**МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ**

**Государственное бюджетное профессиональное образовательное**

**учреждение Московской области «Люберецкий техникум имени Героя**

**Советского Союза, летчика-космонавта Ю. А. Гагарина»**

**ОТЧЕТ**

**ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИЧЕСКОЙ ПОДГОТОВКЕ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шумейковой Татьяны Александровны\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество студента)

по профессиональному модулю

*ПМ. 01 Разработка модулей программного обеспечения для компьютерных систем*

*\_\_\_\_\_\_\_\_\_\_\_ПМ. 02 Осуществление интеграции программных модулей\_\_\_\_\_\_\_\_\_\_*

Специальность \_*09.02.07 "Информационные системы и программирование"*\_

(код, название)

Курс \_4\_ Группа № \_185\_

Период практики с «\_16\_»\_февраля\_2022 г. по «\_19\_»\_апреля\_2022 г.

Руководитель практической подготовки

от техникума \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_преподаватель Жирнова Ю. В.\_\_

(подпись) (должность, Ф.И.О.)

Руководитель практической подготовки

от организации \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_ген. директор Лобзин Р. В.\_\_\_

(подпись) (должность, Ф.И.О.)

М.П.

Дзержинский 2022

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc100905537)

[Описание предметной области 5](#_Toc100905538)

[Изучение программных средств 7](#_Toc100905539)

[1. Создание объектов базы данных 9](#_Toc100905540)

[2. Разработка API 15](#_Toc100905541)

[3. Разработка библиотеки классов 17](#_Toc100905542)

[4. Разработка модульных тестов (Unit-tests) 20](#_Toc100905543)

[5. Проектирование диаграммы Use Case 22](#_Toc100905544)

[6. Проектирование диаграммы Sequence 24](#_Toc100905545)

[7. Программная работа с Excel и Word для экспорта данных в отчет 28](#_Toc100905546)

[Заключение 36](#_Toc100905547)

[Список литературы 38](#_Toc100905548)

Введение

Производственная практика для студентов — это важная составляющая учебного процесса, позволяющая сориентироваться на рынке труда и найти себя в будущей профессии.

Основная цель производственной практики — закрепление и углубление теоретических знаний, полученных в процессе обучения, реализация адаптационных возможностей студента к новым условиям работы, а также выработка навыков и овладение профессиональными знаниями.

Для достижения данной цели требуется решить следующие задачи:

* Проектирование ERD, создание словаря данных, разработка базы данных по ERD и импорт данных;
* Создание объектов базы данных;
* Подключение базы данных к API, создание контроллера и интерфейса;
* Разработка библиотеки классов;
* Создание интернет запросов и работа с файлами JSON;
* Проектирование требований;
* Проектирование UML-диаграмм;
* Создание отчета;
* Разработка модульных тестов (Unit-tests);
* Создание пояснительной записки.

В ходе производственной практики планируется освоение следующих видов деятельности: «Разработка модулей программного обеспечения для компьютерных систем», «Осуществление интеграции программных модулей» (ПК):

ПК 1.1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием.

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК 1.4. Выполнять тестирование программных модулей.

ПК 1.5. Осуществлять рефакторинг и оптимизацию программного кода.

ПК 1.6. Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 2.1. Разрабатывать требования к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент.

ПК 2.2. Выполнять интеграцию модулей в программное обеспечение.

ПК 2.3. Выполнять отладку программного модуля с использованием специализированных программных средств.

ПК 2.4. Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.

ПК 2.5. Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования.

Описание предметной области

Акционерное Общество «МЕТТЭМ-БЕТАР» является официальным представителем ООО ПКФ «БЕТАР» — одного из крупнейших производителей приборов учета воды и газа, электрической энергии, тепла, компонентов систем водоснабжения.

Осуществляет оптовую и розничную продажу юридическим и физическим лицам следующих категорий товаров: счетчики воды бытовые, счетчики воды промышленные, счетчики газа, регуляторы давления, преобразователи воды магнитные, счетчики электричества, теплосчетчики, индикаторы магнитного поля, фитинги для труб. Кроме того, организация занимается оптовыми продажами широкого ассортимента фасованной таблетированной соли.

Процесс деятельности организации начинается с поступления заказа от клиента.

Аббревиатура UML расшифровывается как Unified Modeling Language, дословно переводится как «унифицированный язык моделирования». UML — это язык моделирования, который позволяет создавать структуры программных систем.

UML состоит из графических обозначений, диаграмм, которые помогают создать дизайн программных проектов. С помощью UML-диаграмм проектные группы коммуницируют между собой, составляют и проверяют архитектурный дизайн ПО.

Работа с UML-диаграммами — важная часть проекта, так как на этом этапе продумывается его структура. Проектирование помогает в дальнейшем не запутаться в коде, снизить количество ошибок и упростить работу.

UML имеет единый синтаксис, поэтому является международным языком. Диаграммы будут понятны любому человеку, знакомому с ним. Также стоит отметить, что UML используется для разработки широкого спектра программ от информационных систем масштаба предприятия до распределенных веб-приложений.

Основные цели дизайна UML:

* Проектирование. Благодаря UML разработчики получили возможность создавать модели различных процессов, анализировать, проектировать и внедрять программные системы, рисовать схемы приложений, по которым впоследствии пишется код.
* Обеспечение роста рынка объектно-ориентированных инструментов и развития отрасли.
* Создание UML таким образом, чтобы можно было работать с любым языком программирования и любому пользователю.
* Коммуникация внутри команды и при общении с заказчиком.

Изучение программных средств

Компьютерное построение диаграмм — это процесс создания масштабируемых диаграмм на ПК, которые могут использоваться в различных приложениях, таких как макетирование, организационные диаграммы, временные графики, планы этажей и даже прототипы пользовательских программных интерфейсов. Диаграммы состоят из фигур, объектов и трафаретов, которые при правильном объединении могут помочь спроецировать много полезной информации.

Современное компьютерное построение диаграмм основано на использовании векторных форм вместо растровых. Растровые фигуры или растровые изображения плохо масштабируются, и их разрешение и качество ухудшаются с каждым редактированием. Векторные же фигуры являются математическим представлением фигур и, следовательно, хорошо масштабируются. На векторные фигуры редактирование не влияет, и их наиболее целесообразно использовать, когда требуются точные измерения.

Программный продукт Microsoft Visio — векторный графический редактор для Windows, предназначенный для быстрой и качественной разработки графических документов любой сложности. С помощью него можно создавать схемы, блок-схемы, планы, технические проекты, модели, диаграммы, различные чертежи и многое другое.

MS Visio является наиболее популярным программным обеспечением для создания диаграмм, которое позволяет создавать диаграммы, визуализировать данные и моделировать процессы в привычном интерфейсе. Visio поставляется с массивом шаблонов и встроенных форм для создания практически любых диаграмм. Visio также позволяет пользователям определять свои собственные формы и импортировать их в чертеж.

Visio предназначен больше для корпоративного класса, поскольку домашним пользователям редко приходится использовать расширенные функции построения диаграмм. Однако многие пользователи покупают стандартную версию Visio для лучшей визуализации простых диаграмм, таких как генеалогическое дерево или планы этажей.

MS Visio тесно интегрируется с другими продуктами Microsoft Office, такими как Word, Excel и Access. Данные могут быть напрямую импортированы из этих программ и преобразованы в значимые диаграммы, которые изменяются в реальном времени в соответствии с данными.

Более новые версии Visio, от Visio 2013 и выше, поддерживают формат .vsdx, который обеспечивает улучшенное сжатие и многие другие функции по сравнению со старым форматом .vsd, поддерживаемым Visio 2010 и более ранними версиями.

Visio, входящий в состав пакета Office 365, постоянно обновляется.

Имеет следующие полезные функции:

* Визуализатор данных.

Data Visualizer помогает автоматически создавать диаграммы процессов из данных Excel. Готовые или настраиваемые шаблоны Excel, содержащие информацию, которая помещается в диаграммы, можно импортировать непосредственно в Visio с помощью визуализатора данных и преобразовывать в диаграмму Visio. Любые изменения, внесенные в исходный файл Excel, также отражаются на диаграмме Visio.

* Визуализация структур базы данных из исходных данных.

Последняя версия Visio позволяет осуществлять реверс-инжиниринг баз данных для создания их визуального представления. Visio может подключаться к различным базам данных, таким как MySQL, SQL Server, Oracle и т. д., и может обновлять диаграммы синхронно с изменениями в базе данных.

* Шаблоны для отраслевых диаграмм.

Последние обновления Visio для Office 365 поставляются с множеством новых стандартизированных шаблонов, предназначенных для таких отраслей, как бизнес-процессы, разработка программного обеспечения, информационные технологии и образование. Эти шаблоны предоставляются как Microsoft, так и другими третьими лицами, расширяющими возможности Visio.

