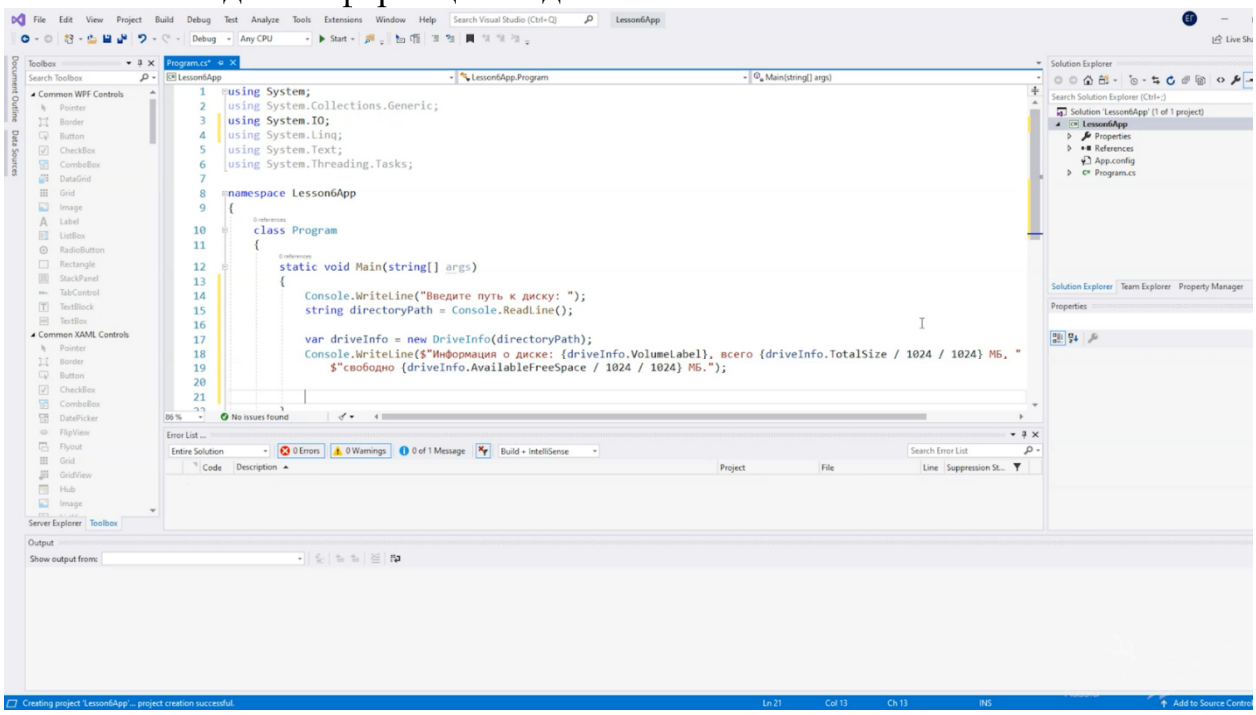
1. Получаем путь к диску



Важно

Путь запрашивается у пользователя и записывается в переменную directoryPath

2. Выводим информацию о диске

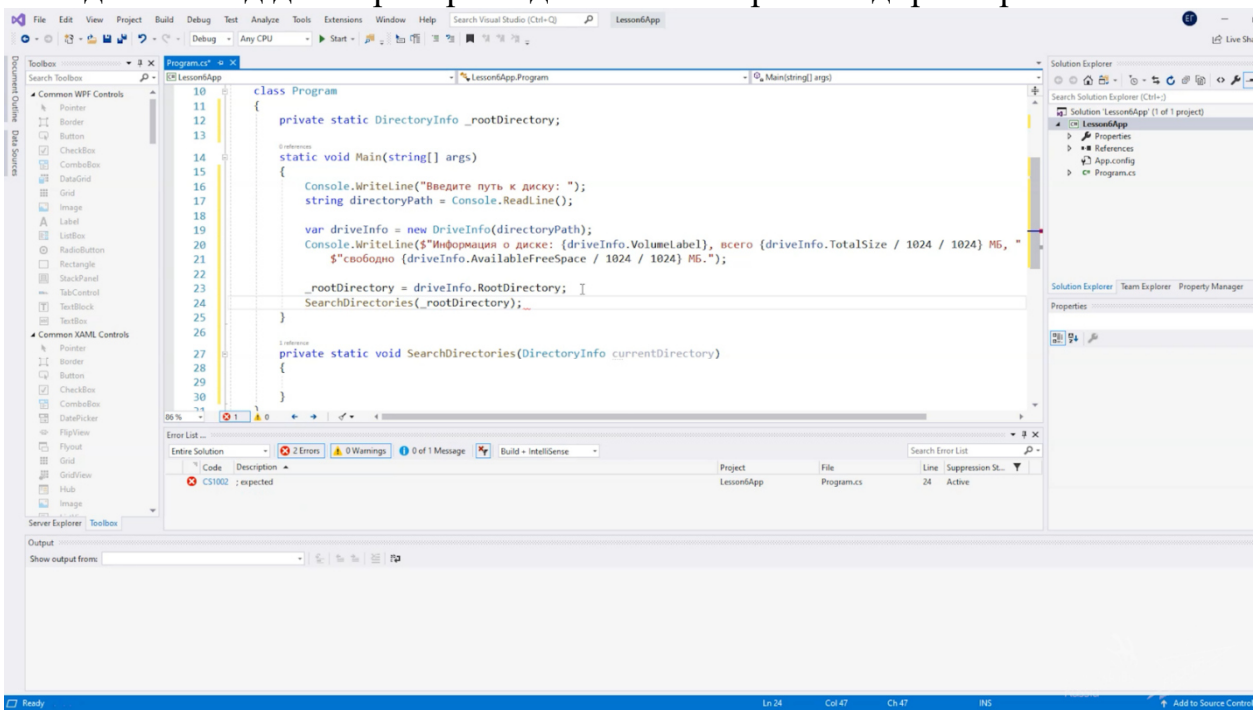


Важно

Информация о диске (название, общий объем и свободное место в мегабайтах) выводятся с помощью свойств класса DriveInfo

Реализация метода перебора папок

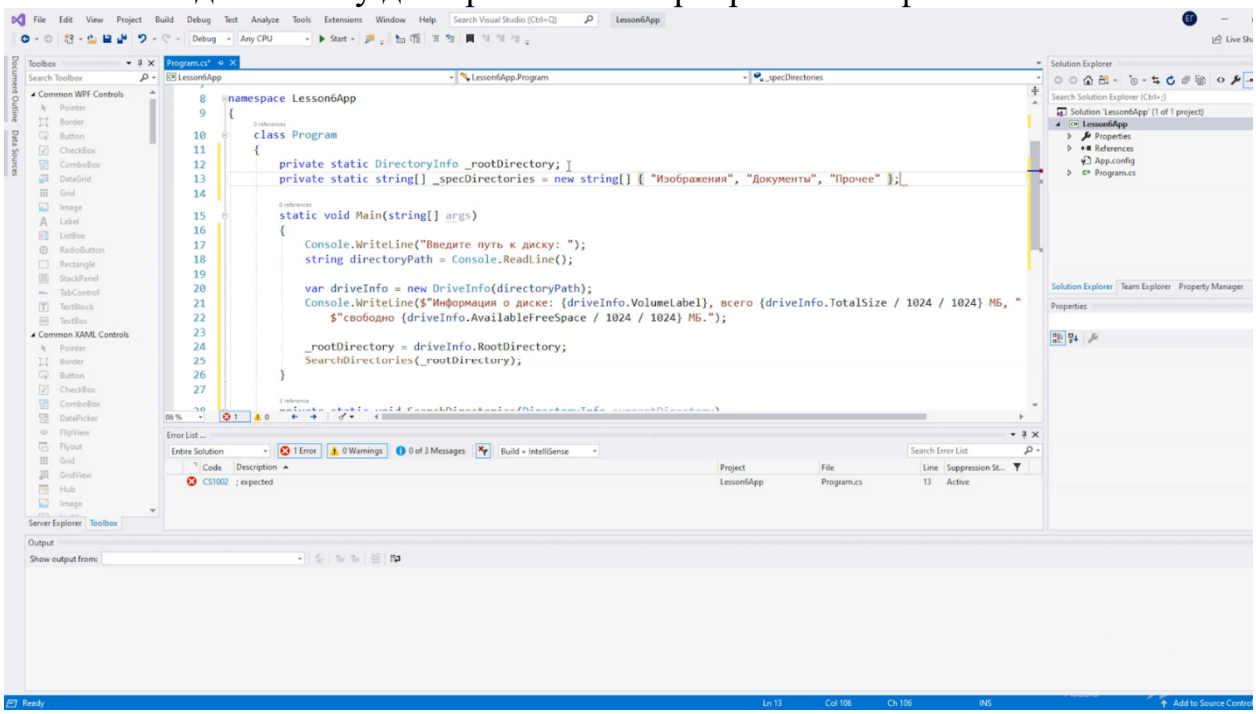
1. Готовим переменную для хранения пути к корневой директории и создаем метод для перебора подкаталогов корневой директории



Важно

Путь к корневой директории записывается в переменную _rootDirectory. Метод SearchDirectories() будет осуществлять поиск в _rootDirectory

2. Создаем папку для хранения отсортированных файлов

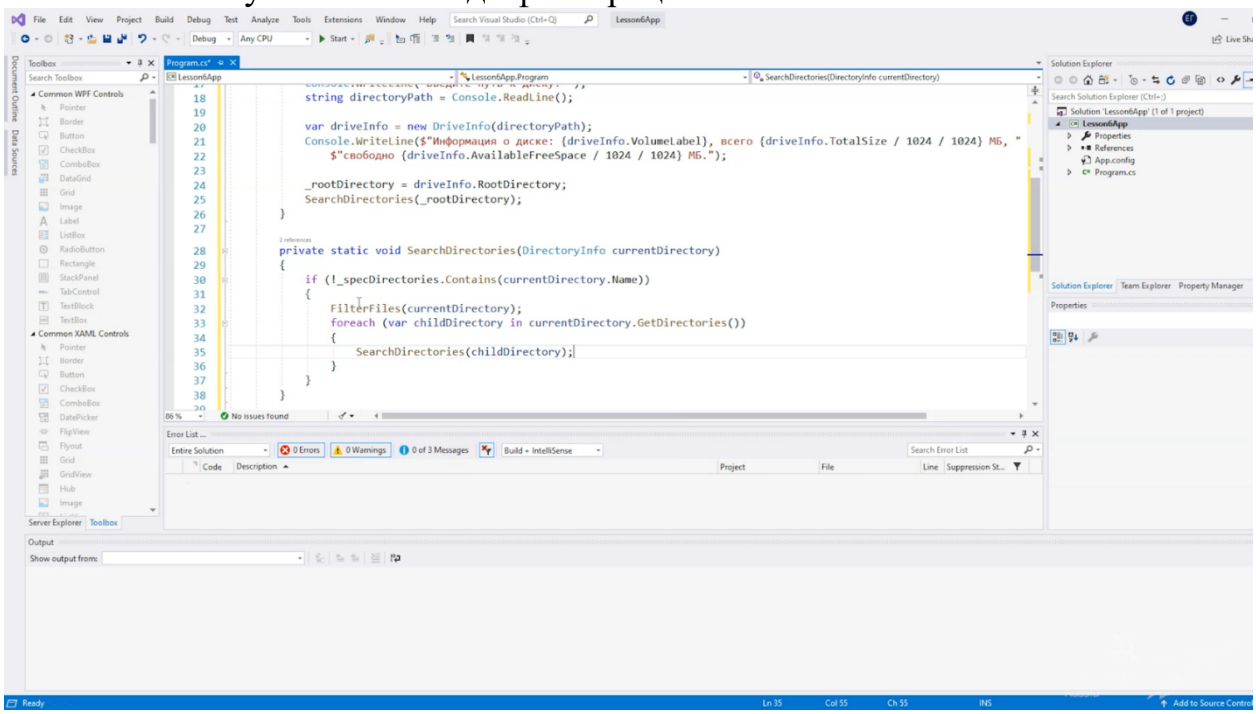


Важно

После сортировки все файлы будут распределены на изображения,

документы и прочее

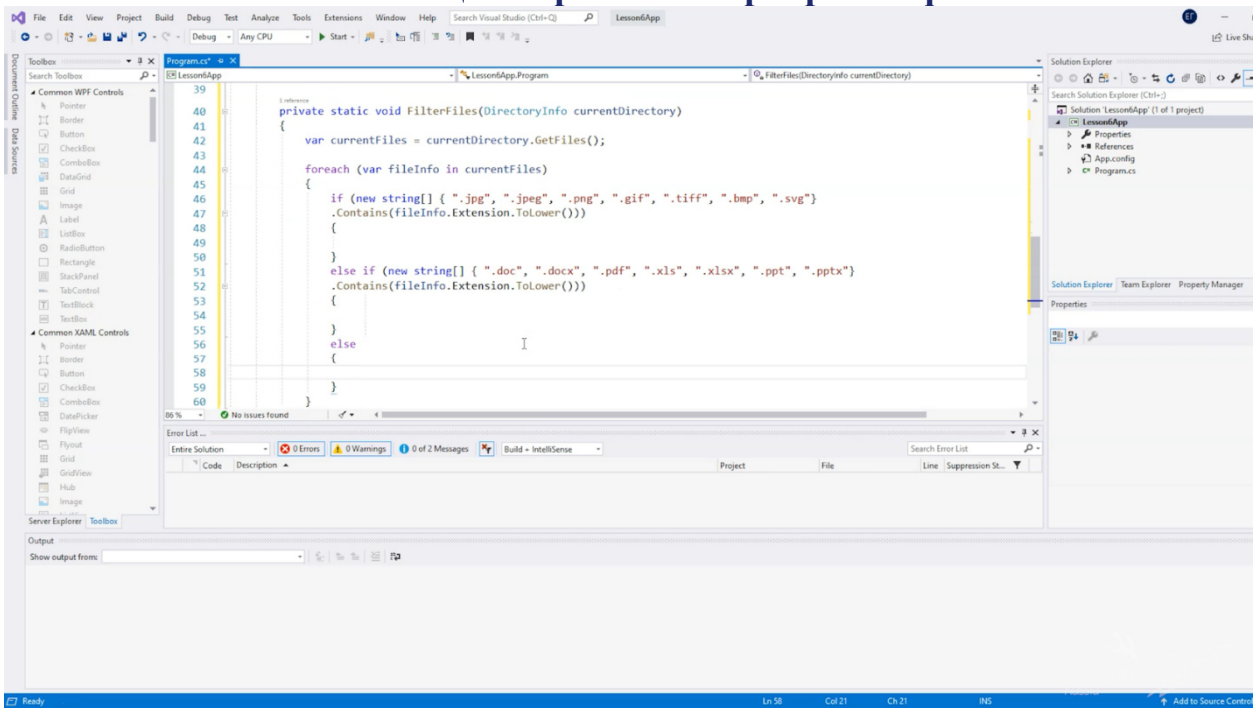
3. Реализуем тело метода фильтрации каталогов



Важно

Метод SearchDirectories() рекурсивно разбирает файлы методом FilterFiles() во всех подкаталогах данного каталога. Функциональность работы для работы с файлами предоставляют классы File и FileInfo

Реализация первичной сортировки файлов

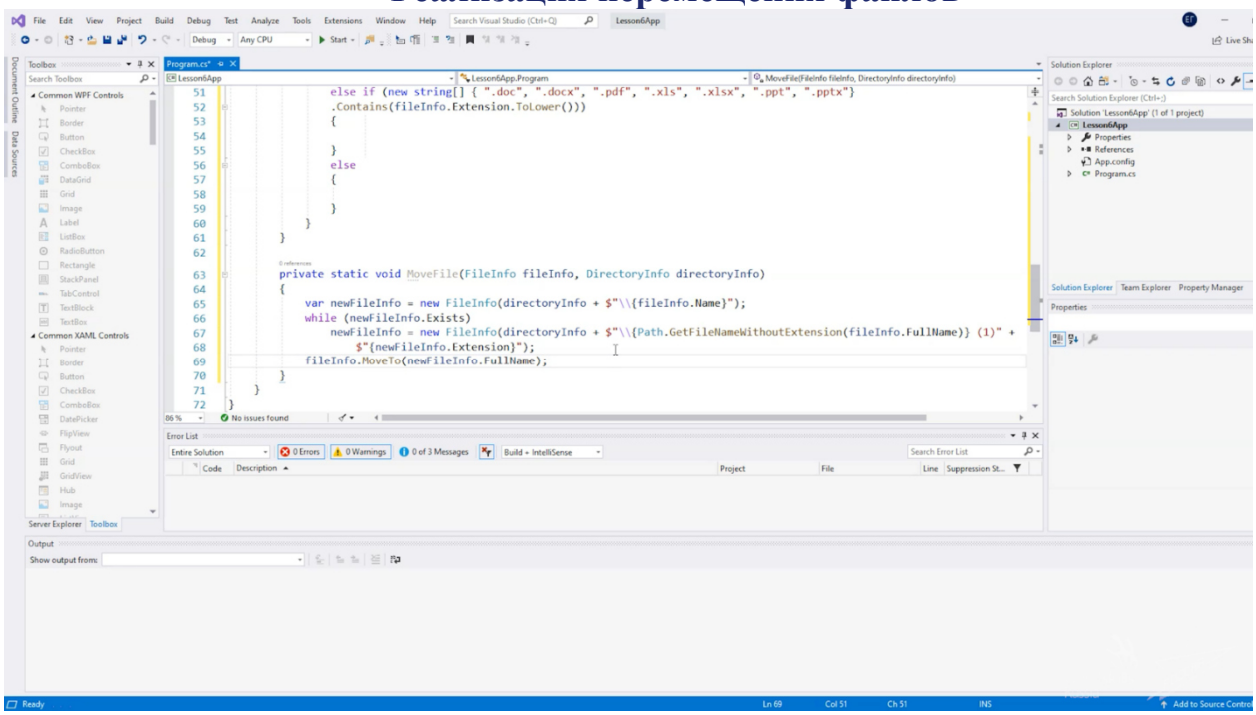


Важно

Метод GetFiles() позволяет получить все файлы в данной директории.

С помощью цикла foreach файлы далее распределяются по папкам в зависимости от их типа (изображения, файлы, прочее)

Реализация перемещения файлов



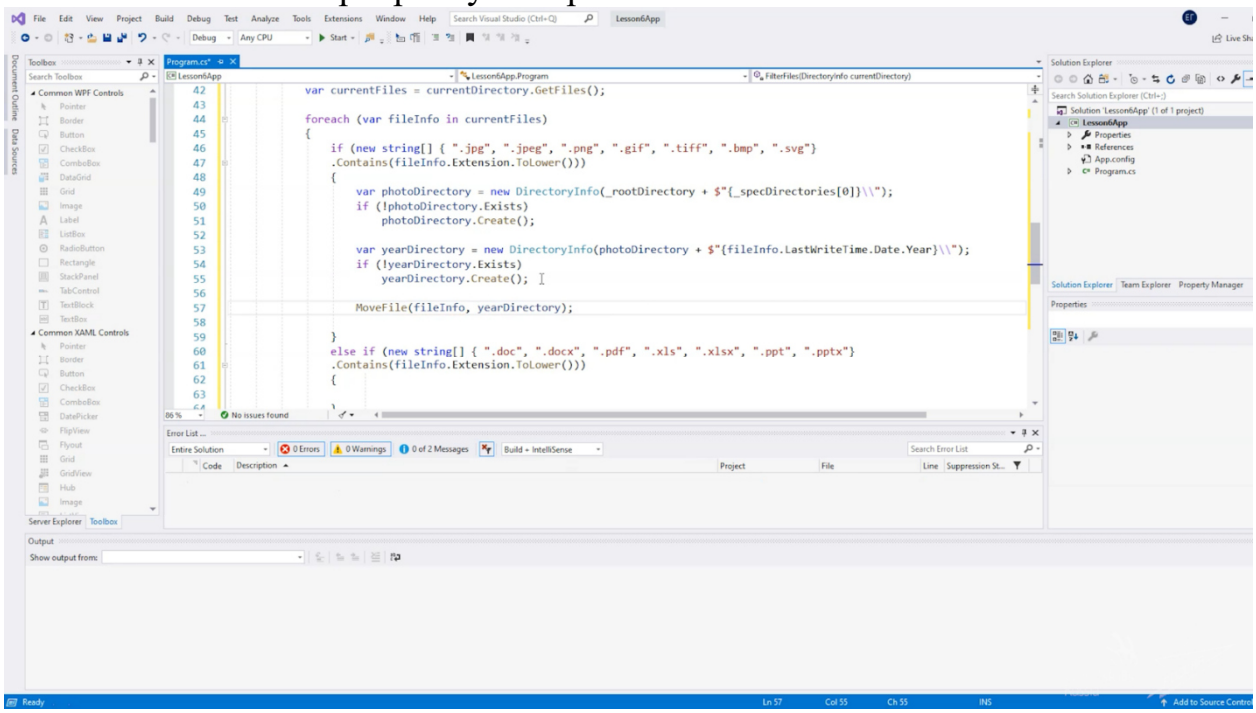
```
            else if (new string[] { ".doc", ".docx", ".pdf", ".xls", ".xlsx", ".ppt", ".pptx"}
            .Contains(fileInfo.Extension.ToLower()))
            {

            }
            else
            {

            }
        }
    }

    private static void MoveFile(FileInfo fileInfo, DirectoryInfo directoryInfo)
    {
        var newFileInfo = new FileInfo(directoryInfo + $"\\{fileInfo.Name}");
        while (newFileInfo.Exists)
            newFileInfo = new FileInfo(directoryInfo + $"\\{Path.GetFileNameWithoutExtension(fileInfo.FullName)} (1)" +
                $"{newFileInfo.Extension}");
        fileInfo.MoveTo(newFileInfo.FullName);
    }
}
```

Важно

Метод MoveFile() перемещает файл (методом MoveTo()) в папку для хранения отсортированных файлов. Наличие дубликатов проверяется свойством Exists. Если файл является дубликатом, в его название с помощью метода Path. GetFileNameWithoutExtension() добавляется «(1)» перед расширением

Уточнение критериев сортировки файлов

1. Уточняем сортировку изображений

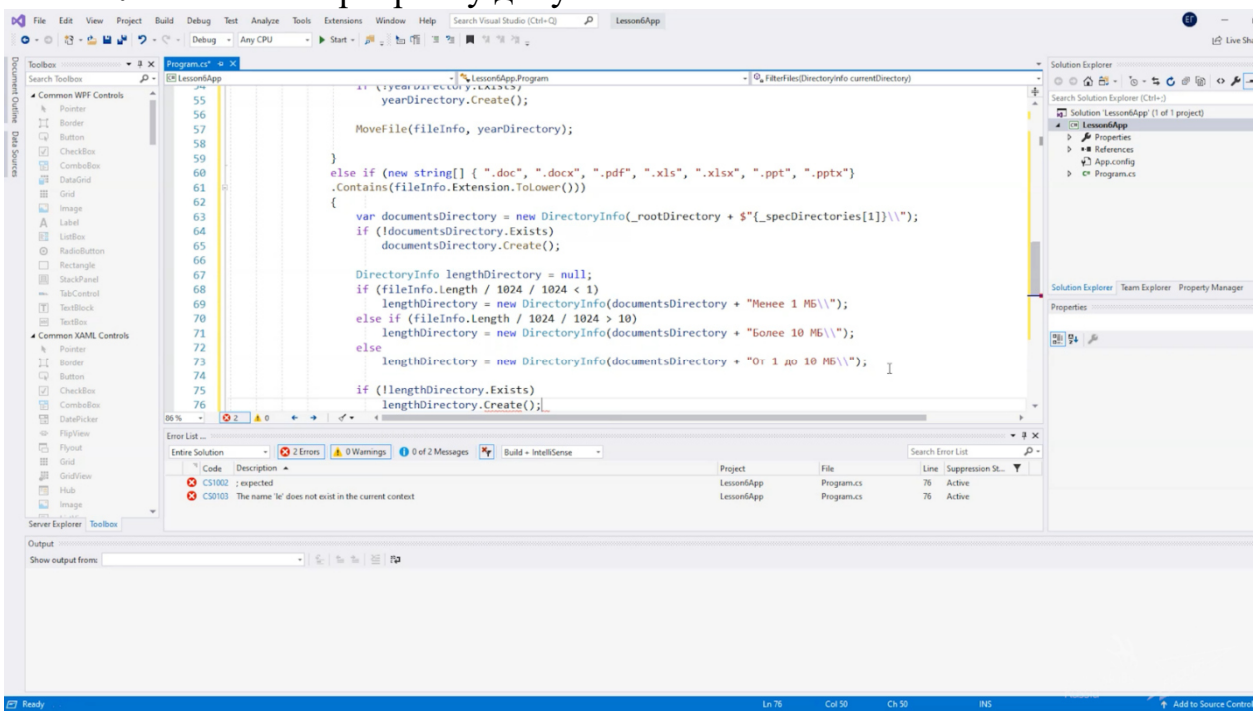


```
        var currentFiles = currentDirectory.GetFiles();

        foreach (var fileInfo in currentFiles)
        {
            if (new string[] { ".jpg", ".jpeg", ".png", ".gif", ".tiff", ".bmp", ".svg"}
            .Contains(fileInfo.Extension.ToLower()))
            {
                var photoDirectory = new DirectoryInfo(_rootDirectory + $"{_specDirectories[0]}\\");
                if (!photoDirectory.Exists)
                    photoDirectory.Create();

                var yearDirectory = new DirectoryInfo(photoDirectory + $"{fileInfo.LastWriteTime.Date.Year}\\");
                if (!yearDirectory.Exists)
                    yearDirectory.Create();

                MoveFile(fileInfo, yearDirectory);

            }
            else if (new string[] { ".doc", ".docx", ".pdf", ".xls", ".xlsx", ".ppt", ".pptx"}
            .Contains(fileInfo.Extension.ToLower()))
            {
```

Важно

Изображения дополнительно сортируются по годам. Свойство LastWriteTime позволяет получить дату создания изображения

2. Уточняем сортировку документов

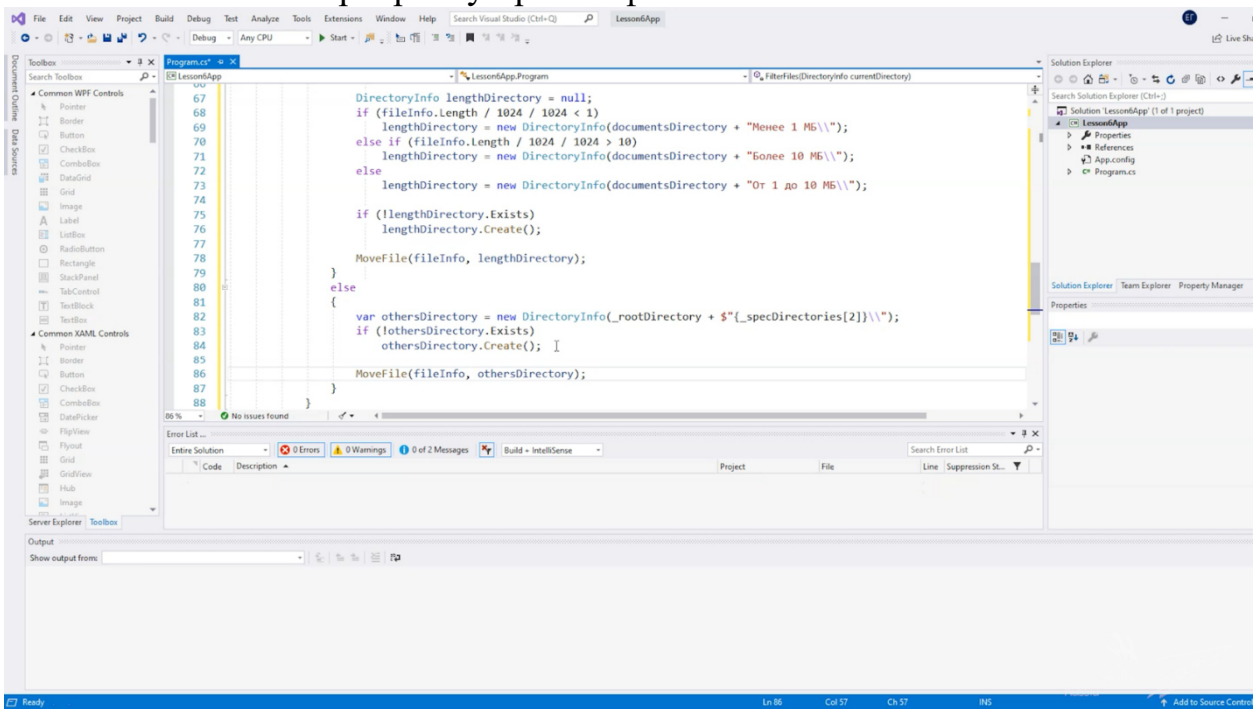


```
            yearDirectory.Create();

        MoveFile(fileInfo, yearDirectory);

    }
    else if (new string[] { ".doc", ".docx", ".pdf", ".xls", ".xlsx", ".ppt", ".pptx"}
    .Contains(fileInfo.Extension.ToLower()))
    {
        var documentsDirectory = new DirectoryInfo(_rootDirectory + $"{_specDirectories[1]}\\");
        if (!documentsDirectory.Exists)
            documentsDirectory.Create();

        DirectoryInfo lengthDirectory = null;
        if (fileInfo.Length / 1024 / 1024 < 1)
            lengthDirectory = new DirectoryInfo(documentsDirectory + "Менее 1 МБ\\");
        else if (fileInfo.Length / 1024 / 1024 > 10)
            lengthDirectory = new DirectoryInfo(documentsDirectory + "Более 10 МБ\\");
        else
            lengthDirectory = new DirectoryInfo(documentsDirectory + "От 1 до 10 МБ\\");

        if (!lengthDirectory.Exists)
            lengthDirectory.Create();
```

Важно

Документы дополнительно сортируются по размеру (менее 1 Мб; более 1 Мб и менее 10 Мб; более 10 Мб). Для получения информации о размере используется свойство Length. Так же, как и изображения, документы далее перемещаются с помощью метода «MoveFile (fileInfo, lengthDirectory)»

3. Уточняем сортировку прочих файлов



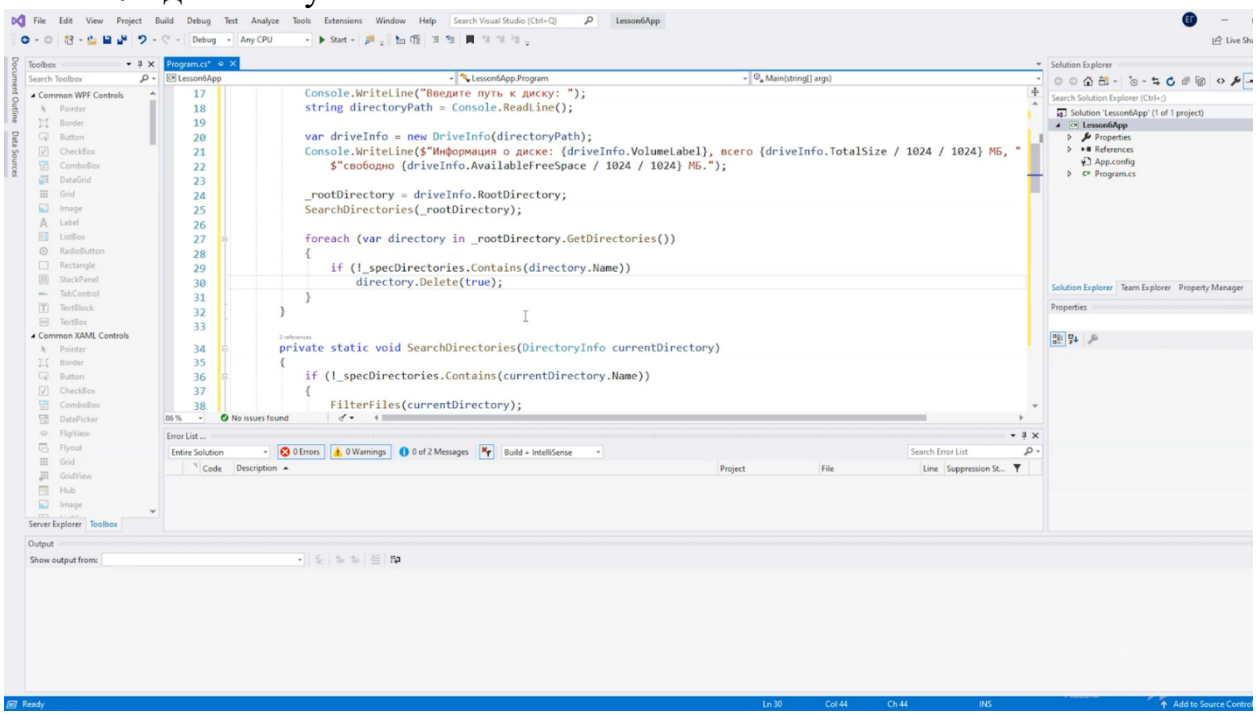
```
        DirectoryInfo lengthDirectory = null;
        if (fileInfo.Length / 1024 / 1024 < 1)
            lengthDirectory = new DirectoryInfo(documentsDirectory + "Менее 1 МБ\\");
        else if (fileInfo.Length / 1024 / 1024 > 10)
            lengthDirectory = new DirectoryInfo(documentsDirectory + "Более 10 МБ\\");
        else
            lengthDirectory = new DirectoryInfo(documentsDirectory + "От 1 до 10 МБ\\");

        if (!lengthDirectory.Exists)
            lengthDirectory.Create();

        MoveFile(fileInfo, lengthDirectory);
    }
    else
    {
        var othersDirectory = new DirectoryInfo(_rootDirectory + $"{_specDirectories[2]}\\");
        if (!othersDirectory.Exists)
            othersDirectory.Create();

        MoveFile(fileInfo, othersDirectory);
    }
}
```

Важно

Прочие файлы (при наличии) перемещаются в папку «Прочее»

Завершение работы с Main()

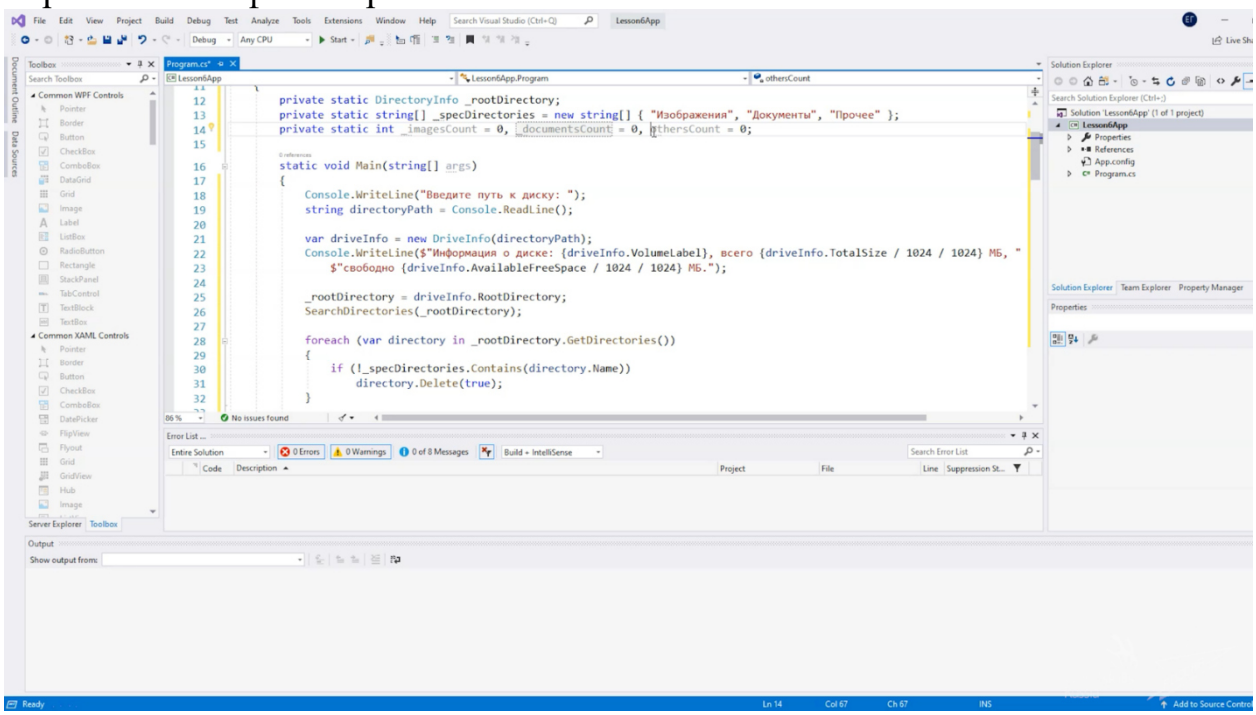
1. Удаляем пустые каталоги



Важно

Пустые каталоги удаляются с помощью метода Delete() с параметром «Recursive=true», который разрешает удалять все подкаталоги

