ЛАБОРАТОРНАЯ РАБОТА №4.
РАБОТА С ОФИСНЫМИ ПАКЕТАМИ

СОДЕРЖАНИЕ

Цель

Изучить работу с офисными пакетами. Получать данные для отчетов, сохранять результаты в различные форматы файлов (doc, xls, pdf). Выводить отчеты на формы.

Задание

1. Создать ветку от ветки третьей лабораторной.
2. *Требуется в приложение формировать следующие отчеты:*
    a. *Выгружать список всех компонент в формате электронного текстового документа*
    б. *Выводить сводную информацию о компонентах с изделиями, в которых они используются на форме и выгружать в виде электронной таблицы.*
    в. *Выводить список заказов за определенный период на форме и выгружать в формате PDF.*
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Проектирование

В первую очередь надо определиться с шаблонами отчетов документов. Для первого документа достаточно заголовка, типа «Список компонент» и далее под ним выводим названия компонент по одному на строку (рисунок 4.1).

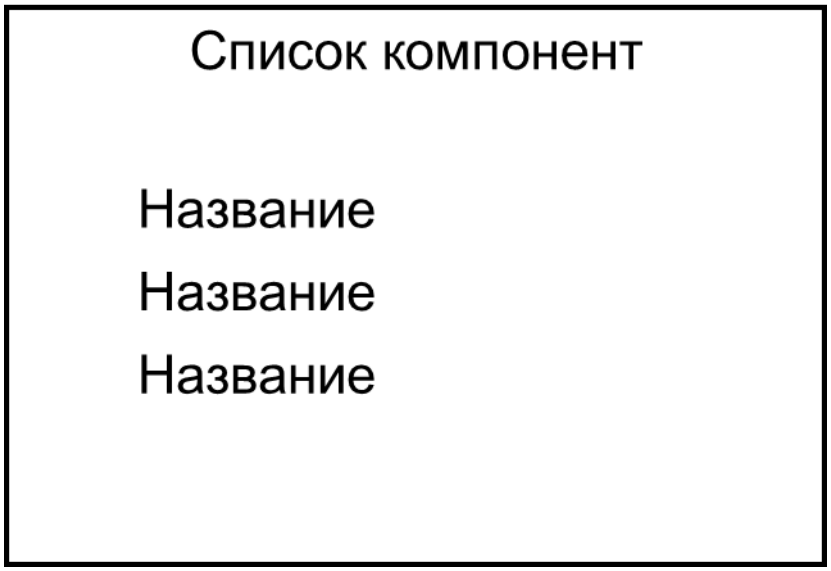


Рисунок 4.1 – Шаблон документа «Список компонент»

Второй документ также будет содержать заголовок, идентичный предыдущему и таблицу из 3-х колонок (рисунок 4.2). Для каждого компонента алгоритм заполнения следующий:

1. В первой строке в первую ячейку выводится название компонента.
2. Начинается цикл по списку изделий, где фигурирует этот компонент.
   2.1. В новой строке во второй ячейке выводится название изделия, а в третьей строке количество компонента в изделии.
3. В последней строке в первой ячейке выводится слово «Итого», а в третьей ячейке сумма по всем количествам компонента в изделиях.

Список компонент		
Компонент		
	Изделие	Количество
	Изделие	Количество
Итого		Сумма
Компонент		

Рисунок 4.2 – Шаблон документа «Сводная информация о компонентах с изделиями»

Третий документ будет содержать заголовок, в котором выводится фраза «Список заказов». Под ним будет подзаголовок с выводом периода, за который выполнялась выборка заказов в виде «с <дата начала периода> по <дата окончания периода>». Под ними выводится таблица с колонками: «Номер заказа», «Дата заказа», «Изделие», «Количество». После таблицы выводится строка «Итого» и итоговая сумма по полю «Сумма» (рисунок 4.3).

Рисунок 4.3 – Шаблон документа «Список заказов»

Следующий этап – определиться с данными, которые требуются для формирования отчетов. Для первого и второго отчетов потребуется только путь и название файла, в котором следует сохранить данные отчетов. Для третьего отчета помимо пути и названия файла потребуются 2 даты (начало и окончания периода) для выбора заказов. Для вывода данных также введем 2 класса-модели:

- Для вывода сводной информации о компонентах с изделиями. В модели будет 2 свойства: название компонента и список с указанием названия изделия и в каком количестве компонент требуется в изделии.
- Для вывода списка заказов. Создадим усеченную версию модели сущности «Заказ» с требуемым перечнем полей.

Для первого документа будет достаточно view-модели сущности «Компонент». В принципе, для третьего документа также можно было бы обойтись view-моделью сущности «Заказ», но решили ввести дополнительную модель, чтобы иметь возможность быстро расширить модель для отчета новыми параметрами в случае необходимости.

Далее следует определится с логикой для отчетов. Логика для отчетов должна включать в себя 2 метода для получения данных:

- для вывода сводной информации о компонентах с изделиями;
- для вывода списка заказов за период.

Это необходимо так как эти данные с этих методов будут использоваться не только в файлах, но и выводится на формах, так что логично вынести в отдельные методы логику их получения. Также логика для отчетов должна включать в себя 3 метода формирования файлов с отчетами, по одному на каждый отчет.

В desktop-приложении потребуется 2 формы, для вывода сводной информации о компонентах с изделиями и вывода списка заказов за период. Для получения первого документа отдельной формы не потребуется, будем вызывать диалоговое окно для выбора места хранения файла и его названия прямо с основной формы.

Последний шаг – определится с методами создания отчетов. Тут тоже не все так просто. Так как существует много различных решений для создания документов нужных нам форматов, то следует задать абстракции формирования документов под каждый формат, без привязки к непосредственно к способу, например, создания файла или добавления в него информации. А уже непосредственно эти действия (создание, добавление, сохранение) расписать в реализациях с применением конкретных решений для нужных форматов файлов.

Для офисных пакетов (Word, Excel) рассмотрим 2 варианта:

- библиотеки Interop;
- библиотеки DocumentFormat.OpenXml.

У каждой из них есть свои плюсы и минусы. Для первых – возможность сохранять в любых форматах (включая старые, например, 2003) и довольно большой набор исходных кодов для примеров, но, при этом, обязательное условие работы – пакет офиса должен быть установлен на компьютере, где будет формироваться документ. Для работы второй библиотеки не требуется предустановленный офисный пакет, но, она не может сохранять документ в старом формате, так как формирует, по сути, xml-файлы, в которых хранятся данные документа в версиях, начиная с 2007. Также, к недостаткам второй библиотеки можно отнести более сложный процесс настройки форматирования документов из-за специфики хранения данных в xml-формате и относительно небольшого числа исходных кодов для примеров. Однако, возможность формирования отчетов без предустановленного офисного пакета делает вторую библиотек более предпочтительным.

Для работы с pdf также рассмотрим 2 варианта:

- библиотека iTextSharp;
- библиотека migraDoc.

К преимуществам первого можно отнести широкую распространенность и простоту разработки. Однако, есть проблемы при создании отчета с использованием кириллицы. Для этого в файловой системе должен располагаться файл с требуемым шрифтом, что может быть не очень удобно при использовании на разных компьютерах. Вторая библиотека не требует наличия отдельного файла со шрифтом и способна напрямую создавать документ с кириллицей. Поэтому будем использовать ее при создании отчета.

Подведем итог. Для отчетов в формате Word и Excel сделаем реализацию с использованием библиотеки OpenXML, а для отчета в формате Pdf – MigraDoc.

Реализация

Начнем с классов-моделей данных от пользователя. В проекте **AbstractShopContracts** создадим в BindingModel создадим класс для

получения данных для отчетов (путь до файла и даты периодов для отчета по заказам) (листинг 4.1). Можно было сделать и два отдельных класса, как планировалось на этапе проектирования, но можно и так. Во ViewModels добавим 2 класса, первый будет возвращать данные для отчета по компонентам (листинг 4.2), второй – по заказам (листинг 4.3). Так как у нас уже есть классы OrderViewModel и ProductComponentViewModel, то для отчетов имя классов зададим иные, приписав им приставку Report.

```
namespace AbstractShopContracts.BindingModels
{
    public class ReportBindingModel
    {
        public string FileName { get; set; } = string.Empty;

        public DateTime? DateFrom { get; set; }

        public DateTime? DateTo { get; set; }
    }
}
```

Листинг 4.1 – Класс ReportBindingModel

```
namespace AbstractShopContracts.ViewModels
{
    public class ReportProductComponentViewModel
    {
        public string ComponentName { get; set; } = string.Empty;

        public int TotalCount { get; set; }

        public List<Tuple<string, int>> Products { get; set; } = new();
    }
}
```

Листинг 4.2 – Класс ReportProductComponentViewModel

```
namespace AbstractShopContracts.ViewModels
{
    public class ReportOrdersViewModel
    {
        public int Id { get; set; }

        public DateTime DateCreate { get; set; }

        public string ProductName { get; set; } = string.Empty;

        public double Sum { get; set; }
    }
}
```

Листинг 4.3 – Класс ReportOrdersViewModel

Создадим интерфейс IReportLogic, где будут описаны методы работы с отчетами (листинг 4.4).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.ViewModels;
```

```
using System.Collections.Generic;

namespace AbstractShopContracts.BusinessLogicsContracts
{
    public interface IReportLogic
    {
        /// <summary>
        /// Получение списка компонент с указанием, в каких изделиях используются
        /// </summary>
        /// <returns></returns>
        List<ReportProductComponentViewModel> GetProductComponent();

        /// <summary>
        /// Получение списка заказов за определенный период
        /// </summary>
        /// <param name="model"></param>
        /// <returns></returns>
        List<ReportOrdersViewModel> GetOrders(ReportBindingModel model);

        /// <summary>
        /// Сохранение компонент в файл-Word
        /// </summary>
        /// <param name="model"></param>
        void SaveComponentsToWordFile(ReportBindingModel model);

        /// <summary>
        /// Сохранение компонент с указаеним продуктов в файл-Excel
        /// </summary>
        /// <param name="model"></param>
        void SaveProductComponentToExcelFile(ReportBindingModel model);

        /// <summary>
        /// Сохранение заказов в файл-Pdf
        /// </summary>
        /// <param name="model"></param>
        void SaveOrdersToPdfFile(ReportBindingModel model);
    }
}
```

Листинг 4.4 – Интерфейс IReportLogic

Для создания отчета по первому документу (doc-файл) потребуется имя файла, список компонентов и название файла. Сделаем под это класс (листинг 4.5). Далее сделаем абстрактный класс для создания отчета. Абстрактный класс в данном случае будет лучше интерфейса, так как в нем можно будет сразу задать логику формирования файла с обработкой нужных данных, оставив открытым лишь способ наполнения файла. Для работы логики потребуется еще дополнительный класс для передачи данных при создании абзаца (листинг 4.6), дополнительный класс для описания свойств абзаца (листинг 4.7) и перечисление типа выравнивания (листинг 4.8).