1. Создание объектов базы данных

**Проектирование ER-диаграммы**

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

ER-диаграмма — интерфейс, позволяющий двум независимым компонентам программного обеспечения обмениваться информацией.

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.

Основные шаги построения ER-диаграммы:

* Добавление сущностей.
* Добавление связей и их настройка.
* Добавление атрибутов.

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

Выделяем сущности и добавляем их на ER-диаграмму (Рис. 1).

Добавим атрибуты к сущностям.

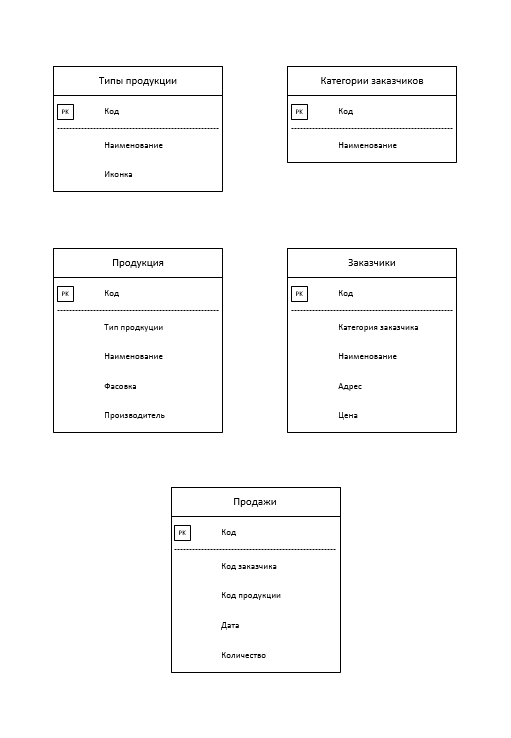


Рисунок 1. Создание сущностей на ER-диаграмме

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

Изображаем связи на ER-диаграмме (Рис. 2).

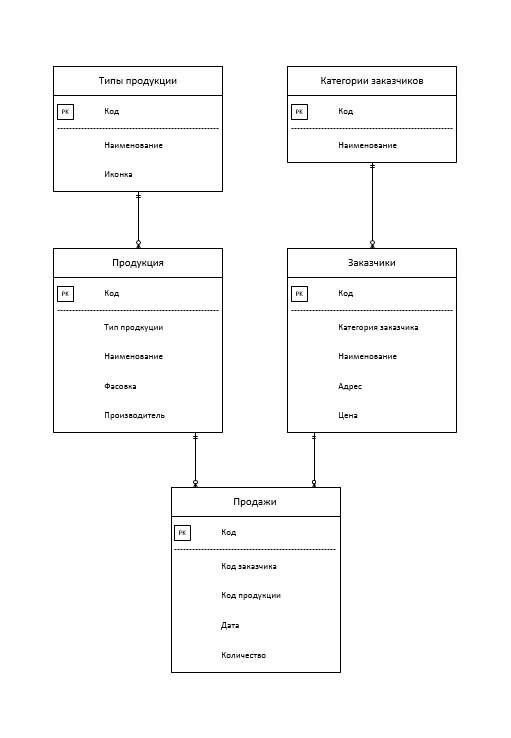


Рисунок 2. Связи ER-диаграммы

**Словарь данных.**

Словарь данных — это хранилище информации обо всех объектах, входящих в состав БД.

Создадим словарь данных, в котором подробно распишем о каждой таблице из бд (все поля таблицы, типы данных, ключи) (Рис. 3).



Рисунок 3. Словарь данных

**Создание представления (View) в БД для отображения списка продукции.**

Представление — это объект базы данных в Microsoft SQL Server, который хранит себе запрос SELECT и в случае обращения к данному объекту — будет возвращен результирующий набор данных, которые формирует запрос, указанный в определении представления.

Иными словами — это некая виртуальная логическая таблица.

Она не содержит себе данных, но к ней можно обращаться как к обычной таблице, и она будет возвращать вам данные.

Для создания представления мы пишем инструкцию "CREATE VIEW".

Затем, указываем название представления.

Далее пишем ключевое слово "AS".

Только после этого мы пишем запрос, результирующий набор которого и будет содержать наши представления.

Выполняем данную инструкцию (Рис. 4).

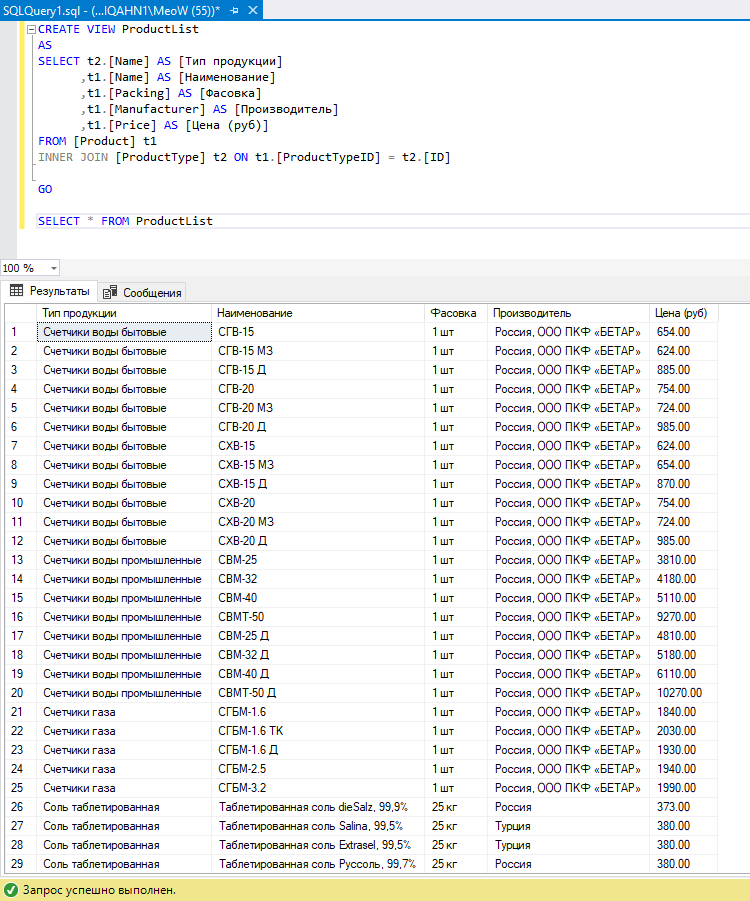


Рисунок 4. Запрос на создание представления (View)

Всё, представление создано. Теперь, для того чтобы получить данные о количестве товаров в категории мы можем обратиться к этому представлению, например, как к обычной таблице.

Также после выполнения скрипта, созданное представление можно найти в разделе «Представления» (Рис. 5).

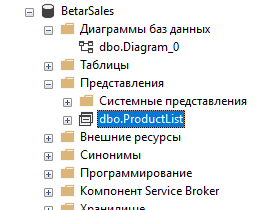


Рисунок 5. Созданное представление

2. Разработка API

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 конечные пользователи обычно его просто не замечают.

Создадим проект «Web API» и добавим модель Entity Framework. Для работы с данными остановимся на формате json, хотя 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.

Для начала настраиваем возвращаемый формат json в файле WebAPIConfig (Рис. 6).

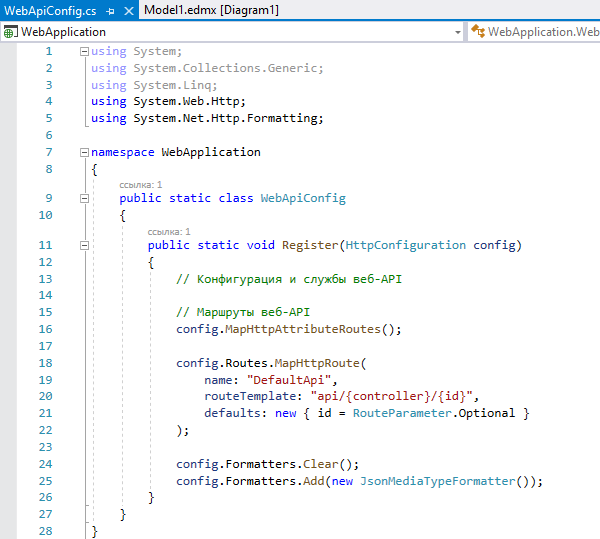


Рисунок 6. Настройка возвращаемого формата

Создадим контроллеры для классов Customers, Products и Sales.

[Route("api/GetSales")]

public IHttpActionResult GetSales(int customerId)

{

var sales = db.Sales.ToList().Where(p => p.CustomerID == customerId).ToList();

return Ok(sales);

}

Запускаем и тестируем в браузере (Рис. 7).