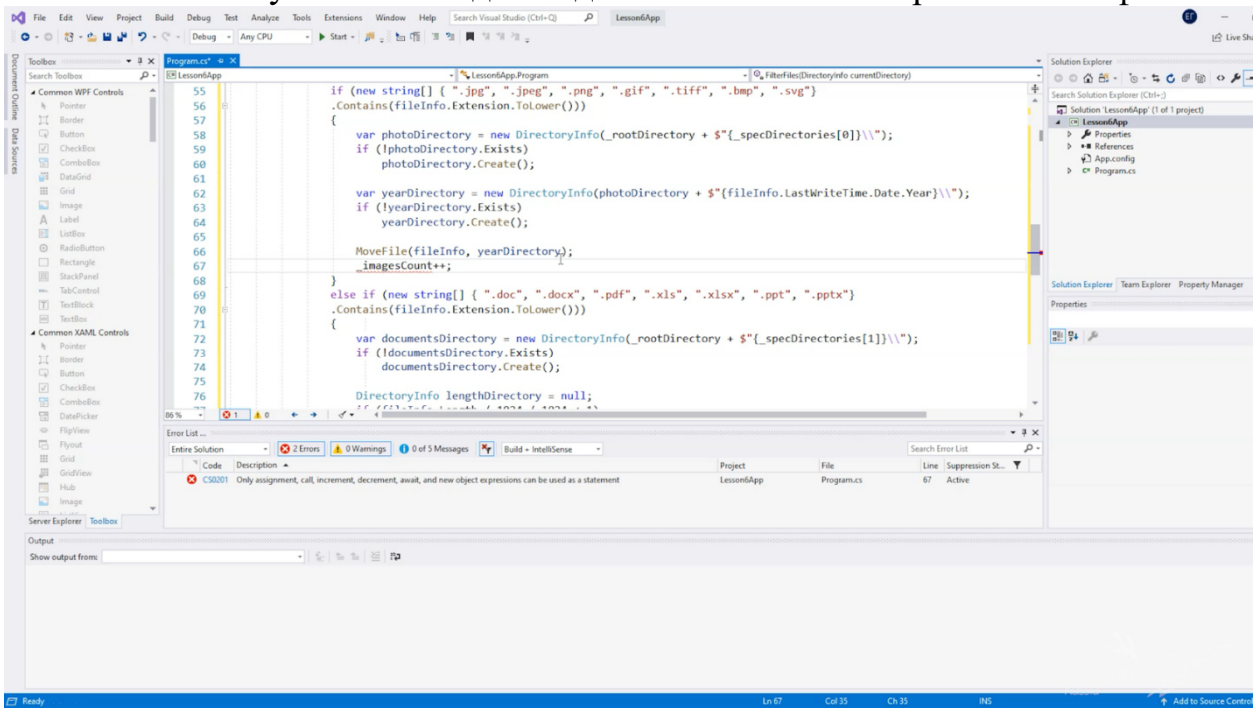
2. Создаем счетчики для хранения информации о количестве обработанных файлов разных типов



Важно

Для каждого типа файлов создается свой счетчик

3. Используем счетчик для подсчета количества обработанных файлов

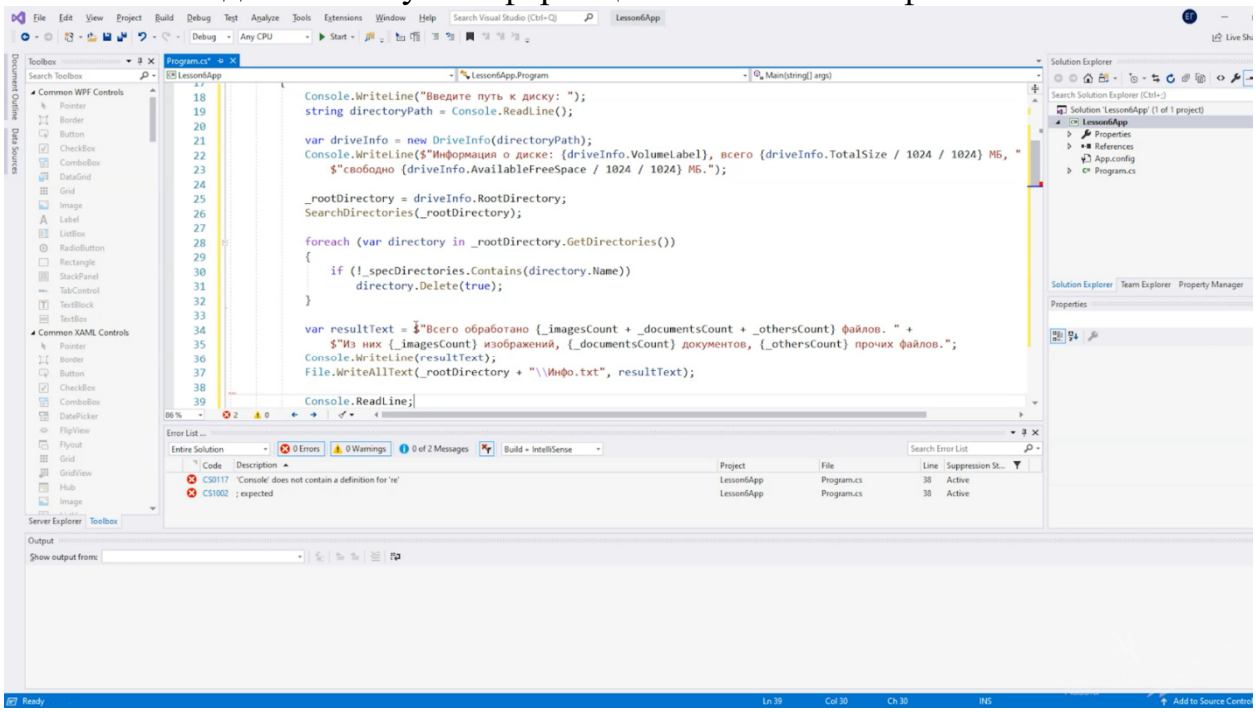


```
if (new string[] { ".jpg", ".jpeg", ".png", ".gif", ".tiff", ".bmp", ".svg"}
.Contains(fileInfo.Extension.ToLower()))
{
    var photoDirectory = new DirectoryInfo(_rootDirectory + $"{_specDirectories[0]}\\");
    if (!photoDirectory.Exists)
        photoDirectory.Create();

    var yearDirectory = new DirectoryInfo(photoDirectory + $"{fileInfo.LastWriteTime.Date.Year}\\");
    if (!yearDirectory.Exists)
        yearDirectory.Create();

    MoveFile(fileInfo, yearDirectory);
    _imagesCount++;
}
else if (new string[] { ".doc", ".docx", ".pdf", ".xls", ".xlsx", ".ppt", ".pptx"}
.Contains(fileInfo.Extension.ToLower()))
{
    var documentsDirectory = new DirectoryInfo(_rootDirectory + $"{_specDirectories[1]}\\");
    if (!documentsDirectory.Exists)
        documentsDirectory.Create();

    DirectoryInfo lengthDirectory = null;
```

Важно

Счетчик увеличивается на один после каждого перемещения файла. Аналогичный код добавляется после перемещения документов и прочих файлов

4. Выводим итоговую информацию в консоль и в файл

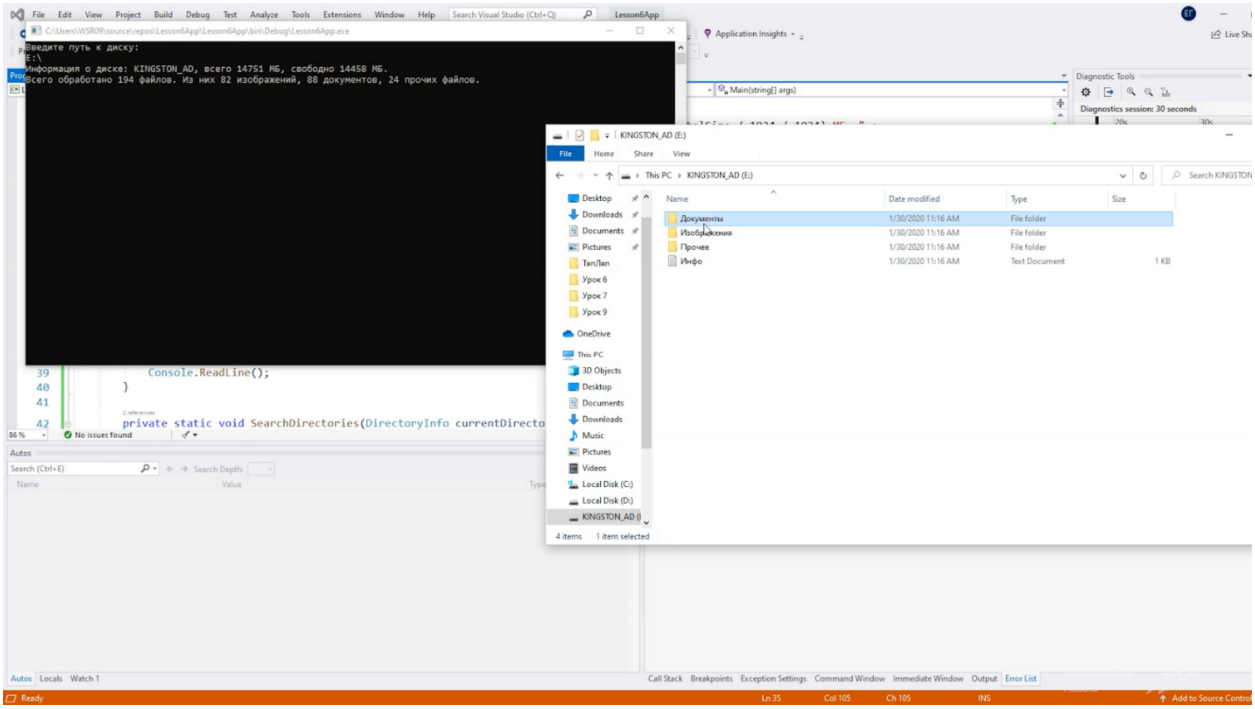


```
Console.WriteLine("Введите путь к диску: ");
string directoryPath = Console.ReadLine();

var driveInfo = new DriveInfo(directoryPath);
Console.WriteLine($"Информация о диске: {driveInfo.VolumeLabel}, всего {driveInfo.TotalSize / 1024 / 1024} МБ, "
    $"свободно {driveInfo.AvailableFreeSpace / 1024 / 1024} МБ.");

_rootDirectory = driveInfo.RootDirectory;
SearchDirectories(_rootDirectory);

foreach (var directory in _rootDirectory.GetDirectories())
{
    if (!_specDirectories.Contains(directory.Name))
        directory.Delete(true);
}

var resultText = $"Всего обработано {_imagesCount + _documentsCount + _othersCount} файлов. " +
    $"Из них {_imagesCount} изображений, {_documentsCount} документов, {_othersCount} прочих файлов.";
Console.WriteLine(resultText);
File.WriteAllText(_rootDirectory + "\\Инфо.txt", resultText);

Console.ReadLine;
```

Важно

Текстовый файл «Инфо.txt» создается с помощью метода WriteAllText()

Завершение работы с Main()



Интерактивное задание:
https://nationalteam.worldskills.ru/skills/programmnaya-rabota-s-faylovoy-sistemoy-s-pomoshchyu-prostranstva-imen-system-io/

Практическая работа № 16. Реализация графиков с помощью компонента Chart (System.Windows.Forms.DataVisualization)

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:

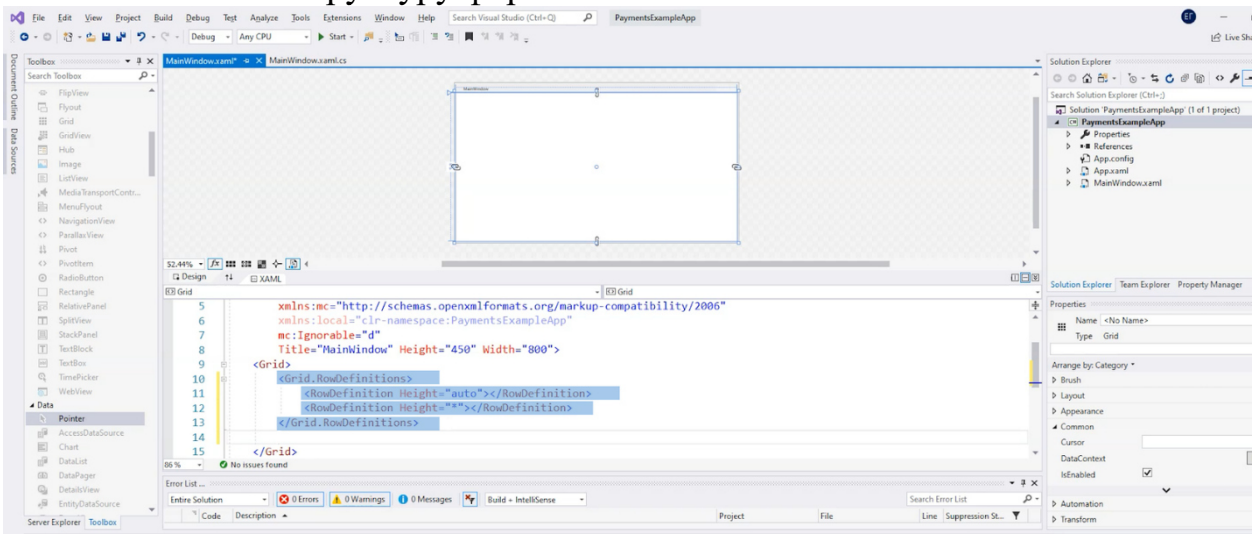
1. Разработка интерфейса приложения
2. Настройка взаимодействия с базой данных
3. Реализация обработки данных

Важно

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

Разработка интерфейса приложения

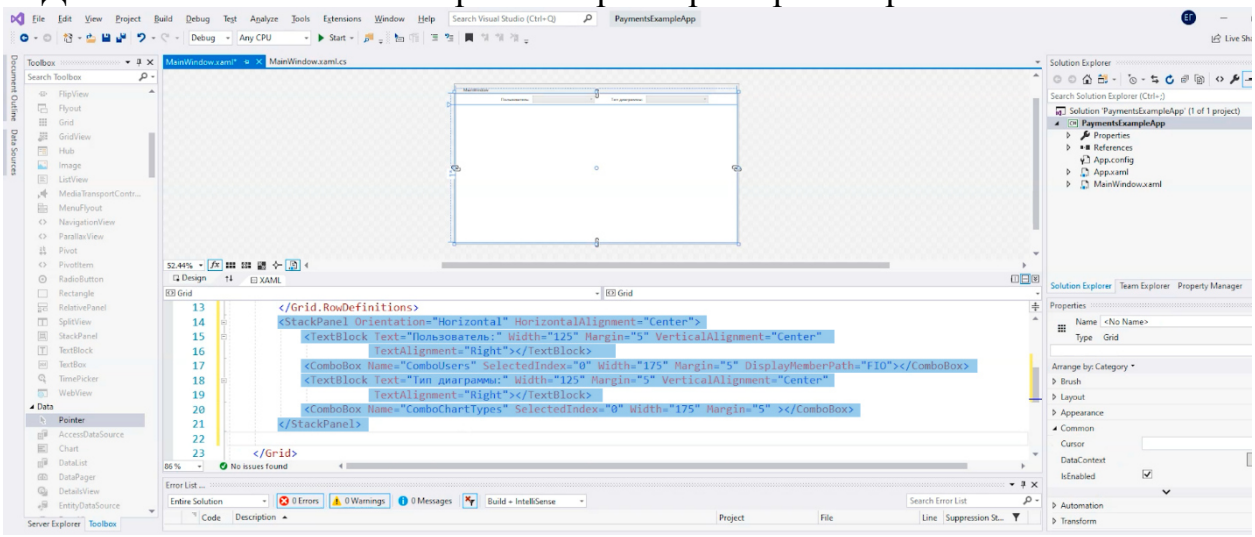
1. Устанавливаем структуру формы



Важно

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

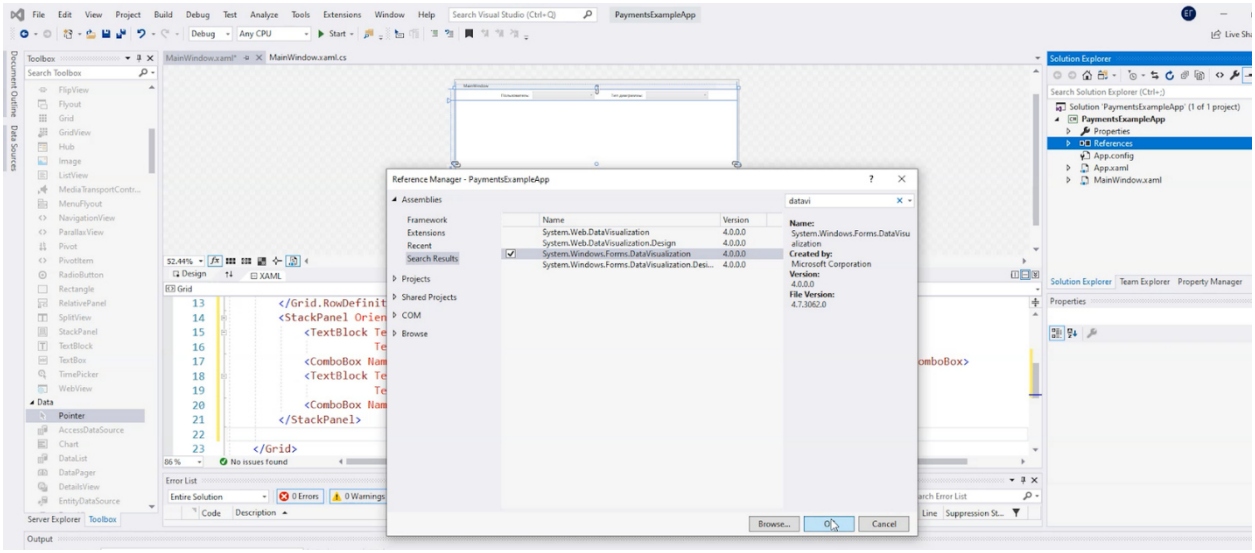
2. Добавляем элементы настройки параметров просмотра



Важно

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

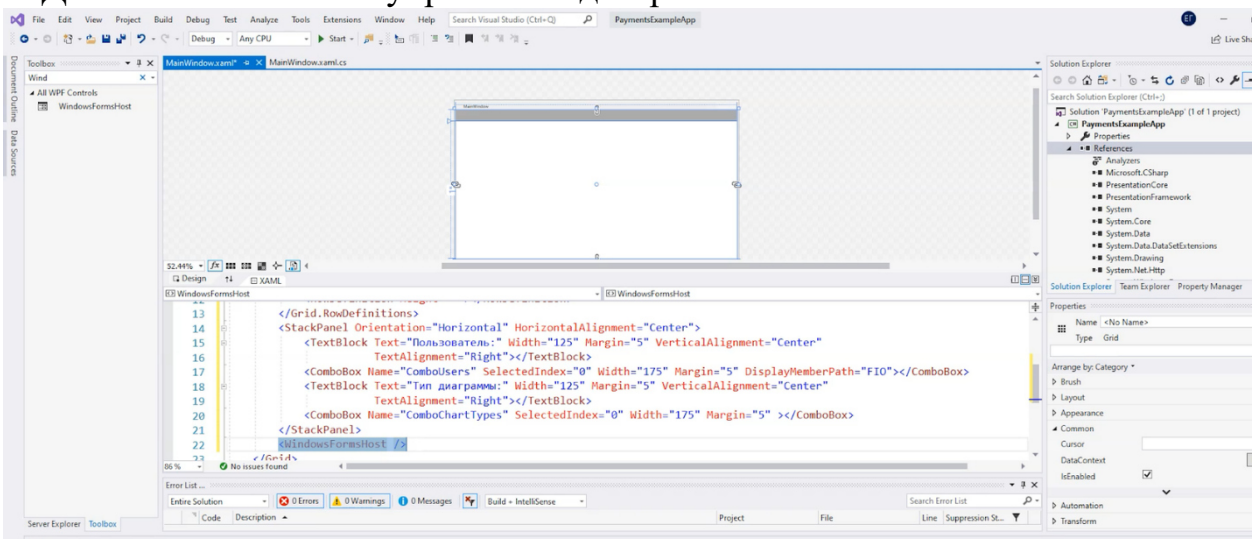
3. Подключаем библиотеки для просмотра диаграмм



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)

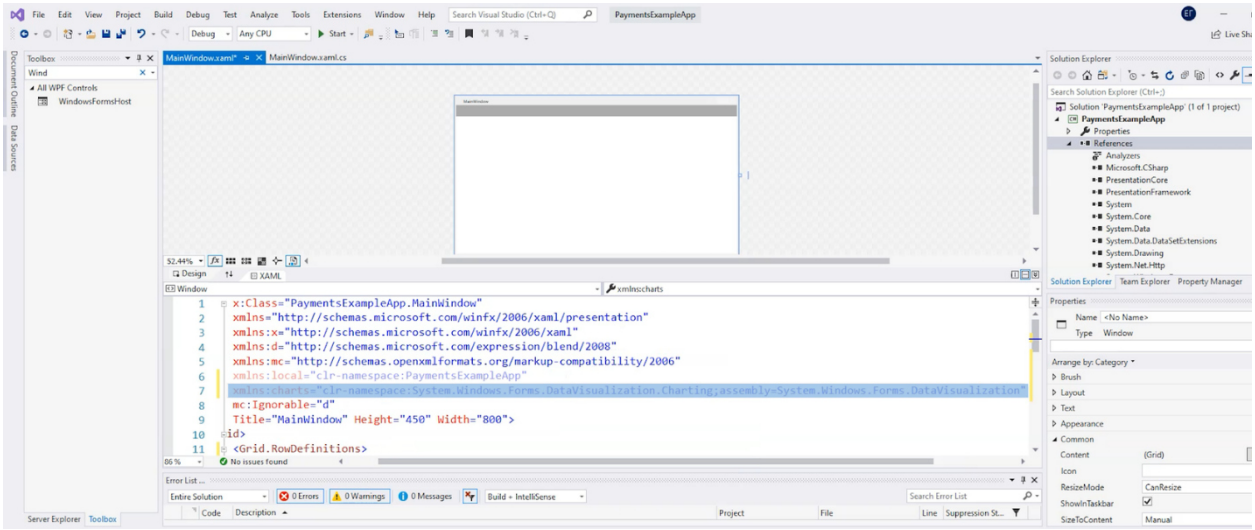
4. Добавляем элементы управления диаграммой



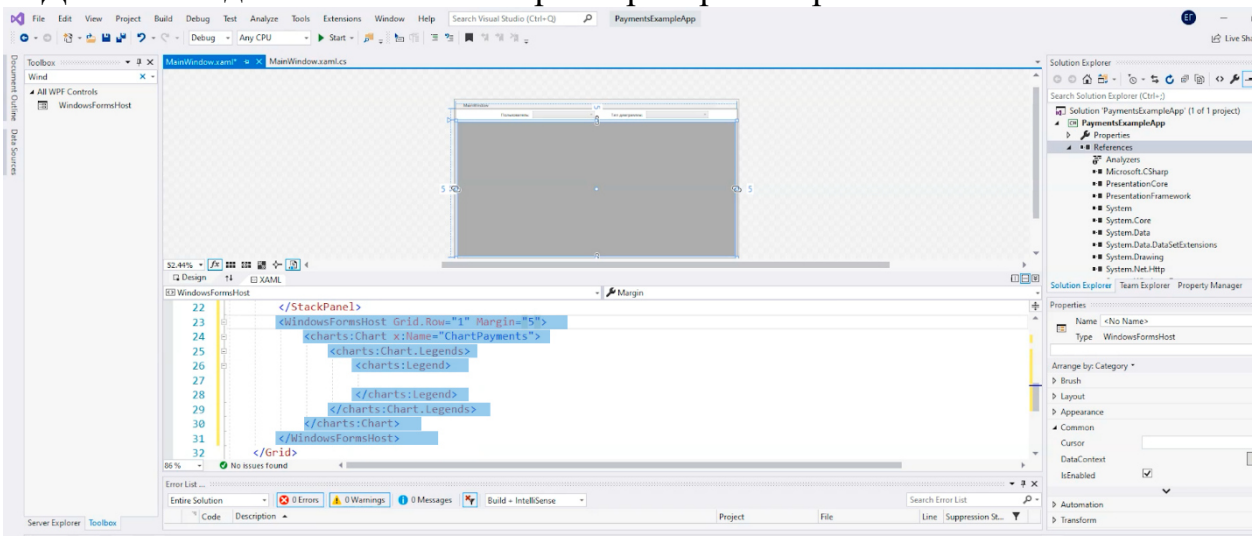
Важно

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

5. Добавляем пространство имен для работы с элементом Chart



6. Добавляем дополнительные параметры просмотра

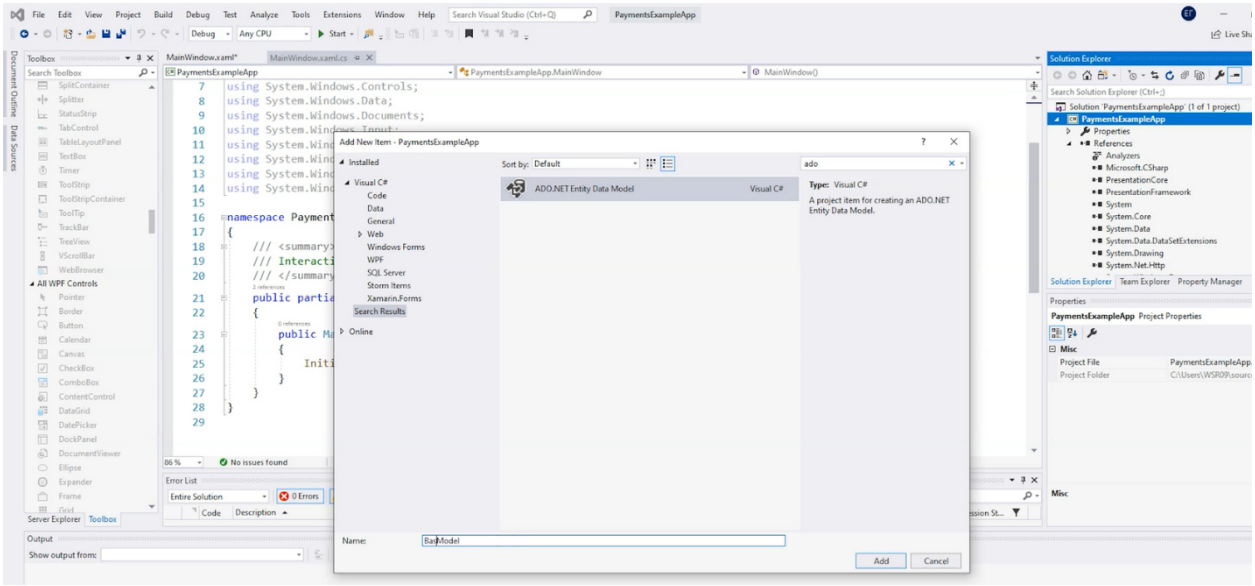


Важно

Дополнительными параметрами являются имя диаграммы, а также легенда

Настройка взаимодействия с базой данных

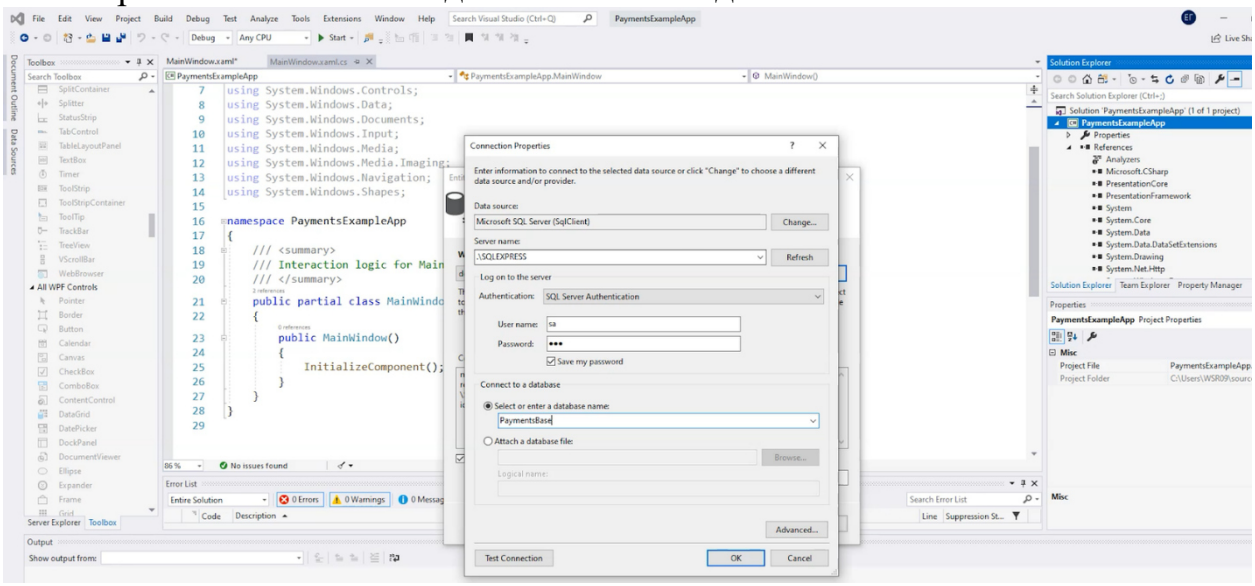
1. Реализуем взаимодействие с базой данных