```
using AbstractShopContracts.ViewModels;

namespace AbstractShopBusinessLogic.OfficePackage.HelperModels
{
    public class WordInfo
```

```
    {
        public string FileName { get; set; } = string.Empty;

        public string Title { get; set; } = string.Empty;

        public List<ComponentViewModel> Components { get; set; } = new();
    }
}
```

Листинг 4.5 – Класс WordInfo

```
namespace AbstractShopBusinessLogic.OfficePackage.HelperModels
{
    public class WordParagraph
    {
        public List<(string, WordTextProperties)> Texts { get; set; } = new();

        public WordTextProperties? TextProperties { get; set; }
    }
}
```

Листинг 4.6 – Класс WordParagraph

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;

namespace AbstractShopBusinessLogic.OfficePackage.HelperModels
{
    public class WordTextProperties
    {
        public string Size { get; set; } = string.Empty;

        public bool Bold { get; set; }

        public WordJustificationType JustificationType { get; set; }
    }
}
```

Листинг 4.7 – Класс WordTextProperties

```
namespace AbstractShopBusinessLogic.OfficePackage.HelperEnums
{
    public enum WordJustificationType
    {
        Center,

        Both
    }
}
```

Листинг 4.8 – Перечисление WordJustificationType

Логика основного метода абстрактного класса будет такова: создать файл, записать в него заголовок, потом пройтись по записям компонент и каждый записать в файл, сохранить итоговый файл. Каждый из перечисленных пунктов выделяются в абстрактные методы, которые должен будет реализовать класс-наследник, чтобы можно было формировать отчет (листинг 4.9).

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;
using AbstractShopBusinessLogic.OfficePackage.HelperModels;
```

```
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.OfficePackage
{
    public abstract class AbstractSaveToWord
    {
        public void CreateDoc(WordInfo info)
        {
            CreateWord(info);

            CreateParagraph(new WordParagraph
            {
                Texts = new List<(string, WordTextProperties)> { (info.Title, new
WordTextProperties { Bold = true, Size = "24", }) },
                TextProperties = new WordTextProperties
                {
                    Size = "24",
                    JustificationType = WordJustificationType.Center
                }
            });

            foreach (var component in info.Components)
            {
                CreateParagraph(new WordParagraph
                {
                    Texts = new List<(string, WordTextProperties)> {
(component.ComponentName, new WordTextProperties { Size = "24", }) },
                    TextProperties = new WordTextProperties
                    {
                        Size = "24",
                        JustificationType = WordJustificationType.Both
                    }
                });
            }

            SaveWord(info);
        }

        /// <summary>
             /// Создание doc-файла
             /// </summary>
             /// <param name="info"></param>
        protected abstract void CreateWord(WordInfo info);

        /// <summary>
        /// Создание абзаца с текстом
        /// </summary>
        /// <param name="paragraph"></param>
        /// <returns></returns>
        protected abstract void CreateParagraph(WordParagraph paragraph);

        /// <summary>
             /// Сохранение файла
             /// </summary>
             /// <param name="info"></param>
        protected abstract void SaveWord(WordInfo info);
    }
}
```

Листинг 4.9 – Абстрактный класс AbstractSaveToWord

Сделаем реализацию абстрактного класса AbstractSaveToWord с использованием библиотеки DocumentFormat.OpenXml (листинг 4.10).

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;
using AbstractShopBusinessLogic.OfficePackage.HelperModels;
using DocumentFormat.OpenXml;
using DocumentFormat.OpenXml.Packaging;
using DocumentFormat.OpenXml.Wordprocessing;

namespace AbstractShopBusinessLogic.OfficePackage.Implements
{
    public class SaveToWord : AbstractSaveToWord
    {
        private WordprocessingDocument? _wordDocument;

        private Body? _docBody;

        /// <summary>
        /// Получение типа выравнивания
        /// </summary>
        /// <param name="type"></param>
        /// <returns></returns>
        private static JustificationValues
GetJustificationValues(WordJustificationType type)
        {
            return type switch
            {
                WordJustificationType.Both => JustificationValues.Both,
                WordJustificationType.Center => JustificationValues.Center,
                _ => JustificationValues.Left,
            };
        }

        /// <summary>
        /// Настройки страницы
        /// </summary>
        /// <returns></returns>
        private static SectionProperties CreateSectionProperties()
        {
            var properties = new SectionProperties();

            var pageSize = new PageSize
            {
                Orient = PageOrientationValues.Portrait
            };

            properties.AppendChild(pageSize);

            return properties;
        }

        /// <summary>
        /// Задание форматирования для абзаца
        /// </summary>
        /// <param name="paragraphProperties"></param>
        /// <returns></returns>
        private static ParagraphProperties?
CreateParagraphProperties(WordTextProperties? paragraphProperties)
        {
            if (paragraphProperties == null)
            {
                return null;
            }

            var properties = new ParagraphProperties();

            properties.AppendChild(new Justification()
            {
```

```
                Val =
GetJustificationValues(paragraphProperties.JustificationType)
            });

            properties.AppendChild(new SpacingBetweenLines
            {
                LineRule = LineSpacingRuleValues.Auto
            });

            properties.AppendChild(new Indentation());

            var paragraphMarkRunProperties = new ParagraphMarkRunProperties();
            if (!string.IsNullOrEmpty(paragraphProperties.Size))
            {
                paragraphMarkRunProperties.AppendChild(new FontSize { Val =
paragraphProperties.Size });
            }
            properties.AppendChild(paragraphMarkRunProperties);

            return properties;
        }

        protected override void CreateWord(WordInfo info)
        {
            _wordDocument = WordprocessingDocument.Create(info.FileName,
WordprocessingDocumentType.Document);
            MainDocumentPart mainPart = _wordDocument.AddMainDocumentPart();
            mainPart.Document = new Document();
            _docBody = mainPart.Document.AppendChild(new Body());
        }

        protected override void CreateParagraph(WordParagraph paragraph)
        {
            if (_docBody == null || paragraph == null)
            {
                return;
            }
            var docParagraph = new Paragraph();


docParagraph.AppendChild(CreateParagraphProperties(paragraph.TextProperties));

            foreach (var run in paragraph.Texts)
            {
                var docRun = new Run();

                var properties = new RunProperties();
                properties.AppendChild(new FontSize { Val = run.Item2.Size });
                if (run.Item2.Bold)
                {
                    properties.AppendChild(new Bold());
                }
                docRun.AppendChild(properties);

                docRun.AppendChild(new Text { Text = run.Item1, Space =
SpaceProcessingModeValues.Preserve });

                docParagraph.AppendChild(docRun);
            }

            _docBody.AppendChild(docParagraph);
        }

        protected override void SaveWord(WordInfo info)
        {
```

```
            if (_docBody == null || _wordDocument == null)
            {
                return;
            }
            _docBody.AppendChild(CreateSectionProperties());

            _wordDocument.MainDocumentPart!.Document.Save();

            _wordDocument.Close();
        }
    }
}
```

Листинг 4.10 – Класс SaveToWord

Перейдем к отчету для excel. Для работы потребуется ряд дополнительных классов и перечисление (листинг 4.11-4.14).

```
using AbstractShopContracts.ViewModels;

namespace AbstractShopBusinessLogic.OfficePackage.HelperModels
{
    public class ExcelInfo
    {
        public string FileName { get; set; } = string.Empty;

        public string Title { get; set; } = string.Empty;

        public List<ReportProductComponentViewModel> ProductComponents { get;
set; } = new();
    }
}
```

Листинг 4.11 – Класс ExcelInfo

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;

namespace AbstractShopBusinessLogic.OfficePackage.HelperModels
{
    public class ExcelCellParameters
    {
        public string ColumnName { get; set; } = string.Empty;

        public uint RowIndex { get; set; }

        public string Text { get; set; } = string.Empty;

        public string CellReference => $"{ColumnName}{RowIndex}";

        public ExcelStyleInfoType StyleInfo { get; set; }
    }
}
```

Листинг 4.12 – Класс ExcelCellParameters

```
namespace AbstractShopBusinessLogic.OfficePackage.HelperModels
{
    public class ExcelMergeParameters
    {
        public string CellFromName { get; set; } = string.Empty;

        public string CellToName { get; set; } = string.Empty;

        public string Merge => $"{CellFromName}:{CellToName}";
    }
```

```
}
```

Листинг 4.13 – Класс ExcelMergeParameters

```
namespace AbstractShopBusinessLogic.OfficePackage.HelperEnums
{
    public enum ExcelStyleInfoType
    {
        Title,

        Text,

        TextWithBroder
    }
}
```

Листинг 4.14 – Перечисление ExcelStyleInfoType

Сделаем аналогичный абстрактный класс. Логику формирования документа уже обсуждали, так что не будем на этом акцентировать внимание (листинг 4.15).

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;
using AbstractShopBusinessLogic.OfficePackage.HelperModels;

namespace AbstractShopBusinessLogic.OfficePackage
{
    public abstract class AbstractSaveToExcel
    {
            /// <summary>
            /// Создание отчета
            /// </summary>
            /// <param name="info"></param>
        public void CreateReport(ExcelInfo info)
        {
            CreateExcel(info);

                InsertCellInWorksheet(new ExcelCellParameters
                {
                    ColumnName = "A",
                    RowIndex = 1,
                    Text = info.Title,
                    StyleInfo = ExcelStyleInfoType.Title
                });

                MergeCells(new ExcelMergeParameters
                {
                    CellFromName = "A1",
                    CellToName = "C1"
                });

                uint rowIndex = 2;
                foreach (var pc in info.ProductComponents)
                {
                    InsertCellInWorksheet(new ExcelCellParameters
                    {
                        ColumnName = "A",
                        RowIndex = rowIndex,
                        Text = pc.ComponentName,
                        StyleInfo = ExcelStyleInfoType.Text
                    });
                    rowIndex++;
```

```
                        foreach (var product in pc.Products)
                        {
                                InsertCellInWorksheet(new ExcelCellParameters
                                {
                                        ColumnName = "B",
                                        RowIndex = rowIndex,
                                        Text = product.Item1,
                                        StyleInfo =
ExcelStyleInfoType.TextWithBroder
                                });

                                InsertCellInWorksheet(new ExcelCellParameters
                                {
                                        ColumnName = "C",
                                        RowIndex = rowIndex,
                                        Text = product.Item2.ToString(),
                                        StyleInfo =
ExcelStyleInfoType.TextWithBroder
                                });

                                rowIndex++;
                        }

                        InsertCellInWorksheet(new ExcelCellParameters
                        {
                                ColumnName = "A",
                                RowIndex = rowIndex,
                                Text = "Итого",
                                StyleInfo = ExcelStyleInfoType.Text
                        });
                        InsertCellInWorksheet(new ExcelCellParameters
                        {
                                ColumnName = "C",
                                RowIndex = rowIndex,
                                Text = pc.TotalCount.ToString(),
                                StyleInfo = ExcelStyleInfoType.Text
                        });
                        rowIndex++;
                }

                SaveExcel(info);
        }

        /// <summary>
        /// Создание excel-файла
        /// </summary>
        /// <param name="info"></param>
    protected abstract void CreateExcel(ExcelInfo info);

        /// <summary>
        /// Добавляем новую ячейку в лист
        /// </summary>
        /// <param name="cellParameters"></param>
        protected abstract void InsertCellInWorksheet(ExcelCellParameters
excelParams);

        /// <summary>
        /// Объединение ячеек
        /// </summary>
        /// <param name="mergeParameters"></param>
        protected abstract void MergeCells(ExcelMergeParameters excelParams);

        /// <summary>
        /// Сохранение файла
        /// </summary>
```

```
            /// <param name="info"></param>
            protected abstract void SaveExcel(ExcelInfo info);
        }
}
```

Листинг 4.15 – Абстрактный класс AbstractSaveToExcel

Реализацию этого класса сделаем также с применением библиотеки DocumentFormat.OpenXml (листинг 4.16).