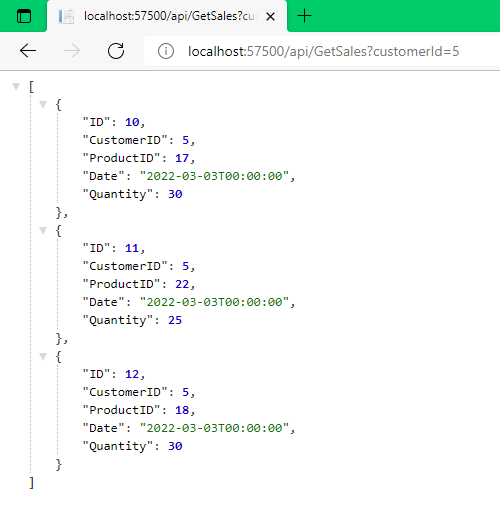


Рисунок 7. Данные формата json в браузере

3. Разработка библиотеки классов

Создадим новый проект. В списке шаблонов проекта найдем пункт Class Library (Рис. 8).

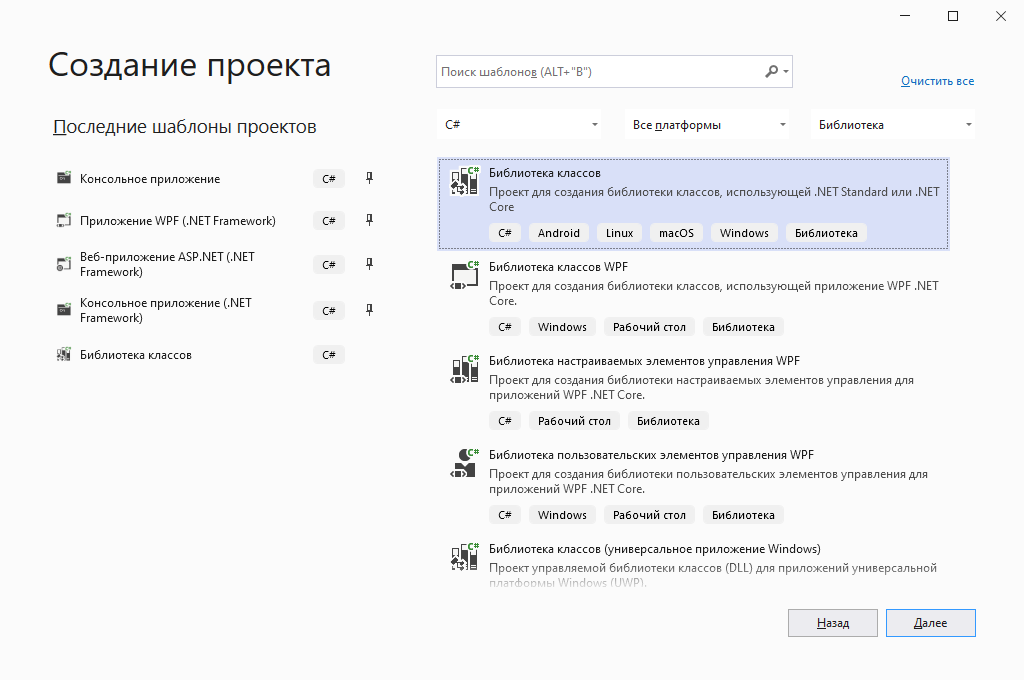


Рисунок 8. Создание проекта

По умолчанию новый проект имеет один пустой класс Class1 в файле Class1.cs. Мы можем этот файл удалить или переименовать, как нам больше нравится. Например, переименуем класс Class1 в Utils.

Далее напишем в классе Utils код:

public class Utils

{

public static string GenReceipt(string customerName, string productName, int productPrice, int quantity, DateTime date)

{

var fio = customerName.Split(' ');

var sum = productPrice \* quantity;

return $"{fio[0]} {fio[1].Substring(0, 1)}. {fio[2].Substring(0, 1)}.\_\_{productName}\_\_{sum}\_\_{date.ToString("dd-MM-yyyy")}";

}

}

Теперь скомпилируем библиотеку классов. Для этого нажмем правой кнопкой на проект библиотеки классов и в контекстном меню выберем пункт «Собрать».

После компиляции библиотеки классов в папке проекта в каталоге bin/Debug/net5.0 мы сможем найти скомпилированный файл dll. Подключим его в основной проект. Для этого в основном проекте нажмем правой кнопкой на узел «Зависимости» и в контекстном меню выберем пункт «Добавить ссылку на проект…».

Далее нам откроется окно для добавления библиотек. С помощью кнопки «Обзор…» найдем местоположение нашего файла dll и нажмем OK.

После успешного подключения библиотеки в главном проекте изменим файл Program.cs, чтобы он использовал класс Utils из библиотеки классов:

using BetarClassLibrary;

namespace BetarConsoleApp

{

class Program

{

static void Main(string[] args)

{

Console.Title = "Console";

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

Console.Clear();

Console.Write("ФИО заказчика: ");

string customerName = Console.ReadLine();

Console.Write("Наименование товара: ");

string productName = Console.ReadLine();

Console.Write("Стоимость товара: ");

int productPrice = Convert.ToInt32(Console.ReadLine());

Console.Write("Кол-во товара: ");

int quantity = Convert.ToInt32(Console.ReadLine());

Console.WriteLine();

Console.WriteLine(Utils.GenReceipt(customerName, productName, productPrice, quantity, DateTime.Now));

Console.ReadKey();

}

}

}

Запустим и проверим работу (Рис. 9):



Рисунок 9. Использование библиотеки классов в консольном приложении

4. Разработка модульных тестов (Unit-tests)

Создадим тестовый проект для метода библиотеки, где и будем создавать тесты (Рис. 10–11).

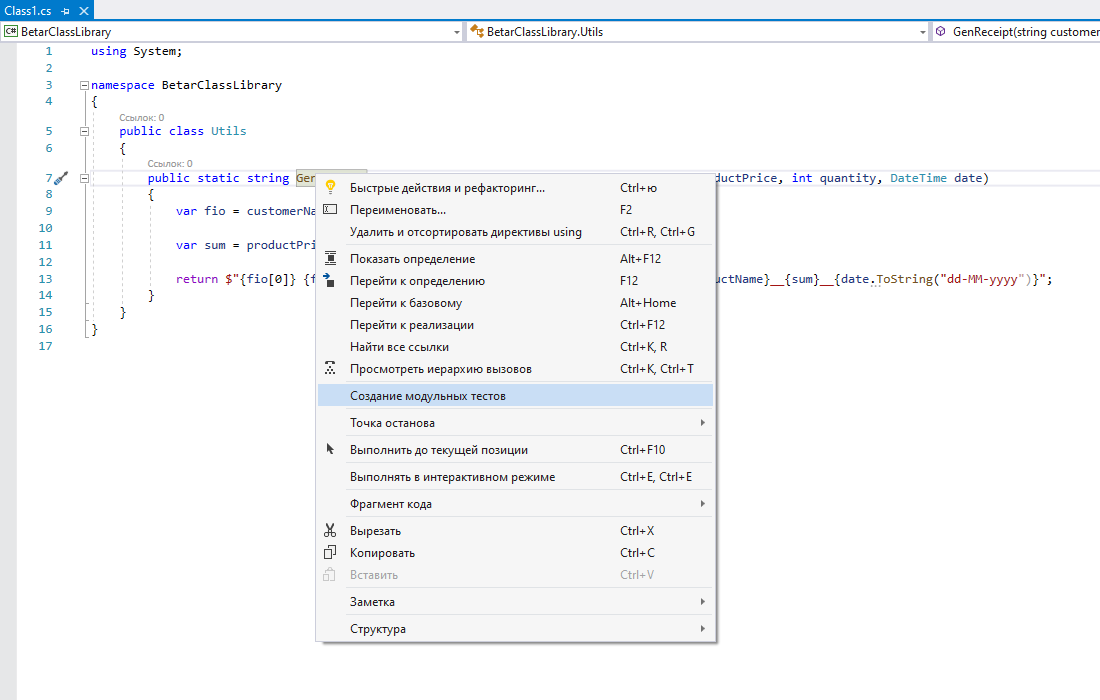


Рисунок 10. Создание модульных тестов для метода

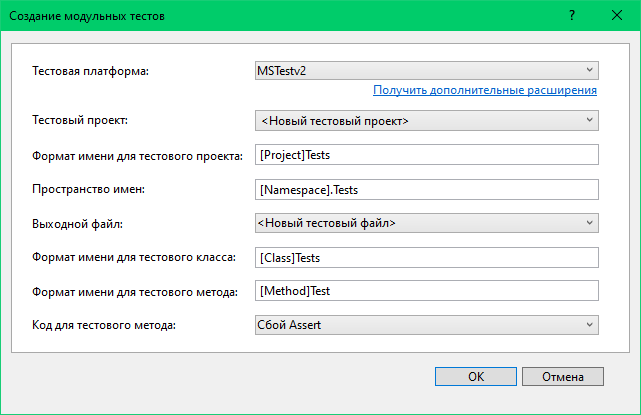


Рисунок 11. Окно создания модульных тестов

Напишем следующий код:

[TestClass()]

public class UtilsTests

{

[TestMethod()]

public void GenReceiptTest()

{

string customerName = "Федоров Виталий Андреевич";

string productName = "СГВ-15";

int productPrice = 1134;

int quantity = 20;

string expected = "Федоров В. А.\_\_СГВ-15\_\_22680\_\_07-04-2022";

string actual = Utils.GenReceipt(customerName, productName, productPrice, quantity, DateTime.Now);

Assert.AreEqual(expected, actual);

}

}

Теперь запустим тесты и проверим корректность их выполнения (Рис. 12).