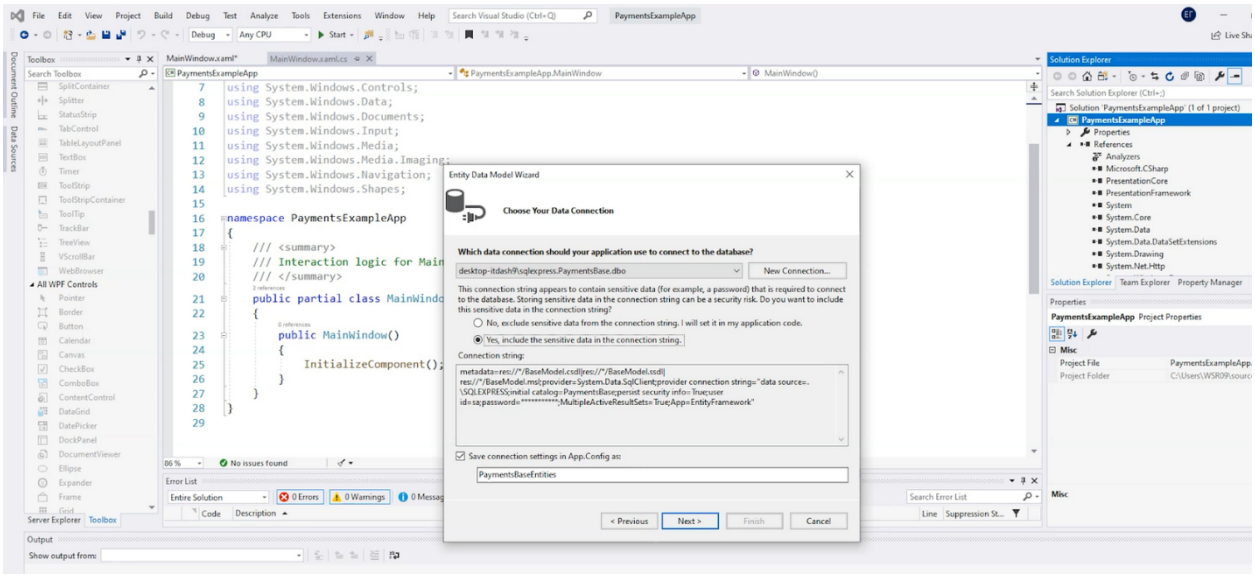
Важно

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

2. Настраиваем свойства подключения базы данных

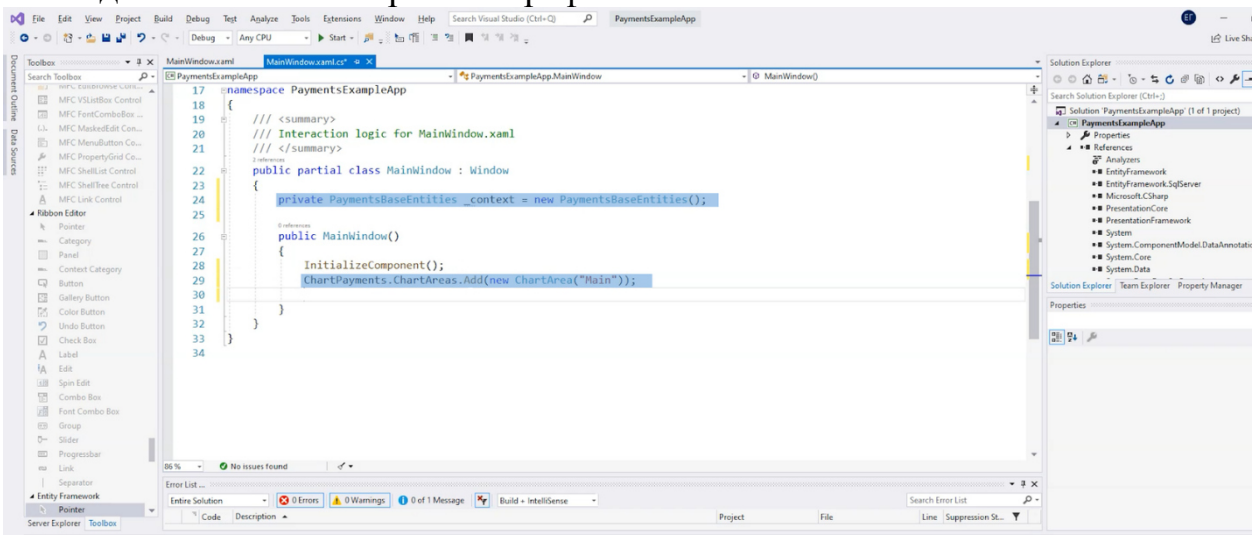


3. Добавляем подключение к базе данных



Реализация обработки данных

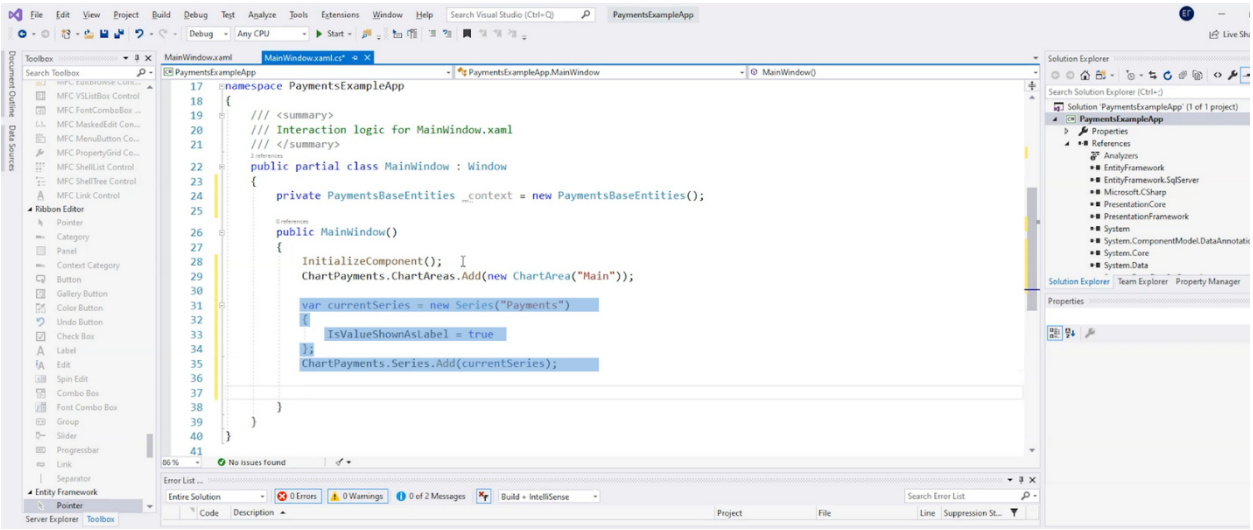
1. Создаем область построения графиков



Важно

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

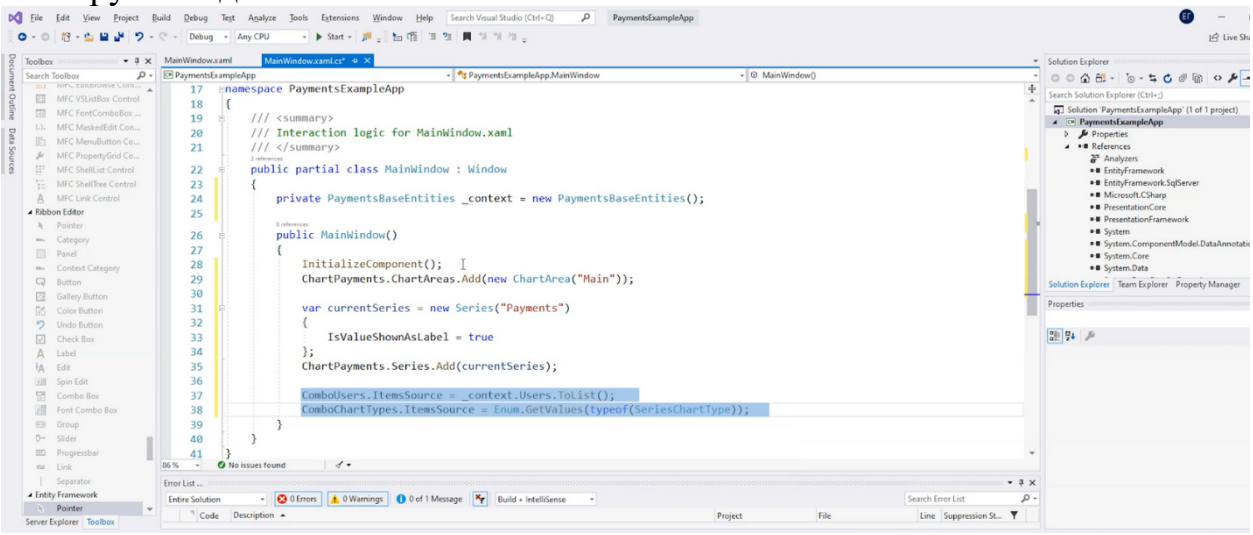
2. Добавляем наборы данных



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

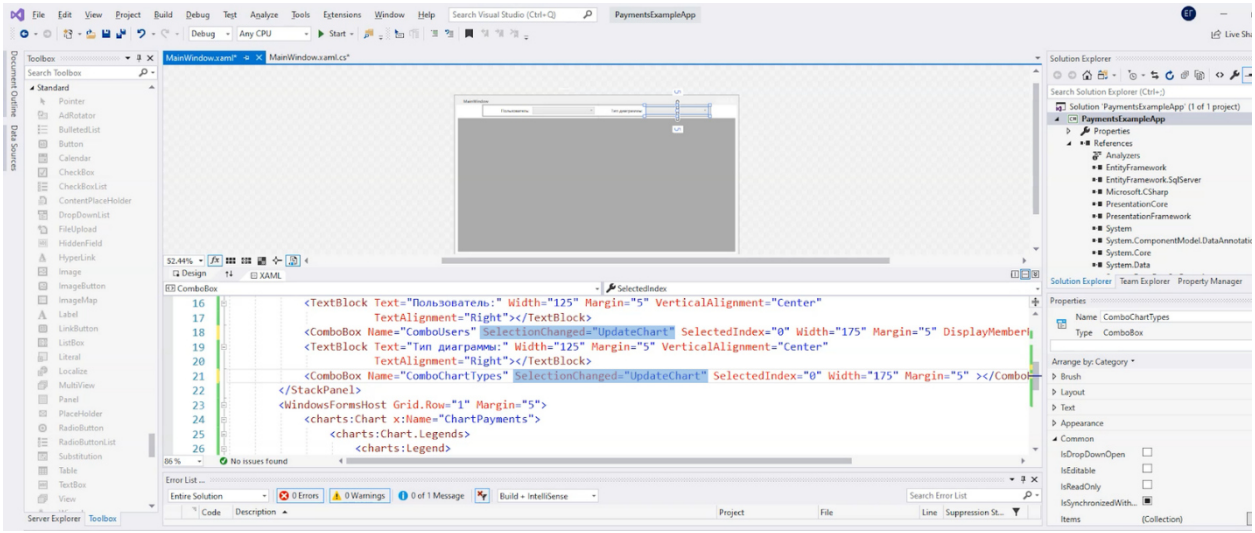
3. Загружаем данные из базы



Важно

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

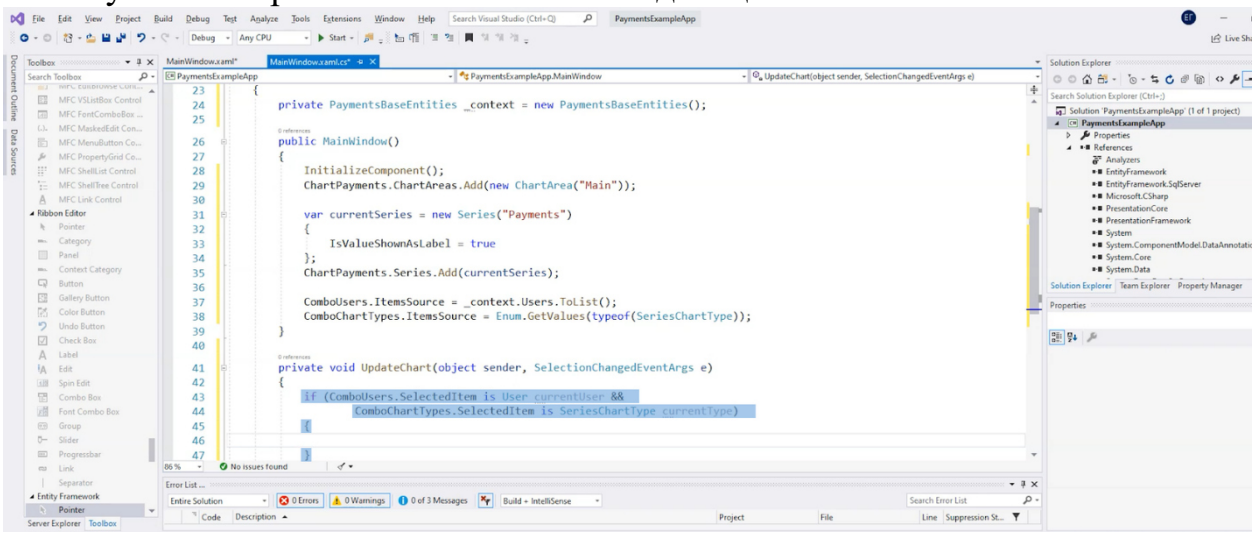
4. Реализуем адаптивное изменение интерфейса



Важно

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

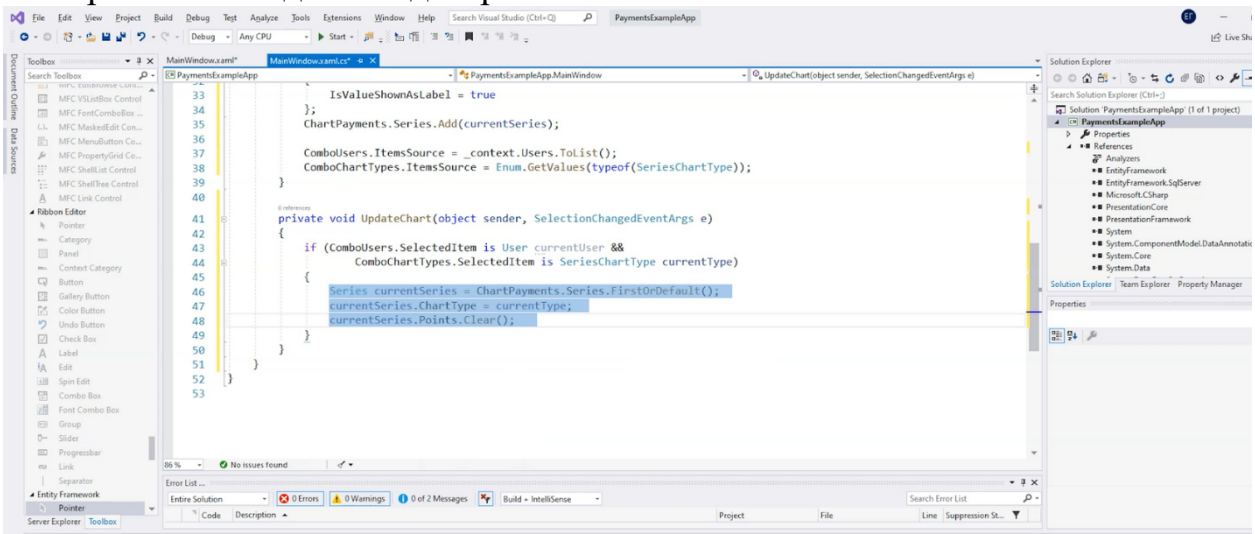
5. Получаем выбранные значения в выпадающих списках