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;
using AbstractShopBusinessLogic.OfficePackage.HelperModels;
using DocumentFormat.OpenXml;
using DocumentFormat.OpenXml.Office2010.Excel;
using DocumentFormat.OpenXml.Office2013.Excel;
using DocumentFormat.OpenXml.Packaging;
using DocumentFormat.OpenXml.Spreadsheet;

namespace AbstractShopBusinessLogic.OfficePackage.Implements
{
    public class SaveToExcel : AbstractSaveToExcel
    {
        private SpreadsheetDocument? _spreadsheetDocument;

        private SharedStringTablePart? _shareStringPart;

        private Worksheet? _worksheet;

        /// <summary>
        /// Настройка стилей для файла
        /// </summary>
        /// <param name="workbookpart"></param>
        private static void CreateStyles(WorkbookPart workbookpart)
        {
            var sp = workbookpart.AddNewPart<WorkbookStylesPart>();
            sp.Stylesheet = new Stylesheet();

            var fonts = new Fonts() { Count = 2U, KnownFonts = true };

            var fontUsual = new Font();
            fontUsual.Append(new FontSize() { Val = 12D });
            fontUsual.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()
{ Theme = 1U });
            fontUsual.Append(new FontName() { Val = "Times New Roman" });
            fontUsual.Append(new FontFamilyNumbering() { Val = 2 });
            fontUsual.Append(new FontScheme() { Val = FontSchemeValues.Minor });

            var fontTitle = new Font();
            fontTitle.Append(new Bold());
            fontTitle.Append(new FontSize() { Val = 14D });
            fontTitle.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()
{ Theme = 1U });
            fontTitle.Append(new FontName() { Val = "Times New Roman" });
            fontTitle.Append(new FontFamilyNumbering() { Val = 2 });
            fontTitle.Append(new FontScheme() { Val = FontSchemeValues.Minor });

            fonts.Append(fontUsual);
            fonts.Append(fontTitle);

            var fills = new Fills() { Count = 2U };

            var fill1 = new Fill();
            fill1.Append(new PatternFill() { PatternType = PatternValues.None });
```

```
            var fill2 = new Fill();
            fill2.Append(new PatternFill() { PatternType = PatternValues.Gray125
});

            fills.Append(fill1);
            fills.Append(fill2);

            var borders = new Borders() { Count = 2U };

            var borderNoBorder = new Border();
            borderNoBorder.Append(new LeftBorder());
            borderNoBorder.Append(new RightBorder());
            borderNoBorder.Append(new TopBorder());
            borderNoBorder.Append(new BottomBorder());
            borderNoBorder.Append(new DiagonalBorder());

            var borderThin = new Border();

            var leftBorder = new LeftBorder() { Style = BorderStyleValues.Thin };
            leftBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()
{ Indexed = 64U });

            var rightBorder = new RightBorder() { Style = BorderStyleValues.Thin
};
            rightBorder.Append(new
DocumentFormat.OpenXml.Office2010.Excel.Color() { Indexed = 64U });

            var topBorder = new TopBorder() { Style = BorderStyleValues.Thin };
            topBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()
{ Indexed = 64U });

            var bottomBorder = new BottomBorder() { Style =
BorderStyleValues.Thin };
            bottomBorder.Append(new
DocumentFormat.OpenXml.Office2010.Excel.Color() { Indexed = 64U });

            borderThin.Append(leftBorder);
            borderThin.Append(rightBorder);
            borderThin.Append(topBorder);
            borderThin.Append(bottomBorder);
            borderThin.Append(new DiagonalBorder());

            borders.Append(borderNoBorder);
            borders.Append(borderThin);

            var cellStyleFormats = new CellStyleFormats() { Count = 1U };
            var cellFormatStyle = new CellFormat() { NumberFormatId = 0U, FontId
= 0U, FillId = 0U, BorderId = 0U };

            cellStyleFormats.Append(cellFormatStyle);

            var cellFormats = new CellFormats() { Count = 3U };
            var cellFormatFont = new CellFormat() { NumberFormatId = 0U, FontId =
0U, FillId = 0U, BorderId = 0U, FormatId = 0U, ApplyFont = true };
            var cellFormatFontAndBorder = new CellFormat() { NumberFormatId = 0U,
FontId = 0U, FillId = 0U, BorderId = 1U, FormatId = 0U, ApplyFont = true,
ApplyBorder = true };
            var cellFormatTitle = new CellFormat() { NumberFormatId = 0U, FontId
= 1U, FillId = 0U, BorderId = 0U, FormatId = 0U, Alignment = new Alignment() {
Vertical = VerticalAlignmentValues.Center, WrapText = true, Horizontal =
HorizontalAlignmentValues.Center }, ApplyFont = true };

            cellFormats.Append(cellFormatFont);
            cellFormats.Append(cellFormatFontAndBorder);
            cellFormats.Append(cellFormatTitle);
```

```
            var cellStyles = new CellStyles() { Count = 1U };

            cellStyles.Append(new CellStyle() { Name = "Normal", FormatId = 0U,
BuiltinId = 0U });

            var differentialFormats = new
DocumentFormat.OpenXml.Office2013.Excel.DifferentialFormats() { Count = 0U };

            var tableStyles = new TableStyles() { Count = 0U, DefaultTableStyle =
"TableStyleMedium2", DefaultPivotStyle = "PivotStyleLight16" };

            var stylesheetExtensionList = new StylesheetExtensionList();

            var stylesheetExtension1 = new StylesheetExtension() { Uri =
"{EB79DEF2-80B8-43e5-95BD-54CBDDF9020C}" };
            stylesheetExtension1.AddNamespaceDeclaration("x14",
"http://schemas.microsoft.com/office/spreadsheetml/2009/9/main");
            stylesheetExtension1.Append(new SlicerStyles() { DefaultSlicerStyle =
"SlicerStyleLight1" });

            var stylesheetExtension2 = new StylesheetExtension() { Uri =
"{9260A510-F301-46a8-8635-F512D64BE5F5}" };
            stylesheetExtension2.AddNamespaceDeclaration("x15",
"http://schemas.microsoft.com/office/spreadsheetml/2010/11/main");
            stylesheetExtension2.Append(new TimelineStyles() {
DefaultTimelineStyle = "TimeSlicerStyleLight1" });

            stylesheetExtensionList.Append(stylesheetExtension1);
            stylesheetExtensionList.Append(stylesheetExtension2);

            sp.Stylesheet.Append(fonts);
            sp.Stylesheet.Append(fills);
            sp.Stylesheet.Append(borders);
            sp.Stylesheet.Append(cellStyleFormats);
            sp.Stylesheet.Append(cellFormats);
            sp.Stylesheet.Append(cellStyles);
            sp.Stylesheet.Append(differentialFormats);
            sp.Stylesheet.Append(tableStyles);
            sp.Stylesheet.Append(stylesheetExtensionList);
        }

        /// <summary>
        /// Получение номера стиля из типа
        /// </summary>
        /// <param name="styleInfo"></param>
        /// <returns></returns>
        private static uint GetStyleValue(ExcelStyleInfoType styleInfo)
        {
            return styleInfo switch
            {
                ExcelStyleInfoType.Title => 2U,
                ExcelStyleInfoType.TextWithBroder => 1U,
                ExcelStyleInfoType.Text => 0U,
                _ => 0U,
            };
        }

        protected override void CreateExcel(ExcelInfo info)
        {
            _spreadsheetDocument = SpreadsheetDocument.Create(info.FileName,
SpreadsheetDocumentType.Workbook);
            // Создаем книгу (в ней хранятся листы)
            var workbookpart = _spreadsheetDocument.AddWorkbookPart();
            workbookpart.Workbook = new Workbook();
```

```
            CreateStyles(workbookpart);

            // Получаем/создаем хранилище текстов для книги
            _shareStringPart =
_spreadsheetDocument.WorkbookPart!.GetPartsOfType<SharedStringTablePart>().Any()
                ?
_spreadsheetDocument.WorkbookPart.GetPartsOfType<SharedStringTablePart>().First()
                :
_spreadsheetDocument.WorkbookPart.AddNewPart<SharedStringTablePart>();

            // Создаем SharedStringTable, если его нет
            if (_shareStringPart.SharedStringTable == null)
            {
                _shareStringPart.SharedStringTable = new SharedStringTable();
            }

            // Создаем лист в книгу
            var worksheetPart = workbookpart.AddNewPart<WorksheetPart>();
            worksheetPart.Worksheet = new Worksheet(new SheetData());

            // Добавляем лист в книгу
            var sheets =
_spreadsheetDocument.WorkbookPart.Workbook.AppendChild(new Sheets());
            var sheet = new Sheet()
            {
                Id =
_spreadsheetDocument.WorkbookPart.GetIdOfPart(worksheetPart),
                SheetId = 1,
                Name = "Лист"
            };
            sheets.Append(sheet);

            _worksheet = worksheetPart.Worksheet;
        }

        protected override void InsertCellInWorksheet(ExcelCellParameters
excelParams)
        {
            if (_worksheet == null || _shareStringPart == null)
            {
                return;
            }
            var sheetData = _worksheet.GetFirstChild<SheetData>();
            if (sheetData == null)
            {
                return;
            }

            // Ищем строку, либо добавляем ее
            Row row;
            if (sheetData.Elements<Row>().Where(r => r.RowIndex! ==
excelParams.RowIndex).Any())
            {
                row = sheetData.Elements<Row>().Where(r => r.RowIndex! ==
excelParams.RowIndex).First();
            }
            else
            {
                row = new Row() { RowIndex = excelParams.RowIndex };
                sheetData.Append(row);
            }

            // Ищем нужную ячейку
            Cell cell;
```

```
            if (row.Elements<Cell>().Where(c => c.CellReference!.Value ==
excelParams.CellReference).Any())
            {
                cell = row.Elements<Cell>().Where(c => c.CellReference!.Value ==
excelParams.CellReference).First();
            }
            else
            {
                // Все ячейки должны быть последовательно друг за другом
расположены
                // нужно определить, после какой вставлять
                Cell? refCell = null;
                foreach (Cell rowCell in row.Elements<Cell>())
                {
                    if (string.Compare(rowCell.CellReference!.Value,
excelParams.CellReference, true) > 0)
                    {
                        refCell = rowCell;
                        break;
                    }
                }

                var newCell = new Cell() { CellReference =
excelParams.CellReference };
                row.InsertBefore(newCell, refCell);

                cell = newCell;
            }

            // вставляем новый текст
            _shareStringPart.SharedStringTable.AppendChild(new
SharedStringItem(new Text(excelParams.Text)));
            _shareStringPart.SharedStringTable.Save();

            cell.CellValue = new
CellValue((_shareStringPart.SharedStringTable.Elements<SharedStringItem>().Count(
) - 1).ToString());
            cell.DataType = new EnumValue<CellValues>(CellValues.SharedString);
            cell.StyleIndex = GetStyleValue(excelParams.StyleInfo);
        }

        protected override void MergeCells(ExcelMergeParameters excelParams)
        {
            if (_worksheet == null)
            {
                return;
            }
            MergeCells mergeCells;

            if (_worksheet.Elements<MergeCells>().Any())
            {
                mergeCells = _worksheet.Elements<MergeCells>().First();
            }
            else
            {
                mergeCells = new MergeCells();

                if (_worksheet.Elements<CustomSheetView>().Any())
                {
                    _worksheet.InsertAfter(mergeCells,
_worksheet.Elements<CustomSheetView>().First());
                }
                else
                {
```

```
                    _worksheet.InsertAfter(mergeCells,
_worksheet.Elements<SheetData>().First());
                }
            }

            var mergeCell = new MergeCell()
            {
                Reference = new StringValue(excelParams.Merge)
            };
            mergeCells.Append(mergeCell);
        }

        protected override void SaveExcel(ExcelInfo info)
        {
            if (_spreadsheetDocument == null)
            {
                return;
            }
            _spreadsheetDocument.WorkbookPart!.Workbook.Save();
            _spreadsheetDocument.Close();
        }
    }
}
```

Листинг 4.16 – Класс SaveToExcel

Последним пунктом будет формирование pdf-документа. Потребуется ряд вспомогательных классов и перечисление (листинг 4.17-4.20).