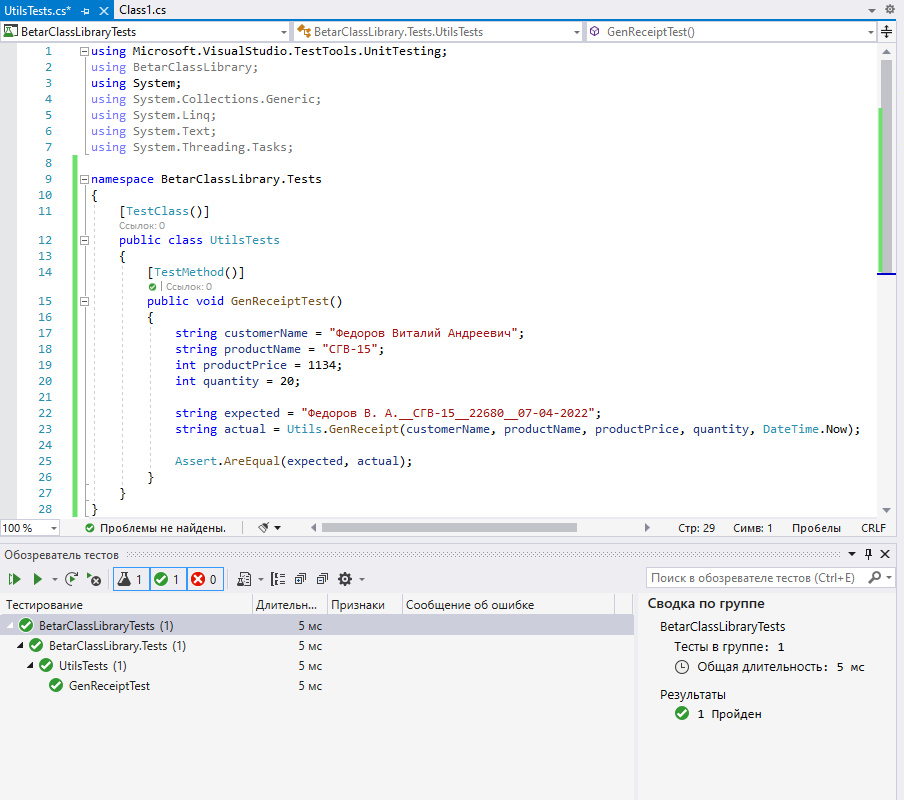


Рисунок 12. Результат выполнения тестов

5. Проектирование диаграммы Use Case

Диаграмма вариантов использования (use case diagram) — диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

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

UML (unified modeling language) — язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

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

Актер (actor) — роль объекта вне системы, который прямо взаимодействует с ее частью — конкретным элементом.

Прецедент (use case, вариант использования) — описание поведения системы, когда она взаимодействует с кем-то (или чем-то) из внешней среды.

Создание диаграммы для организации:

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

2. Определение основных групп пользователей (ролей) и размещение на диаграмме. Это те, кто будет использовать систему, и в нашем случае — это заказчик, 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.

Для администратора:

* Сформировать список продукции
* Редактировать список продукции

Для менеджера:

* Зарегистрировать заказчика
* Зарегистрировать заявку
* Выставить счет: внести поправки в заявку по желанию заказчика
* Подобрать продукцию: указать предпочтения клиента, указать строки поставки, указать границы стоимости

Для клиента и менеджера:

* Получить информацию по заявке

Для клиента:

* Подать заявку на поставку продукции
* Получить счет

4. Разграничение прецедентов между актерами размещение отношений.

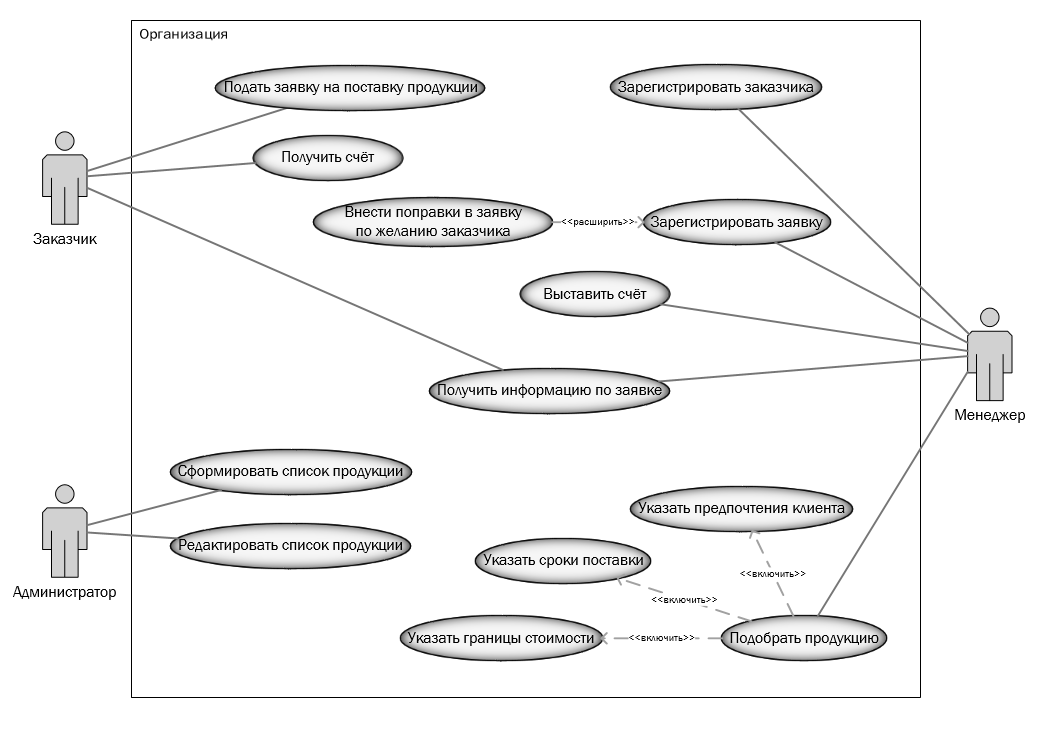


Рисунок 13. Диаграмма вариантов использования Use Case

6. Проектирование диаграммы Sequence

Диаграмма последовательности (sequence diagram) — диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определенного объекта и взаимодействие актеров ИС в рамках какого-либо определенного прецедента.

Диаграмма последовательности позволяет наглядно увидеть взаимодействие отдельных объектов.

Актер (actor) — роль объекта вне системы, который прямо взаимодействует с ее частью — конкретным элементом.

Прецедент (use case, вариант использования) — описание поведения системы, когда она взаимодействует с кем-то (или чем-то) из внешней среды.

Диаграмма последовательности позволяет изобразить поведение нескольких объектов в рамках одного прецедента. Она 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.

В ходе производственной практики было выполнено построение диаграммы последовательности ответа дилера на заявку заказчика.

Основные шаги построения диаграммы последовательности:

* Добавление основных элементов.
* Работа с сообщениями.

Создадим диаграмму последовательности, используя шаблон UML Sequence из раздела Software and Database программы Visio.

* Добавление основных элементов.

1. Добавим объекты.

Объекты обычно подписываются в формате «объект:класс» и изображаются как в виде обычных прямоугольников, так и с использованием дополнительных обозначений.

В представленной ниже диаграмме объектами являются: запрос, обозначенный прямоугольником («Заявка»), а также Дилер и Заказчик, обозначенные элементом «Актер» (Рис. 14).