Важно

Значения получаются как currentUser и currentType

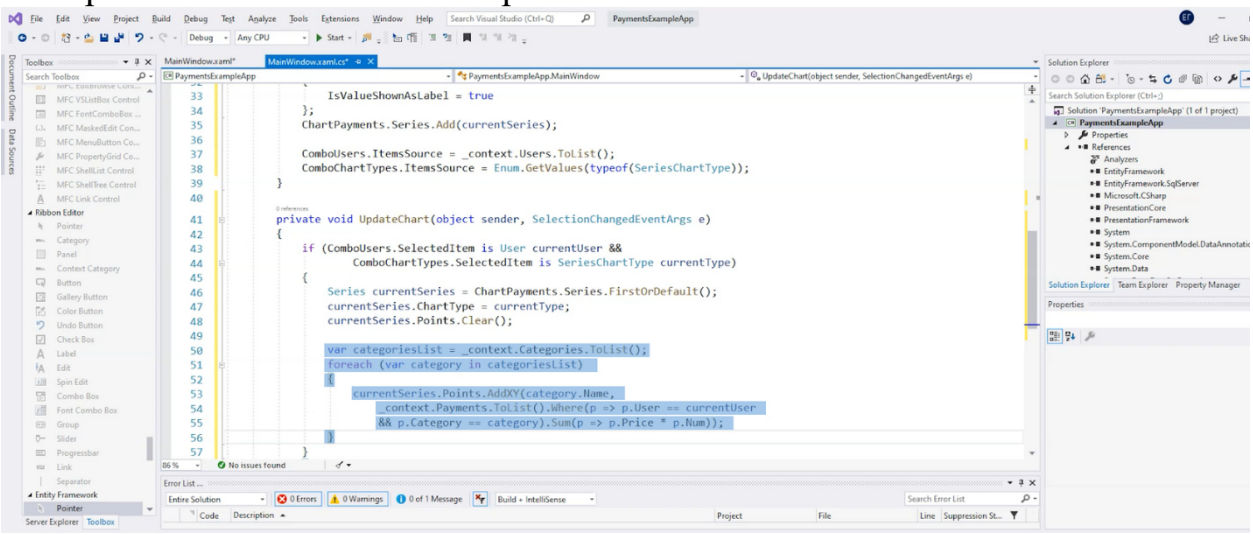
6. Обрабатываем данные диаграммы



Важно

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

7. Обрабатываем список категорий

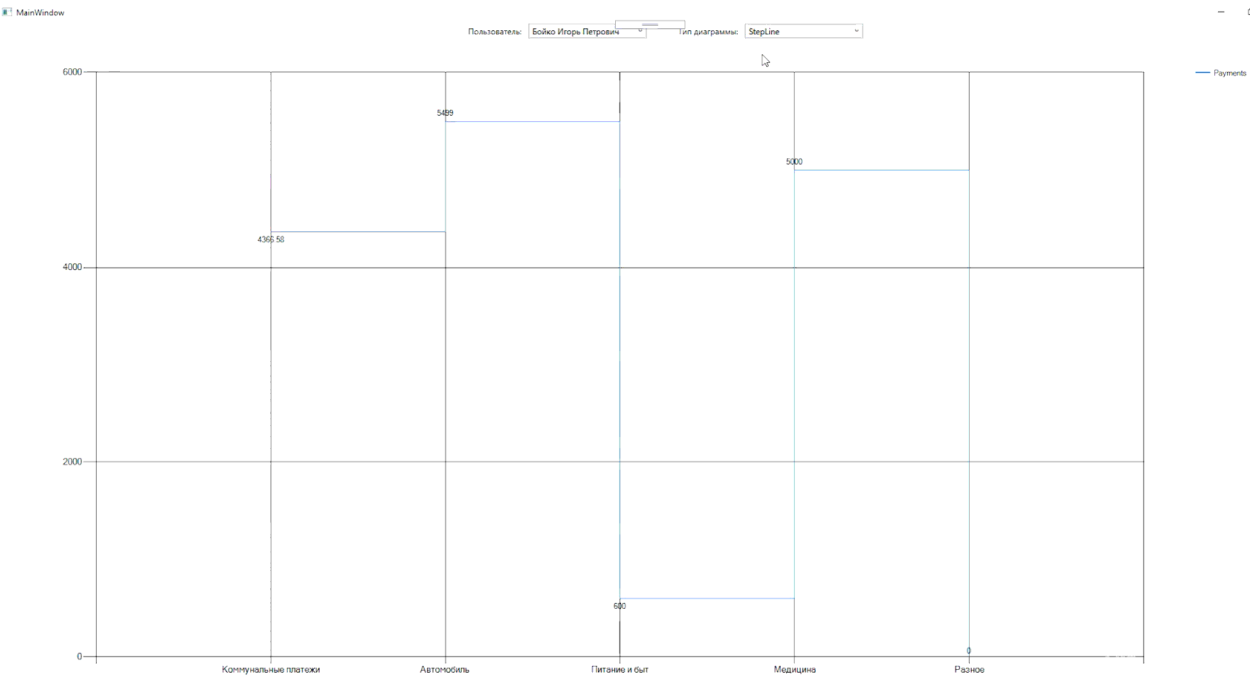


```
            IsValueShownAsLabel = true
        };
        ChartPayments.Series.Add(currentSeries);

        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)
        {
            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



Интерактивное задание: https://nationalteam.worldskills.ru/skills/realizatsiya-grafikov-s-pomoshchyu-komponenta-chart-system-windows-forms-datavisualization/

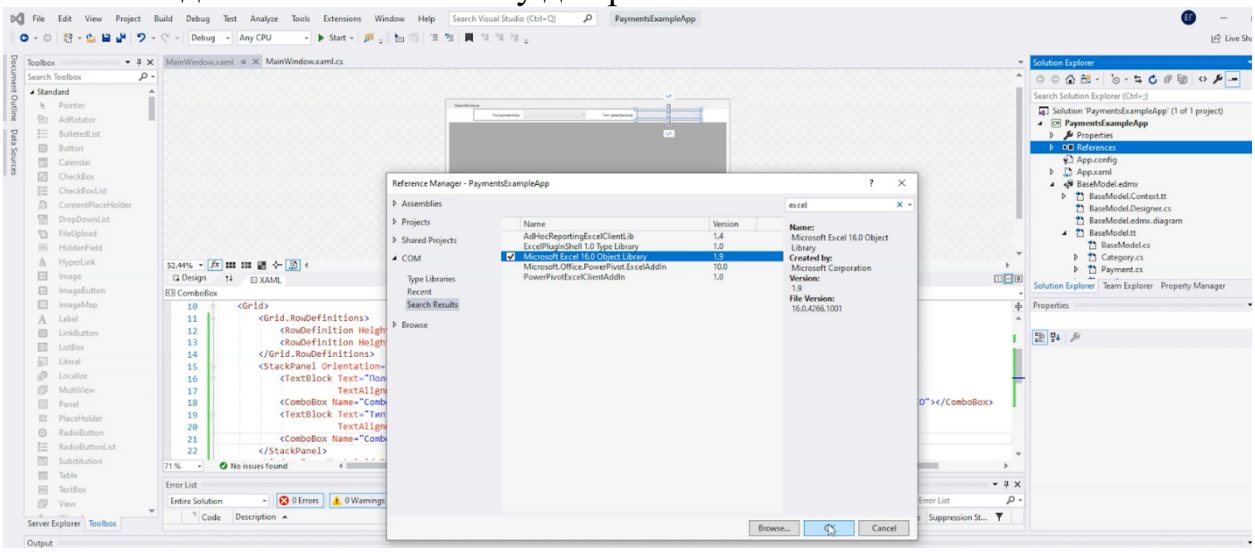
Практическая работа № 17. Программная работа с таблицами Excel с помощью библиотеки Microsoft.Office.Interop.Excel

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:

1. Предварительные шаги
2. Реализация экспорта
3. Проверка корректной работы приложения

Предварительные шаги

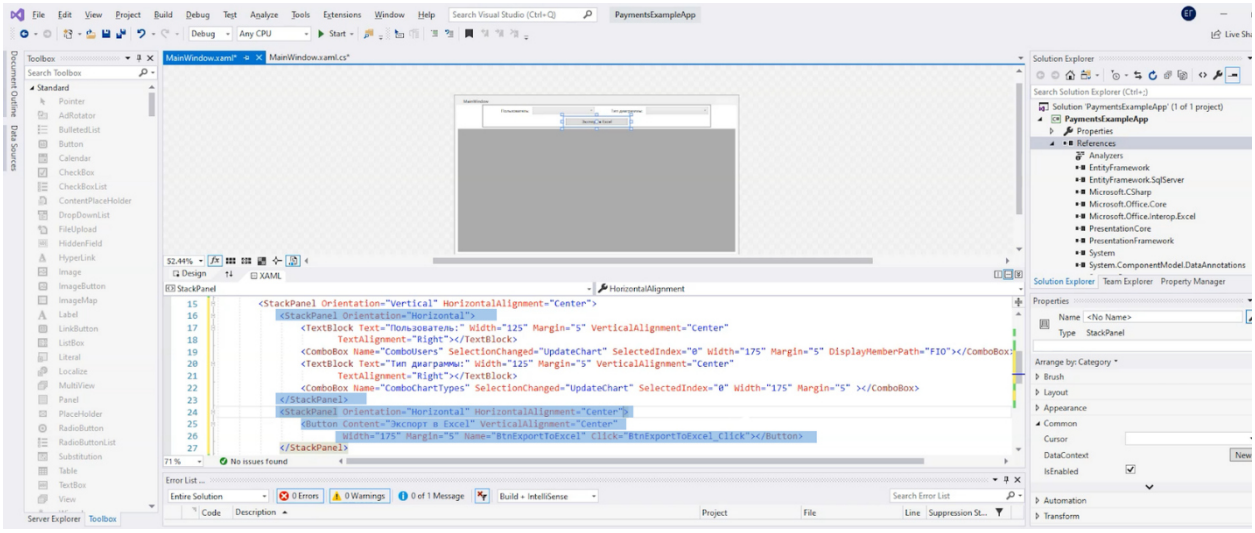
1. Подключаем библиотеку для работы с Excel



Важно

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

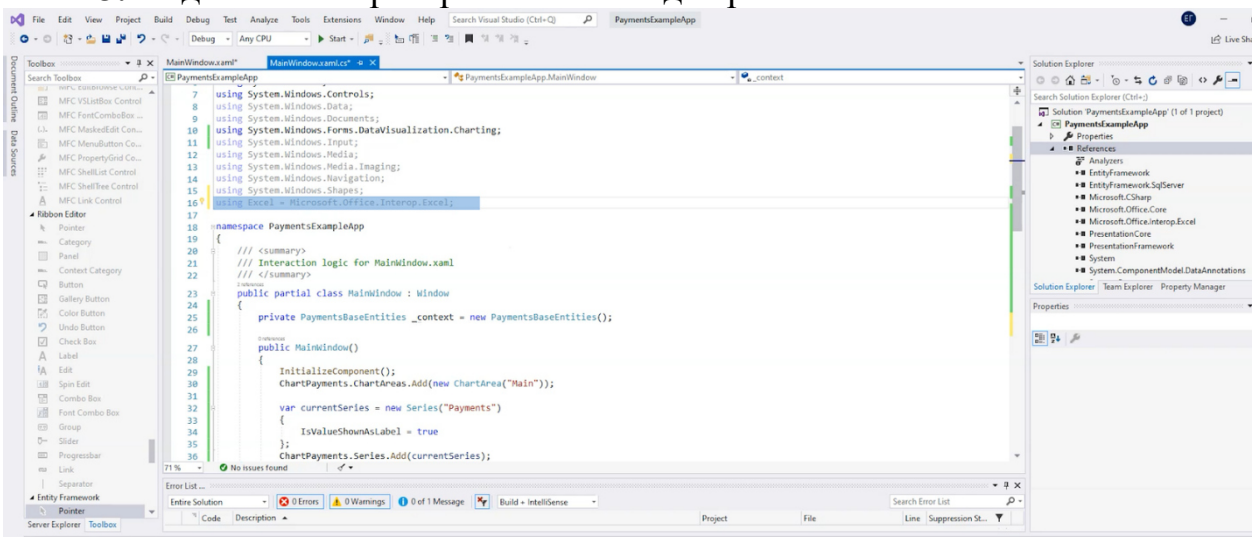
2. Добавляем кнопку экспорта



Важно

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

3. Подключаем пространство имен для работы с Excel

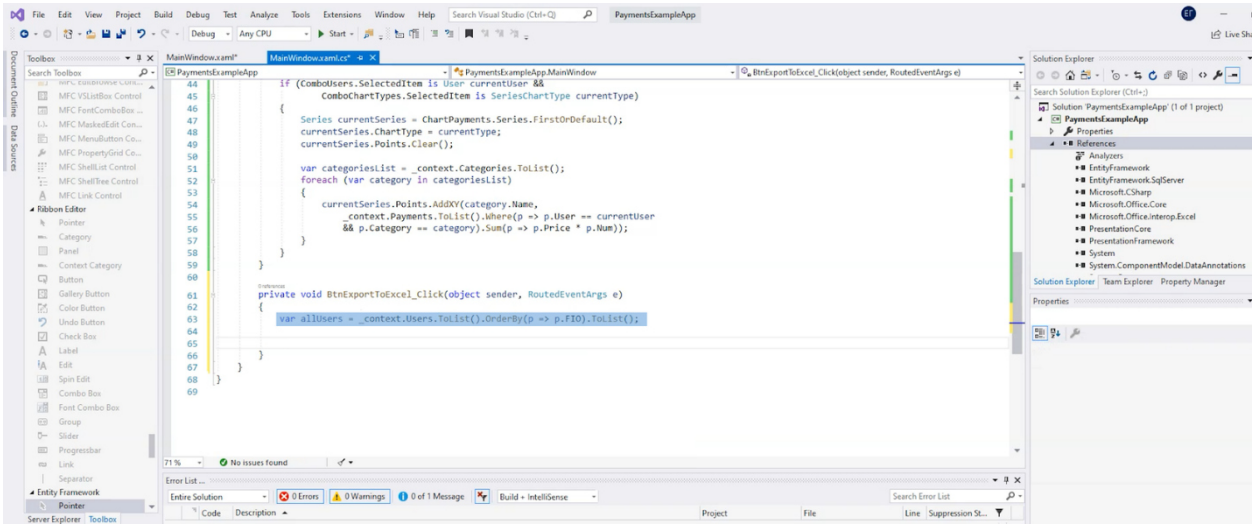


Важно

Требуемое пространство имен подключается с помощью директивы using

Реализация экспорта

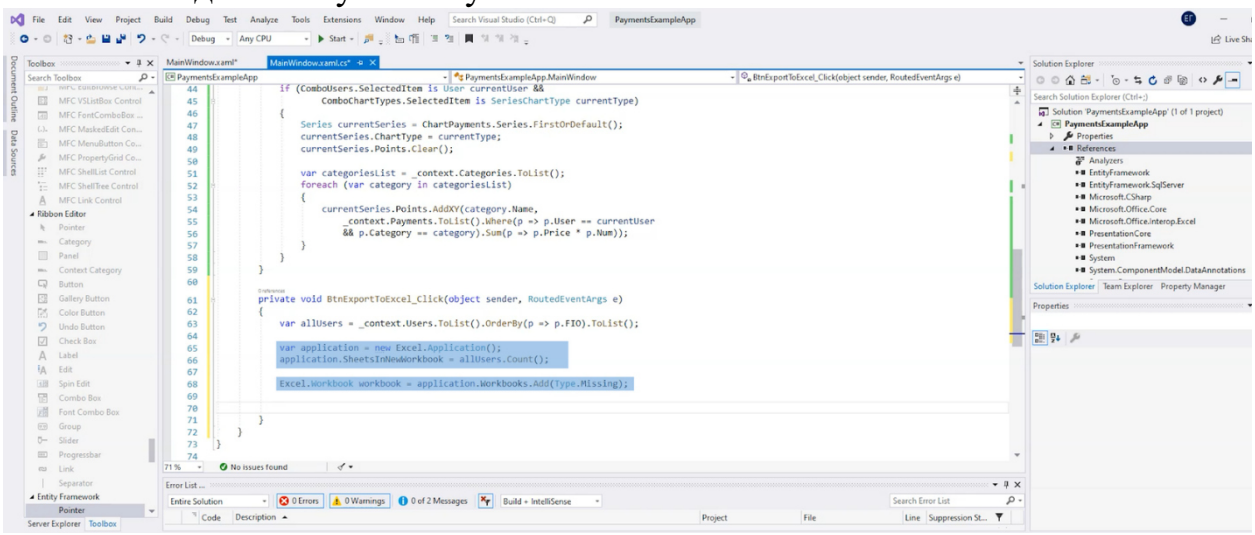
1. Получаем список пользователей



Важно

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

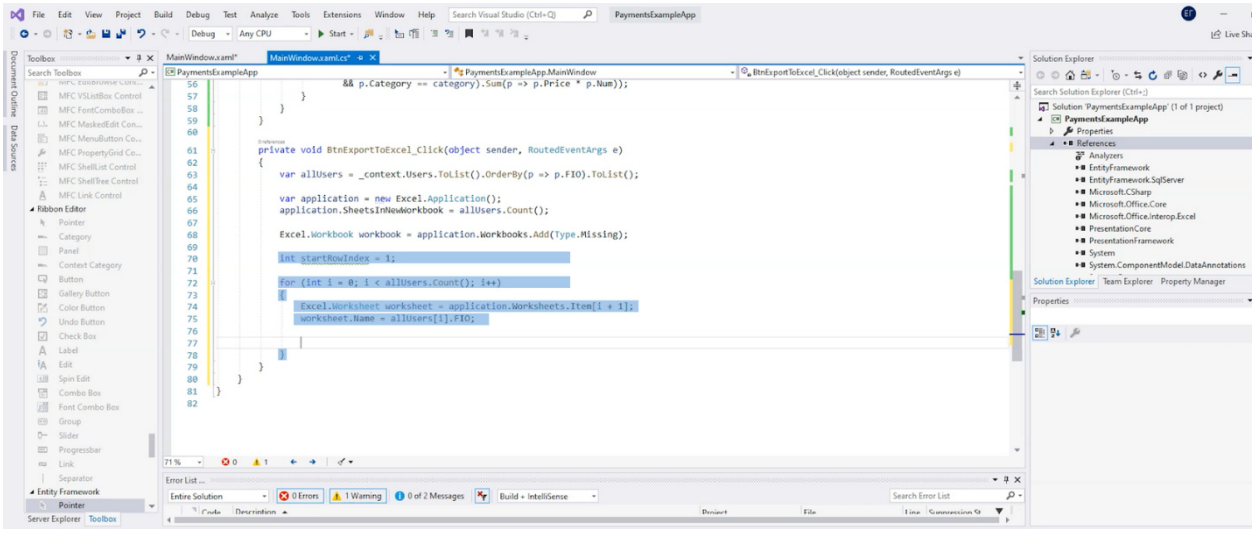
2. Создаем новую книгу Excel



Важно

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

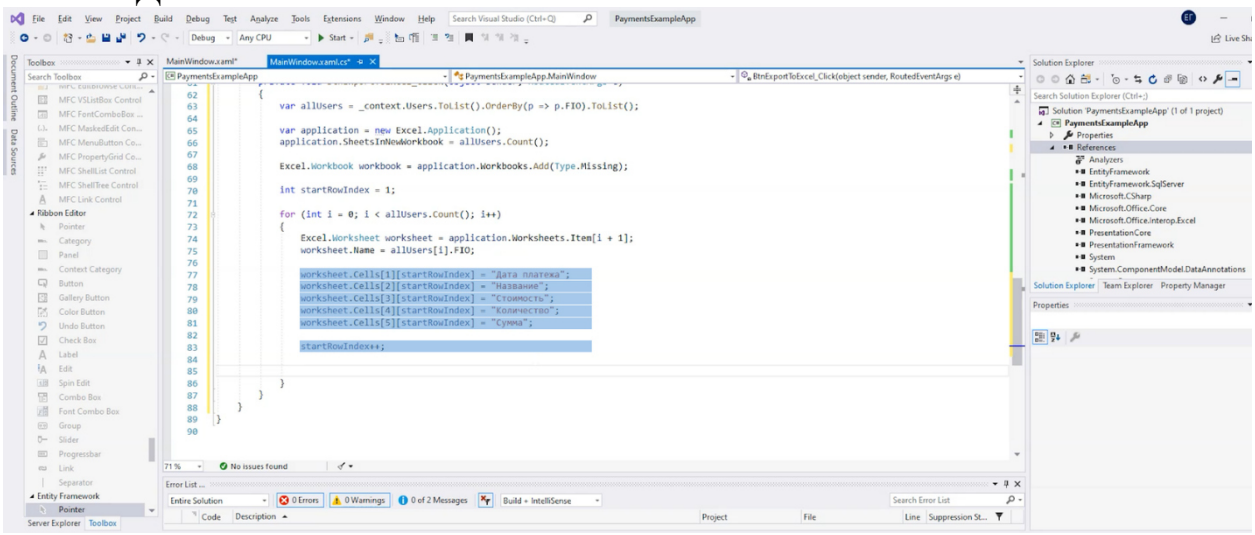
3. Называем листы



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

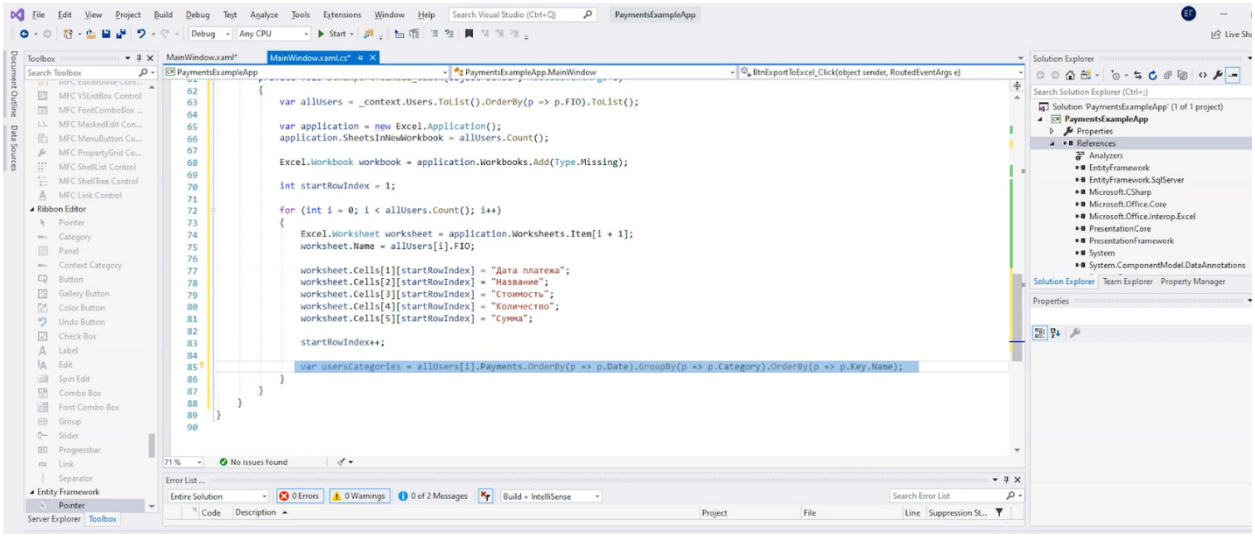
4. Добавляем название колонок



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

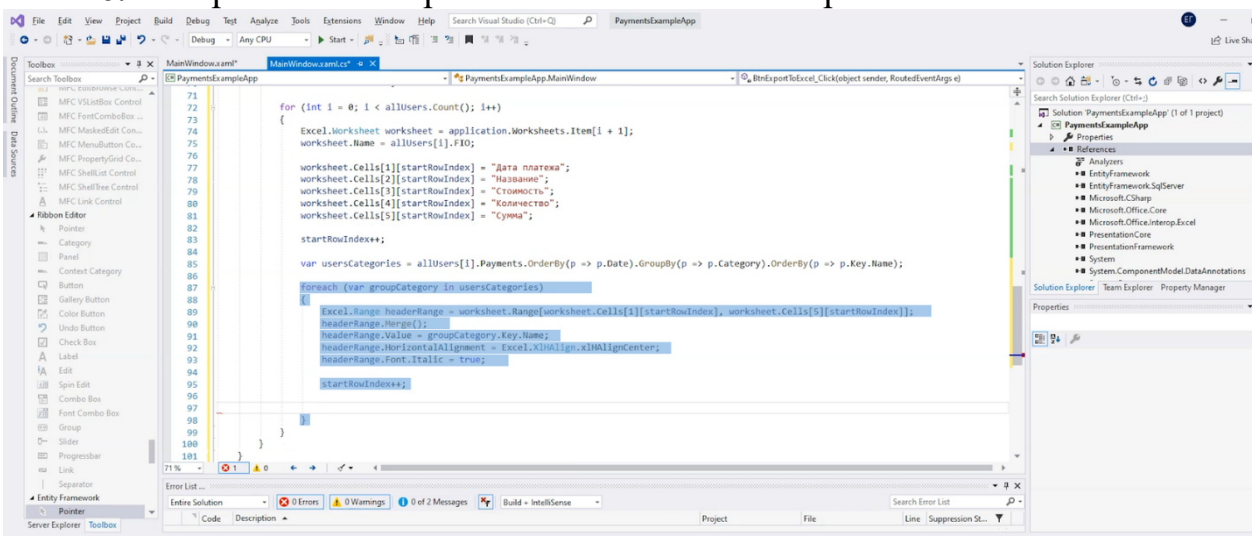
5. Сгруппируем платежи по категориям



Важно

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

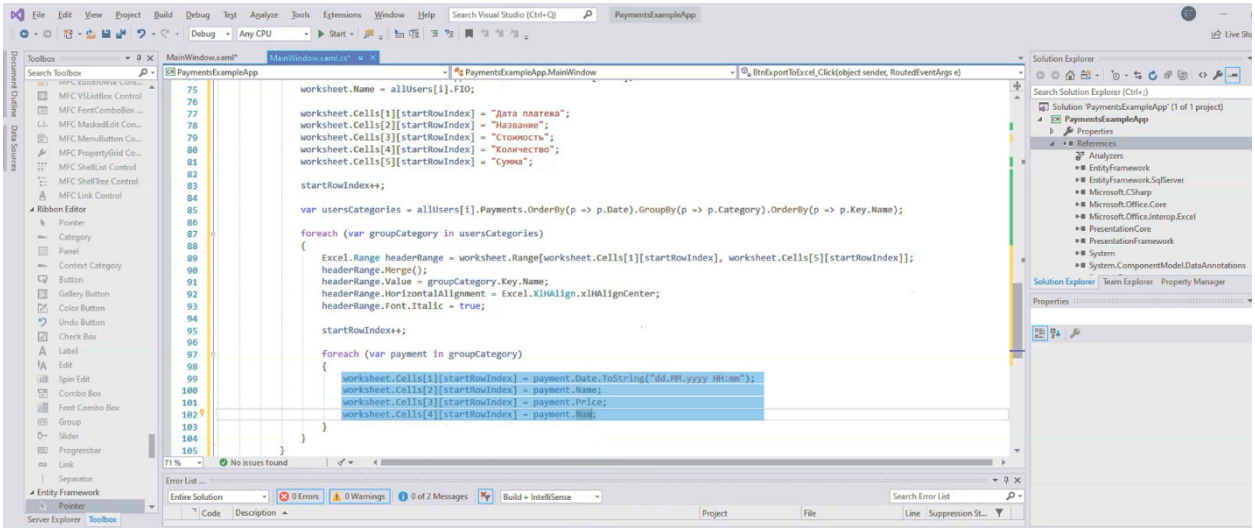
6. Настраиваем отображение названий категорий



Важно

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

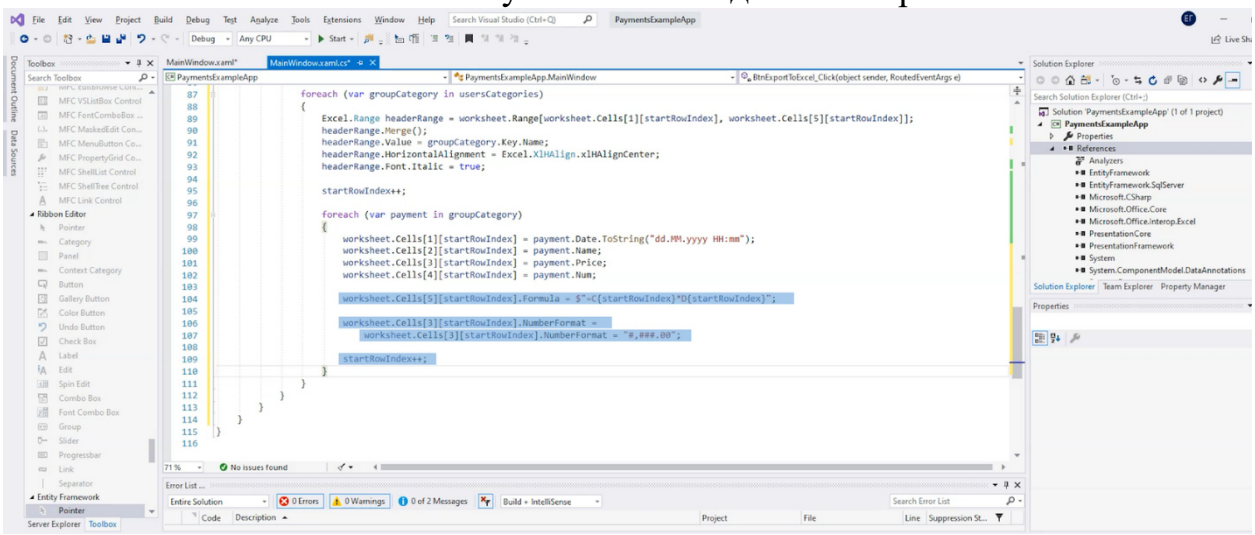
7. Добавляем информацию о платежах



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

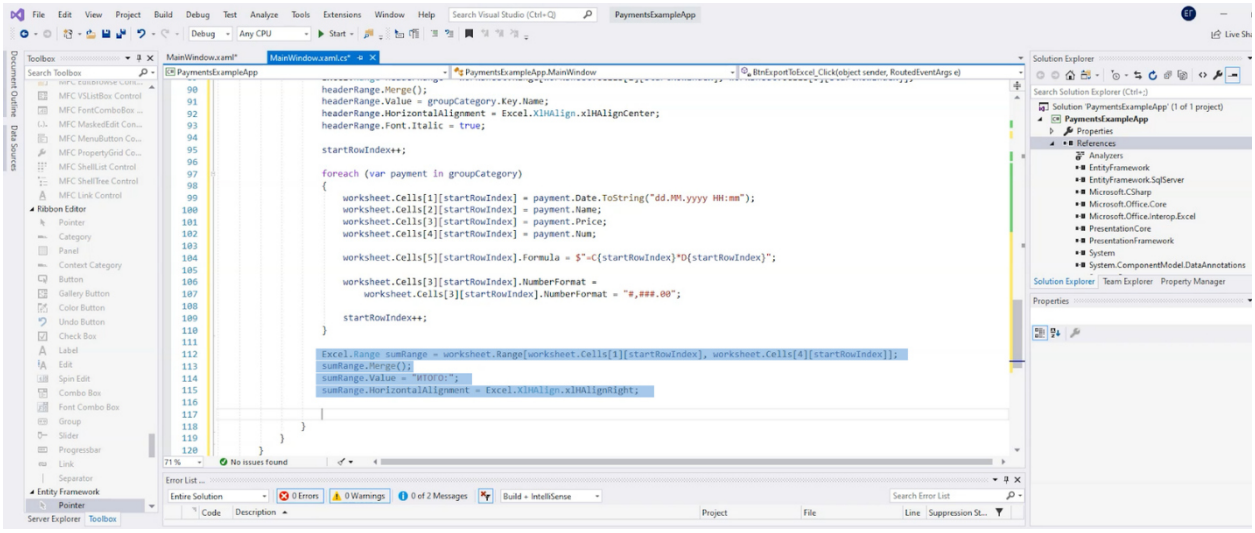
8. Рассчитываем величину платежа каждой категории



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

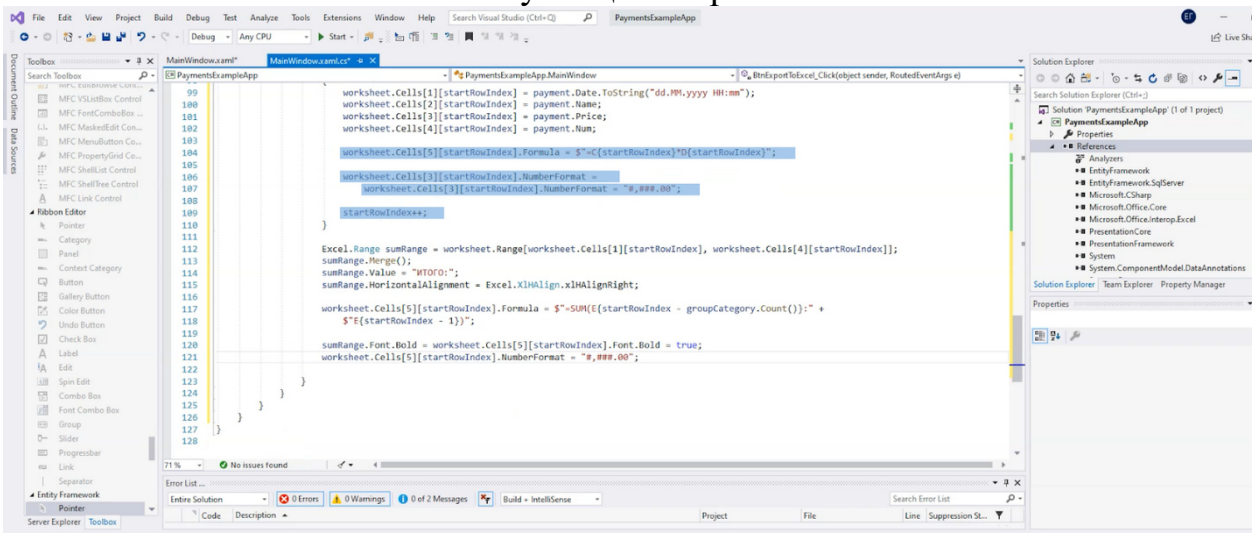
9. Добавляем название к ячейкам для хранения общих затрат



Важно

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

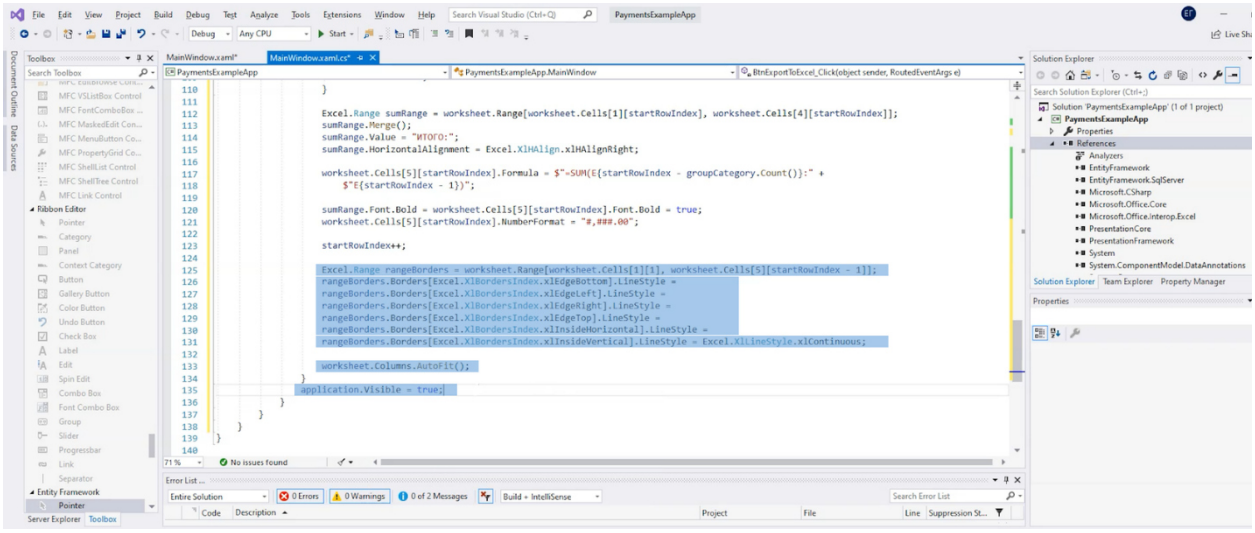
10. Рассчитываем величину общих затрат



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

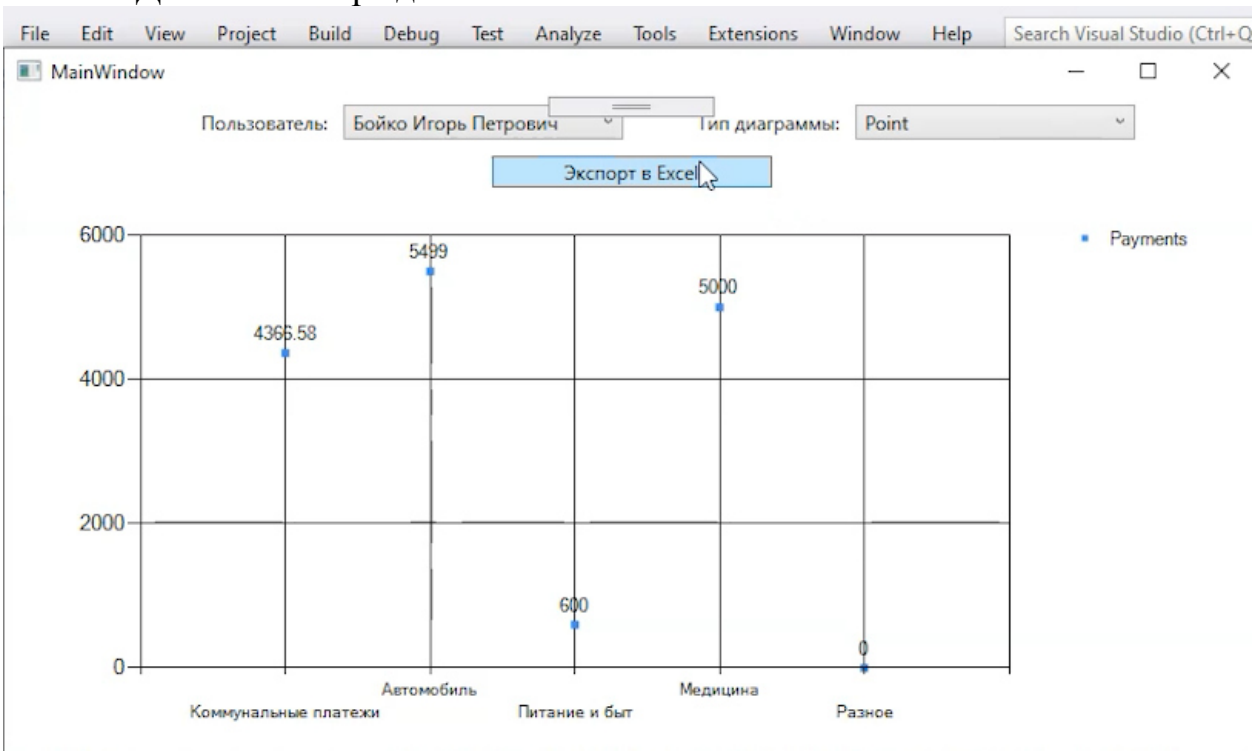


Важно

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

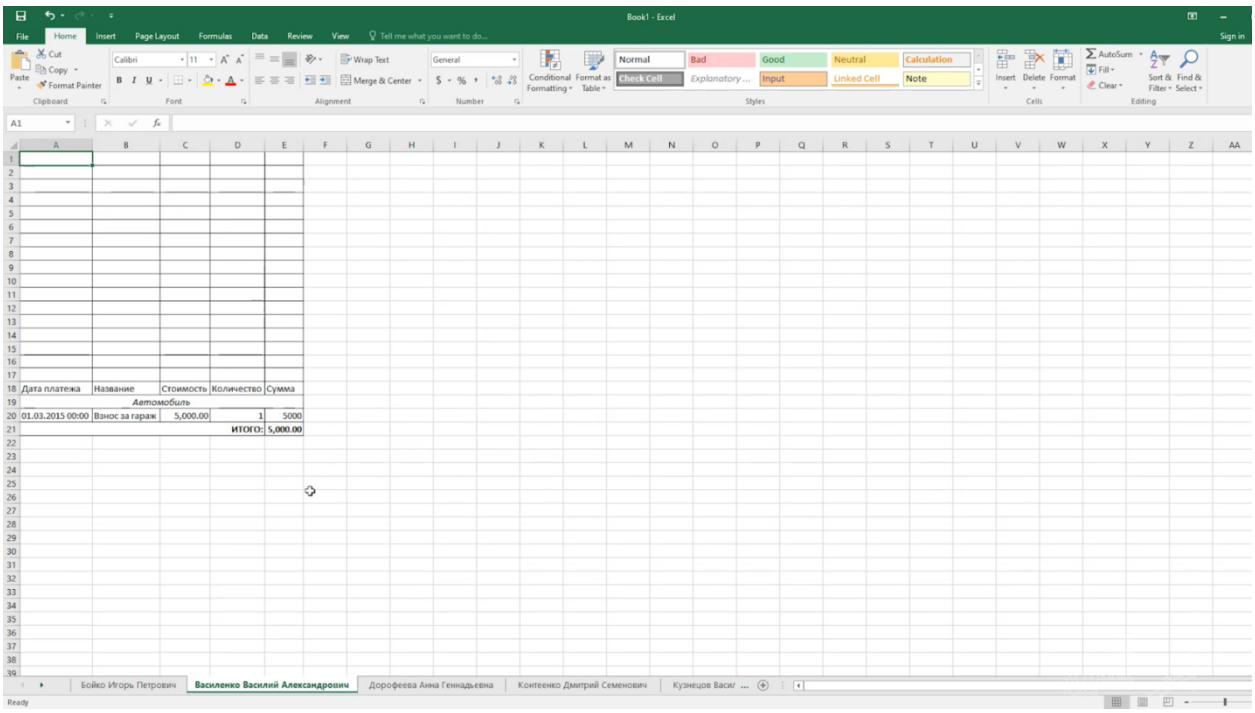
1. Делаем экспорт данных



Важно

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

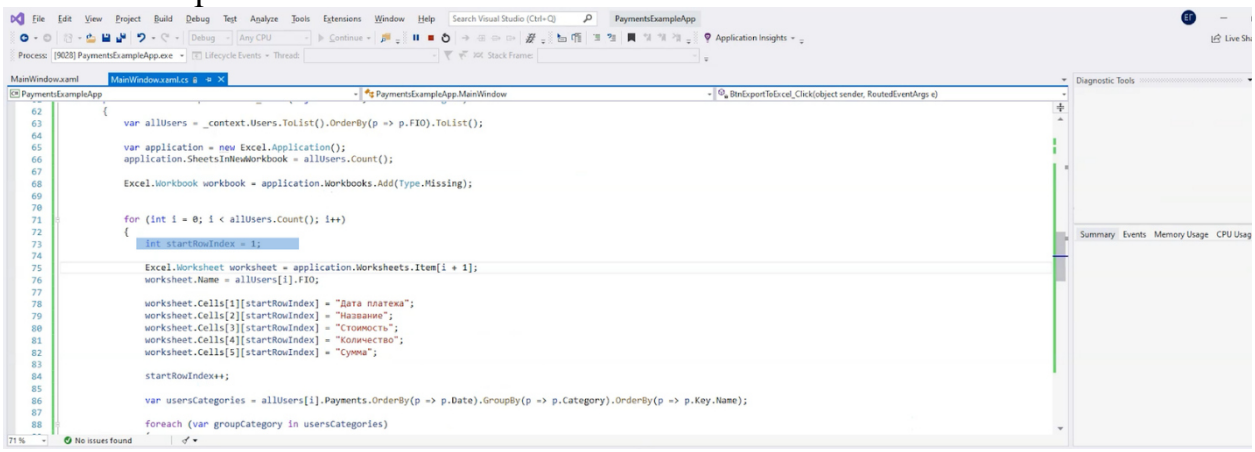
2. Проводим анализ корректности работы



Важно

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

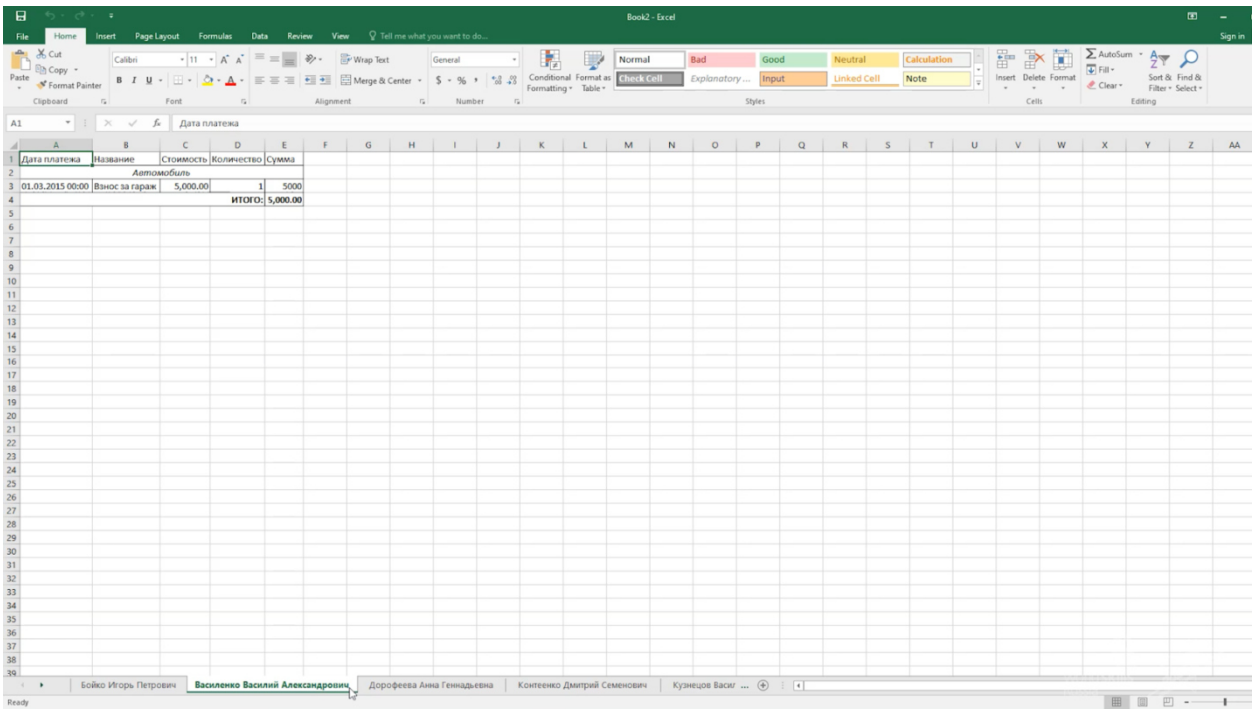
3. Устраняем ошибки



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



Интерактивное задание:
https://nationalteam.worldskills.ru/skills/programmnaya-rabota-s-tablitsami-excel-s-pomoshchyu-biblioteki-microsoft-office-interop-excel/

Практическая работа № 18. Программная работа с документами Word с помощью библиотеки Microsoft.Office.Interop.Word

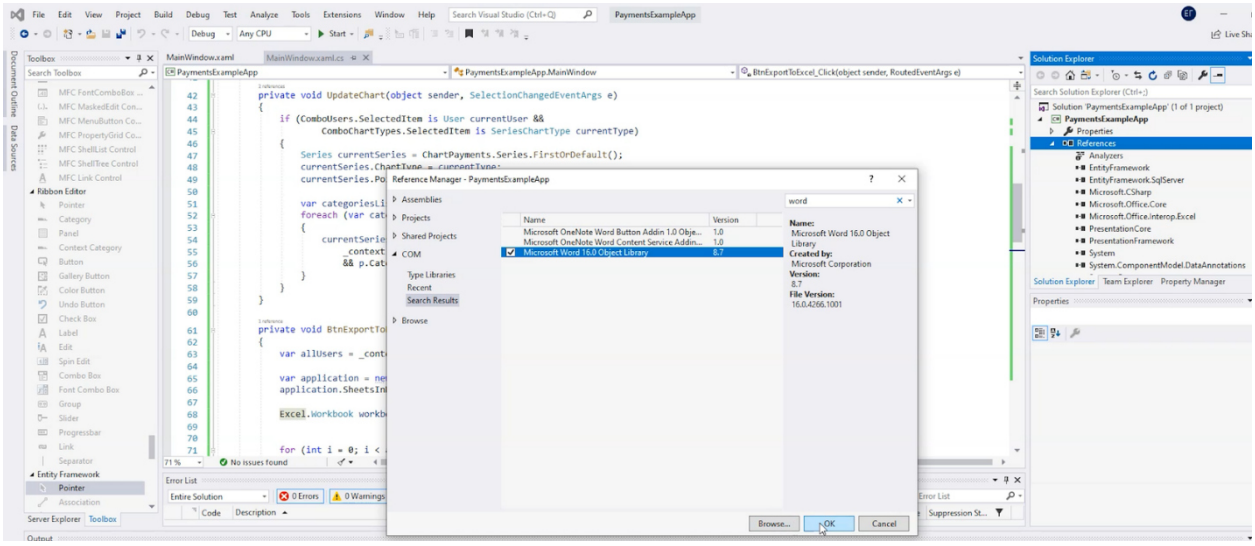
Демонстрация работы с документами Word в WPF

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

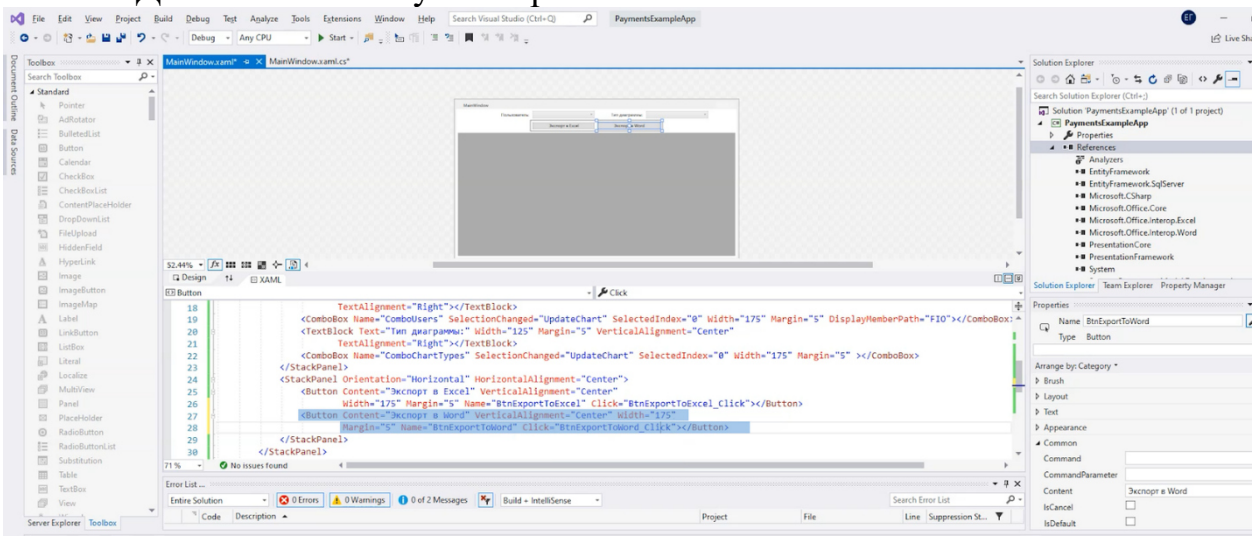
1. Подключаем библиотеку для работы с Word



Важно

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

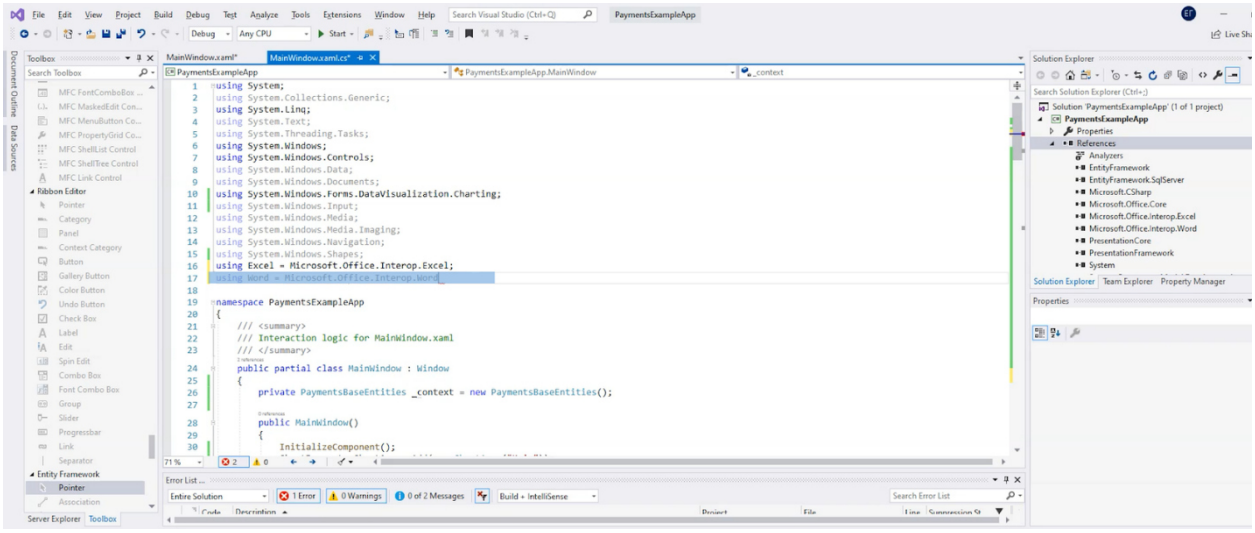
2. Добавляем кнопку экспорта



Важно

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

3. Подключаем пространство имен для работы с Word