```
using AbstractShopContracts.ViewModels;

namespace AbstractShopBusinessLogic.OfficePackage.HelperModels
{
    public class PdfInfo
    {
        public string FileName { get; set; } = string.Empty;

        public string Title { get; set; } = string.Empty;

        public DateTime DateFrom { get; set; }

        public DateTime DateTo { get; set; }

        public List<ReportOrdersViewModel> Orders { get; set; } = new();
    }
}
```

Листинг 4.17 – Класс PdfInfo

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;

namespace AbstractShopBusinessLogic.OfficePackage.HelperModels
{
    public class PdfParagraph
    {
        public string Text { get; set; } = string.Empty;

        public string Style { get; set; } = string.Empty;

        public PdfParagraphAlignmentType ParagraphAlignment { get; set; }
    }
}
```

Листинг 4.18 – Класс PdfParagraph

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;

namespace AbstractShopBusinessLogic.OfficePackage.HelperModels
{
    public class PdfRowParameters
    {
        public List<string> Texts { get; set; } = new();

        public string Style { get; set; } = string.Empty;

        public PdfParagraphAlignmentType ParagraphAlignment { get; set; }
    }
}
```

Листинг 4.19 – Класс PdfRowParameters

```
namespace AbstractShopBusinessLogic.OfficePackage.HelperEnums
{
    public enum PdfParagraphAlignmentType
    {
        Center,

        Left,

        Rigth
    }
}
```

Листинг 4.20 – Перечисление PdfParagraphAlignmentType

Также сделаем абстрактный класс. Основной метод будет создавать документ, заголовок, таблицу, заполнять строки и сохранять файл (листинг 4.21).

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;
using AbstractShopBusinessLogic.OfficePackage.HelperModels;
using System.Collections.Generic;
using System.Linq;

namespace AbstractShopBusinessLogic.OfficePackage
{
    public abstract class AbstractSaveToPdf
    {
        public void CreateDoc(PdfInfo info)
        {
            CreatePdf(info);
            CreateParagraph(new PdfParagraph { Text = info.Title, Style =
"NormalTitle", ParagraphAlignment = PdfParagraphAlignmentType.Center });
            CreateParagraph(new PdfParagraph { Text = $"с
{info.DateFrom.ToShortDateString()} по {info.DateTo.ToShortDateString()}", Style
= "Normal", ParagraphAlignment = PdfParagraphAlignmentType.Center });

            CreateTable(new List<string> { "2cm", "3cm", "6cm", "3cm" });

            CreateRow(new PdfRowParameters
            {
                Texts = new List<string> { "Номер", "Дата заказа", "Изделие",
"Сумма" },
                Style = "NormalTitle",
                ParagraphAlignment = PdfParagraphAlignmentType.Center
            });

            foreach (var order in info.Orders)
```

```
            {
                CreateRow(new PdfRowParameters
                {
                    Texts = new List<string> { order.Id.ToString(),
order.DateCreate.ToShortDateString(), order.ProductName, order.Sum.ToString() },
                    Style = "Normal",
                    ParagraphAlignment = PdfParagraphAlignmentType.Left
                });
            }
            CreateParagraph(new PdfParagraph { Text = $"Итого: {info.Orders.Sum(x
=> x.Sum)}\t", Style = "Normal", ParagraphAlignment =
PdfParagraphAlignmentType.Rigth });

            SavePdf(info);
        }

        /// <summary>
             /// Создание doc-файла
             /// </summary>
             /// <param name="info"></param>
        protected abstract void CreatePdf(PdfInfo info);

        /// <summary>
        /// Создание параграфа с текстом
        /// </summary>
        /// <param name="title"></param>
        /// <param name="style"></param>
        protected abstract void CreateParagraph(PdfParagraph paragraph);

        /// <summary>
        /// Создание таблицы
        /// </summary>
        /// <param name="title"></param>
        /// <param name="style"></param>
        protected abstract void CreateTable(List<string> columns);

        /// <summary>
        /// Создание и заполнение строки
        /// </summary>
        /// <param name="rowParameters"></param>
        protected abstract void CreateRow(PdfRowParameters rowParameters);

        /// <summary>
             /// Сохранение файла
             /// </summary>
             /// <param name="info"></param>
        protected abstract void SavePdf(PdfInfo info);
    }
}
```

Листинг 4.21 – Абстрактный класс AbstractSaveToPdf

И создадим реализацию абстрактного класса с использованием пакета MigraDoc (листинг 4.22).

```
using AbstractShopBusinessLogic.OfficePackage.HelperEnums;
using AbstractShopBusinessLogic.OfficePackage.HelperModels;
using MigraDoc.DocumentObjectModel;
using MigraDoc.DocumentObjectModel.Tables;
using MigraDoc.Rendering;

namespace AbstractShopBusinessLogic.OfficePackage.Implements
{
    public class SaveToPdf : AbstractSaveToPdf
```

```
    {
        private Document? _document;

        private Section? _section;

        private Table? _table;

        private static ParagraphAlignment
GetParagraphAlignment(PdfParagraphAlignmentType type)
        {
            return type switch
            {
                PdfParagraphAlignmentType.Center => ParagraphAlignment.Center,
                PdfParagraphAlignmentType.Left => ParagraphAlignment.Left,
                PdfParagraphAlignmentType.Rigth => ParagraphAlignment.Right,
                _ => ParagraphAlignment.Justify,
            };
        }

        /// <summary>
        /// Создание стилей для документа
        /// </summary>
        /// <param name="document"></param>
        private static void DefineStyles(Document document)
        {
            var style = document.Styles["Normal"];
            style.Font.Name = "Times New Roman";
            style.Font.Size = 14;

            style = document.Styles.AddStyle("NormalTitle", "Normal");
            style.Font.Bold = true;
        }

        protected override void CreatePdf(PdfInfo info)
        {
            _document = new Document();
            DefineStyles(_document);

            _section = _document.AddSection();
        }

        protected override void CreateParagraph(PdfParagraph pdfParagraph)
        {
            if (_section == null)
            {
                return;
            }
            var paragraph = _section.AddParagraph(pdfParagraph.Text);
            paragraph.Format.SpaceAfter = "1cm";
            paragraph.Format.Alignment =
GetParagraphAlignment(pdfParagraph.ParagraphAlignment);
            paragraph.Style = pdfParagraph.Style;
        }

        protected override void CreateTable(List<string> columns)
        {
            if (_document == null)
            {
                return;
            }
            _table = _document.LastSection.AddTable();

            foreach (var elem in columns)
            {
                _table.AddColumn(elem);
```

```
            }
        }

        protected override void CreateRow(PdfRowParameters rowParameters)
        {
            if (_table == null)
            {
                return;
            }
            var row = _table.AddRow();
            for (int i = 0; i < rowParameters.Texts.Count; ++i)
            {
                row.Cells[i].AddParagraph(rowParameters.Texts[i]);

                if (!string.IsNullOrEmpty(rowParameters.Style))
                {
                    row.Cells[i].Style = rowParameters.Style;
                }

                Unit borderWidth = 0.5;

                row.Cells[i].Borders.Left.Width = borderWidth;
                row.Cells[i].Borders.Right.Width = borderWidth;
                row.Cells[i].Borders.Top.Width = borderWidth;
                row.Cells[i].Borders.Bottom.Width = borderWidth;

                row.Cells[i].Format.Alignment =
GetParagraphAlignment(rowParameters.ParagraphAlignment);
                row.Cells[i].VerticalAlignment = VerticalAlignment.Center;
            }
        }

        protected override void SavePdf(PdfInfo info)
        {
            var renderer = new PdfDocumentRenderer(true)
            {
                Document = _document
            };
            renderer.RenderDocument();
            renderer.PdfDocument.Save(info.FileName);
        }
    }
}
```

Листинг 4.22 – Класс SaveToPdf

Остается сделать только класс ReportLogic. Нам потребуется подключить 3 интерфейса для получения списка компонент, изделий и заказов и 3 абстрактных класса для создания отчетов. Так как заказы следует выбирать за период, класс OrderSearchModel потребуется расширить 2 параметрами (листинг 4.23) и поменять логику в реализациях получения списка заказов.

```
namespace AbstractShopContracts.SearchModels
{
    public class OrderSearchModel
    {
        public int? Id { get; set; }

        public DateTime? DateFrom { get; set; }

        public DateTime? DateTo { get; set; }
```

```
    }
}
```