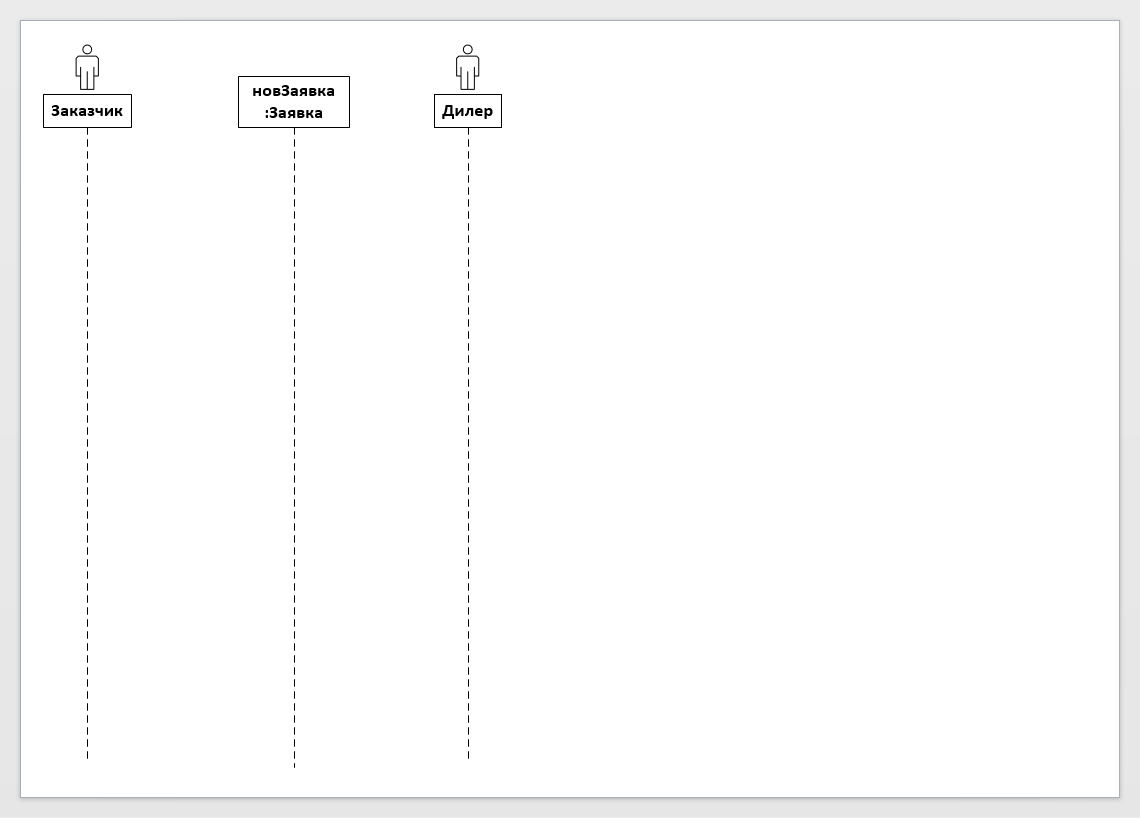


Рисунок 14. Объекты диаграммы последовательности

2. Добавим полосы активности на линии жизни (Рис. 15).

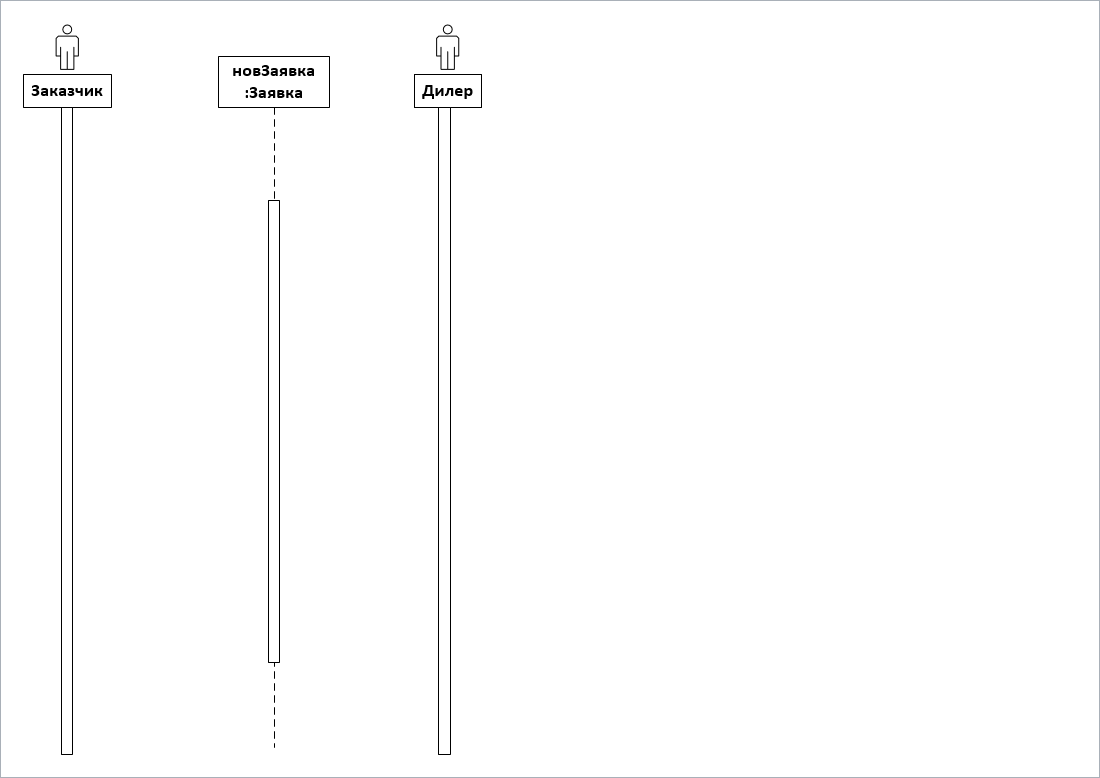


Рисунок 15. Полосы активности на линиях жизни объектов

Линия жизни (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 (вертикальный прямоугольник), показывающую интервал активности участника при взаимодействии.

* Работа с сообщениями.

1. Отобразим основные взаимодействия.

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 «нов» («new»).

На рисунке 16 изображены сообщения, показывающие взаимодействие между объектами.

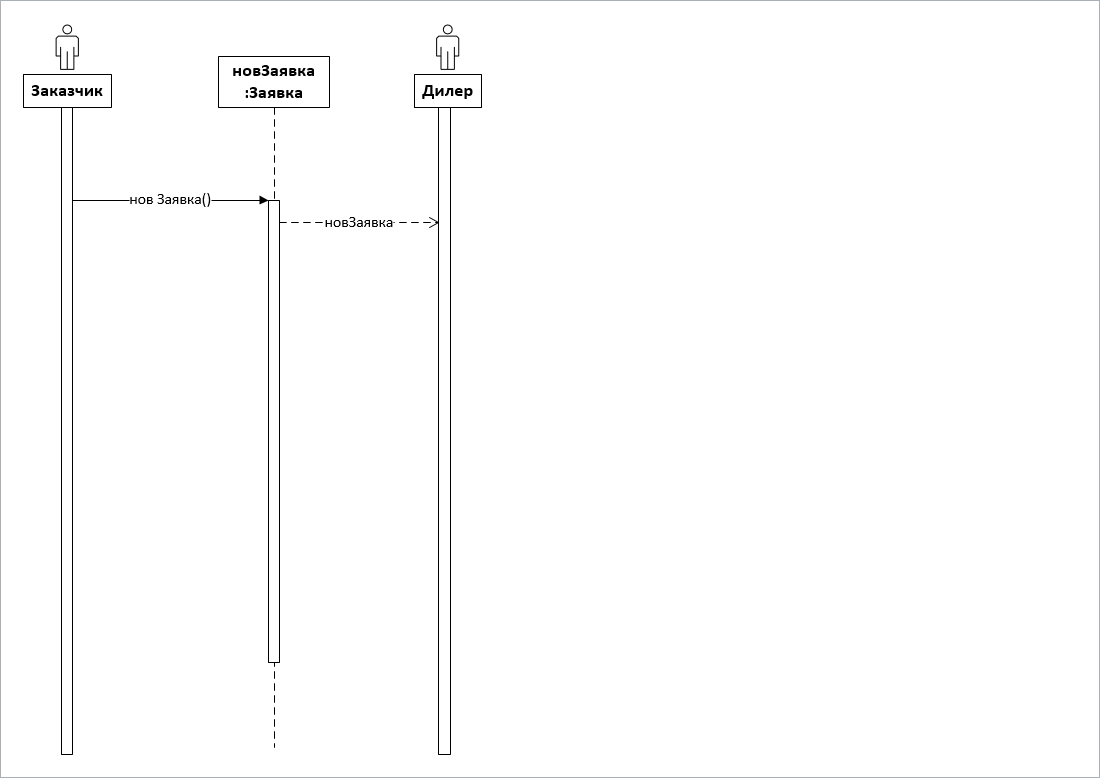


Рисунок 16. Сообщения в виде горизонтальных стрелок

2. Отобразим условия.