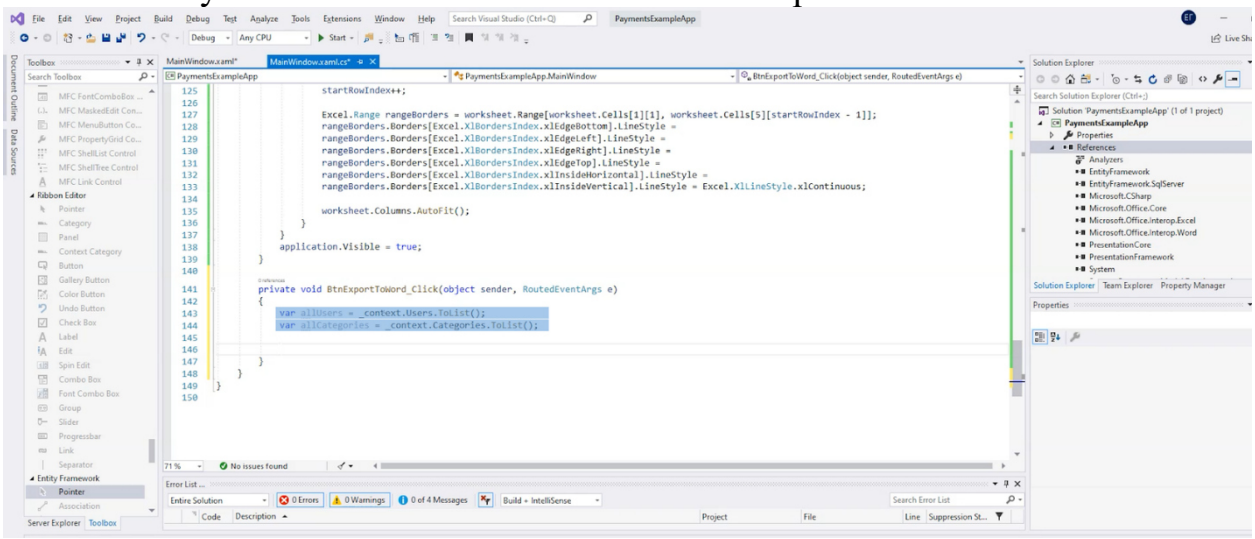


Важно

Требуемое пространство имен подключается с помощью директивы using

Реализация экспорта в документ Word

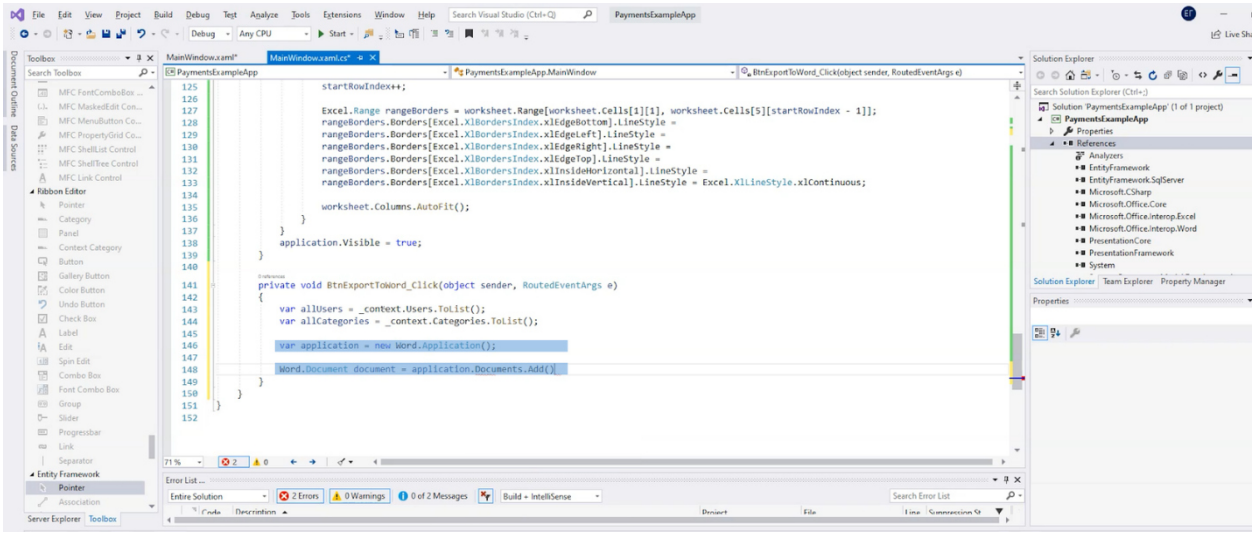
1. Получаем список пользователей и категорий



Важно

Список пользователей и категорий выгружается из базы данных

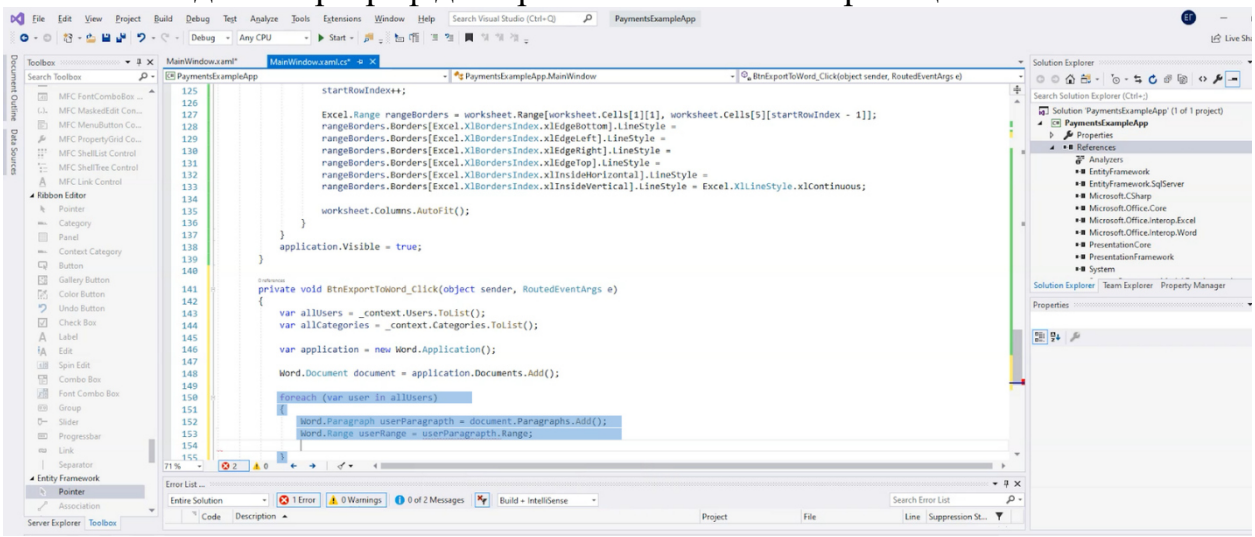
2. Создаем новый документ Word



Важно

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

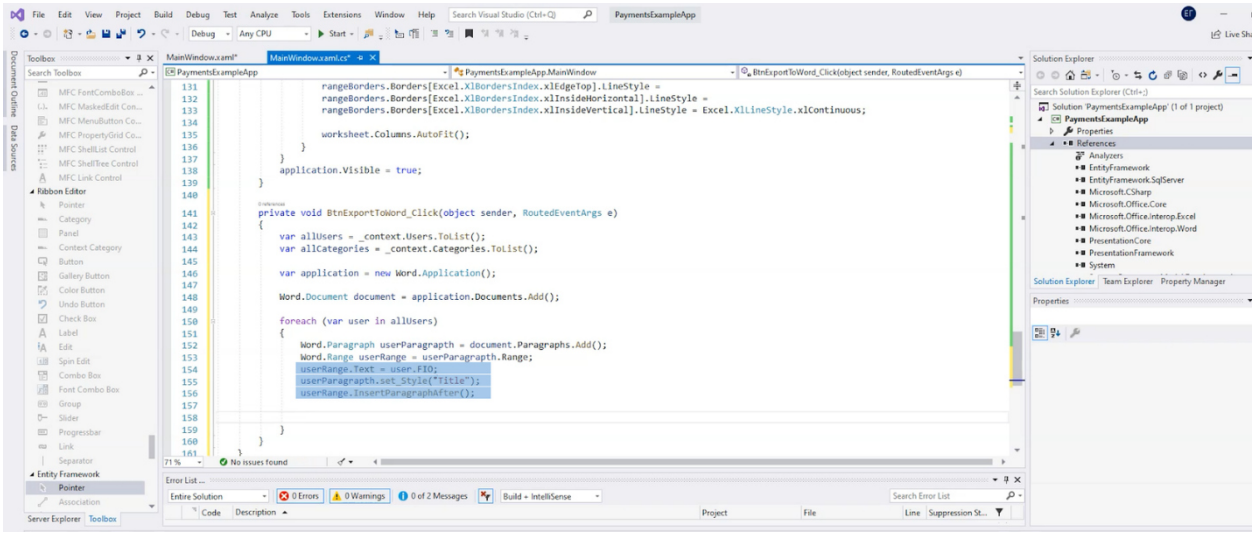
3. Создаем параграф для хранения названий страниц



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

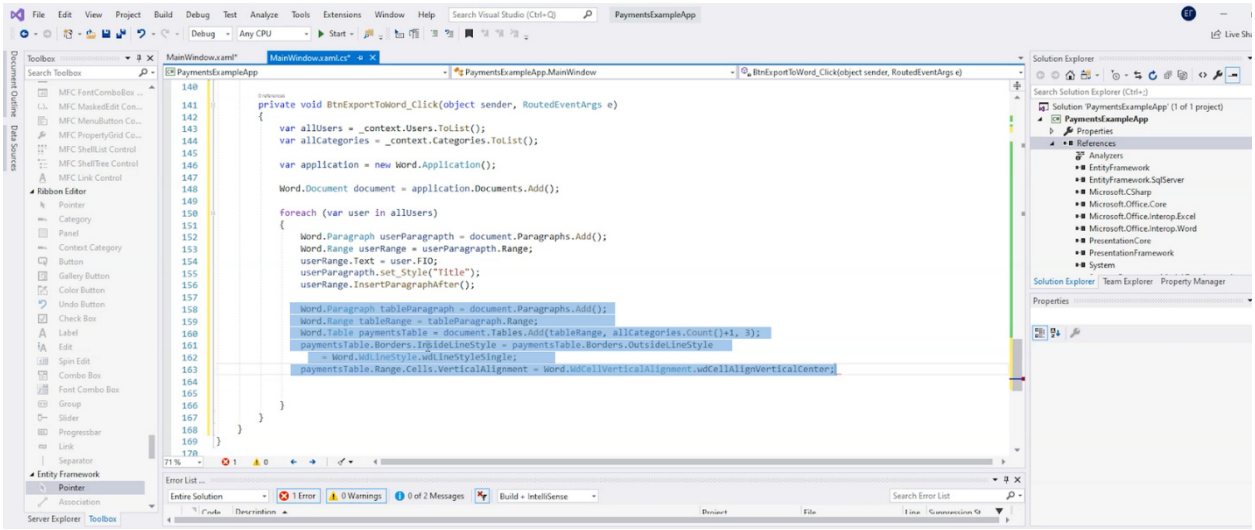
4. Добавляем названия страниц



Важно

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

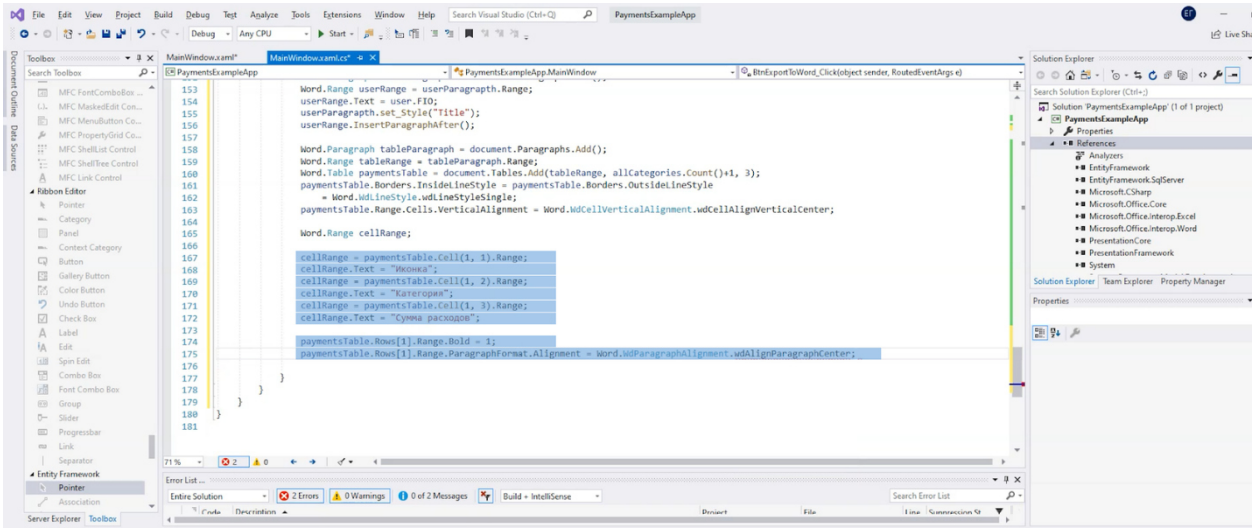
5. Добавляем и форматируем таблицу для хранения информации о платежах



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)

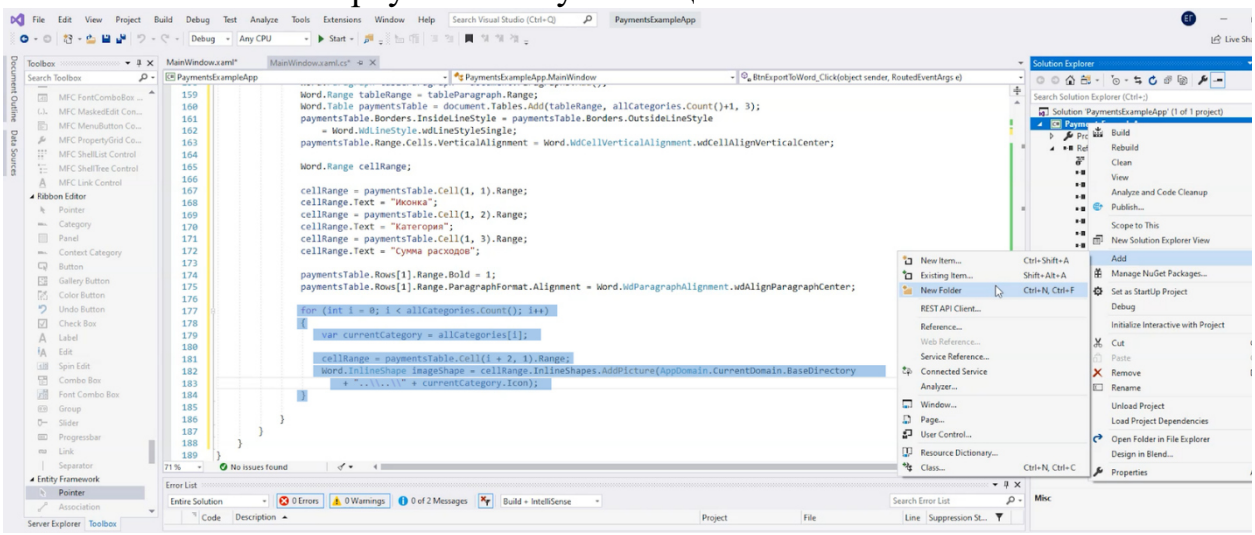
6. Добавляем названия колонок и их форматирование



Важно

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

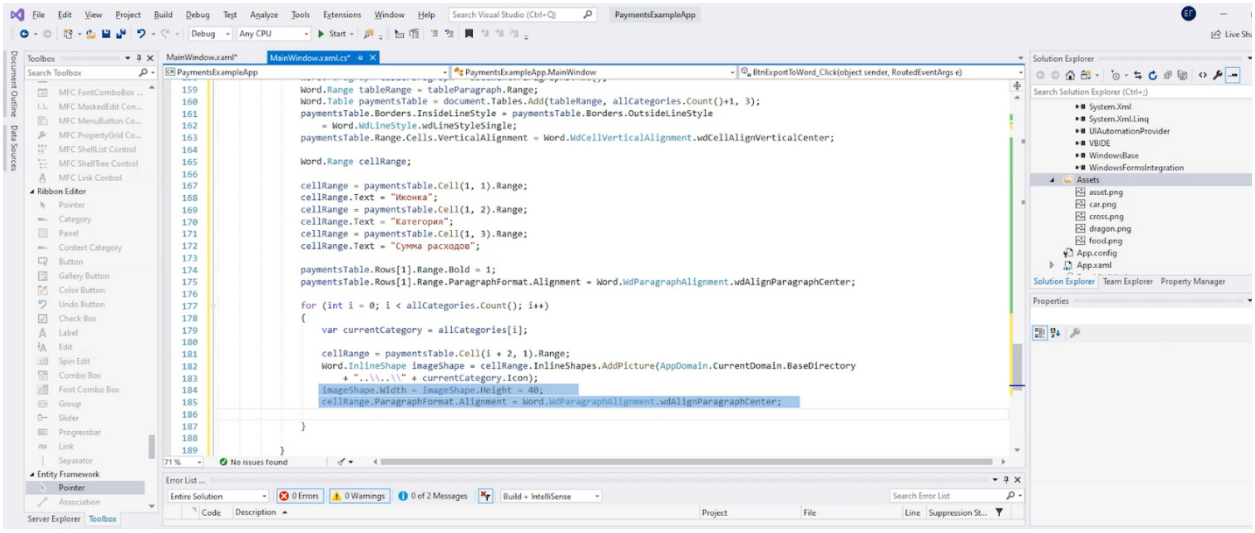
7. Заполняем первую колонку таблицы



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

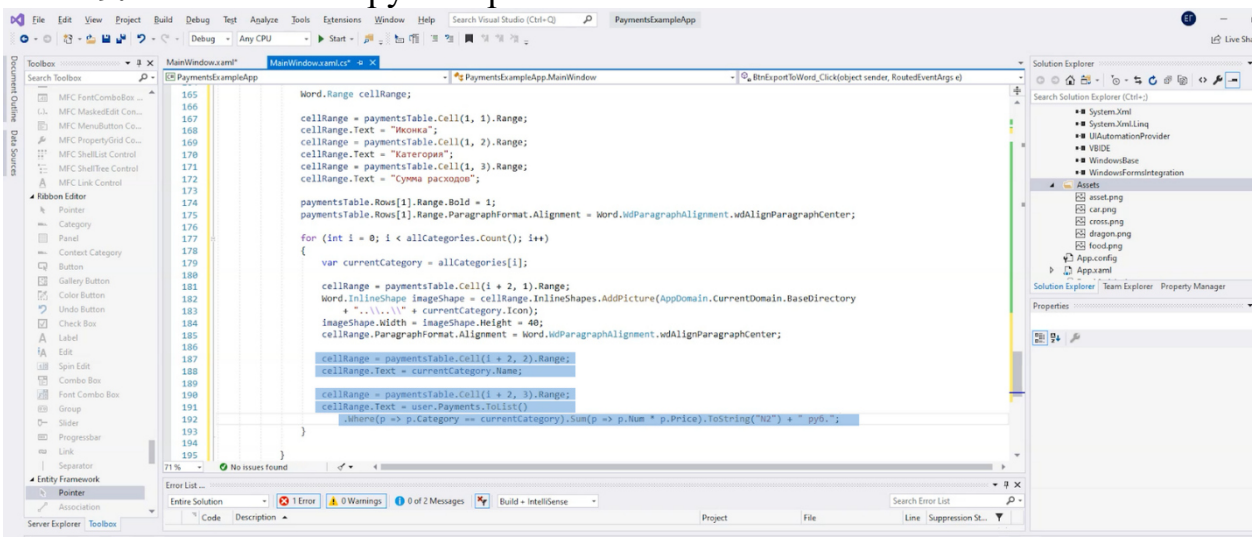
8. Форматируем первую колонку таблицы



Важно

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

9. Заполняем вторую и третью колонки

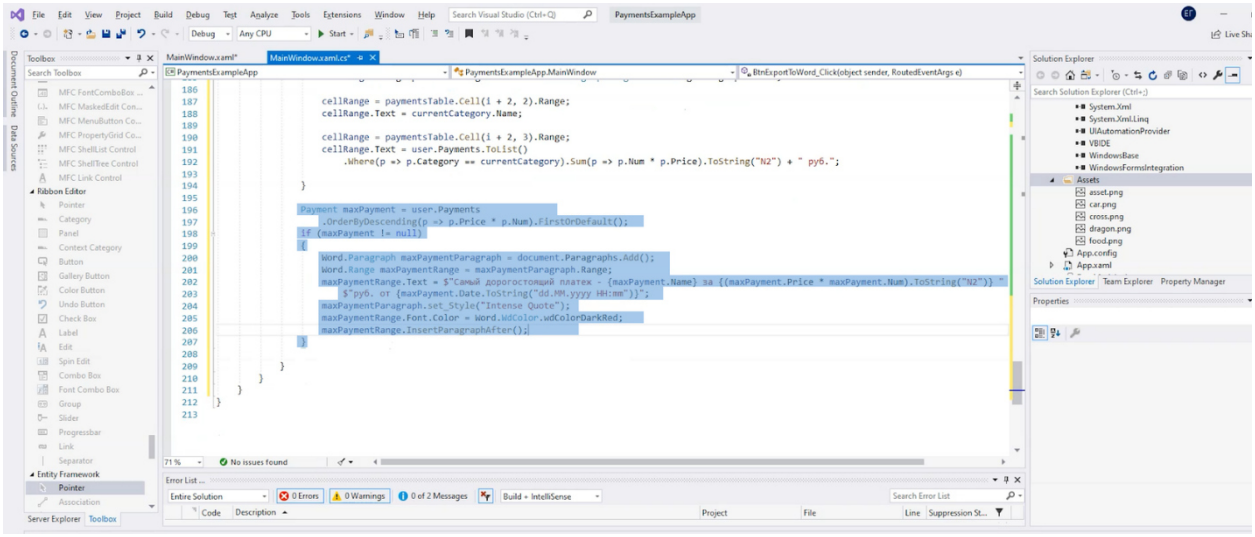


Важно

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

Завершение оформления документа Word

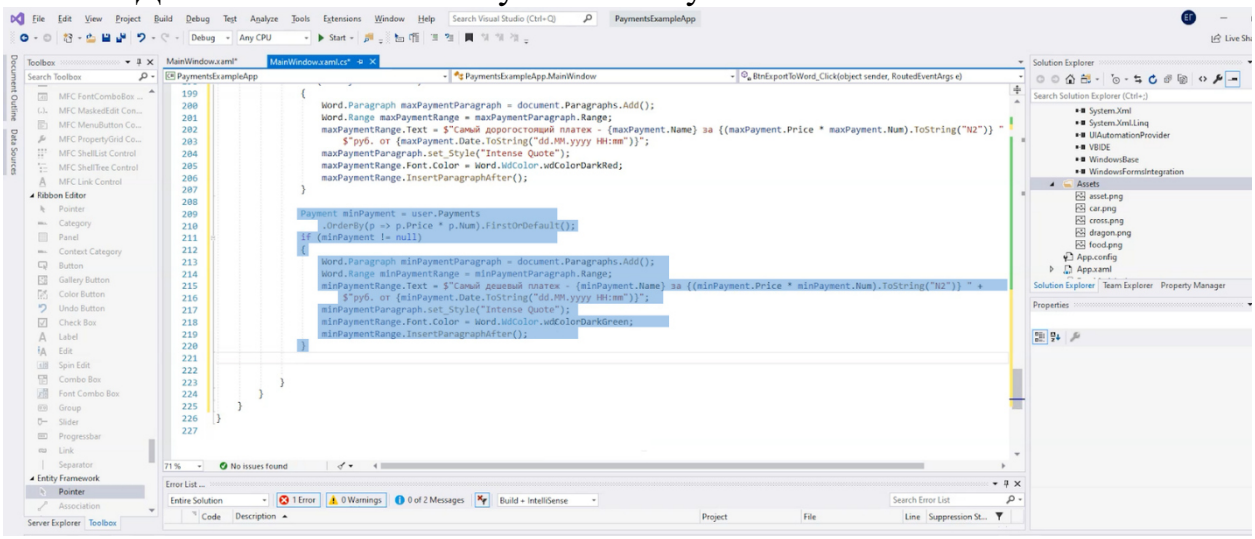
1. Добавляем максимальную величину платежа



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)

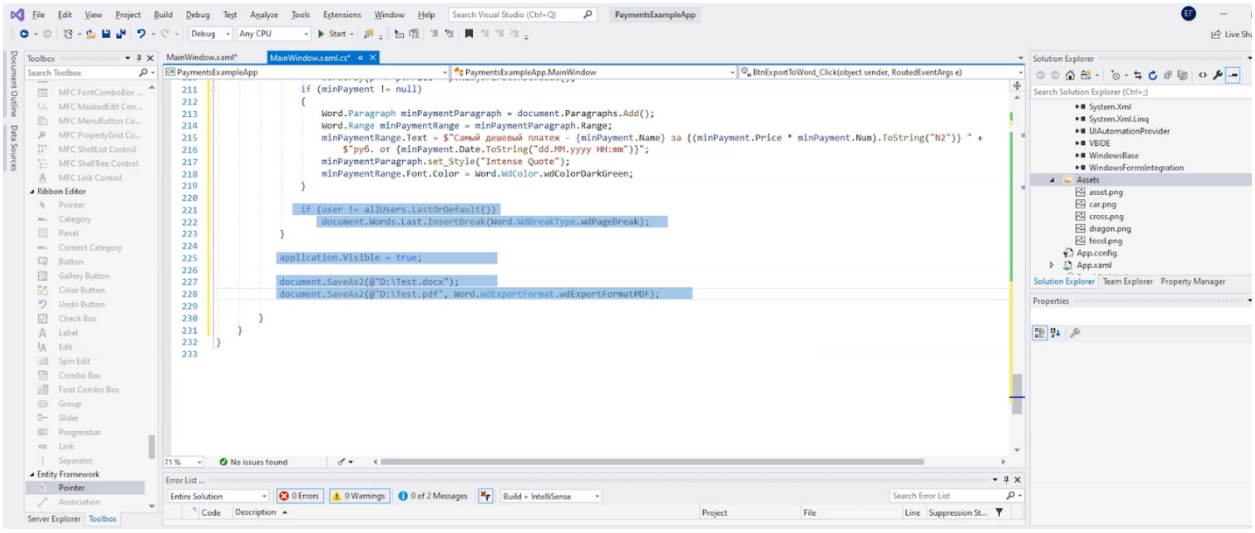
2. Добавляем минимальную величину платежа



Важно

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

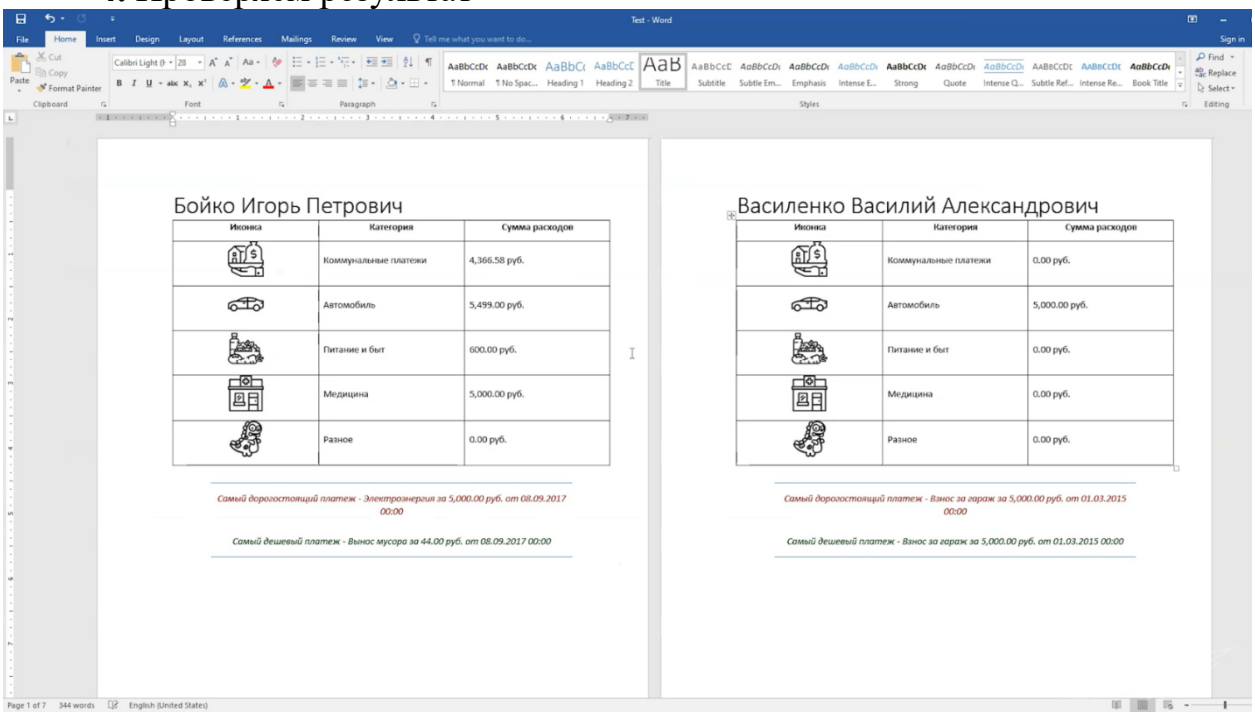
3. Делаем заключительные шаги



Важно

По завершении работы с данными пользователя добавляется разрыв страницы. Далее, разрешается отображение таблицы по завершении экспорта. Наконец, документ сохраняется в формате .docx и .pdf

4. Проверяем результат



Интерактивное задание:
https://nationalteam.worldskills.ru/skills/programmnaya-rabota-s-dokumentami-word-s-pomoshchyu-biblioteki-microsoft-office-interop-word/

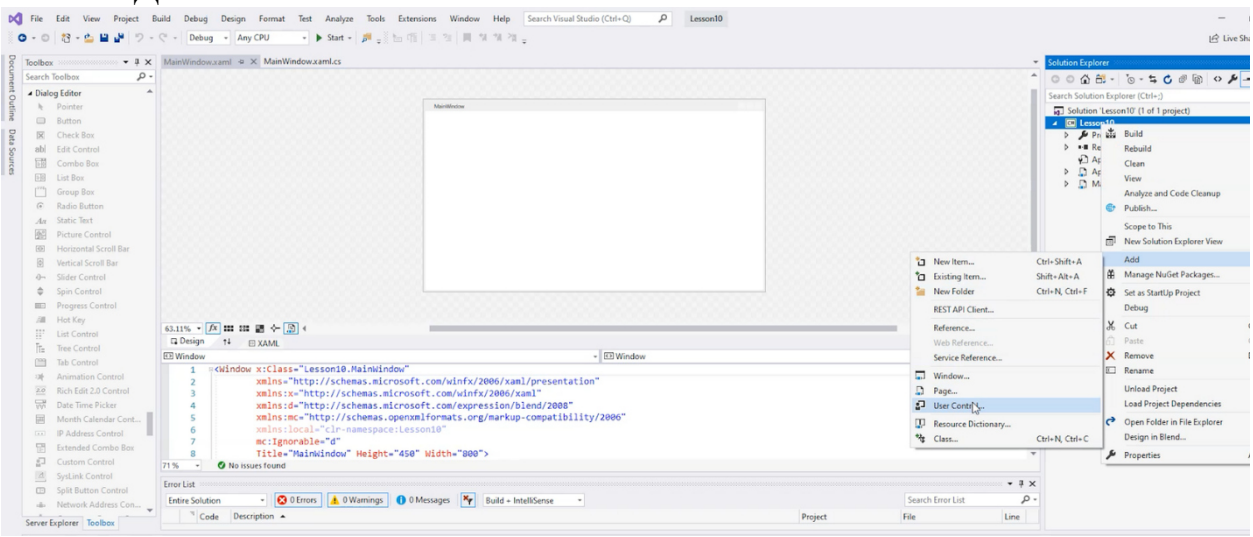
Практическая работа № 19. Реализация пользовательских элементов управления (UserControl)

На данном занятии будет разработан пользовательский элемент управления (User Control). User Control предоставляет простой способ создания собственных элементов управления и является объединением нескольких уже существующих элементов управления в один элемент, допускающий простое повторное использование. В данном примере разрабатывается элемент ромб, внутри которого будет находиться изображение (требуется несколько таких элементов). По умолчанию демонстрируется черно-белое изображение, при наведении курсора на изображение демонстрируется его цветная версия. Под изображением располагается заголовок и описание, разделенные линией. При нажатии на изображение демонстрируется текст. Основные шаги реализации элемента User Control:

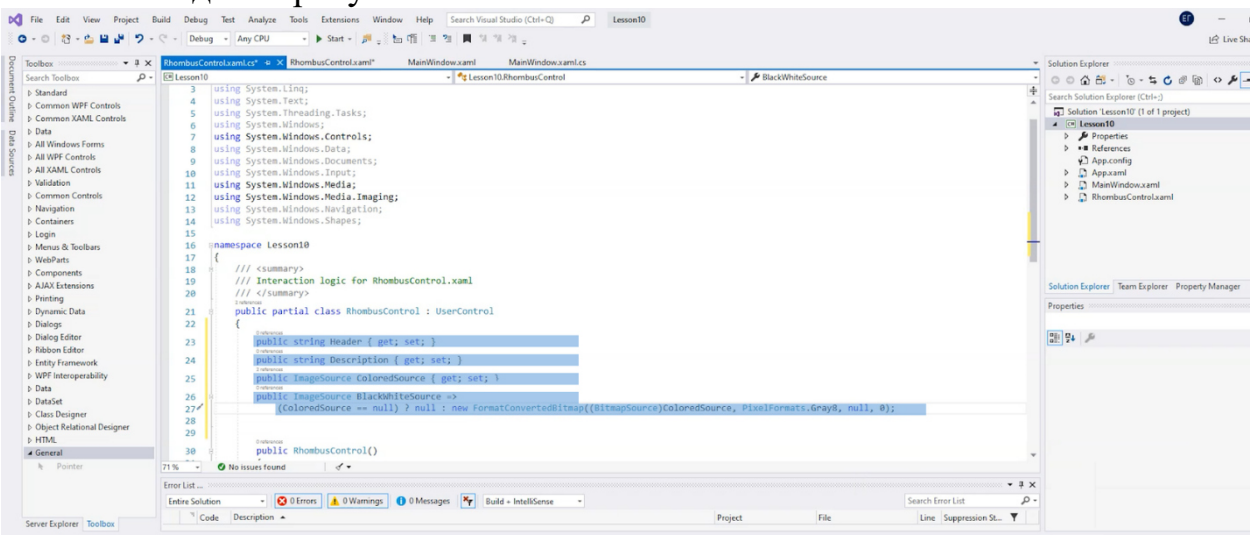
1. Создание
2. Реализация
3. Использование

Создание элемента User Control

1. Добавляем элемент User Control



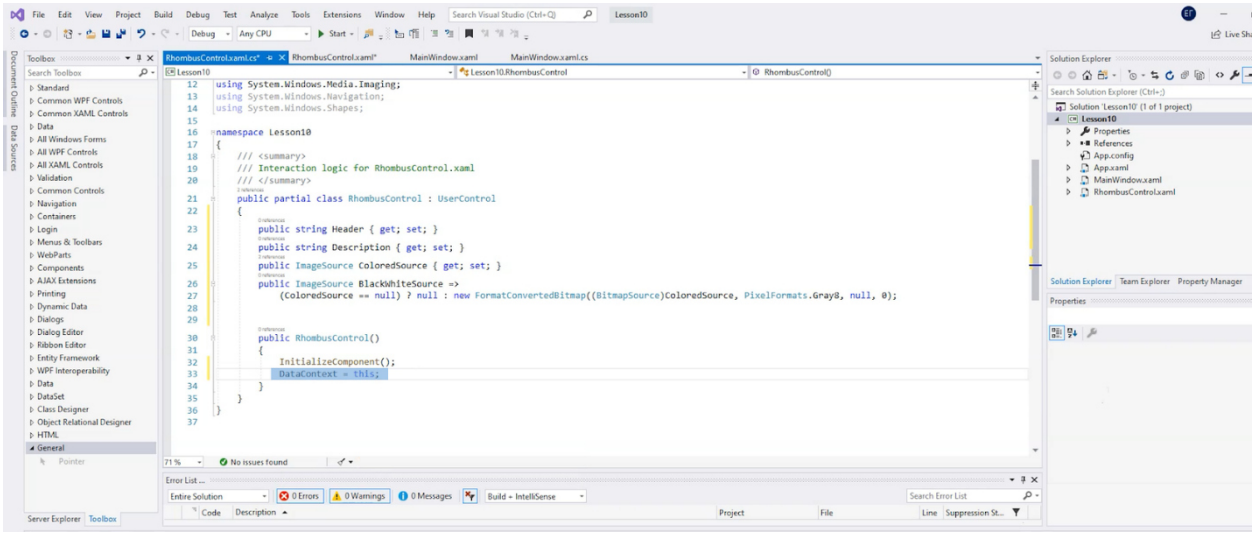
2. Создаем требуемые свойства элемента



Важно

К свойствам элемента относятся: заголовок, описание, цветная версия изображения, а также черно-белая версия, которая будет конвертироваться из цветного изображения

3. Устанавливаем DataContext



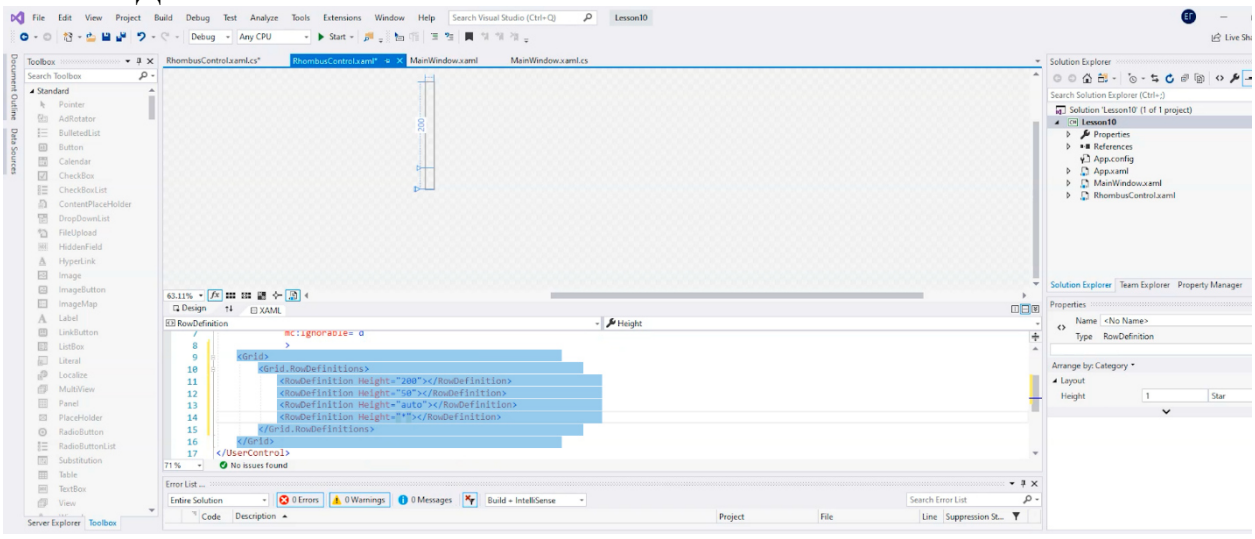
```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Lesson10
{
    /// <summary>
    /// Interaction logic for RhombusControl.xaml
    /// </summary>
    public partial class RhombusControl : UserControl
    {
        public string Header { get; set; }
        public string Description { get; set; }
        public ImageSource ColoredSource { get; set; }
        public ImageSource BlackWhiteSource =>
            (ColoredSource == null) ? null : new FormatConvertedBitmap((BitmapSource)ColoredSource, PixelFormats.Gray8, null, 0);