Листинг 4.23 – Дополненный класс OrderSearchModel

Класс ReportLogic будет следующим (листинг 4.24).

```
using AbstractShopBusinessLogic.OfficePackage;
using AbstractShopBusinessLogic.OfficePackage.HelperModels;
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.StoragesContracts;
using AbstractShopContracts.ViewModels;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    public class ReportLogic : IReportLogic
    {
        private readonly IComponentStorage _componentStorage;

        private readonly IProductStorage _productStorage;

        private readonly IOrderStorage _orderStorage;

        private readonly AbstractSaveToExcel _saveToExcel;

        private readonly AbstractSaveToWord _saveToWord;

        private readonly AbstractSaveToPdf _saveToPdf;

        public ReportLogic(IProductStorage productStorage, IComponentStorage
componentStorage, IOrderStorage orderStorage,
            AbstractSaveToExcel saveToExcel, AbstractSaveToWord saveToWord,
AbstractSaveToPdf saveToPdf)
        {
            _productStorage = productStorage;
            _componentStorage = componentStorage;
            _orderStorage = orderStorage;

            _saveToExcel = saveToExcel;
            _saveToWord = saveToWord;
            _saveToPdf = saveToPdf;
        }

        /// <summary>
        /// Получение списка компонент с указанием, в каких изделиях используются
        /// </summary>
        /// <returns></returns>
        public List<ReportProductComponentViewModel> GetProductComponent()
        {
            var components = _componentStorage.GetFullList();

            var products = _productStorage.GetFullList();

            var list = new List<ReportProductComponentViewModel>();

            foreach(var component in components)
            {
                var record = new ReportProductComponentViewModel
                {
                    ComponentName = component.ComponentName,
                    Products = new List<Tuple<string, int>>(),
                    TotalCount = 0
                };
```

```
                foreach (var product in products)
                {
                    if(product.ProductComponents.ContainsKey(component.Id))
                    {
                        record.Products.Add(new Tuple<string,
int>(product.ProductName, product.ProductComponents[component.Id].Item2));
                        record.TotalCount +=
product.ProductComponents[component.Id].Item2;
                    }
                }

                list.Add(record);
            }

            return list;
        }

        /// <summary>
        /// Получение списка заказов за определенный период
        /// </summary>
        /// <param name="model"></param>
        /// <returns></returns>
        public List<ReportOrdersViewModel> GetOrders(ReportBindingModel model)
        {
            return _orderStorage.GetFilteredList(new OrderSearchModel { DateFrom
= model.DateFrom, DateTo = model.DateTo })
                    .Select(x => new ReportOrdersViewModel
                    {
                        Id = x.Id,
                        DateCreate = x.DateCreate,
                        ProductName = x.ProductName,
                        Sum = x.Sum
                    })
                    .ToList();
        }

        /// <summary>
        /// Сохранение компонент в файл-Word
        /// </summary>
        /// <param name="model"></param>
        public void SaveComponentsToWordFile(ReportBindingModel model)
        {
            _saveToWord.CreateDoc(new WordInfo
            {
                FileName = model.FileName,
                Title = "Список компонент",
                Components = _componentStorage.GetFullList()
            });
        }

        /// <summary>
        /// Сохранение компонент с указаеним продуктов в файл-Excel
        /// </summary>
        /// <param name="model"></param>
        public void SaveProductComponentToExcelFile(ReportBindingModel model)
        {
            _saveToExcel.CreateReport(new ExcelInfo
            {
                FileName = model.FileName,
                Title = "Список компонент",
                ProductComponents = GetProductComponent()
            });
        }

        /// <summary>
```

```
        /// Сохранение заказов в файл-Pdf
        /// </summary>
        /// <param name="model"></param>
        public void SaveOrdersToPdfFile(ReportBindingModel model)
        {
            _saveToPdf.CreateDoc(new PdfInfo
            {
                FileName = model.FileName,
                Title = "Список заказов",
                DateFrom = model.DateFrom!.Value,
                DateTo = model.DateTo!.Value,
                Orders = GetOrders(model)
            });
        }
    }
}
```

Листинг 4.24 – Класс ReportLogic

Переходим к проекту **AbstractShopView**. На основной форме делаем пункт меню «Отчеты» и в нем 3 подпункта: «Список компонентов», «Компоненты по изделиям» и «Список заказов» (рисунок 4.4).

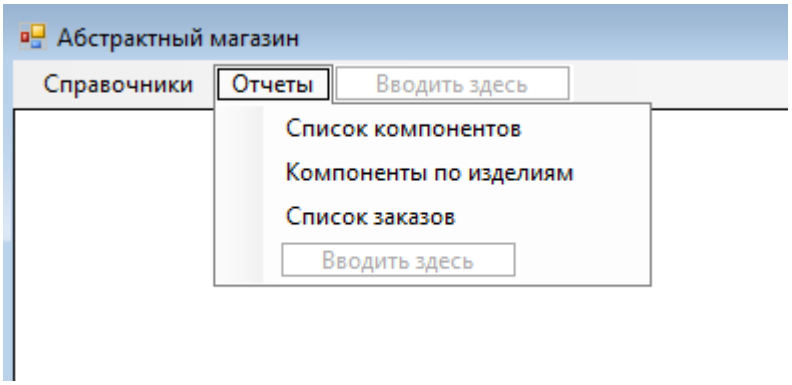


Рисунок 4.4 – Новые пункты меню на форме

Для первого пункта отдельная форма не требуется, так что весь код будет располагаться в FormMain. Для второго отчета создадим форму и поместим туда DataGridView. Колонки зададим руками. Добавим 3 колонки: Компонент, Изделие и Количество (рисунок 4.5).

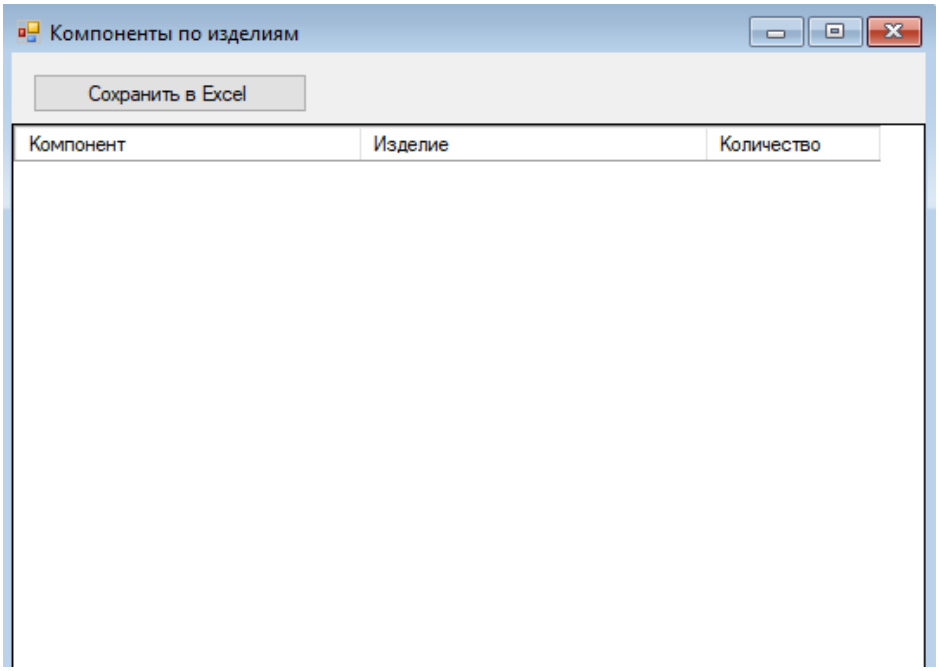


Рисунок 4.5 – Добавление колонок

При загрузке формы, будем подгружать данные по компонентам. Если получили данные, очищаем строки (хотя они и так будут пустыми) и добавляем новые. При этом в каждой строке будут заполняться разные ячейки. Добавим сюда кнопку для сохранения в Excel (листинг 4.25).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using Microsoft.Extensions.Logging;

namespace AbstractShopView
{
    public partial class FormReportProductComponents : Form
    {
        private readonly ILogger _logger;

        private readonly IReportLogic _logic;

        public FormReportProductComponents(ILogger<FormReportProductComponents>
logger, IReportLogic logic)
        {
            InitializeComponent();
            _logger = logger;
            _logic = logic;
        }

        private void FormReportProductComponents_Load(object sender, EventArgs e)
        {
            try
            {
                var dict = _logic.GetProductComponent();
                if (dict != null)
                {
                    dataGridView.Rows.Clear();
                    foreach (var elem in dict)
                    {
```