Условия, как и циклы, изображаются с помощью фреймов взаимодействий (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 нашей диаграмме для условной логики используется оператор «альт» («alt») и будет выполнено условие, защита которого имеет истинное значение. То есть, либо принятие заявки «ApplyRequest ()», либо откладывание заявки с указанием причины «DeclineRequest (String reason)» (Рис. 17).

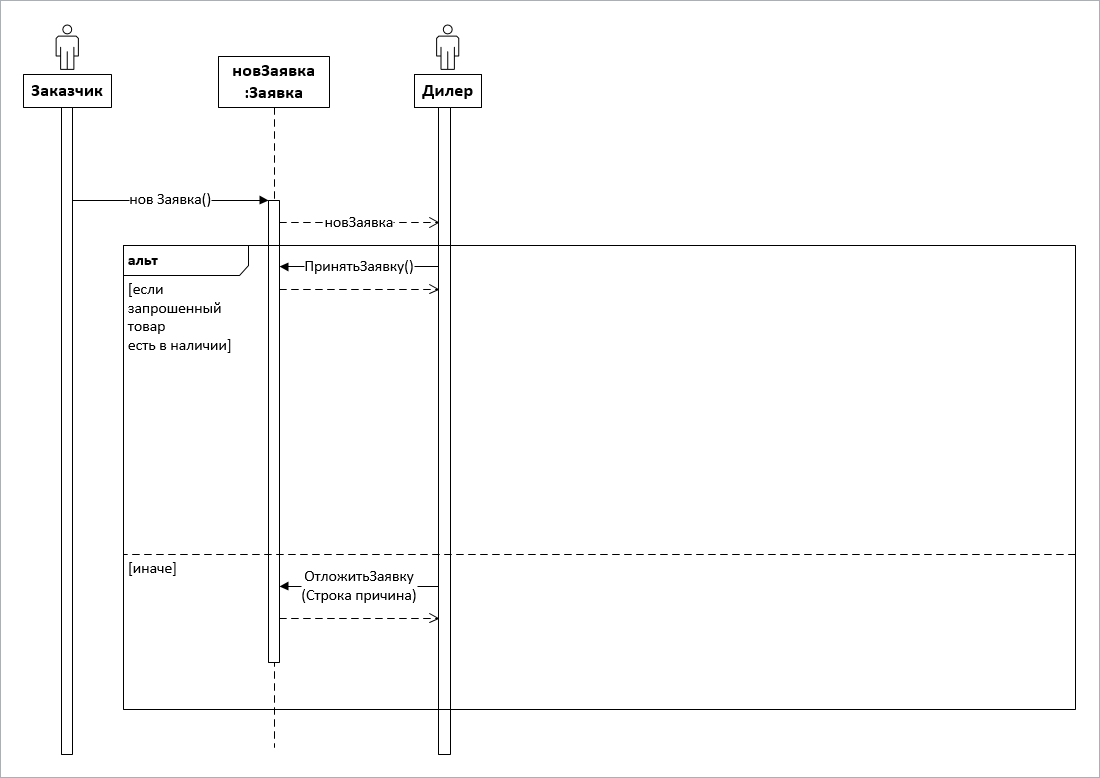


Рисунок 17. Отображение условий на диаграмме последовательности

3. Отобразим циклы.

Для отображения цикла применяется оператор «цикл» («loop») с единственным фрагментом, причем тело итерации помещается в защиту (Рис. 18).

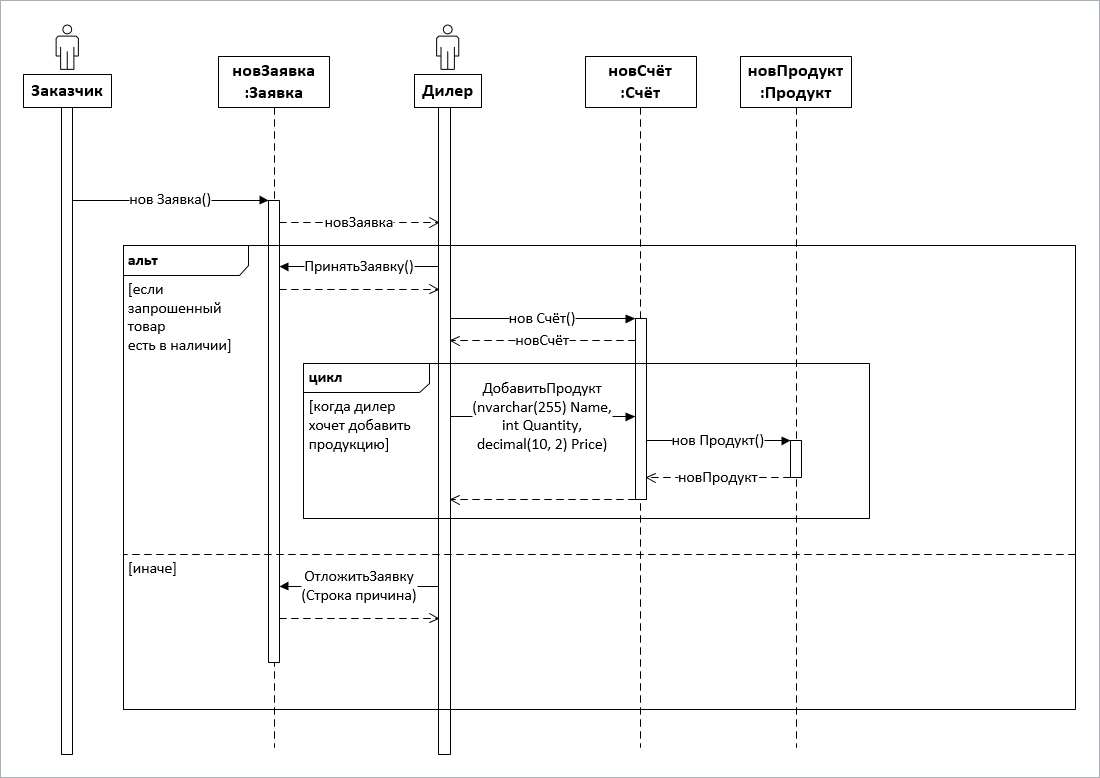


Рисунок 18. Отображение циклов на диаграмме последовательности

7. Программная работа с Excel и Word для экспорта данных в отчет

**Работа с таблицами Excel в WPF**

Реализуема возможность экспорта данных из приложения для визуализации платежей заказчика в таблицу Excel. Платежи каждого заказчика будут экспортироваться в отдельный лист, названием которого будет наименование заказчика. Платежи будут распределены по категориям продукции, причем по каждой категории будут указываться общие затраты.

Основные шаги построения приложения:

* Предварительные шаги
* Реализация экспорта
* Проверка корректной работы приложения

Предварительные шаги:

1. Подключим библиотеку для работы с Excel (Рис. 19).

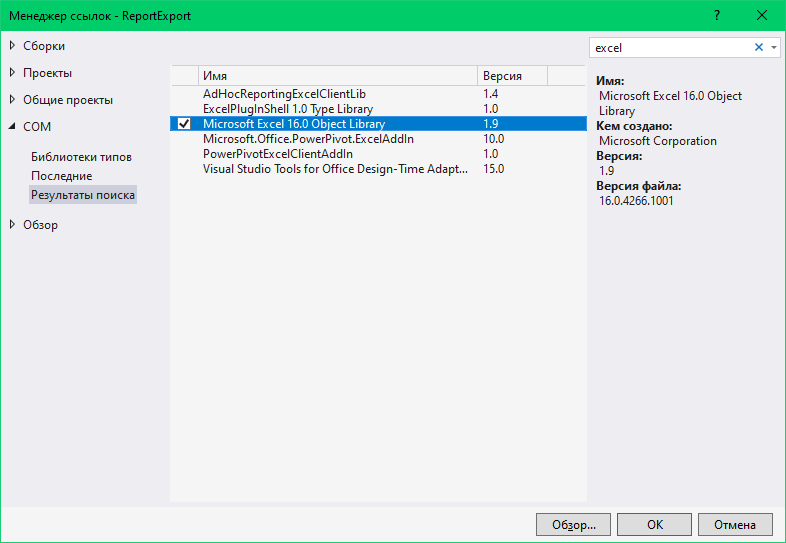


Рисунок 19. Подключение библиотеки для работы с Exel

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

2. Подключим пространство имен для работы с Excel с помощью директивы using.