        public RhombusControl()
        {
            InitializeComponent();
            DataContext = this;
        }
    }
}
```

Важно

Установка DataContext (контекста данных) позволяет делать привязку элементов разметки к свойствам

Реализация элемента User Control

1. Добавляем элемент Grid

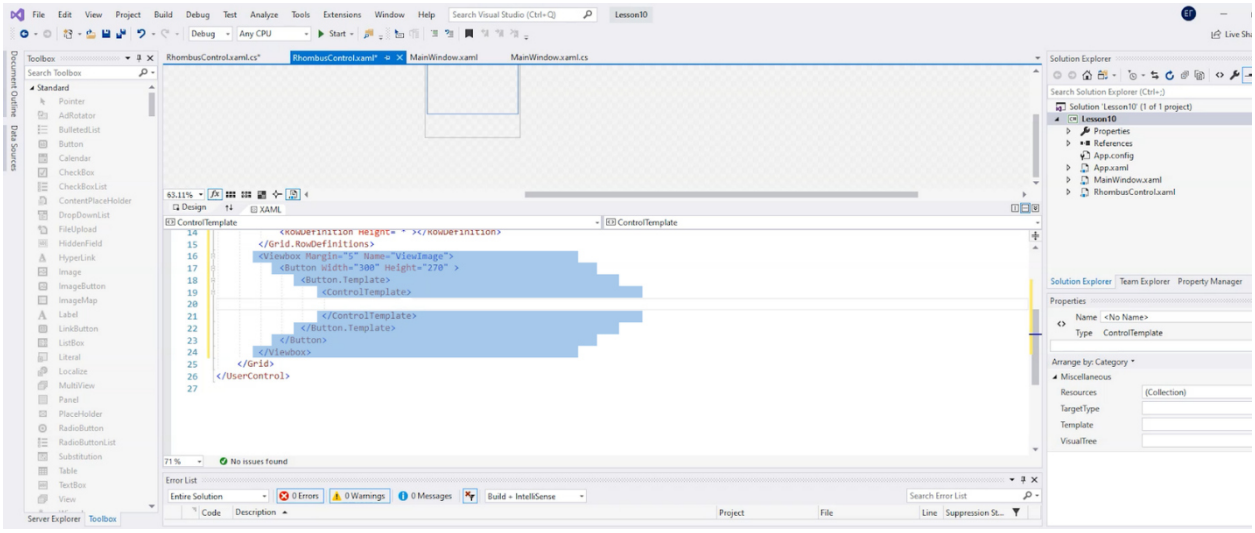