```
                        dataGridView.Rows.Add(new object[] { elem.ComponentName,
"", "" });
                        foreach (var listElem in elem.Products)
                        {
                            dataGridView.Rows.Add(new object[] { "",
listElem.Item1, listElem.Item2 });
                        }
                        dataGridView.Rows.Add(new object[] { "Итого", "",
elem.TotalCount });
                        dataGridView.Rows.Add(Array.Empty<object>());
                    }
                }
                _logger.LogInformation("Загрузка списка изделий по компонентам");
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка загрузки списка изделий по
компонентам");
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void ButtonSaveToExcel_Click(object sender, EventArgs e)
        {
                    using var dialog = new SaveFileDialog { Filter = "xlsx|*.xlsx"
};
                    if (dialog.ShowDialog() == DialogResult.OK)
                    {
                           try
                           {
                                  _logic.SaveProductComponentToExcelFile(new
ReportBindingModel
                                  {
                                         FileName = dialog.FileName
                                  });
                    _logger.LogInformation("Сохранение списка изделий по
компонентам");
                    MessageBox.Show("Выполнено", "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                           }
                           catch (Exception ex)
                {
                    _logger.LogError(ex, "Ошибка сохранения списка изделий по
компонентам");
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                           }
                    }
             }
    }
}
```

Листинг 4.25 – Логика формы FormReportProductComponents

И самое сложное – вывод отчета через шаблон отчетов. Для этого нам может понадобиться сделать ряд манипуляций. Нам нужны будут 2 вещи: файл-отчет и элемент для формы для вывода этого файла (ReportViewer, подключить библиотеку ReportViewerCore.WinForms). Файл создается через добавление нового элемента (рисунок 4.6).

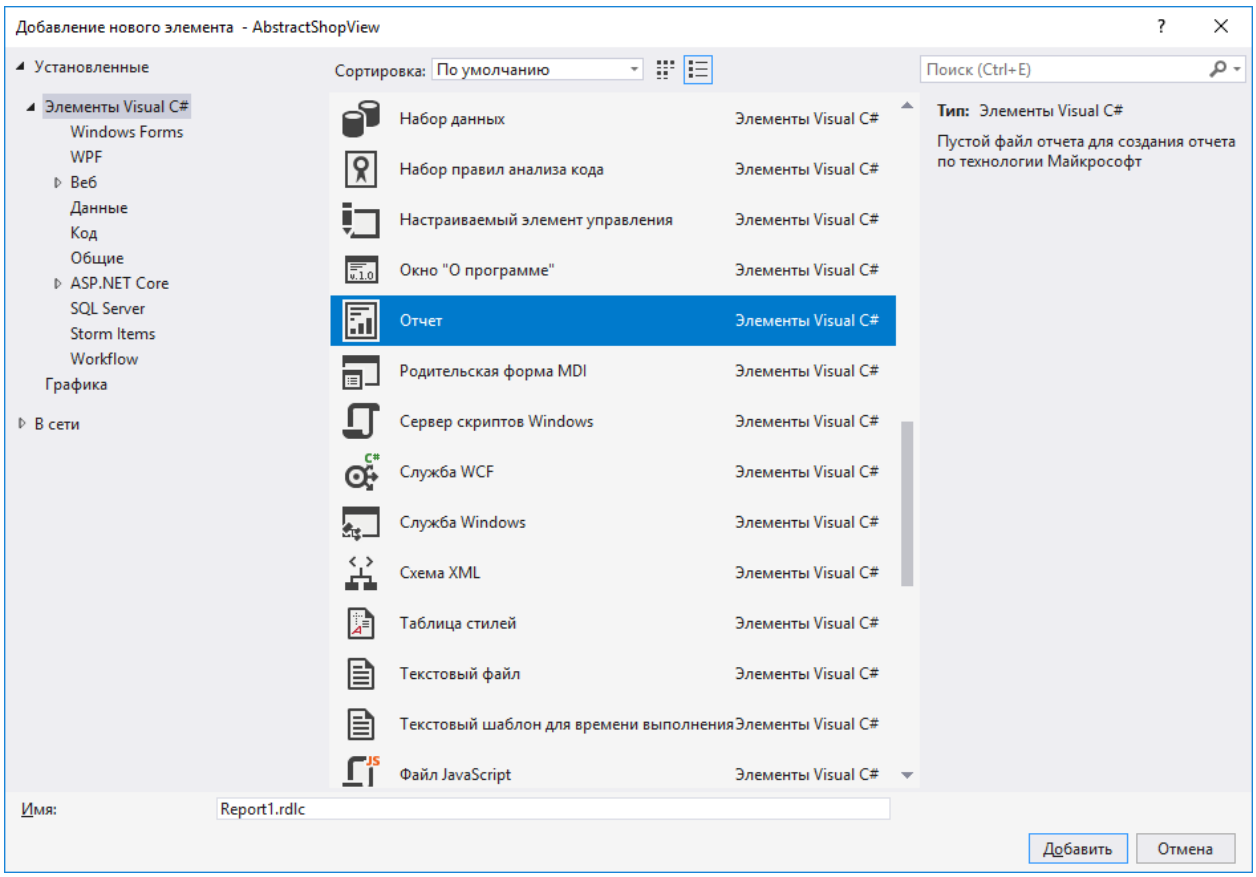


Рисунок 4.6 – Добавление нового элемента – отчета

Если у вас нет такого элемента, то нужно сделать следующие шаги:

1. Через установку программ найти Visual Studio и установить в ней Microsoft SQL Server Data Tools.
2. Скачать и установить 2 компонента для вашей версии Visual Studio: Microsoft.DataTools.ReportingServices.vsix и Microsoft.RdlcDesigner.vsix.

Создадим отчет (файл с расширением *.rdlc). Через панель элементов для отчета перемещаем на форму отчета текстовое поле (это будет заголовок). Далее в панели «Данные отчета» в «Параметры» добавляем новый параметр для вывода дат периода (рисунок 4.7).

И добавляем в «Наборы данных» новый набор. В качестве типа источника выбираем «Объект», так как данные будем получать не напрямую из БД, а через сервис (рисунок 4.8).

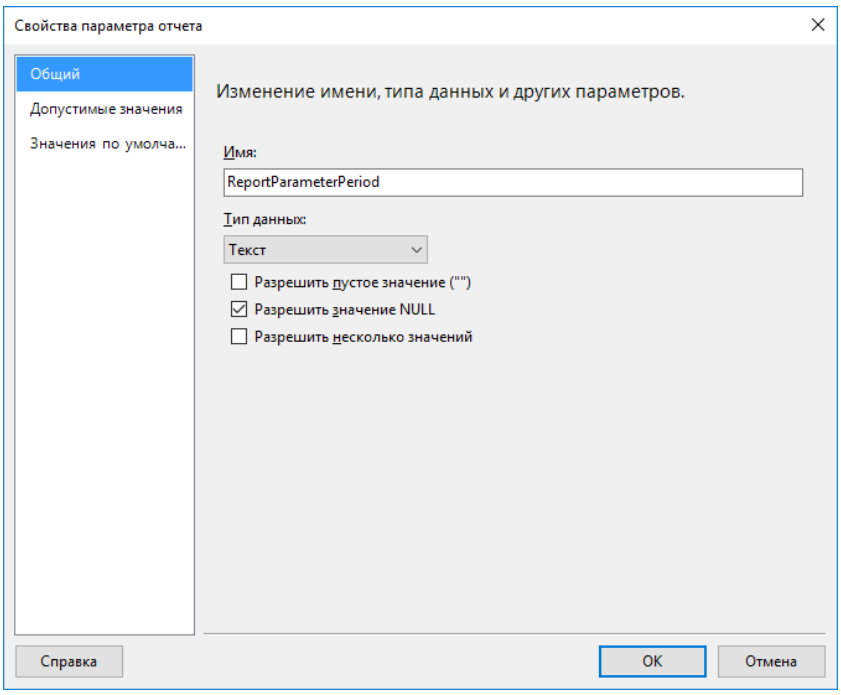


Рисунок 4.7 – Добавление параметра в отчет

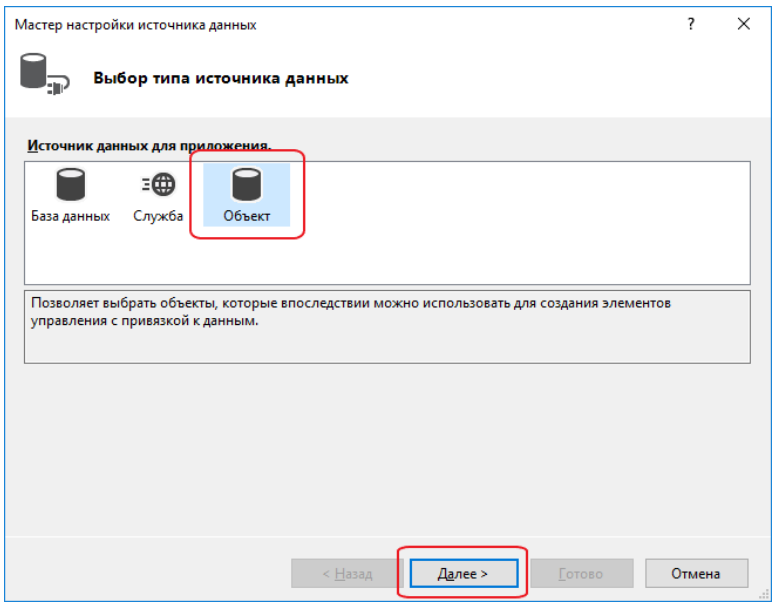


Рисунок 4.8 – Выбор типа источника данных

В открывшемся окне находим класс ViewModel сервиса, который хранит в себе описание полей, возвращаемых по запросу получения списка заказов. В результате создается набор данных. Остается только дать ему имя (рисунок 4.9).

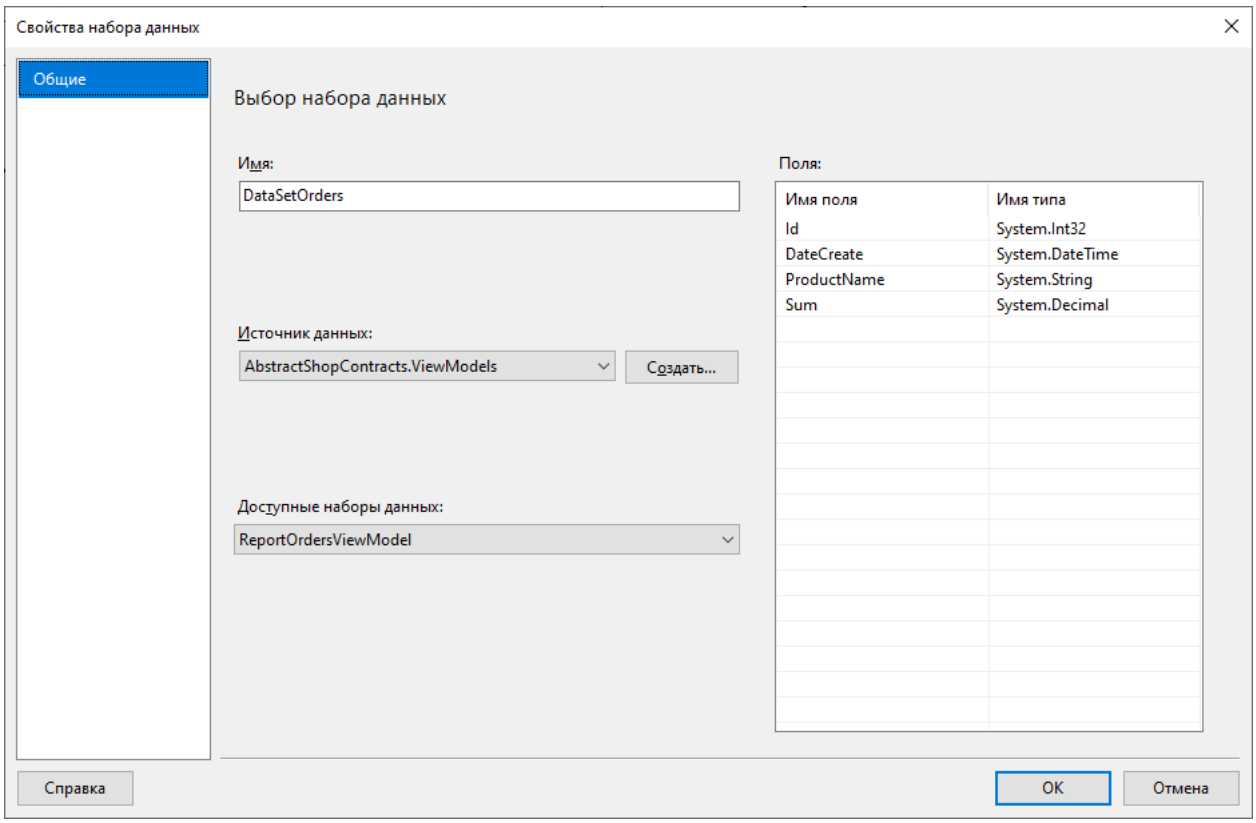


Рисунок 4.9 – Создание набора данных

С панели элементов переносим на отчет Таблицу и указываем ей, что данные в нее будут вставляться из набора DataSetOrders. Теперь в таблице настраиваем шапку и указываем где-какие данные будут располагаться. После таблицы будем выводить «Итого». Текст выводим через обычное текстовое поле, а значение берем из набора данных по полю «Sum». По желанию, меняем шрифт и прочее для текста (рисунок 4.10).

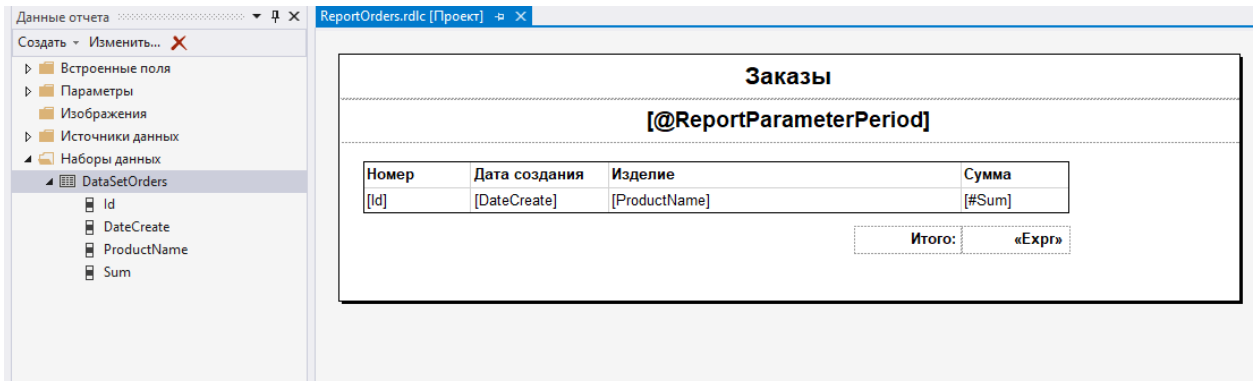


Рисунок 4.10 – Макет отчета

Создадим форму. Сверху расположим панель с двумя dataPicker и парой кнопок (вывести отчет и сохранить в pdf). Остальное пространство займет reportViewer (будем добавлять через код) (рисунок 4.11).

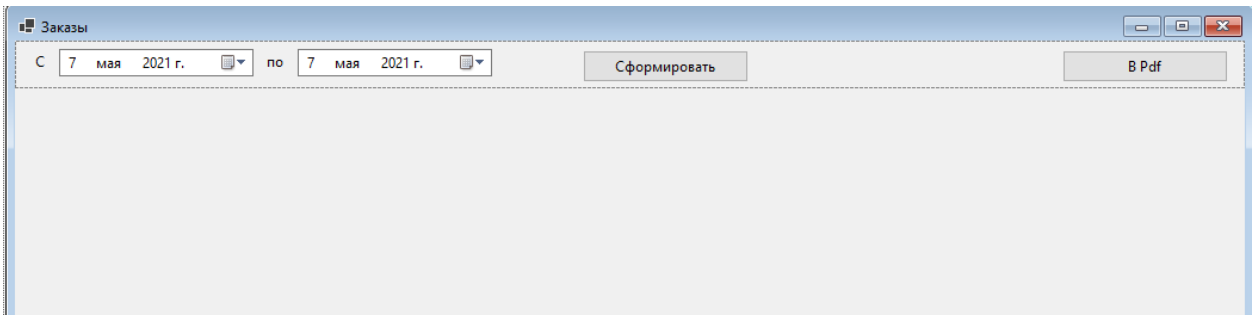


Рисунок 4.11 – Форма отчета по заказам с выбором отчета для reportViewer

Теперь в форме добавим обработку нажатия кнопок и логику инициализации для объекта от ReportViewer. Инициализацию сделаем в конструкторе формы (расположение на форме и укажем какой отчет использовать). В обработке кнопки «Сформировать» зададим проверку, чтобы дата начала была меньше даты окончания. Для отчета заполним параметр ReportParameterPeriod и добавим его в отчет. Получим список из сервиса и также передадим его в набор DataSetOrders. Для сохранения в Pdf также будет проверка по датам, получение имени файла и вызов команды сервиса (листинг 4.26).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using Microsoft.Extensions.Logging;
using Microsoft.Reporting.WinForms;

namespace AbstractShopView
{
    public partial class FormReportOrders : Form
    {
        private readonly ReportViewer reportViewer;

        private readonly ILogger _logger;

        private readonly IReportLogic _logic;

        public FormReportOrders(ILogger<FormReportOrders> logger, IReportLogic
logic)
        {
            InitializeComponent();
            _logger = logger;
            _logic = logic;
                    reportViewer = new ReportViewer
                    {
                            Dock = DockStyle.Fill
                    };
            reportViewer.LocalReport.LoadReportDefinition(new
FileStream("ReportOrders.rdlc", FileMode.Open));
            Controls.Clear();
            Controls.Add(reportViewer);
            Controls.Add(panel);
        }

        private void ButtonMake_Click(object sender, EventArgs e)
```

```
        {
            if (dateTimePickerFrom.Value.Date >= dateTimePickerTo.Value.Date)
            {
                MessageBox.Show("Дата начала должна быть меньше даты окончания",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            try
            {
                var dataSource = _logic.GetOrders(new ReportBindingModel
                {
                    DateFrom = dateTimePickerFrom.Value,
                    DateTo = dateTimePickerTo.Value
                });
                var source = new ReportDataSource("DataSetOrders", dataSource);
                reportViewer.LocalReport.DataSources.Clear();
                reportViewer.LocalReport.DataSources.Add(source);
                var parameters = new[] {  new
ReportParameter("ReportParameterPeriod",
                                          $"с
{dateTimePickerFrom.Value.ToShortDateString()} по
{dateTimePickerTo.Value.ToShortDateString()}") };
                reportViewer.LocalReport.SetParameters(parameters);

                reportViewer.RefreshReport();
                _logger.LogInformation("Загрузка списка заказов на период {From}-
{To}", dateTimePickerFrom.Value.ToShortDateString(),
dateTimePickerTo.Value.ToShortDateString());
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка загрузки списка заказов на период");
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void ButtonToPdf_Click(object sender, EventArgs e)
        {
            if (dateTimePickerFrom.Value.Date >= dateTimePickerTo.Value.Date)
            {
                MessageBox.Show("Дата начала должна быть меньше даты окончания",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
                using var dialog = new SaveFileDialog { Filter = "pdf|*.pdf"
};
                if (dialog.ShowDialog() == DialogResult.OK)
                {
                      try
                      {
                            _logic.SaveOrdersToPdfFile(new ReportBindingModel
                            {
                                  FileName = dialog.FileName,
                                  DateFrom = dateTimePickerFrom.Value,
                                  DateTo = dateTimePickerTo.Value
                            });
                     _logger.LogInformation("Сохранение списка заказов на период
{From}-{To}", dateTimePickerFrom.Value.ToShortDateString(),
dateTimePickerTo.Value.ToShortDateString());
                      MessageBox.Show("Выполнено", "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                      }
                      catch (Exception ex)
                {
```

```
                    _logger.LogError(ex, "Ошибка сохранения списка заказов на
период");
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                        }
                    }
                }
    }
}
```