using Excel = Microsoft.Office.Interop.Excel;

Реализация экспорта

1. Получим список пользователей

var allCustomers = \_context.Customers.ToList().OrderBy(p => p.Name).ToList();

2. Создадим новую книгу Excel. Для этого объявием переменную с приложением Excel, указывая количество листов (sheets) равным количеству пользователей в БД. Также добавим рабочую книгу (workbook).

var exсelApp = new Excel.Application();

exсelApp.SheetsInNewWorkbook = allCustomers.Count();

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

3. Назовём листы. 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.

for(int i = 0; i < allCustomers.Count; i++)

{

int startRowIndex = 1;

Excel.Worksheet worksheet = exсelApp.Worksheets.Item[i + 1];

worksheet.Name = allCustomers[i].Name;

}

4. Добавим название колонок. Название столбцов добавляется в верхнюю строчку листа, после чего увеличивается значение счетчика startRowIndex. При обращении к ячейке сначала указывается номер ее столбца, а затем — номер строки.

worksheet.Cells[1][startRowIndex] = "Дата продажи";

worksheet.Cells[2][startRowIndex] = "Наименование товара";

worksheet.Cells[3][startRowIndex] = "Стоимость, руб";

worksheet.Cells[4][startRowIndex] = "Количество, шт";

worksheet.Cells[5][startRowIndex] = "Сумма, руб";

startRowIndex++;

5. Сгруппируем платежи по категориям

var customersCategories = allCustomers[i].Sales.OrderBy(p => p.Date).GroupBy(p => p.Product.ProductType).OrderBy(p => p.Key.ID);

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

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

foreach (var productCategory in customersCategories)

{

Excel.Range headerRange = worksheet.Range[worksheet.Cells[1][startRowIndex], worksheet.Cells[5][startRowIndex]];

headerRange.Merge();

headerRange.Value = productCategory.Key.Name;

headerRange.HorizontalAlignment = Excel.XlHAlign.xlHAlignCenter;

headerRange.Font.Italic = true;

startRowIndex++;}

7. Добавим информацию о платежах и рассчитываем величину платежа каждой категории.

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.

foreach(var sale in productCategory)

{

worksheet.Cells[1][startRowIndex] = sale.Date;

worksheet.Cells[2][startRowIndex] = sale.Product.Name;

worksheet.Cells[3][startRowIndex] = sale.Product.Price;

worksheet.Cells[4][startRowIndex] = sale.Quantity;

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

worksheet.Cells[3][startRowIndex].NumberFormat =

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

startRowIndex++;

}

8. Рассчитаем величину общих затрат

Excel.Range sumRange = worksheet.Range[worksheet.Cells[1][startRowIndex], worksheet.Cells[4][startRowIndex]];

sumRange.Merge();

sumRange.Value = "ИТОГО:";

sumRange.HorizontalAlignment = Excel.XlHAlign.xlHAlignRight;

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

sumRange.Font.Bold = worksheet.Cells[5][startRowIndex].Font.Bold = true;

worksheet.Cells[5][startRowIndex].NumberFormat = "# ###,00";

startRowIndex++;

9. Завершаем оформление таблицы и реализацию приложения. Оформление включает в себя: добавление границ (внешних и внутренних), установку автоширины всех столбцов листа.

Excel.Range rangeBorders = worksheet.Range[worksheet.Cells[1][1],

rangeBorders.Borders[Excel.XlBordersIndex.xlEdgeBottom].LineStyle =

rangeBorders.Borders[Excel.XlBordersIndex.xlEdgeLeft].LineStyle =

rangeBorders.Borders[Excel.XlBordersIndex.xlEdgeRight].LineStyle =

rangeBorders.Borders[Excel.XlBordersIndex.xlEdgeTop].LineStyle =

rangeBorders.Borders[Excel.XlBordersIndex.xlInsideHorizontal].LineStyle =

rangeBorders.Borders[Excel.XlBordersIndex.xlInsideVertical].LineStyle = Excel.XlLineStyle.xlContinuous;

worksheet.Columns.AutoFit();

Запустим приложение и проверим корректность работы (Рис. 20).

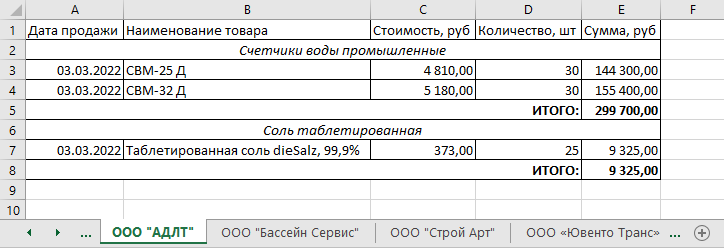


Рисунок 20. Созданный отчет Exel

**Работа с документами Word в WPF.**

Реализуем возможность экспорта данных из приложения для визуализации платежей заказчиков в документ Word. Платежи каждого заказчика будут экспортироваться на отдельную страницу, названием которой будет наименование заказчика. Платежи будут просуммированы по категориям и представлены в виде таблицы. Под таблицей будет размещена информация о наибольшем и наименьшем платежах данного заказчика.

Основные шаги построения приложения:

* Подготовительный этап
* Реализация экспорта в документ Word
* Завершение оформления документа Word

Подготовительный этап

1. Подключим библиотеку для работы с Word (Рис. 21).

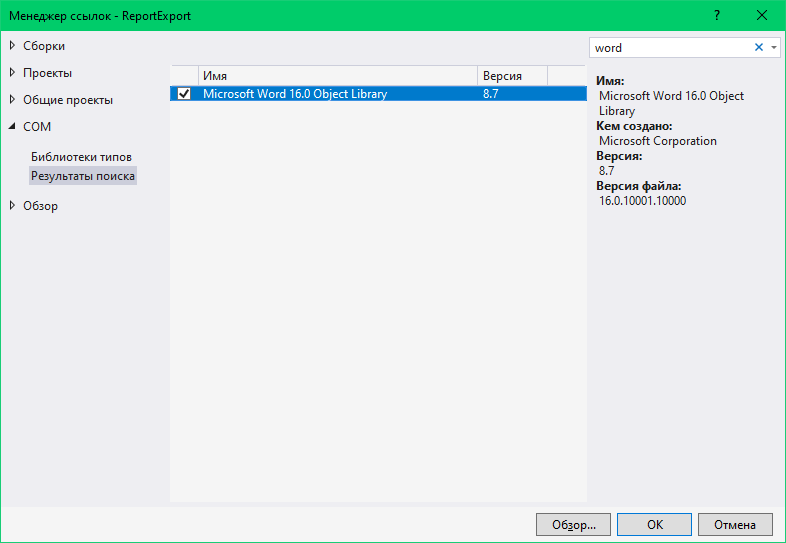


Рисунок 21. Подключение библиотеки для работы с Word

2. Подключим пространство имен для работы с Word

using Word = Microsoft.Office.Interop.Word;

Реализация экспорта в документ Word

1. Получим список пользователей и категорий

var allCustomers = \_context.Customers.ToList().OrderBy(p => p.Name).ToList();

var allCategories = \_context.ProductTypes.ToList();

2. Создадим новый документ Word.

var wordApp = new Word.Application();

Word.Document document = wordApp.Documents.Add();

3. Создаем параграф для хранения названий страниц

foreach(var customer in allCustomers)

{

Word.Paragraph customerParagraph = document.Paragraphs.Add();

Word.Range customerRange = customerParagraph.Range;

}

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

customerRange.Text = customer.Name;

customerParagraph.set\_Style("Заголовок");

customerRange.InsertParagraphAfter();

5. Добавим и отформатируем таблицу для хранения информации о платежах:

Word.Paragraph tableParagraph = document.Paragraphs.Add();

Word.Range tableRange = tableParagraph.Range;

Word.Table paymentsTable = document.Tables.Add(tableRange, allCategories.Count() + 1, 3);

paymentsTable.Borders.InsideLineStyle = paymentsTable.Borders.OutsideLineStyle = Word.WdLineStyle.wdLineStyleSingle;

paymentsTable.Range.Cells.VerticalAlignment = Word.WdCellVerticalAlignment.wdCellAlignVerticalCenter;

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.

cellRange = salesTable.Cell(1, 1).Range;

cellRange.Text = "Иконка";

cellRange = salesTable.Cell(1, 2).Range;

cellRange.Text = "Категория товара";

cellRange = salesTable.Cell(1, 3).Range;

cellRange.Text = "Сумма расходов";

salesTable.Rows[1].Range.Bold = 1;

salesTable.Rows[1].Range.ParagraphFormat.Alignment = Word.WdParagraphAlignment.wdAlignParagraphCenter;

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

7. Заполним все колонки таблицы

for(int i = 0; i < allCategories.Count(); i++)

{

var currentCategory = allCategories[i];

cellRange = salesTable.Cell(i + 2, 1).Range;

Word.InlineShape imageShape = cellRange.InlineShapes.AddPicture(AppDomain.CurrentDomain.BaseDirectory + "..\\..\\" + currentCategory.Icon);

imageShape.Width = imageShape.Height = 40;

cellRange.ParagraphFormat.Alignment = Word.WdParagraphAlignment.wdAlignParagraphCenter;

cellRange = paymentsTable.Cell(i + 2, 2).Range;

cellRange.Text = currentCategory.Name;

cellRange = salesTable.Cell(i + 2, 3).Range;

cellRange.Text = customer.Sales.ToList().Where(p => p.Product.ProductType == currentCategory).Sum(p => p.Quantity \* p.Product.Price).ToString("N2") + " руб.";

}

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

Завершение оформления документа Word

1. Добавим максимальную величину платежа. Для его поиска платежи сначала 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).