```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="200"></RowDefinition>
        <RowDefinition Height="50"></RowDefinition>
        <RowDefinition Height="auto"></RowDefinition>
        <RowDefinition Height="*"></RowDefinition>
    </Grid.RowDefinitions>
</Grid>
</UserControl>
```

Важно

Grid (сетка) разделяется на четыре строки заданной высоты. Обратите внимание на рабочую область, где демонстрируется вид приложения

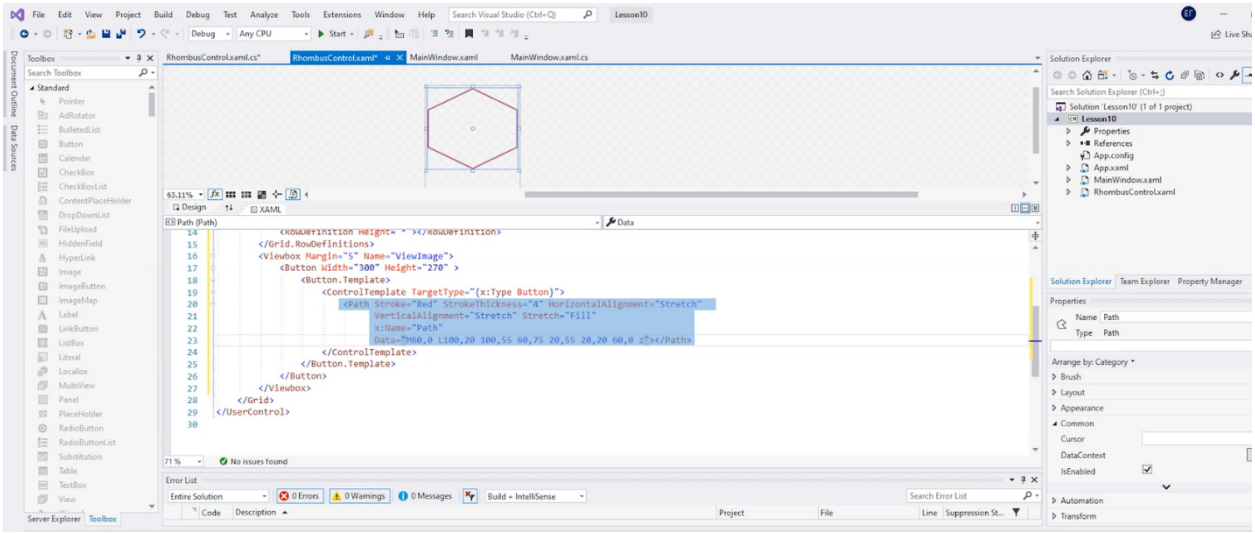
2. Настраиваем представление основного элемента-ромба



Важно

В первой строке Grid будет располагаться ромб. Элемент ViewBox позволит растягивать ромб с изображением по размеру контейнера. Также внутри ViewBox будет находиться кнопка с переопределенным template (шаблоном)

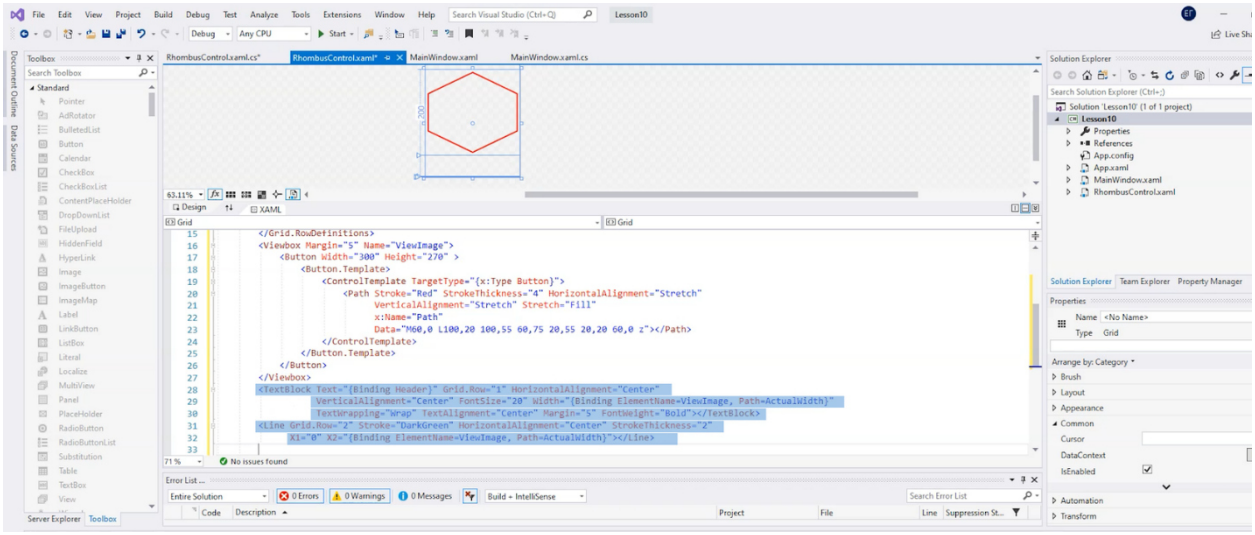
3. Добавляем содержимое кнопки внутри ромба



Важно

Кнопка представляет собой элемент Path, для которого задаются координаты вершин, толщина и цвет границ

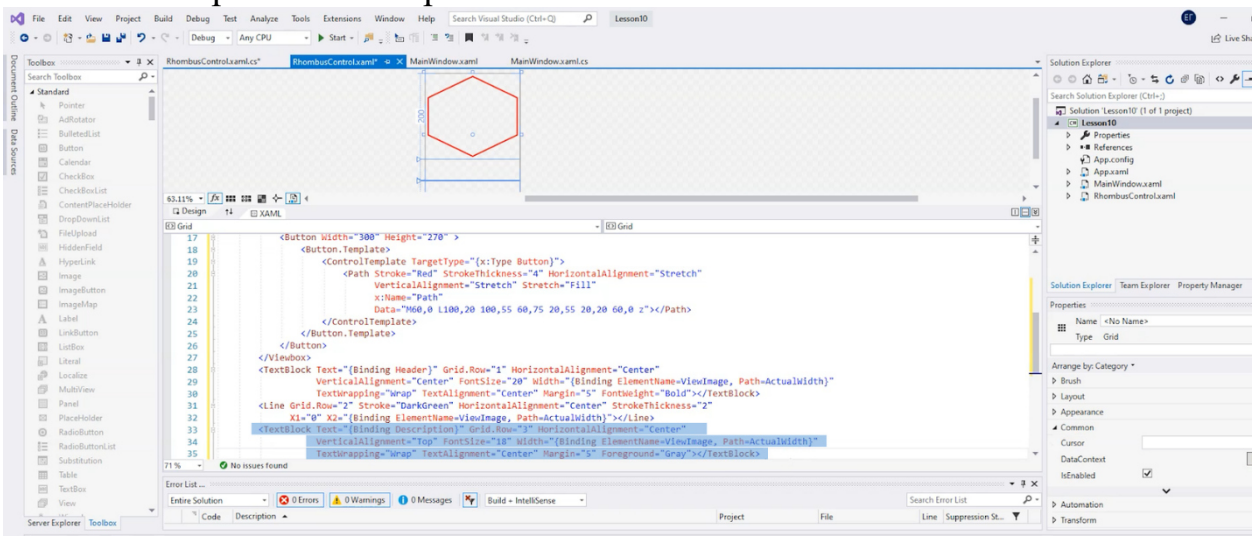
4. Настраиваем отображение заголовка



Важно

Заголовок размещается внутри элемента TextBlock, текст которого привязывается к свойству Header, а ширина — к текущим размерам ViewBox. Зеленая линия цвета DarkGreen добавляется для того, чтобы разделить заголовок и описание

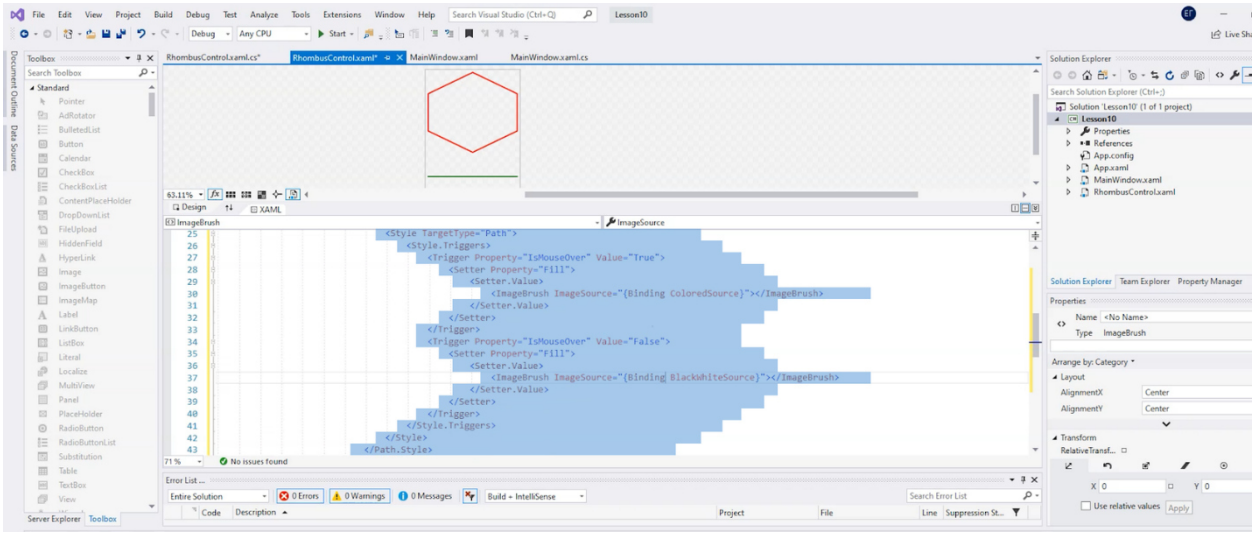
5. Настраиваем отображение описания



Важно

TextBlock для описания добавляется по аналогии с заголовком с тем лишь отличием, что цвет шрифта будет серым

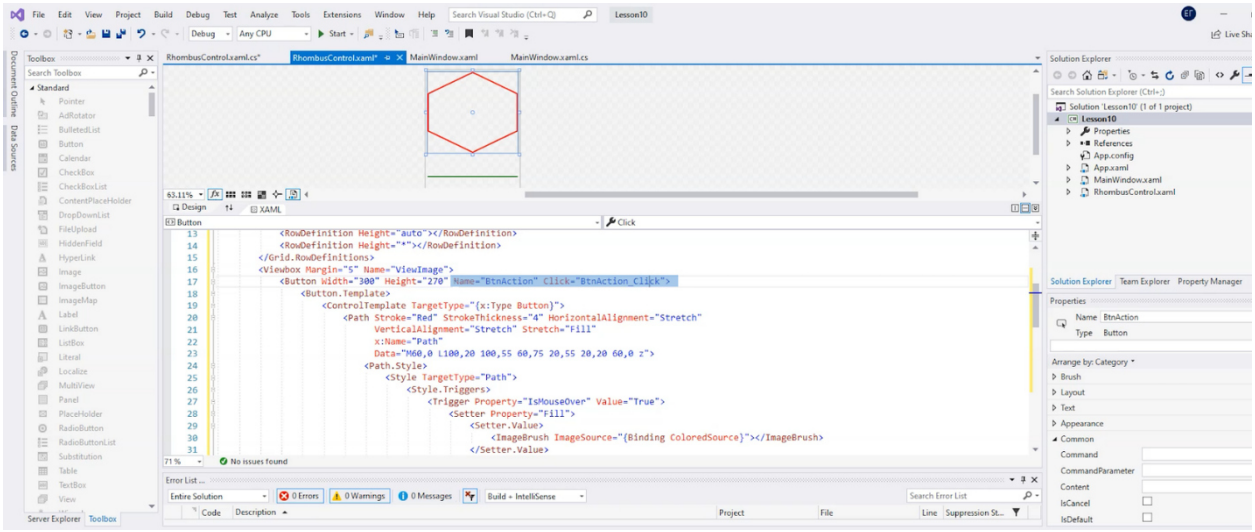
6. Настраиваем поведение изображения



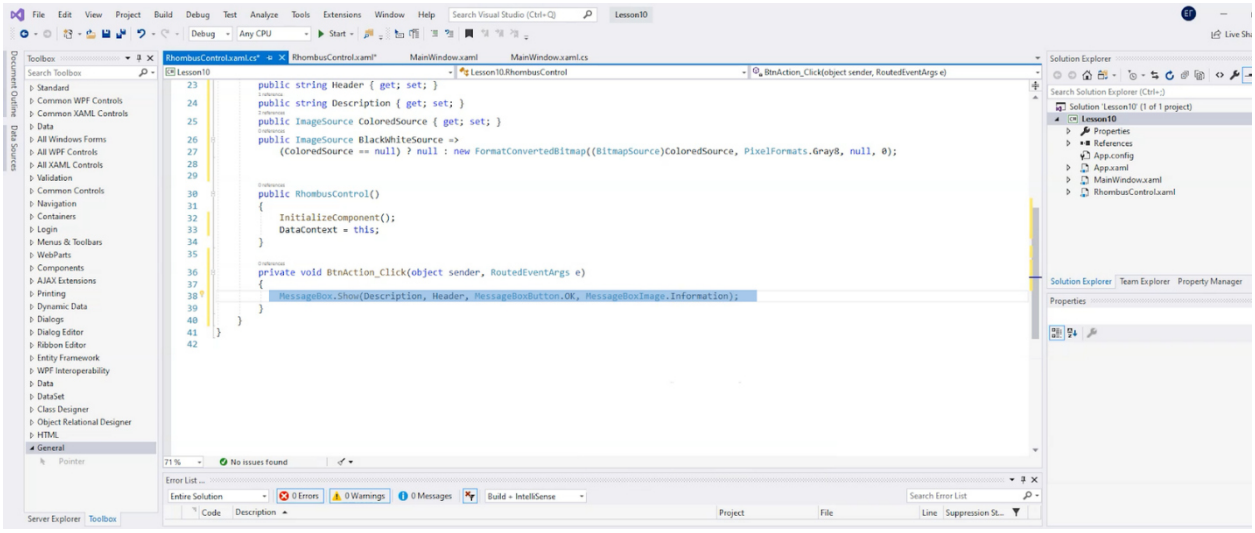
Важно

Поведение изображения будет зависеть от значения свойства IsMouseOver. Если свойство принимает значение true, демонстрируется цветное изображение ColoredSource и черно-белое BlackWhiteSource в противном случае

7. Добавляем кнопку для вывода информации об изображении



8. Реализуем обработку клика

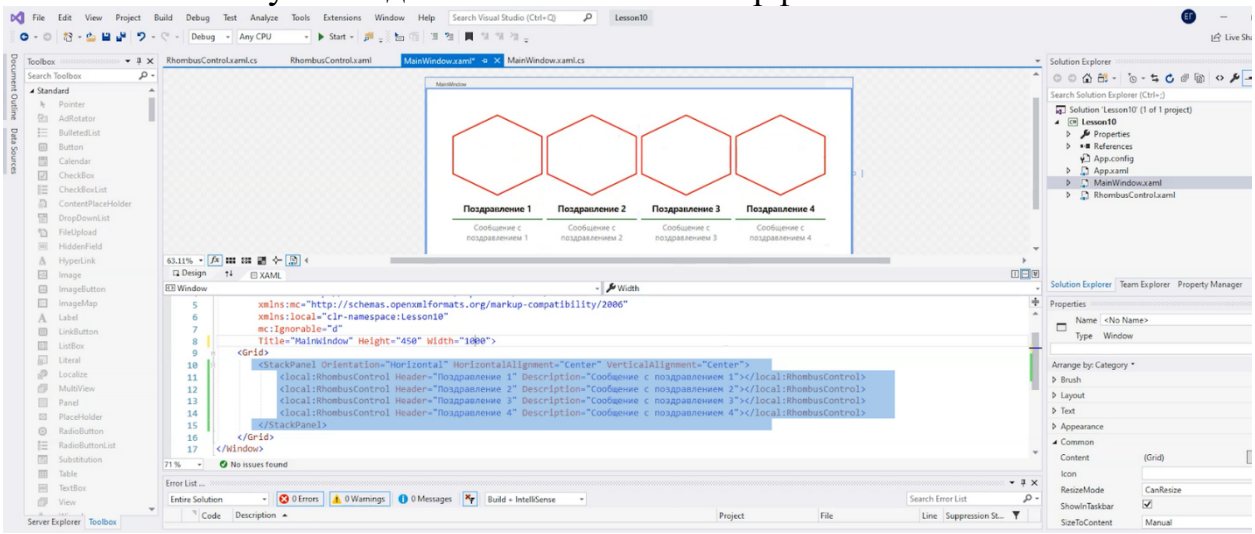


Важно

Обработкой будет служить вывод информации об элементе в MessageBox

Использование User Control

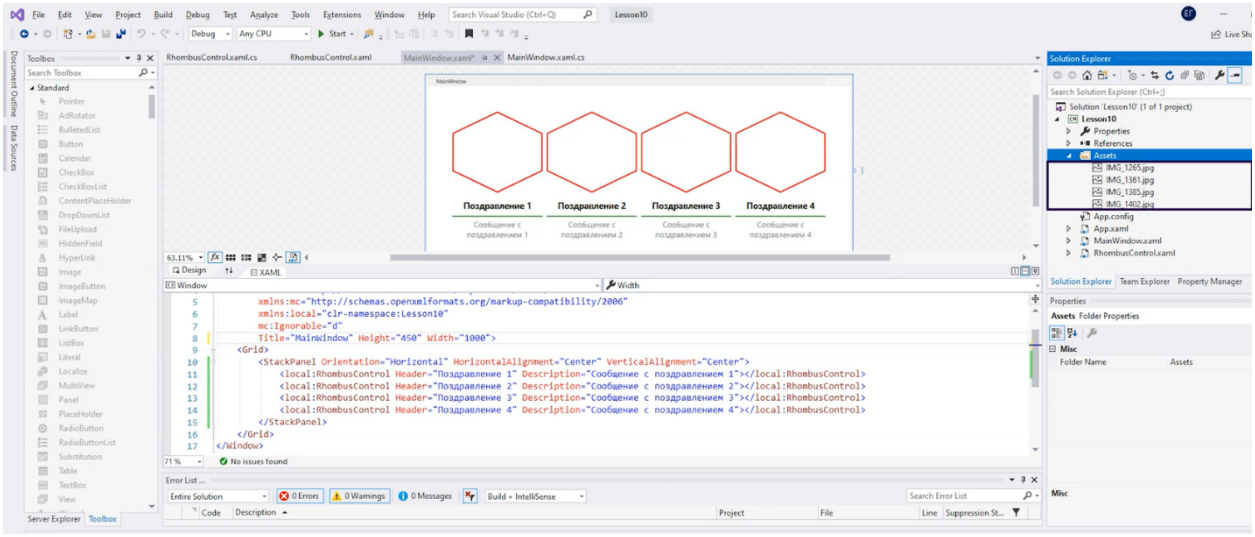
1. Используем созданный элемент в интерфейсе



Важно

Несколько созданных элементов располагаются на MainWindow в StackPanel с горизонтальной ориентацией. Следует обратить внимание, что для использования пользовательского элемента управления используется префикс local

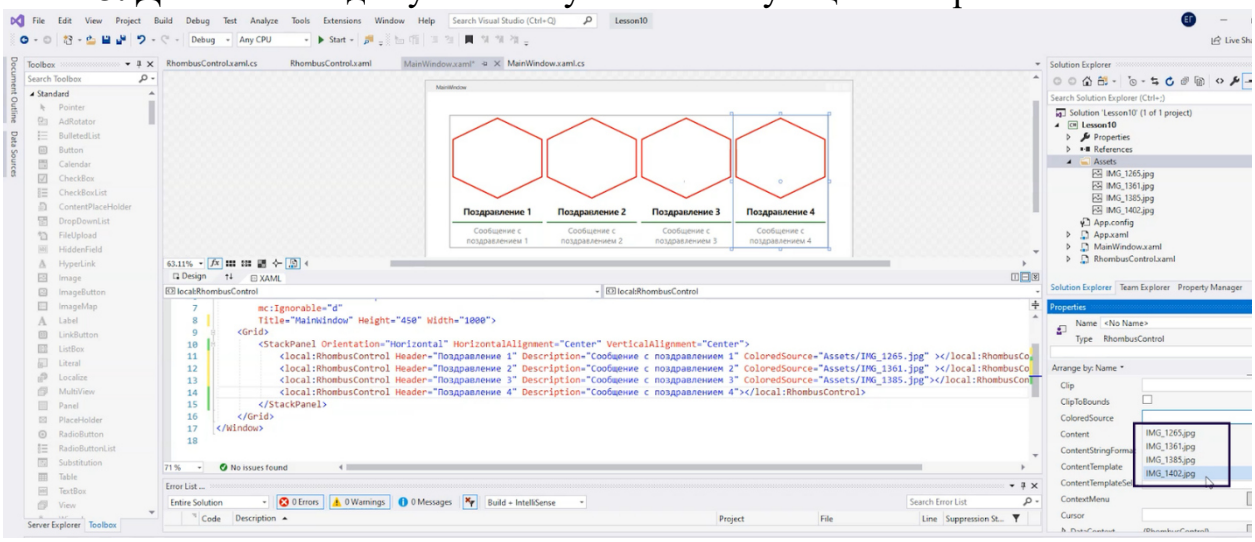
2. Добавим изображения в проект



Важно

Изображения для каждого элемента подгружаются из папки Assets, куда они предварительно добавляются (см. на область внутри черной рамки)

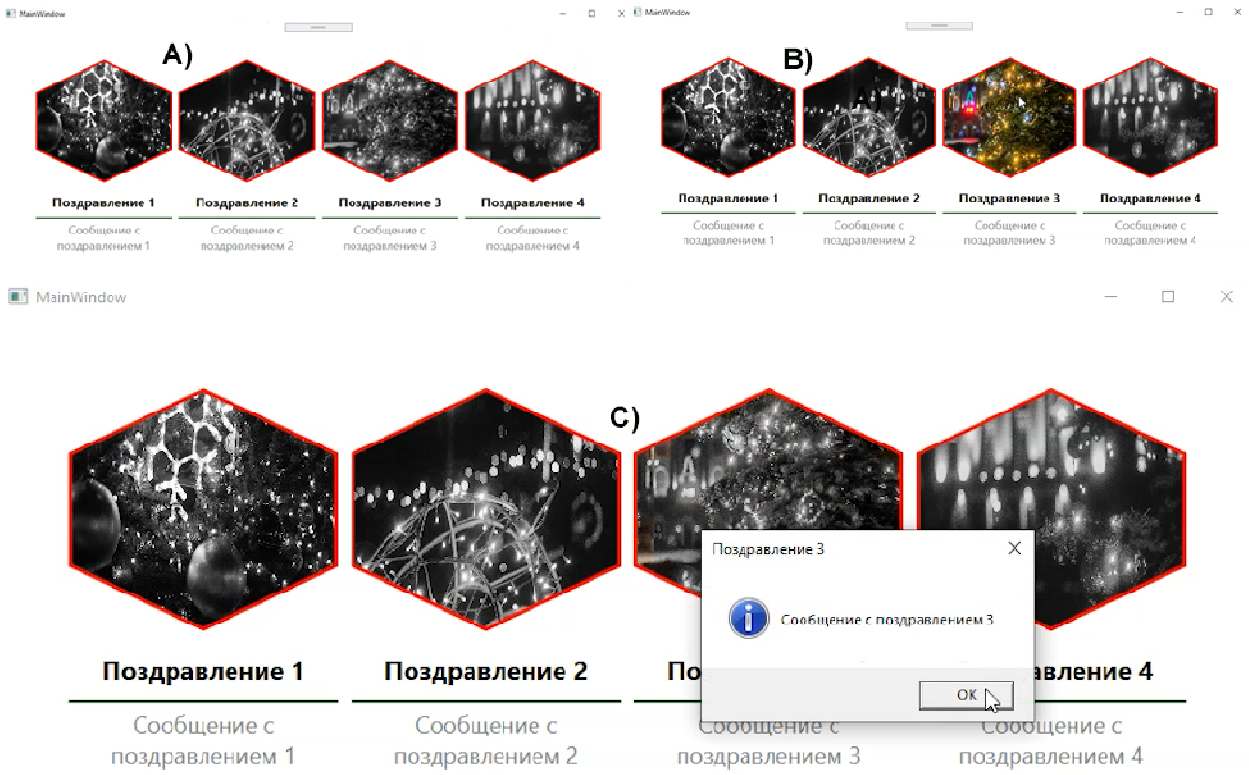
3. Добавим каждому элементу соответствующее изображение



Важно

Изображения можно добавлять как вручную, так и путем выбора в окне свойств. В данном случае требуемое изображение выбирается из нескольких, хранящихся в папке Assets (см. на область внутри черной рамки)

4. Проверяем результат



A) Курсор не на изображениях
B) Курсор на изображении
C) Обработка нажатия

Интерактивное задание:
https://nationalteam.worldskills.ru/skills/realizatsiya-polzovatelskikh-elementov-upravleniya-usercontrol/

Практическая работа № 20. Модульное тестирование (Unit-tests)

Краткие теоретические сведения

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:

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

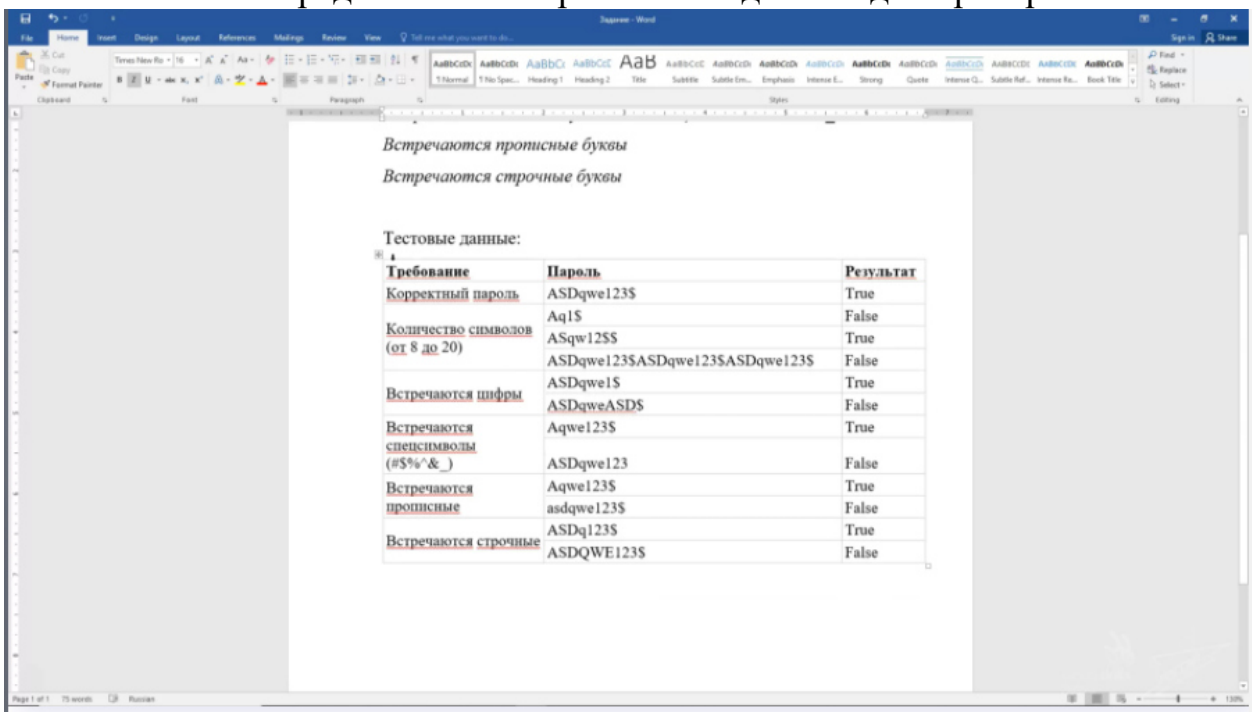
2. создают реализацию возможностей и добиваются того, чтобы она успешно прошла тестирование

3. 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



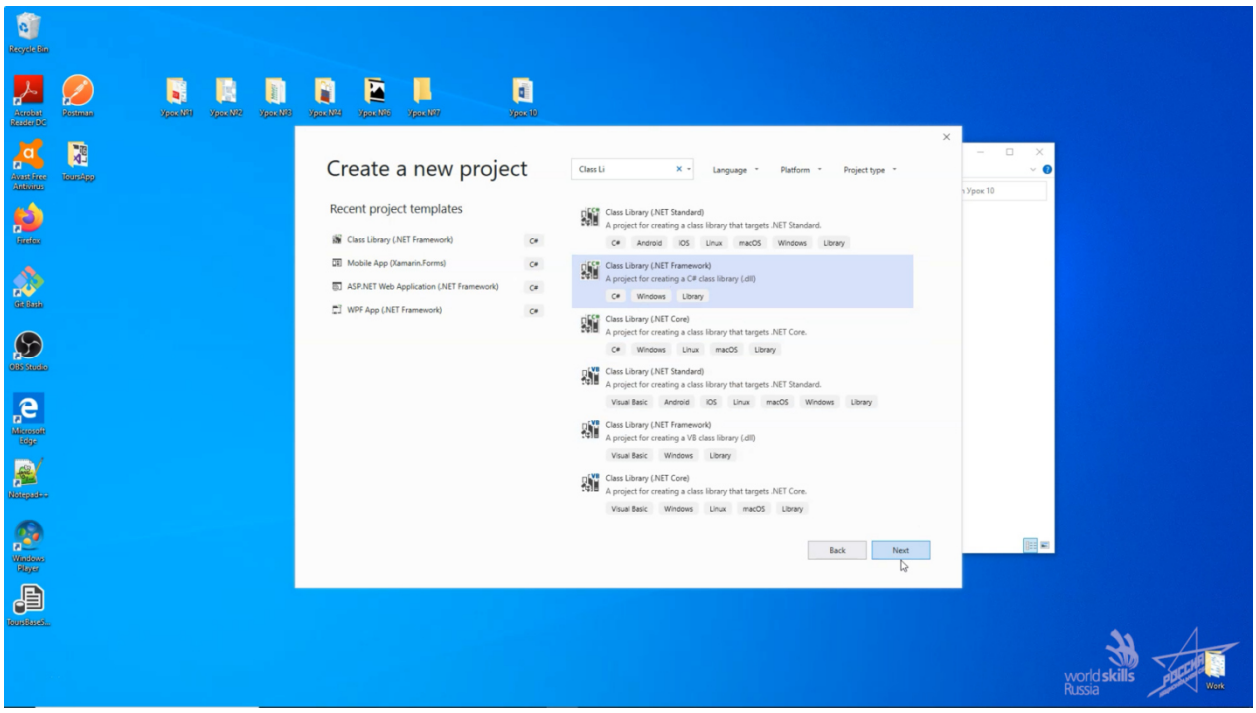
Встречаются прописные буквы

Встречаются строчные буквы

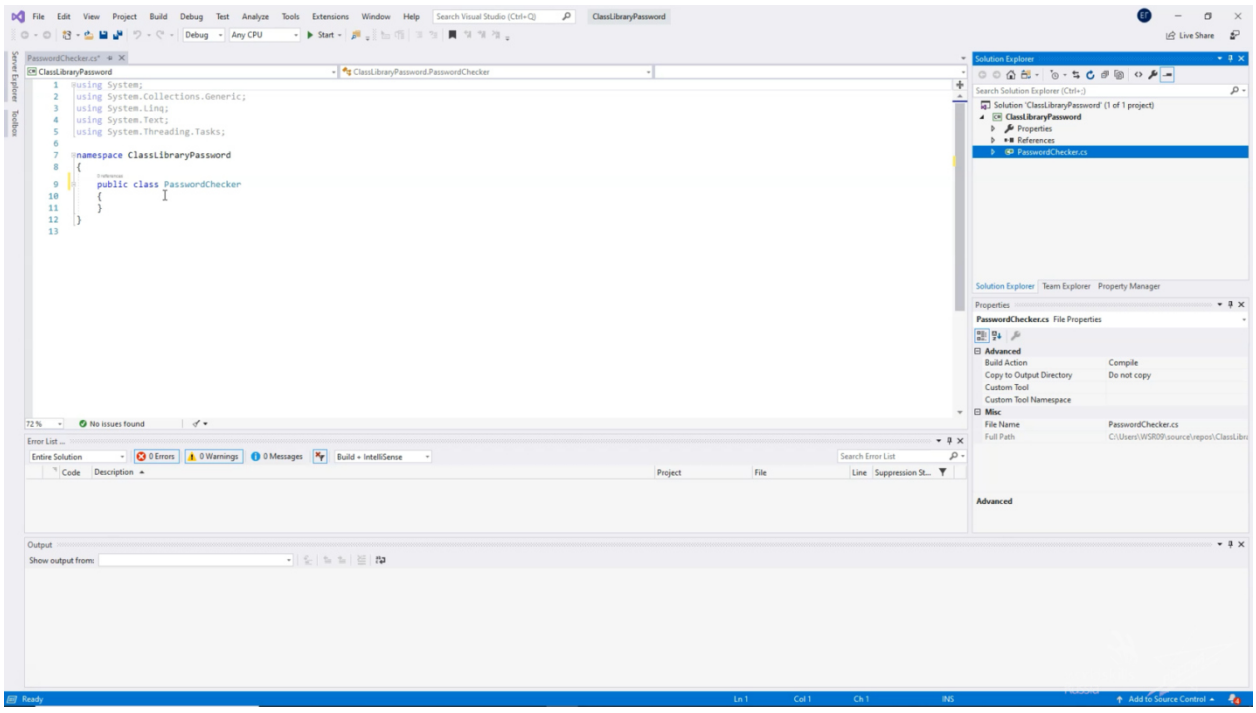
Тестовые данные:

Требование	Пароль	Результат
Корректный пароль	ASDqwe123$	True
Количество символов (от 8 до 20)	Aq1$	False
	ASqw12$$	True
	ASDqwe123$ASDqwe123$ASDqwe123$	False
Встречаются цифры	ASDqwe1$	True
	ASDqweASD$	False
Встречаются спецсимволы (#$%^&_)	Aqwe123$	True
	ASDqwe123	False
Встречаются прописные	Aqwe123$	True
	asdqwe123$	False
Встречаются строчные	ASDq123$	True
	ASDQWE123$	False

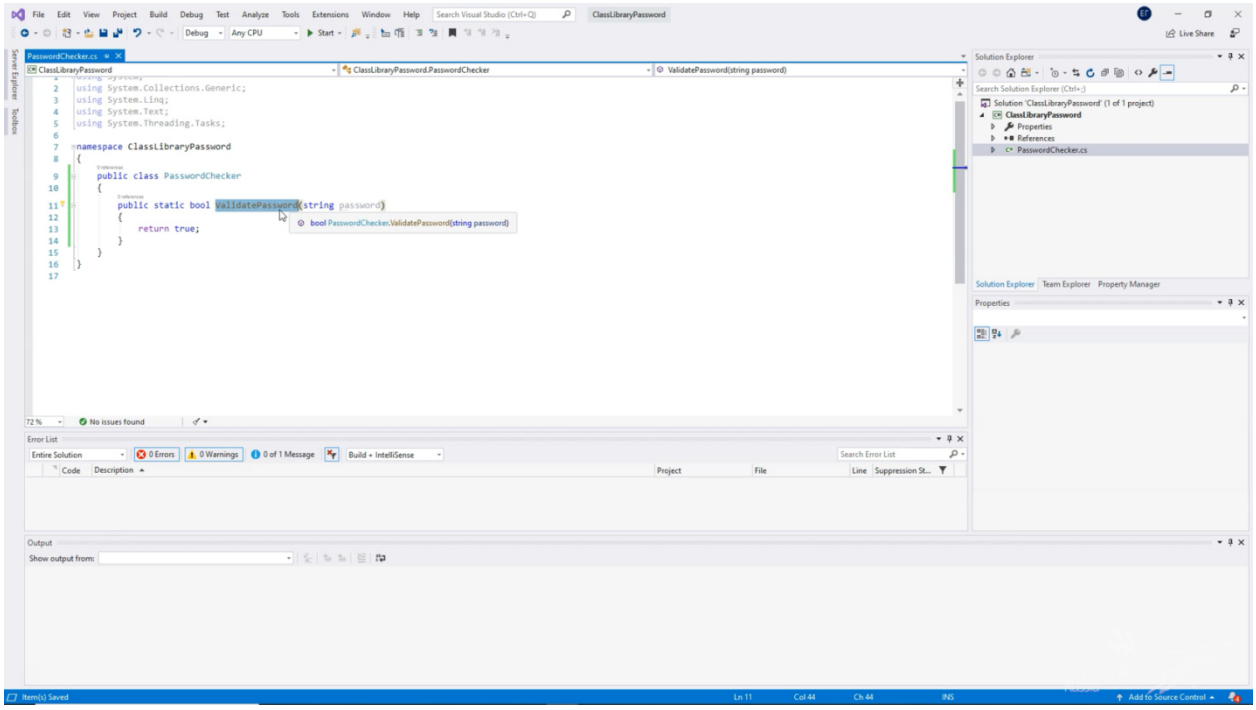
1. Создаем библиотеку .NET Framework



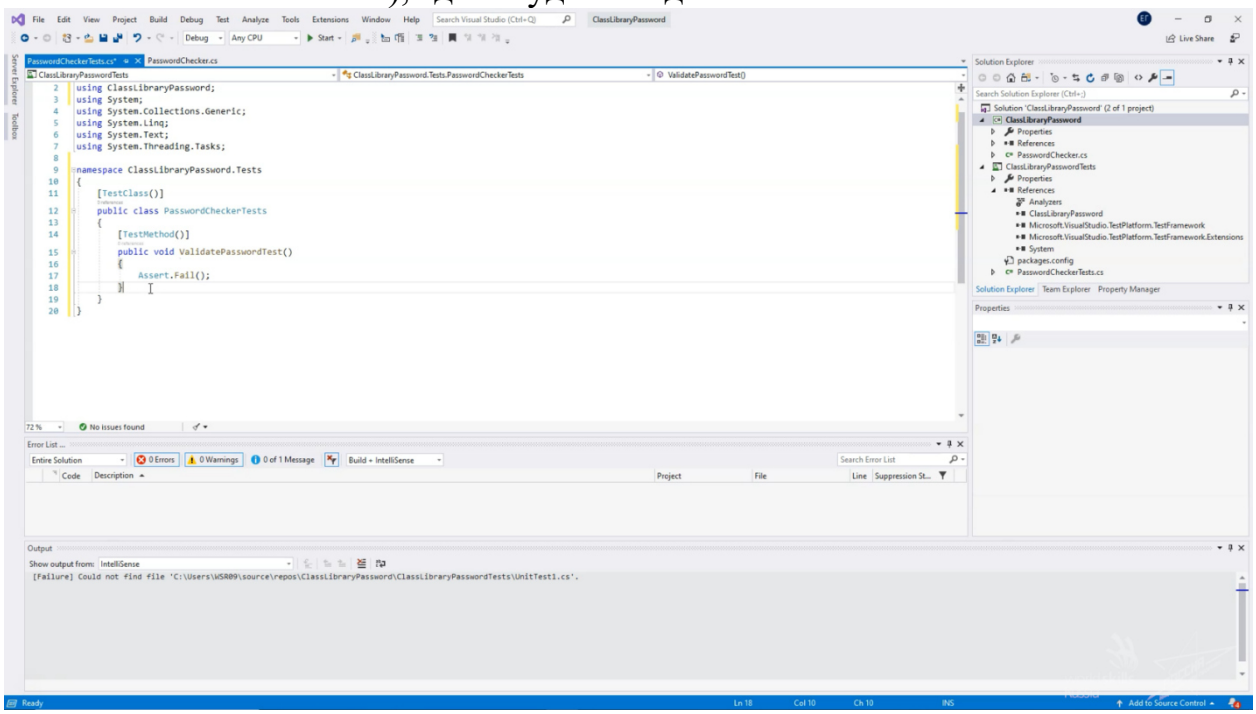
2. Переименовываем стандартный класс в PasswordChecker



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

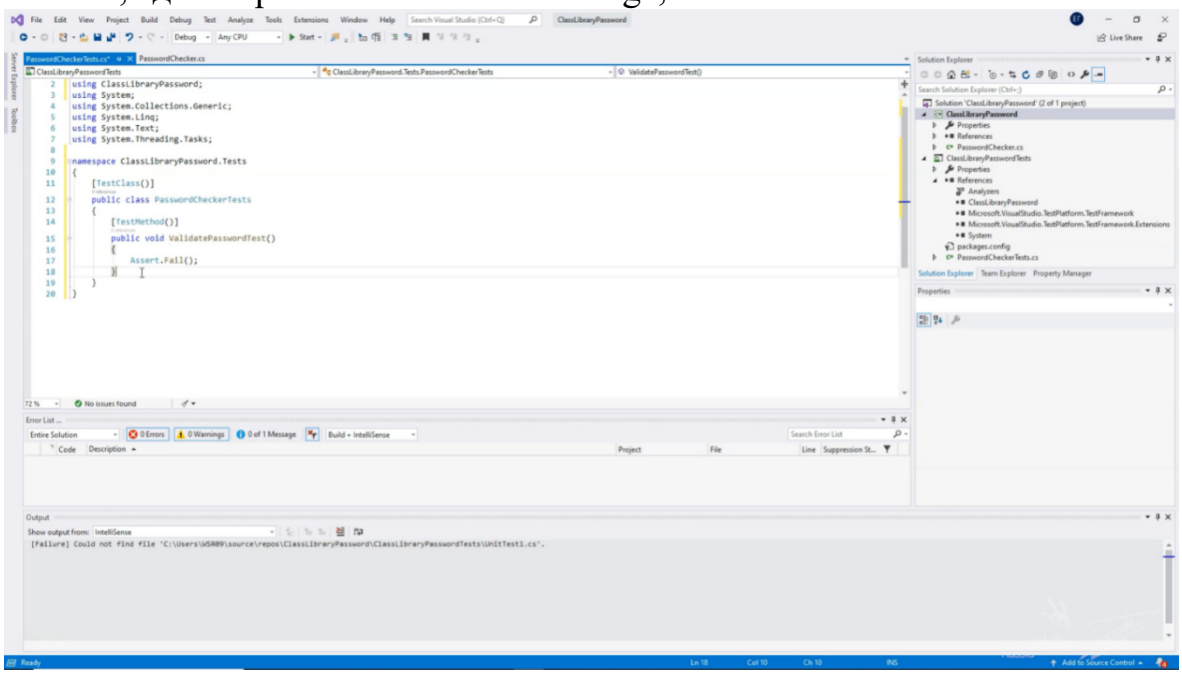


5. Есть очень хороший подход оформления тестов, который формулируется Arrange-Act-Assert. Суть его заключается в том, чтобы в модульном тесте четко определить:

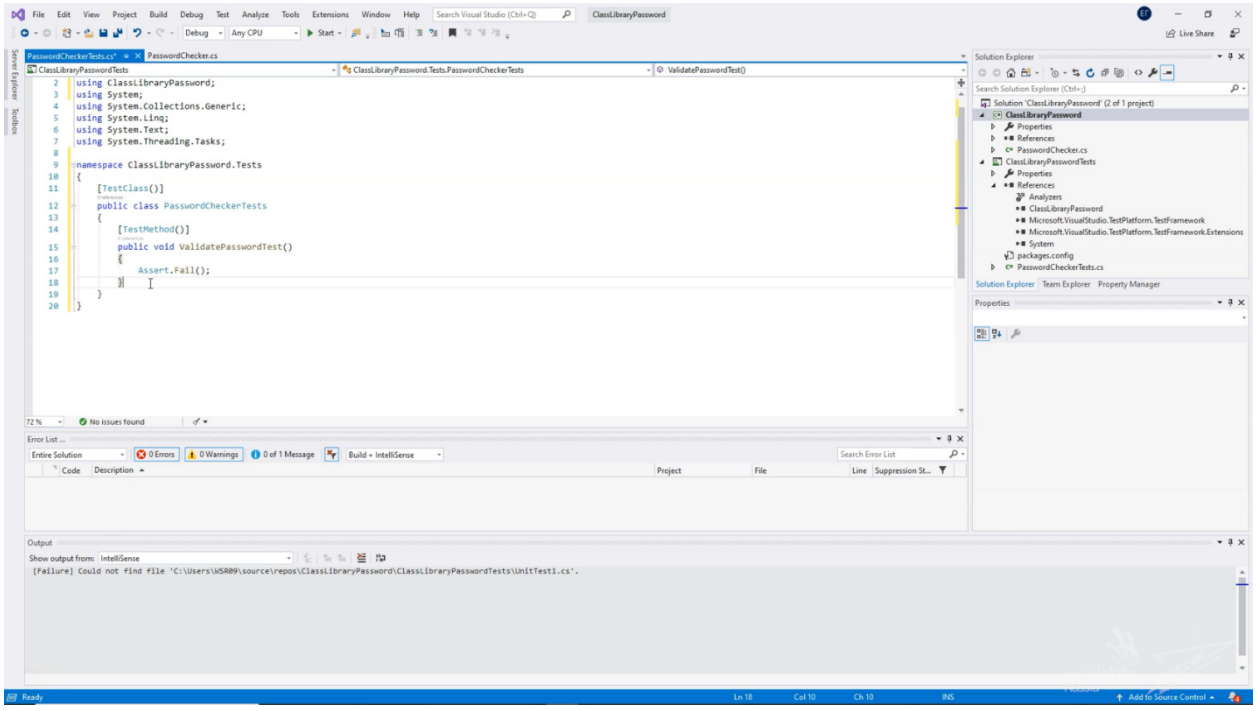
- предусловие (блок Arrange) — устанавливает начальные условия для выполнения теста
- действие (блок Act) — выполняет сам тест
- постусловие (блок Assert) — верифицирует результат теста, и, в данном случае, оформление — повышает читаемость текста и облегчает