Листинг 4.26 – Логика формы FormReportOrders

В логике главной формы прописываем вызовы форм для отчетов (листинг 4.27).

```
        private void ComponentsToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            using var dialog = new SaveFileDialog { Filter = "docx|*.docx" };
            if (dialog.ShowDialog() == DialogResult.OK)
            {
                _reportLogic.SaveComponentsToWordFile(new ReportBindingModel {
FileName = dialog.FileName });
                MessageBox.Show("Выполнено", "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
        }

        private void ComponentProductsToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            var service =
Program.ServiceProvider?.GetService(typeof(FormReportProductComponents));
            if (service is FormReportProductComponents form)
            {
                form.ShowDialog();
            }
        }

        private void OrdersToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var service =
Program.ServiceProvider?.GetService(typeof(FormReportOrders));
            if (service is FormReportOrders form)
            {
                form.ShowDialog();
            }
        }
```

Листинг 4.27 – Фрагмент логики формы FormMain с новыми методами

Контрольный пример

Результат вызова пункта «Список компонентов». Результат работы метода представлен на рисунке 4.12.

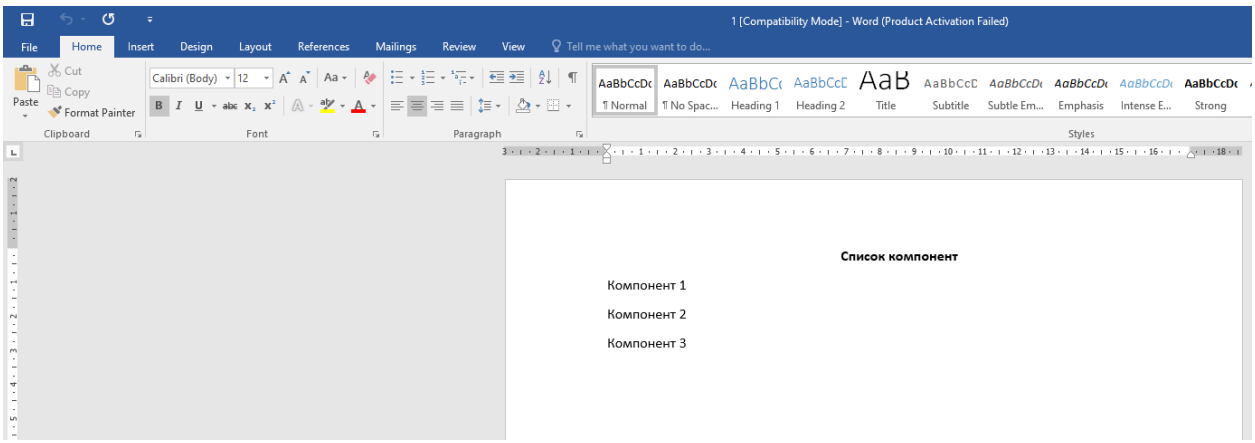
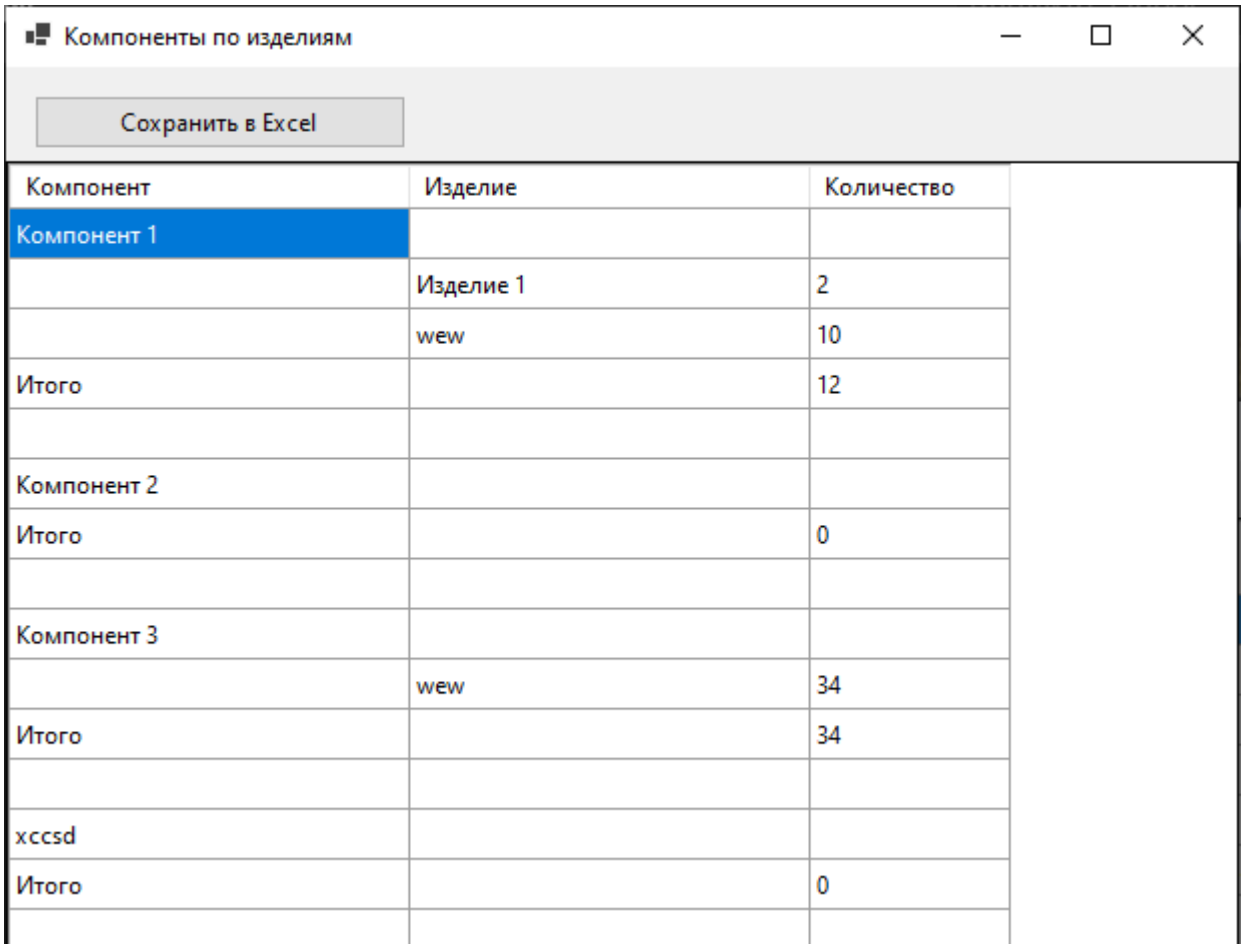


Рисунок 4.12 – Отчет по перечню компонент

Форма отчетов по компонентам в разрезе изделий представлена на рисунке 4.13.