Sale maxSale = customer.Sales.OrderByDescending(p => p.Product.Price \* p.Quantity).FirstOrDefault();

if(maxSale != null)

{

Word.Paragraph maxSaleParagraph = document.Paragraphs.Add();

Word.Range maxSaleRange = maxSaleParagraph.Range;

maxSaleRange.Text = $"Наибольший платеж - {maxSale.Product.Name} за {(maxSale.Product.Price \* maxSale.Quantity).ToString("N2")} руб. от {maxSale.Date.ToString("dd.MM.yyyy")}";

maxSaleParagraph.set\_Style("Выделенная цитата");

maxSaleRange.Font.Color = Word.WdColor.wdColorDarkRed;

maxSaleRange.InsertParagraphAfter();

}

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

Sale minSale = customer.Sales.OrderBy(p => p.Product.Price \* p.Quantity).FirstOrDefault();

if (minSale != null)

{

Word.Paragraph minSaleParagraph = document.Paragraphs.Add();

Word.Range minSaleRange = minSaleParagraph.Range;

minSaleRange.Text = $"Наименьший платеж - {minSale.Product.Name} за {(minSale.Product.Price \* minSale.Quantity).ToString("N2")} руб. от {maxSale.Date.ToString("dd.MM.yyyy")}";

minSaleParagraph.set\_Style("Выделенная цитата");

minSaleRange.Font.Color = Word.WdColor.wdColorDarkGreen;

minSaleRange.InsertParagraphAfter();

}

Результат (Рис. 22).

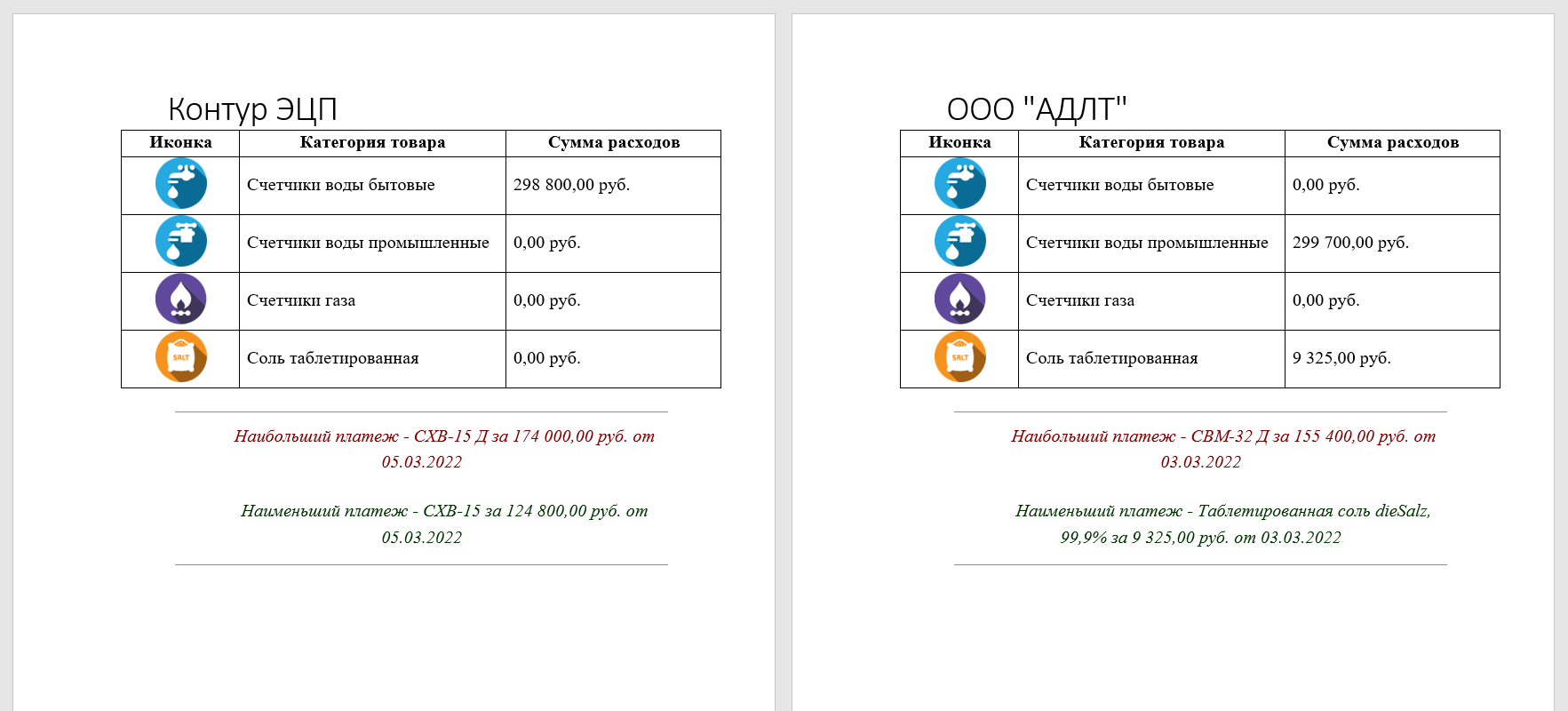


Рисунок 22. Созданный отчет Word

Заключение

В ходе прохождения производственной практической подготовки в АО «МЕТТЭМ-БЕТАР» было выполнено индивидуальное задание по производственной практике и достигнута цель по приобретению знаний, умений и навыков по:

* Проектированию ERD, созданию словаря данных, разработке базы данных по ERD и импорту данных;
* Созданию объектов базы данных;
* Подключению базы данных к API, созданию контроллера и интерфейса;
* Разработке библиотеки классов;
* Созданию интернет запросов и работе с файлами JSON;
* Проектированию требований;
* Проектированию UML-диаграмм;
* Созданию отчета;
* Разработке модульных тестов (Unit-tests);
* Созданию пояснительной записки.

Освоены следующие профессиональные компетенции:

ПК 1.1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием.

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК 1.4. Выполнять тестирование программных модулей.

ПК 1.5. Осуществлять рефакторинг и оптимизацию программного кода.

ПК 1.6. Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 2.1. Разрабатывать требования к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент.

ПК 2.2. Выполнять интеграцию модулей в программное обеспечение.

ПК 2.3. Выполнять отладку программного модуля с использованием специализированных программных средств.

ПК 2.4. Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.

ПК 2.5. Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования.

Освоены следующие общие компетенции:

ОК 1. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 2. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

ОК 3. Планировать и реализовывать собственное профессиональное и личностное развитие.

ОК 4. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.

ОК 5. Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.

ОК 6. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, применять стандарты антикоррупционного поведения.

ОК 7. Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.

ОК 8. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.

ОК 9. Использовать информационные технологии в профессиональной деятельности.

ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языке.

ОК 11. Использовать знания по финансовой грамотности, планировать предпринимательскую деятельность в профессиональной сфере.

Список литературы

1. GitHub репозиторий — URL https://github.com/MeoW-2020-2/Practice\_8sem
2. Обучение работе с Visio — URL https://support.microsoft.com/ru-ru/office/обучение-работе-с-visio-e058bcfa-1d90-4653-afc6-e84d54cf94a6
3. Программные решения для бизнеса — Национальная сборная Worldskills Россия — URL https://nationalteam.worldskills.ru/skills/programmnye-resheniya-dlya-biznesa/
4. Продукция — URL http://betar.ru/catalog/
5. Счетчики воды бытовые, промышленные, Счетчики газа, тепла и электричества, Меттэм-Бетар, 8(985)764-9834, www.betar-moskva.ru — URL http://www.betar-moskva.ru/production.htm
6. https://metanit.com/sql/sqlserver/
7. https://metanit.com/sharp/tutorial/3.46.php