его использование в качестве документации к тестируемой функциональности

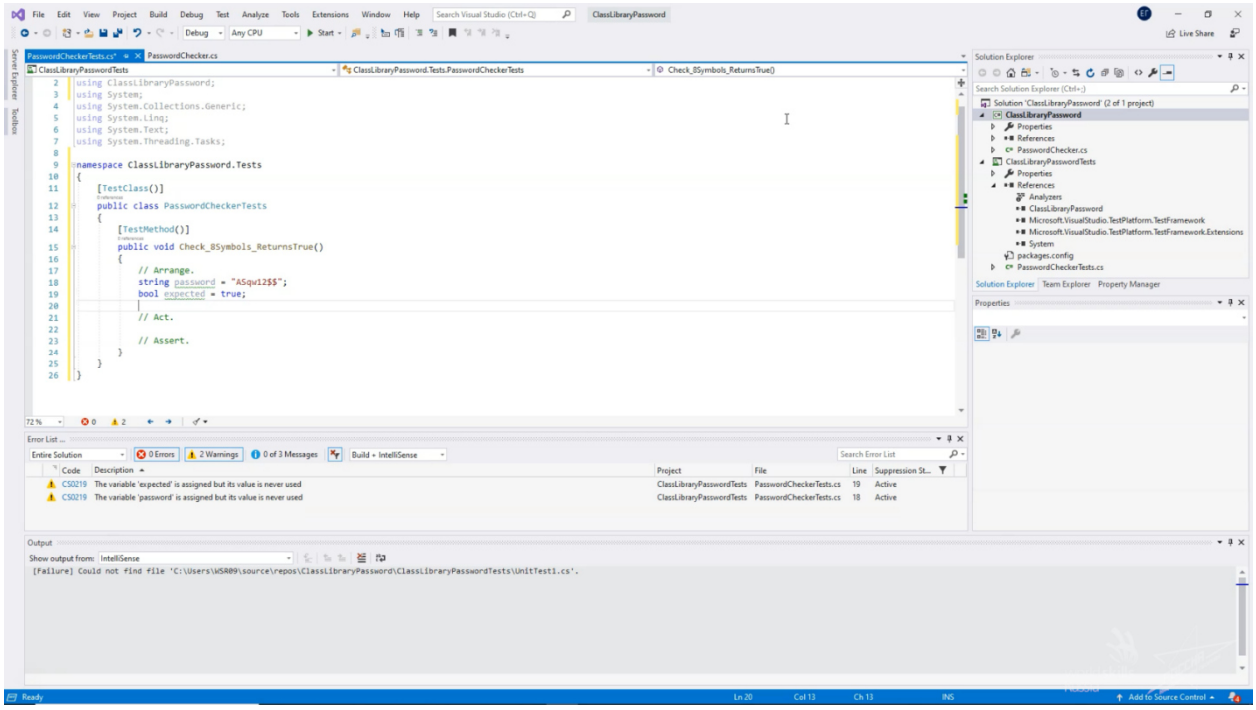
Например, напишем первый тест. Он будет проверять количество символов, где мы разместим блоки Arrange, Act и Assert



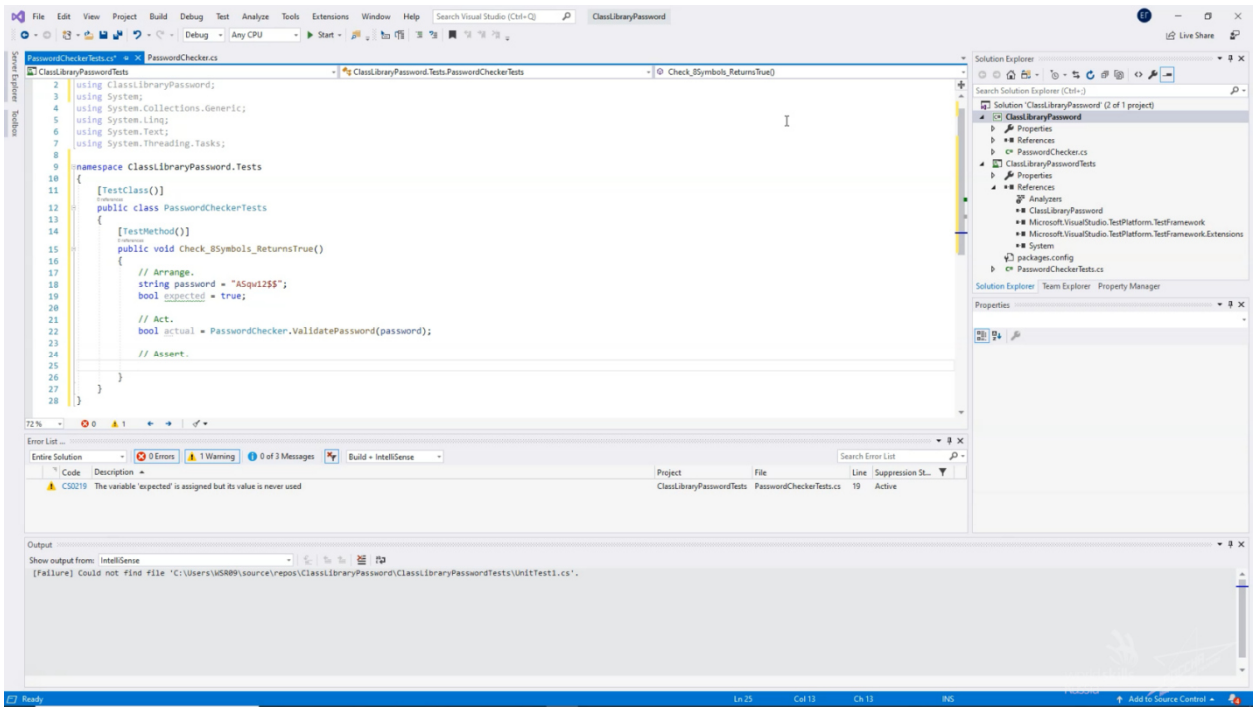
- Объявляем переменную для установки пароля из тестовых данных



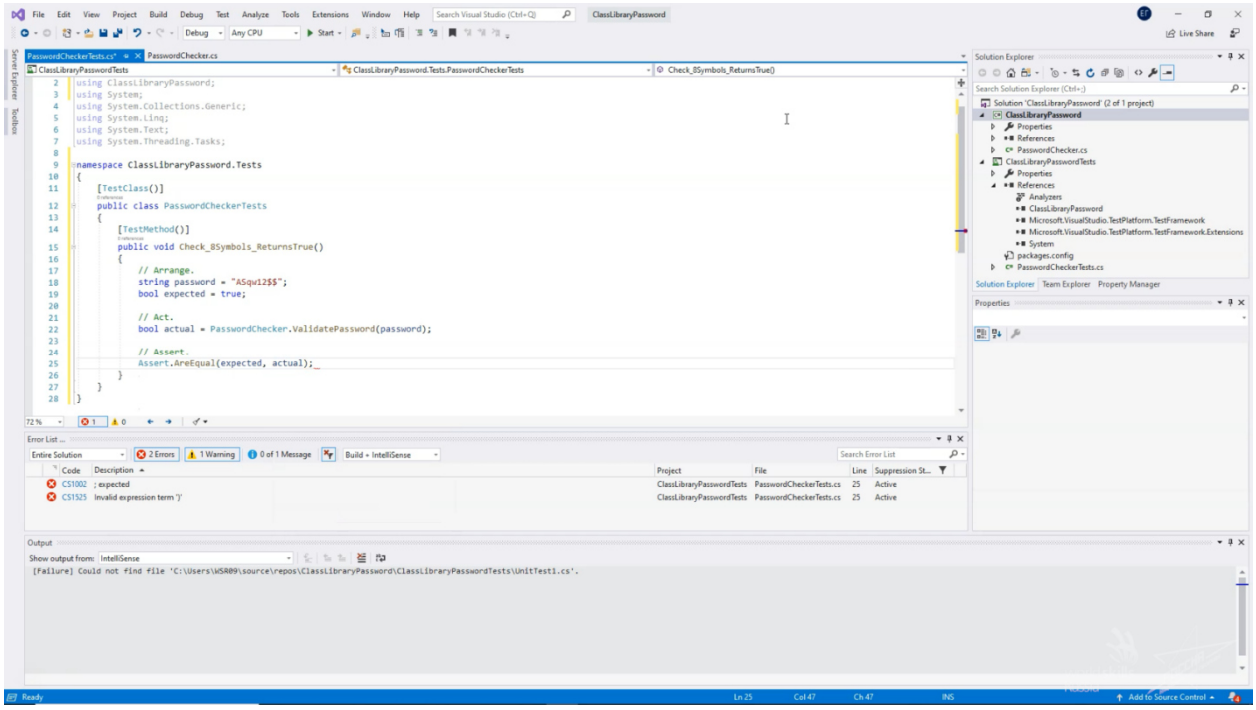
- Здесь же объявляем ожидаемое значение в результате выполнения теста



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

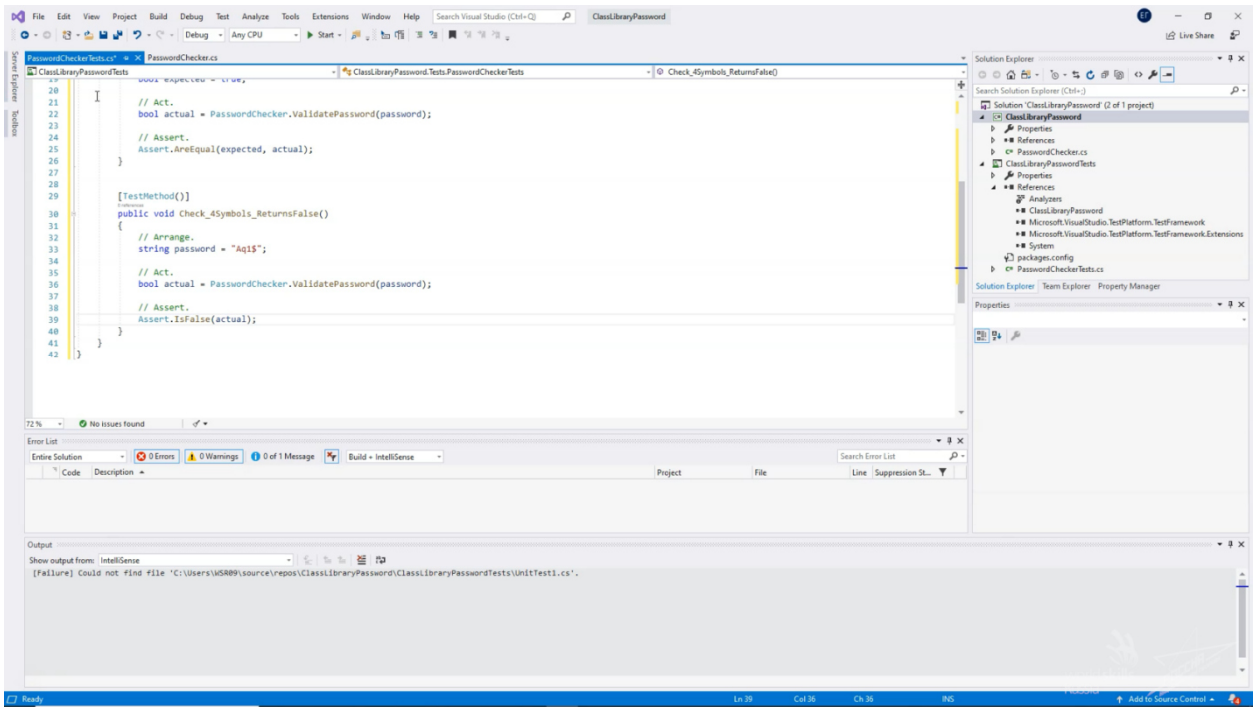


- С помощью класса Assert сравниваем два значения: ожидаемое и реальное, метод ArEquel, и в качестве аргумента — наши данные

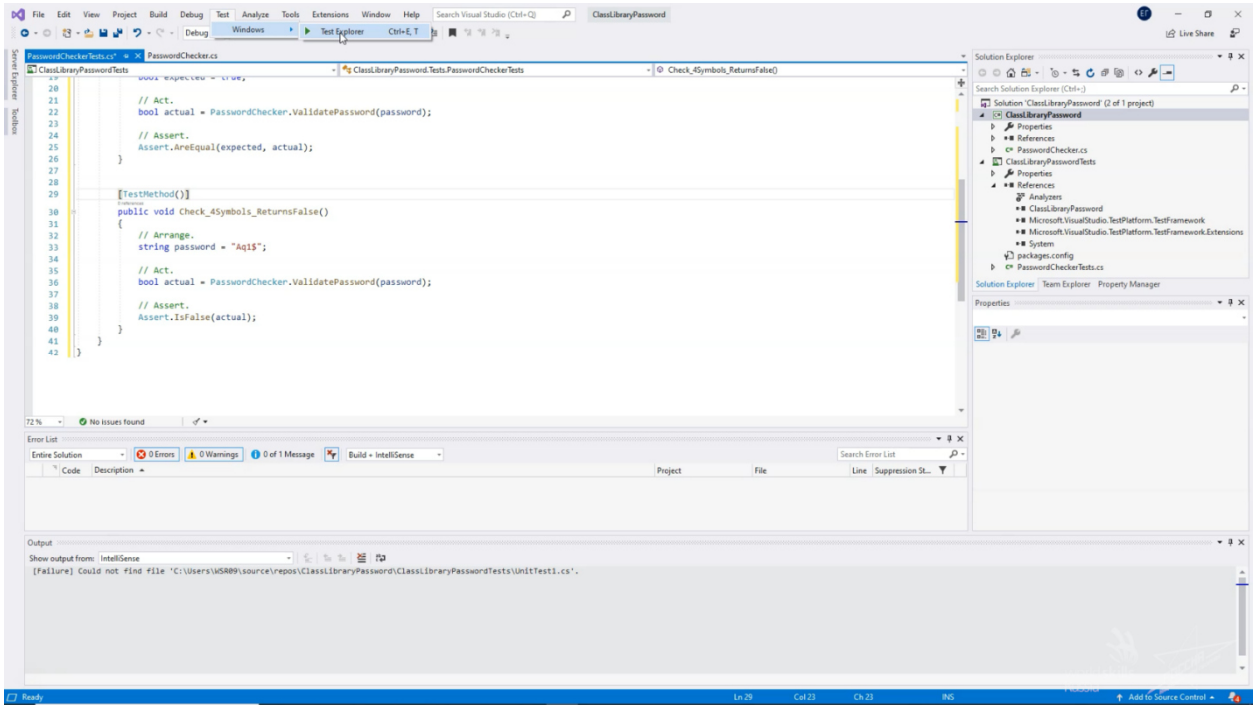


Для проверки результата в классе Assert, помимо ArEquel, определен ряд методов, среди которых можно выделить, например, следующие: All, Matches, Empty, IsTrue, IsNull, Throws и другие

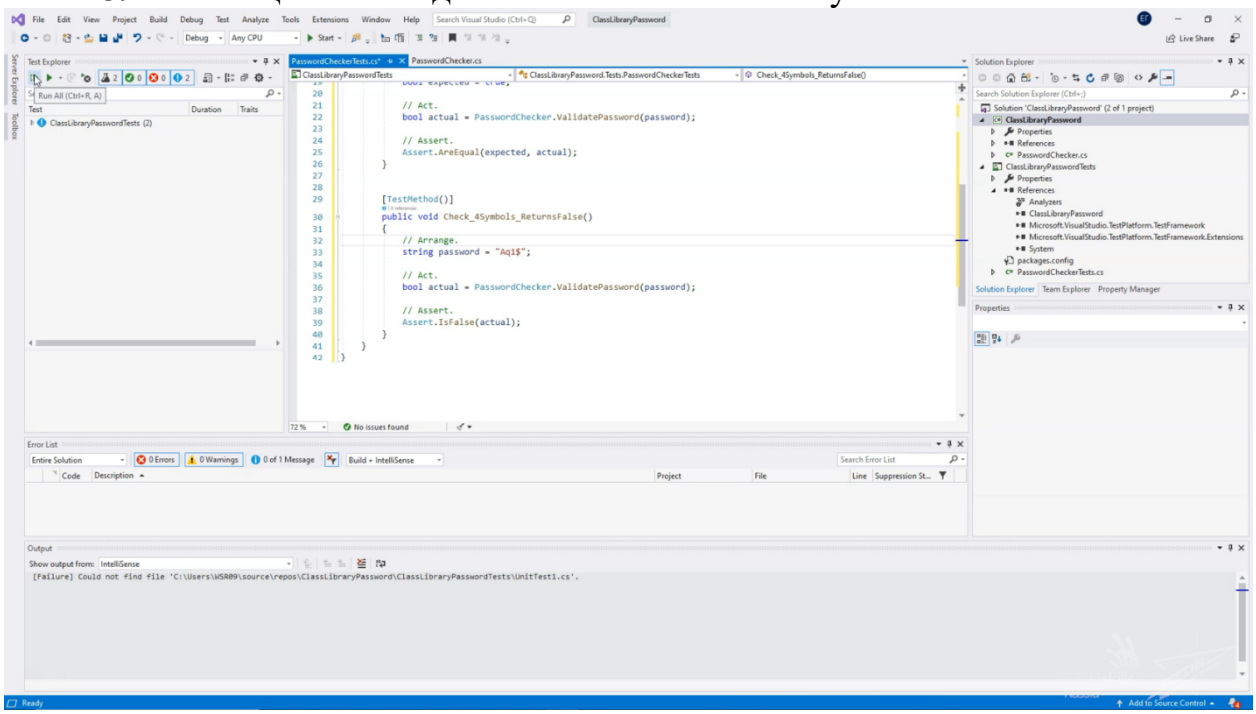
6. Давайте во втором тестовом методе воспользуемся классом IsFalse в блоке Assert



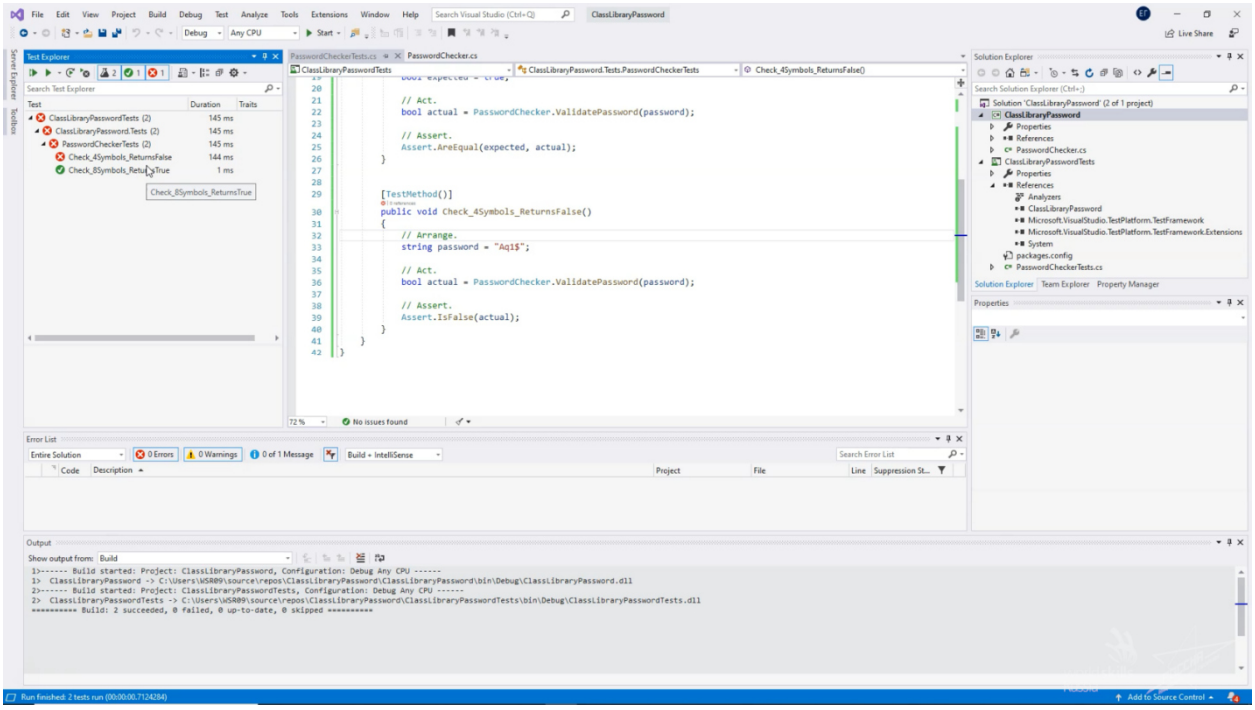
7. Запускаем тесты: на вкладке Test Windows можно открыть обозреватель тестов



8. С помощью команды Run All можно их запустить



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

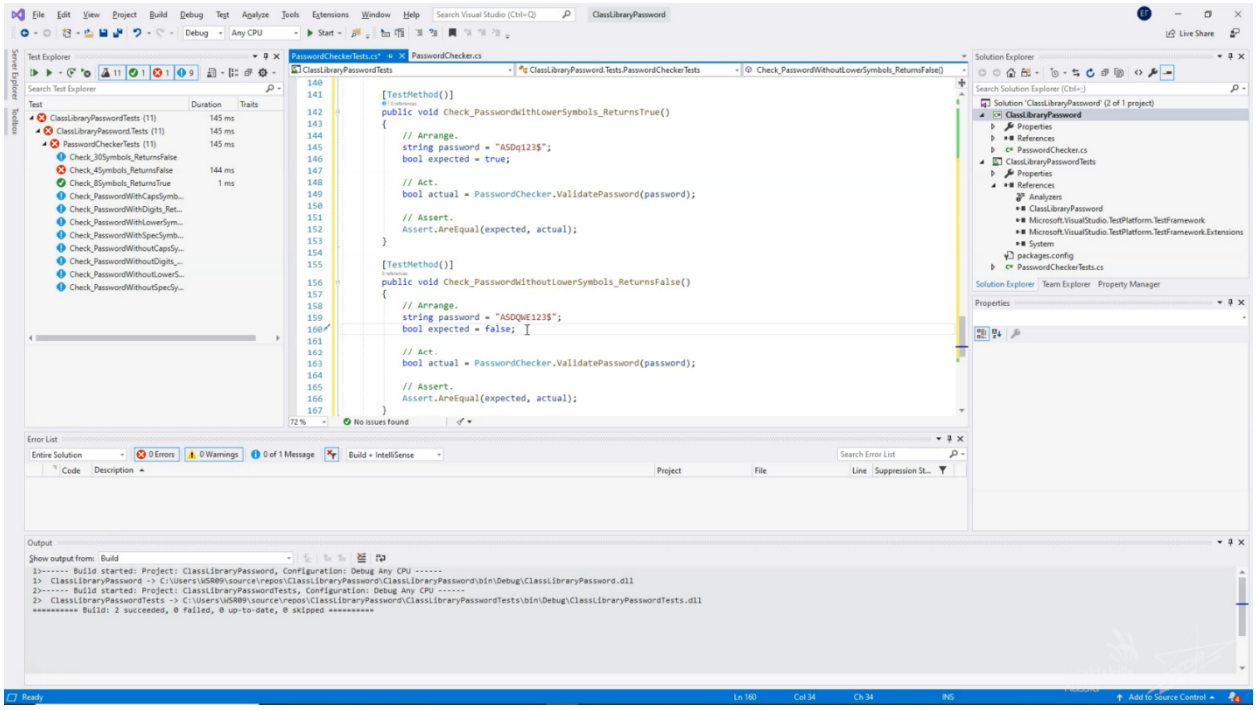


10. Обращаем ваше внимание на наименование тестовых методов. Имя теста должно состоять из трех частей

1. имя тестируемого метода
2. сценарий, в котором выполняется тестирование
3. ожидаемое поведение при вызове сценария

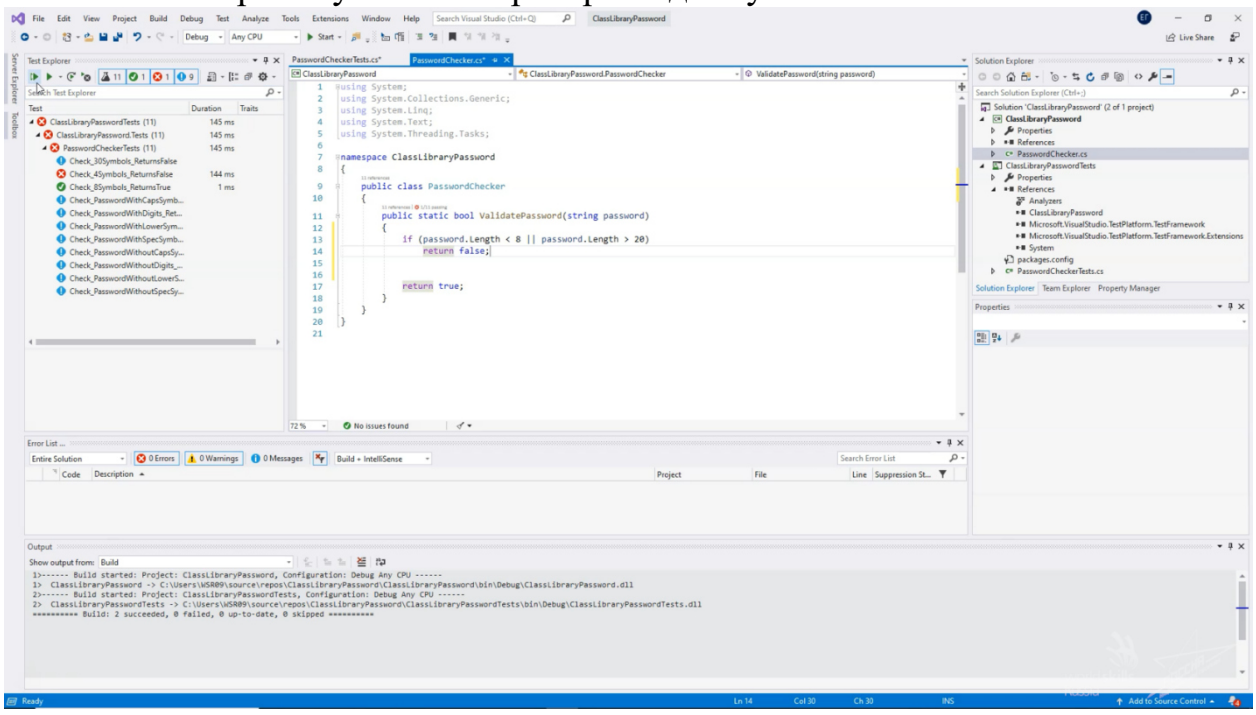
Например, в методе Check_4SymbolsReturnsFalse имя тестируемого метода находится в первой части названия — Check, затем идет сценарий — то, что у нас используется 4 символа, а затем ожидаемое поведение, у нас вернется False

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)

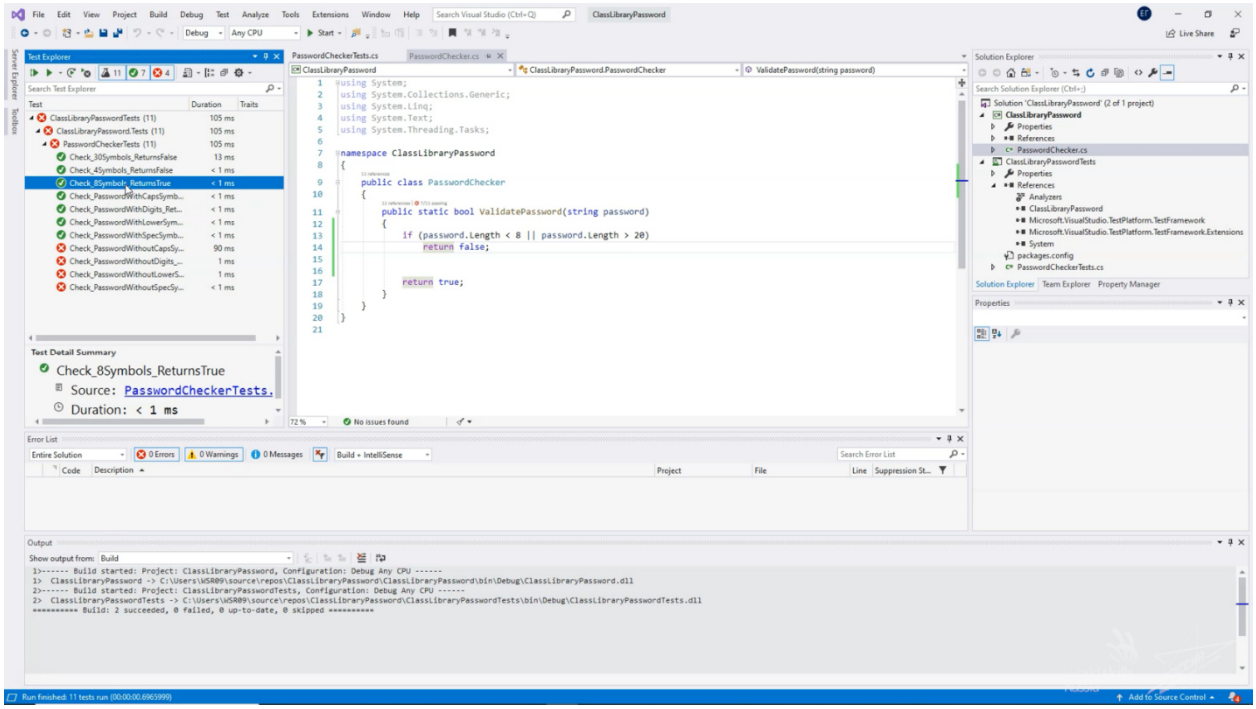


12. Теперь, когда все тесты готовы, самое время приступить к написанию тела метода, проверяющего пароль

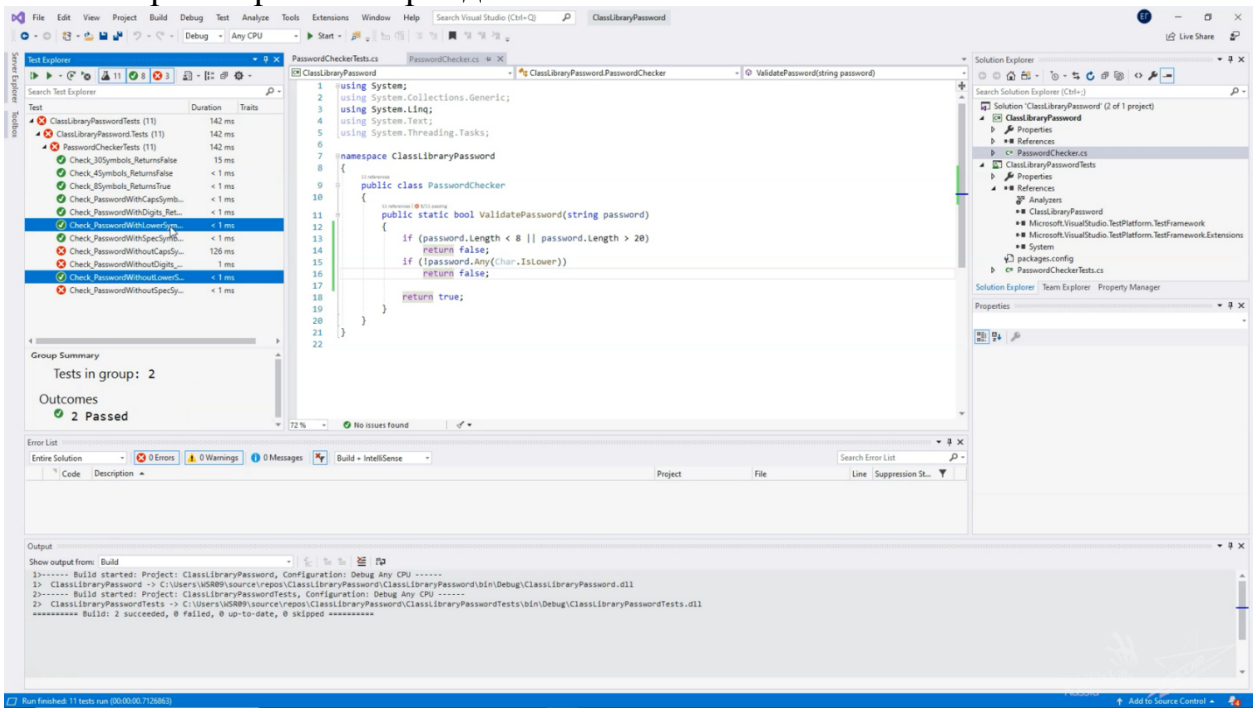
- Первым условием проверяем длину



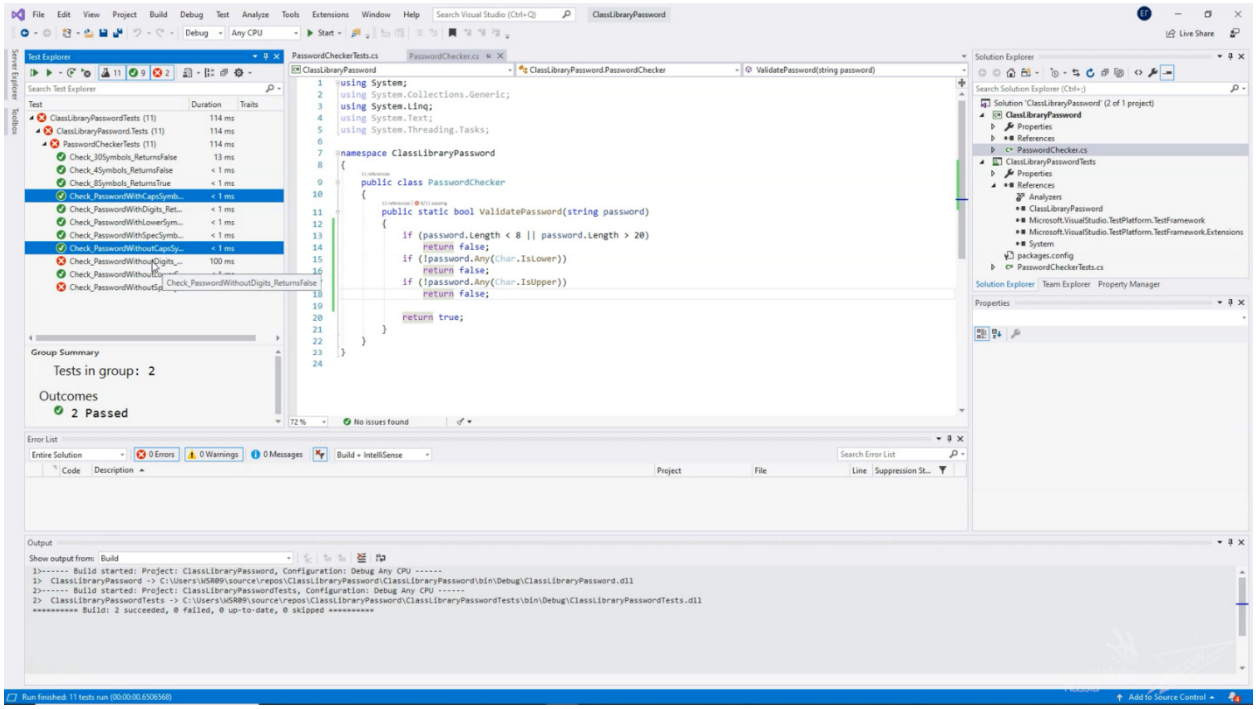
- Запускаем тесты. Как видно, тесты на количество символов 34,8 прошли



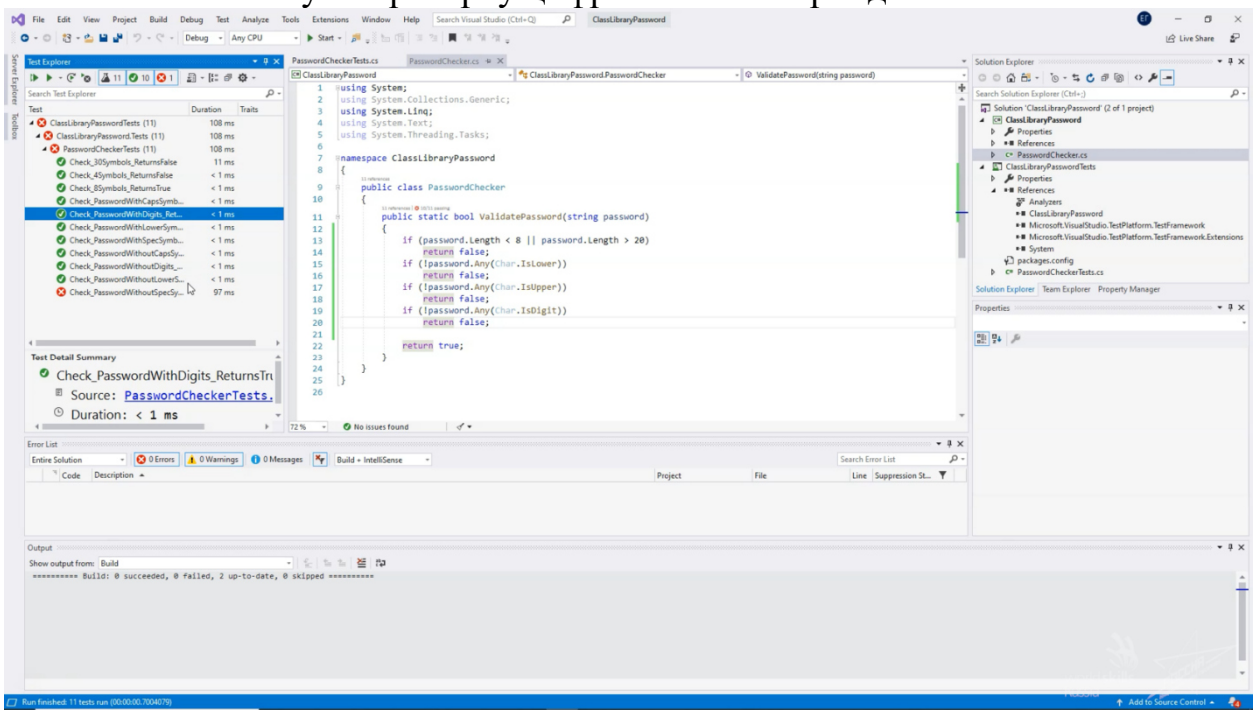
- Далее реализуем проверку строчных букв. Два теста на проверку нижнего регистра тоже пройдены



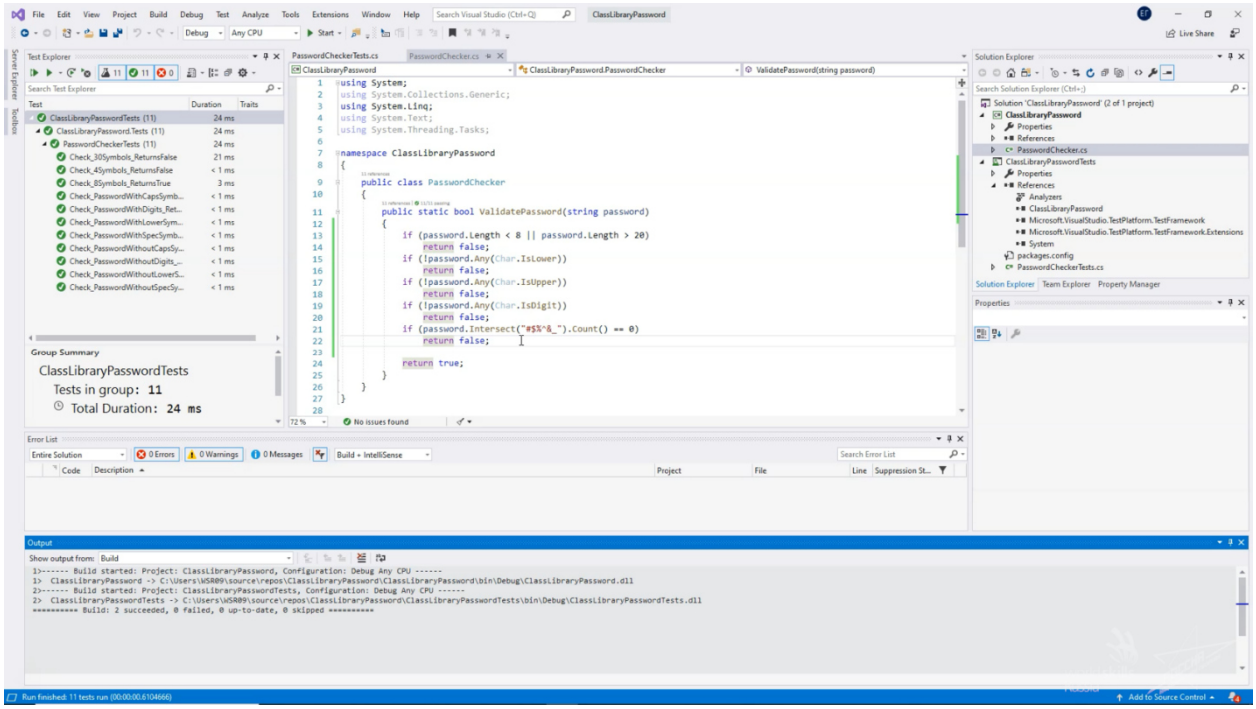
- Затем идет проверка заглавных букв. И они тоже оба пройдены



- 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

Интерактивное задание
https://nationalteam.worldskills.ru/skills/modulnoe-testirovanie-unit-tests/

Практическая работа № 20-1. Работа с системой контроля версий. Создание и управление репозиторием

- Работа с локальным и удаленным репозиториями
- Создание и слияние веток
- 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 новом репозитории будет реализована верстка названий. Основные шаги:

1. Настройка репозитория в Visual Studio
2. Работа в master ветке: разработка функционала

3. Работа в новой ветке: изменение верстки, демонстрация отката изменений, слияние веток

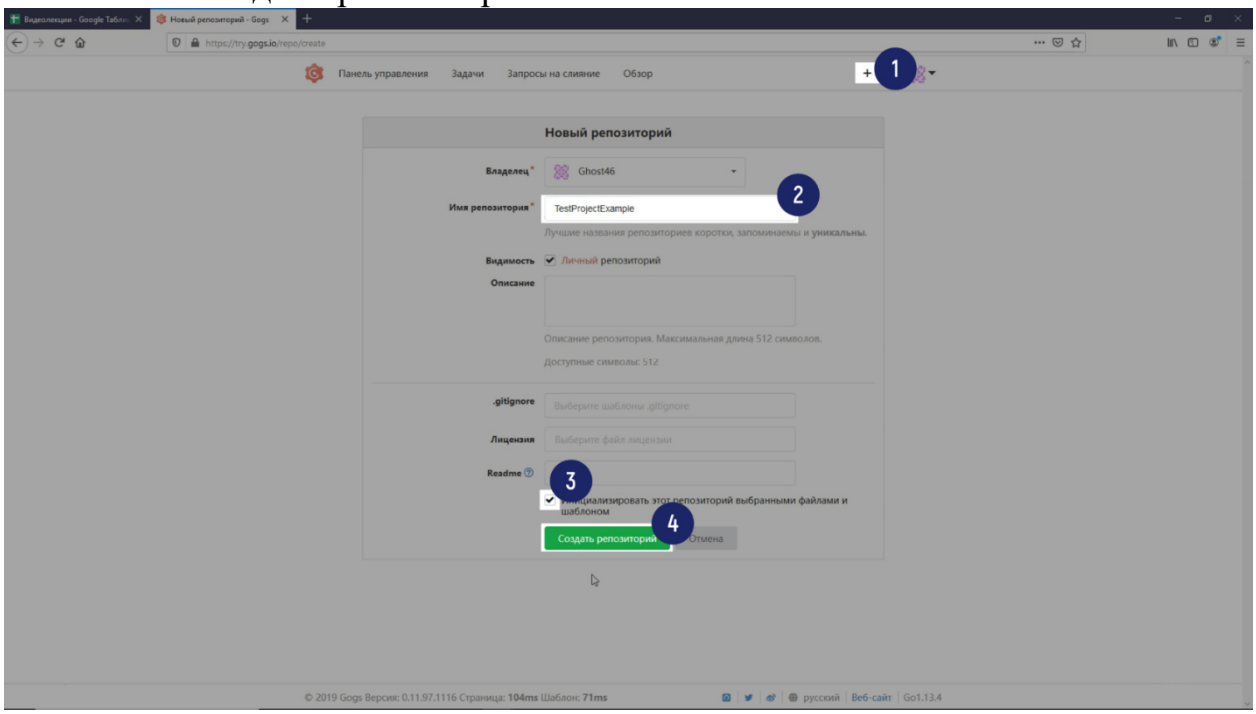
Важно

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

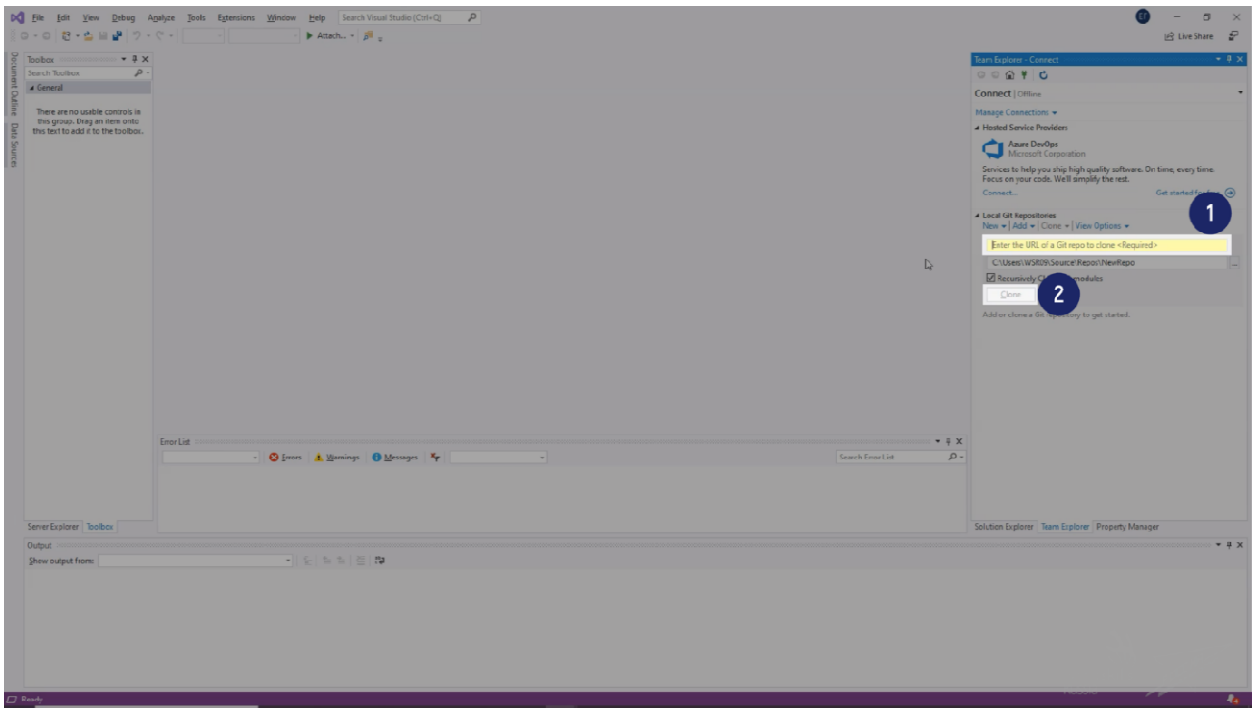
Создаем репозиторий

1. Открываем шаблон
2. Вводим название репозитория
3. Выбираем необходимые параметры
4. Создаем репозиторий



Клонируем репозиторий в среде разработки

1. Указываем путь
2. Клонируем репозиторий



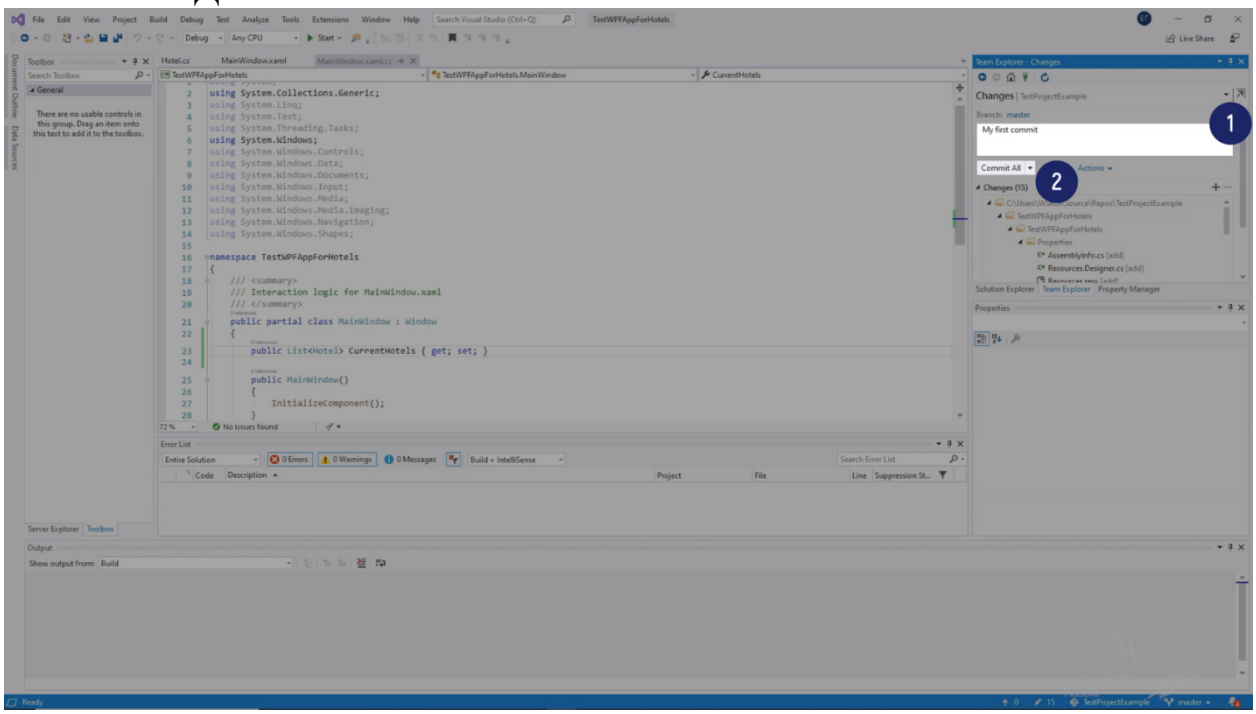
Работа в master ветке

Работаем с кодом

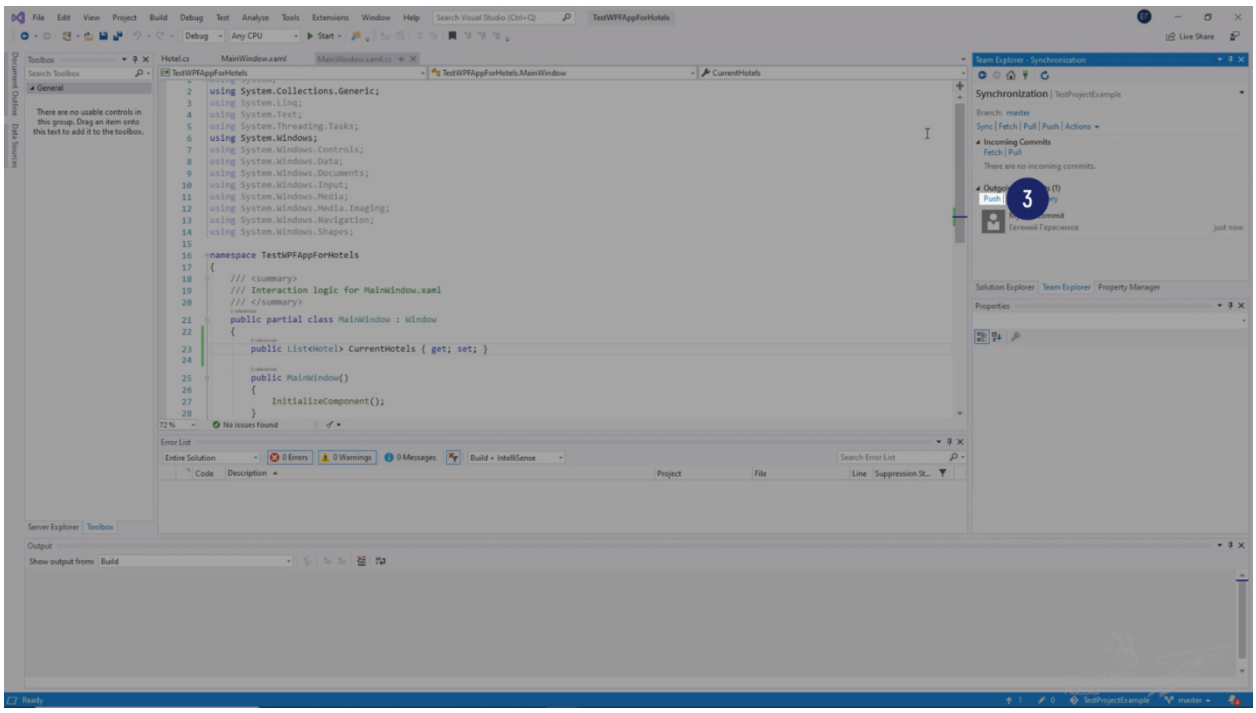
1. Создаем новый проект в репозитории
2. Вносим изменения в код

Добавляем изменения в репозиторий

1. Указываем название commit'a
2. Делаем commit



3. Делаем push изменений

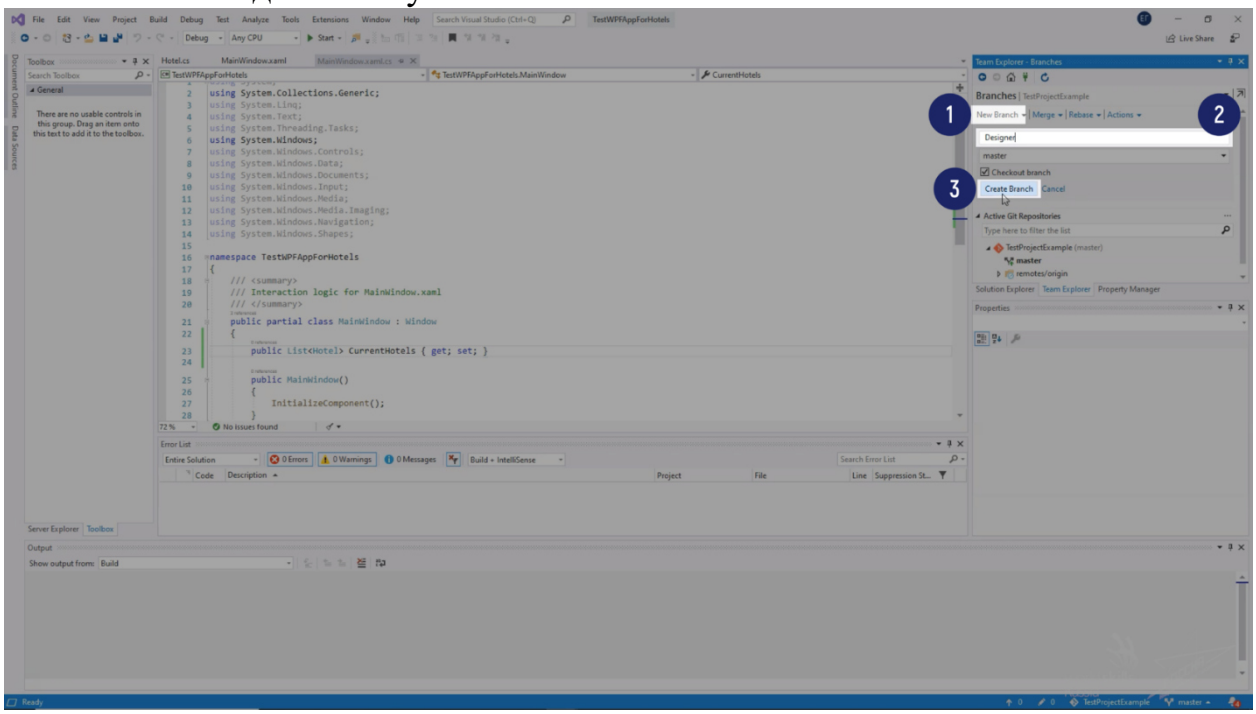


Важно

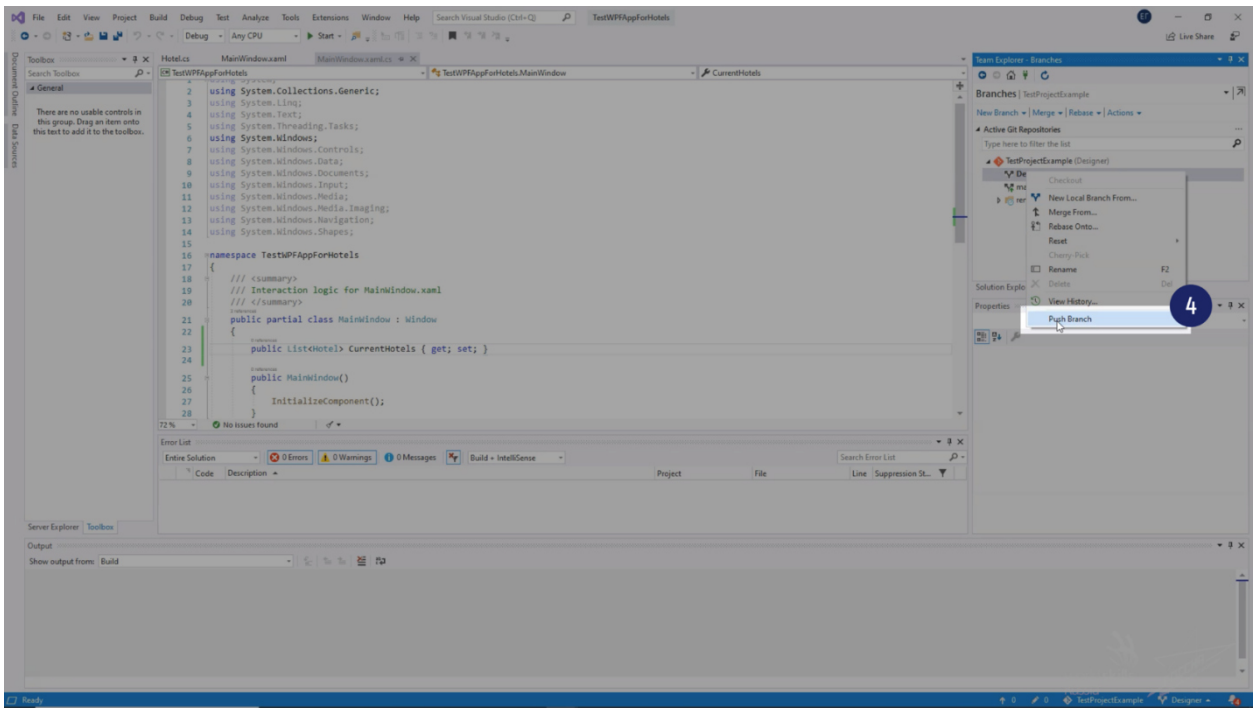
Commit является основным объектом в системе управления версиями и содержит информацию о внесенных изменениях

Создаем новую ветку

1. Переходим к созданию ветки
2. Указываем ее имя
3. Создаем ветку



4. Делаем push новой ветки

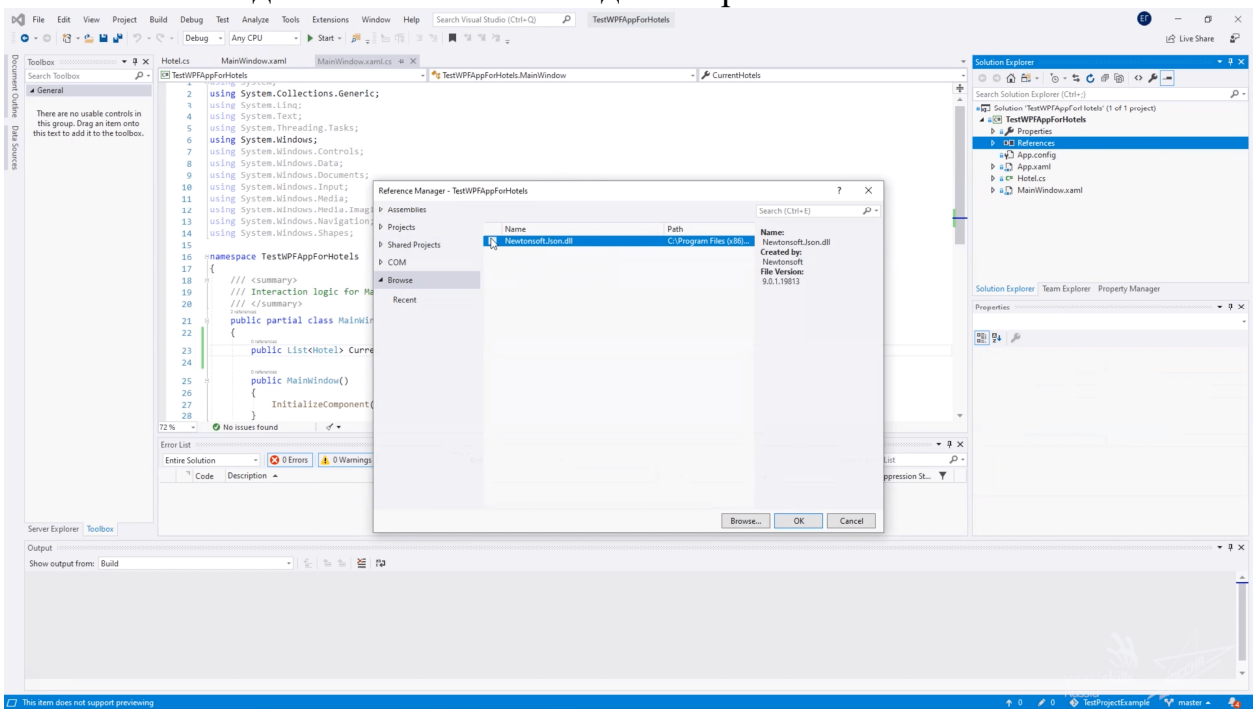


Важно

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

Изменяем код в master ветке

1. Добавляем код метода для вывода списка отелей
2. Создаем новый commit и делаем push изменений



Важно

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

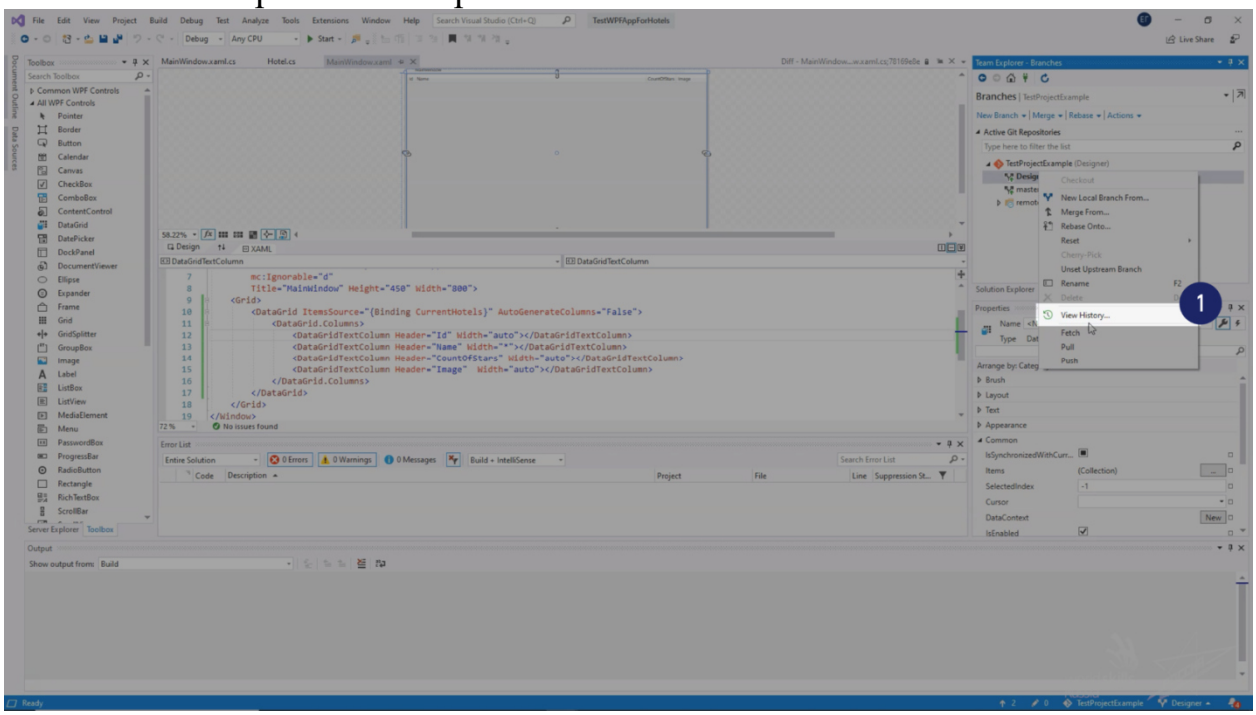
Работа в новой ветке

Работаем с кодом

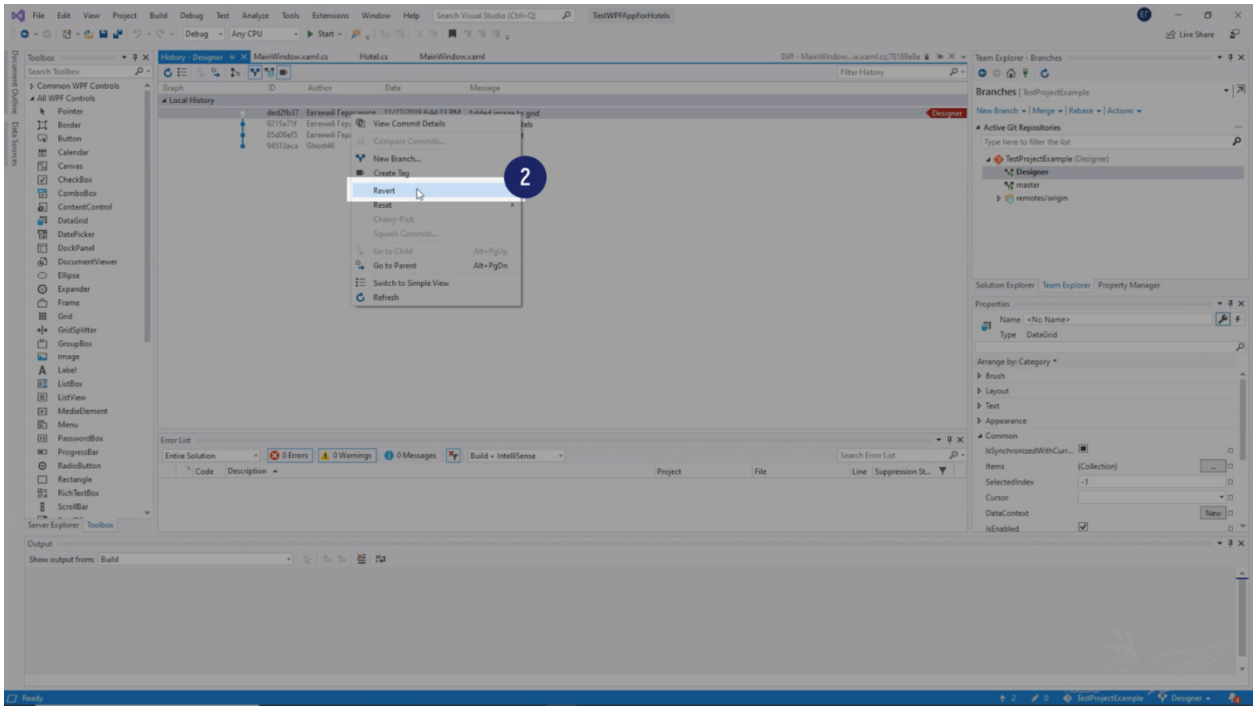
1. Реализуем верстку названий отелей
2. Делаем commit

Как откатить изменения?

1. Открываем историю commit'ов



2. Откатываем изменения



Важно

После отката изменений новая ветка вернется к предыдущему состоянию

Как слить ветки?

1. Выбираем ветку для слияния
2. Сливаем ветки