Компонент	Изделие	Количество
Компонент 1		
	Изделие 1	2
	wew	10
Итого		12
Компонент 2		
Итого		0
Компонент 3		
	wew	34
Итого		34
xccsd		
Итого		0

Рисунок 4.13 – Форма отчета по компонентам

Результат сохранения в Excel представлен на рисунке 4.14.

	A	B	C	D
1	Список компонент			
2	Компонент 1			
3		Изделие 1	2	
4			2	
5	Компонент 2			
6			0	
7	Компонент 3			
8			0	
9				
10				
11				

Рисунок 4.14 – Отчет по складам

Форма отчетов по заказам за период представлена на рисунке 4.15.

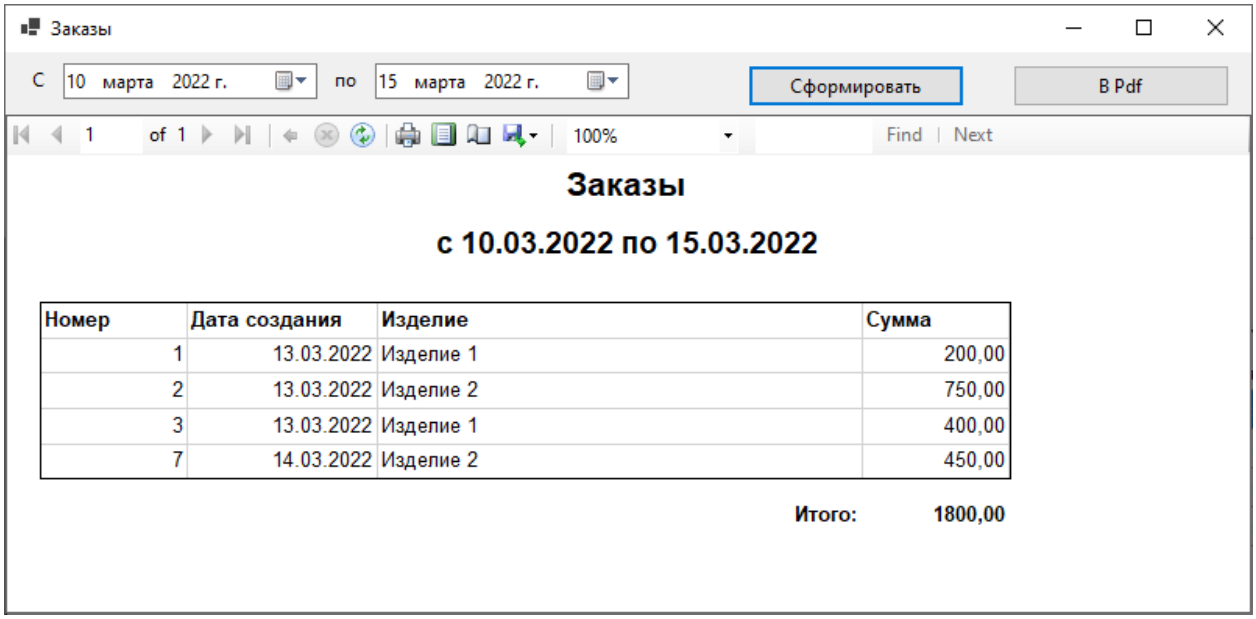


Рисунок 4.15 – Форма отчета по заказам за период

Результат сохранения в Pdf представлен на рисунке 4.16.

Список заказов

с 10.03.2022 по 15.03.2022

Номер	Дата заказа	Изделие	Сумма
1,00	13.03.2022	Изделие 1	200,00
2,00	13.03.2022	Изделие 2	750,00
3,00	13.03.2022	Изделие 1	400,00
7,00	14.03.2022	Изделие 2	450,00

Итого: 1800,00

Рисунок 4.16 – Отчет по заказам за период

Требования

1. Название проектов должны ОТЛИЧАТЬСЯ от названия проектов, приведенных в примере и должны соответствовать логике вашего задания по варианту.
2. Название форм, классов, свойств классов должно соответствовать логике вашего задания по варианту.
3. НЕ ИСПОЛЬЗОВАТЬ в названии класса, связанного с изделием слово «Product» (во вариантах в скобках указано название класса для изделия)!!!
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Для всех 3 реализаций логики хранения данных по сущности «Заказ» дополнить логику получения списка фильтрованных заказов, чтобы можно было получать заказы за определенный период.
6. В doc-файл выводить **список изделий** с выделением названия жирным, а цену обычным шрифтом.

7. В excel-файл (и на форму через dataGridView) выводить **изделия** с указанием, какие **компоненты в них используются** и подсчетом общего количества используемых компонент в изделии.
8. В pdf-файл (и в отчет) добавить колонку «Статус заказа» и выводить там значение статуса заказа.

Порядок сдачи базовой части

1. Создать текстовый документ (Word) с 2-3 изделиями.
2. Открыть форму по изделиям с компонентами (должно быть по 2-3 компонента у изделий).
3. Создать табличный документ (Excel) на основе выводимых на форме данных.
4. Открыть форму по заказам, изменить даты, вывести список заказов (не менее 3 заказов).
5. Создать pdf-документ на основе выводимых на форме данных.
6. Показать 3 созданных документа (можно показывать каждый документ сразу после этапов 1, 3 и 4).
7. Ответить на вопрос преподавателя.

Контрольные вопросы к базовой части

1. Как записываются данные в doc-файл (показать в коде)?
2. Как формируется таблица в excel-файле (показать в коде)?
3. Какие манипуляции потребовалось сделать для добавления колонки «Статус заказа» в отчет?

Усложненная лабораторная (необязательно)

Для усложненного варианта требуется:

1. Выводить в doc-файл список магазинов в табличном виде (3 колонки: название, адрес и дата открытия).

2. Выводить в excel-файл и на форму через dataGridView информацию по загруженности магазинов с указанием какие изделия и в каком количестве имеются в магазинах и общее количество изделий в магазине.
3. Выводить pdf-файл и на форму через reportViewer информацию о заказах (за весь период), объединенных по датам (3 колонки: дата, кол-во заказов на эту дату, сумма по заказам).

Порядок сдачи усложненной части

1. Создать текстовый документ (Word) с 2-3 магазинами.
2. Открыть форму по магазинам с изделиями (должно быть по 2-3 изделия в каждом магазине).
3. Создать табличный документ (Excel) на основе выводимых на форме данных.
4. Открыть форму по заказам, вывести список заказов (по несколько заказов на каждый день и на 2-3 разные даты).
5. Создать pdf-документ на основе выводимых на форме данных.
6. Показать 3 созданных документа (можно показывать каждый документ сразу после этапов 1, 3 и 4).
7. Ответить на вопрос преподавателя.

Контрольные вопросы к усложненной части

1. Как формируется doc-файл (показать в коде)?
2. Как формируется excel-файл (показать в коде)?
3. Как формируется pdf-файл (показать в коде)?

Варианты

1. Кондитерская. В качестве компонентов выступают различные виды шоколада и наполнители, типа орехов, изюма и т.п. Изделие – кондитерское изделие (pastry).

2. Автомастерская. В качестве компонентов выступают различные масла, смазки и т.п. Изделия – ремонт автомобиля (repair).
3. Моторный завод. В качестве компонентов выступают различные детали для производства двигателей. Изделия – двигатели (engine).
4. Суши-бар. В качестве компонентов выступают различные продукты для суши (рыба, водоросли, соусы). Изделия – суши (sushi).
5. Продажа компьютеров. В качестве компонентов выступают различные части для компьютеров (планки памяти, жесткие диски и т.п.). Изделия – компьютеры (computer).
6. Сборка мебели. В качестве компонентов выступают различные заготовки (ножки, спинки и т.п.). Изделия – мебель (furniture).
7. Рыбный завод. В качестве компонентов выступают различные виды рыб + дополнения к ним, типа соусов и т.п. Изделия – консервы (canned).
8. Установка ПО. В качестве компонентов выступают различное ПО. Изделия – пакеты установки, например, пакет установки офисных приложений, пакет разработчика и т.п. (package).
9. Ремонтные работы в помещении. В качестве компонентов выступают различные расходные материалы (клей, обои, краска, плитка, цемент и т.п.). Изделия – ремонтные работы в различных помещениях (repair).
10. Кузнечная мастерская. В качестве компонентов выступают различные болванки (заготовки), из которых изготавливаются подковы, кочерги и т.п. Изделия – кузнечные изделия (manufacture).
11. Пиццерия. В качестве компонентов выступают различные ингредиенты для пицц (тесто, соусы, паста и т.д.). Изделия – пиццы (pizza).
12. Завод ЖБИ. В качестве компонентов выступают различные виды бетона и металлоконструкций. Изделия – железобетонные изделия (reinforced).

13. Закусочная. В качестве компонентов выступают различные продукты для закусок (колбаса, сыр, хлеб и т.п.). Изделия – различные закуски (snack).
14. Пошив платьев. В качестве компонентов выступают различные ткани, нитки и т.п. Изделия – платья (dress).
15. Типография. В качестве компонентов выступают различные типы бумаг, тонер или чернила и т.п. Изделия – печатная продукция (листовки, брошюры, книги) (printed).
16. Автомобильный завод. В качестве компонентов выступают различные части для сборки автомобилей (кузов, двигатель, стекла и т.п.). Изделия – автомобили (car).
17. Юридическая фирма. В качестве компонентов выступают различные бланки для документов. Изделия – пакеты документов, например, для страховки или завещания (document).
18. Туристическая фирма. В качестве компонентов выступают различные условия поездки (отель проживания, туры в рамках поездок). Изделия – туристические путевки (travel).
19. Цветочная лавка. В качестве компонентов выступают различные цветы и украшения к ним. Изделия – цветочные композиции (flower).
20. Ювелирная лавка. В качестве компонентов выступают различные драгоценные камни и металлы. Изделия – драгоценности (jewel).
21. Авиастроительный завод. В качестве компонентов выступают различные части для сборки самолета (двигатели, крылья, фюзеляж и т.п.). Изделия – самолеты (plane).
22. Магазин подарков. В качестве компонентов выступают различные упаковочные материалы, ленты и подарки. Изделия – подарочные наборы (gift).
23. Система безопасности. В качестве компонентов выступают различные камеры, датчики и т.п. Изделия – базовые комплектации охраны, продвинутые, для предприятий, для частных и т.п. (secure).

24. Заказы еды. В качестве компонентов выступают различные блюда. Изделия – это наборы блюд (типа обеденный набор, или утренний набор, или набор для пикника) (dish).
25. Ремонт сантехники. В качестве компонентов выступают различные трубы, прокладки, смесители т.п. Изделия – замены смесителей, труб и т.п. (work).
26. Лавка с мороженым. В качестве компонентов выступают различные виды мороженного и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icecream).
27. Судостроительный завод. В качестве компонентов выступают различные части для сборки судов (корпуса, двигатели и т.п.). Изделия – суда (ship).
28. Столярная мастерская. В качестве компонентов выступают различные деревянные заготовки. Изделия – деревянные игрушки, утварь и т.п. (wood).
29. Бар. В качестве компонентов выступают различные ингредиенты для коктейлей. Изделия – коктейли (cocktail).
30. Швейная фабрика. В качестве компонентов выступают различные заготовки для штор, покрывал и т.п. (textile).